Exploring the Usage of Divvy Bikes

PRESENTED BY:
McKenzie Campbell, Louise Wang, Howard Lin

Team

DIVY

Howard Lin

Louise Wang

McKenzie Campbell

Business Cases

- **Identifying** factors that affect bike usage on a localized level
- **Optimizing** the allocation of bikes numbers to each bike station
- **Building** new Divvy stations in areas with potential for growth

Executive Summary

- The purpose of this system is to support decision making process for Divvy Bike company including resource and infrastructure planning.
- By extracting, organizing, analyzing, visualizing data using a variety of platforms, we aim to identify the factors contributing to the usage and trends of Divvy Bike in Chicago.
- System functionality includes ETL process from raw data collectors, relational database for storage and access, BI reporting, connections to visualization tools.

Data Profile

Time Horizon: April 2020 - October 2020

	Description	Data Size	Shape	Key Variables	Matching Fields
Divvy Trips	Data broken down for each trip taken	814.4 MB	6,870,685 rows, 10 columns	start_station, start_datetime, end_station, end_datetime	Date, hour
Divvy Stations	Geographical information on each Divvy Bike Station	66 KB	669 row, 8 columns	station_geometry, station_total_docks	Zip code, region
Weather	Weather taken from Midway broken down by hour	9.6 MB	23,443 rows, 20 columns	temperature, weather type	Hour
Traffic	Traffic for each neighborhood in Chicago	799.1 MB	1,048,575 rows, 17 columns	region, ave_speed	Hour, region
Covid Cases	Cases by zip code in Chicago	26.6 MB	183,317 rows, 48 columns	zip codes, case measures	Date, zip code
Region	Geographical information for each neighborhood	5 KB	29 rows 3 columns	geometry	Region

Tools

Data Collection

Data preparation

Data Storage

Visualization

NoSQL

Data Preparation *[Trip]*

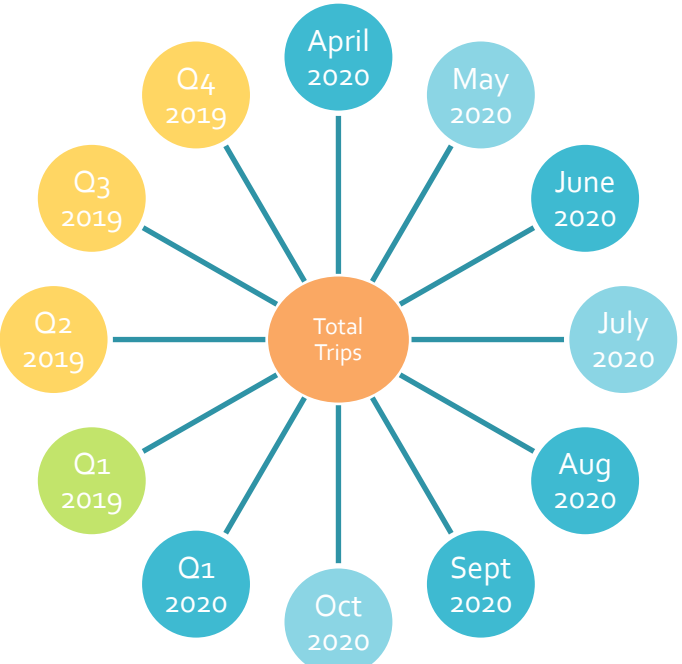
Jupyter Notebook

```
##cleaned.trips.Q2.2019.csv##

read_file = pd.read_csv ('/Users/KenzieKay/Desktop/Engineering Platforms    MSCA 31012/T

read_file.to_csv ('/Users/KenzieKay/Desktop/Engineering Platforms    MSCA 31012/Trip Dat

data10 = pd.read_csv('/Users/KenzieKay/Desktop/Engineering Platforms    MSCA 31012/Trip

data10 = data10.rename(columns={'01 - Rental Details Rental ID':'ride_id'})
data10.drop('01 - Rental Details Bike ID', inplace=True, axis=1)
data10.drop('01 - Rental Details Duration In Seconds Uncapped', inplace=True, axis=1)
data10 = data10.rename(columns={'03 - Rental Start Station ID':'start_station_id'})
data10 = data10.rename(columns={'03 - Rental Start Station Name':'start_station_name'})
data10 = data10.rename(columns={'02 - Rental End Station ID':'end_station_id'})
data10 = data10.rename(columns={'02 - Rental End Station Name':'end_station_name'})
data10 = data10.rename(columns={'User Type':'member_casual'})
data10.drop('Member Gender', inplace=True, axis=1)
data10.drop('05 - Member Details Member Birthday Year', inplace=True, axis=1)

data10['start_station_name'].replace('', np.nan, inplace=True)
data10['start_station_id'].replace('', np.nan, inplace=True)
data10['end_station_name'].replace('', np.nan, inplace=True)
data10['end_station_id'].replace('', np.nan, inplace=True)

data10.dropna(subset=['start_station_name'], inplace=True)
data10.dropna(subset=['start_station_id'], inplace=True)
data10.dropna(subset=['end_station_name'], inplace=True)
data10.dropna(subset=['end_station_id'], inplace=True)

new10a = data10["01 - Rental Details Local Start Time"].str.split(" ", n = 1, expand = True)
data10["started_date"]= new10a[0]
data10["started_time"]= new10a[1]
data10.drop(columns =["01 - Rental Details Local Start Time"], inplace = True)

new10b = data10["01 - Rental Details Local End Time"].str.split(" ", n = 1, expand = True)
data10["ended_date"]= new10b[0]
data10["ended_time"]= new10b[1]
data10.drop(columns =["01 - Rental Details Local End Time"], inplace = True)
```

Tools and Methods Utilized

Jupyter Notebook

- ❑ Drop columns that were not across files
- ❑ Dropped rows that did not have a start or stop station
- ❑ Rename columns to match
- ❑ Read to CSV for files that were were originally txt files.
- ❑ **12 original files**
- ❑ **Ended up with over 6 million rows**



Data Preparation *[Station]*

Jupyter Notebook

```
!pip install geopandas
```

...

```
import pandas as pd
import geopandas as gpd
```

```
stationpoints = pd.read_csv('/Users/howardlin/Desktop/DEPA data/Divvy_Bicycle_Stations.csv', skiprows=0, encoding='utf-
```

...

```
stationpoints=stationpoints.drop(columns='Location')
```

```
gdf_station = gpd.GeoDataFrame(stationpoints,
                               geometry=gpd.points_from_xy(stationpoints.Longitude,stationpoints.Latitude),
                               crs="EPSG:4326")
```

...

```
gdf_zip = gpd.read_file('/Users/howardlin/Desktop/DEPA data/ZIP_Codes.geojson')
```

...

```
sjoined_station = gpd.sjoin(gdf_station, gdf_zip, op="within")
```

Output

	A	B	C	D	E	F	G	H	I	J	K
1	ID	Station Name	Total Docks	Docks in Serv	Status	geometry	zip	REGION_ID	REGION		
2	715	Major Taylor Trail & 124th St	15	15	In Service	POINT (-87.63711333299999 41.668696353)	60628	28	Riverdale-Hegewisch		
3	695	Dauphin Ave & 103rd St	15	15	In Service	POINT (-87.607662678 41.707250368)	60628	26	Washington Hts-Roseland-Pullman		
4	698	Cottage Grove Ave & 111th Pl	15	15	In Service	POINT (-87.610001564 41.691709758)	60628	26	Washington Hts-Roseland-Pullman		
5	699	Vernon Ave & 107th St	15	14	In Service	POINT (-87.61225998399999 41.700121379)	60628	26	Washington Hts-Roseland-Pullman		
6	700	Halsted St & 96th St	15	15	In Service	POINT (-87.643019557 41.719712184)	60628	26	Washington Hts-Roseland-Pullman		
7	703	Greenwood Ave & 97th St	15	15	In Service	POINT (-87.597035766 41.718703155)	60628	26	Washington Hts-Roseland-Pullman		
8	1441	S Wentworth Ave & W 111th St	5	5	In Service	POINT (-87.62805179999999 41.69261760000001)	60628	26	Washington Hts-Roseland-Pullman		
9	725	Halsted St & 104th St	15	15	In Service	POINT (-87.64242410700001 41.7051397 89)	60628	26	Washington Hts-Roseland-Pullman		

