Annotated follow-along guide_ Data structures in Python

May 30, 2023

1 Annotated follow-along guide: Data structures in Python

This notebook contains the code used in the instructional videos from Week 4: Data structures in Python.

As a reminder, an in-video message will appear to advise that the video you are viewing contains coding instruction and examples. This follow-along notebook has different sections for each video included in the week's content. The in-video message will direct you to the relevant section in the notebook for the specific video you are viewing.

To skip directly to the code for a particular video, use the following links:

1. Introduction to lists

[1]:
```
# Assign a list using brackets, with elements separated by commas
x = ["Now", "we", "are", "cooking", "with", 7, "ingredients"]

# Print element at index 3
print(x[3])
```

```
cooking
```

[5]:
```
# Trying to access an index not in list will result in IndexError
print(x[7])
```

```
␣
↪---------------------------------------------------------------------------

       IndexError                                Traceback (most recent call␣
↪last)

       <ipython-input-5-0f1a2bb8a182> in <module>
         1 # Trying to access an index not in list will result in IndexError
    ----> 2 print(x[7])


       IndexError: list index out of range
```

[ ]:
```
# Access part of a list by slicing
x[1:3]
```

[ ]:
```
# Omitting the first value of the slice implies a value of 0
x[:2]
```

[ ]:
```
# Omitting the last value of the slice implies a value of len(list)
x[2:]
```

[ ]:
```
# Check the data type of an object using type() function
type(x)
```

[6]:
```
# The `in` keyword lets you check if a value is contained in the list
x = ["Now", "we", "are", "cooking", "with", 7, "ingredients"]
"This" in x
```

[6]: False

2. Modify the contents of a list

[7]:
```
# The append() method adds an element to the end of a list
fruits = ['Pineapple', 'Banana', 'Apple', 'Melon']
fruits.append('Kiwi')
print(fruits)
```

```
['Pineapple', 'Banana', 'Apple', 'Melon', 'Kiwi']
```

[8]:
```
# The insert() method adds an element to a list at the specified index
fruits.insert(1, 'Orange')
print(fruits)
```

```
['Pineapple', 'Orange', 'Banana', 'Apple', 'Melon', 'Kiwi']
```

[9]:
```
# The insert() method adds an element to a list at the specified index
fruits.insert(0, 'Mango')
print(fruits)
```

```
['Mango', 'Pineapple', 'Orange', 'Banana', 'Apple', 'Melon', 'Kiwi']
```

[10]:
```
# The remove() method deletes the first occurrence of an element in a list
fruits.remove('Banana')
print(fruits)
```

```
['Mango', 'Pineapple', 'Orange', 'Apple', 'Melon', 'Kiwi']
```

[33]:
```
# Trying to remove an element that doesn't exist results in an error
fruits.remove('Strawberry')
print(fruits)
```

```
␣
↪---------------------------------------------------------------------------

ValueError                                Traceback (most recent call␣
↪last)

<ipython-input-33-50fbba439db1> in <module>
  1 # Trying to remove an element that doesn't exist results in an error
----> 2 fruits.remove('Strawberry')
  3 print(fruits)


ValueError: list.remove(x): x not in list
```

[55]:
```
# The pop() method removes the element at a given index and returns it.
# If no index is given, it removes and returns the last element.
fruits.pop(2)
print(fruits)
```

```
['Mango', 'Pineapple', 'Apple', 'Melon', 'Kiwi']
```

[56]:
```
# Reassign the element at a given index with a new value
fruits[1] = 'Mango'
```

[57]:
```
print(fruits)
```

```
['Mango', 'Mango', 'Apple', 'Melon', 'Kiwi']
```

[58]:
```
# Strings are immutable because you need to reassign them to modify them
power = '1.21'
power = power + ' gigawatts'
print(power)
```

```
1.21 gigawatts
```

[59]:
```
# You cannot reassign a specific character within a string
power[0] = '2'
```

```
␣
↪---------------------------------------------------------------------------

       TypeError                                 Traceback (most recent call␣
↪last)

       <ipython-input-59-8817eab4e829> in <module>
         1 # You cannot reassign a specific character within a string
    ----> 2 power[0] = '2'


       TypeError: 'str' object does not support item assignment
```

[60]:
```
# Lists are mutable because you can overwrite their elements
power = [1.21, 'gigawatts']
power[0] = 2.21
print(power)
```

```
[2.21, 'gigawatts']
```

3. Introduction to tuples

[61]:
```
# Tuples are instantiated with parentheses
fullname = ('Masha', 'Z', 'Hopper')

# Tuples are immutable, so their elements cannot be overwritten
fullname[2] = 'Copper'
print(fullname)
```

```
␣
↪---------------------------------------------------------------------------

       TypeError                                 Traceback (most recent call␣
↪last)
```

```
<ipython-input-61-9a0cd4c754f6> in <module>
  3
  4 # Tuples are immutable, so their elements cannot be overwritten
----> 5 fullname[2] = 'Copper'
  6 print(fullname)


TypeError: 'tuple' object does not support item assignment
```

[62]:
```
# You can combine tuples using addition
fullname = fullname + ('Jr',)
print(fullname)
```

```
('Masha', 'Z', 'Hopper', 'Jr')
```

[63]:
```
# The tuple() function converts an object's data type to tuple
fullname = ['Masha', 'Z', 'Hopper']
fullname = tuple(fullname)
print(fullname)
```

```
('Masha', 'Z', 'Hopper')
```

[64]:
```
# Functions that return multiple values return them in a tuple
def to_dollars_cents(price):
    '''
    Split price (float) into dollars and cents.
    '''
    dollars = int(price // 1)
    cents = round(price % 1 * 100)

    return dollars, cents
```

[65]:
```
# Functions that return multiple values return them in a tuple
to_dollars_cents(6.55)
```

[65]:
```
(6, 55)
```

[66]:
```
# "Unpacking" a tuple allows a tuple's elements to be assigned to variables
dollars, cents = to_dollars_cents(6.55)
print(dollars + 1)
print(cents + 1)
```

```
7
56
```

[67]:
```
# The data type of an element of an unpacked tuple is not necessarily a tuple
type(dollars)
```

[67]:
```
int
```

[68]:
```
# Create a list of tuples, each representing the name, age, and position of a
# player on a basketball team
team = [('Marta', 20, 'center'),
        ('Ana', 22, 'point guard'),
        ('Gabi', 22, 'shooting guard'),
        ('Luz', 21, 'power forward'),
        ('Lorena', 19, 'small forward'),
        ]
```

