Grazing into the Future

Arthur, Guillaume (n11371200)
IFN704 Assessment 3

Executive Summary

This report presents an analysis of pasture yield prediction using statistical and deep learning models, specifically comparing the performance of a Vector Autoregression (VAR) model with a Recurrent Neural Network (RNN) featuring Long Short-Term Memory (LSTM) layers. The primary objective was to predict TSDM (Total Standing Dry Matter) at various time horizons, utilizing multivariate climate data.

The results indicate that the VAR model consistently outperformed the LSTM across all prediction horizons, aligning with findings from the literature on similar datasets. The RNN model's increased complexity did not yield significant benefits, likely due to the limited dataset size, which makes it less suitable for deep learning applications. Among the deep learning architectures tested, including convolutional and autoregressive layers, the best results were achieved with an RNN.

Although the VAR model used aggregated data across paddocks, further improvements could be made by training individual models for each paddock to capture localized variations. The influence of specific climate variables was not fully explored, but the multivariate approach was shown to improve predictions over univariate models. Future research could involve using transformer-based architectures and conducting variable importance analysis to better understand the contributions of individual climate factors to the model's predictions.

In summary, while the VAR model demonstrated superior performance in this study, further refinement and testing of advanced neural network architectures on larger datasets could potentially unlock additional insights and predictive power for pasture yield forecasting.

Contents

Introduction

Barbecue is an integral part of Australian culture. As in most Western cultures, meat is an important component of daily meal planning. However, from the cradle to the plate, the growth of animals needs to be carefully planned to optimize quantity and minimize cost. Livestock refers to animals raised for human consumption. Among these, cattle are members of the bovinae family, commonly referred to as cows and bulls, and was an industry worth 22 billion dollars in 2022-2023 [1].

The most common way to feed livestock is by grazing in pastures. Although it seems simple at first glance, this approach has one major flaw: overgrazing. When repeated over multiple years, heavy grazing leads to a decline in the vigor and production of biodiversity.

To counteract this, rotational or seasonal grazing is implemented. The idea is simple: rotate the pasture being grazed so that some areas can grow and rejuvenate while others are being consumed.[2]

Before consumption, livestock need to eat, and this requirement varies based on the species, sex, and status of the animal. For example, a pregnant cow requires 0.13 megajoules per kilogram (MJ/kg) for growth, 0.27 MJ/kg for a lactating cow, and 0.2 MJ/kg for a bull. Given that animal sizes range from 350 kg for a cow to 800 kg for a bull, it's essential to know the amount of energy available in pastures at any given time [3].

Pasture yield prediction is the process of forecasting the amount of feed that will be produced in a given area. Various factors are used to predict pasture yield:

Soil: The type of soil affects crop yield through its fertility.

Crop variety: Different varieties have different growth curves and are more resistant to environmental factors.

Even in uniform pastures, feed availability can vary widely. The high uncertainty associated with the temporal and spatial availability of feed makes accurately estimating pasture yield an important tool.

Time Serie Forecasting

Statistical Models

Time series are ordered sequences of measurements, repeated at fixed sampling intervals. Forecasting involves the use of time series to predict future data based on historical data. A key assumption of traditional modelling techniques is the independence of observations; however, it is not the case with time series, where points from the same source are correlated. The first attempt at time series analysis was the autoregression model, which uses past values as predictors in a linear regression. Additional

contributions were made with developments in the 1980s. Recent advancements in machine learning, particularly with neural networks, have further enhanced our understanding of time series analysis.

The first model, built in 1927, was the autoregression (AR) model, in which the variable of interest is modeled via a linear combination of its past values. It is a regression of the variable against itself, with no additional predictors included.

Moving average (MA) models, on the other hand, use the moving average of past values instead of the values themselves.

Both components were combined in the ARIMA model, with "I" standing for "integrated," meaning the difference between previous values is taken to help remove trends and seasonality from the data. ARIMA has become one of the most common approaches to time series forecasting, as it can explain both long-term trends and sudden changes in the data. However, it is best suited for short-term forecasts, as it will eventually predict the mean of the data.

ARIMA is limited to univariate data, with no external inputs. Variations have been developed to increase flexibility, such as VARIMA (Vector ARIMA), introduced in 1981, which can handle multivariate data, and VARIMAX, which allows for exogenous data.

These models were groundbreaking at the time, but certain limitations affect their efficiency: stationary time series, where the mean and standard deviation do not change, are required but rare in practice. Additionally, not all relationships are linear. More adaptive models were needed to tackle these challenges.

Although still widely used, recent advancements in neural networks have opened new possibilities in time series forecasting. Previously, neural networks struggled with time series data due to the vanishing gradient problem; however, modern techniques have overcome this barrier, expanding their applicability in time series forecasting [4].

Deep-Learning Models

Deep learning initially struggled to make useful predictions. However, the advent of natural language processing enabled a substantial boost in performance. First, variations of CNN models were adapted, employing causal convolutions to prevent information leakage. This approach helps prevent gradient vanishing and exploding by adding backpropagation paths in the temporal direction. To address the challenge of modeling long-term dependencies, models incorporating attention mechanisms were developed to capture both local and global temporal patterns. The DSANet model, for example, uses

CNNs to capture complex mixed patterns and leverages attention to learn correlations among multiple time series [5].

Recurrent neural networks (RNNs) were a positive addition to the field. Instead of a purely feed-forward pass, they incorporate recurrent connections within the network, allowing information to persist over time. RNNs have been combined with LSTMs to encode temporal data and select the most important time steps.

More recently, transformer-based models have started to emerge in the domain of time series forecasting. Relying solely on attention mechanisms, they can, in theory, capture correlations within sequences through self-attention. This approach is potentially more interpretable and relies entirely on attention to characterize the global dependencies between inputs and outputs. A major hurdle with using such models is that the computational complexity increases exponentially with the length of the input sequence. The Informer model introduced a workaround and demonstrated ways to address these issues by creating sparse attention with key and important queries, focusing on modeling the keys with high attention scores. This, combined with the LogSparse mask technique, significantly reduces computational complexity [5].

The rapid development of new approaches and techniques is not integrated equally across all domains. Domain generalization aims to design models that can effectively generalize to unseen target domain by learning from an observed domain. The challenge is particularly aggravated in the study of time series, due to varying data distribution and temporal dependencies. It is therefore essential to test these advancements within each domain to assess how they can improve results. [6]

Yield Prediction in Agriculture

In agriculture, crop yield prediction is one of the most important metrics. It is a challenging task, as it depends on various meteorological factors, such as climate (maximum and minimum temperature, rainfall, precipitation, humidity, etc.), soil, location, seed variety, and more. Meteorological conditions are among the most crucial factors, as climate varies throughout the year for a given location. Previous studies have treated meteorological data as static. Among all variables, weather parameters exhibit the most variability during a season, and the optimal conditions vary for each crop variety.

Efforts focus on predicting the quantity of available materials during the peak season, as crops used for grazing do not show a constant growth pattern. Most models have been trained with meteorological data from the entire year, which may reduce accuracy. Crop yields are reportedly most affected by meteorological conditions during the growing season. Van Klompenburg et al. concluded

that neural networks, namely CNN, LSTM, and DNN, are widely applied for yield prediction [7]. A review by Dhanari et al. on crop yield prediction found that a hybrid RNN-LSTM network outperformed all other neural networks. However, there appears to be limited review in this field [5].

For forage production, the most commonly used weather features are air temperature, precipitation, solar radiation, relative humidity, and wind speed. Many machine learning techniques have been used to predict forage yield, such as gradient boosting, K-nearest neighbour, linear and multiple linear regression, and artificial neural networks, with comparable results across many models. Most current techniques now involve using remote sensing data due to its availability [8].

Figure 1: Model Performances

Objective

The aim of this study is to compare two methods to forecast TSDM in pastures: a statistical mode, the vectorial ARIMA model, and a deep leaning, a combination of a LSTM backbone with a recurrent neural network. Performance will be assessed on short term (1 month), medium (6 months) and long term(12 months).

