Chicago Micro-mobility Analysis

February 22, 2024

1 Chicago Micro-mobility Analysis (Divvy)

2 1 - Data setup and preliminary analysis

You can fetch the data from the API by using the City of Chicago Data Portal URL (https://dev.socrata.com/foundry/data.cityofchicago.org/fg6s-gzvg). You need to sign up in order to create an APP token and then use your token for authentication to access the data of Divvy trips. Below are the codes you can use for this purpose:

[2]:
```
# First you need to install sodapy
#pip install sodapy
```

[3]:
```
# Then you can achieve data
#import pandas as pd
#from sodapy import Socrata

# paste your copied app token in place of #APP_TOKEN
#client = Socrata("data.austintexas.gov", "#APP_TOKEN")

# First 2000 results, returned as JSON from API / converted to Python list of␣
 ↪dictionaries by sodapy.
# "fg6s-gzvg" is taken from the last part of the URL of API Endpoint
#results = client.get("fg6s-gzvg", limit=2000)

# Convert results to pandas DataFrame
#df = pd.DataFrame.from_records(results)
```

For this exercise, I have downloaded the csv dataset directly from City of Chicago Data Portal (https://data.cityofchicago.org/Transportation/Divvy-Trips/fg6s-gzvg/about_data) and read the file from that csv file

[1]:
```
# The path of input files
```

[7]:
```
# The file of Divvy trips
input_micro_mobility = "Divvy_Trips_20240103.csv"
```

[8]:
```
# The file of city wards that we are going to use further in geopandas
input_boundaries = "Boundaries_-_Wards__2023-__20240103.csv"
```

[5]:
```
import pandas as pd
```

[6]:
```
# Reading csv file
```

[7]:
```
df_micro_mobility = pd.read_csv(input_micro_mobility, on_bad_lines = 'warn',␣
↪sep=',')
```

[8]:
```
df_boundaries = pd.read_csv(input_boundaries, on_bad_lines = 'warn', sep=',')
```

[12]:
```
# Showing a few rows of the tables
```

[13]:
```
# Micro Mobolity
df_micro_mobility.head()
```

[13]:

	TRIP ID	START TIME	STOP TIME	BIKE ID
0	8546790	12/31/2015 05:35:00 PM	12/31/2015 05:44:00 PM	979
1	8546793	12/31/2015 05:37:00 PM	12/31/2015 05:41:00 PM	1932
2	8546795	12/31/2015 05:37:00 PM	12/31/2015 05:40:00 PM	1693
3	8546797	12/31/2015 05:38:00 PM	12/31/2015 05:55:00 PM	3370
4	8546798	12/31/2015 05:38:00 PM	12/31/2015 05:41:00 PM	2563

	TRIP DURATION	FROM STATION ID	FROM STATION NAME	TO STATION ID
0	521	117	Wilton Ave & Belmont Ave	229
1	256	301	Clark St & Schiller St	138
2	134	465	Marine Dr & Ainslie St	251
3	995	333	Ashland Ave & Blackhawk St	198
4	177	48	Larrabee St & Kingsbury St	111

	TO STATION NAME	USER TYPE	GENDER	BIRTH YEAR
0	Southport Ave & Roscoe St	Subscriber	Female	1991.0
1	Clybourn Ave & Division St	Subscriber	Male	1992.0
2	Clarendon Ave & Leland Ave	Subscriber	Female	1987.0
3	Green St (Halsted St) & Madison St	Subscriber	Male	1975.0
4	Sedgwick St & Huron St	Subscriber	Male	1990.0

	FROM LATITUDE	FROM LONGITUDE	FROM LOCATION	TO LATITUDE
0	41.940180	-87.653040	POINT (-87.65304 41.94018)	41.943739
1	41.907993	-87.631501	POINT (-87.631501 41.907993)	41.904613
2	41.971600	-87.650154	POINT (-87.650154 41.9716)	41.967968
3	41.907066	-87.667252	POINT (-87.667252 41.907066)	41.881892
4	41.897764	-87.642884	POINT (-87.642884 41.897764)	41.894666

	TO LONGITUDE	TO LOCATION
0	-87.664020	POINT (-87.66402 41.943739)
1	-87.640552	POINT (-87.640552 41.904613)
2	-87.650001	POINT (-87.650001 41.967968)
3	-87.648789	POINT (-87.648789 41.881892)

```
4    -87.638437  POINT (-87.638437 41.894666)
```

[15]:
```
# Boundaries
df_boundaries.head()
```

[15]:

	Ward	the_geom	objectid	\
0	1	MULTIPOLYGON (((-87.68777205374418 41.92858465…	51	
1	2	MULTIPOLYGON (((-87.62517201063106 41.90399836…	52	
2	5	MULTIPOLYGON (((-87.56030308695986 41.76635735…	55	
3	6	MULTIPOLYGON (((-87.61794321281114 41.77292489…	56	
4	7	MULTIPOLYGON (((-87.54393108740227 41.76029599…	57	

	edit_date	ward_id	globalid	st_area_sh	\
0	06/01/2022	1	{DB2A2A7D-FAF1-42A4-B061-AE18C31A80BB}	6.589346e+07	
1	06/01/2022	2	{88F300F6-D6DF-4337-8DE3-0C2D27A5B338}	3.128511e+07	
2	06/01/2022	5	{0A109A41-9DED-47D7-934E-1EA1CC7EE025}	1.120803e+08	
3	06/01/2022	6	{FD74A999-4BBA-4CE3-BEBB-CC76423037E8}	1.392022e+08	
4	06/01/2022	7	{279FCBD9-EA0D-4FFC-A8CA-EDA2676C0721}	1.414924e+08	

	st_length_
0	61878.821587
1	74175.949239
2	88207.690241
3	80779.851890
4	98906.567862

In this section, I am trying to do some pre-processing and clean the data and also do some exploratory analysis to know better the dataset and basic information hidden behind the micro-mobility data. I am going to answer simple questions such as: - The number of records before and after data cleaning. - When did collection of data start(start-date and time) for micro-mobility dataset and what is the most recent date and time available. - Number of records per year and month in micro-mobility dataset. - identify patterns e.g.,: - Are there more vehicles as years go on ? - Is there some change in usage patterns among different days of the week , months is there a trend – seasonal or weekly ? - Are there any trends based on the gender and age of the user ?

[16]:
```
# Using count() method to see the missing values in schema of the dataframe.
```

[17]:
```
df_micro_mobility.count()
```

[17]:
```
TRIP ID              21242740
START TIME           21242740
STOP TIME            21242740
BIKE ID              21242740
TRIP DURATION        21242740
FROM STATION ID      21242740
FROM STATION NAME    21242740
TO STATION ID        21242740
```

```
TO STATION NAME      21242740
USER TYPE            21242740
GENDER               16347870
BIRTH YEAR           16376026
FROM LATITUDE        21242476
FROM LONGITUDE       21242476
FROM LOCATION        21242476
TO LATITUDE          21242063
TO LONGITUDE         21242063
TO LOCATION          21242063
dtype: int64
```

……

- The number of records before and after data cleaning.

As observed, prior to data cleaning, there are 21,242,740 rows (TRIP ID). However, it is evident that certain columns contain missing values (LATITUDE, LONGITUDE, GENDER, BIRTH YEAR), requiring attention. The most important column for us to calculate OD matrix in further sections are LATITUDE and LONGITUDE, so in this level I am cleaning data based on these columns, however I will clean data based on GENDER and BIRTH YEAR separately. Various approaches exist for handling missing data, and for this exercise, I have decided to address it by removing the rows with missing values.

[18]:
```
# Removing null values
```

[9]:
```
df_MM_clean = df_micro_mobility.dropna(subset=["FROM LATITUDE", "TO LATITUDE"])
```

[20]:
```
df_MM_clean.count()
```

[20]:
```
TRIP ID              21241850
START TIME           21241850
STOP TIME            21241850
BIKE ID              21241850
TRIP DURATION        21241850
FROM STATION ID      21241850
FROM STATION NAME    21241850
TO STATION ID        21241850
TO STATION NAME      21241850
USER TYPE            21241850
GENDER               16347237
BIRTH YEAR           16375343
FROM LATITUDE        21241850
FROM LONGITUDE       21241850
FROM LOCATION        21241850
TO LATITUDE          21241850
TO LONGITUDE         21241850
TO LOCATION          21241850
```

```
dtype: int64
```

Following the data cleaning process, the dataset now consists of 21,241,850 rows for each column, and all necessary columns are non-null except for "GENDER" and "BIRTH YEAR".

……

- When did collection of data start(start-date and time) for micro-mobility dataset and what is the most recent date and time available.

[21]:
```
# Turning "Start Time" column to datetime object
```

[10]:
```
df_MM_clean_time = pd.to_datetime(df_MM_clean['START TIME'], format="%m/%d/%Y␣
↪%I:%M:%S %p")
```

[23]:
```
# Find the minimum value in the "Start Time" column
```

[24]:
```
print("Minimum Start Time:", df_MM_clean_time.min())
```

```
Minimum Start Time: 2013-06-27 01:06:00
```

Collection of data start(start-date and time) for micro-mobility dataset starts from 01:06:00 of 27th of June 2013.

[25]:
```
print("Most Recent Start Time:", df_MM_clean_time.max())
```

```
Most Recent Start Time: 2019-12-31 23:57:17
```

The most recent date and time available in dataset is 23:57:17 of 31st December 2019.

……

- Number of records per year and month in micro-mobility dataset.

[10]:
```
# Create new columns that we need for further analysis (Year, Season, Month,␣
↪Week, etc.)
```

[11]:
```
import datetime as dt
```

[12]:
```
df_MM_clean["YEAR"] = df_MM_clean_time.dt.strftime("%Y")
```

```
/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  from ipykernel.kernelbase import Kernel
```

[13]:
```
df_MM_clean["MONTH"] = df_MM_clean_time.dt.strftime("%m")
```

```
/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  from ipykernel.kernelbase import Kernel
```

[14]:
```python
# Function for seasons
def get_season(month):
    if 3 <= month <= 5:
        return 'Spring'
    elif 6 <= month <= 8:
        return 'Summer'
    elif 9 <= month <= 11:
        return 'Autumn'
    else:
        return 'Winter'
```

[15]:
```python
df_MM_clean["SEASON"] = df_MM_clean_time.dt.month.apply(get_season)
```

```
/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  from ipykernel.kernelbase import Kernel
```

[16]:
```python
df_MM_clean["DAY OF WEEK"] = df_MM_clean_time.dt.strftime("%A")
```

```
/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  from ipykernel.kernelbase import Kernel
```

[20]:
```python
# See the final result
```

[21]:
```python
df_MM_clean.head()
```

[21]:
```
   TRIP ID              START TIME               STOP TIME  BIKE ID  \
0  8546790  12/31/2015 05:35:00 PM  12/31/2015 05:44:00 PM      979
1  8546793  12/31/2015 05:37:00 PM  12/31/2015 05:41:00 PM     1932
2  8546795  12/31/2015 05:37:00 PM  12/31/2015 05:40:00 PM     1693
3  8546797  12/31/2015 05:38:00 PM  12/31/2015 05:55:00 PM     3370
4  8546798  12/31/2015 05:38:00 PM  12/31/2015 05:41:00 PM     2563

   TRIP DURATION  FROM STATION ID           FROM STATION NAME  TO STATION ID  \
0            521              117    Wilton Ave & Belmont Ave            229
1            256              301      Clark St & Schiller St            138
2            134              465      Marine Dr & Ainslie St            251
3            995              333  Ashland Ave & Blackhawk St            198
4            177               48  Larrabee St & Kingsbury St            111

                     TO STATION NAME   USER TYPE  … FROM LATITUDE  \
0           Southport Ave & Roscoe St  Subscriber  …     41.940180
1          Clybourn Ave & Division St  Subscriber  …     41.907993
2          Clarendon Ave & Leland Ave  Subscriber  …     41.971600
3  Green St (Halsted St) & Madison St  Subscriber  …     41.907066
4              Sedgwick St & Huron St  Subscriber  …     41.897764

   FROM LONGITUDE                 FROM LOCATION  TO LATITUDE TO LONGITUDE  \
0      -87.653040    POINT (-87.65304 41.94018)    41.943739   -87.664020
1      -87.631501  POINT (-87.631501 41.907993)    41.904613   -87.640552
2      -87.650154    POINT (-87.650154 41.9716)    41.967968   -87.650001
3      -87.667252  POINT (-87.667252 41.907066)    41.881892   -87.648789
4      -87.642884  POINT (-87.642884 41.897764)    41.894666   -87.638437

                    TO LOCATION  YEAR MONTH  SEASON DAY OF WEEK
0   POINT (-87.66402 41.943739)  2015    12  Winter    Thursday
1  POINT (-87.640552 41.904613)  2015    12  Winter    Thursday
2  POINT (-87.650001 41.967968)  2015    12  Winter    Thursday
3  POINT (-87.648789 41.881892)  2015    12  Winter    Thursday
4  POINT (-87.638437 41.894666)  2015    12  Winter    Thursday

[5 rows x 22 columns]
```

[11]:
```
# Number of trips per year
```

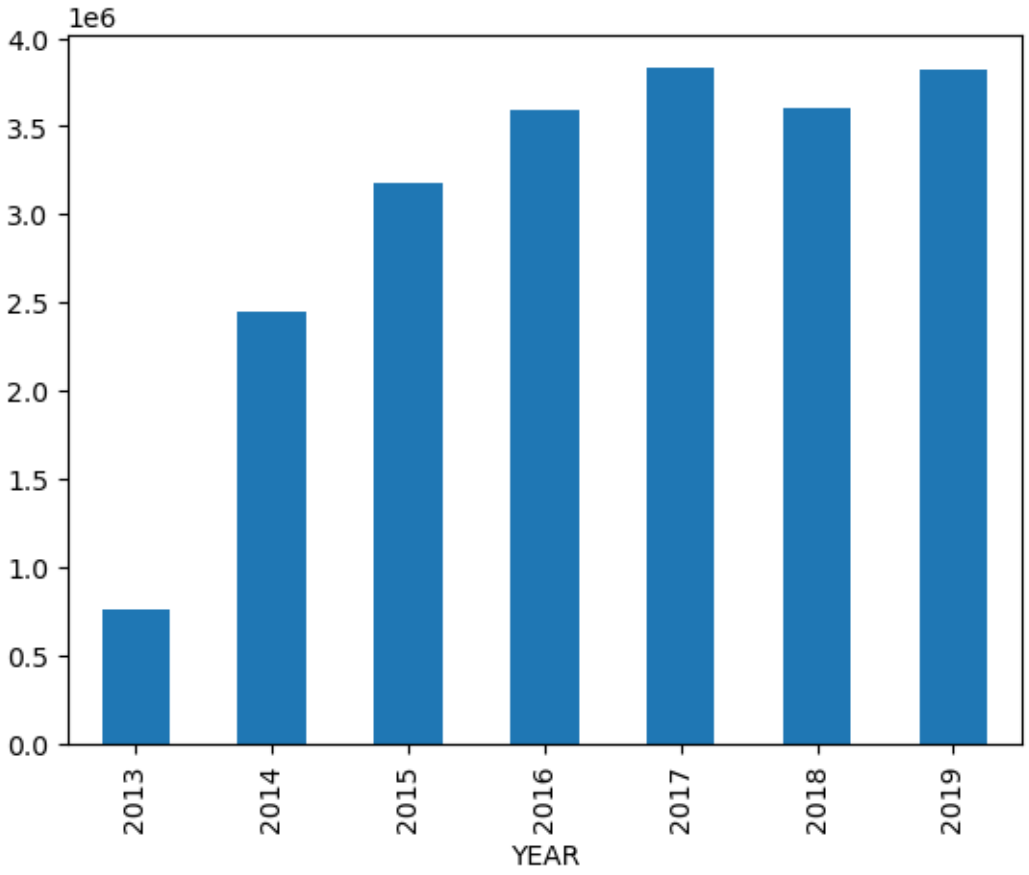
[28]:
```
year_groups = df_MM_clean.groupby(["YEAR"]).size()
```

[37]:
```
print(year_groups)
```

```
YEAR
2013     759788
2014    2454634
2015    3183236
```

```
2016    3595317
2017    3828984
2018    3602180
2019    3817711
dtype: int64
```

[38]:
```
bp_year_groups = year_groups.plot(kind="bar")
```



It is shown that there is an increasing trend for years and this growth was dramatic from 2013 to 2014 and one of the reasons is that our data start from the June 2013. Also, there was a litle decrease from 2017 to 2018.

