CitiBike: A Problem in Optimal Bicycle Redistribution

Problem Statement:

From a database of IOT data for bicycles in Chicago, determine a business process for redistributing bicycles to stations to maintain low variance in bikes available and optimize for fewest number of bicycles transported.

Data Structure and Exploratory Data Analysis:

We have two major tables. The first is the bicycle log data (citibike), which is our fact table for customer rides. The second is the station table, our dimension table that supplies descriptive information about each bicycle station.

The citibike ride table is shown below in figure 1, and the station table in figure 2.

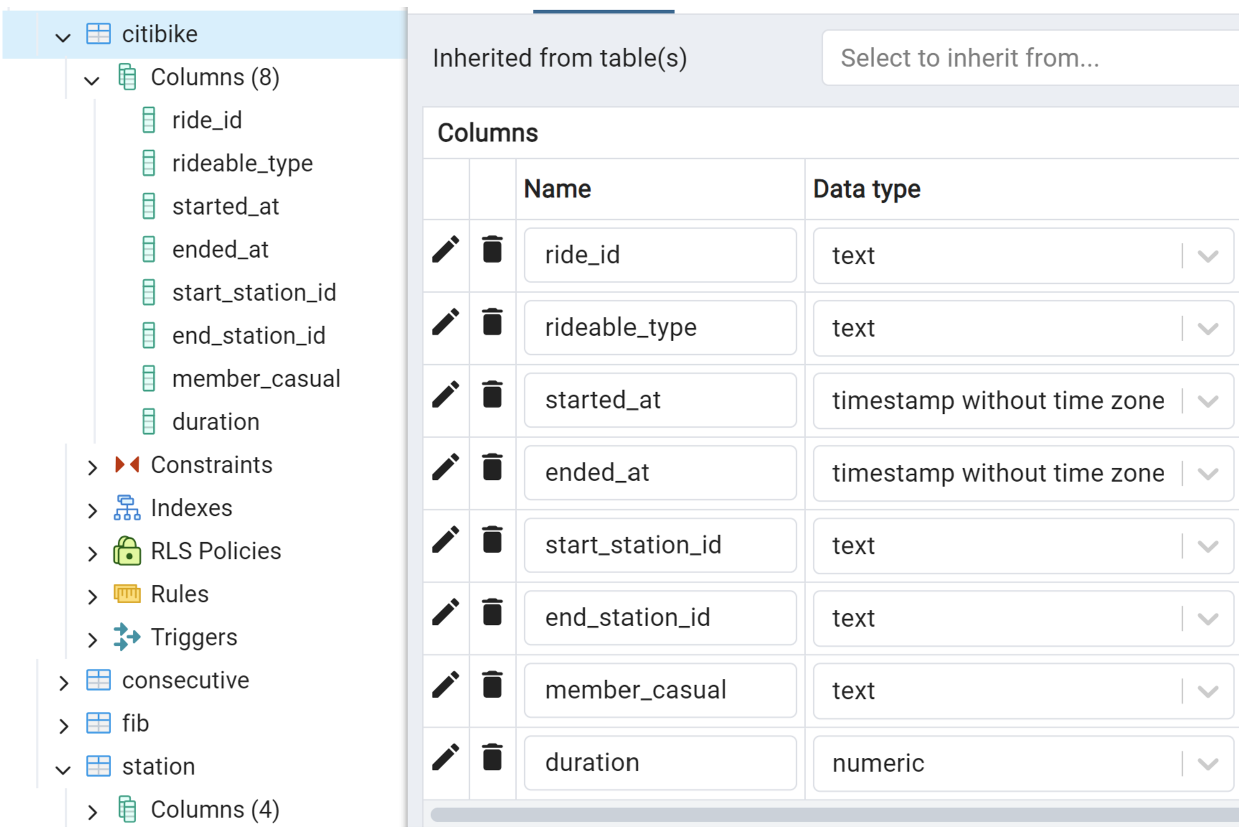


Fig 1. Structure of citibike table.

