COSC 5010 - Data Science for Security Final Project

Libao Jin (ljin1@uwyo.edu)

May 2, 2018

1 Motivation

In metropolitan areas, different kind of public transportations such as buses, taxis, metros, etc., play an essential role in people's daily life. However, it is not as efficient as we expected sometimes. Hence, this project mainly looks into the public transportation, i.e., BART, to make some useful suggestion. Here are a bunch questions this project aims to answer:

1. Which BART station is the busiest?
2. What is the least popular BART route?
3. When is the best time to go to SF if you want to find a seat?
4. Which day of the week is the busiest?
5. How many people take the BART late at night?
6. Does the BART ever stop in a station without anyone going off and on?

2 Background

BART, short for "Bay Area Rapid Transit", is the transit system severing the San Francisco Bay Area in California. BART operates six routes, 46 stations, and and 112 miles of track. It serves an average weekday ridership of 423,000 people, making it the fifth-busiest rapid transit system in the United States.
This dataset contains daily information on BART ridership for a period covering all of 2016 and part of 2017. Unlike some other rapid transit system datasets, this data includes movements between specific stations (there are just over 2000 station-to-station combinations).

3 Project Summary & Major Tasks

3.1 Interpretation of the Datasets

We get the data from [?], which provides the following three datasets:

- `data-hour-soo-dest-2016.csv`: Number of passengers (Throughput) that went between two stations (Origin and Destination) in a given time (DateTime) in 2016 as Table ??.
- `data-hour-soo-dest-2017.csv`: Number of passengers (Throughput) that went between two stations (Origin and Destination) in a given time (DateTime) in 2017.
- `station_info.csv`: Information about different BART stations (Location, Description, etc.) as Table ??.

3.2 Data Preprocessing/Data Cleaning

Since the data is well-collected, there is not much cleaning required. But in order to make the data analysis process easier, we'd like to preprocess the data such as converting the string read from text files into desired type and format.

Table 1: First five rows of data-hour-soo-dest-2016.csv

Origin	Destination	Throughput	DateTime
12TH	12TH	1	2016-01-01 00:00:00
12TH	16TH	1	2016-01-01 00:00:00
12TH	24TH	4	2016-01-01 00:00:00
12TH	ASHB	4	2016-01-01 00:00:00
12TH	BALB	2	2016-01-01 00:00:00

3.2.1 Dataset Reconstruction

- Convert the string of column `DateTime` of data `data-hour-soo-dest-2016.csv` and `data-hour-soo-dest-2017.csv` to `pandas.Datetime` type as Table **??**.

- Extract `Longitude` and `Latitude` from `Location` for the visualization of routes later on, the preprocessed table is as shown in Table **??**.

3.2.2 Merge the Datasets of 2016 and 2017

Using Pandas to read the `.csv` file, and store each dataset as a `DataFrame`, then using the `pandas.concat` to merge two `DataFrames`.

3.3 Assumptions

In order to find how busy each station is, we consider the throughput of each station as the origin, where riders take the transit. To do so, we made following assumptions:

1. In the dataset, the value of the column `DateTime` is the time that riders arrive at the Origin and take the transit to Destination. Thus, the arriving time is unknown in this case.
2. The throughput of a station is, actually, the number of riders that the train/trainsit carry, rather than the number of people who are waiting in the station.

3.4 Calculation of the Daily Throughput of Each Station from 2016 to 2017

We sum up all the throughput of the same origin as the total throughput of that station in the year of 2016 and 2017. Then, we sort the stations by the total throughput and use visualization to find the busiest station.

3.5 Calculation of the Total Hourly Throughput of Each Station of Weeks/-Months

We group the stations by weekdays/months, then sum up the hourly throughput of each station and then visualize them to find the best time to take the transit on a certain weekday or in certain month.

3.6 Build a Predictor of Throughput for BART Using Machine Learning

We pick the following features to train the model: month of year, day of month, day of week, time, origin, destination, origin longitude, origin latitude, destination latitude, destination longtitude. And then split the datasets into training dataset and test dataset. Using linear regression and decision tree, which provided by the Python module `scikit-learn`. Before we start, we set the criteria to indicate the level of workload of the BART (Table **??**).

Table 2: Stations information from stat_info.csv

Abbreviation	Description	Location	Name
12TH	1245 Broadway,...	-122.2714,37.8037,0	12th St. Oakland City Center (12TH)
16TH	2000 Mission S...	-122.4196,37.7650,0	16th St. Mission (16TH)
19TH	1900 Broadway,...	-122.2686,37.8083,0	19th St. Oakland (19TH)
24TH	2800 Mission S...	-122.4181,37.7524,0	24th St. Mission (24TH)
ASHB	3100 Adeline S...	-122.2700,37.8528,0	Ashby (ASHB)
BALB	401 Geneva Ave...	-122.4475,37.7215,0	Balboa Park (BALB)
BAYF	15242 Hesperia...	-122.1265,37.6969,0	Bay Fair (BAYF)
CAST	3301 Norbridge...	-122.0756,37.6907,0	Castro Valley (CAST)
CIVC	1150 Market St...	-122.4141,37.7797,0	Civic Center/UN Plaza (CIVC)
COLS	7200 San Leand...	-122.1968,37.7536,0	Coliseum/Oakland Airport (COLS)
COLM	365 D Street, ...	-122.4662,37.6846,0	Colma (COLM)
CONC	1451 Oakland A...	-122.0290,37.9737,0	Concord (CONC)
DALY	500 John Daly ...	-122.4690,37.7061,0	Daly City (DALY)
DBRK	2160 Shattuck ...	-122.2681,37.8701,0	Downtown Berkeley (DBRK)
DELN	6400 Cutting B...	-122.3167,37.9250,0	El Cerrito del Norte (DELN)
DUBL	5801 Owens Dr....	-121.8991,37.7016,0	Dublin/Pleasanton (DUBL)
EMBR	298 Market Str...	-122.3970,37.7928,0	Embarcadero (EMBR)
FRMT	2000 BART Way,...	-121.9766,37.5574,0	Fremont (FRMT)
FTVL	3401 East 12th...	-122.2241,37.7748,0	Fruitvale (FTVL)
GLEN	2901 Diamond S...	-122.4338,37.7330,0	Glen Park (GLEN)
HAYW	699 'B' Street...	-122.0870,37.6697,0	Hayward (HAYW)
LAFY	3601 Deer Hill...	-122.1246,37.8931,0	Lafayette (LAFY)
LAKE	800 Madison St...	-122.2651,37.7970,0	Lake Merritt (LAKE)
MCAR	555 40th Stree...	-122.2670,37.8290,0	MacArthur (MCAR)
MLBR	200 North Roll...	-122.3867,37.6002,0	Millbrae (MLBR)
MONT	598 Market Str...	-122.4010,37.7894,0	Montgomery St. (MONT)
NBRK	1750 Sacrament...	-122.2834,37.8739,0	North Berkeley (NBRK)
NCON	3700 Port Chic...	-122.0246,38.0031,0	North Concord/Martinez (NCON)
OAKL	1 Airport Driv...	-122.2121,37.7132,0	Oakland Airport (OAKL)
ORIN	11 Camino Pabl...	-122.1837,37.8783,0	Orinda (ORIN)
PHIL	1365 Treat Blv...	-122.0560,37.9284,0	Pleasant Hill/Contra Costa Centre (PHIL)
PITT	1700 West Lela...	-121.9451,38.0189,0	Pittsburg/Bay Point (PITT)
PLZA	6699 Fairmount...	-122.2989,37.9026,0	El Cerrito Plaza (PLZA)
POWL	899 Market Str...	-122.4079,37.7844,0	Powell St. (POWL)
RICH	1700 Nevin Ave...	-122.3530,37.9368,0	Richmond (RICH)
ROCK	5660 College A...	-122.2513,37.8447,0	Rockridge (ROCK)
SANL	1401 San Leand...	-122.1608,37.7219,0	San Leandro (SANL)
SBRN	1151 Huntingto...	-122.4162,37.6377,0	San Bruno (SBRN)
SFIA	International ...	-122.3924,37.6159,0	San Francisco Int'l Airport (SFIA)
SHAY	28601 Dixon St...	-122.0571,37.6343,0	South Hayward (SHAY)
SSAN	1333 Mission R...	-122.4439,37.6642,0	South San Francisco (SSAN)
UCTY	10 Union Squar...	-122.0173,37.5906,0	Union City (UCTY)
WARM	45193 Warm Spr...	-121.9393,37.5021,0	Warm Springs/South Fremont (WARM)
WCRK	200 Ygnacio Va...	-122.0675,37.9055,0	Walnut Creek (WCRK)
WDUB	6501 Golden Ga...	-121.9282,37.6997,0	West Dublin/Pleasanton (WDUB)
WOAK	1451 7th Stree...	-122.2951,37.8048,0	West Oakland (WOAK)

Table 3: First five rows of converted DataFrame of data-hour-soo-dest-2016.csv

Origin	Destination	Throughput	DateTime	Date	Time	DayOfWeek
12TH	12TH	1	2016-01-01 00:00:00	2016-01-01	00:00:00	Friday
12TH	16TH	1	2016-01-01 00:00:00	2016-01-01	00:00:00	Friday
12TH	24TH	4	2016-01-01 00:00:00	2016-01-01	00:00:00	Friday
12TH	ASHB	4	2016-01-01 00:00:00	2016-01-01	00:00:00	Friday
12TH	BALB	2	2016-01-01 00:00:00	2016-01-01	00:00:00	Friday

4 Results

4.1 Visualization of the Data

4.1.1 Visualization of the Routes of BART Lines

To visualize the routes, we'd like to utilize the geographic location provided by Location columns in `station_info.csv`, which consists of longitude and latitude. The dataset, however, does not provide the exact lines directly. Hence, we obtained the five lines on the website of the BART [?]. Then we obtained the plots of BART lines as Figure **??** shown.

4.1.2 Visualization of the Throughput of Each Station

As shown in Figure **??**, it is easy to find that the busiest stations (bubbles with large radius or of magenta) are in the downtown of San Francisco, i.e., Montgomery St. Station, Embarcadero Station, Powell St. Station, Civic Center/UN Plaza Station, etc.

