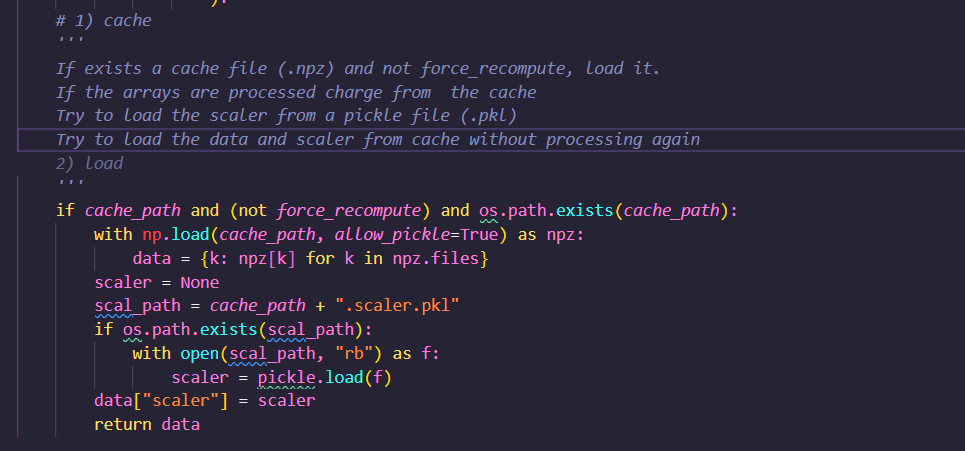
Function parts

1. Cache



In this part resumed check if there exist a cache file (cahe\_path) and that the recompute was offset (force\_recompute)

Then charge the arrays already processed form .npz (npz is a compressed file that save multiple arrays form NumPy, like a .zip were all the filles inside are arrays) .

I used a npz because the function generate different results like x\_train, y\_train, xtest, dates, etc. and npz is optimized for NumPy.

Then charge the scaler from (.scaler.pkl) if exist, .pkl because there is saved the scaler of scikit-learn.

And finally returns the processed and saved data.

1. Read data

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

In this part I used the subfunction “\_read\_raw” to read the original data set and return it clear starting by checking the format, if it is csv or parquet to support the most common, for a future, I could implement the excel format to cover more formats

Then validates the date column, converts to pandas format and set in ascendant order to finalize cleaning the “errors” and reset the index to give continuity.

Finally normalize the columns to minuscules and quit spaces

1. Selected columns

Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.

In this part I use the subfunction “\_resolve\_features” where the columns passed as arguments to “target\_col and feature\_cols)” are normalizes to minuscule to match with the DataFrame

1. NaNs

Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.

In this part I use the subfunction “\_handle\_nans” where first, creates a sub DataFrame with the targets and features, then with the strategy (drop, mean or ffill\_bfill) eliminates or supplant the NaN values finally replaces the original data fram with the clan data of the sub DataFrame in case there still Nan values, it eliminates.

Strategy examples:

data = {

"date": ["2023-01-01","2023-01-02","2023-01-03"],

"close": [10, None, 12],

"volume": [100, 200, None]

}

df = pd.DataFrame(data)

Drop:

date close volume

0 2023-01-01 10.0 100.0

Mean:

date close volume

0 2023-01-01 10.0 100.0

1 2023-01-02 11.0 200.0 # close = mean(10,12)

2 2023-01-03 12.0 150.0 # volume = mean(100,200)

Ffill\_bfill:

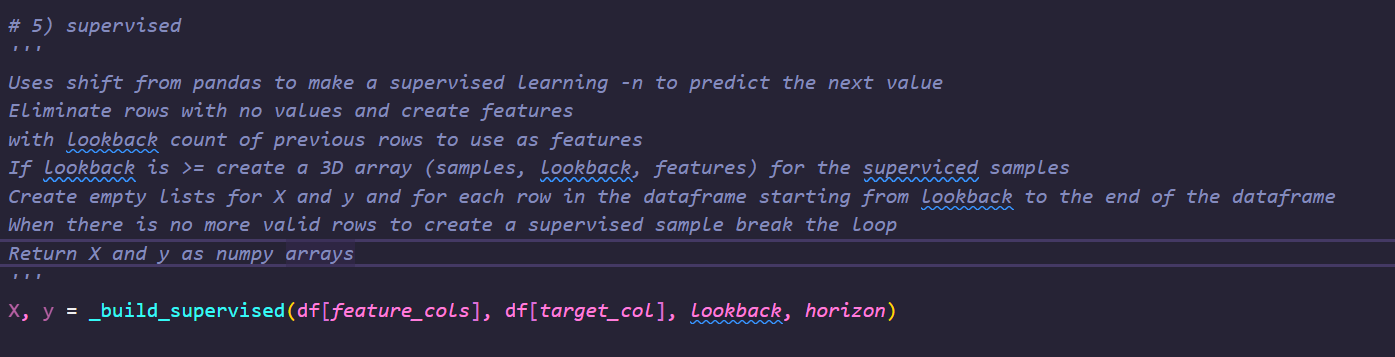
date close volume

0 2023-01-01 10.0 100.0

1 2023-01-02 10.0 200.0 # close with previous value(10)

2 2023-01-03 12.0 200.0 # volume with previous value (200)

1. Superviced

Texto

El contenido generado por IA puede ser incorrecto.

In this section of the code I used the subfunction “\_build\_superviced” to make a supervised problem.

First the Target “y” is the variable that want to be predicted, It would be obtained by shift function from pandas, which shifts the series to align at each instant “t”, the value that happens in “t+horizon”. That is, it converts the future into the value that will be used as the prediction label.

Then the feature “x” are the values in the past that will be used by entrance for the model, the parameter lookback defines how many steps in the past will be used. Example: lookback=3 and horizon=1 will use [t0, t1, t2] to predict “t3”.

Then the rows that get incomplete by the shift will be eliminated

Then empty list will be initialized to save the pairs of “X\_woindow, y\_target”. For all positions.

* Takes for all instant t a block of lookback values and it used like feature set
* It will be associated for the target that corresponds to that windo
* The loop stops when there are no more future data

Finaly, that lists will convert to NumPy arrays:

* X: array 3D with (n\_samples, lookback, n\_features).
* Y: array 2D with (n\_samples,)

Example:

With an original series with only one feature

t: 0 1 2 3 4 5

value: 10 11 12 13 14 15

with lookback=3 and horizon=1

The model looks 3 past value to predict the next “1” value

Windows:

Window X0: [10, 11, 12] → Target y0 = 13

Window X1: [11, 12, 13] → Target y1 = 14

Window X2: [12, 13, 14] → Target y2 = 15

Then

X = [

[10, 11, 12],

[11, 12, 13],

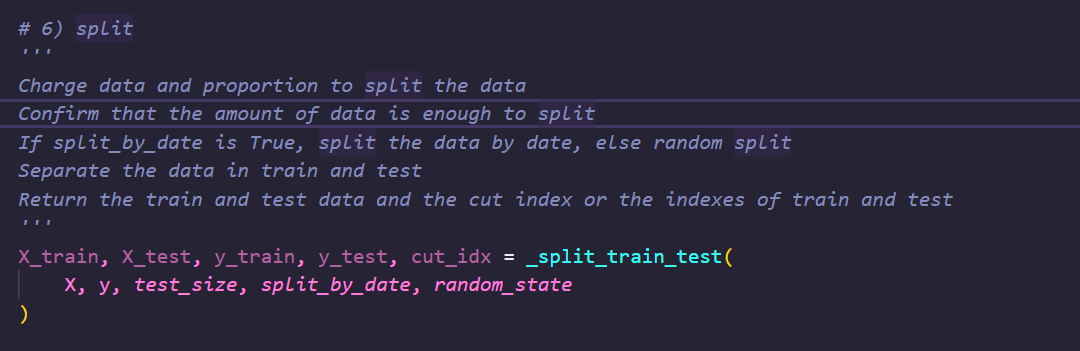
[12, 13, 14]

]

y = [13, 14, 15]

So in resume this creates windows of features with the longitude of lookback, for each “n\_lookback” numbers predict ne next “n\_horzon” value.

1. Split

Texto

El contenido generado por IA puede ser incorrecto.

In this section, I use the “\_split\_tranin\_test” function to make the split for test and train.

First, import train\_test\_split from sklearn because it permit ti divide arrays and matrices on random from.