Method

Dataset Description

TSDM data inferred from remote sensing imagery were available for paddocks in the Natural Resource Management (NRM) region 1010 in Australia. Data were calculated between 2017 and 2023, from remote sensing imagery from the sentinel-2 satellite. The Central Tablelands region is located in central New South Wales and covers an area of approximately 31,365 km². Paddock data included

information on size, usage, crop type, and creation date. TSDM data were smoothed using a 30-day median to reduce noise from remote sensing. These data were processed every 15 days.

Figure 2: Central Tableland Regional Area [9]

Climate data were available for each paddock. They were collected from SILO, which uses Bureau of Meteorology data and interpolation techniques to create spatial grids. The attributes included were rainfall (mm), temperature (°C), humidity (%), evaporation (mm), and radiation (MJ/m²). Daily measurements from 2017 to 2022 were available.

Of all the paddocks, those reserved for grazing and containing natural grass were selected. They constituted the vast majority of all paddocks, which helped control for the different growth rates of various species. From these paddocks, only those with TSDM data were retained. To address the high number of missing values, the data were first grouped by paddock, month, and year. For rainfall, the monthly sum was calculated; for temperature, the monthly minimum and maximum were calculated; similarly, for RH_MAX and RH_MIN, as well as the mean evaporation and radiation.

The data were then grouped by month and year, and paddocks with missing TSDM time data were removed, totaling only 32. In total, 1,208 paddocks were selected, with data from 2017 to 2022 included.

The following represents the summary statistics of the selected variables. The coefficient of variation is the ratio of the standard deviation to the mean, representing the relative dispersion. Skewness measures the symmetry of the data distribution: a skew of 0 indicates a symmetric distribution, a positive skew indicates more high values, and a negative skew indicates more low values. Kurtosis indicates the "tailedness" of the distribution, with a value of 3 indicating a normal distribution, values greater than 3 indicating more extreme values, and values less than 3 indicating fewer extreme values.

We observe a large variation in TSDM, with a mean of 1740.44 and a standard deviation of 673.8.

Table 1: Summary Statistics

	Min	Max	Median	Mean	StDev	CoefVar	Skew	Kurtosis
TSDM	44.85	7160.14	1752.63	1740.44	673.8	38.71	0.3	3.18
RAIN	0	257.6	22.4	30.06	30	99.78	1.75	7.32
MAX_TEMP	9.3	46.2	27	27.33	7.97	29.17	0.2	2.11
MIN_TEMP	-7.7	19.9	2	2.86	5.51	192.71	0.43	2.27
RH_TMAX	18	100	70.7	68.82	14.63	21.25	-0.58	3.01
RH_TMIN	23.3	100	82	78.35	17.6	22.47	-0.63	2.5
EVAP	0.33	12.04	3.66	3.95	2.36	59.74	0.7	2.74
RADIATION	5.84	28.95	16.14	16.39	5.66	34.55	0.05	1.78

The following graph shows the monthly variation of Mean TSDM.

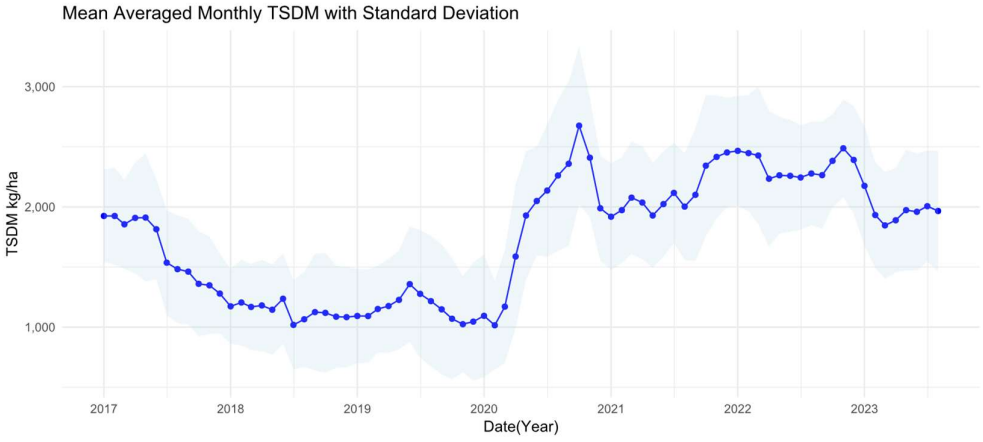


Figure 3: TSDM Mean and Standard Deviation

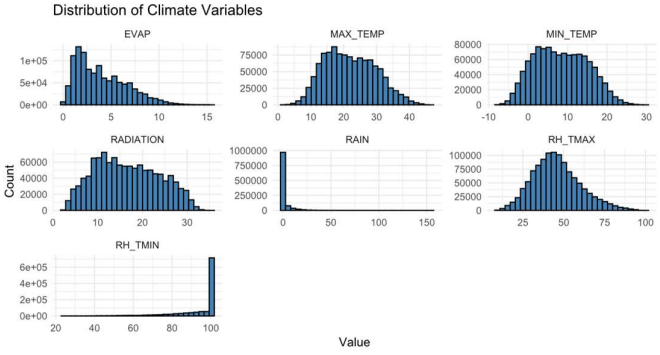


Figure 4: Distribution of Climate Variables

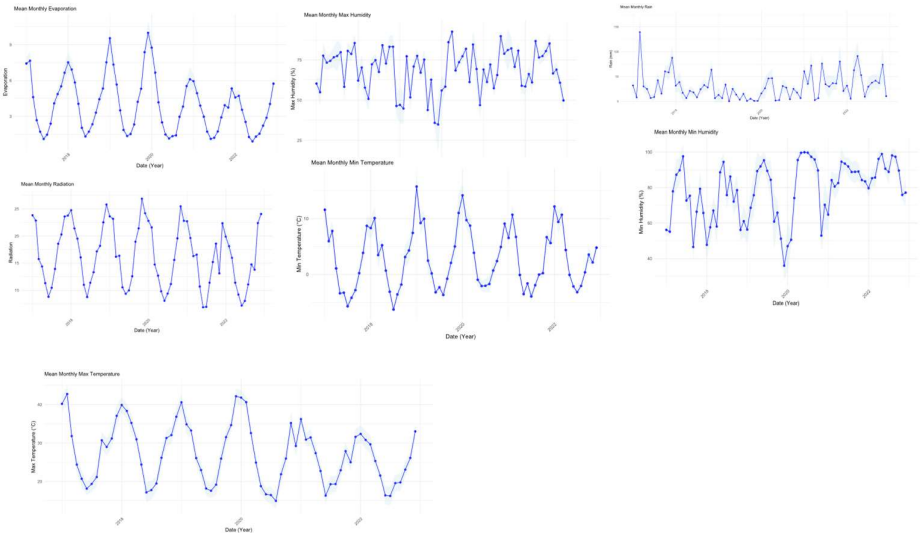


Figure 5: Time Serie Distributions

Methods

The VAR analysis was conducted in R. The Kruskal-Wallis test, from the seastests library, was used to determine whether the mean is identical across each month. The decompose function from the stats library was applied to analyze seasonal components. Stationarity was assessed with the augmented Dickey-Fuller test from the tseries library.

For the deep learning analysis, a template from TensorFlow was used [10], in python, with the TensorFlow library.

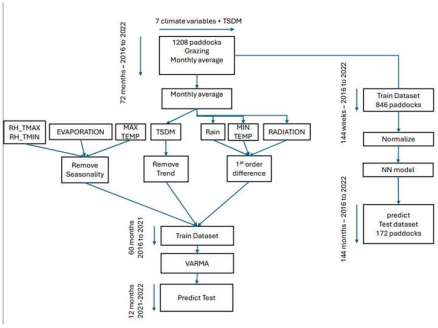


Figure 6: Methods of Analysis

VAR Analysis

A Kruskal-Wallis test was performed on each variable to assess seasonality with a period of 12 months. Seasonality was detected in MAX and MIN temperature, RH_TMAX, RH_TMIN, EVAPORATION, and RADIATION. No seasonality was detected in the total averaged monthly RAIN and TSDM.

