NUMPY

- Chúng ta sẽ sử dụng numpy để làm việc với bộ dữ liệu taxi của new York
- Cấu trúc trong Numpy là mảng ndarray hoặc n-chiều array

- Để sử dụng Num py, ta dùng cú pháp sau

```
1 import numpy as np
```

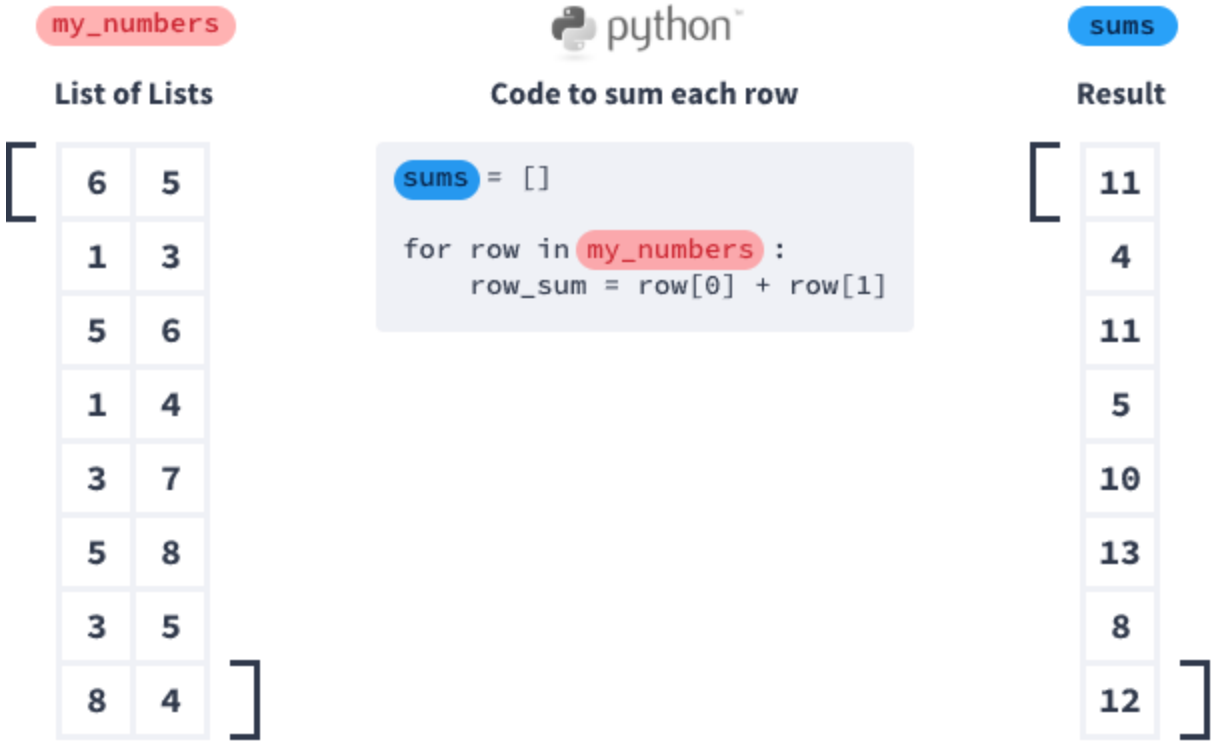
- Sau đó ta sẽ chuyển đổi trực tiếp từ 1 list thành ndarray với numpy.array().Ví dụ.
-

```
3 data_ndarray = np.array([5, 10, 15, 20])
```

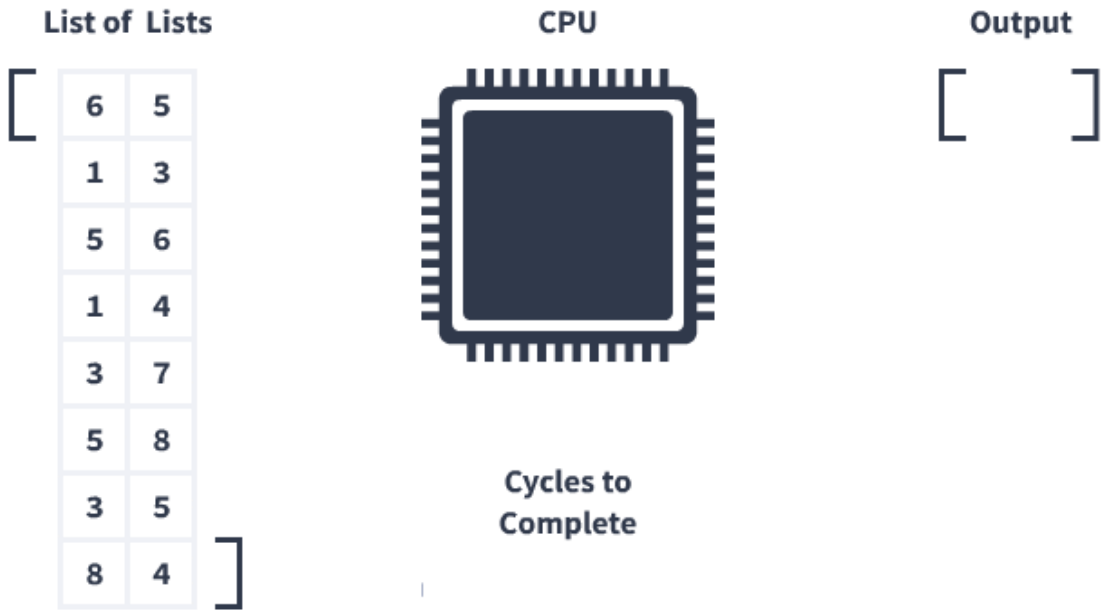
-
- Task 1:

1. Import `numpy`, and assign it to the alias `np`.
2. Create a NumPy ndarray from the list `[10, 20, 30]`. Assign the result to the variable `data_ndarray`.
3. Click "Run Code" to run your code and get feedback.
4. Click "Submit Answer" to check your answer.

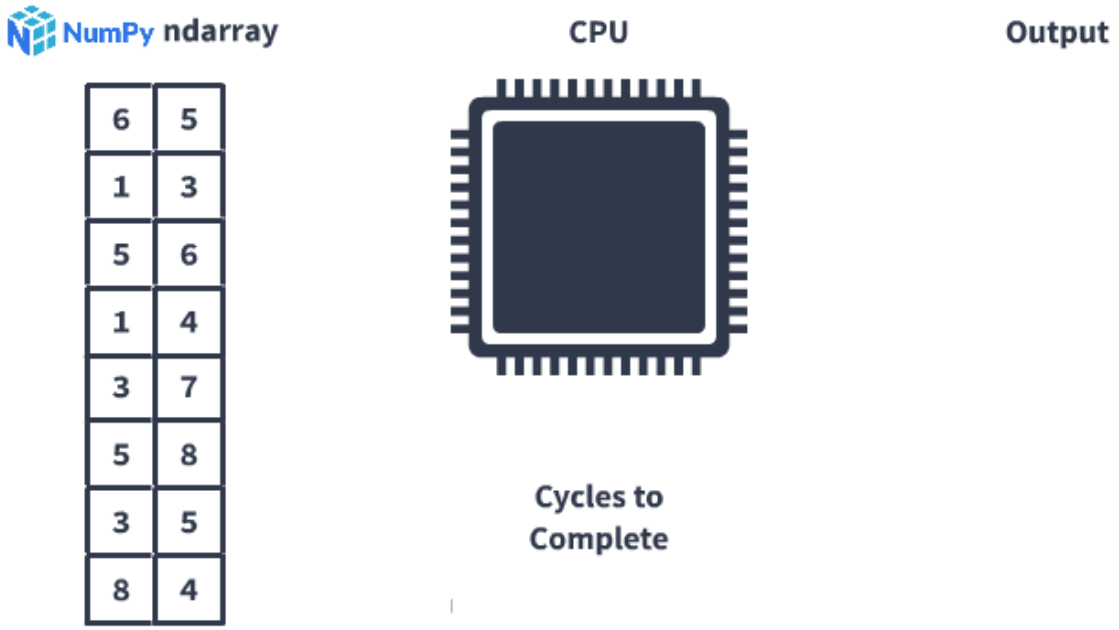
- Trước đấy, chúng ta đã nói về list của list, Khi làm việc với các tập dữ liệu cỡ 10k data không phải là vấn đề, Nhưng khi dữ liệu đến vài triệu nó xử lý rất kém, do tính chất của list perf không tốt. Nhất là khi truy cập với vòng lặp for



