Handling Temporal Characteristics of Data in Sequential Modelling

Faculty of Computer Science

Master's Thesis

to obtain the degree

Master of Science

submitted by

Seema Ganpat More

Matriculation number: 310906

First Supervisor:	Prof. Dr. Martin Golz
Second Supervisor:	M.Sc. Martin Patrick Pauli
Third Supervisor:	Dr. Rolf Bardeli (Vodafone GmbH)

3rd January 2024

Statutory Declaration

I hereby declare that I have written this Master's thesis on my own and that I have not used any aids that are not explicitly stated in the thesis. The thoughts taken directly or indirectly from external sources are marked as such. I further declare that no part of this thesis has been submitted for any other degree or qualification at any other university.

Schmalkalden, 3rd January 2024

Seema Ganpat More

Acknowledgement

My journey to completing this Master's thesis has been supported by many, and I would like to extend my gratitude to each one of them. First and foremost, my parents, **Mr. Ganpat More** and **Mrs. Alka More**, have been my pillars of strength throughout my master's journey. I am also deeply thankful to my brother, **Mr. Sameer More**, who has always had faith in me, provided guidance, and stood by my side at all times.

Special appreciation goes to my first supervisor, **Professor Dr. Martin Golz**, for his valuable suggestions, which led me to explore new ideas and approaches. I would also like to express my special thanks to **Dr. Rolf Bardeli**, for his guidance and support through several meetings and for ensuring this work was crowned with success. His insightful suggestions were invaluable to me throughout the Master's thesis.

Lastly, I am grateful to my second supervisor, **Martin Patrick Pauli**, for examining my practical work and providing his insights.

Abstract

Sequential models are crucial in machine learning, particularly when time dependency is essential for identifying patterns and making predictions.

This master's thesis investigates the consistency of the performance of sequential models on two distinct time dependent datasets of different characteristics. The study specifically examines a dataset of mobile phone sales from various Vodafone retail stores and the US drought levels dataset for precipitation prediction. By focusing on the processing of both static and dynamic features, the study seeks to determine if similar results emerge across both datasets, or if the outcomes are specific to one. The study delves into the sequential relationships among various data features, focusing on the significance of time-series data.

At the core of this study are two primary models: a sequential model of keras (with dense layers) that processes all data features, and an lstm model that employs a windowing technique, allowing it to sequentially view previous data points.

This master's thesis introduces three hybrid models combining static and dynamic data. The first model processes static features and uses an autoencoder to reconstruct dynamic features for the keras sequential framework. The second model uses a keras sequential model for static features and an lstm model for dynamic features. The outputs from both are merged and input into another sequential model with dense layers of keras, yielding predictions for mobile phone sales and prectot values. The third model processes static features and uses an autoencoder to compress dynamic features for the keras sequential framework, specifically for the precipitation dataset.

The insights derived from this study aim to refine our understanding and employment of feature processing strategies. Moreover, the research suggests that differentiating between static and dynamic data processing does not always enhance the model's ability to discern data patterns.

Table of Contents

Statutory Declaration **I**

Acknowledgment **II**

Abstract **III**

Table of Contents **IV**

1 Introduction **1**
1.1 Research Questions . . . 1

2 Theoretical Background **2**
2.1 Sequential API in Keras . . . 2
2.1.1 Dense Layer . . . 3
2.2 Recurrent Neural Network . . . 3
2.2.1 Long Short Term Memory (LSTM) . . . 4
2.3 Activation Function . . . 6
2.4 Optimizer - ADAM . . . 7
2.5 Absolute Error And Mean-Absolute-Error . . . 9

3 Methodology **11**
3.1 Dataset Selection . . . 11
3.1.1 Vodafone Dataset . . . 11
3.1.2 Precipitation Dataset . . . 13
3.2 Dataset Preprocessing . . . 15
3.3 Dataset Splitting . . . 15
3.4 Implementations . . . 16
3.4.1 First Base Model Architecture . . . 16
3.4.2 Second Base Model Architecture . . . 17
3.4.3 First Strategy - Autoencoder Model . . . 18
3.4.4 Second Strategy - Dynamic Embeddings Model . . . 20
3.4.5 Third Strategy - Hybrid Model . . . 21
3.5 IDE and Frameworks . . . 24

4 Results **27**
4.1 Performance of First Base Model Architecture . . . 27
4.2 Performance of Second Base Model Architecture . . . 30
4.3 Performance of First Strategy - Autoencoder Model . . . 32
4.4 Performance of Second Strategy - Dynamic Embeddings Model . . . 35
4.5 Performance of Third strategy - Hybrid Model . . . 36

5 Summary and Outlook **41**
5.1 Answering the Research Questions . . . 41
5.2 Future Work . . . 41

Bibliography **V**

1. Introduction

Data plays a crucial role in machine learning, its understanding, preprocessing, and feature extraction are important aspects of this field. The individual characteristics of datasets play an important role in prediction accuracy, which is essential for the growth of various industries. In machine learning, one challenge is finding a good combination of feature extraction and model choice adapted to the data characteristics.

Thus, this thesis focuses on finding an approach to handle these features properly. The objective of this thesis is to explore methods for combining static and dynamic data to improve the predictions of sequential models and to check if these methods are effective with two different datasets. The first dataset is the Vodafone retail stores dataset, where an important task is understanding their sales data to develop strategies for improving product sales. Specifically, this thesis works on the predictions of mobile phone sales in different types of Vodafone retail stores in Germany.

The second dataset is related to weather and provides information on drought levels and other weather-related features in different regions of the United States [1]. Understanding weather patterns and identifying areas with higher risks of drought is important for taking preventive measures.

This research study uses these two datasets of different characteristics with one commonality of having both static and dynamic features. The thesis focuses on the strategies for combining these types of data and check the results of these strategies on the two distinct datasets provided.

1.1 Research Questions

The following questions have been identified as most relevant to this Thesis:

1. What are the best practices for preprocessing and integrating static and dynamic data in machine learning models to maximize sequential model performance?

2. How does the number of static and dynamic features in the dataset impact the performance of hybrid models?

3. Is there a need for using different strategies for integrating static and dynamic data for specific characteristics of datasets?

2. Theoretical Background

This chapter will focus on the foundational concepts on which this thesis is based on. A concise overview of concepts used in this study is presented, namely: The Keras Sequential API, Recurrent Neural Network (RNN), Long Short Term Memory (LSTM), ADAM Optimizer, ReLU Activation Function and the Mean Absolute Error (MAE) Metric.

2.1 Sequential API in Keras

The sequential API in keras handles the organization and sequencing of layers within a model (referenced as fig 2.1). It integrates these layers into neural networks to ensure smooth functionality with the keras library. This type of model type is adept at handling simple, layer-based problems. The layer-based problems are those problems that can be solved by straightforward linear arrangement of layers [2].

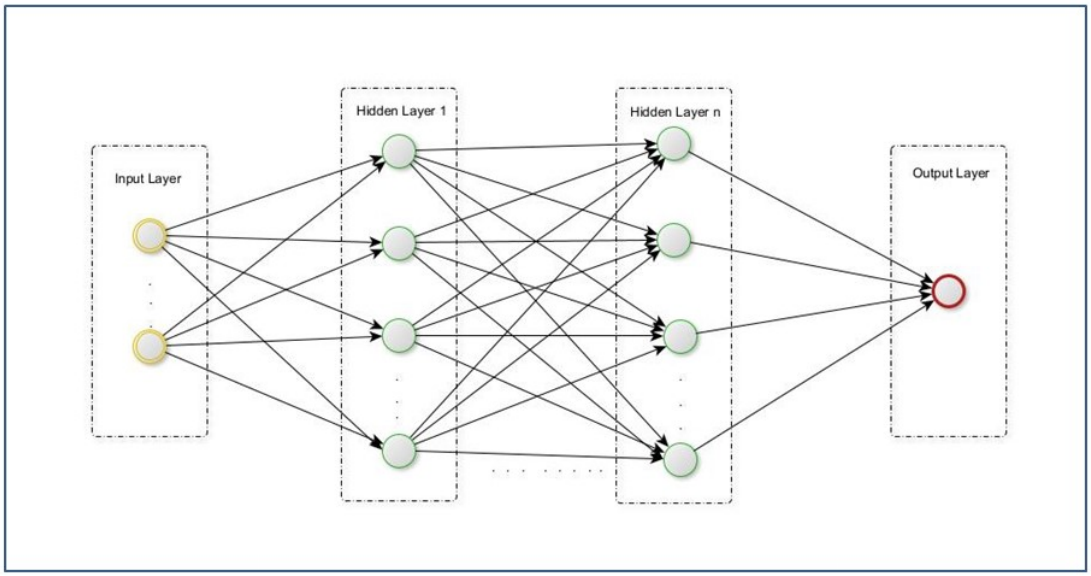


Figure 2.1: Sequential Neural Network Architecture for Regression

.

A sequential API is utilized to construct models as a sequence of layers. It uses an instance of a sequential class that contains an representation of neural network structure. It also contains the shape of the output, the quantity of parameters, which serve as weights in each layer, and the total parameter count[2].

The sequential model consist of a linear, single stack of layers, including a single input layer where network receives its input data. Each node in this layer represents one feature of input data. Then multiple hidden layers which do not directly interact with the external environment. Finally, a single output layer with single neuron is used for a regression task because its predicting a single continuous value, as shown in Figure 2.1 [3].

2.1.1 Dense Layer

Models utilize various types of layers, one of which is the dense layer, frequently used in deep neural networks. Each neuron in these dense layers is interconnected with neurons from the previous layer. The neurons in the dense layers take the output from every neuron in their preceding layer, which is then subject to matrix vector multiplication within the layer's neurons (as depicted in the equation below).

Matrix vector multiplication involves multiplying the matrix, which represents the output from previous layers, with a column vector that represents the weights of the dense layer. The multiplication follows a rule that the number of columns in the matrix must be equal to the number of rows in the vector.

The matrix equation is as follows:

$$Wx = \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \dots & w_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} w_{11}x_1 + w_{12}x_2 + \cdots + w_{1n}x_n \\ w_{21}x_1 + w_{22}x_2 + \cdots + w_{2n}x_n \\ \vdots \\ w_{m1}x_1 + w_{m2}x_2 + \cdots + w_{mn}x_n \end{pmatrix}$$

Assuming $\boldsymbol{W}$ is an $m \times n$ matrix and $\boldsymbol{x}$ is an $n \times 1$ matrix, the product $\boldsymbol{Wx}$ yields the activation for the next layer in the neural network. The matrix entries represent the parameters of the model, which can be further updated through back-propagation[4].

2.2 Recurrent Neural Network

Neural networks are useful for solving multivariate and non-linear mappings due to their ability to learn and adapt. Specifically, recurrent neural networks are very effective for detecting patterns specifically in a sequence data, the data could be text, handwriting, genome sequences, time series data, or industry-specific data like stock market prices or sensor readings.

RNNs are a type of neural network that includes connections which feed information from the outputs of neurons back into the network. This allows RNNs to remember previous inputs, using them in combination with the current input to make predictions. In other words, they are not solely dependent on the current input $\boldsymbol{X_t}$, but also consider past inputs $\boldsymbol{X_{0:t-1}}$.

The hidden state in RNNs, denoted as $\boldsymbol{H_t}$, and the input $\boldsymbol{X_t}$ at time step t, can be defined in specific dimensions:

$$\boldsymbol{H_t} \in \mathbb{R}^{n \times h}$$

$$\boldsymbol{X_t} \in \mathbb{R}^{n \times d}$$

where:

$\boldsymbol{n}$ is the number of samples,

$\boldsymbol{d}$ is the number of inputs for each sample,
and $\boldsymbol{h}$ is the number of hidden units.

The process also involves weight matrices $\boldsymbol{W_{xh}}$ (of dimension $d \times h$) for input-to-hidden state, and $\boldsymbol{W_{hh}}$ (of dimension $h \times h$) for hidden-state-to-hidden-state transitions, along with a bias parameter $\boldsymbol{b_h}$ for the hidden layer and $\boldsymbol{b_o}$ for the output. This RNN represents a single recurrent layer, characterized by the hidden state $\boldsymbol{H_t}$ and the weight matrices $\boldsymbol{W_{xh}}$ and $\boldsymbol{W_{hh}}$.

The transformation of information within the RNN is managed through an activation function $\boldsymbol{\phi}_{\text{hidden}}$, which could be a logistic sigmoid, tanh, or relu function. This is instrumental for computing gradients for back-propagation during learning.

The equation for the hidden state in a RNN is as follows:

$$\boldsymbol{H_t} = \phi_{\text{h}}(\boldsymbol{X_t W_{xh}} + \boldsymbol{H_{t-1} W_{hh}} + \boldsymbol{b_h})$$

and the equation for the output is below:

$$\boldsymbol{o_t} = \phi_{\text{o}}(\boldsymbol{H_t W_{ho}} + \boldsymbol{b_o})$$

This sums up the fundamental principles of how RNNs work, it explains how RNNs use a series of inputs over time and emphasizes the use of activation functions in the learning process[5][6].

2.2.1 Long Short Term Memory (LSTM)

In RNNs, gradients are utilized to update the weights of the neural network during training. One of the challenges with RNNs is their ability to retain long-term dependencies, often referred to as the memory duration problem. This issue is further intensified by the vanishing gradient problem, where, during back-propagation, gradients can become extremely small, leading to minimal weight updates. Conversely, if the gradients become too large, RNNs can face the exploding gradient problem. LSTMs were introduced to mitigate these challenges, offering mechanisms to capture long-term dependencies without the gradient vanishing or exploding[7].

LSTM units incorporate an input gate and a forget gate to tackle the issues of vanishing and exploding gradients[7]. They consist of three logistic sigmoid gates and use the tanh activation function. These components determine how information flows, allowing certain information to pass through, be stored, or be discarded based on its relevance[8]. Specifically, lstm consist of the following key components:

1. Forget gate ($\boldsymbol{f}$): In the forget gate, the input is combined with the previous output to generate a fraction between 0 and 1. This fraction determines what information to retain and which information to discard. This output is then multiplied with the previous state. An activation output 1 means "remember everything" and 0 means "forget everything"[9].

2. Input gate ($\boldsymbol{i}$): The input gate determines which information should enter the lstm state. The output of the input gate is multiplied with the output of the tanh block, and the resulting values are added to the previous state. This vector, when added to the previous state, generates the current state[9].

3. Input modulation gate (g): This is considered a subpart of the input gate. The input modulation gate modulates the information that the input gate will write to the internal state cell. It adds non-linearity to this information and makes it zero-mean. This process is done to reduce the learning rate[9].

4. Output gate (o): In the output gate, the input gate and the previous state are gated to produce a scaling fraction. This scaling fraction is then merged with the output of a tanh block, which represents the current state. The resulting output is then emitted[9].

The working of lstm units is detailed as follows (2.2):

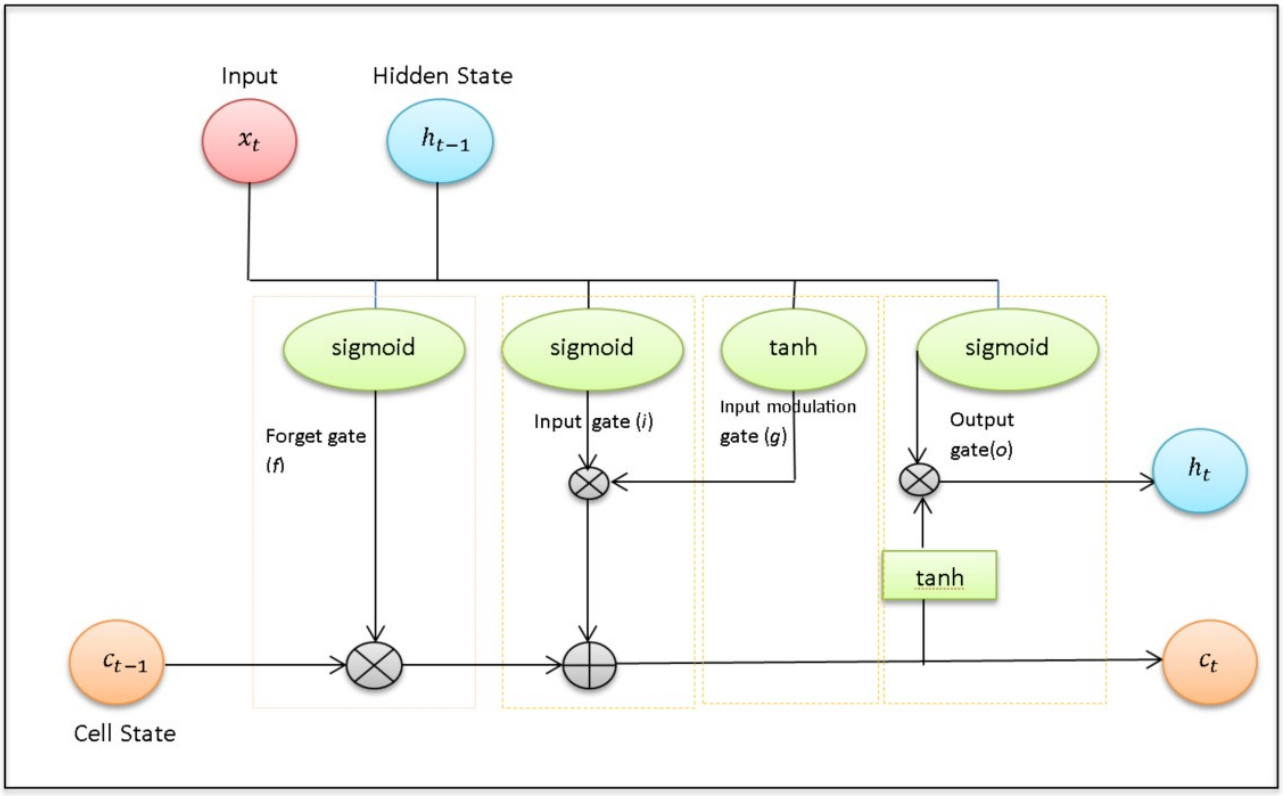


Figure 2.2: LSTM Cell Architecture

1. Take the current input, previous hidden state and previous internal cell state.

2. For each gate in the lstm, the current input and previous hidden state are combined through a weighted sum using respective weights for each gate. Then, this combination is passed through an activation function applied element-wise to compute the gate's activations.