[69]:
```
# Use a for loop to loop over the list, unpack the tuple at each iteration, and
# print one of the values
for name, age, position in team:
    print(name)
```

```
Marta
Ana
Gabi
Luz
Lorena
```

[70]:
```
# This code produces the same result as the code in the cell above
for player in team:
    print(player[0])
```

```
Marta
Ana
Gabi
Luz
Lorena
```

4. More with loops, lists, and tuples

[71]:
```
# Create a list of tuples, each representing the name, age, and position of a
# player on a basketball team
team = [
    ('Marta', 20, 'center'),
    ('Ana', 22, 'point guard'),
    ('Gabi', 22, 'shooting guard'),
    ('Luz', 21, 'power forward'),
    ('Lorena', 19, 'small forward'),
    ]
```

[72]:
```
# Create a function to extract and names and positions from the team list and
# format them to be printed. Returns a list.
def player_position(players):
    result = []
```

```
    for name, age, position in players:
        result.append('Name: {:>19} \nPosition: {:>15}\n'.format(name,␣
↪position))

    return result
```

[73]:
```
# Loop over the list of formatted names and positions produced by
# player_position() function and print them
for player in player_position(team):
    print(player)
```

```
Name:               Marta
Position:          center

Name:                 Ana
Position:     point guard

Name:                Gabi
Position:  shooting guard

Name:                 Luz
Position:   power forward

Name:              Lorena
Position:   small forward
```

[74]:
```
# Nested loops can produce the different combinations of pips (dots) in
# a set of dominoes
for left in range(7):
    for right in range(left, 7):
        print(f"[{left}|{right}]", end=" ")
    print('\n')
```

```
[0|0] [0|1] [0|2] [0|3] [0|4] [0|5] [0|6]

[1|1] [1|2] [1|3] [1|4] [1|5] [1|6]

[2|2] [2|3] [2|4] [2|5] [2|6]

[3|3] [3|4] [3|5] [3|6]

[4|4] [4|5] [4|6]

[5|5] [5|6]

[6|6]
```

[75]:
```
# Create a list of dominoes, with each domino reprented as a tuple
dominoes = []
for left in range(7):
    for right in range(left, 7):
        dominoes.append((left, right))
dominoes
```

[75]:
```
[(0, 0),
 (0, 1),
 (0, 2),
 (0, 3),
 (0, 4),
 (0, 5),
 (0, 6),
 (1, 1),
 (1, 2),
 (1, 3),
 (1, 4),
 (1, 5),
 (1, 6),
 (2, 2),
 (2, 3),
 (2, 4),
 (2, 5),
 (2, 6),
 (3, 3),
 (3, 4),
 (3, 5),
 (3, 6),
 (4, 4),
 (4, 5),
 (4, 6),
 (5, 5),
 (5, 6),
 (6, 6)]
```

[76]:
```
# Select index 1 of the tuple at index 4 in the list of dominoes
dominoes[4][1]
```

[76]:
```
4
```

In the code cells below are two ways to add the total number of pips on each individual domino to a list, as indicated in this diagram:

The first way uses a for loop. The second way uses a list comprehension.

[77]:
```
# You can use a for loop to sum the pips on each domino and append
# the sum to a new list
pips_from_loop = []
for domino in dominoes:
    pips_from_loop.append(domino[0] + domino[1])
print(pips_from_loop)
```

```
[0, 1, 2, 3, 4, 5, 6, 2, 3, 4, 5, 6, 7, 4, 5, 6, 7, 8, 6, 7, 8, 9, 8, 9, 10, 10,
11, 12]
```

[78]:
```
# A list comprehension produces the same result with less code
pips_from_list_comp = [domino[0] + domino[1] for domino in dominoes]
pips_from_loop == pips_from_list_comp
```

[78]:
```
True
```

5. Introduction to dictionaries

[79]:
```
# Create a dictionary with pens as keys and the animals they contain as values.
# Dictionaries can be instantiated using braces.
zoo = {
    'pen_1': 'penguins',
    'pen_2': 'zebras',
    'pen_3': 'lions',
    }

# Selecting the `pen_2` key returns `zebras`--the value stored at that key
zoo['pen_2']
```

[79]:
```
'zebras'
```

[80]:
```
# You cannot access a dictionary's values by name using bracket indexing
# because the computer interprets this as a key, not a value
zoo['zebras']
```

```
␣
↪---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call␣
↪last)

<ipython-input-80-037471d7b0ba> in <module>
  1 # You cannot access a dictionary's values by name using bracket␣
↪indexing
  2 # because the computer interprets this as a key, not a value
```

```
----> 3 zoo['zebras']


KeyError: 'zebras'
```

[81]:
```
# Dictionaries can also be instantiated using the dict() function
zoo = dict(
    pen_1='monkeys',
    pen_2='zebras',
    pen_3='lions',
    )

zoo['pen_2']
```

[81]:
```
'zebras'
```

[82]:
```
# Another way to create a dictionary using the dict() function
zoo = dict(
    [
     ['pen_1', 'monkeys'],
     ['pen_2', 'zebras'],
     ['pen_3', 'lions'],
    ]
)

zoo['pen_2']
```

[82]:
```
'zebras'
```

[83]:
```
# Assign a new key:value pair to an existing dictionary
zoo['pen_4'] = 'crocodiles'
zoo
```

[83]:
```
{'pen_1': 'monkeys',
 'pen_2': 'zebras',
 'pen_3': 'lions',
 'pen_4': 'crocodiles'}
```

[84]:
```
# Dictionaries are unordered and do not support numerical indexing
zoo[2]
```

```
␣
↪---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call␣
↪last)
```

```
<ipython-input-84-5e45321afb99> in <module>
  1 # Dictionaries are unordered and do not support numerical indexing
----> 2 zoo[2]


KeyError: 2
```

[85]:
```
# Use the `in` keyword to produce a Boolean of whether a given key exists in a dictionary
print('pen_1' in zoo)
print('pen_7' in zoo)
```

```
True
False
```

6. Dictionary methods

[86]:
```
# Create a list of tuples, each representing the name, age, and position of a
# player on a basketball team
team = [
    ('Marta', 20, 'center'),
    ('Ana', 22, 'point guard'),
    ('Gabi', 22, 'shooting guard'),
    ('Luz', 21, 'power forward'),
    ('Lorena', 19, 'small forward'),
    ]
```

