Appendices

Appendix A: Full Regression Results Table

Table 6: Linear Regression Coefficients

Parameter	Estimate (Binary)	SE (Binary)	Estimate (Multivariate)	SE (Multivariate)
Intercept	212.98	1.11	216.70	1.15
RapidRide	-13.94	2.74	-105.32	39.29
Distance Traveled	NA	NA	36.09	1.13
Traffic (Day/Hour)	NA	NA	8.84	1.10
Traffic (Location)	NA	NA	-6.77	1.16
Population Density	NA	NA	19.08	1.66
Route Ridership	NA	NA	-7.66	2.56
Percentage White	NA	NA	0.08	1.43
Median HHI	NA	NA	5.46	1.92
avg_traffic_dayhour:spatial_congestion	NA	NA	0.10	1.05
shape_dist_traveled:avg_traffic_dayhour	NA	NA	4.69	1.06
shape_dist_traveled:spatial_congestion	NA	NA	-11.55	1.21
rapid_ride:factor(g_weekday)2	NA	NA	31.59	10.70
rapid_ride:factor(g_weekday)3	NA	NA	44.80	11.33
rapid_ride:factor(g_weekday)4	NA	NA	51.83	11.36
rapid_ride:factor(g_weekday)5	NA	NA	41.49	11.88
rapid_ride:factor(g_weekday)6	NA	NA	48.57	12.80
rapid_ride:factor(g_weekday)7	NA	NA	55.07	11.24
rapid_ride:factor(g_hr)4	NA	NA	29.19	84.83
rapid_ride:factor(g_hr)5	NA	NA	3.48	46.31
rapid_ride:factor(g_hr)6	NA	NA	-25.96	35.60
rapid_ride:factor(g_hr)7	NA	NA	-16.51	35.34
rapid_ride:factor(g_hr)8	NA	NA	5.18	37.95
rapid_ride:factor(g_hr)9	NA	NA	31.98	39.68
rapid_ride:factor(g_hr)10	NA	NA	33.74	38.16
rapid_ride:factor(g_hr)11	NA	NA	39.44	37.87
rapid_ride:factor(g_hr)12	NA	NA	29.30	37.40
rapid_ride:factor(g_hr)13	NA	NA	64.96	38.99
rapid_ride:factor(g_hr)14	NA	NA	62.70	38.66
rapid_ride:factor(g_hr)15	NA	NA	83.67	40.35
rapid_ride:factor(g_hr)16	NA	NA	85.16	40.76
rapid_ride:factor(g_hr)17	NA	NA	95.47	42.09
rapid_ride:factor(g_hr)18	NA	NA	90.48	41.11
rapid_ride:factor(g_hr)19	NA	NA	65.12	40.18
rapid_ride:factor(g_hr)20	NA	NA	88.39	37.38
rapid_ride:factor(g_hr)21	NA	NA	68.81	35.74
rapid_ride:factor(g_hr)22	NA	NA	66.97	36.68
rapid_ride:factor(g_hr)23	NA	NA	111.42	35.60
rapid_ride:factor(g_hr)24	NA	NA	63.80	37.13
rapid_ride:spatial_congestion	NA	NA	-9.01	3.34
rapid_ride:route_ridership	NA	NA	9.70	2.35
rapid_ride:avg_traffic_dayhour	NA	NA	-19.89	7.85

Appendix B: Python Code

Code and datasets for this project can be found on GitHub: https://github.com/Peter-Silverstein/bus-delay-modeling

```
# <---------- API PULL FOR REAL-TIME GTFS DATA ---------->

# Used with AWS Lambda for automation
import requests
from google.transit import gtfs_realtime_pb2
from google.protobuf.json_format import MessageToDict
import boto3
import json
from datetime import datetime
import logging

def lambda_handler(event, context):
    # Define API details
    API_KEY = "2c97496e-e814-4cd6-bb23-14413a2a480d"
    FEED_URL = f"""
    http://api.pugetsound.onebusaway.org/api/gtfs_realtime/trip-
    updates-for-agency/1.pb?key={API_KEY}"""

    logger = logging.getLogger()
    logger.setLevel(logging.INFO)

    # Main workflow
    try:
        # Fetch and parse feed
        feed_content = fetch_gtfs_realtime(FEED_URL)
        parsed_feed = parse_gtfs_feed(feed_content)

        # Extract relevant data
        trips = extract_trip_data(parsed_feed)

        # Convert trips to JSON string
        json_data = json.dumps(trips, indent=4)

        # Setting name for file
        current_time = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
        OBJECT_NAME = f"realtime_trip_data_{current_time}.json"
        BUCKET_NAME = "gtfs-data-run1"

        # Save JSON data to S3
        s3_client = boto3.client('s3')
```

```
        s3_client.put_object(
            Bucket=BUCKET_NAME,
            Key=OBJECT_NAME,
            Body=json_data,
            ContentType='application/json'
        )

        logger.info(f"""
        Data successfully saved to S3 bucket '{BUCKET_NAME}' as '{OBJECT_NAME}'
        """)
    except Exception as e:
        logger.error(f"Error occurred: {e}")

# Function to fetch GTFS-Realtime feed
def fetch_gtfs_realtime(feed_url):
    response = requests.get(feed_url)
    response.raise_for_status()  # Raise an exception for HTTP errors
    return response.content

# Function to parse GTFS-Realtime feed
def parse_gtfs_feed(feed_content):
    feed = gtfs_realtime_pb2.FeedMessage()
    feed.ParseFromString(feed_content)
    return MessageToDict(feed)

# Function to extract trip data
def extract_trip_data(parsed_feed):
    trip_data = []
    for entity in parsed_feed.get("entity", []):
        if "tripUpdate" in entity:
            trip_update = entity["tripUpdate"]
            trip_id = trip_update.get("trip", {}).get("tripId", "")
            route_id = trip_update.get("trip", {}).get("routeId", "")
            agency_id = trip_update.get("trip", {}).get("agencyId", "")
            stop_time_updates = trip_update.get("stopTimeUpdate", [])

            for stop_time_update in stop_time_updates:
                stop_id = stop_time_update.get("stopId", "")
                arrival_time = stop_time_update.get("arrival", {}).get(
                  "time", "")
                departure_time = stop_time_update.get("departure", {}).get(
                  "time", "")

                # Append relevant data as a dictionary
```

```
                trip_data.append({
                    "trip_id": trip_id,
                    "route_id": route_id,
                    "agency_id": agency_id,
                    "stop_id": stop_id,
                    "arrival_time": arrival_time,
                    "departure_time": departure_time,
                })
    return trip_data
```

```
# <---------- PULLING JSON FROM AWS S3 STORAGE ---------->

import boto3
import os
import pandas as pd
import json

def download_all_files(bucket_name, local_dir):
    s3 = boto3.client('s3')
    paginator = s3.get_paginator('list_objects_v2')
    pages = paginator.paginate(Bucket = bucket_name)

    for page in pages:
        if 'Contents' in page:
            for obj in page['Contents']:
                key = obj['Key']
                local_file_path = os.path.join(local_dir, key)

                # Create directories if they don't exist already
                os.makedirs(os.path.dirname(local_file_path), exist_ok = True)

                # Download the file
                s3.download_file(bucket_name, key, local_file_path)

# Use function
bucket_name = 'gtfs-data-run1'
local_dir = 'raw_from_awsS3'
download_all_files(bucket_name, local_dir)

print("All files downloaded")

# Function to extract trip data
def extract_trip_data(parsed_feed):
    trip_data = []
```

```python
    for entity in parsed_feed.get("entity", []):
        if "tripUpdate" in entity:
            trip_update = entity["tripUpdate"]
            trip_id = trip_update.get("trip", {}).get("tripId", "")
            route_id = trip_update.get("trip", {}).get("routeId", "")
            agency_id = trip_update.get("trip", {}).get("agencyId", "")
            stop_time_updates = trip_update.get("stopTimeUpdate", [])