[12]:
```
# Number of trips per month
```

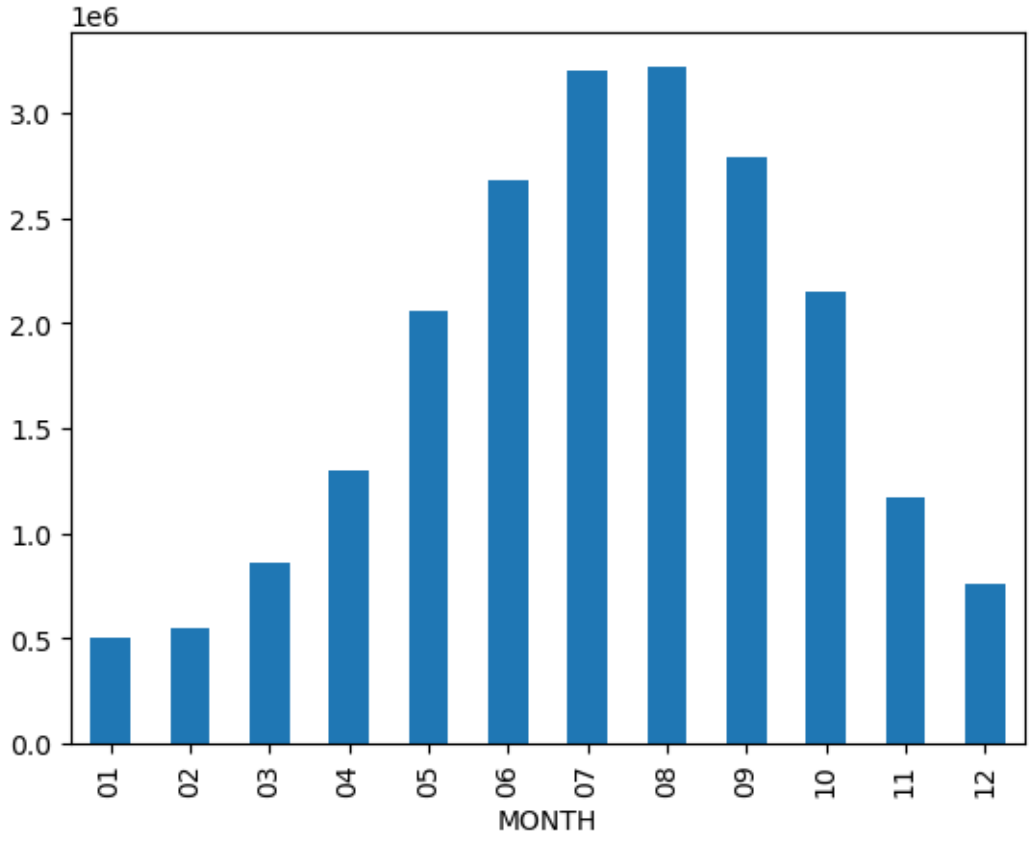
[29]:
```
month_groups = df_MM_clean.groupby(["MONTH"]).size()
```

[41]:
```
print(month_groups)
```

```
MONTH
01     501857
```

```
02     546764
03     856606
04    1300373
05    2058411
06    2682608
07    3204041
08    3220608
09    2792511
10    2149156
11    1173737
12     755178
dtype: int64
```

[42]:
```
bp_month_groups = month_groups.plot(kind="bar")
```



It shows that the majority of trips occur from June to October and it is less in cold months.

[13]:
```
# Number of trips per year and month
```

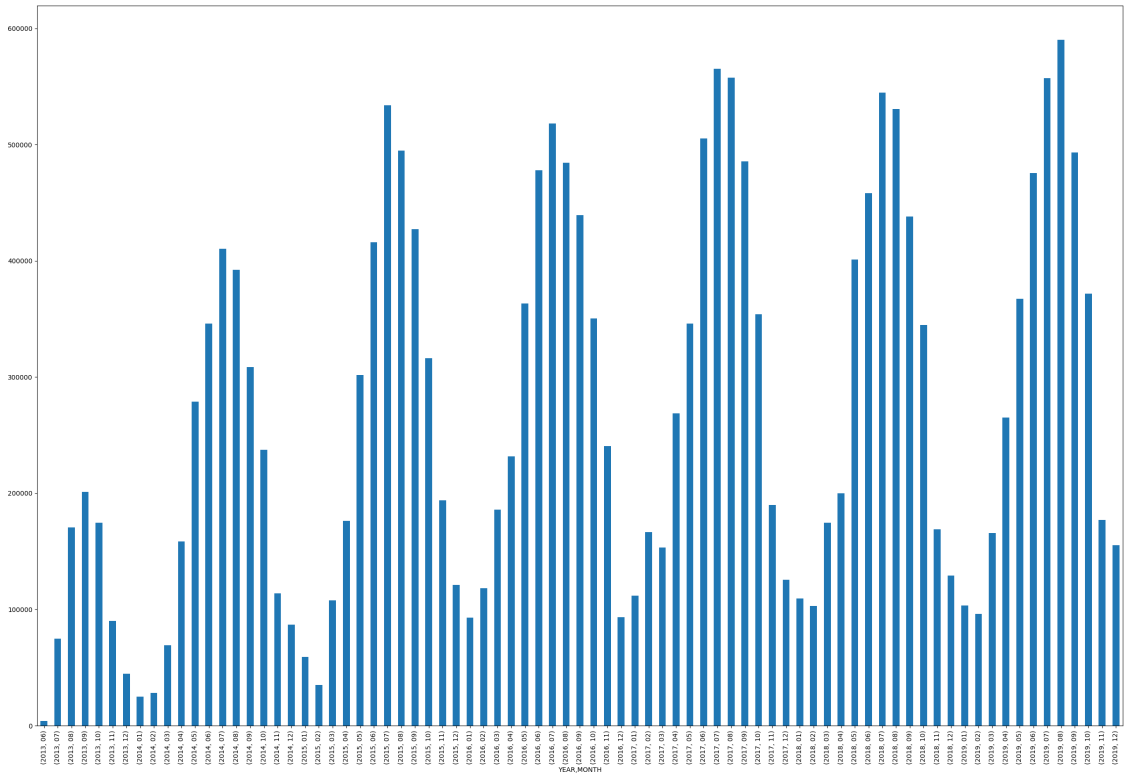
[17]:
```
month_year_groups = df_MM_clean.groupby(["YEAR", "MONTH"]).size()
```

[18]:
```
print(month_year_groups)
```

```
YEAR  MONTH
2013  06          4005
      07         74867
      08        170508
      09        201030
      10        174695
                  …
2019  08        590125
      09        493192
      10        371761
      11        177155
      12        155079
Length: 79, dtype: int64
```

[46]:
```
# Bar plot to show the year-month pattern of data
```

[19]:
```
bp_month_year_groups = month_year_groups.plot(kind="bar", figsize=(30, 20))
```



Number of records per year and month in micro-mobility dataset is shown above. It can be seen that majority of data belong to the middle of each year (from June to October) and generaly every year the trend was increasing.

……..

- identify patterns e.g.,:
- Are there more vehicles as years go on ?
- Is there some change in usage patterns among different days of the week , months is there a trend – seasonal or weekly ?
- Are there any trends based on the gender and age of the user ?

[48]:
```
df_MM_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21241850 entries, 0 to 21242739
Data columns (total 22 columns):
 #   Column             Dtype
---  ------             -----
 0   TRIP ID            int64
 1   START TIME         object
 2   STOP TIME          object
 3   BIKE ID            int64
 4   TRIP DURATION      int64
 5   FROM STATION ID    int64
 6   FROM STATION NAME  object
 7   TO STATION ID      int64
 8   TO STATION NAME    object
 9   USER TYPE          object
 10  GENDER             object
 11  BIRTH YEAR         float64
 12  FROM LATITUDE      float64
 13  FROM LONGITUDE     float64
 14  FROM LOCATION      object
 15  TO LATITUDE        float64
 16  TO LONGITUDE       float64
 17  TO LOCATION        object
 18  YEAR               object
 19  MONTH              object
 20  SEASON             object
 21  DAY OF WEEK        object
dtypes: float64(5), int64(5), object(12)
memory usage: 3.6+ GB
```

[14]:
```
# Comparing used bicycles in different years
```

[50]:
```
vehicle_groups = df_MM_clean.groupby(["YEAR", "BIKE ID"]).size()
```

[51]:
```
print(vehicle_groups)
```

```
YEAR  BIKE ID
2013  1          397
```

```
      2          342
      3          430
      4          352
      5          346
                 …
2019  6929       111
      6931        91
      6941        16
      6942        99
      6946        67
Length: 34712, dtype: int64
```

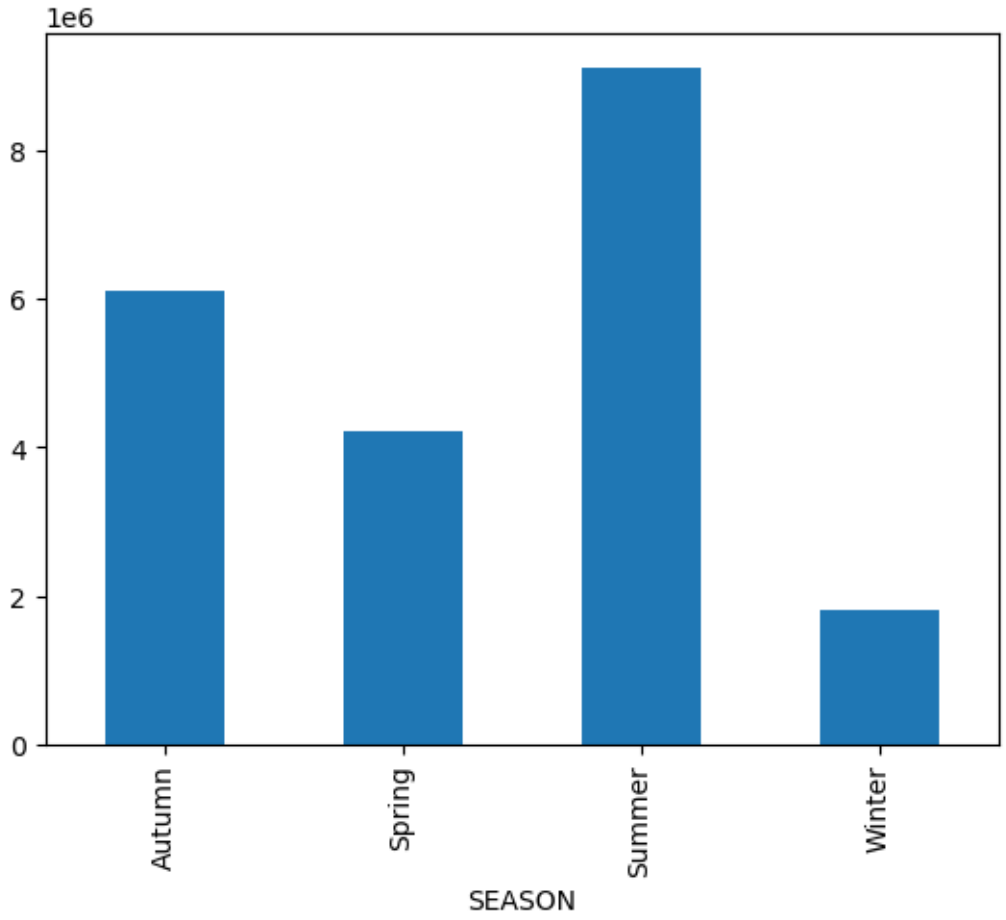
[52]:
```
# Analyzing seasonal trends
```

[18]:
```
season_groups = df_MM_clean.groupby(["SEASON"]).size()
```

[19]:
```
print(season_groups)
```

```
SEASON
Autumn    6115404
Spring    4215390
Summer    9107257
Winter    1803799
dtype: int64
```

[20]:
```
bp_season_groups = season_groups.plot(kind="bar")
```



It shows that most of the trips happened on Summer and after that on Autumn

[53]:
```
vehicle_season_groups = df_MM_clean.groupby(["SEASON", "BIKE ID"]).size()
```

[54]:
```
print(vehicle_season_groups)
```

```
SEASON  BIKE ID
Autumn  1           995
        2          1026
        3          1016
        4           904
        5           803
                   …
Winter  6913         11
        6929         41
        6931         21
        6942         35
        6946          8
```

```
Length: 25647, dtype: int64
```

[56]:
```
vehicle_month_groups = df_MM_clean.groupby(["YEAR", "MONTH", "BIKE ID"]).size()
```

[57]:
```
print(vehicle_month_groups)
```

```
YEAR  MONTH  BIKE ID
2013  06     1           7
             4           3
             5           4
             8           4
             19          2
                        ..
2019  12     6913       11
             6929       41
             6931       21
             6942       35
             6946        8
Length: 324337, dtype: int64
```

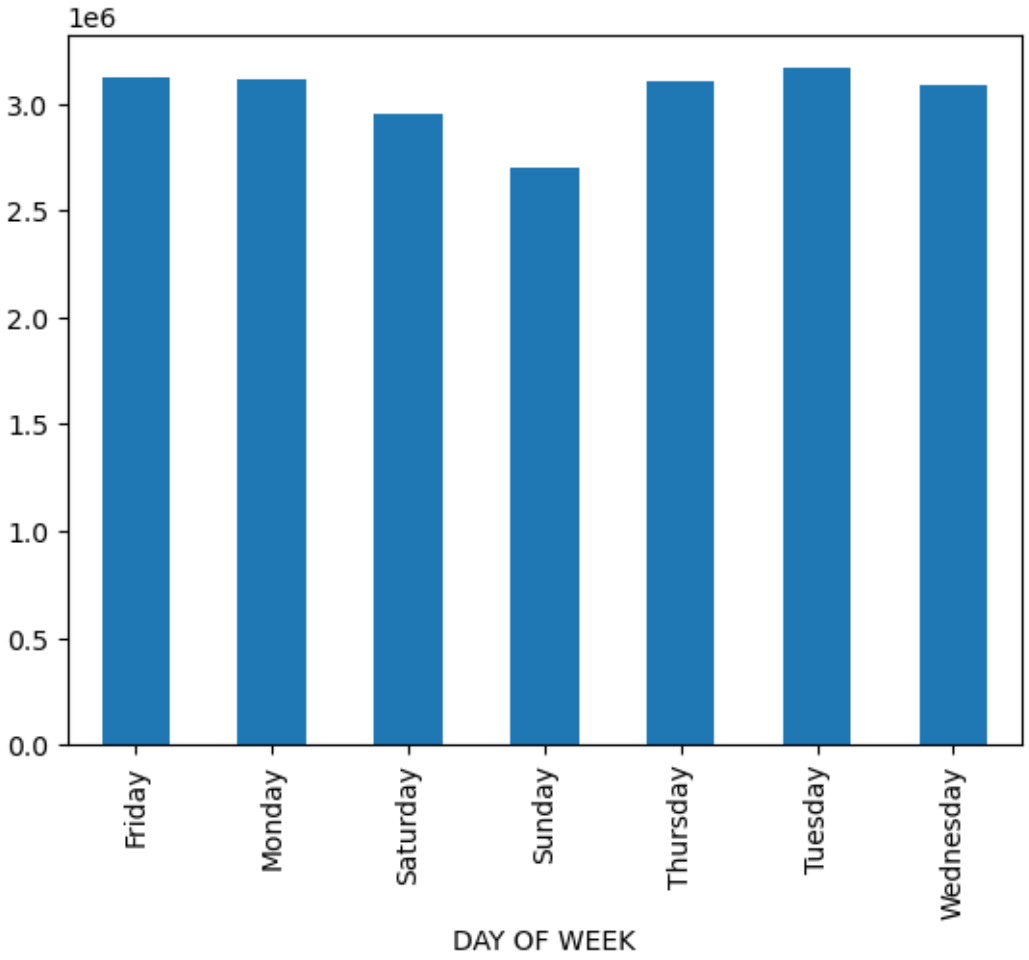
[59]:
```
# Analyzing weekly trends
```

[24]:
```
week_groups = df_MM_clean.groupby(["DAY OF WEEK"]).size()
```

[25]:
```
print(week_groups)
```

```
DAY OF WEEK
Friday       3125963
Monday       3114367
Saturday     2949008
Sunday       2700688
Thursday     3100101
Tuesday      3164302
Wednesday    3087421
dtype: int64
```

