Pandas Tutorial

April 18, 2021

1 Introduction To Pandas Tutorial

Author: Vi Ly
Date: 21 Mar 2021
LinkedIn: https://www.linkedin.com/in/vi-ly-2810ba59/

The easiest way to install Python and pandas is through Anaconda: https://www.anaconda.com/products/individual.

pandas is a very expansive package and this tutorial only covers a portion of its capability. For further reading, refer to the documentation: https://pandas.pydata.org/docs/reference/index.html

This tutorial will use the **Auto MPG** Dataset from UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/auto+mpg

1.1 Table of Contents

2 Importing Pandas

It is standard convention to alias **pandas** as **pd**.

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

Import additional libraries.

[2]:
```
import os

import numpy as np
```

3 Reading in Data

Python Variable Naming Rules - Can only contain letters, numbers, and underscores - Must start with either letter or underscore; cannot start with number - Case-sensitive

Use pandas to read in a csv, using the **.read_csv()** method, as a DataFrame and assign it to the variable mpg_df.

Pandas offers a vast array of functions to read in different file types. Refer to the documentation.

[3]:
```
mpg_df = pd.read_csv(f'c:/users/{os.getlogin()}/desktop/mpg dataset.csv')
```

4 Basic Functionality

Get the column names by calling the **.columns** attribute

[4]:
```
mpg_df.columns
```

[4]:
```
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model_year', 'origin', 'car_name'],
      dtype='object')
```

Get the DataFrame dimensions by calling the **.shape** attribute.

[5]:
```
mpg_df.shape
```

[5]:
```
(406, 9)
```

You can also get the # of rows and columns using the **len()** function.
Get the number of columns in the DataFrame.

[6]:
```
len(mpg_df.columns)
```

[6]:
```
9
```

Get the number of rows in the DataFrame.

[7]:
```
len(mpg_df)
```

[7]:
```
406
```

The **.head()** method, by default, returns the first 5 rows in the DataFrame.

[8]:
```
mpg_df.head()
```

[8]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
0	18.0	8	307.0	130.0	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	1	ford torino

Providing an integer to the **.head()** method returns the specified first X rows.

[9]:
```
mpg_df.head(10)
```

[9]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
0	18.0	8	307.0	130.0	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	1	ford torino
5	15.0	8	429.0	198.0	4341	10.0	70	1	ford galaxie 500
6	14.0	8	454.0	220.0	4354	9.0	70	1	chevrolet impala
7	14.0	8	440.0	215.0	4312	8.5	70	1	plymouth fury iii
8	14.0	8	455.0	225.0	4425	10.0	70	1	pontiac catalina
9	15.0	8	390.0	190.0	3850	8.5	70	1	amc ambassador dpl

The **.tail()** method, by default, returns the last 5 rows in the DataFrame.

[10]:
```
mpg_df.tail()
```

[10]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
401	27.0	4	140.0	86.0	2790	15.6	82	1	ford mustang gl
402	44.0	4	97.0	52.0	2130	24.6	82	2	vw pickup
403	32.0	4	135.0	84.0	2295	11.6	82	1	dodge rampage
404	28.0	4	120.0	79.0	2625	18.6	82	1	ford ranger
405	31.0	4	119.0	82.0	2720	19.4	82	1	chevy s-10

Similarly, providing an integer to the **.tail()** method provides the X last rows.

[11]:
```
mpg_df.tail(10)
```

[11]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
396	26.0	4	156.0	92.0	2585	14.5	82	1	chrysler lebaron medallion
397	22.0	6	232.0	112.0	2835	14.7	82	1	ford granada l
398	32.0	4	144.0	96.0	2665	13.9	82	3	toyota celica gt
399	36.0	4	135.0	84.0	2370	13.0	82	1	dodge charger 2.2
400	27.0	4	151.0	90.0	2950	17.3	82	1	chevrolet camaro
401	27.0	4	140.0	86.0	2790	15.6	82	1	ford mustang gl
402	44.0	4	97.0	52.0	2130	24.6	82	2	vw pickup
403	32.0	4	135.0	84.0	2295	11.6	82	1	dodge rampage
404	28.0	4	120.0	79.0	2625	18.6	82	1	ford ranger
405	31.0	4	119.0	82.0	2720	19.4	82	1	chevy s-10

The **help()** function provides information on the function / method.

[12]:
```
help(pd.DataFrame.head)
```

```
Help on function head in module pandas.core.generic:
```

```
head(self: ~FrameOrSeries, n: int = 5) -> ~FrameOrSeries
    Return the first `n` rows.

    This function returns the first `n` rows for the object based
    on position. It is useful for quickly testing if your object
    has the right type of data in it.

    For negative values of `n`, this function returns all rows except
    the last `n` rows, equivalent to ``df[:-n]``.

    Parameters
    ----------
    n : int, default 5
        Number of rows to select.

    Returns
    -------
    same type as caller
        The first `n` rows of the caller object.

    See Also
    --------
    DataFrame.tail: Returns the last `n` rows.

    Examples
    --------
    >>> df = pd.DataFrame({'animal': ['alligator', 'bee', 'falcon', 'lion',
    …                     'monkey', 'parrot', 'shark', 'whale', 'zebra']})
    >>> df
          animal
    0  alligator
    1        bee
    2     falcon
    3       lion
    4     monkey
    5     parrot
    6      shark
    7      whale
    8      zebra

    Viewing the first 5 lines

    >>> df.head()
          animal
    0  alligator
    1        bee
    2     falcon
```

```
3       lion
4     monkey

Viewing the first `n` lines (three in this case)

>>> df.head(3)
      animal
0  alligator
1        bee
2     falcon

For negative values of `n`

>>> df.head(-3)
      animal
0  alligator
1        bee
2     falcon
3       lion
4     monkey
5     parrot
```

The **.info()** method provides basic information on the DataFrame such as: - # of rows - # of columns - column names - how data is stored - # of missing values

[13]:
```
mpg_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 406 entries, 0 to 405
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     406 non-null    int64
 2   displacement  406 non-null    float64
 3   horsepower    400 non-null    float64
 4   weight        406 non-null    int64
 5   acceleration  406 non-null    float64
 6   model_year    406 non-null    int64
 7   origin        406 non-null    int64
 8   car_name      406 non-null    object
dtypes: float64(4), int64(4), object(1)
memory usage: 28.7+ KB
```

The **.rename()** method allows you to rename the index or column names.

[14]:
```
mpg_df.rename(columns={'mpg': 'miles_per_gallon', 'horsepower': 'hp'})
```

[14]:
```
     miles_per_gallon  cylinders  displacement     hp  weight  acceleration  \
0                18.0          8         307.0  130.0    3504          12.0
1                15.0          8         350.0  165.0    3693          11.5
2                18.0          8         318.0  150.0    3436          11.0
3                16.0          8         304.0  150.0    3433          12.0
4                17.0          8         302.0  140.0    3449          10.5
..                ...        ...           ...    ...     ...           ...
401              27.0          4         140.0   86.0    2790          15.6
402              44.0          4          97.0   52.0    2130          24.6
403              32.0          4         135.0   84.0    2295          11.6
404              28.0          4         120.0   79.0    2625          18.6
405              31.0          4         119.0   82.0    2720          19.4

     model_year  origin                  car_name
0            70       1  chevrolet chevelle malibu
1            70       1          buick skylark 320
2            70       1         plymouth satellite
3            70       1              amc rebel sst
4            70       1                ford torino
..          ...     ...                       ...
401          82       1            ford mustang gl
402          82       2                  vw pickup
403          82       1              dodge rampage
404          82       1                ford ranger
405          82       1                 chevy s-10

[406 rows x 9 columns]
```

4.1 Accessing Columns

You can access a column in the DataFrame using **bracket** notation or **dot** notation. The recommended approach is to use **bracket** notation.

Bracket Notation

[15]:
```
mpg_df['cylinders']
```

[15]:
```
0      8
1      8
2      8
3      8
4      8
      ..
401    4
402    4
403    4
404    4
405    4
```

```
Name: cylinders, Length: 406, dtype: int64
```

Dot Notation

[16]:
```
mpg_df.cylinders
```

[16]:
```
0      8
1      8
2      8
3      8
4      8
      ..
401    4
402    4
403    4
404    4
405    4
Name: cylinders, Length: 406, dtype: int64
```

Individual columns in a DataFrame are referred to as Series.