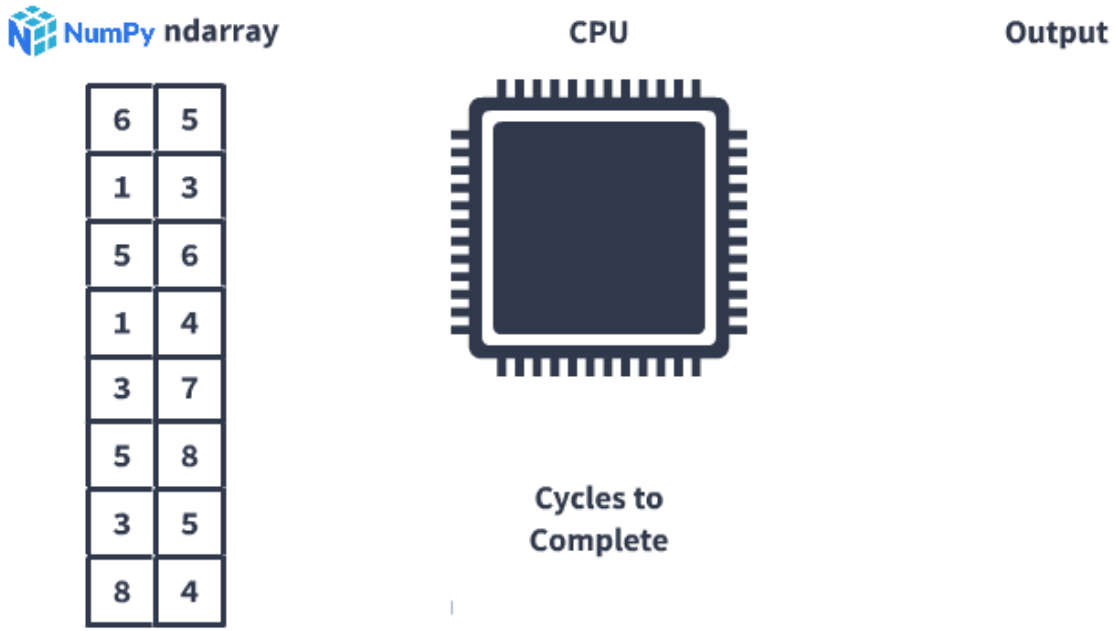
- Trong vòng lặp for, python biến mã thành mã bytecode và bytecode yêu cầu cpu cộng 2 số lại với nhau.



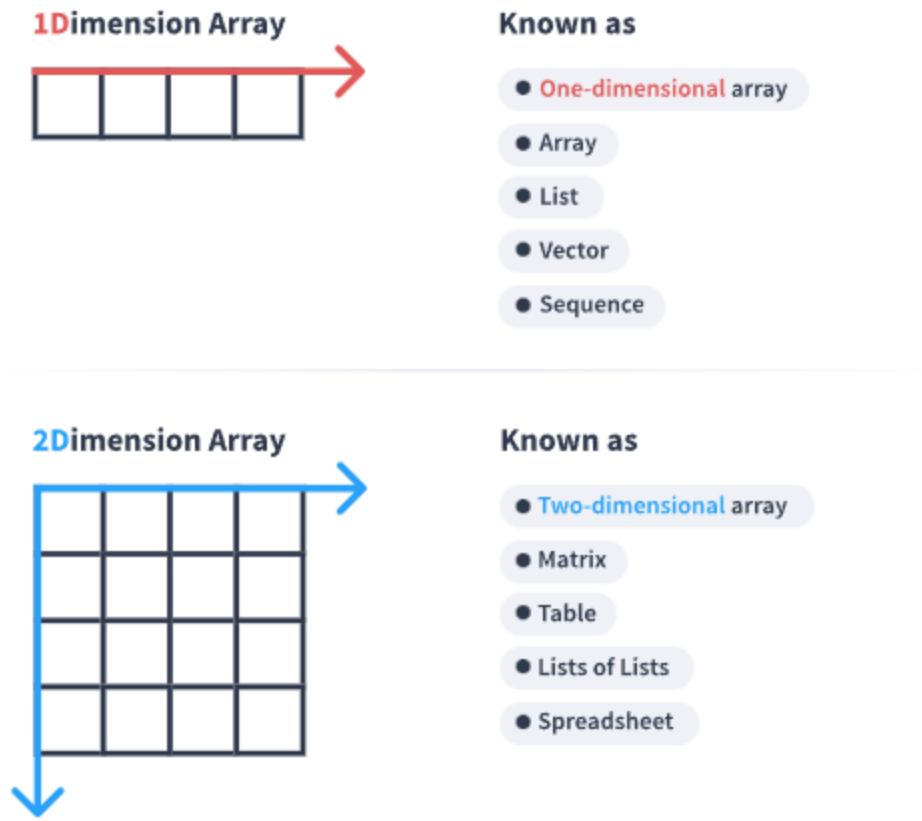
- Với numpy thì khác, nó tận dụng tính năng được gọi là SIMD(**Single Instruction Multiple Data) .** SIMD cho phép một CPUthực hiện cùng một hoạt động trên nhiều điểm dữ liệu trong một chu kỳ bộ xử lý:



- Với numpy thì khác, nó tận dụng tính năng được gọi là SIMD(**Single Instruction Multiple Data) .** SIMD cho phép một CPUthực hiện cùng một hoạt động trên nhiều điểm dữ liệu trong một chu kỳ bộ xử lý



- Ở dưới là 2 mảng 2 chiều, vector 2 chiều,....



- Ta sẽ đánh giá bộ dữ liệu taxi của NY city

- Task1: sử dụng bộ dữ liệu taxi CSV convert thành NumPy ndrrray,
- Task1.1: Bộ dữ liệu sau khi chuyển đổi được lưu lại dưới tên biến là: taxi
- Cú pháp gợi ý

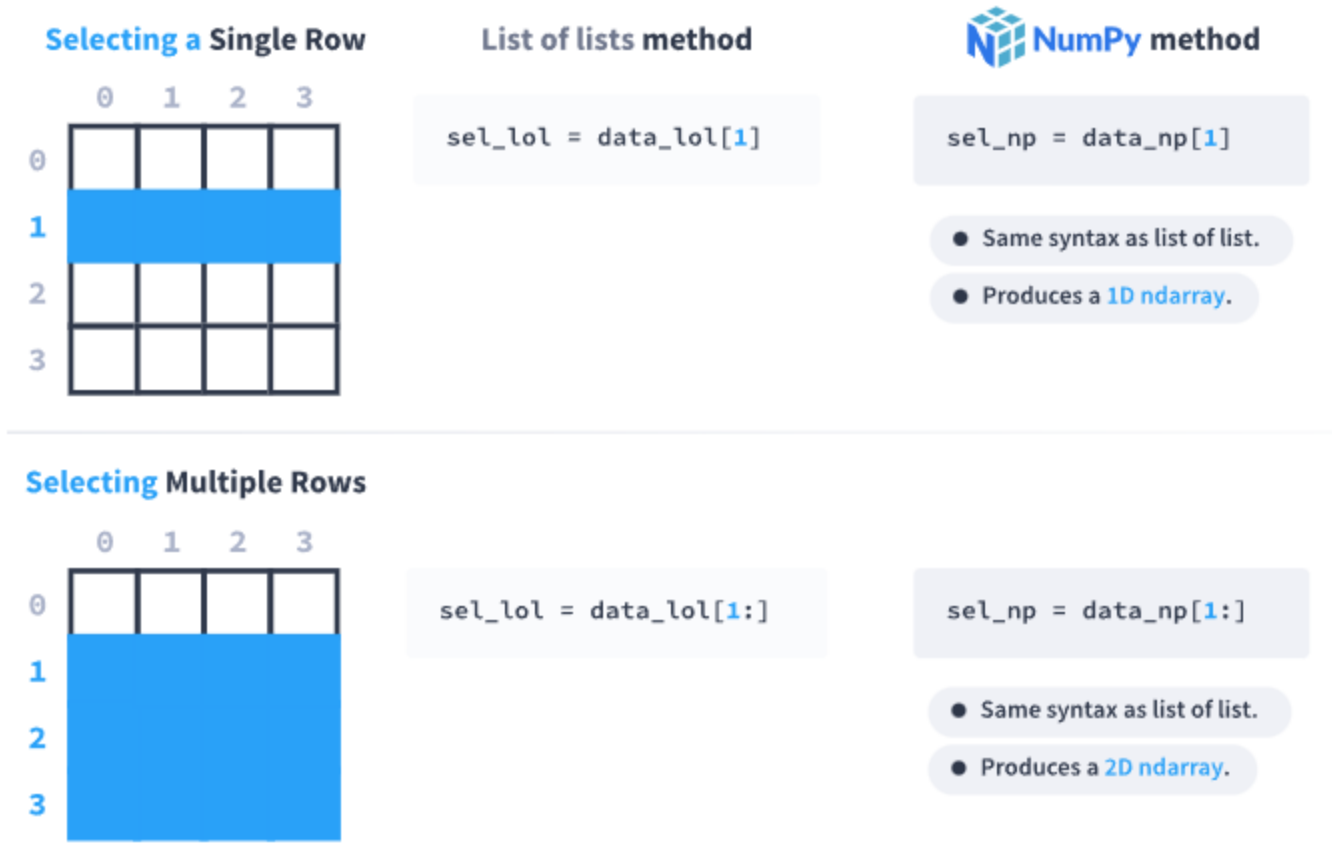
```
# our list of lists is stored as data_list
data_ndarray = np.array(data_list)
```

- Trong numpy có 1 cú pháp là ndarray.shape. Hãy giải thích nó sau khi hoàn thành đoạn code sau:

```
import numpy as np

arr2= np.array([[4, 2, 3, 2, 1, 8],
                [5, 4,6,7,8,9]])
res = np.shape(arr2)

print(res)
```

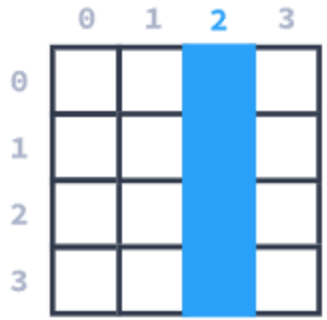
- Áp dụng vào bài taxi, gõ taxi.shape. Giải thích con số có nghĩa của kết quả in ra

- Lựa chọn hang or côt theo list of list



- Ta cũng có thể lấy như sau

Selecting a Single Column



List of lists method

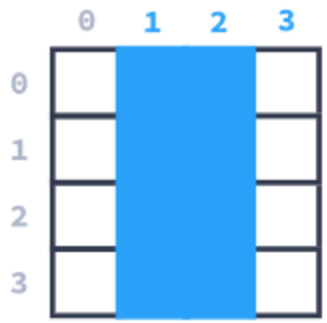
```
sel_lol = []

for row in data_lol:
    col3 = row[2]
    sel_lol.append(col3)
```

NumPy method

```
sel_np = data_np[:,2]
```

- Produces a 1D ndarray.

