TS_Final_Project

Vishal Vincent Joseph

11/15/2022

1) Importing libraries

```
library(tidyverse)
library(tseries)
library(fpp)
library(ggplot2)
library(forecast)
library(arfima)
library(TSA)
```

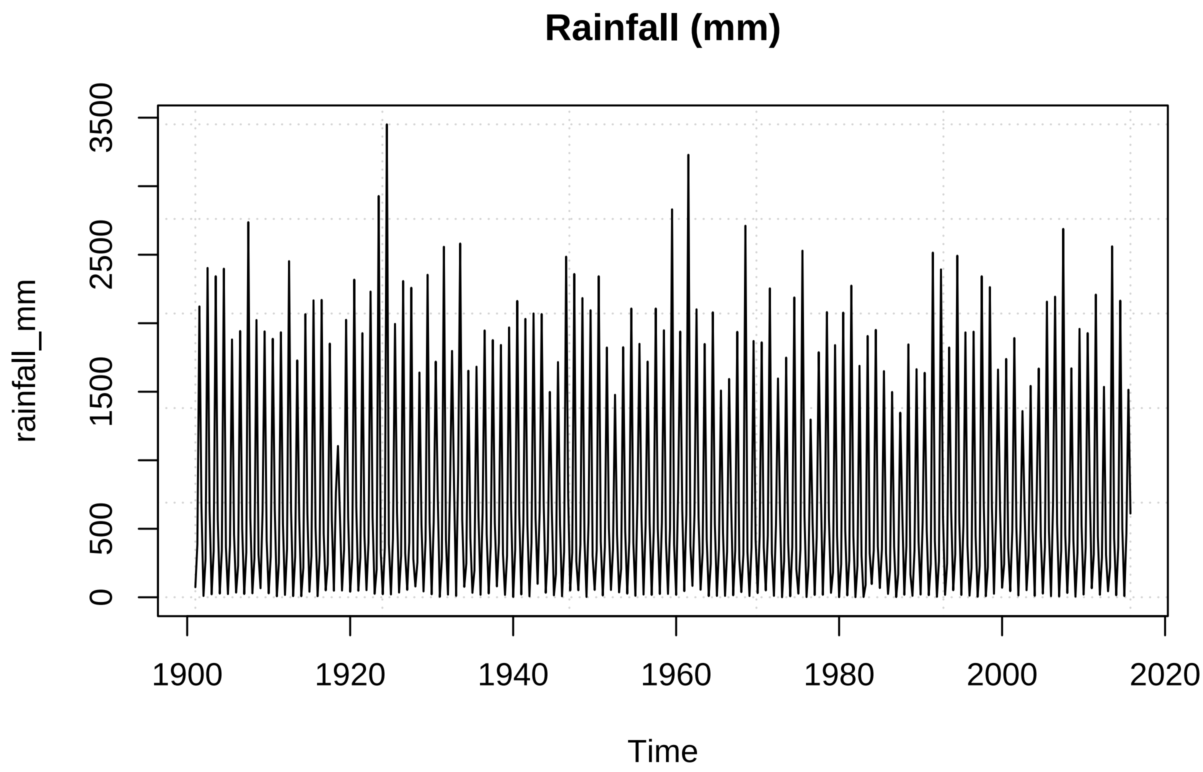
2) Data preparation

```
# loading raw csv
rainfall_all <- read_csv("rainfall in india 1901-2015.csv")

# filtering to quarterly 'Kerala' data
rainfall_KL <- rainfall_all %>%
  filter(SUBDIVISION == "KERALA") %>%
  select(c(2, 16:19)) %>%
  pivot_longer(!YEAR, names_to = "month", values_to = "rainfall_mm")

rainfall_ts <- ts(rainfall_KL['rainfall_mm'], start=1901, frequency=4)
```

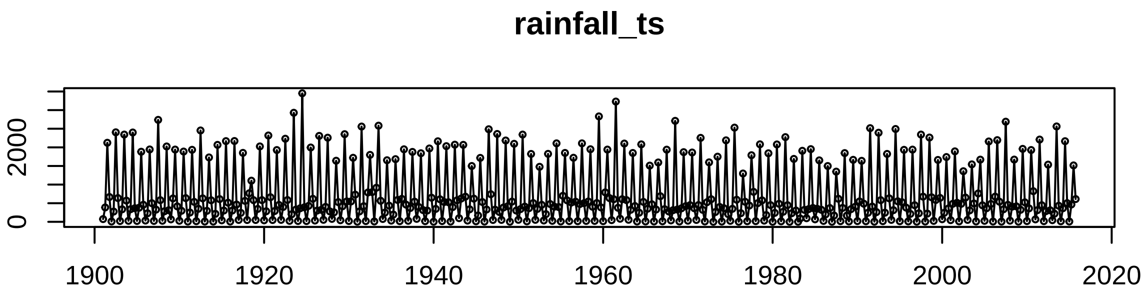
3) Exploratory Data Analysis

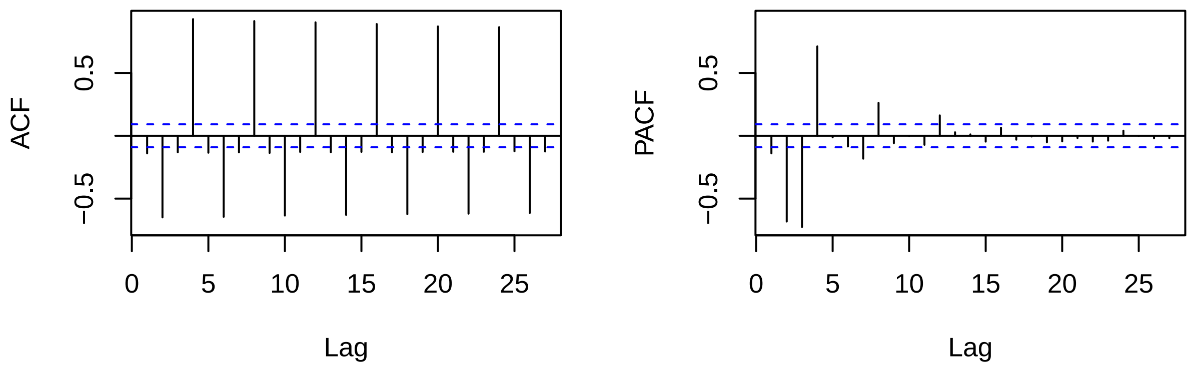
Basic TS plotting

```
# visualising TS plot
plot(rainfall_ts, main="Rainfall (mm)", panel.first = grid())
```




```r
# visualising TS plot with TS components
tsdisplay(rainfall_ts)
```

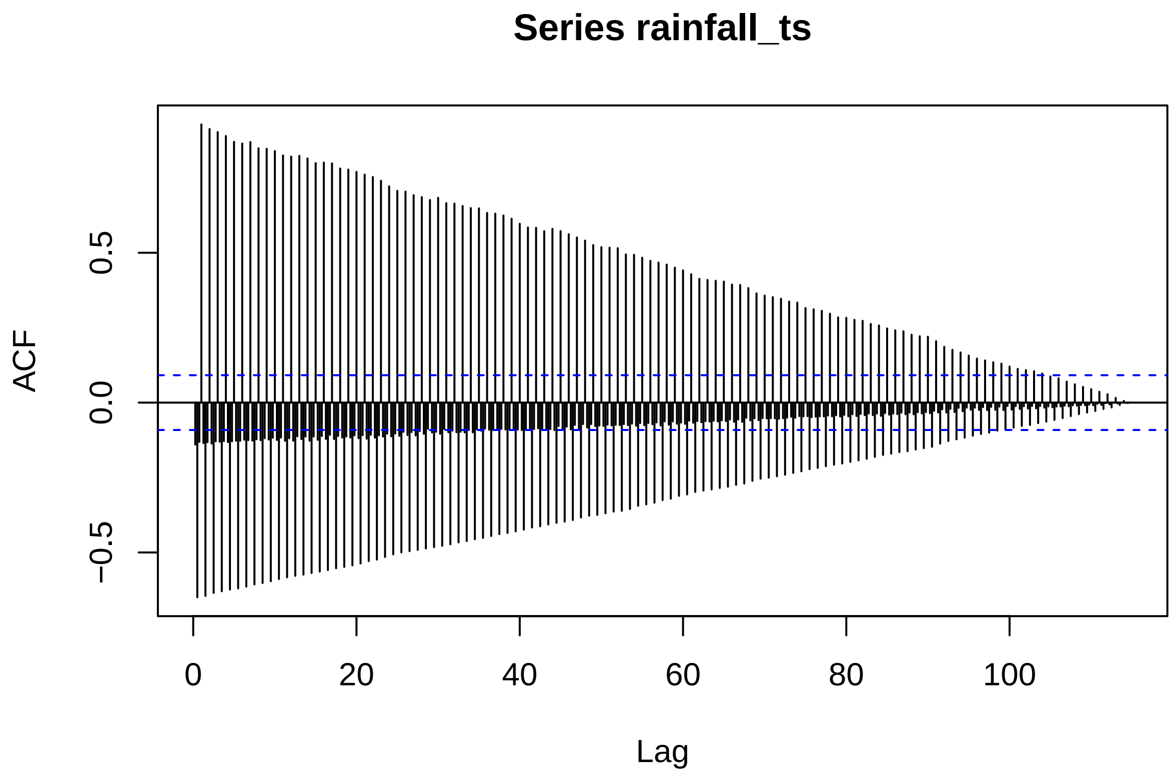




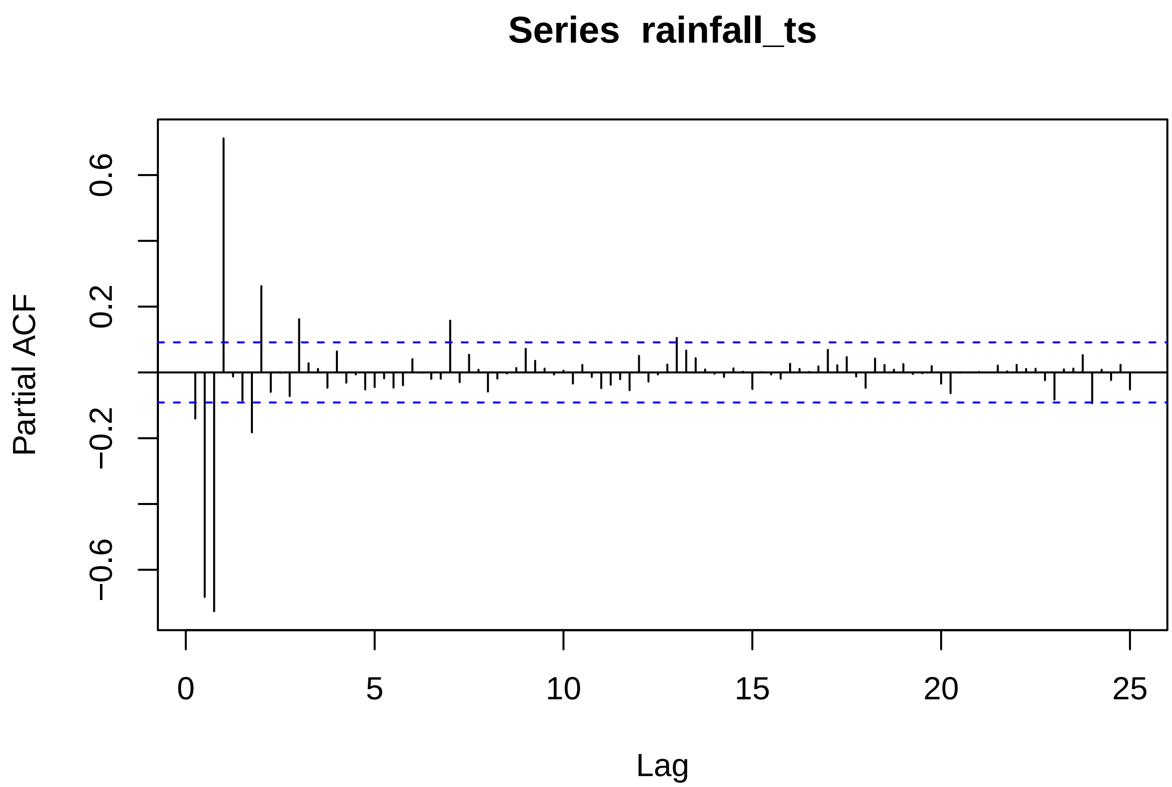
Dataset Characteristics: 1) No visible trend 2) From a first look at the chart, it looks like quarterly seasonality is present 3) There is varying variance and data will require Box-Cox transformation 4) Even without running any tests, we could come to the conclusion that data is non-stationary 5) The ACF chart seems to appear constant due to only 25 lags being displayed, and it looks like there is probably a slow decay. It will require a deeper look with greater number of lags to be certain. The PACF drops off around lag 4. At this point the process seems like an AR(4) process but such a conclusion would make sense only after making the data stationary.

In-depth view of ACF and PACF

```
acf(rainfall_ts, 500)
```



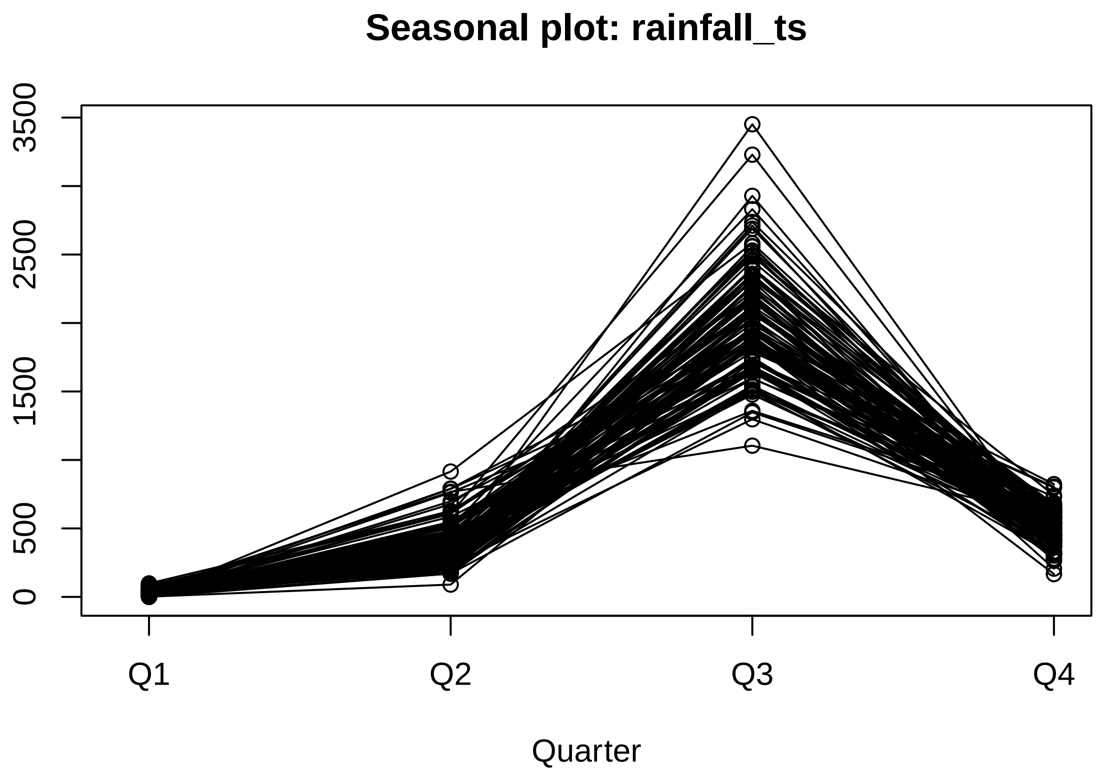

```
pacf(rainfall_ts, 100)
```



Thus, this proves that the ACF plot is slowly decaying and is an indicator of non-stationarity. It also indicates a long-term memory and ARFIMA might be a suitable option here.

Seasonal plot

```
seasonplot(rainfall_ts)
```



Comments: For most years, the peak seems to be at Q3, which coincides with the monsoon season.

