A picture containing logo

Description automatically generated

|  |  |
| --- | --- |
| **STUDENT NAME:** | **Isaac Christopher Ong Wen Xuan** |
| **TITLE:** | **TMA01** |
| **COURSE CODE:** | **ANL252, Python for Data Analytics** |
| **PI NUMBER:** | **Q2180896** |
| **TG:** | **03** |
| **SUBMISSION DATE:** | **14 September 2023** |

**Q1A)**

**Open-source Software**

Online Repositories such as Github and Stack Overflow makes solution readily available (Lee, 2020). These platforms encourage collaboration among coders to build on ideas which differs from an academic setting, where assignments demonstrates knowledge of concepts.

**Lack of understanding**

Students who do not grasp coding concepts will rely heavily on copying codes they have access to. Without a strong understanding, they are unable to edit or adjust the reference code and simply copy the whole code which results in plagiarism.

**Write Original Codes**

To avoid plagiarism, coders can create original solutions without relying on online materials. This requires a high level of understanding and competency which takes time and practice to understand the logic behind the code.

**Documentation**

To demonstrate concept knowledge when coding assignments, students can document the coding process improving readability and usability of the code (Ayomide, 2022). This helps explain the thought process and what algorithms were used in the code to showcase original codes.

**Citations and attributions**

To prevent plagiarism, any function or algorithm that came from someone else should be cited or attributed to give the creator credit (University of Arkansas, 2023). Additionally, citing codes helps future researchers reproduce and use the code.

Word count: 200 words

**Q1B)**

This code compares the prices of 5 clothing brand and calculate the average price of the items in store.

First, it lists down the prices of each clothing by brand and store it into a dictionary ‘brand\_price’. Each brand has a NumPy array with their clothing price. Using NumPy arrays makes the code more compact which reduces clutter and increases efficiency (NumPy, 2022).

Next, the code calculates the mean and standard deviation price of each store using NumPy functions. The mean calculates the average price of each store and standard deviation measures how much the price ranges.

To calculate the market average price, the code combines all the prices into a single NumPy array to understand market positioning of each brand by comparing the store average to the market average. With this data, we can categorize the brands into 3 categories (luxury, fast fashion, budget friendly). Luxury brands have a higher mean than the average, fast fashion have the same mean, and budget friendly brands have lower mean prices than the market average.

The code displays the results of a brand mean price and standard deviation followed by the market statictic and the brand categorization.

Word count: 195

import numpy as np

# Price of clothes

brand\_price = {

'Brand A': np.array([100, 110, 127, 96, 130]),

'Brand B': np.array([28, 33, 42, 38, 52]),

'Brand C': np.array([10, 45, 21, 37, 29]),

'Brand D': np.array([70, 65, 55, 75, 62]),

'Brand E': np.array([80, 55, 75, 60, 65]) }

# Calculate mean and standard deviation for each brand

individual\_means = {brand: np.mean(prices) for brand, prices in brand\_prices.items()}

individual\_std\_devs = {brand: np.std(prices) for brand, prices in brand\_prices.items()}

# Calculate mean and standard deviation for all brands

all\_prices = np.concatenate(list(brand\_prices.values()))

collective\_mean = np.mean(all\_prices)

collective\_std\_dev = np.std(all\_prices)

# Categorize brands into 3 categories based on price

price\_categories = {

'Luxury': [brand for brand, mean\_price in individual\_means.items() if mean\_price > collective\_mean],

'Fast Fashion': [brand for brand, mean\_price in individual\_means.items() if mean\_price == collective\_mean],

'Budget Friendly': [brand for brand, mean\_price in individual\_means.items() if mean\_price < collective\_mean]

}

# Output

print("Individual Mean Prices:")

for brand, mean\_price in individual\_means.items():

print(f"{brand}: {mean\_price:.2f}")

print("\nIndividual Standard Deviations:")

for brand, std\_dev in individual\_std\_devs.items():

print(f"{brand}: {std\_dev:.2f}")

print("\nCollective Mean Price:", collective\_mean)

print("Collective Standard Deviation:", collective\_std\_dev)

print("\nBrand Price Categories:")

for category, brands in price\_categories.items():

print(f"{category}: {brands}")

