Deep Dive 7

Jersey Citi Bike

Predicting Bike Demand through Rideshare Network

Our Team

Nikhil Arora
na32@illinois.edu
www.linkedin.com/in/nikhil-arora-uiuc

Gaurav Bhandari
gauravb4@illinois.edu
www.linkedin.com/in/gaurav-bhandari-52417411b

Srushti Manjunath
srushti5@illinois.edu
https://www.linkedin.com/in/srushti-manjunath/

Sarath Saroj
ssaroj2@illinois.edu
https://www.linkedin.com/in/sarathsaroj/

Agenda

1) Problem Statement
2) Initial Setup
3) Data Exploratory Analysis
4) Model Implementation
5) Results/Conclusion
6) Technical Challenges
7) Learning Experience
8) Future Considerations

Timeline

1 — 2 — 3 — 4

Milestone 1

Setting up the Project

Milestone 2

Data Exploration

Milestone 3

Baseline Learning Model

Milestone 4

Spatial Temporal Model

Problem Statement

Problem

- The problem was to use the CitiBike Data to predict number of rides for all stations in the network.

- With this prediction model we can optimize the placement of bikes throughout the network

- The model uses **Jersey Citibike** data to predicted using historical data the **net rides from each station.**

- We utilized the **SpatioTemporal Graph Convolutional Network(STGCN) model**

Exploratory Data Analysis

Data

- Data used for analysis is from https://ride.citibikenyc.com/system-data
- Rides data from 01-01-2017 to 10-01-2023 in Jersey City
- This data range was chosen as it would help us study patterns in the data over a number of years and hence we can make predictions with greater confidence.

Merged Dataframe

	ride_id	started_at	ended_at	start_station_name	start_station_id	end_station_name	end_station_id	start_lat	start_lng	end_lat	end_lng	member_casual
0	121DD7DD23CB1335	2021-02-03 23:11:28	2021-02-03 23:18:28	Hoboken Ave at Monmouth St	JC105	Christ Hospital	JC034	40.735208	-74.046964	40.734786	-74.050444	member
1	FD73FB85F008349D	2021-02-27 16:34:05	2021-02-27 16:56:40	Newport Pkwy	JC008	Marin Light Rail	JC013	40.728744	-74.032108	40.714584	-74.042817	member
2	39F9E6663CB5FDF6	2021-02-26 23:16:04	2021-02-26 23:22:25	Journal Square	JC103	Baldwin at Montgomery	JC020	40.733670	-74.062500	40.723659	-74.064194	member
3	A64745CB0792EC6F	2021-02-24 16:51:50	2021-02-24 17:16:09	Hoboken Ave at Monmouth St	JC105	Hoboken Ave at Monmouth St	JC105	40.735208	-74.046963	40.735208	-74.046964	casual
4	75CC76EB9543764A	2021-02-24 20:44:16	2021-02-24 20:44:46	Hoboken Ave at Monmouth St	JC105	Hoboken Ave at Monmouth St	JC105	40.735208	-74.046963	40.735208	-74.046964	member

- Missing start_station_ids: 77 removed
- Missing end_station_ids: 9168 removed
- Final Dataframe shape: (3678270,11)
- We did our exploratory data analysis on around 3 million rides in the Jersey City area

Distance Outliers Removed

	started_at	ended_at	start_station_name	start_station_id	end_station_name	end_station_id	start_lat	start_lng	end_lat	end_lng	member_casual	distance
112790	2017-09-01 00:15:32	2017-09-01 00:22:26	Grove St PATH	3186	WS Don't Use	3480	40.719586	-74.043117	0.0	0.0	casual	8670.281780
112899	2017-09-01 07:47:02	2017-09-01 07:54:08	Leonard Gordon Park	3281	JSQ Don't Use	3215	40.745910	-74.057271	0.0	0.0	member	8671.990798
113019	2017-09-01 08:44:31	2017-09-01 08:56:06	Oakland Ave	3207	WS Don't Use	3480	40.737604	-74.052478	0.0	0.0	member	8671.424519
113080	2017-09-01 09:17:42	2017-09-01 09:30:54	Christ Hospital	3212	WS Don't Use	3480	40.734786	-74.050444	0.0	0.0	member	8671.198551
113143	2017-09-01 10:05:21	2017-09-01 10:12:48	Dixon Mills	3279	WS Don't Use	3480	40.721630	-74.049968	0.0	0.0	member	8670.890976

- We use haversine distance to calculate the distance between the start station and end station of each ride specified by their latitude and longitude coordinates.
- We drop ride outliers that are 3 standard deviations away from the mean distance of all rides (787 rides).
- After finding the durations of the rides, we also drop another 97 rows which have negative durations.