CitiBike: A Problem in Optimal Bicycle Redistribution

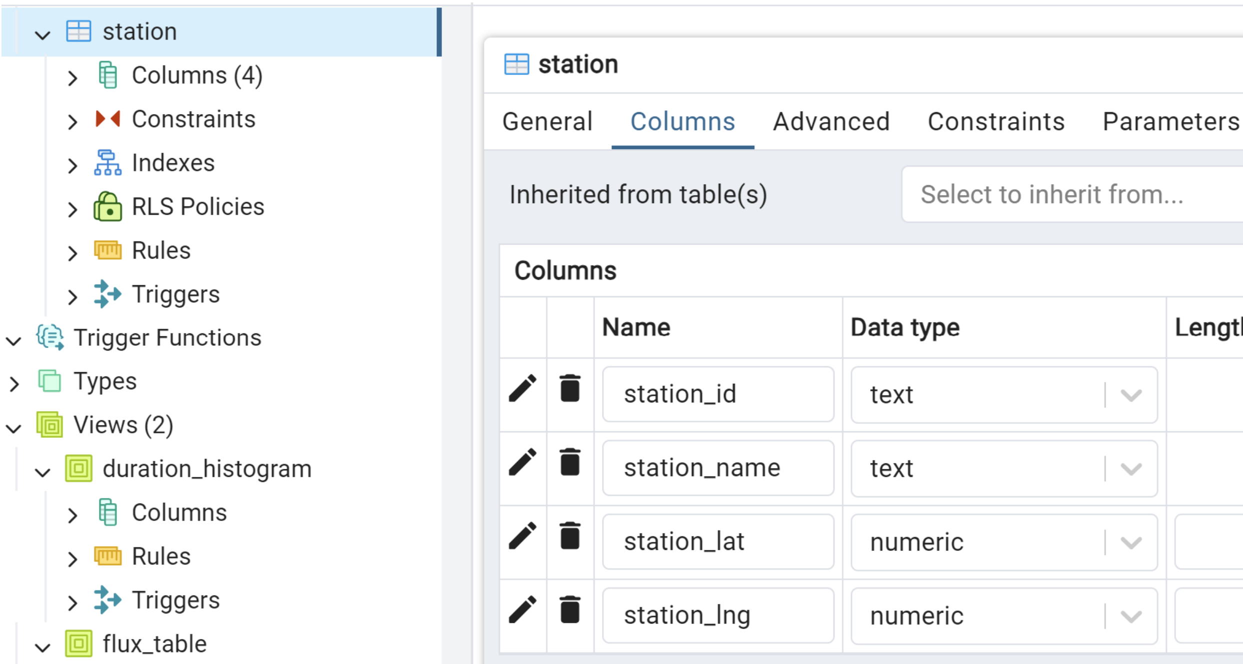
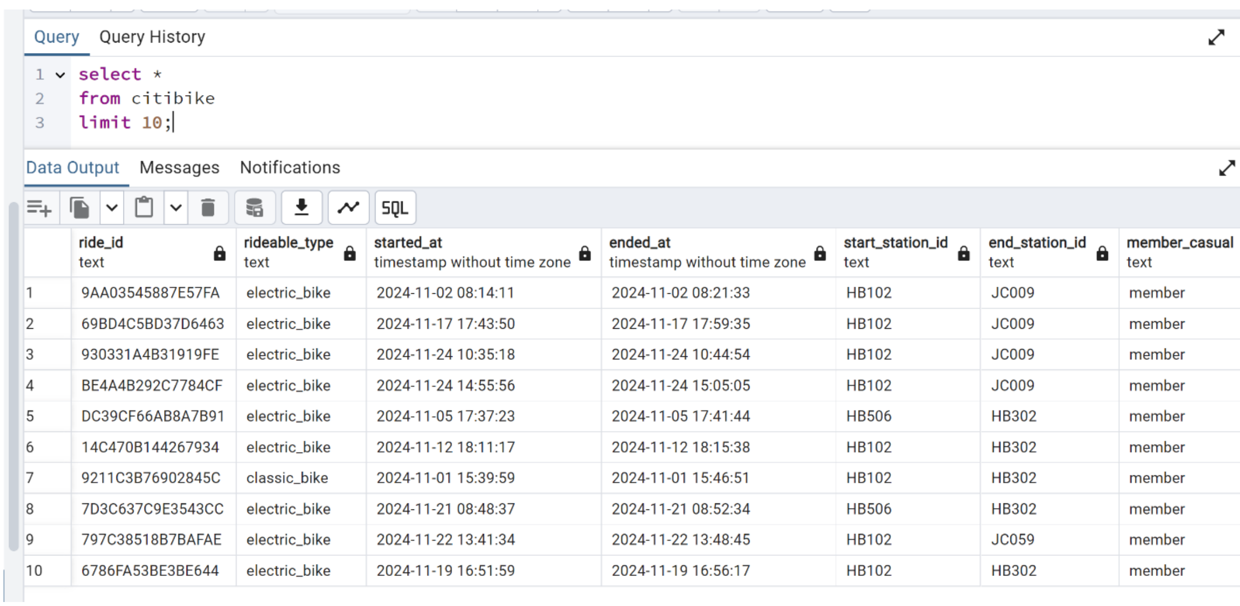


