Analysis of Wildlife Strikes to Aircraft

Chandresh Lokesha

Spring 24

```
library(RMySQL)

killDbConnections <- function () {

  all_cons <- dbListConnections(MySQL())

  print(all_cons)

  for(con in all_cons)
    +  dbDisconnect(con)

  print(paste(length(all_cons), " connections killed."))

}
killDbConnections()
```

Practicum I CS5200

Connect to local DB

```
# Clearing the global vars
rm(list = ls())

# imports and connecting to DB
library(DBI)
library(RMySQL)
library(lubridate)
```

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

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

```
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
```

```
birdStrikeDBCon = dbConnect(RMySQL::MySQL(),
                            dbname='birdstrike',
                            host='localhost',
                            port=3306,
                            user='root',
                            password='password')
```

Connect to online DB

```
# Clearing the global vars
rm(list = ls())

# imports and connecting to DB
library(DBI)
library(RMySQL)
library(lubridate)
library(dplyr)
birdStrikeDBCon = dbConnect(RMySQL::MySQL(),
                            dbname='sql5690859',
                            host='sql5.freemysqlhosting.net',
                            port=3306,
                            user='sql5690859',
                            password='1vtjP7tGcZ')
```

not required for connecting to online DB

```
# to enable batch uploading of CSV to DB from client side
set global local_infile = TRUE;
```

```
# Drop the tables if they already exist
dbSendQuery(birdStrikeDBCon, "DROP TABLE IF EXISTS strikes;")
```

```
## <MySQLResult:1392579808,0,0>
```

```
dbSendQuery(birdStrikeDBCon, "DROP TABLE IF EXISTS conditions;")
```

```
## <MySQLResult:1392579808,0,1>
```

```
dbSendQuery(birdStrikeDBCon, "DROP TABLE IF EXISTS flights;")
```

```
## <MySQLResult:1393112264,0,2>
```

```
dbSendQuery(birdStrikeDBCon, "DROP TABLE IF EXISTS airports;")
```

```
## <MySQLResult:654547672,0,3>
```

Q4 creating tables

```
# Create table airport
dbSendQuery(birdStrikeDBCon, "CREATE TABLE airports (
  aid INTEGER PRIMARY KEY,
  airportName TEXT,
  airportState TEXT,
  airportCode VARCHAR(3) DEFAULT 'ZZZ'
);
")
```

```
## <MySQLResult:12,0,4>
```

```
# Create table Flights
dbSendQuery(birdStrikeDBCon, "CREATE TABLE flights (
  fid INTEGER PRIMARY KEY,
  date DATE,
  originAirport INTEGER,
  airlineName TEXT,
  aircraftType TEXT,
  isHeavy BOOLEAN,
  FOREIGN KEY (originAirport) REFERENCES airports(aid)
);
")
```

```
## <MySQLResult:12,0,5>
```

```
# Create table conditions
dbSendQuery(birdStrikeDBCon, "CREATE TABLE conditions (
  cid INTEGER PRIMARY KEY,
  sky_condition TEXT,
  explanation TEXT
);
")
```

```
## <MySQLResult:12,0,6>
```

```
# create table strikes
dbSendQuery(birdStrikeDBCon, "CREATE TABLE strikes (
  sid INTEGER PRIMARY KEY,
```

```
  fid INTEGER,
  numbirds INTEGER,
  impact TEXT,
  damage BOOLEAN,
  altitude INTEGER CHECK (altitude >= 0),
  conditions INTEGER,
  FOREIGN KEY (fid) REFERENCES flights(fid),
  FOREIGN KEY (conditions) REFERENCES conditions(cid)
);
")
```

```
## <MySQLResult:12,0,7>
```

```
# Insert sample data into the 'airports' table
dbSendQuery(birdStrikeDBCon, "INSERT INTO airports (aid, airportName, airportState, airportCode) VALUES
res <- dbExecute(birdStrikeDBCon, "SELECT * from airports")
print(res)

# Insert sample data into the 'flights' table
dbSendQuery(birdStrikeDBCon, "INSERT INTO flights (fid, date, originAirport, airlineName, aircraftType,
res <- dbExecute(birdStrikeDBCon, "SELECT * from flights")
print(res)

# Insert sample data into the 'conditions' table
dbSendQuery(birdStrikeDBCon, "INSERT INTO conditions (cid, sky_condition, explanation) VALUES (2, 'Over
res <- dbExecute(birdStrikeDBCon, "SELECT * from conditions")
print(res)

# Insert sample data into the 'strikes' table
dbSendQuery(birdStrikeDBCon, "INSERT INTO strikes (sid, fid, numbirds, impact, damage, altitude, condit
VALUES (3, 2, 5, 'Impact1', TRUE, 10000, 2);")
res <- dbExecute(birdStrikeDBCon, "SELECT * from strikes")
print(res)

# Fetch and print data from the 'airports' table
airports_data <- dbGetQuery(birdStrikeDBCon, "SELECT * FROM airports;")
print(airports_data)

# Fetch and print data from the 'flights' table
flights_data <- dbGetQuery(birdStrikeDBCon, "SELECT * FROM flights;")
print(flights_data)

# Fetch and print data from the 'conditions' table
conditions_data <- dbGetQuery(birdStrikeDBCon, "SELECT * FROM conditions;")
print(conditions_data)

strikes_data <- dbGetQuery(birdStrikeDBCon, "SELECT * FROM strikes;")
print(strikes_data)

dbSendQuery(birdStrikeDBCon, "DELETE FROM strikes;")
dbSendQuery(birdStrikeDBCon, "DELETE FROM flights;")
dbSendQuery(birdStrikeDBCon, "DELETE FROM airports;")
q4 <- dbSendQuery(birdStrikeDBCon, "DELETE FROM conditions;")
```

```
dbClearResult(q4)
```

