Regular Expressions & Types of Missing Values

Regular Expressions

What is a Regular Expression

- A regular expression (also call regex) is a sequence of characters that defines a search pattern
- They are used to match and manipulate text
- Common uses:
  - Pattern matching - search for and extract specific data, such as emails, phone numbers, etc.
  - Data validation - check that input data matches a specific format
  - Text cleaning - remove punctuation, whitespace, or replace parts of or whole strings with other text
  - Web-scraping - find specific information with HTML

Examples

- "`[a-z]+`": matches any sequence of one or more lowercase letters
- "`\d{3}-\d{2}-\d{4}`": matches a number (e.g. SSN) in the format of 123-45-6789
- "`^https?://`": matches the beginning of a URL (either http or https)
- "`[aeiouyAEIOUY]`": matches any vowel (and y)

Syntax of Regular Expressions

- Characters: a sequence of characters that define a search pattern
- Metacharacters: special characters that have a specific meaning
  . ^ $ * + ? { } [ ] \ | ( )
- Character classes: groups of characters that can match any one of a set of characters (specific by [ ])
- Quantifiers: used to specify the number of occurrences of a character or group of characters
- Anchors: used to match the position
- Alternation

Metacharacters

- [ ]: Specifies character classes
- \: escapes other meta characters and specifies special sequences
- { }: Quantifier - specifies number of repeats
- *: Quantifier - specifies zero or more times
- +: Quantifier - specifies one or more times
- ?: Quantifier - specifies zero or one time (has special meaning inside groups)
- ^: Specifies start of pattern OR opposite if start of character class
- $: Specifies the end of a pattern
- . : matches any character except a newline
- | : Alternation (the “or” operator)
- () : Specifies groups

Character classes

- Character classes are specified using []
- These are groups of characters that any one of which will satisfy the search
  - Ranges can also be specified: `[a-zA-Z]`
- Metacharacters (except \) are not active inside classes
  - `[akm$]` will match any of “a”, “k”, “m”, or “$” (even though $ is a special character, it doesn’t have special meaning inside the class)
- If the character “^” is the first character inside a class, then it indicates the complement set
  - `[^aeiou]` will match any character except lowercase vowels
  - If ^ appears later in a class then it has no special meaning

The backslash \

- The metacharacter \
  - Signals various special sequences (next slide)
  - Used to escape metacharacters so they can be matched in patterns
    - It can even escape itself: `\\` will search for the character \ in the pattern

Groups

- Capturing groups allow extraction of specific parts of the text matched by a regex pattern
  - Groups are enclosed in `()`
- Once a pattern finds a match, the part of the string matched by the capture group can be accessed separately from the entire match
- More than one capturing group is allowed
- Using `?:` at the start of the string inside `()` will instead use a "non-capturing" group

Groups

- Example:

```
pattern = r'\d{2}-\d{2}-(\d{4})'
st = "Today's date is 03-05-2024"
re.findall(pattern, st)

['2024']
```

- Example:

```
pattern = r'(\d{2})-(\d{2})-(\d{4})'
st = "Today's date is 03-05-2024"
match = re.search(pattern, st)

match.group(1) ## returns 03
match.group(2) ## returns 05
match.group(3) ## returns 2024

match.group(0) ## returns 03-05-2024
match.group()  ## same as group(0)
```

Lookahead and Lookbehind

- "lookahead" assertions match pattern only if it occurs before another specific pattern
  - Positive **`(?=…)`**
    `r'\d(?=%)'`: Match digit only if before "%"
  - Negative **`(?!…)`**
    `r'\d(?!%)'`: Match digit only if not before "%"
- "lookbehind" assertions match a pattern only if it occurs after another specific pattern
  - Positive **`(?<=…)`**
    `r'(?<=%)\d``: Match digit only if after "%"
  - Negative **`(?<!…)`**
    `r``(?<!%)\d`**`: Match digits only if **not** after "%"

Lookahead and Lookbehind

```
text = '3% 4 %5 $6 78'
print(re.findall(r'\d(?=%)',text))
print(re.findall(r'\d(?!%)',text))
print(re.findall(r'(?<=%)\d',text))
print(re.findall(r'(?<!%)\d',text))
```

0.0s

```
['3']
['4', '5', '6', '7', '8']
['5']
['3', '4', '6', '7', '8']
```

Some special sequences

Character	Description	Example code	Example match
\d	A digit	file_\d\d	file_21
\w	Alphanumeric	\w-\w\w\w	A-b23
\s	White space	a\sb\sc	a b c
\D	A non digit	\D\D\D	abc
\W	Non-alphanumeric	\W\W\W\W\W	*-+=)
\S	Non-whitespace	\S\S	Hi

These sequences can be used inside character classes.
For example `[\s,.]` will match any whitespace, “,” or “.”

