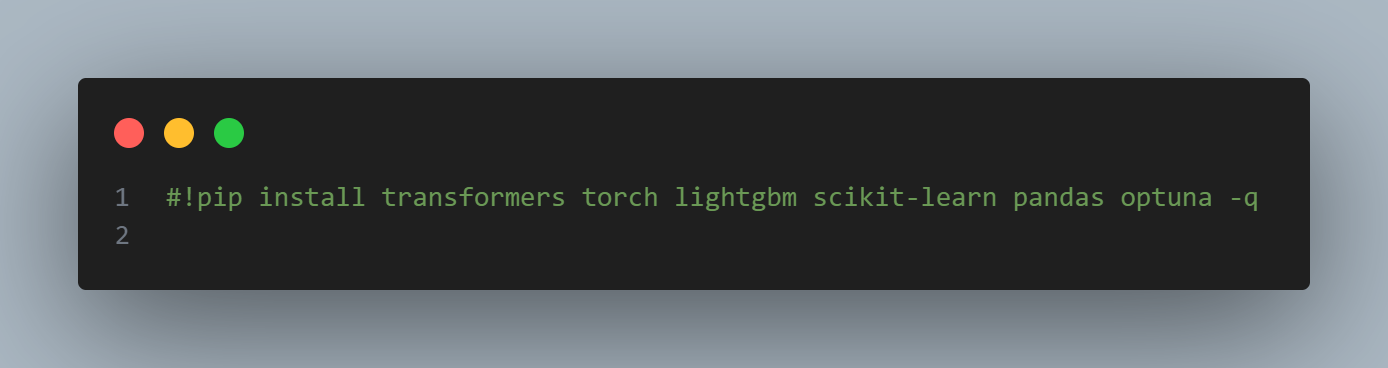
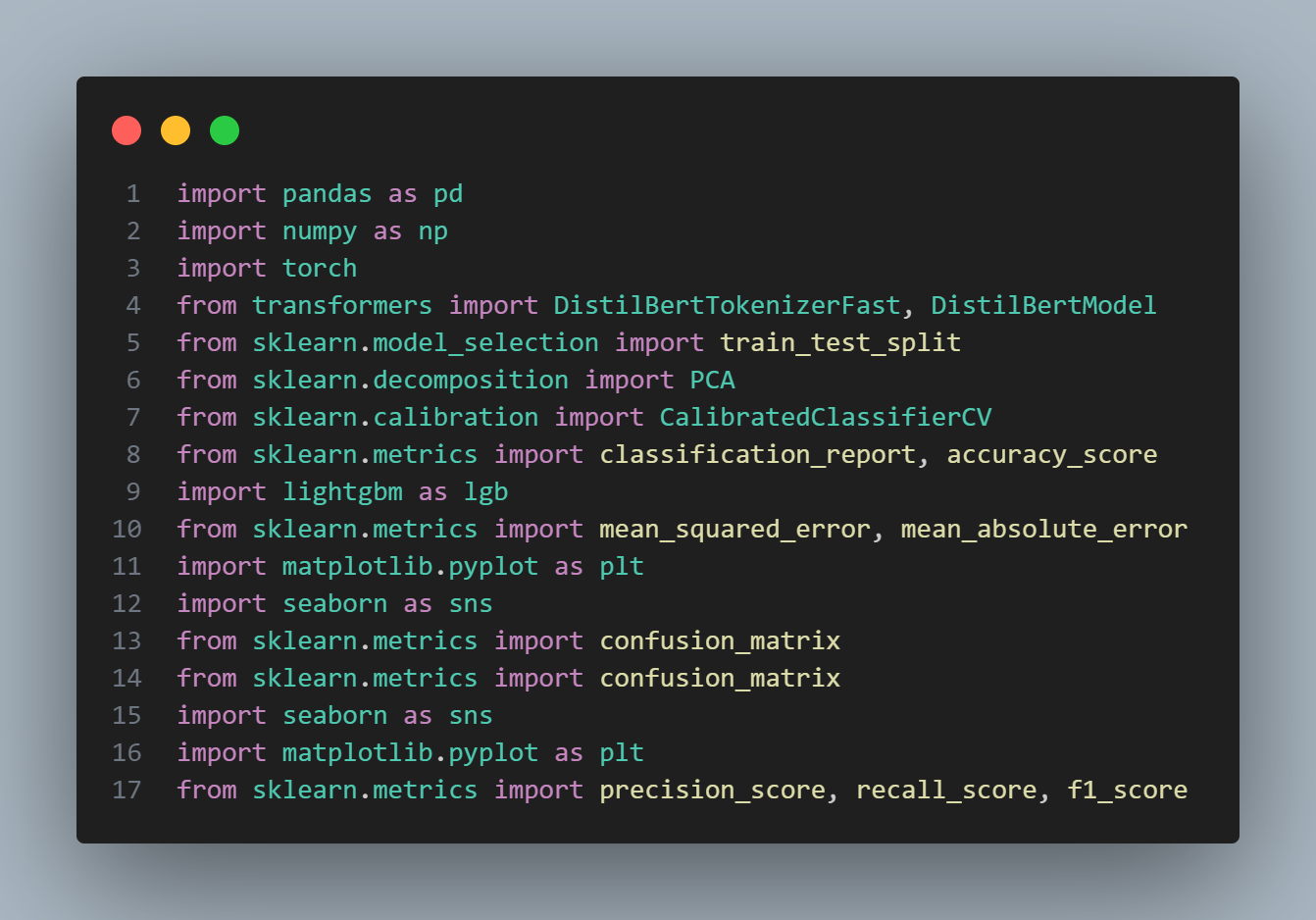
**Documentation for Hybrid BERT + LightGBM**