Q5 load CSV into dataframe

```
# Load CSV file into a dataframe
bds.raw <- read.csv("BirdStrikesData-V3.csv", header = TRUE, sep = ",")
```

drop half of the rows as the db size exceeds 5mb on mysqlfree.net and suspends the account

```
bds.raw <- bds.raw[(nrow(bds.raw)/2 + 1):nrow(bds.raw), ]
```

```
print(head(bds.raw))
print(colnames(bds.raw))
print(nrow(bds.raw))
print(nrow(unique(bds.raw)))
```

Data Cleaning and pre-processing to format the date

1. We are normalizing the airline, airport using 'unknown'

```
bds.raw$flight_date<- parse_date_time(bds.raw$flight_date, orders=c("m-d-y H:M","m/d/y H:M"))
bds.raw$flight_date <- format(bds.raw$flight_date,"%Y-%m-%d")
```

```
arilineCase <- which(bds.raw$airline=='')
bds.raw[arilineCase,"airline"] <- 'unknown'

aircraftCase <- which(bds.raw$aircraft=='')
bds.raw[aircraftCase,"aircraft"] <- 'unknown'

airportCase <- which(bds.raw$airport=='')
bds.raw[airportCase,"airport"] <- 'unknown'
```

Add size mappings for wildlife size

```
size_mapping <- c("Small" = 1, "Medium" = 2, "Large" = 3)
bds.raw$wildlife_size_numeric <- size_mapping[bds.raw$wildlife_size]
```

omit flights without flight information

```
bds.raw <- bds.raw[which(bds.raw$airport!='' & bds.raw$model!='' & bds.raw$origin!='' ),]
```

```
print(head(bds.raw))
print(colnames(bds.raw))
print(nrow(bds.raw))
```

