Group 22 Data Management Parts A1(DDL and Queries), A2, B, C, D

A1: DDL

```sql
-- DDL using SQLite 3 syntax based on https://www.sqlite.org/

CREATE TABLE Guest (
  Guest_id INTEGER PRIMARY KEY AUTOINCREMENT,
  Post_code TEXT NOT NULL,
  State TEXT NOT NULL,
  City TEXT NOT NULL,
  Country TEXT Not NULL,
  Street_name TEXT NOT NULL,
  Street_number TEXT NOT NULL,
  Phone_num_work TEXT,
  Phone_num_cell TEXT,
  Phone_num_home TEXT,
  Email_address TEXT,
  First_name TEXT NOT NULL,
  Middle_name TEXT,
  Last_name TEXT NOT NULL,
);

CREATE TABLE Reservation (
  Reservation_id INTEGER PRIMARY KEY AUTOINCREMENT,
  Guest_id INTEGER NOT NULL,
   -- Smoking_allowed could be boolean, but it would be converted to numeric in SQLite
  Smoking_preferred INTEGER,
  Nr_beds_preferred INTEGER,
  High_or_low_floor_preferred TEXT,
  -- SQLite does not have a Date data type,
  --it can be stored as either Text, Real or Integer
  -- In this case Integer is chosen for simplicity.
  --Date operations work on all of them."https://www.sqlite.org/datatype3.html"
  Arrival_date INTEGER NOT NULL,
  Departure_date INTEGER NOT NULL,
  Credit_card_num INTEGER NOT NULL,
  Credit_card_expiry_year INTEGER NOT NULL,
  credit_card_expiry_month INTEGER NOT NULL,
  Channel_id INTEGER,
  Channel_fee REAL,
  FOREIGN KEY (Guest_id) REFERENCES Guest(Guest_id),
  FOREIGN KEY (Channel_id) REFERENCES Booking_channel(Channel_id)
);

```

```
42
43  CREATE TABLE Additional_services (
44    Add_serv_id PRIMARY KEY AUTOINCREMENT,
45    Reservation_id INTEGER,
46    Service_name TEXT NOT NULL,
47    FOREIGN KEY (Reservation_id) REFERENCES Reservation(Reservation_id)
48  );
49
50  CREATE TABLE Stay (
51    Stay_id PRIMARY KEY AUTOINCREMENT,
52    Guest_id INTEGER NOT NULL,
53    -- SQLite does not have a Date data type,
54    --it can be stored as either Text, Real or Integer
55    -- In this case Integer is chosen for simplicity.
56    --Date operations work on all of them."https://www.sqlite.org/datatype3.html"
57    Arrival_date INTEGER NOT NULL,
58    Departure_date INTEGER NOT NULL,
59    Channel_id TEXT,
60    Channel_fee REAL,
61    -- Assuming official invoice number cannot contain
62    -- letters of the alphabet, only numbers.
63    Invoice_number INTEGER,
64    FOREIGN KEY (Guest_id) REFERENCES Guest(Guest_id),
65    FOREIGN KEY (Channel_id) REFERENCES Booking_channel(Channel_id)
66
67  );
68
69  CREATE TABLE Invoice_charges (
70    Charge_item_id INTEGER PRIMARY KEY AUTOINCREMENT,
71    Stay_id INTEGER NOT NULL,
72    Item_name TEXT,
73    Ex_tax_amount REAL,
74    Tax_amount REAL,
75    FOREIGN KEY (Stay_id) REFERENCES Stay(Stay_id)
76
77  );
78
79  CREATE TABLE Invoice_payments (
80    Payment_item_id INTEGER PRIMARY KEY AUTOINCREMENT,
81    Stay_id INTEGER NOT NULL,
82    Payment_type TEXT NOT NULL,
83    Amount REAL NOT NULL,
84    FOREIGN KEY (Stay_id) REFERENCES Stay(Stay_id)
85  );
86
87  CREATE TABLE Booking_channel (
88    Channel_id PRIMARY KEY AUTOINCREMENT,
89    Channel_name TEXT
90  );
91
92  CREATE TABLE Hotel (
93    Hotel_id INTEGER PRIMARY KEY AUTOINCREMENT,
94    Name TEXT NOT NULL,
95    Home_page TEXT NOT NULL,
```

```
  Post_code TEXT NOT NULL,
  State TEXT NOT NULL,
  City TEXT NOT NULL,
  Street_name TEXT NOT NULL,
  Street_number TEXT NOT NULL,
  Primary_phone_number TEXT NOT NULL
);

CREATE TABLE Additional_facilities (
  Add_facility_id INTEGER PRIMARY KEY AUTOINCREMENT,
  Add_facility_name TEXT NOT NULL,
  Add_facility_cost float NOT NULL,
  Hotel_id INTEGER,
  FOREIGN KEY (Hotel_id) REFERENCES Hotel(Hotel_id)
);

CREATE TABLE Room (
  Hotel_id INTEGER NOT NULL,
  Room_name_or_number TEXT NOT NULL,
  floor INTEGER NOT NULL,
  Nr_beds INTEGER NOT NULL,
  -- Smoking_allowed could be boolean, but it would be converted to numeric in SQLite
  Smoking_allowed INTEGER NOT NULL,
  PRIMARY KEY (Hotel_id, Room_name_or_number),
  FOREIGN KEY (Hotel_id) REFERENCES Hotel(Hotel_id)
);

CREATE TABLE Room_allocation (
  Hotel_id INTEGER NOT NULL,
  Room_name_or_number INTEGER NOT NULL,
    -- SQLite does not have a Date data type,
  --it can be stored as either Text, Real or Integer
  -- In this case Integer is chosen for simplicity.
  --Date operations work on all of them."https://www.sqlite.org/datatype3.html"
  Date INTEGER NOT NULL,
  Channel_id INTEGER DEFAULT NULL,
  Reservation_id INTEGER DEFAULT NULL,
  Stay_id INTEGER DEFAULT NULL,
  PRIMARY KEY (Hotel_id, Room_name_or_number, Date),
  FOREIGN KEY (Channel_id) REFERENCES Booking_channel(Channel_id),
  FOREIGN KEY (Reservation_id) REFERENCES Reservation(Reservation_id),
  FOREIGN KEY (Stay_id) REFERENCES Stay(Stay_id),
  FOREIGN KEY (Hotel_id, Room_name_or_number)
  REFERENCES Room(Hotel_id,Room_name_or_number)
  -- Ensure a room is only allocated to one purpose.
  CONSTRAINT Only_one_key CHECK
    ((Channel_id NOT NULL
    OR Reservation_id NOT NULL
    OR Stay_id NOT NULL)
    AND NOT (Channel_id NOT NULL AND Reservation_id NOT NULL)
    AND NOT (Reservation_id NOT NULL AND Stay_id NOT NULL)
    AND NOT (Channel_id NOT NULL AND Stay_id NOT NULL))
);
```