Tools and Methods Utilized

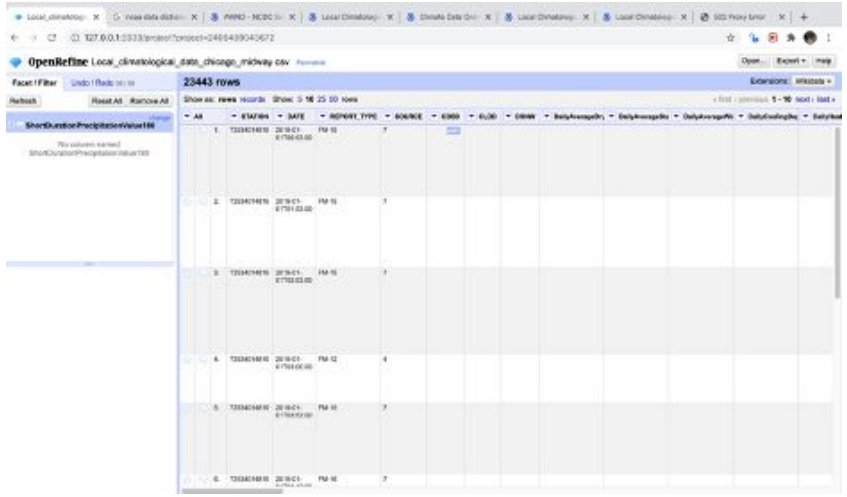
Jupyter Notebook (packages pandas,geopandas): Given the geojason file for area boundaries, we use Spatial Join in Geopandas to match each Divvy Bike station into **Zip Code** and **Neighborhood region**.

Data Preparation *[Weather]*

Jupyter Notebook

```
import pandas as pd

data = pd.read_csv('/Users/howardlin/Documents/Class documents/Data Engineering
/team4 project/data/Local_climatological_data_chicago_midway_daily.csv', skipro

ata['DATE'] = pd.to_datetime(data['DATE'])

ata['DAY_OF_WEEK'] = data.DATE.dt.dayofweek

data.rename(columns={'DATE': 'DATETIME'}, inplace=True)

data['DATE'] = data.DATETIME.dt.date

data['HOUR'] = data.DATETIME.dt.hour

data["TIME_CHECK"] = data["DATE"].astype(str) +" "+ data["HOUR"].astype(str)

data.REPORT_TYPE.value_counts()

data[data.DATE.astype(str)=='2019-09-15']
```

OpenRefine



Output

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	DATETIME	REPORT_TYF	SOURCE	HourlyAltime	HourlyDewPc	HourlyDryBul	HourlyPrecip	HourlyPreser	HourlyPressu	HourlyPressu	HourlyRelati	HourlySeaLe	HourlySkyCor	HourlyStatio	HourlyVisibil	HourlyWetBu	HourlyWindD	HourlyWindG	HourlyWindS	DAY_OF_WE	DATE
2	2019/1/1 0:53	FM-15	7	29.98	33	36	0		0	0	89	30	BKN:07 14 O	29.32	10	35	320	Null	10	1	2019/1/1
3	2019/1/1 1:53	FM-15	7	30.03	32	36	0		0	0	86	30.06	BKN:07 14 O	29.37	10	34	330	Null	13	1	2019/1/1
4	2019/1/1 2:53	FM-15	7	30.08	32	35	0		-0.12	3	89	30.1	BKN:07 12 B	29.41	8	34	340	Null	9	1	2019/1/1
5	2019/1/1 3:53	FM-15	7	30.11	31	34	0.005	Drizzle	0	0	89	30.14	OVC:08 13	29.44	7	33	330	Null	10	1	2019/1/1
6	2019/1/1 4:53	FM-15	7	30.13	31	33	0.005	Drizzle|Mist	0	0	92	30.16	BKN:07 7 BK	29.47	5	32	360	Null	7	1	2019/1/1
7	2019/1/1 5:53	FM-15	7	30.17	29	32	0.005	Drizzle|Mist	-0.09	3	88	30.2	SCT:04 8 BKN	29.5	5	31	20	Null	11	1	2019/1/1
8	2019/1/1 6:53	FM-15	7	30.22	27	31	0.005	Drizzle|Mist	0	0	85	30.25	OVC:08 14	29.55	6	30	10	20	15	1	2019/1/1
9	2019/1/1 7:53	FM-15	7	30.24	26	31	0		0	0	82	30.27	OVC:08 16	29.57	10	29	10	Null	11	1	2019/1/1
10	2019/1/1 8:53	FM-15	7	30.27	27	31	0		-0.09	1	85	30.3	OVC:08 14	29.6	10	30	350	Null	9	1	2019/1/1
11	2019/1/1 9:53	FM-15	7	30.29	27	31	0.005	Drizzle	0	0	85	30.32	OVC:08 13	29.62	7	30	340	Null	9	1	2019/1/1
12	2019/1/1 10:53	FM-15	7	30.29	28	31	0.005	Drizzle|Mist	0	0	89	30.32	BKN:07 10 O	29.62	6	30	330	Null	9	1	2019/1/1
13	2019/1/1 11:53	FM-15	7	30.28	28	31	0.005	Drizzle	-0.02	0	89	30.32	OVC:08 12	29.61	7	30	360	Null	11	1	2019/1/1
14	2019/1/1 12:53	FM-15	7	30.28	26	30	0.005	Drizzle	0	0	85	30.32	OVC:08 11	29.61	8	29	340	Null	10	1	2019/1/1
15	2019/1/1 13:53	FM-15	7	30.28	25	29	0.005	Drizzle	0	0	85	30.32	OVC:08 10	29.61	7	28	350	Null	8	1	2019/1/1
16	2019/1/1 14:53	FM-15	7	30.29	25	29	0.005	Drizzle	0	3	85	30.32	OVC:08 10	29.62	8	28	20	Null	8	1	2019/1/1
17	2019/1/1 15:53	FM-15	7	30.31	25	29	0.005	Drizzle	0	0	85	30.34	OVC:08 11	29.64	8	28	340	Null	8	1	2019/1/1
18	2019/1/1 16:53	FM-15	7	30.32	24	28	0.005	Drizzle	0	0	85	30.35	BKN:07 12 O	29.65	9	27	340	Null	6	1	2019/1/1
19	2019/1/1 17:53	FM-15	7	30.32	24	28	0.005	Drizzle	-0.03	1	85	30.36	SCT:04 13 BK	29.65	9	27	330	Null	7	1	2019/1/1

Tools and Methods Utilized

Jupyter Notebook (packages pandas,datetime) to get columns, rows and format needed.

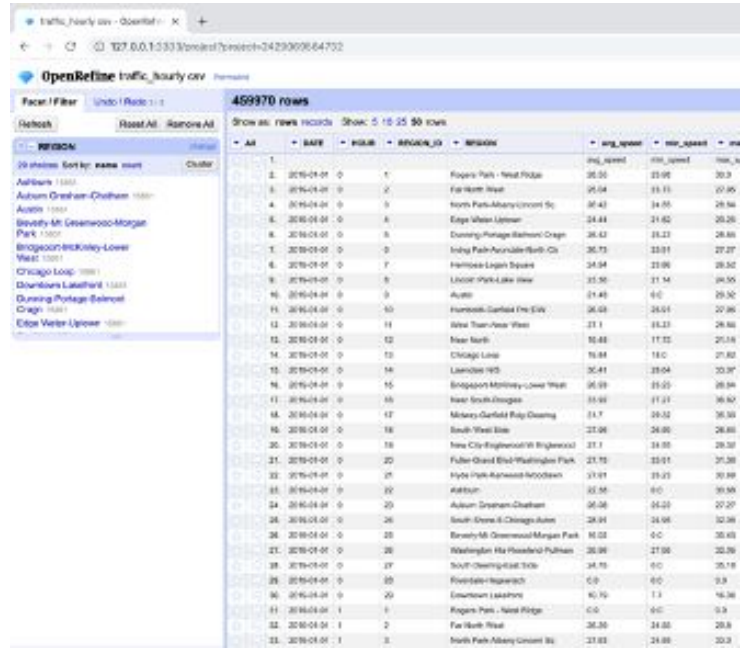
OpenRefine to translate technical weather code in the data.

