StemGNN for Divvy Bike Re-balancing

Noah Anderson

December 4th 2023

Abstract

This study applies the StemGNN model, originally developed for traffic forecasting, to the challenge of bike rebalancing in Chicago's Divvy bike-share program. Utilizing data from the Chicago Data Portal, the research focuses on forecasting dock availability at various stations. The model treats each station as a node within a network, analyzing spatial and temporal correlations. Optimal model configurations were determined through fine-tuning learning rates and window sizes. The most effective model, with a learning rate of 10^{-3} and a window size of 6, achieved RMSEs of 5.09% and 5.04% for 30 and 60-minute forecasts, respectively. When normalized by the standard deviation of the test data, the results were 21.4% and 21.2%, indicating the model's potential in urban bike-sharing systems for predictive modeling and urban mobility management.

Introduction

Divvy, the city-owned bike share program of Chicago, operated in partnership with Lyft, has seen substantial growth since its inception in 2013. However, a significant challenge that has emerged with this expansion is the imbalance in bike station capacities across the city. Some stations frequently experience shortages, while others have an excess of bikes. To manage this issue, Divvy has employed manual rebalancing strategies, involving paid workers who use vans to redistribute bikes from crowded stations to those with fewer bikes. Despite these efforts, the dynamic nature of station statuses, particularly during peak hours, underscores the critical need for advanced predictive modeling to enhance operational efficiency.

Previous attempts to tackle this challenge, employing regression techniques and decision trees, have factored in variables like weather conditions and historical station data. However, these methods have met with limited success in accurately predicting station imbalances (Krumlinde, 2019). This paper introduces a novel approach by exploring the StemGNN model, a machine learning algorithm originally designed for complex network systems and proven effective in traffic forecasting. In this study, each Divvy station is conceptualized as a node within a network, with a focus on the percentage of occupied docks. The application of StemGNN, which has demonstrated its efficacy in traffic pattern predictions (Cao et al., 2020), offers a unique perspective to address the challenges faced by Divvy's bike rebalancing efforts.

The research leverages historical data on dock occupancy to forecast bike availability at various stations. The methods section elaborates on how the StemGNN model was adapted to Divvy's dataset, which encompasses data from specific time periods and various neighborhoods in Chicago. The results are then presented, highlighting the model's performance and how different parameters influence its forecasting accuracy. Ultimately, this paper seeks to demonstrate the potential of StemGNN in urban bike-sharing systems, providing valuable insights for future enhancements in predictive modeling and urban mobility management.

Methods

In this section, the methodology employed in forecasting bicycle shortages at various Divvy Stations in Chicago using a modified version of the StemGNN model in python originally developed by Microsoft (StemGNN repository, Microsoft, 2023) will be described.

The Data

The dataset used for this study, sourced from the Chicago Data Portal (City of Chicago, 2023), covers the period from March to September 2022. This dataset provided comprehensive historical information regarding the status of Divvy stations, including the availability and total number of docks at each station, with data recorded at 10-minute intervals.

The data, queried from the Chicago Data Portal (2022), spans from 2022-07-07 09:15:27 CDT to 2022-09-10 18:25:43 CDT. For model tuning and exploratory analysis, focus was placed on specific neighborhoods, namely Uptown, North Center, and Lincoln Square (depicted in Figure 1), which collectively encompass 41 bike stations (as shown in Figure 2). The data was then reformatted in R to create a $T * N$ matrix, with T representing time stamps and N representing nodes. Each matrix entry reflected the percentage of docks in use, serving as the predictor. The final dataset comprised 41 nodes and 9288 rows. Detailed information on the data cleaning process is available in Appendix Section B.