This is a **Jupyter Notebook** or **Google Colab** style command used to install Python packages.

* #!pip install ...:  
  The #! at the beginning is a **comment** in Python, which means this line **won’t execute** unless the # is removed.
* pip install:  
  This is the **Python package installer**, used to install libraries from the Python Package Index (PyPI).
* **Packages listed**:
  + transformers: Hugging Face library for state-of-the-art NLP models like BERT, GPT, etc.
  + torch: PyTorch — a deep learning framework.
  + lightgbm: A fast, efficient gradient boosting framework by Microsoft.
  + scikit-learn: For traditional machine learning models and utilities.
  + pandas: For data manipulation and analysis (especially with DataFrames).
  + optuna: An automatic hyperparameter optimization framework.
* -q: This stands for **quiet mode**, which suppresses the output during installation to make logs cleaner.

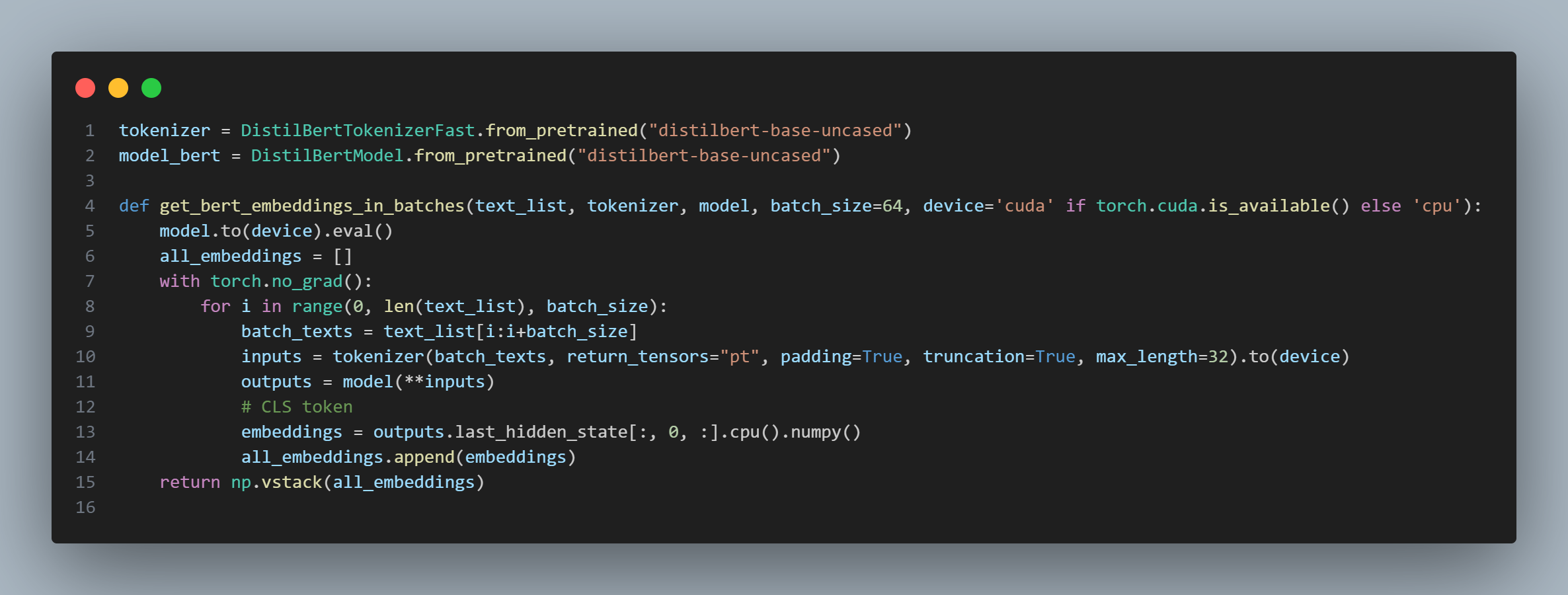


This code imports essential libraries for a hybrid machine learning workflow combining NLP and traditional ML. It includes pandas and numpy for data manipulation, torch and Hugging Face’s transformers for extracting embeddings using DistilBERT, and scikit-learn for splitting data, dimensionality reduction (PCA), model calibration, and evaluating metrics like precision, recall, F1, and accuracy. It also brings in lightgbm for building efficient gradient boosting models, and matplotlib/seaborn for data visualization, especially for plotting confusion matrices. This setup is ideal for tasks like text classification using deep learning features combined with LightGBM.

