Exploring Street-Level Temperature Variations in New York City: A Hyperlocal Analysis

Le' Sean Roberts (NYC Office of Management and Budget college Aide)

Title: Exploring Street-Level Temperature Variations in New York City: A Hyperlocal Analysis

Summary

This study delves into the analysis of hyperlocal temperature data collected as part of the Cool Neighborhoods Initiative in New York City. The dataset encompasses hourly average temperatures across approximately 475 locations during the summers of 2018 and 2019, aiming to understand street-level temperature variations. Through exploratory data analysis (EDA), data visualization, and machine learning techniques, this research seeks to uncover patterns, trends, and predictive models to enhance our understanding of urban temperature dynamics. The findings offer insights into temperature disparities across neighborhoods and provide valuable information for urban planning and public health initiatives.

Introduction

Urban heat islands pose significant challenges for cities, exacerbating heat-related health risks and impacting overall urban resilience. As cities continue to grapple with rising temperatures, understanding hyperlocal temperature variations becomes crucial for implementing effective mitigation strategies. In this context, the Cool Neighborhoods Initiative, a collaborative effort between the NYC Parks Department, Mayor's Office of Resilience, and NYC Department of Health and Mental Hygiene, collected street-level temperature data across select neighborhoods in New York City during the summers of 2018 and 2019. This dataset offers a unique opportunity to analyze and explore temperature dynamics at a granular level. Through this study, we aim to conduct a comprehensive analysis of the hyperlocal temperature dataset, employing EDA, data visualization, and machine learning methods to uncover insights into temperature variations and develop predictive models for future temperature monitoring and urban planning efforts.

Data Cleaning/Wrangling

```
library(readr)
Hyperlocal_Temperature_Monitoring_20240308 <-
read_csv("C:/Users/verlene/Downloads/Hyperlocal_Temperature_Monitoring_202403
08.csv")
```

You can add options to executable code like this

```
library(tidyverse)
glimpse(Hyperlocal_Temperature_Monitoring_20240308)
```

```
Rows: 2,097,150
Columns: 10
$ Sensor.ID    <chr> "Bk-BR_01", "Bk-BR_01", "Bk-BR_01", "Bk-BR_01", "Bk-
BR_01…
$ AirTemp      <dbl> 71.18900, 70.24333, 69.39267, 68.26317, 67.11400,
65.9655…
$ Day          <chr> "06/15/2018", "06/15/2018", "06/15/2018", "06/15/2018",
"…
$ Hour         <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17…
$ Latitude     <dbl> 40.66621, 40.66621, 40.66621, 40.66621, 40.66621,
40.6662…
$ Longitude    <dbl> -73.91691, -73.91691, -73.91691, -73.91691, -73.91691, -
7…
$ Year         <dbl> 2018, 2018, 2018, 2018, 2018, 2018, 2018, 2018, 2018,
201…
$ Install.Type <chr> "Street Tree", "Street Tree", "Street Tree", "Street
Tree…
$ Borough      <chr> "Brooklyn", "Brooklyn", "Brooklyn", "Brooklyn",
"Brooklyn…
$ ntacode      <chr> "BK81", "BK81", "BK81", "BK81", "BK81", "BK81", "BK81",
"…
```

Extracting columns of variables of interest concerning data wrangling, exploratory data, data visualization and machine learning techniques.

```
library(tidyverse)
data <- Hyperlocal_Temperature_Monitoring_20240308 |>
  select(AirTemp, Day, Hour, Latitude, Longitude, Borough, Year) |>
  na.omit()
```

The Boroughs of the city identify geopolitical boundaries for residency, unlike geographical coordinates such as latitute and longitude which concern highly local places, hence such categorical specification (as in boroughs) can prove quite interesting.

```
# Identifying unique instances for the Borough feature.
unique(data$Borough)
```

```
[1] "Brooklyn"  "Bronx"     "Manhattan" "Queens"
```

With “Borough” being a categorical feature, for it to be statistically meaningful its instances need to be converted to numeric representation. As well, to not lose track of the conversation representation columns in dual to be constructed.

```
# Create a factor for the Borough column
data$Borough_factor<- factor(data$Borough)

# Create a new column for numeric codes
```

```r
data$Borough_numeric <- as.integer(data$Borough_factor)

# Print the original and recoded COUNTRY columns
print(data[, c("Borough", "Borough_numeric")])
```

```
# A tibble: 2,085,443 × 2
   Borough  Borough_numeric
   <chr>              <int>
 1 Brooklyn               2
 2 Brooklyn               2
 3 Brooklyn               2
 4 Brooklyn               2
 5 Brooklyn               2
 6 Brooklyn               2
 7 Brooklyn               2
 8 Brooklyn               2
 9 Brooklyn               2
10 Brooklyn               2
# ℹ 2,085,433 more rows
```

```r
# Observing transformed data frame.. Showing first six rows
head(data)
```

```
# A tibble: 6 × 9
  AirTemp Day         Hour Latitude Longitude Borough   Year Borough_factor
    <dbl> <chr>      <dbl>    <dbl>     <dbl> <chr>    <dbl> <fct>
1    71.2 06/15/2018     1     40.7     -73.9 Brooklyn  2018 Brooklyn
2    70.2 06/15/2018     2     40.7     -73.9 Brooklyn  2018 Brooklyn
3    69.4 06/15/2018     3     40.7     -73.9 Brooklyn  2018 Brooklyn
4    68.3 06/15/2018     4     40.7     -73.9 Brooklyn  2018 Brooklyn
5    67.1 06/15/2018     5     40.7     -73.9 Brooklyn  2018 Brooklyn
6    66.0 06/15/2018     6     40.7     -73.9 Brooklyn  2018 Brooklyn
# ℹ 1 more variable: Borough_numeric <int>
```

```r
# Creating direct conversion view. Observing first 6 rows.
good_data = data |>
  select(-Borough_factor)
head(good_data)
```

```
# A tibble: 6 × 8
  AirTemp Day         Hour Latitude Longitude Borough   Year Borough_numeric
    <dbl> <chr>      <dbl>    <dbl>     <dbl> <chr>    <dbl>           <int>
1    71.2 06/15/2018     1     40.7     -73.9 Brooklyn  2018               2
2    70.2 06/15/2018     2     40.7     -73.9 Brooklyn  2018               2
3    69.4 06/15/2018     3     40.7     -73.9 Brooklyn  2018               2
4    68.3 06/15/2018     4     40.7     -73.9 Brooklyn  2018               2
5    67.1 06/15/2018     5     40.7     -73.9 Brooklyn  2018               2
6    66.0 06/15/2018     6     40.7     -73.9 Brooklyn  2018               2
```

```r
# Checking the unique instances for "Borough_numeric".
unique(good_data$Borough_numeric)
```

```
[1] 2 1 3 4
```

```
#Figuring out the "Queens" instance conversion.
Queens_instance <- good_data |>
  filter(Borough == "Queens")
# Showing first 6 rows.
head(Queens_instance)
```

```
# A tibble: 6 × 8
  AirTemp Day         Hour Latitude Longitude Borough  Year Borough_numeric
    <dbl> <chr>      <dbl>    <dbl>     <dbl> <chr>   <dbl>            <int>
1    70.7 06/15/2018     1     40.7     -73.7 Queens   2018                4
2    69.7 06/15/2018     2     40.7     -73.7 Queens   2018                4
3    69.0 06/15/2018     3     40.7     -73.7 Queens   2018                4
4    68.2 06/15/2018     4     40.7     -73.7 Queens   2018                4
5    67.1 06/15/2018     5     40.7     -73.7 Queens   2018                4
6    70.2 06/15/2018     6     40.7     -73.7 Queens   2018                4
```

Hence, the “Queens” instance has been converted to 4.

```
#Figuring out the "Manhattan" instance conversion.
Manhattan_instance <- good_data |>
  filter(Borough == "Manhattan")
# Showing first 6 rows.
head(Manhattan_instance)
```

```
# A tibble: 6 × 8
  AirTemp Day         Hour Latitude Longitude Borough    Year Borough_numeric
    <dbl> <chr>      <dbl>    <dbl>     <dbl> <chr>     <dbl>            <int>
1    70.0 06/15/2018     1     40.8     -73.9 Manhattan  2018                3
2    69.2 06/15/2018     2     40.8     -73.9 Manhattan  2018                3
3    68.1 06/15/2018     3     40.8     -73.9 Manhattan  2018                3
4    66.9 06/15/2018     4     40.8     -73.9 Manhattan  2018                3
5    65.6 06/15/2018     5     40.8     -73.9 Manhattan  2018                3
6    64.6 06/15/2018     6     40.8     -73.9 Manhattan  2018                3
```

Hence, the “Manhattan” instance has been converted to 3.

```
#Figuring out the "Bronx" instance conversion.
Bronx_instance <- good_data |>
  filter(Borough == "Bronx")
# Showing first 6 rows.
head(Bronx_instance)
```

```
# A tibble: 6 × 8
  AirTemp Day         Hour Latitude Longitude Borough  Year Borough_numeric
    <dbl> <chr>      <dbl>    <dbl>     <dbl> <chr>   <dbl>            <int>
1    68.6 06/15/2018     1     40.9     -73.8 Bronx    2018                1
2    67.6 06/15/2018     2     40.9     -73.8 Bronx    2018                1
3    66.9 06/15/2018     3     40.9     -73.8 Bronx    2018                1
4    65.9 06/15/2018     4     40.9     -73.8 Bronx    2018                1
```

```
5    64.7 06/15/2018     5     40.9     -73.8 Bronx    2018                1
6    63.9 06/15/2018     6     40.9     -73.8 Bronx    2018                1
```

Hence, the "Bronx" instance has been converted to 1.

What about Staten Island?

```
#Figuring out the "Staten Island" instance conversion.
Statland_instance <- good_data |>
  filter(Borough == "Staten Island")
# Showing first 6 rows.
head(Statland_instance)

# A tibble: 0 × 8
# ℹ 8 variables: AirTemp <dbl>, Day <chr>, Hour <dbl>, Latitude <dbl>,
#   Longitude <dbl>, Borough <chr>, Year <dbl>, Borough_numeric <int>
```

Data doesn't include Staten Island.

BROOKYLN = 2, QUEENS = 4, MANHATTAN = 3, BRONX = 1.

Exploratory Data Analysis

An exploratory data analysis (EDA) synopsis provides a summary of the main findings and insights obtained from analyzing a dataset. It typically includes key statistics, visualizations, and observations about the data, helping to understand its structure, identify patterns, and uncover relationships between variables.

Summary Statistics for Air Temperature

```
# Dtermining the dimension of the data frame.
dim(good_data)

[1] 2085443       8

# Variables Review
str(good_data)

tibble [2,085,443 × 8] (S3: tbl_df/tbl/data.frame)
 $ AirTemp        : num [1:2085443] 71.2 70.2 69.4 68.3 67.1 ...
 $ Day            : chr [1:2085443] "06/15/2018" "06/15/2018" "06/15/2018"
"06/15/2018" ...
 $ Hour           : num [1:2085443] 1 2 3 4 5 6 7 8 9 10 ...
 $ Latitude       : num [1:2085443] 40.7 40.7 40.7 40.7 40.7 ...
 $ Longitude      : num [1:2085443] -73.9 -73.9 -73.9 -73.9 -73.9 ...
 $ Borough        : chr [1:2085443] "Brooklyn" "Brooklyn" "Brooklyn"
"Brooklyn" ...
 $ Year           : num [1:2085443] 2018 2018 2018 2018 2018 ...
 $ Borough_numeric: int [1:2085443] 2 2 2 2 2 2 2 2 2 2 ...
 - attr(*, "na.action")= 'omit' Named int [1:11707] 132661 132662 132663
132664 132665 132666 132667 132668 132669 132670 ...
  ..- attr(*, "names")= chr [1:11707] "132661" "132662" "132663" "132664" ...
```

Only the summary statistics for temperature will be done because temperature is the only possible natural target (or label or dependent variable) and being a continuous variable.

```
summary(good_data$AirTemp)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  46.03   70.59   75.71   75.98   81.85  114.96 

# Standard Deviation
sd(good_data$AirTemp)

[1] 8.998371
```

Identifying the Mode for a large Data Set

```
# Use table() function to count occurrences of each value
air_temp_table <- table(good_data$AirTemp)

# Find the maximum count (frequency)
max_count <- max(air_temp_table)

# Find values with maximum count (mode)
mode_values <- as.numeric(names(air_temp_table[air_temp_table == max_count]))

# Print the mode(s)
print(mode_values)

[1] 73.58
```

Alternatively:

```
library(DescTools)
# Find the mode(s)
mode_value <- Mode(good_data$AirTemp)

# Print the mode(s)
print(mode_values)

[1] 73.58
```

Skew and Kurtosis for Air Temperature

Only skew and kurtosis measures for temperature will be done because temperature is the only possible natural target (or label or dependent variable) and being a continuous variable.