The decompose method in R was used to separate the effects of the general trend, seasonality, and the residuals not explained by these two components.

For the variables with detected seasonality, the estimated seasonal components were subtracted from the original time series. For TSDM, a trend was present that did not become stationary after first-order differencing, so a trend removal was applied.

Next, the stationarity of each time series was assessed using the augmented Dickey-Fuller test. Stationarity is a prerequisite for ARIMA modeling. It was found that RAIN, MIN_temp, RADIATION, and TSDM were non-stationary. To address this issue, the difference between two consecutive time points was used for these parameters. As a result, the first month was removed from the dataset, reducing it to 71 months.

Table 2: Augmented Dickey-Fuller test Results

Variable	Value	P-value	Stationary
TSMD_detrended	-3.912	0.018	True
RAIN	-2.971	0.18	False
MAX_TEMP	-3.977	0.016	True
MIN_TEMP	-2.467	0.385	False
RH_TMAX	-3.706	0.031	True
RH_TMIN	-3.724	0.029	True
RADIATION	-2.867	0.222	False
EVAP	-5.806	0.01	True

Model Parameter

To assess the performance of the model, predictions were made for TSDM one month, three months (or a season), six months, and one year into the future. Approximately 17% of the data, or about one year's worth, was set aside for training.

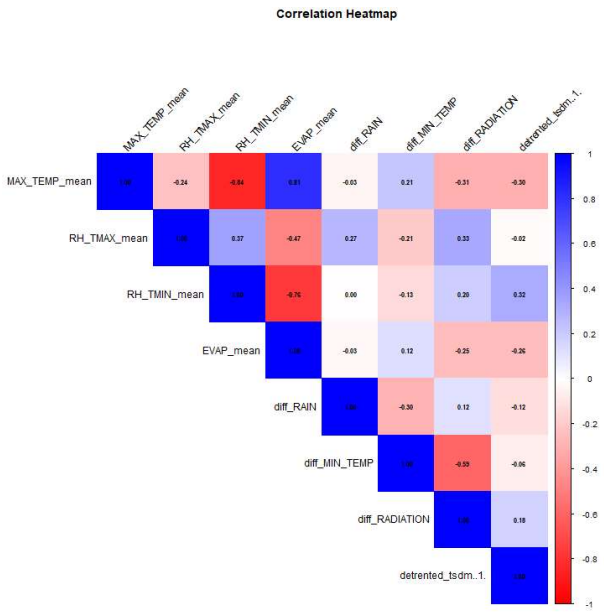


Figure 7: Correlation Matrix

To avoid collinearity, the variable RH_TMIN was removed from the analysis due to its high correlation with MAX_TEMP and EVAP, while having a low correlation with the TSDM mean. The input length of 6 was determined by fitting different models with different lags, and selecting the one with the lowest BCI, an information criterion that penalizes model complexity.

Deep Learning Analysis

To analyze and compare the performance of the deep learning models, each paddock in the dataset was assigned to either the training set (967 paddocks) or the validation set (241 paddocks). Sequences of inputs consisted of 24 consecutive entries for each paddock as X, with predicted values as the following time series y. TheAll climate parameters and TSDM were included as input. The mean and standard deviation of the training dataset were used to normalize both the test and training sets. Then, sequences were shuffled. The chosen evaluation metrics were root mean squared error (RMSE) and mean absolute error (MAE), computed on the testing set. Values were scaled back by multiplying the metrics by the standard deviation of the training dataset.

$$\text{RMSE} = \sqrt{\frac{\sum_{i=0}^{n}(f(x_i)-y_i)^2}{n}}$$ were f(x) is the prediction function and y the ground truth.

$$\text{MAE} = \frac{\sum_{i=0}^{n}|f(x_i)-y_i|}{n}$$

Different input lengths were tested, but no significant improvement was observed for sequences longer than 12 points, or 24 weeks. To find an effective model, various architectures were tested. In all cases, the predicted output was a sequence of size n.

Architectures

The first model tested predicted the output one point at a time. Various architectures were evaluated: a multi-step dense model, which uses a selected window size as input, flattens the array, and follows

it with dense layers to predict the output. Next, a convolutional neural network (CNN) was tested, using a Conv1D layer with 32 filters and a kernel size equal to the desired input sequence length. A recurrent neural network (RNN) with a Long Short-Term Memory (LSTM) layer was also tested.

For the model that performed the best, the RNN, two types of predictions were tested: predicting the entire series in a single shot or an autoregressive prediction, where the model makes a single-step prediction, and the output is then fed back as input. The selected architecture consisted of an LSTM cell with 3 units, wrapped into a recurrent neural network layer, followed by a dense layer. Two training modes were compared: predicting only the TSDM value or predicting all parameters to see if this improved results.

The patience was set to 2 epochs without improvement in the validation loss. The loss function was mean squared error, and the optimizer was Adam. On the figure is presented the mode if prediction using a convolution layer(left), a LSTM (middle) and autoregressive LSTM (right). Code is presented in appendix.

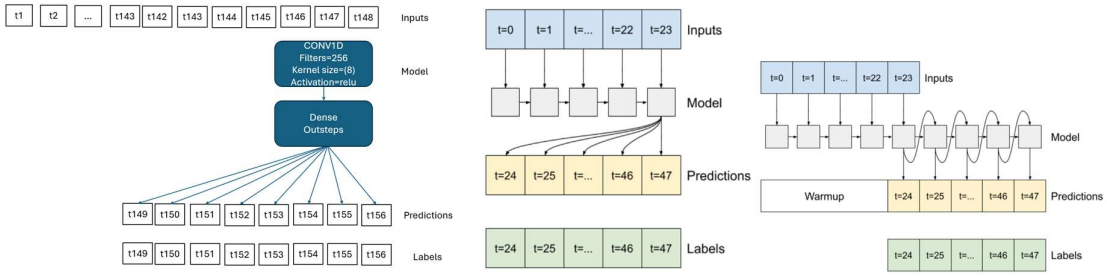
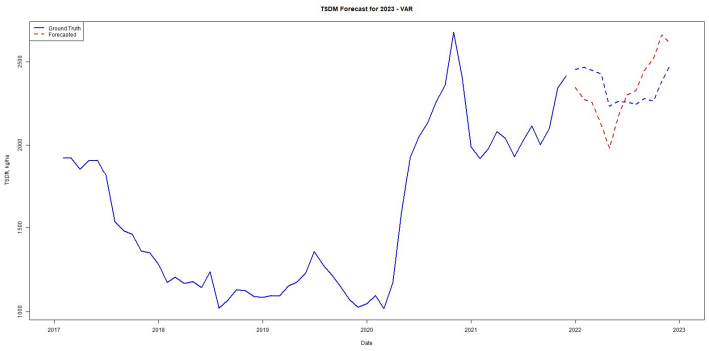


Figure 8: Schema Showing the Convolutional Network Model (left), recurrent neural network (RNN) middle, and auto-regressive Neural Network (AR-RNN)

Results

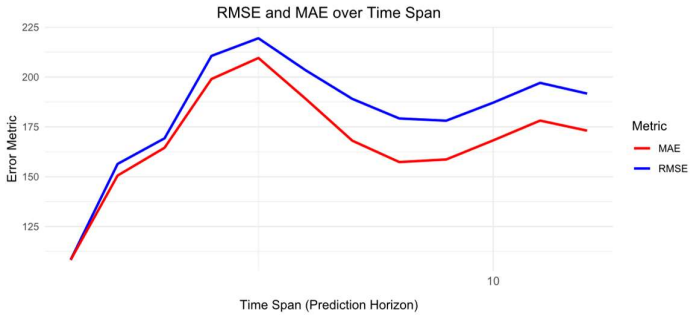
For the VAR analysis, results are presented below.



