A Model-Driven Framework for Domain-Specific Adaptation of Time Series Forecasting Pipeline

Neeraj Katiyar

Department of Electrical and Computer Engineering

McGill University, Montreal

July, 2023

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of

Master of Science

Abstract

Time series forecasting (TSF), predicting future values based on historical data, has emerged as a vital tool in various fields, enabling intelligent systems and data-driven decision-making. However, the repetitive nature of implementing similar components and procedures across multiple machine learning (ML) projects often leads to inefficiencies and reduced scalability, resulting in a lack of consistency. To address this challenge, this Master's thesis focuses on designing a model-based software architecture defined using the Unified Modeling Language (UML) to promote reusability and adaptability in time series forecasting pipelines. The effectiveness of the framework is examined through two independent research projects: (1) Predicting early undergraduate students' performance and (2) Thermo-acoustic instability prediction in gas turbine combustion. The first case study aims to predict the performance of early undergraduate students, enabling educational institutions to provide appropriate interventions and support. The second case study predicts thermo-acoustic instability in gas turbine combustion, focusing on the crucial role of time series forecasting in improving overall performance and efficiency of gas turbines. The research questions of the thesis compare time-series-based forecasting with traditional engineering models to validate its effectiveness. Together, the framework and case studies contribute to enhancing the efficiency, scalability, and adaptability of TSF pipelines and validating the effectiveness of TSF techniques in predicting student performance in a course and thermoacoustic instability prediction in combustion system of gas turbines.

Abrégé

La prévision des séries chronologiques (TSF) est devenue un outil essentiel dans divers domaines, permettant des systèmes intelligents et une prise de décision basée sur les données. Cependant, la nature répétitive de la mise en œuvre de composants et de procédures similaires dans plusieurs projets d'apprentissage automatique (ML) entraîne souvent des inefficacités et une évolutivité réduite, ce qui entraîne un manque de cohérence. Pour relever ce défi, cette thèse se concentre sur la conception d'une architecture logicielle basée sur un modèle définie à l'aide du langage de modélisation unifié (UML) pour promouvoir la réutilisabilité et l'adaptabilité dans les pipelines de prévision de séries chronologiques. L'efficacité du cadre est examinée à travers deux projets de recherche indépendants : (1) Prédiction des performances de premier cycle et (2) Prédiction de l'instabilité thermo-acoustique dans la combustion des turbines à gaz. La première étude de cas vise à prédire la performance des premiers étudiants de premier cycle, permettant aux établissements d'enseignement de fournir des interventions et un soutien appropriés. La deuxième étude de cas prédit l'instabilité thermo-acoustique dans la combustion des turbines à gaz, en se concentrant sur le rôle crucial de la prévision des séries temporelles dans l'amélioration des performances globales des turbines à gaz. Les questions de recherche de la thèse comparent la prévision basée sur des séries chronologiques avec des modèles d'ingénierie traditionnels pour valider son efficacité. Ensemble, le cadre et les études de cas contribuent à améliorer l'efficacité, l'évolutivité et l'adaptabilité des pipelines TSF et à valider l'efficacité des techniques TSF pour prédire les performances des étudiants dans un cours et la prédiction de l'instabilité thermoacoustique dans le système de combustion des turbines à gaz.

Acknowledgements

As I approach the end of my MSc studies, I would like to acknowledge and reflect on the journey that involved motivation, collaboration and guidance from inspiring individuals.

Foremost, I am immensely grateful to my guru (supervisor), Daniel Varro, for believing in my abilities and creating an environment of independence where ideas could flourish. His guidance on the subject matter and funding made this journey remarkably smooth. I also extend my thanks to my Siemens supervisor, Martin Staniszewski, whose regular support and feedback helped me to optimize my work. My lab mates Percy, Sebestian, Aren, and Fozail have always been a great support in answering all of my questions.

I am thankful to the OSE director, Marcy, and my co-supervisor, Armin Yazdani for welcoming me into the incredible OSE team and providing the opportunity to delve into a valuable research case study. Your humility and patience have been an inspiration to me. I would also like to extend my heartfelt thanks to the entire SciLearn Team (Kira, Valerie, and others) for their continuous support.

I am truly grateful for my mentors and family, Rijul and Ruchika who have consistently been a source of unwavering support and motivation. Their guidance and encouragement have pushed me to strive for the best. They are my eternal inspirations. Additionally, a special thanks to Katyayani for providing peer review assistance until the very end.

I am appreciative to many friends in Canada who always supported and guided me whenever required. Each one of you have made a positive impact on me and my work but I am particularly grateful to Niranjan, and Kireet for unwavering support throughout.

This thesis would not have been possible without the support of my family and friends. As a first-generation scholar, I believe this accomplishment is a result of my family's prayers and blessings from oceans away. My parents (Sunita and Satish) sacrificed a lot to ensure I had this opportunity to pursue my interests.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Motivation

Time series forecasting (TSF) is emerging as an essential tool from machine learning (ML) in different fields, allowing the development of intelligent systems and data-driven decision-making. It plays a crucial role in predicting future values based on historical data patterns. It involves analyzing sequential data points collected over time to uncover underlying trends, seasonality, and patterns [7]. TSF has gained significant importance across various domains and industries due to its ability to provide valuable insights for decision-making.

Some common and popular application areas of TSF include the finance sector (stock market prediction, portfolio management, and risk assessment) [8], the retail sector (demand forecasting helps optimize inventory levels and improve supply chain efficiency), weather, and energy production. Weather forecasting relies on time series analysis to predict temperature, rainfall, and other meteorological variables [9]. In the energy production field it enables companies to optimize energy generation, distribution, and resource planning. These examples highlight the wide-ranging applications of TSF and its importance in driving operational efficiency, productivity, and scalability.

As companies and practitioners often engage in multiple TSF projects, a common challenge emerges: different teams repeatedly develop the similar components which leads to repetition. This redundancy leads to inefficiency and reduced scalability in the development

process. According to Chui, Manyika, and Bughin [10], organizations often face difficulties in reusing existing models, algorithms, and workflows across projects while adopting to large scale machine learning projects. This lack of reusability hampers productivity and hinders the ability to efficiently leverage prior knowledge and expertise.

1.2 Research Questions, Objectives and Contributions

To overcome the aforementioned challenges, there is a need for a standardized and scalable approach that promotes component reusability, reduces redundant efforts, and enhances the adaptability of TSF projects, allowing for more efficient development and deployment of time series analysis pipelines. Figure 1.1 show the overall scope of the thesis. This thesis investigates three major research questions related to challenges and implementation of TSF techniques as discussed below:

1.2.1 Enhancing adaptability and reusability in TSF pipelines

Implementing a TSF pipeline involves various stages such as data aggregation, cleaning, processing, transformation, model selection, training and evaluation. Each stage has specific methods to deal with the use case-specific requirements. Hence when working on two or more TSF projects, one often repeats many operations, which reduces the overall productivity and efficiency of the developer. A few existing machine learning automation tools (abbreviated as AutoML) offer to streamline and automate the overall ML pipeline to make it available for even non-machine learning experts. One such example is Azure ML, which enables users to feed data, select the model, and train it without dealing with technical aspects of the ML pipeline [11]. However, these solutions do not facilitate domain-specific adaptation of different use cases. They are efficient only for general classification (prediction of categories) and regression (prediction of continuous values) problems, where there is no temporal dependency analysis (time-dependent analysis) involved, which makes TSF pipelines more complex than classification and regression ML problems. Hence, there is a scope for enhancing such automation tools to include more complex ML techniques, such as

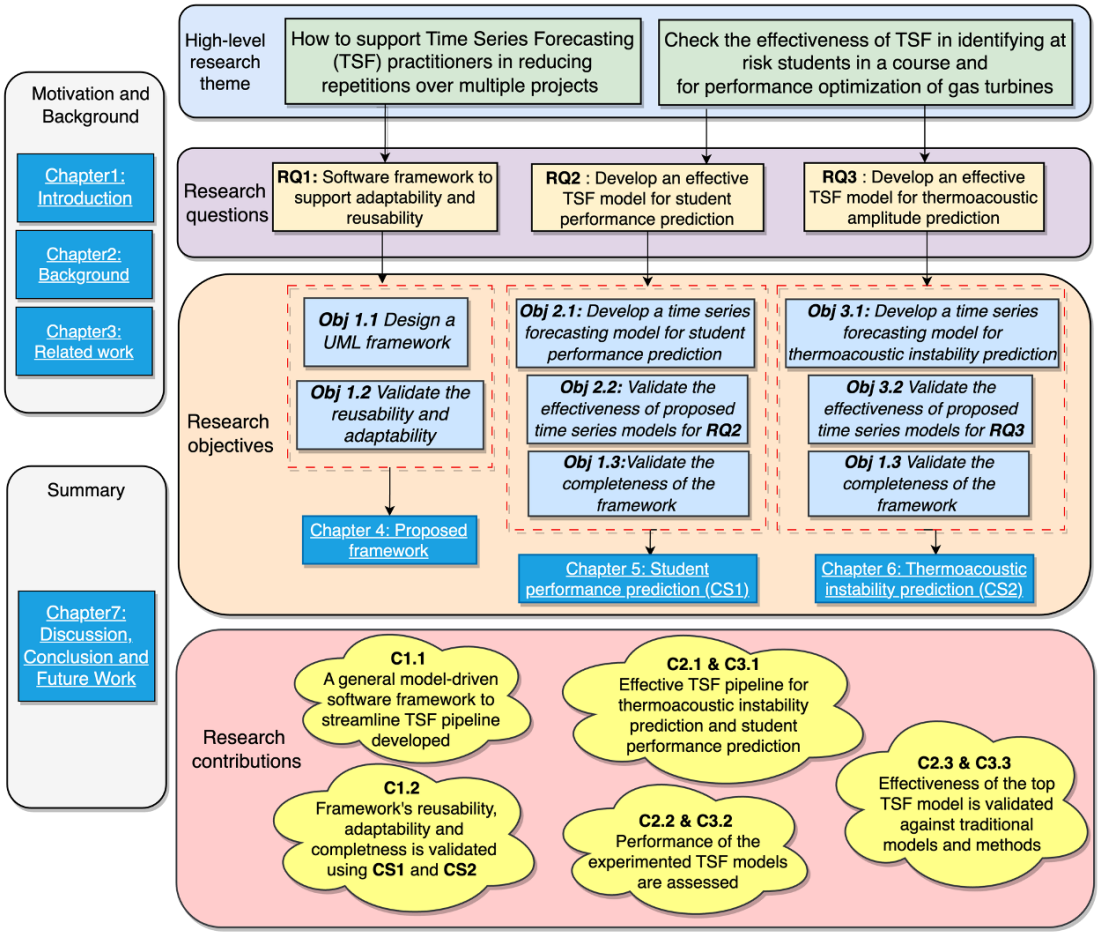


Figure 1.1: Scope of the thesis

TS forecasting, which streamlines the processes and allows the developer to reuse and adjust the methods based on specific requirement.

Hence the thesis aims to overcome the above challenge by designing a software framework where the components can be adapted in other domains and facilitates common component reuse, enhancing the overall efficiency of implementing forecasting projects.

Research Question

RQ1

Can we design a model-based software architecture to support component adaptability and reuseability for end-to-end time series forecasting pipelines?

Objectives

To address the RQ1, the objective (obj) of the thesis are defined as follows:

Obj 1.1: Design a general software framework for developing TSF pipelines.

Obj 1.2: Validate the framework's adaptability, reusability and completeness.

To achieve Obj 1.2, the TSF framework is examplified and validated using two independent research projects as case studies (CS): (1) Predicting student performance and (2) Thermoacoustic (TA) instability prediction in the combustion of gas turbines. These case studies, along with the software architecture, and its validation contribute to the overall objectives of this thesis. The next two subsections introduce these two case studies along with their respective RQs and objectives.

Contributions

By addressing RQ1 and achieving its associated objectives, the following are the contributions in this thesis:

C1.1 We designed a general model-based software framework to streamline implementation of TSF pipelines.

C1.2 We validated the framework's adaptability, reusability, and completeness using CS1 and CS2, the results demonstrated its effectiveness in addressing the existing challenges of TSF projects.

1.2.2 Enhancing student performance prediction

The first case study is related to a collaborative research project funded by DGDM Family Foundation. In this case study we focus on predicting the student performance in a university level undergraduate course.

Introductory STEM courses are known to be challenging with a higher dropout, withdrawal, and failure (DWF) rate for undergraduates transitioning from secondary to university-level education. To address these challenges, the Office of Science Education (OSE) at McGill University has implemented various evidence-based initiatives, including SciLearn (Science of Learning), aimed at supporting all students and fostering self-regulated (students becoming self aware on learning how to learn) learning. However, the traditional approach to predict student performance for STEM courses are time-consuming and ineffective.

In light of this, OSE is exploring data-driven methodologies to accurately predict student performance and provide timely support for at-risk students, such as those likely to drop out or fail the course. Additionally, data can be shared with students so that they can improve metacognition, ability to regulate one's own thinking processes, and become self-regulated learners, ability to direct and control one's own learning process.

Research Question

RQ2

How to develop effective TSF techniques for predicting student performance?

Therefore, the highlighted RQ2 is guiding the work of this study. By building, and implementing TSF models, this applied research aims to leverage the capabilities of ML to enhance the effectiveness of predicting student performance and ultimately improve the support provided to STEM students.

Objectives

To address the RQ2, the objective (obj) of the thesis are defined as follows:

Obj 2.1: Develop a TSF pipeline to predict student performance.

Obj 2.2: Validate the performance and effectiveness of the TSF model.

To achieve objectives 2.1 and 2.2, we investigated and developed ML and deep learning TSF models, trained using the training dataset and evaluated the performance using Mean Absolute Error and Root Mean Squared Error (RMSE) metrics. Further, to validate the effectiveness of the top performing model in the context of SciLearn program, it has been compared with the existing Learning Strategies Inventory (LSI) based methodology. The results of the work has been presented and published at [12].

Contributions

For RQ2, following are the contributions of the thesis:

C2.1 We designed an effective TSF pipeline specifically to forecast student performance in a course.

C2.2 We validated the performance of the TSF model, showcased its potential to accurately predict student performance and improve support provided to STEM students.

C2.3 We compared our methodology with traditional processes, highlighting the superiority of the proposed TSF model over existing approaches.

1.2.3 Enhancing TA instability prediction

The second case study is related to a five-year collaborative research project funded by Natural Sciences and Engineering Research Council of Canada (NSERC) and Siemens Energy, Montreal. In this case study, we focus on the prediction of thermo-acoustic instability in the combustion of *aeroderivative gas turbines (AGT)*. Combustion is an essential and critical part of a gas turbine, playing a fundamental role in the overall performance and efficiency of the system [13]. In an AGT, combustion refers to the process of burning fuel (such as natural gas, diesel, or aviation fuel) in the presence of compressed air to produce high-temperature (thermo) and high-pressure (acoustic) gases. These gases expand rapidly, driving the turbine blades and generating power.

Sometimes this expansion results in self-sustained, high-amplitude pressure oscillations within a combustion system which referred as Thermoacoustic (TA) instability, which contributes to overall combustion noise and downgrade in combustion efficiency. Hence prediction of these instabilities is of importance, as it directly affects the performance, safety, and efficiency of gas turbine systems.

Siemens Energy has significant business interest in AGT, offering numerous AGT models with a wide range of electrical power generation capacity. Hence we built a prediction model to forecast acoustic amplitudes in combustion using TSF techniques which is compared against the baseline standard performance of the existing Support Vector Machine (SVM) model. This would further help the domain experts at Siemens Energy to forecast instability, enabling timely interventions and preventive measures.

Research Question

RQ3

How to develop effective time series forecasting techniques for predicting thermoacoustic instabilities in aeroderivative gas turbines?

The highlighted RQ3 is the foundation of this study. Similar to case study (CS) 1, this applied research aims to leverage the capabilities of TSF models to enhance the effectiveness of predicting thermoacoustic instability prediction in AGT.

Objectives

To address the RQ3, the objective (obj) of the thesis are defined as follows:

Obj 3.1: Develop a TSF pipeline to predict TA instability in AGT.

Obj 3.2: Validate the performance and effectiveness of the proposed TSF model.

Similar to CS1, to achieve objectives 3.1 and 3.2, we investigated and developed variants of deep learning TSF models, trained using the training dataset and evaluated the performance using Mean Absolute Error and Root Mean Squared Error (RMSE) metrics. Further, to validate the effectiveness of the top performing model in the context of Aeroderivative Gas Turbines (AGT), we compared it with the industry standard baseline ML model support vector machine (SVM).

Contributions

In the process to achieve the objectives of this section, the following contributions are made:

C3.1 We designed an effective TSF pipeline specifically for forecasting acoustic amplitudes in AGT combustion systems.

C3.2 We validated the performance of the TSF model, showcased its potential to accurately predict acoustic amplitudes in AGT in order to accurately predict TA instabilities in AGT.

C3.3 We validated the effectiveness of the proposed TSF model by comparing against the industry standard baseline SVM model. The results demonstrate the superiority of the TSF model, enabling timely interventions and preventive measures for AGT systems, thereby improving their overall performance and efficiency.

1.3 Thesis Organization

Figure 1.1 provides an overview of the organization of the thesis.

- **Chapter 2 and Chapter 3:** Before delving deep into the framework design and case studies implementation, we first provide the reader with background information and define key terms that we will use throughout this thesis. Also, in order to situate this thesis with respect to related research, we present a survey of research on the use

of UML class diagrams in designing the effective frameworks, implementation of TSF techniques in predicting thermo-acoustic instabilities and student performance.

Next, we shift our focus to the main body of the thesis as briefly explained next.

- **Chapter 4:** This chapter presents the proposed framework and its advantages. Due to the multidisciplinary nature of the case studies and proposed framework, the validation of the framework is spanned across Chapter 4, Chapter 6 and Chapter 5.

- **Chapter 5** This chapter addresses RQ2 and discusses the experimental setup, initial data preparation, methodology, model evaluation sections. The results are validated with the traditional method followed at OSE for Student performance prediction. Followed by completeness validation of the proposed framework.

- **Chapter 6:** This chapter focuses on RQ3 and discusses the experimental setup, initial data preparation, methodology, model evaluation and results with respect to the implementation of time series predictive modeling for TA instability prediction at Siemens. Towards the end it also discusses the coverage from the proposed framework.

- **Chapter 7:** This chapter presents summarizes key accomplishments of the thesis and presents the future work.

Chapter 2

Background

In this section, we describe the foundations for the relevant research areas of this thesis. As shown in Figure 2.1, this thesis connects concepts, techniques, and tools from mainly three areas. The *application area* of this thesis is TSF pipelines, specifically the framework is designed to streamline the TSF pipelines, promote resuability and adaptability when implemented in different domains.

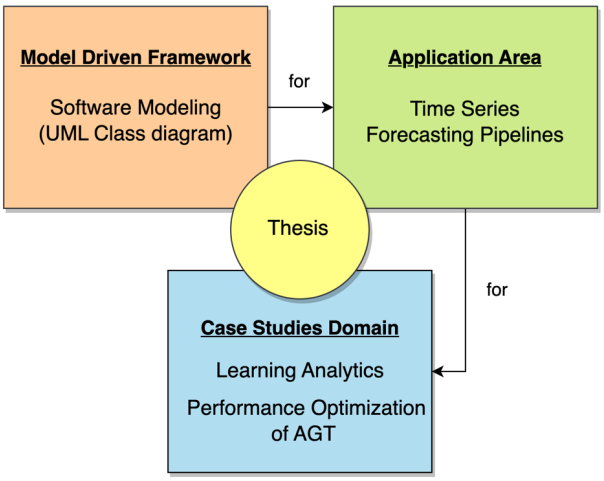


Figure 2.1: Main research areas related to this thesis

In this thesis we use *software modeling*, specifically UML class diagrams, to design a model driven framework. Finally, the framework is validated by implementing TSF techniques in two separate *domains*, learning analytics, in predicting students performance and performance optimization of AGT, to be specific in predicting TA instability in AGT.

The remainder of this section is organized as follows: Section 2.1 describes the core concepts in software modeling, Section 2.2 describes some important terms and techniques related to TSF, and then Section 2.3.1 and Section 2.4 introduces the core concepts related to the two case studies: student performance and TA in AGT respectively.

2.1 Software Modeling

In this section, we first describe models in general and their role in software engineering. Next, we discuss the most important terms and concepts for class diagrams (domain models) that are relevant to this thesis.

2.1.1 Models

The *model* term is derived from the Latin word modulus, which means measure, rule, or pattern. The usage of models can be traced back to early civilizations such as Ancient Egypt, Greece, and Rome, where models were used to demonstrate visionary plans in art and architecture [14].

A model represents a selective representation of some system that captures essential and relevant properties for a given set of concerns precisely. In addition, the purpose of models is to reduce complexity to the human scale which is suitable for reasoning. The characteristics of models include abstraction, understandability, accuracy, prediction, and low cost. Combemale et al. introduce the models and data (MODA) framework that provides the foundations for identifying the various models and their respective roles – *descriptive*, *prescriptive*, and *predictive* [15]. A model plays a descriptive role if it documents some current or past aspect of the system under consideration for communicating understanding and design intent to others. Moreover, a model plays a prescriptive role if it provides a description of the envisioned system, which acts as a blueprint to guide system implementation. Finally, a model plays a predictive role if it facilitates the prediction of information that is either not possible or difficult to measure. This information is further processed to construct knowledge. This knowledge is then used to enable decision-making and perform trade-off analysis.

2.1.2 UML Class Diagrams (Domain Models)

One of the most widely used models is the domain model [16], which is also the focus of this thesis. Domain models capture the structural and configurable aspects of a system as a part of the domain modelling activity. In domain modelling, modellers transform a problem description that expresses requirements in natural language, into domain concepts in the form of classes, attributes, relationships, and association cardinalities. Domain models can be captured in various notations including i∗ [17], EMF models [18] or UML class diagrams. In the thesis, we assume that domain models are captured using the Unified Modelling Language (UML), which is considered the de facto standard language for software specification and design [19]. UML was accepted as a standard in 1997 by OMG and is still being developed further [20]. UML defines different types of diagrams such as static diagrams (e.g., class and object diagrams), behavior diagrams (e.g., state-chart diagrams), and implementation diagrams (e.g., deployment diagrams). The focus of this thesis is on static diagrams hence we will discuss more on static diagrams.

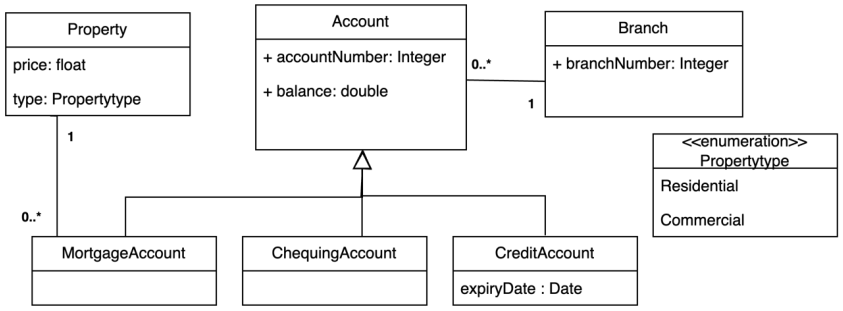


Figure 2.2: An example of Class Diagram capturing a domain model [1]

Based on UML specifcation for class diagrams, Figure 2.2 illustrates a subset of symbols which are commonly used in a domain model (class diagram). First, classes which include "Branch", "Account", "MortgageAccount", "CheckingAccount", "CreditCardAccount", and "Property", are units of data abstraction and represent the types of data themselves. Second, association relationships show how the instances of classes reference instances of other classes, e.g., the association relationship between "Account" and "Branch" classes. Symbols indicating multiplicities (cardinalities) are used at each end of an association relationship.

These cardinalities indicate how many instances of the class at this end of the association relationship can be linked to an instance of the class at the other end of this association relationship. For example, the cardinality "0..*" indicates that an instance of the "Branch" class can be linked from zero to many instances of the "Account" class. Third, attributes which include "branchNumber", "accountNumber", "balance", "expiryDate", "price", and "type" represent simple data found in the respective instances of the classes. In addition, the type property of these attributes is also shown, e.g., String for "accountNumber"

Fourth, an enumeration is a finite set of named identifiers that represent the values of the enumeration. These values are called enumeration literals or items. For example, the enumeration class "PropertyType" has two enumeration items – "ResidentialProperty" and "CommercialProperty". Finally, a relationship between a subclass and an immediate superclass is called generalization where the subclass is called a specialization. A hierarchy with one or more generalizations is known as an inheritance hierarchy, a generalization hierarchy, or an is a hierarchy. For example, the relationship between "Account" superclass and "CheckingAccount" subclass is represented by generalization.

2.2 Time Series Forecasting Pipeline

Time series forecasting is fundamental for various use cases in different domains such as energy systems and economics. Creating a forecasting model for a specific use case requires an iterative and complex design process. The typical design process includes five sections (1) data cleaning, (2) data processing, (3) data transformation, (4) forecasting model, hyper-parameters selection and training, and (5) evaluation, which are commonly organized in a pipeline structure as demonstrated in Figure 2.3. We'll briefly discuss these five stages sequentially in the next section.

2.2.1 Data Cleaning

Since most forecasting methods rely on assumptions about data properties, data cleaning is of crucial importance. Data cleaning includes detection and handling of missing values,

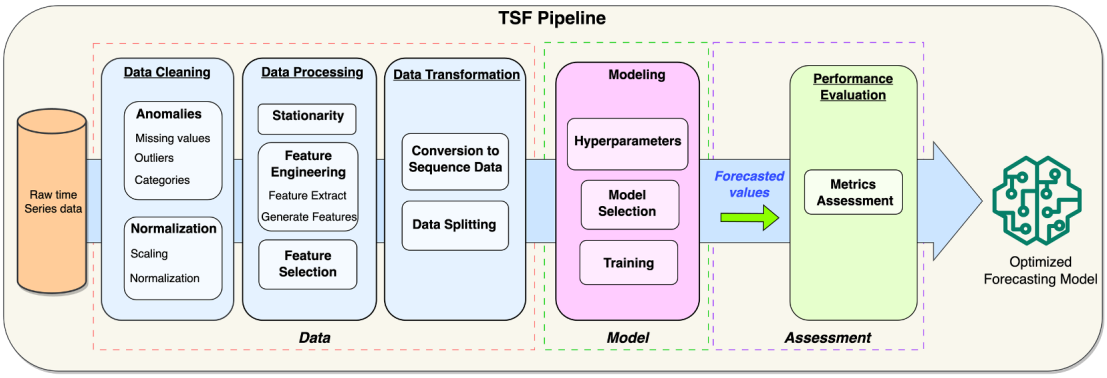


Figure 2.3: Standard TSF pipeline based on [2]

outliers and categories while normalization deals with scaling the time series data. In the following subsections, we briefly describe these anomalies and the respective solutions.

- **Missing values** Missing values generally refers to blank values in the data. There are two most adapted technique to deal with this problem, 1. **Interpolation**: In this method we replace the blank values with the previous or the next time stamp values, to signify the constant behaviour of the feature for a specific time stamp. 2. **Replace with zero:** In this method we replace the blank values with zero to signify no value capture during this time stamp. If the above two doesn't apply the simplest approach is to remove the missing values from the dataset. The solution depends on the nature of the dataset and its interpretation.

- **Outliers** refers to the observations that deviate significantly from the expected patterns or values in the data [21]. These are data points that are unusually high or low compared to the majority of the data points in the time series. There are various methods available in the literature to deal with outliers such as *statistical methods*, *windowing*, *z-score transformation*, and *smoothing*. But in our case study as the these values are very limited and has very minimal impact hence we instead decided to remove it.

- **Categories** refers to the text value based categories, most of the TSF require data input to be in numerical values for model compatibility and data standarization. Hence

mapping categories to numerical values is called *encoding* essential based on the use case requirement. There are various encoding techniques exist in the literature such as *binary encoding*, *ordinal encoding*, *labeled encoding*, and *one-hot encoding*.

Label encoding refers to assigns a unique numerical label to each category. It is suitable for ordinal categories with an inherent order. For example, assigning 0, 1, 2, and so on to categories like "low," "medium," and "high" based on their respective order. We have used this encoding technique in our case studies.

- **Normalization** in time series refers to the process of transforming the data to a common scale or range. It ensures all data points in the time series are on a comparable scale and have similar magnitudes. There are various methods available in literature which are used of normalization of data such as: *min-max*, *z-score*, and *log transformation*. For this thesis we have leveraged *min-max* scaling techniques for data normalization.

  Min-max scaling also known as feature scaling, re-scales the data to a specific range, typically between 0 and 1. It involves subtracting the minimum value from each data point and then dividing it by the range (maximum value minus the minimum value).