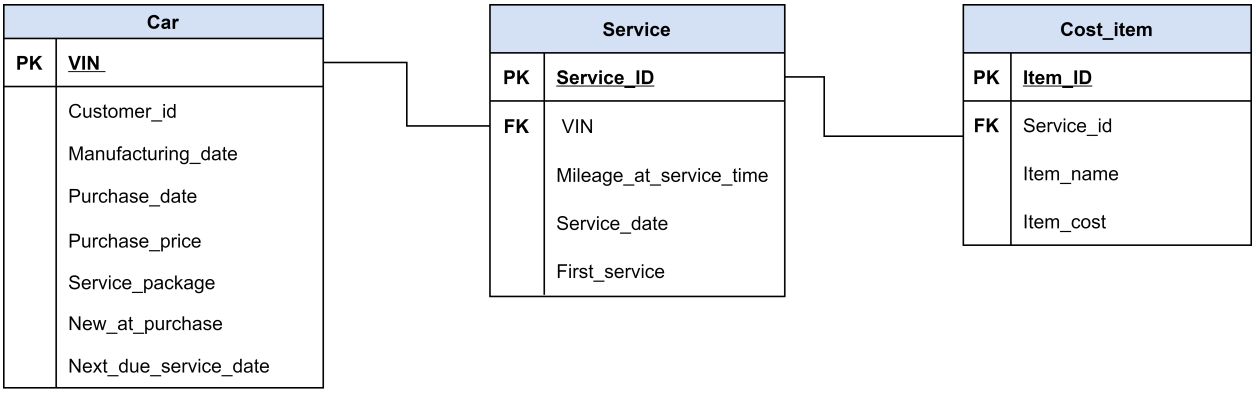
A1 Queries

```sql
-- 1. The total spent for the customer for a particular stay (checkout invoice).

-- Assuming total spent includes taxes paid.
SELECT S.Guest_id, S.Stay_id, SUM(IP.Amount) FROM Stay S
INNER JOIN Invoice_payments IP ON IP.Stay_id = S.Stay_id
-- To specify a particular stay:
WHERE S.Stay_id = 12345;


-- 2. The most valuable customers in (a) the last two months,
-- (b) past year and (c) from the beginning of the records.

-- a

-- Assuming last two months means to count all
-- stays that started in the last two months.
-- Assuming value means total spent including taxes paid
SELECT G.Guest_id, SUM(IP.Amount) AS Total_spent FROM Stay S
INNER JOIN Guest G ON G.Guest_id = S.Guest_id
INNER JOIN Invoice_payments IP ON IP.Stay_id = S.Stay_id
-- Based on https://www.sqlite.org/lang_datefunc.html
WHERE date(S.Arrival_date) >= date('now',"-2 months")
GROUP BY G.Guest_id
ORDER BY Total_spent DESC LIMIT 10;

-- b

SELECT G.Guest_id, SUM(IP.Amount) AS Total_spent FROM Stay S
INNER JOIN Guest G ON G.Guest_id = S.Guest_id
INNER JOIN Invoice_payments IP ON IP.Stay_id = S.Stay_id
-- Based on https://www.sqlite.org/lang_datefunc.html,
-- Assuming by past year, the last 365 days are meant.
-- If all stays since the start of the year are meant,
-- "-1 year" would be replaced by "start of year".
WHERE date(S.Arrival_date) >= date('now',"-1 year")
GROUP BY G.Guest_id
ORDER BY Total_spent DESC LIMIT 10;

-- c

SELECT G.Guest_id, SUM(IP.Amount) AS Total_spent FROM Stay S
INNER JOIN Guest G ON G.Guest_id = S.Guest_id
INNER JOIN Invoice_payments IP ON IP.Stay_id = S.Stay_id
GROUP BY G.Guest_id
ORDER BY Total_spent DESC LIMIT 10;


-- 3. Which are the top countries where our customers come from?

SELECT Country, COUNT(Guest_id) AS Frequency FROM Guest
GROUP BY Country
```

```sql
ORDER BY Frequency DESC LIMIT 10;


-- 4. How much did the hotel pay in referral fees for each
-- of the platforms that we have contracted with?

-- Assuming arrival date on the last day of the month means
-- channel fees are charged for that month still.
SELECT C.Channel_id, C.Channel_name, SUM(S.Channel_fee) AS Total_fees
FROM Booking_channel C
INNER JOIN Stay S on S.Channel_id=C.Channel_id
-- Only include stays that finished on or before the end of the last month
WHERE date(S.Arrival_date) <= date("now","start of month","-1 day")
GROUP BY C.Channel_name;


-- 5. What is the utilization rate for each hotel (that is the average
-- billable days of a hotel specified as the average utilization
-- of room bookings for the last 12 months)?

SELECT Name, Hotel_id, AVG(Utilisation) FROM
    (
    SELECT H.Name, R.Hotel_id, R.Room_name_or_number,
    COUNT(RA.date)/365.0 AS Utilisation
    FROM Room_allocation RA
    -- Line break below to ensure fit on the page.
    INNER JOIN Room R
    ON RA.(Hotel_id, Room_name_or_number) = R.(Hotel_id, Room_name_or_number)
    -- Only joining to retrieve hotel name:
    INNER JOIN Hotel H ON R.Hotel_id = H.Hotel_id
    WHERE date(RA.Date) >= date("now","-1 year")
    -- To ensure that only stays count as utilised days:
    AND Stay_id NOT NULL
    GROUP BY R.Hotel_id, R.Room_name_or_number
    )
GROUP BY Hotel_id;


-- 6. Calculate the Customer Value in terms of total spent
-- for each customer before the current booking.

-- Assuming total spent includes taxes paid
-- Stay only includes previous stays, so no need to filter anything.
SELECT S.Guest_id, SUM(IP.Amount) AS Total_spent FROM Stay S
INNER JOIN Invoice_payments IP ON IP.Stay_id = S.Stay_id
GROUP BY S.Guest_id
-- Only include Guest_ids that also have reservations.
HAVING S.Guest_id IN (SELECT DISTINCT Guest_id FROM Reservation);
-- Guest_id could be specified if looking for specific
-- HAVING S.Guest_id = 12345
```

A2

Logical schema:



Queries:

```sql
-- How many customers have stopped bringing their cars
-- after the first encounter with the dealer?

-- Assuming Customers have only one car
SELECT COUNT(*) AS Churns FROM (
  -- Subquery yields table with the number of times a customer brought
  -- their car to service and when the next service would be due,
  -- which can be analysed further to reveal more about
  -- the likelihood of coming to service more times.
  SELECT C.VIN, COUNT(S.Service_id), C.Next_due_service_date AS Total_services
  FROM Car C
  INNER JOIN Service S ON C.VIN=S.VIN
  GROUP BY C.VIN
  )
-- Only consider those customers churns, whose due service dates have already passed.
WHERE date(Next_due_service_date)<date("now")
AND Total_services = 1;


-- What is the relationship between the price of the service
-- and the age of the car in terms of
--(a)actual car age (e.g., mileage) and b) time with the current owner?

-- One query combined to extract information for both questions:
-- (a) can be answered via Mileage_at_service_time and
-- (b) can be answered with days_since_purchase


SELECT SUM(Costs.Item_cost), S.Mileage_at_service_time,
(julianday(S.service_date)-julianday(C.Purchase_date)) AS days_since_purchase
FROM Service S
INNER JOIN Car C ON C.VIN=S.VIN
```

```sql
INNER JOIN Cost_item Costs ON Costs.Service_id=S.Service_id
GROUP BY Service_id
```

Part B

```r
# Find all country directories
dirs <- list.dirs("./partB_data_files/")

# Drop root directory
dirs <- dirs[-1]

# Extract file names from every directory and save them
# in a named list of character vectors
files <- sapply(dirs,list.files,pattern="*.xlsx",USE.NAMES=T,simplify = F)

# Create empty list
datalist = list()


# Loop through directories and files within every directory
for (countrydir in names(files)) {
  for (file in files[[countrydir]]) {

    # Read in xlsx file, suppressing "New names" messages
    suppressMessages(excel<-countrydir %>% paste0("/",file) %>% read_xlsx())
    # Record sheet dimensions to use as reference points
    rowcount <- dim(excel)[1]
    colcount <- dim(excel)[2]
    # Extract main table, removing empty row and column in the
    # second row and column of the resulting table
    data<-excel[c(5,seq(7,rowcount-3)),c(1,seq(3,colcount))]
    # Use first row as header
    colnames(data) <- data[1,]
    # Drop header names from the dataframe
    data <- data[-1,]
    # Rename first column to year
    colnames(data)[1] <- "year"
    # Convert to long format
    data <- data %>% pivot_longer(2:length(data[1,]),
                                  names_to = "product",

                                  values_to = "value")
    # Add flow and country from the sheet
    data <- data %>% add_column(flow=excel[2,3] %>% unlist %>% unname,
                                country=excel[4,3] %>% unlist %>% unname)
    # Reorder columns to fit format requirements
    data <- data %>% select(country,year,flow,product,value)
    # Add resulting dataframe to the list
    datalist[[file]] <- data
  }
}
```

```r

# Bind dataframes together
fulldata <- do.call(bind_rows, datalist)

# Change to numeric data type and simultaneously coerce ".." values to NA
fulldata$value <- fulldata$value %>% as.numeric
```

```
## Warning in fulldata$value %>% as.numeric: NAs introduced by coercion
```

```r
# Count total rows in the dataframe and display
(totalrecords <- fulldata %>% nrow())
```

```
## [1] 573950
```

```r
# Group data by product and print number of records.
productrecords <- fulldata %>% group_by(product) %>% summarise(records=n())
# In case of every product, there are 8830 records.
(productrecords %>% count(records))
```

```
## # A tibble: 1 x 2
##   records     n
##     <int> <int>
## 1    8830    65
```

