# **📦 Cell 1 — Import Libraries**

### **🔹 What it does**

This cell imports all the libraries you’ll use throughout the project:

* **Data handling**: pandas, numpy
* **Visualization**: matplotlib, seaborn
* **Statistics**: statsmodels
* **Machine learning**: scikit-learn (SVR, MLPRegressor, GridSearchCV, metrics, StandardScaler, TimeSeriesSplit)
* **Utilities**: pathlib/Path (file paths), logging, warnings, cpu\_count (parallelization).

### **🔹 Why it’s needed**

* A machine learning project usually requires:
  1. **Data I/O & manipulation** → pandas, numpy
  2. **Exploration/visualization** → seaborn, matplotlib
  3. **Statistical tools** → statsmodels
  4. **Model training** → SVR and MLP from sklearn
  5. **Model evaluation & tuning** → metrics, GridSearchCV, TimeSeriesSplit
  6. **Scaling** → StandardScaler for normalization
  7. **Practical coding support** → logging for clean output, warnings to hide clutter, pathlib for paths, cpu\_count for efficiency.

### **🔹 Underlying Concepts to Know**

**Basic:**

* Why we import libraries: to reuse existing packages instead of coding everything from scratch.
* Difference between pandas (tabular) and numpy (array-based).
* Why visualization is essential before modeling.

**Intermediate:**

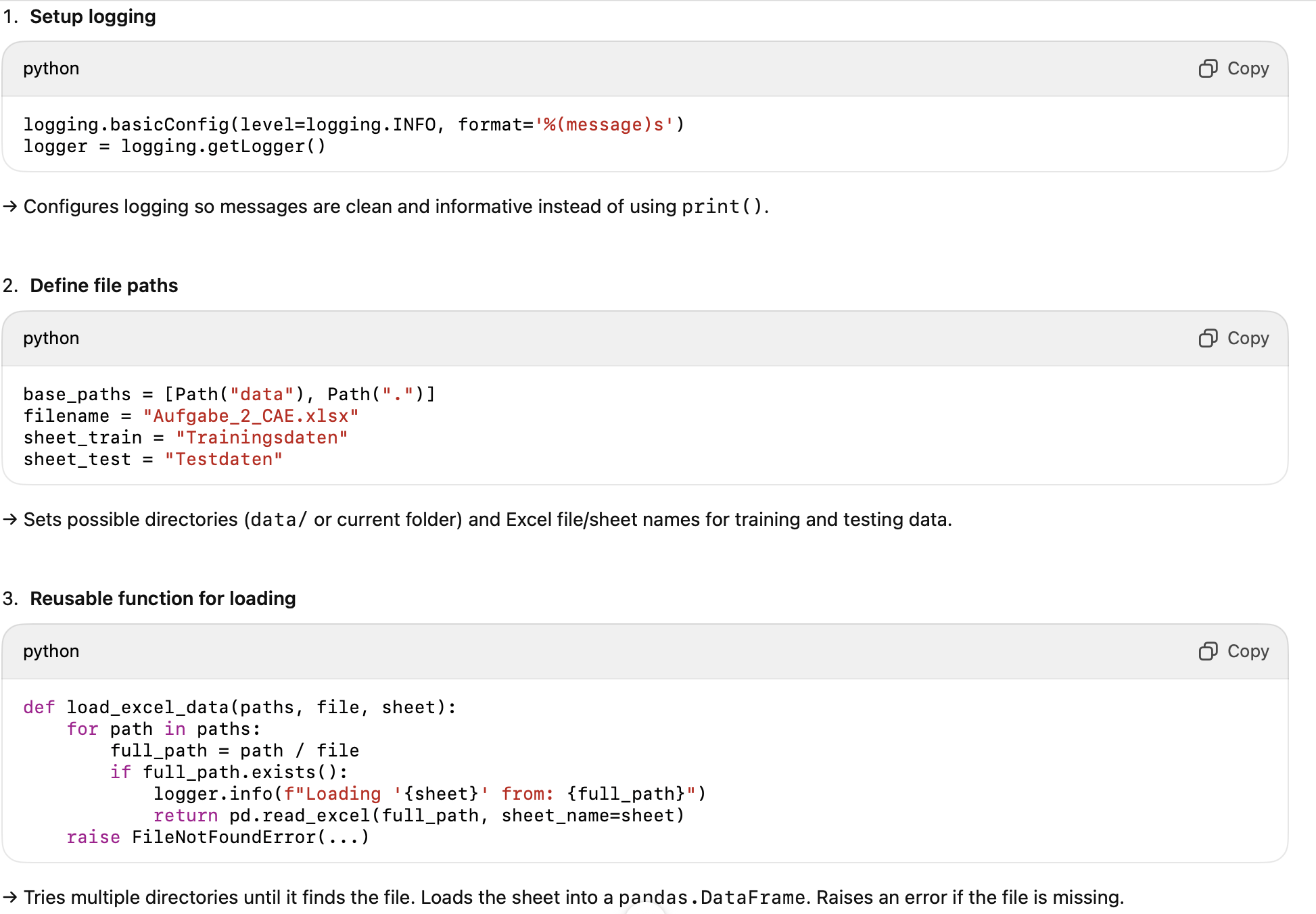
* Why scaling (StandardScaler) is important for ML algorithms like SVR/MLP.
* Why we need GridSearchCV for hyperparameter tuning.
* Why time series requires special splitting (TimeSeriesSplit).

