Python Libraries

For Data Science

Last lecture reminder

We learned about:

- Numpy array methods
- Numpy arithmetic methods (min, max, argmin, argmax, sum, mean)
- Deep copy in Numpy arrays
- Conditional selection in Numpy Arrays
- Introduction to Pandas library
- Pandas Series functions and methods

Pandas DataFrame Introduction

Until now we only use one data structure Pandas provide called Series, but Series can be limited because we can only using it to store 1 value as the data and 1 label as the key.
In real life we will need more complex data structure that can hold multiple values for a given key (like a table with columns and rows) and for that we need the DataFrame data structure.

A **Pandas DataFrame** is a two-dimensional, size-mutable data structure with labelled axes (rows and columns). The DataFrame can be used for a lot of different tasks such as data cleaning, data transformation, statistical analysis, and machine learning modeling. It is designed to handle a wide variety of data types such as integers, floats, strings and also more complex structures like lists and dictionaries.

Pandas is using the Series to represent each column in the dataframe, meaning dataframe can be considered as a list of Series with the same keys.

Creating Basic DataFrame

pd.DataFrame() → Creating a new Pandas Dataframe from the given data source. We can pass to the DataFrame() method a matrix and it will create a Dataframe built from the matrix values.

```
In [5]: np.random.seed(100)
        my_data = np.random.randint(0,101,(4,3))
        my_df = pd.DataFrame(my_data)

        print(my_df)
```

```
    0   1   2
0   8  24  67
1  87  79  48
2  10  94  52
3  98  53  66
```

We are passing 4x3 matrix of random numbers as the dataframe data source

When printing the df we are getting default numeric indexes on the df columns and on the df rows

We can also provide our own labels to the df by providing '**index**' parameter.

```
In [7]: my_data = np.random.randint(0,101,(4,3))
        my_index = ['CA', 'NY', 'AZ', 'TX']
        my_df = pd.DataFrame(data=my_data, index=my_index)

        print(my_df)
```

```
     0   1    2
CA  98  14   34
NY  24  15  100
AZ  60  58   16
TX   9  93   86
```

We are passing a list of labels that will be handled as the row indexes in addition to the default numeric indexes. The order of the labels in the list determine the order of the index each row is getting

Creating Basic DataFrame

In addition to the row index, we can also provide column labeling (those are not indexes).

By providing the '**columns'** parameter to the Dataframe() method we will be able to determine the label of each column in our Dataframe.

```
In [8]: my_data = np.random.randint(0,101,(4,3))
        my_index = ['CA', 'NY', 'AZ', 'TX']
        my_cols = ['Jan', 'Feb', 'Mar']
        my_df = pd.DataFrame(data=my_data, index=my_index, columns=my_cols)

        print(my_df)
```

```
    Jan  Feb  Mar
CA    2   27    4
NY   31    1   13
AZ   83  100    4
TX   91   59   67
```

We are passing a list of labels that will be handled as the columns labels in addition to the default numeric indexes. The order of the labels in the list determine the order of the label each column is getting

Note: As we already saw in Pandas Series, when we adding another label indexing to the df rows or to the df columns we are not replacing the default numeric indexes but adding another type of index beside them.

Excel to Dataframe

Because of the table like structure Dataframes can be used to represent any SQL table or Excel file.

In order to allow that Pandas provides us a quick and easy way to turn any Dataframe into excel file and to load any excel file into a Dataframe.

pd.read_csv() → This Pandas method allow as to take any CSV file saved on our computer and turn it into a Pandas df. We need to pass to this method the full directory of the CSV file that we want to turn into a df.

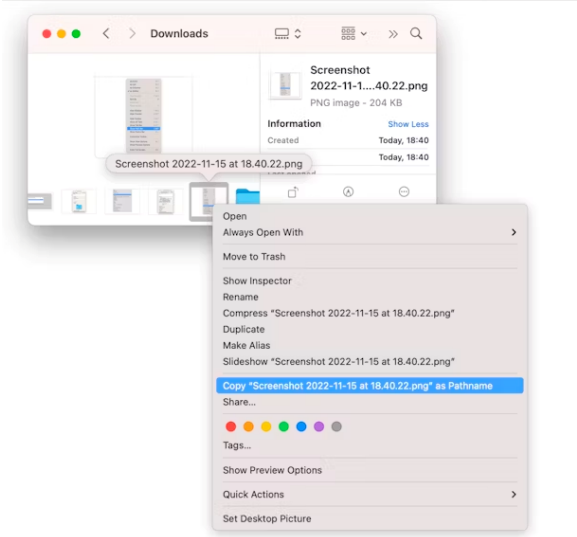
```
In [13]: excel_df = pd.read_csv('/Users/ben.meir/Downloads/UNZIP_FOR_NOTEBOOKS_FINAL/03-Pandas/example.csv')
         print(excel_df)

    a   b   c   d
0   0   1   2   3
1   4   5   6   7
2   8   9  10  11
3  12  13  14  15
```

