OKARA PRICE IN NEPAL FROM 2013 TO 2021 ANALYSIS

Group 5

Date: November 18, 2025

Presenter: Group 5

Purpose of the Proposal

INTRODUCTION

Why okara ?

01 Okara harvests have frequently fluctuated seasonally, making it difficult to effectively determine the price

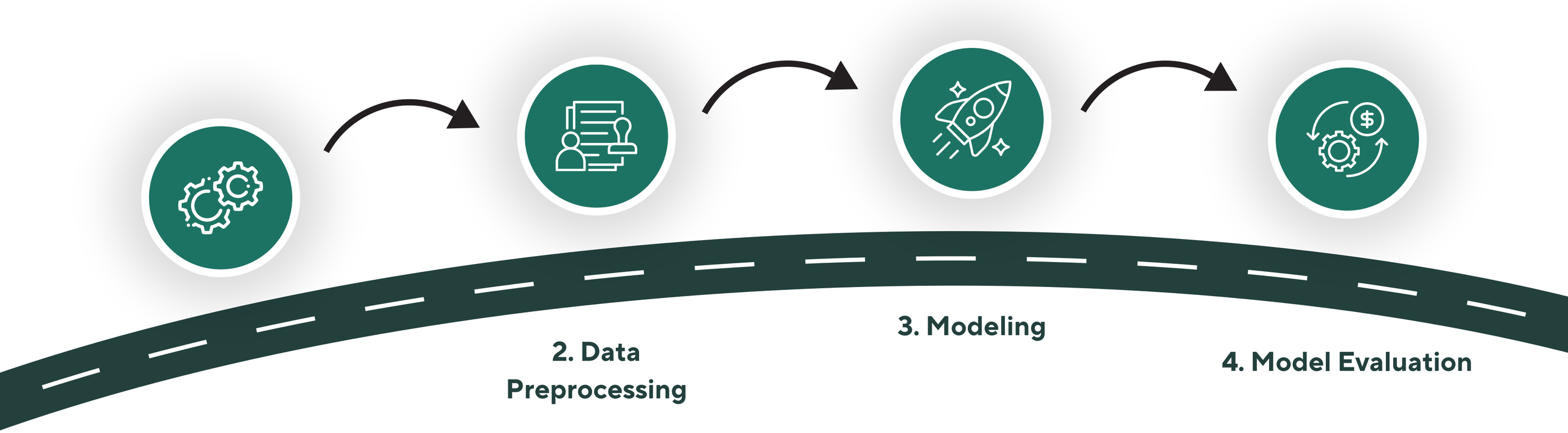
02 With the rise in demand for plant-based meat, Okara can be used in various ways to create plant-based meat products

Objective

01 Predicting minimum daily price of Okara

02 Observe seasonality and trend of Okara price over the last few year

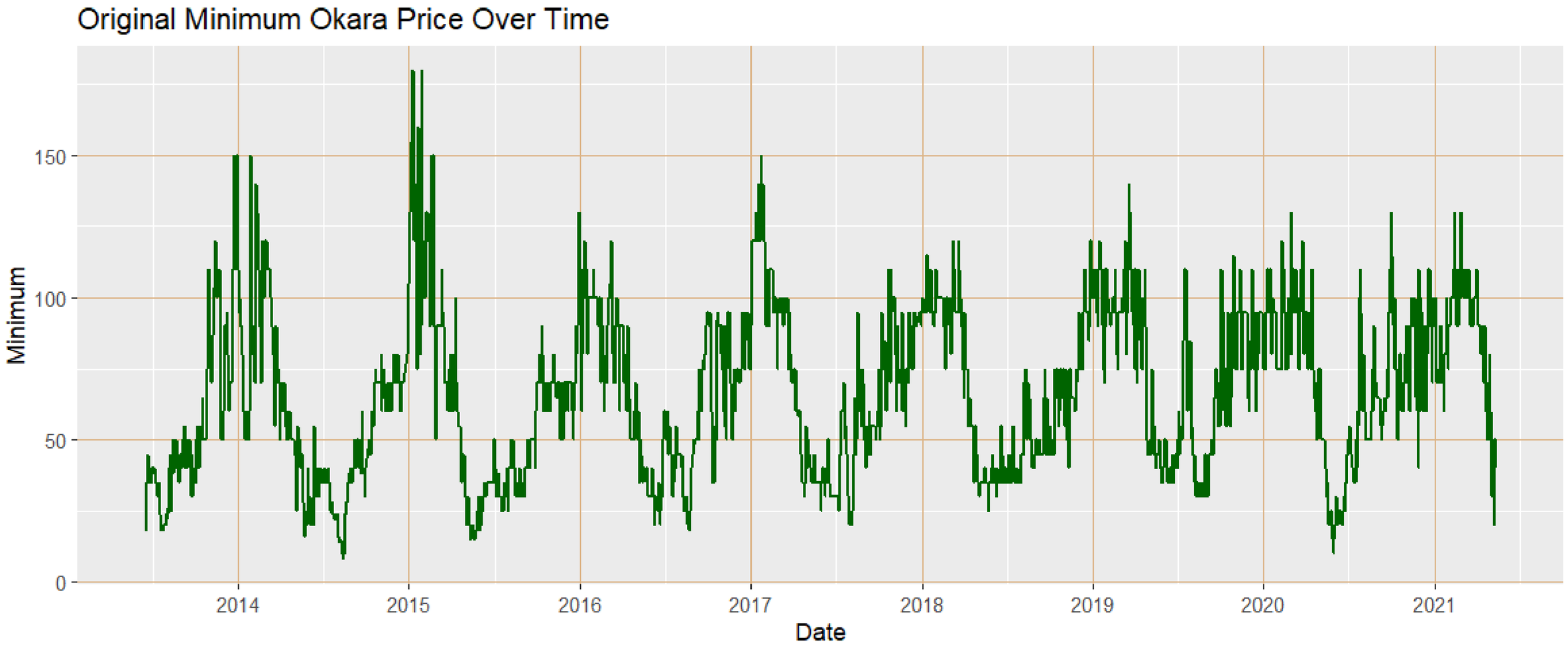
RESEARCH PROCESS



1. **Exploratory Data Analysis**
2. **Data Preprocessing**
3. **Modeling**
4. **Model Evaluation**

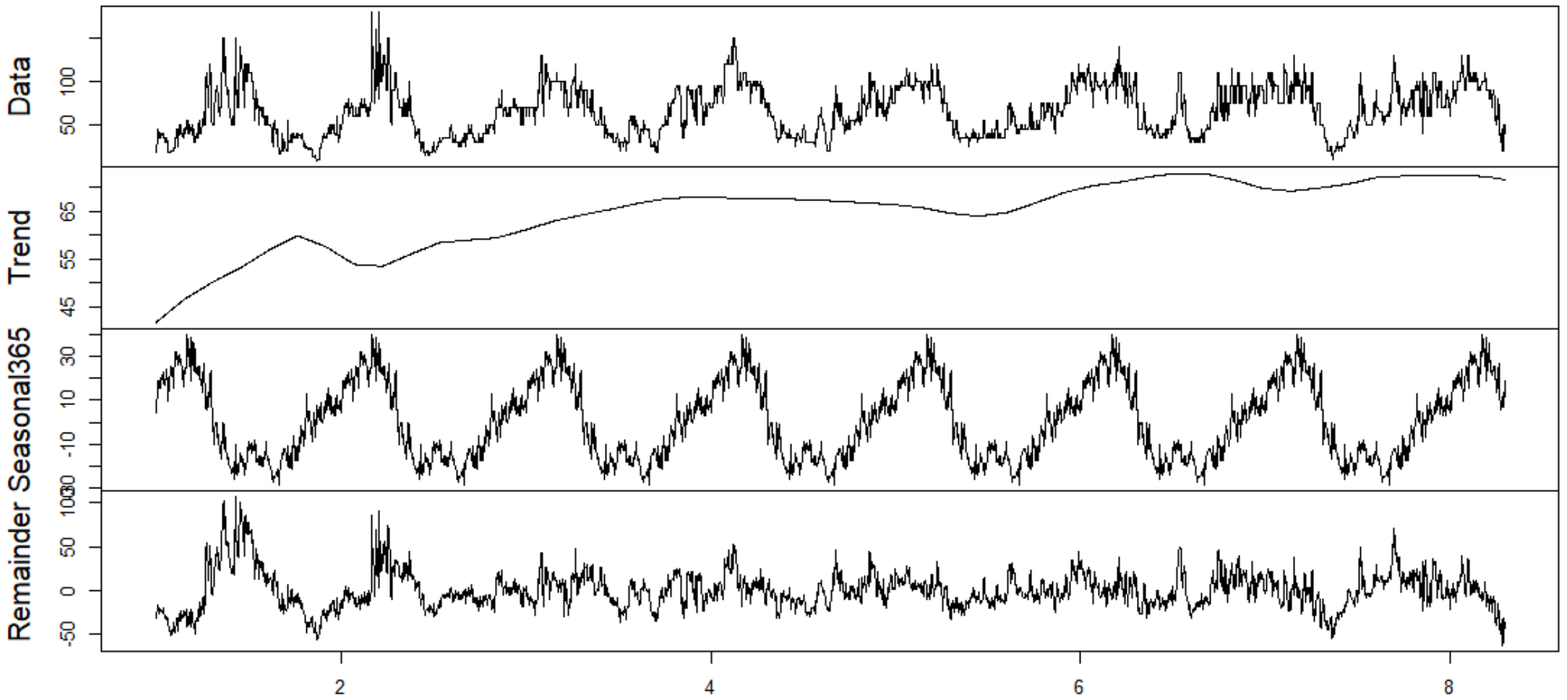
EXPLORATORY DATA ANALYSIS

OVERVIEW OF OKARA PRICE OVER TIME



DECOMPOSITION OF OKARA PRICE

Additive Decomposition using stl()



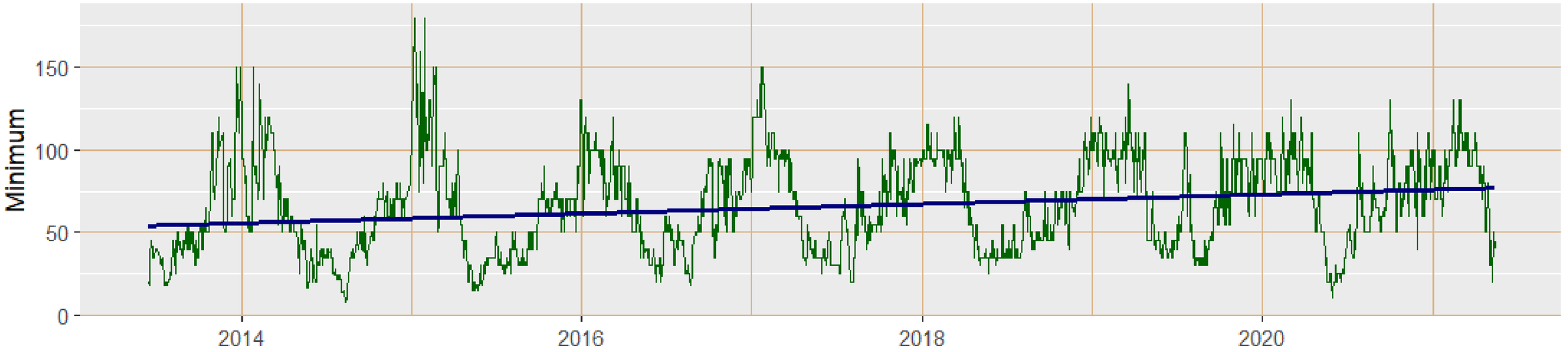
- After doing a additive decomposition, we observe that the it is **unclear** to determine the type of **trend** since the scale are very small
- This data has a **seasonal** period of 1 year or 365 days.
- We also seen a the remainder from this composition as well

Reason for additive decomposition :

From this plot and last plot from last, Seasonal effect does not differentiate greatly for each point in time.

TREND AND STATIONARITY OBSERVATION

Minimum Price of Okara with Regression Line



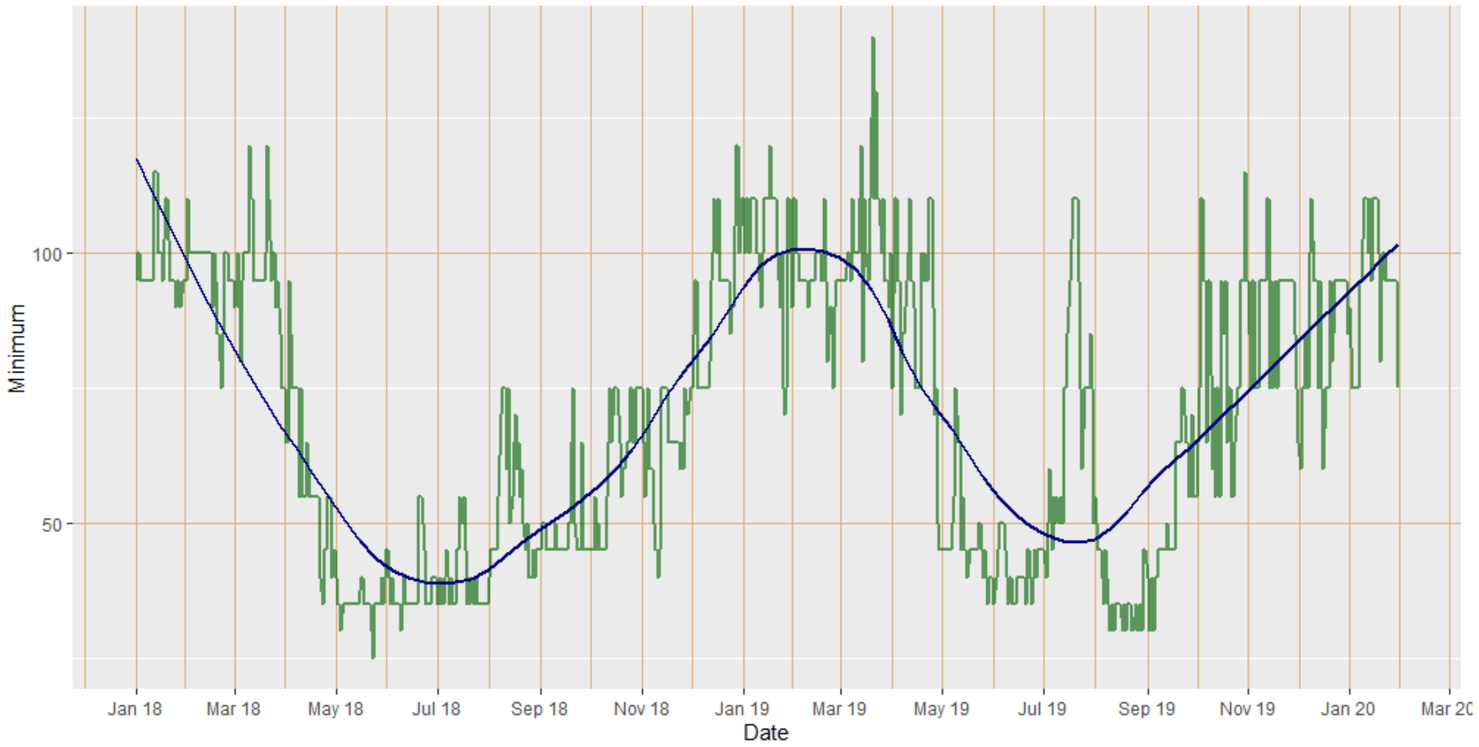

```
        Augmented Dickey-Fuller Test

data:  veg.ts
Dickey-Fuller = -4.3999, Lag order = 13, p-value = 0.01
alternative hypothesis: stationary
```

From the plot above, It is unclear to determine whether the data is stationarity.

By using Augmented Dickey-Fuller Test, the p -value is under 0.05. Therefore this data achieve stationarity.

SEASONALITY OF OKARA OBSERVATION

Minimum Price of Okara with Loess Smoothing from 2018 to 2020

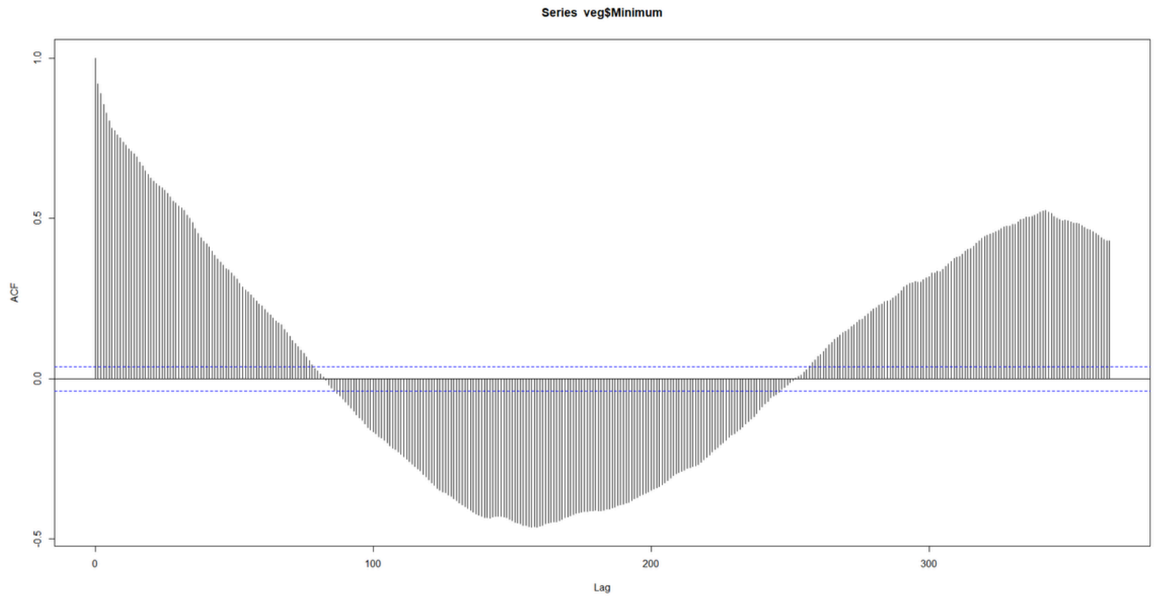


Seasonality of Okara Price

- Looking at the plot with Loess Smoothing, data show a clear seasonal effects
- The minimum price of Okara is at the **lowest** around June to September which is the **monsoon** season
- The minimum price of Okara **peak** around January to March which is the **winter** season
- Soybeans typically grow better in the monsoon season in Nepal due to ideal warm temperatures and abundant rainfall