            for stop_time_update in stop_time_updates:
                stop_id = stop_time_update.get("stopId", "")
                arrival_time = stop_time_update.get("arrival", {}).get(
                  "time", "")
                departure_time = stop_time_update.get("departure", {}).get(
                  "time", "")

                # Append relevant data as a dictionary
                trip_data.append({
                    "trip_id": trip_id,
                    "route_id": route_id,
                    "agency_id": agency_id,
                    "stop_id": stop_id,
                    "arrival_time": arrival_time,
                    "departure_time": departure_time,
                })
    return trip_data

# Function to iterate over the folder
def json_to_df(folder_path):
    all_data = [] # Empty list to store all our dataframes

    # Iterate through all JSON files in the folder
    for filename in os.listdir(folder_path):
        if filename.endswith('.json'):
            file_path = os.path.join(folder_path, filename)

            # Open and load the JSON file
            with open(file_path, 'r') as f:
                parsed_feed = json.load(f)

            # Extract trip data
            trip_data = extract_trip_data(parsed_feed)

            # Convert to dataframe
            df = pd.DataFrame(trip_data)
```

```
            # Add a column for the source filename
            df['source_file'] = filename

            # Append to the list of dataframes
            all_data.append(df)

    # Combine all dataframes into one
    combined_df = pd.concat(all_data, ignore_index = True)

    return combined_df

# Apply the function
local_dir = 'raw_from_awsS3'
combined_df = json_to_df(local_dir)

print(combined_df.head(30))
```

```
# <---------- SAVING TO POSTGRESQL DATABASE FOR FUTURE USE ---------->

import os
import pandas as pd
import numpy as np
import json
import psycopg2
from datetime import datetime
from datetime import timedelta
from datetime import timezone
from psycopg2 import sql
from io import StringIO
from zoneinfo import ZoneInfo

# Extract Trip Data function
def extract_trip_data(parsed_feed):
    trip_data = []
    # Check if parsed_feed is a list
    if isinstance(parsed_feed, list):
        for item in parsed_feed:
            # Directly access the trip data from each item
            trip_id = item.get('trip_id', '')
            route_id = item.get('route_id', '')
            stop_id = item.get('stop_id', '')
            arrival_time = item.get('arrival_time', '')
            departure_time = item.get('departure_time', '')
```

```
            # Append relevant data as a dictionary
            trip_data.append({
                'trip_id': trip_id,
                'route_id': route_id,
                'stop_id': stop_id,
                'arrival_time': arrival_time,
                'departure_time': departure_time,
            })
    return trip_data

# Function to iterate over the folder and process JSON files
def json_to_df(folder_path):
    all_data = []  # List to store all dataframes

    # Iterate through all JSON files in the folder
    for filename in os.listdir(folder_path):
        if filename.endswith('.json'):
            file_path = os.path.join(folder_path, filename)

            # Open and load the JSON file
            with open(file_path, 'r') as f:
                parsed_feed = json.load(f)

            # Extract trip data using the updated function
            trip_data = extract_trip_data(parsed_feed)

            # Convert to dataframe
            df = pd.DataFrame(trip_data)

            # Add a column for the source filename
            df['source_file'] = filename

            # Append to the list of dataframes
            all_data.append(df)

    # Combine all dataframes into one
    combined_df = pd.concat(all_data, ignore_index=True)

    return combined_df

# Function to convert Unix time to date and seconds after midnight in PST
def convert_to_date_and_time(unix_time):
    if unix_time:
        utc_time = datetime.fromtimestamp(int(unix_time), tz = timezone.utc)
```

```
        # Adjust for PST (UTC-8)
        pst_time = utc_time.astimezone(ZoneInfo("America/Los_Angeles"))
        combined_datetime = pst_time.strftime('%Y-%m-%d %H:%M:%S')
        return combined_datetime
    return None

# Application
local_dir = 'raw_from_awsS3'
combined_df = json_to_df(local_dir)

# Cleaning Data

# Converting columns to more useful types
combined_df = combined_df.astype({"source_file": "string"})

# Replace empty strings with NaN in some columns
columns_to_replace = ['trip_id', 'route_id', 'stop_id']
combined_df[columns_to_replace] = combined_df[columns_to_replace].replace(
  '', pd.NA)

# Remove rows where the arrival_time column has an empty value
combined_df = combined_df.dropna(subset=['arrival_time'])

# Apply the conversion function to arrival and departure times
combined_df["arrival_datetime"] = combined_df["arrival_time"].apply(
  convert_to_date_and_time)
combined_df["departure_datetime"] = combined_df["departure_time"].apply(
  convert_to_date_and_time)

# Use the source_file column to extract the day/time of the pull
combined_df['pull_datetime'] = combined_df['source_file'].str.extract(
  r'_(\d{4}-\d{2}-\d{2}_\d{2}-\d{2}-\d{2})\.json$')
combined_df[['pull_date', 'pull_time']] = combined_df[
  'pull_datetime'].str.split('_', expand=True)
combined_df['pull_time'] = combined_df['pull_time'].str.replace('-', ':')
combined_df['pull_datetime'] = pd.to_datetime(
  combined_df['pull_date'] + ' ' + combined_df['pull_time'],
  errors='coerce', utc = True)
combined_df['pull_datetime'] = combined_df['pull_datetime'].dt.tz_convert(
  ZoneInfo("America/Los_Angeles"))

# Compare the pull and arrival day/time to check if the time is forecasted
combined_df["projection"] = np.where((
  combined_df["arrival_datetime"] > combined_df["pull_datetime"]), 1, 0)
```

```python
# Removing the source_file,
combined_df = combined_df.drop(["arrival_time",
                                "departure_time",
                                "source_file",
                                "pull_date",
                                "pull_time"],
                                axis = 1)

# Re-typing the columns; replacing NA values with None
combined_df.replace({pd.NA: None, pd.NaT: None}, inplace=True)

combined_df = combined_df.astype({
    "trip_id": "string",
    "route_id": "string",
    "stop_id": "string",
    "arrival_datetime": "datetime64[ns, America/Los_Angeles]",
    "departure_datetime": "datetime64[ns, America/Los_Angeles]",
    "pull_datetime": "datetime64[ns, America/Los_Angeles]"
})

# Print to check
print(combined_df.dtypes)
print(combined_df.head())