Also, we obtained the barplot for the avergae hourly throughput of each station of 2016 and 2017 as shown in **??**. Obviously, the throughputs of aforementioned four stations far outweight that of the rest stations.

4.2 Daily Throughput of Each Station of 2016 and 2017

The plots of throughput of each station of 2016 and 2017 is as shown in Figure **??**, from which we can have a sense of the volume of riderships in each station.

4.3 Hourly Throughput of Each Station of a Week

It is shown in Figure **??**, **??**, **??**, **??**, **??** that busy hours of each stations on each day of the week. Each row represents the throughput of a certain station while the columns represent the days of week, i.e., the first column stands for Monday, the second for Tuesday, and the like.

In the top four busiest stations, the peak hours are mainly around 17:00-18:00, during which people take the transit to go home. These stations are in downtown San Francisco. Presumbly that people come here to work/shopping, etc. As for those stations have peak hours around 9:00-10:00, that indicate those stations are close to the dwelling where people live. And from these plots, we can get the best time (by avoiding the rush hours) to take the BART in order to find a seat.

4.4 Linear Regression

Turns the accuracy of the linear regression is pretty low (score: 0.8%). So... we skip this one.

4.5 Decision Tree

We used Decision Tree to build a predictor of the throughput of stations. Among features such as month of year, day of month, day of week, time, origin, destination, origin longitude, origin latitude, destination latitude, destination longtitude, we find that the combination of features day of week, time, origin and desination are most effetive. Only use 0.02% of the data as our training dataset (Figure **??**), we can reach

Table 4: Preprocessed stations information from stat_info.csv

Abbreviation	Name	Longitude	Latitude	Description
12TH	12th St. Oakland City Center (12TH)	-122.271450	37.803768	1245 Broad...
16TH	16th St. Mission (16TH)	-122.419694	37.765062	2000 Missi...
19TH	19th St. Oakland (19TH)	-122.268602	37.808350	1900 Broad...
24TH	24th St. Mission (24TH)	-122.418143	37.752470	2800 Missi...
ASHB	Ashby (ASHB)	-122.270062	37.852803	3100 Adeli...
BALB	Balboa Park (BALB)	-122.447506	37.721585	401 Geneva...
BAYF	Bay Fair (BAYF)	-122.126514	37.696924	15242 Hesp...
CAST	Castro Valley (CAST)	-122.075602	37.690746	3301 Norbr...
CIVC	Civic Center/UN Plaza (CIVC)	-122.414123	37.779732	1150 Marke...
COLS	Coliseum/Oakland Airport (COLS)	-122.196869	37.753661	7200 San L...
COLM	Colma (COLM)	-122.466233	37.684638	365 D Stre...
CONC	Concord (CONC)	-122.029095	37.973737	1451 Oakla...
DALY	Daly City (DALY)	-122.469081	37.706121	500 John D...
DBRK	Downtown Berkeley (DBRK)	-122.268133	37.870104	2160 Shatt...
DELN	El Cerrito del Norte (DELN)	-122.316794	37.925086	6400 Cutti...
DUBL	Dublin/Pleasanton (DUBL)	-121.899179	37.701687	5801 Owens...
EMBR	Embarcadero (EMBR)	-122.397020	37.792874	298 Market...
FRMT	Fremont (FRMT)	-121.976608	37.557465	2000 BART ...
FTVL	Fruitvale (FTVL)	-122.224175	37.774836	3401 East ...
GLEN	Glen Park (GLEN)	-122.433817	37.733064	2901 Diamo...
HAYW	Hayward (HAYW)	-122.087018	37.669723	699 ’B’ St...
LAFY	Lafayette (LAFY)	-122.124630	37.893176	3601 Deer ...
LAKE	Lake Merritt (LAKE)	-122.265180	37.797027	800 Madiso...
MCAR	MacArthur (MCAR)	-122.267040	37.829065	555 40th S...
MLBR	Millbrae (MLBR)	-122.386702	37.600271	200 North ...
MONT	Montgomery St. (MONT)	-122.401066	37.789405	598 Market...
NBRK	North Berkeley (NBRK)	-122.283440	37.873967	1750 Sacra...
NCON	North Concord/Martinez (NCON)	-122.024653	38.003193	3700 Port ...
OAKL	Oakland Airport (OAKL)	-122.212191	37.713238	1 Airport ...
ORIN	Orinda (ORIN)	-122.183791	37.878361	11 Camino ...
PHIL	Pleasant Hill/Contra Costa Centre (PHIL)	-122.056012	37.928468	1365 T... ...
PITT	Pittsburg/Bay Point (PITT)	-121.945154	38.018914	1700 West ...
PLZA	El Cerrito Plaza (PLZA)	-122.298904	37.902632	6699 Fairm...
POWL	Powell St. (POWL)	-122.407974	37.784471	899 Market...
RICH	Richmond (RICH)	-122.353099	37.936853	1700 Nevin...
ROCK	Rockridge (ROCK)	-122.251371	37.844702	5660 Colle...
SANL	San Leandro (SANL)	-122.160844	37.721947	1401 San L...
SBRN	San Bruno (SBRN)	-122.416287	37.637761	1151 Hunti...
SFIA	San Francisco Int’l Airport (SFIA)	-122.392409	37.615966	Internatio...
SHAY	South Hayward (SHAY)	-122.057189	37.634375	28601 Dixo...
SSAN	South San Francisco (SSAN)	-122.443960	37.664245	1333 Missi...
UCTY	Union City (UCTY)	-122.017388	37.590630	10 Union S...
WARM	Warm Springs/South Fremont (WARM)	-121.939313	37.502171	45193 Warm...
WCRK	Walnut Creek (WCRK)	-122.067527	37.905522	200 Ygnaci...
WDUB	West Dublin/Pleasanton (WDUB)	-121.928240	37.699756	6501 Golde...
WOAK	West Oakland (WOAK)	-122.295140	37.804872	1451 7th S...

Table 5: Criterion for setting levels of throughput

Throughput	Level
≤ 5	Very low
$6 \sim 15$	Low
$16 \sim 30$	Medium
$31 \sim 80$	High
> 80	Very high

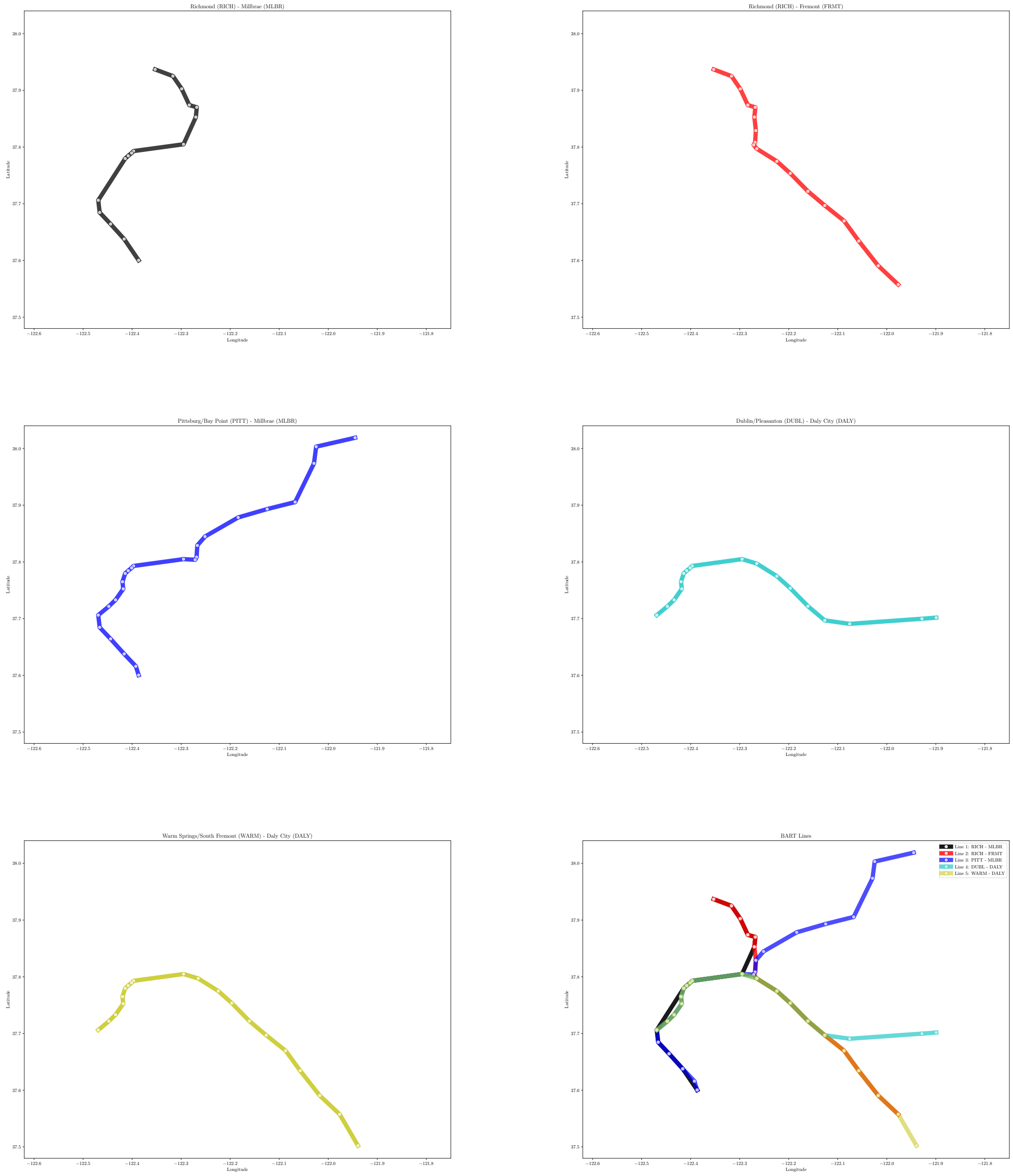
Richmond (RICH) - Millbrae (MLBR)
Richmond (RICH) - Fremont (FRMT)
Pittsburg/Bay Point (PITT) - Millbrae (MLBR)
Dublin/Pleasanton (DUBL) - Daly City (DALY)
Warm Springs/South Fremont (WARM) - Daly City (DALY)
BART Lines
Line 1: RICH - MLBR
Line 2: RICH - FRMT
Line 3: PITT - MLBR
Line 4: DUBL - DALY
Line 5: WARM - DALY
Latitude
Longitude
38.0
37.9
37.8
37.7
37.6
37.5
−122.6
−122.5
−122.4
−122.3
−122.2
−122.1
−122.0
−121.9
−121.8