The topics of skew and kurtosis may arise in a course of Computational Statistics, stemming from the method of moments for point estimation. The method of moments involves equating sample moments with theoretical moments. Well, the skew corresponds to the third moment, while the kurtosis corresponds to the fourth moment.

```
library(moments)
# Determine the skew for Air Temperature.
skewness(good_data$AirTemp)
```

```
[1] -0.04970579
```

```
# Determine the kurtosis for Air Temperature.
kurtosis(good_data$AirTemp)
```

```
[1] 3.109107
```

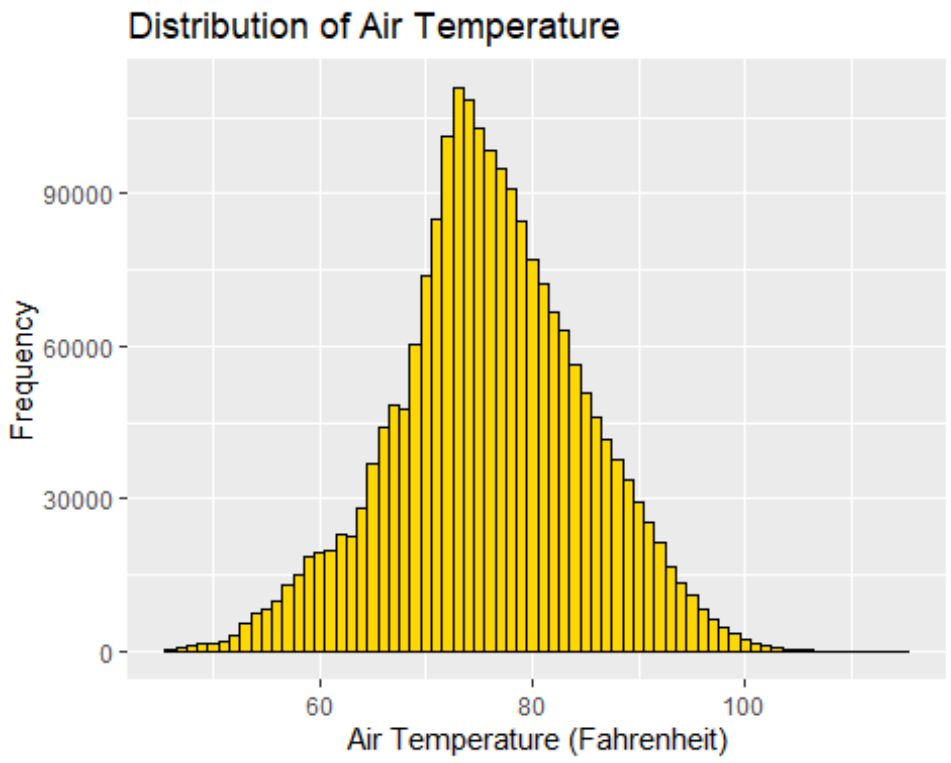
Interpreting the skew. Data distributions that have a skew of 0 are symmetric. Interpreting the kurtosis. For the kurtosis of value 3, such is mesokurtic, or data that's normally distributed; such value serving as a baseline for describing data distributions. Leptokurtic (Kurtosis > 3.0) while Platykurtic (Kurtosis < 3.0). Our result of 3.109107 identifies leptokurtic behavior, but the distribution is close to normal.

Data Visualization

Histogram Plot

A conventional display of a histogram for air temperature for both 2018 and 2019 with all boroughs:

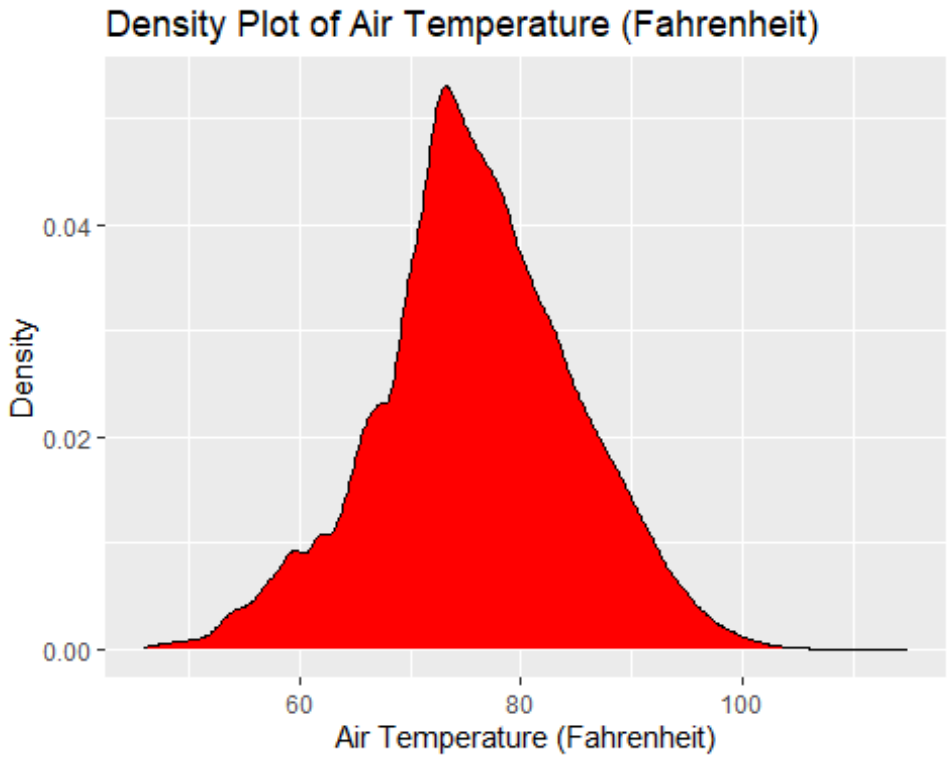
```
ggplot(good_data, aes(x =AirTemp)) +
  geom_histogram(binwidth = 1, fill = "gold", color = "black") +
  labs(title = "Distribution of Air Temperature", x = "Air Temperature
(Fahrenheit)", y = "Frequency")
```



Density Plots

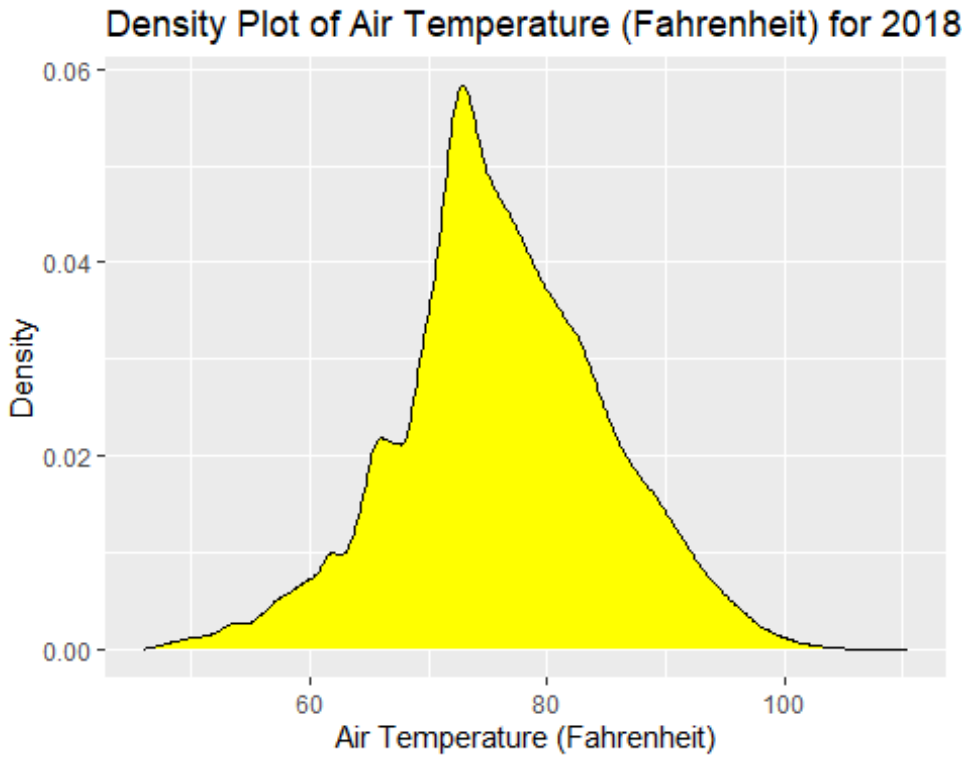
A (**smoothed**) density plot of the air temperature based on kernel density for 2018 and 2019 with all boroughs:

```
ggplot(good_data, aes(x = AirTemp)) +
  geom_density(fill = "red", color = "black") +
  labs(x = "Air Temperature (Fahrenheit)",
y = "Density",
title = "Density Plot of Air Temperature (Fahrenheit)")
```



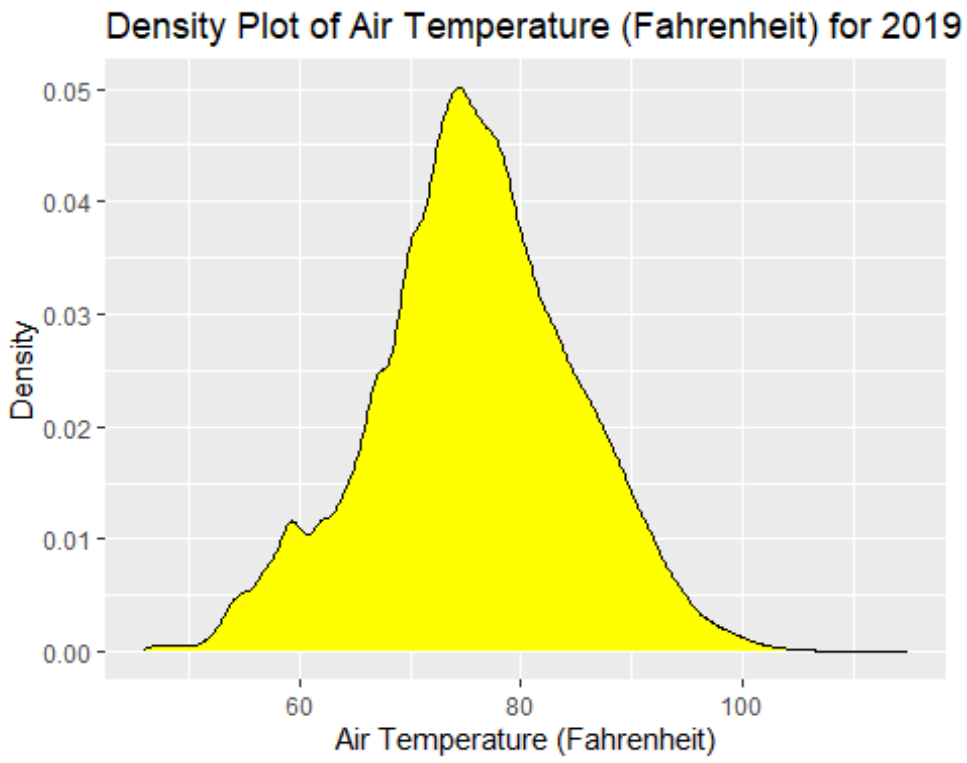
Case for only year 2018:

```
# Specifiying the year 2018
temp_data_2018 <- good_data |>
  filter(Year == 2018)
density_plot_2018 <- ggplot(temp_data_2018, aes(x = AirTemp)) +
  geom_density(fill = "yellow", color = "black") +
  labs(x = "Air Temperature (Fahrenheit)",
y = "Density",
title = "Density Plot of Air Temperature (Fahrenheit) for 2018")
density_plot_2018
```