[17]:
```
type(mpg_df['cylinders'])
```

[17]:
```
pandas.core.series.Series
```

[18]:
```
type(mpg_df)
```

[18]:
```
pandas.core.frame.DataFrame
```

You can only use **bracket** notation to access several columns. Note that this syntax uses double square bracket; additionally, it returns back a DataFrame instead of a Series.

[19]:
```
mpg_df[['mpg', 'cylinders', 'displacement']]
```

[19]:
```
      mpg  cylinders  displacement
0    18.0          8         307.0
1    15.0          8         350.0
2    18.0          8         318.0
3    16.0          8         304.0
4    17.0          8         302.0
..    …          …           …
401  27.0          4         140.0
402  44.0          4          97.0
403  32.0          4         135.0
404  28.0          4         120.0
405  31.0          4         119.0

[406 rows x 3 columns]
```

4.2 Descriptive Statistics

The **.value_counts()** method provides the counts for each unique value in the column. The output will be in descending order with largest count first.

[20]:
```
mpg_df['cylinders'].value_counts()
```

[20]:
```
4    207
8    108
6     84
3      4
5      3
Name: cylinders, dtype: int64
```

When the argument **normalize=True** is provided to **.value_counts()** method, the output will be in percentages.

Note: **True** (along with its counterpart **False**) are reserved keywords in Python.

[21]:
```
mpg_df['cylinders'].value_counts(normalize=True)
```

[21]:
```
4    0.509852
8    0.266010
6    0.206897
3    0.009852
5    0.007389
Name: cylinders, dtype: float64
```

The **.describe()** method provides basic summary statistics including: - count - mean - min - max - std dev - 25th, 50th, 75th percentiles

[22]:
```
mpg_df.describe()
```

[22]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin
count	398.000000	406.000000	406.000000	400.000000	406.000000	406.000000	406.000000	406.000000
mean	23.514573	5.475369	194.779557	105.082500	2979.413793	15.519704	75.921182	1.568966
std	7.815984	1.712160	104.922458	38.768779	847.004328	2.803359	3.748737	0.797479
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000	1.000000
25%	17.500000	4.000000	105.000000	75.750000	2226.500000	13.700000	73.000000	1.000000
50%	23.000000	4.000000	151.000000	95.000000	2822.500000	15.500000	76.000000	1.000000
75%	29.000000	8.000000	302.000000	130.000000	3618.250000			
max	46.600000	8.000000	455.000000	230.000000	5140.000000			

```
75%       17.175000   79.000000    2.000000
max       24.800000   82.000000    3.000000
```

Providing a sequence of decimals to the **.describe()** method provides the specified percentiles and overrides the default percentiles.

[23]:
```
mpg_df.describe([.1, .2, .3, .4, .5, .6, .7, .8, .9, 1])
```

[23]:
```
                mpg   cylinders  displacement  horsepower       weight  \
count   398.000000  406.000000    406.000000  400.000000   406.000000
mean     23.514573    5.475369    194.779557  105.082500  2979.413793
std       7.815984    1.712160    104.922458   38.768779   847.004328
min       9.000000    3.000000     68.000000   46.000000  1613.000000
10%      14.000000    4.000000     90.000000   67.000000  1987.500000
20%      16.000000    4.000000     98.000000   72.000000  2155.000000
30%      18.000000    4.000000    112.000000   80.700000  2315.000000
40%      20.000000    4.000000    122.000000   88.000000  2587.000000
50%      23.000000    4.000000    151.000000   95.000000  2822.500000
60%      25.000000    6.000000    225.000000  100.000000  3102.000000
70%      27.490000    6.000000    250.000000  112.000000  3427.500000
80%      31.000000    8.000000    305.000000  142.600000  3830.000000
90%      34.330000    8.000000    350.000000  160.500000  4265.500000
100%     46.600000    8.000000    455.000000  230.000000  5140.000000
max      46.600000    8.000000    455.000000  230.000000  5140.000000

        acceleration  model_year      origin
count     406.000000  406.000000  406.000000
mean       15.519704   75.921182    1.568966
std         2.803359    3.748737    0.797479
min         8.000000   70.000000    1.000000
10%        12.000000   71.000000    1.000000
20%        13.200000   72.000000    1.000000
30%        14.000000   73.000000    1.000000
40%        14.800000   75.000000    1.000000
50%        15.500000   76.000000    1.000000
60%        16.000000   77.000000    1.000000
70%        16.800000   78.000000    2.000000
80%        17.700000   80.000000    2.000000
90%        19.000000   81.000000    3.000000
100%       24.800000   82.000000    3.000000
max        24.800000   82.000000    3.000000
```

Get the column mean with the **.mean()** method.

[24]:
```
mpg_df['mpg'].mean()
```

[24]:
```
23.514572864321615
```

Get the column min with the **.min()** method.

[25]:
```
mpg_df['mpg'].min()
```

[25]:
```
9.0
```

Get the column max with the **.min()** method.

[26]:
```
mpg_df['mpg'].max()
```

[26]:
```
46.6
```

Get the column std dev with the **.std()** method.

[27]:
```
mpg_df['mpg'].std()
```

[27]:
```
7.815984312565782
```

Get the column quantile with the **.quantile()** method.

[28]:
```
mpg_df['mpg'].quantile(.3)
```

[28]:
```
18.0
```

Get the column total with the **.sum()** method.

[29]:
```
mpg_df['weight'].sum()
```

[29]:
```
1209642
```

Some methods can be applied to multiple columns at once.

[30]:
```
mpg_df[['mpg', 'displacement', 'horsepower']].min()
```

[30]:
```
mpg             9.0
displacement    68.0
horsepower      46.0
dtype: float64
```

Additionally, they can be applied to the entire DataFrame.

[31]:
```
mpg_df.min()
```

[31]:
```
mpg                                 9
cylinders                           3
displacement                       68
horsepower                         46
weight                           1613
acceleration                        8
model_year                         70
origin                              1
car_name        amc ambassador brougham
dtype: object
```

Some methods can also be applied row-wise, by using the **axis=1** argument. The example below calculates the row-wise minimum of mpg, displacement, and horsepower columns.

[32]:
```
mpg_df[['mpg', 'displacement', 'horsepower']].min(axis=1)
```

[32]:
```
0      18.0
1      15.0
2      18.0
3      16.0
4      17.0
       …
401    27.0
402    44.0
403    32.0
404    28.0
405    31.0
Length: 406, dtype: float64
```

The **.corr()** method provides the correlation matrix as a DataFrame.

[33]:
```
mpg_df.corr()
```

[33]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin
mpg	1.000000	-0.775396	-0.804203	-0.778427	-0.831741	0.420289	0.579267	0.563450
cylinders	-0.775396	1.000000	0.951787	0.844158	0.895220	-0.522452	-0.360762	-0.567478
displacement	-0.804203	0.951787	1.000000	0.898326	0.932475	-0.557984	-0.381714	-0.613056
horsepower	-0.778427	0.844158	0.898326	1.000000	0.866586	-0.697124	-0.424419	-0.460033
weight	-0.831741	0.895220	0.932475	0.866586	1.000000	-0.430086	-0.315389	-0.584109
acceleration	0.420289	-0.522452	-0.557984	-0.697124	-0.430086	1.000000	0.301992	0.218845
model_year	0.579267	-0.360762	-0.381714	-0.424419	-0.315389	0.301992	1.000000	0.187656
origin	0.563450	-0.567478	-0.613056	-0.460033	-0.584109	0.218845	0.187656	1.000000

4.3 Sorting

Sorting is done using the **.sort_values()** method.