Fig 2. Structure of station table.

Here is a screenshot of the first few records in the citibike table.



```
select *
from citibike
limit 10;
```

	ride_id text	rideable_type text	started_at timestamp without time zone	ended_at timestamp without time zone	start_station_id text	end_station_id text	member_casual text
1	9AA03545887E57FA	electric_bike	2024-11-02 08:14:11	2024-11-02 08:21:33	HB102	JC009	member
2	69BD4C5BD37D6463	electric_bike	2024-11-17 17:43:50	2024-11-17 17:59:35	HB102	JC009	member
3	930331A4B31919FE	electric_bike	2024-11-24 10:35:18	2024-11-24 10:44:54	HB102	JC009	member
4	BE4A4B292C7784CF	electric_bike	2024-11-24 14:55:56	2024-11-24 15:05:05	HB102	JC009	member
5	DC39CF66AB8A7B91	electric_bike	2024-11-05 17:37:23	2024-11-05 17:41:44	HB506	HB302	member
6	14C470B144267934	electric_bike	2024-11-12 18:11:17	2024-11-12 18:15:38	HB102	HB302	member
7	9211C3B76902845C	classic_bike	2024-11-01 15:39:59	2024-11-01 15:46:51	HB102	HB302	member
8	7D3C637C9E3543CC	electric_bike	2024-11-21 08:48:37	2024-11-21 08:52:34	HB506	HB302	member
9	797C38518B7BAFAE	electric_bike	2024-11-22 13:41:34	2024-11-22 13:48:45	HB102	JC059	member
10	6786FA53BE3BE644	electric_bike	2024-11-19 16:51:59	2024-11-19 16:56:17	HB102	HB302	member

And similarly for the station table:

CitiBike: A Problem in Optimal Bicycle Redistribution

Query Query History

```sql
select *
from station
limit 10;
```

Data Output Messages Notifications

	station_id text	station_name text	station_lat numeric	station_lng numeric
1	JC080	Leonard Gordon Park	40.74590997	-74.05727148
2	HB302	6 St & Grand St	40.74439783	-74.03450087
3	HB608	2 St & Park Ave	40.73915273	-74.03308198
4	JC059	Heights Elevator	40.74871595	-74.0404433
5	JC008	Newport Pkwy	40.7287448	-74.0321082
6	HB203	Bloomfield St & 15 St	40.75453	-74.02658
7	HB603	8 St & Washington St	40.74598388	-74.02819902
8	5279.03	Centre St & Worth St	40.71494807	-74.00234482
9	5216.07	Vesey St & Greenwich St	40.71254669	-74.01113051
10	JC009	Hamilton Park	40.72759597	-74.04424731

Now, let us gather some information about the data. For a related case study, we may wish to analyze trends among members vs. casual riders. For the present study, we will just enumerate the rides taken by each, just to give us an idea of what type of customer ride is most common.

Query Query History

```sql
select 'Total Number of Rides', count(*)
from citibike
union
select '# Members', count(*)
from citibike
where member_casual = 'member'
union
select '# Non-Members', count(*)
from citibike
where member_casual = 'casual';
```

Data Output Messages Notifications

	?column? text	count bigint
1	Total Number of Rides	85073
2	# Non-Members	18146
3	# Members	66927

So, the vast majority of rides are by members.

