Python 101 - Python Libraries for Data Analysis - Numpy and Pandas

December 8, 2024

1 TASK #1: DEFINE SINGLE AND MULTI-DIMENSIONAL NUMPY ARRAYS

[1]:
```
# NumPy is a Linear Algebra Library used for multidimensional arrays
# NumPy brings the best of two worlds: (1) C/Fortran computational efficiency,␣
 ↪(2) Python language easy syntax

import numpy as np

# Let's define a one-dimensional array
list_1 = [50, 60, 80, 100, 200, 300, 500, 600]
list_1
```

[1]:
```
[50, 60, 80, 100, 200, 300, 500, 600]
```

[2]:
```
# Let's create a numpy array from the list "my_list"
my_numpy_array = np.array(list_1)
my_numpy_array
```

[2]:
```
array([ 50,  60,  80, 100, 200, 300, 500, 600])
```

[3]:
```
type(my_numpy_array)
```

[3]:
```
numpy.ndarray
```

[4]:
```
# Multi-dimensional (Matrix definition)
my_matrix = np.array([[2, 5, 8], [7,3,6]])
my_matrix
```

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

MINI CHALLENGE #1: - Write a code that creates the following 2x4 numpy array

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

[5]:
```
challenge1 = np.array([[3, 7, 9, 3], [4, 3, 2, 2]])
challenge1
```

[5]:
```
array([[3, 7, 9, 3],
       [4, 3, 2, 2]])
```

2 TASK #2: LEVERAGE NUMPY BUILT-IN METHODS AND FUNCTIONS

[6]:
```
# "rand()" uniform distribution between 0 and 1
x = np.random.rand(20)
x
```

[6]:
```
array([0.35935739, 0.43325741, 0.88238455, 0.31420639, 0.45434715,
       0.22535573, 0.48179596, 0.95975847, 0.03825129, 0.60871258,
       0.82256522, 0.11334626, 0.84976678, 0.41353044, 0.23532522,
       0.96353187, 0.36422946, 0.53042833, 0.45259285, 0.97201079])
```

[7]:
```
# you can create a matrix of random number as well
x = np.random.rand(3,3)
x
```

[7]:
```
array([[0.89305307, 0.30931093, 0.7313589 ],
       [0.2738033 , 0.68030488, 0.85975632],
       [0.60179017, 0.41297283, 0.87441933]])
```

[8]:
```
# "randint" is used to generate random integers between upper and lower bounds
x = np.random.randint(1, 50)
x
```

[8]:
```
16
```

[9]:
```
# "randint" can be used to generate a certain number of random itegers as␣
↪follows
x = np.random.randint(1 ,100, 15)
x
```

[9]:
```
array([23, 54, 72, 40, 22, 22, 60, 27, 62, 12, 82, 73, 49, 17, 95])
```

[10]:
```
# np.arange creates an evenly spaced values within a given interval
x = np.arange(1,30)
x
```

[10]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])
```

[11]:
```
# create a diagonal of ones and zeros everywhere else
x = np.eye(7)
x
```

[11]:
```
array([[1., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 1.]])
```

[12]:
```
# Matrix of ones
x = np.ones((7,7))
x
```

[12]:
```
array([[1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.]])
```

[13]:
```
# Array of zeros
x = np.zeros(8)
x
```

[13]:
```
array([0., 0., 0., 0., 0., 0., 0., 0.])
```

MINI CHALLENGE #2: - Write a code that takes in a positive integer "x" from the user and creates a 1x10 array with random numbers ranging from 0 to "x"

[14]:
```
X = int(input("Please enter a positive value"))
```

```
Please enter a positive value2
```

[15]:
```
x = np.random.randint(1, X, 10)
x
```

[15]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

3 TASK #3: PERFORM MATHEMATICAL OPERATIONS IN NUMPY

[16]:
```
# np.arange() returns an evenly spaced values within a given interval
```

[17]:
```
x = np.arange(1, 10)
x
```

[17]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[18]:
```
# Add 2 numpy arrays together
y = np.arange(1, 10)
y
```

[18]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[19]:
```
sum = x + y
sum
```

[19]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18])
```

