Predicting Traffic Flow on Bay Area Expressways

MATH 748 – Theory and Applications of Statistical Machine Learning

Thanoj Muddana

Professor Tao He

San Francisco State University

Date: Dec 18, 2024

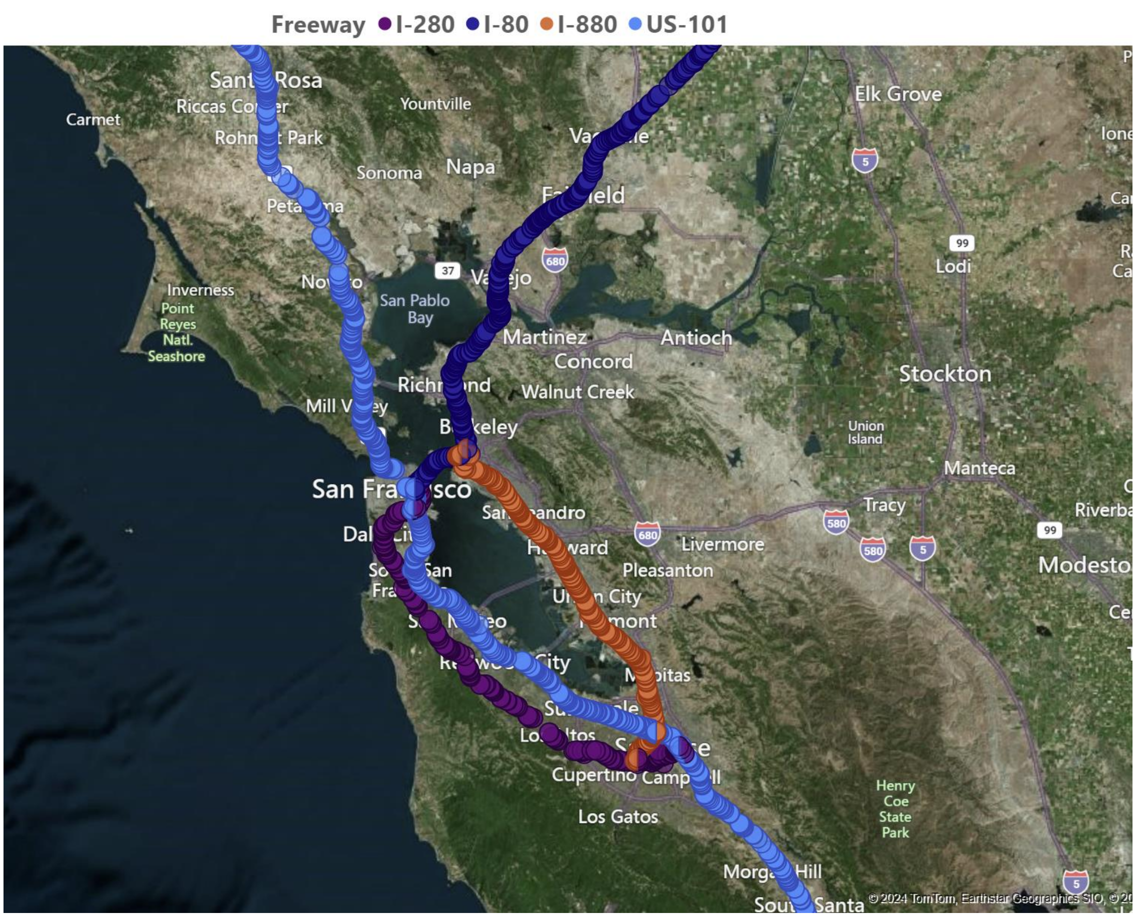


Table of Contents

1. Executive Summary

I undertook this project to predict **traffic flow** on Bay Area expressways using station-hourly data from the Caltrans PeMS database. The focus was on understanding traffic patterns and modeling total flow using statistical and machine learning techniques. I addressed significant challenges in handling missing data and optimizing the dataset for modeling by summarizing data into meaningful dimensions. XGBoost emerged as the best-performing model, achieving an R-squared of 0.95. This project demonstrates how machine learning can be effectively applied to traffic prediction and lays the groundwork for future improvements.

2. Introduction

Traffic congestion remains a critical issue in urban planning and transportation management, with significant economic and environmental impacts. This project focuses on predicting **traffic flow** across key expressways in the Bay Area, specifically **US-101**, **I-80**, **I-280**, and **I-880**, leveraging historical data from the **Caltrans PeMS database (District 4)**.

The primary goal was to model **Total Flow**, defined as the number of vehicles passing a station per hour, using data-driven approaches. This objective aligns with the broader aim of understanding traffic behavior, enabling smarter transportation systems, and providing a foundation for potential real-time applications like congestion prediction.

This project involved several stages, beginning with **data exploration** to identify trends, anomalies, and key features. Data preprocessing was an extensive process, given the large volume of data and the challenges of missing values. Feature engineering played a vital role in enhancing the dataset by adding temporal flags like **peak hour** and **weekend indicators** and computing metrics like **lane count** from raw lane flow data.

I utilized techniques from **The Elements of Statistical Learning** to guide the model selection process, applying both classical regression models and modern machine learning algorithms. The project concludes with an evaluation of these models and a discussion on potential improvements.

3. Data Description

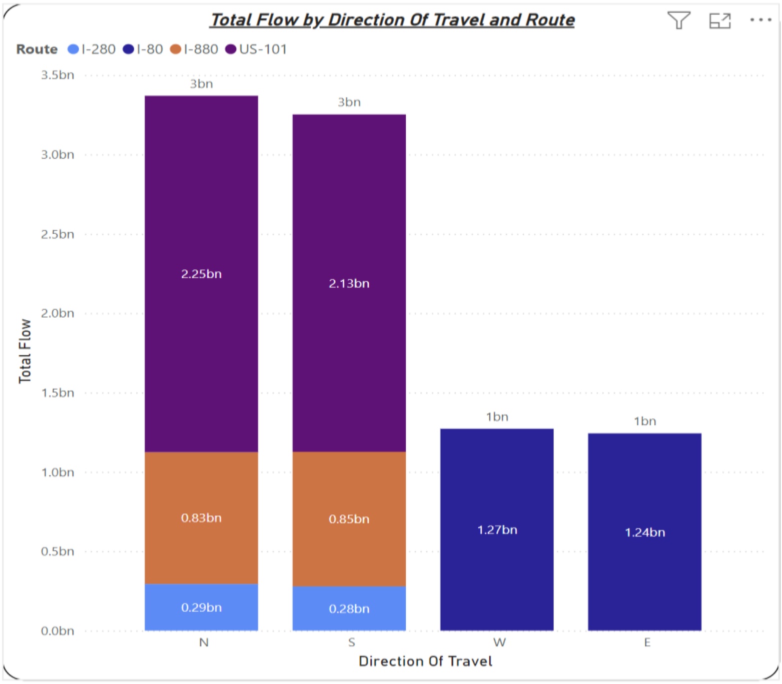
The data for this project was sourced from the **Caltrans PeMS database**, specifically the **station-hourly dataset for District 4**, which encompasses the Bay Area. This dataset provides traffic information from strategically placed sensors along major expressways, including **US-101**, **I-80**, **I-280**, and **I-880**.

