Analýza dat pomocí vestavěných nástrojů Pythonu

Petr Pošík

Katedra kybernetiky, FEL ČVUT v Praze

OI, B4B33RPH - Řešení problémů a hry, 2022, 2018, 2016

Credits

Tato přednáška a text jsou silně inspirovány přednáškou

David Beazley: **Builtin Superheroes!**, PyData Chicago, 2016 (recording, screencast)

Proč toto téma a proč právě teď?

- V minulých přednáškách jsme vám ukazovali
  - jak programovat v Pythonu a
  - jaké soft-skills byste jako programátoři měli pěstovat a rozvíjet (psát kód čistě, testovat ho, upravovat ho, aby byl čistší, srozumitelnější, efektivnější).
- V následujících motivačních přednáškách vám kolegové ukážou, čeho lze dosáhnout se znalostmi, kterými vás vybaví další studium.
- Tato přednáška vlastně patří do obou skupin:
  - Stále budeme ponořeni v Pythonu, ale
  - ukážeme si jeho použití na příkladu průzkumové analýzy reálných dat.
- Datové struktury a rysy Pythonu, které si ukážeme, vám navíc mohou pomoct při řešení závěrečné semestrální úlohy Spam filtr:
  - Uvidíte, jak snadno počítat četnost výskytů nějakého prvku v nějaké kolekci prvků.
  - Uvidíte, jak pracovat s textovými řetězci.

Otevřená data

> Otevřená data jsou informace a čísla bezplatně a volně dostupná na internetu ve strukturované a strojově čitelné podobě a jsou zpřístupněna způsobem, který jejich využití neklade zbytečné technické či jiné překážky.

Z http://data.gov.cz:

- Nejzajímavější by byla data ze státní správy:
  - Ministerstva
  - ČOI, Statistický úřad
  - Města a obce
  - ...
- V ČR bohužel stále v plenkách, mnoho zveřejněných dat už předzpracováno a sumarizováno, což do značné míry znemožňuje vlastní analýzu.

City of Chicago data portal

City of Chicago data portal: https://data.cityofchicago.org/

- Data prakticky o všem:
  - Rozpočet, platy úředníků
  - Kriminalita
  - Opravy děr na silnicích
  - ...

Náš projekt

Data: Food Inspections https://data.cityofchicago.org/Health-Human-Services/Food-Inspections/4ijn-s7e5

- Soubor o velikosti ~270 MB s ~250 tis. řádky.
- Využijeme jen vestavěné nástroje Pythonu.

(Specializované nástroje, např. `pandas`, by tuto analýzu umožnily provést ještě snazším způsobem; my ale chceme demonstrovat sílu vestavěných nástrojů.)

Pracovní adresář

In [1]:
```
import os
os.getcwd()
```

Out[1]:
```
'C:\\P\\0Teaching\\rph\\repos\\rph-lectures\\data-analysis'
```

Import dat

Pomocí modulu `csv` Pythonu načteme data jako **seznam slovníků**.

- Možná nejméně efektivní repezentace dat.
- Ale velmi snadno se s ní pracuje.

In [2]:
```
import csv
with open('Food_Inspections.csv', 'r', encoding='utf-8') as f:
    all = list(csv.DictReader(f))
len(all)
```

Out[2]:
```
245390
```

Načetlo se přes 245 tisíc záznamů. A takto vypadá jeden z nich:

In [3]:
```
all[-4]
```

Out[3]:
```
{'Inspection ID': '2565459',
 'DBA Name': 'EZI MART',
 'AKA Name': 'EZI MART',
 'License #': '2867531',
 'Facility Type': 'Grocery Store',
 'Risk': 'Risk 3 (Low)',
 'Address': '4804 N CENTRAL AVE',
 'City': 'CHICAGO',
 'State': 'IL',
 'Zip': '60630',
 'Inspection Date': '10/18/2022',
 'Inspection Type': 'License',
 'Results': 'Fail',
 'Violations': '5. PROCEDURES FOR RESPONDING TO VOMITING AND DIARRHEAL EVENTS -
Comments: FOUND NO VOMIT AND DIARRHEAL CLEANUP PROCEDURE WITH SPILL KIT. PRIORIT
Y VIOLATION.NO CITATION ISSUED.7-38-005.MUST PROVIDE. | 51. PLUMBING INSTALLED;
PROPER BACKFLOW DEVICES - Comments: FOUND NO BACKFLOW PREVENTION DEVICES INSTALL
ED AT MOP SINK FAUCET.MUST PROVIDE WHERE IT CAN BE SERVICED /LOCATED. | 53. TOIL
ET FACILITIES: PROPERLY CONSTRUCTED, SUPPLIED, & CLEANED - Comments: FOUND NO SE
LF CLOSING DOOR AT RESTROOM. MUST PROVIDE. | 54. GARBAGE & REFUSE PROPERLY DISPO
SED; FACILITIES MAINTAINED - Comments: FOUND NO OUTSIDE DUMPSTER.PRIORITY VIOLAT
ION. NO CITATION ISSUED.7-38-020B.MUST PROVIDE. | 56. ADEQUATE VENTILATION & LIG
HTING; DESIGNATED AREAS USED - Comments: FOUND NO EXHAUST VENT INSIDE RESTROOM .
MUST PROVIDE.',
 'Latitude': '41.967942032062496',
 'Longitude': '-87.76762783599236',
 'Location': '(41.967942032062496, -87.76762783599236)'}
```