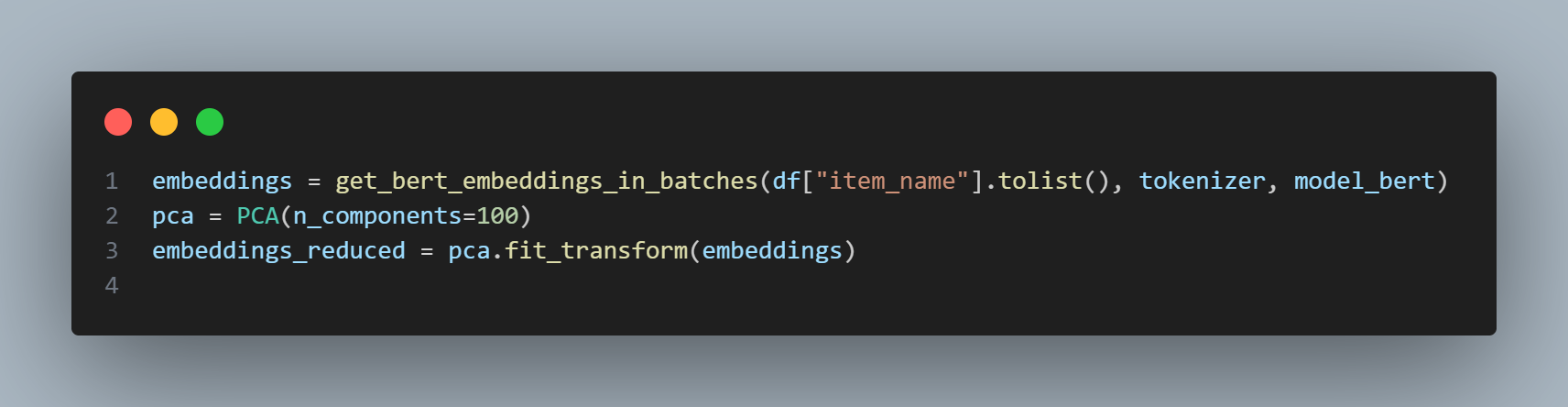


This code processes a synthetic retail dataset to prepare it for modeling **future sale predictions**:

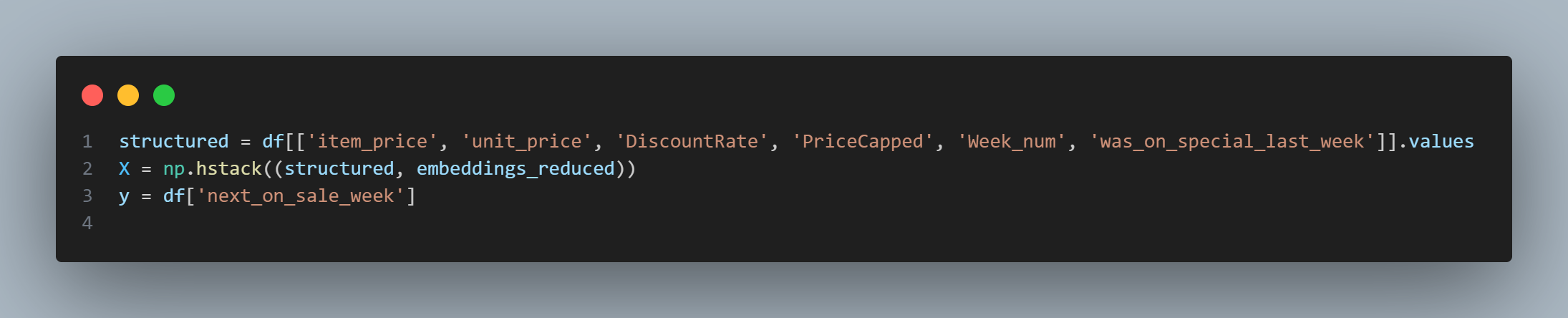
1. **Data Loading & Preprocessing**:
   * It reads Coles\_synthetic\_8weeks\_v3\_cleaned.csv.
   * Extracts week numbers from a column named Week using regex and sorts the dataset by product\_code and week\_num.
2. **Feature Engineering**:
   * Adds a new binary feature was\_on\_special\_last\_week, indicating whether each product was on special in the **previous week**, using groupby().shift().
3. **Target Generation – next\_on\_sale\_week**:
   * A function find\_next\_sale\_weeks loops through each product’s weekly records to find the **next week when that product will be on special**.
   * If a future sale is found, it adds that week number; otherwise, it appends None.
4. **Apply Function**:
   * Applies this function across each product\_code group to get the next\_on\_sale\_week column.
5. **Clean-up**:
   * Removes rows where the next sale week is missing.
   * Converts next\_on\_sale\_week to integer type.



