Modeling Household Electricity Demand

Xin Guan, Vera Hudak, Yuqi Zhang

2024-05-24

```
library(electBook)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(proxy)
```

```
##
## Attaching package: 'proxy'
```

```
## The following objects are masked from 'package:stats':
##
##     as.dist, dist
```

```
## The following object is masked from 'package:base':
##
##     as.matrix
```

```
library(tibble)
library(ggplot2)
library(bookdown)
library(rjags)
```

```
## Loading required package: coda
```

```
## Linked to JAGS 4.3.2
```

```
## Loaded modules: basemod,bugs
```

Introduction

Electricity demand forecasting is crucial for energy management, planning, and policy-making. In this report, we explore advanced statistical techniques to model and predict household electricity demand using a dataset of Irish residential consumers. The dataset, provided by the CER Smart Metering Project, includes half-hourly electricity consumption data and survey information from over 2000 households. Given the extensive nature of the data, a primary challenge is to effectively aggregate and analyse it to extract meaningful insights and make accurate predictions.

Our approach involves the following steps:

1. **Exploratory Data Analysis (EDA):** We begin with a thorough exploratory analysis of the dataset to understand the distribution and variability of electricity demand across different households and identify key patterns and trends. This step helps in detecting outliers, understanding the data structure, and selecting relevant features for modelling.
2. **Aggregation:** To manage the large dataset, we use an aggregation method to group households based on daily demand profile similarity. This step reduces the dimensionality of the data and enables us to focus on representative groups, making the modelling process more efficient.
3. **Prediction:** We develop a Bayesian time series regression model that incorporates both temporal dependencies and non-linear relationships between electricity demand and explanatory variables. Bayesian methods offer a robust framework for parameter estimation and uncertainty quantification, providing comprehensive insights into demand patterns.
4. **Evaluation:** The predictive performance of our model is evaluated using appropriate metrics to ensure its accuracy and reliability. We compare the predicted values against the actual data to assess the model's effectiveness in capturing the underlying demand patterns.

This report aims to demonstrate the application of advanced Bayesian techniques in electricity demand forecasting and highlight the benefits of data aggregation in handling large datasets. By leveraging Bayesian methods, we aim to contribute to the development of more adaptive and reliable forecasting models that can meet the evolving challenges of more complicated dataset.

Explanatory Data Analysis

Loading Data:

```r
load("Irish.RData")
```

The file `Irish` contains three data sets, the data set`Irish$indCons` is the electricity demand for 2671 household, with a resolution of half hour over the time period 2019-12-29 to 2020-12-29.

```r
head(Irish$indCons[,1:10])
```

```
##      I1002 I1003 I1004 I1005 I1013 I1015 I1018 I1020 I1022 I1024
## 8114 0.022 0.593 2.002 0.755 0.035 0.398 0.547 0.376 0.229 1.030
## 8115 0.133 0.707 1.602 0.898 0.112 0.689 0.603 0.275 0.198 0.807
## 8116 0.094 0.684 1.525 0.736 0.046 0.407 0.511 0.259 0.201 0.859
## 8117 0.023 0.563 1.393 0.738 0.036 0.223 0.593 0.249 0.212 0.210
## 8118 0.133 0.489 1.221 0.849 0.065 0.132 0.570 0.241 0.121 0.056
## 8119 0.090 0.521 1.032 0.695 0.093 0.117 0.481 0.122 0.127 0.169
```

The data set `Irish$extra` contains variables such as `tod` (time of day), `toy` (time of year), `dow` (day of the week), and more.

```r
head(Irish$extra)
```

```
##   time       toy dow  holy tod temp            dateTime
## 1    1 0.9863014 Wed FALSE   0    4 2009-12-29 23:00:00
## 2    2 0.9863014 Wed FALSE   1    4 2009-12-29 23:30:00
## 3    3 0.9863014 Wed FALSE   2    4 2009-12-30 00:00:00
## 4    4 0.9863014 Wed FALSE   3    4 2009-12-30 00:30:00
## 5    5 0.9863014 Wed FALSE   4    4 2009-12-30 01:00:00
## 6    6 0.9863014 Wed FALSE   5    4 2009-12-30 01:30:00
```

The data set `Irish$survey` contains responses from a household survey, including variables such as SOCIALCLASS, OWNERSHIP, BUILT.YEAR, and others. The variables in this data frame doesn't show any significant relationship with demand, hence we will not use this data in our project.

```r
head(Irish$survey)
```

```
##      ID   meanDem SOCIALCLASS OWNERSHIP BUILT.YEAR HEAT.HOME HEAT.WATER
## 1 I1002 0.2081436          DE         O       1975     Other       Elec
## 2 I1003 0.6215765          C1         O       2004     Other      Other
## 3 I1004 0.9617103          C1         O       1987     Other       Elec
## 4 I1005 0.6402214          C1         O       1930     Other      Other
## 5 I1013 0.2414805          C2         O       2003     Other       Elec
## 6 I1015 0.4631413          DE         R       1989      Elec      Other
##   WINDOWS.doubleglazed HOME.APPLIANCE..White.goods. Code ResTariffallocation
## 1                  All                            1    1                   E
## 2                  All                            5    1                   A
## 3                  All                            5    1                   A
## 4                  All                            4    1                   D
## 5                  All                            3    1                   D
## 6                  All                            2    1                   C
##   ResStimulusallocation
## 1                     E
## 2                     4
```

```
## 3                    2
## 4                    4
## 5                    4
## 6                    3
```

Count zeros in dataset

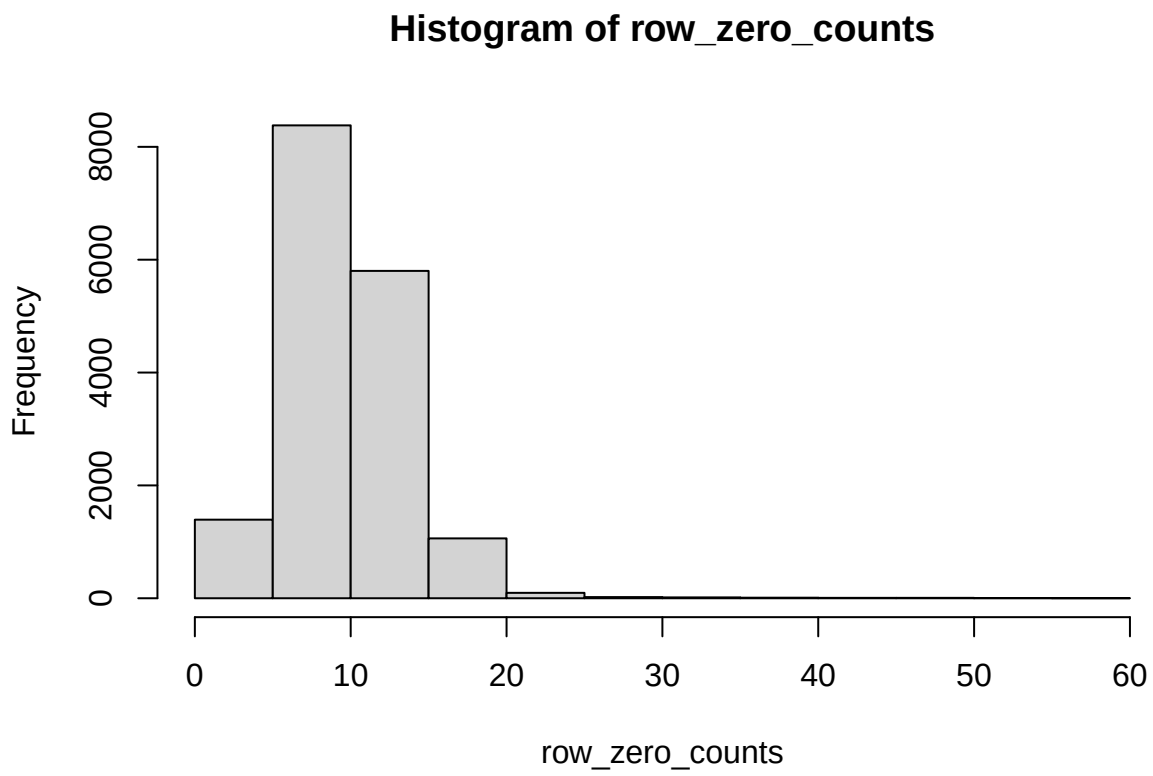
```
sum(apply(Irish$indCons, 2, function(x) sum(x == 0)))
```

```
## [1] 170228
```

```
print(16799*2674)
```

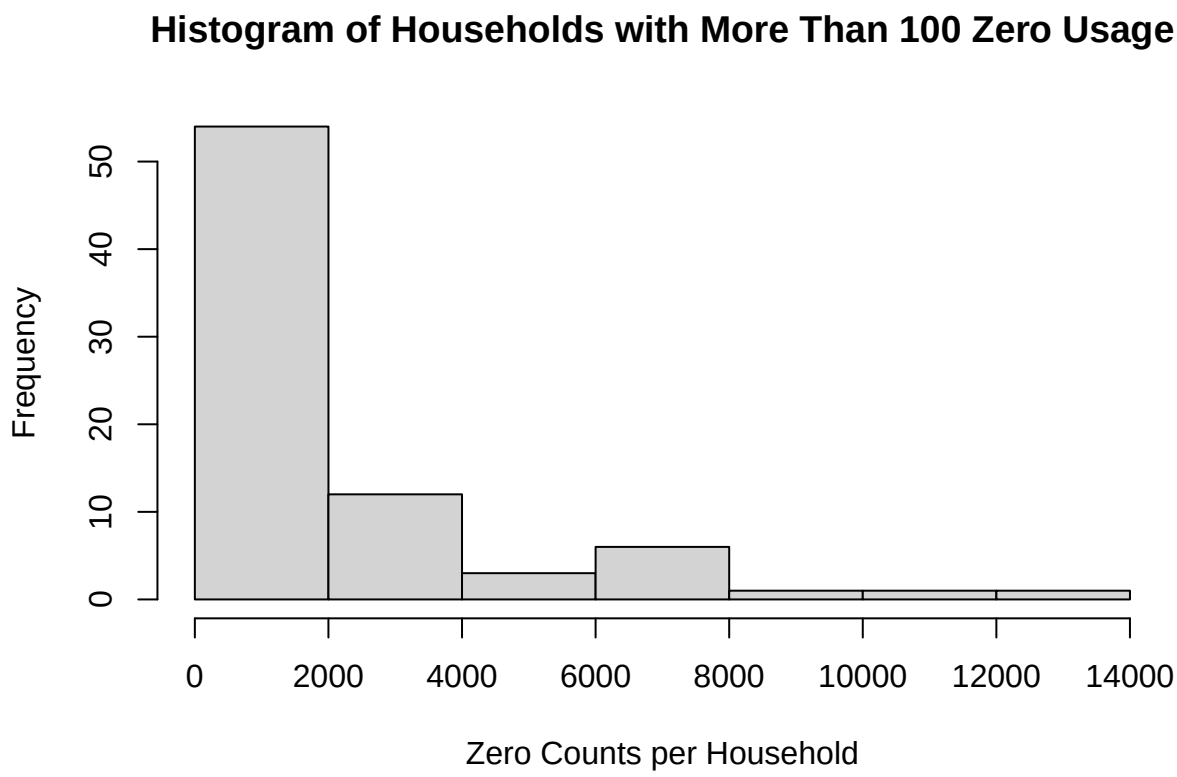
```
## [1] 44920526
```

```
row_zero_counts <- rowSums(Irish$indCons == 0)

hist(row_zero_counts)
```



There are more than 17,000 zero values in the dataset. Summing over the rows for each time point, there is no any time point have significantly large number of zeros.

```r
# Count zeros in each column
col_zero_counts <- colSums(Irish$indCons == 0)

# Histogram of columns with more than 100 zeros
hist(col_zero_counts[col_zero_counts > 100],
     main="Histogram of Households with More Than 100 Zero Usage",
     xlab= "Zero Counts per Household")
```




```r
sum(col_zero_counts > 30*48)
```

```
## [1] 27
```

```r
cols_to_remove <- which(col_zero_counts > 30*48)

df0 <- Irish$indCons[,-cols_to_remove]

df0$time_mean_dem <- rowSums(Irish$indCons)/ncol(Irish$indCons)
```

By examining the histogram of zero counts for each household below, we found that some households had zero electricity usage for over one month during the year. Therefore, we decided to remove these 27 households from our dataset.

Visualizing main characteristics

Compute the mean demand for each time point, for visualization and examining pattern.

```r
df <- cbind(df0[,"time_mean_dem"],Irish$extra)
colnames(df) <- c("time_mean_demand", colnames(Irish$extra))
head(df)
```

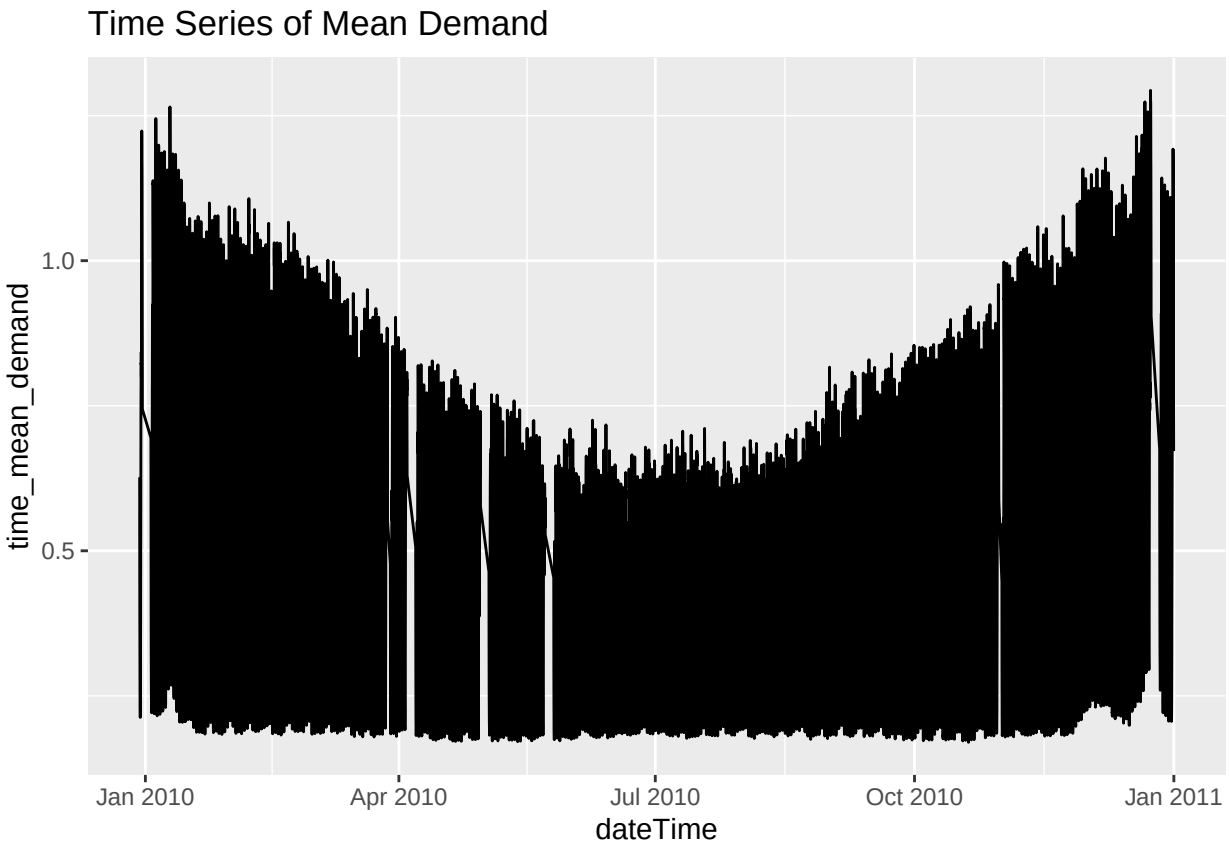
```
##   time_mean_demand time       toy dow  holy tod temp            dateTime
## 1        0.6266460    1 0.9863014 Wed FALSE   0    4 2009-12-29 23:00:00
## 2        0.5256755    2 0.9863014 Wed FALSE   1    4 2009-12-29 23:30:00
## 3        0.4419034    3 0.9863014 Wed FALSE   2    4 2009-12-30 00:00:00
## 4        0.3827193    4 0.9863014 Wed FALSE   3    4 2009-12-30 00:30:00
## 5        0.3282253    5 0.9863014 Wed FALSE   4    4 2009-12-30 01:00:00
## 6        0.2903952    6 0.9863014 Wed FALSE   5    4 2009-12-30 01:30:00
```

```r
# Basic summary of each column
summary(df)
```

```
##  time_mean_demand      time            toy           dow          holy
##  Min.   :0.1699   Min.   :    1   Min.   :0.0000   Sun:2208   Mode :logical
##  1st Qu.:0.3005   1st Qu.: 4200   1st Qu.:0.2411   Thu:2496   FALSE:16799
##  Median :0.4854   Median : 8400   Median :0.5041   Mon:2400
##  Mean   :0.4994   Mean   : 8400   Mean   :0.4975   Tue:2400
##  3rd Qu.:0.6320   3rd Qu.:12600   3rd Qu.:0.7452   Wed:2544
##  Max.   :1.2936   Max.   :16799   Max.   :0.9918   Sat:2352
##                                                    Fri:2399
##       tod             temp            dateTime
##  Min.   : 0.0   Min.   :-10.000   Min.   :2009-12-29 23:00:00.00
##  1st Qu.:12.0   1st Qu.:  4.000   1st Qu.:2010-03-31 10:45:00.00
##  Median :24.0   Median :  9.000   Median :2010-07-05 22:30:00.00
##  Mean   :23.5   Mean   :  8.616   Mean   :2010-07-03 00:08:03.46
##  3rd Qu.:35.5   3rd Qu.: 14.000   3rd Qu.:2010-10-01 10:15:00.00
##  Max.   :47.0   Max.   : 24.000   Max.   :2010-12-31 22:30:00.00
##
```

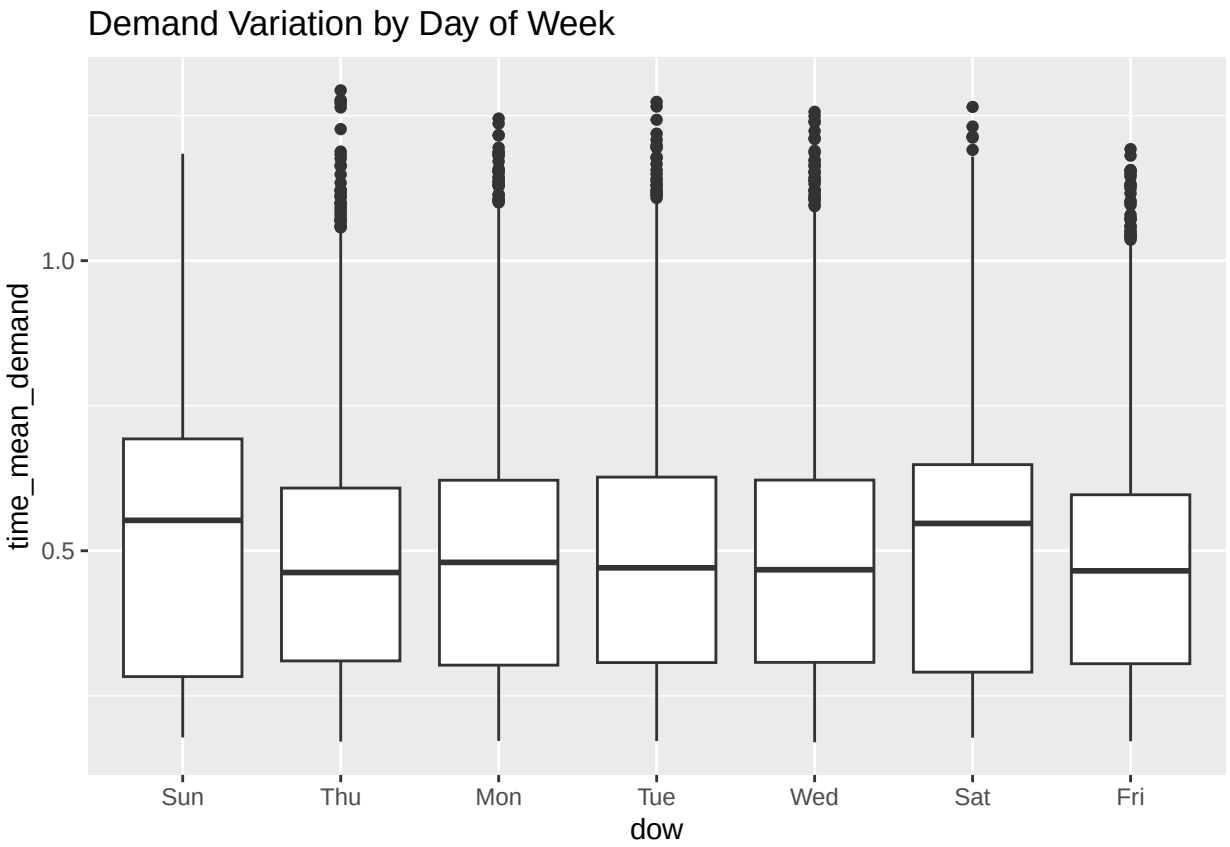
The variable `holy` is all FALSE, we will remove this variable.

```r
# Time series plot of time_mean_demand
ggplot(df, aes(x=dateTime, y=time_mean_demand)) + geom_line() +
  ggtitle("Time Series of Mean Demand")
```



Plot the time series of mean daily demand throughout the year. The plot presents a periodic pattern of electricity demand, with higher demand during the winter months and lower demand during the summer months.