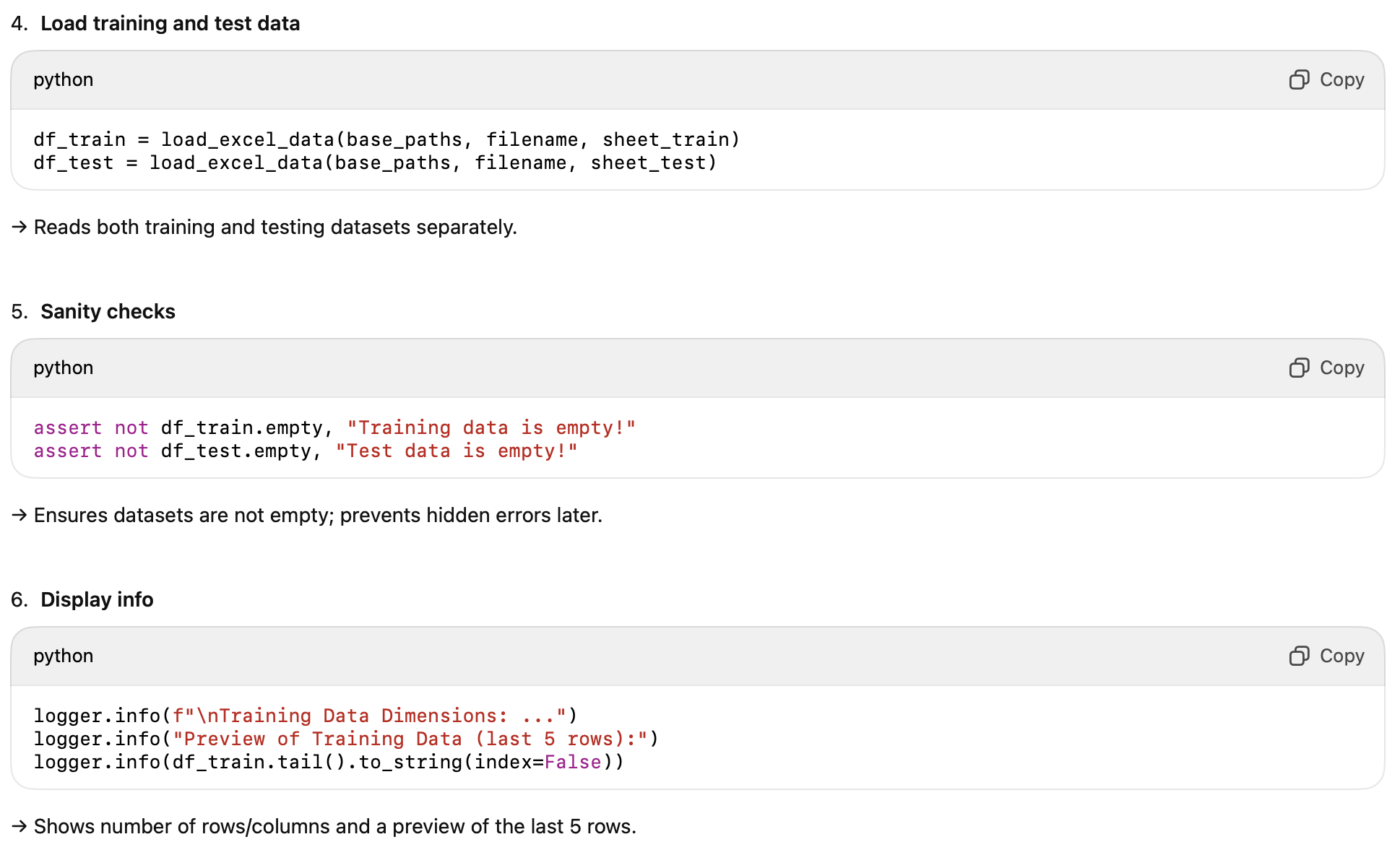
**Advanced:**

* Concept of parallelization (cpu\_count) to speed up grid search.
* Difference between statistics-based modeling (statsmodels) vs machine learning (sklearn).
* Why logging is preferred over print statements in professional projects.

# **📑 Cell 2 — Data Loading & Logging**

### **🔹 What it does**





### **🔹 Why it’s needed**

* **Logging** → professional, flexible output (can be turned on/off or filtered by severity).
* **Reusable function** → avoids code duplication, improves readability.
* **Multiple path search** → makes code portable (works whether file is in data/ or working dir).
* **Sanity checks** → catch missing/empty datasets early.
* **Info display** → quick validation that the data was loaded correctly and looks reasonable.

### **🔹 Underlying Concepts to Know**

**Basic:**

* What is a DataFrame (pandas).
* Why we preview datasets before modeling.
* Purpose of .shape and .tail().