Tento typ dat Chicago publikuje o každé inspekci. Najdete tam název podniku, adresu, typ podniku, datum inspekce, výsledky inspekce, porušení předpisů, atd.

Může být užitečné si data seřadit, např. podle ID inspekcí. Pozor! Pokud chcete řadit podle čísel (nikoli abecedně podle řetězců), je třeba jako klíč pro řazení vytvořit z ID číslo!

In [4]:
```
all.sort(key=lambda x: int(x["Inspection ID"]))
```

Nyní první záznam o kontrole měl být poměrně starý, zatímco poslední záznam jen pár dní starý:

In [5]:
```
all[0]["Inspection Date"], all[-1]["Inspection Date"]
```

Out[5]:
```
('01/05/2010', '11/22/2022')
```

Výsledky inspekcí

Zkusme se do dat trochu zavrtat. Může nás zajímat, jaké jsou všechny možné výsledky inspekcí. Použijme tzv. set comprehension. Následující kód projde všechny záznamy o inspekcích a posbírá jejich výsledky do množiny. Využijeme přitom vlastnosti, že množina neuchovává duplicitní hodnoty.

In [6]:
```
{insp['Results'] for insp in all}
```

Out[6]:
```
{'Business Not Located',
 'Fail',
 'No Entry',
 'Not Ready',
 'Out of Business',
 'Pass',
 'Pass w/ Conditions'}
```

Výsledkem inspekce je tedy nejspíš jedna z několika předdefinovaných hodnot.

Inspekce s negativním výsledkem

Zkusíme dále analyzovat ty inspekce, které skončily s negativním výsledkem `Fail`. Profiltrujeme data pomocí list comprehension, tj. vytvoříme seznam obsahující jen inspekce s výsledkem `Fail`.

In [7]:
```
fail = [insp for insp in all if insp['Results'] == 'Fail']
len(fail)
```

Out[7]:
```
47591
```

In [8]:
```
len(fail) / len(all)
```

Out[8]:
```
0.19394025836423653
```

Cca 48 tis. inspekcí z cca 245 tis. bylo negativních, tj. téměř 20 %.

Nejhorší podnik v Chicagu, kde se najíst

Když už jsme se vydali tímto směrem, jaký je v Chicagu nejhorší podnik, kde se najíst?

Spočtěme, kolikrát každý podnik při inspekci selhal. Podnik identifikujeme podle pole "DBA Name" (DBA - Doing Business As). Využijeme `Counter`, který inicializujeme hodnotami generátorového výrazu. A podívejme se na prvních 5 položek:

In [9]:
```
from collections import Counter
worst = Counter(insp['DBA Name'] for insp in fail)
worst.most_common(5)
```

Out[9]:
```
[('SUBWAY', 433),
 ('DUNKIN DONUTS', 273),
 ("MCDONALD'S", 125),
 ('7-ELEVEN', 74),
 ('MCDONALDS', 62)]
```

Abychom byli féroví, důvodem, proč tyto podniky mají nejvíc negativních inspekcí je patrně to, že také mají v Chicagu nejvíce provozoven... Později se k tomu vrátíme.

Další věc, která je zajímavá, je **jak se vlastně píše McDonalds?** :-)

Zde narážíme na velmi běžný problém při zpracování dat. Nejdříve bychom měli data nějak vyčistit. Pokud se na sloupeček DBA Name podíváte podrobněji, zjistíte, že některé hodnoty používají malá písmena, některá velká, některá obsahují apostrofy, jiná ne ...