Data Preparation *[Traffic]*

Jupyter Notebook

```
import pandas as pd
import numpy as np

ata = pd.read_csv('/Users/howardlin/Downloads/Chicago_Traffic_Tracker_-_Historical_Congestion_Estimates_by_Regi

ta['TIME1'] = pd.to_datetime(data['TIME'])

ta['DATE'] = data.TIME1.dt.date

ataselect = data[(data["TIME1"]>'2019-01-01 00:00:00') & (data["TIME1"]<='2020-11-15 23:59:59')]

dataselect=dataselect.drop(columns=['WEST','EAST','SOUTH','NORTH'])

dataselect=dataselect.sort_values(by=['TIME1','REGION_ID'])

dataselect['TIME'] = pd.to_datetime(dataselect['TIME'])

dataselect=dataselect.drop(columns=['TIME1'])

dataselect.to_csv('/Users/howardlin/Desktop/DEPA data/traffic.csv',
            index=False)

all_together = (dataselect.groupby(['DATE','MONTH','DAY_OF_WEEK','HOUR','REGION_ID','REGION','NW_LOCATION','SE_L
                  .agg({'SPEED': [np.mean, np.min, np.max]})
                  .rename(columns={'mean': 'avg_speed', 'amin': 'min_speed', 'amax': 'max_speed'}))

all_together=all_together.reset_index()
```

OpenRefine



Output

	A	B	C	D	E	F	G	H	I
1	DATE	MONTH	DAY_OF_WE	HOUR	avg_speed	min_speed	max_speed	REGION_ID	REGION
2	2019/1/1	1	1	0	28.55	25.98	30	1	Rogers Park - West Ridge
3	2019/1/1	1	1	0	25.04	23.73	27.95	2	Far North West
4	2019/1/1	1	1	0	26.42	24.55	28.64	3	North Park-Albany-Linconl Sq
5	2019/1/1	1	1	0	24.44	21.82	29.25	4	Edge Water-Uptown
6	2019/1/1	1	1	0	26.42	25.23	28.64	5	Dunning-Portage-Belmont Cragn
7	2019/1/1	1	1	0	26.73	25.91	27.27	6	Irving Park-Avondale-North Ctr

Tools and Methods Utilized

Jupyter Notebook (packages pandas,numpy, datetime): We adjusted the time granularity from 10 min to hourly by Groupedby function.

OpenRefine filter out redundant row and column.

Data Preparation *[Covid-19]*

Jupyter Notebook

```
In [20]: covid.head()
```

Out[20]:

	date	zipcode	confirmed_cases	confirmed_cases_change	total_tested	total_tested_change	confirmed_cases_2(
0	2020-11-27T00:00:00Z	60827	988	7	11878	91	
1	2020-11-27T00:00:00Z	60707	2966	22	31312	294	
2	2020-11-27T00:00:00Z	60661	444	3	7828	104	
3	2020-11-27T00:00:00Z	60660	1484	16	28208	315	
4	2020-11-27T00:00:00Z	60659	2181	18	21799	192	

5 rows × 49 columns

```
In [21]: covid['date2'] = covid.date1.dt.date

In [15]: covid['day_of_week'] = covid.date1.dt.dayofweek

In [16]: covid['month'] = covid.date1.dt.month
```

Microsoft Store

Output

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Fetch rows:

covid_record_id	date	late_update	month	day_of_week	weekend	zipcode	confirmed_cases_cumulative	confirmed_cases_increasing	tested_cases_cumulative	tested_cases_inc
1	2020-11-27	2020-12-05 03:50:55	11	4	Weekday	60827	988	7	11878	91
2	2020-11-27	2020-12-05 03:50:55	11	4	Weekday	60707	2966	22	31312	294
3	2020-11-27	2020-12-05 03:50:55	11	4	Weekday	60661	444	3	7828	104
4	2020-11-27	2020-12-05 03:50:55	11	4	Weekday	60660	1484	16	28208	315
5	2020-11-27	2020-12-05 03:50:55	11	4	Weekday	60659	2181	18	21799	192
6	2020-11-27	2020-12-05 03:50:55	11	4	Weekday	60657	2669	28	55367	699
7	2020-11-27	2020-12-05 03:50:55	11	4	Weekday	60656	1577	11	16076	104
8	2020-11-27	2020-12-05 03:50:55	11	4	Weekday	60655	1891	24	19074	163
9	2020-11-27	2020-12-05 03:50:55	11	4	Weekday	60654	1070	9	16474	227
10	2020-11-27	2020-12-05 03:50:55	11	4	Weekday	60653	1436	15	25050	181
11	2020-11-27	2020-12-05 03:50:55	11	4	Weekday	60652	3198	28	26009	217
12	2020-11-27	2020-12-05 03:50:55	11	4	Weekday	60651	4577	40	37243	297
13	2020-11-27	2020-12-05 03:50:55	11	4	Weekday	60649	1898	22	29708	223
14	2020-11-27	2020-12-05 03:50:55	11	4	Weekday	60647	4848	54	74772	747
15	2020-11-27	2020-12-05 03:50:55	11	4	Weekday	60646	1339	11	19142	135
16	2020-11-27	2020-12-05 03:50:55	11	4	Weekday	60645	2599	20	35393	317
17	2020-11-27	2020-12-05 03:50:55	11	4	Weekday	60644	2526	26	31269	133
18	2020-11-27	2020-12-05 03:50:55	11	4	Weekday	60643	2431	24	40567	317

Design Consideration

Data Types	Dealing with N/As	Using Dimensional Table	Expected Output of Data Analysis
• INT: id, key, numbers without decimals • DATE: date • VARCHAR: name, type • Decimal: Other non-integral number	• Weather: fill N/A with nearest previous value • Covid: treat N/A with 0 • Other dataset: drop row with N/A	• Aggregate and maintain historical data • Optimized for analytical queries • More denormalized and reduces time on processing	• Consumer behavior revealed by data pattern that assists operational decision-making • Relationship between ridership and environmental factors: weather, covid-19, traffic

Data Quality

Completeness

Missing values in covid-19 dataset were treated 0.
Cut down redundant dataset to make sure each dataset has same time period.

Consistency

Data format is consistent throughout the database.

Conformity

Standardize the data format (datetime, floats, etc.) to make sure they are following the set of standard data definitions throughout the database.

Accuracy

Data are transformed by the processes (sum, average, max, min, etc.) which do not affect their reflection of reality.

Uniqueness

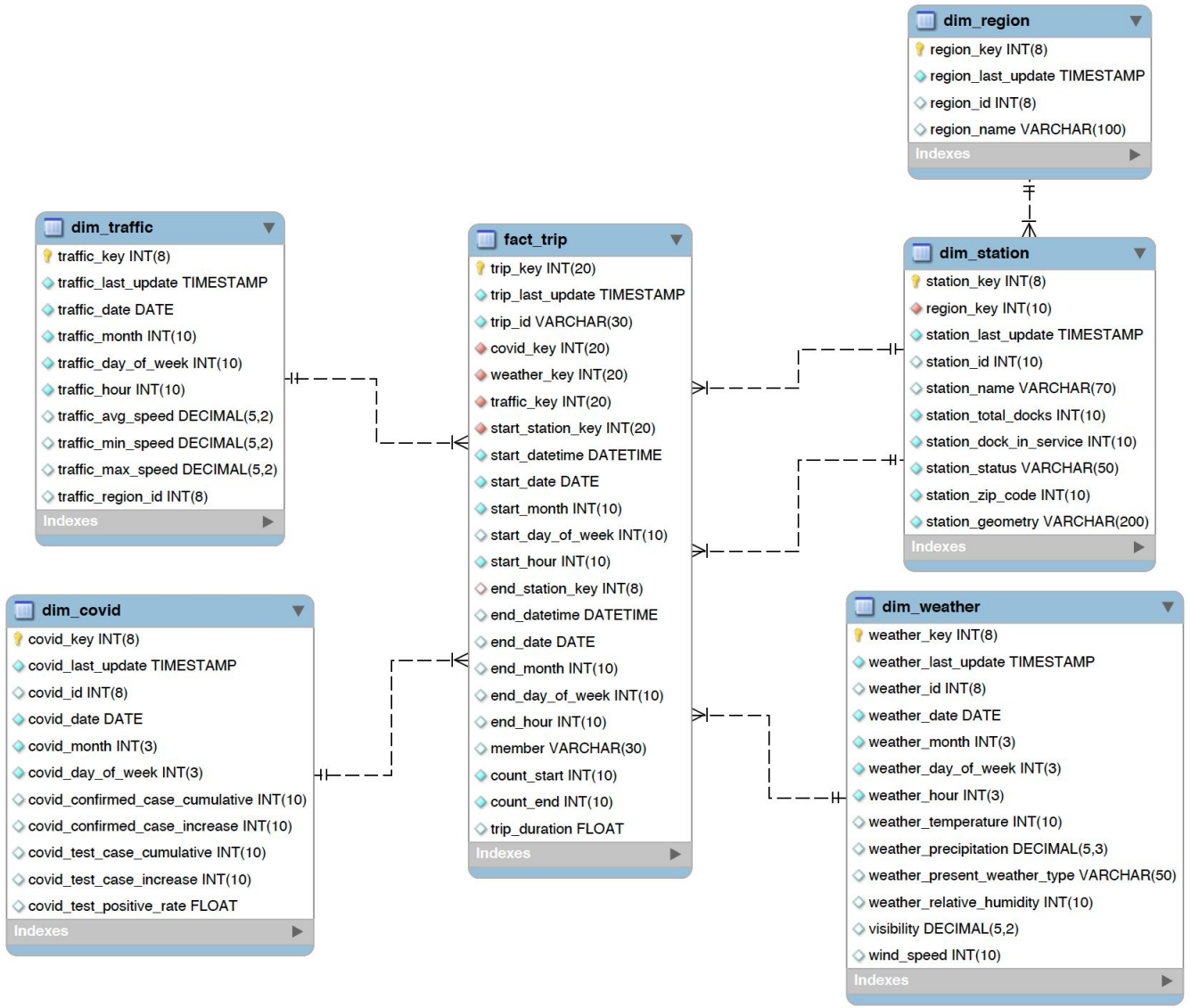
No duplicated data.

Timeliness

Data collected reflect up-to-date information or include every entries within the time period as designed.

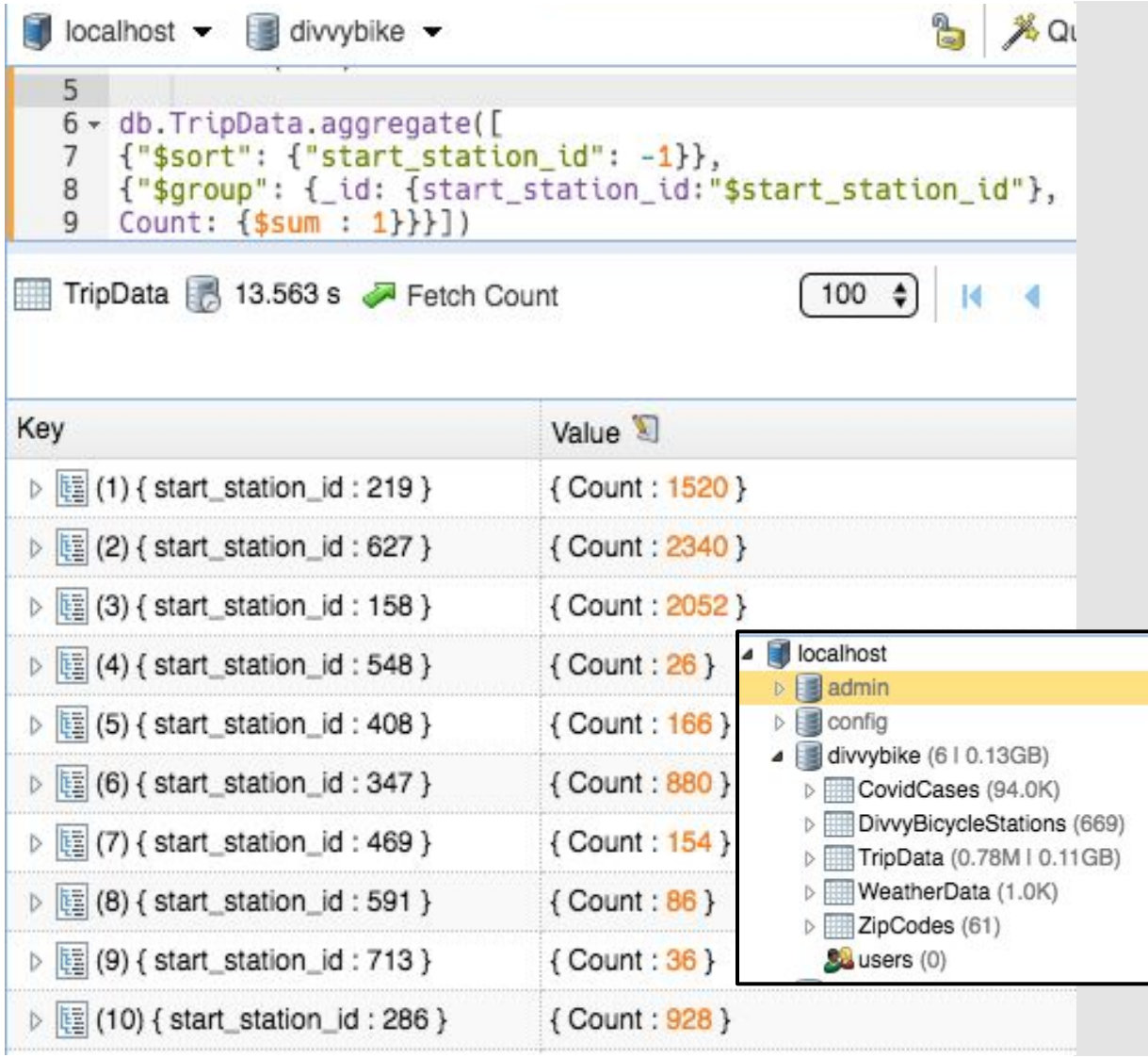
Dimensional Model

Linked by Start Time
or
Linked by Start Region

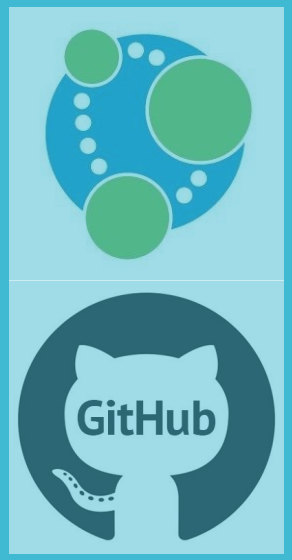


NoSQL
[MongoDB]

- Easy to manage large amounts of data
- High speed quarries
- Easily scalable
- Limited analytical capacity

