**Fundamentals of Programming Using Python(1PGDMDSA01)**

**PGDM-trimester-1**

**Final Trimester Project**

Project on

**Black Friday Excel Data**

**using Python Language**

**Implemented as a Part of the Subject:** Fundamentals of Programming Using Python

**Subject Code:** 1PGDMDSA01

**Implemented By: Deep Prateek**

**Roll Number:** PGDMDSA23-004

**Branch and Section Details:** Post Graduate Diploma in Management (Data Science & Analytics), 2023-25, Narayana Business School

**Instructed and Guided By:** Mr. Tushar Kakaiya

**Designation:** Professor of Practice

**Department:** Information Technology and Analytics

**Institute Name:** Narayana Business School



**Problem Statements for the Analysis:**

**NOTE: You should be writing SQL Scripts to achieve the solution to the below Problem Statements**

**Final Project: Black Friday Excel Data**

**import** openpyxl **as** xl

*#from tax calculator import tax calculator*

**from** pprint **import** pprint

**Part 1**

First, read in maven\_ski\_shop\_data.xlsx

Then, the orders worksheet to an intuitive variable name.

Workbook **=** xl. load\_workbook(filename**=**'D:\TriSem-1\Python\_Project\maven\_ski\_shop\_data (1).xlsx')

​

orders **=** workbook ['Orders\_Info']

**Column Printer Function**

Since we'll be working with Excel frequently, let's create a helper function to print all the rows a specified column.

This will help us view imported data without looking at Excel, and review the data we've written into columns without needing to save a file to review the changes.

Your function should take two arguments:

* Worksheet where data is located
* Column Letter of data to print

Your function should:

* Print Cell Coordinate (e.g. A1, D4).
* Print the contents of the cell.

Then call your function twice:

* Once to print Order ID (column A in 'Orders\_Info')
* Once to print subtotal (column C in 'Orders\_Info')

*# Loop through specified sheet and column - through cells with range*

*# use f-strings to navigate to cells*

​

**def** column\_printer(sheet, column):

**for** i **in** range(1, sheet.max\_row **+** 1):

print(f'{column}{i}', sheet[f'{column}{i}'].value)

*# Loop through specified sheet and column - looping through column*

*# use f-strings to navigate to cells*

​

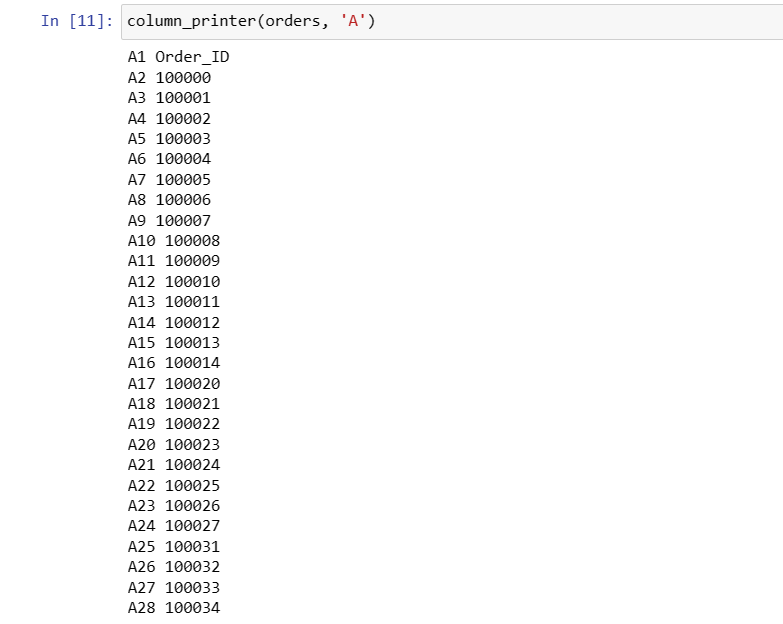
**def** column\_printer(sheet, column):

**for** cell **in** sheet[column]:

print(f'{cell.coordinate}', sheet[cell.coordinate].value)