Network Visualization

- We created an interactive map by overlaying the latitudes and longitudes of each station.

- Hover over the station to check station names

Centrality Analysis

Nodes with highest degrees:

	Station_Name	Degree
0	Harborside	109
1	14 St Ferry - 14 St & Shipyard Ln	98
2	Grand St	84
3	12 St & Sinatra Dr N	84
4	Washington St	83

- Degree centrality of a station shows the number of neighbors a station has.
- The above stations with high degree centrality can be considered as hubs or highly connected stations in the network.

Top 5 nodes by closeness centrality:

	Station_Name	Closeness
0	Harborside	0.607967
1	14 St Ferry - 14 St & Shipyard Ln	0.582297
2	Grand St	0.568403
3	12 St & Sinatra Dr N	0.565947
4	Washington St	0.564728

- Closeness centrality of a station shows how close a station is to all the other stations in the network.
- The above stations with high closeness centrality act as a central hub that allows for quick travel to other locations.

Top 5 nodes by betweenness centrality:

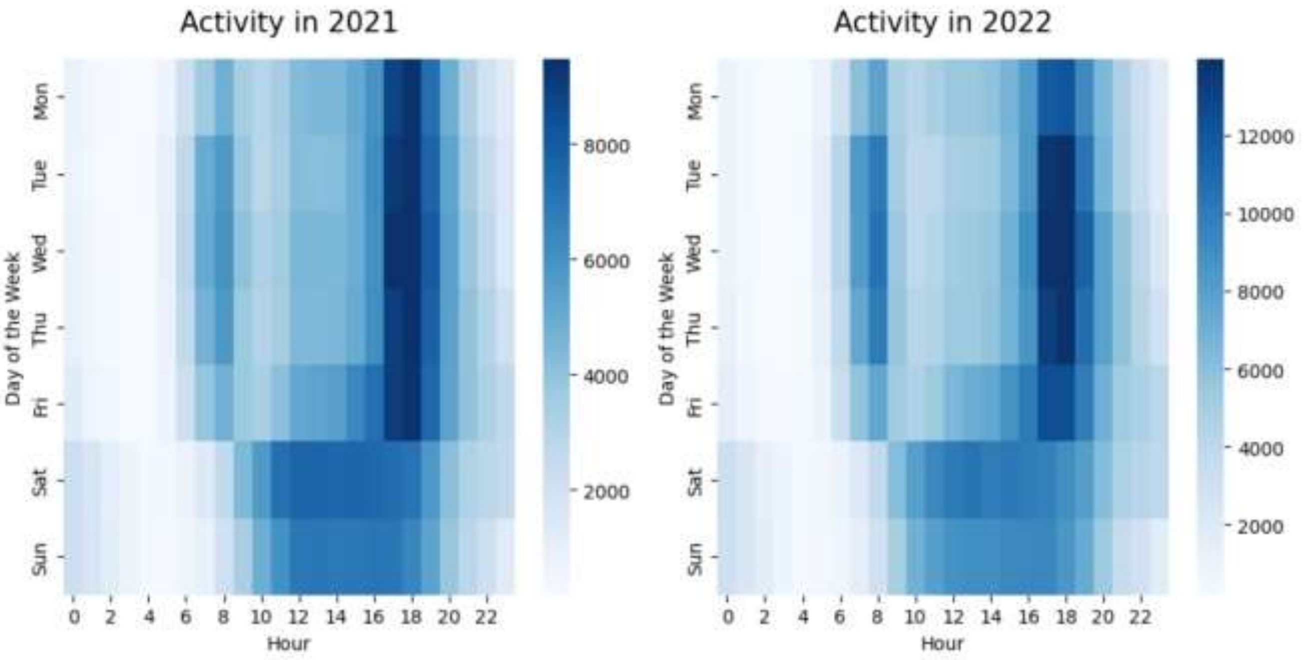
	Station_Name	Betweenness
0	Harborside	0.188173
1	14 St Ferry - 14 St & Shipyard Ln	0.124864
2	Grand St	0.088152
3	Bergen Ave	0.085875
4	Washington St	0.079340

- Betweenness centrality shows the extent to which a station acts as a bridge between other stations in the network.
- The above nodes with high betweenness centrality can easily connect the stations that are not directly connected.

