**Uber Data Analysis using R**

**NIKHIL**

**23MSM40015**

**Project for Fundamentals of Programming (R/Python)**

**Department of Mathematics, University Institute of Sciences, Chandigarh University**

**ABSTRACT**

This study analyzes the patterns of urban mobility in a large city using data from a ride-sharing platform. The data consists of the total number of trips taken by hour and month from April to September 2023. The study aims to identify the peak and off-peak hours, the monthly trends, and the average level of travel activity in the city. The results show that the number of trips varies by time of day and season, with the highest number of trips in July at 8 PM and the lowest number in April at 4 AM. The number of trips generally increases from April to July, then decreases from July to September. The peak hour for travel is 18:00 and the lowest hour is 3:00. The average number of trips per hour is 140,000. The study concludes that the city has a high and dynamic demand for travel that is influenced by various factors such as weather, daylight, and human activity.

**Keywords**: R programming, Trends and Patterns, Data Cleaning, Data, Insights.

**R Code**

Step 1

# Importing useful libraries

**library(ggplot2)**

**library(ggthemes)**

**library(lubridate)**

**library(dplyr)**

**library(tidyr)**

**library(DT)**

**library(scales)**

* **ggplot2**: This is a data visualization package for R. It helps in creating visually appealing plots, charts, and graphs.
* **ggthemes**: This package provides additional themes, scales, and geoms for the ggplot: package, allowing you to modify the appearance of your plots.
* **lubridate**: This package simplifies working with dates and times in R. It provides functions to parse, manipulate, and do arithmetic with date-times.
* **dplyr**: This package provides a set of tools for efficiently manipulating datasets in R. It includes functions to filter, mutate, summarize, and arrange data frames.
* **tidyr**: This package helps to create tidy data, where each column is a variable, each row is an observation, and each cell contains a single value. It provides functions for changing the shape and hierarchy of a dataset.
* **DT**: This package provides an interface to the JavaScript library ‘DataTables’. It allows R data objects (matrices or data frames) to be displayed as interactive tables on HTML pages.
* **scales**: This package provides the internal scaling infrastructure used by ggplot2. It allows you to override the default breaks, labels, transformations, and palettes in your plots.

Step 2

# Data loading

**apr <- read.csv("uberdataset/uber-raw-data-apr14.csv")**

**may <- read.csv("uberdataset/uber-raw-data-may14.csv")**

**june <- read.csv("uberdataset/uber-raw-data-jun14.csv")**

**july <- read.csv("uberdataset/uber-raw-data-jul14.csv")**

**aug <- read.csv("uberdataset/uber-raw-data-aug14.csv")**

**sept <- read.csv("uberdataset/uber-raw-data-sep14.csv")**

Step 3

#Combining the monthly data and finding its dimensions

**data <- rbind(apr, may, june, july, aug, sept)**

**cat("The dimensions of the data are:", dim(data))**

#Showing the head of the combined data

**head(data)**

Step 4

**Data$Date.Time <- as.POSIXct(data$Date.Time, format="%m/%d/%Y %H:%M:%S")**

**data$Time <- format(as.POSIXct(data$Date.Time, format = "%m/%d/%Y %H:%M:%S"), format="%H:%M:%S")**

data$Date.Time <- as.POSIXct(data$Date.Time, format="%m/%d/%Y %H:%M:%S"): This line is converting the Date.Time column in your data dataframe to a POSIXct time format. The as.POSIXct function converts the time data into a format that R can understand and manipulate. The format argument specifies the current format of the dates and times in the Date.Time column.

data$Time <- format(as.POSIXct(data$Date.Time, format="%m/%d/%Y %H:%M:%S"), format="%H:%M:%S"): This line is creating a new column Time in your data dataframe. It’s taking the Date.Time column (which is now in POSIXct format), converting it back to a character string with the format function, but this time only keeping the time portion (%H:%M:%S for hours, minutes, and seconds). So, the Time column will only contain the time part of the Date.Time column.

Step 5

**data$hour = factor(hour(hms(data$Time)))**

**data$month <- factor(month(data$Date.Time, label=TRUE)) (dim(df))**

These lines of code are manipulating the Time and Date.Time columns in your data dataframe.

* data$hour = factor(hour(hms(data$Time))): This line is creating a new column hour in your data dataframe. It’s taking the Time column, converting it to an hms object using the hms function, extracting the hour component using the hour function, and then converting this to a factor variable with the factor function. A factor is a categorical variable that can hold a limited number of different values. Factors are often used in statistical modeling.
* data$month <- factor(month(data$Date.Time, label=TRUE)): This line is creating a new column month in your data dataframe. It’s taking the Date.Time column, extracting the month component using the month function, and then converting this to a factor variable with the factor function. The label=TRUE argument in the month function means that the months will be returned as their full names (e.g., “January”, “February”, etc.) rather than their numeric values (1, 2, etc.).