**Intermediate:**

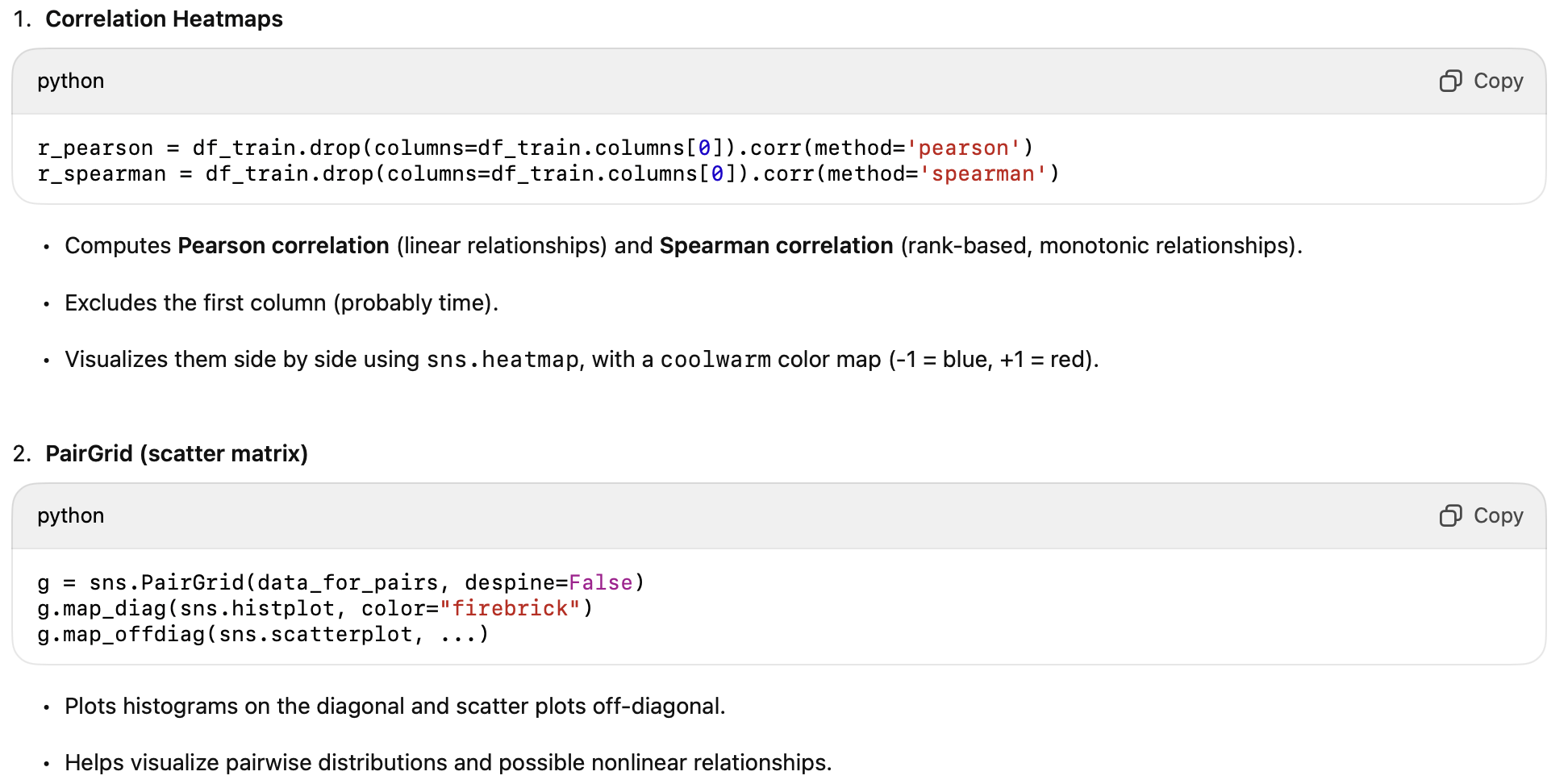
* Why training and testing data are separated.
* Difference between using assert vs if.
* Why logging > print in professional projects.

**Advanced:**

* Error handling with exceptions (FileNotFoundError).
* Why pathlib.Path is preferred over raw string paths.
* How early validation (sanity checks) helps reproducibility and debugging.

# **📊 Cell 3 — Correlation, Pair Plots & Time Series Visualization**

### **🔹 What it does**



1. **Time Series Plots**
   * Plots training and test data side by side, with the same scaling for consistency.
   * Uses **time in hours** (converted from seconds).
   * Variables:
     + Current density (Stromdichte)
     + Inlet temperature (Eintrittstemperatur)
     + Cell temperature (Zelltemperatur)
   * Consistent color mapping (red shades = temperatures, blue = current density).
   * Adds legends, labels, consistent axis limits across train/test.

### **🔹 Why it’s needed**

* **Correlation analysis** → Check for multicollinearity (important for regression-type models), and see which features relate to the target.
* **PairGrid** → Visual inspection of linearity/nonlinearity, distributions, outliers.
* **Time series plots** → Understand dynamic behavior, trends, variability, operating ranges.
* **Consistency (same colors, axis scaling)** → Makes comparisons between training and test easier and avoids misleading interpretations.

### **🔹 Underlying Concepts to Know**

**Basic:**

* Difference between **Pearson** (linear) and **Spearman** (monotonic) correlations.
* What a heatmap, scatter plot, histogram show.
* Why we visualize train/test separately.

**Intermediate:**

* Why high correlation between features can be a problem (multicollinearity).
* How scatter patterns hint at nonlinear relationships → guiding model choice (e.g., linear regression vs SVR/MLP).
* Why scaling axes and using consistent colors improves interpretability.

