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]
```

```
In [5]: high_salary = (df['ConvertedComp'] > 70000)

In [7]: df.loc[high_salary, ['Country', 'LanguageWorkedWith', 'ConvertedComp']]
Out[7]:
```

Respondent	Country	LanguageWorkedWith	ConvertedComp
6	Canada	Java;R;SQL	366420.0
9	New Zealand	Bash/Shell/PowerShell;C#;HTML/CSS;JavaScript;P...	95179.0
13	United States	Bash/Shell/PowerShell;HTML/CSS;JavaScript;PHP;...	90000.0
16	United Kingdom	Bash/Shell/PowerShell;C#;HTML/CSS;JavaScript;T...	455352.0
22	United States	Bash/Shell/PowerShell;C++;HTML/CSS;JavaScript;...	103000.0
...	...	...	...
88876	United States	Bash/Shell/PowerShell;C#;HTML/CSS;Java;Python;...	180000.0
88877	United States	Bash/Shell/PowerShell;C;Clojure;HTML/CSS;Java;...	2000000.0
88878	United States	HTML/CSS;JavaScript;Scala;TypeScript	130000.0
88879	Finland	Bash/Shell/PowerShell;C++;Python	82488.0

```
In [8]: countries = ['United States', 'India', 'United Kingdom', 'Germany', 'Canada']
        filt = df['Country'].isin(countries)

In [10]: df.loc[filt, 'Country']
Out[10]: Respondent
         1        United Kingdom
         4         United States
         6                Canada
         8                 India
         10                India
```

Dealing with string

```
In [13]: filt = df['LanguageWorkedWith'].str.contains('Python', na=False)

In [14]: df.loc[filt, 'LanguageWorkedWith']
Out[14]: Respondent
         1                          HTML/CSS;Java;JavaScript;Python
         2                                      C++;HTML/CSS;Python
         4                                      C;C++;C#;Python;SQL
         5                C++;HTML/CSS;Java;JavaScript;Python;SQL;VBA
         8             Bash/Shell/PowerShell;C;C++;HTML/CSS;Java;Java...
                                        ...
         84539         Bash/Shell/PowerShell;C;C++;HTML/CSS;Java;Java...
         85738           Bash/Shell/PowerShell;C++;Python;Ruby;Other(s):
         86566           Bash/Shell/PowerShell;HTML/CSS;Python;Other(s):
         87739                C;C++;HTML/CSS;JavaScript;PHP;Python;SQL
```

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)
```

In [24]: df.drop(index=4)

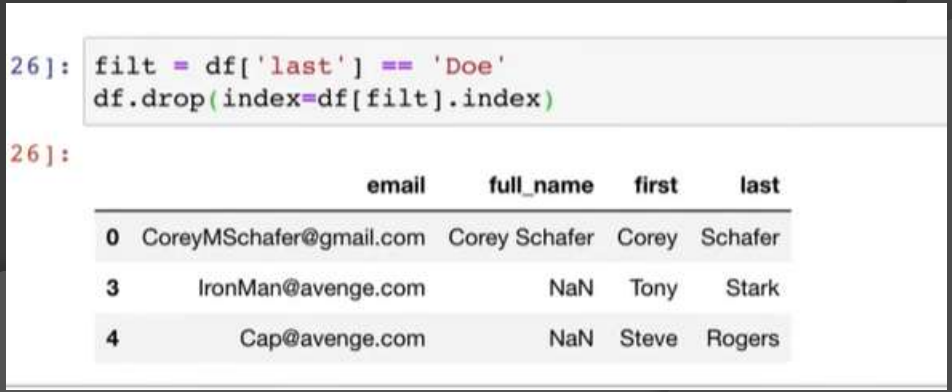
Out[24]:

	email	full_name	first	last
0	CoreyMSchafer@gmail.com	Corey Schafer	Corey	Schafer
1	JaneDoe@email.com	Jane Doe	Jane	Doe
2	JohnDoe@email.com	John Doe	John	Doe
3	IronMan@avenge.com	NaN	Tony	Stark

In [25]: df.drop(index=df[df['last'] == 'Doe'].index)

Out[25]:

	email	full_name	first	last
0	CoreyMSchafer@gmail.com	Corey Schafer	Corey	Schafer
3	IronMan@avenge.com	NaN	Tony	Stark
4	Cap@avenge.com	NaN	Steve	Rogers



26]: filt = df['last'] == 'Doe'
df.drop(index=df[filt].index)