2.2.2 Data Processing

This stage of the TSF pipeline transform the data to stationary, if non stationary, extract features or generate new features and finally select the top features to be used further in the pipeline.

- **Stationarity**: While working with time series data often times we end up having data with trends (refers to a long-term pattern or direction in the data that shows a systematic increase or decrease over time), seasonality (systematic fluctuations in the data that occur due to factors such as calendar months, quarters, seasons, or other predictable cycles) or other time-depened patterns that changes over time. Hence this makes data non-stationary which means the statistical properties of the data

also changes over time which makes the model difficult to capture feature relationships accurately. Hence making data stationry is also one of the important steps in TSF. There are various methods available to test non-stationarity in the data such as, *Augmented Dickey-Fuller (ADF) Test*, *Phillips-Perron (PP) Test*, and *Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test*. Below we exemplify the *ADF* test.

ADF Test: The test is based on an auto-regressive model and examines the presence of a unit root in time series data. The test equation for the ADF test is as follows:

$$\Delta y_t = \alpha + \beta y_{t-1} + \gamma \Delta y_{t-1} + \delta_1 \Delta y_{t-2} + ... + \delta_k \Delta y_{t-k} + \varepsilon_t$$

where, Δy_t represents the differenced series at time t, α is the intercept term, β is the coefficient of the lagged level variable $(y_t - 1)$, γ is the coefficient of the lagged first-difference variable Δy_{t-1}, $\delta_1, \delta_2, ..., \delta_k$ are the coefficients of the additional lagged differenced variables, and ε_t is the error term.

If the ADF test statistic is smaller (more negative) than the critical value at a chosen significance level (e.g., 5%), the null hypothesis (a unit root is present, indicating non-stationarity in the data) of non-stationarity is rejected [22]. This suggests evidence in favor of stationarity in the data.

- **Feature Engineering** in time series involves transforming or creating new variables from the existing time series data to improve the performance of predictive models. It involves several methods for extracting new variables or features. Some common techniques include: *Lagging* (by shifting the time series data by a certain number of time steps), *Statistical Measures* (such as taking mean, medium, minimum, maximum or variance over specific time intervals), and *domain-specific transformation*.

- **Feature Selection** in time series data involves choosing the most relevant and informative features from a dataset to improve model performance. The most common methods for feature selection in time series are *correlation analysis* and *feature importance* as discussed below:

Correlation Analysis: This analysis involves assessing the relationship between different variables to identify patterns or dependencies. It helps in understanding the degree and direction of association between variables [23]. One of the most famous and common method for correlation analysis in time series include *Pearson correlation coefficient*: measures the linear relationship between two continuous variables. It ranges from -1 to 1, with a value close to 1 indicating a strong positive correlation, close to -1 indicating a strong negative correlation, and close to 0 indicating no linear correlation.

Feature Importance: This analysis determine the relative importance or contribution of each feature in predicting the target variable. It helps in identifying the most influential features and excluding less informative ones, thereby improving model performance. In contrast to Pearson correlation coefficient it not only considers the correlation between variables but also the predictive power of each feature in the context of the target variable. One of the tree based feature importance method is *Random Forest*. The feature importance in *Random Forest* is calculated by measuring the total reduction in impurity (measure of disorder or uncertainty in the data) achieved by splitting on a particular feature across all trees in the forest. Features that lead to higher impurity reduction are considered more important in determining the outcome or prediction.

2.2.3 Data Transformation

- **Sequence Data Conversion:** Many machine learning models, especially those designed for sequential data processing, require input data in a sequential format. By converting raw time series data into sequences, we can directly feed it into these models, making the data compatible with a wide range of time series forecasting techniques, recurrent neural networks (RNNs), Long Short-Term Memory (LSTM) networks, and other sequential models. This transformation also takes care of incorporating the temporal dependencies in the form of lags.

- **Data Splitting:** It is a crucial aspect of time series modeling. It involves dividing the sequence dataset into two subsets: training and testing sets. The training set trains the time series model, allowing it to learn patterns and relationships within the data. On the other hand, the testing set is used to assess the model's performance by evaluating its predictions on unseen data. The training and testing split aims to simulate real-world scenarios where the model encounters new, unseen examples. It helps to estimate how well the model generalizes to unseen data and provides insights into its predictive capabilities. Common data-splitting approaches include random sampling, cross validation, rolling window, stratified sampling, fixed or time-based splitting [24].

2.2.4 Modeling and Training

Model Selection

Selection of the appropriate model is another crucial aspect of time series forecasting, that best captures the underlying patterns and relationships within the data. Time series forecasting models can be categorized into three main types: univariate, multivariate, and multi-step forecasting.

1. **Univariate:** In univariate time series forecasting, the model uses only a single variable (time series) to make predictions about future values. Classical statistical methods like ARIMA (AutoRegressive Integrated Moving Average) and Exponential Smoothing fall under this category and are widely used for their simplicity and interpretability.

2. **Multivariate:** Multivariate time series forecasting involves using multiple variables, often with temporal dependencies, to predict the future values of one or more target variables. Vector Autoregressive (VAR) models and its extensions, such as VARMA (Vector Autoregressive Moving Average) and VARMAX, are commonly employed in multivariate forecasting tasks along with neural networks such as Recurrent Neural Network (RNN) and Long Short Term Memory (LSTM).

3. **Multi-step:** Multi-step time series forecasting focuses on predicting multiple future time steps ahead. This type of forecasting is particularly challenging as it involves forecasting beyond the next immediate step. Methods like Long Short-Term Memory (LSTM) networks, and other deep learning approaches are utilized to handle the complexities of multistep forecasting tasks.

Long Short Term Memory(LSTM)

LSTMs belong to a family of neural networks called recurrent neural networks (RNNs; [25]), which are used primarily for sequential data such as time series signals. Their architecture includes different gates that control the flow of gradients through their memory units which are called cells. When the gates are closed, gradients pass through a cell unchanged, alleviating the vanishing gradients problem (as error signals are backpropagated across many time steps, their gradients decay to zero [26]). Given the input x_t at the current time step t, the vector of previous hidden states h_{t-1}, and the previous cell state c_{t-1}, the series of operations of each unit is described in the equation below. where W_l and U_l are, respectively, input and recurrent weights of the corresponding layer l. Bias terms are omitted for brevity. The operations are also shown in Figure 2.4. The first type of gate, known as the forget gate,

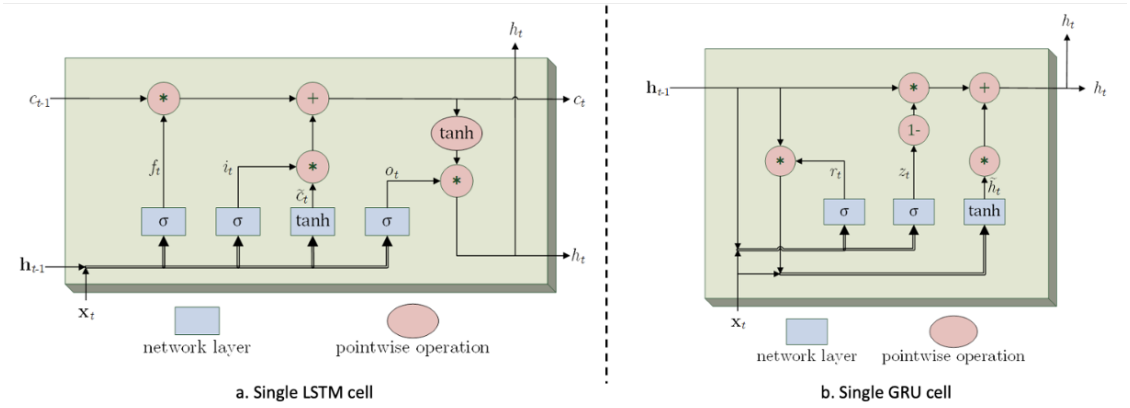


Figure 2.4: Operations involved in a single cell, a. LSTM and b. GRU

determines how much of the previous cell state is kept versus discarded (Equation 1.1a). The sigmoid function σ restricts the output f_t between 0, where the previous cell state is

discarded completely, and 1, where all the previous cell state is kept. The second type is the input gate. The output it of this gate decides which information from the input x_t to use to update the cell state (Equation 1.1b).

$$f_t = \sigma\left(pW_f \cdot x_t + U_f \cdot h_{t-1}\right), \tag{1.1a}$$

$$i_t = \sigma\left(pW_i \cdot x_t + U_i \cdot h_{t-1}\right), \tag{1.1b}$$

$$\tilde{c}_t = \tanh\left(pW_c \cdot x_t + U_c \cdot h_{t-1}\right), \tag{1.1c}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \tag{1.1d}$$

$$o_t = \sigma\left(pW_o \cdot x_t + U_o \cdot h_{t-1}\right), \tag{1.1e}$$

$$h_t = o_t \odot \tanh(c_t). \tag{1.1f}$$

A hyperbolic tangent activation restricts the input values between -1 and 1 to produce candidate values $\tilde{c}_t$ with which to update the cell state (Eq 1.1c). Given the previous cell state c_{t-1}, the new cell state c_t is calculated from the results of the forget and input gates (Eq 1.1d). The final output gate determines how to produce the new hidden state h_t from the new cell state (Eq 1.1e and 1.1f).

Gated Recurrent Unit

Cho, van Merrienboer [27] aim to improve on LSTMs with their invention of a simpler RNN hidden unit called gated recurrent unit (GRU). Figure 2.4 depicts the new process to update the hidden state. Given the input x_t at time step t, the vector of previous hidden states h_{t-1}, and weights W_l and U_l corresponding to layer l, the unit update equations are as given below.

Two new types of gates are defined, with the first being a reset gate $r \in [0, 1]$ which determines whether to forget the previously computed hidden state (Equation 6.2a). The update gate, which is the other type of gate, outputs coefficients z_t used in the convex

combination of the previous and candidate hidden states (Equations $6.2c - 6.2d$).

$$r_t = \sigma\left(pW_r \cdot x_t + U_r \cdot h_{t-1}\right), \tag{2.1a}$$

$$\tilde{h}_t = \tanh\left(pW_h \cdot x_t + U_h \cdot (p_r \odot h_{t-1})\right), \tag{2.1b}$$

$$z_t = \sigma\left(pW_z \cdot x_t + U_z \cdot h_{t-1}\right), \tag{2.1c}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t. \tag{2.1d}$$

Hyper-parameters

Hyperparameters are settings or configurations that determine the behaviour and performance of the model but are not learned from the data. Examples of hyperparameters include learning rates, optimizer, regularization parameters, and the number of layers in a neural network. The selection of optimal hyperparameters can significantly impact the model's accuracy and generalization ability. Hyperparameter tuning involves systematically exploring different combinations of hyperparameters to find the best configuration that maximizes the model's performance on a validation set. Techniques such as grid search, random search, and Bayesian optimization are commonly used for hyperparameter selection and tuning.

Training

Learning rate Scheduler The learning rate is a hyperparameter that determines the step size at which the model updates its parameters during gradient descent. Setting an appropriate learning rate is crucial for achieving optimal convergence and preventing the model from getting stuck in local minima. The learning rate callback scheduler implement a time-based decay approach, where the learning rate decreased gradually over time. This decay strategy allow for a more fine-tuned optimization process, where larger steps takes place initially for faster progress, and smaller steps later to ensure convergence.

2.2.5 Evaluation

Model evaluation involves measuring how well the model generalizes to unseen data and its ability to make accurate predictions. Various evaluation metrics are used depending on the

nature of the problem (classification or regression), such as accuracy, precision, recall, F1-score, or area under the receiver operating characteristic curve (AUC-ROC) for classification while for regression, in the context of time series forecasting, commonly used evaluation metrics for time series forecasting include mean absolute error (MAE), root mean square error (RMSE), mean absolute percentage error (MAPE), and symmetric mean absolute percentage error (SMAPE). More in-depth discussion on these is provided in the next section..

- **Root Mean Squared Error** abbreviated as *RMSE*, is a measure of the average magnitude of the residuals, or errors, between predicted and observed values. RMSE gives more weight to larger errors, making it particularly useful when larger errors are considered more significant or impactful. The lower the RMSE, the better the model's predictive accuracy, as it indicates a smaller average discrepancy between predicted and observed values.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \quad (1)$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \quad (2)$$

Figure 2.5: RMSE and MAE

- **Mean Absolute Error** abbreviated as *MAE*, provides an average measure of the absolute differences between predicted and observed values. MAE treats all errors equally, regardless of their magnitude, making it a useful metric when all errors have equal importance. Similar to RMSE, a lower MAE indicates a better predictive model, as it implies a smaller average absolute discrepancy between predicted and observed values. Equation 1 and Equation 2 shows the formula to calculate the RMSE and MAE.

Due to the fixed number of potential artifacts, the implementation of discussed TSF pipeline in multiple projects end up generating similar artifacts such as train data, test data and, trained model; this leads to repetitions of the same steps to generate above mentioned artifacts and reduced productivity. Through this research we want to leverage the reusability

and adaptability properties of UML class diagram to propose model driven software framework which would facilitate domain adaptation and avoid the repetitions of the same steps by offering reusability of artifacts geenrated in TSF pipeline.

2.3 Learning Science

The transition from secondary school education to university education is a critical phase in a student's academic journey, marked by significant changes and challenges. It is during this transition that students face new academic expectations, different learning environments, increased independence, and diverse social interactions. Due to these transitional challenges the observed repercussion is the academic dropout or failure in a specific course. Hence helping students navigate this transition is vital, and it aligns closely with the principles and goals of learning science.

Learning science focuses on understanding how individuals learn, develop knowledge, and acquire skills [28]. It encompasses research, theories, and methodologies from various disciplines such as cognitive psychology, neuroscience, educational psychology, and computer science. Hence learning science plays a crucial role in helping students in this academic transition.

With this aim, The Office of Science Education (OSE) at McGill University started an initiative called SciLearn in Fall 2020. Below we explain how thesis contributions integrate with this thesis.

2.3.1 SciLearn

It is a neuroscience-based learning skills program helps undergraduates adjust to university studies [29]. SciLearn uses insights from the learning sciences as depicted in Figure 2.6, specifically from neuroscience, and education psychology to help students gain awareness about how their brain works and how to become self-regulated learners. The program is designed and facilitated by neuroscientists and education specialists and comprises a series

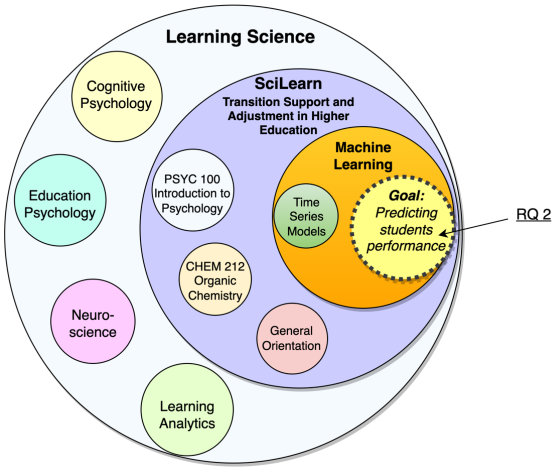


Figure 2.6: Demonstration on how RQ2 fits in learning science

of orientation workshops, labs, peer collaboration sessions, and special events. Since its inception, a significant amount of student learning data has been generated.

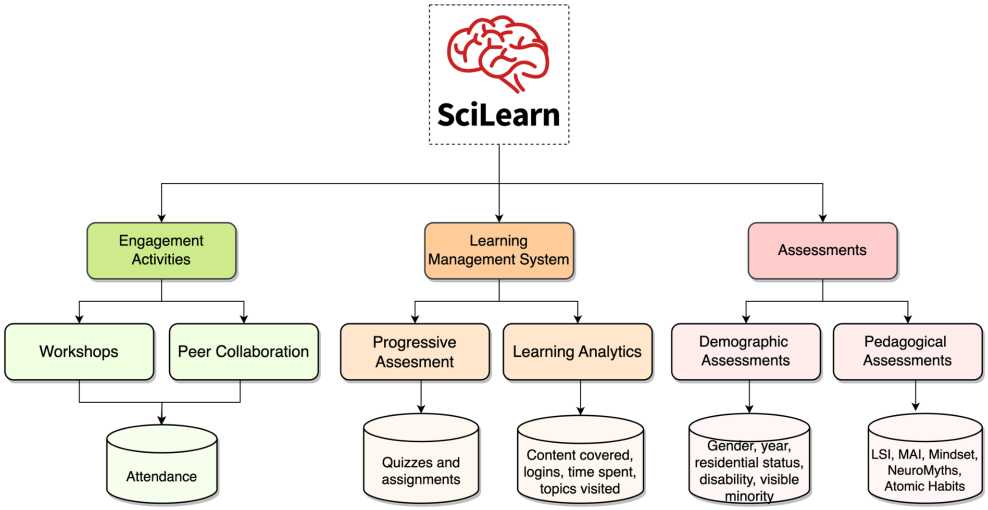


Figure 2.7: SciLearn Program Modules

Figure 2.7 reflects the various modules of the SciLearn program and how the various data is being collected for further study from these modules. Since student's success in a course is one of the key contributor to the overall academic transition success hence SciLearn focus is on to predict students final performance (grades) in a course so that an early intervention could be done by the educators to help at risk (poor performance) students. Below we briefly discuss various modules and data collected as part of this program as shown in Figure 2.7.

SciLearn Engagement Activities

a. Orientation sessions: Through a series of orientation workshops, the aim is to guide participating students on "How the Brain Works" under the guidance of three neuroscientist-facilitators. These orientations also introduces the five atomic habits that we discuss in Section 2.3.1.

b. Peer Collaborations : SciLearn Peer Collaboration is a space for collaborative learning supported by teaching assistants (TAs), TEAM students (part of the SciLearn workshop mentors), and course instructors. The aim is to facilitate space where the participating students can work collaboratively and learn from each other, Ask for guidance from TEAM students and TAs, study in a shared space. and make friends.

Learning Management System

To capture student engagement with the course content and academic performance throughout the term, we utilized McGill's Learning Management System (LMS): myCourse. The LMS, offered a comprehensive view of student engagement with the course content (learning analytic), progressive and actual final grades. This data enabled us to assess student academic progress, track their continuous performance and content engagement, and identify potential correlations between engagement and performance.

Assessments

As part of the SciLearn program, students go through self-reported surveys towards the beginning and end of the term. These survey aims to collect student's demographic, and learning profile (learning inventory data). As the students go through the SciLearn workshops, we track how their learning profile are changing. This helps in assessing the quality of the orientations and perform improvisation in the next offering.

a. Demographic As part of the SciLearn program students have consented to provide their demographic information such as "gender", "year of enrollment", "residential status",

'disability status", and "visible minority status". Please refer to Chapter 5 for further information on this.

b. Pedagogical Assessment(Learning Inventories) Learning inventories are valuable tools used in educational research and practice to assess various aspects of learning, such as learning strategies, metacognitive awareness, and cognitive processes. These inventories provide researchers and educators with valuable insights into students' learning behaviors, preferences, and areas of strength or weakness. There are two widely used LIs are Metacognition Awareness Inventory (MAI) and Learning Strategies Inventory (LSI) as described below:

1. **Learning Strategies**: Developed by Saundra McGuire, this 11-item inventory provides a prediction for a given student's grade by virtue of the study skills they implement regularly, see below. Figure 2.8 depicts the relationship between LSI score and grade.

LSI Score	Predicted grade
9 or more	A
6-8	B
4-5	C
2-3	D
Less than 2	F

(a) LSI score range

Score	Mindset
22-30	Strong growth
17-21	Growth with some fixed ideas
11-16	Fixed with some growth ideas
0-10	Strong fixed

(a) Mindset score range

Figure 2.8: Range scale to estimate (a) grades (b) mindset

2. **Metacognitive Awareness**: The MAI, developed by Schraw and Dennison in 1994, is designed to measure individuals' meta-cognitive awareness—their ability to monitor and regulate their own thinking processes [30]. This 19-question inventory focuses on the 2 larger categories: (1) knowledge of cognition, and (2) regulation of cognition. We have also completed the data analysis for all the subcategories as well, please see below.

3. **Mindset**: This 30-item questionnaire provides insight into an individual's mindset on a scale of strong fixed mindset and strong growth mindset, both well-studied concepts, largely attributed to Carol Dweck [31]. Please refer to the rate scale shown in Figure 2.8.

4. **Neuromyths:** This 10-item questionnaire give a snapshot of the fundamental misconceptions about how the brain works and how adult learning takes place [32].

5. **Atomic Habits:** Introduced as part of teaching content in the orientations, ther are five atomic habit that the program introduce to the students and its significance in academic success such as "notes taking", "me time", "teaching others", "sleeping better", and "avoid multi-tasking". Through the survey towards the end of the term we aim to asses how many of these habits students are adopting and applying in the course.

Before this MSc thesis project, the SciLearn team has been leveraging the LI range scales that we discussed in Section 2.3.1 to estimate the student's grade in a course which is not effective in terms of efforts and the accuracy. ML is advancing rapidly in the field of education, specifically in predicting student's performance, Hence with this research we aimed to investigate the efficiency of TSF models and employ the TSF pipeline by leveraging the data collected.

2.4 Gas Turbines

Gas turbines [33] operate in all parts of the world for power generation (converts chemical energy of fuel either into mechanical energy or into kinetic energy) and as a source of energy for pumps and compressors. The conversion of fuel energy into shaft power requires interaction of several components of the engine (as shown in Figure 2.9) within each of them a chain of energy conversion takes place.

Gas turbine engines work according to the Brayton cycle and are divided into three main parts, compressor, combustor and turbine as shown in Figure 2.9(b). Compressed air from

the compressor enters the combustor, where fuel is injected, and the mixture is burnt. The combustion gases are expanded through the turbine, which drives the compressor and gives a net power output that can be used to drive a generator or a pump. The primary fuel used in modern gas turbines is natural gas, but many engines have the capability to operate on liquid fuel as well as alternative gas mixtures. Figure 2.9 reflects the only a brief description of the gas turbine working principle is given here; more details can be found in, e.g. the gas turbine handbook by Boyce [34]. Gas turbines have the capability to start quickly. Therefore, gas turbines will be a central part within the power generation business for many years to come.

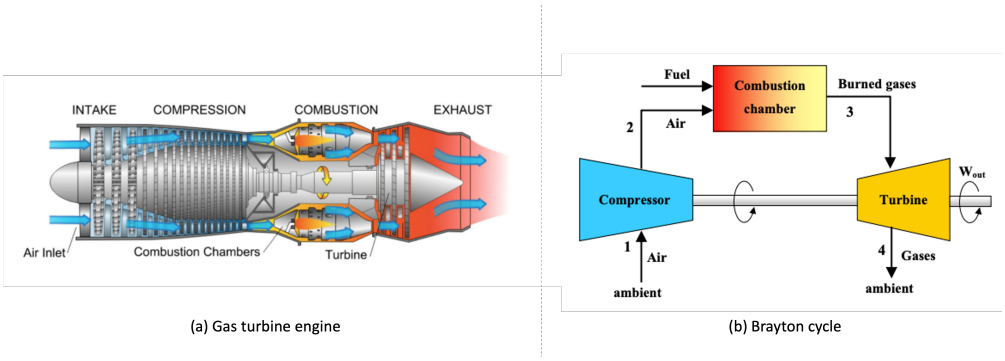


Figure 2.9: Gas turbine design (a) [3] and working principle(b) [4]

Next we discuss the combustion system part of the gas turbine which is the main central focus of this research.

2.4.1 Combustion System

Combustion process is primarily responsible for the production of high-temperature and high-pressure gases that drive the turbine and generate mechanical work. The combustor (facilitates combustion process) features eight individual cans which provides single digit NOx capabilities in a wide range of operation conditions. One of the eight combustor cans is shown in Figure 2.10. Each of the combustor cans are fed by compressor discharge air through a common annular casing. Convective and impingement cooling techniques are utilized for cooling of the combustion chamber walls. The burner comprises two separate main fuel lines (main 1 and main 2) as shown in Figure 2.10 for further improved tuning exibilities. An optimized aerodynamic design ensures a well-defined re-circulation zone for

stabilizing the flame. In addition, a pilot and a RPL (Rich Pilot Lean) burner are used for central stabilization of the main flame. After combustion, the combustion gases are led through a transition duct to the inlet of the first turbine stage.

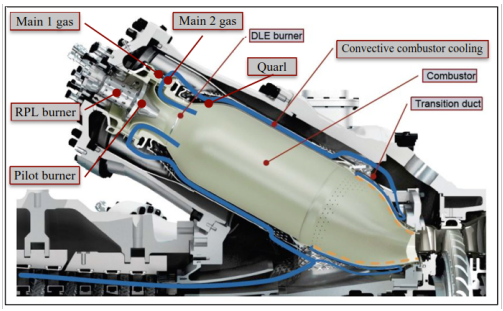


Figure 2.10: Combustion system [5]

Combustion Instabilities

Combustion instabilities refer to undesired and uncontrolled oscillations in the combustion process within a combustion chamber. Combustion instabilities can be divided in two categories, combustion noise and TA instabilities. Those are both driven by the combustion process but the characteristics and physical phenomena is different.

1. **Combustion Noise** The flow in gas turbine combustors is inherently turbulent. This turbulence creates flow variations that affects the combustion process and results in combustion noise. This noise is sometimes called "combustion roar" and is of a broadband character with relatively low amplitude, [35].

2. **TA Instabilities** on the other hand commonly appears as large amplitude oscillations at one of the systems natural frequencies. Those instabilities are spontaneously excited and the oscillations are maintained by a feedback loop between the combustion and the acoustic field. The principle for TA instabilities is illustrated in Figure 2.11. The unsteady heat release in the flame generates acoustic waves which are reflected at the system boundaries and standing waves are formed. The acoustic fluctuations give rise to flow and mixture perturbations which in turn affects the flame with a fluctuation of the heat release as the result, the loop is closed [36]. The oscillations will be amplified or damped depending on the phase between the heat release and the pressure. In

contrast to combustion noise, TA instabilities are characterized by high amplitude oscillations at distinct frequencies. Those large oscillations in velocity and pressure are highly unwanted and can cause severe wear and structural damage to the gas turbine.

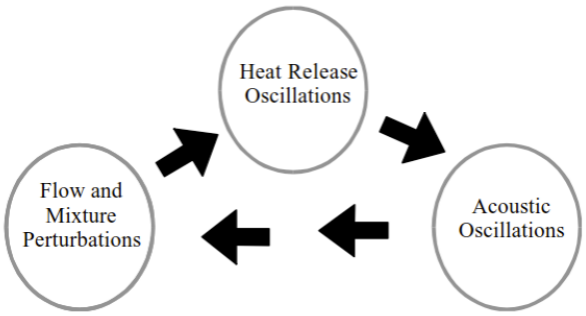


Figure 2.11: The feedback loop responsible for TA instability [6]

Siemens is a well known manufacturer of industrial gas turbines with a portfolio including gas turbines. The Siemens gas turbines are sold to customers all around the world. Consequently, Siemens invest a significant portion of time towards the maintenance and performance optimization of gas turbines. There are various factors which contributes towards the performance degrade of the overall gas turbines and combustion instabilities are one of the most dominant reasons. These instabilities arise due to the coupling between combustion and acoustics within the gas turbine system as discussed above Section 2.4.1. Hence Siemens specifically invest time doing research and development (R&D) on how accurately the instabilities can be predicted before time so that a timely intervention (such as alteration in fuel and operating conditions, combustion control etc) is done to mitigate damaging effects. This research is a contribution towards this sector of Siemens Energy to investigate the efficiency of time series forecasting in predicting TA instabilities and integrating the TSF pipeline.

Chapter 3

Related Work

In this chapter, we surveyed the related research on software modeling for designing ML frameworks, TSF in predicting student performance and thermoacoustic instability in combustion system. Moreover, we describe how the related work motivates our work in the area of domain modelling and TSF and the associated research questions in this thesis.

3.1 Frameworks for Time Series Forecasting

Model-driven engineering plays a vital role in software development, providing a structured approach for designing, analyzing, and implementing software systems. In the domain of machine learning and deep learning [37], model-driven approaches have gained attention for their potential to enhance the development process and improve the quality of software systems. By leveraging modeling techniques such as UML class diagrams, these frameworks enable the specification and visualization of system architectures, components, and relationships. The use of model-driven engineering in the machine learning and deep learning domains offers several advantages, including improved software quality, enhanced maintainability, and increased productivity [38]. UML, in particular, promotes usability and adaptability by providing a standardized notation that fosters effective communication among stakeholders, facilitates model comprehension, and allows for seamless integration with other modeling languages and tools [39]. By utilizing model-driven frameworks that embrace UML, devel-

opers can create scalable software systems that incorporate best practices and adhere to industry standards. In this section, we explore the related work on UML class-driven or model-driven frameworks for machine learning and deep learning, focusing on the methodologies, challenges, and benefits reported in the existing literature.

