C3M1_Assignment

November 17, 2025

1 Module 1 Assignment: Analyzing retail sales with Python

Welcome to this module's assignment! This is the first graded lab you will complete in Python. Exciting, right?

When you are done, submit your solution by saving it, then clicking on the blue submit button at the top of the page.

1.1 Background:

You just started working for a small retail store selling household appliances. The owners are an elderly couple who want to start digitizing their sales for analysis to stay competitive. You start by recording the sales for the first two weeks (Mon-Sat), resulting in the data below. You will work with the following features:

- `invoice_id`: id of the purchase. The invoice_id will be repeated as many times as different products were in the purchase. In other words, there will be one observation (row) for each product sold in each purchase.
- `date`: date of the purchase
- `day_of_week`: Monday - Saturday
- `product_name`: description of the product
- `units_sold`: number of units sold of the product
- `price_per_unit`: price for each individual unit of the product

1.2 In order for your submission to be graded correctly, you MUST:

- **Use the provided variable names**, otherwise the autograder will not be able to locate the variable for grading.
- **Replace any instances of `None` with your own code.**
- **Only modify the cells that start with the comment `# GRADED CELL`.**
- **Use the provided cells for your solution.** You can add new cells to experiment, but these will be omitted when grading.

To submit your solution, save it, then click on the blue submit button at the top of the page.

Important note: Code blocks with None will not run properly. If you run them before completing the exercise, you will likely get an error.

2 Table of Contents

- Step 1: Load the data
  - Exercise 1: Number of observations
- Step 2: Data Exploration
  - Preliminary Exploration: Unit Prices and Total Amount of Items Sold
    - Exercise 2: Priciest Item
    - Exercise 3: Find the total number of units sold
    - Adding sales to the dataset
      - Exercise 4: Adding elements to the lists
    - Finding the sale amount
      - Exercise 5: Find the sales amount for each observation
      - Exercise 6: Total and average amounts
    - Sales per product
      - Exercise 7: Find all the unique products
      - Exercise 8: Sales amount and number of sold units per product

2.1 Step 1: Load the data

```
<strong>   Directions</strong>
    <ol>
        <li>Run the cell below to load the data with the aid of a helper function.</li>
    </ol>
```

Each feature is contained in its own list. For example, the variable `invoice_ids` is a list containing the sales identification numbers. It has the same elements as the `invoice_id` column of the CSV. Each element of the list corresponds to a row in the `invoice_id` column.

If you look at the first element of each variable, you're accessing the same information as the second row of the spreadsheet (since the first row contains headers). Remember that Python uses 0-based indexing.

[ ]:
```
#   This cell is locked. You will not be able to edit it.

from helper_functions import get_list

invoice_ids = get_list("invoice_id")
dates = get_list("date")
days_of_week = get_list("day_of_week")
product_names = get_list("product_name")
units_sold = get_list("units_sold")
prices_per_unit = get_list("price_per_unit")
```

[ ]:
```
# print the first element of each list
print("invoice_ids", invoice_ids[0])
print("dates", dates[0])
print("days_of_week", days_of_week[0])
print("product_names", product_names[0])
print("units_sold", units_sold[0])
```

```
print("prices_per_unit", prices_per_unit[0])
```

Exercise 1: Number of observations

Use the next cell to find the length of the list `invoice_ids`. Save it in the variable `num_observations`.

[ ]:
```
# GRADED CELL: Exercise 1

### START CODE HERE ###

num_observations = None

### END CODE HERE ###

print(num_observations)
```

- Remember that the `len()` function returns the number of items in a list

```
<strong>  Directions</strong>
    <ol>
        <li>Run the cell below to print the lengths of all the variables.</li>
        <li>Use the results to check that all the lists have the same length.</li>
    </ol>
```

[ ]:
```
print("invoice_ids: ", len(invoice_ids))
print("dates: ", len(dates))
print("days_of_week: ", len(days_of_week))
print("product_names: ", len(product_names))
print("units_sold: ", len(units_sold))
print("prices_per_unit: ", len(prices_per_unit))
```