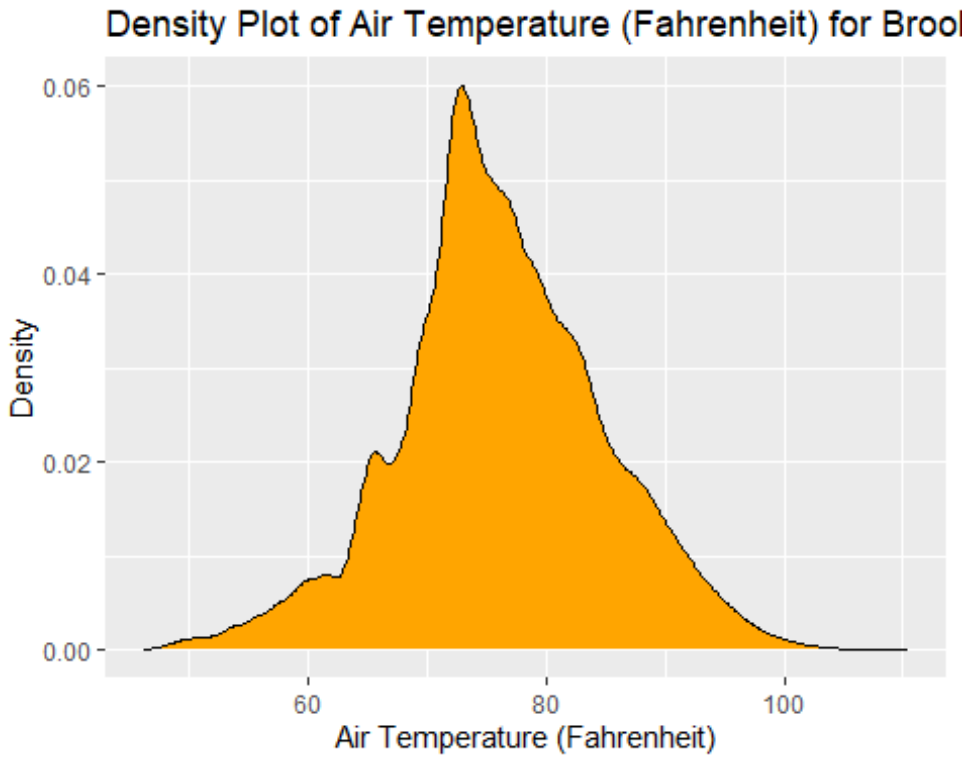
Case for only year 2019:

```
# Specifiying the year 2019
temp_data_2019 <- good_data |>
  filter(Year == 2019)
density_plot_2019 <- ggplot(temp_data_2019, aes(x = AirTemp)) +
  geom_density(fill = "yellow", color = "black") +
  labs(x = "Air Temperature (Fahrenheit)",
y = "Density",
title = "Density Plot of Air Temperature (Fahrenheit) for 2019")
density_plot_2019
```



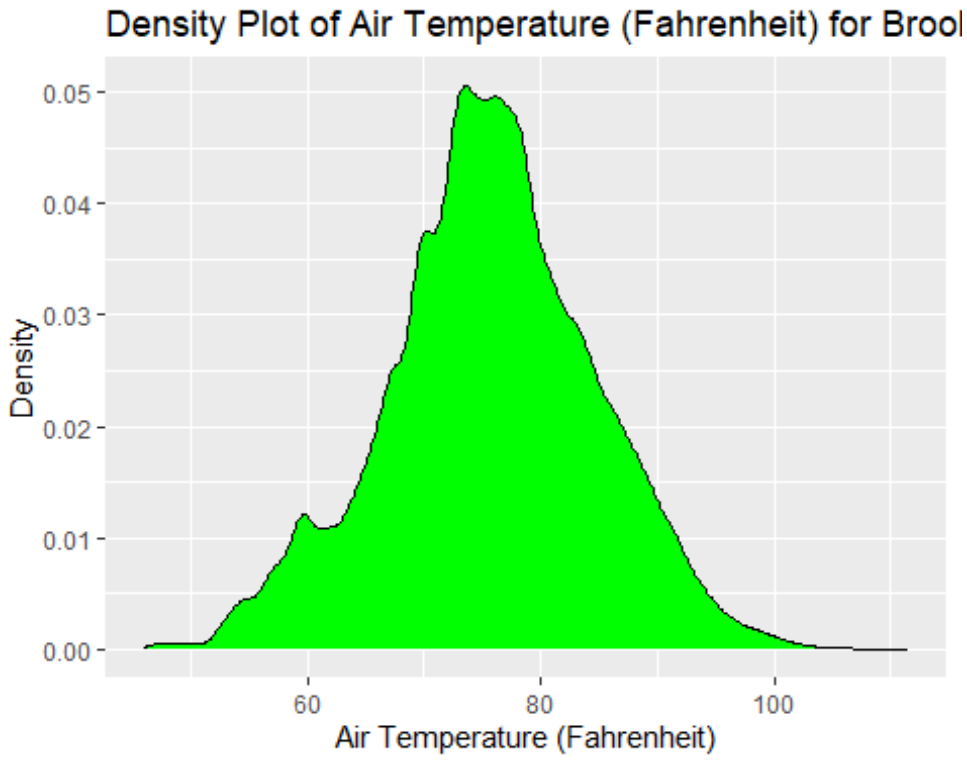
Case for Brooklyn for the year 2018:

```
# Specifiying the year 2018
brooklyn_temp_data_2018 <- good_data |>
  filter(Borough == "Brooklyn", Year == 2018)
brooklyn_density_plot_2018 <-
  ggplot(brooklyn_temp_data_2018, aes(x = AirTemp)) +
  geom_density(fill = "orange", color = "black") +
  labs(x = "Air Temperature (Fahrenheit)",
y = "Density",
title = "Density Plot of Air Temperature (Fahrenheit) for Brooklyn in 2018")
brooklyn_density_plot_2018
```



Case for Brooklyn for the year 2019:

```
# Specifiying the year 2019
brooklyn_temp_data_2019 <- good_data |>
  filter(Borough == "Brooklyn", Year == 2019)
brooklyn_density_plot_2018 <-
  ggplot(brooklyn_temp_data_2019, aes(x = AirTemp)) +
  geom_density(fill = "green", color = "black") +
  labs(x = "Air Temperature (Fahrenheit)",
y = "Density",
title = "Density Plot of Air Temperature (Fahrenheit) for Brooklyn in 2018")
brooklyn_density_plot_2018
```

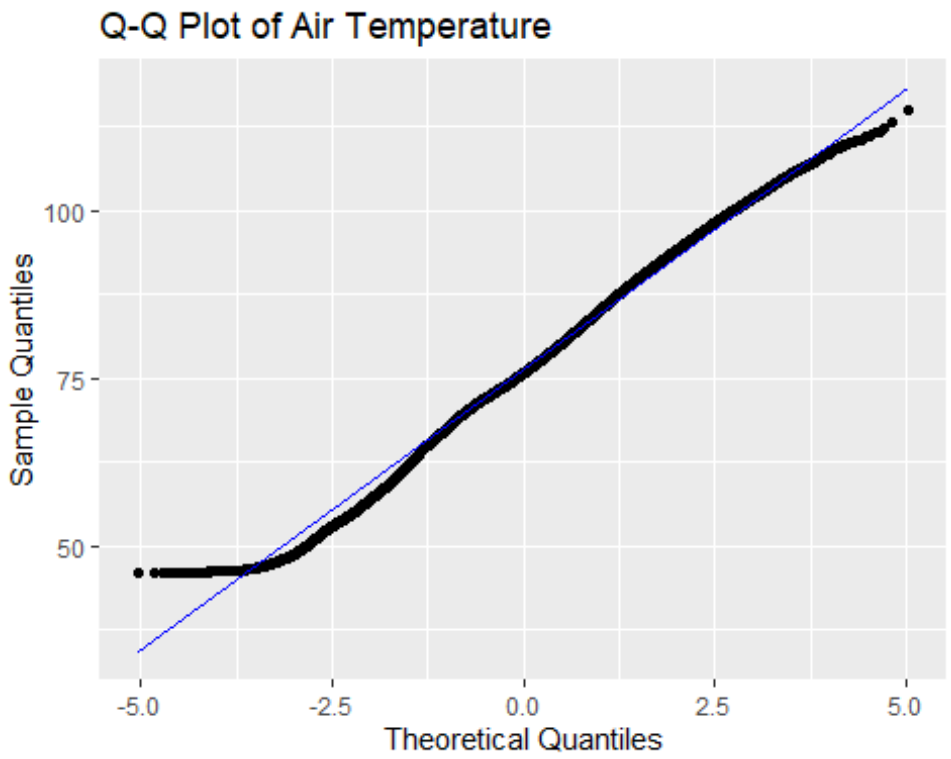


NOTE: the same can be done for the other boroughs concerning 2018 and 2019, respectively.

Quantile- Quantile Plot

A Q-Q plot, short for Quantile-Quantile plot, is a graphical method used to compare two probability distributions by plotting their quantiles against each other. It is particularly useful for assessing whether a dataset follows a specific theoretical distribution, such as the normal distribution.

```
ggplot(good_data, aes(sample = AirTemp)) +
stat_qq() +
stat_qq_line(color = "blue") +
labs(x = "Theoretical Quantiles",
y = "Sample Quantiles",
title = "Q-Q Plot of Air Temperature")
```



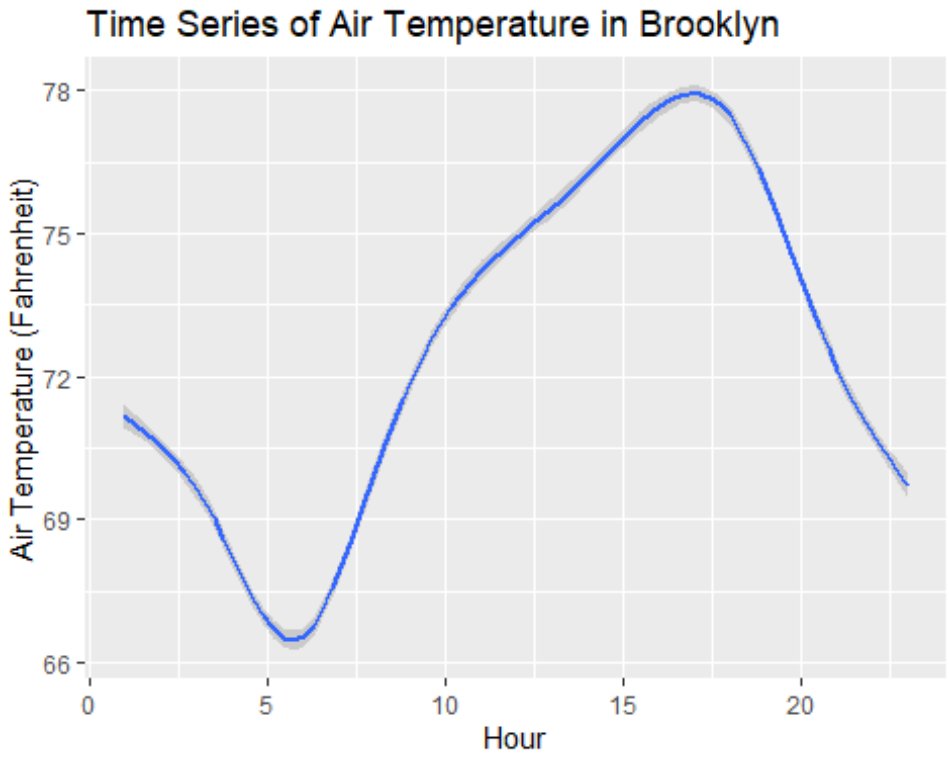
From above observation there is high convergence by the real data to the "theoretical distribution line"; assumed theoretical distribution by default is normal.

Time series

Time Series exhibition is expected since the air temperature measures are chronological. However, each day has a twenty four phase, and there are generally 365 days in a year. In the data the hour range is from 1 to 23. So, developing a time series beyond a data is proving to be a challenge; requires a bit of innovative column mutation (something I'm still trying to figure out). For convenience will specify a particular day of the year, subject to borough.