Heatmap: Activity by the Hour: Weekday vs Weekend

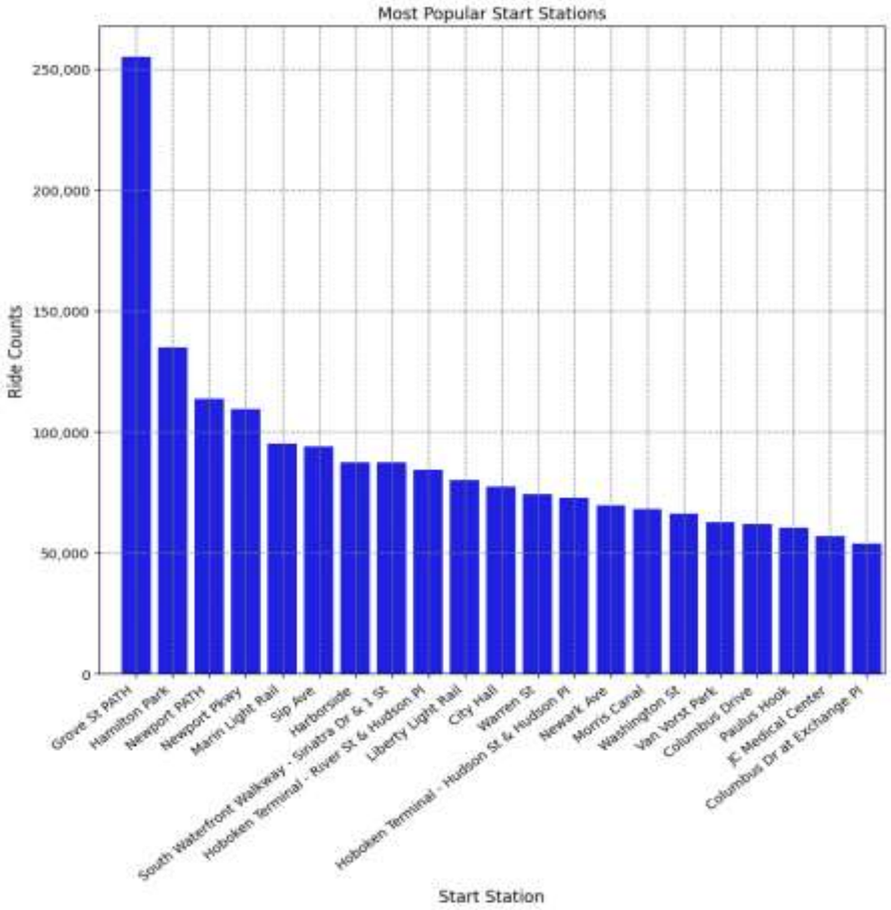
- We see that the total number of rides on weekdays is much higher prior and after office hours.

- On weekends, activity increases midday with a higher concentration of rides between 12 and 4 pm.



