What is the difference between a NumPy array and a list?

PRACTICING CODING INTERVIEW QUESTIONS IN PYTHON

Kirill Smirnov
Data Science Consultant, Altran

NumPy array

```
import numpy as np
```

```
num_array = np.array([1, 2, 3, 4, 5])
print(num_array)
```

```
[1 2 3 4 5]
```

```
num_list = [1, 2, 3, 4, 5]
print(num_list)
```

```
[1, 2, 3, 4, 5]
```

Similarities between an array and a list

```
num_array = np.array([1, 2, 3, 4, 5])
```

```
num_list = [1, 2, 3, 4, 5]
```

```
for item in num_array:
    print(item)
```

```
for item in num_list:
    print(item)
```

```
1
2
3
4
5
```

```
1
2
3
4
5
```

Similarities between an array and a list

```
num_array = np.array([1, 2, 3, 4, 5])
```

```
num_list = [1, 2, 3, 4, 5]
```

```
num_array[1]
```

```
num_list[1]
```

```
2
```

```
2
```

```
num_array[1:4]
```

```
num_list[1:4]
```

```
array([2, 3, 4])
```

```
[2, 3, 4]
```

Similarities between an array and a list

```
num_array = np.array([1, 2, 3, 4, 5])
```

```
num_array[3] = 40
print(num_array)
```

```
[1 2 3 40 5]
```

```
num_array[0:3] = [10, 20, 30]
print(num_array)
```

```
[10 20 30 40 5]
```

```
num_list = [1, 2, 3, 4, 5]
```

```
num_list[3] = 40
print(num_list)
```

```
[1, 2, 3, 40, 5]
```

```
num_list[0:3] = [10, 20, 30]
print(num_list)
```

```
[10, 20, 30, 40, 5]
```

Difference between an array an a list

NumPy arrays are designed for high efficiency computations

- NumPy arrays store values of the same type

.dtype property

```
num_array = np.array([1, 2, 3, 4, 5])
```

```
num_array.dtype
```

```
dtype('int64')
```

Changing the data type of an element

```
num_array = np.array([1, 2, 3, 4, 5])
```

```
num_array[2] = 'three'
```

```
ValueError
```

```
num_list = [1, 2, 3, 4, 5]
```

```
num_list[2] = 'three'
print(num_list)
```

```
[1, 2, 'three', 4, 5]
```

Specifying the data type explicitly

```
num_array = np.array([1, 2, 3, 4, 5])
```

```
num_array = np.array([1, 2, 3, 4, 5], dtype = np.dtype('int64'))
print(num_array)
```

```
[1 2 3 4 5]
```

```
num_array.dtype
```

```
dtype('int64')
```

Specifying the data type explicitly

```
num_array = np.array([1, 2, 3, 4, 5])
```

```
num_array = np.array([1, 2, 3, 4, 5], dtype = np.dtype('str'))
print(num_array)
```

```
['1' '2' '3' '4' '5']
```

```
num_array.dtype
```

```
dtype('<U1')
```

Object as a data type

```
num_array = np.array([1, 2, 3, 4, 5], dtype = np.dtype('O'))
```

```
num_array[2] = 'three'
print(num_array)
```

```
[1 2 'three' 4 5]
```

Difference between an array and a list

NumPy arrays are designed for high efficiency computations

- NumPy arrays store values of a concrete data type
- NumPy arrays have a special way to access its elements

Accessing items

```
list2d = [
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10],
    [11, 12, 13, 14, 15]
]
```

```
# Retrieve 8
list2d[1][2]
```

```
8
```

```
array2d = np.array([
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10],
    [11, 12, 13, 14, 15]
])
```

```
# Retrieve 8
array2d[1][2]
```

```
8
```

Accessing items

```
list2d = [
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10],
    [11, 12, 13, 14, 15]
]
```

```
# Retrieve 8
list2d[1][2]
```

```
8
```

```
array2d = np.array([
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10],
    [11, 12, 13, 14, 15]
])
```

```
# Retrieve 8
array2d[1, 2]
```

```
8
```

Accessing items

```
list2d = [
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10],
    [11, 12, 13, 14, 15]
]
```

```
# Retrieve [[2, 3, 4], [7, 8, 9]]
```

```
array2d = np.array([
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10],
    [11, 12, 13, 14, 15]
])
```

```
# Retrieve [[2, 3, 4], [7, 8, 9]]
```

Accessing items

```
list2d = [
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10],
    [11, 12, 13, 14, 15]
]
```

```
array2d = np.array([
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10],
    [11, 12, 13, 14, 15]
])
```

```
# Retrieve [[2, 3, 4], [7, 8, 9]]
[
    [list2d[j][1:4] for j in range(0, 2)]
]
```

```
# Retrieve [[2, 3, 4], [7, 8, 9]]
```

```
[[2, 3, 4], [7, 8, 9]]
```

Accessing items

```
list2d = [
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10],
    [11, 12, 13, 14, 15]
]
```

```
# Retrieve [[2, 3, 4], [7, 8, 9]]
[
    [list2d[j][1:4] for j in range(0, 2)]
]
```

```
[[2, 3, 4], [7, 8, 9]]
```

```
array2d = np.array([
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10],
    [11, 12, 13, 14, 15]
])
```

```
# Retrieve [[2, 3, 4], [7, 8, 9]]
array2d[0:2, 1:4]
```

```
array([[2, 3, 4],
       [7, 8, 9]])
```

Difference between an array and a list

NumPy arrays are designed for high efficiency computations

- NumPy arrays store values of a concrete data type
- NumPy arrays have a special way to access its elements
- NumPy arrays have efficient way to perform operations on them.

Operations +, -, *, / with lists

```
num_list1 = [1, 2, 3]
num_list2 = [10, 20, 30]
```

```
num_list1 + num_list2
```

```
[1, 2, 3, 10, 20, 30]
```

```
num_list2 - num_list1
```

```
TypeError
```

```
num_list1 * num_list2
```

```
TypError
```

```
num_list2 / num_list1
```

```
TypeError
```

Operations +, -, *, / with arrays

```
num_array1 = np.array([1, 2, 3])
num_array2 = np.array([10, 20, 30])
```

```
num_array1 + num_array2
```

```
array([11, 22, 33])
```

```
num_array2 - num_array1
```

```
array([9, 18, 27])
```

```
num_array1 * num_array2
```

```
array([10, 40, 90])
```

```
num_array2 / num_array1
```

```
array([10, 10, 10])
```

Operations +, -, *, / with multidimensional arrays

```
num_array1 = np.array([
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10],
    [11, 12, 13,14, 15]
])
num_array2 = np.array([
    [10, 20, 30, 40, 50],
    [60, 70, 80, 90, 100],
    [110, 120, 130,140, 150]
])
```

```
num_array1 + num_array2
```

```
array([[ 11,  22,  33,  44,  55],
       [ 66,  77,  88,  99, 110],
       [121, 132, 143, 154, 165]])
```

```
num_array2 / num_array1
```

```
array([[10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10.]])
```

Conditional operations

`>` , `<` , `>=` , `<=` , `==` , `!=`

```
num_array = np.array([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5])
```

```
num_array < 0
```

```
array([True,  True,  True, False, False, False, False])
```

```
num_array[num_array < 0]
```

```
array([-5, -4, -3, -2, -1])
```

Broadcasting

```
num_array = np.array([1, 2, 3])
```

```
num_array * 3
```

```
array([3, 6, 9])
```

```
num_array + 3
```

```
array([4, 5, 6])
```

```
num_list = [1, 2, 3]
```

```
num_list * 3
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Broadcasting with multidimensional arrays

`array2d` $(3 \text{ x } 4)$

```
array2d = np.array([
    [1, 2, 3, 4],
    [1, 2, 3, 4],
    [1, 2, 3, 4]
])
```

`array1d` $(1 \text{ x } 4)$

```
array1d = np.array([1, 2, 3, 4])
```

```
array2d / array1d
```

```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

Broadcasting with multidimensional arrays

`array2d` $(3 \text{ x } 4)$

```
array2d = np.array([
    [1, 2, 3, 4],
    [1, 2, 3, 4],
    [1, 2, 3, 4]
])
```

`array1d` $(3 \text{ x } 1)$

```
array1d = np.array([[1], [2], [3]])
```

```
array2d / array1d
```

```
array([[1.   , 2.   , 3.   , 4.   ],
       [0.5  , 1.   , 1.5  , 2.   ],
       [0.333, 0.667, 1.   , 1.333]])
```

Let's practice

PRACTICING CODING INTERVIEW QUESTIONS IN PYTHON

How to use the .apply() method on a DataFrame?