Selecting Multiple Columns

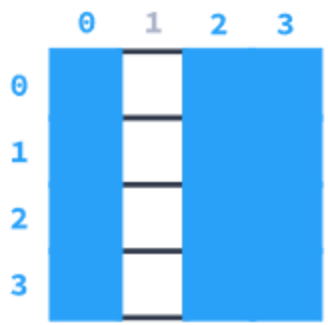



```
sel_lol = []

for row in data_lol:
    col23 = row[1:3]
    sel_lol.append(col23)
```

```
sel_np = data_np[:,1:3]
```

- Produces a 2D ndarray.

Selecting Multiple, Specific Column

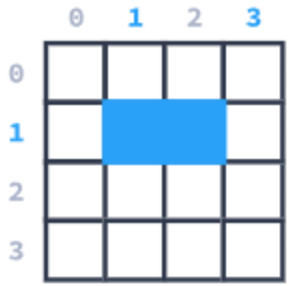



```
sel_lol = []

for row in data_lol:
    cols = [row[0],
            row[2],
            row[3]]
    sel_lol.append(cols)
```

```
cols = [0,2,3]
sel_np = data_np[:,cols]
```

- Produces a 2D ndarray.

Selecting a 1D slice (row)



List of lists method

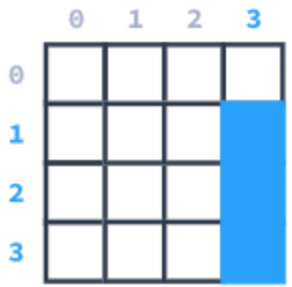
```
sel_lol = data_lol[1][1:3]
```

NumPy method

```
sel_np = data_np[1,1:3]
```

- Produces a 1D ndarray.

Selecting a 1D slice (column)



List of lists method

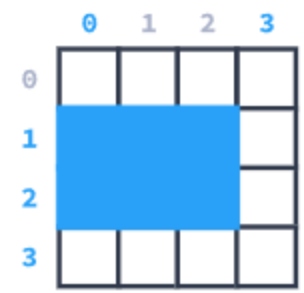
```
sel_lol = []

rows = data_lol[1:]
for r in rows:
    col4 = r[3]
    sel_lol.append(col4)
```

NumPy method

```
sel_np = data_np[1:,3]
```

- Produces a 1D ndarray.

Selecting a 2D slice



List of lists method

```
sel_lol = []

rows = data_lol[1:3]
for r in rows:
    new_row = r[:3]
    sel_lol.append(new_row)
```

NumPy method

```
sel_np = data_np[1:3,:3]
```

- Returns a 1D ndarray.

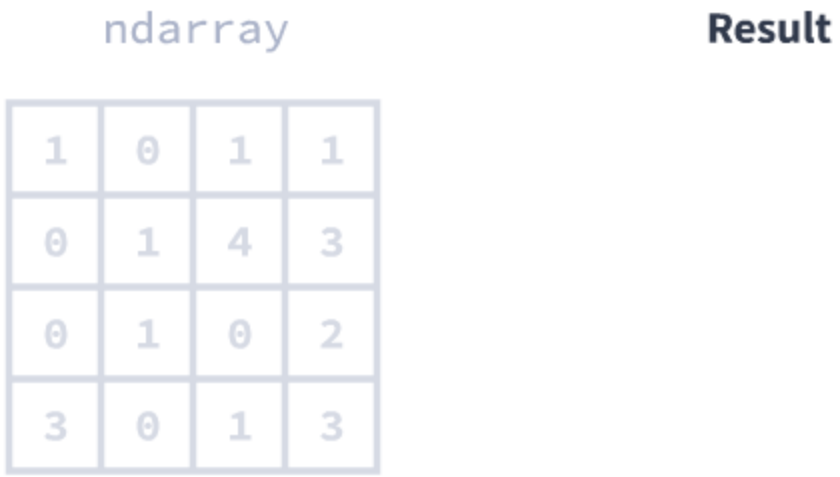
- Làm sao để cộng 2 cột với numpy?

```
# convert the list of lists to an ndarray
my_numbers = np.array(my_numbers)

# select each of the columns - the result
# of each will be a 1D ndarray
col1 = my_numbers[:,0]
col2 = my_numbers[:,1]

# add the two columns
sums = col1 + col2
```

- Trong slide trước, ta đã có kết quả của trip_mph. Hãy tìm ghía trị max, min, trung bình của mảng vừa tính.

- Tốc độ chuyến đi trung bình (trung bình) (làm tròn): 170 dặm / giờ
  Tốc độ chuyến đi tối đa (làm tròn): 82.000 dặm / giờ
  Tốc độ chuyến đi 82.000 dặm / giờ chắc chắn không thể thực hiện được trong giao thông ở New York - nhanh hơn gần 20 lần so với máy bay nhanh nhất thế giới! Và tốc độ chuyến đi trung bình 170 dặm / giờ cũng không thể thực hiện được. Những kết quả này có thể là do lỗi trong các thiết bị ghi dữ liệu hoặc có lẽ lỗi được thực hiện ở đâu đó trong đường dẫn dữ liệu.

- Tiếp theo, ta sẽ tính toán thống kê cho ndarrays 2D. Nếu chúng ta sử dụng phương thức ndarray.max () trên 2D ndarray mà không có bất kỳ tham số bổ sung nào, nó sẽ trả về một giá trị duy nhất, giống như mảng 1D



ndarray

1	0	1	1
0	1	4	3
0	1	0	2
3	0	1	3

Result

ndarray

1	0	1	1
0	1	4	3
0	1	0	2
3	0	1	3

Result

- Ta thấy: fare amount+fees amount+tolls amount+tip amount=total amount
- Task: Hãy kiểm tra xem các dữ liệu có đúng không, thống kê xem bao nhiêu dữ liệu sai

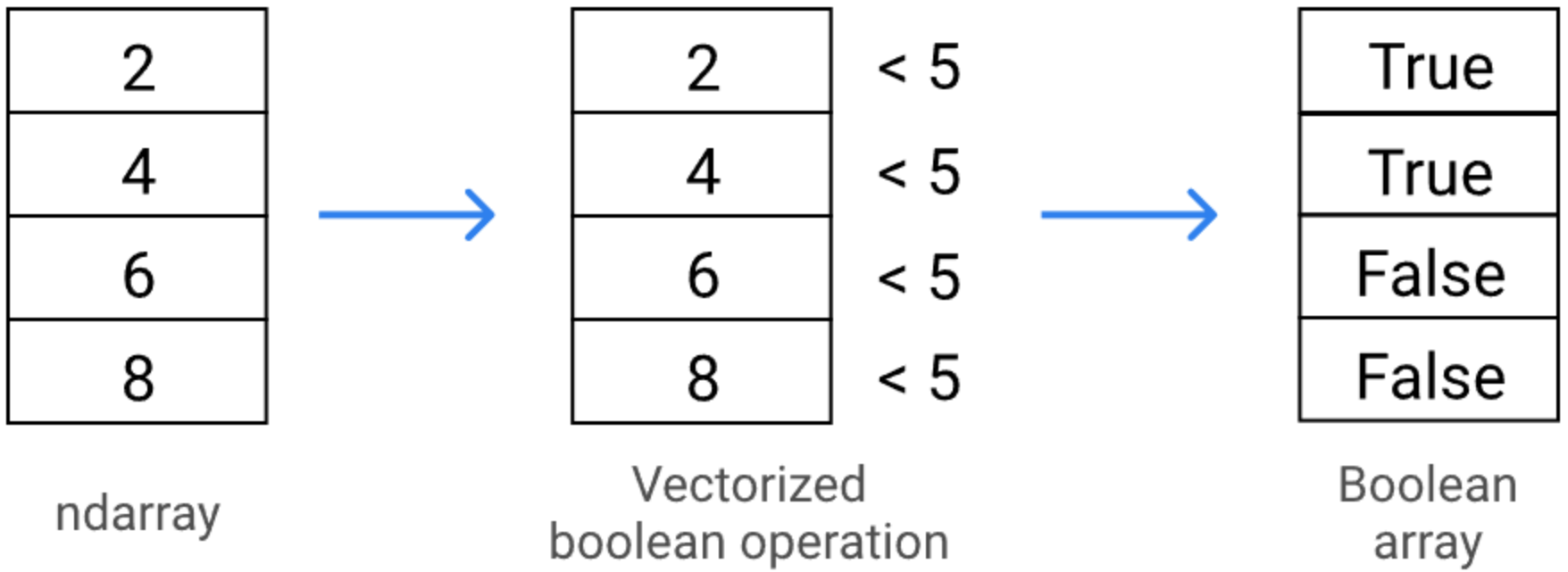
fare_amount	fees_amount	tolls_amount	tip_amount	total_amount
52.0	0.8	5.54	11.65	69.99
45.0	1.3	0.00	8.00	54.3
36.5	1.3	0.00	0.00	37.8
26.0	1.3	0.00	5.46	32.76
17.5	1.3	0.00	0.00	18.8

