Aim

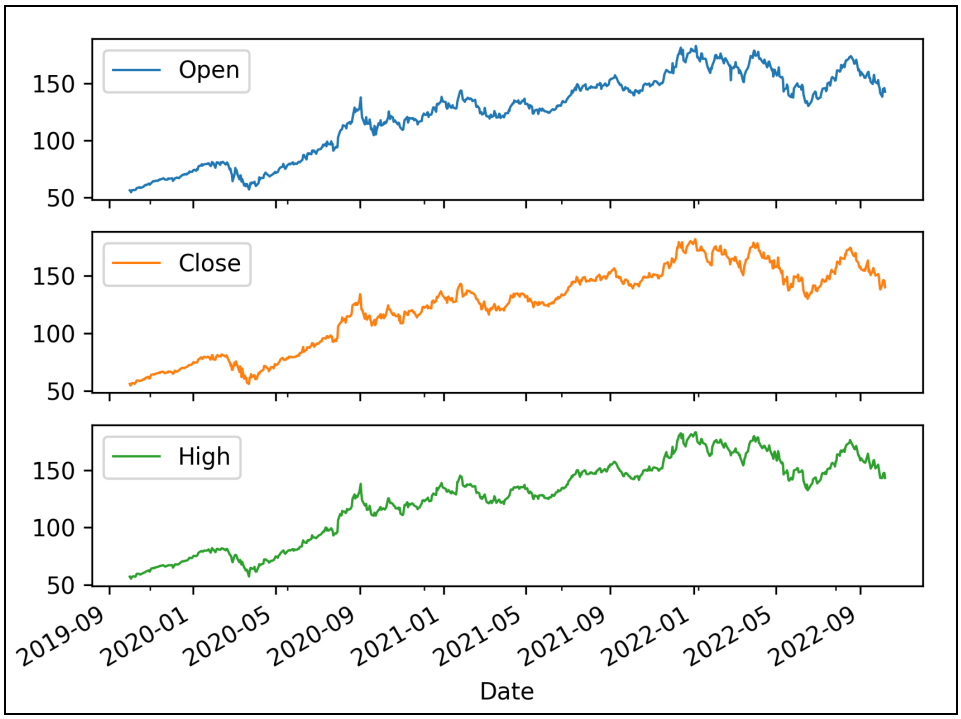
Using the data on stock quotes, create a model for predicting quotes based on the LSTM (GRU) network. Keep in mind that some pre-processing of the data may be required before training the model. The feature vector can be expanded using the simpler algorithms that we discussed in the lecture. In the conclusions, note which type of network - LSTM or GRU - is better suited for this task. It is also important to set up callbacks and checkpoints.

Dataset

As a dataset, the information about Apple Inc stock quotes will be used. A research period is between October 2019 and October 2022, which is 762 samples. Features are Date,Open,High,Low,Close,Adj Close.

	Date	Open	High	Low	Close	Adj Close
0	2019-10-01	56.267502	57.055000	56.049999	56.147499	54.932762
1	2019-10-02	55.764999	55.895000	54.482498	54.740002	53.555717
2	2019-10-03	54.607498	55.240002	53.782501	55.205002	54.010658
3	2019-10-04	56.410000	56.872501	55.972500	56.752499	55.524677
4	2019-10-07	56.567501	57.482498	56.459999	56.764999	55.536915
..	...	...	...	...	...	...
757	2022-10-03	138.210007	143.070007	137.690002	142.449997	142.449997
758	2022-10-04	145.029999	146.220001	144.259995	146.100006	146.100006
759	2022-10-05	144.070007	147.380005	143.009995	146.399994	146.399994
760	2022-10-06	145.809998	147.539993	145.220001	145.429993	145.429993
761	2022-10-07	142.539993	143.100006	139.449997	140.089996	140.089996

We can observe graphs of some features below. Though values can differ a bit, a general trend is noticeable.



