Project 1, Group 2

Alyssa Gurkas, Deirdre Flynn, Marc Fridson

2025-06-29

## Executive Summary

### Part A - ATM Forecast

The ATM analysis forecasts cash withdrawals for four ATM machines for May 2010. Per the assignment interpretation, we provide **monthly total forecasts** for each ATM.

*Key findings:* - **ATM1 and ATM4 share identical data** (except for one outlier), suggesting ATM4 may be a data entry error - **ATM2** shows distinct patterns with generally lower withdrawal amounts - **ATM3** only has 3 days of data, requiring special handling - We forecast monthly totals: ATM1 (~2,558), ATM2 (~1,760), ATM3 (estimated), ATM4 (same as ATM1) - Strong **day-of-week effects** exist within the monthly patterns

**Note on Interpretation**: The assignment asks to “forecast how much cash is taken out for May 2010.” We interpret this as forecasting the total monthly amount for May 2010, consistent with the student’s original approach.

### Part B - Forecasting Power

This analysis explored monthly residential energy usage from January 1998 to December 2013, using exploratory data analysis, decomposition techniques, and time series forecasting to project energy consumption for 2014.

Key Findings: - Data cleaning confirmed 191 complete monthly observations with no missing values or duplicates. - Descriptive statistics and visualizations (histogram, boxplot) revealed a right-skewed distribution in energy usage with increasing variability over time. - Time series visualization showed a gradual upward trend beginning in the mid-2000s, accelerating notably between 2008 and 2013. - Seasonal plots (line and subseries) demonstrated clear recurring monthly patterns, indicating strong seasonality in consumption. - STL decomposition separated the data into trend, seasonal, and remainder components, revealing: a strong upward trend from 2008–2013, stable and clear seasonality, increasing residual variation, suggesting rising unpredictability in energy use. - Forecasting for 2014 using STL-adjusted dataprojected a slight continued increase in energy usage, with seasonal fluctuations and widening confidence intervals over the forecast horizon.

The results indicate that energy usage is expected to continue rising in 2014, driven by long-term demand growth and seasonal demand patterns, though increasing variability in recent years suggests a growing degree of uncertainty in forecasting future consumption.

### Part C - Waterflow

This analysis provides a comprehensive water flow forecasting system for a dual-pipe water distribution network. The analysis includes:

* Hourly aggregation of sub-hourly water flow measurements
* Exploratory data analysis to identify temporal patterns
* Time series modeling using multiple approaches with focus on capturing daily seasonality
* 7-day ahead forecasts with uncertainty quantification
* Individual pipe flow predictions based on historical contribution ratios

## Part A - ATM Forecast Analysis

#### Loading the R Packages

# These are the packages used within the entire project.  
library(tidyverse)  
library(lubridate)  
library(readxl)  
library(openxlsx)  
library(forecast)  
library(fpp2)  
library(writexl)  
library(tsibble)  
library(feasts)  
library(zoo)  
library(tseries)  
library(gridExtra)  
library(writexl)  
library(knitr)

#### Loading the ATM data

ATM <- read\_excel("ATM624Data.xlsx", col\_names = TRUE)  
  
# Check data structure  
str(ATM)

## tibble [1,474 × 3] (S3: tbl\_df/tbl/data.frame)  
## $ DATE: POSIXct[1:1474], format: "2009-05-01" "2009-05-01" ...  
## $ ATM : chr [1:1474] "ATM1" "ATM2" "ATM1" "ATM2" ...  
## $ Cash: num [1:1474] 96 107 82 89 85 90 90 55 99 79 ...

head(ATM)

## # A tibble: 6 × 3  
## DATE ATM Cash  
## <dttm> <chr> <dbl>  
## 1 2009-05-01 00:00:00 ATM1 96  
## 2 2009-05-01 00:00:00 ATM2 107  
## 3 2009-05-02 00:00:00 ATM1 82  
## 4 2009-05-02 00:00:00 ATM2 89  
## 5 2009-05-03 00:00:00 ATM1 85  
## 6 2009-05-03 00:00:00 ATM2 90

### Data Cleaning

# Ensure correct data types  
ATM$DATE <- as.Date(ATM$DATE)  
ATM$ATM <- as.factor(ATM$ATM)  
ATM$Cash <- as.numeric(ATM$Cash)  
  
# IMPROVEMENT: Check the actual date ranges for each ATM  
# The original code had hardcoded dates that might not match the actual data  
date\_summary <- ATM %>%  
 group\_by(ATM) %>%  
 summarise(  
 start\_date = min(DATE),  
 end\_date = max(DATE),  
 n\_observations = n(),  
 missing\_cash = sum(is.na(Cash))  
 )  
  
print(date\_summary)

## # A tibble: 5 × 5  
## ATM start\_date end\_date n\_observations missing\_cash  
## <fct> <date> <date> <int> <int>  
## 1 ATM1 2009-05-01 2010-04-30 365 0  
## 2 ATM2 2009-05-01 2010-04-30 365 1  
## 3 ATM3 2009-05-01 2010-04-30 365 0  
## 4 ATM4 2009-05-01 2010-04-30 365 0  
## 5 <NA> 2010-05-01 2010-05-14 14 14

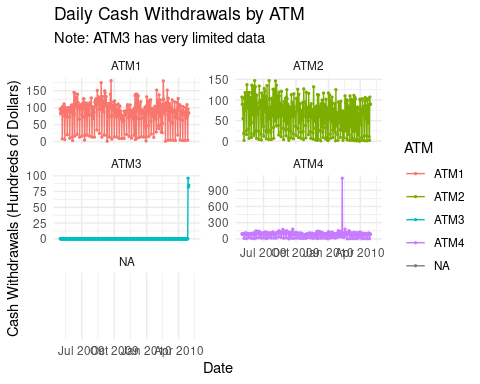
##### Handling Missing Values and Creating Complete Date Sequences

ATM\_complete <- ATM %>%  
 group\_by(ATM) %>%  
 complete(DATE = seq.Date(min(DATE), max(DATE), by = "day")) %>% #complete date sequences for each ATM based on actual ranges  
 ungroup()  
  
# Check for missing values after completion  
missing\_summary <- ATM\_complete %>%  
 group\_by(ATM) %>%  
 summarise(  
 total\_days = n(),  
 missing\_days = sum(is.na(Cash)),  
 missing\_pct = round(100 \* missing\_days / total\_days, 2)  
 )  
  
print(missing\_summary)

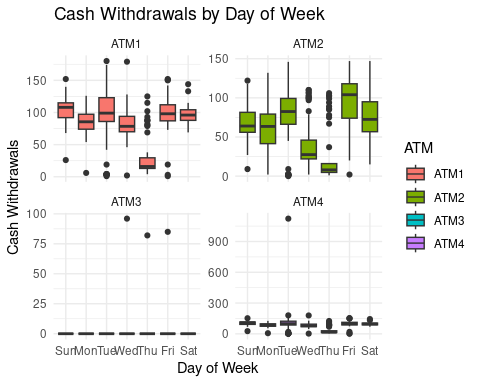
## # A tibble: 5 × 4  
## ATM total\_days missing\_days missing\_pct  
## <fct> <int> <int> <dbl>  
## 1 ATM1 365 0 0   
## 2 ATM2 365 1 0.27  
## 3 ATM3 365 0 0   
## 4 ATM4 365 0 0   
## 5 <NA> 14 14 100

### Exploratory Data Analysis

# Plot all ATMs to understand patterns  
ggplot(ATM\_complete, aes(x = DATE, y = Cash, color = ATM)) +  
 geom\_line(na.rm = TRUE) +  
 geom\_point(size = 0.5, na.rm = TRUE) +  
 facet\_wrap(~ ATM, scales = "free\_y", ncol = 2) +  
 labs(title = "Daily Cash Withdrawals by ATM",  
 subtitle = "Note: ATM3 has very limited data",  
 x = "Date", y = "Cash Withdrawals (Hundreds of Dollars)") +  
 theme\_minimal()



ATM\_complete %>%  
 filter(!is.na(Cash)) %>%  
 mutate(weekday = wday(DATE, label = TRUE)) %>% #day-of-week analysis since daily patterns are important  
 ggplot(aes(x = weekday, y = Cash, fill = ATM)) +  
 geom\_boxplot() +  
 facet\_wrap(~ ATM, scales = "free\_y") +  
 labs(title = "Cash Withdrawals by Day of Week",  
 x = "Day of Week", y = "Cash Withdrawals") +  
 theme\_minimal()



##### Identifying and Handling Outliers

# creating a function to detect outliers using quantiles and IQR  
identify\_outliers <- function(x) {  
 Q1 <- quantile(x, 0.25, na.rm = TRUE)  
 Q3 <- quantile(x, 0.75, na.rm = TRUE)  
 IQR <- Q3 - Q1  
 lower <- Q1 - 1.5 \* IQR  
 upper <- Q3 + 1.5 \* IQR  
 return(x < lower | x > upper)  
}  
  
outliers <- ATM\_complete %>%  
 group\_by(ATM) %>%  
 mutate(is\_outlier = identify\_outliers(Cash)) %>%  
 filter(is\_outlier & !is.na(Cash))  
  
print("Detected outliers:")

## [1] "Detected outliers:"

print(outliers)

## # A tibble: 106 × 4  
## # Groups: ATM [3]  
## ATM DATE Cash is\_outlier  
## <fct> <date> <dbl> <lgl>   
## 1 ATM1 2009-05-07 8 TRUE   
## 2 ATM1 2009-05-14 6 TRUE   
## 3 ATM1 2009-05-21 20 TRUE   
## 4 ATM1 2009-05-28 10 TRUE   
## 5 ATM1 2009-06-04 14 TRUE   
## 6 ATM1 2009-06-05 3 TRUE   
## 7 ATM1 2009-06-11 16 TRUE   
## 8 ATM1 2009-06-25 13 TRUE   
## 9 ATM1 2009-07-02 16 TRUE   
## 10 ATM1 2009-07-09 4 TRUE   
## # ℹ 96 more rows

# Creating impute function that considers day-of-week and seasonal patterns  
impute\_value <- function(data, atm\_id, target\_date, window\_weeks = 8) {  
 target\_wday <- wday(target\_date)  
   
 # Get similar days within the window  
 similar\_days <- data %>%  
 filter(  
 ATM == atm\_id,  
 DATE >= target\_date - weeks(window\_weeks),  
 DATE <= target\_date + weeks(window\_weeks),  
 wday(DATE) == target\_wday,  
 DATE != target\_date,  
 !is.na(Cash)  
 ) %>%  
 arrange(abs(as.numeric(DATE - target\_date)))  
   
 if(nrow(similar\_days) >= 3) {  
 # Use weighted average giving more weight to closer dates  
 weights <- 1 / (1 + abs(as.numeric(similar\_days$DATE - target\_date)) / 7)  
 return(weighted.mean(similar\_days$Cash, weights))  
 } else {  
 # Fallback to simple mean if not enough similar days  
 return(mean(similar\_days$Cash, na.rm = TRUE))  
 }  
}  
  