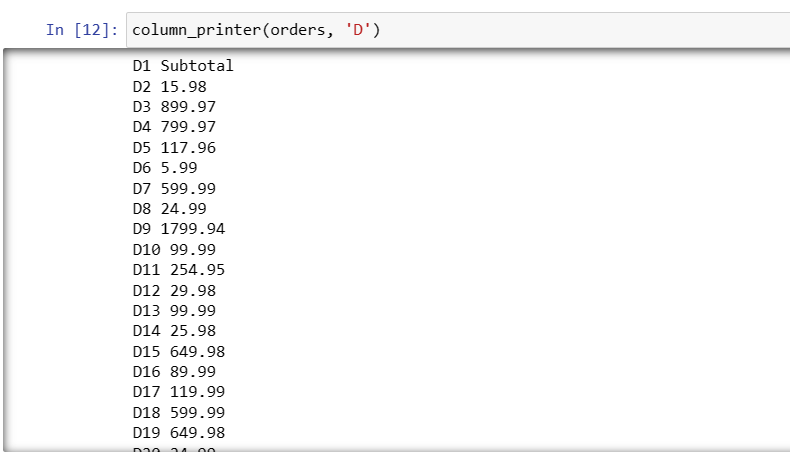
column\_printer(orders, 'A')

**Output:**



column\_printer(orders, 'D')

**Output:**



D1 Subtotal

D2 15.98

D3 899.97

D4 799.97

D5 117.96

D6 5.99

D7 599.99

D8 24.99

D9 1799.94

D10 99.99

D11 254.95

D12 29.98

D13 99.99

D14 25.98

D15 649.98

D16 89.99

D17 119.99

D18 599.99

D19 649.98

D20 24.99

D21 99.99

D22 99.99

D23 5.99

D24 24.99

D25 999.96

D26 99.99

D27 399.97

D28 89.99

**Order Data Dictionary**

Create a dictionary with all of the information contained in the 'Orders\_Info' Worksheet.

* The dictionary keys should be Order IDs (Column A)
* The values should a list storing the data in the rest of the columns.
* The columns in the list stored as values should be B, C, D, G, and H (after converting to list).
* For example: The first order, column H should be the nested list: [10001, 10002]

A few notes:

* Convert the Items\_Ordered field into a list (we covered a helpful string method for this)
* This can be done with a dictionary comprehension
* DO NOT INCLUDE EXCEL COLUMN HEADERS

*# Dictionary Comprehension with Order\_ID as Keys, rest of data as values stored in list*