[87]:
```
# Add new players to the list
team = [
    ('Marta', 20, 'center'),
    ('Ana', 22, 'point guard'),
    ('Gabi', 22, 'shooting guard'),
    ('Luz', 21, 'power forward'),
    ('Lorena', 19, 'small forward'),
    ('Sandra', 19, 'center'),
    ('Mari', 18, 'point guard'),
    ('Esme', 18, 'shooting guard'),
    ('Lin', 18, 'power forward'),
    ('Sol', 19, 'small forward'),
    ]
```

[88]:
```
# Instantiate an empty dictionary
new_team = {}

# Loop over the tuples in the list of players and unpack their values
for name, age, position in team:
```

```
    if position in new_team:                        # If position already a key in␣
↪new_team,
        new_team[position].append((name, age))      # append (name, age) tup to␣
↪list at that value
    else:
        new_team[position] = [(name, age)]          # If position not a key in␣
↪new_team,
                                                    # create a new key whose value␣
↪is a list
                                                    # containing (name, age) tup
new_team
```

[88]:
```
{'center': [('Marta', 20), ('Sandra', 19)],
 'point guard': [('Ana', 22), ('Mari', 18)],
 'shooting guard': [('Gabi', 22), ('Esme', 18)],
 'power forward': [('Luz', 21), ('Lin', 18)],
 'small forward': [('Lorena', 19), ('Sol', 19)]}
```

[89]:
```
# Examine the value at the 'point guard' key
new_team['point guard']
```

[89]:
```
[('Ana', 22), ('Mari', 18)]
```

[90]:
```
# You can access the a dictionary's keys by looping over them
for x in new_team:
    print(x)
```

```
center
point guard
shooting guard
power forward
small forward
```

[91]:
```
# The keys() method returns the keys of a dictionary
new_team.keys()
```

[91]:
```
dict_keys(['center', 'point guard', 'shooting guard', 'power forward', 'small
forward'])
```

[92]:
```
# The values() method returns all the values in a dictionary
new_team.values()
```

[92]:
```
dict_values([[('Marta', 20), ('Sandra', 19)], [('Ana', 22), ('Mari', 18)],
[('Gabi', 22), ('Esme', 18)], [('Luz', 21), ('Lin', 18)], [('Lorena', 19),
('Sol', 19)]])
```

[93]:
```
# The items() method returns both the keys and the values
for a, b in new_team.items():
    print(a, b)
```

```
center [('Marta', 20), ('Sandra', 19)]
point guard [('Ana', 22), ('Mari', 18)]
shooting guard [('Gabi', 22), ('Esme', 18)]
power forward [('Luz', 21), ('Lin', 18)]
small forward [('Lorena', 19), ('Sol', 19)]
```

7. Introduction to sets

[94]:
```
# The set() function converts a list to a set
x = set(['foo', 'bar', 'baz', 'foo'])
print(x)
```

```
{'foo', 'baz', 'bar'}
```

[95]:
```
# The set() function converts a tuple to a set
x = set(('foo','bar','baz', 'foo'))
print(x)
```

```
{'foo', 'baz', 'bar'}
```

[96]:
```
# The set() function converts a string to a set
x = set('foo')
print(x)
```

```
{'f', 'o'}
```

[97]:
```
# You can use braces to instantiate a set
x = {'foo'}
print(type(x))

# But empty braces are reserved for dictionaries
y = {}
print(type(y))
```

```
<class 'set'>
<class 'dict'>
```

[98]:
```
# Instantiating a set with braces treats the contents as literals
x = {'foo'}
print(x)
```

```
{'foo'}
```

[99]:
```
# The intersection() method (&) returns common elements between two sets
set1 = {1, 2, 3, 4, 5, 6}
set2 = {4, 5, 6, 7, 8, 9}
print(set1.intersection(set2))
print(set1 & set2)
```

```
{4, 5, 6}
{4, 5, 6}
```

[100]:
```
# The union() method (|) returns all the elements from two sets, each␣
 ↪represented once
x1 = {'foo', 'bar', 'baz'}
x2 = {'baz', 'qux', 'quux'}
print(x1.union(x2))
print(x1 | x2)
```

```
{'qux', 'baz', 'foo', 'quux', 'bar'}
{'qux', 'baz', 'foo', 'quux', 'bar'}
```

[101]:
```
# The difference() method (-) returns the elements in set1 that aren't in set2
set1 = {1, 2, 3, 4, 5, 6}
set2 = {4, 5, 6, 7, 8, 9}
print(set1.difference(set2))
print(set1 - set2)
```

```
{1, 2, 3}
{1, 2, 3}
```

[102]:
```
# ... and the elements in set2 that aren't in set1
print(set2.difference(set1))
print(set2 - set1)
```

```
{8, 9, 7}
{8, 9, 7}
```

[103]:
```
# The symmetric_difference() method (^) returns all the values from each set␣
 ↪that
# are not in both sets.
set1 = {1, 2, 3, 4, 5, 6}
set2 = {4, 5, 6, 7, 8, 9}
set2.symmetric_difference(set1)
set2 ^ set1
```

[103]:
```
{1, 2, 3, 7, 8, 9}
```

8. Introduction to NumPy

[104]:
```
# Lists cannot be multiplied together
list_a = [1, 2, 3]
list_b = [2, 4, 6]

list_a * list_b
```

```
      ␣
↪---------------------------------------------------------------------------

       TypeError                                 Traceback (most recent call␣
↪last)

       <ipython-input-104-f6837e8a9bfd> in <module>
         3 list_b = [2, 4, 6]
         4
   ----> 5 list_a * list_b


       TypeError: can't multiply sequence by non-int of type 'list'
```

[105]:
```
# To perform element-wise multiplication between two lists, you could
# use a for loop
list_c = []
for i in range(len(list_a)):
    list_c.append(list_a[i] * list_b[i])

list_c
```

[105]: [2, 8, 18]

[106]:
```
# NumPy arrays let you perform array operations

# Import numpy, aliased as np
import numpy as np

# Convert lists to arrays
array_a = np.array(list_a)
array_b = np.array(list_b)

# Perform element-wise multiplication between the arrays
array_a * array_b
```

[106]: array([2, 8, 18])