Several previous studies and research have demonstrated the application of model-driven engineering or UML class diagrams in conceptualizing frameworks for process flows or software systems, encompassing machine learning, deep learning, and time series forecasting.

One notable study by Zongben Xu and Jian Sun [40] have conducted research on "Model-driven deep learning", which focuses on integrating model-driven methodologies with deep learning techniques. Author has used the concept of models to categorize the overall deep learning approach into three classes: 1. "models family", provides a very rough and broad definition of the solution space, 2. "algorithm family", refer to the algorithm with unknown parameters for minimizing the model family in the function space, and then it is unfolded to 3. "deep network", a deep network with which parameter learning is performed as a deep learning approach. It is worth highlighting that the proposed topology only focuses on identifying the right deep learning network architecture and works as one of the motivation to extend the idea to TSF pipeline.

Another significant progress is being made towards building automated machine learning AutoML frameworks to facilitate the use of ML pipeline by non ML experts, automate and streamline the machine and deep learning pipelines. Xin and Zhao [41] in their review paper compared various state of the art AutoML library frameworks developed so far and its contributions in improving the efficiency of implementing ML and DL pipelines, however these frameworks have their dedicated scope defined, for instance, "TPOT" [42], "Auto-WEAK" [43], and "AutoSklearn" [44] are built on top of "scikit-learn" [45] for building classification and regression pipelines, but they are only efficient for implementing traditional ML models (such as SVM and KNN). They fail to capture the adaptability and configurability of these models based on specific use case requirements. Also these frameworks are generalized to ML and DL specific prediction pipelines which are not suitable to capture

temporal dependencies in TS data. Hence they not efficient in automating and streamlining TSF pipelines.

Furthermore, the integration of UML and TSF techniques in designing the use case specific time series frameworks such as "UML framework for smart cities with forecasting electrical consumption" [46] by Ricardo Alirio and "UML model based failure time series prediction" [47] by Wang Xin have encouraged and laid the foundation for using UML for designing reusable framework for TSF pipeline.

The existing literature in the field of TSF and software modeling has provided valuable insights into various modeling techniques and approaches. However, there is a noticeable gap when it comes to the development of frameworks that promote the reusability and adaptability of time series analysis across multiple domains. This gap serves as the motivation for the proposed conceptual design of our TSF framework that can accommodate diverse domain-specific requirements.

3.2 Students Performance Prediction

Much research has been done in the area of educational mentor support where a predictive model is built to forecast student performance to identify at-risk students as well as contributing features in the course. These researches are taking place because of the complex nature of learning, e.g., performance depends on many characteristics related to the learner. Possible characteristics include the student's recent academic assessments, demographics, psychological , culture, and educational background [48], and engagement with course content. Demographic factors consist of family background, gender, disability and age, and all of these are considered important attributes [49]. If we look at academic progress, the student's grade is among the most important attributes that can be used to assess performance in a specific course [50], whereas academic potential can be evaluated by the student's GPA especially during transition to university education. Our research introduces a few new at-

tributes that focus on using descriptive features from the learning sciences related to study behaviours as well as content engagement and their mutual effect on performance.

The field of early grades prediction using machine learning has seen notable advancements in recent years. Baashar and Alkawsi (2021) [51] conducted a systematic literature review on predicting student academic performance using machine learning algorithms. Their study explored various approaches, including decision trees, support vector machines, and artificial neural networks, highlighting the strengths and limitations of each method. A review done on almost thirty (30) data-driven studies revealed that (i) research in this area has significantly increased in recent years; (ii) Academic (CGPA; attendance), demographic (Gender), internal assessment (Quiz; assignment) and family/personal attributes play an important role in the prediction; (iii) Artificial neural network (ANN) perform relatively better than other classical ML algorithms.

Another significant research was done by Chen and Cui [52] for early prediction of students performance using time series analysis. They incorporated a deep learning approach (LSTM) to analyze student online temporal behaviours (learning analytics) using their Learning Management System (LMS) data for the early prediction of course performance. The results indicated that using the deep learning approach, time series information about mouse click frequencies on LMS successfully provided early detection of at-risk students with moderate prediction accuracy. Also time series model perform better than classifier. This was also one of the motivation for our study to move towards time series forecasting rather than the classifier approach.

Further, research in the field of learning science by Aurah [53] and Dill [54] used metacognitive, demographic, and LSI data to asses the performance of 12th grade students. The results revealed statistically significant differences in metacognition in form of metacognitive prompts between groups. Gender effects were also noted with female students outperforming male students on the genetics problem solving test. Subsequent qualitative data suggested that highly efficacious students (with high scores on metacognition) did better on the tests

than less efficacious students. This was also a motivation to include LI data as part of our study as this shows a significant correlation with the student performance.

The related work discussed in the above paragraphs has shed light on the application of learning science and machine learning for early grades prediction in undergraduate education. While these studies have made valuable contributions and worked as the guiding light, there are still improvement areas. One such example is including learning analytic, LSI and MAI, and their day to day implementation of atomic habits data together with demographics to understand the student academic, demographic and learning behavioural style. Another improvement area is inclusion of comprehensive longitudinal data to capture the dynamic nature of student performance over time. Hence by using TSF and student overall profile together we aim to fill these gaps, enhance the predictive accuracy, and provide a more nuanced understanding of student performance.

3.3 Thermoacoustic Instability Prediction

Thermoacoustic instabilities in gas turbines pose a significant challenge to their performance, safety, and efficiency. Hence the research in this field has gained considerable attention. Researchers and engineers strive to develop advanced techniques that can forecast these instabilities, allowing for proactive management and optimization of gas turbine operations. By predicting instabilities in advance, it becomes possible to implement control strategies, modify operational parameters, or apply design modifications to prevent or mitigate the adverse effects associated with thermoacoustic instabilities.

An et al. [55] has demonstrated the use of variance and the lag-1 auto-regressive coefficient using pressure signal showing the role of critical slowing down in early detection of thermoacousic instabilities. They developed an algorithm based on the system dynamics theory of critical slowing down to predict imminent thermoacoustic instabilities. Results showed that large-amplitude combustion instabilities could be robustly detected using rolling window lag-1 auto-regressive coefficient and variance, from the pressure signals. Combined

with other tools and techniques such as machine learning, this method could potentially enhance the performance of identifying the areas of high amplitudes. This research was the foundation of our study as this was the first paper published in line with the theracoustic instabilities prediction at Siemens.

Another related research by Zengyi Lyu and Yuanqi Fang [56] explored deep learning techniques to predict the future growth of acoustic pressure signals to detect precursors of combustion instability. Through this experiment the practicability of S-LSTM as a prediction tool is investigated using acoustic pressure data on stable and unstable regimes. Which was further compared with the base line ML model support vector machine (SVM). The results indicated the supremacy of deep learning models over classical ML model in this domain. Though this study signals a high potential of deep learning in this domain but one identified improvement area is to include other surrounding parameters such as air velocity and temperature to investigate the possibilities of correlations of these features collectively in predicting the TA instabilities.

Further, literature review surveys [57], [58] on integrating ML and deep learning techniques for performance optimization of gas turbines and multidisciplinary optimization has demonstrated the capable role of deep learning models in this domain specifically neural networks and transformers. Chandrachur Bhattacharya [59] in his research focused on using advanced deep learning networks for early prediction of thermoacoustic instability in a multi-nozzle combustor by leveraging short data lengths (short time dependent features) and highlighted the importance of window length, down-sampling and noise removal in TSF. Three methods, namely, fast Fourier transform (FFT) [60], symbolic time series analysis (STSA) [61], and hidden Markov modeling (HMM) [62] was investigated with HMM being the top prefroming model. This research helped in emphasising importance on downsampling and selecting the appropriate windowing length in our TSF analysis.

Hence, the above related research has tremendously helped in identifying notable potential improvements such as inclusion of surrounding parameters like air velocity, temperature and pressure might help in increasing the predictive capability of TSF models, importance of

down sampling if required, and the importance of choosing an appropriate windowing length also important in integrating TSF pipeline.

3.4 Summary

In this chapter, we survey prior research along with software modeling for ML, existing frameworks for ML and TSF, use of deep learning models in predicting student performance and thermoacoustic instabilities in gas turbines that are central to this thesis. We find that despite the existing presence of various ML and deep learning AutoML frameworks, these tools have not been feasible to capture the time series specific characteristics such as temporal dependencies and also do not facilitate the domain specific adaptation of the framework. While implementation of software modeling on some of the ML use cases has motivated and presented an opportunity to apply the capabilities of software modeling in ML to fill those gaps and design an adaptable and reusable framework for end-to-end implementation of TSF pipeline.

Also, the existing use of deep learning techniques and its efficiency in students performance assessment and thermoacoustic instabilities has motivated us to employ these methods in our RQ2 and RQ3 with the identified gaps such as use of learning analytic, demographic, LSI and metacognitive data (overall student personality) in performance prediction, and inclusion of surrounding sensor data in combustion such as air velocity, pressure and temperature along with the importance of selecting the appropriate window length in thermoacoustic instability prediction.

Broadly speaking, the remainder of this thesis presents our approach and implementation which aims to address the above key takeaways. We begin, in the next chapter, by presenting an UML architecture which aims streamline TSF pipeline implementation by offering reusable and adaptable components.

Chapter 4

Proposed Framework

Objective 1

1. Design a general software framework for developing TSF pipelines.
2. Validate the framework's adaptability, reusability and completeness.

This chapter presents a conceptual model-driven framework for TSF that facilitates domain adaptation and promotes reusability . The framework is designed to provide flexibility and modularity, to enable data evaluation on defined parameters, data processing with different techniques, domain specific model selection, and model evaluation on selected metrics. The framework architecture is designed using a UML class diagram [63].

This chapter is organized in the following sections; towards the end, each section describes how the respective component's classes interact to facilitate the data processing, modeling and evaluation pipeline.

1. **High-level Component Overview**: This section briefly overviews the proposed design framework, highlighting the key components and the workflow.
2. **Data Component**: In this section we focus on the data-related aspects of the framework and the workflow. We explain associated class elements, attributes, methods, and relationship that play an important role in facilitating data collection, processing, and transformation techniques.

3. **Model Component**: This section delves into the modeling aspect of the framework. It elaborates on the **Modeler** class and its associated subclasses, attributes and methods which together facilitates the model selection and training pipeline.

4. **Evaluation Component**: This part explains the evaluation and performance assessment of the forecasting models. It details the **ModelEvaluator** and **DataEvaluator** classes, which handle model evaluation and data quality assessment, respectively.

5. **Advantages**: This section highlights the benefits of the proposed framework. Explains how the framework facilitates component reuse, component modularity, and adaptability to different TSF scenarios.

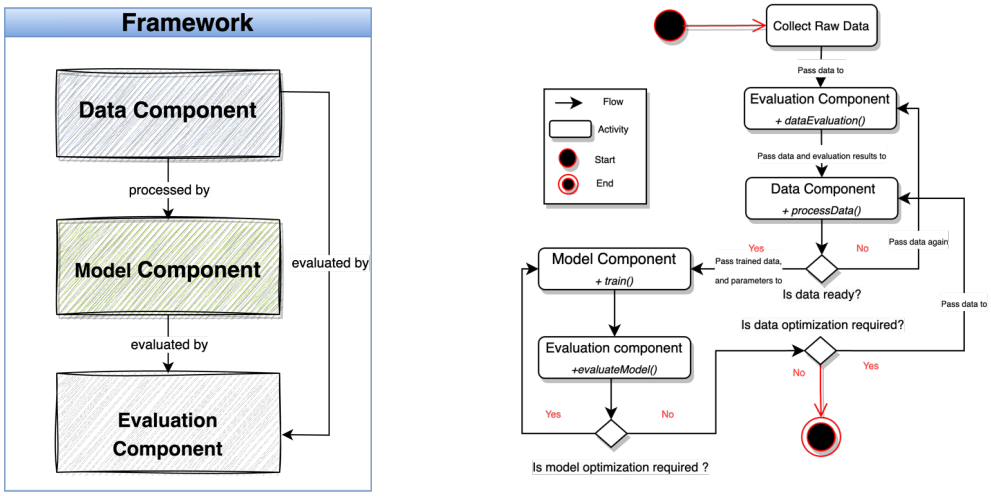
6. **Architecture Validation:** This section defines the metrics to asses the validity of proposed framework along with its validation using case studies explained in Chapter 5 and Chapter 6.

7. **Summary**: Finally, we conclude the chapter with outlining the overall benefits and findings of the validation.

4.1 High-level Component Overview

The proposed conceptual framework (class diagram) consists of several key components that work together to provide a comprehensive and systematic solution for TSF. The UML class diagram architecture allows the definition of high-level classes that capture the common characteristics and behaviors of different TSF artifacts.

It contains three key components: Data, Model, and Evaluation. The Data component (DC) comprises two main classes: **Dataset** and **ProcessingPlanner** as depicted in Figure 4.2. On the other hand, the Model component (MC) comprises **Modeler** class as shown in Figure 4.4, and the Evaluation component (EC) comprises **'Evaluator'** class as reflected in Figure 4.6. The DC is responsible for preparing the data, including operations such as data cleaning, processing and transformation. The MC access the transformed train data



(a) Components interaction

(b) Components activity diagram

Figure 4.1: Interaction of framework's component and activity diagram

from the DC and trains a selected model to generate a trained model. Finally, the EC takes the trained model and test data from the Model and Data components. It then evaluates the performance of the model. Once instantiated, the processing within each component is automatic, while user can modify the input parameters of the next component based on the output of the current component while advancing to the next component. The user can also add custom methods to a class in a component but only at configuration level not during run-time. Figure 4.1b shows the activity flow discussed among these components. The class diagram shown in Figure 4.1a depicts the three components and their mutual interactions.

4.2 Data Component

The Data component handles raw time series data collection and processing, including cleaning, transformation, and data splitting for training and validation. It provides functionalities to handle various data cleanings tasks such as data formats, missing values, and outlier handling. Additionally, it incorporates feature selection and extraction methods to select and derive meaningful features from raw time-series data, which can enhance the performance of forecasting models. The data component comprises two main classes **DataSet** and **Pro-**

cessingPlanner, which are further expanded to their corresponding subclasses as shown in Figure 4.2 and discussed below.

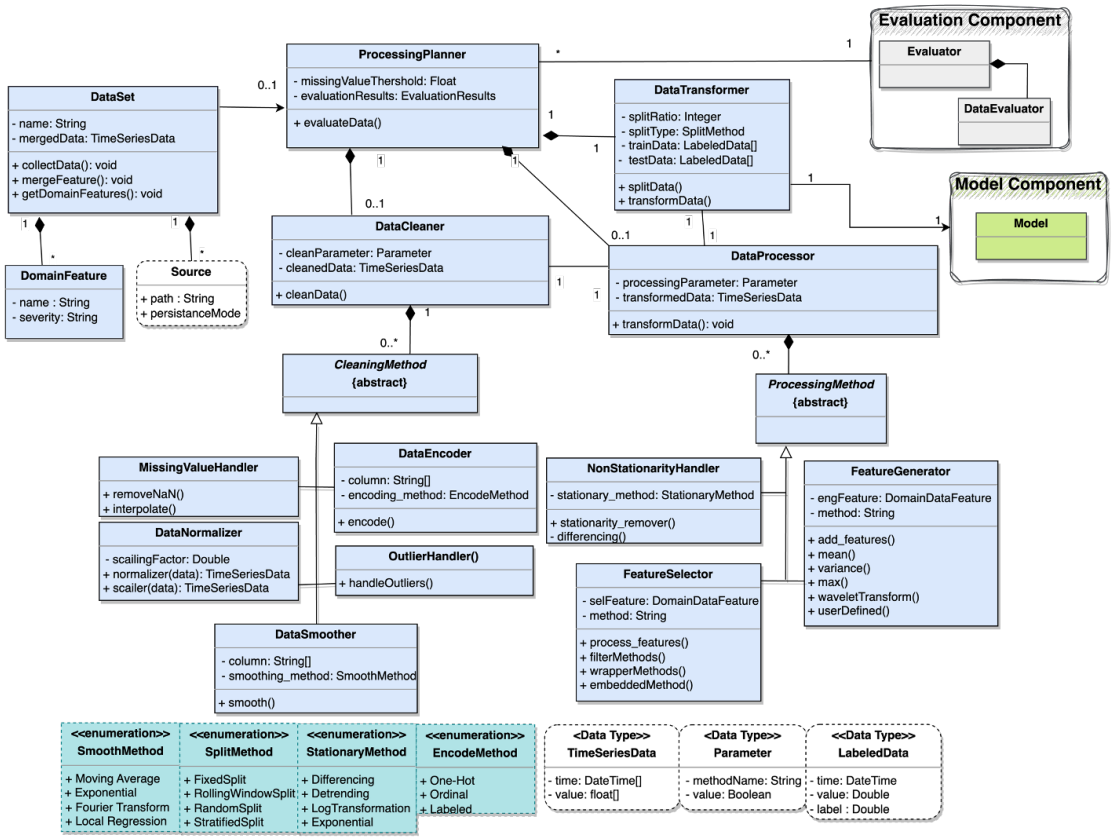


Figure 4.2: Class diagram for data component

4.2.1 DataSet

DataSet class is the central entity that manages the final dataset to be used by **ProcessingPlanner** , its sources and domain-specific characteristics. As depicted in Figure 4.2, it represents the final data set merged from one or many data streams from various sources. The class also facilitates the management of storage and retrieval of the data used in the predictive modeling process of time series. Below we discuss the attributes, relationships with other classes and operations that could be performed within the scope of this class:

Attributes

1. *'name'*: This is of type string and stores the name of the final data set to be used by class **ProcessingPlanner** for further processing.

2. *'mergedData'*: This attribute represents the main time series data that is collected and aggregated within the DataSet. It is of type **TimeSeriesData**, which is a data structure specifically designed to hold univariate and multivariate time series data. The object **TimeSeriesData** contains attributes such as *'time'* (representing the timestamps) and *'values'* (storing the actual measurements).

Relationships

1. **DataSet** and **DomainFeature**

   The **Dataset** class has a composition relationship with the **DomainFeature** class. Based on the use cases, **DataSet** class may or may not have domain-specific features, which are features listed by domains experts that need to be included even if they do not reflect strong correlations with the target label. By having a composition relationship, the **Dataset** class takes ownership of the **DomainFeature** objects, and the **Dataset** object manages their lifetimes. This composition allows the **Dataset** class to gather and store various domain-specific features along with the dataset, enabling more comprehensive analysis and modeling in the TSF process.

2. **DataSet** and **Source**

   The **Dataset** class exhibits a composition relationship with the **Source** class with a one-to-many multiplicity. By having this composition relationship, the **Dataset** class can gather data from multiple sources and consolidate them into a cohesive dataset for time series analysis.

Methods

1. ***collectData():*** This method collects the time series data from various sources. Depending on the specific requirements, it may involve retrieving data from databases, files, or external APIs. The method handles the data collection process, ensuring that the required data is obtained.

2. ***mergeData():*** It handles the aggregation of the data from multiple sources into a single dataset. In TSF, it is common for data to come from different sources or split across multiple files or databases. This method combines the data collected from various sources, aligns the timestamps, if necessary, and merges the data points to create a unified and comprehensive time-series data set.

3. ***getDomainFeature():*** This method retrieves the domain-specific characteristics associated with the time series data. These features capture additional information or characteristics relevant to the specific domain or context of the time-series forecasting problem.

4.2.2 DomainFeature

This class provides a mechanism to prioritize and assess the importance of different features within the domain context. This information can be used to guide feature selection, preprocessing steps, and model development, ensuring that critical aspects of the time series data are considered during the forecasting analysis. The class has a composition relationship with **DataSet** that we have already covered in Section 4.2.1. **DomainFeature** doesn't have any methods associated.

Attributes

1. *'name'* specifies the name of the feature.

2. *'severity'* defines the importance or significance of the feature. This attribute allows domain experts and machine learning engineers to quantify the relevance or impact of the feature on the forecasting process.

4.2.3 Source

The class enables the identification and configuration of different data sources, allowing flexibility in accessing and managing the time series data for forecasting purposes. **Source**

class has composition relationship with **Dataset** which has been covered in Section 4.2.1. This class doesn't have any method.

Attributes

1. *'path'* specifies the path or location of the data source.

2. *'persistanceMode'* defines how the data source is accessed or stored, such as a local system, server, or database etc.

4.2.4 ProcessingPlanner

The scope of the **ProcessPlanner** class is to serve as the central orchestrator for the time series data processing pipeline. Provides a cohesive and reusable component that handles the data cleaning, processing and transformation tasks. The class is responsible for coordinating the activities of its composed subclasses, namely **DataCleaner**, **DataProcessor**, and **DataTransformer**, ensuring that the data is processed efficiently and effectively. The **ProcessPlanner** class accepts time series data from the **DataSet** class and triggers the evaluation of the data using the **Evaluator** class. It uses the *'evaluationResults'* to determine the appropriate data cleaning, processing and transformation methods to apply.

Attributes

1. *'missingValueThershold'*: This attribute represents a user-defined threshold that helps determine how missing values in the time series data should be handled. The attribute serves as a parameter that can be adjusted on the basis of the user's preferences or domain-specific requirements. It influences the behavior of the **DataCleaner** class within **ProcessingPlanner** by guiding the choice of appropriate techniques to address missing values. For example, if the number of missing values in a particular time series exceeds the specified threshold, **DataCleaner** may decide to remove those instances or apply data imputation techniques such as interpolation or mean substitution to fill in the missing values.

2. *'evaluationResults'*: This attribute holds the results of the data evaluation performed by the **Evaluator** class. The results stored in this attribute provide valuable insights into the quality and characteristics of the time series data, enabling informed choices for subsequent data processing (cleaning and transformation) steps.

Relationships

1. **ProcessingPlanner** and **DataCleaner**

   The ProcessingPlanner class has a composition relationship with the **DataCleaner** class, indicated by the multiplicity of 0 to 1. This implies that each instance of the **ProcessingPlanner** class is associated with a maximum of one instance of the **DataCleaner** class.

2. **ProcessingPlanner** and **DataProcessor**

   Similarly, the **ProcessingPlanner** class has a composition relationship with the **DataProcessor** class with a multiplicity of 0 to 1. The **ProcessingPlanner** leverages the functionalities of the **DataProcessor** to perform data transformation operations on time-series data.

3. **ProcessingPlanner** and **DataTransformer**

   The **ProcessingPlanner** class also has a composition relationship with the **DataTransformer** class, indicated by a multiplicity of 1 to 1. This means that each instance of the **ProcessingPlanner** class is associated with exactly one instance of the DataTransformer class.

Methods

1. ***evaluateData()***: This method is responsible for evaluating the time series data on various parameters, such as missing values, trends, seasonality, outliers, and other relevant aspects. This method utilizes the capabilities of the **Evaluator** class to perform comprehensive data analysis and generate evaluation results.

4.2.5 DataCleaner

As reflected in Figure 4.2, the **DataCleaner** class facilitates data cleaning operations on raw time series data received from its composite class **ProcessingPlanner** and applies the appropriate data cleaning method.

Attributes

1. *'cleanParameter'*: This attribute holds the decision on which cleaning method to use based on the evaluation results stored in the attribute *'evaluationResults'* of the **ProcessingPlanner** class.

2. *'cleanedData'*: This attribute holds the cleaned time-series data after the data-cleaning operations have been performed.

Relationships :

1. **DataCleaner** and **CleaningMethod** The DataCleaner class has a composition relationship with the abstract class ***CleaningMethod*** with the multiplicity of zero-to-many. This composition represents that the **DataCleaner** class has one or more instances of an **CleaningMethod** abstract class associated with it. The abstract class **CleaningMethod** serves as a base class for representing various cleaning methods in the context of the **DataCleaner** class. It does not have any attributes or methods but defines a common interface or contract that its subclasses must implement.

2. **DataCleaner** and **DataProcessor** The class is associated with **DataProcessor** so that after the data cleaning, the cleaned data can be shared with the **DataProcessor** class for further data processing.

Methods :

1. ***cleanData()***: This method coordinates the data cleaning tasks within the **ProcessingPlanner** class. This method uses the capabilities of the **CleaningMethod** class to perform data cleaning operations on time series data based on the evaluation results obtained from the ***evaluateData()*** method.

4.2.6 CleaningMethod

This abstract class serves as the base class for various cleaning methods in the TSF framework. Based on the use case requirement, it provides a generalized interface and common functionality for cleaning operations on time series data.

Child Classes :

1. **MissingValueHandler**: This class extends the **CleaningMethod** abstract class and implements methods such as interpolation, replaced with zero methods for handling missing values in the time series data.

2. **DataNormalizer**: This class extends the **CleaningMethod** abstract class and implements a specific normalization method fit for the domain, ensuring consistency and scale across different features.

3. **DataSmoother**: This class extends the **CleaningMethod** abstract class and implements methods for smoothing the time series data, reducing noise and variability.

4. **DataEncoder**: This class extends the **CleaningMethod** abstract class and implements specific methods for encoding categorical or non-numeric features in the time series data.

5. **OutlierHandler**: This extends the **CleaningMethod** class and helps in implementing the selected removal operations.

4.2.7 DataProcessor

This class is responsible for applying some of the processing methods, such as feature engineering, transforming non-stationary data to stationary data, selecting top features and reshaping the input data such that it can be compatible for training with the model selected.

Attributes :

1. *'transParameters'*: This attribute stores the parameters related to the transformation methods required to transform the data, such as transforming data to stationary, creating new features, selecting highly correlated features, and reshaping the data based on the model selected.

2. *'transformedData'*: This attribute stores the transformed data after the transformation process.

Relationships :

1. **DataProcessor** and **ProcessingMethod**

   The **DataProcessor** class has a composition relationship with the abstract class **ProcessingMethod**. This relationship allows the **DataProcessor** class to access and utilize various transformation methods implemented in the subclasses of **ProcessingMethod**.

2. **DataProcessor** and **DataTransformer**

   The class is associated with **DataTransformer** so that after the data has been transformed to be compatible with modeling, it can be shared with the **DataTransformer** class for further data splitting into train and test data for training and validation respectively.

Methods

1. ***processData()***: This method is responsible for performing the necessary transformation operations based on the selected transformation techniques in *'transParameter'*. It applies various transformation techniques implemented in the subclasses of **TransformTechnique** to prepare the time-series data for further processing or analysis. The transformed data is then returned as a **LabelledData** object.

4.2.8 ProcessingMethod

This abstract class serves as the base class for various transform methods in the TSF framework. Based on project requirements, it provides a generalized interface and common functionality for cleaning operations on time series data.

Child Classes :

1. **NonStationarityHandler**: This class extends the **ProcessingMethod** abstract class and implements methods such as differencing, detrending, and exponential methods for handling non-stationarity in the time series data.

2. **FeatureSelector**: This class extends its parent's class and implements a specific feature selection method as reflected in Figure 4.2.

3. **FeatureGenerator**: This class is responsible for implementing standard methods configured for creating new features such as creating new features by using mean, variance or any other user-defined logic.

4.2.9 DataTransformer

The **DataTransformer** class is responsible for splitting the data into training and testing datasets. It has the following attributes:

Attributes

1. *'splitRatio'*: An integer value representing the ratio or percentage of data to be allocated for training. The remaining portion will be allocated for testing.

2. *'splitType'*: An instance of the **SplitMethod** class, which defines the method used for splitting the data (e.g., random split, chronological split).

3. *'trainData'*: stores the training dataset.

4. *'testData'*: stores the testing dataset.

Relationships

1. **DataTransformer** and **Model**

   The **DataTransformer** class has a composition with the **Modeler** class. This association facilitates the functionality for sharing the training data with the Model component. This relationship ensures that the split and labeled data generated by the "DataTransformer" class are accessible to the Model component for training purposes. It allows for seamless integration and data flow between the two components.

Methods :

1. ***splitData()***: This method takes the input data from **DataProcessor** and splits it into training and testing datasets based on the provided split ratio and split method. The resulting training and testing datasets are stored in the *'trainData'* and *'testData'* attributes.

2. ***transformData()***: This method reshapes and transforms the data in the required dimension based on the rolling window or lags, which will be used as the labels for the model during training. This step ensures that the data is prepared in the appropriate format to be fed into the selected model.

4.2.10 Component Workflow

Below detailed explanation provides a step-by-step walkthrough of the expanded activity diagram Figure 4.3.

1. Start: The activity starts by collecting raw data and domain characteristics, if present.

2. Trigger: Invoke the object from the **DataEvalation** class of the Evaluation component to access the data evaluation method.

