Pandas Recipes for New Python Users

Some Jupyter lab notes:

- Jupyter lab let's us make cells and run code in a nicely formatted way
- We also can use things like magic cells - these allow us to do special operations on code
- Rerunning cells is super easy
- Has built in support for dataframes
- Nice support for CSV or TSV viewing

Here's a brief demo:

In [1]:
```
x = 100 * 100
x
```

Out[1]:
```
10000
```

In [2]:
```
import random
for something in range(random.randint(1,10)):
    print(something)
```

```
0
1
2
3
4
5
6
7
8
```

Loading Data

In [3]:
```
import pandas as pd
```

In [4]:
```
df = pd.read_csv('mpg.tsv', sep='\t')
df
```

Out[4]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86.0	2790.0	15.6	82	1	ford mustang gl
394	44.0	4	97.0	52.0	2130.0	24.6	82	2	vw pickup
395	32.0	4	135.0	84.0	2295.0	11.6	82	1	dodge rampage
396	28.0	4	120.0	79.0	2625.0	18.6	82	1	ford ranger
397	31.0	4	119.0	82.0	2720.0	19.4	82	1	chevy s-10

398 rows × 9 columns

In [5]:
```
df.head()
```

Out[5]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino

What is a DataFrame?

"Two-dimensional, size-mutable, potentially heterogeneous tabular data."

DataFrames have:

- Values
- An Index
- Columns
- a Shape

In [6]:
```
df.axes
```

Out[6]:
```
[RangeIndex(start=0, stop=398, step=1),
 Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
        'acceleration', 'model_year', 'origin', 'car_name'],
       dtype='object')]
```

What if I want a quick summary?

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
mpg             398 non-null float64
cylinders       398 non-null int64
displacement    398 non-null float64
horsepower      392 non-null float64
weight          398 non-null float64
acceleration    398 non-null float64
model_year      398 non-null int64
origin          398 non-null int64
car_name        398 non-null object
dtypes: float64(5), int64(3), object(1)
memory usage: 28.1+ KB
```

Accessing Data

In [8]:
```
name_df = df['car_name']
name_df
```

Out[8]:
```
0      chevrolet chevelle malibu
1              buick skylark 320
2             plymouth satellite
3                  amc rebel sst
4                    ford torino
                  ...
393               ford mustang gl
394                     vw pickup
395                dodge rampage
396                  ford ranger
397                   chevy s-10
Name: car_name, Length: 398, dtype: object
```

We can see this looks a little different? Why?

In [9]:
```
type(name_df)
```

Out[9]:
```
pandas.core.series.Series
```

In [10]:
```
type(df)
```

Out[10]:
```
pandas.core.frame.DataFrame
```

Under the hood, each column in a Pandas DataFrame is a Series.

A series can be thought of as a one-dimensional array.

With a series we're able to get back to what we are familiar with.

In [11]:
```
name_df[100]
```

Out[11]: 'ford maverick'

Creating New Columns

To create a new column in Pandas, the strategy is to "pretend it was always there".

In [12]:
```
df['disp_div_cyl'] = df['displacement'].div(df['cylinders'])
df
```

Out[12]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86.0	2790.0	15.6	82	1	ford mustang gl
394	44.0	4	97.0	52.0	2130.0	24.6	82	2	vw pickup
395	32.0	4	135.0	84.0	2295.0	11.6	82	1	dodge rampage
396	28.0	4	120.0	79.0	2625.0	18.6	82	1	ford ranger
397	31.0	4	119.0	82.0	2720.0	19.4	82	1	chevy s-10

398 rows × 10 columns

In [13]:
```
new_df = df.copy()
new_df['acceleration'] = df['acceleration'] * 100
new_df
```

Out[13]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
0	18.0	8	307.0	130.0	3504.0	1200.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	1150.0	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	1100.0	70	1	plymouth satellite

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
3	16.0	8	304.0	150.0	3433.0	1200.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	1050.0	70	1	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86.0	2790.0	1560.0	82	1	ford mustang gl
394	44.0	4	97.0	52.0	2130.0	2460.0	82	2	vw pickup
395	32.0	4	135.0	84.0	2295.0	1160.0	82	1	dodge rampage
396	28.0	4	120.0	79.0	2625.0	1860.0	82	1	ford ranger
397	31.0	4	119.0	82.0	2720.0	1940.0	82	1	chevy s-10

398 rows × 10 columns

Iterating over a DataFrame

We have the ability to load data, create new data, inspect data... How do we process it?

In [14]:
```
for index,row in df.head(3).iterrows():
    print(row)
```