Step 6

**hourly\_data <- data %>%**

**group\_by(hour) %>%**

**dplyr::summarize(Total = n())**

**datatable(hourly\_data)**

This code is performing a grouping operation on the data dataframe and then creating a new dataframe hourly\_data.

* group\_by(hour): This line is grouping the data by the hour column. The group\_by function from the dplyr package groups the dataframe by the specified column(s).
* dplyr::summarize(Total = n()): This line is summarizing the grouped data. The summarize (or summarise) function from the dplyr package creates a new dataframe that summarizes the grouped data. Here, it’s creating a new column Total that contains the count of rows in each group. The n() function returns the number of rows in each group.
* datatable(hourly\_data): This line is creating an interactive HTML table widget using the datatable function from the DT package. The table displays the data in the hourly\_data dataframe. The datatable function creates an HTML widget to display rectangular data (a matrix or data frame) using the JavaScript library DataTables.

Step 7

**ggplot(hourly\_data, aes(hour, Total)) +**

**geom\_bar(stat="identity",**

**fill="steelblue",**

**color="red") +**

**ggtitle("Trips Every Hour", subtitle = "aggregated today") +**

**theme(legend.position = "none",**

**plot.title = element\_text(hjust = 0.5),**

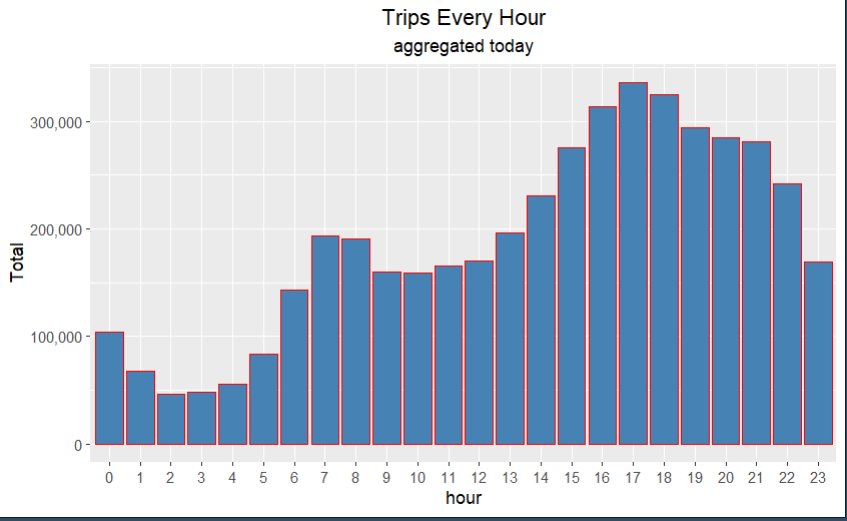
**plot.subtitle = element\_text(hjust = 0.5)) +**

**scale\_y\_continuous(labels=comma)**

This code is creating a bar plot using the ggplot2 package in R. Here’s what each part of the code does:

* ggplot(hourly\_data, aes(hour, Total)): This initializes the plot with hourly\_data as the data source. The aes function maps the hour variable to the x-axis and the Total variable to the y-axis.
* geom\_bar(stat="identity", fill="steelblue", color="red"): This adds a bar geometry to the plot. The stat="identity" argument means that the heights of the bars represent the values in the data. The bars are filled with the color “steelblue” and outlined in “red”.
* ggtitle("Trips Every Hour", subtitle = "aggregated today"): This adds a main title “Trips Every Hour” and a subtitle “aggregated today” to the plot.
* theme(legend.position = "none", plot.title = element\_text(hjust = 0.5), plot.subtitle = element\_text(hjust = 0.5)): This modifies the theme of the plot. The legend is removed with legend.position = "none". The hjust = 0.5 inside element\_text centers the plot title and subtitle.
* scale\_y\_continuous(labels=comma): This modifies the y-axis scale. The labels=comma argument formats the y-axis labels with commas as thousand separators.

So, this code is creating a bar plot of the total number of trips for each hour of the day, with the data aggregated for today. The bars are colored steel blue with red outlines, and there is no legend. The title and subtitle are centered, and the y-axis labels have comma separators. The height of each bar represents the total number of trips for that hour.