# Writing data to PostgreSQL (!!)
# Function to create the table if it does not exist
def create_table_if_not_exists(conn):
    with conn.cursor() as cur:
        cur.execute("""
            CREATE TABLE IF NOT EXISTS sea_gtfs_data (
                unique_id SERIAL PRIMARY KEY,
                trip_id TEXT,
                route_id TEXT,
                stop_id TEXT,
                arrival_datetime TIMESTAMPTZ,
                departure_datetime TIMESTAMPTZ,
                pull_datetime TIMESTAMPTZ,
                projection BOOLEAN
            )
        """)
        conn.commit()

# Function to bulk load a DataFrame into the PostgreSQL table
def bulk_insert_dataframe(conn, df, table_name):
```

```python
    buffer = StringIO()
    df.to_csv(buffer, index=False, header=False)
    buffer.seek(0)

    columns = ["trip_id", "route_id", "stop_id",
               "arrival_datetime", "departure_datetime",
               "pull_datetime", "projection"]

    with conn.cursor() as cur:
        cur.copy_expert(
            sql.SQL("COPY {} ({}) FROM STDIN WITH CSV").format(
                sql.Identifier(table_name),
                sql.SQL(', ').join(map(sql.Identifier, columns))
            ),
            buffer
        )
        conn.commit()

# Main script
# MODIFY THESE LINES IN YOUR CODE
# In the main script section:

def main():
    # Database connection parameters
    db_params = {
        "dbname": "sea-gtfs-data",
        "user": "postgres",
        "password": "Parkour",
        "host": "localhost",
        "port": 5432
    }

    # Connect to the database
    conn = psycopg2.connect(**db_params)

    try:
        # Create table if not exists
        create_table_if_not_exists(conn)

        # NEW: Process data in batches
        chunk_size = 50000  # Adjust based on your system's capacity
        total_rows = len(combined_df)

        for start in range(0, total_rows, chunk_size):
```

```
            end = min(start + chunk_size, total_rows)
            chunk = combined_df.iloc[start:end]

            print(f"Processing rows {start+1}-{end} of {total_rows}")

            # NEW: Clear buffers after each chunk
            with conn:
                with conn.cursor() as cur:
                    buffer = StringIO()
                    chunk.to_csv(buffer, index=False, header=False, columns=[
                        "trip_id", "route_id", "stop_id",
                        "arrival_datetime", "departure_datetime",
                        "pull_datetime", "projection"
                    ])
                    buffer.seek(0)

                    copy_sql = sql.SQL("""
                        COPY sea_gtfs_data (
                            trip_id, route_id, stop_id,
                            arrival_datetime, departure_datetime,
                            pull_datetime, projection
                        ) FROM STDIN WITH CSV
                    """)

                    cur.copy_expert(copy_sql, buffer)
                    conn.commit()

                    # Explicitly clean up resources
                    buffer.close()
                    del buffer

    finally:
        conn.close()


if __name__ == "__main__":
    main()
```

Appendix C: R Code

```
# Loading Libraries

# General Use
```

```r
library(tidyverse)
library(ggplot2)
library(here)
library(patchwork)
library(modelsummary)
library(knitr)
library(keyring)
library(lubridate)
library(data.table)

# Modeling
library(stan4bart)
library(bartCause)
library(rstanarm)
library(bayesplot)

# PostgreSQL
library(DBI)
library(RPostgres)

# GIS and Mapping
library(sf)
library(tmap)
```

```r
# <---------- GENERAL DATA LOADING, CLEANING, MANAGEMENT ---------->

# Loading Data
con <- dbConnect(RPostgres::Postgres(),
                 dbname = "sea-gtfs-data",
                 host = "localhost",
                 port = 5432,
                 user = "postgres",
                 password = "Parkour")

gtfs_realtime <- dbReadTable(con, "sea_gtfs_data")
gtfs_realtime <- tibble(gtfs_realtime)
```

```r
# Filtering dataset to work with my spatial congestion dataset +
# include RapidRide C, D, G lines

# Current CRS NAD83
kc_route_shp <- st_read(here("Predictor Data Sets",
            "KCMetro_Transit_Lines",
            "Transit_Routes_for_King_County_Metro___transitroute_line.shp")) %>%
```

```r
  select(ROUTE_ID, geometry) %>%
  distinct(ROUTE_ID, .keep_all = TRUE)

routes <- read.csv(here("Predictor Data Sets",
                        "gtfs-static-files/routes.txt")) %>%
  filter(agency_id == 1) %>% # Filtering to only include King County Metro
  select(route_id, route_short_name) %>%
  mutate(rapid_ride = case_when(
    str_detect(route_short_name, "Line") ~ 1,
    TRUE ~ 0
  )) %>%
  replace_na(list(rapid_ride = 0)) %>%
  mutate(route_id = as.numeric(route_id),
         rapid_ride = as.factor(rapid_ride)) %>%
  select(route_id, rapid_ride, route_short_name)
```

```r
# Joining
routes_shp <- kc_route_shp %>%
  left_join(routes,
            by = c("ROUTE_ID" = "route_id")) %>%
  st_transform(crs = 2285)

ylims <- c(184191.9, 271524.6)
xlims <- c(1250336, 1293480)
box_coords <- tibble(x = xlims, y = ylims) %>%
  st_as_sf(coords = c("x", "y")) %>%
  st_set_crs(2285)

bounding_box <- st_bbox(box_coords) %>% st_as_sfc()

routes_subset <- st_filter(routes_shp, bounding_box, .predicate = st_within)
routes_inbb <- routes_subset$ROUTE_ID

gtfs_realtime <- gtfs_realtime %>%
  filter(route_id %in% routes_inbb)
```

```r
# Filtering out duplicates and future projections
gtfs_main <- gtfs_realtime %>%
  filter(projection == FALSE) %>%
  distinct(trip_id, stop_id, arrival_datetime, .keep_all = TRUE) %>%
  select(trip_id, route_id, stop_id, arrival_datetime, departure_datetime,
         pull_datetime) %>%
  mutate(trip_id = as.factor(trip_id),
         route_id = as.factor(route_id),
```

```r
         stop_id = as.factor(stop_id)) %>%
  mutate(arrival_datetime = with_tz(arrival_datetime, "America/Los_Angeles"),
         departure_datetime = with_tz(departure_datetime, "America/Los_Angeles"),
         pull_datetime = with_tz(pull_datetime, "America/Los_Angeles"))

print(paste("Number of rows reduced from", nrow(gtfs_realtime), "to",
            nrow(gtfs_main)))
```

```r
# Helper function for dealing with hh > 23 in schedule file
roll_over <- function(time_str) {
  # Split the time string into hours, minutes, seconds
  parts <- as.numeric(strsplit(time_str, ":")[[1]])
  total_seconds <- parts[1] * 3600 + parts[2] * 60 + parts[3]
  # Use modulo operator to get seconds within a day
  remainder <- total_seconds %% 86400
  # Convert remainder seconds back into hh:mm:ss format
  sprintf("%02d:%02d:%02d",
          remainder %/% 3600,
          (remainder %% 3600) %/% 60,
          remainder %% 60)
}
```

```r
# Importing scheduled stop times
stop_times <- read_csv(here("Predictor Data Sets",
                            "gtfs-static-files/stop_times.txt")) %>%
  select(trip_id, arrival_time, departure_time, stop_id, stop_sequence,
         shape_dist_traveled) %>%
  mutate(trip_id = as.factor(trip_id),
         stop_id = as.factor(stop_id),
         arrival_time = as.character(arrival_time),
         departure_time = as.character(departure_time)) %>%
  rename(sched_arrival_time = arrival_time,
         sched_departure_time = departure_time) %>%
   mutate(