[26]:
```
bp_week_groups = week_groups.plot(kind="bar")
```



It shows that most of the trips happened on weekdays but the difference is not very significant

[60]:
```
vehicle_week_groups = df_MM_clean.groupby(["DAY OF WEEK", "BIKE ID"]).size()
```

[61]:
```
print(vehicle_week_groups)
```

```
DAY OF WEEK  BIKE ID
Friday       1          634
             2          516
             3          631
             4          524
             5          540
                        ...
Wednesday    6913         9
             6929        16
             6931        17
             6942        18
```

```
              6946        13
Length: 45556, dtype: int64
```

……

[13]:
```
# Now I am going to clean the data again based on GENDER and BIRTH YEAR
df_MM_clean_GB = df_MM_clean.dropna(axis=0)
```

[65]:
```
df_MM_clean_GB.count()
```

[65]:
```
TRIP ID              16346709
START TIME           16346709
STOP TIME            16346709
BIKE ID              16346709
TRIP DURATION        16346709
FROM STATION ID      16346709
FROM STATION NAME    16346709
TO STATION ID        16346709
TO STATION NAME      16346709
USER TYPE            16346709
GENDER               16346709
BIRTH YEAR           16346709
FROM LATITUDE        16346709
FROM LONGITUDE       16346709
FROM LOCATION        16346709
TO LATITUDE          16346709
TO LONGITUDE         16346709
TO LOCATION          16346709
YEAR                 16346709
MONTH                16346709
SEASON               16346709
DAY OF WEEK          16346709
dtype: int64
```

Now there are 16,346,709 rows for all the columns.

[66]:
```
df_MM_clean_GB['GENDER'].value_counts()
```

[66]:
```
Male      12234473
Female     4112236
Name: GENDER, dtype: int64
```

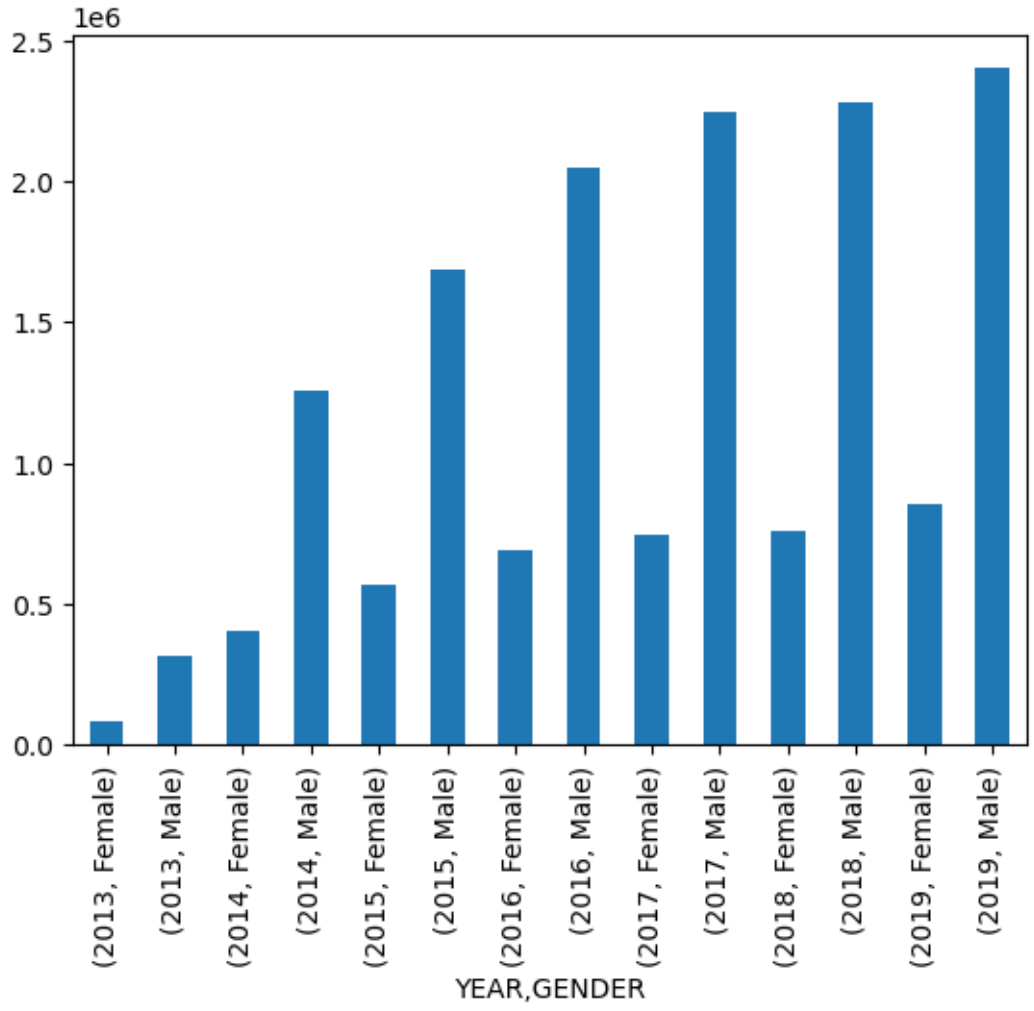
[67]:
```
gender_year_groups = df_MM_clean_GB.groupby(["YEAR", "GENDER"]).size()
```

[68]:
```
print(gender_year_groups)
```

```
YEAR  GENDER
2013  Female      84450
      Male       318459
```

```
2014  Female     407621
      Male      1255655
2015  Female     567305
      Male      1685970
2016  Female     689757
      Male      2047132
2017  Female     746633
      Male      2245291
2018  Female     758533
      Male      2281309
2019  Female     857937
      Male      2400657
dtype: int64
```

[69]:
```
bp_gender_year_groups = gender_year_groups.plot(kind="bar")
```



It is shown that in general through these years the number of trips increased but the share of males have been always way more than females

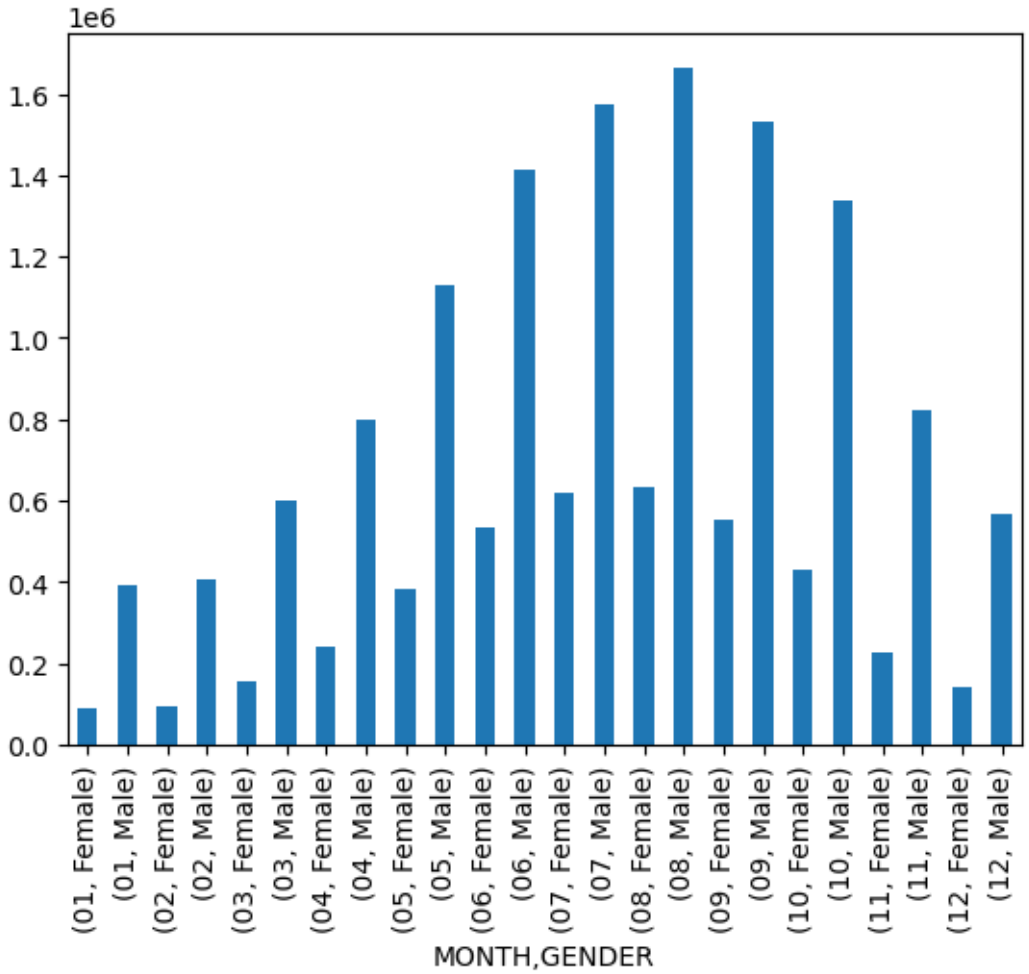
[70]:
```
# Evaluating the gender trends within months
```

[71]:
```
gender_month_groups = df_MM_clean_GB.groupby(["MONTH", "GENDER"]).size()
```

[72]:
```
print(gender_month_groups)
```

```
MONTH  GENDER
01     Female      89020
       Male       392150
02     Female      96448
       Male       407178
03     Female     157775
       Male       598513
04     Female     239797
       Male       798489
05     Female     381690
       Male      1128445
06     Female     534988
       Male      1415071
07     Female     621202
       Male      1576116
08     Female     633053
       Male      1663349
09     Female     555491
       Male      1531561
10     Female     432367
       Male      1336743
11     Female     229574
       Male       820540
12     Female     140831
       Male       566318
dtype: int64
```

[73]:
```
bp_gender_month_groups = gender_month_groups.plot(kind="bar")
```



The same pattern is here for both males and females, warmer months are more delightful to use these modes of transport.

……

[74]:
```
# Calculating the minimum and maximum of "BIRTH YEAR" in order to find out the␣
↪data
```

[75]:
```
df_MM_clean_GB['BIRTH YEAR'].min()
```

[75]: 1759.0

[76]:
```
df_MM_clean_GB['BIRTH YEAR'].max()
```

[76]: 2017.0

[77]:
```
# Check the "birth year" column to find out "bad data"
```

[78]:
```
df_MM_clean_GB['BIRTH YEAR'].value_counts()
```

[78]:
```
1989.0    928536
1988.0    850181
1990.0    837904
1987.0    823726
1986.0    789516
           ...
1925.0         2
1895.0         1
2005.0         1
1759.0         1
1790.0         1
Name: BIRTH YEAR, Length: 103, dtype: int64
```

It is shown that there are some data which are obviously incorrect such as 1790. So, I am getting rid of these data.

[14]:
```
# Drop incorrect rows
df_MM_clean_B = df_MM_clean_GB[df_MM_clean_GB['BIRTH YEAR'] >= 1925]
```

[80]:
```
df_MM_clean_B.count()
```

[80]:
```
TRIP ID              16340176
START TIME           16340176
STOP TIME            16340176
BIKE ID              16340176
TRIP DURATION        16340176
FROM STATION ID      16340176
FROM STATION NAME    16340176
TO STATION ID        16340176
TO STATION NAME      16340176
USER TYPE            16340176
GENDER               16340176
BIRTH YEAR           16340176
FROM LATITUDE        16340176
FROM LONGITUDE       16340176
FROM LOCATION        16340176
TO LATITUDE          16340176
TO LONGITUDE         16340176
TO LOCATION          16340176
YEAR                 16340176
MONTH                16340176
SEASON               16340176
DAY OF WEEK          16340176
dtype: int64
```

[81]:
```
df_MM_clean_B['BIRTH YEAR'].min()
```

[81]:
```
1925.0
```

Now we can say that the minimum of BIRTH YEAR is 1925 and the maximum of it is 2017.

[ ]:
```
df_MM_clean_B = df_MM_clean_B.dropna(subset=['BIRTH YEAR'])
```

[15]:
```
# Convert BIRTH YEAR and YEAR to integer
df_MM_clean_B['BIRTH YEAR'] = df_MM_clean_B['BIRTH YEAR'].astype('int')
df_MM_clean_B['YEAR'] = df_MM_clean_B['YEAR'].astype('int')
```

```
/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  from ipykernel.ipkernel import IPythonKernel
/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[16]:
```
# Create age groups based on the age distribution of users
df_MM_clean_B['age'] = df_MM_clean_B['YEAR'] - df_MM_clean_B['BIRTH YEAR']
```

```
/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  from ipykernel.ipkernel import IPythonKernel
```

[17]:
```
# Define age groups
age_bins = [0, 18, 30, 40, 50, 60, float('inf')]  # Define age bins/ranges
age_labels = ['0-18', '19-30', '31-40', '41-50', '51-60', '61+']  # Define␣
 ↪corresponding labels
```

[18]:
```
# Create 'age group' column using pd.cut
df_MM_clean_B['age group'] = pd.cut(df_MM_clean_B['age'], bins=age_bins,␣
↪labels=age_labels, right=False)
```

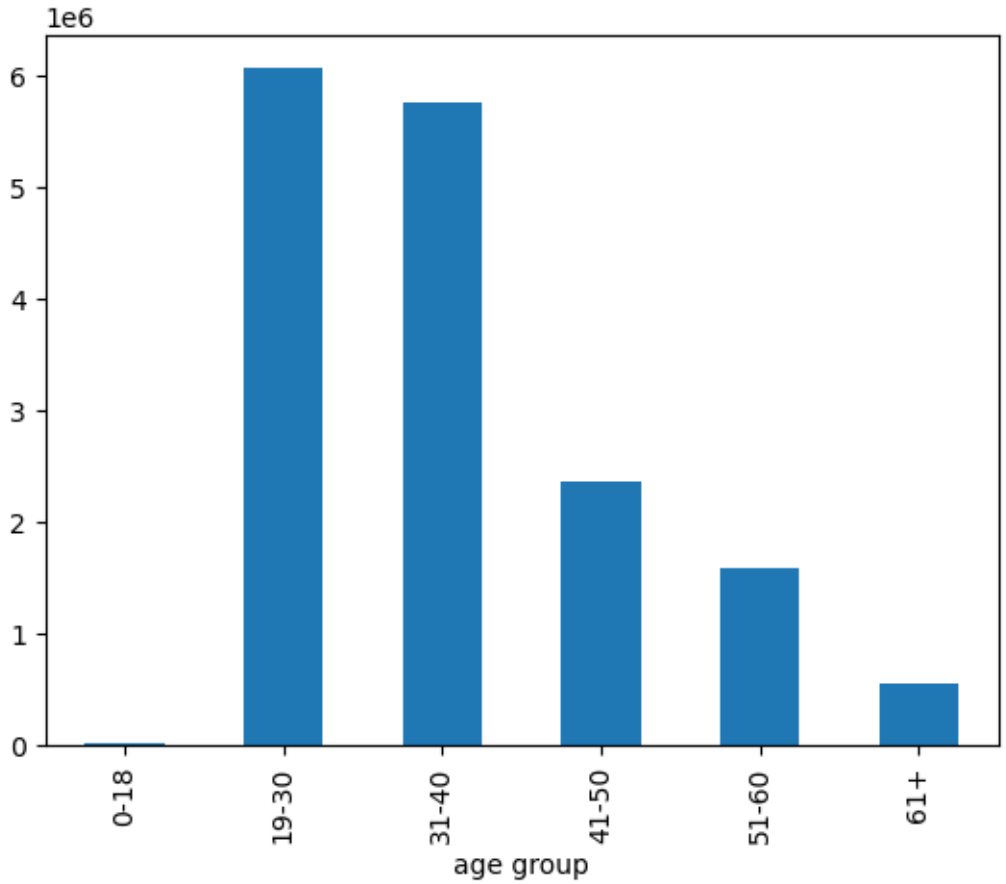
```
/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  from ipykernel.ipkernel import IPythonKernel
```

[19]:
```
age_groups = df_MM_clean_B.groupby("age group").size()
```

[20]:
```
print(age_groups)
```