Quantifiers

Quantifier should go after a pattern to specify the quantity in the search

Character	Description	Example code	Example match
+	Occurs one or more times	\w-\w+	A-bbb32
{d}	Occurs exactly d times	\D{3}	Abc or)))
{d1,d2}	Occurs d1 to d2 times	\d{2,4}	12 or 345
{d,}	Occurs d or more times	\w{3,}	Anything3ormore
*	Occurs zero or more times	A*B*C*	AAACCC
?	Occurs one or zero times	colou?r	color or colour

Python `re` library

- **Performing matches**

- `re.findall(pattern, string)` - find all substrings where the RE matches and returns as a list

- `re.match(pattern, string)` - determines if RE matches at the beginning of the string - returns a "match" object or None

- `re.search(pattern, string)` - Finds first location where the RE matches - returns a "match" object or None

- `re.finditer(pattern, string)` - returns an iterable of match objects

- `re.compile(pattern)` - creates a pattern object from which these methods can be called

Python `re` library

- **Modifying String**

- `re.split(pattern, string)` - splits string where RE matches

- `re.sub(pattern, replace, string)` - replaces RE matches in the string with replace

- `re.subn(pattern, replace, string)` - same as sub() but returns a tuple with (new string, number of replacements)

`re` isn't always necessary

- Python's built-in string methods are powerful and can do a lot by themselves
  - `.endswith()`
  - `.startswith()`
  - `.join()`
  - `.find()`
  - `.replace()`
  - `.strip(), .rstrip(), .lstrip()`
    - `.strip('#')` (*strip characters besides whitespace)*
  - `.split()`
  - `.lower(), .upper()`

Applying to pandas

- String and regular expression methods are called off of **string** objects
- To apply to a Series or DataFrame
  - Use the `.str` attribute
  - Use the `.apply()` method and a lambda function

Types of Missing Values

Missing values

- Dealing with missing values is a common challenge in data analysis
- Understanding the nature of missing values is important for finding strategies to handle them
- Types of missing values:
  - Structurally missing
  - Missing completely at random (MCAR)
  - Missing at random (MAR)
  - Missing not at random (MNAR)

Structurally Missing

- Missing due to the structure of the data
- Examples
  - In a survey about employment, the question "What is your current employer?" would be structurally missing for respondents who are unemployed because the question does not apply to them.
  - In a medical dataset, data about pregnancy tests would be structurally missing for male patients, as they are not applicable.

Missing Completely at Random (MCAR)

- The probability of data being missing is the same for all observations
- The missingness is unrelated to any observed or unobserved data
- Examples:
  - If a survey is conducted both online and on paper, and some of the paper surveys were lost due to an accident, the responses from these surveys would be missing completely at random.
  - During data entry from physical forms to a digital database, some data entries are accidentally skipped or lost due to human error, unrelated to any specific characteristic of the forms or the data being entered.

Missing at Random (MAR)

- Missingness is related to the observed data but not the unobserved data
- For any set of data with the same observed values, the missingness is random
- Examples
  - In a study that collects data on income and education level, if higher education level individuals are more likely to skip income-related questions, the missingness in income data is related to education level (an observed variable) but not to the income itself (unobserved).
  - In a health survey, younger participants may be less likely to report their weight. Here, the missingness of weight information is related to the age of the participants (observed data) but not necessarily to their actual weight (unobserved data).

Missing Not at Random (MNAR)

- Missingness depends on information that is missing from the dataset
- The unobserved data is the *reason* why it is missing
- Examples:
  - If individuals with very high or very low incomes are less likely to disclose their income on a survey, the missing income data is related to the income level itself, which is unobserved for those who did not disclose it.
  - In a clinical trial, if patients experiencing severe side effects are more likely to drop out, the missing data on these patients' outcomes are not random and are related to the unobserved severity of their side effects.

How to handle missing values?

- Structurally missing
  - Exclude variable that do not apply to all data
  - Create an indicate variable to denote applicability
- MCAR
  - Drop rows or columns with missing
  - Simple imputation with variable mean, median or mode is usually suitable

How to handle missing values?

- MAR
  - Multiple imputation method (MICE is common)
    - Uses multiple imputation techniques and then combines the results to produce a final imputed dataset
  - Predict missing from other variable
- MNAR
  - (Hard)
  - Sensitivity analysis of different methods