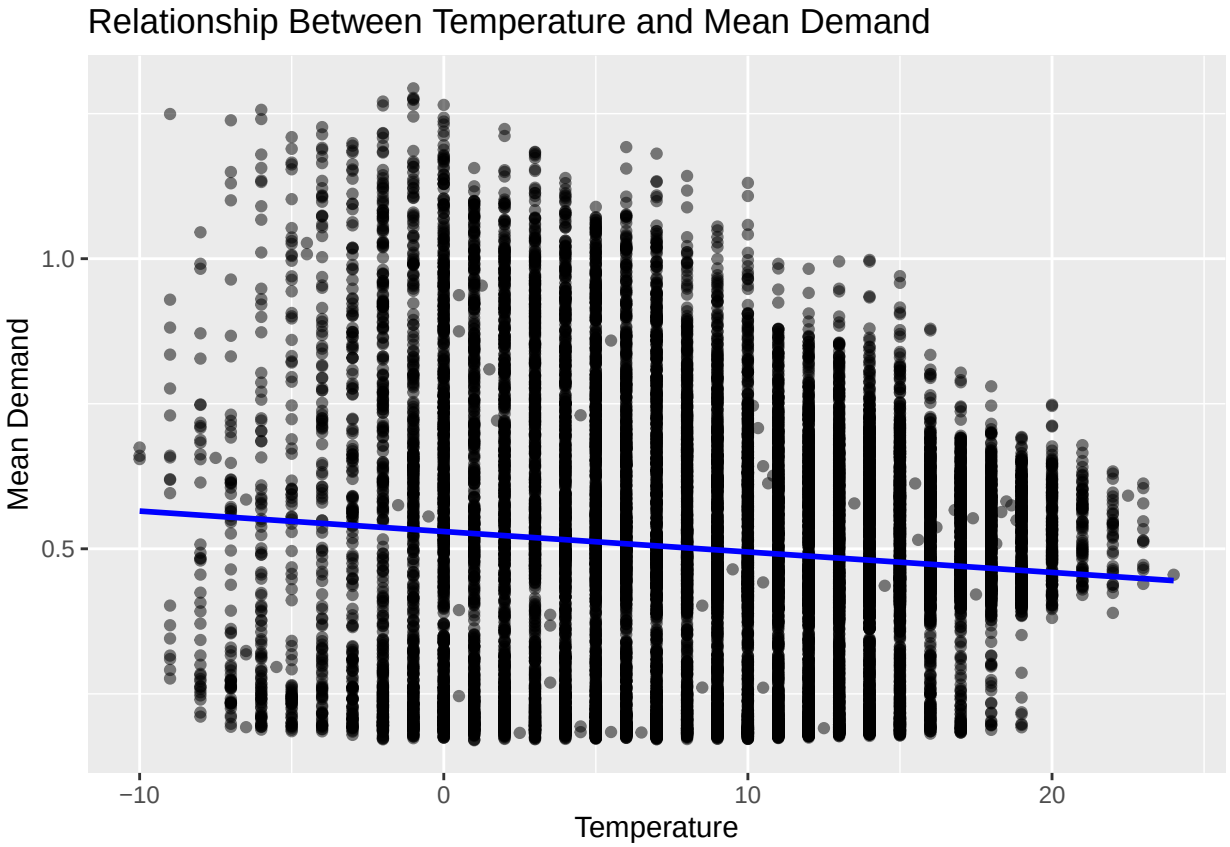
```
# Boxplots to check variation of time_mean_demand across days of the week
ggplot(df, aes(x=dow, y=time_mean_demand)) + geom_boxplot() +
  ggtitle("Demand Variation by Day of Week")
```



The box plots of mean demand for each day of the week shows that the demand patterns are relatively consistent across the weekdays, with slight variations during the weekends. The box plots reveal that weekends tend to have slightly higher demand compared to weekdays.

```
# Scatter plot of time_mean_demand vs. temperature
ggplot(df, aes(x=temp, y=time_mean_demand)) +
    geom_point(alpha=0.5) +
    geom_smooth(method="lm", se=FALSE, color="blue") +
    labs(x="Temperature", y="Mean Demand",
         title="Relationship Between Temperature and Mean Demand")
```

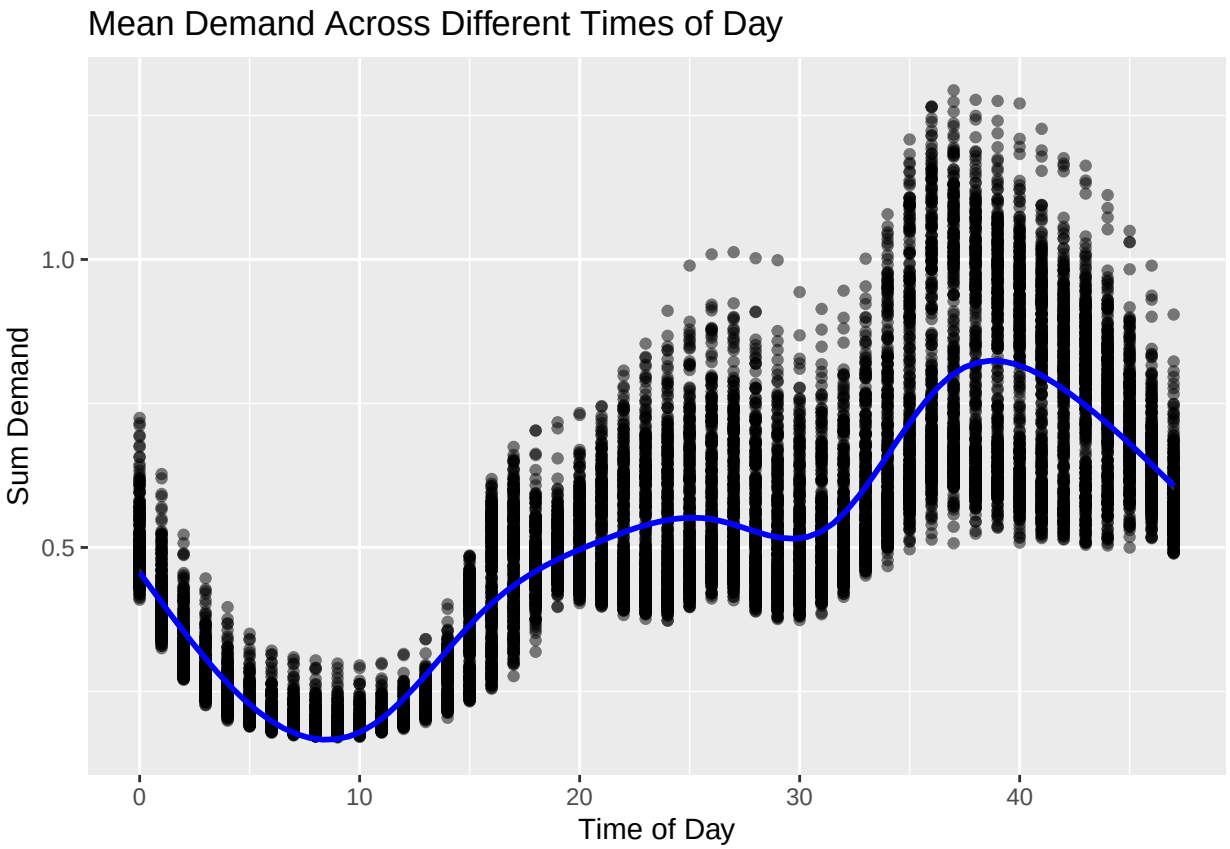
```
## `geom_smooth()` using formula = 'y ~ x'
```



The above plot explores the relationship between temperature and mean demand. The scatter plot suggests an inverse relationship, where higher temperatures generally correlate with lower electricity demand.

```
# Line plot for time_mean_demand across different times of day
ggplot(df, aes(x=tod, y=time_mean_demand, group=1)) +
    geom_point(alpha=0.5) +
    geom_smooth(color="blue") +
    labs(x="Time of Day", y="Sum Demand",
         title="Mean Demand Across Different Times of Day")
```

```
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```



The plot above illustrates the mean demand across different times of the day. The plot shows a clear pattern, with demand peaking in the early morning, around noon, and in the evening, likely reflecting typical household activity patterns.

Aggregation

Formatting the data for clustering

```
# Load the dataset
data(Irish)

# Calculate the number of zero values in each column
col_zero_counts <- colSums(Irish$indCons == 0)

# Identify columns to remove (columns with more than 30*48 zero values)
cols_to_remove <- which(col_zero_counts > 30 * 48)

# Create a data frame with demand data and remove identified columns
df <- Irish$indCons
df <- df[,-cols_to_remove]

# Add date and time columns
df$date <- as.Date(Irish$extra$dateTime)
df$time <- format(Irish$extra$dateTime, "%H:%M:%S")
```

```r
# Gather the data into long format
df_long <- df %>%
  pivot_longer(cols = -c(date, time), names_to = "household_id",
               values_to = "demand")

# Calculate the average demand over the year for each 30-minute interval
avg_demand <- df_long %>%
  group_by(household_id, time) %>%
  summarise(average_demand = mean(demand, na.rm = TRUE)) %>%
  ungroup()
```

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

```r
avg_demand_wide <- avg_demand %>%
  pivot_wider(names_from = time, values_from = average_demand)

# Display the result
print(avg_demand_wide)
```

```
## # A tibble: 2,645 x 49
##    household_id `00:00:00` `00:30:00` `01:00:00` `01:30:00` `02:00:00`
##    <chr>             <dbl>      <dbl>      <dbl>      <dbl>      <dbl>
##  1 I1002             0.103     0.0721     0.0708     0.0630     0.0641
##  2 I1003             0.515     0.476      0.431      0.402      0.370
##  3 I1004             0.870     0.763      0.667      0.610      0.546
##  4 I1005             0.332     0.309      0.274      0.258      0.256
##  5 I1013             0.153     0.124      0.105      0.0947     0.0937
##  6 I1015             0.218     0.195      0.194      0.188      0.183
##  7 I1018             0.696     0.656      0.652      0.625      0.616
##  8 I1020             0.267     0.220      0.201      0.194      0.186
##  9 I1022             0.146     0.127      0.0873     0.0815     0.0780
## 10 I1024             0.131     0.132      0.134      0.145      0.143
## # i 2,635 more rows
## # i 43 more variables: `02:30:00` <dbl>, `03:00:00` <dbl>, `03:30:00` <dbl>,
## #   `04:00:00` <dbl>, `04:30:00` <dbl>, `05:00:00` <dbl>, `05:30:00` <dbl>,
## #   `06:00:00` <dbl>, `06:30:00` <dbl>, `07:00:00` <dbl>, `07:30:00` <dbl>,
## #   `08:00:00` <dbl>, `08:30:00` <dbl>, `09:00:00` <dbl>, `09:30:00` <dbl>,
## #   `10:00:00` <dbl>, `10:30:00` <dbl>, `11:00:00` <dbl>, `11:30:00` <dbl>,
## #   `12:00:00` <dbl>, `12:30:00` <dbl>, `13:00:00` <dbl>, `13:30:00` <dbl>, ...
```

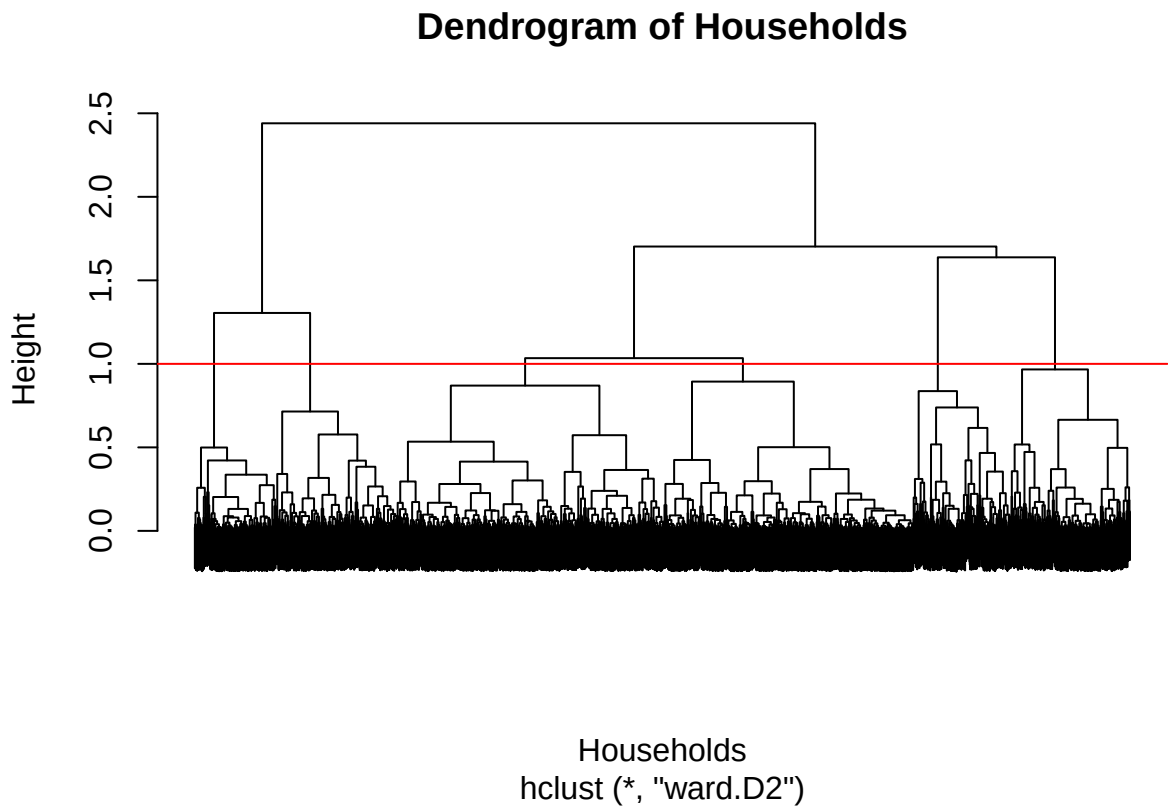
Compute cosine similarity and corresponding distance

```r
# Compute the cosine similarity, and use 1-similarity as distance
compute_cosine_distance_matrix <- function(data) {
  data_matrix <- as.matrix(data[-1])  # Remove the household_id column
  similarity_matrix <- proxy::simil(data_matrix, method = "cosine")
  dist_matrix <- 1 - similarity_matrix
  return(as.matrix(dist_matrix))
}
```

```r
cosine_distances <- compute_cosine_distance_matrix(avg_demand_wide)
```

Perform hierarchical slustering

```r
# Hierarchical clustering
hc <- hclust(as.dist(cosine_distances), method = "ward.D2")

# Plot the dendrogram
plot(hc, labels = FALSE, main = "Dendrogram of Households", xlab = "Households",
     ylab = "Height")
abline(h=1,col="red")
```



Clusters information

```r
# Create clusters
clusters <- cutree(hc, k = 6)
avg_demand_wide$cluster <- clusters
# Summarize the number of households in each cluster
cluster_summary <- avg_demand_wide %>%
  group_by(cluster) %>%
  summarise(num_households = n())
```

```r
# Display the summary
print(cluster_summary)
```

```
## # A tibble: 6 x 2
##   cluster num_households
##     <int>          <int>
## 1       1            754
## 2       2            341
## 3       3            705
## 4       4            277
## 5       5            232
## 6       6            336
```

```r
# Reshape avg_demand back to long format
avg_demand_long <- avg_demand_wide %>%
  pivot_longer(cols = -c(household_id, cluster), names_to = "time",
               values_to = "daily_demand")


# Join cluster information back to the original dataframe
df_with_clusters <- df_long %>%
  left_join(avg_demand_long, by = c("household_id", "time"))

# Analyze cluster characteristics
cluster_analysis <- df_with_clusters %>%
  group_by(cluster) %>%
  summarise(
    average_demand = mean(daily_demand, na.rm = TRUE)
  )

print(cluster_analysis)
```

```
## # A tibble: 6 x 2
##   cluster average_demand
##     <int>          <dbl>
## 1       1          0.518
## 2       2          0.440
## 3       3          0.542
## 4       4          0.494
## 5       5          0.472
## 6       6          0.482
```

Aggregate data for households in the same cluster

```r
df_t <- as.data.frame(t(df[,-c(ncol(df)-1,ncol(df))]))

# Step 2: Add the clusters as a new column to the transposed data frame
df_t$cluster <- clusters

# Step 3: Group by cluster and calculate the mean for each row within each
```

```r
#cluster
mean_by_cluster <- df_t %>%
  group_by(cluster) %>%
  summarise(across(everything(), mean, na.rm = TRUE))
```

```
## Warning: There was 1 warning in `summarise()`.
## i In argument: `across(everything(), mean, na.rm = TRUE)`.
## i In group 1: `cluster = 1`.
## Caused by warning:
## ! The `...` argument of `across()` is deprecated as of dplyr 1.1.0.
## Supply arguments directly to `.fns` through an anonymous function instead.
##
##   # Previously
##   across(a:b, mean, na.rm = TRUE)
##
##   # Now
##   across(a:b, \(x) mean(x, na.rm = TRUE))
```

```r
mean_by_cluster <- as.data.frame(mean_by_cluster[,-1])
rownames(mean_by_cluster) <- c("Cluster 1","Cluster 2","Cluster 3","Cluster 4",
                               "Cluster 5","Cluster 6")
# View the result
head(mean_by_cluster[,1:10])
```

```
##                8114      8115      8116      8117      8118      8119      8120
## Cluster 1 0.5690531 0.4515690 0.3734496 0.3057414 0.2624218 0.2491910 0.2107865
## Cluster 2 0.6211584 0.4668358 0.3567683 0.3211408 0.2705337 0.2262287 0.1992346
## Cluster 3 0.7272908 0.6324426 0.5447660 0.4731830 0.3912837 0.3270000 0.2870894
## Cluster 4 0.4869314 0.4188953 0.3694910 0.3158375 0.3068123 0.2934188 0.2780686
## Cluster 5 0.8591121 0.8098621 0.7029784 0.6303922 0.5516552 0.4941983 0.4133491
## Cluster 6 0.5408839 0.4471399 0.3716131 0.3346250 0.2846935 0.2430030 0.2201429
##                8121      8122      8123
## Cluster 1 0.2028024 0.1918488 0.1879536
## Cluster 2 0.1907243 0.1852698 0.1756364
## Cluster 3 0.2505915 0.2411986 0.2386610
## Cluster 4 0.2765523 0.2959097 0.2905487
## Cluster 5 0.3627716 0.3327328 0.3117888
## Cluster 6 0.2109643 0.2046845 0.1964881
```

```r
df0 <- Irish$extra
df0 <- df0 %>% mutate(dow = ifelse(dow %in% c("Sat", "Sun"), "True", "False")) %>% select(-c(holy,time,
df0 <-t(df0)
```

```r
colnames(df0) <- colnames(mean_by_cluster)
mean_by_cluster <- rbind(mean_by_cluster,df0)
mean_by_cluster <- as.data.frame(t(mean_by_cluster))
# Convert Cluster  and temp columns to numeric, weekend into logic
mean_by_cluster <- mean_by_cluster %>%
  mutate(across(starts_with("Cluster"), as.numeric),
         weekend = as.logical(weekend),
         temp = as.numeric(temp))
```

The data is formatted and ready for model fitting, uncomment the following code to save data as CSV file/

```
#write.csv(mean_by_cluster, file = "AggregatedData1.csv", row.names = FALSE)
```

```
avg_demand_wide$cluster <- clusters
#write.csv(avg_demand_wide, file = "cluster_data.csv", row.names = FALSE)
```

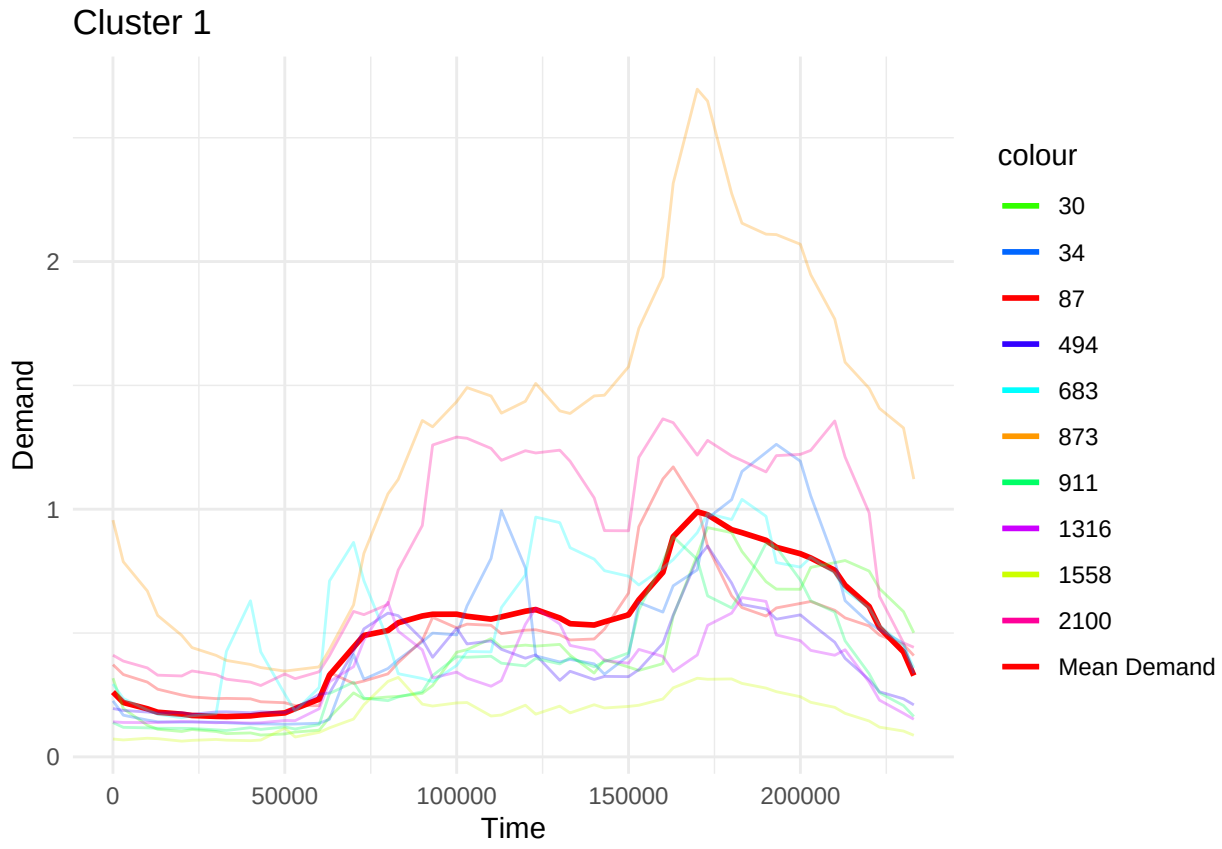
Plot the mean demand and sample demands for each cluster

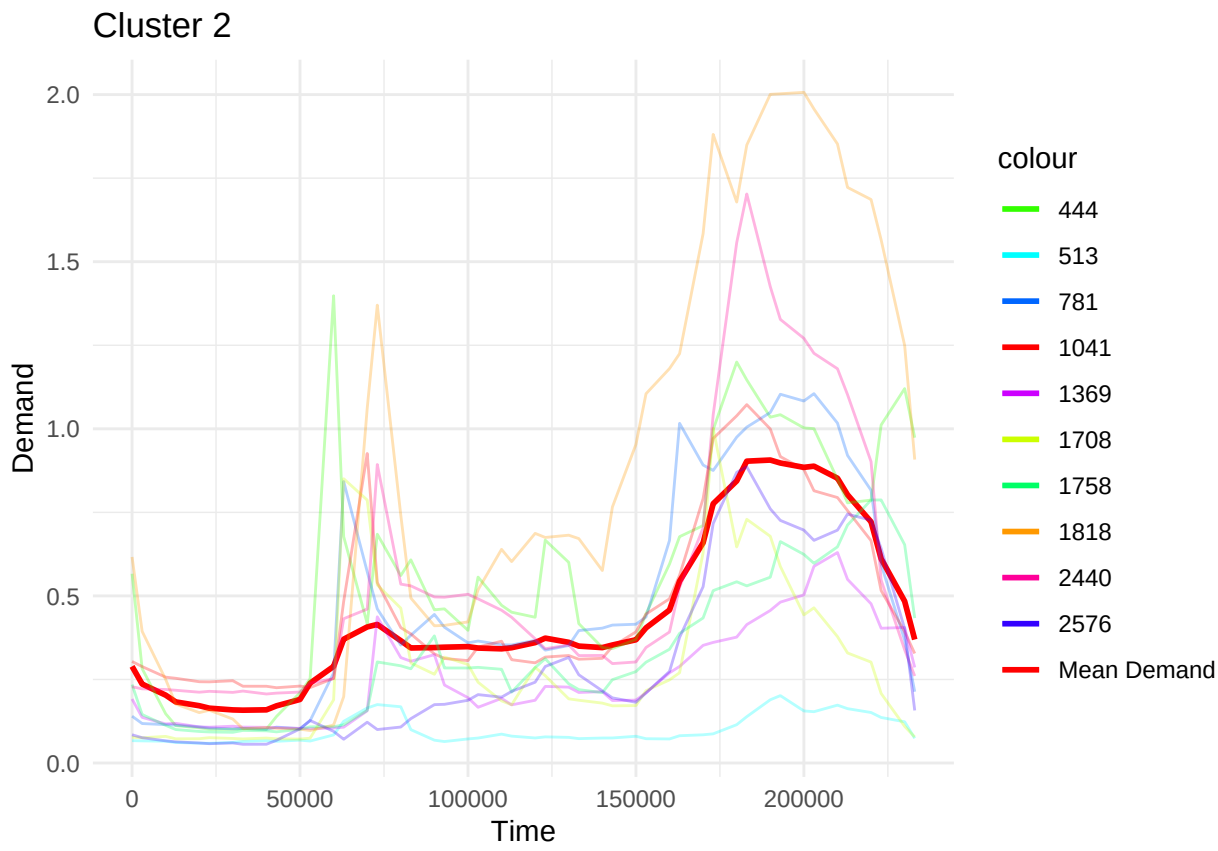
In the following plots, we want to visually show if the clustering give a large between-clusters discrepancy and a small within-cluster discrepancy.

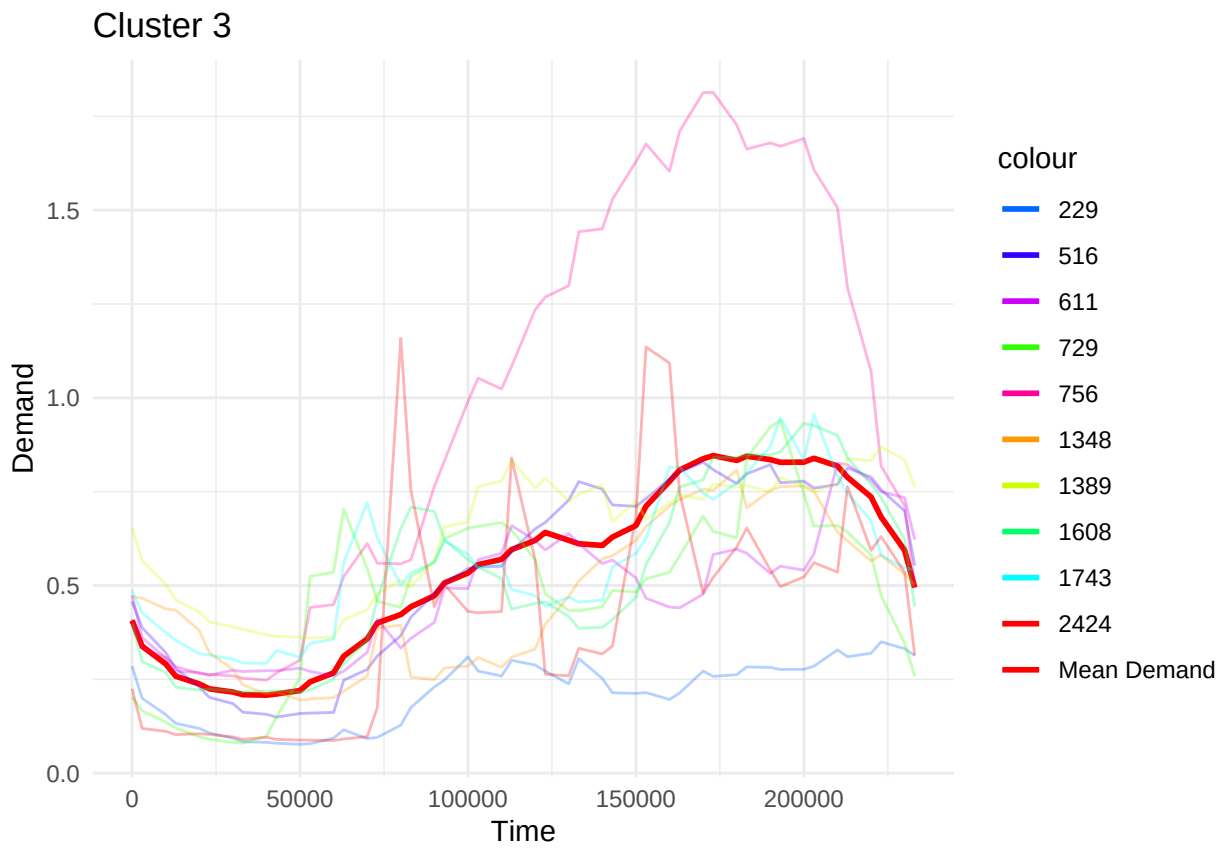
```
# Identify the time columns by excluding non-time columns
time_columns <- grep("^\\d{2}:\\d{2}:\\d{2}$", colnames(avg_demand_wide),
                     value = TRUE)