[20]:
```
squared = x ** 2
squared
```

[20]:
```
array([ 1,  4,  9, 16, 25, 36, 49, 64, 81])
```

[21]:
```
sqrt = np.sqrt(squared)
sqrt
```

[21]:
```
array([1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

[22]:
```
z = np.exp(y)
z
```

[22]:
```
array([2.71828183e+00, 7.38905610e+00, 2.00855369e+01, 5.45981500e+01,
       1.48413159e+02, 4.03428793e+02, 1.09663316e+03, 2.98095799e+03,
       8.10308393e+03])
```

MINI CHALLENGE #3: - Given the X and Y values below, obtain the distance between them

```
X = [5, 7, 20]
Y = [9, 15, 4]
```

[23]:
```
X = np.array([5, 7, 20])
Y = np.array([9, 15, 4])
distance = np.sqrt(X ** 2 + Y ** 2)
distance
```

[23]:
```
array([10.29563014, 16.55294536, 20.39607805])
```

4 TASK #4: PERFORM ARRAYS SLICING AND INDEXING

[24]:
```
my_numpy_array = np.array([3, 5, 6, 2, 8, 10, 20, 50])
my_numpy_array
```

[24]:
```
array([ 3,  5,  6,  2,  8, 10, 20, 50])
```

[25]:
```
# Access specific index from the numpy array
my_numpy_array[0]
```

[25]:
```
3
```

[26]:
```
# Starting from the first index 0 up until and NOT including the last element
my_numpy_array[0:len(my_numpy_array) - 1]
```

[26]:
```
array([ 3,  5,  6,  2,  8, 10, 20])
```

[27]:
```
# Broadcasting, altering several values in a numpy array at once
my_numpy_array[0:4] = 7
my_numpy_array
```

[27]:
```
array([ 7,  7,  7,  7,  8, 10, 20, 50])
```

[28]:
```
# Let's define a two dimensional numpy array
matrix = np.random.randint(1, 10, (4, 4))
matrix
```

[28]:
```
array([[8, 4, 4, 2],
       [7, 7, 8, 4],
       [2, 4, 4, 9],
       [2, 4, 1, 5]])
```

[29]:
```
# Get a row from a mtrix
matrix[-1]
```

[29]:
```
array([2, 4, 1, 5])
```

[30]:
```
# Get one element
matrix[0][2]
```

[30]:
```
4
```

MINI CHALLENGE #4: - In the following matrix, replace the last row with 0

```
X = [2 30 20 -2 -4]
```

```
[3 4  40 -3 -2]
[-3 4 -6 90 10]
[25 45 34 22 12]
[13 24 22 32 37]
```

[31]:
```
X = np.array([[2, 30, 20, -2, -4],
    [3, 4,  40, -3, -2],
    [-3, 4, -6, 90, 10],
    [25, 45, 34, 22, 12],
    [13, 24, 22, 32, 37]])
X[-1 ] = 0
X
```

[31]:
```
array([[ 2, 30, 20, -2, -4],
       [ 3,  4, 40, -3, -2],
       [-3,  4, -6, 90, 10],
       [25, 45, 34, 22, 12],
       [ 0,  0,  0,  0,  0]])
```

5 TASK #5: PERFORM ELEMENTS SELECTION (CONDITIONAL)

[32]:
```
matrix = np.random.randint(1, 10,(5, 5))
matrix
```

[32]:
```
array([[8, 3, 3, 9, 8],
       [7, 3, 7, 4, 4],
       [5, 4, 1, 9, 6],
       [6, 1, 9, 8, 2],
       [2, 9, 2, 8, 6]])
```

[33]:
```
new_matrix = matrix[ matrix > 7 ]
new_matrix
```

[33]:
```
array([8, 9, 8, 9, 9, 8, 9, 8])
```

[34]:
```
# Obtain odd elements only
new_matrix = matrix[ matrix % 2 == 1]
new_matrix
```

[34]:
```
array([3, 3, 9, 7, 3, 7, 5, 1, 9, 1, 9, 9])
```

MINI CHALLENGE #5: - In the following matrix, replace negative elements by 0 and replace odd elements with -2

