ADS506 Project Australia Rainfall Forecast

Marinela Inguito, Jose Guarneros, Robert Marriott

```
library(dplyr)
library(ggplot2)
library(lubridate)
library(tidyr)
library(tsibble)
library(tseries)
library(feasts)
library(fpp3)
library(corrplot)
library(patchwork)
library(fable)
library(Metrics)
library(xgboost)
library(fastDummies)
library(caret)
library(forecast)
library(kableExtra)
```

Importing the Data

```
weather_data <- read.csv("weatherAUS.csv")
str(weather_data$Date)
```

```
##  chr [1:145460] "2008-12-01" "2008-12-02" "2008-12-03" "2008-12-04" ...
```

Preview of Data

```
str(weather_data)      # Structure of the dataset
```

```
## 'data.frame':    145460 obs. of  23 variables:
##  $ Date         : chr  "2008-12-01" "2008-12-02" "2008-12-03" "2008-12-04" ...
##  $ Location     : chr  "Albury" "Albury" "Albury" "Albury" ...
##  $ MinTemp      : num  13.4 7.4 12.9 9.2 17.5 14.6 14.3 7.7 9.7 13.1 ...
##  $ MaxTemp      : num  22.9 25.1 25.7 28 32.3 29.7 25 26.7 31.9 30.1 ...
##  $ Rainfall     : num  0.6 0 0 0 1 0.2 0 0 0 1.4 ...
##  $ Evaporation  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Sunshine     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ WindGustDir  : chr  "W" "WNW" "WSW" "NE" ...
##  $ WindGustSpeed: int  44 44 46 24 41 56 50 35 80 28 ...
##  $ WindDir9am   : chr  "W" "NNW" "W" "SE" ...
##  $ WindDir3pm   : chr  "WNW" "WSW" "WSW" "E" ...
##  $ WindSpeed9am : int  20 4 19 11 7 19 20 6 7 15 ...
##  $ WindSpeed3pm : int  24 22 26 9 20 24 24 17 28 11 ...
##  $ Humidity9am  : int  71 44 38 45 82 55 49 48 42 58 ...
```

```
##  $ Humidity3pm  : int  22 25 30 16 33 23 19 19 9 27 ...
##  $ Pressure9am  : num  1008 1011 1008 1018 1011 ...
##  $ Pressure3pm  : num  1007 1008 1009 1013 1006 ...
##  $ Cloud9am     : int  8 NA NA NA 7 NA 1 NA NA NA ...
##  $ Cloud3pm     : int  NA NA 2 NA 8 NA NA NA NA NA ...
##  $ Temp9am      : num  16.9 17.2 21 18.1 17.8 20.6 18.1 16.3 18.3 20.1 ...
##  $ Temp3pm      : num  21.8 24.3 23.2 26.5 29.7 28.9 24.6 25.5 30.2 28.2 ...
##  $ RainToday    : chr  "No" "No" "No" "No" ...
##  $ RainTomorrow : chr  "No" "No" "No" "No" ...
```

```
summary(weather_data)  # Summary statistics for each column
```

```
##      Date             Location           MinTemp         MaxTemp
##  Length:145460      Length:145460      Min.   :-8.50   Min.   :-4.80
##  Class :character   Class :character   1st Qu.: 7.60   1st Qu.:17.90
##  Mode  :character   Mode  :character   Median :12.00   Median :22.60
##                                        Mean   :12.19   Mean   :23.22
##                                        3rd Qu.:16.90   3rd Qu.:28.20
##                                        Max.   :33.90   Max.   :48.10
##                                        NA's   :1485    NA's   :1261
##     Rainfall        Evaporation        Sunshine     WindGustDir
##  Min.   :  0.000   Min.   :  0.00   Min.   : 0.00   Length:145460
##  1st Qu.:  0.000   1st Qu.:  2.60   1st Qu.: 4.80   Class :character
##  Median :  0.000   Median :  4.80   Median : 8.40   Mode  :character
##  Mean   :  2.361   Mean   :  5.47   Mean   : 7.61
##  3rd Qu.:  0.800   3rd Qu.:  7.40   3rd Qu.:10.60
##  Max.   :371.000   Max.   :145.00   Max.   :14.50
##  NA's   :3261      NA's   :62790    NA's   :69835
##  WindGustSpeed     WindDir9am         WindDir3pm         WindSpeed9am
##  Min.   :  6.00   Length:145460      Length:145460      Min.   :  0.00
##  1st Qu.: 31.00   Class :character   Class :character   1st Qu.:  7.00
##  Median : 39.00   Mode  :character   Mode  :character   Median : 13.00
##  Mean   : 40.03                                        Mean   : 14.04
##  3rd Qu.: 48.00                                        3rd Qu.: 19.00
##  Max.   :135.00                                        Max.   :130.00
##  NA's   :10263                                         NA's   :1767
##   WindSpeed3pm    Humidity9am      Humidity3pm      Pressure9am
##  Min.   : 0.00   Min.   :  0.00   Min.   :  0.00   Min.   : 980.5
##  1st Qu.:13.00   1st Qu.: 57.00   1st Qu.: 37.00   1st Qu.:1012.9
##  Median :19.00   Median : 70.00   Median : 52.00   Median :1017.6
##  Mean   :18.66   Mean   : 68.88   Mean   : 51.54   Mean   :1017.6
##  3rd Qu.:24.00   3rd Qu.: 83.00   3rd Qu.: 66.00   3rd Qu.:1022.4
##  Max.   :87.00   Max.   :100.00   Max.   :100.00   Max.   :1041.0
##  NA's   :3062    NA's   :2654     NA's   :4507     NA's   :15065
##   Pressure3pm        Cloud9am        Cloud3pm        Temp9am
##  Min.   : 977.1   Min.   :0.00    Min.   :0.00    Min.   :-7.20
##  1st Qu.:1010.4   1st Qu.:1.00    1st Qu.:2.00    1st Qu.:12.30
##  Median :1015.2   Median :5.00    Median :5.00    Median :16.70
##  Mean   :1015.3   Mean   :4.45    Mean   :4.51    Mean   :16.99
##  3rd Qu.:1020.0   3rd Qu.:7.00    3rd Qu.:7.00    3rd Qu.:21.60
##  Max.   :1039.6   Max.   :9.00    Max.   :9.00    Max.   :40.20
##  NA's   :15028    NA's   :55888   NA's   :59358   NA's   :1767
##     Temp3pm       RainToday         RainTomorrow
##  Min.   :-5.40   Length:145460      Length:145460
##  1st Qu.:16.60   Class :character   Class :character
```

```
##  Median :21.10   Mode  :character   Mode  :character
##  Mean   :21.68
##  3rd Qu.:26.40
##  Max.   :46.70
##  NA's   :3609
```

```
head(weather_data)
```

```
##         Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine WindGustDir
## 1 2008-12-01   Albury    13.4    22.9      0.6          NA       NA           W
## 2 2008-12-02   Albury     7.4    25.1      0.0          NA       NA         WNW
## 3 2008-12-03   Albury    12.9    25.7      0.0          NA       NA         WSW
## 4 2008-12-04   Albury     9.2    28.0      0.0          NA       NA          NE
## 5 2008-12-05   Albury    17.5    32.3      1.0          NA       NA           W
## 6 2008-12-06   Albury    14.6    29.7      0.2          NA       NA         WNW
##   WindGustSpeed WindDir9am WindDir3pm WindSpeed9am WindSpeed3pm Humidity9am
## 1            44          W        WNW           20           24          71
## 2            44        NNW        WSW            4           22          44
## 3            46          W        WSW           19           26          38
## 4            24         SE          E           11            9          45
## 5            41        ENE         NW            7           20          82
## 6            56          W          W           19           24          55
##   Humidity3pm Pressure9am Pressure3pm Cloud9am Cloud3pm Temp9am Temp3pm
## 1          22      1007.7      1007.1        8       NA    16.9    21.8
## 2          25      1010.6      1007.8       NA       NA    17.2    24.3
## 3          30      1007.6      1008.7       NA        2    21.0    23.2
## 4          16      1017.6      1012.8       NA       NA    18.1    26.5
## 5          33      1010.8      1006.0        7        8    17.8    29.7
## 6          23      1009.2      1005.4       NA       NA    20.6    28.9
##   RainToday RainTomorrow
## 1        No           No
## 2        No           No
## 3        No           No
## 4        No           No
## 5        No           No
## 6        No           No
```

Data Preprocessing

Convert Date Column

```
weather_data$Date <- as.Date(weather_data$Date, format = "%Y-%m-%d")
#weather_data <- as_tsibble(weather_data, index = Date)
```

```
weather_data_monthly <- weather_data |>
  mutate(
    Date = as.Date(Date),  # Convert Date to proper format
    year_month = format(Date, "%Y-%m"),  # Extract Year-Month for grouping
    AvgWindSpeed = (WindSpeed9am + WindSpeed3pm) / 2,
    AvgHumidity = (Humidity9am + Humidity3pm) / 2,
    AvgPressure = (Pressure9am + Pressure3pm) / 2,
    AvgCloud = (Cloud9am + Cloud3pm) / 2,
    AvgTemp = (Temp9am + Temp3pm) / 2
  ) |>
  group_by(Location, year_month) |>  # Group by Location and year_month
```

```
  summarize(
    AvgWindSpeed = mean(AvgWindSpeed, na.rm = TRUE),
    AvgHumidity = mean(AvgHumidity, na.rm = TRUE),
    AvgPressure = mean(AvgPressure, na.rm = TRUE),
    AvgCloud = mean(AvgCloud, na.rm = TRUE),
    AvgTemp = mean(AvgTemp, na.rm = TRUE),
    MinTemp = mean(MinTemp, na.rm = TRUE),
    MaxTemp = mean(MaxTemp, na.rm = TRUE),
    Rainfall = sum(Rainfall, na.rm = TRUE),
    .groups = "drop"
  ) |>
  mutate(
    Rained = ifelse(Rainfall > 0, 1, 0)  # Binary column: 1 if Rainfall > 0, else 0
  )
```

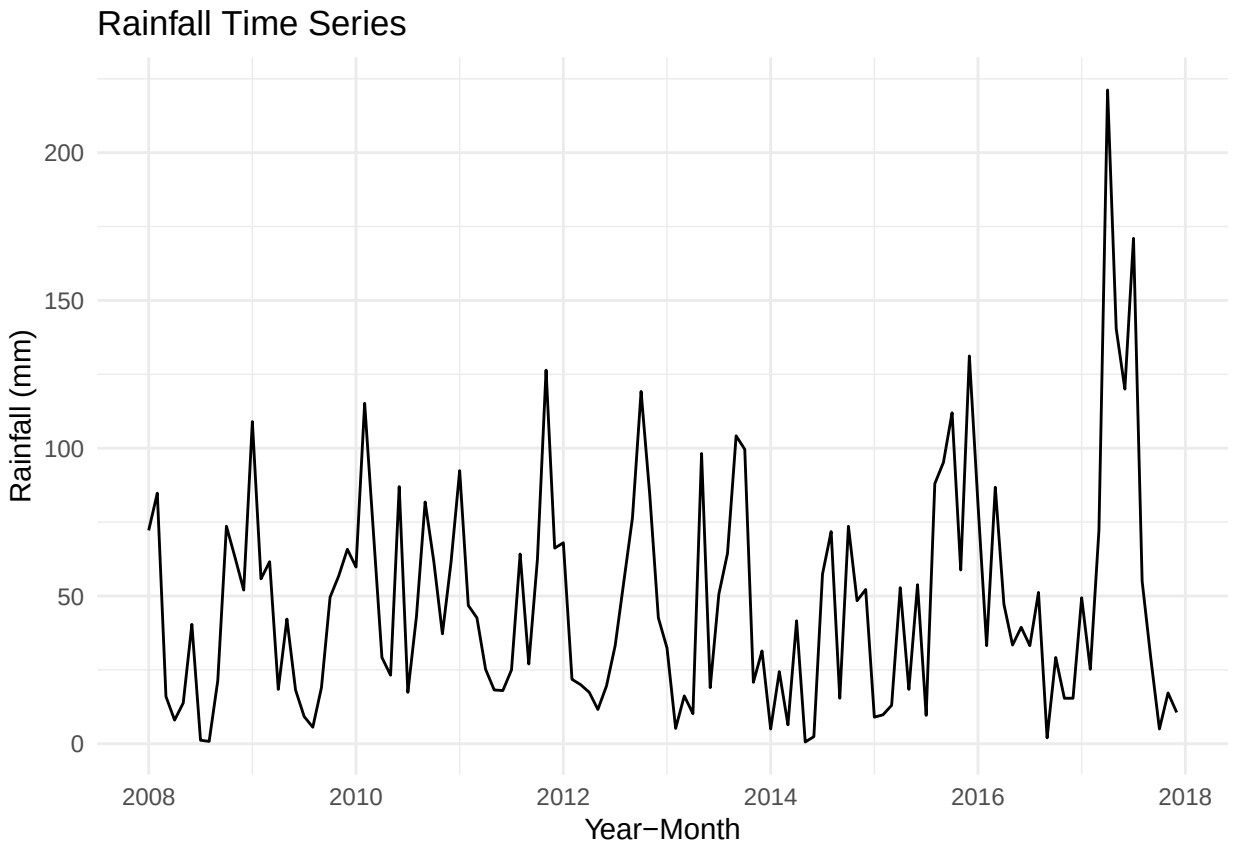
```
weather_data_monthly$year_month <- yearmonth(weather_data_monthly$year_month)
class(weather_data_monthly$year_month)
```

```
## [1] "yearmonth"  "vctrs_vctr"
```

```
weather_data_ts <- weather_data_monthly |>
  as_tsibble(index = year_month, key = Location)

head(weather_data_ts)
```

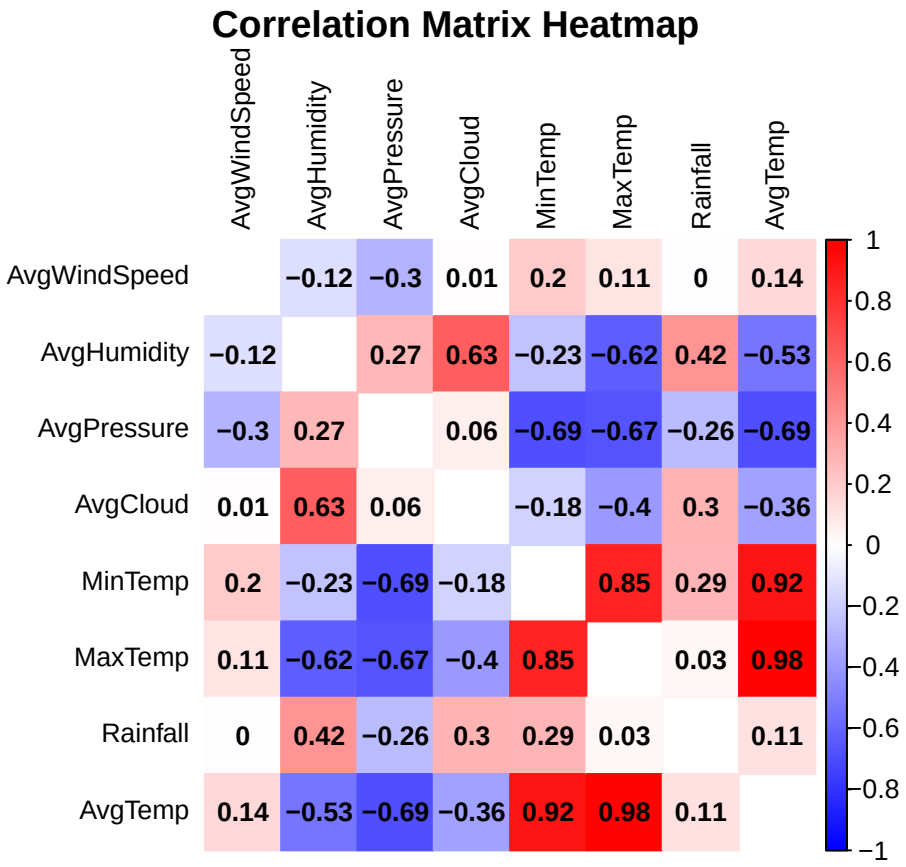
```
## # A tsibble: 6 x 11 [1M]
## # Key:       Location [1]
##   Location year_month AvgWindSpeed AvgHumidity AvgPressure AvgCloud AvgTemp
##   <chr>         <mth>        <dbl>       <dbl>       <dbl>    <dbl>   <dbl>
## 1 Adelaide   2008 Jul         14.6        64.2       1018.      NaN    12.3
## 2 Adelaide   2008 Aug         13.4        66.6       1024.      NaN    12.0
## 3 Adelaide   2008 Sep         16.4        45         1018.      NaN    17.3
## 4 Adelaide   2008 Oct         14.3        40.2       1020.      NaN    20.1
## 5 Adelaide   2008 Nov         15.4        44.2       1012.      NaN    20.7
## 6 Adelaide   2008 Dec         14.3        48.5       1011.      NaN    21.6
## # i 4 more variables: MinTemp <dbl>, MaxTemp <dbl>, Rainfall <dbl>,
## #   Rained <dbl>
```

```
# Create the time series plot
weather_data_ts$Rainfall <- as.numeric(weather_data_ts$Rainfall)
rainfall_ts <- ts(weather_data_ts$Rainfall, start = c(2008, 1), end = c(2017, 12), frequency = 12)

# Use autoplot() to plot the time series
autoplot(rainfall_ts) +
  labs(title = "Rainfall Time Series", x = "Year-Month", y = "Rainfall (mm)") +
  theme_minimal()
```



Correlation matrix

```r
# Numeric features
numeric_features <- weather_data_ts[c("AvgWindSpeed", "AvgHumidity", "AvgPressure", "AvgCloud", "MinTem

# Ensure all selected features are numeric
numeric_features <- sapply(numeric_features, as.numeric)

# Calculate the correlation matrix for all numeric columns
cor_matrix <- cor(numeric_features, use = "complete.obs")

# Generate the correlation heatmap
corrplot(cor_matrix,
         method = "color",  # Use colors to represent correlation values
         col = colorRampPalette(c("blue", "white", "red"))(200),  # Color scale
         title = "Correlation Matrix Heatmap",
         addCoef.col = "black",  # Add correlation coefficients on the plot
         number.cex = 0.8,  # Size of the coefficients
         diag = FALSE,  # Hide diagonal
         tl.col = "black",  # Text label color
         tl.cex = 0.8,  # Text label size
         mar = c(0, 0, 1, 0))  # Margins around the plot
```



Scatter plot features vs Rainfall

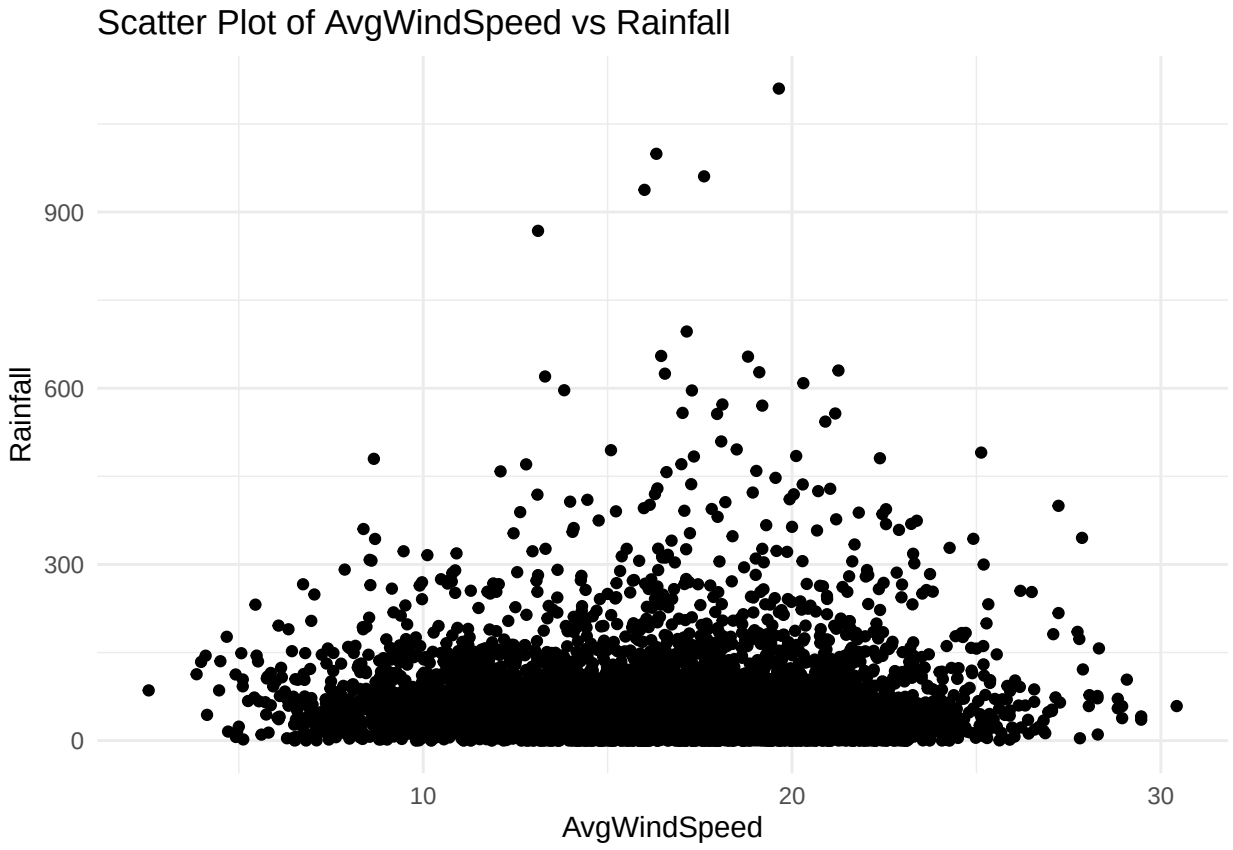
```
# List of features to plot against Rainfall
features <- c("AvgWindSpeed", "AvgHumidity", "AvgPressure", "AvgCloud", "MinTemp", "MaxTemp", "Rainfall

# Check for missing values in these columns and impute the mean if there are any
weather_data_ts[features] <- suppressWarnings(weather_data_ts[features] |>
  mutate(across(
    all_of(features),
    ~ ifelse(is.na(.), mean(., na.rm = TRUE), .)
  )))

# Verify that missing values have been handled
colSums(is.na(weather_data_ts[features]))
```

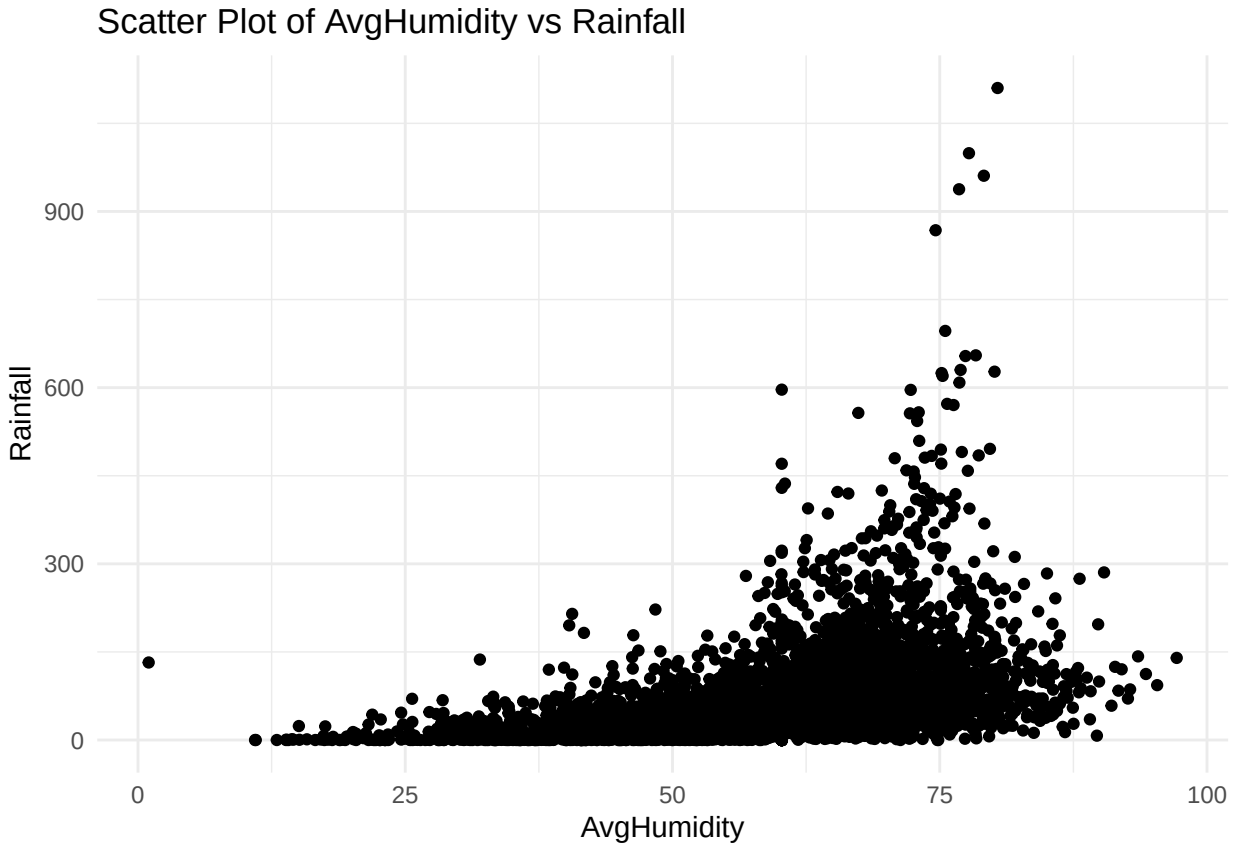
```
## AvgWindSpeed  AvgHumidity  AvgPressure      AvgCloud      MinTemp      MaxTemp
##            0            0            0            0            0            0
##     Rainfall      AvgTemp
##            0            0
```

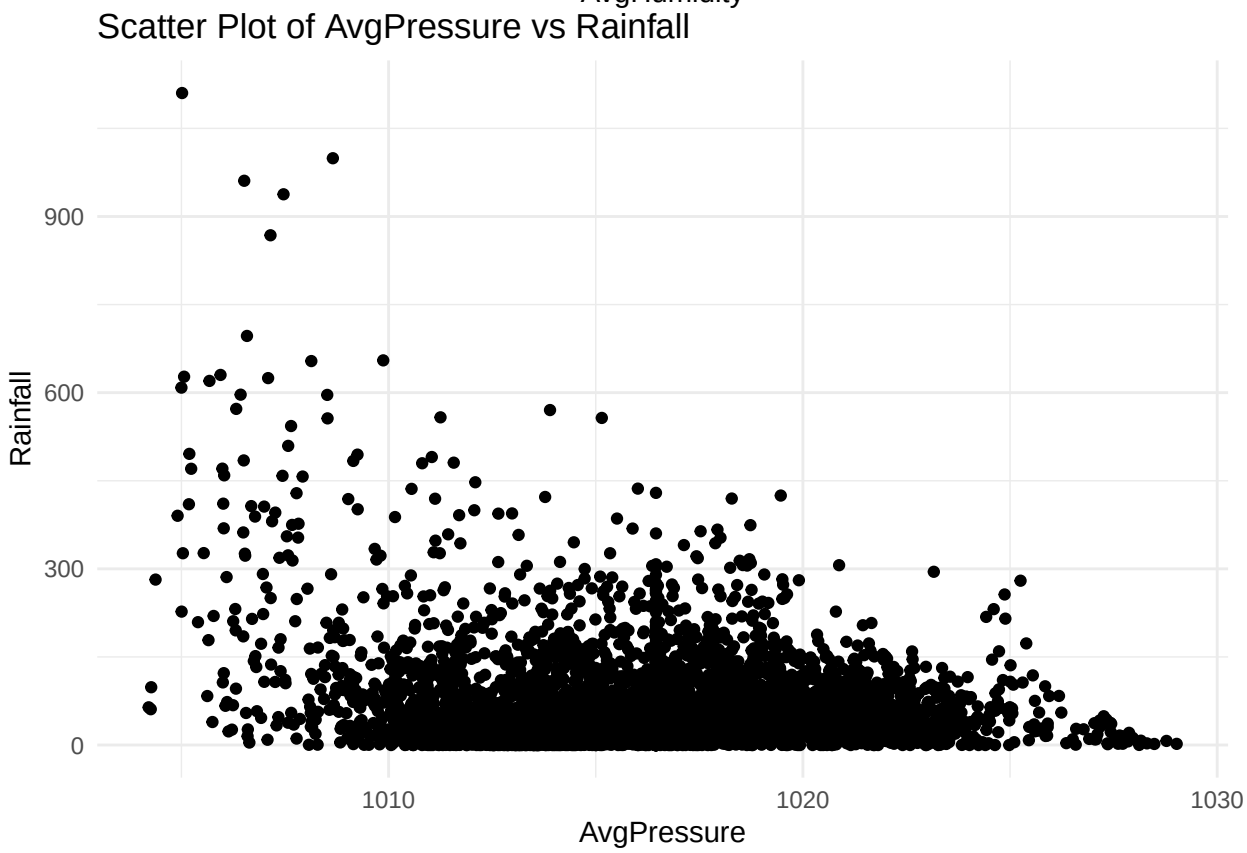
```
# Create scatter plots for each feature vs Rainfall
for (feature in features) {
  plot <- ggplot(weather_data_ts, aes_string(x = feature, y = "Rainfall")) +
    geom_point() +
    labs(title = paste("Scatter Plot of", feature, "vs Rainfall"),
         x = feature,
```

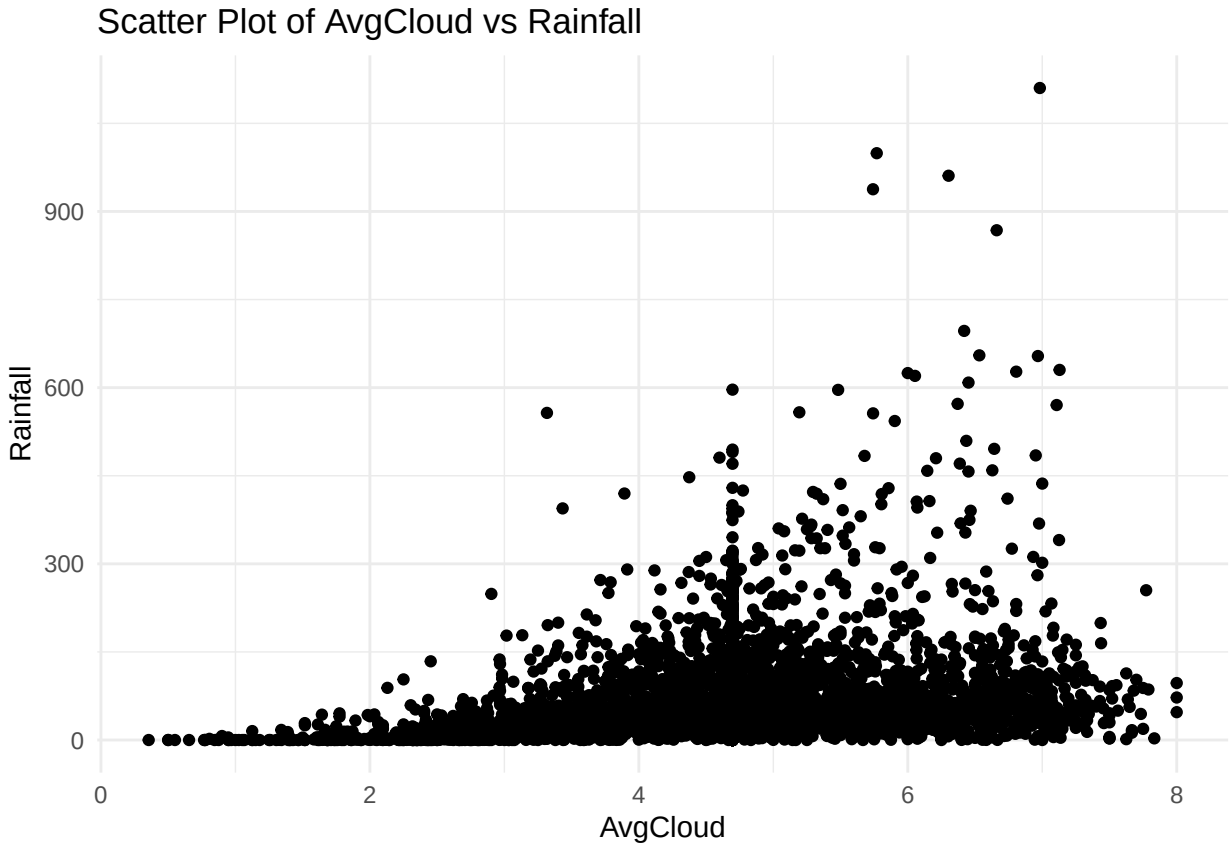
```
         y = "Rainfall") +
    theme_minimal()
  print(plot)
}
```

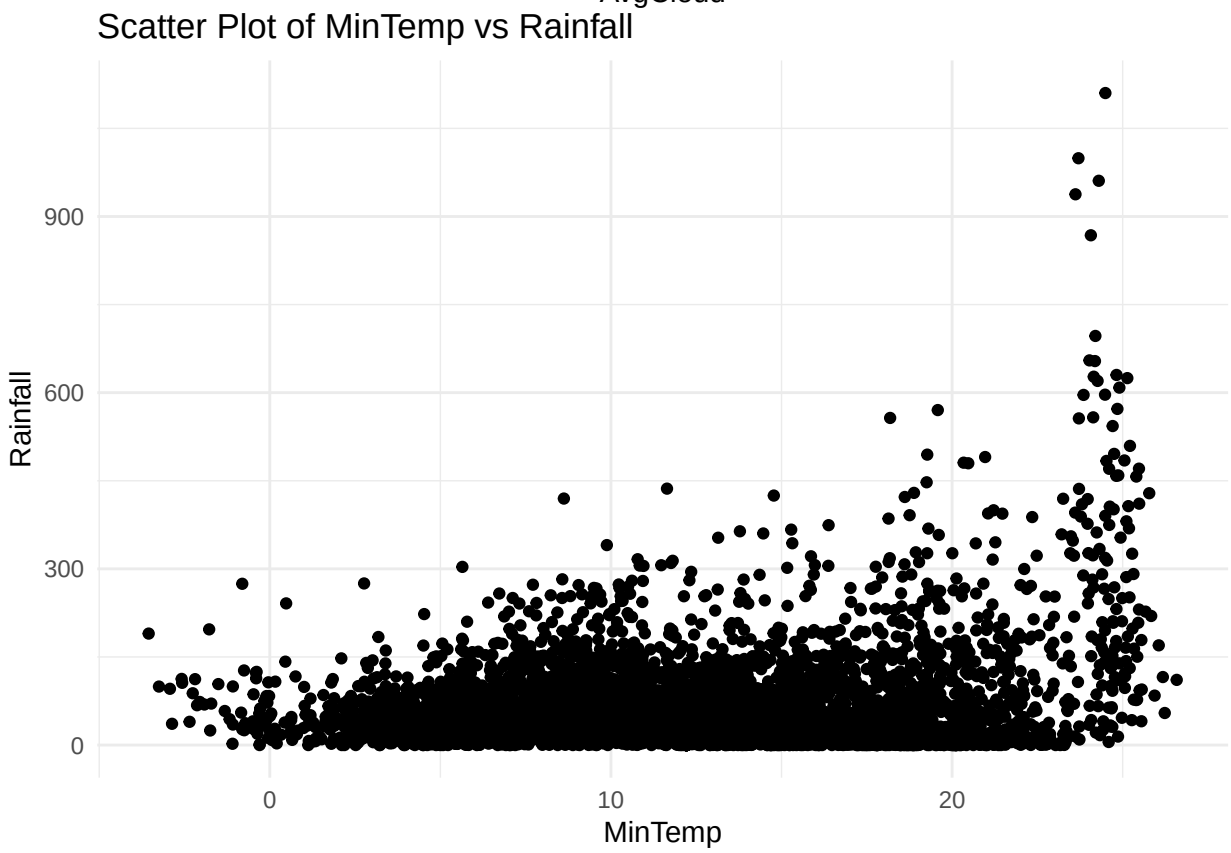
```
## Warning: `aes_string()` was deprecated in ggplot2 3.0.0.
## i Please use tidy evaluation idioms with `aes()`.
## i See also `vignette("ggplot2-in-packages")` for more information.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

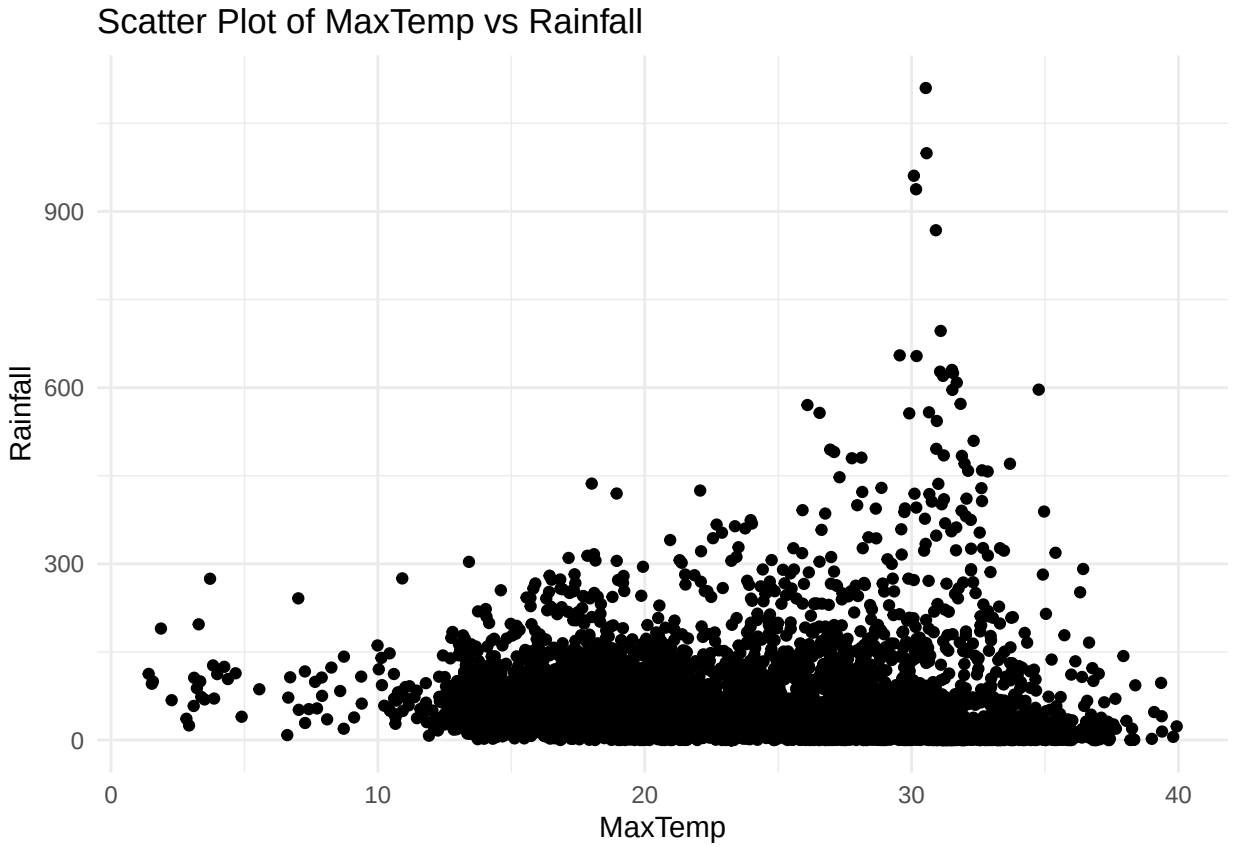


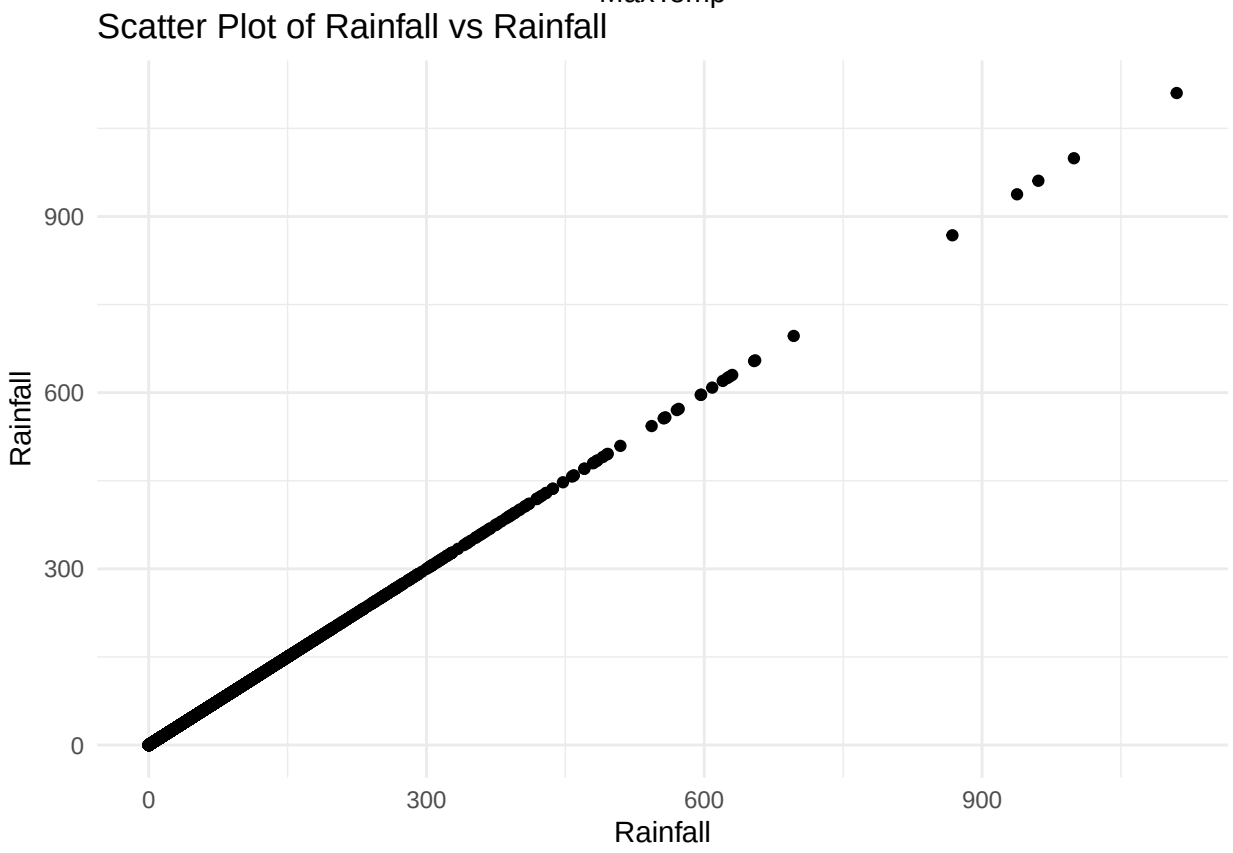

Scatter Plot of AvgHumidity vs Rainfall
Rainfall
0
300
600
900
0
25
50
75
100
AvgHumidity

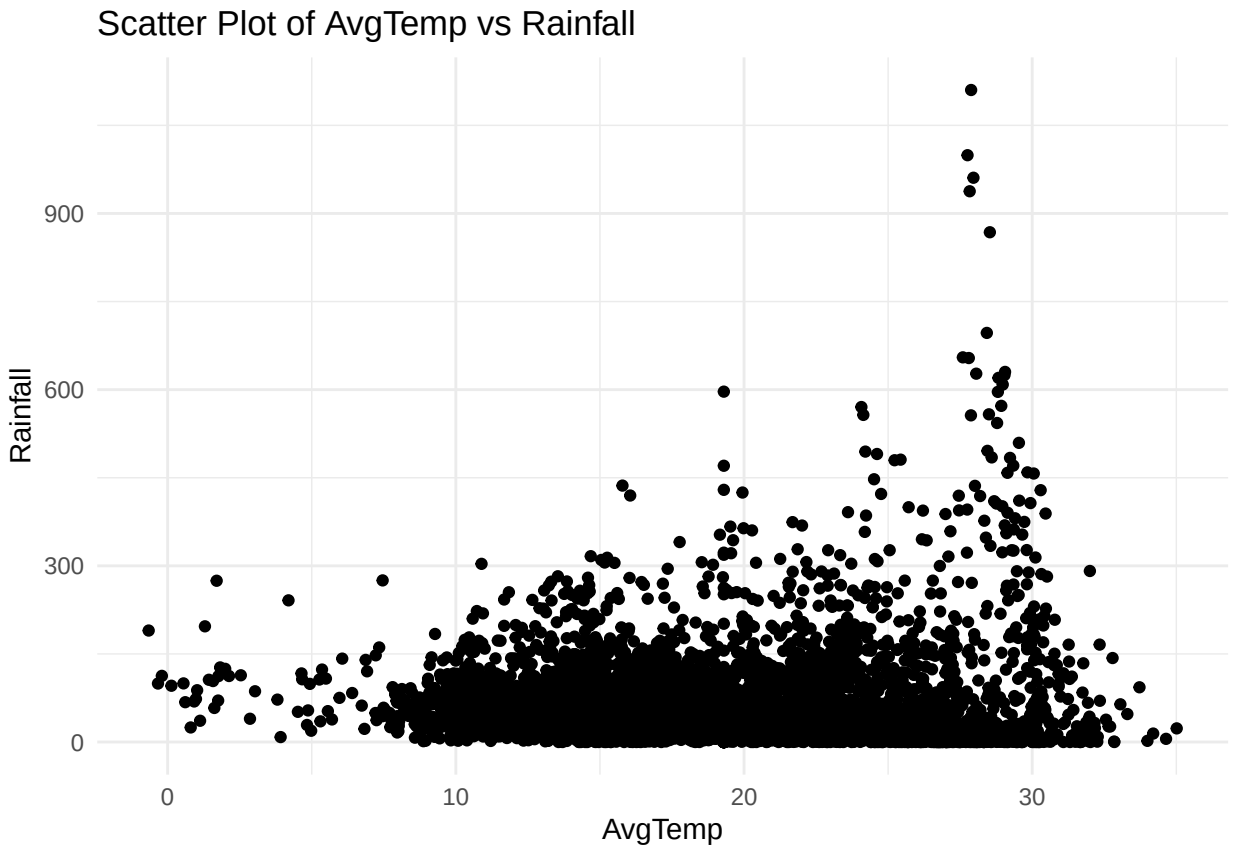

Scatter Plot of AvgPressure vs Rainfall
Rainfall
0
300
600
900
1010
1020
1030
AvgPressure


Scatter Plot of AvgCloud vs Rainfall
Rainfall
900
600
300
0
0
2
4
6
8
AvgCloud


Scatter Plot of MinTemp vs Rainfall
Rainfall
900
600
300
0
0
10
20
MinTemp


Scatter Plot of MaxTemp vs Rainfall
Rainfall
900
600
300
0
0
10
20
30
40
MaxTemp


Scatter Plot of Rainfall vs Rainfall
Rainfall
900
600
300
0
0
300
600
900
Rainfall



Data Cleaning

Handle Missing Values

```
colSums(is.na(weather_data_monthly))
```

```
##     Location   year_month AvgWindSpeed  AvgHumidity  AvgPressure     AvgCloud
##            0            0           64           99          476         1459
##      AvgTemp      MinTemp      MaxTemp     Rainfall       Rained
##           85           17           17            0            0
```

Impute missing values

```
weather_data_monthly$AvgWindSpeed[is.na(weather_data_monthly$AvgWindSpeed)] <- mean(weather_data_monthly
weather_data_monthly$MinTemp[is.na(weather_data_monthly$MinTemp)] <- mean(weather_data_monthly$MinTemp,
weather_data_monthly$AvgHumidity[is.na(weather_data_monthly$AvgHumidity)] <- mean(weather_data_monthly$
weather_data_monthly$AvgPressure[is.na(weather_data_monthly$AvgPressure)] <- mean(weather_data_monthly$
weather_data_monthly$AvgCloud[is.na(weather_data_monthly$AvgCloud)] <- mean(weather_data_monthly$AvgClou
weather_data_monthly$AvgTemp[is.na(weather_data_monthly$AvgTemp)] <- mean(weather_data_monthly$AvgTemp,
weather_data_monthly$MaxTemp[is.na(weather_data_monthly$MaxTemp)] <- mean(weather_data_monthly$MaxTemp,
colSums(is.na(weather_data_monthly))
```

```
##     Location   year_month AvgWindSpeed  AvgHumidity  AvgPressure     AvgCloud
##            0            0            0            0            0            0
##      AvgTemp      MinTemp      MaxTemp     Rainfall       Rained
```

```
##            0            0            0            0            0
```

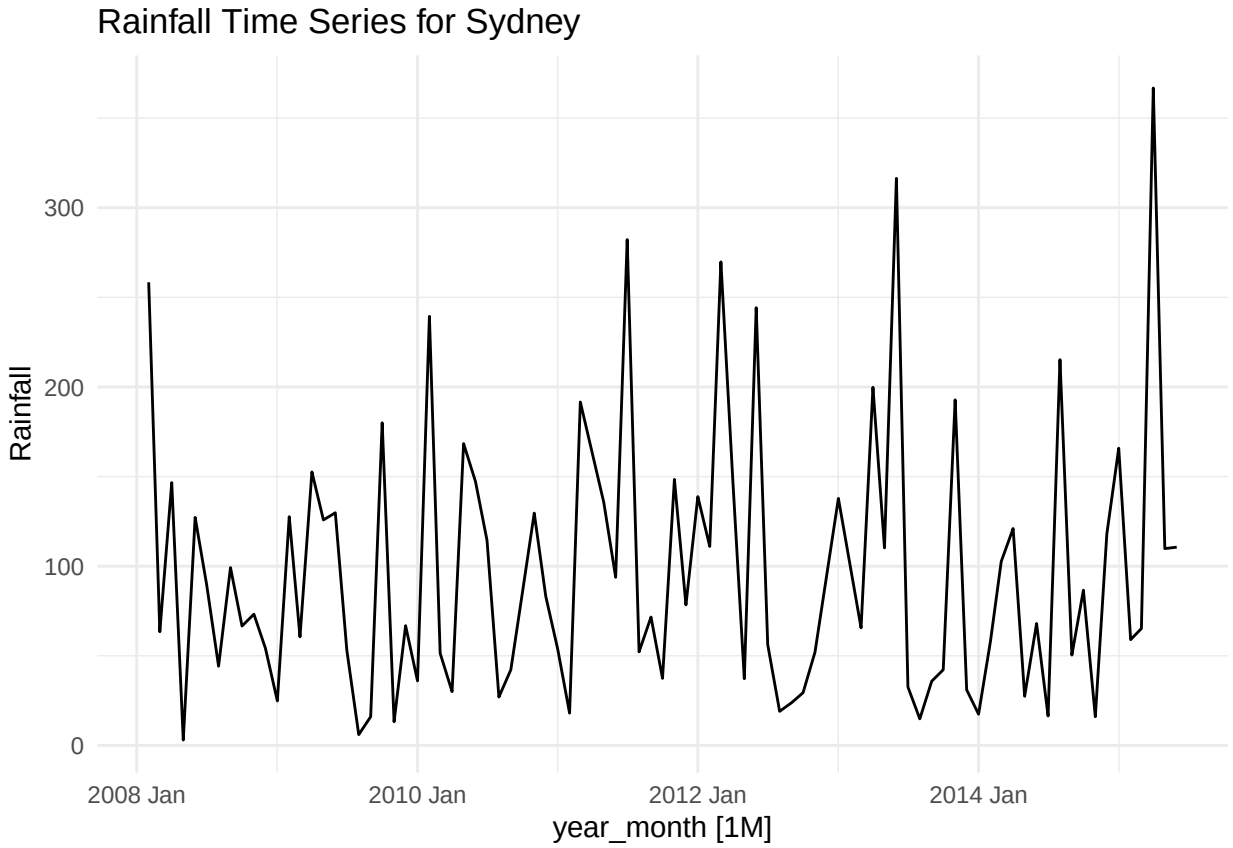
Split the data into train and test sets

```
train_data <- weather_data_monthly |>
  filter(year_month >= yearmonth("2007 Nov") & year_month <= yearmonth("2015 Jun"))

test_data <- weather_data_monthly |>
  filter(year_month >= yearmonth("2015 Jul") & year_month <= yearmonth("2017 Jun"))
```

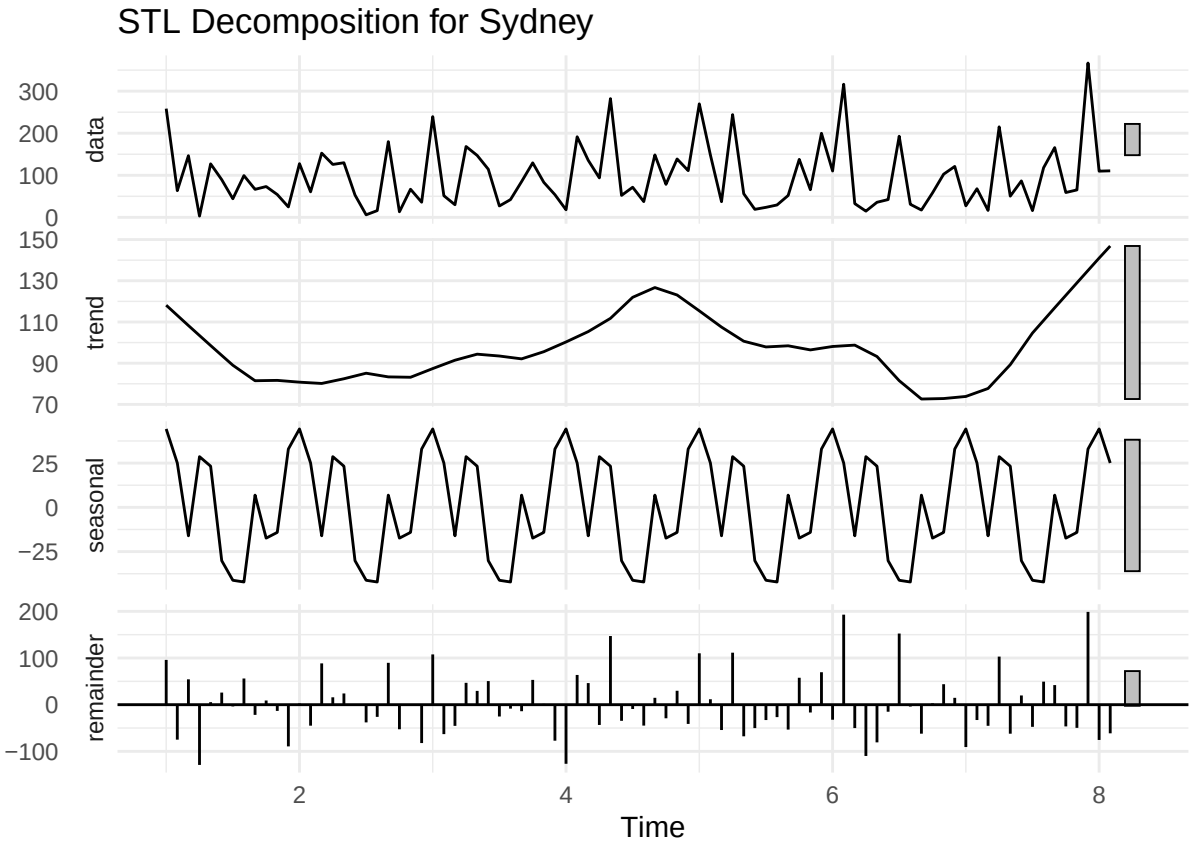
Time Series Plots for four cities, decompose, differencing

```
# Initialize an empty list to store results for each city
city_results <- list()

# Split the data into train and test sets
train_data <- weather_data_monthly |>
  filter(year_month >= yearmonth("2007 Nov") & year_month <= yearmonth("2015 Jun"))

test_data <- weather_data_monthly |>
  filter(year_month >= yearmonth("2015 Jul") & year_month <= yearmonth("2017 Jun"))

# Time Series Plots for four cities, decompose, differencing
cities <- c("Sydney", "Perth", "Darwin", "Melbourne")

for (city in cities) {
  # Filter data for the current city
  city_train_data <- train_data |> filter(Location == city)

  # Convert to tsibble and create a time series object
  city_train_tsibble <- city_train_data |>
    as_tsibble(index = year_month, key = Location)

  # Ensure Rainfall is a numeric vector and create a time series object
  rainfall_ts <- ts(city_train_tsibble$Rainfall, frequency = 12)

  # Time Series Plot
  ts_plot <- city_train_tsibble |>
    autoplot(Rainfall) +
    ggtitle(paste("Rainfall Time Series for", city)) +
    theme_minimal()

  print(ts_plot)  # Display the time series plot

  # Stationarity check using Augmented Dickey-Fuller Test
  adf_result <- adf.test(rainfall_ts)

  cat("ADF Test for", city, "\n")
  print(adf_result)

  # If the time series is non-stationary, apply differencing once
  if (adf_result$p.value >= 0.05) {
```

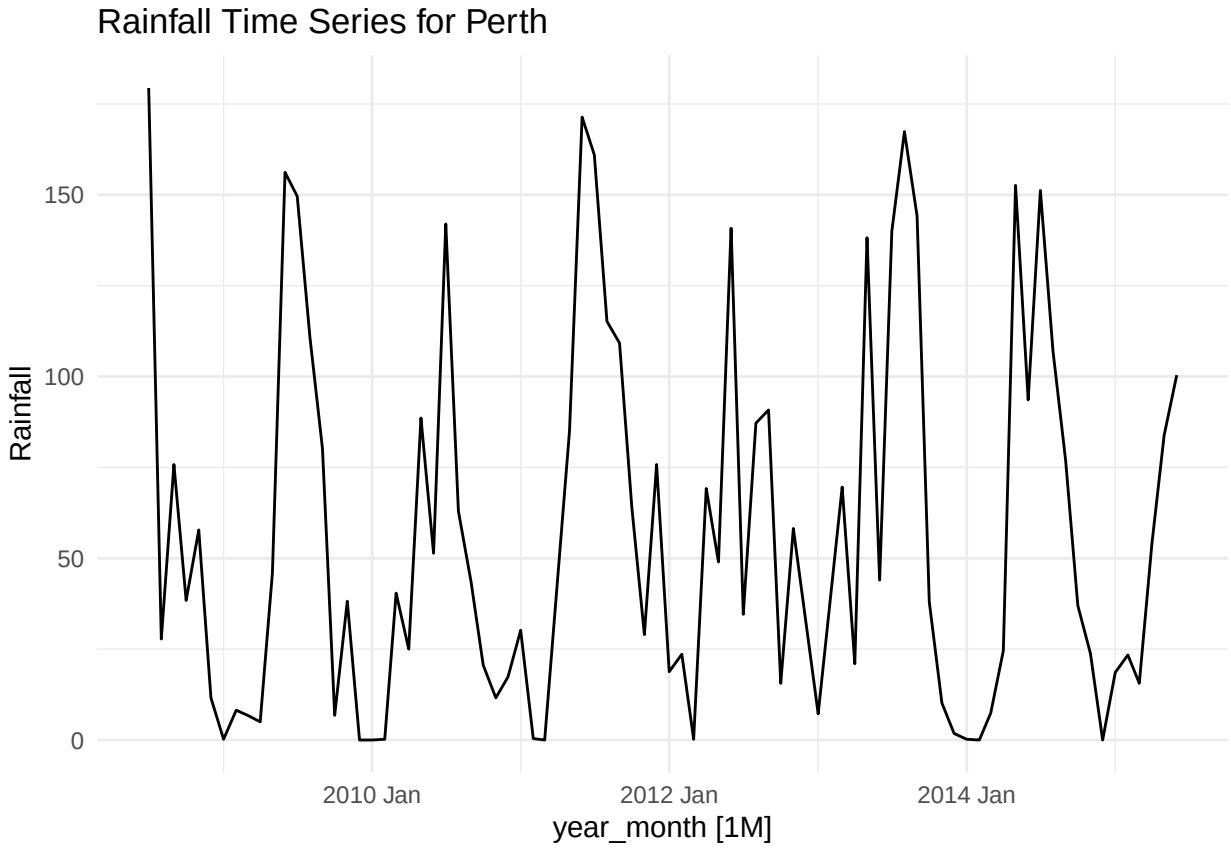
```
    cat("Conclusion: The time series for", city, "is non-stationary. Applying differencing.\n\n")

    # Apply differencing to make the series stationary (only once)
    diff_rainfall_ts <- diff(rainfall_ts)

    # Store differenced series in the results list
    city_results[[city]]$diff_rainfall_ts <- diff_rainfall_ts

    # Re-perform the ADF Test on the differenced series
    adf_result_diff <- adf.test(diff_rainfall_ts)

    cat("ADF Test after differencing for", city, "\n")
    print(adf_result_diff)

    if (adf_result_diff$p.value < 0.05) {
      cat("Conclusion: The differenced series for", city, "is stationary.\n\n")

      # Apply decomposition on the differenced series
      city_decomposition <- stl(diff_rainfall_ts, s.window = "periodic")

      # Plot decomposition
      decomposition_plot <- autoplot(city_decomposition) +
        ggtitle(paste("STL Decomposition for", city, "After Differencing")) +
        theme_minimal()

      print(decomposition_plot)  # Display the decomposition plot
    } else {
      cat("Conclusion: The differenced series for", city, "is still non-stationary. Additional differen
    }
  } else {
    cat("Conclusion: The time series for", city, "is stationary.\n\n")

    # If stationary, apply decomposition directly
    city_decomposition <- stl(rainfall_ts, s.window = "periodic")

    # Plot decomposition
    decomposition_plot <- autoplot(city_decomposition) +
      ggtitle(paste("STL Decomposition for", city)) +
      theme_minimal()

    print(decomposition_plot)  # Display the decomposition plot
  }

  # Store results for the current city
  city_results[[city]] <- list(
    time_series_plot = ts_plot,
    decomposed = city_decomposition
  )
}
```



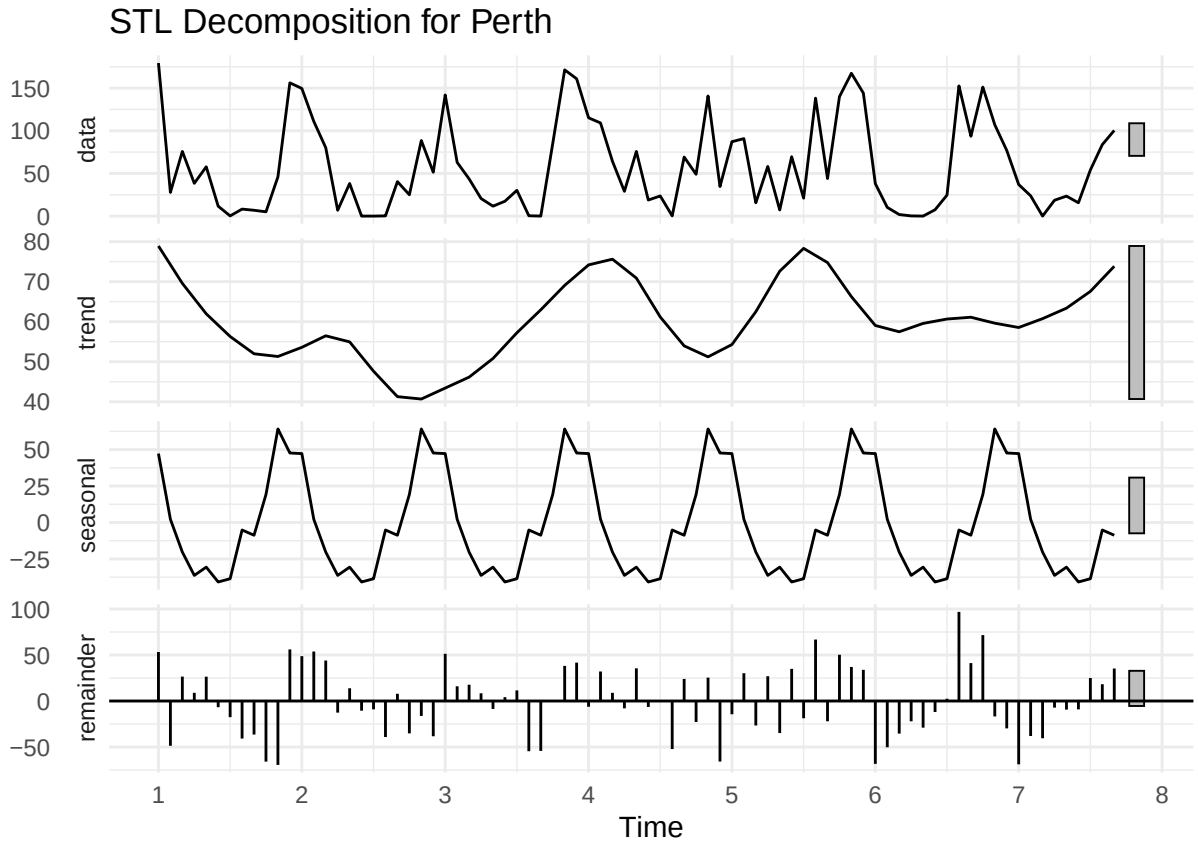

```
## ADF Test for Sydney
##
##  Augmented Dickey-Fuller Test
##
## data:  rainfall_ts
## Dickey-Fuller = -3.901, Lag order = 4, p-value = 0.01801
## alternative hypothesis: stationary
##
## Conclusion: The time series for Sydney is stationary.
```



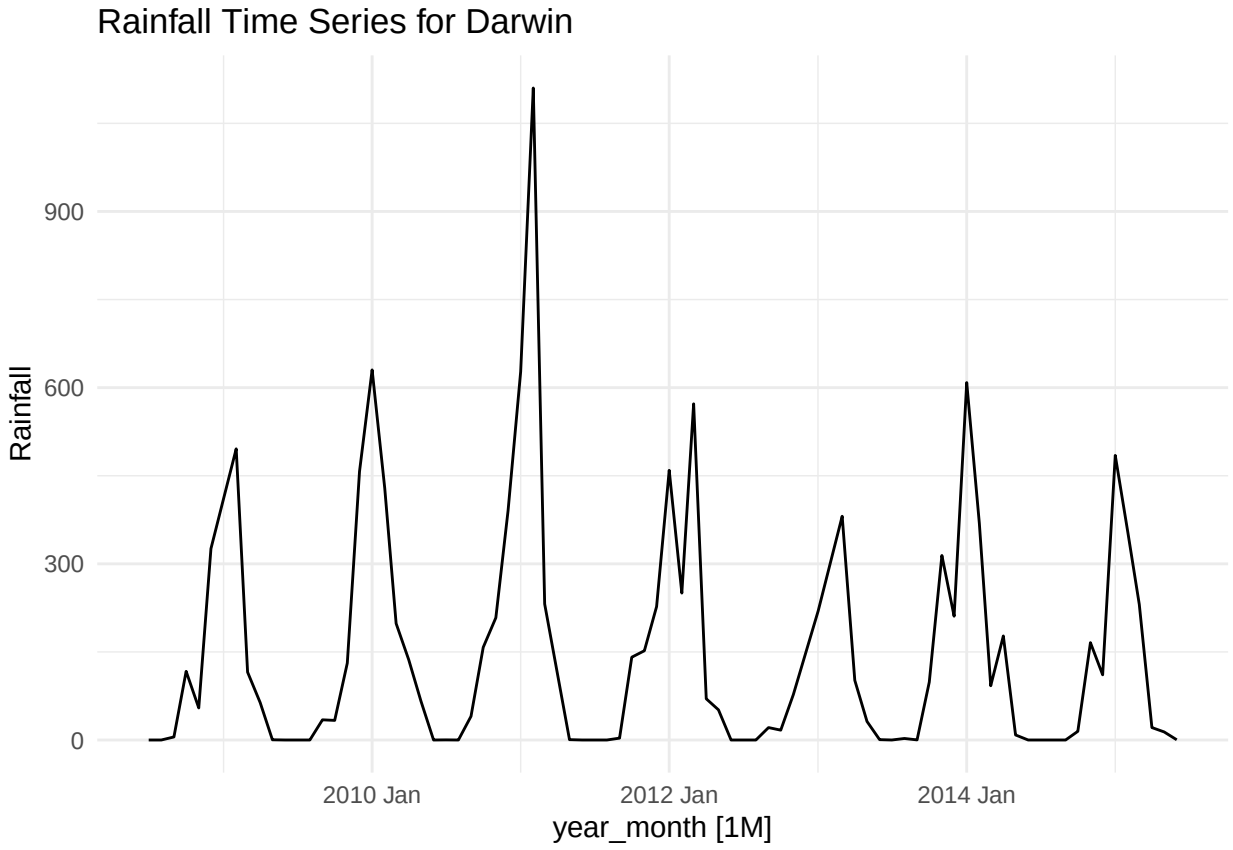

```
## Warning in adf.test(rainfall_ts): p-value smaller than printed p-value
```



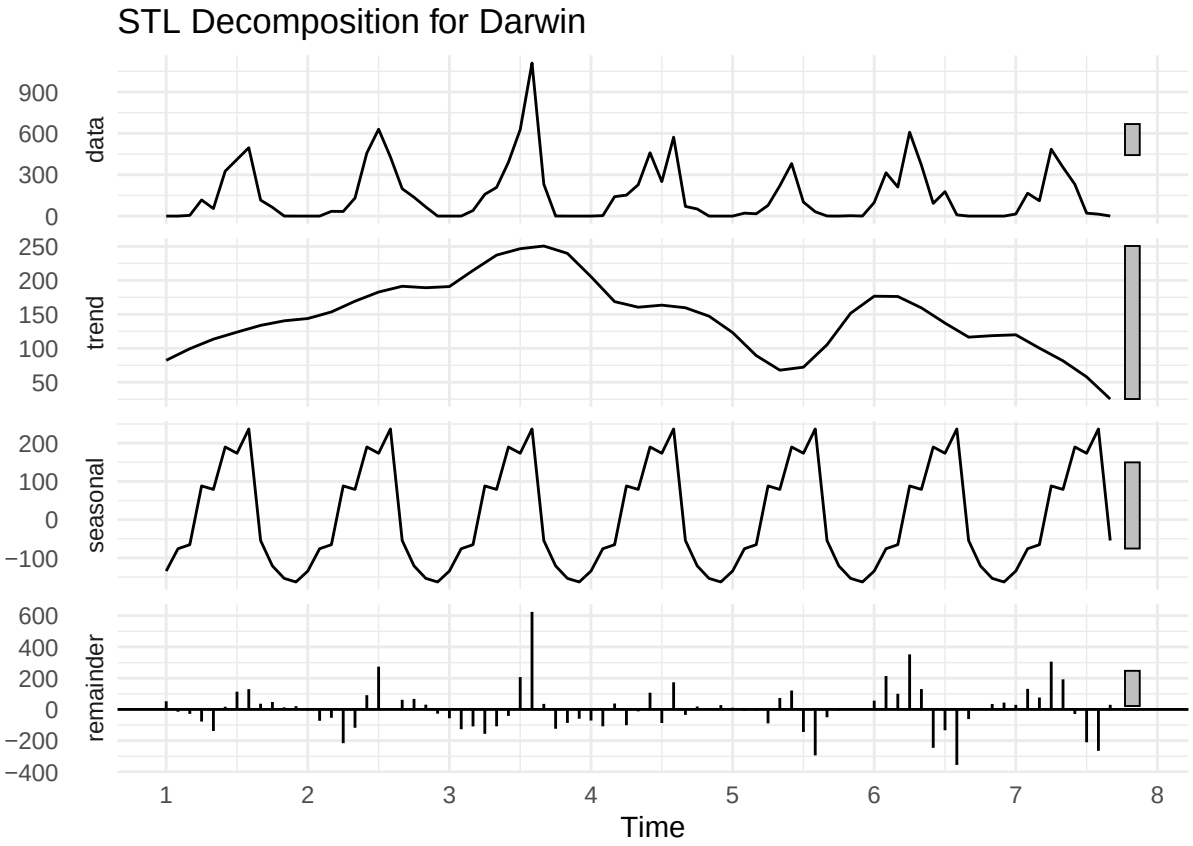

```
## ADF Test for Perth
##
##  Augmented Dickey-Fuller Test
##
## data:  rainfall_ts
## Dickey-Fuller = -6.2144, Lag order = 4, p-value = 0.01
## alternative hypothesis: stationary
##
## Conclusion: The time series for Perth is stationary.
```



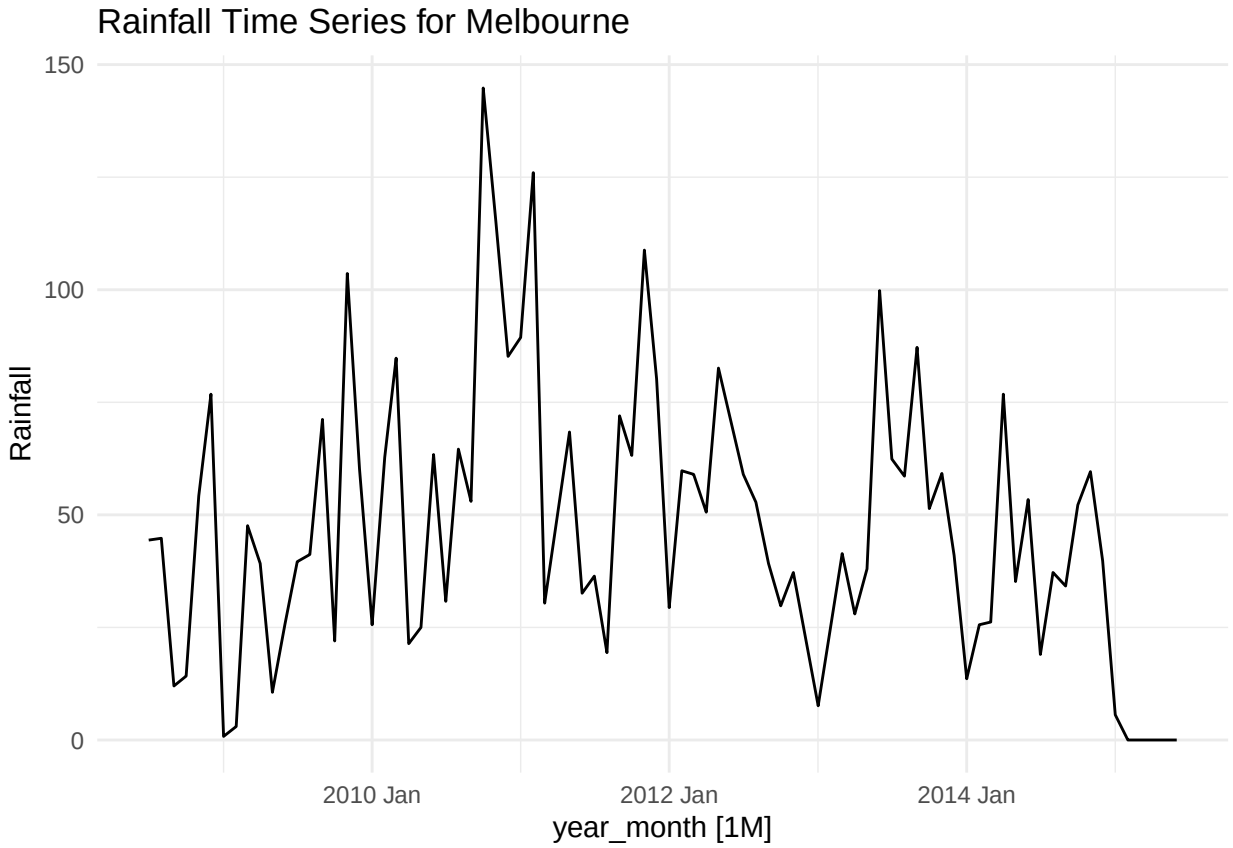

```
## Warning in adf.test(rainfall_ts): p-value smaller than printed p-value
```



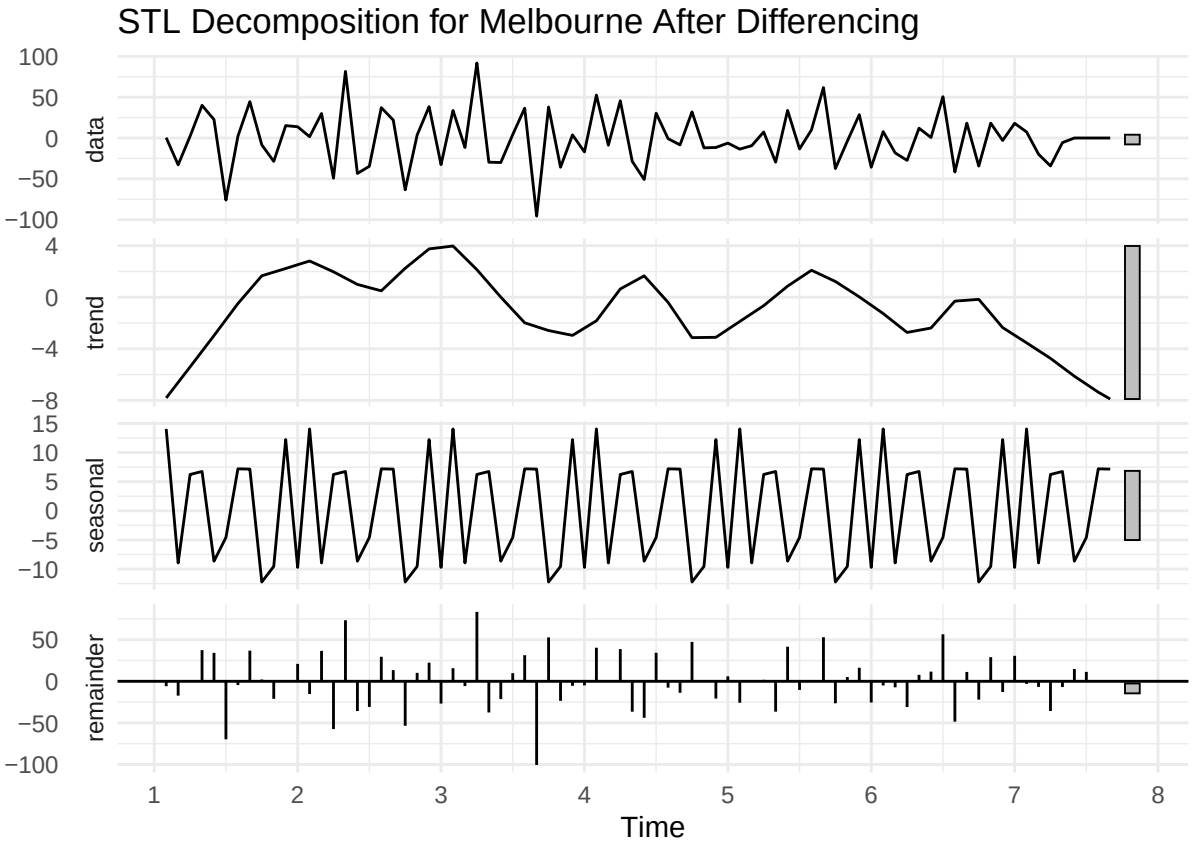

```
## ADF Test for Darwin
##
##  Augmented Dickey-Fuller Test
##
## data:  rainfall_ts
## Dickey-Fuller = -5.7776, Lag order = 4, p-value = 0.01
## alternative hypothesis: stationary
##
## Conclusion: The time series for Darwin is stationary.
```




```
## ADF Test for Melbourne
##
##  Augmented Dickey-Fuller Test
##
## data:  rainfall_ts
## Dickey-Fuller = -3.2429, Lag order = 4, p-value = 0.08686
## alternative hypothesis: stationary
##
## Conclusion: The time series for Melbourne is non-stationary. Applying differencing.
## Warning in adf.test(diff_rainfall_ts): p-value smaller than printed p-value
```



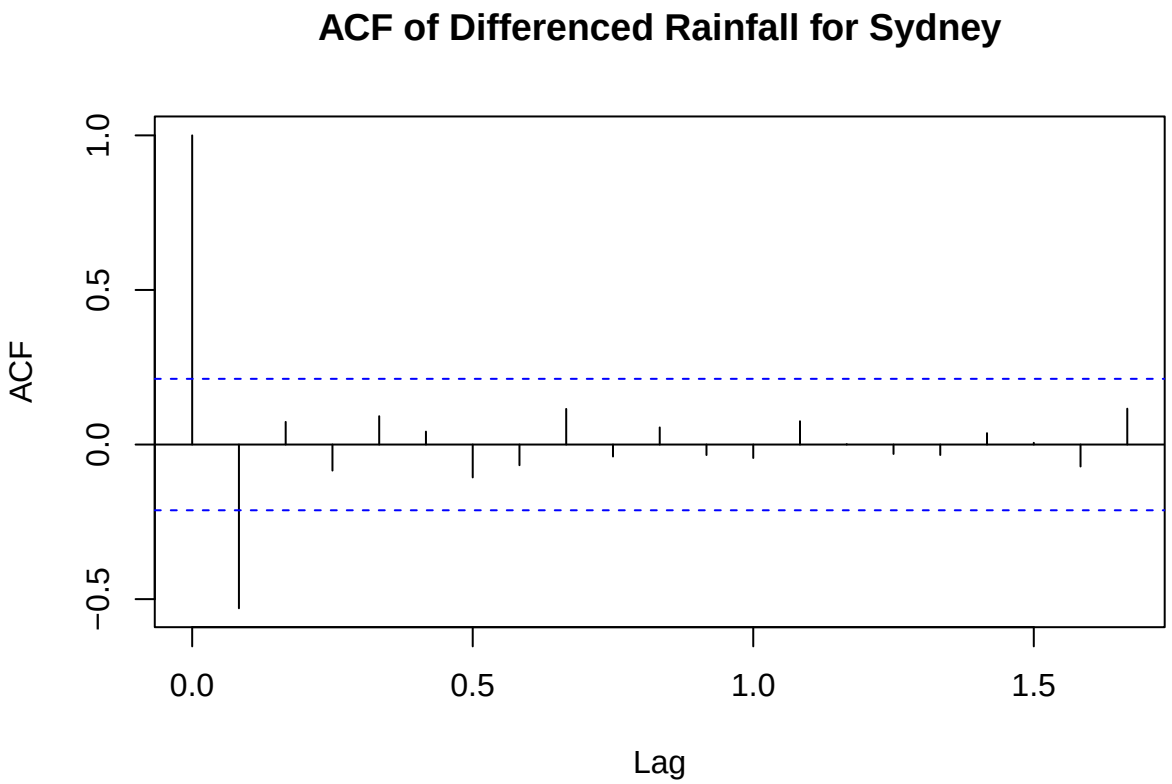

```
## ADF Test after differencing for Melbourne
##
##  Augmented Dickey-Fuller Test
##
## data:  diff_rainfall_ts
## Dickey-Fuller = -5.2042, Lag order = 4, p-value = 0.01
## alternative hypothesis: stationary
##
## Conclusion: The differenced series for Melbourne is stationary.
```

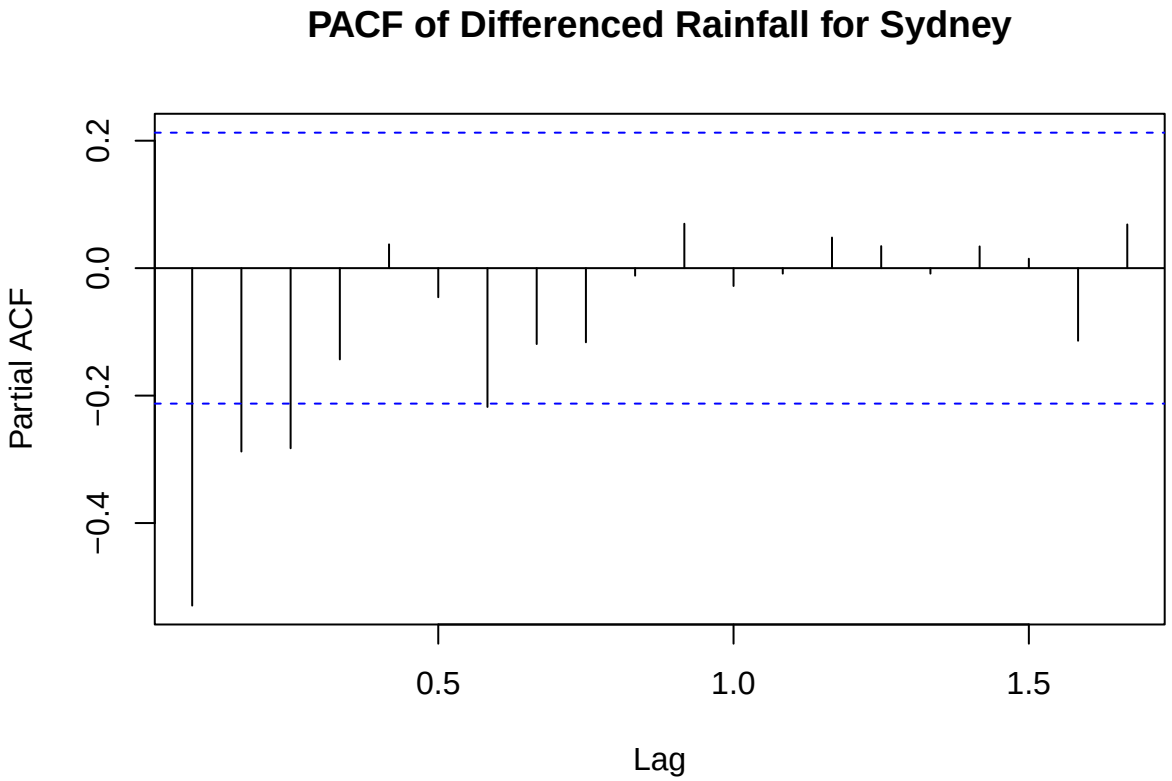


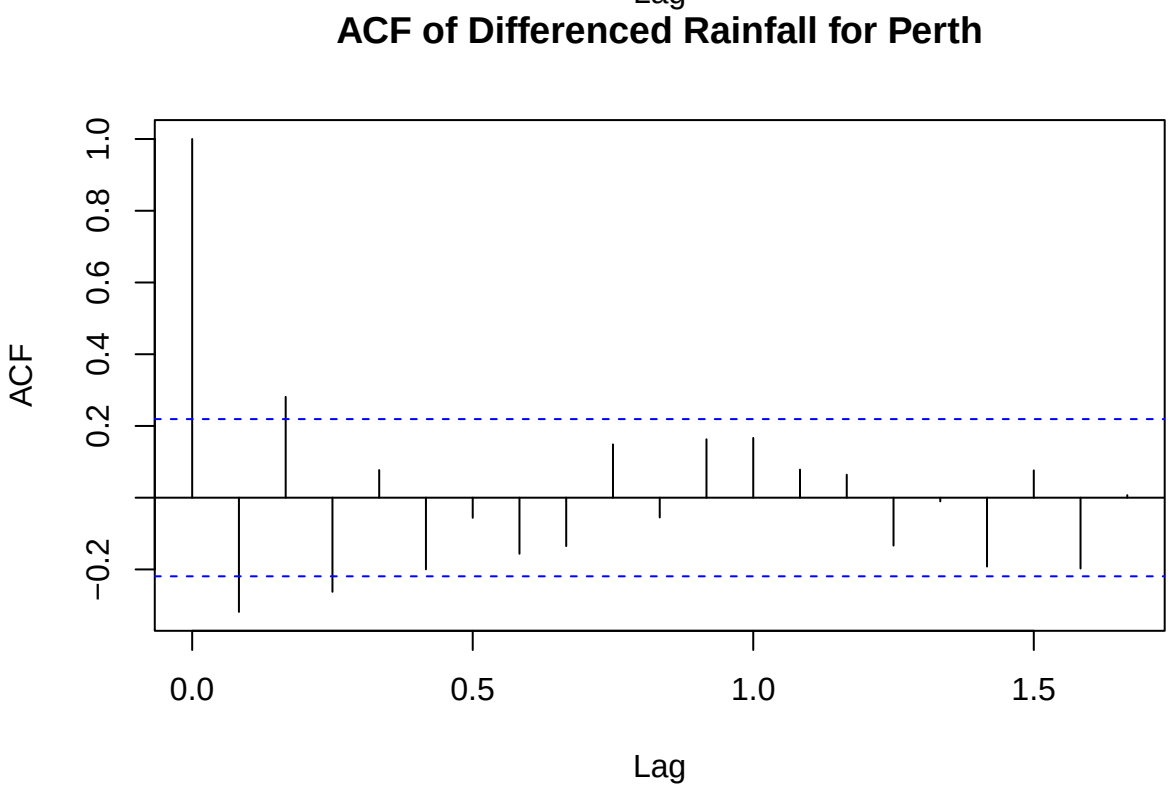
ACF and PACF for ARIMA Model Identification

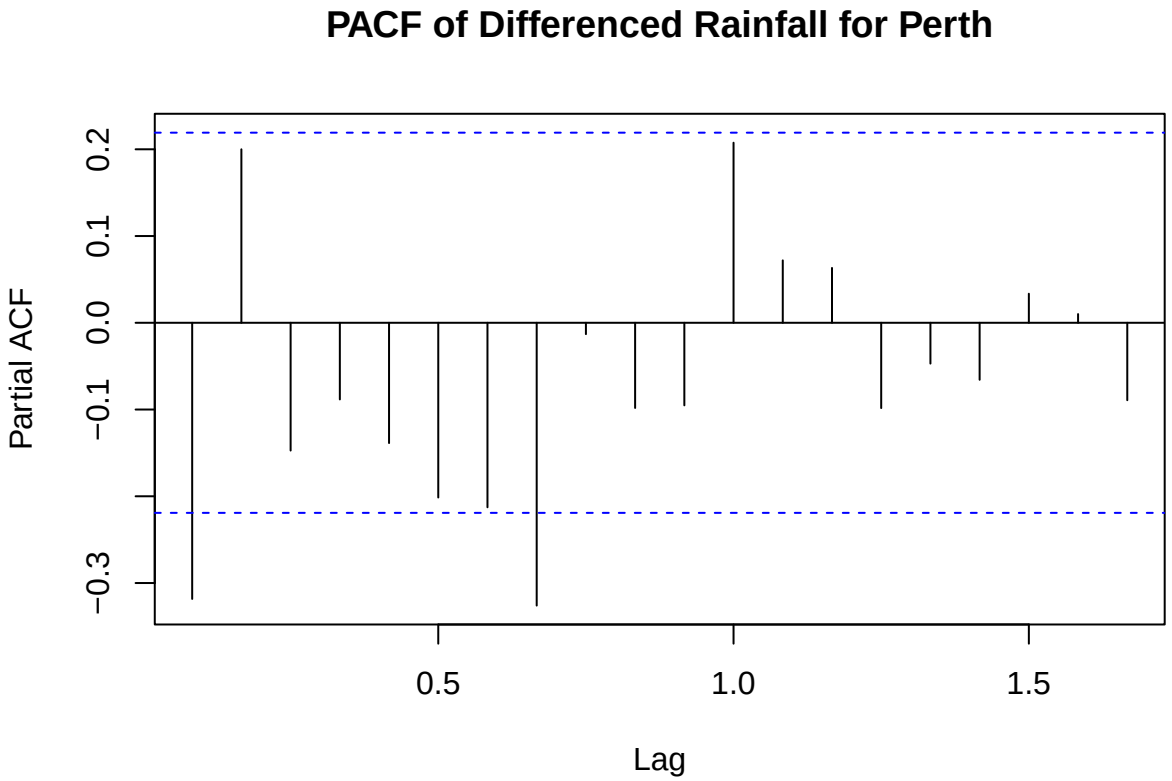
```r
# Define the four cities of interest
cities <- c("Sydney", "Perth", "Darwin", "Melbourne")

# Loop through the four cities to generate ACF and PACF plots
for (city in cities) {
  # Filter data for the current city
  city_train_data <- train_data |> filter(Location == city)

  # Check if data for the city exists
  if (nrow(city_train_data) == 0) {
    message(paste("No data available for", city, "- Skipping."))
    next
  }

  # Convert to tsibble and create a time series object
  city_train_tsibble <- city_train_data |>
    as_tsibble(index = year_month, key = Location)

  # Ensure Rainfall is a numeric vector and create a time series object
  rainfall_ts <- ts(city_train_tsibble$Rainfall, frequency = 12)

  # Apply differencing (only once)
  diff_rainfall_ts <- diff(rainfall_ts)
```

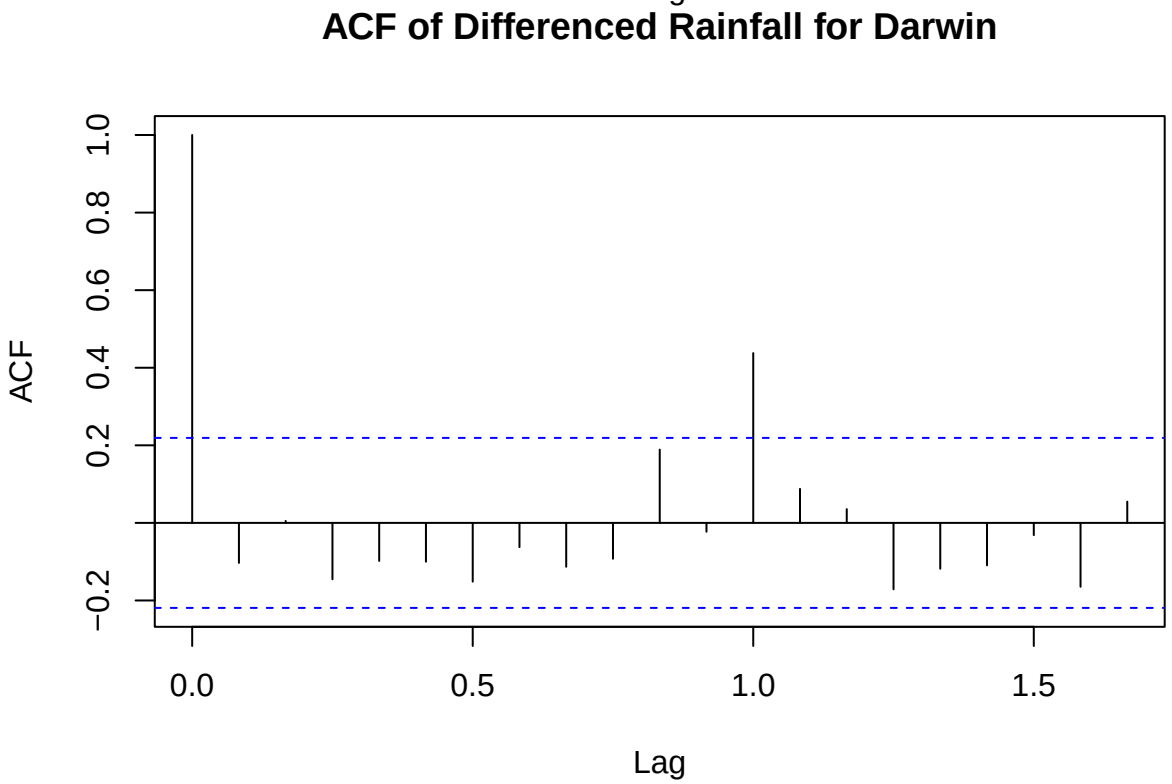
```r
  # ACF Plot for Differenced Rainfall for the current city
  acf(diff_rainfall_ts, lag.max = 20, main = paste("ACF of Differenced Rainfall for", city))

  # PACF Plot for Differenced Rainfall for the current city
  pacf(diff_rainfall_ts, lag.max = 20, main = paste("PACF of Differenced Rainfall for", city))
}
```

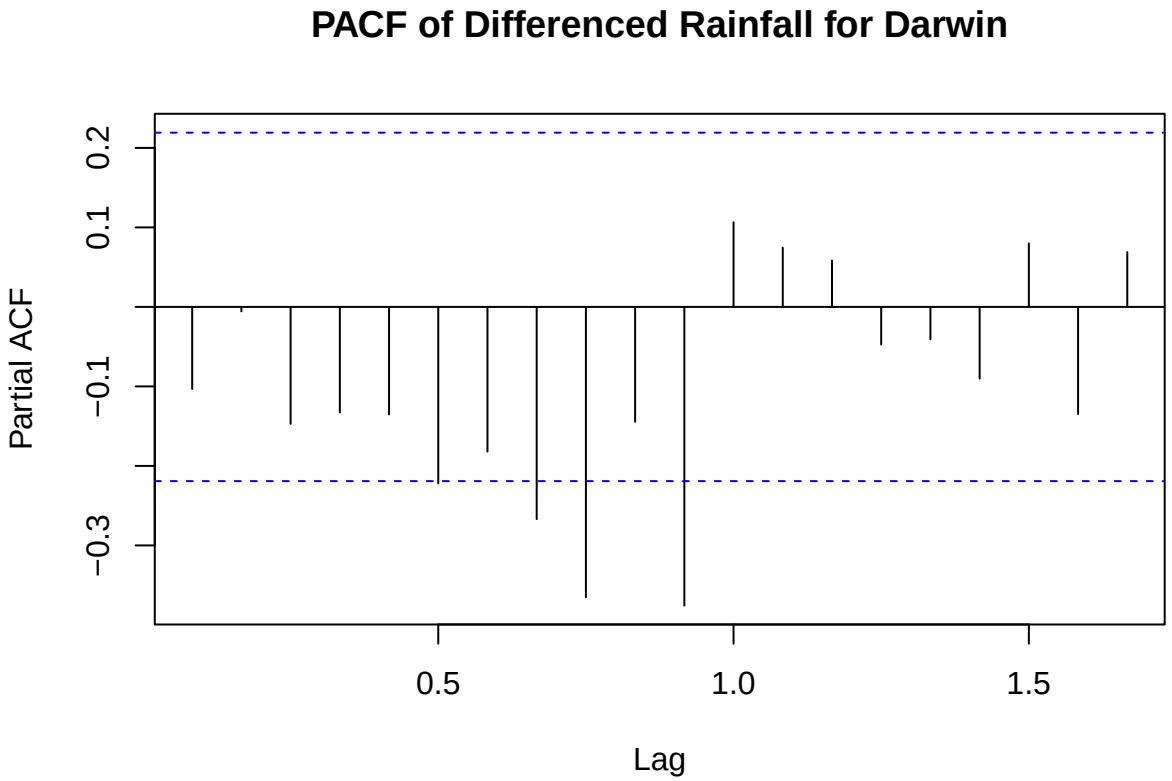


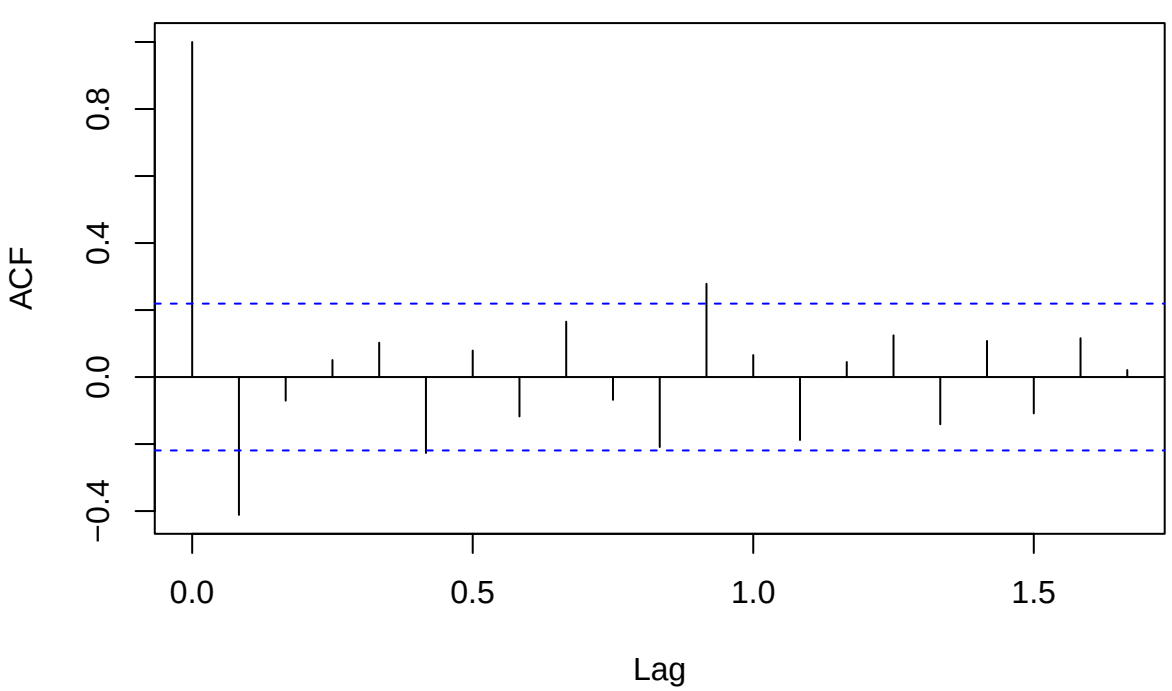

PACF of Differenced Rainfall for Sydney
Partial ACF
0.2
0.0
−0.2
−0.4
0.5
1.0
1.5
Lag

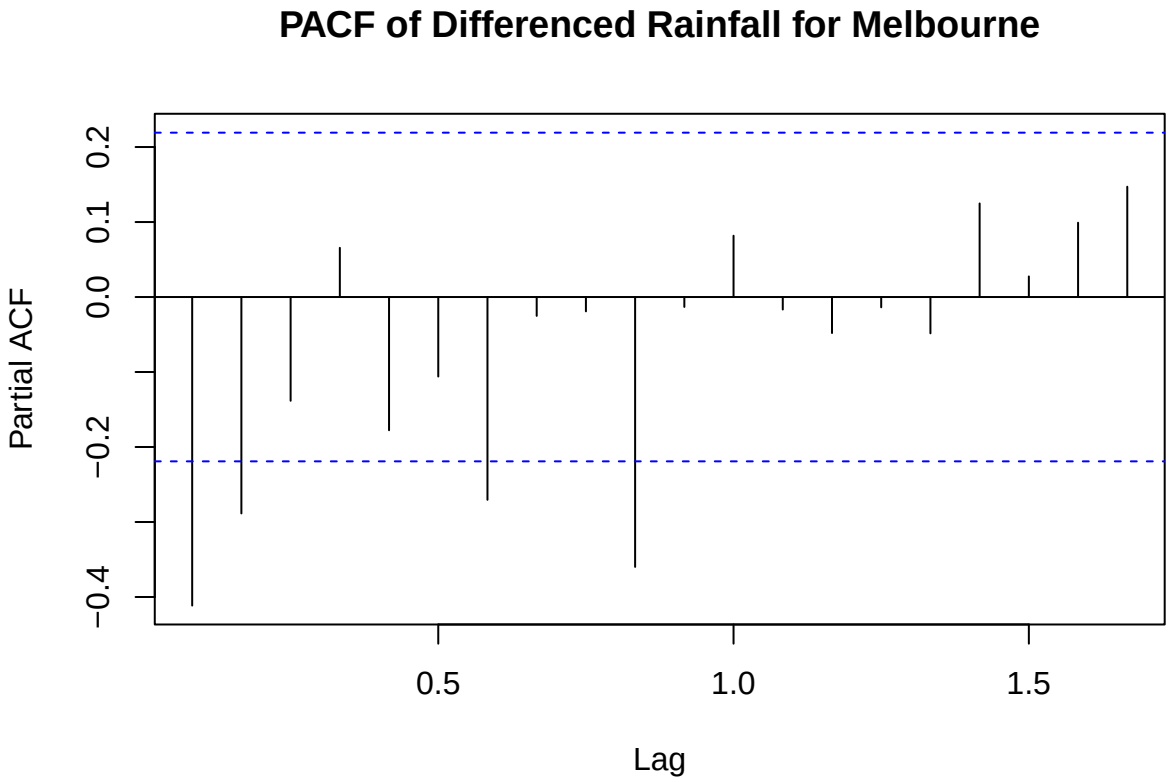

ACF of Differenced Rainfall for Perth
ACF
1.0
0.8
0.6
0.4
0.2
−0.2
0.0
0.5
1.0
1.5
Lag


PACF of Differenced Rainfall for Perth
Partial ACF
0.2
0.1
0.0
−0.1
−0.3
0.5
1.0
1.5
Lag


ACF of Differenced Rainfall for Darwin
ACF
1.0
0.8
0.6
0.4
0.2
−0.2
0.0
0.5
1.0
1.5
Lag


PACF of Differenced Rainfall for Darwin
Partial ACF
0.2
0.1
−0.1
−0.3
0.5
1.0
1.5
Lag


ACF of Differenced Rainfall for Melbourne
ACF
0.8
0.4
0.0
−0.4
0.0
0.5
1.0
1.5
Lag



ARIMA MODELLING

```r
# Ensure knitr package is loaded
library(knitr)

# Initialize an empty list to store ARIMA models for each city
city_results <- list()

# Create a data frame to store the ARIMA model details for each city
arima_summary_df <- data.frame(
  City = character(),
  ARIMA_Order = character(),
  AIC = numeric(),
  BIC = numeric(),
  Coefficients = character(),
  stringsAsFactors = FALSE
)

# Loop through the cities to apply ARIMA modeling
cities <- c("Sydney", "Perth", "Darwin", "Melbourne")

for (city in cities) {
  # Filter data for the current city
  city_train_data <- train_data |> filter(Location == city)

  # Check if data for the city exists
  if (nrow(city_train_data) == 0) {
    message(paste("No data available for", city, "- Skipping."))
```

```
    next
  }

  # Convert to tsibble and create a time series object
  city_train_tsibble <- city_train_data |>
    as_tsibble(index = year_month, key = Location)

  # Ensure Rainfall is a numeric vector and create a time series object
  rainfall_ts <- ts(city_train_tsibble$Rainfall, frequency = 12)

  # Fit an ARIMA model directly to the stationary data
  arima_model <- auto.arima(rainfall_ts)

  # Extract ARIMA order (p, d, q)
  arima_order <- paste(arima_model$arma[1], arima_model$arma[6], arima_model$arma[2], sep = ",")

  # Extract AIC and BIC
  aic_value <- arima_model$aic
  bic_value <- arima_model$bic

  # Extract coefficients and convert to string
  coeffs <- paste(names(arima_model$coef), round(arima_model$coef, 4), collapse = ", ")

  # Store the ARIMA summary in the data frame
  arima_summary_df <- rbind(arima_summary_df, data.frame(
    City = city,
    ARIMA_Order = arima_order,
    AIC = aic_value,
    BIC = bic_value,
    Coefficients = coeffs
  ))

  # Store the ARIMA model for the current city
  city_results[[city]] <- list(
    arima_model = arima_model
  )
}

# Display the ARIMA summary table using kable
kable(arima_summary_df, caption = "ARIMA Model Summary for Each City") |>
  kable_styling(
    bootstrap_options = c("striped", "hover", "condensed", "responsive"),
    full_width = FALSE,
    position = "center"
  ) |>
  column_spec(2, bold = TRUE, border_right = TRUE) |>
  column_spec(3:4, width = "10em") |>
  column_spec(5, width = "20em", extra_css = "word-wrap: break-word;")
```

Table 1: ARIMA Model Summary for Each City

City	ARIMA_Order	AIC	BIC	Coefficients
Sydney	**0,0,0**	992.7719	997.6806	intercept 97.4605

Perth	**0,0,2**	833.6316	847.9983	ma1 0.3382, ma2 0.4223, sma1 0.3371, s 0.4999, intercept 58.2675
Darwin	**1,0,0**	1044.5578	1054.1356	ar1 0.4344, sar1 0.5386, intercept 144.8
Melbourne	**2,0,0**	770.7475	785.1142	ar1 0.2589, ar2 0.2465, sma1 0.126, sma -0.4342, intercept 47.6892

Assess ARIMA Forecast Accuracy on Test Data

```
# List of cities
cities <- c("Sydney", "Perth", "Darwin", "Melbourne")

# Create a data frame to store forecast accuracy results
forecast_table <- data.frame()

for (city in cities) {
  # Filter data for the current city
  city_train_data <- train_data |> filter(Location == city)
  city_test_data <- test_data |> filter(Location == city)

  # Ensure data exists for the city
  if (nrow(city_train_data) == 0 || nrow(city_test_data) == 0) {
    message(paste("No data available for", city, "- Skipping."))
    next
  }

  # Convert training data to a time series object
  rainfall_train_ts <- ts(city_train_data$Rainfall, frequency = 12)

  # Fit an ARIMA model to the training data
  arima_model <- auto.arima(rainfall_train_ts)

  # Forecast on the test data range
  forecast_steps <- nrow(city_test_data)
  arima_forecast <- forecast(arima_model, h = forecast_steps)

  # Calculate forecast accuracy using actual test data
  test_actuals <- city_test_data$Rainfall
  forecast_accuracy <- accuracy(arima_forecast, test_actuals)

  # Extract key metrics and add to the table
  forecast_table <- rbind(
    forecast_table,
    data.frame(
      City = city,
      AIC = round(arima_model$aic, 2),
      BIC = round(arima_model$bic, 2),
      RMSE = round(forecast_accuracy["Test set", "RMSE"], 3),
      MAE = round(forecast_accuracy["Test set", "MAE"], 3),
      MAPE = round(forecast_accuracy["Test set", "MAPE"], 2)
    )
  )
}
```

Table 2: Forecast Accuracy Metrics for Each City

City	AIC	BIC	RMSE	MAE	
Sydney	992.77	997.68	89.026	68.760	
Perth	833.63	848.00	31.379	24.475	
Darwin	1044.56	1054.14	163.666	119.949	
Melbourne	770.75	785.11	34.365	31.326	

```
# Create a styled table with wider columns
forecast_table %>%
  kbl(
    col.names = c("City", "AIC", "BIC", "RMSE", "MAE", "MAPE (%)"),
    caption = "Forecast Accuracy Metrics for Each City",
    align = "lccccr"
  ) %>%
  kable_styling(
    full_width = FALSE,
    bootstrap_options = c("striped", "hover", "condensed", "responsive"),
    position = "center"
  ) %>%
  column_spec(2, width = "3cm") %>% # Widen AIC column
  column_spec(3, width = "3cm") %>% # Widen BIC column
  column_spec(4:6, width = "4cm")  # Widen RMSE, MAE, columns
```

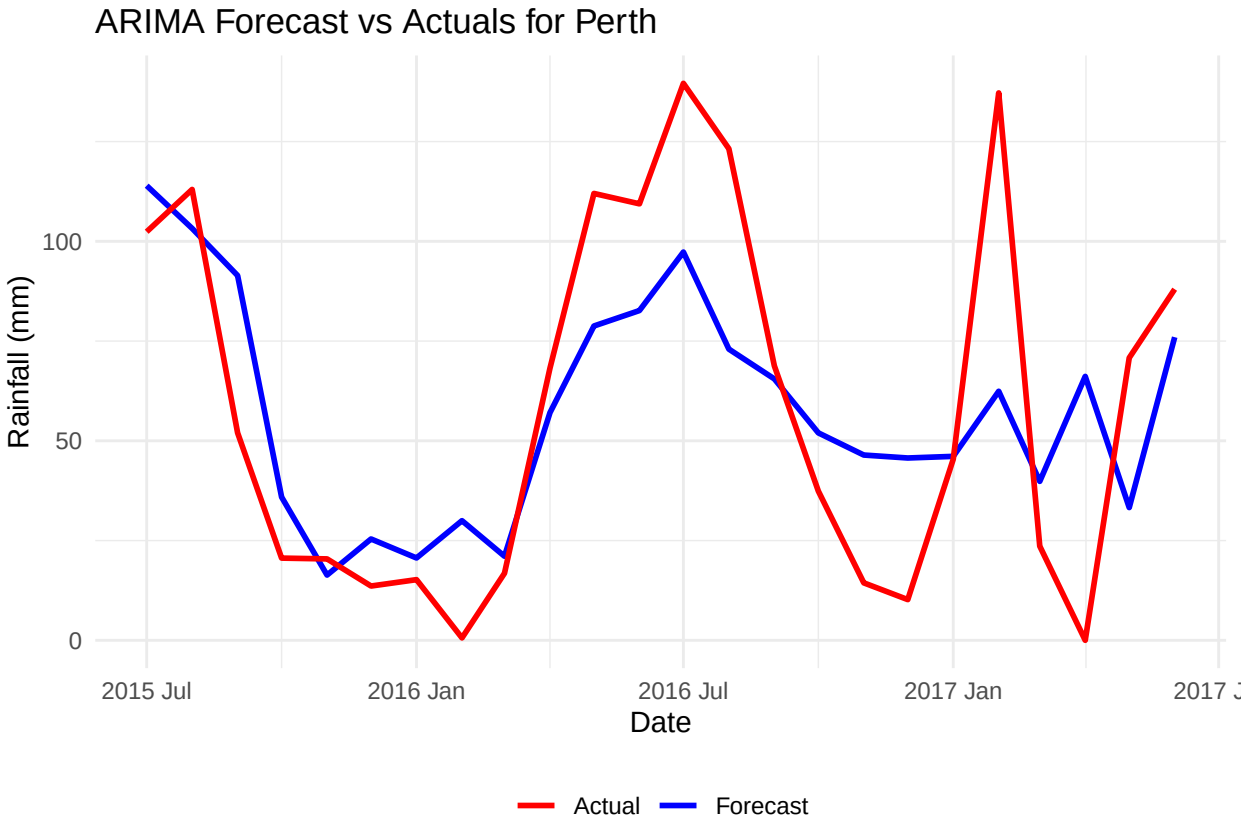
Perth had best ARIMA results, plot forecasted and actual values