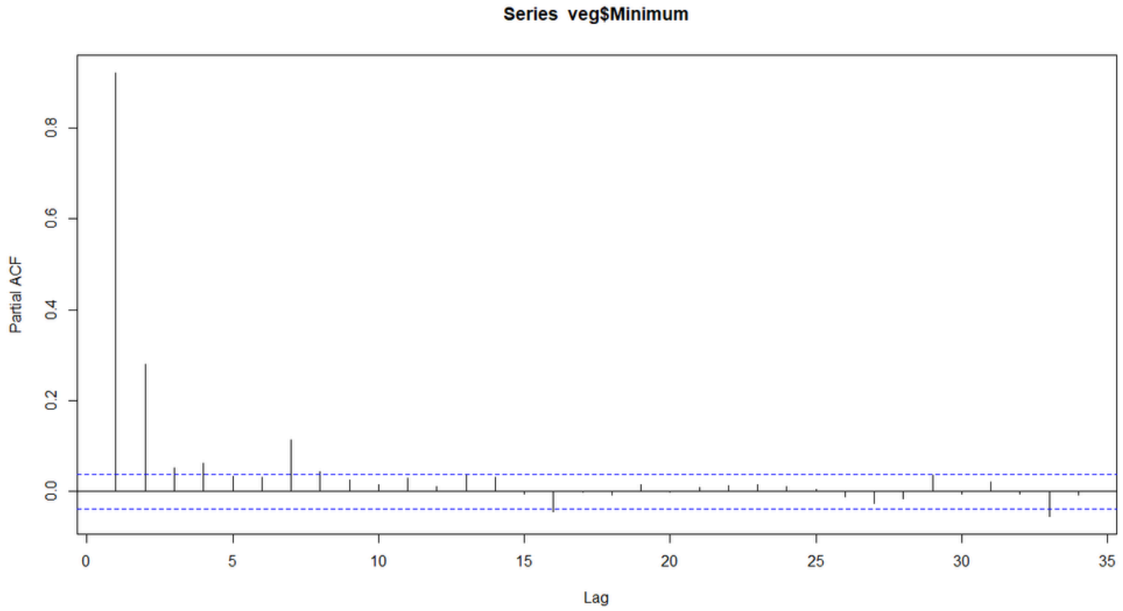
CORRELATION PLOTS

Auto correlation function



- From ACF, data is significantly autocorrelated at many lags
- Exhibit clear seasonal pattern

Partial auto correlation function



- The PACF plot indicates **significant** autocorrelation in the data at **lags 1 through 4**

using lag.max = 365 to observe seasonal pattern of the data

DATA PREPROCESSING

DEALING WITH MISSING DATE

Date is missing at random

6	Okara	2013-06-21	45
7	Okara	2013-06-25	40
8	Okara	2013-06-26	35
37	Okara	2013-08-01	22
38	Okara	2013-08-02	26
39	Okara	2013-08-04	24

- There are 226 missing date from start date to end date

Solution

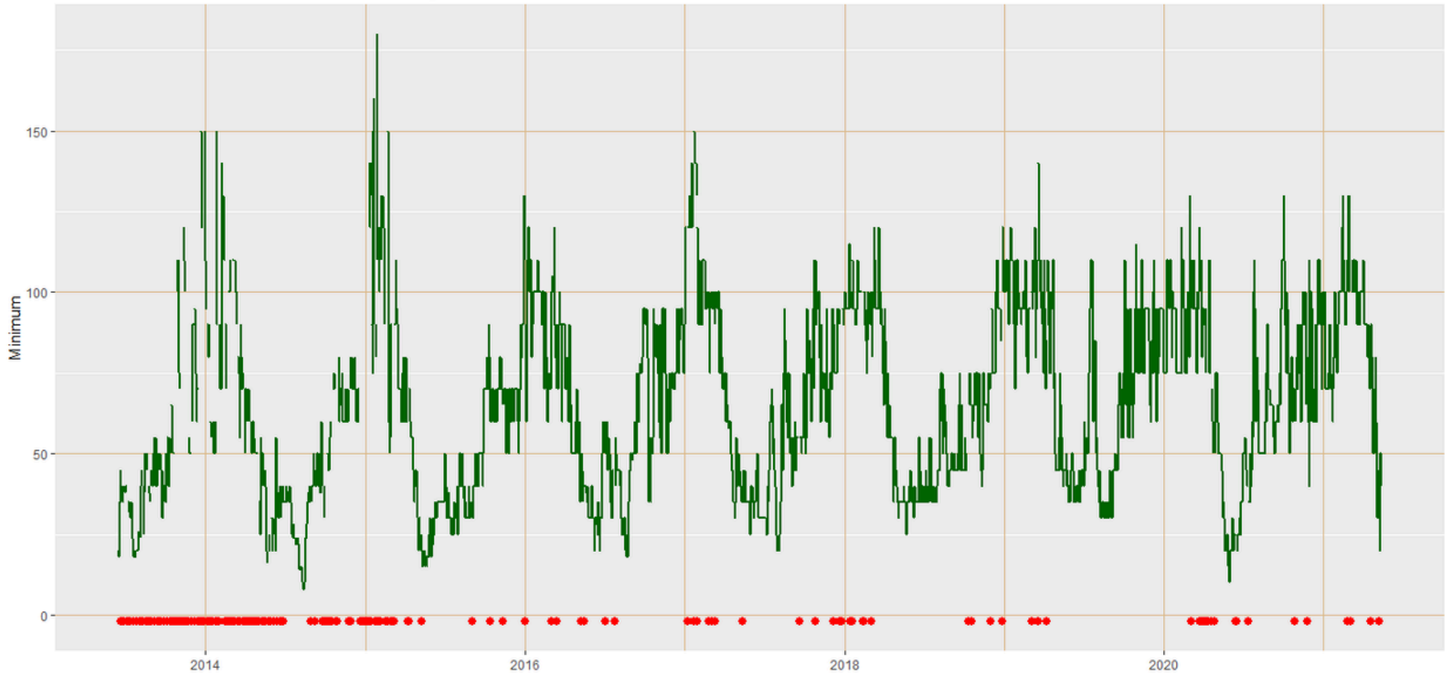
```
#Padding Missing Date
veg_pad = pad(veg, interval = "day")
veg_pad$Commodity = "Okara"
```

6	Okara	2013-06-21	45
7	Okara	2013-06-22	*NA*
8	Okara	2013-06-23	*NA*
9	Okara	2013-06-24	*NA*
10	Okara	2013-06-25	40

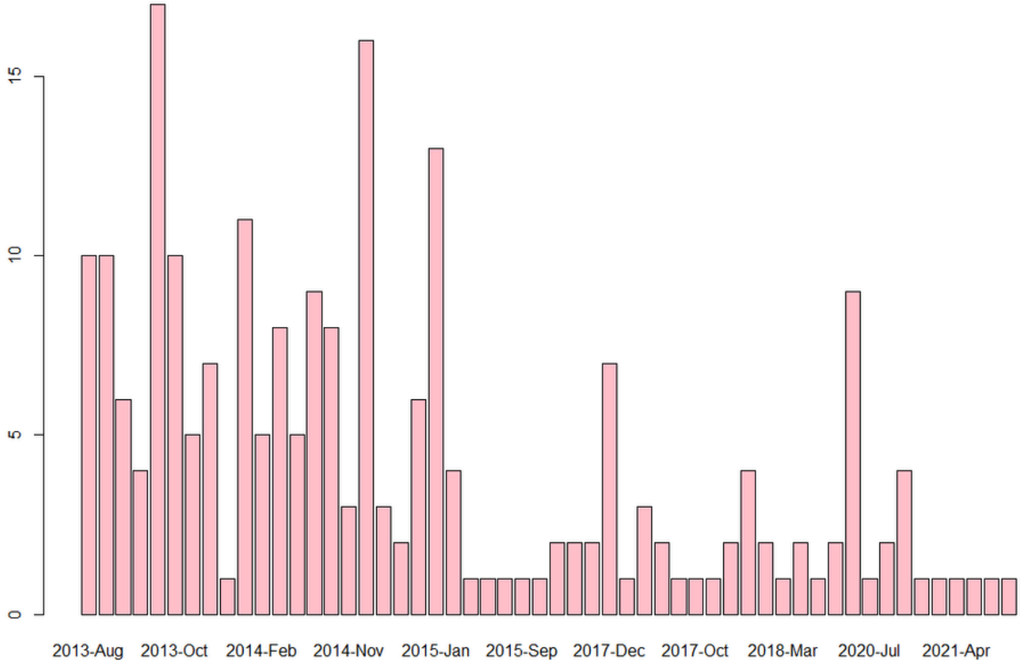
- Using pad function from padr library to fill in the missing date

OBSERVING MISSING VALUE

Plot of missing value



Histogram of missing values by date



Both graphs reveal a high frequency of missing values, particularly at the beginning of the dataset. This suggests potential data collection issues or inconsistencies during the initial period.

SPLITTING THE DATA INTO TRAINING AND TESTING SETS

Splitting technique

```
train_set = veg_pad %>%
  filter(Date >= "2013-06-16", Date <= "2019-10-06")

test_set = veg_pad %>%
  filter(Date > "2019-10-06")
```

- Employ an 80/20 chronological train-test split for the time series data. The first 80% of the time series will constitute the training set, and the remaining 20% will serve as the test set
- Split point is 2019-10-06
- To prevent data leakage, the dataset is divided into training and testing sets prior to imputation. This ensures that information from the testing set does not inadvertently influence the imputation process for the training set

MISSING VALUE IMPUTATION USING KALMAN FILTER

Function for Kalman Imputation

```r
impute_ts_kalman <- function(df){
  ts = ts(df[,3])
  ts_imputed = na_kalman(ts,model = "StructTS",smooth = TRUE)
  temp = df
  df_imputed = cbind(temp,data.frame(ts_imputed))
  df_imputed = df_imputed[,-3]
  names(df_imputed)[3] <- "Imputed_Minimum"
  return(df_imputed)
}
```

Imputation Method\ Data set	Mean Rank	Mean Best Rank
KNN	4	4
Mean	5	5
Missing Indicators	3	3
Regression	5	5
Pattern	8	6
Pattern & Kalman	2	1
Decomposition	9	8
Additive Decomposition	11	9
Decomp. & Kalman	7	7
Add. Decomp & Kalman	6	6
Kalman	1	2

Imputation methods

- Kalman imputation uses a unique approach to predict and refine missing values based on prior states and observed measurements and can captures periodic patterns , suitable for data with trends, seasonality, and noise.

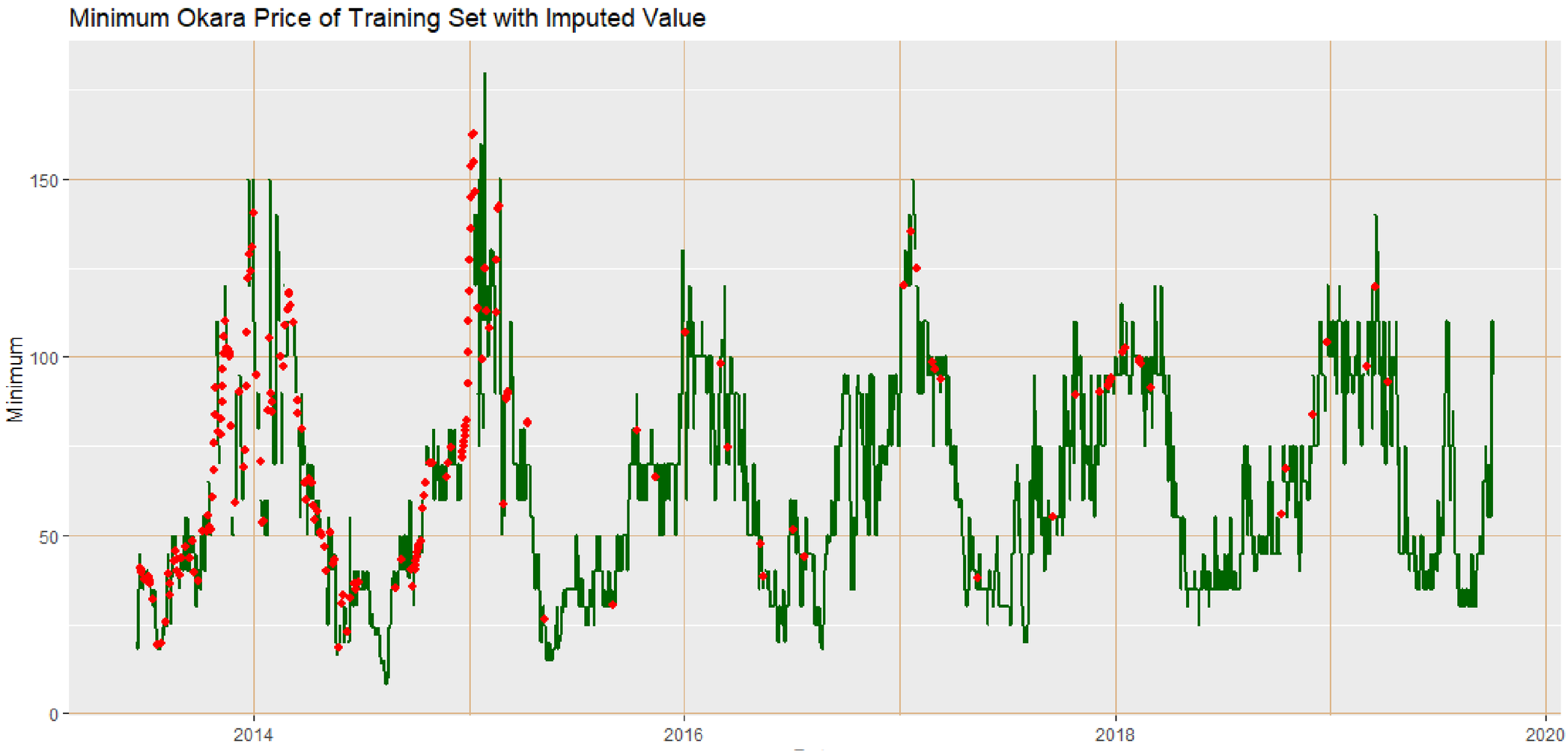
- From research which use seasonal data, Kalman approach rank at the top by the RMSE metric comparing to other imputation method

- Our data is noisy and seasonal. Perfect for Kalman Imputation

source : Ribeiro, S. M. (2022). Imputation by decomposition and by time series nature: Novel imputation methods for missing data in time series (Master's thesis, Universidade Federal de Minas Gerais)

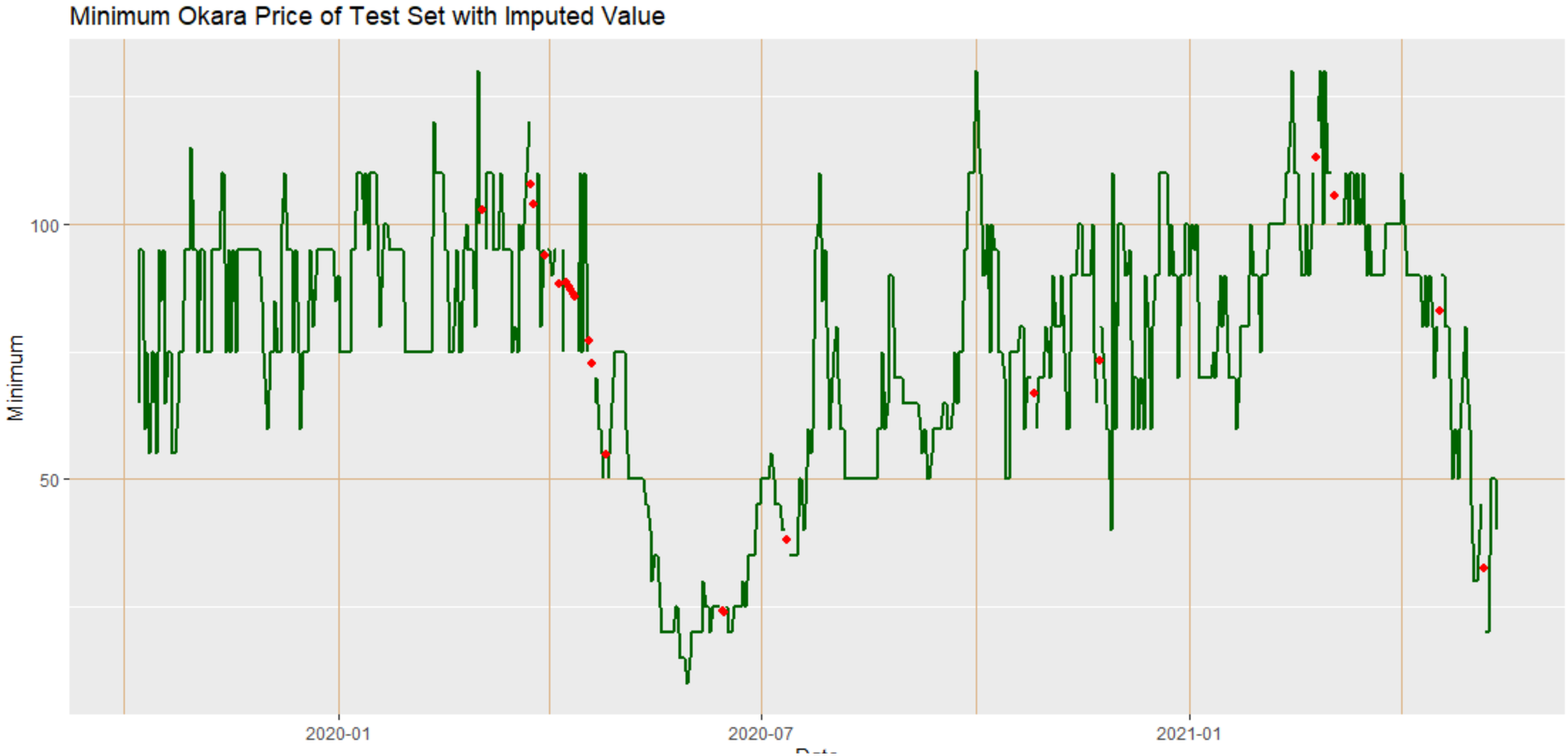
RESULT OF IMPUTATION

Training set



RESULT OF IMPUTATION

Testing set



Kalman Imputation provide satisfactory result with their reasonable estimates

OUTLIERS DETECTION

USING TSOUTLIERS() FUNCTION

Outliers detection

```
> outliers = tsoutliers(ts(train_imputed$Imputed_Minimum))
> print(outliers)
$index
[1] 571 592
```

	Commodity	Date	Imputed_Minimum
571	okara	2015-01-07	180
592	okara	2015-01-28	180

- There are 2 outliers in the training set
- Replacing outlier by using Kalman Imputation

tsoutliers() function process

1. decompose data into trend and seasonality and remainder
2. find outlier in the remainder series
3. Observations are labelled as outliers if their remainder are less than Q1 – 3× IQR or greater than Q3 + 3× IQR