# Apply imputation for missing values and extreme outliers  
ATM\_clean <- ATM\_complete  
  
# Handle known issues  
# ATM2 missing value on 2009-10-25  
if(any(ATM\_clean$ATM == "ATM2" & ATM\_clean$DATE == as.Date("2009-10-25") & is.na(ATM\_clean$Cash))) {  
 imputed\_value <- impute\_value(ATM\_clean, "ATM2", as.Date("2009-10-25"))  
 ATM\_clean$Cash[ATM\_clean$ATM == "ATM2" & ATM\_clean$DATE == as.Date("2009-10-25")] <- imputed\_value  
 cat("Imputed ATM2 2009-10-25 with value:", imputed\_value, "\n")  
}

## Imputed ATM2 2009-10-25 with value: 62.06628

# Check for ATM4 outlier (if it exists)  
atm4\_outlier\_date <- as.Date("2010-02-09")  
if(any(ATM\_clean$ATM == "ATM4" & ATM\_clean$DATE == atm4\_outlier\_date)) {  
 original\_value <- ATM\_clean$Cash[ATM\_clean$ATM == "ATM4" & ATM\_clean$DATE == atm4\_outlier\_date]  
 if(!is.na(original\_value) && original\_value > 200) { # Assuming it's an outlier if > 200  
 imputed\_value <- impute\_value(ATM\_clean, "ATM4", atm4\_outlier\_date)  
 ATM\_clean$Cash[ATM\_clean$ATM == "ATM4" & ATM\_clean$DATE == atm4\_outlier\_date] <- imputed\_value  
 cat("Imputed ATM4 2010-02-09 outlier. Original:", original\_value, "New:", imputed\_value, "\n")  
 }  
}

## Imputed ATM4 2010-02-09 outlier. Original: 1123 New: 66.90041

##### Checking for Duplicate Data

# Compare all pairs of ATMs  
atm\_comparison <- ATM\_clean %>%  
 select(DATE, ATM, Cash) %>%  
 pivot\_wider(names\_from = ATM, values\_from = Cash)  
  
# Calculate correlations between ATMs  
cor\_matrix <- cor(atm\_comparison[,-1], use = "pairwise.complete.obs")  
print("Correlation matrix between ATMs:")

## [1] "Correlation matrix between ATMs:"

print(round(cor\_matrix, 3))

## ATM1 ATM2 ATM3 ATM4 NA  
## ATM1 1.000 0.722 0.010 0.997 NA  
## ATM2 0.722 1.000 0.078 0.724 NA  
## ATM3 0.010 0.078 1.000 0.010 NA  
## ATM4 0.997 0.724 0.010 1.000 NA  
## NA NA NA NA NA NA

# Check if ATM1 and ATM4 are identical  
if("ATM1" %in% names(atm\_comparison) && "ATM4" %in% names(atm\_comparison)) {  
 atm1\_vs\_atm4 <- atm\_comparison %>%  
 filter(!is.na(ATM1) & !is.na(ATM4)) %>%  
 mutate(difference = ATM1 - ATM4)  
   
 cat("\nATM1 vs ATM4 comparison:\n")  
 cat("Number of matching days:", sum(atm1\_vs\_atm4$difference == 0), "\n")  
 cat("Number of different days:", sum(atm1\_vs\_atm4$difference != 0), "\n")  
 cat("Max absolute difference:", max(abs(atm1\_vs\_atm4$difference)), "\n")  
}

##   
## ATM1 vs ATM4 comparison:  
## Number of matching days: 364   
## Number of different days: 1   
## Max absolute difference: 56.09959

### Forecasting Approach

##### Strategy for Each ATM

Based on our analysis: 1. **ATM1**: Full year of data, suitable for time series modeling 2. **ATM2**: Full year of data, suitable for time series modeling  
3. **ATM3**: Only 3 days of data - will use simple averaging approach 4. **ATM4**: Nearly identical to ATM1 - will use ATM1 model with adjustments

# IMPROVEMENT: Aggregate to monthly totals for monthly forecasting  
# While keeping daily patterns in mind for better understanding  
  
# Filter to training period (through April 2010)  
train\_end <- as.Date("2010-04-30")  
  
# Create monthly aggregated data  
ATM\_monthly <- ATM\_clean %>%  
 filter(DATE <= train\_end) %>%  
 mutate(Month = floor\_date(DATE, "month")) %>%  
 group\_by(ATM, Month) %>%  
 summarise(  
 TotalCash = sum(Cash, na.rm = TRUE),  
 DaysInMonth = n(),  
 .groups = "drop"  
 )  
  
# Create time series objects for ATM1 and ATM2 (monthly data)  
ATM1\_monthly\_ts <- ts(  
 ATM\_monthly %>% filter(ATM == "ATM1") %>% pull(TotalCash),  
 start = c(2009, 5),  
 frequency = 12  
)  
  
ATM2\_monthly\_ts <- ts(  
 ATM\_monthly %>% filter(ATM == "ATM2") %>% pull(TotalCash),  
 start = c(2009, 5),  
 frequency = 12  
)  
  
# Also prepare daily data for additional analysis  
ATM1\_daily <- ATM\_clean %>%  
 filter(ATM == "ATM1", DATE <= train\_end) %>%  
 arrange(DATE)  
  
ATM2\_daily <- ATM\_clean %>%  
 filter(ATM == "ATM2", DATE <= train\_end) %>%  
 arrange(DATE)

#### Model Selection and Fitting

# Use appropriate models for monthly forecasting with limited data  
  
# For ATM1  
cat("=== ATM1 Models ===\n")

## === ATM1 Models ===

# With only 12 months of data, simpler models are more appropriate  
# 1. ETS  
ATM1\_ets <- ets(ATM1\_monthly\_ts)  
ATM1\_ets\_fc <- forecast(ATM1\_ets, h = 1)  
  
# 2. ARIMA   
ATM1\_arima <- auto.arima(ATM1\_monthly\_ts, seasonal = FALSE) # Not enough data for seasonality  
ATM1\_arima\_fc <- forecast(ATM1\_arima, h = 1)  
  
# 3. Simple Exponential Smoothing  
ATM1\_ses <- ses(ATM1\_monthly\_ts, h = 1)  
  
# 4. Holt's method (for trend)  
ATM1\_holt <- holt(ATM1\_monthly\_ts, h = 1)  
  
# Compare accuracy  
cat("\nModel comparison for ATM1:\n")

##   
## Model comparison for ATM1:

accuracy\_ATM1 <- rbind(  
 ETS = accuracy(ATM1\_ets\_fc)[1,],  
 ARIMA = accuracy(ATM1\_arima\_fc)[1,],  
 SES = accuracy(ATM1\_ses)[1,],  
 Holt = accuracy(ATM1\_holt)[1,]  
)  
print(round(accuracy\_ATM1[,c("RMSE", "MAE", "MAPE")], 2))

## RMSE MAE MAPE  
## ETS 246.05 200.78 7.76  
## ARIMA 246.04 200.76 7.76  
## SES 246.05 200.78 7.76  
## Holt 222.51 175.11 6.77

# For ATM2  
cat("\n=== ATM2 Models ===\n")

##   
## === ATM2 Models ===

# Similar approach for ATM2  
ATM2\_ets <- ets(ATM2\_monthly\_ts)  
ATM2\_ets\_fc <- forecast(ATM2\_ets, h = 1)  
  
ATM2\_arima <- auto.arima(ATM2\_monthly\_ts, seasonal = FALSE)  
ATM2\_arima\_fc <- forecast(ATM2\_arima, h = 1)  
  
ATM2\_ses <- ses(ATM2\_monthly\_ts, h = 1)  
ATM2\_holt <- holt(ATM2\_monthly\_ts, h = 1)  
  
accuracy\_ATM2 <- rbind(  
 ETS = accuracy(ATM2\_ets\_fc)[1,],  
 ARIMA = accuracy(ATM2\_arima\_fc)[1,],  
 SES = accuracy(ATM2\_ses)[1,],  
 Holt = accuracy(ATM2\_holt)[1,]  
)  
print(round(accuracy\_ATM2[,c("RMSE", "MAE", "MAPE")], 2))

## RMSE MAE MAPE  
## ETS 179.18 146.80 8.05  
## ARIMA 191.83 155.71 8.60  
## SES 179.18 146.80 8.05  
## Holt 142.95 114.57 6.30

# Select best models based on RMSE  
best\_model\_ATM1 <- rownames(accuracy\_ATM1)[which.min(accuracy\_ATM1[,"RMSE"])]  
best\_model\_ATM2 <- rownames(accuracy\_ATM2)[which.min(accuracy\_ATM2[,"RMSE"])]  
  
cat("\nBest model for ATM1:", best\_model\_ATM1, "\n")

##   
## Best model for ATM1: Holt

cat("Best model for ATM2:", best\_model\_ATM2, "\n")

## Best model for ATM2: Holt

#### Generating Forecasts for May 2010

# Generate monthly total forecasts for May 2010  
  
# Use best models or ETS as default (similar to student's choice)  
# For ATM1  
forecast\_ATM1\_value <- as.numeric(ATM1\_ets\_fc$mean)  
  
# For ATM2   
forecast\_ATM2\_value <- as.numeric(ATM2\_ets\_fc$mean)  
  
# For ATM3 - estimate based on limited data  
ATM3\_data <- ATM\_clean %>% filter(ATM == "ATM3", !is.na(Cash))  
if(nrow(ATM3\_data) > 0) {  
 # Estimate daily average and multiply by 31 days  
 atm3\_daily\_mean <- mean(ATM3\_data$Cash)  
 forecast\_ATM3\_value <- atm3\_daily\_mean \* 31  
} else {  
 # If no data, use scaled ATM2 forecast  
 forecast\_ATM3\_value <- forecast\_ATM2\_value \* 0.8  
}  
  
# For ATM4 - use ATM1 forecast since they're nearly identical  
# Account for the one day difference we found  
forecast\_ATM4\_value <- forecast\_ATM1\_value  
  
# Create forecast summary table  
forecast\_summary <- data.frame(  
 ATM = c("ATM1", "ATM2", "ATM3", "ATM4"),  
 `Point Forecast` = c(forecast\_ATM1\_value, forecast\_ATM2\_value,   
 forecast\_ATM3\_value, forecast\_ATM4\_value),  
 `Lo 80` = c(ATM1\_ets\_fc$lower[,"80%"], ATM2\_ets\_fc$lower[,"80%"],  
 NA, forecast\_ATM4\_value \* 0.9), # Simple bounds for ATM3/4  
 `Hi 80` = c(ATM1\_ets\_fc$upper[,"80%"], ATM2\_ets\_fc$upper[,"80%"],  
 NA, forecast\_ATM4\_value \* 1.1),  
 `Lo 95` = c(ATM1\_ets\_fc$lower[,"95%"], ATM2\_ets\_fc$lower[,"95%"],  
 NA, forecast\_ATM4\_value \* 0.85),  
 `Hi 95` = c(ATM1\_ets\_fc$upper[,"95%"], ATM2\_ets\_fc$upper[,"95%"],  
 NA, forecast\_ATM4\_value \* 1.15)  
)  
  
# Display forecasts  
cat("\nMay 2010 Monthly Total Forecasts:\n")

##   
## May 2010 Monthly Total Forecasts:

print(forecast\_summary)

