THIS IS NOT NORMAL

Detecting Anomalous Events in Time Series Data

Introduction

The global vehicle market is in a fluid state, as it constantly undergoes disruptive changes resulting from global conflicts, national regulations and technological shifts. To tackle with this instability, BMW intends to improve its current time series prediction algorithm. This study aims to identify events altering the course of BMW's sales/ registration forecasts from one year to another by developing a time series anomaly detection algorithm and derive a generalizable nomenclature classifying such events. To recognize patterns in monthly global vehicle registrations data sequences, a Long Short-Term Memory (LSTM) model, which is a form of recurrent neural networks (RNNs), is employed.

The Repository

In the repository you will find here, scientific Python programs have been implemented to implement the project steps.

It consists of the following parts:

/data : This is where the input data and output data is stored as .csv file. Output data is the predictions and true value specified country and brand with year.

/figures : A place to register your figures. This can be the average number of registrations by country, a yearly display of estimated and actual values found by country and brand, and more.

/tasks : The section where the data is analyzed, the model is built and predictions are made, in short, the targets are tried to be completed.

- 1.0.Task.DataUnderstanding.ipynb loads the data, plays the data and performs some data analysis and visualizations.

- 1.1.Task.Analyzing.ipynb implements the functions to find all-time highs, lifecycle highs, and identify offer interruptions in the dataset.
- 1.2.Task.ModelBuilding(LSTM).ipynb is where the LSTM model is set up and where anomalies are found and visualized. The detected anomalies are returned as a dictionary with the index of the anomalies as keys and the corresponding timestamps as values.
- 1.3.Task.HyperparameterTuning.ipynb performs hyperparameter tuning using RandomizedSearchCV to find the best hyperparameters for our LSTM-based time series prediction model.
- 2.0.Bonus1.Nomenclature&Regularization.ipynb implements the anomaly detection function using LSTM in Python. The function then categorizes and labels the anomalies based on the defined nomenclature, such as magnitude, trend, seasonality, spikes and drops, and persistence. It prints the categorized anomalies along with their corresponding indices and dates.
- 2.1.Bonus1.WithHyperparameters demonstrates the anomaly detection process by applying it to with different hyperparameters and thresholds which are found bt HyperparameterTuning method.
- 3.0.Bonus2.Predictions does the prediction using the LSTM model for car registrations according to country and brand.The Mean Squared Error (MSE) is calculated to evaluate the model's performance, and the predictions were saved to a CSV file. It also creats a plot to visualize the true values and predictions.

/src : The most important directory: the sources root. It contains multiple .py files:

- utils.py stores some utility functions you can call in other files.
- data_preparation.py contains a few methods to pre-process the input dataset.
- constants.py declares all constants in UPPERCASE.
- hyperparameter.py does the hyperparameter tuning which is used for our LSTM model.
- Finally, anomaly_detection.py is where you can plot the anomalies.

main.py is left open except for a few methods. The reason for not adding anything new to main.py is that each operation is performed sequentially by the. ipynb files in the tasks folder. If functions are to be executed in main, they can be implemented by calling them.

In the repo, main consists of a few methods Mr. Simon wrote.

get_data() will read the .csv file and do the split by underscores into several feature columns.

get_filtered_timeseries() takes on some basic formatting and filters to one or more values per feature column.

def main():

df_prep = get_data()

timeseries = get_filtered_timeseries(df_prep, 'Deutschland', 'BMW', 'Limousine', '3er')

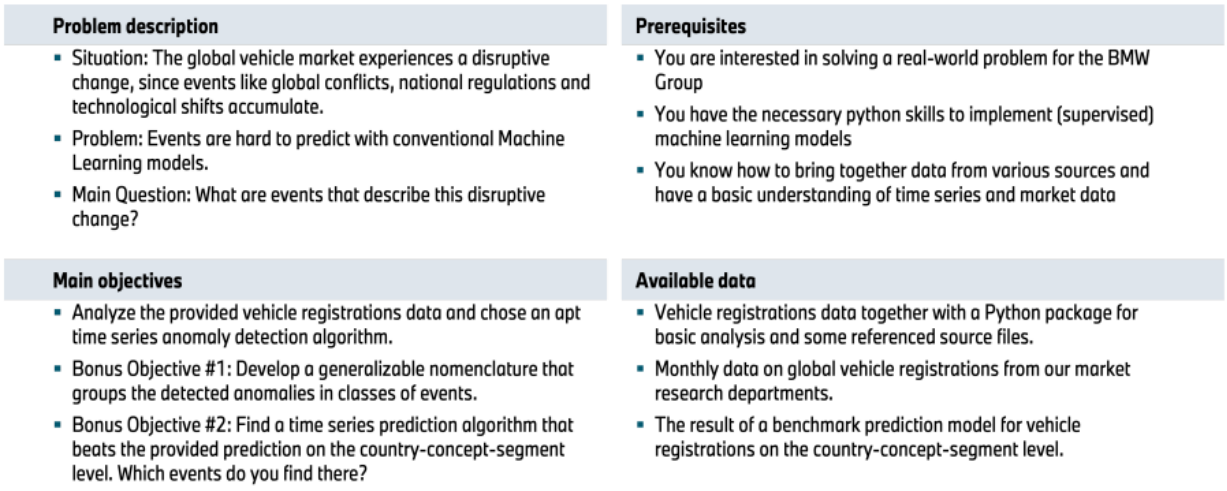
plot_sample()