PRACTICING CODING INTERVIEW QUESTIONS IN PYTHON

Kirill Smirnov
Data Science Consultant, Altran

Dataset

```
import pandas as pd

scores = pd.read_csv('exams.csv')
scores = scores[['math score', 'reading score', 'writing score']]
print(scores.head())
```

```
   math score  reading score  writing score
0          74             86             82
1          44             49             53
2          54             46             43
3          88             95             92
4          85             81             81
```

Default .apply()

`df.apply(function)`

```
print(scores.head())
```

```
   math score  reading score  writing score
0          74             86             82
1          44             49             53
2          54             46             43
3          88             95             92
4          85             81             81
```

```
import numpy as np

scores_new = scores.apply(np.sqrt)
print(score_new)
```

```
   math score  reading score  writing score
0    8.602325       9.273618       9.055385
1    6.633250       7.000000       7.280110
2    7.348469       6.782330       6.557439
3    9.380832       9.746794       9.591663
4    9.219544       9.000000       9.000000
...
```

Default .apply()

`df.apply(function)`

```
print(scores.head())
```

```
   math score  reading score  writing score
0          74             86             82
1          44             49             53
2          54             46             43
3          88             95             92
4          85             81             81
```

```
import numpy as np

scores_new = scores.apply(np.mean)
print(score_new.head())
```

```
math score       65.18
reading score    69.28
writing score    67.96
dtype: float64
```

```
type(scores_new)
```

```
pandas.core.series.Series
```

Default .apply()

`df.apply(function)`

```
print(scores.head())
```

```
   math score  reading score  writing score
0          74             86             82
1          44             49             53
2          54             46             43
3          88             95             92
4          85             81             81
```

`function(pd.Series)`

input size n

$\longrightarrow$ `np.sqrt(pd.Series)`

$\longrightarrow$ output size n

input size n

$\longrightarrow$ `np.mean(pd.Series)`

$\longrightarrow$ single value

Default .apply(): own functions

`df.apply(function)`

```
print(scores.head())
```

```
   math score  reading score  writing score
0          74             86             82
1          44             49             53
2          54             46             43
3          88             95             92
4          85             81             81
```

```
def divide_scores(x):
    return x / 2
```

```
scores_new = scores.apply(divide_scores)
print(scores_new)
```

```
    math score  reading score  writing score
0         37.0           43.0           41.0
1         22.0           24.5           26.5
2         27.0           23.0           21.5
3         44.0           47.5           46.0
4         42.5           40.5           40.5
...
```

Default .apply(): own functions

`df.apply(function)`

```
print(scores.head())
```

```
   math score  reading score  writing score
0          74             86             82
1          44             49             53
2          54             46             43
3          88             95             92
4          85             81             81
```

```
def perfect_score(x):
    return 100
```

```
scores_new = scores.apply(perfect_score)
print(scores_new)
```

```
math score      100
reading score   100
writing score   100
dtype: int64
```

Lambda expressions

```
def divide_scores(x):
    return x / 2
```

```
scores_new = scores.apply(divide_scores)
print(scores_new)
```

```
    math score  reading score  writing score
0         37.0           43.0           41.0
1         22.0           24.5           26.5
2         27.0           23.0           21.5
3         44.0           47.5           46.0
4         42.5           40.5           40.5
...
```

```
def perfect_score(x):
    return 100
```

```
scores_new = scores.apply(perfect_score)
print(scores_new)
```

```
math score       100
reading score    100
writing score    100
dtype: int64
```

Lambda expressions

```
scores_new = scores.apply(lambda x: x / 2)
print(scores_new)
```

```
    math score  reading score  writing score
0         37.0           43.0           41.0
1         22.0           24.5           26.5
2         27.0           23.0           21.5
3         44.0           47.5           46.0
4         42.5           40.5           40.5
...
```

```
scores_new = scores.apply(lambda x: 100)
print(scores_new)
```

```
math score       100
reading score    100
writing score    100
dtype: int64
```

Additional arguments: axis

```
df.apply(function, axis= )
```

Additional arguments: axis

```
df.apply(function, axis=0)
```

Additional arguments: axis

```
df.apply(function, axis=1)
```

Additional arguments: axis

`df.apply(function, axis= )`

`axis=0` - `function` is applied over columns

`axis=1` - `function` is applied over rows

```
print(scores.head())
```

```
   math score  reading score  writing score
0          74             86             82
1          44             49             53
2          54             46             43
3          88             95             92
4          85             81             81
```

```
import numpy as np

scores_new = scores.apply(np.mean)
print(score_new.head())
```

```
math score       65.18
reading score    69.28
writing score    67.96
dtype: float64
```

Additional arguments: axis

`df.apply(function, axis= )`

`axis=0` - `function` is applied over columns

`axis=1` - `function` is applied over rows

```
print(scores.head())
```

```
   math score  reading score  writing score
0          74             86             82
1          44             49             53
2          54             46             43
3          88             95             92
4          85             81             81
```

```
import numpy as np

scores_new = scores.apply(np.mean, axis=0)
print(score_new.head())
```

```
math score       65.18
reading score    69.28
writing score    67.96
dtype: float64
```

Additional arguments: axis

`df.apply(function, axis= )`

`axis=0` - `function` is applied over columns

`axis=1` - `function` is applied over rows

```
print(scores.head())
```

```
   math score  reading score  writing score
0          74             86             82
1          44             49             53
2          54             46             43
3          88             95             92
4          85             81             81
```

```
import numpy as np

scores_new = scores.apply(np.mean, axis=1)
print(score_new.head())
```

```
0     80.666667
1     48.666667
2     47.666667
3     91.666667
4     82.333333
5     84.000000
6     75.000000
7     70.666667
...
```

Additional arguments: result_type

```
df.apply(function, result_type= )
```

```
result_type='expand'
```

```
print(scores.head())
```

	math score	reading score	writing score
0	74	86	82
1	44	49	53
2	54	46	43
3	88	95	92
4	85	81	81

```
import numpy

def span(x):
    return [np.min(x), np.max(x)]
```

```
scores_new = scores.apply(span)
print(scores_new)
```

```
math score       [27, 100]
reading score    [33, 100]
writing score    [30, 100]
dtype: object
```

Additional arguments: result_type

```
df.apply(function, result_type= )
```

```
result_type='expand'
```

```
print(scores.head())
```

	math score	reading score	writing score
0	74	86	82
1	44	49	53
2	54	46	43
3	88	95	92
4	85	81	81

```
import numpy

def span(x):
    return [np.min(x), np.max(x)]

scores.apply(span, result_type='expand')
```

	math score	reading score	writing score
0	27	33	30
1	100	100	100

Additional arguments: result_type

`df.apply(function, result_type= )`

`result_type='expand'`

```
print(scores.head())
```

	math score	reading score	writing score
0	74	86	82
1	44	49	53
2	54	46	43
3	88	95	92
4	85	81	81

```
import numpy

def span(x):
    return [np.min(x), np.max(x)]

scores.apply(span, result_type='expand', axis=1)
```

	0	1
0	74	86
1	44	53
2	43	54
3	88	95
4	81	85
...		

