C: Data cleaning techniques

1. Methods for cleaning up strings

   a. Making **case** consistent for subsequent joins:

```
data['clean'] = data['raw'].str.lower()
```

   b. Making spacing consistent:

```
raw_parts = data['raw'].str.split()
data['clean'] = raw_parts.apply(' '.join)
```

- Applies a function to the values of a Series
- Can apply most functions that take a single argument

Example:

Raw:

Index	raw
0	'some words here '
1	'other words there'

After split:

Index	
0	['some','words','here']
1	['other','words','there']

After join:

Index	clean
0	'some words here'
1	'other words there'

c. Removing characters via regular expressions (REs)

- Discussed earlier in the semester
- **Very** powerful **pattern-matching** tools for manipulating text.
- For reference, repeating some frequently used **single-character** REs:

RE	**What it matches**
\s	Any whitespace character: space, tab, newline, return, etc.
\S	Any character that is NOT whitespace
\d	Any digit: 0-9
\D	Any character that is NOT a digit
\w	Any word character: letters, digits, underscore
\W	Any character that is NOT a word character
.	Any character except a newline
^	Just before the first character of the string
$	Just after the last character of the string

- Combining to match a **sequence** of characters:

RE	**What it matches**
\d\d	Any two digits
\w\w\w	Any three word characters
\w\S	A word character followed by any non-whitespace character
\A\w	A word character at the beginning of the string

- Matching **sequences of varying** length:

RE	What it matches
\d+	One or more digits
\w+	One or more word characters
\s+	One or more whitespace characters

- Combining with a vertical bar to match one or the other:

RE	What it matches
\d|\s	A digit or a space
a|\d	Letter 'a' or a digit

- Prefix RE string **with r** to avoid excess \'s

```
new = old.str.replace( r'\D', '', regex=True )
```

Would remove any non-digit from the string.

2. Combining two series using where()

Builds a new series based on an element-by-element Boolean test

```
new_series = old_series.where( test_condition, alt_value )
```

For elements where **test_condition is True**:
new_series ← **old_series**

For elements where **test_condition is False**:
new_series ← **alt_value**

Example:

```
new_series = old_series.where( old_series >= 0 , None )
```

old_series new_series

Index	
0	-1
1	2
2	3
3	-5
4	0

Index	
0	NaN
1	2
2	3
3	NaN
4	0

3. Filling missing data using fillna()

Basic usage: replaces missing data with the argument it was given

Example:

```
filled = original.fillna(0)
```

Can also fill using data in other ways using the method keyword:

method='ffill': Carry **previous** non-missing value **forward**
method='bfill': Carry **next** non-missing value **backward**