```
# Filter data for Perth
city_train_data <- train_data |> filter(Location == "Perth")
city_test_data <- test_data |> filter(Location == "Perth")

# Convert training data to a time series object
rainfall_train_ts <- ts(city_train_data$Rainfall, frequency = 12)

# Fit an ARIMA model to the training data
arima_model <- auto.arima(rainfall_train_ts)

# Forecast on the test data range
forecast_steps <- nrow(city_test_data)
arima_forecast <- forecast(arima_model, h = forecast_steps)

# Create a data frame for the forecast and actual values
forecast_df <- data.frame(
  Date = city_test_data$year_month,
  Forecast = arima_forecast$mean,
  Actual = city_test_data$Rainfall
)

# Plot the forecasted vs actual values using ggplot2
ggplot(forecast_df, aes(x = Date)) +
  geom_line(aes(y = Forecast, color = "Forecast"), size = 1) +    # Forecast line in blue
  geom_line(aes(y = Actual, color = "Actual"), size = 1) +         # Actual data line in red
```

```
  labs(title = "ARIMA Forecast vs Actuals for Perth",
       x = "Date", y = "Rainfall (mm)") +
  scale_color_manual(values = c("Forecast" = "blue", "Actual" = "red")) + # Set colors
  theme_minimal() +
  theme(legend.title = element_blank(), legend.position = "bottom") # Position legend
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## Don't know how to automatically pick scale for object of type <ts>. Defaulting
## to continuous.
```



Modelling

XGBoost

```
# Loop through each city to train and assess the model
# List of cities
cities <- c("Sydney", "Perth", "Darwin", "Melbourne")

# Create a data frame to store forecast accuracy results
forecast_results <- data.frame()

# Create a list to store predictions for plotting later
predictions_list <- list()
```

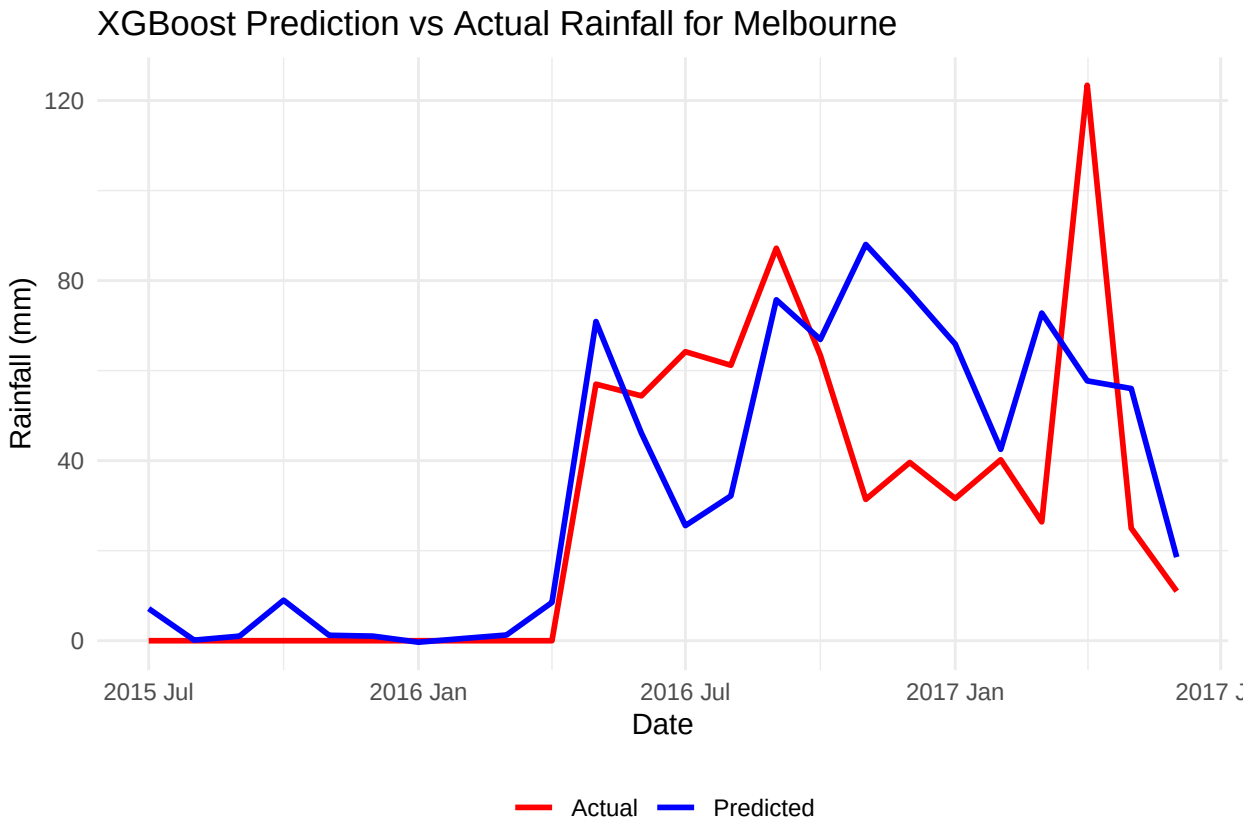
```r
# Loop through each city to train and assess the model
for (city in cities) {

  # Filter data for the current city
  city_train_data <- train_data |> filter(Location == city)
  city_test_data <- test_data |> filter(Location == city)

  # Ensure data exists for the city
  if (nrow(city_train_data) == 0 || nrow(city_test_data) == 0) {
    message(paste("No data available for", city, "- Skipping."))
    next
  }

  # Prepare the features (use all columns except 'Rainfall' for features)
  train_features <- city_train_data %>%
    select(-Location, -Rainfall, -year_month) %>%
    as.matrix()  # Convert to matrix format for XGBoost

  test_features <- city_test_data %>%
    select(-Location, -Rainfall, -year_month) %>%
    as.matrix()  # Convert to matrix format for XGBoost

  # Prepare the target variable (Rainfall)
  train_target <- city_train_data$Rainfall
  test_target <- city_test_data$Rainfall

  # Convert data to xgboost-friendly format
  dtrain <- xgb.DMatrix(data = train_features, label = train_target)
  dtest <- xgb.DMatrix(data = test_features, label = test_target)

  # Set parameters for XGBoost
  params <- list(
    booster = "gbtree",  # Tree-based model
    objective = "reg:squarederror",  # Regression task
    eval_metric = "rmse",  # Root mean square error as the evaluation metric
    max_depth = 6,  # Maximum depth of the trees
    eta = 0.3,  # Learning rate
    nthread = 2  # Number of threads
  )

  # Train the model
  xgboost_model <- xgb.train(
    params = params,
    data = dtrain,
    nrounds = 100,  # Number of boosting rounds
    watchlist = list(train = dtrain, test = dtest),  # Watch the performance on both train and test set
    verbose = 0  # Suppress progress during training
  )

  # Make predictions on the test set
  xgboost_predictions <- predict(xgboost_model, newdata = dtest)

  # Calculate RMSE, MAE, and MAPE for the current city
```

```
  rmse_value <- sqrt(mean((test_target - xgboost_predictions)^2))
  mae_value <- mean(abs(test_target - xgboost_predictions))
  mape_value <- mean(abs((test_target - xgboost_predictions) / test_target)) * 100

  # Store the results in the data frame
  forecast_results <- rbind(
    forecast_results,
    data.frame(
      City = city,
      RMSE = round(rmse_value, 3),
      MAE = round(mae_value, 3),
      MAPE = round(mape_value, 2)
    )
  )

  # Store the predictions for plotting later
  predictions_list[[city]] <- list(
    actual = test_target,
    predicted = xgboost_predictions,
    date = city_test_data$year_month
  )
}

# Print the forecast accuracy results
print(forecast_results)
```

```
##        City   RMSE    MAE  MAPE
## 1    Sydney 83.202 52.196 42.83
## 2     Perth 33.023 24.536   Inf
## 3    Darwin 88.583 60.525   Inf
## 4 Melbourne 26.023 17.362   Inf
```

Melbourne had best results on XGBoost, plot forecasted and actual values

```
# Extract the predictions for Melbourne from predictions_list
melbourne_predictions <- predictions_list[["Melbourne"]]

# Create the plot
ggplot() +
  geom_line(aes(x = melbourne_predictions$date, y = melbourne_predictions$actual, color = "Actual"), si
  geom_line(aes(x = melbourne_predictions$date, y = melbourne_predictions$predicted, color = "Predicted
  labs(title = "XGBoost Prediction vs Actual Rainfall for Melbourne",
       x = "Date", y = "Rainfall (mm)") +
  scale_color_manual(values = c("Actual" = "red", "Predicted" = "blue")) +
  theme_minimal() +
  theme(legend.title = element_blank(), legend.position = "bottom")
```



Linear Regression

```
# Loop through each city to train and assess the model
# List of cities
cities <- c("Sydney", "Perth", "Darwin", "Melbourne")

# Create a data frame to store forecast accuracy results
forecast_results <- data.frame()

# Create a list to store predictions for plotting later
predictions_list <- list()

# Loop through each city to train and assess the model
for (city in cities) {

  # Filter data for the current city
  city_train_data <- train_data |> filter(Location == city)
  city_test_data <- test_data |> filter(Location == city)

  # Ensure data exists for the city
  if (nrow(city_train_data) == 0 || nrow(city_test_data) == 0) {
    message(paste("No data available for", city, "- Skipping."))
    next
  }

  # Prepare the features (use all columns except 'Rainfall' for features)
  train_features <- city_train_data %>%
```

```
    select(-Location, -Rainfall, -year_month)  # Remove 'Location' and 'Rainfall' for features

  test_features <- city_test_data %>%
    select(-Location, -Rainfall, -year_month)  # Remove 'Location' and 'Rainfall' for features

  # Prepare the target variable (Rainfall)
  train_target <- city_train_data$Rainfall
  test_target <- city_test_data$Rainfall

  # Train the Linear Regression model
  linear_model <- lm(Rainfall ~ ., data = city_train_data %>% select(-Location, -year_month))

  # Make predictions on the test set
  linear_predictions <- predict(linear_model, newdata = test_features)

  # Calculate RMSE, MAE, and MAPE for the current city
  rmse_value <- sqrt(mean((test_target - linear_predictions)^2))
  mae_value <- mean(abs(test_target - linear_predictions))
  mape_value <- mean(abs((test_target - linear_predictions) / test_target)) * 100

  # Store the results in the data frame
  forecast_results <- rbind(
    forecast_results,
    data.frame(
      City = city,
      RMSE = round(rmse_value, 3),
      MAE = round(mae_value, 3),
      MAPE = round(mape_value, 2)
    )
  )

  # Store the predictions for plotting later
  predictions_list[[city]] <- list(
    actual = test_target,
    predicted = linear_predictions,
    date = city_test_data$year_month
  )
}
```

```
## Warning in predict.lm(linear_model, newdata = test_features): prediction from
## rank-deficient fit; attr(*, "non-estim") has doubtful cases
```

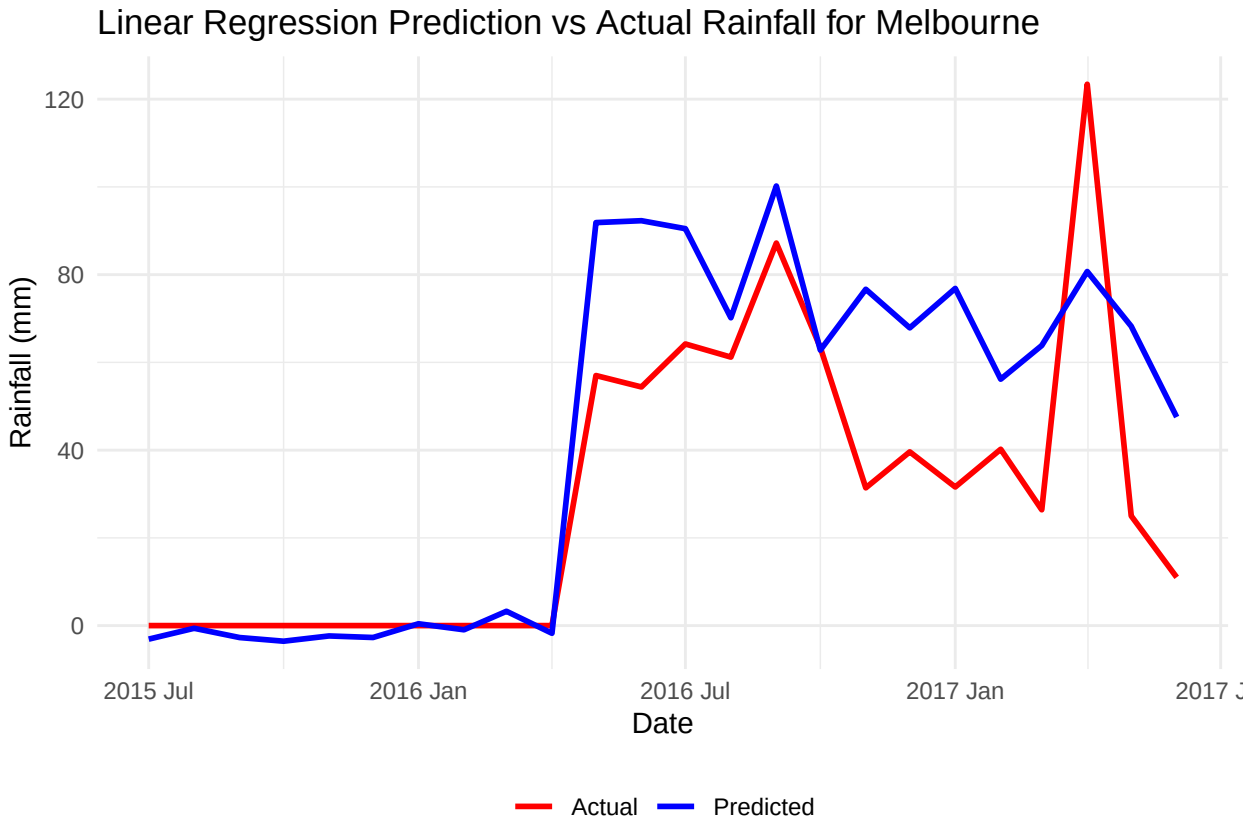
```
# Print the forecast accuracy results
print(forecast_results)
```

```
##        City    RMSE    MAE  MAPE
## 1    Sydney  78.779 55.604 61.11
## 2     Perth  28.942 21.967   Inf
## 3    Darwin 113.066 91.307   Inf
## 4 Melbourne  25.190 18.238   Inf
```

Linear Regression: Melbourne had best result

plot forecasted and actual values

```
# Extract the predictions for Melbourne from predictions_list
melbourne_predictions <- predictions_list[["Melbourne"]]

# Create the plot
ggplot() +
  geom_line(aes(x = melbourne_predictions$date, y = melbourne_predictions$actual, color = "Actual"), si
  geom_line(aes(x = melbourne_predictions$date, y = melbourne_predictions$predicted, color = "Predicted
  labs(title = "Linear Regression Prediction vs Actual Rainfall for Melbourne",
       x = "Date", y = "Rainfall (mm)") +
  scale_color_manual(values = c("Actual" = "red", "Predicted" = "blue")) +
  theme_minimal() +
  theme(legend.title = element_blank(), legend.position = "bottom")
```



Plot Table Summary of Models