We are passing the full directory path of the CSV file we want to create as a dataframe

Finding File Full Directory - Mac

For Mac users:

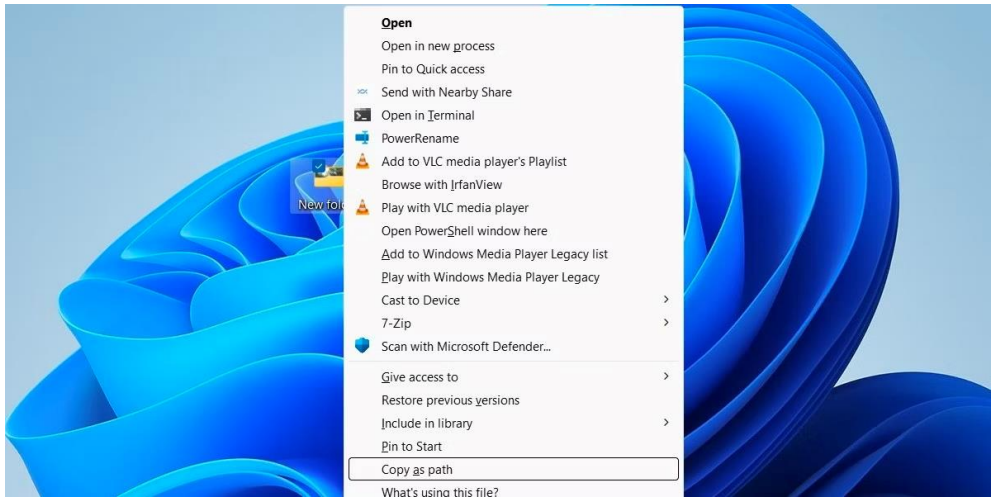
1. Open Finder and search for the CSV file you want to use
2. Select the file and press **right-click** on your mouse
3. While selected press on the “**option**” keyboard button
4. Select “**Copy as path name**”



Finding File Full Directory - Windows

For Windows users:

1. Navigate to your CSV file folder
2. Press and hold the “**Shift**” keyboard button
3. While pressing, press the **right-click** on your mouse
4. Select “**Copy as path**”



Basic DataFrame Functions

df.columns → Return all the Dataframe columns as a Pandas list object.

df.index → Return all the Dataframe indexes as a Pandas list object.

```
In [20]: print(excel_df.columns)
         print()
         print(excel_df.index)

         Index(['a', 'b', 'c', 'd'], dtype='object')

         RangeIndex(start=0, stop=4, step=1)
```

df.head() → Return the first number of rows in the Dataframe that we passed to this method as parameter. For example → df.head(2) will return the first 2 rows in the Dataframe

```
In [23]: excel_df.head(2)
```

Out[23]:

	a	b	c	d
0	0	1	2	3
1	4	5	6	7

Basic DataFrame Functions

df.tail() → Return the last number of rows in the Dataframe that we passed to this method as parameter. For example → df.tail(2) will return the last 2 rows in the Dataframe.

```
In [24]: excel_df.tail(2)
```

Out[24]:

	a	b	c	d
2	8	9	10	11
3	12	13	14	15

df.info() → Will return basic information about the Dataframe, such as the label and type of each column and the amount of memory this Dataframe is using.

```
In [26]: excel_df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   a       4 non-null      int64
 1   b       4 non-null      int64
 2   c       4 non-null      int64
 3   d       4 non-null      int64
dtypes: int64(4)
memory usage: 256.0 bytes
```

Basic DataFrame Functions

df.describe() → Return basic statistic information about each column in the Dataframe, such as the mean, std, min, max, percentile etc…

```
In [27]: excel_df.describe()
```

