Data 624 - P1 (Fall 2024)

Khyati Naik

Part A

ATM Forecast, ATM624Data.xlsx - In part A, I want you to forecast how much cash is taken out of 4 different ATM machines for May 2010. The data is given in a single file. The variable 'Cash' is provided in hundreds of dollars, other than that it is straight forward. I am being somewhat ambiguous on purpose to make this have a little more business feeling. Explain and demonstrate your process, techniques used and not used, and your actual forecast. I am giving you data via an excel file, please provide your written report on your findings, visuals, discussion and your R code via an RPubs link along with the actual.rmd file Also please submit the forecast which you will put in an Excel readable file.

```
library(readxl)
library(tidyverse)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3

## Warning: package 'tidyr' was built under R version 4.3.2

## Warning: package 'readr' was built under R version 4.3.2

## Warning: package 'dplyr' was built under R version 4.3.2

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(fpp3)
```

```
## Warning: package 'fpp3' was built under R version 4.3.3

## -- Attaching packages ---------------------------------------- fpp3 1.0.0 --
## v tsibble     1.1.3     v fable       0.3.4
## v tsibbledata 0.4.1     v fabletools  0.4.2
## v feasts      0.3.2
```

```
## Warning: package 'tsibble' was built under R version 4.3.2

## Warning: package 'tsibbledata' was built under R version 4.3.3

## Warning: package 'feasts' was built under R version 4.3.3

## Warning: package 'fabletools' was built under R version 4.3.3

## Warning: package 'fable' was built under R version 4.3.3

## -- Conflicts ------------------------------------------------ fpp3_conflicts --
## x lubridate::date()    masks base::date()
## x dplyr::filter()      masks stats::filter()
## x tsibble::intersect() masks base::intersect()
## x tsibble::interval()  masks lubridate::interval()
## x dplyr::lag()         masks stats::lag()
## x tsibble::setdiff()   masks base::setdiff()
## x tsibble::union()     masks base::union()
```

```
library(fable)
library(purrr)
library(zoo)
```

```
## Warning: package 'zoo' was built under R version 4.3.2

##
## Attaching package: 'zoo'
##
## The following object is masked from 'package:tsibble':
##
##     index
##
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(lubridate)
```

```
# Step 1: Download and Load Excel Data from GitHub
file_url <- "https://github.com/Naik-Khyati/data_624/raw/main/p1/ATM624Data.xlsx"
temp_path <- tempfile(fileext = ".xlsx")
download.file(file_url, temp_path, mode = "wb")

# Step 2: Load the Excel data into a DataFrame
ATMData <- read_excel(temp_path)

# Step 3: Display the first few rows of the dataset
head(ATMData)
```

```
## # A tibble: 6 x 3
##    DATE ATM    Cash
##   <dbl> <chr> <dbl>
## 1 39934 ATM1     96
## 2 39934 ATM2    107
## 3 39935 ATM1     82
## 4 39935 ATM2     89
## 5 39936 ATM1     85
## 6 39936 ATM2     90
```

```
# Step 4: Convert numeric 'DATE' column into proper Date format (YYYY-MM-DD)
ATMData$TransactionDate <- as.Date(ATMData$DATE, origin = "1899-12-30")

# Step 5: Generate a summary of the dataset
summary(ATMData)
```

```
##       DATE           ATM                 Cash          TransactionDate
##  Min.   :39934   Length:1474        Min.   :    0.0   Min.   :2009-05-01
##  1st Qu.:40026   Class :character   1st Qu.:    0.5   1st Qu.:2009-08-01
##  Median :40118   Mode  :character   Median :   73.0   Median :2009-11-01
##  Mean   :40118                      Mean   :  155.6   Mean   :2009-10-31
##  3rd Qu.:40210                      3rd Qu.:  114.0   3rd Qu.:2010-02-01
##  Max.   :40312                      Max.   :10919.8   Max.   :2010-05-14
##                                     NA's   :19
```

```
# Step 6: Handle Missing Data

# Identify rows where 'Cash' column contains NA values
na_rows <- which(is.na(ATMData$Cash))
ATMData[na_rows, ]
```

```
## # A tibble: 19 x 4
##     DATE ATM    Cash TransactionDate
##    <dbl> <chr> <dbl> <date>
##  1 39977 ATM1     NA 2009-06-13
##  2 39980 ATM1     NA 2009-06-16
##  3 39982 ATM2     NA 2009-06-18
##  4 39986 ATM1     NA 2009-06-22
##  5 39988 ATM2     NA 2009-06-24
##  6 40299 <NA>     NA 2010-05-01
##  7 40300 <NA>     NA 2010-05-02
##  8 40301 <NA>     NA 2010-05-03
##  9 40302 <NA>     NA 2010-05-04
## 10 40303 <NA>     NA 2010-05-05
## 11 40304 <NA>     NA 2010-05-06
## 12 40305 <NA>     NA 2010-05-07
## 13 40306 <NA>     NA 2010-05-08
## 14 40307 <NA>     NA 2010-05-09
## 15 40308 <NA>     NA 2010-05-10
## 16 40309 <NA>     NA 2010-05-11
## 17 40310 <NA>     NA 2010-05-12
## 18 40311 <NA>     NA 2010-05-13
## 19 40312 <NA>     NA 2010-05-14
```

```
# Explanation:
# There are 19 rows with missing values. Out of those, 14 have no ATM or Cash data.
# Since we lack sufficient data for these rows, we'll exclude them entirely.
# The remaining 5 rows with missing 'Cash' values but valid ATM data will be imputed using an ARIMA mod
# which accounts for dependencies and patterns in the time series data.

# Step 7: Remove rows where the 'ATM' column has missing values
ATMData <- ATMData[!is.na(ATMData$ATM), ]

# Sort the data by ATM column
ATMData <- ATMData %>%
  arrange(ATM)

# Step 8: Recheck rows with missing 'Cash' values after removal
ATMData[na_rows, ]
```

```
## # A tibble: 19 x 4
##     DATE ATM    Cash TransactionDate
##    <dbl> <chr> <dbl> <date>
##  1 40020 ATM1    108 2009-07-26
##  2 40026 ATM1     90 2009-08-01
##  3 40031 ATM1     13 2009-08-06
##  4 40038 ATM1     34 2009-08-13
##  5 40043 ATM1    132 2009-08-18
##  6 39934 ATM3      0 2009-05-01
##  7 39935 ATM3      0 2009-05-02
##  8 39936 ATM3      0 2009-05-03
##  9 39937 ATM3      0 2009-05-04
## 10 39938 ATM3      0 2009-05-05
## 11 39939 ATM3      0 2009-05-06
## 12 39940 ATM3      0 2009-05-07
## 13 39941 ATM3      0 2009-05-08
## 14 39942 ATM3      0 2009-05-09
## 15 39943 ATM3      0 2009-05-10
## 16 39944 ATM3      0 2009-05-11
## 17 39945 ATM3      0 2009-05-12
## 18 39946 ATM3      0 2009-05-13
## 19 39947 ATM3      0 2009-05-14
```

```
# Display the indices of rows with missing 'Cash' values
missing_indices <- which(is.na(ATMData$Cash))
missing_indices
```

```
## [1]  44  47  53 414 420
```

```
# Step 9: Convert the data into a tsibble for time series analysis
atm_series <- as_tsibble(ATMData, index = TransactionDate, key = ATM)

# Step 10: Interpolating missing 'Cash' values using ARIMA model
atm_series <- atm_series %>%
  model(ARIMA(Cash)) %>%
  interpolate(atm_series)
```

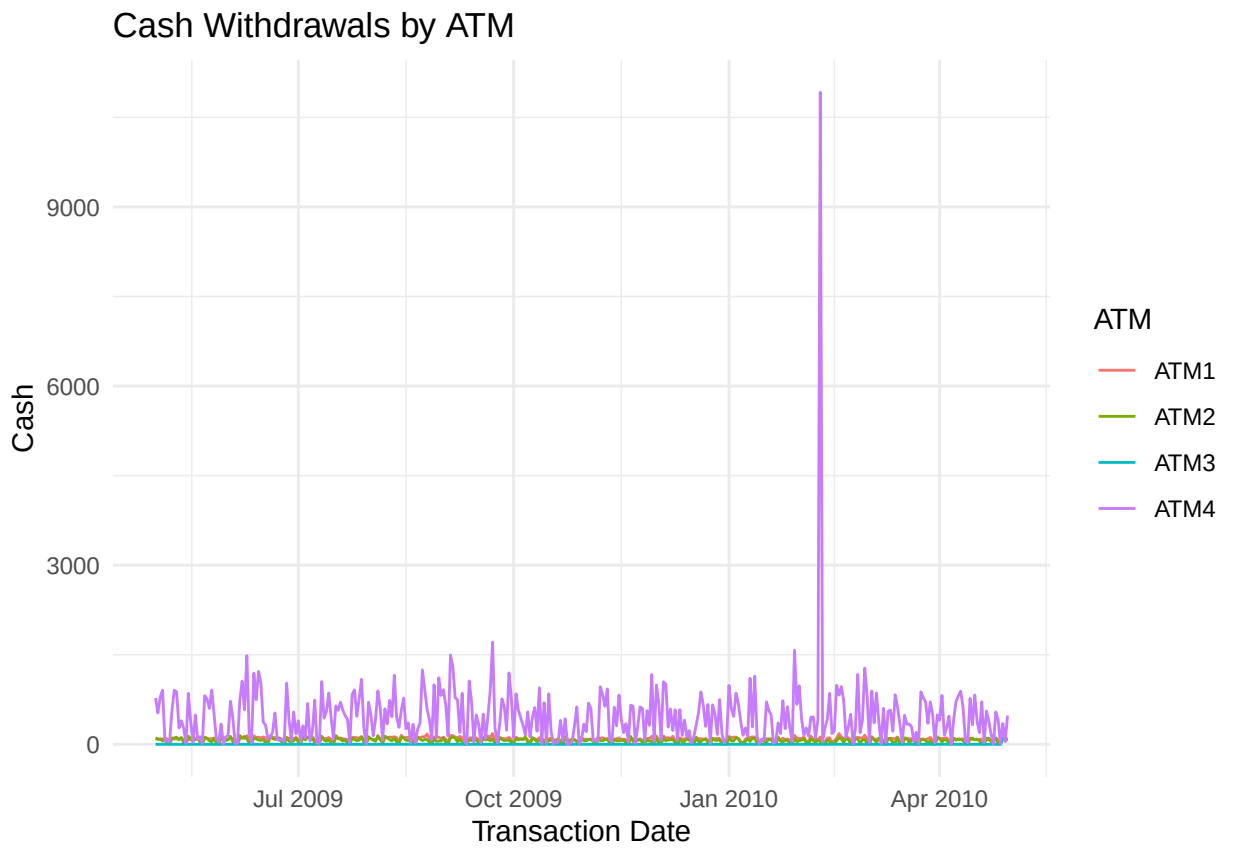
```r
# Verify that missing 'Cash' values are filled
atm_series[missing_indices, ]
```

```
## # A tsibble: 5 x 3 [1D]
## # Key:       ATM [2]
##   ATM   TransactionDate   Cash
##   <chr> <date>           <dbl>
## 1 ATM1  2009-06-13      111.
## 2 ATM1  2009-06-16      124.
## 3 ATM1  2009-06-22      100.
## 4 ATM2  2009-06-18        9.42
## 5 ATM2  2009-06-24       29.4
```

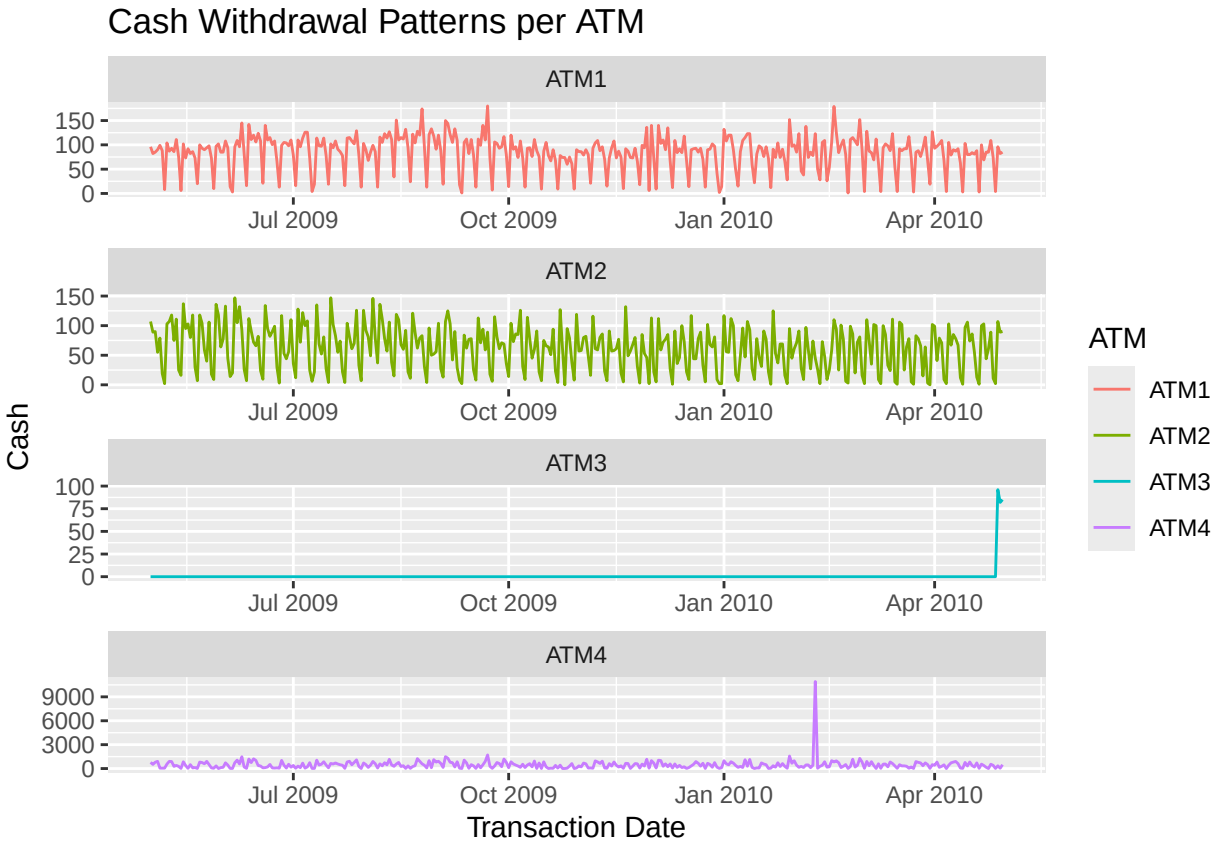
```r
# Step 11: Time Series Analysis and Visualization

# Prevent scientific notation for better readability
options(scipen = 999)

# Summary statistics of 'Cash' withdrawals grouped by ATM
aggregate(Cash ~ ATM, data = atm_series, summary)
```

```
##    ATM     Cash.Min.  Cash.1st Qu.   Cash.Median     Cash.Mean  Cash.3rd Qu.
## 1 ATM1     1.0000000    73.0000000    91.0000000    84.1156368   108.0000000
## 2 ATM2     0.0000000    25.0000000    66.0000000    62.3420282    93.0000000
## 3 ATM3     0.0000000     0.0000000     0.0000000     0.7205479     0.0000000
## 4 ATM4     1.5632595   124.3344146   403.8393360   474.0433449   704.5070416
##       Cash.Max.
## 1   180.0000000
## 2   147.0000000
## 3    96.0000000
## 4 10919.7616377
```

```r
# Plot a time series graph showing 'Cash' withdrawals over time for each ATM
ggplot(atm_series, aes(x = TransactionDate, y = Cash, color = ATM)) +
  geom_line() +
  labs(title = "Cash Withdrawals by ATM", x = "Transaction Date", y = "Cash") +
  theme_minimal()
```

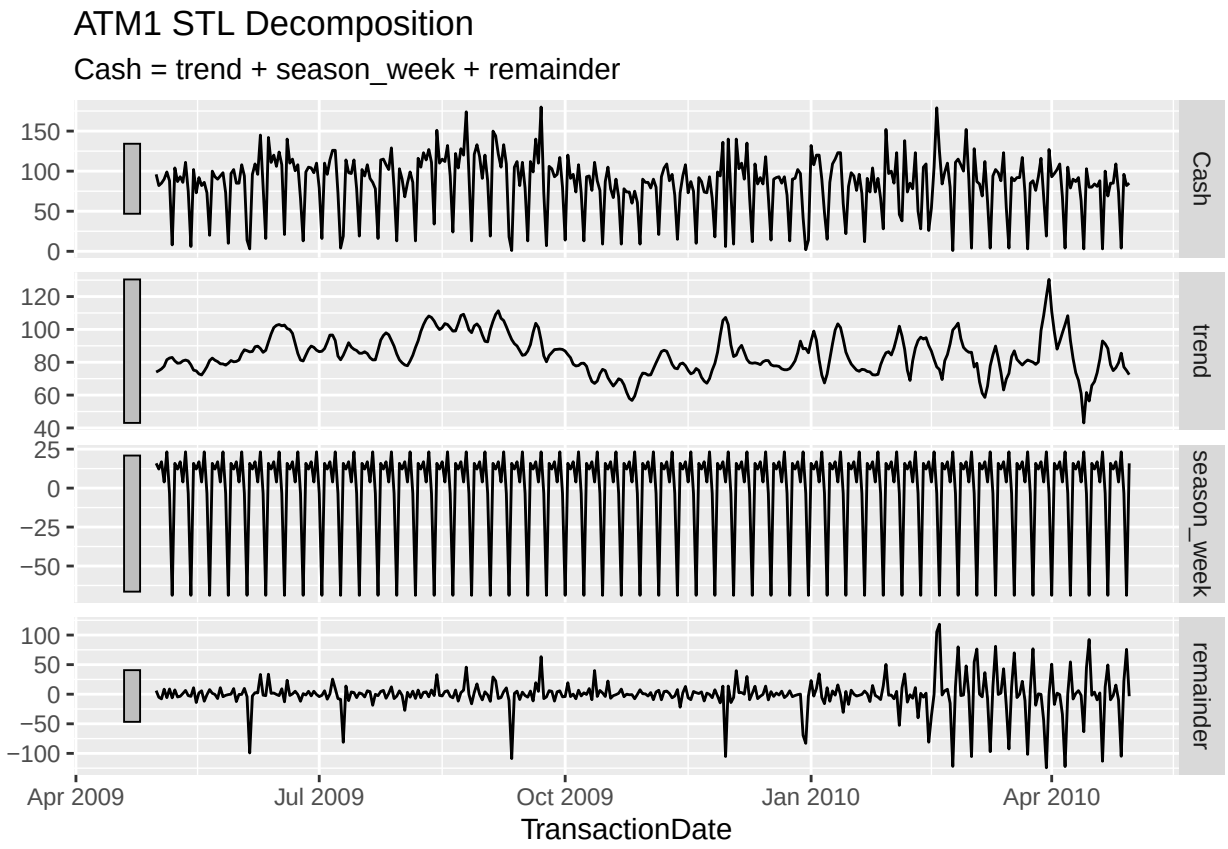


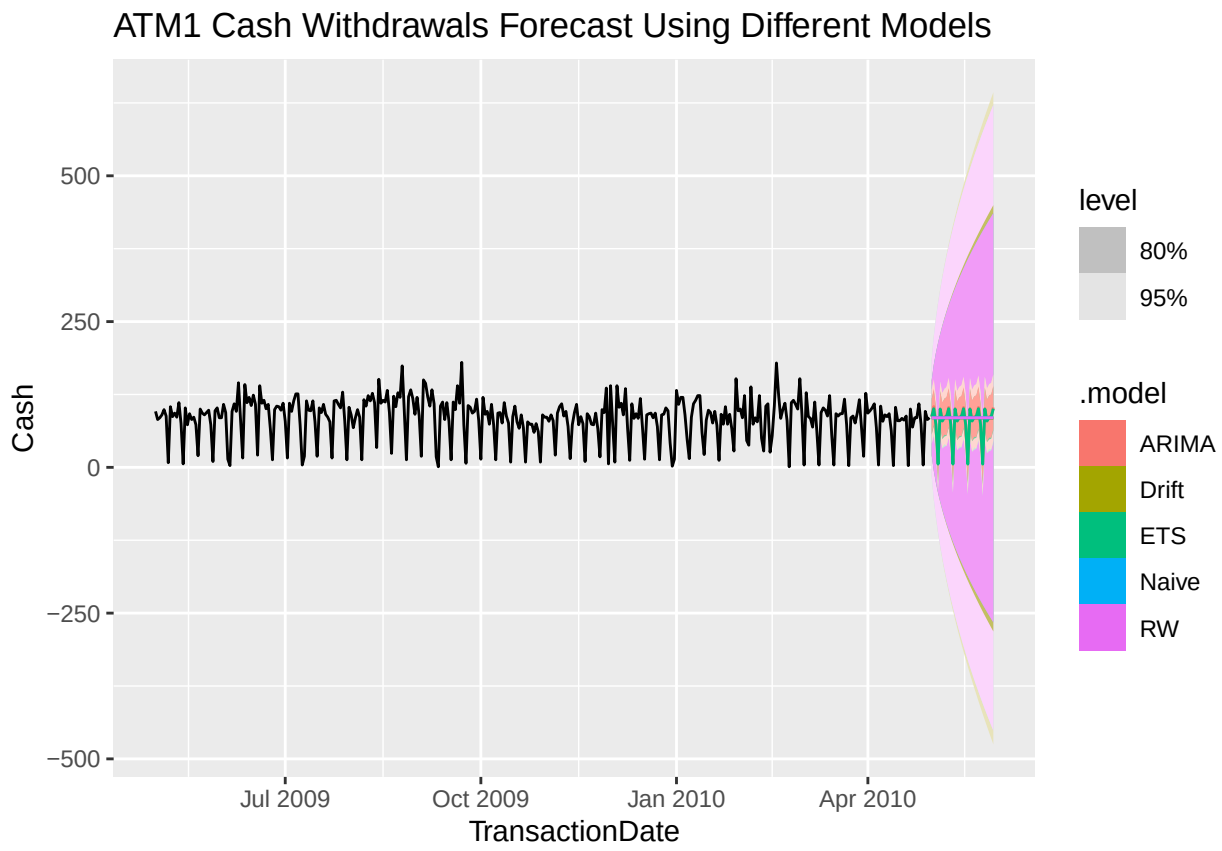

```r
# Faceted time series plot, one for each ATM
atm_series %>%
  autoplot(Cash) +
  facet_wrap(~ATM, scales = "free", nrow = 4) +
  labs(title = "Cash Withdrawal Patterns per ATM", x = "Transaction Date", y = "Cash")
```



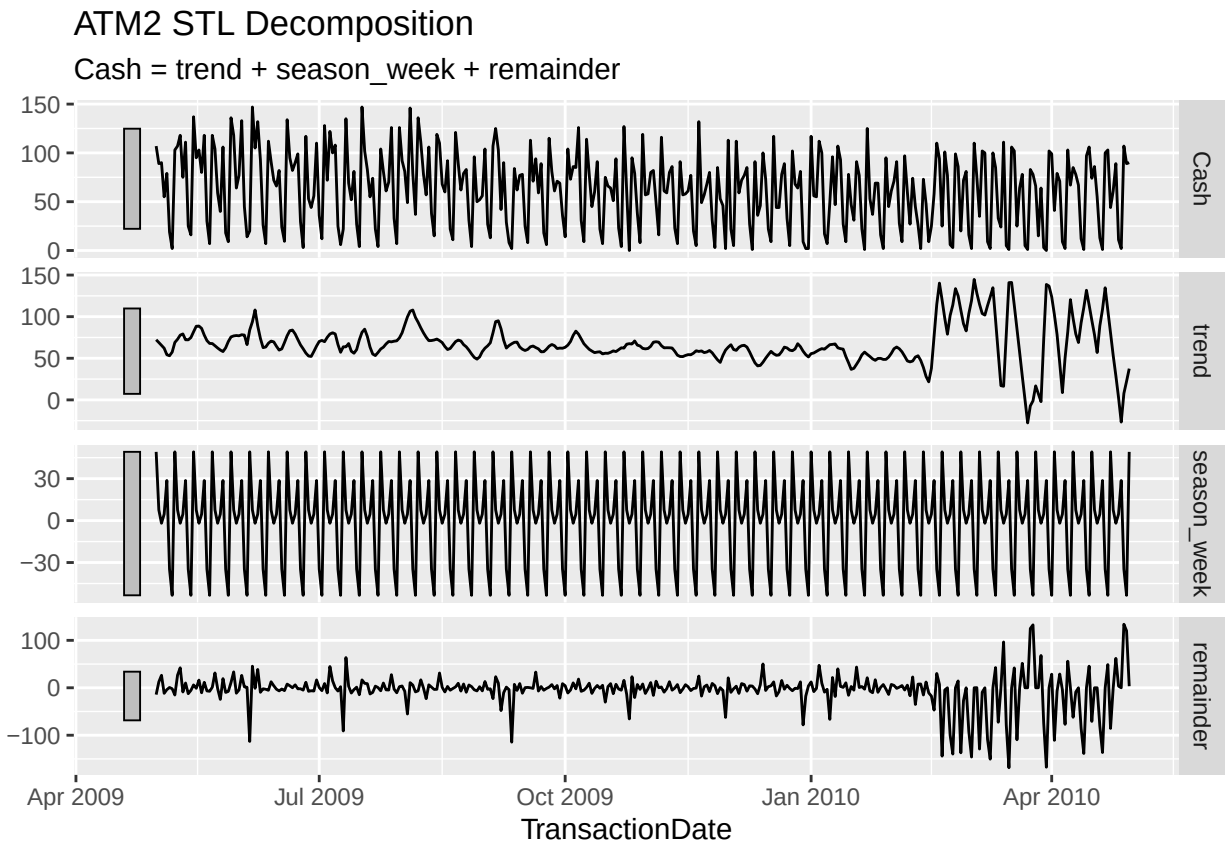

```r
# Function to model and forecast cash withdrawals for a given ATM
atm_forecast <- function(atm_data, atm_name, forecast_horizon = 30) {
  # Filter data for the specific ATM
  atm_ts_filtered <- atm_data %>%
    filter(ATM == atm_name)

  # STL decomposition
  stl_decomposition <- atm_ts_filtered %>%
    model(STL(Cash ~ trend(window = 7) + season(window = "periodic"), robust = TRUE)) %>%
    components()

  # Plot STL decomposition
  stl_plot <- stl_decomposition %>%
    autoplot() +
    labs(title = paste(atm_name, "STL Decomposition"))

  print(stl_plot)

  # Modeling using different time series methods
  model_fits <- atm_ts_filtered %>% model(
    RW = RW(Cash),
    ETS = ETS(Cash),
    Naive = NAIVE(Cash),
    Drift = NAIVE(Cash ~ drift()),
    ARIMA = ARIMA(Cash)
  )
```

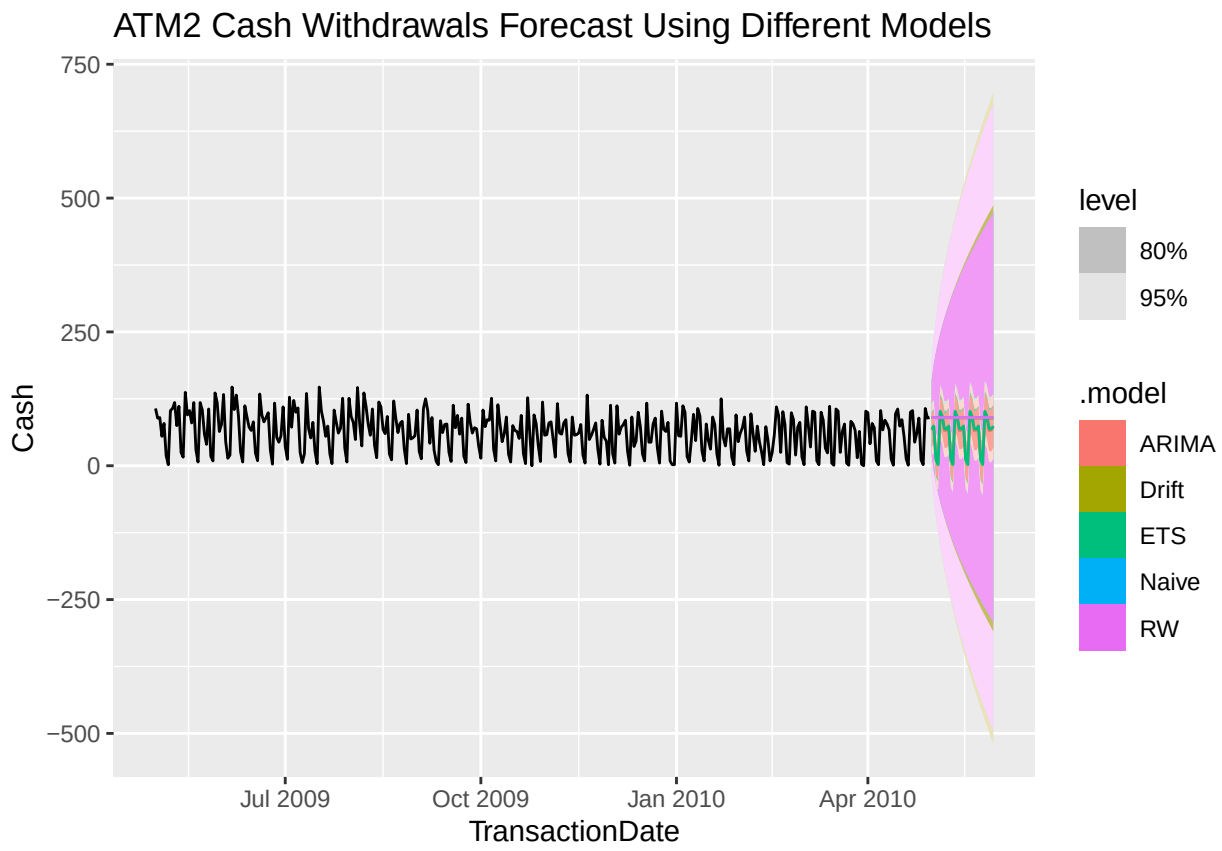
```r
  # Forecast for the specified horizon
  forecast_results <- model_fits %>% forecast(h = forecast_horizon)

  # Plot forecasts
  forecast_plot <- forecast_results %>%
    autoplot(atm_ts_filtered) +
    labs(title = paste(atm_name, "Cash Withdrawals Forecast Using Different Models"))

  print(forecast_plot)

  # Accuracy metrics for each model
  model_list <- list(
    ARIMA = atm_ts_filtered %>% model(ARIMA(Cash)),
    ETS = atm_ts_filtered %>% model(ETS(Cash)),
    Naive = atm_ts_filtered %>% model(NAIVE(Cash))
  )

  accuracy_metrics <- map_df(model_list, accuracy, .id = "model")

  # Clean accuracy table
  accuracy_table <- accuracy_metrics %>% select(-.type, -.model, -ME)
  print(accuracy_table)

  # Total forecast for 30 days
  total_forecast <- sum(forecast_results$.mean) * 100

  return(c(atm_name, total_forecast))
}

# Example usage of the function for four ATMs
forecast_ATMs <- function(atm_data) {
  atm1_results <- atm_forecast(atm_data, "ATM1")
  atm2_results <- atm_forecast(atm_data, "ATM2")
  atm3_results <- atm_forecast(atm_data, "ATM3")
  atm4_results <- atm_forecast(atm_data, "ATM4")

  # Combine results into a data frame
  forecast_df <- data.frame(matrix(c(atm1_results, atm2_results, atm3_results, atm4_results), nrow = 4,
  colnames(forecast_df) <- c("ATM", "Total Dollars")

  print(forecast_df)
}

# Calling the function
forecast_ATMs(atm_series)
```


ATM1 STL Decomposition
Cash = trend + season_week + remainder
Cash
trend
season_week
remainder
150
100
50
0
120
100
80
60
40
25
0
−25
−50
100
50
0
−50
−100
Apr 2009
Jul 2009
Oct 2009
Jan 2010
Apr 2010
TransactionDate

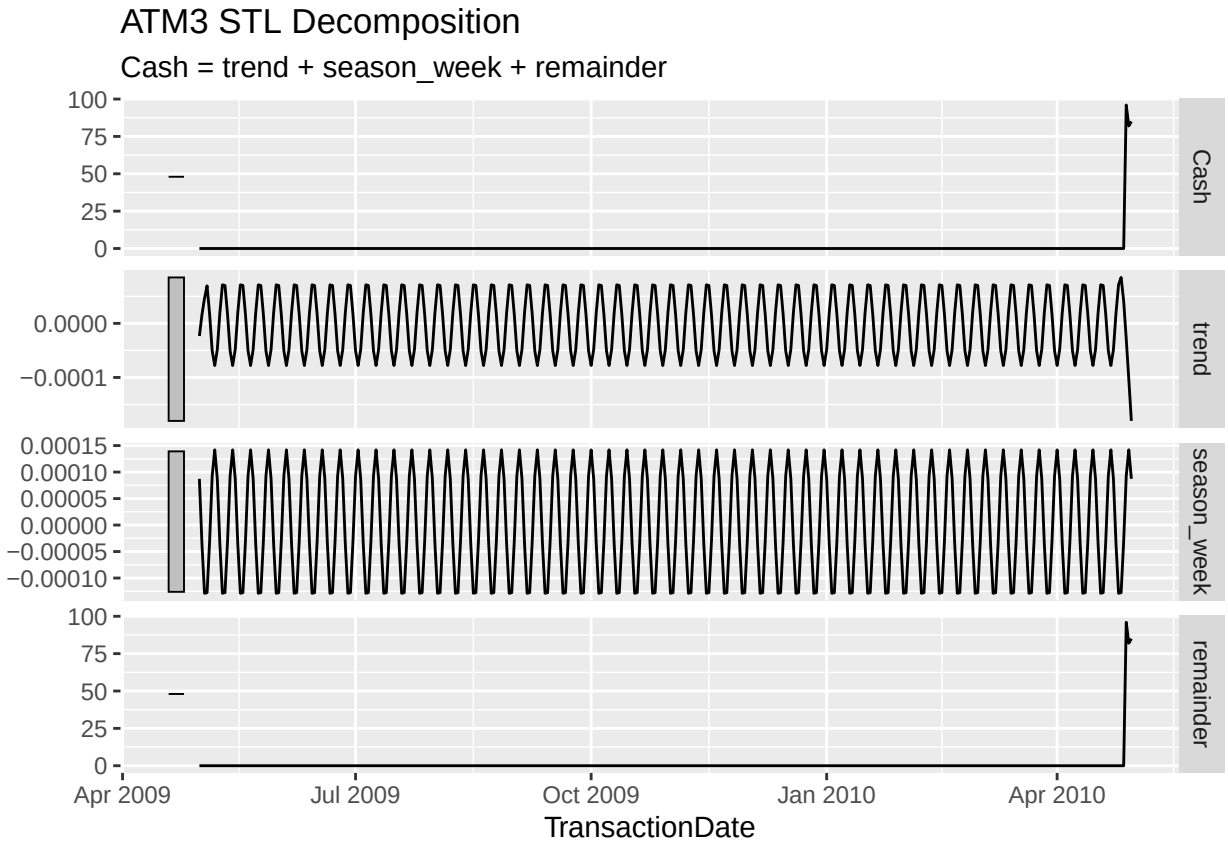


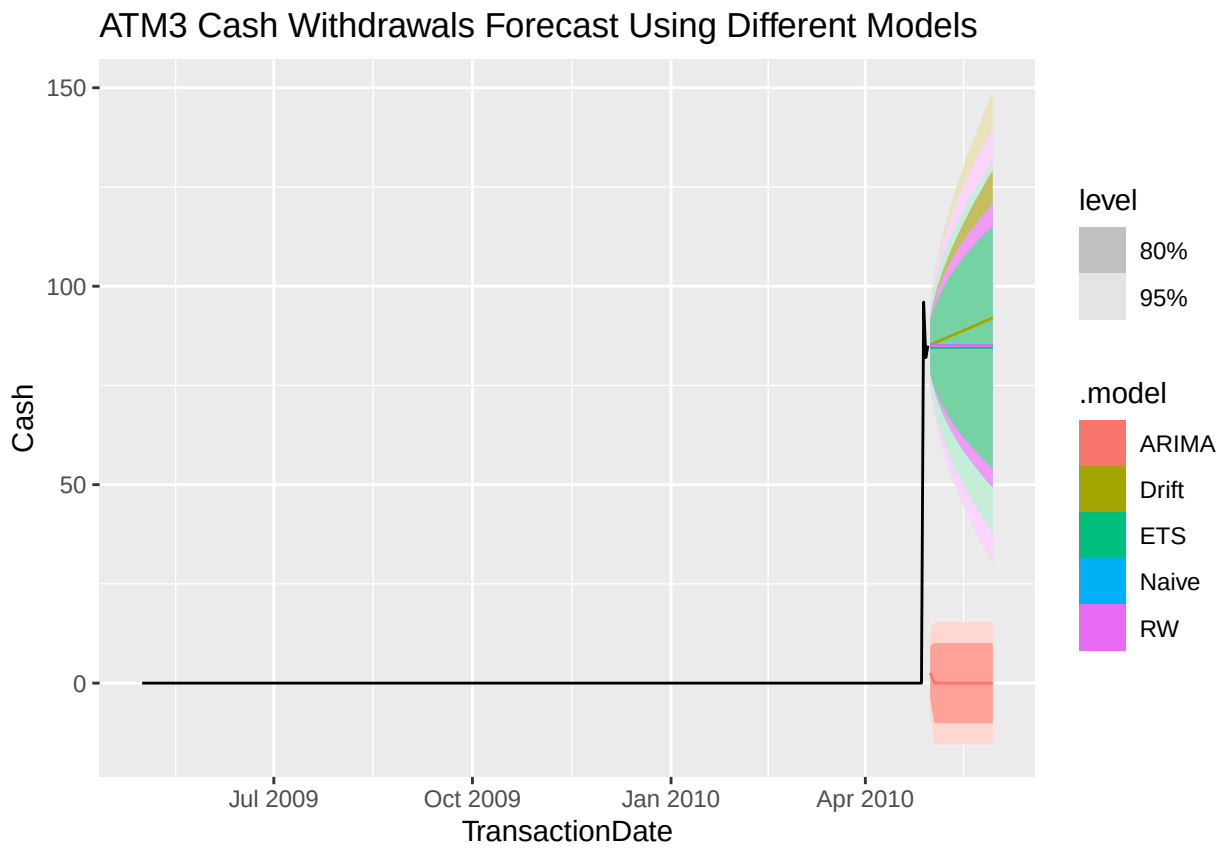

```
## # A tibble: 3 x 9
##   model ATM    RMSE   MAE   MPE  MAPE  MASE RMSSE     ACF1
##   <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>    <dbl>
## 1 ARIMA ATM1   23.2  14.4 -102.  117. 0.822 0.840 -0.00838
## 2 ETS   ATM1   23.7  15.0 -107.  122. 0.853 0.859  0.142
## 3 Naive ATM1   50.0  37.4 -132.  167. 2.13  1.81  -0.358
```


ATM2 STL Decomposition
Cash = trend + season_week + remainder
Cash
trend
season_week
remainder
150
100
50
0
30
−30
−100
Apr 2009
Jul 2009
Oct 2009
Jan 2010
Apr 2010
TransactionDate

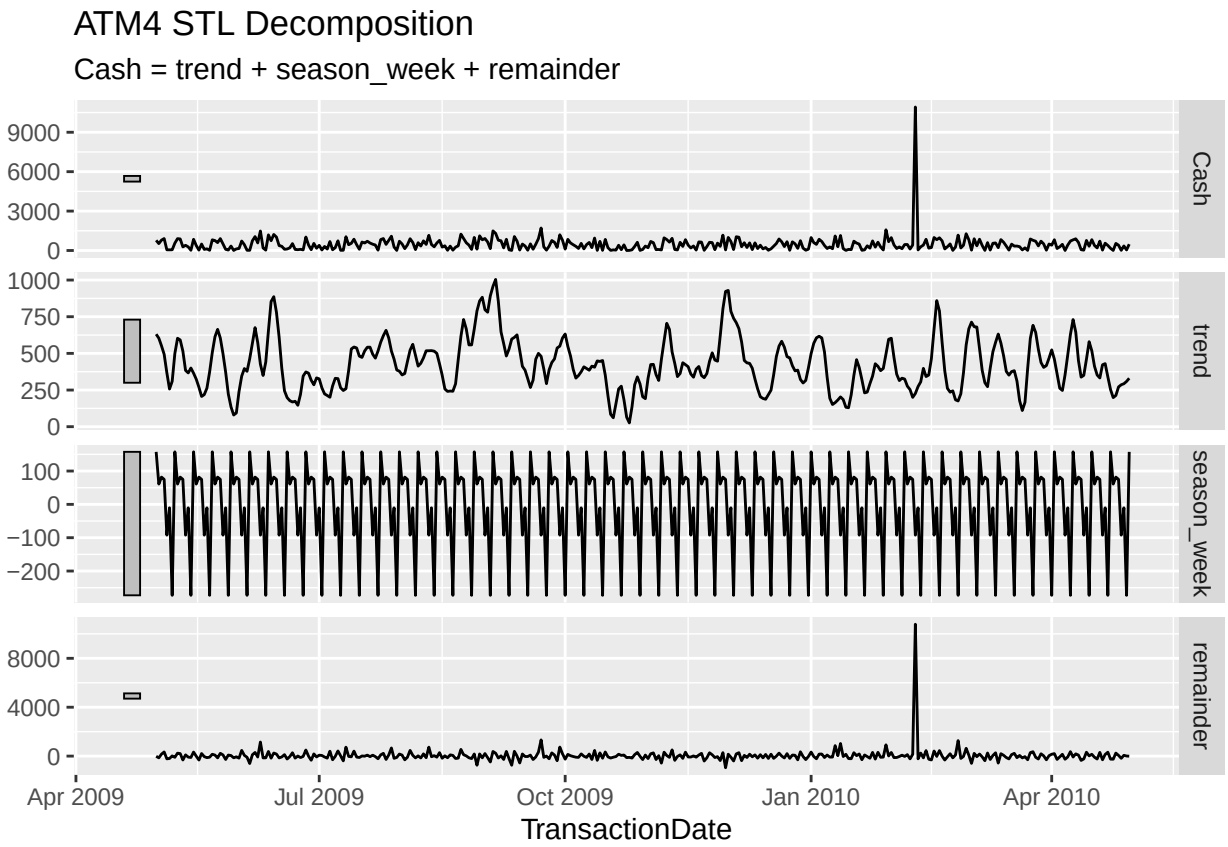


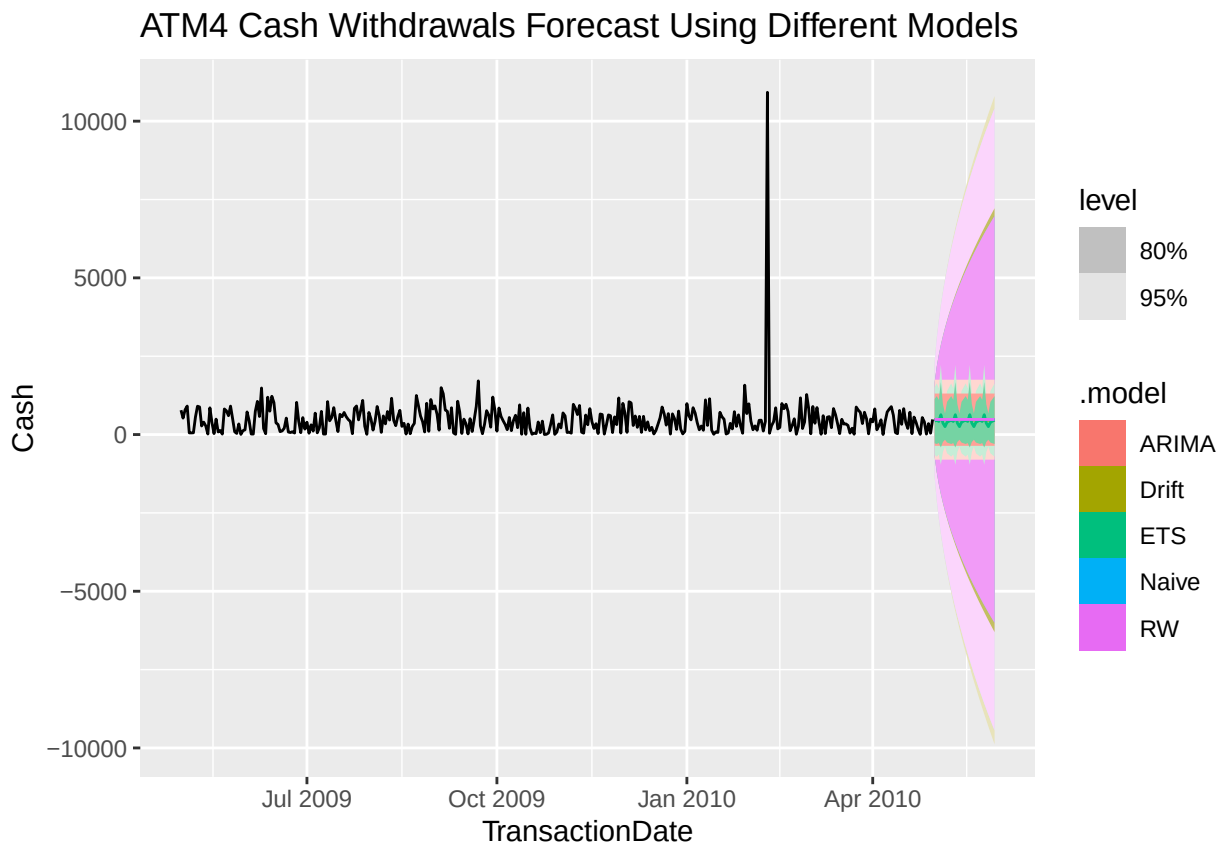

```
## # A tibble: 3 x 9
##   model ATM    RMSE   MAE   MPE  MAPE  MASE RMSSE     ACF1
##   <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>    <dbl>
## 1 ARIMA ATM2   23.8  16.7  -Inf   Inf 0.823 0.803 -0.00268
## 2 ETS   ATM2   24.6  17.2  -Inf   Inf 0.847 0.831  0.0119
## 3 Naive ATM2   54.4  42.6  -Inf   Inf 2.10  1.84  -0.306
```


ATM3 STL Decomposition
Cash = trend + season_week + remainder
Cash
trend
season_week
remainder
100
75
50
25
0
0.0000
−0.0001
0.00015
0.00010
0.00005
0.00000
−0.00005
−0.00010
Apr 2009
Jul 2009
Oct 2009
Jan 2010
Apr 2010
TransactionDate




```
## # A tibble: 3 x 9
##   model ATM    RMSE   MAE   MPE  MAPE  MASE RMSSE     ACF1
##   <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>    <dbl>
## 1 ARIMA ATM3   5.03 0.271  34.6  34.6 0.370 0.625  0.0124
## 2 ETS   ATM3   5.03 0.273 Inf   Inf   0.371 0.625 -0.00736
## 3 Naive ATM3   5.09 0.310  28.8  40.2 0.423 0.632 -0.149
```


ATM4 STL Decomposition
Cash = trend + season_week + remainder
Cash
trend
season_week
remainder
9000
6000
3000
0
1000
750
500
250
0
100
0
−100
−200
8000
4000
0
Apr 2009
Jul 2009
Oct 2009
Jan 2010
Apr 2010
TransactionDate




```
## # A tibble: 3 x 9
##   model ATM    RMSE   MAE   MPE  MAPE  MASE RMSSE     ACF1
##   <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>    <dbl>
## 1 ARIMA ATM4   650.  324. -617.  647. 0.805 0.725 -0.00936
## 2 ETS   ATM4   645.  312. -510.  552. 0.777 0.720 -0.0106
## 3 Naive ATM4   925.  444. -555.  615. 1.10  1.03  -0.488
##    ATM    Total Dollars
## 1 ATM1 1230067.23387713
## 2 ATM2 1167324.38823796
## 3 ATM3  1030012.7844437
## 4 ATM4 6994944.27098873
```

Part B

Forecasting Power, ResidentialCustomerForecastLoad-624.xlsx - Part B consists of a simple dataset of residential power usage for January 1998 until December 2013. Your assignment is to model these data and a monthly forecast for 2014. The data is given in a single file. The variable 'KWH' is power consumption in Kilowatt hours, the rest is straight forward. Add this to your existing files above.

```
# Step 1: Download and Load Excel Data from GitHub
# Setting up the file URL from GitHub and saving it to a temporary path for further use.
data_url <- "https://github.com/Naik-Khyati/data_624/raw/main/p1/ResidentialCustomerForecastLoad-624.xls
temp_file <- tempfile(fileext = ".xlsx")
```