Result of outliers replacements

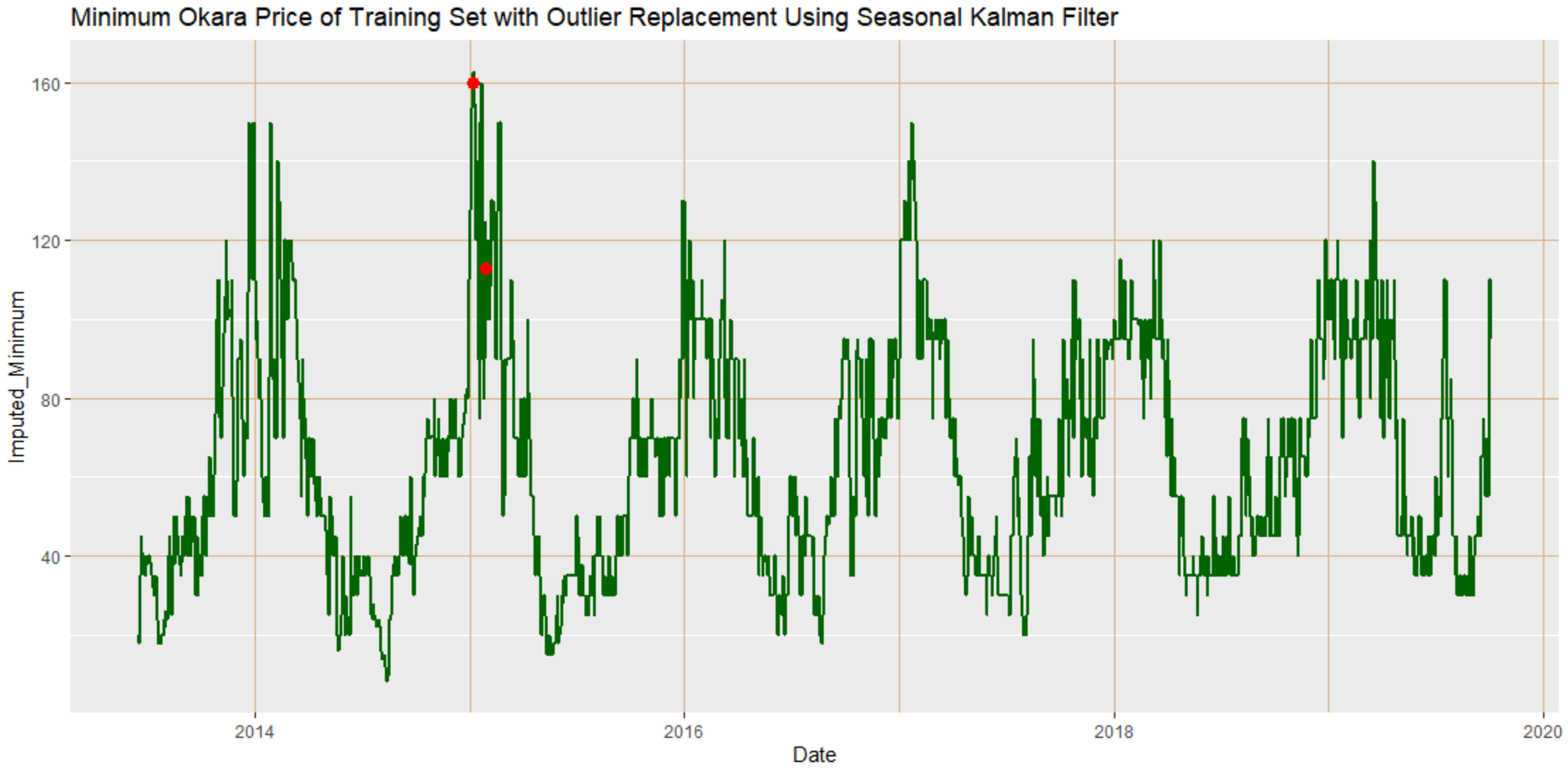
	Commodity	Date	Imputed_Minimum
571	okara	2015-01-07	159.829
592	okara	2015-01-28	112.951

source : https://robjhyndman.com/hyndsight/tsoutliers/

OUTLIERS DETECTION

USING TSOUTLIERS() FUNCTION

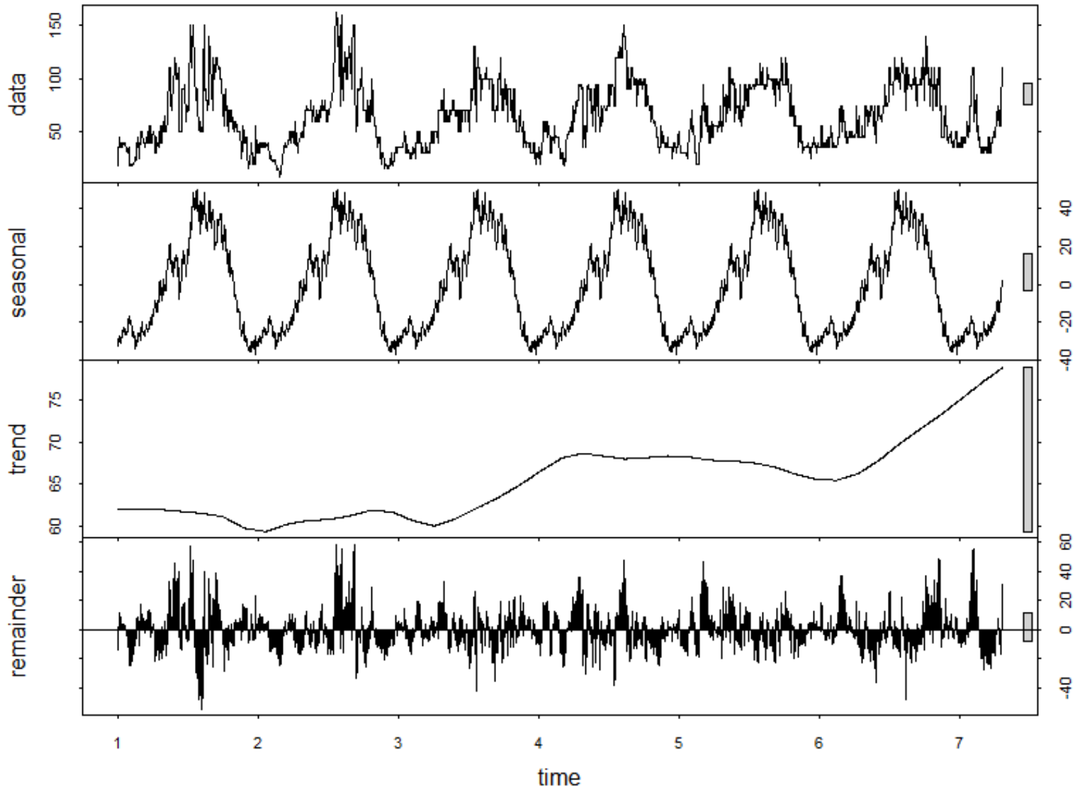
Result of outliers replacements



MODELING

ETS

Model 2: ETS (Daily)



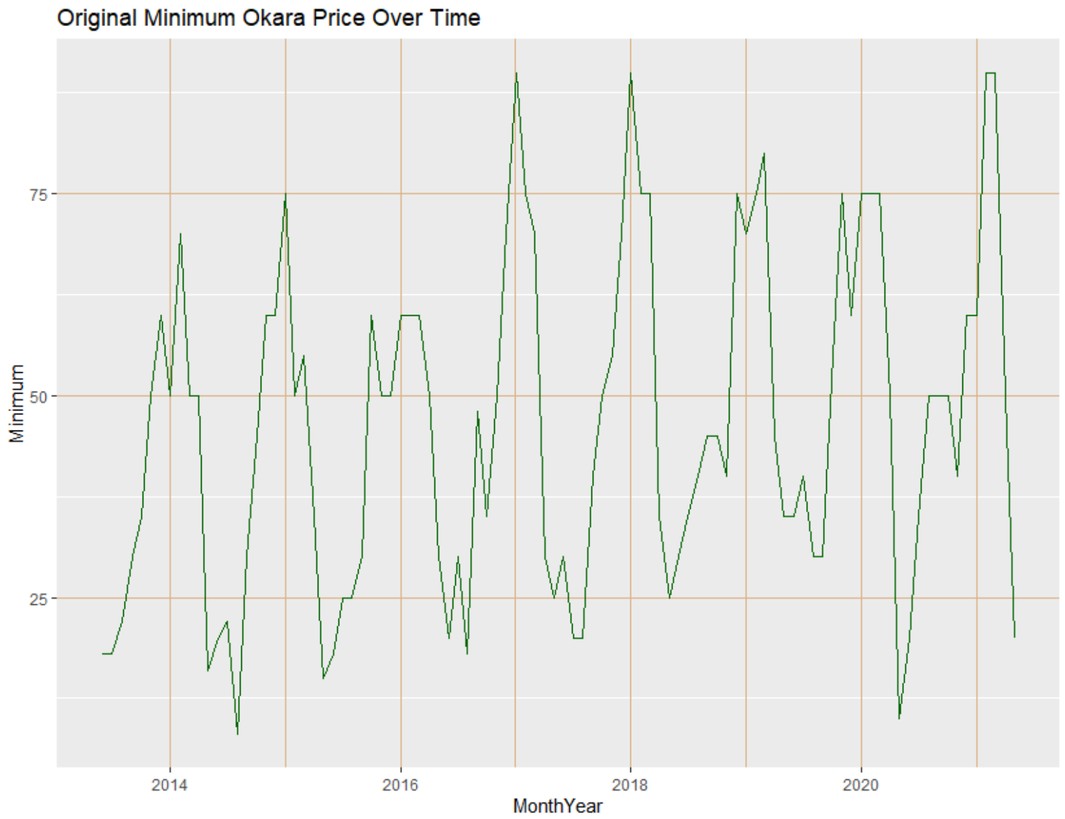

```
 # Fit a manually specified ETS model
 manual_ets_model <- ets(train_ts, model = "MNM")
rror in ets(train_ts, model = "MNM") : Frequency too high
```

The error "Frequency too high" occurs because the time series object has a high seasonal frequency that exceeds the limitations of the ETS (Exponential Smoothing State Space) model in the forecast package.

To resolve this error, we aggregate the data to a lower frequency—monthly.

ETS

Plotting the time series after aggregating the data into monthly intervals



The data is not very different from the original data with daily frequency.

ETS

ETS (Monthly) : Independence test for full data

```
> #Independence Testing
> Box.test(monthlyfulldata.ts)

        Box-Pierce test

data:  monthlyfulldata.ts
X-squared = 43.193, df = 1, p-value = 4.959e-11
```

- Since the p-value is extremely small, we reject the null hypothesis.
- This indicates that the residuals or data are not independent, meaning there is likely autocorrelation present in the data.

ETS (Monthly): Outliers, consistent and NA check for full data

```
> print(outliers)
[1] index        replacements
<0 rows> (or 0-length row.names)
> nrow(outliers)
[1] 0

> is.regular(monthlyfulldata.ts)
[1] TRUE
>
> is.na(monthlyfulldata.ts)
  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [21] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [41] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [81] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

From the results, it is indicated that there are no outliers, the data is consistent, and there are no missing values (NA). Therefore, there is no need for imputation.

ETS

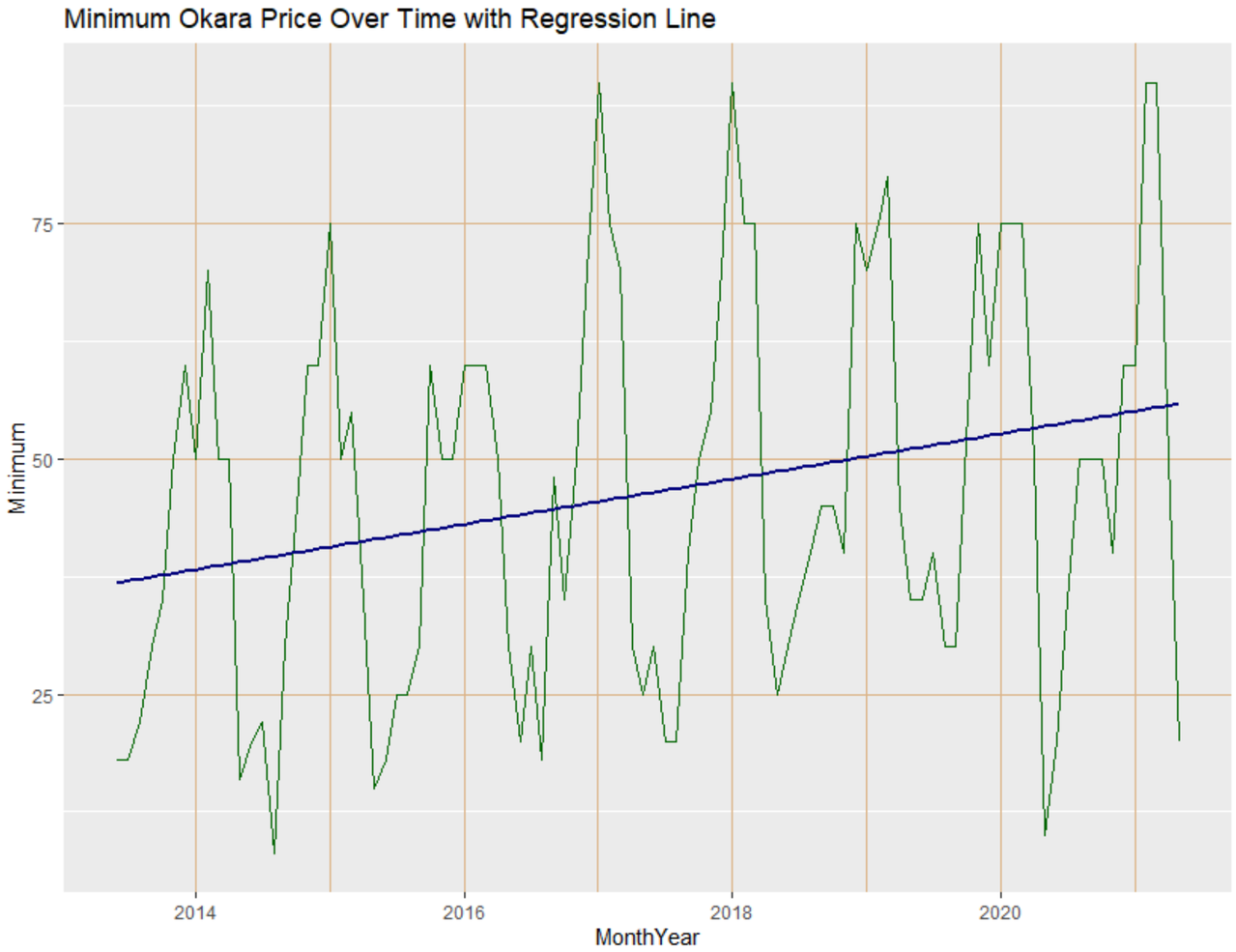
ETS (Monthly): Outlier checking on training set

```
> #Detecting outlier on training set
> outliers = tsoutliers(train.ts)
> print(outliers)
$index
integer(0)

$replacements
numeric(0)
```

From the results, it is indicated that there are no outliers for training set.

ETS

ETS (Monthly)



- The regression line indicates unclear in the monthly minimum Okara price over the years

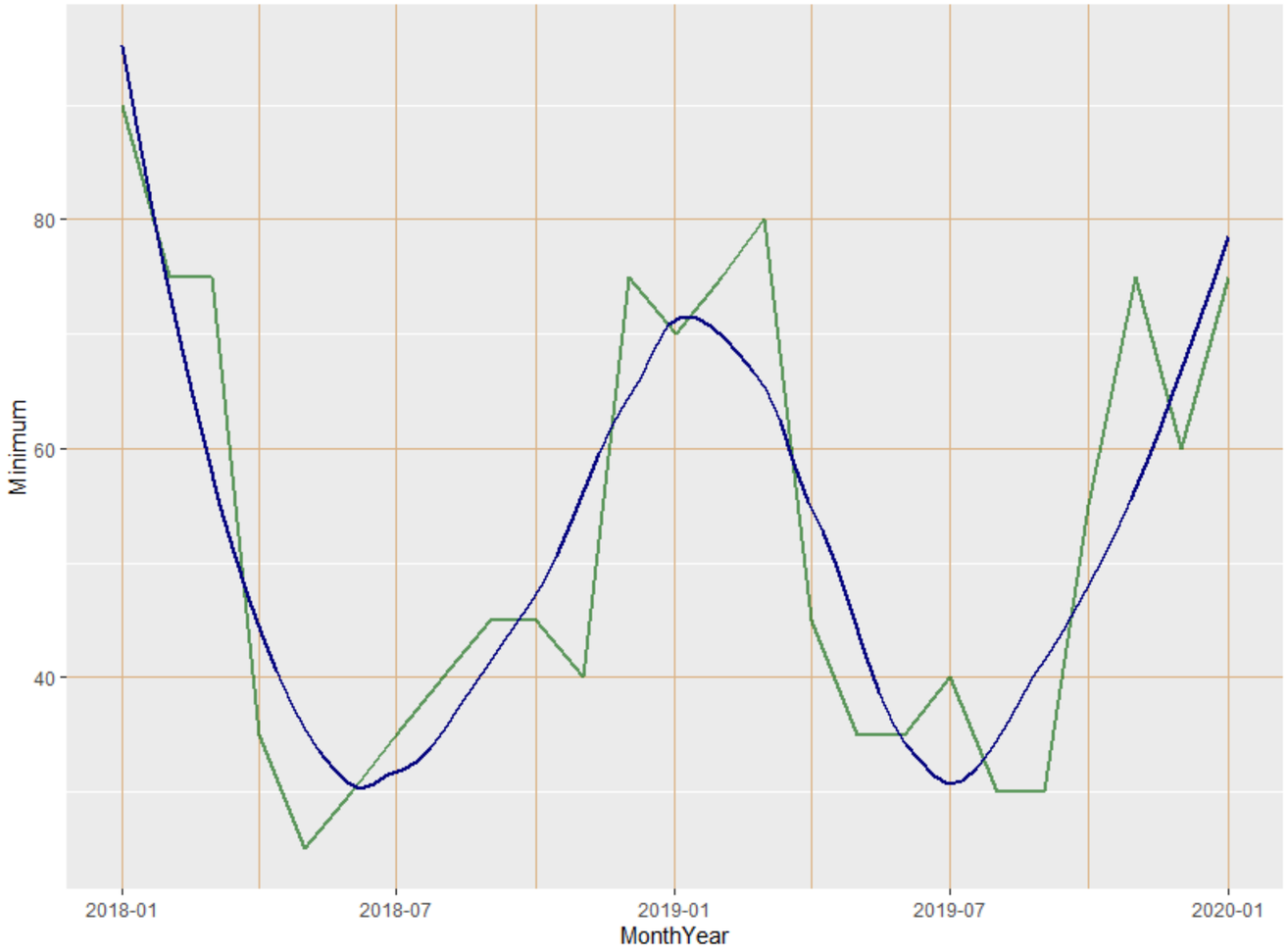
```
        Augmented Dickey-Fuller Test

data:  trainmonth$Minimum
Dickey-Fuller = -7.4374, Lag order = 4, p-value = 0.01
alternative hypothesis: stationary
```

```
> kpss.test(trainmonth$Minimum)