# Add rowid to avg_demand_wide for sampling
avg_demand_wide <- avg_demand_wide %>%
  mutate(rowid = row_number())

# Convert data to long format for plotting
avg_demand_long <- avg_demand_wide %>%
  pivot_longer(cols = all_of(time_columns),
               names_to = "Time",
               values_to = "Demand") %>%
  mutate(Time = as.numeric(gsub(":", "", Time)))

# Calculate mean demand for each cluster
mean_demand <- avg_demand_long %>%
  group_by(cluster, Time) %>%
  summarize(mean_demand = mean(Demand), .groups = 'drop')

# Plot mean demand for each cluster
for (cl in unique(avg_demand_wide$cluster)) {
  # Filter data for the cluster
  cluster_data <- avg_demand_long %>% filter(cluster == cl)

  # Sample 10 houses from the cluster
  sampled_houses <- sample(unique(cluster_data$rowid), 10)
  sample_data <- cluster_data %>% filter(rowid %in% sampled_houses)

  # Plot mean demand
  p <- ggplot() +
    geom_line(data = mean_demand %>% filter(cluster == cl),
              aes(x = Time, y = mean_demand, color = "Mean Demand"), size = 1) +
    geom_line(data = sample_data,
              aes(x = Time, y = Demand, group = rowid,
                  color = as.factor(rowid)), alpha = 0.3) +
    labs(title = paste("Cluster", cl),
         x = "Time",
         y = "Demand") +
    scale_color_manual(values = c("Mean Demand" = "red", setNames(rainbow(10),
                                                    sampled_houses))) +
```

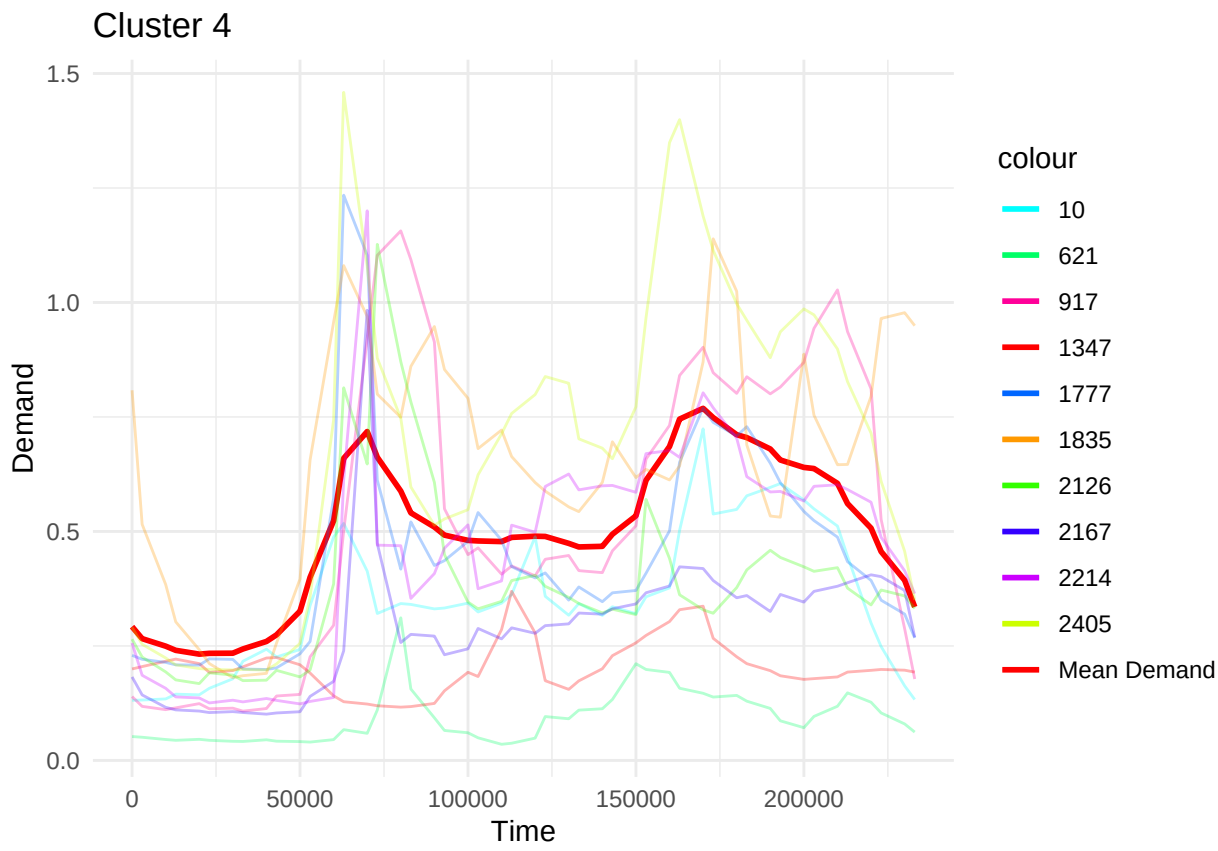
```
    theme_minimal()

  print(p)
}
```

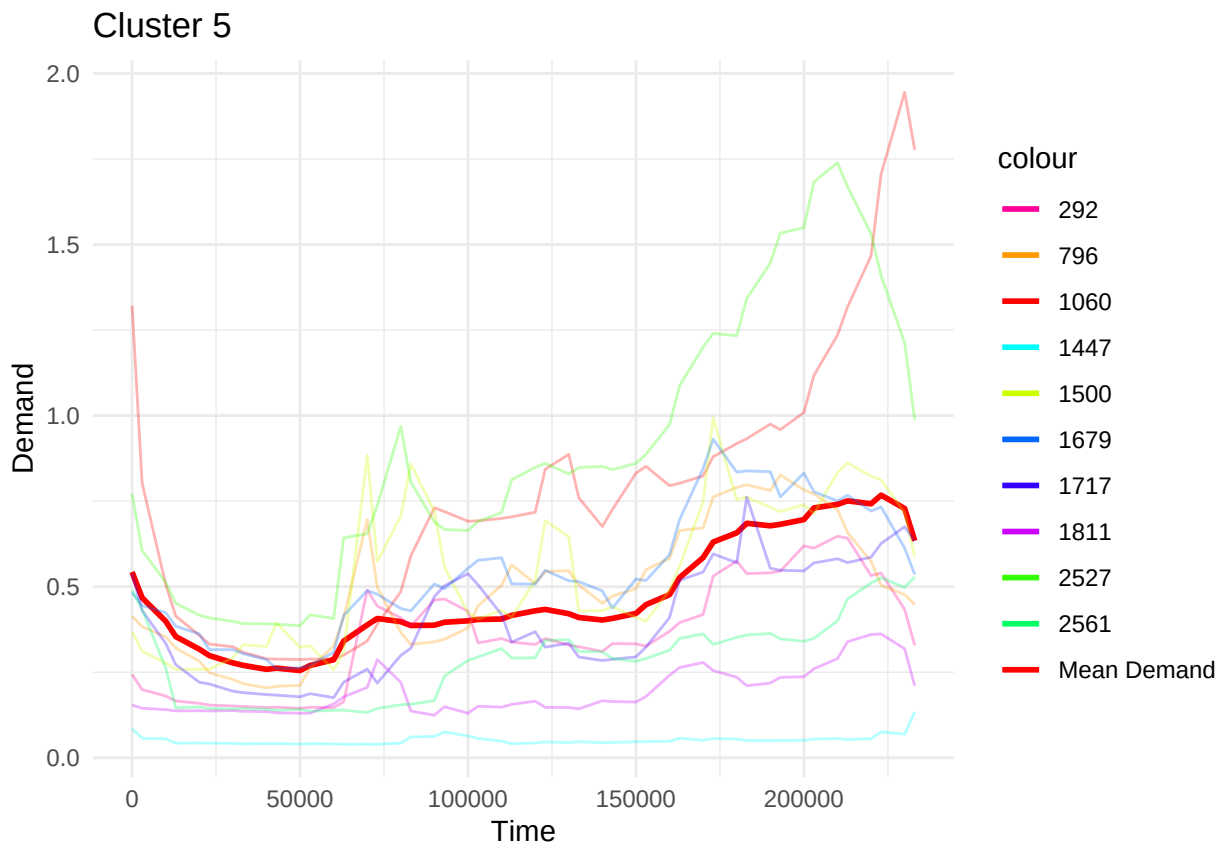
```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

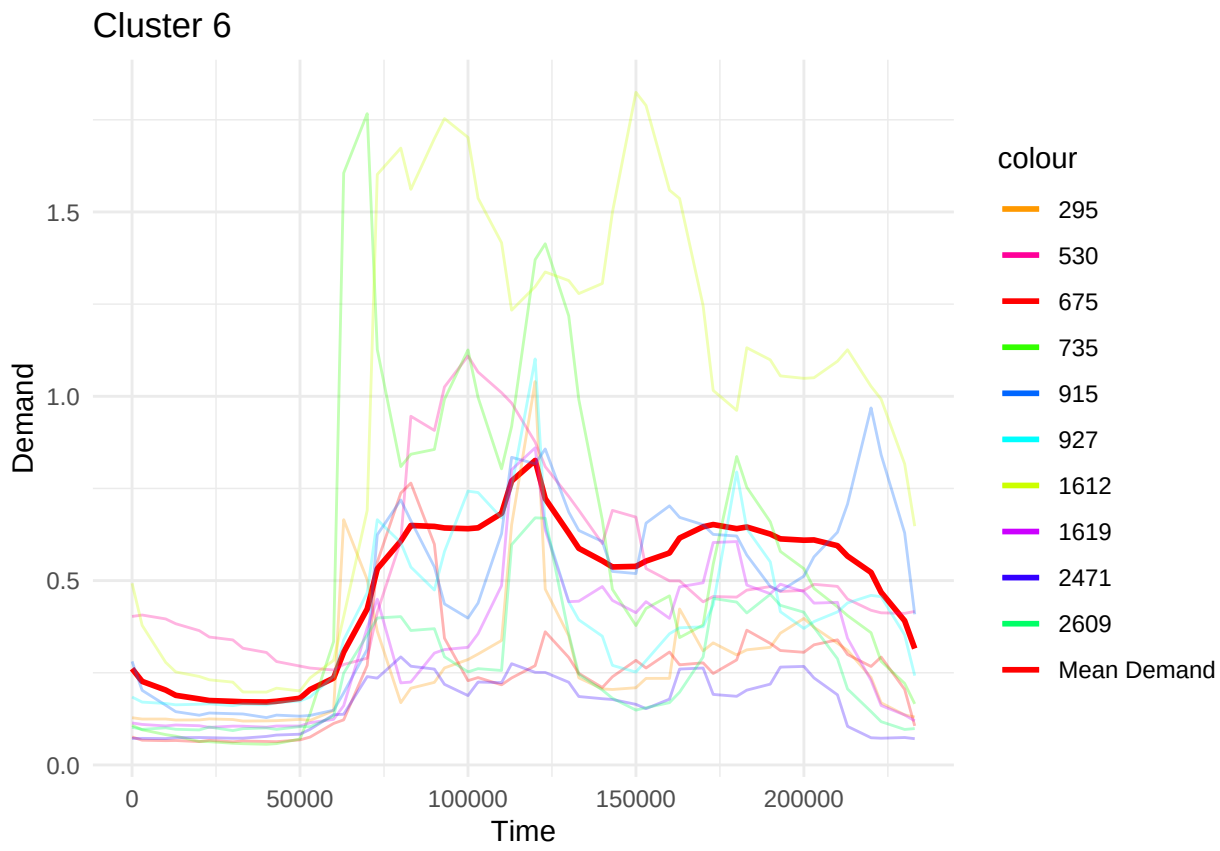



Cluster 2
Demand
2.0
1.5
1.0
0.5
0.0
0
50000
100000
150000
200000
Time
colour
444
513
781
1041
1369
1708
1758
1818
2440
2576
Mean Demand


Cluster 3
Demand
1.5
1.0
0.5
0.0
0
50000
100000
150000
200000
Time
colour
229
516
611
729
756
1348
1389
1608
1743
2424
Mean Demand


Cluster 4
1.5
1.0
0.5
0.0
Demand
0
50000
100000
150000
200000
Time
colour
10
621
917
1347
1777
1835
2126
2167
2214
2405
Mean Demand


Cluster 5
2.0
1.5
1.0
0.5
0.0
Demand
0
50000
100000
150000
200000
Time
colour
292
796
1060
1447
1500
1679
1717
1811
2527
2561
Mean Demand



Linear Regression Model

Data Preparation

We first prepare the data for each cluster.

```
aggregated_data <- read.csv("AggregatedData1.csv")
daily_data <- read.csv("Daily_AggregatedData1.csv")

cluster1 <- aggregated_data[, -c(2,3,4,5,6)]
cluster2 <- aggregated_data[, -c(1,3,4,5,6)]
cluster3 <- aggregated_data[, -c(1,2,4,5,6)]
cluster4 <- aggregated_data[, -c(1,2,3,5,6)]
cluster5 <- aggregated_data[, -c(1,2,3,4,6)]
cluster6 <- aggregated_data[, -c(1,2,3,4,5)]

cluster1_trans <- cluster1 %>%
  mutate(Cluster.1_lag1 = lag(Cluster.1, n = 1)) %>%
  mutate(
    tod_poly1 = tod,
    tod_poly2 = tod^2,
    tod_poly3 = tod^3,
    tod_poly4 = tod^4,
    weekend_dummy = ifelse(weekend == "TRUE", 1, 0),
    toy_sin = sin(toy),
```

```
    toy_cos = cos(toy)
  ) %>%
  na.omit()

cluster2_trans <- cluster2 %>%
  mutate(Cluster.2_lag1 = lag(Cluster.2, n = 1)) %>%
  mutate(
    tod_poly1 = tod,
    tod_poly2 = tod^2,
    tod_poly3 = tod^3,
    tod_poly4 = tod^4,
    weekend_dummy = ifelse(weekend == "TRUE", 1, 0),
    toy_sin = sin(toy),
    toy_cos = cos(toy)
  ) %>%
  na.omit()

cluster3_trans <- cluster3 %>%
  mutate(Cluster.3_lag1 = lag(Cluster.3, n = 1)) %>%
  mutate(
    tod_poly1 = tod,
    tod_poly2 = tod^2,
    tod_poly3 = tod^3,
    tod_poly4 = tod^4,
    weekend_dummy = ifelse(weekend == "TRUE", 1, 0),
    toy_sin = sin(toy),
    toy_cos = cos(toy)
  ) %>%
  na.omit()


cluster4_trans <- cluster4 %>%
  mutate(Cluster.4_lag1 = lag(Cluster.4, n = 1)) %>%
  mutate(
    tod_poly1 = tod,
    tod_poly2 = tod^2,
    tod_poly3 = tod^3,
    tod_poly4 = tod^4,
    weekend_dummy = ifelse(weekend == "TRUE", 1, 0),
    toy_sin = sin(toy),
    toy_cos = cos(toy)
  ) %>%
  na.omit()

cluster5_trans <- cluster5 %>%
  mutate(Cluster.5_lag1 = lag(Cluster.5, n = 1)) %>%
  mutate(
    tod_poly1 = tod,
    tod_poly2 = tod^2,
    tod_poly3 = tod^3,
    tod_poly4 = tod^4,
    weekend_dummy = ifelse(weekend == "TRUE", 1, 0),
    toy_sin = sin(toy),
```

```
    toy_cos = cos(toy)
  ) %>%
  na.omit()

cluster6_trans <- cluster6 %>%
  mutate(Cluster.6_lag1 = lag(Cluster.6, n = 1)) %>%
  mutate(
    tod_poly1 = tod,
    tod_poly2 = tod^2,
    tod_poly3 = tod^3,
    tod_poly4 = tod^4,
    weekend_dummy = ifelse(weekend == "TRUE", 1, 0),
    toy_sin = sin(toy),
    toy_cos = cos(toy)
  ) %>%
  na.omit()
```

Clusrer 1

Linear Regression Model for Cluster 1

On the basis of `cluster1_trans`, we fit a linear regression model and evaluate its performance:

```
# Split the data into training and testing sets
train_index_1 <- 1:floor(0.8 * nrow(cluster1_trans))
train_data_1 <- cluster1_trans[train_index_1, ]
test_data_1 <- cluster1_trans[-train_index_1, ]

# Fit the linear regression model
linear_model_1 <- lm(Cluster.1 ~ Cluster.1_lag1 + temp + tod_poly1 + tod_poly2
                     + tod_poly3 + tod_poly4 + weekend_dummy + toy_sin +
                       toy_cos, data = train_data_1)
```

Performance of Predictions

```
# Make predictions on the testing set
test_data_1$predictions <- predict(linear_model_1, newdata = test_data_1)

# Calculate prediction error metrics
mae_1 <- mean(abs(test_data_1$Cluster.1 - test_data_1$predictions))
mse_1 <- mean((test_data_1$Cluster.1 - test_data_1$predictions)^2)
rmse_1 <- sqrt(mse_1)

# Print the results
cat("Mean Absolute Error (MAE):", mae_1, "\n")
```

```
## Mean Absolute Error (MAE): 0.04186565
```

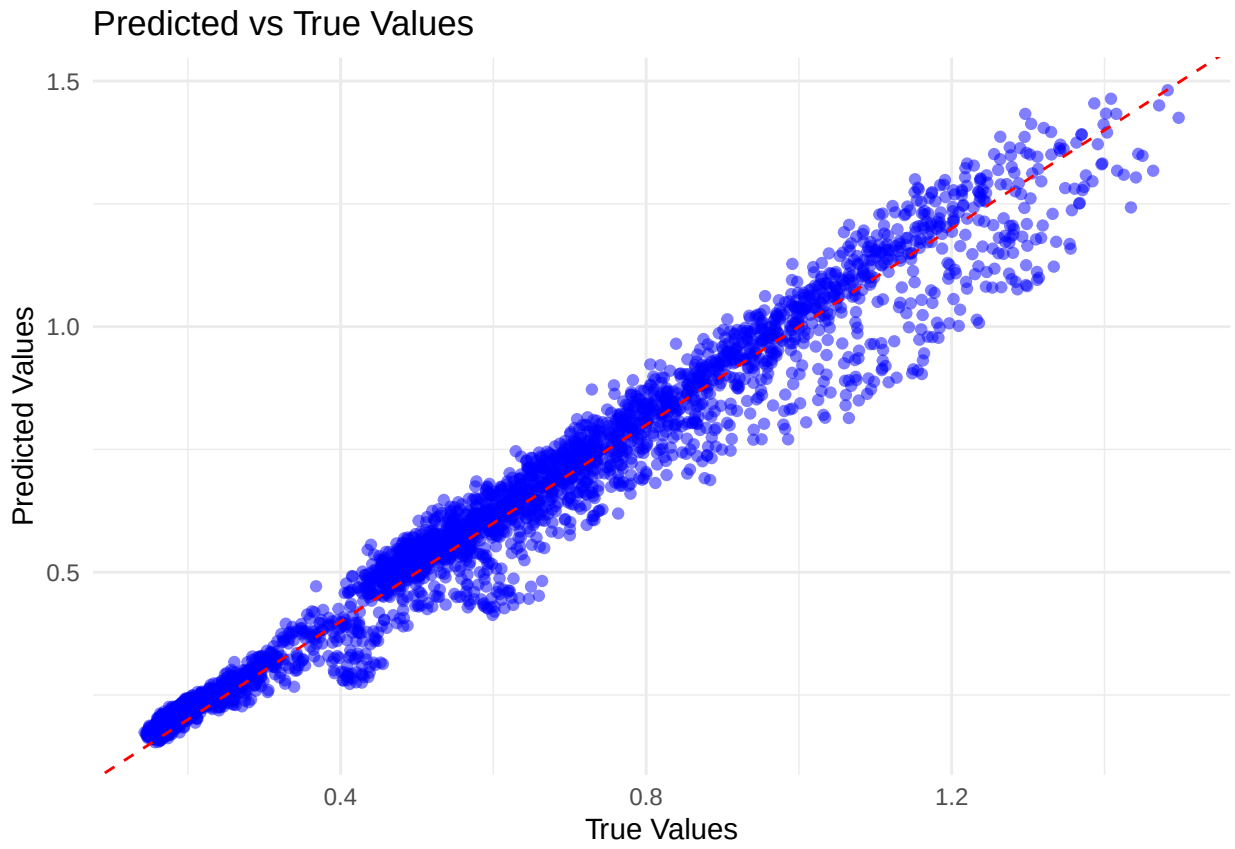
```
cat("Mean Squared Error (MSE):", mse_1, "\n")
```

```
## Mean Squared Error (MSE): 0.003343571
```

```r
cat("Root Mean Squared Error (RMSE):", rmse_1, "\n")
```

```
## Root Mean Squared Error (RMSE): 0.05782362
```

```r
# Plot predicted vs true values
ggplot(test_data_1, aes(x = Cluster.1, y = predictions)) +
  geom_point(color = 'blue', alpha = 0.5) +
  geom_abline(intercept = 0, slope = 1, color = 'red', linetype = "dashed") +
  labs(title = "Predicted vs True Values",
       x = "True Values",
       y = "Predicted Values") +
  theme_minimal()
```



Linear Regression Model Summary

```r
summary(linear_model_1)
```

```
##
## Call:
## lm(formula = Cluster.1 ~ Cluster.1_lag1 + temp + tod_poly1 +
##     tod_poly2 + tod_poly3 + tod_poly4 + weekend_dummy + toy_sin +
##     toy_cos, data = train_data_1)
##
## Residuals:
```

```
##       Min        1Q    Median        3Q       Max
## -0.165153 -0.026932 -0.008028  0.017576  0.300232
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -3.660e-02  1.999e-02  -1.831   0.0671 .
## Cluster.1_lag1  9.265e-01  3.346e-03 276.867  < 2e-16 ***
## temp           -9.932e-04  1.286e-04  -7.725  1.2e-14 ***
## tod_poly1       2.293e-02  6.069e-04  37.780  < 2e-16 ***
## tod_poly2      -1.602e-03  5.276e-05 -30.367  < 2e-16 ***
## tod_poly3       5.057e-05  1.632e-06  30.989  < 2e-16 ***
## tod_poly4      -5.649e-07  1.682e-08 -33.597  < 2e-16 ***
## weekend_dummy   1.519e-03  9.309e-04   1.632   0.1027
## toy_sin        -9.073e-03  9.531e-03  -0.952   0.3411
## toy_cos        -3.571e-02  1.860e-02  -1.920   0.0549 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.048 on 13428 degrees of freedom
## Multiple R-squared:  0.9653, Adjusted R-squared:  0.9652
## F-statistic: 4.145e+04 on 9 and 13428 DF,  p-value: < 2.2e-16
```

Cluster 2

Linear Regression Model for Cluster 2

```r
# Split the data into training and testing sets
train_index_2 <- 1:floor(0.8 * nrow(cluster2_trans))
train_data_2 <- cluster2_trans[train_index_2, ]
test_data_2 <- cluster2_trans[-train_index_2, ]

# Fit the linear regression model
linear_model_2 <- lm(Cluster.2 ~ Cluster.2_lag1 + temp + tod_poly1 + tod_poly2
                     + tod_poly3 + tod_poly4 + weekend_dummy + toy_sin
                     + toy_cos, data = train_data_2)
```

Performance of Predictions

```r
# Make predictions on the testing set
test_data_2$predictions <- predict(linear_model_2, newdata = test_data_2)

# Calculate prediction error metrics
mae_2 <- mean(abs(test_data_2$Cluster.2 - test_data_2$predictions))
mse_2 <- mean((test_data_2$Cluster.2 - test_data_2$predictions)^2)
rmse_2 <- sqrt(mse_2)

# Print the results
cat("Mean Absolute Error (MAE):", mae_2, "\n")
```

```
## Mean Absolute Error (MAE): 0.03925669
```