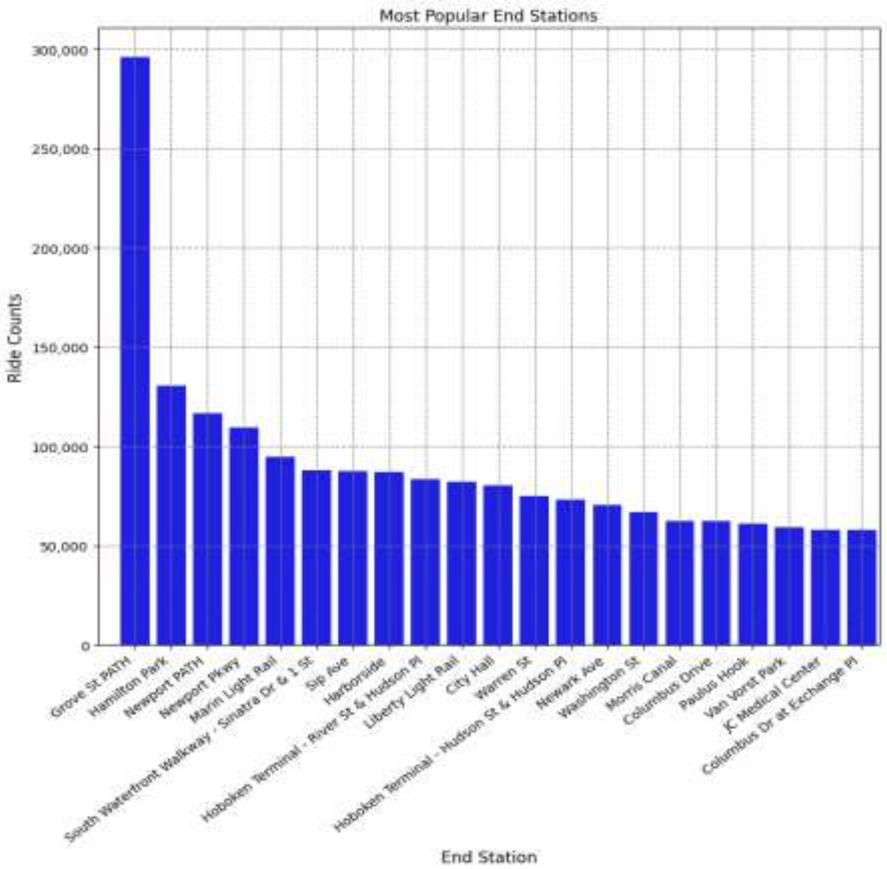
Top 20 Start Stations

- The number of rides that are originating from the top 20 most popular start stations give us a better idea of which stations to place more bikes at the start of each day, as this will follow the same trend if we look at it on a daily basis.



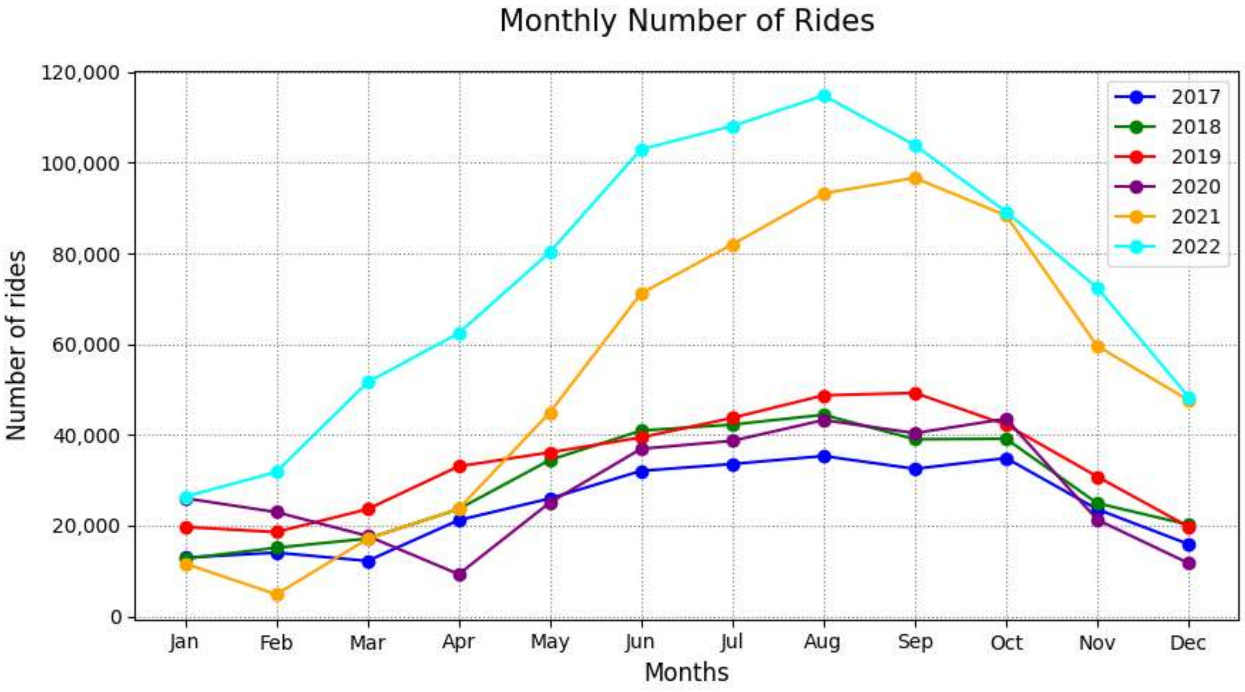
Top 20 End Stations

- If an end station is very popular, but we find that the same station does not have many rides originating from there, then we can direct rebalancing vans to move bikes from that end station to more popular start stations so supply meets demand at the popular start stations.
- However we see that both the top 20 start and end stations are very similar in ride counts, so these can be considered as hubs in the network.



