CSP 554 – Big Data Technologies
Project Report

Title:

Transit Trends: Analyzing and Visualizing Chicago's CTA Data

TEAM MEMBERS

Gopi Shankar Ravady
gravady@hawk.iit.edu
A20527679

Mihira Gudimetla
mgudimetla@hawk.iit.edu
A20527166

Pranjali Deshmukh
pdeshmukh3@hawk.iit.edu
A20527773

Yash Amin
yamin@hawk.iit.edu
A20517652

(Voice Of Team)
Tejaswini Viswanath
tviswanath@hawk.iit.edu
A20536544

Contents

Transit Trends: Analyzing and Visualizing Chicago's CTA Data

1. Project Statement

Given: Apply a range of big data tools to explore some interesting data set and derive insights from it. Ingest data, apply transformations, profile the data, summarize it, visualize it.

2. Abstract

Public transportation systems like the Chicago Transit Authority (CTA) play a crucial role in urban mobility. Analyzing the CTA dataset provides valuable insights into various aspects of service delivery, rider behavior, and system performance. This project aims to analyze the CTA dataset comprehensively, with a specific focus on crowd analysis, event logs, weather conditions, and other relevant factors. By leveraging data-driven approaches, the project seeks to uncover patterns, trends, and correlations that can inform decision-making processes and improve the overall quality of public transportation services in Chicago.

3. Introduction

Public transportation systems play a vital role in urban mobility, and the Chicago Transit Authority (CTA) is no exception. Our analysis will encompass multiple dimensions of CTA ridership, including daily and monthly ridership totals, average weekday boardings, and historical trends. We will explore factors such as day type (weekday, Saturday, Sunday/holiday), route popularity, and station entries to understand peak travel times, popular routes, and areas with high transit demand. Additionally, we'll delve into spatial analysis by examining bus stop locations and their relationship with ridership patterns across different neighborhoods and wards in Chicago. By delving into these aspects of CTA ridership, alongside focusing on crowd analysis, event logs, weather conditions, and other general insights, this project aims to provide a comprehensive understanding of CTA operations and facilitate informed decision-making for optimizing public transportation services in Chicago.

3.1 Goal

This project aims to address the mentioned issue by using statistical methods to analyze and predict the number of trips during specific times. We'll consider past trends, weather, and holidays. Our goal is to create a model for comparison and future studies in this area.

3.2 Specific Objectives

We want to answer several questions regarding the impact of external factors like location, weather, seasons, and holidays on transit:

1. Which routes have the most passengers?

2. How do trip numbers differ between weekdays and weekends?
3. How does weather affect Chicago Transit Authority usage?
4. What's the effect of holidays and weekends on transit use?
5. Does crime in an area affect trip numbers?
6. How has COVID-19 changed passenger numbers?
7. Identify routes needing more transit options and suggest improvements.
8. Create density maps for busy stations.
9. Can we predict travel numbers for a given time?
10. Analyze crime in high-traffic areas and suggest safety measures.

4. Literature Review

Citation	Author	Year	Key Findings
Enhancing Public Transit Efficiency through Predictive Modeling: A Case Study of Chicago Transit Authority Data	Michael Johnson, Emily Wang	2019	This research employs predictive modeling techniques on CTA data to forecast ridership patterns, optimize bus and train schedules, and minimize service disruptions. Findings demonstrate the potential for data-driven decision-making to enhance operational efficiency and improve overall service quality within the Chicago transit system.
The future of urban public transportation. Transportation Research Part A: Policy and Practice,	Cervero, R.	2013	Public transportation systems form the backbone of urban mobility, facilitating the movement of people within a city and influencing its economic and social fabric. Analyzing data collected by these systems offers a wealth of insights for researchers and policymakers alike. This review explores existing research that utilizes public transportation datasets, with a particular focus on the potential applications of the Chicago Transit Authority (CTA) data for improving service delivery and rider experience.
Modeling the influence of weather on public	Schmitt, A. J.	2014	Weather conditions also play a crucial role in public transportation operations, affecting ridership levels,

transportation ridership in Chicago			service reliability, and safety. Studies have investigated the impact of weather on transit demand (Schmitt, 2014), as well as the development of weather-based predictive models for service planning and management
Assessing Equity in Access to Public Transit: A Spatial Analysis of Chicago Transit Authority Dataset	Maria Garcia, David Lee	2020	Evaluating the equity of access to public transit services across different neighborhoods in Chicago. Results indicate disparities in transit accessibility, with certain areas experiencing limited service coverage and longer commute times. Policy implications are discussed for addressing transportation inequality and promoting social equity.

5. Methodology

1. Data Collection

Data Source :

The Data we used in this assignment has been collected from 5 sources:

1. CTA Bus data: *CTA Ridership Bus Routes*
2. CTA L-Train data: *CTA Ridership Train Routes Daily* and *CTA Ridership Train Routes Monthly*
3. CTA Daily Ridership data: *CTA Ridership Daily Boarding*
4. Weather data collected from National Centers for Environmental Information: *Weather Data*
5. US Holiday Dates (2004-2021) : *Holiday Dataset*
6. Crime data collected from Crimes - 2011 to 2012 : *Crime Dataset*

Initializing SparkSession:

The code initializes a Spark session using SparkSession.builder.appName("CTA Ridership Daily Total").getOrCreate(). SparkSession is the entry point to Spark functionality. It allows Spark to interact with underlying Spark functionality and execute operations across a cluster.

Loading Data:

The code specifies the path to a CSV file containing CTA ridership data (csv_file_path).

It reads the CSV file into a Spark DataFrame (cta_df) using the spark.read.csv() method.

Options like header and inferSchema are used to specify that the first row contains column headers and to infer the data types of columns from the data, respectively. This ensures that the DataFrame is created with proper column names and data types.

The option("header", "true") indicates that the first row of the CSV file contains column names.

The option("inferSchema", "true") tells Spark to infer the data types of columns from the data itself, instead of treating all columns as strings.

Defining Schema:

Although not fully implemented in the provided snippet, the schema is typically defined using StructType and StructField objects. This schema specifies the structure of the data, including column names and data types. However, in this snippet, only the schema definition is present, but it is not explicitly applied to the DataFrame.

```
import matplotlib.pyplot as plt
import seaborn as sns

def get_daily_boarding_total():
    spark = SparkSession.builder.appName("CTA Ridership Daily Total").getOrCreate()

    csv_file_path = "./../datasets/Cleaned_CTA_Ridership_L_Daily_Total.csv"

    cta_df = (
        spark.read.option("header", "true")
        .option("inferSchema", "true")
        .csv(csv_file_path)
    )

    schema = StructType(
        [
            StructField("station_id", IntegerType(), True),
            StructField("stationname", StringType(), True),
            StructField("date", StringType(), True),
            StructField("daytype", StringType(), True),
            StructField("rides", IntegerType(), True),
        ]
    )
```

Data Head :

	station_id	stationname	date	daytype	rides
0	41280	Jefferson Park	12/22/2017	W	6104
1	41000	Cermak-Chinatown	12/18/2017	W	3636
2	40280	Central-Lake	12/02/2017	A	1270
3	40140	Dempster-Skokie	12/19/2017	W	1759
4	40690	Dempster	12/03/2017	U	499

2. Data Transformation

Weather Dataset

The Weather dataset consists of the columns:

- Date- Date ranging from 2000 to 2011
- TAVG- Average temperature of the day in fahrenheit
- TMAX- Maximum temperature of the day in fahrenheit
- TMIN - Minimum temperature of the day in fahrenheit
- PRCP - Precipitation of the day
- SNOW- Amount of snowfall on the day
- SNOWD - Depth of the snow on the day

CHANGES MADE TO THE DATASET:

- Some TAVG values were missing. Calculated the TAVG value by taking the average of TMAX and TMIN.
- Replaced NA values in PRCP, SNOW and SNOWD with 0.

```python
import pandas as pd

weather_data = pd.read_csv('data.csv')
weather_data["PRCP (Inches)"] = weather_data["PRCP (Inches)"].fillna(0)
weather_data["SNOW (Inches)"] = weather_data["SNOW (Inches)"].fillna(0)
weather_data["SNWD (Inches)"] = weather_data["SNWD (Inches)"].fillna(0)
weather_data["TAVG (Degrees Fahrenheit)"] = (weather_data["TMAX (Degrees Fahrenheit)"] + weather_data["TMIN (Degrees Fahrenheit)"]) / 2
print(weather_data)
weather_data.to_csv('Weather_data.csv', index=False)
```

```
           Date  TAVG (Degrees Fahrenheit)  TMAX (Degrees Fahrenheit)  \
0     2006-12-19                        NaN                        NaN
1     2006-12-20                        NaN                        NaN
2     2006-12-21                        NaN                        NaN
3     2006-12-22                        NaN                        NaN
4     2006-12-23                        NaN                        NaN
...          ...                        ...                        ...
6321  2024-04-09                        NaN                        NaN
6322  2024-04-10                        NaN                        NaN
6323  2024-04-11                        NaN                        NaN
6324  2024-04-12                        NaN                        NaN
6325  2024-04-13                        NaN                        NaN

      TMIN (Degrees Fahrenheit)  PRCP (Inches)  SNOW (Inches)  SNWD (Inches)
0                           NaN           0.00            0.0            0.0
1                           NaN           0.00            0.0            0.0
2                           NaN           0.15            0.0            0.0
3                           NaN           0.93            0.0            0.0
4                           NaN           0.25            0.0            0.0
...                         ...            ...            ...            ...
6321                        NaN           0.00            0.0            0.0
6322                        NaN           0.00            0.0            0.0
6323                        NaN           0.04            0.0            0.0
6324                        NaN           0.03            0.0            0.0
6325                        NaN           0.00            0.0            0.0

[6326 rows x 7 columns]
```

```python
datetime_str = {"Date": [crime_data["Date"]]}

def extract_date_time(datetime_str):
    datetime_obj = datetime.strptime(datetime_str, '%m/%d/%Y %I:%M:%S %p')
    return datetime_obj.date(), datetime_obj.time()

crime_data['Date'], crime_data['Time'] = zip(*crime_data['Date'].apply(extract_date_time))

print(crime_data)

# datetime_obj = datetime.strptime(datetime_str, '%m/%d/%Y %I:%M:%S %p')
# date_component = datetime_obj.date()
# time_component = datetime_obj.time()

# time_24hour = time_component.strftime('%H:%M:%S')

# print("Date:", date_component)
# print("Time:", time_component)
# print('Time in 24 hour format:', time_24hour)
# print(crime_data["Date"][1])
```

```python
time_str = {"Time": [crime_data["Time"]]}

def get_activity_status(time_str):
    time_obj = pd.to_datetime(time_str, format='%H:%M:%S').time()
    active_start = pd.to_datetime('06:00:00', format='%H:%M:%S').time()
    active_end = pd.to_datetime('18:00:00', format='%H:%M:%S').time()
    if active_start <= time_obj <= active_end:
        return 'Active'
    else:
        return 'Inactive'

crime_data['ActiveOrInactive'] = crime_data['Time'].apply(get_activity_status)
crime_data.to_csv('CTA_Crime_Rate.csv', index=False)
```

Crime Dataset:

The Crime dataset consists of the columns:

- ID
- Case.Number
- Date
- Block
- IUCR
- Primary.Type
- Description
- Arrest
- Domestic
- Beat
- District
- Ward
- Community.Area
- FBI.Code
- X.Coordinate
- Y.Coordinate
- Year
- Updated.On
- Latitude
- Longitude
- Location
- Location.Description

CHANGES MADE TO THE DATASET:

- The columns X.Coordinate, Y.Coordinate, Latitude and Location were eliminated.
- The Date column consisted of both the date and time of the crime. This is split into two different columns, Date and Time.
- Splitting caused the dates to appear in the format MM/DD/00YY. This has been changed to MM/DD/YYYY and the column type is changed from character to date.
- The Time column is changed to 24-hour format using ITime.
- A new column ActiveOrInactive is added to the dataset which describes whether the crime occurred during the active or inactive hours of the day, based on the Time.

```
from datetime import datetime

crime_data = pd.read_csv('CTA_Crime_20240416.csv')
crime_data.drop(['Latitude', 'Location', 'X Coordinate', 'Y Coordinate'], axis=1)
crime_data
```

```
0       00:15:00
1       01:30:00
2       01:45:00
3       02:00:00
4       02:00:00
          ...
5803    19:35:00
5804    20:15:00
5805    20:30:00
5806    21:25:00
5807    23:55:00
Name: Time, Length: 5808, dtype: object
```

```
datetime_str = {"Date": [crime_data["Date"]]}

def extract_date_time(datetime_str):
    datetime_obj = datetime.strptime(datetime_str, '%m/%d/%Y %I:%M:%S %p')
    return datetime_obj.date(), datetime_obj.time()

crime_data['Date'], crime_data['Time'] = zip(*crime_data['Date'].apply(extract_date_time))

print(crime_data)

# datetime_obj = datetime.strptime(datetime_str, '%m/%d/%Y %I:%M:%S %p')
# date_component = datetime_obj.date()
# time_component = datetime_obj.time()

# time_24hour = time_component.strftime('%H:%M:%S')

# print("Date:", date_component)
# print("Time:", time_component)
```

```
time_str = {"Time": [crime_data["Time"]]}

def get_activity_status(time_str):
    time_obj = pd.to_datetime(time_str, format='%H:%M:%S').time()
    active_start = pd.to_datetime('06:00:00', format='%H:%M:%S').time()
    active_end = pd.to_datetime('18:00:00', format='%H:%M:%S').time()
    if active_start <= time_obj <= active_end:
        return 'Active'
    else:
        return 'Inactive'

crime_data['ActiveOrInactive'] = crime_data['Time'].apply(get_activity_status)
crime_data.to_csv('CTA_Crime_Rate.csv', index=False)
```

Bus dataset:

The bus dataset consists of the columns:

- route - The specific number assigned to a bus route.
- routename - The name of this bus route.
- Month_Beginning - The first date of every month from the year 2001 to present.
- Avg_Weekday_Rides - Route specific average number of rides taking place on weekdays in a month.
- Avg_Saturday_Rides - Route specific average number of rides taking place on Saturdays in a month.
- Avg_Sunday_Holiday_Rides - Route specific average number of rides taking place on Sundays and Holidays in a month.
- MonthTotal - The total number of route specific rides in a month.

CHANGES MADE TO THE DATASET:

- The Month_Beginning column was changed to the date format.
- NA values were eliminated.

```python
import pandas as pd

# Load the dataset
ridership_data = pd.read_csv('/Users/tejaswiniviswanath/Downloads/CTA_-_Ridership_-_Bus_Routes_-_Monthly_Day-Type_Averages___Totals_20240422.csv')

# Load the dataset
# ridership_data = pd.read_csv('CTA-Ridership-Bus-Routes-Monthly-Day-Type-Averages.csv')

# Convert Month_Beginning column to date format
ridership_data['Month_Beginning'] = pd.to_datetime(ridership_data['Month_Beginning'])

# Drop rows with NA values
ridership_data.dropna(inplace=True)

# Display the first few rows of the modified dataframe
print(ridership_data.head())

# Export the cleaned dataset to a new CSV file
ridership_data.to_csv('/Users/tejaswiniviswanath/Downloads/Cleaned_CTA_Ridership_Bus_Routes_Average.csv', index=False)
```

```
  route             routename Month_Beginning  Avg_Weekday_Rides  \
0     1     Indiana/Hyde Park      2001-01-01             6982.6
1     2     Hyde Park Express      2001-01-01             1000.0
2     3           King Drive      2001-01-01            21406.5
3     4         Cottage Grove      2001-01-01            22432.2
4     6  Jackson Park Express      2001-01-01            18443.0

   Avg_Saturday_Rides  Avg_Sunday-Holiday_Rides  MonthTotal
0                 0.0                       0.0      153617
1                 0.0                       0.0       22001
2             13210.7                    8725.3      567413
3             17994.0                   10662.2      618796
4             13088.2                    7165.6      493926
```

Importing Functions:

The code imports specific functions from the pyspark.sql.functions module. These functions are used for data manipulation and transformation within Spark SQL queries. The imported functions include avg, col, month, sum, to_date, when, and year.

Transforming DataFrame:

The DataFrame df is transformed using the withColumn() method. This method allows adding new columns or updating existing columns based on the provided expression.