- Ta có sẽ số hóa dữ liệu với tên gọi Boolean Indexing nhằm tối ưu và dễ dàng hơn trong việc sử lý dữ liệu. Chạy dòng lệnh dưới và đưa ra nhận xét.
  Và làm cách nào loại bỏ dòng vô nghĩa

```
1 import numpy as np
2 taxi = np.genfromtxt('nyc_taxis.csv', delimiter=',')
3 taxi_shape = taxi.shape
```

```
1 taxi = np.genfromtxt('nyc_taxis.csv', delimiter=',', skip_header=1)
2 taxi_shape = taxi.shape
```

- Ta sẽ nói tới 1 khái niệm tiếp theo về Boolean vectors hoặc Boolean masks, hoặc Boolean array

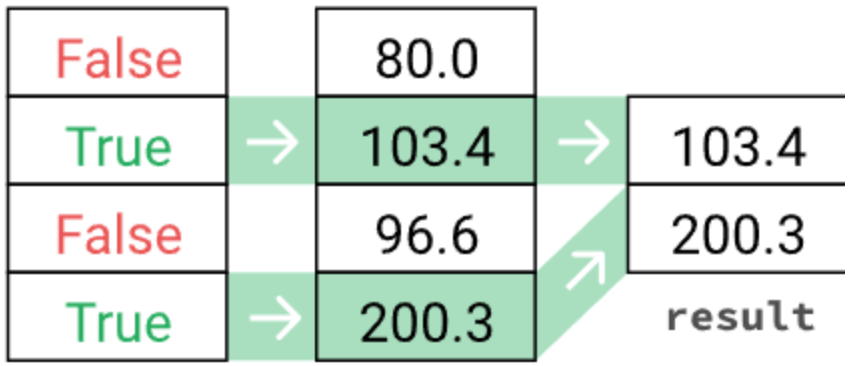


- Ta sẽ nói tới 1 khái niệm tiếp theo về Boolean vectors hoặc Boolean masks, hoặc Boolean array

```
1 a = np.array([1, 2, 3, 4, 5])
2 b = np.array(["blue", "blue", "red", "blue"])
3 c = np.array([80.0, 103.4, 96.9, 200.3])
4 a_bool = a < 3
5 b_bool = b == "blue"
6 c_bool = c > 100
```

- Ta sẽ lập chỉ mục mới như sau.
- Mảng Boolean hoạt động như một bộ lọc để các giá trị tương ứng với True trở thành một phần của kết quả và các giá trị tương ứng với False sẽ bị loại bỏ.

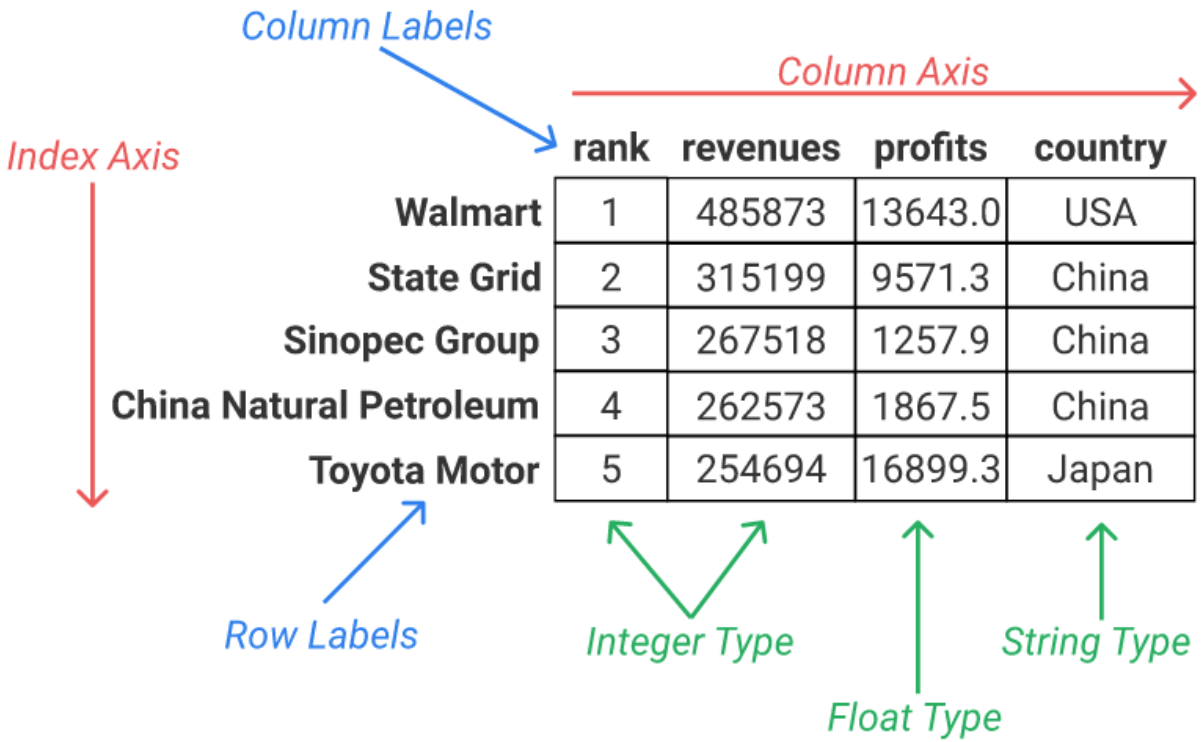
```
result = c[c_bool]
```



- Task: Hãy sử dụng lập chỉ mục Boolean để xác nhận số lượt đi taxi trong tập dữ liệu trong tháng 2.

PADAS

- Ta


Column Labels
Column Axis
Index Axis
rank
revenues
profits
country
Walmart
1
485873
13643.0
USA
State Grid
2
315199
9571.3
China
Sinopec Group
3
267518
1257.9
China
China Natural Petroleum
4
262573
1867.5
China
Toyota Motor
5
254694
16899.3
Japan
Row Labels
Integer Type
Float Type
String Type

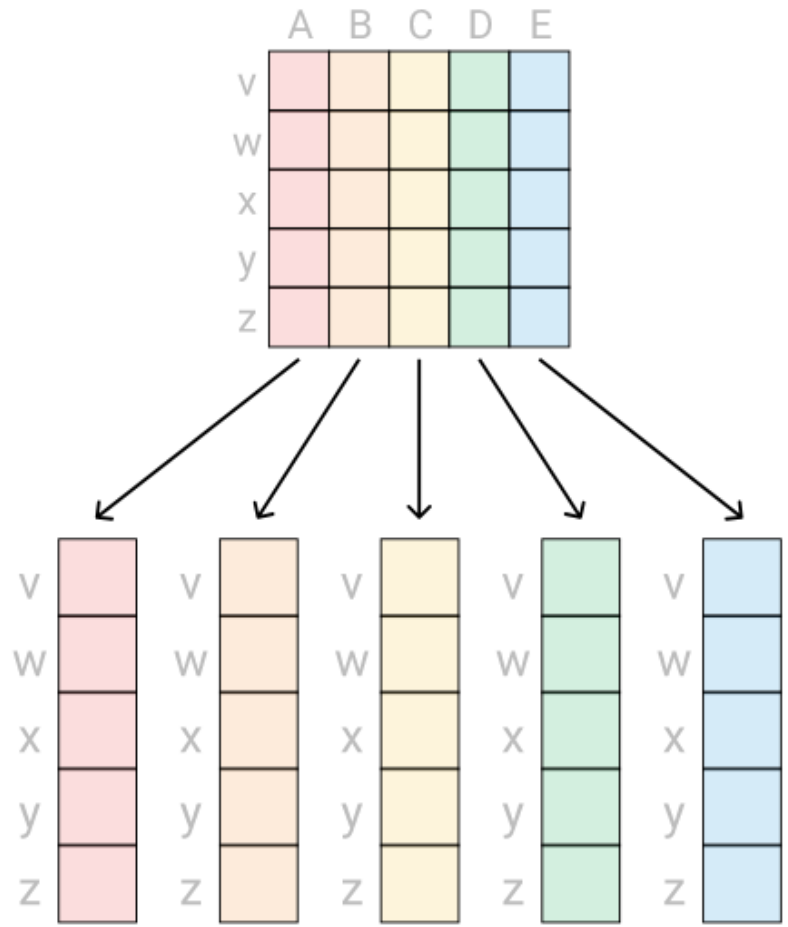
- We'll work with a dataset from Fortune magazine's 2017 Global 500 list, which ranks the top 500 corporations worldwide by revenue
- Analyze it
- Task1:
- 1. Use Python's type() function to assign the type of f500 to f500_type.
- 2. Use the DataFrame.shape attribute to assign the shape of f500 to f500_shape.
- 3. Run your code, then use the variable inspector to look at the variables f500, f500_type, and f500_shape.