Insert the data into airports table

```
# Function: insert_airports
# Description: This function inserts airport data into the 'airports' table.
#
# Parameters:
#   df - Dataframe containing airport information.
#   dbconn - database connection string
insert_airports <- function(dbconn, df) {
  # Transform and clean data if needed

  df <- df[, c("airport", "origin")]

  df$airport <- trimws(df$airport)

  colnames(df) <- c("airportName", "airportState")

  # Create a unique set of airports
  unique_airports <- unique(df)

  # Add synthetic primary key 'aid'
  unique_airports$aid <- seq_len(nrow(unique_airports))

  # Insert data into the 'airports' table
  dbWriteTable(dbconn, "airports", unique_airports, append = TRUE, row.names = FALSE)
}

# Insert data into the 'airports' table
insert_airports(birdStrikeDBCon, bds.raw)
```

```
## [1] TRUE
```

```
# Display part of the 'airports' table
partial_airports <- dbGetQuery(birdStrikeDBCon, "SELECT * FROM airports LIMIT 5")
print(partial_airports)
total_rows <- dbGetQuery(birdStrikeDBCon, "SELECT COUNT(*) FROM airports")
print(total_rows)
partial_airports <- dbGetQuery(birdStrikeDBCon, "SELECT * FROM airports where airportName = 'ABERDEEN RI
print(partial_airports)
```

Insert the data into flights table

```
# Function: insert_flights
# Description: Inserts flight data into the 'flights' table.
```

```
#
# Parameters:
#   conn - Database connection.
#   df - Dataframe containing flight information.
insert_flights <- function(conn, df) {
  # Transform and clean data if needed
  # print(df)
  df <- df[, c("airline", "flight_date", "origin",  "aircraft", "heavy_flag")]
  colnames(df) <- c("airlineName", "date", "origin",  "aircraftType", "isHeavy")
  #print(df)
  df$date <- as.Date(df$date)  # Convert date to Date type

  # Map 'isHeavy' values to boolean
  df$isHeavy <- tolower(df$isHeavy) == "yes"

  # Map 'originAirport' values to corresponding 'aid' values from the 'airports' table
  airports_mapping <- dbGetQuery(conn, "SELECT * FROM airports")
  df$originAirport <- airports_mapping$aid[match(df$origin, airports_mapping$airportState)]

  df$origin <- NULL

  # Create a unique set of flights
  unique_flights <- unique(df)

  # Add synthetic primary key 'fid'
  unique_flights$fid <- seq_len(nrow(unique_flights))

  # Insert data into the 'flights' table
  dbWriteTable(conn, "flights", unique_flights, append = TRUE, row.names = FALSE)
}

# insert into flights
insert_flights(birdStrikeDBCon, bds.raw)
```

```
## [1] TRUE
```

```
# Display part of the 'flights' table
partial_airports <- dbGetQuery(birdStrikeDBCon, "SELECT * FROM flights LIMIT 5")
print(partial_airports)

total_cnt_airlines <- dbGetQuery(birdStrikeDBCon, "SELECT COUNT(*) FROM flights")
print(total_cnt_airlines)
```

Inserting into conditions table

```
# Function: insert_conditions
# Description: Inserts sky conditions data into the 'conditions' table.
#
# Parameters:
#   conn - Database connection.
#   df - Dataframe containing sky conditions information.
```

```
insert_conditions <- function(conn, df) {
  # Transform and clean data if needed
  df <- df[, c("sky_conditions")]

  # Create a data frame with the desired column name
  unique_conditions <- unique(data.frame(sky_condition = df))

  # Add an empty 'explanation' column
  unique_conditions$explanation <- NA

  # Add synthetic primary key 'cid'
  unique_conditions$cid <- seq_len(nrow(unique_conditions))

  # print(unique_conditions)

  # Insert data into the 'conditions' table
  dbWriteTable(conn, "conditions", unique_conditions, append = TRUE, row.names = FALSE)
}

insert_conditions(birdStrikeDBCon, bds.raw)
```

```
## [1] TRUE
```

```
# Display part of the 'conditions' table
partial_conditions <- dbGetQuery(birdStrikeDBCon, "SELECT * FROM conditions LIMIT 5")
print(partial_conditions)

total_cnt_condts <- dbGetQuery(birdStrikeDBCon, "SELECT COUNT(*) FROM conditions")
print(total_cnt_condts)
```

inserting into strikes table

```
# Function: insert_strikes
# Description: Inserts bird strike data into the 'strikes' table.
#
# Parameters:
#   conn - Database connection.
#   df - Dataframe containing bird strike information.

insert_strikes <- function(conn, df) {
  df <- df[, c("flight_date","wildlife_size_numeric", "impact", "damage", "altitude_ft", "sky_conditions
  colnames(df) <- c("date","numbirds", "impact", "damage", "altitude", "conditions", "aircraft", "airli

  df$isHeavy <- tolower(df$isHeavy) == "yes"

  # Map 'damage' values to boolean
  df$damage <- tolower(df$damage) == "damage"


  # Convert altitude to integer
```

```
  df$altitude <- as.integer(df$altitude)

  # Map 'airport' to corresponding 'fid' values from the 'flights' table
  flights_mapping <- dbGetQuery(conn, "SELECT * from flights")

  df$fid <- flights_mapping$fid[match(
  df$date, flights_mapping$date) &
  match(df$aircraft, flights_mapping$aircraftType) &
  match(df$airline, flights_mapping$airlineName) &
  match(df$isHeavy, flights_mapping$isHeavy)]

  # Map 'sky_conditions' to corresponding 'cid' values from the 'conditions' table
  conditions_mapping <- dbGetQuery(conn, "SELECT * FROM conditions")
  df$conditions <- conditions_mapping$cid[match(df$conditions, conditions_mapping$sky_condition)]

  # Create a unique set of strikes
  unique_strikes <- unique(df)
  # print(length(unique_conditions))  # Print the count of unique values
  # Add synthetic primary key 'sid'
  unique_strikes$sid <- seq_len(nrow(unique_strikes))

  # Remove 'date', 'aircraft', and 'airlineName' columns before inserting into the 'strikes' table
  columns_to_remove <- c("date", "aircraft", "airline", "isHeavy", "origin")
  unique_strikes <- unique_strikes[, !(names(unique_strikes) %in% columns_to_remove)]

  # Insert data into the 'strikes' table
  dbWriteTable(conn, "strikes", unique_strikes, append = TRUE, row.names = FALSE)
}

insert_strikes(birdStrikeDBCon, bds.raw)
```

```
## [1] TRUE
```

```
# Display part of the 'strikes' table
partial_strikes <- dbGetQuery(birdStrikeDBCon, "SELECT * FROM strikes LIMIT 5")
print(partial_strikes)
total_cnt_strikes <- dbGetQuery(birdStrikeDBCon, "SELECT COUNT(*) FROM strikes")
print(total_cnt_strikes)
```

Q8 Top Airports with Strikes

```
SELECT a.airportState, COUNT(s.sid) as incidentCount
FROM airports a
JOIN flights f ON a.aid = f.originAirport
JOIN strikes s ON f.fid = s.fid
GROUP BY a.airportState
ORDER BY incidentCount DESC
LIMIT 10;
```

Table 1: Displaying records 1 - 10

airportState	incidentCount
California	1548
Texas	1474
Florida	1335
New York	857
Pennsylvania	669
Missouri	640
Illinois	638
Kentucky	530
Ohio	528
Hawaii	473

Analysis by Airline

Online DB on freeSql.net is not supporting WITH i.e CTE clause and hence added another chunk below

```
WITH AirlineIncidentCounts AS (
  SELECT f.airlineName, COUNT(s.sid) as incidentCount
  FROM flights f
  JOIN strikes s ON f.fid = s.fid
  GROUP BY f.airlineName
)

SELECT airlineName, incidentCount
FROM AirlineIncidentCounts
WHERE incidentCount > (SELECT AVG(incidentCount) FROM AirlineIncidentCounts)
ORDER BY incidentCount DESC;
```

```
SELECT f.airlineName, COUNT(s.sid) AS incidentCount
FROM flights f
JOIN strikes s ON f.fid = s.fid
GROUP BY f.airlineName
HAVING COUNT(s.sid) > (
    SELECT AVG(incidentCount)
    FROM (
        SELECT COUNT(sid) AS incidentCount
        FROM strikes
        GROUP BY fid
    ) AS subquery
)
ORDER BY incidentCount DESC;
```

