1. Hello. My name is Anton. Today I am going to talk about financial time series forecasting with Transformers and I am going to present the outcome of my analysis on the impact of additional features and different cross-validation techniques on model perfomance.

2. When you deal with financial time series data and want to train a model, 2 questions arise natually: Which data should be used to help the model understand complex relationships within financial data and how can we design a crossvalidation technique that better captures the temporal nature of **nonstationary** time series? So thats what we test: we test whether introducing more features in the form of additional features related to the stock or in the form prices of correlated stocks increase the model perfomance. And we test whether our custom cross-validation technique yields lower error.

3. For the crossvalidation:

    1. Figure 3 (on the top) illustrates the default train/validation splitting approach. In this approach, the last 20% of the time series data are reserved for validation.
    2. In our custom crossvalidation approach (seen on the bottom), we allocate validation batches throughout the entire data timeline. We hope that this approach makes the model generalize better. **Comment** We can think that we train the model basically on various distributions and make the model understand which "law" (which is practially random) the given data follows. The size of the training/validation batches is optimized by iteratively adjusting the training and validation window sizes to maximize the total data used while ensuring that the validation set comprises 15% to 20% of the total data.

4. Moving on to the data we used: The study is based on the historical stock market data which we downloaded through APIs. The dataset consists of historical daily closing prices for three correlated stocks. Additional features includes daily technical indicators and balance sheet statements.

    1. The time period is $N = 6092$ days (1 day - 1 observation).
    2. We perform feature selection using *Random Forest* and *XGBoost* algorithms (calculation: for each tree, whenever a feature is used to split a node, the impurity reduction is recorded. The feature importance is the average of these reductions over all trees in the forest)
    3. *Sliding window*: segmenting the time series data into patches (patch length $(L) = 64$, stride $= 1$)
    4. *Preprocessing*: standardization, mean of 0 and a variance of 1 (after train / validation splitting)

5. Now, for the model: As a Transfomer backbone model we use PatchTST introduced in 2023, which currently holds the state-of-the-art position for multivariate time series forecasting tasks. Unfortunately, PatchTST does

not support an inconsistent number of input and output channels (see Figure 5), meaning that that the number of features used for training should equal to the number of predicted features. Thats what we change in the architecture, namely we keep the backbone of the model and change only the head. Additionally, to enable the model to output prices and also probabilities for three defined classes, we design a custom head with two outputs. **Comment**: The first output, for price prediction, is produced by a fully connected layer with an output dimension of 1. The second output, for label classification, is produced by a fully connected layer with an output dimension of 3.

6. You might ask: which classes? We define the directional labels, which represent an increase in price by more than 5%, a decrease in price by more than 5%, and last one which represents no substantial increase or decrease in a price (meaning between these two labels). Also note that all of the model hyperpamaters (meaning number of encoder layers, number of attention heads, etc.) were found using Optuna library with 400 trials and TPESampler sampler.

7. For the results: we were testing 3 ideas (+1 that I will talk about later). Task **a**: we predict price using price, task **b**: predict price using price + additional features related to the stock, and task **c**: predict price using price + prices of correlated stocks. None of our experiments supported our hypotheses: model train on task **a**, the simplest one, for the default split outperformed every other model. Also, we compare the models' performances against the naive approach called **baseline**, in which the next stock price value is predicted as today's value. As you can see, none of our methods actually beat the baseline approach. Also, regarding labels, each model demonstrates inability to successfully predict the direction of market prices throughout time. Probabilities remain consistently below 50% no matter the label. So, since our initial approach does not yield satisfactory results, we thought that Transformers are not well-suited for univariate financial time series prediction. However, they might be better for a multivariate case. Thus, we consider a task **d** in which prices of three correlated stocks are used in training as well as in predicting; directional labels are ignored. However, again, model trained on the default split outperfomces the one trained on the custom split, but does not outperform the baseline model.

8. So, in conclusion:

    1. We utilized PatchTST as the backbone model with custom-designed heads for univariate and multivariate prediction.
    2. We proposed a cross-validation technique for time series data.
    3. Tested whether incorporating additional features in the training data can improve the Transformer model performance

9. Ultimately, none of these strategies are proven to be effective. However,

we know what might have gone wrong and what ideas to try next. These are described in the report under “Discussion” section.

10. Thank you.

- **Modify custom cross-validation approach**: after the study, we noticed that our train / validation splitting approach might conflict with the sliding window technique. Namely, since all of the training batches are used during one training iteration, at some point during training model has to predict indices that are in the next training batch. The same applies to the validation set. Naturally, this creates confusion for the model. One way to address this is to train the model iteratively, batch by batch. First, train the model on the first batch, then evaluate it on the first batch. Next, train the model on the second batch and evaluate it on the second batch. Continue this process for each batch until the last batch is reached. This entire cycle would constitute one epoch. This approach would eliminate all inconsistencies with the training and validation indices.
- **Utilize time series K-Fold cross validation**: another way to improve the forecasting performance of the model is to use time series K-Fold cross validation. This will likely enhance the model’s accuracy, although it will substantially increase the training time.
- **Try other models**: PatchTST may not be the best choice for this task. Exploring other models could help identify one that better fits the requirements.
- **Ensemble prediction**: after identifying models with the lowest error values, combining predictions from multiple models into a single weighted prediction could help mitigate individual model weaknesses and improve overall forecasting performance.