Data preparation

We have decided to predict values of the 'Open' feature, so data needs to be prepared beforehand. The main thing to do is to do a scaling. It can be implemented in many ways, a MinMaxScaler tool from 'sklearn.preprocessing' library was chosen:

```
scaler = MinMaxScaler((0, 1))
arr_data = df_data_general.loc[:, 'Open'].values
arr_data = scaler.fit_transform(arr_data.reshape(-1, 1)).reshape(762)
```

Here we chose a column 'Open' from our dataset, scale its values and make a 1d array of it.

Next step is to split our 'Open' data to train and test values and labels. The principle is as follows: the input to our model is 99 sequential values taken from a dataset and a 100th one is a label. We need to make as many combinations as we can, for example the first train vector is dataset[0:98], label is dataset[99], the second train vector is dataset[1:99]

and label is dataset[100], etc…The number of such sequences is *dataset_len - num_of_sequence*.

```
def prepare_data(data, train_start, train_end, test_start, test_end):
    print(f"Number of data samples is {len(data)}")
    print(f"Number of train samples is {train_end-train_start}")
    print(f"Number of test samples is {test_end-test_start}")

    arr_X_train, arr_X_test = np.empty((0,99)), np.empty((0,99))
    arr_Y_train, arr_Y_test = np.array([]), np.array([])

    for i in range(train_start, train_end - sequence_length):
        arr_X_train = np.append(arr_X_train, np.array([data[i: i + sequence_length - 1]]), axis=0)
        arr_Y_train = np.append(arr_Y_train, data[i + sequence_length - 1: i + sequence_length], axis=0)

    for i in range(test_start, test_end - sequence_length):
        arr_X_test = np.append(arr_X_test, np.array([data[i: i + sequence_length - 1]]), axis=0)
        arr_Y_test = np.append(arr_Y_test, data[i + sequence_length - 1: i + sequence_length], axis=0)


    print("Shape y_test:", np.shape(arr_Y_test))
    print("Shape X_train:", np.shape(arr_X_train))
    print("Shape X_test:", np.shape(arr_X_test))
    print("Shape y_train:", np.shape(arr_Y_train))

    return arr_X_train, arr_Y_train, arr_X_test, arr_Y_test
```

Model

Mainly we have two popular types of neural network (N.N.) models for the time series forecast, they are GRU and LSTM, we will try both. Our N.N. can be implemented with the help of tensorflow framework. It has three layers, with a number of units - 32, 128, 100. We have a linear activation function in our output and an optimizer ‘rmsprop’. Checkpoints mechanism is also created here, so we can work with the model quicker and more effectively.

```
├── checkpoints
│   ├── checkpoint
│   ├── rnn_model_checkpoint.data-00000-of-00001
│   └── rnn_model_checkpoint.index
```