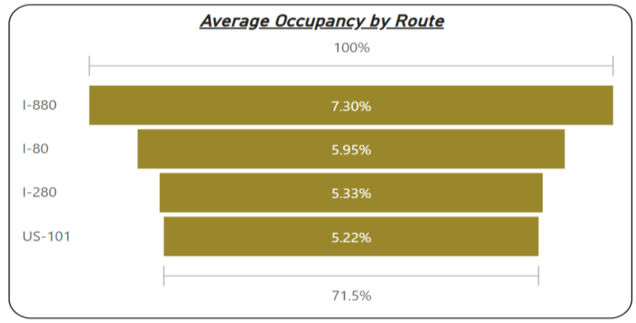
Traffic Data: Each month's file contained approximately 5 million rows, representing hourly observations from various stations.

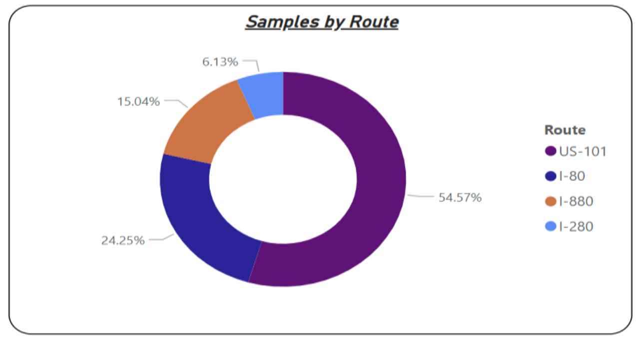
<u>Key Features:</u>

Timestamp: The date and hour of the observation.
Station ID: A unique identifier for each monitoring station.
Route: The highway (e.g., US-101).
Direction of Travel: The traffic flow direction (North, South, East, or West).
Lane-Type: Classification of lanes (e.g., mainline, off-ramp).
Total Flow: The total number of vehicles passing a station within the hour.
Average Speed: The average vehicle speed for the hour.
Lane-specific Flow Data: Detailed flow, occupancy, and speed for up to 8 lanes.

4. Exploratory Data Analysis (EDA)

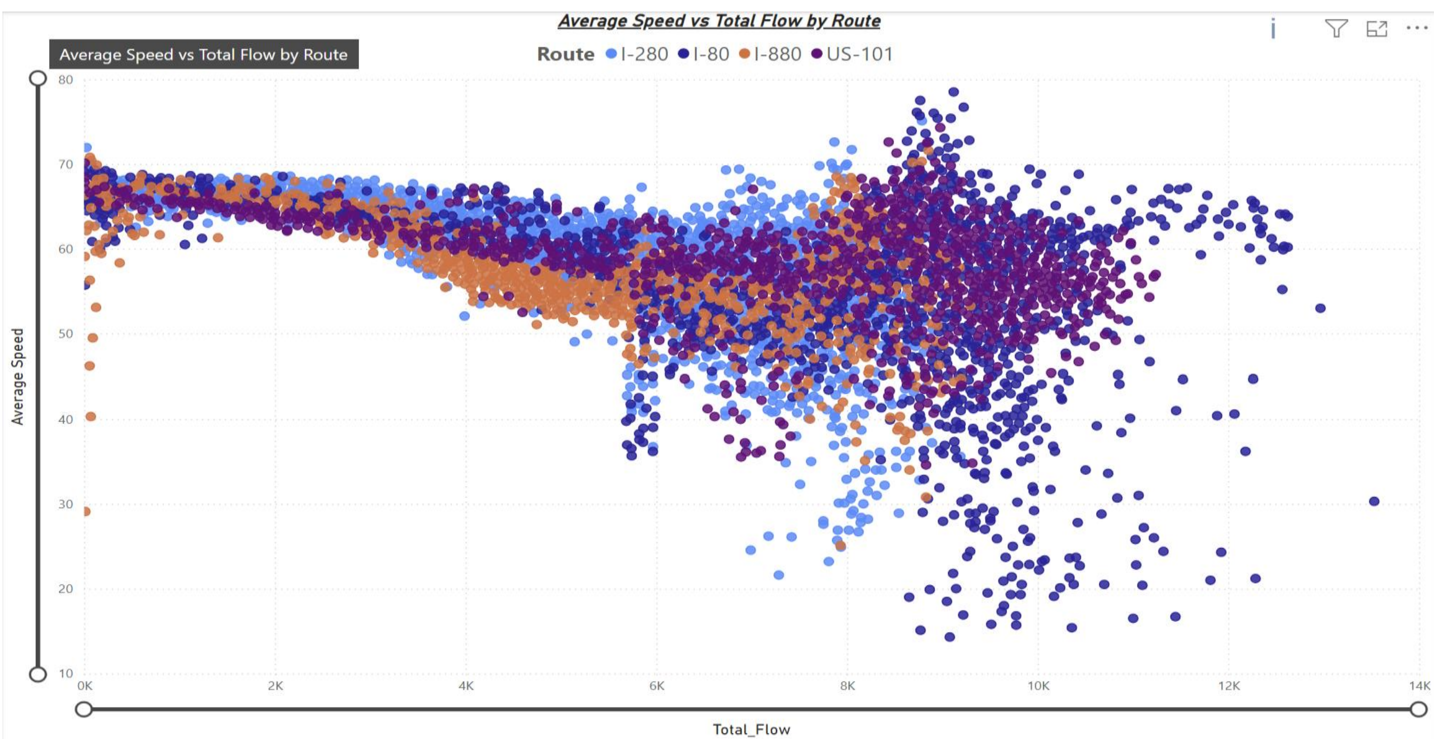




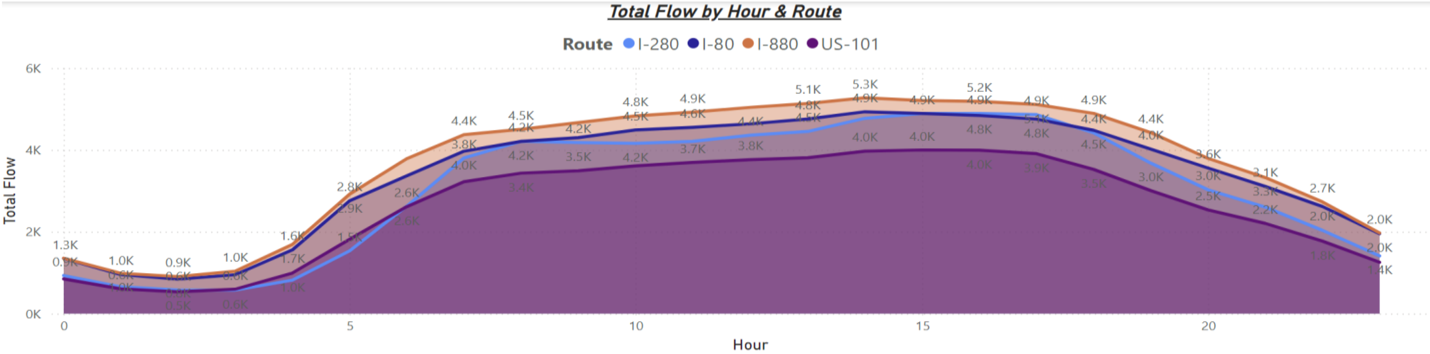


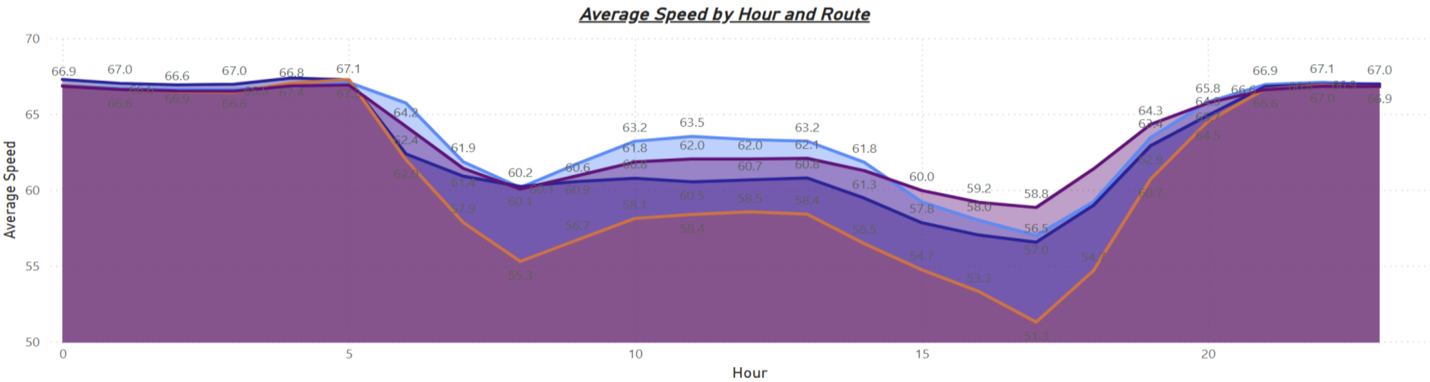
US-101 plays a significant role in traffic flow across all directions, indicating it as a major corridor in District 4.

Data samples are predominantly from **US-101** (**54.57%**), followed by **I-80** (**24.25%**) and **I-880** (**15.04%**).

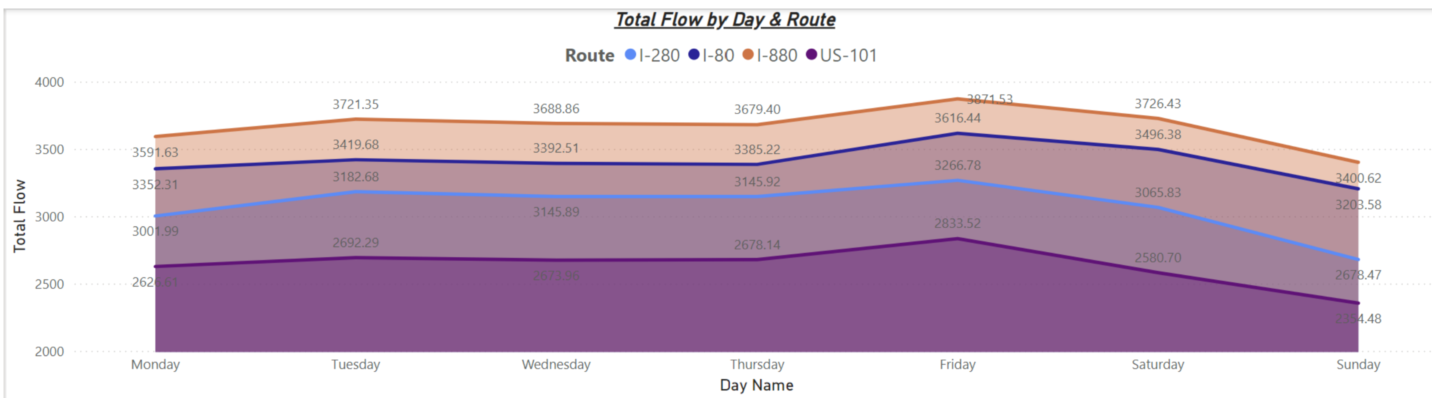


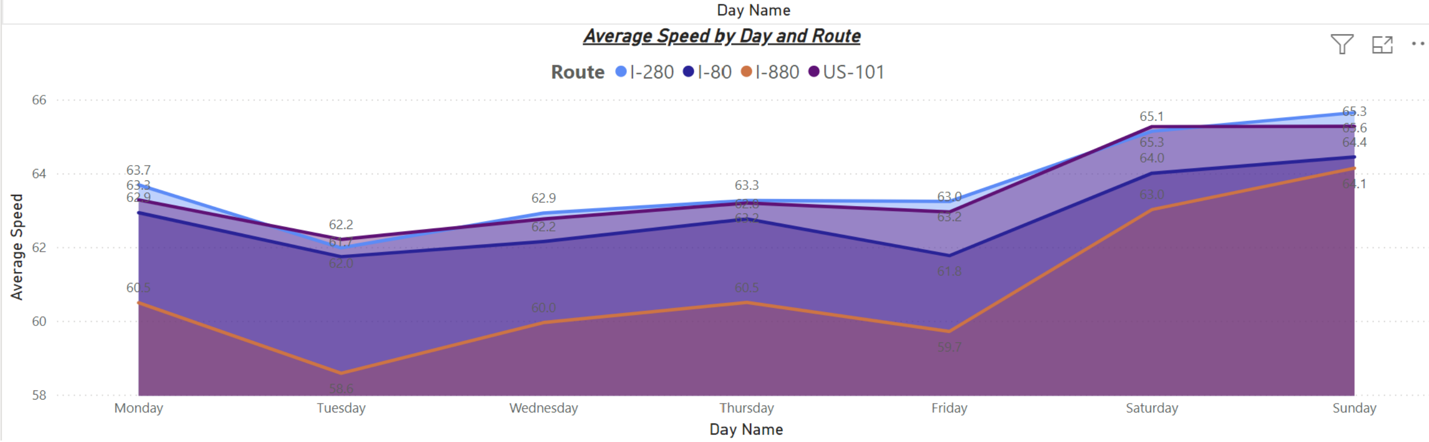
High flow volumes generally result in reduced speeds, highlighting congestion during peak flow times.





Peak traffic hours are clearly visible, indicating the necessity for congestion mitigation strategies during these times. Higher flow during peak hours results in reduced speeds, highlighting congestion trends.





Weekdays, especially **Fridays**, experience the heaviest traffic flows, indicating commuter behavior trends. Traffic congestion is heavier during weekdays, particularly on routes like I-880.

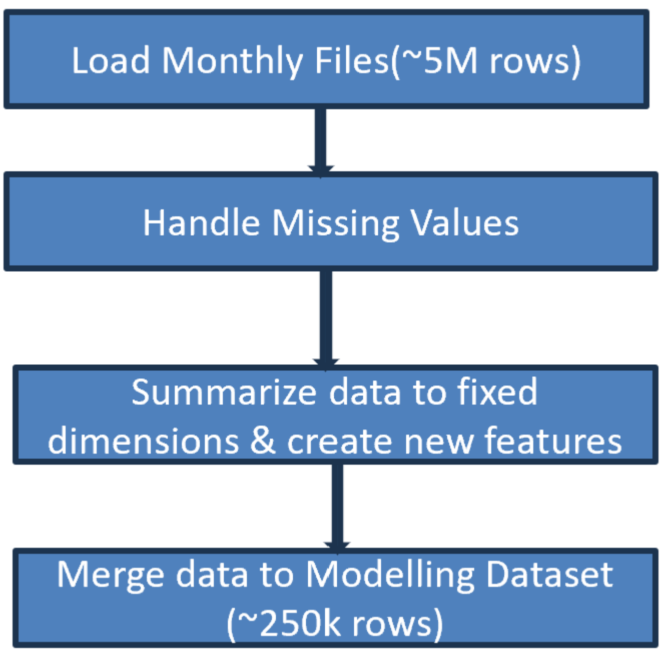
5. Data Preparation & Feature Engineering

Key Challenges:

- Large Data Volume: Raw data contained approximately 5M rows per month, leading to a total of 15M rows for three months. Managing such a massive dataset posed computational limitations.

- Missing Values: ~29% of records had missing values in key metrics like Total Flow, Avg Speed, and Avg Occupancy. These records were handled efficiently to avoid data inconsistencies.

- Memory Efficiency: Sequential file loading was implemented to efficiently handle monthly data files. Data was grouped and summarized to reduce the dataset size while retaining critical details



Data Transformation:

Aggregated data using fixed dimensions:

- **Station** (unique traffic monitoring location)
- **Day** (day of the week)
- **Hour** (hourly trends)
- **Route** (e.g., US-101, I-80, I-280, I-880)
- **Direction of Travel** (North, South, East, West)

Engineered Features:

- Day Name: Extracted from the timestamp to identify weekday/weekend patterns.
- Hour: Extracted to analyze hourly traffic flow.
- Lane Count: Derived as the average number of active lanes per station.
- Weekend Flag: Binary flag to differentiate weekend (1) and weekday (0).
- Peak Hour Flag: Binary feature to mark peak hours (1) and non-peak hours (0).

Final Processed Dataset:

- Reduced data size to approximately 250k rows while preserving key metrics.
- Numerical features included:
    - Avg Lanes Count
    - Avg Speed
    - Avg Total Occupancy
    - Total Flow (Target Variable).

6. Model Selection and Evaluation

6.1 Data Splitting:

- The data was split into Training (70%) and Testing (30%) sets to evaluate model generalization.
- The caret library in R was used to ensure reproducibility and an efficient split.
- Training Set: 70% of the total data used to fit the models.
- Testing Set: 30% used for evaluation

6.2 Models and Evaluation:

The following models were chosen based on their robustness in regression problems, as discussed in the *Elements of Statistical Learning*:

1. **Linear Regression**
    - A simple and interpretable baseline model.
    - It helps in identifying linear relationships between predictors and the target variable.
2. **Decision Tree**
    - A non-linear model that splits data recursively on significant features.
    - It can capture complex relationships without requiring extensive preprocessing.
3. **XGBoost (Extreme Gradient Boosting)**
    - An advanced boosting algorithm capable of handling large datasets efficiently.
    - It combines multiple weak learners (trees) to form a strong predictive model.
4. **Ridge Regression**
    - A variant of linear regression that applies L2 regularization to reduce overfitting.
    - Handles multicollinearity effectively by shrinking coefficients.
5. **Lasso Regression**
    - Similar to Ridge Regression but uses L1 regularization.
    - It performs **feature selection** by shrinking some coefficients to zero, thus reducing model complexity.

