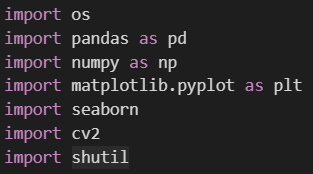
In the 3rd week, we examined our data and decided how to pull certain data from the data. Below, we have printed in detail how we examine our transaction sequences and data. We have explained which libraries we will use and which parts of the data we will use.

We tried to research and print it in detail so that we would not forget what we did in the codes in the future.



1. **os:**
   * Python built-in library for interacting with the operating system. It provides a way of using operating system-dependent functionality, like reading or writing to the file system.
2. **pandas (as pd):**
   * Pandas is a powerful data manipulation and analysis library for Python. It provides data structures like DataFrames, which are similar to tables in a relational database, making it easy to analyze and manipulate structured data.
3. **numpy (as np):**
   * NumPy is a fundamental package for scientific computing with Python. It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays.
4. **matplotlib.pyplot (as plt):**
   * Matplotlib is a 2D plotting library for Python. **pyplot** is a collection of functions that make it easy to create various kinds of plots and charts.
5. **seaborn:**
   * Seaborn is a statistical data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
6. **cv2:**
   * OpenCV (Open Source Computer Vision) is a library for computer vision and image processing. **cv2** is the OpenCV module for Python.
7. **shutil:**
   * The **shutil** module provides a higher-level interface for file operations. It includes functions for copying files and directories, archiving, and more.

In summary:

* **os** is used for interacting with the operating system.
* **pandas** and **numpy** are used for data manipulation and analysis.
* **matplotlib.pyplot** and **seaborn** are used for data visualization.
* **cv2** is used for computer vision and image processing.
* **shutil** is used for file operations.

metin, yazı tipi, ekran görüntüsü, grafik içeren bir resim

Açıklama otomatik olarak oluşturuldu

1. **new\_directory = '/kaggle/working/tas/':**
   * This line creates a variable named **new\_directory** and assigns it the string '/kaggle/working/tas/'. This string represents the path to the new directory that you want to create. In this case, it looks like the directory is intended to be created in the '/kaggle/working/' path.
2. **os.makedirs(new\_directory, exist\_ok=True):**
   * The **os.makedirs** function is then called to create the directory specified by the **new\_directory** path.
   * The **exist\_ok=True** parameter is used to prevent raising an exception if the directory already exists. If **exist\_ok** is set to **True** and the directory already exists, the function will not raise an error; otherwise, it would raise a **FileExistsError**.

So, the overall purpose of this code is to create a new directory at the specified path ('/kaggle/working/tas/') in the file system. If the directory already exists, it will not raise an error due to the **exist\_ok=True** parameter. This is a common practice in data science and machine learning projects to organize and store results, logs, or intermediate files in a structured manner.





1. **pd.read\_csv('/kaggle/input/fashion-product-images-dataset/fashion-dataset/styles.csv', on\_bad\_lines='skip'):**
   * **pd.read\_csv** is a function from the **pandas** library used to read data from a CSV (Comma-Separated Values) file into a DataFrame.
   * The argument **'./fashion-dataset/styles.csv'** specifies the path to the CSV file to be read. In this case, the file is located in the '/kaggle/input/fashion-product-images-dataset/' directory.
   * The **on\_bad\_lines='skip'** parameter is used to skip lines with too many fields instead of raising an error. This is useful when dealing with potentially messy data where some lines may not conform to the expected structure.
2. **data.head():**
   * After reading the CSV file, **data.head()** is used to display the first few rows of the DataFrame. By default, **head()** displays the first 5 rows, but you can specify a different number inside the parentheses if you want to see more or fewer rows.

So, the overall purpose of this code is to read a CSV file containing fashion product data into a pandas DataFrame named **data**, and then display the first few rows of that DataFrame using the **head()** method. This is a common initial step in a data analysis or machine learning project to understand the structure and contents of the dataset.





1. **data['articleType']:**
   * This part of the code is selecting the 'articleType' column from the DataFrame **data**. The result is a pandas Series containing the values in the 'articleType' column.
2. **.unique():**
   * The **unique()** method is then called on the selected Series. This method returns an array of unique values in the Series, preserving the order of their first occurrence.

So, the overall purpose of this code is to extract the unique values from the 'articleType' column in the **data** DataFrame. This can be useful for understanding the different categories or types of articles present in the dataset. It's a common step in data analysis to explore the unique values in categorical columns to gain insights into the variety of data within that particular feature.