    sched_arrival_time = sapply(sched_arrival_time, roll_over),
    sched_departure_time = sapply(sched_departure_time, roll_over),
    sched_arrival_time = hms::as_hms(sched_arrival_time),
    sched_departure_time = hms::as_hms(sched_departure_time)
  ) %>%
  distinct(trip_id, stop_id, .keep_all = TRUE)

# Joining to main
gtfs_main_withdelays <- gtfs_main %>%
  left_join(stop_times, by = c("trip_id" = "trip_id",
```

```r
                              "stop_id" = "stop_id")) %>%
  mutate(actual_arrival_time = hms::as_hms(format(with_tz(
    arrival_datetime, "America/Los_Angeles"), "%H:%M:%S")),
         actual_departure_time = hms::as_hms(format(with_tz(
           departure_datetime, "America/Los_Angeles"), "%H:%M:%S")),
         date = as.POSIXct(format(with_tz(
           arrival_datetime, "America/Los_Angeles"), "%Y-%m-%d"))) %>%
  mutate(arrival_delay = as.numeric(
    actual_arrival_time - sched_arrival_time)) %>%
  mutate(arrival_delay = ifelse(
    arrival_delay < 80000, arrival_delay,
    arrival_delay - 86400
    )) %>%
  select(date,
         trip_id,
         route_id,
         stop_id,
         sched_arrival_time,
         sched_departure_time,
         actual_arrival_time,
         actual_departure_time,
         arrival_delay,
         stop_sequence,
         shape_dist_traveled,
         pull_datetime)
```

```r
# Loading data
congestion_temporal <- read_csv(here("Predictor Data Sets",
                                "Traffic_Count_Studies_by_Hour_Bins-2.csv"))

# Converting times, setting up day/hour lookup
congestion_dayhour <- congestion_temporal %>%
  filter(TOTAL > 0) %>%
  mutate(datetime = as.POSIXct(ADD_DTTM,
                               format = "%m/%d/%Y %I:%M:%S %p",
                               tz = "America/Los_Angeles")) %>%
  filter(datetime > as.POSIXct("01-01-2015 00:00:00",
                               format = "%m-%d-%Y %H:%M:%S",
                               tz = "America/Los_Angeles")) %>%
  filter(datetime < as.POSIXct("01-31-2020 00:00:00",
                               format = "%m-%d-%Y %H:%M:%S",
                               tz = "America/Los_Angeles") |
           datetime > as.POSIXct("12-31-2021 23:59:59",
                               format = "%m-%d-%Y %H:%M:%S",
```

```r
                                   tz = "America/Los_Angeles")) %>%
  group_by(WEEKDAY) %>%
  summarize(HR01 = mean(HR01_TOTAL),
            HR02 = mean(HR02_TOTAL),
            HR03 = mean(HR03_TOTAL),
            HR04 = mean(HR04_TOTAL),
            HR05 = mean(HR05_TOTAL),
            HR06 = mean(HR06_TOTAL),
            HR07 = mean(HR07_TOTAL),
            HR08 = mean(HR08_TOTAL),
            HR09 = mean(HR09_TOTAL),
            HR10 = mean(HR10_TOTAL),
            HR11 = mean(HR11_TOTAL),
            HR12 = mean(HR12_TOTAL),
            HR13 = mean(HR13_TOTAL),
            HR14 = mean(HR14_TOTAL),
            HR15 = mean(HR15_TOTAL),
            HR16 = mean(HR16_TOTAL),
            HR17 = mean(HR17_TOTAL),
            HR18 = mean(HR18_TOTAL),
            HR19 = mean(HR19_TOTAL),
            HR20 = mean(HR20_TOTAL),
            HR21 = mean(HR21_TOTAL),
            HR22 = mean(HR22_TOTAL),
            HR23 = mean(HR23_TOTAL),
            HR24 = mean(HR24_TOTAL)) %>%
  mutate(WEEKDAY_NAME = case_when(
    WEEKDAY == 1 ~ "Monday",
    WEEKDAY == 2 ~ "Tuesday",
    WEEKDAY == 3 ~ "Wednesday",
    WEEKDAY == 4 ~ "Thursday",
    WEEKDAY == 5 ~ "Friday",
    WEEKDAY == 6 ~ "Saturday",
    WEEKDAY == 7 ~ "Sunday")) %>%
  mutate(AVG_VOL = select(., HR01:HR24) %>% rowMeans(na.rm = TRUE)) %>%
  relocate(WEEKDAY_NAME, .after = WEEKDAY) %>%
  relocate(AVG_VOL, .after = WEEKDAY_NAME)

# Setting up longer format
congestion_dh_longer <- congestion_dayhour %>%
  select(WEEKDAY_NAME, HR01:HR24) %>%
  pivot_longer(cols = -WEEKDAY_NAME,
               names_to = "HOUR",
               values_to = "avg_traffic_dayhour") %>%
```

```
  mutate(HOUR = as.numeric(gsub("[^0-9]","", HOUR)))
```

```
# Joining
gtfs_main_withcongestion <- gtfs_main_withdelays %>%
  mutate(WEEKDAY = weekdays(date),
         HR = (as.numeric(sched_arrival_time) %/% 3600) + 1) %>%
  left_join(select(.data = congestion_dayhour, WEEKDAY_NAME, AVG_VOL),
            by = c("WEEKDAY" = "WEEKDAY_NAME")) %>%
  left_join(congestion_dh_longer,
            by = c("WEEKDAY" = "WEEKDAY_NAME",
                   "HR" = "HOUR")) %>%
    rename("weekday" = "WEEKDAY",
         "hr" = "HR",
         "avg_traffic_day" = "AVG_VOL")

seq_standardized <- read.csv(here("Predictor Data Sets",
                                  "gtfs-static-files/stop_times.txt")) %>%
  select(trip_id, stop_sequence) %>%
  group_by(trip_id) %>%
  arrange(stop_sequence) %>%
  mutate(new_seq = row_number()) %>%
  ungroup() %>%
  arrange(trip_id, stop_sequence) %>%
  mutate(stop_sequence = as.numeric(stop_sequence),
         trip_id = as.factor(trip_id))

gtfs_main_final <- gtfs_main_withcongestion %>%
  inner_join(seq_standardized,
            by = c("stop_sequence" = "stop_sequence",
                   "trip_id" = "trip_id"))

gtfs_main_final <- gtfs_main_final[, c("date",
                                       "route_id",
                                       "trip_id",
                                       "stop_id",
                                       "sched_arrival_time",
                                       "sched_departure_time",
                                       "actual_arrival_time",
                                       "actual_departure_time",
                                       "arrival_delay",
                                       "stop_sequence",
                                       "new_seq",
                                       "shape_dist_traveled",
                                       "pull_datetime",
```

```
                                                   "weekday",
                                                   "hr",
                                                   "avg_traffic_day",
                                                   "avg_traffic_dayhour")]

gtfs_main_final
```

```
routes <- read.csv(here("Predictor Data Sets",
                        "gtfs-static-files/routes.txt")) %>%
  filter(agency_id == 1) %>% # Filtering to only include King County Metro
  select(route_id, route_short_name) %>%
  mutate(rapid_ride = case_when(
    str_detect(route_short_name, "Line") ~ 1,
    TRUE ~ 0
  )) %>%
  replace_na(list(rapid_ride = 0)) %>%
  mutate(route_id = as.factor(route_id),
         rapid_ride = as.factor(rapid_ride)) %>%
  select(route_id, rapid_ride)

gtfs_main_withrapidride <- gtfs_main_final %>%
  left_join(routes,
            by = "route_id")

stop_predictors <- gtfs_main_withrapidride %>%
  select(route_id, trip_id, stop_id, rapid_ride, arrival_delay,
         shape_dist_traveled, weekday, hr, avg_traffic_dayhour) %>%
  filter(!is.na(arrival_delay)) %>%
  mutate(weekday = as.factor(weekday))

write_csv(stop_predictors, "../predictor_tables/stop_predictors.csv")
```

```
set.seed(50)

# Train/Test Partition
subset_size <- 100000
subset_indices <- sample(seq_len(nrow(stop_predictors)), size = subset_size)

subset <- stop_predictors[subset_indices, ]
rest <- stop_predictors[-subset_indices, ]
```

```
# Loading trip data (directionality)
trips <- read.csv("../Predictor Data Sets/gtfs-static-files/trips.txt") %>%