[34]:
```
mpg_df.head(10)
```

[34]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
0	18.0	8	307.0	130.0	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	1	ford torino
5	15.0	8	429.0	198.0	4341	10.0	70	1	ford galaxie 500
6	14.0	8	454.0	220.0	4354	9.0	70	1	chevrolet impala
7	14.0	8	440.0	215.0	4312	8.5	70	1	plymouth fury iii
8	14.0	8	455.0	225.0	4425	10.0	70	1	pontiac catalina
9	15.0	8	390.0	190.0	3850	8.5	70	1	amc ambassador dpl

Sort the DataFrame by the displacement column. By default, sorting is done in ascending order.

[35]:
```
displacement_ascending_df = mpg_df.sort_values(['displacement'])
displacement_ascending_df.head(10)
```

[35]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
124	29.0	4	68.0	49.0	1867	19.5	73	2	fiat 128
341	23.7	3	70.0	100.0	2420	12.5	80	3	mazda rx-7 gs
78	19.0	3	70.0	97.0	2330	13.5	72	3	mazda rx2 coupe
118	18.0	3	70.0	90.0	2124	13.5	73	3	maxda rx3
60	31.0	4	71.0	65.0	1773	19.0	71	3	toyota corolla 1200
138	32.0	4	71.0	65.0	1836	21.0	74	3	toyota corolla 1200
61	35.0	4	72.0	69.0	1613	18.0	71	3	
151	31.0	4	76.0	52.0	1649	16.5	74	3	
253	32.8	4	78.0	52.0	1985	19.4	78	3	
350	39.1	4	79.0	58.0	1755	16.9	81	3	

```
61           71       3          datsun 1200
151          74       3        toyota corona
253          78       3     mazda glc deluxe
350          81       3       toyota starlet
```

Sort the DataFrame by the displacement column in descending order, by using the ascending argument.

[36]:
```
displacement_descending_df = mpg_df.sort_values(['displacement'],␣
↪ascending=[False])
displacement_descending_df.head(10)
```

[36]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  \
19   14.0          8         455.0       225.0    3086          10.0
8    14.0          8         455.0       225.0    4425          10.0
102  12.0          8         455.0       225.0    4951          11.0
6    14.0          8         454.0       220.0    4354           9.0
101  13.0          8         440.0       215.0    4735          11.0
7    14.0          8         440.0       215.0    4312           8.5
74   11.0          8         429.0       208.0    4633          11.0
97   12.0          8         429.0       198.0    4952          11.5
5    15.0          8         429.0       198.0    4341          10.0
98   13.0          8         400.0       150.0    4464          12.0

     model_year  origin                       car_name
19           70       1      buick estate wagon (sw)
8            70       1              pontiac catalina
102          73       1     buick electra 225 custom
6            70       1               chevrolet impala
101          73       1  chrysler new yorker brougham
7            70       1             plymouth fury iii
74           72       1               mercury marquis
97           73       1     mercury marquis brougham
5            70       1               ford galaxie 500
98           73       1    chevrolet caprice classic
```

Sorting by multiple columns.

[37]:
```
multi_sort_df = mpg_df.sort_values(['displacement', 'mpg'], ascending=[False,␣
↪True])
multi_sort_df.head(10)
```

[37]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  \
102  12.0          8         455.0       225.0    4951          11.0
8    14.0          8         455.0       225.0    4425          10.0
19   14.0          8         455.0       225.0    3086          10.0
6    14.0          8         454.0       220.0    4354           9.0
101  13.0          8         440.0       215.0    4735          11.0
```

```
7    14.0          8         440.0       215.0    4312          8.5
74   11.0          8         429.0       208.0    4633         11.0
97   12.0          8         429.0       198.0    4952         11.5
5    15.0          8         429.0       198.0    4341         10.0
110  11.0          8         400.0       150.0    4997         14.0

     model_year  origin                       car_name
102          73       1      buick electra 225 custom
8            70       1               pontiac catalina
19           70       1        buick estate wagon (sw)
6            70       1                chevrolet impala
101          73       1  chrysler new yorker brougham
7            70       1              plymouth fury iii
74           72       1                mercury marquis
97           73       1      mercury marquis brougham
5            70       1                ford galaxie 500
110          73       1                chevrolet impala
```

4.4 Comparison Operators

- >= : greater than or equal to
- > : greater than
- <= : less than or equal to
- < : less than
- == : equals
- != : not equals

[38]:
```
mpg_df['cylinders'] == 8
```

[38]:
```
0       True
1       True
2       True
3       True
4       True
       …
401    False
402    False
403    False
404    False
405    False
Name: cylinders, Length: 406, dtype: bool
```

[39]:
```
(mpg_df['cylinders'] != 8).sum()
```

[39]:
```
298
```

[40]:
```
mpg_df['mpg'] >= mpg_df['acceleration']
```

[40]:
```
0      True
1      True
2      True
3      True
4      True
       …
401    True
402    True
403    True
404    True
405    True
Length: 406, dtype: bool
```

[41]:
```
(mpg_df['mpg'] >= mpg_df['acceleration']) & (mpg_df['cylinders'] >= 4)
```

[41]:
```
0      True
1      True
2      True
3      True
4      True
       …
401    True
402    True
403    True
404    True
405    True
Length: 406, dtype: bool
```

[42]:
```
(mpg_df['mpg'] >= mpg_df['acceleration']) | (mpg_df['cylinders'] >= 4)
```

[42]:
```
0      True
1      True
2      True
3      True
4      True
       …
401    True
402    True
403    True
404    True
405    True
Length: 406, dtype: bool
```

[43]:
```
mpg_df['cylinders'].isin([4, 8])
```

[43]:
```
0      True
1      True
2      True
```

```
3      True
4      True
       …
401    True
402    True
403    True
404    True
405    True
Name: cylinders, Length: 406, dtype: bool
```

[44]:
```
~mpg_df['cylinders'].isin([4, 8])
```

[44]:
```
0      False
1      False
2      False
3      False
4      False
       …
401    False
402    False
403    False
404    False
405    False
Name: cylinders, Length: 406, dtype: bool
```

4.5 Dealing With Missing Values

[45]:
```
mpg_df['mpg'].isna()
```

[45]:
```
0      False
1      False
2      False
3      False
4      False
       …
401    False
402    False
403    False
404    False
405    False
Name: mpg, Length: 406, dtype: bool
```

[46]:
```
mpg_df['mpg'].isna().sum()
```

[46]:
```
8
```

[47]:
```
mpg_df['horsepower'].isna().sum()
```

[47]:
```
6
```

[48]:
```
mpg_df['mpg'].isna().sum()
```

[48]:
```
8
```

[49]:
```
mpg_df.isna()
```

[49]:
```
       mpg  cylinders  displacement  horsepower  weight  acceleration  \
0    False      False         False       False   False         False
1    False      False         False       False   False         False
2    False      False         False       False   False         False
3    False      False         False       False   False         False
4    False      False         False       False   False         False
..     …          …             …           …       …             …
401  False      False         False       False   False         False
402  False      False         False       False   False         False
403  False      False         False       False   False         False
404  False      False         False       False   False         False
405  False      False         False       False   False         False

     model_year  origin  car_name
0         False   False     False
1         False   False     False
2         False   False     False
3         False   False     False
4         False   False     False
..          …       …         …
401       False   False     False
402       False   False     False
403       False   False     False
404       False   False     False
405       False   False     False

[406 rows x 9 columns]
```

Checking for # of blank values column wise.

[50]:
```
mpg_df.isna().sum()
```

[50]:
```
mpg             8
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model_year      0
origin          0
```

```
car_name        0
dtype: int64
```

Checking for number of missing values row-wise.

[51]:
```
mpg_df.isna().sum(axis=1)
```

[51]:
```
0      0
1      0
2      0
3      0
4      0
      ..
401    0
402    0
403    0
404    0
405    0
Length: 406, dtype: int64
```

Fill blank values with 0's.

[52]:
```
mpg_df['mpg_fill_0'] = mpg_df['mpg'].fillna(0)
```

[53]:
```
mpg_df['mpg_fill_0'].isna().sum()
```

[53]:
```
0
```

[54]:
```
mpg_df[['mpg', 'mpg_fill_0']].describe()
```

[54]:
```
              mpg  mpg_fill_0
count  398.000000  406.000000
mean    23.514573   23.051232
std      7.815984    8.401777
min      9.000000    0.000000
25%     17.500000   17.000000
50%     23.000000   22.350000
75%     29.000000   29.000000
max     46.600000   46.600000
```

Fill blank values with the average value.