3. The current internal cell is calculated by performing element-wise multiplication of the vectors from the input gate and the input modulation gate. Next, compute the element-wise multiplication of the forget gate vector and the previous internal cell state. Finally, add these two resulting vectors.

$$\boldsymbol{c_t = i \odot g + f \odot c_{t-1}}$$

4. The current hidden state is calculated by taking the element-wise tanh of the current internal

cell state vector and then multiplying this results element-wise with the vector from the output gate[9].

$$\boldsymbol{h_t} = \boldsymbol{o_t} \odot \mathbf{tanh}(\boldsymbol{c_t})$$

In the equations above and in Figure 2.2,

$\boldsymbol{f}$: forget gate output
$\boldsymbol{i}$: input gate output
$\boldsymbol{g}$: input modulation gate output
$\boldsymbol{o}$: output gate output
$\boldsymbol{c}_{t-1}$: prior cell state at time *t-1*
$\boldsymbol{c}_t$: new cell state at time t
$\boldsymbol{h}_{t-1}$: previous hidden state at time *t-1*
$\boldsymbol{w}$: weight (adjusted during learning)
$\boldsymbol{h}_t$: hidden state at time t

2.3 Activation Function

Activation functions are mathematical functions that determine the output of a neuron in an artificial neural network based on the outcome of the input function. They establish slide non-linearity into the model which allows the network to learn from an error and make complex decisions. While inspired by biological neurons, which transmit signals based on input stimuli, to the cell body, they have overcome the axon hilllock and this is modeled by the activation function with a threshold. Below the threshold no signal or weaker signal comes through and beyond threshold value a much stronger input signal comes through.

These functions are broadly categorized into linear and non-linear activation functions. The relu, which is mostly used in deep learning and is a type of non-linear activation function, was used for this master's thesis[10].

Rectified Linear Unit (ReLU)

The linear activation function is a straightforward function that applies simpler transformation to the input data. Even though it is simple to implement, its hypothesis class is not as reach as that of other non-linear activation functions. In contrast, non-linear activation functions, such as the sigmoid and hyperbolic tangent (tanh), can have the property of saturation that contributes to model learning. They are, run-time efficient and unlike relu they do not have property of non-saturation and raise unknown stability issue [11].

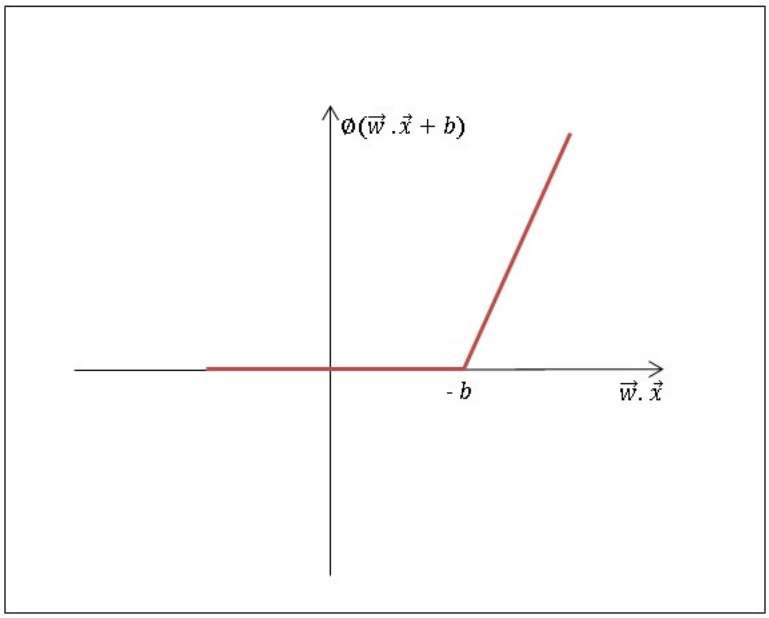


Figure 2.3: Relu Activation Function

To tackle the issue of gradient vanishing, the relu activation function was introduced to dynamic networks by Hahnloser and his colleagues in 2000. By 2011, relu was demonstrated to enable better training of deep neural networks compared to other popular activation functions. Currently, relu is one of the most favoured choices in current deep learning models[12].

The relu function is expressed as follows (refer fig. 2.3):

$$\text{ReLU}(\vec{\boldsymbol{w}} \cdot \vec{\boldsymbol{x}} + \boldsymbol{b}) = \mathbf{max}(\mathbf{0}, \vec{\boldsymbol{w}} \cdot \vec{\boldsymbol{x}} + \boldsymbol{b})$$

Here, $\vec{\boldsymbol{w}}$ is a weight vector, $\vec{\boldsymbol{x}}$ is a input vector and $\boldsymbol{b}$ is a bias. The relu activation function is applied to the sum of the dot product and the bias. if input value is less than 0, it returns 0, but if input value is equal to or greater than 0, it returns the input itself. The function's output is directly proportional to the input. However, for all non-positive values (0 or negative), the function's output is consistently 0. This behavior of relu helps in mitigating the gradient vanishing problem [13].

2.4 Optimizer - ADAM

Adaptive Moment Estimation (Adam) (2.4) is a deep learning optimization algorithm designed to update neural network weights based on training data. It combines features from other optimization algorithms like AdaGrad and RMSProp, allowing it to efficiently handle sparse gradients and noisy problems. Adam computes adaptive learning rates for each parameter [14].

The steps of adam optimization algorithm are as follows [15]:

1. Gradient (g): The gradient of objective function E with respect to weights w at iteration t. The loss function is what the optimization algorithm tries to minimize. This gradient indicates

the direction in which the weights need to be adjusted to minimize the loss.

$$\boldsymbol{g = \nabla_w E(w_t)}$$

2. First Moment Vector (m_t): The m_t represents a decaying average of past gradients. At each iteration t, the optimizer updates this vector. It is a combination of the previous vector m_{t-1} and the current gradient g. The hyperparameter β_1 controls the exponential decay rate for m_t dictating how much influence past gradients have on the current update. The term $(1 - \beta_1)$ is used for scaling the current gradient to ensure stability in the optimization process.

$$\boldsymbol{m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_w E(w_t)}$$

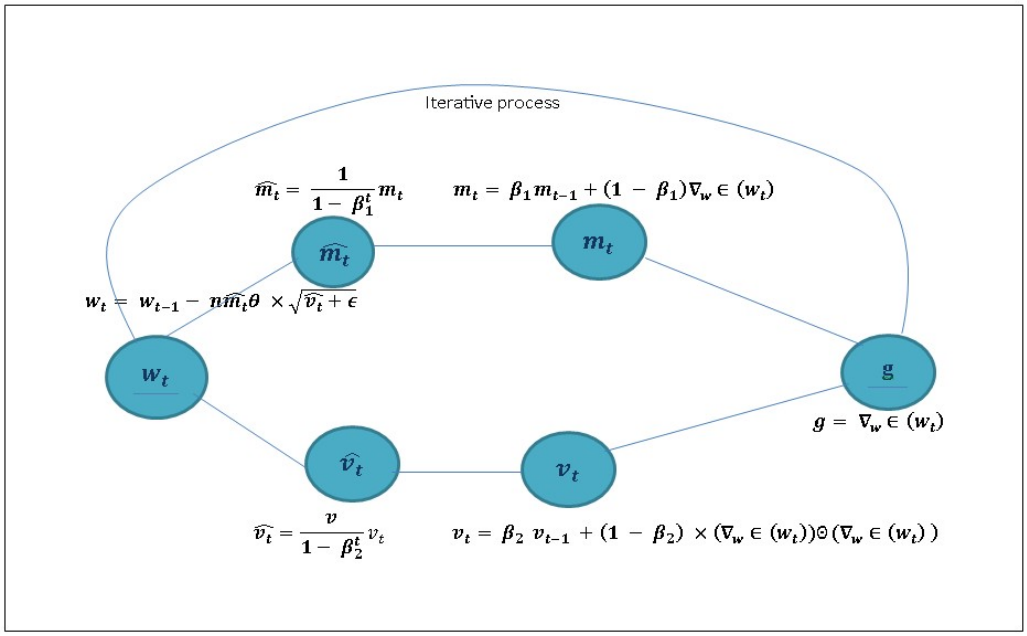


Figure 2.4: Working of ADAM optimizer

2. Second Moment Vector (v_t): The second moment vector, v_t, in the adam optimization algorithm represents a moving average of the squares of the gradients which reflects the magnitude of changes to each parameter over time. This helps optimizer to adjust the learning rates for each parameter. The hyperparameter β_2 controls the exponential decay rate for the second moment estimates, determining how much influence past squared gradients have on the current update. The second moment vector from the previous iteration, v_{t-1}, maintains the decaying average of these squared gradients. The element-wise multiplication of $(\nabla_w E(w_t)) \odot (\nabla_w E(w_t))$, is used to square each component of the gradient vector. This squared term amplifies the influence of larger gradients while diminishing that of smaller ones. The factor $(1 - \beta_2)$ is then used to scale this current squared gradient contribution to ensure stability in the optimization process.

$$\boldsymbol{v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_w E(w_t)) \odot (\nabla_w E(w_t))}$$

3. Bias-Corrected first moment estimate ($\hat{m}_t$): Since m_t and s_t are initialized as vectors of 0's, they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small (i.e., β_1 and β_2 are close to 1). To address this initial bias, the bias-corrected first moment, $\hat{m}_t$, is computed. This adjustment is done for the initial bias by scaling m_t in proportion to the degree of bias present at each iteration t. As the number of iterations increases, the bias correction term $1 - \beta_1^t$ decreases, that reduce the effect of the adjustment and ensure that the first moment estimate accurately reflects the moving average of the gradients without the initial bias.

$$\hat{\boldsymbol{m}}_{\boldsymbol{t}} = \frac{\mathbf{1}}{\mathbf{1}-\boldsymbol{\beta}_{\mathbf{1}}^{\boldsymbol{t}}}\boldsymbol{m}_{\boldsymbol{t}}$$

Bias-corrected second moment estimate ($\hat{v}_t$): Similar to the bias correction for the first moment, the bias correction for the second moment is also calculated. The term $\hat{v}_t$ represents the bias-corrected second moment at iteration t. Here, v_t is the second moment vector, β_2 is the hyperparameter controlling the decay rate for the second moment estimates, and the term $1 - \beta_2^t$ is used to correct the bias that results from initializing the second moment vector as a zero vector. This correction ensures that the second moment estimate accurately reflects the moving average of the squared gradients without the initial bias.

$$\hat{\boldsymbol{v}}_{\boldsymbol{t}} = \frac{\mathbf{1}}{\mathbf{1} - \boldsymbol{\beta}_{\mathbf{2}}^{\boldsymbol{t}}}\boldsymbol{v}_{\boldsymbol{t}}$$

4. Weight Update (w_t): The weight update w_t is an important step in the optimizer. The learning rate η controls the step size of the update. This update rule relies on two components: the bias-corrected first moment ($\hat{m}_t$), which approximates the mean of the gradients, and the square root of the bias-corrected second moment ($\sqrt{\hat{v}_t}$), which approximates the root mean square of the gradients. Additionally, a small constant ϵ is added to prevent division by zero, enhancing numerical stability. w_{t-1} represents the model's weights from the previous iteration or time step.

$$\boldsymbol{w}_{\boldsymbol{t}} = \boldsymbol{w}_{\boldsymbol{t-1}} - \boldsymbol{\eta}\hat{\boldsymbol{m}}_{\boldsymbol{t}}\boldsymbol{\theta}\sqrt{\hat{\boldsymbol{v}}_{\boldsymbol{t}} + \boldsymbol{\varepsilon}}$$

Adam is known for being easy to implement, computationally efficient, and requiring little memory, making it a preferred choice for problems with large data and/or parameters[14].

2.5 Absolute Error And Mean-Absolute-Error

Absolute Error: The absolute error is the difference between the observed value and the actual value. It evaluates the degree of inaccuracies in measurements. The formula for the absolute error $(\boldsymbol{\Delta x})$ is as follows:

$$\boldsymbol{\Delta x = |x_i - x|}$$

In the above equation,

$\boldsymbol{x_i}$ stands for the observed measurement,

$\boldsymbol{x}$ represents the actual value[16].

Mean Absolute Error (MAE):

MAE computes the average of all absolute errors for a set of predictions and actual observations, providing a measure of the overall error magnitude for the group. Often referred to as the L1 loss function, MAE is a widely used loss function in regression problems. Its simplicity and ease of understanding make it a popular choice for measuring errors in regression tasks. MAE also aids in converting learning issues into optimization problems[17].

In the calculation of MAE,

$$\text{MAE} = \frac{\mathbf{1}}{\boldsymbol{n}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} |\boldsymbol{x_i - x}|$$

$\boldsymbol{n}$ signifies the total number of errors,

$\boldsymbol{\Sigma}$ symbolizes the summation operation, implying the total sum of all elements,

$\boldsymbol{|x_i - x|}$ denotes the absolute errors[16].

3. Methodology

This chapter explains the methods used to combine static and dynamic data. Also Vodafone and precipitation dataset ([1]) description and model architectures used for predictions. The methodology adopted for the master's thesis is illustrated in Figure 3.1, which outlines the steps taken to accomplish the thesis objective.

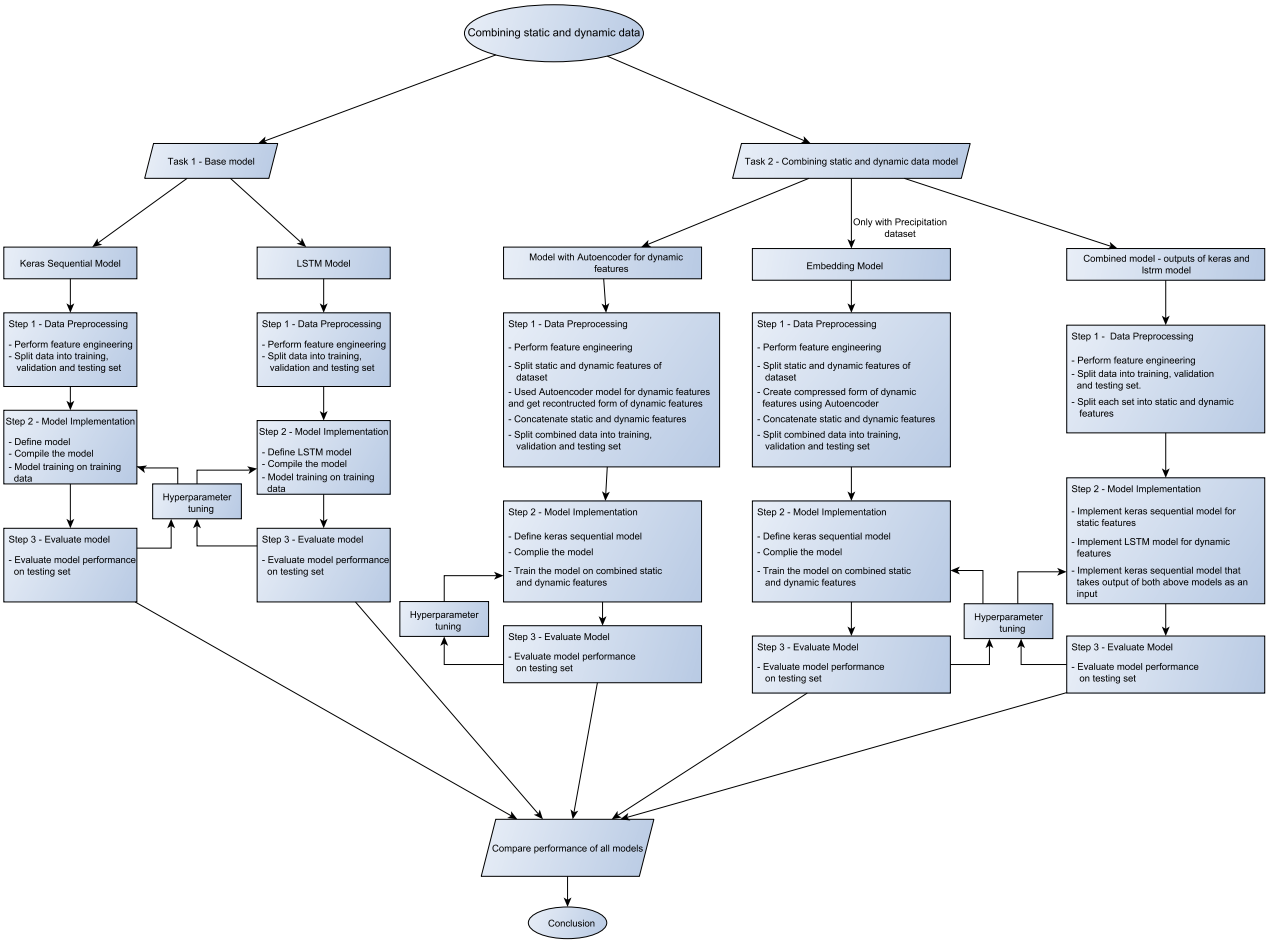


Figure 3.1: Thesis Flow

.

