ADS-506 Team 1 Final Project

Graham Ward, Jun Clemente, & Sasha Libolt

```r
library(tidyverse)
library(fpp3)
library(gt)
library(skimr)
library(scales)
```

Exploratory Data Analysis

Quick Summary

```r
df <- read_csv("C:/Users/sasha/OneDrive/Documents/Datasets/calls.csv")
```

```
Rows: 275655 Columns: 8
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (2): Communication Type, Sub Communication Type
dbl  (4): Wait Time, Time Interacting, Hold Time, Wrap Up Time
dttm (2): Start Time, End Time

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
# remove spaces from column names
colnames(df) <- gsub(" ", "", colnames(df))
head(df)
```

```
# A tibble: 6 x 8
  StartTime           EndTime             CommunicationType SubCommunicationType
```

```
  <dttm>              <dttm>              <chr>              <chr>
1 2022-03-21 14:13:52 2022-03-21 14:57:26 messaging          text
2 2022-03-21 16:48:33 2022-03-21 17:02:46 messaging          text
3 2022-03-21 16:51:18 2022-03-21 16:57:17 messaging          text
4 2022-03-21 16:57:05 2022-03-21 17:00:37 phone              inbound
5 2022-03-21 17:03:31 2022-03-21 17:04:01 messaging          text
6 2022-03-21 17:09:18 2022-03-21 17:09:49 phone              inbound
# i 4 more variables: WaitTime <dbl>, TimeInteracting <dbl>, HoldTime <dbl>,
#   WrapUpTime <dbl>
```

```
# quick summary of dataframe
summary(df)
```

```
   StartTime                          EndTime
 Min.   :2022-03-21 14:13:52.00   Min.   :2022-03-21 14:57:26.00
 1st Qu.:2022-11-17 15:41:36.00   1st Qu.:2022-11-17 15:46:31.50
 Median :2023-07-06 22:04:04.00   Median :2023-07-06 22:09:31.00
 Mean   :2023-07-14 15:29:03.66   Mean   :2023-07-14 15:36:04.43
 3rd Qu.:2024-03-12 13:09:51.00   3rd Qu.:2024-03-12 13:13:58.00
 Max.   :2024-10-31 23:57:27.00   Max.   :2024-10-31 23:59:40.00

 CommunicationType  SubCommunicationType    WaitTime        TimeInteracting
 Length:275655      Length:275655        Min.   :      0   Min.   :    0
 Class :character   Class :character     1st Qu.:     10   1st Qu.:   37
 Mode  :character   Mode  :character     Median :     18   Median :   96
                                         Mean   :    119   Mean   :  132
                                         3rd Qu.:     86   3rd Qu.:  177
                                         Max.   :4126179   Max.   :93296
                                                           NA's   :1
    HoldTime          WrapUpTime
 Min.   :   0.000   Min.   :     0.0
 1st Qu.:   0.000   1st Qu.:     5.0
 Median :   0.000   Median :    25.0
 Mean   :   6.317   Mean   :   141.4
 3rd Qu.:   0.000   3rd Qu.:   159.0
 Max.   :1284.000   Max.   :112600.0
 NA's   :1          NA's   :1
```

Columns with missing values

```r
# show columns with missing values
sapply(df, function(x) sum(is.na(x)))
```

```
           StartTime              EndTime    CommunicationType 
                   0                    0                    0 
SubCommunicationType             WaitTime      TimeInteracting 
                   0                    0                    1 
            HoldTime           WrapUpTime 
                   1                    1 
```

```r
# show rows that have missing values
rows_with_na <- df[!complete.cases(df),]
rows_with_na
```

```
# A tibble: 1 x 8
  StartTime           EndTime             CommunicationType SubCommunicationType
  <dttm>              <dttm>              <chr>             <chr>               
1 2024-10-05 18:15:15 2024-10-05 18:16:49 phone             inbound             
# i 4 more variables: WaitTime <dbl>, TimeInteracting <dbl>, HoldTime <dbl>,
#   WrapUpTime <dbl>
```

```r
# only one row missing values. remove row from dataset
df_clean <- na.omit(df)
df_clean[!complete.cases(df_clean),]
```

```
# A tibble: 0 x 8
# i 8 variables: StartTime <dttm>, EndTime <dttm>, CommunicationType <chr>,
#   SubCommunicationType <chr>, WaitTime <dbl>, TimeInteracting <dbl>,
#   HoldTime <dbl>, WrapUpTime <dbl>
```

```r
# check for rows with NA, Inf, or -Inf
rows_with_non_finite <- df_clean %>%
  filter(
    if_any(c(HoldTime, TimeInteracting, WaitTime, WrapUpTime), ~ !is.finite(.))
  )
rows_with_non_finite
```

```
# A tibble: 0 x 8
# i 8 variables: StartTime <dttm>, EndTime <dttm>, CommunicationType <chr>,
#   SubCommunicationType <chr>, WaitTime <dbl>, TimeInteracting <dbl>,
#   HoldTime <dbl>, WrapUpTime <dbl>
```

Detailed view of data

```
skim(df_clean)
```

Table 1: Data summary

Name	df_clean
Number of rows	275654
Number of columns	8
Column type frequency:	
character	2
numeric	4
POSIXct	2
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
CommunicationType	0	1	5	9	0	3	0
SubCommunicationType	0	1	4	9	0	5	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
WaitTime	0	1	118.69	7975.21	0	10	18	86	4126179	
TimeInteracting	0	1	132.02	303.75	0	37	96	177	93296	
HoldTime	0	1	6.32	34.76	0	0	0	0	1284	
WrapUpTime	0	1	141.40	1034.73	0	5	25	159	112600	

Variable type: POSIXct

skim_variable	n_missing	complete_rate	min	max	median	n_unique
StartTime	0	1	2022-03-21 14:13:52	2024-10-31 23:57:27	2023-07-06 22:03:34	274871
EndTime	0	1	2022-03-21 14:57:26	2024-10-31 23:59:40	2023-07-06 22:08:16	274981

Observation

The dataset has 8 features and 275,655 records.

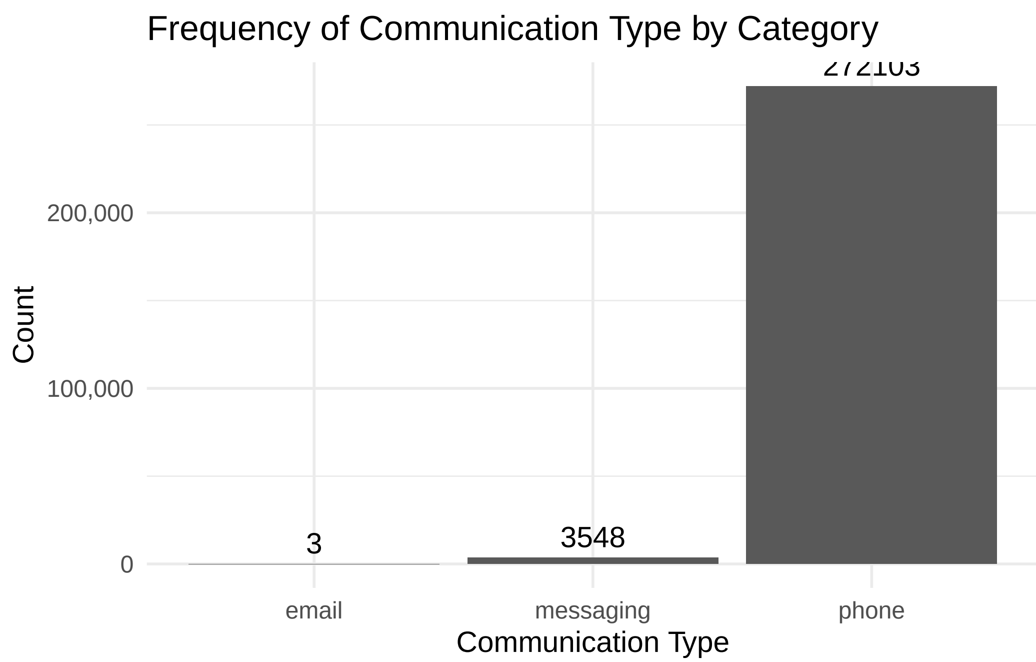
Two of the features are datetime information.

Two features are categorical. Four features are continuous.