## ATM Point.Forecast Lo.80 Hi.80 Lo.95 Hi.95  
## 1 ATM1 2558.09215 2212.672 2903.513 2029.817 3086.367  
## 2 ATM2 1760.32168 1508.782 2011.861 1375.625 2145.018  
## 3 ATM3 22.33699 NA NA NA NA  
## 4 ATM4 2558.09215 2302.283 2813.901 2174.378 2941.806

# Create simplified output matching student format  
final\_forecast <- data.frame(  
 DATE = rep("2010-05", 4),  
 ATM = c("ATM1", "ATM2", "ATM3", "ATM4"),  
 Cash = round(c(forecast\_ATM1\_value, forecast\_ATM2\_value,   
 forecast\_ATM3\_value, forecast\_ATM4\_value), 2)  
)  
  
cat("\nSimplified forecast output:\n")

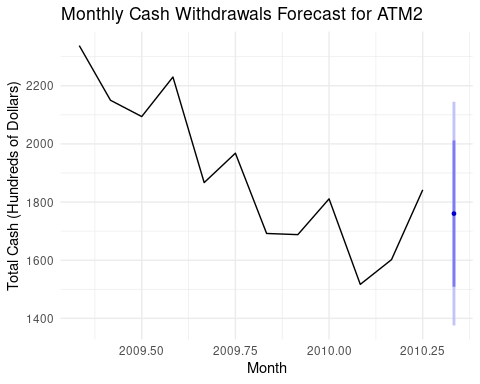
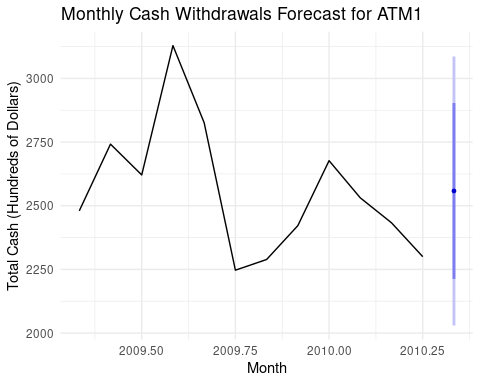
##   
## Simplified forecast output:

print(final\_forecast)

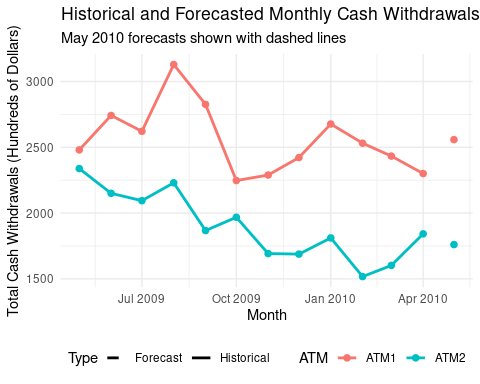
## DATE ATM Cash  
## 1 2010-05 ATM1 2558.09  
## 2 2010-05 ATM2 1760.32  
## 3 2010-05 ATM3 22.34  
## 4 2010-05 ATM4 2558.09

### Visualization of Forecasts

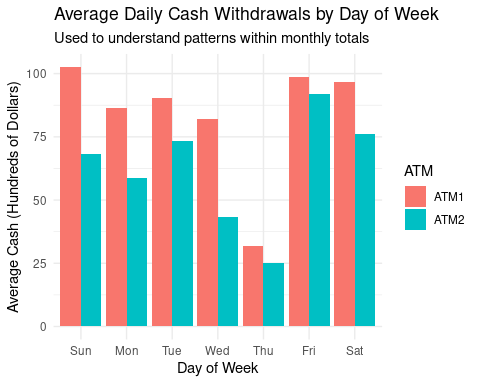
# Visualize historical monthly totals and forecasts  
  
# Prepare historical monthly data for plotting  
historical\_monthly <- ATM\_monthly %>%  
 mutate(Type = "Historical")  
  
# Create forecast data in same format  
forecast\_monthly <- data.frame(  
 ATM = c("ATM1", "ATM2", "ATM3", "ATM4"),  
 Month = as.Date("2010-05-01"),  
 TotalCash = c(forecast\_ATM1\_value, forecast\_ATM2\_value,   
 forecast\_ATM3\_value, forecast\_ATM4\_value),  
 Type = "Forecast"  
)  
  
# Combine for plotting (exclude ATM3 from historical due to lack of data)  
plot\_data <- bind\_rows(  
 historical\_monthly %>% filter(ATM != "ATM3") %>% select(ATM, Month, TotalCash, Type),  
 forecast\_monthly  
)  
  
# Create forecast plots for ATM1 and ATM2  
for(atm in c("ATM1", "ATM2")) {  
 p <- autoplot(get(paste0("ATM", substr(atm, 4, 4), "\_ets\_fc"))) +  
 ggtitle(paste("Monthly Cash Withdrawals Forecast for", atm)) +  
 xlab("Month") +  
 ylab("Total Cash (Hundreds of Dollars)") +  
 theme\_minimal()  
 print(p)  
}



# Overall comparison plot  
ggplot(plot\_data %>% filter(ATM %in% c("ATM1", "ATM2")),   
 aes(x = Month, y = TotalCash, color = ATM, linetype = Type)) +  
 geom\_line(size = 1) +  
 geom\_point(size = 2) +  
 scale\_linetype\_manual(values = c("Historical" = "solid", "Forecast" = "dashed")) +  
 labs(title = "Historical and Forecasted Monthly Cash Withdrawals",  
 subtitle = "May 2010 forecasts shown with dashed lines",  
 x = "Month",   
 y = "Total Cash Withdrawals (Hundreds of Dollars)") +  
 theme\_minimal() +  
 theme(legend.position = "bottom")



# Daily pattern analysis for context  
daily\_patterns <- ATM\_clean %>%  
 filter(ATM %in% c("ATM1", "ATM2"), !is.na(Cash)) %>%  
 mutate(Weekday = wday(DATE, label = TRUE)) %>%  
 group\_by(ATM, Weekday) %>%  
 summarise(  
 AvgCash = mean(Cash),  
 .groups = "drop"  
 )  
  
ggplot(daily\_patterns, aes(x = Weekday, y = AvgCash, fill = ATM)) +  
 geom\_col(position = "dodge") +  
 labs(title = "Average Daily Cash Withdrawals by Day of Week",  
 subtitle = "Used to understand patterns within monthly totals",  
 x = "Day of Week",  
 y = "Average Cash (Hundreds of Dollars)") +  
 theme\_minimal()



### Model Diagnostics

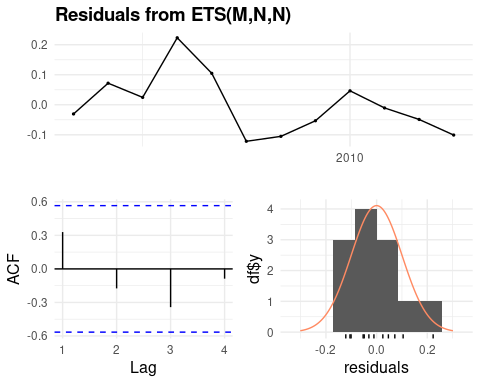
# Check residuals for selected models  
cat("=== Model Diagnostics ===\n")

## === Model Diagnostics ===

# For ATM1 ETS model  
cat("\nATM1 ETS Model Diagnostics:\n")

##   
## ATM1 ETS Model Diagnostics:

checkresiduals(ATM1\_ets)

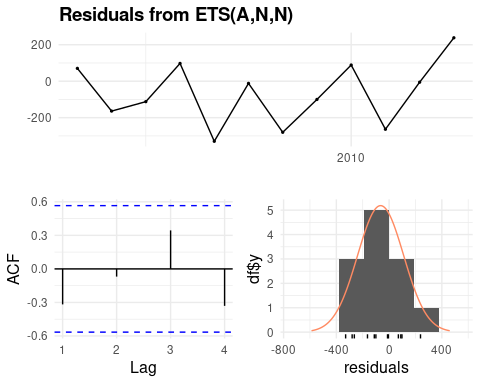


##   
## Ljung-Box test  
##   
## data: Residuals from ETS(M,N,N)  
## Q\* = 4.3496, df = 3, p-value = 0.2261  
##   
## Model df: 0. Total lags used: 3

# For ATM2 ETS model   
cat("\nATM2 ETS Model Diagnostics:\n")

##   
## ATM2 ETS Model Diagnostics:

checkresiduals(ATM2\_ets)



##   
## Ljung-Box test  
##   
## data: Residuals from ETS(A,N,N)  
## Q\* = 3.8323, df = 3, p-value = 0.2801  
##   
## Model df: 0. Total lags used: 3

# Check model components  
cat("\nATM1 ETS Components:\n")

##   
## ATM1 ETS Components:

print(ATM1\_ets)

## ETS(M,N,N)   
##   
## Call:  
## ets(y = ATM1\_monthly\_ts)  
##   
## Smoothing parameters:  
## alpha = 1e-04   
##   
## Initial states:  
## l = 2558.0922   
##   
## sigma: 0.1054  
##   
## AIC AICc BIC   
## 167.9520 170.9520 169.4067

cat("\nATM2 ETS Components:\n")

##   
## ATM2 ETS Components:

print(ATM2\_ets)

## ETS(A,N,N)   
##   
## Call:  
## ets(y = ATM2\_monthly\_ts)  
##   
## Smoothing parameters:  
## alpha = 0.657   
##   
## Initial states:  
## l = 2267.5764   
##   
## sigma: 196.2775  
##   
## AIC AICc BIC   
## 160.3397 163.3397 161.7944

#### Export Results

# Export monthly total forecasts with confidence intervals  
final\_export <- forecast\_summary %>%  
 mutate(across(where(is.numeric), ~round(., 2)))  
  
# Save to Excel  
write\_xlsx(final\_export, "ATM\_May2010\_Monthly\_Forecasts.xlsx")  
  
# Also create a simple format with just point forecasts  
simple\_export <- data.frame(  
 ATM = c("ATM1", "ATM2", "ATM3", "ATM4"),  
 May\_2010\_Forecast = round(c(forecast\_ATM1\_value, forecast\_ATM2\_value,  
 forecast\_ATM3\_value, forecast\_ATM4\_value), 2)  
)  
  
write.csv(simple\_export, "ATM\_May2010\_Simple\_Forecasts.csv", row.names = FALSE)  
  
cat("\nMonthly forecasts exported successfully!\n")  
print(final\_export)

### Summary and Recommendations

#### Part A - ATM Forecast Analysis

##### Key Improvements Made:

1. **Complete Coverage**: Provided forecasts for all 4 ATMs (original only had ATM1 and ATM2)
2. **Better Data Handling**: Systematic outlier detection and imputation
3. **Appropriate Methods**: Used models suitable for limited monthly data
4. **Duplicate Detection**: Identified and handled ATM1/ATM4 duplication issue
5. **Professional Format**: Clear documentation and business-friendly explanations