3. Decision: Check the results received from data evaluation if data cleaning is required:

   - Branch 1: Initiate the object **DataCleaner**.

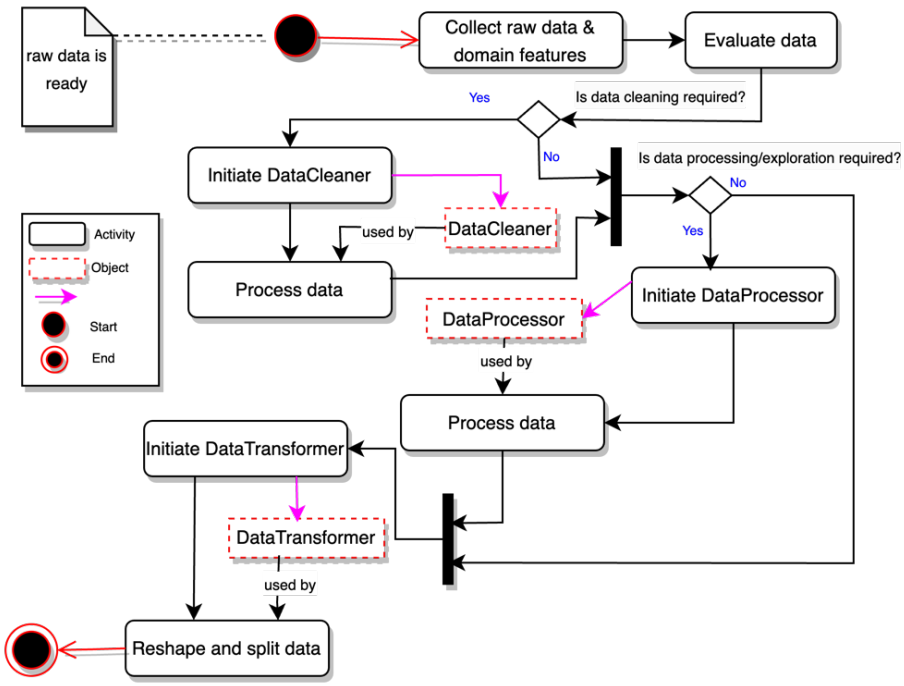


Figure 4.3: Activity diagram for data component

- Apply the required data cleaning methods.

4. Decision: Check the results received from data evaluation if data processing or exploration (in case of correlation analysis) is required:
   - Branch 2: Initiate the object **DataProcessor**.
   - Apply the required data transformation methods.
5. Data Transformation:
   - Initiate **DataTransformer** object.
   - Transform the data into sequence data.
   - Access the relevant split method and divide the data into training and testing datasets.
6. Store Train and Test Data: Store the train and test data sets for further use.
7. Stop: The activity is completed.

4.3 Model Component

The model layer encapsulates the implementation of different TSF algorithms and techniques based on the type of problem domain (based on *'dataCategory'*). It mainly facilitates two tasks model selection through **Model** class and training through **Trainer** class while **Modeler** class overseas and responsible for initiating the appropriate object of these classes when required. This layer includes a wide range of univariate, multivariate, and multistep models such as autoregressive, exponential smoothing, ensemble, and neural network models.

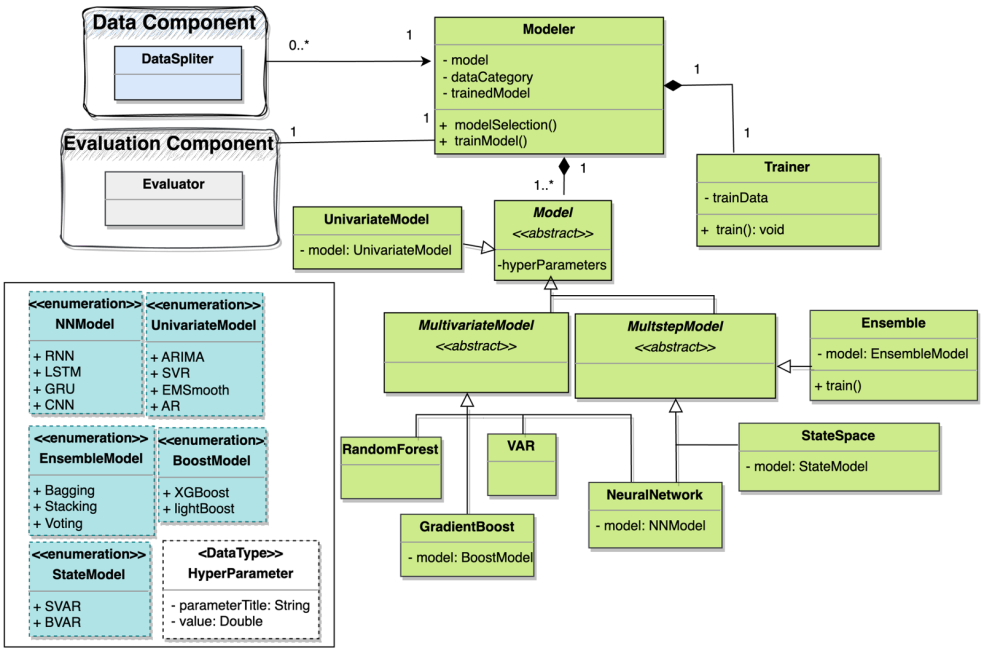


Figure 4.4: Class diagram for model component

4.3.1 Modeler

It facilitates the encapsulation of the necessary information and functionality to select and train a model on a given dataset. We discuss the attributes, methods, and associations bound with the **'Modeler'** class.

Attributes

1. *'model'*: Holds the selected model with the parameters.

2. *'dataCategory'*: Indicates the domain type of the data, whether it is univariate, multivariate or multistep ahead data.

3. *'trainedModel'*: Holds the trained model for further processing.

Relationships

1. **Modeler** and **Model**

   This abstract class serves as a base for different types of TSF model. The association with the **'Modeler'** class allows for categorizing and identifying the models based on domain characteristics and can further be generalized into three abstract classes.

2. **Modeler** and **Trainer**

   Modeler has a composition relationship with **Trainer** class with 1 to 1 multiplicity. This means that **Modeler** class and initiate one object at a time for **Trainer** class. As we only required one training process across TSF pipeline.

Methods

1. ***modelSelection():*** This method is responsible for facilitating model selection based on *'dataCategory'*.

2. ***trainModel():*** This method facilitates initiating the object for **Trainer** class to subsequently call the ***train()*** method for training the selected model.

4.3.2 Model

This abstract class serves as the base class for adapting and implementing suitable models based on the data category of the task such as univariate, multivariate, and multistep ahead in TSF. Also this class holds *'hyperParameters'* variable which facilitates run time configuration of model hyperparameters.

Child Classes :

1. **UnivariateModel**: This class extends the **Model** abstract class and implements models recommended for univariate time series analysis. User can also define a custom model if the use case necessitates.

2. **MultiVariateModel**: This class is responsible for implementing the models recommended for multivariate time series analysis such as neural network, random forest, and gradient boost. User can also define a custom models if the use case necessitates.

3. **MultistepModel**: This class implements the standardized models for multi-step ahead forecasting such as neural network, state space and ensemble based models.

4.3.3 Trainer

This class is mainly responsible for training the selected model by feeding the model selected along with the hyperparameters and train data.

Attributes

1. *'trainData'*: Holds the training data.

Methods

1. ***train():*** Performs the training process.

4.3.4 Modeler Component Workflow

Below is a detailed explanation providing a step-by-step walkthrough of the expanded activity diagram Figure 4.5.

1. Start: The evaluation process begins.

2. Access Train Data: The train data, which is ready for training the model, is accessed.

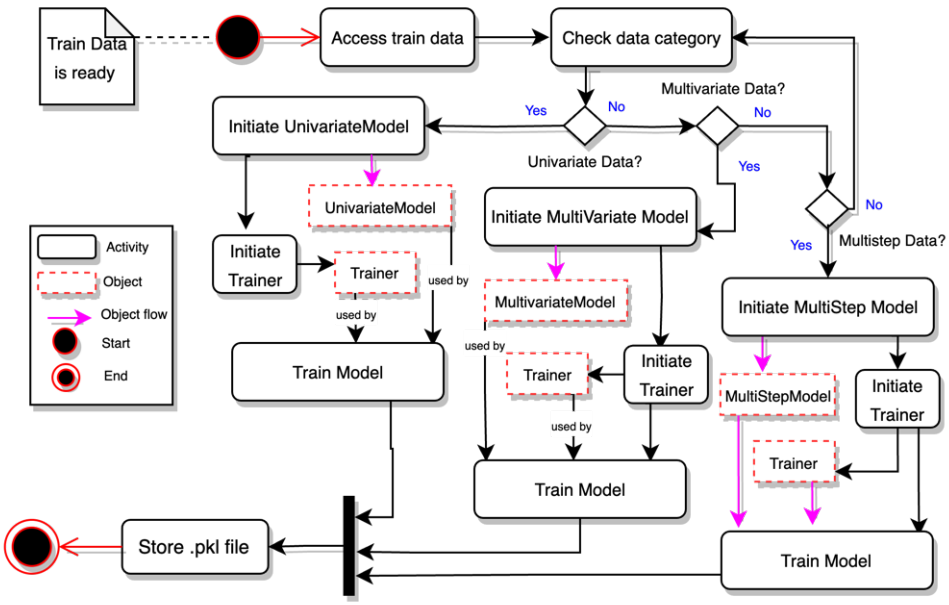


Figure 4.5: Activity diagram for model component

3. Decision: A decision point is reached to determine the type of category. If the category type is "Univariate", the flow proceeds to the "Univariate" branch. If the category type is "Multivariate", the flow proceeds to the "Multivariate" branch. Otherwise, the flow proceeds to the "Multistep" branch.

4. Univariate Branch:

   - Instantiate **Univariate** abstract class: An object of the **Univariate** abstract class is created.
   - Access Relevant Model: The relevant model for the univariate category is accessed.
   - Train the Model: The model is trained using the training data.

5. Multivariate Branch:

   - Instantiate **Multivariate** abstract class: An object of the abstract class **Multivariate** is created.
   - Access Relevant Model: The relevant model for the multivariate category is accessed.

- Train the Model: The model is trained using the training data.

6. Multistep Branch:

   - Instantiate **Multistep** abstract class: An object of the abstract class **Multistep** is created.
   - Access Relevant Model: The relevant model for the multi-step category is accessed.
   - Train the Model: The model is trained using the training data.

7. Store Model as .pkl (Python pickle file) File: Once the training process is complete, the trained model is stored as a.pkl file (Python pickle file is a useful Python tool that allows you to save ML models) for future use or deployment.

8. Stop: The evaluation process ends.

4.4 Evaluation Component

The evaluation component is a key component of the architecture, allowing the evaluation of the model performance and finding the key issues in the raw data. It allows practitioners to reuse pre-trained models, adapt them to specific needs, such as parameter adjustments for tuning purposes, and achieve accurate and reliable forecasts in diverse domains. As shown in Figure 4.6 the evaluation component of the architecture consists of the **Evaluator** class, which serves as the central class for coordinating the evaluation process.

4.4.1 Evaluator

The **Evaluator** class serves as a critical component in the TSF framework as it evaluates the performance of trained models and the statistical properties of the raw data to understand what data cleaning and transformation methods are required to make data clean and processed. This enables the users to assess and improve the forecasting models.

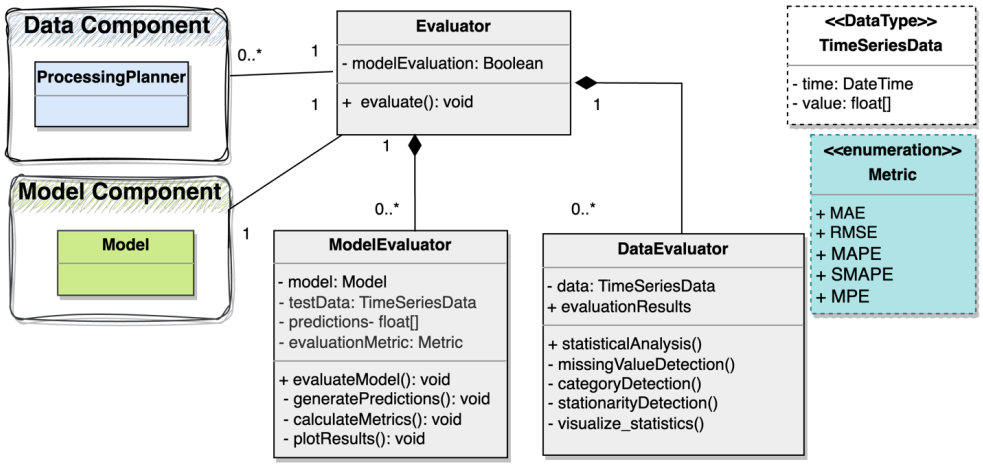


Figure 4.6: Class diagram for evaluation component

Attributes

1. *'modelEvaluation'*: This attribute is a Boolean flag indicating whether the model evaluation is required or data evaluation. So if this attribute holds true, then as depicted in Figure 4.1b, the **Evaluator** class initializes the object for **ModelEvaluator**; otherwise, the object for **DataEvalator**.

Relationships

1. **Evaluator** and **ModelEvaluator**

   The **Evaluator** class exhibits composition relationships with the **ModelEvaluator**. It enables the **Evaluator** class to access the methods and capabilities implemented within the **ModelEvaluator**, facilitating the evaluation of the trained model.

2. **Evaluator** and **DataEvaluator**

   The **Evaluator** class exhibits composition relationships with the **DataEvaluator**. It enables the **Evaluator** class to access the methods and capabilities implemented within the **DataEvaluator**, which provides statistical evaluation of the raw data.

Methods

1. ***evaluate():*** This method triggers the evaluation process. It orchestrates the evaluation by calling the appropriate methods within the **ModelEvaluator** and **DataEvaluator** classes.

4.4.2 ModelEvaluator

The ModelEvaluator class is responsible for evaluating the performance of the forecasting model. It utilizes various evaluation metrics and techniques to assess the model's accuracy, precision, recall, or other customized, relevant performance indicators. This class encapsulates the functionality for model evaluation. It has three attributes: *'trainedModel'*, *'testData'*, and *'predictions'*. The *'trainedModel'* attribute represents the path for trained time series model, while the *'testData'* attribute represents the testing data to be used for evaluation. The *'predictions'* attribute stores the predictions made by the model.

Attributes

1. *'trainedModel'*: This attribute refers to the forecasting model that is being evaluated. It represents the trained model instance that has been previously fitted on the training data.

2. *'testData'*: This attribute represents the labeled data that is used for evaluating the model. It typically consists of a set of input features (time series data) along with their corresponding true values (labels) for the evaluation period.

3. *'predictions'*: It hold the predicted values generated by the model.

4. *'evaluationMetric'*: It refers to the specific evaluation metric provided by the user to assess the performance of the model, such as RMSE, MAE, and Squared Error etc.

Methods

1. ***evaluateModel()***: This method is responsible for evaluating the performance of the forecasting model. It internally calls the ***generatePredictions()*** method to generate

predictions using the model and test data. Then, it calls the ***calculateMetrics()*** method to calculate the evaluation metrics based on the predicted and true values.

2. ***generatePrediction()***: This method is responsible for generating predictions using the trained forecasting model. It takes the testData as input and utilizes the model's predict() or forecast() function to compute the predicted values for the evaluation period.

3. ***calculateMetrics()***: This method calculates the evaluation metrics based on the predicted values and true values from the test data. It utilizes the *'evaluationMetric'* attribute, which represents the chosen evaluation metric, to calculate the specific metric for assessing the model's performance.

4. ***plotResults()***: It facilitates the visualization of evaluation results and predictions. It may generate plots, charts, or graphs to illustrate the performance of the forecasting model.

4.4.3 DataEvaluator

The **DataEvaluator** facilitates the assessment of quality, characteristics, and suitability of the raw data for TSF. This class employs statistical analysis, data visualization, or other evaluation techniques to examine the data's properties, identify any anomalies or inconsistencies, and provide recommendations for data cleaning, processing and refinement.

Attributes

1. *'data'*: This holds the raw data that needs to be evaluated.

2. *'evaluationResults'*: This attribute holds the results obtained after the data evaluation process (**DataEvaluator**). It stores the statistical analysis results in three aspects: data cleaning, data transformation methods required, and visual representations of these two. it enables easy access and retrieval of the analysis outcomes and sharing with **ProcessingPlanner** class for further decision-making.

Methods

1. ***missingValueDetection()***: It is used to detect the missing or irrelevant values in the dataset. and generates the appropriate statistical report for this to be used by the user to decide which data cleaning method to adopt to handle it.

2. ***categoryDetection()***: This method focuses on detecting categorical features in the dataset. This information is important for selecting appropriate encoding methods and handling categorical data in the forecasting process.

3. ***stationarityDetection()***: Stationarity is a key concept in time series analysis. This method performs tests and checks for trends, seasonality, and other patterns in the data to determine if it is stationary or requires preprocessing steps to achieve stationarity.

4. ***visualizeResults()***: This method is responsible for generating visualizations of the statistical analysis results performed on the data.

Overall, the evaluation component of the framework offers assessment of the model performance and the requirement for various data cleaning and transformation methods.

4.4.4 Evaluation Component Workflow

Below detailed explanation provides a step-by-step walkthrough of the expanded activity diagram Figure 4.7, highlighting the flow of the data evaluation and model evaluation branches and the key actions performed within each branch.

1. Start: The evaluation process begins.

2. Decision: A decision point is reached to determine the request type. The flow proceeds to the "yes" branch if the request type is for data evaluation. Otherwise, it proceeds to the "no" branch.

3. Data Evaluation Branch:

    - Instantiate DataEvaluator: The DataEvaluator class is instantiated to perform data evaluation.

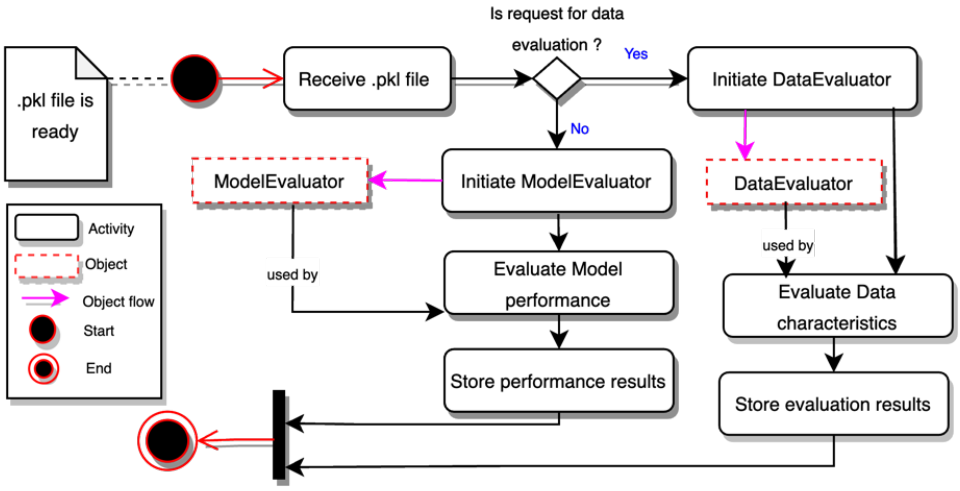


Figure 4.7: Activity diagram for evaluation component

- Call DataEvaluator methods: The necessary methods of the DataEvaluator class are called to perform the data evaluation process. These methods may include data preprocessing, statistical analysis, feature engineering, or any other relevant data evaluation tasks.

4. Model Evaluation Branch:

   - Instantiate ModelEvaluator: The ModelEvaluator class is instantiated to perform model evaluation.
   - Call ModelEvaluator methods: The required methods of the ModelEvaluator class are called to conduct the model evaluation process. These methods may involve model training, performance metrics calculation, prediction analysis, or any other relevant model evaluation tasks.

5. Perform data evaluation or model evaluation: In both branches, the respective evaluation tasks are executed. This step involves applying the appropriate techniques, algorithms, or methodologies to effectively evaluate the data or model.

6. Generate results and visualization report: The results may include performance metrics, statistical summaries, insights, or any other relevant findings from the evaluation

process. The visualization report may consist of plots, charts, graphs, or any other visual representations to aid in understanding the evaluation outcomes.

7. Store results and report in attribute: The results and visualization report are stored in an attribute within the DataEvaluator or ModelEvaluator object. This attribute serves as a container to hold the evaluation outcomes for further processing or presentation.

8. Return results and report to the user: Finally, the stored results and visualization report are returned to the user. This allows the user to access and analyze the evaluation outcomes, make informed decisions, or use them for reporting purposes.

9. Stop: The evaluation process concludes.

4.5 Advantages

Reusability & Adaptability: Adaptability and reusability can be defined at two levels, run time configuration (allowing to add custom methods or opt for different defined operations within the class keeping the definition originality intact of the facilitating class) and at definition level (changing the actual definition or implementation logic of the facilitating class). This framework offers run time adaptability, means user can add custom operations or opt for various defined operations based on use case requirements.

The proposed UML class diagram promotes re-usability and adaptability in several ways, making it suitable for a wide range of TSF problems, including multivariate-multistep forecasting. Here are some key benefits:

1. **Modularity**: The framework is designed with modular components that can be easily reused and combined in different configurations. Each component, such as **'ProcessingPlanner'**, **'Modeler'**, **'Evaluator'**, and **DataEvaluator**, encapsulates specific functionality and can be independently developed, tested, and reused in different forecasting projects.

2. **Abstraction**: The framework utilizes abstract classes and interfaces, such as **Model**, **'ProcessingMethod'**, and **'CleaningMethod'**, to define common behaviors and provide extensibility. These abstractions allow users to create new concrete configuration specific to their forecasting problem. For example, implementing the appropriate abstract classes or interfaces can easily add new model types, transformation techniques, or cleaning methods.

3. **Configuration and Parameters**: The framework incorporates parameterization and configuration options to customize the behaviour of its components. For example, the **Trainer** class has a *hyperParameters* field that allows users to map various model-specific parameters which are specific to the model type. This run time reconfigurability enables the framework to be adapted to different forecasting problems without requiring code modifications.

By combining these features, It allows users to reuse and extend existing components, integrate new techniques and models easily, and configure the framework to meet specific forecasting requirements. The framework's modular design, abstraction, composition, and flexibility make it well-suited for a wide range of TSF problems, promoting code reuse and reducing efforts by avoiding receptions.

4.5.1 Reusable and Adaptable Components

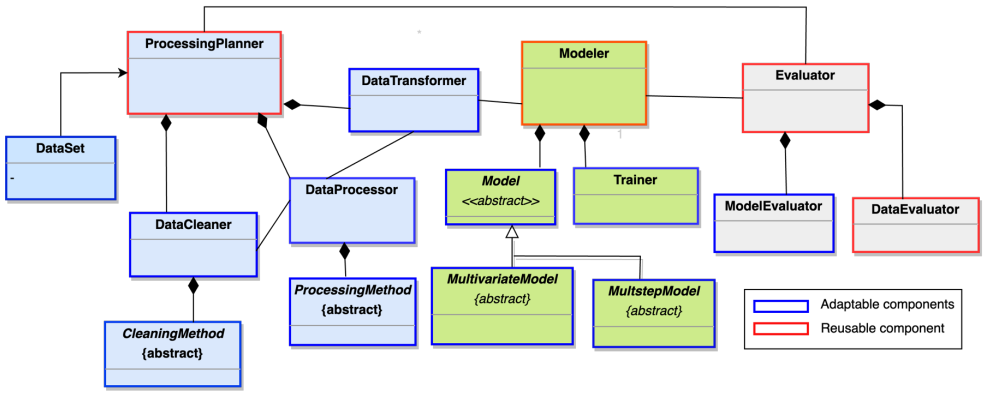


Figure 4.8: Depiction of reusable and adaptable components

Figure 4.8 depicts the minimum reusable and maximum adaptable components in the proposed framework. It offers at least 4 class components (25%), highlighted in red, that can be reused in any TSF pipeline due to its fixed behaviour (no custom methods are allowed) across the forecasting pipeline such as **ProcessingPlanner**, **Modeler**, **Evaluator** and **DataEvaluator**. The rest other 11 classes (75%) can be customized, such as addition of user defined operations or selection of defined operations based on use case requirements or run time parameterize configuration..

4.6 Architecture Validation

RQ 1

Can we design a model-based software architecture to support component adaptability and reuseability for end-to-end time series forecasting pipelines?

4.6.1 Metrics

To address research question 1, we utilize two separate research projects, one conducted in academia (Chapter 5) and another in the industrial (Chapter 6) domain. These projects are developed and investigated as part of the master's research and serve as case studies to validate our framework. The inclusion of these case studies allowed us to examine and validate our framework on three different metrics, as explained in the following sections:

1. **Reusability:** Reusability is another important aspect in TSF pipelines. It directly proportionate with the productivity of a practitioner involved in multiple TSF implementation. and reflects the degree of code reuse in the implementation of different projects. To assess this, a reusability metric is calculated by comparing the total number classes with fixed methods in multiple implementation by total number of core classes present in the framework.

   Target reusability threshold: To assess the degree of reusability, a target threshold is established. For the proposed framework, the thesis aims to achieve a minimum

target threshold of 25% based on Section 4.5.1, which means that at least 25% of the classes (4/15) of the core software architecture need to be reused in the end-to-end implementation of two or more TS projects. Equation 3 represent the formula used to calculate the reusability.

$$\text{reusability} = \left(\frac{\text{No. of classes with fixed or common behaviour}}{\text{No. of total classes in the framework}} \right) \times 100 \qquad (3)$$

2. **Adaptability:** Adaptability allows domain specific adjustments in implementing the time series techniques when working on two or more projects with different domains. In the context of the framework it reflects the degree of flexibility it provides in introducing user defined functions in the classes or have flexibility to opt different existing defined functionalities based on the use case requirement. To asses the degree of adaptability, an adaptability metric is calculated by comparing the total number classes which allow to define new functions or provide flexibility to choose different functions based on use case requirement by total number of core classes present in the framework. Equation 3 represent the formula used to calculate the adaptability while comparing two or more projects.

$$\text{adaptability} = \left(\frac{\text{No. of classes with custom and different methods}}{\text{No. of total classes in the framework}} \right) \times 100 \qquad (4)$$

Target adaptability threshold: As we have a fixed number of classes that provide the functionality discussed above, hence here we define the highest degree of adaptability this framework has to offer which is 75%, based on Section 4.5.1, as depending on the projects, adaptable components can also be reused contributing to reusability metric overall.

3. **Completeness:** The completeness of the time-series forecasting pipeline is a fundamental aspect to consider. It reflects the extent to which the framework covers the necessary classes and components required for implementing a functional pipeline. In this validation process, a completeness metric is calculated by comparing the total num-

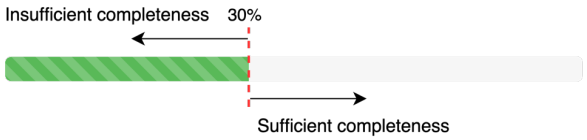


Figure 4.9: Target threshold for completeness

ber of components covered in the case studies with the total number of components present in the framework.

$$Completeness = \left(\frac{\text{No. of classes covered}}{\text{No. of total classes in the framework}} \right) \times 100 \tag{5}$$

Target completeness threshold: To assess the level of completeness, a minimum target threshold is established. For the proposed framework, the thesis aims to achieve a target threshold of 30%, which means that at least 30% of the classes of the core software architecture need to be used in the case studies. This threshold ensures a comprehensive inclusion of the essential components in the framework. If the implemented completeness exceeds the established threshold of 30%, it indicates a higher degree of completeness (effective in employing complex TSF pipeline). On the other hand, a completeness value below the threshold would imply the need for further enhancements and adjustments in the framework.

The validation of reusability and adaptability is demonstrated in the following section, showcasing the utilization of common classes in two diverse case studies and highlighting their adaptability to meet specific requirements. The completeness validation is discussed towards the end of Chapter 5 and Chapter 6. We highly recommend readers to refer to Chapter 5 and Chapter 6 before going in the next section for a complete understanding of the implementation of the case studies.

4.6.2 Reusability and Adaptability Validation

Figure 4.10 shows the comparison of operations used in the two case studies cs1 and cs2 from Chapter 5 and Chapter 6. There are a total of 7 classes (**Dataset**, **DataCleaner**, **Clean-**

ingMethod, **DataProcessor**, **ProcessingMethod**, **Model**, and **MultivariateModel**) that have used at least one operation different than the others. Hence these classes reflects the adaptability behaviour. While classes (**ProcessingPlanner**, **DataTransformer**, **Modeler**, **Trainer**, **Evaluator**, **ModelEvaluator**, and **DataEvaluator**) have been used as it is alonh with the similar operations, therefore these classes reflects the reusability. Following we asses the reusability and adaptability metrics.