Part C

```r
# Root website link
link<-"https://data.food.gov.uk/catalog/datasets/38dd8d6a-5ab1-4f50-b753-ab33288e3200"

# Extract data download links from html page
xmllinks<-read_html(link) %>% html_nodes(".o-dataset-distribution--link") %>%
  html_attr("href")
# Filter to only .xml files (to remove a link that points to webpage index)
xmllinks<-xmllinks[xmllinks %>% str_detect(".xml$")]
# Filter to English language files (some were duplicates in Welsh)
xmllinks<-xmllinks[xmllinks %>% str_detect("en-GB")]

# Create directory to store XMLs.
dir.create("xmldata",showWarnings = F)

# Create empty list to store dataframes
xmldatalist=list()

# Create SQLite connection
con<-RSQLite::dbConnect(RSQLite::SQLite(),"ratings.db")

# Around 10 minutes runtime excl. download
if (!RSQLite::dbExistsTable(con,"fact_Rating")&!file.exists("C_output.rds")) {
  for (i in 1:length(xmllinks)) {
```

```
    # Get link based on loop variable.
    link <- xmllinks[i]

    # Create file path for file to be saved
    filename <- paste0("xmldata/",link %>% str_extract("(?<=/)[^/]+$"))

    # Download file if the file does not yet exist.
    if (!file.exists(filename)){
      download.file(link,filename)
    }

    # Read xml file and find all EstablishmentDetail nodes.
    nodes <- read_xml(filename) %>% xml_find_all("//EstablishmentDetail")

    # Extract ID column to be used as an index later.
    ids<-xml_child(nodes,"FHRSID") %>% xml_text %>% trimws

    # Find the amount of fields for every node that are populated
    nodelengths<-xml_length(nodes)

    # Find the names of the populated fields for every node
    nodenames<-xml_children(nodes) %>% xml_name

    # Find the values of the populated fields for every node
    nodevalues<-xml_children(nodes) %>% xml_text %>% trimws

    # Create a long dataframe with node names and values, indexed by
    # repeating the index node-length times.
    # After that, pivot to a wide-format.
    # Finally, add the values of the nested fields (Geocode and Scores)
    # separately, as they would have been merged into one column otherwise.
    df<-tibble(ID=rep(ids,nodelengths),
                   variable=nodenames,
                   values=nodevalues) %>%
      pivot_wider(names_from="variable",values_from="values") %>%
      add_column(Longitude=xml_child(nodes,search="Geocode") %>%
                   xml_child("Longitude") %>% xml_text,
                 Latitude=xml_child(nodes,search="Geocode") %>%
                   xml_child("Latitude") %>% xml_text,
                 Hygiene=xml_child(nodes,search="Scores") %>%
                   xml_child("Hygiene") %>% xml_text,
                 Structural=xml_child(nodes,search="Scores") %>%
                   xml_child("Structural") %>% xml_text,
                 ConfidenceInManagement=xml_child(nodes,search="Scores") %>%
                   xml_child("ConfidenceInManagement") %>% xml_text)

    # Remove duplicate and unnecessary columns
    df$Scores<-NULL
    df$Geocode<-NULL
    df$ID<-NULL

    # Progress tracker for the parsing process, prints a message every 10 files.
    xmldatalist[[filename]]<-df
```

```r
  if (i %% 10 == 0){
    paste0(i, " files processed from a total of ", length(xmllinks),".") %>%
      print()
  }

}
print("XML parsing finished.")

# Binds data together and detects column types.
ratingsdata<-do.call(bind_rows,xmldatalist) %>% type_convert()

# Decode HTML encoded characters and remove tags, newlines and tags.
ratingsdata$RightToReply<-ifelse(is.na(ratingsdata$RightToReply),
                                 NA,
                                 ratingsdata$RightToReply[
                                   !is.na(ratingsdata$RightToReply)] %>%
                                   textutils::HTMLdecode() %>%
                                   str_remove_all("</?p>") %>%
                                   str_remove_all("[\n\t]") %>%
                                   trimws)

# Convert to dimensional model
dim_LocalAuthority<-ratingsdata %>% select(LocalAuthorityCode,
                                           LocalAuthorityName,
                                           LocalAuthorityWebSite,
                                           LocalAuthorityEmailAddress) %>%
                                    distinct
dim_Date<-ratingsdata %>%
  select(RatingDate) %>%
  mutate(year = lubridate::year(RatingDate),
         month = lubridate::month(RatingDate),
         day = lubridate::day(RatingDate)) %>%
  distinct()

dim_BusinessType<-ratingsdata %>%
  select(BusinessType,BusinessTypeID) %>%
  distinct()

dim_RatingTypes<-ratingsdata %>%
  select(RatingKey,SchemeType,RatingValue) %>%
  distinct

fact_Rating <- ratingsdata %>% select(-c(LocalAuthorityName,
                                         LocalAuthorityWebSite,
                                         LocalAuthorityEmailAddress,
                                         BusinessType,
                                         RatingValue,
                                         SchemeType))

# Write to SQLite database
RSQLite::dbWriteTable(con,"fact_Rating",fact_Rating)
RSQLite::dbWriteTable(con,"dim_Date",dim_Date)
RSQLite::dbWriteTable(con,"dim_LocalAuthority",dim_LocalAuthority)
```

```
  RSQLite::dbWriteTable(con,"dim_BusinessType",dim_BusinessType)
  RSQLite::dbWriteTable(con,"dim_RatingTypes",dim_RatingTypes)

  # Save ratingsdata as RDS
  saveRDS(ratingsdata,"C_output.rds")
}

# Disconnect from database
RSQLite::dbDisconnect(con)
```

Part D

```
# Read in dataframe from part C
ratingsdata<-readRDS("C_output.rds")

paste(object.size(ratingsdata) / 1048576,"MB") %>% print
```

```
## [1] "217.903747558594 MB"
```

```
# As the dataframe is only about 218 MB, loading it into memory
# is easily possible on any modern computer
# Keeping the object in memory and running dplyr code
# proved faster than SQL in practice.

# SQL code provided for illustration throughout the code.
# Conencting to database
# db <- dbConnect(SQLite(), 'ratings.db')

# getting unique business type and Id for filters
BusinessTypeId <- ratingsdata %>% select(BusinessType,BusinessTypeID) %>% distinct()
# BusinessTypeId <- dbGetQuery(db, 'SELECT DISTINCT
#                              BusinessType, BusinessTypeID FROM dim_BusinessType;')


ui <- dashboardPage(

        dashboardHeader(
          title = "Ratings dashboard"
        ),
        #Sidebar definition
        dashboardSidebar(
          sidebarMenu(
            menuItem("Ratings over time",
                     tabName = "Tab1",
                     icon = icon("history")),
            menuItem("Sub-scores by ratings",
                     tabName = "Tab2",
                     icon = icon("project-diagram")),
            menuItem("Top local authorities by rating",
                     tabName = "Tab3",
```

```
                    icon = icon("map-marked"))
      )
    ),
    dashboardBody(
      tabItems(

        # First tab
        tabItem(tabName = "Tab1",
          fluidRow(
            box(width=12,
                h2("Density of specific rating values
                   over time based on business type"),
                br()," 
                Lower ratings (0-2) gaining more
                weight in the last few years.")
          ),
          fluidRow(
            box(title ="Rating Value slider", width=6,
               sliderInput(inputId = 'RatingValues','Rating Values:',0,5,2)),
            box(title="Business Type select", width=6,
               selectInput(
                     inputId = 'BusinessType',
                     label = 'Business Types:',
                     choices = BusinessTypeId$BusinessType,
                     selected="Restaurant/Cafe/Canteen"
          ))),
          fluidRow(
            box(width=12,
               #Plot Output
               plotlyOutput(
                outputId = 'Play1'
               )
        ))),

        # Second tab
        tabItem(tabName ="Tab2",
                fluidRow(
                  box(width=12,h2("Sub-score (penalties)
                                  distributions given rating value"),
                      br(),
                      "Most penalties (highest sub-scores) awarded
                      for lack of confidence in management.")
                ),
                fluidRow(
                  # Business type distribution plot
                  box(title ="Rating Value slider",width=12,
                         sliderInput(
                           inputId = 'RatingValues2',
                           'Rating Values:',
                           0,
                           5,
                           1,
                 ))),
```

```
                      fluidRow(
                         box(width=12,
                            #Plot Output
                            plotlyOutput(outputId = 'Play2',
                                         height = "500px",
                                         width="100%")
                         )
            )),

            # Third tab
            tabItem(tabName ="Tab3",
                    fluidRow(
                      box(width=12,
                          h2("Local authorities with highest
                             proportion of a rating value"),
                          br(),
                          "Waltham Forest has the highest proportion (4%)
                          of establishments with a rating of 0,
                          nearly 4 times as many as the runner-up.")
                    ),
                    fluidRow(
                      box(width=12,title="Rating Value Slider",
                          sliderInput(
                           inputId = 'RatingValues3',
                           'Rating Values:', 0, 5, 0)
                     )),
                     fluidRow(
                       box(width=12,
                        #Plot Output
                        plotlyOutput(
                           outputId = 'Play3',height = "500px",width="100%")
                     ))
            )
          )
        )
)


server <- function(input, output, session) {

    output$Play1 <- renderPlotly(
        {
          df <- ratingsdata %>%
            filter(SchemeType=="FHRS"& BusinessType==input$BusinessType) %>%
            select(RatingValue,RatingDate) %>%
            mutate(year=lubridate::year(RatingDate)) %>%
            count(year,RatingValue) %>%
            left_join(ratingsdata %>%
                        filter(SchemeType=="FHRS" &
                                 BusinessType==input$BusinessType) %>%
                        mutate(year=lubridate::year(RatingDate))  %>%
                        count(year) %>% rename(yearlycount=n),
                      by="year") %>%
```