3.1 Dataset Selection

3.1.1 Vodafone Dataset

For this master's thesis, the primary dataset used was the Vodafone retail shops dataset, consisting of both static and dynamic features. This dataset provides information about various Vodafone shops in Germany, originating from the company's consumer sales department. It

consists of data from June 2017 to April 2022 and contains 162 distinctive features. Each record in the dataset provides a monthly summary of sales and associated shop details. Key features within this dataset include rms_id, denotes as a unique identifier for each shop, with each record separated by its unique rms_id. The calmonth feature indicates the date in terms of month and year, and shop_type which represent the nature of the retail shop, whether it's an 'own_shop', 'partner_shop', or 'indirect_shop'.

In addition, the dataset has sales figures for various products, including mobile phones, DSL connections, and cable connections, among others. The "inflow_mobile" target column specifies the quantity of mobile phone sales across different shops. Additionally, the dataset provides further details about each shop, such as the city, zip code, and location. The following Figures 3.23.33.43.5 show the distribution of the numerical features of the dataset, with the index on the x-axis and the corresponding feature values on the y-axis.

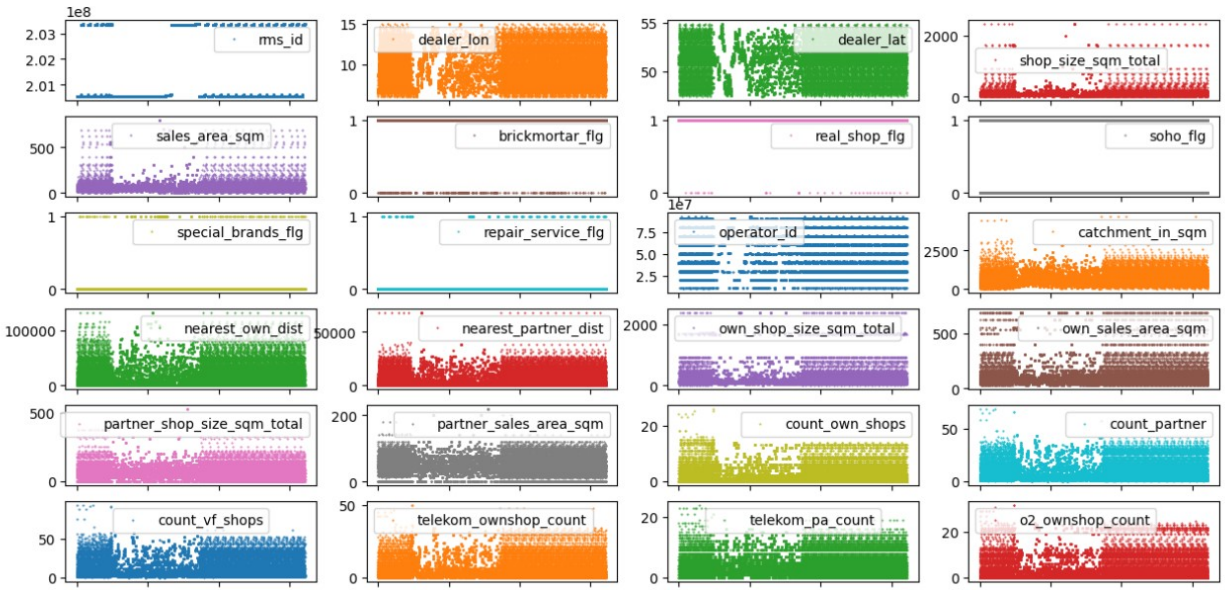


Figure 3.2: Scatter Plot for First 24 Features Numerical Features

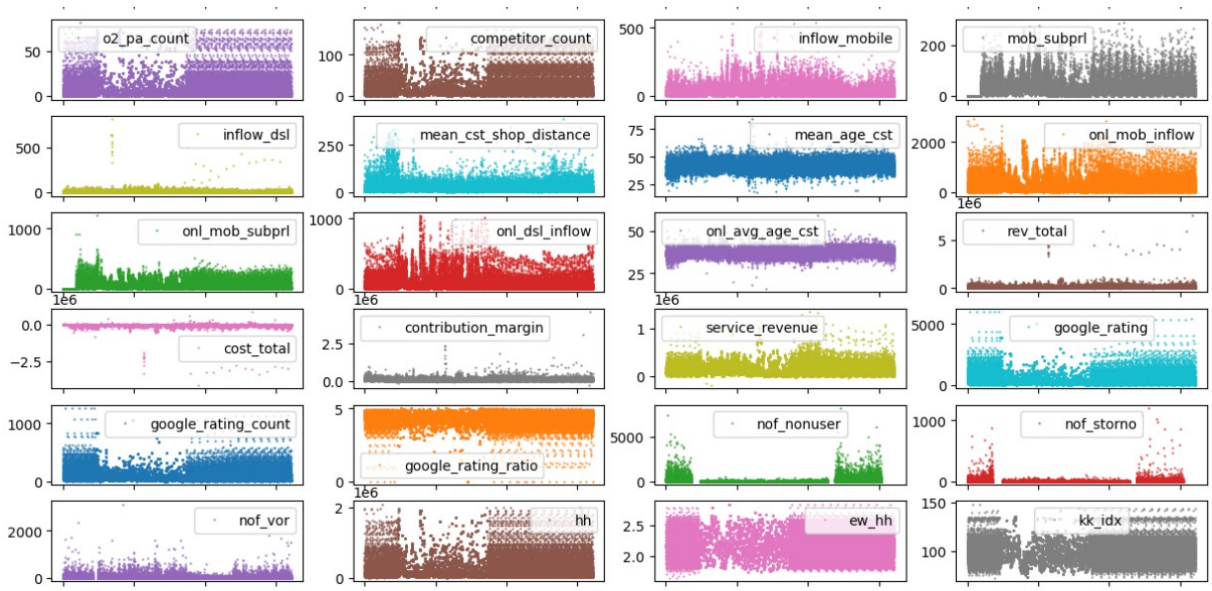


Figure 3.3: Scatter Plot for Next 24 Features Numerical Features

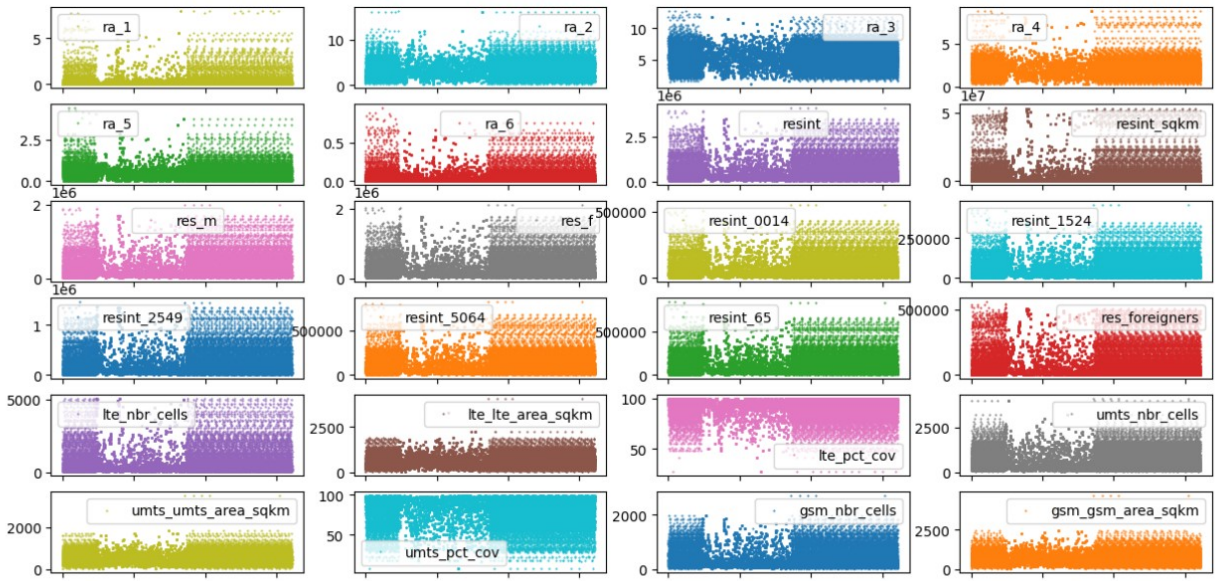


Figure 3.4: Scatter Plot for Next 24 Features Numerical Features

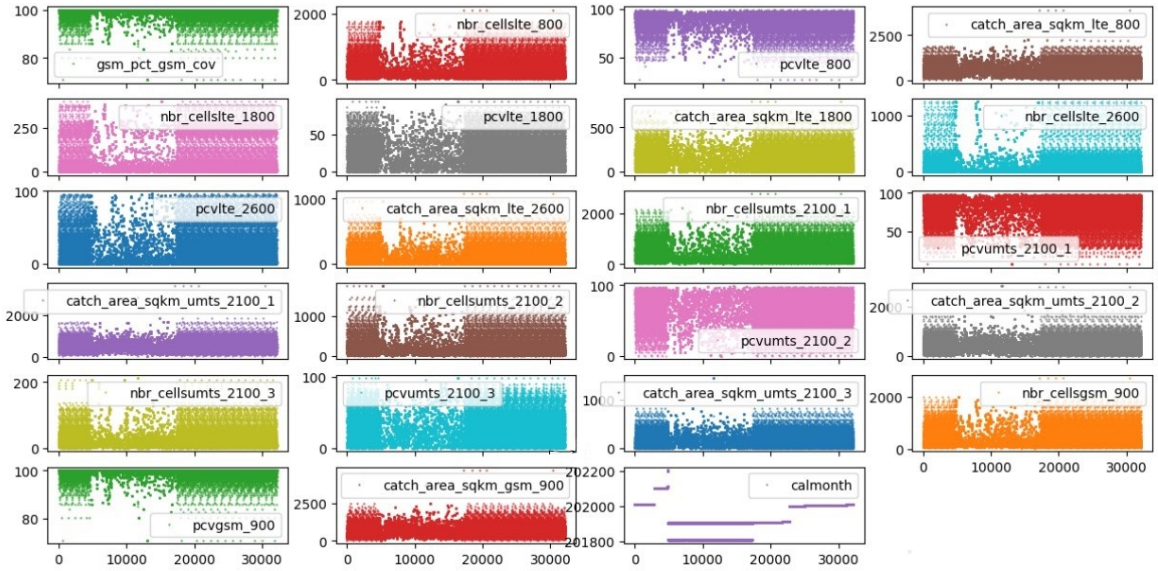


Figure 3.5: Scatter Plot for Last 24 Features Numerical Features

3.1.2 Precipitation Dataset

The second precipitation dataset chose for the comparison of performance is sourced from kaggle's meteorological and soil data[1]. The precipitation data contains features such as surface pressure, wind speeds, temperature, precipitation, etc., for each region in the US, also some weather-related information. This dataset is daily dataset from 2000 to 2020([18]). The soil dataset includes static features such as longitude, latitude, elevation, and geographical distribution of U.S. regions (e.g., north, south, east, west, unknowns). Additionally, it describes land types such as watery land, vegetated land, grasslands, and forest lands ([19]). The primary objective is to predict six distinct drought levels throughout the U.S. which are available on a weekly basis, and stored in a target column 'score'.

For the purpose of this thesis, some changes were made to the original kaggle challenge. While the initial objective on kaggle is to predict drought levels across various U.S. regions using 18 preceding 90-day meteorological indicators for classification task, this master thesis focuses on a regression task. For the changes, the original target column was dropped, and PRECTOT was set as the new target column. PRECTOT represents the percentage of precipitation in various U.S. regions. The dataset is already splitted into training, validation, and testing sets , the training data includes meteorological information from 2000 to 2016, the validation set contains data from 2017 to 2018, and the test set contains from 2019 to 2020. The soil data, consisting of static features, does not contain any date column. Both the meteorological and soil datasets are linked through a common unique identifier, fips. The distribution of some of the features are show in following Figures 3.63.7 also with the index on the x-axis and the corresponding feature values on the y-axis..

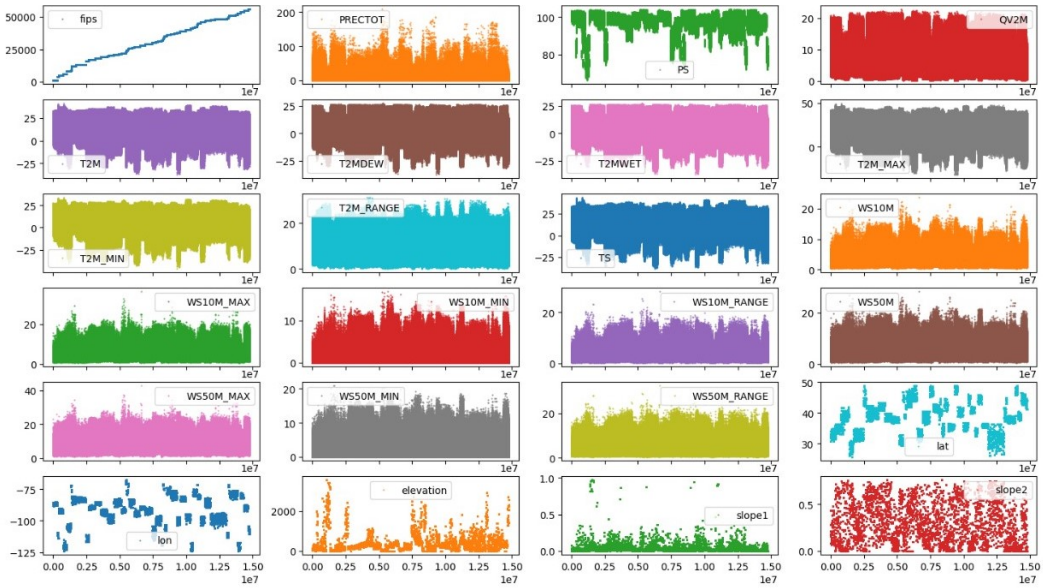


Figure 3.6: Scatter Plot for First 24 Features Numerical Features

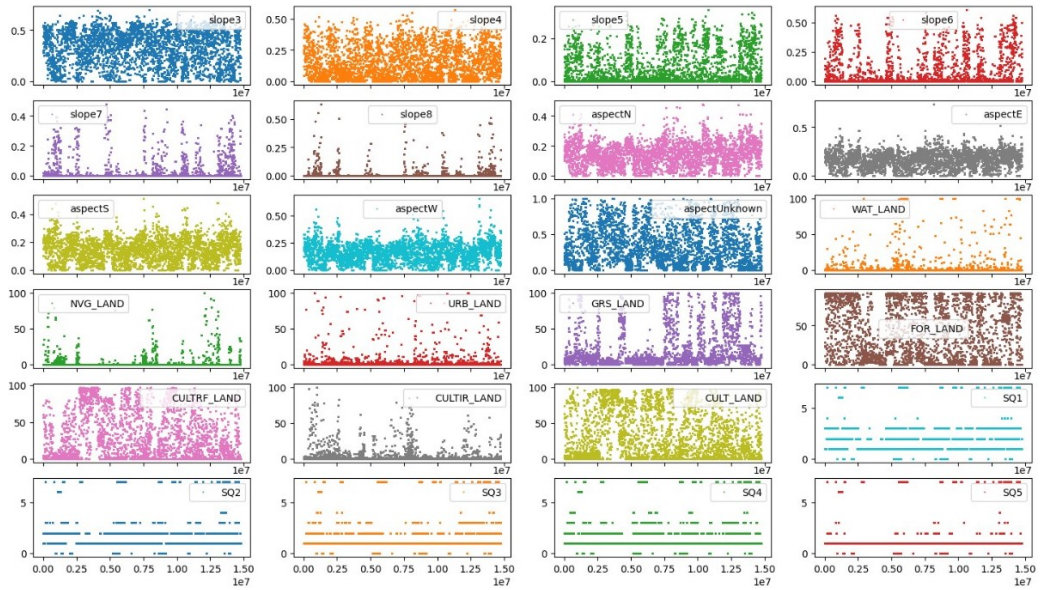


Figure 3.7: Scatter Plot for Next 24 Features Numerical Features

3.2 Dataset Preprocessing

To predict mobile phone sales based on shop types, the dataset was preprocessed first to ensure data is arranged properly before modelling. Initially, the data was adjusted by the date column in ascending order, ensuring the dates were correctly formatted in the 'month-year' structure, specifically in the "%m%y" format. Then, only month was kept for the model, and year column was discarded. The `shop_type` feature, initially had four categories: `own_shop`, `partner_shop`, `indirect_shop` and `other_shops`. For this study, however, the `other_shop` was discarded and chose only the first three types. To treat the null values of columns, firstly calculated the percentage of null values for each column and only columns with less than 75% null values were selected for prediction.