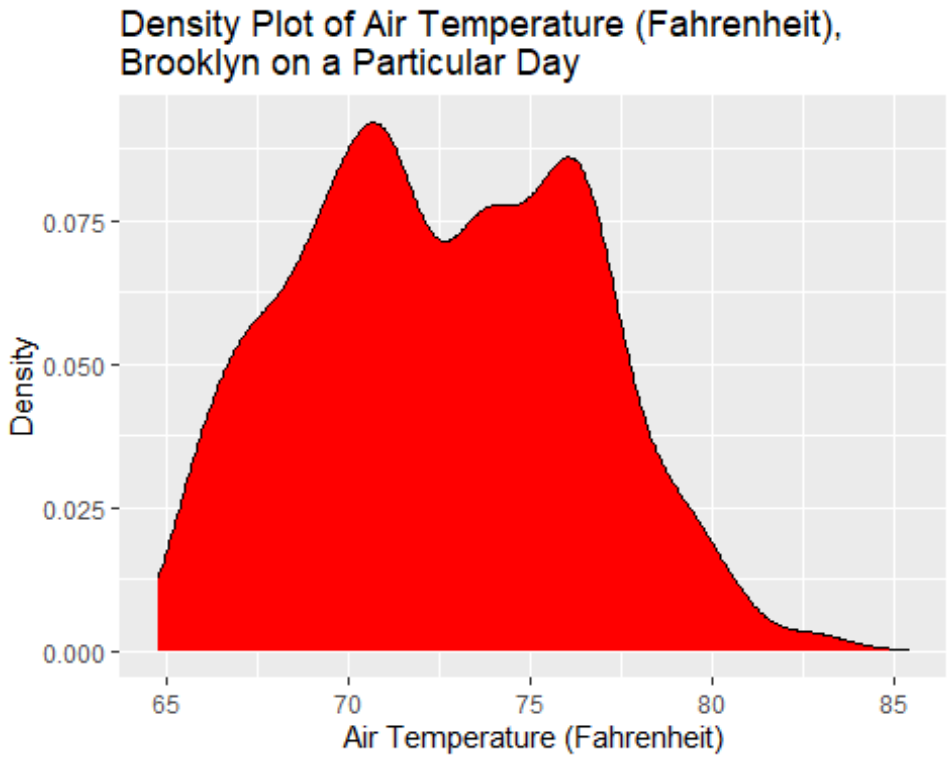
```
unique_day_air_temp_brooklyn <- good_data |>
  filter(Day == "06/15/2018", Borough == "Brooklyn")

unique_day_air_temp_queens <- good_data |>
  filter(Day == "06/15/2018", Borough == "Queens")

unique_day_air_temp_manhattan <- good_data |>
  filter(Day == "06/15/2018", Borough == "Manhattan")

unique_day_air_temp_bronx <- good_data |>
  filter(Day == "06/15/2018", Borough == "Bronx")

# Time Series of Air Temperature in Brooklyn
ggplot(unique_day_air_temp_brooklyn, aes(x = Hour, y = AirTemp)) +
  geom_smooth() +
```

```
  labs(title = "Time Series of Air Temperature in Brooklyn",
       x = "Hour", y = "Air Temperature (Fahrenheit)")

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



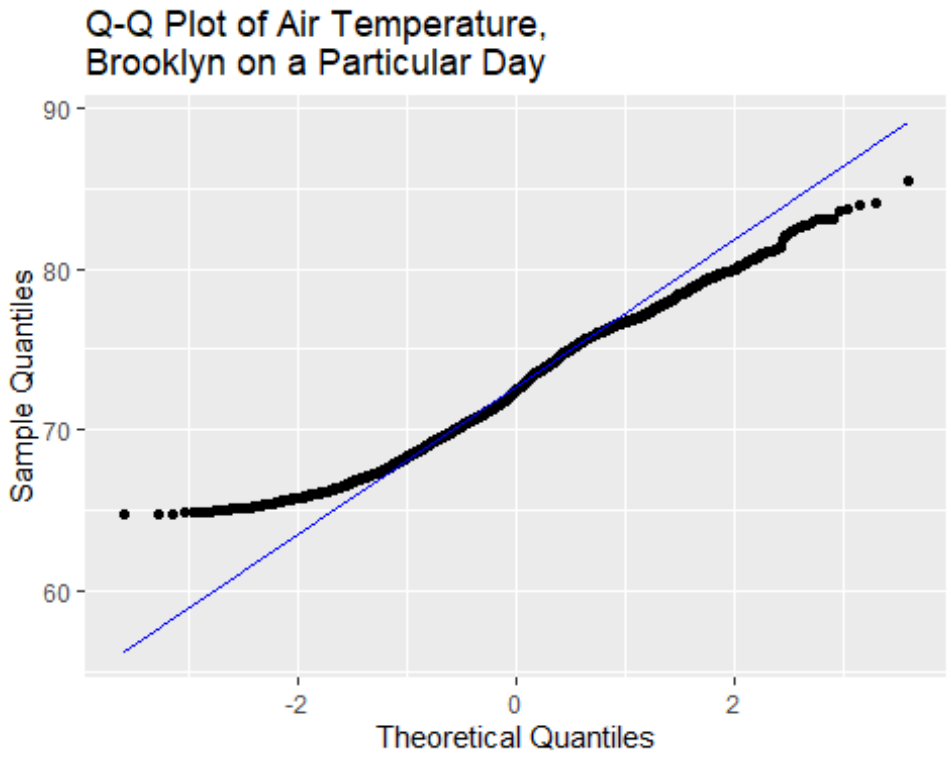
Density Plot for Brooklyn for Day in Question

```
ggplot(unique_day_air_temp_brooklyn, aes(x = AirTemp)) +
  geom_density(fill = "red", color = "black") +
  labs(x = "Air Temperature (Fahrenheit)",
y = "Density",
title = "Density Plot of Air Temperature (Fahrenheit),
Brooklyn on a Particular Day")
```



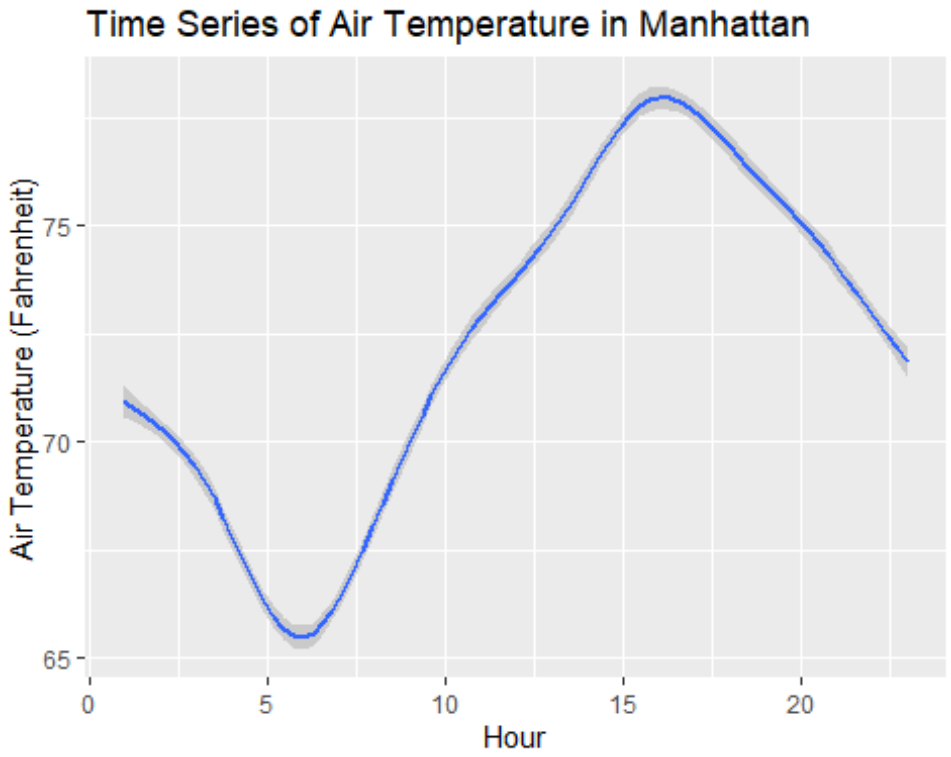
Q-Q Plot for Brooklyn for Day in Question

```
ggplot(unique_day_air_temp_brooklyn, aes(sample = AirTemp)) +
stat_qq() +
stat_qq_line(color = "blue") +
labs(x = "Theoretical Quantiles",
y = "Sample Quantiles",
title = "Q-Q Plot of Air Temperature,
Brooklyn on a Particular Day")
```



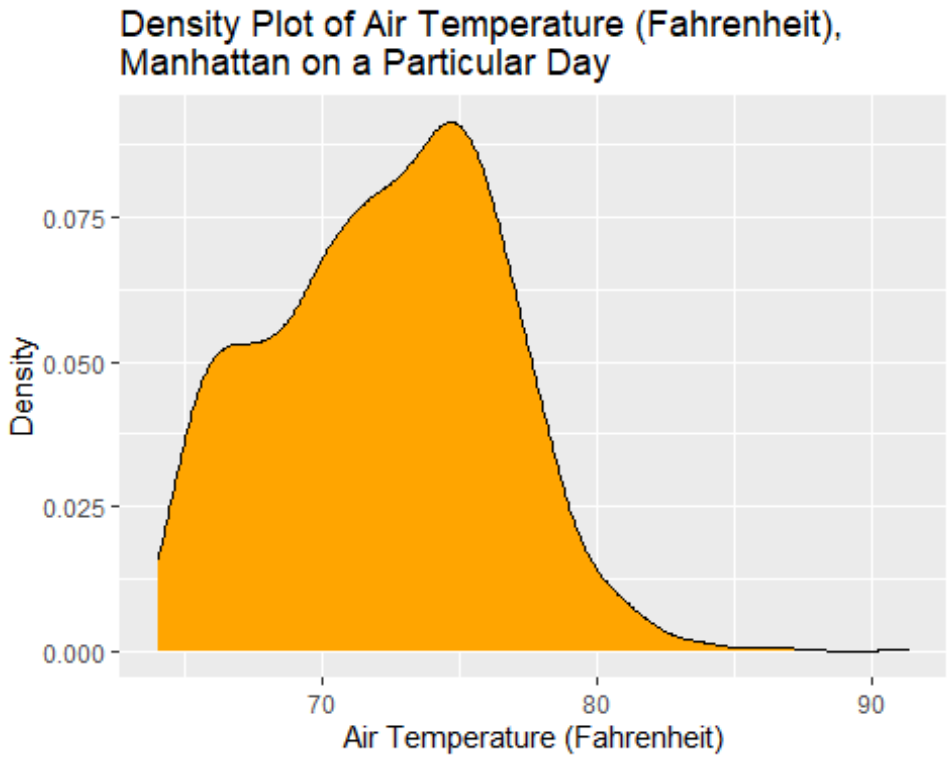

```
# Time Series of Air Temperature in Manhattan
ggplot(unique_day_air_temp_manhattan, aes(x = Hour, y = AirTemp)) +
  geom_smooth() +
  labs(title = "Time Series of Air Temperature in Manhattan",
       x = "Hour", y = "Air Temperature (Fahrenheit)")
```

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



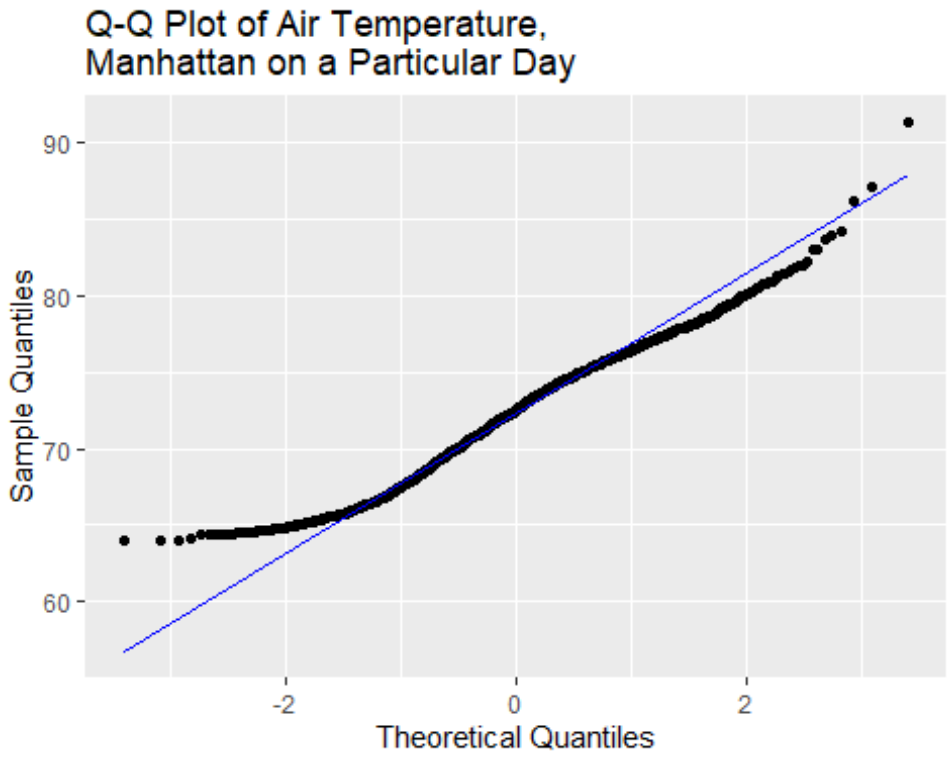
Density Plot for Manhattan for Day in Question