[55]:
```
mpg_df['mpg_fill_mean'] = mpg_df['mpg'].fillna(mpg_df['mpg'].mean())
```

.fillna() method also allows you to fill forward / backward.

[56]:
```
mpg_df.iloc[list(range(9, 18)) + [38, 39, 366, 367]]
```

[56]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  \
9    15.0          8         390.0       190.0    3850           8.5
```

```
10    NaN          4         133.0       115.0    3090          17.5
11    NaN          8         350.0       165.0    4142          11.5
12    NaN          8         351.0       153.0    4034          11.0
13    NaN          8         383.0       175.0    4166          10.5
14    NaN          8         360.0       175.0    3850          11.0
15   15.0          8         383.0       170.0    3563          10.0
16   14.0          8         340.0       160.0    3609           8.0
17    NaN          8         302.0       140.0    3353           8.0
38   25.0          4          98.0         NaN    2046          19.0
39    NaN          4          97.0        48.0    1978          20.0
366  28.1          4         141.0        80.0    3230          20.4
367   NaN          4         121.0       110.0    2800          15.4

     model_year  origin                       car_name  mpg_fill_0  \
9            70       1              amc ambassador dpl        15.0
10           70       2             citroen ds-21 pallas         0.0
11           70       1  chevrolet chevelle concours (sw)         0.0
12           70       1                 ford torino (sw)         0.0
13           70       1           plymouth satellite (sw)         0.0
14           70       1                amc rebel sst (sw)         0.0
15           70       1               dodge challenger se        15.0
16           70       1               plymouth 'cuda 340        14.0
17           70       1             ford mustang boss 302         0.0
38           71       1                         ford pinto        25.0
39           71       2      volkswagen super beetle 117         0.0
366          81       2        peugeot 505s turbo diesel        28.1
367          81       2                        saab 900s         0.0

     mpg_fill_mean
9        15.000000
10       23.514573
11       23.514573
12       23.514573
13       23.514573
14       23.514573
15       15.000000
16       14.000000
17       23.514573
38       25.000000
39       23.514573
366      28.100000
367      23.514573
```

[57]:
```
mpg_df.iloc[list(range(9, 18)) + [38, 39, 366, 367]].fillna(method='ffill')
```

[57]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  \
9    15.0          8         390.0       190.0    3850           8.5
```

```
10   15.0          4         133.0       115.0    3090          17.5
11   15.0          8         350.0       165.0    4142          11.5
12   15.0          8         351.0       153.0    4034          11.0
13   15.0          8         383.0       175.0    4166          10.5
14   15.0          8         360.0       175.0    3850          11.0
15   15.0          8         383.0       170.0    3563          10.0
16   14.0          8         340.0       160.0    3609           8.0
17   14.0          8         302.0       140.0    3353           8.0
38   25.0          4          98.0       140.0    2046          19.0
39   25.0          4          97.0        48.0    1978          20.0
366  28.1          4         141.0        80.0    3230          20.4
367  28.1          4         121.0       110.0    2800          15.4

     model_year  origin                       car_name  mpg_fill_0  \
9            70       1              amc ambassador dpl        15.0
10           70       2             citroen ds-21 pallas         0.0
11           70       1  chevrolet chevelle concours (sw)         0.0
12           70       1                 ford torino (sw)         0.0
13           70       1           plymouth satellite (sw)         0.0
14           70       1                amc rebel sst (sw)         0.0
15           70       1               dodge challenger se        15.0
16           70       1               plymouth 'cuda 340        14.0
17           70       1             ford mustang boss 302         0.0
38           71       1                         ford pinto        25.0
39           71       2      volkswagen super beetle 117         0.0
366          81       2        peugeot 505s turbo diesel        28.1
367          81       2                        saab 900s         0.0

     mpg_fill_mean
9        15.000000
10       23.514573
11       23.514573
12       23.514573
13       23.514573
14       23.514573
15       15.000000
16       14.000000
17       23.514573
38       25.000000
39       23.514573
366      28.100000
367      23.514573
```

[58]:
```
mpg_df.iloc[list(range(9, 18)) + [38, 39, 366, 367, 368]].fillna(method='bfill')
```

[58]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  \
9    15.0          8         390.0       190.0    3850           8.5
```

```
10   15.0          4         133.0       115.0    3090          17.5
11   15.0          8         350.0       165.0    4142          11.5
12   15.0          8         351.0       153.0    4034          11.0
13   15.0          8         383.0       175.0    4166          10.5
14   15.0          8         360.0       175.0    3850          11.0
15   15.0          8         383.0       170.0    3563          10.0
16   14.0          8         340.0       160.0    3609           8.0
17   25.0          8         302.0       140.0    3353           8.0
38   25.0          4          98.0        48.0    2046          19.0
39   28.1          4          97.0        48.0    1978          20.0
366  28.1          4         141.0        80.0    3230          20.4
367  30.7          4         121.0       110.0    2800          15.4
368  30.7          6         145.0        76.0    3160          19.6

     model_year  origin                       car_name  mpg_fill_0  \
9            70       1              amc ambassador dpl        15.0
10           70       2            citroen ds-21 pallas         0.0
11           70       1  chevrolet chevelle concours (sw)         0.0
12           70       1                ford torino (sw)         0.0
13           70       1          plymouth satellite (sw)         0.0
14           70       1               amc rebel sst (sw)         0.0
15           70       1             dodge challenger se        15.0
16           70       1              plymouth 'cuda 340        14.0
17           70       1           ford mustang boss 302         0.0
38           71       1                       ford pinto        25.0
39           71       2     volkswagen super beetle 117         0.0
366          81       2       peugeot 505s turbo diesel        28.1
367          81       2                       saab 900s         0.0
368          81       2                    volvo diesel        30.7