Additional arguments: result_type

```
df.apply(function, result_type= )
```

```
result_type='broadcast'
```

```
print(scores.head())
```

	math score	reading score	writing score
0	74	86	82
1	44	49	53
2	54	46	43
3	88	95	92
4	85	81	81

```
import numpy as np

scores_new = scores.apply(np.mean)
print(score_new.head())
```

```
math score       65.18
reading score    69.28
writing score    67.96
dtype: float64
```

Additional arguments: result_type

```
df.apply(function, result_type= )
```

```
result_type='broadcast'
```

```
print(scores.head())
```

	math score	reading score	writing score
0	74	86	82
1	44	49	53
2	54	46	43
3	88	95	92
4	85	81	81

```
import numpy as np

scores.apply(np.mean, result_type='broadcast')
```

	math score	reading score	writing score
0	65	69	67
1	65	69	67
2	65	69	67
3	65	69	67
4	65	69	67
5	65	69	67
6	65	69	67
7	65	69	67
...			

More than one argument in a function

```
function(pd.Series)
```

More than one argument in a function

```
function(pd.Series, arg1, arg2, ..., kwarg1=val1, kwarg2=val2, ...)
```

```
def check_mean(x, a, b, inside=True):
    mean = np.mean(x)
    if inside:
        return mean > a and mean < b
    else:
        return mean < a or mean > b
```

Applying the function

```
print(scores.head())
```

```
   math score  reading score  writing score
0          74             86             82
1          44             49             53
2          54             46             43
3          88             95             92
4          85             81             81
```

```
import numpy as np

scores.apply(check_mean)
```

```
TypeError
```

Additional arguments: args

`df.apply(function, args= )`

`args` - `[arg1, arg2, ...]`

```
print(scores.head())
```

	math score	reading score	writing score
0	74	86	82
1	44	49	53
2	54	46	43
3	88	95	92
4	85	81	81

```
import numpy as np

scores.apply(check_mean, args=[67, 70])
```

```
math score       False
reading score     True
writing score     True
dtype: bool
```

Additional arguments: args

`df.apply(function, args= )`

`args` - `(arg1, arg2, ...)`

```
print(scores.head())
```

	math score	reading score	writing score
0	74	86	82
1	44	49	53
2	54	46	43
3	88	95	92
4	85	81	81

```
import numpy as np

scores.apply(
    check_mean, args=[67, 70], inside=False
)
```

```
math score        True
reading score    False
writing score    False
dtype: bool
```

Let's practice!

PRACTICING CODING INTERVIEW QUESTIONS IN PYTHON

How to use the .groupby() method on a DataFrame?

PRACTICING CODING INTERVIEW QUESTIONS IN PYTHON

Kirill Smirnov
Data Science Consultant, Altran

Dataset

```
retinol = pd.read_csv('retinol.csv')
retinol = retinol[['age','gender','smoking','bmi','vitamin use','plasma B-carotene','plasma retinol']]
print(retinol.head())
```

```
   age  gender smoking       bmi       vitamin use  plasma B-carotene  plasma retinol
0   64  Female  Former  21.48380  Yes_fairly_often                200             915
1   76  Female   Never  23.87631  Yes_fairly_often                124             727
2   38  Female  Former  20.01080     Yes_not_often                328             721
3   40  Female  Former  25.14062                No                153             615
4   72  Female   Never  20.98504  Yes_fairly_often                 92             799
```

background factors → `plasma B-carotene`, `plasma retinol`

.groupby()

groups the data according to some criteria allowing to perform an operation on each group.

`df.groupby(column_name(s))`

```
gens = retinol.groupby('gender')
print(gens)
```

```
<pandas.core.groupby.groupby.DataFrameGroupBy object at 0x00000262DB5E2780>
```

```
gensmoks = retinol.groupby(['gender', 'smoking'])
print(gensmoks)
```

```
<pandas.core.groupby.groupby.DataFrameGroupBy object at 0x00000262DB5F57B8>
```

Iterating through .groupby() output

```
gens = retinol.groupby('gender')

for group in gens:
    # Each group is a tuple
    # First element is a grouping factor
    print(group[0])
    # Second element is a DataFrame
    print(group[1].head(3))
```

```
Female
   age  gender smoking       bmi  ...
0   64  Female  Former  21.48380  ...
1   76  Female   Never  23.87631  ...
2   38  Female  Former  20.01080  ...
Male
    age gender smoking       bmi  ...
12   57   Male   Never  31.73039  ...
14   66   Male   Never  27.31916  ...
15   64   Male  Former  31.44674  ...
```

```
len(gens)
```

```
2
```

Iterating through .groupby() output

```
gensmoks = retinol.groupby(['gender', 'smoking'])

for group in gensmoks:
    # Each group is a tuple
    # First element is a grouping factor
    print(group[0])
    # Second element is a DataFrame
    print(group[1].head(3))
```

```
len(gensmoks)
```

```
6
```

```
('Female', 'Current_Smoker')
    age  gender         smoking       bmi ...
32   74  Female  Current_Smoker  16.33114 ...
35   44  Female  Current_Smoker  25.87867 ...
43   31  Female  Current_Smoker  23.34593 ...
('Female', 'Former')
   age  gender smoking       bmi ...
0   64  Female  Former  21.48380 ...
2   38  Female  Former  20.01080 ...
3   40  Female  Former  25.14062 ...
('Female', 'Never')
   age  gender smoking       bmi ...
1   76  Female   Never  23.87631 ...
4   72  Female   Never  20.98504 ...
...
```

Standard operations on groups

```
gens = retinol.groupby('gender')
```

```
retinol['plasma retinol'].mean()
```

```
602.790476
```

```
retinol['vitamin use'].count()
```

```
315
```

```
gens['plasma retinol'].mean()
```

```
        plasma retinol
gender
Female      587.721612
Male        700.738095
```

```
gens['vitamin use'].count()
```

```
        vitamin use
gender
Female          273
Male             42
```

The .agg() method

`.agg(function, axis= , args= )` - almost identical to the `.apply()` method

```
import numpy as np

retinol['plasma retinol'].agg(np.mean)
```

```
602.790476
```

The .agg() method

`.agg(function, axis= , args= )` - almost identical to the `.apply()` method

```
import numpy as np


retinol[['plasma B-carotene', 'plasma retinol']].agg(np.mean)
```

```
plasma B-carotene    189.892063
plasma retinol       602.790476
dtype: float64
```

The .agg() method

`.agg(function, axis= , args= )` - almost identical to the `.apply()` method

```
import numpy as np


retinol[['plasma B-carotene', 'plasma retinol']].agg([np.mean, np.std])
```

```
      plasma B-carotene  plasma retinol
mean          189.892063      602.790476
std           183.000803      208.895474
```

.groupby() followed by .agg()

```
gens = retinol.groupby('gender')
```

```
gens['plasma retinol'].agg([np.mean, np.std])
```

```
       plasma retinol
                 mean         std
gender
Female     587.721612  185.430687
Male       700.738095  307.808783
```

```
gensmoks = retinol.groupby(['gender', 'smoking'])
```

```
gensmoks['plasma retinol'].agg([np.mean, np.std])
```

```
                       plasma retinol
                                 mean         std
gender smoking
Female Current_Smoker     556.111111  191.112649
       Former             607.752688  187.983733
       Never              582.687500  182.182398
Male   Current_Smoker     598.857143  289.618961
       Former             798.500000  323.196203
       Never              590.153846  249.307991
```