**Advanced:**

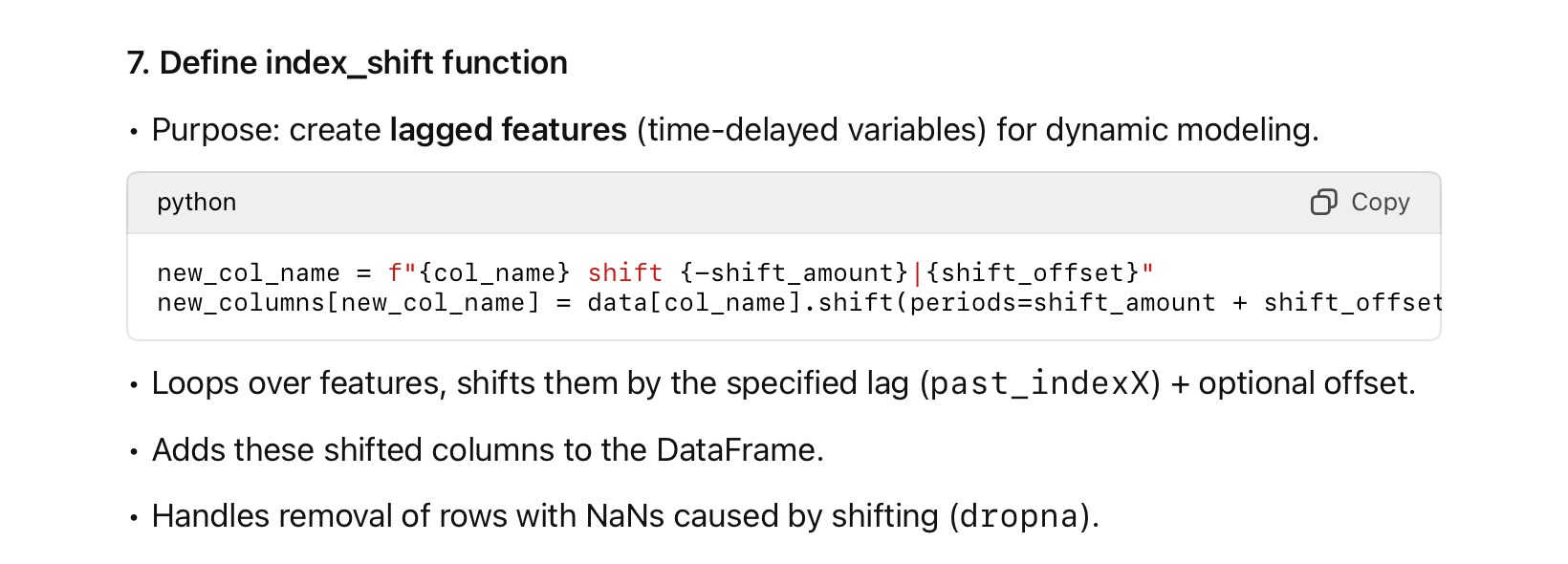
* Time series characteristics: trends, seasonality, noise.
* Data leakage: why you don’t mix train/test when plotting ranges.
* Using visualization to decide feature engineering (lags, transformations).

# **⚙️ Cell 4 — Scaling & Feature Shifting**

### **🔹 What it does**







Why lag features?

* The surrogate model predicts cell temperature dynamically, so past feature values (e.g., previous inlet temperature or current density) are needed to capture temporal dependencies.

### **🔹 Why it’s needed**

* Scaling: SVR and MLP are sensitive to feature magnitude.
* Separate target scaler: target range should also be normalized for stable training.
* Reconstruction as DataFrame: easier visualization and tracking of column names.
* Lagged features: essential for modeling time-dependent behavior of the electrolysis cell.

### **🔹 Concepts to Know**

Basic:

* Why standardization matters for ML models.
* Difference between fit and transform (avoid leaking test data info).
* What lagged features are in time series.

Intermediate:

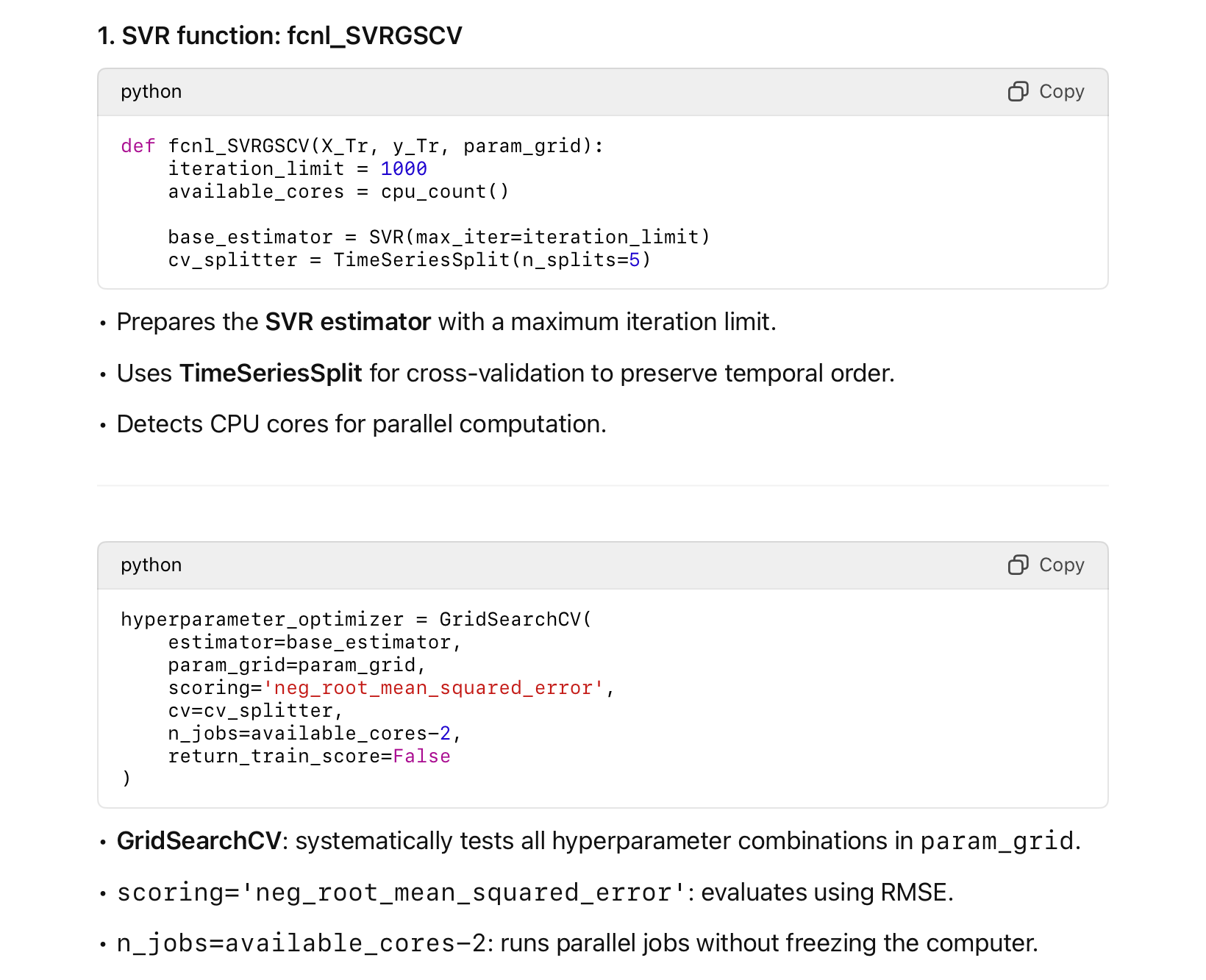
* How reshaping works for single-column scaling.
* Why train/test split must be scaled using training statistics only.
* Data leakage risks if test data is scaled independently.