Each model was trained on the training set and predictions were made on the testing set.
Model performance was measured using:

- **RMSE (Root Mean Square Error):** Indicates average error magnitude.
- **R-Squared:** Represents how well the model explains variance in the target variable.
- **MAE (Mean Absolute Error):** Measures average absolute differences between predictions and actual values.

Model	**RMSE**	**R-Squared**	**MAE**
Linear Regression	629.14	0.894	464.2
Decision Tree	653.01	0.886	506.11
Ridge Regression	658.99	0.885	498.15
Lasso Regression	629.2	0.894	463.8
XGBoost	427.55	0.951	303.03

7. Results & Conclusion

- XGBoost emerged as the best-performing model with:
  - The lowest RMSE (427.55), indicating minimal prediction error.
  - The highest R-Squared (0.9515), showcasing its ability to explain 95% of the variance in the target variable.
  - The lowest MAE, further solidifying its accuracy.

- Linear Regression and Lasso Regression provided comparable results, demonstrating that simple linear models can still perform well in predicting traffic flow, but they failed to capture the non-linear relationships in the data as effectively as XGBoost.

- Decision Tree showed decent results but was outperformed by ensemble methods like XGBoost, which leverage multiple weak learners to make robust predictions.

- Regularized Models (Ridge and Lasso): While Ridge and Lasso Regression helped mitigate overfitting, their performance was similar to Linear Regression, indicating limited multicollinearity in the data.

Best Model:

The **XGBoost** model outperformed all other models in terms of accuracy and error metrics:

- **Lowest RMSE:** Indicates minimal prediction error.
- **Highest R-squared:** Captures 95% of the variance in traffic flow.
- **Lowest MAE:** Reflects high precision in predictions.

Top Predictors for XGBoost:

The following features emerged as the most significant contributors to the traffic flow prediction:

1. **Avg_Total_Occupancy:**
   A strong indicator of road congestion and traffic density.

2. **Avg_Lanes_Count:**
   The number of active lanes directly impacts the traffic flow capacity.

3. **Avg_Speed:**
   Traffic speed provides insights into road efficiency and congestion levels.

4. **Peak_Hour_Flag:**
   Identifies critical time windows (rush hours) that see a significant increase in flow.

5. **Route:**
   Represents expressways (US-101, I-80, I-280, I-880), where traffic flow patterns vary.

8. Future Work

1. Hyperparameter Tuning

- There is significant potential to **enhance accuracy** and **reduce errors** further.
- While XGBoost showed strong performance, it was trained using basic parameters. A thorough **hyperparameter tuning** process can improve its predictive power.

2. Incorporating Additional Features

To better capture real-world complexities, additional data can be included:

- **Incident Data Integration:** Incorporating data on **accidents**, lane blockages, and diversions can help understand their impact on traffic flow.
- **Weather Data:** Features such as **precipitation** and **temperature** can provide deeper insights into traffic behavior during adverse conditions.
- **FasTrak Lane Availability:** Including **high-occupancy toll (HOT) lane usage** can help determine its role in managing congestion.

3. Enhancing Temporal Analysis

- **Time-Series Modeling:** Incorporating advanced techniques like ARIMA, LSTM, or other temporal models to predict **traffic trends** over days, weeks, or months.
- **Hourly Congestion Patterns:** Extending the analysis to **seasonal trends** and identifying specific hourly patterns for actionable insights.

9.Appendix

R CODE

#R code for data preprocessing

```
---
title: "MATH 748- Term Project"
output:
  html_document:
    df_print: paged
  html_notebook: default
  word_document: default
---
```

Load Required Libraries

```{r}
# Load required libraries
library(dplyr)      # For data manipulation
library(lubridate)  # For working with dates and timestamps
library(readr)      # For reading and writing CSV files
library(tidyr)      # For tidying and reshaping data
library(ggplot2)    # For visualizations (if needed later)
library(stringr)    # For string manipulation
```

Define Paths and Settings

```{r}
# Paths for raw and processed data
traffic_raw_path <- "D:/Masters/Semesters/Fall 2024/Math 748/Project/Datasets/raw/pems_traffic_volumes/"

incidents_raw_path <- "D:/Masters/Semesters/Fall 2024/Math 748/Project/Datasets/raw/chp_incidents/"

# Define column names for traffic data
traffic_col_names <- c("Timestamp", "Station", "District", "Route", "Direction_of_Travel",
                "Lane_Type", "Station_Length", "Samples", "Percent_Observed",
                "Total_Flow", "Avg_Occupancy", "Avg_Speed", "Delay_V35",
                "Delay_V40", "Delay_V45", "Delay_V50", "Delay_V55", "Delay_V60",
                "Lane1_Flow", "Lane1_Avg_Occ", "Lane1_Avg_Speed",
                "Lane2_Flow", "Lane2_Avg_Occ", "Lane2_Avg_Speed",
                "Lane3_Flow", "Lane3_Avg_Occ", "Lane3_Avg_Speed",
                "Lane4_Flow", "Lane4_Avg_Occ", "Lane4_Avg_Speed",
                "Lane5_Flow", "Lane5_Avg_Occ", "Lane5_Avg_Speed",
                "Lane6_Flow", "Lane6_Avg_Occ", "Lane6_Avg_Speed",
                "Lane7_Flow", "Lane7_Avg_Occ", "Lane7_Avg_Speed",
                "Lane8_Flow", "Lane8_Avg_Occ", "Lane8_Avg_Speed")
```

~~~
# Columns to retain for traffic data
traffic_selected_columns <- c("Timestamp", "Station", "District", "Route", "Direction_of_Travel",
                              "Lane_Type", "Samples", "Percent_Observed", "Total_Flow",
                              "Avg_Occupancy", "Avg_Speed", "Lanes_Count")  # Added Lanes_Count

# Define column names for CHP Incidents data
incident_col_names <- c("Incident_ID", "CC_Code", "Incident_Number", "Timestamp", "Description",
                        "Location", "Area", "Zoom_Map", "TB_xy", "Latitude", "Longitude",
                        "District", "County_FIPS_ID", "City_FIPS_ID", "Freeway_Number",
                        "Freeway_Direction", "State_Postmile", "Absolute_Postmile",
                        "Severity", "Duration")

# Define selected columns based on the analysis
incident_selected_columns <- c("Incident_ID", "Timestamp", "District", "Freeway_Number",
                               "Freeway_Direction", "Severity", "Duration",
                               "Location", "Latitude", "Longitude")

traffic_processed_path <- "D:/Masters/Semesters/Fall 2024/Math
748/Project/Datasets/processed/traffic_data_processed.csv"

incidents_processed_path <- "D:/Masters/Semesters/Fall 2024/Math
748/Project/Datasets/processed/chp_incidents_processed.csv"


