City Weather Explorer Mining Project

Abigail Burns

The University of Texas at Dallas

EPPS 6323 Knowledge Mining

Spring 2025

18 May 2025

Abstract

A number of major United States cities are predicted to experience increasingly severe weather as a result of climate change and the geographic characteristics which heighten their vulnerability to it (e.g. high population, ocean proximity). These changes are significant enough to impact commerce, lifestyle choices, and intra-national migration.

As an avid traveler and climate advocate, I chose to develop an interactive application for weather insights for three U.S. cities expected to have worsening climate. Weather data and news were mined for each city and subsequently visualized. A shiny application wrapped these elements together to present weather data and news for Los Angeles, Orlando, and New Orleans over the past 50 years.

Data and Methods

I began by collecting weather data from the Wunderground website for the three cities to include temperature, wind, and precipitation metrics over 50 years (*Weather History*). Scraping was accomplished with Selenium; preprocessing with pandas organized the data into spreadsheets. Because the scraping process required many workarounds for timeouts and errors along with tedious pivoting and formatting steps, I consulted ChatGPT to define the Python scraping functions for this step (OpenAI).

To supplement the weather information, I parsed 100 “severe weather” Google News RSS feeds from 2024 and 2025 for each city and loaded them into Excel files using tidyRSS (*Google News*). Packages quanteda, tm, and syuzhet were applied to produce frequency-based word clouds and sentiment analysis (Mhatre, S.).

I then programmed the Shiny web app by building on previously written applications and using custom CSS styling (Strayer, *W3Schools.com*). The app uses a sidebar layout featuring

dropdown menus for city and weather variable and slider input for year range (Wickham, H.). Weather data is presented over a time period of up to 50 years using Grammar of Graphics (ggplot2) methods. Textual analysis presents news headline themes and sentiments using a word cloud and bar chart with colors ascribed to match sentiment (red for anger, etc.). These included many climatic keywords describing event type and severity, which expressed primarily fear, anticipation, and anger.

Although word association was biased by the specific inclusion of "severe weather" in the newsfeed search, the results confirm the phenomenon of changing weather and climate vulnerability for the three selected cities alongside quantitative data showing changes and shifts in temperature, precipitation and wind over time. I developed the app in a series of modules before combining them in Shiny, encountering minimal issues with the final deployments such as missing titles, unloaded libraries and visually unbalanced elements.

Conclusions

The quantitative and qualitative analysis presented in this project has the advantage of being easily modified and recollected for new input data and text. However, they could be expanded to include more and better visualizations. One particularly admirable program for insightful weather graphs and charts is hosted by Weather Spark (*New Orleans Climate*).

This knowledge mining project produces practical and useful implementation of graphical and NLP methods. Developed content presents climate characterization and changes for three particularly volatile U.S. cities, to be applied iteratively and for any number of locations in the future. I recommend the addition of 17 U.S. cities to complete the list of the top ten best and worst U.S. cities for climate change as identified by Sachon and Howard (2022).

References

Google News RSS Feed XML. (n.d.). Google News. https://news.google.com/rss/search?q=Los%20Angeles%20severe%20weather&hl=en-US&gl=US&ceid=US:en

Mhatre, S. (2024, August 30). *Text Mining and Sentiment Analysis: Analysis with R*. Simple Talk. https://www.red-gate.com/simple-talk/databases/sql-server/bi-sql-server/text-mining-and-sentiment-analysis-with-r/

OpenAI. (2025). ChatGPT (version 4o) [Large language model]. https://chat.openai.com

New Orleans Climate, weather by month, Average temperature (Louisiana, United States) - Weather Spark. (n.d.). Weather Spark. https://weatherspark.com/y/11799/Average-Weather-in-New-Orleans-Louisiana-United-States-Year-Round

Sachon, L., & Howard, P. (2022, December 22). *Best & worst cities for climate change*. https://www.policygenius.com/homeowners-insurance/best-and-worst-cities-climate-change/

Strayer, N. (n.d.). *Using custom CSS in your app.* Shiny. https://shiny.posit.co/r/articles/build/css/

W3Schools.com. (n.d.). https://www.w3schools.com/cssref/css_colors.php

Weather History | Weather underground. (n.d.). https://www.wunderground.com/history/monthly/us/ca/los-angeles/KCALOSAN1265/date/1975-3

Wickham, H. (n.d.). *Chapter 6 Layout, themes, HTML | Mastering Shiny*. https://mastering-shiny.org/action-layout.html