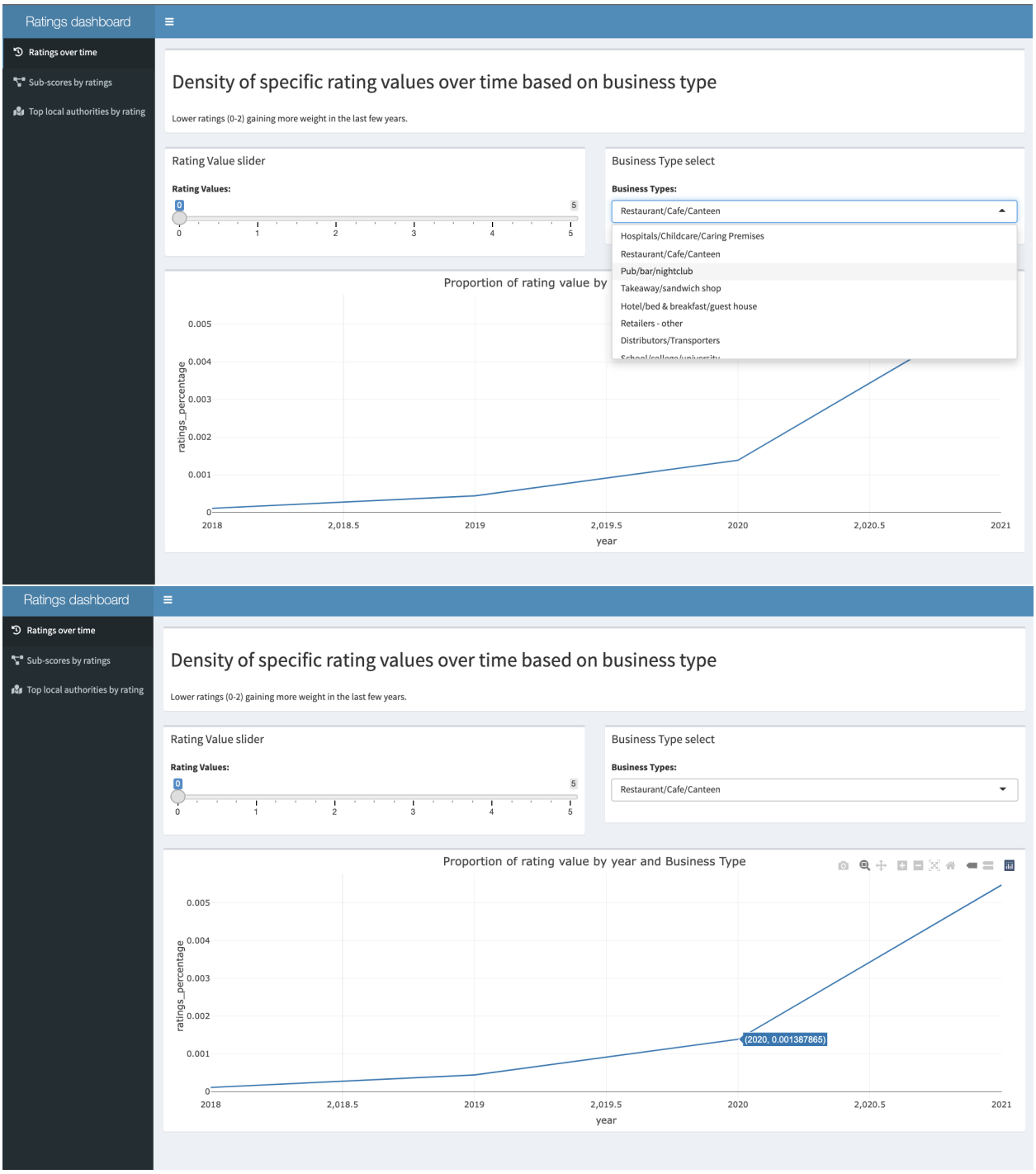
```
            mutate(ratings_percentage=n/yearlycount) %>%
            filter(RatingValue==input$RatingValues)

          # df <- dbGetQuery(db,
          #      statement =
          #      'SELECT D.ratingValue,
          #      COUNT(D.ratingValue)/CAST(yearly_count AS REAL) AS ratings_percentage,
          #      Date.Year FROM fact_rating F
          #      INNER JOIN dim_RatingTypes D ON F.RatingKey=D.RatingKey
          #      INNER JOIN dim_BusinessType B ON B.BusinessTypeID=F.BusinessTypeID
          #      INNER JOIN dim_Date Date ON Date.RatingDate=F.RatingDate
          #      LEFT JOIN (
          #          SELECT Date.Year, COUNT(D.ratingValue) AS yearly_count
          #          FROM fact_rating F
          #          INNER JOIN dim_RatingTypes D ON F.RatingKey=D.RatingKey
          #          INNER JOIN dim_BusinessType B ON B.BusinessTypeID=F.BusinessTypeID
          #          INNER JOIN dim_Date Date ON Date.RatingDate=F.RatingDate
          #          WHERE D.SchemeType = $Scheme
          #          AND B.BusinessType = $Btype
          #          GROUP BY Date.Year) DT ON DT.Year=Date.Year
          #      WHERE D.SchemeType = $Scheme
          #      AND B.BusinessType = $Btype
          #      AND D.ratingValue = $Rating
          #      GROUP BY D.ratingValue, Date.Year;'
          # ,list(Scheme="FHRS",Btype=input$BusinessType,Rating=input$RatingValues))

          plt1 <- plot_ly(df,x=~year,y=~ratings_percentage) %>%
            layout(title="Proportion of rating value by year and Business Type") %>%
            add_lines
          plt1
        }
    )

# 2. This navigation option describes the sub-score distributions of every rating value.

   output$Play2 <- renderPlotly(
      {
          df <- ratingsdata %>%
            filter(RatingValue==input$RatingValues2) %>%
            count(Hygiene) %>%
            add_column(Score="Hygiene") %>%
            rename(level=Hygiene) %>%
            bind_rows(ratingsdata %>%
                        filter(RatingValue==input$RatingValues2) %>%
                        count(Structural) %>%
                        add_column(Score="Structural") %>%
                        rename(level=Structural)
                      ) %>%
            bind_rows(ratingsdata %>%
                        filter(RatingValue==input$RatingValues2) %>%
                        count(ConfidenceInManagement) %>%
                        add_column(Score="ConfidenceInManagement") %>%
                        rename(level=ConfidenceInManagement)
```

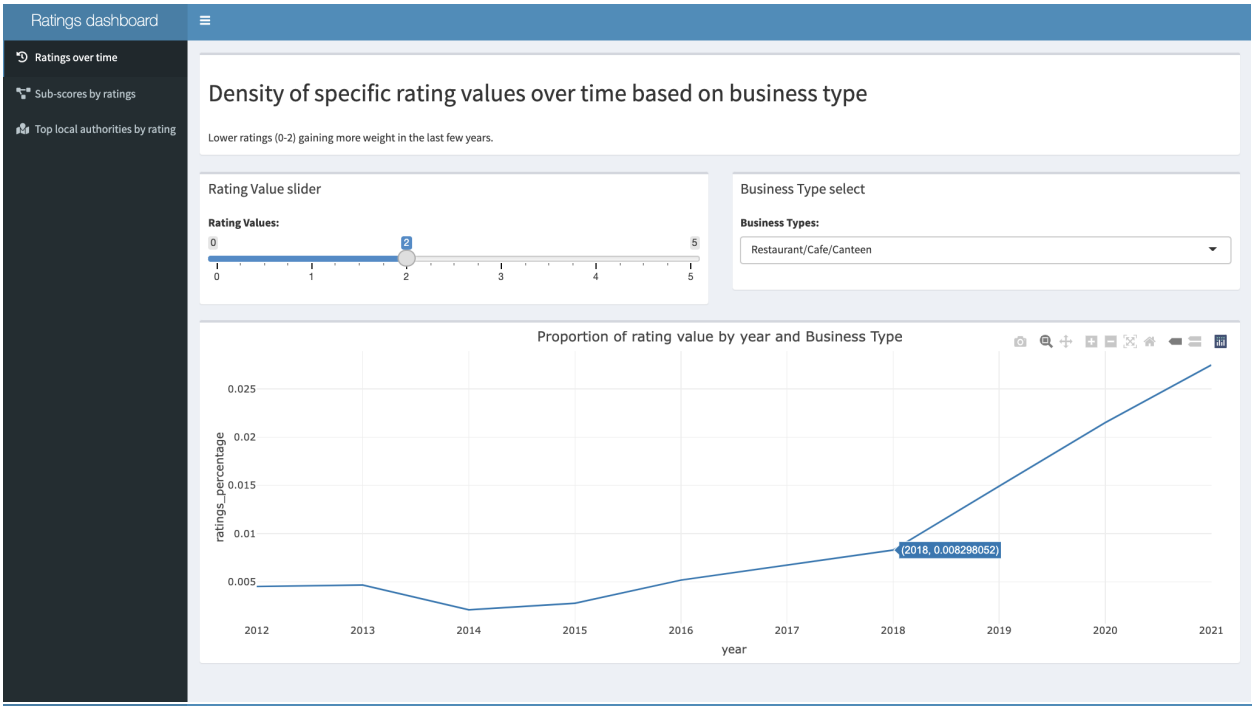
```
                ) %>%
    rename(Frequency=n) %>%
    drop_na()

# df<-dbGetQuery(db,
#   statement=
#   "SELECT D.RatingValue, 'Hygiene' AS Score,
#    F.Hygiene AS level, COUNT(F.Hygiene) AS Frequency
#    FROM fact_rating F
#    INNER JOIN dim_RatingTypes D ON F.RatingKey=D.RatingKey
#    INNER JOIN dim_BusinessType B ON B.BusinessTypeID=F.BusinessTypeID
#    WHERE D.RatingValue = $Rating
#    AND F.Hygiene NOT NULL
#    GROUP BY F.Hygiene
#    UNION
#    SELECT D.RatingValue, 'Structural' AS Score,
#    F.Structural AS level, COUNT(F.Structural) AS Frequency
#    FROM fact_rating F
#    INNER JOIN dim_RatingTypes D ON F.RatingKey=D.RatingKey
#    INNER JOIN dim_BusinessType B ON B.BusinessTypeID=F.BusinessTypeID
#    WHERE D.RatingValue = $Rating
#    AND F.Structural NOT NULL
#    GROUP BY F.Structural
#    UNION
#    SELECT D.RatingValue, 'ConfidenceInManagement' AS Score,
#    F.ConfidenceInManagement AS level,
#    COUNT(F.ConfidenceInManagement) AS Frequency
#    FROM fact_rating F
#    INNER JOIN dim_RatingTypes D ON F.RatingKey=D.RatingKey
#    INNER JOIN dim_BusinessType B ON B.BusinessTypeID=F.BusinessTypeID
#    WHERE D.RatingValue = $Rating
#    AND F.ConfidenceInManagement NOT NULL
#    GROUP BY F.ConfidenceInManagement;",
#  list(Rating=input$RatingValues2))




plt2<-plot_ly(df %>%
                filter(Score=="Hygiene"),
              x=~level,y=~Frequency) %>%
  add_bars(name="Hygiene")

plt3<-plot_ly(df %>%
                filter(Score=="Structural"),
              x=~level, y=~Frequency) %>%
  add_bars(name="Structural")

plt4<-plot_ly(df %>%
                filter(Score=="ConfidenceInManagement"),
              x=~level,y=~Frequency) %>%
  add_bars(name="Confidence In Management")
```

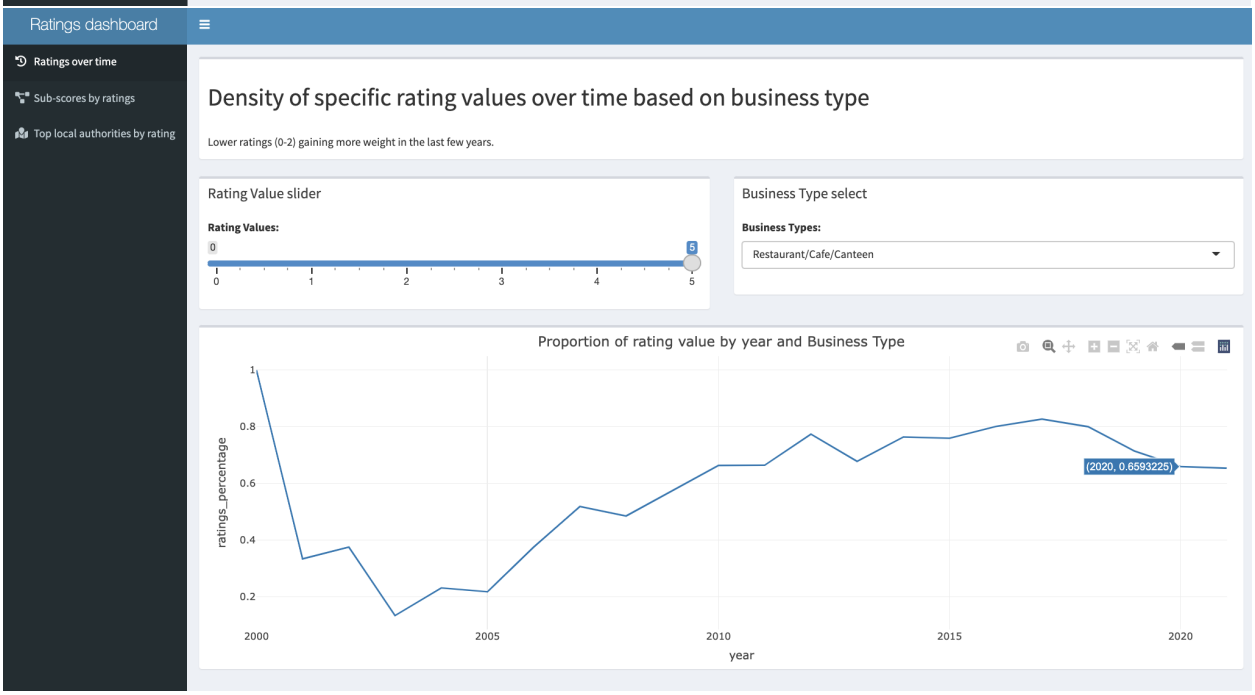
```
          subplot(plt2,plt3,plt4,nrows=1,shareY=T) %>%
            layout(title="Sub-score values (lower is better)")
      }
  )

# 3. Top ten Authority for selected rating value

# This chart will render the  plot showing top-10
# local Authorities based on rating at each level
    output$Play3 <- renderPlotly(
      {
          df<-ratingsdata %>%
            filter(SchemeType=="FHRS") %>%
            select(LocalAuthorityName,RatingValue) %>%
            group_by(LocalAuthorityName) %>%
            count(RatingValue) %>%
            left_join(ratingsdata %>%
                        filter(SchemeType=="FHRS") %>%
                        group_by(LocalAuthorityName) %>%
                        summarize(All_reviews=n()),
                      by="LocalAuthorityName") %>%
            mutate(Rating_percentage=n/All_reviews) %>%
            filter(RatingValue==input$RatingValues3) %>%
            arrange(desc(Rating_percentage)) %>%
            head(10)

          df <- dbGetQuery(db,
                           statement=
                           "SELECT D.ratingValue,
                           COUNT(D.ratingValue)/CAST(LocalAuthorityCount AS REAL)
                           AS Rating_percentage,
                           L.LocalAuthorityName
                           FROM fact_rating F
                           INNER JOIN dim_RatingTypes D
                           ON F.RatingKey=D.RatingKey
                           INNER JOIN dim_LocalAuthority L
                           ON L.LocalAuthorityCode=F.LocalAuthorityCode
                           LEFT JOIN (
                               SELECT L.LocalAuthorityName,
                               COUNT(D.ratingValue) AS LocalAuthorityCount
                               FROM fact_rating F
                               INNER JOIN dim_RatingTypes D
                               ON F.RatingKey=D.RatingKey
                               INNER JOIN dim_LocalAuthority L
                               ON L.LocalAuthorityCode=F.LocalAuthorityCode
                               WHERE D.SchemeType = $Scheme
                               GROUP BY L.LocalAuthorityName) DT
                           ON DT.LocalAuthorityName=L.LocalAuthorityName
                           WHERE D.SchemeType = $Scheme
                           AND D.ratingValue = $Rating
                           GROUP BY L.LocalAuthorityName, D.RatingValue
                           ORDER BY Rating_percentage DESC
                           LIMIT 10;",
```