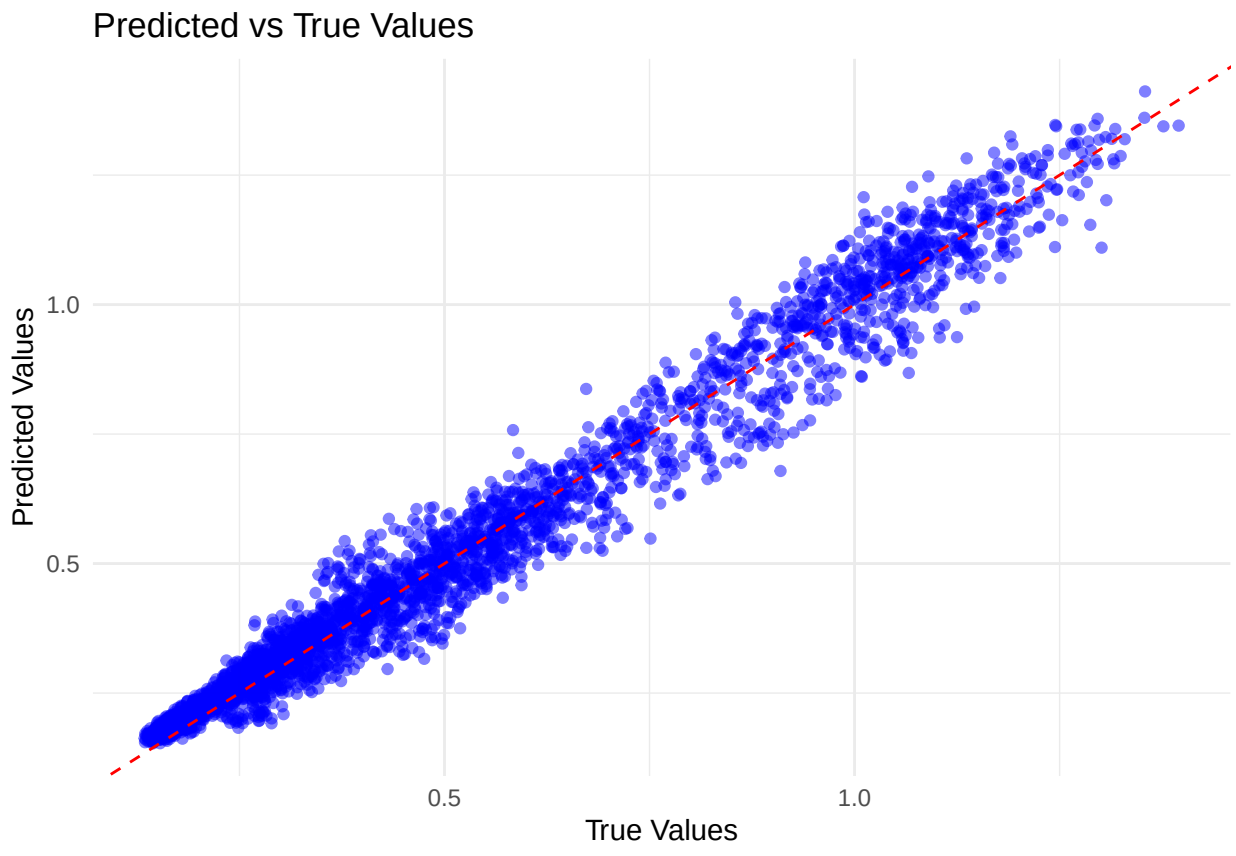
```r
cat("Mean Squared Error (MSE):", mse_2, "\n")
```

```
## Mean Squared Error (MSE): 0.002707433
```

```r
cat("Root Mean Squared Error (RMSE):", rmse_2, "\n")
```

```
## Root Mean Squared Error (RMSE): 0.052033
```

```r
# Plot predicted vs true values
ggplot(test_data_2, aes(x = Cluster.2, y = predictions)) +
  geom_point(color = 'blue', alpha = 0.5) +
  geom_abline(intercept = 0, slope = 1, color = 'red', linetype = "dashed") +
  labs(title = "Predicted vs True Values",
       x = "True Values",
       y = "Predicted Values") +
  theme_minimal()
```



Linear Regression Model Summary

```r
summary(linear_model_2)
```

```
##
## Call:
```

```
## lm(formula = Cluster.2 ~ Cluster.2_lag1 + temp + tod_poly1 +
##     tod_poly2 + tod_poly3 + tod_poly4 + weekend_dummy + toy_sin +
##     toy_cos, data = train_data_2)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.196463 -0.024745 -0.005851  0.020245  0.263215
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -1.276e-01  1.877e-02  -6.799 1.10e-11 ***
## Cluster.2_lag1  9.571e-01  2.936e-03 326.014  < 2e-16 ***
## temp           -9.467e-04  1.217e-04  -7.778 7.87e-15 ***
## tod_poly1       3.788e-02  5.538e-04  68.407  < 2e-16 ***
## tod_poly2      -3.201e-03  4.786e-05 -66.870  < 2e-16 ***
## tod_poly3       1.053e-04  1.510e-06  69.782  < 2e-16 ***
## tod_poly4      -1.148e-06  1.579e-08 -72.691  < 2e-16 ***
## weekend_dummy   1.161e-03  8.849e-04   1.312    0.190
## toy_sin         1.321e-02  8.965e-03   1.474    0.141
## toy_cos         5.188e-03  1.752e-02   0.296    0.767
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.04535 on 13428 degrees of freedom
## Multiple R-squared:  0.9669, Adjusted R-squared:  0.9669
## F-statistic: 4.359e+04 on 9 and 13428 DF,  p-value: < 2.2e-16
```

Cluster 3

Linear Regression Model for Cluster 3

```r
# Split the data into training and testing sets
train_index_3 <- 1:floor(0.8 * nrow(cluster3_trans))
train_data_3 <- cluster3_trans[train_index_3, ]
test_data_3 <- cluster3_trans[-train_index_3, ]

# Fit the linear regression model
linear_model_3 <- lm(Cluster.3 ~ Cluster.3_lag1 + temp + tod_poly1 + tod_poly2
                     + tod_poly3 + tod_poly4 + weekend_dummy + toy_sin
                     + toy_cos, data = train_data_3)
```

Performance of Predictions

```r
# Make predictions on the testing set
test_data_3$predictions <- predict(linear_model_3, newdata = test_data_3)

# Calculate prediction error metrics
mae_3 <- mean(abs(test_data_3$Cluster.3 - test_data_3$predictions))
mse_3 <- mean((test_data_3$Cluster.3 - test_data_3$predictions)^2)
rmse_3 <- sqrt(mse_3)

# Print the results
cat("Mean Absolute Error (MAE):", mae_3, "\n")
```

```
## Mean Absolute Error (MAE): 0.0290271
```

```
cat("Mean Squared Error (MSE):", mse_3, "\n")
```

```
## Mean Squared Error (MSE): 0.001502561
```

```
cat("Root Mean Squared Error (RMSE):", rmse_3, "\n")
```

```
## Root Mean Squared Error (RMSE): 0.03876289
```

```
# Plot predicted vs true values
ggplot(test_data_3, aes(x = Cluster.3, y = predictions)) +
  geom_point(color = 'blue', alpha = 0.5) +
  geom_abline(intercept = 0, slope = 1, color = 'red', linetype = "dashed") +
  labs(title = "Predicted vs True Values",
       x = "True Values",
       y = "Predicted Values") +
  theme_minimal()
```

Linear Regression Model Summary

```
summary(linear_model_3)
```

```
##
## Call:
## lm(formula = Cluster.3 ~ Cluster.3_lag1 + temp + tod_poly1 +
##     tod_poly2 + tod_poly3 + tod_poly4 + weekend_dummy + toy_sin +
##     toy_cos, data = train_data_3)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.118353 -0.018677 -0.002823  0.016148  0.198274
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -2.665e-02  1.459e-02  -1.827  0.06779 .
## Cluster.3_lag1  9.349e-01  3.359e-03 278.292  < 2e-16 ***
## temp           -7.840e-04  8.744e-05  -8.966  < 2e-16 ***
## tod_poly1       1.486e-02  5.223e-04  28.443  < 2e-16 ***
## tod_poly2      -7.698e-04  4.278e-05 -17.994  < 2e-16 ***
## tod_poly3       2.094e-05  1.248e-06  16.775  < 2e-16 ***
## tod_poly4      -2.247e-07  1.237e-08 -18.173  < 2e-16 ***
## weekend_dummy   1.569e-03  6.391e-04   2.455  0.01409 *
## toy_sin        -1.191e-02  6.667e-03  -1.786  0.07406 .
## toy_cos        -3.624e-02  1.299e-02  -2.790  0.00529 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.03262 on 13428 degrees of freedom
## Multiple R-squared:  0.9797, Adjusted R-squared:  0.9797
## F-statistic: 7.193e+04 on 9 and 13428 DF,  p-value: < 2.2e-16
```

Cluster 4

Linear Regression Model for Cluster 4

```r
# Split the data into training and testing sets
train_index_4 <- 1:floor(0.8 * nrow(cluster4_trans))
train_data_4 <- cluster4_trans[train_index_4, ]
test_data_4 <- cluster4_trans[-train_index_4, ]

# Fit the linear regression model
linear_model_4 <- lm(Cluster.4 ~ Cluster.4_lag1 + temp + tod_poly1 + tod_poly2
                     + tod_poly3 + tod_poly4 + weekend_dummy + toy_sin
                     + toy_cos, data = train_data_4)

# Make predictions on the testing set
test_data_4$predictions <- predict(linear_model_4, newdata = test_data_4)
```

Performance of Predictions

```r
# Calculate prediction error metrics
mae_4 <- mean(abs(test_data_4$Cluster.4 - test_data_4$predictions))
mse_4 <- mean((test_data_4$Cluster.4 - test_data_4$predictions)^2)
rmse_4 <- sqrt(mse_4)
```

```r
# Print the results
cat("Mean Absolute Error (MAE):", mae_4, "\n")
```

```
## Mean Absolute Error (MAE): 0.04281978
```

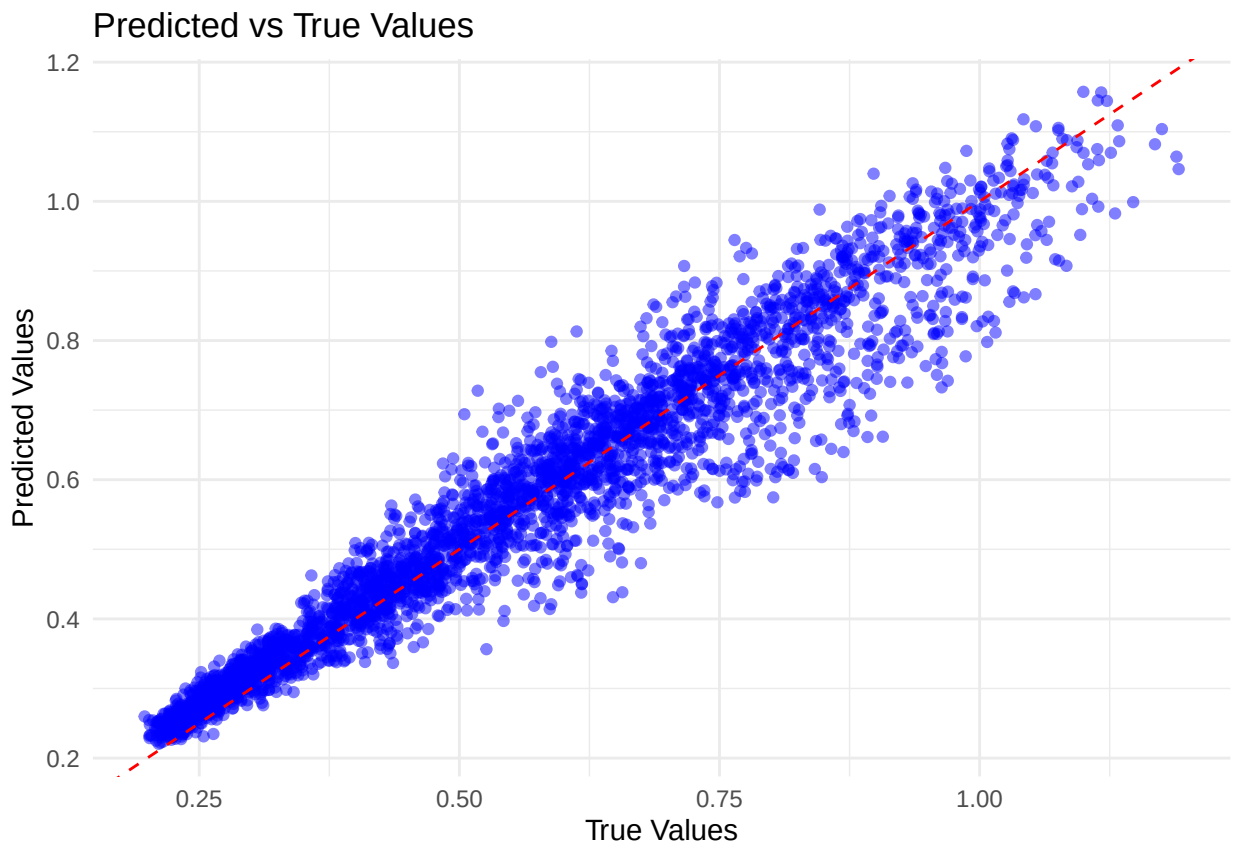
```r
cat("Mean Squared Error (MSE):", mse_4, "\n")
```

```
## Mean Squared Error (MSE): 0.003455547
```

```r
cat("Root Mean Squared Error (RMSE):", rmse_4, "\n")
```

```
## Root Mean Squared Error (RMSE): 0.0587839
```

```r
# Plot predicted vs true values
ggplot(test_data_4, aes(x = Cluster.4, y = predictions)) +
  geom_point(color = 'blue', alpha = 0.5) +
  geom_abline(intercept = 0, slope = 1, color = 'red', linetype = "dashed") +
  labs(title = "Predicted vs True Values",
       x = "True Values",
       y = "Predicted Values") +
  theme_minimal()
```



Linear Regression Model Summary

```r
summary(linear_model_4)
```

```
##
## Call:
## lm(formula = Cluster.4 ~ Cluster.4_lag1 + temp + tod_poly1 +
##     tod_poly2 + tod_poly3 + tod_poly4 + weekend_dummy + toy_sin +
##     toy_cos, data = train_data_4)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.220407 -0.030273 -0.006641  0.024947  0.279164
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)     3.487e-02  2.205e-02   1.581  0.11389
## Cluster.4_lag1  8.858e-01  3.853e-03 229.912  < 2e-16 ***
## temp           -1.079e-03  1.438e-04  -7.502  6.7e-14 ***
## tod_poly1       2.221e-02  6.335e-04  35.058  < 2e-16 ***
## tod_poly2      -1.660e-03  5.618e-05 -29.545  < 2e-16 ***
## tod_poly3       5.144e-05  1.786e-06  28.805  < 2e-16 ***
## tod_poly4      -5.492e-07  1.866e-08 -29.431  < 2e-16 ***
## weekend_dummy   1.669e-03  1.035e-03   1.613  0.10670
## toy_sin        -2.580e-02  1.059e-02  -2.436  0.01486 *
## toy_cos        -6.427e-02  2.065e-02  -3.112  0.00186 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05364 on 13428 degrees of freedom
## Multiple R-squared:  0.9081, Adjusted R-squared:  0.908
## F-statistic: 1.474e+04 on 9 and 13428 DF,  p-value: < 2.2e-16
```

Cluster 5

Linear Regression Model for Cluster 5

```r
# Split the data into training and testing sets
train_index_5 <- 1:floor(0.8 * nrow(cluster5_trans))
train_data_5 <- cluster5_trans[train_index_5, ]
test_data_5 <- cluster5_trans[-train_index_5, ]

# Fit the linear regression model
linear_model_5 <- lm(Cluster.5 ~ Cluster.5_lag1 + temp + tod_poly1 + tod_poly2
                     + tod_poly3 + tod_poly4 + weekend_dummy + toy_sin
                     + toy_cos, data = train_data_5)
```

Performance of Predictions

```r
# Make predictions on the testing set
test_data_5$predictions <- predict(linear_model_5, newdata = test_data_5)

# Calculate prediction error metrics
mae_5 <- mean(abs(test_data_5$Cluster.5 - test_data_5$predictions))
```

```r
mse_5 <- mean((test_data_5$Cluster.5 - test_data_5$predictions)^2)
rmse_5 <- sqrt(mse_5)

# Print the results
cat("Mean Absolute Error (MAE):", mae_5, "\n")
```

```
## Mean Absolute Error (MAE): 0.03355503
```

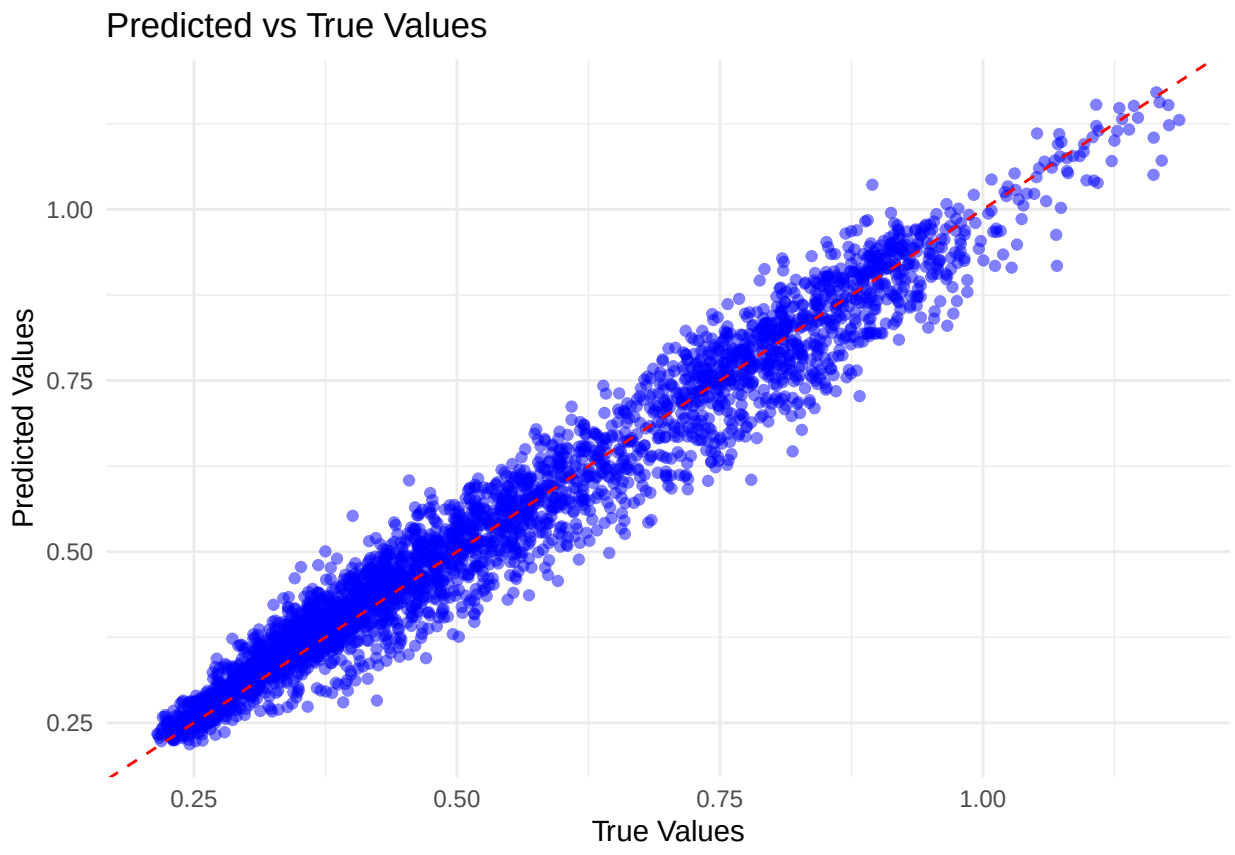
```r
cat("Mean Squared Error (MSE):", mse_5, "\n")
```

```
## Mean Squared Error (MSE): 0.001909648
```

```r
cat("Root Mean Squared Error (RMSE):", rmse_5, "\n")
```

```
## Root Mean Squared Error (RMSE): 0.04369951
```

```r
# Plot predicted vs true values
ggplot(test_data_5, aes(x = Cluster.5, y = predictions)) +
  geom_point(color = 'blue', alpha = 0.5) +
  geom_abline(intercept = 0, slope = 1, color = 'red', linetype = "dashed") +
  labs(title = "Predicted vs True Values",
       x = "True Values",
       y = "Predicted Values") +
  theme_minimal()
```



Linear Regression Model Summary

```
summary(linear_model_5)
```

```
##
## Call:
## lm(formula = Cluster.5 ~ Cluster.5_lag1 + temp + tod_poly1 +
##     tod_poly2 + tod_poly3 + tod_poly4 + weekend_dummy + toy_sin +
##     toy_cos, data = train_data_5)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.139892 -0.024421 -0.002343  0.020704  0.202596
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)     8.670e-02  1.768e-02   4.905 9.45e-07 ***
## Cluster.5_lag1  8.866e-01  4.144e-03 213.968  < 2e-16 ***
## temp           -1.149e-03  1.040e-04 -11.049  < 2e-16 ***
## tod_poly1       4.041e-03  6.185e-04   6.535 6.60e-11 ***
## tod_poly2      -9.191e-05  4.936e-05  -1.862   0.0626 .
## tod_poly3       1.671e-06  1.483e-06   1.126   0.2601
## tod_poly4      -1.001e-08  1.514e-08  -0.661   0.5086
## weekend_dummy   3.885e-03  7.573e-04   5.131 2.93e-07 ***
## toy_sin        -3.745e-02  7.988e-03  -4.688 2.78e-06 ***
## toy_cos        -7.989e-02  1.546e-02  -5.167 2.41e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0387 on 13428 degrees of freedom
## Multiple R-squared:  0.9521, Adjusted R-squared:  0.9521
## F-statistic: 2.966e+04 on 9 and 13428 DF,  p-value: < 2.2e-16
```

Cluster 6

Linear Regression Model for Cluster 6

```
# Split the data into training and testing sets
train_index_6 <- 1:floor(0.8 * nrow(cluster6_trans))
train_data_6 <- cluster6_trans[train_index_6, ]
test_data_6 <- cluster6_trans[-train_index_6, ]

# Fit the linear regression model
linear_model_6 <- lm(Cluster.6 ~ Cluster.6_lag1 + temp + tod_poly1 + tod_poly2
                     + tod_poly3 + tod_poly4 + weekend_dummy + toy_sin
                     + toy_cos, data = train_data_6)
```

Performance of Predictions

```
# Make predictions on the testing set
test_data_6$predictions <- predict(linear_model_6, newdata = test_data_6)
```

```r
# Calculate prediction error metrics
mae_6 <- mean(abs(test_data_6$Cluster.6 - test_data_6$predictions))
mse_6 <- mean((test_data_6$Cluster.6 - test_data_6$predictions)^2)
rmse_6 <- sqrt(mse_6)

# Print the results
cat("Mean Absolute Error (MAE):", mae_6, "\n")
```

```
## Mean Absolute Error (MAE): 0.03879818
```

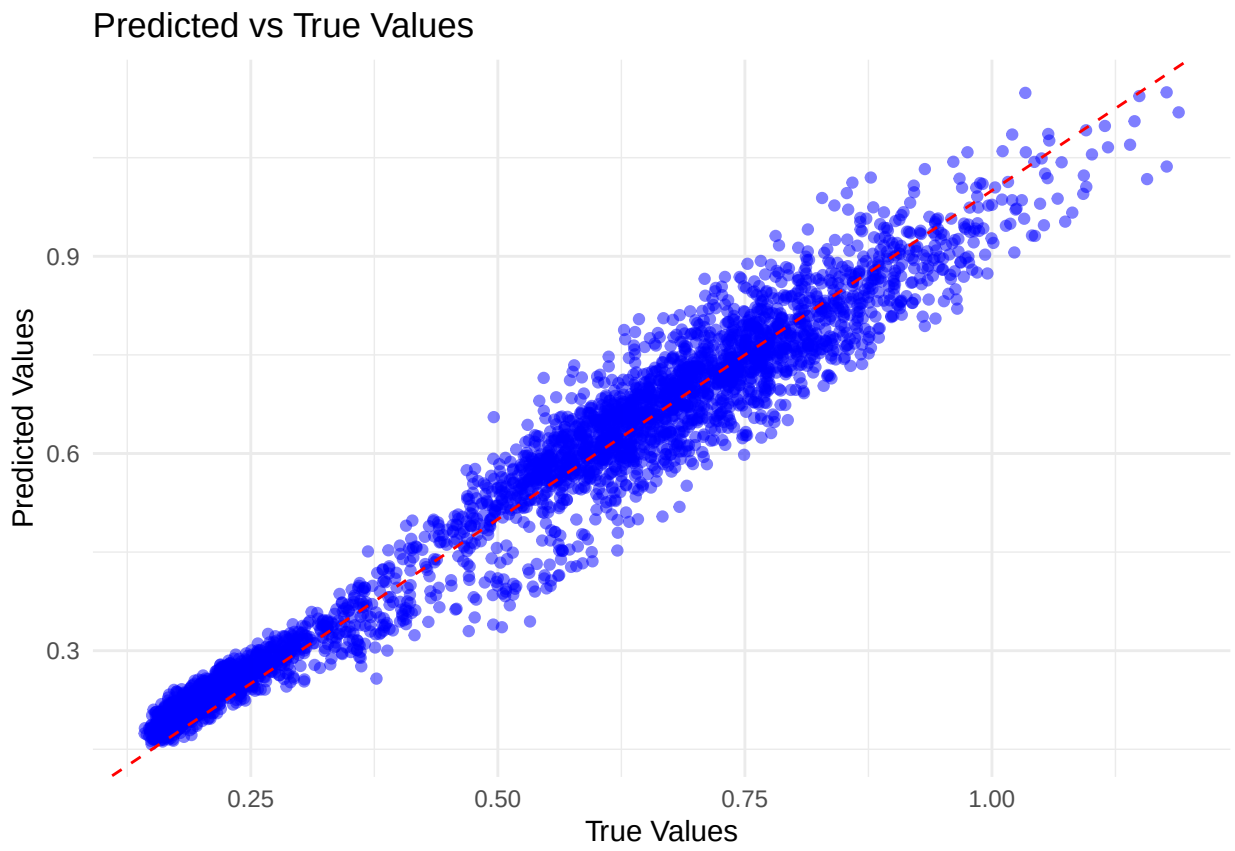
```r
cat("Mean Squared Error (MSE):", mse_6, "\n")
```

```
## Mean Squared Error (MSE): 0.002503867
```

```r
cat("Root Mean Squared Error (RMSE):", rmse_6, "\n")
```

```
## Root Mean Squared Error (RMSE): 0.05003866
```

```r
# Plot predicted vs true values
ggplot(test_data_6, aes(x = Cluster.6, y = predictions)) +
  geom_point(color = 'blue', alpha = 0.5) +
  geom_abline(intercept = 0, slope = 1, color = 'red', linetype = "dashed") +
  labs(title = "Predicted vs True Values",
       x = "True Values",
       y = "Predicted Values") +
  theme_minimal()
```