The Dataset

- The column c_SRC contanis the respective time series entity - registrations stands for historical vehicle registrations.
- The column c_SMP_KEY is a unique identifier and can be split up by underscores using split_by_underscores(column)
- The column c_DATE contains the timestamp for the timeseries data in YYYYMM format.
- The column c_VALUE contains the registration value for the respective month.

The Objectives

Your main objective is to analyze the provided vehicle registrations data and choose an apt time series anomaly detection algorithm.



Taks 1.1: Analyze the time series data

It is important to understand how vehicle lifecycles evolve. Try to find registration all time highs, lifecycle highs and offer interruptions.

Step 1: Find All-Time Highs

In this step, we are finding the records with the highest values for the column 'c_VALUE' in the given DataFrame df. These records represent the all-time highs for car registrations. The function find_all_time_high takes the DataFrame df as input and returns the rows with all-time high values.

Step 2: Find Lifecycle Highs

Here, we aim to find the records with the highest values for the column 'c_VALUE' for each combination of 'c_FEATURE_1', 'c_FEATURE_2', 'c_FEATURE_3', and 'c_FEATURE_4' in the given DataFrame df. The function find_lifecycle_highs takes df, c_FEATURE_1, and lifecycle_threshold as input. c_FEATURE_1 is the filter condition to identify rows with the desired value in the 'c_FEATURE_1' column, and lifecycle_threshold is a value used to filter the records that have values higher than the threshold.

Step 3: Identify Offer Interruptions

In this step, we are identifying the records with the lowest values for the column 'c_VALUE' for each combination of 'c_FEATURE_1', 'c_FEATURE_2', 'c_FEATURE_3', and 'c_FEATURE_4' in the given DataFrame df. These records represent the offer interruptions, where the car registrations experienced a significant drop in value. The function identify_offer_interruptions takes df, c_FEATURE_1, and interruption_threshold as input. c_FEATURE_1 is the filter condition to identify rows with the desired value in the 'c_FEATURE_1' column, and interruption_threshold is a value used to filter the records that have values lower than the threshold.

Step 4: Develop a Dynamic Method

In this step, we combine the results from the previous steps to identify significant events in the car registrations data. The function identify_events takes df, c_FEATURE_1, lifecycle_threshold, and interruption_threshold as inputs. It first calls the functions from steps 1, 2, and 3 to obtain the all-time highs, lifecycle highs, and offer interruptions, respectively. Then, it concatenates these records together and removes any duplicate rows to get a comprehensive list of significant events in the data.

Task 1.2: Detect anomalies

In general, the detect_anomalies function is built to be effective in detecting anomalies in time series data using an LSTM model. It follows a systematic approach by preprocessing the data, training the model, and doing predictions to identify anomalies.

The function starts by calling the get_data() function to retrieve the data. The function then calls the get_filtered_timeseries function with the provided c_FEATURE_1, c_FEATURE_2, c_FEATURE_3, and c_FEATURE_4. This function filters the DataFrame to extract the time series data corresponding to the specified feature values.

Next, the data is normalized using MinMaxScaler from sklearn.preprocessing. Normalization scales the data to a range between 0 and 1, making it easier for the neural network to learn. The prepare_data function is used to create sequences of data and corresponding labels for the LSTM model. It takes the normalized time series data and the lookback parameter as input. It creates sequences of length lookback from the time series data and uses the next data point as the label.