```
ggplot(unique_day_air_temp_manhattan, aes(x = AirTemp)) +
  geom_density(fill = "orange", color = "black") +
  labs(x = "Air Temperature (Fahrenheit)",
y = "Density",
title = "Density Plot of Air Temperature (Fahrenheit),
Manhattan on a Particular Day")
```



Q-Q Plot for Brooklyn for Day in Question

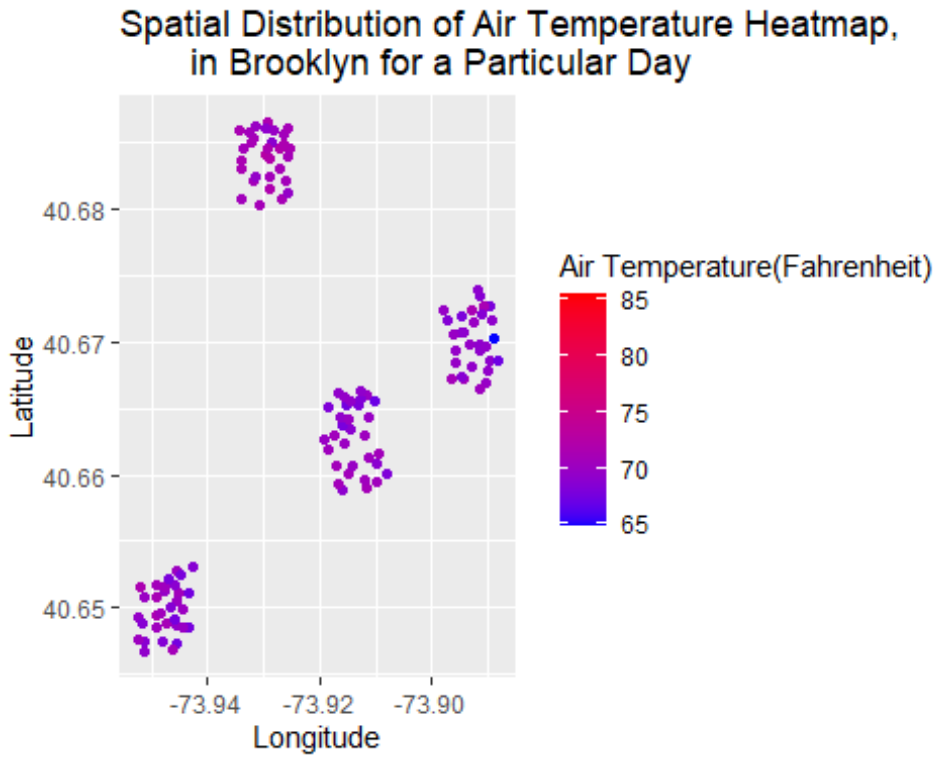
```
ggplot(unique_day_air_temp_manhattan, aes(sample = AirTemp)) +
stat_qq() +
stat_qq_line(color = "blue") +
labs(x = "Theoretical Quantiles",
y = "Sample Quantiles",
title = "Q-Q Plot of Air Temperature,
Manhattan on a Particular Day")
```



NOTE: such Q-Q plot display can also be done for Queens and the Bronx.

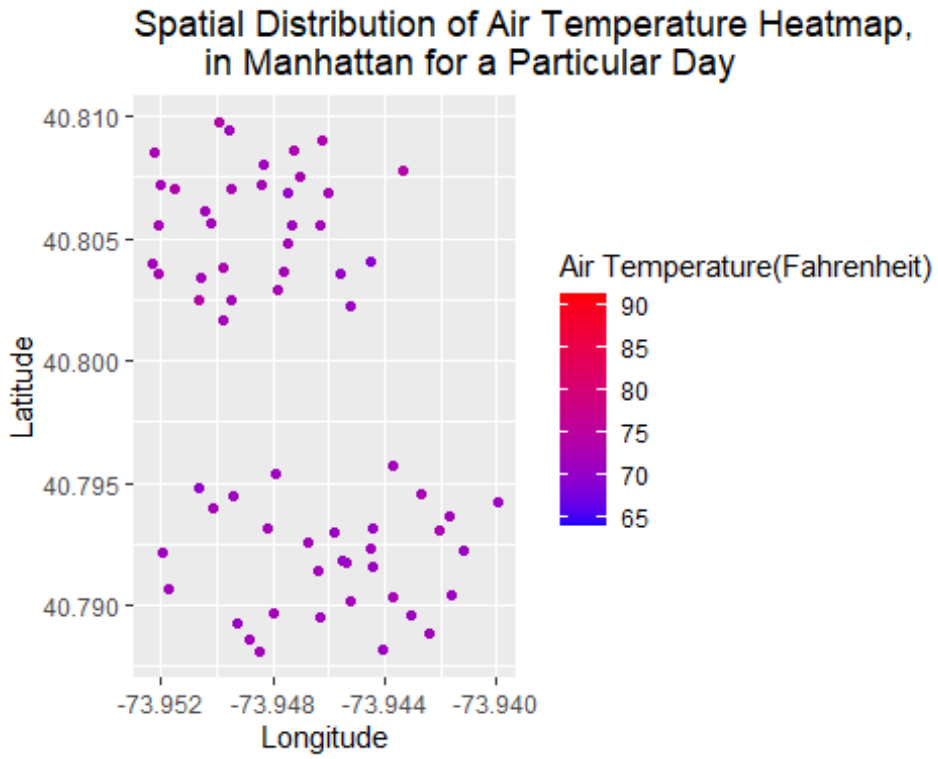
Spatial Visualization of Temperature (for a particular day) in Brooklyn

```
ggplot(unique_day_air_temp_brooklyn, aes(x = Longitude,
                                         y = Latitude, color = AirTemp)) +
  geom_point() +
  scale_color_gradient(low = "blue", high = "red") +
  labs(title = "Spatial Distribution of Air Temperature Heatmap,
       in Brooklyn for a Particular Day",
x ="Longitude", y = "Latitude", color ="Air Temperature(Fahrenheit)")
```



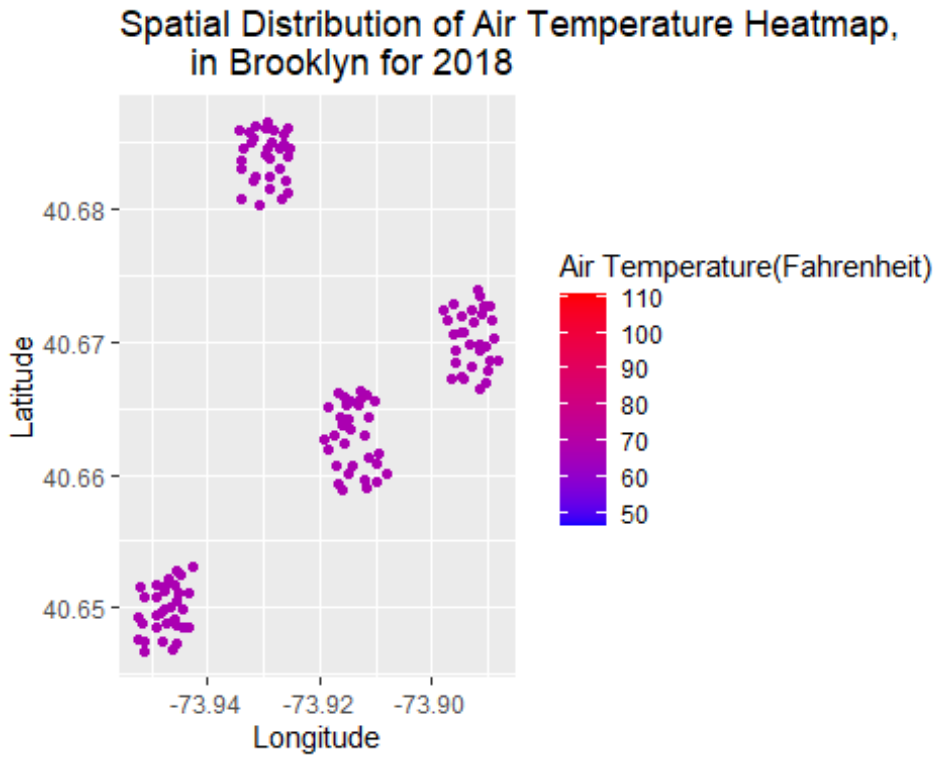
Spatial Visualization of Temperature (for a particular day) in Manhattan

```
ggplot(unique_day_air_temp_manhattan, aes(x = Longitude,
                                  y = Latitude, color = AirTemp)) +
  geom_point() +
  scale_color_gradient(low = "blue", high = "red") +
  labs(title = "Spatial Distribution of Air Temperature Heatmap,
       in Manhattan for a Particular Day",
x ="Longitude", y = "Latitude", color ="Air Temperature(Fahrenheit)")
```



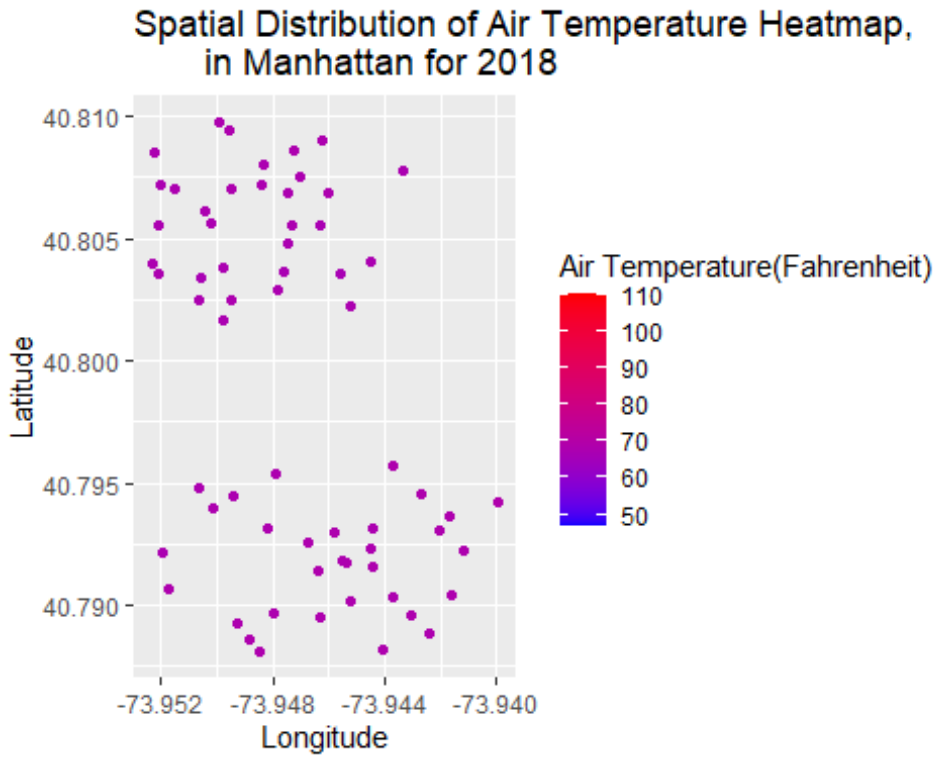
Spatial Visualization of Temperature (for a particular day) in Brooklyn for 2018 Total

```
spatial_temp_data_brooklyn_2018 <- good_data |>
  filter(Borough == "Brooklyn", Year == 2018)
ggplot(spatial_temp_data_brooklyn_2018, aes(x = Longitude,
                                    y = Latitude, color = AirTemp)) +
  geom_point() +
  scale_color_gradient(low = "blue", high = "red") +
  labs(title = "Spatial Distribution of Air Temperature Heatmap,
       in Brooklyn for 2018",
x ="Longitude", y = "Latitude", color ="Air Temperature(Fahrenheit)")
```



Spatial Visualization of Temperature (for a particular day) in Manhattan for 2018 Total

```
spatial_temp_data_manhattan_2018 <- good_data |>
  filter(Borough == "Manhattan", Year == 2018)
ggplot(spatial_temp_data_manhattan_2018, aes(x = Longitude,
                                  y = Latitude, color = AirTemp)) +
  geom_point() +
  scale_color_gradient(low = "blue", high = "red") +
  labs(title = "Spatial Distribution of Air Temperature Heatmap,
       in Manhattan for 2018",
x ="Longitude", y = "Latitude", color ="Air Temperature(Fahrenheit)")
```



NOTE: such spatial visualization display can also be done for Queens and the Bronx.

NOTE: from the prior visual displays, "clustering" has been accomplished without formerly applying the unsupervised learning method, simply by specifying the borough of interest.

Time Series for Aggregated Time in a Borough (Case Examples)

More Data cleaning

The data frame has an integer hour column that starts from 1 up to 23, then 0 to 23 repeatedly. I don't want a recycling of 0 to 23, rather for it to count on continuously. Instead of going back to 0, it should count on with 24 up to whatever concerning row size.