Months predicted	RMSE	MAE
1	108.2764	108.2764
2	156.3928	150.5664
3	169.2441	164.5071
4	210.6044	198.997
5	219.4485	209.545
6	203.4798	189.1848
7	189.03	168.0523
8	179.2048	157.3449
9	178.0951	158.6348

Table 3: Results for the VAR Analysis

10	187.1399	168.2179
11	197.0638	178.1454
12	191.7053	173.102



An RMSE ranging from 108.27 to 219.45 kg/ha was observed, comparing favourably with the observed variation in TSDM globally, which has a standard deviation of 675. The model is more accurate for shorter horizons; however, its accuracy stabilizes after 4 months.

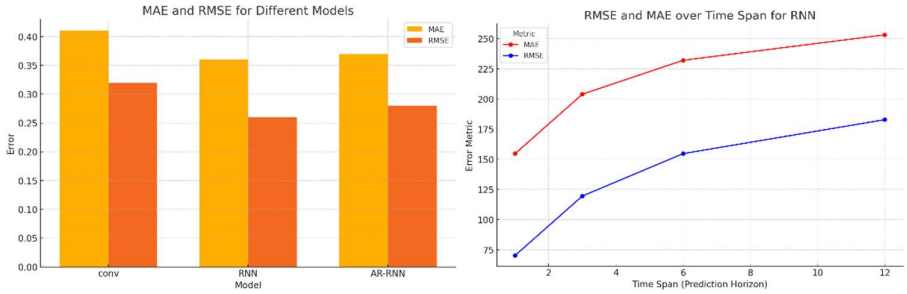


Figure 9: Model Comparison: convolution(conv), recurrent neural network(RNN), Auto-regressive RNN(AR-RNN)

The recurrent neural network performed the best at a 12-month horizon, though by a small margin (Table 3, Figure 6). The model performed better at a 1-month horizon and performed similarly to the VAR model at a 12-month horizon (RMSE of 182 vs. 191). However, the higher MAE for the RNN indicates greater variation in the results, with more predictions farther from the actual values.

Table 4: Model comparison Results – TSDM (kg/ha)

Models – 12 Month Horizon	**MAE**	**RMSE**
Conv	288.34	225.05
RNN	253.18	182.85
AR-RNN	260.21	196.92

Table 5: Error for RNN at different horizon - TSDM (kg/ha)

Horizon (month)	MAE	RMSE
1	154.7209	70.3277
3	203.9503	119.5571
6	232.0814	154.7209
12	253.1797	182.852

Results show that both models were able to predict future yield. Overall, the VAR model outperformed the neural network (NN) across all time horizons studied. This may be due to the limited amount of available data. The increased complexity of the NN model was not justified, as it took longer to train—13 minutes for 20 epochs—compared to the much faster training time for the VAR model. However, the results cannot be directly compared, as the models were evaluated differently.

For the VAR model, training was done on the averaged values across all paddocks for the first five years of data, with evaluation on the last year of data. In contrast, for the NN model, paddocks were separated into training and testing sets, and values for a paddock were batches together during training and testing.

Quantifying the error is challenging. With a mean TSDM of 1740, an RMSE of 100 corresponds to a 17.4% error, 200 to a 34% error, and 400 to a 43% error. To get a real sense of the error, we should compare the obtained values with the intra-paddock variability, as we worked with the median paddock TSDM. Comparing the different errors to the standard deviation for TSDM at 673.8, it is lower, meaning the models are relatively accurate.

Perspective and Limitations

The result showing that the statistical model outperformed the LSTM has been reported in the literature for different datasets. The size of this dataset likely makes it a poor candidate for deep learning. Different architectures were tested, including convolutional layers, predicting all variables as a secondary task, or predicting only the TSDM, with the best results observed for the recurrent network.

The validity of VAR analysis could have been greater if, similarly to the LSTM, multiple models had been trained for each paddock, and the error measured specifically within each paddock. Other architectures mentioned in the introduction, particularly transformer-based models, might also have yielded better results.

The influence of different climate variables was not evaluated. Results compared to predictions using only the TSDM time series show that using a multivariate model improved accuracy. However, for the neural network, it is harder to investigate the impact of individual variables. To further study the effect of different factors, the values could be shuffled, and the time series broken for each variable

separately. An evaluation of the model in this setting would indicate if removing the time series information for any specific variable affected the predictions.

References

[1] 'Australian Agriculture: Livestock'. Accessed: Nov. 02, 202AD. [Online]. Available: https://www.abs.gov.au/statistics/industry/agriculture/australian-agriculture-livestock/latest-release#:~:text=Key%20statistics,-Local%20value%20of&text=There%20were%2029.9%20million%20cattle,and%202.1%20million%20dairy%20cattle.

[2] G. L. Jan P.Bakker, *Grazing for conservation management in historical perspective*, vol. 1. in Conservation Biology Series, vol. 1.

[3] E. H. Cabezas-Garcia, D. Lowe, and F. Lively, 'Energy Requirements of Beef Cattle: Current Energy Systems and Factors Influencing Energy Requirements for Maintenance.', *Anim. Open Access J. MDPI*, vol. 11, no. 6, Jun. 2021, doi: 10.3390/ani11061642.

[4] Douglas C. Montgomery, Cheryl L. Jennings, Murat Kulahci, *Introduction to Time Series Analysis and Forecasting*, 2nd Edition. 2015.

[5] W. Li and K. L. E. Law, 'Deep Learning Models for Time Series Forecasting: A Review', *IEEE Access*, vol. 12, pp. 92306–92327, 2024, doi: 10.1109/ACCESS.2024.3422528.

[6] S. Deng, O. Sprangers, M. Li, S. Schelter, and M. de Rijke, 'Domain Generalization in Time Series Forecasting', *ACM Trans Knowl Discov Data*, vol. 18, no. 5, Feb. 2024, doi: 10.1145/3643035.

[7] T. van Klompenburg, A. Kassahun, and C. Catal, 'Crop yield prediction using machine learning: A systematic literature review', *Comput. Electron. Agric.*, vol. 177, p. 105709, 2020, doi: https://doi.org/10.1016/j.compag.2020.105709.

[8] S. N. Subhashree *et al.*, 'Tools for Predicting Forage Growth in Rangelands and Economic Analyses—A Systematic Review', *Agriculture*, vol. 13, no. 2, 2023, doi: 10.3390/agriculture13020455.

[9] 'Central Tablelands Local Land Services'. Accessed: Nov. 02, 2024. [Online]. Available: https://www.dpi.nsw.gov.au/climate_applications/ssu-archive/2024/April-2024/monthly-regional-breakdown/central-tablelands-lls-region

[10] 'Time series forecasting | TensorFlow'. [Online]. Available: https://www.tensorflow.org/tutorials/structured_data/time_series

[11] H. Silyn-Roberts, *Writing for science and engineering: papers, presentations, and reports*, Second edition. in Elsevier insights. Amsterdam: Elsevier, 2013.

[12] 'QUT cite|write - QUT cite'. Accessed: Jun. 20, 2016. [Online]. Available: http://www.citewrite.qut.edu.au/cite/

[13] Various authors, *Zotero*. Roy Rosenzweig Center for History and New Media, 2020. [Online]. Available: https://www.zotero.org/

[14] 'Reproducible Research Literate Programming: Explore a working knitr document'. Accessed: May 18, 2020. [Online]. Available: https://datacarpentry.org/rr-literate-programming/03-explore-knitr/