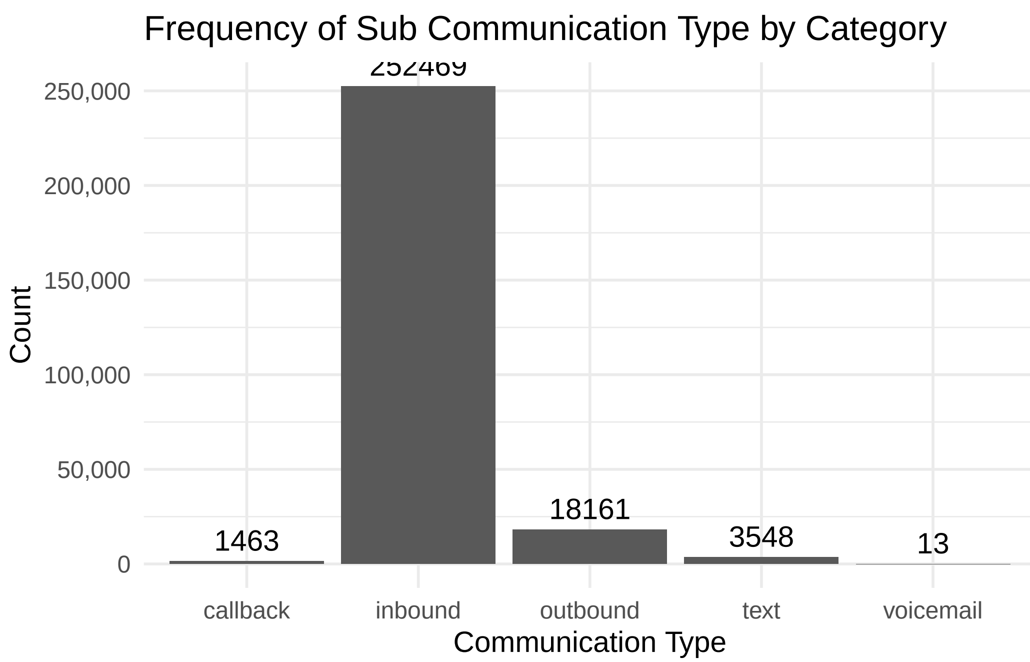
Out of all the records, only one row is missing data.

Categorical Variables

```r
df_clean %>%
  count(CommunicationType) %>%
  ggplot(aes(x = CommunicationType, y = n)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = n), vjust = -0.5) +
  labs(
    title = "Frequency of Communication Type by Category",
    x = "Communication Type",
    y = "Count"
  ) +
  scale_y_continuous(labels = comma) +
  theme_minimal()
```




```r
df_clean %>%
  count(SubCommunicationType) %>%
  ggplot(aes(x = SubCommunicationType, y = n)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = n), vjust = -0.5) +
  labs(
    title = "Frequency of Sub Communication Type by Category",
    x = "Communication Type",
    y = "Count"
  ) +
  scale_y_continuous(labels = comma) +
  theme_minimal()
```



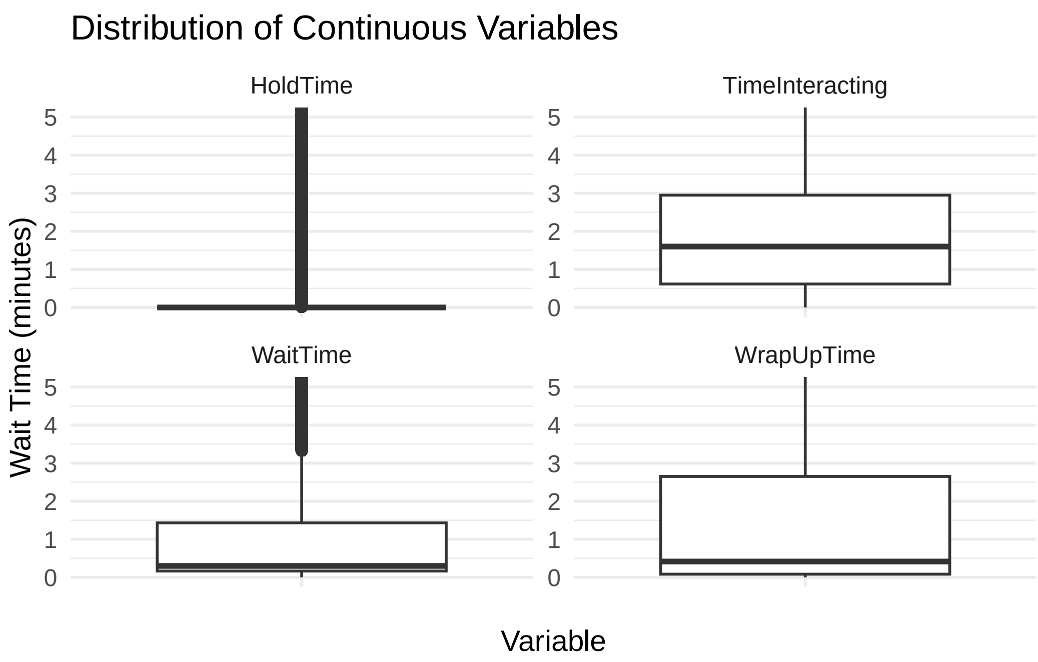
Observations

Most communication type is by phone. Dataset contains mostly inbound communication.

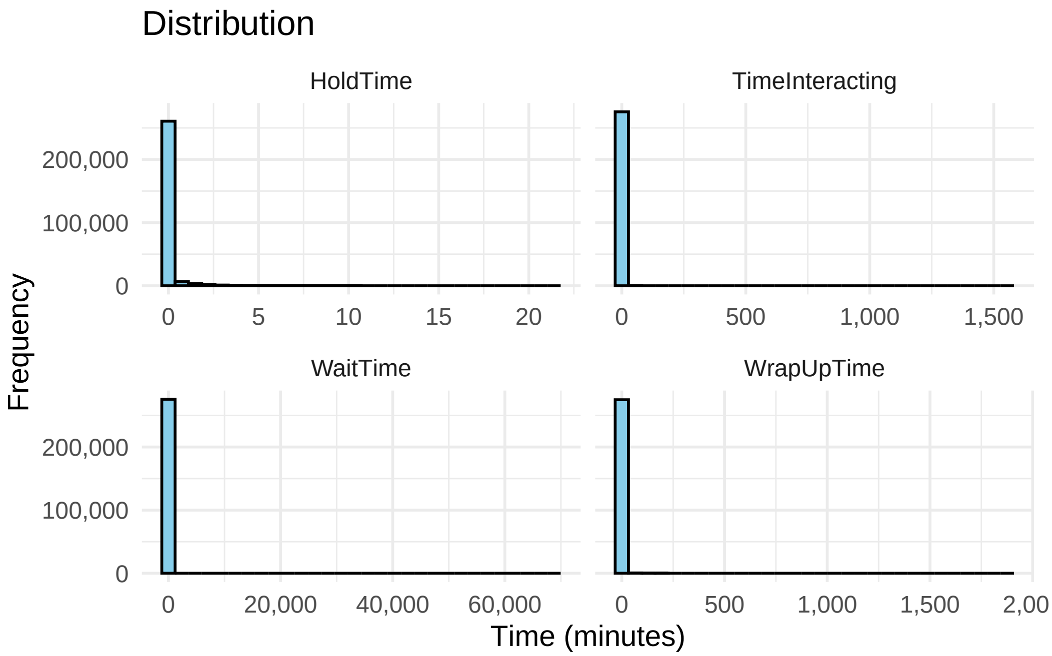
Continuous Variables

```
# reshape data to long format
df_long <- df_clean %>%
  pivot_longer(
    cols = c(WaitTime, TimeInteracting, HoldTime, WrapUpTime),
    names_to = "Variable", values_to = "Value"
  ) %>%
  mutate(Value = Value / 60)

# create box plots
ggplot(df_long, aes(x = "", y = Value)) +
  geom_boxplot() +
  facet_wrap(~ Variable, scales = "free_y") +
  coord_cartesian(ylim = c(0, 5)) +
  labs(
```

```
    title = "Distribution of Continuous Variables",
    x = "Variable",
    y = "Wait Time (minutes)"
  ) +
  scale_y_continuous( labels = comma ) +
  theme_minimal()
```




```
ggplot(df_long, aes(x = Value)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  facet_wrap(~ Variable, scales = "free_x") +
  labs(
    title = "Distribution",
    x = "Time (minutes)",
    y = "Frequency"
  ) +
  scale_y_continuous(labels = comma) +
  scale_x_continuous(labels = comma) +
  theme_minimal()
```




```
continuous_var <- c("WaitTime", "TimeInteracting", "HoldTime", "WrapUpTime")

skim(df_clean[, continuous_var]) %>%
  select(skim_variable, numeric.mean, numeric.sd, numeric.p0, numeric.p25, numeric.p50, nume
  gt() %>%
  fmt_number(
    columns = everything(),
    decimals = 2
  ) %>%
  cols_label(
    skim_variable = "Variable",
    numeric.mean = "Mean",
    numeric.sd = "SD",
    numeric.p0 = "Min",
    numeric.p25 = "25%",
    numeric.p50 = "50%",
    numeric.p75 = "75%",
    numeric.p100 = "Max"
  ) %>%
  tab_header(
    title = "Statistics for Continuous Variables (Minutes)"
  ) %>%
  tab_style(
```

Statistics for Continuous Variables (Minutes)

Variable	Mean	SD	Min	25%	50%	75%	Max
WaitTime	118.69	7,975.21	0.00	10.00	18.00	86.00	4,126,179.00
TimeInteracting	132.02	303.75	0.00	37.00	96.00	177.00	93,296.00
HoldTime	6.32	34.76	0.00	0.00	0.00	0.00	1,284.00
WrapUpTime	141.40	1,034.73	0.00	5.00	25.00	159.00	112,600.00

```
    style = cell_text(weight = "bold"),
    locations = cells_column_labels(everything())
  ) %>%
  cols_align(
    align = "center",
    columns = c(numeric.mean, numeric.sd, numeric.p0, numeric.p25, numeric.p50, numeric.p75,
  )
```