     mpg_fill_mean
9        15.000000
10       23.514573
11       23.514573
12       23.514573
13       23.514573
14       23.514573
15       15.000000
16       14.000000
17       23.514573
38       25.000000
39       23.514573
366      28.100000
367      23.514573
368      30.700000
```

The **.dropna()** method, by default, drops rows with any missing values.

[59]:
```
mpg_dropna_df = mpg_df.dropna()
```

[60]:
```
len(mpg_dropna_df)
```

[60]:
```
392
```

[61]:
```
len(mpg_df)
```

[61]:
```
406
```

[62]:
```
mpg_dropna_hp_df = mpg_df.dropna(subset=['horsepower'])
```

[63]:
```
len(mpg_dropna_hp_df)
```

[63]:
```
400
```

4.6 Dealing With Duplicates

The **.unique()** method returns a pd.Series of unique values for the specified column.

[64]:
```
mpg_df['cylinders'].unique()
```

[64]:
```
array([8, 4, 6, 3, 5], dtype=int64)
```

The **.nunique()** method returns the number of unique values in the specified column.

[65]:
```
mpg_df['cylinders'].nunique()
```

[65]:
```
5
```

The **duplicated()** method returns a pd.Series of boolean values denoting of the value is a duplicate. By default, the first value is not considered a duplicate.

[66]:
```
mpg_df['cylinders']
```

[66]:
```
0      8
1      8
2      8
3      8
4      8
      ..
401    4
402    4
403    4
404    4
405    4
Name: cylinders, Length: 406, dtype: int64
```

[67]:
```
mpg_df['cylinders'].duplicated()
```

[67]:
```
0      False
1       True
2       True
3       True
4       True
       …
401     True
402     True
403     True
404     True
405     True
Name: cylinders, Length: 406, dtype: bool
```

[68]:
```
mpg_df['cylinders'].duplicated().sum()
```

[68]:
```
401
```

The **.drop_duplicates()** method removes rows that are duplicated. By default, it removes rows that are exact duplicates. Additionally, by default, the first occurrence is not considered a duplicate and is retained.

[69]:
```
mpg_df_drop_dup = mpg_df.drop_duplicates()
```

[70]:
```
len(mpg_df_drop_dup) == len(mpg_df)
```

[70]:
```
True
```

You can specify which columns to check for duplicates and remove them.

[71]:
```
mpg_df_drop_dup_cyl = mpg_df.drop_duplicates('cylinders')
mpg_df_drop_dup_cyl
```

[71]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	\
0	18.0	8	307.0	130.0	3504	12.0	
10	NaN	4	133.0	115.0	3090	17.5	
21	22.0	6	198.0	95.0	2833	15.5	
78	19.0	3	70.0	97.0	2330	13.5	
281	20.3	5	131.0	103.0	2830	15.9	

	model_year	origin	car_name	mpg_fill_0	mpg_fill_mean
0	70	1	chevrolet chevelle malibu	18.0	18.000000
10	70	2	citroen ds-21 pallas	0.0	23.514573
21	70	1	plymouth duster	22.0	22.000000
78	72	3	mazda rx2 coupe	19.0	19.000000
281	78	2	audi 5000	20.3	20.300000

[72]:
```
len(mpg_df_drop_dup_cyl) == len(mpg_df)
```

[72]:
```
False
```

[73]:
```
mpg_df_drop_dup_cyl_origin = mpg_df.drop_duplicates(['cylinders', 'origin'])
```

[74]:
```
mpg_df_drop_dup_cyl_origin
```

[74]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	\
0	18.0	8	307.0	130.0	3504	12.0	
10	NaN	4	133.0	115.0	3090	17.5	
20	24.0	4	113.0	95.0	2372	15.0	
21	22.0	6	198.0	95.0	2833	15.5	
36	28.0	4	140.0	90.0	2264	15.5	
78	19.0	3	70.0	97.0	2330	13.5	
130	20.0	6	156.0	122.0	2807	13.5	
218	16.5	6	168.0	120.0	3820	16.7	
281	20.3	5	131.0	103.0	2830	15.9	

	model_year	origin	car_name	mpg_fill_0	mpg_fill_mean
0	70	1	chevrolet chevelle malibu	18.0	18.000000
10	70	2	citroen ds-21 pallas	0.0	23.514573
20	70	3	toyota corona mark ii	24.0	24.000000
21	70	1	plymouth duster	22.0	22.000000
36	71	1	chevrolet vega 2300	28.0	28.000000
78	72	3	mazda rx2 coupe	19.0	19.000000
130	73	3	toyota mark ii	20.0	20.000000
218	76	2	mercedes-benz 280s	16.5	16.500000
281	78	2	audi 5000	20.3	20.300000

[75]:
```
len(mpg_df_drop_dup_cyl_origin) == len(mpg_df)
```

[75]:
```
False
```

4.7 Math Operations with DataFrames

[76]:
```
mpg_df['weight_standardized'] = (mpg_df['weight'] - mpg_df['weight'].mean()) /␣
↪mpg_df['weight'].std()
```

[77]:
```
mpg_df[['weight', 'weight_standardized']].head()
```

[77]:

	weight	weight_standardized
0	3504	0.619343
1	3693	0.842482
2	3436	0.539060
3	3433	0.535518
4	3449	0.554408

4.8 A Word About Indices

[78]:
```
df1 = pd.DataFrame({'x': [1, 2, 3]})
df1
```

[78]:
```
   x
0  1
1  2
2  3
```

[79]:
```
df2 = pd.DataFrame({'y': [1, 2, 3]})
df2.index = [2, 1, 0]
df2
```

[79]:
```
   y
2  1
1  2
0  3
```

[80]:
```
df1['x'] + df2['y']
```

[80]:
```
0    4
1    4
2    4
dtype: int64
```

4.9 Functions in Python

Functions are defined using the reserved keyword **def**. Indented code block lets Python know what included in the function definition.

[81]:
```
def subtract_none(a, b):
    a - b
```

[82]:
```
bad_sub = subtract_none(5, 1)
```

You will notice that when we call the variable bad_sub, we got nothing back; we should be expecting the value 4. The reason bad_sub returns nothing back is because our function definition did not include a return statement. Therefore, the function does the operation, but does not save the results.

[83]:
```
bad_sub
```

We'll define the function correctly this time with a return statement.

[84]:
```
def subtract(a, b):
    return a - b
```

[85]:
```
good_sub = subtract(5, 1)
```

[86]:
```
good_sub
```

[86]:
```
4
```

In Python, functions do not execute any code after the first return statement. In the following example, we have 2 return statements for our subtract_return2 function. However, when we execute this function, nothing after **return a - b** is executed.

[87]:
```
def subtract_return2(a, b):
    return a - b
    return b - a
```

[88]:
```
subtract_return2(5, 1)
```

[88]:
```
4
```

Here is a more relevant example where we create our own standardize column function. Note: The scikit-learn (sklearn) package performs standardization for you.

[89]:
```
def standardize_col(col_values):
    return (col_values - col_values.mean()) / col_values.std()
```

[90]:
```
standardize_col(mpg_df['weight'])
```

[90]:
```
0      0.619343
1      0.842482
2      0.539060
3      0.535518
4      0.554408
         …
401   -0.223628
402   -1.002845
403   -0.808040
404   -0.418432
405   -0.306272
Name: weight, Length: 406, dtype: float64
```

[91]:
```
mpg_df['weight_standardized_from_func'] = standardize_col(mpg_df['weight'])
```

[92]:
```
mpg_df['weight_standardized'] != mpg_df['weight_standardized_from_func']
```

[92]:
```
0      False
1      False
2      False
3      False
4      False
       …
401    False
402    False
```

```
403    False
404    False
405    False
Length: 406, dtype: bool
```

[93]:
```
(mpg_df['weight_standardized'] != mpg_df['weight_standardized_from_func']).sum()
```

[93]:
```
0
```

It is good practice to add **docstrings** to your functions. This allows your to call the **help()** function, which was discussed at the beginning of this tutorial.

[94]:
```
def standardize_col_docstring(col_values):
    """
    This function standardizes a given pd.Series and returns the standardized␣
↪values as another pd.Series.
    """
    return (col_values - col_values.mean()) / col_values.std()
```

[95]:
```
help(standardize_col_docstring)
```

```
Help on function standardize_col_docstring in module __main__:

standardize_col_docstring(col_values)
    This function standardizes a given pd.Series and returns the standardized
values as another pd.Series.
```

[96]:
```
help(standardize_col)
```

```
Help on function standardize_col in module __main__:

standardize_col(col_values)
```

4.10 Groupby

[97]:
```
mpg_df.groupby('cylinders')['weight'].mean()
```

[97]:
```
cylinders
3    2398.500000
4    2312.685990
5    3103.333333
6    3198.226190
8    4105.194444
Name: weight, dtype: float64
```

[98]:
```
mpg_df.groupby('cylinders')['weight'].apply(standardize_col)
```

[98]:
```
0     -1.350689
1     -0.926067
2     -1.503463
3     -1.510203
4     -1.474256
         …
401    1.358938
402   -0.520116
403   -0.050353
404    0.889174
405    1.159644
Name: weight, Length: 406, dtype: float64
```

[99]:
```
mpg_df.groupby('cylinders')['weight'].describe()
```

[99]:

cylinders	count	mean	std	min	25%	50%	75%	max
3	4.0	2398.500000	247.566153	2124.0	2278.50	2375.0	2495.00	2720.0
4	207.0	2312.685990	351.240579	1613.0	2045.50	2234.0	2573.50	3270.0
5	3.0	3103.333333	374.343870	2830.0	2890.00	2950.0	3240.00	3530.0
6	84.0	3198.226190	332.297419	2472.0	2941.25	3201.5	3430.50	3907.0
8	108.0	4105.194444	445.102182	3086.0	3810.00	4137.5	4382.75	5140.0

You can transpose DataFrames with the **.T**.

[100]:
```
mpg_df.groupby('cylinders')['weight'].describe().T
```

[100]:

cylinders	3	4	5	6	8
count	4.000000	207.000000	3.000000	84.000000	108.000000
mean	2398.500000	2312.685990	3103.333333	3198.226190	4105.194444
std	247.566153	351.240579	374.343870	332.297419	445.102182
min	2124.000000	1613.000000	2830.000000	2472.000000	3086.000000
25%	2278.500000	2045.500000	2890.000000	2941.250000	3810.000000
50%	2375.000000	2234.000000	2950.000000	3201.500000	4137.500000
75%	2495.000000	2573.500000	3240.000000	3430.500000	4382.750000
max	2720.000000	3270.000000	3530.000000	3907.000000	5140.000000

[101]:
```
mpg_df.groupby(['cylinders', 'origin']).mean()
```

[101]:
```
                          mpg  displacement  horsepower       weight  \
cylinders origin
3         3       20.550000     72.500000   99.250000  2398.500000
4         1       27.840278    124.284722   80.956522  2437.166667
          2       28.411111    104.803030   78.906250  2343.318182
          3       31.595652     99.768116   75.579710  2153.492754
5         2       27.366667    145.000000   82.333333  3103.333333
6         1       19.663514    226.283784   99.671233  3213.905405
          2       20.100000    159.750000  113.500000  3382.500000
          3       23.883333    156.666667  115.833333  2882.000000
8         1       14.963107    345.203704  158.453704  4105.194444