```
age group
0-18       15967
19-30    6058860
31-40    5759775
41-50    2366332
51-60    1584683
61+       554559
dtype: int64
```

[21]:
```
bp_age_groups = age_groups.plot(kind="bar")
```



It can be seen that the majority of users belong to the age group between 19-40 years old.

[89]:
```
# Evaluating the age trend within years
```

[90]:
```
age_year_groups = df_MM_clean_B.groupby(["YEAR", "age group"]).size()
```

[91]:
```
print(age_year_groups)
```

```
YEAR  BIRTH YEAR RANGE
2013  1925-1936               71
      1936-1947             1961
      1947-1957            17987
      1957-1967            52476
      1967-1977            86489
                             ...
2019  1967-1977           363921
      1977-1987           816792
      1987-1997          1632355
      1997-2007           125920
```

```
      2007-2017                  5
Length: 63, dtype: int64
```

[ ]:
```
bp_age_year_groups = age_year_groups.plot(kind="bar", figsize=(30, 20))
```

It is observed that the same pattern exists for each year.

---

3 2 - OD Matrices

In this section, I am going to do these tasks: - Associate each trip in the dataset to an origin and destination ward by combining the trip information with the information about the wards. Check which wards the FROM_LOCATION and TO_LOCATION fields belong to. - Compute then the O-D matrix, i.e., the number of bookings starting in ward i and ending in ward j. - Prepare OD matrices for different years and different age groups(3 OD matrices for each age-group of 3 consecutive years). Are there any periodicity or trends noticed? Is there a difference between the OD matrices for different age groups? - Based on observation, visualise selected OD matrices that show some trends/periodicity on a map. - Create a flowmap for the OD matrices.

[15]:
```
# First I installed geopandas in order to do spatial analysis and determine␣
↪each trip point (from/to) is within which ward of Chicago based on latitude␣
↪and longitude
```

[14]:
```
pip install geopandas
```

```
Collecting geopandas
  Using cached geopandas-0.10.2-py2.py3-none-any.whl (1.0 MB)
Collecting shapely>=1.6
  Using cached
shapely-2.0.2-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.4 MB)
Collecting fiona>=1.8
  Using cached fiona-1.9.5-cp37-cp37m-manylinux2014_x86_64.whl (15.6 MB)
Collecting pyproj>=2.2.0
  Using cached pyproj-3.2.1-cp37-cp37m-manylinux2010_x86_64.whl (6.3 MB)
Requirement already satisfied: pandas>=0.25.0 in /opt/conda/lib/python3.7/site-
packages (from geopandas) (1.3.5)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.7/site-
packages (from fiona>=1.8->geopandas) (59.8.0)
Collecting cligj>=0.5
  Using cached cligj-0.7.2-py3-none-any.whl (7.1 kB)
Requirement already satisfied: attrs>=19.2.0 in /opt/conda/lib/python3.7/site-
packages (from fiona>=1.8->geopandas) (22.1.0)
Requirement already satisfied: importlib-metadata in
/opt/conda/lib/python3.7/site-packages (from fiona>=1.8->geopandas) (4.11.4)
Requirement already satisfied: click~=8.0 in /opt/conda/lib/python3.7/site-
packages (from fiona>=1.8->geopandas) (8.0.4)
Collecting click-plugins>=1.0
```

```
  Using cached click_plugins-1.1.1-py2.py3-none-any.whl (7.5 kB)
Requirement already satisfied: certifi in /opt/conda/lib/python3.7/site-packages
(from fiona>=1.8->geopandas) (2023.7.22)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages
(from fiona>=1.8->geopandas) (1.16.0)
Requirement already satisfied: python-dateutil>=2.7.3 in
/opt/conda/lib/python3.7/site-packages (from pandas>=0.25.0->geopandas) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /opt/conda/lib/python3.7/site-
packages (from pandas>=0.25.0->geopandas) (2022.4)
Requirement already satisfied: numpy>=1.17.3 in /opt/conda/lib/python3.7/site-
packages (from pandas>=0.25.0->geopandas) (1.21.6)
Requirement already satisfied: typing-extensions>=3.6.4 in
/opt/conda/lib/python3.7/site-packages (from importlib-
metadata->fiona>=1.8->geopandas) (4.4.0)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.7/site-
packages (from importlib-metadata->fiona>=1.8->geopandas) (3.9.0)
Installing collected packages: shapely, pyproj, cligj, click-plugins, fiona,
geopandas
Successfully installed click-plugins-1.1.1 cligj-0.7.2 fiona-1.9.5
geopandas-0.10.2 pyproj-3.2.1 shapely-2.0.2
Note: you may need to restart the kernel to use updated packages.
```

[11]:
```
pip install rtree
```

```
Requirement already satisfied: rtree in /opt/conda/lib/python3.7/site-packages
(1.0.1)
Requirement already satisfied: typing-extensions>=3.7 in
/opt/conda/lib/python3.7/site-packages (from rtree) (4.4.0)
Note: you may need to restart the kernel to use updated packages.
```

[15]:
```
pip install pygeos
```

```
Collecting pygeos
  Using cached
pygeos-0.14-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.1 MB)
Requirement already satisfied: numpy>=1.13 in /opt/conda/lib/python3.7/site-
packages (from pygeos) (1.21.6)
Installing collected packages: pygeos
Successfully installed pygeos-0.14
Note: you may need to restart the kernel to use updated packages.
```

[16]:
```
# In this part, because the size of the file is too much large and I␣
↪encountered memory issues for spatial joining, I decided to select only 3␣
↪years to work on.
```

[92]:
```
df_MM_clean_2016 = df_MM_clean[df_MM_clean['YEAR'] == '2016']
```

[14]:
```
import geopandas as gpd
from shapely import wkt
```

```
/opt/conda/lib/python3.7/site-packages/geopandas/_compat.py:115: UserWarning:
The Shapely GEOS version (3.11.2-CAPI-1.17.2) is incompatible with the GEOS
version PyGEOS was compiled with (3.10.4-CAPI-1.16.2). Conversions between both
will be slow.
  shapely_geos_version, geos_capi_version_string
```

[93]:
```
# Convert 'the_geom' in wards_chicago from WKT(Well Known text) to shapely␣
 ↪objects
# 'the_geom' column contains MULTIPOLYGON data in text format
df_boundaries['the_geom'] = df_boundaries['the_geom'].apply(wkt.loads)
wards_gdf = gpd.GeoDataFrame(df_boundaries, geometry='the_geom')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_4041/1687105298.py in <module>
      1 # Convert 'the_geom' in wards_chicago from WKT(Well Known text) to␣
 ↪shapely objects
      2 # 'the_geom' column contains MULTIPOLYGON data in text format
----> 3 df_boundaries['the_geom'] = df_boundaries['the_geom'].apply(wkt.loads)
      4 wards_gdf = gpd.GeoDataFrame(df_boundaries, geometry='the_geom')

/opt/conda/lib/python3.7/site-packages/pandas/core/series.py in apply(self,␣
 ↪func, convert_dtype, args, **kwargs)
   4355         dtype: float64
   4356         """
-> 4357         return SeriesApply(self, func, convert_dtype, args, kwargs).
 ↪apply()
   4358
   4359     def _reduce(

/opt/conda/lib/python3.7/site-packages/pandas/core/apply.py in apply(self)
   1041             return self.apply_str()
   1042
-> 1043         return self.apply_standard()
   1044
   1045     def agg(self):

/opt/conda/lib/python3.7/site-packages/pandas/core/apply.py in␣
 ↪apply_standard(self)
   1099                     values,
   1100                     f,  # type: ignore[arg-type]
-> 1101                     convert=self.convert_dtype,
   1102                 )
   1103
```

```
/opt/conda/lib/python3.7/site-packages/pandas/_libs/lib.pyx in pandas._libs.lib
 ↪map_infer()

/opt/conda/lib/python3.7/site-packages/shapely/wkt.py in loads(data)
     20     Shapely geometry object
     21     """
---> 22     return shapely.from_wkt(data)
     23
     24

/opt/conda/lib/python3.7/site-packages/shapely/io.py in from_wkt(geometry,␣
 ↪on_invalid, **kwargs)
    280     invalid_handler = np.uint8(DecodingErrorOptions.
 ↪get_value(on_invalid))
    281
--> 282     return lib.from_wkt(geometry, invalid_handler, **kwargs)
    283
    284

TypeError: Expected bytes or string, got MultiPolygon
```

[94]:
```
# Convert 'FROM LOCATION' and 'TO LOCATION' in df_chicago from WKT to shapely␣
 ↪Point objects
# These columns contain POINT data in text format
df_MM_clean_2016['from_point'] = df_MM_clean_2016['FROM LOCATION'].apply(wkt.
 ↪loads)
df_from_gdf_2016 = gpd.GeoDataFrame(df_MM_clean_2016, geometry='from_point')
```

```
/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[95]:
```
df_MM_clean_2016['to_point'] = df_MM_clean_2016['TO LOCATION'].apply(wkt.loads)
df_to_gdf_2016 = gpd.GeoDataFrame(df_MM_clean_2016, geometry='to_point')
```

```
/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
```

```
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  from ipykernel.kernelbase import Kernel
```

[96]:
```
# Ensure the CRS for both GeoDataFrames are the same
# This can be done by setting the CRS(Co-ordinate Reference System) of␣
 ↪df_from_gdf and df_to_gdf to match that of wards_gdf

df_from_gdf_2016 = df_from_gdf_2016.set_crs('EPSG:4326',␣
 ↪inplace=True,allow_override = True)
df_to_gdf_2016 = df_to_gdf_2016.set_crs('EPSG:4326',␣
 ↪inplace=True,allow_override = True)
wards_gdf = wards_gdf.set_crs('EPSG:4326', inplace=True,allow_override = True)
```

[ ]:
```
# Perform spatial joins
# Join df_from_gdf and df_to_gdf with wards_gdf to find the corresponding wards
from_joined_2016 = gpd.sjoin(df_from_gdf_2016, wards_gdf, how="left",␣
 ↪op='within')
```

```
/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3472:
FutureWarning: The `op` parameter is deprecated and will be removed in a future
release. Please use the `predicate` parameter instead.
  if (await self.run_code(code, result,  async_=asy)):
```

[34]:
```
to_joined_2016 = gpd.sjoin(df_to_gdf_2016, wards_gdf, how="left", op='within')
```

```
/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3472:
FutureWarning: The `op` parameter is deprecated and will be removed in a future
release. Please use the `predicate` parameter instead.
  if (await self.run_code(code, result,  async_=asy)):
```

[35]:
```
# Add the ward information back to the original df_chicago DataFrame
df_MM_clean_2016['FROM WARD'] = from_joined_2016['Ward']  # Replace 'Ward' with␣
 ↪the actual column name in wards_gdf
df_MM_clean_2016['TO WARD'] = to_joined_2016['Ward']      # Replace 'Ward' with␣
 ↪the actual column name in wards_gdf
```

```
/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  from ipykernel.ipkernel import IPythonKernel
/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

[36]:
```
df_MM_clean_2016.to_csv("df_MM_clean_2016", index=False)
```

[30]:
```
# Display the first few rows of the updated df_chicago to check the results
df_MM_clean_2016.head()
```

[30]:

	TRIP ID	START TIME	STOP TIME	BIKE ID	\
6397658	10000000	06/09/2016 06:24:00 PM	06/09/2016 06:33:00 PM	5	
6397659	10000001	06/09/2016 06:24:00 PM	06/09/2016 06:27:00 PM	4201	
6397660	10000002	06/09/2016 06:25:00 PM	06/09/2016 06:41:00 PM	1976	
6397661	10000003	06/09/2016 06:25:00 PM	06/09/2016 06:33:00 PM	5354	
6397662	10000005	06/09/2016 06:25:00 PM	06/09/2016 06:38:00 PM	4365	

	TRIP DURATION	FROM STATION ID	FROM STATION NAME	\
6397658	516	26	McClurg Ct & Illinois St	
6397659	163	238	Ravenswood Ave & Montrose Ave (*)	
6397660	987	91	Clinton St & Washington Blvd	
6397661	518	310	Damen Ave & Charleston St	
6397662	813	20	Sheffield Ave & Kingsbury St	

	TO STATION ID	TO STATION NAME	USER TYPE	…	\
6397658	90	Millennium Park	Subscriber	…	
6397659	234	Clark St & Montrose Ave	Subscriber	…	
6397660	289	Wells St & Concord Ln	Subscriber	…	
6397661	327	Sheffield Ave & Webster Ave	Subscriber	…	
6397662	153	Southport Ave & Wellington Ave	Subscriber	…	

	FROM LONGITUDE	FROM LOCATION	TO LATITUDE	\
6397658	-87.617300	POINT (-87.6173 41.89102)	41.881032	
6397659	-87.674365	POINT (-87.674365 41.961615)	41.961588	
6397660	-87.641170	POINT (-87.64117 41.88338)	41.912133	
6397661	-87.677855	POINT (-87.677855 41.920082)	41.921540	
6397662	-87.653106	POINT (-87.653106 41.910522)	41.935733	

	TO LONGITUDE	TO LOCATION	YEAR	\
6397658	-87.624084	POINT (-87.624084 41.881032)	2016	
6397659	-87.666036	POINT (-87.666036 41.961588)	2016	
6397660	-87.634656	POINT (-87.634656 41.912133)	2016	
6397661	-87.653818	POINT (-87.653818 41.92154)	2016	
6397662	-87.663576	POINT (-87.663576 41.935733)	2016	

	from_point	to_point	FROM WARD	\
6397658	POINT (-87.61730 41.89102)	POINT (-87.62408 41.88103)	42.0	

```
6397659  POINT (-87.67436 41.96162)  POINT (-87.66604 41.96159)      47.0
6397660  POINT (-87.64117 41.88338)  POINT (-87.63466 41.91213)      42.0
6397661  POINT (-87.67785 41.92008)  POINT (-87.65382 41.92154)      32.0
6397662  POINT (-87.65311 41.91052)  POINT (-87.66358 41.93573)      27.0

        TO WARD
6397658    42.0
6397659    47.0
6397660     2.0
6397661    43.0
6397662    32.0

[5 rows x 23 columns]
```

[14]:
```
df_MM_clean_2017 = df_MM_clean[df_MM_clean['YEAR'] == '2017']
```

[18]:
```
# Convert 'FROM LOCATION' and 'TO LOCATION' in df_chicago from WKT to shapely␣
 ↪Point objects
# These columns contain POINT data in text format
df_MM_clean_2017['from_point'] = df_MM_clean_2017['FROM LOCATION'].apply(wkt.
 ↪loads)
df_from_gdf_2017 = gpd.GeoDataFrame(df_MM_clean_2017, geometry='from_point')
df_MM_clean_2017['to_point'] = df_MM_clean_2017['TO LOCATION'].apply(wkt.loads)
df_to_gdf_2017 = gpd.GeoDataFrame(df_MM_clean_2017, geometry='to_point')
```

```
/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  def init(spark_master="local"):
```

[19]:
```
# Ensure the CRS for both GeoDataFrames are the same
# This can be done by setting the CRS(Co-ordinate Reference System) of␣
 ↪df_from_gdf and df_to_gdf to match that of wards_gdf
df_from_gdf_2017 = df_from_gdf_2017.set_crs('EPSG:4326',␣
 ↪inplace=True,allow_override = True)
```

```
df_to_gdf_2017 = df_to_gdf_2017.set_crs('EPSG:4326',␣
 ↪inplace=True,allow_override = True)
wards_gdf = wards_gdf.set_crs('EPSG:4326', inplace=True,allow_override = True)
```

[20]:
```
# Perform spatial joins
# Join df_from_gdf and df_to_gdf with wards_gdf to find the corresponding wards
from_joined_2017 = gpd.sjoin(df_from_gdf_2017, wards_gdf, how="left",␣
 ↪op='within')
to_joined_2017 = gpd.sjoin(df_to_gdf_2017, wards_gdf, how="left", op='within')
```

```
/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3472:
FutureWarning: The `op` parameter is deprecated and will be removed in a future
release. Please use the `predicate` parameter instead.
  if (await self.run_code(code, result,  async_=asy)):
/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3472:
FutureWarning: The `op` parameter is deprecated and will be removed in a future
release. Please use the `predicate` parameter instead.
  if (await self.run_code(code, result,  async_=asy)):
```

[21]:
```
# Add the ward information back to the original df_chicago DataFrame
df_MM_clean_2017['FROM WARD'] = from_joined_2017['Ward']  # Replace 'Ward' with␣
 ↪the actual column name in wards_gdf
df_MM_clean_2017['TO WARD'] = to_joined_2017['Ward']      # Replace 'Ward' with␣
 ↪the actual column name in wards_gdf
```

```
/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  from ipykernel.ipkernel import IPythonKernel
/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[22]:
```
df_MM_clean_2017.to_csv("df_MM_clean_2017", index=False)
```

[24]:
```
# Display the first few rows of the updated df_chicago to check the results
df_MM_clean_2017.head()
```

[24]:

	TRIP ID	START TIME	STOP TIME	BIKE ID	\
9992991	12979230	01/01/2017 12:00:00 AM	01/01/2017 12:06:00 AM	2511	
9992992	12979231	01/01/2017 12:02:00 AM	01/01/2017 12:08:00 AM	3660	
9992993	12979232	01/01/2017 12:06:00 AM	01/01/2017 12:18:00 AM	4992	
9992994	12979233	01/01/2017 12:07:00 AM	01/01/2017 12:12:00 AM	5637	
9992995	12979234	01/01/2017 12:07:00 AM	01/01/2017 12:20:00 AM	2209	

	TRIP DURATION	FROM STATION ID	FROM STATION NAME	\
9992991	356	414	Canal St & Taylor St	
9992992	327	28	Larrabee St & Menomonee St	
9992993	745	620	Orleans St & Chestnut St (NEXT Apts)	
9992994	323	287	Franklin St & Monroe St	
9992995	776	300	Broadway & Barry Ave	

	TO STATION ID	TO STATION NAME	USER TYPE	…	\
9992991	191	Canal St & Monroe St (*)	Customer	…	
9992992	20	Sheffield Ave & Kingsbury St	Subscriber	…	
9992993	333	Ashland Ave & Blackhawk St	Subscriber	…	
9992994	68	Clinton St & Tilden St	Subscriber	…	
9992995	118	Sedgwick St & North Ave	Subscriber	…	

	FROM LONGITUDE	FROM LOCATION	TO LATITUDE	\
9992991	-87.639474	POINT (-87.639474 41.870257)	41.880884	
9992992	-87.643320	POINT (-87.64332 41.91468)	41.910522	
9992993	-87.637536	POINT (-87.637536 41.898203)	41.907066	
9992994	-87.635185	POINT (-87.635185 41.880317)	41.875885	
9992995	-87.644095	POINT (-87.644095 41.937725)	41.911386	

	TO LONGITUDE	TO LOCATION	YEAR	\
9992991	-87.639525	POINT (-87.639525 41.880884)	2017	
9992992	-87.653106	POINT (-87.653106 41.910522)	2017	
9992993	-87.667252	POINT (-87.667252 41.907066)	2017	
9992994	-87.640795	POINT (-87.640795 41.875885)	2017	
9992995	-87.638677	POINT (-87.638677 41.911386)	2017	

	from_point	to_point	FROM WARD	\
9992991	POINT (-87.63947 41.87026)	POINT (-87.63953 41.88088)	28.0	
9992992	POINT (-87.64332 41.91468)	POINT (-87.65311 41.91052)	43.0	
9992993	POINT (-87.63754 41.89820)	POINT (-87.66725 41.90707)	2.0	
9992994	POINT (-87.63519 41.88032)	POINT (-87.64079 41.87588)	42.0	
9992995	POINT (-87.64409 41.93773)	POINT (-87.63868 41.91139)	44.0	

	TO WARD
9992991	42.0
9992992	27.0
9992993	1.0
9992994	28.0

```
9992995     2.0

[5 rows x 23 columns]
```

[23]:
```
df_MM_clean_2018 = df_MM_clean[df_MM_clean['YEAR'] == '2018']
```

[24]:
```
# Convert 'FROM LOCATION' and 'TO LOCATION' in df_chicago from WKT to shapely␣
 ↪Point objects
# These columns contain POINT data in text format
df_MM_clean_2018['from_point'] = df_MM_clean_2018['FROM LOCATION'].apply(wkt.
 ↪loads)
df_from_gdf_2018 = gpd.GeoDataFrame(df_MM_clean_2018, geometry='from_point')
df_MM_clean_2018['to_point'] = df_MM_clean_2018['TO LOCATION'].apply(wkt.loads)
df_to_gdf_2018 = gpd.GeoDataFrame(df_MM_clean_2018, geometry='to_point')
```

```
/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  def init(spark_master="local"):
```

[25]:
```
# Ensure the CRS for both GeoDataFrames are the same
# This can be done by setting the CRS(Co-ordinate Reference System) of␣
 ↪df_from_gdf and df_to_gdf to match that of wards_gdf
df_from_gdf_2018 = df_from_gdf_2018.set_crs('EPSG:4326',␣
 ↪inplace=True,allow_override = True)
df_to_gdf_2018 = df_to_gdf_2018.set_crs('EPSG:4326',␣
 ↪inplace=True,allow_override = True)
wards_gdf = wards_gdf.set_crs('EPSG:4326', inplace=True,allow_override = True)
```

[26]:
```
# Perform spatial joins
# Join df_from_gdf and df_to_gdf with wards_gdf to find the corresponding wards
from_joined_2018 = gpd.sjoin(df_from_gdf_2018, wards_gdf, how="left",␣
 ↪op='within')
to_joined_2018 = gpd.sjoin(df_to_gdf_2018, wards_gdf, how="left", op='within')
```

```
/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3472:
FutureWarning: The `op` parameter is deprecated and will be removed in a future
release. Please use the `predicate` parameter instead.
  if (await self.run_code(code, result,  async_=asy)):
/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3472:
FutureWarning: The `op` parameter is deprecated and will be removed in a future
release. Please use the `predicate` parameter instead.
  if (await self.run_code(code, result,  async_=asy)):
```

[27]:
```
# Add the ward information back to the original df_chicago DataFrame
df_MM_clean_2018['FROM WARD'] = from_joined_2018['Ward']  # Replace 'Ward' with␣
 ↪the actual column name in wards_gdf
df_MM_clean_2018['TO WARD'] = to_joined_2018['Ward']      # Replace 'Ward' with␣
 ↪the actual column name in wards_gdf
```

```
/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  from ipykernel.ipkernel import IPythonKernel
/usr/local/share/jupyter/kernels/pyspark_yarn/bdkernelyarn.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[28]:
```
df_MM_clean_2018.to_csv("df_MM_clean_2018", index=False)
```

[30]:
```
# Display the first few rows of the updated df_chicago to check the results
df_MM_clean_2018.head()
```

[30]:

	TRIP ID	START TIME	STOP TIME	BIKE ID	\
11544487	18511623	05/21/2018 05:07:29 PM	05/21/2018 05:11:48 PM	153	
11546857	18002370	04/01/2018 02:24:43 PM	04/01/2018 02:38:54 PM	3101	
11546858	18002371	04/01/2018 02:25:30 PM	04/01/2018 02:32:41 PM	5226	
11546859	18002372	04/01/2018 02:25:51 PM	04/01/2018 02:42:22 PM	4861	
11546860	18002373	04/01/2018 02:25:58 PM	04/01/2018 02:42:23 PM	4706	

	TRIP DURATION	FROM STATION ID	FROM STATION NAME	\
11544487	259	51	Clark St & Randolph St	
11546857	851	106	State St & Pearson St	

```
11546858            431              426     Ellis Ave & 60th St
11546859            991              110   Dearborn St & Erie St
11546860            985              214   Damen Ave & Grand Ave

          TO STATION ID               TO STATION NAME   USER TYPE  …  \
11544487             91  Clinton St & Washington Blvd  Subscriber  …
11546857            174         Canal St & Madison St  Subscriber  …
11546858            322         Kimbark Ave & 53rd St  Subscriber  …
11546859             31     Franklin St & Chicago Ave  Subscriber  …
11546860             47          State St & Kinzie St  Subscriber  …

         FROM LONGITUDE                          FROM LOCATION  TO LATITUDE  \
11544487     -87.631890      POINT (-87.63188991 41.884576228)    41.883380
11546857     -87.628722           POINT (-87.628722 41.897448)    41.882091
11546858     -87.601073  POINT (-87.6010727606 41.78509714636)    41.799568
11546859     -87.629318           POINT (-87.629318 41.893992)    41.896776
11546860     -87.676860             POINT (-87.67686 41.89122)    41.889187

          TO LONGITUDE                    TO LOCATION  YEAR  \
11544487    -87.641170    POINT (-87.64117 41.88338)  2018
11546857    -87.639833  POINT (-87.639833 41.882091)  2018
11546858    -87.594747  POINT (-87.594747 41.799568)  2018
11546859    -87.635633  POINT (-87.635633 41.896776)  2018
11546860    -87.627754  POINT (-87.627754 41.889187)  2018

                          from_point                    to_point FROM WARD  \
11544487  POINT (-87.63189 41.88458)  POINT (-87.64117 41.88338)      42.0
11546857  POINT (-87.62872 41.89745)  POINT (-87.63983 41.88209)      42.0
11546858  POINT (-87.60107 41.78510)  POINT (-87.59475 41.79957)       5.0
11546859  POINT (-87.62932 41.89399)  POINT (-87.63563 41.89678)      42.0
11546860  POINT (-87.67686 41.89122)  POINT (-87.62775 41.88919)      36.0

         TO WARD
11544487    42.0
11546857    42.0
11546858     4.0
11546859    27.0
11546860    42.0

[5 rows x 23 columns]
```

……

- Compute then the O-D matrix, i.e., the number of bookings starting in ward i and ending in ward j.

[42]:
```
# OD Matrix 2016 calculation using pivot table
matrix_2016 = (
```

```
    df_MM_clean_2016.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_2016
```

[42]:

TO WARD	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	11.0	12.0	\
FROM WARD											
1.0	42043	4799	36	470	4	0	0	0	11	4	
2.0	5666	29050	1399	3143	11	0	0	0	62	0	
3.0	46	1328	5832	10919	1165	2	0	1	2519	29	
4.0	482	3534	12072	26365	9868	0	4	2	2880	24	
5.0	3	7	1107	9704	32426	48	126	212	32	0	
6.0	0	0	3	1	46	133	0	61	0	0	
7.0	0	0	0	3	114	3	356	29	0	0	
8.0	0	0	1	1	207	68	36	147	0	0	
11.0	34	45	2365	2742	19	0	0	0	11505	431	
12.0	2	1	51	38	0	0	0	0	414	199	
15.0	0	0	1	17	0	0	0	0	67	45	
16.0	1	0	3	1	3	5	0	0	4	2	
17.0	1	0	0	0	0	0	0	0	1	0	
20.0	0	0	617	342	1086	37	2	17	196	7	
22.0	4	0	0	0	0	0	0	0	3	4	
24.0	45	0	1	2	0	0	0	0	0	6	
25.0	560	104	906	1067	16	0	0	0	1637	252	
26.0	6498	357	1	7	0	0	0	0	3	2	
27.0	15413	20016	457	2271	2	0	0	0	408	145	
28.0	777	556	907	2774	1	0	0	0	1470	37	
29.0	1	0	1	0	0	0	0	0	0	0	
30.0	232	2	0	1	0	0	0	0	0	0	
32.0	16348	3622	6	101	28	0	0	0	5	3	
33.0	460	56	0	0	0	0	0	0	0	0	
34.0	8598	21895	12189	21173	32	0	0	0	2331	86	
35.0	5596	102	0	8	0	0	0	0	1	1	
36.0	6557	822	9	43	0	0	0	0	11	5	
37.0	0	0	0	0	0	0	0	0	0	0	
39.0	92	5	0	0	0	0	0	0	0	0	
40.0	86	16	0	7	0	0	0	0	0	1	
42.0	17588	63031	9717	26721	114	1	0	0	1426	4	
43.0	5807	30248	134	1050	9	0	0	0	11	0	
44.0	1821	6400	8	95	3	0	0	0	2	0	
45.0	114	3	0	0	0	0	0	0	0	0	
46.0	244	2562	9	26	0	0	0	0	0	0	
47.0	1663	892	0	34	0	0	0	0	0	0	

```
48.0          39    230      2      7      1     0     0     0      0     0
49.0           2     12      1      2      0     0     0     0      0     0
50.0           2      0      0      0      0     0     0     0      0     0

TO WARD    …  40.0    42.0   43.0   44.0  45.0   46.0   47.0   48.0  49.0  \
FROM WARD  …
1.0        …    45   16561   5696   1931   103    332   1834     41     4
2.0        …    21   58381  34166   7603     2   3214    832    395    20
3.0        …     0    9427    199      3     1      8      3      2     0
4.0        …     9   27605   1154    125     2     43     24      8     1
5.0        …     0     132      3      2     0      2      0      2     0
6.0        …     0       0      0      0     0      0      0      0     0
7.0        …     0       0      0      0     0      0      0      0     0
8.0        …     0       0      0      0     0      0      0      0     0
11.0       …     0    1589     23      5     0      2      1      1     0
12.0       …     0       2      1      0     0      0      0      0     0
15.0       …     0       1      0      0     0      0      0      0     0
16.0       …     0       0      0      0     0      0      0      0     0
17.0       …     0       0      0      0     0      0      0      0     0
20.0       …     0       1      0      0     0      0      0      0     0
22.0       …     0       1      0      0     0      0      0      0     0
24.0       …     0       5      0      0     9      0      2      0     0
25.0       …     2    3185    119     18     0      4      0     14     0
26.0       …     2    1302    304     51     6     15     82      4     0
27.0       …    14   62672  16815   4424     3   1151    802     31     7
28.0       …     0    7875    341     56     1      8      7      3     3
29.0       …     0       1      0      0     0      0      0      0     0
30.0       …    12       3      2     54   113     14    107      0     0
32.0       …   343    5527  25166  15397   122   2545   9112    297    14
33.0       …   577      10    165    275   230    422   1803    124     7
34.0       …     8  123180   9069   1435     0    233    171     18    13
35.0       …   135     332    708    709   377    138   1025     56     4
36.0       …     6    3573    369     90     3     26    109      2     0
37.0       …     0       0      0      0     0      0      0      0     0
39.0       …   324       0      7     18   179    116    463    129    14
40.0       …  3329      33    284    493    64    840   6624   2771  1807
42.0       …    27  234102  42308   7507     6   2077    815    425    44
43.0       …   240   34691  99908  27845     8  12764   5195   3223    56
44.0       …   502    5601  24680  37549    56  20203  10325   2724   145
45.0       …    36       0     19     72   702     73    317      7     0
46.0       …   880    1524  12065  20376    63  44674  11799   6453   397
47.0       …  7206     382   4469   9372   253  11695  40315   5765   467
48.0       …  2935     179   2826   2556     4   5665   5096  12888  2308
49.0       …  1871      26     58    138     0    366    474   2096  8675
50.0       …   181      13      1      4     0     13     94     93   170

TO WARD    50.0
```

```
FROM WARD
1.0           1
2.0           0
3.0           0
4.0           2
5.0           0
6.0           0
7.0           0
8.0           0
11.0          0
12.0          0
15.0          0
16.0          0
17.0          0
20.0          0
22.0          0
24.0          0
25.0          0
26.0          0
27.0          4
28.0          0
29.0          0
30.0          0
32.0          5
33.0         21
34.0          6
35.0          2
36.0          0
37.0          0
39.0         19
40.0        189
42.0         12
43.0          1
44.0          4
45.0          0
46.0         12
47.0         94
48.0        137
49.0        187
50.0         21

[39 rows x 39 columns]
```

[17]:
```
# OD Matrix 2017 calculation using pivot table
matrix_2017 = (
    df_MM_clean_2017.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
```

```
                values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_2017
```

[17]:

FROM WARD \ TO WARD	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	11.0	12.0
1.0	43441	5944	48	495	10	0	0	0	23	8
2.0	6570	55925	1693	8216	151	0	0	0	128	3
3.0	57	1535	8353	19513	1729	4	0	3	3523	68
4.0	547	8826	21660	80686	14984	6	9	1	5316	20
5.0	12	123	1632	15477	41837	27	167	216	82	4
6.0	0	0	4	3	28	881	10	220	1	0
7.0	0	0	1	8	158	2	970	74	0	0
8.0	0	0	2	5	221	220	70	383	0	0
11.0	39	115	2918	4898	66	2	0	0	12175	485
12.0	1	1	140	28	2	0	0	0	458	480
15.0	2	0	14	17	2	0	0	0	202	74
16.0	7	0	21	5	10	21	1	0	23	19
17.0	0	0	0	0	0	0	0	0	0	0
20.0	0	5	782	506	1751	47	2	27	264	33
22.0	0	0	0	6	0	0	0	0	1	6
24.0	7	1	2	6	0	0	0	0	9	16
25.0	568	289	1286	1626	35	0	0	0	1843	388
26.0	7367	359	3	24	0	0	0	0	3	5
27.0	18166	25614	444	2634	38	0	0	0	328	129
28.0	690	765	1394	4419	8	0	0	0	1418	43
29.0	1	1	0	1	0	0	0	0	0	0
30.0	230	5	0	1	0	0	0	0	0	0
32.0	18910	4774	8	216	29	0	0	0	7	2
33.0	476	49	0	2	0	0	0	0	0	0
34.0	10811	29154	14630	32089	172	2	1	0	2960	117
35.0	6198	213	3	11	1	0	0	0	2	1
36.0	8051	1071	15	89	0	0	0	2	13	2
37.0	2	0	0	0	0	0	0	0	0	0
39.0	103	8	0	0	1	0	0	0	1	0
40.0	126	51	1	16	0	0	0	0	0	0
42.0	20908	108470	11556	70192	1047	5	5	2	2010	21
43.0	7680	49991	207	2540	65	0	0	0	33	2
44.0	2254	9500	33	290	8	0	0	0	6	1
45.0	213	17	1	1	0	0	0	0	0	0
46.0	324	3777	14	153	4	0	0	0	5	0
47.0	1724	968	2	63	0	0	0	0	5	0
48.0	94	663	6	63	6	0	0	0	4	0
49.0	91	69	0	8	1	0	0	0	1	0

```
50.0          3       3      0      1      1     0     0     0      0     0

TO WARD    …  40.0    42.0    43.0   44.0  45.0   46.0   47.0   48.0   49.0  \
FROM WARD  …
1.0        …    81   19523    7921   2268   206    405   1989     89    110
2.0        …    42   98255   54460  11373    15   4576    988    762     95
3.0        …     0   11604     302     50     0     15      5      6      0
4.0        …     8   77324    3330    438     2    210     70     78     11
5.0        …     0    1075      61     17     0      7      0      4      0
6.0        …     0       3       0      0     0      0      0      0      0
7.0        …     0       1       1      0     0      0      0      0      0
8.0        …     0       1       0      0     0      0      0      0      0
11.0       …     0    1975      40      7     0      3      2      2      0
12.0       …     0       9       0      0     0      0      0      0      0
15.0       …     0       0       0      0     0      0      0      0      0
16.0       …     0       0       0      0     0      0      0      0      0
17.0       …     0       0       0      0     0      0      0      0      0
20.0       …     0      14       5      0     0      0      0      0      0
22.0       …     0       2       0      0     0      0      0      0      0
24.0       …     0      14       6      1     0      0      0      0      0
25.0       …     1    3363     104     39     0      8      5      2      0
26.0       …    21    1167     543     92    17     15    238      5      0
27.0       …    18   69599   20149   4852     7   1034    845     81     16
28.0       …     1    9782     327    133     3     16     10      1      0
29.0       …     0       1       1      0     2      0      0      0      1
30.0       …     6       3      22     54   310      8     88      2      0
32.0       …   392    6445   29296  16283   253   2752   9532    328     69
33.0       …   655      20     212    355   440    539   2003    143     11
34.0       …    17  152442   11483   2147    11    354    203     70     19
35.0       …   203     450    1058    756   475    167   1071     49      7
36.0       …     5    3527     587    209     6     45    199     11      1
37.0       …     0       0       0      0     1      0      0      0      0
39.0       …   362       3      16     28   187     67    554    175     24
40.0       …  4156      60     323    685    88   1024   7172   3376   1885
42.0       …    62  371506   77950  12228    21   3951    991    972    106
43.0       …   263   66598  141220  39999    19  18595   5670   4707    180
44.0       …   739   10069   37410  48755    85  24818  10522   4701    397
45.0       …    57       9      21     93  1166     86    406     31      4
46.0       …   994    3293   17842  24189    84  48647  11836   8914    666
47.0       …  7668     664    5251  10085   436  11738  41035   6310    626
48.0       …  3566     793    4320   4605    37   7642   5508  15243   3297
49.0       …  1987     106     165    198     2    602    595   3130  15682
50.0       …   281       4       3     12     1     17     94    339    191

TO WARD    50.0
FROM WARD
1.0            1
```

```
2.0           2
3.0           0
4.0           1
5.0           0
6.0           0
7.0           0
8.0           0
11.0          0
12.0          0
15.0          0
16.0          0
17.0          0
20.0          0
22.0          0
24.0          0
25.0          0
26.0          2
27.0          3
28.0          0
29.0          0
30.0          1
32.0          2
33.0          9
34.0          1
35.0          3
36.0          0
37.0          0
39.0         20
40.0        299
42.0         12
43.0         10
44.0         18
45.0          0
46.0         14
47.0        105
48.0        404
49.0        231
50.0         64

[39 rows x 39 columns]
```

[18]:
```
# OD Matrix 2018 calculation using pivot table
matrix_2018 = (
    df_MM_clean_2018.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
```

```
    .astype(int)
).sort_values('FROM WARD')

matrix_2018
```

[18]:

TO WARD	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	11.0	12.0	\
FROM WARD											
1.0	42245	5377	148	699	37	0	0	2	123	137	
2.0	5974	50698	1541	7276	457	2	5	1	251	10	
3.0	167	1637	8206	15103	1756	11	0	7	3276	92	
4.0	671	7861	17104	52161	14150	16	21	31	4240	47	
5.0	25	283	1731	14741	50077	94	233	179	173	4	
6.0	0	0	7	6	96	906	18	221	0	0	
7.0	0	6	1	19	193	9	1176	53	0	0	
8.0	0	0	4	33	200	218	78	335	1	0	
11.0	115	259	2858	3930	155	1	0	1	11983	420	
12.0	36	3	152	66	5	0	0	0	385	533	
15.0	1	0	11	32	17	2	0	0	167	226	
16.0	1	1	13	7	9	51	0	13	19	10	
17.0	0	0	0	1	0	10	0	0	0	0	
20.0	2	7	516	482	1835	102	10	62	155	26	
22.0	2	19	1	34	0	0	0	0	5	0	
24.0	17	8	3	24	1	1	0	1	2	13	
25.0	637	461	984	1347	94	2	0	2	1730	380	
26.0	6368	374	2	37	3	0	0	0	12	1	
27.0	20261	24788	776	2914	80	1	0	0	528	175	
28.0	1067	879	1510	3587	42	3	1	0	1302	101	
29.0	10	6	0	3	0	0	0	0	1	0	
30.0	164	23	0	1	1	0	0	0	0	0	
32.0	17092	5131	35	271	16	0	0	0	17	2	
33.0	515	73	0	7	0	0	0	0	0	0	
34.0	11486	25822	15017	29069	418	16	0	5	3003	148	
35.0	5351	329	5	52	3	0	0	0	8	1	
36.0	6793	979	22	224	4	0	0	0	40	3	
37.0	6	0	0	0	0	0	0	0	0	1	
39.0	40	30	0	2	0	0	0	0	0	0	
40.0	140	92	5	15	0	0	0	0	1	0	
42.0	22568	93985	11986	50638	2161	8	12	10	2367	47	
43.0	7586	39065	445	3688	228	0	0	0	126	5	
44.0	2674	9276	85	846	64	1	0	0	43	1	
45.0	138	20	0	7	0	0	0	0	0	0	
46.0	723	4416	34	368	28	0	2	0	12	1	
47.0	1508	1277	4	109	6	0	0	0	1	0	
48.0	157	1184	15	207	30	0	0	0	7	0	
49.0	78	126	7	26	5	0	0	0	4	0	
50.0	0	8	0	0	1	0	0	0	0	0	

TO WARD	…	40.0	42.0	43.0	44.0	45.0	46.0	47.0	48.0	49.0	\
FROM WARD	…										
1.0	…	166	20375	7525	2511	187	751	1715	143	97	
2.0	…	131	83062	42812	10784	11	5531	1514	1182	177	
3.0	…	4	12094	667	111	1	59	7	23	4	
4.0	…	14	57861	4433	921	7	472	130	236	27	
5.0	…	4	2565	196	41	0	29	3	25	9	
6.0	…	0	7	0	0	0	0	0	0	0	
7.0	…	0	10	2	0	0	0	0	0	0	
8.0	…	0	6	1	1	0	0	0	0	0	
11.0	…	0	2279	135	39	0	7	3	12	0	
12.0	…	0	35	2	0	0	0	0	0	0	
15.0	…	0	1	0	0	0	0	0	0	0	
16.0	…	0	8	0	0	0	0	0	0	0	
17.0	…	0	0	0	0	0	0	0	0	0	
20.0	…	0	43	3	0	0	0	0	2	0	
22.0	…	0	12	1	0	0	0	0	0	0	
24.0	…	0	55	3	1	0	0	1	0	0	
25.0	…	2	3584	230	48	0	30	17	9	1	
26.0	…	15	1363	629	204	13	53	126	20	0	
27.0	…	75	76685	19879	5316	36	1402	1277	227	31	
28.0	…	5	9288	411	94	12	34	27	12	2	
29.0	…	0	10	3	0	3	0	0	0	0	
30.0	…	12	23	21	85	510	22	61	9	0	
32.0	…	425	8394	28219	15832	142	3120	8691	480	90	
33.0	…	473	121	216	373	340	479	1818	189	35	
34.0	…	46	144664	14344	3480	32	1090	615	235	63	
35.0	…	154	1129	1056	823	529	255	1132	106	12	
36.0	…	16	3869	603	212	13	63	186	8	2	
37.0	…	0	0	1	0	1	0	0	0	0	
39.0	…	280	21	34	56	184	137	635	155	24	
40.0	…	3509	230	506	725	122	1008	6846	3294	1915	
42.0	…	171	341830	65682	16282	59	7802	2337	2079	305	
43.0	…	447	57386	116232	34299	36	13446	5554	3777	367	
44.0	…	770	12774	31623	47432	98	22656	10137	3523	462	
45.0	…	130	32	49	90	1387	112	420	72	1	
46.0	…	1137	6441	12972	22505	110	46273	10764	7625	725	
47.0	…	7579	1797	5104	9259	391	10521	38031	5620	522	
48.0	…	3301	1727	3562	3252	86	6629	5012	15465	2513	
49.0	…	2107	219	484	330	5	524	535	2261	15411	
50.0	…	139	5	29	16	11	23	83	193	330	

TO WARD	50.0
FROM WARD	
1.0	4
2.0	4
3.0	0

```
4.0           2
5.0           0
6.0           0
7.0           0
8.0           0
11.0          0
12.0          0
15.0          0
16.0          0
17.0          0
20.0          0
22.0          0
24.0          0
25.0          0
26.0          1
27.0         27
28.0          0
29.0          0
30.0          2
32.0          7
33.0         11
34.0          2
35.0         10
36.0          0
37.0          0
39.0         14
40.0        143
42.0          9
43.0         22
44.0         16
45.0         13
46.0         23
47.0        110
48.0        235
49.0        284
50.0         76

[39 rows x 39 columns]
```

[19]:
```
# Saving OD Matrices
matrix_2016.to_csv("matrix_2016", index=False)
matrix_2017.to_csv("matrix_2017", index=False)
matrix_2018.to_csv("matrix_2018", index=False)
```

……

- Prepare OD matrices for different years and different age groups(3 OD matrices for each age-group of 3 consecutive years). Are there any periodicity or trends noticed? Is there a

difference between the OD matrices for different age groups?

[2]:
```
import pandas as pd
```

[3]:
```
# Using saved dataframes with "from wards" and "to wards" to avoid kernel␣
 ↪disconnecting.
df_MM_clean_2016 = pd.read_csv("df_MM_clean_2016", on_bad_lines = 'warn',␣
 ↪sep=',')
df_MM_clean_2017 = pd.read_csv("df_MM_clean_2017", on_bad_lines = 'warn',␣
 ↪sep=',')
df_MM_clean_2018 = pd.read_csv("df_MM_clean_2018", on_bad_lines = 'warn',␣
 ↪sep=',')
```

[4]:
```
# Drop null values
df_MM_clean_2016 = df_MM_clean_2016.dropna(subset=['BIRTH YEAR'])
```

[5]:
```
# Convert BIRTH YEAR to integer
df_MM_clean_2016['BIRTH YEAR'] = df_MM_clean_2016['BIRTH YEAR'].astype('int')
```

[6]:
```
# Create age groups based on the agr distribution of users
df_MM_clean_2016['age'] = df_MM_clean_2016['YEAR'] - df_MM_clean_2016['BIRTH␣
 ↪YEAR']
```

[7]:
```
# Define age groups
age_bins = [0, 18, 30, 40, 50, 60, float('inf')]  # Define age bins/ranges
age_labels = ['0-18', '19-30', '31-40', '41-50', '51-60', '61+']  # Define␣
 ↪corresponding labels
```

[8]:
```
# Create 'age group' column using pd.cut
df_MM_clean_2016['age group'] = pd.cut(df_MM_clean_2016['age'], bins=age_bins,␣
 ↪labels=age_labels, right=False)
```

[9]:
```
age_group_under_18_2016 = df_MM_clean_2016[df_MM_clean_2016['age group'] ==␣
 ↪'0-18']
# OD Matrix
matrix_age_under_18_2016 = (
    age_group_under_18_2016.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_age_under_18_2016
```

[9]:
```
TO WARD    1.0   2.0   3.0   4.0   5.0   7.0   8.0   11.0  12.0  20.0  …  \
FROM WARD                                                              …
```

1.0	1	0	0	0	0	0	0	0	0	0	…
2.0	3	22	0	0	0	0	0	0	0	0	…
3.0	0	0	4	4	3	0	0	4	0	0	…
4.0	0	0	0	43	24	1	0	6	0	1	…
5.0	0	0	2	8	57	2	2	0	0	0	…
7.0	0	0	0	0	5	0	1	0	0	0	…
8.0	0	0	0	0	3	0	0	0	0	0	…
11.0	0	0	12	3	0	0	0	171	2	1	…
12.0	0	0	0	0	0	0	0	3	4	0	…
20.0	0	0	0	0	0	0	0	0	0	2	…
24.0	0	0	0	0	0	0	0	0	0	0	…
25.0	0	0	0	3	0	0	0	3	0	0	…
26.0	0	0	0	0	0	0	0	0	0	0	…
27.0	4	9	0	0	0	0	0	2	0	0	…
28.0	0	0	0	1	0	0	0	4	0	0	…
32.0	1	6	0	0	0	0	0	0	0	0	…
33.0	0	0	0	0	0	0	0	0	0	0	…
34.0	0	3	0	38	0	0	0	12	0	0	…
35.0	0	0	0	0	0	0	0	0	0	0	…
36.0	18	0	0	0	0	0	0	0	0	0	…
39.0	0	0	0	0	0	0	0	0	0	0	…
40.0	0	0	0	0	0	0	0	0	0	0	…
42.0	0	30	0	26	0	0	0	0	0	0	…
43.0	4	62	0	1	0	0	0	0	0	0	…
44.0	1	3	0	0	0	0	0	0	0	0	…
45.0	0	0	0	0	0	0	0	0	0	0	…
46.0	0	0	0	0	0	0	0	0	0	0	…
47.0	1	0	0	0	0	0	0	0	0	0	…
48.0	0	0	0	0	0	0	0	0	0	0	…
49.0	0	0	0	0	0	0	0	0	0	0	…

TO WARD	39.0	40.0	42.0	43.0	44.0	45.0	46.0	47.0	48.0	49.0
FROM WARD										
1.0	0	0	1	3	0	0	0	1	0	0
2.0	0	0	31	65	1	0	1	2	0	0
3.0	0	0	2	0	0	0	0	0	0	0
4.0	0	0	29	1	0	0	0	0	0	0
5.0	0	0	0	0	0	0	0	0	0	0
7.0	0	0	0	0	0	0	0	0	0	0
8.0	0	0	0	0	0	0	0	0	0	0
11.0	0	0	0	0	0	0	0	0	0	0
12.0	0	0	0	0	0	0	0	0	0	0
20.0	0	0	0	0	0	0	0	0	0	0
24.0	0	0	0	0	0	0	0	0	0	0
25.0	0	0	1	0	0	0	0	0	0	0
26.0	0	0	0	0	0	0	0	0	0	0
27.0	0	0	7	13	0	0	0	1	0	0