*# Use split on ', ' to create a list out of column H text*

​

order\_dict**=** {}

**for** i **in** range(2,orders.max\_row **+** 1):

order\_dict[orders[f'A{i}'].value] **=** [orders[f'B{i}'].value,

orders[f'C{i}'].value,

orders[f'D{i}'].value,

orders[f'E{i}'].value,

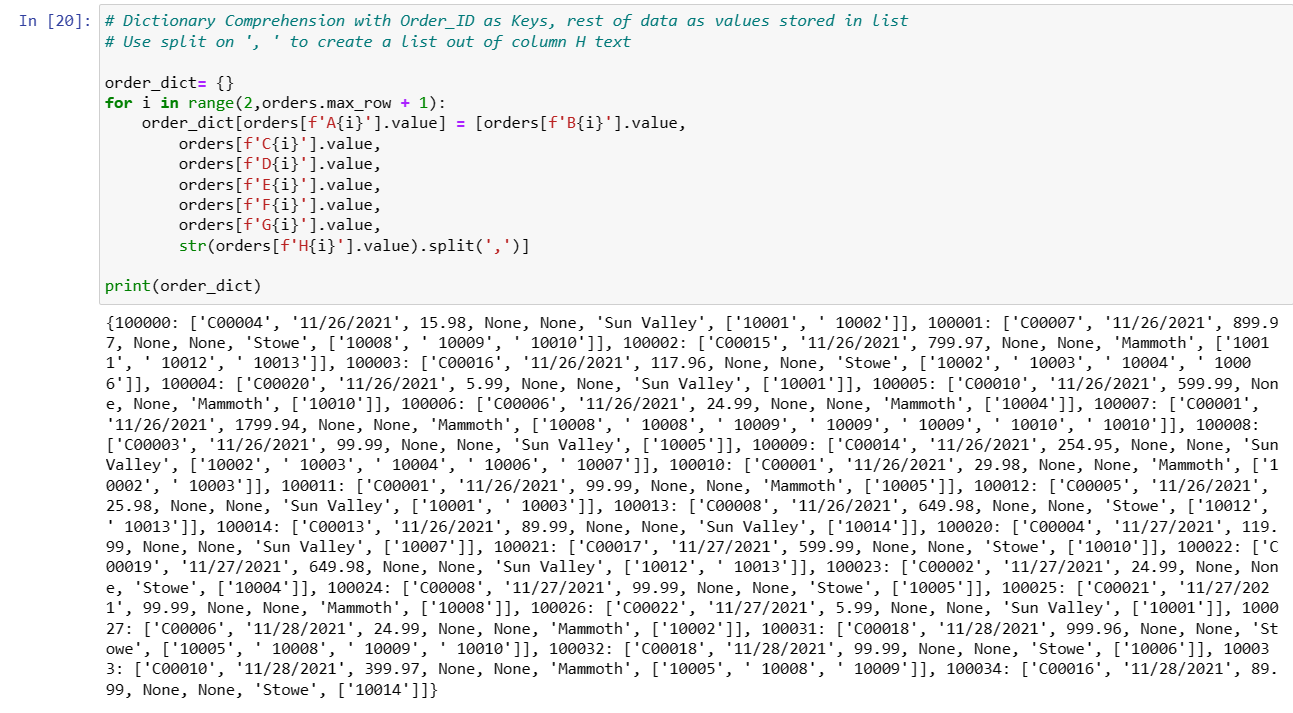
orders[f'F{i}'].value,

orders[f'G{i}'].value,

str(orders[f'H{i}'].value).split(',')]

pprint(order\_dict)

**Output:**

**Sales Tax Calculation (Last time!)**

We need to calculate the sales tax and total amount owed for every order in this sheet.

* If location is Sun Valley, apply a sales tax of 8%
* If location is Mammoth, apply a sales tax of 7.75%.
* If location is Stowe, apply a sales tax of 6%

Use the tax\_calculator function to apply sales tax to each subtotal.

**Insert** the calculated sales tax and total amounts into your customer dictionary.

**Tip**: Figure out how to do this once and then loop through the column.

%%writefile tax\_calculator.py

# Loop through dictionary values

# Check location (index 3)

# Use conditional logic to call sales tax function

# with tax rate based on location.

# Then insert tax and total into dictionary at specified location

for order in order\_dict.values():

if order[3] == 'Sun Valley':

transaction = tax\_calculator(order[2], .08)

elif order[3] == 'Mammoth':

transaction = tax\_calculator(order[2], .0775)

else:

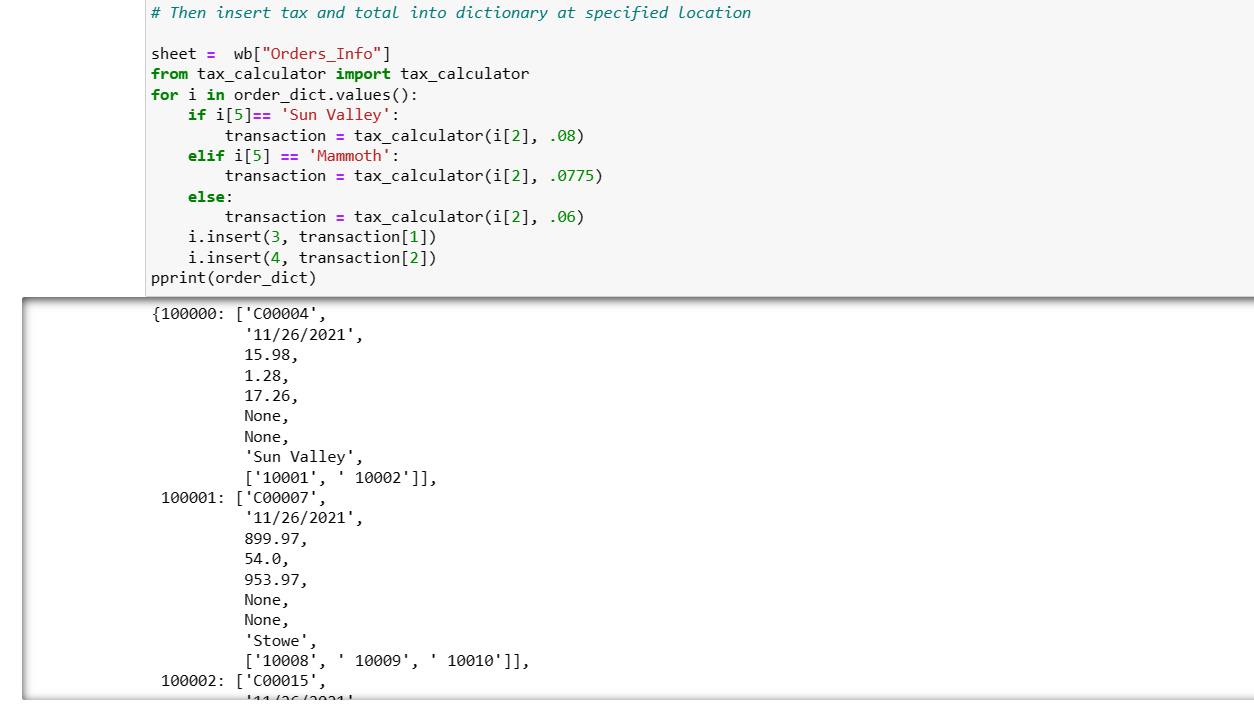
transaction = tax\_calculator(order[2], .06)