Own functions and lambda expressions

```
gens = retinol.groupby('gender')
```

```
def n_more_than_mean(series):
    result = series[series > np.mean(series)]
    return len(result)
```

```
gens[['plasma B-carotene', 'retinol']].agg(n_more_than_mean)
```

```
        plasma B-carotene  plasma retinol
gender
Female                 87             119
Male                   13              19
```

Own functions and lambda expressions

```
gens = retinol.groupby('gender')
```

```
def n_more_than_mean(series):
    result = series[series > np.mean(series)]
    return len(result)
```

```
gens[['plasma B-carotene', 'plasma retinol']].agg([n_more_than_mean, lambda x: len(x)])
```

	plasma B-carotene		plasma retinol	
	count_more_than_mean	<lambda>	count_more_than_mean	<lambda>
gender				
Female	87	273	119	273
Male	13	42	19	42

Renaming the output

```
gens = retinol.groupby('gender')
```

```
def n_more_than_mean(series):
    result = series[series > np.mean(series)]
    return len(result)
```

```
gens[['plasma B-carotene', 'plasma retinol']].agg({'count': n_more_than_mean, 'len': lambda x: len(x)})
```

	count		len	
	plasma B-carotene	plasma retinol	plasma B-carotene	plasma retinol
gender				
Female	87	119	273	273
Male	13	19	42	42

The .transform() method

`.transform(function, axis= , args= )` - almost identical to the `.apply()` method

- The input and output must have the same size

```
import numpy as np

def center_scale(series):
    return (series - np.mean(series))/np.std(series)
```

DataFrame and the .transform() method

```
compounds = ['plasma B-carotene', 'retinol']
df = retinol[compounds].transform(center_scale)

print(df)
```

```
     plasma B-carotene  plasma retinol
0             0.055322        1.496951
1            -0.360637        0.595547
2             0.755886        0.566779
3            -0.201916        0.058541
4            -0.535778        0.940766
5            -0.229282        0.245534
6             0.372765        1.108580
...
309          -0.251174        0.715415
310          -0.141711       -1.854544
311          -0.601456       -1.317538
312           0.602637       -0.483260
313          -0.377057        0.389375
314           0.235936        1.070223
```

.groupby() followed by .transform()

```
gensmoks = retinol.groupby(['gender', 'smoking'])
```

```
compounds = ['plasma B-carotene', 'retinol']
df = gensmoks[compounds].transform(center_scale)

print(df)
```

```
     plasma B-carotene  plasma retinol
0            -0.018568        1.643294
1            -0.436191        0.794897
2             0.629616        0.605697
3            -0.256573        0.038762
4            -0.597427        1.191485
5            -0.281892        0.247351
6             0.238985        1.384270
...
309          -0.302148        0.771498
310          -0.200869       -2.095267
311          -0.657891       -1.402860
312           0.450607       -0.444440
313          -0.418619        0.407804
314           0.113019        1.340205
```

.groupby() followed by .transform()

```
gensmoks = retinol.groupby(['gender', 'smoking'])
```

```
compounds = ['plasma B-carotene', 'retinol']
df = gensmoks[compounds].transform(
    lambda x: (x - np.mean(x))/np.std(x)
)

print(df)
```

```
     plasma B-carotene  plasma retinol
0            -0.018568        1.643294
1            -0.436191        0.794897
2             0.629616        0.605697
3            -0.256573        0.038762
4            -0.597427        1.191485
5            -0.281892        0.247351
6             0.238985        1.384270
...
309          -0.302148        0.771498
310          -0.200869       -2.095267
311          -0.657891       -1.402860
312           0.450607       -0.444440
313          -0.418619        0.407804
314           0.113019        1.340205
```

The .filter() method of DataFrameGroupBy object

`.filter(function)`

`function` → `True` - group stays

`function` → `False` - group leaves

`function(pd.DataFrame)` - the function acts on the whole DataFrame in each group.

.groupby() followed by .filter()

```
gensmoks = retinol.groupby(['gender', 'smoking'])
len(gensmoks)
```

```
6
```

```
def check_bmi(dataframe):
    return np.mean(dataframe['bmi']) > 26
```

```
retinol_filtered = gensmoks.filter(check_bmi)
print(retinol_filtered)
```

```
     age  gender          smoking       bmi ...
1     76  Female            Never  23.87631 ...
4     72  Female            Never  20.98504 ...
6     65  Female            Never  22.01154 ...
7     58  Female            Never  28.75702 ...
8     35  Female            Never  23.07662 ...
11    40  Female            Never  36.43161 ...
13    66  Female            Never  21.78854 ...
...
299   47  Female            Never  37.27761 ...
302   41  Female            Never  34.61493 ...
306   66  Female            Never  33.10759 ...
311   45  Female            Never  23.82703 ...
312   49  Female            Never  24.26126 ...
314   45  Female            Never  26.50808 ...
```

.groupby() followed by .filter()

```
gensmoks = retinol.groupby(['gender', 'smoking'])
len(gensmoks)
```

```
6
```

```
def check_bmi(dataframe):
    return np.mean(dataframe['bmi']) > 26
```

```
retinol_filtered = gensmoks.filter(check_bmi)
len(retinol_filtered.groupby(['gender', 'smoking']))
```

```
3
```

Let's practice!

PRACTICING CODING INTERVIEW QUESTIONS IN PYTHON

How to visualize data in Python?