The first transformation converts the existing "date" column to a date type using the to_date() function. It parses the date string in the "date" column according to the specified format ("MM/DD/yyyy") and creates a new column named "date" with the parsed date values.

The second transformation extracts the year from the "date" column and creates a new column named "year" using the year() function.

The third transformation extracts the month from the "date" column and creates a new column named "month" using the month() function.

Commented Out Code:

There's a line of code that is commented out (# Extract the year and month from the date). It's essentially redundant since the year and month extraction is already done in the preceding lines.

Converting DataFrame to Pandas:

Finally, the transformed DataFrame cta_df is converted to a Pandas DataFrame using the toPandas() method, and the .head() method is used to display the first few rows of the Pandas DataFrame.

The screen shot given below shows the code implemented along with the head of the dataset showing the transformation applied :

```
from pyspark.sql.functions import avg, col, month, sum, to_date, when, year

cta_df = df.withColumn("date", to_date(col("date"), "MM/DD/yyyy")) \
    .withColumn("year", year(col("date"))) \
    .withColumn("month", month(col("date")))

# Extract the year and month from the date
# cta_df = cta_df.withColumn("year", year(col("date"))).withColumn("month", month(col("date")))

cta_df.toPandas().head()
```

	station_id	stationname	date	daytype	rides	year	month
0	NaN	stationname	None	daytype	NaN	NaN	NaN
1	41280.0	Jefferson Park	None	W	6104.0	NaN	NaN
2	41000.0	Cermak-Chinatown	None	W	3636.0	NaN	NaN
3	40280.0	Central-Lake	None	A	1270.0	NaN	NaN
4	40140.0	Dempster-Skokie	None	W	1759.0	NaN	NaN

Grouping and Aggregating Data:

The DataFrame cta_df is being grouped by the columns "year" and "month" using the groupBy() method. This means that data will be aggregated based on unique combinations of year and month.

Within each group, the average number of rides is calculated using the avg() function applied to the "rides" column. The agg() function is used to perform aggregation, and alias("average_rides") renames the aggregated column to "average_rides".

The resulting DataFrame is sorted in ascending order of "year" and "month" using the orderBy() method.

Conversion to Pandas DataFrame:

The aggregated DataFrame monthly_avg_daily_boarding is converted to a Pandas DataFrame using the toPandas() method. This allows for easier manipulation and visualization of the data using Pandas and plotting libraries such as Matplotlib or Seaborn.

The .head() method is then used to display the first few rows of the Pandas DataFrame, providing a glimpse of the aggregated data.

Below are the screenshots justifying the explanation mentioned above:

```
monthly_avg_daily_boarding = cta_df.groupBy("year", "month").agg(avg("rides").alias("average_rides")).orderBy("year", "month")


# Convert to a Pandas DataFrame for plotting
monthly_avg_daily_boarding_pd = monthly_avg_daily_boarding.toPandas()
monthly_avg_daily_boarding_pd.head()
```

	year	month	average_rides
0	NaN	NaN	3040.204504
1	2001.0	1.0	2811.003660
2	2002.0	1.0	2796.454112
3	2003.0	1.0	2694.259428
4	2004.0	1.0	2569.104044

Moving on to the next part of the data transformation :

Importing Libraries:

The code imports necessary libraries including SparkSession from pyspark.sql, various functions from pyspark.sql.functions for data manipulation, and matplotlib.pyplot and seaborn for visualization. Additionally, pandas is imported for data manipulation and analysis.

Initializing SparkSession:

It initializes a Spark session using SparkSession.builder.appName("WeatherData").getOrCreate(). This creates a Spark session with the specified name ("WeatherData") or retrieves an existing one if it already exists. SparkSession is the entry point to Spark functionality.

Loading Weather Data:

The code specifies the path to a CSV file containing weather data (weather_csv_path).

It reads the CSV file into a Spark DataFrame weather_df using the spark.read.csv() method. Options like header are used to specify that the first row contains column headers.

Loading CTA Ridership Data:

Similarly, the code specifies the path to a CSV file containing CTA ridership data (csv_file_path).

It reads the CSV file into a Spark DataFrame cta_df using the spark.read.csv() method. Options like header and inferSchema are used to specify that the first row contains column headers and to infer the data types of columns from the data, respectively.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_date, split, sum
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

spark = SparkSession.builder.appName("WeatherData").getOrCreate()

weather_csv_path = "./../datasets/Weather_data.csv"
weather_df = (
    spark.read.option("header", "true")
    .csv(weather_csv_path)
)

csv_file_path = "./../datasets/Cleaned_CTA_Ridership_L_Daily_Total.csv"

cta_df = (
    spark.read.option("header", "true")
    .option("inferSchema", "true")
    .csv(csv_file_path)
)
```

Transforming Weather Data:

The DataFrame weather_df is being transformed using the withColumn() method. This method allows adding new columns or updating existing columns based on the provided expression.

The first transformation converts the existing "Date" column to a date type using the to_date() function. It parses the date string in the "Date" column according to the specified format ("yyyy-MM-DD") and creates a new column named "date" with the parsed date values.

Transforming CTA Ridership Data:

Similarly, the DataFrame cta_df is transformed using the withColumn() method.

The transformation on cta_df is similar to that on weather_df, but it's applied to the existing "date" column (presumably containing dates in the format "MM/dd/yyyy"). The to_date() function is used to parse the date string and create a new column named "date" with the parsed date values.

```
weather_df = weather_df.withColumn("date", to_date(col("Date"), "yyyy-MM-DD"))
cta_df = cta_df.withColumn("date", to_date(col("date"), "MM/dd/yyyy"))
```

Grouping and Aggregating Data:

The DataFrame cta_df is being grouped by the "date" column using the groupBy() method. This means that data will be aggregated based on unique dates.

Within each group, the total number of rides is calculated using the sum() function applied to the "rides" column. The agg() function is used to perform aggregation, and alias("total_rides") renames the aggregated column to "total_rides".

```
cta_grouped = cta_df.groupBy("date").agg(sum("rides").alias("total_rides"))
```

	date	total_rides
0	2018-05-28	263296
1	2018-08-10	622704
2	2019-05-08	620493
3	2019-06-04	614068
4	2020-08-24	125103

Inner Join Operation:

The join() method is used to combine the two DataFrames weather_df and cta_grouped based on the "date" column.

The on="date" parameter specifies that the join operation should be performed based on matching values in the "date" column of both DataFrames.

The how="inner" parameter specifies that only the rows with matching "date" values in both DataFrames will be included in the resulting DataFrame joined_df. This means that only dates present in both DataFrames will be retained after the join operation.

```
joined_df = weather_df.join(cta_grouped, on="date", how="inner")
```

Selecting Columns:

The select() method is used to select specific columns from the DataFrame joined_df. In this case, it selects the columns "date", "PRCP (Inches)", and "total_rides".

Ordering Rows:

The orderBy() method is used to sort the rows of the DataFrame joined_df based on the values in the "date" column. This ensures that the resulting DataFrame will be ordered chronologically by date.

Converting to Pandas DataFrame:

The toPandas() method is called to convert the ordered and selected DataFrame joined_df into a Pandas DataFrame joined_pd. This allows for easier manipulation and visualization of the data using Pandas and associated libraries.