```
28.0          0     0     2     0     0     0     0     0     0     0
32.0          0     0     2    31    21     0     1    70     0     0
33.0          1     0     0     0     0    15     1     1     0     0
34.0          0     0    22     0     0     0     0     0     0     0
35.0          0     0     0     0     0     1     0     2     0     0
36.0          0     0     0     0     0     0     0     0     0     0
39.0          1     0     0     0     0     0     0     1     0     0
40.0          0     0     0     0     0     1     0     0     0     0
42.0          0     0    56    29     1     0     0     0     0     0
43.0          0     0    32    93    28     0    14     5     1     0
44.0          0     0     4    18    63     0     5    81     0     0
45.0          1     1     0     0     0     2     0     3     0     0
46.0          0     0     0    14     1     0     7    11     0     0
47.0          0     2     0     7   170     5     9    49     0     0
48.0          0     0     0     2     0     0     1     0     0     0
49.0          0     0     0     0     0     0     0     0     0     1

[30 rows x 29 columns]
```

It shows the OD matrix of under-18 age group users in 2016

[10]:
```
age_group_19_30_2016 = df_MM_clean_2016[df_MM_clean_2016['age group'] ==␣
 ↪'19-30']
# OD Matrix
matrix_age_19_30_2016 = (
    age_group_19_30_2016.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_age_19_30_2016
```

[10]:
```
TO WARD     1.0    2.0   3.0    4.0    5.0   6.0   7.0   8.0    11.0  12.0  \
FROM WARD
1.0        20052   2388     7    163      1     0     0     0      7     2
2.0         2913  11535   309    962      4     0     0     0     21     0
3.0           14    351  1883   5250    321     0     0     1   1324     9
4.0          122   1079  5768  10836   4338     0     1     0   1242    20
5.0            1      4   390   4367  16485     6    51    46      4     0
6.0            0      0     0      1      7    31     0    18      0     0
7.0            0      0     0      0     50     3    55     9      0     0
8.0            0      0     1      0     40    14    12    18      0     0
11.0          13     13  1291   1073      4     0     0     0   6040   191
12.0           2      1    10     11      0     0     0     0    204    78
15.0           0      0     0      2      0     0     0     0      5     5
```

16.0	0	0	0	1	1	3	0	0	2	2
17.0	1	0	0	0	0	0	0	0	1	0
20.0	0	0	228	215	541	1	0	6	15	7
22.0	4	0	0	0	0	0	0	0	2	1
24.0	44	0	0	0	0	0	0	0	0	0
25.0	231	30	338	407	1	0	0	0	556	71
26.0	3032	73	1	4	0	0	0	0	1	1
27.0	7156	7824	206	650	0	0	0	0	122	14
28.0	355	237	265	1193	0	0	0	0	776	3
29.0	0	0	0	0	0	0	0	0	0	0
30.0	106	0	0	0	0	0	0	0	0	0
32.0	6569	1470	0	32	1	0	0	0	3	0
33.0	189	10	0	0	0	0	0	0	0	0
34.0	3276	5943	2916	5624	13	0	0	0	1003	13
35.0	2247	58	0	1	0	0	0	0	0	0
36.0	2381	287	3	17	0	0	0	0	2	1
37.0	0	0	0	0	0	0	0	0	0	0
39.0	33	0	0	0	0	0	0	0	0	0
40.0	28	6	0	5	0	0	0	0	0	1
42.0	6786	20661	2197	7664	48	1	0	0	524	2
43.0	3001	13692	62	375	3	0	0	0	4	0
44.0	1015	2614	3	53	2	0	0	0	1	0
45.0	57	1	0	0	0	0	0	0	0	0
46.0	138	856	0	7	0	0	0	0	0	0
47.0	698	214	0	3	0	0	0	0	0	0
48.0	16	68	0	1	0	0	0	0	0	0
49.0	1	10	1	0	0	0	0	0	0	0
50.0	2	0	0	0	0	0	0	0	0	0

TO WARD	…	40.0	42.0	43.0	44.0	45.0	46.0	47.0	48.0	49.0	50.0
FROM WARD	…										
1.0	…	14	6142	3044	1006	49	204	942	10	2	0
2.0	…	6	19307	15869	3339	0	1097	223	119	4	0
3.0	…	0	1949	78	1	0	1	0	1	0	0
4.0	…	3	8038	478	65	0	21	2	3	0	0
5.0	…	0	60	2	2	0	0	0	2	0	0
6.0	…	0	0	0	0	0	0	0	0	0	0
7.0	…	0	0	0	0	0	0	0	0	0	0
8.0	…	0	0	0	0	0	0	0	0	0	0
11.0	…	0	458	5	1	0	1	0	0	0	0
12.0	…	0	0	1	0	0	0	0	0	0	0
15.0	…	0	0	0	0	0	0	0	0	0	0
16.0	…	0	0	0	0	0	0	0	0	0	0
17.0	…	0	0	0	0	0	0	0	0	0	0
20.0	…	0	1	0	0	0	0	0	0	0	0
22.0	…	0	1	0	0	0	0	0	0	0	0
24.0	…	0	1	0	0	0	0	2	0	0	0

```
25.0       …     1    746     22      4     0      2     0     0     0     0
26.0       …     1    426    160     23     3     11    30     1     0     0
27.0       …     3  18163   8666   2202     0    560   197    13     0     0
28.0       …     0   2273    246     31     0      1     4     2     0     0
29.0       …     0      0      0      0     0      0     0     0     0     0
30.0       …     0      0      0      8    13      0     9     0     0     0
32.0       …    84   1779  12884   7957    47   1256  2478   133     5     2
33.0       …   141      4     43     79    81    187   362    15     1    10
34.0       …     5  29315   3319    505     0     68    18     7     0     0
35.0       …    38     52    224    294   140     46   311     4     4     1
36.0       …     0   1299    169     40     0     10    40     1     0     0
37.0       …     0      0      0      0     0      0     0     0     0     0
39.0       …   110      0      0      1    13     52   121    20     1     4
40.0       …   635      4    158    192    12    268  1441   668   385    46
42.0       …     7  61306  17914   2675     1    675   123   106    14     0
43.0       …   117  14456  53347  14609     7   5722  1758   996    14     0
44.0       …   232   2236  13252  18954    10   8930  3790   999    86     1
45.0       …    11      0     15     11   118     33   112     1     0     0
46.0       …   337    475   5620   9341    29  18050  3851  1872   157     7
47.0       …  1485     75   1675   3413    57   3822  9403  1488   162    42
48.0       …   690     87    861   1043     0   1775  1218  2855   733    53
49.0       …   413     13     23     88     0    137   246   571  2025    34
50.0       …    45      0      0      1     0      9    59    30    18     7

[39 rows x 39 columns]
```

[11]:
```
age_group_31_40_2016 = df_MM_clean_2016[df_MM_clean_2016['age group'] ==␣
 ↪'31-40']
# OD Matrix
matrix_age_31_40_2016 = (
    age_group_31_40_2016.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_age_31_40_2016
```

[11]:
```
TO WARD     1.0    2.0   3.0   4.0   5.0   6.0   7.0   8.0   11.0  12.0  … \
FROM WARD                                                              …
1.0       15373   1675    18   226     2     0     0     0     1     2  …
2.0        2048   9473   663  1284     2     0     0     0    33     0  …
3.0          18    561  2398  2985   406     0     0     0   644     3  …
4.0         293   1516  3404  7516  2940     0     1     1  1164     3  …
5.0           1      2   324  2761  9099    28    12   137    27     0  …
6.0           0      0     2     0    29    27     0     4     0     0  …
```

7.0	0	0	0	0	15	0	11	7	0	0	…
8.0	0	0	0	0	136	4	11	73	0	0	…
11.0	17	29	623	1051	14	0	0	0	3393	120	…
12.0	0	0	0	2	0	0	0	0	92	65	…
15.0	0	0	1	12	0	0	0	0	59	38	…
16.0	1	0	1	0	1	1	0	0	2	0	…
17.0	0	0	0	0	0	0	0	0	0	0	…
20.0	0	0	199	67	287	7	1	4	10	0	…
22.0	0	0	0	0	0	0	0	0	1	3	…
24.0	1	0	0	0	0	0	0	0	0	4	…
25.0	265	34	323	520	13	0	0	0	626	55	…
26.0	2425	255	0	2	0	0	0	0	2	1	…
27.0	5955	7215	112	811	2	0	0	0	130	21	…
28.0	363	208	378	815	0	0	0	0	459	9	…
29.0	1	0	1	0	0	0	0	0	0	0	…
30.0	89	2	0	1	0	0	0	0	0	0	…
32.0	6782	1222	3	35	1	0	0	0	2	0	…
33.0	203	15	0	0	0	0	0	0	0	0	…
34.0	3787	8111	5064	6796	10	0	0	0	702	34	…
35.0	2544	30	0	7	0	0	0	0	1	1	…
36.0	2588	429	3	19	0	0	0	0	8	4	…
37.0	0	0	0	0	0	0	0	0	0	0	…
39.0	54	0	0	0	0	0	0	0	0	0	…
40.0	42	5	0	1	0	0	0	0	0	0	…
42.0	8213	22561	3891	9766	38	0	0	0	660	0	…
43.0	2055	8958	47	240	4	0	0	0	6	0	…
44.0	562	2157	3	22	1	0	0	0	1	0	…
45.0	22	2	0	0	0	0	0	0	0	0	…
46.0	90	1044	6	13	0	0	0	0	0	0	…
47.0	627	348	0	31	0	0	0	0	0	0	…
48.0	11	96	1	1	1	0	0	0	0	0	…
49.0	1	1	0	1	0	0	0	0	0	0	…
50.0	0	0	0	0	0	0	0	0	0	0	…

TO WARD	40.0	42.0	43.0	44.0	45.0	46.0	47.0	48.0	49.0	50.0
FROM WARD										
1.0	18	7907	1903	654	26	111	658	8	2	1
2.0	9	20354	10586	2638	1	1387	344	190	7	0
3.0	0	4060	91	1	1	5	3	1	0	0
4.0	6	9573	297	42	2	14	20	3	0	0
5.0	0	36	0	0	0	1	0	0	0	0
6.0	0	0	0	0	0	0	0	0	0	0
7.0	0	0	0	0	0	0	0	0	0	0
8.0	0	0	0	0	0	0	0	0	0	0
11.0	0	886	17	2	0	1	1	1	0	0
12.0	0	1	0	0	0	0	0	0	0	0
15.0	0	1	0	0	0	0	0	0	0	0

```
16.0          0      0      0      0      0      0      0      0      0      0
17.0          0      0      0      0      0      0      0      0      0      0
20.0          0      0      0      0      0      0      0      0      0      0
22.0          0      0      0      0      0      0      0      0      0      0
24.0          0      0      0      0      0      0      0      0      0      0
25.0          1   1801     75     14      0      0      0      3      0      0
26.0          1    560     80     24      3      3     36      2      0      0
27.0          7  25314   5424   1688      2    440    495     14      0      0
28.0          0   3272     74     20      0      5      2      0      3      0
29.0          0      1      0      0      0      0      0      0      0      0
30.0          2      1      2     42     45     14     71      0      0      0
32.0        139   2393   6895   5048     29    903   3852     95      5      1
33.0        187      2     75    160     68    180    857     60      3      2
34.0          3  44024   3238    606      0    127     74      6     10      5
35.0         39    257    367    330    175     75    512     41      0      0
36.0          1   1458    156     41      3      5     50      0      0      0
37.0          0      0      0      0      0      0      0      0      0      0
39.0        121      0      3      8     55     46    169     69     10      6
40.0        915     26    100    195     41    420   2837   1088    848     39
42.0         15  83533  15170   3023      1   1024    468    260      6     10
43.0         96  11421  26725   7852      1   4493   2279   1213     24      0
44.0        160   2045   6589  11882     26   6893   4030    997     35      3
45.0         21      0      4     32    418     27    131      5      0      0
46.0        341    726   3922   6939     28  14462   4932   2411    181      3
47.0       3096    150   1683   3602    141   4879  18727   2523    135     25
48.0       1064     63    932    897      3   2029   2368   4726    787     35
49.0        926      7     18     35      0    174    136    639   2944     84
50.0         39     13      1      2      0      2     13     40    111      2