- The code we wrote on the previous screen informed us our data has 500 rows, 16 columns, and is stored as a pandas.core.frame.DataFrame object — or just **dataframe**, the primary pandas data structure.
- Recall that one of the features that makes pandas better for working with data is its support for string column and row labels:
- **Axis values can have string labels, not just numeric ones**.
- Dataframes can contain columns with multiple data types: including integer, float, and string.
- Let's verify this next. To view the first few rows of our dataframe, we can use the DataFrame.head() method. By default, it returns the first five rows of our dataframe. However, it also accepts an optional integer parameter, which specifies the number of rows:
- f500.head(3)
- U can try
- F500.tail(5) what is it?

- Since our axis in pandas have labels, we can select data using those labels — unlike in NumPy, where we needed to know the exact index location. To do this, we can use the DataFrame.loc[] attribute. The syntax for DataFrame.loc[] is:

```
df.loc[row_label, column_label]
```

- Task:
- Select the industry column and assign the result to the variable name industries.
- Use Python's type() function to assign the type of industries to industries_type.
- Run your code, then use the variable inspector to look at the variables.


A B C D E
v
w
x
y
z

- Select multuple columns

```
f500_selection.loc[:,["country","rank"]]
```

```
f500_selection[["country","rank"]]
```

	country	rank
Walmart	USA	1
State Grid	China	2
Sinopec Group	China	3
China Natural Petroleum	China	4
Toyota Motor	Japan	5

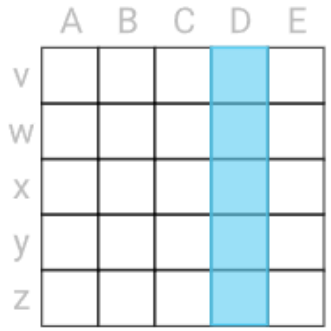
```
f500_selection.loc[:,"rank":"profits"]
```

	rank	revenues	profits
Walmart	1	485873	13643.0
State Grid	2	315199	9571.3
Sinopec Group	3	267518	1257.9
China Natural Petroleum	4	262573	1867.5
Toyota Motor	5	254694	16899.3

- Task:
- Select the country column. Assign the result to the variable name countries.
- In order, select the revenues and years_on_global_500_list columns. Assign the result to the variable name revenues_years.
- In order, select all columns from ceo up to and including sector. Assign the result to the variable name ceo_to_sector.
- After you have run your code, use the variable inspector to view the variables.

- Task:
- By selecting data from f500:
- 1.Create a new variable toyota, with:
- Just the row with index Toyota Motor.
- All columns.
- 2.Create a new variable, drink_companies, with:
- Rows with indicies Anheuser-Busch InBev, Coca-Cola, and Heineken Holding, in that order.
- All columns.
- 3.Create a new variable, middle_companies with:
- All rows with indicies from Tata Motors to Nationwide, inclusive.
- All columns from rank to country, inclusive.

Original Dataframe

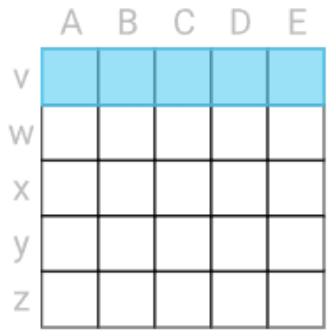


Code

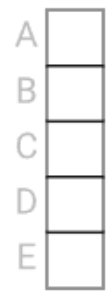
```
single_col = df["D"]
```

Result

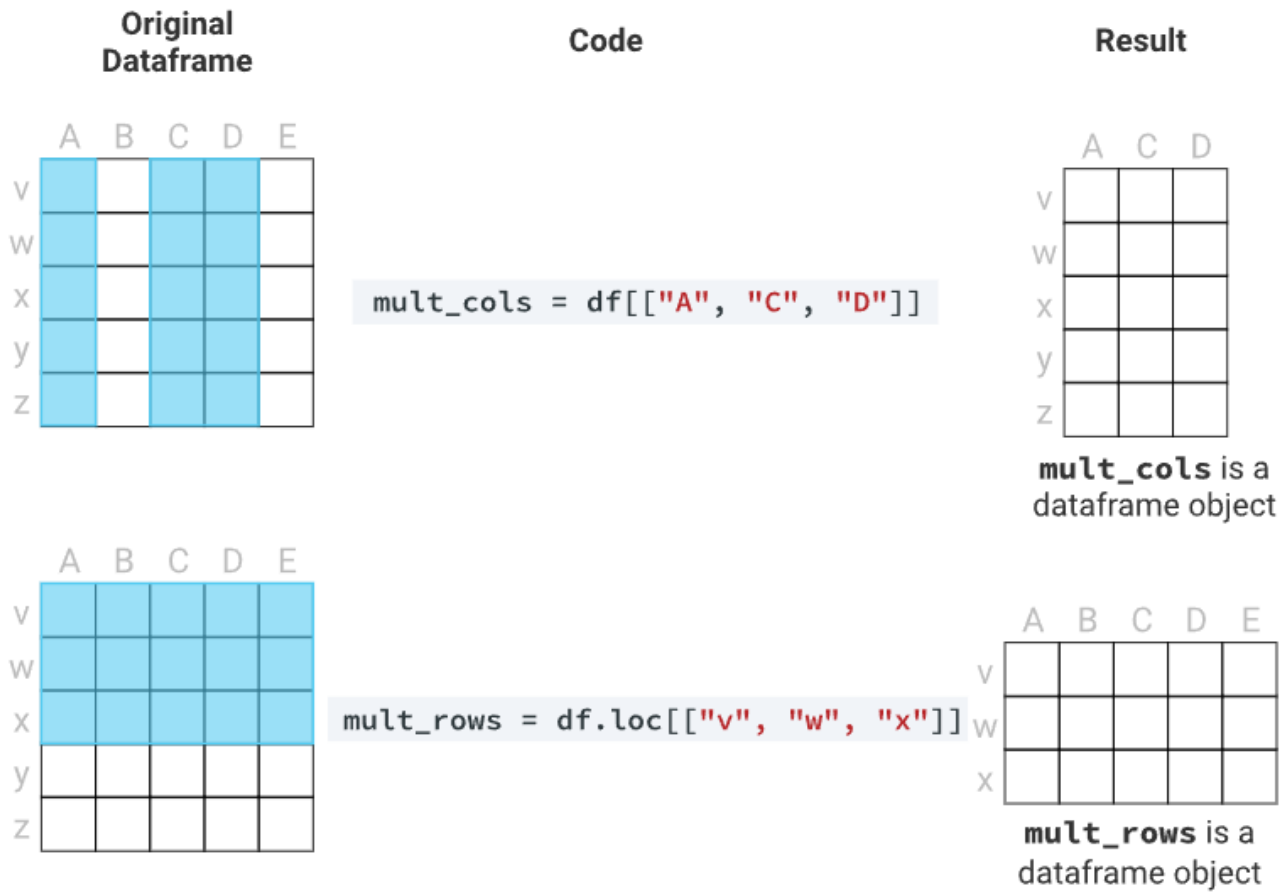
`single_col` is a series object




```
single_row = df.loc["v"]
```



`single_row` is a series object


Original Dataframe
Code
Result
A B C D E
v
w
x
y
z
mult_cols = df[["A", "C", "D"]]
A C D
mult_cols is a dataframe object
mult_rows = df.loc[["v", "w", "x"]]
mult_rows is a dataframe object

- What is it?

```
sectors = f500["sector"]
print(type(sectors))
```

- And

```
sectors_value_counts = sectors.value_counts()
print(sectors_value_counts)
```

- Task:

Find the counts of each unique value in the country column in the f500 dataframe:

1. Select the country column in the f500 dataframe. Assign it to a variable named countries.

2.Use the Series.value_counts() method to return the value counts for countries. Assign the results to country_counts.

- Task:

- Select the item at index label India. Assign the result to the variable name india.
- In order, select the items with index labels USA, Canada, and Mexico. Assign the result to the variable name north_america.

- Task:

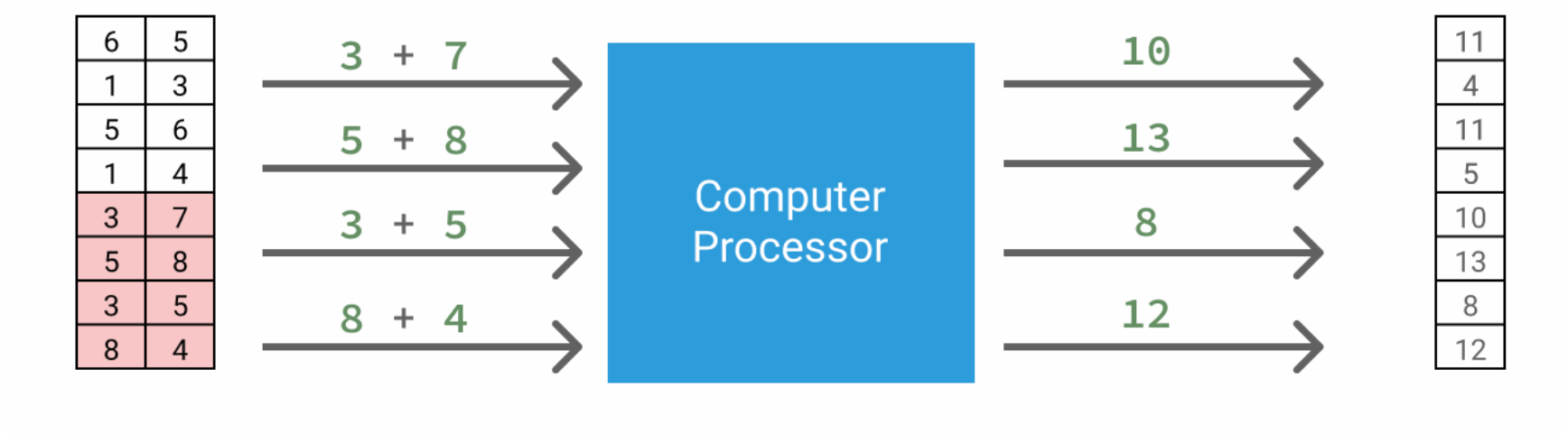
By selecting data from f500:

1.Create a new variable big_movers, with:

- Rows with indices Aviva, HP, JD.com, and BHP Billiton, in that order.
- The rank and previous_rank columns, in that order.
- 2.Create a new variable, bottom_companies with:
- All rows with indices from National Gridto AutoNation, inclusive.
- The rank, sector, and country columns.