After the removal of columns based on null values, the Vodafone dataset contained 110 columns and 438,196 rows. Among these columns, 14 were categorical features, and 95 were numerical features. For the categorical features, missing values were replaced with the most frequent value, and 'LabelEncoder()' was used to convert them to numerical format. Missing values in numerical columns were filled with the median of each column.

In contrast, the precipitation dataset was already divided into training, validation, and test sets based on date. The dataset was very large, and because of memory issues, only the training set was used for model prediction this set contained data from the years 2000 to 2016 available in the original training set, and the validation and test sets were not used. The dataset was available in two separate files: the meteorological data file (training set) contained 21 time-dependent features, and the other was a soil data file with 32 static features. The first step in preprocessing involved merging these files using the unique fips identifier. Also, the original 'score' target column was removed, and 'PRECTOT' was set as the new target column.

The precipitation dataset had no missing values, and as all features were numerical, more preprocessing was not required. But, for better model predictions, the date column was preprocessed by splitting it into separate day, month, and year columns, and then added to the dataframe as distinct features.

3.3 Dataset Splitting

After preprocessing, the Vodafone dataset was divided into training, validation, and testing sets. To ensure that each `rms_id`, `calmonth`, and `shop_type` was included in all subsets, stratified splitting was used. The data was first grouped by these columns using the `groupby()` function in Python.

From the original Vodafone dataset, 50% was allocated to the training set. From the data remaining after training set allocation, 30% was then added to the validation set. After the allocation of the training and validation data, another 30% of the dataset that was left was allocated to the testing set. This approach ensured that each unique id and all shop types were evenly included in all sets, for accurate model predictions. And the target column 'inflow_mobile' was separated from each of these sets.

After the stratified splitting, the sizes of the Vodafone datasets were as follows: the training set consisted of 171006×110 entries, the validation set had 53304×110 entries, and the testing set

comprised 44562×110 entries.

The precipitation dataset was already distributed, as specified in Section 3.2. Thus, the training set had data from the years 2000 to 2008. The validation set contained data from 2009 to 2010 and the data from time period 2011 to 2012 were allocated to test set. This division resulted in 10,219,104 entries, each with 53 features, in the training set. The validation set included 2,268,840 rows, each with 53 features, and the test set contained 2,271,948 rows, each also containing 53 features.

3.4 Implementations

For the purposes of this thesis, two different base models were used, along with three additional models that aim to integrate both static and dynamic data through varied data handling techniques shown in Figure 3.1. The chosen base models were the sequential model of keras and the lstm.

In this thesis, these models were selected based on their inherent capabilities. The sequential model of keras, with its straightforward architecture, suitable for variety of neural network applications. In contrast, the lstm, with its memory cells, effectively captures temporal dynamics inherent in time series data.

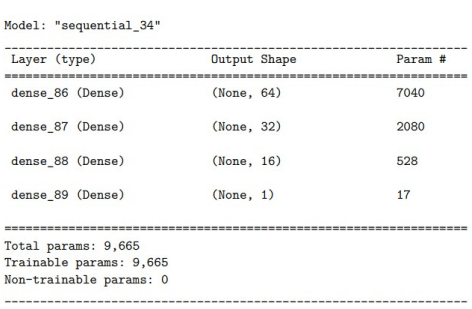
For combining static and dynamic data, various methodologies were implemented. The first approach was using input data reconstructed by an autoencoder 3.4.3, specifically applied to dynamic features for data denoising purposes. This restructured data was then fed into a sequential model with dense layers from keras, combined with static features; this strategy was applied to both Vodafone and precipitation datasets. In a second strategy 3.4.5, a sequential model with dense layers was implemented for the static features, and an lstm model for dynamic features. The results from both static and dynamic data were combined and then input into the another sequential model with dense layers. Furthermore, another model was developed that generates dynamic embeddings for the dynamic features of the precipitation dataset, using the bottleneck layer of the autoencoder 3.4.4. This compressed data was then combined with static features and fed into the sequential model with dense layers of keras. This strategy was applied only to the precipitation dataset. For all models the adam optimizer was used during compilation, MAE as the loss function.

The following sections provide a detailed description of each model architecture and the strategies used to combine static and dynamic data.

3.4.1 First Base Model Architecture

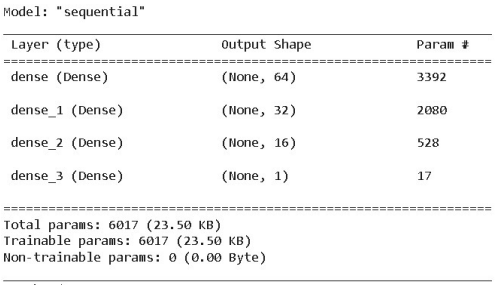
The preprocessed data is fed into a sequential model with dense layers (as shown in Figure 3.83.9), structured with different input, hidden, and output layers. This sequential model serves as a foundational framework, as benchmarks for comparisons with the final models. To predict mobile phone sales of Vodafone shops, the training set, of size 171006 × 109 was used for model training. The validation set, having 53304×109 entries, used during the training phase to check model's performance, while the test set, sized 44562×109, was used for the evaluation process.

For the prediction of PRECTOT, the training set of 10,219,104 × 52, validation set of 2,268,840 × 52, and test set of 2,271,948 × 52 were used. The relu activation function was used at the input layer to address potential non-linearities in the data.



```
Model: "sequential_34"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_86 (Dense)            (None, 64)                7040

 dense_87 (Dense)            (None, 32)                2080

 dense_88 (Dense)            (None, 16)                528

 dense_89 (Dense)            (None, 1)                 17

=================================================================
Total params: 9,665
Trainable params: 9,665
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.8: Keras Sequential model vodafone



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                3392

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 16)                528

 dense_3 (Dense)             (None, 1)                 17

=================================================================
Total params: 6017 (23.50 KB)
Trainable params: 6017 (23.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 3.9: Keras Sequential Model Precipitation

After applying various model architectures with different input layers and different neurons to the Vodafone dataset, the following model architecture was selected as the base architecture for predicting the target columns. It was chosen because it fit to the data well and provided better predictions also the lower MAE compared to other model architectures. As an experiment to see if same architecture would work better with another dataset, it was also fitted to precipitation dataset. The detailed architectures are as follows (Figures 3.8 and 3.9):

- It begins with a dense layer with 64 neurons, prepared to handle an input size of 109 units and 52 units.
- After that, there were dense layers encompassing 32 and 16 neurons.
- The architecture concludes with an output layer, containing a single neuron.

In total, the model designed for the Vodafone dataset used 9,665 trainable parameters, whereas the one for the precipitation dataset had 6,017 trainable parameters. The Vodafone model was trained for 50 epochs, and the precipitation model was trained for 20 epochs, both using an early stopping mechanism to prevent overfitting.

3.4.2 Second Base Model Architecture

The Long Short Term Memory (LSTM) was selected as the second base model for this thesis. In this model, a windowing technique was applied to the data. Specifically, the training, validation, and testing datasets were restructured into sequences of arrays, each with a window size of 3 for the Vodafone dataset and 7 for the precipitation dataset. For Vodafone dataset. a window size of 3 was chosen as its a monthly sales dataset and last 3 months of data allow the model to learn and capture seasonal variations. A small window is enough for the model to understand recent changes in dataset and give correct predictions. And for the precipitation dataset, which is daily weather dataset a window size of 7 was selected. This helps model to learn from weekly cycles and allow it to learn from consistent patterns. Also, for the precipitation dataset, data generators were used to feed data into the model in batches of 8. After forming these windows,

the datasets were reconstructed to the format required by the lstm model, consisting of the number of samples, time steps, and features per time step.

After the restructuring, the training data for Vodafone was reshaped to (171002, 3, 109), the validation set to (53300, 3, 109), and the test set to (44558, 3, 109). For the precipitation dataset, the training data was reshaped to (10219104, 7, 53), the validation set to (2268840, 7, 53), and the test set to (2271948, 7, 53). The following lstm architecture was used for predictions as it was the best performing model compared to other architectures (3.10 3.11):

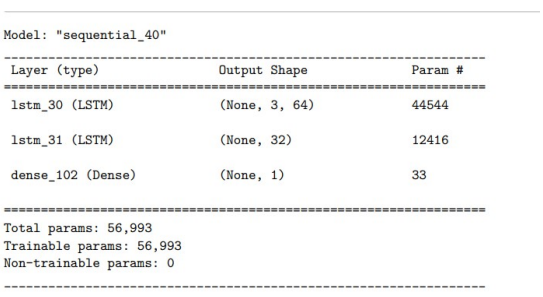


Figure 3.10: LSTM Model Architecture Vodafone

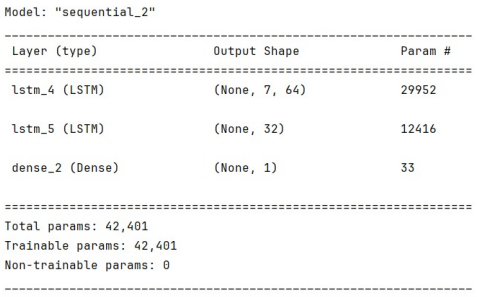


Figure 3.11: LSTM Model Architecture Precipitation

Detailing of architectures:

- The model initiates with an lstm layer containing 64 units.
- Following that, the subsequent layer consists of 32 lstm units respectively.
- The architecture concludes with a dense layer containing a single neuron.

Overall, the lstm model for Vodafone contained 56,993 trainable parameters, while the lstm model for the precipitation dataset used 42,401 trainable parameters. The models were trained for 40 epochs for Vodafone and 20 epochs for the precipitation dataset, with an early stopping mechanism to prevent overfitting.

3.4.3 First Strategy - Autoencoder Model

The Vodafone dataset contained 102 static and 7 dynamic features, not including the target column. In contrast, the precipitation dataset included 32 static and 20 dynamic features, also excluding the target column.

First, both datasets were divided into training, validation, and test sets, as described in Section 3.3. After this split, each set was further splitted based on the feature type. This categorization resulted in subsets named static_train (for static features) and dynamic_train (for dynamic features), static_valid and dynamic_valid, as well as static_test and dynamic_test. The target data (i.e., y_train, y_valid, and y_test) was not included in the static and dynamic splitting process. The dynamic features were then fed into the autoencoder, and the reconstructed dynamic features were obtained, as depicted in Figure 3.12.

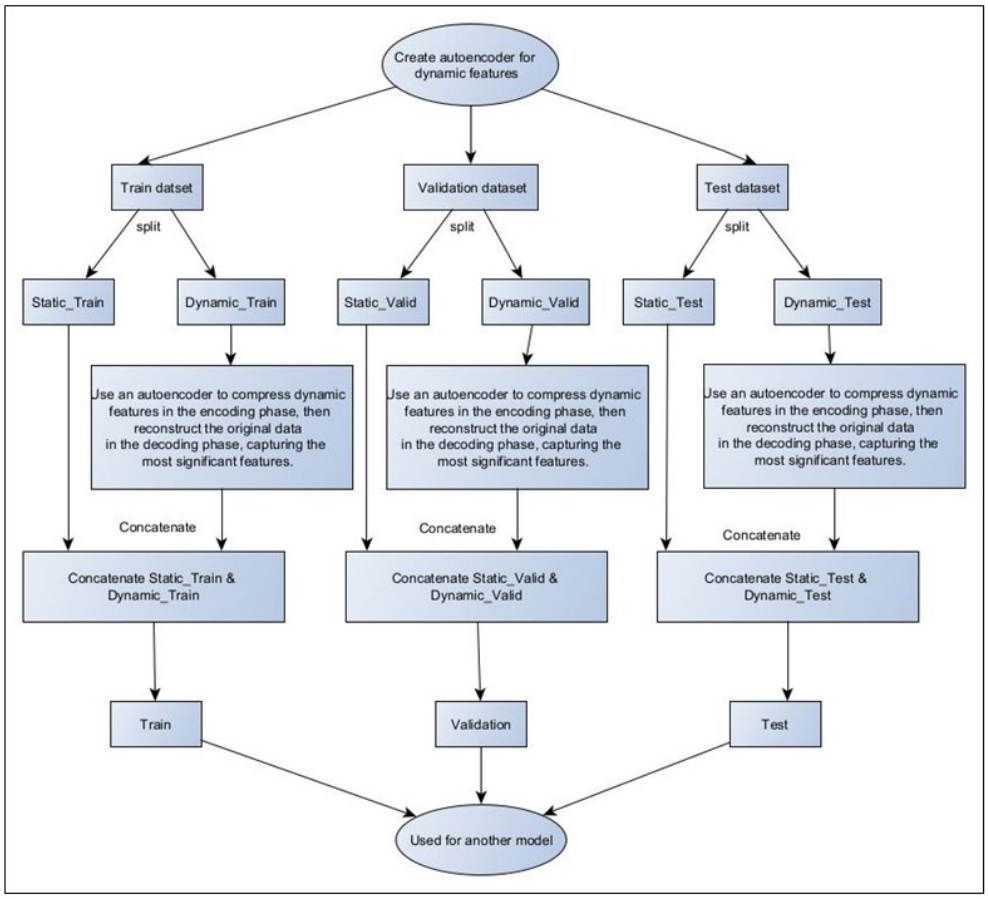


Figure 3.12: First Strategy With Using AutoEncoders

Following are the steps for the autoencoder (Figure 3.12):

- The size of the dynamic training data was determined.
- The goal was to utilize the dynamic data reconstructed by the autoencoder, which uses a dynamic embedding of size 8. This embedding size was chosen because it can work with data well without making the model too complex and it is big enough to capture important patterns in the data but small enough for quick and simple calculations.
- An autoencoder model was created:
  - The encoder, as an initial layer, compresses the data to an 8 dimensional form.
  - The decoder, the another layer, aims to reconstruct the data to its initial form.

The model was trained using the dynamic training data, with the objective of closely getting the original dynamic training data. After this training phase, the autoencoder was used to obtain a reconstructed version of the dynamic data for the training, validation, and test sets.

These reconstructed dynamic data subsets were then combined with their corresponding static datasets, resulting in new combined sets: train (`static_train` + `dynamic_train`), validation (`static_valid` + `dynamic_valid`), and test (`static_test` + `dynamic_test`). These combined datasets were then used for training, validating and testing purpose for another model.

Furthermore, the predefined model structures, as mentioned under 3.4.1 and 3.9, were applied to predict `inflow_mobile` sales and *PRECTOT*. The objective was to observe if there were any performance variations.

3.4.4 Second Strategy - Dynamic Embeddings Model

This model was implemented for a another experiment with the precipitation dataset for PRECTOT prediction, not for the Vodafone dataset. The preprocessing steps applied were the same as those in the 3.4.3 Section. The difference was in the handling of dynamic features in the precipitation dataset. The encoder model was used to encode these features and create compressed versions of the dynamic data, namely `dynamic_train`, `dynamic_valid`, and `dynamic_test`. This compressed data was then combined with the static data: `static_train`, `static_valid`, and `static_test`. The merged data was fed into another keras sequential model architecture, as shown in the Figure 3.13.

The dynamic embedding model architecture description is as follows:

- First, `dynamic_input_dim` was assigned the number of features in `dynamic_train`, which were 20 features. The encoding dimension size was set to 8 same like used in autoencoder model.
- The `dynamic_autoencoder` sequential model was then created. It included an 'encoder' layer, a dense layer that reduced the input to an 8 dimensional encoded form. The relu activation function was used for non-linearity.
- Following the encoder, a 'decoder' layer was implemented to reconstruct the original input from its encoded state. This layer also used the relu activation function.
- The model was compiled using MAE as the loss function to measure the difference between the predicted and actual values.
- The autoencoder was then trained over 10 epochs on the training data, using a batch size of 32. This process involved adjusting the model's internal parameters iteratively to minimize the loss function.
- After training, a new model, referred to as 'encoder,' was created from the original autoencoder. This model, defined as **encoder = Model(inputs=`dynamic_autoencoder.input`, outputs=`dynamic_autoencoder.layers`[0].output)**, consisted only the encoder part. It accepted the same inputs but output the encoded form derived from the autoencoder's first layer.
- Finally, this encoder model was used to make predictions on `dynamic_train`, `dynamic_valid`, and `dynamic_test` datasets. With the goal of transforming the original high dimensional data into a lower dimensional space.

After obtaining the compressed versions of the dynamic data, these were concatenated with the static data and fed into the keras sequential model depicted in the Figure below 3.13. This

model performed well with the combined data and had a lower MAE compared to other model architectures.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 64)                2624

 dense_3 (Dense)             (None, 32)                2080

 dense_4 (Dense)             (None, 16)                528

 dense_5 (Dense)             (None, 1)                 17

=================================================================
Total params: 5,249
Trainable params: 5,249
Non-trainable params: 0
```

Figure 3.13: Sequential Model of Keras With Dynamic Embeddings

.

