Illustration for each file:

data by weeks.ipynb & data by weeks.py & data_by_weeks_collab_nextmontuepred.ipynb:

- Data Acquisition: Initially, the code uses the yfinance library to obtain historical price data for a set of stocks. This data includes the opening price, highest price, lowest price, closing price, and trading volume.

- Technical Indicator Calculation: Next, the code defines a function `calculate_ma` to calculate the moving averages (MA) for the stock data and uses the talib library to calculate the MACD indicators, including the MACD line, signal line, and MACD histogram.

- Data Visualization: Then, the code uses the mplfinance library to plot the stock price data and the calculated technical indicators into charts. These charts include candlestick charts, trading volume, moving averages, and MACD indicators.

- Image Processing: The generated charts are saved as PNG files and are cropped using the PIL library to remove unnecessary parts, enhancing the clarity and readability of the images.

- Label Generation: Finally, the code extracts the stock code, year, and week from the cropped image filenames and generates labels based on the stock price changes. These labels include whether the stock has risen (Rise) and the return rate of the stock (Return), which are then saved to a CSV file to provide data support for subsequent data analysis or machine learning model training.

- Differences among the three files: `data by weeks.ipynb` and `data by weeks.py` use the return rate of the following week as the label for this week, while `data_by_weeks_collab_nextmontuepred.ipynb` uses the return rate between the next Monday and Tuesday as the label for this week (to adjust the prediction time length to observe changes in prediction effects).

EDA.ipynb:

- Reading CSV File: Reads a CSV file named `labels.csv` from a specified path and stores it in the `dataframes` variable.

- Split DataFrame: Splits the original DataFrame based on the stock code (Stock Code) into multiple DataFrames, each corresponding to a unique stock code.

- Sorting: Sorts each split DataFrame by year (Year) and week (Week).

- Output DataFrame Head: Outputs the first few rows of each split DataFrame for easy viewing.

- Statistical Analysis: Calculates a series of statistical data for each split DataFrame, including Mean, Standard Deviation (Sd), Skewness, Kurtosis, and Autocorrelation.

- Creating Statistical DataFrame: Summarizes all statistical data into a new DataFrame (`statistics_df`).

- Setting Seaborn Style: Sets the Seaborn plotting style for more aesthetically pleasing charts.

- Drawing Histograms: Uses Seaborn's `histplot` function to draw histograms of stock returns (Return) for each stock code, adding kernel density estimation (KDE).

- Drawing Time Series Charts: Calculates the number of consecutive weeks and plots the line chart of stock returns over time.

- Drawing Rolling Statistics Charts: Calculates rolling averages and standard deviations, and plots these rolling statistics over time.

- Drawing Count Charts: Based on whether the stock return has increased (Rise), draws count charts showing the count of positive and negative returns.

- Drawing Pie Charts: Based on the count of positive and negative returns, draws pie charts showing the proportion of positive and negative returns.

- Chart Layout and Display: In all plotting steps, the script uses Matplotlib's `subplots` to organize the layout of the charts and uses `tight_layout` to automatically adjust the subplot parameters, making the chart layout more compact. Finally, it uses the `show` function to display the charts.

Algorithm-20data-week-0-71acc.ipynb&Algorithm-20data-week-0-71acc-imageaug-resize.ipynb:

- Importing Necessary Libraries: The code begins by importing libraries needed for processing images, data manipulation, machine learning model construction, and training.

- Setting File Paths: Defines the paths for image data and CSV files.

- Data Loading and Preprocessing: Loads the CSV file into `data_df_original`, groups it by stock code, and sorts it by time. Optionally performs data shifting and missing value processing.

- Data Visualization: Uses Seaborn and matplotlib libraries to count and visualize labels in the data, such as the positive or negative returns of stocks.

- Image and Label Processing: Reads images from the image folder, resizes them, converts them to grayscale, and stores them with labels in Numpy arrays. Divides the data into training, validation, and test sets.

- Model Definition: Defines a convolutional neural network model `ConvNet` for image classification.

- Model Training and Evaluation: Defines training function `pretrain` and evaluation function `evaluate_batch`. Trains the model and evaluates the accuracy on the validation set after each epoch. Saves the weights of the best model.

- Loss Function Visualization: Plots the training and validation losses, as well as the early stopping checkpoint.

- Confusion Matrix and ROC Curve: Calculates the confusion matrix, accuracy, recall, precision, F1 score, and AUC value for the test set, and visualizes them.