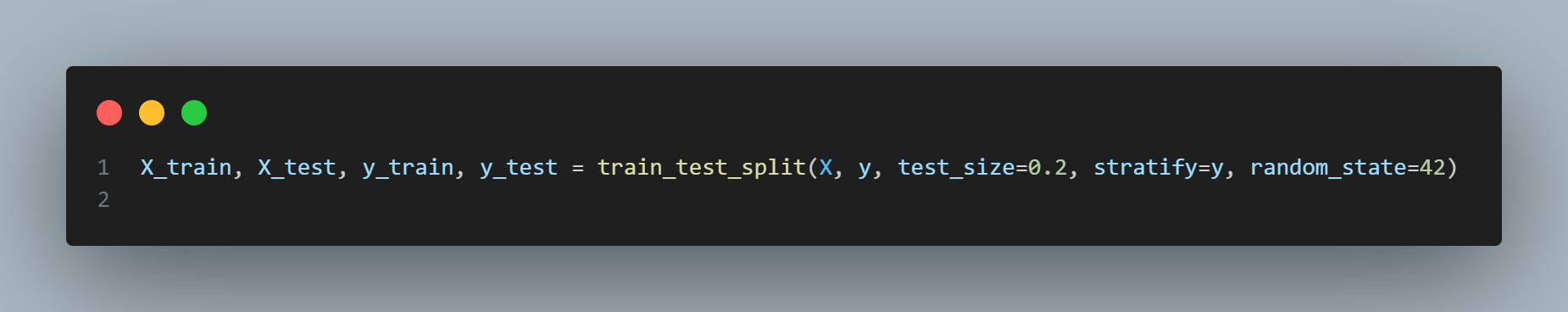
This code defines a function to generate BERT embeddings in batches using the pretrained DistilBERT model. It tokenizes a list of text inputs, runs them through the model without computing gradients (inference mode), and extracts the [CLS] token embedding for each input. The embeddings are then stacked into a single NumPy array. This batch processing is optimized for GPU if available, making it suitable for efficiently encoding large text datasets.



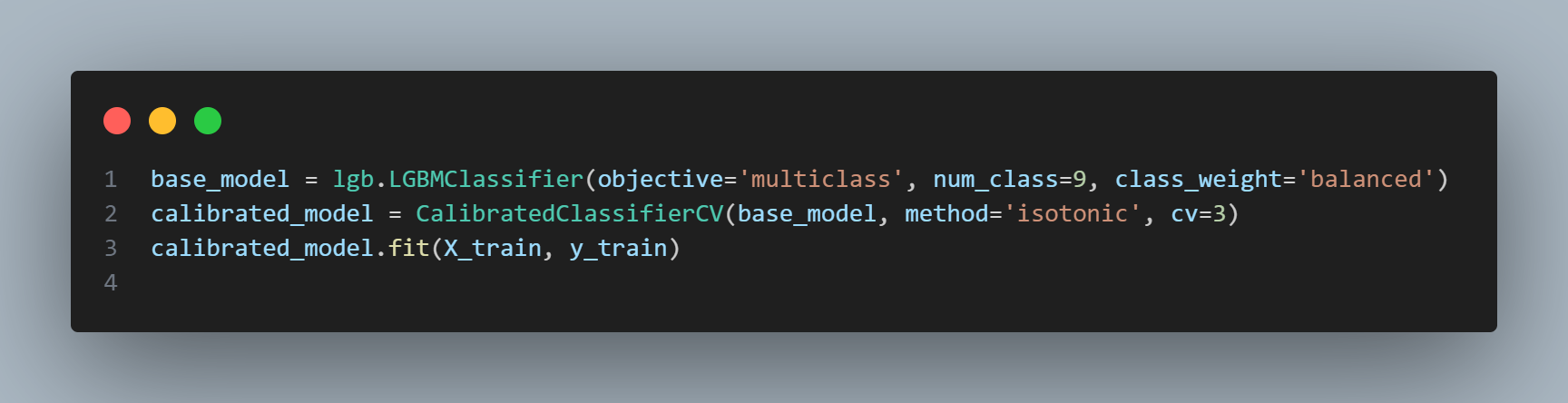
This code generates BERT embeddings for all item names in the dataset and reduces their dimensionality using PCA. It first calls get\_bert\_embeddings\_in\_batches to convert the item names into high-dimensional vectors, then applies PCA to reduce these to 100 dimensions. This results in compact, information-rich embeddings suitable for combining with structured features in a machine learning model.



