ST422 Assignment

23074

December 08, 2020

Contents

1 Q1 & Q2

1.1 Sunshine series

1.1.1 Preliminary analysis

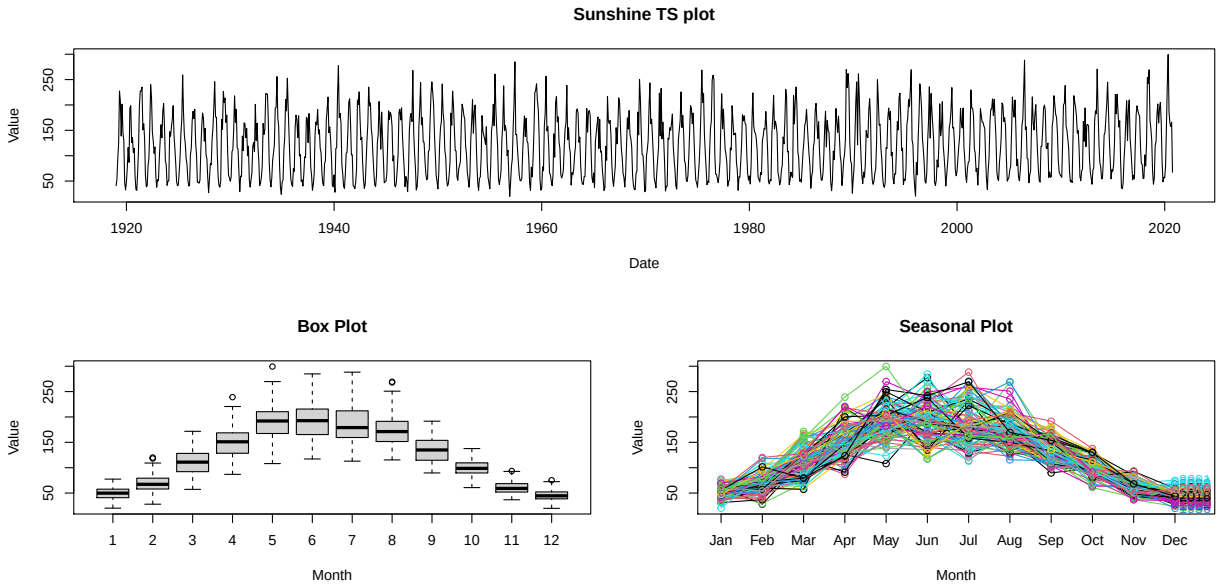


Figure 1: Preliminary analysis on entire Sunshine series

By comparing the preliminary analysis based on the entire dataset (Figure 1) and partial dataset (Figure 3), using the entire dataset is a bit of stretch and there is no major difference in terms of the unique patterns and trends observed between these two datasets. Hence, for better intuitive understanding on analysis, the preliminary analysis is conducted on the partial dataset (21th century) but the final modelling will be using the entire population.

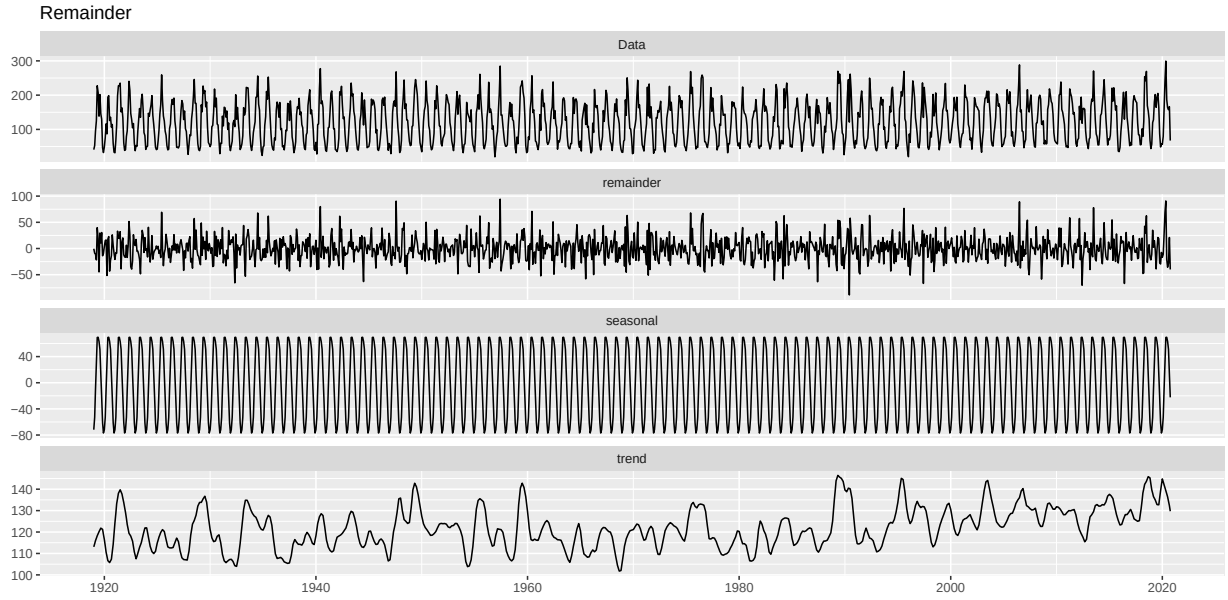


Figure 2: Decomposition analysis on entire Sunshine series

Based on preliminary analysis plots on **Sunshine** series, following observations are made:

1. Monthly time-series plot shows high fluctuation but no visible trend nor increasing variance.
2. Box plots and seasonal plot indicate a definite existence of seasonality.
3. Decomposition analysis provide some additional insights that the trend cycle and the seasonal plot shows there's seasonal fluctuation occurred with no specific trend and fairly random remainder/residual.

Conducted analysis indicates that sunshine peaks during summer (around July) and reaches the lowest point during winter (around Dec). Which is a common climate condition in countries with 4 distinct seasons (e.g. UK, Korea and Japan).

Given this observed seasonality in time series, I further delve into the existence of seasonal persistence and the time series model identification with ACF & PACF plots.

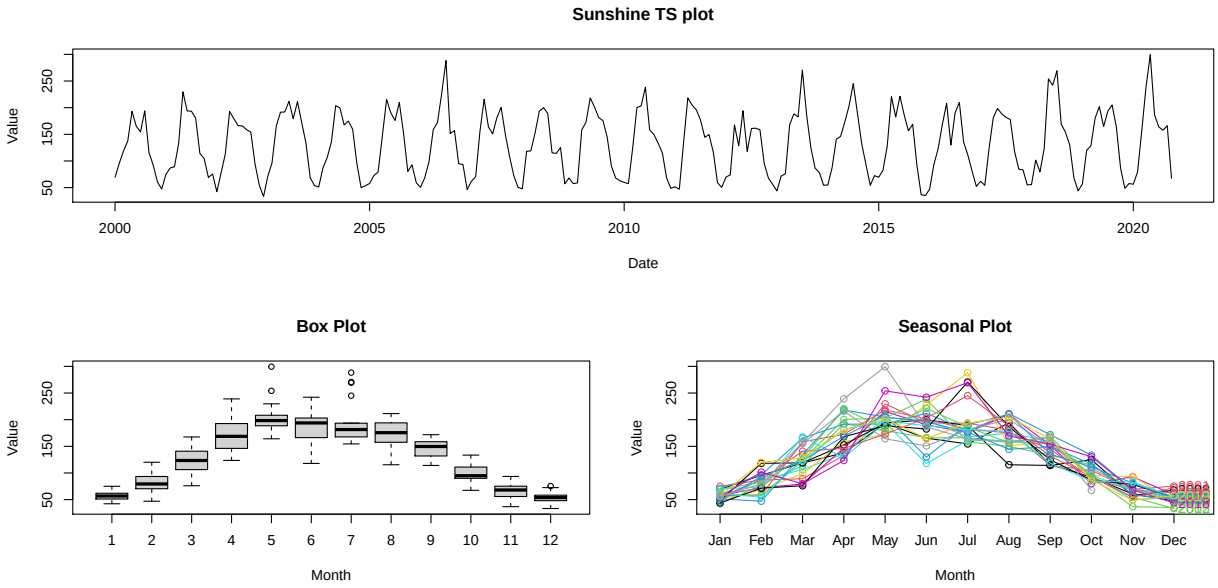


Figure 3: Preliminary analysis on trimmed Sunshine series

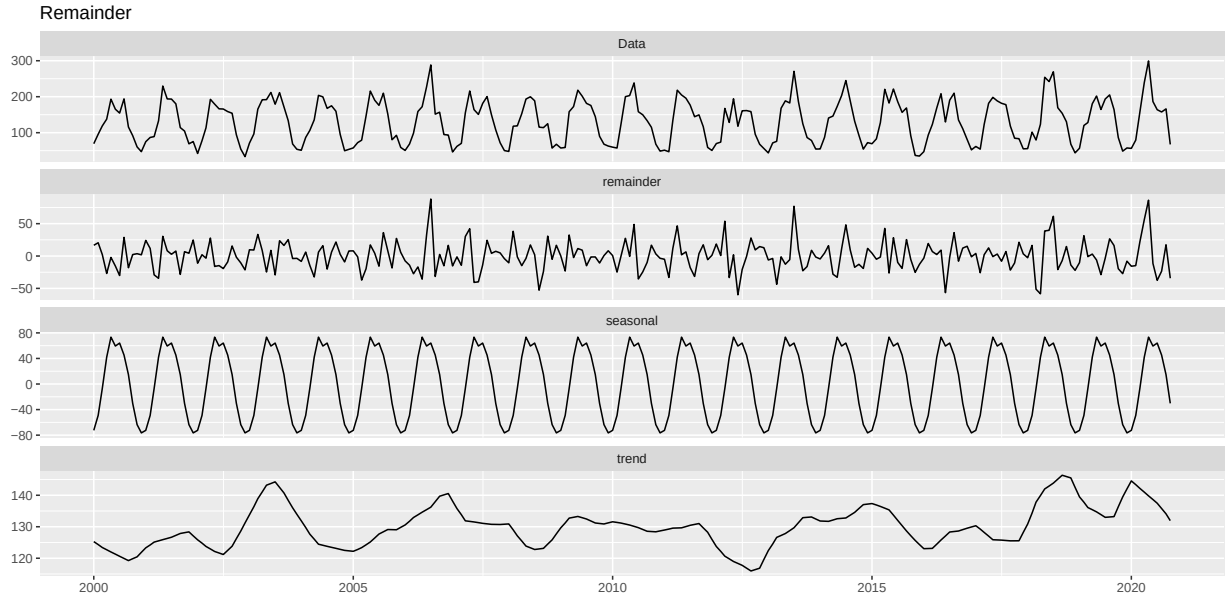


Figure 4: Decomposition analysis on trimmed Sunshine series

1.1.2 ACF, PACF and stationarity analysis

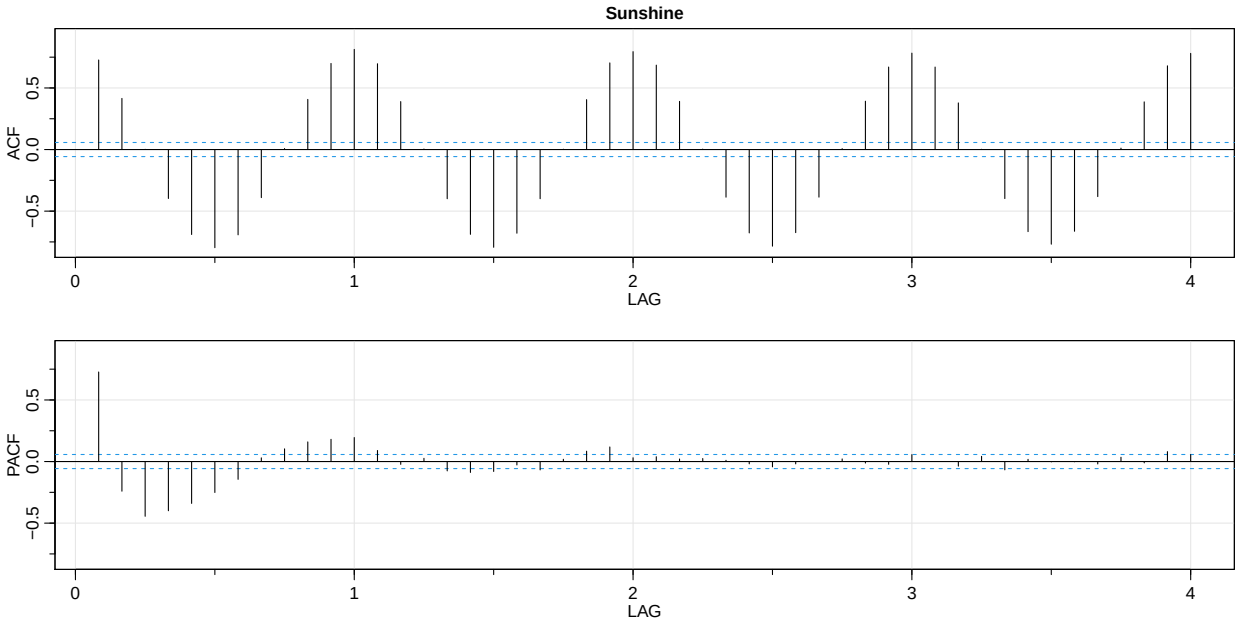


Figure 5: ACF & PACF plots of Sunshine series

```
##
##  Augmented Dickey-Fuller Test
##
## data:  ts_v
## Dickey-Fuller = -7.1097, Lag order = 12, p-value = 0.01
## alternative hypothesis: stationary
```

Prior to ACF and PACF assessment, there is one of the most important conditions required for time-series analysis that I need to check, stationarity. Augmented Dickey-Fuller test is the most commonly used test method to attest the stationarity of series. Based on the conducted analysis, I could reject the null hypothesis at 5% significance level that the series is stationary.

In addition, based on ACF and PACF plots of Sunshine series, I can verify the existence of seasonal persistence around 12 every month and can gain further insights and draw the blueprint of our model specification (Figure 5):

1. ACF cuts off at lag 2 and PACF tails off -> MA(2)
2. Other model suggestion & exploration with modifying AR and MA component by ± 1

Based on seasonal pattern observed in the preliminary analysis, consideration of a multiplicative seasonal ARIMA model is reasonable. our variable of interest, monthly sunshine x_t as being modeled as:

$$x_t = S_t + w_t$$

where S_t is a seasonal component that varies a little from one year to the next.

As a next step, 12 month differencing is going to be examined to find a roughly stationary series and then find a multiplicative seasonal ARIMA to fit the resulting residual series.

1.1.3 ∇_{12} series analysis

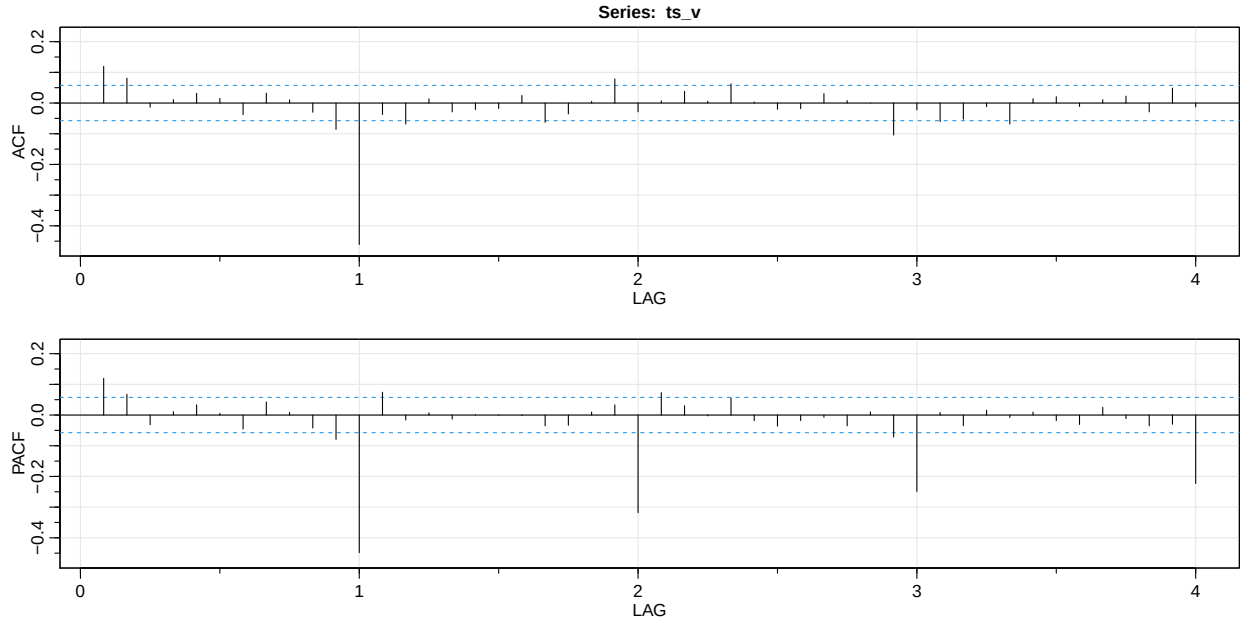


Figure 6: ACF & PACF plots of seasonally differenced Sunshine series

```
##
##  Augmented Dickey-Fuller Test
##
## data:  ts_v
## Dickey-Fuller = -14.374, Lag order = 12, p-value = 0.01
## alternative hypothesis: stationary
```

The resulting ACF and PACF possibly indicate that (Figure 6):

1. ACF cuts off at lag 1 and PACF decays quickly -> MA(1)
2. Other model suggestion & exploration with modifying AR and MA component by ± 1

1.1.4 Model fitting using SARIMA

In this section, I fit the optimal parameters of the time-series model based on insights gained from previous analysis and extend the scope with other models based on the suggestion with modifying AR and MA component by ± 1. For model exploration, the exhaustive approach (i.e. grid-search) will be used to calculate the goodness of fit of a possible set models and rank them based on the performance (i.e. AIC). Furthermore, an additional candidate model using auto.arima() function from the third party R package is going to be examined as well. Hence, total 3 different specifications will be compared:

1. $ARMA(0,0,2) \times (0,1,1)_{12}$ <- Best guess model based on preliminary analysis.
2. Grid-search model
3. auto.arima() model

Initial non-seasonal & seasonal parameters for grid-search approach are derived from Model 1: $ARMA(0,0,2) \times (0,1,1)_{12}$.