Appendix 1: Supplementary Information

```
class WindowGenerator():

    def __init__(self, input_width, label_width, shift,
                 train_df, val_df, test_df,
                 label_columns=None):
        # Store the raw data.
        self.train_df = train_df
        self.val_df = val_df
        self.test_df = test_df

        # Work out the label column indices.
        self.label_columns = label_columns
        if label_columns is not None:
          self.label_columns_indices = {name: i for i, name in
                                        enumerate(label_columns)}
        self.column_indices = {name: i for i, name in
                               enumerate(train_df.columns)}

        # Work out the window parameters.
        self.input_width = input_width
        self.label_width = label_width
        self.shift = shift

        self.total_window_size = input_width + shift

        self.input_slice = slice(0, input_width)
        self.input_indices = np.arange(self.total_window_size)[self.input_slice]

        self.label_start = self.total_window_size - self.label_width
        self.labels_slice = slice(self.label_start, None)
        self.label_indices = np.arange(self.total_window_size)[self.labels_slice]



    @property
    def train(self):
        return self.make_dataset(self.train_df)

    @property
    def val(self):
        return self.make_dataset(self.val_df)

    @property
    def test(self):
        return self.make_dataset(self.test_df)

    @property
    def example(self):

        result = getattr(self, '_example', None)
        if result is None:
            # No example batch was found, so get one from the `.train` dataset
            result = next(iter(self.train))
            # And cache it for next time
            self._example = result
        return result

def split_window(self, features):
    inputs = features[:, self.input_slice, :]
    labels = features[:, self.labels_slice, :]
    if self.label_columns is not None:
        labels = tf.stack(
            [labels[:, :, self.column_indices[name]] for name in self.label_columns],
            axis=-1)

        # Slicing doesn't preserve static shape information, so set the shapes
        # manually. This way the `tf.data.Datasets` are easier to inspect.
        inputs.set_shape([None, self.input_width, None])
        labels.set_shape([None, self.label_width, None])

    return inputs, labels


def make_dataset(self, data):
    # Ensure data is a numpy array for processing
    data = np.array(data, dtype=np.float32)

    # Initialize an empty list to hold datasets for each paddock
    paddock_datasets = []

    # Group data by paddock column
    grouped_data = data.groupby(self.paddock_column)

    # Iterate over each paddock
    for _, paddock_data in grouped_data:
        # Convert paddock data to numpy array
        paddock_data = np.array(paddock_data.drop(columns=[self.paddock_column]), dtype=np.float32)

        # Create a timeseries dataset for each paddock without shuffling
        ds = tf.keras.utils.timeseries_dataset_from_array(
            data=paddock_data,
            targets=None,
            sequence_length=self.total_window_size,
            sequence_stride=1,
            shuffle=False,  # Ensure sequential order within each paddock
            batch_size=1    # Temporarily set batch size to 1 for each window
        )

        # Use the split_window function to process each window into inputs and labels
        ds = ds.map(self.split_window)

        # Add this paddock's dataset to the list
        paddock_datasets.append(ds)

    # Concatenate all paddock datasets into a single dataset
    paddock_dataset = tf.data.Dataset.concatenate(*paddock_datasets)

    # Set the batch size for the entire concatenated dataset
    return paddock_dataset.batch(32)
```

```
def compile_and_fit(model, window, patience=2):
  early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                    patience=patience,
                                                    mode='min')

  model.compile(loss=tf.keras.losses.MeanSquaredError(),
                optimizer=tf.keras.optimizers.Adam(),
                metrics=[tf.keras.metrics.MeanAbsoluteError(), tf.keras.metrics.MeanSquaredError()])

  history = model.fit(window.train, epochs=MAX_EPOCHS,
                      validation_data=window.val,
                      callbacks=[early_stopping])
  return history
```

```
OUT_STEPS = 24
multi_window = WindowGenerator(input_width=24,
                               label_width=OUT_STEPS,
                               shift=OUT_STEPS,
                               label_columns=['TSDM_MEAN'],
                              train_df = train_df,
                              val_df = val_df,
                              test_df = val_df)


multi_window.plot()
multi_window
```

```
num_features = 1
CONV_WIDTH = 8
multi_conv_model = tf.keras.Sequential([
    # Shape [batch, time, features] => [batch, CONV_WIDTH, features]
    tf.keras.layers.Lambda(lambda x: x[:, -CONV_WIDTH:, :]),
    # Shape => [batch, 1, conv_units]
    tf.keras.layers.Conv1D(256, activation='relu', kernel_size=(CONV_WIDTH)),
    # Shape => [batch, 1,  out_steps*features]
    tf.keras.layers.Dense(OUT_STEPS*num_features,
                          kernel_initializer=tf.initializers.zeros()),
    # Shape => [batch, out_steps, features]
    tf.keras.layers.Reshape([OUT_STEPS, num_features])
])
```

```
multi_lstm_model = tf.keras.Sequential([
    # Shape [batch, time, features] => [batch, lstm_units].
    # Adding more `lstm_units` just overfits more quickly.
    tf.keras.layers.LSTM(32, return_sequences=False),
    # Shape => [batch, out_steps*features].
    tf.keras.layers.Dense(OUT_STEPS*num_features,
                          kernel_initializer=tf.initializers.zeros()),
    # Shape => [batch, out_steps, features].
    tf.keras.layers.Reshape([OUT_STEPS, num_features])
])
```

```
num_features = 1
class FeedBack(tf.keras.Model):
  def __init__(self, units, out_steps):
    super().__init__()
    self.out_steps = out_steps
    self.units = units
    self.lstm_cell = tf.keras.layers.LSTMCell(units)
    # Also wrap the LSTMCell in an RNN to simplify the `warmup` method.
    self.lstm_rnn = tf.keras.layers.RNN(self.lstm_cell, return_state=True)
    self.dense = tf.keras.layers.Dense(num_features)

def warmup(self, inputs):
  # inputs.shape => (batch, time, features)
  # x.shape => (batch, lstm_units)
  x, *state = self.lstm_rnn(inputs)
  # predictions.shape => (batch, features)
  prediction = self.dense(x)
  return prediction, state

FeedBack.warmup = warmup
```

```
feedback_model = FeedBack(units=32, out_steps=OUT_STEPS)
```

```
def call(self, inputs, training=None):
  # Use a TensorArray to capture dynamically unrolled outputs.
  predictions = []
  # Initialize the LSTM state.
  prediction, state = self.warmup(inputs)
  print(se)
  # Insert the first prediction.
  predictions.append(self.out_steps)
  # Run the rest of the prediction steps.
  for n in range(1, self.out_steps):
    # Use the last prediction as input.
    x = prediction
    # Execute one lstm step.
    x, state = self.lstm_cell(x, states=state,
                              training=training)
    # Convert the lstm output to a prediction.
    prediction = self.dense(x)
    # Add the prediction to the output.
    predictions.append(prediction)