```
X = [2 30 20 -2 -4]
    [3 4  40 -3 -2]
```

```
[-3 4 -6 90 10]
[25 45 34 22 12]
[13 24 22 32 37]
```

[35]:
```
X = np.array([[2, 30, 20, -2, -4],
    [3, 4, 40, -3, -2],
    [-3, 4, -6, 90, 10],
    [25, 45, 34, 22, 12],
    [13, 24, 22, 32, 37]])

X[X < 0] = 0
X[X % 2 == 1] = -2
X
```

[35]:
```
array([[ 2, 30, 20,  0,  0],
       [-2,  4, 40,  0,  0],
       [ 0,  4,  0, 90, 10],
       [-2, -2, 34, 22, 12],
       [-2, 24, 22, 32, -2]])
```

6 TASK #6: UNDERSTAND PANDAS FUNDAMENTALS

[36]:
```
# Pandas is a data manipulation and analysis tool that is built on Numpy.
# Pandas uses a data structure known as DataFrame (think of it as Microsoft excel in Python).
# DataFrames empower programmers to store and manipulate data in a tabular fashion (rows and columns).
# Series Vs. DataFrame? Series is considered a single column of a DataFrame.
```

[37]:
```
import pandas as pd
```

[38]:
```
# Let's define a two-dimensional Pandas DataFrame
# Note that you can create a pandas dataframe from a python dictionary

bank_client_df = pd.DataFrame({'Bank Client ID': [ 111, 222, 333, 444],
                               'Bank Clien Name': ['Chanel', 'Steve', 'Mitch', 'Ryan'],
                              'Net Worth [$]': [3500, 2900, 1000, 2000],
                             'Years with bank': [3, 4, 9, 5]})

bank_client_df
```

[38]:

	Bank Client ID	Bank Clien Name	Net Worth [$]	Years with bank
0	111	Chanel	3500	3
1	222	Steve	2900	4
2	333	Mitch	1000	9

```
3              444             Ryan            2000                 5
```

[39]:
```
# Let's obtain the data type
type(bank_client_df)
```

[39]:
```
pandas.core.frame.DataFrame
```

[40]:
```
# you can only view the first couple of rows using .head()
bank_client_df.head(2)
```

[40]:
```
   Bank Client ID Bank Clien Name  Net Worth [$]  Years with bank
0             111          Chanel           3500                3
1             222           Steve           2900                4
```

[41]:
```
# you can only view the last couple of rows using .tail()
bank_client_df.tail(2)
```

[41]:
```
   Bank Client ID Bank Clien Name  Net Worth [$]  Years with bank
2             333           Mitch           1000                9
3             444            Ryan           2000                5
```

MINI CHALLENGE #6: - A porfolio contains a collection of securities such as stocks, bonds and ETFs. Define a dataframe named 'portfolio_df' that holds 3 different stock ticker symbols, number of shares, and price per share (feel free to choose any stocks) - Calculate the total value of the porfolio including all stocks

[42]:
```
portfolio_df = pd.DataFrame({'Stock Symbols': ['MSFT', 'SP500','BTC'],
                             'Number of shares': [0.8400093, 6.976221,0.007413],
                             'Price per Shares': [415.80, 113.42, 93724.53]})
Number_Shares = portfolio_df['Number of shares']
Price_Shares = portfolio_df['Price per Shares']

Price_Shares

Total = Number_Shares * Price_Shares
Total.sum()
```

[42]:
```
1835.2987936499999
```

7 TASK #7: PANDAS WITH CSV AND HTML DATA

[43]:
```
# Pandas is used to read a csv file and store data in a DataFrame
import pandas as pd
house_price_df = pd.read_html('https://www.livingin-canada.com/
 ↪house-prices-canada.html')
```

