**Python Case Study**

Q1**.**

First, we will read the data from the file and load it as JSON

Then we will use pandas.DataFrame and pass the data and the key “people” to turn the JSON data to a Pandas Data Frame

We can also rename the columns we have from the default text file

**import** **pandas** **as** **pd**

**import** **json**

filepath = 'EmoplyeeDetails.txt'

**with** open(filepath, 'r') **as** file:

json\_data = file.read()

data = json.loads(json\_data)

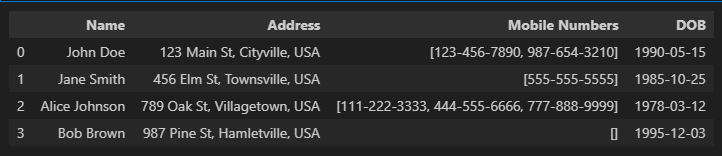
df = pd.DataFrame(data['people'])

df.rename(columns={'name':'Name','address':'Address',

"mobile\_numbers":"Mobile Numbers",

"date\_of\_birth":"DOB"}, inplace=True)

df

****

Q2**.**

First Let us create Test Files for the code we are going to write.

**import** **pandas** **as** **pd**

**import** **os**

**import** **datetime** **as** **dt**

filepath = 'data'

date\_list = []

current\_date = dt.datetime.now()

**for** i **in** range(**7**):

date\_string = current\_date.strftime('%Y-%m-%d\_%H-%M-%S')

date\_list.append(date\_string)

current\_date += dt.timedelta(days=**1**)

**print**(date\_list)

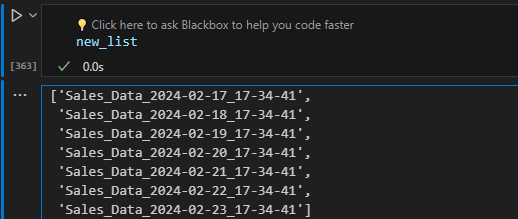
This will create a list of timestamps with the help of the current date and adding 1 day to it. It will create 7 strings with 7 days timestamp for the dummy data.

Let’s make a new list that will have the full name of the files we want to create.

name\_str = 'Sales\_Data'

new\_list = [name\_str + '\_' + date **for** date **in** date\_list]

This will give us the names that we require for the dummy files.



Once we have the names for the files let us make some dummy data with 7 data points in json format and enter this data into our 7 files

json\_data =[

{

"product": "Widget A",

"quantity": **100**,

"price\_per\_unit": **10.50**,

"total\_sales": **1050.00**

},

{

"product": "Widget B",

"quantity": **50**,

"price\_per\_unit": **25.75**,

"total\_sales": **1287.50**

},

{

"product": "Widget C",

"quantity": **75**,

"price\_per\_unit": **15.00**,

"total\_sales": **1125.00**

},

{

"product": "Widget D",

"quantity": **30**,

"price\_per\_unit": **50.00**,

"total\_sales": **1500.00**

},

{

"product": "Widget E",

"quantity": **20**,

"price\_per\_unit": **40.25**,

"total\_sales": **805.00**

},

{

"product": "Widget F",

"quantity": **60**,

"price\_per\_unit": **18.99**,

"total\_sales": **1139.40**

},

{

"product": "Widget G",

"quantity": **45**,

"price\_per\_unit": **30.00**,

"total\_sales": **1350.00**

}

]

**import** **csv**

**for** i **in** range(len(new\_list)):

filename = os.path.splitext(new\_list[i])[**0**] + ".csv"

**with** open(os.path.join(filepath, filename), 'w', newline='') **as** f:

writer = csv.DictWriter(f, fieldnames=["product", "quantity", "price\_per\_unit","total\_sales"])

writer.writeheader()

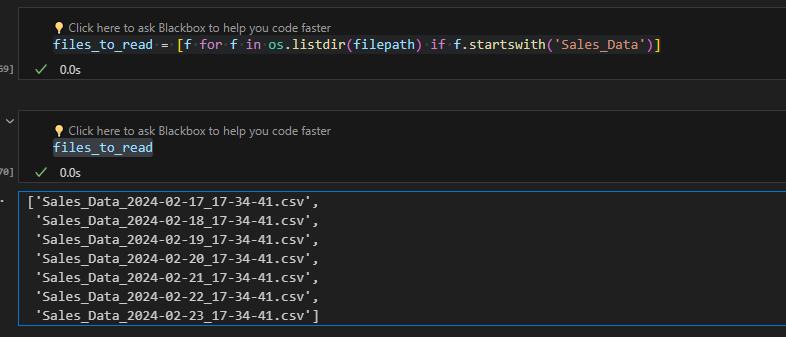
writer.writerow(json\_data[i])

Once we get the data loaded and files created, we can now work on our function to load all the data into one table and keep in check if the file data entered is distinct also keeping in mind if data is modified for a previous date to delete and update as required.