        KPSS Test for Level Stationarity

data:  trainmonth$Minimum
KPSS Level = 0.18692, Truncation lag parameter = 3, p-value = 0.1
```

- Using ADF test, KPSS test, both indicate the monthly price is stationary

ETS

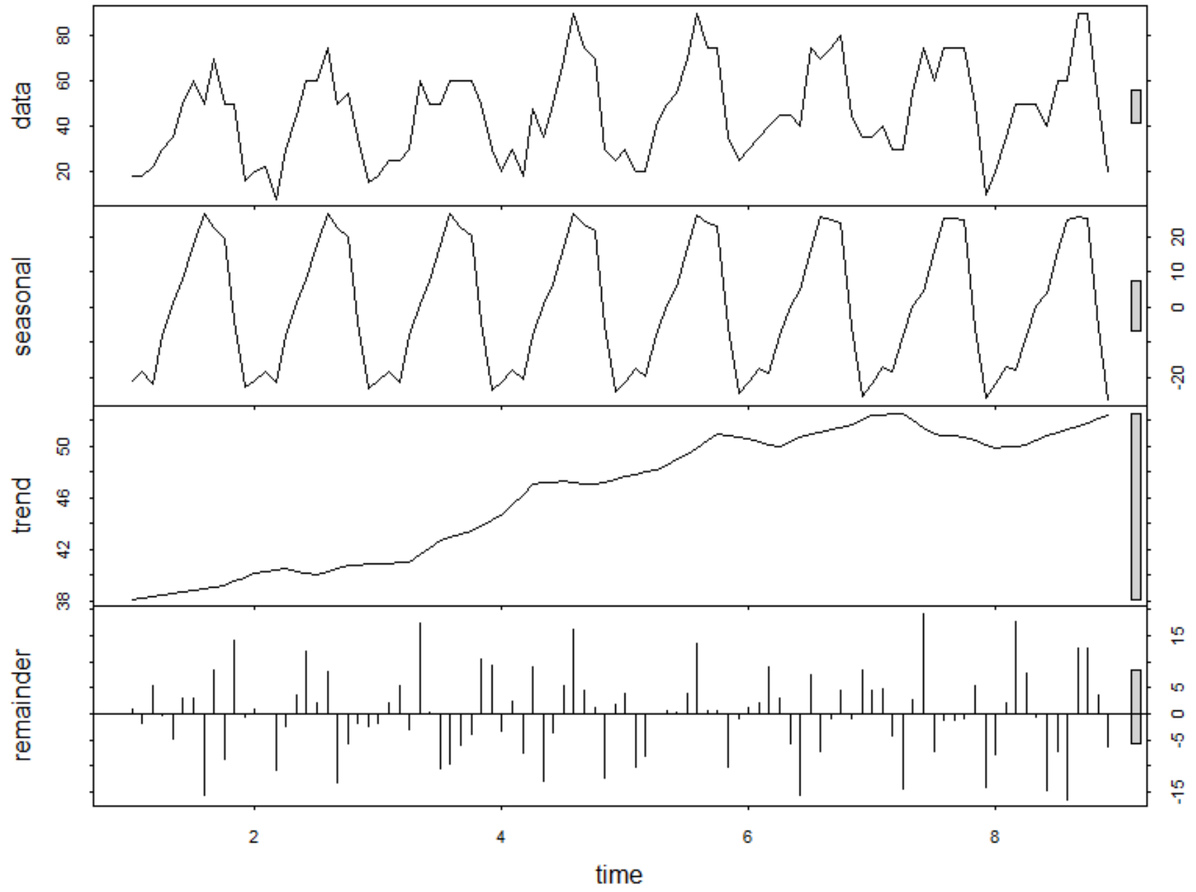
ETS (Monthly)



- From the plot using the monthly minimum price of Okara with Loess smoothing, the plot exhibit similiar seasonal pattern to the daily plot too.
- High price during winter season
- Low price during monsoon season

ETS

ETS (Monthly)



ETS

ETS (Monthly)

```
ETS(A,N,A)

Call:
ets(y = train_ts, model = "ANA")

  Smoothing parameters:
    alpha = 0.1025
    gamma = 1e-04

  Initial states:
    l = 41.8012
    s = -21.4335 -4.8328 19.8819 22.3653 27.3581 19.7522
           6.053 1.7354 -8.951 -22.1405 -17.7873 -22.0009

  sigma:  8.9555

     AIC     AICc      BIC
686.6299 694.4987 721.7870

Training set error measures:
                    ME     RMSE     MAE       MPE     MAPE      MASE          ACF1
Training set 1.152021 8.100516 6.35306 -1.11949 16.49127 0.6940317 -0.006859548
```

Quick summary

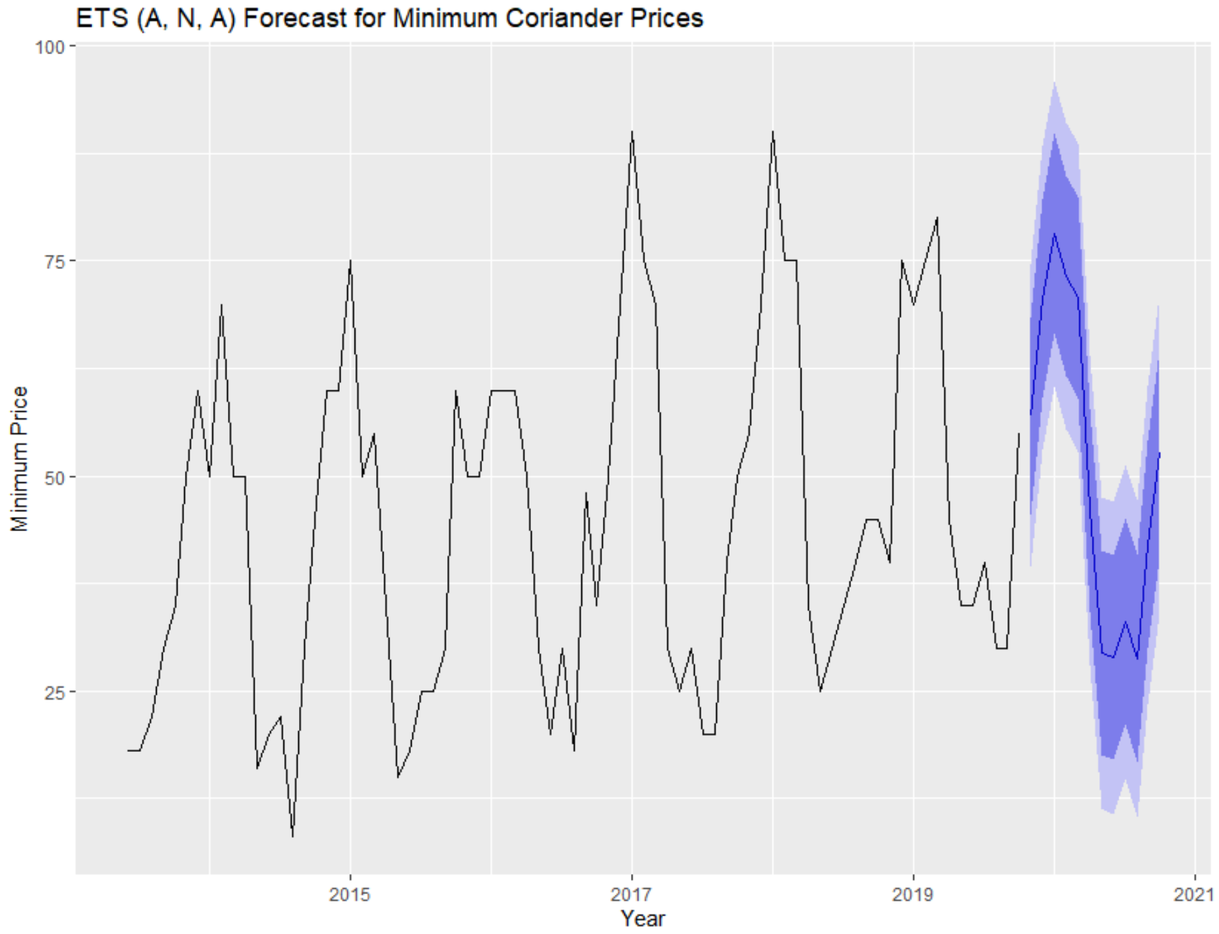
Error : Additive ; constant remainder from decompose

Trend : None ; according to ADF test

Seasonal: Additive ; constant seasonality components

ETS

Model 2: ETS (Monthly)




```
> # Print evaluation metrics
> cat("MAE:", mae, "\n")
MAE: 12.92536
> cat("RMSE:", rmse, "\n")
RMSE: 15.83527
> cat("MAPE:", mape, "%\n")
MAPE: 46.37578 %
```

1. **MAE:** On average, the forecast is off by approximately 12.92536 units from the actual values.
2. **RMSE:** Penalizes larger errors more heavily compared to MAE.
3. **MAPE:** The forecast error is about 46.38% of the actual values on average, which is relatively high.

SARIMA
DAILY

ADF Test

```
Warning message:
In adf.test(train_ts, alternative = "stationary") :
  p-value smaller than printed p-value

        Augmented Dickey-Fuller Test

data:  train_ts
Dickey-Fuller = -4.1435, Lag order = 13, p-value = 0.01
alternative hypothesis: stationary
```

- The result form ADF test indicate that **p-value < 0.01**.

Since p-value is less than 0.05, we reject the null Hypothesis (H_0) that the series has a unit root (non-stationary).

- This suggest that the series is **stationary**.

KPSS Test

```
        KPSS Test for Level Stationarity

data:  train_ts
KPSS Level = 0.49191, Truncation lag parameter = 8, p-value = 0.04349
```

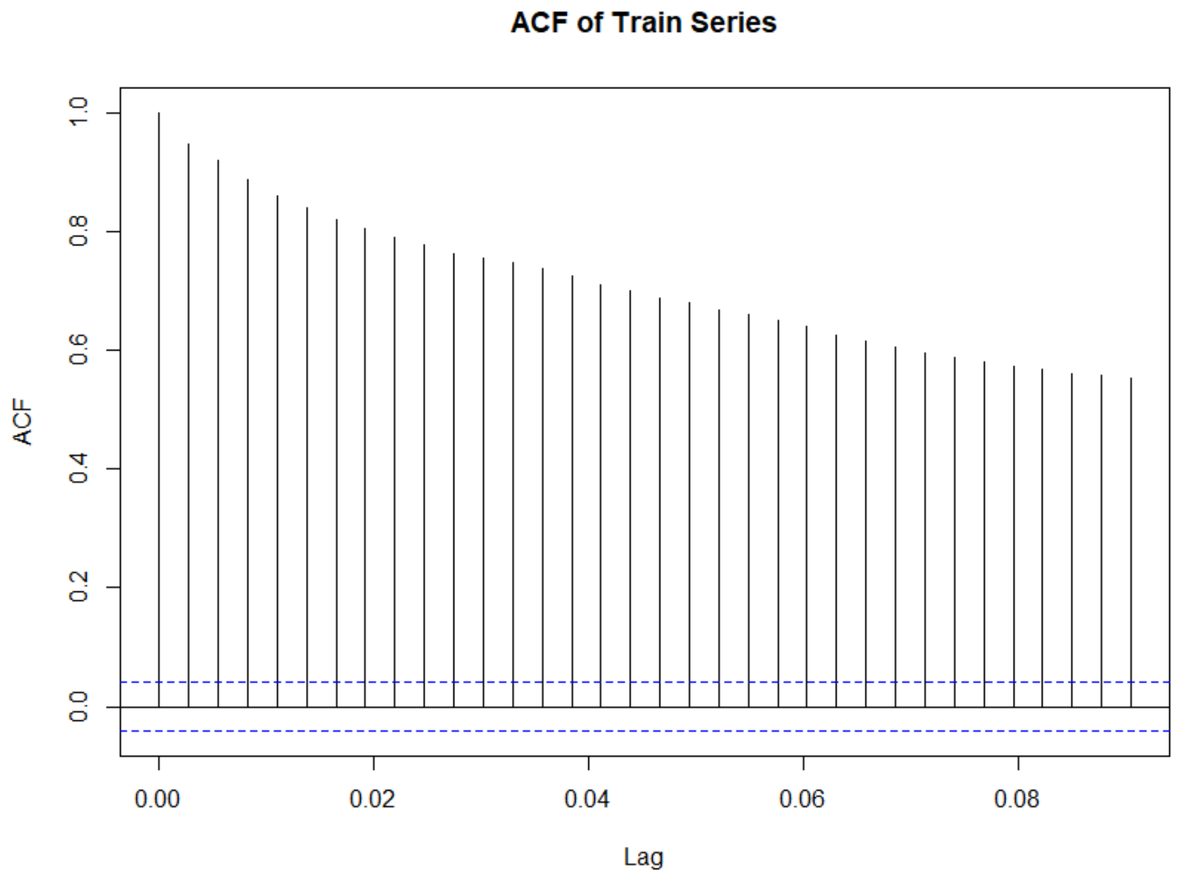
- The result form ADF test indicate that **p-value = 0.04349**.

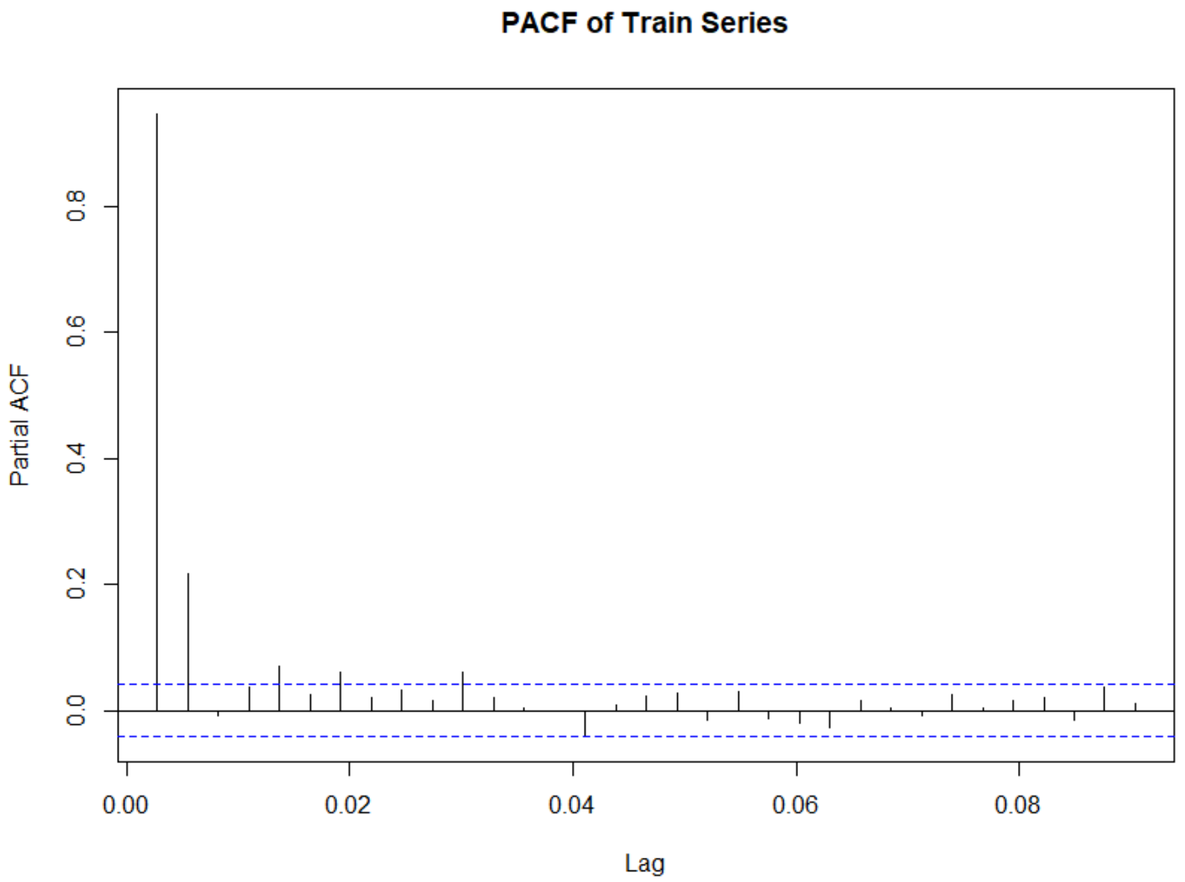
Since p-value is less than 0.05, we reject the null Hypothesis (H_0) that the series is stationary.

- This suggest that the series is **non- stationary**.

SARIMA
DAILY

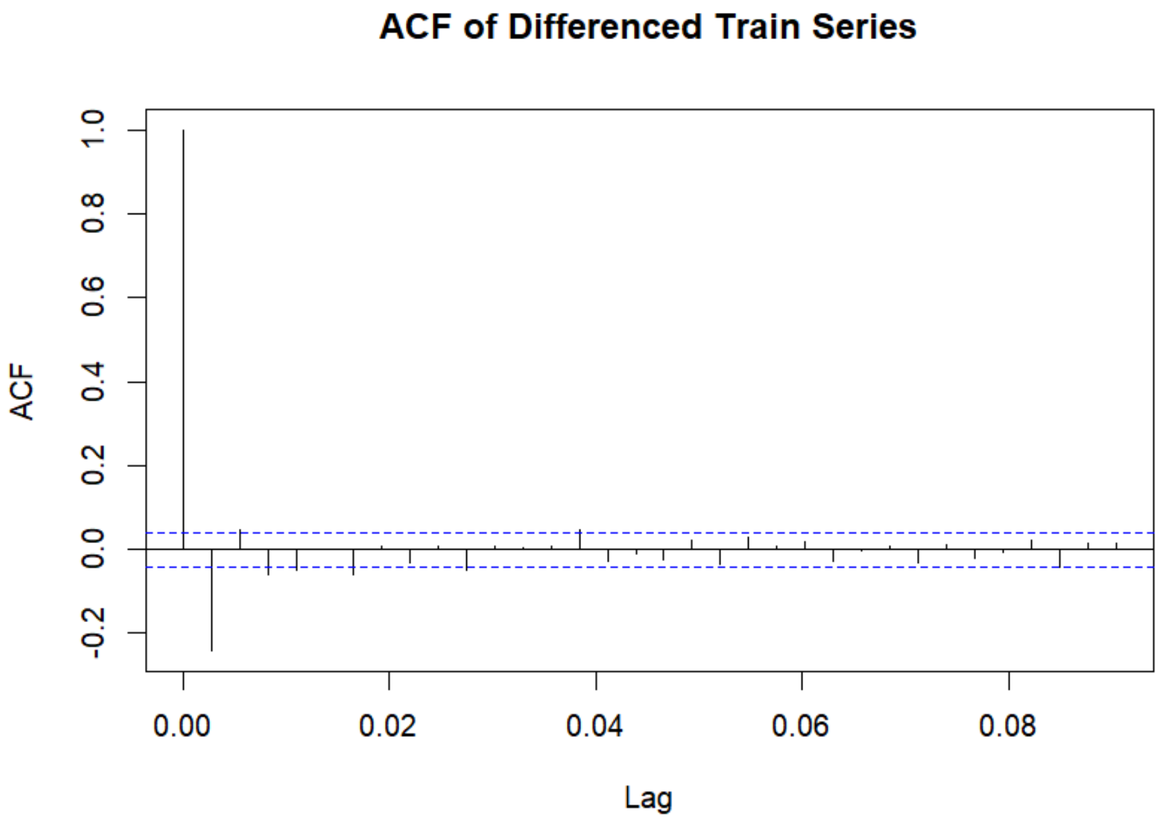
ACF & PACF plot

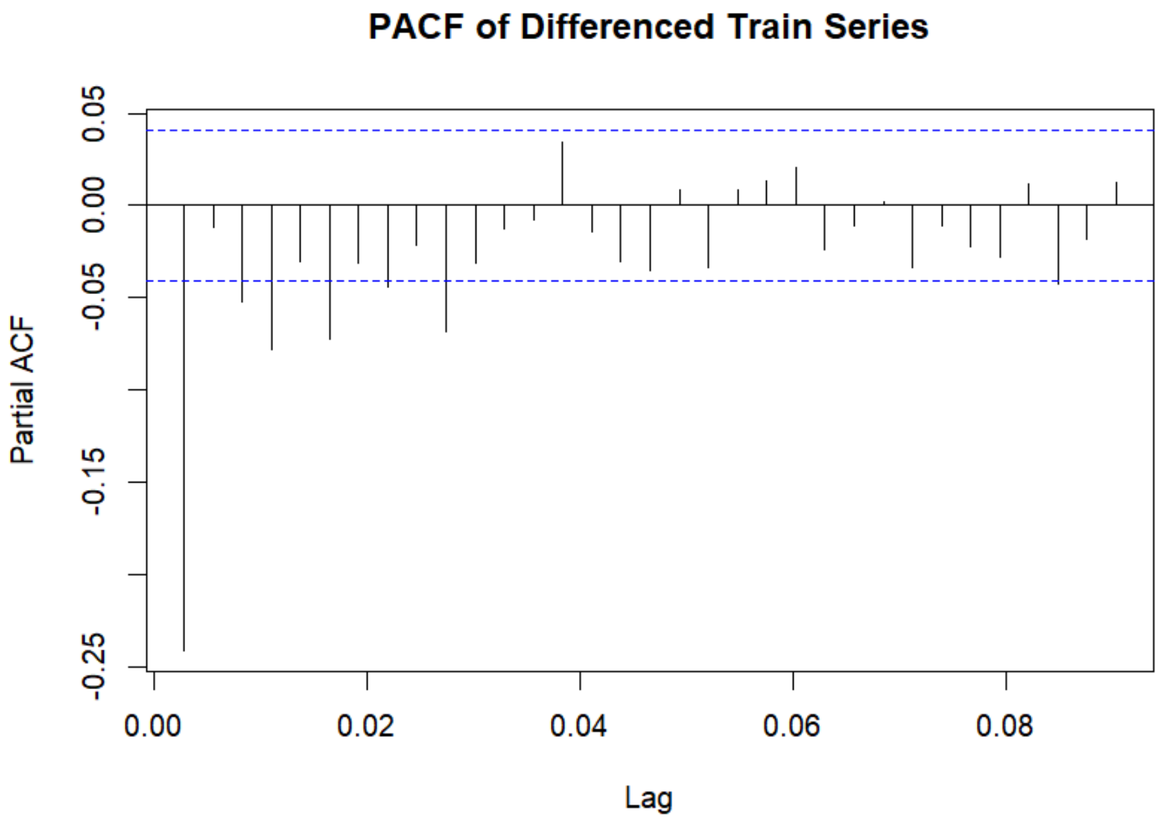




SARIMA
DAILY

ACF & PACF plot after 1st order differencing





SARIMA
DAILY