1.1.5 Model diagnosis and final model selection

```
## Series: ss_ts
## ARIMA(0,0,2)(0,1,1)[12]
##
## Coefficients:
##          ma1     ma2     sma1
##       0.1315  0.0627  -0.9533
## s.e.  0.0288  0.0282   0.0107
##
## sigma^2 estimated as 661.5:  log likelihood=-5658.96
## AIC=11325.91   AICc=11325.95   BIC=11346.31
```

```
## Series: ss_ts
## ARIMA(0,0,2)(0,1,1)[12]
##
## Coefficients:
##          ma1     ma2     sma1
##       0.1315  0.0627  -0.9533
## s.e.  0.0288  0.0282   0.0107
##
## sigma^2 estimated as 661.5:  log likelihood=-5658.96
## AIC=11325.91   AICc=11325.95   BIC=11346.31
```

```
## Series: ss_ts
## ARIMA(0,0,2)(2,1,0)[12]
##
## Coefficients:
##          ma1     ma2     sar1     sar2
##       0.1410  0.0839  -0.6171  -0.3370
## s.e.  0.0289  0.0284   0.0273   0.0277
##
## sigma^2 estimated as 868.5:  log likelihood=-5811.72
## AIC=11633.43   AICc=11633.48   BIC=11658.93
```

Resulted 3 different models can be found below:

1. $ARMA(0,0,2) \times (0,1,1)_{12}$ <- Model(1) Best guess
2. $ARMA(0,0,2) \times (0,1,1)_{12}$ <- Model(2) Grid-search
3. $ARMA(0,0,2) \times (2,1,0)_{12}$ <- Model(3) auto.arima

Based on the goodness of fit metrics (i.e. AIC) of 3 different models, Model(1) and Model(2) are identical and provide better model-fit results compared to Model(3). Hence, the final model is: $ARMA(1,0,0) \times (0,1,1)_{12}$.

```r
# Residual analysis
checkresiduals(best_guess_ss)
```

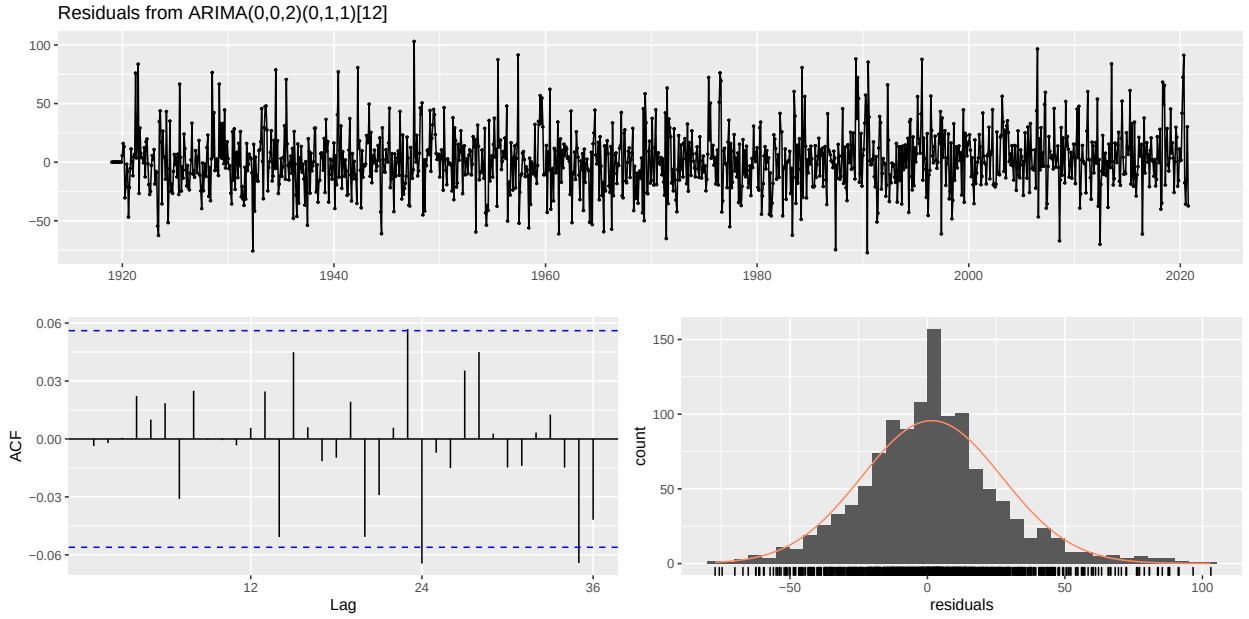


Figure 7: Residual diagnosis

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,0,2)(0,1,1)[12]
## Q* = 23.931, df = 21, p-value = 0.2964
##
## Model df: 3.   Total lags used: 24
```

```r
tsdiag(best_guess_ss)
```

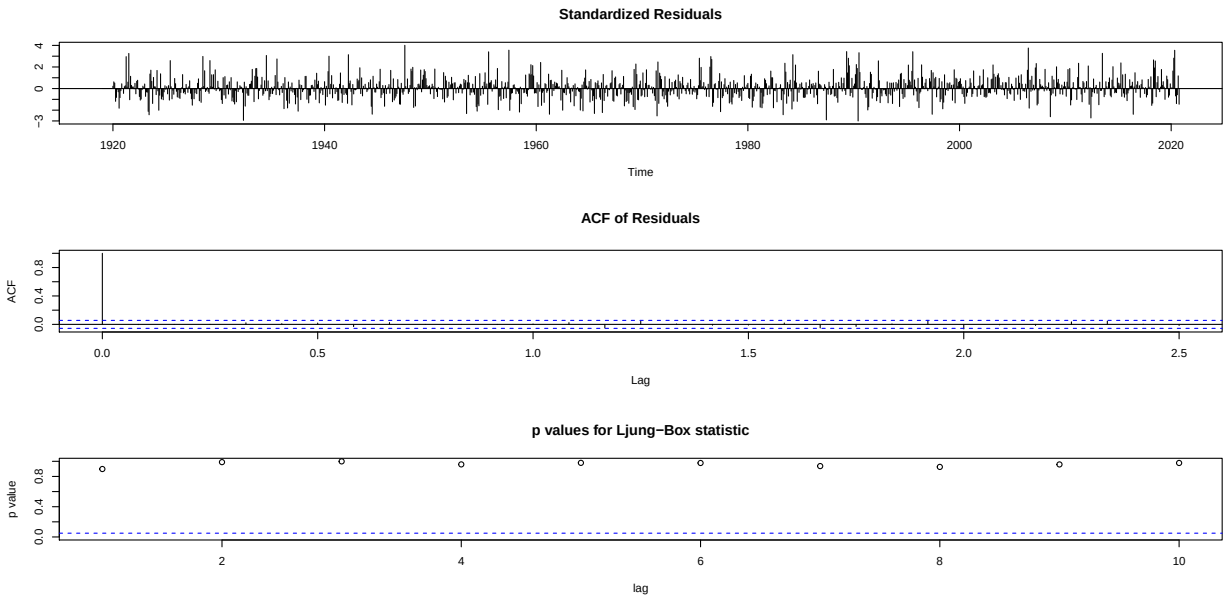


Figure 8: Residual diagnosis

Based on Figure (7) & (8), the Ljung-Box test show no autocorrelation on residuals and no seasonality observed on the ACF plot of model residuals. Hence, it is concluded that the final model is appropriate for forecasting since its residuals show white noise behavior and uncorrelated against each other. As a next step, the forecasting of the final model is going to be examined.

1.1.6 Model forecasting

As a finale, I plot 2 months forecasting plot (Figure 9) and 24 months (Figure 10) forecast plot to inspect the behaviour of estimated values.

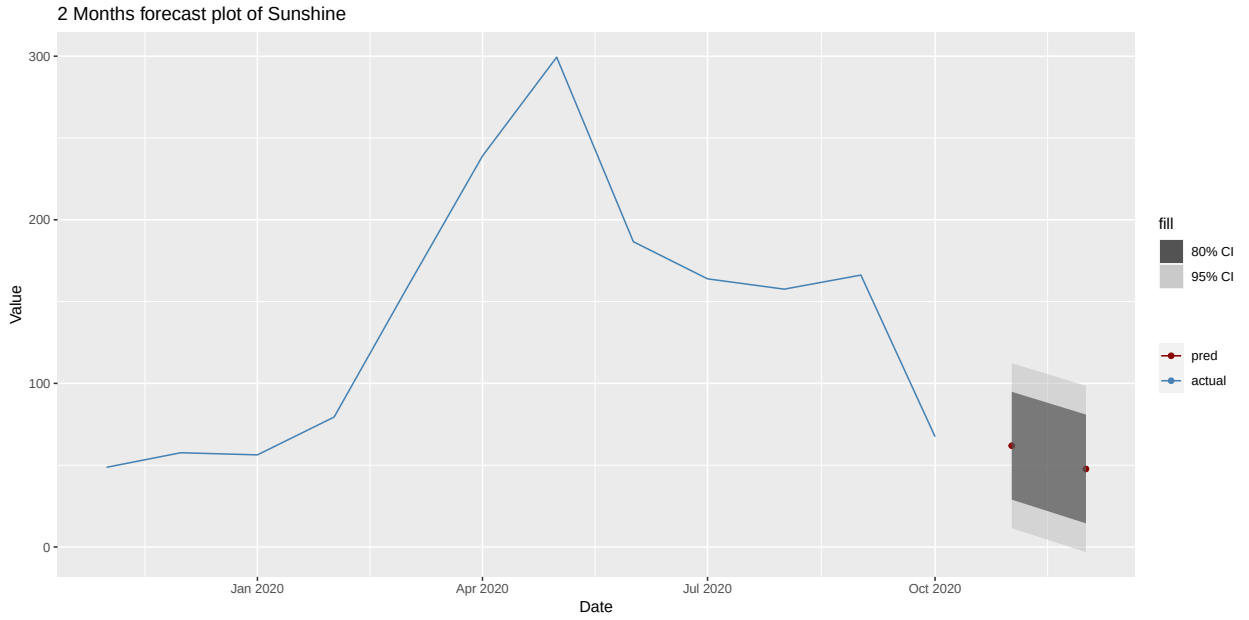


Figure 9: 2 Month forecasting