```
house_price_df[0]
```

[43]:

	City	Average House Price	12 Month Change
0	Vancouver, BC	$1,036,000	+ 2.63 %
1	Toronto, Ont	$870,000	+10.2 %
2	Ottawa, Ont	$479,000	+ 15.4 %
3	Calgary, Alb	$410,000	- 1.5 %
4	Montreal, Que	$435,000	+ 9.3 %
5	Halifax, NS	$331,000	+ 3.6 %
6	Regina, Sask	$254,000	- 3.9 %
7	Fredericton, NB	$198,000	- 4.3 %
8	(adsbygoogle = window.adsbygoogle || []).push(…	(adsbygoogle = window.adsbygoogle || []).push(…	(adsbygoogle = window.adsbygoogle || []).push(…

[44]:
```
house_price_df[1]
```

[44]:

	Province
0	British Columbia
1	Ontario
2	Alberta
3	Quebec
4	Manitoba
5	Saskatchewan
6	Nova Scotia
7	Prince Edward Island
8	Newfoundland / Labrador

```
9                                        New Brunswick
10                                     Canadian Average
11  (adsbygoogle = window.adsbygoogle || []).push(…

                                    Average House Price  \
0                                              $736,000
1                                              $594,000
2                                              $353,000
3                                              $340,000
4                                              $295,000
5                                              $271,000
6                                              $266,000
7                                              $243,000
8                                              $236,000
9                                              $183,000
10                                             $488,000
11  (adsbygoogle = window.adsbygoogle || []).push(…

                                       12 Month Change
0                                               + 7.6 %
1                                               - 3.2 %
2                                               - 7.5 %
3                                               + 7.6 %
4                                               - 1.4 %
5                                               - 3.8 %
6                                               + 3.5 %
7                                               + 3.0 %
8                                               - 1.6 %
9                                               - 2.2 %
10                                              - 1.3 %
11  (adsbygoogle = window.adsbygoogle || []).push(…
```

[45]:
```
# Read tabular data using read_html
```

[ ]:
```

```

[ ]:
```

```

MINI CHALLENGE #7: - Write a code that uses Pandas to read tabular US retirement data - You can use data from here: https://www.ssa.gov/oact/progdata/nra.html

[46]:
```
Us_retirement_df = pd.read_html('https://www.ssa.gov/oact/progdata/nra.html')
Us_retirement_df[0]
```

[46]:
```
                                    Year of birth  \
0                                   1937 and prior
1                                             1938
```

```
2                                                  1939
3                                                  1940
4                                                  1941
5                                                  1942
6                                                1943-54
7                                                  1955
8                                                  1956
9                                                  1957
10                                                 1958
11                                                 1959
12                                      1960 and later
13  Notes: 1. Persons born on January 1 of any yea…

                                                     Age
0                                                     65
1                                       65 and 2 months
2                                       65 and 4 months
3                                       65 and 6 months
4                                       65 and 8 months
5                                      65 and 10 months
6                                                     66
7                                       66 and 2 months
8                                       66 and 4 months
9                                       66 and 6 months
10                                      66 and 8 months
11                                     66 and 10 months
12                                                    67
13  Notes: 1. Persons born on January 1 of any yea…
```

8 TASK #8: PANDAS OPERATIONS

[47]:
```
# Let's define a dataframe as follows:
bank_client_df = pd.DataFrame({'Bank Client ID': [ 111, 222, 333, 444],
                               'Bank Clien Name': ['Chanel', 'Steve', 'Mitch',␣
 ↪'Ryan'],
                              'Net Worth [$]': [3500, 29000, 10000, 2000],
                             'Years with bank': [3, 4, 9, 5]})

bank_client_df
```

[47]:

	Bank Client ID	Bank Clien Name	Net Worth [$]	Years with bank
0	111	Chanel	3500	3
1	222	Steve	29000	4
2	333	Mitch	10000	9
3	444	Ryan	2000	5

[48]:
```
# Pick certain rows that satisfy a certain criteria
df_loyal = bank_client_df[ bank_client_df['Years with bank'] >= 5]
df_loyal
```

[48]:

	Bank Client ID	Bank Clien Name	Net Worth [$]	Years with bank
2	333	Mitch	10000	9
3	444	Ryan	2000	5

[49]:
```
# Delete a column from a DataFrame
del bank_client_df['Bank Client ID']
bank_client_df
```

[49]:

	Bank Clien Name	Net Worth [$]	Years with bank
0	Chanel	3500	3
1	Steve	29000	4
2	Mitch	10000	9
3	Ryan	2000	5

MINI CHALLENGE #8: - Using "bank_client_df" DataFrame, leverage pandas operations to only select high networth individuals with minimum $5000 - What is the combined networth for all customers with 5000+ networth?

[51]:
```
df_network = bank_client_df[ bank_client_df['Net Worth [$]'] >= 5000]
df_network

