Energy Demand and the Impact of Covid-19

2023-04-30

Data Origins

The raw data for this project was downloaded from GridWatch on 2023-04-29 in one-year chunks from 2012 to 2022. The data from GridWatch is downloaded every five minutes from the BM Reports site provided by Elexon and the University of Sheffield. The data includes id, timestamp and demand. More information about the variables can be found in the codebook.txt file in the notes folder. Still, demand is the near-to-real-time electricity demand on the United Kingdoms electricity grid recorded in megawatts (MW) and timestamp is when the data was scrapped from the BM reports site by GridWatch.

Research Question

How did Covid-19 affect electrical demand in the United Kingdom?

The visualisation will attempt to show the impact of Covid-19 on electricity demand in the UK. It is hypothesised that there will be a significant effect that is easy to interpret on a graph because many events during Covid could have impacted energy consumption, such as national lockdowns, meaning many businesses had to close, and people were ordered to work from home. The extensive data set includes over 1.1 million entries, meaning visualising the data is very valuable.

Data Preparation

The code for this project was separated into five script files, including utils, config, data-management, visualisation and main. The visualisation can be generated and saved by running main by itself. This format was inspired by kmishra9 and was done for better organisation and easier debugging.

Step 1: Utility Function

Here I created a reusable function to help convert date time to a date format

```
#' Function to convert date to date format
#'
#' @param value: date to be converted
#' @param new_format: format of date to be converted to
#' @return: formatted date
#' @export
#'
#' @examples:
#' convert_date("2021-04-23 00:00:00", new_format= "%Y-%m-%d")
convert_date = function(value,
                        new_format = "%Y-%m-%d %H:%M:%S") {
  # Format date
  new_value = as.Date(strftime(value, new_format))

  # Return formatted date
```

```
  return(new_value)
}
```

Step 2: Configuration

The code below is the setup for loading the required libraries and specifying the file paths.

```
#################################
# Load libraries
#################################
library(here)
library(tidyverse)
library(ggplot2)
library(scales)
library(plotly)
library(tidyr)
library(purrr)
library(stringr)
library(rmarkdown) #this is specific to rmarkdown
library(webshot) #this is specfic to markdown
install_phantomjs() #this is specific to rmarkdown to enable ggplotly to render in pdf
#################################
# Set file directories
#################################
raw_dir = here("raw/")
figs_dir = here("figs/")
processed_dir = here("processed/")

#################################
# Set file paths
#################################
processed_merge_data_path = paste0(processed_dir, "gridwatch_merge.csv")
processed_data_path = paste0(processed_dir, "gridwatch_processed.csv")
processed_unique_years_path = paste0(processed_dir, "gridwatch_unique_years.csv")
```

Step 3: Data Management

The code below is reading in multiple csv files of raw data and combining them into one data frame. Initially, I downloaded all the years' data in one go. However, due to the size, it was difficult to open the file; Excel and numbers on Mac both gave warning messages. I think doing it in steps and combining them is more easily maintainable.

```
# Merge all csv files into one csv file
merged_data <- raw_dir %>%
  #list all csv files within raw directory
  list.files(pattern = '(?i)\\.csv$', full.names = TRUE) %>%
  #set file names ignoring the file extension
  set_names(tools::file_path_sans_ext(basename(.))) %>%
  #read file one by one into dataframe and create a new column called file name
  map_dfr(read.csv,
          col.names = c("id", "timestamp", "demand"),
          .id = "filename")
```

A sample of raw data from the 2012 file can be seen below

```
##      id            timestamp demand
## 1 62694  2012-01-01 00:00:01  30590
## 2 62695  2012-01-01 00:05:06  30490
## 3 62696  2012-01-01 00:10:01  30802
## 4 62697  2012-01-01 00:15:01  31180
## 5 62698  2012-01-01 00:20:01  31241
## 6 62699  2012-01-01 00:25:10  31340
```

During the merging processed I added a new column called filename which showed which csv each row belongs to. The table below shows a random sample of the merged data.

filename	id	timestamp	demand
gridwatch_2014	324529	2014-06-30 08:10:02	35956
gridwatch_2012	108991	2012-06-09 23:47:43	27403
gridwatch_2020	969989	2020-08-23 09:30:35	23735
gridwatch_2016	522675	2016-05-20 14:05:04	28978
gridwatch_2022	1207982	2022-11-28 00:20:40	29752

To ensure there were no duplicate values after combining data sets, I removed duplicates from the merged data.

```
# Remove duplicate by id
formatted_data <- distinct(merged_data, id, .keep_all = TRUE)
```