We also want to know the data collection period … what is the date range for all the rides? We can accomplish this by finding the earliest and latest records for the rides.

Query   Query History

```sql
select started_at as first_record
from citibike
order by started_at
limit 1;
```

Data Output   Messages   Notifications

	first_record timestamp without time zone
1	2024-10-31 23:03:03

Query   Query History

```sql
select ended_at as last_record
from citibike
order by ended_at desc
limit 1;
```

Data Output   Messages   Notifications

	last_record timestamp without time zone
1	2024-11-30 23:59:46

So, the ride data encompasses one month near the end of 2024.

Finally, let's find how many unique stations we have in this dataset.

Query   Query History

```sql
select count(*) as num_stations
from station;
```

Data Output   Messages   Notifications

	num_stations bigint
1	195

It is useful to understand some basic metric like the above: how many records we are working with, what is the time span of the data, etc.

CitiBike: A Problem in Optimal Bicycle Redistribution

Now, we will dive into some deeper analysis. I want to understand how long these rides last, and how they are distributed. To do that, we first need to computer time duration of each ride. For this, we can use the epoch keyword.

```
select ended_at - started_at, extract(epoch from ended_at - started_at)
from citibike
limit 10;
```

Data Output Messages Notifications

	?column? interval	extract numeric
1	00:18:00	1080.000000
2	00:13:54	834.000000
3	00:17:50	1070.000000

You can verify that the first record in this relation (18 minutes) is equal to 1080 seconds (second column).

We update the ride table, filling in the duration column with values using the epoch. The result is shown below.

```
select started_at, ended_at, duration
from citibike
limit 10;
```

Data Output Messages Notifications

	started_at timestamp without time zone	ended_at timestamp without time zone	duration numeric
1	2024-11-13 18:27:41	2024-11-13 18:42:50	9.000000
2	2024-11-02 08:14:11	2024-11-02 08:21:33	22.000000
3	2024-11-17 17:43:50	2024-11-17 17:59:35	45.000000
4	2024-11-24 10:35:18	2024-11-24 10:44:54	36.000000
5	2024-11-20 05:42:09	2024-11-20 05:49:33	24.000000
6	2024-11-24 14:55:56	2024-11-24 15:05:05	9.000000
7	2024-11-05 17:37:23	2024-11-05 17:41:44	21.000000
8	2024-11-12 18:11:17	2024-11-12 18:15:38	21.000000
9	2024-11-01 15:39:59	2024-11-01 15:46:51	52.000000
10	2024-11-21 08:48:37	2024-11-21 08:52:34	57.000000

To visualize how rides are distributed by duration, I will create a histogram. An efficient way to to this when you want a large number of bins is to use a recursive query that creates the bins.

CitiBike: A Problem in Optimal Bicycle Redistribution

Query   Query History

```
with recursive cte(i, j) as (

    select 0, 1
    union all
    select cte.i+1, cte.i + 2
    from cte
    where cte.i < 59
)

select 60*cte.i, 60*cte.j, count(*)
from cte, citibike
where duration between 60*cte.i and 60*cte.j
group by 60*cte.i, 60*cte.j
order by 60*cte.i asc;
```

Data Output   Messages   Notifications

	?column? integer	?column? integer	count bigint
1	0	60	10
2	60	120	3252
3	120	180	8088
4	180	240	11345
5	240	300	11729

Here, we have 60 bins of size 60 seconds. The third record in this relation shows that 8,088 rides had a duration between 2 and 3 minutes.