Figure 1: Routes of the BART Lines

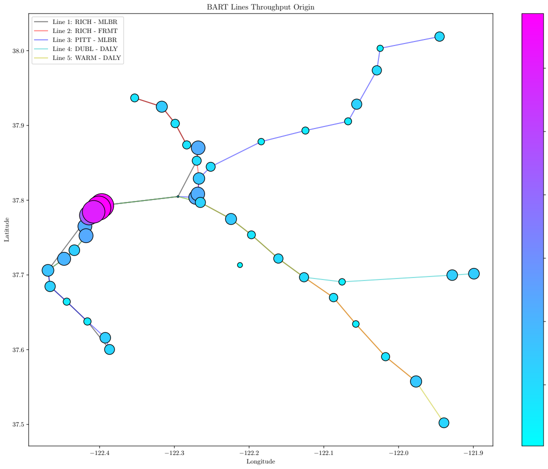

Figure 2: Average hourly throughput of each station of 2016 and 2017

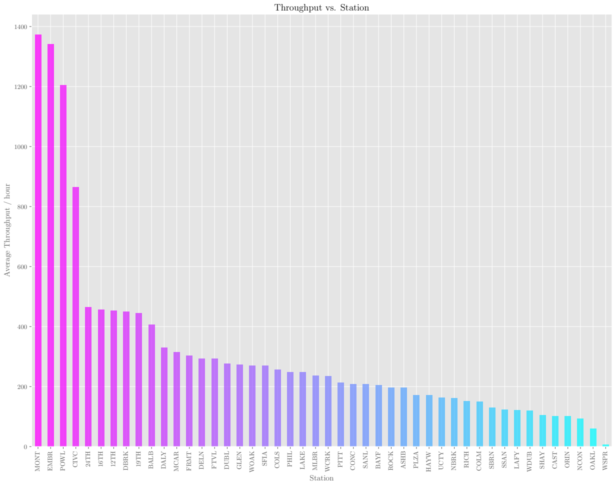

Figure 3: Barplot for average hourly throughput of each station of 2016 and 2017

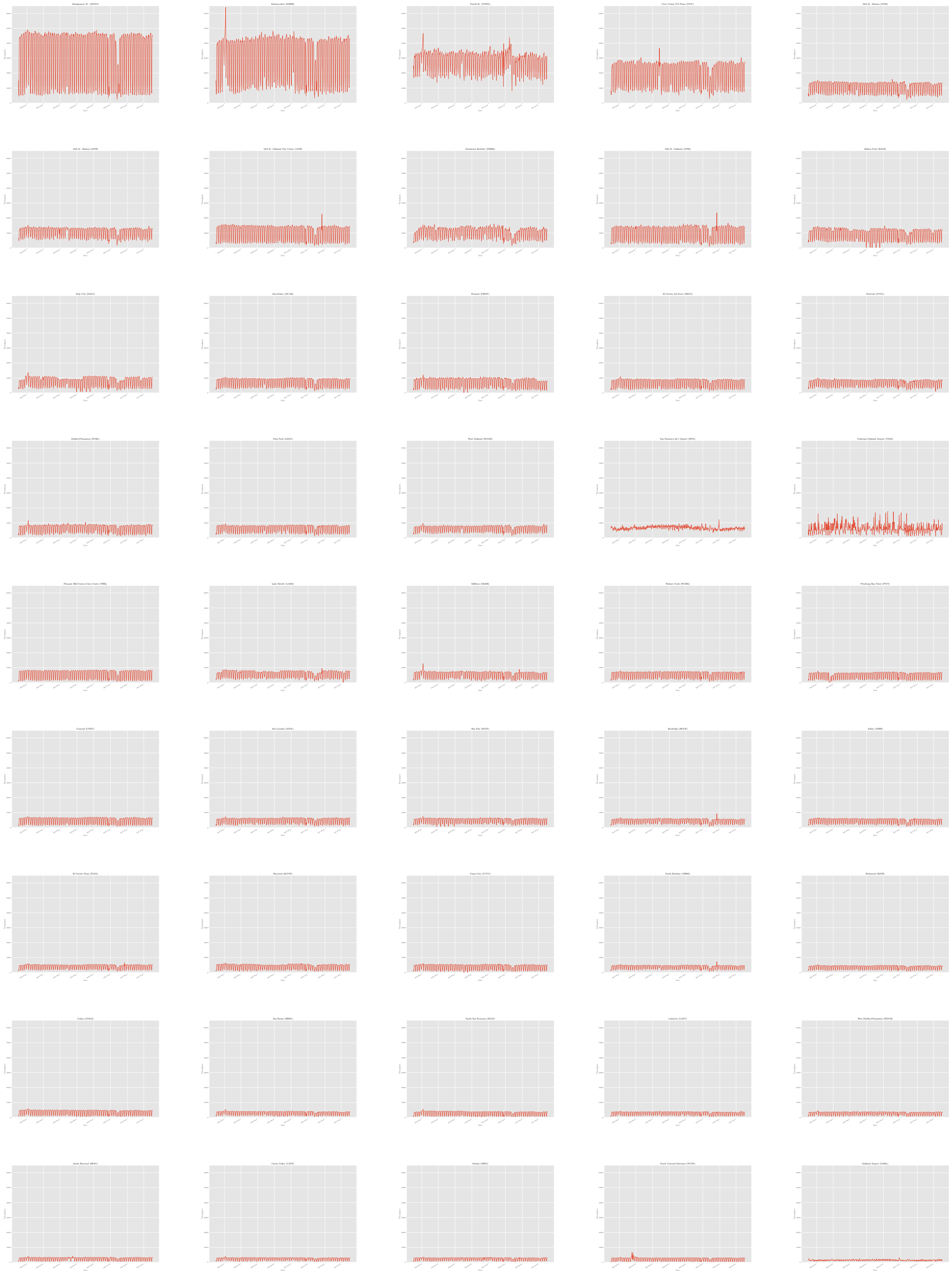

Figure 4: Throughput of each station of 2016 and 2017

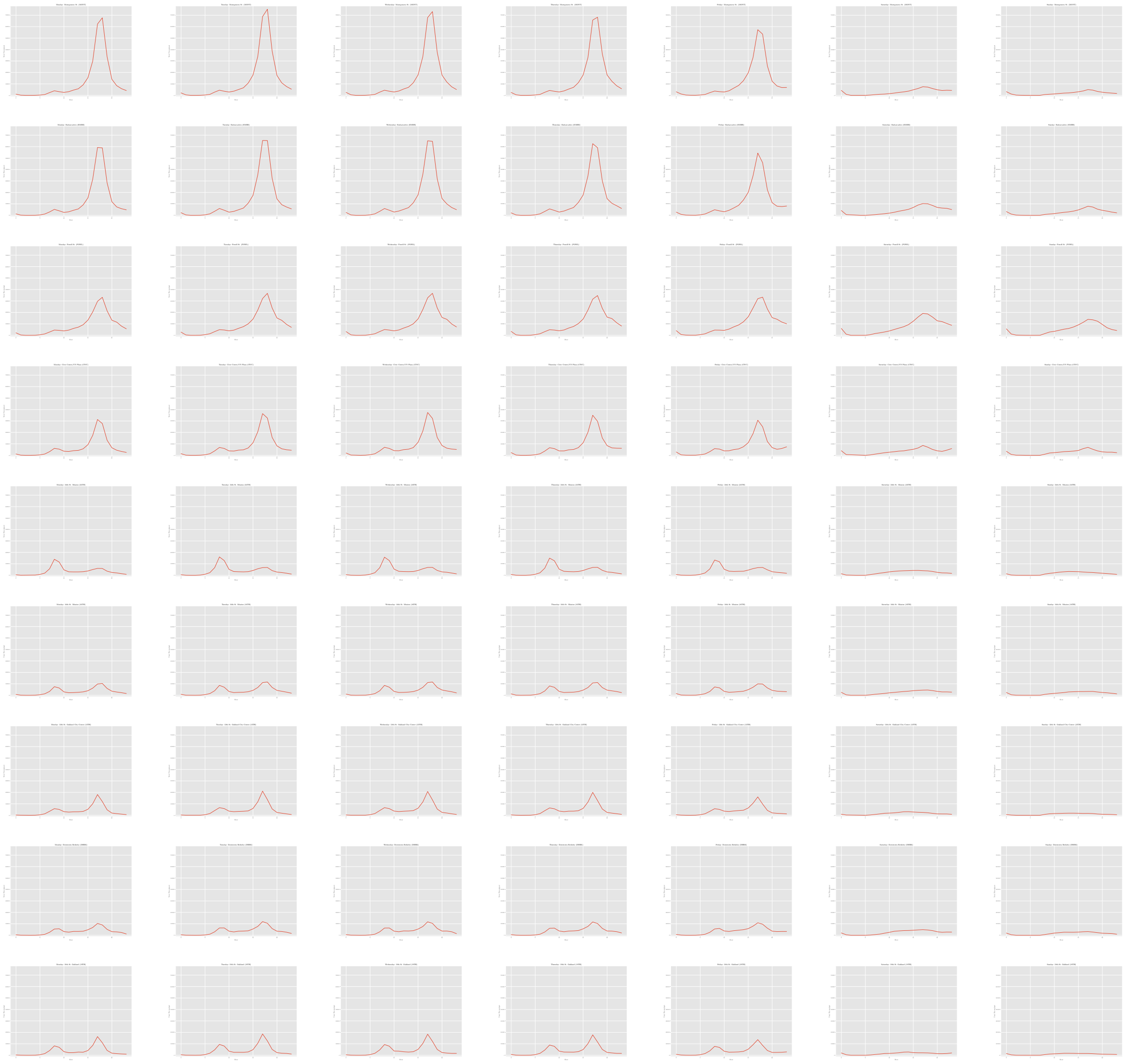

Figure 5: Total hourly throughput of a week of stations MONT, EMBR, POWL, CIVC, 24TH, 16TH, 12TH, DBRK, 19TH