```
Error in makeARIMA(trarma[[1L]], trarma[[2L]], Delta, kappa, SSinit) :
  maximum supported lag is 350
```

SARIMA

ADF Test

```
Warning message:
In adf.test(train_ts, alternative = "stationary") :
  p-value smaller than printed p-value

        Augmented Dickey-Fuller Test

data:  train_ts
Dickey-Fuller = -7.4374, Lag order = 4, p-value = 0.01
alternative hypothesis: stationary
```

- The result form ADF test indicate that **p-value < 0.01**.

Since p-value is less than 0.05, we reject the null Hypothesis (H_0) that the series has a unit root (non-stationary).

- This suggest that the series is **stationary**.

KPSS Test

```
Warning message:
In kpss.test(train_ts) : p-value greater than printed p-value

        KPSS Test for Level Stationarity

data:  train_ts
KPSS Level = 0.18692, Truncation lag parameter = 3, p-value = 0.1
```

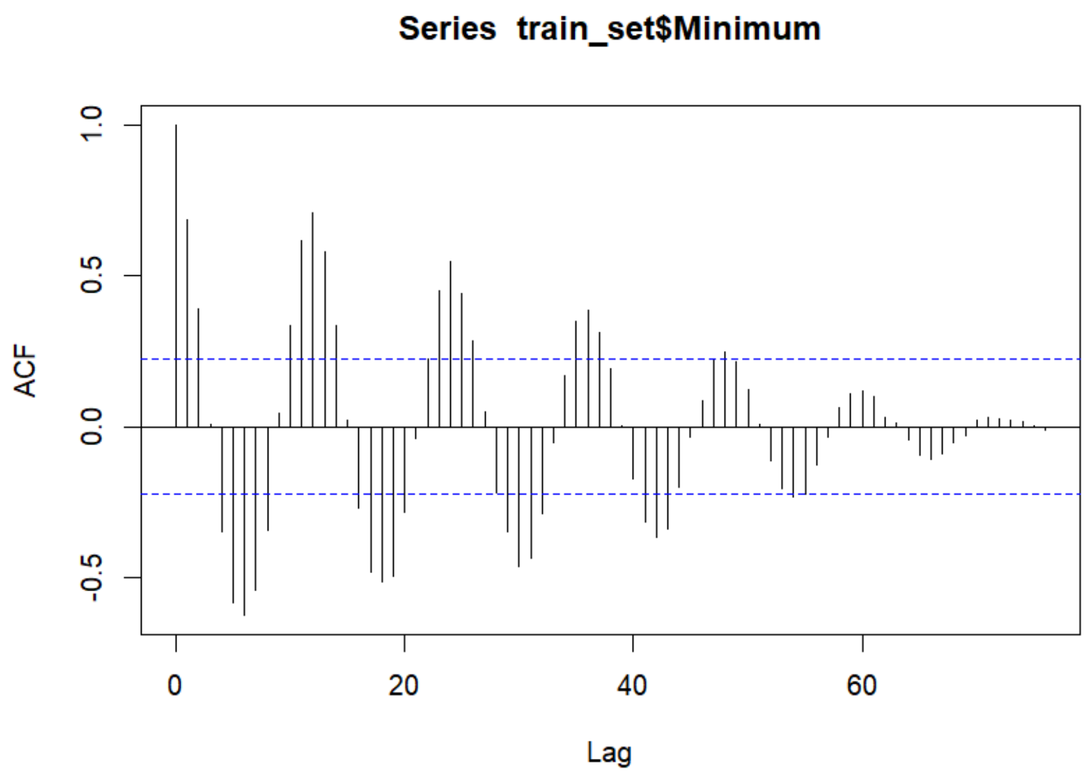
- The result form ADF test indicate that **p-value >0.1**.

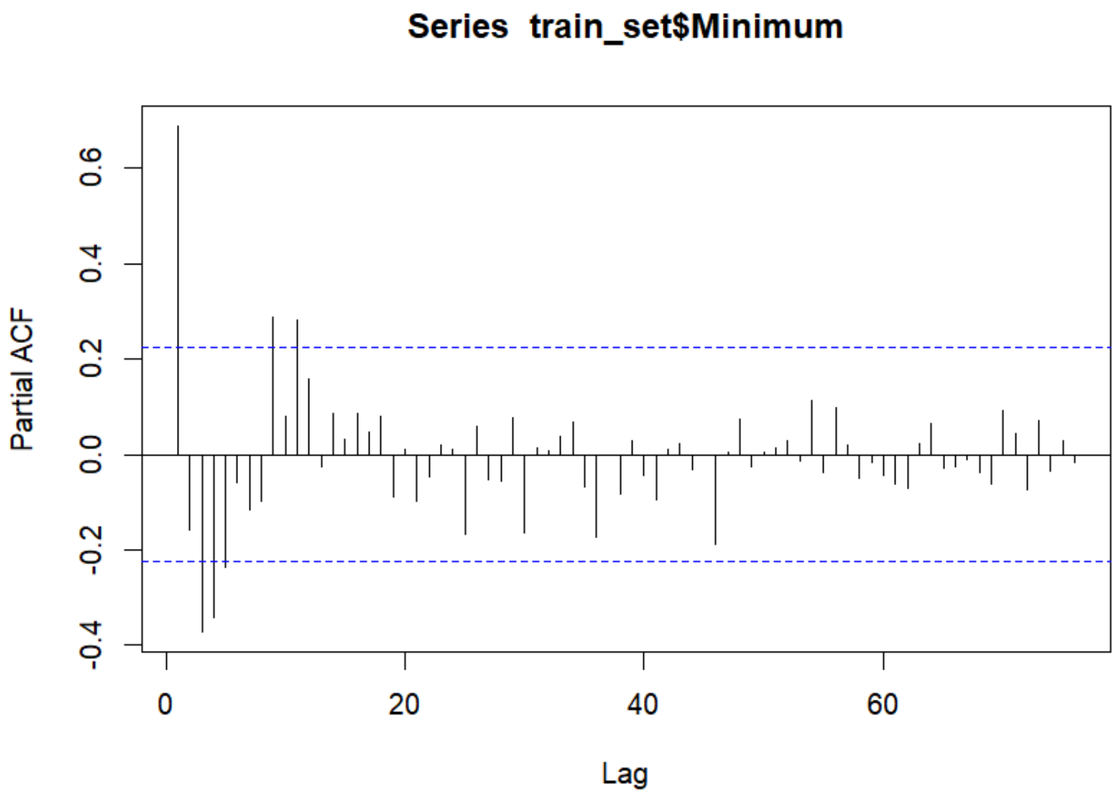
Since p-value is more than 0.05, we failed to reject the null Hypothesis (H_0) that the series is stationary.

- This suggest that the series is **stationary**.

SARIMA

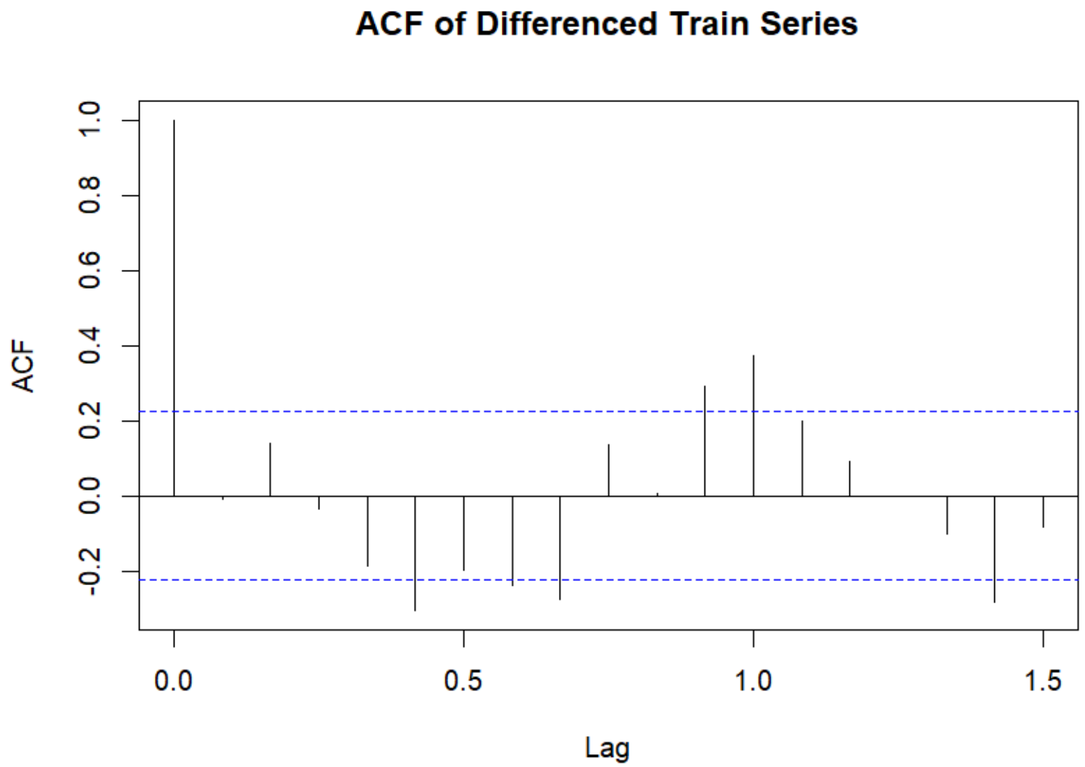
ACF & PACF plot

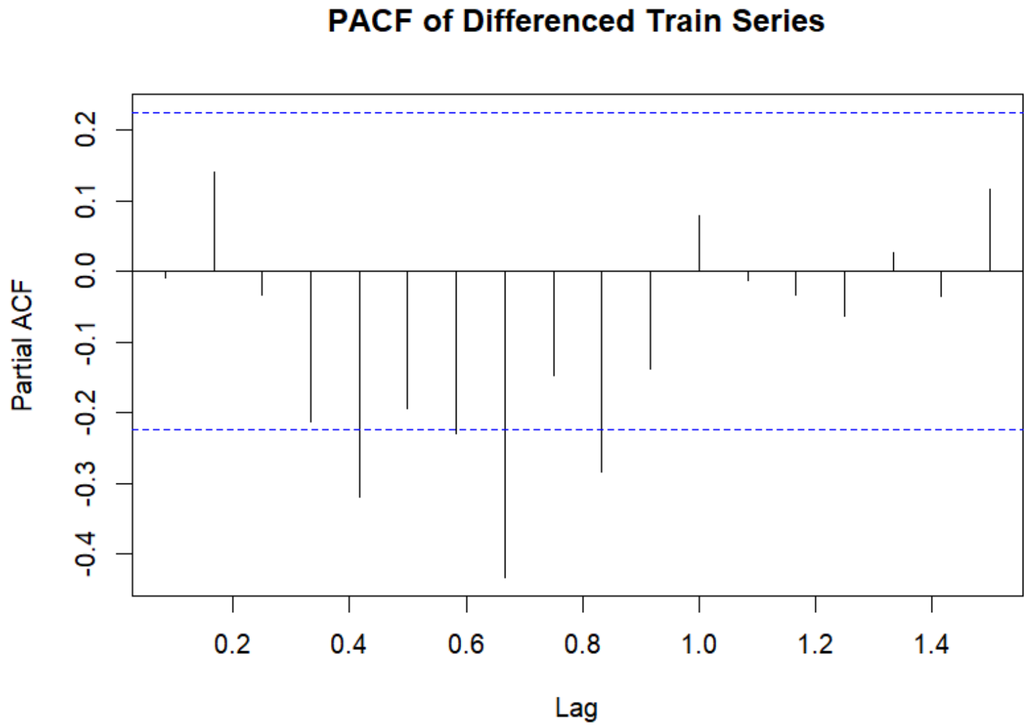




SARIMA

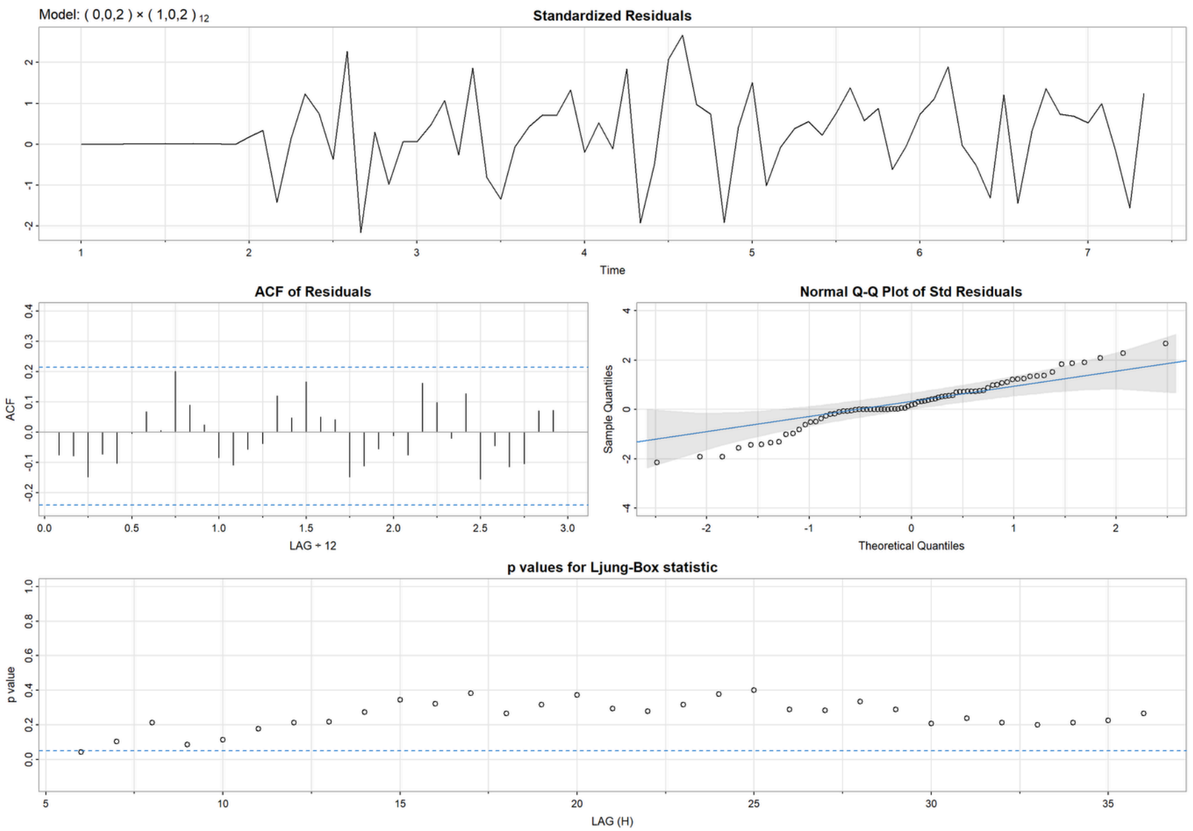
ACF & PACF plot after 1 st order differencing





SARIMA

Forecast using SARIMA (0, 0, 2) x (1, 0, 2) $_{12}$

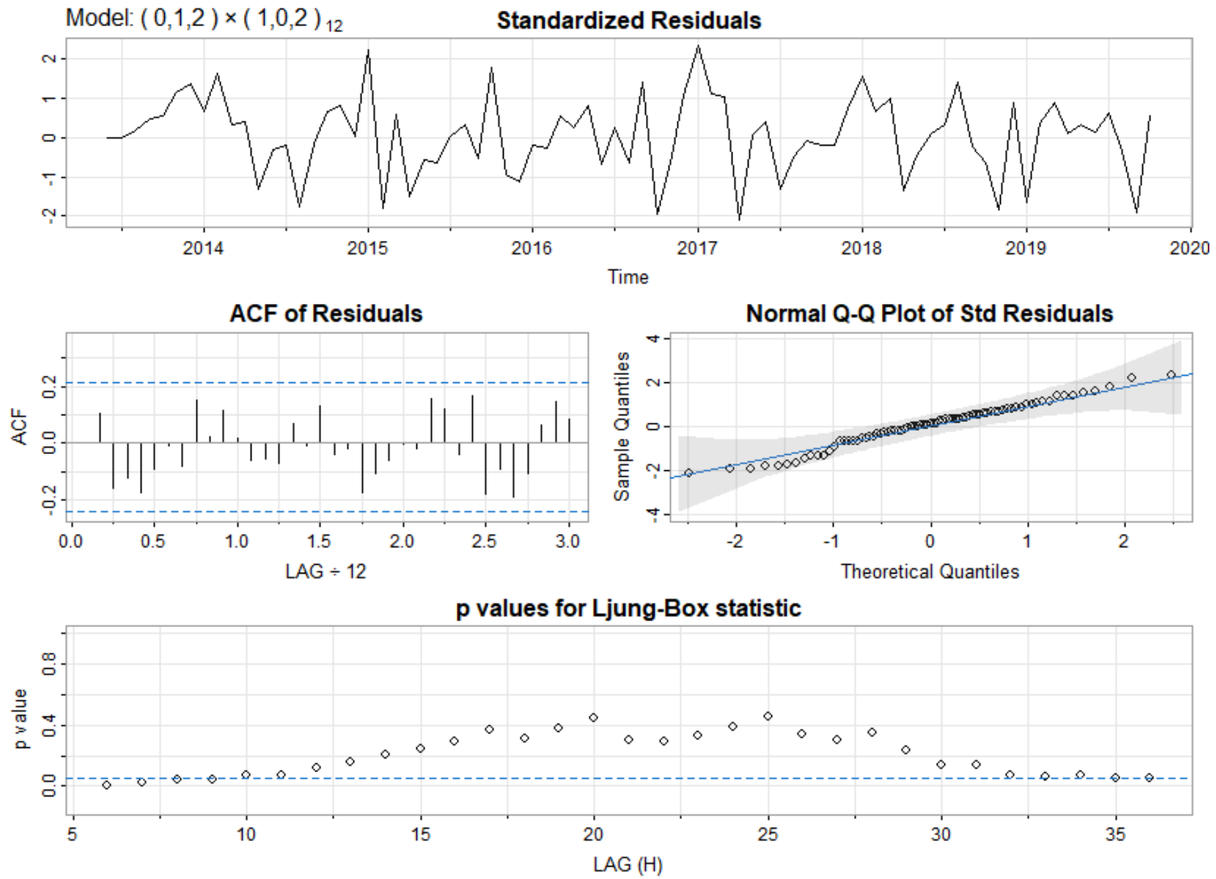