1. **IMG\_PATH = '/kaggle/input/fashion-product-images-dataset/fashion-dataset/images/':**
   * This line creates a variable named **IMG\_PATH** and assigns it the string '/kaggle/input/fashion-product-images-dataset/fashion-dataset/images/'.
   * The variable name suggests that this path points to a directory containing images related to a fashion product dataset. This is a common practice in machine learning and computer vision projects where images are stored in a specific directory.
2. **SAVE\_PATH = '/kaggle/working/tas/':**
   * This line creates a variable named **SAVE\_PATH** and assigns it the string '/kaggle/working/tas/'.
   * The variable name suggests that this path is intended for saving or storing processed or generated results. The 'tas' directory might be used to organize files or data generated during the course of the analysis or modeling.

In summary, these paths are often used in data science and machine learning projects to specify the locations of input data (such as images in **IMG\_PATH**) and the locations where results or processed data should be saved (such as in **SAVE\_PATH**). The specific use of these paths would depend on the context of the project and the tasks being performed.



1. **data.groupby(['articleType']):**
   * The **groupby** method is used to group the DataFrame **data** by the values in the 'articleType' column. This creates separate groups for each unique value in the 'articleType' column.
2. **.agg(count\_img=('articleType', 'count')):**
   * The **agg** method is then used to perform an aggregation operation on each group. In this case, it's counting the number of occurrences in each group. The result is a new DataFrame with a column named 'count\_img' that contains the count of occurrences for each 'articleType' group.
3. **.sort\_values(by='count\_img', ascending=False):**
   * The **sort\_values** method is used to sort the DataFrame based on the values in the 'count\_img' column in descending order (**ascending=False**). This is done so that the article types with the highest counts appear first in the result.
4. **.reset\_index():**
   * The **reset\_index** method is used to reset the index of the DataFrame, and it converts the grouped 'articleType' from an index to a regular column. This is often done after grouping and aggregation operations to make the DataFrame more readable and accessible.
5. **data\_per\_type:**
   * Finally, the result is assigned to the variable **data\_per\_type**, which now holds a DataFrame summarizing the count of occurrences for each unique 'articleType' in the original **data** DataFrame.

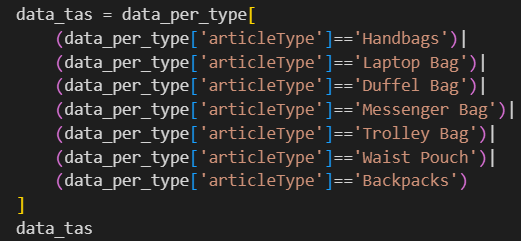
So, the overall purpose of this code is to create a new DataFrame (**data\_per\_type**) that summarizes the count of occurrences for each unique 'articleType' in the original **data** DataFrame. The result is sorted in descending order based on the counts. This type of analysis is useful for understanding the distribution of different article types in the dataset.

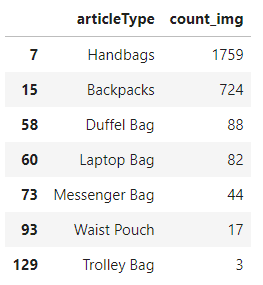




1. **data\_per\_type['articleType']:**
   * This part of the code is selecting the 'articleType' column from the DataFrame **data\_per\_type**. The result is a pandas Series containing the values in the 'articleType' column.
2. **.unique():**
   * The **unique()** method is then called on the selected Series. This method returns an array of unique values in the Series, preserving the order of their first occurrence.

So, the overall purpose of this code is to extract and display the unique values from the 'articleType' column in the **data\_per\_type** DataFrame. This can be useful for understanding the different categories or types of articles that were present in the original dataset, aggregated and sorted by their counts. It's a common step in data analysis to explore the unique values in categorical columns to gain insights into the variety of data within that particular feature.





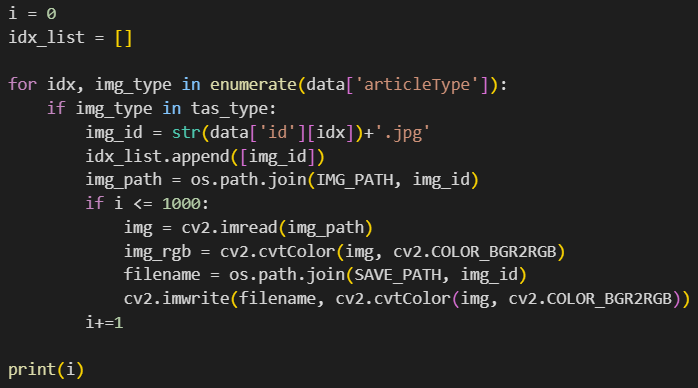
1. **Filtering using Conditions:**
   * The code uses boolean conditions to filter rows from the **data\_per\_type** DataFrame. Each condition is enclosed in parentheses and separated by the **|** (OR) operator. The conditions check if the 'articleType' is equal to one of the specified bag types.
2. **Creating a New DataFrame:**
   * The entire expression is used to index the original **data\_per\_type** DataFrame, resulting in a new DataFrame (**data\_tas**) containing only the rows where the 'articleType' matches one of the specified bag types.