4) Stationarity analysis

Checking stationarity before Box-Cox

```
kpss.test(rainfall_ts)
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  rainfall_ts
## KPSS Level = 0.13073, Truncation lag parameter = 5, p-value = 0.1
```

```
adf.test(rainfall_ts)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  rainfall_ts
## Dickey-Fuller = -6.0883, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

Thus, the data is stationary even before applying any Box-Cox transformation.

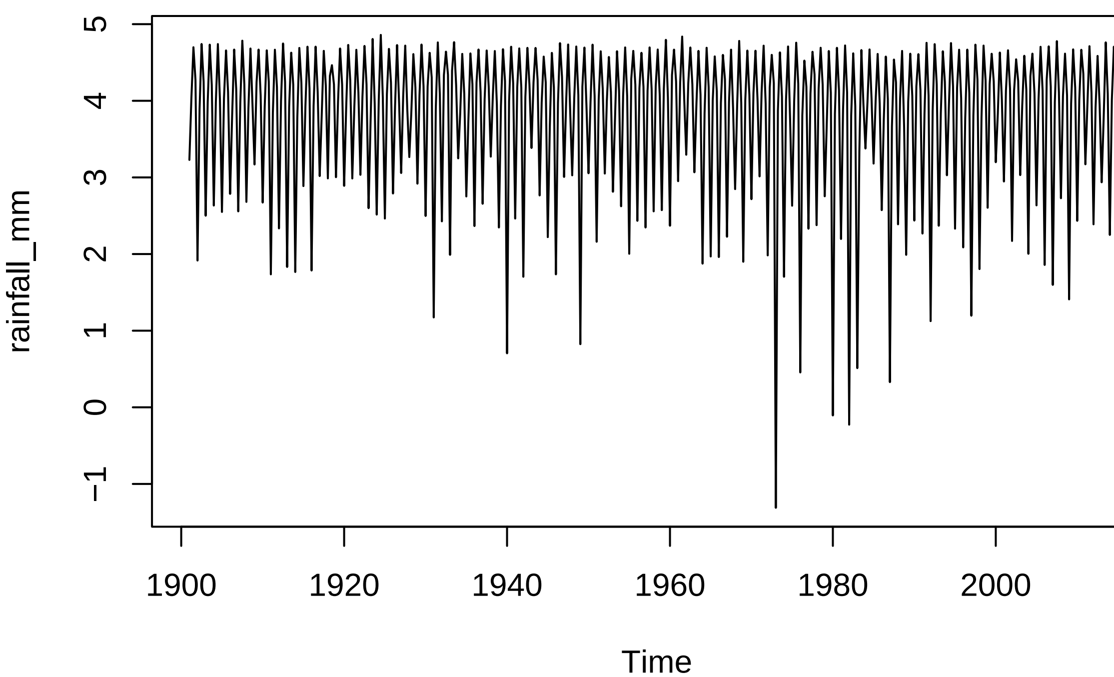
Box-Cox transformation

```r
# checking for Box-Cox transformation
BoxCox.lambda(rainfall_ts)
```

Estimating lambda

```
## [1] -0.1436275
```

```r
# transforming raw data
rainfall_ts_BC <- BoxCox(rainfall_ts, lambda = -0.14)

# plotting BC transformed data
plot(rainfall_ts_BC)
```



Transforming the data

Test for stationarity

```
kpss.test(rainfall_ts_BC)
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  rainfall_ts_BC
## KPSS Level = 0.34244, Truncation lag parameter = 5, p-value = 0.1
```

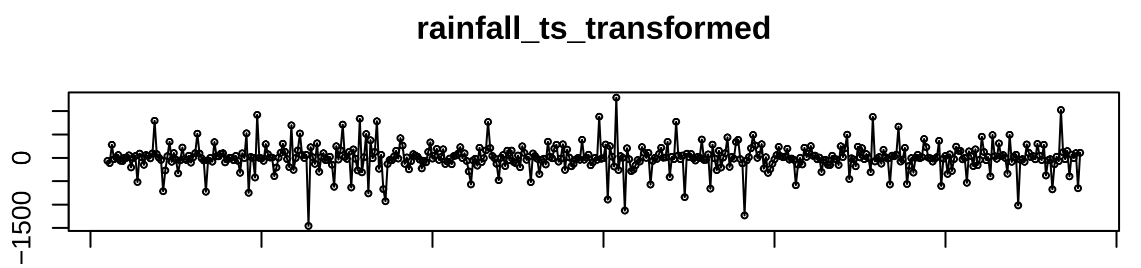
```
adf.test(rainfall_ts_BC)
```

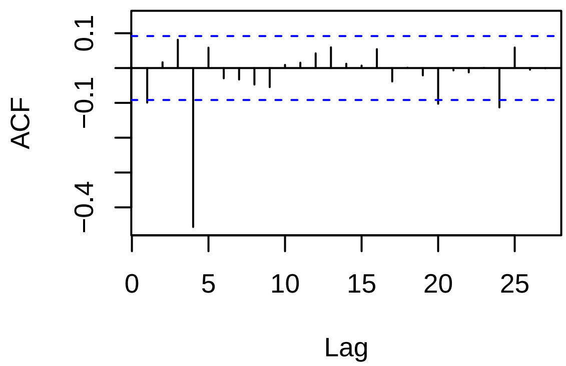
```
##
##  Augmented Dickey-Fuller Test
##
## data:  rainfall_ts_BC
## Dickey-Fuller = -6.0847, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

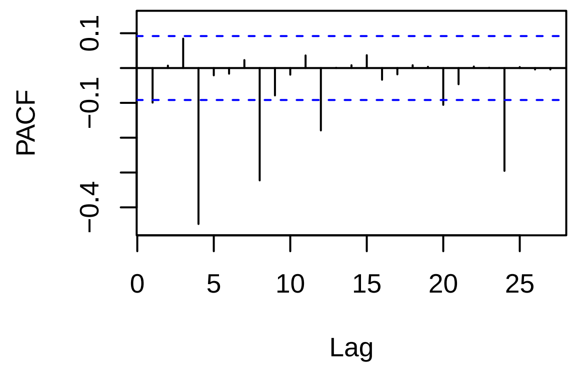
Comments: From both tests above, it is evident that the data is stationary both before and after BC and does not require any differencing for trend or level. However, de-seasonalization is necessary and that is shown below.

Estimating de-seasonalization differencing level (without BC) and transforming data

```
rainfall_ts_transformed <- diff(rainfall_ts, 4)
tsdisplay(rainfall_ts_transformed)
```







Thus, a differencing of lag 4 results in de-seasonalization.

Final verification for stationarity

```
kpss.test(rainfall_ts_transformed)
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  rainfall_ts_transformed
## KPSS Level = 0.0075204, Truncation lag parameter = 5, p-value = 0.1
```

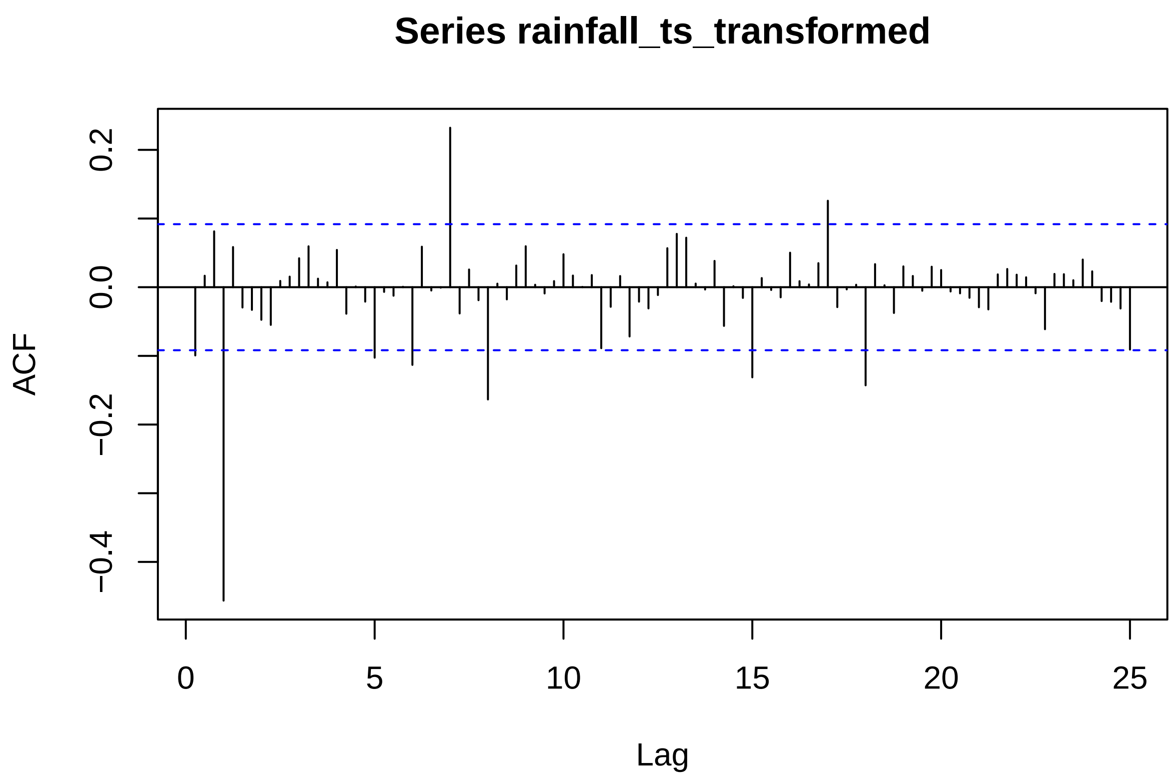
```
adf.test(rainfall_ts_transformed)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  rainfall_ts_transformed
## Dickey-Fuller = -12.426, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

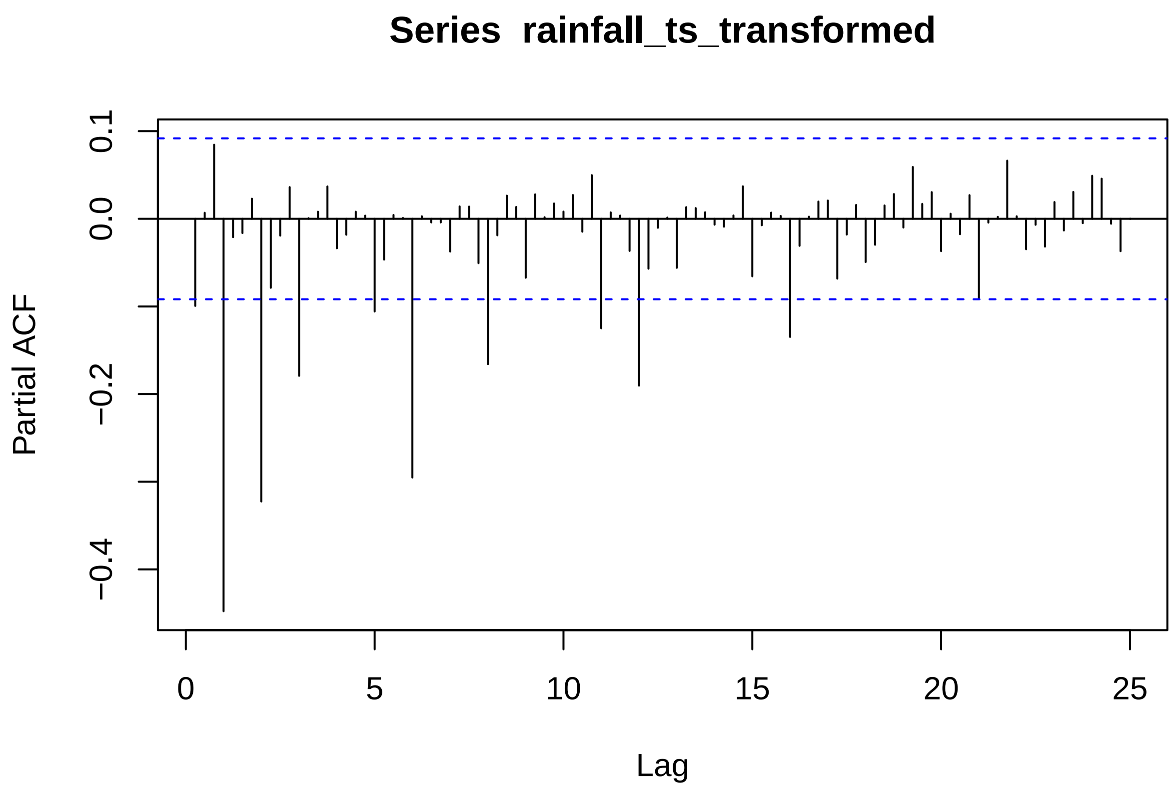
Thus, this completes the transformation to a stationary process.

Plotting ACF and PACF for stationary data

```
# ACF
acf(rainfall_ts_transformed, 100)
```



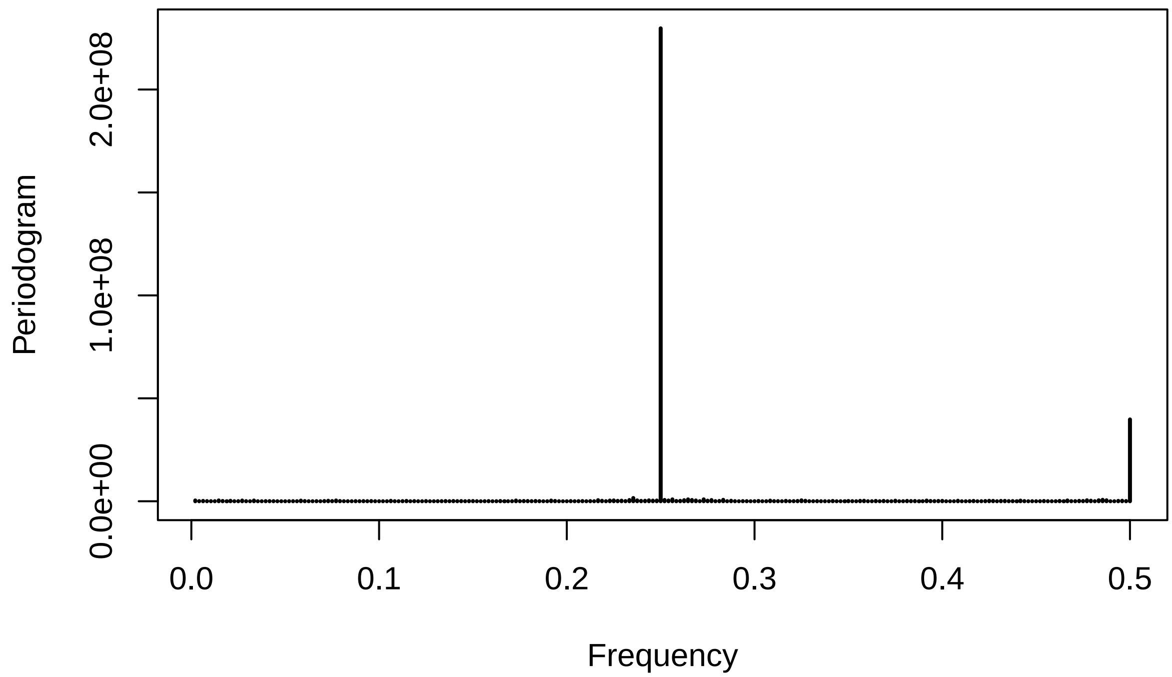

```
# PACF
pacf(rainfall_ts_transformed, 100)
```



Final comments on data: When considering the seasonality component, the PACF decays exponentially with most significant lags at the seasonal lags of 4, 8, etc while the ACF drops off abruptly post the seasonal lag of 4. This points to the fact that the stationary process is most probably an ARIMA(0,0,0)(0,1,1).

5) Spectral analysis