Potřebovali bychom na hodnoty aplikovat transformaci, která např. odstraní z názvů apostrofy a mezery a převede je na velká písmena:

In [10]:
```
"Mc Donald's".replace("'", "").replace(" ","").upper()
```

Out[10]:
```
'MCDONALDS'
```

Dává dobrý smysl si z této sekvence úprav udělat funkci, kterou budeme aplikovat na název podniku před dalším zpracováním.

In [11]:
```
def normalize(s):
    return s.replace("'", "").replace(" ","").upper()
```

Abychom tuto tranformaci mohli aplikovat na každý záznam, použijeme relativně novou funkci Pythonu 3.5, která se může zdát neintuitivní a exotická. Berte to jako příklad, co lze dělat pomocí vestavěných nástrojů.

Následující řádek obsahuje list comprehension. Na každý prvek ze seznamu `fail` aplikujeme transformaci. Transformace je uvedena ve složených závorkách. Vezme slovník `insp` tak, jak je, ale pak jeho pole `'DBA Name'` přepíše novou hodnotou odvozenou od té původní. Právě část `**insp, key: value` je nová v Pythonu 3.5: vezme slovník, sloučí ho s novým párem `key: value` a vrátí nový slovník.

In [12]:
```
fail = [ {**insp, 'DBA Name': normalize(insp['DBA Name'])} for insp in fail ]
```

A nyní můžeme zopakovat naší analýzu, kde je nejhorší místo k jídlu v Chicagu:

In [13]:
```
worst = Counter(insp['DBA Name'] for insp in fail)
worst.most_common(5)
```

Out[13]:
```
[('SUBWAY', 464),
 ('DUNKINDONUTS', 298),
 ('MCDONALDS', 282),
 ('DUNKINDONUTS/BASKINROBBINS', 108),
 ('7-ELEVEN', 103)]
```

Toto je jen jeden krok čištění dat, další nedostatky v datech asi stále existují, ale to nyní opravovat nebudeme.

Normalizovaný počet selhání

Jak jsme si všimli, výše uvedené podniky mají nevyšší počet případů, kdy neprošly inspekcí, patrně proto, že mají v Chicagu nejvíc provozoven. Spíše než absolutní počet selhání nás zajímá poměr mezi počtem selhání a celkovým počtem kontrol, které byly v podnicích provedeny.

Pro jistotu nejprve normalizujeme název podniku jak v seznamu `fail`, tak v seznamu `all`:

In [14]:
```
fail = [ {**insp, 'DBA Name': normalize(insp['DBA Name'])} for insp in fail ]  #
all = [ {**insp, 'DBA Name': normalize(insp['DBA Name'])} for insp in all ]
```

Nyní můžeme spočítat jak počty neúspěšných inspekcí v jednotlivých podnicích, tak i celkový počet inspekcí v každém z nich:

In [15]:
```
fail_count = Counter(insp["DBA Name"] for insp in fail)
all_count = Counter(insp["DBA Name"] for insp in all)
```

Zajímají nás vlastně všechny podniky, jejichž názvy jsou uvedeny v counteru `all_count`:

In [16]:
```
names = {name for name in all_count}
```

Nyní stačí názvy podniků projít a pro každý název spočítat podíl neúspěšných inspekcí vůči všem inspekcím. Poměry si budeme ukládat opět do čítače, protože se nám bude následně hodit jeho metoda `most_common()`. Můžeme ho tentokrát zkusit vytvořit pomocí tzv. dictionary compehension:

In [17]:
```
fail_rate = Counter({name: fail_count[name]/all_count[name] for name in names})
fail_rate.most_common(5)
```

Out[17]:
```
[('WISHCO,INC.', 1.0),
 ('LATERMINALJESUSMINIMARKET', 1.0),
 ('MR.PORKIESBAKERY', 1.0),
 ('ALVAROJIMENEZ', 1.0),
 ('VERNSSWEETTREATS', 1.0)]
```

Hmm, to možná není úplně to, co jsme chtěli. Nejpravděpodobnější příčinou toho, že podnik selhal ve všech inspekcích, které v něm byly provedeny, je to, že v něm byla provedena jediná kontrola, ve které shodou okolností selhal. Možná je smysluplnější omezit se jen na ty podniky, v nichž bylo provedeno např. více než 5 inspekcí.