Table 2: Displaying records 1 - 10

airlineName	incidentCount
SOUTHWEST AIRLINES	2749

airlineName	incidentCount
BUSINESS	2015
AMERICAN AIRLINES	1310
DELTA AIR LINES	932
US AIRWAYS	875
AMERICAN EAGLE AIRLINES	622
SKYWEST AIRLINES	567
JETBLUE AIRWAYS	439
UPS AIRLINES	367
UNITED AIRLINES	336

Analysis by Month on numberOfStrikes

```
# Execute SQL query and store the result in a dataframe
monthly_analysis_by_strikes <- dbGetQuery(birdStrikeDBCon, "
  SELECT
    MONTHNAME(f.date) AS Month,
    COUNT(s.sid) AS NumberOfStrikes
  FROM
    flights f
  JOIN
    strikes s ON f.fid = s.fid
  GROUP BY
    MONTHNAME(f.date)
  ORDER BY
    NumberOfStrikes DESC;
")
```

```
# Display the dataframe
print(monthly_analysis_by_strikes)
```

Analysis by NumberOfBirds

```
monthly_analysis <- dbGetQuery(birdStrikeDBCon, "SELECT
  MONTHNAME(f.date) AS Month,
  SUM(s.numbirds) AS NumberOfBirds
FROM
  flights f
JOIN
  strikes s ON f.fid = s.fid
GROUP BY
  MONTHNAME(f.date)
ORDER BY
  NumberOfBirds DESC;
")

# print(monthly_analysis)
```