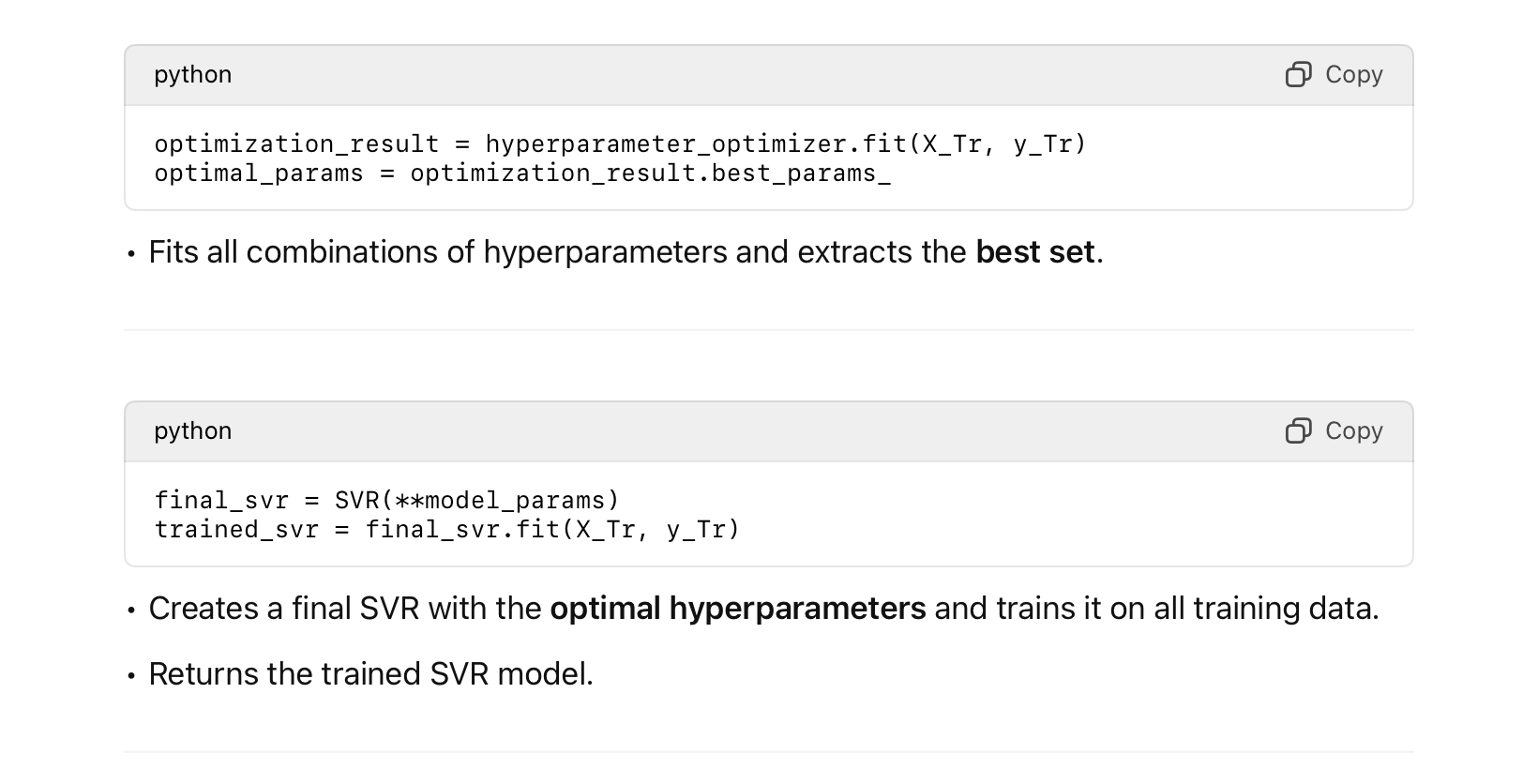
Advanced:

* How lagged features approximate system memory (dynamic response).
* How the index\_shift function supports multi-step lagging for different columns.
* Alternatives: using rolling windows, ARX/ARIMA models, or state-space features.

