**df = pd.read\_csv**

This line reads a CSV file containing sales data into a Pandas DataFrame df.

**df = df[(df != 0).all(axis=1)]:** This line removes rows from the DataFrame df where any column contains a zero value

If any value in a row is zero, that row is excluded from df.

**df.groupby(['product\_id', 'sales\_week of year'])**: This part groups the DataFrame df by two columns: product\_id and sales\_week of year. It means that the data will be organized into groups where each unique combination of product\_id and sales\_week of year forms a group.

**['sales\_product\_quantity'].mean()**: Within each group, it selects the sales\_product\_quantity column and calculates the mean (average) value of this column. So, for each group defined by product\_id and sales\_week of year, it computes the average sales quantity.

**.reset\_index():** After computing the mean, this method resets the index of the resulting DataFrame. This is typically done to convert the group labels (product\_id and sales\_week of year) from indices back into columns.

**avg\_sales['sales\_product\_quantity']**: This selects the column sales\_product\_quantity from the avg\_sales DataFrame.

**.astype(float):** This method converts the data type of the selected column to float.

**avg\_sales.columns**: This property accesses the column names of the avg\_sales DataFrame.

**print(avg\_sales.head())** : .head() is a method that returns the first 5 rows. When you run print(avg\_sales.head()), it will print the first 5 rows of the avg\_sales.

**avg\_sales['product\_id'] = avg\_sales['product\_id'] / avg\_sales['product\_id'].max()** : This calculates the maximum value in the product\_id column and divides each value in the product\_id column by the maximum value found in that column. The result is that each value in the product\_id column is transformed to a value between 0 and 1.

**avg\_sales['sales\_week of year'] = avg\_sales['sales\_week of year'] / avg\_sales['sales\_week of year'].max()** : This calculates the maximum value in the sales\_week of year column and divides each value in the sales\_week of year column by the maximum value found in that column.

**avg\_sales[['product\_id', 'sales\_week of year']].values** : This selects two columns, product\_id and sales\_week of year. .values converts the selected columns into a NumPy array.

**avg\_sales['avg\_sales\_product\_quantity'].values** : This selects a column product quantity in avg\_sales Dataframe. .values converts the selected columns into a NumPy array.

**X = X.reshape((X.shape[0], X.shape[1], 1**)) : changes the array X from a 2D shape of (rows, columns) to a 3D shape of (rows, columns, 1), adding a third dimension of size 1.

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42**) : This line splits the data X and labels y into training and testing sets, with 20% of the data used for testing and a fixed random state for reproducibility.

**model.add(Conv1D(filters=64, kernel\_size=2, activation='relu', input\_shape=(X\_train.shape[1], X\_train.shape[2])))** : This line adds a 1D convolutional layer to the model with 64 filters, a kernel size of 2, ReLU activation, and an input shape matching the training data's features.

**model.add(Dense(50, activation='relu'))** : This line adds a fully connected (dense) layer to the model with 50 neurons and ReLU activation.

**model.add(Dense(1))** : This line adds a fully connected (dense) layer with a single neuron to the model, typically used for the output layer in regression or binary classification tasks.

**model.compile(optimizer=Adam(learning\_rate=0.001), loss='mse', metrics=['mae'])** :

This line configures the model to use the Adam optimizer with a learning rate of 0.001, mean squared error (MSE) as the loss function, and mean absolute error (MAE) as a metric to evaluate performance.

**model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_test, y\_test)) :** This line trains the model on the training data X\_train and y\_train for 50 epochs with a batch size of 32, and evaluates performance on the validation data X\_test and y\_test during training.

**loss, mae = model.evaluate(X\_test, y\_test)** : This line evaluates the model on the test data X\_test and y\_test, returning the loss and mean absolute error (MAE) values.

**print(f'Mean Absolute Error: {mae}')** : This line prints the mean absolute error (MAE) value with a descriptive message.

**model.save('cnn\_sales\_model.h5')** : This line saves the trained model to a file named 'cnn\_sales\_model.h5' in a format that can be later loaded and used for predictions without needing to retrain.

**model = load\_model('cnn\_sales\_model.h5') :** This line loads a pre-trained model stored in the file 'cnn\_sales\_model.h5' into memory, allowing it to be used for making predictions without needing to train it again.