```
> mod1 = Arima(traints, order = c(0,0,2), seasonal = c(1,0,2))
> summary(mod1)
Series: traints
ARIMA(0,0,2)(1,0,2)[12] with non-zero mean

Coefficients:
         ma1     ma2  sar1     sma1    sma2    mean
      0.1432  0.2127     1  -0.5269  0.0693  9.9722
s.e.  0.1137  0.1218   NaN   0.1235  0.1652     NaN

sigma^2 = 92.95:  log likelihood = -289.25
AIC=592.5   AICc=594.12   BIC=608.9

Training set error measures:
                    ME     RMSE      MAE      MPE     MAPE      MASE         ACF1
Training set 2.356998 9.257605 6.872481 1.844197 17.42029 0.7507752 -0.07531462
```

1. From the standardized Residuals plot, the residuals fluctuate around 0 with no obvious pattern.
2. From ACF of Residuals plot, there are no significants lags, meaning the residuals behave like white noise.
3. The residuals mostly follow the theoretical quantile line.
4. The Ljung-Box p-values are mostly above 0.05.

SARIMA

Forecast using SARIMA (0, 1, 2) x (1, 0, 2) $_{12}$




```
Coefficients:
          ma1      ma2     sar1     sma1    sma2  constant
      -0.8840  -0.1160   0.9705  -0.7030  0.1579    0.2142
s.e.   0.1103   0.1098   0.0392   0.1721  0.2040    0.0580

sigma^2 estimated as 85.12:  log likelihood = -288.23,  aic = 590.45

$degrees_of_freedom
[1] 70

$ttable
         Estimate     SE t.value p.value
ma1       -0.8840 0.1103 -8.0147  0.0000
ma2       -0.1160 0.1098 -1.0562  0.2945
sar1       0.9705 0.0392 24.7465  0.0000
sma1      -0.7030 0.1721 -4.0859  0.0001
sma2       0.1579 0.2040  0.7741  0.4415
constant   0.2142 0.0580  3.6927  0.0004

$ICs
     AIC     AICc      BIC
7.769122 7.785140 7.983795
```

1. From the standardized Residuals plot, the residuals fluctuate around 0 with no obvious pattern.
2. From ACF of Residuals plot, there are no significants lags, meaning the residuals behave like white noise.
3. The residuals mostly follow the theoretical quantile line, but there are deviations in the tails.
4. The Ljung-Box p-values are mostly above 0.05.

SARIMA

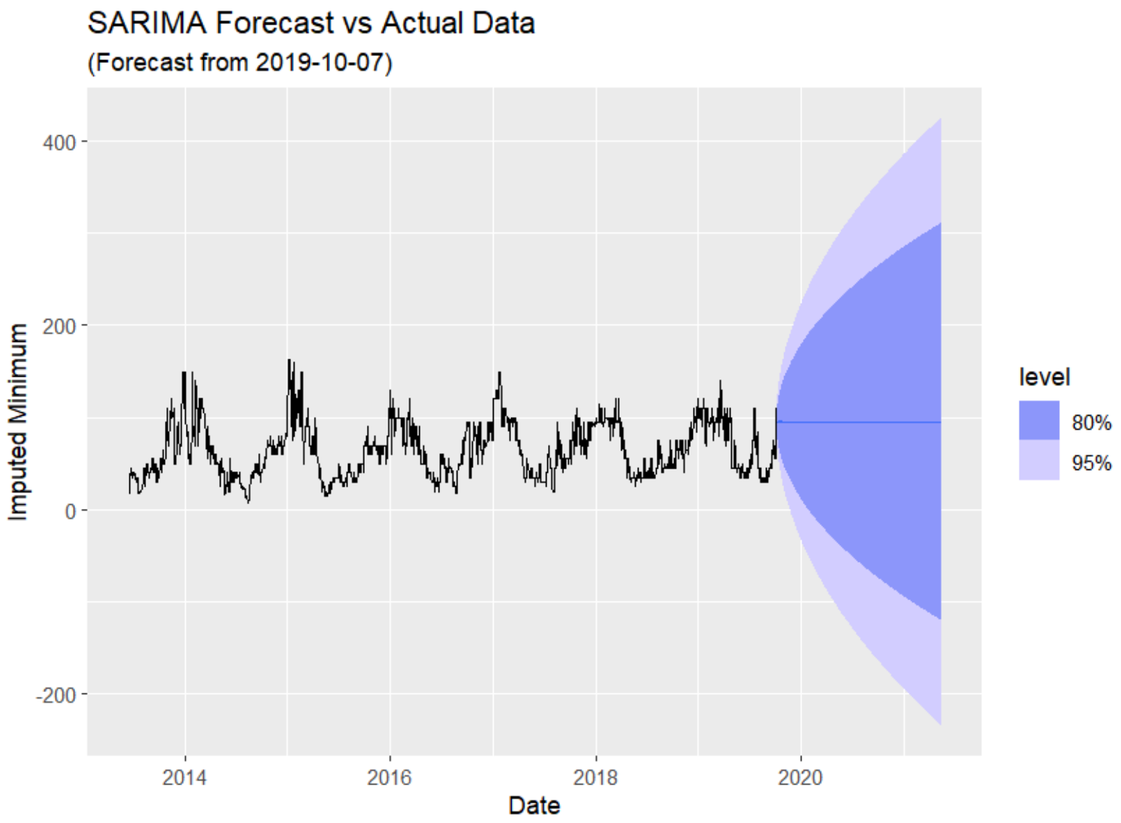
Forecast using SARIMA $(0, 1, 2) \times (1, 0, 2)_{12}$

```
        Box-Ljung test

data:  residuals_mod1
X-squared = 15.098, df = 20, p-value = 0.7708
```

SARIMA model's residuals are approximately independent, meaning the model has likely captured the underlying patterns in the data well.

SARIMA

Forecast using SARIMA $(0, 1, 2) \times (1, 0, 2)_{12}$

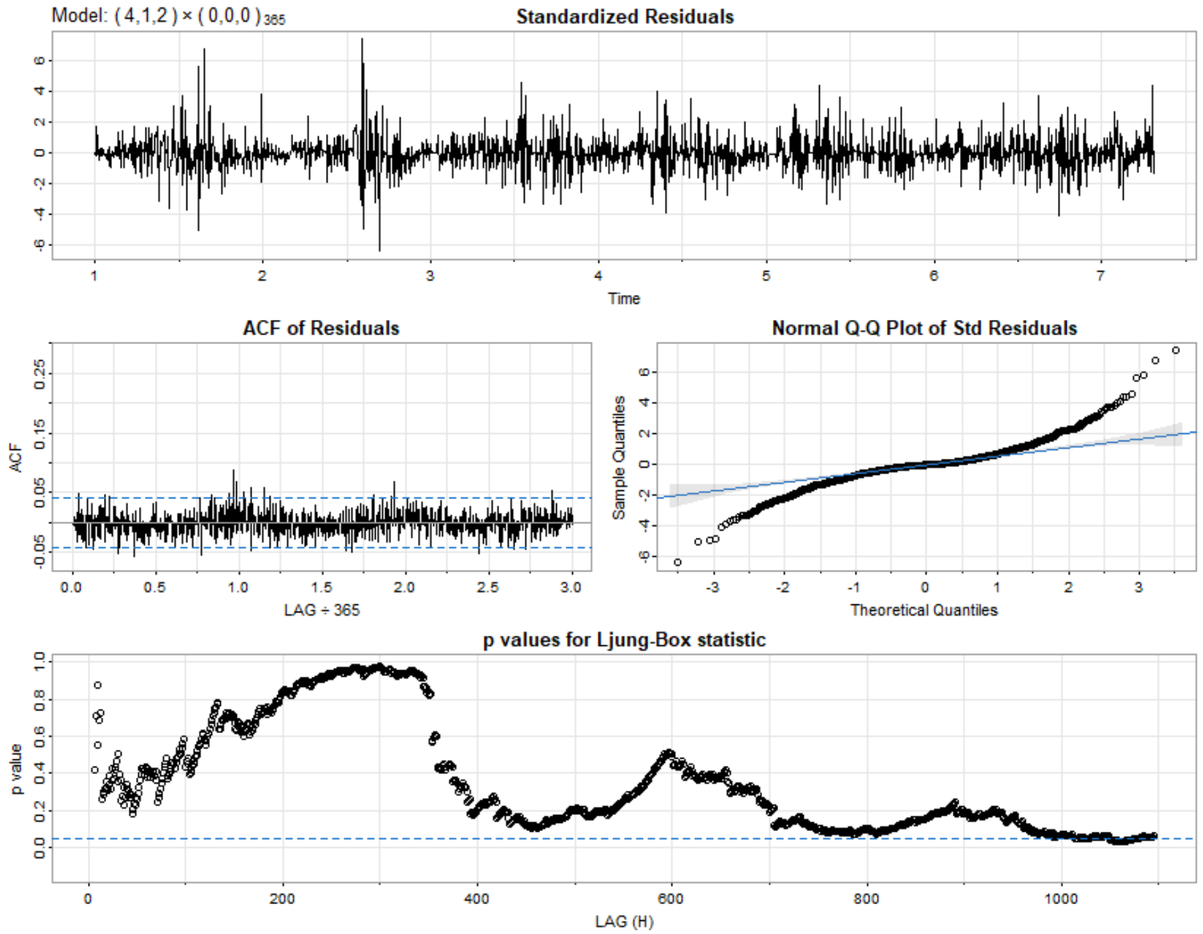



```
> # Print results4
> cat("MAE:", mae, "\n")
MAE: 10.26963
> cat("RMSE:", rmse, "\n")
RMSE: 12.20615
> cat("MAPE:", mape, "%\n")
MAPE: 32.86382 %
```

1. **MAE:** On average, the SARIMA forecast is off by approximately 10.27 units from the actual values.
2. **RMSE:** Penalizes larger errors more heavily compared to MAE.
3. **MAPE:** The forecast error is about 32.86% of the actual values on average, which is relatively high.

SARIMA

SARIMA $(4, 1, 2) \times (0, 0, 0)_{365}$

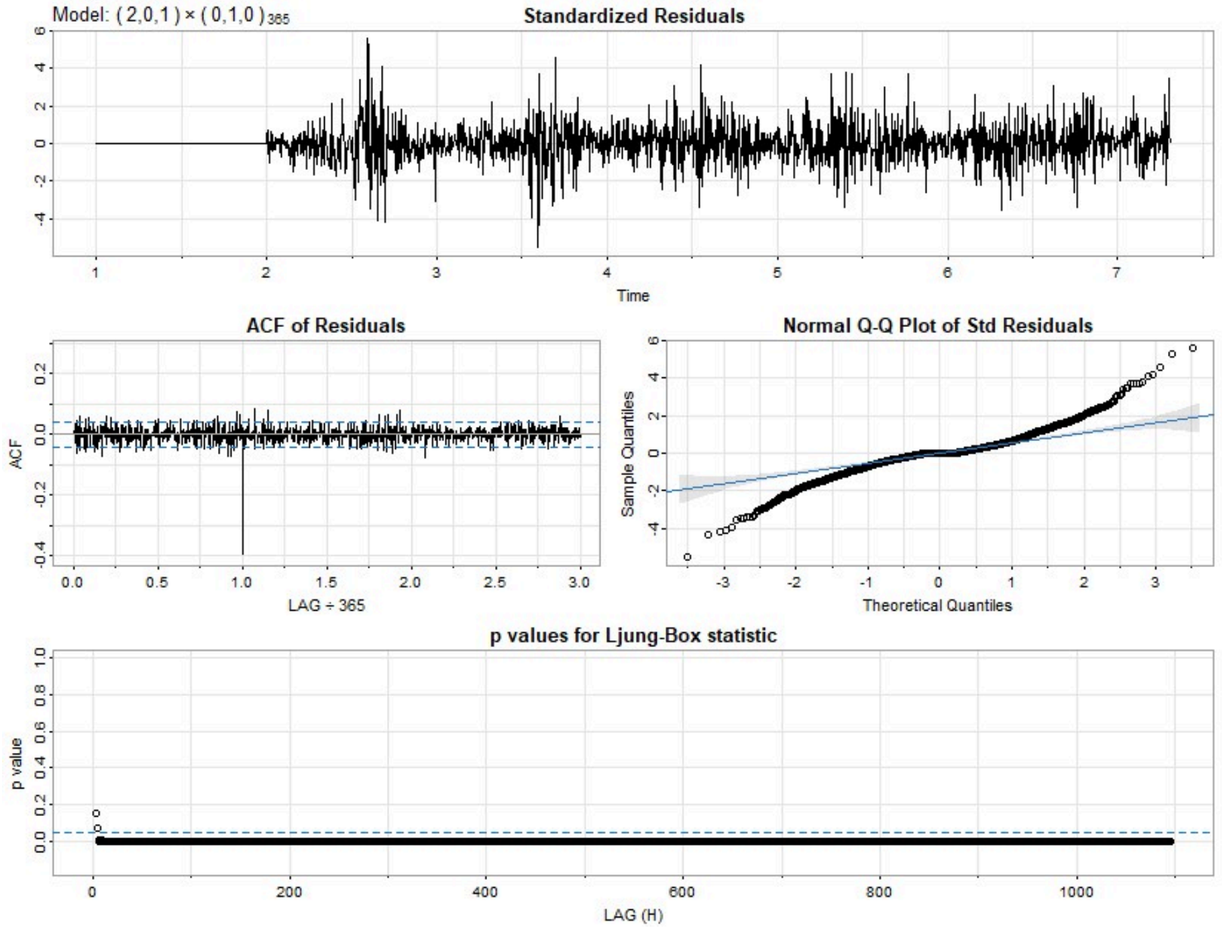



```
Coefficients:
          Estimate     SE t.value p.value
ar1        -0.1514 0.1214 -1.2469  0.2126
ar2         0.6687 0.0869  7.6956  0.0000
ar3         0.0983 0.0327  3.0059  0.0027
ar4        -0.0736 0.0227 -3.2479  0.0012
ma1        -0.1123 0.1203 -0.9338  0.3505
ma2        -0.6852 0.1116 -6.1425  0.0000
constant    0.0242 0.0830  0.2912  0.7709

sigma^2 estimated as 80.7156 on 2296 degrees of freedom

AIC = 7.23585  AICc = 7.235871  BIC = 7.255796
```

1. From the standardized Residuals plot, the residuals fluctuate around 0 with no obvious pattern.
2. From ACF of Residuals plot, there are no significants lags, meaning the residuals behave like white noise.
3. The residuals mostly follow the theoretical quantile line, but there are deviations in the tails.
4. The Ljung-Box p-values are mostly above 0.05.

SARIMA

SARIMA (2, 0, 1) x (0, 1, 0) $_{365}$

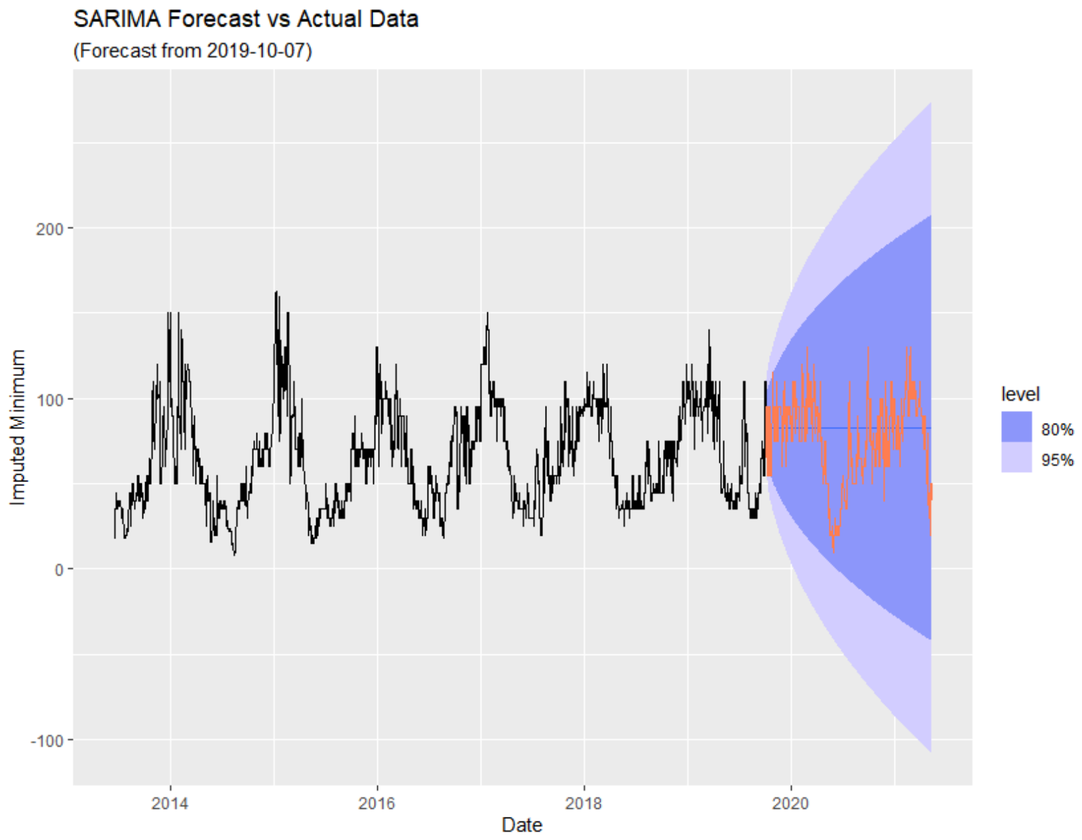