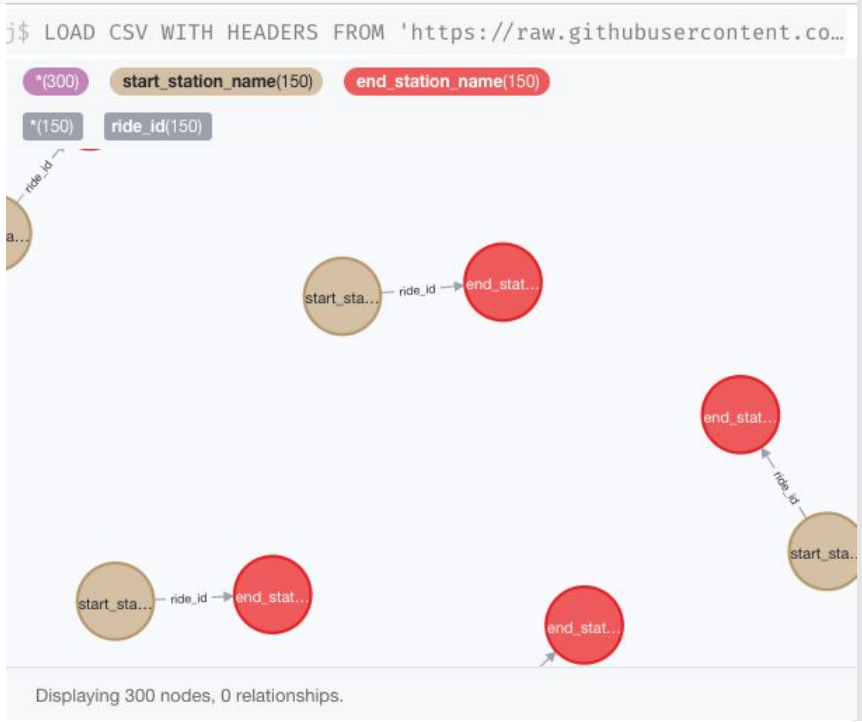


NoSQL
[Neo4j]




```
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/McKenzieKay/DivvyBike/main/202004-divvy-tripdata.csv'AS line
MERGE(R:ride_id{name:'ride_id'})
CREATE (m:start_station_name { name: 'start_station_name'})
CREATE (n:end_station_name {name: 'end_station_name'})
CREATE (m)-[:ride_id]→(n)
RETURN m,n
```

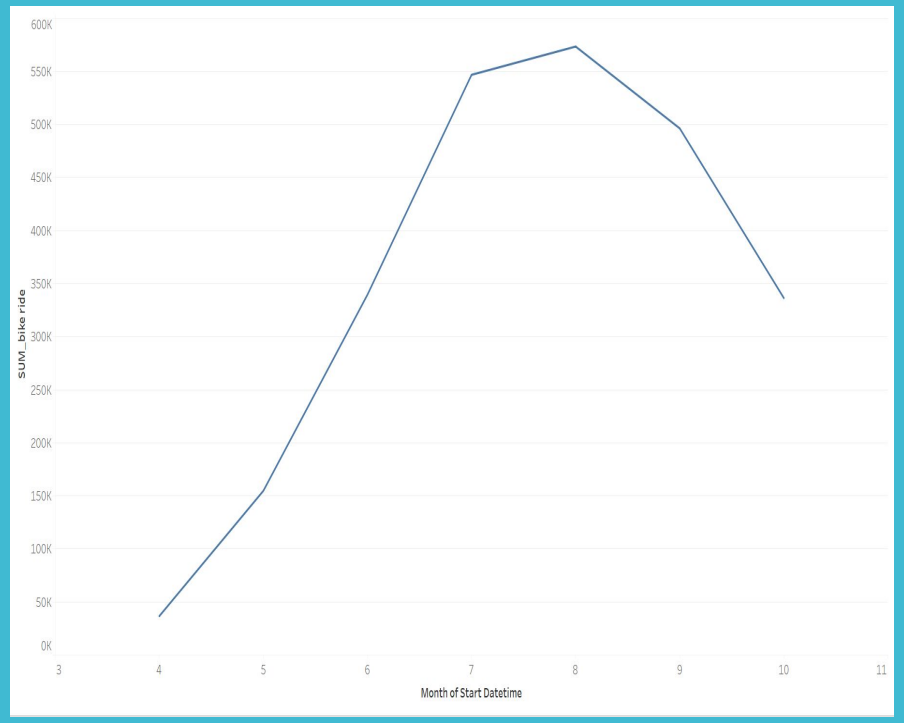
- Had More Troubles
- Created a URL on GitHub to Upload .CSV File
- Still Needs Work on Matching and Creating Relationships



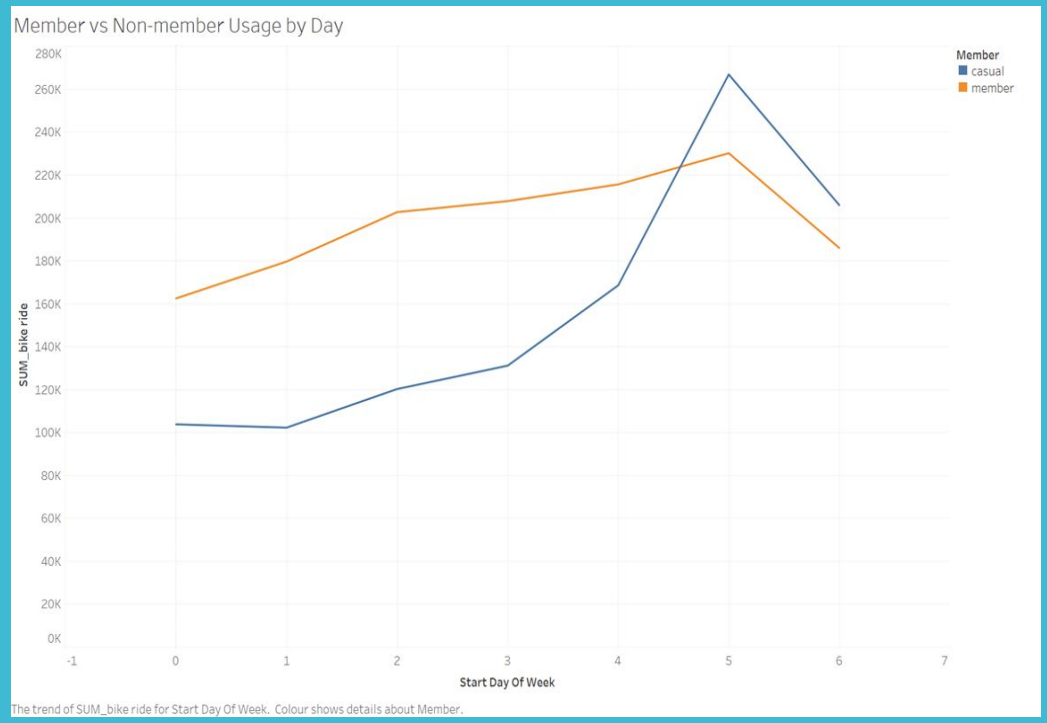
Identifying Factors

- Day of week / Time of Year
- Weather Type/ Temperature
- Covid Cases
- Automobile Traffic

Bike Usage by Month



Member vs Non-Member Bike Usage by Day-of-Week



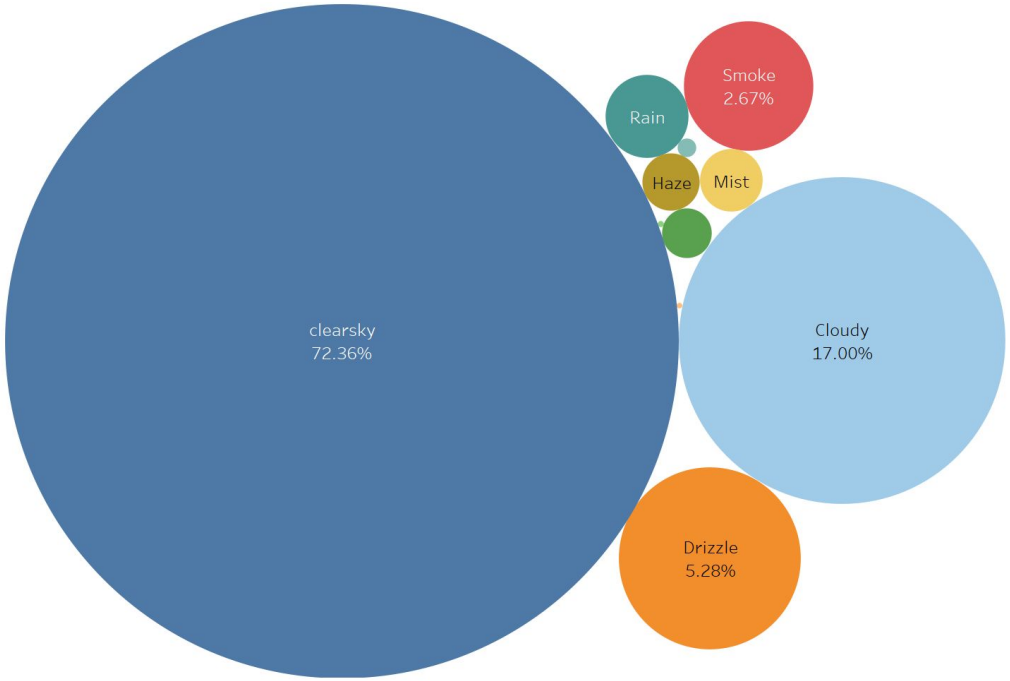
Weather vs Avg Daily Ridership per Hour

Weather..	Apr	May	Jun	Jul	Aug	Sep	Oct
Clearsky	196.2	285.8	577.7	816.0	809.6	800.7	522.9
Cloudy	81.6	281.9	347.0	611.6	807.6	657.9	365.7
Drizzle	73.5	125.3	404.3	566.5	940.4	429.6	317.3
Rain	34.1	28.7	340.3	414.9	376.4	190.4	310.0