total_network = df_network['Net Worth [$]'].sum()
total_network
```

[51]: 39000

9 TASK #9: PANDAS WITH FUNCTIONS

[ ]:
```
# Let's define a dataframe as follows:
bank_client_df = pd.DataFrame({'Bank client ID':[111, 222, 333, 444],
                               'Bank Client Name':['Chanel', 'Steve', 'Mitch',␣
 ↪'Ryan'],
                               'Net worth [$]':[3500, 29000, 10000, 2000],
                               'Years with bank':[3, 4, 9, 5]})
bank_client_df
```

[ ]:
```
# Define a function that increases all clients networth (stocks) by a fixed␣
 ↪value of 20% (for simplicity sake)
def increase_networth(balance):
    return balance * 1.2
```

[ ]:
```
# You can apply a function to the DataFrame
bank_client_df['Net Worth [$]'].apply(increase_networth)
```

[ ]:
```
bank_client_df['Bank Client Name'].apply(len)
```

MINI CHALLENGE #9: - Define a function that triples the stock prices and adds $200 - Apply the function to the DataFrame - Calculate the updated total networth of all clients combined

[ ]:
```
def triple_stock(value):
    return value * 3.0 + 200

triple_stock = bank_client_df['Net worth [$]'].apply(triple_stock)
triple_stock.sum()
```

10 TASK #10: PERFORM SORTING AND ORDERING IN PANDAS

[52]:
```
# Let's define a dataframe as follows:
bank_client_df = pd.DataFrame({'Bank client ID':[111, 222, 333, 444],
                               'Bank Client Name':['Chanel', 'Steve', 'Mitch',␣
 ↪'Ryan'],
                               'Net worth [$]':[3500, 29000, 10000, 2000],
                               'Years with bank':[3, 4, 9, 5]})
bank_client_df
```

[52]:

	Bank client ID	Bank Client Name	Net worth [$]	Years with bank
0	111	Chanel	3500	3
1	222	Steve	29000	4
2	333	Mitch	10000	9
3	444	Ryan	2000	5

[53]:
```
# You can sort the values in the dataframe according to number of years with␣
 ↪bank
bank_client_df.sort_values(by = 'Years with bank')
```

[53]:

	Bank client ID	Bank Client Name	Net worth [$]	Years with bank
0	111	Chanel	3500	3
1	222	Steve	29000	4
3	444	Ryan	2000	5
2	333	Mitch	10000	9

[54]:
```
# Note that nothing changed in memory! you have to make sure that inplace is␣
 ↪set to True
bank_client_df
```

[54]:

	Bank client ID	Bank Client Name	Net worth [$]	Years with bank
0	111	Chanel	3500	3
1	222	Steve	29000	4
2	333	Mitch	10000	9
3	444	Ryan	2000	5

[55]:
```
# Set inplace = True to ensure that change has taken place in memory
bank_client_df.sort_values(by = 'Years with bank', inplace = True)
```

[56]:
```
# Note that now the change (ordering) took place
bank_client_df
```

[56]:

	Bank client ID	Bank Client Name	Net worth [$]	Years with bank
0	111	Chanel	3500	3
1	222	Steve	29000	4
3	444	Ryan	2000	5
2	333	Mitch	10000	9

11 TASK #11: PERFORM CONCATENATING AND MERGING WITH PANDAS

[ ]:
```
# Check this out: https://pandas.pydata.org/pandas-docs/stable/user_guide/
↪merging.html
```

[57]:
```
df1 = pd.DataFrame({ 'A': ['A0', 'A1', 'A2', 'A3'],
                     'B': ['B0', 'B1', 'B2', 'B3'],
                     'C': ['C0', 'C1', 'C2', 'C3'],