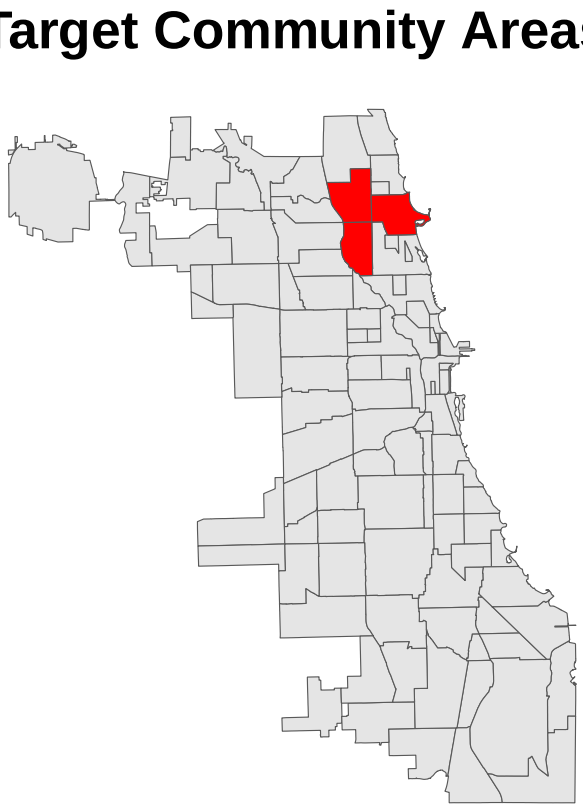

Figure 1: Figure 1: Map of target community areas (red indicates targets)

The Model

The StemGNN model, a groundbreaking approach introduced by Cao et al. (2020), innovatively combines Discrete Fourier Transform (DFT) and Graph Fourier Transform (GFT) to effectively analyze time-series data. The DFT component of StemGNN plays a pivotal role in modeling temporal dependencies, enabling the model to detect and interpret patterns like seasonality and autocorrelation, which are common in time-series data. This temporal analysis is crucial for understanding the underlying trends and cyclic behaviors in the dataset. In contrast, the GFT component focuses on interseries correlations, analyzing the spatial interactions between nodes. By employing GFT, the model is adept at uncovering the spatial relationships that exist in the data, providing insights into how different data points (or nodes) interact and influence each other in a given space.

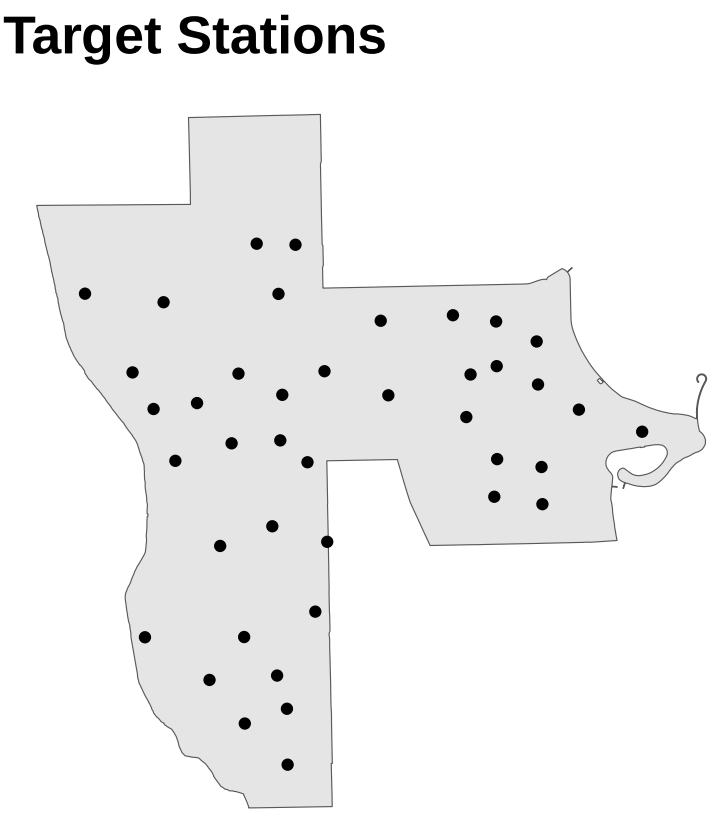

Figure 2: Figure 2: Map of target stations overlayed on target neighborhoods (black dots indicate target stations)

The model operates on data formatted in a $T * N$. At its core, StemGNN is composed of multiple neural network layers, each contributing to the deep learning capabilities of the model. The journey through the model begins with the Latent Correlation Layer, designed to initially explore and learn the spatial correlations present within the data. This layer sets the stage for more intricate spatial analyses which are further enhanced by subsequent layers.