Adapted
Reused

Component	Classes	Operations(CS1)	Operations(CS2)
Data Component	DataSet	customAggregation()	customAggregation()
	ProcessingPlanner	evaluationResults	evaluationResults
	DataCleaner->CleaningMethod	cleanData()	cleanData()
		MissingValueHandler->interpolate()	MissingValueHandler->interpolate()
		DataNormalizer->min-Max()	DataNormalizer->min-Max()
		DataEncoder	OutlierHandler
	DataProcessor->ProcessingMethod	processData()	processData()
		FeatureGenerator->custom()	FeatureGenerator-max()
		FeatureSelector->featureImportance()	FeatureSelector->pearsonCorrelation()
			Non-StationarityHandler
	DataTransformer	convertToSequence, fixedSplit	convertToSequence, fixedSplit
Model Component	Modeler	modelSelection(), trainModel()	modelSelection(), trainModel()
	Model->MultivariateModel	NeuralNetwork	NeuralNetwork
			SVM
	Trainer	train()	train
Evaluation Component	Evaluator	evaluate()	evaluate()
	ModelEvaluator	evaluateModel()	evaluateModel()
		generatePrediction()	generatePrediction()
		calculateMetrics()	calculateMetrics()
		plotResults()	plotResults()
	DataEvaluator	statisticalAnalysis()	statisticalAnalysis()

Figure 4.10: Demonstration of adaptable and reusable components in case studies

Reusability

RQ1.1

How effective the framework is with respect to reusability?

Overall, from the Figure 4.10, we can conclude there are 7 classes out of 15 core classes that has been used as it as with similar defined operations in both the case studies. Figure 4.11 depicts reusable and adaptable behaviour in a class diagram format. From Equation 3 the reusability metric is as follows:

$$Reusability = \left(\frac{7}{15}\right) \times 100 = 46.3\% \approx 45\%$$

Thus $\approx 45\%$ classes were reused from the framework to implement CS1 and CS2.

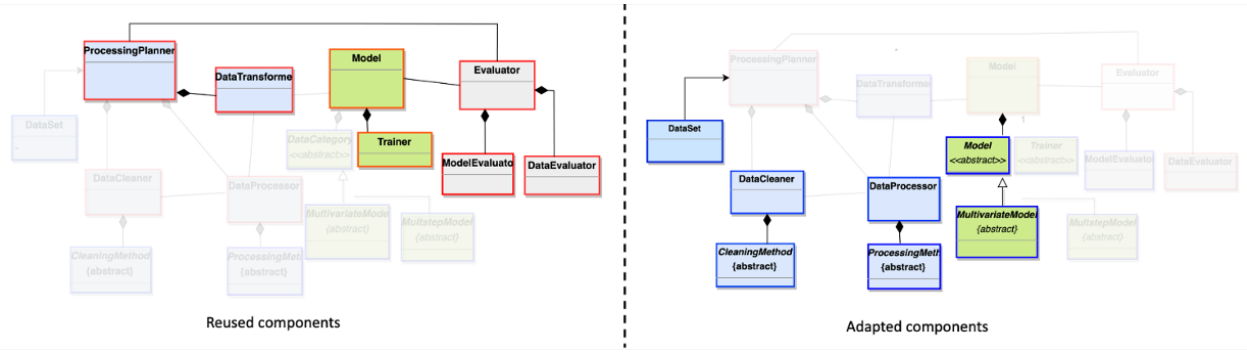


Figure 4.11: Framework's reusability demonstration on case studies

Adaptability

RQ1.2

How effective the framework is with respect to adaptability?

$$Adaptability = \left(\frac{7}{15}\right) \times 100 = 46.6\% \approx 45\%$$

Overall, as demonstrated from Figure 4.10 and Figure 4.11 there are 7 that have shown adaptable behaviour with the use of at least one different operation. Hence using the Equation 4 the adaptability metric is as below, Hence there are $\approx 45\%$ classes has been adapted from the framework as per the use case requirement in CS1 and CS2.

4.7 Summary

In this chapter, we propose a model driven TSF framework based on UML class diagram that offers reusability and adaptation of core class components based on the specific use case requirements in case of multiple implementation of TSF pipeline. We also showed that the framework offers a minimum of 4 classes that can be reused while other 11 classes can be adapted. Finally we validated the framework's adaptability and reusability based on the metrics (Equation 3 and Equation 4) decided by comparing the two case studies implemented in Chapter 5 and Chapter 6. The results demonstrated that the framework offered 7 core class components ($\approx 45\%$) were reused while other 7 core class components ($\approx 45\%$) were adapted by using at least one different operation or configuration. The rest 10% classes were not used.

Chapter 5

Student Performance Prediction

Objective 2

1. Develop a TSF pipeline to predict student performance.
2. Validate the performance and effectiveness of the TSF model.

The earlier versions of the work in this chapter has been presented and published at the following conferences:

- Proceedings of the 23rd international conference on Computer supported collaborative learning - ISLS'23 (to be published) [12].
- Presented at Society for 40th Annual Teaching and Learning Higher Education (STLHE) 2022.

5.1 Introduction

Learning is known to be challenging and stressful for incoming undergraduate STEM students, especially given the workload [39]. Navigating this transition to university-level learning can be overwhelming, highlighting the need for effective support systems to promote academic success.

As part of this research, we tracked students as they progressed through the SciLearn program in two large introductory science courses and collected self-reported scores on inventories that assess their learning strategies, metacognitive awareness, mindset, and misconceptions about how the brain works, discussed in Section 2.3.1. This data was collected right before the completion of the program together with demographic data, uptake of atomic habits, and progressive engaggement with course content grades from McGill's Learning Management System (LMS). Our data inventory so far includes over 400 students in an introductory organic chemistry course and over 500 students in an introductory psychology course.

ML is a promising tool for analyzing complex patterns, and recent research shows its potential to help students become self-regulated learners [64] [65]. Previous research has applied ML to predict student performance in higher education using demographics and cumulative GPA [50].

With this study, we aim to leverage recent work in ML and employ a TSF algorithm to predict the academic performance of students from "Organic Chemistry" and "Introduction to Psychology" courses, using course engagement (learning analytic), progression data and learning profile of the students and also investigate how effective it would be compared to the current pedagogical based method.

5.2 Study Overview

This section provides an end-to-end brief overview of the experiment conducted to predict student performance using TSF. As reflected in Figure 5.1 the overall study can be divided into three main stages, depicted as dashed red lines: Data Collection, Initial Data Preparation, and TSF. Each stage plays a crucial role in capturing, preparing, and analyzing the data to gain insights into student behaviour and predict academic outcomes.

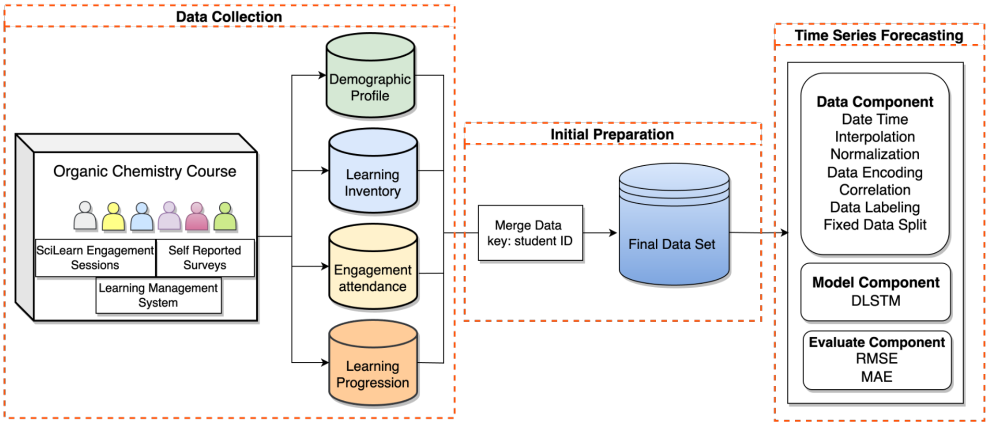


Figure 5.1: Study overview pipeline

5.2.1 Stage 1: Data Collection

As reflected in Figure 5.1 this is the first stage of the pipeline; in this study, we collaborated with two challenging introductory undergraduate courses: *Organic Chemistry* and *Introduction to Psychology*. Students were incentivised to participate by offering a 1-2% bonus in the course.

The SciLearn program began with orientation sessions. These sessions aimed to familiarize students with the program's objectives and principles, providing them with evidence-based learning and self-regulation techniques. Peer collaboration activities were scheduled throughout the term to foster engagement and collaborative learning among participating students. These two platforms served as the primary sources for collecting content engagement data in specific courses, including attendance and adopted atomic habits introduced during orientation sessions. We gathered valuable information on student active involvement and commitment by leveraging these logistics.

In addition to the platform-based data, we employed surveys in the form of quizzes to collect self-reported information from the students. These surveys covered various aspects, including demographic details and learning inventories (LI). By incorporating self-reported data, we aimed to gain insights into student perceptions of their learning approaches, metacognitive awareness, mindset, and potential misconceptions Section 2.3.1

about how the brain works. This rich dataset allowed us to explore the relationship between expected and actual performance.

Through the collaborative efforts of the SciLearn program, the course platforms, the learning management system, and the self-reported surveys, we collected the data from more than 800 participating students on course engagement, academic progression, and self-reported profiles. These diverse data sources form the foundation for our subsequent analyses and give us a holistic understanding of the factors influencing student performance.

5.2.2 Stage 2: Initial Preparation

As shown in Figure 5.1 this stage primarily deals in consolidating and harmonizing the diverse datasets collected during the first stage. It involves merging data from multiple sources, including demographic profiles, LR, engagement with the course content, SciLearn program attendance, and early grades. By integrating these datasets, we aim to create a comprehensive, unified dataset that captures the various aspects of student behaviour and academic performance. By merging the data based on *'student_id'* field as the key, we ensure that a single view represents each student profile. This unified dataset is the foundation for subsequent steps in the TSF pipeline, enabling us to explore the relationships between different variables and their impact on student performance.

5.2.3 Stage 3: Time Series Forecasting

The last stage, TSF, marks the initiation of the actual ML pipeline. Following the conceptual proposed framework in Chapter 4, we have divided this stage into three key components: Data Component, Model Component, and Evaluation Component as depicted in Figure 5.2.

1. **Data Component**: It implements various statistical analyses, data cleaning, transformation, and splitting operations such as DateTime conversion, null values handling, normalization, encoding, correlation, and data labelling on the unified dataset obtained from Section 5.2.2. This ensures the dataset is appropriately prepared for TSF.

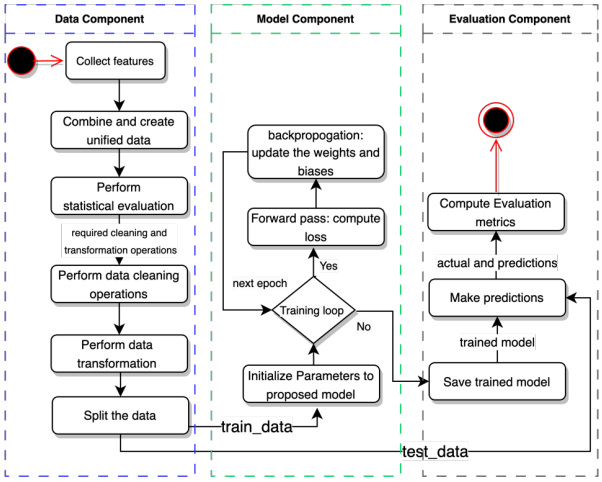


Figure 5.2: Overall activity diagram for the case study

2. **Model Component**: It selects appropriate forecasting models and trains them using the prepared training dataset. For this study, we have used various variants of Gated Recurrent Unit (GRU) and Long Short Term Memory (LSTM) to train our time series model, and then used the trained data as an output artifact captured from the Data Component.

   Training a model involves various steps such as defining model hyperparameters, fitting the models to the training data and optimization if required. The trained models capture the underlying patterns and dependencies within the data, enabling them to make predictions on unseen datasets. We'll discuss these steps in detail concerning our study in the later section of this chapter.

3. **Evaluation Component**: The Evaluation Component assesses the performance of our trained GRU and LSTM forecasting models using an independent testing dataset kept separate during data split and accessed from Data Component. We have used evaluation metrics such as mean absolute error (MAE) and root mean square error (RMSE) to quantify the performance of our model and the accuracy of the predictions.

In subsequent sections, we delve into the specific techniques, algorithms, and methodologies employed within each TSF stage.

5.3 Prediction Methodology

The methodology is divided into three subsections: Data Processing Pipeline depicting the **ProcessingPlanner** class, Model Development and Training simulate **Modeler**, and Model Performance Assessment maps **Evaluator** class behaviour from the proposed framework in Chapter 4.

5.3.1 Data Processing Pipeline

A. Data Features and Aggregation

Features: As depicted in Figure 6.4, we have captured 18 features reflecting demographic, learning inventory, SciLearn program attendance, habits adopted, engagement with course content, and the early performance of a student profile.

The dataset encompasses a range of information as shown in Figure 5.3, capturing demographic characteristics, LR, engagement data, and learning progression. These features collectively provide a comprehensive view of student profiles and behaviours, enabling a holistic analysis of their academic performance.

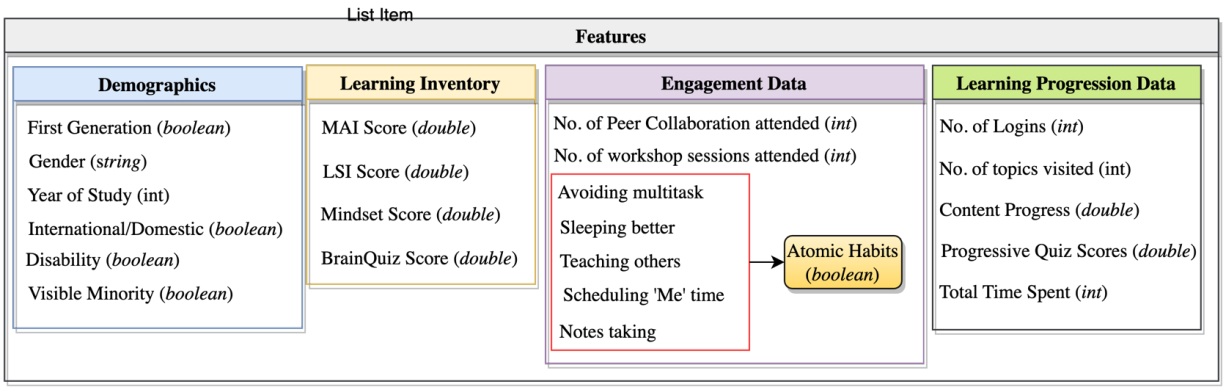


Figure 5.3: Features collected from SciLearn

1. Demographic: Following features are collected as part of this category.

1a. *First Generation*: A binary indicator representing whether the student is the first in their family to attend university.

1b. *Gender*: Categorization of students based on their gender identity.

1c. *Year of Study*: Indicates the student academic year.

1d. *Status*: Distinguishes between international and domestic students.

1e. *Disability*: Indicates if the student has a documented disability.

1f. *Visible Minority*: A categorical variable capturing the visibility of minority status.

2. Learning Inventories (LI): LI provide insights into student learning strategies, metacognitive awareness, mindset, and misconceptions about how the brain works. The following LR are included in the dataset.

2a. *Learning Strategies Inventory (LSI) Score*: A numerical value ranging from 0-11 based on the response on the LSI survey questionnaire.

2b. *Meta-cognition Awareness Inventory (MAI) Score*: A numerical value ranging from 0-19 based on the response on the MAI survey questionnaire.

2c. *Mindset Score*: As discussed in mindset section: The collective score from 30 questionnaire on the survey to understand students mindset.

2d. *BrainQuiz Score*: This feature reflects the students misconceptions about how brain works. The score ranges from 0-10.

3. Engagement Data: This data reflects student participation and involvement in various activities within the SciLearn program. The engagement features in the dataset comprise.

3a. *Attendance in engagement activities*: Number of peer-collaboration and orientation sessions attended by a student.

3b. *Atomic habits*: The binary outcome (1:adopted or 0:not adopted) of each of the adopted atomic habits.

4. Learning Progression Data: Through LMS system Section 2.3.1 we capture the following features.

4a. *Logins*: The number of times a student has logged in to the LMS for a specific course within the examined week.

4b. *Topics visited*: Total number of topics students have navigated in a course within the examined week.

4c. *Content progress*: Percentage of content coverage for a course.

4d. *Assessment score*: Scores of quizzes, assignments or any other weekly tasks on a scale of 0-100.

4e. *Time spent*: Total minutes spend within the LMS for a specific course in a given week.

The rationale behind incorporating these diverse data features is to explore the relationships between student characteristics, engagement patterns, and learning progression to predict student performance.

Aggregation: The collected raw data was organized and aggregated into a suitable time series dataset as reflected in Figure 5.6 to apply TSF and predict student final grades. The dataset includes information for each student, indexed by a time component, week, that represents the different time points throughout the term. Each row in the dataset corresponds to a specific student at a specific time index, capturing their corresponding assessment score, demographic information, LI, engagement data, and learning progression. For example, in Figure 5.6, we have two students (001 and 002) and their corresponding data for each week (Time Index) over a term of 16 weeks. Organizing the data in this manner makes it feasible to apply TSF algorithms and analyze the temporal patterns and relationships among the various factors that influence student performance.

B. Data Evaluation

For data evaluation, various tests for statistical properties were performed. Table 5.1 shows the mean and existing value range of raw features. Here are a few observations from the evaluation:

1. 120 missing values were identified, most of which belonged to the LI and learning analytics data.

	Background		Learning Inventories				Engagement Data		Learning Analytics			Grades
Time Index	Student ID	Demographic Score	MAI	LSI	BrainQuiz	Mindset	Atomic Habits	Attendance	Content Interaction	Content progress	Assesment scores	
1	ID1	3	5	6	5	14	3	0	20	10%	0	-
2	ID1	3	5	6	5	14	3	0	40	22%	60%	-
3	ID1	3	5	6	5	14	3	1	60	25%	65%	-
...	...	...	...	...	...	...	...	...	...	...	...	
16	ID1	3	7	8	7	20	5	7	899	90%	95%	82%
1	ID2	0	9	7	4	22	1	0	10	11%	0	
2	ID2	0	9	7	4	22	2	1	40	30%	95%	-
...	...	...	...	...	...	...	...	...	...	...	...	-
16	ID2	0	10	9	8	28	4	6	900	100%	100%	92%

Figure 5.4: Demonstration of data aggregation

Feature	Value range	Mean
LSI	0-11	6
MAI	0-19	14
Mindset	0-30	17
BrainQuiz	0-10	6
Atomic habits	0-5	2
Orientation attendance	0-4	2
Peer-collaboration	0-5	1.8
Content covered	**0-100**	**80**
Content interaction	**0-1000**	**600**
Assessment score	**0-100**	**60**

Table 5.1: Features statistics

2. As highlighted in Table 5.1, there is a significant difference in the value range of features such as *content interaction*, *content progress*, and *assessment scores* from other predictor features. Hence data normalization needs to be employed.

3. Features with categorical values such as *disability* and *residence status* required an appropriate encoding mechanism.

C. Data Cleaning

Based on the above observations following data-cleaning operations were employed:

1. **Missing values**: To handle missing values in the dataset, two approaches were employed: replacing missing values with zero and using interpolation techniques, based on the nature of the missing values and the specific features they represented.

   For variables where blank values indicated no activity during a particular week, such as learning analytics, missing values were replaced with zero.

   In contrast, certain features, such as LI and engagement data, were collected twice during the term: once at the beginning and again towards the end (11th week). These features aimed to capture the impact of the SciLearn program on student LI scores. To handle missing values in these cases, interpolation was applied. This ensured that the values remained constant until they changed, providing a more accurate representation of the data over time.

2. **Data Normalization**: As reflected in Table 5.1, content progress, assessment scores, and content interaction values vary greatly, hence to transform these features into a consistent and comparable scale with other features, we used min-max scalar to fit it under 0-10 range.

3. **Encoding**: We employed one hot encoding to encode binary category features such as disability and status. For instance, in the status feature, "International" status was encoded as (1), while the "domestic" status was represented as (0). We ensured that the resulting numerical representations were suitable for further analysis and predictive modeling.

D. Data Processing

1. **Feature Engineering**: A new feature called *demographic score* was created for the demographic profile of a student. This score was derived by aggregating various demographic attributes such as *visible minority*, *disability*, *first-generation status*, and *international status*. The rationale behind this aggregation was to capture the overall

demographic profile of each student while condensing multiple features into a single representative score.

Additionally, we applied a similar feature engineering approach to the atomic habits data. We generated an aggregated feature, referred to as the *atomic habit score*, by combining relevant atomic habit features. This score aimed to capture the collective influence of various atomic habits on student learning and academic outcomes.

2. **Correlation Analysis and Feature Selection**:

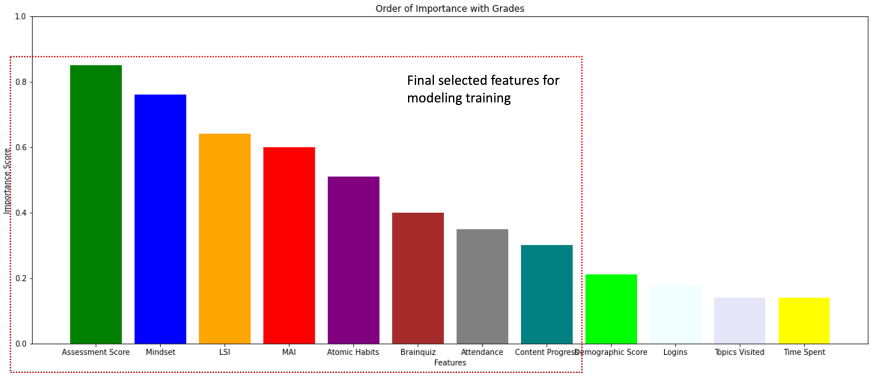


Figure 5.5: Order of feature importance with grades

Random forest feature importance was analysed to determine the importance of collected features in predicting grades. The resulting analysis yielded valuable insights into the relative significance of factors influencing grades. The results are visualized in Figure 5.5, showcasing the importance order with corresponding importance scores. The graph incorporates twelve key features (features are mentioned with the importance score) , including *assessment score:0.85*, *mindset:0.76*, *lsi:64*, *MAI:0.62, atomic habits:0.51*, *brainquiz:0.42*, *attendance:0.35*, *content progress:0.23*, *demographic score:0.18*, *logins:0.14*, *topics visited:0.12*, and *time spent:0.11*. After a thorough discussion with domain experts, we selected the top 8 features based on this score as the final features for modeling, thereby dropping *Demographic Score*, *Logins*, *Topics Visited*, and *Time Spent*. This is justifiable because the dropped features are highly

dependent on students' studying pattern and behaviour while online, which may skew the results.

E. Data Transformation

		Learning Inventories				Engagement Data		Learning Analytics			Grades
Time Index	Student ID	MAI	LSI	BrainQuiz	Mindset	Atomic Habits	Attendance	Content Interaction	Content progress	Assessment scores	
1	ID1	5	6	5	14	3	0	0.2	1.0	0	-
2	ID1	5	6	5	14	3	0	0.4	2.2	6.0	-
...	...	...	...	...	...	...	...	...	...	...	-
15	ID1	6	8	7	20	5	6	6.9	8.5	7.5	-
16	ID1	7	8	7	20	5	7	8.99	9.0	9.5	8.2
1	ID2	9	7	4	22	1	0	0.10	1.1	0	
2	ID2	9	7	4	22	2	1	0.40	3.0	9.5	-
...	...	...	...	...	...	...	...	...	...	...	-
15	ID2	10	9	8	28	4	6	9.0	9.5	9.6	
16	ID2	10	9	8	28	4	6	9.79	10.0	10.0	9.2

Figure 5.6: Demonstration of data transformation into training samples

- **Conversion to sequence data**

  To align our data with TSF techniques and facilitate the training of our models, we performed lag based data transformation on the original aggregated dataset. The initial shape of the data was (900x16, 9), where 900 are number of students, 16 is number of weeks, and 9 is the total features used for prediction. However, we needed to restructure the data in a specific format for TSF. Therefore, we applied a lag of 16 to create sequential input data for each student as reflected in Figure 5.6. This lag-based transformation involved creating a vector of data spanning 16 weeks associated with one target variable (final grades). As a result, the updated shape of the transformed data became (900, 9, 1) representing training samples, total features utilized, and the target variable. This data transformation allowed us to align the dataset with the requirements of TSF models and enabled us to capture the temporal dependencies and patterns inherent in the data.

- **Data Split**

  There are several approaches to splitting the data, including random splitting, temporal

splitting, and fixed data splitting. In the context of time series data, where the order and temporal dependencies of the data points matter, a fixed data split approach is preferable because it maintains the temporal order, avoids information leakage, and enables a realistic evaluation of the model's forecasting performance [66]. Hence we have used fixed data split with 70-30% ratio, leveraging 630 sequence samples for 16 weeks for training while 270 sample sequences for testing.

5.3.2 Model Development and Training

Architecture: Stacked LSTM and GRU

Variants of LSTM and GRU models were trained by including a dropout layer to evaluate performance improvement. The study uses a stacked architecture with double layers for the basic uses of LSTM and GRU. The detailed design of the proposal is shown in Figure 5.7. The dropout layer was the main difference between (a) and (b) in Figure 5.7.

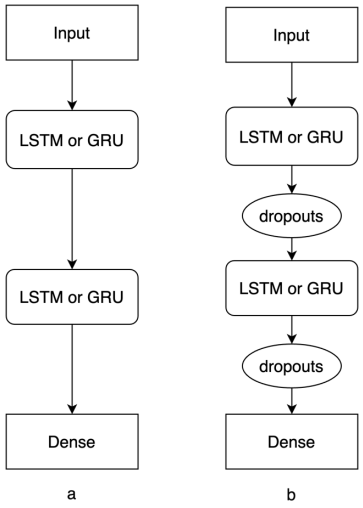


Figure 5.7: Proposed architecture, (a) without dropout, (b) with dropout

Training

Based on [67] and [68], batch size affects model generalization and optimization, we chose this as one of the hyperparameters. We used ReLu [69] as an activation function, which is

commonly used in deep learning models because of its simplicity and effectiveness in dealing with the vanishing gradient problem. We applied this to the output of each LSTM unit

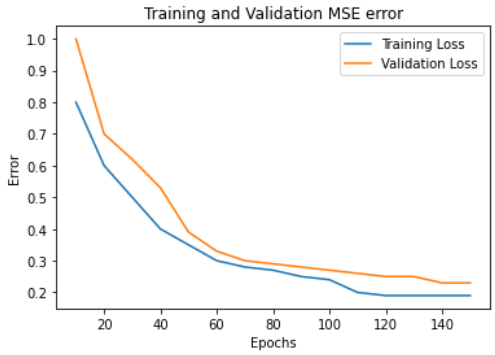


Figure 5.8: Training and validation loss (MSE)

to introduce non-linearity and allow the model to learn complex patterns in the data. The model was trained on the 70% data sequences hence leveraging 630 sequences of student data for 16 consecutive weeks. We have used mean squared error (MSE) [70] as the loss function as we try to minimize the larger errors. A learning rate scheduler (explained later in Section 6.3.2) was also employed for faster convergence of loss function. As demonstrated in Figure 5.8 it took 120 epochs for the loss to converge on the training dataset while for validation it took 140 epochs.

5.4 Experimental Validation

RQ 2

How to develop effective TSF techniques for predicting student performance?

For an in-depth investigation of RQ2, we address two specific research questions on selecting the best TSF model (RQ2.1) and then evaluating its actual effectiveness (RQ2.2) as discussed in the next two subsections.

5.4.1 Model Performance Assessment

RQ 2.1

Which are the most effective machine learning models for predicting student performance?

Metrics calculation

We assessed the performance of our models using two evaluation metrics: MAE and RMSE. As depicted in Figure 5.9, the calculation of these metrics can be described in three steps as follow:

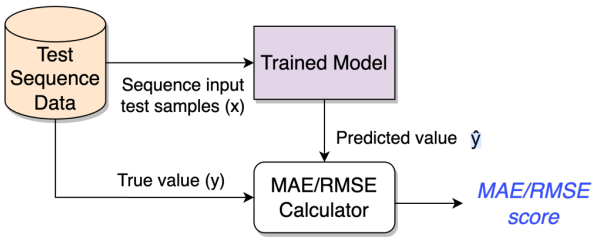


Figure 5.9: Overview of MAE and RMSE calculation

1. The input samples (x) from the test dataset, which were kept separate during the data split, are fed into the trained model.

