Filtering columns and rows in pandas

This notebook has a little more detail on selecting and filtering data in pandas. We'll use the MLB salary data as an example -- a CSV lives at `'../data/mlb.csv'` .

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

In [ ]:
```
# use the read_csv() method to create a data frame
df = pd.read_csv('../data/mlb.csv')
```

In [ ]:
```
# use the `head()` method to check out what we've got
df.head()
```

Selecting one column of data

You can select a column of data using dot notation `.` or bracket notation: `[]` .

If you want to select a single column of data and the column name doesn't have spaces, you can use a period ("dot notation"). You could also pass the name of the column as a string inside square brackets ("bracket notation"); if your column names have spaces (avoid this if you can), you *must* use bracket notation.

Let's say we wanted to select the `TEAM` column. We could do this:

In [ ]:
```
df.TEAM
```

... or we could do this:

In [ ]:
```
df['TEAM']
```

Either works. And: Not a huge deal, but it's generally good practice to pick one and stay consistent, at least within the same script.

Selecting multiple columns of data

To select multiple columns of data, you use *bracket notation* -- but instead of putting a single column name inside the brackets, you hand it a *list* of column names.

👉 For a refresher on *lists*, check out this notebook.

Let's select the `NAME` and `TEAM` columns.

In [ ]:
```
name_and_team = df[['NAME', 'TEAM']]

name_and_team.head()
```

Lots of square brackets happening there! An alternative: You could assign the list of column names that you want to select to its own variable to make things a little clearer.

In [ ]:
```
cols_of_interest = ['TEAM', 'NAME']
df[cols_of_interest].head()
```

A good rule of thumb: If you can do anything in your script to make things clearer, or more explicit, for other people reading your code (including your future self), do it.

Filtering rows of data

You can also filter your data set to keep just the rows that meet your filtering condition(s) -- like using the filter dropdowns in Excel or a `WHERE` clause in SQL.

Let's say you wanted to filter our MLB data to include just the Los Angeles Dodgers.

First, we need to make sure that we understand how "Los Angeles Dodgers" is represented in the data. We can use the `unique()` method to get a unique list of values in the `TEAM` column.

In [ ]:
```
df.TEAM.unique()
```

So we want to find all the records in our data where the value in the `TEAM` column is "LAD". The equivalent SQL would be:

```
SELECT *
FROM mlb
WHERE TEAM = "LAD"
```

With pandas, the basic syntax is to pass your filtering condition to the data frame in square brackets `[]`. We'll use Python's `==` comparison operator to test for equality. Typically, you'd also want to "save" the results of your filtering by assigning the result to a variable that you can access later:

In [ ]:
```
lad = df[df['TEAM'] == 'LAD']
```

In [ ]:
```
lad.head()
```

You can do numerical comparisons -- let's get just the players who make $1 million or more:

In [ ]:
```
millionaires = df[df['SALARY'] >= 1000000]
```

In [ ]:
```
millionaires.head()
```

Again, if it makes your script clearer, more readable, you can break your filter up into multiple pieces -- "save" the filtering condition under a new variable and *then* hand that off to the data frame:

In [ ]:
```
is_a_2_millionaire = df['SALARY'] >= 2000000
two_millionaires = df[is_a_2_millionaire]
```

In [ ]:
```
two_millionaires.head()
```

Filtering against multiple matches

You can use the `isin()` method to test a value against multiple matches -- just hand it your list of values to check against.

Let's say we wanted to return all of the players for the Texas Rangers and Houston Astros.

In [ ]:
```
tx = df[df['TEAM'].isin(['TEX', 'HOU'])]
```

In [ ]:
```
tx.TEAM.unique()
```

NOT filtering

You can filter for data that do *not* meet some criteria by prepending a tilde `~`.

Let's filter for all players that *aren't* in Texas. It'll be the same as the filtering statement above but with a tilde at the beginning.

In [ ]:
```
not_tx = df[~df['TEAM'].isin(['TEX', 'HOU'])]
```

In [ ]:
```
not_tx.TEAM.unique()
```

Filtering on multiple criteria

You can filter your data on multiple criteria. A few gotchas:

- Don't use Python's `and` and `or` operators to chain the statements -- pandas wants you to use `&` and `|`
- Don't forget to use parentheses to group your statements

Let's filter for all catchers who make the league minimum of $535,000.

In [ ]:
```
catchers_lm = df[(df['POS'] == 'C') & (df['SALARY'] == 535000)]
```

In [ ]:
```
catchers_lm
```

Much of the time, though, it's clearer to just break up your filtering into multiple statements, like this:

In [ ]:
```
catchers = df[df['POS'] == 'C']
catchers_lm = catchers[catchers['SALARY'] == 535000]

catchers_lm
```