  # predictions.shape => (time, batch, features)
  predictions = tf.stack(predictions)
  # predictions.shape => (batch, time, features)
  predictions = tf.transpose(predictions, [1, 0, 2])
  return predictions

FeedBack.call = call
```

```
setwd("C:/Users/lorie/OneDrive/Bureau/agriculture")
# Install and load ggplot2 if not already installed
if (!require(ggplot2)) install.packages("ggplot2")
if (!require(lubridate)) install.packages("lubridate")
if (!require(dplyr)) install.packages("dplyr")
if (!require(zoo)) install.packages("zoo")
if (!require(tidyr)) install.packages("tidyr")
if (!require(scales)) install.packages("scales")
if (!require(moments)) install.packages("moments")
if (!require(forecast)) install.packages("forecast")
if (!require(tseries)) install.packages("tseries")
if (!require(vars)) install.packages("vars")
if (!require(corrplot)) install.packages("corrplot")
library(corrplot)
library(vars)
library(tseries)
library(forecast)
library(moments)
library(tidyr)
library(ggplot2)
library(dplyr)
library(lubridate)
library(zoo)
library(scales)
library(patchwork)
#load the dataframe
tsdm_raw <- read.csv("data/nrm1010_tsdm.csv", header=TRUE, sep=",")
tsdm_raw$OBSERVATION_DATE <- as.Date(tsdm_raw$OBSERVATION_DATE, format="%d/%m/%Y")

#to upper
tsdm_raw$PADDOCK_ID <- toupper(tsdm_raw$PADDOCK_ID)
meta_raw <-  read.csv("data/global_paddocks_new.csv", header=TRUE, sep=",")
#the climate data
climate_2017 <- read.csv("data/climate_2017.csv", header=TRUE, sep=",")
climate_2018 <- read.csv("data/climate_2018.csv", header=TRUE, sep=",")
climate_2019 <- read.csv("data/climate_2019.csv", header=TRUE, sep=",")
climate_2020 <- read.csv("data/climate_2020.csv", header=TRUE, sep=",")
climate_2021 <- read.csv("data/climate_2021.csv", header=TRUE, sep=",")
climate_2022 <- read.csv("data/climate_2022.csv", header=TRUE, sep=",")
#save them into a list
climate_list <- list()
climate_list[[1]]<- climate_2017
climate_list[[2]]<- climate_2018
climate_list[[3]]<- climate_2019
climate_list[[4]]<- climate_2020
climate_list[[5]]<- climate_2021
climate_list[[6]]<- climate_2022
#set the name
names(climate_list) <-
c("climate_2017","climate_2018","climate_2019","climate_2020","climate_2021","climate_2022"
)
#select the unique paddocks with TSDM
unique_paddocks_ids <- unique(tsdm_raw$PADDOCK_ID)
#filter the meta to have only the one present in TSDM
meta_with_tsdm <- meta_raw[meta_raw$PADDOCK_ID%in%unique_paddocks_ids,]
#filter out the paddocks that are not grazing
grazing_paddock_ids <- meta_with_tsdm %>%
  filter(PASTURE_STATE == "Grazing") %>%
  pull(PADDOCK_ID) %>%
  unique()
#filter out paddock without grazing
meta_with_tsdm_grazing <-
meta_with_tsdm[meta_with_tsdm$PADDOCK_ID%in%grazing_paddock_ids,]
```

```
#filter the climate data also
climate_list_tsdm <- lapply(climate_list, function(df){
  df[df$PADDOCK_ID %in% grazing_paddock_ids, ]
})
tsdm_grazing = tsdm_raw[tsdm_raw$PADDOCK_ID%in%grazing_paddock_ids,]

#get the monthly df to deal with the missing values
monthly_mean_df <- tsdm_grazing %>%
  mutate(
    Year = year(OBSERVATION_DATE),
    Month = month(OBSERVATION_DATE, label = TRUE)
  ) %>%
  group_by(PADDOCK_ID, Year, Month) %>%
  summarise(
    monthly_mean = mean(TSDM_MEAN, na.rm = TRUE)
  ) %>%
  arrange(PADDOCK_ID, Year, Month)

monthly_count_df <- tsdm_grazing %>%
  mutate(
    Year = year(OBSERVATION_DATE),
    Month = month(OBSERVATION_DATE, label = TRUE, abbr = TRUE)
  ) %>%
  group_by(PADDOCK_ID, Year, Month) %>%
  summarise(
    observation_count = n()  # Count observations for each month and year per paddock
  ) %>%
  ungroup()
#check if all combiantion have values
total_counts_df <- monthly_count_df %>%
  group_by(PADDOCK_ID) %>%
  summarise(
    total_observations = sum(observation_count)
  ) %>%
  ungroup()
# Step 3: Check which paddocks have a different total count than expected (72)
paddocks_with_missing_counts <- total_counts_df %>%
  filter(total_observations != 162) %>%
  pull(PADDOCK_ID)
#remove them from the final_df
final_included_paddocks <- tsdm_grazing[!tsdm_grazing$PADDOCK_ID %in%
paddocks_with_missing_counts, ]
#get the total, so weighted by landsize_ha
final_included_paddocks <- final_included_paddocks %>%
  inner_join(meta_with_tsdm_grazing  %>% select(PADDOCK_ID, LANDSIZE_HA), by =
"PADDOCK_ID") %>%
  mutate(TOTAL = TSDM_MEAN * LANDSIZE_HA) %>%
  select(-LANDSIZE_HA)  # Remove landsize_area if you don't need it in the final data
frame

#now, create descriptive
paddock_monthly_stats <- final_included_paddocks %>%
  mutate(
    Year = year(OBSERVATION_DATE),
    Month = month(OBSERVATION_DATE, label = TRUE, abbr = TRUE),
    Date = as.Date(paste(Year, month(OBSERVATION_DATE), "01", sep = "-"))  # Use the first
day of each month
  ) %>%
  group_by(Year, Month, Date) %>%
  summarise(
    mean_value = mean(TSDM_MEAN, na.rm = TRUE),
    std_dev = sd(TSDM_MEAN, na.rm = TRUE)
  ) %>%
  ungroup()
#plot the graph of TSDM
```

```
ggplot(paddock_monthly_stats, aes(x = Date, y = mean_value, group = 1)) +
  geom_line(color = "blue") +
  geom_point(color = "blue") +
  geom_ribbon(aes(ymin = mean_value - std_dev, ymax = mean_value + std_dev), fill =
"lightblue", alpha = 0.2) +
  labs(title = "Mean Averaged Monthly TSDM with Standard Deviation",
       x = "Date(Year)",
       y = "TSDM kg/ha") +

  scale_x_date(date_breaks = "1 year", date_labels = "%Y") +
  scale_y_continuous(labels = comma) +
  theme_minimal() +
  theme(theme(
    panel.grid = element_blank(),                 # Remove all grid lines
    axis.text.x = element_text(angle = 45, hjust = 1)
  ))

#aggregate all climate df them
combined_climate <- do.call (rbind, climate_list_tsdm)
#filter df to have the same as for TSDM
combined_climate_filtered <- combined_climate[combined_climate$PADDOCK_ID %in%
final_included_paddocks$PADDOCK_ID, ]
#group by month and year and paddock
combined_climate_filtered$DATE = as.Date(combined_climate_filtered$DATE)
#plot the frquence of each variable
create_histogram <- function(data, variable, title) {
  ggplot(data, aes_string(x = variable)) +
    geom_histogram(bins = 30, fill = "steelblue", color = "black") +
    labs(title = title, x = variable, y = "Frequency") +
    theme_minimal()
}


# Convert the data to a long format
combined_climate_long <- combined_climate_filtered %>%
  pivot_longer(cols = c(RAIN, MAX_TEMP, MIN_TEMP, RH_TMAX, RH_TMIN, EVAP, RADIATION),
               names_to = "Variable", values_to = "Value")

# Plot the count histogram for each variable in one figure
ggplot(combined_climate_long, aes(x = Value)) +
  geom_histogram(bins = 30, fill = "steelblue", color = "black") +
  labs(title = "Distribution of Climate Variables",
       x = "Value",
       y = "Count") +
  facet_wrap(~ Variable, scales = "free") +  # Separate plot for each variable with free
scales
  theme_minimal()

#Step 1: Group by Paddock, Year, and Month to calculate initial aggregations
paddock_monthly_climate <- combined_climate_filtered %>%
  mutate(Year = year(DATE), Month = month(DATE, label = TRUE)) %>%
  group_by(PADDOCK_ID, Year, Month) %>%
  summarise(
    RAIN = sum(RAIN, na.rm = TRUE),
    MAX_TEMP = max(MAX_TEMP, na.rm = TRUE),
    MIN_TEMP = min(MIN_TEMP, na.rm = TRUE),
    RH_TMAX = max(RH_TMAX, na.rm = TRUE),
    RH_TMIN = min(RH_TMIN, na.rm = TRUE),
    EVAP = mean(EVAP, na.rm = TRUE),
    RADIATION = mean(RADIATION, na.rm = TRUE)
  ) %>%