First let us try to get all the files that starts with ‘Sales\_Data’

files\_to\_read = [f **for** f **in** os.listdir(filepath) **if** f.startswith ('Sales\_Data')]

We can get the names of all the files in the file path using os.listdir .



Once we get the names of the files, we can now get to the next step which is to open all of these and then add the data into a data frame and keep on concatenating the data from different files to the table.

**def** **into\_table**(filepath: str, distinct: set, initial\_df: pd.DataFrame) -> pd.DataFrame:

**try**:

files\_to\_read = [f **for** f **in** os.listdir(filepath) **if** f.startswith ('Sales\_Data')]

**for** file **in** files\_to\_read:

date = file.split('\_')[**2**]

**if** date **not** **in** distinct:

df = pd.read\_csv(os.path.join(filepath, file))

df['Date'] = date

initial\_df = pd.concat([initial\_df, df], ignore\_index=True)

distinct.add(date)

**else**:

initial\_df = initial\_df [~initial\_df['Date'] .str.startswith (date)]

df = pd.read\_csv(os.path.join(filepath, file))

df['Date'] = date

initial\_df = pd.concat([initial\_df, df], ignore\_index=True)

**return** initial\_df

**except** **Exception** **as** e:

**print**("An error occurred:", e)

**return** initial\_df

In this function we take an empty data frame as set named distinct and the file path. We then get the names in the list for the files to be read and use a for loop to open and extract the data from the files while checking:

1. The date is not in distinct. We open the file using pd.read\_csv and add the date which we extracted from the title of the file and then we concatenate the df to the initial df and add the date into the set
2. If the date is in distinct, we delete the previous data while searching it with the date read the file once again and concatenate the data

This Function However doesn’t check if the file is modified and each time just delete’s the previous data and concatenate the same data.

To fix this we can use the time we got in our time stamp and check the already added time column with the time we get from the new name of the file.

**def** **into\_table\_modified**(filepath: str, distinct: set, initial\_df: pd.DataFrame) -> pd.DataFrame:

**try**:

files\_to\_read = [f **for** f **in** os.listdir(filepath) **if** f.startswith('Sales\_Data')]

**for** file **in** files\_to\_read:

date = file.split('\_')[**2**]

time = file.split('\_')[**3**].split('.')[**0**]

formatted\_time = dt.datetime.strptime(time, '%H-%M-%S').strftime('%H:%M:%S')

**if** date **not** **in** distinct:

df = pd.read\_csv(os.path.join(filepath, file))

df['Date'] = date

df['Time Stamp'] = formatted\_time

initial\_df = pd.concat([initial\_df, df], ignore\_index=True)

distinct.add(date)

**else**:

max\_time\_for\_date = initial\_df.loc[initial\_df['Date'] == date, 'Time Stamp'].max()

**if** formatted\_time > max\_time\_for\_date:

initial\_df = initial\_df[~initial\_df['Date'].str.startswith(date)]

df = pd.read\_csv(os.path.join(filepath, file))

df['Date'] = date

df['Time Stamp'] = formatted\_time

initial\_df = pd.concat([initial\_df, df], ignore\_index=True)

**return** initial\_df

**except** **Exception** **as** e:

**print**("An error occurred:", e)

**return** initial\_df

Here we have made some changes into the function where we are also checking the previously appended date column with the new date extracted from the file name.

This Fixes all our problems and all the criteria for the problem statement is met.

We can now test the function..

distinct = set()

initial\_df = pd.DataFrame()

df\_new = into\_table\_modified(filepath,distinct,initial\_df)

df\_new

This is the resultant table we get



Now let us add a new file with same date but different time.

current\_date1 = dt.datetime.now()

date\_string = current\_date1.strftime('%Y-%m-%d\_%H-%M-%S')

name = 'Sales\_Data'+ '\_' + date\_string

json =[ {

"product": "Widget H",

"quantity": **42**,

"price\_per\_unit": **31.00**,

"total\_sales": **1250.00**

}

]

filename = os.path.splitext(name)[**0**] + ".csv"