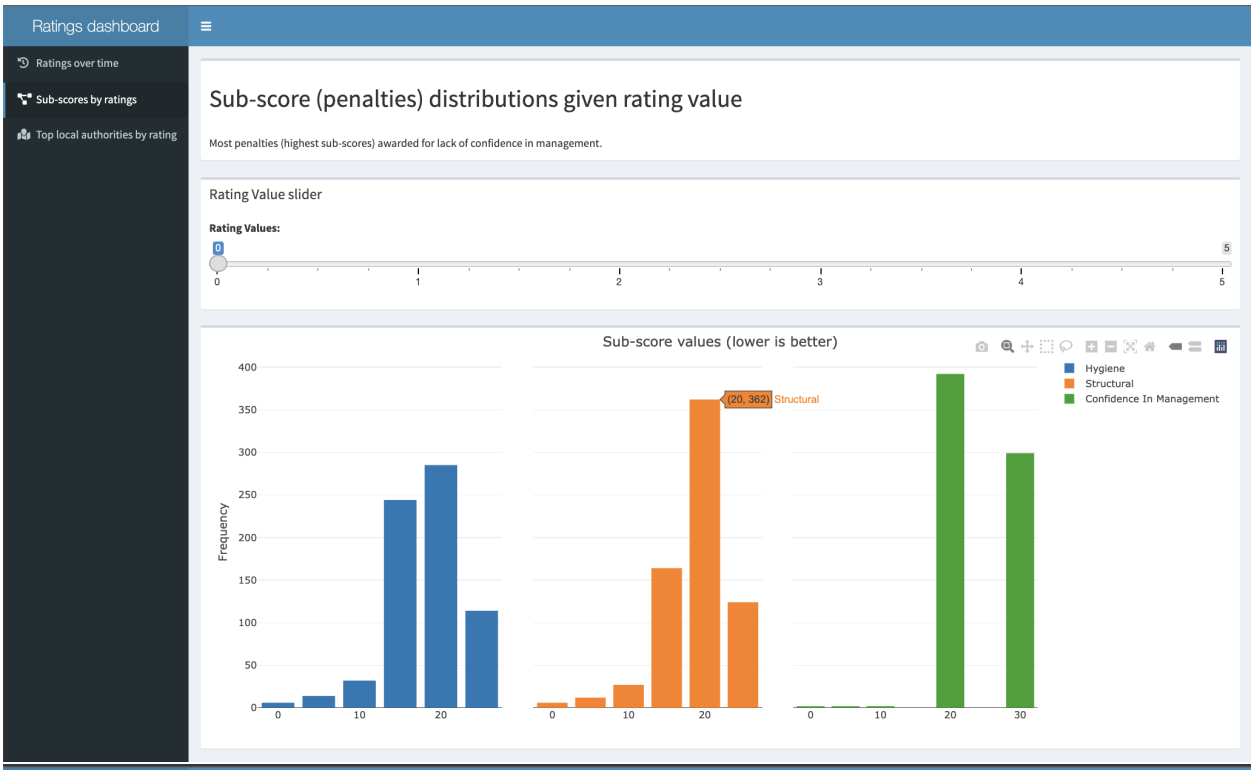
```
                            list(Scheme="FHRS",Rating=input$RatingValues3))

          plt4 <- plot_ly(df,x=~LocalAuthorityName,y=~Rating_percentage) %>%
            add_bars()
          plt4  %>% layout(xaxis = list(categoryorder = "total descending"),
                           title="Local Authorities with highest
                           percentage of given rating value")
      }
    )
}

# Complete app with UI and server components
# (commented out to allow for running through when knitting)
# shinyApp(ui, server)
# dbDisconnect(db)
```

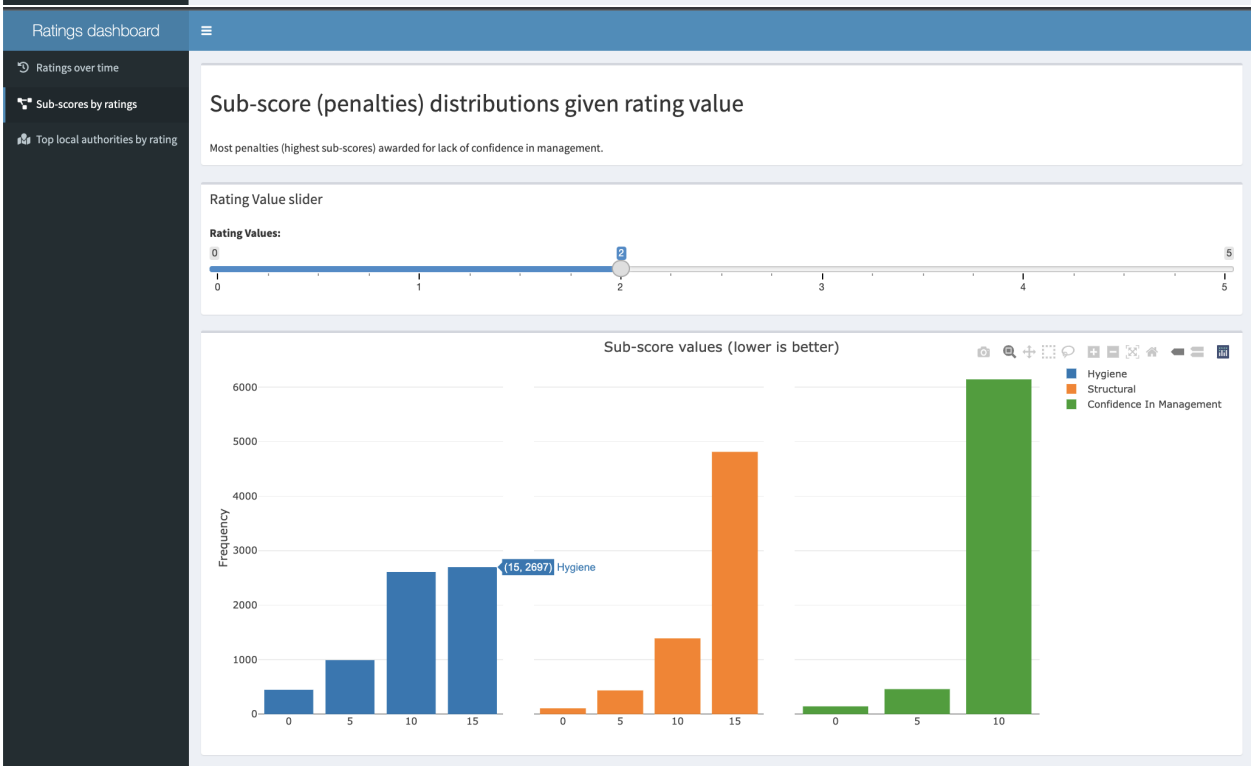
Screenshots of dashboard:



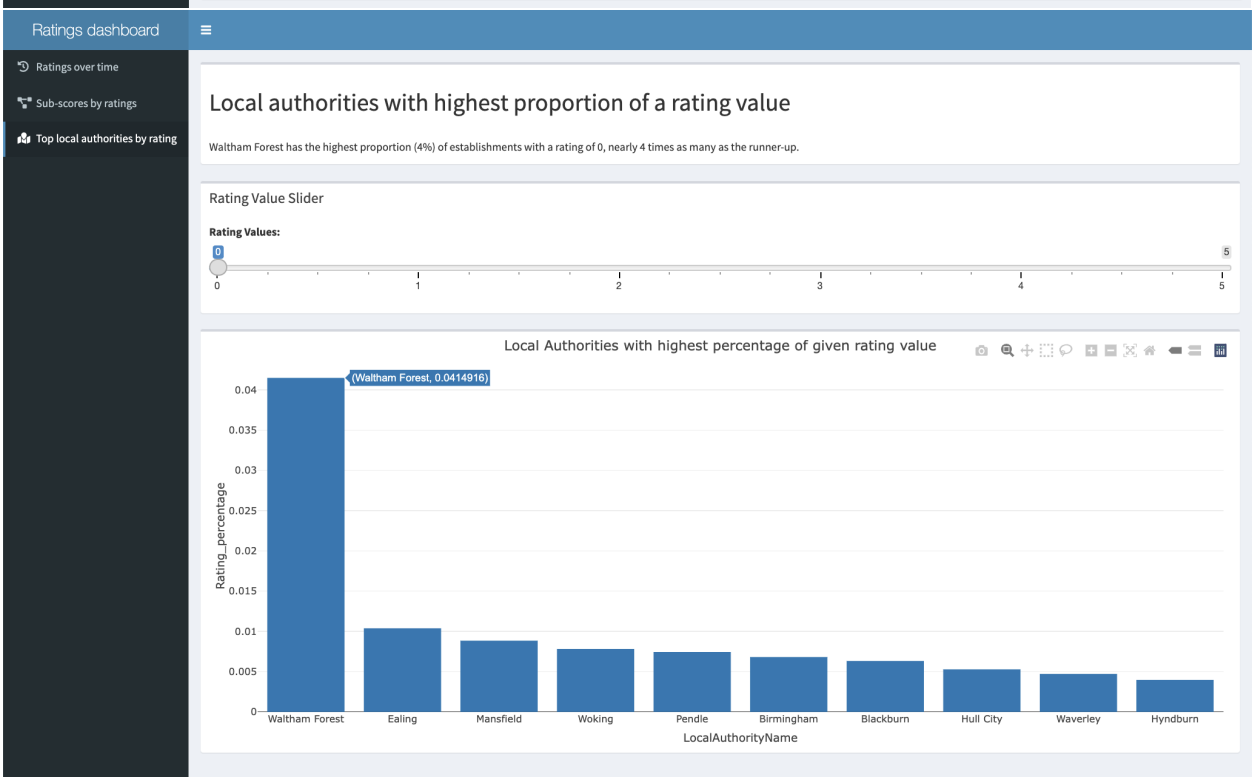

Ratings dashboard
Ratings over time
Sub-scores by ratings
Top local authorities by rating
Density of specific rating values over time based on business type
Lower ratings (0-2) gaining more weight in the last few years.
Rating Value slider
Rating Values:
0
2
5
0
1
2
3
4
5
Business Type select
Business Types:
Restaurant/Cafe/Canteen
Proportion of rating value by year and Business Type
ratings_percentage
0.025
0.02
0.015
0.01
0.005
(2018, 0.008298052)
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
year

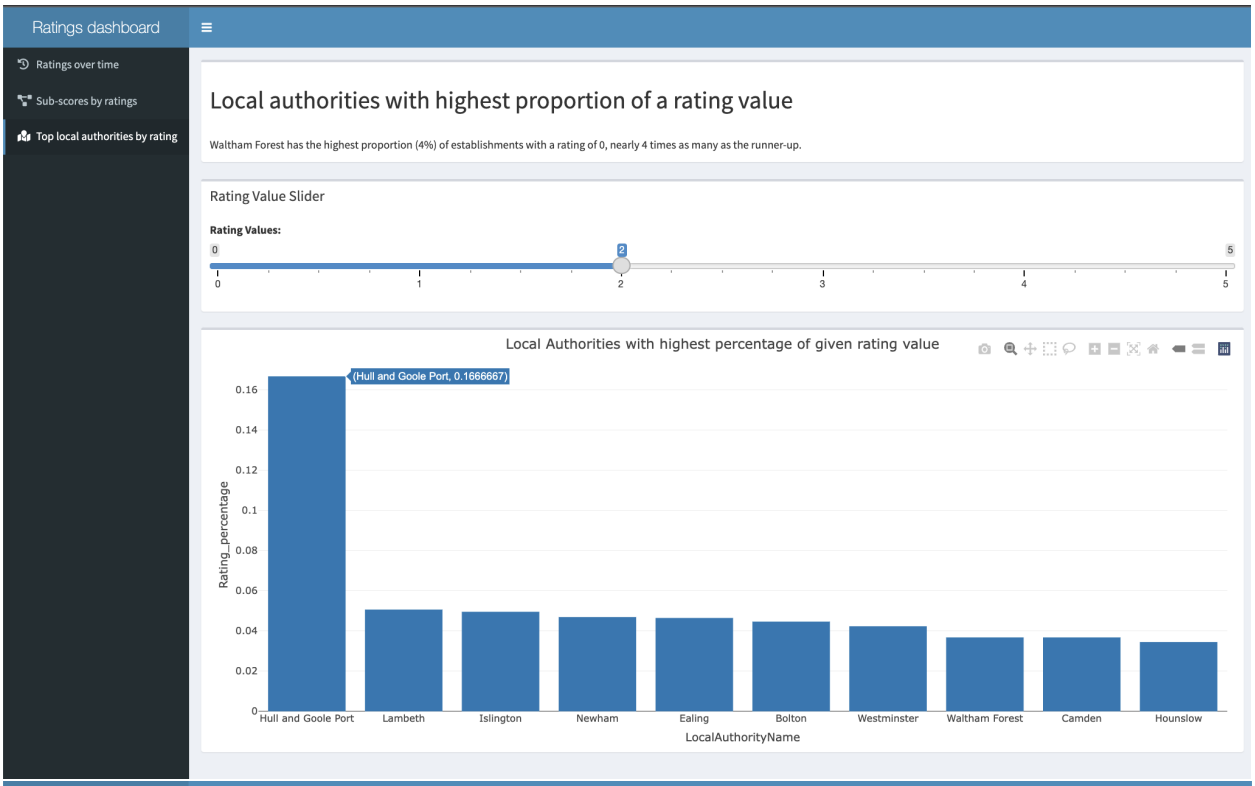

Ratings dashboard
Ratings over time
Sub-scores by ratings
Top local authorities by rating
Density of specific rating values over time based on business type
Lower ratings (0-2) gaining more weight in the last few years.
Rating Value slider
Rating Values:
0
5
0
1
2
3
4
5
Business Type select
Business Types:
Restaurant/Cafe/Canteen
Proportion of rating value by year and Business Type
ratings_percentage
1
0.8
0.6
0.4
0.2
(2020, 0.6593225)
2000
2005
2010
2015
2020
year