9. Basic array operations

[107]:
```
import numpy as np

# The np.array() function converts an object to an ndarray
x = np.array([1, 2, 3, 4])
x
```

[107]:
```
array([1, 2, 3, 4])
```

[108]:
```
# Arrays can be indexed
x[-1] = 5
x
```

[108]:
```
array([1, 2, 3, 5])
```

[109]:
```
# Trying to access an index that doesn't exist will throw an error
x[4] = 10
```

```
␣
↪---------------------------------------------------------------------------

IndexError                                Traceback (most recent call␣
↪last)

<ipython-input-109-43f18e2bca94> in <module>
  1 # Trying to access an index that doesn't exist will throw an error
----> 2 x[4] = 10


IndexError: index 4 is out of bounds for axis 0 with size 4
```

[110]:
```
# Arrays cast every element they contain as the same data type
arr = np.array([1, 2, 'coconut'])
arr
```

[110]:
```
array(['1', '2', 'coconut'], dtype='<U21')
```

[111]:
```
# NumPy arrays are a class called `ndarray`
print(type(arr))
```

```
<class 'numpy.ndarray'>
```

[112]:
```
# The dtype attribute returns the data type of an array's contents
arr = np.array([1, 2, 3])
arr.dtype
```

[112]:
```
dtype('int64')
```

[113]:
```
# The shape attribute returns the number of elements in each dimension
# of an array
arr.shape
```

[113]:
```
(3,)
```

[114]:
```
# The ndim attribute returns the number of dimensions in an array
arr.ndim
```

[114]:
```
1
```

[115]:
```
# Create a 2D array by passing a list of lists to np.array() function
arr_2d = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
print(arr_2d.shape)
print(arr_2d.ndim)
arr_2d
```

```
(4, 2)
2
```

[115]:
```
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]])
```

[116]:
```
# Create a 3D aray by passing a list of two lists of lists to np.array()␣
↪function
arr_3d = np.array([[[1, 2, 3],
                    [3, 4, 5]],

                   [[5, 6, 7],
                    [7, 8, 9]]]
)

print(arr_3d.shape)
print(arr_3d.ndim)
arr_3d
```

```
(2, 2, 3)
3
```

[116]:
```
array([[[1, 2, 3],
        [3, 4, 5]],

       [[5, 6, 7],
```

```
       [7, 8, 9]]])
```

[117]:
```
# The reshape() method changes the shape of an array
arr_2d = arr_2d.reshape(2, 4)
arr_2d
```

[117]:
```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

[118]:
```
# Create new array
arr = np.array([1, 2, 3, 4, 5])

# The mean() method returns the mean of the elements in an array
np.mean(arr)
```

[118]:
```
3.0
```

[119]:
```
# The log() method returns the natural logarithm of the elements in an array
np.log(arr)
```

[119]:
```
array([0.        , 0.69314718, 1.09861229, 1.38629436, 1.60943791])
```

[120]:
```
# The floor() method returns the value of a number rounded down
# to the nearest integer
np.floor(5.7)
```

[120]:
```
5.0
```

[121]:
```
# The floor() method returns the value of a number rounded up
# to the nearest integer
np.ceil(5.3)
```

[121]:
```
6.0
```

10. Introduction to pandas

[122]:
```
# NumPy and pandas are typically imported together.
# np and pd are conventional aliases.
import numpy as np
import pandas as pd
```

[123]:
```
# Read in data from a .csv file
dataframe = pd.read_csv('https://raw.githubusercontent.com/adacert/titanic/main/
 ↪train.csv')

# Print the first 25 rows
dataframe.head(25)
```

[123]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th…	female	38.0	1
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0
5	6	0	3	Moran, Mr. James	male	NaN	0
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0
12	13	0	3	Saundercock, Mr. William Henry	male	20.0	0
13	14	0	3	Andersson, Mr. Anders Johan	male	39.0	1
14	15	0	3	Vestrom, Miss. Hulda Amanda Adolfina	female	14.0	0
15	16	1	2	Hewlett, Mrs. (Mary D Kingcome)	female	55.0	0
16	17	0	3	Rice, Master. Eugene	male	2.0	4
17	18	1	2	Williams, Mr. Charles Eugene	male	NaN	0
18	19	0	3	Vander Planke, Mrs. Julius (Emelia Maria Vande…	female	31.0	1
19	20	1	3				
20	21	0	2				
21	22	1	2				
22	23	1	3				
23	24	1	1				
24	25	0	3				

```
19                            Masselmani, Mrs. Fatima  female   NaN      0
20                               Fynney, Mr. Joseph J    male  35.0      0
21                              Beesley, Mr. Lawrence    male  34.0      0
22                        McGowan, Miss. Anna "Annie"  female  15.0      0
23                       Sloper, Mr. William Thompson    male  28.0      0
24                      Palsson, Miss. Torborg Danira  female   8.0      3

    Parch            Ticket     Fare Cabin Embarked
0       0         A/5 21171   7.2500   NaN        S
1       0          PC 17599  71.2833   C85        C
2       0  STON/O2. 3101282   7.9250   NaN        S
3       0            113803  53.1000  C123        S
4       0            373450   8.0500   NaN        S
5       0            330877   8.4583   NaN        Q
6       0             17463  51.8625   E46        S
7       1            349909  21.0750   NaN        S
8       2            347742  11.1333   NaN        S
9       0            237736  30.0708   NaN        C
10      1          PP 9549  16.7000    G6        S
11      0            113783  26.5500  C103        S
12      0         A/5. 2151   8.0500   NaN        S
13      5            347082  31.2750   NaN        S
14      0            350406   7.8542   NaN        S
15      0            248706  16.0000   NaN        S
16      1            382652  29.1250   NaN        Q
17      0            244373  13.0000   NaN        S
18      0            345763  18.0000   NaN        S
19      0              2649   7.2250   NaN        C
20      0            239865  26.0000   NaN        S
21      0            248698  13.0000   D56        S
22      0            330923   8.0292   NaN        Q
23      0            113788  35.5000    A6        S
24      1            349909  21.0750   NaN        S
```

[124]:
```
# Calculate the mean of the Age column
dataframe['Age'].mean()
```

[124]:
```
29.69911764705882
```

[125]:
```
# Calculate the maximum value contained in the Age column
dataframe['Age'].max()
```

[125]:
```
80.0
```

[126]:
```
# Calculate the minimum value contained in the Age column
dataframe['Age'].min()
```

[126]:
```
0.42
```

[127]:
```
# Calculate the standard deviation of the values in the Age column
dataframe['Age'].std()
```

[127]:
```
14.526497332334044
```

[128]:
```
# Return the number of rows that share the same value in the Pclass column
dataframe['Pclass'].value_counts()
```

[128]:
```
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