- We'll learn another way pandas makes working with data easier. It has many built-in methods and functions for common exploration and analysis tasks. As we learn these, we'll also explore how pandas uses many of the concepts we learned in the NumPy lessons, including vectorized operations and boolean indexing.
- We'll continue working with a data set from Fortune magazine's Global 500 list 2017, which ranks the top 500 corporations worldwide by revenue.

- Because pandas is designed to operate like NumPy, a lot of concepts and methods from Numpy are supported. Recall that one of the ways NumPy makes working with data easier is with **vectorized operations**, or operations applied to multiple data points at once:



- Task:
- Subtract the values in the `rank` column from the values in the `previous_rank` column. Assign the result to `rank_change`.

- Task:
- Use the `Series.max()` method to find the maximum value for the `rank_change` series. Assign the result to the variable `rank_change_max`.
- Use the `Series.min()` method to find the minimum value for the `rank_change` series. Assign the result to the variable `rank_change_min`.
- After running your code, use the variable inspector to view the new variable you created.

- Problem
- Biggest increase in rank: `226`
- Biggest decrease in rank: `-500`
- Task:
- Return a series of descriptive statistics for the `rank` column in `f500`.
  - Select the `rank` column. Assign it to a variable named `rank`.
  - Use the `Series.describe()` method to return a series of statistics for `rank`. Assign the result to `rank_desc`.
- Return a series of descriptive statistics for the `previous_rank` column in `f500`.
  - Select the `previous_rank` column. Assign it to a variable named `prev_rank`.
  - Use the `Series.describe()` method to return a series of statistics for `prev_rank`. Assign the result to `prev_rank_desc`.
- After you have run your code, use the variable inspector to view each of the new variables you created. Try to identify any potential issues with the data before moving onto the next screen.

PANDAS: INTERMEDIATE

Task

- We've already read the data set into a pandas dataframe and assigned it to a variable named f500. We also replaced all 0 values in the previous_rank column with NaN, like we did in the previous lesson.
- Select the rank, revenues, and revenue_change columns in f500. Then, use the DataFrame.head() method to select the first five rows. Assign the result to f500_selection.
- Use the variable inspector to view f500_selection. Compare the results to the first few lines of our CSV file above.
- Take a look at the documentation for the pandas.read_csv() function to try to understand the results. If you have trouble understanding, don't worry! We'll explain the results on the next screen

PANDAS: INTERMEDIATE

Task

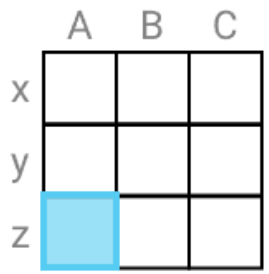
- Use the pandas.read_csv() function to read the f500.csv CSV file as a pandas dataframe. Assign it to the variable name f500.
  - Do not use the index_col parameter.
- Use the code below to insert the NaN values into the previous_rank column: f500.loc[f500["previous_rank"] == 0, "previous_rank"] = np.nan

PANDAS: INTERMEDIATE

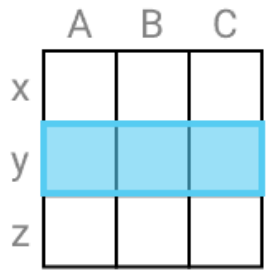
```
f500 = pd.read_csv("f500.csv")
print(f500[['company', 'rank', 'revenues']].head())
```

```
_                    company  rank  revenues
0                    Walmart     1    485873
1                State Grid      2    315199
2             Sinopec Group      3    267518
3  China National Petroleum      4    262573
4              Toyota Motor      5    254694
```

PANDAS: INTERMEDIATE 3




```
df.iloc[2,0]
```




```
df.iloc[1]
```

PANDAS: INTERMEDIATE 3

Task:

- Select just the fifth row of the `f500` dataframe. Assign the result to `fifth_row`.
- Select the value in first row of the `company` column. Assign the result to `company_value`.

```
first_column = f500.iloc[:,0]
print(first_column)
```

```
second_to_sixth_rows = f500[1:5]
```

Task:

- Select the first three rows of the `f500` dataframe. Assign the result to `first_three_rows`.
- Select the first and seventh rows and the first five columns of the `f500` dataframe. Assign the result to `first_seventh_row_slice`.
- After running your code, use the variable inspector to examine each of the objects you created.

```
rev_is_null = f500["revenue_change"].isnull()
print(rev_is_null.head())
```

```
rev_change_null = f500[rev_is_null]
print(rev_change_null[["company","country","sector"]])
```

Task:

- Use the `Series.isnull()` method to select all rows from `f500` that have a null value for the `previous_rank` column. Select only the `company`, `rank`, and `previous_rank` columns. Assign the result to `null_previous_rank`.

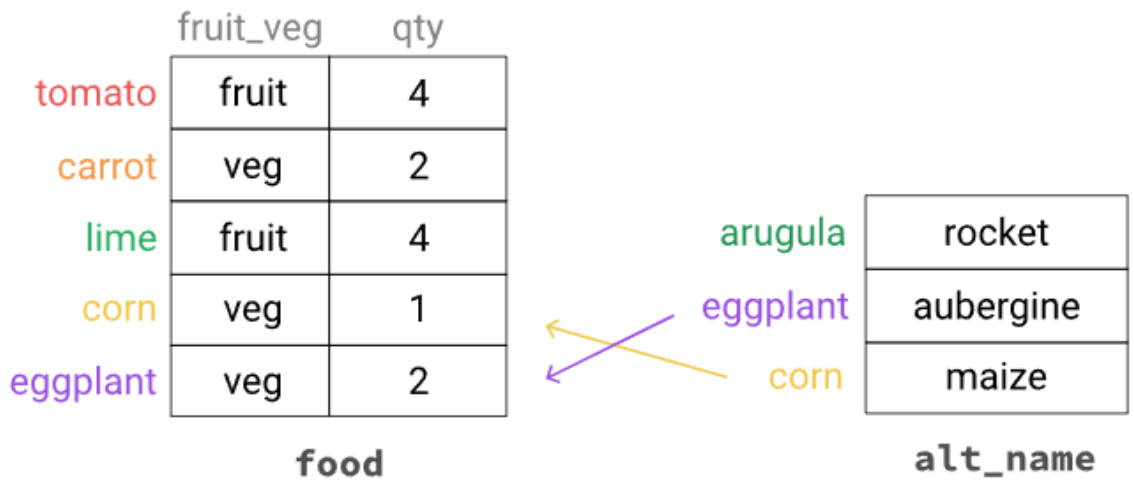
	company	rank	previous_rank
48	Legal & General Group	49	NaN
90	Uniper	91	NaN
123	Dell Technologies	124	NaN

```
first_null_prev_rank = null_previous_rank.iloc[0]
print(first_null_prev_rank)
```

- Task:
- Assign the first five rows of the `null_previous_rank` dataframe to the variable `top5_null_prev_rank` by choosing the correct method out of either `loc[]` or `iloc[]`.

```
previously_ranked = f500[f500["previous_rank"].notnull()]
```

```
rank_change = previously_ranked["previous_rank"] - previously_ranked["rank"]
print(rank_change.shape)
print(rank_change.tail(3))
```



	fruit_veg	qty	alt_name
tomato	fruit	4	NaN
carrot	veg	2	NaN
lime	fruit	4	NaN
corn	veg	1	maize
eggplant	veg	2	aubergine

food

- Task:
- Use the `Series.notnull()` method to select all rows from `f500` that have a non-null value for the `previous_rank` column. Assign the result to `previously_ranked`
- From the `previously_ranked` dataframe, subtract the `rank` column from the `revious_rank` column. Assign the result to `rank_change`.
- Assign the values in the `rank_change` to a new column in the `f500` dataframe, `"rank_change"`.
- Once you have run your code, use the variable inspector to look at the `f500` dataframe and observe how the new column aligns with the existing data.