  ungroup()
#create a summary table
variables <- c( "RAIN", "MAX_TEMP", "MIN_TEMP", "RH_TMAX", "RH_TMIN", "EVAP", "RADIATION")
```

```
# Initialize an empty data frame to store the results
climate_stats <- data.frame(
  Variable = character(),
  Min = numeric(),
  Max = numeric(),
  Median = numeric(),
  Mean = numeric(),
  StDev = numeric(),
  CoefVar = numeric(),
  Skew = numeric(),
  Kurtosis = numeric(),
  stringsAsFactors = FALSE
)

# Calculate summary statistics for each variable and store in data frame
for (var in variables) {
  data <- na.omit(paddock_monthly_climate[[var]])  # Remove NA values for the variable

  # Append row with statistics for each variable and format to 2 decimal places
  climate_stats <- rbind(climate_stats, data.frame(
    Variable = var,
    Min = format(round(min(data), 2), nsmall = 2),
    Max = format(round(max(data), 2), nsmall = 2),
    Median = format(round(median(data), 2), nsmall = 2),
    Mean = format(round(mean(data), 2), nsmall = 2),
    StDev = format(round(sd(data), 2), nsmall = 2),
    CoefVar = format(round(ifelse(mean(data) != 0, (sd(data) / mean(data)) * 100, NA), 2),
nsmall = 2),
    Skew = format(round(skewness(data), 2), nsmall = 2),
    Kurtosis = format(round(kurtosis(data), 2), nsmall = 2)
  ))
}
# for tsdm
data_monthly <- na.omit(monthly_mean_df[['monthly_mean']])
climate_stats <- rbind(climate_stats , data.frame(
  Variable = var,
  Min = format(round(min(data_monthly), 2), nsmall = 2),
  Max = format(round(max(data_monthly), 2), nsmall = 2),
  Median = format(round(median(data_monthly), 2), nsmall = 2),
  Mean = format(round(mean(data_monthly), 2), nsmall = 2),
  StDev = format(round(sd(data_monthly), 2), nsmall = 2),
  CoefVar = format(round(ifelse(mean(data_monthly) != 0, (sd(data_monthly) /
mean(data_monthly)) * 100, NA), 2), nsmall = 2),
  Skew = format(round(skewness(data_monthly), 2), nsmall = 2),
  Kurtosis = format(round(kurtosis(data_monthly), 2), nsmall = 2)
))
# Step 2: Group overall (across paddocks), apply aggregations, and create Date column
climate_monthly_stats <- paddock_monthly_climate %>%
  group_by(Year, Month) %>%
  summarise(
    RAIN_mean = mean(RAIN, na.rm = TRUE),
    RAIN_sd = sd(RAIN, na.rm = TRUE),
    MAX_TEMP_mean = mean(MAX_TEMP, na.rm = TRUE),
    MAX_TEMP_sd = sd(MAX_TEMP, na.rm = TRUE),
    MIN_TEMP_mean = mean(MIN_TEMP, na.rm = TRUE),
    MIN_TEMP_sd = sd(MIN_TEMP, na.rm = TRUE),
    RH_TMAX_mean = mean(RH_TMAX, na.rm = TRUE),
    RH_TMAX_sd = sd(RH_TMAX, na.rm = TRUE),
    RH_TMIN_mean = mean(RH_TMIN, na.rm = TRUE),
    RH_TMIN_sd = sd(RH_TMIN, na.rm = TRUE),
    EVAP_mean = mean(EVAP, na.rm = TRUE),
    EVAP_sd = sd(EVAP, na.rm = TRUE),
    RADIATION_mean = mean(RADIATION, na.rm = TRUE),
    RADIATION_sd = sd(RADIATION, na.rm = TRUE)
  ) %>%
```

```
  ungroup()
# Plotting each variable
climate_monthly_stats <- climate_monthly_stats %>%
  mutate(Date = as.Date(ISOdate(Year, match(Month, month.abb), 1)))

plot_variable <- function(data, mean_col, sd_col, title, y_label) {
  ggplot(data, aes(x = Date, y = .data[[mean_col]])) +
    geom_line(color = "blue") +
    geom_point(color = "blue") +
    geom_ribbon(aes(ymin = .data[[mean_col]] - .data[[sd_col]], ymax = .data[[mean_col]] +
.data[[sd_col]]),
                fill = "lightblue", alpha = 0.2) +
    labs(title = title, x = "Date (Year)", y = y_label) +
    theme_minimal(base_size = 10) +  # Adjust base size for better fit
    theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 8),
          plot.title = element_text(size = 10),
          plot.margin = margin(5, 5, 5, 5))  # Reduce margins
}
# Generate each plot
p1 <- plot_variable(climate_monthly_stats, "RAIN_mean", "RAIN_sd", "Mean Monthly Rain",
"Rain (mm)")
p2 <- plot_variable(climate_monthly_stats, "MAX_TEMP_mean", "MAX_TEMP_sd", "Mean Monthly
Max Temperature", "Max Temperature (°C)")
p3 <- plot_variable(climate_monthly_stats, "MIN_TEMP_mean", "MIN_TEMP_sd", "Mean Monthly
Min Temperature", "Min Temperature (°C)")
p4 <- plot_variable(climate_monthly_stats, "RH_TMAX_mean", "RH_TMAX_sd", "Mean Monthly Max
Humidity", "Max Humidity (%)")
p5 <- plot_variable(climate_monthly_stats, "RH_TMIN_mean", "RH_TMIN_sd", "Mean Monthly Min
Humidity", "Min Humidity (%)")
p6 <- plot_variable(climate_monthly_stats, "EVAP_mean", "EVAP_sd", "Mean Monthly
Evaporation", "Evaporation")
p7 <- plot_variable(climate_monthly_stats, "RADIATION_mean", "RADIATION_sd", "Mean Monthly
Radiation", "Radiation")

# Combine all plots into a single figure with 2 columns and padding
combined_plot <- wrap_plots(p1, p2, p3, p4, p5, p6, p7, ncol = 2) +
  plot_annotation(title = "Monthly Climate Variables with Mean and Standard Deviation")

################################## TIME SERIE STUDY ######################
#Test for seasonality
start_year <- 2017
end_year <- 2022
frequency <- 12
tsdmtimeseries <- ts(na.approx(paddock_monthly_stats$mean_value),
                     frequency = frequency,
                     start = c(start_year, 1),
                     end = c(end_year,12))

tsdmtimeseriescomponents <- stl(tsdmtimeseries, s.window = "periodic", t.window = 13)
#remove the trend
trend_component <- tsdmtimeseriescomponents$time.series[, "trend"]
detrented_tsdm = tsdmtimeseries - trend_component


seasonality_test <- kruskal.test(detrented_tsdm ~ cycle(tsdmtimeseries))
print(paste("Seasonality test for", var, ": p-value =", seasonality_test$p.value))
if (seasonality_test$p.value < 0.05) {
  print("Significant seasonality detected.")
} else {
  print("No significant seasonality detected.")
}
#however, there is a distinc trend

seasonal_variables <-c()
for (var in variables) {
```

```
  # Create time series for each variable
  var = paste0(var, "_mean")
  ts_data <- ts(climate_monthly_stats[[var]], frequency = frequency, start = c(start_year,
1))

  # Decompose the time series to check for seasonality
  ts_components <- decompose(ts_data, type = "additive")
  seasonality_test <- kruskal.test(ts_data ~ cycle(ts_data))

  # Print the result of the seasonality test
  print(paste("Seasonality test for", var, ": p-value =", seasonality_test$p.value))
  if (seasonality_test$p.value < 0.05) {
    print("Significant seasonality detected.")
    seasonal_variables <- append(seasonal_variables, var)
  } else {
    print("No significant seasonality detected.")
  }
}

#for each seasonal variable, remove seasonlity
deseasonalized_df <- data.frame(Date = climate_monthly_stats$Date)
for (var in seasonal_variables) {
  # Create the time series for the variable
  ts_data <- ts(climate_monthly_stats[[var]], frequency = frequency, start = c(start_year,
1))

  # Decompose the time series to obtain the seasonal component
  ts_components <- decompose(ts_data, type = "additive")

  # Subtract the seasonal component to get the deseasonalized series
  deseasonalized_series <- ts_data - ts_components$seasonal

  # Add the deseasonalized series to the new data frame
  deseasonalized_df[[var]] <- as.numeric(deseasonalized_series)
}
################## VAR ##############################
#define the different parameter
RAIN <- ts(climate_monthly_stats$RAIN_mean,
           start = c(start_year, 1),
           end = c(end_year, 12),
           frequency = frequency)

EVAPORATION <- ts(climate_monthly_stats$EVAP_mean,
           start = c(start_year, 1),
           end = c(end_year, 12),
           frequency = frequency)