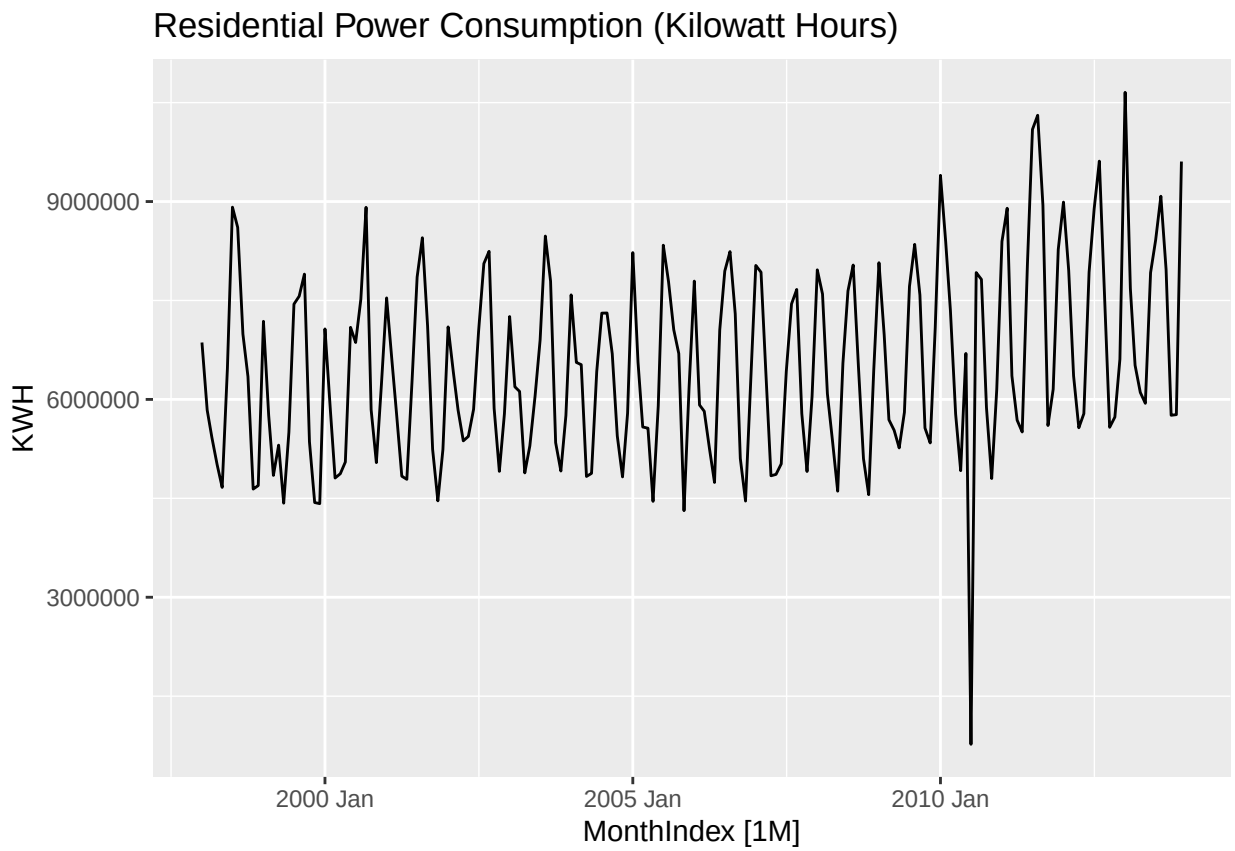
```r
download.file(data_url, temp_file, mode = "wb")

# Step 2: Load the Excel Data into a DataFrame
# Import the downloaded Excel file into an R data frame for analysis.
residential_power_data <- read_excel(temp_file)

# Step 3: Rename the Date Column and Prepare Time Series Data
# Renaming the date column for consistency and transforming the date format to a Year-Month index.
residential_power_data <- rename(residential_power_data, observation_date = `YYYY-MMM`)
power_data_ts <- residential_power_data %>%
  mutate(MonthIndex = yearmonth(observation_date)) %>%  # Convert to year-month format
  select(-CaseSequence, -observation_date) %>%  # Remove unnecessary columns
  tsibble(index = MonthIndex)  # Convert to tsibble format for time series analysis

# Step 4: Identifying Missing Values
# Checking for rows with missing KWH (kilowatt-hour) values.
power_data_ts[which(is.na(power_data_ts$KWH)), ]
```

```
## # A tsibble: 1 x 2 [1M]
##     KWH MonthIndex
##   <dbl>      <mth>
## 1    NA   2008 Sep
```

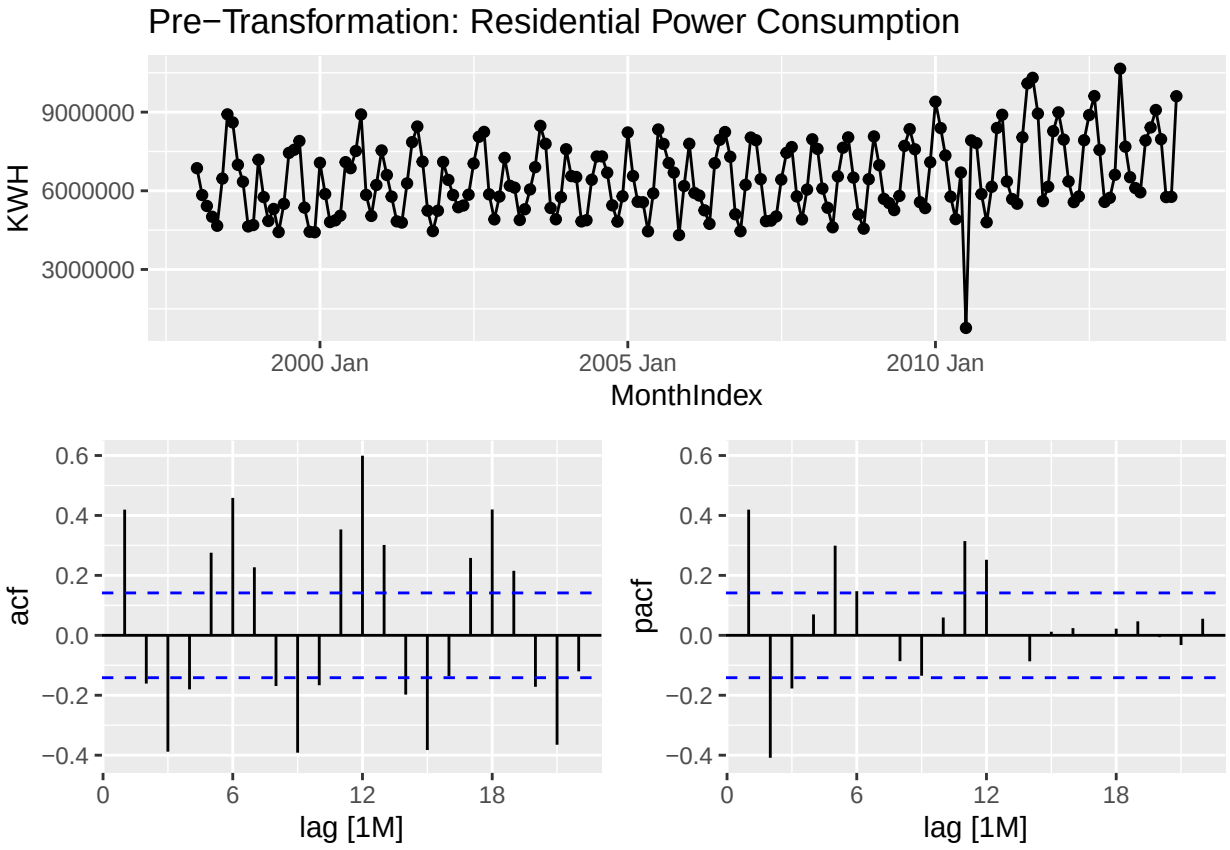
```r
# Step 5: Calculate the Mean for Missing Value Imputation
# Calculate the mean of KWH excluding missing values to replace NA values.
average_KWH <- mean(power_data_ts$KWH, na.rm = TRUE)

# Step 6: Impute Missing KWH Values with Mean
# Replace missing KWH values with the calculated mean for consistent analysis.
power_data_ts$KWH[which(is.na(power_data_ts$KWH))] <- average_KWH

# Step 7: Summarize the KWH Data
# View summary statistics of the KWH column after imputation.
summary(power_data_ts$KWH)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##   770523  5434539  6314472  6502475  7608792 10655730
```

```r
# Step 8: Plotting the Time Series of KWH
# Visualizing the KWH data as a time series to understand overall patterns in usage.
power_data_ts %>%
  autoplot(KWH) +
  labs(title = "Residential Power Consumption (Kilowatt Hours)")
```




```
# Step 9: Time Series Display Before Transformation
# Display partial autocorrelation for the time series data before any transformations.
power_data_ts %>%
  gg_tsdisplay(KWH, plot_type = 'partial') +
  labs(title = "Pre-Transformation: Residential Power Consumption")
```




```
# Step 10: Estimate Lambda for Box-Cox Transformation
# Estimating the best lambda value for the Box-Cox transformation using Guerrero's method.
lambda_value <- power_data_ts %>%
  features(KWH, features = guerrero) %>%
  pull(lambda_guerrero)

# Step 11: Conduct Unit Root Test Using KPSS
# Performing a KPSS test to determine the level of differencing required for stationarity.
power_data_ts %>%
  features(box_cox(KWH, lambda_value), unitroot_ndiffs)
```

```
## # A tibble: 1 x 1
##   ndiffs
##    <int>
## 1      1
```

```
# Step 12: Apply Box-Cox Transformation and Display Results
# Apply the Box-Cox transformation and display the time series data post-transformation.
power_data_ts %>%
  gg_tsdisplay(difference(box_cox(KWH, lambda_value)), plot_type = 'partial') +
  labs(title = paste("Post-Transformation Power Consumption (  = ", round(lambda_value, 2), ")"))
```

```
## Warning: Removed 1 row containing missing values or values outside the scale range
## (`geom_line()`).
```

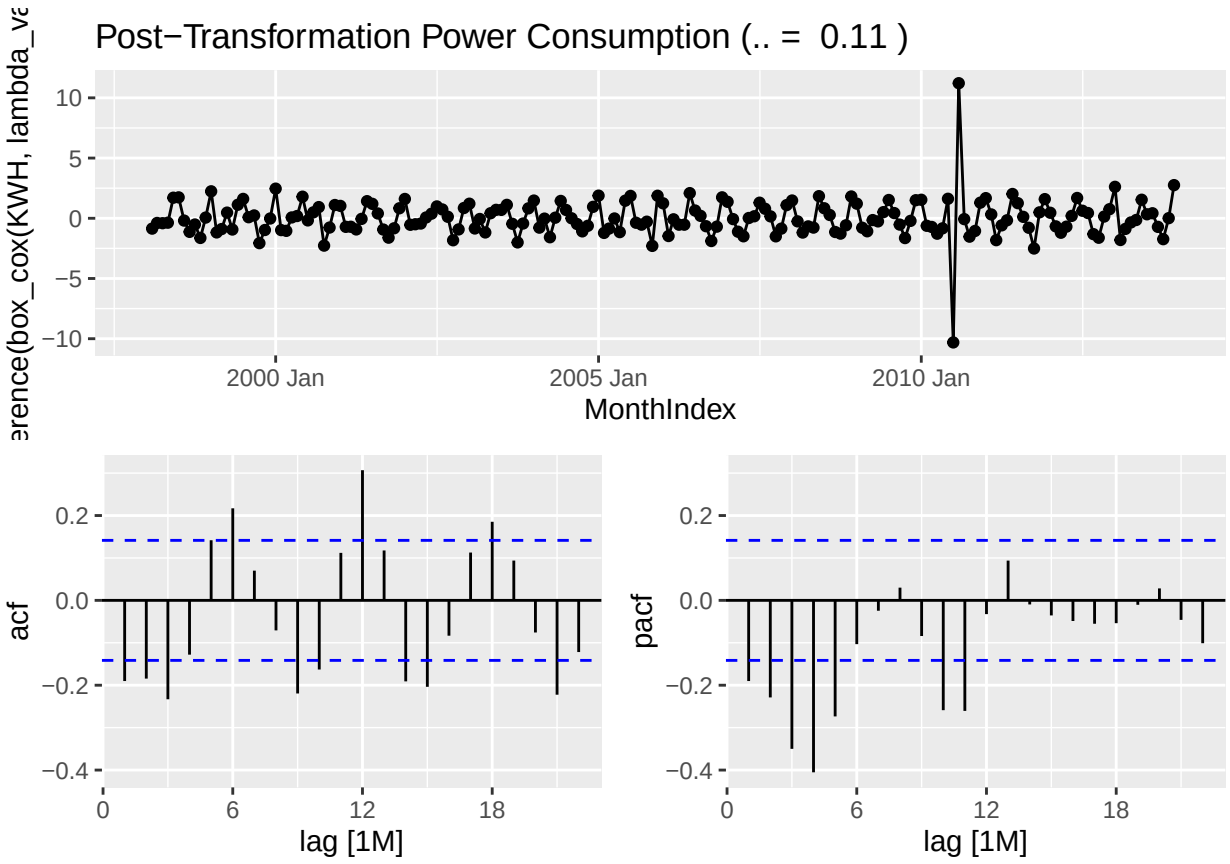
```
## Warning: Removed 1 row containing missing values or values outside the scale range
## (`geom_point()`).
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Post-Transformation Power Consumption (  = 0.11 )' in
## 'mbcsToSbcs': dot substituted for <ce>
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Post-Transformation Power Consumption (  = 0.11 )' in
## 'mbcsToSbcs': dot substituted for <bb>
```

```
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Post-Transformation Power Consumption (  = 0.11 )' in
## 'mbcsToSbcs': dot substituted for <ce>
```

```
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Post-Transformation Power Consumption (  = 0.11 )' in
## 'mbcsToSbcs': dot substituted for <bb>
```