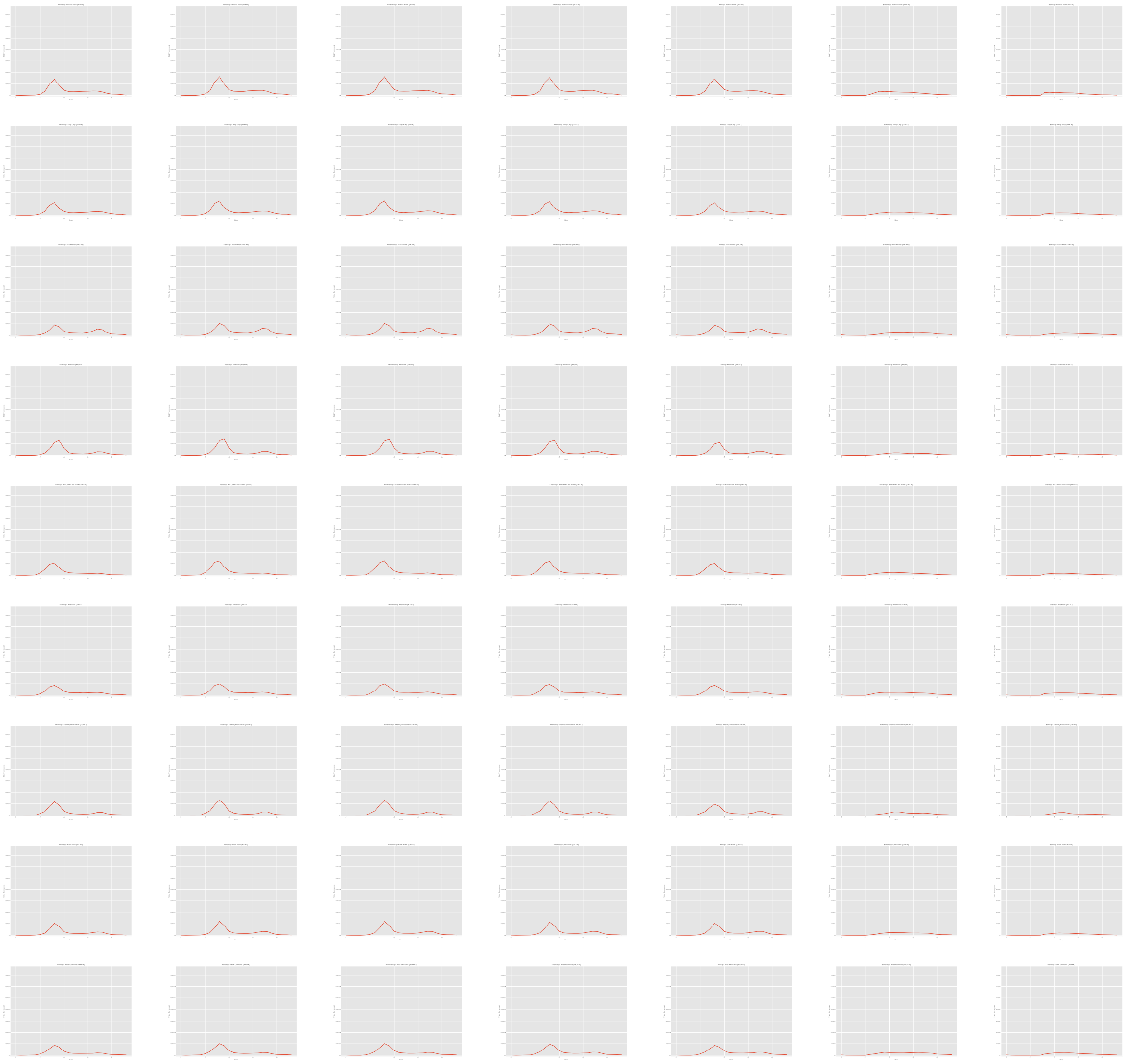

Figure 6: Total hourly throughput of a week of stations BALB, DALY, MCAR, FRMT, DELN, FTVL, DUBL, GLEN, WOAK

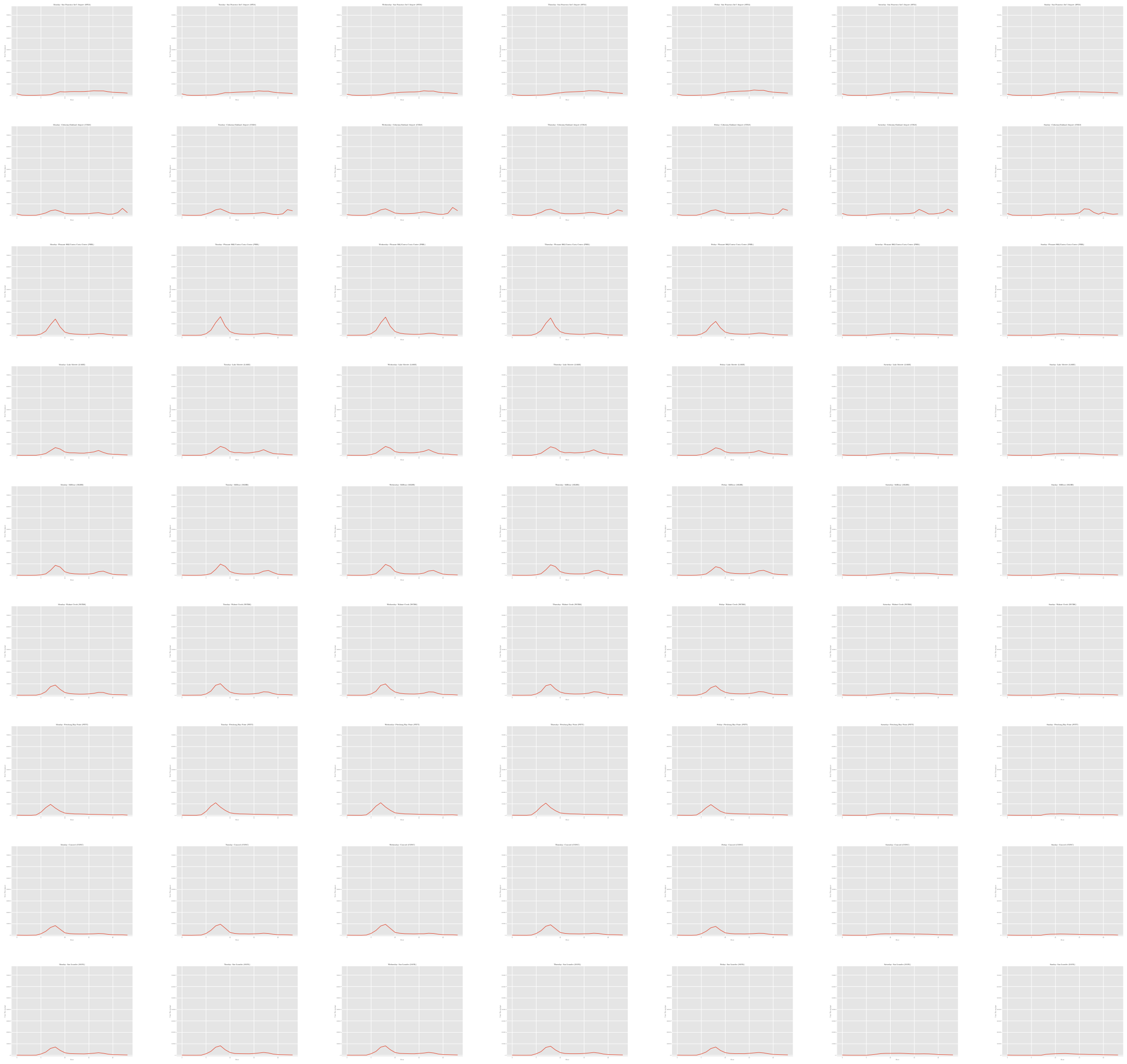

Figure 7: Total hourly throughput of a week of stations SFIA, COLS, PHIL, LAKE, MLBR, WCRK, PITT, CONC, SANL