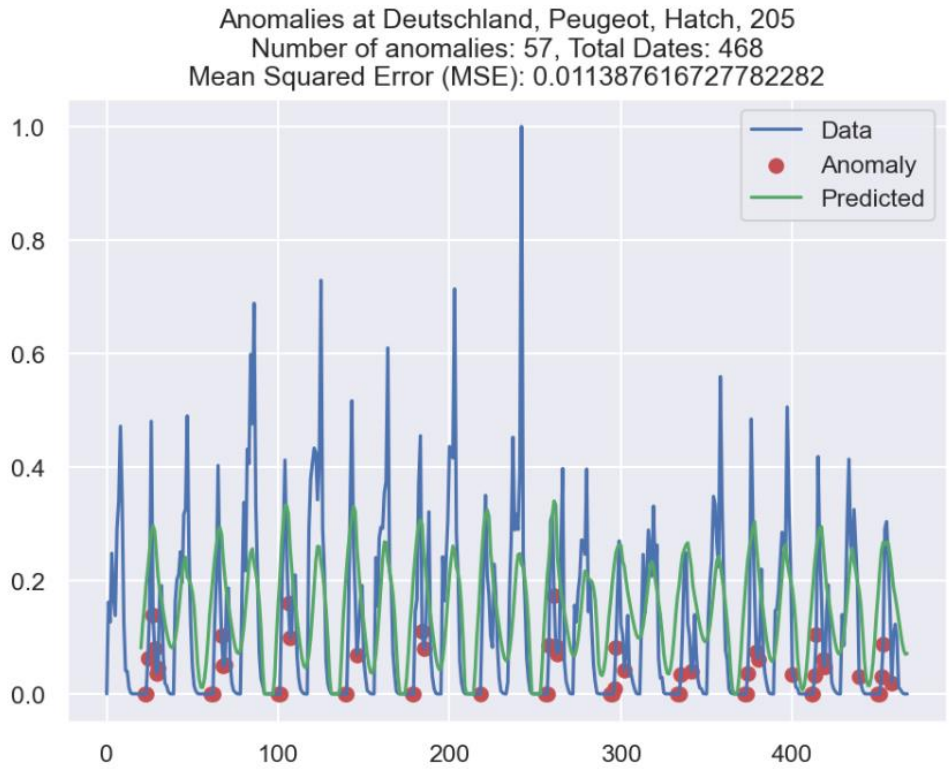
The data is split into training and testing sets. 80% of the data is used for training (X_train and y_train), and the remaining 20% is used for testing (X_test and y_test).

The function builds an LSTM (Long Short-Term Memory) model using Keras. The model consists of one LSTM layer with 64 units and a dense layer with one unit (for regression). The model uses the 'adam' optimizer and 'mean_squared_error' as the loss function. The LSTM model is trained on the training data (X_train and y_train) using the model.fit function..

Once the model is trained, it makes predictions on the entire dataset X, which includes both the training and testing data. It predicts the next data point based on the lookback sequence.

Anomalies are detected by comparing the predicted data with the actual data. If the difference between the predicted value and the actual value exceeds the given threshold, it is considered an anomaly. The threshold is a hyperparameter that determines the sensitivity of anomaly detection.

The function plots the detected anomalies on a graph along with the original data and predicted data. Anomalies are marked with a red dot, and predicted values are plotted in green.



Finally, the function returns two dictionaries: anomalies_dict and dates_dict. anomalies_dict contains the indices of the anomalies and their corresponding dates, sorted in chronological order. dates_dict contains all the dates in the dataset along with their indices.

To summarize, the detect_anomalies function takes specific feature values as input, filters the data accordingly, prepares it for LSTM training, builds the LSTM model, and trains it on the data. It then uses the trained model to detect anomalies in the data and visualizes the results. The function returns the detected anomalies and their respective dates as dictionaries.

Hyperparameter Tuning

We have performed hyperparameter tuning using RandomizedSearchCV to find the best hyperparameters for your LSTM-based time series prediction model. The hyperparameter tuning process was conducted for different combinations of features and resulted in different sets of hyperparameters along with their corresponding mean squared error (mse_score) values. Here are the results:

Here are some cases which we are doing hyperparameter tuning for some features:

1. 'Belgien', 'MINI', 'Cabrio', and ' ':

Best Hyperparameters: {'threshold': 0.1, 'lookback': 20, 'epochs': 20, 'batch_size': 32} Best mse_score: 0.046489269869879335

2. 'Niederlande', 'PGO', 'Roadster', and 'unspec.':

Best Hyperparameters: {'threshold': 0.2, 'lookback': 10, 'epochs': 10, 'batch_size': 32} Best mse_score: 0.078790028069067

3. 'Belgien', 'Alfa Romeo', 'Coupe', and '4C':

Best Hyperparameters: {'threshold': 0.2, 'lookback': 20, 'epochs': 10, 'batch_size': 32} Best mse_score: 0.04456632745333974

It seems that different combinations of features have led to different optimal hyperparameters, which is expected in this context. Hyperparameter tuning helps identify the best set of hyperparameters for a given model and dataset, leading to improved model performance. With these hyperparameters, we can create the LSTM model with the best configurations to predict time series data effectively.