First, a new sequential model was created, taking the combined training data (`static_train` + `dynamic_train`) as input. This model comprised four layers: three Dense layers with 64, 32, and 16 neurons, respectively, each followed by a relu activation function, and a final Dense layer with a single neuron for output (refer Figure 3.13).

The model was trained for 20 epochs with a batch size of 8 and used 5249 trainable parameters. An early stopping callback was employed to halt training if there was no improvement in '`val_loss`', preventing overfitting.

3.4.5 Third Strategy - Hybrid Model

The preprocessed train, validation and test sets mentioned in Section 3.3 were used for this strategy. The same way mentioned in 3.4.3 the dataframes were splitted into `static_train`, `dynamic_train`, `static_valid`, `dynamic_valid`, `static_test` and `dynamic_test` without a target columns `inflow_mobile` and `PRECTOT`.

Model for Static Features

The model started by taking in 102 static features of the Vodafone dataset and 32 static features of the precipitation dataset, i.e., `static_train`, `static_valid`, and `static_test`. These were first processed in a layer with 64 units. After this, the data flowed into a second layer with 32 units. In both of these layers, a relu function was used as an activation function. After being

processed through the previous layers, the model produced an outcome using a single unit. All these steps were combined together using the Model class, ensuring they worked in sequence. Further, this model was used as an input for another combined model mentioned in Section 3.4.5.

Model for Dynamic Features

Firstly, the datasets `dynamic_train`, `dynamic_valid`, and `dynamic_test` were transformed into numpy arrays for both datasets. These arrays were then reshaped to be accepted with the lstm format, each having a structure of 1 timestep with 7 dynamic features for the Vodafone dataset and 1 timestep with 20 features for the precipitation dataset. The data then fed into an input layer shaped (1, 7) for Vodafone and (1, 20) for precipitation, then going through two lstm layers. The first lstm layer, consisting of 64 units, was set to return sequences, for stacking of lstm layers. The following lstm layer, contained with 32 units, did not return sequences. The information was then processed by an output layer with a single neuron. The entire structure was combined using the Model class, connecting the input and output layers.

Static and Dynamic Features Hybrid Model

Firstly, the architecture combines the outputs of two models: the `static_model` (3.4.5), which was a feed-forward neural network, and the `dynamic_model` (3.4.5), an lstm model (refer Figure 3.14). Once combined, the data was processed by two dense layers. The first layer had 64 neurons and used a relu activation function. Then the second layer with 32 neurons, also with same activation function. The result is a combined tensor that holds information from both static and dynamic parts.

The combined data was then given to an output layer, consisting a single neuron that used a linear activation. This structure made the model suitable for regression tasks, where the aim was to predict a continuous value. The entire flow, from both the static and dynamic inputs to the output, was then combined using the Model class.

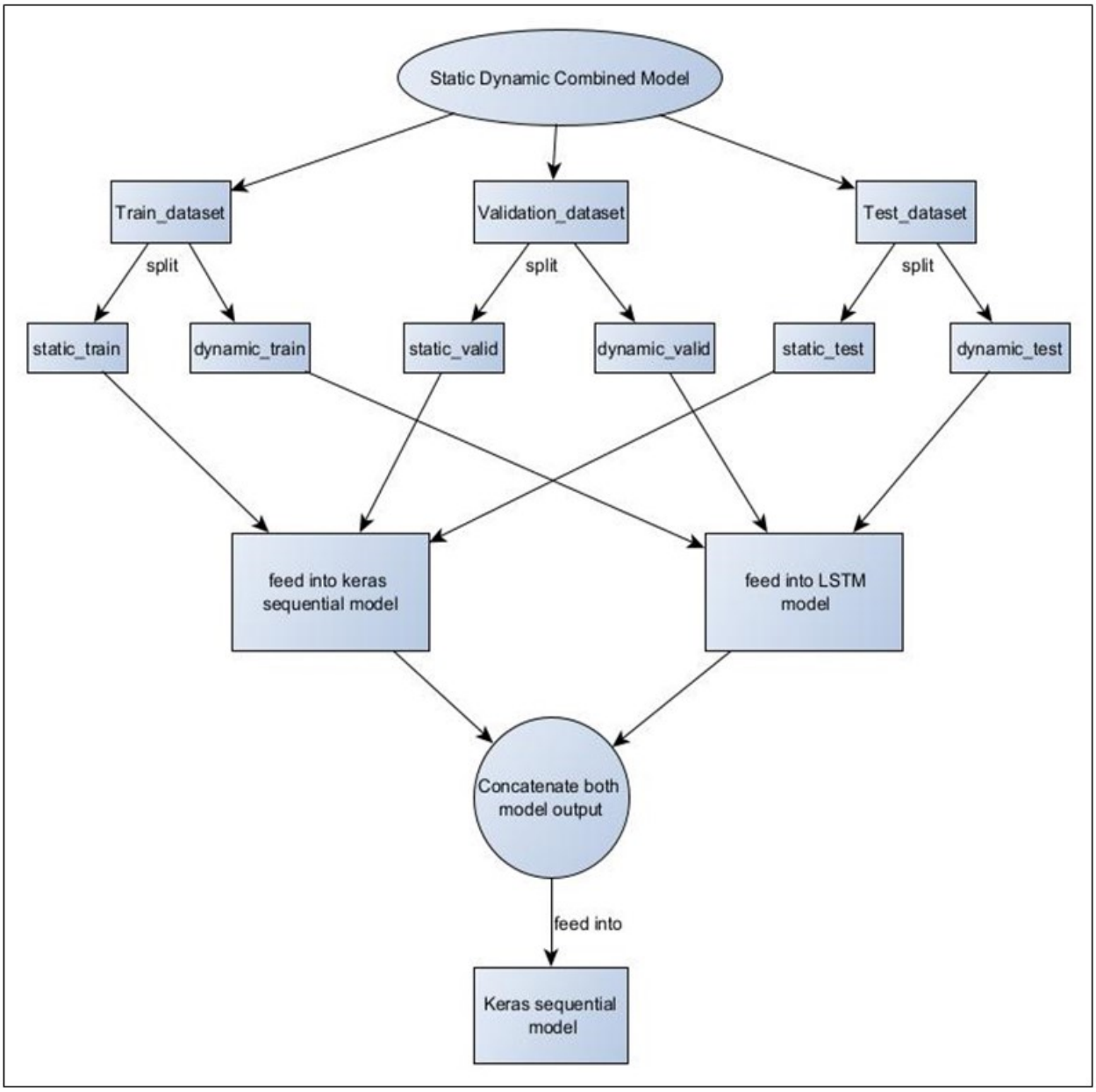


Figure 3.14: Hybrid Model

.

Before the training phase, the model was compiled using the adam optimizer and the MAE as its loss function.

The training was performed on both the `static_train` and `dynamic_train` datasets, targeting the `y_train` (inflow_mobile and PRECTOT) values. This training was done on 50 epochs in batches of 32. Also, the model's performance on unseen data was evaluated using validation datasets: `static_valid`, `dynamic_valid`, and `y_valid`.

3.5 IDE and Frameworks

This section describes the tools, programming languages, and libraries used in this thesis for the purpose of data analysis, data preprocessing, and model implementation. For the thesis tasks, the Python programming language was used as it provides various library support, and its easy syntax helped in complex data operations and model development. For data preprocessing, data visualization, and model implementation, different Python libraries were used, like Seaborn, Matplotlib, NumPy, Pandas, and Tensorflow. The IDEs used for performing all these tasks were Jupyter Notebook and Google Colab. For the Vodafone `inflow_mobile` sales prediction task, Jupyter Notebook was the preferred IDE due to its interactive computing environment, which helped in making the analysis more easy. For the precipitation `PRECTOT` prediction task, as the dataset was huge and to avoid RAM issues, Google Colab was used, specifically the Pro version, which have pre-installed all necessary libraries and provides more memory and faster GPUs.

Tensorflow

Tensorflow is an open-source software library developed by researchers and engineers on the Google Brain Team within Google's Machine Intelligence research organization. It was released to the public in 2015. The library is used for numerical computation and large-scale machine learning, combining machine learning and deep learning models and algorithms [20][21].

The developers can create data-flow graphs in Tensorflow, where the nodes represents mathematical operations, and the graph edges denotes multidimensional data arrays, which are known as tensors. The keras library is used for creating nodes and layers and creating connections between them. Both nodes and tensors in Tensorflow are python objects, and Tensorflow applications are themselves python applications[20][21].

Tensorflow provides multiple application programming interfaces (APIs) classified as Low-Level APIs and high-level APIs. The Tensorflow core, denotes the low-level API, provides broad programming control and is recommended for machine learning researchers who require good control over their models. In contrast, the high-level APIs, such as tf.contrib.learn, are built on top of Tensorflow core, providing ease of use and consistency for common repetitive tasks, so providing a more user-friendly learning ways[20][21].

Tensorflow is capable of training and executing deep neural networks for various tasks, including digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models, natural language processing, and others. It also supports production predictions, allowing the usage of the same models for training and model prediction processes[20][21].

Pandas

Pandas is an open-source python library developed by Wes McKinney and released in 2008. It offers fast, powerful, and flexible tools for data analysis and manipulation. The "Pandas" name is derived from the term "Panel Data," a term for multidimensional structured data sets. Pandas increases python's capabilities for data loading, alignment, manipulation, and merging, making it a useful tool for data wrangling, the process of converting and arranging raw data into a more

accessible format. Pandas is better in handling statistical data in the tabular form, including matrices with row and column labels, and time series data. It was specifically designed to work with 2 dimensional data structures known as DataFrames, where each column contains values of one variable, and each row consists of one set of values from each column[22][23].

Pandas is useful in managing missing data and offers the flexibility to easily insert or delete columns from dataframes. It provides useful tools for data grouping, allows split-apply-combine operations on datasets for aggregation or transformation. Datasets can be easily merged and restructured, that enhances the usability of dataset. Pandas is also useful in different stages of the data science pipeline, including data cleaning, transformation, analysis, modeling, visualization, and reporting[23].

NumPy

NumPy is a popular scientific computing library for python, used for complex mathematical and scientific calculations. It provides high-level mathematical functions and a multidimensional array structure, known as ndarrays, ndarrays are used for manipulating large datasets. Also, it works as an efficient multidimensional container of generic data. The term "NumPy" is short for "Numerical Python"[24][25].

While NumPy offers limited functions for data analysis, many other data analysis libraries, such as SciPy, matplotlib, and pandas, depend on it. These libraries rely on NumPy arrays for input and output. Libraries like Tensorflow and scikit-learn also uses NumPy arrays for matrix multiplications. As data science includes handling large tables and matrices and performing complex calculations to extract information from data, NumPy simplifies this process with providing different mathematical functions. Furthermore, NumPy has the ability to define arbitrary data types and allow them to to integrate properly with a wide array of datasets [24][25].

Matplotlib

Matplotlib is a cross-platform data visualization and graphical plotting library for python. John D. Hunter developed the first version between 2002 and 2003. Since then, Matplotlib's popularity has grown with the rising interest in the python programming language. This library provides the creation of various diagrams and visual representations in multiple formats suitable for publications. With just a few lines of code, Matplotlib enables the development of plots, histograms, power spectra, bar charts, error charts, and scatterplots, among others. While Matplotlib can be invoked within python scripts, it is also compatible with jupyter notebooks and web application servers. Matplotlib considered as an alternative to MATLAB's plotting capabilities, Matplotlib offers integration with numpy and scipy, distinguishing itself as a free, open-source solution compared to MATLAB's costly, closed-source platform[26][27][28].

Matplotlib's scripting layer interacts with two APIs: the first, known as the pyplot API, is encapsulated within matplotlib.pyplot and operates on top of the hierarchy of python code objects. The second, an object-oriented API, grants direct access to Matplotlib's backend layers, comprising a collection of objects that users can assemble with greater flexibility than is permissible via pyplot[27][28][26].

Moreover, Matplotlib accommodates object-oriented programming approaches. It further increases user interaction by offering options within the display window, such as the ability to zoom into graphs, navigate charts bidirectionally, and adjust the display format[27][28][26].

4. Results

This chapter presents the results of the strategies explained in the methodology chapter 3, applied to both the Vodafone and precipitation datasets. It includes the performance of base models, as well as the results of static and dynamic data combining strategies on training and validation data of both datasets.

4.1 Performance of First Base Model Architecture

The first base model architectures, as described in Section 3.4.1, were employed for two distinct tasks: predicting mobile sales and PRECTOT values. These models revealed insightful patterns. For mobile sales prediction, the model (Figure 3.8) showed a smooth learning process, with both training and validation losses showing a steady decrease (as shown in Figures 4.1 and 4.2). This indicated not only effective learning from the training data but also better generalization to unseen data. The losses were constant after a certain number of epochs, showing that no further improvements were achieved with additional training.

The MAE achieved by this model was **12.5**, which was relatively better compared to the mean of the target column inflow_mobile, which was **17.7**. The MAE was approximately 71% of the mean, a substantial portion of the average target value. Also, the MAE was lower than the standard deviation of **33.7** of the target column, showing that the model's predictions were not precisely accurate but closer to the actual values.

In contrast, the same base model architecture (Figure 3.9) applied for the PRECTOT predictions showed a different pattern. While the training loss decreased steadily and stabilized (as indicated in Figure 4.3), the validation loss (as seen in Figure 4.4) showed the fluctuations. The model had challenges with generalizing on the unseen data. Although there were fluctuations with validation data, the model still managed to achieve a better MAE of **2.4** with the precipitation data.

The mean of the target column PRECTOT of the precipitation dataset was **2.6**, and the standard deviation was **6.1**. The MAE being close to the mean indicated that the average error was a significant portion of the average precipitation target values. Also, the MAE being lower than the standard deviation suggested that the model's predictions, despite the fluctuations, were more consistent than the wide spread of the actual data.

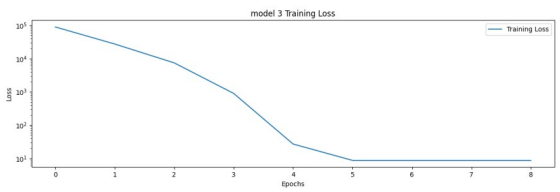


Figure 4.1: Vodafone Base Model Training Loss

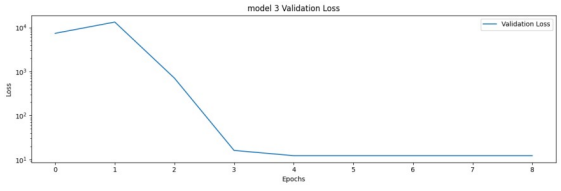


Figure 4.2: Vodafone Base Model Validation Loss

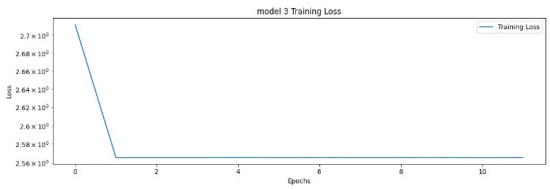


Figure 4.3: Precipitation Base Model Training Loss

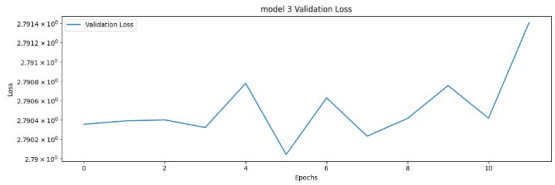


Figure 4.4: Precipitation Base Model Validation Loss

Serving as a comparative benchmark for final results, the base models demonstrated different learning process across various datasets. As shown in Figure 4.5, the models performance on the Vodafone dataset, was checked by MAE among different shop types: the MAE was **111.2** for own_shop which was slightly better compared to the mean of own_shop column at **110** but not as good when considering standard deviation of **70.7**. The MAE for partner_shop was **29.1** and better compared to the mean of the partner_shop column at **31** and standard deviation of **25.7**. For indirect_shop, the MAE was **2.7**, which was quite good considering the mean of the indirect_shop column was **4.4** and the standard deviation was **12.6**. This indicated that the model performed well with indirect_shop predictions but was less accurate for own_shop.

These results are further illustrated in Figure 4.6, which depicts the distribution of absolute errors across the different shop types. The errors were more widely distributed for the own_shop and, to a lesser extent, for the partner_shop, while most of the error frequencies for the indirect_shop clustered near zero.