Trend by Month of Number of Birds

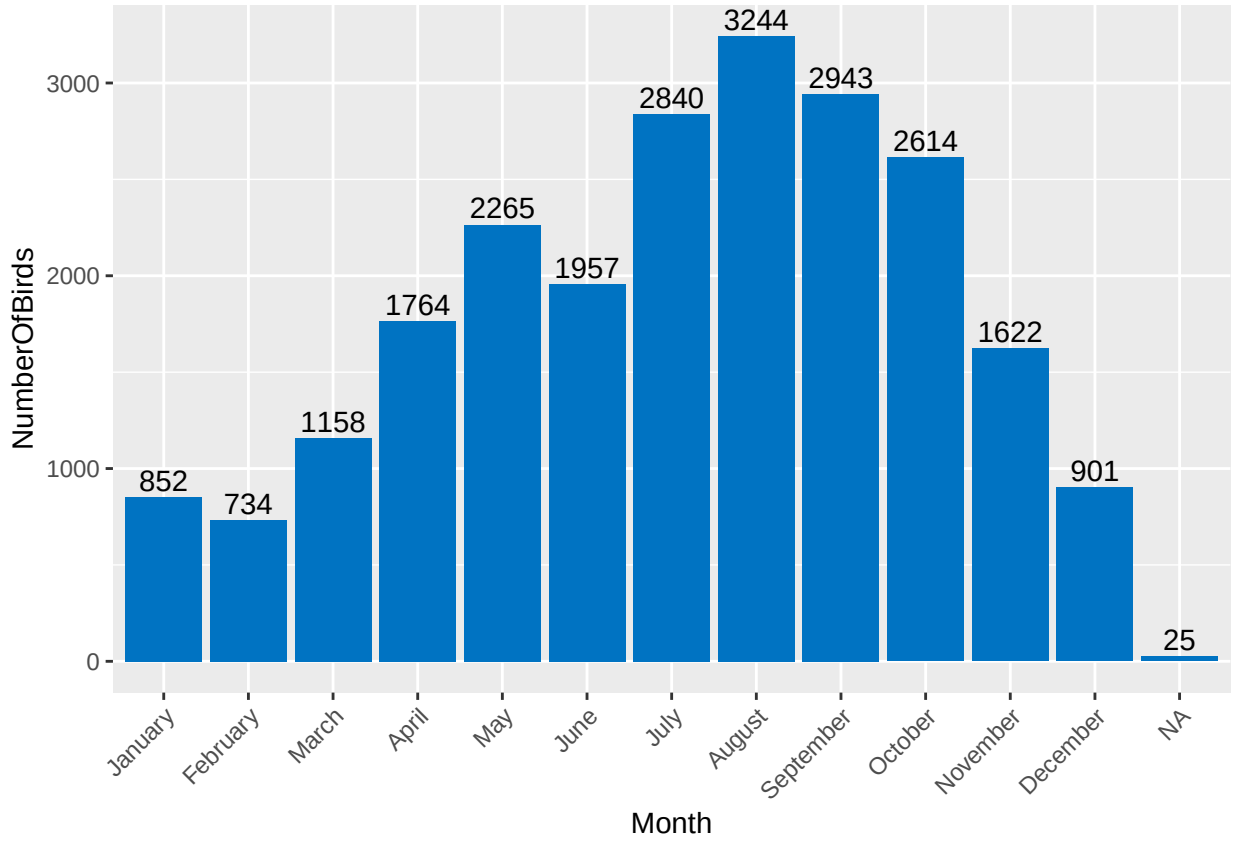
```
library(ggplot2)

# Assuming 'monthly_analysis' is the dataframe from Question 10
# Define the order of months
month_order <- c("January", "February", "March", "April", "May", "June", "July", "August", "September",

# Convert 'Month' to a factor with specified levels
monthly_analysis$Month <- factor(monthly_analysis$Month, levels = month_order)

# Order the dataframe by the new factor levels
monthly_analysis <- monthly_analysis[order(monthly_analysis$Month), ]

df <- monthly_analysis
f <- ggplot(df, aes(x = Month, y = NumberOfBirds))
f + geom_col(fill = "#0073C2FF") +
  geom_text(aes(label = NumberOfBirds), vjust = -0.3) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Trend by Month of Number of Strikes

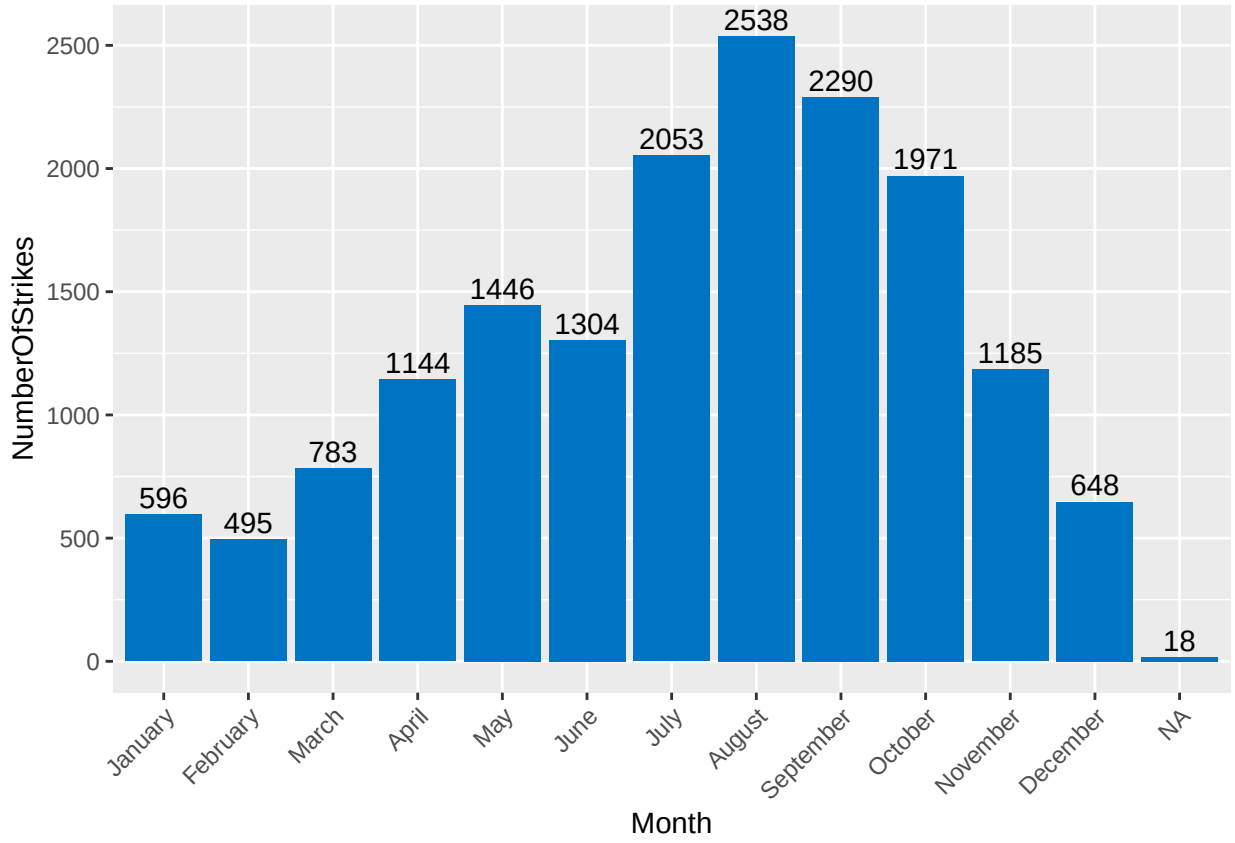
```r
library(ggplot2)