```

```r
  select(route_id, trip_id, direction_id, shape_id) %>%
  filter(route_id %in% routes_inbb)

# Loading spatial data for routes
shapes <- read.csv("../Predictor Data Sets/gtfs-static-files/shapes.txt") %>%
  arrange(shape_id, shape_pt_sequence) %>%
  st_as_sf(coords = c("shape_pt_lon", "shape_pt_lat"), crs = 4326) %>%
  group_by(shape_id) %>%
  summarise(do_union = FALSE) %>%
  st_cast("LINESTRING") %>%
  st_transform(2285)

# Loading stop sequence for each trip
stop_times <- read.csv("../Predictor Data Sets/gtfs-static-files/
                       stop_times.txt") %>%
  select(trip_id, stop_id, shape_dist_traveled, arrival_time) %>%
  mutate(shape_dist_traveled = shape_dist_traveled) %>%
  mutate(stop_id = as.character(stop_id),
         trip_id = as.character(trip_id)) %>%
  distinct(stop_id, trip_id,
           .keep_all = TRUE)

subset_directionality <- subset %>%
  left_join(trips,
            by = "trip_id")

# Return Direction-Conscious Linestring for TripIDs
subset_withshapes <- subset_directionality %>%
  left_join(shapes,
             by = c("shape_id" = "shape_id")) %>%
  mutate(geometry = case_when(
    direction_id == 1 ~ st_reverse(geometry),
    TRUE ~ geometry),
    trip_id = as.character(trip_id)
  ) %>%
  st_as_sf() %>%
  st_set_crs(st_crs(shapes))

# Clipping route lines
subset_clipped <- subset_withshapes %>%
  mutate(trip_id = as.character(trip_id),
         shape_dist_traveled = case_when(
           shape_dist_traveled == 0 ~ 1,
           TRUE ~ shape_dist_traveled
```

```r
        )) %>%
  select(!route_id.y) %>%
  rename("route_id" = "route_id.x") %>%
  mutate(
    total_length = as.numeric(st_length(geometry)),
    # Normalize distance to [0,1] fraction
    to_fraction = pmin(shape_dist_traveled / total_length, 1)
  ) %>%
  rowwise() %>%
  mutate(
    geometry = lwgeom::st_linesubstring(
      geometry,
      from = 0,
      to = to_fraction,
      normalize = FALSE
    )
  ) %>%
  ungroup() %>%
  st_as_sf() %>%
  st_set_crs(st_crs(2285))

get_weighted_traffic_average <- function(geometry) {
  buffered_route <- st_buffer(geometry, dist = 1)
  intersection <- st_intersection(congestion_spatial, buffered_route)
  intersection_line <- st_collection_extract(intersection, "LINESTRING")

  # Note this is avg of traffic data we have, so not all routes
  # have complete coverage
  processed_intersection <- intersection_line %>%
    # Calculate length of each segment
    mutate(length = units::set_units(
      st_length(geometry), "ft", mode = "standard")) %>%
    # Filtering out any segment less than 5ft in length to remove noise
    filter(length > units::set_units(
      5, "ft", mode = "standard"))

  total_length <- sum(processed_intersection$length)

  processed_intersection <- processed_intersection %>%
    mutate(wgt_traffic = AWDT * (length / total_length))

  segment_traffic <- as.numeric(sum(processed_intersection$wgt_traffic))
  return(segment_traffic)
}
```

```r
# CRS is NAD83
congestion_spatial <- st_read(here("Predictor Data Sets",
                                   "2018_Traffic_Flow_Counts-shp",
                                   "2018_Traffic_Flow_Counts.shp")) %>%
  select(AWDT, geometry) %>%
  st_transform(crs = 2285)

subset_spatialcongestion <- subset_clipped %>%
  rowwise() %>%
  mutate(spatial_congestion = get_weighted_traffic_average(geometry)) %>%
  ungroup()

subset_fixed <- subset_spatialcongestion %>%
  left_join(trips %>% select(trip_id, route_id),
            by = "trip_id") %>%
  select(!route_id.x) %>%
  rename("route_id" = "route_id.y") %>%
  mutate(route_id = as.factor)

# Ridership
get_weighted_acs <- function(route_id) {
   route_shape <- routes_subset %>%
     filter(ROUTE_ID == route_id)
  # approximate 0.5 mile buffer (in feet)
  buffered_route <- st_buffer(route_shape, dist = 2640)

  # Filter ACS polygons to include only ones with over 50% within buffer
   blocks_filtered <- kcacs_blocks %>%
     filter(lengths(st_intersects(., buffered_route)) > 0) %>%
     rowwise() %>%
     mutate(
       inter_geom = list(st_intersection(geometry, buffered_route)),
       inter_area = {
         ig <- inter_geom[]
         if(length(ig) == 0 || all(st_is_empty(ig))) {
           0
           } else {
             sum(st_area(ig))
             }
         },
       total_area = st_area(geometry),
       overlap_ratio = as.numeric(inter_area / total_area)
       ) %>%
     ungroup() %>%
```

```r
    filter(overlap_ratio >= 0.5)

  # Weighted average
  total_population <- sum(blocks_filtered$tot_popE)
  total_buffer_area <- sum(st_area(blocks_filtered$geometry))

  blocks_filtered <- blocks_filtered %>%
    mutate(wgt_ridership = transp_mthd_public_perc * (
      tot_popE/total_population),
           wgt_percwhite = white_perc * (tot_popE/total_population),
           wgt_medHHI = median_HHI * (tot_popE/total_population))

  list(
    pop_density = total_population/total_buffer_area,
    route_ridership = as.double(sum(blocks_filtered$wgt_ridership)),
    perc_white = as.double(sum(blocks_filtered$wgt_percwhite)),
    median_hhi = as.double(sum(blocks_filtered$wgt_medHHI))
  )
}
```

```r
# Using keyring package to keep my API key hidden
tidycensus_api_key <- key_get(service = "tidycensus_API",
                              username = "my_tidycensus")
census_api_key(tidycensus_api_key)

ACSlist <- load_variables(2022, "acs5", cache = TRUE)

# Projection is NAD83(!!)
kingcounty_acs_blocks <- get_acs(state = "WA",
                                 county = "King",
                                 geography = "block group",
                                 variables = c(tot_pop = "B01003_001",
                                              transp_basetotal = "B08134_001",
                                              transp_mthd_public = "B08134_061",
                                              race_base = "B02001_001",
                                              race_white = "B02001_002",
                                              median_HHI = "B19013_001"),
                                 geometry = TRUE,
                                 keep_geo_vars = TRUE,
                                 year = 2023,
                                 output = "wide") %>%
  filter(ALAND != 0) %>% # Filter tracts that are 100% water
  mutate(GEOID = as.double(GEOID))
```

```r
kcacs_blocks <- kingcounty_acs_blocks %>%
  mutate(ALAND_miles = ALAND/2589988) %>% # Converting sq meters to sq miles
  mutate(transp_mthd_public_perc = transp_mthd_publicE / transp_basetotalE,
         pop_density = tot_popE / ALAND_miles,
         white_perc = race_whiteE / race_baseE,
         median_HHI = median_HHIE) %>%
  filter(!is.na(median_HHI)) %>%
  select(tot_popE,
         pop_density,
         transp_mthd_public_perc,
         white_perc,
         median_HHI,
         geometry) %>%
  st_transform(crs = 2285)

route_demos <- routes_subset %>%
  rowwise() %>%
  mutate(
     acs_data = list(get_weighted_acs(ROUTE_ID))
   ) %>%
  mutate(
    pop_density = acs_data$pop_density, # POP PER SQUARE FOOT
    route_ridership = acs_data$route_ridership,
    perc_white = acs_data$perc_white,
    median_hhi = acs_data$median_hhi
  ) %>%
  ungroup() %>%
  select(!acs_data, !geometry) %>%
  filter(!is.na(ROUTE_ID))

route_demos <- route_demos %>%