**with** open(os.path.join(filepath, filename), 'w', newline='') **as** f:

writer = csv.DictWriter(f, fieldnames=["product", "quantity", "price\_per\_unit", "total\_sales"])

writer.writeheader()

writer.writerow(json[**0**])

Let us run the function again

into\_table\_modified(filepath,distinct,df\_new)

The resultant data frame is now

****

As you can see the time stamp of the newly and updated data for 17-02-2024 is changed and the function works fine

Q3**.**

**def** **evenPos**(string:str) -> str:

result = ""

**for** i **in** range(**0**, len(string), **2**):

result += string[i]

**return** result

**if** \_\_name\_\_ == "\_\_main\_\_":

str = input("Enter a string: ")

string = evenPos(str)

**print**("The string without even posistions are : ",string)

****

**SQL Case Study**

Before answering the questions lets make the tables and insert the data we want to add:

CREATE DATABASE IF NOT EXISTS Org;

USE Org;

CREATE TABLE IF NOT EXISTS Department (

    Department\_id INT UNIQUE NOT NULL PRIMARY KEY,

    Department\_name VARCHAR(255)

);

CREATE TABLE IF NOT EXISTS Employee (

    Emp\_id INT UNIQUE NOT NULL PRIMARY KEY,

    Emp\_Name VARCHAR(255),

    Department\_id INT,

    Manager\_id INT,

    Salary INT,

    email\_Address VARCHAR(255),

    INDEX (Department\_id),

    FOREIGN KEY (Department\_id) REFERENCES Department(Department\_id)

);

-- Inserting data into Department table

INSERT INTO Department (Department\_id, Department\_name) VALUES

(1, 'IT'),

(3, 'HR'),

(5, 'Payroll');

INSERT INTO Employee (Emp\_Name, Emp\_id, Department\_id, Manager\_id, Salary, email\_Address)

VALUES ('Naved', 1, 1, 6, 1000, 'Naved@gmail.com'),

       ('Atul', 2, 1, 6, 1290, 'Atul@Hotmail.com'),

       ('Raja', 3, 3, 6, 1500, 'Raja@hotmail.com'),

       ('Akash', 4, 3, 5, 2000, 'Akash@Outlook.com'),

       ('Rajan', 6, 5, 6, 30000, 'Rajan@Outlook.com'),

       ('Balam', 5, 5, 6, 4000, 'Balam@outlook.com'),

       ('Ravi', 7, 1, 5, 2000, 'Ravi@gmail.com'),

       ('Ram', 9, 1, 4, 7000, 'Ram@Hotmail.com'),

       ('Paul', 8, 3, 5, 6000, 'Paul@gmail.com');

Now Beginning with the questions

Q1.

-- Count of employees in each department

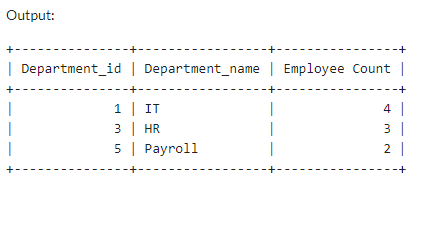
SELECT e.Department\_id,d.Department\_name,COUNT(e.Emp\_id) AS 'Employee Count'

FROM Employee as e

LEFT JOIN Department as d

ON e.Department\_id = d.Department\_id

GROUP BY e.Department\_id,d.department\_name;

****

Q2**.**

-- Highest Salary by Department with Employee Name

SELECT d.Department\_name, e.Emp\_Name

FROM Department AS d

INNER JOIN(

    SELECT Department\_id, MAX(Salary) AS 'Max\_Salary'

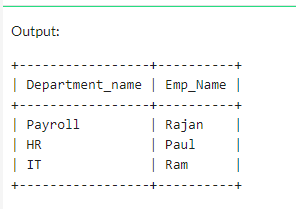
    FROM Employee

    GROUP BY Department\_id

) AS max\_sal

ON d.Department\_id = max\_sal.Department\_id

INNER JOIN Employee AS e ON e.Department\_id = d.Department\_id AND e.Salary = max\_sal.Max\_Salary;

****

Q3**.**

-- Total Salary by Department

SELECT d.department\_name, SUM(e.salary) AS 'Total Salary'

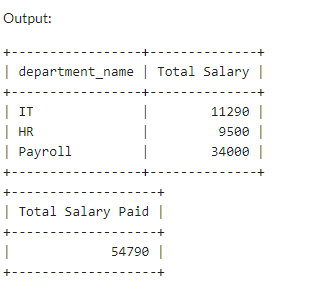
FROM Employee as e

JOIN Department as d on e.department\_id=d.department\_id

GROUP BY d.department\_id;