PRACTICING CODING INTERVIEW QUESTIONS IN PYTHON

Kirill Smirnov
Data Science Consultant, Altran

datacamp

matplotlib

```
import matplotlib.pyplot as plt
```

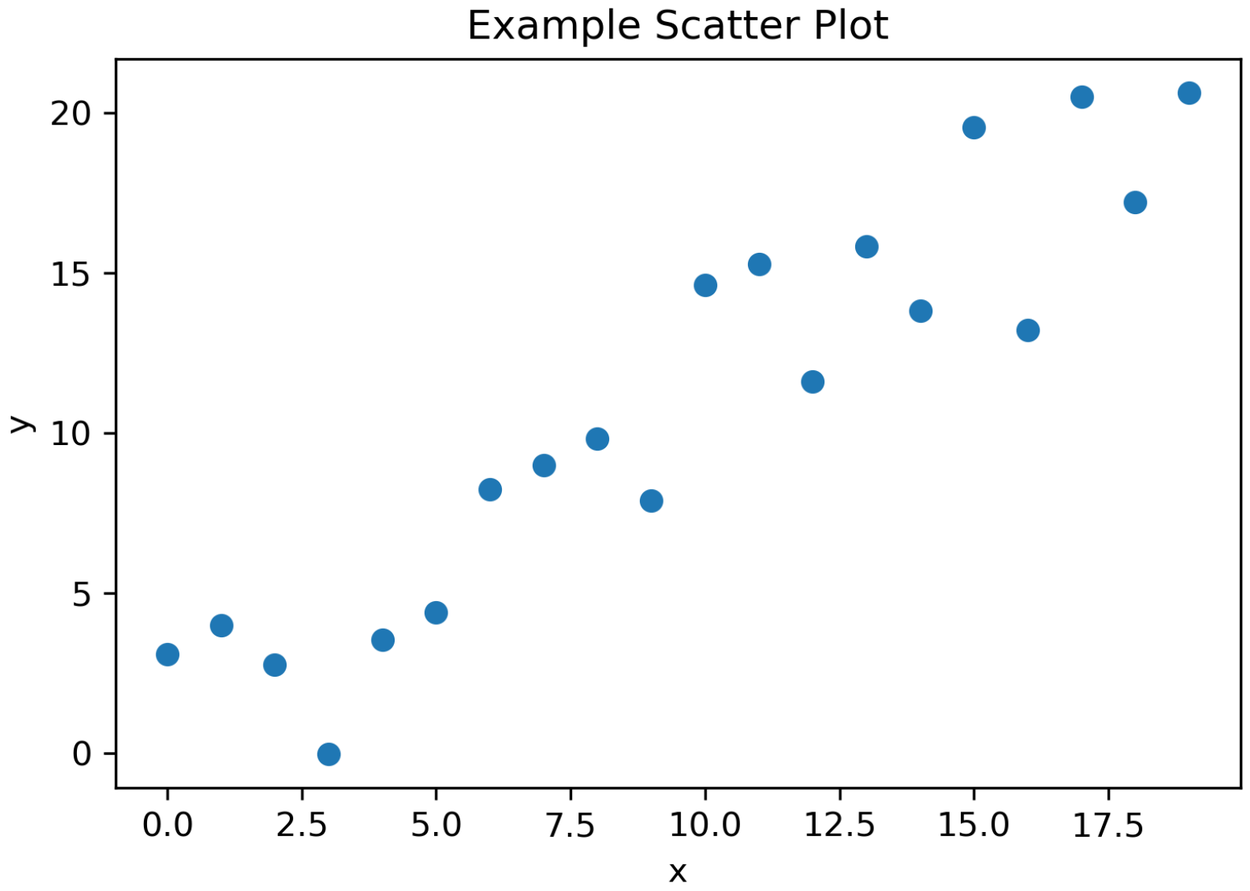
- scatter plot
- histogram
- boxplot

Dataset

```
import pandas as pd

diabetes = pd.read_csv('diabetes.csv')
diabetes = diabetes[[
    'n pregnant', 'plasma glucose', 'blood pressure', 'skin thickness',
    'serum insulin', 'bmi', 'age', 'test result']]
print(diabetes.head())
```

```
   n pregnant  plasma glucose  blood pressure  skin thickness  serum insulin   bmi  age test result
0           6           148.0            72.0            35.0            NaN  33.6   50    positive
1           1            85.0            66.0            29.0            NaN  26.6   31    negative
2           8           183.0            64.0             NaN            NaN  23.3   32    positive
3           1            89.0            66.0            23.0           94.0  28.1   21    negative
4           0           137.0            40.0            35.0          168.0  43.1   33    positive
```

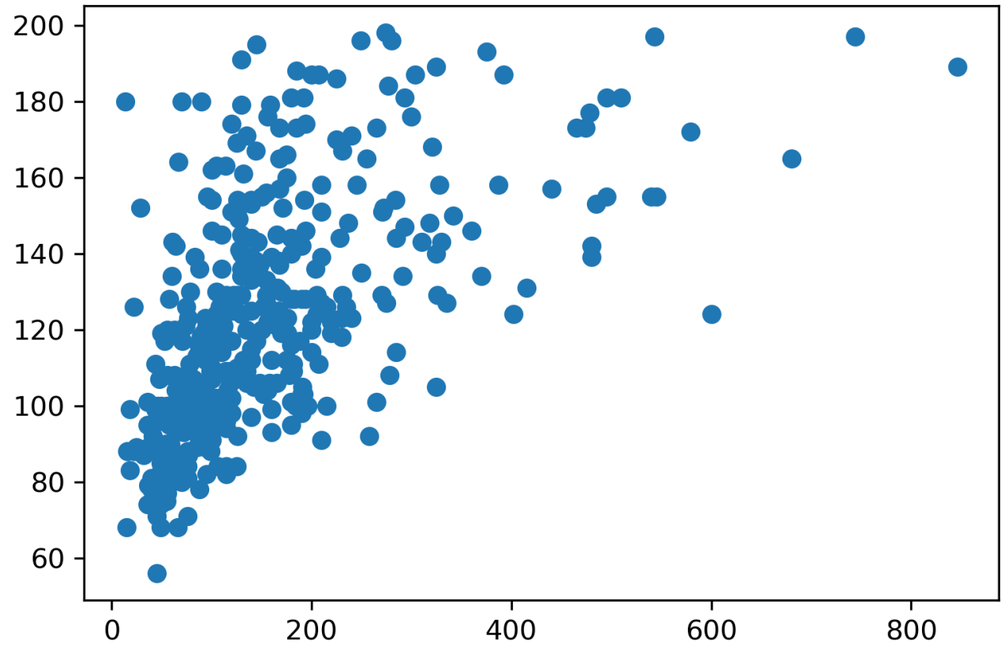
Scatter plot



Create a scatter plot

```
import matplotlib.pyplot as plt
```

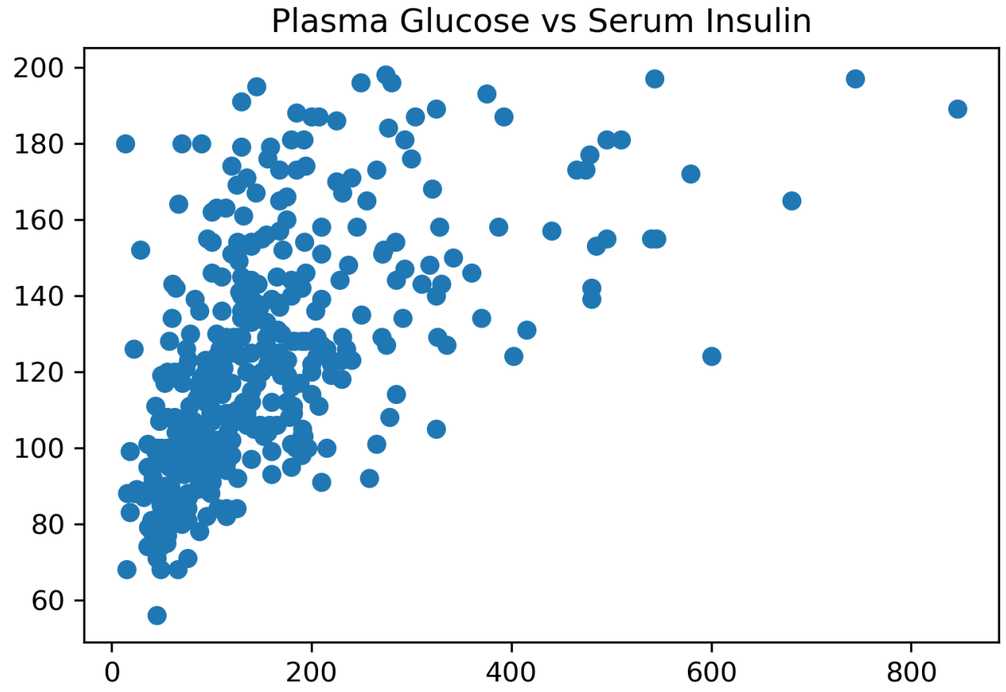
```
plt.scatter(
    diabetes['serum insulin'],
    diabetes['plasma glucose']
)
plt.show()
```