2. The model then generates the predicted values ($\hat{y}$) for these input samples (x). Access the true values (y) of the input samples (x).

3. Use Equation 2 and Equation 1 to calculate the MAE and RMSE scores respectively.

The assessment is performed on the test dataset. Table 5.3 presents the results of these evaluations, along with the corresponding accuracy values. Please note the accuracy scores are calculated after mapping the final grades into grade letters used for the collaborated courses as shown in Table 5.2.

We tested two variants among the GRU models: one without dropout and another with dropout. For the GRU without dropout, we observed an RMSE of 0.61 and 0.58 for batch

Numerical scale	Grade letter
10.0-8.5/8.4-8.0	A/A-
7.9-7.5/7.4-7.0/6.9-6.5	B+/B/B-
6.4-6.0/5.9-5.5	C+/C
5.4-5.0/4.9-0	D/F

Table 5.2: Mapping of grade letters

Model	Batch size	RMSE	MAE	Accuracy
GRU(b)	32/64	0.61/0.58	0.48/0.41	40.2/48.3%
GRU(a)	32/64	6.54	3.9	35.1/39.8%
LSTM(a)	32/**64**	0.32/**0.31**	0.30/**0.28**	81.2/**86.5%**
LSTM(b)	32/64	0.38/0.36	0.37/0.35	74.1/77.8%

Table 5.3: Performance comparison of different variants of GRU and LSTM models

sizes 32 and 64, respectively. The MAE scores were 0.48 and 0.41, with the accuracy score of 40.2% and 48.3% for the respective batch sizes. On the other hand, the GRU model with dropout exhibited slightly higher RMSE and MAE values (0.67/0.69 and 0.56/0.52) and lower accuracy (35.1/39.8) compared to the model without dropout.

For the LSTM (Long Short-Term Memory) models, again two variants without dropout and with dropout were tested. The LSTM model without dropout demonstrated superior performance, with lower RMSE (0.32/0.31) and MAE (0.30/0.28) values compared to the LSTM model with dropout. Additionally, the LSTM without dropout accuracy was substantially higher at 81.2% and 86.5% for the respective batch sizes, while the LSTM model with dropout achieved slightly lower accuracy values of 74.1% and 77.8%.

These results indicate that the stacked LSTM models generally outperformed the GRU models, regardless of the presence of dropout. The LSTM models without dropout achieved the highest score with 86.5% accuracy, indicating better overall predictive performance.

Findings

Figure 5.10 demonstrates the execution of our best-performing model LSTM(a) on test data. We observed that the model tends to under-predict and over-predict with a pattern as described below:

1. **Under prediction**: The model tends to overpredict the grades of students with the final grade score of 30-61% and 81-100%, with a mean MAE of 0.25, which is a better score than the average performance.

2. **Over prediction**: The model tends to underpredict the grades of students with a final grade score of 61-81%, with a mean MAE of 0.30, which is a slightly bad score than the average performance.

The students with low score grades are the focus of our study and this model performs better in that specific region than the overall average performance.

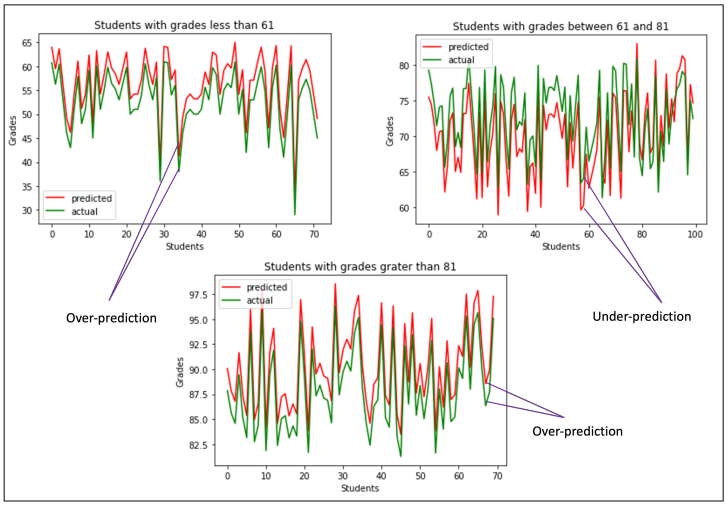


Figure 5.10: Evaluation on the validation set with three grade categories

5.4.2 Assessment of effectiveness

RQ 2.2

How effective are TSF techniques for predicting student performance?

Assessment Metric

In order to asses the effectiveness of our top performing model, we conducted a comparative analysis of our approach compared to the traditional method in estimating grades based on the LI recommended range scale discussed in Section 2.3.1. For this comparison, we randomly selected a subset of 10 students from our dataset for validation purposes. This subset consisted of three students with D grades, two with F grades, three with A grades, and two with B grades as depicted in Figure 5.11. We estimated the grades for these students using two approaches: the traditional approach based on the LI recommended range and our best-performing trained model, which incorporated various features discussed in the case study.

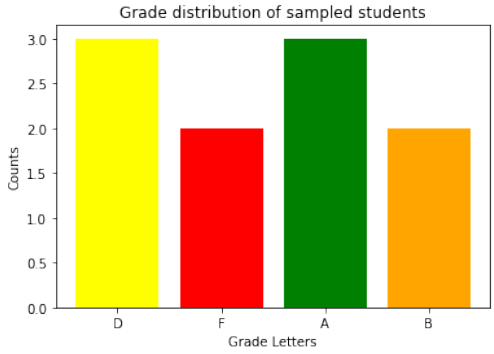


Figure 5.11: Grades distribution of randomly sampled students

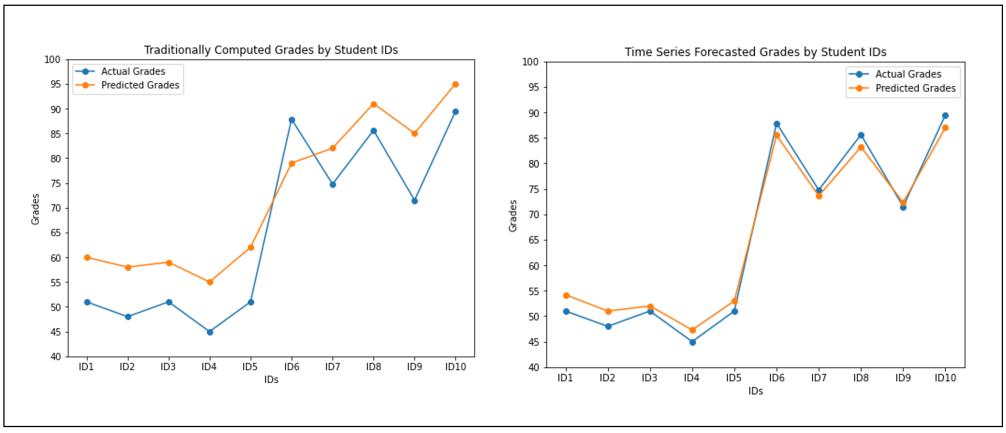


Figure 5.12: Comparing the predicted results of tradition approach vs TSF for 10 randomly selected students

Results and findings The results of this comparison, as illustrated in Figure 5.12, highlighted the disparities between the two approaches in estimating actual grades. On average, with the traditional approach, we observed a tendency to overestimate the grades by a significant margin of 6-8%. This overestimation was consistent across both low-performing and high-performing students. In contrast, when using our trained model, we found that the estimates for low-performing students still exhibited a slight overestimation but with a reduced average absolute scale of 3-4%. This discrepancy further decreased to 2-3% for high-performing students. Please note these comparisons are in % on the scale of 1-100 rather than 1-10 point scale.

The comparison of the two approaches revealed notable differences in their accuracy and performance. The traditional approach tended to overestimate grades across the board and was time-consuming to perform on an individual scale. The trained model showed improved estimation accuracy, especially for low-performing students in significantly less time. The reduction in overestimation by our model for low performers indicates its ability to capture the most critical section of the class, which is also one of the key goals of the SciLearn program.

5.5 Architecture Completeness Validation

RQ1.3

How effective is the proposed framework in terms of completeness?

In line with addressing RQ1, as discussed in Section 4.6, we use Section 1.2.3 to asses the completeness of proposed framework. Figure 5.12 demonstrates the coverage of various classes and elements used to implement the full scale TSF pipeline for this case study.

Below we briefly discuss the mapping and role of framework's component with the actual implementation of TSF technique to predict student performance.

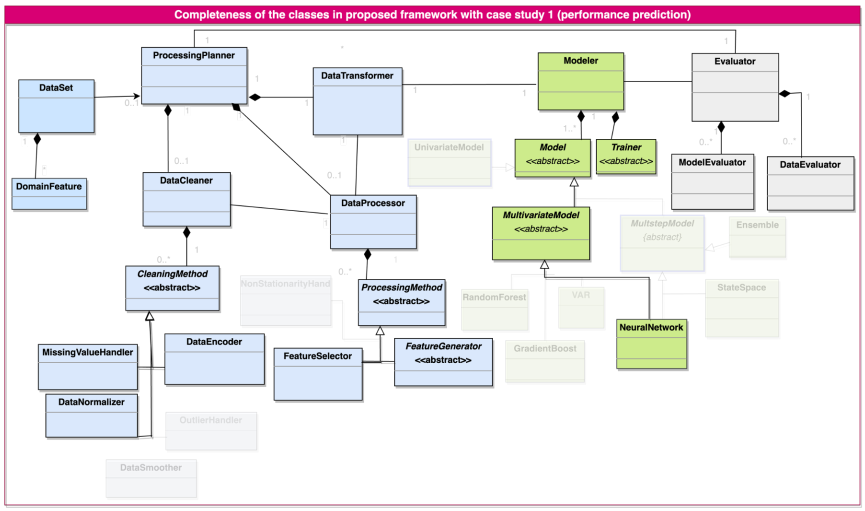


Figure 5.13: Completeness validation of framework (CS1)

1. **ProcessingPlanner**

   DataSet: This class facilitates the feature collection and data aggregation as discussed in Section 5.3.1.

   Evaluator: Once the data is aggregated, this class helps in evaluating the data quality along with some time series specific statistical checks discussed in Section 5.3.1.

   DataCleaner: Based on the inputs from **Evaluator** class, **ProcessingPlanner** class helps in facilitating various data cleaning methods through **DataCleaner**. For this case study we have used interpolation, zero value replacement, min-max and one hot encoding cleaning methods discussed in Section 5.3.1.

   DataProcessor: Based on the use case requirement this class helps in identifying the stationarity, feature engineering, and feature selection methods. This case study only leverages the feature engineering (as we create a new feature based on user defined logic discussed in Section 5.3.1), and feature selection using feature importance method as there is no trend or seasonality in the data.

   DataTransform: The case study covers this class to convert data samples to time series sequence data based on defined lags and number of features. And then uses

fixed split method to split the data into training and testing dataset as discussed in Section 5.3.1.

2. **Modeler, Model and Trainer:** As the data category of the case study is of multivariate nature, and the models selected for training (GRU and LSTM units) are sub-type of neural network as discussed in Section 5.3.2, hence **Modeler, Model and Trainer** class provide necessary interface for model selection and training.

3. **Evaluator:** The case study leverage this class to calcualte RMSE and MAE values along with generating the predictions on test data for trained model.

Overall, the experiment used 21 classes out of 32 classes in the framework hence based on the Equation 5, the coverage value calculates to:

$$Coverage = \left(\frac{21}{32}\right) \times 100 = 65.6\%$$

Overall, with ≈65% class coverage, this case study covers all the necessary steps and processes for implementing the end to end TSF pipeline. Hence it validates the minimum coverage value required (25%) as discussed in Section 4.6 to validate the completeness of the framework.

5.6 Summary

In this chapter, we investigated variants of stacked LSTM and GRU models to predict the student performance. In order to address RQ2.1, we evaluated the performance of different variants of mentioned models by using MAE and RMSE metrics and demonstrated the supremacy of stacked-LSTM over others with the lowest MAE of 0.28 (highest accuracy of 86.5%). Two interesting observations were found: 1. The model tends to overpredict the grades of students with the final grade score of 30-61% and 81-100%, with a mean MAE of 0.25, which is a better score than the average performance, 2. The model tends to underpredict the grades of students with a final grade score of 61-81%, with a mean MAE of

0.30, which is a slightly bad score than the average performance. To check the effectiveness of the proposed model (RQ2.2) with the traditional approach we did a random sampling of 10 students from test data and compared it to our top TSF model, results revealed notable differences of 4% improved accuracy of our proposed model. Finally we concluded with the validation of framework's completeness (RQ1.3) with the score of 65% which is higher than the threshold of 30%.

Chapter 6

Thermoacoustic Instability Prediction

6.1 Introduction

Objective 3

1. Develop a TSF pipeline to predict TA instability in AGT.
2. Validate the performance and effectiveness of the proposed TSF model.

TA instability in gas turbines refers to the phenomenon of undesirable pressure oscillations that can occur within the combustion chamber. These oscillations are caused by the interaction between the combustion process and the system's acoustics, leading to potentially harmful effects such as increased mechanical stress, reduced efficiency, and even damage to the turbine components.

Traditional early detection methods rely on manual inspection, such as empirical rules, experience-based methods, or limited sensor (flame and pressure) data, which may not provide timely or accurate information about impending instability. This can result in operational disruptions, unplanned downtime, and increased maintenance costs. Moreover, the complex nature of TA instability makes it time-consuming and challenging to detect and predict using conventional techniques alone.

This chapter presents a comprehensive experiment performed using TSF to predict acoustic pressure amplitude, which further assist domain engineers to predict TA instability in

gas turbines. The experiment encompasses key stages such as data collection, processing, model training, and model evaluation. Through this experiment, we also aim to investigate the effectiveness of the deep learning TSF approach and validate if it can contribute to developing reliable and efficient predictive models for amplitude predictions leading to a better early prediction of TA instability. The findings from the study have shown the potential and furter avenues of integrating of deep learning TSF techniques to enhance the operational stability and performance of gas turbines, leading to improved energy efficiency and reduced maintenance costs at Siemens. At the end, we also use this experiment as a case study for time seires forecasting pipeline to validate the coverage of our proposed conceptual framework in Chapter 4.

The study was carried out as a research collaboration with Siemens Energy. The experiment focused on predicting the acoustic noise amplitude at E frequency (*UCAN4E* and *UCAN8E*). We experimented with three different time series models, but to confine the scope of the thesis, we will only discuss and present the results of our proposed top-performing D-LSTM model.

This chapter is organized in the following manner below to examine how deep learning can be leveraged to improve the TA instability prediction compared to the traditional approaches used in the industrial setting and how the case study validates the proposed model-driven framework.

1. **Study Overview**: This section will comprehensively describe the high level architecture of the experiment which experimental setup, data collection and TSF.

2. **Methodology**: It outlines the approach employed for data processing, transformation and modeling to predict the E frequency acoustic amplitude using a time-series-based forecasting method and multivariate data collected during setup.

3. **Validation, Results and Findings**: This section will present the analysis outcomes, highlighting the forecasting model's performance and effectiveness compared to exist-

ing processes. Also at the end we validate the predictive model using the proposed architecture Chapter 4, emphasizing its coverage.

6.2 Study Overview

This section provides an end-to-end brief overview of the experiment to predict acoustic amplitude using TSF. As reflected in Figure 6.1, The overall study can be divided into three main stages, depicted as dashed red lines: Data Collection, Initial Data Preparation, and TSF. Each stage plays a crucial role in capturing, preparing, and analyzing the data to gain insights into the thermo and acoustic patterns to help predict acoustic amplitudes.

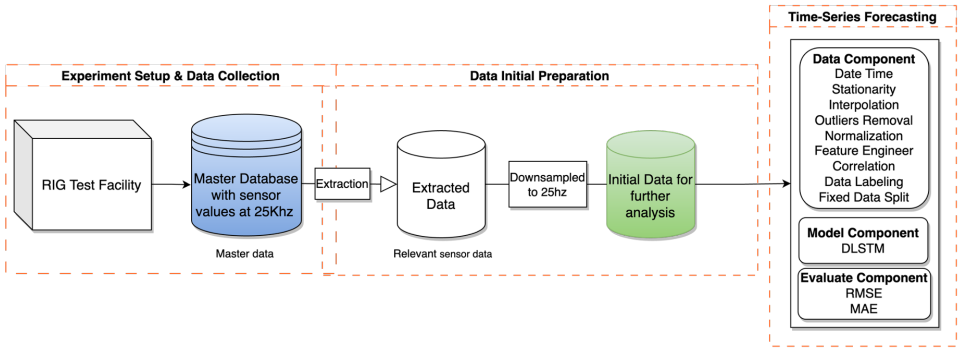


Figure 6.1: High-level experiment pipeline

6.2.1 Stage 1: Experimental setup and data collection

This section presents the experimental setup used to collect the data for TA instability predictive modeling. While the combustor's technical details are not the thesis's main focus, we provide a general overview of the setup used for data collection.

- **Test Rig Description** The experiments were conducted using test rig data collected as part of the annual simulation testing of gas turbine.
- **Instrumentation** Siemens utilized a range of instruments and sensors strategically placed within the test rig to capture the necessary data for our TSF pipeline.

- **Experimental Conditions** The experiments were conducted under a controlled set of operating conditions, with varied parameters such as fuel flow rate, air pressure, and equivalence ratio to explore a wide range of TA behaviours.

- **Data Acquisition** The system allowed simultaneous data collection from multiple channels with a sampling rate of 25 kHz. Hence the data for the study was derived from this master dataset. The dataset included readings from various sensors located at different engine parts. To ensure the relevance of the features used in the models, we consulted with domain experts at Siemens and extracted only the recommended features that could potentially impact or contribute towards instability in combustion for further processing.

6.2.2 Stage 2: Initial Preparation

The data collected during the rig test was extensive, comprising a vast number of samples captured at a high frequency of 25 KHz over a duration of approximately 50 minutes, resulting in more than 80 million individual samples as reflected in the Figure 6.2. However, processing and analyzing such large datasets can present significant challenges, including computational limitations and time constraints.

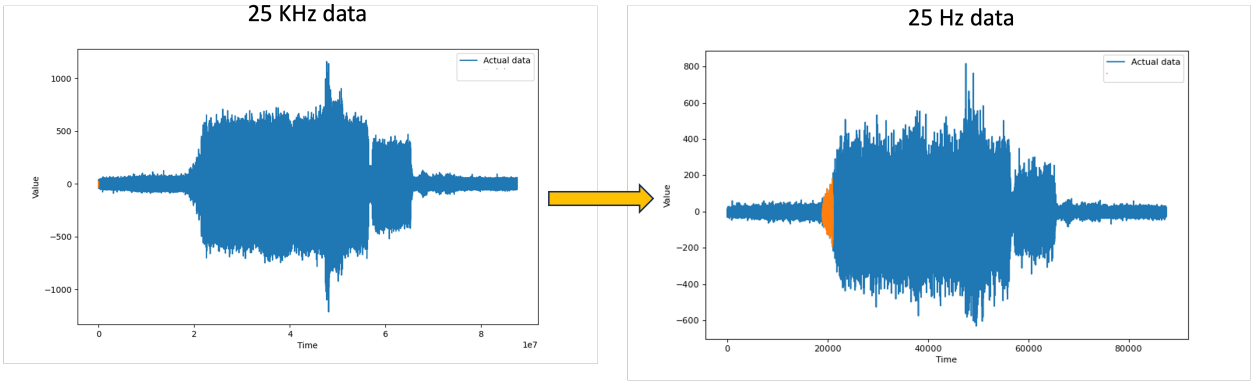


Figure 6.2: Demonstration of downsampling on a sample feature

To address these challenges, for the selected features, we strategically downsampled, without losing the pattern, the data to 25Hz from 25kHz frequency as depicted in Figure 6.2,

and focused on a representative subset of the data for our experimentation. By selecting a specific chunk of the dataset, we aimed to balance computational feasibility and maintain the integrity of the captured information. This approach allowed us to work with a manageable portion of the data while still capturing the essential characteristics and patterns of the TA instability phenomenon. Please note Figure 6.2 demonstrate the behaviour of downsampling, as the very initial data preparation stage, of only one of many captured sensor data. The selection of features and data subsets for initial analysis was made with the coordination of domain experts at Siemens and discussed in Section 6.3.1.

6.2.3 Stage 3: Time Series Forecasting

Finally, TSF marks the initiation of the actual machine learning pipeline; following the conceptual proposed framework in Chapter 4, similar to Figure 5.2 we have divided this stage into three key components: Data, Model, and Evaluation Component.

1. **Data Component**: It implements data evaluation and data cleaning methods such as DateTime conversion, null values handling, normalization, encoding, correlation, and data labelling on the unified dataset obtained from Section 6.2.2 based on inputs from data evaluation. These operations ensure that the dataset is prepared appropriately for TSF. Further, it implements the data processing steps, such as making data stationary, generating new features, and feature selection, followed by data transformation operations, such as reshaping the data to keep it in synch for training purposes and splitting the data.

2. **Model Component**: It implements the selected forecasting model and trains it using the prepared training dataset. In this study, we have employed Support Vector Regressor (SVR), Recurrent Neural Network (RNN) and Deep LSTM to train our time series model. We train the model using the training dataset and hyperparameters, utilizing the time-based historical data. As the D-LSTM is the top-performing model, we have confined the scope of the thesis to discuss the architecture and results of various

variants of D-LSTM neural network only. But we have used the other two models as the baseline for the comparison.

Training a model is an iterative process and involves various steps such as defining model hyperparameters, fitting the models to the training data and optimization if required. We'll discuss these steps in detail concerning our study in the later section of this chapter.

3. **Evaluation Component**: The Evaluation Component assesses the performance of the trained D-LSTM forecasting models using the independent testing dataset kept separate during the data split. We have used evaluation metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) to quantify the performance of our model and the accuracy of the predictions.

As we have already explained the operations performed in Stage 1and Stage 2, now in the subsequent sections, we expand our work to explain Stage 3, which reflects the actual implementation of TSF and adapts to the proposed architecture in Chapter 4. The methodology section is a collective representation of three components briefly described above.

6.3 Prediction Methodology

The methodology is divided into three subsections: Data Processing Pipeline depicting the **ProcessingPlanner** class, Model Development and Training employing **Modeler** class, and Model Performance Assessment simulate **evaluator** class behaviour from the proposed framework in Chapter 4. Each subsection is described in detail below:

6.3.1 Data Processing Pipeline

A. Data Features and Aggregation

The features shown in Table 6.1 relevant to the TA instability were extracted from the master database as initial data preparation discussed in Section 5.2.2. Table 6.1 show the features

and their brief description. It is important to note that the above feature names in Table 6.1

Parameters	Description
LHOUR, LMINUTE, LSECOND	The hour, minute, and second component of the timestamp for each data point.
UW31	The axial velocity.
UPFRDEM	The upstream fuel demarcation pressure.
AITMEAN	The mean temperature of the ambient air surrounding the engine.
UP1	The upstream pressure.
T31MEAN	The mean temperature.
P3MEAN	The mean pressure.
UTPZ, UTSZ	The velocity of the turbine primary, secondary zone flow.
UCAN4E, UCAN8E	The E frequency noise amplitude at the U chamber of the CAN4 and CAN8.

Table 6.1: Features extracted from master dataset

correspond to the specific naming convention provided by Siemens Energy for their sensors identification. Due to the collaborative nature of this study, the use of these specific names ensures accuracy and consistency in reporting the collected data. The aim is to predict the E frequency acoustic amplitude hence our focus for the target features are *UCAN4E* and *UCAN8E*. These are two acoustic amplitudes recorded at two different frequency levels 4E and 8E.

B. Data Evaluation

Considering the computation limitation, we selected the subset of 250000 unstable samples of 25Hz frequency data and then further downsampled it to 1Hz data to fit it into the one-second window, resulting in 10000 samples overall as depicted in Figure 6.3. This downsampling was done by taking the max amplitude value in a consecutive 25 seconds and assigning it to the one-second time step, as we needed to focus on the max noise amplitude of the sensor.

To understand the statistical properties of the selected features, we performed statistical analysis (min, max, median, data distribution, missing value analysis, outliers and trends). Results and handling of outliers and trends have been discussed later in the section. Fig-

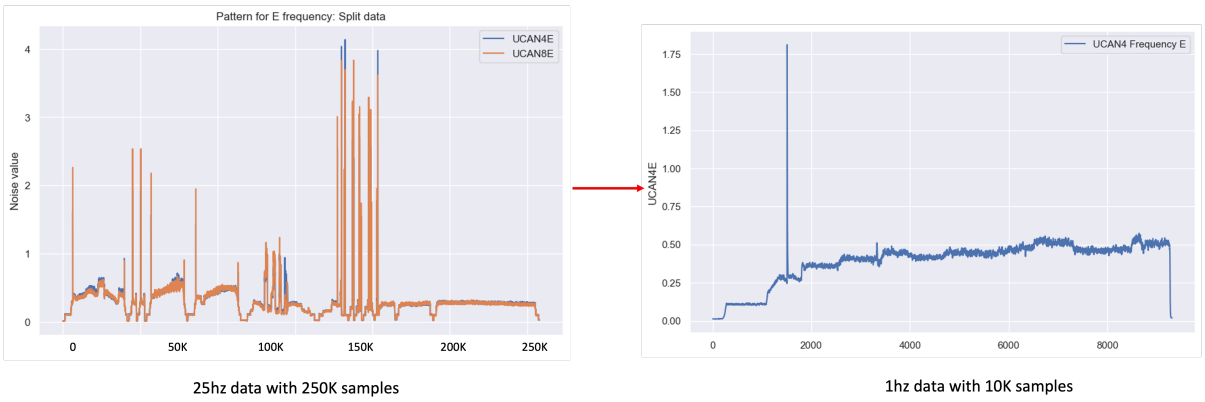
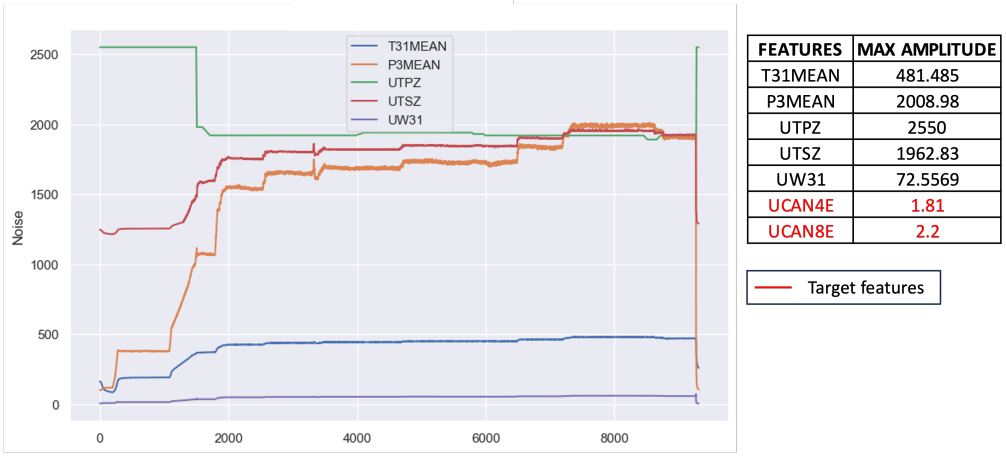


Figure 6.3: Demonstration of further downsampling from 25Hz to 1Hz using target feature

ure 6.4 reflects the data distribution of the features used in the prediction of target acoustic amplitude along with its statistical values. Here are a few observations for further processing of the data.



FEATURES	MAX AMPLITUDE
T31MEAN	481.485
P3MEAN	2008.98
UTPZ	2550
UTSZ	1962.83
UW31	72.5569
UCAN4E	1.81
UCAN8E	2.2

Figure 6.4: Data distribution of features (1Hz)

1. There is a significant difference in the value range of target features *UCAN4E*, *UCAN8E* and other predictor features. Hence data normalization would be required.

2. As shown in Figure 6.3, outliers are present; hence an appropriate outlier handling technique needs to be employed. We have discussed this in the next sections.

3. Data distribution shows a high potential of non-stationarity in the data; hence appropriate stationarity test needs to be evaluated, and if present non-stationarity handling methods need to be employed. We have discussed this in the next sections.

C. Data Cleaning

1. **DateTime Conversion** The first step involved converting the timestamp data into a standardized time format. This conversion allowed us to analyze and manipulate the temporal aspects of the data effectively as reflected on Figure 6.5.