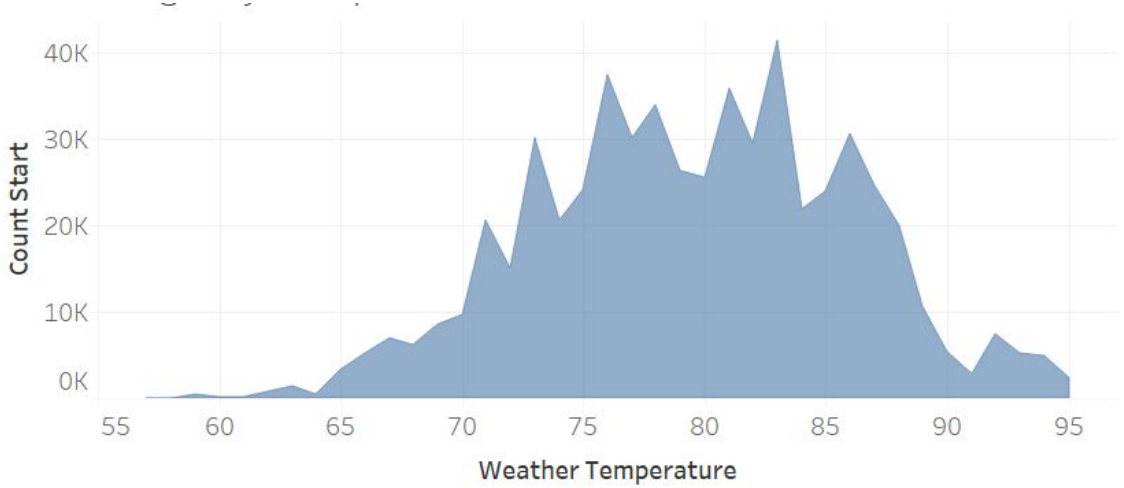
Avg Daily Ridership per..