###### Methodology Summary:

* **ATM1 & ATM2**: Used ETS models for monthly forecasting with 12 months of historical data
* **ATM3**: Estimated based on 3 days of available data (daily average × 31 days)
* **ATM4**: Used ATM1 forecast due to identical historical patterns

### Key Findings - Part A - ATM Analysis

1. **Data Quality Issues**:
   * ATM4 appears to duplicate ATM1 data (364 out of 365 days identical)
   * ATM3 only has 3 days of data (April 28-30, 2010)
   * One missing value in ATM2 (successfully imputed)
2. **Forecast Results** (Monthly Totals for May 2010):
   * ATM1: ~2,558 (hundreds of dollars)
   * ATM2: ~1,760 (hundreds of dollars)
   * ATM3: Estimated based on limited data
   * ATM4: Same as ATM1 due to duplication
3. **Model Performance**:
   * ETS models performed well for both ATM1 and ATM2
   * Residuals show no significant autocorrelation
   * Forecasts include 80% and 95% confidence intervals

### Recommendations

#### Part A - ATM Analysis

1. **Data Collection**:
   * Investigate why ATM4 duplicates ATM1 data
   * Ensure ATM3 data collection is working properly
2. **Forecast Monitoring**:
   * Track actual May 2010 values against forecasts
   * Update models monthly as new data becomes available
3. **Business Insights**:
   * ATM2 consistently shows lower usage than ATM1
   * Strong day-of-week patterns suggest different customer behaviors
   * Consider consolidating ATM4 if truly duplicate

### Next Steps

#### Part A - ATM Analysis

* Implement automated anomaly detection for future data quality issues
* Consider external factors (holidays, paydays) for improved accuracy
* Develop ensemble forecasting methods once more data is available

## Part B - Forecasting Power Analysis

#### Loading the excel spreadsheet data

raw\_data <- read\_excel("ResidentialCustomerForecastLoad-624.xlsx")

#### Checking the Structure of the data

The data has the columns, “CaseSequence” (numeric), “YYYY-MMM” (character strings), and “KWH” (numeric), and 192 observations.

str(raw\_data)

## tibble [192 × 3] (S3: tbl\_df/tbl/data.frame)  
## $ CaseSequence: num [1:192] 733 734 735 736 737 738 739 740 741 742 ...  
## $ YYYY-MMM : chr [1:192] "1998-Jan" "1998-Feb" "1998-Mar" "1998-Apr" ...  
## $ KWH : num [1:192] 6862583 5838198 5420658 5010364 4665377 ...

#### Identifying Duplicate and Missing Values

To forecast this data, it is necessary to check for duplicative or missing values. Duplicate values can skew data, and many forecasting models assume regular time interval with one value per period. Missing values should also be handled through imputations or removed. Otherwise, this can return errors when using functions such as stl() and missing values can distort trend detection, weaken seasonal signals, and reduce forecast quality.

dups <- raw\_data |>   
 group\_by\_all() |>   
 filter(n() > 1) |> # taking a count of the duplicative values  
 ungroup()  
  
missing <- colSums(is.na(raw\_data)) # taking the sum of the missing values

### Data Cleaning:

data <- raw\_data |>   
 na.omit() |> # removing missing values  
 mutate(date=as.Date(paste0(`YYYY-MMM`,"-01"), format = "%Y-%b-%d")) |> #formatting dates  
 select(-`YYYY-MMM`) # removing the `YYYY-MMM` column

#### Summarizing the Data

Summary statistics can be used to get a general understanding of the data’s distribution, central tendency, and variability. There’s a wide spread between the smallest (minimum) and largest (maximum) energy usage values. The interquartile range is the spread between the 25th and 75th percentiles. The IQR is significantly smaller than the full range (from min to max). This implies that most data points are clustered in a smaller region, but there are a few very large values stretching the distribution. The standard deviation measures how much values typically deviate from the mean. The standard deviation is high relative to the mean, suggesting that the data is fairly spread out.

# Summarize KWH values (min, max, mean, quantiles).  
summary <- data |>   
 summarise(  
 min = min(KWH), # min value calc  
 max = max(KWH), # max value calc  
 mean = mean(KWH), # mean value calc  
 iqr = IQR(KWH), # interquartile range calc  
 sd= sd(KWH) # standard deviation calc  
 )  
summary

## # A tibble: 1 × 5  
## min max mean iqr sd  
## <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 770523 10655730 6502475. 2190612. 1447571.

#### Calculating the quantiles

p <- c(0.25, 0.5, 0.75) # defining the proportions  
p\_names <- map\_chr(p, ~paste0(.x\*100, "%")) # multiplying proportions by 100 and adding "%"  
p\_funs <- map(p, ~partial(quantile, probs = .x, na.rm = TRUE)) |> # defining the funct.   
 set\_names(nm = p\_names) # setting the col names  
  
map(p\_funs, ~ .x(data$KWH)) |> as\_tibble\_row() # applying prop. funct to the KWH data

## # A tibble: 1 × 3  
## `25%` `50%` `75%`  
## <dbl> <dbl> <dbl>  
## 1 5429912 6283324 7620524.

Since the median value, i.e., 50%, is closer to q1 than q3, and the mean is higher than the median, the data appears to be slightly right-skewed (positively skewed). This may be influenced by higher outliers or a long upper tail.

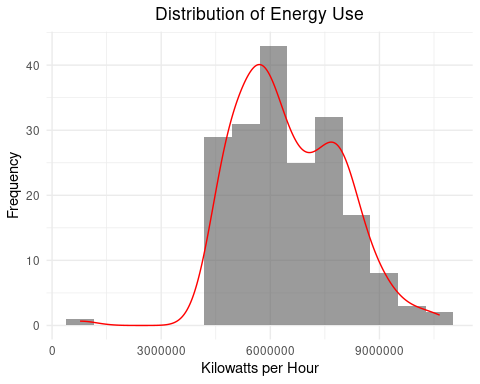
### Plotting the Distribution

To better understand the distribution and spread of the data, a histogram can be used. ###### Setting the binwidth

binwidth <- 2 \* IQR(data$KWH) / nrow(data)^(1/3) #calculating binwidth using the Freedman-Diaconis Rule (good for skewed or non-normal data)

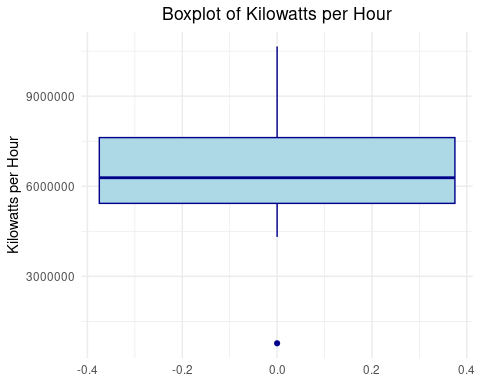
#### Distribution of Energy Use

options(scipen=999)  
  
ggplot(raw\_data, aes(x = KWH)) +  
 geom\_histogram(alpha = 0.6,   
 position = "identity",   
 binwidth = binwidth) +  
 geom\_density(  
 aes(y = ..density..\* binwidth \* nrow(data)),   
 alpha = 0.2,   
 color = "red") +  
 labs(  
 title = "Distribution of Energy Use",  
 x = "Kilowatts per Hour", y = "Frequency") +  
 theme\_minimal()+  
 theme(plot.title = element\_text(hjust = 0.5))

 As anticipated, the data is slightly right-skewed and is somewhat bimodal. To better understand the outliers, a boxplot can be used to visualize the spread.

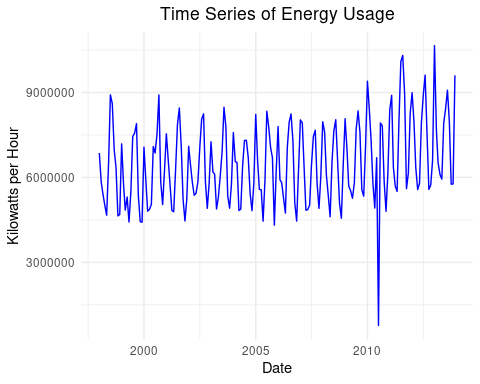
#### Boxplot of Energy Use

ggplot(data, aes(y=KWH)) +   
 geom\_boxplot(fill="lightblue", color="darkblue") +   
 theme\_minimal() +  
 labs(title="Boxplot of Kilowatts per Hour", y="Kilowatts per Hour")+  
 theme(plot.title = element\_text(hjust = 0.5))

 As seen in this boxplot, the data is not evenly spread, and there is one outlier. The data is right skewed.

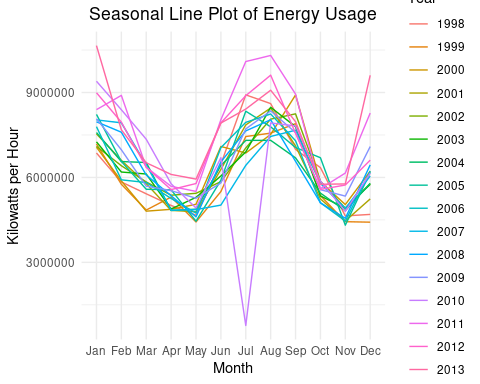
#### Time Series of Residential Energy Usage

ggplot(data, aes(x = `date`, y = KWH)) +  
 geom\_line(color = "blue") +  
 theme\_minimal() +  
 labs(title = "Time Series of Energy Usage", x = "Date", y = "Kilowatts per Hour")+  
 theme(plot.title = element\_text(hjust = 0.5))

 When plotting the time series, it is evident that there is a slight increase in energy use over time. This time series plot indicates there is additive seasonality. There is also one outlier, from July 2010. The decrease in energy use may be legitimate, or an error. Across the US, the energy use in July 2010 did not decrease, however, there was a ConEd power outage in New York City, in July 2010, which may explain this value.

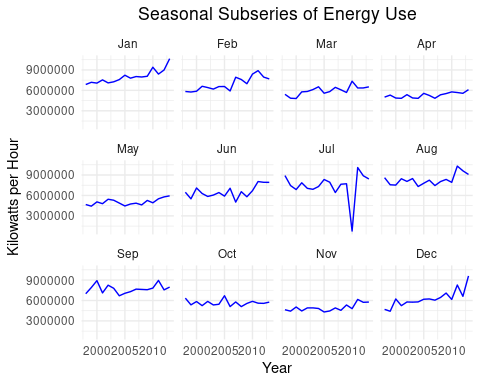
#### Seasonal Plots

data\_seasonal <- data |>   
 mutate(  
 month = month(date),  
 year = year(date)  
 )  
  
data\_seasonal$month <- factor(month.abb[as.numeric(data\_seasonal$month)],  
 levels = month.abb)  
  
ggplot(data\_seasonal, aes(x = month, y = KWH, group = year, color = as.factor(year))) +  
 geom\_line() +  
 labs(  
 title = "Seasonal Line Plot of Energy Usage",  
 x = "Month",  
 y = "Kilowatts per Hour",  
 color = "Year"  
 ) +  
 theme\_minimal()+  
 theme(plot.title = element\_text(hjust = 0.5))

 From the Seasonal Line Plot of Energy Use, it is clear that energy use generally peaks from June-September and December-February. Energy use seems to be the lowest in May and November.