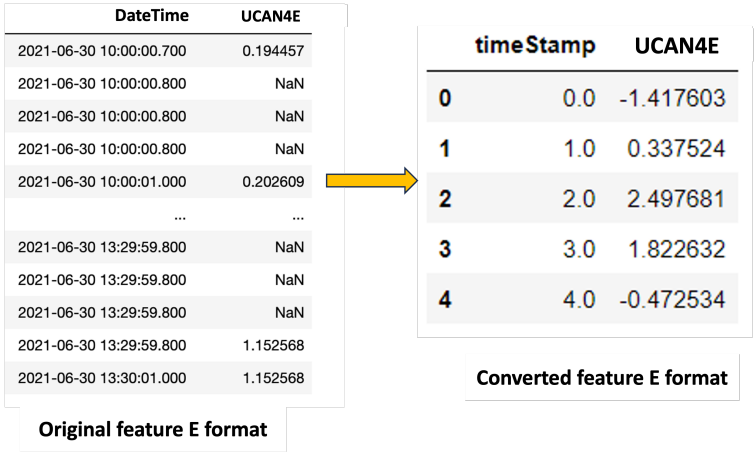


Figure 6.5: Depiction of date time conversion using target feature *UCAN4E*

2. **Missing values and Temporal Alignment:** Statistical analysis had shown missing values in the dataset and hence we employed a combination of interpolation techniques and zero-value replacement to address the missing values in the dataset. The one potential reason for missing values were the intermittent activation of certain sensors during combustion resulted in periodic blank values in the data. Here's how we handled this scenario:

   - **Interpolation:** as depicted in Figure 6.6, we used the interpolation technique to estimate and fill in the missing values for features with intermittent missing values.

```
In [125]: df_t05['Z0360A01_A05']

Out[125]: 0      0.166273
          1      0.172318
          2           NaN
          3           NaN
          4           NaN
                    ...
```

```
In [126]: df_A05_t05['Z0360A01_A05']

Out[126]: 0      0.166273
          1      0.172318
          2      0.172318
          3      0.162167
          4      0.159855
                    ...
```

Figure 6.6: Interpolation of missing data

- **Zero Value Replacement:** In addition to interpolation, we also replaced the blank values with zero. Since the inactive sensors during certain periods recorded no data, assigning a zero value can indicate the absence of any measurable quantity. With this, we preserve the temporal structure of the dataset and ensure that the model can differentiate between active and inactive periods of sensor data. The decision between interpolation and zero value replacement is taken after extensive discussion with domain experts to ensure data integrity.

3. **Data Scaling:** As already discussed in Section 6.3.1, in our dataset, we encountered

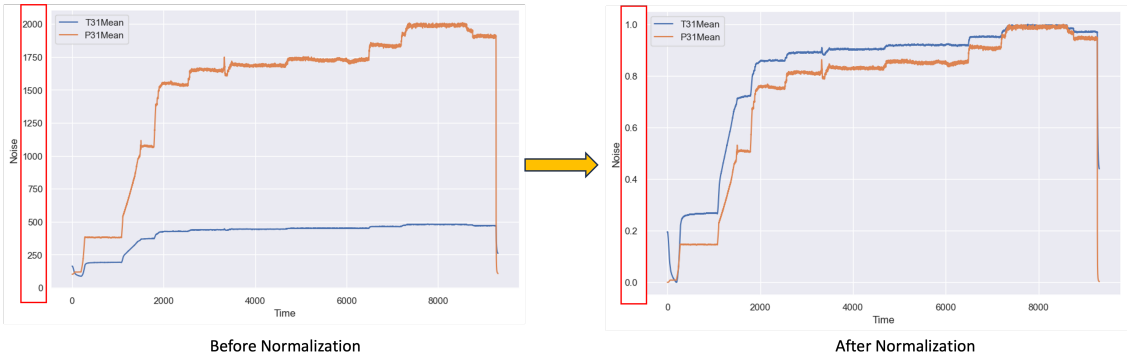


Figure 6.7: Before and after normalization of *T31MEAN* and *P31MEAN*

features such as *T31MEAN* and *P31MEAN* that had a much larger magnitude range as compared to other features as shown in Figure 6.7. Such variations in scale can lead to biased interpretations and impact the performance of machine learning models. To address this issue, we employed min-max normalization as discussed in Section 2.2.1 techniques.

4. **Outliers:** Detecting and removing outliers is an important step in data cleaning to ensure the accuracy and reliability of the analysis. As referenced in observations of Section 6.3.1, in our analysis, we encountered features with very few values falling beyond the distribution range of most data points for the same feature even after taking the max of consecutive 25 seconds. Hence, these values were considered outliers as they deviated significantly from the expected pattern. We considered the samples

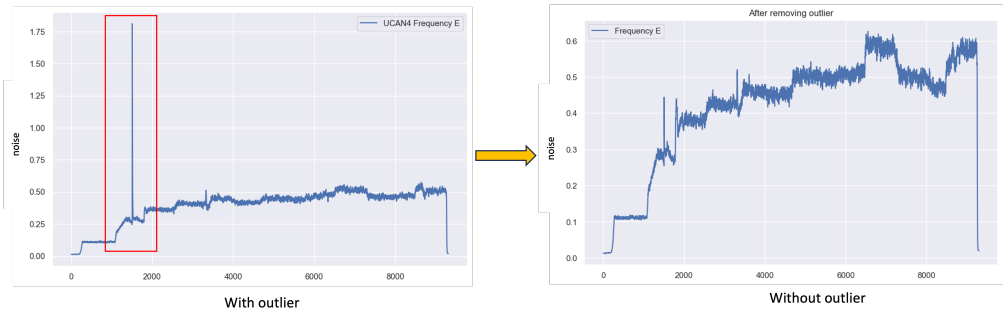


Figure 6.8: An example of outlier detection and removal in E frequency

three standard deviations away as the outliers and were subsequently removed from the dataset.

D. Data Processing

1. **Stationary Test:** The ADF test was conducted on the dataset to assess the stationarity of the features. The ADF test revealed that a few features depicted non-stationary behaviour with a p-value less than 0.5. Non-stationarity implies that the statistical properties of the time series, such as the mean and variance, are not constant over time. To address this issue and achieve stationarity, the first differencing method (difference between the consecutive values) was employed.

   By taking the first difference of the features with trends, the respective data was transformed into a new series where each value represents the change between consecutive observations. Figure 6.9 reflects the transformation of actual data to non-stationary data.

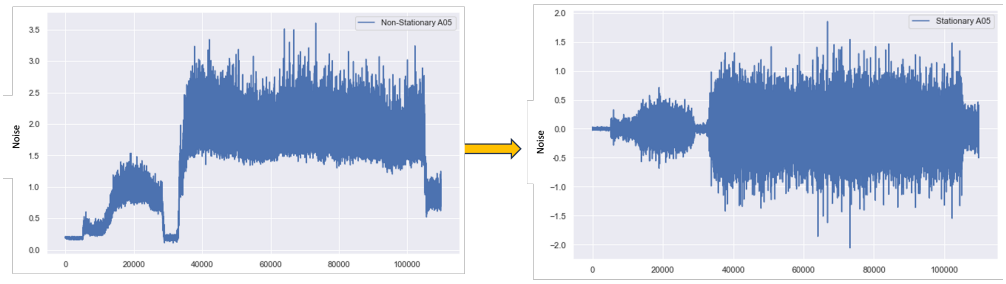


Figure 6.9: Demonstration of non-stationarity handling using A05

2. **Feature Engineering**: We created a new feature called UCAN-E. This step aimed

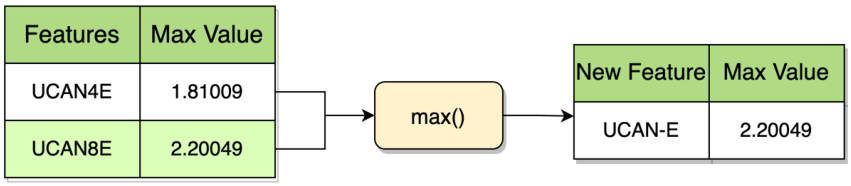


Figure 6.10: New target feature (*UCAN4-E*)

to capture the maximum amplitude value of the E frequency from both the sensors, which is crucial in predicting and understanding TA instability in combustion systems. This approach Figure 6.10 was based on the observation that the patterns exhibited by *UCAN4E* and *UCAN8E* were similar demonstrated in Figure 6.3, indicating a strong correlation between the two features.

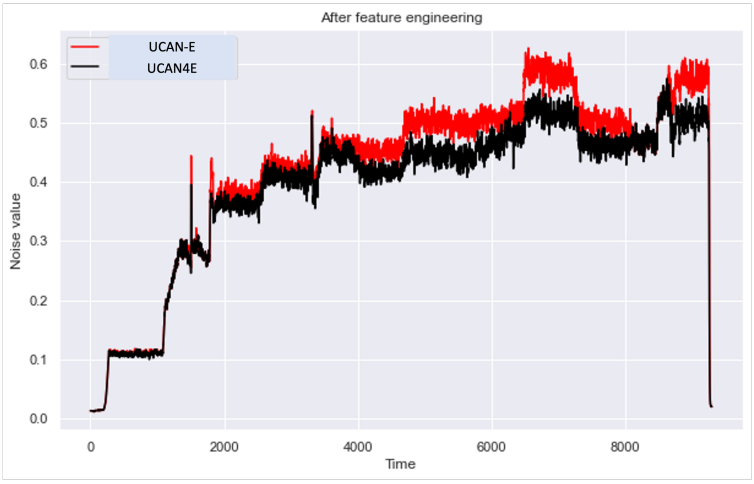
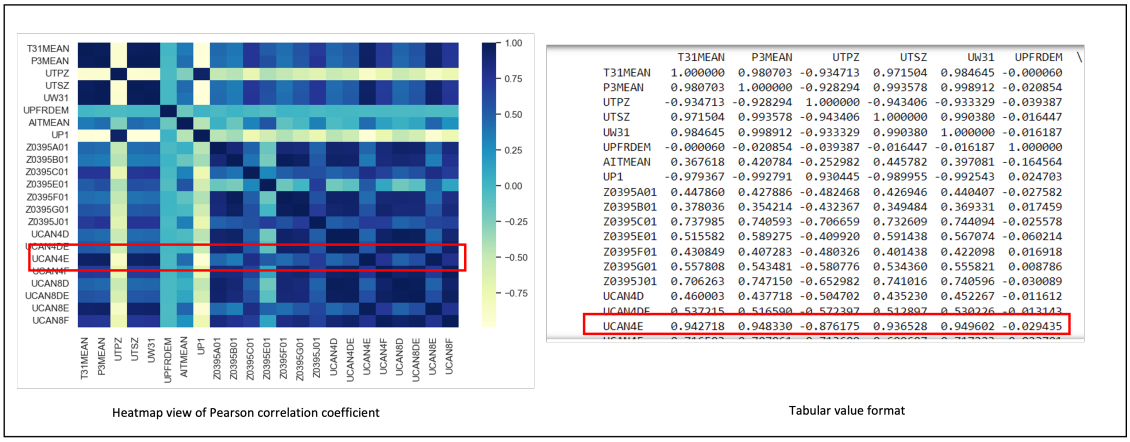


Figure 6.11: Noise distribution of *UCA4E* vs *UCAN-E*

By calculating the maximum value between *UCAN4E* and *UCAN8E* for each timestamp, we obtained a consolidated representation of the maximum amplitude of the E frequency. This new feature, *UCAN-E*, now serves as the target variable for our prediction models. We aim to capture the most significant and critical information related to the E frequency's impact on TA instability by focusing on the maximum amplitude value.

3. **Feature Selection**: We performed pearson correlation analysis to select the relevant features for modeling. Figure 6.12 reflects the correlation of original target feature (*UCAN4E*) before feature engineering while Figure 6.13 reflects the updated correlation scores after the new target feature *UCAN-E* is introduced. Table 6.2 highlights the improved correlations score of *UCAN-E* against original target feature *UCAN4E* with other feature.



	T31MEAN	P3MEAN	UTPZ	UTSZ	UW31	UPFRDEM
T31MEAN	1.000000	0.980703	-0.934713	0.971504	0.984645	-0.000060
P3MEAN	0.980703	1.000000	-0.928294	0.993578	0.998912	-0.020854
UTPZ	-0.934713	-0.928294	1.000000	-0.943406	-0.933329	-0.039387
UTSZ	0.971504	0.993578	-0.943406	1.000000	0.990380	-0.016447
UW31	0.984645	0.998912	-0.933329	0.990380	1.000000	-0.016187
UPFRDEM	-0.000060	-0.020854	-0.039387	-0.016447	-0.016187	1.000000
AITMEAN	0.367618	0.420784	-0.252982	0.445782	0.397081	-0.164564
UP1	-0.979367	-0.992791	0.930445	-0.989955	-0.992543	0.024703
Z0395A01	0.447860	0.427886	-0.482468	0.426946	0.440407	-0.027582
Z0395B01	0.378036	0.354214	-0.432367	0.349484	0.369331	0.017459
Z0395C01	0.737985	0.740593	-0.706659	0.732609	0.744094	-0.025578
Z0395E01	0.515582	0.589275	-0.409920	0.591438	0.567074	-0.060214
Z0395F01	0.430849	0.407283	-0.480326	0.401438	0.422098	0.016918
Z0395G01	0.557808	0.543481	-0.580776	0.534360	0.555821	0.008786
Z0395J01	0.706263	0.747150	-0.652982	0.741016	0.740596	-0.030089
UCAN4D	0.460003	0.437718	-0.504702	0.435230	0.452267	-0.011612
UCAN4DE	0.537215	0.516590	-0.572397	0.512897	0.530226	-0.013143
UCAN4E	0.942718	0.948330	-0.876175	0.936528	0.949602	-0.029435

Figure 6.12: Relationship analysis of E frequency with other features before feature engineering

In conjunction with person correlation analysis, which helps in understanding the linear relationship of the features, we also performed rolling window correlation in order to observe the average temporal dynamics (how the relationship evolves over different time stamps) and select an optimized window size based on the overall lagged relationship observation of features with the target feature.

	T31MEAN	P3MEAN	UTPZ	UTSZ	UW31	UPFRDEM	AITMEAN	UP1	UCAN-D	UCAN-E	UCAN-DE	UCAN-F
T31MEAN	1.000000	0.980830	-0.935001	0.971694	0.984728	-0.000127	0.367341	-0.979511	0.460180	0.947073	0.548307	0.717345
P3MEAN	0.980830	1.000000	-0.929103	0.993593	0.998918	-0.021001	0.420220	-0.992787	0.436619	0.958860	0.526580	0.707789
UTPZ	-0.935001	-0.929103	1.000000	-0.944363	-0.934016	-0.039421	-0.253335	0.931291	-0.506649	-0.876342	-0.584731	-0.715124
UTSZ	0.971694	0.993593	-0.944363	1.000000	0.990418	-0.016606	0.445181	-0.989967	0.434367	0.951896	0.523986	0.700840
UW31	0.984728	0.998918	-0.934016	0.990418	1.000000	-0.016316	0.396556	-0.992550	0.451381	0.958334	0.540287	0.717912
UPFRDEM	-0.000127	-0.021001	-0.039421	-0.016606	-0.016316	1.000000	-0.164760	0.024857	-0.011367	-0.044790	-0.012122	-0.023379
AITMEAN	0.367341	0.420220	-0.253335	0.445181	0.396556	-0.164760	1.000000	-0.433743	-0.043848	0.482382	0.024596	0.151716
UP1	-0.979511	-0.992787	0.931291	-0.989967	-0.992550	0.024857	-0.433743	1.000000	-0.454776	-0.961784	-0.544012	-0.718827
UCAN-D	0.460180	0.436619	-0.506649	0.434367	0.451381	-0.011367	-0.043848	-0.454776	1.000000	0.459306	0.992776	0.930099
UCAN-E	0.947073	0.958860	-0.876342	0.951896	0.958334	-0.044790	0.482382	-0.961784	0.459306	1.000000	0.556455	0.734018
UCAN-DE	0.548307	0.526580	-0.584731	0.523986	0.540287	-0.012122	0.024596	-0.544012	0.992776	0.556455	1.000000	0.963487
UCAN-F	0.717345	0.707789	-0.715124	0.700840	0.717912	-0.023379	0.151716	-0.718827	0.930099	0.734018	0.963487	1.000000

Figure 6.13: Correlation analysis of E frequency with other features after feature engineering

Feature	UCAN4E	UCAN-E
T31MEAN	0.942	**0.947**
P3MEAN	0.948	**0.958**
UTSZ	0.936	**0.951**
UW31	0.949	**0.958**

Table 6.2: Improved correlation score with feature engineering

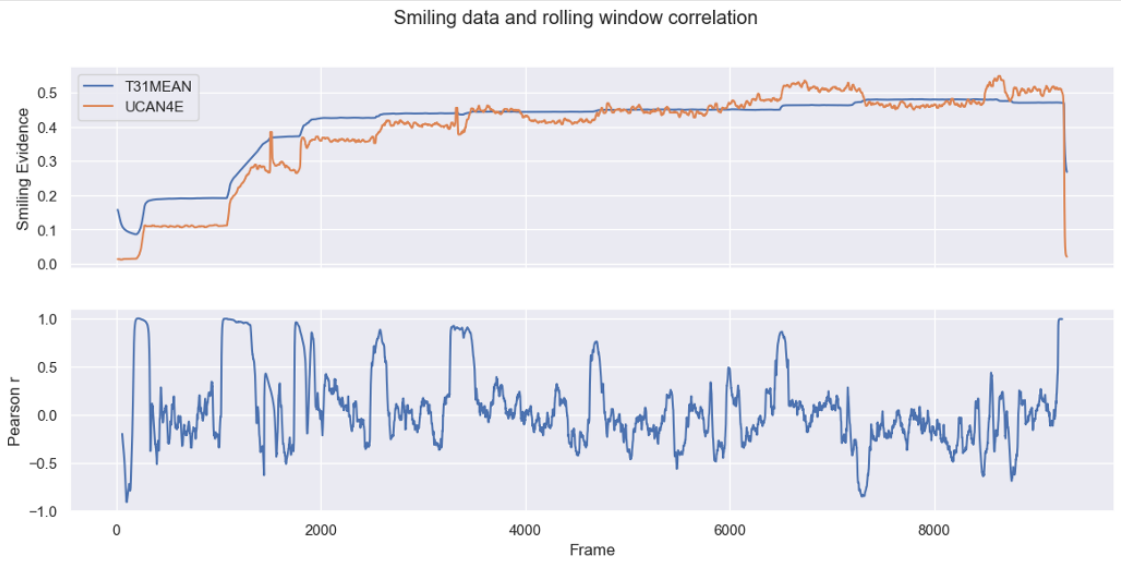


Figure 6.14: Rolling window correlation analysis of E frequency with other features before feature engineering

Figure 6.14 demonstrate the analysis for *T31MEAN* with our target *UCAN4E*. As Figure 6.12 reflects a strong positive linear relationship of *T31MEAN* with *UCAN4E* with a high value of 0.94, which we can also validate with Figure 6.14.

We selected the top eight features *T31MEAN*, *P31MEAN*, *UTSZ*, *UW31Z*, *AITMEAN* with a strong positive correlation, and *UP1*, *UPFRDEM*, *UTPZ* with strong negative correlation over the average lag of 60 seconds.

E. Data Transformation

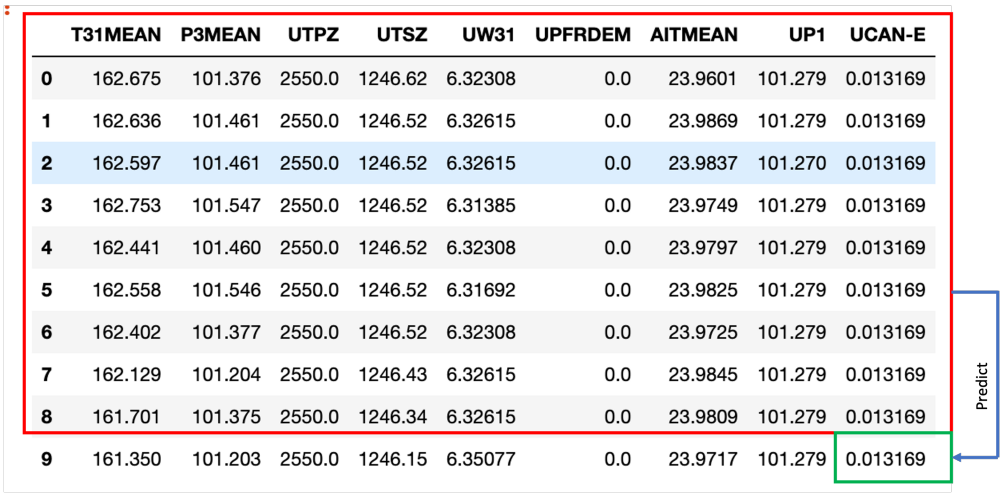
1. **Conversion to sequence data**: After the data cleaning and processing steps, the next stage involves transforming the data to prepare it for training a machine learning model, specifically an LSTM model for sequence prediction. For the `Data_E` dataset, which had an initial shape of (9298,2) representing 9298 seconds of sensor data and nine features, including the noise amplitude E, we needed to reshape the data into a suitable format for the LSTM model.

   To achieve this, we applied the concept of lookback, which involves using a specific number of past time steps to predict the subsequent time step. In our case, we chose a lookback window of 60 seconds. This means that for each sample, we considered the first 60 seconds of data as input features and the 61st-second value as the corresponding target label. Figure 6.15 reflects a sample architecture for the data transformation where the previous 9 values are used to predict 10th value.

   This transformation reshaped the data into a 3D format with dimensions (9250,60,1): 9250 samples, 60-time steps, and one feature. Each sample in this reshaped dataset contained 60 input features representing the past 60 seconds of data, and a single predicted label corresponding to the next second.

   This transformation allows the LSTM model to learn the temporal dependencies and patterns present in the sensor data. It captures the sequential nature of the data and enables the model to effectively learn the underlying patterns and dynamics necessary for accurate prediction.

2. **Data Split**: In the data splitting phase, we utilized the chronological percentage split method discussed in Section 2.2.3 to divide the overall data into training and test sets.



	T31MEAN	P3MEAN	UTPZ	UTSZ	UW31	UPFRDEM	AITMEAN	UP1	UCAN-E
0	162.675	101.376	2550.0	1246.62	6.32308	0.0	23.9601	101.279	0.013169
1	162.636	101.461	2550.0	1246.52	6.32615	0.0	23.9869	101.279	0.013169
2	162.597	101.461	2550.0	1246.52	6.32615	0.0	23.9837	101.270	0.013169
3	162.753	101.547	2550.0	1246.52	6.31385	0.0	23.9749	101.279	0.013169
4	162.441	101.460	2550.0	1246.52	6.32308	0.0	23.9797	101.279	0.013169
5	162.558	101.546	2550.0	1246.52	6.31692	0.0	23.9825	101.279	0.013169
6	162.402	101.377	2550.0	1246.52	6.32308	0.0	23.9725	101.279	0.013169
7	162.129	101.204	2550.0	1246.43	6.32615	0.0	23.9845	101.279	0.013169
8	161.701	101.375	2550.0	1246.34	6.32615	0.0	23.9809	101.279	0.013169
9	161.350	101.203	2550.0	1246.15	6.35077	0.0	23.9717	101.279	0.013169

Figure 6.15: Sample visual representation of data transformation for training

We adopted a 70-30% split. This means that 70% of the data (6475 samples) was allocated to the training set, while the remaining 30% (2775 samples) was assigned to the test set. Using a chronological split, we ensured that the training set contained earlier time points, allowing the model to learn from past observations. The test set, on the other hand, encompassed more recent time points, which served to evaluate the model's performance on unseen data.

6.3.2 Model Development and Training

We use Vector Autoregression (VAR), Support Vector Machine (SVM) and variants of our proposed D-LSTM model to train on the data and generate results on test data. Please note we only present the architecture for our top performing model while we use the results of other models to validate the efficiency and effectiveness of our proposed TSF model.

Architecture: Deep LSTM (DLSTM) recurrent neural network

It has been widely demonstrated that increasing the depth of a neural network is an effective approach for enhancing overall performance [37]. Inspired by the remarkable learning capabilities of deep recurrent network architectures [71], we propose D-LSTM recurrent network tailored for predicting TA amplitude for combustion system. In the proposed D-LSTM

model, we stack multiple LSTM blocks, as depicted in Figure 6.16, consecutively connected in a deep recurrent network fashion to leverage the advantages of a single LSTM layer. The main objective of employing this hierarchical architecture is to handle large or complex dataset (as in our case multivariate in nature with high frequency) with improved generalization capabilities.

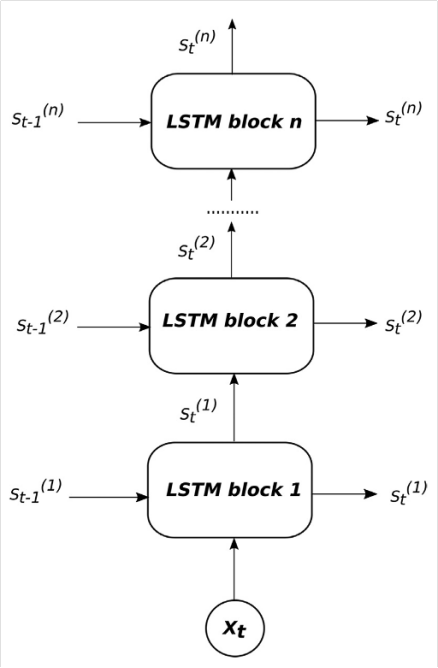


Figure 6.16: The architecture of DLSTM recurrent network

In the architecture, illustrated in Figure 6.16, the input at time t, denoted as X_t, is fed into the first LSTM block, alongside the previous hidden state $S^{(1)}_{(t-1)}$ (where the superscript *(1)* refers to the first LSTM layer). The hidden state at time t, represented as S^1_t, is computed following the process explained in Section 2.2.4. This computed state is then propagated to the subsequent time step and also forwarded to the second LSTM block. In the second LSTM block, the hidden state S^1_t is utilized along with the previous hidden state $S^{(2)}_{(t-1)}$ to compute S^2_t. This newly computed state is then carried forward to the next time step and simultaneously forwarded to the third LSTM block. This process continues iteratively until the last LSTM block is included in the sequence.

Training

As discussed in Section 2.2.3 we used 70% (6475 samples) of the total data for training purpose. For the training, we employ four training hyperparameters: the number of epochs, the number of hidden neurons, lag size, and batch size (batch size greatly affects the model generalization and optimization [67, 68]).

During training, the model iteratively adjusted its parameters to minimize the discrepancy between the predicted outputs and the actual values. The used mean squared error (MSE) as the loss function, which measures the average squared difference between the predicted and actual values. We track the training and validation loss after each epoch to monitor the model's performance during training. This allowed for the detection of overfitting (excessively complex and captures noise in the data) or under-fitting (model is too simple to capture the underlying patterns in the data) phenomena, ensuring that the model was effectively learning from the data without memorizing it. We also implement the early stopping strategy as shown in Figure 6.17 to prevent overfitting, the training process terminates if the validation loss did not improve for a specified number of consecutive epochs.

Learning Rate Scheduler During the training process, we implement a learning rate callback scheduler Section 2.2.4 to adjust the learning rate of the optimizer dynamically.

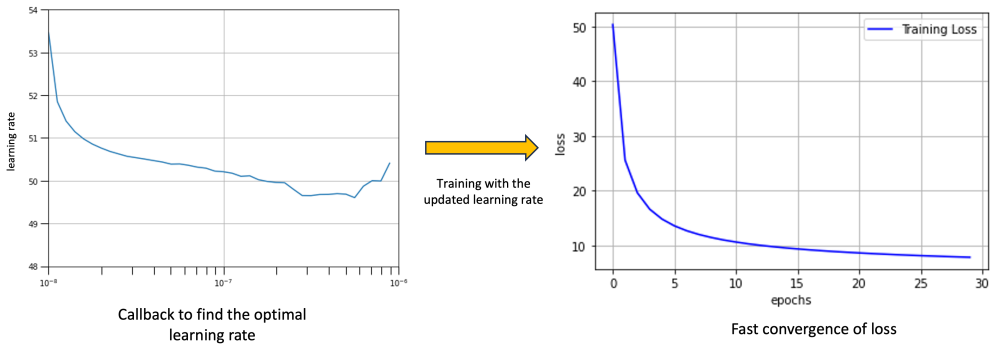


Figure 6.17: Loss convergence with number of epocs

Figure 6.17 reflects the results of optimal learning rate and then using it for training to make the convergence faster. Figure 6.17 also demonstrate the number of epochs it took to

converge before (100) vs after (35) a learning scheduler. It saved us a significant amount of training time.

6.4 Experiment Validation

RQ3

How to develop effective time series forecasting techniques for predicting thermoacoustic instabilities in gas turbines?