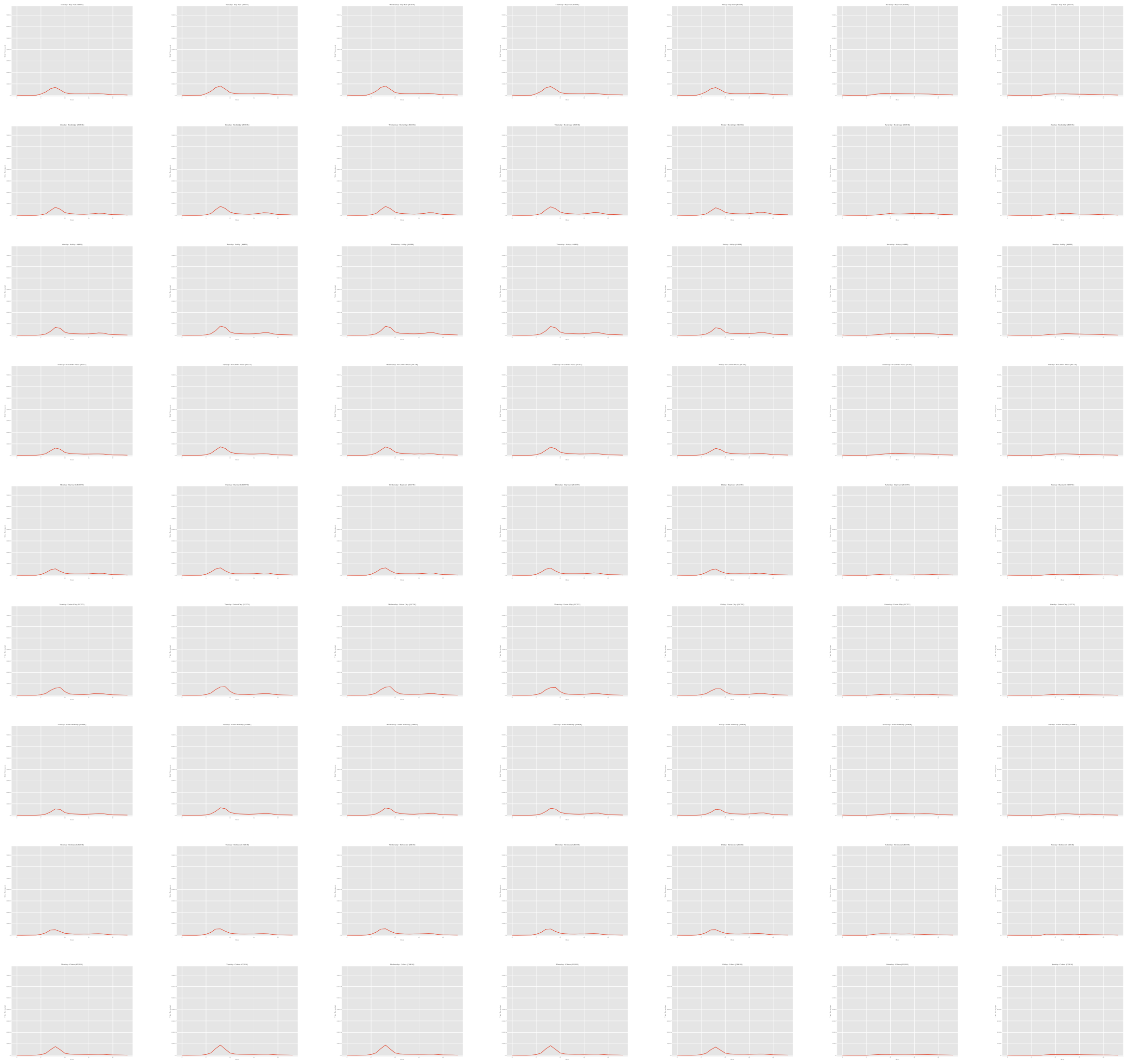

Figure 8: Total hourly throughput of a week of stations BAYF, ROCK, ASHB, PLZA, HAYW, UCTY, NBRK, RICH, COLM

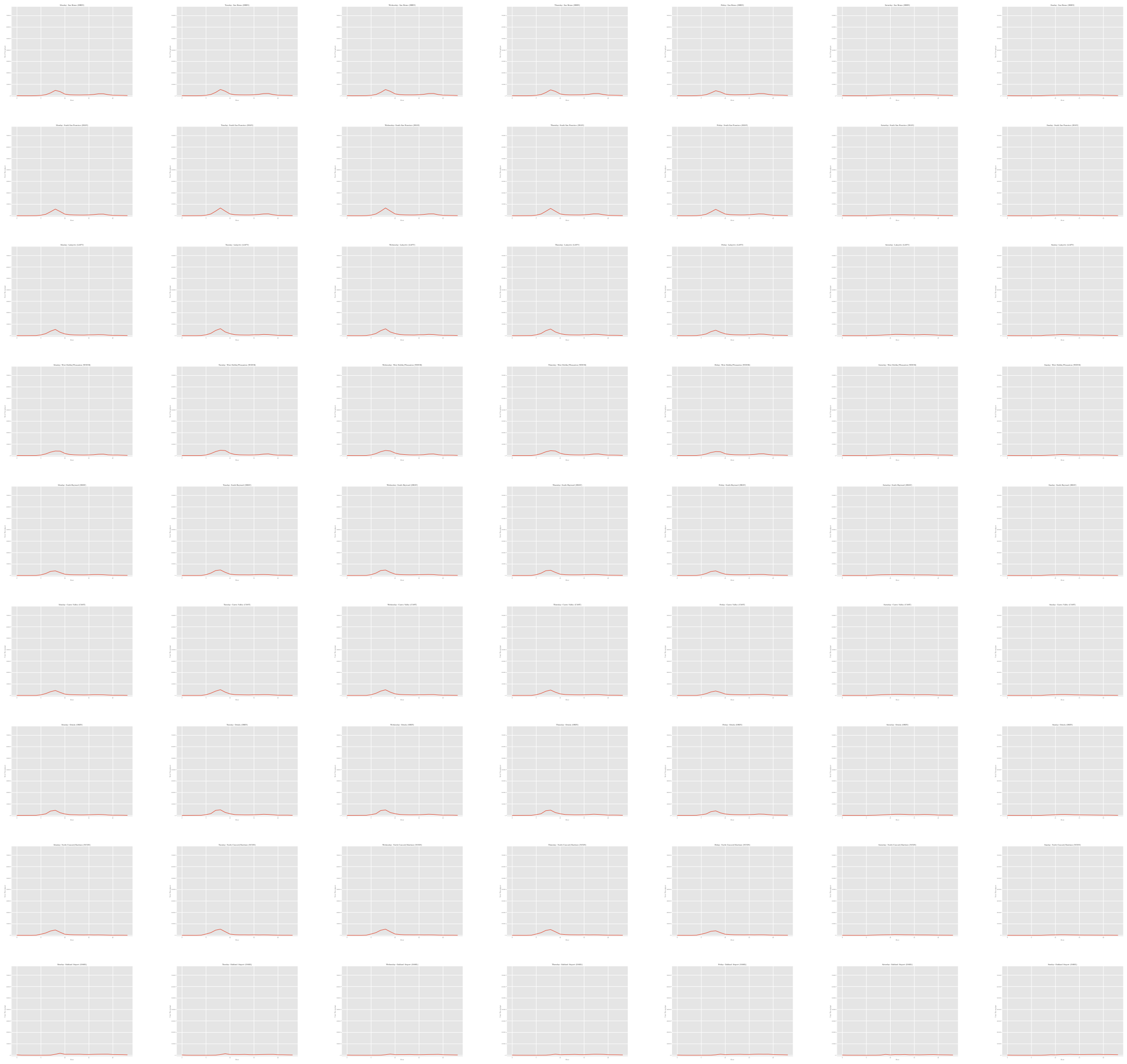

Figure 9: Total hourly throughput of a week of stations SBRN, SSAN, LAFY, WDUB, SHAY, CAST, ORIN, NCON, OAKL

the accuracy rate to 45%, and using 10% of the data as training dataset, the decision tree can reach at 77.59% accuracy rate. As the size of training dataset becomes larger, the accuracy rate goes higher, at the same time the decision tree becomes more complicated (more branches).