```
library(lubridate)

# Acquiring all hours for summer 2018
Aggregated_time_for_temp <- good_data |>
  select(Borough, Year, AirTemp, Hour) |>
  filter(Borough == "Brooklyn", Year == 2018)

str(Aggregated_time_for_temp)
```

```
tibble [376,636 × 4] (S3: tbl_df/tbl/data.frame)
 $ Borough: chr [1:376636] "Brooklyn" "Brooklyn" "Brooklyn" "Brooklyn" ...
 $ Year   : num [1:376636] 2018 2018 2018 2018 2018 ...
 $ AirTemp: num [1:376636] 71.2 70.2 69.4 68.3 67.1 ...
 $ Hour   : num [1:376636] 1 2 3 4 5 6 7 8 9 10 ...
```

```
 - attr(*, "na.action")= 'omit' Named int [1:11707] 132661 132662 132663
132664 132665 132666 132667 132668 132669 132670 ...
  ..- attr(*, "names")= chr [1:11707] "132661" "132662" "132663" "132664" ...
```

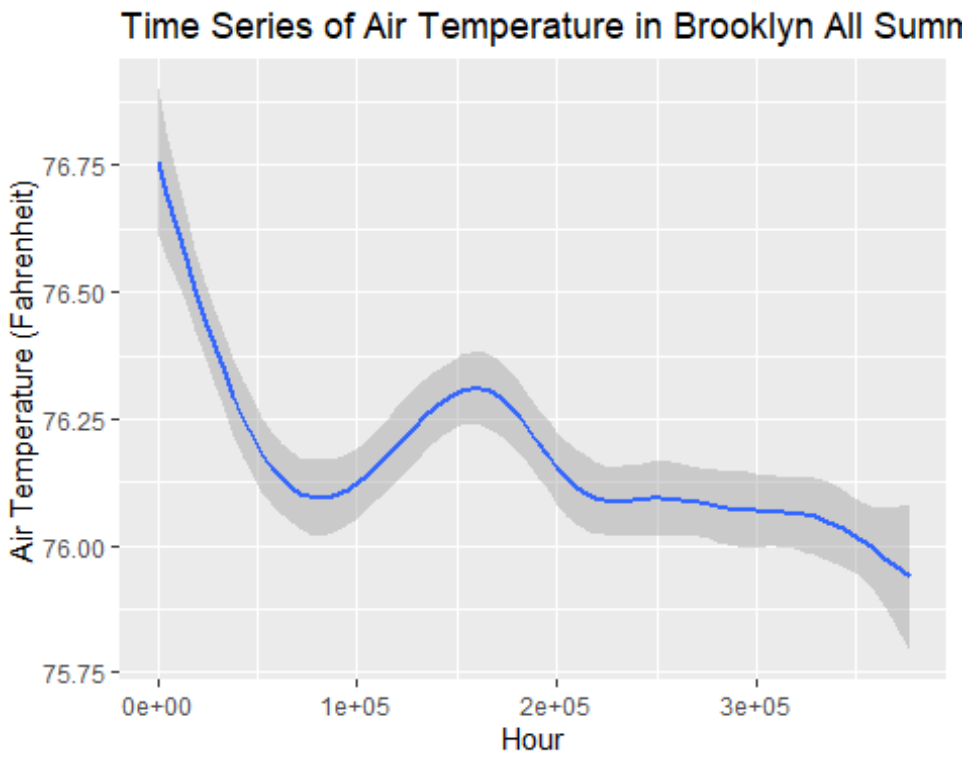
```
# Calculate the continuous sequence
continuous_seq <- seq(1, nrow(Aggregated_time_for_temp))

Aggregated_time_for_temp$Hour <- continuous_seq

# Print the modified dataframe
print(Aggregated_time_for_temp)
```

```
# A tibble: 376,636 × 4
   Borough   Year AirTemp  Hour
   <chr>    <dbl>   <dbl> <int>
 1 Brooklyn  2018    71.2     1
 2 Brooklyn  2018    70.2     2
 3 Brooklyn  2018    69.4     3
 4 Brooklyn  2018    68.3     4
 5 Brooklyn  2018    67.1     5
 6 Brooklyn  2018    66.0     6
 7 Brooklyn  2018    67.1     7
 8 Brooklyn  2018    68.9     8
 9 Brooklyn  2018    71.1     9
10 Brooklyn  2018    73.3    10
# ℹ 376,626 more rows
```

Now to Attempt Time Series Analysis

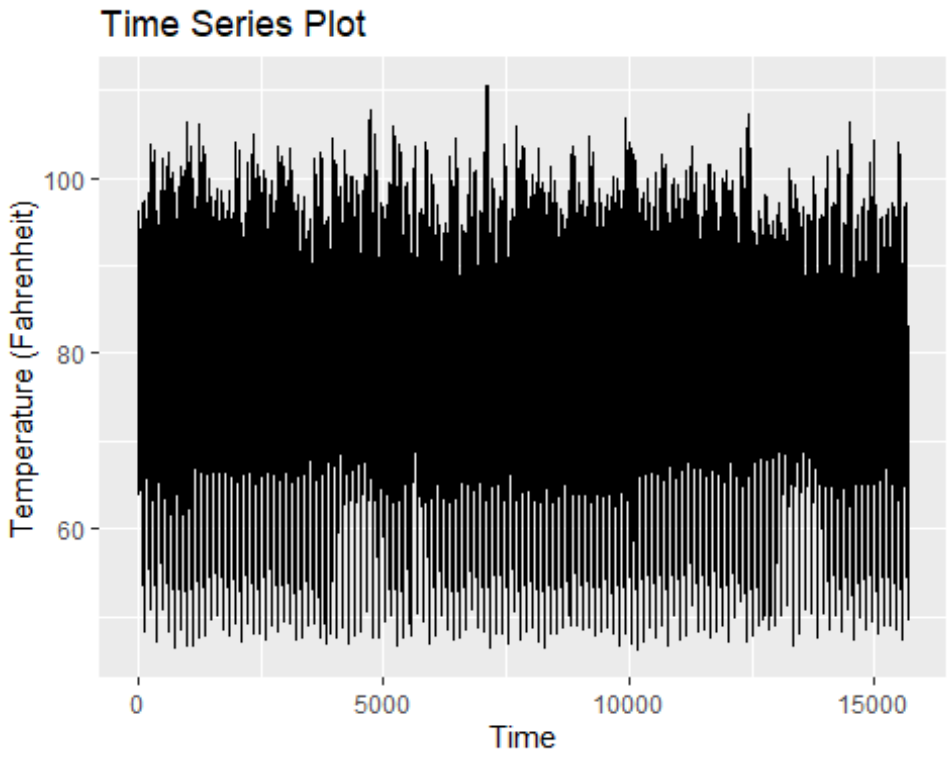
```
# Time Series of Air Temperature in Brooklyn for for all summer 2018
aggregate_time_bklyn_2018 <- ggplot(Aggregated_time_for_temp, aes(x = Hour, y
= AirTemp)) +
  geom_smooth() +
  labs(title = "Time Series of Air Temperature in Brooklyn All Summer 2018",
       x = "Hour", y = "Air Temperature (Fahrenheit)")
aggregate_time_bklyn_2018
```

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

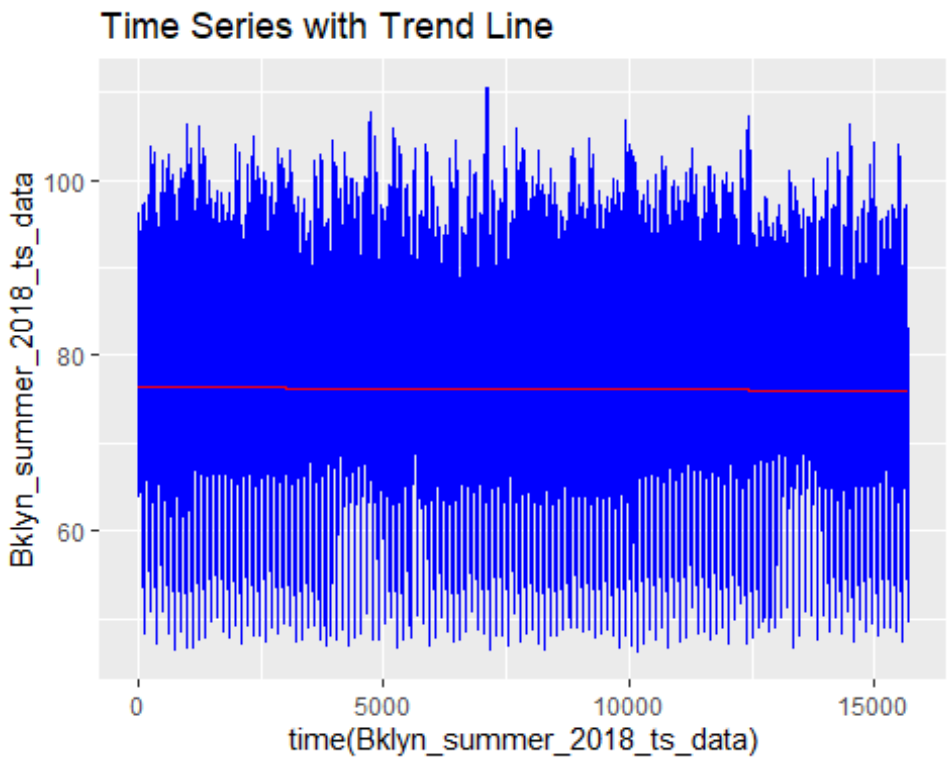


If the time series data has hourly increments, then the frequency of the time series is 24. This means that there are 24 observations within one complete cycle or unit of time. In other words, there are 24 hours in a day, so the frequency of your time series is 24.
If your time series has less than two periods, seasonal decomposition cannot be performed directly because the algorithm needs at least two full periods to identify a seasonal pattern accurately. In such cases, you can still perform trend analysis and explore autocorrelation patterns without decomposing the time series.

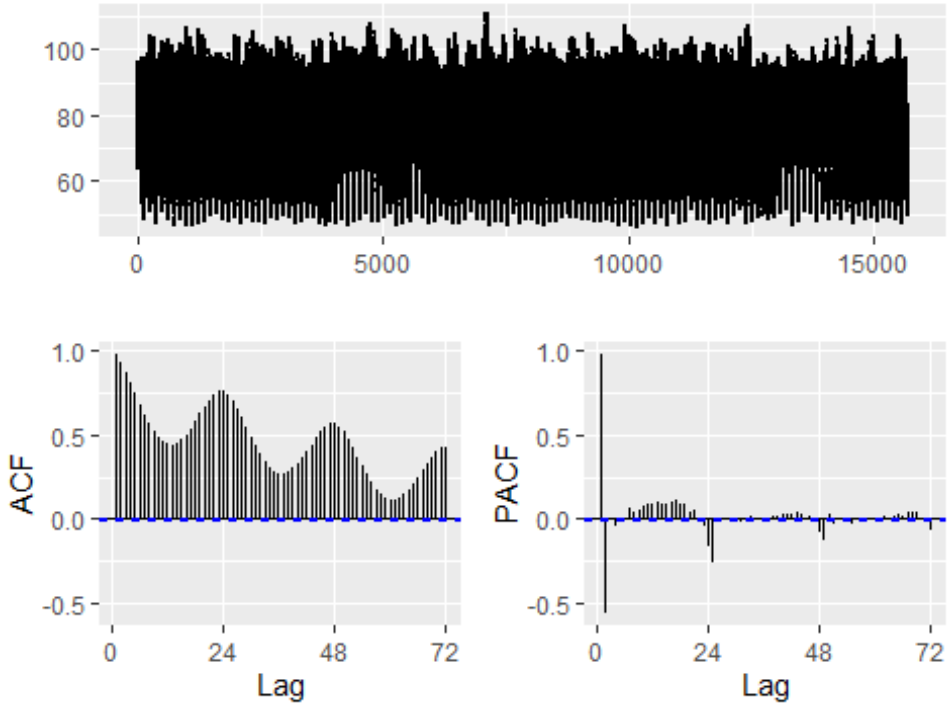
```
library(forecast)
# Create ts object from temperature and hour sequence
Bklyn_summer_2018_ts_data <- ts(Aggregated_time_for_temp$AirTemp, frequency =
24)

# Plot the time series
autoplot(Bklyn_summer_2018_ts_data, ylab = "Temperature (Fahrenheit)") +
  ggtitle("Time Series Plot")
```




```
# Fit a linear trend line
trend_model <- lm(Bklyn_summer_2018_ts_data ~
time(Bklyn_summer_2018_ts_data))
trend_line <- data.frame(time = time(Bklyn_summer_2018_ts_data),
                         trend = predict(trend_model))
ggplot() +
  geom_line(data = Bklyn_summer_2018_ts_data, aes(x =
time(Bklyn_summer_2018_ts_data), y = Bklyn_summer_2018_ts_data), color =
"blue") +
  geom_line(data = trend_line, aes(x = time, y = trend), color = "red") +
  labs(title = "Time Series with Trend Line")
```




```
# Plot autocorrelation
ggtsdisplay(Bklyn_summer_2018_ts_data)
```



```
summary(trend_model)


Call:
lm(formula = Bklyn_summer_2018_ts_data ~ time(Bklyn_summer_2018_ts_data))

Residuals:
    Min      1Q  Median      3Q     Max
-29.901  -5.042  -0.478   5.516  34.306