This code prepares the input features (X) and target variable (y) for model training. It combines structured numeric data like price, discount rate, and week number with reduced BERT embeddings of item names to form a comprehensive feature matrix. The target variable y represents the week when each item will next go on sale. This hybrid feature setup enables the model to leverage both tabular and textual information for accurate sale prediction.



This code splits the full dataset into training and testing sets using an 80-20 ratio. The stratify=y argument ensures that the distribution of the target variable (next\_on\_sale\_week) is preserved in both sets, which is important for balanced learning. The random\_state=42 ensures reproducibility of the split.



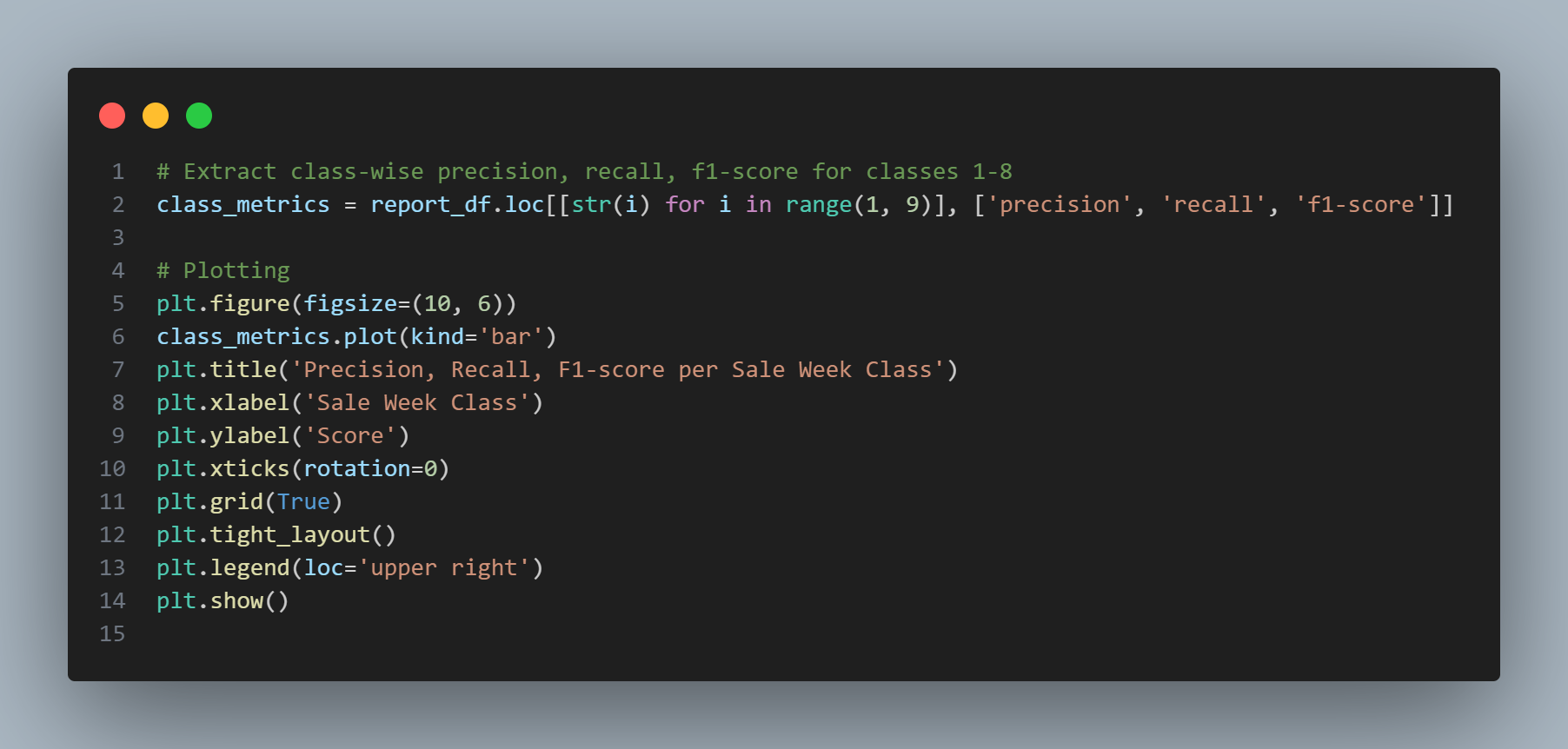
This code defines and trains a multiclass LightGBM model with calibrated probabilities. The LGBMClassifier is configured for 9 classes with class balancing to handle class imbalance. It is then wrapped in CalibratedClassifierCV using isotonic regression and 3-fold cross-validation to improve the accuracy of predicted probabilities. Finally, the calibrated model is trained on the training data.