```
Series: train_ts
ARIMA(2,0,1)(0,1,0)[365]

Coefficients:
         ar1     ar2     ma1
      0.481  0.3314  0.1833
s.e.  0.091  0.0737  0.0946

sigma^2 = 150.3:  log likelihood = -7609.94
AIC=15227.89   AICc=15227.91   BIC=15250.17
```

1. From the standardized Residuals plot, the residuals fluctuate around 0 with no obvious pattern.
2. From ACF of Residuals plot, there are one significants lags.
3. The residuals mostly follow the theoretical quantile line, but there are deviations in the tails.
4. The Ljung-Box p-values are mostly under 0.05.

SARIMA

Model 1 : Forecast using SARIMA (2, 1, 2) x (0, 0, 0)$_{365}$



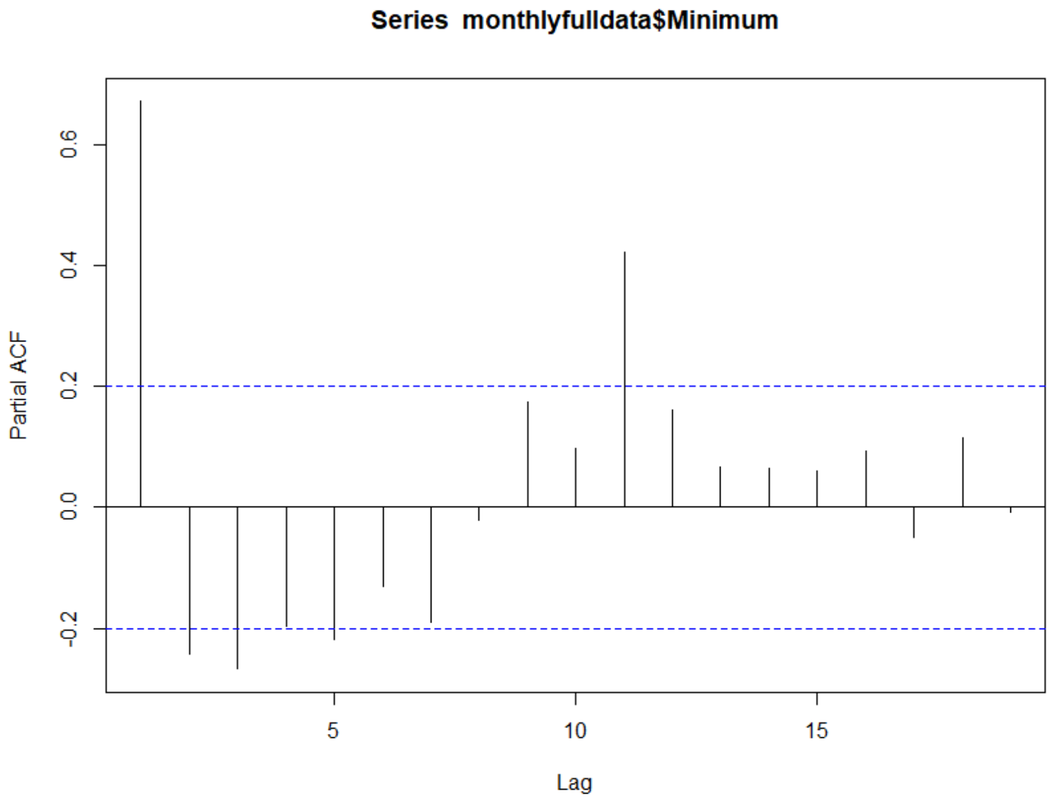

```
> cat("MAE:", mae, "\n")
MAE: 20.23601
> cat("RMSE:", rmse, "\n")
RMSE: 25.35773
> cat("MAPE:", mape, "%\n")
MAPE: 43.91877 %
```

1. MAE: On average, the SARIMA forecast is off by approximately 20.24 Euro from the actual values.
2. RMSE: Penalizes larger errors more heavily compared to MAE.
3. MAPE: The forecast error is about 43.92% of the actual values on average, which is relatively high.

MODEL2

ETS

Model 2: ETS (Monthly)



MODEL2

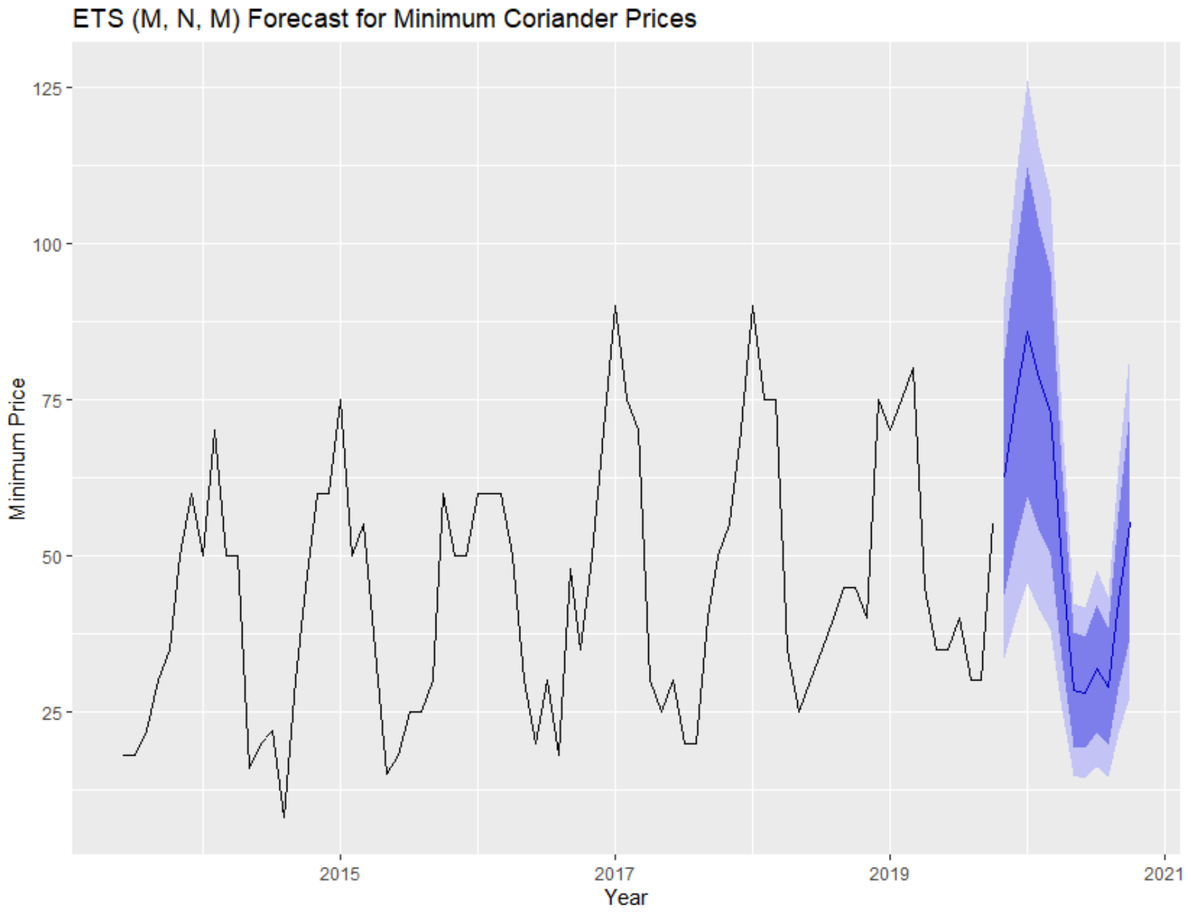
ETS

Model 2: ETS (Monthly)

```
> summary(manual_ets_model)
ETS(M,N,M)

Call:
ets(y = train_ts, model = "MNM")

  Smoothing parameters:
    alpha = 0.1236
    gamma = 1e-04

  Initial states:
    l = 41.8242
    s = 0.5367 0.9101 1.3699 1.4731 1.613 1.4103
           1.1749 1.0366 0.8058 0.5438 0.5995 0.5265

  sigma:  0.2357

     AIC     AICc      BIC
692.3304 700.1992 727.4875

Training set error measures:
                    ME     RMSE      MAE        MPE     MAPE      MASE        ACF1
Training set 1.4392 8.469181 6.797757 -2.434829 18.28993 0.7426121 0.03551126
```

MODEL2
ETS

Model 2: ETS (Monthly)




```
> # Print evaluation metrics
> cat("MAE:", mae, "\n")
MAE: 13.33233
> cat("RMSE:", rmse, "\n")
RMSE: 16.72321
> cat("MAPE:", mape, "%\n")
MAPE: 48.99434 %
```

MODEL2

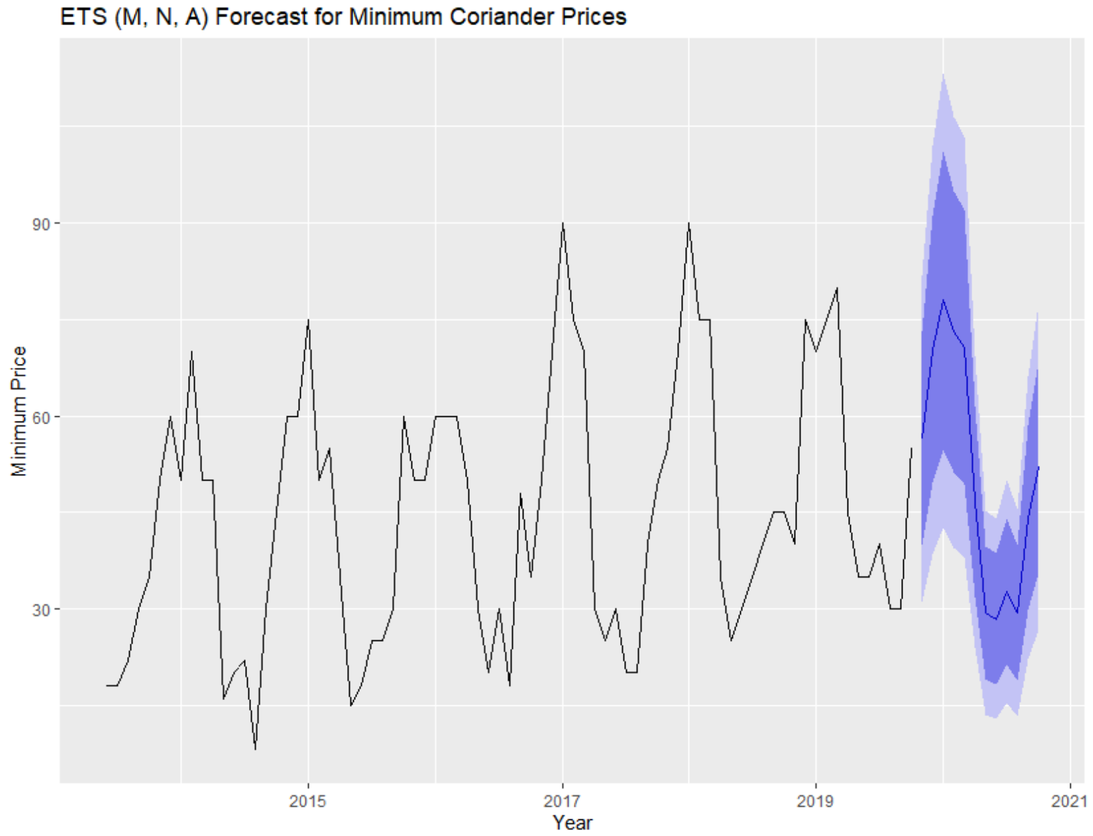
ETS

Model 2: ETS (Monthly)

```
ETS(M,N,A)

Call:
ets(y = train_ts, model = "MNA")

  Smoothing parameters:
    alpha = 0.1175
    gamma = 1e-04

  Initial states:
    l = 41.2693
    s = -21.5742 -4.1784 19.7787 22.1622 27.0716 19.3636
           5.5029 1.1241 -6.8618 -21.6029 -18.2308 -22.555

  sigma:  0.2296

     AIC     AICc      BIC
687.4437 695.3126 722.6008

Training set error measures:
                    ME     RMSE      MAE       MPE     MAPE      MASE         ACF1
Training set 1.066514 8.145093 6.449563 -1.448732 16.88829 0.7045741 -0.01568734
```

MODEL2
ETS

Model 2: ETS (Monthly)




```
> # Print evaluation metrics
> cat("MAE:", mae, "\n")
MAE: 12.935
> cat("RMSE:", rmse, "\n")
RMSE: 15.91249
> cat("MAPE:", mape, "%\n")
MAPE: 46.75677 %
```

MODEL2

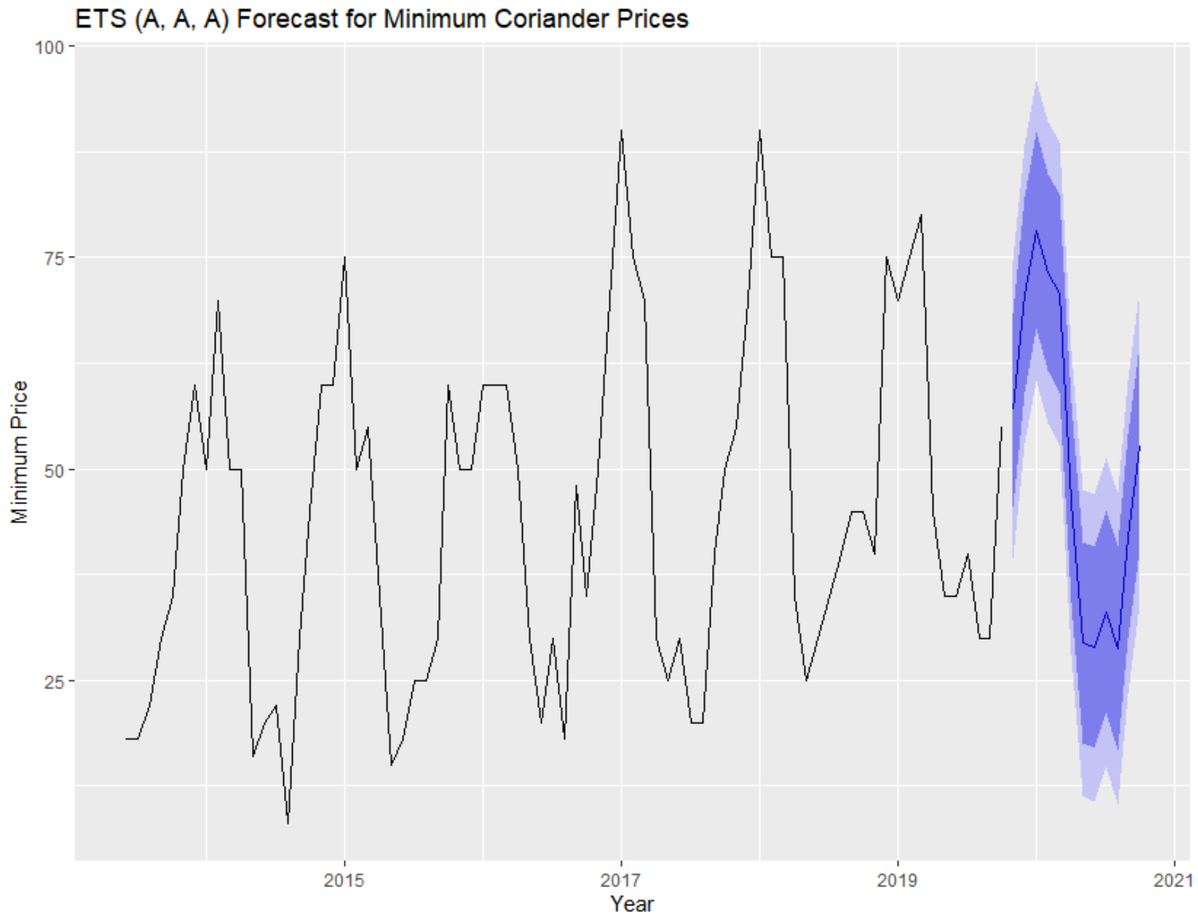
ETS

Model 2: ETS (Monthly)

```
ETS(A,A,A)

Call:
ets(y = train_ts, model = "AAA")