Now, let's see what types of data you are dealing with. For that you can use the `type` function. This function returns what type of variable (a list, an int, a float, a str, etc.) you are dealing with.

```
<strong>  Directions</strong>
    <ol>
        <li>Run the cell below to print the types of the variables `invoice_id` and `day_of_wee
    </ol>
```

[ ]:
```
print("invoice_ids: ", type(invoice_ids))
print("days_of_week: ",type(days_of_week))
```

The `type` function tells you that both variables are lists. While that is true, it's not very informative about the type of data each list is storing. How could you get that information?

```
<strong>  Directions</strong>
    <ol>
        <li>Run the cell below to print the type of the first element of each list.</li>
    </ol>
```

```
[ ]: print("invoice_ids: ", type(invoice_ids[0]))
     print("dates: ", type(dates[0]))
     print("days_of_week: ", type(days_of_week[0]))
     print("product_names: ", type(product_names[0]))
     print("units_sold: ", type(units_sold[0]))
     print("prices_per_unit: ", type(prices_per_unit[0]))
```

2.2 Step 2: Data exploration

After recording the data for the shop owners and loading it into your Python notebook, your next step is to explore the dataset. You decide to begin with a few straightforward analyses to get to know the data.

2.2.1 Preliminary exploration: unit prices, and total amount of items sold

Exercise 2: Price range Complete the cell below to find the price range of the products. Start with finding the price of the most expensive and the cheapest products in the dataset. Save the values in two variables called `highest_price` and `lowest_price` respectively. Then subtract the two to find the range of prices and save it to the `price_range` variable.

```
<strong>   Directions</strong>
    <ol>
        <li>Define a variable <code>highest_price</code> and use an appropriate function to fi
        <li>Define a variable <code>lowest_price</code> and use an appropriate function to fin
        <li>Define a variable called `price_range` and calculate the difference between the hig
    </ol>
```

- Remember that the `max()` function returns the highest value in a list
- Remember that the `min()` function returns the lowest value in a list

```
[ ]: # GRADED cell: Exercise 2

     ### START CODE HERE ###

     # Find the highest price in the prices_per_unit variable
     highest_price = None

     # Find the lowest price in the prices_per_unit variable
     lowest_price = None

     # Find the price range
     price_range = None

     ### END CODE HERE ###
```

```
[ ]: # Print out the results!
     print("Highest price:", highest_price)
     print("Lowest price:", lowest_price)
     print("Price range:", price_range)
```

Exercise 3: Find the total number of units sold

Next, find the total number of units sold and save it in the variable `total_units_sold`.

```
<strong>   Directions</strong>
    <ol>
        <li>Use an appropriate function to calculate the sum of all the <code>units_sold</code
    </ol>
```

- Remember that the `sum()` function returns the sum of all the items.
- If the grader tells you that total_units_sold in both Exercise 3 and Exercise 6 are wrong, please check first that you updated the units sold in Exercise 6 (hint: it should increase by more than 1). If you fixed that there, this exercise will likely also be correctly graded without changing anything.

[ ]:
```
# GRADED cell: Exercise 3

### START CODE HERE ###

total_units_sold = None

### END CODE HERE ###
```

[ ]:
```
# Print the result!
print("Total number of units sold: ", total_units_sold)
```

Adding sales to the dataset

Exercise 4: Adding elements to the lists You just realized that you forgot to load the last sale that happened on Saturday! Here's the info for that sale:

Column	Value
invoice_id	536994
date	"2010-12-04"
day_of_week	"Saturday"
product_name	"CHRISTMAS CRAFT TREE TOP ANGEL"
units_sold	2
price_per_unit	2.1

Use the next cell to add this purchase to all the variables. Replace all the `None` placeholders with your solution.

```
<strong>   Directions</strong>
    <ol>
        <li>Add the new value to the <code>invoice_ids</code> variable. </li>
        <li>Repeat for <code>dates</code>, <code>days_of_week</code>, <code>product_names</code
    </ol>
```

If you are stuck, click here for extra hints!

- Remember that lists have the `append()` method, which adds an element at the end of the list.