The code below shows my steps for data wrangling and exporting the processed data.

```
########################################
# Data wrangling, cleaning and then export processed data
########################################

# Convert date column to date format
formatted_data = formatted_data %>%
  mutate(date = convert_date(timestamp, new_format = "%Y-%m-%d")) %>%
  mutate(year = strftime(timestamp, format = "%Y"))

# Select subset of columns
formatted_data = formatted_data %>%
  select(date, demand, year)

#creating a new column called group month
formatted_data <-
  mutate(formatted_data, group_month = convert_date(date, "%2023-%m-01"))
#grouping by group month and year and getting the mean average per month
formatted_data <-
  formatted_data %>% group_by(year, group_month) %>%
  summarise(average_per_month = mean(demand /1000), .groups = "drop") #.group is to override a summaris

#creating a unique list of years and counting the number of months within the years
unique_years_list <-
  formatted_data %>% group_by(year) %>% summarise(month_count = n_distinct(group_month)) %>%
  arrange(desc("year"))
```

```
# Export processed data
write_csv(merged_data, processed_merge_data_path)
write_csv(formatted_data, processed_data_path)
write_csv(unique_years_list, processed_unique_years_path)
```

A sample of processed data can be seen below

year	group_month	average_per_month
2021	2023-04-01	28.90974
2018	2023-05-01	27.56869
2021	2023-01-01	34.84935
2013	2023-05-01	33.35355
2012	2023-04-01	36.10796

Visualisation

I tried plotting many different graphs before reaching my final visualisation; they can be found in the attic folder with accompanying test plots r.script. Ultimately, they led me to decide an average of demand per month instead of demand per day was better for visualisation as fewer points made the graph more easily interpretable.

I started by reading the processed csvs into a data frame

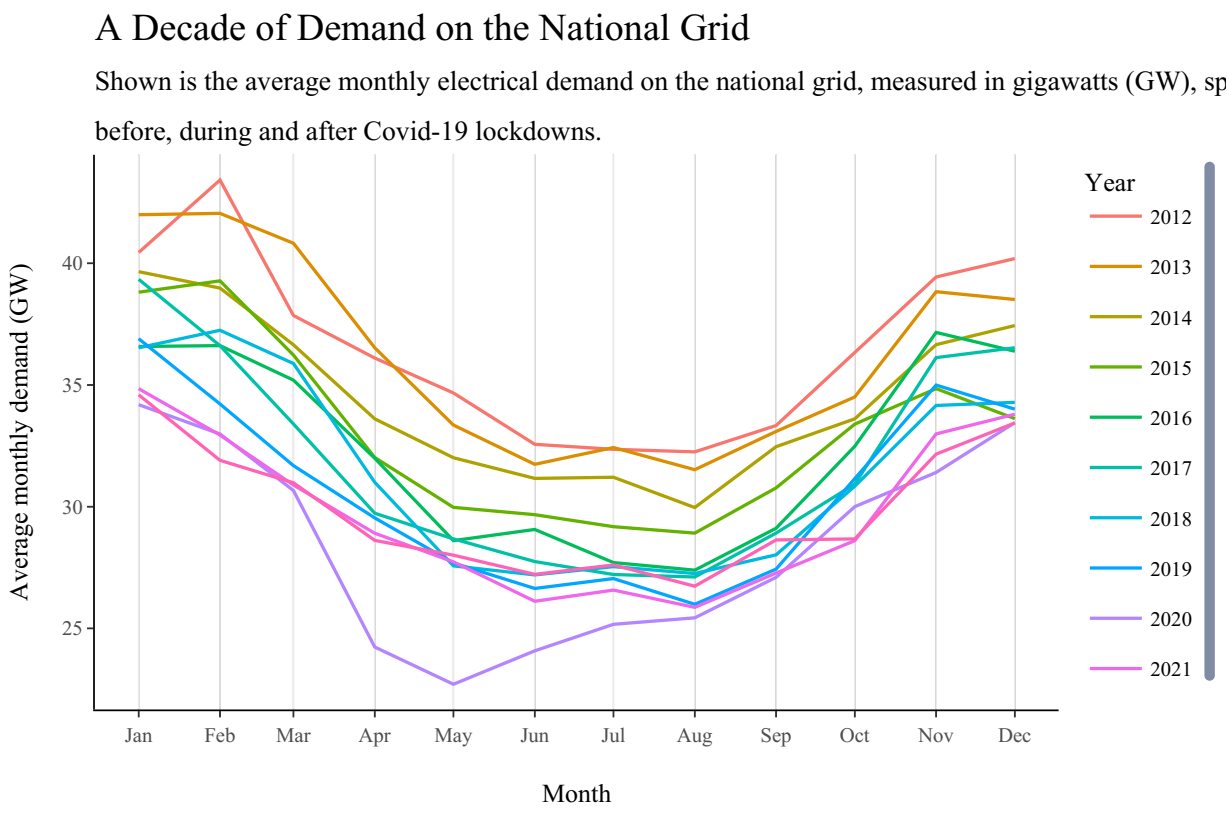
```
# Reading processed data into a dataframe and specifying the column types
# Processed data is average demand of each month in each year
processed_data = read_csv(
  processed_data_path,
  col_types = cols(
    year = col_double(),
    group_month = col_date(),
    average_per_month = col_double()
  )
)

# Reading unique years data into dataframe and specifying column types
processed_unique_years = read_csv(
  processed_unique_years_path,
  col_types = cols(year = col_double(),
                   month_count = col_double())
)
```