```
~~~

Process Traffic Data

~~~
```{r}
# Define a function to process traffic data files
process_traffic_file <- function(file) {
  data <- read.csv(file, header = FALSE, stringsAsFactors = FALSE)

  # Assign column names
  colnames(data) <- traffic_col_names

  # Calculate the number of lanes
  data <- data %>%
    mutate(
    Lanes_Count = rowSums(!is.na(select(., starts_with("Lane") & ends_with("_Flow"))))
   )%>%
    select(all_of(traffic_selected_columns)) %>%
    mutate(
      Timestamp = trimws(Timestamp),  # Remove leading/trailing spaces
      Timestamp = as.POSIXct(Timestamp, format = "%m/%d/%Y %H:%M:%S", tz = "UTC"),  # Convert to datetime
      Timestamp = floor_date(Timestamp, unit = "hour"),  # Round to the nearest hour
      District = as.integer(District),
~~~

````
    Route = factor(Route, levels = c(101, 80, 280, 880), labels = c("US-101", "I-80", "I-280", "I-880")),
    Lane_Type = as.factor(Lane_Type),
    Direction_of_Travel = as.factor(Direction_of_Travel),
    Samples = as.integer(Samples),
    Percent_Observed = as.numeric(Percent_Observed),
    Total_Flow = as.integer(Total_Flow),
    Avg_Occupancy = as.numeric(Avg_Occupancy),
    Avg_Speed = as.numeric(Avg_Speed)
   ) %>%
   filter(District == 4 & Route %in% c("US-101", "I-80", "I-280", "I-880"))  # Filter for relevant data

  return(data)
}

# Process all traffic files
traffic_files <- list.files(path = traffic_raw_path, pattern = "*.txt", full.names = TRUE, recursive = TRUE)
traffic_data <- do.call(rbind, lapply(traffic_files, process_traffic_file))

# Save processed traffic data
#write.csv(traffic_data %>% mutate(Timestamp = format(Timestamp, "%Y-%m-%d %H:%M:%S")), file
=traffic_processed_path, row.names = FALSE)

# Preview processed data
head(traffic_data)
str(traffic_data)

```
````

Process CHP Incidents Data

````
```{r}
# Set paths for raw data and output processed data
incidents_raw_path <- "D:/Masters/Semesters/Fall 2024/Math 748/Project/Datasets/raw/chp_incidents/"

# Define column names for CHP Incidents data
incident_col_names <- c("Incident_ID", "CC_Code", "Incident_Number", "Timestamp", "Description",
                "Location", "Area", "Zoom_Map", "TB_xy", "Latitude", "Longitude",
                "District", "County_FIPS_ID", "City_FIPS_ID", "Freeway_Number",
                "Freeway_Direction", "State_Postmile", "Absolute_Postmile",
                "Severity", "Duration")

# Define selected columns based on the analysis
incident_selected_columns <- c("Incident_ID", "Timestamp", "District", "Freeway_Number",
                     "Freeway_Direction", "Severity", "Duration",
                     "Location", "Latitude", "Longitude")

# Define a function to process each CHP Incidents file
process_incidents_file <- function(file) {
````

```
  data <- read.csv(file, header = FALSE, stringsAsFactors = FALSE)
  colnames(data) <- incident_col_names
  data <- data %>%
    select(all_of(incident_selected_columns)) %>%
    mutate(
      Timestamp = trimws(Timestamp), # Remove leading/trailing spaces
      Timestamp = as.POSIXct(Timestamp, format = "%m/%d/%Y %H:%M:%S", tz = "UTC"), # Convert to datetime
      Timestamp = floor_date(Timestamp, unit = "hour"), # Round to the nearest hour
      District = as.integer(District),
      Freeway_Number = factor(Freeway_Number, levels = c(101, 80, 280, 880), labels = c("US-101", "I-80", "I-280",
"I-880")),
      Freeway_Direction = as.factor(Freeway_Direction),
      Severity = as.character(Severity),
      Duration = as.numeric(Duration),
      Latitude = as.numeric(Latitude),
      Longitude = as.numeric(Longitude)
    ) %>%
    filter(District == 4 &  Freeway_Number %in% c("US-101", "I-80", "I-280", "I-880"))
  return(data)
}

# Process all CHP Incidents files
incident_files <- list.files(path = incidents_raw_path, pattern = "*.txt", full.names = TRUE, recursive = TRUE)
incidents_data <- do.call(rbind, lapply(incident_files, process_incidents_file))


# Export processed CHP Incidents data
#write.csv(incidents_data %>% mutate(Timestamp = format(Timestamp, "%Y-%m-%d %H:%M:%S")), file =
incidents_processed_path, row.names = FALSE)
# Preview processed data
head(incidents_data)
str(incidents_data)

```

Missing Value Analysis

```{r}

# Function to calculate and visualize missing values
analyze_missing_data <- function(data, dataset_name) {
  missing_summary <- data.frame(
    Column = names(data),
    Missing_Percentage = sapply(data, function(col) mean(is.na(col)) * 100)
  )
```

```
  print(paste("Missing Data Summary for", dataset_name))
  print(missing_summary)

  # Visualize missing data (optional, if using ggplot2)
  if ("ggplot2" %in% rownames(installed.packages())) {
    library(ggplot2)
    ggplot(missing_summary, aes(x = reorder(Column, -Missing_Percentage), y = Missing_Percentage)) +
      geom_bar(stat = "identity", fill = "steelblue") +
      coord_flip() +
      labs(
        title = paste("Missing Data Analysis for", dataset_name),
        x = "Columns",
        y = "Percentage Missing"
      ) +
      theme_minimal()
  }

  return(missing_summary)
}

# Analyze missing data for both datasets
traffic_missing_summary <- analyze_missing_data(traffic_data, "Traffic Data")
incidents_missing_summary <- analyze_missing_data(incidents_data, "Incidents Data")