**def model\_predict(product\_id, week, model):** This function model\_predict is designed to predict something related to a product identified by product\_id during a specific week using a machine learning model.

**max\_product\_id = avg\_sales['product\_id'].max() :** This line finds the maximum value of the 'product\_id' column in the dataframe avg\_sales, representing the highest product identifier present in the dataset.

**max\_week = avg\_sales['sales\_week of year'].max() :**

This line finds the highest value in the 'sales\_week of year' column of the avg\_sales dataframe, representing the latest week of sales data in the dataset.

**normalized\_product\_id = product\_id / max\_product\_id :** This line calculates the normalized value of product\_id by dividing it by the maximum product identifier max\_product\_id, scaling product\_id to a range between 0 and 1 based on the highest product identifier present in the dataset.

**normalized\_week = week / max\_week :** This line calculates the normalized value of week by dividing it by the maximum week value max\_week, scaling week to a range between 0 and 1 based on the latest week of sales data in the dataset.

**input\_data = np.array([[normalized\_product\_id, normalized\_week]]) :** This line creates a numpy array input\_data containing normalized values of product\_id and week, ready to be used as input for a machine learning model that expects these features scaled between 0 and 1.

**input\_data = input\_data.reshape((input\_data.shape[0], input\_data.shape[1], 1))** : This line reshapes the numpy array input\_data from a 2D array with shape (1, 2) to a 3D array with shape (1, 2, 1), adding a third dimension of size 1. This is often required for certain types of data formats expected by machine learning models, such as Convolutional Neural Networks (CNNs) in some cases.

**prediction = model.predict(input\_data)** : This line uses the machine learning `model` to predict an outcome based on the `input\_data`, which typically includes normalized values of `product\_id` and `week`.

**print(f"Prediction for product\_id: {product\_id}, week: {week}: {prediction[0][0]}")** : This line prints the predicted outcome from the machine learning model for a specific `product\_id` and `week`, displaying the predicted value.

**def forecast\_sales\_for\_product(product\_id, start\_week, product\_price, model**): This function `forecast\_sales\_for\_product` predicts sales for a specific `product\_id` starting from a given `start\_week`, considering `product\_price`, using a machine learning `model`.

**for week in range(start\_week, start\_week + 52):** This loop iterates over a range of weeks starting from `start\_week` up to `start\_week + 52`, encompassing a full year (52 weeks), executing code for each week in this range.

**normalized\_week = (week - 1) % 52 + 1** : This line calculates a normalized week value within a yearly cycle, adjusting `week` to range from 1 to 52 by using modulo arithmetic, ensuring consistent representation of weeks throughout the year.

**prediction = model\_predict(product\_id, normalized\_week, model)** : This line predicts an outcome using a machine learning `model` for a specific `product\_id` and `normalized\_week`, which represents the week adjusted to a standard range within a yearly cycle.

**predictions.append(prediction)** : This line appends a predicted outcome (`prediction`) to a list (`predictions`), storing the result for later use or analysis.

**forecast\_df = pd.DataFrame({** : This line creates a new pandas DataFrame (`forecast\_df`) using a dictionary where keys are column names, allowing structured storage and manipulation of data in tabular form.

**'product\_id': [product\_id] \* 52**, : This code creates a list containing 52 copies of the value `product\_id`, which is then used as the data for the 'product\_id' column in a DataFrame or similar structure.

**'sales\_week of year': np.arange(start\_week, start\_week + 52) % 52 + 1, # Week numbers 1 to 52** : This code generates a sequence of week numbers (`sales\_week of year`) ranging from `start\_week` to `start\_week + 52`, ensuring the numbers wrap around within a yearly cycle using modulo arithmetic to produce values from 1 to 52.

**'product\_price': [product\_price] \* 52** : This code creates a list containing 52 copies of the value `product\_price`, which will be used as the data for the 'product\_price' column in a DataFrame or similar data structure.

**print(f"Forecast for product\_id: {product\_id} completed.")** : This line prints a message indicating that the forecast process for a specific `product\_id` has been successfully completed.

**return forecast\_df :** This line returns the DataFrame `forecast\_df`, which contains the forecasted sales data for a product over a period of 52 weeks, allowing the forecast to be used or analyzed further in the program. ..