Tenzin Tashi

Prof. Hesham A Auda

CSc - 33600 Introduction to Database Systems

May 11, 2021

Assignment 4

CitiBike

Assignment:

[I]

1- Study the data provided for the use of Citi bikes to gain an understanding of the program utilization.

2- Explore the information on the program available on the Web to verify the basic facts and collect additional information regarding the utilization of Citi Bikes. Write down the additional information and/or requirements you gathered. Give the URLs for the sites contacted.

After Studying the data provided for the use of Citi bikes, I was able to get the basic understanding of the data and what it is trying to show, helping me to create this analysis report. This repot helps to better understand the Citibike ridership for the 2013 year. It containeds

- ⇨ The most frequent trips between any two stations by the day of the week
- ⇨ Categorization of data based on age group
- ⇨ Categorization of data based on gender
- ⇨ Categorization of data based on days of week
- ⇨ Categorization of data based on dormant and vacant bike station
- ⇨ Finding the area that uses the program the most with the help of zip code.

The day of the week (start day and end day), and zip code was not provided in the data

set so it was explicitly entered. For the day of the week I used the SQL in build function DATENAME with the help of the start time and end time to find the start day and end day of the week.

Similarly, for the zip code I coded a python code that uses the Google Geolocation API that takes two parameter the latitude and longitude which were obtained from the Stations table, and it returns a csv file with all the station and its respective zip codes.

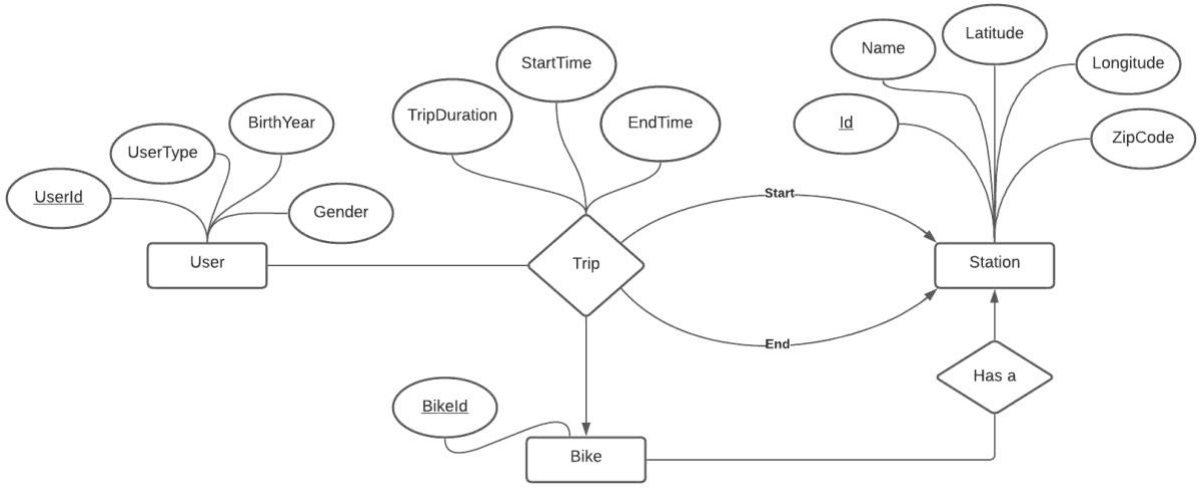
NOTE: The python code snippet is posted at the end of this report and the file containing the source code is attached together with the reported.

Reference:

https://www.citibikenyc.com/system-data [Citibike Data Trip file]

https://developers.google.com/maps/documentation/geolocation/cloud-setup [Google Geolocation API]

3- Draw an appropriate E/R diagram that satisfies the basic and additional facts, indicating, weak and subclass entity sets, whenever exist, multiplicity of relationships, and the key, or keys, for each entity set. Distinguish between the parts of the E/R diagram pertaining to the given and extra basic facts.



4- Translate the E/R diagram in [3] to a relational database schema.

Relational Schema:

Station (<u>Id</u>, Name, Latitude, Longitude, ZipCode)
Trip (TripDuration, StartTime, EndTime)
User (<u>UserId</u>, UserType, BirthYear, Gender)
Bike (<u>BikeId</u>)

5- Specify a number of essential functional dependencies for each relation. Identify possible

keys, whenever exist, and the primary key and foreign keys for each relation.

Functional dependencies:

Stations:

Id-> Name, Latitude, Longitude, ZipCode (primary key) : Given station Id we can find the name of the station, latitude and longitude of the station, and the zip code of the station.
(latitude, longitude) -> ZipCode : ZipCode is determined by the latitude and longitude

User:

UserId -> BirthYear, Gender, UserType : Given the user id we can find the information of the user such as his/her birth year, gender, user type.

Bike:

BikeId -> stationId : Given a BikeId we can find where the bike is at the current type such as which station it is currently docked at.

Trip:

(StartTime, EndTime) -> TripDuration: Given the start time and end time we can calculate the trip duration.

6- Examine the database relations for possible BCNF and/or 3NF violations. Decompose the relations as appropriate.

Based on my current Relational schema, the Station relation violates the 3NF so it also violates BCNF as 3NF cannot have a transitive dependency that is (latitude, longitude) -> ZipCode.

To make it accepted I decompose the relation station into two different tables namely Stations(Id, Name, Latitude, Longitude) and StationZipCode(Latitude, Longitude, ZipCode).

[II]

Create SQL expressions to break the data provided in the file into the following tables:

Stations(Id, Name, Latitude, Longitude)

CODE:

```
/* Create Stations Table */

CREATE TABLE Stations (

	Id INT,

	Name VARCHAR(64),

	Latitude FLOAT,

	Longitute FLOAT

);

/* Bike Station data: Partition StartStation072013 */

WITH StartStation072013 AS (

SELECT  StartStationId AS 'Station Id', StartStationName AS 'Name', StartStationLatitude AS 'Latitude',

StartStationLongitude AS 'Longitude',

	ROW_NUMBER() OVER(PARTITION BY StartStationId

			ORDER BY StartStationId) AS RowNumber

FROM    CitiBike072013)
```

```
/* Insert into Stations Table from the Partition */

INSERT INTO Stations (Id, Name, Latitude, Longitute)

SELECT [Station Id], Name, Latitude, Longitude

FROM StartStation072013

WHERE RowNumber = 1;

GO



/* Display Stations data */

SELECT DISTINCT * FROM Stations

ORDER BY Id;

GO
```

OUTPUT:

	Id	Name	Latitude	Longitute
1	72	W 52 St & 11 Ave	40.76727216	-73.99392888
2	79	Franklin St & W Broadway	40.71911552	-74.00666661
3	82	St James Pl & Pearl St	40.71117416	-74.00016545
4	83	Atlantic Ave & Fort Green…	40.68382604	-73.97632328
5	116	W 17 St & 8 Ave	40.74177603	-74.00149746
6	119	Park Ave & St Edwards St	40.69608941	-73.97803415
7	120	Lexington Ave & Classon A…	40.68676793	-73.95928168
8	127	Barrow St & Hudson St	40.73172428	-74.00674436
9	128	MacDougal St & Prince St	40.72710258	-74.00297088
10	137	E 56 St & Madison Ave	40.761628	-73.972924

Narration:

In the above SQL expression, I first partitioned the main table CitiBike072013 into StationId, StationName, StartStationLatitude, and StartStationLongitute then using those values I created another table name Stations and populated the table. The reason I am only using StartStationLatitude and StartStationLongitute but not the EndStationLatitude and EndStationLongitude is because we need the list of station and since the start station can be end station and we don't want the duplicate so I obmited the end station latitude and longitude.