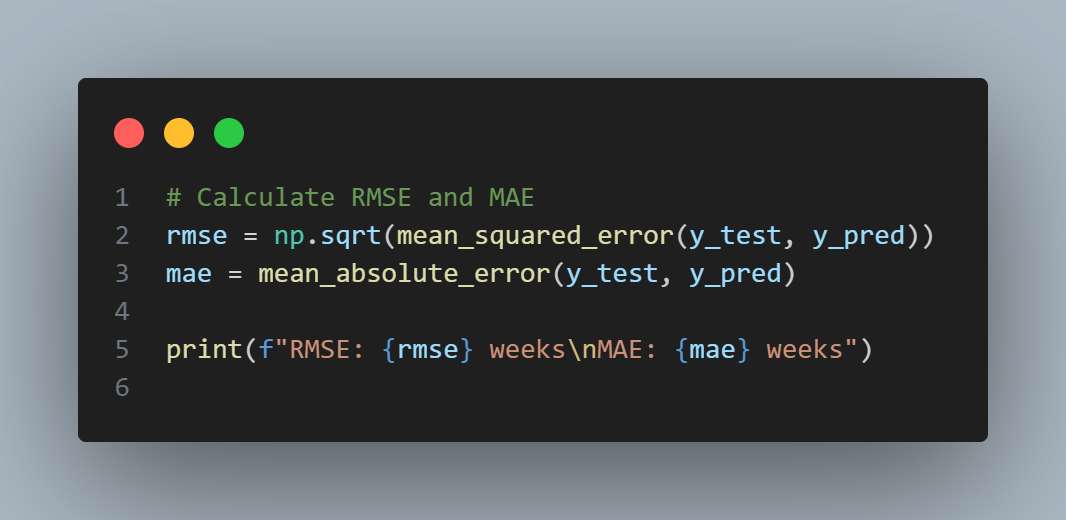
This code evaluates the trained model’s performance on the test set. It starts by predicting y\_pred and printing the overall accuracy and classification report. It then calculates precision, recall, and F1-score using **macro**, **micro**, and **weighted** averaging strategies to assess model performance across multiple classes. Macro gives equal weight to each class, micro aggregates across all instances, and weighted adjusts for class imbalance — offering a comprehensive view of how well the model predicts the next sale week.



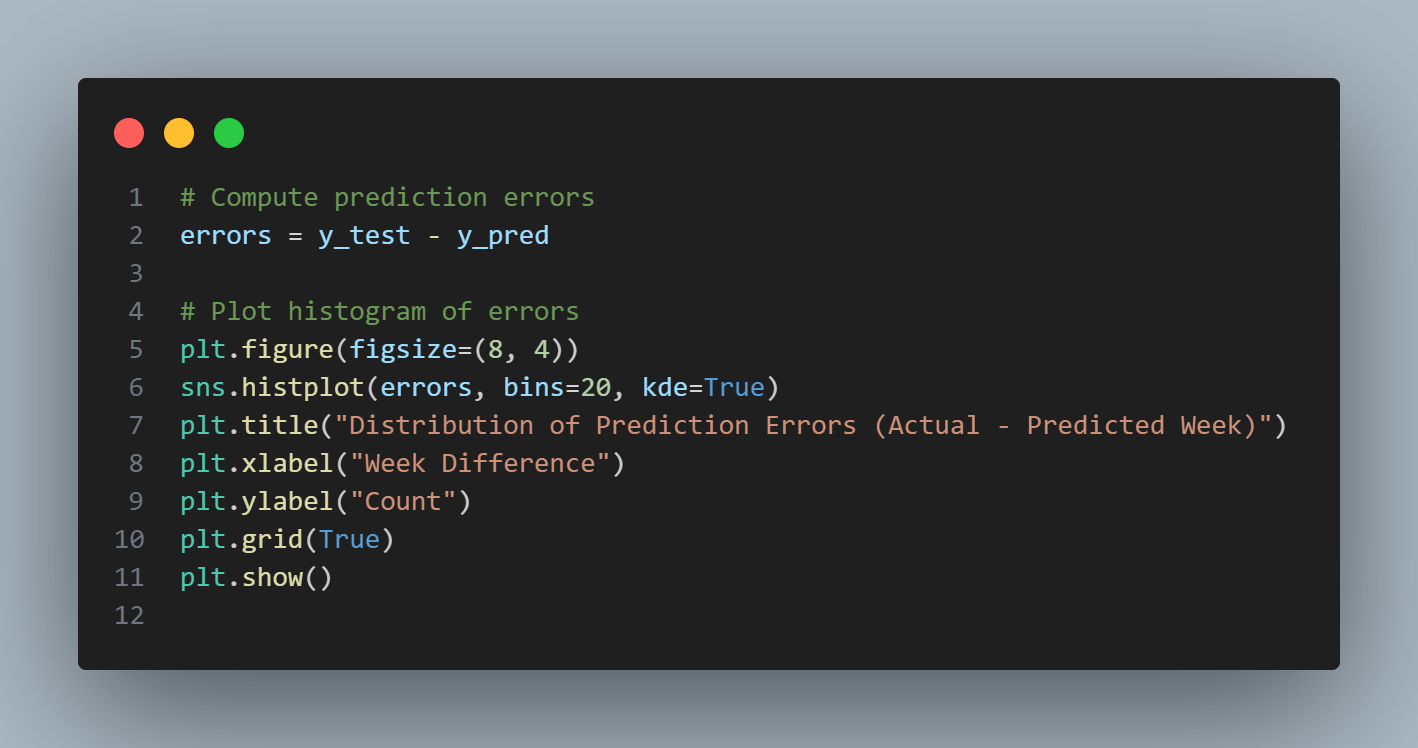
This code converts the classification report into a structured pandas DataFrame for easier analysis and presentation. It first generates a dictionary of classification metrics, converts it into a DataFrame, and transposes it so that each class has its own row. The metrics are then rounded to three decimal places and displayed, making the model evaluation results cleaner and more interpretable.