monthly_analysis <- monthly_analysis_by_strikes
# Assuming 'monthly_analysis' is the dataframe from Question 10
# Define the order of months
month_order <- c("January", "February", "March", "April", "May", "June", "July", "August", "September",

# Convert 'Month' to a factor with specified levels
monthly_analysis$Month <- factor(monthly_analysis$Month, levels = month_order)

# Order the dataframe by the new factor levels
monthly_analysis <- monthly_analysis[order(monthly_analysis$Month), ]

df <- monthly_analysis
f <- ggplot(df, aes(x = Month, y = NumberOfStrikes))
f + geom_col(fill = "#0073C2FF") +
  geom_text(aes(label = NumberOfStrikes), vjust = -0.3) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```sql
-- Create the audit_log table if it does not exist
CREATE TABLE IF NOT EXISTS audit_log (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    modification_type VARCHAR(50),
    table_name VARCHAR(50),
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    removed_sid INT
);
```

```
# Drop the existing procedure if it exists

dbExecute(birdStrikeDBCon, "DROP PROCEDURE IF EXISTS RemoveStrikeAndLog;")
```

```
## [1] 0
```

```
-- Create the stored procedure

CREATE PROCEDURE RemoveStrikeAndLog(IN sidToRemove INT)
BEGIN
    DECLARE removed_fid INT;

    -- Get the fid associated with the strike to be removed
    SELECT fid INTO removed_fid FROM strikes WHERE sid = sidToRemove;

    -- Log the removal in the audit_log table
    INSERT INTO audit_log (modification_type, table_name, removed_sid)
    VALUES ('Removal', 'strikes', sidToRemove);

    -- Remove the strike from the strikes table
    DELETE FROM strikes WHERE sid = sidToRemove;

    -- Optionally, additional cleanup/handle cascading deletes

    SELECT 'Strike removed successfully' AS result;
END
```

```
# Call the stored procedure
dbSendQuery(birdStrikeDBCon, "CALL RemoveStrikeAndLog(1)")
```

```
## <MySQLResult:1401561120,0,26>
```

```
dbDisconnect(birdStrikeDBCon)
```

```
## [1] TRUE
```

```
# test for local
birdStrikeDBCon = dbConnect(RMySQL::MySQL(),
                            dbname='birdstrike',
                            host='localhost',
                            port=3306,
                            user='root',
                            password='password')
result <- dbGetQuery(birdStrikeDBCon, "SELECT * FROM audit_log")
print(result)
result2 <- dbGetQuery(birdStrikeDBCon, "SELECT * from strikes where sid = '1'")
print(result2)

# dbSendQuery(birdStrikeDBCon, "Delete from audit_log")
dbDisconnect(birdStrikeDBCon)
```