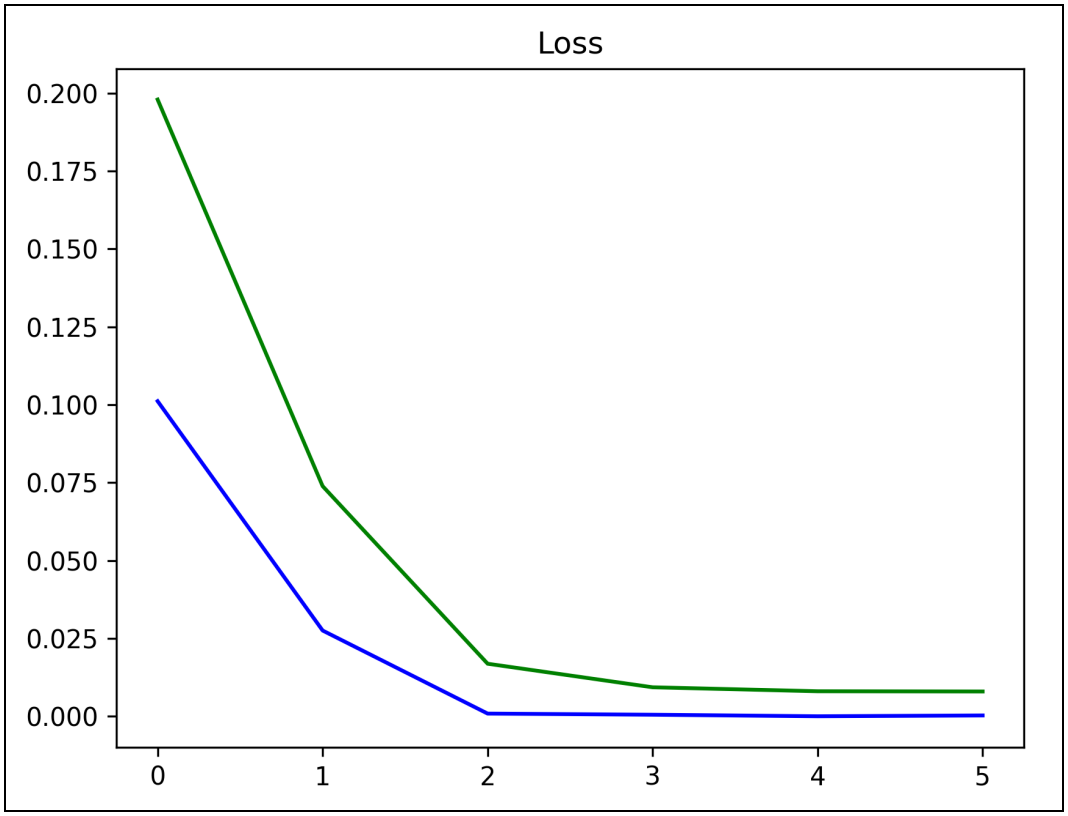
```
def build_model(v_s):
    checkpoint_filepath = 'checkpoints/rnn_model_checkpoint'
    model_checkpoint_callback = ModelCheckpoint(
        filepath=checkpoint_filepath,
        save_weights_only=True,
        monitor='val_loss',
        mode='max',
        save_best_only=True)

    model = Sequential()

    model.add(GRU(input_shape=(sequence_length - 1, 1), units=32, return_sequences=True))
    model.add(Dropout(0.2))

    model.add(GRU(units=128,return_sequences=True))
    model.add(Dropout(0.2))

    model.add(GRU(units=100,return_sequences=False))
    model.add(Dropout(0.2))

    model.add(Dense(units=1))
    model.add(Activation('linear'))

    model.compile(loss='mean_squared_error', optimizer='rmsprop', metrics= ['accuracy'])
    # 0.05

    history = model.fit(X_train, y_train, batch_size=batch_size, epochs=2, validation_split=v_s, callbacks=[model_ch
    return model, history

model, history = build_model(0.05)
```