Stačí nám změnit vlastně jen množinu názvů podniků, pro které poměr počítáme, zbytek analýzy zůstane stejný:

In [18]:
```
names = {name for name, count in all_count.items() if count > 5}
fail_rate = Counter({name: fail_count[name]/all_count[name] for name in names})
fail_rate.most_common(5)
```

Out[18]:
```
[('NICKSFOODMART', 0.8571428571428571),
 ('WELLSFOODMARTINC.', 0.8333333333333334),
 ('ORLYS/JALAPENO', 0.8),
 ('UBITASICECREAM', 0.7777777777777778),
 ('MRTS', 0.7142857142857143)]
```

Nejhorší adresa, kde se v Chicagu najíst

Můžeme se také ptát: kde je nejhorší adresa, kde se můžeme v Chicagu najíst?

In [19]:
```
bad = Counter(insp['Address'] for insp in fail)
bad.most_common(5)
```

Out[19]:
```
[('11601 W TOUHY AVE', 372),
 ('2300 S THROOP ST', 128),
 ('500 W MADISON ST', 95),
 ('324 N LEAVITT ST', 91),
 ('5700 S CICERO AVE', 64)]
```

Výrazně největší počet negativních inspekcí byl zaznamenán na 11601 West Touhy Avenue. Ale možná, že toto vysoké číslo má původ v dávno minulém období a v současnosti je už vše v pořádku. Možná je to naopak. Možná je to stabilní trend. Pojďme to zjistit.

Nejprve, jak zjistíme rok, kdy byla inspekce provedena? Můžeme se např. podívat na poslední 4 znaky v `insp["Inspection Date"]` :

In [20]:
```
fail[0]["Inspection Date"][-4:]
```

Out[20]:
```
'2010'
```

Nyní můžeme zjistit počet negativních inspekcí podle jednotlivých let. Sestavíme slovník čítačů, kde klíčem ve slovníku bude rok inspekce. Využijeme k tomu speciální druh slovníku, `defaultdict` . Pokud v takovém slovníku nebude existovat hodnota pro nějaký klíč, slovník tuto hodnotu sám vytvoří a následně vrátí. My budeme chtít, aby vytvořil prázdný `Counter` , pokud ve slovníku nebude existovat položka pro nějaký rok.

In [21]:
```
from collections import defaultdict
by_year = defaultdict(Counter)
```

Zkusíme tyto čítače naplnit, tentokrát "ručně": nejprve zjistíme rok, kdy kontrola proběhla, a následně zvýšíme hodnotu čítače pro daný rok a danou adresu:

In [22]:
```
for insp in fail:
    year = insp['Inspection Date'][-4:]
    address = insp['Address']
    by_year[year][address] += 1
```

Když se nyní podíváme, jak si adresa "11601 W Touhy Ave" vedla v několika minulých letech, zjistíme, že velký počet selhání není náhoda:

In [23]:
```
by_year['2022'].most_common(5)
```

Out[23]:
```
[('7601 S CICERO AVE', 13),
 ('500 W MADISON ST', 10),
 ('11601 W TOUHY AVE', 10),
 ('2300 S THROOP ST', 9),
 ('2053 W DIVISION ST', 8)]
```

In [24]:
```
by_year['2021'].most_common(5)
```

Out[24]:
```
[('11601 W TOUHY AVE', 16),
 ('505 N STATE ST', 10),
 ('2300 S THROOP ST', 9),
 ('135 N KEDZIE AVE', 7),
 ('4367 N MILWAUKEE AVE', 6)]
```

In [25]:
```
by_year['2020'].most_common(5)
```

Out[25]:
```
[('11601 W TOUHY AVE', 7),
 ('108 N STATE ST', 6),
 ('2218-2222 N WESTERN AVE', 6),
 ('2300 S THROOP ST', 6),
 ('5631 S Kimbark (1300E)', 5)]
```

In [26]:
```
by_year['2019'].most_common(5)
```

Out[26]:
```
[('11601 W TOUHY AVE', 36),
 ('2300 S THROOP ST', 25),
 ('2002 S WENTWORTH AVE', 6),
 ('6560 W FULLERTON AVE', 5),
 ('100 W 87TH ST', 5)]
```

Zdá se, že počet negativních kontrol za 1 rok na adrese 11601 Touhy Avenue je větší než na jakékoli jiné adrese a že je to stabilní trend. Ale možná je to tím, že je na této adrese mnoho stravovacích podniků...

Co je na adrese 11601 W TOUHY AVE?

Když adresu zadáte do Google Maps, zjistíte, že patrně patří ...

... **O'Hare International Airport**!