Following this, the Graph Fourier Transform is applied, transforming the data into the frequency domain with a specific focus on spatial relationships. This transformation is a crucial step in distilling the spatial characteristics of the data, enabling the model to handle complex spatial structures effectively. The model then shifts its focus to temporal aspects, employing the Discrete Fourier Transform (DFT). By applying DFT, StemGNN is able to transform the time-series data into the frequency domain, a step that is fundamental in unveiling and understanding temporal dynamics and patterns that unfold over time.

The model further refines its analysis with a 1D Convolutional layer, which is adept at extracting key temporal features from the data. This layer methodically scans the transformed data, pinpointing significant temporal features and patterns that are essential for accurate forecasting. The Spectral Graph Convolution layer then takes over, applying Graph Fourier Transform (GFT) to capture the intricate spatial correlations between different nodes. This layer is integral to understanding how different series or nodes within the graph are interconnected and influence each other, revealing the complex web of spatial dependencies in the dataset.

An additional layer, the Gated Linear Unit (GLU), is incorporated into the model, serving as a sophisticated filter. The GLU layer critically evaluates and determines the relevance of the information processed by previous layers, ensuring that only pertinent features are carried forward for predictions. This layer adds an element of decision-making to the model's processing, enhancing its ability to focus on the most relevant aspects of the data.

The final step in the model's processing pipeline is the application of the Inverse Discrete Fourier Transform (IDFT). This transformation is pivotal as it converts the frequency-domain data back into the time domain, making the model's outputs interpretable and directly applicable to the original temporal structure of the data. This step is essential for rendering the model's predictions and analyses in a format that is meaningful and actionable in real-world contexts.

Originally, Cao et al. (2020) trained the StemGNN model on a diverse array of multivariate time-series datasets, including those related to traffic flow and COVID-19 transmission. This initial training involved fine-tuning the model to reduce prediction errors across various domains, ensuring its versatility and effectiveness. For the current project, this pre-trained model was adapted and fine-tuned specifically for the Divvy bike dataset. This adaptation process required reformatting the dataset to conform to the $ T * N $ structure thereby enabling effective training and accurate forecasting for this specific application.

Model Tuning

In this research, the focus was on fine-tuning two primary parameters of the StemGNN model: the learning rate and window size, while maintaining the default settings for other parameters. The learning rate influences the model's learning speed, and the window size specifies the quantity of past data points used for forecasting. Experiments were conducted with forecast horizons of 3 and 6 (equivalent to 30 and 60 minutes into the future), learning rates of 0.01, 0.001, and 0.0001, and window sizes of 6, 12, 18, were considered. All three learning rates were only run across window size 6 in the interest of time since the run time increases roughly in proportion to the increase in window size.

Model Metrics

Root Mean Square Error (RMSE) served as the primary evaluation metric to assess model performance. It is worth noting that RMSE can be calculated in various ways for a joint time series, as it effectively consists of different models, one for each node. In this study, RMSE for the entire output, encompassing both time and space, was computed as the chosen metric for evaluation. This approach provides a comprehensive

Table 1: Table 1: Model results by window size (Learning Rate .001)

Horizon	WindowSize	RMSE	NormalizedRMSE
3	6	5.087128	0.2137161
3	12	5.017701	0.2107154
3	18	5.049127	0.2119052
6	6	5.041065	0.2117744
6	12	5.114447	0.2147713
6	18	5.237997	0.2198242

Table 2: Table 2: Metrics by learning rate (Horizon 3, Window Size 6

LearningRate	RMSE	NormalizedRMSE
1e-02	7.820171	0.3285344
1e-03	5.087128	0.2137161
1e-04	14.911723	0.6264587

measure of model performance across both temporal and spatial dimensions. In order to assess the model performance in context with the variability of the data, RMSE normalized by the standard deviation of the test data was also included.

Results

This section presents the findings from the analysis focused on tuning the StemGNN model for forecasting bicycle dock availability at Divvy stations in Chicago. The objective was to ascertain the most effective model parameters, specifically window size and learning rate, for accurate predictions of dock availability. The impact of these parameters on the model's performance was thoroughly evaluated, leading to the identification of configurations that strike a balance between efficiency and precision. The insights gained here illuminate the complexities of model tuning and pave the way for future predictive modeling in urban mobility scenarios.

Model Tuning

The window size proved to be unimportant for model accuracy with a spread in normalized RMSE of only .002 for horizon 3 and .008 for horizon 6. As shown in Table 1, for horizon 3, a window size of 12 proved most accurate and for horizon 6, a window size of 6 performed best. The negligible improvement in performance for window size 12 for horizon 3 does not justify the fact that it doubled the run time. The key take away is that the smallest possible window size yields reasonable accuracy that either beat or closley rival models that consider more past data points.

Learning rate proved far more sensitive of a tuning parameter as evidenced by Figure 3 which plots RMSE by the exponents of learning rates on a base 10 scale. From this we can see 10^{-4} proves too slow and 10^{-2} proves too fast with a learning rate of 10^{-3} proving to yield optimal results.

Most Optimal Model

The most optimal models, balancing efficiency and accuracy, used a learning rate of 10^{-3} and a window size of 6 for both 30 minute and 60 minute forecasts. This resulted in a RMSE of 5.09% and 5.04% for each model respectively. These are very optimistic results since it implies that dock availability percentage

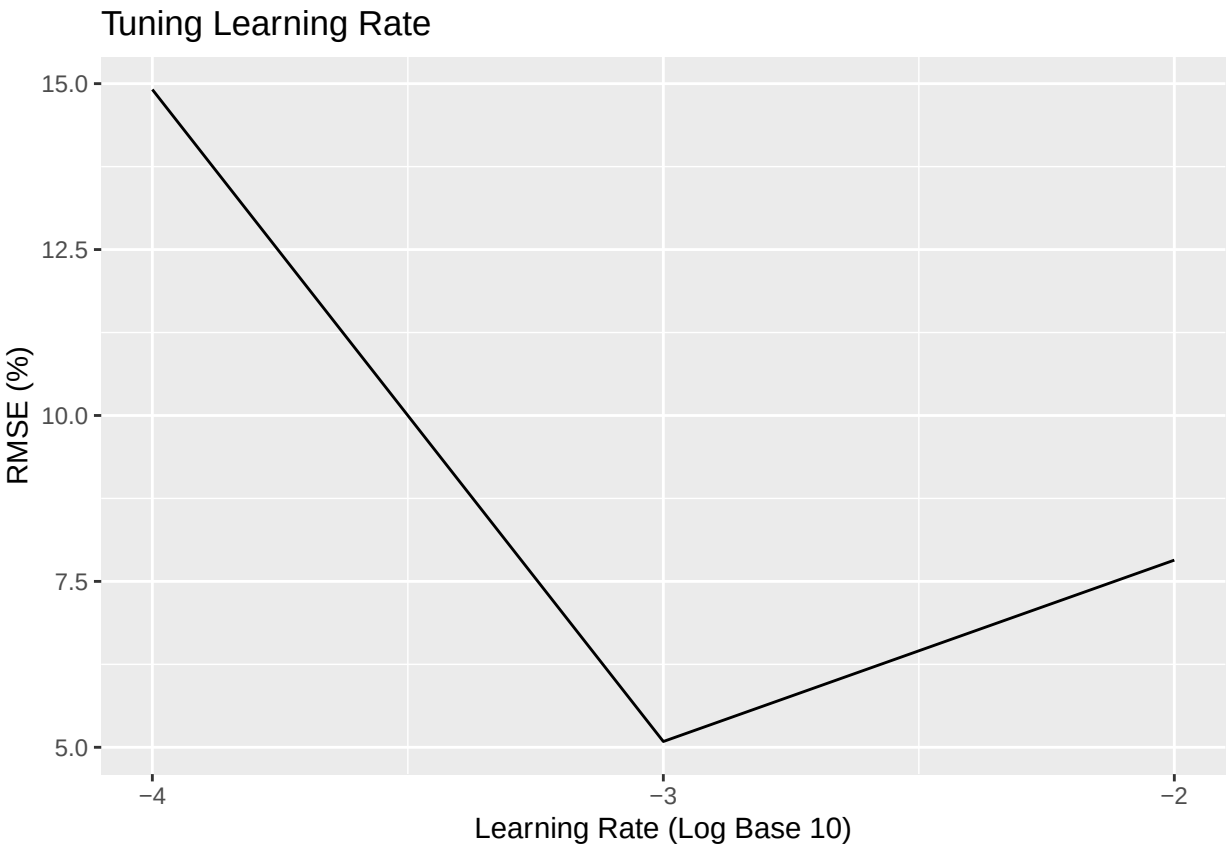

Figure 3: Figure 3: RMSE across different learning rates for the 30-minute forecast.

can be predicted with approximately 5% accuracy. This does not consider though the variability of the data, so RMSE was calculated as well normalizing by the standard deviation of the test data. These yield less optimistic results but still imply reasonable accuracy with results of 21.4% and 21.2% of the test datas standard deviation for horizons 3 and 6.

Discussion

The results found in this study are promising showcasing the power of StemGNN for geospatial joint time series. With the continuing growth of bike share programs more accurate rebalancing models will continue to be an important field of study cutting down on the potential cost and improving the reliability of available bikes for commuters. The results were not directly comparable to the project done by Krumlinde (2019) since different predictors were used (trips from in Krumlinde's case and percent full in this study). Another model that was in the development phase is also underway for Divvy that relies on a Poisson model. This is being developed by The Data Science For Social Good (DSSG) foundation which has proven its success with the Capital Hill Bike Share program in Washington D.C. (Henderson and Fishman 2013).

Limitiations

Both the Poisson model by DSSG and the model by Krumlinde either dealt with or plan to deal with Divvy data focusing around the Loop, the business district of Chicago. This study merely focused on four dense and populous residential neighborhoods each with their own unique restaurant and entertainment scenes. These neighborhoods were chosen due to largely not being accessible to one another by the CTA trains making Divvy bikes an optimal method of travel between neighborhoods. The idea was that these neighborhoods would be less complex than the Loop and serve as a good testing grounds for the viability of this model since they contain far less data points.

Future Steps

The next step for this model could include exploring some other tuning parameters such as Leaky Relu rate. Next adapting the code from Krumlinde and the DSSG for the percentage full data would help compare the StemGNN performance to other possible models.

The ultimate test will be to check its viability for the Loop which has the most Divvy activity and suffers from the worst balancing problems. It is the hope that this study will provide some working knowledge of optimal tuning parameters. There is no guarantee that the same parameters for the subsection examined in this project will be optimal for more complex networks, but it can at least serve as a starting point for model tuning.

Appendix

Code listed here can also be found in its full context on the authors GitHub here https://github.com/noahba65/cap-stone-PDAT. The code for the model can be found in this subdirectory located here: https://github.com/noahba65/cap-stone-PDAT/tree/main/src/stem_gnn_model.

A: Packages

```r
library(RSocrata) # For reading in data from the City of Chicago
library(tidyverse) # For data cleaning and visualization
library(sf) # For geospatial data manipulation
library(stringr) # For scrubbing directories for tuning parameters
library(kableExtra) # For visualizing results
```