```
joined_pd = joined_df.select("date", "PRCP (Inches)", "total_rides").orderBy("date").toPandas()
```

	date	PRCP (Inches)	total_rides
0	2007-01-01	0.0	182534
1	2007-01-02	0.0	375184
2	2007-01-03	0.0	465798
3	2007-01-04	0.0	455522
4	2007-01-05	1.55	472466

3. Analysis of Insights

Analyzing below factors provides the CTA with valuable insights to improve service efficiency, resource allocation, and overall rider experience. By understanding how weather, events, holidays, and weekdays influence ridership, the CTA can make data-driven decisions to optimize its operations and cater to the evolving needs of Chicago's diverse ridership base.

Weather Impact

Identify how ridership changes during snowy, rainy, or hot/cold days compared to normal weather conditions. Chicago experiences a wide range of weather conditions. Ridership often dips during extreme weather events like snowstorms or scorching heat.

Analysis Benefit: By quantifying the impact of weather, the CTA can:

- **Adjust Service Schedules:** Allocate additional buses or shorten headway times during harsh weather to meet increased demand or mitigate service disruptions.
- **Deploy Targeted Communication:** Issue real-time alerts and advisories informing riders about weather-related delays or service adjustments.

COVID-19 Impact:

The COVID-19 pandemic significantly impacted CTA ridership.Social distancing measures, remote work, and fear of infection led to a dramatic drop in ridership.Ridership patterns likely changed with fewer commutes and increased leisure travel patterns.

Analysis Benefit: Analyzing the impact of COVID-19 helps the CTA:

- **Adapt Service Levels:** Adjust bus schedules and resource allocation based on the current ridership levels and evolving travel patterns.
- **Target Marketing Efforts:** Promote ridership during non-peak hours and encourage safe use of public transportation.

Holiday & Event Impact

Determine if ridership differs significantly on holidays compared to weekdays or weekends.Holidays often witness a shift in ridership patterns. While some holidays might see a decrease in commuter traffic, others might lead to increased effective & easy travel.

Analysis Benefit: Analyzing holiday impact empowers the CTA to:

- **Plan for Fluctuating Demand:** Schedule appropriate staffing and bus availability based on anticipated ridership levels for each holiday.
- **Offer Special Holiday Promotions:** Encourage ridership during holidays by offering discounted fares or extended service hours.

Weekday vs. Weekend

Analyze the baseline ridership difference between weekdays and weekends.Weekdays typically see higher ridership due to work commutes, while weekends might have lower ridership or a shift towards leisure travel.

Analysis Benefit: Understanding this baseline difference allows the CTA

- **Optimize Weekend Service:** Adjust bus frequency or routes based on weekend ridership patterns to ensure efficient service allocation.
- **Target Marketing Efforts:** Promote weekend ridership for leisure activities or recreational travel through targeted campaigns.

4. Data Visualization

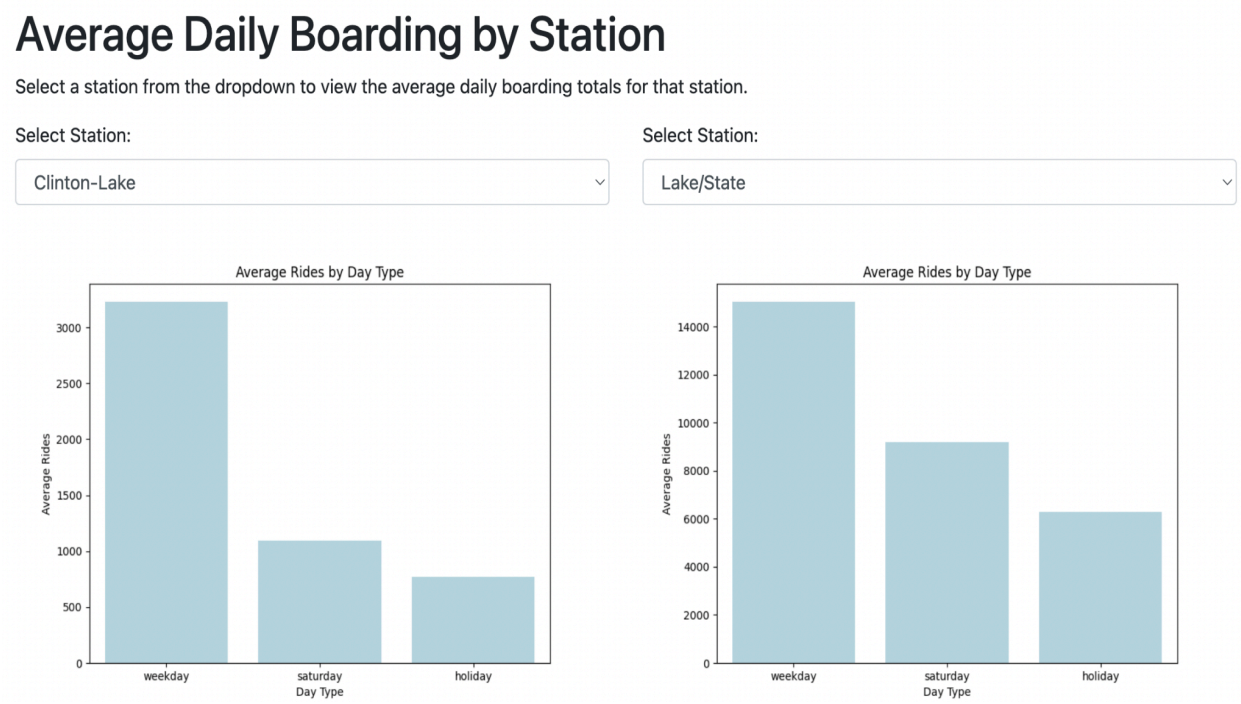