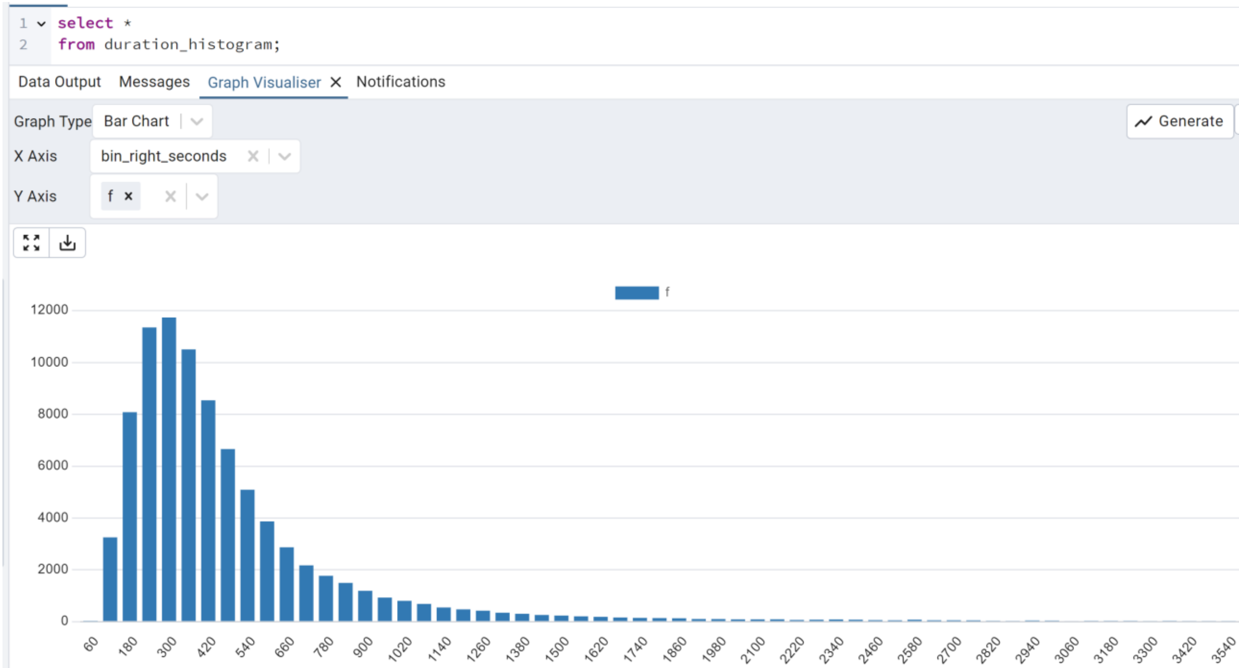
We can encapsulate this query into a view for reuse as needed.

Query   Query History

```
Create View Duration_Histogram(bin_left_seconds, bin_right_seconds, f) as (

with recursive cte(i, j) as (

    select 0, 1
    union all
    select cte.i+1, cte.i + 2
    from cte
    where cte.i < 59
)

select 60*cte.i, 60*cte.j, count(*)
from cte, citibike
where duration between 60*cte.i and 60*cte.j
group by 60*cte.i, 60*cte.j
order by 60*cte.i asc
)
```

Here is a visualization of this results set.

CitiBike: A Problem in Optimal Bicycle Redistribution



Most of the rides are only between 3 and 15 minutes. Finally, it might be useful to know, for any set of ride data, the duration that bounds 90% of the data. To do this, we can create a common curve in mathematics called a CDF (cumulative Density Function). This function tells us the percentage of data that falls below each value of interest (in our case, time).

We can create a CDF for a distribution by utilizing SQL window functions as follows.

Query   Query History

```sql
select bin_left_seconds, bin_right_seconds,
sum(f) over (order by bin_right_seconds
                rows between unbounded preceding
                and current row) as cumulative_f
from Duration_Histogram;
```

Data Output   Messages   Notifications

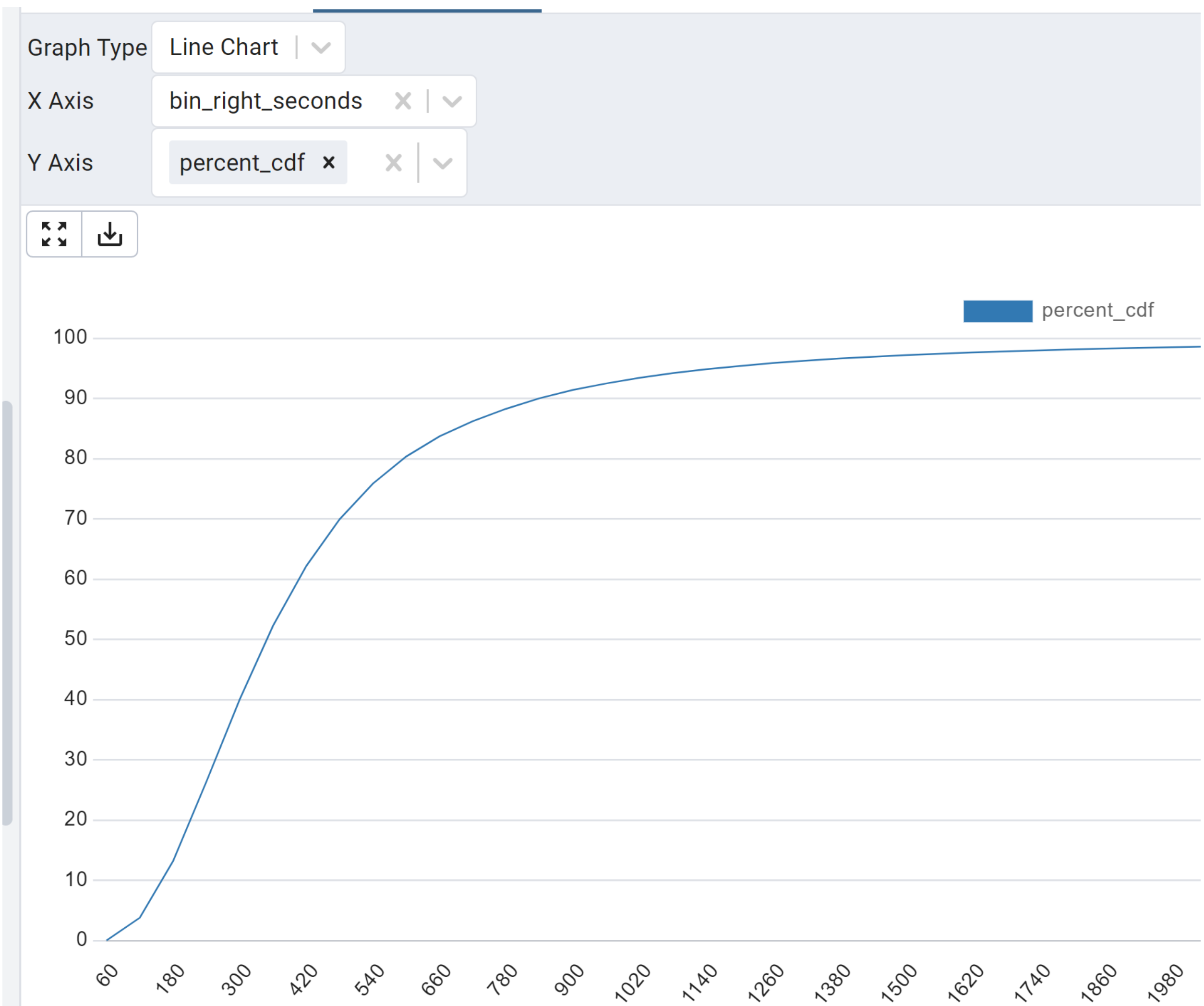
	bin_left_seconds integer	bin_right_seconds integer	cumulative_f numeric
1	0	60	10
2	60	120	3262
3	120	180	11350
4	180	240	22695

We can compute the percentages as follows.

Query   Query History

```
with cte as (
    select bin_left_seconds, bin_right_seconds,
    sum(f) over (order by bin_right_seconds
                rows between unbounded preceding
                and current row) as cumulative_f
    from Duration_Histogram
)

select bin_left_seconds, bin_right_seconds,
round(100 * cumulative_f / (select max(cumulative_f) from cte), 2) as percent_cdf
from cte;
```

This gives us a nice CDF for the data.



From the graph, 90% of the rides are less than 780 seconds in duration.

CitiBike: A Problem in Optimal Bicycle Redistribution

Now, for the main problem. We want to find how the stations accumulate or lose bikes over a period of time. In this case, 1 month. Denote this as the flux of a particular station. If the flux is negative, it means a bike was taken from the station, and if it is positive, the meaning is that a bike was left at the station. Below is a query for generating a table of flux for each ride.