26]:

	email	full_name	first	last
0	CoreyMSchafer@gmail.com	Corey Schafer	Corey	Schafer
3	IronMan@avenge.com	NaN	Tony	Stark
4	Cap@avenge.com	NaN	Steve	Rogers

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()
```

Casting Datatypes

1. Convert a column to numeric:

```
df['numeric_column'] = pd.to_numeric(df['numeric_column'], errors='coerce’)
```

2. Converting a column to categorical:

```
df['category_column'] = df['category_column'].astype('category’)
```

3. Converting a column to integer:

```
df['column_name'] = df['column_name'].astype(int) #float .astype(float) str , categorical
```

Cleaning Data

1. Dropping missing values:

```
df.dropna(axis=1, supset=['column'],inplace=True)   # axis = 0 meaning row / 1 == columns
```

```
df.dropna(axis=1, how='all',inplace=True) # how ='all' mean all the axis is null / 'any' mean any value
```

2. F

```
df['column_name'].fillna(df['column_name'].mean(), inplace=True) #.mode()[0]
```

str.strip() method is used to remove leading and trailing whitespace (spaces, tabs, newlines) from the values in a column

```
In [56]: df[["Street_Address", "State", "Zip_Code"]] = df["Address"].str.split(',',2, expand=True)
         df
```

Out[56]:

	CustomerID	First_Name	Last_Name	Phone_Number	Address	Paying Customer	Do_Not_Contact	Street_Address	State	Zip_Code
0	1001	Frodo	Baggins	123-545-5421	123 Shire Lane, Shire	Yes	No	123 Shire Lane	Shire	None
1	1002	Abed	Nadir	123-643-9775	93 West Main Street	No	Yes	93 West Main Street	None	None
2	1003	Walter	White		298 Drugs Driveway	N	NaN	298 Drugs Driveway	None	None
3	1004	Dwight	Schrute	123-543-2345	980 Paper Avenue, Pennsylvania, 18503	Yes	Y	980 Paper Avenue	Pennsylvania	18503
4	1005	Jon	Snow	876-678-3469	123 Dragons Road	Y	No	123 Dragons Road	None	None
5	1006	Ron	Swanson	304-762-2467	768 City Parkway	Yes	Yes	768 City Parkway	None	None
6	1007	Jeff	Winger		1209 South Street	No	No	1209 South Street	None	None
7	1008	Sherlock	Holmes	876-678-3469	98 Clue Drive	N	No	98 Clue Drive	None	None

Grouping data

grouping data ➔ using the `groupby()` method to grouping your data .

This method allows you to split your data into groups based on some criteria and perform using some operation to this groups.

Grouping by one columns ➔ df.groupby(‘name_of_column’)

Grouping by multilabel columns ➔ df.groupby([‘name_of_first_column’,’name_of_saceond’])

```
gender_occupation = users.groupby('occupation')['gender'].value_counts().unstack()
```

Aggregating ➔ this is computing summary statistics or transforming the data in some way based on groups defined by one or more categorical variables. This process is typically done using the groupby method to make a multi func in group

an aggregation function such as sum , mean , count ,etc

agg() to use multiable func as a time

```
min_max_age = users.groupby('occupation').agg({
    'age': ['min', 'max']})
min_max_age
```

Dates and Time Series Data

this slide explain to how Working with dates and time series data in pandas involves several key operations such as parsing dates, setting date indices, and performing date-based aggregations.

Parsing Dates

When importing data with date columns, pandas can automatically parse them into datetime objects using (parse_dates) parameter in read_csv method Alternatively, you can use `pd.to_datetime()` to convert strings to datetime objects.

- **Parsing Dates**: Use `pd.to_datetime()` or `parse_dates` parameter in `read_csv()` to convert strings to datetime objects.
- **Setting Date Indices**: Use `set_index()` to set date columns as indices for time-based operations.
- **Resampling and Aggregating**: Use `resample()` with aggregation functions to aggregate time series data over different frequencies.
- **Date Range Generation**: Generate date ranges with `pd.date_range()` for creating time series data.
- **Time Zone Handling**: Use `tz_localize()` and `tz_convert()` to handle time zones.

3-matplotlib

Matplotlib ➔ is a popular plotting library for Python, useful for creating static, animated, and interactive visualizations.

1- to import ➔ `import matplotlib.pyplot as plt`

2- ’ %matplotlib inline’ ➔ In Jupyter Notebooks, "magic commands" are special commands prefixed by % (line magics) or %% (cell magics) that provide enhanced functionality. The ’ %matplotlib inline’ used command is a line magic that configures the notebook to display Matplotlib plots inline, meaning the plots will appear within the notebook itself rather than in a separate window.

3- subplots() ➔ function in Matplotlib is a powerful tool for creating and managing multiple plots within a single figure. It provides a high level of control over the layout and appearance of the plots.