```
##          Point Forecast    Lo 80    Hi 80     Lo 95     Hi 95
## Nov 2020       61.89698 28.93614 94.85783 11.487717 112.30625
## Dec 2020       47.69690 14.45247 80.94134 -3.146075  98.53988
```

Based on forecast visual plots, I can conclude that the final model is following a general trend without any observed outlines.

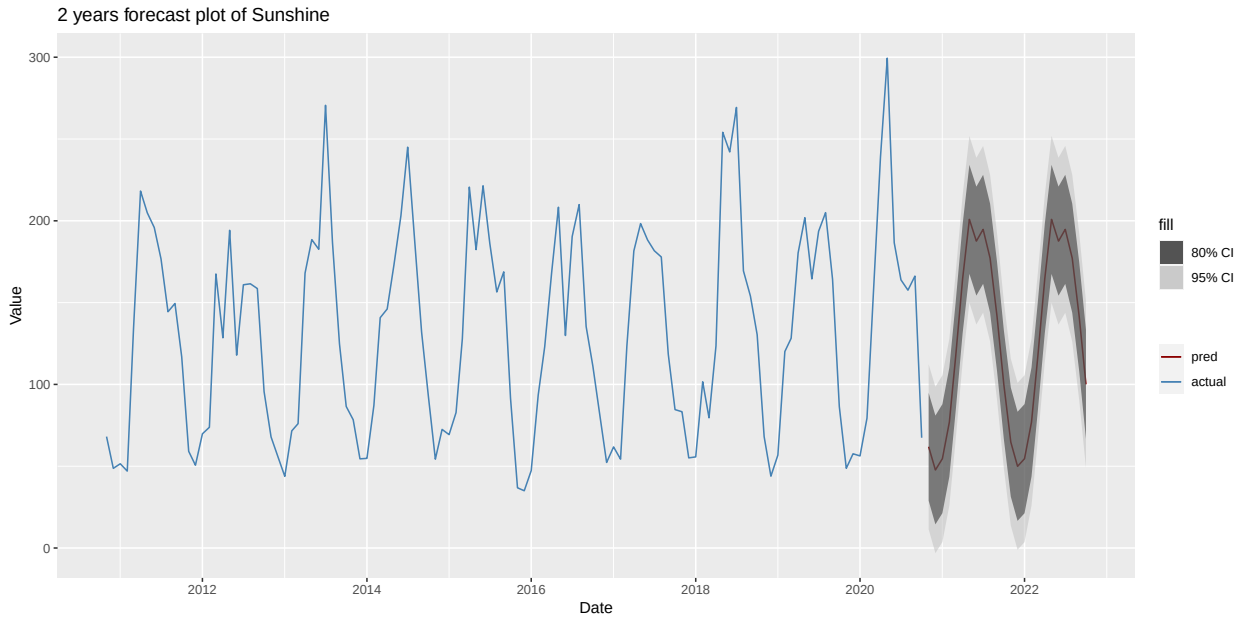


Figure 10: 24 Month forecasting

1.2 Rainfall series

1.2.1 Preliminary analysis

By comparing the preliminary analysis based on the entire dataset (Figure 11) and partial dataset (Figure 13), using the entire dataset is a bit of stretch and there is no major difference in terms of the unique patterns and trends observed between these two datasets. Hence, for better intuitive understanding on analysis, the preliminary analysis is conducted on the partial dataset (21th century) but the final modelling will be using the entire population.

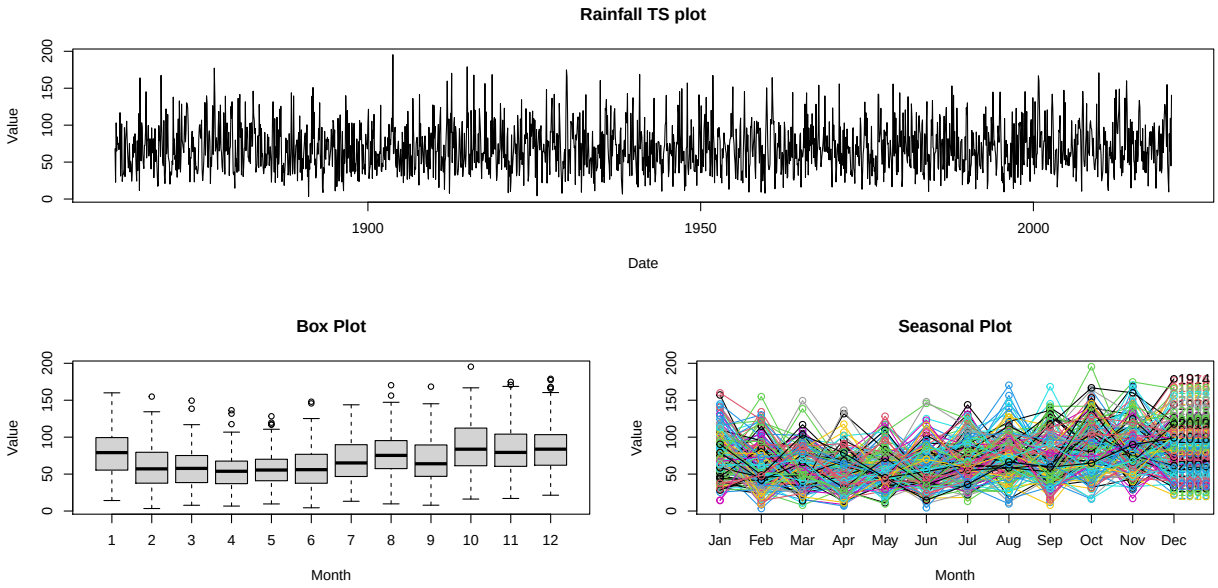


Figure 11: Preliminary analysis on entire Rainfall series

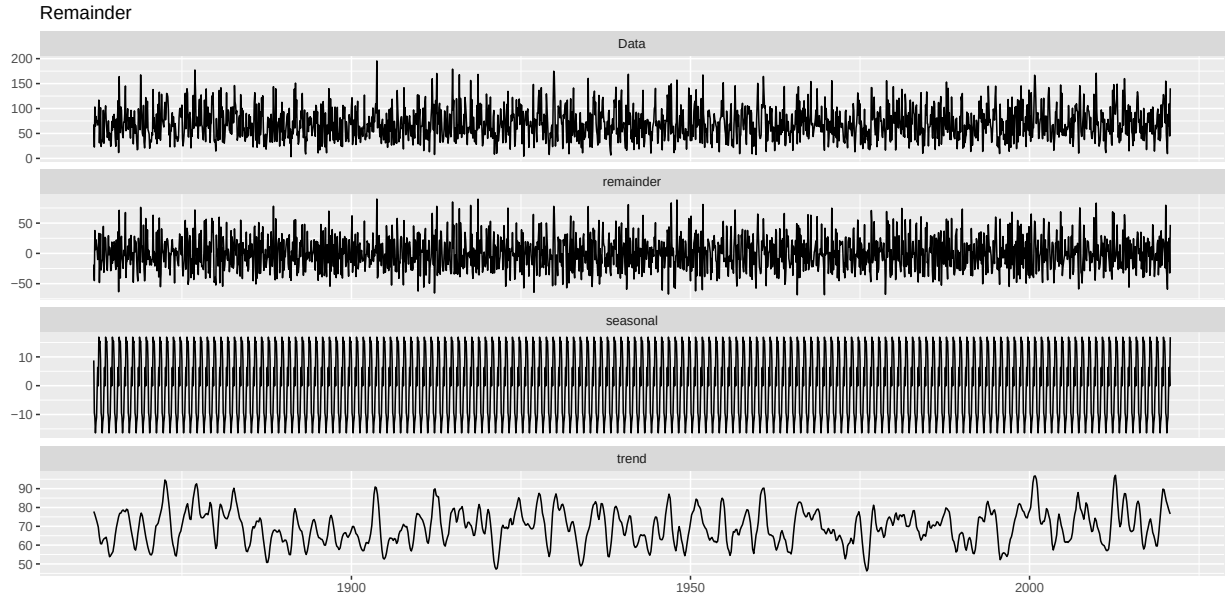


Figure 12: Decomposition analysis on entire Rainfall series

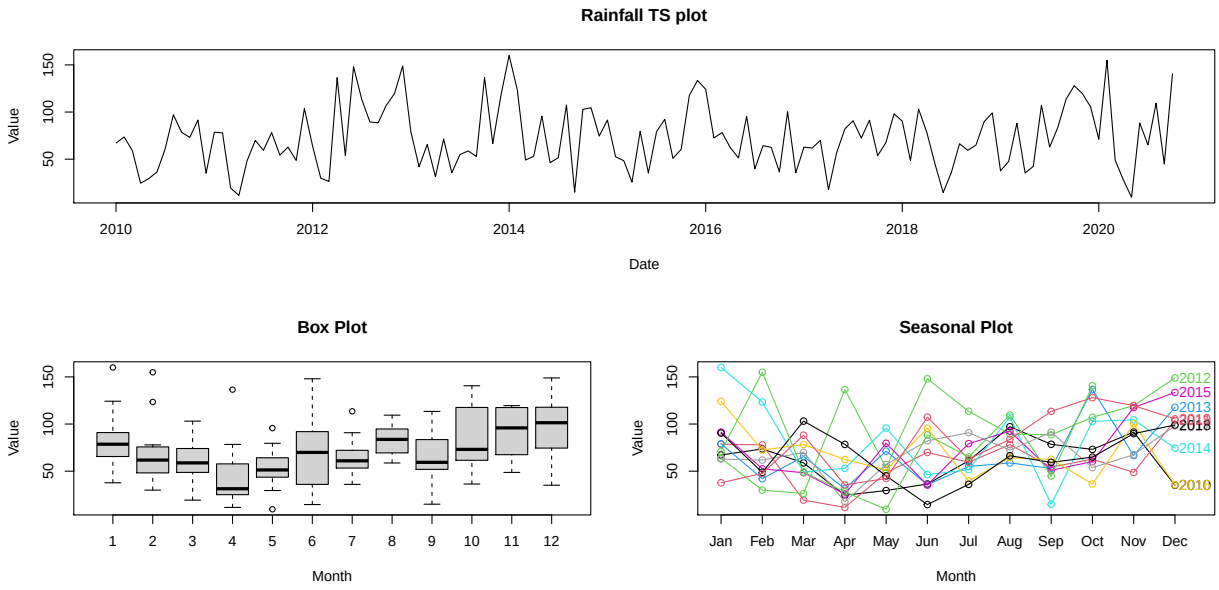


Figure 13: Preliminary analysis on trimmed Rainfall series

Based on preliminary analysis plots on **Rainfall** series, following observations are made:

1. Monthly time-series plot shows high fluctuation but no visible trend nor increasing variance.
2. Box plots and seasonal plot indicate the existence of seasonal trend.
3. Decomposition analysis provide some additional insights that the trend cycle and the seasonal plot shows there's seasonal fluctuation occurred with no specific trend and fairly random remainder/residual.

Conducted analysis indicate that rainfall seems to be well dispersed throughout the whole year. Notably, July and August are the wettest months and late winter - spring (Feb ~ Mar) are the driest months as well as autumn - winter (Oct ~ Jan) the wettest.

Given this observed seasonality in time series, I further delve into the existence of seasonal persistence and the time series model identification with ACF & PACF plots (Figure 15).

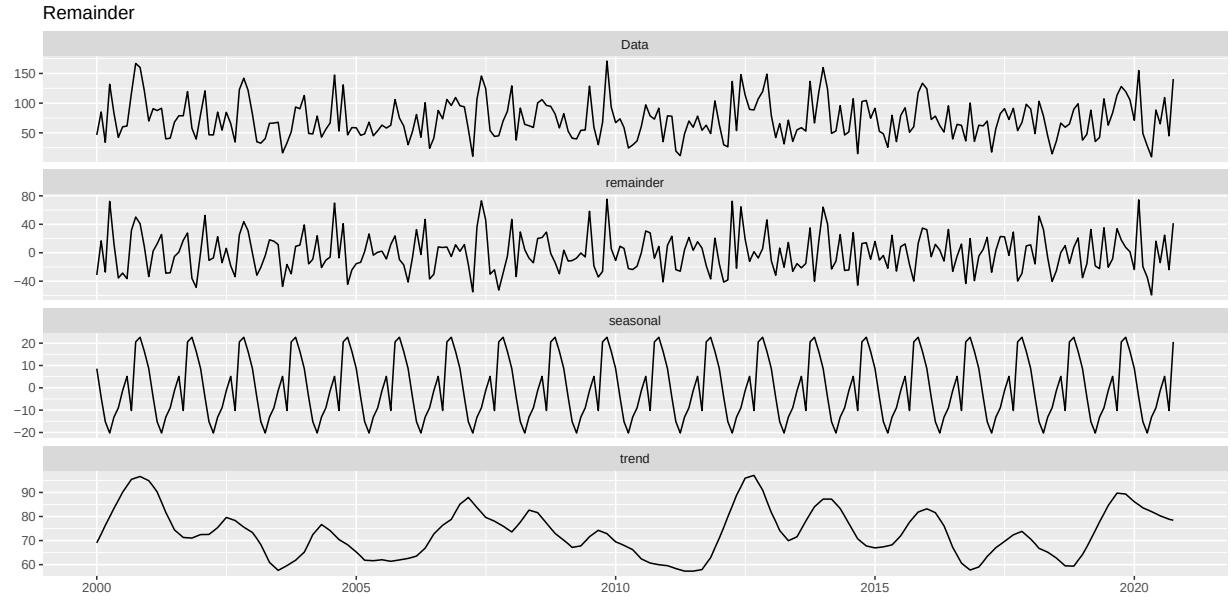


Figure 14: Decomposition analysis on trimmed Rainfall series

1.2.2 ACF, PACF and stationarity analysis

```
##
##  Augmented Dickey-Fuller Test
##
## data:  ts_v
## Dickey-Fuller = -10.874, Lag order = 12, p-value = 0.01
## alternative hypothesis: stationary
```

Prior to ACF and PACF assessment, there is one of the most important conditions required for time-series analysis that I need to check, stationarity. Augmented Dickey-Fuller test is the most commonly used test method to attest the stationarity of series. Based on the conducted analysis, the result could reject the null hypothesis at 5% significance level that the series is stationary.

In addition, based on ACF and PACF plots, I can verify the existence of seasonal persistence around 12 every month and can gain further insights and draw the blueprint of our model specification:

1. Both ACF and PACF tail off -> ARMA(1,1)
2. Other model suggestion & exploration with modifying AR and MA component by ± 1

Based on seasonal pattern observed in the preliminary analysis, consideration of a multiplicative seasonal ARIMA model is reasonable. Our variable of interest, monthly rainfall x_t as being modeled as:

$$x_t = S_t + w_t$$

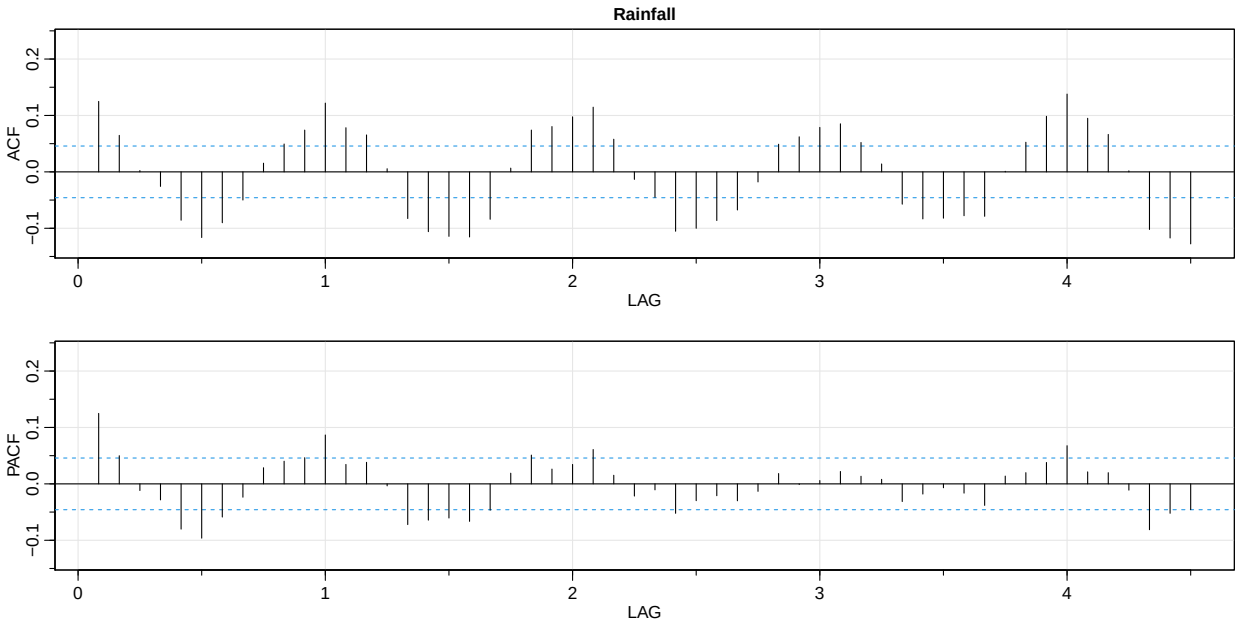


Figure 15: ACF & PACF plots of rainfall series

where S_t is a seasonal component that varies a little from one year to the next.

As a next step, 12 month differencing is going to be examined to find a roughly stationary series and then find a multiplicative seasonal ARIMA to fit the resulting residual series.

1.2.3 ∇_{12} series analysis

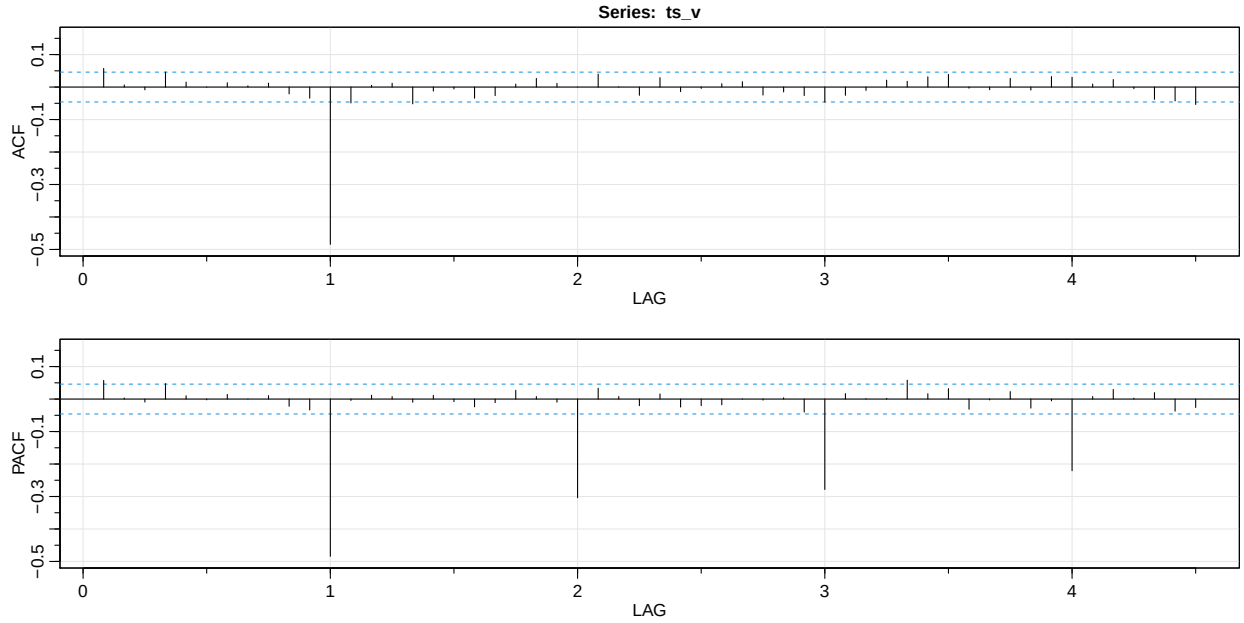


Figure 16: ACF & PACF plots of seasonally differenced rainfall series

```
##
##  Augmented Dickey-Fuller Test
##
## data:  ts_v
## Dickey-Fuller = -19.281, Lag order = 12, p-value = 0.01
## alternative hypothesis: stationary
```

The resulting ACF and PACF (Figure 16) possibly indicate that:

1. ACF cuts off at lag 1 and PACF decays quickly -> MA(1)
2. Other model suggestion & exploration with modifying AR and MA component by ± 1

1.2.4 Model fitting using SARIMA

In this section, I fit the optimal parameters of the time-series model based on insights gained from previous analysis and extend the scope with other models based on the suggestion with modifying AR and MA component by ± 1. For model exploration, an exhaustive approach (i.e. grid-search) will be used to calculate the goodness of fit of a possible set models and rank them based on the performance (i.e. AIC). Furthermore, an additional candidate model using auto.arima() function from the third party R package is going to be examined as well. Hence, total 3 different specifications will be compared:

1. $ARMA(1,0,1) \times (0,1,1)_{12}$ <- Best guess model based on preliminary analysis.
2. Grid-search model
3. auto.arima() model

Initial non-seasonal & seasonal parameters for grid-search approach are derived from Model 1: $ARMA(1,0,1) \times (0,1,1)_{12}$.

1.2.5 Model diagnosis and final model selection

```
# Models
best_guess_rf # Best guess
```

```
## Series: rf_ts
## ARIMA(1,0,1)(0,1,1)[12]
##
## Coefficients:
##          ar1      ma1     sma1
##       0.6180  -0.5841  -0.9952
## s.e.  0.4512   0.4617   0.0170
##
## sigma^2 estimated as 908.2:  log likelihood=-9162.88
## AIC=18333.75   AICc=18333.77   BIC=18355.94
```

```
optimal_rf # Grid-search
```

```
## Series: rf_ts
## ARIMA(1,0,0)(0,1,1)[12]
##
## Coefficients:
##          ar1     sma1
##       0.0366  -0.9949
## s.e.  0.0230   0.0159
##
## sigma^2 estimated as 908.4:  log likelihood=-9163.39
## AIC=18332.77   AICc=18332.78   BIC=18349.41
```

```
auto_arima_rf # auto.arima
```

```
## Series: rf_ts
## ARIMA(1,0,0)(2,0,0)[12] with non-zero mean
##
## Coefficients:
##          ar1    sar1    sar2     mean
##       0.1011  0.1031  0.0704  69.8524
## s.e.  0.0232  0.0231  0.0233   0.9733
##
## sigma^2 estimated as 1003:  log likelihood=-9288.14
## AIC=18586.28   AICc=18586.31   BIC=18614.04
```

Resulted 3 different models can be found below:

1. $ARMA(1,0,1) \times (0,1,1)_{12}$ <- Model(1) Best guess

2. $ARMA(1,0,0) \times (0,1,1)_{12}$ <- Model(2) Grid-search
3. $ARMA(1,0,0) \times (2,0,0)_{12}$ <- Model(3) auto.arima

Based on the goodness of fit metrics of 3 different models, Model(1) and Model(2) provided better model-fit results compared to Model(3). Even though there is a marginal difference between Model(1) and Model(2), given the fact that Model(1) shows insignificant coefficients, the final model is narrowed down to Model(2): $ARMA(1,0,0) \times (0,1,1)_{12}$.

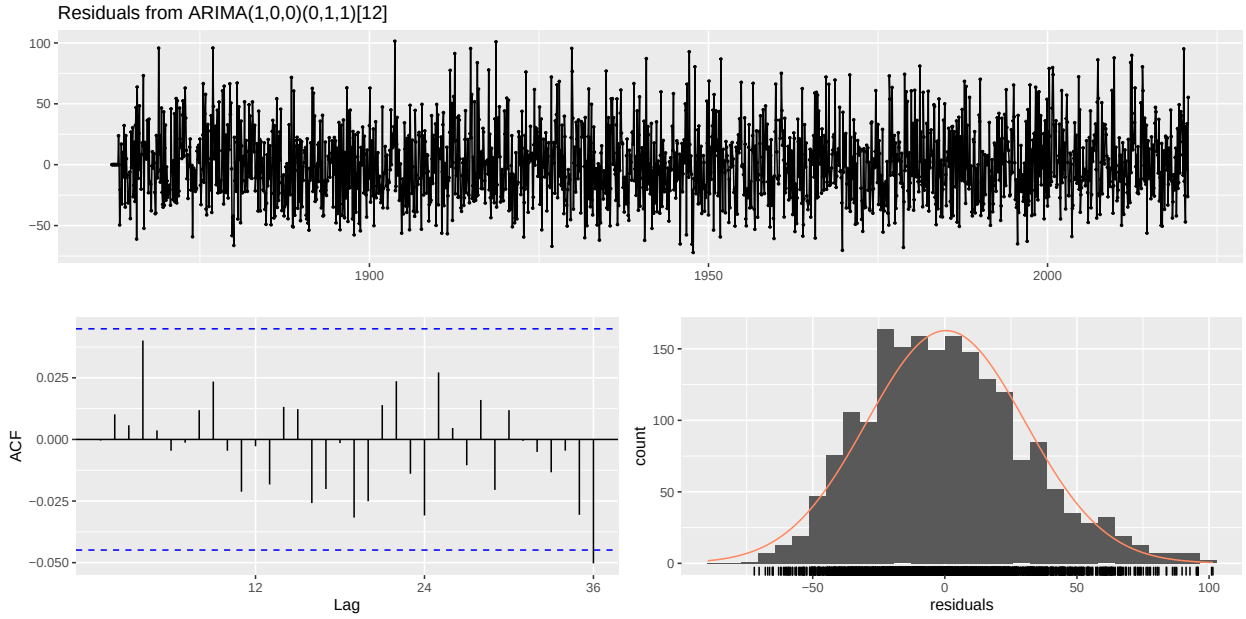


Figure 17: Residual diagnosis

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,0,0)(0,1,1)[12]
## Q* = 15.809, df = 22, p-value = 0.8253
##
## Model df: 2.   Total lags used: 24
```

Based on the Ljung-Box test and ACF plot of model residuals (Figure 17, 18), it can be concluded that this model is appropriate for forecasting since its residuals show white noise behavior and uncorrelated against each other.

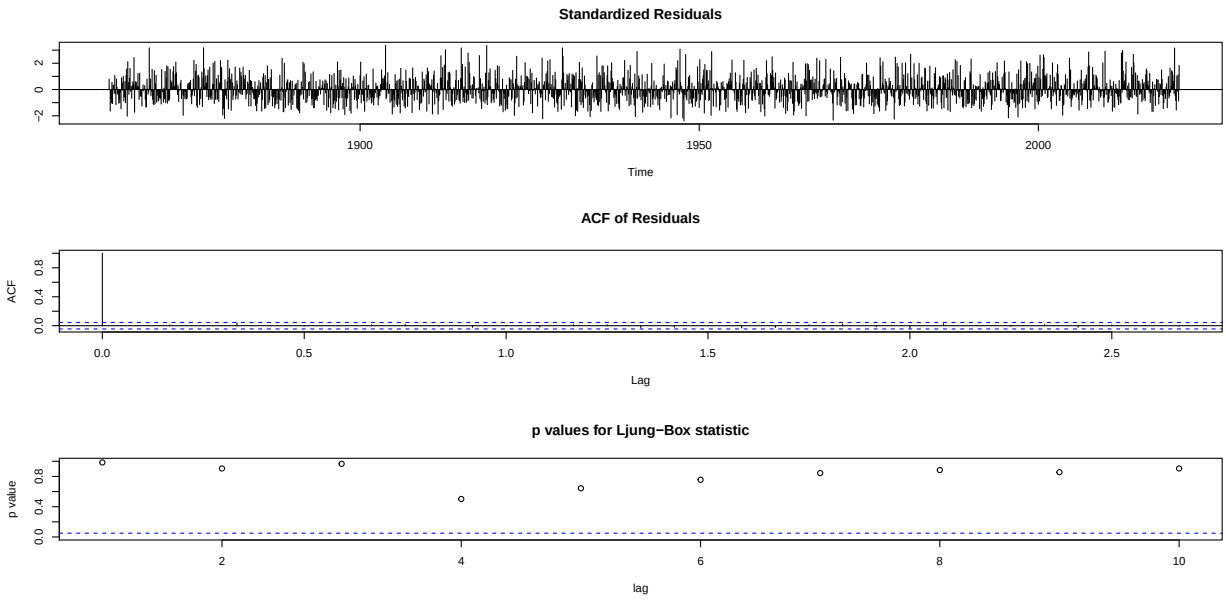


Figure 18: Residual diagnosis

1.2.6 Model forecasting

As a finale, I plot 2 months forecasting plot and 24 months forecast plot to inspect the behaviour of estimated values.

```
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## Nov 2020       86.92119 48.24621 125.5962 27.77291 146.0695
## Dec 2020       85.53629 46.83539 124.2372 26.34837 144.7242
```

Based on forecast visual plots, it can be concluded that the final model is following a general trend without any observed outlines.

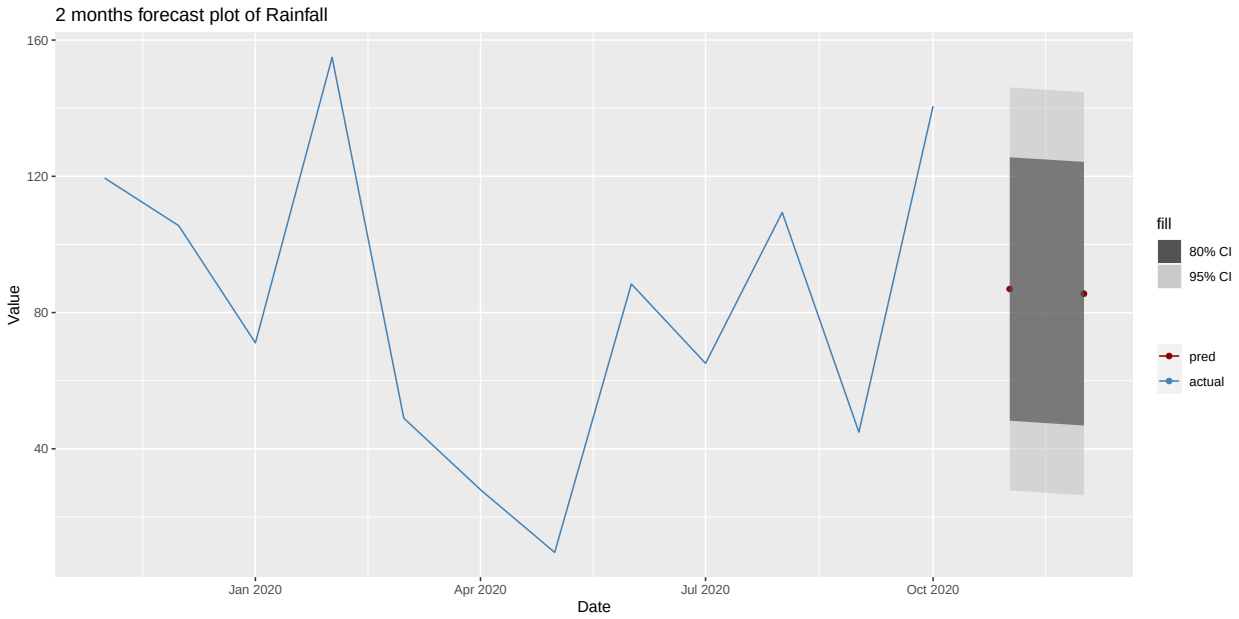


Figure 19: 2 Month forecasting

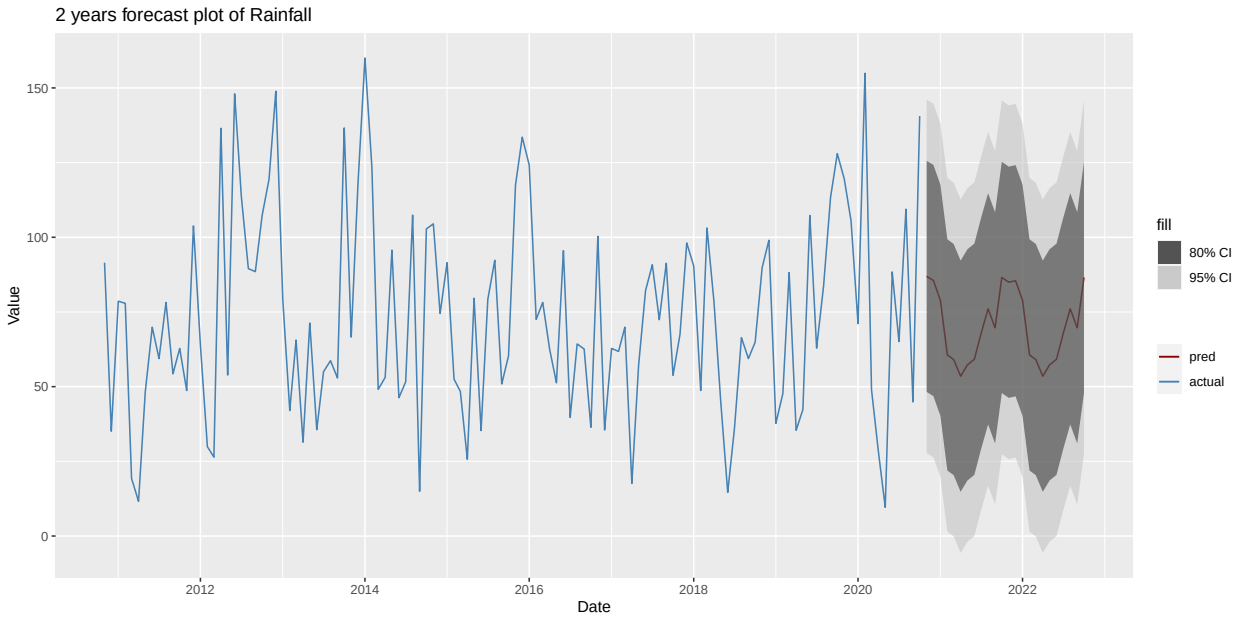


Figure 20: 24 Month forecasting

1.3 Mean Temperat series

1.3.1 Preliminary analysis

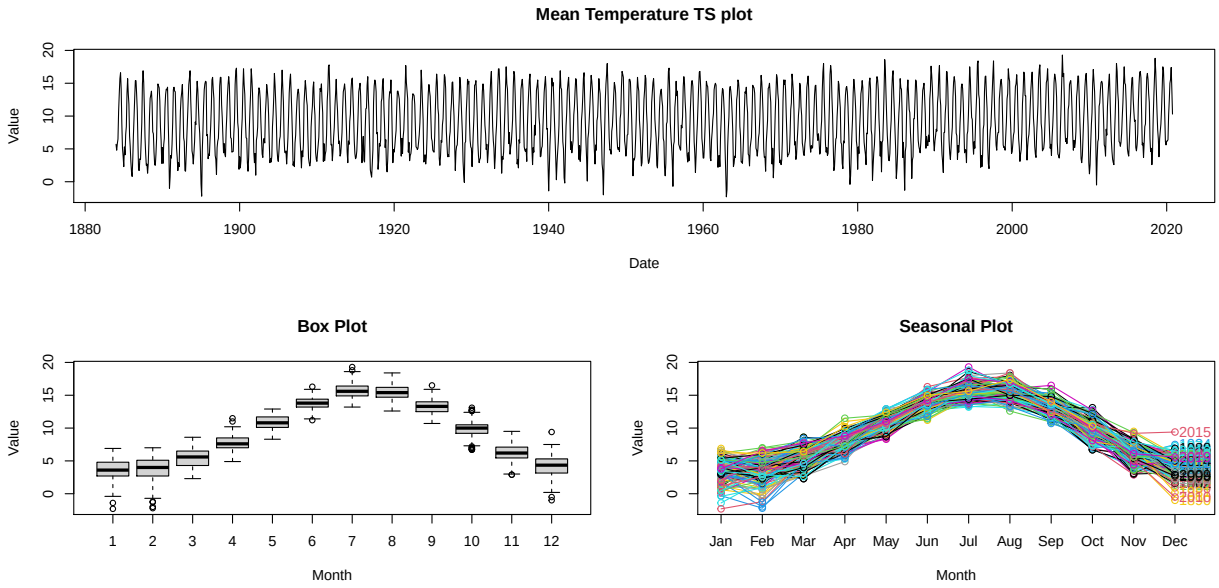


Figure 21: Preliminary analysis on entire Rainfall series

By comparing the preliminary analysis based on the entire dataset (Figure 21) and partial dataset (Figure 23), using the entire dataset is a bit of stretch and there is no major difference in terms of the unique patterns and trends observed between these two datasets. Hence, for better intuitive understanding on analysis, the preliminary analysis is conducted on the partial dataset (21th century) but the final modelling will be using the entire population.

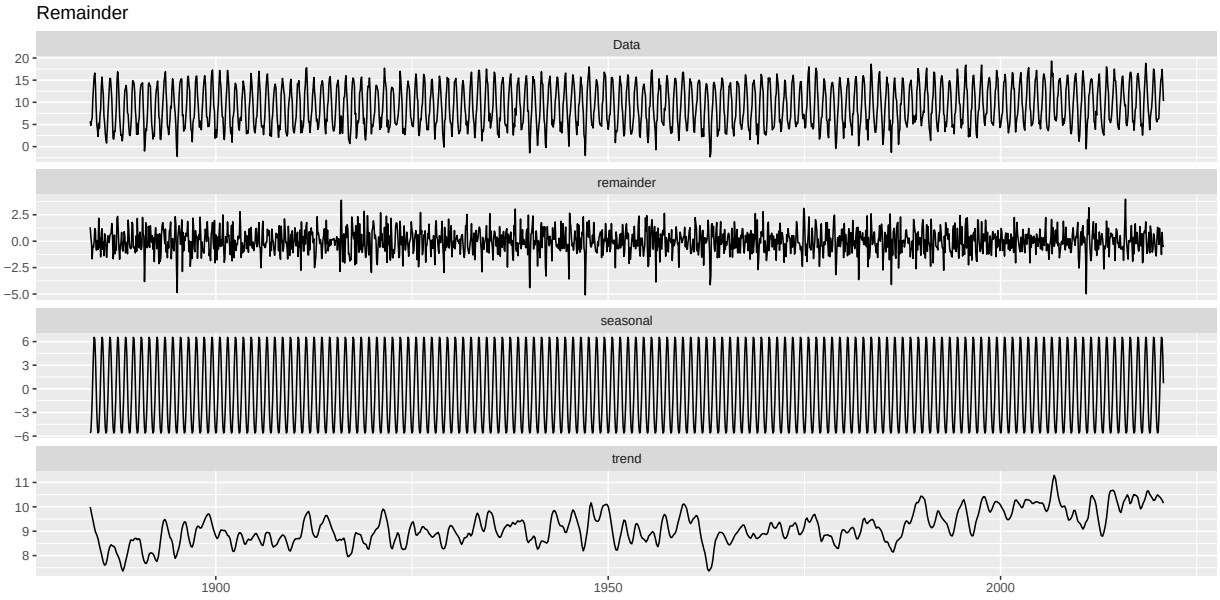


Figure 22: Decomposition analysis on entire Rainfall series

Based on preliminary analysis plots on **Mean Temperature** series, following observations are made:

1. Monthly time-series plot shows high fluctuation but no visible trend nor increasing variance.
2. Box plots and seasonal plot indicate the existence of seasonal trend.
3. Decomposition analysis provide some additional insights that the trend cycle and the seasonal plot shows there's seasonal fluctuation occurred with no specific trend and fairly random remainder/residual.

Conducted analysis indicate that mean temperature seems to be reach the peak during summer (Jul ~ Aug) and reaches to the lowest point during winter (Dec ~ Jan).

Given this observed seasonality in time series, I further delve into the existence of seasonal persistence and the time series model identification with ACF & PACF plots.