Out[27]:

	a	b	c	d
count	4.000000	4.000000	4.000000	4.000000
mean	6.000000	7.000000	8.000000	9.000000
std	5.163978	5.163978	5.163978	5.163978
min	0.000000	1.000000	2.000000	3.000000
25%	3.000000	4.000000	5.000000	6.000000
50%	6.000000	7.000000	8.000000	9.000000
75%	9.000000	10.000000	11.000000	12.000000
max	12.000000	13.000000	14.000000	15.000000

df.describe().transpose() → Changing the describe() information order, The rows will be the columns names and the columns will be the statistic information.

```
In [29]: excel_df.describe().transpose()
```

Out[29]:

	count	mean	std	min	25%	50%	75%	max
a	4.0	6.0	5.163978	0.0	3.0	6.0	9.0	12.0
b	4.0	7.0	5.163978	1.0	4.0	7.0	10.0	13.0
c	4.0	8.0	5.163978	2.0	5.0	8.0	11.0	14.0
d	4.0	9.0	5.163978	3.0	6.0	9.0	12.0	15.0

Working With Dataframes

df[${column name}] → Return a Dataframe with only the column we passed inside the []

```
In [31]: excel_df['a']

Out[31]: 0     0
         1     4
         2     8
         3    12
         Name: a, dtype: int64
```

We can select multiple columns by passing a Python list containing the columns names in the []

```
In [34]: excel_df[['a', 'c','b']]

Out[34]:
```

	a	c	b
0	0	2	1
1	4	6	5
2	8	10	9
3	12	14	13

We also can combine columns together if they are from the same type using the **'+'** operator

```
In [35]: excel_df['a'] + excel_df['b']

Out[35]: 0     1
         1     9
         2    17
         3    25
         dtype: int64
```

We added together the numbers from the 'a' column and the 'b' column

Working With Dataframes

df[${new column name}] → By providing a new column name to the [] we will add this column to the existing Dataframe. If we provide existing column name with new values it will override the previous column values.

```
In [41]: excel_df['calculation'] = 10 * excel_df['a'] / excel_df['b']
         excel_df
```

Out[41]:

	a	b	c	d	calculation
0	0	1	2	3	0.000000
1	4	5	6	7	8.000000
2	8	9	10	11	8.888889
3	12	13	14	15	9.230769

df.drop() → Removing existing column or row from the Dataframe. We will need to pass the column name or the row index that we want to remove from the Dataframe.

Using the drop() method won't change the original Dataframe but just the printed result. If we will want to really remove the column from the original Dataframe we will need to pass the **'inplace'** parameter as true.
Another option is to save the new Dataframe that returned from the drop() method into a variable.

Working With Dataframes

df.drop() → The 'axis' parameter determine what we want to remove from the Dataframe:

- axis = 0 → removing a row from the Dataframe with the provided index name
- axis = 1 → removing a column from the Dataframe with the provided column name

```
In [47]: excel_df.drop(0, axis=0)
```

Out[47]:

	a	b	c	d	calculation
1	4	5	6	7	8.000000
2	8	9	10	11	8.888889
3	12	13	14	15	9.230769

We are passing axis=0 meaning we want to delete the row with the 0 index

```
In [56]: excel_df.drop('calculation', axis=1, inplace=True)
         excel_df
```

Out[56]:

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

We are passing axis=1 meaning we want to delete the column with the 'calculation' label.
In addition the change will effect the original Dataframe

Class Exercise - Dataframe Functions

Instructions:

For this class exercise we will use the tips.csv file that should look like that:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325168603410	Sun2959
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071379779230	Sun4608
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	6011812112971322	Sun4458
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676137647685994	Sun5260
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	4832732618637221	Sun2251
...	...	...	...	...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3	9.68	Michael Avila	5296068606052842	Sat2657
240	27.18	2.00	Female	Yes	Sat	Dinner	2	13.59	Monica Sanders	3506806155565404	Sat1766
241	22.67	2.00	Male	Yes	Sat	Dinner	2	11.34	Keith Wong	6011891618747196	Sat3880
242	17.82	1.75	Male	No	Sat	Dinner	2	8.91	Dennis Dixon	4375220550950	Sat17
243	18.78	3.00	Female	No	Thur	Dinner	2	9.39	Michelle Hardin	3511451626698139	Thur672

244 rows × 11 columns

Class Exercise - Dataframe Functions

Instructions:

Open the class exercise CSV file called '**tips.csv**' and implement the following:

- Import the CSV file into a Pandas Dataframe and save it in a new variable.
- Create a new Dataframe contain only the columns - Payer Name, tips & total_bill and save it to a new variable
- Add to the new Dataframe two new columns - 'total_payment' which should calculate for each payer the total_bill + the tip amount

  'tip_precentage' which should calculate the percentage of the tip from the total payment
- Update your new Dataframe by removing the two columns you added, the remove should affect the Dataframe itself and not just the print result

Class Exercise Solution - Dataframe Functions

Working With Dataframes

In the following examples we will keep using the ‘tips.csv’ file.

df.set_index() → Allow us to add additional index to the rows in our Dataframe. For example we can set the ‘Payment ID’ column as the Dataframe index. In order to do that we just need to pass the column name as parameter to the set_index() method.

Note: Once we are using a column as index it will not appear at the Dataframe

```
In [58]: tips_df.set_index('Payment ID')
```

Out[58]:

Payment ID	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number
Sun2959	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325168603410
Sun4608	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071379779230
Sun4458	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	6011812112971322
Sun5260	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676137647685994
Sun2251	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	4832732618637221
...	...	...	...	...	...	...	...	...	...	...
Sat2657	29.03	5.92	Male	No	Sat	Dinner	3	9.68	Michael Avila	5296068606052842
Sat1766	27.18	2.00	Female	Yes	Sat	Dinner	2	13.59	Monica Sanders	3506806155565404
Sat3880	22.67	2.00	Male	Yes	Sat	Dinner	2	11.34	Keith Wong	6011891618747196
Sat17	17.82	1.75	Male	No	Sat	Dinner	2	8.91	Dennis Dixon	4375220550950
Thur672	18.78	3.00	Female	No	Thur	Dinner	2	9.39	Michelle Hardin	3511451626698139

244 rows × 10 columns

Working With Dataframes

df.reset_index() → Will remove the existing indexing and will return it to appears as a Dataframe column again.

df.iloc[] → Will return the data of a specific row according to numeric index or label index if its exist

```
In [59]: tips_df.iloc[0]

Out[59]: total_bill                      16.99
         tip                              1.01
         sex                            Female
         smoker                             No
         day                               Sun
         time                           Dinner
         size                                2
         price_per_person                 8.49
         Payer Name          Christy Cunningham
         CC Number             3560325168603410
         Payment ID                     Sun2959
         Name: 0, dtype: object
```

df.loc[] → Will return the data of a specific row according to labeled index or label index if its exist

```
In [75]: tips_df.loc['Sun2959']

Out[75]: total_bill                      16.99
         tip                              1.01
         sex                            Female
         smoker                             No
         day                               Sun
         time                           Dinner
         size                                2
         price_per_person                 8.49
         Payer Name          Christy Cunningham
         CC Number             3560325168603410
         Name: Sun2959, dtype: object
```

Working With Dataframes

We can get multiple rows using the iloc[] method by providing the range of the row's index we want to get. For example → **df.iloc[0:4]** → will return the the Dataframe rows from index 0 to index 4.

In Addition, we can get multiple rows using the loc[] method by providing Python list with all the row's index we want to get. For example → **df.loc[[Sun2959', Sun2876', Sun5476']]**

pd.concat() → Concatenate two Dataframes together, if the columns numbers and names are not matched between the two Dataframes Pandas will populate the missing values with NaN.
We can also use the **df.append()** method to add a new row to existing Dataframe but this method will be deprecated in the future so its not recommended to use it.

```
In [87]: first_payer = tips_df.head(1)
         last_payer = tips_df.tail(1)
         concat_df = pd.concat([first_payer, last_payer])
         concat_df
```

Out[87]:

Payment ID	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number
Sun2959	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325168603410
Thur672	18.78	3.00	Female	No	Thur	Dinner	2	9.39	Michelle Hardin	3511451626698139

Dataframes Conditional Filtering

By using **conditional filtering** we can get specific rows that match the condition we are passing. This can be very useful when we want to analyze Dataframe data.