Create a scatter plot

```
import matplotlib.pyplot as plt
```

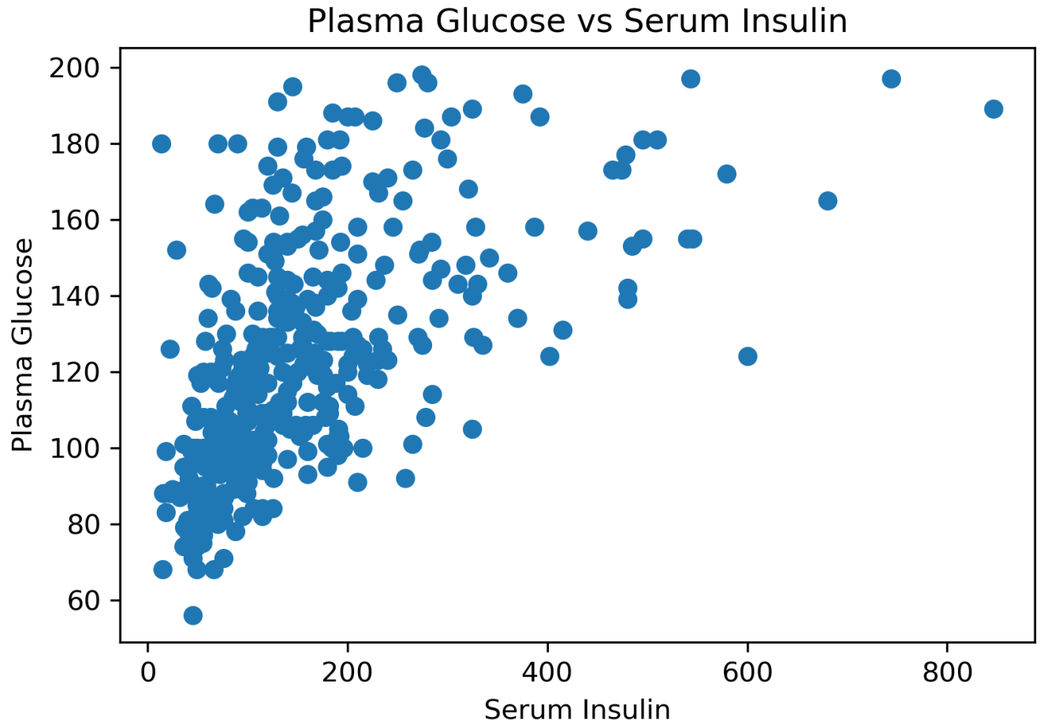
```
plt.scatter(
    diabetes['serum insulin'],
    diabetes['plasma glucose']
)

plt.title('Plasma Glucose vs Serum Insulin')

plt.show()
```



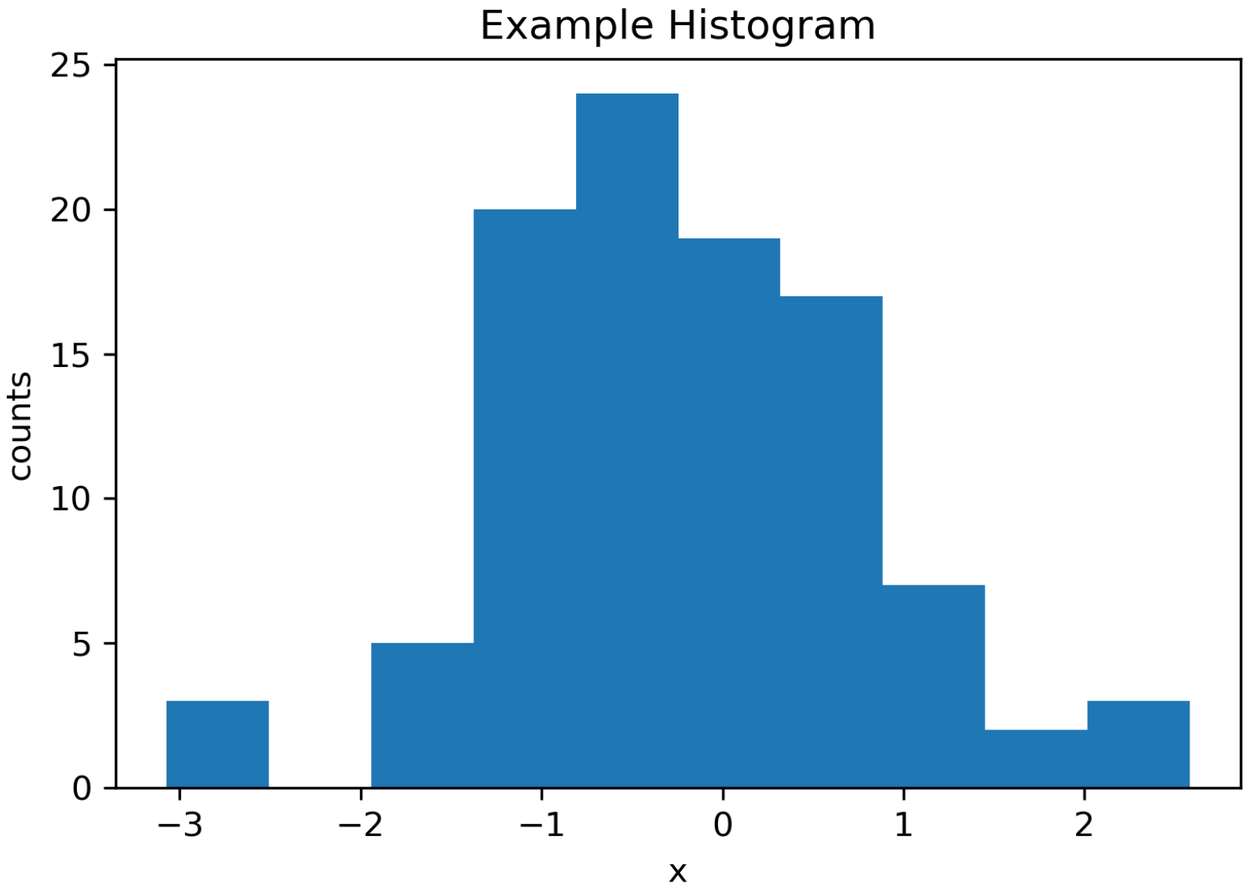
Create a scatter plot

```
import matplotlib.pyplot as plt
```

```
plt.scatter(
    diabetes['serum insulin'],
    diabetes['plasma glucose']
)

plt.title('Plasma Glucose vs Serum Insulin')

plt.xlabel('Serum Insulin')
plt.ylabel('Plasma Glucose')

plt.show()
```