Observations

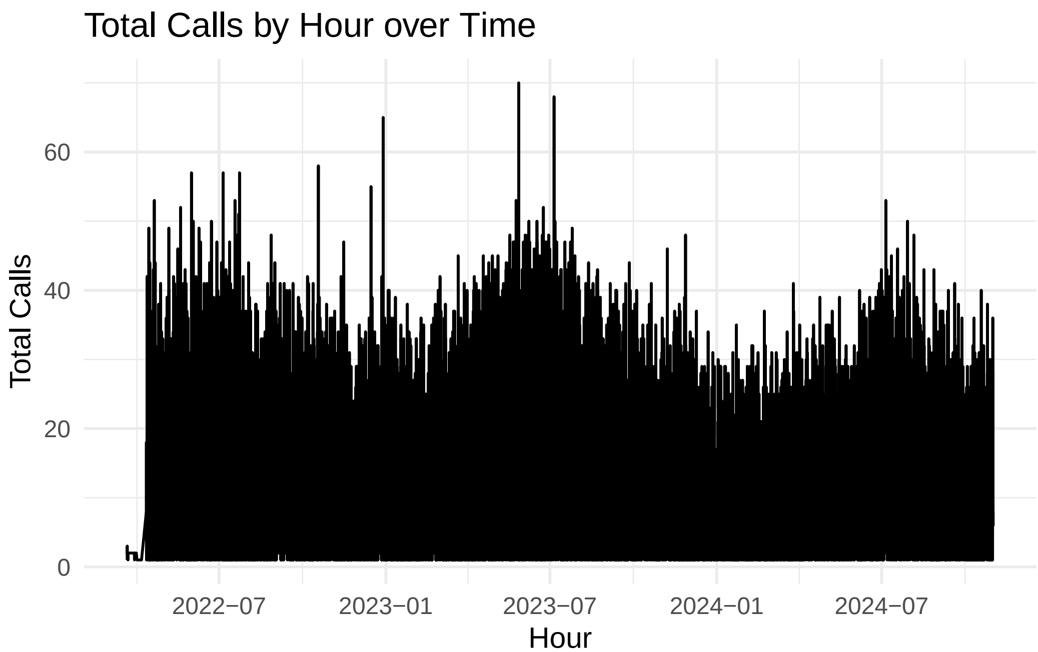
Each of the continuous variables have relatively low means and they also contain extremely high outliers.

Time Series

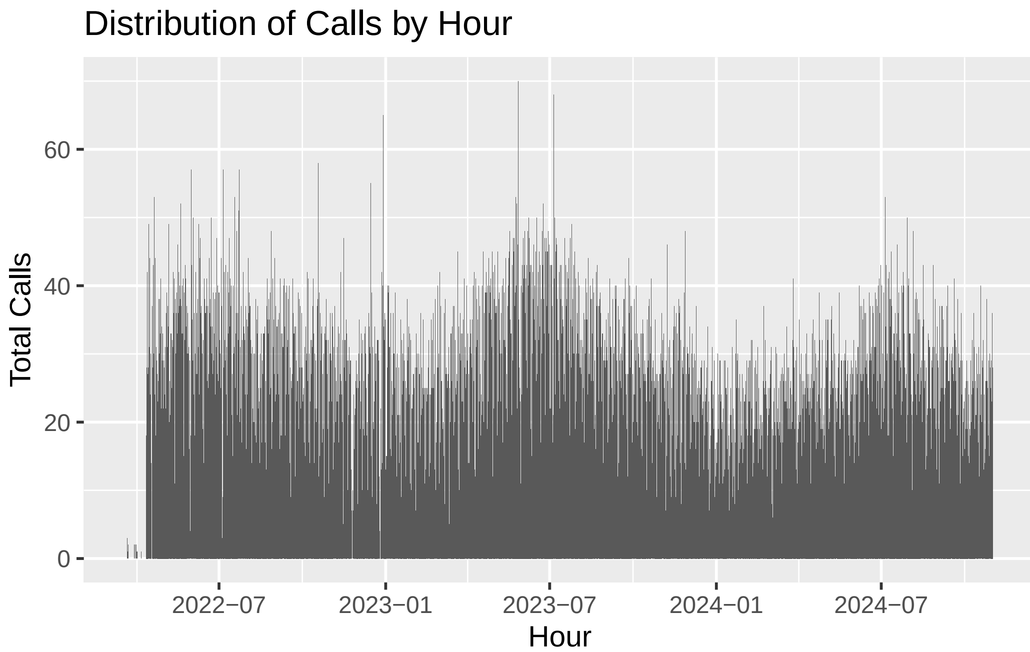
Hourly

```
# aggregate to hourly
df_hourly <- df_clean %>%
  mutate(hour = floor_date(StartTime, "hour")) %>%
  group_by(hour) %>%
  summarise(total_calls = n())

# convert to tsibble
df_hourly_ts <- df_hourly %>%
  as_tsibble(index = hour)

# plot using autoplot
autoplot(df_hourly_ts, total_calls) +
  labs(
```

```
    title = "Total Calls by Hour over Time",
    x = "Hour",
    y = "Total Calls"
  ) +
  theme_minimal()
```



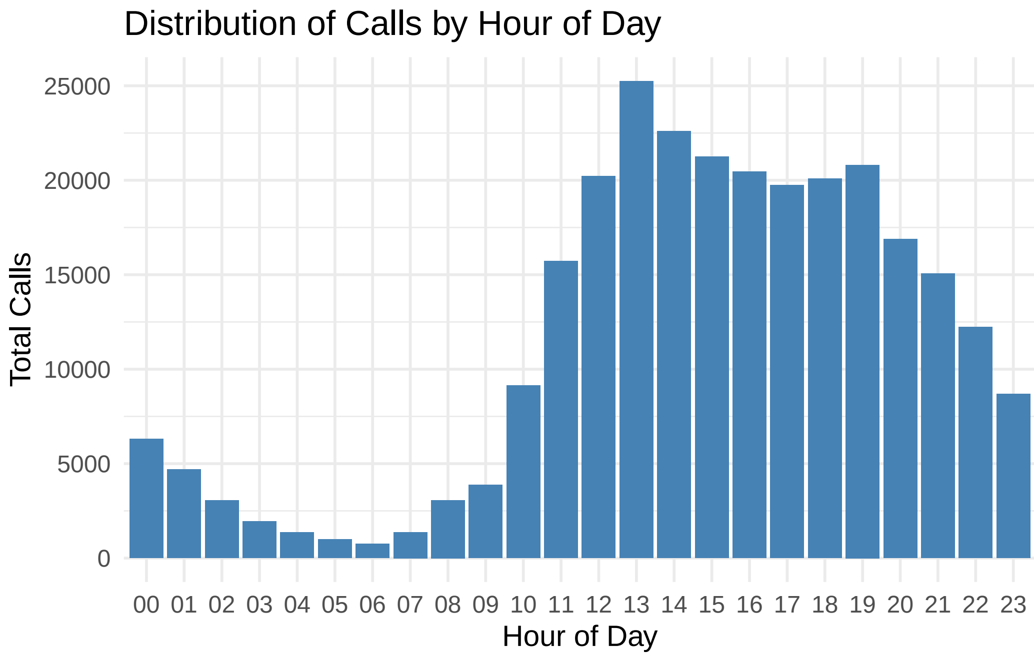

```
# distribution of calls by hour
ggplot(df_hourly, aes(x = hour, y = total_calls)) +
  geom_bar(stat = "identity") +
  labs(
    title = "Distribution of Calls by Hour",
    x = "Hour",
    y = "Total Calls"
  )
```