  Smoothing parameters:
    alpha = 1e-04
    beta  = 1e-04
    gamma = 1e-04

  Initial states:
    l = 38.8841
    b = 0.177
    s = -22.0456 -4.5804 19.7131 22.238 27.4701 19.4109
           5.4512 1.5061 -6.6719 -22.8449 -18.3949 -21.2514

  sigma:  8.6033

     AIC     AICc      BIC
681.9669 692.3398 721.8116

Training set error measures:
                     ME     RMSE      MAE       MPE     MAPE      MASE        ACF1
Training set -0.4637195 7.657415 6.075272 -5.345844 16.47812 0.6636852 -0.0090014
```

MODEL2
ETS

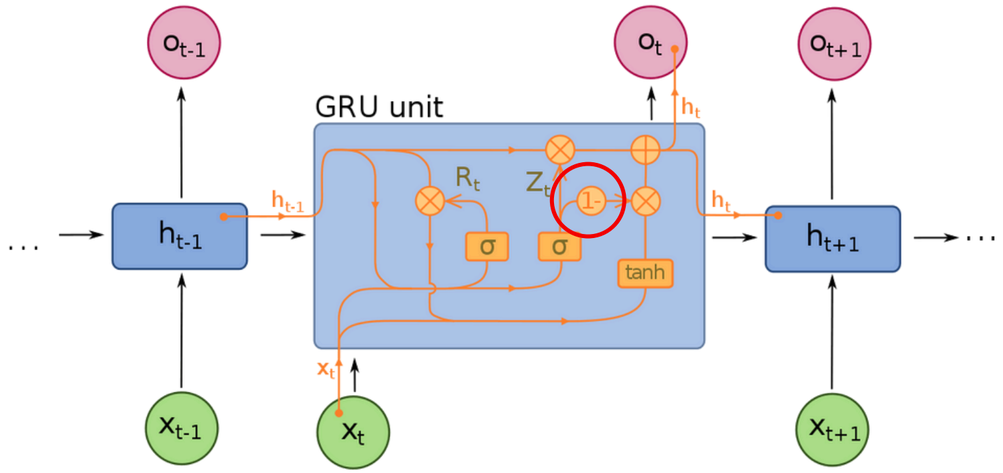
Model 2: ETS (Monthly)

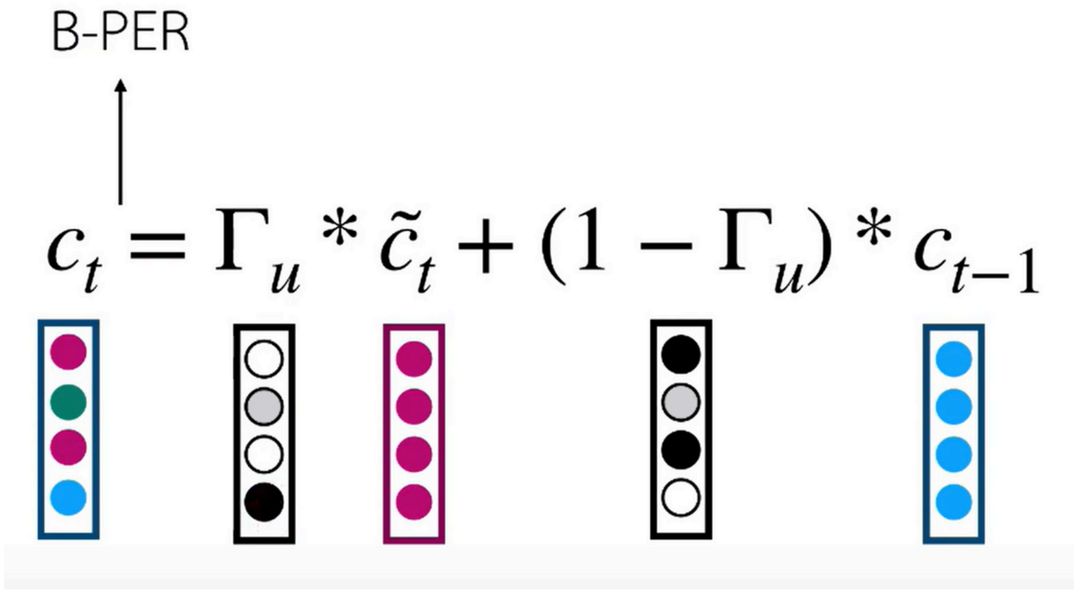



```
> # Print evaluation metrics
> cat("MAE:", mae, "\n")
MAE: 12.92536
> cat("RMSE:", rmse, "\n")
RMSE: 15.83527
> cat("MAPE:", mape, "%\n")
MAPE: 46.37578 %
```

GRU

What is GRU?

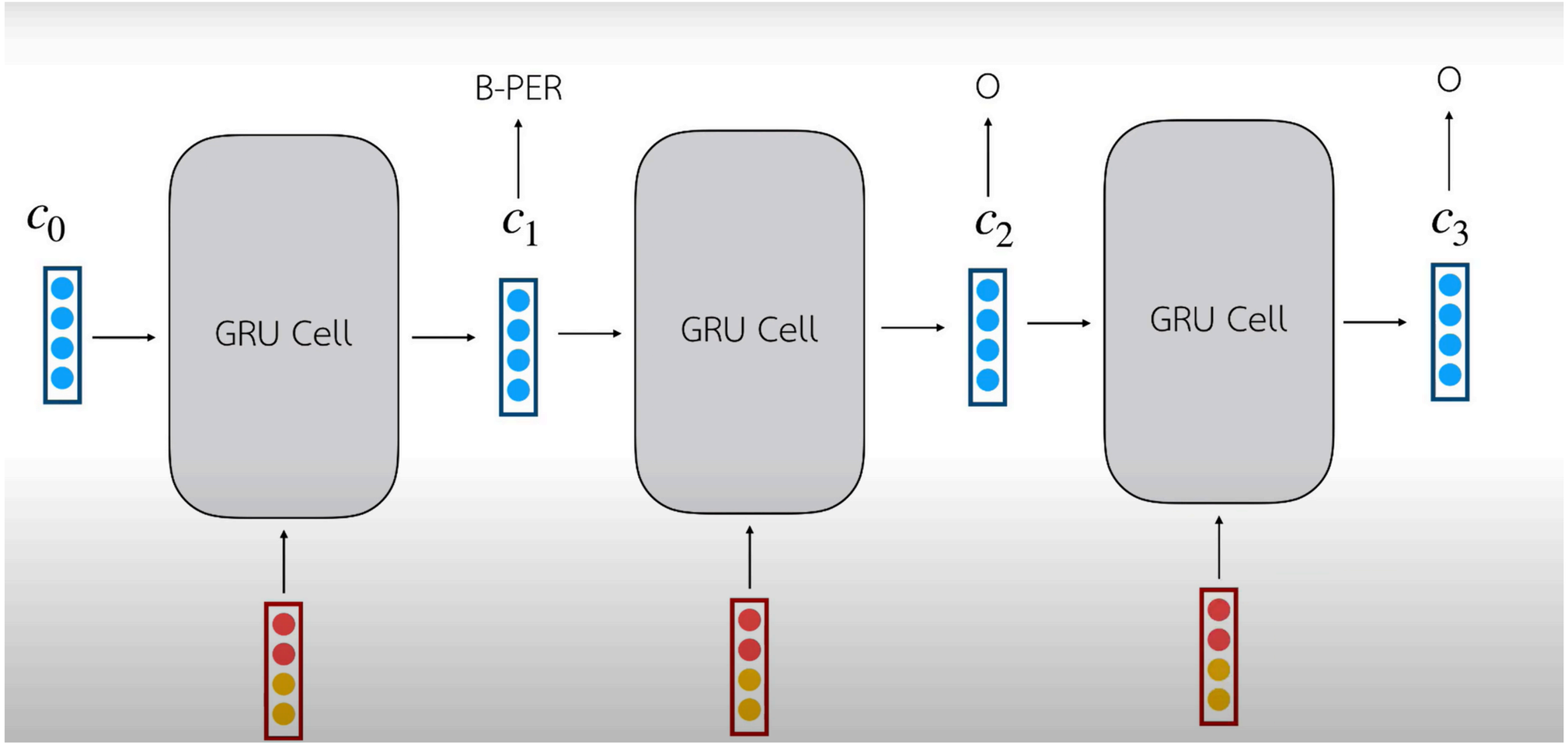




GRU (Gated Recurrent Unit) is a type of Recurrent Neural Network (RNN) designed to address some limitations of traditional RNNs, such as vanishing and exploding gradients.

$$\Gamma_u = \sigma(W_u \cdot [c_{t-1}; x_t] + b_u)$$

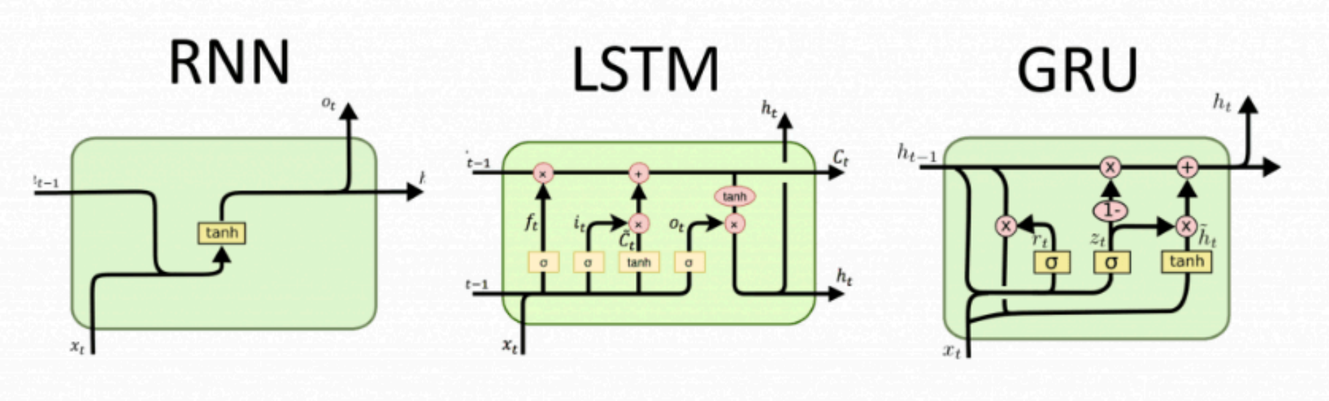
GRU

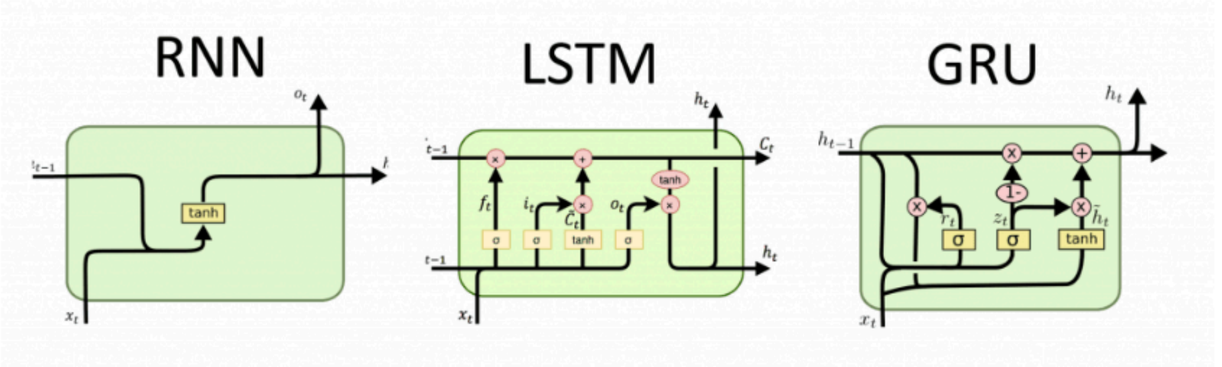


GRU

Why Choosing GRU?

1. **Simplicity and Efficiency:**
   - If computational resources or training time is a constraint, GRUs provide a simpler alternative to LSTMs without significant loss in performance.
2. **Performance with Smaller Datasets:**
   - GRUs often perform comparably to LSTMs for tasks with smaller datasets due to fewer trainable parameters.





B-PER

$$c_t = \Gamma_u * \tilde{c}_t + (1 - \Gamma_u) * c_{t-1}$$

MODEL3
GRU

Code implemetations: Normalize the Data

```
# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
train_scaled = scaler.fit_transform(train_data[['Imputed_Minimum']])
test_scaled = scaler.transform(test_data[['Imputed_Minimum']])
```

Normalization scales all values to a range (0, 1 in this case). This prevents large-scale values from dominating model training and helps gradient descent converge faster. The MinMaxScaler is fitted on the training data and applied to both training and test data to ensure consistent scaling.

Code implemetations: Create sequences

```
# Create sequences for supervised learning
def create_sequences(data, time_steps=365):
    x, y = [], []
    for i in range(len(data) - time_steps):
        x.append(data[i:i + time_steps])
        y.append(data[i + time_steps])
    return np.array(x), np.array(y)
```

```
#time_steps
time_steps= 365
```

- The GRU is designed for sequence data (e.g., time series). This function transforms the dataset into overlapping input-output pairs:
  - x: Sequences of time_steps length.
  - y: The value following each sequence (target for prediction).
- Our data is daily data with year trends, so the time_step = 365.

MODEL3
GRU

Code implemetations: Reshape Data

```
# Reshape data to match GRU input requirements
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

GRU layers in Keras expect input in the form of (samples, time_steps, features). Reshaping ensures that the data matches this structure, where: 1. samples: Number of sequences.

2. time_steps: Number of steps in each sequence.
3. features: Number of features (1 in this case, for a univariate time series).

Code implemetations: Build and Compile the GRU Model

```
# Build the GRU model
model = Sequential([
    GRU(units=30, activation='relu', input_shape=(time_steps, 1)),
    Dropout(0.2),  # Dropout with 20% probability
    Dense(units=1)
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
```

- The GRU layer captures time dependencies in the data.
- Dropout helps prevent overfitting by randomly deactivating 20% of neurons during training.
- The Dense layer outputs a single prediction for the next time step.
- The Adam optimizer is efficient for non-stationary objectives, and MSE is used to measure prediction errors.

MODEL3
GRU

Code implemetations: Train model

```
# Set up Early Stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the model
history = model.fit(
    X_train, y_train,
    epochs=30,
    batch_size=32,
    callbacks=[early_stopping]
)
```

Training stops after 30 epochs or when the early stopping condition is met (val_loss improvement halts for 5 epochs) to avoid overfitting.

MODEL3
GRU

Code implemetations: Reshape Data

```
# Evaluate the model
loss = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")
```

Evaluating the model on the test dataset gives an unbiased estimate of performance using the loss function (MSE).

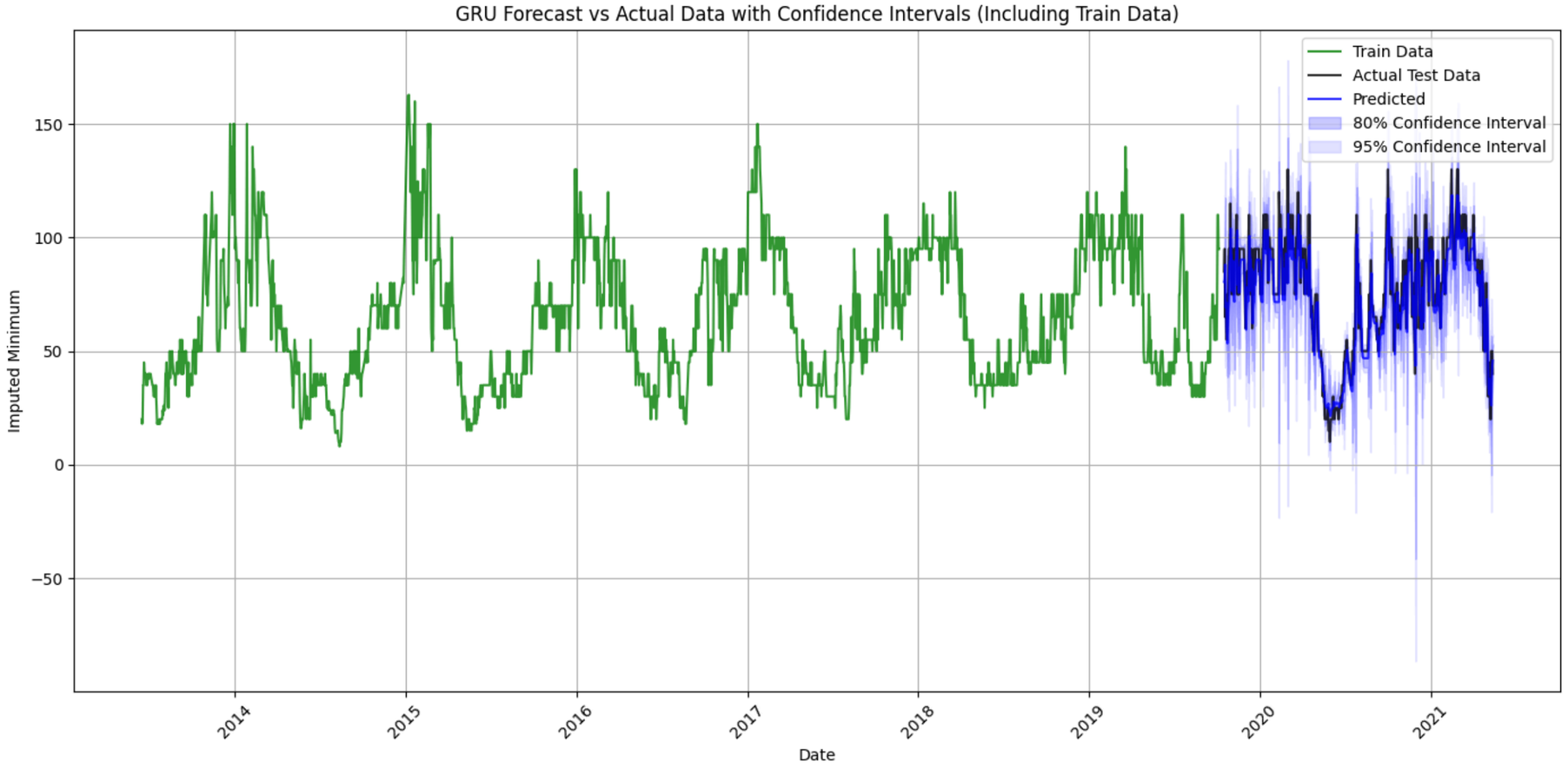
Code implemetations: Make Predictions and Rescale Data

```
# Make predictions
predictions = model.predict(X_test)

# Rescale predictions and actual values back to original scale
predictions_rescaled = scaler.inverse_transform(predictions)
y_test_rescaled = scaler.inverse_transform(y_test.reshape(-1, 1))
```

Predictions and actual values are in the normalized scale (0 to 1). To interpret the results in the original context, they must be rescaled back to their original range using the inverse transform of the MinMaxScaler.

GRU

Model 3 : RNN



GRU

Visual analysis

- From the graph, the "Actual" and "Predicted" lines closely overlap, indicating the model performs well at capturing trends in the data.
- Some divergence exists, but it is generally consistent, which is supported by the relatively low MAE and MAPE values.

Model Performance Metrics

```
Mean Absolute Error (MAE): 7.357300868892848
Mean Squared Error (MSE): 110.43163530327918
Root Mean Squared Error (RMSE): 10.508645740687959
Mean Absolute Percentage Error (MAPE): 10.66%
R-squared (R2): 0.8236969265410269
```

1. **MAE:** On average, the forecast is off by approximately 7.36 units from the actual values.
2. **RMSE:** Representing typical prediction error. It suggests the prediction error is around 10.51 units.
3. **MAPE:** Reflects prediction accuracy as a percentage of the actual values, with an error rate of about 10.6%
4. .**R-squared:** An R^2 value of ~82.4% shows strong predictive performance.

MODELING EVALUATION

MODEL COMPARISON

SARIMA $(0, 1, 2) \times (1, 0, 2)_{12}$	**MAE : 10.27** **RMSE : 12.21** **MAPE : 32.86%**
ETS (A,N,A)	**MAE : 12.92** **RMSE : 15.83** **MAPE : 46.375%**
GRU	**MAE : 7.36** **RMSE : 10.51** **MAPE : 10.66%**

Summary :

After doing comparison with 3 models on the testing set, The best model is GRU with superior performance on all model evaluation metrics

LIMITATIONS AND CONCLUSION

Limitations

SARIMA

PERIOD CAN'T EXCEED 350

ETS

PERIOD CAN'T EXCEED 24

GRU

INCOMPATIBLE ENVIRONMENT BETWEEN
KERAS AND TENSORFLOW IN R

Conclusion

The findings enhance understanding of historical price trends and support decision-making, promoting long-term sustainability in Nepal's agriculture industry.