Split into training and holdout validation

Training can use rolling window fo all models

Ml models use lagged features

Evaluation is between the training data and the prediction on the training data

1. Divide training and testing by 80% and 20%, respectively.2. Training model for rolling window origin forecast 3. k-fold cross-validation to evaluate the trained model on the training set 4. grid search fine tuning using k-fold cross validation for evaluating and selecting the best parameters based on result 5. Predict the training set and select the best models based on residuals. 6. Test the best model on the test set.

1. Split 80% 20% into training and testing sets.
2. Preprocess the training data by creating lagged features.
3. For each candidate model: a. Define a grid of possible hyperparameters. b. For each combination of hyperparameters: i. Train the model on the training set using the current combination of hyperparameters and a rolling window origin forecast. ii. Use k-fold rolling window cross-validation on the training set to evaluate the model's performance for the current combination of hyperparameters. iii. Calculate performance metrics (e.g., MAE, RMSE) for the current combination of model and hyperparameters. c. After evaluating all combinations of hyperparameters for the current model, select the best combination based on the performance metrics.
4. Train the best models with the optimal hyperparameters on the entire training set using the rolling window origin forecast.
5. Analyze the residuals of the best models and perform diagnostic tests as needed.
6. Test the best model on the holdout test set to see how well it performs on unseen data.
7. Start with 5 years of data and create a holdout test set of the last 30 days. This set will be used only for final evaluation.
8. Preprocess the remaining data (5 years minus the 30 days) by creating lagged features to capture the temporal dependencies in the data.
9. Use the preprocessed data (with lagged features) for training, evaluation, cross-validation, and optimization of your model. a. Perform a rolling window approach for training and evaluation. b. Use cross-validation techniques, such as time series cross-validation, to fine-tune your model's hyperparameters. c. Optimize your model based on the evaluation metrics.
10. Analyze the residuals of the models (e.g., check for autocorrelation, normality, and homoscedasticity) and perform diagnostic tests.
11. Select the best model based on a combination of performance metrics, robustness, and model diagnostics.
12. Preprocess the entire non-holdout dataset (5 years minus the 30 days) by creating lagged features. Then, train the selected model on this preprocessed dataset.
13. Preprocess the holdout set (the last 30 days) by creating lagged features using the same approach as before. Finally, test the model on this preprocessed holdout set to see how well it performs on unseen data.

INPUT: HISTORICAl time values

* Date tiem feature
* Lag feature

Output: future time values

X: (samples, time steps, features) input for prediction

Y: (samples, horizon) prediction made

Univariate + lag

Multivariate + lags

Multiple output forecast develop one model and apply it to predict the

next two hours in one single computation

long term forecast

3 month horizon

1. Inputs vs. Outputs
   1. What are the inputs and outputs for a forecast?
2. Endogenous vs. Exogenous
   1. What are the endogenous and exogenous variables?
3. Unstructured vs. Structured
   1. Are the time series variables unstructured or structured?
4. Regression vs. Classification
   1. Are you working on a regression or classification predictive modeling problem?
   2. What are some alternate ways to frame your time series forecasting problem?
5. Univariate vs. Multivariate
   1. Are you working on a univariate or multivariate time series problem?
6. Single-step vs. Multi-step
   1. Do you require a single-step or a multi-step forecast?
7. Static vs. Dynamic
   1. Do you require a static or a dynamically updated model?

esis that ts1 predicts ts2, is this a univariate probllem or multivariate, my application is time series prediction using machine learning and deep learning

If you have a hypothesis that ts1 predicts ts2, this is a multivariate time series problem. In a univariate time series problem, we only have one time series variable to work with, whereas in a multivariate time series problem, we have multiple time series variables that may be related to each other. In your case, ts1 and ts2 are two different time series variables, and you are trying to predict ts2 using ts1 as a predictor variable. Therefore, this is a multivariate time series problem.

In terms of machine learning and deep learning, there are various techniques that can be used to model multivariate time series data, such as vector autoregression (VAR), recurrent neural networks (RNNs), and long short-term memory (LSTM) networks. These models can take into account the relationships between the different time series variables and can be used to make predictions based on the input variables.

how do I set up my data into a supervised learning problem for ts1 to predict ts2

Sure, I'd be happy to help you set up your data for a supervised learning problem to predict ts2 using ts1 as a predictor variable. Here's a step-by-step guide:

1. Load the time series data for ts1 and ts2 into a pandas DataFrame.
2. Define the prediction window (i.e. the number of time steps to predict into the future).
3. Create a new column for the target variable (ts2) by shifting the time series data by the prediction window.
4. Define the lagged time periods to include as features (i.e. the number of past time steps to use as input to the model).
5. Create a new DataFrame with the selected features (ts1) and the target variable (ts2).
6. Remove any rows with missing values.
7. Split the input features (train\_x) and target variable (train\_y) into separate DataFrames.
8. Split the data into train and test sets.

Here's an example code snippet that demonstrates these steps:

import pandas as pd

# Load the time series data for ts1 and ts2 into a pandas DataFrame

df = pd.read\_csv('time\_series\_data.csv', index\_col='date', parse\_dates=True)

# Define the prediction window (e.g. 7 days)

prediction\_window = 7

# Create a new column for the target variable (ts2)

df['target'] = df['ts2'].shift(-prediction\_window)

# Define the lagged time periods to include as features (ts1)

lags = [1, 5, 10, 20, 50]

# Create a new DataFrame with the selected features (ts1) and the target variable (ts2)

df\_train = pd.DataFrame()

for lag in lags:

df\_train[f'ts1\_lag\_{lag}'] = df['ts1'].shift(lag)

df\_train['ts2'] = df['ts2']

df\_train['target'] = df['target']

# Remove any rows with missing values

df\_train.dropna(inplace=True)

# Split the input features (train\_x) and target variable (train\_y)

train\_x = df\_train[[f'ts1\_lag\_{lag}' for lag in lags]]

train\_y = df\_train['target']

# Split the data into train and test sets

train\_size = int(len(df) - prediction\_window)

train\_x, test\_x = train\_x[:train\_size], train\_x[train\_size:]

train\_y, test\_y = train\_y[:train\_size], train\_y[train\_size:]

In this example code, we first load the time series data for ts1 and ts2 into a pandas DataFrame and define the prediction window (e.g. 7 days). We then create a new column for the target variable (ts2) by shifting the ts2 data by the prediction window. We define the lagged time periods to include as features (1