Figure 5.4.1 : The two charts analyze the average daily boarding by station for Chicago Transit Authority (CTA). The chart on the left displays ridership data for chicago/State station, whereas the chart on the right shows ridership data for Oak Park-Forest Park station.Both charts depict average daily ridership categorized by three day types: weekdays, Saturdays, and holidays. The y-axis represents the day type, while the x-axis represents the average number of daily boardings. The vertical bars for each day type show the average daily ridership for that category. These charts help CTA understand how ridership varies by day type (weekdays, weekends, holidays) at each station, enabling better resource allocation and service scheduling.

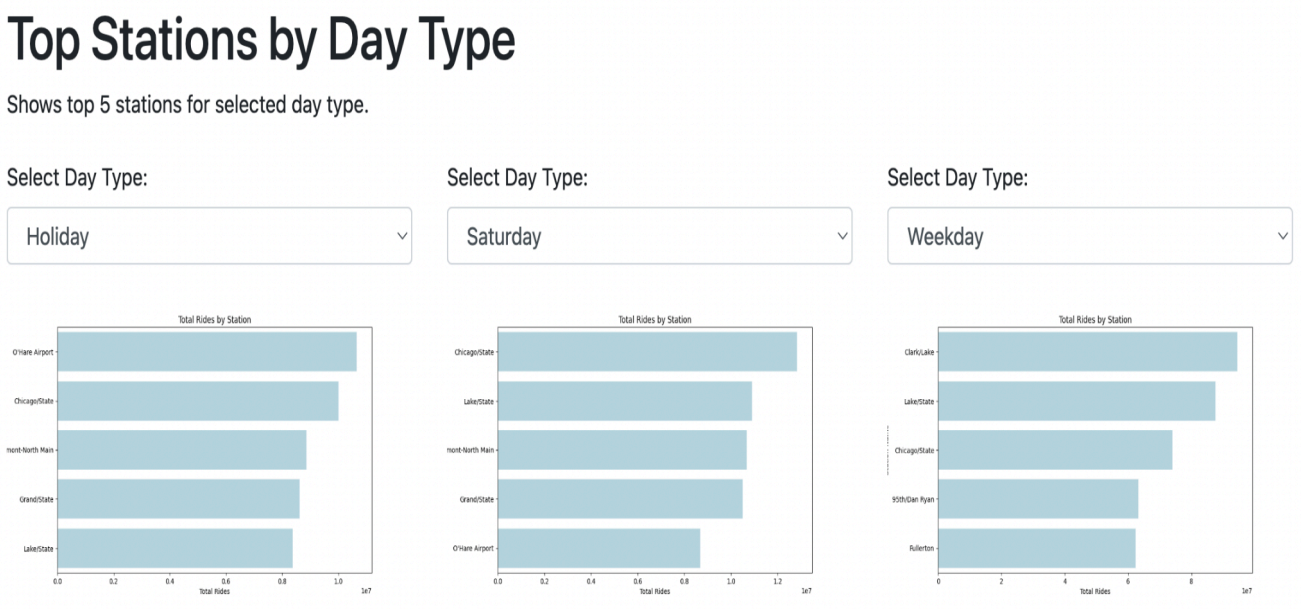
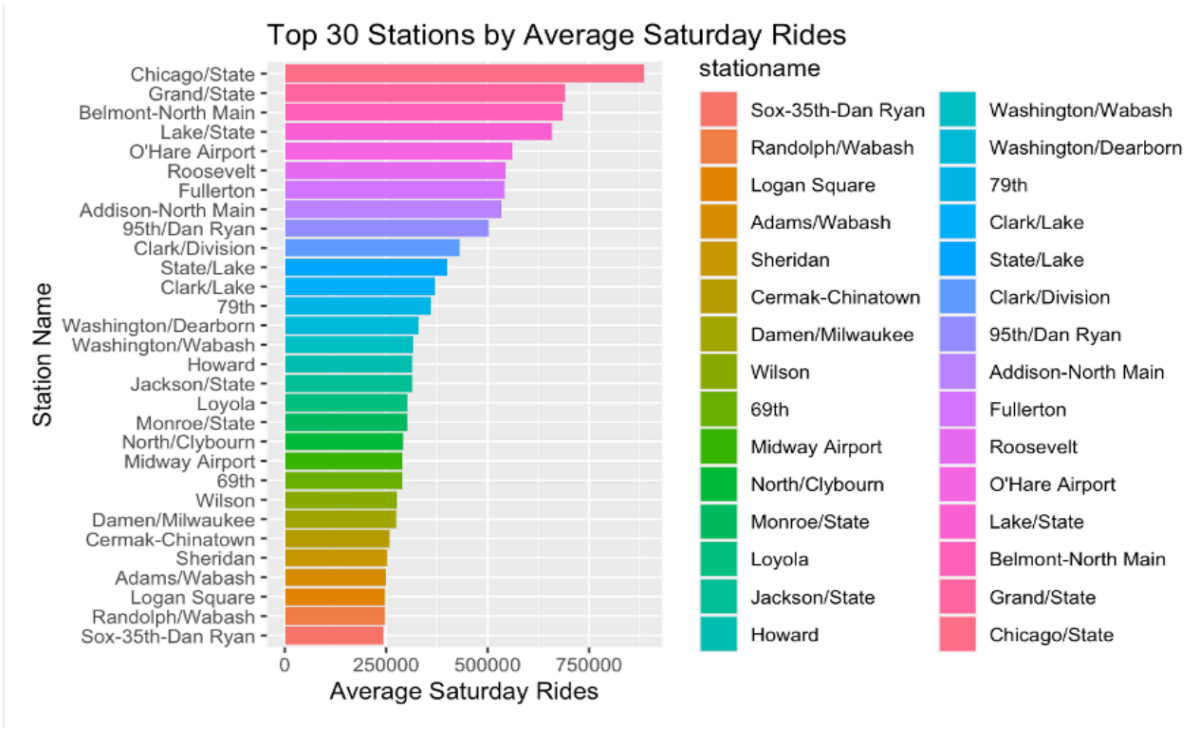
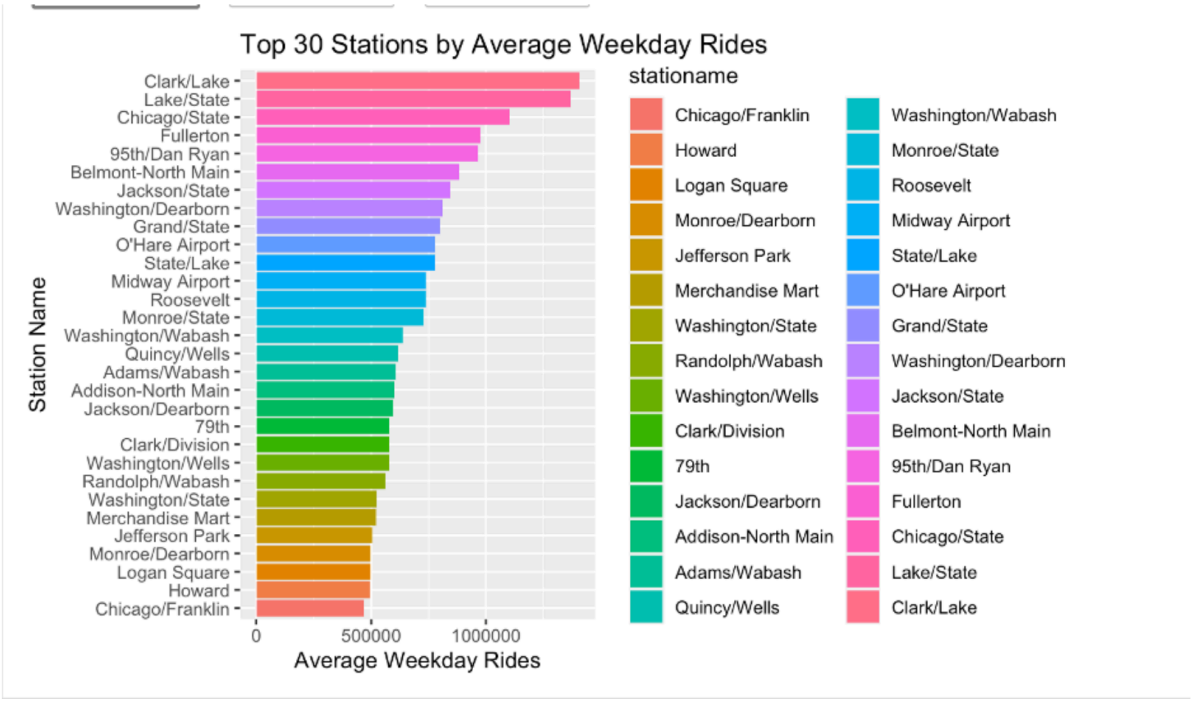
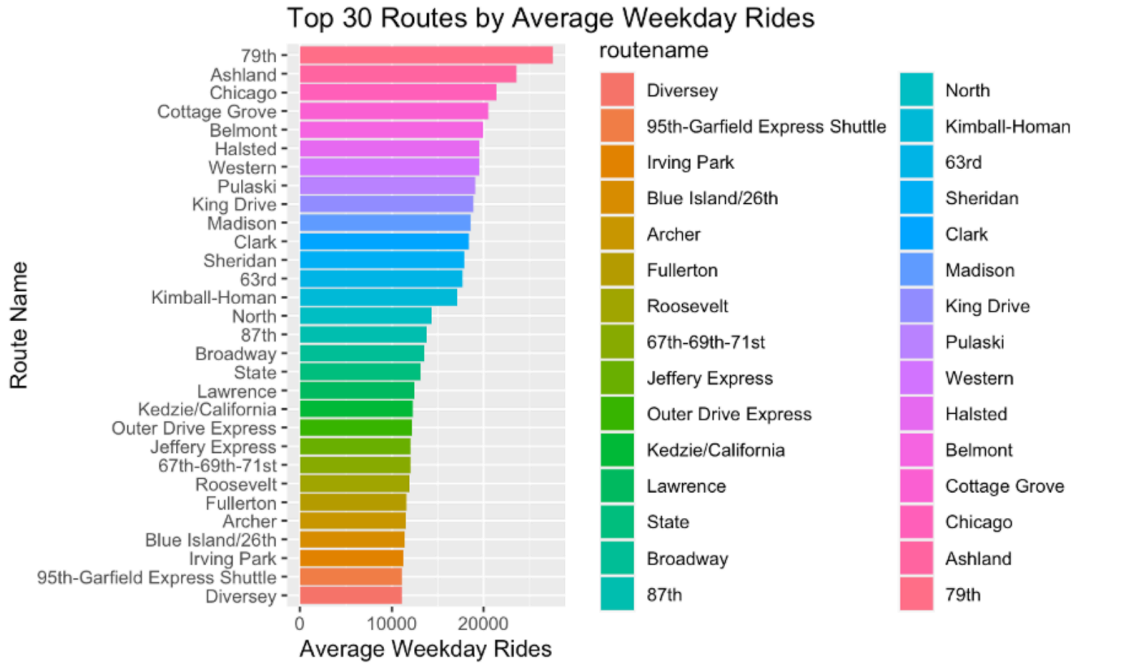


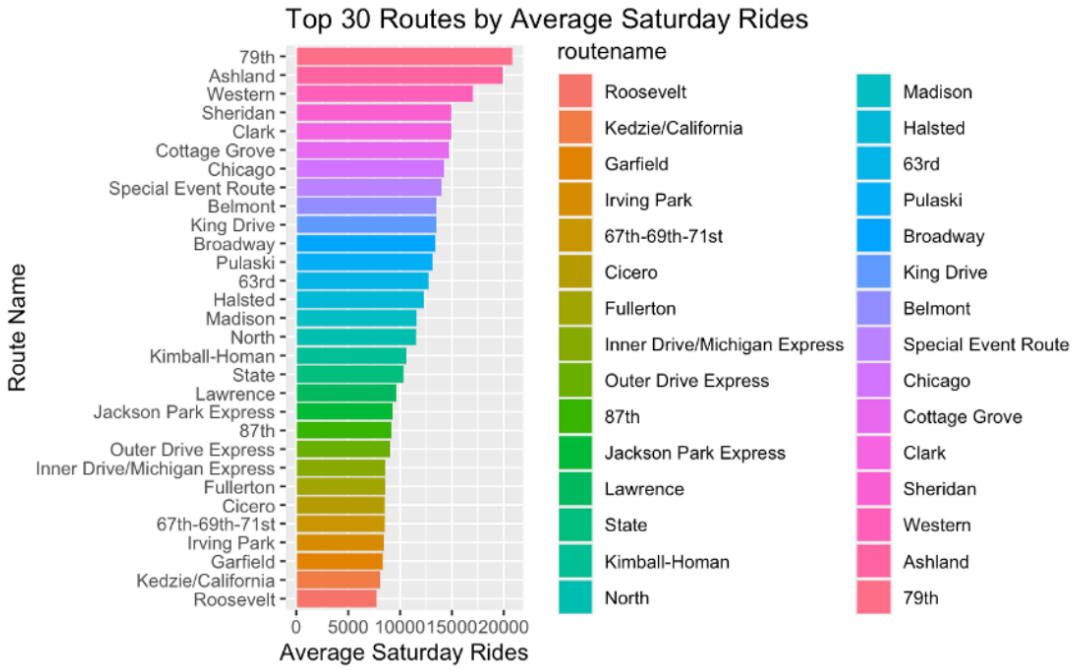
Figure 5.4.2: The figure displays bar charts side-by-side, titled "Top Stations by Day Type." Each chart has a dropdown menu likely labeled "Select Day Type" with options for weekdays, Saturdays/Sundays, and holidays.The x-axis of each chart represents the station name, while the y-axis would indicate the number of boardings. The bars depict the ridership volume at each station for the chosen day type. Stations with the highest ridership would be positioned at the top of the chart, giving a comparative analysis of bus stations and help understand the frequency distribution of bus boardings across all stations with respect to days, events, holidays and weekends.

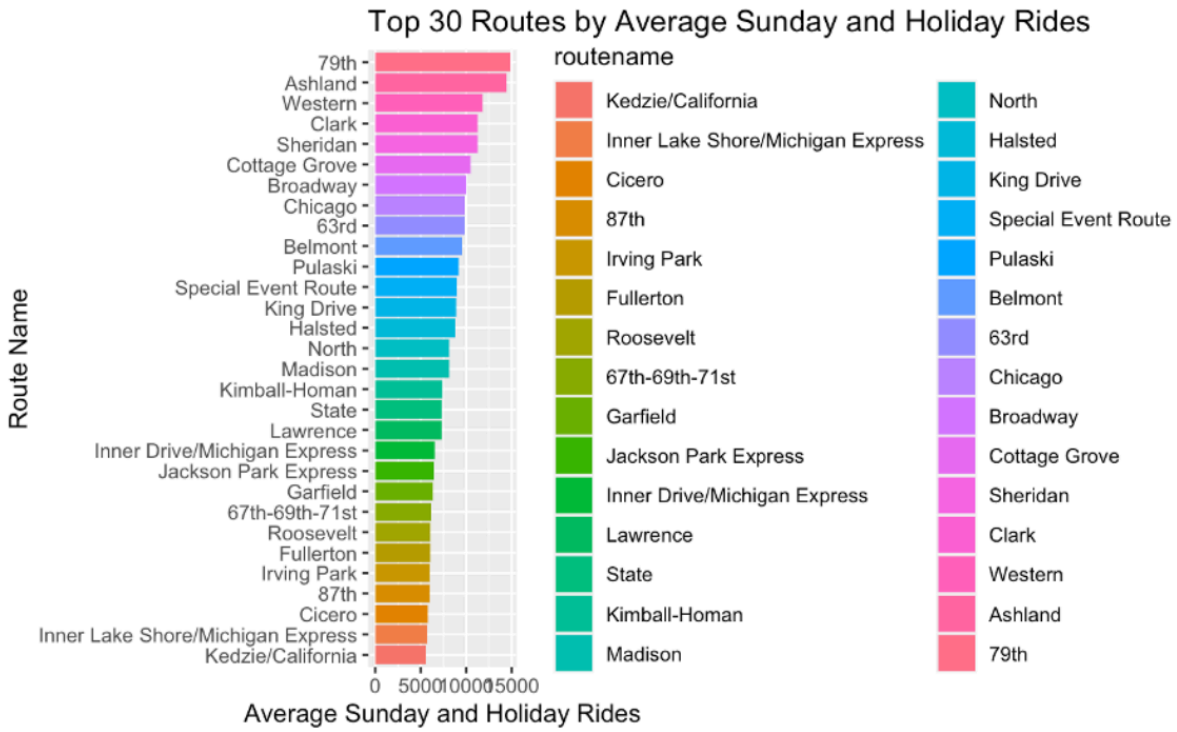
Here’s a list of more trends we observed:

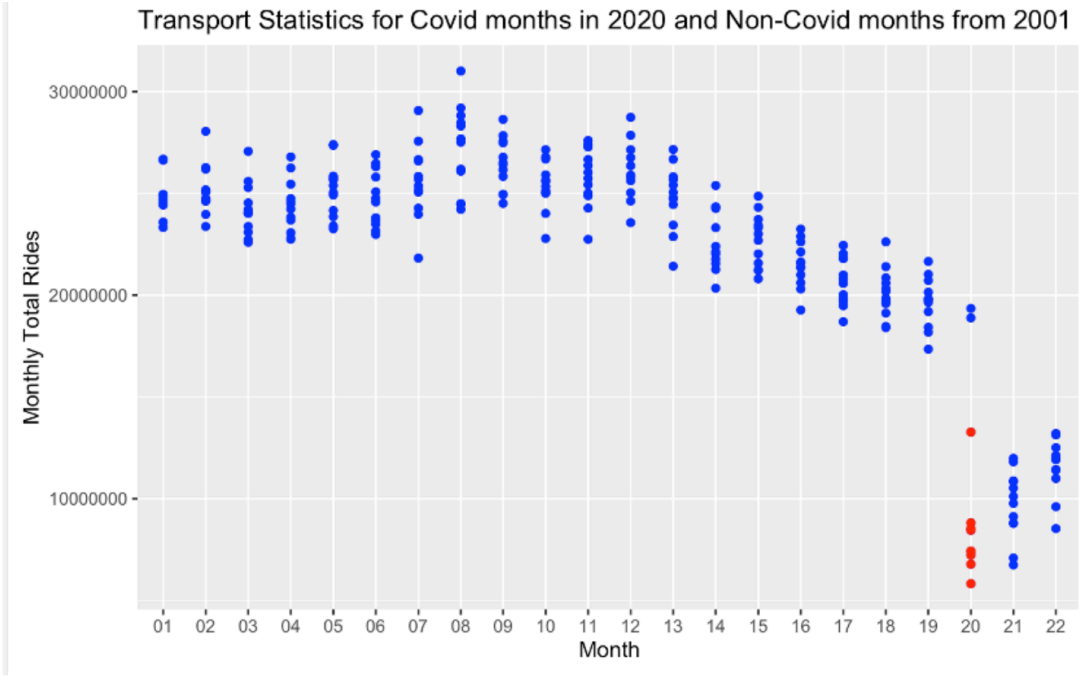

Top 30 Stations by Average Saturday Rides
Station Name
Chicago/State
Grand/State
Belmont-North Main
Lake/State
O'Hare Airport
Roosevelt
Fullerton
Addison-North Main
95th/Dan Ryan
Clark/Division
State/Lake
Clark/Lake
79th
Washington/Dearborn
Washington/Wabash
Howard
Jackson/State
Loyola
Monroe/State
North/Clybourn
Midway Airport
69th
Wilson
Damen/Milwaukee
Cermak-Chinatown
Sheridan
Adams/Wabash
Logan Square
Randolph/Wabash
Sox-35th-Dan Ryan
0
250000
500000
750000
Average Saturday Rides
stationame
Sox-35th-Dan Ryan
Randolph/Wabash
Logan Square
Adams/Wabash
Sheridan
Cermak-Chinatown
Damen/Milwaukee
Wilson
69th
Midway Airport
North/Clybourn
Monroe/State
Loyola
Jackson/State
Howard
Washington/Wabash
Washington/Dearborn
79th
Clark/Lake
State/Lake
Clark/Division
95th/Dan Ryan
Addison-North Main
Fullerton
Roosevelt
O'Hare Airport
Lake/State
Belmont-North Main
Grand/State
Chicago/State


Top 30 Stations by Average Weekday Rides
Station Name
Clark/Lake
Lake/State
Chicago/State
Fullerton
95th/Dan Ryan
Belmont-North Main
Jackson/State
Washington/Dearborn
Grand/State
O'Hare Airport
State/Lake
Midway Airport
Roosevelt
Monroe/State
Washington/Wabash
Quincy/Wells
Adams/Wabash
Addison-North Main
Jackson/Dearborn
79th
Clark/Division
Washington/Wells
Randolph/Wabash
Washington/State
Merchandise Mart
Jefferson Park
Monroe/Dearborn
Logan Square
Howard
Chicago/Franklin
0
500000
1000000
Average Weekday Rides
stationame