# **🧠 Cell 5 — SVR & Neural Network Training with GridSearchCV**

### **🔹 What it does**





#### **2.**

#### **MLP function: fcnl\_train\_hyperp\_nn**

* Very similar to SVR, but adapted for neural networks.
* Steps:
  1. Initialize MLPRegressor with max iterations and random state.
  2. Use TimeSeriesSplit for cross-validation.
  3. Perform GridSearchCV on param\_grid.
  4. Extract best parameters.
  5. Create a new MLPRegressor with the optimized parameters.
  6. Return both the trained model and the GridSearchCV object (useful for evaluation and metrics).

### **🔹 Why it’s needed**

* Hyperparameter optimization: SVR and MLP have critical parameters (e.g., C, epsilon, kernel for SVR; hidden\_layer\_sizes, alpha, activation for MLP).
* TimeSeriesSplit: preserves temporal order; prevents “future leakage.”
* GridSearchCV: automates the search across parameter combinations.
* Parallelization (n\_jobs): speeds up computation.
* Separate final model training: ensures the final model uses all training data with optimal hyperparameters.

### **🔹 Concepts to Know**

Basic:

* What SVR and MLPRegressor do.
* What a hyperparameter is.
* RMSE as an evaluation metric.

Intermediate:

* Why TimeSeriesSplit is needed for time-dependent data.
* Why fit vs fit\_transform matters (applies to scaling too).
* Difference between training during GridSearchCV and final model training.

Advanced:

* How GridSearchCV avoids overfitting by using cross-validation.
* Why parallelization is important for large grids or large datasets.
* Alternative optimization strategies: RandomizedSearchCV, Bayesian optimization.

# **🔑 Cell 6 — Pseudo-ARX Model Function (**

# **fcnl\_pseudoARX**

# **)**

### **🔹 What it does**

1. Function Purpose

def fcnl\_pseudoARX(modell, df\_train\_in, df\_test\_in, yheader, xshift, yshift, td=0):}Implements a pseudo-ARX (AutoRegressive with eXogenous inputs) surrogate model.

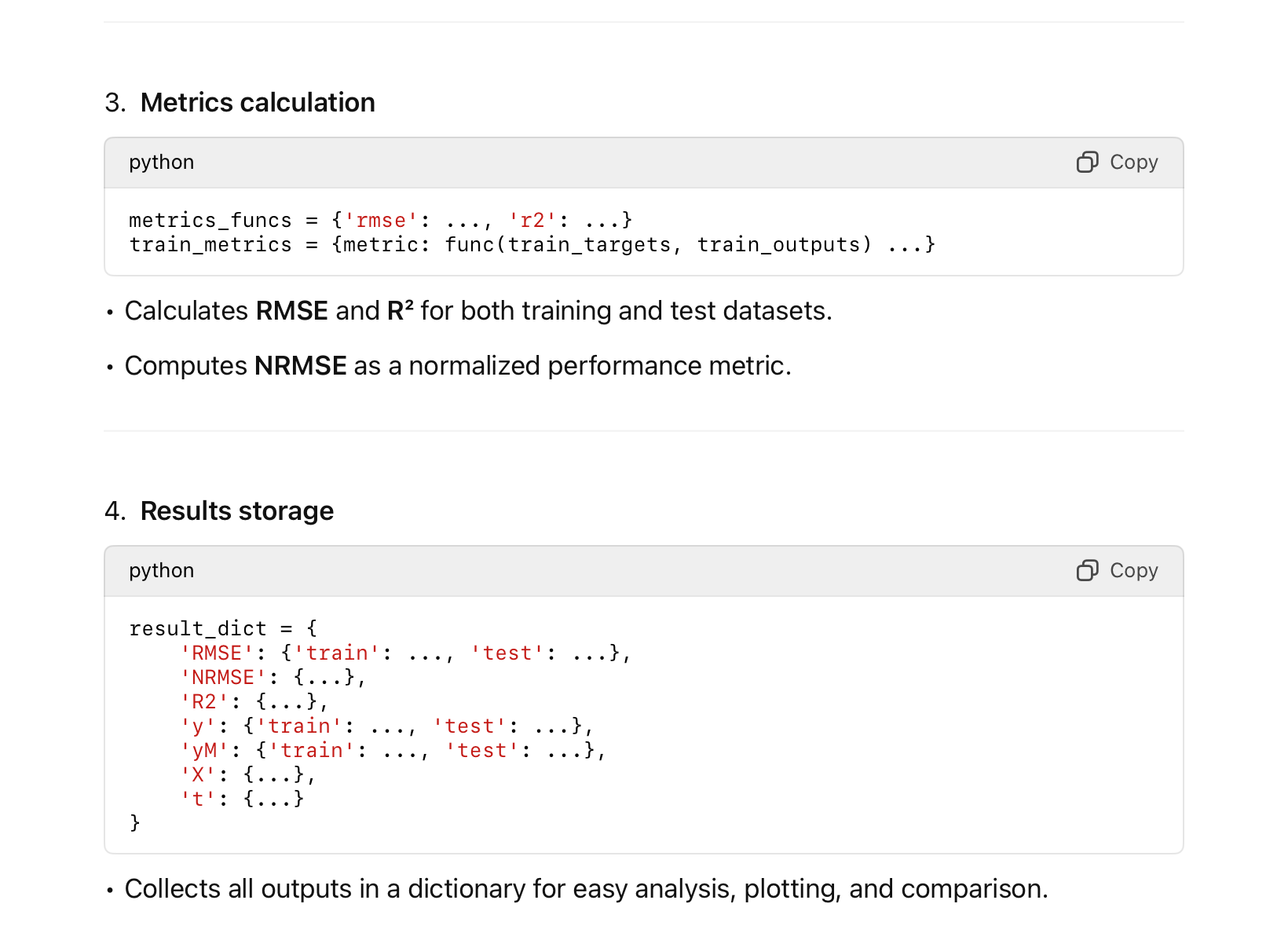
* Can handle OLS, SVR, or NN models.
* Supports lagged inputs (xshift) and lagged output (yshift) to capture dynamics.
* Can also include optional time delays (td).