- Boolean indexing is a powerful tool which allows us to select or exclude parts of our data based on their values. However, to answer more complex questions, we need to learn how to combine boolean arrays.
- To recap, boolean arrays are created using any of the Python standard **comparison operators**: == (equal), > (greater than), < (less than), != (not equal).
- We combine boolean arrays using **boolean operators**. In Python, these boolean operators are and, or, and not. In pandas, the operators are slightly different:

pandas	Python equivalent	Meaning
`a & b`	`a and b`	`True` if both `a` and `b` are `True`, else `False`
`a | b`	`a or b`	`True` if either `a` or `b` is `True`
`~a`	`not a`	`True` if `a` is `False`, else `False`

```
cols = ["company",
        "revenues",
        "country"]
f500_sel = f500[cols].head()
```

	company	revenues	country
0	Walmart	485873	USA
1	State Grid	315199	China
2	Sinopec Group	267518	China
3	China Nation...	262573	China
4	Toyota Motor	254694	Japan

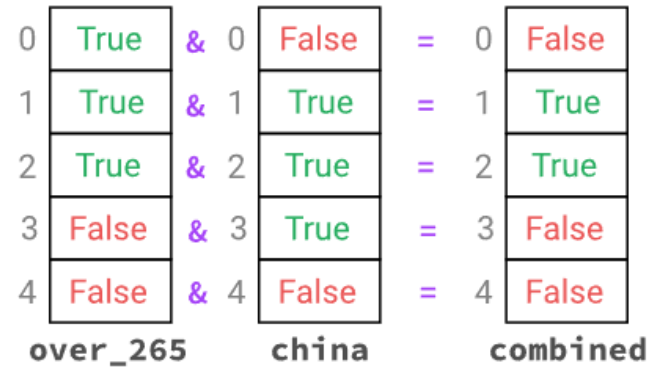
f500_sel

```
over_265 = f500_sel["revenues"] > 265000
china = f500_sel["country"] == "China"
```

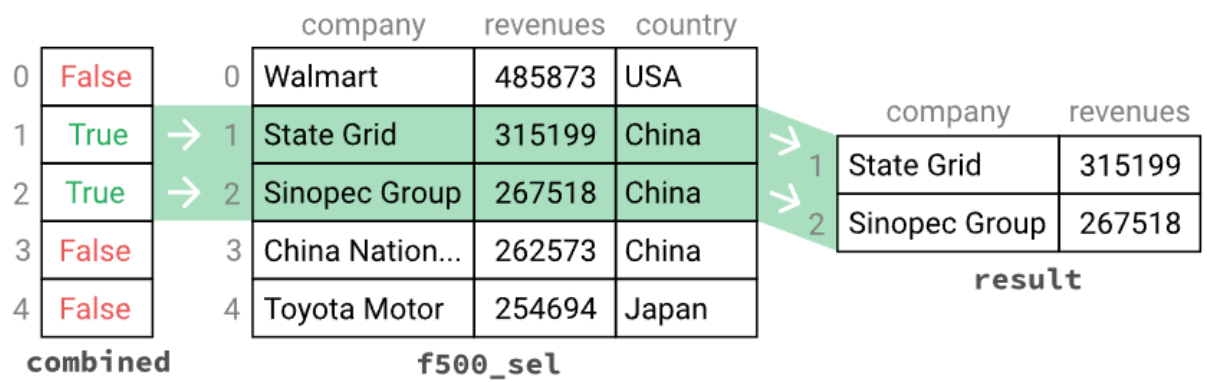
	over_265
0	True
1	True
2	True
3	False
4	False

	china
0	False
1	True
2	True
3	True
4	False

```
combined = over_265 & china
```




```
final_cols = ["company","revenues"]
result = f500_sel.loc[combined,final_cols]
```



- Task
- Select all companies with revenues over 100 billion and negative profits from the `f500` dataframe. The result should include all columns.
  - Create a boolean array that selects the companies with revenues greater than 100 billion. Assign the result to `large_revenue`.
  - Create a boolean array that selects the companies with profits less than 0. Assign the result to `negative_profits`.
  - Combine `large_revenue` and `negative_profits`. Assign the result to `combined`.
  - Use `combined` to filter `f500`. Assign the result to `big_rev_neg_profit`.

- Task
- Select all rows for companies whose `country` value is either Brazil or Venezuela. Assign the result to `brazil_venezuela`.
- Select the first five companies in the Technology sector for which the country is **not** the USA from the `f500` dataframe. Assign the result to `tech_outside_usa`.

- Task
- Find the company headquartered in Japan with the largest number of employees.
  - Select only the rows that have a country name equal to `Japan`.
  - Use `DataFrame.sort_values()` to sort those rows *by* the `employees` column in *descending* order.
  - Use `DataFrame.iloc[]` to select the first row from the sorted dataframe.
  - Extract the company name from the index label `company` from the first row. Assign the result to `top_japanese_employer`.
- After running your code, use the variable inspector to view the top employer for `Japan`.

```
# Create an empty dictionary to store the results
avg_rev_by_country = {}

# Create an array of unique countries
countries = f500["country"].unique()

# Use a for loop to iterate over the countries
for c in countries:
    # Use boolean comparison to select only rows that
    # correspond to a specific country
    selected_rows = f500[f500["country"] == c]
    # Calculate the mean average revenue for just those rows
    mean = selected_rows["revenues"].mean()
    # Assign the mean value to the dictionary, using the
    # country name as the key
    avg_rev_by_country[c] = mean
```

- Results:

```
{'USA': 64218.371212121216,
 'China': 55397.880733944956,
 'Japan': 53164.03921568627,
 'Germany': 63915.0,
 'Netherlands': 61708.92857142857,
 'Britain': 51588.708333333336,
 'South Korea': 49725.6,
 ...
 }
```

- And we're the following dictionary of the top employer in each country

```
{'USA': 'Walmart',
 'China': 'China National Petroleum',
 'Japan': 'Toyota Motor',
 ...
 'Turkey': 'Koc Holding',
 'U.A.E': 'Emirates Group',
 'Israel': 'Teva Pharmaceutical Industries'}
```

- Task:
- Create an empty dictionary, `top_employer_by_country` to store the results of the exercise.
- Use the `Series.unique()` method to create an array of unique values from the `country` column.
- Use a for loop to iterate over the array unique countries. In each iteration:
  - Select only the rows that have a country name equal to the current iteration.
  - Use `DataFrame.sort_values()` to sort those rows *by* the `employees` column in *descending* order.
  - Select the first row from the sorted dataframe.
  - Extract the company name from the index label `company` from the first row.
  - Assign the results to the `top_employer_by_country` dictionary, using the country name as the key, and the company name as the value.
- After running your code, use the variable inspector to view the top employer for each country.

FINAL PROJECT FOR DATA CLEANING

- we'll work with a dataset of used cars from *eBay Kleinanzeigen*, a classifieds section of the German eBay website.
- The data dictionary provided with data is as follows:

`dateCrawled` - When this ad was first crawled. All field-values are taken from this date.
`name` - Name of the car.
`seller` - Whether the seller is private or a dealer.
`offerType` - The type of listing
`price` - The price on the ad to sell the car.
`abtest` - Whether the listing is included in an A/B test.
`vehicleType` - The vehicle Type.
`yearOfRegistration` - The year in which the car was first registered.
`gearbox` - The transmission type.
`powerPS` - The power of the car in PS.
`model` - The car model name.
`kilometer` - How many kilometers the car has driven.
`monthOfRegistration` - The month in which the car was first registered.
`fuelType` - What type of fuel the car uses.
`brand` - The brand of the car.
`notRepairedDamage` - If the car has a damage which is not yet repaired.
`dateCreated` - The date on which the eBay listing was created.
`nrOfPictures` - The number of pictures in the ad.
`postalCode` - The postal code for the location of the vehicle.
`lastSeenOnline` - When the crawler saw this ad last online.

FINAL PROJECT FOR DATA CLEANING

- Start by writing a paragraph in a markdown cell introducing the project and the dataset.
- Import the pandas and NumPy libraries
- Read the `autos.csv` CSV file into pandas, and assign it to the variable name `autos`.
  - Try without specifying any encoding (which will default to `UTF-8`)
  - If you get an encoding error, try the next two most popular encodings (`Latin-1` and `Windows-1252`) until you are able to read the file without error.
- Create a new cell with just the variable `autos` and run this cell.
  - A neat feature of jupyter notebook is its ability to render the first few and last few values of any pandas object.
- Use the `DataFrame.info()` and `DataFrame.head()` methods to print information about the `autos` dataframe, as well as the first few rows.
  - Write a markdown cell briefly describing your observations.

FINAL PROJECT FOR DATA CLEANING

- The dataset contains 20 columns, most of which are strings.
- Some columns have null values, but none have more than ~20% null values.
- The column names use camelcase instead of Python's preferred snakecase, which means we can't just replace spaces with underscores.
- Let's convert the column names from camelcase to snakecase and reword some of the column names based on the data dictionary to be more descriptive.

FINAL PROJECT FOR DATA CLEANING

- Task:
- Use the `DataFrame.columns` attribute to print an array of the existing column names.
- Copy that array and make the following edits to columns names:
  - `yearOfRegistration` to `registration_year`
  - `monthOfRegistration` to `registration_month`
  - `notRepairedDamage` to `unrepaired_damage`
  - `dateCreated` to `ad_created`
  - The rest of the column names from camelcase to snakecase.
- Assign the modified column names back to the `DataFrame.columns` attribute.
- Use `DataFrame.head()` to look at the current state of the `autos` dataframe.
- Write a markdown cell explaining the changes you made and why.