28.7 940.4

Bike Usage by Weather Type

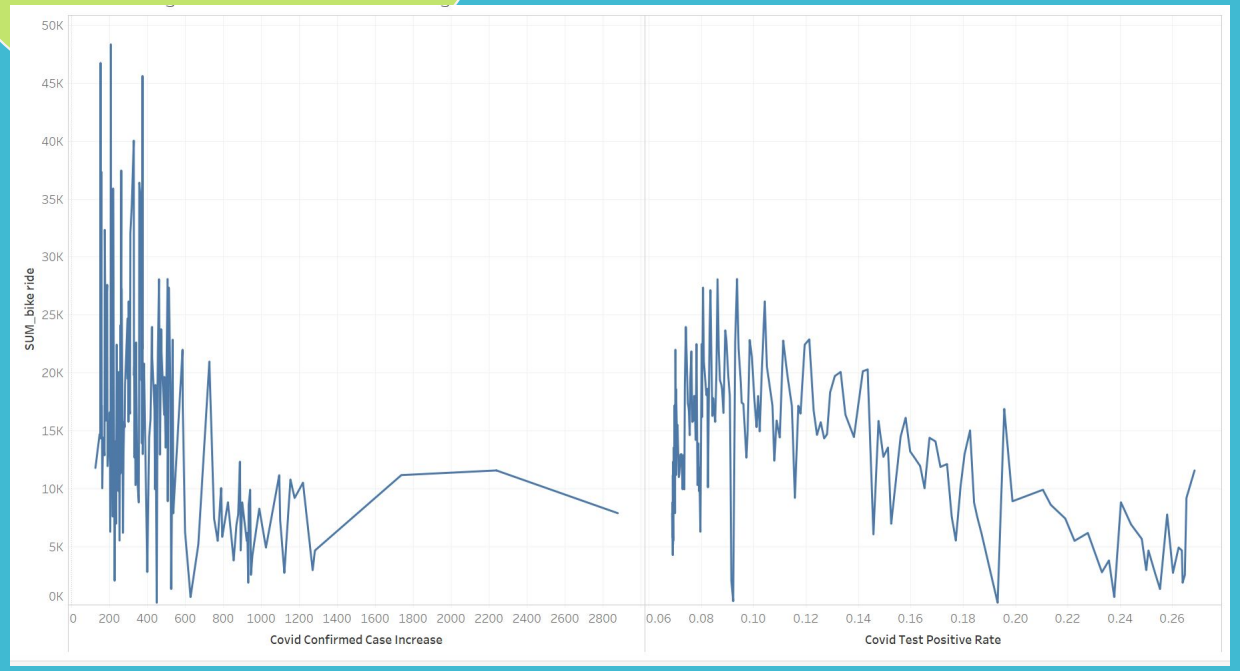


Bike Usage by Temperature



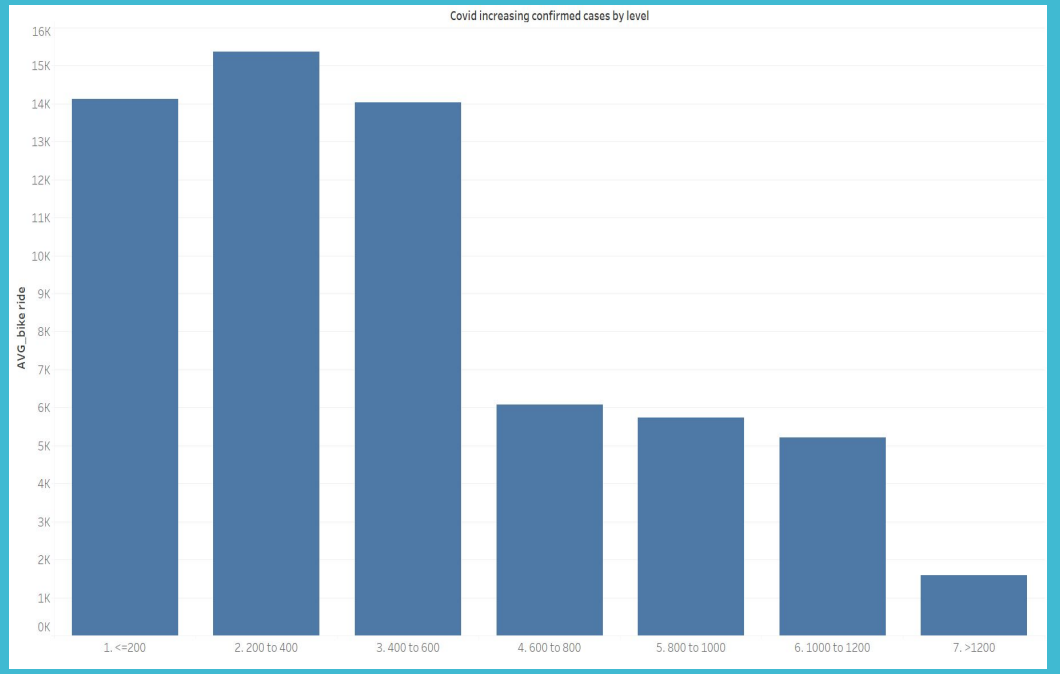
Analysis

Count_Bike rides



Covid Daily Cases

Covid Daily Test Positive Rate



Covid Daily Increasing Cases by Group vs Avg Bike Rides

Traffic Speed vs Ridership by Region

Color: avg traffic speed | Height: Ridership

Region Name	Range of avg speed			
	< 10	10 to 20	20 to 30	> 30
Lincoln Park-Lake View	78.2	246.0	137.8	8.2
Chicago Loop	96.4	114.0	40.4	3.5
Near North	33.4	110.9	72.7	8.3
West Town-Near West	93.5	100.1	66.9	3.2
Downtown Lakefront	15.3	99.0	58.5	10.8
Edge Water-Uptown	8.7	55.4	31.9	3.2
Hermosa-Logan Square	7.3	31.1	21.4	5.8
Near South-Douglas	12.4	10.8	30.4	12.0
Bridgeport-McKinley-Lower West	15.2	4.8	12.4	2.5
Irving Park-Avondale-North Ctr	5.7	11.3	11.0	2.2
Hyde Park-Kenwood-Woodlawn	7.7	5.8	17.9	4.0
North Park-Albany-Linconl Sq	9.9	4.4	12.2	1.8
Rogers Park - West Ridge	2.7	2.3	8.0	2.0
Humboldt-Garfield Prk E/W	4.5	2.9	4.1	
South Shore-S Chicago-Avlon	2.3	1.8	2.6	1.6
Fuller-Grand Blvd-Washington Park	3.9	1.5	3.5	1.6
Lawndale N/S	1.8	2.1	2.2	1.4
New City-Englewood-W Englewood	1.8	1.3	1.8	1.3
Austin	1.6	1.4	1.7	1.5
Riverdale-Hegewisch	1.7	1.8	1.3	1.5
South Deering-East Side		2.0	1.4	1.4
Dunning-Portage-Belmont Cragn	1.0	1.3	1.3	1.0
Auburn Gresham-Chatham	1.2	1.4	1.5	
Washington Hts-Roseland-Pullman	1.5	1.0	1.7	1.7
Beverly-Mt Greenwood-Morgan Pa..	1.0		1.5	1.4