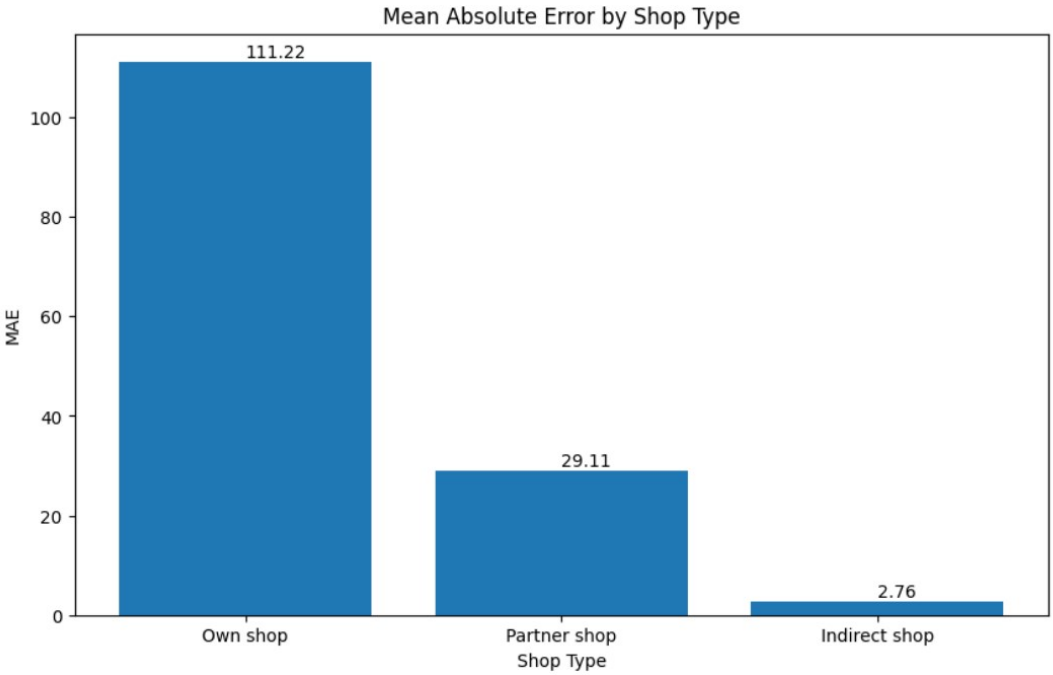


Figure 4.5: Base Model MAE Plot of Vodafone Shop Types

Figure 4.6: Base Model Absolute Error Distribution of Shops Types Vodafone

In contrast, the model with a lower MAE for the precipitation dataset performed well, as shown in Figure 4.7, where the distribution of absolute error was clustered around zero, indicating comparatively good performance. This pattern was also observed in another figure 4.8, which displayed the distribution of absolute errors for each region in the precipitation dataset, with error frequencies concentrated near zero for all regions.

The model also performed well considering the mean and standard deviations of the region columns. Specifically, the 'aspectN' column (the north region of the US) had mean of **0.14** and a standard deviation of **0.07**, the 'aspectW' column (west region of the US) had a mean of **0.16** and a standard deviation of **0.08**, the 'aspectS' column (south region of the US) had a mean of **0.14** and a standard deviation of **0.08**, the 'aspectE' column (east region of the US) had a mean of **0.16** and a standard deviation of **0.08**, and the 'aspectUnknown' column (not specific region in the US) had a mean of **0.35** and a standard deviation of **0.27**. Given the range of mean and standard deviation values across regions, the clustering of errors near zero shows the model's better performance in predicting PRECTOT values.

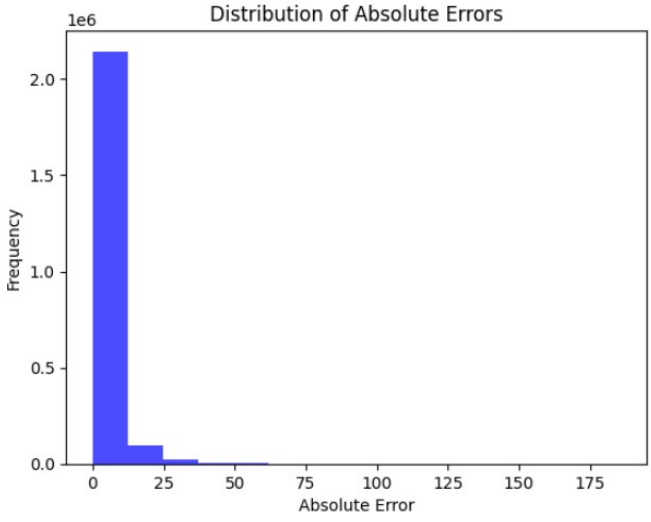


Figure 4.7: Base Model Absolute Error Distribution Precipitation

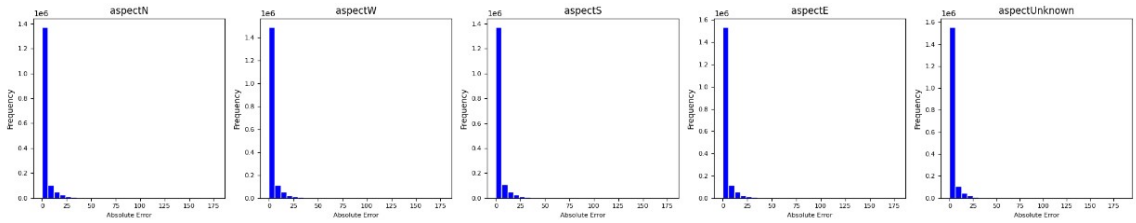


Figure 4.8: Base Model Absolute Error Distribution Regions of Precipitation

4.2 Performance of Second Base Model Architecture

The second base model used for comparison with the main strategy was an lstm with a windowing approach, as detailed in Figure 3.10 and described in Section 3.4.2. This lstm achieved the same MAE of **12.5** on the Vodafone dataset. Its training and validation losses for the first model (Figures 4.9 and 4.10) showed a consistent decline. However, the irregular pattern in the validation loss, particularly in the initial phase, indicated challenges in model's learning stability.

In contrast, the same lstm model applied to the precipitation dataset (Figures 4.11 and 4.12) showed a decline in losses, achieving an MAE of **2.4**, the same performance as the first base model.

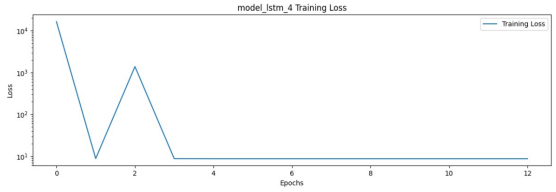


Figure 4.9: Vodafone LSTM Training Loss

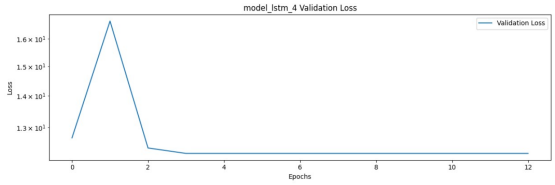


Figure 4.10: Vodafone LSTM Validation Loss

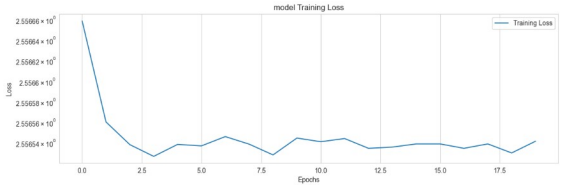


Figure 4.11: Precipitation LSTM Training Loss

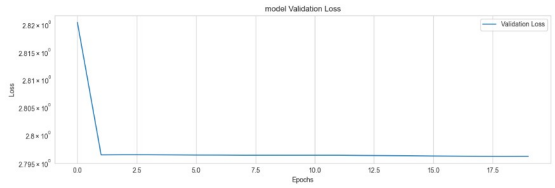


Figure 4.12: Precipitation LSTM Validation Loss

With a window of the past 7 months, the lstm achieved similar MAEs for each individual shop types also, as demonstrated in Figure 4.13. However, the model showed better performance in terms of absolute errors. The distribution of errors for the own_shop and partner_shop had a higher number of error values concentrated near 0 compared to the first base model, as seen in Figure 4.14. The model also performed well with indirect_shop.

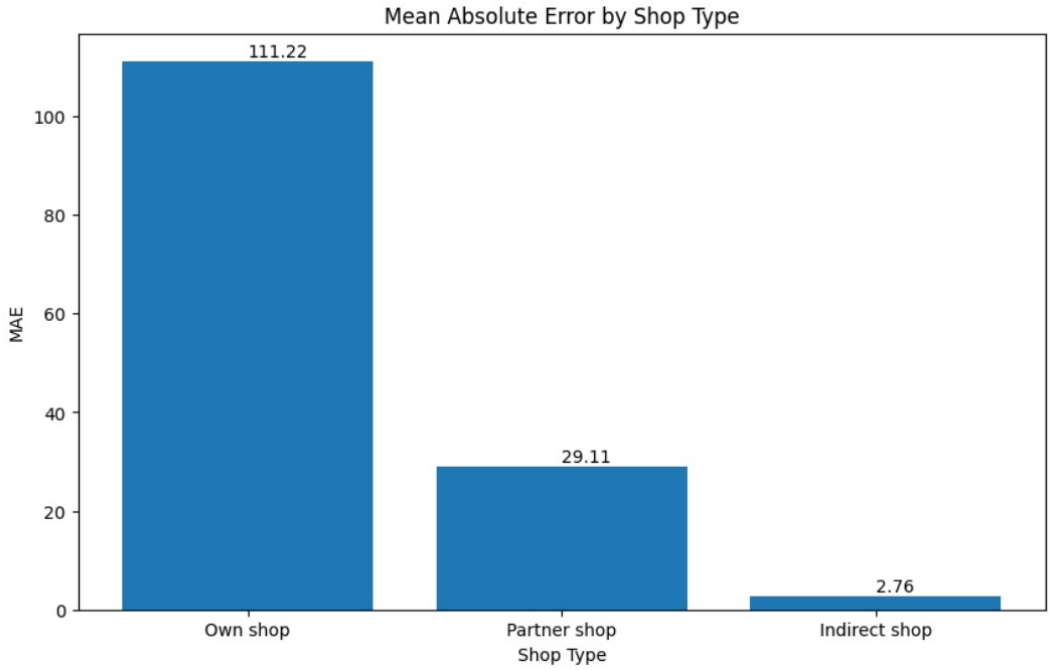


Figure 4.13: LSTM Model MAE Plot of Vodafone Shop Types

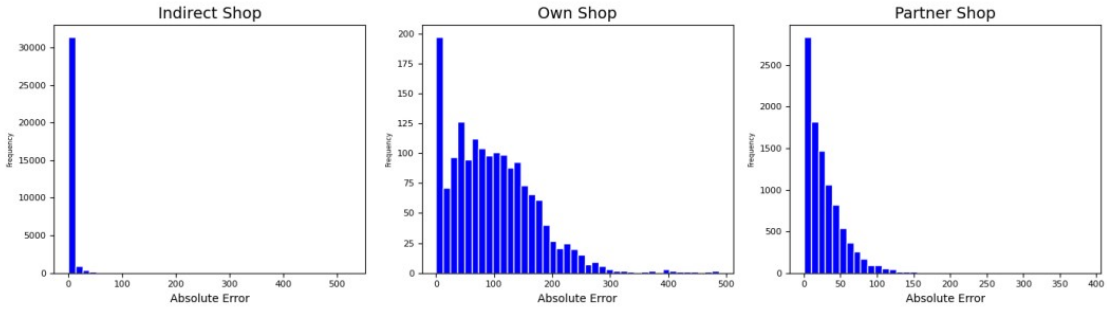


Figure 4.14: LSTM Model Absolute Error Distribution Shop Types Vodafone

For the precipitation dataset, the lstm model with a window of the past 3 days also performed similar to the first base model. Figure 4.15 shows that many of the error values were clustered near zero, also with a good MAE. Similarly, the distribution of errors across the corresponding regions showed a higher number of values near zero (refer Figure 4.16).

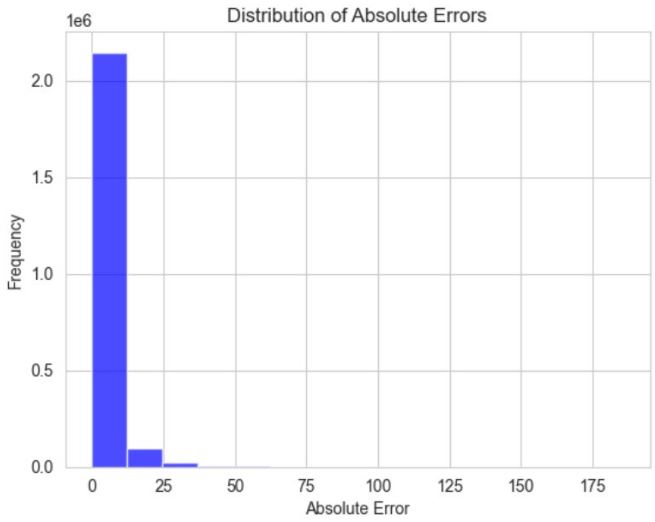


Figure 4.15: LSTM Model Absolute Error Distribution Precipitation

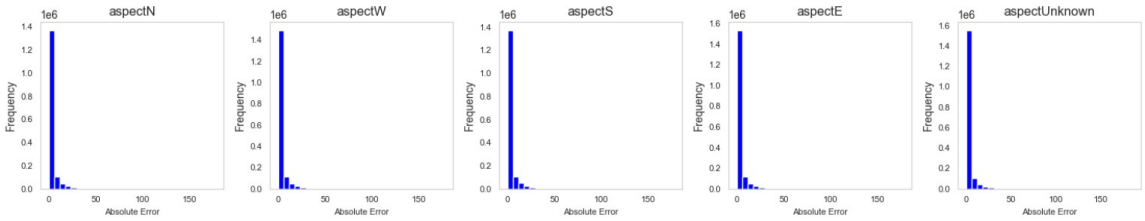


Figure 4.16: LSTM Model Absolute Error Distribution of Regions Precipitation

4.3 Performance of First Strategy - Autoencoder Model

The method used for this strategy is detailed in Section 3.4.3. The same model architecture used for the initial base model (see Section 3.4.1) was applied to the Vodafone dataset. After reconstructing dynamic features, both the training and validation losses consistently decreased, as demonstrated in Figures 4.17 and 4.18, indicating effective model performance. However, the model MAE remained at **12.3**, similar with the MAEs of the first and second base models.

With the precipitation dataset, the model performance was similar to that of the base model, with training loss decreasing and validation loss initially decreasing, then rising slightly before stabilizing (as shown in Figures 4.19 and 4.20). Despite these fluctuations, the model achieved an MAE of **2.4**, also similar with the results from the first and second base models.

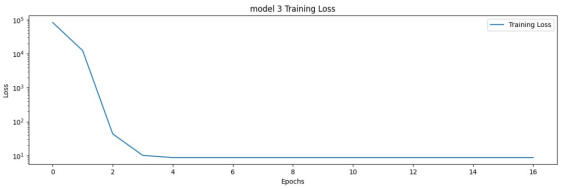


Figure 4.17: Autoencoder Model Vodafone Training Loss

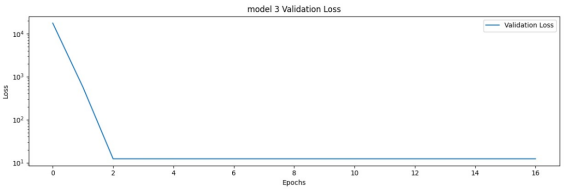


Figure 4.18: Autoencoder Model Vodafone Validation Loss

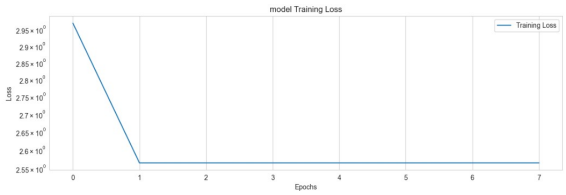


Figure 4.19: Autoencoder Model Precipitation Training Loss

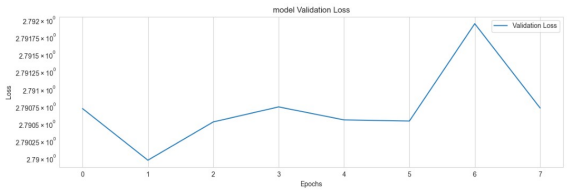


Figure 4.20: Autoencoder Model Precipitation Validation Loss

The model showed better performance in terms of MAE for each shop type. The MAE for own_shop type was lower compared to the base models, at **103.7** (as shown in Figure 4.21), whereas the base models MAE for own_shop type was **111.2** . The MAEs for indirect and partner_shop types remained the same as those of the base models. Also, the distribution of absolute errors, as shown in Figure 4.22, indicated that the model had more values clustered near zero compared to the first base model for both partner_shop and own_shop types. The first base model had **1750** values near zero for partner_shop and **120** for own_shop, whereas the model with reconstructed dynamic data had **2500** values near zero for partner_shop and **160** for own_shop.