                     'D': ['D0', 'D1', 'D2', 'D3']},
                    index  = [0,1,2,3])
```

[58]:
```
df1
```

[58]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

[59]:
```
df2 = pd.DataFrame({ 'A': ['A4', 'A5', 'A6', 'A7'],
                     'B': ['B4', 'B5', 'B6', 'B7'],
                     'C': ['C4', 'C5', 'C6', 'C7'],
                     'D': ['D4', 'D5', 'D6', 'D7']},
                    index  = [4,5,6,7])
```

[60]:
```
df2
```

[60]:

	A	B	C	D
0	A4	B4	C4	D4
1	A5	B5	C5	D5
2	A6	B6	C6	D6
3	A7	B7	C7	D7

[62]:
```
df3 = pd.DataFrame({ 'A': ['A8', 'A9', 'A10', 'A11'],
                     'B': ['B8', 'B9', 'B10', 'B11'],
                     'C': ['C8', 'C9', 'C10', 'C11'],
                     'D': ['D8', 'D9', 'D10', 'D11']},
                    index  = [8,9,10,11])
```

[63]:
```
df3
```

[63]:

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

[65]:
```
pd.concat([df1, df2, df3])
```

[65]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
0	A4	B4	C4	D4
1	A5	B5	C5	D5
2	A6	B6	C6	D6
3	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

12 TASK #12: PROJECT AND CONCLUDING REMARKS

- Define a dataframe named 'Bank_df_1' that contains the first and last names for 5 bank clients with IDs = 1, 2, 3, 4, 5
- Assume that the bank got 5 new clients, define another dataframe named 'Bank_df_2' that contains a new clients with IDs = 6, 7, 8, 9, 10
- Let's assume we obtained additional information (Annual Salary) about all our bank customers (10 customers)
- Concatenate both 'bank_df_1' and 'bank_df_2' dataframes
- Merge client names and their newly added salary information using the 'Bank Client ID'
- Let's assume that you became a new client to the bank

- Define a new DataFrame that contains your information such as client ID (choose 11), first name, last name, and annual salary.
- Add this new dataframe to the original dataframe 'bank_df_all'.

[125]:
```
bank_df_1 = pd.DataFrame({'Bank Client ID': [1,2,4,5],
                          'First Name': ['Emily', 'Alex','Sarah','Michael'],
                          'Last Name': ['Thompson', 'Johson',␣
 ↪'Robinson','Ngueyn']})
```

[128]:
```
bank_df_2 = pd.DataFrame({
    'Bank Client ID': [6, 7, 8, 9, 10],
    'First Name': ['Daniel', 'Sophia', 'James', 'Olivia', 'Liam'],
    'Last Name': ['Smith', 'Brown', 'Taylor', 'Garcia', 'Clark']
})

raw_data = {'Bank Client ID': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
            'Annual Salary': [2500, 3500, 4500, 4800, 3200, 3300, 3400, 2300,␣
 ↪2200, 35000]}

bank_df_salary = pd.DataFrame(raw_data, columns = ['Bank Client ID', 'Annual␣
 ↪Salary'])
bank_df_salary
```

[128]:

	Bank Client ID	Annual Salary
0	1	2500
1	2	3500
2	3	4500
3	4	4800
4	5	3200
5	6	3300
6	7	3400
7	8	2300
8	9	2200
9	10	35000

[129]:
```
bank_df_all = pd.concat([bank_df_1, bank_df_2])
bank_df_all
```

[129]:

	Bank Client ID	First Name	Last Name
0	1	Emily	Thompson
1	2	Alex	Johson
2	4	Sarah	Robinson
3	5	Michael	Ngueyn
0	6	Daniel	Smith
1	7	Sophia	Brown
2	8	James	Taylor
3	9	Olivia	Garcia