order.insert(3, transaction[1])

order.insert(4, transaction[2])

pprint(order\_dict)

**Output:**



### Write Sales Tax and Total Into the Excel Sheet

Great job! Now we just need to write this data into the workbook.

Write the sales tax and total you just calculated into the workbook, then save!

Call this workbook 'maven\_ski\_shop\_data\_fixed'.

**Tip:** There are a few ways to do this. As always, be patient, solve one step at a time.

*# iterate through order\_dict*

*# set start point to 2: + 1 to handle 0 indexing, + 1 to skip header*

**for** index, order **in** enumerate(order\_dict.values(), start**=**2):

orders[f'E{index}'] **=** order[3]

orders[f'F{index}'] **=** order[4]

*# write sales tax and total into workbook*

​

wb.save('maven\_ski\_shop\_data\_fixed.xlsx')

## Part 2

### Analysis Time!

Now that we've fixed the data - it's time to perform analysis on our sales.

Our starting point will be the order\_dict we created, after we added the taxes and totals columns.

Our first step will be to write a function that calculates the sum of a 'column' of data in our dictionary.

A 'column' for example, would be subtotals, which is at index 2 in the list stored as our dictionary values.

Your function should take the following arguments:

* Column Index
* Dictionary Name

It should output:

* The sum of values in the column (rounded to two decimal places)

Assume only numeric values will be in the column (You can develop cleaning logic later :D)

**Tip:** Use a list comprehension to retrieve the values of interest.

*# Use a list comprehension to grab all elements at a given index*

*# in all the dictionary values (which are lists)*

​

**def** column\_sum(column\_index, dictionary):

**return** round(sum([value[column\_index] **for** value **in** dictionary.values()]), 2)

#### Sum The Subtotal, Tax, and Total Columns

Now that we have our column sum function, calculate the sum of:

* Subtotals
* Taxes
* Totals

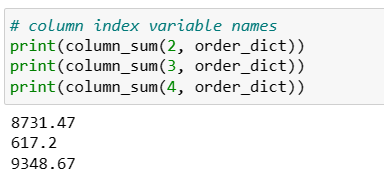
*# column index variable names*

print(column\_sum(2, order\_dict))

print(column\_sum(3, order\_dict))

print(column\_sum(4, order\_dict))

**Output:**



### What is the average of our subtotals?

Calculate the average value of our transactions.

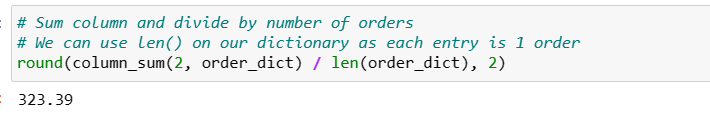
Remember that each entry in your dictionary is one order.

*# Sum column and divide by number of orders*

*# We can use len() on our dictionary as each entry is 1 order*

round(column\_sum(2, order\_dict) **/** len(order\_dict), 2)

**Output:**



*# average transaction without function*

*# Use a list comprehension to collect all subtotals,*

*# then sum and divide by the length*

​

subtotals **=** [value[2] **for** value **in** order\_dict.values()]

​

round(sum(subtotals) **/** len(subtotals), 2)

**Output:**

### 

### How many unique customers did we have?

Calculate the total number of unique customers in our sales data.

Then calculate the number of orders per customer (total orders/ unique customers).