Query Query History

```
create view flux_table as (

with cte1 as

(   select station_name,
    count(citibike.end_station_id) as flux
    from station join citibike
    on citibike.end_station_id = station.station_id
    group by station_name
),

cte2 as
(   select station_name,
    -1 * count(citibike.start_station_id) as flux
    from station join citibike
    on citibike.start_station_id = station.station_id
    group by station_name
)

select cte1.station_name, cte1.flux
from cte1
union
select cte2.station_name, cte2.flux
from cte2

);
```

Here, we are joining based on station id, which acts as a primary key for the station table, and the foreign key for the ride table.

Now, we need to find the net flux of each station by aggregating the flux of each ride by station as follows.

CitiBike: A Problem in Optimal Bicycle Redistribution

Query Query History

```sql
select station_name, sum(flux) as net_flux
from flux_table
group by station_name;
```

Data Output Messages Notifications

	station_name text	net_flux numeric
1	1 Ave & E 38 St	1
2	10 Ave & W 14 St	1
3	11 Ave & W 27 St	2
4	11 Ave & W 41 St	1
5	11 St & Washington St	1
6	12 Ave & W 40 St	4
7	12 St & Sinatra Dr N	17
8	14 St Ferry - 14 St & Shipyard Ln	-40
9	2 St & Park Ave	-26
10	4 St & Grand St	-12
11	4 St & River St	8

As we can see, some stations have almost no variance from the initial timestamp of the data, while others, like 2 St & Park Ave, have 26 fewer bikes.

These values are a bit large, because we are assuming no redistribution activity was done over the month, and the system was left to itself. For example, we don't even know if there were 40 bikes to begin with at 14 St Ferry … The point of this study is to model a potential solution for redistribution, not to be completely realistic. In reality, redistributing the bikes might be done on a weekly, or even daily basis.

Now, for the redistribution, here are our objectives:

1. Redistribute bikes such that the variance at each station does not exceed 10.
2. Optimize for the total number of bikes that need to be transported, minimizing this quantity.

Mathematically, define X_{ij} to be the number of bikes moved from station i to station j, let X_i denote the initial net flux at station i (the constants in the above table). The first condition is equivalent to

CitiBike: A Problem in Optimal Bicycle Redistribution

$$-10 \leq X_i + \sum X_{ji} - \sum X_{ij} \leq 10$$

Where the sums are over all paris I,j.

Another set of conditions is on the values of the variables themselves. They should be integers, and nonnegative.

$$X_{ij} \geq 0$$

Finally, the objective is to minimize $\sum X_{ij}$.

The total number of variables is equal to the square of the number of stations. There are 195 stations, and so this gives a need for 38,025 variables! Remarkably, this problem can by solved in less than 2 seconds on a laptop, using python's PULP framework for linear programming.

We start by querying all the net flux data and saving this into a csv file.

Query Query History

```
with cte as (
    select station_name, sum(flux) as net_flux
    from flux_table
    group by station_name
)
select station_name, net_flux
from cte;
```

Data Output Messages Notifications

	station_name text	net_flux numeric
1	1 Ave & E 38 St	1
2	10 Ave & W 14 St	1
3	11 Ave & W 27 St	2
4	11 Ave & W 41 St	1
5	11 St & Washington St	1
6	12 Ave & W 40 St	4
7	12 St & Sinatra Dr N	17
8	14 St Ferry - 14 St & Shipyard Ln	-40
9	2 St & Park Ave	-26
10	4 St & Grand St	-12

CitiBike: A Problem in Optimal Bicycle Redistribution

In python, we start by importing necessary libraries. In this case, pulp and pandas.

```python
1  import pulp as p # type: ignore
2  import pandas as pd # type: ignore
```

Next, we define our problem as a minimization problem and define the variables (lines 8-12), objective (line 15) and import our data (lines 17-18).

```python
4  Distribution = p.LpProblem('Distribution', p.LpMinimize)
5
6  #Create problem variables.
7
8  possibleTrips = [tuple(c) for c in p.permutation(range(1,196), 2)]
9
10 X = p.LpVariable.dicts(
11     "X", possibleTrips, lowBound=0, upBound = None, cat=p.LpInteger
12 )
13
14 #Objective function
15 Distribution += p.lpSum([X[Trip] for Trip in possibleTrips])
16
17 df = pd.read_csv('station_net_flux.csv')
18 initialData = df['net_flux'].to_list()
```