B: Data Setup

1: Data Acquisition

This code reads in three data sets, Boundaries - Community Areas (2018), Divvy Bicycle Stations (2022), and Divvy Bicycle Stations - Historical (2022). Boundaries - Community Areas is neccesary for geospatial data filtering in combination with Divvy Bicycle stations. Divvy Bicyle Stations, read in with R Socrata is simply a list of Divvy stations with location data which works with the Community Areas dataset. Divvy Bicycle Stations - Historical is the crucial time series data set outlined in the Methods section.

```r
# Read in a local geojson file containing the shapes of Chicago Neighborhoods
chicago_ca_raw <- st_read("/Users/noahanderson/Documents/Data/Boundaries-Neighborhoods.geojson")

# Read in Divvy names and locations data set
divvy_stations_url <- "https://data.cityofchicago.org/resource/bk89-9dk7.json"
divvy_stations_raw <- read.socrata(divvy_stations_url)

# Read in historical Divvy data
divvy_raw <- read_csv("/Users/noahanderson/Documents/Data/Divvy_Bicycle_Stations_-_Historical.csv")
```

2: Data Cleaning

The Divvy Stations and Community Areas datasets are used to create `id_filter` that is used to filter the time series data only spanning the target neighborhoods.

```r
# Define vector for community areas to be queried
target_ca <- c( "Uptown", "North Center", "Lincoln Square")

# Set coordinate reference system for community areas
chicago_ca_clean <- chicago_ca_raw %>%
  st_transform(4269)

# Convert stations dataset to sf object and set coordinate reference system
divvy_stations_clean <- divvy_stations_raw %>%
  st_as_sf(coords = c("longitude", "latitude"), crs = 4269)

# Join data frames to identity stations in target community areas
divvy_stations_in_ca <- st_join(chicago_ca_clean %>% filter(pri_neigh %in% target_ca), divvy_stations_cl

# Create filter for station ID's in the target Community Areas
id_filter <- divvy_stations_in_ca$id

# Clean divvy data
divvy_10_min_clean_percent <- divvy_raw %>%
  rename( percent_full = `Percent Full` ) %>%
```

```
  filter( ID %in% id_filter) %>%
  select(Timestamp, ID, percent_full) %>%
  mutate(Timestamp = as.POSIXct(Timestamp, format = "%m/%d/%Y %I:%M:%S %p"))


# Pivot data to wide T*N format
divvy_10_min_wide_percent <- divvy_10_min_clean_percent %>%
  pivot_wider(names_from = ID, values_from = percent_full) %>%
  arrange(Timestamp) %>%
  select(-Timestamp)

# Remove column names
colnames(divvy_10_min_wide_percent) <- NULL

# Save csv to StemGNN sub directory
save_csv(divvy_10_min_wide_percent, "/Users/noahanderson/Documents/GitHub/cap-stone-PDAT/src/stem_gnn_m
```

C: Evaluation

1: Custom Functions

This code builds custom functions for reading in model output data specific to the structure of the project repository along with functions for computing model metrics.

```
read_error_and_target_files <- function(base_dir) {
  # List all subdirectories within the base directory
  subdirs <- list.dirs(base_dir, full.names = TRUE, recursive = FALSE)

  # Initialize a list to store the data frames
  data_list <- list()

  # Loop through each subdirectory
  for (dir in subdirs) {
    # Construct file paths
    error_file_path <- file.path(dir, "predict_abs_error.csv")
    target_file_path <- file.path(dir, "target.csv")

    # Check if the files exist and read them
    if (file.exists(error_file_path) && file.exists(target_file_path)) {
      error_df <- read_csv(error_file_path, col_names = FALSE)
      target_df <- read_csv(target_file_path, col_names = FALSE)

      # Add the data frames to the list
      data_list[[basename(dir)]] <- list(Error = error_df, Target = target_df)
    }
  }

  return(data_list)
}

# Function to calculate RMSE and normalized RMSE by standard deviation
calculate_rmse <- function(data_list, dir_name) {
```

```
  error_df <- data_list$Error
  target_df <- data_list$Target

  # Calculate RMSE
  rmse <- sqrt(mean(as.matrix(error_df^2)))

  # Calculate the standard deviation of the target values
  std_dev <- sd(as.vector(as.matrix(target_df)))

  # Normalize RMSE by the standard deviation
  normalized_rmse <- rmse / std_dev

  return(data.frame(Directory = dir_name, RMSE = rmse, NormalizedRMSE = normalized_rmse))
}

# Function to extract window size and horizon from directory name
extract_info <- function(dir_name) {
  window_size <- as.numeric(str_extract(dir_name, "(?<=window_size_)[0-9]+"))
  horizon <- as.numeric(str_extract(dir_name, "(?<=horizon_)[0-9]+"))
  learning_rate <- as.numeric(str_extract(dir_name, "(?<=lr_)[0-9\\.e-]+"))
  return(c(WindowSize = window_size, Horizon = horizon, LearningRate = learning_rate))
}
```