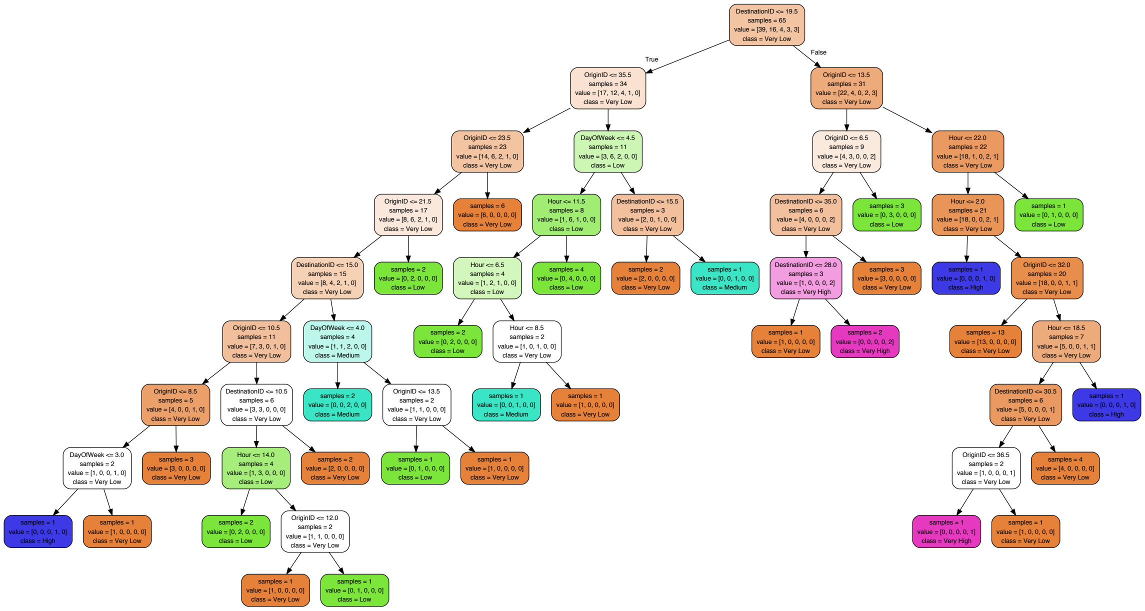
DestinationID <= 19.5 samples = 65 value = [39, 16, 4, 3, 3] class = Very Low
True
False
OriginID <= 35.5 samples = 34 value = [17, 12, 4, 1, 0] class = Very Low
OriginID <= 13.5 samples = 31 value = [22, 4, 0, 2, 3] class = Very Low
OriginID <= 23.5 samples = 23 value = [14, 6, 2, 1, 0] class = Very Low
DayOfWeek <= 4.5 samples = 11 value = [3, 6, 2, 0, 0] class = Low
OriginID <= 6.5 samples = 9 value = [4, 3, 0, 0, 2] class = Very Low
Hour <= 22.0 samples = 22 value = [18, 1, 0, 2, 1] class = Very Low
OriginID <= 21.5 samples = 17 value = [8, 6, 2, 1, 0] class = Very Low
samples = 6 value = [6, 0, 0, 0, 0] class = Very Low
Hour <= 11.5 samples = 8 value = [1, 6, 1, 0, 0] class = Low
DestinationID <= 15.5 samples = 3 value = [2, 0, 1, 0, 0] class = Very Low
DestinationID <= 35.0 samples = 6 value = [4, 0, 0, 0, 2] class = Very Low
samples = 3 value = [0, 3, 0, 0, 0] class = Low
Hour <= 2.0 samples = 21 value = [18, 0, 0, 2, 1] class = Very Low
samples = 1 value = [0, 1, 0, 0, 0] class = Low
DestinationID <= 15.0 samples = 15 value = [8, 4, 2, 1, 0] class = Very Low
samples = 2 value = [0, 2, 0, 0, 0] class = Low
Hour <= 6.5 samples = 4 value = [1, 2, 1, 0, 0] class = Low
samples = 4 value = [0, 4, 0, 0, 0] class = Low
samples = 2 value = [2, 0, 0, 0, 0] class = Very Low
samples = 1 value = [0, 0, 1, 0, 0] class = Medium
DestinationID <= 28.0 samples = 3 value = [1, 0, 0, 0, 2] class = Very High
samples = 3 value = [3, 0, 0, 0, 0] class = Very Low
samples = 1 value = [0, 0, 0, 1, 0] class = High
OriginID <= 32.0 samples = 20 value = [18, 0, 0, 1, 1] class = Very Low
OriginID <= 10.5 samples = 11 value = [7, 3, 0, 1, 0] class = Very Low
DayOfWeek <= 4.0 samples = 4 value = [1, 1, 2, 0, 0] class = Medium
samples = 2 value = [0, 2, 0, 0, 0] class = Low
Hour <= 8.5 samples = 2 value = [1, 0, 1, 0, 0] class = Very Low
samples = 1 value = [1, 0, 0, 0, 0] class = Very Low
samples = 2 value = [0, 0, 0, 0, 2] class = Very High
samples = 13 value = [13, 0, 0, 0, 0] class = Very Low
Hour <= 18.5 samples = 7 value = [5, 0, 0, 1, 1] class = Very Low
OriginID <= 8.5 samples = 5 value = [4, 0, 0, 1, 0] class = Very Low
DestinationID <= 10.5 samples = 6 value = [3, 3, 0, 0, 0] class = Very Low
samples = 2 value = [0, 0, 2, 0, 0] class = Medium
OriginID <= 13.5 samples = 2 value = [1, 1, 0, 0, 0] class = Very Low
samples = 1 value = [0, 0, 1, 0, 0] class = Medium
samples = 1 value = [1, 0, 0, 0, 0] class = Very Low
DestinationID <= 30.5 samples = 6 value = [5, 0, 0, 0, 1] class = Very Low
samples = 1 value = [0, 0, 0, 1, 0] class = High
DayOfWeek <= 3.0 samples = 2 value = [1, 0, 0, 1, 0] class = Very Low
samples = 3 value = [3, 0, 0, 0, 0] class = Very Low
Hour <= 14.0 samples = 4 value = [1, 3, 0, 0, 0] class = Low
samples = 2 value = [2, 0, 0, 0, 0] class = Very Low
samples = 1 value = [0, 1, 0, 0, 0] class = Low
samples = 1 value = [1, 0, 0, 0, 0] class = Very Low
OriginID <= 36.5 samples = 2 value = [1, 0, 0, 0, 1] class = Very Low
samples = 4 value = [4, 0, 0, 0, 0] class = Very Low
samples = 1 value = [0, 0, 0, 1, 0] class = High
samples = 1 value = [1, 0, 0, 0, 0] class = Very Low
samples = 2 value = [0, 2, 0, 0, 0] class = Low
OriginID <= 12.0 samples = 2 value = [1, 1, 0, 0, 0] class = Very Low
samples = 1 value = [0, 0, 0, 0, 1] class = Very High
samples = 1 value = [1, 0, 0, 0, 0] class = Very Low
samples = 1 value = [1, 0, 0, 0, 0] class = Very Low
samples = 1 value = [0, 1, 0, 0, 0] class = Low

Figure 10: The decision tree obtained by using 0.02% of the data

5 Conclusions & Future Work

5.1 Conclusions

Now we are able to answer some of the quetions related to the BART as follows:

1. The busiest station is Montgomery St.
2. The least popular BART route would be Richmond – Fremont.
3. The best time to go to downtown San Francisco from, say Dublin/Pleasanton Station, if you want to find a seat is around 11:00 AM.
4. The busiest day of the week is Tuesday.

5.2 Future Work

Due to the time limitation, we are unable to build visualize the route of BART lines on a map. So we'd like to realize that and if possible, we'd like to use D3.js to build a interactive interface which enables us to understand the data better by the animation.

References

[1] Kaggle, *Bay Area Rapid Transit Ridership Datasets*, https://www.kaggle.com/saulfuh/bart-ridership/data, Obtained on 04/30/2018.

[2] BART Official Website, *BART Schedules*, https://www.bart.gov/schedules/bystation, Obtained on 04/30/2018.

[3] Jonathan Bouchet, *BART Transit System*, https://www.kaggle.com/jonathanbouchet/bart-transit-system, Obtained on 04/30/2018.

[4] Xiuhua Han, et al., *Research on Data Mining of Public Transit IC Card and Application*, 2010 International Conference on Intelligent Computation Technology and Automation, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=5522901tag=1, Obtained on 04/30/2018.

[5] Roy Ka-Wei Lee, et al., *Time-Series Data Mining in Transportation: A Case Study on Singapore Public Train Commuter Travel Patterns*, IACSIT International Journal of Engineering and Technology, Vol. 6, No. 5, October 2014, https://pdfs.semanticscholar.org/ffc1/399342b36e178f511af57723f38e37ac7793.pdf, Obtained on 04/30/2018.

6 Python Program

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib.backends.backend_pdf import PdfPages
# from IPython.core.interactiveshell import InteractiveShell
# InteractiveShell.ast_node_interactivity = 'all'

plt.rc('text', usetex=True)
plt.rc('font', family='serif')

__author__ = 'Libao Jin'
__date__ = '04/24/2018'
__email__ = 'ljin1@uwyo.edu'
__copyright__ = 'Copyright (c) 2018 Libao Jin'


def bart_preprocess(bart):
    # Convert string to datetime, split datetime into date and time, and get day of week
    bart['DateTime'] = pd.to_datetime(bart.DateTime)
    bart['Date'] = bart['DateTime'].dt.date
    bart['Time'] = bart['DateTime'].dt.time
    bart['DayOfWeek'] = bart['DateTime'].dt.weekday_name
    # bart.drop(columns='DateTime')
    return bart


def stat_preprocess(stat):
    # Split location into longitude and latitude for visualization later on
    loc = stat.Location.str.split(',', expand=True)
    loc = [pd.to_numeric(loc[i]) for i in loc.columns]
    stat['Longitude'], stat['Latitude'] = loc[0], loc[1]
    columns = ['Abbreviation', 'Name', 'Longitude', 'Latitude', 'Description']
    stat = stat[columns]
    return stat


def generate_bart_routes(bart_lines, stat):
    bart_routes = []

    for line in bart_lines:

        abbr = []
        name = []
        fullname = []
        location = []

        for station in line:
            tmp = stat[stat['Name'].str.contains(station)]
            abbr.append(tmp.iloc[0]['Abbreviation'])
            name.append(station)
            fullname.append(tmp.iloc[0]['Name'])
            location.append([tmp.iloc[0]['Longitude'],
                             tmp.iloc[0]['Latitude']])

        bart_routes.append({
            'abbr': abbr,
            'name': name,
            'fullname': fullname,
            'location': location
        })
    return bart_routes


def visualize_bart_routes(bart_routes, filename='bart_routes.pdf'):
    color = list('krbcy')
    alpha = [0.9, 0.8, 0.7, 0.6, 0.5]
    x_min, x_max, y_min, y_max = -122.62, -121.75, 37.48, 38.04

    with PdfPages(filename) as pdf:
        plt.style.use('default')
        plt.rc('text', usetex=True)
        plt.rc('font', family='serif')
        for i in range(len(bart_routes)):
            fig = plt.figure(figsize=(16, 12))
            loc = np.array(bart_routes[i]['location'])
            # x, y = loc[:, 0], loc[:, 1]
            # plt.plot(x, y, '-o', c=color[i], ms=6, lw=4, alpha=0.5)
            plt.plot(
                loc[:, 0],
                loc[:, 1],
                '-o',
                c=color[i],
                mfc='w',
                ms=8,
                lw=9,
                alpha=0.75
            )
            plt.xlim([x_min, x_max])
            plt.ylim([y_min, y_max])
            plt.xlabel('Longitude')
            plt.ylabel('Latitude')
            title = '{} - {}'.format(bart_routes[i]['fullname'][0],
                                     bart_routes[i]['fullname'][-1])
            plt.title(title)
            plt.show(block=False)
            pdf.savefig(fig)

        fig = plt.figure(figsize=(16, 12))
        for i in range(len(bart_routes)):
            loc = np.array(bart_routes[i]['location'])
            label = 'Line {:d}: {} - {}'.format(i + 1,
                                                bart_routes[i]['abbr'][0],
                                                bart_routes[i]['abbr'][-1])
            # x, y = loc[:, 0], loc[:, 1]
            # plt.plot(x, y, '-o', c=color[i], ms=6, lw=4, alpha=0.5)
            plt.plot(
                loc[:, 0],
                loc[:, 1],
                '-o',
                c=color[i],
                mfc='w',
                ms=8,
                lw=9,
                alpha=alpha[i],
                label=label
            )
        plt.legend()
        plt.xlim([x_min, x_max])
        plt.ylim([y_min, y_max])
        plt.xlabel('Longitude')
        plt.ylabel('Latitude')
        plt.title('BART Lines')
        plt.show(block=False)
        # plt.axis('equal')
        pdf.savefig(fig)