```

Total_Flow, Avg_Occupancy and Avg_Speed

```{r}
# Step 1: Remove all NA values from Timestamp,Total_Flow, Avg_Speed
traffic_data <- traffic_data %>%
  filter(!is.na(Timestamp))

traffic_data <- traffic_data %>%
  filter(Avg_Speed > 0 & !is.na(Avg_Speed))

traffic_data <- traffic_data %>%
  filter(Total_Flow>0 & !is.na(Total_Flow))



# Step 3: Summarize missing value analysis and final dataset structure
missing_summary <- traffic_data %>%
  summarise(
    Total_Flow_Missing_Percentage = sum(is.na(Total_Flow)) / n() * 100,
    Avg_Occupancy_Missing_Percentage = sum(is.na(Avg_Occupancy)) / n() * 100,
    Avg_Speed_Missing_Percentage = sum(is.na(Avg_Speed)) / n() * 100
  )
```

````
# Display missing value summary
print("Summary of Missing Values After Handling:")
print(missing_summary)

# Summarize the final dataset
print("Final Dataset Summary:")
print(summary(traffic_data[, c("Total_Flow", "Avg_Occupancy", "Avg_Speed")]))

# Save the updated traffic & incidents data to the processed file
write.csv(traffic_data%>% mutate(Timestamp = format(Timestamp, "%Y-%m-%d %H:%M:%S")),
     traffic_processed_path,
     row.names = FALSE)

write.csv(incidents_data %>% mutate(Timestamp = format(Timestamp, "%Y-%m-%d %H:%M:%S")),
     file = incidents_processed_path,
     row.names = FALSE)

```



```{r}
str(traffic_data)
head(traffic_data)
str(incidents_data)
head(incidents_data)
```
#summarize data
```{r}
# Ensure proper data types
traffic_data <- traffic_data %>%
 mutate(
  Timestamp = as.POSIXct(Timestamp, format = "%Y-%m-%d %H:%M:%S"),
  Route = as.factor(Route),
  Direction_of_Travel = as.factor(Direction_of_Travel),
  Lane_Type = as.factor(Lane_Type)
 )

# Add additional features
traffic_data <- traffic_data %>%
 mutate(
  Day_Name = weekdays(Timestamp),
  Hour = hour(Timestamp),
  Weekend_Flag = as.factor(ifelse(Day_Name %in% c("Saturday", "Sunday"), 1, 0)),
  Peak_Hour_Flag = as.factor(ifelse(Hour %in% c(5,6,7, 8, 9, 10,11,12,13,14,15,16, 17, 18,19), 1, 0))
 )
````

```
# Summarize the data
traffic_summary <- traffic_data %>%
  group_by(Station, Day_Name, Hour, Route, Direction_of_Travel, Lane_Type) %>%
  summarize(
    Avg_Lanes_Count = mean(Lanes_Count, na.rm = TRUE),  # Average lane count
    Avg_Total_Occupancy = mean(Avg_Occupancy, na.rm = TRUE),  # Total occupancy sum
    Avg_Speed = mean(Avg_Speed, na.rm = TRUE),  # Average speed
    Avg_Total_Flow = mean(Total_Flow, na.rm = TRUE),  # Total flow sum
    Weekend_Flag = first(Weekend_Flag),  # Consistent flag per group
    Peak_Hour_Flag = first(Peak_Hour_Flag),
    .groups = "drop"  # Prevent grouped data frame in result
  )

# Convert categorical variables to factors (ensuring consistency)
traffic_summary <- traffic_summary %>%
  mutate(
    Day_Name = as.factor(Day_Name),
    Route = as.factor(Route),
    Direction_of_Travel = as.factor(Direction_of_Travel),
    Lane_Type = as.factor(Lane_Type)
  )

# Preview the summarized data
cat("Preview of Summarized Data:\n")
print(head(traffic_summary))
cat("\nSummary of Data:\n")
print(summary(traffic_summary))

# Data Quality Checks
cat("\nTotal Records in Summarized Data: ", nrow(traffic_summary), "\n")
cat("Checking for Missing Values:\n")
print(sapply(traffic_summary, function(x) sum(is.na(x))))

# Save summarized data for further analysis
summarized_path <- "D:/Masters/Semesters/Fall 2024/Math
748/Project/Datasets/processed/traffic_summary.csv"
write.csv(traffic_summary, summarized_path, row.names = FALSE)

cat("\nSummarized data saved to: ", summarized_path, "\n")
```

R code for modelling

```{r}
library(dplyr)
```

````
library(caret)
library(rpart)
library(ggplot2)
library(randomForest)
```


```{r}
# Load required libraries
library(dplyr)
library(ggplot2)
library(caret)
library(lubridate)

# Load the summarized dataset
summarized_path <- "D:/Masters/Semesters/Fall 2024/Math
748/Project/Datasets/processed/traffic_summary.csv"

traffic_summary <- read.csv(summarized_path)

# Ensure proper data types
traffic_summary <- traffic_summary %>%
  select(-Lane_Type) %>%  # Remove Lane_Type
  mutate(
    Station = as.factor(Station),
    Day_Name = as.factor(Day_Name),
    Hour = as.factor(Hour),  # Hour as factor for time-based trends
    Route = as.factor(Route),
    Direction_of_Travel = as.factor(Direction_of_Travel),
    Weekend_Flag = as.factor(Weekend_Flag),
    Peak_Hour_Flag = as.factor(Peak_Hour_Flag),
    Avg_Lanes_Count = as.numeric(Avg_Lanes_Count),
    Avg_Total_Occupancy = as.numeric(Avg_Total_Occupancy),
    Avg_Speed = as.numeric(Avg_Speed),
    Avg_Total_Flow = as.numeric(Avg_Total_Flow)
  )

# Preview dataset
cat("Preview of Traffic Summary Dataset:\n")
print(head(traffic_summary))

# Check data structure
cat("\nData Structure:\n")
str(traffic_summary)

# # Data Quality Checks
````

````
# cat("\nChecking for Missing Values:\n")
# missing_values <- sapply(traffic_summary, function(x) sum(is.na(x)))
# print(missing_values)
#
# # Visualize Response Variable
# cat("\nVisualizing the Response Variable (Total Flow):\n")
# ggplot(traffic_summary, aes(x = Avg_Total_Flow)) +
#   geom_histogram(bins = 30, fill = "steelblue", color = "black") +
#   labs(title = "Distribution of Total Flow", x = "Total Flow", y = "Frequency") +
#   theme_minimal()
#
# cat("\nVisualizing the Relationship between Avg Speed and Total Flow:\n")
# ggplot(traffic_summary, aes(x = Avg_Speed, y = Avg_Total_Flow)) +
#   geom_point(alpha = 0.5) +
#   labs(title = "Avg Speed vs Total Flow", x = "Avg Speed", y = "Total Flow") +
#   theme_minimal()