Code Appendix

```
library(shiny)
library(shinyWidgets)
library(bslib)
library(readxl)
library(quanteda)
library(quanteda.textmodels)
library(quanteda.textplots)
library(wordcloud)
library(tm)
library(syuzhet)
library(ggplot2)
library(dplyr)
library(lubridate)
library(tidyr)
library(zoo)

cities <- c("los-angeles"="Los Angeles",
              "orlando"="Orlando",
              "new-orleans"="New Orleans")

remove <- c("los","angeles","california",
            "orlando","florida","new","orleans",
            "louisiana","l.a","weather","today",
            "fox","wesh","wftv","wkmg","nbc","usa",
            "wwltv","wdsu","nola","times","newsroom",
            "wgno","entergy","wwltv.com",
            "fox8live.com","abc7","news","fox8live",
            "11","13","35","cnn","national","service",
            "gov","channel","eastern","southern","south","central")

# Define UIs for application
ui <- fluidPage(
  tags$head(
    # Note the wrapping of the string in HTML()
    tags$style(HTML("
      @import url('https://fonts.googleapis.com/css2?family=Roboto');
      body {
        background-color: #E4E4E4;
        color: blue
      }
      h2 {
        font-family: 'Roboto', sans-serif
      }
      .shiny-input-container {
        color: darkorchid
      }"))
  ),
  titlePanel("City Weather Explorer"),
  sidebarLayout(
    sidebarPanel(
      selectInput("city", "Select City", choices = c("los-angeles",
"orlando", "new-orleans")),
      selectInput("variable", "Select Variable", choices = c(
        "Avg_Temperature_Average",
```

```
        "Avg_Temperature_Max",
        "Avg_Temperature_Min",
        "Max_Temperature_Average",
        "Max_Temperature_Max",
        "Max_Temperature_Min",
        "Min_Temperature_Average",
        "Min_Temperature_Max",
        "Min_Temperature_Min",
        "Dew_Point_Average",
        "Dew_Point_Max",
        "Dew_Point_Min",
        "Precipitation_Average",
        "Precipitation_Max",
        "Precipitation_Min",
        "Wind_Average",
        "Wind_Max",
        "Wind_Min",
        "Gust_Wind_Average",
        "Gust_Wind_Max",
        "Gust_Wind_Min")),
      checkboxInput("show_anomaly", "Show Anomalies (Z-scores)", FALSE),
      chooseSliderSkin("Round"),
      sliderInput("years", "Year Range", sep = "",
                  min = 1975, max = 2025, value = c(1975, 2025), step = 1),
      plotOutput("wordcloud"),
      width=4
    ),
    mainPanel(
      plotOutput("weatherPlot"),
      plotOutput("barplot")
    )
  )
)

server <- function(input, output, session) {
  # Reactively load the weather data .xlsx file for selected city
  city_data <- reactive({
    file_path <- paste0(input$city, "_weather.xlsx")
    df <- read_excel(file_path)
    df$Date <- as.yearmon(df$Date)  # convert to yearmon
    df
  })
  # Filtered data based on year range
  filtered_data <- reactive({
    req(city_data())
    city_data() %>%
      filter(year(Date) >= input$years[1],
             year(Date) <= input$years[2])
  })
  output$weatherPlot <- renderPlot({
    req(input$variable)
    df <- filtered_data()
    if (input$show_anomaly) {
      df <- df %>%
        mutate(Month = month(Date),
               Year = year(Date)) %>%
        group_by(Month) %>%
```

```
        mutate(
          Month_Mean = mean(.data[[input$variable]], na.rm = TRUE),
          Anomaly = .data[[input$variable]] - Month_Mean,
          Month_SD = sd(.data[[input$variable]], na.rm = TRUE),
          Z = Anomaly / Month_SD
        ) %>%
        ungroup()

      ggplot(df, aes(x = Date, y = Z)) +
        geom_col(fill = "tomato") +
        labs(title = paste("Anomalies for", input$variable, "in",
cities[input$city]),
             y = "Anomaly from Monthly Mean", x = NULL)
    } else {
      ggplot(df, aes(x = Date, y = .data[[input$variable]])) +
        geom_line(color = "blue") +
        geom_point() +
        labs(title = paste(input$variable, "over Time in",
cities[input$city]),
             y = input$variable, x = NULL)
    }
  })
  # Reactively load the news headlines .xlsx file for selected city
  output$wordcloud <- renderPlot({
    req(input$city)
    # Quanteda corpus
    corp <- corpus(
      x = read_excel(paste0(input$city, "_gnews.xlsx")),
      docid_field = "item_link",
      text_field = "item_title",
      meta = list(),
      unique_docnames = TRUE
    )
    # Wordcloud
    dfm <- corp %>%
      tokens(remove_punct = TRUE) %>%
      tokens_remove(stopwords("english")) %>%
      tokens_remove(remove) %>%
      dfm() %>%
      dfm_trim(min_termfreq = 2, verbose = FALSE)
    my_palette <- c("blueviolet","blue","chartreuse","red")
    set.seed(100)
    wordcloud <- textplot_wordcloud(dfm, min_size=1,
                                    max_size=4, color=my_palette)
  })
  # Reactively load the news headlines .xlsx file for selected city
  output$barplot <- renderPlot({
    req(input$city)
    # tm corpus
    x <- read_excel(paste0(input$city, "_gnews.xlsx"))
    docs <- data.frame(
      x,
      doc_id=x$item_link,
      text=x$item_title)
    ds <- DataframeSource(docs)
    words.corpus <- Corpus(ds)
    # Turn all words to lower case
```

```
    words.corpus <- tm_map(words.corpus, content_transformer(tolower))
    # Remove punctuations, numbers, common terms
    words.corpus <- tm_map(words.corpus, removePunctuation)
    words.corpus <- tm_map(words.corpus, removeNumbers)
    words.corpus <- tm_map(words.corpus, removeWords,
                           remove)
    # Remove stopwords, then create uniform bag of words
    words.corpus <- tm_map(words.corpus, removeWords, stopwords("english"))
    # Create Term Document Matrix
    tdm <- TermDocumentMatrix(words.corpus)
    m <- as.matrix(tdm)
    wordCounts <- rowSums(m)
    wordCounts <- sort(wordCounts, decreasing=TRUE)
    # Create Wordcloud using wordcloud package
    cloudFrame <- data.frame(word=names(wordCounts),freq=wordCounts)
    # Sentiment classification
    d <- get_nrc_sentiment(cloudFrame$word)
    # transpose
    td<-data.frame(t(d))
    # rowSums computes column sums across rows for each level of a grouping
variable.
    td_new <- data.frame(rowSums(td))
    # Transformation and cleaning
    names(td_new)[1] <- "count"
    td_new <- cbind("sentiment" = rownames(td_new), td_new)
    rownames(td_new) <- NULL
    td_new2<-td_new[1:8,]
    # Plot count of words associated with each sentiment
    ggplot(data=td_new2,
           mapping = aes(x=sentiment, y=count, fill=sentiment)) +
      geom_bar(stat='identity') +
      scale_fill_manual(values=c("red",
                                 "darkviolet",
                                 "#7cae00",
                                 "gray20",
                                 "#ff61cc",
                                 "darkblue",
                                 "orange",
                                 "#008b4b")) +
      labs(title = paste("'Severe Weather' News Sentiment for",
cities[input$city]))
  })
}

shinyApp(ui = ui, server = server)
```

