- Data loading and preprocessing
    - I load the data, handle missing values and split the data as features (x) and targes (y) with pandas
    - The data includes categorical and numerical data so i used OneHotEncoder for categorical data and StandardScaler for numerical data to encode them. This way i can be sure that all the features are in comparable state.
- Dataset / Dataloader
    - Since you asked me to write the code in OOP standards i choose to create a class for dataset(CustomDataset). This wat the PyTorch can use the processed data as in being able to use DataLoader for batch processing.
- Since we have more that one target class i choose to create a multi-task NN
    - This way i get a seperate output heads for each task or target class(seller_number, customer_number, main_account)
- Training and evaluation
    - Training loop works in a way that it can handle all tasks simultaneously. And i used accuracy for each target class to evaluate the model performance

- I split the data as in Testin and Training with train_test_split
    - The size that you want me to use is 0.25 so i defined the test_size=0.25 to ensure that 25% of the data(50 of 200) is held for testing.
    - I used LableEncoder to include all the classes from dataset
    - I used normalization for “amount” class in a way that the mean of it is to be 0 and the deviation of 1. Whit this i aim to achieve to stabilize the convergence of the network while training.
- So i used tuples for data processing but i had some issues with it:

  In CustoDataset calss __getitem__ returns a tuple (x[idx], y[idx]). This is the PyTorch standard format. Where X[idx] is the feature vector for the one data point and Y[idx] contains the targets where defines the model to be a multi-task NN

- Model architecture
    - The main part of the model consist of 2 shared layers but there is task specific output layers for each target class: output_seller, output_customer, output_main_account. With task specific models, I achived to make the classifications on one model and not 3 seperate models. The two layers are seemd fine but i might endup increasing it to 3

or 4 because i kina seem to approve the layers's output size to decrese to 1 but its not necessary or might be worse. And since i only have 200 documents to use both training and testing it seemd alright for to choose a less deeper model and just increase the epoch number and batch size or the learning rate.

- I had some input size issues while designning the model layers: The input sizes were not the same with the data size the layers accept. I fixed it with the line input_size = x_training_final.shape[1]. With this line the model dynamically adjust to the correct number of features.
- I used acc_score to quantify the model performance
- I aslo had some problems with as in how should i encode testing and training data, do i choose to encode them all together before splitting them was a hard choice.

here are some preformance values of the model:

Seller Accuracy: 60.00% Customer Accuracy: 75.00% Main Account Accuracy: 85.00%
when 500 epochs for training. Other epoch numbers was not very effective or different than this output so i chose not to include them. I also reviewed the code to chatgpt. You can read it from my github. I used google colab. I could explain more of the problems that i encountered especially about encoding, but to be hones i am a bit tired. So pls excuse my laziness.