Bonus 1: Nomenclature

The function works like detect_anomalies described in task 1.1. In addition, the function then categorizes and labels the anomalies based on the defined nomenclature, such as magnitude, trend, seasonality, spikes and drops, and persistence. It prints the categorized anomalies along with their corresponding indices and dates.

The nomenclature for anomalies in the model refers to the labels or categories assigned to the detected anomalies based on their characteristics in detect_anomalies function. These categories provide a descriptive classification of the anomalies, allowing for a better understanding of their nature and behavior. The nomenclature used in the model includes different aspects or dimensions of anomalies:

Magnitude:

- High Magnitude: We look at the position of the anomaly index in the normalized dataset. Normalized dataset is between 0, 1. And if the value of our anomaly is greater than 0.8, we say high magnitude.
- Low Magnitude: And if the value of our anomaly is lower than 0.2, we say high magnitude.

Trend:

- Increasing Trend: Anomaly exhibiting an increasing pattern over time.
- Decreasing Trend: Anomaly exhibiting a decreasing pattern over time.
- No Change: Anomaly with no significant change in the pattern.

Seasonality:

- Seasonal Peak: Anomaly occurring during a seasonal peak period.
- Seasonal Dip: Anomaly occurring during a seasonal dip period.

Spikes and Drops:

- Spike: Anomaly characterized by a sudden and significant increase in value. The anomaly and the previous anomaly are checked and if the difference is greater than 0.2, it is a spike.
- Drop: Anomaly characterized by a sudden and significant decrease in value. If it is smaller than 0.2, it is a drop.

Persistence:

- Persistent High: Anomaly with a persistently high value over a period of time. If the average of the anomaly index and the next 5 anomaly indices is greater than 0.8, it is Persistent High.
- Persistent Low: Anomaly with a persistently low value over a period of time. If the average of the anomaly index and the next 5 anomaly indices is greater than 0.2, it is Persistent Low.

```
Anomalies: 8
Dates: 24
Categorized Anomalies:
Anomaly at index 10 on date 2004-06-01 00:00:00: Low Magnitude + Decreasing Trend + Seasonal Peak + Drop
Anomaly at index 12 on date 2004-07-01 00:00:00: Low Magnitude + Decreasing Trend + Drop
Anomaly at index 13 on date 2005-07-01 00:00:00: Low Magnitude + No Change
Anomaly at index 15 on date 2005-08-01 00:00:00: Low Magnitude + Decreasing Trend + Seasonal Dip + Drop
Anomaly at index 17 on date 2005-09-01 00:00:00: Low Magnitude + Decreasing Trend + Drop
Anomaly at index 19 on date 2005-10-01 00:00:00: Low Magnitude + Decreasing Trend
Anomaly at index 21 on date 2005-11-01 00:00:00: Low Magnitude + Decreasing Trend + Drop
Anomaly at index 23 on date 2005-12-01 00:00:00: Low Magnitude + Decreasing Trend + Seasonal Peak + Drop + Persistent Low
```

Bonus 2: Predictions

The get_data_byGrouping function is defined to read the time series data from a CSV file using the read_csv function and filter it based on the specified country and brand. The data is then prepared by converting the date column to a datetime format and setting it as the index.