1. Internal helper function (fcnl\_loop)

* Performs training and recursive prediction for each row in the dataset.
* Steps:
  1. Prepare features using index\_shift (create lagged variables).
  2. Slice time vector and target vector to align with lagged features.
  3. Initialize measured target history (measured\_y) for autoregressive predictions.
  4. Prepare combined training matrix (lagged inputs + lagged outputs).
  5. If modell is svr or nn, train with hyperparameter tuning (using previously defined functions).
  6. Loop through each row:
     + Predict next step using current input + past outputs.
     + Update the autoregressive measured target array.
     + Store predictions and references in arrays for later evaluation.

Key concept:

* This is a one-step-ahead prediction loop, essential for dynamic systems, simulating real-time prediction.



### **🔹 Why it’s needed**

* Captures dynamic behavior: CAE cell temperature depends on past inputs and past outputs.
* Supports multiple model types: OLS, SVR, NN.
* Recursive prediction mimics real-time control: each new prediction uses previous predicted values.
* Provides metrics for model comparison and validation.
* Prepares outputs for plotting, evaluation, and further optimization.

### **🔹 Concepts to Know**

Basic:

* ARX models: use past inputs and outputs to predict current output.
* RMSE, R², NRMSE: evaluate prediction quality.
* Difference between training and test evaluation.

Intermediate:

* Why lagged outputs are necessary for time-dependent systems.
* How recursive prediction works in dynamic models.
* Data alignment: slicing, shifting, and resetting indices to avoid misalignment.

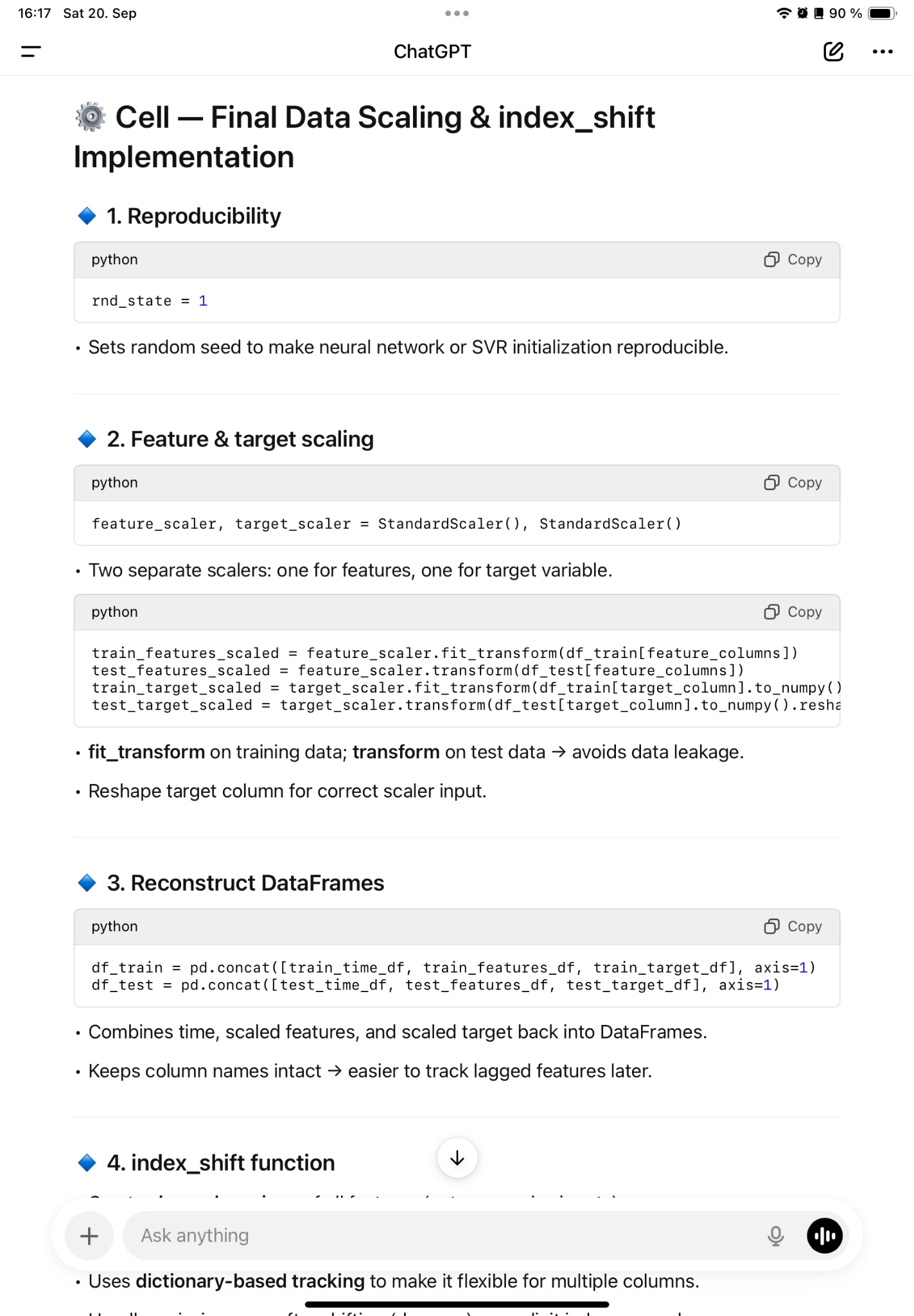
Advanced:

* Pseudo-ARX vs full ARX: you simulate autoregression without fitting a full state-space model.
* How hyperparameter tuning is integrated inside a dynamic prediction loop.
* Handling of missing data from shifting (dropna) and padding strategies.
* Generalization: why recursive prediction is harder than one-step prediction.