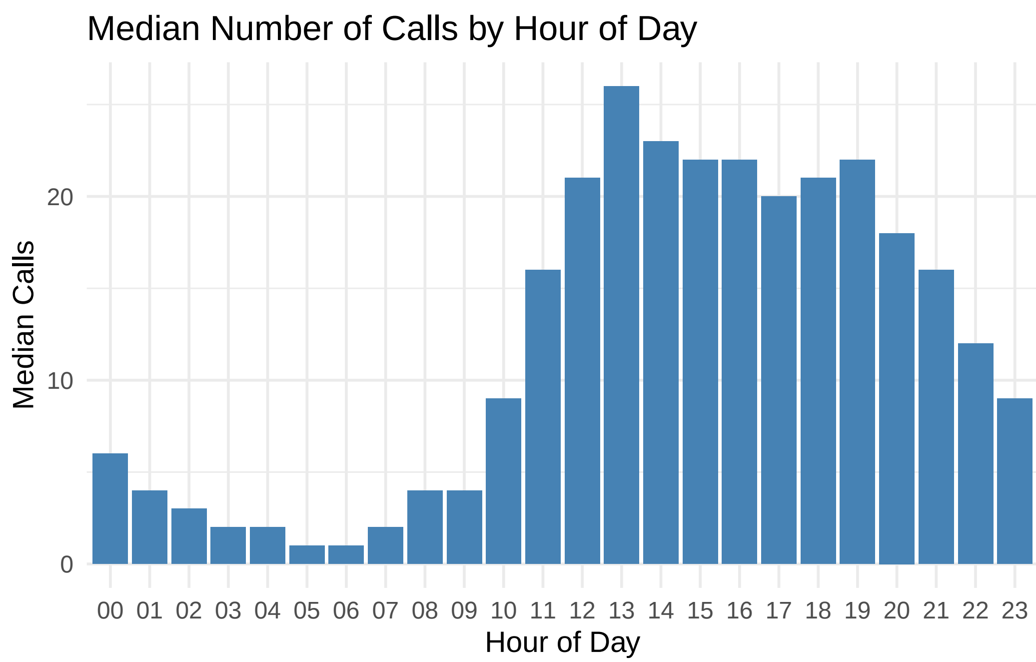
Observations

Time series at the hourly granularity is too noisy and visually cluttered. Better information could be gathered at a lower frequency: daily, weekly, or monthly.

```
# aggregate to hour of day
df_hour_of_day <- df_clean %>%
  mutate(hour_of_day = format(StartTime, "%H")) %>%
  group_by(hour_of_day) %>%
  summarise(total_calls = n())

# plot histogram of total counts
ggplot(df_hour_of_day, aes(x = hour_of_day, y = total_calls)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(
    title = "Distribution of Calls by Hour of Day",
    x = "Hour of Day",
    y = "Total Calls"
  ) +
  theme_minimal()
```




```r
# create df for median calls per hour
df_daily_hourly_calls <- df_clean %>%
  mutate(date = as.Date(StartTime),
         hour_of_day = format(StartTime, "%H")) %>%
  group_by(date, hour_of_day) %>%
  summarise(total_calls = n(), .groups = 'drop')

# create df to calc median calls/hour
df_hourly_median <- df_daily_hourly_calls %>%
  group_by(hour_of_day) %>%
  summarise(median_calls = median(total_calls), .groups = 'drop')

# plot median calls by hour
ggplot(df_hourly_median, aes(x = hour_of_day, y = median_calls)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(
    title = "Median Number of Calls by Hour of Day",
    x = "Hour of Day",
    y = "Median Calls"
  ) +
  theme_minimal()
```



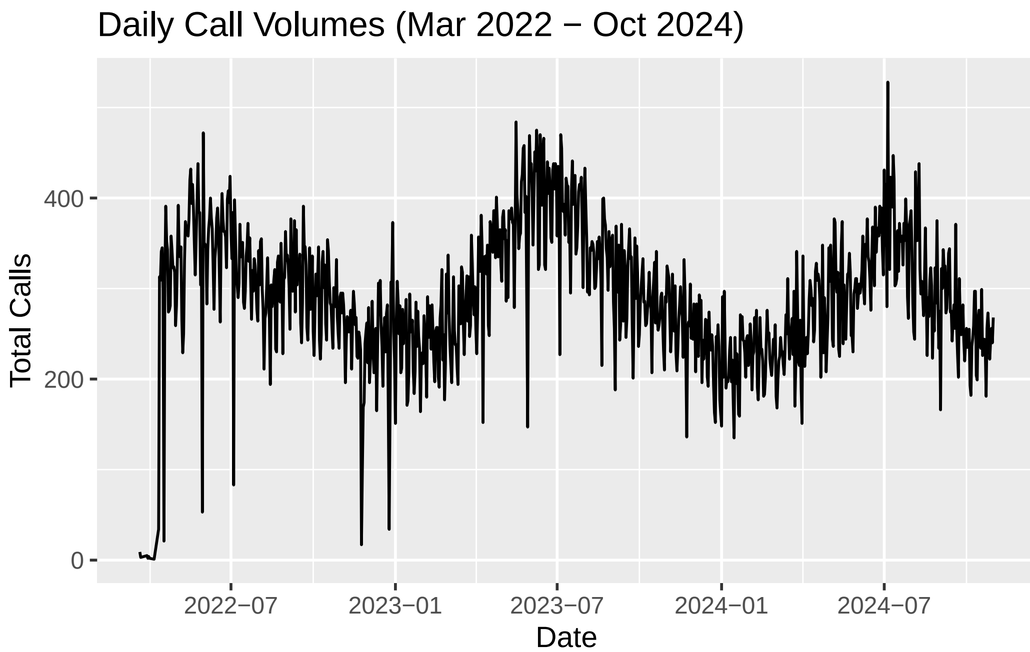
Observations

Call volumes are greater than 15 calls/hour from 11am - 9pm.

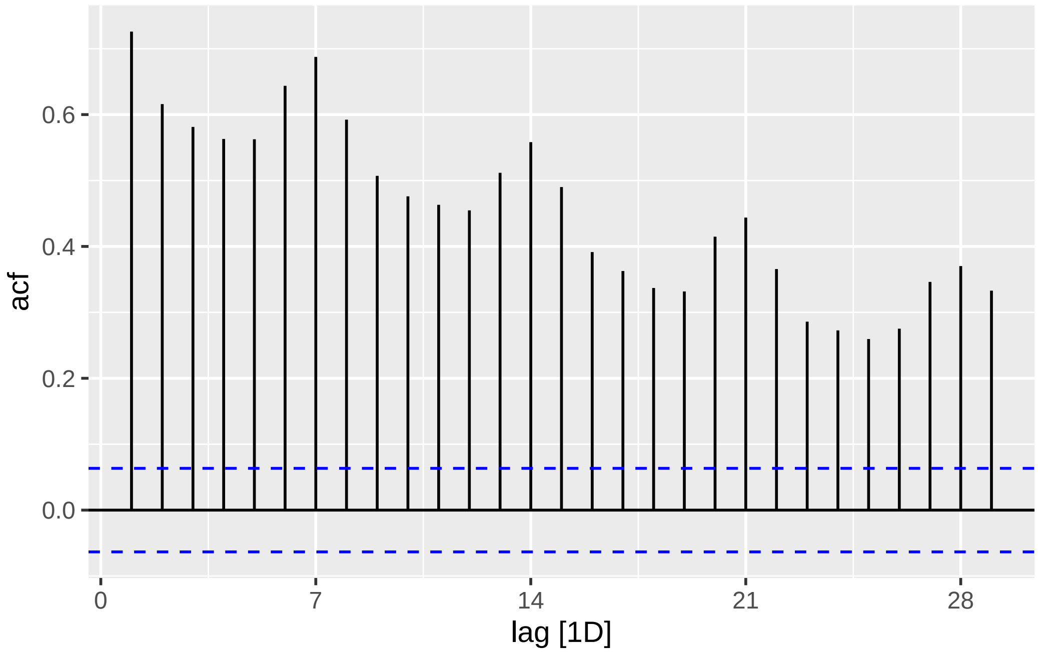
Daily

```
# aggregate to daily
df_daily_calls <- df_clean %>%
  mutate(date = as.Date(StartTime)) %>%
  group_by(date) %>%
  summarise(total_calls = n(), .groups = 'drop')

# convert to tsibble
df_daily_calls_ts <- df_daily_calls %>%
  as_tsibble(index = date)

# plot time series of daily call vols
df_daily_calls_ts %>%
  autoplot(total_calls) +
  labs(
    title = "Daily Call Volumes (Mar 2022 - Oct 2024)",
```

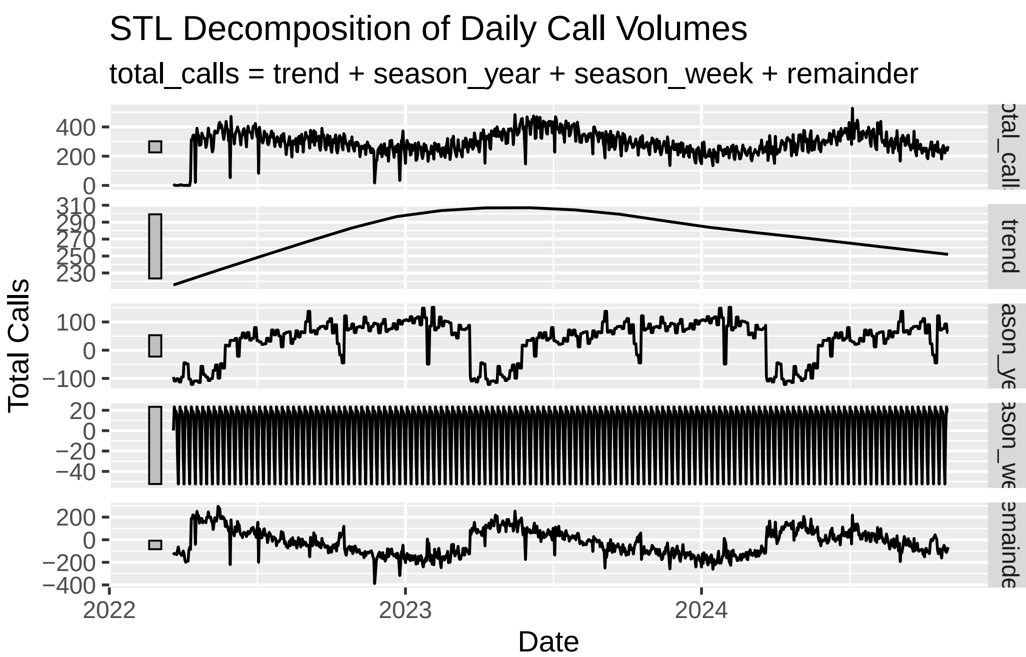
```
    y = "Total Calls",
    x = "Date"
  )
```




```
# autocorrelation
df_daily_calls_ts %>%
  fill_gaps(total_calls = 0) %>%
  ACF(total_calls) %>%
  autoplot()
```




```
# decomp of daily total call volume
decomp_daily_calls <- df_daily_calls_ts %>%
  fill_gaps(total_calls = 0) %>%
  model(stl = STL(total_calls ~ season(window = "periodic")))

# extract and view decomp components
components_calls_daily <- decomp_daily_calls %>%
  components()

# plot decomp
components_calls_daily %>%
  autoplot() +
  labs(
    title = "STL Decomposition of Daily Call Volumes",
    y = "Total Calls",
    x = "Date"
  )
```