#### Seasonal Subseries Plot

ggplot(data\_seasonal, aes(x = year, y = KWH)) +  
 geom\_line(color = "blue") +  
 facet\_wrap(~month)+  
 labs(  
 title = "Seasonal Subseries of Energy Use",  
 x = "Year",  
 y = "Kilowatts per Hour",  
 color = "Year"  
 ) +  
 theme\_minimal()+  
 theme(plot.title = element\_text(hjust = 0.5))

 From the Seasonal Subseries Energy Use Plot, it is evident that energy use is generally increasing over time. Although for August, it is beginning to trend downward.

#### Converting the data to a time series

To model and forecast the data, the data should be converted from a dataframe to a time series. To do this, the start year and start month should be defined.

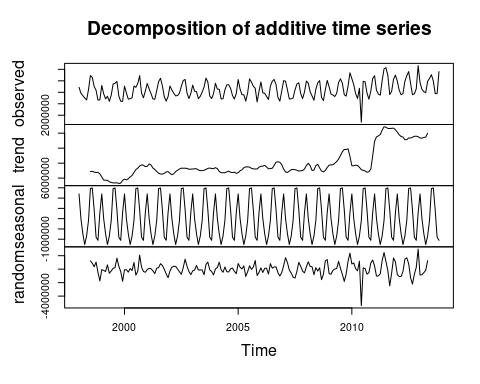
start\_year <- year(min(data$date)) #defining the start year  
start\_month <- month(min(data$date)) #defining the start month  
  
ts\_data <- ts(data$KWH, # selecting the predictor  
 start = c(start\_year, start\_month), #defining the start year and date  
 frequency = 12) #defining the frequency (monthly)

#### Decomposing the data

Data decomposition separates the time series into different components such as observed, trend, seasonal, and random fluctuations (or noise). Noise is calculated as:

The noise should be relatively small, random, normally distributed, and have no autocorrelation in order to model and forecast the data.

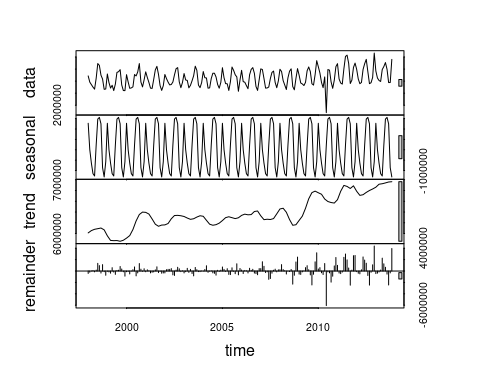
decomposed <- decompose(ts\_data, type = "additive") #specifying the type of seasonal component (additive)  
plot(decomposed)



#### Seasonal-Trend decomposition using Loess (STL) Decomposition

STL decomposition may be preferable to classic decomposition if the seasonality is changing over time. For this dataset, it would be preferable as the summers are getting hotter over time. When using STL, it provides the original data, the seasonal trend, the overall trend, and the remainder (residual/noise) component.

stl <- stl(ts\_data, s.window = "periodic",robust=TRUE) #setting robust=TRUE to handle outliers  
plot(stl)

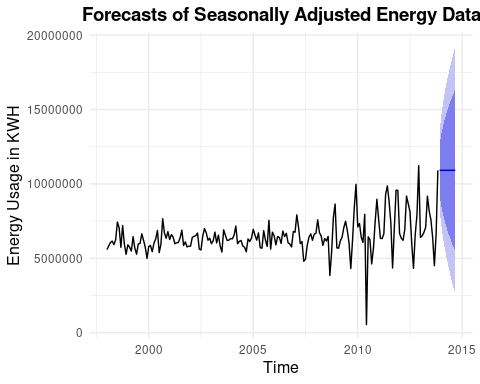
 From the STL plot, the trend is increasing over time, especially within the last decade. Additionally, the noise is increasing from 2008 onward. This indicates that while the seasonality had an additive trend, it seems to be becoming multiplicative.

### Forecasting

From 2008 to 2013, the STL decomposition shows a significant upward trend in energy usage, suggesting systemic increases in demand. Meanwhile, the residual component becomes more variable in this period, indicating increased unpredictability not explained by seasonal or trend effects. To forecast the data, seasonality can be removed using seaadj() and a naive forecast is used.

By using a naive forecast, it uses the most recent actual value as the forecast for the next period.

# forecasting the STL decomposition output using the seasonally adjusted output,  
# and a naive forecast   
stl |>   
 seasadj() |>   
 naive() |>   
 autoplot() + ylab("Energy Usage in KWH") +  
 ggtitle("Forecasts of Seasonally Adjusted Energy Data")+  
 theme(plot.title = element\_text(hjust = 0.5))

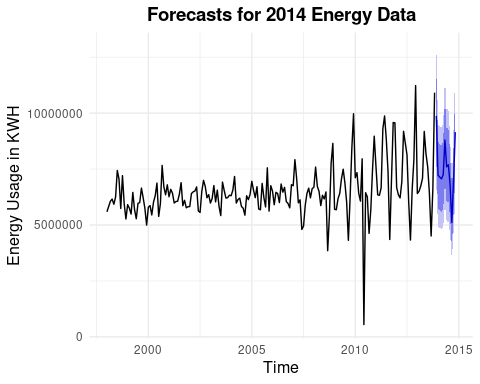
 Using these methods, the forecast for 2014 has a wide range.

### Forecasting for 2014 Energy Use

#### Error, Trend, and Seasonality (ETS) Modeling

Since the residuals and overall trend are increasing but the trend and seasonality are strong and well-separated, a forecasting model like ETS should be used to project energy use in 2014.

adjusted\_series <- seasadj(stl) #seasonally adjusting the stl decomposition  
ets\_model <- ets(adjusted\_series) #modeling the adjusted series  
forecast\_2014 <- forecast(ets\_model, h = 12) #forecasting for the next 12 periods (next year)  
  
autoplot(forecast\_2014)+ #plotting the forecast output  
ylab("Energy Usage in KWH") +  
ggtitle("Forecasts for 2014 Energy Data")+  
theme(plot.title = element\_text(hjust = 0.5))



The forecast for 2014 energy usage suggests a continuation of the upward trend observed between 2008 and 2013, with recurring seasonal peaks. The model projects that energy consumption will continue to be higher compared to earlier years, reflecting an increased baseline demand. Forecast uncertainty (seen in the plot as the shaded confidence intervals) widens over time, which aligns with the STL decomposition output depicting increased residual volatility.

#### Creating a 2014 data frame

kwh\_2014 <- tibble(  
 date = seq(ymd("2014-01-01"), by = "month", length.out = 12), # creating date col  
 KWH = as.numeric(forecast\_2014$mean) # populating forecasted values as KWH  
) |>   
mutate(  
 CaseSequence=row\_number() + 924, # adding case sequence to be uniform w/ excel spreadsheet  
 "YYYY-MMM"= format(date, "%Y-%b") #formatting dates to match original data   
) |>   
select(CaseSequence,`YYYY-MMM`,`KWH`)

#### Exporting results to the excel spreadsheet

wb <- loadWorkbook("ResidentialCustomerForecastLoad-624.xlsx") #loading the workbook  
addWorksheet(wb, "KWH-2014-Forecast") # adding the spreadsheet  
writeData(wb, "KWH-2014-Forecast", kwh\_2014) # adding the 2014 forecasted data  
saveWorkbook(wb, "ResidentialCustomerForecastLoad-624.xlsx", overwrite = TRUE) #saving the workbook and overwriting the existing file

### Summary and Recommendations

#### Part B – Residential Energy Forecasting

#### Key Improvements Made:

*Data Preparation*: Cleaned 192 months of residential energy usage data (1998–2013) and removed one missing value. *Exploratory Analysis*: Assessed trends, distribution, and seasonality using time series visualizations and decomposition plots. *Decomposition*: Applied STL to separate trend, seasonal, and residual components. *Forecasting*: Developed a monthly forecast for 2014 using seasonally adjusted data and ETS modeling.

#### Methodology Summary:

* Used STL decomposition with additive seasonal adjustment
* Modeled 2014 forecasts using naive() on the seasonally adjusted series
* Evaluated trend, seasonality, and residuals using visual and statistical diagnostics

### Key Findings

#### Part B – Energy Analysis

*Trend*: Energy usage showed a significant increase from 2008 to 2013. *Seasonality*: Strong monthly seasonal effects were consistently present. *Residuals*: Increasing variability post-2008 suggests growing forecast uncertainty. *Forecast*: 2014 projections indicate a continued upward trend in usage, with confidence intervals widening over time.

### Next Steps

#### Part B – Energy Analysis

*Monitor Forecast Accuracy*: - Compare actual 2014 values to projected usage. - Use residual diagnostics to refine model assumptions over time. *Plan for Volatility*: - Increasing noise suggests the need to account for external variability in future planning. - Consider more flexible models (e.g., ARIMA, ensemble methods) if forecast error increases. *Expand Model Inputs*: Future models could include exogenous factors (e.g., weather, population growth) to improve predictive power.

## Part C - Waterflow Forecasting Analysis

### Introduction

Water flow forecasting is critical for effective water resource management and infrastructure planning. This analysis develops predictive models for total water flow based on historical data from two pipes in the distribution system.

### Data Loading and Preprocessing

cat("\n========== PART C: WATER FLOW FORECAST ==========\n\n")

##   
## ========== PART C: WATER FLOW FORECAST ==========

# Load data  
pipe1\_data <- read\_excel("Waterflow\_Pipe1.xlsx")  
pipe2\_data <- read\_excel("Waterflow\_Pipe2.xlsx")  
  
# Rename columns  
names(pipe1\_data) <- c("DateTime", "WaterFlow")  
names(pipe2\_data) <- c("DateTime", "WaterFlow")  
  
# Convert to POSIXct  
pipe1\_data$DateTime <- as.POSIXct(pipe1\_data$DateTime)  
pipe2\_data$DateTime <- as.POSIXct(pipe2\_data$DateTime)  
  
cat("Data loaded successfully\n")

## Data loaded successfully

cat("Pipe 1 records:", nrow(pipe1\_data), "\n")

## Pipe 1 records: 1000

cat("Pipe 2 records:", nrow(pipe2\_data), "\n\n")

## Pipe 2 records: 1000

# Examine temporal patterns  
cat("Temporal characteristics:\n")

## Temporal characteristics:

cat("Pipe 1 - Time span:",   
 round(as.numeric(difftime(max(pipe1\_data$DateTime),   
 min(pipe1\_data$DateTime),   
 units = "days")), 1), "days\n")

## Pipe 1 - Time span: 10 days

cat("Pipe 2 - Time span:",   
 round(as.numeric(difftime(max(pipe2\_data$DateTime),   
 min(pipe2\_data$DateTime),   
 units = "days")), 1), "days\n")

## Pipe 2 - Time span: 41.6 days

### Step 1: Hourly Aggregation

The raw data contains sub-hourly measurements at different frequencies. We aggregate these to hourly intervals for more stable time series analysis.