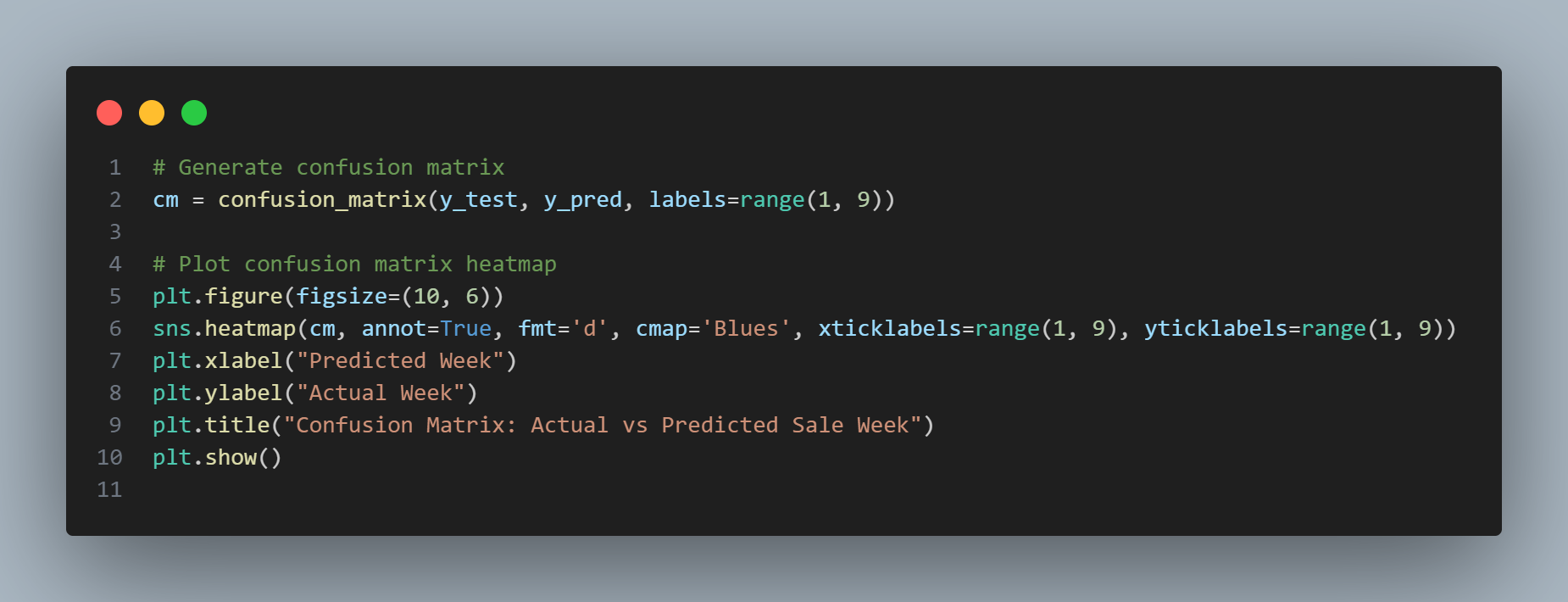
This code extracts precision, recall, and F1-score for each sale week class (1 to 8) from the classification report DataFrame and visualizes them using a grouped bar chart. The plot helps in quickly assessing model performance across different target classes, highlighting which sale weeks are predicted more accurately and where the model may be underperforming.