[129]:
```
# The describe() method returns summary statistics of the dataframe
dataframe.describe()
```

[129]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

[130]:
```
# Filter the data to return only rows where value in Age column is greater than␣
↪60
# and value in Pclass column equals 3
dataframe[(dataframe['Age'] > 60) & (dataframe['Pclass'] == 3)]
```

[130]:

	PassengerId	Survived	Pclass	Name	Sex	Age
116	117	0	3	Connors, Mr. Patrick	male	70.5
280	281	0	3	Duane, Mr. Frank	male	65.0
326	327	0	3	Nysveen, Mr. Johan Hansen	male	61.0

```
483          484         1       3     Turkula, Mrs. (Hedwig)  female  63.0
851          852         0       3        Svensson, Mr. Johan    male  74.0

     SibSp  Parch  Ticket    Fare Cabin Embarked
116      0      0  370369  7.7500   NaN        Q
280      0      0  336439  7.7500   NaN        Q
326      0      0  345364  6.2375   NaN        S
483      0      0    4134  9.5875   NaN        S
851      0      0  347060  7.7750   NaN        S
```

[131]:
```
# Create a new column called 2023_Fare that contains the inflation-adjusted
# fare of each ticket in 2023 pounds
dataframe['2023_Fare'] = dataframe['Fare'] * 146.14
dataframe
```

[131]:
```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           …       …     …
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                 Name     Sex   Age  SibSp  \
0                             Braund, Mr. Owen Harris    male  22.0      1
1    Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                                Heikkinen, Miss. Laina  female  26.0      0
3          Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                               Allen, Mr. William Henry    male  35.0      0
..                                                   …     …   …    …
886                               Montvila, Rev. Juozas    male  27.0      0
887                        Graham, Miss. Margaret Edith  female  19.0      0
888            Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1
889                               Behr, Mr. Karl Howell    male  26.0      0
890                                 Dooley, Mr. Patrick    male  32.0      0

     Parch            Ticket     Fare Cabin Embarked     2023_Fare
0        0         A/5 21171   7.2500   NaN        S   1059.515000
1        0          PC 17599  71.2833   C85        C  10417.341462
2        0  STON/O2. 3101282   7.9250   NaN        S   1158.159500
3        0            113803  53.1000  C123        S   7760.034000
4        0            373450   8.0500   NaN        S   1176.427000
```

```
..     …                …       …   …      …          …
886    0           211536  13.0000   NaN      S   1899.820000
887    0           112053  30.0000   B42      S   4384.200000
888    2       W./C. 6607  23.4500   NaN      S   3426.983000
889    0           111369  30.0000  C148      C   4384.200000
890    0           370376   7.7500   NaN      Q   1132.585000

[891 rows x 13 columns]
```

[132]:
```
# Use iloc to access data using index numbers.
# Select row 1, column 3.
dataframe.iloc[1][3]
```

[132]:
```
'Cumings, Mrs. John Bradley (Florence Briggs Thayer)'
```

[133]:
```
# Group customers by Sex and Pclass and calculate the total paid for each group
# and the mean price paid for each group
fare = dataframe.groupby(['Sex', 'Pclass']).agg({'Fare': ['count', 'sum']})
fare['fare_avg'] = fare['Fare']['sum'] / fare['Fare']['count']
fare
```

[133]:
```
               Fare                fare_avg
              count        sum
Sex    Pclass
female 1         94  9975.8250  106.125798
       2         76  1669.7292   21.970121
       3        144  2321.1086   16.118810
male   1        122  8201.5875   67.226127
       2        108  2132.1125   19.741782
       3        347  4393.5865   12.661633
```

11. pandas basics

[134]:
```
import pandas as pd

# Use pd.DataFrame() function to create a dataframe from a dictionary
data = {'col1': [1, 2], 'col2': [3, 4]}
df = pd.DataFrame(data=data)
df
```

[134]:
```
   col1  col2
0     1     3
1     2     4
```

[135]:
```
# Use pd.DataFrame() function to create a dataframe from a numpy array
df2 = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]),
                   columns=['a', 'b', 'c'], index=['x', 'y', 'z'])
```

```
df2
```

[135]:
```
   a  b  c
x  1  2  3
y  4  5  6
z  7  8  9
```

[136]:
```
# Use pd.read_csv() function to create a dataframe from a .csv file
# from a URL or filepath
df3 = pd.read_csv('https://raw.githubusercontent.com/adacert/titanic/main/train.
 ↪csv')
df3.head()
```

[136]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

[137]:
```
# Print class of first row
print(type(df3.iloc[0]))

# Print class of "Name" column
print(type(df3['Name']))
```

```
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
```

[138]:
```
# Create a copy of df3 named 'titanic'
titanic = df3

# The head() method outputs the first 5 rows of dataframe
```

```
titanic.head()
```

[138]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th…	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

[139]:
```
# The columns attribute returns an Index object containing the dataframe's columns
titanic.columns
```

[139]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

[140]:
```
# The shape attribute returns the shape of the dataframe (rows, columns)
titanic.shape
```

[140]:
```
(891, 12)
```

[141]:
```
# The info() method returns summary information about the dataframe
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
```

```
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

[142]:
```
# You can select a column by name using brackets
titanic['Age']
```

[142]:
```
0      22.0
1      38.0
2      26.0
3      35.0
4      35.0
       …
886    27.0
887    19.0
888     NaN
889    26.0
890    32.0
Name: Age, Length: 891, dtype: float64
```

[143]:
```
# You can select a column by name using dot notation
# only when its name contains no spaces or special characters
titanic.Age
```

[143]:
```
0      22.0
1      38.0
2      26.0
3      35.0
4      35.0
       …
886    27.0
887    19.0
888     NaN
889    26.0
890    32.0
Name: Age, Length: 891, dtype: float64
```

[144]:
```
# You can create a DataFrame object of specific columns using a list
# of column names inside brackets
```