Coefficients:
                                Estimate Std. Error  t value Pr(>|t|)
(Intercept)                    7.636e+01  2.805e-02 2722.589  < 2e-16 ***
time(Bklyn_summer_2018_ts_data) -2.444e-05  3.095e-06   -7.897 2.87e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8.606 on 376634 degrees of freedom
Multiple R-squared:  0.0001655, Adjusted R-squared:  0.0001629
F-statistic: 62.36 on 1 and 376634 DF,  p-value: 2.871e-15
```

Explanation of Prior Summary Statistics

1. **Coefficients**: This table shows the estimated coefficients for each predictor variable in the model. Each row corresponds to a predictor variable, and the columns provide information such as the estimated coefficient (**`Estimate`**), standard error (**`Std. Error`**), t-value (**`t value`**), and corresponding p-value (**`Pr(>|t|)`**).
2. **Residuals**: This section provides summary statistics for the residuals of the model, including the mean (**`Residuals: Min 1Q Median 3Q Max`**), which should ideally be close to zero if the model is well-fitted, and other measures of spread and shape.
3. **Coefficients' significance**: The **`Pr(>|t|)`** column in the coefficients table indicates the p-value associated with each coefficient. This p-value is used to test the null hypothesis that the coefficient is equal to zero. If the p-value is less than a chosen significance level (commonly 0.05), then we reject the null hypothesis and conclude that the predictor variable is statistically significant in predicting the response variable.
4. **R-squared**: The **`R-squared`** value represents the proportion of variance in the response variable that is explained by the predictor variables in the model. It ranges from 0 to 1, with higher values indicating a better fit of the model to the data.
5. **Adjusted R-squared**: The **`Adjusted R-squared`** value adjusts the R-squared value for the number of predictor variables in the model. It penalizes adding unnecessary variables to the model and provides a better measure of model fit, especially when comparing models with different numbers of predictors.
6. **F-statistic**: The **`F-statistic`** tests the overall significance of the regression model. It evaluates whether the overall regression model is statistically significant compared to a model with no predictor variables. The associated p-value (**`Prob (F-statistic)`**) indicates the probability of obtaining the observed F-statistic under

the null hypothesis of no relationship between the predictor variables and the response variable.

7. **Residual standard error**: The **`Residual standard error`** (or residual standard deviation) is an estimate of the standard deviation of the residuals. It measures the average distance between the observed values and the values predicted by the model.
8. **Degrees of Freedom**: The **`Degrees of Freedom`** represent the number of independent observations in the data and are used in calculating the F-statistic and p-values.

Time Series Forecast

Concerning forecasting for the next 24 hours, due to over 2 millions instances of temperature measures, unfortunately, the result can't be acquired in practical time. However, the script for such is provided:

```
# Fit an ARIMA model to the time series data
arima_model <- auto.arima(Bklyn_summer_2018_ts_data)

# Forecast future values
forecast_values <- forecast(arima_model, h = 24) # Forecasting 24 hours ahead

# Plot the forecast
plot(forecast_values, xlab = "Hour", ylab = "Temperature", main = "Temperature Forecast")
```

However, a parallelized version to acquire the 24 hour forecast (**which still isn't practical to operate on the computer I apply**):

```
library(foreach)
library(parallel)

# Detect available CPU cores
num_cores <- detectCores()

# Create cluster
cl <- makeCluster(num_cores)

# Register parallel backend
registerDoParallel(cl)

# Fit ARIMA models

arima_models <- foreach(i = 1:num_cores) %dopar% {
library(forecast) # Load forecast package inside parallel loop
auto.arima(Bklyn_summer_2018_ts_data, trace = FALSE) # Fit ARIMA model without
printing trace }
```

```
# Stop the parallel backend
stopCluster(cl)

# Combine Results
combined_arima_model <- combine(arima_models)

# Forecast future values
forecast_values <- forecast(combined_arima_model, h = 24)

# Plot the forecast
plot(forecast_values, xlab = "Hour", ylab = "Temperature", main = "Temperature Forecast")
```

Correlation Analysis

Correlation analysis is a statistical method used to measure the strength and direction of the linear relationship between two quantitative variables. It helps to determine whether and to what extent changes in one variable are associated with changes in another variable. Here's how correlation analysis works:

Calculating Pearson Correlation:

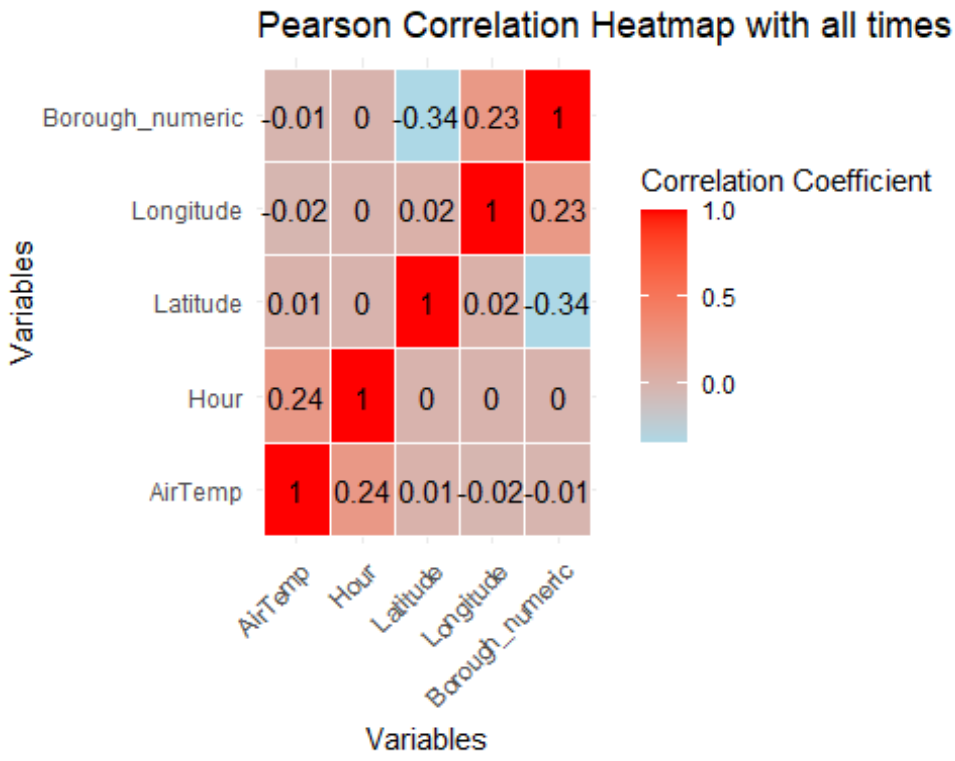
For large data sets, applying computational tools such as R with base packages or others of choice such correlation formula is applied. For such measure I reverted to the 0-23 hour cycle per day to preserve the daily attribute, rather than association determination of other variables with a time scale that has lost meaning, say 78 days (by 24 hours) = 1872 hours - 1 (due to starting from 1). **Note:** the temperature measurment started on 06/15/2018, so approximately 92 days of summer minus 14 days in June. The 1871 consectutive hours orientation is more appropriate for time series.

```
correlation_variables <- good_data |>
  select(AirTemp, Hour, Latitude, Longitude, Borough_numeric)
```

Case for all considered boroughs with all times

```
library(reshape2)

# Calculate Pearson correlation coefficients
correlation_matrix <- cor(correlation_variables, method = "pearson")

# Convert correlation matrix to long format
correlation_data <- melt(correlation_matrix)

# Plot heat map with correlation measures
ggplot(correlation_data, aes(Var1, Var2,
                             fill = value, label = round(value, 2))) +
  geom_tile(color = "white") +
```

```
  geom_text(color = "black", size = 4) +  # Add correlation labels
  scale_fill_gradient(low = "lightblue", high = "red") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "Pearson Correlation Heatmap with all times",
       x = "Variables",
       y = "Variables",
       fill = "Correlation Coefficient")
```



Case for a specified borough with all times

Such is not feasible because a column with all values being the same is typically referred to as a constant variable or a constant feature. This means that the variable does not vary across observations and provides no information for modeling or analysis.
When the standard deviation of a variable is zero, it means that all the values of that variable are the same, leading to division by zero when calculating correlations.
Constant variables are often considered uninformative because they do not contribute to explaining or predicting variability in the outcome of interest. In statistical analysis or machine learning tasks, constant variables are usually removed or handled separately to avoid introducing noise or bias into the analysis.

Omission of Regression

Due to the observed correlation results prior, namely, correlation coefficients close to 0, suggesting weak or no linear relationships. Multilinear regression will not be pursued.

Questions to Answer

Given the following questions to answer:
1. How much difference in temperature is there within a "neighborhood"?
Well, We can calculate the temperature difference within each neighborhood by grouping the data by **Borough_numeric** and computing summary statistics such as mean, median, or standard deviation of temperature.

```
# Calculate temperature difference within each neighborhood

neighborhood_temp_diff <- correlation_variables |>
  group_by(Borough_numeric) |>
  summarize(Temp_Difference = max(AirTemp) - min(AirTemp))
neighborhood_temp_diff
```

```
# A tibble: 4 × 2
  Borough_numeric Temp_Difference
            <int>           <dbl>
1               1            68.9
2               2            65.5
3               3            67.1
4               4            62.2
```

Okay, well, how to associate such outputs to latitude, longitude and hour?

In this code:

- We first calculate the mean temperature for each unique combination of latitude, longitude, and hour within each neighborhood.
- Then, we calculate the temperature difference within each neighborhood for each hour by taking the difference between the maximum and minimum mean temperatures.
- Next, we merge the temperature difference with latitude, longitude, and hour by using the **merge()** function, specifying the columns to merge on (**Borough_numeric** and **Hour**).
- Finally, we print the resulting dataframe **neighborhood_temp_diff**, which contains the temperature difference along with latitude, longitude, and hour information for each neighborhood.

```
# Calculate mean temperature for each location (latitude, longitude, hour)
within each neighborhood
location_hour_mean_temp <- correlation_variables %>%
  group_by(Borough_numeric, Latitude, Longitude, Hour) %>%
  summarize(Mean_Temperature = mean(AirTemp))
```

```
`summarise()` has grouped output by 'Borough_numeric', 'Latitude',
'Longitude'.
You can override using the `.groups` argument.
```

```
# Calculate temperature difference within each neighborhood for each hour
neighborhood_temp_diff <- location_hour_mean_temp %>%
  group_by(Borough_numeric, Hour) %>%
  summarize(Temp_Difference = max(Mean_Temperature) - min(Mean_Temperature))
```

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

```
# Associate temperature difference with latitude, longitude, and hour
neighborhood_temp_diff <- merge(neighborhood_temp_diff,
location_hour_mean_temp, by = c("Borough_numeric", "Hour"))

# Output neighborhood temperature difference with latitude, longitude, and
hour
head(neighborhood_temp_diff)
```

```
  Borough_numeric Hour Temp_Difference Latitude Longitude Mean_Temperature
1               1    0        5.634561 40.80257 -73.90781         73.22548
2               1    0        5.634561 40.85262 -73.91330         72.90115
3               1    0        5.634561 40.85198 -73.90730         73.79589
4               1    0        5.634561 40.85116 -73.91220         73.78114
5               1    0        5.634561 40.80656 -73.91180         72.89748
6               1    0        5.634561 40.84999 -73.90940         74.20676
```

```
str(neighborhood_temp_diff)
```

```
'data.frame':   10440 obs. of  6 variables:
 $ Borough_numeric : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Hour            : num  0 0 0 0 0 0 0 0 0 0 ...
 $ Temp_Difference : num  5.63 5.63 5.63 5.63 5.63 ...
 $ Latitude        : num  40.8 40.9 40.9 40.9 40.8 ...
 $ Longitude       : num  -73.9 -73.9 -73.9 -73.9 -73.9 ...
 $ Mean_Temperature: num  73.2 72.9 73.8 73.8 72.9 ...
```

```
unique_values <- unique(neighborhood_temp_diff$Borough_numeric)

# Print unique values
print(unique_values)
```

```
[1] 1 2 3 4
```

2. What time of day is the hottest/coolest?
   We can find the hottest and coolest time of day by aggregating temperature data across all neighborhoods and examining the hourly trends.

```
# Calculate hourly mean temperature
hourly_temp <- correlation_variables |>
  group_by(Hour) |>
```

```
  summarize(Mean_Temperature = mean(AirTemp))

# Hottest time of day
hottest_hour <- hourly_temp$Hour[which.max(hourly_temp$Mean_Temperature)]
cat("Hottest time of day:", hottest_hour, "hour\n")
```

```
Hottest time of day: 15 hour
```

```
# Coolest time of day
coolest_hour <- hourly_temp$Hour[which.min(hourly_temp$Mean_Temperature)]
cat("Coolest time of day:", coolest_hour, "hour\n")
```

```
Coolest time of day: 6 hour
```

3. What locations are hotter/cooler than others in the same neighborhood?
   We can compare temperature differences within each neighborhood by examining the temperature distribution across different locations (Latitude and Longitude).