Linear Regression Model Summary

```
summary(linear_model_6)
```

```
##
## Call:
## lm(formula = Cluster.6 ~ Cluster.6_lag1 + temp + tod_poly1 +
##     tod_poly2 + tod_poly3 + tod_poly4 + weekend_dummy + toy_sin +
##     toy_cos, data = train_data_6)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.213397 -0.027366 -0.004184  0.025137  0.239803
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)     3.241e-02  2.120e-02   1.529 0.126388
## Cluster.6_lag1  8.936e-01  4.408e-03 202.700  < 2e-16 ***
## temp           -1.131e-03  1.308e-04  -8.649  < 2e-16 ***
## tod_poly1       1.288e-02  7.602e-04  16.943  < 2e-16 ***
## tod_poly2      -4.028e-04  7.078e-05  -5.692 1.28e-08 ***
## tod_poly3       4.116e-06  2.202e-06   1.870 0.061538 .
## tod_poly4      -1.317e-08  2.220e-08  -0.593 0.553046
## weekend_dummy   1.425e-03  9.487e-04   1.502 0.133196
## toy_sin        -2.796e-02  9.974e-03  -2.803 0.005074 **
## toy_cos        -6.734e-02  1.937e-02  -3.476 0.000511 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.04909 on 13428 degrees of freedom
## Multiple R-squared:  0.9454, Adjusted R-squared:  0.9453
## F-statistic: 2.582e+04 on 9 and 13428 DF,  p-value: < 2.2e-16
```

Conclusion for Linear Regression Models

```
coefficients_1 <- summary(linear_model_1)$coefficients
coefficients_2 <- summary(linear_model_2)$coefficients
coefficients_3 <- summary(linear_model_3)$coefficients
coefficients_4 <- summary(linear_model_4)$coefficients
coefficients_5 <- summary(linear_model_5)$coefficients
coefficients_6 <- summary(linear_model_6)$coefficients

performance_1 <- data.frame(MAE = mae_1, MSE = mse_1, RMSE = rmse_1)
performance_2 <- data.frame(MAE = mae_2, MSE = mse_2, RMSE = rmse_2)
performance_3 <- data.frame(MAE = mae_3, MSE = mse_3, RMSE = rmse_3)
performance_4 <- data.frame(MAE = mae_4, MSE = mse_4, RMSE = rmse_4)
performance_5 <- data.frame(MAE = mae_5, MSE = mse_5, RMSE = rmse_5)
performance_6 <- data.frame(MAE = mae_6, MSE = mse_6, RMSE = rmse_6)

performance_summary <- rbind(
  data.frame(Cluster = "Cluster 1", performance_1),
  data.frame(Cluster = "Cluster 2", performance_2),
  data.frame(Cluster = "Cluster 3", performance_3),
  data.frame(Cluster = "Cluster 4", performance_4),
  data.frame(Cluster = "Cluster 5", performance_5),
  data.frame(Cluster = "Cluster 6", performance_6)
)

knitr::kable(performance_summary,
             caption = "Performance Metrics for Each Cluster",
             booktabs = TRUE)
```

Table 1: Performance Metrics for Each Cluster

Cluster	MAE	MSE	RMSE
Cluster 1	0.0418657	0.0033436	0.0578236
Cluster 2	0.0392567	0.0027074	0.0520330
Cluster 3	0.0290271	0.0015026	0.0387629
Cluster 4	0.0428198	0.0034555	0.0587839
Cluster 5	0.0335550	0.0019096	0.0436995
Cluster 6	0.0387982	0.0025039	0.0500387

```
coef_summary <- data.frame(
  #Variable = rownames(coefficients_1),
  `Cluster 1` = coefficients_1[, "Estimate"],
  `Cluster 2` = coefficients_2[, "Estimate"],
```

```
  `Cluster 3` = coefficients_3[, "Estimate"],
  `Cluster 4` = coefficients_4[, "Estimate"],
  `Cluster 5` = coefficients_5[, "Estimate"],
  `Cluster 6` = coefficients_6[, "Estimate"]
)

knitr::kable(coef_summary, caption = "Coefficients Summary for Each Cluster",
             booktabs = TRUE)
```

Table 2: Coefficients Summary for Each Cluster

	Cluster.1	Cluster.2	Cluster.3	Cluster.4	Cluster.5	Cluster.6
(Intercept)	-0.0365963	-0.1276174	-0.0266471	0.0348692	0.0867022	0.0324100
Cluster.1_lag1	0.9264920	0.9571412	0.9348833	0.8857962	0.8866198	0.8935789
temp	-0.0009932	-0.0009467	-0.0007840	-0.0010790	-0.0011490	-0.0011315
tod_poly1	0.0229293	0.0378848	0.0148552	0.0222100	0.0040415	0.0128805
tod_poly2	-0.0016023	-0.0032007	-0.0007698	-0.0016599	-0.0000919	-0.0004028
tod_poly3	0.0000506	0.0001053	0.0000209	0.0000514	0.0000017	0.0000041
tod_poly4	-0.0000006	-0.0000011	-0.0000002	-0.0000005	0.0000000	0.0000000
weekend_dummy	0.0015193	0.0011610	0.0015693	0.0016690	0.0038854	0.0014246
toy_sin	-0.0090731	0.0132123	-0.0119092	-0.0257991	-0.0374516	-0.0279551
toy_cos	-0.0357117	0.0051879	-0.0362438	-0.0642703	-0.0798886	-0.0673353

The tables referred to here as Table 1 and Table 2 provide a detailed overview of the initial performance metrics (using MAE (Mean Absolute Error), MSE (Mean Squared Error), and RMSE (Root Mean Squared Error)) and model coefficients for each cluster. These serve as a foundational baseline for assessing the effectiveness of the linear regression models. The primary purpose of these tables is to establish initial values and benchmarks for evaluating model performance. The insights gained from these metrics and coefficients are critical for understanding the model's predictive capabilities and identifying areas for potential improvement.

This foundational information will guide further analysis and model refinement, thereby enhancing our understanding of the underlying structure of the dataset and the effectiveness of the linear regression models.

Bayesian Model

Load aggregated dataset:

```
aggregated_data <- read.csv("AggregatedData1.csv")
```

First, we filter for cluster 1 data only from the aggregated dataset, and use the first 80% of it as training data, and the last 20 as the testing data. Note that we do this and not use random partitioning because that would disturb the time series nature of the data.

```
# use cluster 1 from dataset
cluster1_data <- aggregated_data[, -c(2,3,4,5,6)]

# take first 80% of the data
num_rows <- nrow(cluster1_data)
num_rows_80_percent <- floor(0.8 * num_rows)
cluster1_data_train <- cluster1_data[1:num_rows_80_percent, ]
```

```r
cluster1_data_test <- cluster1_data[(num_rows_80_percent + 1):num_rows, ]

# define the dependent variable and teh explanatory variables
y <- cluster1_data_train$Cluster.1
x4 <- cluster1_data_train$toy                    # time of year
x3 <- as.integer(cluster1_data_train$weekend)    # weekend
x1 <- cluster1_data_train$temp                   # temperature
x2 <- cluster1_data_train$tod                    # time of day
```

Next, we define our model. We assume that

$$y_t \sim Normal(\mu_t, \sigma^2),$$

where

$$\mu_t = \beta_0 + \alpha y_{t-1} + \beta_1 x_{1,t} + \beta_2 x_{2,t} + \beta_3 x_{2,t}^2 + \beta_4 x_{2,t}^3 + \beta_5 x_{2,t}^4 + \beta_6 x_{3,t} + \beta_7 \sin(x_{4,t}) + \beta_8 \cos(x_{4,t}).$$

We need to set priors on α, β_i for all i and since JAGS works with precision instead of variance in the normal distribution, also on $\tau = 1/\sigma^2$. We use the following priors on α, β_i:

$$\beta_i \sim Normal(0, 0.01) \text{ for all } i \in \{0, 8\} \alpha \sim Normal(0, 0.01),$$

where 0.01 is the precision of the parameters (not the variance). As for τ, we use a distribution that is often used in literature as a non informative prior on precision:

$$\tau \sim Gamma(0.01, 0.01).$$

We write the model described above as a string that we feed to JAGS with the data to fit our model.

```r
model_string2 <- "model {
  # Priors for the coefficients
  beta0 ~ dnorm(0, 0.01)
  beta1 ~ dnorm(0, 0.01)
  beta2 ~ dnorm(0, 0.01)
  beta3 ~ dnorm(0, 0.01)
  beta4 ~ dnorm(0, 0.01)
  beta5 ~ dnorm(0, 0.01)
  beta6 ~ dnorm(0, 0.01)
  beta7 ~ dnorm(0, 0.01)
  beta8 ~ dnorm(0, 0.01)
  alpha ~ dnorm(0, 0.01)

  # Prior for the precision (inverse of variance)
  tau ~ dgamma(0.01, 0.01)
  sigma <- 1 / sqrt(tau)

  # Initial value for y[1]
  y[1] ~ dnorm(0, 0.01)

  # Likelihood
  for (t in 2:N) {
    y[t] ~ dnorm(mu[t], tau)
```

```
    mu[t] <- beta0 + alpha * y[t-1] +                                          # intercept and y[t-
             beta1 * x1[t] +                                                   # temp x1
             beta2 * x2[t] + beta3 * x2[t]^2 + beta4 * x2[t]^3 + beta5 * x2[t]^4 +  # time of day x2
             beta6 * x3[t] +                                                   # weekend x3
             beta7 * sin(x4[t]) + beta8 * cos(x4[t])                           # time of year x4
             }

}

"
```

We define a function that sets a seed for the initial values for the chains, for reproducibility.

```
# Define the initial values function
inits <- function(chain) {
  set.seed(12 + chain) # Different seed for each chain
  list(
    .RNG.name ="base::Mersenne-Twister",
    .RNG.seed = 12 + chain
  )
}
```

We fit the model above using 3 chains, a burn-in of 11000 and simulate a posterior sample of size 15000.

```
set.seed(12)

# Data for the model
datalist <- list(N = length(y), x1 = x1, x2 = x2, x3 = x3, x4 = x4, y = y)

# Compile and run with 3 chains, burn-in of 9000 and total sample size of 13000
model <- jags.model(
  file = textConnection(model_string2),
  data = datalist,
  inits = list(inits(4), inits(5), inits(3)), # Seed for initial values for each chain
  n.chains = 3
)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 13439
##    Unobserved stochastic nodes: 11
##    Total graph size: 95395
##
## Initializing model
```

```
update(model, n.iter = 11000)

Nrep = 15000

posterior_sample <- coda.samples(
```

```
  model,
  variable.names = c("tau", "beta0", "beta1", "beta2", "beta3", "beta4",
                     "beta5", "beta6", "beta7", "beta8", "alpha"),
  n.iter = Nrep
)
```

To check for convergence of chains, we use the Gelman–Rubin convergence diagnostic. The Gelman-Rubin diagnostic compares the variance between multiple chains (inter-chain variance) to the variance within each chain (intra-chain variance). The basic idea is to run multiple MCMC chains and then evaluate whether these chains have converged to the same distribution. Mathematically, it can be expressed as:

$$\hat{R} = \frac{\frac{L-1}{L}W + \frac{1}{L}B}{W}$$

where L is the total simulated sample size minus the burn in, B is between chain variance and W is the average within-chain variance. We want the $\hat{R}$ (or psrf) values to be close to 1, generally taking values under 1.2 or more strictly, under 1.1 to indicate convergence. We print the Gelman–Rubin convergence diagnostic below for each variable.

```
gelman.diag(posterior_sample)
```

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## alpha       1.00       1.01
## beta0       1.13       1.35
## beta1       1.01       1.05
## beta2       1.02       1.07
## beta3       1.03       1.10
## beta4       1.04       1.12
## beta5       1.04       1.14
## beta6       1.00       1.00
## beta7       1.09       1.27
## beta8       1.11       1.33
## tau         1.00       1.01
##
## Multivariate psrf
##
## 1.14
```

We can see that each individual point estimate, as well as the multivariate psrf is under 1.2, so this confirms convergence.

We see the summary of the posterior samples below.

```
summary(posterior_sample)
```

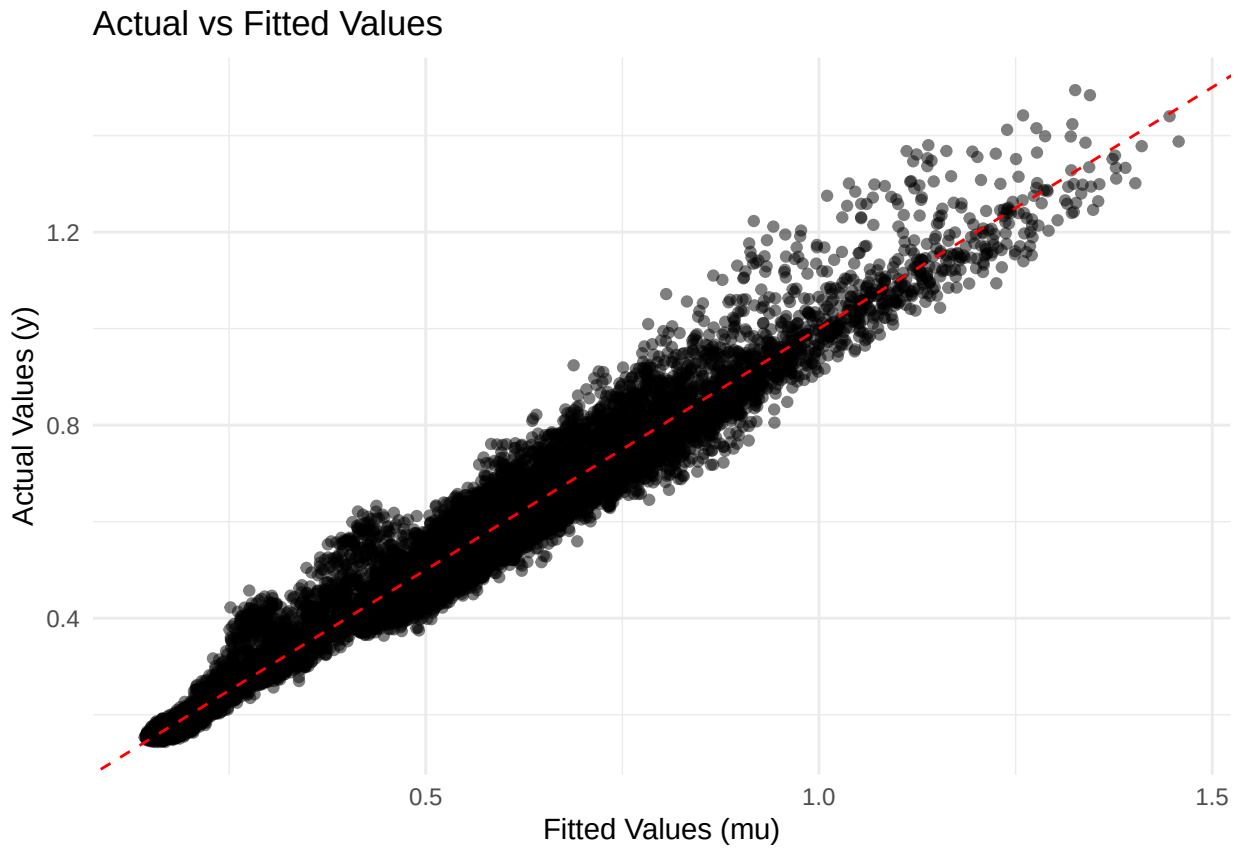
```
##
## Iterations = 11001:26000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 15000
```

```
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean        SD  Naive SE Time-series SE
## alpha  9.102e-01 4.194e-03 1.977e-05      2.809e-04
## beta0 -1.085e-02 1.489e-02 7.021e-05      3.639e-03
## beta1 -1.365e-03 1.397e-04 6.585e-07      6.792e-06
## beta2  1.419e-02 1.503e-03 7.085e-06      5.955e-04
## beta3 -8.104e-04 1.356e-04 6.392e-07      6.396e-05
## beta4  2.626e-05 4.167e-06 1.964e-08      1.965e-06
## beta5 -3.226e-07 4.172e-08 1.967e-10      0.000e+00
## beta6  2.068e-03 9.443e-04 4.451e-06      7.112e-06
## beta7 -4.606e-03 7.383e-03 3.481e-05      9.501e-04
## beta8 -3.522e-02 1.374e-02 6.478e-05      3.132e-03
## tau    4.264e+02 5.766e+00 2.718e-02      2.428e-01
##
## 2. Quantiles for each variable:
##
##             2.5%        25%        50%        75%      97.5%
## alpha  9.018e-01  9.073e-01  9.104e-01  9.132e-01  9.180e-01
## beta0 -5.372e-02 -1.851e-02 -9.592e-03 -1.401e-03  1.468e-02
## beta1 -1.639e-03 -1.461e-03 -1.365e-03 -1.270e-03 -1.095e-03
## beta2  1.137e-02  1.293e-02  1.433e-02  1.562e-02  1.637e-02
## beta3 -9.973e-04 -9.389e-04 -8.158e-04 -6.973e-04 -5.581e-04
## beta4  1.864e-05  2.256e-05  2.646e-05  3.023e-05  3.200e-05
## beta5 -3.816e-07 -3.616e-07 -3.246e-07 -2.847e-07 -2.473e-07
## beta6  2.009e-04  1.432e-03  2.066e-03  2.708e-03  3.906e-03
## beta7 -1.805e-02 -9.664e-03 -4.808e-03 -1.653e-04  1.265e-02
## beta8 -5.900e-02 -4.455e-02 -3.589e-02 -2.763e-02  1.084e-03
## tau    4.150e+02  4.225e+02  4.264e+02  4.303e+02  4.377e+02
```

Next, we take the posterior means of the regression parameters as estimated by the model and calculate $\hat{\mu}_t$, which is the mean of the distribution of y_t at time t, as estimated by our model. We can then plot the means against the true values of y_t to see how well the model fits the data.

```r
stat <- as.vector(unlist(summary(posterior_sample)[1])[1:11])
mu <- vector(length = length(y))
mu[1] <- y[1]

# Compare fitted mu with true values of y_t
for (t in 2:length(y)) {
  mu[t] <- stat[2] + stat[1] * y[t-1] +
             stat[3] * x1[t] +
             stat[4] * x2[t] + stat[5] * x2[t]^2 + stat[6] * x2[t]^3 + stat[7] * x2[t]^4 +
             stat[8] * x3[t] +
             stat[9] * sin(x4[t]) + stat[10] * cos(x4[t])
}

# Create a time variable
t <- 1:length(y)

# Create a data frame
data <- data.frame(
```

```
  t = t,
  y = y,
  mu = mu
)

# Load ggplot2
library(ggplot2)

# Create the scatter plot
ggplot(data, aes(x = mu, y = y)) +
  geom_point(alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  labs(title = "Actual vs Fitted Values",
       x = "Fitted Values (mu)",
       y = "Actual Values (y)") +
  theme_minimal()
```



We can see that the model fits the data well, as the means agree with the true values.

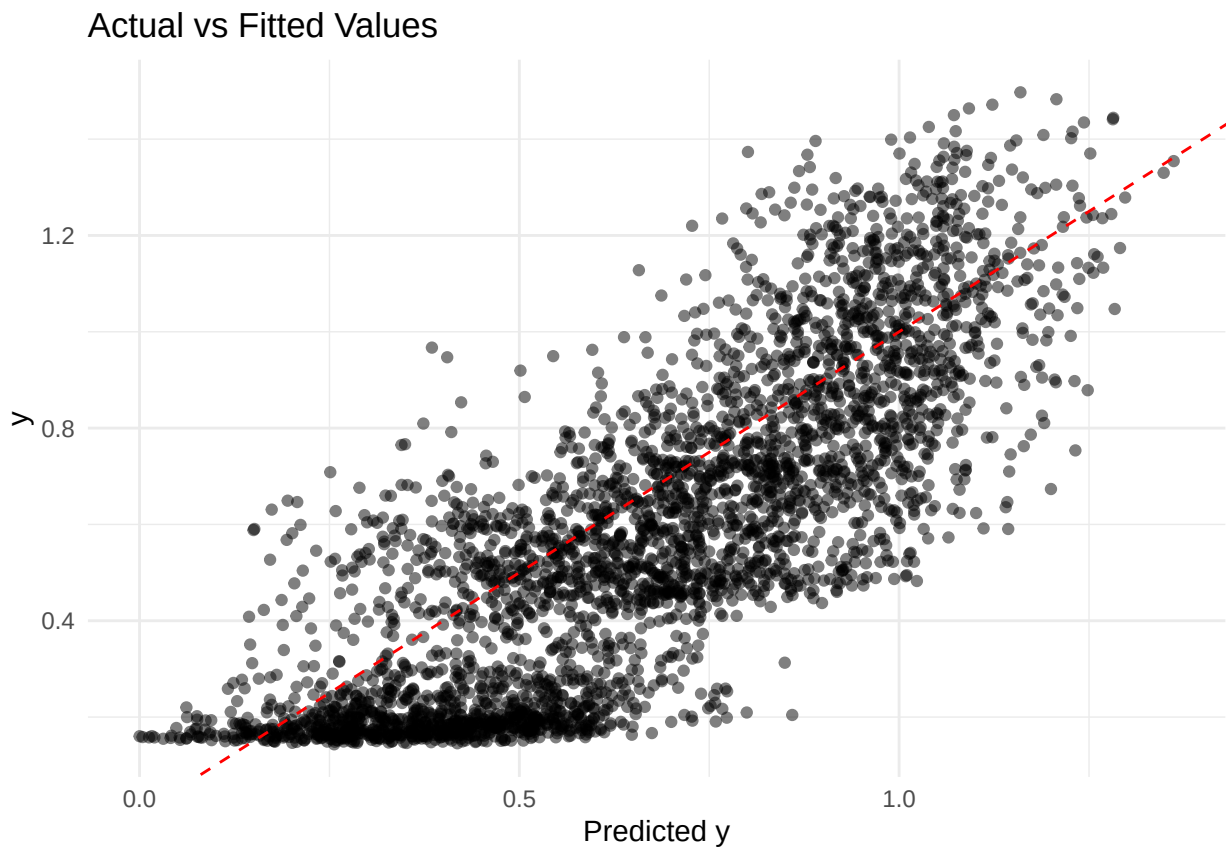
Now we want to find the predicted values of y for the test set. This allows us to assess the model performance using performance metrics like MSE.