- Model Saving and Loading: Saves the trained model weights and loads them when needed.

- Grad-CAM Visualization: Uses Grad-CAM technology to visualize the heatmap of model predictions, showing the areas the model focuses on.

- Continuing Model Training: Optionally continues training the model from the last interrupted point.

- Test Set Evaluation: Evaluates the model's performance on the test set.

- Output Results: Outputs performance metrics and the confusion matrix.

- Image Visualization: Combines the heatmap of model predictions with the original images and visualizes them.

- Differences between the two files: The `Algorithm-20data-week-0-71acc.ipynb` file does not perform data augmentation on the stock price and volume charts, while `Algorithm-20data-week-0-71acc-imageaug-resize.ipynb` performs data augmentation on the stock price and volume charts to increase the robustness of the model by making appropriate distortions.

algorithm-pretrainedweight-single+portfolio-stock-ret-sr-0711finalversion-Copy.ipynb:

- Creating Directory: Creates a directory in the Kaggle workspace to store cropped images.

- Image Cropping: Reads all images from the original image directory, crops each image, and saves the cropped images to a new directory.

- Installing yfinance Library: Uses the pip command to install the yfinance library for obtaining stock data.

- Generating Label Data: Extracts the stock code, year, and week from the cropped image filenames and generates label data based on stock price changes, saving it to a CSV file.

- Data Preparation: Reads label data, groups it by stock code, and sorts each group.

- Image and Label Processing: Reads image files, resizes them, converts them to grayscale format, and stores the image data with labels in Numpy arrays. Then divides the data into training, validation, and test sets.

- Model Definition: Defines a convolutional neural network model `ConvNet` for image classification.

- Model Training and Evaluation: Defines training function `pretrain` and evaluation function `evaluate_batch`. Trains the model and evaluates the accuracy on the validation set after each epoch. Saves the weights of the best model.

- Confusion Matrix and ROC Curve: Calculates the confusion matrix, accuracy, recall, precision, F1 score, and AUC value for the test set, and visualizes them.

- Model Saving and Loading: Saves the trained model weights and loads them when needed.

- Strategy Evaluation: Uses the model to predict each stock index image in the test set individually, calculates the strategy return rate for each stock index, plots the comparison chart of actual price return and strategy return, and calculates and outputs the annualized return rate and Sharpe ratio for each stock index using the strategy.

- Equal Weight Long and Flat Portfolio Strategy Evaluation: Predicts the test data for all stocks and calculates the cumulative return rate of the equal weight long and flat portfolio strategy. Plots the portfolio cumulative return chart. Calculates and outputs the annualized return rate and Sharpe ratio for the portfolio strategy.

- Equal Weight Long and Short Portfolio Strategy Evaluation: Predicts the test data for all stocks and calculates the cumulative return rate of the equal weight long and short portfolio strategy. Plots the portfolio cumulative return chart. Calculates and outputs the annualized return rate and Sharpe ratio for the portfolio strategy.

weight_lasttrained_0705.pth: The weight model derived from `Algorithm-20data-week-0-71acc-imageaug-resize.ipynb`, which can be used as input pre-trained weights in `algorithm-pretrainedweight-single+portfolio-stock-ret-sr-0711finalversion-Copy.ipynb` to evaluate strategy performance.

Instruction for code running and main pipeline of ultimate algorithm:

Each piece of code can be run independently. The recommended platforms for running are Kaggle or Google Colab. It should be noted that some codes, such as algorithm-20data-week-0-71acc.ipynb and Algorithm-20data-week-0-71acc-imageaug-resize.ipynb, require the output datasets good_data and labels.csv from data by weeks.py or data by weeks.ipynb or data_by_weeks_collab_nextmontuepred.ipynb as input datasets. The execution of

algorithm-pretrainedweight-single+portfolio-stock-ret-sr-0711finalversion-Copy.ipynb requires the output datasets good_data and labels.csv from algorithm-20data-week-0-71acc.ipynb or algorithm-20data-week-0-71acc-imageaug-resize.ipynb, as well as the trained weight weight_lasttrained_0705.pth. Additionally, you need to appropriately adjust the reference addresses and names of the datasets to suit your computer. The final algorithm main pipeline is shown in the figure, where the blocks represent code segments, and the arrows represent the output datasets or weights from the previous segment.