The other was to get the same solution can be by using the end station latitude and end station longitude instead of start station latitude and start station longitude in the above SQL expression.

Trips(StationId, MinTripDuration, MaxTripDuration, AvgTripDuration, NumberUsers)

CODE:

```
/* Create Trips Table */

CREATE TABLE Trips (

	StationId INT,

	MinTripDuration INT,

	MaxTripDuration INT,

	AvgTripDuration FLOAT,

	NumberOfStartUsers INT,

	NumberOfReturnUsers INT

);




/* Trip statistics partition Trips072013 */

WITH Trips072013 AS(

SELECT S.[Station ID], S.[Minimum Trip Duration], S.[Maximum Trip Duration], S.[Average Trip Duration],

	S.[Number of Start Users], T.[Number of Return Users]

FROM	(SELECT DISTINCT StartStationId AS 'Station ID',

	MIN(TripDuration) OVER(PARTITION BY StartStationId) AS 'Minimum Trip Duration',
```

```
        MAX(TripDuration) OVER(PARTITION BY StartStationId) AS 'Maximum Trip Duration',

        AVG(TripDuration) OVER(PARTITION BY StartStationId) AS 'Average Trip DUration',

                COUNT(*) OVER(PARTITION BY StartStationId) AS 'Number of Start Users'

        From CitiBike072013) S

        INNER JOIN

        (SELECT DISTINCT EndStationId AS 'Station Id',

                COUNT(*) OVER(PARTITION BY EndStationId) AS 'Number of Return Users'

        FROM CitiBike072013) T

        ON S.[Station ID] = T.[Station Id]

)

/* Insert into Trips Table from the Partition */

INSERT INTO Trips (StationId, MinTripDuration, MaxTripDuration, AvgTripDuration, NumberOfStartUsers,

NumberOfReturnUsers)

SELECT *

FROM Trips072013

ORDER BY [Station ID];



/* Display Trips data */
```

```
SELECT StationId AS 'Station Id', MinTripDuration AS 'Minimum Trip Duration', MaxTripDuration AS 'Maximum Trip
Duration', AvgTripDuration AS 'Average Trip Duration', NumberOfStartUsers AS 'Number of Start Users',
NumberOfReturnUsers AS 'Number of Return Users'
FROM Trips;
GO
```

OUTPUT:

	Station Id	Minimum Trip Duration	Maximum Trip Duration	Average Trip Duration	Number of Start Users	Number of Return Users
1	389	60	88516	1425.650234741784	852	903
2	397	70	45492	1234.5570866141732	508	515
3	533	64	37555	918.59375	2432	1706
4	278	68	22628	992.6884057971015	552	558
5	375	61	20733	844.0087583719732	3882	3883
6	352	65	76280	1228.0859060402684	3725	3916
7	285	60	74461	866.9444676991768	7167	7052
8	406	64	37167	1001.5339987523394	1603	1592
9	224	60	30742	1087.0105359596885	2183	2062
10	268	62	69345	936.345555171221	2891	2874

Narrative:

Similar to the earlier SQL expression we partitioned the Citibike072013 table to obtain the station id, trip duration, and number of users. Then with this data we first find the minimum number of Trip that started at the given station id, then the maximum number of Trip that started at the given station id, Average of it, and counted the number of users that started and ended at the given station id.

UsageByDay(StationId, NumberUsersWeekday, NumberUsersWeekend)

CODE:

```
/* Create UsageByDay Table */
CREATE TABLE UsageByDay(
```

```
	StationId INT,

	NumberUsersWeekday INT,

	NumberUsersWeekend INT

);

/* User Statistics by weekday/weekend: Partition DayStatistics072013 */

WITH DayStatistics072013 AS(

SELECT R.Id AS 'Station Id', MAX(R.MondayUsers) + MAX(R.TuesdayUsers) + MAX(R.WednesdayUsers) +

MAX(R.ThursdayUsers) + MAX(R.FridayUsers) AS 'Weekday Users',

			MAX(R.SaturdayUsers) + MAX(R.SundayUsers) AS 'Weekend Users'

FROM	(SELECT StartStationId AS 'Id',

		[MondayUsers] = CASE WHEN StartDay = 'Monday' THEN COUNT(*) OVER(PARTITION BY

StartStationId, StartDay) END,

		[TuesdayUsers] = CASE WHEN StartDay = 'Tuesday' THEN COUNT(*) OVER(PARTITION BY

StartStationId, StartDay) END,

		[WednesdayUsers] = CASE WHEN StartDay = 'Wednesday' THEN COUNT(*) OVER(PARTITION BY

StartStationId, StartDay) END,

		[ThursdayUsers] = CASE WHEN StartDay = 'Thursday' THEN COUNT(*) OVER(PARTITION BY

StartStationId, StartDay) END,
```

```
            [FridayUsers] = CASE WHEN StartDay = 'Friday' THEN COUNT(*) OVER(PARTITION BY StartStationId,
StartDay) END,
            [SaturdayUsers] = CASE WHEN StartDay = 'Saturday' THEN COUNT(*) OVER(PARTITION BY
StartStationId, StartDay) END,
            [SundayUsers] = CASE WHEN StartDay = 'Sunday' THEN COUNT(*) OVER(PARTITION BY
StartStationId, StartDay) END
      FROM CitiBike072013) R
GROUP BY R.Id)
/* Insert into UsageByDay Table */
INSERT INTO UsageByDay (StationId, NumberUsersWeekday, NumberUsersWeekend)
SELECT *
FROM DayStatistics072013
ORDER BY [Station Id];
/* Display UsageByDay data */
SELECT StationId AS 'Station Id', NumberUsersWeekday AS 'Weekday Users', NumberUsersWeekend AS
'Weekend Users'
FROM UsageByDay
GO
```

OUTPUT:

	Station Id	Weekday Users	Weekend Users
1	82	818	323
2	83	1105	484
3	116	2811	710
4	147	2931	1081
5	150	1188	416
6	164	1909	434
7	167	2427	476
8	216	519	336
9	217	1170	863
10	232	754	394

Narrative:

Using the StartDay we first partitioned the CitiBike072013 table into its respective days of the week then we counted the numbers of users in all those days and filtered out the max of it days. Then lastly, we added all the max users during the days of weekdays and similarly for the weekends. Therefore, we obtained the above results.

UsageByGender(StationId, NumberMaleUsers, NumberFemaleUsers)

CODE:

```
/* Create UsageByGender Table */

CREATE TABLE UsageByGender(

    StationId INT,

    NumberMaleUsers INT,

    NumberFemaleUsers INT,

    UnidentifiedUsers INT
```

```
);


/* User Statistics by gender: Partition UsageByGender */

WITH GenderStatistics072013 AS(

SELECT R.Id AS 'Station Id', MAX(R.MaleUsers) AS 'Male Users', MAX(R.FemaleUsers) AS 'Female Users',

MAX(R.UnidentifiedUsers) AS 'Unidentified Users'

FROM    (SELECT StartStationId AS 'Id',

            [MaleUsers] = CASE WHEN Gender = 1 THEN COUNT(*) OVER(PARTITION BY StartStationId, Gender)

END,

            [FemaleUsers] = CASE WHEN Gender = 2 THEN COUNT(*) OVER(PARTITION BY StartStationId,

Gender) END,

            [UnidentifiedUsers] = CASE WHEN Gender = 0 THEN COUNT(*) OVER(PARTITION BY StartStationId,

Gender) END

      FROM CitiBike072013) R

GROUP BY R.Id)


INSERT INTO UsageByGender (StationId, NumberMaleUsers, NumberFemaleUsers, UnidentifiedUsers)

SELECT *

FROM GenderStatistics072013
```

```
ORDER BY [Station Id];


/* Display UsageByGender data */

SELECT StationId AS 'Station Id', NumberMaleUsers AS 'Male Users', NumberFemaleUsers AS 'Female Users',

UnidentifiedUsers AS 'Unidentified Users'

FROM UsageByGender

GO
```

OUTPUT:

Results Messages

	Station Id	Male Users	Female Users	Unidentified Users
1	72	2144	645	786
2	79	2624	1076	1334
3	82	592	203	346
4	83	820	305	464
5	116	2350	676	495
6	119	54	39	24
7	120	393	179	79
8	127	2731	957	800
9	128	2854	974	810
10	137	358	90	199
11	143	557	211	148
12	144	140	53	224

Narrative:

Using the information obtained from the citibikenyc website (link given at the beginning of this report) we found out that datafield of 1 in gender column represent the male, 2 represent the female, and 0 represent unidentified gender. So instead of creating two column of male and female I added an extra column as Unidentified User. This columns value was simply calculated by counting the number of 1's, 2's and 0's at the specific station id for get the number of Male user, Female user, and Unidentified user respectively at that given station.

UsageByAge(StationId, NumberUsersUnder18, NumberUsers18To40, NumberUsersOver40)

CODE:

```
/* Create UsageByAge Table */

CREATE TABLE UsageByAge(

	StationId INT,

	NumberUsersUnder18 INT,

	NumberUsers18To40 INT,

	NumberUsersOver40 INT

);

/* User Statistics by age group: Partition AgeStatistics072013 */

DECLARE @YearStartTime INT = 2013;

WITH AgeStatistics072013 AS (

SELECT S.Id AS 'Station Id', SUM(S.A) AS 'User Under 18 Years Old', SUM(S.B) AS 'User 18-40 Years Old',

SUM(S.C) AS 'Users Over 40 Years Old'

FROM	(SELECT R.Id AS 'Id', MAX(R.UsersUnder18) AS A, MAX(R.Users18To40) AS B, MAX(R.UsersOver40) AS

C

	FROM	(SELECT	StartStationId AS 'Id', BirthYear,
```

```
                    [UsersUnder18] = CASE WHEN (BirthYear <> '\N' AND (@YearStartTime - CAST(BirthYear AS
INT)) < 18) THEN COUNT(BirthYear) OVER(PARTITION BY StartStationId, BirthYear) END,
                [Users18To40] = CASE WHEN (BirthYear <> '\N' AND (@YearStartTime - CAST(BirthYear AS INT)) >=
18 AND (@YearStartTime - CAST(BirthYear AS INT)) <= 40) THEN COUNT(BirthYear) OVER(PARTITION BY
StartStationId, BirthYear) END,
                    [UsersOver40] = CASE WHEN (BirthYear <> '\N' AND (@YearStartTime - CAST(BirthYear AS INT))
> 40) THEN COUNT(BirthYear) OVER(PARTITION BY StartStationId, BirthYear) END
            FROM CitiBike072013) R
      GROUP BY R.Id, R.BirthYear) S
GROUP BY S.Id)
/* Insert into UsageByAge Table */
INSERT INTO UsageByAge (StationId, NumberUsersUnder18, NumberUsers18To40, NumberUsersOver40)
SELECT *
FROM AgeStatistics072013
ORDER BY [Station Id];
/* Display UsageByAge data */
SELECT StationId AS 'Station Id', NumberUsersUnder18 AS 'Users Under 18 Years Old', NumberUsers18To40 AS
'Users 18-40 Years Old', NumberUsersOver40 AS 'Users Over 40 Years Old'
FROM UsageByAge
```

```
GO
```

OUTPUT:

	Station Id	User Under 18 Years Old	User 18-40 Years Old	Users Over 40 Years Old
1	72	6	1851	931
2	79	14	2444	1242
3	82	*NULL*	486	309
4	83	1	831	293
5	116	2	2045	979
6	119	*NULL*	70	23
7	120	*NULL*	427	145
8	127	10	2417	1261
9	128	2	2478	1348
10	137	*NULL*	312	136
11	143	3	474	291
12	144	*NULL*	147	46

Narrative:

Using the birth year from the CitiBike072013 we calculated age of the riders age at the time of trip by subtracting the year of the ride i.e., 2013 by the birth year and partition the ages into three different section of riders under the age of 18, riders between the age of 18 to 40, and riders over the age of 40 at each given station id.

[III]

1- Create appropriate SQL expressions to determine the most frequent trips between any two stations by the day of the week.

CODE:

```
/* Most frequent trips between any two stations: Partition MostFrequentTrip072013 */

WITH MostFrequentTrip072013 AS(

SELECT StartStationId AS 'Start Station', EndStationId AS 'End Station', COUNT(*) AS 'Number of Trips',
```

```
    AVG(TripDuration) AS 'Average Trip Duration'

FROM CitiBike072013

GROUP BY StartStationId, EndStationId)



/* Display TOP(50) MostFrequentTrip072017 data */

SELECT TOP(50) *

FROM MostFrequentTrip072013

ORDER BY [Number of Trips] DESC, [Start Station];
```

OUTPUT:

Results | Messages

	Start Station	End Station	Number of Trips	Average Trip Duration
1	2006	2006	2099	2491.211052882325
2	281	281	909	2553.794279427943
3	499	499	643	2442.98600311042
4	387	387	503	1708.1093439363817
5	327	327	447	1455.7069351230425
6	363	327	422	672.3791469194313
7	426	426	421	1427.6080760095012
8	363	363	388	1628.8994845360826
9	457	457	354	2675.6186440677966
10	281	2006	337	2929.0741839762613
11	426	363	333	957.6006006006006
12	327	363	326	711.3865030674847

Narrative:

We partitioned the CitiBike072013 table to obtain the start station id and end station id, then using this id we calculated the total number of trips and average trip duration using SQL query. The above output shows the final result obtained.

2- Create appropriate SQL expressions to determine usage patterns by gender and age for any given station.

CODE:

```
DECLARE @YearStartTime INT = 2013;
WITH AgeStatistics072013 AS (
SELECT S.Id AS 'Station Id', SUM(S.A) AS 'User Under 18 Years Old', SUM(S.B) AS 'User 18-40 Years Old',
SUM(S.C) AS 'Users Over 40 Years Old'
FROM    (SELECT R.Id AS 'Id', MAX(R.UsersUnder18) AS A, MAX(R.Users18To40) AS B, MAX(R.UsersOver40) AS
C
      FROM    (SELECT     StartStationId AS 'Id', BirthYear,
                  [UsersUnder18] = CASE WHEN (BirthYear <> '\N' AND (@YearStartTime - CAST(BirthYear AS
INT)) < 18) THEN COUNT(BirthYear) OVER(PARTITION BY StartStationId, BirthYear) END,
                  [Users18To40] = CASE WHEN (BirthYear <> '\N' AND (@YearStartTime - CAST(BirthYear AS
INT)) >= 18 AND (@YearStartTime - CAST(BirthYear AS INT)) <= 40) THEN COUNT(BirthYear) OVER(PARTITION
BY StartStationId, BirthYear) END,
                  [UsersOver40] = CASE WHEN (BirthYear <> '\N' AND (@YearStartTime - CAST(BirthYear AS INT))
> 40) THEN COUNT(BirthYear) OVER(PARTITION BY StartStationId, BirthYear) END
            FROM CitiBike072013) R
      GROUP BY R.Id, R.BirthYear) S
GROUP BY S.Id)

SELECT *
FROM AgeStatistics072013
ORDER BY [Station Id];

WITH GenderStatistics072013 AS(
SELECT R.Id AS 'Station Id', MAX(R.MaleUsers) AS 'Male Users', MAX(R.FemaleUsers) AS 'Female Users',
MAX(R.UnidentifiedUsers) AS 'Unidentified Users'
FROM    (SELECT StartStationId AS 'Id',
```

```
        [MaleUsers] = CASE WHEN Gender = 1 THEN COUNT(*) OVER(PARTITION BY StartStationId, Gender)
END,
        [FemaleUsers] = CASE WHEN Gender = 2 THEN COUNT(*) OVER(PARTITION BY StartStationId,
Gender) END,
        [UnidentifiedUsers] = CASE WHEN Gender = 0 THEN COUNT(*) OVER(PARTITION BY StartStationId,
Gender) END
    FROM CitiBike072013) R
GROUP BY R.Id)

SELECT *
FROM GenderStatistics072013
ORDER BY [Station Id];
```

Output:

	Station Id	User Under 18 Years Old	User 18-40 Years Old	Users Over 40 Years Old
1	72	6	1851	931
2	79	14	2444	1242
3	82	*NULL*	486	309
4	83	1	831	293
5	116	2	2045	979
6	119	*NULL*	70	23
7	120	*NULL*	427	145
8	127	10	2417	1261
9	128	2	2478	1348
10	137	*NULL*	312	136
11	143	3	474	291
12	144	*NULL*	147	46
13	146	3	1085	895
14	147	2	2087	1106
15	150	*NULL*	1040	335
16	151	1	3191	1134
17	152	4	1267	809

	Station Id	Male Users	Female Users	Unidentified Users
1	72	2144	645	786
2	79	2624	1076	1334
3	82	592	203	346
4	83	820	305	464
5	116	2350	676	495
6	119	54	39	24
7	120	393	179	79
8	127	2731	957	800
9	128	2854	974	810
10	137	358	90	199
11	143	557	211	148
12	144	140	53	224
13	146	1461	522	385
14	147	2371	824	817
15	150	1043	332	229
16	151	3161	1165	1329
17	152	1642	438	574

Narrative:

From the main SQL table CitiBike072013, first I partition the user into three different subsection by their age. To calculate the age I use a variable named @YearStartTime and set it to 2013 because the data for the citi bike is of 2013 then with the use of the BirthYear field I subtract the YearStartTime to BirthYear to get the age of the rider at that year. After obtaining the age of the rider I created a sub-query to count the number of user who were under the age of 18, between the age of 18 to 40, and over the age of 40 who ride the citibike at the given station. Therefore, the above output was obtained.

Similarly, for the partition of the table with respect to the gender I counted the number of male user i.e., counted the number of 1's in the Gender field of the CitiBike072013 at the given station, same for the female I counted the number of 2's in the Gender field of the CitiBike072013, But since some of the field values were 0 meaning the gender of the rider were

unidentified so I created and extra column with total number of 0's at the given station as an unidentified users.

3- Create appropriate SQL expressions to identify any dormant or vacant bike stations.

CODE:

```
/* RED FLAG -- Dormant Station: All End Stations that are NOT Start Stations */

WITH DormantStations072013 AS(

SELECT  EndStationId AS 'Dormant Station', EndStationName AS 'Name', EndStationLatitude AS 'Latitude',

      EndStationLongitude AS 'Longitude'

FROM    CitiBike072013

WHERE   EndStationId NOT IN (SELECT StartStationId

                  FROM   CitiBike072013))



/* Display Dormant Station */

SELECT *

FROM DormantStations072013

ORDER BY [Dormant Station];

GO
```

OUTPUT:

	Dormant Sta…	Name	Latitude	Longitude

Narrative:

The above output is empty shows that all the stations are not a dormant station meaning there is no station such that it is end station for all riders and not a start station for any riders.

CODE:

```
/* RED FLAG -- Vacant Station: All Start Stations that are NOT End Stations will eventually become vacant */

WITH VacantStations072013 AS(

SELECT  StartStationId AS 'Vacant Station', StartStationName AS 'Name', StartStationLatitude AS 'Latitude',

      StartStationLongitude AS 'Longitude'

FROM    CitiBike072013

WHERE   StartStationId NOT IN (SELECT EndStationId

                  FROM   CitiBike072013))

/* Display Vacant Station */

SELECT *

FROM VacantStations072013

ORDER BY [Vacant Station];

GO
```

OUTPUT:

	Vacant Stat…	Name	Latitude	Longitude

Narrative:

The above output is empty because we do not have any station which is every user start point but not end point, meaning at the end of the day some riders use it as end station and others as a start station.

4- Reconstruct the tables and recreate the SQL expressions above to include zip codes, hence allowing for aggregation by area. Identify the area with the highest usage of the program. Note that zip codes are not included in the data provided but may be related to the StationId by its Latitude and Longitude. This could be part of your research and may require some coding!

CODE:

```
import pandas as pd
import requests
import json

def Stations072013():

    file_name = '2013-07 - Citi Bike trip data.txt'
    Stations = {}

    StartStationId = [] #5
    StartStationName = [] #6
    StartStationLatitude = [] #7
    StartStationLongitute = [] #8


    with open(file_name) as f:
        for n, line in enumerate(f):
            if n == 0:
                continue
            line_items = line.split('\t')

            stationId = int(line_items[5])
            name = line_items[6]
            latitude = float(line_items[7])
            longitute = line_items[8]
            if stationId not in StartStationId:
                StartStationId.append(stationId)
                StartStationName.append(name)
                StartStationLatitude.append(latitude)
                StartStationLongitute.append(longitute)


    Stations = {
        'StationId': StartStationId,
        'StationName': StartStationName,
        'StationLatitude': StartStationLatitude,
        'StationLongitute': StartStationLongitute
    }

    df = pd.DataFrame(Stations)
```

```
    df1 = df.sort_values(by=['StationId'])
    return df1

def reverse_geo_coding(lat,long):
    geo_json =
requests.get('https://maps.googleapis.com/maps/api/geocode/json?latlng={latitude},{logitute}&k
ey=<MY_API_KEY>.format(latitude=lat,logitute=long))
    geo_api_data=json.loads(geo_json.text)
    zipcode = geo_api_data["results"][0]["address_components"][-1]["long_name"]
    return zipcode


def add_zip_code(df):
    zipcodes = []
    for index, row in df.iterrows():
        lat = row['StationLatitude']
        long = row['StationLongitute']
        zipcodes.append(reverse_geo_coding(lat,long))
    return zipcodes



if __name__ == '__main__':
    df = Stations072013()
    df['ZipCodes'] = add_zip_code(df)
    df.to_csv('StationsZipCode.csv', index=False)
```

OUTPUT:

	StationId	StationName	StationLatitude	StationLongitute	ZipCodes
1	72	W 52 St & 11 Ave	40.76727216	-73.99392888	10019
2	79	Franklin St & W Broadway	40.71911552	-74.00666661	10013
3	82	St James Pl & Pearl St	40.71117416	-74.00016545	10038
4	83	Atlantic Ave & Fort Green…	40.68382604	-73.97632328	11217
5	116	W 17 St & 8 Ave	40.74177603	-74.00149746	10011
6	119	Park Ave & St Edwards St	40.69608941	-73.97803415	11205
7	120	Lexington Ave & Classon A…	40.68676793	-73.95928168	11238
8	127	Barrow St & Hudson St	40.73172428	-74.00674436	10014
9	128	MacDougal St & Prince St	40.72710258	-74.00297088	10012
10	137	E 56 St & Madison Ave	40.761628	-73.972924	10022
11	143	Clinton St & Joralemon St	40.69239502	-73.99337909	11201
12	144	Nassau St & Navy St	40.69839895	-73.98068914	11201

Narrative:

The above output containing the zip code is obtained by reverse geo coding the latitude and longitude. The source code was written in Python programming language.

CODE:

```
SELECT DISTINCT StationsZipCode.ZipCodes AS 'Zip Code', COUNT(*) AS 'Total Number of Trip'

FROM StationsZipCode, CitiBike072013

GROUP BY StationsZipCode.ZipCodes

ORDER BY [Total Number of Trip];
```

OUTPUT:

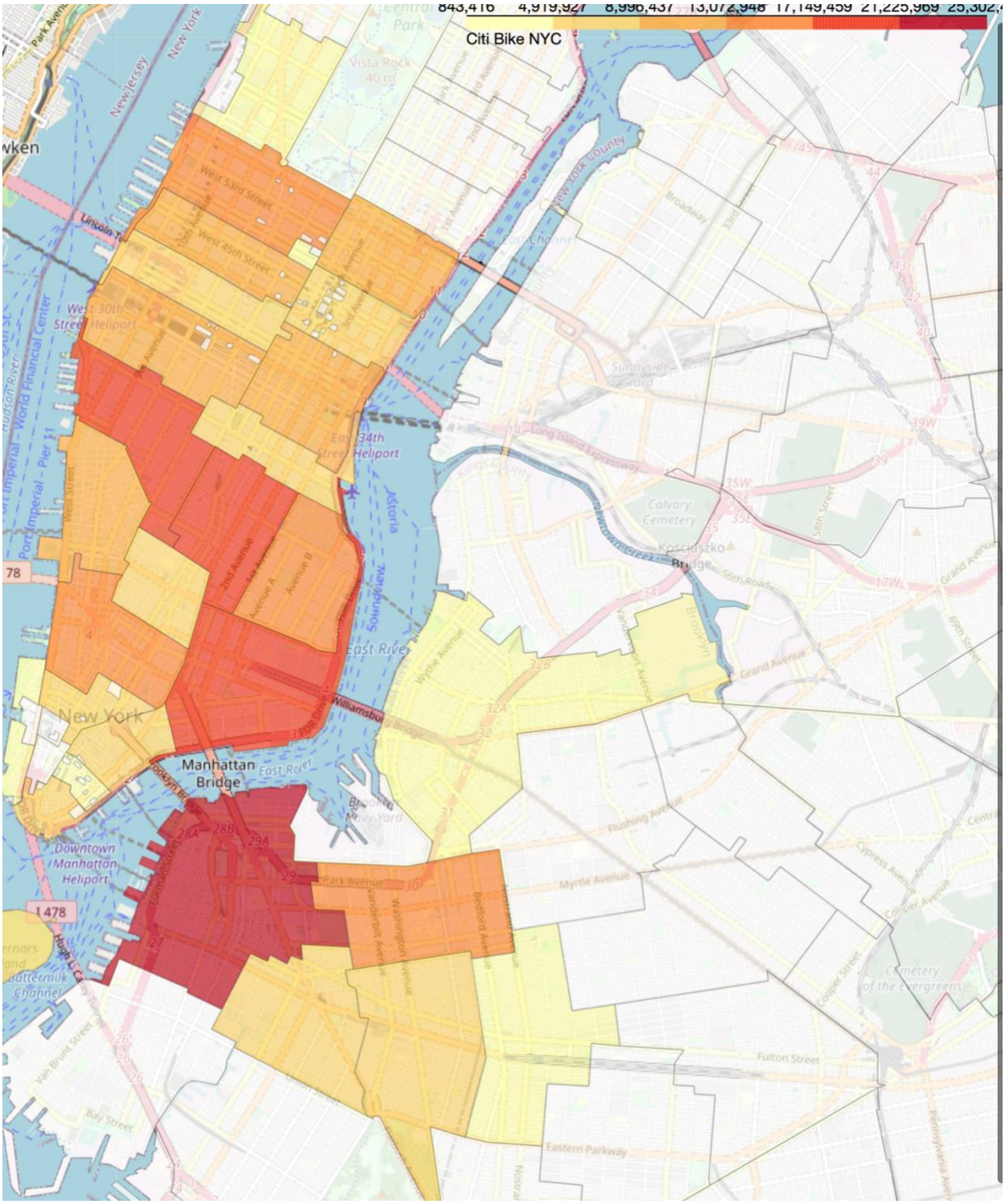
Results Messages

	Zip Code	Total Number of Trip
1	10023	843416
2	10199	843416
3	11243	843416
4	10280	1686832
5	10282	1686832
6	10168	1686832
7	10020	1686832
8	11211	2530248
9	11216	2530248
10	10005	3373664
11	10007	5903912
12	10004	5903912

Narrative:

Using the obtained zip code we calculated the number of trips made in a specific area to find out the highest uses of Citi bike in the give area.

The more visualization of the above output is shown in the figure below:



SQL FULL SOURCE CODE:

```
USE Assignment4;
GO

IF OBJECT_ID('dob.Trips072013') IS NOT NULL DROP View Trips072013
IF OBJECT_ID('dob.StartStation072013') IS NOT NULL DROP View StartStation072013
IF OBJECT_ID('dob.DayStatistics072013') IS NOT NULL DROP View DayStatistics072013
IF OBJECT_ID('dob.GenderStatistics072013') IS NOT NULL DROP View GenderStatistics072013
IF OBJECT_ID('dob.AgeStatistics072013') IS NOT NULL DROP View AgeStatistics072013
IF OBJECT_ID('dob.MostFrequentTrip072013') IS NOT NULL DROP View MostFrequentTrip072013
IF OBJECT_ID('dob.DormantStations072013') IS NOT NULL DROP View DormantStations072013
IF OBJECT_ID('dob.VacantStations072013') IS NOT NULL DROP View VacantStations072013
IF EXISTS(SELECT 1 FROM sys.tables WHERE NAME = 'CitiBike072013') DROP TABLE CitiBike072013
IF EXISTS(SELECT 1 FROM sys.tables WHERE NAME = 'Stations') DROP TABLE Stations
IF EXISTS(SELECT 1 FROM sys.tables WHERE NAME = 'Trips') DROP TABLE Trips
IF EXISTS(SELECT 1 FROM sys.tables WHERE NAME = 'UsageByDay') DROP TABLE UsageByDay
IF EXISTS(SELECT 1 FROM sys.tables WHERE NAME = 'UsageByGender') DROP TABLE UsageByGender
IF EXISTS(SELECT 1 FROM sys.tables WHERE NAME = 'UsageByAge') DROP TABLE UsageByAge
GO


/*----------------------CitiBike Table-------------------------*/

/* Citi Bike statistics for July 2003: Table CitiBike072013 */
CREATE TABLE CitiBike072013(
    TripDuration FLOAT,
    StartTime VARCHAR(30),
    StartDay VARCHAR(9) DEFAULT NULL,
    EndTime VARCHAR(30),
    EndDay VARCHAR(9) DEFAULT NULL,
    StartStationId INT,
    StartStationName VARCHAR(64),
    StartStationLatitude FLOAT,
    StartStationLongitute FLOAT,
    EndStationId INT,
    EndStationName VARCHAR(64),
```

```
    EndStationLatitude FLOAT,
    EndStationLongitute FLOAT,
    BikeId INT,
    UserType VARCHAR(10),
    BirthYear VARCHAR(4),
    Gender INT
);

BULK INSERT CitiBike072013
FROM '/2013-07\ -\ Citi\ Bike\ trip\ data.txt'
WITH (FIELDTERMINATOR = '\t',
      ROWTERMINATOR = '\n',
      FIRSTROW = 2);

/* Pupulate StartDay and EndDay fields */
UPDATE CitiBike072013
   SET StartDay = DATENAME(WEEKDAY, StartTime),
       EndDay = DATENAME(WEEKDAY, EndTime);
GO

/* Display CitiBike072013 data */
SELECT *
FROM CitiBike072013;
GO


/*-------------------------Stations Table--------------------------*/

/* Create Stations Table */
CREATE TABLE Stations (
       Id INT,
       Name VARCHAR(64),
       Latitude FLOAT,
       Longitute FLOAT
);

/* Bike Station data: Partition StartStation072013 */
```

```
WITH StartStation072013 AS (
SELECT  StartStationId AS 'Station Id', StartStationName AS 'Name', StartStationLatitude AS 'Latitude',
StartStationLongitude AS 'Longitude',
      ROW_NUMBER() OVER(PARTITION BY StartStationId
                  ORDER BY StartStationId) AS RowNumber
FROM    CitiBike072013)

/* Insert into Stations Table from the Partition */
INSERT INTO Stations (Id, Name, Latitude, Longitute)
SELECT [Station Id], Name, Latitude, Longitude
FROM StartStation072013
WHERE RowNumber = 1;
GO

/* Display Stations data */
SELECT DISTINCT * FROM Stations
ORDER BY Id;
GO

/*----------------------------Trips Table----------------------------*/

/* Create Trips Table */
CREATE TABLE Trips (
      StationId INT,
      MinTripDuration INT,
      MaxTripDuration INT,
      AvgTripDuration FLOAT,
      NumberOfStartUsers INT,
      NumberOfReturnUsers INT
);

/* Trip statistics partition Trips072013 */
WITH Trips072013 AS(
SELECT S.[Station ID], S.[Minimum Trip Duration], S.[Maximum Trip Duration], S.[Average Trip Duration],
     S.[Number of Start Users], T.[Number of Return Users]
FROM    (SELECT DISTINCT StartStationId AS 'Station ID',
           MIN(TripDuration) OVER(PARTITION BY StartStationId) AS 'Minimum Trip Duration',
```

```
        MAX(TripDuration) OVER(PARTITION BY StartStationId) AS 'Maximum Trip Duration',
        AVG(TripDuration) OVER(PARTITION BY StartStationId) AS 'Average Trip DUration',
        COUNT(*) OVER(PARTITION BY StartStationId) AS 'Number of Start Users'
    From CitiBike072013) S
    INNER JOIN
    (SELECT DISTINCT EndStationId AS 'Station Id',
        COUNT(*) OVER(PARTITION BY EndStationId) AS 'Number of Return Users'
    FROM CitiBike072013) T
    ON S.[Station ID] = T.[Station Id]
)

/* Insert into Trips Table from the Partition */
INSERT INTO Trips (StationId, MinTripDuration, MaxTripDuration, AvgTripDuration, NumberOfStartUsers,
NumberOfReturnUsers)
SELECT *
FROM Trips072013
ORDER BY [Station ID];

/* Display Trips data */
SELECT StationId AS 'Station Id', MinTripDuration AS 'Minimum Trip Duration', MaxTripDuration AS 'Maximum Trip
Duration', AvgTripDuration AS 'Average Trip Duration', NumberOfStartUsers AS 'Number of Start Users',
NumberOfReturnUsers AS 'Number of Return Users'
FROM Trips;
GO


/*-------------------------UsageByDay Table--------------------------*/

/* Create UsageByDay Table */
CREATE TABLE UsageByDay(
    StationId INT,
    NumberUsersWeekday INT,
    NumberUsersWeekend INT
);

/* User Statistics by weekday/weekend: Partition DayStatistics072013 */
WITH DayStatistics072013 AS(
```

```
SELECT R.Id AS 'Station Id', MAX(R.MondayUsers) + MAX(R.TuesdayUsers) + MAX(R.WednesdayUsers) +
MAX(R.ThursdayUsers) + MAX(R.FridayUsers) AS 'Weekday Users',
                    MAX(R.SaturdayUsers) + MAX(R.SundayUsers) AS 'Weekend Users'
FROM    (SELECT StartStationId AS 'Id',
            [MondayUsers] = CASE WHEN StartDay = 'Monday' THEN COUNT(*) OVER(PARTITION BY
StartStationId, StartDay) END,
            [TuesdayUsers] = CASE WHEN StartDay = 'Tuesday' THEN COUNT(*) OVER(PARTITION BY
StartStationId, StartDay) END,
            [WednesdayUsers] = CASE WHEN StartDay = 'Wednesday' THEN COUNT(*) OVER(PARTITION BY
StartStationId, StartDay) END,
            [ThursdayUsers] = CASE WHEN StartDay = 'Thursday' THEN COUNT(*) OVER(PARTITION BY
StartStationId, StartDay) END,
            [FridayUsers] = CASE WHEN StartDay = 'Friday' THEN COUNT(*) OVER(PARTITION BY StartStationId,
StartDay) END,
            [SaturdayUsers] = CASE WHEN StartDay = 'Saturday' THEN COUNT(*) OVER(PARTITION BY
StartStationId, StartDay) END,
            [SundayUsers] = CASE WHEN StartDay = 'Sunday' THEN COUNT(*) OVER(PARTITION BY
StartStationId, StartDay) END
      FROM CitiBike072013) R
GROUP BY R.Id)


/* Insert into UsageByDay Table */
INSERT INTO UsageByDay (StationId, NumberUsersWeekday, NumberUsersWeekend)
SELECT *
FROM DayStatistics072013
ORDER BY [Station Id];

/* Display UsageByDay data */
SELECT StationId AS 'Station Id', NumberUsersWeekday AS 'Weekday Users', NumberUsersWeekend AS
'Weekend Users'
FROM UsageByDay
GO


/*----------------------UsageByGender Table----------------------*/
```

```
/* Create UsageByGender Table */
CREATE TABLE UsageByGender(
	StationId INT,
	NumberMaleUsers INT,
	NumberFemaleUsers INT,
	UnidentifiedUsers INT
);

/* User Statistics by gender: Partition UsageByGender */
WITH GenderStatistics072013 AS(
SELECT R.Id AS 'Station Id', MAX(R.MaleUsers) AS 'Male Users', MAX(R.FemaleUsers) AS 'Female Users',
MAX(R.UnidentifiedUsers) AS 'Unidentified Users'
FROM	(SELECT StartStationId AS 'Id',
		[MaleUsers] = CASE WHEN Gender = 1 THEN COUNT(*) OVER(PARTITION BY StartStationId, Gender)
END,
		[FemaleUsers] = CASE WHEN Gender = 2 THEN COUNT(*) OVER(PARTITION BY StartStationId,
Gender) END,
		[UnidentifiedUsers] = CASE WHEN Gender = 0 THEN COUNT(*) OVER(PARTITION BY StartStationId,
Gender) END
	FROM CitiBike072013) R
GROUP BY R.Id)

INSERT INTO UsageByGender (StationId, NumberMaleUsers, NumberFemaleUsers, UnidentifiedUsers)
SELECT *
FROM GenderStatistics072013
ORDER BY [Station Id];

/* Display UsageByGender data */
SELECT StationId AS 'Station Id', NumberMaleUsers AS 'Male Users', NumberFemaleUsers AS 'Female Users',
UnidentifiedUsers AS 'Unidentified Users'
FROM UsageByGender
ORDER BY [Station Id];
GO

/*--------------------------UsageByAge Table-------------------------*/

/* Create UsageByAge Table */
```

```
CREATE TABLE UsageByAge(
	StationId INT,
	NumberUsersUnder18 INT,
	NumberUsers18To40 INT,
	NumberUsersOver40 INT
);

/* User Statistics by age group: Partition AgeStatistics072013 */
DECLARE @YearStartTime INT = 2013;
WITH AgeStatistics072013 AS (
SELECT S.Id AS 'Station Id', SUM(S.A) AS 'User Under 18 Years Old', SUM(S.B) AS 'User 18-40 Years Old',
SUM(S.C) AS 'Users Over 40 Years Old'
FROM	(SELECT R.Id AS 'Id', MAX(R.UsersUnder18) AS A, MAX(R.Users18To40) AS B, MAX(R.UsersOver40) AS
C
	FROM	(SELECT	StartStationId AS 'Id', BirthYear,
			[UsersUnder18] = CASE WHEN (BirthYear <> '\N' AND (@YearStartTime - CAST(BirthYear AS
INT)) < 18) THEN COUNT(BirthYear) OVER(PARTITION BY StartStationId, BirthYear) END,
			[Users18To40] = CASE WHEN (BirthYear <> '\N' AND (@YearStartTime - CAST(BirthYear AS
INT)) >= 18 AND (@YearStartTime - CAST(BirthYear AS INT)) <= 40) THEN COUNT(BirthYear) OVER(PARTITION
BY StartStationId, BirthYear) END,
			[UsersOver40] = CASE WHEN (BirthYear <> '\N' AND (@YearStartTime - CAST(BirthYear AS INT))
> 40) THEN COUNT(BirthYear) OVER(PARTITION BY StartStationId, BirthYear) END
		FROM CitiBike072013) R
	GROUP BY R.Id, R.BirthYear) S
GROUP BY S.Id)

/* Insert into UsageByAge Table */
INSERT INTO UsageByAge (StationId, NumberUsersUnder18, NumberUsers18To40, NumberUsersOver40)
SELECT *
FROM AgeStatistics072013
ORDER BY [Station Id];

/* Display UsageByAge data */
SELECT StationId AS 'Station Id', NumberUsersUnder18 AS 'Users Under 18 Years Old', NumberUsers18To40 AS
'Users 18-40 Years Old', NumberUsersOver40 AS 'Users Over 40 Years Old'
FROM UsageByAge
GO
```

```
/*---------------------------Most frequent trips----------------------*/

/* Most frequent trips between any two stations: Partition MostFrequentTrip072013 */
WITH MostFrequentTrip072013 AS(
SELECT StartStationId AS 'Start Station', EndStationId AS 'End Station', COUNT(*) AS 'Number of Trips',
      AVG(TripDuration) AS 'Average Trip Duration'
FROM CitiBike072013
GROUP BY StartStationId, EndStationId)

/* Display TOP(50) MostFrequentTrip072017 data */
SELECT TOP(50) *
FROM MostFrequentTrip072013
ORDER BY [Number of Trips] DESC, [Start Station];

/*-----------------------Dormant Station----------------------*/

/* RED FLAG -- Dormant Station: All End Stations that are NOT Start Stations */
WITH DormantStations072013 AS(
SELECT  EndStationId AS 'Dormant Station', EndStationName AS 'Name', EndStationLatitude AS 'Latitude',
       EndStationLongitude AS 'Longitude'
FROM    CitiBike072013
WHERE   EndStationId NOT IN (SELECT StartStationId
                        FROM   CitiBike072013))

/* Display Dormant Station */
SELECT *
FROM DormantStations072013
ORDER BY [Dormant Station];
GO

/*-----------------------Vacant Station----------------------*/

/* RED FLAG -- Vacant Station: All Start Stations that are NOT End Stations will eventually become vacant */
WITH VacantStations072013 AS(
SELECT  StartStationId AS 'Vacant Station', StartStationName AS 'Name', StartStationLatitude AS 'Latitude',
```

```
        StartStationLongitude AS 'Longitude'
FROM    CitiBike072013
WHERE   StartStationId NOT IN (SELECT EndStationId
                        FROM   CitiBike072013))

/* Display Vacant Station */
SELECT *
FROM VacantStations072013
ORDER BY [Vacant Station];
GO


/*---------------Alter Station Table to Add Zip Code----------------------*/

/* Citi Bike statistics for July 2003: Table StationsZipcode */
CREATE TABLE StationsZipCode(
   StationId INT,
   StationName VARCHAR(64),
   StationLatitude FLOAT,
   StationLongitute FLOAT,
   ZipCodes INT
);

BULK INSERT CitiBike072013
FROM '/StationsZipCode.csv'
WITH (FIELDTERMINATOR = ',',
     ROWTERMINATOR = '\n',
     FIRSTROW = 2);
GO

SELECT * FROM StationsZipCode

SELECT DISTINCT StationsZipCode.ZipCodes AS 'Zip Code', COUNT(*) AS 'Total Number of Trip'
FROM StationsZipCode, CitiBike072013
GROUP BY StationsZipCode.ZipCodes
ORDER BY [Total Number of Trip];
GO
```