  select(ROUTE_ID, pop_density, route_ridership, perc_white, median_hhi) %>%
  rename("route_id" = "ROUTE_ID") %>%
  mutate(pop_density = as.double(pop_density),
         route_id = as.factor(route_id))

subset_withacs <- subset_fixed %>%
  select(!geometry) %>%
  tibble() %>%
  left_join(route_demos,
            by = "route_id")
```

```
# Standardize function
standardize <- function(x) {
  (x - mean(x, na.rm = TRUE)) / sd(x, na.rm = TRUE)
}

subset_standardized <- subset_withacs %>%
  select(route_id,
         stop_id,
         trip_id,
         rapid_ride,
         arrival_delay,
         shape_dist_traveled,
         avg_traffic_dayhour,
         spatial_congestion,
         pop_density,
         route_ridership,
         perc_white,
         median_hhi,
         weekday,
         hr) %>%
  mutate(across(c("shape_dist_traveled",
                  "avg_traffic_dayhour",
                  "spatial_congestion",
                  "pop_density",
                  "route_ridership",
                  "perc_white",
                  "median_hhi"),
                standardize)) %>%
  mutate(abs_dev = abs(arrival_delay))
```

```
# Setting some definitions
weekdays <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
weekends <- c("Saturday", "Sunday")
peak <- c(6, 7, 8, 9, 16, 17, 18, 19) # weekdays only
non_peak <- c(1, 2, 3, 4, 5, 10, 11, 12, 13, 14, 15, 20, 21, 22, 23, 24)

data_final <- subset_standardized %>%
  mutate(g_weekend = case_when(
    weekday %in% weekdays ~ 0,
    weekday %in% weekends ~ 1
  ),
  g_peak = case_when(
    weekday %in% weekdays & hr %in% peak ~ 1,
    TRUE ~ 0
```

```
  )) %>%
  dplyr::select(route_id,
         stop_id,
         trip_id,
         rapid_ride,
         arrival_delay,
         abs_dev,
         shape_dist_traveled,
         avg_traffic_dayhour,
         spatial_congestion,
         pop_density,
         route_ridership,
         perc_white,
         median_hhi,
         weekday,
         hr,
         g_peak,
         g_weekend) %>%
  mutate(
    weekday = case_when(
      weekday == "Monday" ~ 1,
      weekday == "Tuesday" ~ 2,
      weekday == "Wednesday" ~ 3,
      weekday == "Thursday" ~ 4,
      weekday == "Friday" ~ 5,
      weekday == "Saturday" ~ 6,
      weekday == "Sunday" ~ 7
    )) %>%
  mutate(rapid_ride = as.factor(rapid_ride),
         weekday = as.numeric(weekday),
         hr = as.numeric(hr)) %>%
  rename("g_routeid" = "route_id",
         "g_weekday" = "weekday",
         "g_hr" = "hr")

write_csv(data_final, "../predictor_tables/final_data.csv")
```

```
# Train/Test Split
set.seed(50)

data_final <- read_csv("../predictor_tables/final_data.csv") %>%
  mutate(
    abs_dev = ifelse(
    abs_dev < 80000, abs_dev,
```

```r
    abs(abs_dev - 86400)
    ))

# Train/Test Partition
smp_size <- floor(0.5 * nrow(data_final))
train_indices <- sample(seq_len(nrow(data_final)), size = smp_size)

train <- data_final[train_indices, ]
test <- data_final[-train_indices, ]

write_csv(train, "../cloud-scripts/train_data.csv")
write_csv(test, "../cloud-scripts/test_data.csv")
```

```r
# <---------- LINEAR REGRESSION MODELING ---------->

train_df <- read_csv(here("cloud-scripts","partial-pooling","train_data.csv"))
test_df <- read_csv(here("cloud-scripts","partial-pooling","test_data.csv"))
df_combo <- rbind(train_df, test_df[test_df$rapid_ride == 1, ])

# A simple comparison of means
fit_binary <- stan_glm(abs_dev ~ rapid_ride,
                       data = df_combo,
                       refresh = FALSE,
                       cores = 4)

# A model with interactions between related variables and
# allow treatment effects to vary by group
fit_int <- stan_glm(abs_dev ~ rapid_ride +
                         shape_dist_traveled +
                         avg_traffic_dayhour +
                         spatial_congestion +
                         pop_density +
                         route_ridership +
                         perc_white +
                         median_hhi +
                         spatial_congestion:avg_traffic_dayhour +
                         shape_dist_traveled:avg_traffic_dayhour +
                         shape_dist_traveled:spatial_congestion +
                         rapid_ride:factor(g_weekday) +
                         rapid_ride:factor(g_hr) +
                         rapid_ride:spatial_congestion +
                         rapid_ride:route_ridership +
                         rapid_ride:avg_traffic_dayhour,
                       data = df_combo,
```

```r
                        refresh = FALSE,
                        cores = 4)

# Table of Coefficients
print(fit_binary, digits = 5)
print(fit_int, digits = 5)

save(fit_binary, file = "../models/fit_binary.RData")
save(fit_int, file = "../models/fit_int.RData")
```

```r
# <---------- BART MODELING (USED GOOGLE CLOUD & DOCKER) ---------->

# Vanilla BART

# Loading libraries
library(dbarts)
library(bartCause)
library(readr)
library(processx)
library(dplyr)

print(installed.packages())

# Get environment variables set by Vertex AI
model_dir <- "gs://bus-delay-modeling-stan4bart-models/models/"
print(paste("AIP_MODEL_DIR:", model_dir))

#remove trailing slashes.
model_dir <- gsub("/+$", "", model_dir)

if (model_dir == "") {
  model_dir <- "models"  # Fallback for local testing
  dir.create(model_dir, recursive = TRUE, showWarnings = FALSE)
} else {
  #create local model folder to save model to before copying to gs.
  dir.create("local_model_dir", recursive = TRUE, showWarnings = FALSE)
}

# Loading and preparing data
df <- read_csv("train_data.csv")
df_test <- read_csv("test_data.csv")

df_combo <- rbind(df, df_test[df_test$rapid_ride == 1, ])
```

```r
df_cf <- df_combo
df_cf$rapid_ride <- ifelse(df_cf$rapid_ride == 1, 0, 1)

# Training the stan4bart model
fit <- bart2(abs_dev ~
                rapid_ride +
                shape_dist_traveled +
                avg_traffic_dayhour +
                spatial_congestion +
                pop_density +
                route_ridership +
                perc_white +
                median_hhi +
                g_weekday +
                g_hr,
              data = df_combo,
              test = df_cf,
              keepTrees = TRUE,
              seed = 50
              )

# Save the model to a local directory.
save(fit, file = file.path("local_model_dir",
                           "vanillabart_rapidride_model_cf_combo.RData"))

# Copy the model to Google Cloud Storage using gsutil.
if(model_dir != "models"){
  local_file_path <- file.path("local_model_dir",
                               "vanillabart_rapidride_model_cf_combo.RData")
  gs_destination <- file.path(model_dir,
                              "vanillabart_rapidride_model_cf_combo.RData")

  # Run gsutil cp command
  result <- processx::run("gsutil", c("cp", local_file_path, gs_destination))

  if (result$status == 0) {
    cat("Model training completed and saved to", gs_destination, "\n")
  } else {
    cat("Error copying model to Google Cloud Storage.\n")
    cat("gsutil output:\n", rawToChar(result$stdout), "\n")
    cat("gsutil error:\n", rawToChar(result$stderr), "\n")
  }

} else {
```

```r
  cat("Model training completed and saved locally to", file.path(
    "local_model_dir", "vanillabart_rapidride_model_cf_combo.RData"), "\n")
}
```

```r