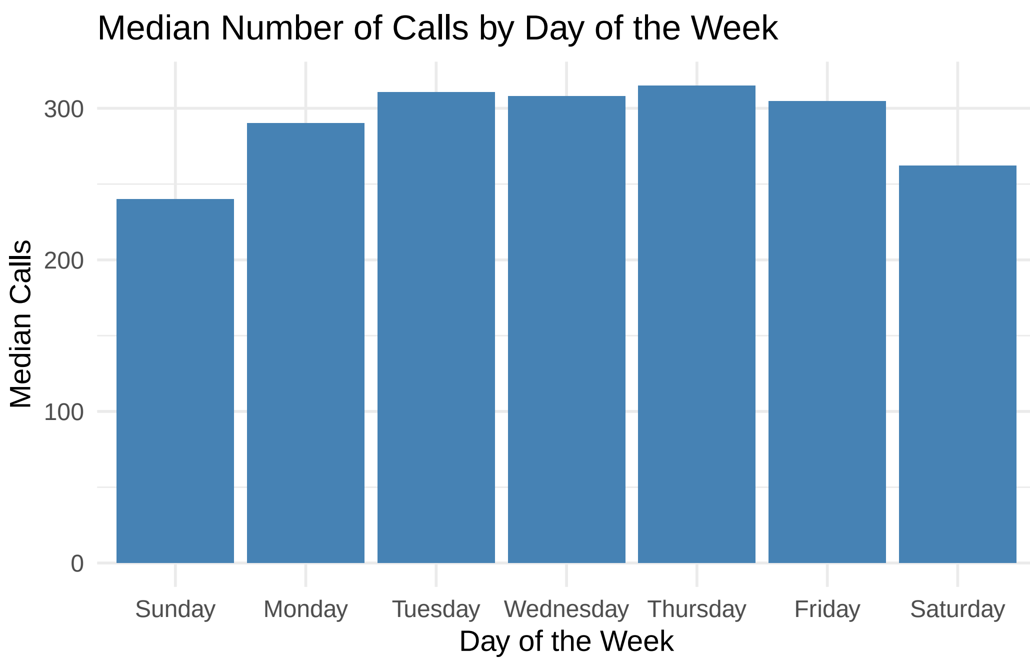
Observations

These plots suggest a seasonal pattern in call volumes.

The ACF plot suggests strong autocorrelation with weekly seasonality as seen in the spikes every 7 days.

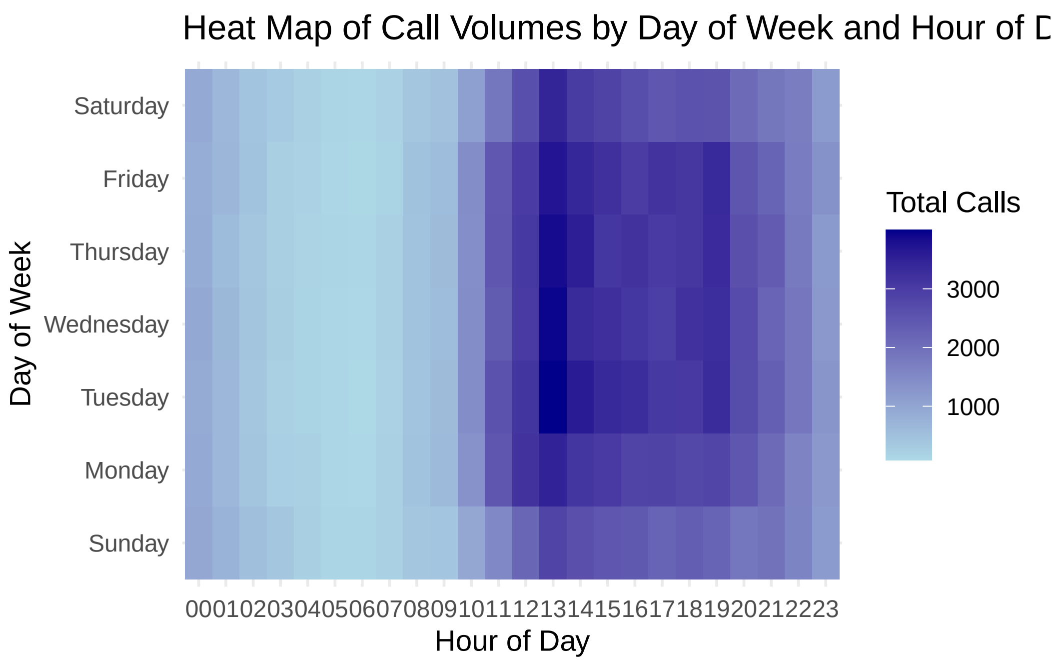
```
# create var for day of week order
days_of_week_order = c("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Sa

# aggregate median calls by date
df_median_calls_by_day <- df_daily_calls %>%
  mutate(day_of_week = weekdays(date)) %>%
  group_by(day_of_week) %>%
  summarise(median_calls = median(total_calls), .groups = 'drop')

# factor to ensure proper day of week order
df_median_calls_by_day$day_of_week <- factor(
  df_median_calls_by_day$day_of_week,
  levels = days_of_week_order
)

#
```

```r
df_median_calls_by_day %>%
  ggplot(aes(x = day_of_week, y = median_calls)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(
    title = "Median Number of Calls by Day of the Week",
    x = "Day of the Week",
    y = "Median Calls"
  ) +
  theme_minimal()
```




```r
# df group by day of week and hour of day
df_day_hour_calls <- df_clean %>%
  mutate(day_of_week = weekdays(as.Date(StartTime)),
         hour_of_day = format(StartTime, "%H")) %>%
  group_by(day_of_week, hour_of_day) %>%
  summarise(total_calls = n(), .groups = 'drop')

# factor for day of week order
df_day_hour_calls$day_of_week <- factor(
  df_day_hour_calls$day_of_week,
  levels = days_of_week_order
)
```

```r
# plot heatmap
df_day_hour_calls %>%
  ggplot(aes(x = hour_of_day, y = day_of_week, fill = total_calls)) +
  geom_tile() +
  scale_fill_gradient(low = "lightblue", high = "darkblue") +
  labs(
    title = "Heat Map of Call Volumes by Day of Week and Hour of Day",
    x = "Hour of Day",
    y = "Day of Week",
    fill = "Total Calls"
  ) +
  theme_minimal()
```



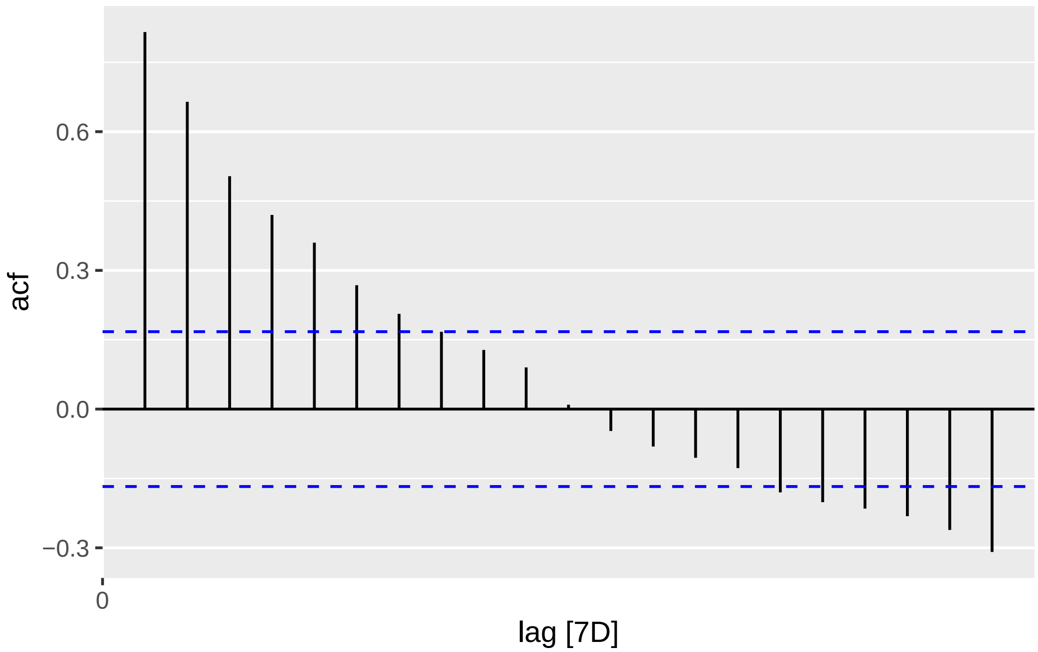
Observations

Call volumes are highest, exceeding 300 calls per day from Tuesday through Friday and mostly concentrated around 1300 hrs.