[ ]:
```
# GRADED cell: Exercise 4 a

### START CODE HERE ###

invoice_ids.None
dates.None
days_of_week.None
product_names.None
units_sold.None
prices_per_unit.None

### END CODE HERE ###
```

[ ]:
```
# Print the length of invoice_ids to check that you have one extra item
print("The new length of invoice_ids is ", len(invoice_ids))
```

Go ahead and update the number of observations, so it matches the new length of the lists.

```
<strong>   Directions</strong>
    <ol>
        <li>Update the variable <code>num_observations</code> </li>
    </ol>
```

If you are stuck, click here for extra hints!

- You can overwrite `num_observations` with the new length of any of the variables (invoice_id, date, or the one you prefer)
- OR you can add 1 to the current value stored in `num_observations`, because you added just one element to the lists.

[ ]:
```
# GRADED cell: Exercise 4b

### START CODE HERE ###

# Update the number of observations
num_observations = None

### END CODE HERE ###
```

Finding the sales amount

Exercise 5: Find the sales amount for each observation.

Your bosses are interested in knowing the total purchase amount sold for each observation (number of units sold times product price). This way, they can tell which of the products are creating the most value.

```
<strong>   Directions</strong>
```

```
<ol>
    <li>Use the <a href="https://docs.python.org/3/library/functions.html#func-range"> <co
    <li>For each index, multiply the number of units sold for the product at index i by the
    <li>Append this product to the <code>amounts</code> variable.
</ol>
```

If you are stuck, click here for extra hints!

- If you want to iterate over the entire list, you need to use `range()` with one argument: the length of the list (or `num_observations - 1`)
- To get the amount for the observation at index i, you need to multiply the value stored in `units_sold[i]` by the one stored in `price_per_unit[i]`
- Remember that lists have the `append()` method, which adds an element at the end of the list.

[ ]:
```
# GRADED cell: Exercise 5

# First, define an empty list. You will be adding the amounts to this list
amounts = []

### START CODE HERE ###

# Iterate over all the elements the dataset.
for i in None:
    # calculate the sale amount for observation i
    amount = None
    # append this value to amounts
    amounts.None

### END CODE HERE ###
```

[ ]:
```
# Print the first 5 elements to see everything worked ok
print("First 5 items from units_sold: ", units_sold[:5])
print("First 5 items from prices_per_unit: ", prices_per_unit[:5])
print("First 5 items from amounts: ", amounts[:5])
```

Exercise 6: Total and average amounts

Now that you have the total sale amount for each product in each purchase, the owners would like you to calculate the average sale price, to help them better understand their customers.

```
<strong>  Directions</strong>
    <ol>
        <li>Create a new variable, <code>total_amount</code>, and store the sum of all sales a
        <li>Update the <code>total_units_sold</code> variable (first defined in Exercise 3) be
        <li>Use a the new variable, <code>average_price</code>, to find the average sales price
    </ol>
```

If you are stuck, click here for extra hints!

- Remember that the `sum()` function returns the sum of all the items.

- Remember that the average is nothing more than the total sum (which you just saved in the `total_amount` variable), divided by the total number of items
- Check the figure in Exercise 4 to see how many additional units were sold. It is more than 1.
- The total number of items, in this case, is not the number of observations, but rather the total number of units sold.

[ ]:
```
# GRADED cell: Exercise 6

### START CODE HERE ###

# Find the total amount sold
total_amount = None

# Update the total units sold
total_units_sold = None

# Find the average price for all sold items using total_amount and␣
 ↪total_units_sold
average_price = None

### END CODE HERE ###
```

[ ]:
```
# Print the results!
print("The total sales amount in two weeks was ", total_amount, ", with an␣
 ↪average unit price of ", average_price)
```

Sales per product In just two weeks, it is very likely that you didn't sell any units of some products. And, since Christmas is right around the corner, many sales are holiday related. That's why the lovely owners want your help to identify all the unique products sold during this two-week period.

