# Objectives

1. Implement the nested time series cross validation strategy for grouped forecasting.
   1. User should provide the dataset, time column and the number of folds to generate
   2. For the given dataset, we will use **"day"** as a single time unit. This means you can split the data at day level
2. Write you code in sci-kit learn format. Refer to the [KFold CV](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html) for inspiration. The class should work on pandas dataframes and a datetime column name.
3. Test your code with some samples.
4. Build a time series model on the dataset above and evaluate it using your cross validation method. Submit the notebook illustrating the model development.
5. Submit the code, test samples and any other recorded observations

**Expected Class Structure:**

Init

params:

k (int): number of folds

split

params:

data (pandas.DataFrame): the training dataset

date\_column (str): column name for the datatime column. If None, use index.

yields:

train (pandas.DataFrame): training set indices

validate (pandas.DataFrame): validation set indices

# Evaluation criteria

The submissions will be evaluated on the following criteria

* Implementation of the nested time series logic
* Sample test cases used to check the implementation
* Use of the cross validation technique for model evaluation
* Performance of the time-series model on the provided dataset
* Effective communication of results and findings in the notebook.

### Explanation

**custom\_kfold Function**:

* + **group**: The DataFrame containing the time series data for a specific hierarchical group (e.g., a specific city, brand).
  + **order**: The order parameters for the ARIMA model (p, d, q).
  + **num\_folds**: The number of folds for KFold cross-validation.
  + **shuffle**: Whether to shuffle the data before splitting.
  + **random\_state**: The random seed for reproducibility.

**KFold Cross-Validation**:

* + The KFold function from scikit-learn splits the data into training and test sets.
  + For each split, we train an ARIMA model on the training set and forecast the values for the test set.
  + The performance of the model is evaluated using the mean squared error (MSE).
  + The average MSE across all folds is returned as the evaluation score for the group.

### Customizing the Function

You can customize the custom\_kfold function according to your needs:

* **Model Parameters**: Change the model type or parameters if you are using a different time series model.
* **Evaluation Metric**: Replace mean\_squared\_error with another metric if needed.
* **Data Handling**: Adjust the way you handle and preprocess the data within the function.

### Key Points in the Code:

**Data Loading and Preparation**:

* 1. Load the dataset and parse the date column.
  2. Ensure price and quantity columns are numeric and handle missing values.
  3. Create a sales column as price \* quantity.

**Nested Time Series Analysis**:

* 1. Group the data by city.
  2. For each group, aggregate daily sales.
  3. Fit an Exponential Smoothing model and store the model and forecast.

**Cross-Validation**:

* 1. Implement time series cross-validation using TimeSeriesSplit.
  2. Calculate and store the mean squared error for each split.

**Performance Evaluation**:

* 1. Compute the MSE for the forecast against the true values for each city.

**Visualization**:

* 1. Plot the true values and forecasts for visual comparison.

This approach ensures that your data is correctly prepared, the models are appropriately nested, and the results are effectively communicated. Adjust the seasonal\_periods parameter based on your dataset’s characteristics (e.g., daily, weekly, monthly seasonality).

### Key Adjustments:

1. **Check City Existence**: Before accessing models['city1'], check if 'city1' is in the models dictionary.
2. **Debugging Loop**: Print an error message if there's an issue processing any city.
3. **Test Case Adjustment**: Ensure the city exists in the dataset before running the test case.
4. **Cross-Validation Adjustment**: Check if the city exists before running cross-validation.
5. **Performance Metrics Calculation**: Ensure the model exists for each city before calculating performance metrics.
6. **Plot Adjustment**: Check if the city and its model exist before plotting.