                  acceleration  model_year  mpg_fill_0  mpg_fill_mean  \
cylinders origin
3         3          13.250000   75.500000   20.550000      20.550000
4         1          16.526389   78.027778   27.840278      27.840278
          2          16.763636   75.439394   27.119697      28.188541
          3          16.569565   77.507246   31.595652      31.595652
5         2          18.633333   79.000000   27.366667      27.366667
6         1          16.474324   75.635135   19.663514      19.663514
          2          16.425000   78.250000   20.100000      20.100000
          3          13.550000   78.000000   23.883333      23.883333
8         1          12.837037   73.722222   14.270370      15.359008

                  weight_standardized  weight_standardized_from_func
cylinders origin
3         3                -0.685845                      -0.685845
4         1                -0.640194                      -0.640194
          2                -0.750995                      -0.750995
          3                -0.975108                      -0.975108
5         2                 0.146303                       0.146303
6         1                 0.276848                       0.276848
          2                 0.475896                       0.475896
          3                -0.115010                      -0.115010
8         1                 1.329132                       1.329132
```

4.11 Merging

Suppose we have a separate DataFrame that contains the grouped weight averages.

[102]:
```
mpg_grpby = mpg_df.groupby(['cylinders', 'origin'])['weight'].mean().
↪reset_index()
```

[103]:
```
mpg_grpby = mpg_grpby.rename(columns={'weight': 'grpby_avg_weight'})
```

[104]:
```
mpg_grpby
```

[104]:

	cylinders	origin	grpby_avg_weight
0	3	3	2398.500000
1	4	1	2437.166667
2	4	2	2343.318182
3	4	3	2153.492754
4	5	2	3103.333333
5	6	1	3213.905405
6	6	2	3382.500000
7	6	3	2882.000000
8	8	1	4105.194444

We would like to merge this DataFrame with our original DataFrame. This can be done using the **pd.merge()** function. By default, this function performs an inner join. You can pass in other arguments to change how to perform the merging.

[105]:
```
mpg_merged = pd.merge(mpg_df, mpg_grpby, on=['cylinders', 'origin'])
mpg_merged.columns
```

[105]:
```
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model_year', 'origin', 'car_name', 'mpg_fill_0',
       'mpg_fill_mean', 'weight_standardized', 'weight_standardized_from_func',
       'grpby_avg_weight'],
      dtype='object')
```

[106]:
```
mpg_merged.head()
```

[106]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	\
0	18.0	8	307.0	130.0	3504	12.0	
1	15.0	8	350.0	165.0	3693	11.5	
2	18.0	8	318.0	150.0	3436	11.0	
3	16.0	8	304.0	150.0	3433	12.0	
4	17.0	8	302.0	140.0	3449	10.5	

	model_year	origin	car_name	mpg_fill_0	mpg_fill_mean	\
0	70	1	chevrolet chevelle malibu	18.0	18.0	
1	70	1	buick skylark 320	15.0	15.0	
2	70	1	plymouth satellite	18.0	18.0	
3	70	1	amc rebel sst	16.0	16.0	
4	70	1	ford torino	17.0	17.0	

	weight_standardized	weight_standardized_from_func	grpby_avg_weight
0	0.619343	0.619343	4105.194444
1	0.842482	0.842482	4105.194444
2	0.539060	0.539060	4105.194444
3	0.535518	0.535518	4105.194444
4	0.554408	0.554408	4105.194444

[107]:
```
mpg_merged.tail()
```

[107]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  \
401  16.2          6         163.0       133.0    3410          15.8
402  30.7          6         145.0        76.0    3160          19.6
403  20.3          5         131.0       103.0    2830          15.9
404  25.4          5         183.0        77.0    3530          20.1
405  36.4          5         121.0        67.0    2950          19.9

     model_year  origin            car_name  mpg_fill_0  mpg_fill_mean  \
401          78       2       peugeot 604sl        16.2           16.2
402          81       2        volvo diesel        30.7           30.7
403          78       2           audi 5000        20.3           20.3
404          79       2  mercedes benz 300d        25.4           25.4
405          80       2  audi 5000s (diesel)        36.4           36.4

     weight_standardized  weight_standardized_from_func  grpby_avg_weight
401             0.508364                       0.508364       3382.500000
402             0.213206                       0.213206       3382.500000
403            -0.176403                      -0.176403       3103.333333
404             0.650039                       0.650039       3103.333333
405            -0.034727                      -0.034727       3103.333333
```

4.12 One-Hot Encoding

[108]:
```
mpg_df[['origin_1', 'origin_2', 'origin_3']] = pd.get_dummies(mpg_df['origin'])
```

[109]:
```
mpg_df[['origin', 'origin_1', 'origin_2', 'origin_3']]
```

[109]:
```
     origin  origin_1  origin_2  origin_3
0         1         1         0         0
1         1         1         0         0
2         1         1         0         0
3         1         1         0         0
4         1         1         0         0
..      …         …         …         …
401       1         1         0         0
402       2         0         1         0
403       1         1         0         0
404       1         1         0         0
405       1         1         0         0

[406 rows x 4 columns]
```

4.13 Mapping

[110]:
```
mpg_df['origin_str'] = mpg_df['origin'].map({1: 'US', 2: 'Europe', 3: 'Japan'})
```

[111]:
```
mpg_df[['origin', 'origin_str']]
```

[111]:
```
     origin origin_str
0         1         US
1         1         US
2         1         US
3         1         US
4         1         US
..      …         …
401       1         US
402       2     Europe
403       1         US
404       1         US
405       1         US

[406 rows x 2 columns]
```

4.14 Subsetting Data

[112]:
```
mpg_df[(mpg_df['cylinders'] > 4) & (mpg_df['mpg'] > 15)]
```

[112]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  \
0    18.0          8         307.0       130.0    3504          12.0
2    18.0          8         318.0       150.0    3436          11.0
3    16.0          8         304.0       150.0    3433          12.0
4    17.0          8         302.0       140.0    3449          10.5
21   22.0          6         198.0        95.0    2833          15.5
..    …          …           …          …      …           …
373  20.2          6         200.0        88.0    3060          17.1
374  17.6          6         225.0        85.0    3465          16.6
394  25.0          6         181.0       110.0    2945          16.4
395  38.0          6         262.0        85.0    3015          17.0
397  22.0          6         232.0       112.0    2835          14.7

     model_year  origin                         car_name  mpg_fill_0  \
0            70       1          chevrolet chevelle malibu        18.0
2            70       1                plymouth satellite        18.0
3            70       1                     amc rebel sst        16.0
4            70       1                       ford torino        17.0
21           70       1                  plymouth duster        22.0
..          …     …                                 …         …
373          81       1                  ford granada gl        20.2
374          81       1             chrysler lebaron salon        17.6
394          82       1              buick century limited        25.0
395          82       1  oldsmobile cutlass ciera (diesel)        38.0
397          82       1                  ford granada l        22.0

     mpg_fill_mean  weight_standardized  weight_standardized_from_func  \
0             18.0             0.619343                       0.619343
```

```
2              18.0              0.539060                       0.539060
3              16.0              0.535518                       0.535518
4              17.0              0.554408                       0.554408
21             22.0             -0.172861                      -0.172861
..              …                    …                             …
373            20.2              0.095143                       0.095143
374            17.6              0.573298                       0.573298
394            25.0             -0.040630                      -0.040630
395            38.0              0.042014                       0.042014
397            22.0             -0.170499                      -0.170499

     origin_1  origin_2  origin_3 origin_str
0           1         0         0         US
2           1         0         0         US
3           1         0         0         US
4           1         0         0         US
21          1         0         0         US
..        …       …       …         …
373         1         0         0         US
374         1         0         0         US
394         1         0         0         US
395         1         0         0         US
397         1         0         0         US

[121 rows x 17 columns]
```

.loc Method

[113]:
```
mpg_df.loc[(mpg_df['cylinders'] > 4) & (mpg_df['mpg'] > 15), ['mpg',␣
↪'cylinders', 'horsepower', 'origin']]
```

[113]:
```
      mpg  cylinders  horsepower  origin
0    18.0          8       130.0       1
2    18.0          8       150.0       1
3    16.0          8       150.0       1
4    17.0          8       140.0       1
21   22.0          6        95.0       1
..    …          …          …        …
373  20.2          6        88.0       1
374  17.6          6        85.0       1
394  25.0          6       110.0       1
395  38.0          6        85.0       1
397  22.0          6       112.0       1

[121 rows x 4 columns]
```

.iloc Method

[114]:
```
mpg_df.iloc[[0, 2, 3], [1, 2]]
```

[114]:
```
    cylinders  displacement
0           8         307.0
2           8         318.0
3           8         304.0
```

[115]:
```
mpg_df.iloc[2:, :-3]
```

[115]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  \
2    18.0          8         318.0       150.0    3436          11.0
3    16.0          8         304.0       150.0    3433          12.0
4    17.0          8         302.0       140.0    3449          10.5
5    15.0          8         429.0       198.0    4341          10.0
6    14.0          8         454.0       220.0    4354           9.0
..    …          …             …           …       …            …
401  27.0          4         140.0        86.0    2790          15.6
402  44.0          4          97.0        52.0    2130          24.6
403  32.0          4         135.0        84.0    2295          11.6
404  28.0          4         120.0        79.0    2625          18.6
405  31.0          4         119.0        82.0    2720          19.4

     model_year  origin            car_name  mpg_fill_0  mpg_fill_mean  \
2            70       1  plymouth satellite        18.0           18.0
3            70       1       amc rebel sst        16.0           16.0
4            70       1         ford torino        17.0           17.0
5            70       1    ford galaxie 500        15.0           15.0
6            70       1    chevrolet impala        14.0           14.0
..          …     …                 …         …            …
401          82       1     ford mustang gl        27.0           27.0
402          82       2           vw pickup        44.0           44.0
403          82       1       dodge rampage        32.0           32.0
404          82       1         ford ranger        28.0           28.0
405          82       1          chevy s-10        31.0           31.0

     weight_standardized  weight_standardized_from_func  origin_1
2               0.539060                       0.539060         1
3               0.535518                       0.535518         1
4               0.554408                       0.554408         1
5               1.607532                       1.607532         1
6               1.622880                       1.622880         1
..                   …                              …       …
401            -0.223628                      -0.223628         1
402            -1.002845                      -1.002845         0
403            -0.808040                      -0.808040         1
404            -0.418432                      -0.418432         1
405            -0.306272                      -0.306272         1

[404 rows x 14 columns]
```

4.15 Concatenating DataFrames

[116]:
```
mpg_df_cyl_4 = mpg_df[mpg_df['cylinders'] == 4]
len(mpg_df_cyl_4)
```

[116]:
```
207
```

[117]:
```
mpg_df_cyl_6 = mpg_df[mpg_df['cylinders'] == 6]
len(mpg_df_cyl_6)
```

[117]:
```
84
```

[118]:
```
mpg_df_cyl_8 = mpg_df[mpg_df['cylinders'] == 8]
len(mpg_df_cyl_8)
```

[118]:
```
108
```

[119]:
```
mpg_df_concat = pd.concat([mpg_df_cyl_4, mpg_df_cyl_6, mpg_df_cyl_8])
len(mpg_df_concat)
```

[119]:
```
399
```

[120]:
```
mpg_df_concat['cylinders'].value_counts()
```

[120]:
```
4    207
8    108
6     84
Name: cylinders, dtype: int64
```

4.16 Lagging Variables

[121]:
```
mpg_df['mpg_lag1'] = mpg_df['mpg'].shift(1)
```

[122]:
```
mpg_df['mpg_lag2'] = mpg_df['mpg'].shift(2)
```

[123]:
```
mpg_df[['mpg', 'mpg_lag1', 'mpg_lag2']].head(10)
```

[123]:

	mpg	mpg_lag1	mpg_lag2
0	18.0	NaN	NaN
1	15.0	18.0	NaN
2	18.0	15.0	18.0
3	16.0	18.0	15.0
4	17.0	16.0	18.0
5	15.0	17.0	16.0
6	14.0	15.0	17.0
7	14.0	14.0	15.0
8	14.0	14.0	14.0
9	15.0	14.0	14.0

4.17 Rolling Functions

[124]:
```
np.random.seed(1234)
stock_prices = pd.DataFrame({
    'day': range(1, 366),
    'stock': np.round(np.abs(np.random.normal(loc=1000, scale=1000, size=365)),␣
 ↪2)
})
```

[125]:
```
stock_prices.head(20)
```

[125]:

	day	stock
0	1	1471.44
1	2	190.98
2	3	2432.71
3	4	687.35
4	5	279.41
5	6	1887.16
6	7	1859.59
7	8	363.48
8	9	1015.70
9	10	1242.68
10	11	2150.04
11	12	1991.95
12	13	1953.32
13	14	1021.25
14	15	665.92
15	16	1002.12
16	17	1405.45
17	18	1289.09
18	19	2321.16
19	20	546.91

[126]:
```
stock_prices['rolling_5_sum'] = stock_prices['stock'].rolling(5).sum()
```

[127]:
```
stock_prices['rolling_5_mean'] = stock_prices['stock'].rolling(5).mean()
```

[128]:
```
stock_prices['rolling_5_std'] = stock_prices['stock'].rolling(5).std()
```

[129]:
```
stock_prices['rolling_5_min'] = stock_prices['stock'].rolling(5).min()
```

[130]:
```
stock_prices['rolling_5_max'] = stock_prices['stock'].rolling(5).max()
```

[131]:
```
stock_prices.head(15)
```

[131]:

	day	stock	rolling_5_sum	rolling_5_mean	rolling_5_std	rolling_5_min	\
0	1	1471.44	NaN	NaN	NaN	NaN	
1	2	190.98	NaN	NaN	NaN	NaN	

```
2     3  2432.71            NaN             NaN            NaN            NaN
3     4   687.35            NaN             NaN            NaN            NaN
4     5   279.41        5061.89        1012.378     941.496266         190.98
5     6  1887.16        5477.61        1095.522    1008.166886         190.98
6     7  1859.59        7146.22        1429.244     904.758765         279.41
7     8   363.48        5076.99        1015.398     797.953636         279.41
8     9  1015.70        5405.34        1081.068     777.454481         279.41
9    10  1242.68        6368.61        1273.722     635.536574         363.48
10   11  2150.04        6631.49        1326.298     705.944904         363.48
11   12  1991.95        6763.85        1352.770     732.908798         363.48
12   13  1953.32        8353.69        1670.738     506.228352        1015.70
13   14  1021.25        8359.24        1671.848     504.435897        1021.25
14   15   665.92        7782.48        1556.496     666.895390         665.92

    rolling_5_max
0             NaN
1             NaN
2             NaN
3             NaN
4         2432.71
5         2432.71
6         2432.71
7         1887.16
8         1887.16
9         1887.16
10        2150.04
11        2150.04
12        2150.04
13        2150.04
14        2150.04
```

4.18 String Methods

[132]:
```
mpg_df['car_name']
```

[132]:
```
0      chevrolet chevelle malibu
1              buick skylark 320
2             plymouth satellite
3                  amc rebel sst
4                    ford torino
                 …
401              ford mustang gl
402                    vw pickup
403                dodge rampage
404                  ford ranger
405                   chevy s-10
Name: car_name, Length: 406, dtype: object
```

Convert strings to all uppercase using **.str.upper()** method.