*# Use a list to get all customer\_Ids, create a set to return only unique*

*# then use len to return count of unique customers*

unique\_customers **=** len(set([value[0] **for** value **in** order\_dict.values()]))

​

*# divide total number of orders by unique customers*

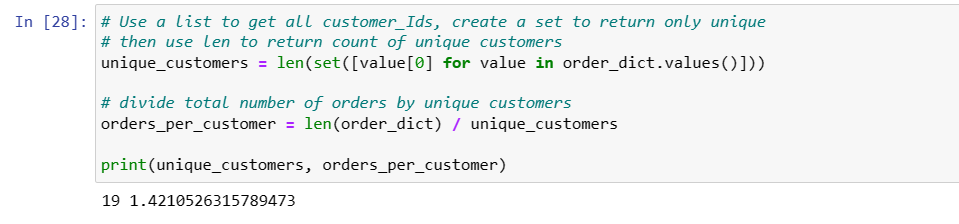
orders\_per\_customer **=** len(order\_dict) **/** unique\_customers

​

print(unique\_customers, orders\_per\_customer)

19 1.4210526315789473

**Output:**



### How many items in total did we sell?

Calculate the total number of items we sold in across all orders.

This information is in Column H, which should be the last element in order\_dict's values.

*# Use a list comprehension to get the length of items ordered*

*# for each order (the length equals number of products), then sum*

order\_dict.values()

*#sum([len(value[6]) for value in order\_dict.values()])*

**Output:**

### Sales By Location

Calculate the sum of subtotals by location.

Create a dictionary to store them, where location is the key, and revenue for that location is the value.

A few steps to consider:

* Loop through you dictionary
* build a dictionary as you go with location as key
* increment revenue every time a transaction matches the location.

Your output should look like {'Location1': sum of subtotals for 'Location1'}

With an entry for each location.

*# create empty dictionary*

location\_sums **=** {}

​

*# loop through dictionary values*

**for** data **in** order\_dict.values():

*# set location variable*

location **=** data[5]

*# if that location is not yet in location\_sums*

**if** location **not** **in** location\_sums:

*# create dictionary entry*

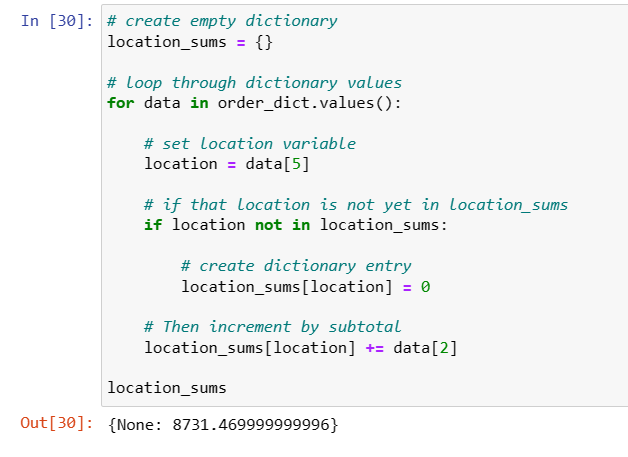
location\_sums[location] **=** 0

*# Then increment by subtotal*

location\_sums[location] **+=** data[2]

location\_sums

**Output:**



### Challenge: Aggregator Function

Now that you've summed revenue by category, can you write a function to generalize calculating a sum of a column, grouped by the unique values in another column? (for example, sum of totals by date or customer\_id).

Your function should take the following arguments:

* index of the 'column' (index position in order\_dict) to group by
* index of the 'column' (index position in order\_dict) to sum by category
* the dictionary where the data is located (assume the same structure as order\_dict.

It should return:

* A dictionary with the categories as keys, and the sum by category as value.

Once you've done so use your function to sum totals by date and customer\_id.

**def** aggregator(category\_index, field\_to\_sum\_index, dictionary):

*# build empty dictionary*

category\_sums **=** {}

*# loop through dictionary values*

**for** data **in** dictionary.values():

*# assign the category of interest in that entry*

category **=** data[category\_index]

*# if that category is not in our dictionary, add it*

**if** category **not** **in** category\_sums:

*# start revenue at 0*

category\_sums[category] **=** 0

*# then increment category revenue*

category\_sums[category] **+=** data[field\_to\_sum\_index]

*# once we've looped through dictionary, return the category\_sums*

**return** category\_sums

For fun, if you got tired of looking up indices, you could assign your indices names. In libraries like Pandas we'll be able to reference DataFrame column names or indices.

Here we're unpacking a tuple to name our column indices

*# for fun, if you got tired of looking up indices, you could*

*# assign your indices names*

*# Here we're unpacking a tuple to name our column indices*

customer\_id, date, subtotals, taxes, totals, location, items\_sold **=** (0, 1, 2, 3, 4, 5, 6)

​

*# Function call with numeric indices*

aggregator(1, 2, order\_dict)

*# Function call with 'named' indices*

aggregator(customer\_id, totals, order\_dict)

**Output:**