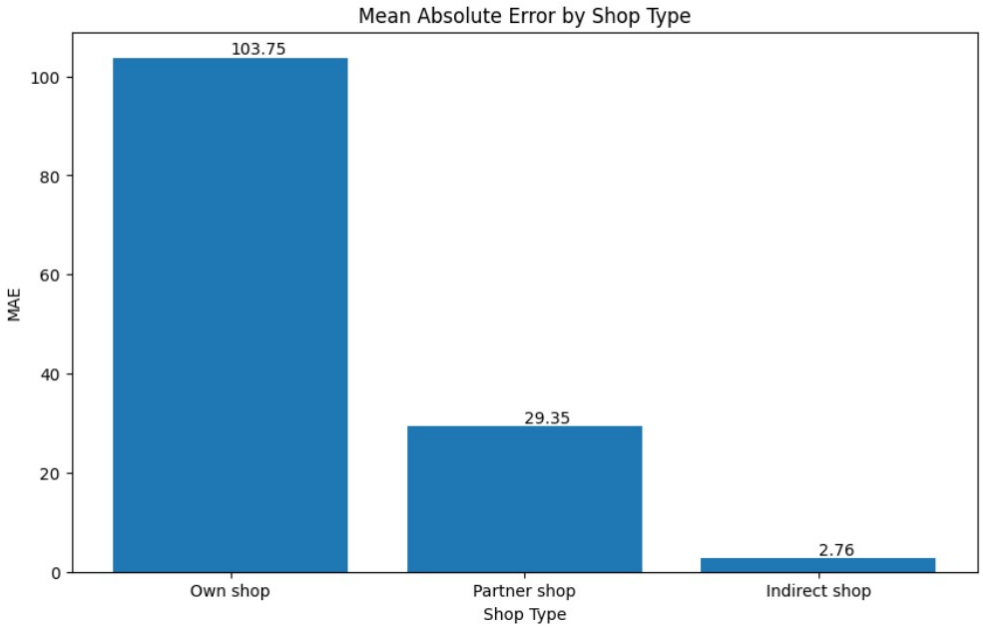


Figure 4.21: Autoencoder Model MAE Plot of Vodafone Shop Types

Figure 4.22: Autoencoder Model Absolute Error Distribution Shop Types Vodafone

There were no changes in performance with the precipitation data compared to the first and second base models. The totals and distribution of absolute errors were also similar, as depicted in Figures 4.23 and 4.24.

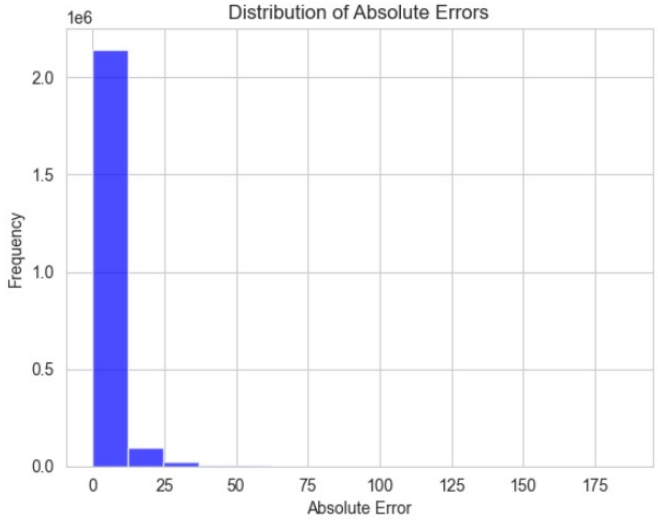


Figure 4.23: Autoencoder Model Absolute Error Distribution Precipitation

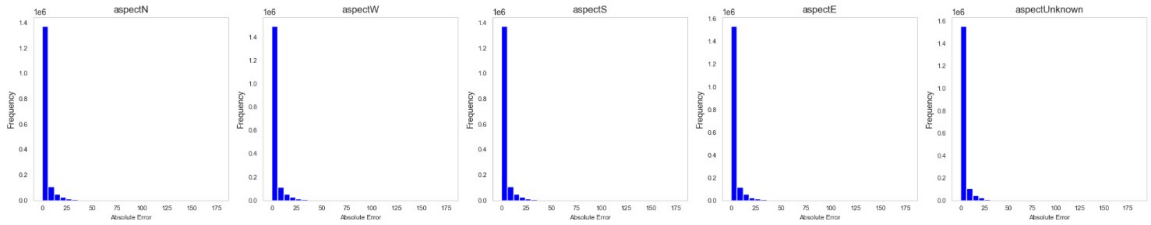


Figure 4.24: Autoencoder Model Absolute Error Distribution of Regions Precipitation

In the case of precipitation data, compressing 20 dynamic features into an 8 dimensional space using an autoencoder did not change the performance of the sequential model, which showed that the autoencoder's encoding-decoding process did not add any another value beyond the original features. This implies that the encoding dimension was too limited to capture the essential information of the dynamic features. Conversely, with the Vodafone dataset 7 dynamic features, expanding them into an 8 dimensional space did not lead to information loss and

may have allowed the autoencoder to capture a beneficial transformation of the data. This transformation helped the autoencoder in identifying more significant patterns, thus improving the performance of the keras sequential model. This highlights the importance of arranging an autoencoder's encoding capacity with the complexity and number of the features being processed.

The use of an autoencoder did not change the original nature of the dynamic features; they remained dynamic, leading to similar results for both datasets when compared to the lstm, the second base model. As lstm models are expert at capturing temporal dynamics of features, these results showed that the autoencoders did not enhance the feature set sufficiently to outperform the lstm.

4.4 Performance of Second Strategy - Dynamic Embeddings Model

The approach used for this model is explained in Section 3.4.4. This method was utilized only on the precipitation dataset, and the results were the same as those of the other models. The model with feeding dynamic embeddings for dynamic features also had the same performance as the base models 4.2 4.1 and the model with an autoencoder 4.3. The training loss was dropping (see in Figure 4.25), and the validation loss was initially going down but then began to increase, as shown in the Figure 4.26, indicating the models difficulty in handling unseen data. This model also achieved an MAE of **2.7**.

Figure 4.25: Embedding Model Precipitation Training Loss

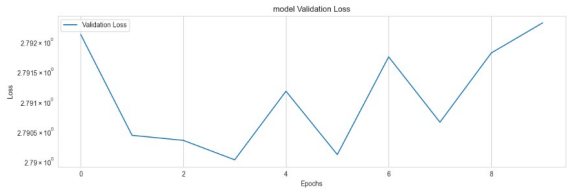


Figure 4.26: Embedding Model Precipitation Validation Loss

Also the absolute error distributions were the same as the other model's results as shown in the following Figures 4.27, 4.28.

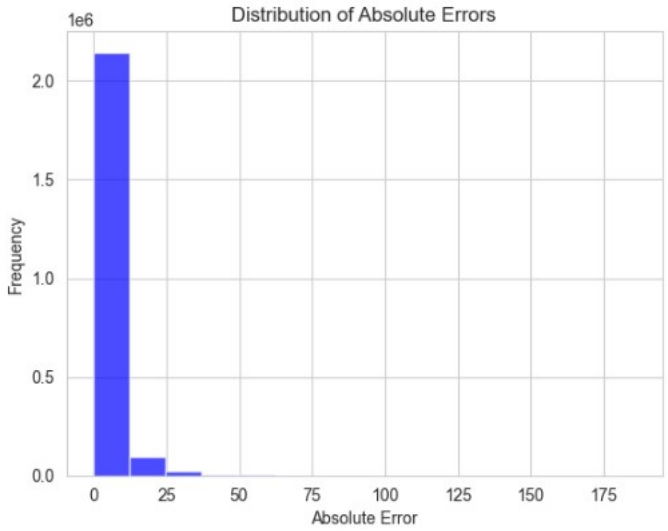


Figure 4.27: Embeddings Absolute Error Distribution Precipitation

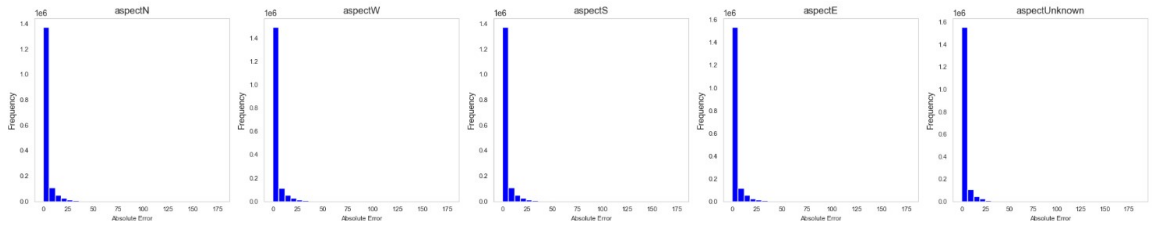


Figure 4.28: Embeddings Absolute Error Distribution of Regions Precipitation

Compressing the dynamic features using an encoder and achieving the same results as the base models indicates that reducing the dimensionality of data helps in managing high dimensional datasets more effectively. The dimensionality reduction can help in faster computation and less memory usage. Along with capturing the intrinsic structure of the data, which are important for understanding the data better, this approach provides competitive results without the loss of any valuable information.

4.5 Performance of Third strategy - Hybrid Model

The performance of the hybrid model with the Vodafone dataset was quite impressive. The training loss declined in the initial epochs (see Figure 4.29), showing better performance on the training data and the validation loss also decreased quickly at first (see Figure 4.30). After some epochs, both losses became stable, showing no further improvement in the model's learning, and it achieved an MAE of **7.3**, which was better than the base models.

With the precipitation dataset, the training loss showed a declining nature at the initial epochs and later became stable (see Figure 4.31), indicating that the model learned to reduce its error on the training set and then made only small improvements after the initial learning. The validation loss however (see Figure 4.32), showed a zigzag pattern with loss, making the model's performance not better on validation data and achieving the same MAE as the base models, which was **2.4**.

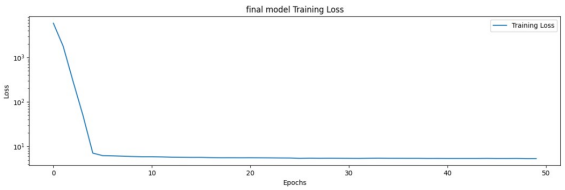


Figure 4.29: Hybrid Model Vodafone Training Loss

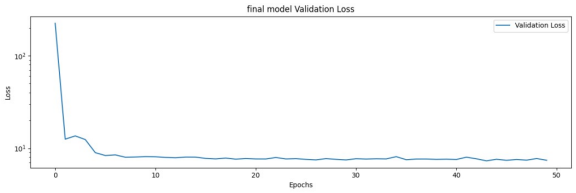


Figure 4.30: Hybrid Model Vodafone Validation Loss

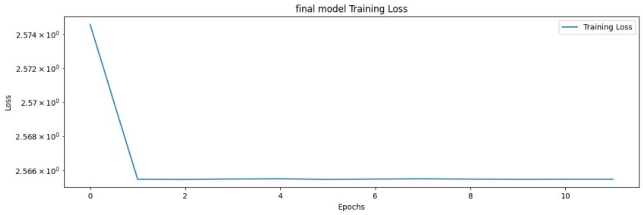


Figure 4.31: Hybrid Model Precipitation Training Loss

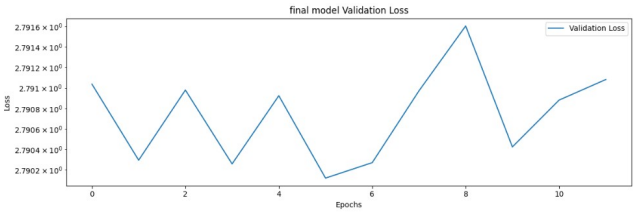


Figure 4.32: Hybrid Model Precipitation Validation Loss

For the Vodafone dataset, the hybrid model consistently performed better on each shop type, as shown in Figure 4.33. The MAE for each shop type was less compared to the base models. The MAE achieved by the model were **45.4** for own_shop type, **16.9** for partner_shops, and **2.6** for indirect_shop. Also, the absolute error for each specific shop type showed a higher number of values clustering near 0, particularly for own_shops and partner_shops, as illustrated in the Figure, when compared to the base models (compare Figure 4.34).

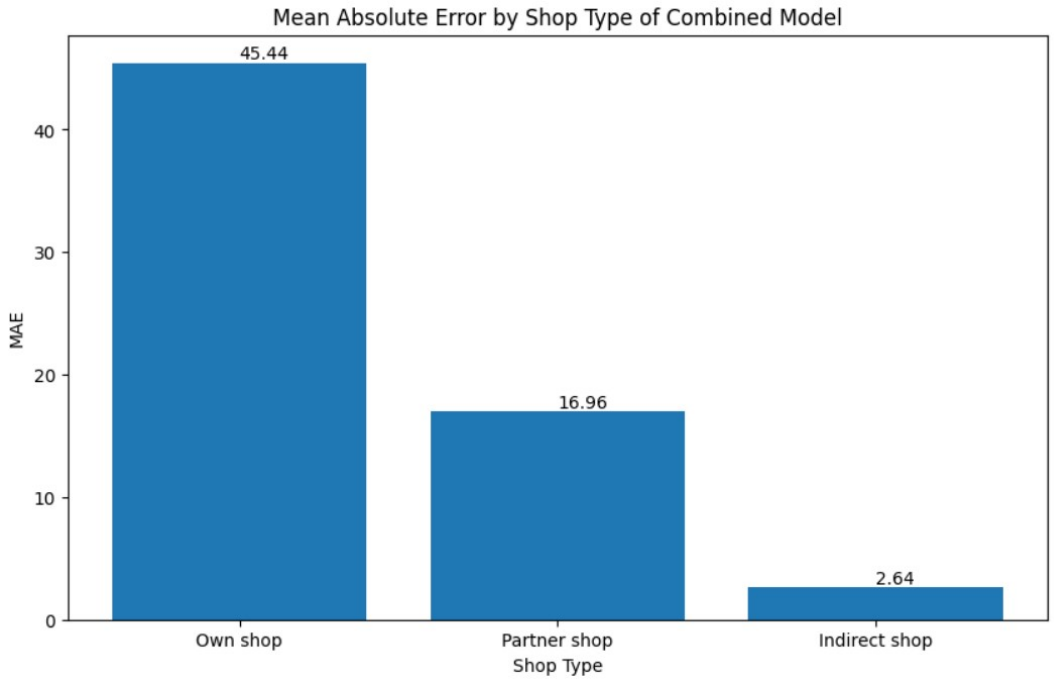


Figure 4.33: Hybrid Model MAE Plot of Vodafone Shop Types

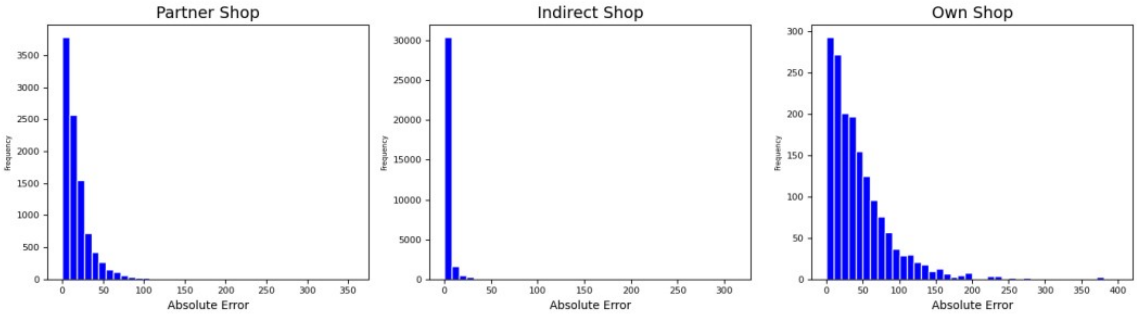


Figure 4.34: Hybrid Model Absolute Error Distribution Shop Types Vodafone

However, when the same type of hybrid model was applied to precipitation data, it showed no difference compared to the base models. The distribution of absolute error (refer to the Figure 4.35) displayed a similar pattern, with many values clustered around 0, and this was consistent across the absolute error distribution for each region (refer to the Figure 4.36).