Nejčastější nedostatky

Podívejme se na popis nedostatků ve výsledcích některé inspekce:

In [27]:
```
viol = fail[1]['Violations']
viol
```

Out[27]:
```
'33. FOOD AND NON-FOOD CONTACT EQUIPMENT UTENSILS CLEAN, FREE OF ABRASIVE DETERG
ENTS - Comments: ADDITIONAL CLEANING OF SIDES OF COOKING EQUIPMENT WITH GREASE A
ND FOOD DEBRIS ACCUMULATION. (STOVE, FRYER ETC) | 34. FLOORS: CONSTRUCTED PER CO
DE, CLEANED, GOOD REPAIR, COVING INSTALLED, DUST-LESS CLEANING METHODS USED - Co
mments: ADDITIONAL CLEANING OF FLOORS IN THE KITCHEN ALONG THE WALLS AND IN ALL
CORNERS ESPECIALLY UNDER AND AROUND REACH-IN COOLERS. | 38. VENTILATION: ROOMS A
ND EQUIPMENT VENTED AS REQUIRED: PLUMBING: INSTALLED AND MAINTAINED - Comments:
MUST INSTALL EXPOSED HANDSINK BEHIND FRONT SERVICE BAR.--- MUST REPAIR VENTILATI
ON IN EMPLOYEE WASHROOM.---  SODA DRAIN LINE DUMPING INTO 1 COMPARTMENT OF THE 3
COMP BAR SINK. INSTRUCTED TO RELOCATE PROPERLY INTO FLOOR DRAIN.'
```

Pole `'Violations'` má jistou vnitřní strukturu. Jednotlivé položky jsou odděleny svislítky (|). Každý nedostatek má číslo, (doufejme) standardizovaný název a mohou být připojeny komentáře.

Můžeme se pokusit extrahovat z těchto polí nějaké informace. Podívejme se nejprve, jak můžeme zpracovat jeden text z pole `'Violations'`. Nejprve text rozdělíme podle svislítek:

In [28]:
```
viol.split('|')
```

Out[28]:
```
['33. FOOD AND NON-FOOD CONTACT EQUIPMENT UTENSILS CLEAN, FREE OF ABRASIVE DETER
GENTS - Comments: ADDITIONAL CLEANING OF SIDES OF COOKING EQUIPMENT WITH GREASE
AND FOOD DEBRIS ACCUMULATION. (STOVE, FRYER ETC) ',
 ' 34. FLOORS: CONSTRUCTED PER CODE, CLEANED, GOOD REPAIR, COVING INSTALLED, DUS
T-LESS CLEANING METHODS USED - Comments: ADDITIONAL CLEANING OF FLOORS IN THE KI
TCHEN ALONG THE WALLS AND IN ALL CORNERS ESPECIALLY UNDER AND AROUND REACH-IN CO
OLERS. ',
 ' 38. VENTILATION: ROOMS AND EQUIPMENT VENTED AS REQUIRED: PLUMBING: INSTALLED
AND MAINTAINED - Comments: MUST INSTALL EXPOSED HANDSINK BEHIND FRONT SERVICE BA
R.--- MUST REPAIR VENTILATION IN EMPLOYEE WASHROOM.---  SODA DRAIN LINE DUMPING
INTO 1 COMPARTMENT OF THE 3 COMP BAR SINK. INSTRUCTED TO RELOCATE PROPERLY INTO
FLOOR DRAIN.']
```

Také bychom měli odstranit bílé znaky a budeme ignorovat (odstraníme) komentáře:

In [29]:
```
[s[:s.find('- Comments')].strip() for s in viol.split('|')]
```

Out[29]:
```
['33. FOOD AND NON-FOOD CONTACT EQUIPMENT UTENSILS CLEAN, FREE OF ABRASIVE DETER
GENTS',
 '34. FLOORS: CONSTRUCTED PER CODE, CLEANED, GOOD REPAIR, COVING INSTALLED, DUST
-LESS CLEANING METHODS USED',
 '38. VENTILATION: ROOMS AND EQUIPMENT VENTED AS REQUIRED: PLUMBING: INSTALLED A
ND MAINTAINED']
```

Můžeme si na tuto úpravu řetězce udělat funkci:

In [30]:
```
def strip_comments(s):
    return s[:s.find('- Comments')].strip()
[strip_comments(s) for s in viol.split('|')]
```

Out[30]:
```
['33. FOOD AND NON-FOOD CONTACT EQUIPMENT UTENSILS CLEAN, FREE OF ABRASIVE DETER
GENTS',
 '34. FLOORS: CONSTRUCTED PER CODE, CLEANED, GOOD REPAIR, COVING INSTALLED, DUST
-LESS CLEANING METHODS USED',
 '38. VENTILATION: ROOMS AND EQUIPMENT VENTED AS REQUIRED: PLUMBING: INSTALLED A
ND MAINTAINED']
```

Fajn. Nyní můžeme tuto transformaci aplikovat na každé pole `'Violations'` každé inspekce a z výsledků zkonstruovat čítač. (Následující konstrukci čítače lze zapsat na jednom řádku, zde je to pro přehlednost rozděleno do několika řádků.)