Step 8

**month\_hour\_data <- data %>% group\_by(month, hour) %>% dplyr::summarize(Total = n())**

**ggplot(month\_hour\_data, aes(hour, Total, fill=month)) +**

**geom\_bar(stat = "identity") +**

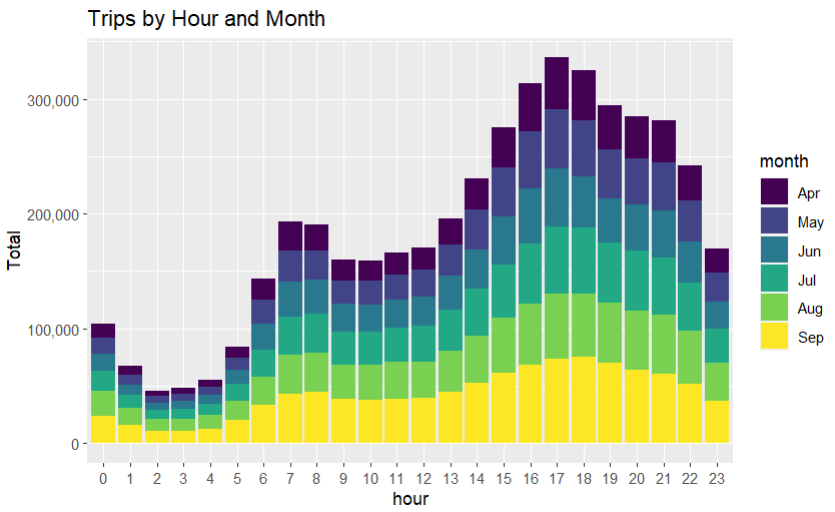
**ggtitle("Trips by Hour and Month") +**

**scale\_y\_continuous(labels = comma)**

This code performs a grouping operation on the data dataframe by both month and hour and then creates a new dataframe month\_hour\_data. It’s also creating a bar plot to visualize the total number of trips by hour and month.

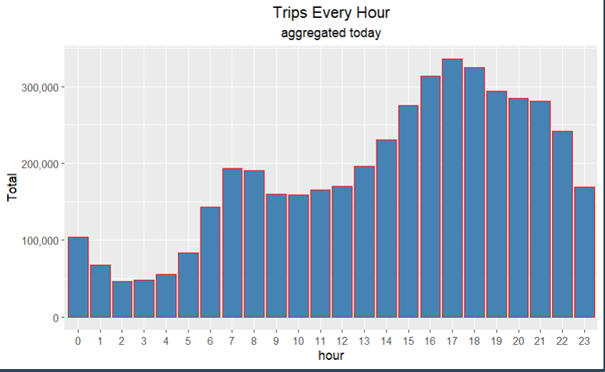
* **month\_hour\_data <- data %>% group\_by(month, hour) %>%** **dplyr::summarize(Total = n()):** This line is grouping the data dataframe by both month and hour, and then summarizing the grouped data to create a new column Total that contains the count of rows in each group. The result is stored in the new dataframe month\_hour\_data.
* **ggplot(month\_hour\_data, aes(hour, Total, fill=month)):** This initializes the plot with month\_hour\_data as the data source. The aes function maps the hour variable to the x-axis, the Total variable to the y-axis, and the month variable to the fill aesthetic (which determines the color of the bars).
* **geom\_bar(stat = "identity"):** This adds a bar geometry to the plot. The stat="identity" argument means that the heights of the bars represent the values in the data.
* **ggtitle("Trips by Hour and Month"):** This adds a main title “Trips by Hour and Month” to the plot.
* **scale\_y\_continuous(labels = comma):** This modifies the y-axis scale. The labels=comma argument formats the y-axis labels with commas as thousand separators.

So, this code is creating a bar plot of the total number of trips for each hour of the day, grouped by month. The height of each bar represents the total number of trips for that hour, and the color of the bar represents the month. The y-axis labels have comma separators.

**Insights:**

The insights we are getting from this graph are:

* **Trips by hour and month**: The graph shows the total number of trips taken by hour and month. We can see how the trips vary by time of day and season.
* **Peak and off-peak hours**: The graph shows that the highest number of trips taken is in the month of July at around 8 PM. This could be due to the summer weather and the evening activities. The lowest number of trips taken is in the month of April at around 4 AM. This could be due to the spring season and the early morning hours.
* **Monthly trends**: The graph shows that the number of trips generally increases from April to July, then decreases from July to September. This could be related to the temperature, rainfall, and daylight hours of each month.



The insights that can be drawn from this graph are:

* Peak hour: The number of trips taken every hour is highest at 17:00 with more than 300,000 trips. This suggests that this is the most popular time for people to travel, possibly due to commuting from work or school.
* Lowest hour: The number of trips taken every hour is lowest at 2:00 with about 20,000 trips. This suggests that this is the least popular time for people to travel, possibly due to sleeping or resting.
* Trend: The number of trips taken every hour gradually increases from 2:00 to 17:00 and then gradually decreases from 17:00 to 23:00. This suggests that there is a daily cycle of travel demand that follows the typical patterns of human activity.
* Average: The average number of trips taken every hour is about 140,000. This suggests that there is a high level of travel activity throughout the day, indicating a busy and dynamic city.