Evaluation

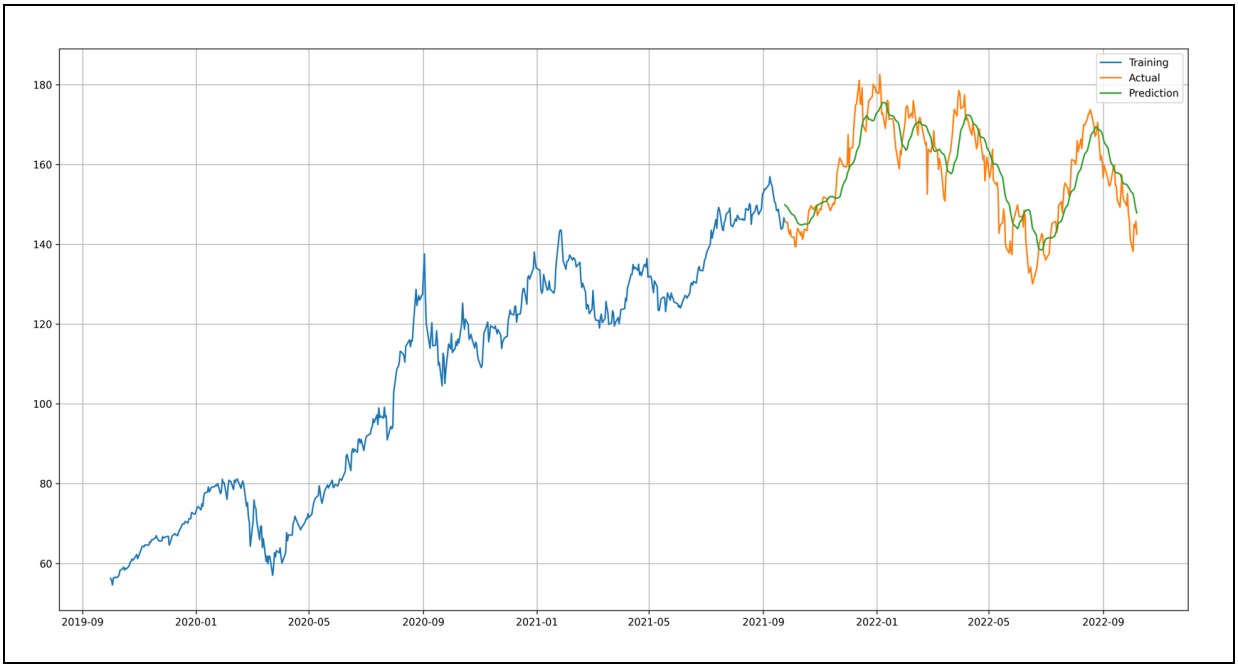
The next step is evaluation of the model and making some prediction, but beforehand It is suitable to use optuna before making any conclusions, so our hyperparameters will be chosen automatically, most probably such combination, that gives the best result (in our case we minimize a loss). Parameters that are being tuned can be seen in the dictionary ‘params’. We run 50 trials of optuna.

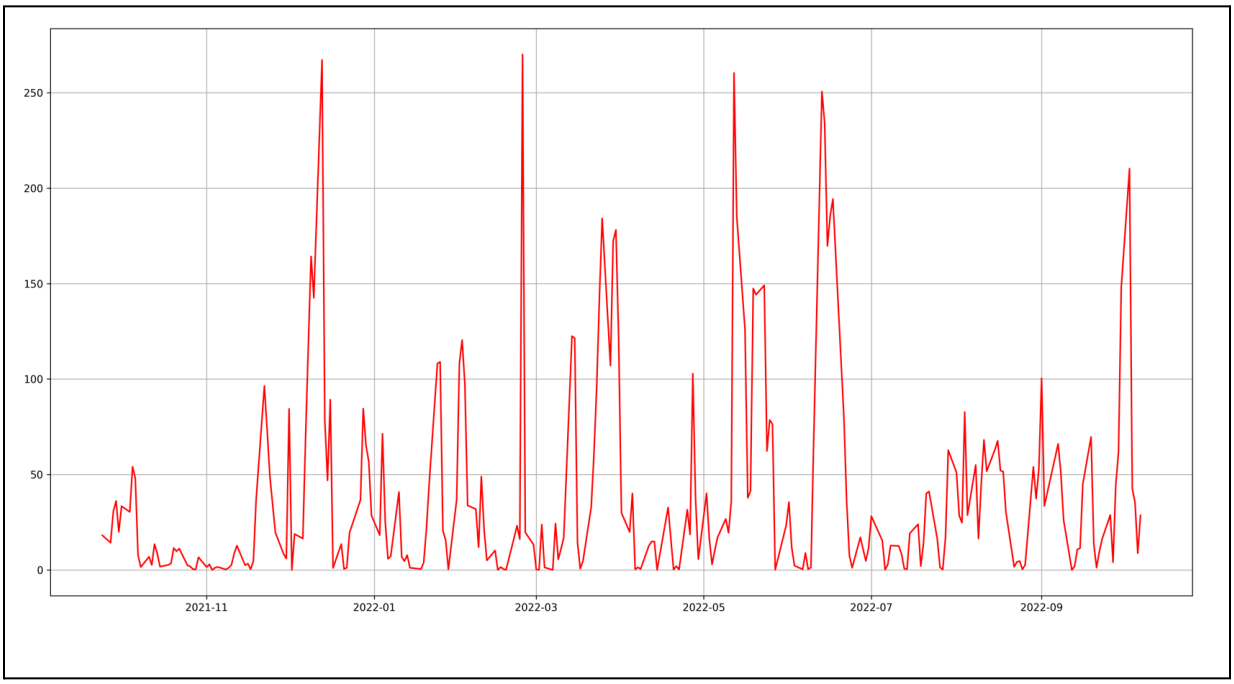
```
def objective(trial):
    params = {
        'validation_split': trial.suggest_float('validation_split', 0.001, 0.1),
        'epochs': trial.suggest_int('epochs', 3, 20),
        'batch_size': trial.suggest_int('batch_size', 20, 300),
        'dropout': trial.suggest_float('dropout', 0.0, 0.4),
        'units1': trial.suggest_int('units1', 1, 200),
        'units2': trial.suggest_int('units2', 1, 200),
        'units3': trial.suggest_int('units3', 1, 200),

    }

    model,  history = build_model(
        validation_split=params['validation_split'],
        epochs = params['epochs'],
        batch_size = params['batch_size'],
        dropout=params['dropout'],
        units1=params['units1'],
        units2=params['units2'],
        units3=params['units3']
    )
    return history.history['val_loss'][-1]

study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=30)
```