Next, define our constraints on final variance.

```python
20 #Constraints
21
22 #Flux
23
24 for i in range(1,196):
25     Distribution += initialData[i-1] + p.lpSum(X[T] for T in possibleTrips if T[1]==i and T[0] != i) \
26                 - p.lpSum(X[T] for T in possibleTrips if T[0] == i and T[1] != i) >= -10
27     Distribution += initialData[i-1] + p.lpSum(X[T] for T in possibleTrips if T[1]==i and T[0] != i) \
28                 - p.lpSum(X[T] for T in possibleTrips if T[0] == i and T[1] != i) <= 10
29
30
```

The final step is to solve the model and print out the values.

```python
31 Distribution.solve()
32 Distribution.writeLP("DistributionModel.lp")
33
34 for v in Distribution.variables():
35     if v.varValue != 0:
36         print(v.name, "=", v.varValue)
37
38 #Show final result after trips completed.
39
```

Here is a screenshot of the first few lines of the output.

```
Result - Optimal solution found

Objective value:                921.00000000
Enumerated nodes:               0
Total iterations:               0
Time (CPU seconds):             0.95
Time (Wallclock seconds):       0.94

Option for printingOptions changed from normal to all
Total time (CPU seconds):       1.20   (Wallclock seconds):       1.20

X_(1,_91) = 11.0
X_(100,_104) = 40.0
X_(100,_114) = 2.0
X_(100,_116) = 4.0
X_(100,_118) = 20.0
X_(100,_122) = 6.0
X_(100,_134) = 13.0
X_(100,_141) = 64.0
X_(100,_21) = 1.0
X_(100,_24) = 6.0
X_(100,_33) = 39.0
X_(100,_35) = 12.0
X_(100,_37) = 7.0
X_(100,_53) = 73.0
X_(100,_86) = 53.0
X_(102,_129) = 10.0
X_(106,_107) = 53.0
```

For this case, we need to move 921 bikes, and the variables give us a directive for accomplishing this. The final step is to use the solution to tell us about the final variance at each of the ten stations. One way to do this is to transfer the data and results to Excel and recompute flux at each station using the SUMIF function.

D2 =B2+SUMIF($G$3:$G$59, C2, $H$3:$H$59) - SUMIF($F$3:$F$59, C2, $H$3:$H$59)

	A	B	C	D	E	F	G	H
1	station_name	net_flux	station_number	final variance		Directive		
2	1 Ave & E 38 St	1	1	-10		From Station	To Station	Number of Bikes
3	10 Ave & W 14 St	1	2	1		1	91	11
4	11 Ave & W 27 St	2	3	2		100	104	40
5	11 Ave & W 41 St	1	4	1		100	114	2
6	11 St & Washington St	1	5	1		100	116	4
7	12 Ave & W 40 St	4	6	4		100	118	20
8	12 St & Sinatra Dr N	17	7	10		100	122	6
9	14 St Ferry - 14 St & Shipyard Ln	-40	8	-10		100	134	13
10	2 St & Park Ave	-26	9	-10		100	141	64
11	4 St & Grand St	-12	10	-10		100	21	1
12	4 St & River St	8	11	8		100	24	6
13	47 Ave & 31 St	2	12	2		100	33	39

The variance at each of the 195 stations is between -10 and 10, as we specified.

Why is this a useful process? One reason is that, counting the data output, we have only 57 non-zero X_ij. So, to redistribute the bikes among 195 stations, we need to only make 57 trips, assuming we have the truck capacity to transport the number needed for each trip.

Final Remarks and Ideas for Further Work

In a real business setting, one idea for implementation would be to encapsulate this query process into a trigger function that runs automatically every week. Then, running the mathematical program provides a business process for employees to redistribute bicycles as needed to balance the available bikes. If the process is done more frequently, there will be less burden in terms of number of bikes we need to move.

The integer linear program could also be made more sophisticated by perhaps optimizing for fewest number of trips rather than number of bicycles.