AI Appendix

Prompts	Prompt Type	Tool Used	Response

➢ "Can you give an example of using Selenium with data from https://www.wunderground.com/history/daily/us/tx/dallas/KDAL ?"
➢ "Can you modify the script for R packages, and scrape multiple dates?"
➢ "Let's see this wrapped into a function."
➢ "Let's switch to Selenium + Python."
➢ "It returned "No data found."""
	Script Generation	ChatGPT	Got it — if the script returned **"No data found"**, then **Selenium likely reached the page**, but failed to **find or extract the weather table rows**.
			This is common with **modern dynamic sites** like Wunderground. Let's debug and fix it step by step.
			
			✅ **Likely Reasons & Solutions**
			🔷 **1. The page layout changed or data is inside Shadow DOM**
			Wunderground **wraps key content inside Shadow DOM** — a modern HTML feature that **Selenium can't access directly** with `find_element(By.XPATH)`.
			👉 **Solution:** We'll use **JavaScript execution** to extract from the Shadow DOM.
			
			✅🔁 **Updated Working Solution Using Shadow DOM Access**
			Let's revise your script to access the Shadow DOM via JavaScript:
			⚠️ **Before You Run: Please keep a real browser open to monitor whether you're getting blocked by CAPTCHA**
			
			🧪 **New Python Script with JavaScript Shadow DOM Access**
			`import time`
			`import pandas as pd`
			`from selenium import webdriver`
			`from selenium.webdriver.chrome.service import Service`
			`from selenium.webdriver.chrome.options import Options`

			from webdriver_manager.chrome import ChromeDriverManager
			from datetime import datetime, timedelta
			
			def init_driver():
			    options = Options()
			    options.add_argument("--headless=new") # modern headless mode
			    options.add_argument("--disable-blink-features=AutomationControlled")
			    options.add_argument("--disable-gpu")
			    options.add_argument("--no-sandbox")
			    options.add_argument("--window-size=1920x1080")
			    service = Service(ChromeDriverManager().install())
			    driver = webdriver.Chrome(service=service, options=options)
			    return driver
			
			def extract_table_data(driver):
			    # Run JS to pierce Shadow DOM and grab the table
			    script = """
			    const table = document.querySelector("lib-city-history-observation");
			    if (!table) return '';
			    const shadowRoot = table.shadowRoot;
			    const rows = shadowRoot.querySelectorAll("table tbody tr");
			    return Array.from(rows).map(row =>
			