```
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(width, height), sharex=False, sharey=False)
    (ax1,ax2)
```

4-scatter plot ➔ is a type of plot used to visualize the relationship between two sets of data points. It's particularly useful for identifying patterns, relationships, and correlations between variables.

```
plt.scatter(older['age'],older['chol’], s=3 , cmap='viridis',c=older['target'],label='Patients')
```

```
older.plot(kind='scatter',x='age’,s=3 ,c ='target' , y='chol',cmap='inferno', label='Patients',figsize=(10,6))
```

- **x, y**: Arrays or lists representing the data for x and y coordinates.
- **color**: Color of markers (default is 'b' for blue).
- **marker**: Marker style (e.g., 'o' for circles, 's' for squares).
- **s**: Marker size.
- **alpha**: Transparency of markers (0 for transparent, 1 for opaque).
- **label**: Label for the legend.

Bar chart ➔ is used to represent categorical data with rectangular bars where the length or height of each bar corresponds to the value it represents. Bar charts are commonly used to compare quantities of different categories or to track changes over time for discrete categories.

```
plt.bar(X, y)
```

Histogram ➔ is a graphical representation of data that shows the distribution of numerical data. It consists of bars where the height of each bar represents the frequency or count of data points within a specific range or "bin" of values.

Histograms are commonly used in statistics to visualize the shape of a distribution, whether it's symmetric, skewed, or uniform. They are particularly useful for understanding the spread and central tendency of data sets.

```
plt.hist(X, bins=30, edgecolor='black’)
heart_disease.plot.hist(subplots=True, bins=20,figsize=(10, 8),edgecolor='black') # hole data
```

```python
sb.get_dataset_names()
tips = sb.load_dataset('tips')
tips

sb.scatterplot(x='x-axis',y='y-axis',data=tips,hue='coler and added auto a legend',size='size',palette='YlGnBu') # sb.boxplot() &
sb.histplot(tips['tip'],kde=True,bins=15)
sb.displot(tips['tip'],kde=True,bins=15)
sb.barplot(x='',y='',palette='YlGnBu')
sb.stripplot(x='x-axis',y='y-axis',data='name of data',hue='coler and added auto a legend',size='size',palette='YlGnBu',dodge=True)
sb.jointplot(x='x-axis',y='y-axis',data='name of data',size='size',cmap='YlGnBu',kind='reg')#kind='kde' , shade=true
sb.pairplot(tips.select_dtypes(['number']),hue='pclass')
sb.scatterplot(x='tip',y='total_bill',data='tips',hue='tip',palette='YlGnBu') # sb.boxplot() &
```

4-seaborn

Seaborn ➔ is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn helps make complex plots with fewer lines of code.

Key features of Seaborn include:

1. **Built-in Themes and Color Palettes**: Simplify the customization of plots.
2. **Statistical Estimation**: Automatically perform and visualize statistical analyses.
3. **High-Level Plotting Functions**: Quickly create complex plots, like regression plots, violin plots, and heatmaps.
4. **Integration with Pandas**: Work seamlessly with data stored in pandas DataFrames.

1. Scatter Plot → Scatter plots are used to observe the relationship between two numeric variables.

Calling way:

```
sb.scatterplot(x='x-axis',y='y-axis',data=tips,hue='coler and added auto a legend',size='size',palette='YlGnBu')
# sb.boxplot() &
```

2. Line Plot→ Line plots are used to track changes over periods of time.

Calling way:

```
sb.lineplot(data=tips, x="size", y="total_bill")
```

3. Bar Plot → Bar plots represent data with rectangular bars with lengths proportional to the values they represent.

Calling way:

```
sb.barplot(data=tips, x="day", y="total_bill", hue="sex")
```

4. Histogram➔ Histograms show the distribution of a single numeric variable.

Calling way:

```
sb.histplot(tips['tip'],kde=True,bins=15)
```

5. Box Plot➔ Box plots show the distribution of quantitative data and can be used to identify outliers.

Calling way:

```
sb.boxplot(data=tips, x="day", y="total_bill", hue="smoker")
```

6. Heatmap➔ Heatmaps are used to visualize data in matrix form.

Calling way:

```
# Create a pivot table or flights_pivot.corr() to the numirc data
flights_pivot = flights.pivot("month", "year", "passengers")

# Create a heatmap
sns.heatmap(data=flights_pivot, annot=True, fmt="d", cmap="YlGnBu")
```

End