Code Reference:

[*https://numpy.org/doc/stable/reference/generated/numpy.concatenate.html*](https://numpy.org/doc/stable/reference/generated/numpy.concatenate.html) *retrieved on 12 Sept 2023*

*ANL252 Python for Data Analytics Study Guide retrieved on 12 Sept 2023*

**Q1C)**

* **Mathematical calculation for the categorization**

The rewritten code categorizes the brands by considering the market’s mean price and standard deviation. This improves categorization precision as it considers the mean price and how consistent the prices are. This ensures that outliers with uniquely high or low prices would not skew the data as the formula considers the variability of the market price. This results in an accurate price range categorization, with expensive would be 1 standard deviation above the market average while the affordable category will be 1 standard deviation below. This also gives the moderate category a range between unlike code 1 where the brand mean price are compared to the market mean.

* **Improved output with 2 decimal place and dollar sign**

Rewriting the code improved the display results making it clearer for viewers to understand the data. Furthermore, adding ‘$’ helps users to distinguish between the mean price and standard deviation.

* **Code structure**

By separating the categorization from other calculations, the rewritten code makes it easier for other coders to understand the code. This enhances the maintainability and extensibility of the code which increases the scalability (Murphy, 2022). Having a legible code is important to edit or import more data into the code without having to change the main flow and logic of it.

* **Reduce calculation**

The rewritten codes calculate the market mean and standard deviation once and assigns a value to it before using that value throughout the rest of the code. This is an improvement from Code 1 where the code is required to calculate the both the mean and brands’ mean and standard deviation every time a brand is being categorized. Therefore, instead of linking the categorization to the calculation, we can make the code run faster by directly linking it to the value.

Word Count: 298

import numpy as np

# Define prices for clothing items

brand\_prices = {

'Brand A': np.array([100, 110, 127, 96, 130]),

'Brand B': np.array([28, 33, 42, 38, 52]),

'Brand C': np.array([40, 45, 21, 37, 29]),

'Brand D': np.array([70, 65, 55, 75, 62]),

'Brand E': np.array([80, 55, 75, 60, 65])

}

# Calculate mean and standard deviation for each brand

brand\_means = {brand: np.mean(prices) for brand, prices in brand\_prices.items()}

brand\_std\_devs = {brand: np.std(prices) for brand, prices in brand\_prices.items()}

# Calculate mean and standard deviation for all brands

all\_prices = np.concatenate(list(brand\_prices.values()))

market\_mean = np.mean(all\_prices)

market\_std\_dev = np.std(all\_prices)

# Categorize brands into price ranges

price\_ranges = {

"Affordable": (market\_mean - market\_std\_dev, market\_mean),

"Moderate": (market\_mean, market\_mean + market\_std\_dev),

"Expensive": (market\_mean + market\_std\_dev, max(all\_prices))

}

# Create dictionary to store category

brand\_categories = {category: [] for category in price\_ranges}

# Categorize each brand

in brand\_prices:

mean\_price = brand\_means[brand]

std\_dev = brand\_std\_devs[brand]

for category, (lower\_limit, upper\_limit) in price\_ranges.items():

if lower\_limit <= mean\_price <= upper\_limit and std\_dev < market\_std\_dev / 2:

brand\_categories[category].append(brand)

# Display results

print("Brand Mean Prices:")

for brand, mean\_price in brand\_means.items():

print(f"{brand}: ${mean\_price:.2f}")

print("\nBrand Standard Deviations:")

for brand, std\_dev in brand\_std\_devs.items():

print(f"{brand}: {std\_dev:.2f}")

print("\nMarket Mean Price: ${:.2f}".format(market\_mean))

print("Market Standard Deviation: {:.2f}".format(market\_std\_dev))

print("\nBrand Categories:")

for category, brands in brand\_categories.items():

print(f"{category}: {brands}")

*Code References:* [*https://www.tutorialspoint.com/python-pandas-categoricalindex-get-the-category-codes-of-this-categorical*](https://www.tutorialspoint.com/python-pandas-categoricalindex-get-the-category-codes-of-this-categorical) *retrieved on 14 Sept 2023*

1B Output:

Brand Mean Prices:

Brand A: 112.60

Brand B: 38.60

Brand C: 34.40

Brand D: 65.40

Brand E: 67.00

Brand Standard Deviations:

Brand A: 13.79

Brand B: 8.19

Brand C: 8.48

Brand D: 6.83

Brand E: 9.27

Market Mean Price: 63.6

Market Standard Deviation: 29.513386793114748

Market Price Categories:

High Price: ['Brand A', 'Brand D', 'Brand E']

Average Price: []

Low Price: ['Brand B', 'Brand C']

1C Output:

Brand Mean Prices:

Brand A: $112.60

Brand B: $38.60

Brand C: $34.40

Brand D: $65.40

Brand E: $67.00

Brand Standard Deviations:

Brand A: 13.79

Brand B: 8.19

Brand C: 8.48

Brand D: 6.83

Brand E: 9.27

Market Mean Price: $63.60

Market Standard Deviation: 29.51

Brand Categories:

Affordable: ['Brand B', 'Brand C']

Moderate: ['Brand D', 'Brand E']

Expensive: ['Brand A']

**Q2)**

1. Include comments before every function to explain what that portion of the code is doing for better understanding and reproducibility in the future (IWConnect, 2023). This serves as a reference point for future coders, which enhances collaboration and knowledge sharing.
2. Have a dictionary instead of lists to helps to store data as a key-value pair which is useful when matching prices to products (Janghu, 2023). Furthermore, using a list makes the code more scalable in future and reduces errors.
3. Currently, if the user input a product that is not part of the list, it would break the cycle and all their previous selection would not appear. Instead of the break function, using a continue function allows user to re-enter their selection without having to start over.
4. Similarly, we can add in a loop function to allow users to retry entering the price and having an output to remind them that they need to enter a numerical number
5. Improve the layout of the final output to show the item beside the price and fix typo from “our” to “your” shopping list.

# Import product and price into a dictionary

products = {

'laptop': 1000,

'mouse': 20,

'webcam': 50,

'keyboard': 30,

'speaker': 40

}

updated\_items = []

# List the available products

print('Our Products:')

for product in products:

print(f'- {product}')

while True:

item = input("Hello! What do you want to buy? ")

if item not in products:

print(f'Wrong product! Please try again.')

continue

while True:

price\_of\_item = input(f"How much is {item} (in SGD)? ")

try:

price = float(price\_of\_item)

break

# Exit the loop if price is a valid number

except ValueError:

print('Invalid price! Please enter a valid number.')

updated\_items.append((item, price))

continue\_shopping = input("Would you like to add another item to your shopping list? (yes/no) ")

if continue\_shopping.lower() != 'yes':

break

#Display final items and prices

print('This is your shopping list:')

for item, price in updated\_items:

print(f'{item}: {price} SGD')

# References

Ayomide, S. I. (2022, September 14). *How To Write Code Documentation* . Retrieved from MadCap: https://www.madcapsoftware.com/blog/write-code-documentation/#:~:text=Code%20documentation%20is%20the%20process,readability%2C%20reproducibility%2C%20and%20usability.

IWConnect. (2023). *The Benefits of Code Documentation: Why it’s a Must-Have for Your Development Team* . Retrieved from IWConnect: https://iwconnect.com/the-benefits-of-code-documentation-why-its-a-must-have-for-your-development-team/

Janghu, R. (2023, August 14). *Difference Between List and Dictionary in Python* . Retrieved from scaler : https://www.scaler.com/topics/difference-between-list-and-dictionary-in-python/

Lee, C. (2020, July 28). *What is Programming Plagiarism? Why Is It on the Rise?* Retrieved from turnitin: https://www.turnitin.com/blog/what-is-programming-plagiarism-why-is-it-on-the-rise

Murphy, C. (2022, November 9). *Best Code Practices for Scalability, Collaboration, and More*. Retrieved from prismic: https://prismic.io/blog/best-code-practices-scalability-collaboration

NumPy. (2022). *NumPy: the absolute basics for beginners*. Retrieved from NumPy: https://numpy.org/doc/stable/user/absolute\_beginners.html

University of Arkansas. (2023, July 26). *Computer Science & Computer Engineering* . Retrieved from University of Arkansas: https://uark.libguides.com/CSCE/CitingCode