# <---------- FINAL ANALYSIS AND VISUALIZATIONS ---------->
library(tidyverse)
library(ggplot2)
library(here)
library(sf)
library(tmap)
library(rstanarm)
library(bartCause)
library(knitr)
library(skimr)
library(corrplot)
library(posterior)
library(bayesplot)
library(dbarts)
library(broom.mixed)
library(kableExtra)
library(extrafont)

extrafont::loadfonts(device = "pdf", quiet = TRUE)
font_import()

# Loading combined final dataset
data <- read_csv(here("predictor_tables","final_data.csv")) %>%
  mutate(
    g_routeid = as.factor(g_routeid),
    stop_id = as.factor(stop_id),
    trip_id = as.factor(trip_id),
    rapid_ride = as.numeric(rapid_ride),
    arrival_delay = as.numeric(arrival_delay),
    abs_dev = as.numeric(abs_dev),
    shape_dist_traveled = as.numeric(shape_dist_traveled),
    avg_traffic_dayhour = as.numeric(avg_traffic_dayhour),
    spatial_congestion = as.numeric(spatial_congestion),
    pop_density = as.numeric(pop_density),
    route_ridership = as.numeric(route_ridership),
    perc_white = as.numeric(perc_white),
    median_hhi = as.numeric(median_hhi),
    g_weekday = as.numeric(g_weekday),
    g_hr = as.numeric(g_hr),
    g_peak = as.factor(g_peak),
```

```r
    g_weekend = as.factor(g_weekend)
  ) %>%
  mutate(
    abs_dev = ifelse(
    abs_dev < 80000, abs_dev,
    abs(abs_dev - 86400)
    ))

train_df <- read_csv(here("cloud-scripts","partial-pooling","train_data.csv"))
test_df <- read_csv(here("cloud-scripts","partial-pooling","test_data.csv"))
df_combo <- rbind(train_df, test_df[test_df$rapid_ride == 1, ])

load("models/fit_binary.RData")
load("models/fit_int.RData")
load("models/models-vanillabart_rapidride_model_cf_combo.RData")
fit_cf_combo <- fit
load("models/models-vanillabart_rapidride_model.RData")
```

```r
# Loading Data
routes_inscope <- unique(data$g_routeid)
stops_inscope <- unique(data$stop_id)
routes_rapidride <- data %>%
  filter(rapid_ride == 1)
routes_rapidride <- unique(routes_rapidride$g_routeid)

routes_shp <- st_read(here("Predictor Data Sets",
          "KCMetro_Transit_Lines",
          "Transit_Routes_for_King_County_Metro___transitroute_line.shp")) %>%
  filter(ROUTE_ID %in% routes_inscope)

routes_rapidride_shp <- routes_shp %>%
  filter(ROUTE_ID %in% routes_rapidride)

stops_shp <- st_read(here("Predictor Data Sets",
                "KCMetro_Transit_Stops",
                "Transit_Stops_for_King_County_Metro___transitstop_point.shp"))
```

```r
# Mapping
route_map <- tm_shape(routes_shp) + tm_lines(col = "#1D7D7A", lwd = 1) +
  tm_shape(routes_rapidride_shp) + tm_lines(col = "#D71D24", lwd = 2) +
  tm_basemap("CartoDB.PositronNoLabels") +
  tm_add_legend(type = "line",
                col = c("#1D7D7A", "#D71D24"),
                labels = c("Standard Bus", "RapidRide"),
```

```
                title = "Route Type",
                lwd = 2) +
  tm_layout(fontfamily = "Times New Roman",
            legend.text.size = 0.8,
            legend.title.size = 1)
tmap_save(route_map, filename = "route_map.png")
```

```
# Creating descriptive statistics table
data_descriptives = df_combo %>%
  select(
    rapid_ride,
    abs_dev,
    shape_dist_traveled,
    avg_traffic_dayhour,
    spatial_congestion,
    pop_density,
    route_ridership,
    perc_white,
    median_hhi,
    g_weekday,
    g_hr
  )

overall_descriptives <- skim(data_descriptives) %>%
  select(skim_variable,
         numeric.mean,
         numeric.sd,
         numeric.p0,
         numeric.p25,
         numeric.p50,
         numeric.p75,
         numeric.p100) %>%
  rename(
    "Variable" = "skim_variable",
    "Mean" = "numeric.mean",
    "Std Dev" = "numeric.sd",
    "Min" = "numeric.p0",
    "25%" = "numeric.p25",
    "Median" = "numeric.p50",
    "75%" = "numeric.p75",
    "Max" = "numeric.p100",
  ) %>%
  mutate(Variable = recode(Variable,
                           "rapid_ride" = "RapidRide",
```

```r
                            "abs_dev" = "Absolute Deviation",
                            "shape_dist_traveled" = "Distance Traveled",
                            "avg_traffic_dayhour" = "Traffic (Day/Hour)",
                            "spatial_congestion" = "Traffic (Location)",
                            "pop_density" = "Population Density",
                            "route_ridership" = "Route Ridership",
                            "perc_white" = "Percentage White",
                            "median_hhi" = "Median HHI",
                            "g_weekday" = "Weekday",
                            "g_hr" = "Hour"))
```

```r
# Assessing balance and overlap for causal inference
# From https://github.com/gperrett/stan4bart-study/blob/master/get_balance.R
# Linked in Dorie et al 2022
get_balance <- function(rawdata, treat,estimand="ATT"){
  if(missing(rawdata)) stop("rawdata is required")
  if(missing(treat)) stop("treatment vector (treat) is required")
  cat("Balance diagnostics assume that the estimand is the",estimand,"\n")
  #
  #raw.dat <- data.frame(rawdata, treat = treat)
  covnames <- colnames(rawdata)
  if (is.null(covnames)){
    cat("No covariate names provided.  Generic names will be generated.")
    covnames = paste("v",c(1:ncol(rawdata)),sep="")
  }
  K <- length(covnames)
  diff.means <- matrix(NA, K, 5)
  var.t <- numeric(K)
  var.c <- numeric(K)
  std.denom <- numeric(K)
  binary <- rep(1,K)

  for (i in 1:K) {
    # separate means by group
    diff.means[i, 1] <- mean(rawdata[treat==1, i])
    diff.means[i, 2] <- mean(rawdata[treat==0, i])
    # separate variances by group == only used as input to calculations below
    var.t[i] <- var(rawdata[(treat == 1), i])
    var.c[i] <- var(rawdata[(treat == 0), i])
    # denominator in standardized difference calculations
    if(estimand=="ATE"){std.denom[i] <- sqrt((var.t[i]+var.c[i])/2)}
    else{
      std.denom[i] <- ifelse(estimand=="ATT",sqrt(var.t[i]),sqrt(var.c[i]))
    }
```

```
    # difference in means
    diff.means[i, 3] <- diff.means[i, 1] - diff.means[i, 2]
    # standardized difference in means (sign intact)
    diff.means[i, 4] <- abs(diff.means[i, 3]/std.denom[i])
    if(length(unique(rawdata[,covnames[i]]))>2){
      binary[i] = 0
      diff.means[i, 5] <- sqrt(var.c[i]/var.t[i])
    }
  }

  dimnames(diff.means) <- list(covnames, c("Treat", "Control", "Difference",
                                           "abs.std.diff", "Ratio"))
  return(diff.means)
}

# Setting up data (covariates in a matrix, treatment vector)
# Removed factor vars
X <- as.matrix(data_descriptives %>% select(!rapid_ride))
y <- data_descriptives$rapid_ride

# Running the function
balance_table <- get_balance(rawdata = X, treat = y, estimand = "ATT")
balance_table <- as_tibble(balance_table) %>%
  dplyr::select(Treat, Control, Difference, Ratio)
rownames(balance_table) <- c("Absolute Deviation", "Distance Traveled",
                          "Traffic (Day/Hour)", "Traffic (Location)",
                          "Population Density", "Route Ridership",
                          "Percentage White", "Median HHI",
                          "Weekday", "Hour")
```

```
# Correlation table
corr_table <- data_descriptives %>%
  rename("RapidRide" = "rapid_ride",
         "Absolute Deviation" = "abs_dev",
         "Distance Traveled" = "shape_dist_traveled",
         "Traffic (Day/Hour)" = "avg_traffic_dayhour",
         "Traffic (Location)" = "spatial_congestion",