```
titanic[['Name', 'Age']]
```

[144]:
```
                                                 Name   Age
0                             Braund, Mr. Owen Harris  22.0
1    Cumings, Mrs. John Bradley (Florence Briggs Th…  38.0
2                              Heikkinen, Miss. Laina  26.0
3        Futrelle, Mrs. Jacques Heath (Lily May Peel)  35.0
4                            Allen, Mr. William Henry  35.0
..                                                ...   …
886                            Montvila, Rev. Juozas  27.0
887                     Graham, Miss. Margaret Edith  19.0
888         Johnston, Miss. Catherine Helen "Carrie"   NaN
889                            Behr, Mr. Karl Howell  26.0
890                              Dooley, Mr. Patrick  32.0

[891 rows x 2 columns]
```

[145]:
```
# Use iloc to return a Series object of the data in row 0
titanic.iloc[0]
```

[145]:
```
PassengerId                     1
Survived                        0
Pclass                          3
Name           Braund, Mr. Owen Harris
Sex                            male
Age                            22.0
SibSp                             1
Parch                             0
Ticket                    A/5 21171
Fare                           7.25
Cabin                           NaN
Embarked                          S
Name: 0, dtype: object
```

[146]:
```
# Use iloc to return a DataFrame view of the data in row 0
titanic.iloc[[0]]
```

[146]:
```
   PassengerId  Survived  Pclass                     Name   Sex   Age  SibSp  \
0            1         0       3  Braund, Mr. Owen Harris  male  22.0      1

   Parch     Ticket  Fare Cabin Embarked
0      0  A/5 21171  7.25   NaN        S
```

[147]:
```
# Use iloc to return a DataFrame view of the data in rows 0, 1, 2
titanic.iloc[0:3]
```

[147]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
```

[148]:
```
# Use iloc to return a DataFrame view of rows 0-2 at columns 3 and 4
titanic.iloc[0:3, [3, 4]]
```

[148]:
```
                                                Name     Sex
0                            Braund, Mr. Owen Harris    male
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female
2                             Heikkinen, Miss. Laina  female
```

[149]:
```
# Use iloc to return a DataFrame view of all rows at column 3
titanic.iloc[:, [3]]
```

[149]:
```
                                                 Name
0                             Braund, Mr. Owen Harris
1    Cumings, Mrs. John Bradley (Florence Briggs Th…
2                               Heikkinen, Miss. Laina
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                             Allen, Mr. William Henry
..                                                 …
886                             Montvila, Rev. Juozas
887                      Graham, Miss. Margaret Edith
888          Johnston, Miss. Catherine Helen "Carrie"
889                             Behr, Mr. Karl Howell
890                               Dooley, Mr. Patrick

[891 rows x 1 columns]
```

[150]:
```
# Use iloc to access value in row 0, column 3
titanic.iloc[0, 3]
```

[150]:
```
'Braund, Mr. Owen Harris'
```

[151]:
```
# Use loc to access values in rows 0-3 at just the Name column
titanic.loc[0:3, ['Name']]
```

[151]:
```
                                                Name
0                            Braund, Mr. Owen Harris
1  Cumings, Mrs. John Bradley (Florence Briggs Th…
2                             Heikkinen, Miss. Laina
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)
```

[152]:
```
# Create a new column in the dataframe containing the value in the Age column +
↪100
titanic['Age_plus_100'] = titanic['Age'] + 100
titanic.head()
```

[152]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked  Age_plus_100
0      0         A/5 21171   7.2500   NaN        S         122.0
1      0          PC 17599  71.2833   C85        C         138.0
2      0  STON/O2. 3101282   7.9250   NaN        S         126.0
3      0            113803  53.1000  C123        S         135.0
4      0            373450   8.0500   NaN        S         135.0
```

12. Boolean masking

[153]:
```
# Instantiate a dictionary of planetary data
data = {'planet': ['Mercury', 'Venus', 'Earth', 'Mars',
                   'Jupiter', 'Saturn', 'Uranus', 'Neptune'],
        'radius_km': [2440, 6052, 6371, 3390, 69911, 58232,
                      25362, 24622],
        'moons': [0, 0, 1, 2, 80, 83, 27, 14]
         }
# Use pd.DataFrame() function to convert dictionary to dataframe
planets = pd.DataFrame(data)
planets
```

[153]:
	planet	radius_km	moons
0	Mercury	2440	0
1	Venus	6052	0
2	Earth	6371	1
3	Mars	3390	2
4	Jupiter	69911	80
5	Saturn	58232	83
6	Uranus	25362	27
7	Neptune	24622	14

[154]:
```
# Create a Boolean mask of planets with fewer than 20 moons
mask = planets['moons'] < 20
mask
```

[154]:
```
0     True
1     True
2     True
3     True
4    False
5    False
6    False
7     True
Name: moons, dtype: bool
```

[155]:
```
# Apply the Boolean mask to the dataframe to filter it so it contains
# only the planets with fewer than 20 moons
planets[mask]
```

[155]:
	planet	radius_km	moons
0	Mercury	2440	0
1	Venus	6052	0
2	Earth	6371	1
3	Mars	3390	2
7	Neptune	24622	14

[156]:
```
# Define the Boolean mask and apply it in a single line
planets[planets['moons'] < 20]
```

[156]:
	planet	radius_km	moons
0	Mercury	2440	0
1	Venus	6052	0
2	Earth	6371	1
3	Mars	3390	2
7	Neptune	24622	14

[157]:
```
# Boolean masks don't change the data. They're just views.
planets
```

[157]:
```
    planet  radius_km  moons
0  Mercury       2440      0
1    Venus       6052      0
2    Earth       6371      1
3     Mars       3390      2
4  Jupiter      69911     80
5   Saturn      58232     83
6   Uranus      25362     27
7  Neptune      24622     14
```

[158]:
```
# You can assign a dataframe view to a named variable
moons_under_20 = planets[mask]
moons_under_20
```

[158]:
```
    planet  radius_km  moons
0  Mercury       2440      0
1    Venus       6052      0
2    Earth       6371      1
3     Mars       3390      2
7  Neptune      24622     14
```

[159]:
```
# Create a Boolean mask of planets with fewer than 10 moons OR more than 50␣
↪moons
mask = (planets['moons'] < 10) | (planets['moons'] > 50)
mask
```

[159]:
```
0     True
1     True
2     True
3     True
4     True
5     True
6    False
7    False
Name: moons, dtype: bool
```

[160]:
```
# Apply the Boolean mask to filter the data
planets[mask]
```

[160]:
```
    planet  radius_km  moons
0  Mercury       2440      0
1    Venus       6052      0
2    Earth       6371      1
3     Mars       3390      2
4  Jupiter      69911     80
5   Saturn      58232     83
```

[161]:
```
# Create a Boolean mask of planets with more than 20 moons, excluding them if␣
↪they
# have 80 moons or if their radius is less than 50,000 km.
mask = (planets['moons'] > 20) & ~(planets['moons'] == 80) &␣
↪~(planets['radius_km'] < 50000)