Then, use the parameters X=features and y=target, test size to define the proportion for test example: 0.2 = 20%, split\_by\_date to know if the split will be sequential or random.

“n” is the total number of samples

Then verify if the test size is a valid number, if the number of samples is minor of 2 there is no sense to divide

Then make the split

Split by date:

Divide the data set into sequential order

Calculate the cut point (the amount of data to test and train), check that there is always a date in train and test

Cut the data in two blocks

Example

Test\_size = [1,2,3,4,5,6,7,8]

Train = [1,2,3,4,5,6] past

Test = [7,8] future

Is determinist because is =the same cut point and maintain the sequence

Split random:

All is disorder, it does not have sequence

Select a random percentage

Shuffle = True mix the rows before assaying the test and train

Random\_state is the aleatorily

Examample

Data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Test\_size=03 random\_state=42

Train: [1, 2, 3, 4, 5, 7,8]

Test: [6, 9, 10]

Then return the train and test arrays and the original idex to reproduce it if is necessary

1. Scale

Texto

El contenido generado por IA puede ser incorrecto. Texto

El contenido generado por IA puede ser incorrecto.

This part create the scaler based in the selection, it uses the subfunction “\_make\_scaler”

First, check that there is a scaler selected, if not continue the next part

Then gets the number of features witout if its 3D or 2D

Then reform to 2D making 3D to 2D

Then Fit in train and tranform in the both

Minimum, maximum, mean only in train

Tranform train and test

Finally reconstruct the original form to continue with the star 3D or 2D

Example:

scaler = MinMaxScaler()

scaler.fit(X\_train)

X\_train = np.array([

[10, 100],

[20, 200],

[30, 300]

])

X\_test = np.array([

[15, 150],

[25, 250]

])

MinMaxScaler

* Fits using **train only**.
* For each feature:

min = [10, 100], max = [30, 300].

* Formula: (x - min) / (max - min)

print(scaler.transform(X\_train))

[[0.0 0.0]

[0.5 0.5]

[1.0 1.0]]

print(scaler.transform(X\_test))

[[0.25 0.25]

[0.75 0.75]]

StandardScaler:

* Fits using **train only**.
* For each feature:

mean = [20, 200], std ≈ [8.16, 81.65].

* Formula: (x - mean) / std

scaler = StandardScaler()

scaler.fit(X\_train)

print(scaler.transform(X\_train))

[[-1.2247 -1.2247]

[ 0.0000 0.0000]

[ 1.2247 1.2247]]

print(scaler.transform(X\_test))

[[-0.6124 -0.6124]

[ 0.6124 0.6124]]

MinMax: squeezes values into [0,1].

Standard: centers at 0 and scales by variance (values can be <0 or >1).

1. Package

Texto

El contenido generado por IA puede ser incorrecto.

This section creates a dictionary (out) that contains all the processed data and metadata. The output can then be saved into .npz (arrays) and .pkl (scaler) files, so it can be reloaded later without recomputing.

* **Prepared datasets**
  + "X\_train", "y\_train": training data.
  + "X\_test", "y\_test": test data.
* **Scaler used**
  + Saves the scaler object.
  + This allows reusing the same normalization later, for example in production.
* **Reference columns (cols)**
  + "features": list of feature columns used.
  + "target": name of the target column.
  + "date\_col": name of the date/time column (for traceability).
* **Metadata (meta)**  
  Stores the entire “recipe” used for preprocessing, which is useful for reproducibility and debugging:
  + Configuration parameters: lookback, horizon, split\_by\_date, test\_size, scaler\_type, cache\_path, random\_state.
  + Dataset information:
    - "n\_samples\_raw": number of rows in the original DataFrame.
    - "n\_samples\_supervised": number of supervised samples (windows) generated.
  + Split information:
    - "train\_end\_index": index cutoff between train and test (or index dict if random split was used).
  + "created\_at": timestamp of execution (useful for caching and versioning).

1. Save

Texto

El contenido generado por IA puede ser incorrecto.

This section saves the data

First check is cache is enable.

Then saves the arrays and dictionaries with NumPy and creates a .npz file that contain the dataset

Saves the scaler with pickle

Finaly returns the dictionary with all the data, columns, metadata and scaler