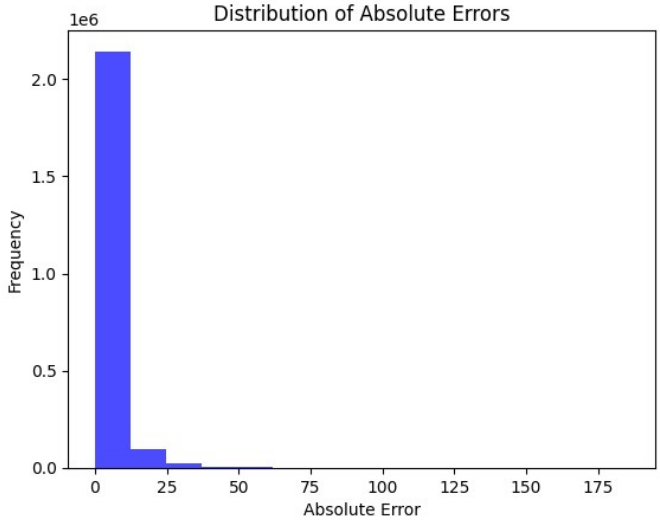


Figure 4.35: Hybrid Model Absolute Error Distribution Precipitation

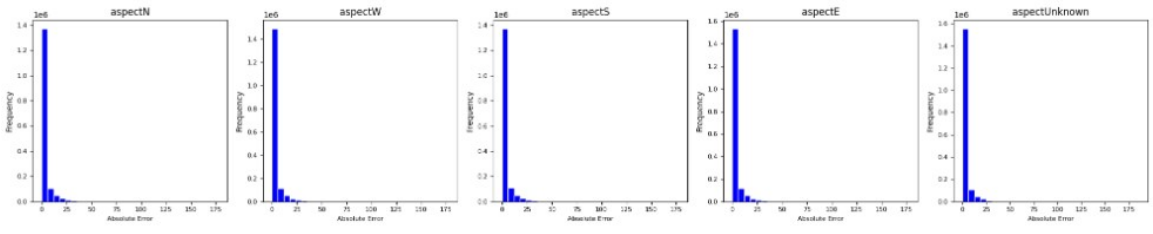


Figure 4.36: Hybrid Model Absolute Error Distribution of Regions Precipitation

The bar plot showing the comparison of the MAE of three different models (see Figure 4.37 4.38) applied to the Vodafone and precipitation datasets for static and dynamic features. It is observed that for the Vodafone dataset, the static model had a lower MAE than the dynamic model. The combined model, which is a hybrid that combined the outputs of both static and dynamic models, achieved the lowest MAE of the three. However, for the precipitation dataset,

the static model had a slightly higher MAE than the dynamic model. Contrary to expectations, the combined model, as shown in the Figure, did not outperform the individual models, it had an MAE that was equal to that of the dynamic model, showing that there was no improvement from integrating static and dynamic features for this particular precipitation dataset.

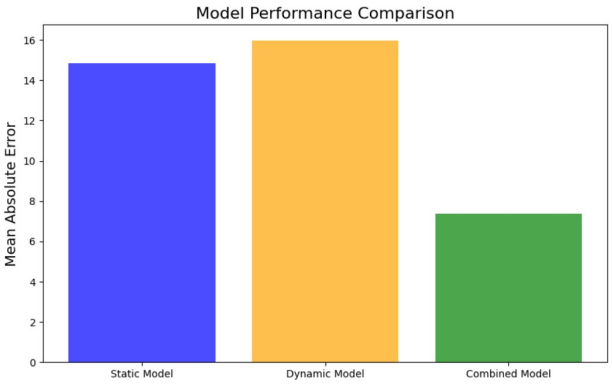


Figure 4.37: MAE Plot of Vodafone Models Comparison

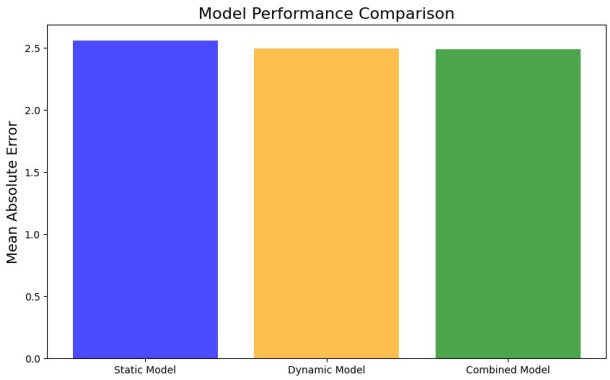


Figure 4.38: MAE Plot of Precipitation Models Comparison

The absolute error distributions for various Vodafone shop types, as shown in Figure 4.39 matched with better MAE values. The hybrid model had a lower frequency of larger absolute errors in partner_shops and own_shops, which was better than the static model. The indirect_shops had the same frequency of errors for both the hybrid and static models. Contrarily, the set of histograms (refer Figure 4.40) for the precipitation dataset indicates that the hybrid model consistently maintained a low frequency of higher absolute errors across all regions. In the region named as 'unknown,' the frequency of errors was slightly higher for both models compared to other regions. Nevertheless, the hybrid models performance matched that of the static model, indicating no reduction in the frequency of larger errors for this region with the hybrid model when compared to the static model.

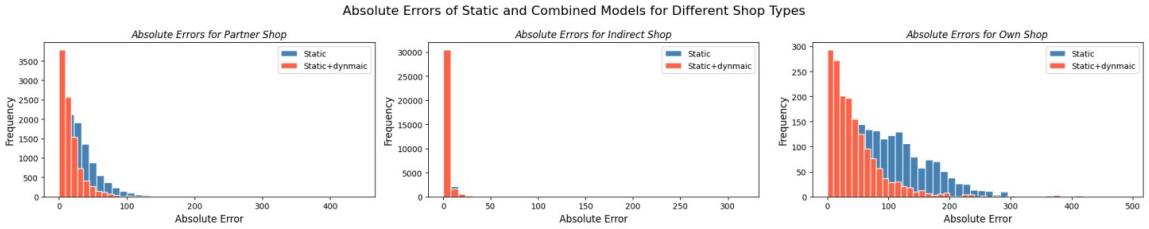


Figure 4.39: Absolute Error Distribution of Two Models Vodafone

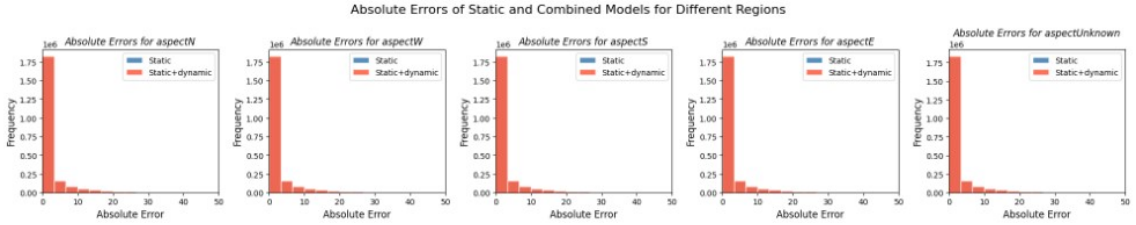


Figure 4.40: Absolute Error Distribution of Two Models Precipitation

In the case of the Vodafone dataset, the hybrid model's MAE was **7.3**, while the static model's MAE was **14.8**, and the dynamic model's MAE was **15.9**. This shows that the combined model improved MAE, showing that integrating static and dynamic features was valuable here. The large number of static features (102) compared to dynamic features (7) implied that the static features were more influential and provided more information. Their proper integration with dynamic features enhanced the model's performance.

Conversely, in the case of the precipitation dataset, the hybrid model's MAE was **2.4**, the static model's MAE was **2.5**, and the dynamic model's MAE was **2.4**, which was slightly better than the static model. This dataset had a more balanced distribution between static (32) and dynamic (20) features. The similar MAE values for both the static and dynamic models showed that both types of features were equally important in this dataset, and the features independently provided most of the necessary information.

Thus, the results from both datasets explains that the predictiveness of the hybrid model varies between datasets. This shows the importance of dataset-specific characteristics in model architecture. The fact that the hybrid model improved performance in a dataset with a higher number of static features and was less effective in a dataset with a more balanced feature distribution shows the necessity of adopting different strategies for handling various types of features depending on their distinct roles and influences within different datasets.

Dataset	Model Name	Architecture	Input Size	Input Layer Type	Batch Size	Epochs	MAE	Model Configuration
Vodafone	Base Model 1	Keras Sequential	(109,)	dense	32	50	12.58	EarlyStopping
	Base Model 2	LSTM	(3 , 109)	lstm, dense	32	40	12.58	window_siye = 3, EarlyStopping
	Autoencoder model	autoencoder	dynamic features size = (7,)	dense	32	20	1.25	encoding_dim = 8
		Keras Sequential	(109,)	dense	32	40	12.35	Reconstructed dynamic features and static features as input
	Hybrid Model	Keras Sequential (static features model	(102,)	dense	32	_	14.82	for static features
		LSTM (dynamic features)	(1, 7)	lstm	32	_	15.95	for dynamic features
		combined model	(109,)	dense	32	50	**7.37**	combined output of static and dynamic model as input
Precipitation	Base model 1	Keras Sequential	(52,)	dense	32	20	2.49	EarlyStopping
	Base model 2	LSTM	(7,52)	lstm, dense	32	20	2.49	Data generators used with batch size = 32, window_size = 7
	Encoder model	Encoder	dynamic features size = (20,)	dense	32	10	5.23	encoding_dim = 8
		Keras Sequential	(40,)	dense	8	20	2.49	Compressed version of dynamic features with static featutres as input
	Autoencoder model	autoencoder	dynamic features size = (20,)	dense	32	10	5.13	encoding_dim = 8
		Keras Sequential	(52,)	dense	8	20	2.49	Reconstructed dynamic features and static features as input
	Hybrid Model	keras sequential (static features model	(32,)	dense	32	_	2.53	for static features
		LSTM (dynamic features)	(1, 20)	lstm	32	_	2.49	for dynamic features
		combined model	(52,)	dense	32	20	**2.48**	combined output of static and dynamic model as input

Figure 4.41: Model Description Table

5. Summary and Outlook

In the last part of this thesis, this chapter provides a summary of the results. It will also give an outlook on future work for integrating static and dynamic data.

5.1 Answering the Research Questions

In this master thesis the different results on two distinct datasets highlighted the importance of feature selection and the proper handling of these features.

The results indicate that the effectiveness of an autoencoder in preprocessing dynamic data for sequential models depends on the number of dynamic features and the encoding dimension of the autoencoder. For instance, in the Vodafone dataset, expanding a smaller number of dynamic features (7) into an 8 dimensional space improved the performance of the Keras sequential model. In opposite, the precipitation dataset shows that when compressing 20 dynamic features into an 8 dimensional space did not improve the model's performance, the reduced encoding dimension is too limited to capture useful information. This shows that careful and accurate handling of the encoding capacity is crucial. The best practice should involve an accurate approach to dimensionality reduction, considering the specific characteristics of the dataset.

The performance of sequential models varies with the proportion of static and dynamic features. In the Vodafone dataset, with a larger number of static features compared to dynamic features, combining outputs of both types improved the model's performance. However, in the precipitation dataset, where the static and dynamic features were more balanced, so the hybrid model did not show any improvement. This suggests that the number and balance of static and dynamic features play a critical role in the better performance of hybrid models.

The results show the necessity of using dataset-specific strategies for integrating static and dynamic data. The different impact of features in the two datasets suggest that the performance of the sequential models is based on the dataset's unique characteristics, including the number and type of features. A one-size-fits-all approach may not be effective, and models should be implemented depending on the specific feature types of each dataset.

The research shows that the combination and preprocessing of static and dynamic data in sequential models should be approached with strategies specifically adapted to the dataset's characteristics. Strategically integrating these features can increase the performance of sequential models.

5.2 Future Work

In this work, some strategies for combining static and dynamic features and feed into sequential models are explained. In the future, this could be improved by the following approaches and considerations:

- **Finding a Suitable Size for Feature Encoding.**

  Future work could try to automatically find the suitable target size for transforming data into an encoded form. This means examining various sizes for the encoding dimension and finding out which size can be used for encoding purposes so that the machine model can learn and understand the dataset well enough to give better predictions. This would involve trying different sizes on various types of data.

- **Improving How Different Types of Data Work Together in Models.**

  As the balance between static and dynamic data affects the model's results, future work could look into developing better ways for these two types of data to work together. For instance, creating new types of models that can determine when to give more importance to static or dynamic features depending on the requirements for prediction.

- **Exploring Static and Dynamic Data Combining Methods on Various Types of Data.**

  In this thesis, the methods were specifically implemented for Vodafone, which was sales data, and for precipitation, which was a weather dataset. Future work could involve testing the feature integration techniques on additional kinds of sequential data, such as sensor data, log data, survey data, and network data. This research would evaluate whether these approaches could improve model predictions for such diverse datasets as well.

Bibliography

[1] Christoph Minixhofer. *Predict Droughts using Weather and Soil Data*. Last Updated - 03/03/2021. URL: https://www.kaggle.com/datasets/cdminix/us-drought-meteorological-data.

[2] Remi M. *ActiveState - What is Keras Model and how to use it to make predictions*. [Last Updated: 25-May-2021]. URL: https://www.activestate.com/resources/quick-reads/what-is-a-keras-model/.

[3] *Keras Models API*. [Last Updated: 2023/06/25]. URL: https://keras.io/api/models/#:~:text=The%20Sequential%20model%2C%20which%20is,that%20supports%20arbitrary%20model%20architectures/.

[4] Yugesh Verma. *A Complete Understanding of Dense Layers in Neural Networks*. [Last Updated: 19-September-2021]. URL: https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks/.

[5] Robin M. Schmidt. "Recurrent Neural Networks (RNNs): A gentle Introduction and Overview". In: *CoRR* abs/1912.05911 (2019). arXiv: 1912.05911. URL: https://arxiv.org/abs/1912.05911.

[6] Larry R Medsker and LC Jain. "Recurrent neural networks". In: *Design and Applications* 5.64-67 (2001), p. 2. URL: https://cdn.preterhuman.net/texts/science_and_technology/artificial_intelligence/Recurrent%20Neural%20Networks%20Design%20And%20Applications%20-%20L.R.%20Medsker.pdf.

[7] Shudong Yang, Xueying Yu, and Ying Zhou. "LSTM and GRU Neural Network Performance Comparison Study: Taking Yelp Review Dataset as an Example". In: *2020 International Workshop on Electronic Communication and Artificial Intelligence (IWECAI)*. 2020, pp. 98–101. DOI: 10.1109/IWECAI50956.2020.00027.

[8] Christopher Olah. *Understanding LSTM Networks*. [Posted on: 27-August-2015]. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[9] AlindGupta. *Long Short Term Memory Networks Explanation*. [Last Updated: 2-January-2023]. URL: https://www.geeksforgeeks.org/long-short-term-memory-networks-explanation/.

[10] *Machine Learning - Activation functions in neural networks*. [Last Updated: 06-July-2023]. URL: https://www.superannotate.com/blog/activation-functions-in-neural-networks/.

[11] Jason Brownlee. *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. [Last Updated: 20-August-2020]. URL: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

[12] Bharath Krishnamurthy. *An Introduction to the ReLU Activation Function*. [Published on: 28-October-2022]. URL: https://builtin.com/machine-learning/relu-activation-function.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[14] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014). URL: https://arxiv.org/pdf/1412.6980.pdf.

[15] Jan Seltitz. *Deep Learning mit künstlichen neuronalen Netzen vom Typ Long Short-Term Memory für die Zeitreihenprognose*. [University of Applied Sciences Schmalkalden, Germany (Year 2021)].

[16] Stephanie Glen. *Absolute Error And Mean Absolute Error (MAE)*. [Accessed: 08-August-2023]. URL: https://www.statisticshowto.com/absolute-error/.

[17] *What is Mean ABsolute Error*. [Accessed: 08-August-2023]. URL: https://c3.ai/glossary/data-science/mean-absolute-error/.

[18] *POWER Project - Global Climate and Weather Modeling*. Accessed: 2023-10-19. URL: https://power.larc.nasa.gov/.

[19] G. Fischer et al. *Harmonized World Soil Database*. Global Agro-ecological Zones Assessment for Agriculture (GAEZ 2008). URL: https://www.fao.org/soils-portal/data-hub/soil-maps-and-databases/harmonized-world-soil-database-v12/en.

[20] Serdar Yegulalp. *What is TensorFlow? The machine learning library explained*. [Updated on: 3-June-2022]. URL: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html.

[21] *Introduction to TensorFlow*. [Last Updated on: 15-September-2023]. URL: https://www.geeksforgeeks.org/introduction-to-tensorflow/.

[22] Nick McCullum. *The Ultimate Guide to the Pandas Library for Data Science in Python*. [Last Updated on: 08-July-2020]. URL: https://www.freecodecamp.org/news/the-ultimate-guide-to-the-pandas-library-for-data-science-in-python/.

[23] *Pandas : the Python library dedicated to Data Science*. [Last Updated on: 15-February-2023]. URL: https://datascientest.com/en/pandas-the-python-library.

[24] *NumPy : the most used Python library in Data Science*. [Last Updated on: 13-March-2023]. URL: https://datascientest.com/en/numpy-the-python-library-in-data-science.

[25] Vinothkumar A.K. et al. *Introduction to NumPy*. [Last Updated on: 14-August-2023]. URL: https://www.geeksforgeeks.org/introduction-to-numpy.

[26] *Was ist matplotlib?* [Last Updated on: 06-February-2018]. URL: https://machine-learning-blog.de/2018/02/06/was-ist-matplotlib/.

[27] Remi M. [Last Updated On: 13-January-2021]. URL: https://www.activestate.com/resources/quick-reads/what-is-matplotlib-in-python-how-to-use-it-for-plotting/.

[28] *Python Course*. [Accessed On: 22-October-2023]. URL: https://www.python-kurs.eu/matplotlib.php.