Python FULL SOURCE CODE:

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

# Libaries
import pandas as pd
import requests
import json
import folium

def Stations072013():

    file_name = '2013-07 - Citi Bike trip data.txt'
    Stations = {}

    StartStationId = [] #5
    StartStationName = [] #6
    StartStationLatitude = [] #7
    StartStationLongitute = [] #8


    with open(file_name) as f:
        for n, line in enumerate(f):
            if n == 0:
                continue
            line_items = line.split('\t')

            stationId = int(line_items[5])
            name = line_items[6]
            latitude = float(line_items[7])
            longitute = line_items[8]
            if stationId not in StartStationId:
                StartStationId.append(stationId)
                StartStationName.append(name)
                StartStationLatitude.append(latitude)
                StartStationLongitute.append(longitute)


    Stations = {
        'StationId': StartStationId,
        'StationName': StartStationName,
        'StationLatitude': StartStationLatitude,
        'StationLongitute': StartStationLongitute
    }

    df = pd.DataFrame(Stations)
    df1 = df.sort_values(by=['StationId'])
    return df1
```

```
def reverse_geo_coding(lat,long):
    geo_json =
requests.get('https://maps.googleapis.com/maps/api/geocode/json?latlng={latit
ude},{logitute}&key=<GOOGLE_GEO_CODER_API_KEY'.format(latitude=lat,logitute=l
ong))
    geo_api_data=json.loads(geo_json.text)
    zipcode = geo_api_data["results"][0]["address_components"][-
1]["long_name"]
    return zipcode


def add_zip_code(df):
    zipcodes = []
    for index, row in df.iterrows():
        lat = row['StationLatitude']
        long = row['StationLongitute']
        zipcodes.append(reverse_geo_coding(lat,long))
    return zipcodes

if __name__ == '__main__':
    df = Stations072013()
    df['ZipCodes'] = add_zip_code(df)
    df.to_csv('StationsZipCode.csv', index=False)

    map = folium.Map(location=[40.693943, -73.985880], default_zoom_start=5)
    nyc_citibike = pd.read_csv('Results.csv')
    nyc_citibike['Zip Code']=nyc_citibike['Zip Code'].astype(str)

    map.choropleth(geo_data="nyc-zip-code-tabulation-areas-polygons.geojson",
                   data=nyc_citibike,
                   columns=['Zip Code', 'Total Number of Trip'],
                   key_on='feature.properties.postalCode', # this path contains
    zipcodes in str type, this zipcodes should match with our ZIP CODE column
                   fill_color='YlOrRd',
    nan_fill_color='white',fill_opacity=0.7, line_opacity=0.2,
                   legend_name='Citi Bike NYC')
    map.save('nyc_bike_map.html')
    map
```