```
mpg                                  18
cylinders                             8
displacement                        307
horsepower                          130
weight                             3504
acceleration                         12
model_year                           70
origin                                1
car_name        chevrolet chevelle malibu
disp_div_cyl                     38.375
Name: 0, dtype: object
mpg                              15
cylinders                         8
displacement                    350
horsepower                      165
weight                         3693
acceleration                   11.5
model_year                       70
origin                            1
car_name        buick skylark 320
disp_div_cyl                43.75
Name: 1, dtype: object
mpg                              18
cylinders                         8
displacement                    318
horsepower                      150
weight                         3436
acceleration                     11
model_year                       70
```

```
origin                          1
car_name        plymouth satellite
disp_div_cyl                39.75
Name: 2, dtype: object
```

Conditional Subsetting

Two main functions: `loc` and `iloc`. We'll be focusing on `loc`.

The `iloc` function is used to numerically index rows.

In [15]:
```
df.loc[:,:]
```

Out[15]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86.0	2790.0	15.6	82	1	ford mustang gl
394	44.0	4	97.0	52.0	2130.0	24.6	82	2	vw pickup
395	32.0	4	135.0	84.0	2295.0	11.6	82	1	dodge rampage
396	28.0	4	120.0	79.0	2625.0	18.6	82	1	ford ranger
397	31.0	4	119.0	82.0	2720.0	19.4	82	1	chevy s-10

398 rows × 10 columns

In [16]:
```
df.loc[:,'mpg']
```

Out[16]:
```
0      18.0
1      15.0
2      18.0
3      16.0
4      17.0
       ...
393    27.0
394    44.0
395    32.0
396    28.0
```

```
397    31.0
Name: mpg, Length: 398, dtype: float64
```

In [17]:
```
df.loc[:,['mpg', 'displacement']]
```

Out[17]:

	mpg	displacement
0	18.0	307.0
1	15.0	350.0
2	18.0	318.0
3	16.0	304.0
4	17.0	302.0
...	...	...
393	27.0	140.0
394	44.0	97.0
395	32.0	135.0
396	28.0	120.0
397	31.0	119.0

398 rows × 2 columns

In [18]:
```
df.loc[df['cylinders'] > 4,['mpg', 'displacement']]
```

Out[18]:

	mpg	displacement
0	18.0	307.0
1	15.0	350.0
2	18.0	318.0
3	16.0	304.0
4	17.0	302.0
...	...	...
365	20.2	200.0
366	17.6	225.0
386	25.0	181.0
387	38.0	262.0
389	22.0	232.0

190 rows × 2 columns

All of these DataFrames can be assigned to their own variable and processed however you like.

Accessors on Columns

In [19]:
```
df['car_name'].dt
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-19-af764a404083> in <module>
----> 1 df['car_name'].dt

~/.pyenv/versions/main-env/lib/python3.8/site-packages/pandas/core/generic.py in
__getattr__(self, name)
   5173             or name in self._accessors
   5174         ):
-> 5175             return object.__getattribute__(self, name)
   5176         else:
   5177             if self._info_axis._can_hold_identifiers_and_holds_name(name
):

~/.pyenv/versions/main-env/lib/python3.8/site-packages/pandas/core/accessor.py i
n __get__(self, obj, cls)
    173             # we're accessing the attribute of the class, i.e., Dataset.
geo
    174             return self._accessor
--> 175         accessor_obj = self._accessor(obj)
    176         # Replace the property with the accessor object. Inspired by:
    177         # http://www.pydanny.com/cached-property.html

~/.pyenv/versions/main-env/lib/python3.8/site-packages/pandas/core/indexes/acces
sors.py in __new__(cls, data)
    341             pass  # we raise an attribute error anyway
    342
--> 343         raise AttributeError("Can only use .dt accessor with datetimelik
e " "values")

AttributeError: Can only use .dt accessor with datetimelike values
```

In [ ]:

```
df['car_name'].str.contains('ford')
```

Explore the API docs for all of the functionality that these afford.