2: Evaluating Metrics

This code reads in and computes model metrics that are presented in the study.

```
base_dir <- "/Users/noahanderson/Documents/GitHub/cap-stone-PDAT/src/stem_gnn_model/output/divvy_10_min_

# Reads in absolute error datasets
abs_error_and_target_data <- read_error_and_target_files(base_dir)

# Apply the RMSE calculation to each dataframe and combine the results into a dataframe
rmse_results_df <- map_df(names(abs_error_and_target_data), ~calculate_rmse(abs_error_and_target_data[[


# Add columns for window size and learning rate
rmse_results_df_clean <- rmse_results_df %>%
  mutate(WindowSize = map_dbl(Directory, ~extract_info(.x)["WindowSize"]),
         Horizon = map_dbl(Directory, ~extract_info(.x)["Horizon"]),
         LearningRate = map_dbl(Directory, ~extract_info(.x)["LearningRate"])) %>%
  select(-Directory)

# Select and arrange data for final format
metrics_df <- rmse_results_df_clean %>%
  select(Horizon, LearningRate, WindowSize, RMSE, NormalizedRMSE) %>%
  arrange(Horizon, WindowSize)

# Save results
write_csv(metrics_df, "/Users/noahanderson/Documents/GitHub/cap-stone-PDAT/src/eval/data/percent_metrics
```

D Visualizations

This code reads in metrics data created in the previous section and builds the vizualistions used in this paper.

```
# Data cleaning for Figures 1 and 2
divvy_stations_in_ca <- st_join(target_area, divvy_stations_clean)
id_filter <- divvy_stations_in_ca$id

# Clean divvy data to give only ID's in target area
divvy_data_ID <- divvy_raw %>%
  filter( ID %in% id_filter) %>%
  select(ID)

target_stations <- divvy_stations_clean %>%
  filter(id %in% divvy_data_ID$ID)

# Generate figure 1
ggplot() +
  geom_sf(data = chicago_ca_clean) +
  geom_sf(data = target_area, fill = "red") +
  ggtitle("Target Community Areas") +
  theme(
    panel.grid.major = element_blank(),   # Remove major grid lines
    panel.grid.minor = element_blank(),   # Remove minor grid lines
    axis.text.x = element_blank(),        # Remove x-axis labels
    axis.text.y = element_blank(),        # Remove y-axis labels
    axis.ticks = element_blank(),         # Remove axis ticks
    panel.background = element_rect(fill = "white"),  # Set background to white
    plot.title = element_text(face = "bold", size = 20) # Bold and larger title
  )

# Generate Figure 2
ggplot() +
  geom_sf(data = st_union(target_area )) +
  geom_sf(data = target_stations) +
  ggtitle("Target Stations") +
  theme(
    panel.grid.major = element_blank(),   # Remove major grid lines
    panel.grid.minor = element_blank(),   # Remove minor grid lines
    axis.text.x = element_blank(),        # Remove x-axis labels
    axis.text.y = element_blank(),        # Remove y-axis labels
    axis.ticks = element_blank(),         # Remove axis ticks
    panel.background = element_rect(fill = "white"),  # Set background to white
    plot.title = element_text(face = "bold", size = 20) # Bold and larger title
  )


# Read in metrics data
percent_metrics_df <- read_csv("/Users/noahanderson/Documents/GitHub/cap-stone-PDAT/src/eval/data/percen

# Create table 1
kable( percent_metrics_df %>% filter(LearningRate == .001) %>% select(-LearningRate), caption = "Table
```

```r
# Create table 2
kable( percent_metrics_df %>% filter(WindowSize == 6, Horizon == 3) %>% select(-Horizon, -WindowSize), 

# Create figure 3
percent_metrics_df %>%
  filter(Horizon == 3, WindowSize == 6) %>%
  ggplot() +
  geom_line(aes(x = log10(LearningRate), y = RMSE)) +
  scale_x_continuous(breaks = c(-4, -3, -2)) +
  ggtitle("Tuning Learning Rate") +
  xlab("Learning Rate (Log Base 10)") +
  labs(caption = "Figure 1: RMSE across different learning rates for the 30-minute forecast.")
```