Seasonality

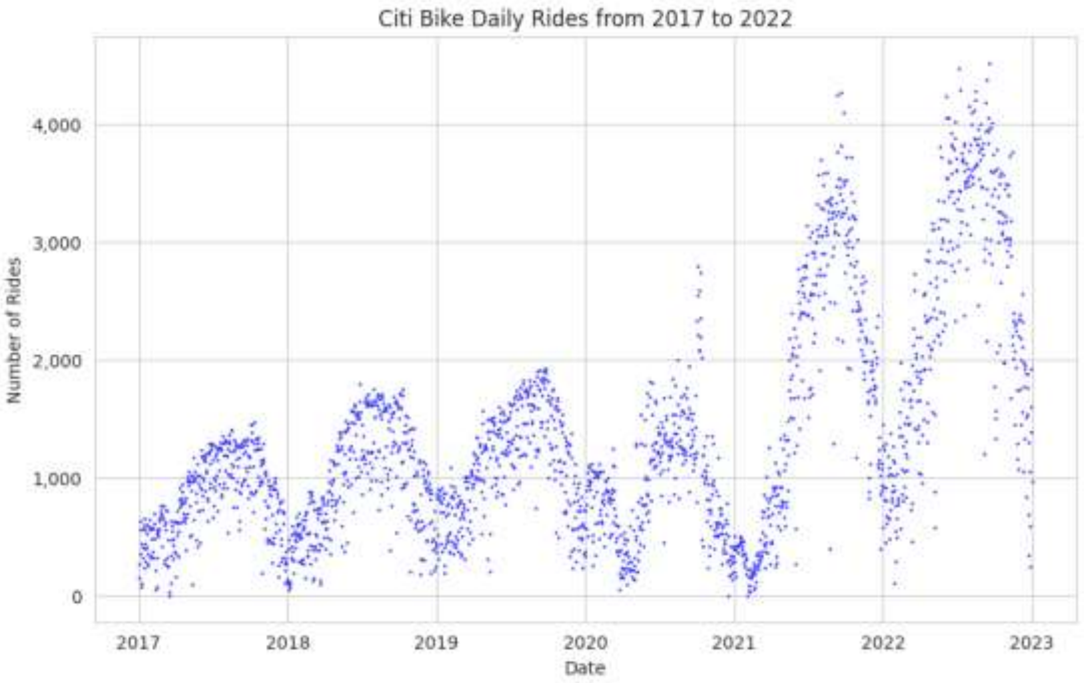
- We see a seasonality trend in the data with a much higher number of rides in the summer months when compared to the winter months.
- We decided to include this as a feature in our baseline model to help our predictions.



Yearly Trends

Cumulative Rides over Years

- There is also a trend in the overall year on year growth.
- There is clearly a rising demand over the years(evident in the last two years- post COVID-19) so we can infer that this is a growing industry.
- It has a cyclical demand pattern. There is seasonality in the data, with peaks in the summer months and valleys in the winter months.



Baseline Model

Feature Selection for Baseline Model

year	month	day_of_week	hour_of_day	1 Ave & E 16 St	1 Ave & E 30 St	1 Ave & E 5 St	1 Ave & E 6 St	1 Ave & E 62 St	1 Ave & E 68 St	...	Whitehall St & Bridge St	William St & Pine St	Willoughby Ave & Hall St	Willoughby Ave & Tompkins Ave	Willow Ave & 12 St	Wilson Ave & Moffat St	Withers St & Kingsland Ave	Wythe Ave & Metropolitan Ave	York St	York St & Marin Blvd
2022	1	5	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2022	1	5	10	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2022	1	5	11	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2022	1	5	13	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2022	1	5	15	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2022	12	4	12	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
2022	12	4	16	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
2022	12	4	19	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
2022	12	5	3	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
2022	12	5	18	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1

- From our exploratory data analysis, we found that there are other factors that change the demand of bikes at stations such as:
  - Year
  - Month of the year
  - Day of the week
  - Hour of the day

- Hot encoded the stations as features in our model.
- Target label - incoming and outgoing rides for each station within every one-hour time frame.

Model - Simple Feed Forward

- 2 hidden layers
- ReLU Activation
- Final Linear Layer

```
[ ] class SimpleNN(nn.Module):
        def __init__(self):
            super(SimpleNN, self).__init__()
            self.fc1 = nn.Linear(X_train.shape[1], 64)
            self.relu = nn.ReLU()
            self.fc2 = nn.Linear(64, 32)
            self.fc3 = nn.Linear(32, 1)

        def forward(self, x):
            x = self.relu(self.fc1(x))
            x = self.relu(self.fc2(x))
            x = self.fc3(x)
            return x
```

Hyperparameter

- Batch size=32
- Adam optimiser
- 10 Epochs

```
train_dataset = TensorDataset(X_train_tensor, y_train_out_tensor)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
```

```
model = SimpleNN()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
import torch
import matplotlib.pyplot as plt


train_losses_out = []
val_losses_out = []


epochs = 10
for epoch in range(epochs):
    model.train()
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        train_losses_out.append(loss.item())


    model.eval()
    with torch.no_grad():
        val_outputs = model(X_test_tensor)
        val_loss = criterion(val_outputs, y_test_in_tensor)
        val_losses_out.append(val_loss.item())

y_pred_out = val_outputs.detach().numpy()
```

Evaluation of Baseline Model

Calculation of Loss and Metric of the Baseline Model

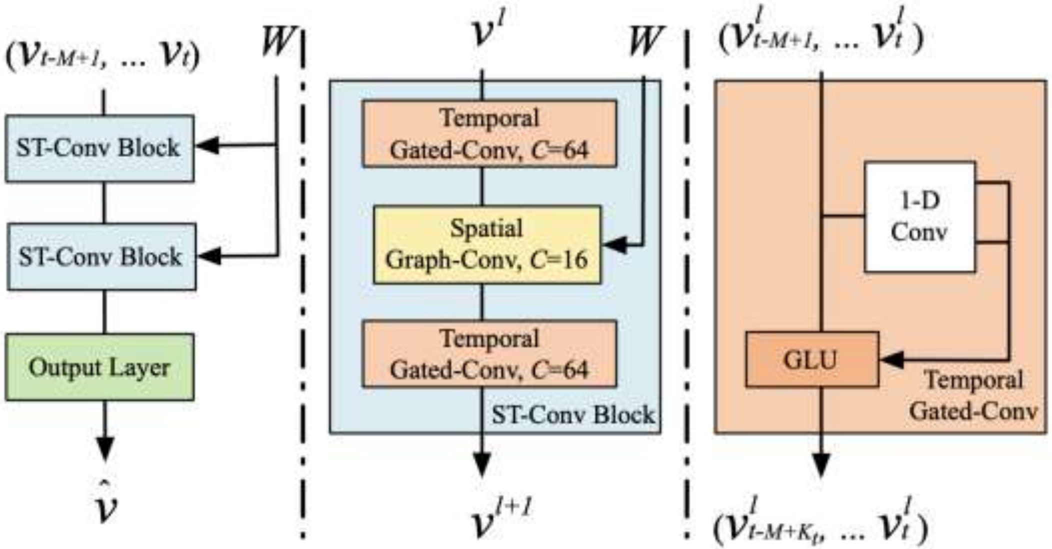
```
mse_in = mean_squared_error(y_test_in_tensor.numpy(), y_pred_in, squared=False)
mae_in = mean_absolute_error(y_test_in_tensor.numpy(), y_pred_in)
mse_out = mean_squared_error(y_test_out_tensor.numpy(), y_pred_out, squared=False)
mae_out = mean_absolute_error(y_test_out_tensor.numpy(), y_pred_out)
print(f"RMSE (Incoming Rides): {mse_in:.2f} rides")
print(f"MAE (Incoming Rides): {mae_in:.2f} rides\n")
print(f"RMSE (Outgoing Rides): {mse_out:.2f} rides")
print(f"MAE (Outgoing Rides): {mae_out:.2f} rides")
```

```
RMSE (Incoming Rides): 2.27 rides
MAE (Incoming Rides): 1.46 rides

RMSE (Outgoing Rides): 2.25 rides
MAE (Outgoing Rides): 1.50 rides
```

Final Model: STGCN

STGCN Model



$(v_{t-M+1}, \ldots v_t)$
W
ST-Conv Block
ST-Conv Block
Output Layer
$\hat{v}$
v^l
W
Temporal Gated-Conv, C=64
Spatial Graph-Conv, C=16
Temporal Gated-Conv, C=64
ST-Conv Block
v^{l+1}
$(v^l_{t-M+1}, \ldots v^l_t)$
1-D Conv
GLU
Temporal Gated-Conv
$(v^l_{t-M+K_t}, \ldots v^l_t)$