Chicago/Franklin
Howard
Logan Square
Monroe/Dearborn
Jefferson Park
Merchandise Mart
Washington/State
Randolph/Wabash
Washington/Wells
Clark/Division
79th
Jackson/Dearborn
Addison-North Main
Adams/Wabash
Quincy/Wells
Washington/Wabash
Monroe/State
Roosevelt
Midway Airport
State/Lake
O'Hare Airport
Grand/State
Washington/Dearborn
Jackson/State
Belmont-North Main
95th/Dan Ryan
Fullerton
Chicago/State
Lake/State
Clark/Lake


Top 30 Routes by Average Weekday Rides
Route Name
79th
Ashland
Chicago
Cottage Grove
Belmont
Halsted
Western
Pulaski
King Drive
Madison
Clark
Sheridan
63rd
Kimball-Homan
North
87th
Broadway
State
Lawrence
Kedzie/California
Outer Drive Express
Jeffery Express
67th-69th-71st
Roosevelt
Fullerton
Archer
Blue Island/26th
Irving Park
95th-Garfield Express Shuttle
Diversey
0
10000
20000
Average Weekday Rides
routename
Diversey
95th-Garfield Express Shuttle
Irving Park
Blue Island/26th
Archer
Fullerton
Roosevelt
67th-69th-71st
Jeffery Express
Outer Drive Express
Kedzie/California
Lawrence
State
Broadway
87th
North
Kimball-Homan
63rd
Sheridan
Clark
Madison
King Drive
Pulaski
Western
Halsted
Belmont
Cottage Grove
Chicago
Ashland
79th


Top 30 Routes by Average Saturday Rides
Route Name
79th
Ashland
Western
Sheridan
Clark
Cottage Grove
Chicago
Special Event Route
Belmont
King Drive
Broadway
Pulaski
63rd
Halsted
Madison
North
Kimball-Homan
State
Lawrence
Jackson Park Express
87th
Outer Drive Express
Inner Drive/Michigan Express
Fullerton
Cicero
67th-69th-71st
Irving Park
Garfield
Kedzie/California
Roosevelt
0
5000
10000
15000
20000
Average Saturday Rides
routename
Roosevelt
Kedzie/California
Garfield
Irving Park
67th-69th-71st
Cicero
Fullerton
Inner Drive/Michigan Express
Outer Drive Express
87th
Jackson Park Express
Lawrence
State
Kimball-Homan
North
Madison
Halsted
63rd
Pulaski
Broadway
King Drive
Belmont
Special Event Route
Chicago
Cottage Grove
Clark
Sheridan
Western
Ashland
79th


Top 30 Routes by Average Sunday and Holiday Rides
Route Name
79th
Ashland
Western
Clark
Sheridan
Cottage Grove
Broadway
Chicago
63rd
Belmont
Pulaski
Special Event Route
King Drive
Halsted
North
Madison
Kimball-Homan
State
Lawrence
Inner Drive/Michigan Express
Jackson Park Express
Garfield
67th-69th-71st
Roosevelt
Fullerton
Irving Park
87th
Cicero
Inner Lake Shore/Michigan Express
Kedzie/California
0
5000
10000
15000
Average Sunday and Holiday Rides
routename
Kedzie/California
Inner Lake Shore/Michigan Express
Cicero
87th
Irving Park
Fullerton
Roosevelt
67th-69th-71st
Garfield
Jackson Park Express
Inner Drive/Michigan Express
Lawrence
State
Kimball-Homan
Madison
North
Halsted
King Drive
Special Event Route
Pulaski
Belmont
63rd
Chicago
Broadway
Cottage Grove
Sheridan
Clark
Western
Ashland
79th


Transport Statistics for Covid months in 2020 and Non-Covid months from 2001
Monthly Total Rides
30000000
20000000
10000000
01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
21
22
Month

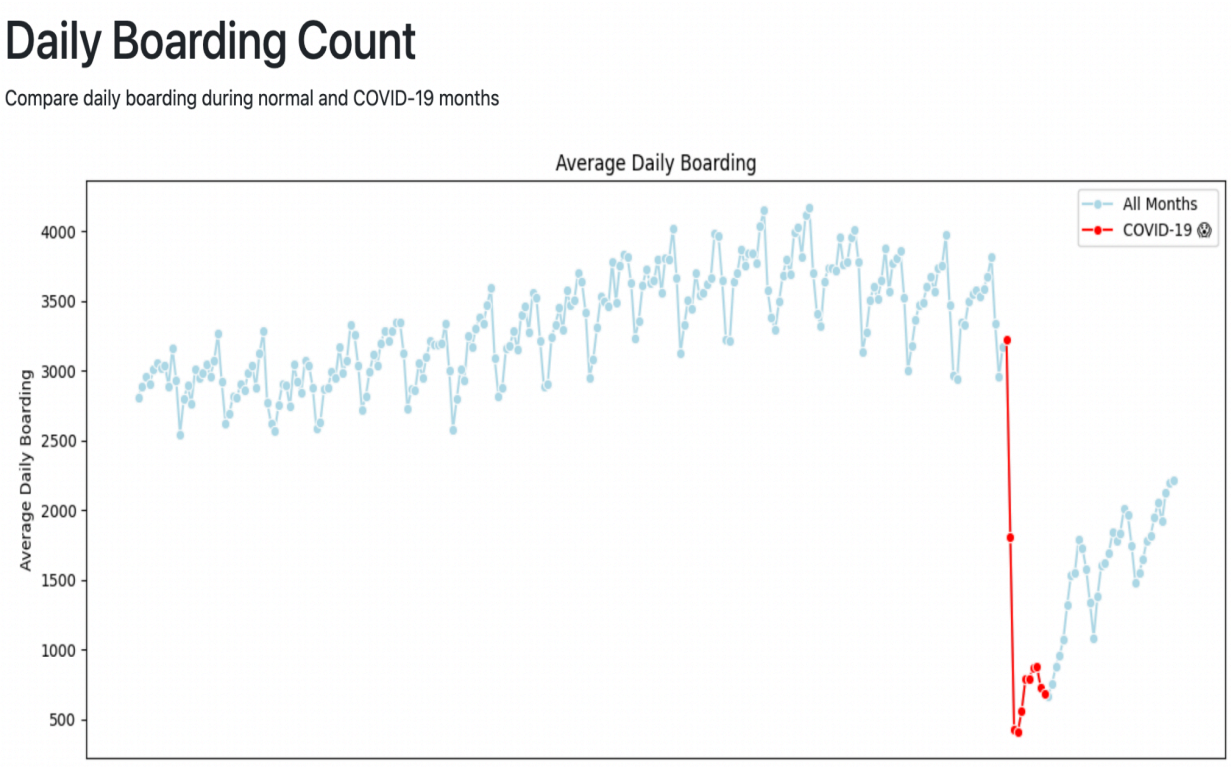