```
periodogram(rainfall_ts)
```

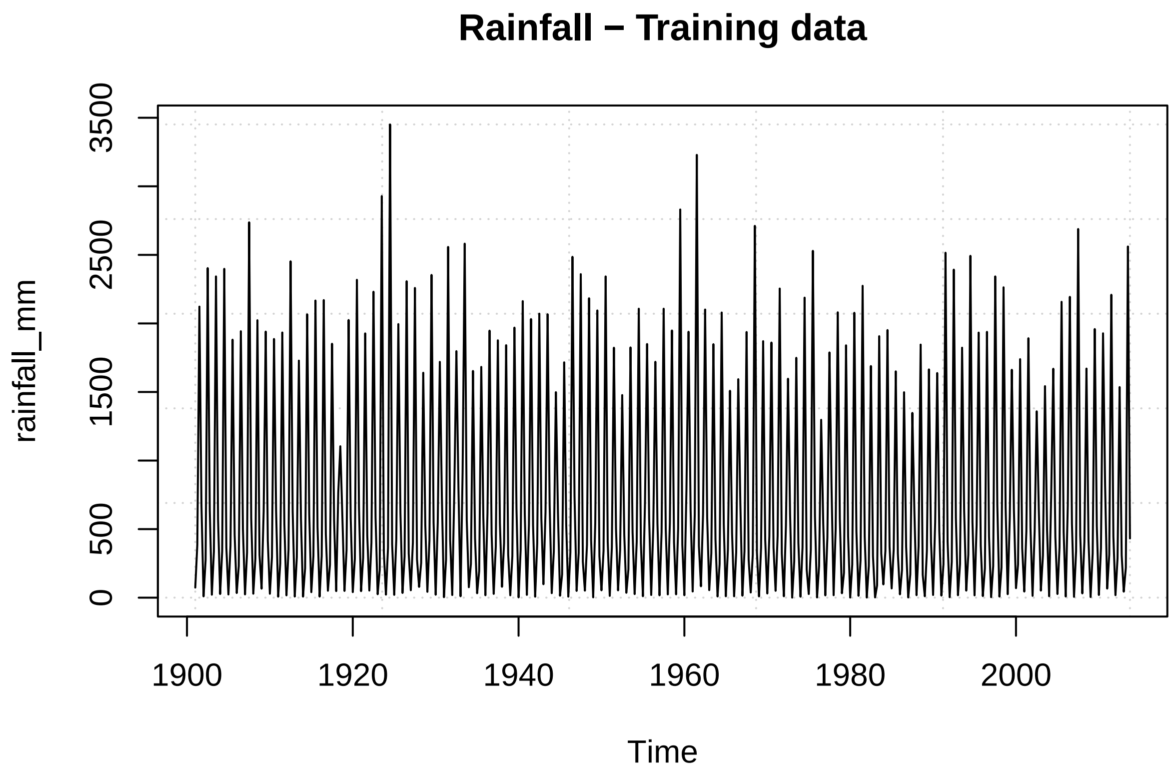


The frequency corresponding to the peak is ~0.25, indicating annual seasonality for the quarterly data.

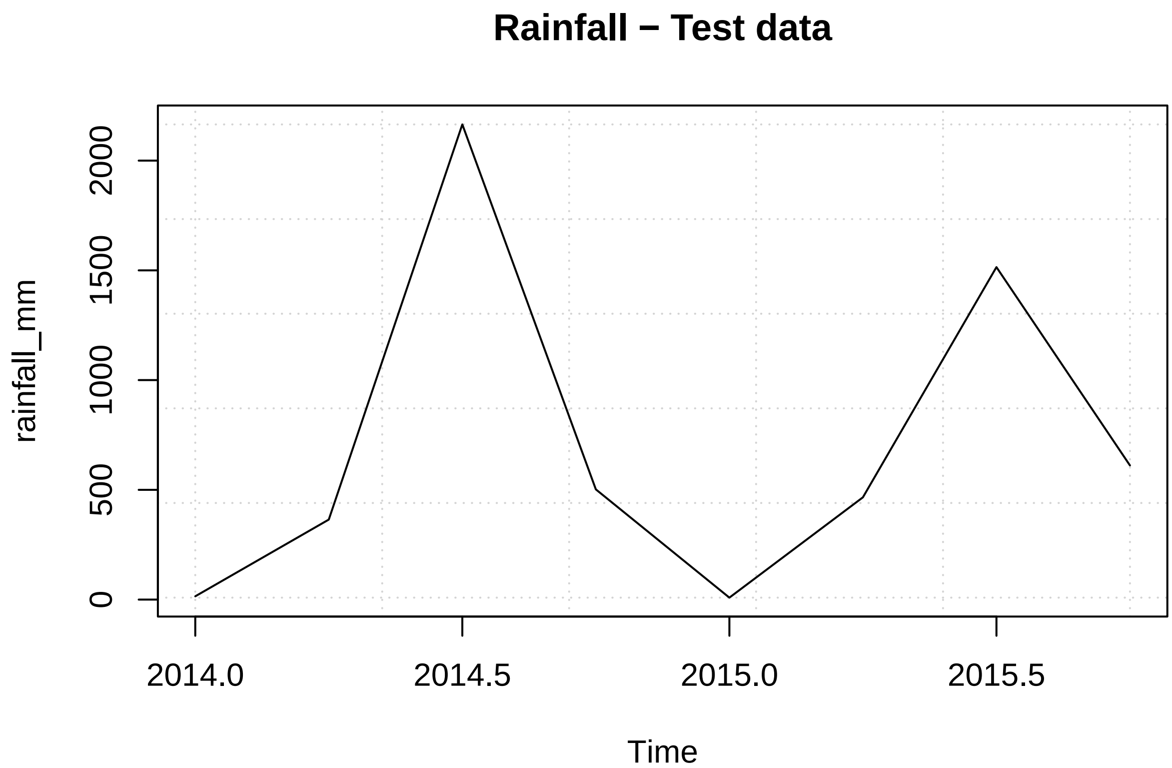
6) Model building

Splitting into train and test & visualizing

```
train_ts <- window(rainfall_ts, start = c(1901, 1), end = c(2013,4))
test_ts <- window(rainfall_ts, start = c(2014, 1), end = c(2015, 4))

plot(train_ts, main="Rainfall - Training data", panel.first = grid())
```

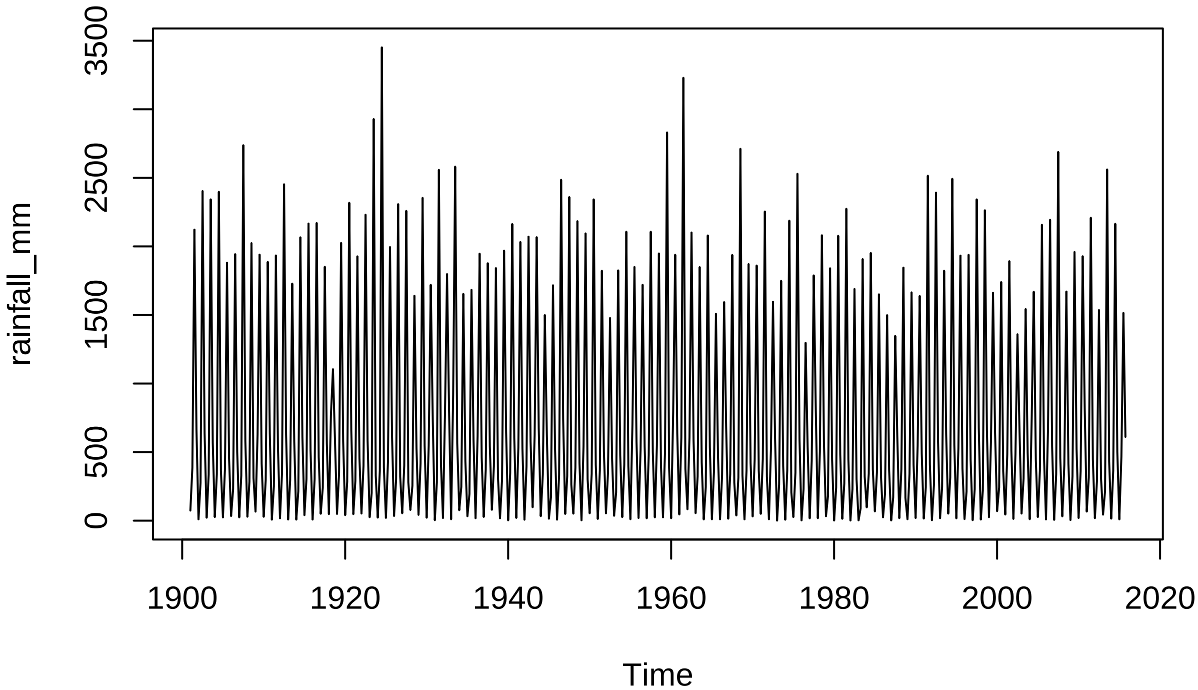



```
plot(test_ts, main="Rainfall - Test data", panel.first = grid())
```



6.1) Classical decomposition model

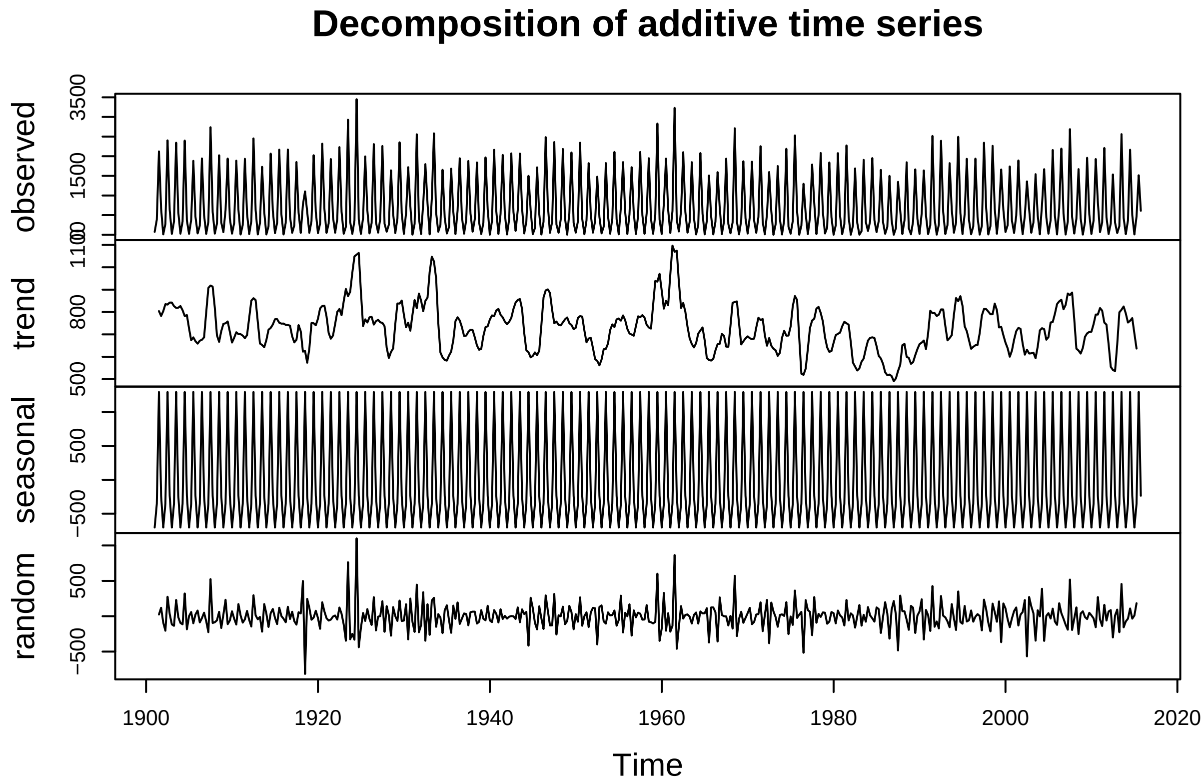
```
plot(rainfall_ts)
```



Looking at the chart above of the raw data, it is unclear whether the seasonality is additive or multiplicative. Thus, we will need to explore both types for the decomposition model. However, it is very clear that no trend exists.

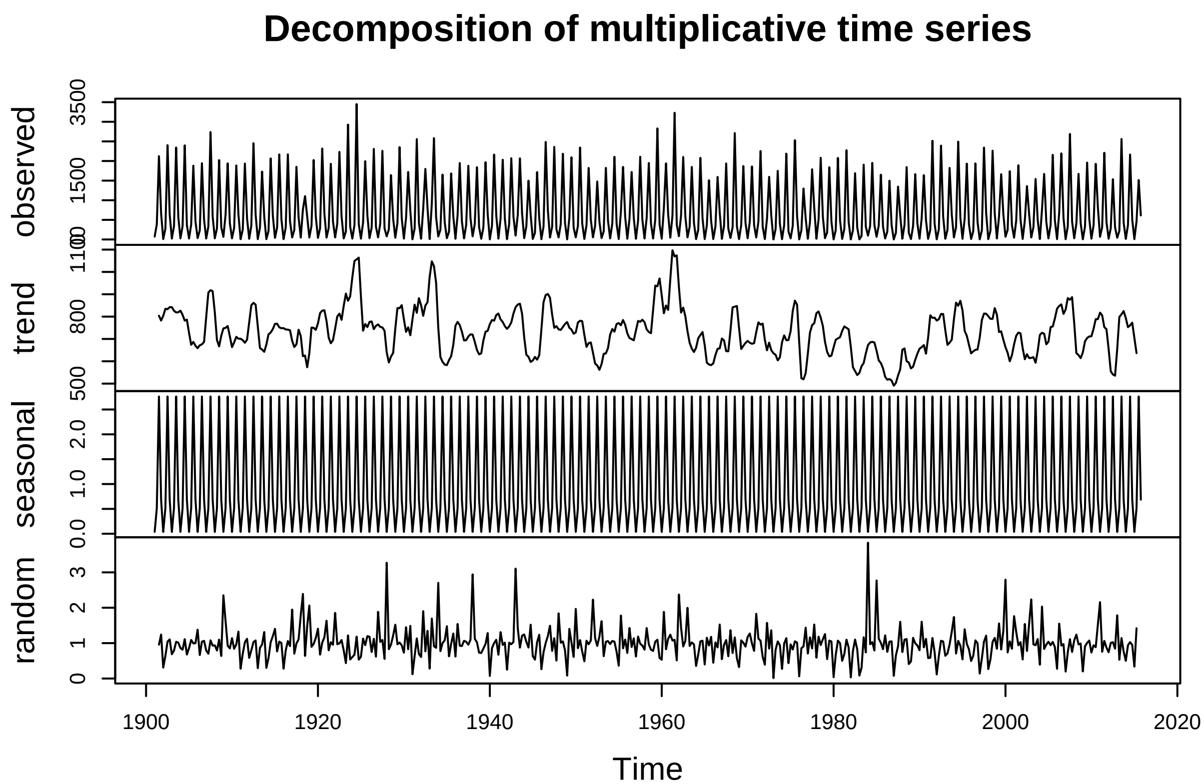
Additive model

```
fit_add <- decompose(rainfall_ts, type="additive")
plot(fit_add)
```



Multiplicative model

```
fit_mult <- decompose(rainfall_ts, type="multiplicative")
plot(fit_mult)
```



Comments: Additive makes more sense here since seasonal amplitude does not consistently vary with time.

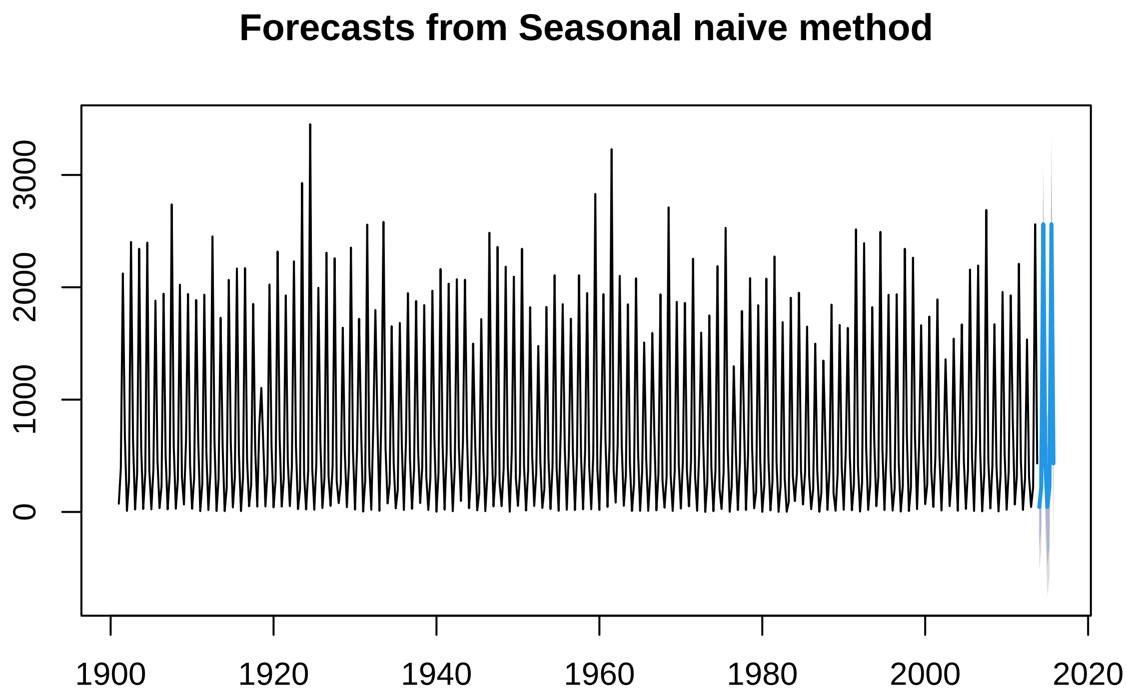
6.2) Seasonal Naive

```
fit_snaive <- snaive(train_ts, h = 8)

print(fit_snaive$model)
```

```
## Call: snaive(y = train_ts, h = 8)
##
## Residual sd: 288.3743
```

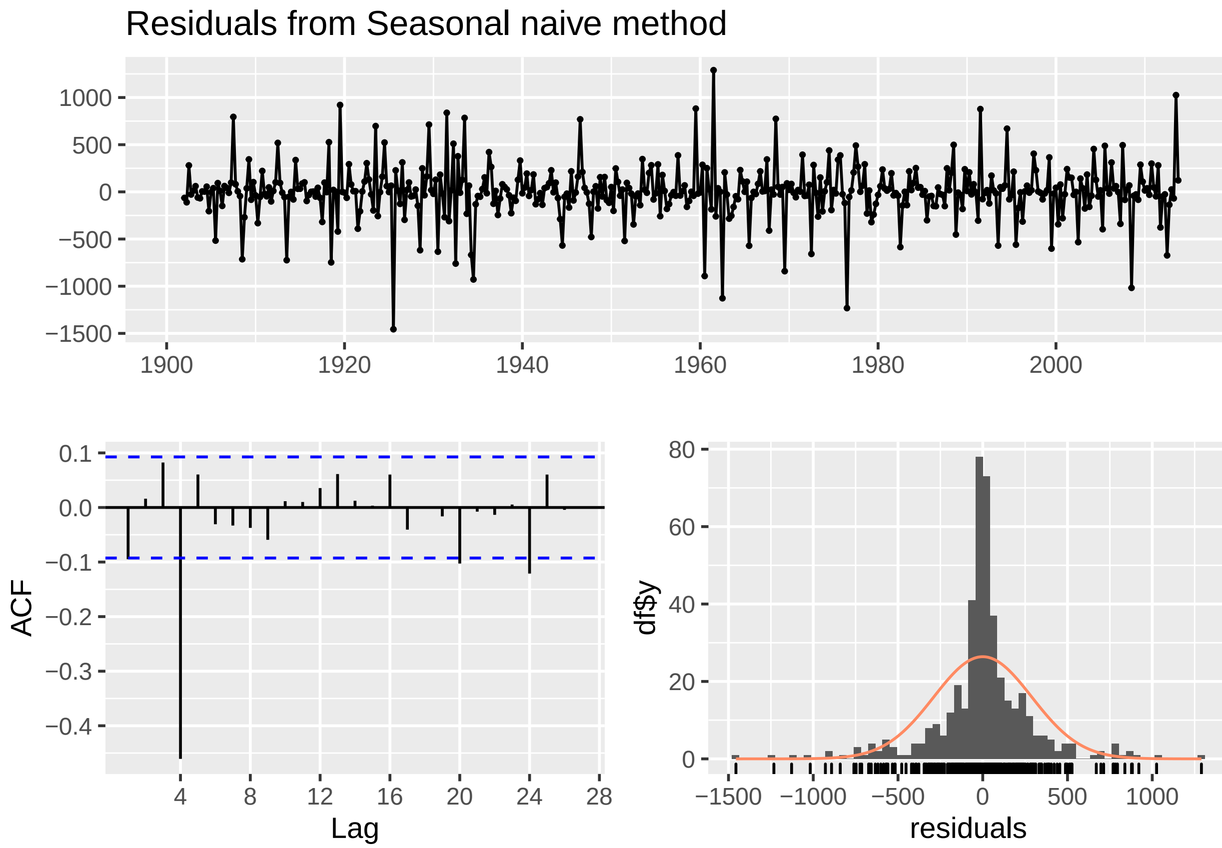
```
plot(fit_snaive)
```

Forecasts from Seasonal naive method



checking residuals

```
checkresiduals(fit_snaive)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from Seasonal naive method
## Q* = 106.81, df = 8, p-value < 2.2e-16
##
## Model df: 0.   Total lags used: 8
```

Thus, residuals don't resemble white noise.

model performance

```
# performance on test data
mse_snaive <- mean((test_ts - fit_snaive$mean)**2)
mape_snaive <- mean((abs(test_ts - fit_snaive$mean) / test_ts) * 100)

print(paste("The Mean Squared Error for snaive model is", mse_snaive))
```

```
## [1] "The Mean Squared Error for snaive model is 171746.81875"
```

```
print(paste("The Mean Absolute Percentage Error for snaive model is",
            mape_snaive))
```

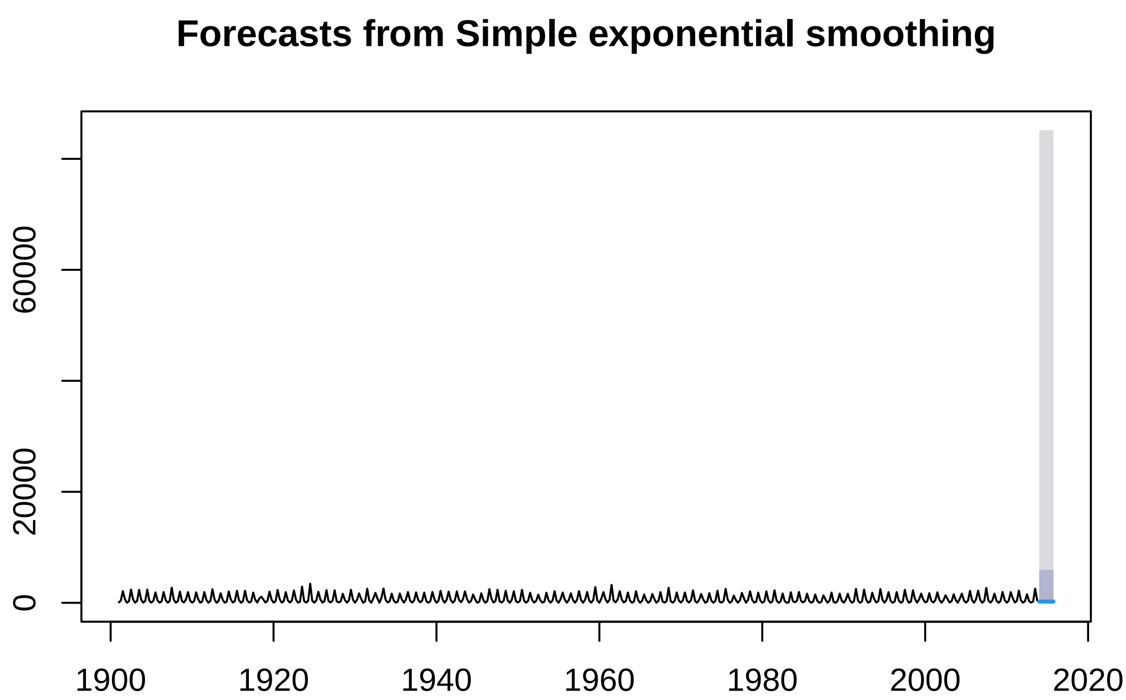
```
## [1] "The Mean Absolute Percentage Error for snaive model is 101.473521324159"
```

6.3) SES

```
fit_ses <- ses(train_ts, h = 8, lambda = -0.14)

print(fit_ses$model)
```

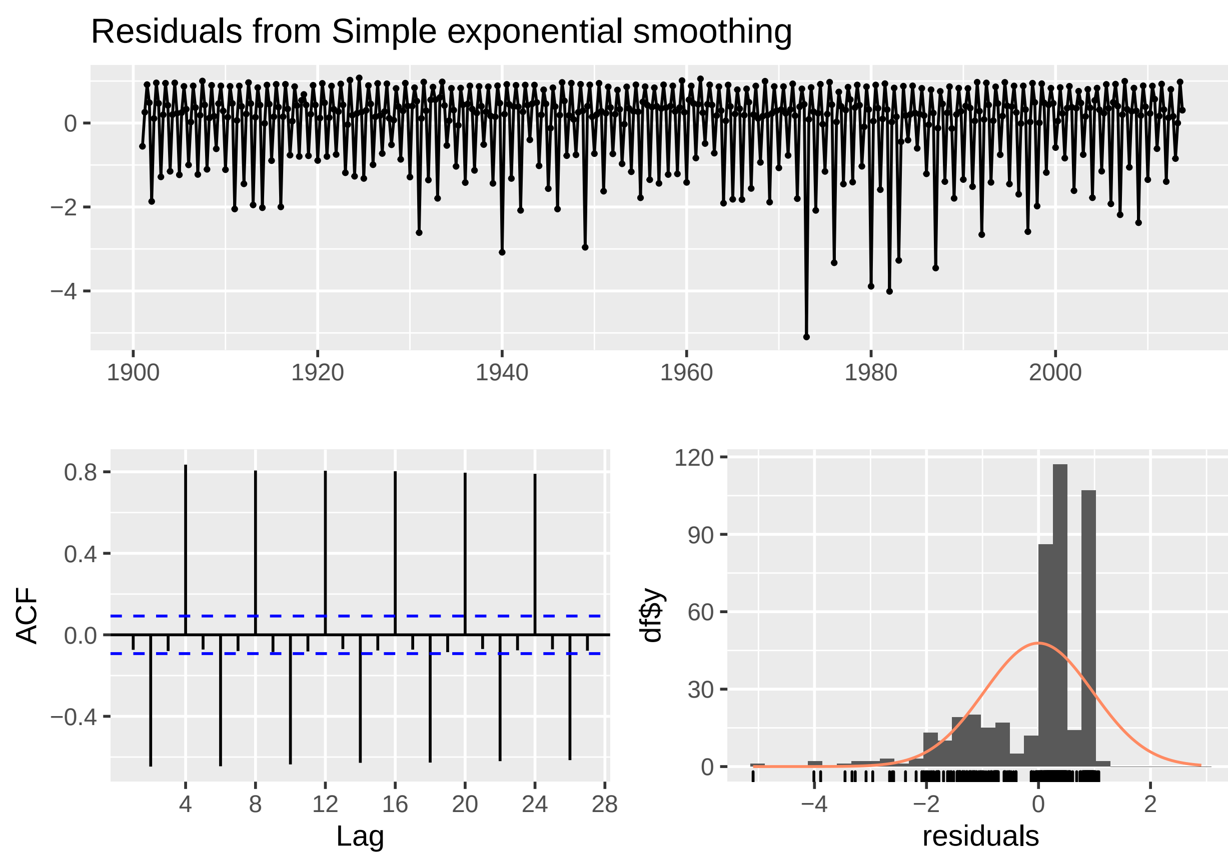
```
## Simple exponential smoothing
##
## Call:
##  ses(y = train_ts, h = 8, lambda = -0.14)
##
##   Box-Cox transformation: lambda= -0.14
##
##   Smoothing parameters:
##     alpha = 1e-04
##
##   Initial states:
##     l = 3.783
##
##   sigma:  0.9705
##
##      AIC     AICc      BIC
## 2740.342 2740.396 2752.683
```

```
plot(fit_ses)
```



checking residuals

```
checkresiduals(fit_ses)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from Simple exponential smoothing
## Q* = 1013, df = 6, p-value < 2.2e-16
##
## Model df: 2.   Total lags used: 8
```

Thus, the residuals don't resemble white noise due to significant ACF lags and this indicates poor model performance.

model performance

```
# performance on training data
aicc_ses <- fit_ses$model$aicc
print(paste("The AICc value for SES model is", aicc_ses))
```

```
## [1] "The AICc value for SES model is 2740.39563677361"
```

```r
# performance on test data
mse_ses <- mean((test_ts - fit_ses$mean)**2)
mape_ses <- mean((abs(test_ts - fit_ses$mean) / test_ts) * 100)

print(paste("The Mean Squared Error for SES model is", mse_ses))
```

```
## [1] "The Mean Squared Error for SES model is 733731.153578899"
```

```r
print(paste("The Mean Absolute Percentage Error for SES model is", mape_ses))
```

```
## [1] "The Mean Absolute Percentage Error for SES model is 514.086910145708"
```

Final comments: 1) An in-between when considering extremes of naivee and average methods 2) Might not be very useful since seasonality is involved 3) From the results, it can be seen that the forecasts are constant in the form of a straight line, thus indicating poor performance

6.4) Holt-Winters Seasonal method

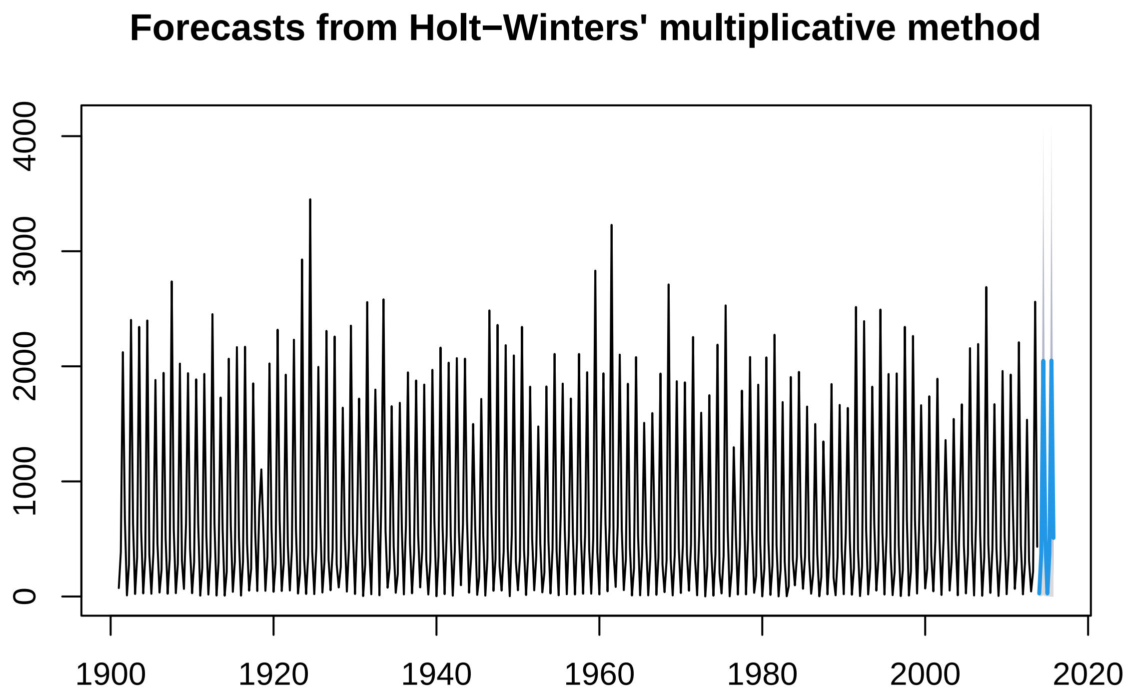
Comments: Can be used for both stationary and non-stationary data; Chose this over Holt's linear method due to presence of seasonality but no trend.

1) Multiplicative seasonal

```r
fit_hw_mult <- hw(train_ts, h = 8, seasonal = "multiplicative")
print(fit_hw_mult$model)
```

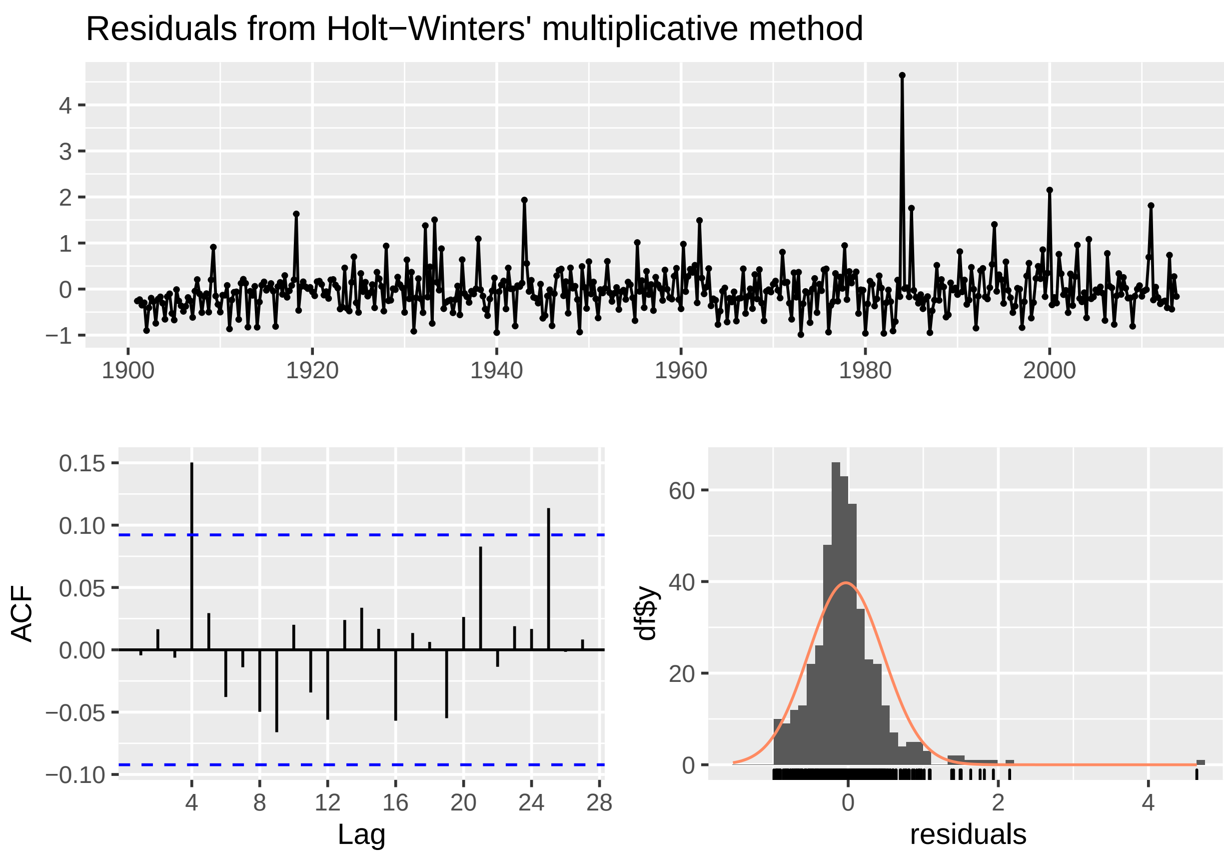
```
## Holt-Winters' multiplicative method
##
## Call:
##  hw(y = train_ts, h = 8, seasonal = "multiplicative")
##
##   Smoothing parameters:
##     alpha = 0.0382
##     beta  = 1e-04
##     gamma = 0.0463
##
##   Initial states:
##     l = 1222.471
##     b = 1.6414
##     s = 0.7967 2.7098 0.4125 0.081
##
##   sigma:  0.5063
##
##      AIC     AICc      BIC
## 7416.041 7416.448 7453.064
```

```r
plot(fit_hw_mult)
```



checking residuals

```
checkresiduals(fit_hw_mult)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from Holt-Winters' multiplicative method
## Q* = 15.539, df = 3, p-value = 0.00141
##
## Model df: 8.   Total lags used: 11
```

Thus, the residuals don't resemble white noise.

model performance

```
# performance on training data
aicc_hw_mult <- fit_hw_mult$model$aicc
print(paste("The AICc value for HW multiplicative model is", aicc_hw_mult))
```

```
## [1] "The AICc value for HW multiplicative model is 7416.44774596371"
```

```
# performance on test data
mse_hw_mult <- mean((test_ts - fit_hw_mult$mean)**2)
mape_hw_mult <- mean((abs(test_ts - fit_hw_mult$mean) / test_ts) * 100)

print(paste("The Mean Squared Error for HW multiplicative model is", mse_hw_mult))
```

```
## [1] "The Mean Squared Error for HW multiplicative model is 39939.6437766148"
```

```r
print(paste("The Mean Absolute Percentage Error for HW multiplicative model is", mape_hw_mult))
```

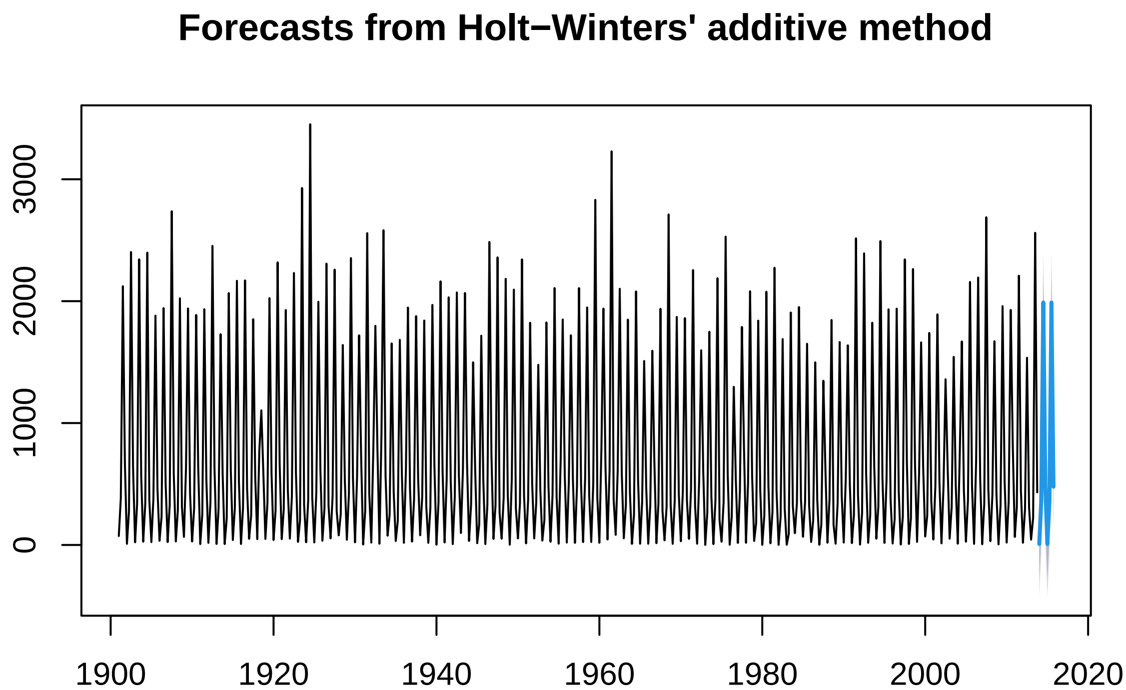
```
## [1] "The Mean Absolute Percentage Error for HW multiplicative model is 44.7150669846437"
```

2) Additive seasonal

```r
fit_hw_add <- hw(train_ts, h = 8, seasonal = "additive")
print(fit_hw_add$model)
```

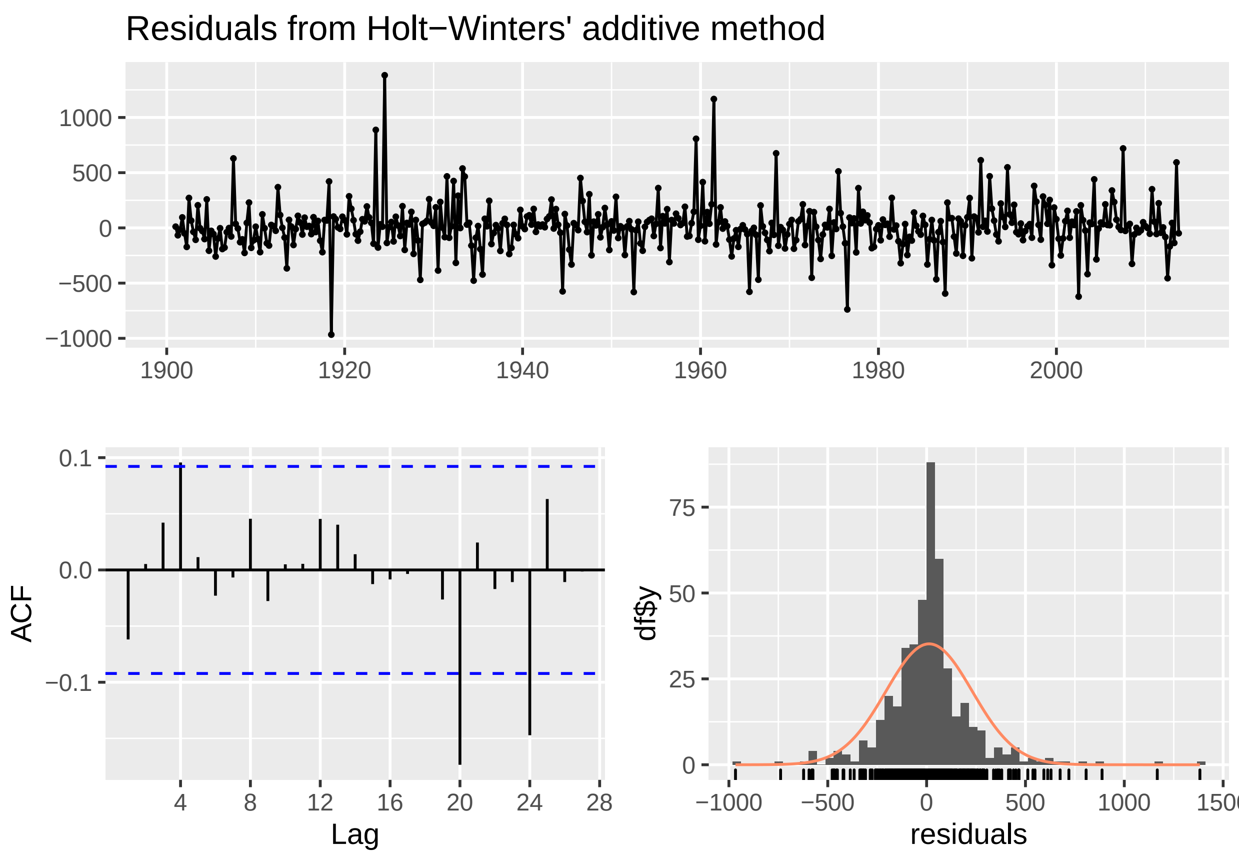
```
## Holt-Winters' additive method
##
## Call:
##  hw(y = train_ts, h = 8, seasonal = "additive")
##
##   Smoothing parameters:
##     alpha = 0.0206
##     beta  = 1e-04
##     gamma = 0.0177
##
##   Initial states:
##     l = 808.927
##     b = -0.8072
##     s = -233.2125 1333.557 -354.4059 -745.9389
##
##   sigma:  220.9279
##
##      AIC     AICc      BIC
## 7652.957 7653.364 7689.980
```

```r
plot(fit_hw_add)
```



checking residuals

```
checkresiduals(fit_hw_add)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from Holt-Winters' additive method
## Q* = 8.4215, df = 3, p-value = 0.03806
##
## Model df: 8.   Total lags used: 11
```

Thus, the residuals don't resemble white noise.

model performance

```
# performance on training data
aicc_hw_add <- fit_hw_add$model$aicc
print(paste("The AICc value for HW additive model is", aicc_hw_add))
```

```
## [1] "The AICc value for HW additive model is 7653.36421139307"
```

```
# performance on test data
mse_hw_add <- mean((test_ts - fit_hw_add$mean)**2)
mape_hw_add <- mean((abs(test_ts - fit_hw_add$mean) / test_ts) * 100)

print(paste("The Mean Squared Error for HW additive model is", mse_hw_add))
```

```
## [1] "The Mean Squared Error for HW additive model is 35658.6215979988"
```

```r
print(paste("The Mean Absolute Percentage Error for HW additive model is", mape_hw_add))
```

```
## [1] "The Mean Absolute Percentage Error for HW additive model is 17.4296122936171"
```

Final comments: From comparing both the multiplicative and additive models, it looks like the former does better on the training data but worse when it comes to the test data.

6.5) State space models

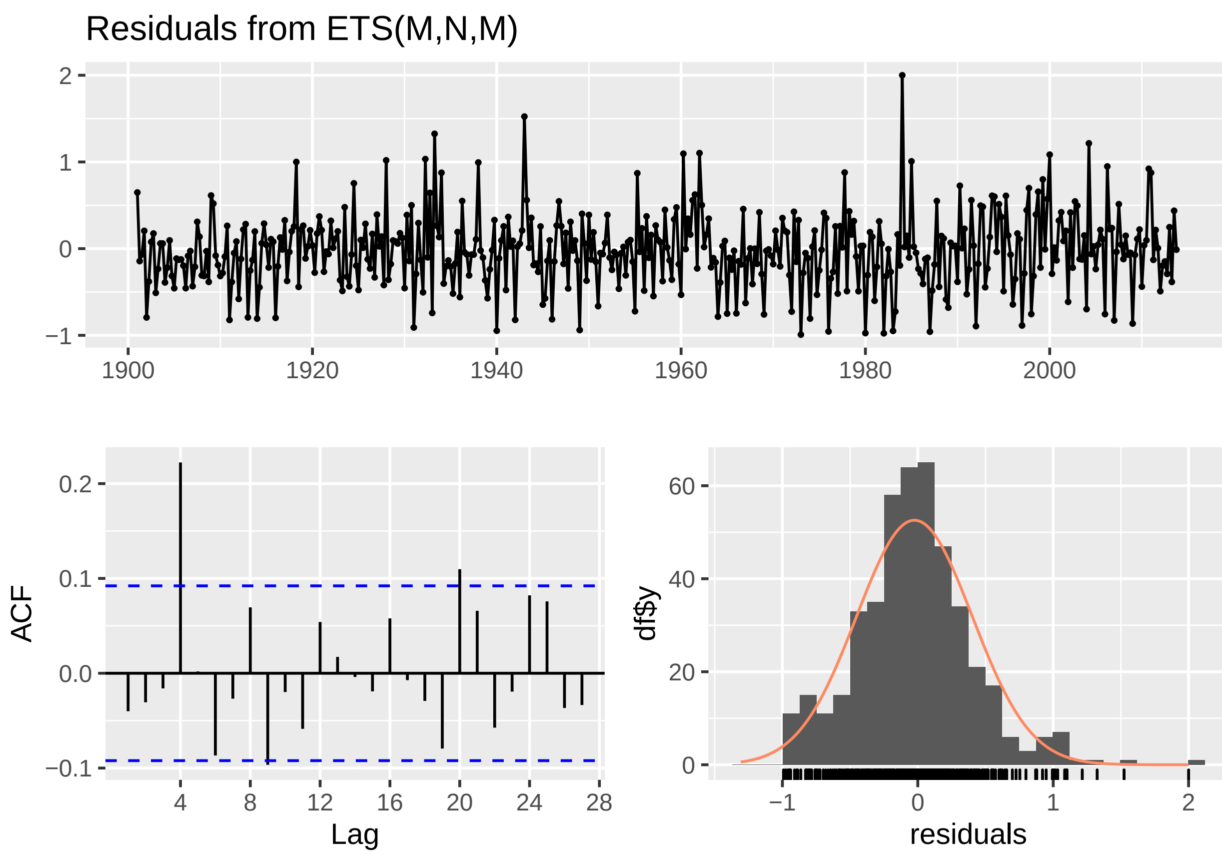
fitting the model

```r
state_space_auto_fit <- ets(train_ts, model="ZZZ")
summary(state_space_auto_fit)
```

```
## ETS(M,N,M)
##
## Call:
##  ets(y = train_ts, model = "ZZZ")
##
##   Smoothing parameters:
##     alpha = 0.0188
##     gamma = 1e-04
##
##   Initial states:
##     l = 823.0082
##     s = 0.6659 2.7385 0.5414 0.0541
##
##   sigma:  0.4307
##
##      AIC     AICc      BIC
## 7259.597 7259.849 7288.392
##
## Training set error measures:
##                   ME     RMSE      MAE       MPE     MAPE      MASE         ACF1
## Training set 38.0828 229.6326 137.7357 -98.72692 118.0903 0.7827869 -0.05013145
```

checking residuals for auto fit

```r
checkresiduals(state_space_auto_fit)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ETS(M,N,M)
## Q* = 34.249, df = 3, p-value = 1.756e-07
##
## Model df: 6.   Total lags used: 9
```

manual fit based on visual observation of raw data plot

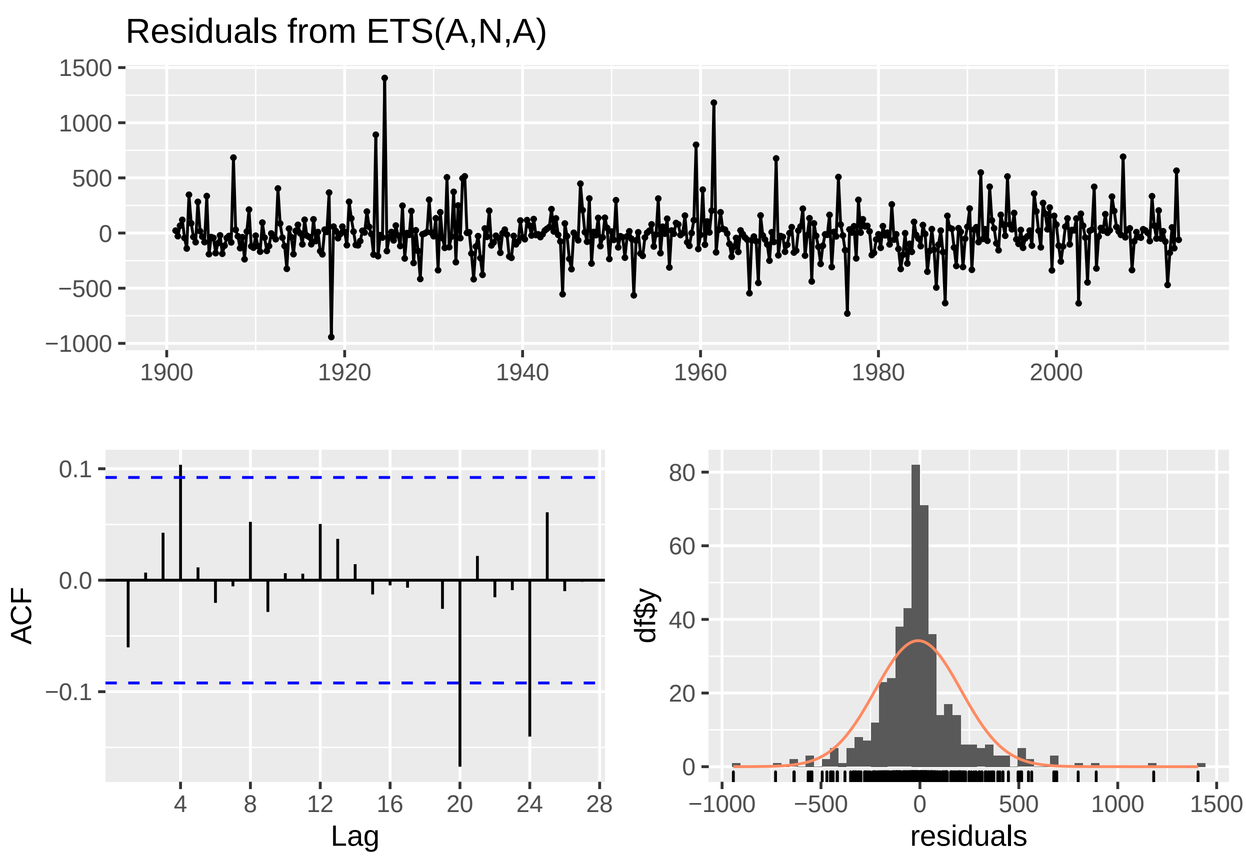
```
state_space_manual_fit <- ets(train_ts, model="ANA")
summary(state_space_manual_fit)
```

```
## ETS(A,N,A)
##
## Call:
##  ets(y = train_ts, model = "ANA")
##
##   Smoothing parameters:
##     alpha = 0.0131
##     gamma = 1e-04
##
##   Initial states:
```

```
##     l = 766.9723
##     s = -221.1412 1289.31 -352.6221 -715.5465
##
##   sigma:  218.5169
##
##      AIC     AICc      BIC
## 7641.069 7641.321 7669.864
##
## Training set error measures:
##                     ME     RMSE      MAE       MPE     MAPE      MASE
## Training set -9.004383 217.0617 136.9752 -47.25703 97.71366 0.7784644
##                     ACF1
## Training set -0.06022874
```

checking residuals

```
checkresiduals(state_space_manual_fit)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ETS(A,N,A)
## Q* = 9.322, df = 3, p-value = 0.0253
##
```

```
## Model df: 6.   Total lags used: 9
```

Cooemnts: Though AIC for MNM is better, the residual chart looks better for ANA. So, decided to go ahead with ANA for now.

model performance on test data

```
ets_mse <- mean((test_ts - forecast(state_space_manual_fit, h=8, level=c(80, 95))$mean)**2)
ets_mape <- mean((abs(test_ts - forecast(state_space_manual_fit, h=8, level=c(80, 95))$mean) / test_ts)

print(paste("The Mean Squared Error for ETS model is", ets_mse))
```

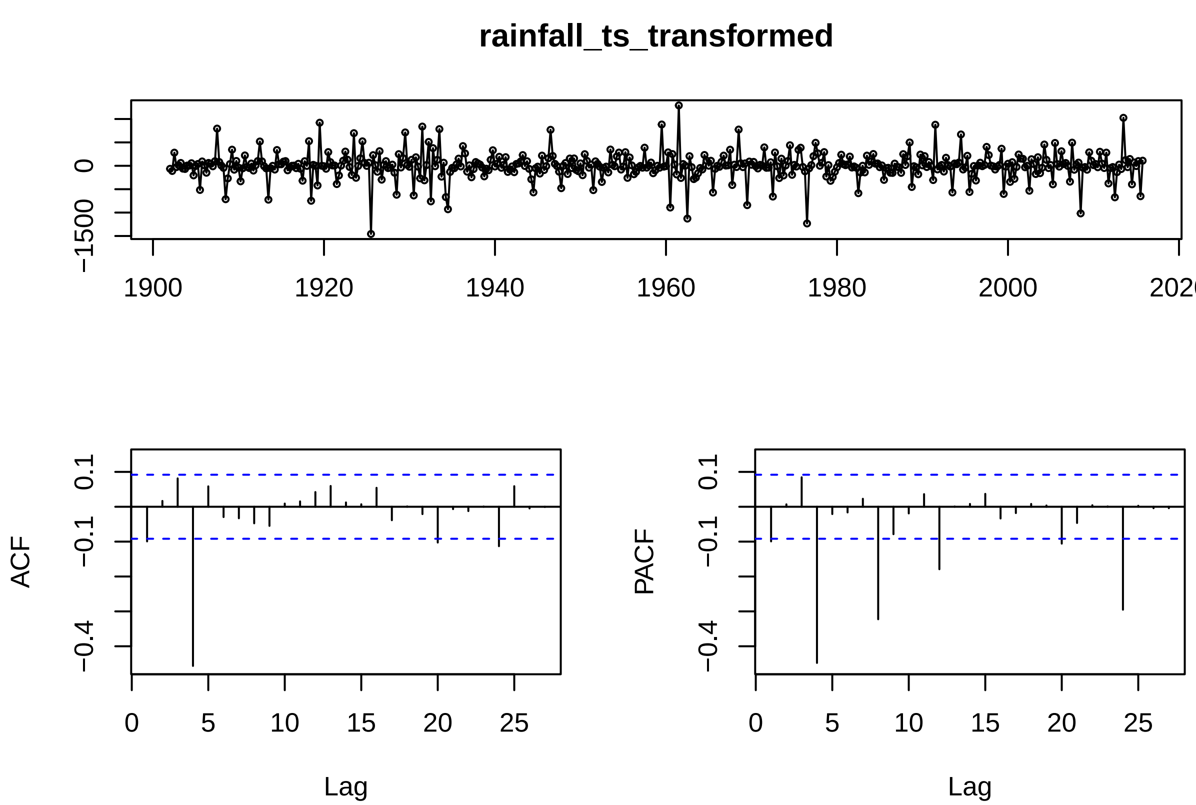
```
## [1] "The Mean Squared Error for ETS model is 36275.0078025827"
```

```
print(paste("The Mean Absolute Percentage Error for ETS model is", ets_mape))
```

```
## [1] "The Mean Absolute Percentage Error for ETS model is 40.0972602668366"
```

6.6) ARIMA model

```
tsdisplay(rainfall_ts_transformed)
```



When considering the seasonality component, the PACF decays exponentially with most significant lags at the seasonal lags of 4, 8, etc while the ACF drops off abruptly post the seasonal lag of 4. This points to the fact that the stationary process is most probably an ARIMA(0,0,0)(0,1,1).

Since the data clearly exhibits seasonality, any rigorous modeling pertaining to non-seasonal ARIMA models were avoided.

1) Non-seasonal ARIMA

using auto.arima

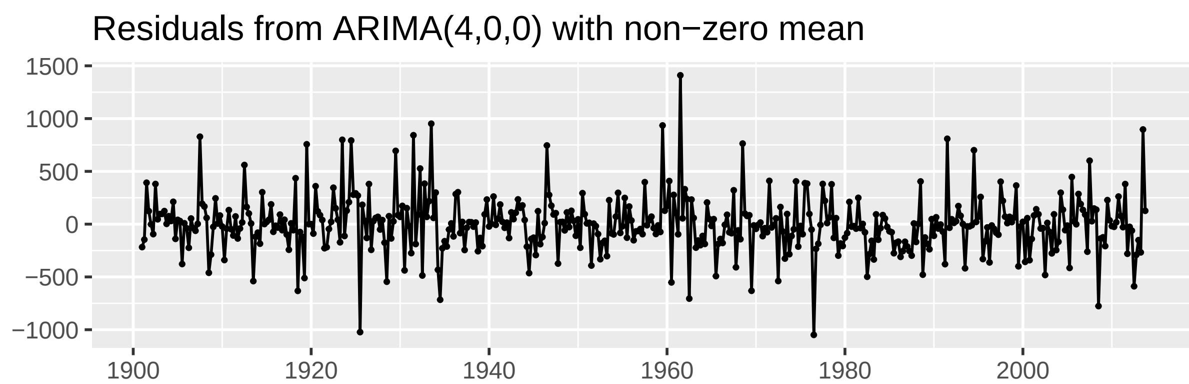
```r
ns_fit <- auto.arima(train_ts, seasonal=FALSE, trace=TRUE, approximation = FALSE)
```

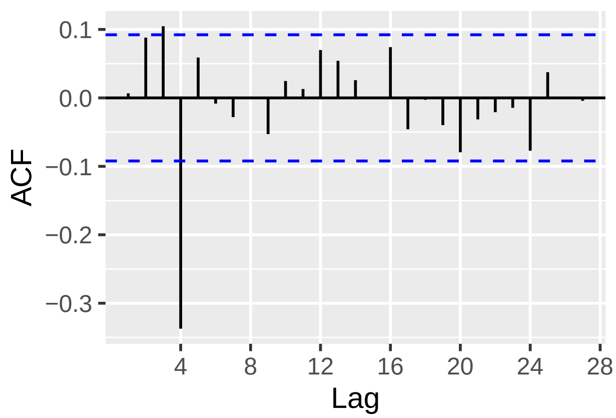
```
##
##  ARIMA(2,0,2)           with non-zero mean : Inf
##  ARIMA(0,0,0)           with non-zero mean : 7326.157
##  ARIMA(1,0,0)           with non-zero mean : 7319.106
##  ARIMA(0,0,1)           with non-zero mean : 7189.51
##  ARIMA(0,0,0)           with zero mean     : 7600.557
##  ARIMA(1,0,1)           with non-zero mean : 7185.841
##  ARIMA(2,0,1)           with non-zero mean : 6936.262
##  ARIMA(2,0,0)           with non-zero mean : 7037.528
##  ARIMA(3,0,1)           with non-zero mean : Inf
##  ARIMA(1,0,2)           with non-zero mean : 7152.784
##  ARIMA(3,0,0)           with non-zero mean : 6700.543
##  ARIMA(4,0,0)           with non-zero mean : 6357.656
##  ARIMA(5,0,0)           with non-zero mean : 6359.691
##  ARIMA(4,0,1)           with non-zero mean : 6359.699
##  ARIMA(5,0,1)           with non-zero mean : 6359.127
##  ARIMA(4,0,0)           with zero mean     : Inf
##
##  Best model: ARIMA(4,0,0)           with non-zero mean
```

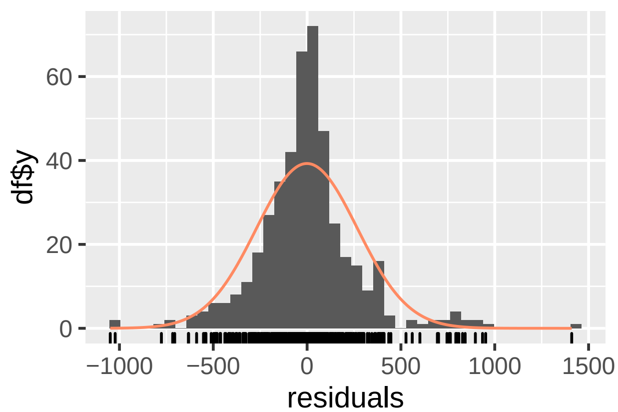
```r
print(ns_fit)
```

```
## Series: train_ts
## ARIMA(4,0,0) with non-zero mean
##
## Coefficients:
##           ar1      ar2      ar3     ar4      mean
##       -0.2011  -0.2290  -0.1932  0.7338  732.5082
## s.e.   0.0318   0.0318   0.0321  0.0318   14.1415
##
## sigma^2 = 72909:  log likelihood = -3172.73
## AIC=6357.47   AICc=6357.66   BIC=6382.15
```

```r
checkresiduals(ns_fit)
```








```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(4,0,0) with non-zero mean
## Q* = 62.627, df = 4, p-value = 8.13e-13
##
## Model df: 4.   Total lags used: 8
```

This gives ARIMA(4,0,0) as the best model but residuals don't resemble white noise.

2) SARIMA modeling

using auto.arima

```
s_fit <- auto.arima(train_ts, seasonal=TRUE, trace=TRUE, approximation=FALSE,
                    D=1)
```

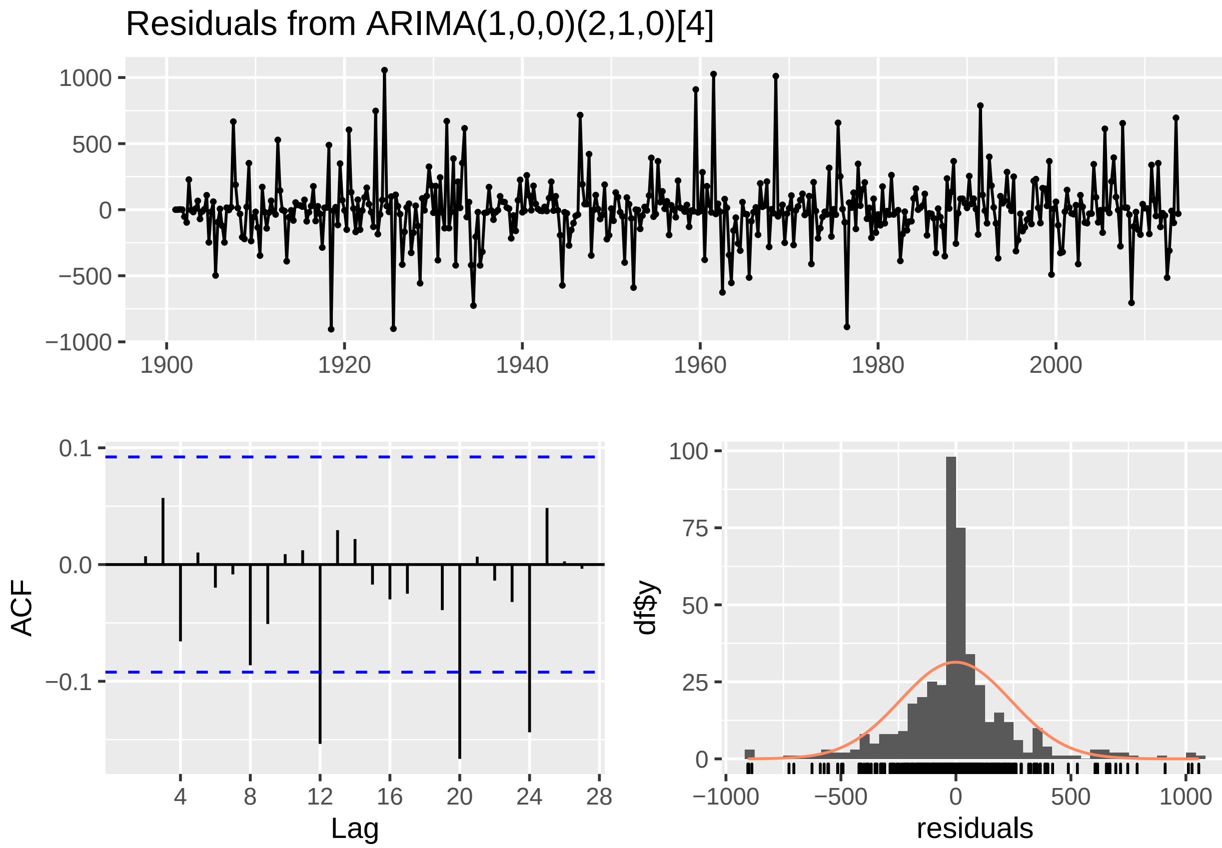
```
##
##  ARIMA(2,0,2)(1,1,1)[4] with drift         : Inf
##  ARIMA(0,0,0)(0,1,0)[4] with drift         : 6350.572
##  ARIMA(1,0,0)(1,1,0)[4] with drift         : 6243.131
##  ARIMA(0,0,1)(0,1,1)[4] with drift         : Inf
##  ARIMA(0,0,0)(0,1,0)[4]                    : 6348.554
```

```
##  ARIMA(1,0,0)(0,1,0)[4] with drift         : 6348.562
##  ARIMA(1,0,0)(2,1,0)[4] with drift         : 6196.151
##  ARIMA(1,0,0)(2,1,1)[4] with drift         : Inf
##  ARIMA(1,0,0)(1,1,1)[4] with drift         : Inf
##  ARIMA(0,0,0)(2,1,0)[4] with drift         : 6196.74
##  ARIMA(2,0,0)(2,1,0)[4] with drift         : 6198.203
##  ARIMA(1,0,1)(2,1,0)[4] with drift         : 6198.205
##  ARIMA(0,0,1)(2,1,0)[4] with drift         : 6196.202
##  ARIMA(2,0,1)(2,1,0)[4] with drift         : Inf
##  ARIMA(1,0,0)(2,1,0)[4]                    : 6194.134
##  ARIMA(1,0,0)(1,1,0)[4]                    : 6241.106
##  ARIMA(1,0,0)(2,1,1)[4]                    : Inf
##  ARIMA(1,0,0)(1,1,1)[4]                    : Inf
##  ARIMA(0,0,0)(2,1,0)[4]                    : 6194.728
##  ARIMA(2,0,0)(2,1,0)[4]                    : 6196.176
##  ARIMA(1,0,1)(2,1,0)[4]                    : 6196.178
##  ARIMA(0,0,1)(2,1,0)[4]                    : 6194.185
##  ARIMA(2,0,1)(2,1,0)[4]                    : Inf
##
##  Best model: ARIMA(1,0,0)(2,1,0)[4]
```

```
print(s_fit)
```

```
## Series: train_ts
## ARIMA(1,0,0)(2,1,0)[4]
##
## Coefficients:
##           ar1     sar1     sar2
##       -0.0768  -0.6147  -0.3247
## s.e.   0.0473   0.0451   0.0450
##
## sigma^2 = 58268:  log likelihood = -3093.02
## AIC=6194.04   AICc=6194.13   BIC=6210.46
```

```
checkresiduals(s_fit)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,0,0)(2,1,0)[4]
## Q* = 7.1977, df = 5, p-value = 0.2063
##
## Model df: 3.   Total lags used: 8
```

The best model is ARIMA(1,0,0)(2,1,0)[4] and from Ljung-Box test, the residuals seem to not be autocorrelated.

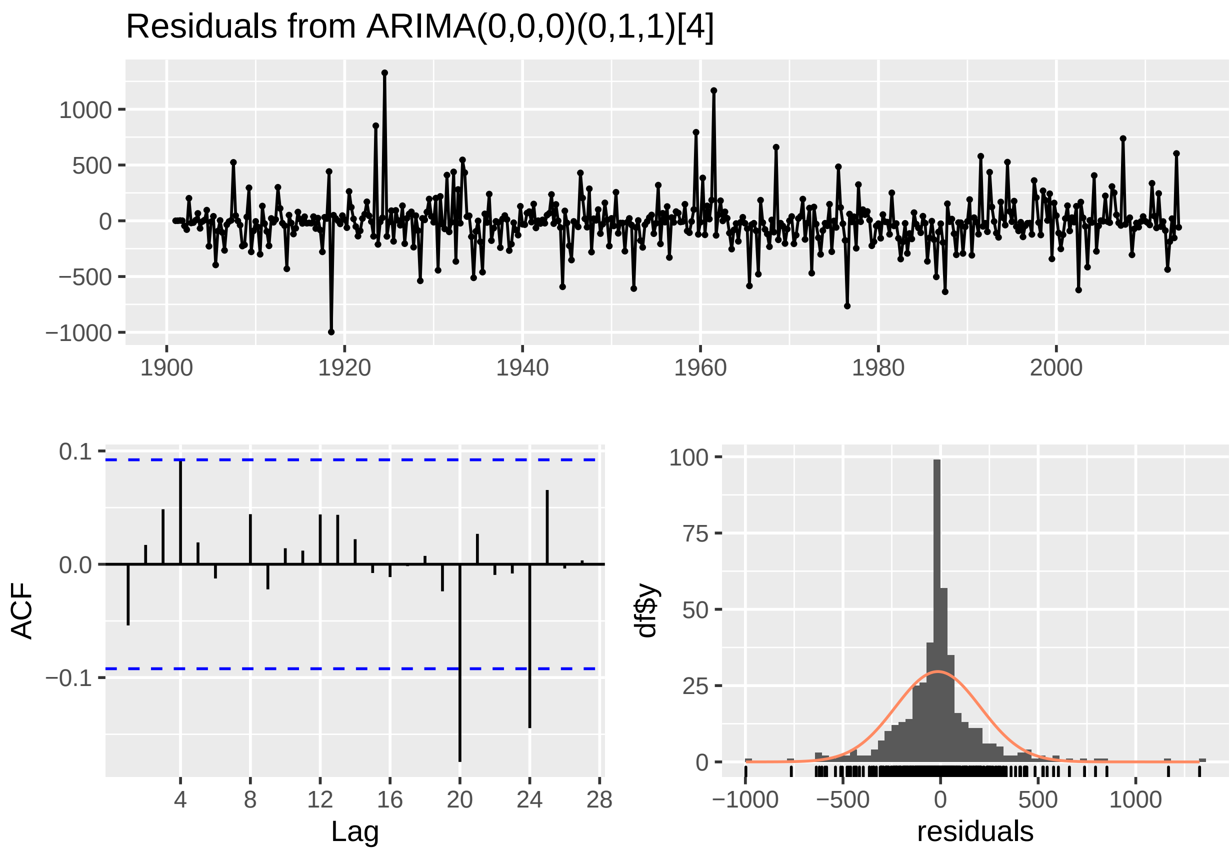
using conclusion from ACF and PACF charts of stationary data

```
s_fit_1 <- Arima(train_ts, order=c(0,0,0), seasonal=c(0, 1, 1))
print(s_fit_1)
```

```
## Series: train_ts
## ARIMA(0,0,0)(0,1,1)[4]
##
## Coefficients:
##          sma1
##       -0.9652
## s.e.   0.0156
```

```
##
## sigma^2 = 48102:  log likelihood = -3055.51
## AIC=6115.02   AICc=6115.05   BIC=6123.23
```

```
checkresiduals(s_fit_1)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,0,0)(0,1,1)[4]
## Q* = 7.5586, df = 7, p-value = 0.3731
##
## Model df: 1.   Total lags used: 8
```

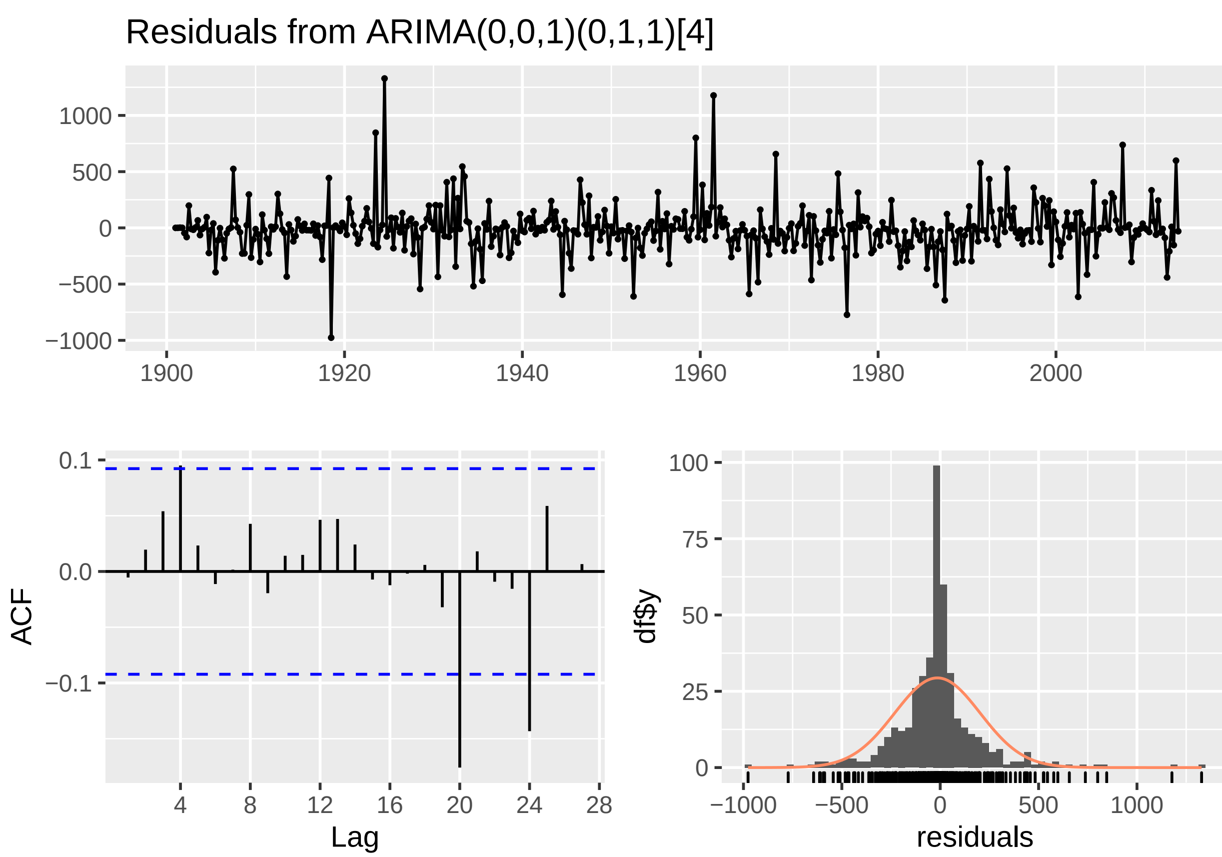
Thus, it seems like this model performs better than the auto.arima one when considering both AICc and BIC. Even the residuals are uncorrelated as per the Ljung-Box test.

experimenting with other P,D,Q combinations

```
# modifying p, q, P, Q values

test_fit_1 <- Arima(train_ts, order=c(0,0,1), seasonal=c(0, 1, 1))
print(test_fit_1)
```

```
## Series: train_ts
## ARIMA(0,0,1)(0,1,1)[4]
##
## Coefficients:
##           ma1     sma1
##       -0.0478  -0.9640
## s.e.   0.0459   0.0158
##
## sigma^2 = 48108:  log likelihood = -3054.97
## AIC=6115.95   AICc=6116   BIC=6128.26
```

```
checkresiduals(test_fit_1)
```



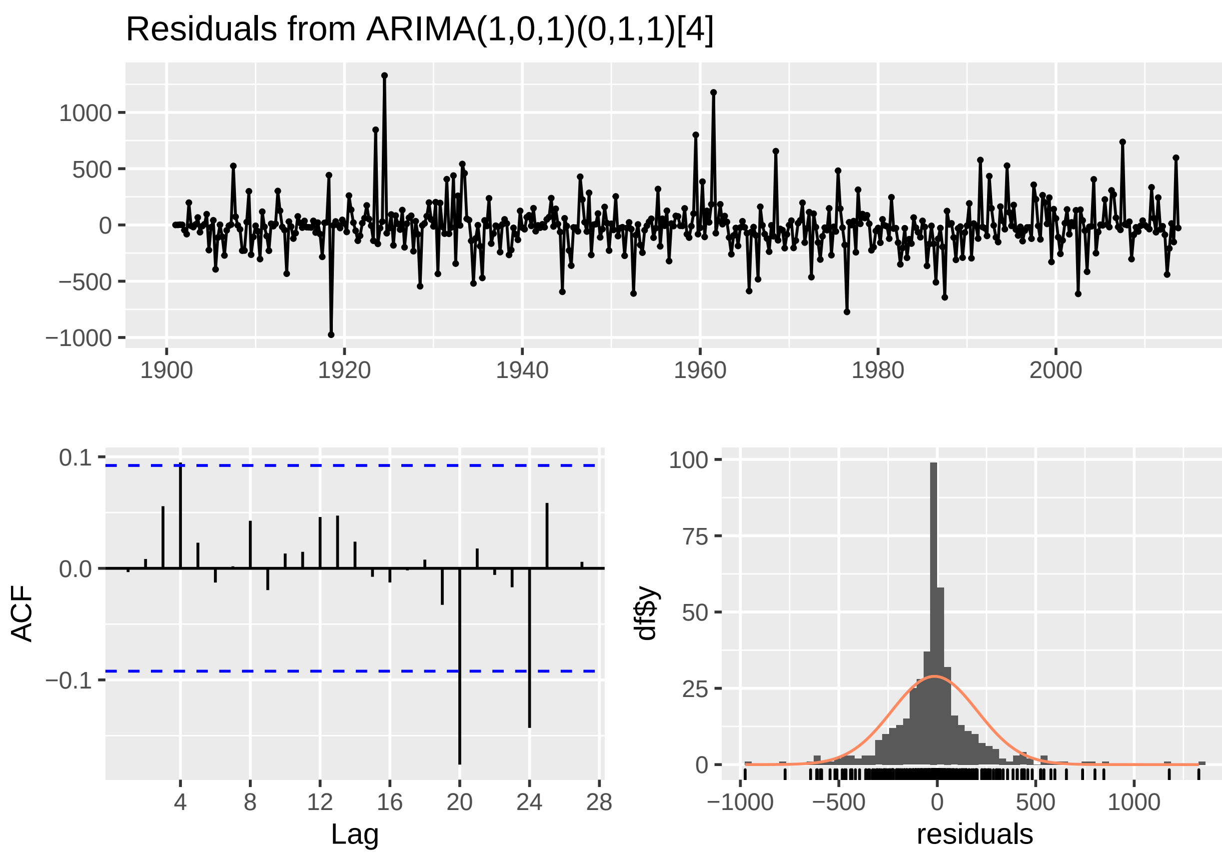

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,0,1)(0,1,1)[4]
## Q* = 6.8081, df = 6, p-value = 0.339
##
## Model df: 2.   Total lags used: 8
```

```
test_fit_2 <- Arima(train_ts, order=c(1,0,1), seasonal=c(0, 1, 1))
print(test_fit_2)
```

```
## Series: train_ts
```

```
## ARIMA(1,0,1)(0,1,1)[4]
##
## Coefficients:
##           ar1     ma1     sma1
##       -0.2115  0.1613  -0.9639
## s.e.   0.7581  0.7647   0.0158
##
## sigma^2 = 48206:  log likelihood = -3054.92
## AIC=6117.84   AICc=6117.93   BIC=6134.26
```

```
checkresiduals(test_fit_2)
```