			Array.from(row.querySelectorAll("td")).map(cell => cell.innerText.trim())
			    );
			    """
			    return driver.execute_script(script)
			
			def scrape_weather_for_date(driver, date_str, city="dallas", state="tx", station="KDAL"):
			    url = f"https://www.wunderground.com/history/daily/us/{state}/{city}/{station}/date/{date_str}"

```
    print(f"Scraping: {date_str}")
    driver.get(url)
    time.sleep(8)  # wait for full JS load

    try:
        rows = extract_table_data(driver)
        daily_data = [[date_str] + row for row in
rows if row]
        return daily_data
    except Exception as e:
        print(f"Error on {date_str}: {e}")
        return []

def scrape_range(start_date, end_date,
city="dallas", state="tx", station="KDAL"):
    driver = init_driver()
    all_data = []
    current_date = datetime.strptime(start_date,
"%Y-%m-%d")
    end_date_obj = datetime.strptime(end_date, "%Y-
%m-%d")

    while current_date <= end_date_obj:
        date_str = current_date.strftime("%Y-%m-%d")
        daily_data = scrape_weather_for_date(driver,
date_str, city, state, station)
        all_data.extend(daily_data)
        current_date += timedelta(days=1)

    driver.quit()

    if all_data:
        # infer column count and fill as needed
        max_cols = max(len(row) for row in all_data)
        columns = ["Date"] + [f"Col{i}" for i in
range(1, max_cols)]
        df = pd.DataFrame(all_data,
columns=columns[:len(all_data[0])])
        return df
    else:
        print("No data found.")
```

			`return pd.DataFrame()`
			
			`# Run scraper`
			`df_weather = scrape_range("2023-03-01", "2023-03-03")`
			`print(df_weather.head())`

			✅ **Output** You should now get a **populated DataFrame** with columns like: **Date Time Temp Dew Point Humidity Wind ...**
			🧠 **Tips** • Shadow DOM requires **JavaScript DOM traversal**, not `.find_element()` • Consider adding `random.uniform(6, 10)` wait times to **avoid rate-limiting** • If you still get blocked, try rotating user agents or using undetected-chromedriver
➢ "I would like to pivot into wide format."	Script Generation	ChatGPT	Perfect! Pivoting your cleaned weather summary data into **wide format** will make it easier to: • Run time series or panel analysis • Feed into ML models (e.g., forecasting, clustering) • Visualize with one row per date and columns like `Temperature_High`, `Humidity_Avg`, etc.
			✅ **Starting Point: Your Current Data** Your scraped data is in **long format**: **Date Section Metric Value** 1975-03-23 Temperature High 56 °F 1975-03-23 Temperature Low 34 °F 1975-03-23 Wind Speed 8 mph

✅ **Goal: Pivot to Wide Format**

We want:

Date	Temperature_High	Temperature_Low	Wind_Spe
1975-03-23	56	34	8

🔁 **Step 1: Create a Wide Format Table**

Here's a function that does the pivot **after scraping**:

```
def pivot_weather_summary(df_long):
    # Create a combined column: Section_Metric
(e.g., Temperature_High)
    df_long["Variable"] =
df_long["Section"].str.replace(r"\s+", "_",
regex=True) + "_" + \

df_long["Metric"].str.replace(r"\s+", "_",
regex=True)

    # Remove units from Value column (optional: keep
only numeric part)
    df_long["Cleaned_Value"] =
df_long["Value"].str.extract(r"([-+]?[0-9]*\.?[0-
9]+)")

    # Pivot the data
    df_wide = df_long.pivot_table(index="Date",

columns="Variable",

values="Cleaned_Value",

aggfunc="first").reset_index()

    return df_wide
```

🧪 **Example Usage:**

```
# After scraping:
df_long = scrape_range("1975-03-23", "1976-03-22",
headless=True)
```

```
# Pivot to wide format
df_wide = pivot_weather_summary(df_long)

# Preview
print(df_wide.head())

# Save
df_wide.to_csv("wunderground_weather_pivoted.csv",
index=False)
```

✅ **Output Format**

Date	Temperature_High	Temperature_Low	Wind_Spe
1975-03-23	56	34	8

- All values are **strings by default**, but you can convert to `float` with:

```
df_wide = df_wide.astype({col: float for col in
df_wide.columns if col != 'Date'})
```