Figure 5.4.3: The figure shows a line graph titled “Daily Boarding Count: Compare daily boarding during normal and COVID-19 months.” The x-axis displays the month, from July 2019 to June 2020. The y-axis shows the average daily boarding count. There are two lines on the graph, colored blue and red, where the blue line represents “All Months” and the red line represents “COVID-19.” The average daily boarding count appears to be higher in the “All Months” line compared to the “COVID-19” line. The highest ridership for all months is in July 2019, with an average daily boarding count of close to 4,000. The ridership then follows a seasonal trend, decreasing until December 2019 and then increasing again in the spring. The average daily boarding count for all months starts to decrease again in May 2020. The “COVID-19” line starts in March 2020, which coincides with a sharp decrease in ridership compared to all months. The average daily boarding count for “COVID-19” months remains consistently lower than for all months throughout the rest of the timeframe shown in the graph.This helps us understand the impact of Covid-19 on CTA ride frequency.

Figure 5.4.4 : The figure “Precipitation vs. Total Rides" that investigates the potential link between daily precipitation and CTA bus ridership.On the y-axis labeled "Precipitation (Inches)", reflects the amount of precipitation measured on a given day. The x-axis labeled "Total Rides", that represents the total number of CTA bus rides recorded on a specific day. The chart employs hexagons to visualize the data distribution.The color of the hexagons signifies the density of average rides of buses for that day. The precipitation rate impacts on the frequency of CTA rides because of increase in travel time, conjunctions, visibility of roads, etc, which can been seen in chart.

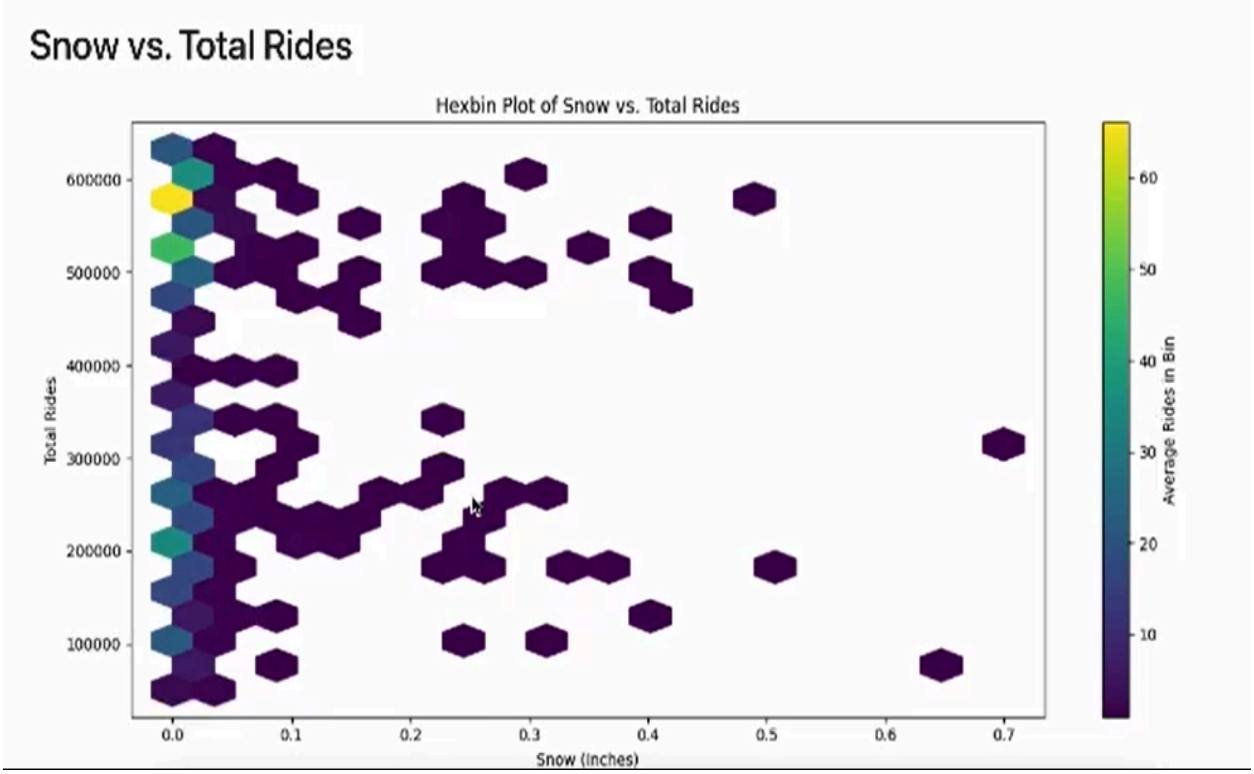


Figure 5.4.5: The heatmap titled, “Snow vs Total Rides” where the y-axis is "Snow (Inches)" that represents the amount of snowfall recorded on a specific day, while x-axis is "Total Rides" and represents the total number of CTA bus rides recorded on a given day.The figure describes the reduce bus frequency or implement delays on snowy days to account for lower ridership and potential road hazards. This will help in timely advisories informing riders about potential delays or service disruptions due to snowfall.

7. Future Scope

This project has explored various aspects of CTA ridership data, providing valuable insights for optimizing public transportation services. Here are some potential areas for future exploration:

- **Expand the Scope of Influencing Factors:**
    - Integrate data on road closures, public transportation disruptions (e.g., signal malfunctions, track repairs) and gas prices to understand how these factors influence ridership choices.
    - Analyze the impact of special events (concerts, festivals) on ridership patterns near event venues.
- **Deeper Dive into Route Analysis:**
    - Explore variations in ridership across different bus routes. This could involve identifying high-traffic routes, underutilized routes, and potential route optimizations.

- **Real-time Analytics:**
    - Implement a real-time data processing pipeline to analyze ridership patterns and service disruptions as they occur. This allows for immediate responses and improved communication with riders.