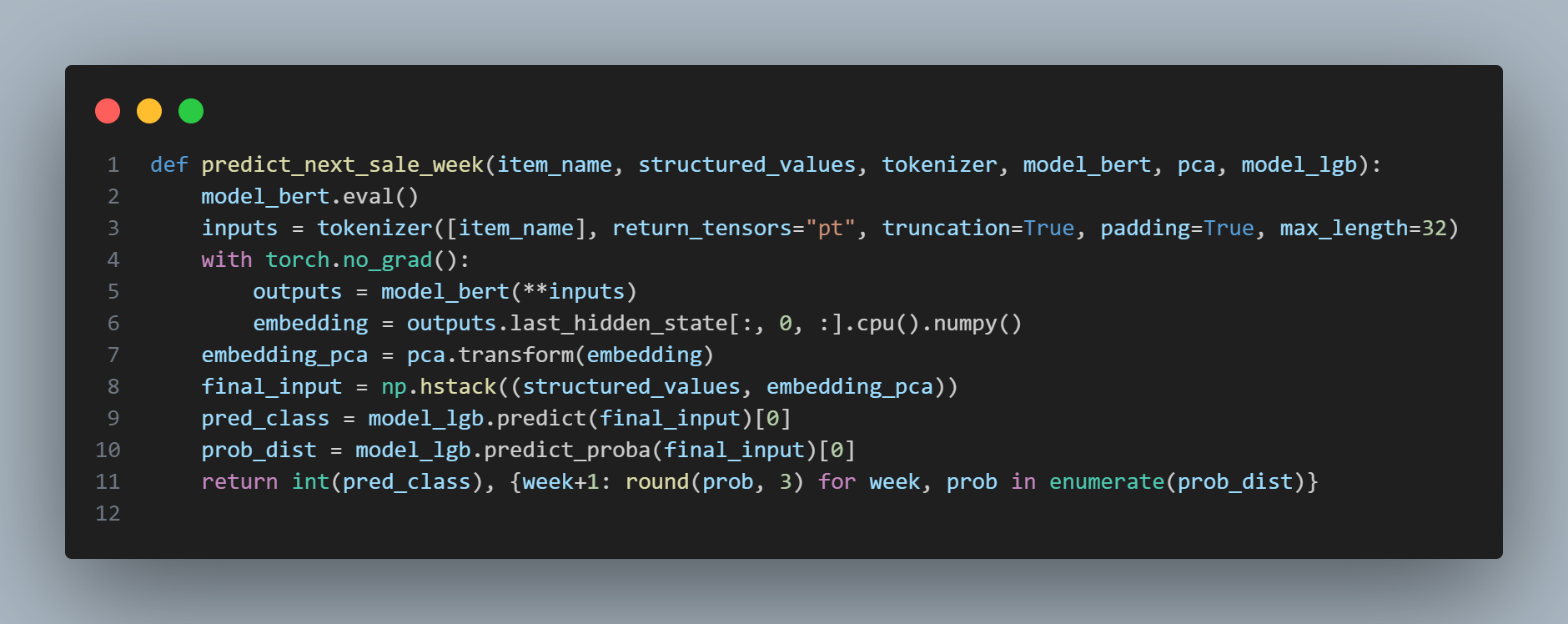
This code calculates two regression metrics—Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE)—to evaluate how close the model’s predicted sale weeks are to the actual ones. RMSE penalizes larger errors more heavily, while MAE provides an average absolute deviation. Both are printed in units of weeks, offering a clear interpretation of prediction accuracy in the context of sale timing.



This code calculates the prediction error for each sample by subtracting predicted sale weeks from the actual values. It then visualizes the distribution of these errors using a histogram with a KDE curve. The plot helps identify how often the model over- or under-predicts the week an item goes on sale and whether the errors are centered around zero, indicating good predictive performance.



This code generates and visualizes a confusion matrix comparing actual versus predicted sale weeks. It uses confusion\_matrix to quantify correct and incorrect predictions per class, then plots the matrix as a heatmap with labeled axes. This visual helps identify which weeks are frequently confused by the model, revealing strengths and weaknesses in its classification performance.



This function predicts the next sale week for a given item using the trained BERT-LightGBM hybrid model. It first tokenizes the item\_name, extracts BERT embeddings, reduces their dimensionality using PCA, and combines them with structured numerical features. The final feature vector is passed to the LightGBM model to predict the most likely sale week (pred\_class) and return both the predicted week and the class-wise probability distribution rounded to 3 decimal places.



This code demonstrates how to use the trained BERT-LightGBM pipeline to predict the next sale week for a specific product. It inputs the item name and its structured features (e.g., price, discount, week info), passes them through the predict\_next\_sale\_week function, and prints both the predicted week and the probability distribution across all week classes. This allows for both a decision and confidence insight for marketing or stocking decisions.