# Wrangling III homework

## Deleting invalid rows from the cancer data

It's been decided by committee that duplicate data in the (smaller version of) the cancer data set is going to go as follows. If a row is identical to the one immediately above it, we'll consider it an accidental entry due to fatigue or whatever. But a row that is *not* identical to the one above it will be considered valid, even it has a duplicate somewhere else in the data set; we'll assume such duplicates represent separate visits.

So our rule is: **if a row is identical to the one above it, we drop it**.

A small pre-cleaned version of the data with only 4 columns is in **small_cancer_data.csv**, so you can read it in directly without having to clean it up.

Spend a minute or two thinking about how you would approach this problem.

If you are ready to go on your own, then go!

Once you have working code – once you can take **small_cancer_data.csv** and trim out the unwanted rows – then wrap your code into a function, so all you have to do drop unwanted rows is call your function!

---

Spend some time thinking about and working on the problem. If you get to an impass and you'd like some hints, read on.

---

## Preliminaries

As usual, we'll load some libraries we'll be likely to use.

In [1]:
```
import pandas as pd
```

---

## Make a mini data set for testing

Rather than taking a crack at the whole set, make a small data frame named `tiny` with 10 rows and two columns. Put successive repeated rows in two places (like rows 2 and 3 could repeat, as could rows 6 and 7). Put an additional repeated row on it's own.

Something like this:

In [2]:
```
tiny = pd.DataFrame(dict(a = [1, 2, 3, 3, 4, 5, 5, 5, 6, 3],
                         b = ['a', 'b', 'c', 'c', 'd', 'e', 'e', 'f', 'g',
```

Check our tiny data frame.

In [3]:
```
# check your test data frame
tiny
```

Out[3]:

| | a | b |
|---|---|---|
| **0** | 1 | a |
| **1** | 2 | b |
| **2** | 3 | c |
| **3** | 3 | c |
| **4** | 4 | d |
| **5** | 5 | e |
| **6** | 5 | e |
| **7** | 5 | f |
| **8** | 6 | g |
| **9** | 3 | c |

There should be two rows that need to be dropped, and one (the last) that should be kept even though it's a duplicate.

Just to be sure, check the output of `.duplicated(keep=False)` – it should show back-to-back `True` values in 2 places, and one solo `True` at the end.

In [4]:
```
# check .duplicated output
tiny.duplicated(keep = False)
```

Out[4]:
```
0    False
1    False
2     True
3     True
4    False
5     True
6     True
7    False
8    False
9     True
dtype: bool
```

---

## Make a plan

There are probably 100 ways to solve this problem. Many are probably very clever and involve using fancy pandas functions.

A straightforward plan using things we already know about might be something like

- go through the rows of the data frame with a `for` loop, starting with the second row
- at each row, compare the current row with the previous one
- if they're the same, save the index of the current row
- after the `for` loop, delete the unwanted rows using the saved indexes

---

# Test the parts of the plan

Now that we've got a plan, let's get the pieces of the plan to work before putting the whole plan together.

## Make sure we can get rows

We should be able to get rows of a data frame in a couple of ways. These are

- using `.loc[]` with the value of rows index (it's name)
- using `.iloc[]` and indexing into the data like it were a numpy array

Let's try the `.loc[]` way.

In [13]:
```
tiny.loc[1]
```

Out[13]:
```
a    2
b    b
Name: 1, dtype: object
```

And let's try the `.iloc[]` method.

In [14]:
```
tiny.iloc[1]
```

Out[14]:
```
a    2
b    b
Name: 1, dtype: object
```

Look's like either will work!

## Figure out how to compare rows

We are going to need to compare rows. Let's see how that is going to work.