Features

- Number of unique stations: 498
- Number of net rides for every 30 minute interval
- Number of 30 minute intervals interval for 2023 data = 13104
- V: Node features - Net rides for each node in the graph at each point in time
- W: Distance for the edges between each pair of nodes(stations)

Model Parameters

```
# model parameters
channels = np.array([[1, 4, 8], [8, 4, 8]]) # sequence of channel sizes

kernel_size = 3 # size of temporal kernel
K = 3 # chebyshev filter size

learning_rate = 0.01
batch_size = 10
num_epochs = 20
num_layers = 1 # number of STConv blocks
n_his = 20 # number of historical time steps to consider
n_pred = 5 # steps in the future we want to predict

train_prop = 0.7
val_prop = 0.2
test_prop = 0.1
```

STGCN Model

```python
class FullyConnLayer(nn.Module):
    def __init__(self, c):
        super(FullyConnLayer, self).__init__()
        self.conv = nn.Conv2d(c, 1, 1)

    def forward(self, x):
        return self.conv(x)

class OutputLayer(nn.Module):
    def __init__(self, c, T, n):
        super(OutputLayer, self).__init__()
        self.tconv1 = nn.Conv2d(c, c, (T, 1), 1, dilation = 1, padding = (0,0))
        self.ln = nn.LayerNorm([n, c])
        self.tconv2 = nn.Conv2d(c, c, (1, 1), 1, dilation = 1, padding = (0,0))
        self.fc = FullyConnLayer(c)

    def forward(self, x):
        x_t1 = self.tconv1(x)
        x_ln = self.ln(x_t1.permute(0, 2, 3, 1)).permute(0, 3, 1, 2)
        x_t2 = self.tconv2(x_ln)

        return self.fc(x_t2)
```

STGCN Model

```python
# final model
# a specified number of STConv blocks, followed by an output layer
class TrafficModel(torch.nn.Module):
    def __init__(self, device, num_nodes, channel_size_list, num_layers,
                 kernel_size, K, window_size, \
                 normalization = 'sym', bias = True):
    # num_nodes = number of nodes in the input graph
    # channel_size_list =  2d array representing feature dimensions throughout the model
    # num_layers = number of STConv blocks
    # kernel_size = length of the temporal kernel
    # K = size of the chebyshev filter for the spatial convolution
    # window_size = number of historical time steps to consider
        super(TrafficModel, self).__init__()
        self.layers = nn.ModuleList([])
        for l in range(num_layers):
            input_size, hidden_size, output_size = \
            channel_size_list[l][0], channel_size_list[l][1], \
            channel_size_list[l][2]
            self.layers.append(STConv(num_nodes, input_size, hidden_size, \
                                      output_size, kernel_size, K, \
                                      normalization, bias))
        # add output layer
        self.layers.append(OutputLayer(channel_size_list[-1][-1], \
                                       window_size - 2 * num_layers * (kernel_size - 1), \
                                       num_nodes))
        for layer in self.layers:
            layer = layer.to(device)

    def forward(self, x, edge_index, edge_weight):
        for layer in self.layers[:-1]:
          x = layer(x, edge_index, edge_weight)
        out_layer = self.layers[-1]
        x = x.permute(0, 3, 1, 2)
        x = out_layer(x)
        return x
```

STGCN Model - Testing

```python
MAE, RMSE = evaluate_metric(best_model, test_iter, scaler, edge_index, edge_weight, device)
l = round(l, 2)
MAE = round(MAE, 2)
RMSE = round(RMSE, 2)
```