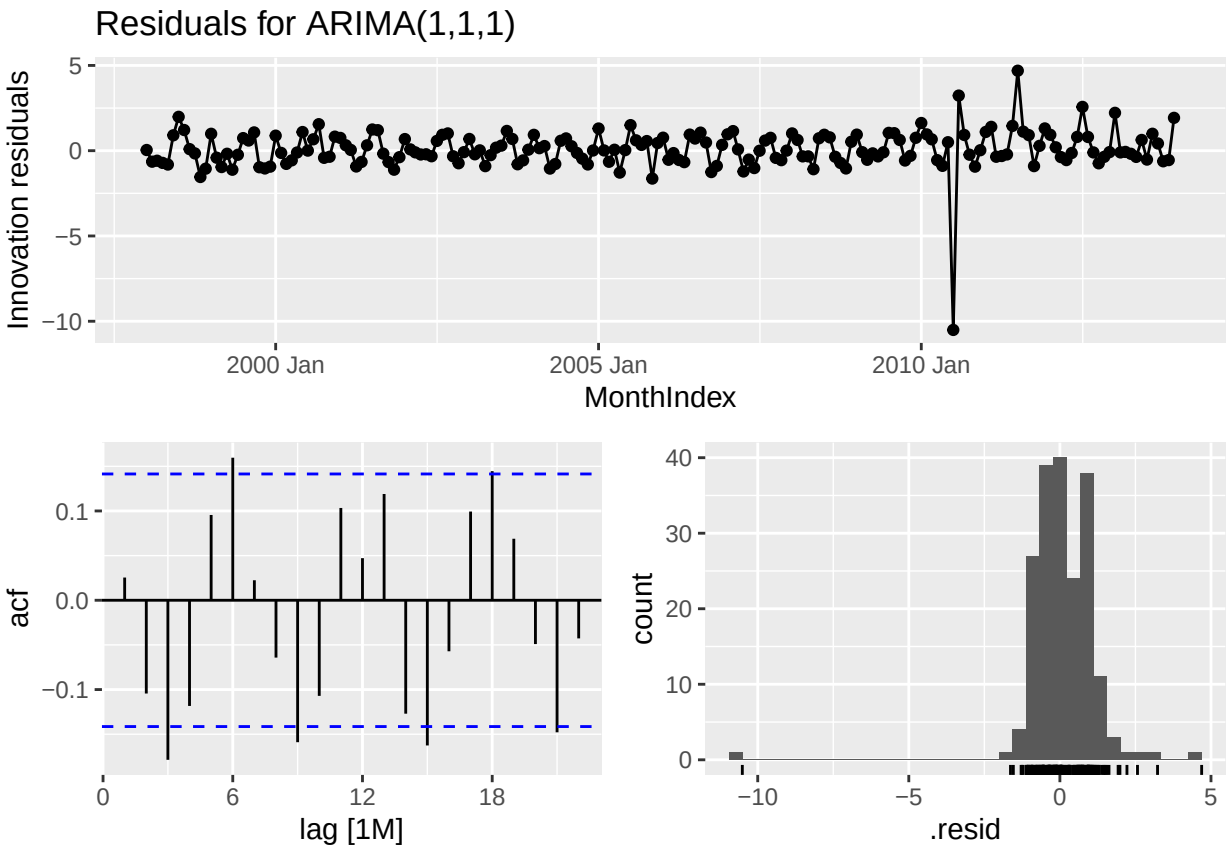



```
# Step 13: Compare Different ARIMA Models
# Fit several ARIMA models with varying orders to identify the best-fitting model.
arima_models <- power_data_ts %>%
  model(
    ARIMA_110 = ARIMA(box_cox(KWH, lambda_value) ~ pdq(1, 1, 0)),
```

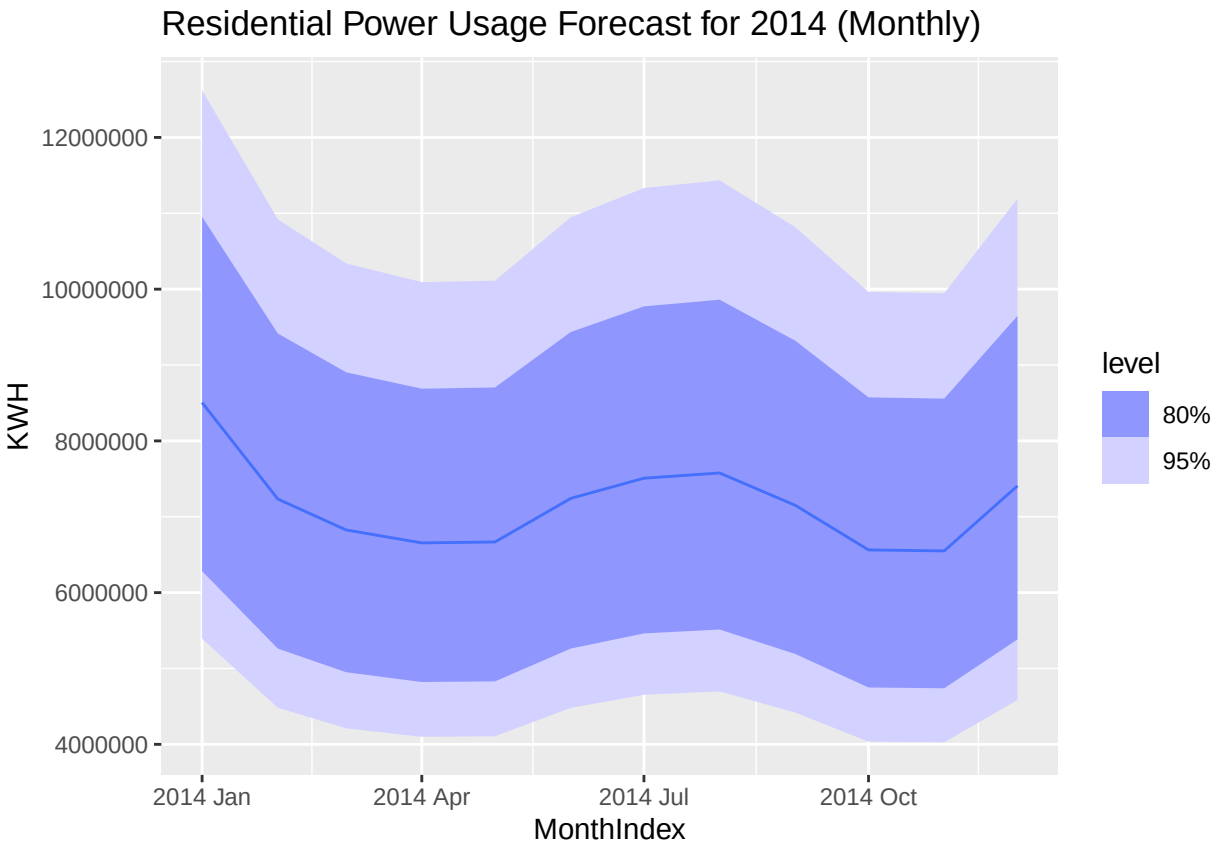
```
    ARIMA_120 = ARIMA(box_cox(KWH, lambda_value) ~ pdq(1, 2, 0)),
    ARIMA_210 = ARIMA(box_cox(KWH, lambda_value) ~ pdq(2, 1, 0)),
    ARIMA_212 = ARIMA(box_cox(KWH, lambda_value) ~ pdq(2, 1, 2)),
    ARIMA_111 = ARIMA(box_cox(KWH, lambda_value) ~ pdq(1, 1, 1))
  )

# Step 14: Review Model Comparison
# Compare the ARIMA models using metrics like AICc and BIC to choose the best fit.
glance(arima_models) %>%
  arrange(AICc) %>%
  select(.model, AICc, BIC)
```

```
## # A tibble: 5 x 3
##   .model     AICc   BIC
##   <chr>     <dbl> <dbl>
## 1 ARIMA_111  617.  633.
## 2 ARIMA_212  624.  640.
## 3 ARIMA_110  663.  676.
## 4 ARIMA_210  676.  692.
## 5 ARIMA_120  793.  806.
```

```
# Step 15: Evaluate Residuals of the Best Model (ARIMA(1,1,1))
# Inspect the residuals of the best-performing ARIMA model to assess fit quality.
arima_models %>%
  select(ARIMA_111) %>%
  gg_tsresiduals() +
  ggtitle("Residuals for ARIMA(1,1,1)")
```




```
# Step 16: Forecasting Next 12 Months (2014) with Best ARIMA Model
# Forecast the next 12 months using the ARIMA(1,1,1) model and visualize the forecast.
forecast_data <- power_data_ts %>%
  model(ARIMA_111 = ARIMA(box_cox(KWH, lambda_value) ~ pdq(1, 1, 1))) %>%
  forecast(h = 12)

# Step 17: Plot Forecast Results for 2014
# Generate a time series plot displaying the forecasted power consumption for 2014.
forecast_data %>%
  autoplot() +
  labs(title = "Residential Power Usage Forecast for 2014 (Monthly)")
```



Two key data transformation steps were undertaken for accurate time series analysis. First, the date column in the dataset was initially formatted as a string. To enable time series processing, this was converted into a year-month index using the yearmonth() function. Following this, the dataset was transformed into a tsibble (time series tibble) format, a structure specifically designed for handling time-indexed data. These steps allow us to leverage advanced time series functions for further analysis, which will be demonstrated in subsequent steps.