```
set.seed(123)
# Extract the test data vectors
x1_test <- cluster1_data_test$temp
x2_test <- cluster1_data_test$tod
```

```r
x3_test <- as.integer(cluster1_data_test$weekend)
x4_test <- cluster1_data_test$toy
y_test <- cluster1_data_test$Cluster.1
N_test <- length(y_test)

# Initialize a vector to store the predicted values
predicted_y <- numeric(N_test)

# Sigma is 1/sqrt(tau), where tau is the precision
sigma <- 1 / sqrt(stat[11])

# Make predictions using the posterior samples
mu_test <- numeric(N_test)
mu_test[1] <- y_test[1]
predicted_y[1] <- y_test[1]  # Initialize with the first observed value

for (t in 2:N_test) {

    mu_test[t] <- stat[2] + stat[1] * predicted_y[t-1] +
              stat[3] * x1_test[t] +
              stat[4] * x2_test[t] + stat[5] * x2_test[t]^2 + stat[6] * x2_test[t]^3+ stat[7]*x2_test[t]
              stat[8] * x3_test[t] +
              stat[9] * sin(x4_test[t]) + stat[10] * cos(x4_test[t])

    predicted_y[t] <- rnorm(1, mean = mu_test[t], sd = sigma)
}
```

We plot the predicted vs. fitted values:

```r
# Create a data frame
t <- 1:length(y_test)

data_test <- data.frame(
  t = t,
  y = y_test,
  predicted_val = predicted_y
)

# Create the scatter plot
ggplot(data_test, aes(x = predicted_val, y = y)) +
  geom_point(alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  labs(title = "Actual vs Fitted Values",
       x = "Predicted y",
       y = "y") +
  theme_minimal()
```



Calculate MSE:

```
# Calculate Mean Squared Error (MSE)
mse <- mean((predicted_y - y_test)^2)

# Print the results
cat("MSE:", mse, "\n")
```

```
## MSE: 0.04399398
```

Repeat for cluster 2:

```
# use cluster 2 from dataset
cluster2_data <- aggregated_data[, -c(1,3,4,5,6)]

# take first 80% of the data
num_rows <- nrow(cluster2_data)
num_rows_80_percent <- floor(0.8 * num_rows)
cluster2_data_train <- cluster2_data[1:num_rows_80_percent, ]
cluster2_data_test <- cluster2_data[(num_rows_80_percent + 1):num_rows, ]

# define the dependent variable and teh explanatory variables
y <- cluster2_data_train$Cluster.2
x4 <- cluster2_data_train$toy        # time of year
x3 <- cluster2_data_train$weekend    # weekend
```

```
x1 <- cluster2_data_train$temp       # temperature
x2 <- cluster2_data_train$tod        # time of day
```

Fit model with burn in 11000 and posterior sample size of 15000.

```
set.seed(12)

# Data for the model
datalist <- list(N = length(y), x1 = x1, x2 = x2, x3 = x3, x4 = x4, y = y)

# Compile and run with 3 chains, burn-in of 9000 and total sample size of 13000
model <- jags.model(
  file = textConnection(model_string2),
  data = datalist,
  inits = list(inits(4), inits(5), inits(3)), # Seed for initial values for each chain
  n.chains = 3
)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 13439
##    Unobserved stochastic nodes: 11
##    Total graph size: 95196
##
## Initializing model
```

```
update(model, n.iter = 11000)

Nrep = 15000

posterior_sample2 <- coda.samples(
  model,
  variable.names = c("tau", "beta0", "beta1", "beta2", "beta3", "beta4",
                     "beta5", "beta6", "beta7", "beta8", "alpha"),
  n.iter = Nrep
)
```

Check convergence:

```
gelman.diag(posterior_sample2)
```

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## alpha       1.00       1.01
## beta0       1.13       1.35
## beta1       1.01       1.03
## beta2       1.00       1.02
## beta3       1.01       1.02
## beta4       1.01       1.03
```

```
## beta5       1.01       1.03
## beta6       1.00       1.00
## beta7       1.07       1.21
## beta8       1.09       1.28
## tau         1.00       1.01
##
## Multivariate psrf
##
## 1.12
```

We can see that each individual point estimate, as well as the multivariate psrf is under 1.2, so this confirms convergence.

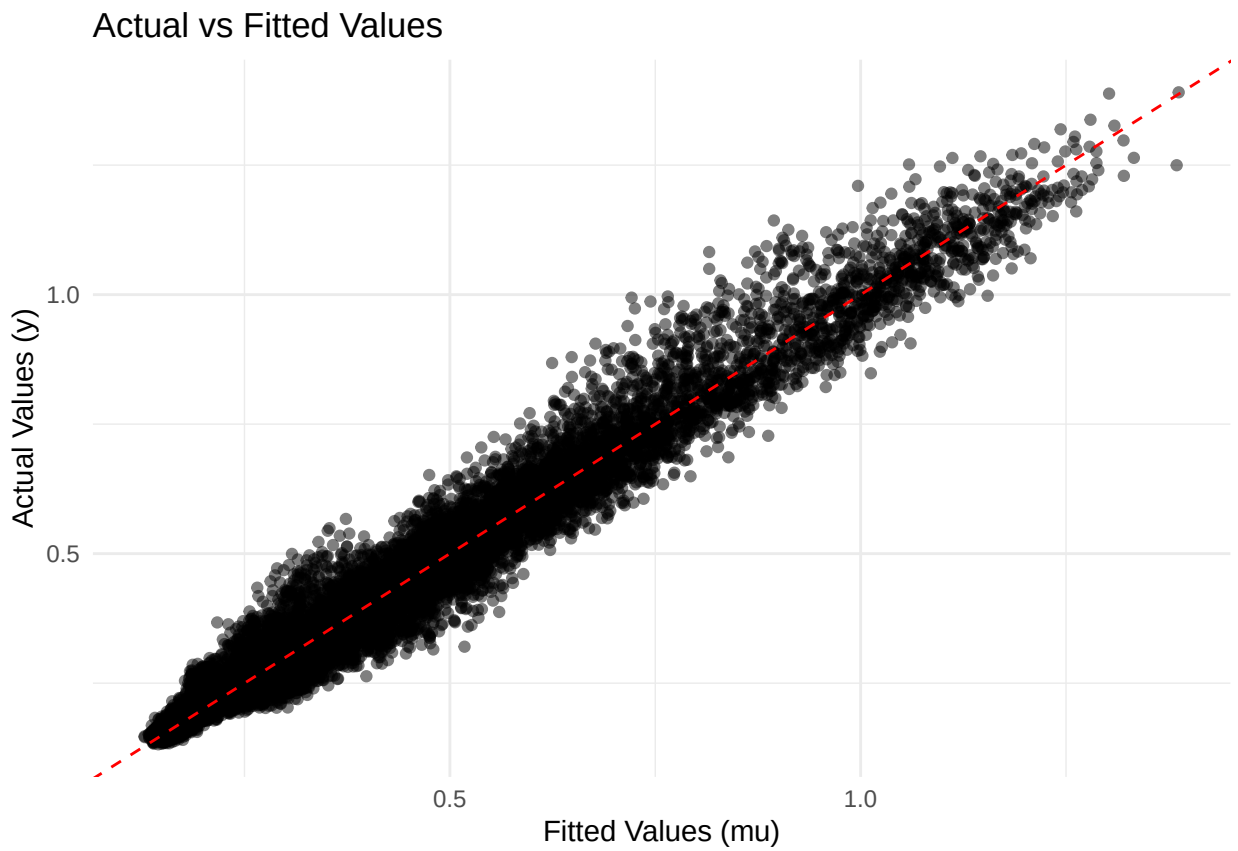
We print the summary of the posterior samples below.

```
summary(posterior_sample2)
```

```
##
## Iterations = 11001:26000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 15000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean        SD  Naive SE Time-series SE
## alpha  9.381e-01 4.658e-03 2.196e-05      3.453e-04
## beta0 -1.018e-01 1.466e-02 6.912e-05      3.526e-03
## beta1 -1.647e-03 1.792e-04 8.449e-07      1.416e-05
## beta2  2.121e-02 3.115e-03 1.469e-05      1.412e-03
## beta3 -1.687e-03 2.825e-04 1.332e-06      1.331e-04
## beta4  5.786e-05 8.862e-06 4.178e-08      4.171e-06
## beta5 -6.631e-07 9.051e-08 4.267e-10      0.000e+00
## beta6  2.075e-03 9.345e-04 4.405e-06      9.611e-06
## beta7  3.187e-02 8.033e-03 3.787e-05      1.130e-03
## beta8  2.498e-02 1.398e-02 6.592e-05      3.108e-03
## tau    4.515e+02 1.310e+01 6.174e-02      3.845e+00
##
## 2. Quantiles for each variable:
##
##             2.5%        25%        50%        75%      97.5%
## alpha  9.289e-01  9.349e-01  9.383e-01  9.415e-01  9.466e-01
## beta0 -1.437e-01 -1.093e-01 -1.008e-01 -9.248e-02 -7.622e-02
## beta1 -1.994e-03 -1.776e-03 -1.642e-03 -1.515e-03 -1.316e-03
## beta2  1.557e-02  1.863e-02  2.163e-02  2.407e-02  2.570e-02
## beta3 -2.087e-03 -1.939e-03 -1.720e-03 -1.449e-03 -1.170e-03
## beta4  4.168e-05  5.029e-05  5.863e-05  6.578e-05  7.045e-05
## beta5 -7.916e-07 -7.438e-07 -6.689e-07 -5.850e-07 -4.985e-07
## beta6  2.341e-04  1.442e-03  2.073e-03  2.709e-03  3.907e-03
## beta7  1.671e-02  2.605e-02  3.207e-02  3.762e-02  4.719e-02
## beta8 -4.801e-04  1.509e-02  2.475e-02  3.397e-02  5.660e-02
## tau    4.257e+02  4.415e+02  4.530e+02  4.618e+02  4.731e+02
```

Next, we take the posterior means of the regression parameters as estimated by the model and calculate $\hat{\mu}_t$, which is the mean of the distribution of y_t at time t, as estimated by our model. We can then plot the means against the true values of y_t to see how well the model fits the data.

```
stat <- as.vector(unlist(summary(posterior_sample2)[1])[1:11])
mu <- vector(length = length(y))
mu[1] <- y[1]

# Compare fitted mu with true values of y_t
for (t in 2:length(y)) {
  mu[t] <- stat[2] + stat[1] * y[t-1] +
             stat[3] * x1[t] +
             stat[4] * x2[t] + stat[5] * x2[t]^2 + stat[6] * x2[t]^3 + stat[7] * x2[t]^4 +
             stat[8] * x3[t] +
             stat[9] * sin(x4[t]) + stat[10] * cos(x4[t])
}

# Create a time variable
t <- 1:length(y)

# Create a data frame
data <- data.frame(
  t = t,
  y = y,
  mu = mu
)

# Create the scatter plot
ggplot(data, aes(x = mu, y = y)) +
  geom_point(alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  labs(title = "Actual vs Fitted Values",
       x = "Fitted Values (mu)",
       y = "Actual Values (y)") +
  theme_minimal()
```



We can see that the model fits the data well, as the means agree with the true values.

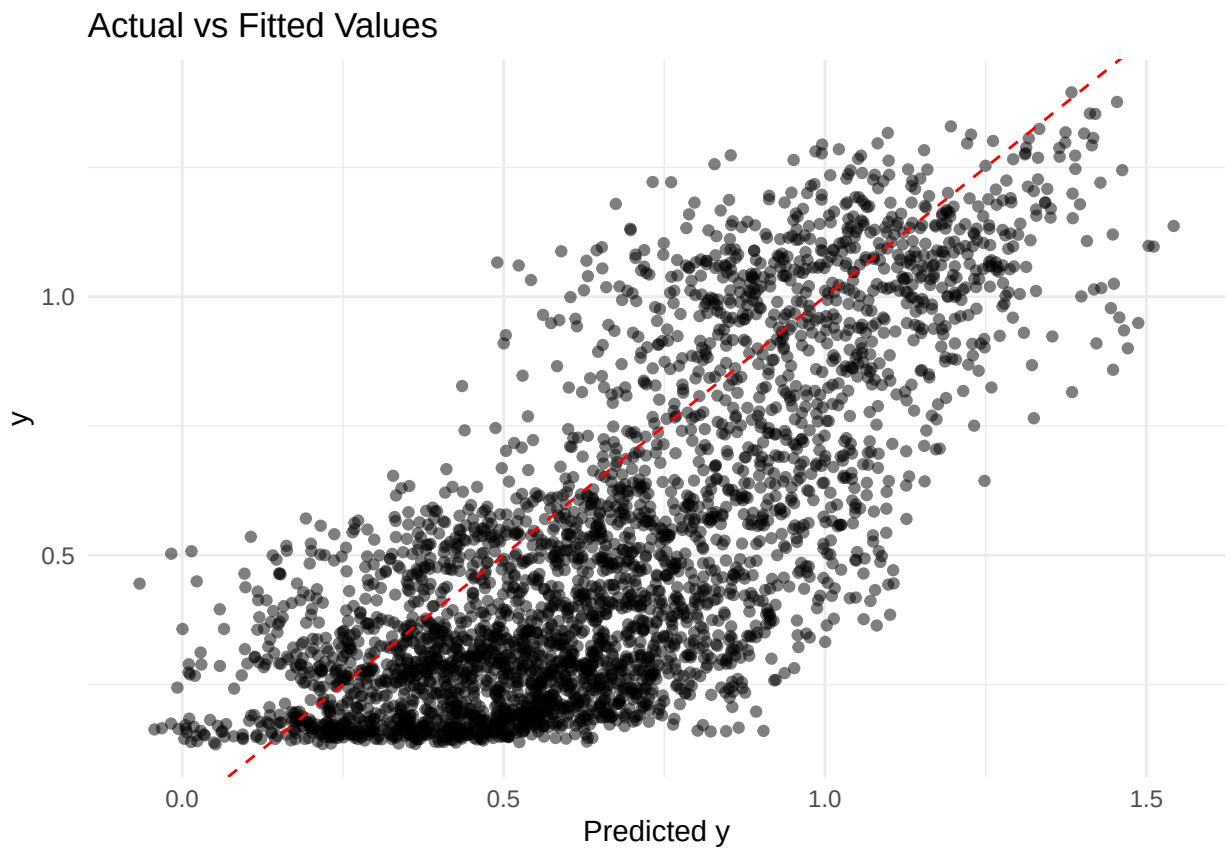
Now we want to find the predicted values of y for the test set. This allows us to assess the model performance using performance metrics like MSE.

```
set.seed(12)
# Extract the test data vectors
x1_test <- cluster2_data_test$temp
x2_test <- cluster2_data_test$tod
x3_test <- as.integer(cluster2_data_test$weekend)
x4_test <- cluster2_data_test$toy
y_test <- cluster2_data_test$Cluster.2
N_test <- length(y_test)

# Initialize a vector to store the predicted values
predicted_y <- numeric(N_test)

# Sigma is 1/sqrt(tau), where tau is the precision
sigma <- 1 / sqrt(stat[11])

# Make predictions using the posterior samples
mu_test <- numeric(N_test)
mu_test[1] <- y_test[1]
predicted_y[1] <- y_test[1]  # Initialize with the first observed value

for (t in 2:N_test) {
```

```
    mu_test[t] <- stat[2] + stat[1] * predicted_y[t-1] +
              stat[3] * x1_test[t] +
              stat[4] * x2_test[t] + stat[5] * x2_test[t]^2 + stat[6] * x2_test[t]^3 + stat[7]*x2_test[t]
              stat[8] * x3_test[t] +
              stat[9] * sin(x4_test[t]) + stat[10] * cos(x4_test[t])

    predicted_y[t] <- rnorm(1, mean = mu_test[t], sd = sigma)
}
```

We plot the predicted vs. fitted values:

```
# Create a data frame
t <- 1:length(y_test)

data_test <- data.frame(
  t = t,
  y = y_test,
  predicted_val = predicted_y
)

# Create the scatter plot
ggplot(data_test, aes(x = predicted_val, y = y)) +
  geom_point(alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  labs(title = "Actual vs Fitted Values",
       x = "Predicted y",
       y = "y") +
  theme_minimal()
```



Calculate MSE:

```
# Calculate Mean Squared Error (MSE)
mse <- mean((predicted_y - y_test)^2)

# Print the results
cat("MSE:", mse, "\n")
```

```
## MSE: 0.06962895
```

Repeat for cluster 3:

```
# use cluster 3 from dataset
cluster3_data <- aggregated_data[, -c(1,2,4,5,6)]

# take first 80% of the data
num_rows <- nrow(cluster3_data)
num_rows_80_percent <- floor(0.8 * num_rows)
cluster3_data_train <- cluster3_data[1:num_rows_80_percent, ]
cluster3_data_test <- cluster3_data[(num_rows_80_percent + 1):num_rows, ]

# define the dependent variable and the explanatory variables
y <- cluster3_data_train$Cluster.3
x4 <- cluster3_data_train$toy        # time of year
x3 <- cluster3_data_train$weekend    # weekend
```

```r
x1 <- cluster3_data_train$temp       # temperature
x2 <- cluster3_data_train$tod        # time of day
```

Fit model with burn in 12000 and posterior sample size of 16000.

```r
set.seed(12)

# Data for the model
datalist <- list(N = length(y), x1 = x1, x2 = x2, x3 = x3, x4 = x4, y = y)

# Compile and run with 3 chains, burn-in of 9000 and total sample size of 13000
model <- jags.model(
  file = textConnection(model_string2),
  data = datalist,
  inits = list(inits(4), inits(5), inits(3)), # Seed for initial values for each chain
  n.chains = 3
)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 13439
##    Unobserved stochastic nodes: 11
##    Total graph size: 95400
##
## Initializing model
```

```r
update(model, n.iter = 12000)

Nrep = 16000

posterior_sample3 <- coda.samples(
  model,
  variable.names = c("tau", "beta0", "beta1", "beta2", "beta3", "beta4",
                     "beta5", "beta6", "beta7", "beta8", "alpha"),
  n.iter = Nrep
)
```

Check convergence:

```r
gelman.diag(posterior_sample3)
```

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## alpha       1.01       1.01
## beta0       1.07       1.15
## beta1       1.01       1.04
## beta2       1.06       1.19
## beta3       1.08       1.25
## beta4       1.10       1.32
```

```
## beta5       1.12       1.36
## beta6       1.00       1.00
## beta7       1.06       1.15
## beta8       1.07       1.17
## tau         1.00       1.00
##
## Multivariate psrf
##
## 1.12
```

We can see that each individual point estimate, as well as the multivariate psrf is under 1.2, so this confirms convergence.

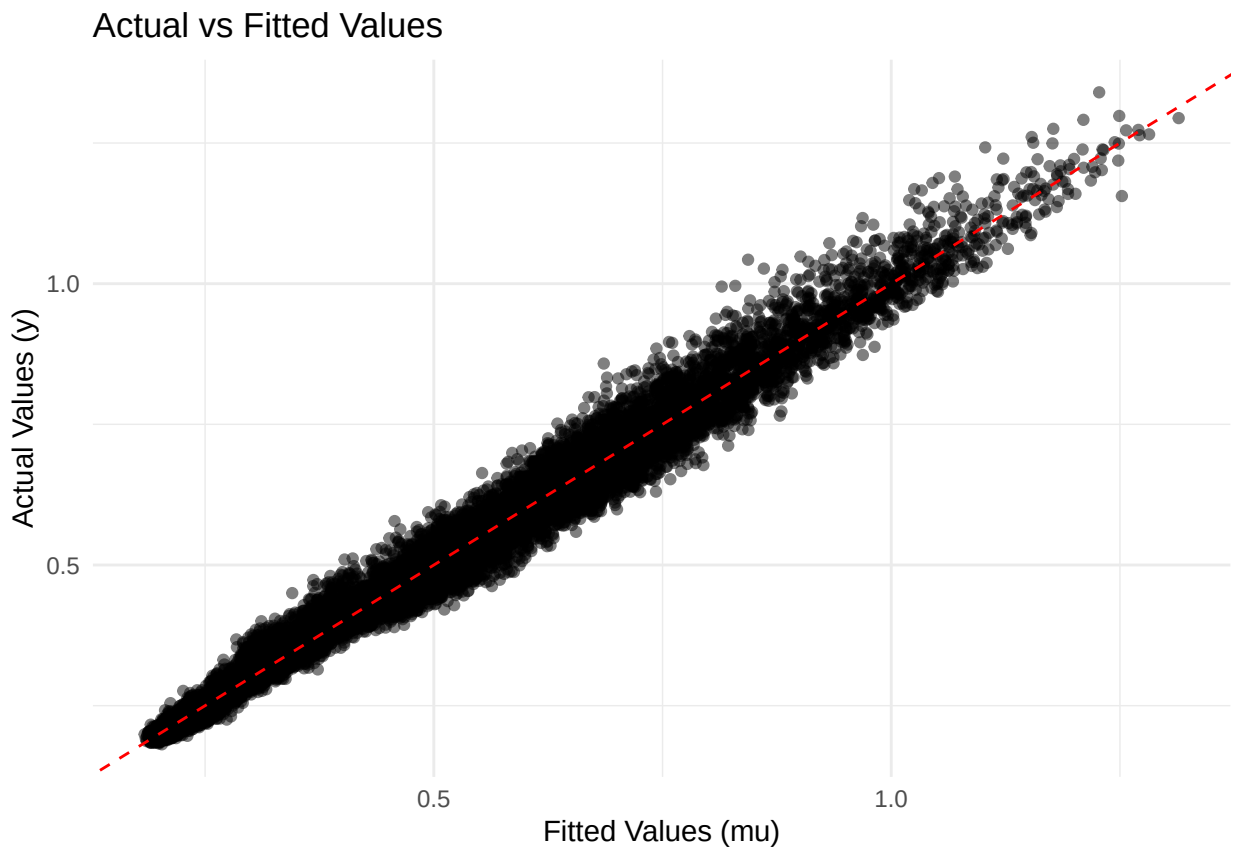
We print the summary of the posterior samples below.

```
summary(posterior_sample3)
```

```
##
## Iterations = 12001:28000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 16000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean        SD  Naive SE Time-series SE
## alpha  9.149e-01 3.522e-03 1.607e-05      2.764e-04
## beta0  1.327e-02 1.123e-02 5.126e-05      3.081e-03
## beta1 -9.676e-04 8.540e-05 3.898e-07      3.130e-06
## beta2  9.879e-03 6.201e-04 2.830e-06      1.915e-04
## beta3 -3.502e-04 5.227e-05 2.386e-07      2.457e-05
## beta4  8.843e-06 1.537e-06 7.017e-09      7.151e-07
## beta5 -1.095e-07 1.514e-08 6.908e-11      0.000e+00
## beta6  2.280e-03 6.417e-04 2.929e-06      5.687e-06
## beta7 -1.949e-02 5.129e-03 2.341e-05      7.407e-04
## beta8 -5.395e-02 9.761e-03 4.455e-05      2.489e-03
## tau    9.318e+02 1.149e+01 5.243e-02      9.448e-02
##
## 2. Quantiles for each variable:
##
##             2.5%        25%        50%        75%      97.5%
## alpha  9.079e-01  9.124e-01  9.150e-01  9.174e-01  9.216e-01
## beta0 -1.753e-02  7.886e-03  1.392e-02  2.031e-02  3.306e-02
## beta1 -1.137e-03 -1.025e-03 -9.663e-04 -9.099e-04 -8.033e-04
## beta2  8.760e-03  9.314e-03  9.957e-03  1.040e-02  1.090e-02
## beta3 -4.342e-04 -3.976e-04 -3.594e-04 -3.018e-04 -2.558e-04
## beta4  6.174e-06  7.484e-06  9.208e-06  1.024e-05  1.121e-05
## beta5 -1.322e-07 -1.224e-07 -1.127e-07 -9.613e-08 -8.359e-08
## beta6  1.014e-03  1.847e-03  2.280e-03  2.716e-03  3.535e-03
## beta7 -2.841e-02 -2.296e-02 -1.978e-02 -1.668e-02 -6.496e-03
## beta8 -7.058e-02 -6.063e-02 -5.450e-02 -4.902e-02 -2.744e-02
## tau    9.093e+02  9.240e+02  9.317e+02  9.395e+02  9.545e+02
```

Next, we take the posterior means of the regression parameters as estimated by the model and calculate $\hat{\mu}_t$, which is the mean of the distribution of y_t at time t, as estimated by our model. We can then plot the means against the true values of y_t to see how well the model fits the data.