[133]:
```
mpg_df['car_name'].str.upper()
```

[133]:
```
0      CHEVROLET CHEVELLE MALIBU
1              BUICK SKYLARK 320
2             PLYMOUTH SATELLITE
3                  AMC REBEL SST
4                    FORD TORINO
                  …
401              FORD MUSTANG GL
402                    VW PICKUP
403                DODGE RAMPAGE
404                  FORD RANGER
405                   CHEVY S-10
Name: car_name, Length: 406, dtype: object
```

Capitalize the first letter of every word using **.str.title()** method.

[134]:
```
mpg_df['car_name'].str.title()
```

[134]:
```
0      Chevrolet Chevelle Malibu
1              Buick Skylark 320
2             Plymouth Satellite
3                  Amc Rebel Sst
4                    Ford Torino
                  …
401              Ford Mustang Gl
402                    Vw Pickup
403                Dodge Rampage
404                  Ford Ranger
405                   Chevy S-10
Name: car_name, Length: 406, dtype: object
```

Check if strings start with specified string using **.str.startswith()** method. Conversely, there is also a **.str.endswith()** method. **Note**: Python is **case-sensitive**.

[135]:
```
mpg_df['car_name'].str.startswith('chev')
```

[135]:
```
0       True
1      False
2      False
3      False
4      False
       …
401    False
402    False
403    False
404    False
```

```
405     True
Name: car_name, Length: 406, dtype: bool
```

[136]:
```
mpg_df['car_name'].str.startswith('chev').sum()
```

[136]:
```
48
```

The **.contains()** method also supports regular expressions. Note: The base Python package **re** is dedicated to regular expressions.

In this example, check if a string contains any digit 0-9.

[137]:
```
mpg_df['car_name'].str.contains('\d').sum()
```

[137]:
```
120
```

Replace characters with the **.replace()** method.

[138]:
```
mpg_df['car_name'].str.replace('c', 'T')
```

[138]:
```
0      Thevrolet Thevelle malibu
1              buiTk skylark 320
2             plymouth satellite
3                  amT rebel sst
4                    ford torino
                 …
401              ford mustang gl
402                    vw piTkup
403                dodge rampage
404                  ford ranger
405                   Thevy s-10
Name: car_name, Length: 406, dtype: object
```

[139]:
```
mpg_df['disp_as_str'] = mpg_df['displacement'].astype(str)
```

[140]:
```
mpg_df['disp_as_str']
```

[140]:
```
0      307.0
1      350.0
2      318.0
3      304.0
4      302.0
       …
401    140.0
402     97.0
403    135.0
404    120.0
405    119.0
Name: disp_as_str, Length: 406, dtype: object
```

The **.str.strip()** method removes leading and trailing characters specified by the user. There are also **.str.lstrip()** method which removes leading characters only and **.str.rstrip()** which removes trailing characters only.

[141]:
```
mpg_df['disp_as_str'].str.strip('0')
```

[141]:
```
0      307.
1      350.
2      318.
3      304.
4      302.
       …
401    140.
402     97.
403    135.
404    120.
405    119.
Name: disp_as_str, Length: 406, dtype: object
```

The **.str.zfill()** method comes in handy dealing with string columns that are usually dealing with accounts. In the example below, let's make a "pretend" account column. Let's suppose that our account column needs to have leading 0's.

[142]:
```
mpg_df['fake_acct_str'] = mpg_df['disp_as_str'].str.rstrip('.0')
mpg_df['fake_acct_str']
```

[142]:
```
0      307
1       35
2      318
3      304
4      302
      …
401     14
402     97
403    135
404     12
405    119
Name: fake_acct_str, Length: 406, dtype: object
```

In the example below, the 9 represents how long the string should be in length. Strings shorter than 9 are left padded with 0's so that the new length is 9. Nothing happens to strings with lengths >= 9.

[143]:
```
mpg_df['fake_acct_str'].str.zfill(9)
```

[143]:
```
0      000000307
1      000000035
2      000000318
3      000000304
```

```
4      000000302
         …
401    000000014
402    000000097
403    000000135
404    000000012
405    000000119
Name: fake_acct_str, Length: 406, dtype: object
```

4.19 Data Conversion

[144]:
```
mpg_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 406 entries, 0 to 405
Data columns (total 21 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   mpg                           398 non-null    float64
 1   cylinders                     406 non-null    int64
 2   displacement                  406 non-null    float64
 3   horsepower                    400 non-null    float64
 4   weight                        406 non-null    int64
 5   acceleration                  406 non-null    float64
 6   model_year                    406 non-null    int64
 7   origin                        406 non-null    int64
 8   car_name                      406 non-null    object
 9   mpg_fill_0                    406 non-null    float64
 10  mpg_fill_mean                 406 non-null    float64
 11  weight_standardized           406 non-null    float64
 12  weight_standardized_from_func 406 non-null    float64
 13  origin_1                      406 non-null    uint8
 14  origin_2                      406 non-null    uint8
 15  origin_3                      406 non-null    uint8
 16  origin_str                    406 non-null    object
 17  mpg_lag1                      397 non-null    float64
 18  mpg_lag2                      396 non-null    float64
 19  disp_as_str                   406 non-null    object
 20  fake_acct_str                 406 non-null    object
dtypes: float64(10), int64(4), object(4), uint8(3)
memory usage: 58.4+ KB
```

We see that the displacement column is stored as float. Let's convert it to integer using the **.astype()** method.

[145]:
```
mpg_df['disp_as_int'] = mpg_df['displacement'].astype(int)
```

[146]:
```
mpg_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 406 entries, 0 to 405
Data columns (total 22 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   mpg                             398 non-null    float64
 1   cylinders                       406 non-null    int64
 2   displacement                    406 non-null    float64
 3   horsepower                      400 non-null    float64
 4   weight                          406 non-null    int64
 5   acceleration                    406 non-null    float64
 6   model_year                      406 non-null    int64
 7   origin                          406 non-null    int64
 8   car_name                        406 non-null    object
 9   mpg_fill_0                      406 non-null    float64
 10  mpg_fill_mean                   406 non-null    float64
 11  weight_standardized             406 non-null    float64
 12  weight_standardized_from_func   406 non-null    float64
 13  origin_1                        406 non-null    uint8
 14  origin_2                        406 non-null    uint8
 15  origin_3                        406 non-null    uint8
 16  origin_str                      406 non-null    object
 17  mpg_lag1                        397 non-null    float64
 18  mpg_lag2                        396 non-null    float64
 19  disp_as_str                     406 non-null    object
 20  fake_acct_str                   406 non-null    object
 21  disp_as_int                     406 non-null    int32
dtypes: float64(10), int32(1), int64(4), object(4), uint8(3)
memory usage: 60.0+ KB
```

Note: You can also call the **.dtype** attribute to check how the column is stored. Notice that there are no parentheses after **.dtype**; this is because we are accessing the attribute and not calling a method.

[147]:
```
mpg_df['displacement'].dtype
```

[147]:
```
dtype('float64')
```

[148]:
```
mpg_df['disp_as_int'].dtype
```

[148]:
```
dtype('int32')
```

[149]:
```
mpg_df['disp_as_str'] = mpg_df['displacement'].astype(str)
```

[150]:
```
mpg_df['disp_as_str']
```

[150]:
```
0      307.0
1      350.0
```

```
2      318.0
3      304.0
4      302.0
       …
401    140.0
402     97.0
403    135.0
404    120.0
405    119.0
Name: disp_as_str, Length: 406, dtype: object
```

4.20 Exporting DataFrames

Just as pandas has many methods for reading in files, it also has several methods to export DataFrames.

[ ]:
```
mpg_df.to_csv(f'c:/users/{os.getlogin()}/mpg_df_csv.csv')
```

[ ]:
```
mpg_df.to_excel(f'c:/users/{os.getlogin()/mpg_df_xl.xlsx', index=False})
```