def visualize_throughput(bart_routes, bart_aggregate, column_name, filename='bart_throughput.pdf'):
    with PdfPages(filename) as pdf:
        color = list('krbcy')
        bart_aggregate = bart_aggregate.set_index('Station')
        x, y = bart_aggregate['Longitude'], bart_aggregate['Latitude']
        n = len(bart_aggregate['Longitude'])
        cmap = mpl.cm.cool
        cs = getattr(bart_aggregate, column_name)
        x_min, x_max, y_min, y_max = -122.62, -121.75, 37.48, 38.04
        fig = plt.figure(figsize=(16, 12))
        plt.style.use('default')
        plt.rc('text', usetex=True)
        plt.rc('font', family='serif')
        for i in range(len(bart_routes)):
            loc = np.array(bart_routes[i]['location'])
            label = 'Line {:d}: {} - {}'.format(i + 1,
                                                bart_routes[i]['abbr'][0],
                                                bart_routes[i]['abbr'][-1])
            # x, y = loc[:, 0], loc[:, 1]
            # plt.plot(x, y, '-o', c=color[i], ms=6, lw=4, alpha=0.5)
            plt.plot(
                loc[:, 0],
                loc[:, 1],
                '-',
                c=color[i],
                # mfc='w',
                # ms=8,
                # lw=6,
                alpha=0.5,
                label=label,
                zorder=1
            )
        plt.legend()

        plt.scatter(
            x,
            y,
            c=cs,
            s=cs,
            marker='o',
            edgecolors='k',
            cmap=cmap,
            alpha=1,
            zorder=2
        )

        plt.colorbar()
        plt.xlim([x_min, x_max])
        plt.ylim([y_min, y_max])
        plt.xlabel('Longitude')
        plt.ylabel('Latitude')
        plt.axis('equal')
        plt.title('BART Lines ' + cs.name)
        plt.show(block=False)
        pdf.savefig(fig)

        fig = plt.figure(figsize=(16, 12))
        plt.style.use('ggplot')
        plt.rc('text', usetex=True)
        plt.rc('font', family='serif')
        bart_aggregate.sort_values(column_name, inplace=True, ascending=False)
        cs = getattr(bart_aggregate, column_name)
        colors = cmap(np.linspace(1, 0, n))
        cs.plot(kind='bar', color=colors, alpha=0.75)
        # print(list(cs.index))
        plt.ylabel('Average Throughput / hour')
        plt.title('Throughput vs. Station')
        plt.show(block=False)
        pdf.savefig(fig)


def bart_aggregate_throughput(bart, filename):
    number_of_days = len(bart['DateTime'].dt.date.unique())
    number_of_hours = len(bart['DateTime'].dt.time.unique())
    bart_grouped = bart['Throughput'].groupby(bart['Origin']).sum().to_frame()
    bart_grouped['Destination'] = bart['Throughput'].groupby(bart['Destination']).sum()
    bart_grouped.index.names = ['Station']
    bart_grouped.columns = ['Throughput Origin', 'Throughput Destination']
    bart_grouped['Throughput All'] = bart_grouped['Throughput Origin'] + bart_grouped['Throughput Destination']
    bart_grouped.set_index(stat['Abbreviation'])
    bart_grouped.reset_index(level=0, inplace=True)
    bart_grouped['Longitude'] = stat['Longitude']
    bart_grouped['Latitude'] = stat['Latitude']
    bart_grouped[['Throughput Origin', 'Throughput Destination', 'Throughput All']] = \
        bart_grouped[['Throughput Origin', 'Throughput Destination', 'Throughput All']] / (number_of_days * number_of_hours)
    visualize_throughput(bart_routes, bart_grouped, 'Throughput Origin', filename)
    # visualize_throughput(bart_routes, bart_grouped, 'Throughput Destination', filename)
    # visualize_throughput(bart_routes, bart_grouped, 'Throughput All', filename)


def visualize_bart(bart, origins, stat, class_type, group_by, plot_option, filename='bart_overview.pdf'):

    plt.style.use('ggplot')
    plt.rc('text', usetex=True)
    plt.rc('font', family='serif')

    with PdfPages(filename) as pdf:
        for stops in origins:
            if len(stops) == 1:
                data = bart[bart['Origin'] == stops[0]]
            elif len(stops) == 2:
                origin, dest = stops
                data = bart[(bart['Origin'] == origin) & (bart['Destination'] == dest)]
            # Plot the throughput with respect to time (hour) each week/month
            if plot_option == 1 or group_by == 'hour':
                k = len(getattr(data['DateTime'].dt, class_type[0]).unique())
                # n = 2                                 # number of columns
                # m = math.ceil(k / n)  # number of rows
                # plt.figure(figsize=(8 * n, 6 * m))
                for i in range(k):
                    fig = plt.figure(figsize=(8, 6))
                    if class_type[0] == 'weekday':
                        j = i
                    else:
                        j = i + 1
                    grouped = data[getattr(data['DateTime'].dt, class_type[0]).values == j].groupby(
                        getattr(data['DateTime'].dt, group_by)).sum()
                    grouped.sort_index(inplace=True)
                    plt.plot(grouped['Throughput'])
                    if len(stops) == 1:
                        tmp = stat[stat['Abbreviation'] == data.iloc[0]['Origin']]
                        title = '{}: {}'.format(class_type[1][i],
                                                tmp.iloc[0]['Name'])
                    else:
                        tmp_1 = stat[stat['Abbreviation'] == data.iloc[0]['Origin']]
                        tmp_2 = stat[stat['Abbreviation'] == data.iloc[0]['Destination']]
                        title = '{}: {} - {}'.format(class_type[1][i],
                                                     tmp_1.iloc[0]['Name'],
                                                     tmp_2.iloc[0]['Name'])
                    plt.title(title)
                    plt.xlabel(group_by.title())
                    ax = plt.gca()
                    # fig = plt.gcf()
                    if group_by == 'date':
                        xfmt = mpl.dates.DateFormatter('%Y-%m-%d')
                        ax.xaxis.set_major_formatter(xfmt)
                        # plt.xticks(rotation=90)
                        fig.autofmt_xdate()
                    plt.ylabel('Total Throughput')
                    plt.ylim([-10000, 780000])
                    # plt.xticks(grouped.index, list(np.arange(24)))
                    plt.show(block=False)
                    pdf.savefig(fig)

            # Plot the throughput with respect to date
            elif plot_option == 2:
                grouped = data.groupby(getattr(data['DateTime'].dt, group_by)).sum()
                grouped.sort_index(inplace=True)
                fig = plt.figure(figsize=(8, 6))
                plt.plot(grouped['Throughput'])
                if len(stops) == 1:
                    tmp = stat[stat['Abbreviation'] == data.iloc[0]['Origin']]
                    title = '{}'.format(tmp.iloc[0]['Name'])
                    # title = class_type[1][i] + ': {}'.format(data.iloc[0]['Origin'])
                else:
                    tmp_1 = stat[stat['Abbreviation'] == data.iloc[0]['Origin']]
                    tmp_2 = stat[stat['Abbreviation'] == data.iloc[0]['Destination']]
                    title = '{} - {}'.format(tmp_1.iloc[0]['Name'],
                                             tmp_2.iloc[0]['Name'])
                    # title = ': {} - {}'.format(data.iloc[0]['Origin'], data.iloc[0]['Destination'])
                # plt.legend()
                plt.xlabel(group_by.title())
                plt.ylabel('Throughput')
                plt.xticks(rotation=90)
                ax = plt.gca()
                # fig = plt.gcf()
                xfmt = mpl.dates.DateFormatter('%Y-%m-%d')
                ax.xaxis.set_major_formatter(xfmt)
                # plt.xticks(rotation=90)
                fig.autofmt_xdate()
                plt.title(title)
                plt.ylim([0, 65000])
                plt.show(block=False)
                pdf.savefig(fig)


if __name__ == '__main__':

    dest_folder = './output'
    plt.rc('text', usetex=True)
    plt.rc('font', family='serif')
    # Load data
    # Data obtained from https://www.kaggle.com/saulfuh/bart-ridership
    date_hour_2016 = '../data/bart-ridership/date-hour-soo-dest-2016.csv'
    date_hour_2017 = '../data/bart-ridership/date-hour-soo-dest-2017.csv'
    stat_info = '../data/bart-ridership/station_info.csv'
    # bart_16 = pd.read_csv(date_hour_2016)
    # bart_17 = pd.read_csv(date_hour_2017)
    # stat = pd.read_csv(stat_info)

    # Data preprocessing
    bart_16 = bart_preprocess(pd.read_csv(date_hour_2016))
    bart_17 = bart_preprocess(pd.read_csv(date_hour_2017))
    bart = pd.concat([bart_16, bart_17], ignore_index=True)
    stat = stat_preprocess(pd.read_csv(stat_info))

    # Visualize the routes according to the BART official website

    line_1 = [
        'Richmond',
        'El Cerrito del Norte',
        'El Cerrito Plaza',
        'North Berkeley',
        'Downtown Berkeley',
        'Ashby',
        'West Oakland',
        'Embarcadero',
        'Montgomery St.',
        'Powell St.',
        'Civic Center/UN Plaza',
        'Daly City',
        'Colma',
        'South San Francisco',
        'San Bruno',
        'Millbrae'
    ]

    line_2 = [
        'Richmond',
        'El Cerrito del Norte',
        'El Cerrito Plaza',
        'North Berkeley',
        'Downtown Berkeley',
        'Ashby',
        'MacArthur',
        '19th St. Oakland',
        '12th St. Oakland City Center',
        'Lake Merritt',
        'Fruitvale',
        'Coliseum/Oakland Airport',
        'San Leandro',
        'Bay Fair',
        'Hayward',
        'South Hayward',
        'Union City',
        'Fremont'
    ]

    line_3 = [
        'Pittsburg/Bay Point',
        'North Concord/Martinez',
        'Concord',
        'Walnut Creek',
        'Lafayette',
        'Orinda',
        'Rockridge',
        'MacArthur',
        '19th St. Oakland',
        '12th St. Oakland City Center',
        'West Oakland',
        'Embarcadero',
        'Montgomery St.',
        'Powell St.',
        'Civic Center/UN Plaza',
        '16th St. Mission',
        '24th St. Mission',
        'Glen Park',
        'Balboa Park',
        'Daly City',
        'Colma',
        'South San Francisco',
        'San Bruno',
        "San Francisco Int'l Airport",
        'Millbrae'
    ]

    line_4 = [
        'Dublin/Pleasanton',
        'West Dublin/Pleasanton',
        'Castro Valley',
        'Bay Fair',
        'San Leandro',
        'Coliseum/Oakland Airport',
        'Fruitvale',
        'Lake Merritt',
        'West Oakland',
        'Embarcadero',
        'Montgomery St.',
        'Powell St.',
        'Civic Center/UN Plaza',
        '16th St. Mission',
        '24th St. Mission',
        'Glen Park',
        'Balboa Park',
        'Daly City'
    ]

    line_5 = [
        'Warm Springs/South Fremont',
        'Fremont',
        'Union City',
        'South Hayward',
        'Hayward',
        'Bay Fair',
        'San Leandro',
        'Coliseum/Oakland Airport',
        'Fruitvale',
        'Lake Merritt',
        'West Oakland',
        'Embarcadero',
        'Montgomery St.',
        'Powell St.',
        'Civic Center/UN Plaza',
        '16th St. Mission',
        '24th St. Mission',
        'Glen Park',
        'Balboa Park',
        'Daly City'
    ]

    bart_lines = [
        line_1,
        line_2,
        line_3,
        line_4,
        line_5
    ]

    bart_routes = generate_bart_routes(bart_lines, stat)
    visualize_bart_routes(bart_routes, '{}/bart_routes.pdf'.format(dest_folder))