### compare the first and second rows - these *should **not*** match

In [91]:
```
# which things in the rows match?
tiny.loc[0] == tiny.loc[1]
```

Out[91]:
```
a    False
b    False
dtype: bool
```

### compare the third and fourth rows - these *should* match

In [17]:
```
# which things in the rows match?
tiny.loc[2] == tiny.loc[3]
```

Out[17]:
```
a    True
b    True
dtype: bool
```

The rows only match if ***all*** the columns match, so we can see if this is the case with the `all()` function.

In [95]:
```
# do all the columns match?

(tiny['a'] == tiny['b']).all()
```

Out[95]:
```
False
```

Now we have a way to compare rows and get a single `True` if the rows are identical, and a `False` if they're not.

And now that we know how to do the row comparison, let's get a `for` loop working.

## Confirm we can get rows with a `for` loop

*Loop through the first few rows*

Let's make sure we can index into rows with a for loop. Let's try to get the first few using `.loc[]` and print them. Like

```
for ... :
    print(...)
```

In [98]:
```
# loop through the first few rows
for i in range(0, 3):
    print(tiny.loc[i])
```

```
a    1
b    a
Name: 0, dtype: object
a    2
b    b
Name: 1, dtype: object
a    3
b    c
Name: 2, dtype: object
```

*Loop through the **all** rows*

To loop through all the rows, we first need to get the number of rows. We can do this using the `shape` attribute.

In [8]:
```
# get the number of rows using shape
tiny.shape[0]
```

Out[8]:
```
10
```

In [97]:
```
# loop through all the rows
for i in range(tiny.shape[0]):
```

```
    print(tiny.loc[i])
```

```
a    1
b    a
Name: 0, dtype: object
a    2
b    b
Name: 1, dtype: object
a    3
b    c
Name: 2, dtype: object
a    3
b    c
Name: 3, dtype: object
a    4
b    d
Name: 4, dtype: object
a    5
b    e
Name: 5, dtype: object
a    5
b    e
Name: 6, dtype: object
a    5
b    f
Name: 7, dtype: object
a    6
b    g
Name: 8, dtype: object
a    3
b    c
Name: 9, dtype: object
```

---

## Putting it all together

Get the number of rows

In [13]:
```
# get the number of rows using shape
row = tiny.shape[0]
row
```

Out[13]:
```
10
```

Make an empty list to hold the indexes of the columns we're going to drop

In [56]:
```
drop_list = []
```

Make a `for` loop that

- goes from 1 (i.e. the second row) to the end
- tests the current row against previous
- stores index for dropping

In [57]:
```
for i in range(1,row):
    if (tiny.loc[i] == tiny.loc[i-1]).all():
        drop_list.append(i)
```

Check that we got the correct indexes.

In [58]:
```
drop_list
```

Out[58]:
```
[3, 6]
```

Make a new data frame with the unwanted rows `.drop` ped.

In [60]:
```
new_tiny = tiny.drop(drop_list)
new_tiny
```

Out[60]:

| | a | b |
|---|---|---|
| **0** | 1 | a |
| **1** | 2 | b |
| **2** | 3 | c |
| **4** | 4 | d |
| **5** | 5 | e |
| **7** | 5 | f |
| **8** | 6 | g |
| **9** | 3 | c |

Use `.reset_index()` to make a new sequental index for our data frame.

In [61]:
```
new_tiny.reset_index()
```

Out[61]:

| | index | a | b |
|---|---|---|---|
| **0** | 0 | 1 | a |
| **1** | 1 | 2 | b |
| **2** | 2 | 3 | c |
| **3** | 4 | 4 | d |
| **4** | 5 | 5 | e |
| **5** | 7 | 5 | f |
| **6** | 8 | 6 | g |
| **7** | 9 | 3 | c |

Marvel at your work!

In [63]:
```
new_tiny.reset_index(drop = True)
```

Out[63]:

| | a | b |
|---|---|---|
| **0** | 1 | a |
| **1** | 2 | b |
| **2** | 3 | c |
| **3** | 4 | d |
| **4** | 5 | e |
| **5** | 5 | f |
| **6** | 6 | g |
| **7** | 3 | c |

If you don't like the "index" column with old indexes (sometimes it's useful to have the old indexes – here it's just annoying), you can set `drop=True` when you call `.reset_index()` above.

---

# Run your code on the cancer data

Try our code on the (small version of the) cancer data!

## Load the data

In [68]:
```
small_bcd = pd.read_csv("data/small_cancer_data.csv")
small_bcd
```

Out[68]:

| | id | thick | chrom | class |
|---|---|---|---|---|
| **0** | 1000025 | 5.0 | 3.0 | benign |
| **1** | 1002945 | 5.0 | 3.0 | benign |
| **2** | 1015425 | 3.0 | 3.0 | benign |
| **3** | 1016277 | 6.0 | 3.0 | benign |
| **4** | 1017023 | 4.0 | 3.0 | benign |
| **...** | ... | ... | ... | ... |
| **694** | 776715 | 3.0 | 1.0 | benign |
| **695** | 841769 | 2.0 | 1.0 | benign |
| **696** | 888820 | 5.0 | 8.0 | malignant |
| **697** | 897471 | 4.0 | 10.0 | malignant |
| **698** | 897471 | 4.0 | 10.0 | malignant |

699 rows × 4 columns

## Get the number of rows

In [72]:
```
# get the number of rows using shape
sbcd_row = small_bcd.shape[0]
sbcd_row
```

Out[72]:
```
699
```

## Make an empty list for indexes

In [70]:
```
small_list = []
```

## Run your `for` loop!

In [73]:
```
for i in range(1,sbcd_row):
    if (small_bcd.loc[i] == small_bcd.loc[i-1]).all():
        small_list.append(i)
```

## Check the indexes you found

In [74]:
```
small_list
```

Out[74]:
```
[208, 322, 443, 561, 684, 690, 698]
```

## Drop the unwanted rows

In [75]:
```
new_sbcd = small_bcd.drop(small_list)
```

## Reset the row indexes

In [78]:
```
new_sbcd.reset_index(drop = True)
```

Out[78]:

| | id | thick | chrom | class |
|---|---|---|---|---|
| **0** | 1000025 | 5.0 | 3.0 | benign |
| **1** | 1002945 | 5.0 | 3.0 | benign |
| **2** | 1015425 | 3.0 | 3.0 | benign |
| **3** | 1016277 | 6.0 | 3.0 | benign |
| **4** | 1017023 | 4.0 | 3.0 | benign |
| **...** | ... | ... | ... | ... |
| **687** | 763235 | 3.0 | 2.0 | benign |
| **688** | 776715 | 3.0 | 1.0 | benign |
| **689** | 841769 | 2.0 | 1.0 | benign |
| **690** | 888820 | 5.0 | 8.0 | malignant |
| **691** | 897471 | 4.0 | 10.0 | malignant |

692 rows × 4 columns

### Check the shape to confirm the rows were dropped!

In [79]:
```
new_sbcd.shape
```

Out[79]:
```
(692, 4)
```

---

## Wrapping it all in a function

Once you've got your code running, put it all in a function so it's reusable!

In [99]:
```
def drop_repeats(df):
    row = df.shape[0]                           # get the number of rows
    drop_list = []                              # empty list to store the inde


    for i in range(1,row):                      # finds all the repeating rows
        if (df.loc[i] == df.loc[i-1]).all():
            drop_list.append(i)

    new_df = df.drop(drop_list)                 # drop the unwanted rows

    new_df.reset_index(drop = True)             # reset the row index

    return new_df
```

Run your function!

In [102…
```
print(small_bcd.shape[0])
new_bcd = drop_repeats(small_bcd)
```

```
699
```

Check the shape to confirm your function worked!

In [90]:
```
new_bcd.shape
```

Out[90]:
```
(692, 4)
```

### High-five the person closest to you!

Because you deserve a high-five right now.

---