In [31]:
```
c = Counter(
    strip_comments(v)
    for insp in fail
    for v in insp['Violations'].split('|')
)
sum(x for x in c.values())
```

Out[31]:
```
294267
```

In [32]:
```
c.most_common(5)
```

Out[32]:
```
[('34. FLOORS: CONSTRUCTED PER CODE, CLEANED, GOOD REPAIR, COVING INSTALLED, DUS
T-LESS CLEANING METHODS USED',
  19370),
 ('35. WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTRUCTED PER CODE: GOOD REPAIR, SU
RFACES CLEAN AND DUST-LESS CLEANING METHODS',
  18255),
 ('33. FOOD AND NON-FOOD CONTACT EQUIPMENT UTENSILS CLEAN, FREE OF ABRASIVE DETE
RGENTS',
  16445),
 ('18. NO EVIDENCE OF RODENT OR INSECT OUTER OPENINGS PROTECTED/RODENT PROOFED,
A WRITTEN LOG SHALL BE MAINTAINED AVAILABLE TO THE INSPECTORS',
  16431),
 ('38. VENTILATION: ROOMS AND EQUIPMENT VENTED AS REQUIRED: PLUMBING: INSTALLED
AND MAINTAINED',
  15522)]
```

Pokud předcházejícímu postupu není rozumět, můžete postupovat po krocích.

Extrahujme nejprve texty nedostatků ze všech inspekcí:

In [33]:
```
all_violation_texts = [insp['Violations'] for insp in fail]
len(all_violation_texts)
```

Out[33]:
```
47591
```

Pak můžeme každý text s nedostatky rozdělit na popis jednotlivých nedostatků a udělat z nich seznam:

In [34]:
```
all_violations = []
for vtext in all_violation_texts:
    all_violations.extend(vtext.split('|'))
len(all_violations)
```

Out[34]:
```
294267
```

A následně ze všech odstraníme komentáře a bílé znaky:

In [35]:
```
all_violation_codes = [strip_comments(v) for v in all_violations]
len(all_violation_codes)
```

Out[35]:
```
294267
```

A nakonec můžeme seznamem "nakrmit" čítač:

In [36]:
```
c = Counter(all_violation_codes)
c.most_common(5)
```

Out[36]:
```
[('34. FLOORS: CONSTRUCTED PER CODE, CLEANED, GOOD REPAIR, COVING INSTALLED, DUS
T-LESS CLEANING METHODS USED',
  19370),
 ('35. WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTRUCTED PER CODE: GOOD REPAIR, SU
RFACES CLEAN AND DUST-LESS CLEANING METHODS',
  18255),
 ('33. FOOD AND NON-FOOD CONTACT EQUIPMENT UTENSILS CLEAN, FREE OF ABRASIVE DETE
RGENTS',
  16445),
 ('18. NO EVIDENCE OF RODENT OR INSECT OUTER OPENINGS PROTECTED/RODENT PROOFED,
A WRITTEN LOG SHALL BE MAINTAINED AVAILABLE TO THE INSPECTORS',
  16431),
 ('38. VENTILATION: ROOMS AND EQUIPMENT VENTED AS REQUIRED: PLUMBING: INSTALLED
AND MAINTAINED',
  15522)]
```

Shrnutí

Tato "hra" s veřejně dostupnými daty měla ukázat, jak vám znalost datových struktur a nástrojů vestavěných do Pythonu umožní poměrně snadno dělat velké množství věcí na poli analýzy dat, včetně čištění a transformace dat. A to je velmi užitečné!

Podobné věci byste mohli dělat ještě rychleji/lépe/snáze, pokud byste uměli použít specializovaný nástroj pro analýzu tabulkových dat, jako je třeba Pandas. Kromě jiných lze k analýze a vizualizaci dat použít i balíky podporované organizací NumFOCUS.