# Apply the mask
planets[mask]
```

[161]:

	planet	radius_km	moons
5	Saturn	58232	83

13. Grouping and aggregation

[162]:
```
import numpy as np
import pandas as pd

# Instantiate a dictionary of planetary data
data = {'planet': ['Mercury', 'Venus', 'Earth', 'Mars',
                   'Jupiter', 'Saturn', 'Uranus', 'Neptune'],
        'radius_km': [2440, 6052, 6371, 3390, 69911, 58232,
                      25362, 24622],
        'moons': [0, 0, 1, 2, 80, 83, 27, 14],
        'type': ['terrestrial', 'terrestrial', 'terrestrial', 'terrestrial',
                 'gas giant', 'gas giant', 'ice giant', 'ice giant'],
        'rings': ['no', 'no', 'no', 'no', 'yes', 'yes', 'yes','yes'],
        'mean_temp_c': [167, 464, 15, -65, -110, -140, -195, -200],
        'magnetic_field': ['yes', 'no', 'yes', 'no', 'yes', 'yes', 'yes', 'yes']
        }

# Use pd.DataFrame() function to convert dictionary to dataframe
planets = pd.DataFrame(data)
planets
```

[162]:

	planet	radius_km	moons	type	rings	mean_temp_c	magnetic_field
0	Mercury	2440	0	terrestrial	no	167	yes
1	Venus	6052	0	terrestrial	no	464	no
2	Earth	6371	1	terrestrial	no	15	yes
3	Mars	3390	2	terrestrial	no	-65	no
4	Jupiter	69911	80	gas giant	yes	-110	yes
5	Saturn	58232	83	gas giant	yes	-140	yes
6	Uranus	25362	27	ice giant	yes	-195	yes
7	Neptune	24622	14	ice giant	yes	-200	yes

[37]:
```
# The groupby() function returns a groupby object
planets.groupby(['type'])
```

[37]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7ff98cfd24d0>
```

[38]:
```
# Apply the sum() function to the groupby object to get the sum
# of the values in each numerical column for each group
planets.groupby(['type']).sum()
```

[38]:

type	radius_km	moons	mean_temp_c
gas giant	128143	163	-250
ice giant	49984	41	-395
terrestrial	18253	3	581

[39]:
```
# Apply the sum function to the groupby object and select
# only the 'moons' column
planets.groupby(['type']).sum()[['moons']]
```

[39]:

type	moons
gas giant	163
ice giant	41
terrestrial	3

[40]:
```
# Group by type and magnetic_field and get the mean of the values
# in the numeric columns for each group
planets.groupby(['type', 'magnetic_field']).mean()
```

[40]:

type	magnetic_field	radius_km	moons	mean_temp_c
gas giant	yes	64071.5	81.5	-125.0
ice giant	yes	24992.0	20.5	-197.5
terrestrial	no	4721.0	1.0	199.5
	yes	4405.5	0.5	91.0

[41]:
```
# Group by type, then use the agg() function to get the mean and median
# of the values in the numeric columns for each group
planets.groupby(['type']).agg(['mean', 'median'])
```

[41]:

type	radius_km		moons		mean_temp_c	
	mean	median	mean	median	mean	median
gas giant	64071.50	64071.5	81.50	81.5	-125.00	-125.0
ice giant	24992.00	24992.0	20.50	20.5	-197.50	-197.5
terrestrial	4563.25	4721.0	0.75	0.5	145.25	91.0

[42]:
```
# Group by type and magnetic_field, then use the agg() function to get the
# mean and max of the values in the numeric columns for each group
planets.groupby(['type', 'magnetic_field']).agg(['mean', 'max'])
```

[42]:

		radius_km		moons		mean_temp_c	
		mean	max	mean	max	mean	max
type	magnetic_field						
gas giant	yes	64071.5	69911	81.5	83	-125.0	-110
ice giant	yes	24992.0	25362	20.5	27	-197.5	-195
terrestrial	no	4721.0	6052	1.0	2	199.5	464
	yes	4405.5	6371	0.5	1	91.0	167

[43]:
```
# Define a function that returns the 90 percentile of an array
def percentile_90(x):
    return x.quantile(0.9)
```

[44]:
```
# Group by type and magnetic_field, then use the agg() function to apply the
# mean and the custom-defined `percentile_90()` function to the numeric
# columns for each group
planets.groupby(['type', 'magnetic_field']).agg(['mean', percentile_90])
```

[44]:

		radius_km		moons		mean_temp_c	
		mean	percentile_90	mean	percentile_90	mean	percentile_90
type	magnetic_field						
gas giant	yes	64071.5	68743.1	81.5	82.7	-125.0	-113.0
ice giant	yes	24992.0	25288.0	20.5	25.7	-197.5	-195.5
terrestrial	no	4721.0	5785.8	1.0	1.8	199.5	411.1
	yes	4405.5	5977.9	0.5	0.9	91.0	151.8

14. Merging and joining data

[45]:
```
import numpy as np
import pandas as pd

# Instantiate a dictionary of planetary data
data = {'planet': ['Mercury', 'Venus', 'Earth', 'Mars'],
        'radius_km': [2440, 6052, 6371, 3390],
        'moons': [0, 0, 1, 2],
        }
# Use pd.DataFrame() function to convert dictionary to dataframe
df1 = pd.DataFrame(data)
df1
```

[45]:

	planet	radius_km	moons
0	Mercury	2440	0
1	Venus	6052	0
2	Earth	6371	1
3	Mars	3390	2

[46]:
```
# Instantiate a dictionary of planetary data
data = {'planet': ['Jupiter', 'Saturn', 'Uranus', 'Neptune'],
        'radius_km': [69911, 58232, 25362, 24622],
        'moons': [80, 83, 27, 14],
        }
# Use pd.DataFrame() function to convert dictionary to dataframe
df2 = pd.DataFrame(data)
df2
```