Weekly - Total Call Volume

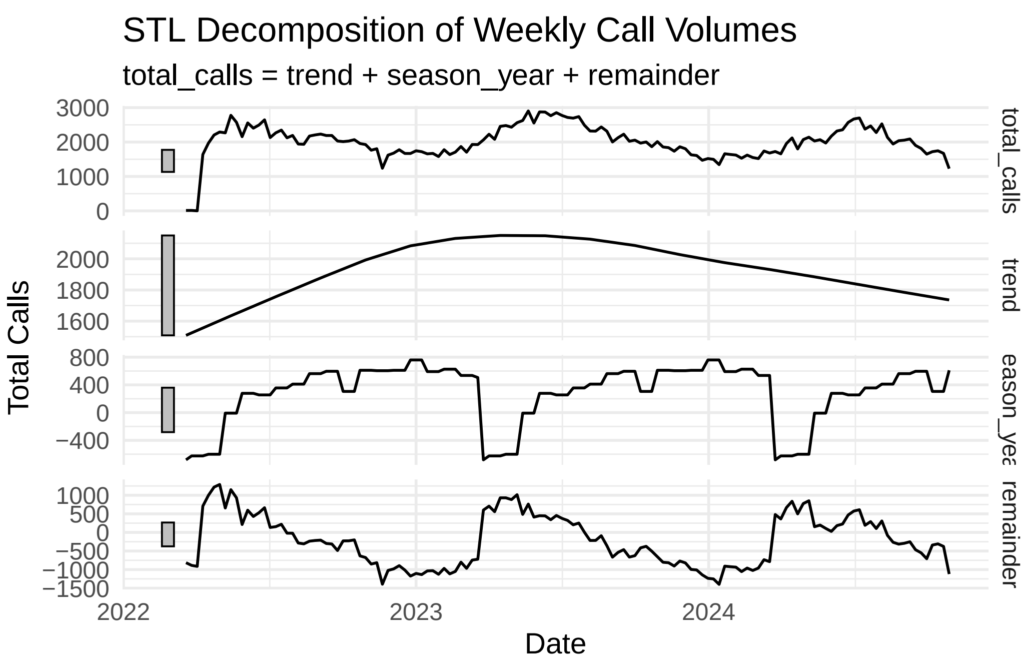
```
df_weekly <- df_clean %>%
  mutate(week = floor_date(as.Date(StartTime), "week")) %>%  # Round StartTime to the beginni
  group_by(week) %>%
  summarise(total_calls = n()) %>%  # Count the number of rows (calls) per week
  ungroup() %>%
  # Fill in missing weeks with 0 calls
  complete(week = seq.Date(min(week), max(week), by = "week"), fill = list(total_calls = 0))

df_weekly_ts <- df_weekly %>%
  as_tsibble(index = week)

# plot chart
df_weekly_ts %>%
  autoplot(total_calls) +
  labs(
    title = "Weekly Call Volumes (Mar 2022 - Oct 2024)",
    y = "Total Calls",
    x = "Date"
  ) +
  theme_minimal()
```

```r
# autocorrelation
df_weekly_ts %>%
  ACF(total_calls) %>%
  autoplot()
```




```r
# decomp of weekly total call volume
decomp_calls <- df_weekly_ts %>%
  fill_gaps(total_calls = 0) %>%
  model(stl = STL(total_calls ~ season(window = "periodic")))

# extract and view decomp components
components_calls <- decomp_calls %>%
  components()

# plot decomp
components_calls %>%
  autoplot() +
  labs(
    title = "STL Decomposition of Weekly Call Volumes",
    y = "Total Calls",
    x = "Date"
  ) +
```

```
  theme_minimal()
```



Observations

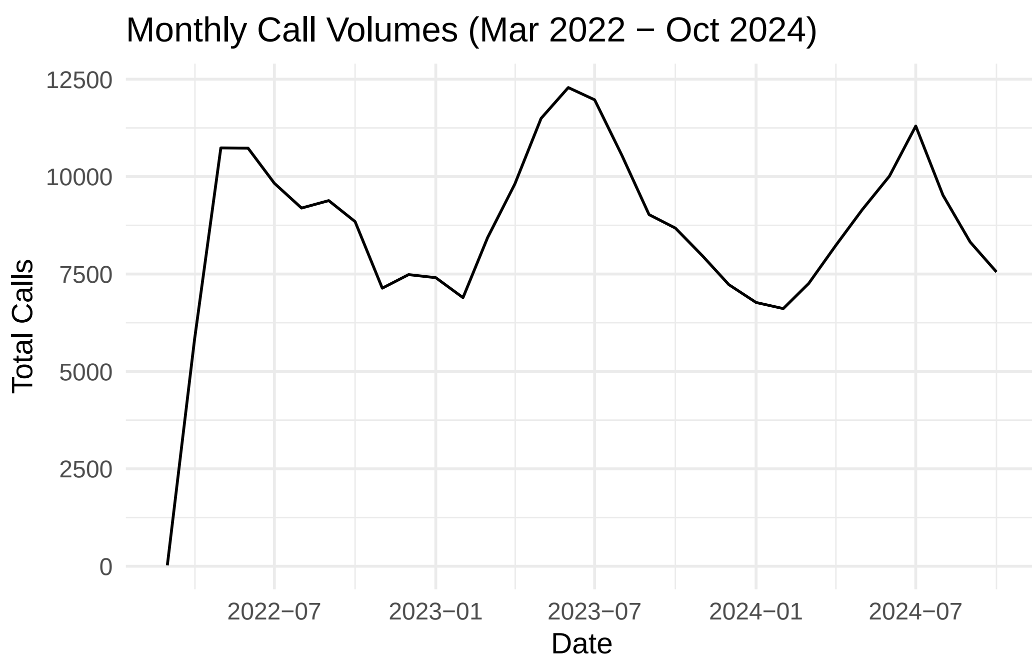
These chart shows that weekly call volumes may be seasonal. The ACF chart suggests there is significant positive autocorrelation.

The Remainder chart does not appear to be random. This would need to be explored further to determine if there are any uncaptured season trends.

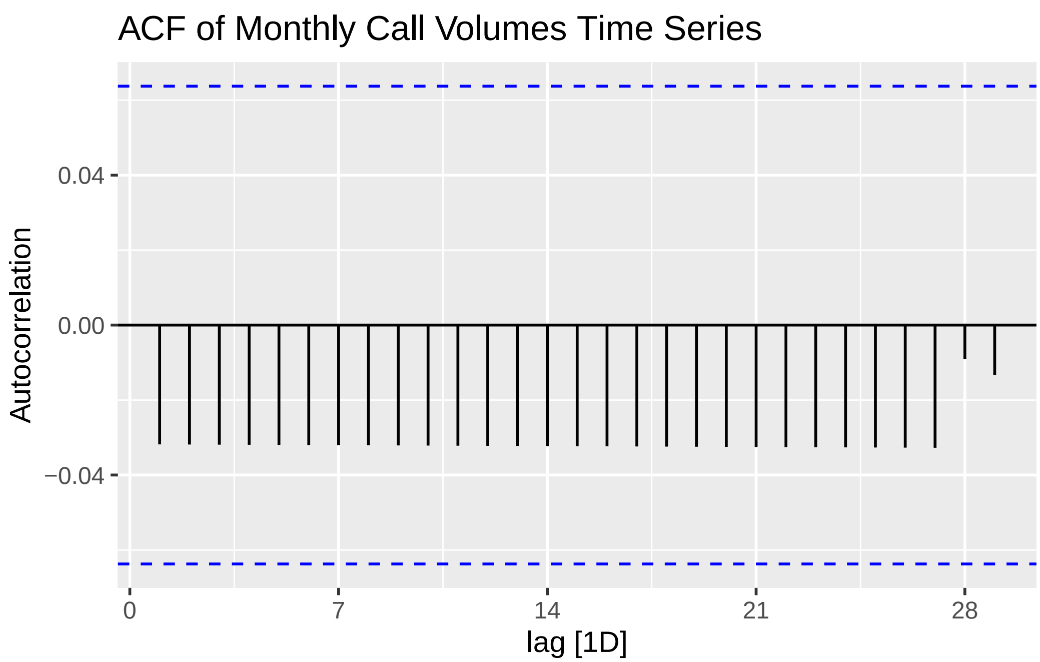
Monthly

```
# aggregate by month
df_monthly_calls <- df_clean %>%
  mutate(month = floor_date(as.Date(StartTime), "month")) %>%
  group_by(month) %>%
  summarise(total_calls = n(), .groups = 'drop')

# convert to tsibble
df_monthly_calls_ts <- df_monthly_calls %>%
  as_tsibble(index = month)
```

