Preprocessing

Exploratory Data Analysis

Before starting to work with the data, it was important to check if the dataset contains all data points and that no information is missing. To structure this procedure, we ran a check for null values on both datasets and no missing values could be found. Nevertheless, we identified some anomalies and clusters using visualisations. The data rows greater than 7000 had much lower variability than the other data. We tried to cut off the features, however we did not achieve any performance gains, the MSEs were even worse probably because the models were trained on the smaller sample. Therefore, in the other experiments we worked with the whole data set.

Scaling

To increase the performance of our models we experimented with data standardization. However, it did not increase the performance as expected probably due to the fact that the data variability was rather small with mean of features oscillating around 0 and standard deviation in many cases lower than 1. Therefore, we abandoned the idea.

Sampling

A very important point is the data partitioning into a training and a test set. Most of the upcoming models will attempt to learn by using the training data and be evaluated on the test data. As the goal is a regression analysis, random sampling was applied first. Since the dataset contains time series data, we also experimented with alternative splitting methods, i.e. that the test set contains the most recent part of the data. In theory this leads to a situation where the training of the model will be evaluated with data coming after it. Also a sampling in even and odd rows. Unfortunately, this method did not lead to a better accuracy of the models, so we stuck to the random sampling.

Training

We have trained different Machine Learning models based on the train/test data split described above. We experienced that the linear models did not perform well in terms of overfitting. Therefore, we put special emphasis on the other models which are better in dealing with time series as MLP or LTSM which resulted in the following accuracies:

Model	**Train MSE**	**Test MSE**
Linear Regression (LR)	0.1507	0.1510
Ridge Regression (RR)	0.1506	0.1513
K Nearest Neighbors (KNN)	0.1693	0.1702
Decision Tree (DT)	0.2137	0.2411
Lasso (L)	0.1514	0.1598
SVR (linear)	0.1509	0.1514
SVR (radial), standardized data	0.1124	0.1350
Random Forest (RF)	0.0620	0.1279
MLP Neural Network	0.1282	0.1284
Gradient Boosting	0.0841	0.1490
LTSM	0.1521	0.1913

Hyperparameter Optimization

To enhance the model performance, cross-validation was always used as a validation technique to better optimize and verify the target model. By splitting the training set into a number of subsets, a series of train and evaluation cycles could be used. We used a five fold cross-validation because of the relatively small size of the dataset. This helped to optimize the model without having to use additional data.

We experienced a very good MSE on the train data, but this was not reproducible by applying the model to the test set. This huge overfitting seems to be caused by the dependence between data points and the relatively small amount of available data.

Tested Models

The ARIMA model

We tried to take advantage of the fact that we were dealing with time series by implementing an ARIMA model.
The goal was to find the optimal arima model for each time series and then use it for prediction.
We decided to work directly on the test set; we changed its shape to have time series as column indices and time as row indices. We split the test set into: test_feature (the first 49 observations of the time series) and test_targets (the 50th observation of the time series). In order to find the hyperparameters (p,q and d) we opted for a grid search approach and ended up with 3141 different sets of hyperparameters.
We fitted each "optimized" ARIMA model to the corresponding complete time series (with 50 observations) and used that model to predict the 51st observation.
Unfortunately the results were quite disappointing. Minimizing the RMSE without imposing any boundary led to overfitting which in the end caused a poor prediction performance. Imposing boundaries on the minimization of the RMSE reduced overfitting and increased the prediction performance but the results were still unsatisfying.
A better approach for selecting hyperparameters and decreasing overfitting would have been the following: use different proportions for the time series split and find the hyperparameters of the model through rolling-window cross validation.

Multivariate LSTM

Considering the serial nature of the data we tried applying an LSTM model to our dataset. We reshaped the data as its transpose, inverting rows and columns, then we split the dataset in short sequences on which we trained our model. Due to the way the LSTM models work we trained it directly on the test set. We tried different LSTM structures and found a stacked bidirectional model to give the best performance, however the RMSE was average and not particularly good, this was probably influenced by the lack of hyperparameter cross-validation due to the high computational cost.

Vanilla LSTM

This particular version seemed like a right choice for analysing the data series. Even though we achieved some initial successes on the first trials getting an MSE of 0.15, we couldn't optimize the parameters due to the extremely high computational cost. The mode with epochs > 200 needed approximately an hour to parse the data and the cross validation wasn't feasible. Therefore, we abandoned the idea.

Optimizing our Final Model

Through the comparison, it can be deduced that Random Forest has the highest accuracy but also has a high overfitting. Therefore, we have chosen MLP Neural Network, since it is the best-suited model to predict the output in terms of accuracy and robustness. The results present a very robust version of the model, while also offering high accuracy. To reduce overfitting gridsearch was used by following a funnel approach to find a more robust model. Starting with huge differences between the selected hyperparameters allows first to locate maxima and then narrow the optimal values in an iterative process. We found out that the most efficient and accurate model was using 'tanh' as our activation function, the solver was 'lbfgs', alpha was set to 1.5 with a 'hidden layer size' at 64 and a constant learning rate. Probably better results would be achieved with higher number of layers, however because of the huge computational cost we couldn't run bigger trials. Yet, other hyperparameters of Neural Networks led to higher accuracy, but lower robustness. To test the stability and generalizability of the models, we tried to further improve the model by changing the sampling method and removing randomness. In order to completely remove setting a starting number to generate a sequence of random numbers, random_state was deleted and the model was applied again. In this case, the test accuracies improved a little bit, but the train accuracy got worse.