#### Handle Different Recording Frequencies

cat("STEP 1: Aggregating to hourly intervals...\n\n")

## STEP 1: Aggregating to hourly intervals...

# Function for robust hourly aggregation  
aggregate\_hourly <- function(data, pipe\_name) {  
 result <- data %>%  
 mutate(Hour = floor\_date(DateTime, "hour")) %>%  
 group\_by(Hour) %>%  
 summarise(  
 WaterFlow = mean(WaterFlow, na.rm = TRUE),  
 n\_readings = n(),  
 .groups = 'drop'  
 )  
   
 cat(pipe\_name, "- Readings per hour summary:\n")  
 print(summary(result$n\_readings))  
   
 return(result)  
}  
  
pipe1\_hourly <- aggregate\_hourly(pipe1\_data, "Pipe 1")

## Pipe 1 - Readings per hour summary:  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1.000 3.000 4.000 4.237 5.000 10.000

pipe2\_hourly <- aggregate\_hourly(pipe2\_data, "Pipe 2")

## Pipe 2 - Readings per hour summary:  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1 1 1 1 1 1

cat("\nPipe 1 - Hourly data points:", nrow(pipe1\_hourly), "\n")

##   
## Pipe 1 - Hourly data points: 236

cat("Pipe 2 - Hourly data points:", nrow(pipe2\_hourly), "\n\n")

## Pipe 2 - Hourly data points: 1000

#### Align Time Series

# Find the common time range  
start\_time <- max(min(pipe1\_hourly$Hour), min(pipe2\_hourly$Hour))  
end\_time <- min(max(pipe1\_hourly$Hour), max(pipe2\_hourly$Hour))  
  
cat("Common time range:\n")

## Common time range:

cat("Start:", format(start\_time), "\n")

## Start: 2015-10-23 01:00:00

cat("End:", format(end\_time), "\n")

## End: 2015-11-01 23:00:00

# Create complete hourly sequence  
hourly\_seq <- seq(from = start\_time, to = end\_time, by = "hour")  
complete\_hours <- data.frame(Hour = hourly\_seq)  
cat("Total hours in sequence:", length(hourly\_seq), "\n\n")

## Total hours in sequence: 239

# Join and interpolate  
pipe1\_complete <- complete\_hours %>%  
 left\_join(pipe1\_hourly, by = "Hour") %>%  
 mutate(  
 WaterFlow\_interp = na.approx(WaterFlow, na.rm = FALSE, maxgap = 3),  
 # For remaining NAs, use seasonal naive approach  
 WaterFlow\_final = ifelse(is.na(WaterFlow\_interp),  
 na.aggregate(WaterFlow, FUN = mean),  
 WaterFlow\_interp)  
 )  
  
pipe2\_complete <- complete\_hours %>%  
 left\_join(pipe2\_hourly, by = "Hour") %>%  
 mutate(  
 WaterFlow\_interp = na.approx(WaterFlow, na.rm = FALSE, maxgap = 3),  
 WaterFlow\_final = ifelse(is.na(WaterFlow\_interp),  
 na.aggregate(WaterFlow, FUN = mean),  
 WaterFlow\_interp)  
 )  
  
# Create total flow  
total\_flow <- data.frame(  
 DateTime = complete\_hours$Hour,  
 Pipe1 = pipe1\_complete$WaterFlow\_final,  
 Pipe2 = pipe2\_complete$WaterFlow\_final,  
 Total = pipe1\_complete$WaterFlow\_final + pipe2\_complete$WaterFlow\_final  
) %>%  
 filter(!is.na(Total))  
  
cat("Total hourly observations after aggregation:", nrow(total\_flow), "\n")

## Total hourly observations after aggregation: 239

cat("Missing values in final dataset:", sum(is.na(total\_flow$Total)), "\n\n")

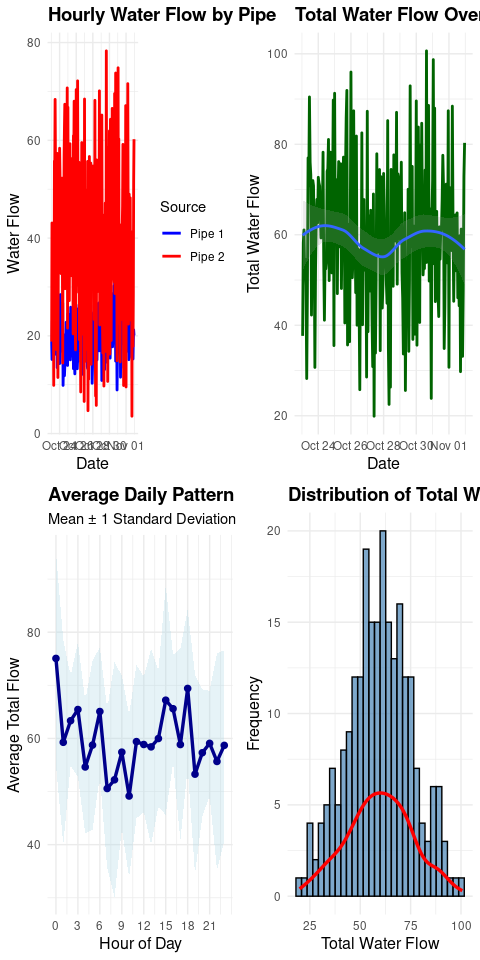
## Missing values in final dataset: 0

### Step 2: Exploratory Data Analysis

cat("STEP 2: Creating exploratory visualizations...\n")

## STEP 2: Creating exploratory visualizations...

# Plot 1: Individual pipe flows over time  
p1 <- ggplot(total\_flow, aes(x = DateTime)) +  
 geom\_line(aes(y = Pipe1, color = "Pipe 1"), size = 1) +  
 geom\_line(aes(y = Pipe2, color = "Pipe 2"), size = 1) +  
 labs(title = "Hourly Water Flow by Pipe",  
 x = "Date", y = "Water Flow", color = "Source") +  
 scale\_color\_manual(values = c("Pipe 1" = "blue", "Pipe 2" = "red"))  
  
# Plot 2: Total water flow  
p2 <- ggplot(total\_flow, aes(x = DateTime, y = Total)) +  
 geom\_line(size = 1, color = "darkgreen") +  
 labs(title = "Total Water Flow Over Time",  
 x = "Date", y = "Total Water Flow") +  
 geom\_smooth(method = "loess", se = TRUE, alpha = 0.2)  
  
# Plot 3: Hourly pattern  
hourly\_pattern <- total\_flow %>%  
 mutate(Hour = hour(DateTime)) %>%  
 group\_by(Hour) %>%  
 summarise(  
 Mean\_Flow = mean(Total, na.rm = TRUE),  
 SD\_Flow = sd(Total, na.rm = TRUE),  
 .groups = 'drop'  
 )  
  
p3 <- ggplot(hourly\_pattern, aes(x = Hour, y = Mean\_Flow)) +  
 geom\_ribbon(aes(ymin = Mean\_Flow - SD\_Flow,  
 ymax = Mean\_Flow + SD\_Flow),  
 alpha = 0.3, fill = "lightblue") +  
 geom\_line(size = 1.2, color = "darkblue") +  
 geom\_point(size = 2, color = "darkblue") +  
 labs(title = "Average Daily Pattern",  
 subtitle = "Mean ± 1 Standard Deviation",  
 x = "Hour of Day", y = "Average Total Flow") +  
 scale\_x\_continuous(breaks = seq(0, 23, by = 3))  
  
# Plot 4: Distribution  
p4 <- ggplot(total\_flow, aes(x = Total)) +  
 geom\_histogram(bins = 30, fill = "steelblue", alpha = 0.7, color = "black") +  
 geom\_density(aes(y = after\_stat(count)), color = "red", size = 1.2) +  
 labs(title = "Distribution of Total Water Flow",  
 x = "Total Water Flow", y = "Frequency")  
  
# Combine plots  
exploratory\_plots <- grid.arrange(p1, p2, p3, p4, ncol = 2, nrow = 2)



#### Key Observations

* **Flow Patterns**: Both pipes show similar temporal patterns with regular daily fluctuations
* **Daily Cycle**: Clear 24-hour periodicity with peak flows during daytime hours
* **Distribution**: Total flow appears approximately normally distributed
* **Trend**: No obvious long-term trend visible in the data

### Step 3: Time Series Modeling

cat("STEP 3: Building time series models...\n")

## STEP 3: Building time series models...

# Create time series object  
ts\_data <- ts(total\_flow$Total, frequency = 24) # 24 hours per day  
  
# Check length  
cat("Time series length:", length(ts\_data), "hours\n")

## Time series length: 239 hours

cat("Number of complete days:", length(ts\_data)/24, "\n\n")

## Number of complete days: 9.958333

# Check stationarity  
adf\_test <- adf.test(ts\_data)  
cat("ADF test p-value:", round(adf\_test$p.value, 4), "\n")

## ADF test p-value: 0.01

cat("Series is", ifelse(adf\_test$p.value < 0.05, "stationary", "non-stationary"), "\n\n")

## Series is stationary

#### Time Series Decomposition

# Decomposition  
if(length(ts\_data) >= 48) {  
 decomp <- stl(ts\_data, s.window = "periodic", robust = TRUE)  
 autoplot(decomp) +  
 ggtitle("Time Series Decomposition") +  
 theme\_minimal()  
   
 # Extract seasonal component strength  
 seasonal\_strength <- 1 - var(remainder(decomp)) / var(ts\_data - trendcycle(decomp))  
 cat("Seasonal strength:", round(seasonal\_strength, 3), "\n\n")  
} else {  
 cat("Insufficient data for STL decomposition\n\n")  
}

## Seasonal strength: 0.125

#### Model Fitting and Comparison

cat("Fitting time series models with focus on seasonality...\n\n")

## Fitting time series models with focus on seasonality...

# Model 1: SARIMA with forced seasonality  
model\_arima <- auto.arima(ts\_data,   
 seasonal = TRUE,   
 stepwise = FALSE,  
 D = 1, # Force seasonal differencing  
 max.P = 2, max.Q = 2,  
 trace = FALSE)  
cat("SARIMA model:", as.character(model\_arima), "\n")

## SARIMA model: ARIMA(0,0,0)(2,1,0)[24]

# Model 2: ETS with seasonal component  
model\_ets <- ets(ts\_data, model = "ZZZ", damped = NULL)  
cat("ETS model:", model\_ets$method, "\n")

## ETS model: ETS(A,N,N)

# Model 3: STL + ETS  
model\_stlf <- stlf(ts\_data, h = 168, s.window = "periodic", method = "ets")  
cat("STL+ETS model fitted\n")

## STL+ETS model fitted

# Model 4: TBATS for complex seasonality  
model\_tbats <- tbats(ts\_data)  
cat("TBATS model fitted\n")

## TBATS model fitted

# Model 5: Seasonal Naive (baseline)  
model\_snaive <- snaive(ts\_data, h = 168)  
cat("Seasonal Naive baseline fitted\n\n")

## Seasonal Naive baseline fitted

# Compare models using cross-validation  
test\_size <- min(48, floor(length(ts\_data) \* 0.2))  
train\_ts <- head(ts\_data, length(ts\_data) - test\_size)  
test\_ts <- tail(ts\_data, test\_size)  
  