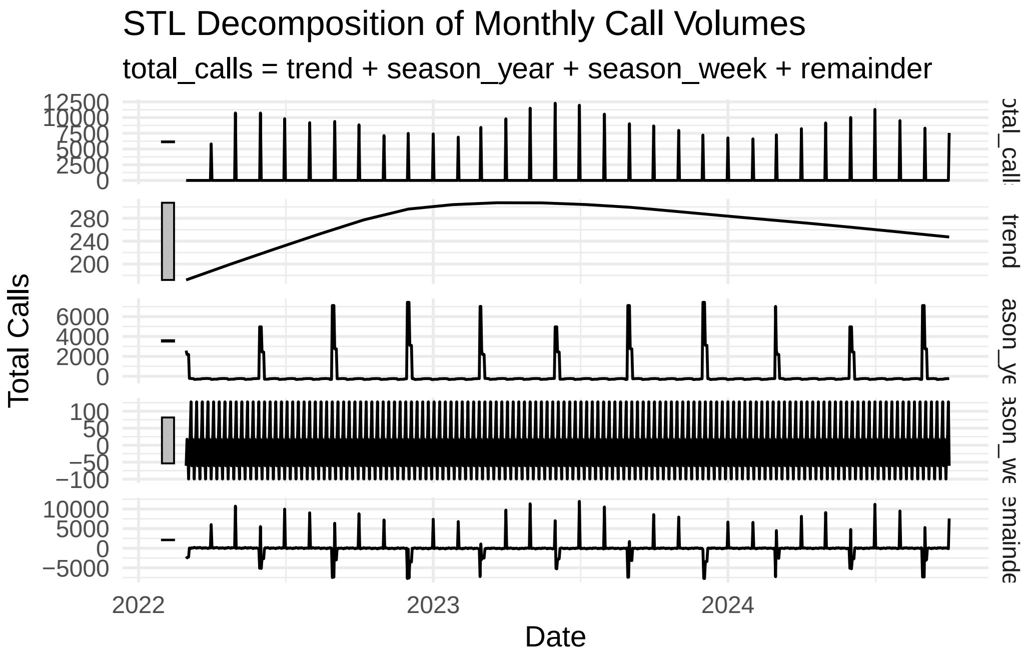
```r
# plot
df_monthly_calls_ts %>%
  autoplot(total_calls) +
  labs(
    title = "Monthly Call Volumes (Mar 2022 - Oct 2024)",
    y = "Total Calls",
    x = "Date"
  ) +
  theme_minimal()
```




```r
# plot autocorrelation
df_monthly_calls_ts %>%
  fill_gaps(total_calls = 0) %>%
  ACF(total_calls) %>%
  autoplot() +
  labs(
    title = "ACF of Monthly Call Volumes Time Series",
    y = "Autocorrelation"
  )
```



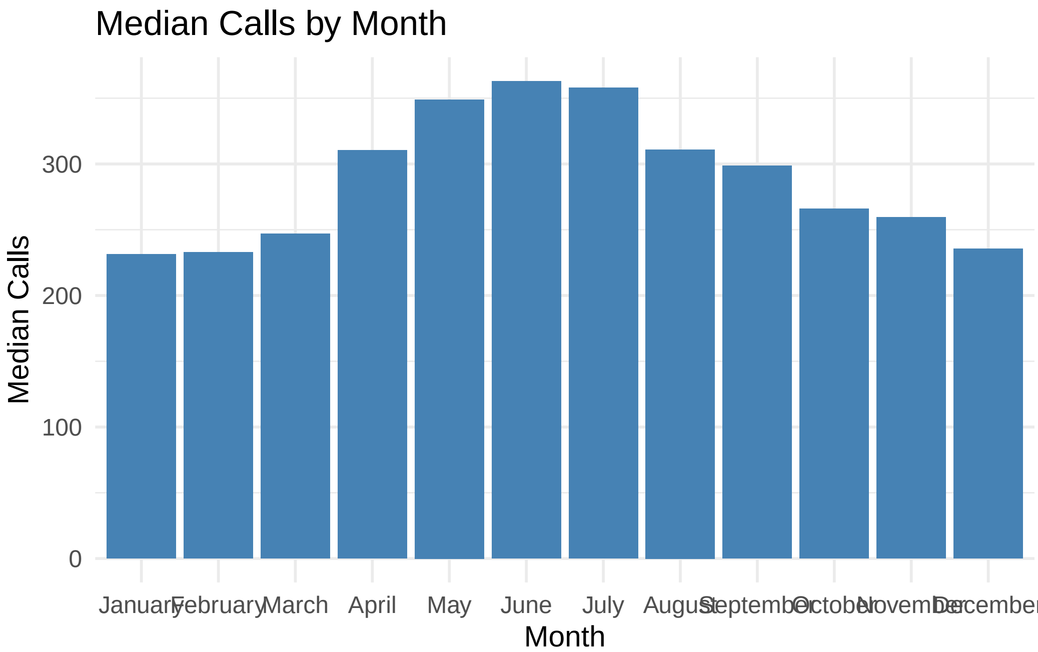

```r
# decomp of weekly total call volume
decomp_calls_monthly <- df_monthly_calls_ts %>%
  fill_gaps(total_calls = 0) %>%
  model(stl = STL(total_calls ~ season(window = "periodic")))

# extract and view decomp components
components_calls_monthly <- decomp_calls_monthly %>%
  components()

# plot decomp
components_calls_monthly %>%
  autoplot() +
  labs(
    title = "STL Decomposition of Monthly Call Volumes",
    y = "Total Calls",
    x = "Date"
  ) +
  theme_minimal()
```



Observations

Monthly call volumes appear to have seasonal pattern. The ACF chart shows that there are no lags outside of the significance threshold indicating low autocorrelation.

```
# aggregate by month
df_median_calls_by_month <- df_daily_calls %>%
  mutate(month = month(date, label = TRUE, abbr = FALSE)) %>%
  group_by(month) %>%
  summarise(median_calls = median(total_calls), .groups = "drop")

# plot
df_median_calls_by_month %>%
  ggplot(aes(x = month, y = median_calls)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(
    title = "Median Calls by Month",
    x = "Month",
    y = "Median Calls"
  ) + theme_minimal()
```



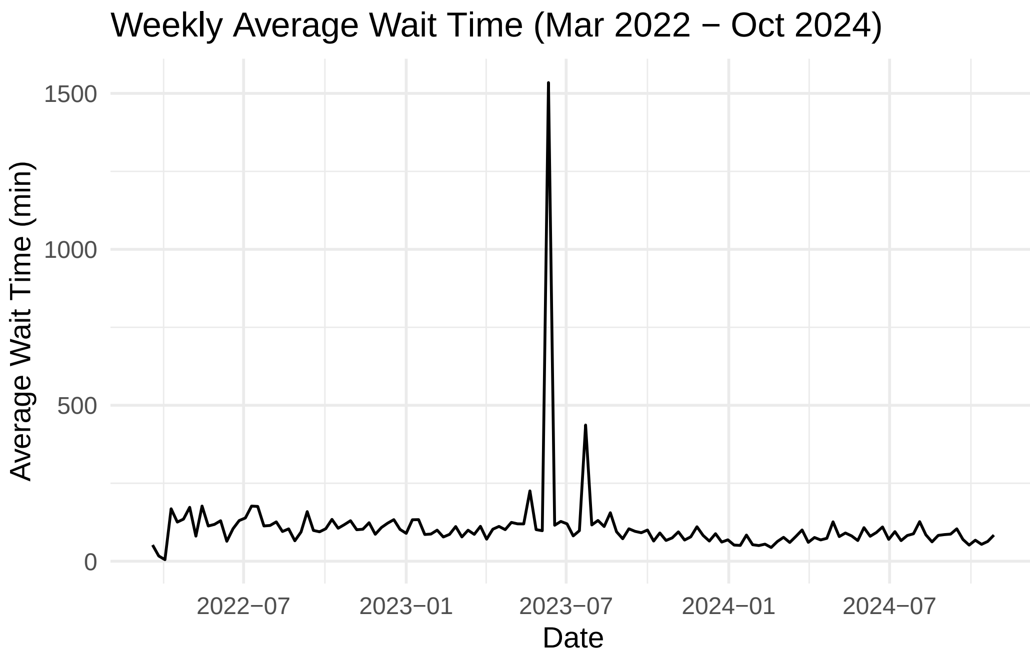
Observations

Median call volumes > 300 call occur between April - August.