References

Cao, D., Wang, Y., Duan, J., Zhang, C., Zhu, X., Huang, C., Tong, Y., Xu, B., Bai, J., Tong, J., & Zhang, Q. (2020). Spectral temporal graph neural network for multivariate time-series forecasting. Advances in neural information processing systems, 33, 17766-17778. https://doi.org/10.48550/arXiv.2103.07719

City of Chicago. (2018, December 18). Boundaries - Community Areas (current). Retrieved [2023 October 30], from https://data.cityofchicago.org/Facilities-Geographic-Boundaries/Boundaries-Community-Areas-current-/cauq-8yn6

City of Chicago. (2022, September 10). Divvy Bicycle Stations. Retrieved [2022 September 10], from https://data.cityofchicago.org/Transportation/Divvy-Bicycle-Stations/bbyy-e7gq

City of Chicago. (2022, September 10). Divvy Bicycle Stations - Historical. Retrieved [2023 November 26], from https://data.cityofchicago.org/Transportation/Divvy-Bicycle-Stations-Historical/eq45-8inv

City of Chicago. "Divvy Trips." Data published by City of Chicago. Accessed on November 27, 2023. Available at: https://data.cityofchicago.org/Transportation/Divvy-Trips/fg6s-gzvg.

Devlin, H., Schenk, T. Jr., Leynes, G., Lucius, N., Malc, J., Silverberg, M., & Schmeideskamp, P. (2023). RSocrata: Tools for downloading and using data from 'Socrata' data portals. R package version [1.7.15-1]. Retrieved from https://github.com/Chicago/RSocrata

Henderson, J., & Fishman, A. (2013, August 9). Divvy: Helping Chicago's New Bike Share Find Its Balance. Data Science For Social Good. https://www.dssgfellowship.org/2013/08/09/divvy-helping-chicagos-new-bike-share-find-its-balance/

Krumlinde, Z. (2019, July 22). Using Machine Learning to Predict Hourly Divvy Bike-Sharing Checkouts per Station. Towards Data Science. https://towardsdatascience.com/predicting-hourly-divvy-bike-sharing-checkouts-per-station-65b1d217d8a4

Microsoft. (2023). StemGNN: Spectral Temporal Graph Neural Network. GitHub. Retrieved from https://github.com/microsoft/StemGNN

Pebesma, E., & Bivand, R. (2023). Spatial Data Science: With applications in R. Chapman and Hall/CRC. doi:10.1201/9780429459016. Retrieved from https://r-spatial.org/book/

Wickham, H. (2023). stringr: Simple, Consistent Wrappers for Common String Operations. R package version 1.5.1. Retrieved from https://github.com/tidyverse/stringr

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L.D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T.L., Miller, E., Bache, S.M., Müller, K., Ooms, J., Robinson, D., Seidel, D.P., Spinu, V., Takahashi, K., Vaughan, D., Wilke, C., Woo, K., & Yutani, H. (2019). Welcome to the tidyverse. Journal of Open Source Software, 4(43), 1686. doi:10.21105/joss.01686. Retrieved from https://doi.org/10.21105/joss.01686

Zhu, H., Travison, T., Tsai, T., Beasley, W., Xie, Y., Yu, G., Laurent, S., Shepherd, R., Sidi, Y., Salzer, B., Gui, G., Fan, Y., Murdoch, D., & Evans, B. (2021). kableExtra: Construct Complex Table with 'kable' and Pipe Syntax. R package version [1.3.4]. Retrieved from https://github.com/haozhu233/kableExtra