[46]:

	planet	radius_km	moons
0	Jupiter	69911	80
1	Saturn	58232	83
2	Uranus	25362	27
3	Neptune	24622	14

[47]:
```
# The pd.concat() function can combine the two dataframes along axis 0,
# with the second dataframe being added as new rows to the first dataframe
df3 = pd.concat([df1, df2], axis=0)
df3
```

[47]:

	planet	radius_km	moons
0	Mercury	2440	0
1	Venus	6052	0
2	Earth	6371	1
3	Mars	3390	2
0	Jupiter	69911	80
1	Saturn	58232	83
2	Uranus	25362	27
3	Neptune	24622	14

[48]:
```
# Reset the row indices
df3 = df3.reset_index(drop=True)
df3
```

[48]:

	planet	radius_km	moons
0	Mercury	2440	0
1	Venus	6052	0
2	Earth	6371	1
3	Mars	3390	2
4	Jupiter	69911	80
5	Saturn	58232	83

```
6   Uranus      25362     27
7  Neptune      24622     14
```

[49]:
```
# NOTE: THIS CELL WAS NOT SHOWN IN THE INSTRUCTIONAL VIDEO, BUT WAS RUN AS A
#       SETUP CELL
data = {'planet': ['Earth', 'Mars','Jupiter', 'Saturn', 'Uranus',
                   'Neptune', 'Janssen', 'Tadmor'],
        'type': ['terrestrial', 'terrestrial','gas giant', 'gas giant',
                 'ice giant', 'ice giant', 'super earth','gas giant'],
        'rings': ['no', 'no', 'yes', 'yes', 'yes','yes', 'no', None],
        'mean_temp_c': [15, -65, -110, -140, -195, -200, None, None],
        'magnetic_field': ['yes', 'no', 'yes', 'yes', 'yes', 'yes', None, None],
        'life': [1, 0, 0, 0, 0, 0, 1, 1]
        }
df4 = pd.DataFrame(data)
```

[50]:
```
df4
```

[50]:
```
    planet         type rings  mean_temp_c magnetic_field  life
0    Earth  terrestrial    no         15.0            yes     1
1     Mars  terrestrial    no        -65.0             no     0
2  Jupiter    gas giant   yes       -110.0            yes     0
3   Saturn    gas giant   yes       -140.0            yes     0
4   Uranus    ice giant   yes       -195.0            yes     0
5  Neptune    ice giant   yes       -200.0            yes     0
6  Janssen  super earth    no          NaN           None     1
7   Tadmor    gas giant  None          NaN           None     1
```

[51]:
```
# Use pd.merge() to combine dataframes.
# Inner merge retains only keys that appear in both dataframes.
inner = pd.merge(df3, df4, on='planet', how='inner')
inner
```

[51]:
```
    planet  radius_km  moons         type rings  mean_temp_c magnetic_field  \
0    Earth       6371      1  terrestrial    no         15.0            yes
1     Mars       3390      2  terrestrial    no        -65.0             no
2  Jupiter      69911     80    gas giant   yes       -110.0            yes
3   Saturn      58232     83    gas giant   yes       -140.0            yes
4   Uranus      25362     27    ice giant   yes       -195.0            yes
5  Neptune      24622     14    ice giant   yes       -200.0            yes

   life
0     1
1     0
2     0
3     0
4     0
```

```
5     0
```

[52]:
```
# Use pd.merge() to combine dataframes.
# Outer merge retains all keys from both dataframes.
outer = pd.merge(df3, df4, on='planet', how='outer')
outer
```

[52]:

	planet	radius_km	moons	type	rings	mean_temp_c	magnetic_field	life
0	Mercury	2440.0	0.0	NaN	NaN	NaN	NaN	NaN
1	Venus	6052.0	0.0	NaN	NaN	NaN	NaN	NaN
2	Earth	6371.0	1.0	terrestrial	no	15.0	yes	1.0
3	Mars	3390.0	2.0	terrestrial	no	-65.0	no	0.0
4	Jupiter	69911.0	80.0	gas giant	yes	-110.0	yes	0.0
5	Saturn	58232.0	83.0	gas giant	yes	-140.0	yes	0.0
6	Uranus	25362.0	27.0	ice giant	yes	-195.0	yes	0.0
7	Neptune	24622.0	14.0	ice giant	yes	-200.0	yes	0.0
8	Janssen	NaN	NaN	super earth	no	NaN	None	1.0
9	Tadmor	NaN	NaN	gas giant	None	NaN	None	1.0

[53]:
```
# Use pd.merge() to combine dataframes.
# Left merge retains only keys that appear in the left dataframe.
left = pd.merge(df3, df4, on='planet', how='left')
left
```

[53]:

	planet	radius_km	moons	type	rings	mean_temp_c	magnetic_field	life
0	Mercury	2440	0	NaN	NaN	NaN	NaN	NaN
1	Venus	6052	0	NaN	NaN	NaN	NaN	NaN
2	Earth	6371	1	terrestrial	no	15.0	yes	1.0
3	Mars	3390	2	terrestrial	no	-65.0	no	0.0
4	Jupiter	69911	80	gas giant	yes	-110.0	yes	0.0
5	Saturn	58232	83	gas giant	yes	-140.0	yes	0.0
6	Uranus	25362	27	ice giant	yes	-195.0	yes	0.0
7	Neptune	24622	14	ice giant	yes	-200.0	yes	0.0

```
0   NaN
1   NaN
2   1.0
3   0.0
4   0.0
5   0.0
6   0.0
7   0.0
```

[54]:
```
# Use pd.merge() to combine dataframes.
# Right merge retains only keys that appear in right dataframe.
right = pd.merge(df3, df4, on='planet', how='right')
right
```

[54]:

	planet	radius_km	moons	type	rings	mean_temp_c	magnetic_field	life
0	Earth	6371.0	1.0	terrestrial	no	15.0	yes	1
1	Mars	3390.0	2.0	terrestrial	no	-65.0	no	0
2	Jupiter	69911.0	80.0	gas giant	yes	-110.0	yes	0
3	Saturn	58232.0	83.0	gas giant	yes	-140.0	yes	0
4	Uranus	25362.0	27.0	ice giant	yes	-195.0	yes	0
5	Neptune	24622.0	14.0	ice giant	yes	-200.0	yes	0
6	Janssen	NaN	NaN	super earth	no	NaN	None	1
7	Tadmor	NaN	NaN	gas giant	None	NaN	None	1