So, **data\_tas** will contain information specifically related to handbags, laptop bags, duffel bags, messenger bags, trolley bags, waist pouches, and backpacks. This kind of filtering is common when you want to focus on a subset of the data that is relevant to a particular analysis or task.



This simple line of code initializes a Python list with two elements: 'Handbags' and 'Backpacks'. Each element in the list is a string. The list can be used to store multiple values in a single variable.

This type of list is often used when you want to specify or work with a predefined set of values, such as different types of items (in this case, types of bags). The list can then be used in various operations, such as filtering data based on these specific types or iterating over the elements for some computation.



1. **Initialization:**
   * **i = 0**: Initializes a counter variable **i** to zero. This variable is used to keep track of the number of processed images.
   * **idx\_list = []**: Initializes an empty list **idx\_list**. This list will be used to store image IDs.
2. **Loop through DataFrame:**
   * **for idx, img\_type in enumerate(data['articleType']):**: This loop iterates through the rows of the DataFrame **data**. The **enumerate** function is used to get both the index (**idx**) and the value (**img\_type**) from the 'articleType' column.
3. **Check Image Type:**
   * **if img\_type in tas\_type:**: Checks if the 'articleType' is in the **tas\_type** list. If true, it means the image type is one of the specified types (e.g., 'Handbags' or 'Backpacks').
4. **Image ID and List Creation:**
   * **img\_id = str(data['id'][idx])+'.jpg'**: Creates an image ID by converting the 'id' value from the DataFrame to a string and appending '.jpg'.
   * **idx\_list.append([img\_id])**: Appends the image ID to the **idx\_list** as a list.
5. **Image Processing and Saving:**
   * **img\_path = os.path.join(IMG\_PATH, img\_id)**: Constructs the full path to the image using **IMG\_PATH** and the image ID.
   * **if i <= 1000:**: Checks if the number of processed images (**i**) is less than or equal to 1000.
     + **img = cv2.imread(img\_path)**: Reads the image using OpenCV.
     + **img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)**: Converts the image from BGR to RGB.
     + **filename = os.path.join(SAVE\_PATH, img\_id)**: Constructs the full path to save the image using **SAVE\_PATH**.
     + **cv2.imwrite(filename, cv2.cvtColor(img, cv2.COLOR\_BGR2RGB))**: Writes the RGB image to the specified file path.
     + **i+=1**: Increments the counter.
6. **Print Total Count:**
   * **print(i)**: Prints the total count of processed images.

In summary, this code filters and processes images from a DataFrame based on their 'articleType'. It saves a subset of these images (up to 1000) to a new directory specified by **SAVE\_PATH**. The list **idx\_list** is created to store the image IDs that match the specified types. The final count of processed images is printed.

We researched which model we should use and in which ways the models would be more useful to us.

**MobileNetV2:**

* MobileNetV2 is a lightweight neural network architecture designed for mobile and edge devices. It is efficient and well-suited for image-related tasks.
* MobileNetV2 is used as a feature extractor. The **weights='imagenet'** argument indicates that the model is initialized with pre-trained weights on the ImageNet dataset. This pre-training helps the model learn useful features from a large and diverse dataset.
* Pros: MobileNetV2 is lightweight and fast, making it suitable for applications with limited computational resources. It has been designed to balance between accuracy and efficiency.
* Cons: It might not capture as complex features as larger models, which could be important if your fashion model fit task requires fine-grained details.

**Sequential Model:**

* The **Sequential** model is a linear stack of layers. In this case, layers are added sequentially to build the complete model.
* The first layer is the MobileNetV2 model, which is added to the sequential model.

**Global Average Pooling 2D Layer:**

* After the MobileNetV2 base model, a Global Average Pooling 2D layer is added.
* Global Average Pooling reduces the spatial dimensions of the feature map to a single value per channel. This helps in reducing the number of parameters in the model and provides a global summary of the features.

**Dense Layer with Sigmoid Activation:**

* The Global Average Pooling layer is followed by a Dense layer with a single neuron and a sigmoid activation function.
* The sigmoid activation is used for binary classification tasks. It squashes the output between 0 and 1, making it suitable for binary decisions.

**Compilation:**

* The model is compiled using the Adam optimizer and binary crossentropy loss. Adam is a popular optimization algorithm, and binary crossentropy is commonly used for binary classification problems.
* The chosen metric for monitoring during training is accuracy.

We import the necessary libraries for our models. Since we explained above what we will use in which model, we will only show importing in this section.

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu