DataCraft

6
Reguläre Ausdrücke

Reguläre Ausdrücke

DataCraft

- Reguläre Ausdrücke sind eine Möglichkeit, Muster in Strings zu finden und zu ersetzen, z.B. zum Erkennen von Email-Adressen
- (A|B){2,3}[0-9]+ bedeutet z.B. 2-3 der Buchstabe 'A' oder 'B', gefolgt von mindestens einer Ziffer, also AB23, AA0, AAA824 usw.
- Reguläre Ausdrücke sind selber eine kleine Programmiersprache. Es gibt sie in fast allen Programmiersprachen
- 1951 von Stephen Cole Kleene entwickelt, seit den 1970ern weit verbreitet
- Gute Tools im Internet: regex101, regexr

Reguläre Ausdrücke - Übersicht

DataCraft

- Der senkrechte Strich | bedeutet oder: Hallo|Hi meldet einen Treffer bei dem Wort Hallo oder dem Wort Hi

- Runde Klammern werden zur Gruppierung benutzt: H(a|e)llo meldet einen Treffer bei Hallo oder Hello

- Quantoren werden nach einem Zeichen bzw. Gruppe gesetzt, um die Häufigkeit festzulegen:
  - ? bedeutet 0 oder 1 Mal: ab?c entspricht ac und abc
  - * bedeutet beliebig oft (auch 0 Mal)
  - + bedeutet mindestens 1 Mal
  - {n} bedeutet genau n Mal
  - {n,m} bedeutet zwischen n und m mal. Sowohl n als auch m können weggelassen werden

Reguläre Ausdrücke - Übersicht

DataCraft

- Mit eckigen Klammern wird eine Menge von Zeichen definiert: [A-Z] sind z.B. alle Großbuchstaben
- Negation einer Zeichenmenge mit ^ in der eckigen Klammer: [^A-Z] sind alle Zeichen außer den Großbuchstaben
- Der Punkt . steht für ein beliebiges Zeichen
- ^ bedeutet der Anfang vom String, $ das Ende
- Zeichen mit Bedeutung wie \, {, }, [, … können mit dem Backslash \ als normale Zeichen behandelt werden.
- \d steht für eine Ziffer ([0-9]), \w für ein alphanumerisches Zeichen [a-zA-Z0-9_], \b für ein Wort

- Um reguläre Ausdrücke in Python zu verwenden, benötigst Du das Package re.
- `re.search(pattern, str)` gibt den ersten Treffer mit Indizes aus
- `re.findall(pattern, str)` gibt eine Liste mit den Treffern aus
- `re.finditer(pattern, str)` gibt einen Iterator mit Matches zurück
- `re.split(pattern, str)` trennt den String anhand des Pattern
- `re.sub(pattern, ersatz, str)` ersetzt alle Vorkommen des Pattern in str mit ersatz (sub = substitute)

- Achtung bei Verwendung von Sonderzeichen
  - Backslash: Die Strings als raw (mit r davor) initialisieren bzw. den Backslash verdoppeln, da Python selber den Backslash verwendet

```
s = r"Bananen\Äpfel\Birnen"
re.split(r"\\", s)  # oder re.split("\\\\", s)
```

  - Der Punkt steht für ein beliebiges Zeichen. Für den tatsächlichen Punkt \. verwenden

```
re.split("\.", "Bananen.Äpfel.Birnen")
```

- Statt die re-Funktionen direkt zu verwenden, kann der Pattern-String auch zuerst mit re.compile(str) in ein Pattern-Objekt umgewandelt werden. Das ist z.B. in einem Loop sinnvoll.

```
pattern = re.compile(", | und ")
str_liste = ["Bananen, Äpfel und Birnen",
             "Gurken und Erdbeeren",
             "Avocado, Weißkohl"]
for s in str_liste:
    print(pattern.split(s))
```

- Dieses Objekt hat dann die gleichen Funktionen. re.compile hat Flags als Parameter, der wichtigste ist re.I bzw. re.IGNORECASE

- reguläre Ausdrücke (und andere String-Operationen) können mittels apply und einer entsprechenden Funktion auf eine Series bzw. Spalte eines DataFrames angewendet werden. Aber Problem bei leeren Feldern (NaN).

- Besser die Funktionen im str-Attribut benutzen:
  - df["Species"].str.contains("Arabica")
  - df["Species"].str.findall(patterns)
  - df["Species"].str.extract(pattern, flags=re.IGNORECASE)

Stringfunktionen für pandas Series

DataCraft

Funktion	Beschreibung
cat	Verbindet mehrere Strings
contains	**check, ob Pattern enthalten**
count	**Anzahl Vorkommen eines Pattern**
extract	**extrahiert Patterngruppe**
endswith	**check Stringende**
startswith	**check Stringanfang**
findall	**alle Vorkommen eines Patterns**
get	Element an Index
isalnum / isalpha / isdecimal / isdigit / isnumeric	check, ob alphanumerisch, Buchstaben, Dezimalzahl, Ziffer, numerisch
islower / isupper	Check, ob Klein-/Großbuchstaben

Funktion	Beschreibung
join	Liste zu einem String
len	Länge des Strings
lower / upper	Groß-/Kleinbuchstaben
match	**check, ob String Pattern entspricht**
pad / center	fügt Leerzeichen ein
repeat	wiederholt Wert
replace	**ersetzt Pattern**
slice	Teilstring
split	**trennt String bei Trennzeichen**
strip / rstrip / lstrip	entfernt Leerzeichen

Gruppen in regulären Ausdrücken

DataCraft

- Gruppen werden durch Klammern gebildet. Diese können mit \1, \2, … nochmal benutzt werden, sie müssen aber identisch sein

```
regex = r"(Hallo|Hi|Hej) \w+, \1 \w+"
m = re.search(regex, "Hallo Alice, Hallo Bob")  # Treffer
m = re.search(regex, "Hallo Alice, Hi Bob")     # kein Treffer
```

- Die Gruppen können im Match-Objekt angesprochen werden

```
regex = r"(a(b+(c)))"
m = re.search(regex, "abbc")
print(m.groups())
```

- Man kann diese auch mit (?P<name>ABC) benennen.

```
regex = r"^(?P<Tag>\d{1,2})\. (?P<Monat>\w+) (?P<Jahr>\d{4})"
m = re.search(regex, "3. September 2022")
print(m.group('Monat'))
print(m.groupdict())
```

Gruppen in regulären Ausdrücken

DataCraft

- Lookaheads sind nützlich, wenn etwas vor oder nach dem gesuchten Pattern stehen soll, das aber nicht extrahiert werden soll
  - positiver Lookahead: **(?=XXX)**, d.h. nach dem Pattern soll noch ein Pattern kommen, was aber nicht zurückgegeben wird. Z.B. ".*(?=\.bat$)" für bat-Dateien, aber ohne die Endung
  - negativer Lookahead: **(?!XXX)**, d.h. Ausschluss-Kriterium
  - positiver Lookbehind: **(?<=XXX)**, wie positiver Lookahead nur nach "hinten" schauen, d.h. vor dem gewünschten Pattern soll noch etwas stehen
  - negativer Lookbehind **(?<!XXX)**, d.h. Ausschluss-Kriterium