```
4              10       Liam     Clark
```

[130]:
```
Bank_df_all = pd.merge(bank_df_all, bank_df_salary, on = 'Bank Client ID')
Bank_df_all
```

[130]:

	Bank Client ID	First Name	Last Name	Annual Salary
0	1	Emily	Thompson	2500
1	2	Alex	Johson	3500
2	4	Sarah	Robinson	4800
3	5	Michael	Ngueyn	3200
4	6	Daniel	Smith	3300
5	7	Sophia	Brown	3400
6	8	James	Taylor	2300
7	9	Olivia	Garcia	2200
8	10	Liam	Clark	35000

[131]:
```
new_client = pd.DataFrame({'Bank Client ID': [11],
                           'First Name': ['Kevin'],
                           'Last Name': ['Truong'],
                           'Annual Salary': [1000]})

bank_df_all = pd.concat([Bank_df_all, new_client], ignore_index = True)
bank_df_all
```

[131]:

	Bank Client ID	First Name	Last Name	Annual Salary
0	1	Emily	Thompson	2500
1	2	Alex	Johson	3500
2	4	Sarah	Robinson	4800
3	5	Michael	Ngueyn	3200
4	6	Daniel	Smith	3300
5	7	Sophia	Brown	3400
6	8	James	Taylor	2300
7	9	Olivia	Garcia	2200
8	10	Liam	Clark	35000
9	11	Kevin	Truong	1000

13 EXCELLENT JOB!

14 MINI CHALLENGES SOLUTIONS

MINI CHALLENGE #1 SOLUTION: - Write a code that creates the following 2x4 numpy array

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

[ ]:
```
x = np.array([[[3, 7, 9, 3] , [4, 3, 2, 2]]])
x
```

MINI CHALLENGE #2 SOLUTION: - Write a code that takes in a positive integer "x" from the user and creates a 1x10 array with random numbers ranging from 0 to "x"

[ ]:
```
x = int(input("Please enter a positive integer value: "))
x = np.random.randint(1, x, 10)
x
```

[ ]:
```

```

MINI CHALLENGE #3 SOLUTION: - Given the X and Y values below, obtain the distance between them

```
X = [5, 7, 20]
Y = [9, 15, 4]
```

[ ]:
```
X = np.array([5, 7, 20])
Y = np.array([9, 15, 4])
Z = np.sqrt(X**2 + Y**2)
Z
```

MINI CHALLENGE #4 SOLUTION: - In the following matrix, replace the last row with 0

```
X = [2 30 20 -2 -4]
    [3 4  40 -3 -2]
    [-3 4 -6 90 10]
    [25 45 34 22 12]
    [13 24 22 32 37]
```

[ ]:
```
X = np.array([[2, 30, 20, -2, -4],
    [3, 4,  40, -3, -2],
    [-3, 4, -6, 90, 10],
    [25, 45, 34, 22, 12],
    [13, 24, 22, 32, 37]])
```

[ ]:
```
X[4] = 0
X
```

MINI CHALLENGE #5 SOLUTION: - In the following matrix, replace negative elements by 0 and replace odd elements with -2

```
X = [2 30 20 -2 -4]
    [3 4  40 -3 -2]
    [-3 4 -6 90 10]
    [25 45 34 22 12]
    [13 24 22 32 37]
```

[ ]:
```
X = np.array([[2, 30, 20, -2, -4],
    [3, 4,  40, -3, -2],
    [-3, 4, -6, 90, 10],
    [25, 45, 34, 22, 12],
```

```
    [13, 24, 22, 32, 37]])