Let's say we have this Dataframe and we want to filter all rows with 'population' above 50:

```
In [92]: countries = ['USA', 'Canada', 'Mexico']
         years = [1776, 1867, 1821]
         population = [328, 38, 126]
         gdp = [20.5, 1.7, 1.22]
         countries_df = pd.DataFrame({
             'Country': countries,
             'Year': years,
             'Population': population,
             'GDP': gdp
         })
         countries_df.set_index('Country')
```

Out[92]:

Country	Year	Population	GDP
USA	1776	328	20.50
Canada	1867	38	1.70
Mexico	1821	126	1.22

By executing this filter conditioning we will get the results we want → df['population'] > 50

```
In [93]: countries_df['Population'] > 50

Out[93]: 0     True
         1    False
         2     True
         Name: Population, dtype: bool
```

We got back a Series with true and false according to the condition we requested

Dataframes Conditional Filtering

Now, in order to apply the conditional filtering on the Dataframe itself we will just need to pass this boolean Series as the column we want to get from the df.

```
In [96]: countries_df[countries_df['Population'] > 50]
```

Out[96]:

	Country	Year	Population	GDP
0	USA	1776	328	20.50
2	Mexico	1821	126	1.22

We can also apply multiple conditions by using the And \ Or \ Not operators:

& → The Dataframe And operator

| → The Dataframe Or operator

~ → The Not operator

Dataframes Conditional Filtering

For example → let's get back all the countries with population above 50 and that the country name is or 'USA'

or

```
In [99]: countries_df[(countries_df['Population'] > 50) & (countries_df['Country'].isin(['Canada', 'USA']))]
```

Out[99]:

	Country	Year	Population	GDP
0	USA	1776	328	20.5

```
In [101]: countries_df[~countries_df['Year'].isin([1776])]
```

s not 1776

Out[101]:

	Country	Year	Population	GDP
1	Canada	1867	38	1.70
2	Mexico	1821	126	1.22

ECOM SCHOOL

המכללה למקצועות הדיגיטל וההייטק

Dataframes Conditional Filtering

df.between() → Will filter only the rows in the Dataframe that their values are between the values provided.

We can also pass the **'inclusive'** parameter in case we want to include the last value as well.

```
In [66]: tips_df[tips_df['total_bill'].between(10,20,inclusive=True)]
```

```
/var/folders/9z/4gkdx92s2nl5l9xd0nmpb6000000gp/T/ipykernel_6231/3120278632.py:1: FutureWarning: Boolean inputs to the
`inclusive` argument are deprecated in favour of `both` or `neither`.
  tips_df[tips_df['total_bill'].between(10,20,inclusive=True)]
```

Out[66]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID	quality
0	16.99	1.01	F	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325168603410	Sun2959	Not Generous
1	10.34	1.66	M	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071379779230	Sun4608	Not Generous
8	15.04	1.96	M	No	Sun	Dinner	2	7.52	Joseph Mcdonald	3522866365840377	Sun6820	Not Generous
9	14.78	3.23	M	No	Sun	Dinner	2	7.39	Jerome Abbott	3532124519049786	Sun3775	Not Generous
10	10.27	1.71	M	No	Sun	Dinner	2	5.14	William Riley	566287581219	Sun2546	Not Generous
...	...	...	...	...	...	...	...	...	...	...	...	...
234	15.53	3.00	M	Yes	Sat	Dinner	2	7.76	Tracy Douglas	4097938155941930	Sat7220	Not Generous
235	10.07	1.25	M	No	Sat	Dinner	2	5.04	Sean Gonzalez	3534021246117605	Sat4615	Not Generous
236	12.60	1.00	M	Yes	Sat	Dinner	2	6.30	Matthew Myers	3543676378973965	Sat5032	Not Generous
242	17.82	1.75	M	No	Sat	Dinner	2	8.91	Dennis Dixon	4375220550950	Sat17	Not Generous
243	18.78	3.00	F	No	Thur	Dinner	2	9.39	Michelle Hardin	3511451626698139	Thur672	Not Generous

130 rows × 12 columns

We are filtering all rows in the Dataframe that their total_bill values are between 10 - 20 including 20

Class Exercise - Dataframe Conditional Filtering

Instructions:

Open the class exercise CSV file called '**tips.csv**' and implement the following:

- Import the CSV file into a Pandas Dataframe and save it in a new variable.
- Apply conditional filtering that get back all the buyers with 'total_bill' above 30 and that their 'sex' is 'Male'
- Apply conditional filtering to get back all the buyers that their left above 10 percent tip from their total bill amount

Class Exercise Solution - Dataframe Conditional Filtering