The second step in preparing the dataset was to handle missing values, specifically focusing on data cleaning and imputation. In this dataset, we identified only a single missing data point, which lacked a value for KWH. Given the seasonal patterns and consistency observed in the data, mean imputation was chosen as the best method to fill in this gap. This approach maintains the dataset's overall structure and minimizes the impact on the seasonal trends present in the data.

To evaluate the best forecasting model, I compared several ARIMA models. Among these, the ARIMA(1,1,1) consistently outperformed others across key metrics: the Akaike Information Criterion (AIC), corrected Akaike Information Criterion (AICc), and Bayesian Information Criterion (BIC). These lower values indicate that the ARIMA(1,1,1) model is the most suitable fit for our data. This result aligns with expectations; given the relatively small number of parameters, a simpler model is preferable as it reduces the risk of overfitting and limits unnecessary noise in the forecast. The ARIMA(1,1,1) model strikes a balance between accuracy and model simplicity, making it the optimal choice for our analysis.

In the forecasting analysis, some residual outliers are evident, particularly noticeable in the ACF (Autocorrelation Function) graph. Here, peaks extending beyond the bounded area indicate significant autocorrelation at those lags, surpassing the significance threshold. Additionally, the residual plot highlights a prominent outlier below -10, which stands out from the other residuals. Despite these outliers, the residuals are generally distributed around zero, suggesting that the forecast errors follow a roughly normal distribution. This overall pattern supports the reliability of the model, even with some outlier presence, as it indicates that most forecast errors are minor and unbiased.

Part C

BONUS, optional (part or all), Waterflow_Pipe1.xlsx and Waterflow_Pipe2.xlsx Part C consists of two data sets. These are simple 2 columns sets, however they have different time stamps. Your optional assignment is to time-base sequence the data and aggregate based on hour (example of what this looks like, follows). Note for multiple recordings within an hour, take the mean. Then to determine if the data is stationary and can it be forecast. If so, provide a week forward forecast and present results via Rpubs and .rmd and the forecast in an Excel readable file.

```
pipe1_url <- "https://github.com/Naik-Khyati/data_624/raw/main/p1/Waterflow_Pipe1.xlsx"
pipe2_url <- "https://github.com/Naik-Khyati/data_624/raw/main/p1/Waterflow_Pipe2.xlsx"

# Create temporary file paths for downloading the Excel files
temp_pipe1 <- tempfile(fileext = ".xlsx")
temp_pipe2 <- tempfile(fileext = ".xlsx")

# Download the water flow data files from GitHub to the temporary file paths
download.file(pipe1_url, temp_pipe1, mode = "wb")
download.file(pipe2_url, temp_pipe2, mode = "wb")

# Read the Excel data into data frames with appropriate column types
pipe1_data_raw <- read_excel(temp_pipe1, col_types = c("date", "numeric"))
pipe2_data_raw <- read_excel(temp_pipe2, col_types = c("date", "numeric"))

# Convert the 'Date Time' column from Excel format to POSIXct format for proper date-time handling
pipe1_data_raw$`Date Time` <- as.POSIXct(pipe1_data_raw$`Date Time`,
                                         origin = "1899-12-30", tz = "GMT")
pipe2_data_raw$`Date Time` <- as.POSIXct(pipe2_data_raw$`Date Time`,
                                         origin = "1899-12-30", tz = "GMT")

# Preview the first few rows of the data to check the structure and values
head(pipe1_data_raw)
```

```
## # A tibble: 6 x 2
##   `Date Time`         WaterFlow
##   <dttm>                  <dbl>
## 1 2015-10-23 00:24:06     23.4
## 2 2015-10-23 00:40:02     28.0
## 3 2015-10-23 00:53:51     23.1
## 4 2015-10-23 00:55:40     30.0
## 5 2015-10-23 01:19:17      6.00
## 6 2015-10-23 01:23:58     15.9
```

```
head(pipe2_data_raw)
```

```
## # A tibble: 6 x 2
##   `Date Time`         WaterFlow
##   <dttm>                  <dbl>
## 1 2015-10-23 01:00:00      18.8
## 2 2015-10-23 02:00:00      43.1
## 3 2015-10-23 03:00:00      38.0
```

```
## 4 2015-10-23 04:00:00      36.1
## 5 2015-10-23 05:00:00      31.9
## 6 2015-10-23 06:00:00      28.2
```

```
# Extract Date and Time from the 'Date Time' variable
pipe1_data_raw$Date <- as.Date(pipe1_data_raw$`Date Time`)  # Extract date part
pipe1_data_raw$Hour <- hour(pipe1_data_raw$`Date Time`) + 1  # Extract hour and adjust by adding 1

# Group the data by Date and Hour, calculating the average water flow for each hour
pipe1_data_aggregated <- pipe1_data_raw %>%
  group_by(Date, Hour) %>%
  summarise(Average_Water_Flow = mean(WaterFlow, na.rm = TRUE)) %>%
  ungroup()  # Remove grouping structure
```

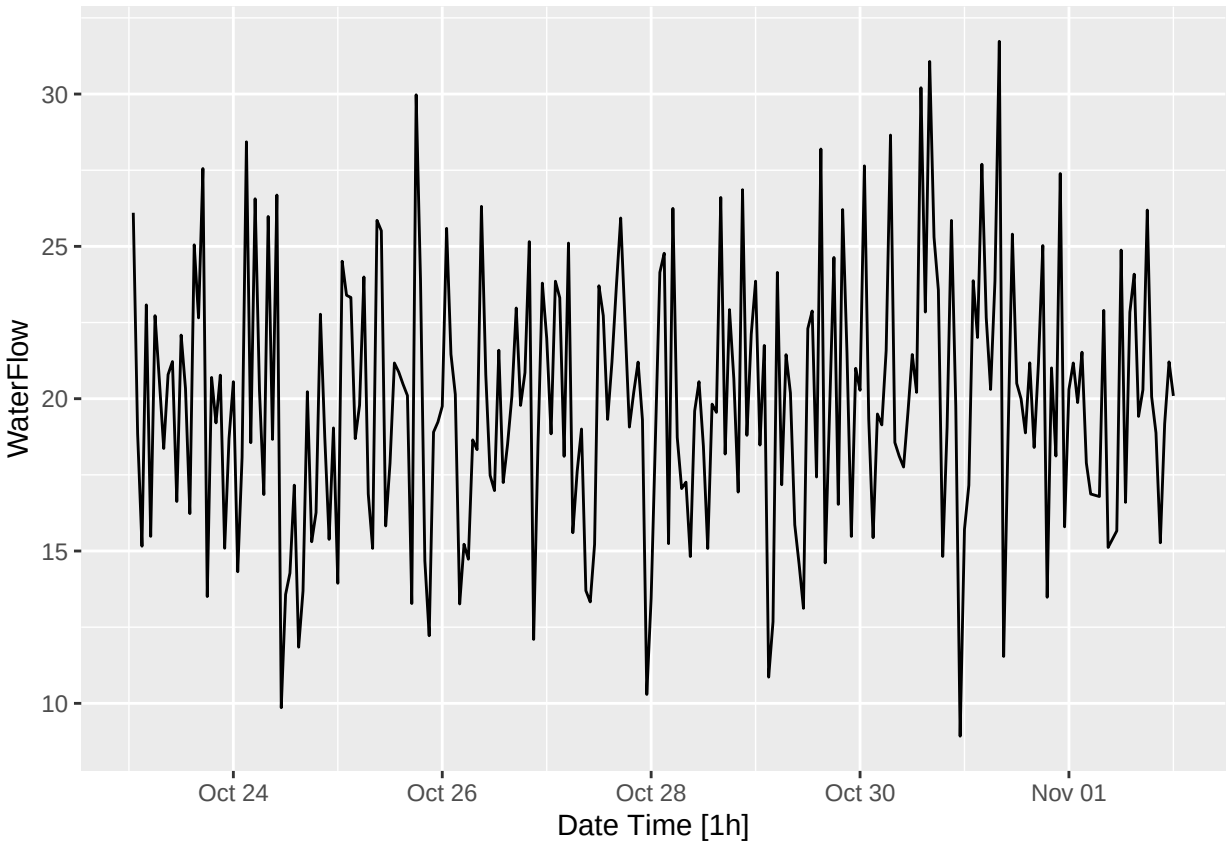
```
## `summarise()` has grouped output by 'Date'. You can override using the
## `.groups` argument.
```

```
# Create a new 'Date Time' column combining Date and Hour for time series analysis
pipe1_data_aggregated$`Date Time` <- with(pipe1_data_aggregated, ymd_h(paste(Date, Hour)))

# Select relevant columns and rename for clarity
pipe1_data_cleaned <- pipe1_data_aggregated %>%
  select(c(`Date Time`, Average_Water_Flow)) %>%
  rename(WaterFlow = Average_Water_Flow)

# Preview the cleaned data
head(pipe1_data_cleaned)
```

```
## # A tibble: 6 x 2
##   `Date Time`         WaterFlow
##   <dttm>                  <dbl>
## 1 2015-10-23 01:00:00      26.1
## 2 2015-10-23 02:00:00      18.9
## 3 2015-10-23 03:00:00      15.2
## 4 2015-10-23 04:00:00      23.1
## 5 2015-10-23 05:00:00      15.5
## 6 2015-10-23 06:00:00      22.7
```

```
pipe1_data_ts <- pipe1_data_cleaned %>%
  as_tsibble(index = `Date Time`)

pipe1_data_ts %>% autoplot()
```

```
## Plot variable not specified, automatically selected `.vars = WaterFlow`
```


30
25
20
15
10
WaterFlow
Oct 24
Oct 26
Oct 28
Oct 30
Nov 01
Date Time [1h]