X[X<0] = 0
X[X%2==1] = -2
X
```

MINI CHALLENGE #6 SOLUTION: - A porfolio contains a collection of securities such as stocks, bonds and ETFs. Define a dataframe named 'portfolio_df' that holds 3 different stock ticker symbols, number of shares, and price per share (feel free to choose any stocks) - Calculate the total value of the porfolio including all stocks

[ ]:
```
portfolio_df = pd.DataFrame({'stock ticker symbols':['AAPL', 'AMZN', 'T'],
                             'price per share [$]':[3500, 200, 40],
                             'Number of stocks':[3, 4, 9]})
portfolio_df
```

[ ]:
```
stocks_dollar_value = portfolio_df['price per share [$]'] *␣
 ↪portfolio_df['Number of stocks']
print(stocks_dollar_value)
print('Total portfolio value = {}'.format(stocks_dollar_value.sum()))
```

MINI CHALLENGE #7 SOLUTION: - Write a code that uses Pandas to read tabular US retirement data - You can use data from here: https://www.ssa.gov/oact/progdata/nra.html

[ ]:
```
# Read tabular data using read_html
retirement_age_df = pd.read_html('https://www.ssa.gov/oact/progdata/nra.html')
retirement_age_df
```

MINI CHALLENGE #8 SOLUTION: - Using "bank_client_df" DataFrame, leverage pandas operations to only select high networth individuals with minimum $5000 - What is the combined networth for all customers with 5000+ networth?

[ ]:
```
df_high_networth = bank_client_df[ (bank_client_df['Net worth [$]'] >= 5000) ]
df_high_networth
```

[ ]:
```
df_high_networth['Net worth [$]'].sum()
```

MINI CHALLENGE #9 SOLUTION: - Define a function that triples the stock prices and adds $200 - Apply the function to the DataFrame - Calculate the updated total networth of all clients combined

[ ]:
```
def networth_update(balance):
    return balance * 3 + 200
```

[ ]:
```
# You can apply a function to the DataFrame
results = bank_client_df['Net worth [$]'].apply(networth_update)
results
```

[ ]:
```
results.sum()
```

PROJECT SOLUTION:

[ ]:
```
# Creating a dataframe from a dictionary
# Let's define a dataframe with a list of bank clients with IDs = 1, 2, 3, 4, 5

raw_data = {'Bank Client ID': ['1', '2', '3', '4', '5'],
            'First Name': ['Nancy', 'Alex', 'Shep', 'Max', 'Allen'],
            'Last Name': ['Rob', 'Ali', 'George', 'Mitch', 'Steve']}

Bank_df_1 = pd.DataFrame(raw_data, columns = ['Bank Client ID', 'First Name',␣
 ↪'Last Name'])
Bank_df_1


# Let's define another dataframe for a separate list of clients (IDs = 6, 7, 8,␣
 ↪9, 10)
raw_data = {
        'Bank Client ID': ['6', '7', '8', '9', '10'],
        'First Name': ['Bill', 'Dina', 'Sarah', 'Heather', 'Holly'],
        'Last Name': ['Christian', 'Mo', 'Steve', 'Bob', 'Michelle']}
Bank_df_2 = pd.DataFrame(raw_data, columns = ['Bank Client ID', 'First Name',␣
 ↪'Last Name'])
Bank_df_2


# Let's assume we obtained additional information (Annual Salary) about our␣
 ↪bank customers
# Note that data obtained is for all clients with IDs 1 to 10
raw_data = {
        'Bank Client ID': ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10'],
        'Annual Salary [$/year]': [25000, 35000, 45000, 48000, 49000, 32000,␣
 ↪33000, 34000, 23000, 22000]}
bank_df_salary = pd.DataFrame(raw_data, columns = ['Bank Client ID','Annual␣
 ↪Salary [$/year]'])
bank_df_salary


# Let's concatenate both dataframes #1 and #2
# Note that we now have client IDs from 1 to 10
bank_df_all = pd.concat([Bank_df_1, Bank_df_2])
bank_df_all


# Let's merge all data on 'Bank Client ID'
bank_df_all = pd.merge(bank_df_all, bank_df_salary, on = 'Bank Client ID')
```

```
bank_df_all
```

[ ]:
```
new_client = {
        'Bank Client ID': ['11'],
        'First Name': ['Ry'],
        'Last Name': ['Aly'],
        'Annual Salary [$/year]' : [1000]}
new_client_df = pd.DataFrame(new_client, columns = ['Bank Client ID', 'First␣
 ↪Name', 'Last Name', 'Annual Salary [$/year]'])
new_client_df
```

[ ]:
```
new_df = pd.concat([bank_df_all, new_client_df], axis = 0)
new_df
```