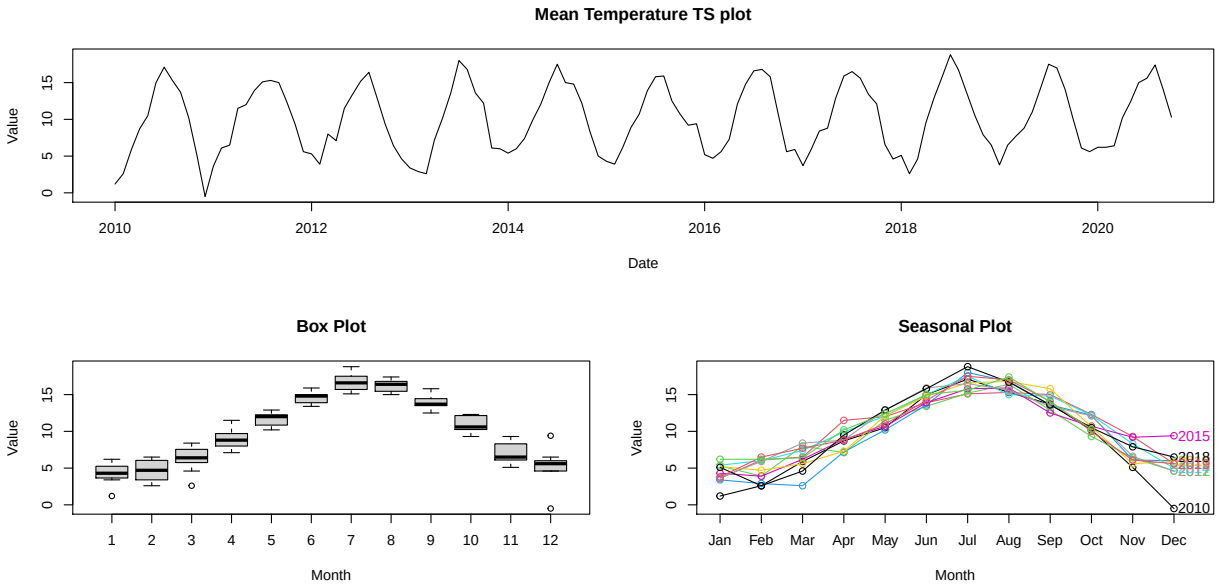


Figure 23: Preliminary analysis on trimmed Mean Temperature series

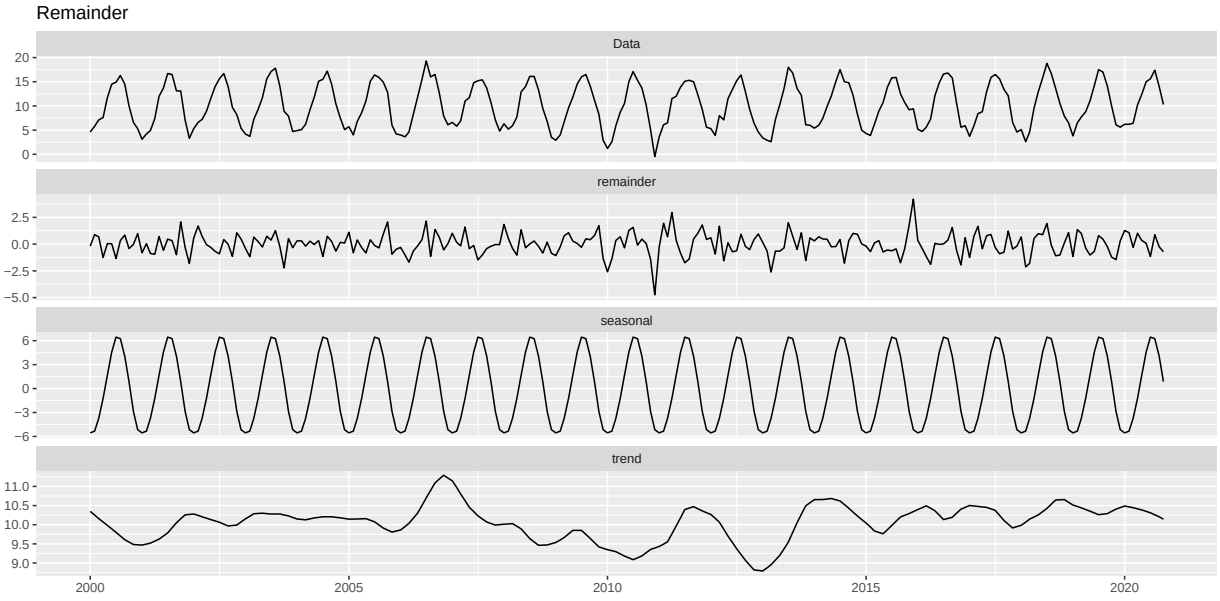


Figure 24: Decomposition analysis on trimmed Mean Temperature series

1.3.2 ACF, PACF and stationarity analysis

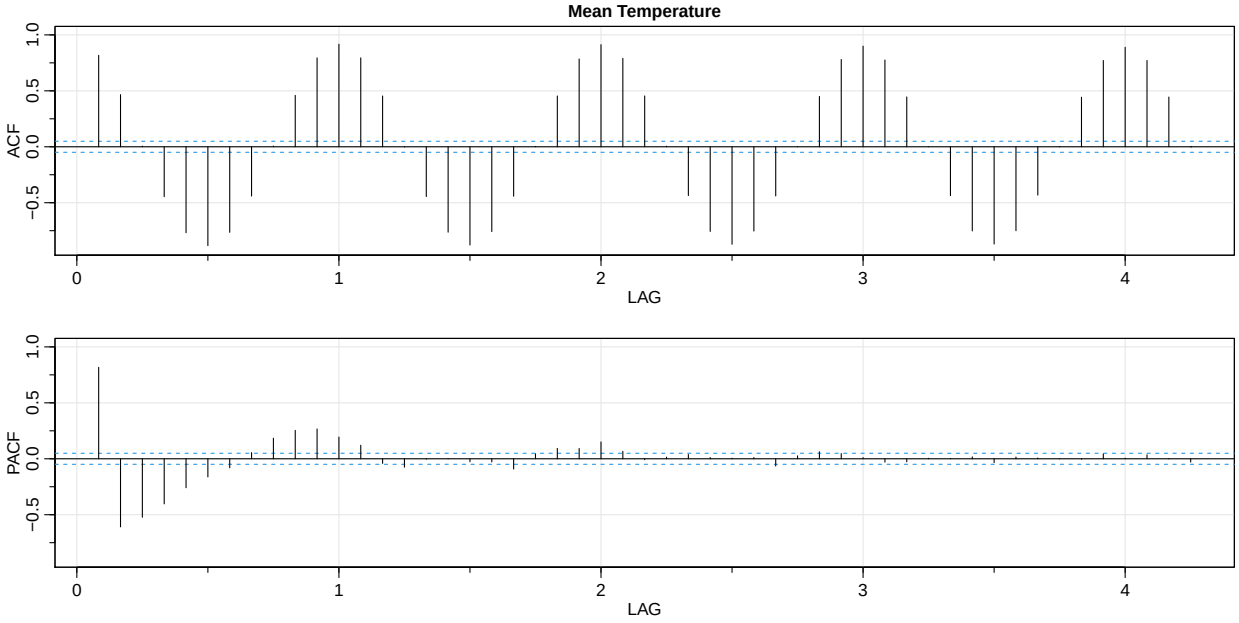


Figure 25: ACF & PACF plots of Mean Temperature series

```
##
##  Augmented Dickey-Fuller Test
##
## data:  ts_v
## Dickey-Fuller = -7.0541, Lag order = 12, p-value = 0.01
## alternative hypothesis: stationary
```

Prior to ACF and PACF assessment, there is one of the most important conditions required for time-series analysis that I need to check, stationarity. Augmented Dickey-Fuller test is the most commonly used test method to attest the stationarity of series. Based on the conducted time-series analysis, the result could reject the null hypothesis at 5% significance level that the series is stationary.

In addition, based on ACF and PACF plots (Figure 25), it can verify the existence of seasonal persistence around 12 every month and can gain further insights and draw the blueprint of our model specification:

1. ACF cuts off at lag 2 and PACF tails off -> MA(2)
2. Other model suggestion & exploration with modifying AR and MA component by ± 1

Based on seasonal pattern observed in the preliminary analysis, consideration of a multiplicative seasonal ARIMA model is reasonable. Our variable of interest, monthly mean temperature x_t as being modeled as:

$$x_t = S_t + w_t$$

where S_t is a seasonal component that varies a little from one year to the next.

As a next step, 12 month differencing is going to be examined to find a roughly stationary series and then find a multiplicative seasonal ARIMA to fit the resulting residual series.

1.3.3 ∇_{12} series analysis

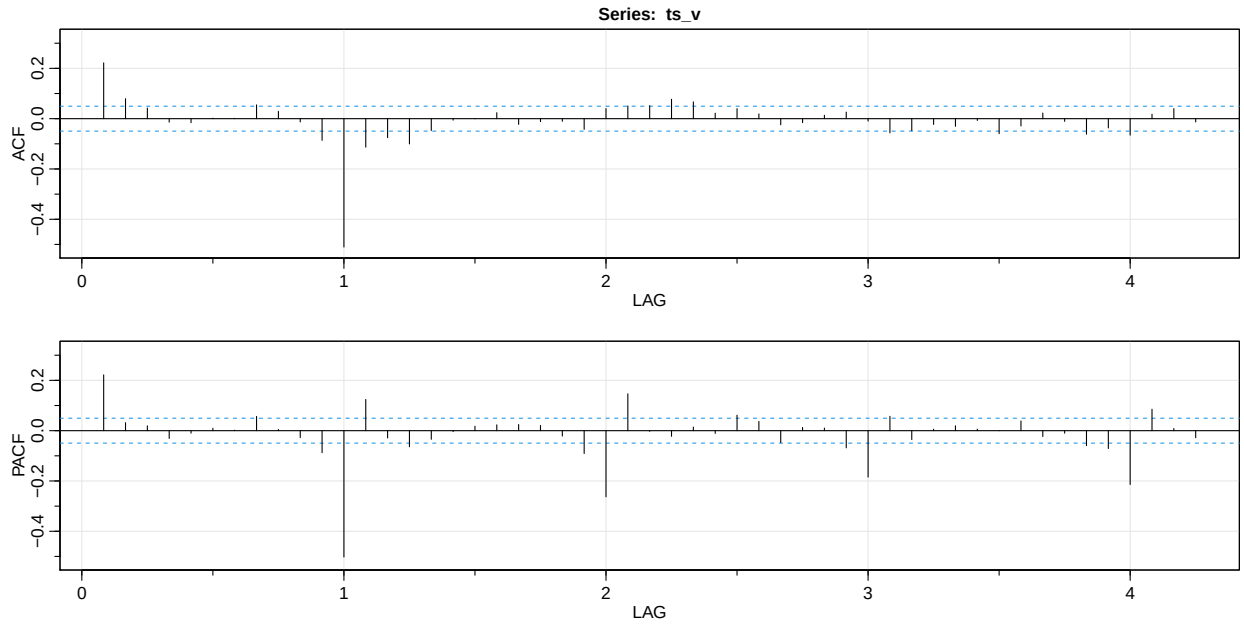


Figure 26: ACF & PACF plots of seasonally differenced Mean Temperature series

```
##
##  Augmented Dickey-Fuller Test
##
## data:  ts_v
## Dickey-Fuller = -16.419, Lag order = 12, p-value = 0.01
## alternative hypothesis: stationary
```

The resulting ACF and PACF (Figure 26) possibly indicate that:

1. ACF cuts off at lag 1 and PACF decays quickly -> MA(1)
2. Other model suggestion & exploration with modifying AR and MA component by ± 1

1.3.4 Model fitting using SARIMA

In this section, I fit the optimal parameters of the time-series model based on insights gained from previous analysis and extend the scope with other models based on the suggestion with modifying AR and MA component by ± 1. For model exploration, an exhaustive approach (i.e. grid-search) will be used to calculate the goodness of fit of a possible set models and rank them based on the performance (i.e. AIC). Furthermore, an additional candidate model using auto.arima() function from the third party R package is going to be examined as well. Hence, total 3 different specifications will be compared:

1. $ARMA(0,0,2) \times (0,1,1)_{12}$ <- Best guess model based on preliminary analysis.
2. Grid-search model
3. auto.arima() model

Initial non-seasonal & seasonal parameters for grid-search approach are derived from Model 1: $ARMA(0,0,2) \times (0,1,1)_{12}$.

1.3.5 Model diagnosis and final model selection

```
# Comparison
best_guess_mt # Best guess
```

```
## Series: mt_ts
## ARIMA(0,0,2)(0,1,1)[12]
##
## Coefficients:
##          ma1     ma2     sma1
##       0.2550  0.0881  -0.9427
## s.e.  0.0248  0.0244   0.0095
##
## sigma^2 estimated as 1.619:  log likelihood=-2717.19
## AIC=5442.38   AICc=5442.41   BIC=5463.97
```

```
optimal_mt # Grid-search
```

```
## Series: mt_ts
## ARIMA(1,0,3)(0,1,1)[12]
##
## Coefficients:
##          ar1      ma1      ma2      ma3     sma1
##       0.9999  -0.7540  -0.1584  -0.0692  -0.9953
## s.e.  0.0002   0.0247   0.0321   0.0248   0.0076
##
## sigma^2 estimated as 1.554:  log likelihood=-2694.7
## AIC=5401.4   AICc=5401.45   BIC=5433.78
```

```
auto_arima_mt # auto.arima
```

```
## Series: mt_ts
## ARIMA(1,0,0)(2,1,0)[12]
##
## Coefficients:
##          ar1     sar1     sar2
##       0.2535  -0.6820  -0.3117
## s.e.  0.0240   0.0236   0.0237
##
## sigma^2 estimated as 2.062:  log likelihood=-2904.27
## AIC=5816.53   AICc=5816.56   BIC=5838.12
```

Resulted 3 different models can be found below:

1. $ARMA(0,0,2) \times (0,1,1)_{12}$ <- Model(1) Best guess

2. $ARMA(1,0,3) \times (0,1,1)_{12}$ <- Model(2) Grid-search
3. $ARMA(1,0,0) \times (2,1,0)_{12}$ <- Model(3) auto.arima

Based on the goodness of fit metrics of 3 different models, Model(2) provides the best model fit with all significant coefficients : $ARMA(1,0,3) \times (0,1,1)_{12}$.

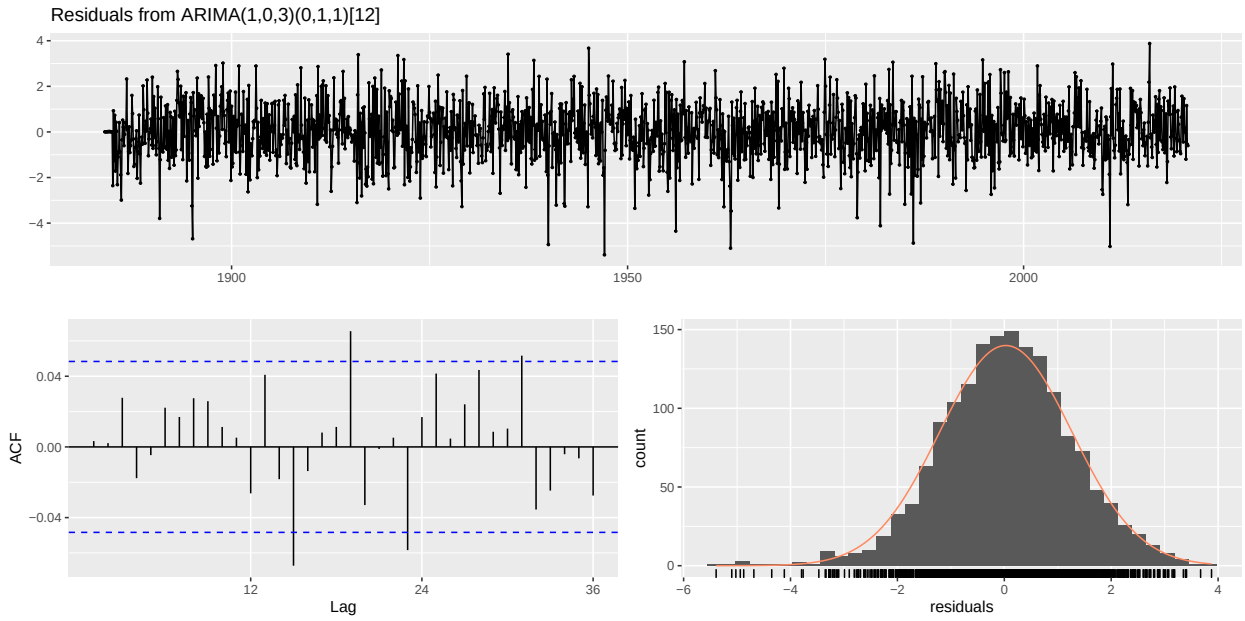


Figure 27: Residual diagnosis

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,0,3)(0,1,1)[12]
## Q* = 33.491, df = 19, p-value = 0.02108
##
## Model df: 5.   Total lags used: 24
```

Based on the Ljung-Box test and ACF plot of model residuals (Figure 27 & 28), it can be concluded that this model is appropriate for forecasting since its residuals show white noise behavior and uncorrelated against each other.

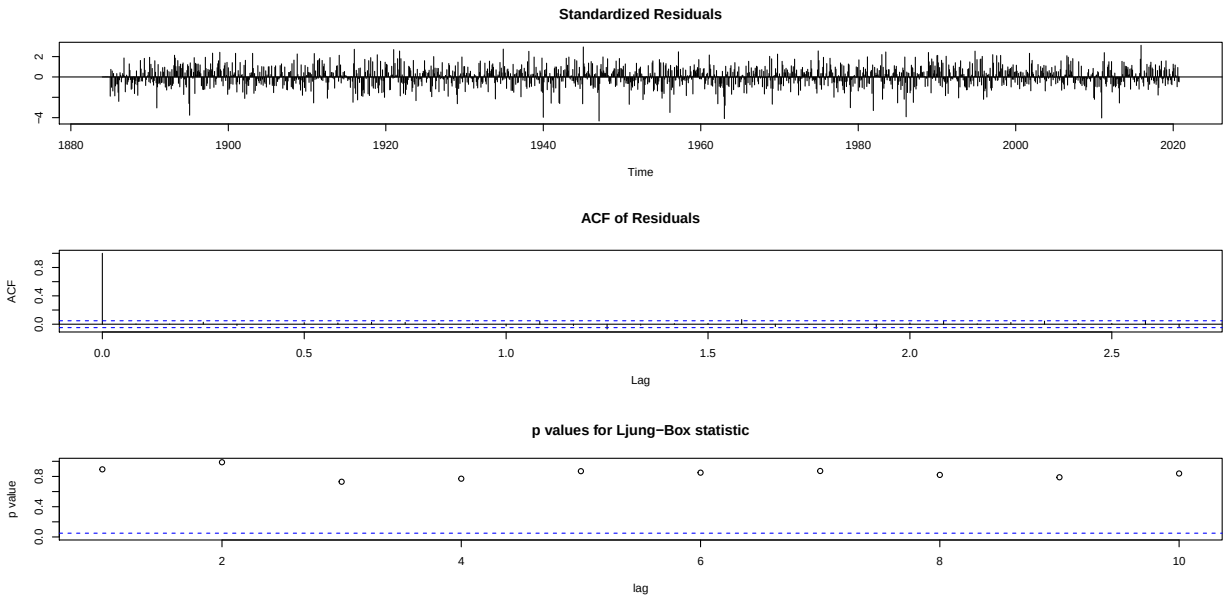


Figure 28: Residual diagnosis

1.3.6 Model forecasting

As a finale, I plot 2 months forecasting plot and 24 months forecast plot to inspect the behaviour of estimated values.