```
# Calculate temperature statistics for each location within each neighborhood
location_temp_stats <- correlation_variables |>
  group_by(Borough_numeric, Latitude, Longitude) |>
  summarize(Mean_Temperature = mean(AirTemp),
            Min_Temperature = min(AirTemp),
            Max_Temperature = max(AirTemp),
            .groups = "drop")  # Drop grouping information

# Identify locations with extreme temperatures within each neighborhood
hot_locations <- location_temp_stats |>
  filter(Mean_Temperature > quantile(Mean_Temperature, 0.75))
cool_locations <- location_temp_stats |>
  filter(Mean_Temperature < quantile(Mean_Temperature, 0.25))
print(hot_locations)
```

```
# A tibble: 109 × 6
   Borough_numeric Latitude Longitude Mean_Temperature Min_Temperature
             <int>    <dbl>     <dbl>            <dbl>           <dbl>
 1               1     40.8     -73.9             77.6            46.4
 2               1     40.8     -73.9             77.8            46.1
 3               1     40.8     -73.9             76.4            46.2
 4               1     40.8     -73.9             77.3            46.4
 5               1     40.8     -73.9             77.3            46.5
 6               1     40.8     -73.9             77.6            46.3
 7               1     40.8     -73.9             77.8            46.9
 8               1     40.8     -73.9             76.5            46.1
 9               1     40.8     -73.9             77.3            47.6
10               1     40.8     -73.9             77.6            46.6
# ℹ 99 more rows
# ℹ 1 more variable: Max_Temperature <dbl>
```

```
print(cool_locations)
```

```
# A tibble: 109 × 6
   Borough_numeric Latitude Longitude Mean_Temperature Min_Temperature
             <int>    <dbl>     <dbl>            <dbl>           <dbl>
 1               1     40.8     -73.9             75.0            46.2
 2               1     40.8     -73.9             75.1            46.2
 3               1     40.8     -73.9             75.3            46.5
 4               1     40.9     -73.9             74.7            46.0
 5               1     40.9     -73.9             75.0            46.3
 6               1     40.9     -73.9             75.3            46.4
 7               1     40.9     -73.9             75.1            46.1
 8               1     40.9     -73.9             74.8            46.3
 9               1     40.9     -73.9             74.9            46.2
10               1     40.9     -73.9             75.2            46.3
# ℹ 99 more rows
# ℹ 1 more variable: Max_Temperature <dbl>
```

```
str(hot_locations)
```

```
tibble [109 × 6] (S3: tbl_df/tbl/data.frame)
 $ Borough_numeric : int [1:109] 1 1 1 1 1 1 1 1 1 1 ...
 $ Latitude        : num [1:109] 40.8 40.8 40.8 40.8 40.8 ...
 $ Longitude       : num [1:109] -73.9 -73.9 -73.9 -73.9 -73.9 ...
 $ Mean_Temperature: num [1:109] 77.6 77.8 76.4 77.3 77.3 ...
 $ Min_Temperature : num [1:109] 46.4 46.1 46.2 46.4 46.5 ...
 $ Max_Temperature : num [1:109] 104 108 101 110 102 ...
```

```
str(cool_locations)
```

```
tibble [109 × 6] (S3: tbl_df/tbl/data.frame)
 $ Borough_numeric : int [1:109] 1 1 1 1 1 1 1 1 1 1 ...
 $ Latitude        : num [1:109] 40.8 40.8 40.8 40.9 40.9 ...
 $ Longitude       : num [1:109] -73.9 -73.9 -73.9 -73.9 -73.9 ...
 $ Mean_Temperature: num [1:109] 75 75.1 75.3 74.7 75 ...
 $ Min_Temperature : num [1:109] 46.2 46.2 46.5 46 46.3 ...
 $ Max_Temperature : num [1:109] 104.9 98 101.3 98.6 98.1 ...
```

4. Which neighborhoods are hotter/cooler than others?
   We can compare the average temperatures across different neighborhoods.

```
# Calculate average temperature for each combination of latitude, longitude,
and neighborhood
neighborhood_avg_temp <- correlation_variables |>
  group_by(Borough_numeric, Latitude, Longitude) |>
  summarize(Avg_Temperature = mean(AirTemp), .groups = "drop")

# Calculate overall average temperature for each neighborhood
neighborhood_avg_temp <- neighborhood_avg_temp |>
  group_by(Borough_numeric) |>
  summarize(Overall_Avg_Temperature = mean(Avg_Temperature), .groups =
"drop")
```

```r
# Identify neighborhoods hotter or cooler than others
hot_neighborhoods <- neighborhood_avg_temp |>
  filter(Overall_Avg_Temperature > quantile(Overall_Avg_Temperature, 0.75))

cool_neighborhoods <- neighborhood_avg_temp |>
  filter(Overall_Avg_Temperature < quantile(Overall_Avg_Temperature, 0.25))

# Print the results
cat("Hotter Neighborhoods:\n")
```

```
Hotter Neighborhoods:
```

```r
print(hot_neighborhoods)
```

```
# A tibble: 1 × 2
  Borough_numeric Overall_Avg_Temperature
            <int>                   <dbl>
1               1                    76.0
```

```r
cat("Cooler Neighborhoods:\n")
```

```
Cooler Neighborhoods:
```

```r
print(cool_neighborhoods)
```

```
# A tibble: 1 × 2
  Borough_numeric Overall_Avg_Temperature
            <int>                   <dbl>
1               4                    75.4
```

```r
str(hot_neighborhoods)
```

```
tibble [1 × 2] (S3: tbl_df/tbl/data.frame)
 $ Borough_numeric        : int 1
 $ Overall_Avg_Temperature: num 76
```

```r
str(cool_neighborhoods)
```

```
tibble [1 × 2] (S3: tbl_df/tbl/data.frame)
 $ Borough_numeric        : int 4
 $ Overall_Avg_Temperature: num 75.4
```

Machine Learning with XGBoost

In supervised learning, the algorithm learns from labeled data, where each data point is associated with a target variable or outcome. The goal of supervised learning is to learn a mapping from input features to the target variable, such that the algorithm can make accurate predictions on unseen data.

XGBoost is classified as a supervised machine learning algorithm. In supervised learning, the algorithm learns from labeled data, where each data point is associated with a target variable or outcome. XGBoost, short for Extreme Gradient Boosting, is a powerful and versatile machine learning algorithm renowned for its performance and accuracy in various predictive modeling tasks. Developed by Tianqi Chen and now maintained by the community, XGBoost has gained widespread popularity due to its scalability, efficiency, and effectiveness across a range of domains and datasets.

In the case of XGBoost, it is used for both regression and classification tasks:

1. **Regression**: In regression tasks, the target variable is continuous, and the goal is to predict a numeric value. XGBoost builds a regression model that learns to predict the numeric target variable based on input features.

2. **Classification**: In classification tasks, the target variable is categorical, and the goal is to predict the class label or category to which a data point belongs. XGBoost builds a classification model that learns to classify data points into different categories based on input features.

XGBoost achieves its predictive power by employing an ensemble of weak learners, typically decision trees, and combining their predictions to form a strong predictive model. It iteratively builds decision trees in a sequential manner, optimizing a loss function to minimize prediction errors. XGBoost's ability to handle both regression and classification tasks, along with its scalability, efficiency, and accuracy, makes it a widely used and versatile supervised learning algorithm in machine learning applications.

Gradient Boosting is a machine learning technique used for regression and classification tasks. It's a type of ensemble learning method where a series of weak learners (often decision trees) are trained sequentially, each one trying to correct the errors of its predecessor. The key idea behind gradient boosting is to fit a new model to the residual errors made by the previous model.

At its core, XGBoost belongs to the family of gradient boosting algorithms, which iteratively combines multiple weak learners, typically decision trees, to create a strong ensemble model. However, what sets XGBoost apart is its innovative implementation, which incorporates several key features:

1. **Regularization**: XGBoost incorporates L1 and L2 regularization terms into the objective function, helping to prevent overfitting and improve generalization performance.

2. **Gradient Boosting with Trees (GBTree)**: XGBoost is designed to optimize gradient boosting with decision trees as base learners, providing flexibility in model complexity and capturing non-linear relationships in the data effectively.

3. **Tree Pruning**: XGBoost employs a technique known as tree pruning to remove splits that have little impact on improving the model's performance, resulting in more efficient and compact trees.

4. **Parallel and Distributed Computing**: XGBoost is optimized for parallel computation, leveraging multi-threading on a single machine and distributed computing across multiple machines to accelerate model training and inference.

5. **Customizable Objective Functions**: Users can define custom objective functions tailored to specific problem domains or tasks, allowing for greater flexibility and adaptability.

6. **Feature Importance**: XGBoost provides built-in methods for assessing feature importance, enabling insights into the relative contribution of each feature to the model's predictions.

7. **Early Stopping**: XGBoost supports early stopping, allowing model training to halt when the performance on a validation dataset stops improving, thus preventing overfitting and saving computational resources.

Due to these features, XGBoost has become a popular choice for a wide range of machine learning tasks, including classification, regression, ranking, and recommendation systems. Its robustness, efficiency, and interpretability make it a valuable tool for both practitioners and researchers seeking high-performance predictive modeling solutions.

Parallelization in XGBoost

Parallelization refers to the process of breaking down a task into smaller sub-tasks that can be executed simultaneously or concurrently on multiple computing resources, such as CPU cores, threads, or distributed computing nodes. The primary goal of parallelization is to speed up the execution of tasks by leveraging the computational power of multiple resources in parallel.

There's interest in parallelization in this project due to over 2 million rows with 5 columns considered.

In the context of machine learning and data analysis, parallelization can be applied to various stages of the modeling pipeline, including data preprocessing, model training, hyperparameter tuning, and model evaluation. Parallelization techniques can significantly improve the efficiency and scalability of machine learning algorithms, enabling them to handle large datasets and complex models more effectively.

Thee approach to parallelization in this demonstration will be multi-threading. **Multi-threading** involves executing multiple threads concurrently within a single process. Each thread performs a different task or computation, and they share the same memory space. Multi-threading is suitable for tasks that can benefit from concurrent execution, such as data preprocessing and model training.

In R, parallelization in XGBoost can be achieved using the **`xgboost`** package, which provides support for multi-threading and distributed computing. Here's how you can enable parallelization in XGBoost in R:

```r
library(caret)
# Split data into train and test sets
set.seed(123)
train_index <- createDataPartition(y = correlation_variables$AirTemp,
                                   p = 0.7, list = FALSE)
train_data <- correlation_variables[train_index, ]
test_data <- correlation_variables[-train_index, ]


library(xgboost)
library(parallel)

# Specify feature names excluding the target variable (AirTemp)
feature_names <- c("Hour", "Latitude", "Longitude", "Borough_numeric")

# Convert data frame to matrix, excluding the target variable
train_matrix <- as.matrix(train_data[, feature_names])

# Detect the number of CPU cores
num_cores <- parallel::detectCores()

# Specify the number of boosting rounds
nrounds <- 10  # Adjust this value based on your problem and dataset

# Train XGBoost model with multi-threading
bst <- xgboost(data = train_matrix,
               label = train_data$AirTemp,
               nthread = num_cores,
               nrounds = nrounds)  # Specify number of boosting rounds
```

```
[1] train-rmse:53.525581
[2] train-rmse:37.910092
[3] train-rmse:27.156944
[4] train-rmse:19.865402
[5] train-rmse:15.053125
[6] train-rmse:12.009414
[7] train-rmse:10.190279
[8] train-rmse:9.167409
[9] train-rmse:8.620696
[10]    train-rmse:8.339375
```

Conclusion

In conclusion, this study sheds light on the complex dynamics of street-level temperature variations in New York City. Through exploratory data analysis and visualization, identified

are temperature values that appear quite stable across neighborhoods throughout the summer seasons of 2018 and 2019. Additionally, machine learning models provided valuable insights into predicting air temperatures based on hourly, geographical, and temporal features. The findings underscore the importance of hyperlocal temperature monitoring in urban areas, offering crucial insights for policymakers, urban planners, and public health officials to develop targeted interventions aimed at mitigating the adverse effects of urban heat islands and enhancing urban resilience. Moving forward, continued monitoring and analysis of hyperlocal temperature data will be essential for informing evidence-based decision-making and fostering climate-resilient cities.

References

Department of Health and Mental Hygiene (DOHMH). (2021, August 20). *Hyperlocal Temperature Monitoring: NYC open data*. Hyperlocal Temperature Monitoring | NYC Open Data. https://data.cityofnewyork.us/dataset/Hyperlocal-Temperature-Monitoring/qdq3-9eqn/about_data

Wikimedia foundation. (2023, December 21). Skewness. Wikipedia. https://en.wikipedia.org/wiki/Skewness

Wikimedia Foundation. (2024, February 29). *Kurtosis*. Wikipedia. https://en.wikipedia.org/wiki/Kurtosis