Exercise 7: Find all the unique products To complete this exercise you will be given a skeleton of what the code should look like.

```
<strong>  Directions</strong>
    <ol>
        <li>Iterate over all the product names using a for loop. </li>
        <li>For each iteration check if the current <code>product_name</code> is already in the
        <li>If it is not, append it to the list of unique products.</li>
    </ol>
```

If you are stuck, click here for extra hints!

- You can create an empty list using empty brackets: `my_list = []`
- To check if a value is inside a list, you can use the comparison `in`. For example:
  - `0 in [3, 2, 1, 4]` should return False
  - `0 in [2, 0, 6]` should return True
- Remember to use the `if` statement to branch your code.
- To append the new product name, you can use the `append()` method for lists, which adds an element at the end of the list.

[ ]:
```
# GRADED cell: Exercise 7

# Create an empty list for the unique product names
unique_products = []

### START CODE HERE ###

# Iterate over all the observations in the dataset using a for loop
for name in None:
    # Check if the current product_name is not in the unique_products list:
    if name not in unique_products:
        # If the expression holds True, then add the current product_name
        # to the unique_products list
        unique_products.None

### END CODE HERE ###
```

[ ]:
```
# Print your results!
print("There are ", len(unique_products), " unique products:")
for product in unique_products:
    print("- ", product)
```

LLM Exercise: Understanding Built-in statements

You have used the `in` statement to find whether a string was already in a list of values or not. To get a better grasp at how it works, ask the LLM to summarize how this method works

```
<strong>Prompt</strong>
```

Please explain how the ***in*** *statement works in Python from this line of code* ***if name not in unique_products****. If relevant, please point to any official documentation so I can read more about* ***in****.*

Exercise 8: Sales amount and number of sold units per product With Christmas fast approaching and the temperatures dropping, your bosses are considering a "Get 50% OFF the second item" sale on select products. They're deciding between the "CHRISTMAS CRAFT TREE TOP ANGEL" and the "HAND WARMER BIRD DESIGN." To assist in their decision-making, they need information on the average number of units sold per purchase. Help them gather this data!

```
<strong>  Directions</strong>
    <ol>
        <li>Iterate over all the list indexes using a <code>for</code> loop. You will need to u
        <li>For each iteration, if the item matches either "HAND WARMER BIRD DESIGN" or "CHRIST
    </ol>
```

If you are stuck, click here for extra hints!

- To iterate over all items in a list, you can use the `range(len(listname))` code, replacing `listname` with the name of a list. Since each list in your data is the same length, you can use any one.

- To check if the product names match one of the products, you can use the comparison `==`
- Since you want to check if it matches one product or the other, you will need to use an `if` statement followed by and `elif`. What you are doing is checking if the current name matches, for example "HAND WARMER BIRD DESIGN". If it does, then you append the units sold to `units_per_sale_handwarmer`. If it doesn't, you check if it matches with "CHRISTMAS CRAFT TREE TOP ANGEL". If it does, then you append the units sold to `units_per_sale_christmas_angel`. If it doesn't you don't need to do anything else, so there is no need for any more code.

[ ]:
```
# GRADED cell: Exercise 8

# Create two empty lists to fill with the number of units sold
units_per_sale_handwarmer = []
units_per_sale_christmas_angel = []

### START CODE HERE ###

# Iterate over each product.
for i in None:
    # Using an if statement, check if the product_name at index i matches "HAND␣
 ↪WARMER BIRD DESIGN"
    if None:
        # If true, append the number of units sold to units_per_sale_handwarmer
        None

    # Now check if the product_name at index i matches "CHRISTMAS CRAFT TREE␣
 ↪TOP ANGEL"
    elif None:
        # If true, append the number of units sold to␣
 ↪units_per_sale_christmas_angel
        None

### END CODE HERE ###
```

[ ]:
```
# Print your results
print("The number of units of HAND WARMER BIRD DESIGN sold:",␣
 ↪sum(units_per_sale_handwarmer))
print("The number of units of CHRISTMAS CRAFT TREE TOP ANGEL sold:",␣
 ↪sum(units_per_sale_christmas_angel))
```

Congratulations on finishing this graded lab! If everything is running correctly, you can go ahead and submit your code for grading.