```
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## Nov 2020       7.138369 5.537849 8.738889 4.690585 9.586153
## Dec 2020       5.257024 3.608838 6.905209 2.736341 7.777706
```

Based on forecast visual plots below, it can be concluded that the final model is following a general trend without any observed outlines.

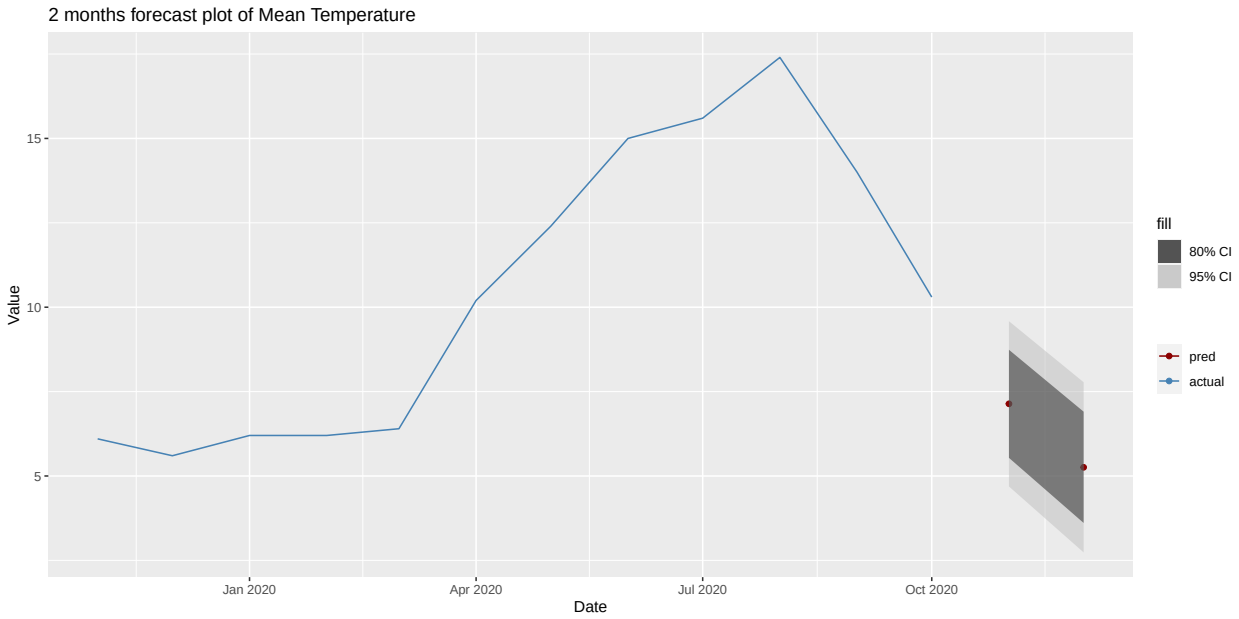


Figure 29: 2 Month forecasting

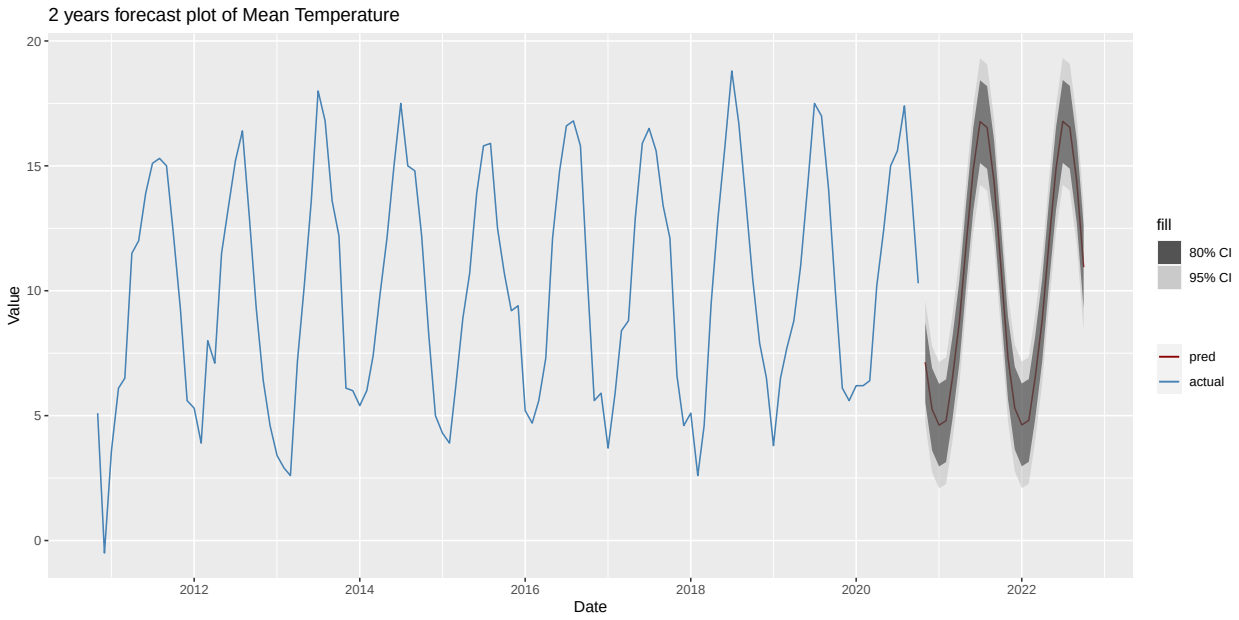


Figure 30: 24 Month forecasting

2 Q3: Rolling Window

2.1 Description

Various rolling window approaches are considered in this section. Given the sheer amount of dataset available for modelling, a list of rolling windows for model development (i.e. fitting), "M" is considered and a fixed value of validation window (i.e. forecast & comparison against the actual values), "A" is considered to find the optimal rolling window length.

With a fixed value of "A", the larger "M" value is expected to fit the model with lower error as larger sample sizes give more reliable results with greater precision and power. Hence, various rolling windows for model development (M = 36, 72, 108, 144, 180; 3 years increment up to 15 years) are considered for the robust result.

Having a larger validation window (A) reduces the number of required iteration for model refitting. Furthermore, the shorter interval of validation window is introduced, the more frequent model refitting is needed. This could lead to one of menaces of statistical model development, the model over-fitting. In addition, having a long validation window is not expected to compromise the final outcome given the size of dataset available (More than 100 years). Hence, the fixed value of **3 years** is deployed for the model validation window (A), parallel to the increment size of rolling window "M".

2.2 Results

2.2.1 Sunshine

Table 1: Optimal window searching result for Sunshine (Validation period, A = 36)

Window	Sample	RMSE	MAE
36	916	38.08084	28.94121
36	306	35.30199	26.46222
72	916	31.68964	23.74663
72	306	29.48377	21.53583
108	916	29.17757	21.74045
108	306	28.33817	20.49545
144	916	29.38550	21.70412
144	306	27.73625	20.02084
180	916	30.35056	22.39793
180	306	30.73793	22.68594

Based on the optimal window searching result for Rainfall, Train sample (Sample=916) has the lowest RMSE value when M=108 whereas Test sample (Sample=306) shows the lowest at M=144 The observed difference of Train RMSE between M=108 (29.17757) and M=144 (29.38550) is marginal. Given M=144 shows a much smaller RMSE value (32.25016) for Test dataset, it is concluded that **M=144** is the most optimal window value for Sunshine series.

2.2.2 Rainfall

Table 2: Optimal window searching result for Rainfall (Validation period, A = 36)

Window	Sample	RMSE	MAE
36	1429	33.27800	26.71494
36	477	40.19060	33.33170
72	1429	32.06189	25.59037
72	477	32.25016	26.35830
108	1429	31.92845	25.50503
108	477	32.41124	26.41661
144	1429	32.04302	25.54894
144	477	32.65402	26.59269
180	1429	31.95238	25.41482
180	477	32.47736	26.40725

Based on the optimal window searching result for Rainfall, Train sample (Sample=1429) has the lowest RMSE value when M=108 whereas Test sample (Sample=477) shows the lowest at M=72 The observed difference of Train RMSE between M=108 (31.92845) and M=72 (32.06189) is marginal. Given M=72 shows a much smaller RMSE value (32.25016) for Test dataset, it is concluded that **M=72** is the most optimal window value for Rainfall series.

2.2.3 Mean Temperature

The result of Mean Temperature follows:

Table 3: Optimal window searching result for Mean Temperature (Validation period, A = 36)

Window	Sample	RMSE	MAE
36	1231	1.719416	1.346605
36	411	1.830766	1.480863
72	1231	1.490044	1.165079
72	411	1.362905	1.090364
108	1231	1.477336	1.153452
108	411	1.357849	1.095313
144	1231	1.446246	1.125278
144	411	1.298890	1.052184
180	1231	1.467112	1.143347
180	411	1.367413	1.111805

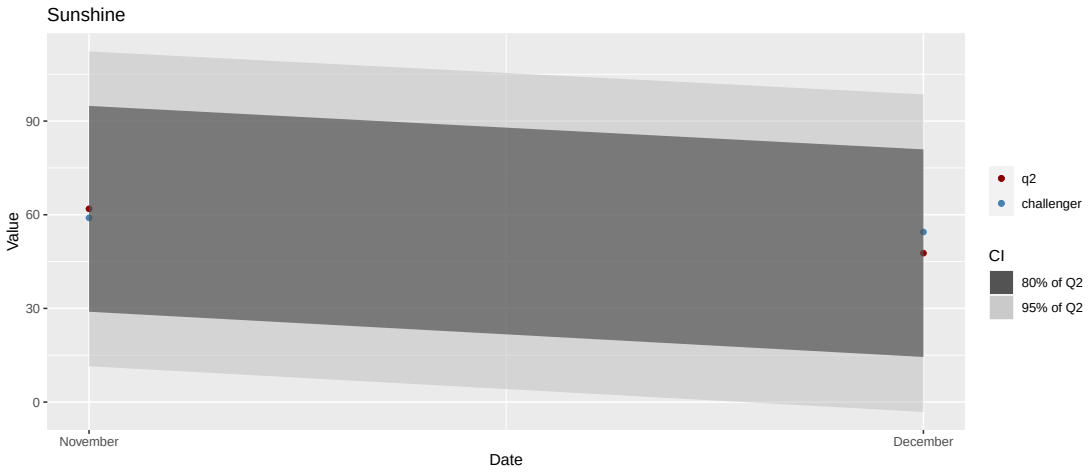
Based on the optimal window searching result for Mean Temperature, the appropriate window value is **144** considering the lowest values of RMSE & MAE values of both Train (N=1231) and

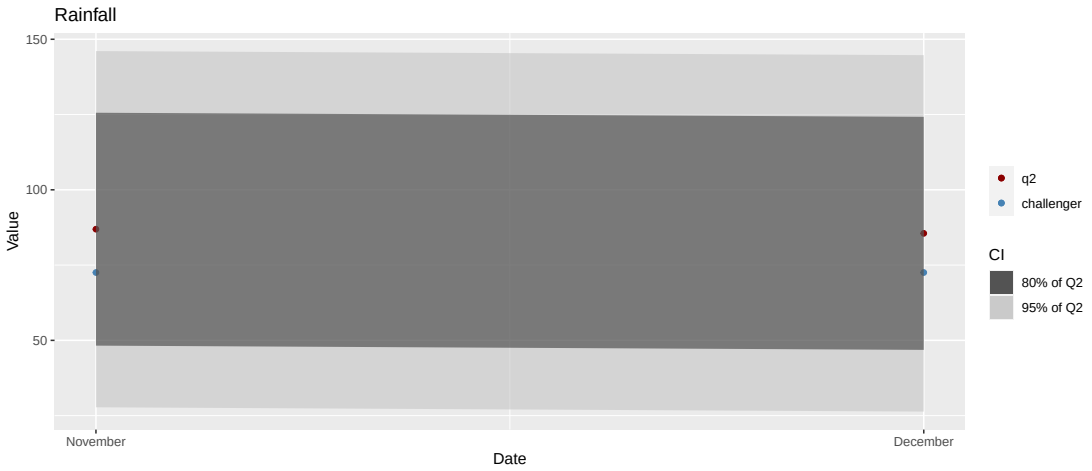
Test (N=411) samples.

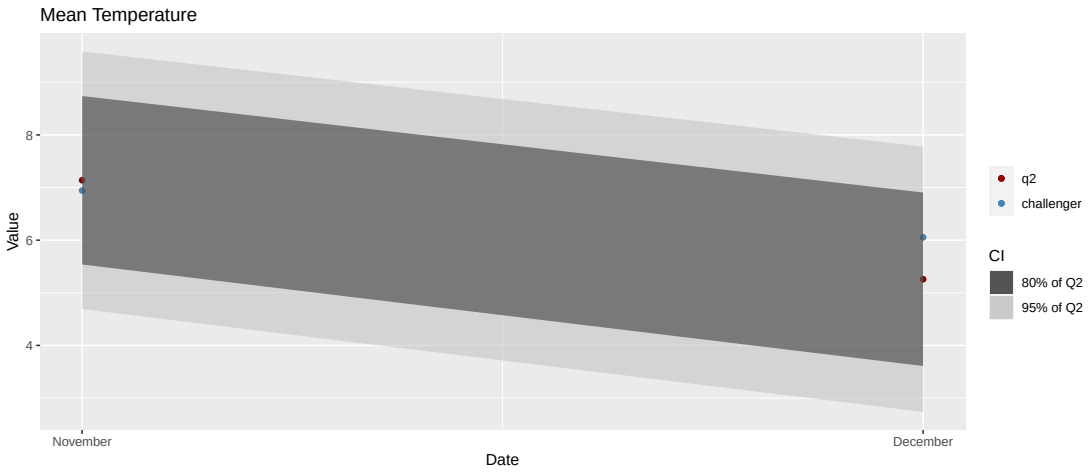
2.3 Forecasting

Established on the above rolling window iteration analysis, 2 months forecast (2020 Nov & Dec) values are derived with selected window lengths of "M" for each time series. Being consistent with the rolling window iteration analysis method, auto.arima() function is deployed to fit the model for the 2 months ahead forecasts production.

Predicated on the below forecast plots, it is observed that the foretasted values of the rolling window approach are not significantly different (Within 80% C.I.) compared to the predicted value derived on Question 2 using SARIMA model.


Sunshine
90
60
30
0
Value
November
December
Date
q2
challenger
CI
80% of Q2
95% of Q2


Rainfall
150
100
50
Value
November
December
Date
q2
challenger
CI
80% of Q2
95% of Q2


Mean Temperature
Value
8
6
4
November
December
Date
q2
challenger
CI
80% of Q2
95% of Q2

3 Q4: VARMA

3.1 Description

As a brief introduction, a multivariate process of ARMA is commonly described as a VARMA process—the initial letter denoting "vector". In other terms, VARMA is an extended version of finite order VAR model. VARMA model is used to develop multivariate time series models which allow the users to study the dynamic relationships between multiple variables. This modelling technique allows the error terms to be autocorrelated rather than a white noise process as in VAR.

General form of VARMA(p,q) process can be represented as:

$$y_t = v + A_1 y_{t-1} + ... + A_p y_{,t-p} + M_1 e_{t-1} + ... + M_q e_{,t-q} \tag{1}$$

where $A_0, A_1, ..., A_p$ are AR parameter matrices, $M_0, M_1, ..., M_p$ are MA parameter matrices and e_t is a white-noise process with zero mean, nonsingular, time-invariant covariance matrix.

The function 'sVARMA' included in 'MTS v 1.0' package is chosen for the model fitting function. The custom grid-search function will be re-deployed to fit the model in a more robust manner. This custom grid-search function will modify AR and MA component by ± 1.

The required amount of CPU-energy and processing time for finding the optimal window length and fitting the model under the rolling-window approach is expected to outstrip what is required for the vanilla approach. Considering sheer amount of available data for model development, the difference in goodness of model-fit between the vanilla and rolling-window approach is expected to be immaterial. Hence, it is decided not to implement the rolling window approach.

As a data cleansing process, each individual variables is needed to share the same length without any omitted values. Hence, the observed date of the final combined design matrix is from Jan 1919 to Oct 2020.

3.2 Model fitting process

Considering the preliminary analysis conducted for each individual variables in previous sections, it is reasonable to set a multiple model assumptions that:

1. There is a seasonality with 12 month frequency.
2. Seasonal component is an integrated process, hence d = 1.
3. There is no trend observed for non-seasonal component.

To begin with, the results of univariate SARIMA models in Question 2 need to be re-examined. Considering all series (Sunshine, Rainfall and Mean Temperature) are sharing the same seasonal MA(1) component, it is reasonable to deploy MA(1) for the seasonal component for sVARMA model and extending the grid-search hinged on this starting point.

In addition, the starting point of non-seasonal component (p,q,d) is arbitrally set to be (1,0,1), which will give us the largest combination of $(1 \pm 1, 0, 1 \pm 1)$ for an extensive model combination of grid-search.

Rolling window approach in Question 3 is not considered due to the observed fact that the optimal window lengths of each series are not identical to construct the design matrix for sVARMA modelling.

Table 4: Seasonal VARMA Iteration table

AIC	BIC	NON_SEASONAL	SEASONAL
13.6623647910908	13.8140520001728	1,0,1	0,1,2
13.6658118562649	13.8174990653469	1,0,0	2,1,1
13.6936070664358	13.8452942755178	0,0,1	2,1,1
13.6954243739923	13.7712679785333	0,0,0	0,1,2
13.6955878950415	13.809353301853	1,0,0	0,1,2
13.7114281462501	13.8631153553321	2,0,0	0,1,2
13.7127996462133	13.8265650530248	0,0,0	2,1,1
13.7293382850828	13.9189472964353	0,0,2	2,1,1
13.7325625585365	13.8084061630775	0,0,1	0,1,1
13.7362890358854	13.8121326404265	1,0,0	0,1,1
13.7420374550165	13.855802861828	2,0,0	0,1,1
13.7423101821685	13.85607558898	0,0,1	1,1,1
13.7492251777483	13.8629905845599	0,0,2	0,1,1
13.77083115954	13.8087529618106	0,0,0	0,1,1
13.7860601183937	13.8619037229347	0,0,0	1,1,1
14.3716003359865	14.485365742798	0,0,1	2,1,0
14.3730732849223	14.5247604940044	0,0,2	2,1,0
14.6742060392855	14.8258932483675	1,0,2	1,1,0
14.6897140820938	14.7655576866348	1,0,0	1,1,0
14.699629505075	14.8133949118865	2,0,0	1,1,0
14.7012590728607	14.7771026774017	0,0,1	1,1,0
14.7053169146378	14.8190823214494	0,0,2	1,1,0
14.7571382970354	14.7950600993059	0,0,0	1,1,0
15.326355313185	15.4401207199965	1,0,0	2,1,0
15.4716761980231	15.5475198025641	0,0,2	0,1,0
NaN	NaN	2,0,2	1,1,0

Based on the Seasonal VARMA Iteration table, it can be concluded that $sVARMA(1,0,1) \times (0,1,2)_{12}$ is the optimal model.

3.3 Forecasting

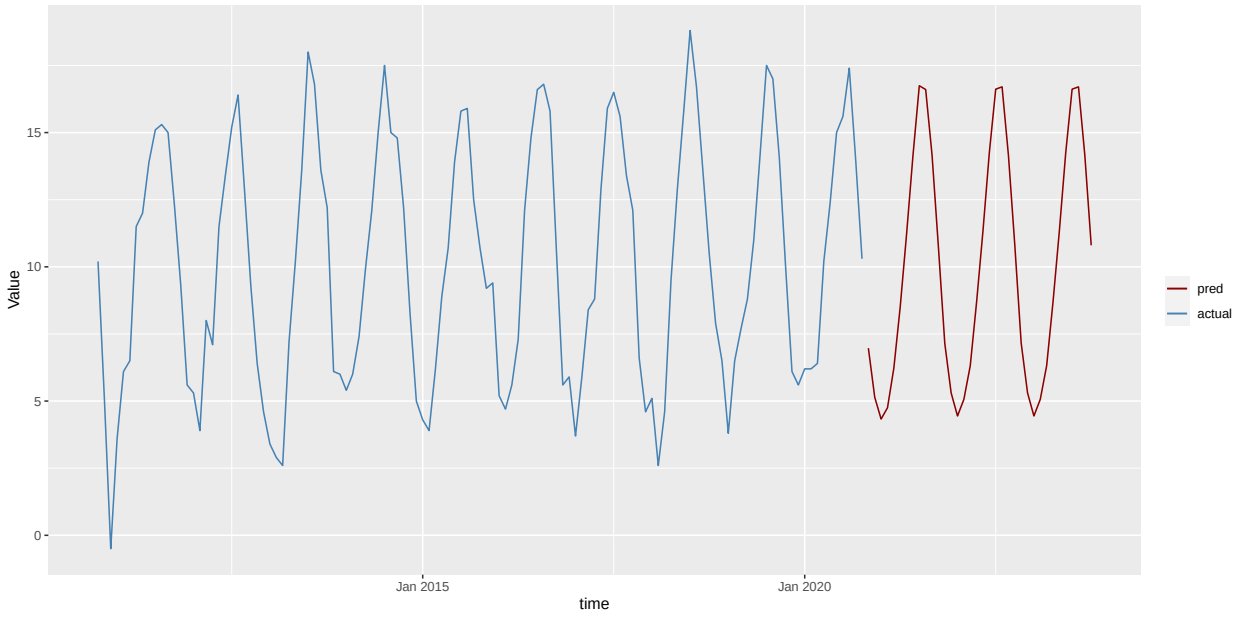


Figure 31: sVARMA 2 years forecast on Mean Temperature

From 2 years forecast plot, it is shown that the predicted values (2 years) of sVARMA model mimics the historic trend well. In addition to this, the predicted values of sVARMA are further examined below by comparing with the 2 months forecast of SARIMA which was identified in the earlier section.

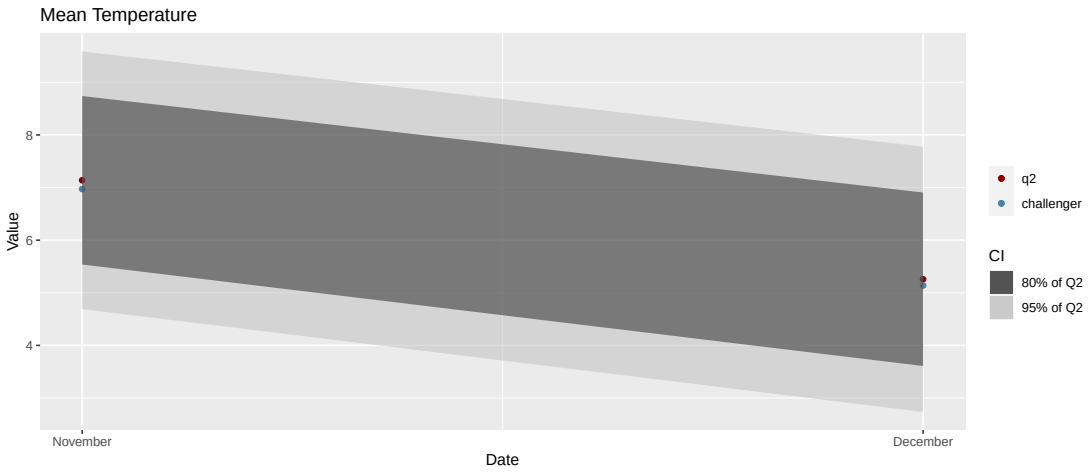


Figure 32: Mean Temperature: Q2 (SARIMA) vs Q4 (challenger, sVARMA)

Based on 2 Months forecast of Mean Temperature of Question 2 (Q2) and Question 4 (Q4) models, the extra 2 months of foretasted points retrieved from Q4 fall within the C.I (incl. 95% and 80%) of Q2 forecasts and the divergence between Q2 and Q4 forecasts is obsereved to be immaterial.

4 Appendix: All code for this report