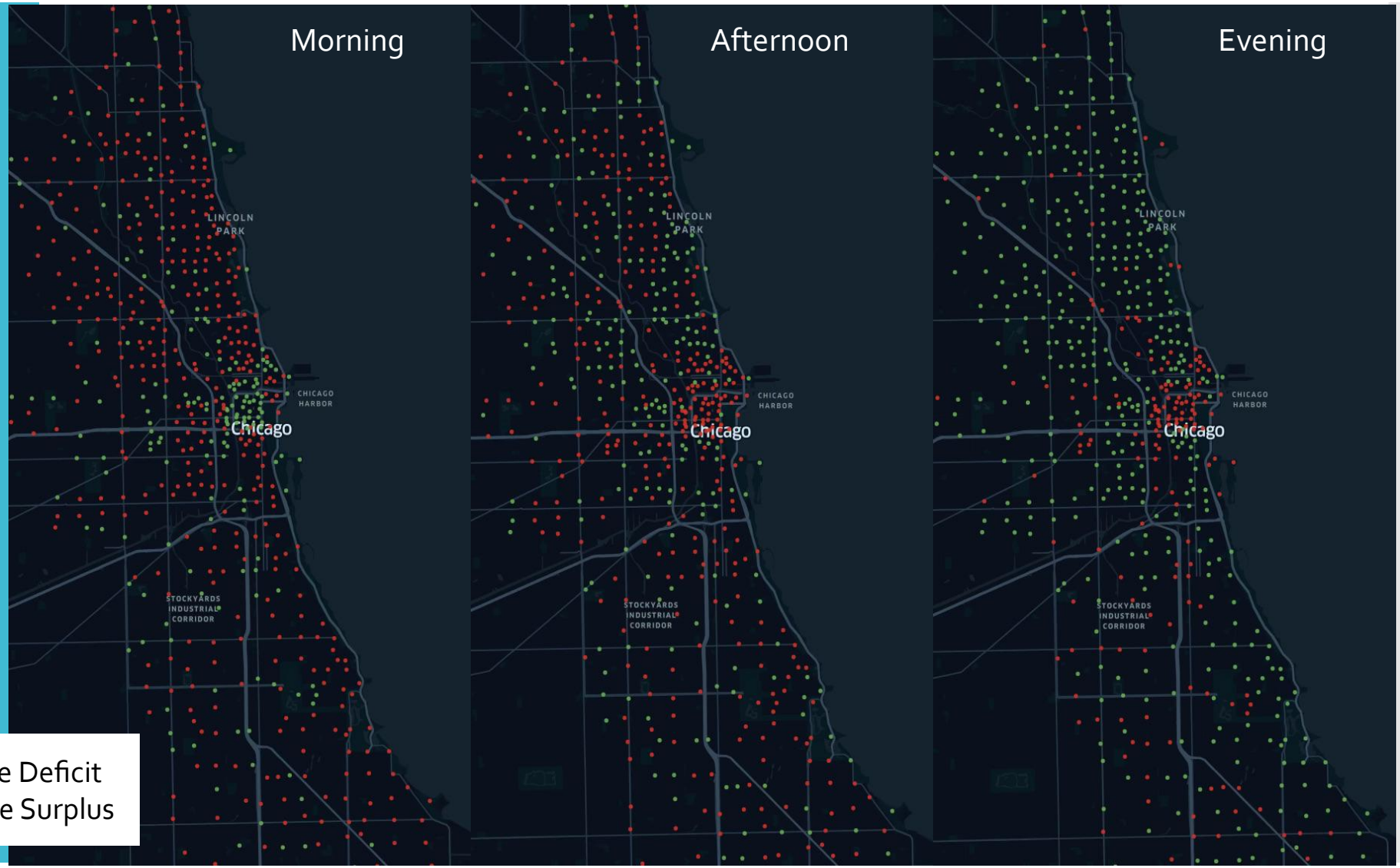
Optimizing Allocation

Morning

Traffic is flowing to 'L' stops and Downtown

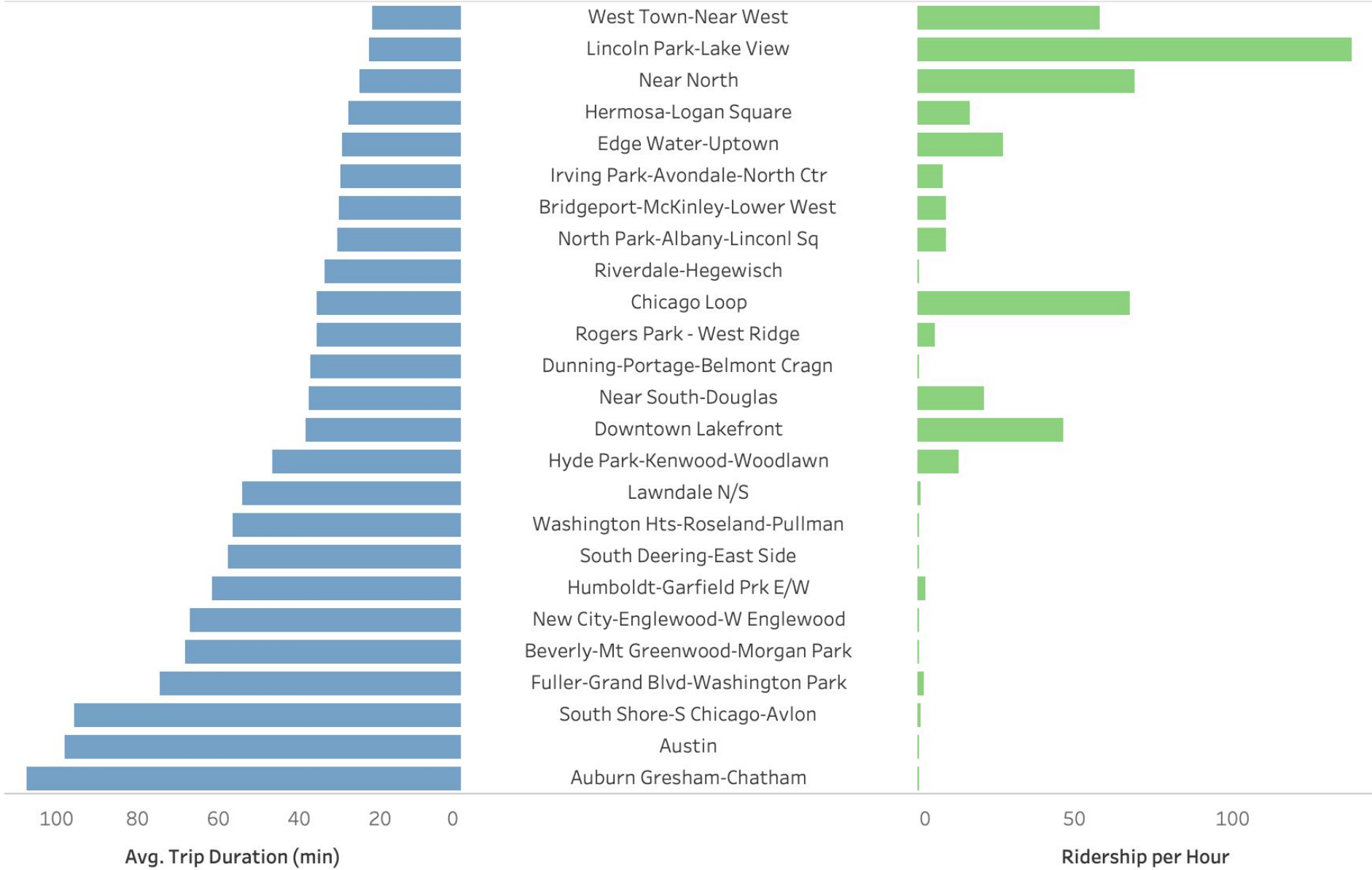
Evening

Flowing from Downtown Out



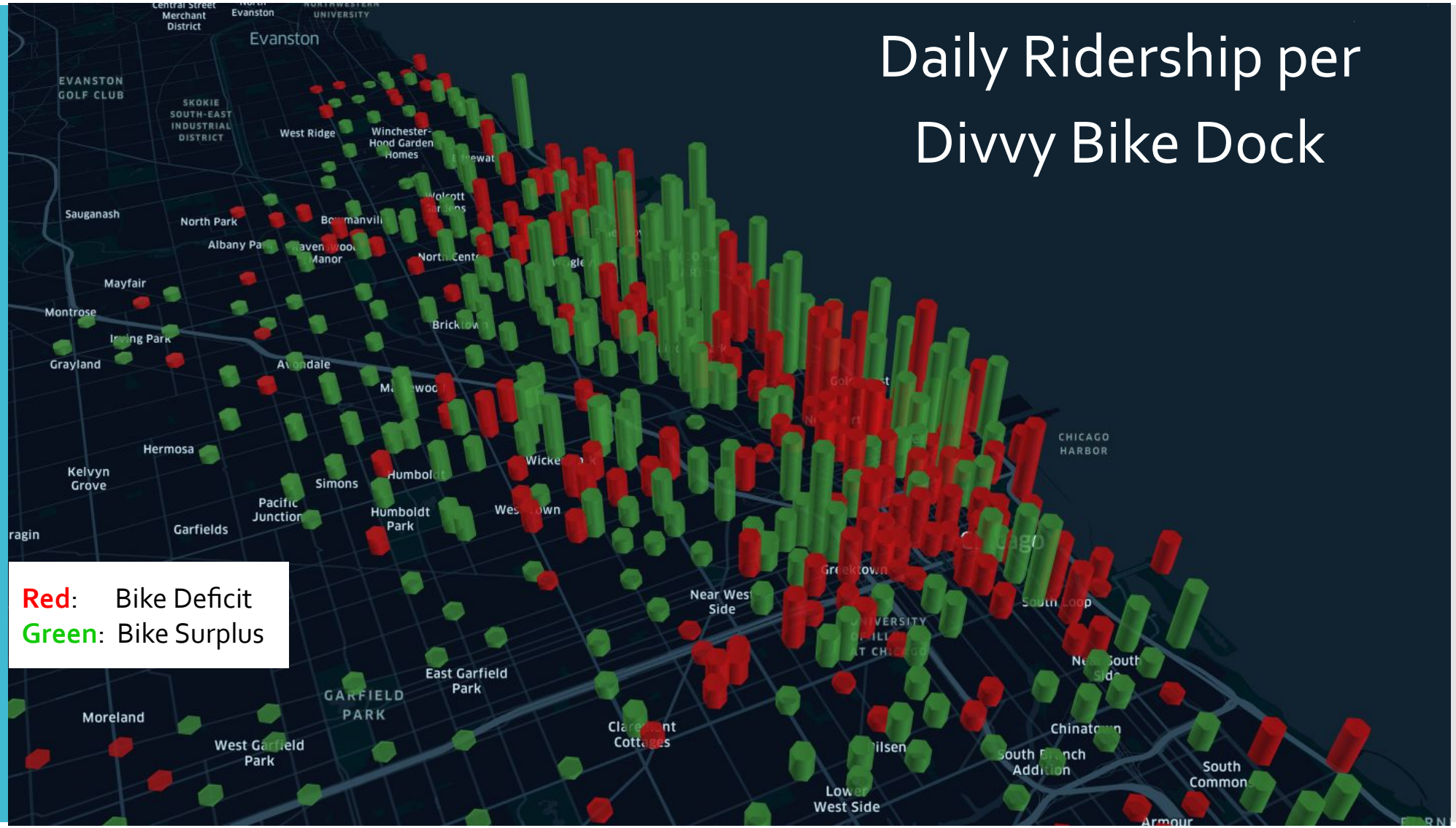
Red: Bike Deficit
Green: Bike Surplus

Trip Duration & Ridership by Region



Building New Stations

Add in the vicinity of TALL and RED column



Recommendations

Identifying Local Factors

Strong Relationship:

- Day of week / Time of Year
- Weather Type/ Temperature
- Covid Cases

Weak Relationship:

- Traffic

Optimizing the allocation of bikes

- Morning: More Bikes in Suburban Areas
- Evening: More Bikes in Downtown and 'L' Station

Building new Divvy Stations

Areas with:

- Bike Deficit
- High Ridership

Thank You For Joining Us!

-McKenzie Campbell, Louise Wang, Howard Lin

Appendix

Future Work

- Covid-19 and Weather data with better geographic granularity. (eg. by neighborhood region or by zip code)
- Introduce more informations (demographic data, surrounding public transportation, etc.)
- Better time granularity of covid-19 data will help us find more correlation with ridership
- Visualizing route pattern will provide more information of rider habit

Lesson Learned

- **Problem**: We had problem directly loading CSV files on MySQL.
  (LOAD DATA LOCAL INFILE "<filelocation>" INTO TABLE <table name>)

  Solution:Import wizard or GCP cloud is used instead.

- **Problem**: Long waiting time importing large data on Mysql shell.

  Solution: Upload data into cloud storage and import them directly into cloud sql.

- **Problem**: Original traffic data contains traffic speed in every 10 mins which is far more than we need as by hour.

  Solution: *pandas.DataFrame.groupby* & *pandas.DataFrame.reset_index*

Lesson Learned

- **Problem**: Data are truncated during insert.

  Solution: Make sure data format in target field match with selected source.

- **Problem**: Fail to populate large fact table due to small script mistake.

  Solution: Try to run the query in smaller dataset first.

- **Problem**: DML with multiple join queries prolong the computation.

  Solution: Adding index helps system navigate data and facilitates the matching process.

References

Data Sources

- Divvy Bike historical trip data:
  divvybikes.com
  https://www.divvybikes.com/system-data
- Divvy Bike station location:
  City of chicago data portal
  https://data.cityofchicago.org/Transportation/Divvy-Bicycle-Stations/bbyy-e7gq
- Geographic mapping (zip codes boundaries):
  City of chicago data portal
  https://data.cityofchicago.org/Facilities-Geographic-Boundaries/Boundaries-ZIP-Codes/gdcf-axmw
- Weather (Chicago midway weather station):
  NOAA data access
  https://www.ncdc.noaa.gov/cdo-web/datatools/lcd
- Covid-19 weekly cases by zip code:
  Illinois Department of Health
  Historical Illinois COVID-19 ZIP code data / COVID-19 in Illinois / Observable (observablehq.com)
- Chicago Traffic Tracker
  City of chicago data portal
  https://data.cityofchicago.org/Transportation/Chicago-Traffic-Tracker-Historical-Congestion-Esti/kf7e-cur8