For an in-depth investigation of RQ3, we address two specific research questions on selecting the best machine learning model (RQ3.1) and then evaluating its actual effectiveness (RQ3.2).

6.4.1 Model Performance Assessment

RQ3.1

Which are the most effective machine learning models for predicting TA instability prediction?

For this case study we have used the similar metrics MAE and RMSE as in case study 1 to asses the performance of our trained model.

Metrics Calculation

Table 6.3 demonstrate the results of different variants of Deep LSTM, along with Vector Auto regression (VAR). The results indicates the variant of D-LSTM model with 5 layers, 30 hidden units and trained in 32 batches for 100 epochs perfromed best with the RMSE and MAE value of 0.046 and 0.042 respectively.

Findings

Figure 6.20 shows the execution of our proposed model on complete data set (train and test), while Figure 6.19 shows the zoomed version of the test predictions to better visualize the results. we found that the model exhibits a tendency to overpredict in regions characterized

Model	Layers	Hidden units	Epochs	Batch Size	RMSE	MAE
VAR	NA	NA	NA	NA	0.13	0.11
D-LSTM	2	20	30	32	0.091	0.088
D-LSTM	3	30	50	32	0.084	0.078
D-LSTM	**5**	**30**	**100**	**32**	**0.046**	**0.042**

Table 6.3: Performance comparison of top-performing model with different parameters

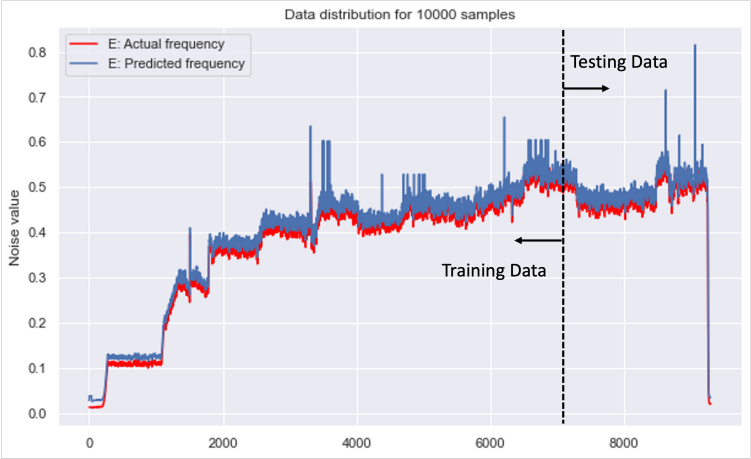


Figure 6.18: Prediction on the training and validation set

by high acoustic amplitude (higher value than peak value of stable region) as highlighted in red, which are high potential regions of reflecting instabilities. This behavior is beneficial for the engineers since overprediction is considered safer for intervention purposes compared to underprediction.

Overall, the findings indicate that hierarchical deep neural networks such as proposed D-LSTM performs better than classical ML models such as VAR in predicting acoustic

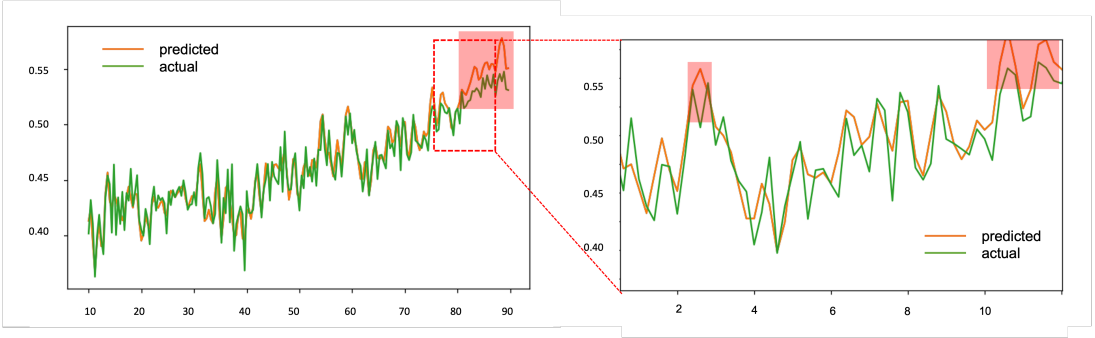


Figure 6.19: Zooming predictions on validation set for 100 and 15 sec

Model	RMSE	MAE
SVM	0.19	0.17
D-LSTM	**0.046**	**0.042**

Table 6.4: Performance comparison for SVM and D-LSTM for instability prediction

amplitude while dealing with multivariate high frequency data. Also, increasing the number of layers and hidden units in the Deep LSTM model, as well as training for a higher number of epochs, contributed to improved predictive performance. Additionally, increasing the batch size had opposite affects.

6.4.2 Assessment of Effectiveness:

RQ3.2

How effective are time series forecasting techniques for predicting thermoacoustic instabilities in gas turbines?

Currently at Siemens team explores Computational Fluid Dynamics (CFD) analysis [72] as the potential method for TA instability prediction, However, this approach requires time and a deep understanding of combustion mechanics, typically carried out by domain experts. Hence, to validate the effectiveness of the TSF methodology, we compared the results of our proposed model with an industry-standard classical machine learning baseline model, Support Vector Machine (SVM).

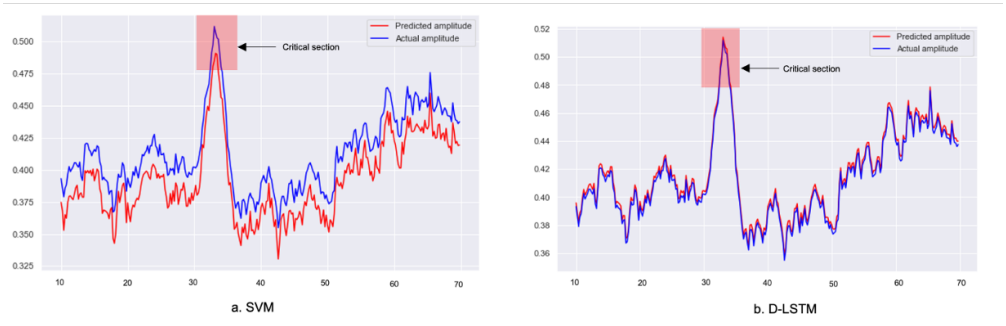


Figure 6.20: Performance comparison of baseline (SVM) and Proposed model (D-LSTM)

Results and findings

Table 6.4 shows the performance of SVM and D-LSTM model on the test data. Figure 6.20 demonstrated the generated amplitude predictions of 70 seconds of high amplitude region (highlighted in red). The overall results indicated that the proposed D-LSTM model is performing better with MAE and RMSE of 0.042 and 0.046. One notable observation is that SVM tend to under predict on high amplitude region while D-LSTM tend to over predict which is also another validation of effectiveness of proposed D-LSTM model.

6.5 Architecture Completeness Validation

RQ1.3

How effective the framework is in terms of completeness?

To address the part of RQ1, we use the similar validation approach demonstrated in Section 4.6.

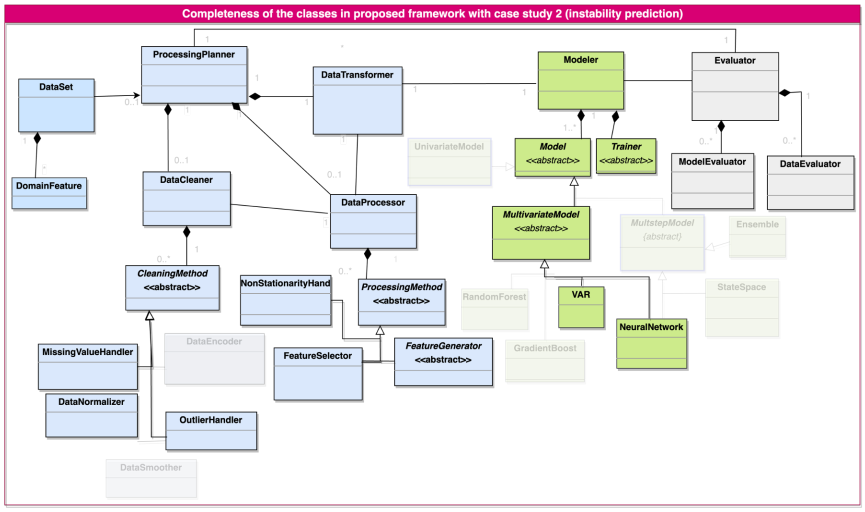


Figure 6.21: Completeness validation of framework (CS2)

Figure 6.21 depicts the framework's component covered to implement TSF pipeline to predict the TA instability prediction. To avoid repetition we would not discuss the mapping of components with TSF processes as the core behaviour of the framework components are similar except for adaptable components that we have discussed in adaptability validation

in Section 4.6. Overall, the experiment used 23 classes out of 32 classes in the framework hence based on the Equation 5, the coverage value calculates to:

$$Coverage = \left(\frac{23}{32}\right) \times 100 = 71.8\%$$

Overall, with ≈72% class coverage, this case study also covers the necessary steps and processes for implementing the end to end TSF pipeline for TA instability prediction. Hence it also validates the minimum coverage value required (30%) as discussed in Section 4.6 to validate the completeness of the framework.

6.6 Summary

In this chapter, we conducted an investigation into various architectural variants of the proposed deep LSTM recurrent neural network and VAR for predicting TA instability. To address RQ3.1, we employed MAE and RMSE metrics to asses the performance of the proposed D-LSTM model. Notably, the D-LSTM model with 5 layers and 30 hidden units exhibited superior performance, yielding the lowest RMSE value of 0.046. Furthermore, our model demonstrated efficiency in predicting high amplitude regions, as evident by better and close predictions within a sample test of 90 seconds sensor data. Additionally, the model's tendency to overpredict the high amplitude region identified as beneficial for domain experts, as underestimating these critical zones may lead to the neglect of valuable information crucial for TA instability prediction.

To address Research Question 3.2, we compared our results with the industry standard baseline model, SVM. The comparison demonstrated the effectiveness of our proposed TSF model, revealing improved outcomes.

Finally, we concluded with the validation of the completeness of our framework (RQ1.3) achieving a score of 72%, exceeding the threshold of 30%. The successful validation further supports the suitability of our proposed framework for dealing with complex TSF scenarios.

Chapter 7

Conclusion

7.1 Contributions and Findings

The overarching goal of this thesis is spread across three areas as follows:

Goal is to present a model driven architecture to better assist practitioners working on multiple time series forecasting projects to reduce the repetitive efforts. To do so, we present a conceptual UML class diagram architecture that aims to promote reusability, adaptability and streamline the TSF pipeline. We also validate its completeness, reusability and adaptability using two independent research projects as discussed next.

Goal is to investigate and implement effective TSF techniques to predict early undergraduate student performance and check its effectiveness. To do so we developed and implemented variants of GRU and LSTM models. And also validated the effectiveness by comparing the results our top performing model (LSTM) on randomly selected sample students form the test set with pedagogical based approach.

Goal is to investigate and apply effective TSF techniques to predict TA amplitude in combustion in order to forecast TA instability in combustion system and validate its effectiveness. In order to do so we presented a deep LSTM model and evaluated its performance again a industry standard baseline model.

Below, we reiterate the main contributions and findings of this thesis by answering the high-level research questions which we present in Chapter 1.

RQ 1

Can we design model-driven architecture in supporting the reusability and adaptability of components in time series forecasting?

RQ1 aimed to design a model-driven architecture to support reuse and adaptation of core components in the multiple or multi-domain implementation of TSF pipeline. The UML class diagram elements played a crucial role in representing and comparing the components of the case studies from different domains. The analysis revealed a considerable percentage (25%) of reusable components, such as **ProcessingPlanner**, **Modeler**, **Evaluator**, and **Trainer** while the rest other classes can be adapted. This indicates the potential for cross-domain knowledge transfer and the benefits of leveraging common components for different projects. We defined reusability, adaptability and completeness metrics along with the threshold to assess the effectiveness and validate the claims. The calculative values of these metrics on the two case studies revealed the higher values than threshold hence confirmed and validated our claims.

RQ 2

How to develop effective TSF techniques for predicting student performance?

In RQ2, we evaluated various stacked LSTM and GRU models, and interestingly, the S-LSTM models consistently outperformed the GRU models, regardless of dropout presence. Among the models tested, the S-LSTM model without dropout exhibited the highest

performance, achieving an impressive MAE of 0.28 (corresponding to the highest accuracy of 86.5%). Visualization of the prediction results allowed us to identify patterns of under-prediction and over-prediction for specific score ranges, providing valuable insights for detecting at-risk students and enabling targeted interventions.

Further, to validate the effectiveness of our proposed model, we compared its results with those obtained using the pedagogical approach. The comparison revealed notable differences, with our proposed model showcasing a 4% improved accuracy. These findings demonstrate the efficacy of the proposed model and its potential to enhance educational outcomes.

RQ 3

How to develop effective time series forecasting techniques for predicting thermoacoustic instabilities in AGT?

For RQ3, we evaluated VAR and variants of D-LSTM model. The D-LSTM model with 5 LSTM block layers, 30 hidden units and 150 training epochs achieved the highest predictive performance, showcasing the long-term temporal dependency of acoustic amplitudes *UCAN4E* and *UCAN8E* with surrounding thermal (*T31MEAN*) and stream pressure (*UPFRDEM*, *UP1*, and *UTPZ*) parameters. Visualization on test data revealed its ability to capture high amplitude areas, high potential regions of TA instability. We also validated the effectiveness by comparing the results with the baseline model SVM. The results indicate the efficacy of TSF effectiveness in predicting high acoustic amplitudes in AGT and suggest potential research directions to optimize and extend the model for other components of gas turbine engines, aiming to enhance overall performance and reduce maintenance costs.

Conclusion Statement This thesis has made significant contributions to the field of Time Series Forecasting (TSF) by introducing a conceptual and comprehensive framework that enhances the productivity of TSF practitioners by offering reusable and adaptable components. The thesis showcases the successful implementation of an effective TSF pipeline, incorporating the proposed top-performing deep neural network models, to predict both student performance and thermoacoustic instability. Through this implementation, we validate the

framework's metrics of reusability, adaptability, and completeness. Broadly speaking, the thesis demonstrates the capabilities of software modeling for Machine Learning (ML) and highlights the potential of ML techniques in predicting student performance and thermoacoustic instabilities.

In summary, this research not only contributes to the advancement of TSF methodologies but also showcases the broader potential of software modeling for streamlining ML processes and ML's practical applications in predicting critical outcomes like student performance and thermoacoustic instabilities in gas turbines. The framework's ability to improve productivity and promote the reusability of components can have far reaching implications for the TSF community and beyond.

7.2 Limitations

In the following subsections we discuss the limitations of the proposed framework and the case studies.

7.2.1 Proposed Framework

- The framework is limited to TSF pipelines though it could be extended to a generic ML or DL pipelines.
- The framework only support the reusability and adaptability at run-time but not at the configuration level.

7.2.2 Student's Performance Prediction

- This was a pilot research study hence we had limited access to the student's data. Further addition of sample or features might present opportunity to identify more correlated features and improve accuracy.
- Although the SciLearn program is incentivized with a small bonus mark, participants are self-selected and therefore not fully representative of McGill's first-year cohort.

- The courses in which data were collected are also taught by instructors known to be excellent teachers, as our sample grows other features related to course design (e.g., type of assessments) could affect the ML modeling process.

7.2.3 TA Instability Prediction

- The data used for the research is rig simulation data hence results reflects the close to reality but could be improved with more real time data in future.
- Proposed model might not account well for external factors that are not explicitly included in the training data but could affect thermoacoustic instability, such as environmental conditions or fuel variations.
- While proposed LSTM model can learn patterns from data, it might not provide deep physical insights into the underlying mechanisms of thermoacoustic instability. Hence the results must be combined with domain knowledge to better understand the mechanism.

7.3 Opportunities for Future Research

Promising avenues for future work, building on the contributions of this thesis, include:

Extension of the proposed framework: The framework can be expanded to encompass the concept of adaptation in existing AutoML tools like TPOT and MLbox. Additionally, the next steps involve code generation for the proposed model-driven architecture to develop efficient software applications.

Enhancing TSF model for student performance prediction: While the current 86.5% accuracy may not suffice as a standalone tool for students and instructors, it has shown great potential and effectiveness of TSF integration in prediction of student performance. Further exploration and inclusion of behavioral data points, course specific parameters and advanced ML algorithms like transformers would be the next steps in this research direction.

Enhancing TSF model for thermoacoustic instability prediction: The results from TA predictions demonstrated the superiority of hierarchical deep learning LSTM models over classical ML and conventional CFD methods. Within Siemens, the next steps might include exploration of these models in unstable and stable regions and extending the pipeline for other performance optimization tasks of AGT, such as age prediction and optimizing the critical slowing down process.

Bibliography

[1] R. Saini, "Automated, interactive, and traceable domain modelling," 2023.

[2] S. Meisenbacher, M. Turowski, K. Phipps, M. Rätz, D. Müller, V. Hagenmeyer, and R. Mikut, "Review of automated time series forecasting pipelines," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 12, no. 6, p. e1475, 2022.

[3] D. S. Endicott, "Experimental development of a lean direct injection combustor utilizing high-low swirl intensity combinations," 2014.

[4] O. Mohamed and A. Khalil, "Progress in modeling and control of gas turbine power generation systems: a survey," *Energies*, vol. 13, no. 9, 2020.

[5] A. Hellberg, "The siemens sgt-750 gas turbine : Developed for the oil and gas industry," 2013.

[6] K. Bengtsson, "Thermoacoustic instabilities in a gas turbine combustor," Master's thesis, KTH, Marcus Wallenberg Laboratory MWL, 2017.

[7] C. Chatfield, *Time-series forecasting*. CRC Press, 2000.

[8] G. Zhang, B. E. Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks:: The state of the art," *International Journal of Forecasting*, vol. 14, no. 1, pp. 35–62, 1998.

[9] D. S. Wilks, *Statistical methods in the atmospheric sciences*, vol. 100. Academic Press, 2011.

[10] J. Bughin, J. Seong, J. Manyika, M. Chui, and R. Joshi, "Notes from the AI frontier: modeling the impact of AI on the world economy," *McKinsey Global Institute*, vol. 4, 2018.

[11] J. Barnes, "Azure machine learning," *Microsoft Azure Essentials. 1st ed, Microsoft*, 2015.

[12] N. Katiyar and A. Yazdani, "Transforming learning data into a machine learning model to help stem students transition university," in *Proceedings of the 23rd International Conference on Computer Supported Collaborative Learning - CSCL'23 (to be published)*, Association for Computational Linguistics, 2023.

[13] C. B. Meher-Homji, J. Zachary, A. F. Bromley, *et al.*, "Gas turbine fuels-system design, combustion, and operability," in *Proceedings of the 39th Turbomachinery Symposium*, Texas A&M University. Turbomachinery Laboratories, 2010.

[14] H. Gomaa, *Software modeling and design: UML, use cases, patterns, and software architectures*. Cambridge University Press, 2011.

[15] B. Combemale, J. Kienzle, G. Mussbacher, H. Ali, D. Amyot, M. Bagherzadeh, E. Batot, N. Bencomo, B. Benni, J.-M. Bruel, *et al.*, "A hitchhiker's guide to model-driven engineering for data-centric systems," *IEEE Software*, vol. 38, no. 4, pp. 71–84, 2020.

[16] G. Reggio, M. Leotta, F. Ricca, and D. Clerissi, "What are the used UML diagrams? a preliminary survey.," in *EESSMod@ MoDELS*, pp. 3–12, 2013.

[17] Á. Malta, M. Soares, E. Santos, J. Paes, F. M. Alencar, and J. Castro, "istartool: modeling requirements using the i* framework.," in *IStar*, pp. 163–165, 2011.

[18] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: eclipse modeling framework*. Pearson Education, 2008.

[19] E. Duala-Ekoko and M. P. Robillard, "Tracking code clones in evolving software," in *29th International Conference on Software Engineering (ICSE'07)*, pp. 158–167, IEEE, 2007.

[20] O. A. Specification, "OMG Unified Modeling Language (OMG UML), superstructure, v2. 1.2," *Object Management Group*, vol. 70, 2007.

[21] R. S. Tsay, D. Pena, and A. E. Pankratz, "Outliers in multivariate time series," *Biometrika*, vol. 87, no. 4, pp. 789–804, 2000.

[22] R. Mushtaq, "Augmented Dickey-Fuller test," *SSRN Electronic Journal*, 2011.

[23] P. Zhang, Y. Huang, S. Shekhar, and V. Kumar, "Correlation analysis of spatial time series datasets: a filter-and-refine approach," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 532–544, Springer, 2003.

[24] Z. Reitermanova *et al.*, "Data splitting," in *WDS*, vol. 10, pp. 31–36, Matfyzpress Prague, 2010.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[26] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[27] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *ArXiv Preprint arXiv:1406.1078*, 2014.

[28] M. J. Jacobson and U. Wilensky, "Complex systems in education: scientific and educational importance and implications for the learning sciences," *The Journal of the Learning Sciences*, vol. 15, no. 1, pp. 11–34, 2006.

[29] F. Grieve, "Scilearn: helping first-year science students learn better," Jan 2023.

[30] G. Schraw and R. S. Dennison, "Assessing metacognitive awareness," *Contemporary Educational Psychology*, vol. 19, no. 4, pp. 460–475, 1994.

[31] C. S. Dweck, *Mindset: the new psychology of success*. Random House, 2006.

[32] E. Pasquinelli, "Neuromyths: why do they exist and persist?," *Mind, Brain, and Education*, vol. 6, no. 2, pp. 89–96, 2012.

[33] M. T. Schobeiri, *Gas turbine design, components and system design integration: second revised and enhanced edition*. Springer Nature, 2019.

[34] M. P. Boyce, *Gas turbine engineering handbook*. Elsevier, 2011.

[35] E. Giulietti, "Thermoacustic instabilities in gas turbine burners: new diagnostic methodology," 2011.

[36] T. C. Lieuwen and V. Yang, *Combustion instabilities in gas turbine engines: operational experience, fundamental mechanisms, and modeling*. American Institute of Aeronautics and Astronautics, 2005.

[37] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[38] J. C. Kirchhof, E. Kusmenko, J. Ritz, B. Rumpe, A. Moin, A. Badii, S. Günnemann, and M. Challenger, "MDE for machine learning-enabled software systems: a case study and comparison of MontiAnna & ML-Quadrat," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pp. 380–387, 2022.

[39] C. Wieman, "Why not try a scientific approach to science education?," *Change: The Magazine of Higher Learning*, vol. 39, pp. 9–15, 01 2007.

[40] Z. Xu and J. Sun, "Model-driven deep-learning," *National Science Review*, vol. 5, no. 1, pp. 22–24, 2018.

[41] X. He, K. Zhao, and X. Chu, "AutoML: a survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, 2021.

[42] G. Squillero, P. Burelli, *et al.*, *Applications of evolutionary computation*. Springer, 2016.

[43] C. Thornton, *Auto-WEKA: combined selection and hyperparameter optimization of supervised machine learning algorithms*. PhD thesis, University of British Columbia, 2014.

[44] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[45] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: machine learning in python," *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[46] R. A. González Bustamante, R. Ferro Escobar, and G. Tarazona Bermúdez, "Development of a UML framework for smart cities with forecasting electrical consumption in Colombia," *Tecnura*, vol. 18, no. SPE, pp. 109–123, 2014.

[47] W. Xin, L. Chao, X. Weiren, and L. Ying, "A failure time series prediction method based on UML model," in *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, vol. 1, pp. 62–70, IEEE, 2015.

[48] K. Shaleena and S. Paul, "Data mining techniques for predicting student performance," in *2015 IEEE International Conference on Engineering and Technology (ICETECH)*, pp. 1–3, IEEE, 2015.

[49] A. Shahiri, W. Husain, and N. Abdul Rashid, "A review on predicting student's performance using data mining techniques," *Procedia Computer Science*, vol. 72, pp. 414–422, 12 2015.

[50] H. Altabrawee, O. Ali, and S. Qaisar, "Predicting students' performance using machine learning techniques," *Journal of University of Babylonfor Pure and Applied Sciences*, vol. 27, pp. 194–205, 04 2019.

[51] Y. Baashar, G. Alkawsi, N. Ali, H. Alhussian, and H. T. Bahbouh, "Predicting student's performance using machine learning methods: a systematic literature review," in *2021 International Conference on Computer & Information Sciences (ICCOINS)*, pp. 357–362, IEEE, 2021.

[52] F. Chen and Y. Cui, "Utilizing student time series behaviour in learning management systems for early prediction of course performance," *Journal of Learning Analytics*, vol. 7, no. 2, pp. 1–17, 2020.

[53] C. M. Aurah, "The effects of self-efficacy beliefs and metacognition on academic performance," 2013.

[54] A. L. Dill, C. A. Justice, S. S. Minchew, L. M. Moran, C.-h. Wang, and C. B. Weed, "The use of the LASSI to predict and evaluate the study habits and academic performance of students in a learning assistance program," *Journal of College Reading and Learning*, vol. 45, no. 1, pp. 20–34, 2014.

[55] Q. An, A. M. Steinberg, S. Jella, G. Bourque, and M. Füri, "Early warning signs of imminent thermoacoustic oscillations through critical slowing down," *Journal of Engineering for Gas Turbines and Power*, vol. 141, no. 5, p. 054501, 2019.

[56] Z. Lyu, Y. Fang, Z. Zhu, X. Jia, X. Gao, and G. Wang, "Prediction of acoustic pressure of the annular combustor using stacked long short-term memory network," *Physics of Fluids*, vol. 34, no. 5, 2022.

[57] S. Le Clainche, E. Ferrer, S. Gibson, E. Cross, A. Parente, and R. Vinuesa, "Improving aircraft performance using machine learning: a review," *Aerospace Science and Technology*, p. 108354, 2023.

[58] P. Ramu, P. Thananjayan, E. Acar, G. Bayrak, J. W. Park, and I. Lee, "A survey of machine learning techniques in structural and multidisciplinary optimization," *Structural and Multidisciplinary Optimization*, vol. 65, no. 9, p. 266, 2022.

[59] C. Bhattacharya, J. O'Connor, and A. Ray, "Data-driven detection and early prediction of thermoacoustic instability in a multi-nozzle combustor," *Combustion Science and Technology*, vol. 194, no. 7, pp. 1481–1512, 2022.

[60] E. O. Brigham, *The Fast Fourier transform and its applications*. Prentice-Hall, Inc., 1988.

[61] S. C. Chin, A. Ray, and V. Rajagopalan, "Symbolic time series analysis for anomaly detection: a comparative evaluation," *Signal Processing*, vol. 85, no. 9, pp. 1859–1868, 2005.

[62] S. R. Eddy, "What is a hidden Markov model?," *Nature Biotechnology*, vol. 22, no. 10, pp. 1315–1316, 2004.

[63] B. Rumpe, *Modeling with UML*. Springer, 2016.

[64] M.-H. Cho and J. Yoo, "Exploring online students' self-regulated learning with self-reported surveys and log files: a data mining approach," *Interactive Learning Environments*, vol. 25, pp. 1–13, 10 2016.

[65] M. Francisco and C. Amado, *Perusall's machine learning towards self-regulated learning*, pp. 49–58. 11 2021.

[66] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. OTexts, 2018.

[67] M. Derezinski, D. Mahajan, S. S. Keerthi, S. V. N. Vishwanathan, and M. Weimer, "Batch-expansion training: an efficient optimization framework," in *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*

(A. Storkey and F. Perez-Cruz, eds.), vol. 84 of *Proceedings of Machine Learning Research*, pp. 736–744, PMLR, 09–11 Apr 2018.

[68] M. Takac, A. Bijral, P. Richtarik, and N. Srebro, "Mini-batch primal and dual methods for svms," in *Proceedings of the 30th International Conference on Machine Learning* (S. Dasgupta and D. McAllester, eds.), vol. 28 of *Proceedings of Machine Learning Research*, (Atlanta, Georgia, USA), pp. 1022–1030, PMLR, 17–19 Jun 2013.

[69] A. F. Agarap, "Deep learning using rectified linear units (ReLu)," *ArXiv Preprint arXiv:1803.08375*, 2018.

[70] V. H. Vu, "On the infeasibility of training neural networks with small mean-squared error," *IEEE Transactions on Information Theory*, vol. 44, no. 7, pp. 2892–2900, 1998.

[71] M. Hermans and B. Schrauwen, "Training and analyzing deep recurrent neural networks," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, (Red Hook, NY, USA), p. 190–198, Curran Associates Inc., 2013.

[72] D. Zhao, "Chapter 8 - CFD studies on thermoacoustic instabilities," in *Thermoacoustic Combustion Instability Control* (D. Zhao, ed.), pp. 585–672, Academic Press, 2023.