Phase two

Data analyst

intro

Anaconda
is a distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment. It comes with a large collection of pre-installed packages used in data science, machine learning, and scientific computing.

Jupyter is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. It supports various programming languages, including Python, R, Julia, and others.

Jupyter Notebooks are the primary interface for working with Jupyter. These notebooks allow you to write and execute code in individual cells, view the results, and add formatted text, equations, and visualizations. Jupyter Notebooks have become very popular in data science and scientific computing due to their ability to create interactive and reproducible computational narratives.

1-NumPy

NumPy > is a Python library that provides support for mathematical and scientific operations with large multidimensional arrays and matrices.

Usage of NumPy → First import the numpy library

- 1.Creating NumPy Arrays using np.array() function.
- 2. Some of Attributes

```
print(arr2d.shape) # Shape of the array
print(arr2d.size) # Number of elements in the array
print(arr2d.dtype) # Datatype of the array
```

3. Array Operations -> + , * , - , multiplication using np.dot(arr1,arr2)

5. Array indexing →

```
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr[0, 0])  # Access element at row 0, column 0
print(arr[1])  # Access entire second row
print(arr[:, 0])  # Access entire first column
```

2-Pands

Pandas - is a popular Python library used for data manipulation and analysis.

The primary data structures in Pandas are:

1-Series one-dimensional labeled array capable of holding any data type.

```
series = df['name_of_columns']
```

2-DataFrame two-dimensional labeled data structure with columns of potentially different types It is similar to SQL table.

df = pd.CreateDataframe('name_of_dictionary)

```
How to call columns:

df.email

df['column_name']
```

```
How to call row? By using loc or iloc
In .loc[], you specify the row and column labels
df.iloc[1:4, 0:2]
while in .iloc[], you specify the row and column indices.
df.loc['row2':'row4', 'A':'B']
```

Df[name_columns].Value_counts()

Count the occurrence of each unique value

set_index() is a method in Pandas used to set the DataFrame index using existing columns.

```
# Set column 'B' as the index
             df.set_index('B', inplace=True) → using inplace to done changes
             df.index
             # Reset the index
             df_reset = df.reset_index()
             Df = pd.read_csv('
name of file',index_col='name_of_column_u_need')
             df.rename(columns={' ':' '},inplace =True) → to change column
name
```

Filter mask -> create a df using Boolean mask.

Create a boolean mask based on a condit mask = df['A'] > 2 # Apply the mask to filter the DataFrame filtered_df = df[mask]

Dealing with string

```
In [13]: filt = df['LanguageWorkedWith'].str.contains('Python', na=False)
In [14]: df.loc[filt, 'LanguageWorkedWith']
         Respondent
Out[14]:
                                      HTML/CSS; Java; JavaScript; Python
                                                  C++; HTML/CSS; Python
                                                  C;C++;C#;Python;SQL
                         C++; HTML/CSS; Java; JavaScript; Python; SQL; VBA
                   Bash/Shell/PowerShell;C;C++;HTML/CSS;Java;Java...
                   Bash/Shell/PowerShell;C;C++;HTML/CSS;Java;Java...
          84539
                     Bash/Shell/PowerShell; C++; Python; Ruby; Other(s):
          85738
                     Bash/Shell/PowerShell; HTML/CSS; Python; Other(s):
          86566
          87739
                            C;C++;HTML/CSS;JavaScript;PHP;Python;SQL
```

```
In [5]: high salary = (df[ ConvertedComp ] > 70000)
        df.loc|high_salary, ['Country', 'LanguageWorkedWith', 'ConvertedComp']]
                                                               LanguageWorkedWith ConvertedComp
                             Canada
                                                                        Java R:SOL
                                                                                          366420.0
                                     Bash/Shell/PowerShell;C#:HTML/CSS:JavaScript;P.
                                                                                            96179.0
                                                                                            90000.0
                                                                                          455352.0
                                      Bash/Shell/PowerShelt;C#:HTML/CSS:,JavaScript;T.
                         United States Bash/Shell/PowerShelt.C++;HTML/CSS:JavaScript:
                                                                                           103000.0
                         United States Bash/Shel/PowerShell.C#;HTML/CSS;Java;Python;,
                                                                                           180000.0
                                                                                          2000000.0
                                                 HTML/CSS;JavaScript;Scala;TypeScript
                                                                                          130000.0
                              Finland
                                                     Bash/Shell/PowerShell;C++;Python
                                                                                            82488.0
```

Updating row in df apply vs applymap

apply():

- •Use Case: Apply a function along an axis of the DataFrame.
- •Function Application: Applies a function along either axis of a DataFrame.
- •Axis: Can be applied along rows (axis=0) or columns (axis=1).
- •Input: Takes a function as an argument.
- •Output: Returns a DataFrame or Series depending on the function used.
- •Use: Typically used to apply complex functions that operate on entire rows or columns.

Apply the function along columns result = df.apply(square_sum, axis=0)

applymap():

- •Use Case: Apply a function element-wise to the entire DataFrame.
- •Function Application: Applies a function to every element of the DataFrame.
- •Input: Takes a function as an argument.
- •Output: Returns a DataFrame.
- •Use: Typically used for element-wise operations, like transforming each element with a simple function.

Apply the function element-wise result = df.applymap(square)

Map func → is used to substitute each value in a Series with another value.

Series.map(arg, na_action=None)

- arg: A function, a dictionary, or a Series.
- na_action: {None, 'ignore'}, default None. If 'ignore', propagate NaN values, without passing them to the mapping function.

```
ex 🗲
```

```
import pandas as pd

# Example Series
s = pd.Series(['cat', 'dog', 'rabbit'])

# Define a dictionary to map values
mapping = {'cat': 'feline', 'dog': 'canine', 'rabbit': 'rodent'}

# Map the values using the dictionary
result = s.map(mapping)
print(result)
```

Add and rem column:

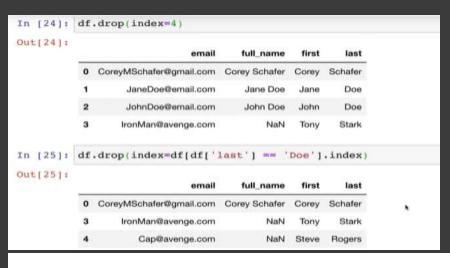
```
df['new_name] = df[' '] + df[' ']
df.drop(columns=['u_want_to_del','another if u want'],inplace=True) to del
```

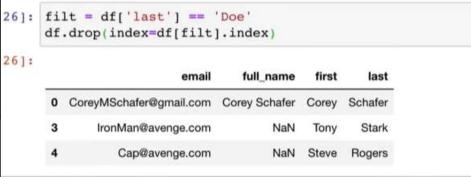
```
# Removing column 'C'

df.drop('C', axis=1, inplace=True)
```

Add and rem row:

```
# Adding multiple rows
new_rows = pd.DataFrame({ 'A': [4, 5], 'B': [7, 8] })
df = df.append(new_rows, ignore_index=True)
# Removing rows with index 1 and 3
df = df.drop([1, 3])
df = df.drop(index=4)
```





Split → df['column']. str.split(pat=None, n=-1, expand=False)

- pat: str, optional. The delimiter to split the string on. If not specified, splits on whitespace.
- n: int, default -1 (all). Maximum number of splits. If None, splits all occurrences.
- expand: bool, default False. If True, return DataFrame/MultiIndex expanding dimensionality.

Sorting →

```
# Sort by values in column 'A'

df_sorted_values = df.sort_values(by='A', ascending=False, inplace=True)

#to done changes
```

df.sort_index()

End