A result is: {'validation_split': 0.0015703092572811 12, 'epochs': 6, 'batch_size': 299, 'dropout': 0.39389618853447506, 'units1': 100, 'units2': 77, 'units3': 137}. The next graph shows the losses while training. Green is training loss and blue one is about validation. Values after the third epoch are so close to zero, which is good for our model.

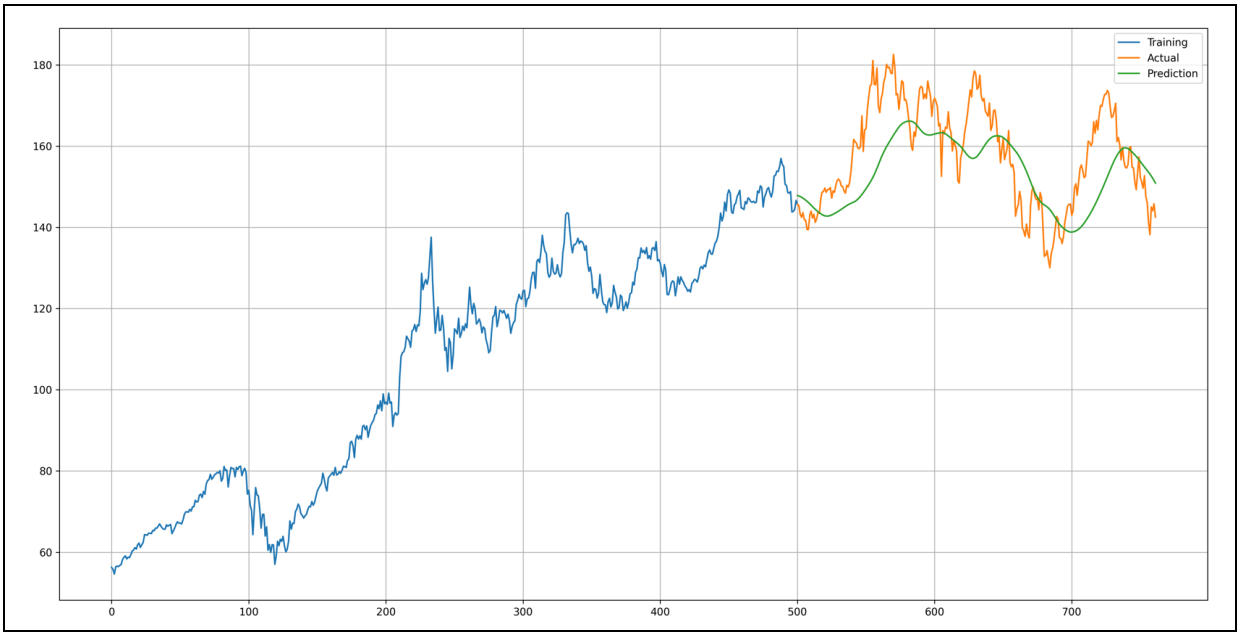


As mentioned above, 267 values will be predicted. Generally, all the changes and jumps are described ok, but values are not precise. The second graph shows how accurate the prediction was (root mean square deviation).