MAX_TEMP <- ts(deseasonalized_df$MAX_TEMP_mean,
               start = c(start_year, 1),
               end = c(end_year, 12),
               frequency = frequency)

MIN_TEMP <- ts(deseasonalized_df$MIN_TEMP_mean,
               start = c(start_year, 1),
               end = c(end_year, 12),
               frequency = frequency)

RH_TMAX_mean <- ts(deseasonalized_df$RH_TMAX_mean,
                   start = c(start_year, 1),
                   end = c(end_year, 12),
                   frequency = frequency)
RH_TMIN_mean <- ts(deseasonalized_df$RH_TMIN_mean,
                   start = c(start_year, 1),
                   frequency = frequency)
EVAP_mean <- ts(deseasonalized_df$EVAP_mean,
```

```
                    start = c(start_year, 1),
                   end = c(end_year, 12),
                    frequency = frequency)
RADIATION <-ts(deseasonalized_df$RADIATION_mean,
               start = c(start_year, 1),
               end = c(end_year, 12),
               frequency = frequency)
TSDM <-detrented_tsdm

ts_list <- list(RAIN = RAIN, MAX_TEMP = MAX_TEMP, MIN_TEMP = MIN_TEMP,
                RH_TMAX_mean = RH_TMAX_mean, RH_TMIN_mean = RH_TMIN_mean,
                RADIATION = RADIATION, TSDM = TSDM, EVAP=EVAPORATION)

#####
#test stationnarity
for (name in names(ts_list)) {
  ts_data <- ts_list[[name]]

  # Perform the ADF test
  adf_result <- adf.test(ts_data)

  # Print the result with a custom message
  if (adf_result$p.value < 0.05) {
    print(paste(name, ": Stationary (p-value =", round(adf_result$p.value, 3),
                ", statistic =", round(adf_result$statistic, 3), ")"))
  } else {
    print(paste(name, ": Non-Stationary (p-value =", round(adf_result$p.value, 3),
                ", statistic =", round(adf_result$statistic, 3), ")"))
  }
}
##compute diff the non-stationnary
diff_ts <-list(RAIN = diff(ts_list[["RAIN"]]),
               MIN_TEMP =  na.approx(diff(ts_list[["MIN_TEMP"]]), rule=2),
               RADIATION = diff(ts_list[["RADIATION"]]))
#retest the stationarity
for (name in names(diff_ts)) {
  ts_data <- diff_ts[[name]]

  # Perform the ADF test
  adf_result <- adf.test(ts_data)

  # Print the result with a custom message
  if (adf_result$p.value < 0.05) {
    print(paste(name, ": Stationary (p-value =", round(adf_result$p.value, 3),
                ", statistic =", round(adf_result$statistic, 3), ")"))
  } else {
    print(paste(name, ": Non-Stationary (p-value =", round(adf_result$p.value, 3),
                ", statistic =", round(adf_result$statistic, 3), ")"))
  }
}
############## recreat a final df
diff_ts_df <- as.data.frame(do.call(cbind, diff_ts))

# Assign column names to the differenced data frame (e.g., "Diff_Series1", "Diff_Series2",
...)
colnames(diff_ts_df) <- paste0("diff_", names(diff_ts))

# Ensure the deseasonalized data frame has the same number of rows by excluding its first
row
deseasonalized_df <- deseasonalized_df[-1, ]

# Combine the deseasonalized data with the differenced data
combined_df <- cbind(deseasonalized_df, diff_ts_df)
#add back the tsdm
combined_df <- cbind(combined_df, detrented_tsdm[-1])
```

```
# Convert to a data frame if needed
combined_df <- as.data.frame(lapply(combined_df, as.numeric))


#forecasting horizon
cor_matrix <- cor(combined_df[, -which(names(combined_df) %in% c("Date", "RADIATION_mean",
"MIN_TEMP_mean"))])
# Print the correlation matrix

png("correlation_heatmap.png", width = 800, height = 800)
corrplot(cor_matrix, method = "color", type = "upper",
        col = colorRampPalette(c("red", "white", "blue"))(200),
        tl.col = "black", tl.srt = 45, # Label color and rotation
        addCoef.col = "black", number.cex = 0.7, # Display correlation coefficients
        title = "Correlation Heatmap", mar = c(0,0,1,0)) # Add title
dev.off()
#check for colinearity
analyses_df = combined_df[, -which(names(combined_df) %in% c("Date", "RADIATION_mean",
"MIN_TEMP_mean","RH_TMIN_mean"))]
############## VAR modelling

VARselect(analyses_df, lag.max = 8, type = "const")[["selection"]]
#check the residuals are uncorrelated
var1 <- VAR(analyses_df, p=1, type="const")
serial.test(var1, lags.pt=10, type="PT.asymptotic")
var2 <- VAR(analyses_df, p=2, type="const")
serial.test(var2, lags.pt=10, type="PT.asymptotic")
###
dev.off()
nhor = 12
var_model <- VAR(analyses_df, p = 6, type = "const")
forecast <- predict(var_model, n.ahead = nhor)
#training gt
gt_main <- paddock_monthly_stats$mean_value[1:59]
last_values_gt <- tail(paddock_monthly_stats$mean_value, nhor)
indexes = deseasonalized_df$Date[1:59]
#forecast
sdm_forecast <- forecast$fcst$detrented_tsdm..1.[, "fcst"]
last_12 = tail(trend_component, nhor)
forecast = last_12+sdm_forecast

last_12_dates <- deseasonalized_df$Date[60:71]
training_dates <- combined_df
#####################################
png("forecast_var.png", width = 1600, height = 800)
xlim_range <- as.Date(c("2017-01-01", "2023-01-01"))
plot(indexes, gt_main, type = "l", col = "blue", lwd = 2,
     main = "TSDM Forecast for 2023 - VAR",
     xlab = "Date", ylab = "TSDM, kg/ha", xlim=xlim_range, xaxt = "n")

axis.Date(1, at = seq(xlim_range[1], xlim_range[2], by = "year"), format = "%Y")


# Add points for clarity
lines(last_12_dates, forecast, col = "red", lwd = 2, lty = 2)
lines(last_12_dates, paddock_monthly_stats$mean_value[60:71], col = "blue", lwd = 2, lty =
2)

legend("topleft", legend = c("Ground Truth", "Forecasted"),
       col = c("blue", "red"), lty = c(1, 2), lwd = 2)
dev.off()
#############################
#analyses
calculate_metrics <- function(forecasted, ground_truth, time_span) {
  # Extract the forecasted and ground truth values for the specified time span
```

```
  forecasted_values <- forecasted[1:time_span]
  ground_truth_values <- ground_truth[1:time_span]

  # Calculate RMSE
  rmse <- sqrt(mean((ground_truth_values - forecasted_values)^2))

  # Calculate MAE
  mae <- mean(abs(ground_truth_values - forecasted_values))

  # Return the results as a named list
  return(list(RMSE = rmse, MAE = mae))
}

results_df <- data.frame(Time_Span = integer(), RMSE = numeric(), MAE = numeric())

# Loop through each time span from 1 to 12
for (span in 1:12) {
  # Calculate the metrics for the current time span
  metrics <- calculate_metrics(forecasted = forecast,
                               ground_truth = paddock_monthly_stats$mean_value[60:71],
                               time_span = span)

  # Append the results to the data frame
  results_df <- rbind(results_df, data.frame(Time_Span = span, RMSE = metrics$RMSE, MAE =
metrics$MAE))
}
############################## final plot
ggplot(results_df, aes(x = Time_Span)) +
  geom_line(aes(y = RMSE, color = "RMSE"), size = 1) +
  geom_line(aes(y = MAE, color = "MAE"), size = 1) +
  labs(title = "RMSE and MAE over Time Span",
       x = "Time Span (Prediction Horizon)",
       y = "Error Metric") +
  scale_color_manual(name = "Metric", values = c("RMSE" = "blue", "MAE" = "red")) +
  scale_x_continuous(breaks = scales::pretty_breaks(0)) +  # Set x-axis to show integers
only
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5),  # Center the title
    axis.text.x = element_text(size = 10),   # Adjust x-axis text size if needed
    axis.title.x = element_text(margin = margin(t = 10))  # Add space above x-axis title
  )
##########################
summary(var_model)

forecast(var_model ) %>%
  autoplot()
```