-- Total Salary Paid

SELECT SUM(salary) AS 'Total Salary Paid' FROM Employee;

****

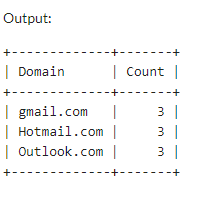
Q4**.**

-- Distinct email domain and their counts

SELECT SUBSTRING\_INDEX(email\_Address, '@', -1) AS 'Domain', COUNT(\*) AS 'Count'

FROM Employee

GROUP BY Domain;

****

Q5**.**

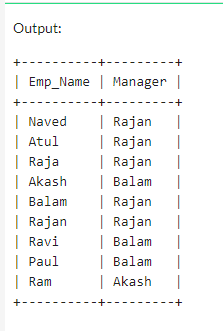
-- Name of Employee and its Manager

SELECT e.Emp\_Name,m.Emp\_Name AS 'Manager'

FROM Employee AS e

LEFT JOIN Employee AS m

ON e.Manager\_id = m.Emp\_id;

****

Q6**.**

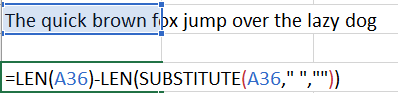
-- Replica of table without duplicating

CREATE TABLE Employee\_Replica LIKE Employee;

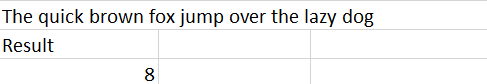
**EXCEL CASE STUDY**

Q1.

Count Whitespaces = Len of the Sentence – Len of sentence without whitespaces



**=LEN(A36)-LEN (SUBSTITUTE (A36,” “,””))**



Q2.

VLOOKUP is an Excel Named Function that is used to search for things in a column and return the answer of the row that matches

The Syntax is as follows

**=VLOOKUP (Cell, Array/Named Range, Column Number, Exact Match)**

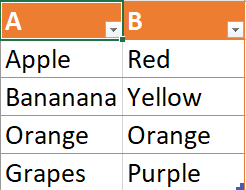
Cell -> The cell to look for

Array -> The array to look from

Col Number -> The column number to look in

Exact Match -> FALSE for exact match

Example: -

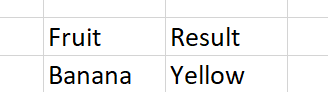


Let us suppose a table with Fruit names and their colour.

To find the Colour of the Fruit we can use VLOOKUP

**=VLOOKUP (D5, Table3, 2, FALSE)**

Will give:



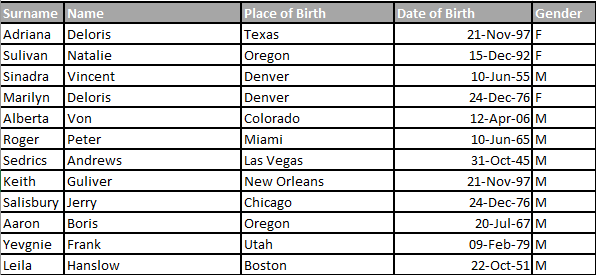
Q3.

We can check all the named ranges by

1. **Formula Tab 🡪 Defined Names 🡪Name Manager**
2. **Pressing CTRL+F3 on keyboard**

Q4.

A)



First Select any nearby cell and press CTRL+T to create it into a table

Create a new column named Full Name

Use the formula

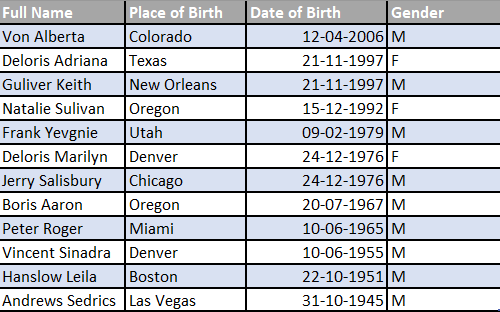
**=CONCAT (A2, " ", B2)**

Excel will automatically apply it to all the other rows.

After this Select the column and copy it go to:

**Home 🡪 Paste 🡪 Paste Values**

Now we can remove the first name and last name column from the table.



Now for reference lets just enter the filtering options in two separate cells.



We can then use the Formula.