    # bart_aggregate_throughput(bart_16)
    # bart_aggregate_throughput(bart_17)
```

```python
    bart_aggregate_throughput(bart, '{}/bart_throughput.pdf'.format(dest_folder))

    by_month = [
        'month',
        ['January', 'Febrary', 'March', 'April', 'May', 'June', 'July',
         'August', 'September', 'October', 'November', 'December']
    ]

    by_weekday = [
        'weekday',
        ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
         'Friday', 'Saturday', 'Sunday']
    ]

    origins_names = [
        'MONT', 'EMBR', 'POWL', 'CIVC', '24TH', '16TH', '12TH',
        'DBRK', '19TH', 'BALB', 'DALY', 'MCAR', 'FRMT', 'DELN',
        'FTVL', 'DUBL', 'GLEN', 'WOAK', 'SFIA', 'COLS', 'PHIL',
        'LAKE', 'MLBR', 'WCRK', 'PITT', 'CONC', 'SANL', 'BAYF',
        'ROCK', 'ASHB', 'PLZA', 'HAYW', 'UCTY', 'NBRK', 'RICH',
        'COLM', 'SBRN', 'SSAN', 'LAFY', 'WDUB', 'SHAY', 'CAST',
        'ORIN', 'NCON', 'OAKL'
    ]

    origins = [[i] for i in origins_names]
    visualize_bart(bart, origins, stat, by_month, 'date', 2, '{}/bart_overview_1.pdf'.format(dest_folder))
    visualize_bart(bart, origins, stat, by_weekday, 'hour', 2, '{}/bart_overview_2.pdf'.format(dest_folder))
    # origins2 = [[origins_names[i], origins_names[i + 1]] for i in range(len(origins_names) - 1)]
    # visualize_bart(bart, origins2, stat, by_month, 'hour', 1, '{}/bart_overview_3.pdf'.format(dest_folder))
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

__author__ = 'Libao Jin'
__date__ = '05/02/2018'
__email__ = 'ljin1@uwyo.edu'


import pandas as pd
# import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
from sklearn import tree
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.externals.six import StringIO
import pydot


class BartClassifier(object):

    def __init__(self):
        plt.style.use('ggplot')
        plt.rc('text', usetex=True)
        plt.rc('font', family='serif')

    def load_data(self, filenames):
        '''Load datasets in batch'''
        datasets = []
        for filename in filenames:
            data = pd.read_csv(filename)
            datasets.append(data)
        print('Data loaded.')
        return datasets

    def data_preparation(self, bart, stat):
        loc = stat['Location'].str.split(',', expand=True)
        loc = [pd.to_numeric(loc[i]) for i in loc.columns]
        stat['Longitude'], stat['Latitude'] = loc[0], loc[1]
        stat_tmp = stat.copy()
        stat_tmp.index.names = ['Stat_ID']
        stat_tmp.reset_index(level=0, inplace=True)
        stat_id = stat_tmp.set_index('Abbreviation')['Stat_ID'].dropna()
        stat_lon = stat_tmp.set_index('Abbreviation')['Longitude'].dropna()
        stat_lat = stat_tmp.set_index('Abbreviation')['Latitude'].dropna()
        bart = bart.drop(bart[(bart['Origin'] == 'WSPR') | (bart['Destination'] == 'WSPR')].index)
        bart['DateTime'] = pd.to_datetime(bart['DateTime'])
        bart['DayOfWeek'] = bart['DateTime'].dt.weekday
        bart['Month'] = bart['DateTime'].dt.month
        bart['Day'] = bart['DateTime'].dt.day
        bart['Hour'] = bart['DateTime'].dt.hour
        print('First session done.')
        bart['OriginID'] = bart['Origin'].replace(stat_id)
        bart['OriginLongitude'] = bart['Origin'].replace(stat_lon)
        bart['OriginLatitude'] = bart['Origin'].replace(stat_lat)
        print('Second session done.')
        bart['DestinationID'] = bart['Destination'].replace(stat_id)
        bart['DestinationLongitude'] = bart['Destination'].replace(stat_lon)
        bart['DestinationLatitude'] = bart['Destination'].replace(stat_lat)
        bart['ThroughputLevel'] = bart['Throughput'].apply(self.throughput_level)
        print('Data prepared.')
        return (stat, bart)

    def throughput_level(self, throughput):
        if throughput <= 5:
            level = 0
        elif throughput <= 15:
            level = 1
        elif throughput <= 30:
            level = 2
        elif throughput <= 80:
            level = 3
        else:
            level = 4
        return level

    def generate_train_test(self, data, feature_keys, target_keys):
        train_features, test_features, train_labels, test_labels = train_test_split(
            data[feature_keys],
            data[target_keys],
            test_size=0.99998
        )
        return (train_features, test_features, train_labels, test_labels)

    def decision_tree(self, data, feature_keys, target_keys, filename='./output/bart_dt.pdf'):
        train_features, test_features, train_labels, test_labels = self.generate_train_test(
            data, feature_keys, target_keys)
        clf = tree.DecisionTreeClassifier()
        clf.fit(train_features, train_labels)
        test_labels_predict = clf.predict(test_features)
        hit_rate = sum([1 for i in range(len(test_labels)) if test_labels_predict[i] ==
                        test_labels.values[i]]) / len(test_labels) * 100
        print('Decision Tree: Prediction hit/accuracy rate: {:.2f}%'.format(hit_rate))
        self.visualize(clf, feature_keys, target_keys, filename)
        train_data, test_data = pd.DataFrame(), pd.DataFrame()
        train_data[feature_keys], train_data[target_keys], test_data[feature_keys], test_data[target_keys] = \
            train_features, train_labels, test_features, test_labels
        test_data = test_data.reset_index(drop=True)
        test_labels_predict = pd.DataFrame(data={'predict': test_labels_predict})
        test_data['predict'] = test_labels_predict
        train_data.to_csv('../data/training_data.csv', encoding='utf-8', index=False)
        test_data.to_csv('../data/test_data.csv', encoding='utf-8', index=False)
        return (test_features, test_labels, test_labels_predict, hit_rate)

    def visualize(self, clf, feature_keys, target_keys, filename):
        feature_names = feature_keys
        target_names = ['Very Low', 'Low', 'Medium', 'High', 'Very High']
        dot_data = StringIO()
        tree.export_graphviz(
            clf,
            out_file=dot_data,
            feature_names=feature_names,
            class_names=target_names,
            filled=True,
            rounded=True,
            impurity=False
        )
        graph = pydot.graph_from_dot_data(dot_data.getvalue())
        graph[0].write_pdf(filename)

    def linear_regression(self, data, feature_keys, target_keys, filename='./output/bart_lr.pdf'):
        lm = linear_model.LinearRegression()
        X = data[feature_keys]
        y = data[target_keys]
        lm.fit(X, y)
        predictions = lm.predict(X)
        print('Linear Regression: Coefficients: {}\nIntercept: {}\nScore: {}'.format(lm.coef_,
                                                                                     lm.intercept_,
                                                                                     lm.score(X, y)))

        with PdfPages(filename) as pdf:
            fig, ax = plt.subplots()
            ax.scatter(y, predictions, edgecolors=(0, 0, 0))
            ax.plot([min(predictions), max(predictions)], [min(predictions), max(predictions)], 'b--', lw=2)
            ax.set_xlabel('Measured')
            ax.set_ylabel('Predicted')
            plt.show(block=False)
            pdf.savefig(fig)

    def run(self):
        # filenames = [
        #     '../data/bart-ridership/date-hour-soo-dest-2016.csv',
        #     '../data/bart-ridership/date-hour-soo-dest-2017.csv',
        #     '../data/bart-ridership/station_info.csv'
        # ]

        # datasets = self.load_data(filenames)
        # bart = pd.concat(datasets[0:2], ignore_index=True)
        # stat = datasets[2]

        filenames = [
            # '../data/bart-ridership/date-hour-soo-dest-2016.csv',
            '../data/bart-ridership/date-hour-soo-dest-2017.csv',
            '../data/bart-ridership/station_info.csv'
        ]
        datasets = self.load_data(filenames)
        bart = datasets[0]
        stat = datasets[1]

        stat, bart = self.data_preparation(bart, stat)
        print(stat.head())
        print(bart.head())

        # feature_keys = [
        #     'OriginID', 'OriginLongitude', 'OriginLatitude',
        #     'DestinationID', 'DestinationLongitude', 'DestinationLatitude',
        #     'Month', 'Day', 'Hour', 'DayOfWeek'
        # ]

        feature_keys = [
            'OriginID',
            'DestinationID',
            'Hour', 'DayOfWeek'
        ]

        target_keys = ['ThroughputLevel']

        self.decision_tree(bart, feature_keys, target_keys)
        self.linear_regression(bart, feature_keys, target_keys)


if __name__ == '__main__':
    bc = BartClassifier()
    bc.run()
```