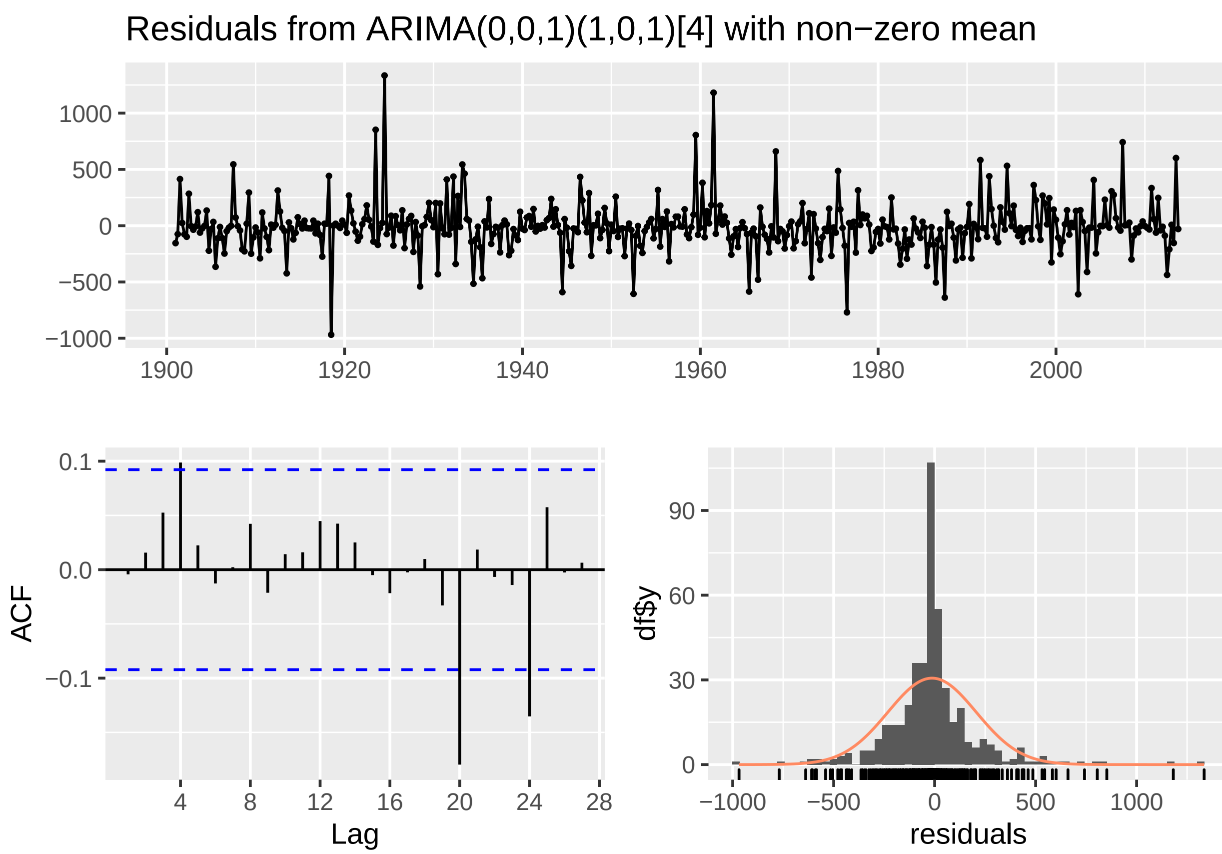

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,0,1)(0,1,1)[4]
## Q* = 6.7362, df = 5, p-value = 0.241
##
## Model df: 3.   Total lags used: 8
```

```
test_fit_3 <- Arima(train_ts, order=c(0,0,1), seasonal=c(1, 0, 1))
print(test_fit_3)
```

```
## Series: train_ts
```

```
## ARIMA(0,0,1)(1,0,1)[4] with non-zero mean
##
## Coefficients:
##           ma1    sar1     sma1      mean
##       -0.0495  0.9999  -0.9626  627.8694
## s.e.   0.0460  0.0001   0.0159  365.5652
##
## sigma^2 = 48362:  log likelihood = -3087.27
## AIC=6184.54   AICc=6184.67   BIC=6205.1
```

```
checkresiduals(test_fit_3)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,0,1)(1,0,1)[4] with non-zero mean
## Q* = 6.9857, df = 5, p-value = 0.2217
##
## Model df: 3.   Total lags used: 8
```

best model performance on test data

```r
sarima_mse <- mean((test_ts - forecast(s_fit_1, h=8, level=c(80, 95))$mean)**2)
sarima_mape <- mean((abs(test_ts - forecast(s_fit_1, h=8, level=c(80, 95))$mean) / test_ts) * 100)

print(paste("The Mean Squared Error for best SARIMA model is", sarima_mse))
```

```
## [1] "The Mean Squared Error for best SARIMA model is 34411.9372875343"
```

```r
print(paste("The Mean Absolute Percentage Error for best SARIMA model is", sarima_mape))
```

```
## [1] "The Mean Absolute Percentage Error for best SARIMA model is 42.8660270702021"
```

6.7) ARFIMA

fitting the model

```r
arfima_fit <- arfima(train_ts)
```

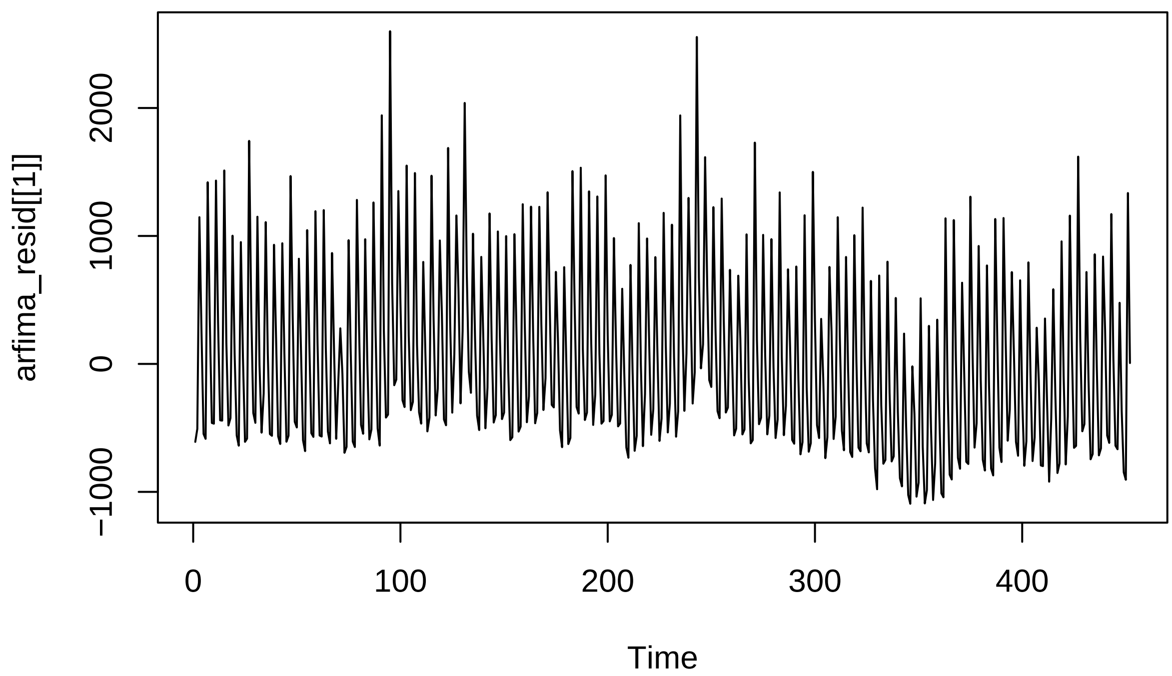
```
## Note: only one starting point.  Only one mode can be found -- this is now the default behavior.
## Beginning the fits with 1 starting values.
```

```r
summary(arfima_fit)
```

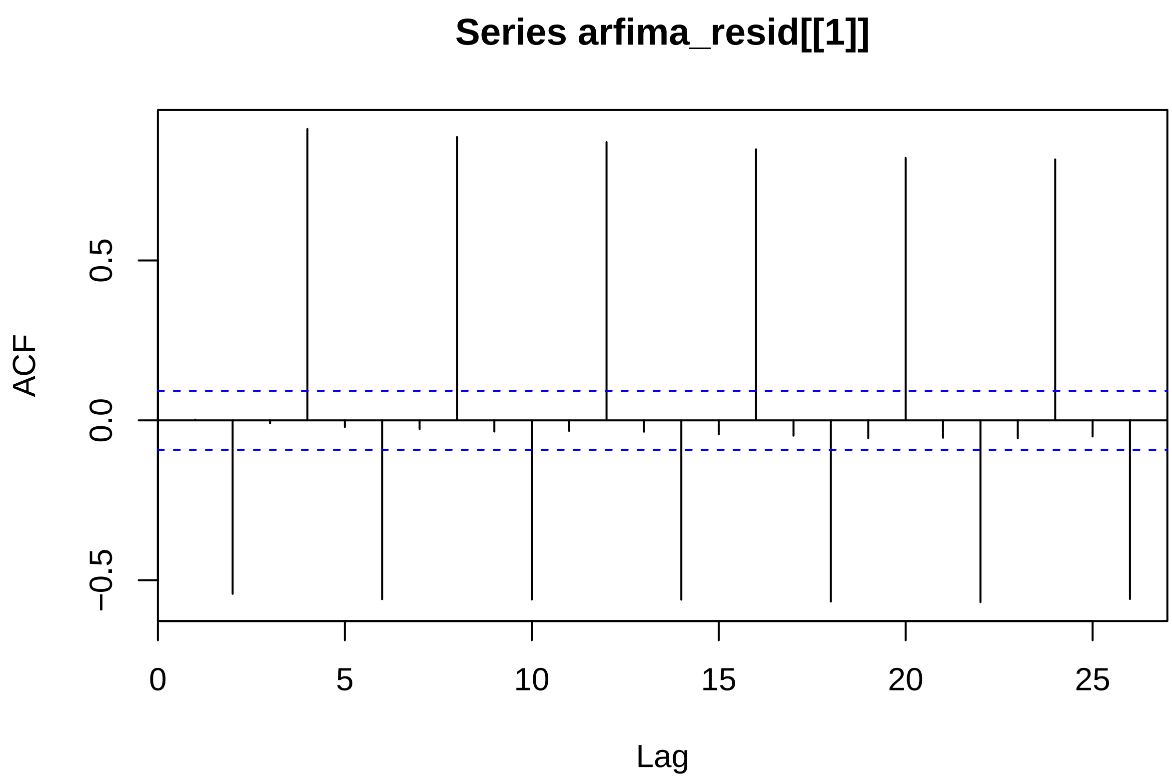
```
##
## Call:
##
## arfima(z = train_ts)
##
##
## Mode 1 Coefficients:
##                Estimate  Std. Error Th. Std. Err.  z-value   Pr(>|z|)
## d.f          -0.3802807   0.0288062     0.0366742 -13.2013 < 2.22e-16 ***
## Fitted mean 731.6574130   4.5923850            NA 159.3197 < 2.22e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## sigma^2 estimated as 525423; Log-likelihood = -2976.35; AIC = 5958.71; BIC = 5971.05
##
## Numerical Correlations of Coefficients:
##             d.f  Fitted mean
## d.f         1.00 0.09
## Fitted mean 0.09 1.00
##
## Theoretical Correlations of Coefficients:
##     d.f
## d.f 1.00
##
## Expected Fisher Information Matrix of Coefficients:
##     d.f
## d.f 1.64
```

inspecting the residuals

```
arfima_resid <- resid(arfima_fit)
plot.ts(arfima_resid[[1]])
```




```
acf(arfima_resid[[1]])
```



ACF of residuals DO NOT resemble white noise and ARFIMA is probably not the best option here, most probably due to the presence of seasonality.

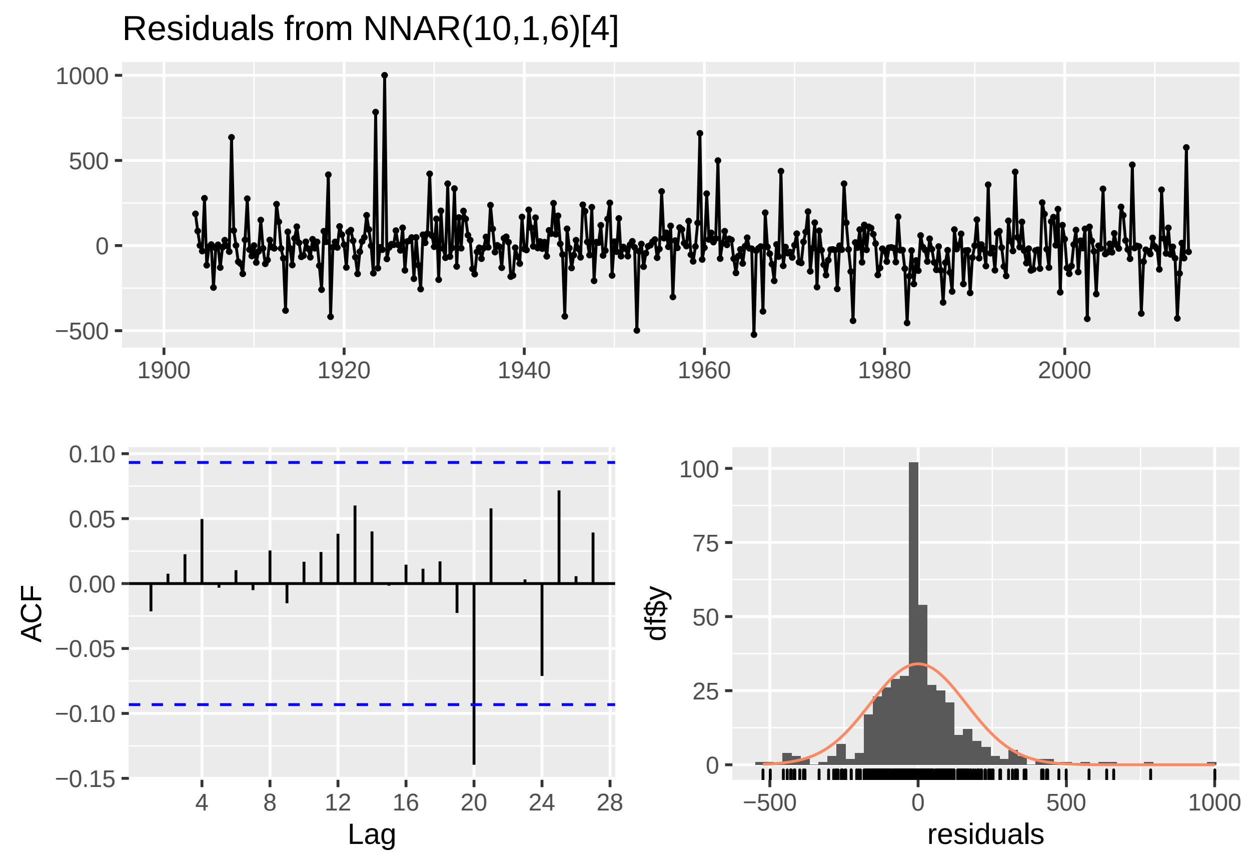
6.8) Neural nets

```
nn_fit <- nnetar(train_ts, p=10, repeats = 30)
print(nn_fit)
```

```
## Series: train_ts
## Model:  NNAR(10,1,6)[4]
## Call:   nnetar(y = train_ts, p = 10, repeats = 30)
##
## Average of 30 networks, each of which is
## a 10-6-1 network with 73 weights
## options were - linear output units
##
## sigma^2 estimated as 25989
```

checking residuals

```
checkresiduals(nn_fit)
```



The residuals are uncorrelated as per Ljung-Box test and resemble white noise for lags < 20.

model performance on test data

```
nn_mse <- mean((test_ts - forecast(nn_fit, h=8, level=c(80, 95))$mean)**2)
nn_mape <- mean((abs(test_ts - forecast(nn_fit, h=8, level=c(80, 95))$mean) / test_ts) * 100)

print(paste("The Mean Squared Error for best NN model is", nn_mse))
```

```
## [1] "The Mean Squared Error for best NN model is 33394.0659905807"
```

```
print(paste("The Mean Absolute Percentage Error for best NN model is", nn_mape))
```

```
## [1] "The Mean Absolute Percentage Error for best NN model is 78.8021988677231"
```

7) Model Evaluation

```
model_eval_df <- data.frame(
 model_type = c("snaive", "HW (additive)", "ETS (ANA)", "SARIMA", "Neural Net"),
 mse = c(mse_snaive, mse_hw_add, ets_mse, sarima_mse, nn_mse),
 mape = c(mape_snaive, mape_hw_add, ets_mape, sarima_mape, nn_mape)
```

```
)

print(model_eval_df)
```

```
##      model_type       mse      mape
## 1        snaive 171746.82 101.47352
## 2 HW (additive)  35658.62  17.42961
## 3     ETS (ANA)  36275.01  40.09726
## 4        SARIMA  34411.94  42.86603
## 5    Neural Net  33394.07  78.80220
```

Conclusions: The SARIMA model performs best on MSE while Holt-Winters' does best on MAPE.