**=FILTER (Table2[Full Name], (Table2[Gender]=A17) \* (Table2[Place of Birth] = B17))**

Here the syntax is: -

**=FILTER (Array, Conditions)**

**We use \* To add multiple conditions.**

**Table2[Gender] = A17 means Gender should be equal to value in A17 which is F and Table2[Place of Birth] = B17 means Place of Birth should be equal to Denver.**

**This will give us the result as: -**

****

We can confirm this with our table



B)

Let us approach this step by step

Firstly, let’s get the Date of Birth of All the Males Using Filter

**=FILTER (Table2[Date of Birth], Table2[Gender]="M")**

This will give us the Values associated to this filter

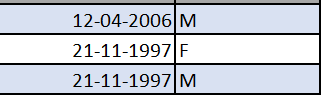
Then we can Use LARGE to get the 2 largest years i.e., the second smallest male

**=LARGE (FILTER (Table2[Date of Birth], Table2[Gender]="M"), 2)**

This will give us the date of birth we need to look for

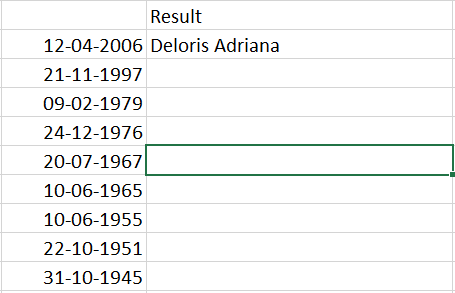


We can also confirm this by our table by sorting it

****

We will Then search for the Name using XLOOKUP

**=XLOOKUP (LARGE (FILTER (Table2[Date of Birth], Table2[Gender]="M"),2), Table2[Date of Birth], Table2[Full Name])**

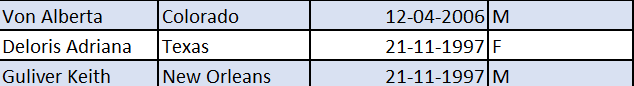
**This will give us our result**

**But since we have 2 people having the same Date of Birth this shows the female candidate**

Now to solve this problem we can use XMATCH to find the row number from down to up in the table it will then give us the correct index to the answer

**=XMATCH (LARGE (FILTER (Table2[Date of Birth], Table2[Gender]="M"), 2), Table2[Date of Birth], 0, -1)**

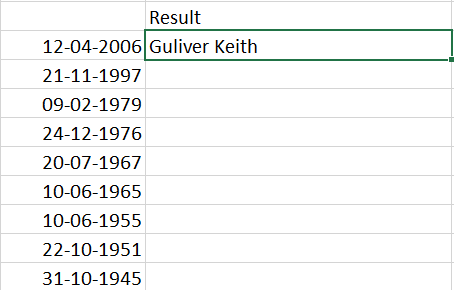
**This gives us the answer as 3 which is the position of our table in descending order**

****

We can then Formulate the formula

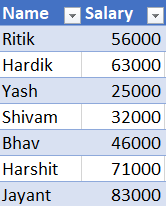
**=INDEX (XMATCH (LARGE (FILTER (Table2[Date of Birth], Table2 [Gender] = "M"),2), Table2[Date of Birth], 0, -1)**

This gives us our result which is **Guliver Keith**

****

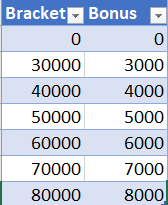
Q5.

First let us create some dummy data to work upon



The salary ranges will cover all of our test cases

We will then create references to all the numbers and respected bonuses



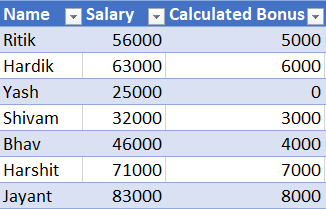
Once this is done, we can create a new column and write down the formula with nested ifs to find out the bonuses applicable.

**=IFS(AND(B11>$A$2,B11<$A$3),$B$2,AND(B11>$A$3,B11<$A$4),$B$3,AND(B11>$A$4,B11<$A$5),$B$4,AND(B11>$A$5,B11<$A$6),$B$5,AND(B11>$A$6,B11<$A$7),$B$6,AND(B11>$A$7,B11<$A$8),$B$7,B11>$A$8,$B$8)**

Here we have use absolute referencing so that it works for every column.

Once done we can drag down the formula to the other cells. We will then copy and Paste values so that the results don’t change.

We will then get the bonus everyone is going to get.



Once we get the calculated bonus, we can get the updated salary by:

**=B11+C11**

And drag to rest of the columns.