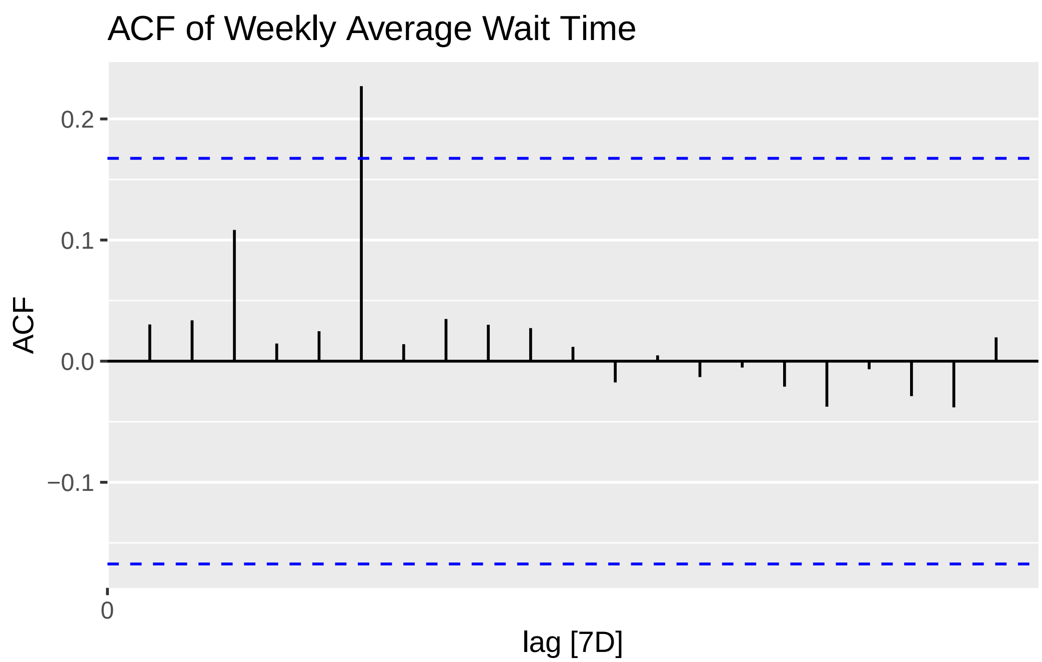
Weekly - Average of WaitTime

```
df_weekly_wait <- df_clean %>%
  mutate(week = floor_date(as.Date(StartTime), "week")) %>%
  group_by(week) %>%
  summarise(avg_wait_time = mean(WaitTime, na.rm = TRUE)) %>%
  ungroup() %>%
  # Fill in missing weeks
  complete(week = seq.Date(min(week), max(week), by = "week"), fill = list(avg_wait_time = 0)

# convert to tsibble
df_weekly_wait_ts <- df_weekly_wait %>%
  as_tsibble(index = week)

# plot chart
df_weekly_wait_ts %>%
  autoplot(avg_wait_time) +
```

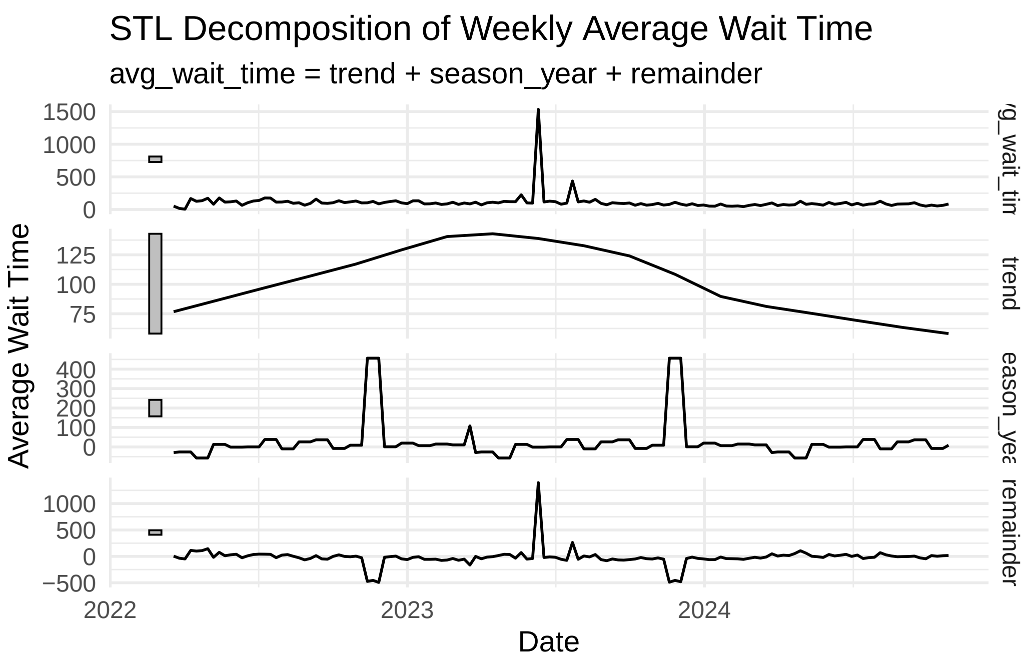
```r
  labs(
    title = "Weekly Average Wait Time (Mar 2022 - Oct 2024)",
    y = "Average Wait Time (min)",
    x = "Date"
  ) +
  theme_minimal()
```




```r
# autocorrelation
df_weekly_wait_ts %>%
  fill_gaps(avg_wait_time = 0) %>%
  ACF(avg_wait_time) %>%
  autoplot() +
  labs(
    title = "ACF of Weekly Average Wait Time",
    y = "ACF"
  )
```




```
# decomposition
decomp_wait <- df_weekly_wait_ts %>%
  fill_gaps() %>%
  mutate(avg_wait_time = if_else(is.na(avg_wait_time), mean(avg_wait_time, na.rm = TRUE), avg
  model(stl = STL(avg_wait_time ~ season(window = "periodic")))

# extract and view decomp components
components_wait <- decomp_wait %>%
  components()

# plot decomp
components_wait %>%
  autoplot() +
  labs(
    title = "STL Decomposition of Weekly Average Wait Time",
    y = "Average Wait Time",
    x = "Date"
  ) +
  theme_minimal()
```



Observations

Based on average wait time, this does not have any seasonality or cyclic elements. Some weeks were missing data and therefore arbitrarily imputed with the mean wait time.

Data Preparation

Step 1: Drop all rows that are not an inbound phone call. 23, 185 rows dropped

```
df_phone <- df %>%
  filter(CommunicationType == "phone") %>%
  filter(SubCommunicationType == "inbound")
  dim(df_phone)
```

```
[1] 252470      8
```

Step 2: Check min/max dates. Final data should be 4/11/22 - 10/31/24 as prior to 4/11/22 was on boarding the phone system and not representative of operations.

```r
max_date <- max(df_phone$StartTime)
min_date <- min(df_phone$StartTime)
print(max_date)
```

```
[1] "2024-10-31 23:57:27 UTC"
```

```r
print(min_date)
```

```
[1] "2022-03-21 16:57:05 UTC"
```

```r
df_date <- df_phone %>%
  filter(StartTime >= as.POSIXct("2022-11-04"))
phone_min <- min(df_date$StartTime)
print(phone_min)
```

```
[1] "2022-11-04 09:39:17 UTC"
```

Step 3: Separate timestamp into date, time, and day of week

```r
df_date <- df_date %>%
  mutate(
    Date = as.Date(StartTime),
    Time = format(StartTime, "%H:%M:%S"),
    Day = weekdays(StartTime)
  )
```

Step 4: Drop unnecessary columns. Since this forecast is focusing on inbound calls, we will drop the details around call length, hold times, etc.

```r
df_columns <- df_date %>%
  select(-EndTime, -CommunicationType, -SubCommunicationType, -WaitTime, -TimeInteracting, -I
```

Step 5: Add weather. Source: https://www.ncei.noaa.gov/access/search/data-search

```r
weather <- read_csv("C:/Users/sasha/OneDrive/Documents/Datasets/weather.csv", col_types = col
   DATE = col_date(format = "%Y-%m-%d"),
  TMAX = col_integer(),
  TMAX_ATTRIBUTES = col_character()
))
head(weather)
```

```
# A tibble: 6 x 3
  DATE        TMAX TMAX_ATTRIBUTES
  <date>     <int> <chr>
1 2024-10-31   217 W
2 2024-10-30   222 D
3 2024-10-29   200 W
4 2024-10-28   217 W
5 2024-10-27   250 W
6 2024-10-26   233 W
```

```r
weather <- weather %>%
  mutate(
    TMAX_CEL = as.numeric(gsub(",", "", TMAX)) / 10
  )
head(weather)
```

```
# A tibble: 6 x 4
  DATE        TMAX TMAX_ATTRIBUTES TMAX_CEL
  <date>     <int> <chr>              <dbl>
1 2024-10-31   217 W                   21.7
2 2024-10-30   222 D                   22.2
3 2024-10-29   200 W                   20
4 2024-10-28   217 W                   21.7
5 2024-10-27   250 W                   25
6 2024-10-26   233 W                   23.3
```

```r
df_weather <- df_columns %>%
  left_join(weather %>% select(DATE, TMAX_ATTRIBUTES, TMAX_CEL), by = c("Date" = "DATE"))
```

```r
na_count <- sapply(df_weather, function(x) sum(is.na(x)))
print(na_count)
```

```
      StartTime            Date            Time             Day TMAX_ATTRIBUTES
              0               0               0               0               0
       TMAX_CEL
              0
```

Step 6: Add shift information

```
df_prepped <- df_weather %>%
  mutate(
    four_shift = case_when(
      Time >= "00:00" & Time < "04:00" ~ 1,
      Time >= "04:00" & Time < "08:00" ~ 2,
      Time >= "08:00" & Time < "12:00" ~ 3,
      Time >= "12:00" & Time < "16:00" ~ 4,
      Time >= "16:00" & Time < "20:00" ~ 5,
      Time >= "20:00" & Time <= "23:59" ~ 6
    )
  )
```

```
df_prepped <- df_prepped %>%
  mutate(
    six_shift = case_when(
      Time >= "00:00" & Time < "06:00" ~ 1,
      Time >= "06:00" & Time < "12:00" ~ 2,
      Time >= "12:00" & Time < "18:00" ~ 3,
      Time >= "18:00" & Time <= "23:59" ~ 4
    )
  )
```

```
write_csv(df_prepped, "C:/Users/sasha/OneDrive/Documents/Datasets/calls_prepped.csv")
```