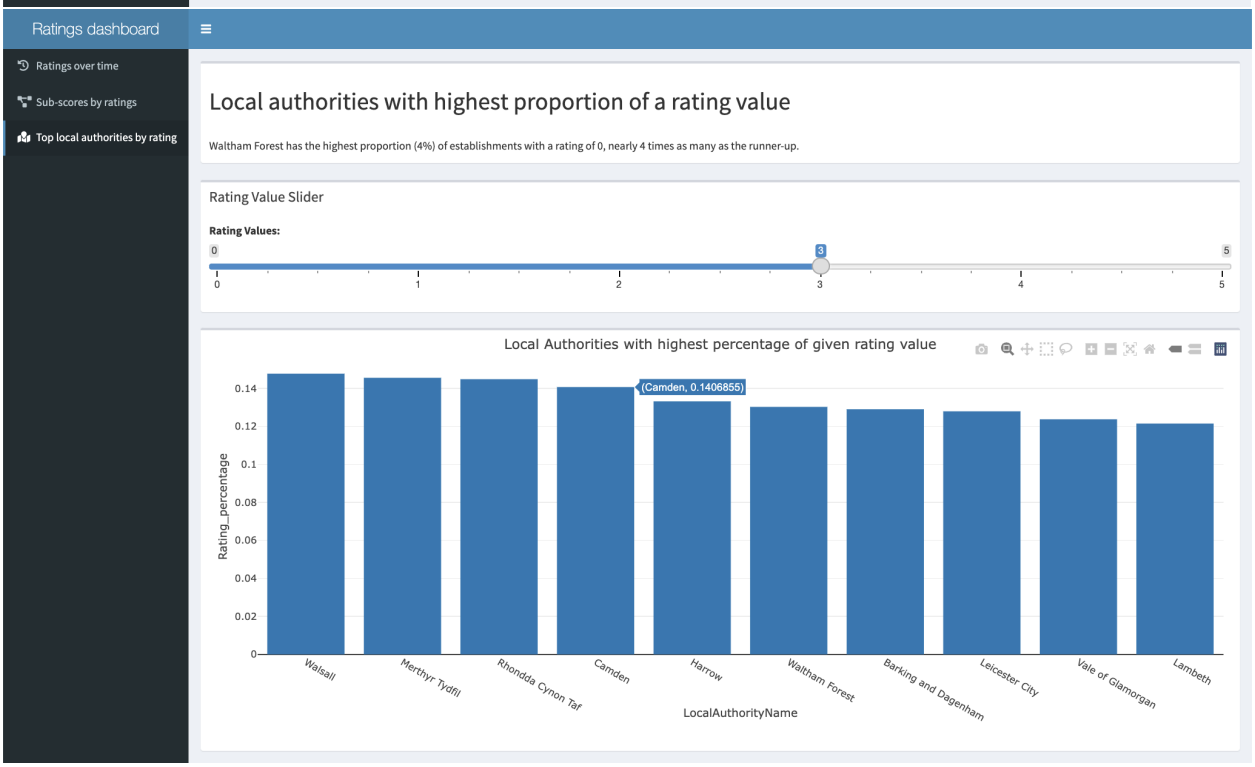

Ratings dashboard
Ratings over time
Sub-scores by ratings
Top local authorities by rating
Sub-score (penalties) distributions given rating value
Most penalties (highest sub-scores) awarded for lack of confidence in management.
Rating Value slider
Rating Values:
0
5
0
1
2
3
4
5
Sub-score values (lower is better)
Hygiene
Structural
Confidence In Management
(20, 362) Structural
Frequency
400
350
300
250
200
150
100
50
0
0
10
20
0
10
20
0
10
20
30

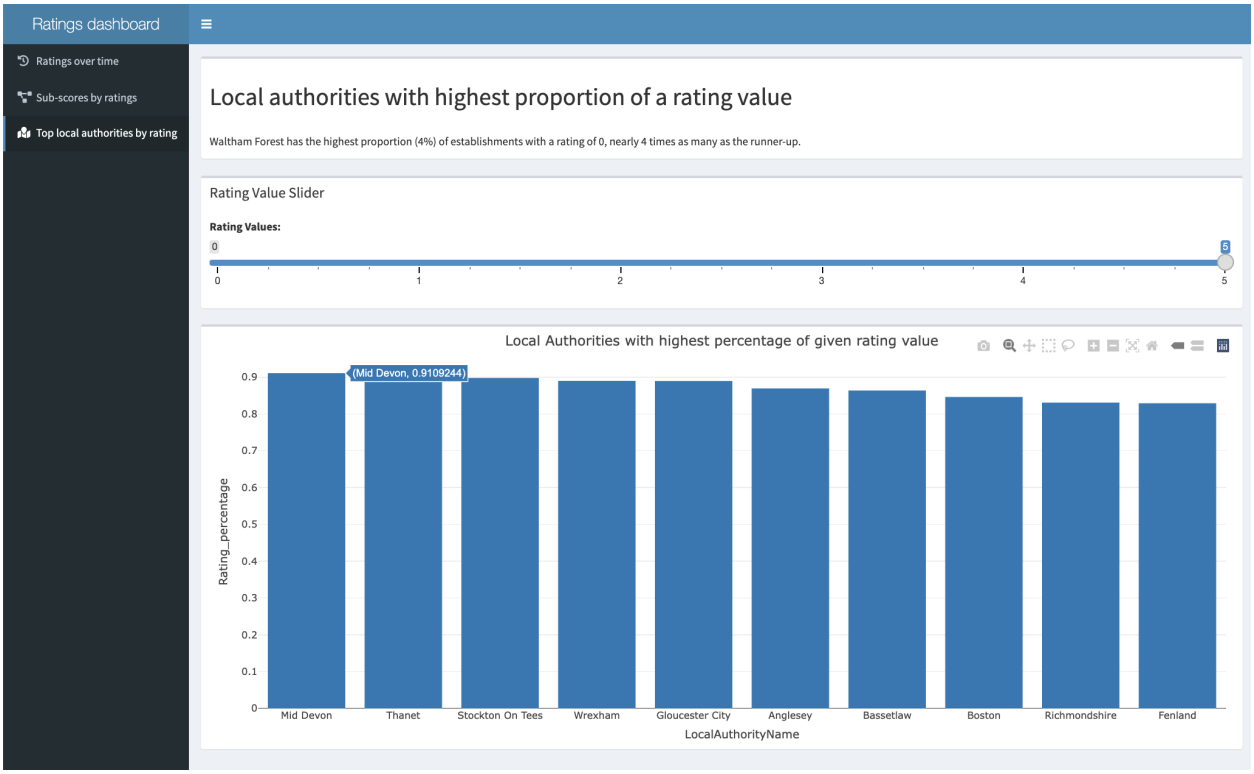

Ratings dashboard
Ratings over time
Sub-scores by ratings
Top local authorities by rating
Sub-score (penalties) distributions given rating value
Most penalties (highest sub-scores) awarded for lack of confidence in management.
Rating Value slider
Rating Values:
0
2
5
0
1
2
3
4
5
Sub-score values (lower is better)
Hygiene
Structural
Confidence In Management
(15, 2697) Hygiene
Frequency
6000
5000
4000
3000
2000
1000
0
0
5
10
15
0
5
10
15
0
5
10


Ratings dashboard
Ratings over time
Sub-scores by ratings
Top local authorities by rating
Sub-score (penalties) distributions given rating value
Most penalties (highest sub-scores) awarded for lack of confidence in management.
Rating Value slider
Rating Values:
0
5
0
1
2
3
4
5
Sub-score values (lower is better)
Hygiene
Structural
Confidence In Management
(0, 163.456k) Confidence I...
Frequency
200k
150k
100k
50k
0
0
5
0
5
0
5


Ratings dashboard
Ratings over time
Sub-scores by ratings
Top local authorities by rating
Local authorities with highest proportion of a rating value
Waltham Forest has the highest proportion (4%) of establishments with a rating of 0, nearly 4 times as many as the runner-up.
Rating Value Slider
Rating Values:
0
5
0
1
2
3
4
5
Local Authorities with highest percentage of given rating value
(Waltham Forest, 0.0414916)
Rating_percentage
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
Waltham Forest
Ealing
Mansfield
Woking
Pendle
Birmingham
Blackburn
Hull City
Waverley
Hyndburn
LocalAuthorityName


Ratings dashboard
Ratings over time
Sub-scores by ratings
Top local authorities by rating
Local authorities with highest proportion of a rating value
Waltham Forest has the highest proportion (4%) of establishments with a rating of 0, nearly 4 times as many as the runner-up.
Rating Value Slider
Rating Values:
Local Authorities with highest percentage of given rating value
(Hull and Goole Port, 0.1666667)
Rating_percentage
Hull and Goole Port
Lambeth
Islington
Newham
Ealing
Bolton
Westminster
Waltham Forest
Camden
Hounslow
LocalAuthorityName


Ratings dashboard
Ratings over time
Sub-scores by ratings
Top local authorities by rating
Local authorities with highest proportion of a rating value
Waltham Forest has the highest proportion (4%) of establishments with a rating of 0, nearly 4 times as many as the runner-up.
Rating Value Slider
Rating Values:
Local Authorities with highest percentage of given rating value
(Camden, 0.1406855)
Rating_percentage
Walsall
Merthyr Tydfil
Rhondda Cynon Taf
Camden
Harrow
Waltham Forest
Barking and Dagenham
Leicester City
Vale of Glamorgan
Lambeth
LocalAuthorityName


Ratings dashboard
Ratings over time
Sub-scores by ratings
Top local authorities by rating
Local authorities with highest proportion of a rating value
Waltham Forest has the highest proportion (4%) of establishments with a rating of 0, nearly 4 times as many as the runner-up.
Rating Value Slider
Rating Values:
0
5
0
1
2
3
4
5
Local Authorities with highest percentage of given rating value
(Mid Devon, 0.9109244)
Rating_percentage
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Mid Devon
Thanet
Stockton On Tees
Wrexham
Gloucester City
Anglesey
Bassetlaw
Boston
Richmondshire
Fenland
LocalAuthorityName