### **🔹 Possible Exam Questions**

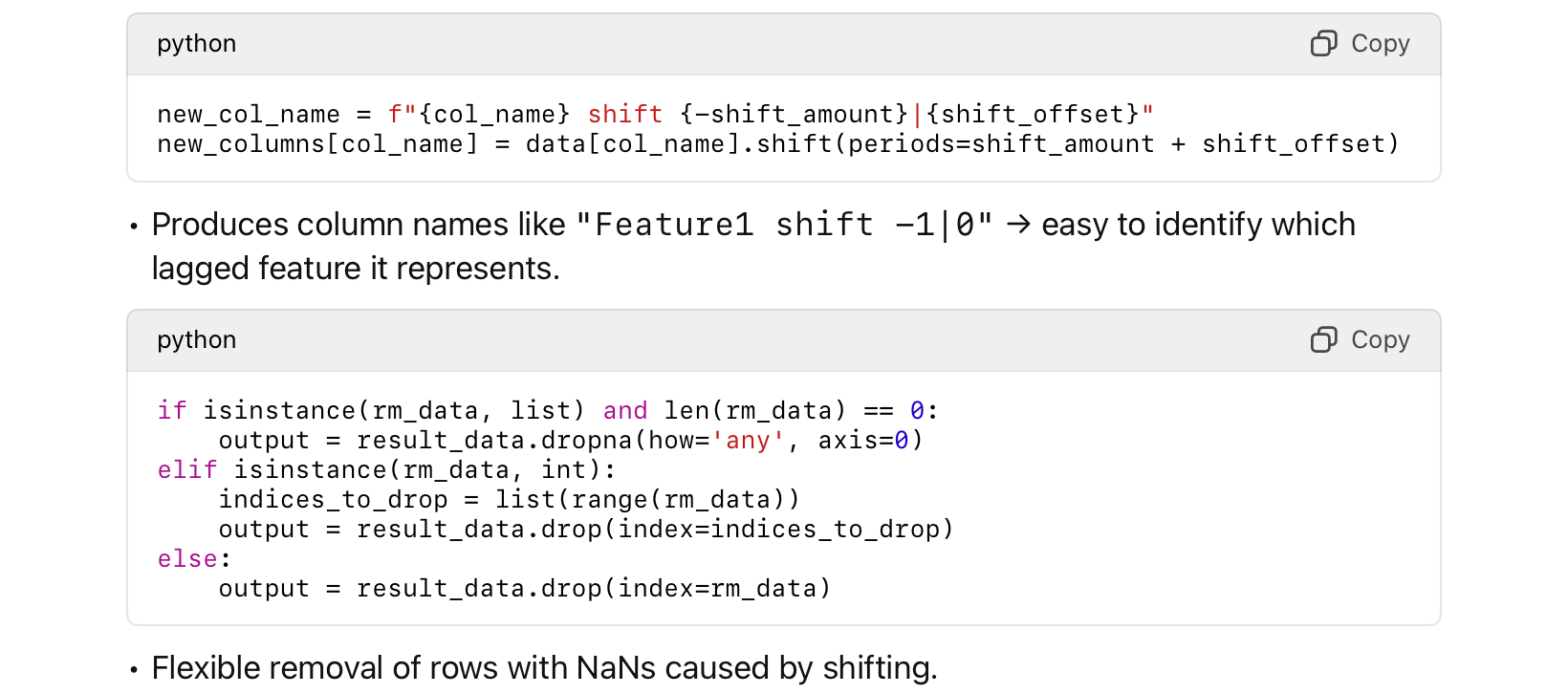
* What is a pseudo-ARX model and why is it used here?
* Why do we need lagged input and output variables?
* Why recursive prediction instead of direct multi-step regression?
* How do you evaluate model performance dynamically?
* What is the difference between train and test prediction loops?
* Why integrate SVR/NN training inside this function?

# **⚙️ Cell — Final Data Scaling & index\_shift Implementation**



### **🔹 4. index\_shift function**

* Creates lagged versions of all features (autoregressive inputs).
* Handles per-column shifts and optional offsets.
* Uses dictionary-based tracking to make it flexible for multiple columns.
* Handles missing rows after shifting (dropna) or explicit index removal.



### **🔹 Why it’s needed**

1. Scaling:
   * Ensures models like SVR and MLP converge properly.
2. Lagged features:
   * Captures dynamic behavior of the CAE system.
   * Previous values of inputs and outputs are critical to predict the next cell temperature.
3. DataFrame reconstruction:
   * Easier to work with column names when generating lagged features.
4. Flexible shifting:
   * Allows different shift amounts per feature.
   * Handles missing data in a controlled way.

### **🔹 Concepts to Know**

Basic:

* Why scaling is important.
* Difference between fit and transform.
* Purpose of lagged features.

Intermediate:

* How to handle missing rows after shifting.
* How to shift different features by different amounts.
* Reshaping target for StandardScaler.

Advanced:

* How lagged features simulate system memory for autoregressive modeling.
* Flexible removal of rows using rm\_data parameter.
* Importance of keeping consistent DataFrame structure for downstream modeling.