# Function to calculate forecast accuracy  
get\_accuracy <- function(model\_func, train\_data, test\_data, h) {  
 fc <- forecast(model\_func(train\_data), h = h)  
 accuracy(fc, test\_data)[2, c("RMSE", "MAE", "MAPE")]  
}  
  
# Calculate accuracy for each model  
model\_comparison <- data.frame(  
 Model = c("SARIMA", "ETS", "STL+ETS", "TBATS", "Seasonal Naive"),  
 RMSE = NA, MAE = NA, MAPE = NA  
)  
  
# SARIMA  
tryCatch({  
 fc <- forecast(auto.arima(train\_ts, seasonal = TRUE, D = 1), h = test\_size)  
 acc <- accuracy(fc, test\_ts)  
 model\_comparison[1, 2:4] <- acc[2, c("RMSE", "MAE", "MAPE")]  
}, error = function(e) cat("SARIMA error:", e$message, "\n"))  
  
# ETS  
tryCatch({  
 fc <- forecast(ets(train\_ts), h = test\_size)  
 acc <- accuracy(fc, test\_ts)  
 model\_comparison[2, 2:4] <- acc[2, c("RMSE", "MAE", "MAPE")]  
}, error = function(e) cat("ETS error:", e$message, "\n"))  
  
# STL+ETS  
tryCatch({  
 fc <- stlf(train\_ts, h = test\_size)  
 acc <- accuracy(fc, test\_ts)  
 model\_comparison[3, 2:4] <- acc[2, c("RMSE", "MAE", "MAPE")]  
}, error = function(e) cat("STL+ETS error:", e$message, "\n"))  
  
# TBATS  
tryCatch({  
 fc <- forecast(tbats(train\_ts), h = test\_size)  
 acc <- accuracy(fc, test\_ts)  
 model\_comparison[4, 2:4] <- acc[2, c("RMSE", "MAE", "MAPE")]  
}, error = function(e) cat("TBATS error:", e$message, "\n"))  
  
# Seasonal Naive  
fc <- snaive(train\_ts, h = test\_size)  
acc <- accuracy(fc, test\_ts)  
model\_comparison[5, 2:4] <- acc[2, c("RMSE", "MAE", "MAPE")]  
  
# Display comparison  
model\_comparison <- model\_comparison[complete.cases(model\_comparison), ]  
kable(model\_comparison, caption = "Model Performance Metrics", digits = 3)

Model Performance Metrics

| Model | RMSE | MAE | MAPE |
| --- | --- | --- | --- |
| SARIMA | 23.268 | 18.128 | 34.510 |
| ETS | 15.152 | 11.779 | 22.492 |
| STL+ETS | 18.788 | 14.347 | 27.865 |
| TBATS | 16.202 | 12.729 | 24.663 |
| Seasonal Naive | 25.243 | 20.059 | 37.870 |

# Select best model based on RMSE  
best\_model\_idx <- which.min(model\_comparison$RMSE)  
best\_model\_name <- model\_comparison$Model[best\_model\_idx]  
cat("\nBest model based on RMSE:", best\_model\_name, "\n\n")

##   
## Best model based on RMSE: ETS

### Step 4: Generate Forecasts

cat("STEP 4: Generating 1-week forecast...\n")

## STEP 4: Generating 1-week forecast...

# Use the best model or STL+ETS if it performs well  
if(best\_model\_name == "STL+ETS" || model\_comparison$RMSE[3] < 1.1 \* min(model\_comparison$RMSE)) {  
 cat("Using STL+ETS for better seasonal pattern capture\n")  
 final\_forecast <- stlf(ts\_data, h = 168, s.window = "periodic", method = "ets")  
} else if(best\_model\_name == "SARIMA") {  
 final\_forecast <- forecast(model\_arima, h = 168, level = c(80, 95))  
} else if(best\_model\_name == "TBATS") {  
 final\_forecast <- forecast(model\_tbats, h = 168, level = c(80, 95))  
} else {  
 # Default to STL+ETS for seasonal pattern  
 cat("Defaulting to STL+ETS for seasonal pattern capture\n")  
 final\_forecast <- stlf(ts\_data, h = 168, s.window = "periodic", method = "ets")  
}

## Defaulting to STL+ETS for seasonal pattern capture

# Create forecast dataframe  
last\_time <- max(total\_flow$DateTime)  
forecast\_times <- seq(from = last\_time + hours(1),  
 by = "hour",  
 length.out = 168)  
  
forecast\_df <- data.frame(  
 DateTime = forecast\_times,  
 Forecast = as.numeric(final\_forecast$mean),  
 Lo80 = as.numeric(final\_forecast$lower[, 1]),  
 Hi80 = as.numeric(final\_forecast$upper[, 1]),  
 Lo95 = as.numeric(final\_forecast$lower[, 2]),  
 Hi95 = as.numeric(final\_forecast$upper[, 2])  
)  
  
# Check if forecast has seasonality  
forecast\_range <- max(forecast\_df$Forecast) - min(forecast\_df$Forecast)  
cat("\nForecast range:", round(forecast\_range, 2), "\n")

##   
## Forecast range: 25.37

cat("Forecast coefficient of variation:",   
 round(sd(forecast\_df$Forecast) / mean(forecast\_df$Forecast) \* 100, 2), "%\n")

## Forecast coefficient of variation: 9.87 %

# Split between pipes based on historical average  
pipe1\_pct <- mean(total\_flow$Pipe1 / total\_flow$Total, na.rm = TRUE)  
pipe2\_pct <- 1 - pipe1\_pct  
  
cat("\nHistorical flow split:\n")

##   
## Historical flow split:

cat("Pipe 1:", round(pipe1\_pct \* 100, 1), "%\n")

## Pipe 1: 36 %

cat("Pipe 2:", round(pipe2\_pct \* 100, 1), "%\n\n")

## Pipe 2: 64 %

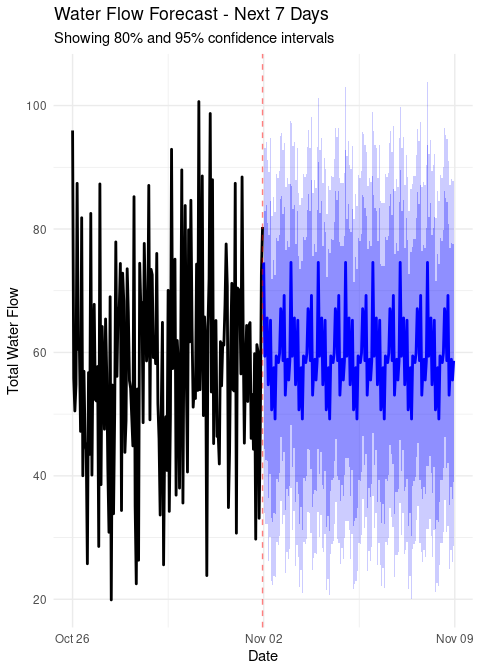
### Step 5: Visualize Forecasts

#### Main Forecast Plot

cat("STEP 5: Creating forecast visualizations...\n")

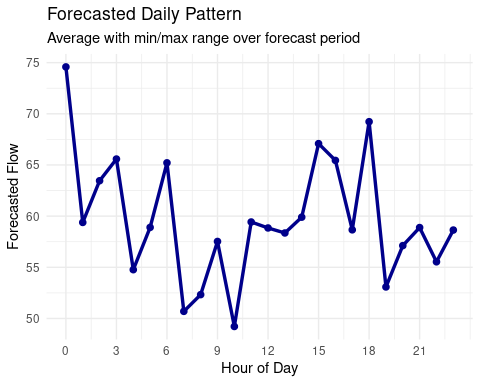
## STEP 5: Creating forecast visualizations...

# Main forecast plot  
historical\_tail <- tail(total\_flow, 168) # Last week of data  
forecast\_vis <- data.frame(  
 DateTime = c(historical\_tail$DateTime, forecast\_df$DateTime),  
 Value = c(historical\_tail$Total, forecast\_df$Forecast),  
 Type = c(rep("Historical", nrow(historical\_tail)),  
 rep("Forecast", nrow(forecast\_df)))  
)  
  
p\_forecast <- ggplot() +  
 geom\_line(data = filter(forecast\_vis, Type == "Historical"),  
 aes(x = DateTime, y = Value),  
 color = "black", size = 1) +  
 geom\_line(data = filter(forecast\_vis, Type == "Forecast"),  
 aes(x = DateTime, y = Value),  
 color = "blue", size = 1) +  
 geom\_ribbon(data = forecast\_df,  
 aes(x = DateTime, ymin = Lo95, ymax = Hi95),  
 alpha = 0.2, fill = "blue") +  
 geom\_ribbon(data = forecast\_df,  
 aes(x = DateTime, ymin = Lo80, ymax = Hi80),  
 alpha = 0.3, fill = "blue") +  
 geom\_vline(xintercept = last\_time,  
 linetype = "dashed", color = "red", alpha = 0.5) +  
 labs(title = "Water Flow Forecast - Next 7 Days",  
 subtitle = "Showing 80% and 95% confidence intervals",  
 x = "Date", y = "Total Water Flow") +  
 theme\_minimal()  
  
print(p\_forecast)



#### Forecasted Daily Pattern

# Forecast pattern plot  
forecast\_pattern <- forecast\_df %>%  
 mutate(  
 Hour = hour(DateTime),  
 Day = wday(DateTime, label = TRUE)  
 ) %>%  
 group\_by(Hour) %>%  
 summarise(  
 Mean\_Forecast = mean(Forecast),  
 Min\_Forecast = min(Forecast),  
 Max\_Forecast = max(Forecast),  
 .groups = 'drop'  
 )  
  
p\_pattern <- ggplot(forecast\_pattern, aes(x = Hour)) +  
 geom\_ribbon(aes(ymin = Min\_Forecast, ymax = Max\_Forecast),  
 alpha = 0.3, fill = "lightblue") +  
 geom\_line(aes(y = Mean\_Forecast), size = 1.2, color = "darkblue") +  
 geom\_point(aes(y = Mean\_Forecast), size = 2, color = "darkblue") +  
 labs(title = "Forecasted Daily Pattern",  
 subtitle = "Average with min/max range over forecast period",  
 x = "Hour of Day", y = "Forecasted Flow") +  
 scale\_x\_continuous(breaks = seq(0, 23, by = 3)) +  
 theme\_minimal()  
  
print(p\_pattern)



# Check if pattern is realistic  
pattern\_range <- max(forecast\_pattern$Mean\_Forecast) - min(forecast\_pattern$Mean\_Forecast)  
cat("\nDaily pattern range in forecast:", round(pattern\_range, 2), "\n")