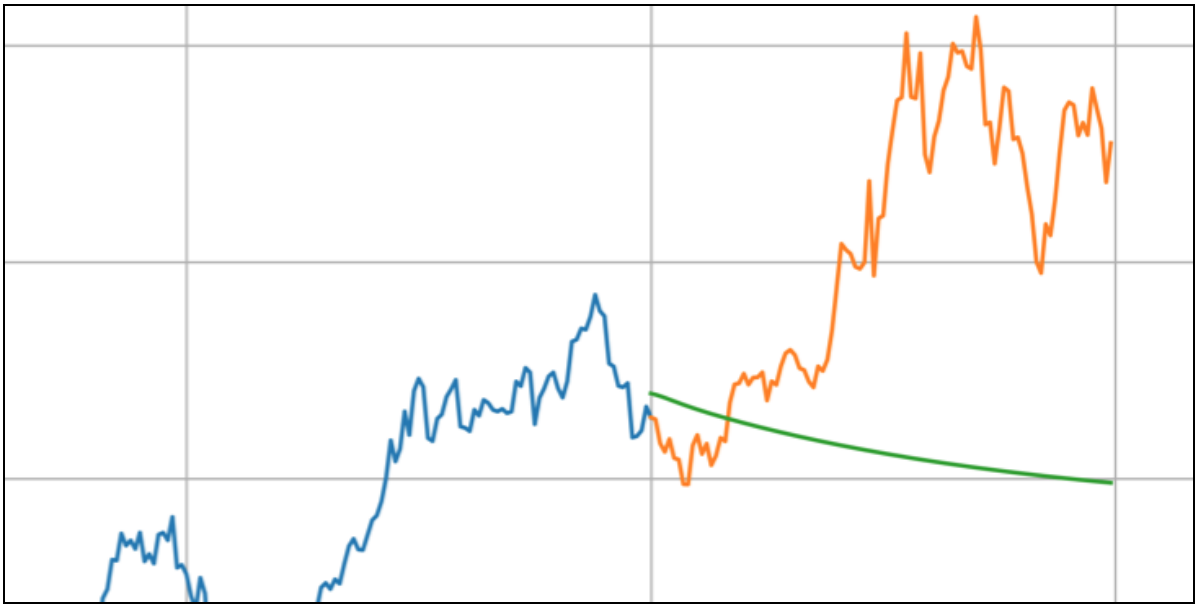
After we've changed GRU to LSTM, the result was quite worse (tested in multiple cases). Though it also describes the general trend, the line (a green one) is smoother, so in a smaller perspective a prediction is not accurate.



We've also tried to implement a prediction made on predicted values. Every time a new value was outputted, we were appending it to the end of our input vector, while the first (zero) value was removed.

```
predict_num = 100
initial_data_for_predict = X_test[0:1]
predicted_values = []
for i in range(predict_num):
    predicted = model.predict(initial_data_for_predict)
    predicted_values.append(float(predicted[0]))
    initial_data_for_predict = initial_data_for_predict[:, 1:]
    initial_data_for_predict = np.append(initial_data_for_predict, predicted).reshape(1, 99)

predicted_values = np.array(predicted_values)
predicted = scaler.inverse_transform(np.array(predicted_values).reshape(len(predicted_values), 1))
predicted = np.reshape(predicted, (predicted.size,))
```

The result is shown below. Instead of a jumping graph, we have a smooth curve.



Conclusion

To sum, it would be worth saying that in our particular case GRU type of RNN performed better and the forecast was quite good, however as we've seen none of them could predict a future based on previous predictions in a long perspective. The model performed not bad, so it can be stated that our NN can be used to predict a small number of future stock quotes changes, at least it will definitely show the sign of stock changes derivative.