Python

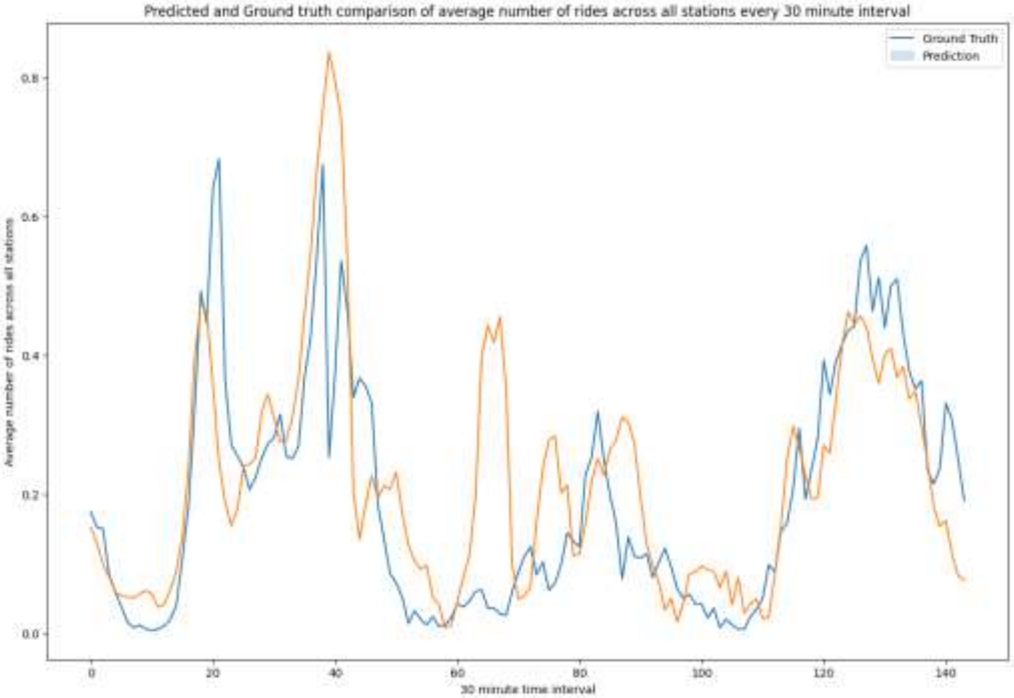
```python
print("MSE: {:.2f} rides \nMAE: {:.2f} rides \nRMSE: {:.2f} rides".format(l, MAE, RMSE))
```

✓ 0.0s

Python

```
MSE: 1.01 rides
MAE: 0.20 rides
RMSE: 0.77 rides
```

STGCN Model - Prediction



Results/Conclusions

Conclusions

- On analyzing the station ride data we find that there is maximum activity in stations such as **Grove St PATH station**, **Hamilton Park** and **Newport PATH**.

- On analysis we were able to identify that the rides data is **very sparse** in the Jersey City impling that most of the rides are on limited ride paths through the city. The demand is generated by only by few stations.

- The stations which are **high** in **demand** such as **Grove St Path and Hamilton Park have** have high degree of imbalance in terms of net rides.

- So a possible strategy to rebalance these is by sending vans to shift bikes from other smaller stations(in the vicinity) to these imbalanced stations at the end of business hours/maintenance hours

Technical Challenges

Some Considerations

- Training the model for all the stations took a lot of computation time. In hindsight, we can run the model only for important stations to reduce the compute time.
- Given more time for tuning better hyperparameters, along with better compute to operate on the samples, results could likely be improved further.
- Sparse/imbalanced data problems, for stations ids.
- Some rides did not have start/end stations.
- Inconsistency of data, where in the same dataset had different formats within columns.
- Weather data was merged and was used for EDA, however not used in prediction model.
- Due to high running time- we experimented clustering- but found that the data was too sparse for small clusters within the Jersey City region and it only worked for New York City data as there were enough number of rides within the smaller clusters.

Learning Experience

Accomplishments

- We able to identify a problem that can be replicated, across network
- As a team worked from start to finish we deployed code over the entire lifecycle of the project.
- Gained experience in graphical neural networks (none of the members in the team, before this project had worked on GNN’s).
- Understood the nuances of the deep learning concepts in application through this course.
- Honed our data handling skills to effectively manage complex data.
- Identified the importance of the teamwork, setting realistic timelines and dividing tasks based on inherent skill sets.

Future Considerations

Looking Forward

- We have shared the code to our Git repository and will continue to work on the project further.
- We intend to work on the model for the whole duration of dataset available.(before 2023)
- Work on improving the prediction window for beyond the present window.
- Work on identifying new station locations, expand the network beyond. Work on the NYC dataset where there are many more rides between stations.
- On an extended timeline work on a live data visualization and prediction model which is available publicly.

Git Repo

https://github.com/hiiamnikhil/IE434_Project_Deep_Dive7.git

Public Repo with all the team members as collaborators.

Link to google drive set up during the the project

https://drive.google.com/drive/folders/1Gj7FmEX9YPC2nalrhpvOYOHUmKhIF4Ct?usp=drive_link