8. Conclusion

This project successfully implements big data tools and techniques to analyze the CTA ridership dataset. By uncovering valuable insights into ridership patterns, weather impact, and special event influences, the project offers valuable information for the CTA to improve service quality, optimize resource allocation, and enhance the overall rider experience. The proposed future scope expands on this analysis, aiming to provide even deeper understanding and predictive capabilities to support data-driven decision-making within the CTA.

9. Scope of the Project

The scope of the project was limited due to the unavailability of a suitable dataset. As a result, we were unable to perform a comprehensive analysis of the desired aspect of the project. However, we were able to identify potential sources for obtaining the required data and explore alternative solutions for data collection.

We can analyze the CTA crime data and can provide insights into the safety of different stations and areas, identifying any patterns or trends in criminal activity. And by combining crime data with ridership data, it would be possible to identify hotspots where crimes are more likely to occur, and this can inform the allocation of resources to improve security in those areas.

COVID-19 has also affected the optimal routes and schedules for CTA buses and trains. Analyzing CTA data and COVID-19 data can help identify areas where service levels can be reduced or increased to better match demand. This can help reduce costs while ensuring that essential services are maintained. Unfortunately, we were unable to link the COVID-19 dataset with the CTA routes dataset, which limited our ability to conduct analysis on the impact of the pandemic on the public transportation system in Chicago.

Despite the limitations, the project was able to provide valuable insights into the available data and highlight the importance of having a well-curated dataset for conducting data analysis.

10. References

1. Python
2. Apache PySpark
3. Singh, P. K. (2021, December 9). Manage Data with PySpark. Apress eBooks. https://doi.org/10.1007/978-1-4842-7777-5_2
4. Ranganathan, G. "Real time anomaly detection techniques using pyspark frame work." *Journal of Artificial Intelligence* 2, no. 01 (2020): 20-30.
5. https://medium.com/@anitateladevalapalli777/big-data-analytics-project-using-kafka-pyspark-and-tableau-b1c28b7cebad
6. https://www.kaggle.com/datasets/chicago/chicago-transit-authority-cta-data/versions/8?resource=download
7. https://www.axios.com/local/chicago/2022/04/21/mapping-cta-crime-statistics-chicago-trains
8. https://www.wsdot.wa.gov/partners/erp/background/ST3%20Draft%20RidershipForecastingMethodologyReport_6March2015.pdf
9. https://www.nctr.usf.edu/pdf/527-14.pdf
10. Schmitt, A. J. (2014). Modeling the influence of weather on public transportation ridership in Chicago. Transportation Research Part A: Policy and Practice, 65, 48-60
11. Cervero, R. (2013). The future of urban public transportation. Transportation Research Part A: Policy and Practice, 48(1), 3–14. DOI: 10.1016/j.tra.2012.09.003: doi:10.1016/j.tra.2012.09.003.