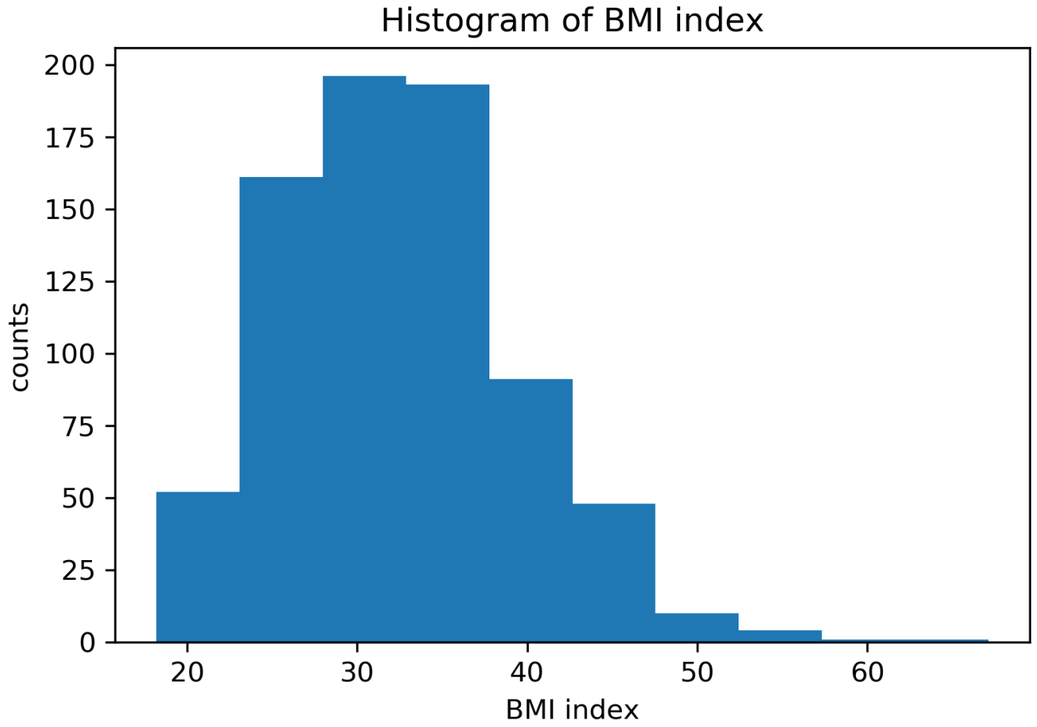
Histogram



Create a histogram

```
import matplotlib.pyplot as plt
```

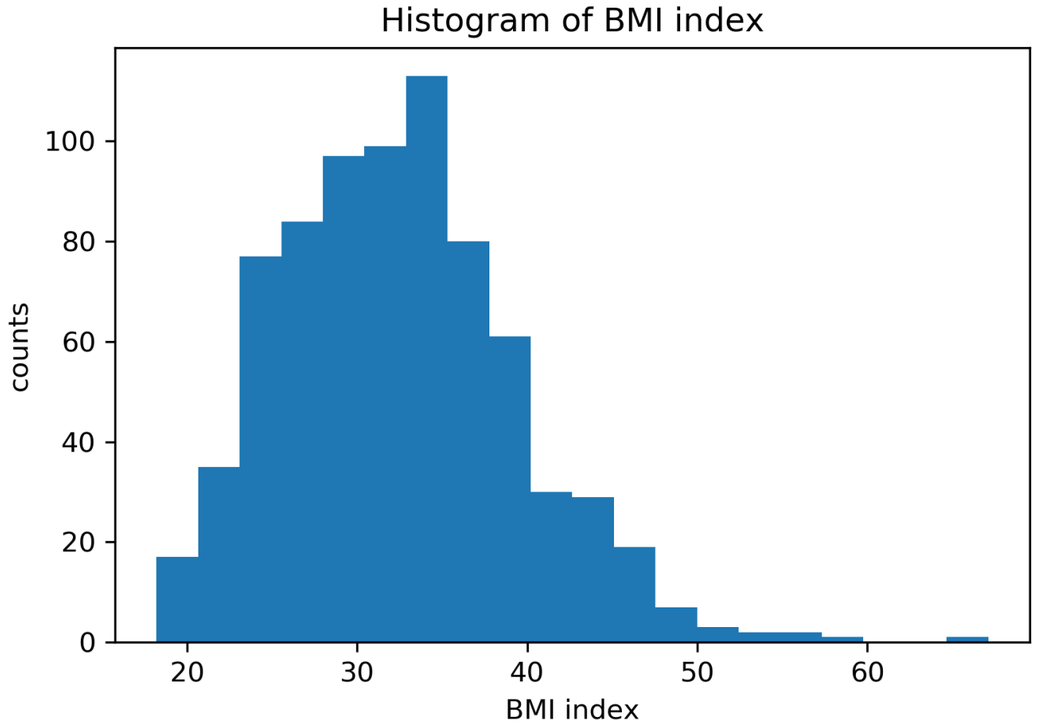
```
plt.hist(diabetes['bmi'])

plt.title('Histogram of BMI index')

plt.xlabel('BMI index')
plt.ylabel('counts')

plt.show()
```



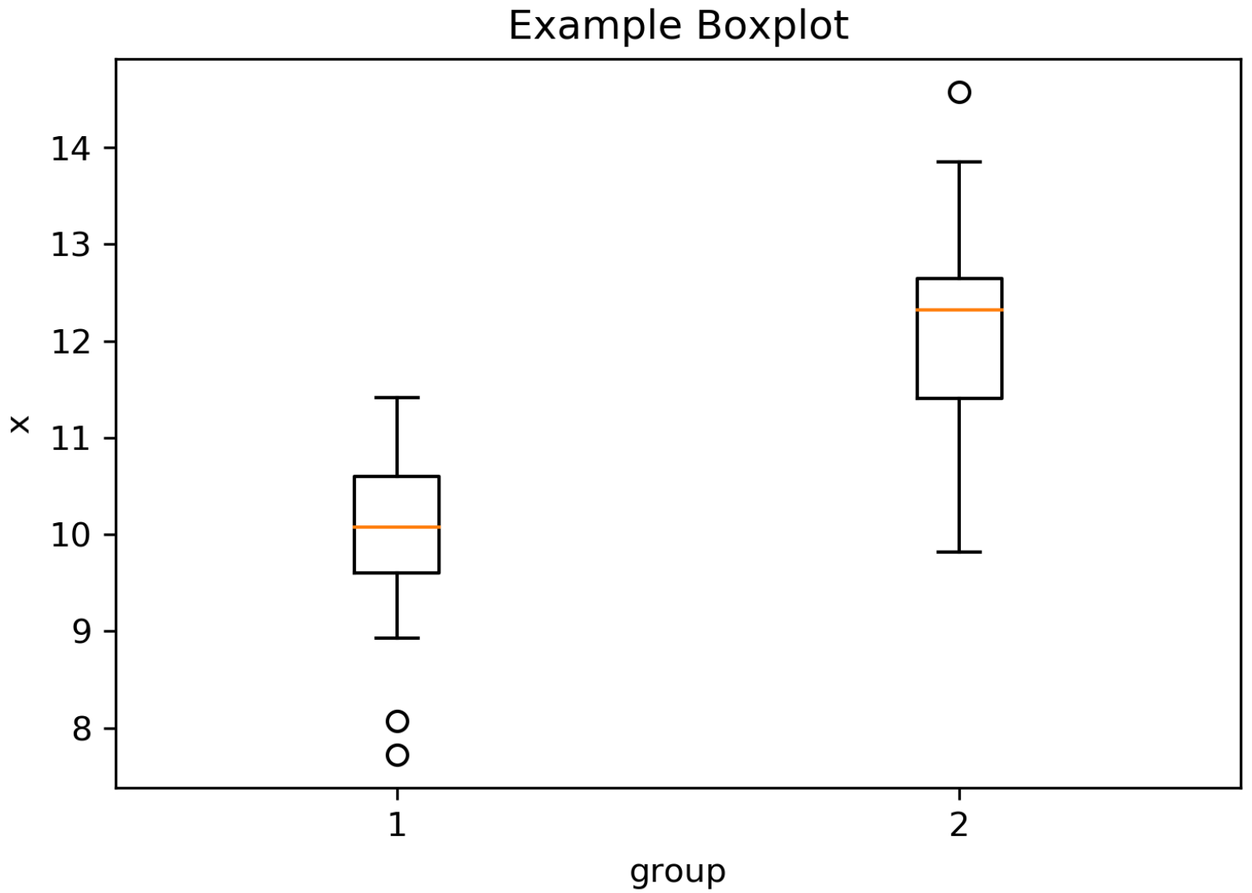
Create a histogram

```
import matplotlib.pyplot as plt
```

```
plt.hist(diabetes['bmi'], bins=20)

plt.title('Histogram of BMI index')

plt.xlabel('BMI index')
plt.ylabel('counts')

plt.show()
```