```
stat <- as.vector(unlist(summary(posterior_sample3)[1])[1:11])
mu <- vector(length = length(y))
mu[1] <- y[1]

# Compare fitted mu with true values of y_t
for (t in 2:length(y)) {
  mu[t] <- stat[2] + stat[1] * y[t-1] +
             stat[3] * x1[t] +
             stat[4] * x2[t] + stat[5] * x2[t]^2 + stat[6] * x2[t]^3 + stat[7] * x2[t]^4 +
             stat[8] * x3[t] +
             stat[9] * sin(x4[t]) + stat[10] * cos(x4[t])
}

# Create a time variable
t <- 1:length(y)

# Create a data frame
data <- data.frame(
  t = t,
  y = y,
  mu = mu
)

# Create the scatter plot
ggplot(data, aes(x = mu, y = y)) +
  geom_point(alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  labs(title = "Actual vs Fitted Values",
       x = "Fitted Values (mu)",
       y = "Actual Values (y)") +
  theme_minimal()
```



We can see that the model fits the data well, as the means agree with the true values.

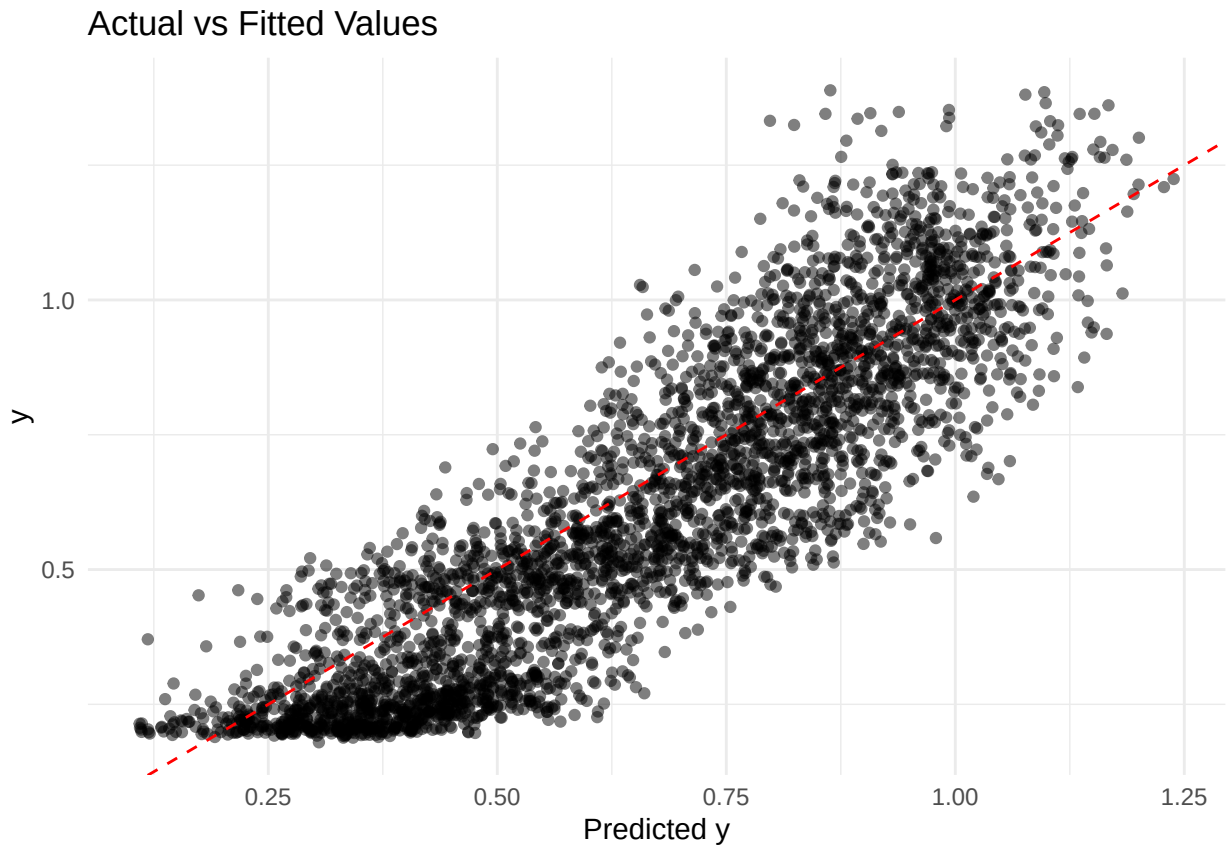
Now we want to find the predicted values of y for the test set. This allows us to assess the model performance using performance metrics like MSE.

```
set.seed(12)
# Extract the test data vectors
x1_test <- cluster3_data_test$temp
x2_test <- cluster3_data_test$tod
x3_test <- as.integer(cluster3_data_test$weekend)
x4_test <- cluster3_data_test$toy
y_test <- cluster3_data_test$Cluster.3
N_test <- length(y_test)

# Initialize a vector to store the predicted values
predicted_y <- numeric(N_test)

# Sigma is 1/sqrt(tau), where tau is the precision
sigma <- 1 / sqrt(stat[11])

# Make predictions using the posterior samples
mu_test <- numeric(N_test)
mu_test[1] <- y_test[1]
predicted_y[1] <- y_test[1]  # Initialize with the first observed value

for (t in 2:N_test) {
```

```
    mu_test[t] <- stat[2] + stat[1] * predicted_y[t-1] +
              stat[3] * x1_test[t] +
              stat[4] * x2_test[t] + stat[5] * x2_test[t]^2 + stat[6] * x2_test[t]^3 + stat[7]*x2_test[t]
              stat[8] * x3_test[t] +
              stat[9] * sin(x4_test[t]) + stat[10] * cos(x4_test[t])

    predicted_y[t] <- rnorm(1, mean = mu_test[t], sd = sigma)
}
```

We plot the predicted vs. fitted values:

```
# Create a data frame
t <- 1:length(y_test)

data_test <- data.frame(
  t = t,
  y = y_test,
  predicted_val = predicted_y
)

# Create the scatter plot
ggplot(data_test, aes(x = predicted_val, y = y)) +
  geom_point(alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  labs(title = "Actual vs Fitted Values",
       x = "Predicted y",
       y = "y") +
  theme_minimal()
```



Calculate MSE:

```
# Calculate Mean Squared Error (MSE)
mse <- mean((predicted_y - y_test)^2)

# Print the results
cat("MSE:", mse, "\n")
```

```
## MSE: 0.02181588
```

Repeat for cluster 4:

```
# use cluster 4 from dataset
cluster4_data <- aggregated_data[, -c(1,2,3,5,6)]

# take first 80% of the data
num_rows <- nrow(cluster4_data)
num_rows_80_percent <- floor(0.8 * num_rows)
cluster4_data_train <- cluster4_data[1:num_rows_80_percent, ]
cluster4_data_test <- cluster4_data[(num_rows_80_percent + 1):num_rows, ]

# define the dependent variable and the explanatory variables
y <- cluster4_data_train$Cluster.4
x4 <- cluster4_data_train$toy        # time of year
x3 <- cluster4_data_train$weekend    # weekend
```

```
x1 <- cluster4_data_train$temp       # temperature
x2 <- cluster4_data_train$tod        # time of day
```

Fit model with burn in 11000 and posterior sample size of 15000.

```
set.seed(12)

# Data for the model
datalist <- list(N = length(y), x1 = x1, x2 = x2, x3 = x3, x4 = x4, y = y)

# Compile and run with 3 chains, burn-in of 9000 and total sample size of 13000
model <- jags.model(
  file = textConnection(model_string2),
  data = datalist,
  inits = list(inits(4), inits(5), inits(3)), # Seed for initial values for each chain
  n.chains = 3
)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 13439
##    Unobserved stochastic nodes: 11
##    Total graph size: 95023
##
## Initializing model
```

```
update(model, n.iter = 11000)

Nrep = 15000

posterior_sample4 <- coda.samples(
  model,
  variable.names = c("tau", "beta0", "beta1", "beta2", "beta3", "beta4",
                     "beta5", "beta6", "beta7", "beta8", "alpha"),
  n.iter = Nrep
)
```

Check convergence:

```
gelman.diag(posterior_sample4)
```

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## alpha       1.00       1.00
## beta0       1.11       1.30
## beta1       1.01       1.05
## beta2       1.03       1.10
## beta3       1.04       1.13
## beta4       1.04       1.15
```

```
## beta5       1.05       1.16
## beta6       1.00       1.00
## beta7       1.08       1.24
## beta8       1.10       1.29
## tau         1.00       1.01
##
## Multivariate psrf
##
## 1.12
```

We can see that each individual point estimate, as well as the multivariate psrf is under 1.2, so this confirms convergence.

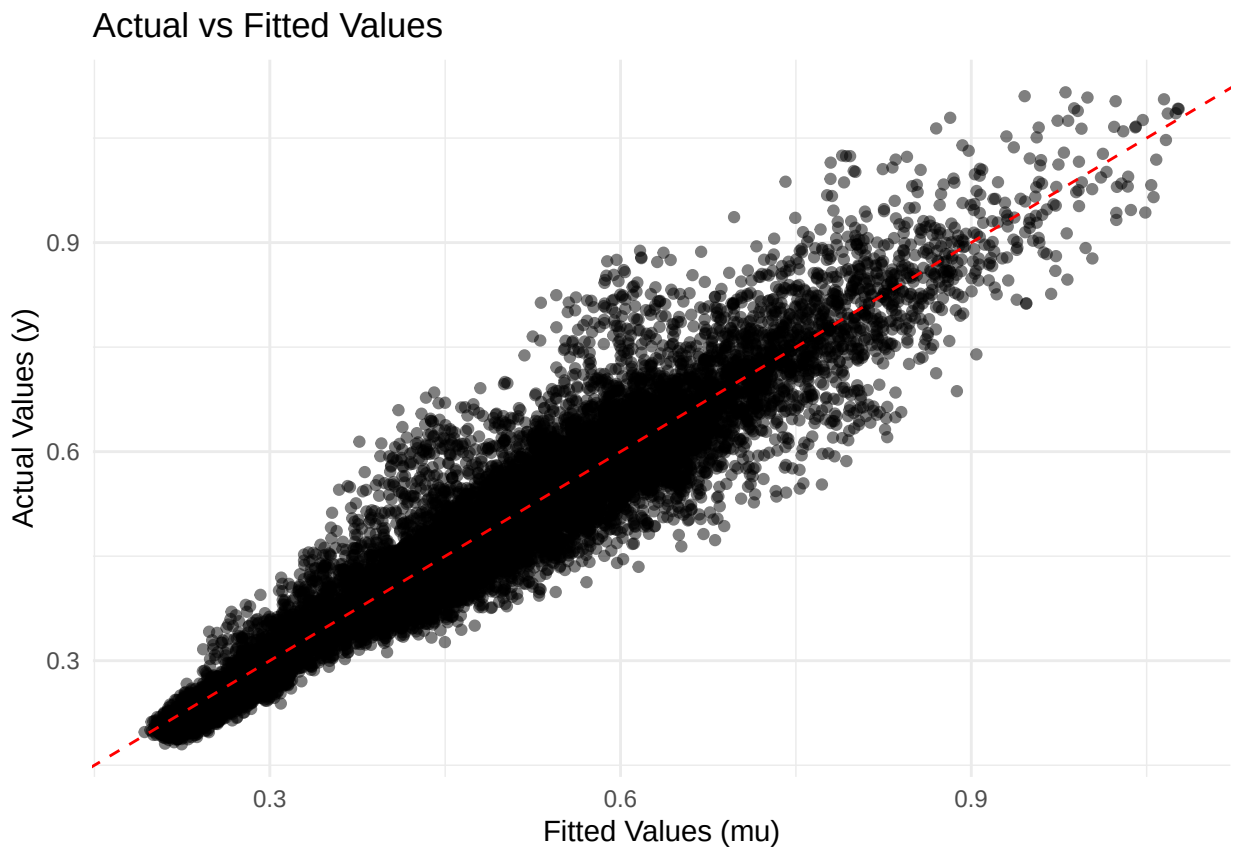
We print the summary of the posterior samples below.

```
summary(posterior_sample4)
```

```
##
## Iterations = 11001:26000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 15000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean        SD  Naive SE Time-series SE
## alpha  8.766e-01 4.214e-03 1.986e-05      1.467e-04
## beta0  4.540e-02 1.613e-02 7.606e-05      3.579e-03
## beta1 -1.417e-03 1.520e-04 7.164e-07      6.750e-06
## beta2  1.417e-02 1.461e-03 6.889e-06      5.465e-04
## beta3 -9.136e-04 1.350e-04 6.362e-07      6.310e-05
## beta4  2.785e-05 4.271e-06 2.013e-08      2.008e-06
## beta5 -3.088e-07 4.371e-08 2.060e-10      0.000e+00
## beta6  1.818e-03 1.044e-03 4.923e-06      6.788e-06
## beta7 -1.609e-02 8.397e-03 3.958e-05      1.106e-03
## beta8 -5.310e-02 1.552e-02 7.317e-05      3.389e-03
## tau    3.428e+02 4.503e+00 2.123e-02      1.414e-01
##
## 2. Quantiles for each variable:
##
##             2.5%        25%        50%        75%      97.5%
## alpha  8.683e-01  8.738e-01  8.767e-01  8.795e-01  8.848e-01
## beta0  4.794e-04  3.685e-02  4.675e-02  5.605e-02  7.310e-02
## beta1 -1.714e-03 -1.522e-03 -1.418e-03 -1.314e-03 -1.123e-03
## beta2  1.142e-02  1.295e-02  1.431e-02  1.556e-02  1.629e-02
## beta3 -1.100e-03 -1.043e-03 -9.178e-04 -8.010e-04 -6.605e-04
## beta4  1.997e-05  2.405e-05  2.797e-05  3.198e-05  3.371e-05
## beta5 -3.706e-07 -3.500e-07 -3.111e-07 -2.692e-07 -2.292e-07
## beta6 -2.445e-04  1.123e-03  1.818e-03  2.521e-03  3.855e-03
## beta7 -3.163e-02 -2.189e-02 -1.618e-02 -1.080e-02  2.389e-03
## beta8 -8.047e-02 -6.383e-02 -5.360e-02 -4.395e-02 -1.452e-02
## tau    3.339e+02  3.397e+02  3.428e+02  3.458e+02  3.516e+02
```

Next, we take the posterior means of the regression parameters as estimated by the model and calculate $\hat{\mu}_t$, which is the mean of the distribution of y_t at time t, as estimated by our model. We can then plot the means against the true values of y_t to see how well the model fits the data.

```
stat <- as.vector(unlist(summary(posterior_sample4)[1])[1:11])
mu <- vector(length = length(y))
mu[1] <- y[1]

# Compare fitted mu with true values of y_t
for (t in 2:length(y)) {
  mu[t] <- stat[2] + stat[1] * y[t-1] +
             stat[3] * x1[t] +
             stat[4] * x2[t] + stat[5] * x2[t]^2 + stat[6] * x2[t]^3 + stat[7] * x2[t]^4 +
             stat[8] * x3[t] +
             stat[9] * sin(x4[t]) + stat[10] * cos(x4[t])
}

# Create a time variable
t <- 1:length(y)

# Create a data frame
data <- data.frame(
  t = t,
  y = y,
  mu = mu
)

# Create the scatter plot
ggplot(data, aes(x = mu, y = y)) +
  geom_point(alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  labs(title = "Actual vs Fitted Values",
       x = "Fitted Values (mu)",
       y = "Actual Values (y)") +
  theme_minimal()
```



We can see that the model fits the data well, as the means agree with the true values.

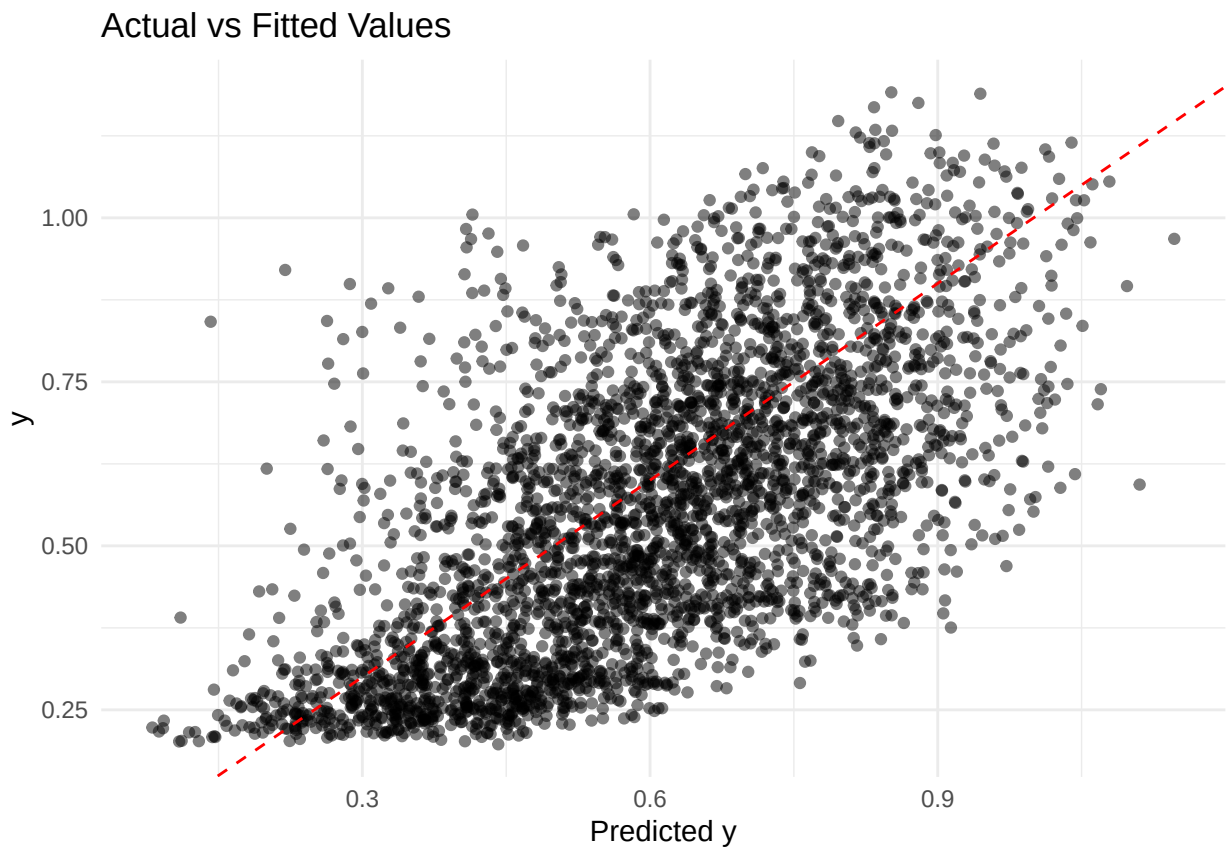
Now we want to find the predicted values of y for the test set. This allows us to assess the model performance using performance metrics like MSE.

```
set.seed(12)
# Extract the test data vectors
x1_test <- cluster4_data_test$temp
x2_test <- cluster4_data_test$tod
x3_test <- as.integer(cluster4_data_test$weekend)
x4_test <- cluster4_data_test$toy
y_test <- cluster4_data_test$Cluster.4
N_test <- length(y_test)

# Initialize a vector to store the predicted values
predicted_y <- numeric(N_test)

# Sigma is 1/sqrt(tau), where tau is the precision
sigma <- 1 / sqrt(stat[11])

# Make predictions using the posterior samples
mu_test <- numeric(N_test)
mu_test[1] <- y_test[1]
predicted_y[1] <- y_test[1]  # Initialize with the first observed value

for (t in 2:N_test) {
```

```
    mu_test[t] <- stat[2] + stat[1] * predicted_y[t-1] +
              stat[3] * x1_test[t] +
              stat[4] * x2_test[t] + stat[5] * x2_test[t]^2 + stat[6] * x2_test[t]^3 + stat[7]*x2_test[t]
              stat[8] * x3_test[t] +
              stat[9] * sin(x4_test[t]) + stat[10] * cos(x4_test[t])

    predicted_y[t] <- rnorm(1, mean = mu_test[t], sd = sigma)
}
```

We plot the predicted vs. fitted values:

```
# Create a data frame
t <- 1:length(y_test)

data_test <- data.frame(
  t = t,
  y = y_test,
  predicted_val = predicted_y
)

# Create the scatter plot
ggplot(data_test, aes(x = predicted_val, y = y)) +
  geom_point(alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  labs(title = "Actual vs Fitted Values",
       x = "Predicted y",
       y = "y") +
  theme_minimal()
```



Calculate MSE:

```
# Calculate Mean Squared Error (MSE)
mse <- mean((predicted_y - y_test)^2)

# Print the results
cat("MSE:", mse, "\n")
```

```
## MSE: 0.033491
```

Repeat for cluster 5:

```
# use cluster 5 from dataset
cluster5_data <- aggregated_data[, -c(1,2,3,4,6)]

# take first 80% of the data
num_rows <- nrow(cluster5_data)
num_rows_80_percent <- floor(0.8 * num_rows)
cluster5_data_train <- cluster5_data[1:num_rows_80_percent, ]
cluster5_data_test <- cluster5_data[(num_rows_80_percent + 1):num_rows, ]

# define the dependent variable and the explanatory variables
y <- cluster5_data_train$Cluster.5
x4 <- cluster5_data_train$toy        # time of year
x3 <- cluster5_data_train$weekend    # weekend
```

```
x1 <- cluster5_data_train$temp       # temperature
x2 <- cluster5_data_train$tod        # time of day
```

Fit model with burn in 12000 and posterior sample size of 16000.

```
set.seed(12)

# Data for the model
datalist <- list(N = length(y), x1 = x1, x2 = x2, x3 = x3, x4 = x4, y = y)

# Compile and run with 3 chains, burn-in of 9000 and total sample size of 13000
model <- jags.model(
  file = textConnection(model_string2),
  data = datalist,
  inits = list(inits(4), inits(5), inits(3)), # Seed for initial values for each chain
  n.chains = 3
)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 13439
##    Unobserved stochastic nodes: 11
##    Total graph size: 94829
##
## Initializing model
```

```
update(model, n.iter = 12000)

Nrep = 16000

posterior_sample5 <- coda.samples(
  model,
  variable.names = c("tau", "beta0", "beta1", "beta2", "beta3", "beta4",
                     "beta5", "beta6", "beta7", "beta8", "alpha"),
  n.iter = Nrep
)
```

Check convergence:

```
gelman.diag(posterior_sample5)
```

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## alpha       1.00       1.01
## beta0       1.04       1.09
## beta1       1.01       1.05
## beta2       1.44       2.19
## beta3       2.04       3.59
## beta4       2.19       3.80
```

```
## beta5       2.01       3.38
## beta6       1.00       1.00
## beta7       1.05       1.12
## beta8       1.05       1.14
## tau         1.00       1.00
##
## Multivariate psrf
##
## 1.74
```

In this case, the distributions of the variables $\beta_2, \beta_3, \beta_4$ and β_5 failed to converge. We tried with different initial values and different burn-in/posterior sample sizes, but this remained the case for all tried values, so we just conclude that for cluster 5, the model failed to converge. However, we can still try and use the posterior means in our model to predict for the test set.

We print the summary of the posterior samples below.

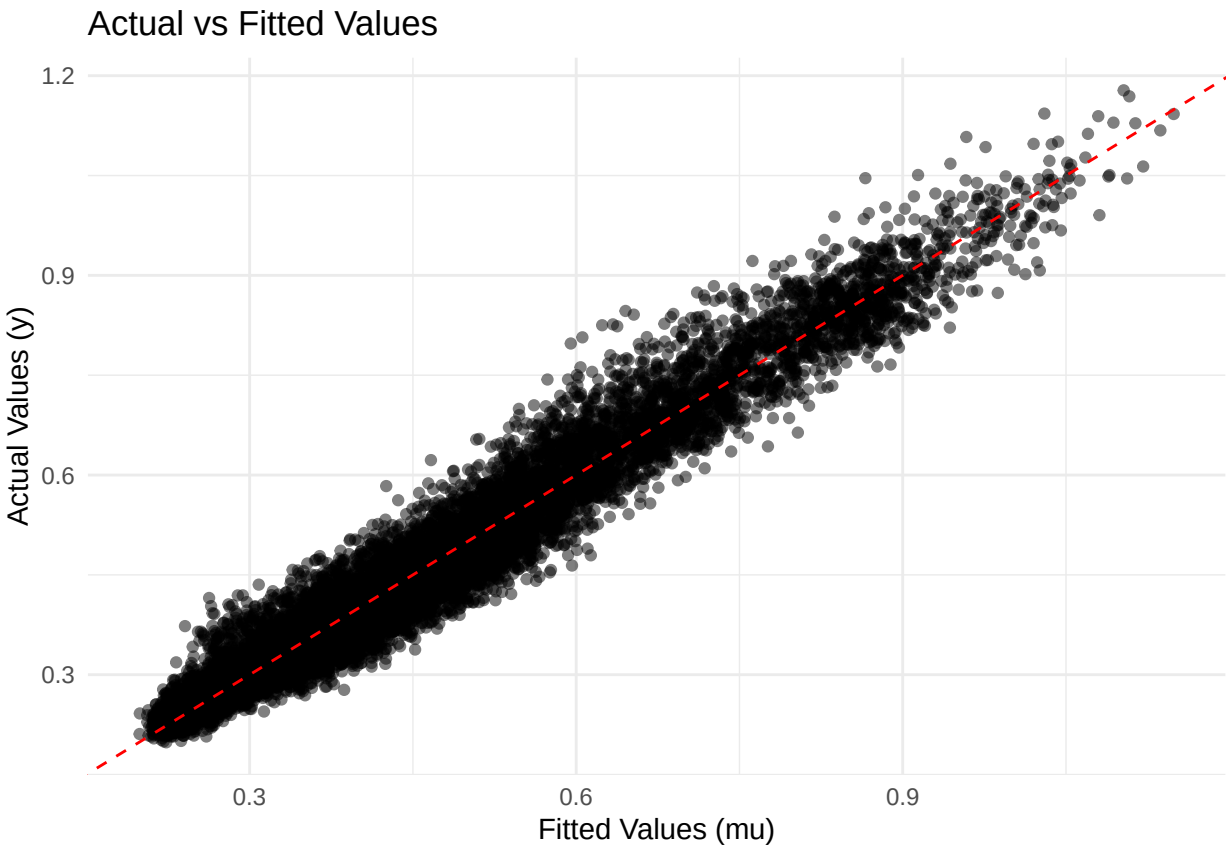
```
summary(posterior_sample5)
```

```
##
## Iterations = 12001:28000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 16000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean        SD  Naive SE Time-series SE
## alpha  8.835e-01 3.355e-03 1.531e-05      1.194e-04
## beta0  9.448e-02 1.237e-02 5.647e-05      2.861e-03
## beta1 -1.174e-03 9.812e-05 4.478e-07      3.142e-06
## beta2  3.363e-03 2.791e-04 1.274e-06      2.856e-05
## beta3 -3.855e-05 1.995e-05 9.108e-08      3.299e-06
## beta4  1.371e-07 6.248e-07 2.852e-09      1.268e-07
## beta5  4.904e-09 6.906e-09 3.152e-11      0.000e+00
## beta6  3.993e-03 7.531e-04 3.438e-06      5.172e-06
## beta7 -3.936e-02 5.993e-03 2.735e-05      8.198e-04
## beta8 -8.386e-02 1.129e-02 5.153e-05      2.462e-03
## tau    6.670e+02 8.135e+00 3.713e-02      3.714e-02
##
## 2. Quantiles for each variable:
##
##             2.5%        25%        50%        75%      97.5%
## alpha  8.768e-01  8.813e-01  8.835e-01  8.858e-01  8.899e-01
## beta0  6.254e-02  8.758e-02  9.562e-02  1.027e-01  1.156e-01
## beta1 -1.369e-03 -1.240e-03 -1.173e-03 -1.108e-03 -9.825e-04
## beta2  2.755e-03  3.212e-03  3.386e-03  3.539e-03  3.870e-03
## beta3 -7.789e-05 -5.239e-05 -4.042e-05 -2.476e-05 -9.405e-08
## beta4 -9.473e-07 -3.871e-07  1.863e-07  5.812e-07  1.345e-06
## beta5 -8.680e-09  3.802e-10  4.317e-09  1.023e-08  1.789e-08
## beta6  2.507e-03  3.488e-03  3.994e-03  4.501e-03  5.464e-03
## beta7 -4.992e-02 -4.354e-02 -3.964e-02 -3.586e-02 -2.505e-02
## beta8 -1.030e-01 -9.200e-02 -8.444e-02 -7.751e-02 -5.511e-02
```

```
## tau    6.511e+02  6.615e+02  6.670e+02  6.725e+02  6.831e+02
```

Next, we take the posterior means of the regression parameters as estimated by the model and calculate $\hat{\mu}_t$, which is the mean of the distribution of y_t at time t, as estimated by our model. We can then plot the means against the true values of y_t to see how well the model fits the data.