```



#dataset splitting
```{r}

# Initialize a list to store results
results <- list()


# Data splitting: 70% Training, 30% Testing
set.seed(1)  # For reproducibility
train_index <- createDataPartition(traffic_summary$Avg_Total_Flow, p = 0.7, list = FALSE)
train_data <- traffic_summary[train_index, ]
test_data <- traffic_summary[-train_index, ]
#Confirm split
cat("Training set size:", nrow(train_data), "\n")
cat("Testing set size:", nrow(test_data), "\n")
```

#linear regression
```{r}
# Linear Regression
linear_model <- lm(Avg_Total_Flow ~ ., data = train_data)
linear_preds <- predict(linear_model, newdata = test_data)
results$Linear_Regression <- postResample(linear_preds, test_data$Avg_Total_Flow)

# Model Summary
cat("\nLinear Regression Summary:\n")
````

~~~~
print(summary(linear_model))

# Predictor Importance
linear_importance <- as.data.frame(summary(linear_model)$coefficients)
linear_importance <- linear_importance[order(abs(linear_importance[, "Estimate"]), decreasing = TRUE), ]
cat("\nLinear Regression - Top Predictors:\n")
print(head(linear_importance))

```


#Decision Tree
```{r}
# Decision Tree
tree_model <- rpart(Avg_Total_Flow ~ ., data = train_data, method = "anova")
tree_preds <- predict(tree_model, newdata = test_data)
results$Decision_Tree <- postResample(tree_preds, test_data$Avg_Total_Flow)

# Model Summary
cat("\nDecision Tree Summary:\n")
print(summary(tree_model))

# Predictor Importance
tree_importance <- as.data.frame(tree_model$variable.importance)
tree_importance <- tree_importance[order(tree_importance[, 1], decreasing = TRUE), , drop = FALSE]
colnames(tree_importance) <- "Importance"
cat("\nDecision Tree - Top Predictors:\n")
print(head(tree_importance))

```



#XGBoost
```{r}
library(xgboost)  # For XGBoost

# Prepare data for XGBoost
xgb_train_data <- model.matrix(Avg_Total_Flow ~ . - 1, data = train_data)
xgb_test_data <- model.matrix(Avg_Total_Flow ~ . - 1, data = test_data)

xgb_dtrain <- xgb.DMatrix(data = xgb_train_data, label = train_data$Avg_Total_Flow)
xgb_dtest <- xgb.DMatrix(data = xgb_test_data, label = test_data$Avg_Total_Flow)

# Train XGBoost Model
xgb_params <- list(objective = "reg:squarederror", eta = 0.1, max_depth = 6)
xgb_model <- xgb.train(params = xgb_params, data = xgb_dtrain, nrounds = 100)
~~~~

~~~
# Predict and Evaluate
xgb_preds <- predict(xgb_model, newdata = xgb_dtest)
results$XGBoost <- postResample(xgb_preds, test_data$Avg_Total_Flow)

# Model Summary
cat("\nXGBoost Summary:\n")
xgb_summary <- xgb.importance(model = xgb_model)
print(head(xgb_summary))

# Predictor Importance
cat("\nXGBoost - Top Predictors:\n")
print(head(xgb_summary))

```
```{r}
# Ridge and Lasso Regression using glmnet
library(glmnet)

# Prepare data for Ridge and Lasso Regression
x_train <- model.matrix(Avg_Total_Flow ~ . - 1, data = train_data)
y_train <- train_data$Avg_Total_Flow
x_test <- model.matrix(Avg_Total_Flow ~ . - 1, data = test_data)
y_test <- test_data$Avg_Total_Flow

# Ridge Regression
ridge_model <- glmnet(x_train, y_train, alpha = 0)
ridge_preds <- predict(ridge_model, s = 0.01, newx = x_test)
results$Ridge_Regression <- postResample(ridge_preds, y_test)

# Ridge Model Summary
cat("\nRidge Regression Summary:\n")
ridge_coefs <- as.matrix(coef(ridge_model, s = 0.01))  # Convert to regular matrix
ridge_importance <- data.frame(
  Predictor = rownames(ridge_coefs),
  Importance = abs(ridge_coefs[, 1])
)
ridge_importance <- ridge_importance[order(-ridge_importance$Importance), ]
ridge_importance <- ridge_importance[ridge_importance$Importance > 0, ]  # Remove zero coefficients
cat("\nRidge Regression - Top Predictors:\n")
print(head(ridge_importance))

# Lasso Regression
lasso_model <- glmnet(x_train, y_train, alpha = 1)
lasso_preds <- predict(lasso_model, s = 0.01, newx = x_test)
results$Lasso_Regression <- postResample(lasso_preds, y_test)
~~~

```
# Lasso Model Summary
cat("\nLasso Regression Summary:\n")
lasso_coefs <- as.matrix(coef(lasso_model, s = 0.01))  # Convert to regular matrix
lasso_importance <- data.frame(
  Predictor = rownames(lasso_coefs),
  Importance = abs(lasso_coefs[, 1])
)
lasso_importance <- lasso_importance[order(-lasso_importance$Importance), ]
lasso_importance <- lasso_importance[lasso_importance$Importance > 0, ]  # Remove zero coefficients
cat("\nLasso Regression - Top Predictors:\n")
print(head(lasso_importance))


```

#Evaluate Results

```{r}
# Compile Results
results_df <- do.call(rbind, results)
colnames(results_df) <- c("RMSE", "R-Squared", "MAE")
rownames(results_df) <- c("Linear Regression", "Decision Tree", "XGBoost", "Ridge Regression", "Lasso Regression")

# Print Results
print("Model Performance Comparison:")
print(results_df)

```