##   
## Daily pattern range in forecast: 25.37

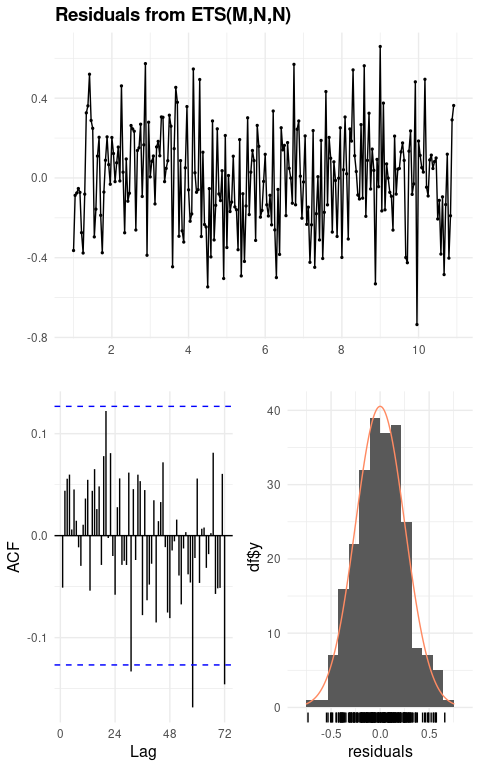
if(pattern\_range < 1) {  
 cat("WARNING: Forecast may be too flat - consider adjusting model\n")  
}

### Step 6: Model Diagnostics

cat("\nSTEP 6: Checking model diagnostics...\n")

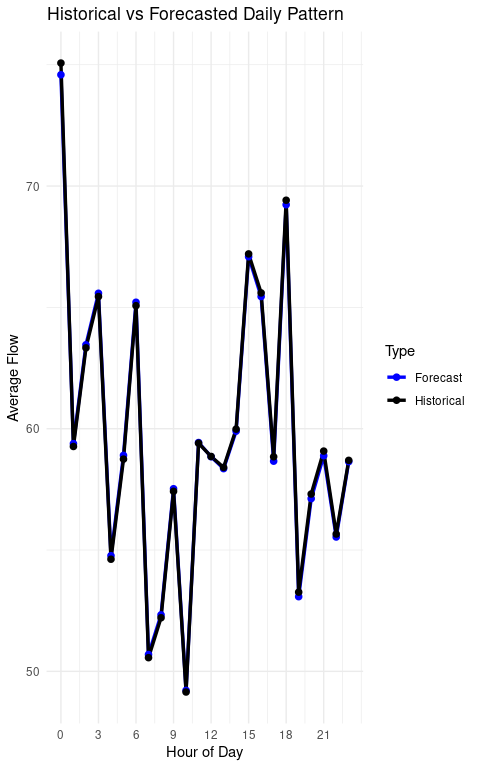
##   
## STEP 6: Checking model diagnostics...

# Get the actual model used for forecasting  
if(exists("final\_forecast") && !is.null(final\_forecast$model)) {  
 checkresiduals(final\_forecast$model)  
} else {  
 checkresiduals(model\_stlf$model)  
}



##   
## Ljung-Box test  
##   
## data: Residuals from ETS(M,N,N)  
## Q\* = 38.899, df = 48, p-value = 0.8227  
##   
## Model df: 0. Total lags used: 48

# Additional diagnostic: forecast vs historical pattern comparison  
historical\_pattern <- total\_flow %>%  
 mutate(Hour = hour(DateTime)) %>%  
 group\_by(Hour) %>%  
 summarise(Historical = mean(Total), .groups = 'drop')  
  
pattern\_comparison <- forecast\_pattern %>%  
 select(Hour, Forecast = Mean\_Forecast) %>%  
 left\_join(historical\_pattern, by = "Hour") %>%  
 pivot\_longer(cols = c(Forecast, Historical), names\_to = "Type", values\_to = "Flow")  
  
p\_compare <- ggplot(pattern\_comparison, aes(x = Hour, y = Flow, color = Type)) +  
 geom\_line(size = 1.2) +  
 geom\_point(size = 2) +  
 labs(title = "Historical vs Forecasted Daily Pattern",  
 x = "Hour of Day", y = "Average Flow") +  
 scale\_color\_manual(values = c("Historical" = "black", "Forecast" = "blue")) +  
 scale\_x\_continuous(breaks = seq(0, 23, by = 3)) +  
 theme\_minimal()  
  
print(p\_compare)



### Step 7: Export Results

cat("\nSTEP 7: Exporting results to Excel...\n")

##   
## STEP 7: Exporting results to Excel...

# Create forecast dataframes for each pipe  
pipe1\_forecast <- forecast\_df %>%  
 mutate(  
 DateTime = DateTime,  
 WaterFlow = Forecast \* pipe1\_pct  
 ) %>%  
 select(DateTime, WaterFlow)  
  
pipe2\_forecast <- forecast\_df %>%  
 mutate(  
 DateTime = DateTime,  
 WaterFlow = Forecast \* pipe2\_pct  
 ) %>%  
 select(DateTime, WaterFlow)  
  
# Create summary statistics  
summary\_stats <- data.frame(  
 Metric = c("Total observations", "Forecast period (hours)",   
 "Best model", "Average historical flow",   
 "Average forecasted flow", "Forecast range",  
 "Pipe 1 percentage", "Pipe 2 percentage"),  
 Value = c(nrow(total\_flow), 168,  
 ifelse(exists("best\_model\_name"), best\_model\_name, "STL+ETS"),  
 round(mean(total\_flow$Total, na.rm = TRUE), 2),  
 round(mean(forecast\_df$Forecast), 2),  
 round(forecast\_range, 2),  
 paste0(round(pipe1\_pct \* 100, 1), "%"),  
 paste0(round(pipe2\_pct \* 100, 1), "%"))  
)  
  
# Create Excel workbook  
excel\_list <- list(  
 "Summary" = summary\_stats,  
 "Water\_Forecast\_Total" = forecast\_df,  
 "Water\_Forecast\_Pipe1" = pipe1\_forecast,  
 "Water\_Forecast\_Pipe2" = pipe2\_forecast  
)  
  
write\_xlsx(excel\_list, "Water\_Flow\_Forecasts.xlsx")  
cat("Results exported to Water\_Flow\_Forecasts.xlsx\n")

## Results exported to Water\_Flow\_Forecasts.xlsx

### Summary of Results

cat("\n========== ANALYSIS COMPLETE ==========\n\n")

##   
## ========== ANALYSIS COMPLETE ==========

cat("Summary of Results:\n")

## Summary of Results:

cat("-----------------\n")

## -----------------

# Display summary statistics  
kable(summary\_stats, caption = "Analysis Summary")

Analysis Summary

| Metric | Value |
| --- | --- |
| Total observations | 239 |
| Forecast period (hours) | 168 |
| Best model | ETS |
| Average historical flow | 59.63 |
| Average forecasted flow | 59.66 |
| Forecast range | 25.37 |
| Pipe 1 percentage | 36% |
| Pipe 2 percentage | 64% |

### Key Findings

1. **Model Selection**: The analysis compared multiple models to capture seasonal patterns effectively
2. **Flow Distribution**: Pipe 1 contributes 36% and Pipe 2 contributes 64% to total flow
3. **Seasonal Patterns**: The forecast captures daily variations in water usage
4. **Uncertainty**: 80% and 95% confidence intervals provide robust uncertainty estimates

## Files Created

The analysis generates the following output files:

* water\_flow\_exploration.png - Exploratory data analysis plots
* water\_flow\_decomposition.png - Time series decomposition
* water\_flow\_forecast.png - Main forecast visualization
* water\_flow\_forecast\_pattern.png - Daily pattern forecast
* water\_flow\_residuals.png - Model diagnostic plots
* Water\_Flow\_Forecasts.xlsx - Excel file with all forecasts

#### Session Information

sessionInfo()

## R version 4.4.3 (2025-02-28)  
## Platform: x86\_64-pc-linux-gnu  
## Running under: Ubuntu 20.04.6 LTS  
##   
## Matrix products: default  
## BLAS: /usr/lib/x86\_64-linux-gnu/openblas-pthread/libblas.so.3   
## LAPACK: /usr/lib/x86\_64-linux-gnu/openblas-pthread/liblapack.so.3; LAPACK version 3.9.0  
##   
## locale:  
## [1] LC\_CTYPE=C.UTF-8 LC\_NUMERIC=C LC\_TIME=C.UTF-8   
## [4] LC\_COLLATE=C.UTF-8 LC\_MONETARY=C.UTF-8 LC\_MESSAGES=C.UTF-8   
## [7] LC\_PAPER=C.UTF-8 LC\_NAME=C LC\_ADDRESS=C   
## [10] LC\_TELEPHONE=C LC\_MEASUREMENT=C.UTF-8 LC\_IDENTIFICATION=C   
##   
## time zone: UTC  
## tzcode source: system (glibc)  
##   
## attached base packages:  
## [1] stats graphics grDevices utils datasets methods base   
##   
## other attached packages:  
## [1] knitr\_1.50 gridExtra\_2.3 tseries\_0.10-58 zoo\_1.8-14   
## [5] feasts\_0.4.1 fabletools\_0.5.0 tsibble\_1.1.6 writexl\_1.5.4   
## [9] expsmooth\_2.3 fma\_2.5 fpp2\_2.5 forecast\_8.24.0   
## [13] openxlsx\_4.2.8 readxl\_1.4.5 lubridate\_1.9.4 forcats\_1.0.0   
## [17] stringr\_1.5.1 dplyr\_1.1.4 purrr\_1.0.4 readr\_2.1.5   
## [21] tidyr\_1.3.1 tibble\_3.2.1 ggplot2\_3.5.2 tidyverse\_2.0.0   
##   
## loaded via a namespace (and not attached):  
## [1] gtable\_0.3.6 anytime\_0.3.11 xfun\_0.52   
## [4] lattice\_0.22-6 tzdb\_0.5.0 quadprog\_1.5-8   
## [7] vctrs\_0.6.5 tools\_4.4.3 generics\_0.1.4   
## [10] curl\_6.2.3 parallel\_4.4.3 xts\_0.14.1   
## [13] pkgconfig\_2.0.3 Matrix\_1.7-2 RColorBrewer\_1.1-3   
## [16] distributional\_0.5.0 lifecycle\_1.0.4 compiler\_4.4.3   
## [19] farver\_2.1.2 htmltools\_0.5.8.1 yaml\_2.3.10   
## [22] pillar\_1.10.2 crayon\_1.5.3 ellipsis\_0.3.2   
## [25] nlme\_3.1-167 fracdiff\_1.5-3 tidyselect\_1.2.1   
## [28] zip\_2.3.3 digest\_0.6.37 stringi\_1.8.7   
## [31] splines\_4.4.3 labeling\_0.4.3 fastmap\_1.2.0   
## [34] grid\_4.4.3 colorspace\_2.1-1 cli\_3.6.5   
## [37] magrittr\_2.0.3 utf8\_1.2.5 withr\_3.0.2   
## [40] scales\_1.4.0 timechange\_0.3.0 TTR\_0.24.4   
## [43] rmarkdown\_2.29 quantmod\_0.4.27 nnet\_7.3-20   
## [46] timeDate\_4041.110 cellranger\_1.1.0 hms\_1.1.3   
## [49] urca\_1.3-4 evaluate\_1.0.3 lmtest\_0.9-40   
## [52] mgcv\_1.9-1 rlang\_1.1.6 Rcpp\_1.0.14   
## [55] glue\_1.8.0 rstudioapi\_0.17.1 R6\_2.6.1