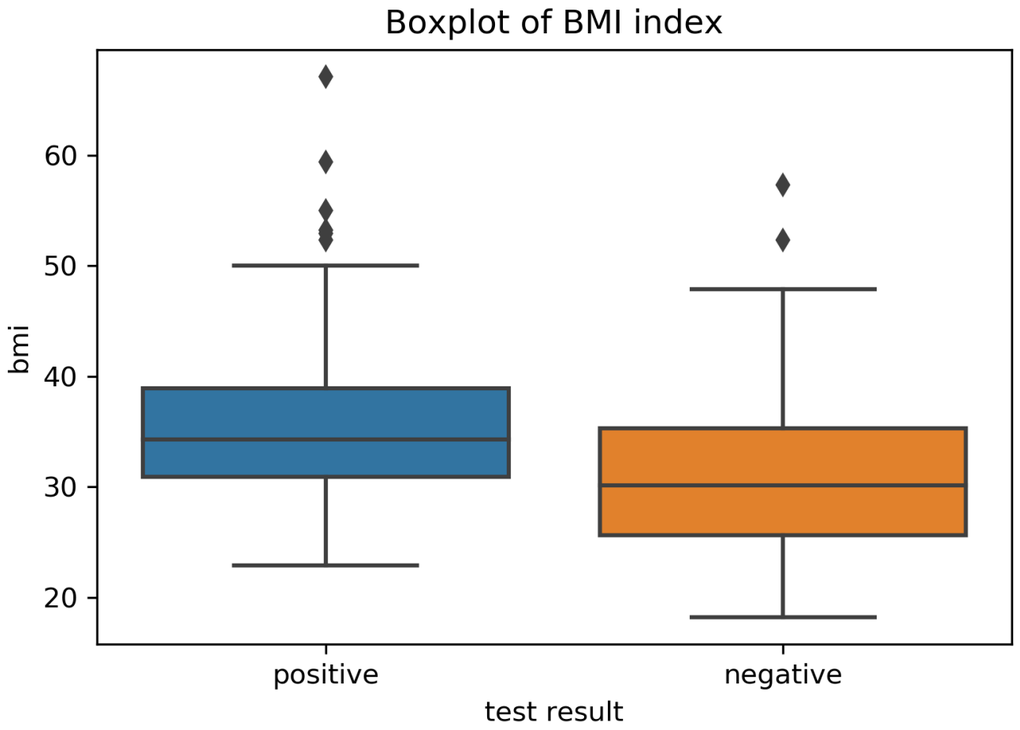
Boxplot



Create a boxplot

```
import seaborn as sns
```

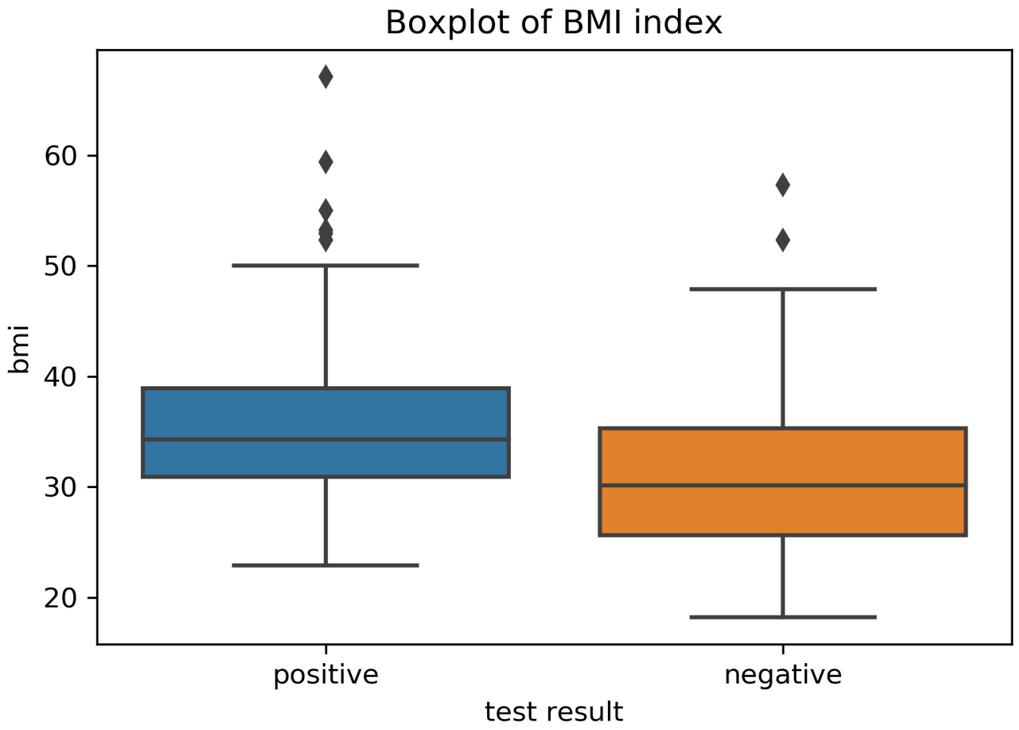
```
sns.boxplot('test_result', 'bmi', data=diabetes)
plt.title('Boxplot of BMI index')

plt.show()
```



Create a boxplot

```
import seaborn as sns
```

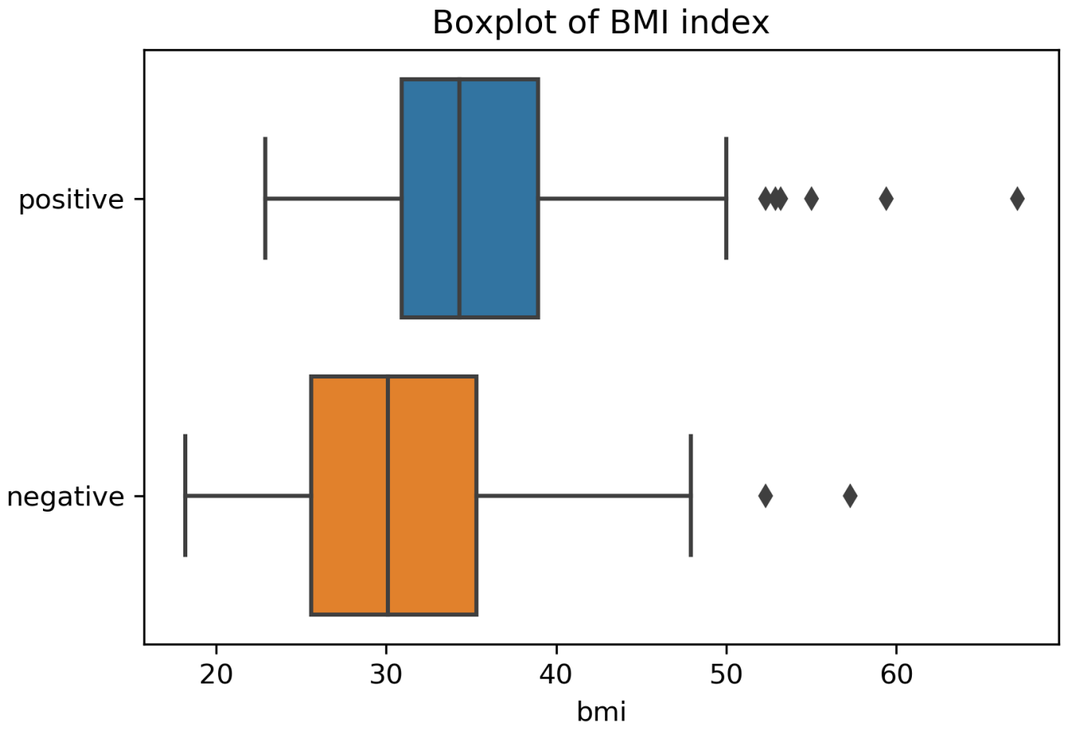
```
sns.boxplot(
    x='test_result',
    y='bmi',
    data=diabetes
)
plt.title('Boxplot of BMI index')

plt.show()
```



Create a boxplot

```
import seaborn as sns
```

```
sns.boxplot(
    y='test_result',
    x='bmi',
    data=diabetes
)
plt.title('Boxplot of BMI index')

plt.show()
```



Let's practice!

PRACTICING CODING INTERVIEW QUESTIONS IN PYTHON

Final thoughts

PRACTICING CODING INTERVIEW QUESTIONS IN PYTHON

Kirill Smirnov
Data Science Consultant, Altran

datacamp

Topics covered

- main data structures in Python
- string manipulation techniques
- iterable objects and their definition
- functions in Python
- NumPy arrays
- operations on DataFrames
- data visualization

Good luck!

PRACTICING CODING INTERVIEW QUESTIONS IN PYTHON