```
# Library Setup
library(zoo)
library(astsa)
library(forecast)
library(ggplot2)
library(ggfortify)
library(tseries)
library(tidyverse)
library(tsutils)
library(MTS)
library(knitr)

# Data retrieved from Git-hub repository.
df_path = "https://raw.githubusercontent.com/Anko-Jipsa/statistics/master/ST422/"

# Mean Temperature
mt_df <- read.csv(
  paste(df_path, "/MonthlyMeanTemp.csv", sep = ""),
  header = TRUE,
  row.names = "year"
)

# Sunshine
ss_df <- read.csv(
  paste(df_path, "/MonthlySunshine.csv", sep = ""),
  header = TRUE,
  row.names = "year"
)

# Rainfall
rf_df <- read.csv(
  paste(df_path, "/MonthlyRainfall.csv", sep = ""),
  header = TRUE,
  row.names = "year"
)

# Vectorisation of CSV imported dataset
vectorised_ts = function(df) {
  #' Generate a time-series vector, ignoring aggregated level data.
  #'
  #' @param df (data.frame): A data-frame that contains the time-series vector.
  #' @return A time-series vector without aggregated level data.

  start_year = as.numeric(row.names(df)[1])
```

```
  vector_df = as.numeric(t(df[, 1:12])) # Flattening 2d data.
  vector_df = na.omit(vector_df) # Drop null values.
  vector_ts = ts(vector_df,
                 start = c(start_year, 1),
                 frequency = 12) # Assigning time-series.
  return(vector_ts)
}

# Transforming data
mt_ts = vectorised_ts(mt_df)
ss_ts = vectorised_ts(ss_df)
rf_ts = vectorised_ts(rf_df)

####################Q1 & Q2#######################
# Preliminary analysis for identification of Time-series model
prelim_analysis = function(ts_v,
                           title = NULL,
                           start_year = NULL,
                           start_month = NULL) {
  #' Preliminary analysis for Time Series.
  #'
  #' @description
  #' Generate time-series analysis graphs:
  #' 1. Time Series plot (General visualisation)
  #' 2. Box plot (Seasonality & volatility assessment)
  #' 3. Seasonal Plot (Deeper insight on pattern)
  #' 4. Decomposition plot (Assess trend, seasonality and remainder)
  #'
  #' @param ts_v (vector): A Time Series vector.
  #' @param title (str): Title of the plot. Defaults to NULL.
  #' @param start_year (int): A starting year of analysis. Defaults to NULL.
  #' @param start_month (int): A starting month of analysis. Defaults to NULL.

  if (!is.null(start)) {
    ts_v = window(ts_v,
                  start = c(start_year, start_month),
                  end = c(2020, 12))
  }
  layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE))
  plot.ts(ts_v,
          ylab = "Value",
          xlab = "Date",
          main = paste(title, "TS plot", sep = " "))
  boxplot(ts_v ~ cycle(ts_v),
          ylab = "Value",
          xlab = "Month",
```

```
          main = "Box Plot")
  seasonplot(ts_v,
             year.labels = TRUE,
             col = 1:13,
           main = "Seasonal Plot",
           ylab= "Value")
}

# Decomposition analysis required for Time-series model identification
decomp_analysis = function(ts_v,
                           title = NULL,
                           start_year = NULL,
                           start_month = NULL) {
  #' Decomposition analysis for Time Series.
  #'
  #' @description
  #' Generate time-series analysis graphs:
  #' 1. Decomposition plot (Assess trend, seasonality and remainder)
  #'
  #' @param ts_v (vector): A Time Series vector.
  #' @param title (str): Title of the plot. Defaults to NULL.
  #' @param start_year (int): A starting year of analysis. Defaults to NULL.
  #' @param start_month (int): A starting month of analysis. Defaults to NULL.

  if (!is.null(start)) {
    ts_v = window(ts_v,
                  start = c(start_year, start_month),
                  end = c(2020, 12))
  }

  decomp <- stl(ts_v, s.window = 'periodic')
  autoplot(decomp) +  ggtitle("Remainder")
}

# Entire dataset
prelim_analysis(ss_ts, "Sunshine")
decomp_analysis(ss_ts, "Sunshine")

# Dataset from 2000
prelim_analysis(ss_ts, "Sunshine", 2000, 1)
decomp_analysis(ss_ts, "Sunshine", 2000, 1)

# Time-series analysis required for model identification
ts_analysis = function(ts_v,
                       title = NULL,
                       start_year = NULL,
                       start_month = NULL){
```

```
  #' ACF & PACF analysis for Time Series.
  #'
  #' @description
  #' Generate time-series analysis graphs:
  #' 1. ACF
  #' 2. PACF
  #' 3. Seasonal Plot (Deeper insight on pattern)
  #' 4. Decomposition plot (Assess trend, seasonality and remainder)
  #'
  #' @param ts_v (vector): A Time Series vector.
  #' @param title (str): Title of the plot. Defaults to NULL.
  #' @param start_year (int): A starting year of analysis. Defaults to NULL.
  #' @param start_month (int): A starting month of analysis. Defaults to NULL.


  acf2(ts_v, main=title)


  adf.test(ts_v, k=12)
}

ts_analysis(ss_ts, "Sunshine")

# Seasonally differenced time-series
d12_ss_ts = diff(ss_ts, lag = 12)
ts_analysis(d12_ss_ts)

# Grid-search method for model exploration
grid_search_sarima = function(ts_v, non_seasonal_pqd, seasonal_pqd) {
  #' Grid-search function to find the multiplicative seasonal ARIMA model with the best fit.
  #'
  #' @description
  #' Exhaustive approach of derive multiplicative seasonal ARIMA model.
  #' Iterate through each P, Q terms of non-seasonal and seasonal components. (D is fixed.)
  #'
  #' @param ts_v (vector): A Time Series vector.
  #' @param non_seasonal_pqd (vector): Initial P, Q, D elements for non-seasonal component.
  #' @param non_seasonal_pqd (vector): Initial P, Q, D elements for seasonal component.

  cross_non_pqd = expand.grid(seq(max(non_seasonal_pqd[1]-1, 0), non_seasonal_pqd[1] + 1),
                              non_seasonal_pqd[2],
                              seq(max(non_seasonal_pqd[3]-1, 0), non_seasonal_pqd[3] + 1))

  cross_pqd = expand.grid(seq(max(seasonal_pqd[1]-1, 0), seasonal_pqd[1] + 1),
                          seasonal_pqd[2],
                          seq(max(seasonal_pqd[3]-1, 0), seasonal_pqd[3] + 1))
```

```
  # Empty model rank table
  mod_fit = data.frame(AIC = numeric(0),
                       BIC = numeric(0),
                       NON_SEASONAL = numeric(0),
                       SEASONAL= numeric(0))
  count = 1
  for (i in 1:dim(cross_non_pqd)[1]){
    for (j in 1:dim(cross_pqd)[1]){
      skip_to_next <- FALSE

      # Status update prints
      print("************************")
      print(paste("Iteration: ", count, sep = ""))
      print(paste("Non-Seasonal: ", cross_pqd[j,], sep = ""))
      print(paste("Seasonal: ",cross_non_pqd[i,], sep = ""))

      # Error catcher for ARIMA fitting (In case of optimisation error)
      mod = tryCatch(Arima(ts_v,
                           order = as.numeric(cross_non_pqd[i,]),
                           seasonal = list(order = as.numeric(cross_pqd[j,]), period = 12)),
                     error = function(e) { skip_to_next <<- TRUE})
      if(skip_to_next) { next }

      # Table filling
      mod_fit[count, ] = c(mod$aic,
                           mod$bic,
                           paste(as.character(cross_non_pqd[i,]), collapse=","),
                           paste(as.character(cross_pqd[j,]), collapse=","))

      count = count + 1
    }
  }
  # Reordering
  mod_fit = mod_fit %>% arrange(AIC)
  return(mod_fit)
}

# Grid-search
grid_ss = grid_search_sarima(ss_ts,
                             c(0,0,2),
                             c(0,1,1))
opt_non_seasonal = as.numeric(unlist(strsplit(grid_ss[1,3], "\\,"))) # Optimal non-seasonal P, D
opt_seasonal = as.numeric(unlist(strsplit(grid_ss[1,4], "\\,"))) # Optimal seasonal P, D, Q para

# Model fitting
best_guess_ss = Arima(ss_ts,
                      order=c(0,0,2),
```

```
                  seasonal=list(order=c(0,1,1), period=12))

optimal_ss = Arima(ss_ts,
                   order=opt_non_seasonal,
                   seasonal=list(order=opt_seasonal, period=12))

auto_arima_ss = auto.arima(ss_ts)

# Model comparison
best_guess_ss # Best guess
optimal_ss # Grid-search
auto_arima_ss # auto.arima

# Residual analysis
checkresiduals(best_guess_ss)
tsdiag(best_guess_ss)

# Custom ggplot function
gg_forecast = function(model_output, n_forecast, include, title){
  #' Custom function for clear ggplot of forecast object.
  #'
  #' @param model_output (obj): a time series or time series model for which forecasts are requi
  #' @param n_forecast (int): Number of periods for forecasting
  #' @param include (int): Number of periods for previous actuals
  #' @param title (str): Title of the plot.
  #'