FINAL PROJECT FOR DATA CLEANING

- Now let's do some basic data exploration to determine what other cleaning tasks need to be done. Initially we will look for:
- Text columns where all or almost all values are the same. These can often be dropped as they don't have useful information for analysis.
- Examples of numeric data stored as text which can be cleaned and converted.
- The following methods are helpful for exploring the data:
- `DataFrame.describe()` (with `include='all'` to get both categorical and numeric columns)
- `Series.value_counts()` and `Series.head()` if any columns need a closer look.

FINAL PROJECT FOR DATA CLEANING

- Task:
- Use `DataFrame.describe()` to look at descriptive statistics for all columns.
- Write a markdown cell noting:
  - Any columns that have mostly one value that are candidates to be dropped
  - Any columns that need more investigation.
  - Any examples of numeric data stored as text that needs to be cleaned.
- If you need to investigate any columns more, do so and write up any additional things you found.
- You likely found that the `price` and `odometer` columns are numeric values stored as text. For each column:
  - Remove any non-numeric characters.
  - Convert the column to a numeric dtype.
  - Use `DataFrame.rename()` to rename the column to `odometer_km`.

FINAL PROJECT FOR DATA CLEANING

- We learned that there are a number of text columns where almost all of the values are the same (`seller` and `offer_type`). We also converted the `price` and `odometer` columns to numeric types and renamed `odometer` to `odometer_km`.
- Let's continue exploring the data, specifically looking for data that doesn't look right. We'll start by analyzing the `odometer_km` and `price` columns. Here's the steps we'll take:
- Analyze the columns using minimum and maximum values and look for any values that look unrealistically high or low (outliers) that we might want to remove.
- We'll use:
  - `Series.unique().shape` to see how many unique values
  - `Series.describe()` to view min/max/median/mean etc
  - `Series.value_counts()`, with some variations:
    - chained to `.head()` if there are lots of values.
    - Because `Series.value_counts()` returns a series, we can use `Series.sort_index()` with `ascending= True` or `False` to view the highest and lowest values with their counts (can also chain to `head()` here).
  - When removing outliers, we can do `df[(df["col"] >= x ) & (df["col"] <= y )]`, but it's more readable to use `df[df["col"].between(x,y)]`

FINAL PROJECT FOR DATA CLEANING

- Task:
- For each of the `odometer_km` and `price` columns:
  - Use the techniques above to explore the data
  - If you find there are outliers, remove them and write a markdown paragraph explaining your decision.
  - After you have removed the outliers, make some observations about the remaining values.

FINAL PROJECT FOR DATA CLEANING

- Let's now move on to the date columns and understand the date range the data covers.
- There are 5 columns that should represent date values. Some of these columns were created by the crawler, some came from the website itself. We can differentiate by referring to the data dictionary:

```
- `date_crawled`: added by the crawler
- `last_seen`: added by the crawler
- `ad_created`: from the website
- `registration_month`: from the website
- `registration_year`: from the website
```

- Right now, the `date_crawled`, `last_seen`, and `ad_created` columns are all identified as string values by pandas. Because these three columns are represented as strings, we need to convert the data into a numerical representation so we can understand it quantitatively. The other two columns are represented as numeric values, so we can use methods like `Series.describe()` to understand the distribution without any extra data processing.

FINAL PROJECT FOR DATA CLEANING

- Let's first understand how the values in the three string columns are formatted. These columns all represent full timestamp values, like so:

```
autos[['date_crawled','ad_created','last_seen']][0:5]
```

	date_crawled	ad_created	last_seen
0	2016-03-26 17:47:46	2016-03-26 00:00:00	2016-04-06 06:45:54
1	2016-04-04 13:38:56	2016-04-04 00:00:00	2016-04-06 14:45:08
2	2016-03-26 18:57:24	2016-03-26 00:00:00	2016-04-06 20:15:37
3	2016-03-12 16:58:10	2016-03-12 00:00:00	2016-03-15 03:16:28
4	2016-04-01 14:38:50	2016-04-01 00:00:00	2016-04-01 14:38:50

-

```
print(autos['date_crawled'].str[:10])
```

```
0         2016-03-26
1         2016-04-04
2         2016-03-26
3         2016-03-12
...
```

FINAL PROJECT FOR DATA CLEANING

- Task:
- Use the workflow we just described to calculate the distribution of values in the `date_crawled`, `ad_created`, and `last_seen` columns (all string columns) as percentages.
  - To include missing values in the distribution and to use percentages instead of counts, chain the `Series.value_counts(normalize=True, dropna=False)` method.
  - To rank by date in ascending order (earliest to latest), chain the `Series.sort_index()` method.
  - Write a markdown cell after each column exploration to explain your observations.
- Use `Series.describe()` to understand the distribution of `registration_year`.
  - Write a markdown cell explaining your observations.

FINAL PROJECT FOR DATA CLEANING

- The `registration_year` column contains some odd values:
- The minimum value is `1000`, before cars were invented
- The maximum value is `9999`, many years into the future
- Because a car can't be first registered after the listing was seen, any vehicle with a registration year above 2016 is definitely inaccurate. Determining the earliest valid year is more difficult. Realistically, it could be somewhere in the first few decades of the 1900s.
- Let's count the number of listings with cars that fall outside the 1900 - 2016 interval and see if it's safe to remove those rows entirely, or if we need more custom logic.

FINAL PROJECT FOR DATA CLEANING

- Task:
- Decide which the highest and lowest acceptable values are for the `registration_year` column.
  - Write a markdown cell explaining your decision and why.
- Remove the values outside those upper and lower bounds and calculate the distribution of the remaining values using `Series.value_counts(normalize=True)`.
  - Write a markdown cell explaining your observations.

FINAL PROJECT FOR DATA CLEANING

When working with data on cars, it's natural to explore variations across different car brands. We can use aggregation to understand the `brand` column.

- If you recall in an earlier lesson, we explored how to use loops to perform aggregation. Here's what the process looks like:
- Identify the unique values we want to aggregate by
- Create an empty dictionary to store our aggregate data
- Loop over the unique values, and for each: - Subset the dataframe by the unique values - Calculate the mean of whichever column we're interested in - Assign the val/mean to the dict as k/v.

FINAL PROJECT FOR DATA CLEANING

- Task:
- Explore the unique values in the `brand` column, and decide on which brands you want to aggregate by.
  - You might want to select the top 20, or you might want to select those that have over a certain percentage of the total values (e.g. > 5%).
  - Remember that `Series.value_counts()` produces a series with index labels, so you can use `Series.index` attribute to access the labels, should you wish.
- Write a short paragraph describing the brand data, and explaining which brands you've chosen to aggregate on.
- Create an empty dictionary to hold your aggregate data.
  - Loop over your selected brands, and assign the mean price to the dictionary, with the brand name as the key.
  - Print your dictionary of aggregate data, and write a paragraph analyzing the results.

FINAL PROJECT FOR DATA CLEANING

We observed that in the top 6 brands, there's a distinct price gap.

- Audi, BMW and Mercedes Benz are more expensive
- Ford and Opel are less expensive
- Volkswagen is in between

For the top 6 brands, let's use aggregation to understand the average mileage for those cars and if there's any visible link with mean price. While our natural instinct may be to display both aggregated series objects and visually compare them, this has a few limitations:

- it's difficult to compare more than two aggregate series objects if we want to extend to more columns
- we can't compare more than a few rows from each series object
- we can only sort by the index (brand name) of both series objects so we can easily make visual comparisons

Instead, we can combine the data from both series objects into a single dataframe (with a shared index) and display the dataframe directly. Suggest:: using pandas dataframe

-

FINAL PROJECT FOR DATA CLEANING

Ex for the series constructor that uses the `brand_mean_prices` dictionary:

```
bmp_series = pd.Series(brand_mean_prices)
print(bmp_series)
```

```
volkswagen       5402
bmw              8332
opel             2975
mercedes_benz    8628
audi             9336
ford             3749
dtype: int64
```

The keys in the dictionary became the index in the series object. We can then create a single-column dataframe from this series object. We need to use the `columns` parameter when calling the dataframe constructor (which accepts an array-like object) to specify the column name (or the column name will be set to `0` by default):

```
df = pd.DataFrame(bmp_series, columns=['mean_price'])
df
```

	mean_price
volkswagen	5402
bmw	8332
opel	2975
mercedes_benz	8628
audi	9336
ford	3749

FINAL PROJECT FOR DATA CLEANING

Task:

- Use the loop method from the last screen to calculate the mean mileage and mean price for each of the top brands, storing the results in a dictionary.
- Convert both dictionaries to series objects, using the series constructor.
- Create a dataframe from the first series object using the dataframe constructor.
- Assign the other series as a new column in this dataframe.
- Pretty print the dataframe, and write a paragraph analyzing the aggregate data.

FINAL PROJECT FOR DATA CLEANING

Task:

- Data cleaning next steps:
  - Identify categorical data that uses german words, translate them and map the values to their english counterparts
  - Convert the dates to be uniform numeric data, so `"2016-03-21"` becomes the integer `20160321`.
  - See if there are particular keywords in the name column that you can extract as new columns
- Analysis next steps:
  - Find the most common brand/model combinations
  - Split the `odometer_km` into groups, and use aggregation to see if average prices follows any patterns based on the mileage.
  - How much cheaper are cars with damage than their non-damaged counterparts?