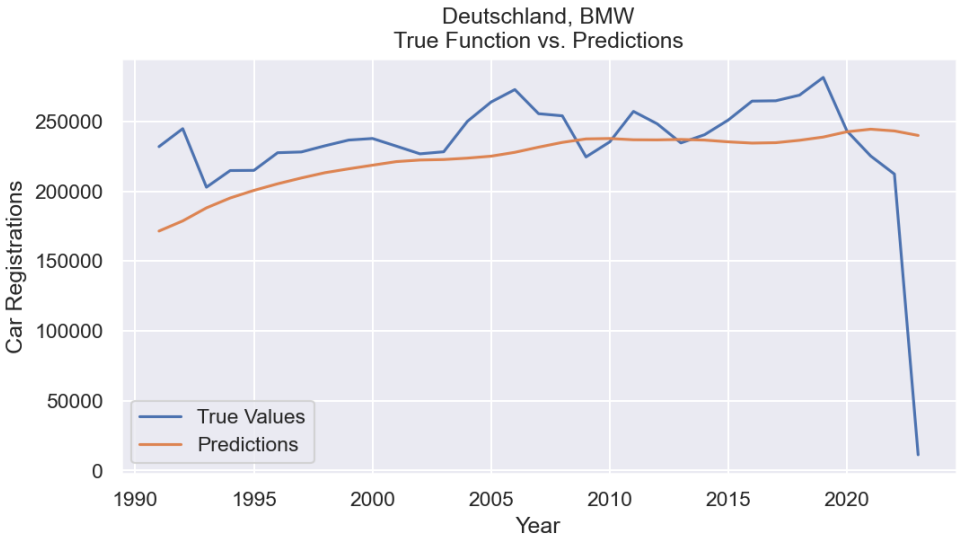
The data is normalized using the MinMaxScaler from sklearn.preprocessing. To prepare the data for training the LSTM model, it is converted into input-output pairs. A fixed number of previous years' car registrations is used as input to predict the car registrations for the following year. The number of previous years used as input is specified by the lookback variable. The input data (X) is a 3D array where each row represents a time window of lookback years, and the output data (y) is a 1D array containing the car registrations for the corresponding year.

The LSTM model is built using the Keras Sequential API. It consists of an LSTM layer with 128 units and a hyperbolic tangent (tanh) activation function. Additionally, a dropout layer with a dropout rate of 0.2 is added to prevent overfitting. The output layer is a Dense layer with 1 unit and a rectified linear unit (ReLU) activation function. The model is compiled with the mean squared error (MSE) loss function and the Adam optimizer. The model is then trained on the input data (X) and output data (y) for 50 epochs with a batch size of 64.

The trained LSTM model is used to make predictions for the available data (not used in training). The predictions are then inverse-transformed using the scaler to get the actual car registrations.The predictions are stored in a DataFrame with the corresponding years.

The mean squared error (MSE) is calculated to evaluate the performance of the LSTM model on the available data.

The true car registrations and predicted car registrations are plotted over the years using matplotlib. The plot is saved as an image named "Predictions-{Country}-{Brand}.png".

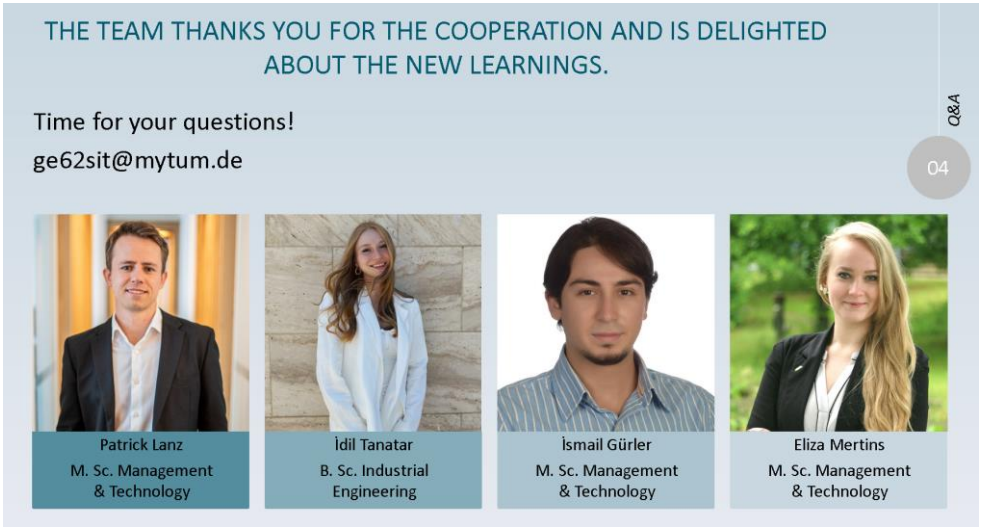


The rounded predictions and true values are saved to a CSV file named "Predictions_{Country}-{Brand}.csv".

This entire process helps in building and training an LSTM model to predict future car registrations for the specified country ("Deutschland") and brand ("BMW") based on historical data. The predictions are evaluated using MSE, and the results are visualized and saved for further analysis and reporting.

Questions

If you really need an urgent response, feel free to contact us directly.



Acknowledgments

We are provided with real-world time series data subject to secrecy from the BMW Group for this project.