  # Actual & Forecasts
  forecast_obj = forecast(model_output, n_forecast)
  actuals = tail(forecast_obj$x, include)
  predicted = forecast_obj$mean

  # Date
  date_window = c(as.Date(actuals), as.Date(predicted))

  # Combined data.frame
  output = data.frame(time = date_window,
                      actual = c(actuals, rep(NA,length(predicted))),
                      pred = c(rep(NA,length(actuals)), predicted),
                      low_95 = c(rep(NA,length(actuals)), forecast_obj$lower[,2]),
                      up_95 = c(rep(NA,length(actuals)), forecast_obj$upper[,2]),
                      low_80 = c(rep(NA,length(actuals)), forecast_obj$lower[,1]),
                      up_80 = c(rep(NA,length(actuals)), forecast_obj$upper[,1])
                      )

  # ggplot
  p = ggplot(output, aes(x=time), na.rm=TRUE) +  geom_line(aes(y=actual, color="actual"), na.rm=
```

```
  # Convert to scatter plots for forecast periods less than 10
    if (n_forecast<10){
      p = p + geom_point(aes(y=pred, color="pred"), na.rm=TRUE)
      } else {
        p = p + geom_line(aes(y=pred, color="pred"), na.rm=TRUE)
      }

  # Confidence interval
  p = p +
    geom_ribbon(aes(ymin=low_95, ymax=up_95, fill="95% CI"),linetype=2, alpha=0.5) +
    geom_ribbon(aes(ymin=low_80, ymax=up_80, fill="80% CI"), linetype=2, alpha=0.5)

  # Further aesthetics
  p = p +
    scale_colour_manual("",
                        breaks = c("pred", "actual"),
                        values = c("darkred", "steelblue")) +
    scale_fill_manual(values=c("grey12", "grey"), name="fill") +
    ylab("Value") + xlab("Date") + ggtitle(title)

  return(p)
}

# Forecasting
gg_forecast(best_guess_ss, n_forecast=2, include=12, title="2 Months forecast plot of Sunshine")
forecast(best_guess_ss, 2)
gg_forecast(best_guess_ss, n_forecast=24, include=120, title="2 years forecast plot of Sunshine"

# Description of the following codes is identical to the above case
prelim_analysis(rf_ts, "Rainfall")
decomp_analysis(rf_ts, "Rainfall")
prelim_analysis(rf_ts, "Rainfall", 2010, 1)
decomp_analysis(rf_ts, "Rainfall", 2000, 1)

# Time-series analysis
ts_analysis(rf_ts, "Rainfall")

# Seasonally differenced Time-series analysis
d12_rf_ts = diff(rf_ts, lag = 12)
ts_analysis(d12_rf_ts)

# Grid-search
grid_rf = grid_search_sarima(rf_ts,
                             c(1,0,1),
                             c(0,1,1))
opt_non_seasonal = as.numeric(unlist(strsplit(grid_rf[1,3], "\\,"))) # Optimal non-seasonal P, D
opt_seasonal = as.numeric(unlist(strsplit(grid_rf[1,4], "\\,"))) # Optimal seasonal P, D, Q para
```

```
# Comparison
best_guess_rf = Arima(rf_ts,
                      order=c(1,0,1),
                      seasonal=list(order=c(0,1,1), period=12))

optimal_rf = Arima(rf_ts,
                   order=opt_non_seasonal,
                   seasonal=list(order=opt_seasonal, period=12))

auto_arima_rf = auto.arima(rf_ts)

# Models
best_guess_rf # Best guess
optimal_rf # Grid-search
auto_arima_rf # auto.arima

# Residual analysis
checkresiduals(optimal_rf)
tsdiag(optimal_rf)

# Forecast
gg_forecast(optimal_rf, n_forecast=2, include=12, title="2 months forecast plot of Rainfall")
forecast(optimal_rf, 2)
gg_forecast(optimal_rf, n_forecast=24, include=120, title="2 years forecast plot of Rainfall")

# Preliminary analysis
prelim_analysis(mt_ts, "Mean Temperature")
decomp_analysis(mt_ts, "Mean Temperature")
prelim_analysis(mt_ts, "Mean Temperature", 2010, 1)

decomp_analysis(mt_ts, "Mean Temperature", 2000, 1)

# Time-series analysis
ts_analysis(mt_ts, "Mean Temperature")

# Seasonally differenced time-series analysis
d12_mt_ts = diff(mt_ts, lag = 12)
ts_analysis(d12_mt_ts)

# Grid-search
grid_mt = grid_search_sarima(mt_ts,
                             c(0,0,2),
                             c(0,1,1))
opt_non_seasonal = as.numeric(unlist(strsplit(grid_mt[1,3], "\\,"))) # Optimal non-seasonal P, D
opt_seasonal = as.numeric(unlist(strsplit(grid_mt[1,4], "\\,"))) # Optimal seasonal P, D, Q para

# Model fitting
```

```
best_guess_mt = Arima(mt_ts,
                      order=c(0,0,2),
                      seasonal=list(order=c(0,1,1), period=12))

optimal_mt = Arima(mt_ts,
                      order=opt_non_seasonal,
                      seasonal=list(order=opt_seasonal, period=12))

auto_arima_mt = auto.arima(mt_ts)

# Comparison
best_guess_mt # Best guess
optimal_mt # Grid-search
auto_arima_mt # auto.arima

# Residual analysis
checkresiduals(optimal_mt)
tsdiag(optimal_mt)

# Forecasting
gg_forecast(optimal_mt, n_forecast=2, include=12, title="2 months forecast plot of Mean Temperat
forecast(optimal_mt, 2)
gg_forecast(optimal_mt, n_forecast=24, include=120, title="2 years forecast plot of Mean Tempera

#####################Q3#######################
# Rolling window function
rolling_window = function(ts_v, M=210,A=120){
  #' Function to calculate the goodness of fit and 2 months forecast for given rolling window pa
  #' This function is a satellite function for the iteration function below.
  #'
  #' @description
  #' For fitting the model, this function deploys auto.arima().
  #' Goodness of fit metrics are following: RMSE, MAE.
  #'
  #' @param ts_v (vector): A Time Series vector.
  #' @param M (int): A size of window for model fitting.
  #' @param A (int): A size of window for validation.

  # Preset
  n = length(ts_v)
  N = ceiling((n-M-A)/A)

  pred_val = list()
  fitted_val = list()
  for (i in 1:N){
    # Start and End window for development & validation
    dev_start = (i-1)*A+1
```

```
    dev_end = min((i-1)*A+M, n)

    val_start = dev_end + 1
    val_end = min(dev_end+A, n)

    # Model fitting
    dev_sample = ts(ts_v[dev_start:dev_end], frequency=12)
    fit = auto.arima(dev_sample)

    # Prediction
    actuals = ts_v[(val_start):(val_end)]
    pred_obj = forecast(fit, A)

    # Status
    print("*********************************")
    print(paste("Iteration: ", i, sep=""))
    print(paste("Modelling window: ", dev_start, "~", dev_end, sep=""))
    print(paste("Validation window: ", val_start, "~", val_end, sep=""))

    # Validation
    if (i==1) {
      metric = accuracy(pred_obj, actuals)
      } else {
        metric = metric + accuracy(pred_obj, actuals)
      }
  }

  # Averaging & Aligning the validation metrics data
  metric = data.frame(metric/N)
  metric$Window = M
  metric$Sample = n
  metric = metric[2, c( "Window", "Sample", "RMSE", "MAE")]
  row.names(metric) = NULL

  return(metric)
}

# List of different window values (M)
window_list = c(36, 72, 108, 144, 180)

# Window selection fuction
window_selection = function(ts_v, window_list, A){
  #' Using 'rolling_window' function to find the optimal window length.
  #'
  #' @description
  #' Split data into train (development) and test (validation), 75% and 25% accordingly.
  #'
```

```
  #' @param ts_v (vector): A Time Series vector.
  #' @param window_list (list): A list of different window value for model fitting.
  #' @param A (int): A size of window for validation.

  # 75% split
  split = floor(length(ts_v)*0.75)
  #Create Train Set
  train <- window(ts_v, end = time(ts_v)[split])
  #Create Test (Validation) Set
  test <- window(ts_v, start = time(ts_v)[split+1])

  # Empty variables
  output_df = data.frame(Window = numeric(0),
                         Sample = numeric(0),
                         RMSE = numeric(0),
                         MAE = numeric(0))
  fitted_list = list()

  # Iterative approach
  for (i in seq_along(window_list)){
    print(paste("Current iteration window (TRAIN): ", window_list[i], sep=""))
    train_output = rolling_window(train, window_list[i], A)

    print(paste("Current iteration window (TEST): ", window_list[i], sep=""))
    tests_output = rolling_window(test, window_list[i], A)

    output_df[i*2-1,] = train_output # Train sample output
    output_df[i*2,] = tests_output # Test sample output
  }

  return(output_df)
}

# Mean Temperature, A = 120
mt_window36 = window_selection(mt_ts, window_list, A=36)

# Rainfall, A = 120
rf_window36 = window_selection(rf_ts, window_list, A=36)

# Sunshine, A = 120
ss_window36 = window_selection(ss_ts, window_list, A=36)

kable(ss_window36, caption = "Optimal window searching result for Sunshine (Validation period, A
kable(rf_window36, caption = "Optimal window searching result for Rainfall (Validation period, A
kable(mt_window36, caption = "Optimal window searching result for Mean Temperature (Validation p

rolling_window_forecast = function(ts_v, M, n_forecast){
```

```
  #' A function to calculate "N" steps forecasts given "M" window using auto.arima()
  #' @param ts_v (vector): A Time Series vector.
  #' @param M (int): A size of window for model fitting.
  #' @param n_forecast (int): "N" steps forecast.

  model = auto.arima(tail(ts_v, M))
  output = forecast(model, n_forecast)
  return(output$mean)
}

roll_window_ss = rolling_window_forecast(ss_ts, 144, 2)
roll_window_rf = rolling_window_forecast(rf_ts, 72, 2)
roll_window_mt = rolling_window_forecast(mt_ts, 144, 2)
# Custom function for comparing 2 months forecast of Q2 results with the new model.
forecast_compare = function(model_output, compare_output, n_forecast, title){
  #' Custom function for clear ggplot of forecast object.
  #'
  #' @param model_output (obj): A time series or time series model for which forecasts are requi
  #' @param compare_output (obj): A time series to be compared with model output forecast
  #' @param n_forecast (int): Number of periods for forecasting
  #' @param title (str): Title of the plot.
  #'

  # Actual & Forecasts
  forecast_obj = forecast(model_output, n_forecast)
  predicted = forecast_obj$mean
  compare_pred = compare_output[1:n_forecast]

  # Confidence interval
  c_95low = forecast_obj$lower[,2]
  c_95up = forecast_obj$upper[,2]
  c_80low = forecast_obj$lower[,1]
  c_80up = forecast_obj$upper[,1]

  # Date
  date_window = as.Date(as.yearmon(time(predicted)))

  # Combined data.frame
  output = data.frame(time = date_window,
                      q2 = as.matrix(predicted),
                      challenger = as.matrix(compare_pred),
                      low_95 = c_95low,
                      up_95 = c_95up,
                      low_80 = c_80low,
                      up_80 = c_80up)

  # ggplot
```

```
  p = ggplot(output, aes(x=time)) +
    geom_point(aes(y=q2, color="q2")) +
    geom_point(aes(y=challenger, color="challenger"), na.rm=TRUE) +
    theme(plot.margin = margin(2, 2, 2, 2, "cm")) + scale_x_date(date_breaks = "1 month",
                                                                 date_labels = "%B")

  # Confidence interval
  p = p +
    geom_ribbon(aes(ymin=low_95, ymax=up_95, fill="95% of Q2"),linetype=2, alpha=0.5) +
    geom_ribbon(aes(ymin=low_80, ymax=up_80, fill="80% of Q2"), linetype=2, alpha=0.5)

  # Further aesthetics
  p = p +
    scale_colour_manual("",
                        breaks = c("q2", "challenger"),
                        values = c("darkred", "steelblue")) +
    scale_fill_manual(values=c("grey12", "grey"), name="CI") +
    ylab("Value") + xlab("Date") + ggtitle(title)

  return(p)
}

# Question 2 and Question 3 forecasts comparison plots
forecast_compare(best_guess_ss,
                 roll_window_ss,
                 n_forecast=2,
                 title="Sunshine")
forecast_compare(optimal_rf,
                 roll_window_rf,
                 n_forecast=2,
                 title="Rainfall")
forecast_compare(optimal_mt,
                 roll_window_mt,
                 n_forecast=2,
                 title="Mean Temperature")

####################Q4#######################
# Combined data.frame for Vector ARMA modelling
varma_dat = data.frame(na.omit(cbind(ss_ts,rf_ts,mt_ts)))

# sVARMA model components
seasonal_pqd = c(1,0,1)
non_seasonal_pqd = c(0,1,1)

# Grid-search
grid_search_svarma = function(ts_df, non_seasonal_pqd, seasonal_pqd, show_step=TRUE) {
  #' Grid-search function to find the multiplicative seasonal sVARMA model with the best fit.
```

```r
#'
#' @section !WARNING!
#' This function is expected to consume a significant time.
#' Hence, the best way to validate the final result is to plug in the model parameters into sV
#'
#' @description
#' Exhaustive approach of derive multiplicative seasonal sVARMA model.
#' Iterate through each P, Q terms of non-seasonal and seasonal components. (D is fixed.)
#'
#' @param ts_df (data.frame): A Time Series dataframe
#' @param non_seasonal_pqd (vector): Initial P, Q, D elements for non-seasonal component.
#' @param non_seasonal_pqd (vector): Initial P, Q, D elements for seasonal component.
# Ignore all warning messages.
options(warn=2)
cross_non_pqd = expand.grid(seq(max(non_seasonal_pqd[1]-1, 0), non_seasonal_pqd[1] + 1),
                            non_seasonal_pqd[2],
                            seq(max(non_seasonal_pqd[3]-1, 0), non_seasonal_pqd[3] + 1))

cross_pqd = expand.grid(seq(max(seasonal_pqd[1]-1, 0), seasonal_pqd[1] + 1),
                        seasonal_pqd[2],
                        seq(max(seasonal_pqd[3]-1, 0), seasonal_pqd[3] + 1))

# Empty model rank table
mod_fit = data.frame(AIC = numeric(0),
                     BIC = numeric(0),
                     NON_SEASONAL = numeric(0),
                     SEASONAL= numeric(0))
count = 1
for (i in 1:dim(cross_non_pqd)[1]){
  for (j in 1:dim(cross_pqd)[1]){
    skip_to_next <- FALSE
    # Status update prints
    if (show_step){
      print(paste("Step", count, sep = ": "))
    }

    # Error catcher for ARIMA fitting (In case of optimisation error)
    mod = tryCatch(sVARMA(ts_df,
                          order = as.numeric(cross_non_pqd[i,]),
                          s = 12,
                          sorder = as.numeric(cross_pqd[j,])),
                   error = function(e) { skip_to_next <<- TRUE})
    if(skip_to_next) { next }

    # Table filling
    mod_fit[count, ] = c(mod$aic,
                         mod$bic,
```

```
                            paste(as.character(cross_non_pqd[i,]), collapse=","),
                            paste(as.character(cross_pqd[j,]), collapse=","))

      count = count + 1
    }
  }
  # Reordering
  mod_fit = mod_fit %>% arrange(AIC)
  options(warn=1)

  return(mod_fit)
}

# NOTE: This code takes a considerable time to execute.
# Hence, in order to compare the results, it is recommended to run individual model separately f
sVarma_output = grid_search_svarma(varma_dat, seasonal_pqd, non_seasonal_pqd)

# Grid-search output
kable(sVarma_output, caption = "Seasonal VARMA Iteration table")


# Model fitting
sVARMA_final = sVARMA(varma_dat, order=c(1,0,1), sorder=c(0,1,2))
sVARMA_est = sVARMApred(sVARMA_final, 0, h=36)

# Mean Temperature prediction
svarma_mt_pred = sVARMA_est$pred[,3]
svarma_mt_err = sVARMA_est$se.err[,3]
svarma_mt_ts = sVARMA_est$pred[,3]
svarma_mt_pred = ts(sVARMA_est$pred[,3], start=c(2020, 11), frequency=12)

date_window = as.yearmon(seq(as.Date(as.yearmon(time(mt_ts))[1]),
                             as.Date(as.yearmon(time(svarma_mt_pred))[length(svarma_mt_ts)]),
                             by = "months"), format = "%Y-%M-%d")

output = data.frame(time = date_window,
                    actual = c(mt_ts, rep(NA,length(svarma_mt_ts))),
                    pred = c(rep(NA,length(mt_ts)), svarma_mt_ts))

# Visualisation
ggplot(output[1522:dim(output)[1],], aes(x=time)) + ylab("Value") +
  geom_line(aes(y=pred, color="pred")) +
  geom_line(aes(y=actual, color="actual")) +
  scale_colour_manual("",
                      breaks = c("pred", "actual"),
                      values = c("darkred", "steelblue"))
```

```
# Question 2 and Question 4 forecasts comparison plot
forecast_compare(optimal_mt,
                 svarma_mt_pred,
                 n_forecast=2,
                 title="Mean Temperature")
```