And then created my final visualisation using ggplot and plotly.

```
visualisation <- ggplot()

visualisation_title <- "A Decade of Demand on the National Grid"

visualisation_subtitle <- "Shown is the average monthly electrical demand on the national grid, measure

# going through each year by getting the number of rows within processed unique years (puy) data
for (i in 1:nrow(processed_unique_years)) {
```

```
  row <-
    processed_unique_years[i, ] #retrieve a row from puy for each iteration
  row_data <-
    filter(processed_data, year == row$year) #filter by the year and return a subset of processed data

  #drawing a plot for each year
  visualisation <-
    visualisation + geom_line(
      #line chart
      data = row_data,
      #using row data created above
      aes(
        x = group_month,
        y = average_per_month,
        group = year,
        color = as.factor(year),
        #as.factor so year is seen as categorical not continuous

        #setting text for the tooltip
        text = paste(
          "Date: ",
          format(group_month, "%b"),
          year,
          "<br>Demand: ",
          format(round(average_per_month, 2), nsmall = 2),
          "GW"
        )
      )
    )
}

visualisation <-
  visualisation + scale_x_date(date_labels = "%b", date_breaks = "1 month") +
  labs(
    x = "Month",
    y = "Average monthly demand (GW)", #labeling the x and y axis
    color = "Year", #adding the key heading
    caption = "Source: GridWatch", #adding a caption
    title = str_wrap(
      visualisation_title,
      width = rel(55) #making the width relative
    ),
    subtitle = str_wrap(
      visualisation_subtitle,
      width = rel(90)
    )
  ) +
  theme_bw() +
  theme(
    text = element_text(family = "Times New Roman"), #changing the font
    plot.title = element_text(
      hjust = 0,
      size = rel(1.5),
```

```
      margin = margin(0, 0, 5, 0)
    ), #adjusting placement and making the title bigger
    plot.subtitle = element_text(
      size = rel(0.9),
      margin = margin(0, 0, 20, 0) #adjusting placement of subtitle
    ),
    plot.caption = element_text(hjust = 0), #adjusting placement of caption
    panel.grid.major.y = element_blank(), #removing the horizontal background lines
    panel.grid.minor.y = element_blank(), #removing the lighter horizontal background lines
    panel.grid.minor.x = element_blank(), #removing the minor vertical background lines
    panel.border = element_blank(),#removing the black border
    plot.margin = margin(20, 10, 20, 10, "mm"), #creating more space around the graph
    axis.title.y = element_text(margin = margin(0, 10, 0, 0)),#made more space between the y axis text
    axis.title.x = element_text(margin = margin(10, 0, 0, 0)) #made more space between the x axis text
  ) +
  theme(axis.line = element_line(color = "black")) #adding black axis lines

ggplotly(
  visualisation,
  tooltip = c("text") #using tooltip
) %>%
  layout(title = list(
    text = paste0(
      visualisation_title,
      "<br>",
      "<span style='font-size:16px;'>",
      str_wrap(visualisation_subtitle, width = rel(120)),
      "</span>"
    )
  )) #layout needed because plotly is not including subtitle
```



The final output is saved in the figs folder

```
#saving the plot
ggsave(filename = here(figs_dir, "markdown_viz220225638.png")) #saving with different name
#to script because of issue with subtitle formatting after knitting
```

Interpretation

It is apparent that there is a significant dip in demand in 2020, the primary year affected by the pandemic. The first lockdown and working from home came into force in March and it is between March and April that demand decreases significantly. However, in pre-pandemic years energy demand also decreased around that time due to the weather changes. The most significant difference in 2020 compared to other years is that the dip in demand is very steep, reaching lower than in any other year. Demand also starts to increase from May to July when, in other years, it would still be fairly consistent across the summer months. This coincides with the ease in restrictions from the first lockdown. I think the tooltip helps with interpretation because where the lines overlap more, you can easily pinpoint the date and average demand for that month. It is also really useful to isolate and view specific years to compare them. Having a period of twelve months on the x-axis and viewing each year on the same graph makes yearly energy demand patterns easier to see and compare how these change across the years.

Summary and possible extensions

Overall, I am pleased with my graph. I took inspiration from charts on Our World In Data for the layout, titles, subtitles and caption at the bottom, and I think my final visualisation would fit in fairly well on that website. However, I think on reflection, it would have been good to add some covid events and dates to the tooltip to tie my graph and my research question together more; this would be the first improvement I would make if I had more time.

I think the other limitation is that although my graph highlights a significant change in electrical demand in 2020, which coincides with the pandemic, more insights could be gained into the effect of the pandemic. For example, visualising how demand changed for different industries such as commercial, transport and domestic.

Finally, another improvement I would like to make would be to get the code to run faster. I think the function to convert the dates is slowing everything down and could be optimised.

A key thing I have learnt from completing this project is it is important to map out what you want to achieve and think about the desired outcome at the start and that the actual mapping onto a graph is a lot easier when the data has been managed properly.