```
stat <- as.vector(unlist(summary(posterior_sample5)[1])[1:11])
mu <- vector(length = length(y))
mu[1] <- y[1]

# Compare fitted mu with true values of y_t
for (t in 2:length(y)) {
  mu[t] <- stat[2] + stat[1] * y[t-1] +
             stat[3] * x1[t] +
             stat[4] * x2[t] + stat[5] * x2[t]^2 + stat[6] * x2[t]^3 + stat[7] * x2[t]^4 +
             stat[8] * x3[t] +
             stat[9] * sin(x4[t]) + stat[10] * cos(x4[t])
}

# Create a time variable
t <- 1:length(y)

# Create a data frame
data <- data.frame(
  t = t,
  y = y,
  mu = mu
)

# Create the scatter plot
ggplot(data, aes(x = mu, y = y)) +
  geom_point(alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  labs(title = "Actual vs Fitted Values",
       x = "Fitted Values (mu)",
       y = "Actual Values (y)") +
  theme_minimal()
```



We can see that the model fits the data well, as the means agree with the true values.

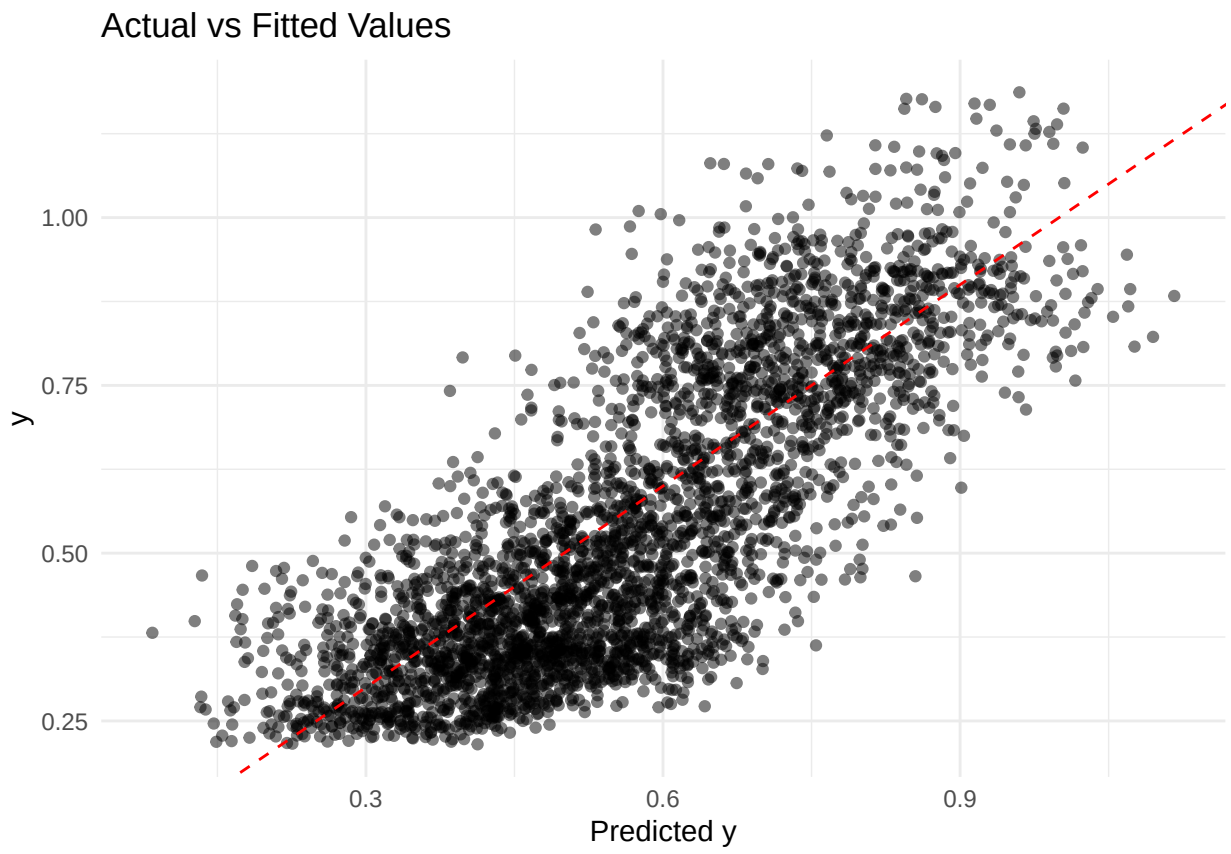
Now we want to find the predicted values of y for the test set. This allows us to assess the model performance using performance metrics like MSE.

```
set.seed(12)
# Extract the test data vectors
x1_test <- cluster5_data_test$temp
x2_test <- cluster5_data_test$tod
x3_test <- as.integer(cluster5_data_test$weekend)
x4_test <- cluster5_data_test$toy
y_test <- cluster5_data_test$Cluster.5
N_test <- length(y_test)

# Initialize a vector to store the predicted values
predicted_y <- numeric(N_test)

# Sigma is 1/sqrt(tau), where tau is the precision
sigma <- 1 / sqrt(stat[11])

# Make predictions using the posterior samples
mu_test <- numeric(N_test)
mu_test[1] <- y_test[1]
predicted_y[1] <- y_test[1]  # Initialize with the first observed value

for (t in 2:N_test) {
```

```
    mu_test[t] <- stat[2] + stat[1] * predicted_y[t-1] +
              stat[3] * x1_test[t] +
              stat[4] * x2_test[t] + stat[5] * x2_test[t]^2 + stat[6] * x2_test[t]^3 + stat[7]*x2_test[t]
              stat[8] * x3_test[t] +
              stat[9] * sin(x4_test[t]) + stat[10] * cos(x4_test[t])

    predicted_y[t] <- rnorm(1, mean = mu_test[t], sd = sigma)
}
```

We plot the predicted vs. fitted values:

```
# Create a data frame
t <- 1:length(y_test)

data_test <- data.frame(
  t = t,
  y = y_test,
  predicted_val = predicted_y
)

# Create the scatter plot
ggplot(data_test, aes(x = predicted_val, y = y)) +
  geom_point(alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  labs(title = "Actual vs Fitted Values",
       x = "Predicted y",
       y = "y") +
  theme_minimal()
```



Calculate MSE:

```
# Calculate Mean Squared Error (MSE)
mse <- mean((predicted_y - y_test)^2)

# Print the results
cat("MSE:", mse, "\n")
```

```
## MSE: 0.01898514
```

So even though convergence failed, the model predicts the test data well.

Repeat for cluster 6:

```
# use cluster 6 from dataset
cluster6_data <- aggregated_data[, -c(1,2,3,4,5)]

# take first 80% of the data
num_rows <- nrow(cluster6_data)
num_rows_80_percent <- floor(0.8 * num_rows)
cluster6_data_train <- cluster6_data[1:num_rows_80_percent, ]
cluster6_data_test <- cluster6_data[(num_rows_80_percent + 1):num_rows, ]

# define the dependent variable and the explanatory variables
y <- cluster6_data_train$Cluster.6
x4 <- cluster6_data_train$toy        # time of year
```

```
x3 <- cluster6_data_train$weekend    # weekend
x1 <- cluster6_data_train$temp       # temperature
x2 <- cluster6_data_train$tod        # time of day
```

Fit model with burn in 12000 and posterior sample size of 16000.

```
set.seed(12)

# Data for the model
datalist <- list(N = length(y), x1 = x1, x2 = x2, x3 = x3, x4 = x4, y = y)

# Compile and run with 3 chains, burn-in of 11000 and total sample size of 15000
model <- jags.model(
  file = textConnection(model_string2),
  data = datalist,
  inits = list(inits(4), inits(5), inits(3)), # Seed for initial values for each chain
  n.chains = 3
)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 13439
##    Unobserved stochastic nodes: 11
##    Total graph size: 95093
##
## Initializing model
```

```
update(model, n.iter = 12000)

Nrep = 16000

posterior_sample6 <- coda.samples(
  model,
  variable.names = c("tau", "beta0", "beta1", "beta2", "beta3", "beta4",
                     "beta5", "beta6", "beta7", "beta8", "alpha"),
  n.iter = Nrep
)
```

Check convergence:

```
gelman.diag(posterior_sample6)
```

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## alpha       1.06       1.19
## beta0       1.04       1.08
## beta1       1.01       1.05
## beta2       1.80       2.98
## beta3       2.56       4.71
```

```
## beta4       2.70       4.91
## beta5       2.40       4.21
## beta6       1.00       1.00
## beta7       1.03       1.09
## beta8       1.04       1.10
## tau         1.00       1.00
##
## Multivariate psrf
##
## 2.08
```

Again, we see that the distributions of the variables $\beta_2, \beta_3, \beta_4$ and β_5 failed to converge. We tried with different initial values and different burn-in/posterior sample sizes, but this remained the case for all tested values, so we conclude that the model also failed to converge for cluster 6. We can still try and use the posterior means in our model to predict for the test set.

We print the summary of the posterior samples below.

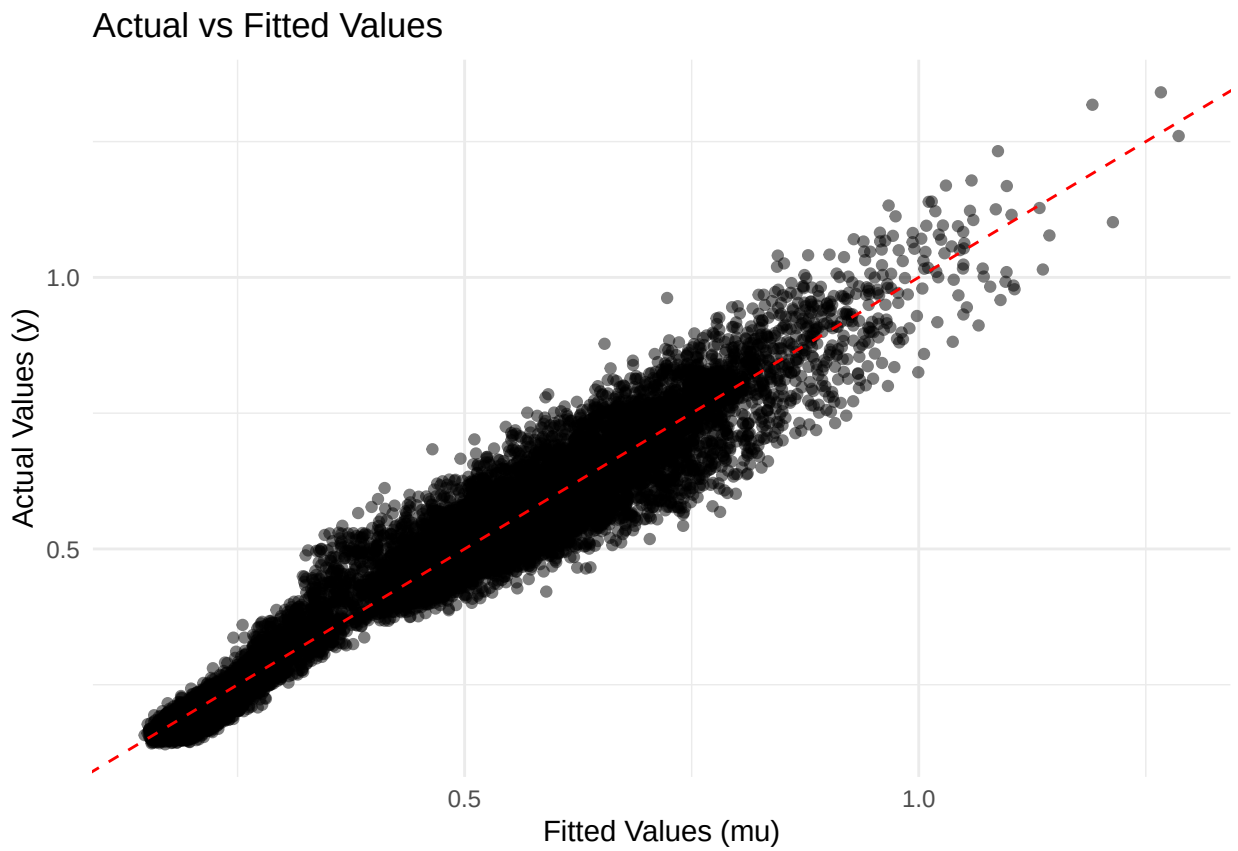
```
summary(posterior_sample6)
```

```
##
## Iterations = 12001:28000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 16000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean        SD  Naive SE Time-series SE
## alpha  8.878e-01 3.322e-03 1.516e-05      1.042e-04
## beta0  4.488e-02 1.504e-02 6.864e-05      3.584e-03
## beta1 -1.176e-03 1.228e-04 5.603e-07      4.057e-06
## beta2  1.149e-02 3.280e-04 1.497e-06      2.542e-05
## beta3 -2.724e-04 2.697e-05 1.231e-07      3.776e-06
## beta4  1.229e-07 8.517e-07 3.887e-09      1.499e-07
## beta5  2.607e-08 9.197e-09 4.198e-11      0.000e+00
## beta6  1.543e-03 9.467e-04 4.321e-06      5.930e-06
## beta7 -3.121e-02 7.496e-03 3.422e-05      1.050e-03
## beta8 -7.430e-02 1.415e-02 6.460e-05      3.350e-03
## tau    4.146e+02 5.057e+00 2.308e-02      2.309e-02
##
## 2. Quantiles for each variable:
##
##             2.5%        25%        50%        75%      97.5%
## alpha  8.813e-01  8.855e-01  8.878e-01  8.900e-01  8.942e-01
## beta0  5.622e-03  3.661e-02  4.628e-02  5.524e-02  6.959e-02
## beta1 -1.420e-03 -1.258e-03 -1.174e-03 -1.094e-03 -9.369e-04
## beta2  1.083e-02  1.127e-02  1.150e-02  1.171e-02  1.213e-02
## beta3 -3.309e-04 -2.927e-04 -2.717e-04 -2.523e-04 -2.227e-04
## beta4 -1.349e-06 -5.595e-07  7.549e-08  7.660e-07  1.925e-06
## beta5  6.859e-09  1.965e-08  2.627e-08  3.302e-08  4.321e-08
## beta6 -3.264e-04  9.105e-04  1.542e-03  2.181e-03  3.396e-03
## beta7 -4.404e-02 -3.651e-02 -3.164e-02 -2.694e-02 -1.296e-02
```

```
## beta8 -9.705e-02 -8.456e-02 -7.521e-02 -6.649e-02 -3.765e-02
## tau    4.047e+02  4.112e+02  4.146e+02  4.180e+02  4.246e+02
```

Next, we take the posterior means of the regression parameters as estimated by the model and calculate $\hat{\mu}_t$, which is the mean of the distribution of y_t at time t, as estimated by our model. We can then plot the means against the true values of y_t to see how well the model fits the data.

```
stat <- as.vector(unlist(summary(posterior_sample6)[1])[1:11])
mu <- vector(length = length(y))
mu[1] <- y[1]

# Compare fitted mu with true values of y_t
for (t in 2:length(y)) {
  mu[t] <- stat[2] + stat[1] * y[t-1] +
             stat[3] * x1[t] +
             stat[4] * x2[t] + stat[5] * x2[t]^2 + stat[6] * x2[t]^3 + stat[7] * x2[t]^4 +
             stat[8] * x3[t] +
             stat[9] * sin(x4[t]) + stat[10] * cos(x4[t])
}

# Create a time variable
t <- 1:length(y)

# Create a data frame
data <- data.frame(
  t = t,
  y = y,
  mu = mu
)

# Create the scatter plot
ggplot(data, aes(x = mu, y = y)) +
  geom_point(alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  labs(title = "Actual vs Fitted Values",
       x = "Fitted Values (mu)",
       y = "Actual Values (y)") +
  theme_minimal()
```



We can see that the model fits the data well, as the means agree with the true values.

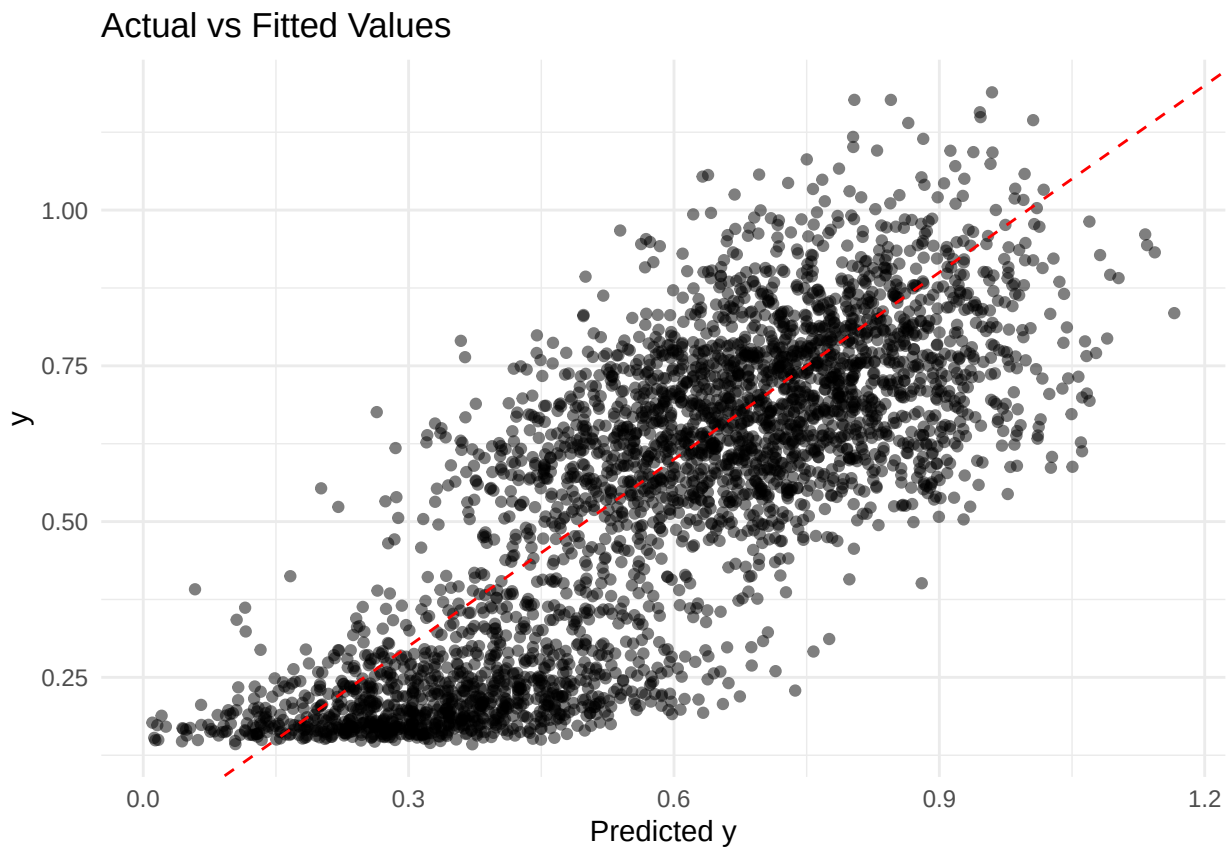
Now we want to find the predicted values of y for the test set. This allows us to assess the model performance using performance metrics like MSE.

```
set.seed(12)
# Extract the test data vectors
x1_test <- cluster6_data_test$temp
x2_test <- cluster6_data_test$tod
x3_test <- as.integer(cluster6_data_test$weekend)
x4_test <- cluster6_data_test$toy
y_test <- cluster6_data_test$Cluster.6
N_test <- length(y_test)

# Initialize a vector to store the predicted values
predicted_y <- numeric(N_test)

# Sigma is 1/sqrt(tau), where tau is the precision
sigma <- 1 / sqrt(stat[11])

# Make predictions using the posterior samples
mu_test <- numeric(N_test)
mu_test[1] <- y_test[1]
predicted_y[1] <- y_test[1]  # Initialize with the first observed value

for (t in 2:N_test) {
```

```
    mu_test[t] <- stat[2] + stat[1] * predicted_y[t-1] +
              stat[3] * x1_test[t] +
              stat[4] * x2_test[t] + stat[5] * x2_test[t]^2 + stat[6] * x2_test[t]^3 + stat[7]*x2_test[t]
              stat[8] * x3_test[t] +
              stat[9] * sin(x4_test[t]) + stat[10] * cos(x4_test[t])

    predicted_y[t] <- rnorm(1, mean = mu_test[t], sd = sigma)
}
```

We plot the predicted vs. fitted values:

```
# Create a data frame
t <- 1:length(y_test)

data_test <- data.frame(
  t = t,
  y = y_test,
  predicted_val = predicted_y
)

# Create the scatter plot
ggplot(data_test, aes(x = predicted_val, y = y)) +
  geom_point(alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  labs(title = "Actual vs Fitted Values",
       x = "Predicted y",
       y = "y") +
  theme_minimal()
```



Calculate MSE:

```
# Calculate Mean Squared Error (MSE)
mse <- mean((predicted_y - y_test)^2)

# Print the results
cat("MSE:", mse, "\n")
```

```
## MSE: 0.02542067
```

So even though convergence failed, the model predicts the test data well.

Conclusion

In this study, we successfully developed and tested a Bayesian time series regression model to forecast household electricity demand. Our exploration began with a detailed exploratory data analysis (EDA), followed by effective data aggregation strategies, and culminated in the adoption of Bayesian methods for prediction, which were rigorously evaluated against standard performance metrics. We summarise our work as follows.

1. **Exploratory Data Analysis (EDA):** We commenced with a comprehensive analysis to understand the distribution and variability of electricity demand. This phase involved identifying key patterns, trends, and outliers, which facilitated the selection of relevant features for further modelling.

2. **Data Aggregation:** By clustering households based on their daily demand profiles, we reduced the dataset's dimensionality. This enabled more focused and efficient modelling, making the process manageable despite the dataset's large size.

3. **Model Development:** We implemented a Bayesian time series regression model that integrates non-linear transformations and accounts for temporal dependencies. This model structure was chosen based on its suitability for the dataset characteristics highlighted in the EDA phase.

4. **Model Evaluation:** The predictive accuracy of the model was assessed using metrics such as Mean Squared Error (MSE). Comparisons were made between our Bayesian approach and traditional linear regression models to underline the strengths of our methodology.

Our findings reveal that Bayesian models, equipped with non-linear transformations and temporal dynamics, provide a robust framework for capturing complex patterns in electricity demand. The performance of the model, highlighted by its ability to accommodate seasonal variations and daily fluctuations, demonstrates its potential utility in practical energy management scenarios. Notably, the comparison of Bayesian approaches with traditional linear regression models showcased the former's superior capability in handling large, noisy datasets typical in smart metering environments.

Looking forward, this research opens several avenues for further investigation. Future studies could explore the integration of additional predictive factors such as demographic and economic indicators to enhance the model's accuracy. Additionally, the application of more sophisticated statistical techniques could be considered to improve the adaptability and predictive performance of the models under varying conditions.

The implications of this research are significant, offering a promising direction for energy policymakers and utility companies to enhance their demand forecasting capabilities, ultimately leading to more efficient and sustainable energy management practices.