         "Population Density" = "pop_density",
         "Route Ridership" = "route_ridership",
         "Percentage White" = "perc_white",
         "Median HHI" = "median_hhi",
         "Weekday" = "g_weekday",
         "Hour" = "g_hr")
corr <- cor(corr_table)
```

```r
actuals <- test_df$abs_dev

# Coefficients for Linear Regressions
binary_tidy <- tidy(fit_binary)
int_tidy <- tidy(fit_int)

merged_df <- full_join(
  binary_tidy %>% select(term, estimate, std.error),
  int_tidy %>% select(term, estimate, std.error),
  by = "term",
  suffix = c("_model1", "_model2")
)

# RMSE
binary_pred <- predict(fit_binary, newdata = test_df)
int_pred <- predict(fit_int, newdata = test_df)
bart_pred <- fitted(fit, type = "ev", sample = "test")

binary_rmse <- sqrt(mean((binary_pred - actuals)^2))
int_rmse <- sqrt(mean((int_pred - actuals)^2))
bart_rmse <- sqrt(mean((bart_pred - actuals)^2))

rmse_df <- tibble(
  Model = c("Binary Linear", "Multivariate Linear",
            "Bayesian Additive Regression Trees"),
  RMSE = c(binary_rmse, int_rmse, bart_rmse)
)
```

```r
# Assessing R-Hat for BART model
sigma_draws <- as_draws_array(fit_cf_combo$sigma)
rhat_values <- round(posterior::rhat(sigma_draws), 2)

df_combo <- rbind(train_df, test_df[test_df$rapid_ride == 1, ])
rr_indices <- df_combo$rapid_ride == 1

# BART SATT
factual_pred <- extract(fit_cf_combo, type = "ev", sample = "train")
counterfactual_pred <- extract(fit_cf_combo, type = "ev", sample = "test")

treated_factual_pred <- factual_pred[, rr_indices]
treated_counterfactual_pred <- counterfactual_pred[, rr_indices]

ind_effects <- treated_factual_pred - treated_counterfactual_pred
```

```r
satt_dist <- rowMeans(ind_effects)

satt_est <- median(satt_dist)
satt_ci <- quantile(satt_dist, probs = c(0.025, 0.975))

satt_df <- tibble(
  Estimate = round(satt_est, 2),
  Lower95 = round(satt_ci[1], 2),
  Upper95 = round(satt_ci[2], 2)
  )
```

```r
# Multivariate SATT
df_combo_rr_f <- df_combo[df_combo$rapid_ride == 1, ]
df_combo_rr_cf <- df_combo_rr_f
df_combo_rr_cf$rapid_ride <- 0

int_factual_pred <- posterior_predict(fit_int, newdata = df_combo_rr_f)
int_counterfactual_pred <- posterior_predict(fit_int, newdata = df_combo_rr_cf)

int_ind_effects <- int_factual_pred - int_counterfactual_pred

int_satt_dist <- rowMeans(int_ind_effects)

int_satt_est <- median(int_satt_dist)
int_satt_ci <- quantile(int_satt_dist, probs = c(0.025, 0.975))

satt_df <- tibble(
  Estimate = c(round(satt_est, 2), round(int_satt_est, 2)),
  Lower95 = c(round(satt_ci[1], 2), round(int_satt_ci[1], 2)),
  Upper95 = c(round(satt_ci[2], 2), round(int_satt_ci[2], 2))
  )
```

```r
# Histplot
satt_dist_df <- tibble(satt_dist)
int_satt_dist_sample <- sample(int_satt_dist, size = 2000)
satt_histplot <- ggplot(data = satt_dist_df, aes(x = satt_dist)) +
  geom_density(aes(fill = "BART"), alpha = 0.75) +
  geom_density(aes(x = int_satt_dist_sample, fill = "Multivariate Linear"),
              alpha = 0.75) +
  geom_vline(xintercept = satt_est, linetype = "dashed",
            color = "#2596be", linewidth = 0.75) +
  geom_vline(xintercept = int_satt_est, linetype = "dashed",
            color = "#fd527e", linewidth = 0.75) +
  theme_minimal() +
```

```r
  annotate("text",
           x = satt_est - 3,
           y = 0.06,
           label = paste("Median:", round(satt_est, 2)),
           color = "#2596be",
           hjust = 1) +
  annotate("text",
           x = int_satt_est - 4,
           y = 0.07,
           label = paste("Median:", round(int_satt_est, 2)),
           color = "#fd527e",
           hjust = 1) +
  labs(title = "Distribution of SATT Estimates",
       # subtitle = paste("Median BART SATT =", round(satt_est, 2)),
       x = "SATT (seconds)",
       y = "Density") +
  scale_fill_manual(name = "Model",
                    values = c("BART" = "lightblue",
                               "Multivariate Linear" = "pink")) +
  theme(
    plot.title = element_text(hjust = 0.5),
    plot.subtitle = element_text(hjust = 0.5)
  )
```

```r
# Calculate SATT by hour
n_groups <- 24
hr_codes <- seq(from = 1, to = 24)
hr_means <- rep(NA, n_groups)
hr_upper <- rep(NA, n_groups)
hr_lower <- rep(NA, n_groups)

for (hr in 1:n_groups) {
  indices <- df_combo$g_hr[df_combo$rapid_ride == 1]

  y1_pred_hr <- treated_factual_pred[, indices == hr]
  y0_pred_hr <- treated_counterfactual_pred[, indices == hr]

  if (ncol(y1_pred_hr) > 0) {

  ind_effects_hr <- y1_pred_hr - y0_pred_hr
  satt_dist_hr <- rowMeans(ind_effects_hr)

  satt_est_hr <- mean(satt_dist_hr)
  satt_ci_upper <- quantile(satt_dist_hr, probs = c(0.975))
```

```
  satt_ci_lower <- quantile(satt_dist_hr, probs = c(0.025))

  hr_means[hr] <- satt_est_hr
  hr_upper[hr] <- satt_ci_upper
  hr_lower[hr] <- satt_ci_lower
  }
  else {
    hr_means[hr] <- 0
    hr_upper[hr] <- 0
    hr_lower[hr] <- 0
  }
}

hr_ests <- tibble(hr_codes, hr_means, hr_upper, hr_lower)

# Multivariate for comparison
# Calculate SATT by day
n_groups <- 24
int_hr_codes <- seq(from = 1, to = 24)
int_hr_means <- rep(NA, n_groups)
int_hr_upper <- rep(NA, n_groups)
int_hr_lower <- rep(NA, n_groups)
n_draws <- 1000


for (hr in 1:n_groups) {
  hrtreated_df <- df_combo %>%
    filter(rapid_ride == 1) %>%
    filter(g_hr == hr)

  if (nrow(hrtreated_df > 0)) {
    hrcounter_df <- hrtreated_df
    hrcounter_df$rapid_ride <- 0

    y1_pred_hr <- posterior_epred(fit_int, newdata = hrtreated_df)
    y0_pred_hr <- posterior_epred(fit_int, newdata = hrcounter_df)
    ind_effects_hr <- y1_pred_hr - y0_pred_hr
    satt_dist_hr <- rowMeans(ind_effects_hr)

    satt_est_hr <- mean(satt_dist_hr)
    satt_ci_upper <- quantile(satt_dist_hr, probs = c(0.975))
    satt_ci_lower <- quantile(satt_dist_hr, probs = c(0.025))

    int_hr_means[hr] <- satt_est_hr
```

```
    int_hr_upper[hr] <- satt_ci_upper
    int_hr_lower[hr] <- satt_ci_lower
  }
  else {
    int_hr_means[hr] <- 0
    int_hr_upper[hr] <- 0
    int_hr_lower[hr] <- 0
  }
}

int_hr_ests <- tibble(int_hr_codes, int_hr_means, int_hr_upper, int_hr_lower)
```

```
hourly_comparison_graph <- ggplot(data=hr_ests) +
  # BART
  geom_point(aes(x=hr_codes, y=hr_means,
                  color = "BART")) +
  geom_errorbar(aes(ymin=hr_lower,
                  ymax=hr_upper,
                  x=hr_codes,
                  color = "BART"), alpha=1, width = 0) +
  # MULTIVARIATE
  geom_point(aes(x=int_hr_codes, y=int_hr_means,
                  color = "Multivariate")) +
  geom_errorbar(aes(ymin=int_hr_lower,
                  ymax=int_hr_upper,
                  x=int_hr_codes,
                  color = "Multivariate"), alpha=1, width = 0) +
  scale_color_manual(name="",
                  values=c("Multivariate"="#fd527e", "BART" = "#2596be")) +
  geom_hline(yintercept = 0, linetype='dashed', col = 'gray')+
  theme_minimal() +
  scale_x_continuous(breaks = seq(min(hr_ests$hr_codes),
                                   max(hr_ests$hr_codes), by = 1)) +
  scale_y_continuous(breaks = c(-150, -75, 0, 75, 150)) +
  labs(title="Treatment Effect by Hour of the Day",
       x="Hour of the Day",
       y="SATT (seconds)")+
  theme(axis.title=element_text(size=10),
        axis.text.y=element_text(size=10),
        axis.text.x=element_text(angle=90,size=8, vjust=0.3),
        legend.title=element_text(size=10),
        legend.text=element_text(size=10))
```