[39 rows x 39 columns]
```

[12]:
```
age_group_41_50_2016 = df_MM_clean_2016[df_MM_clean_2016['age group'] ==␣
 ↪'41-50']
# OD Matrix
matrix_age_41_50_2016 = (
    age_group_41_50_2016.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_age_41_50_2016
```

[12]:
```
TO WARD    1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   11.0  12.0  …  \
FROM WARD                                                              …
1.0        4836   473     5    54     1     0     0     0     3     0  …
```

2.0	449	3549	222	400	2	0	0	0	2	0	…
3.0	11	251	768	1342	207	0	0	0	260	4	…
4.0	29	417	1458	3812	1114	0	0	1	252	1	…
5.0	1	1	160	1117	3175	11	6	16	1	0	…
6.0	0	0	0	0	4	10	0	0	0	0	…
7.0	0	0	0	0	6	0	3	4	0	0	…
8.0	0	0	0	1	15	0	2	6	0	0	…
11.0	3	0	238	265	1	0	0	0	1002	57	…
12.0	0	0	4	0	0	0	0	0	32	33	…
15.0	0	0	0	3	0	0	0	0	0	1	…
16.0	0	0	1	0	1	0	0	0	0	0	…
20.0	0	0	77	26	159	2	0	4	6	0	…
22.0	0	0	0	0	0	0	0	0	0	0	…
24.0	0	0	1	1	0	0	0	0	0	0	…
25.0	41	32	151	79	1	0	0	0	255	42	…
26.0	764	25	0	1	0	0	0	0	0	0	…
27.0	1564	2484	100	437	0	0	0	0	10	109	…
28.0	33	48	154	334	0	0	0	0	101	24	…
29.0	0	0	0	0	0	0	0	0	0	0	…
30.0	31	0	0	0	0	0	0	0	0	0	…
32.0	2002	394	3	24	0	0	0	0	0	3	…
33.0	41	6	0	0	0	0	0	0	0	0	…
34.0	1249	4391	2107	4376	4	0	0	0	427	39	…
35.0	585	10	0	0	0	0	0	0	0	0	…
36.0	1224	68	3	4	0	0	0	0	1	0	…
37.0	0	0	0	0	0	0	0	0	0	0	…
39.0	4	4	0	0	0	0	0	0	0	0	…
40.0	9	1	0	1	0	0	0	0	0	0	…
42.0	2008	8876	1992	4431	10	0	0	0	159	1	…
43.0	516	3801	13	243	0	0	0	0	0	0	…
44.0	212	809	1	15	0	0	0	0	0	0	…
45.0	32	0	0	0	0	0	0	0	0	0	…
46.0	11	434	0	5	0	0	0	0	0	0	…
47.0	199	162	0	0	0	0	0	0	0	0	…
48.0	10	22	0	5	0	0	0	0	0	0	…
49.0	0	1	0	1	0	0	0	0	0	0	…
50.0	0	0	0	0	0	0	0	0	0	0	…

TO WARD	40.0	42.0	43.0	44.0	45.0	46.0	47.0	48.0	49.0	50.0
FROM WARD										
1.0	5	1790	518	227	25	15	173	20	0	0
2.0	1	8050	3970	847	1	424	124	46	5	0
3.0	0	1860	15	1	0	1	0	0	0	0
4.0	0	4856	176	17	0	6	1	2	1	2
5.0	0	10	0	0	0	0	0	0	0	0
6.0	0	0	0	0	0	0	0	0	0	0
7.0	0	0	0	0	0	0	0	0	0	0

8.0	0	0	0	0	0	0	0	0	0	0
11.0	0	187	0	2	0	0	0	0	0	0
12.0	0	1	0	0	0	0	0	0	0	0
15.0	0	0	0	0	0	0	0	0	0	0
16.0	0	0	0	0	0	0	0	0	0	0
20.0	0	0	0	0	0	0	0	0	0	0
22.0	0	0	0	0	0	0	0	0	0	0
24.0	0	4	0	0	9	0	0	0	0	0
25.0	0	537	16	0	0	2	0	11	0	0
26.0	0	266	37	3	0	0	14	1	0	0
27.0	3	10430	1522	344	1	93	65	2	5	4
28.0	0	1220	3	3	1	2	1	1	0	0
29.0	0	0	0	0	0	0	0	0	0	0
30.0	8	2	0	3	16	0	26	0	0	0
32.0	56	770	3007	1278	38	217	1282	45	1	0
33.0	49	1	30	14	27	36	223	15	1	6
34.0	0	24950	1426	210	0	29	43	3	3	1
35.0	15	10	53	57	33	10	103	3	0	1
36.0	0	563	23	7	0	2	11	0	0	0
37.0	0	0	0	0	0	0	0	0	0	0
39.0	54	0	3	9	20	15	134	16	0	6
40.0	573	3	12	62	3	69	1326	562	419	26
42.0	4	42810	4968	1102	4	240	152	47	23	2
43.0	10	4662	9822	2837	0	1262	785	598	13	1
44.0	64	882	2557	3335	11	1900	1257	396	16	0
45.0	2	0	0	8	101	10	40	0	0	0
46.0	100	127	1228	2073	6	6023	1557	1196	30	1
47.0	1560	132	666	1184	24	1591	7091	1001	111	17
48.0	581	14	561	354	0	954	930	3131	509	26
49.0	378	6	16	9	0	29	51	604	1881	43
50.0	38	0	0	1	0	2	14	14	22	3

[38 rows x 38 columns]

[13]:
```
age_group_51_60_2016 = df_MM_clean_2016[df_MM_clean_2016['age group'] ==␣
 ↪'51-60']
# OD Matrix
matrix_age_51_60_2016 = (
    age_group_51_60_2016.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_age_51_60_2016
```

[13]:

TO WARD	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	11.0	12.0	…	\
FROM WARD											…	
1.0	1426	237	6	14	0	0	0	0	0	0	…	
2.0	196	3083	170	312	2	0	0	0	5	0	…	
3.0	3	127	563	1071	191	0	0	0	206	3	…	
4.0	13	325	1137	2924	1156	0	1	0	127	0	…	
5.0	0	0	214	1105	2281	1	52	7	0	0	…	
6.0	0	0	1	0	0	14	0	4	0	0	…	
7.0	0	0	0	3	35	0	279	7	0	0	…	
8.0	0	0	0	0	9	1	10	38	0	0	…	
11.0	1	2	122	261	0	0	0	0	744	34	…	
12.0	0	0	2	4	0	0	0	0	43	15	…	
15.0	0	0	0	0	0	0	0	0	1	1	…	
16.0	0	0	1	0	0	1	0	0	0	0	…	
17.0	0	0	0	0	0	0	0	0	0	0	…	
20.0	0	0	103	30	82	5	1	1	162	0	…	
22.0	0	0	0	0	0	0	0	0	0	0	…	
24.0	0	0	0	1	0	0	0	0	0	1	…	
25.0	16	6	71	44	1	0	0	0	156	14	…	
26.0	240	3	0	0	0	0	0	0	0	0	…	
27.0	443	1829	31	338	0	0	0	0	7	0	…	
28.0	18	52	67	231	1	0	0	0	90	0	…	
29.0	0	0	0	0	0	0	0	0	0	0	…	
30.0	6	0	0	0	0	0	0	0	0	0	…	
32.0	744	468	0	9	26	0	0	0	0	0	…	
33.0	25	25	0	0	0	0	0	0	0	0	…	
34.0	214	2434	1624	2664	5	0	0	0	151	0	…	
35.0	203	4	0	0	0	0	0	0	0	0	…	
36.0	281	32	0	3	0	0	0	0	0	0	…	
37.0	0	0	0	0	0	0	0	0	0	0	…	
39.0	1	1	0	0	0	0	0	0	0	0	…	
40.0	7	4	0	0	0	0	0	0	0	0	…	
42.0	404	8275	1236	3707	16	0	0	0	64	0	…	
43.0	195	2849	11	151	2	0	0	0	1	0	…	
44.0	24	662	1	5	0	0	0	0	0	0	…	
45.0	2	0	0	0	0	0	0	0	0	0	…	
46.0	5	194	2	1	0	0	0	0	0	0	…	
47.0	106	72	0	0	0	0	0	0	0	0	…	
48.0	2	43	1	0	0	0	0	0	0	0	…	
49.0	0	0	0	0	0	0	0	0	0	0	…	
50.0	0	0	0	0	0	0	0	0	0	0	…	

TO WARD	40.0	42.0	43.0	44.0	45.0	46.0	47.0	48.0	49.0	50.0
FROM WARD										
1.0	8	437	184	37	3	2	56	3	0	0
2.0	4	7880	2885	656	0	282	55	37	3	0
3.0	0	1078	10	0	0	1	0	0	0	0

```
4.0          0   3607   150     1     0     1     1     0     0     0
5.0          0     24     0     0     0     1     0     0     0     0
6.0          0      0     0     0     0     0     0     0     0     0
7.0          0      0     0     0     0     0     0     0     0     0
8.0          0      0     0     0     0     0     0     0     0     0
11.0         0     52     0     0     0     0     0     0     0     0
12.0         0      0     0     0     0     0     0     0     0     0
15.0         0      0     0     0     0     0     0     0     0     0
16.0         0      0     0     0     0     0     0     0     0     0
17.0         0      0     0     0     0     0     0     0     0     0
20.0         0      0     0     0     0     0     0     0     0     0
22.0         0      0     0     0     0     0     0     0     0     0
24.0         0      0     0     0     0     0     0     0     0     0
25.0         0     83     1     0     0     0     0     0     0     0
26.0         0     50    24     1     0     1     2     0     0     0
27.0         1   6809   919   142     0    41    39     1     2     0
28.0         0    721    16     0     0     0     0     0     0     0
29.0         0      0     0     0     0     0     0     0     0     0
30.0         2      0     0     1    37     0     1     0     0     0
32.0        56    324  1701   868     8   139   842    21     2     2
33.0       178      3    17    21    25    12   300    34     2     2
34.0         0  19273   823    61     0     7    36     2     0     0
35.0        41     13    59    26    25     5    84     8     0     0
36.0         5    166    20     2     0     9     8     1     0     0
37.0         0      0     0     0     0     0     0     0     0     0
39.0        31      0     1     0    91     3    30    10     2     3
40.0      1115      0    13    31     7    70   916   392   111    76
42.0         1  34839  3210   598     0    96    61    10     0     0
43.0        16   3084  7598  1906     0   933   328   326     3     0
44.0        36    355  1608  1974     9  1609   925   212     6     0
45.0         0      0     0    21    60     3    24     1     0     0
46.0        83    148   925  1239     0  4693   965   701    26     1
47.0       902     24   352   772    19   947  4025   608    41    10
48.0       534      8   304   192     1   653   462  1759   228    20
49.0       102      0     1     5     0    23    30   246  1260    23
50.0        58      0     0     0     0     0     5     7    16     8

[39 rows x 39 columns]
```

[14]:
```
age_group_above_61_2016 = df_MM_clean_2016[df_MM_clean_2016['age group'] ==␣
 ↪'61+']
# OD Matrix
matrix_age_above_61_2016 = (
    age_group_above_61_2016.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
```

```
    .astype(int)
).sort_values('FROM WARD')

matrix_age_above_61_2016
```

[14]:

TO WARD	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	11.0	12.0	…	\
FROM WARD											…	
1.0	355	26	0	13	0	0	0	0	0	0	…	
2.0	57	1388	35	185	1	0	0	0	1	0	…	
3.0	0	38	216	267	37	2	0	0	81	10	…	
4.0	25	197	305	1234	296	0	0	0	89	0	…	
5.0	0	0	17	346	1329	2	3	4	0	0	…	
6.0	0	0	0	0	6	51	0	35	0	0	…	
7.0	0	0	0	0	3	0	8	1	0	0	…	
8.0	0	0	0	0	4	49	1	12	0	0	…	
11.0	0	1	79	89	0	0	0	0	155	27	…	
12.0	0	0	35	21	0	0	0	0	40	4	…	
15.0	0	0	0	0	0	0	0	0	2	0	…	
16.0	0	0	0	0	0	0	0	0	0	0	…	
17.0	0	0	0	0	0	0	0	0	0	0	…	
20.0	0	0	10	4	17	22	0	2	3	0	…	
22.0	0	0	0	0	0	0	0	0	0	0	…	
24.0	0	0	0	0	0	0	0	0	0	1	…	
25.0	7	2	23	14	0	0	0	0	41	70	…	
26.0	37	1	0	0	0	0	0	0	0	0	…	
27.0	291	655	8	35	0	0	0	0	137	1	…	
28.0	8	11	43	200	0	0	0	0	40	1	…	
29.0	0	0	0	0	0	0	0	0	0	0	…	
30.0	0	0	0	0	0	0	0	0	0	0	…	
32.0	250	62	0	1	0	0	0	0	0	0	…	
33.0	2	0	0	0	0	0	0	0	0	0	…	
34.0	72	1013	478	1675	0	0	0	0	36	0	…	
35.0	17	0	0	0	0	0	0	0	0	0	…	
36.0	65	6	0	0	0	0	0	0	0	0	…	
37.0	0	0	0	0	0	0	0	0	0	0	…	
39.0	0	0	0	0	0	0	0	0	0	0	…	
40.0	0	0	0	0	0	0	0	0	0	0	…	
42.0	177	2628	401	1127	2	0	0	0	19	1	…	
43.0	36	886	1	40	0	0	0	0	0	0	…	
44.0	7	155	0	0	0	0	0	0	0	0	…	
45.0	1	0	0	0	0	0	0	0	0	0	…	
46.0	0	34	1	0	0	0	0	0	0	0	…	
47.0	32	96	0	0	0	0	0	0	0	0	…	
48.0	0	1	0	0	0	0	0	0	0	0	…	
49.0	0	0	0	0	0	0	0	0	0	0	…	
50.0	0	0	0	0	0	0	0	0	0	0	…	

TO WARD	40.0	42.0	43.0	44.0	45.0	46.0	47.0	48.0	49.0	50.0
FROM WARD										
1.0	0	284	44	7	0	0	4	0	0	0
2.0	1	2759	791	122	0	23	84	3	1	0
3.0	0	478	5	0	0	0	0	0	0	0
4.0	0	1502	52	0	0	1	0	0	0	0
5.0	0	2	1	0	0	0	0	0	0	0
6.0	0	0	0	0	0	0	0	0	0	0
7.0	0	0	0	0	0	0	0	0	0	0
8.0	0	0	0	0	0	0	0	0	0	0
11.0	0	6	1	0	0	0	0	0	0	0
12.0	0	0	0	0	0	0	0	0	0	0
15.0	0	0	0	0	0	0	0	0	0	0
16.0	0	0	0	0	0	0	0	0	0	0
17.0	0	0	0	0	0	0	0	0	0	0
20.0	0	0	0	0	0	0	0	0	0	0
22.0	0	0	0	0	0	0	0	0	0	0
24.0	0	0	0	0	0	0	0	0	0	0
25.0	0	17	5	0	0	0	0	0	0	0
26.0	0	0	3	0	0	0	0	0	0	0
27.0	0	1949	271	48	0	17	5	1	0	0
28.0	0	387	2	2	0	0	0	0	0	0
29.0	0	0	0	0	0	0	0	0	0	0
30.0	0	0	0	0	2	0	0	0	0	0
32.0	8	259	648	225	0	29	588	3	1	0
33.0	22	0	0	1	14	6	60	0	0	1
34.0	0	5596	263	53	0	2	0	0	0	0
35.0	2	0	5	2	3	2	13	0	0	0
36.0	0	87	1	0	0	0	0	0	0	0
37.0	0	0	0	0	0	0	0	0	0	0
39.0	8	0	0	0	0	0	8	14	1	0
40.0	91	0	1	13	0	13	104	61	44	2
42.0	0	11558	1017	108	0	42	11	2	1	0
43.0	1	1036	2323	613	0	340	40	89	2	0
44.0	10	79	656	1341	0	866	242	120	2	0
45.0	1	0	0	0	3	0	7	0	0	0
46.0	19	48	356	783	0	1439	483	273	3	0
47.0	161	1	86	231	7	447	1020	145	18	0
48.0	66	7	166	70	0	253	118	417	51	3
49.0	52	0	0	1	0	3	11	36	564	3
50.0	1	0	0	0	0	0	3	2	3	1

[39 rows x 38 columns]

……

[15]:
```
# Drop null values
df_MM_clean_2017 = df_MM_clean_2017.dropna(subset=['BIRTH YEAR'])
```

[16]:
```
# Convert BIRTH YEAR to integer
df_MM_clean_2017['BIRTH YEAR'] = df_MM_clean_2017['BIRTH YEAR'].astype('int')
```

[17]:
```
# Create age groups based on the agr distribution of users
df_MM_clean_2017['age'] = df_MM_clean_2017['YEAR'] - df_MM_clean_2017['BIRTH␣
 ↪YEAR']
```

[18]:
```
# Define age groups
age_bins = [0, 18, 30, 40, 50, 60, float('inf')]  # Define age bins/ranges
age_labels = ['0-18', '19-30', '31-40', '41-50', '51-60', '61+']  # Define␣
 ↪corresponding labels
```

[19]:
```
# Create 'age group' column using pd.cut
df_MM_clean_2017['age group'] = pd.cut(df_MM_clean_2017['age'], bins=age_bins,␣
 ↪labels=age_labels, right=False)
```

[20]:
```
age_group_under_18_2017 = df_MM_clean_2017[df_MM_clean_2017['age group'] ==␣
 ↪'0-18']
# OD Matrix
matrix_age_under_18_2017 = (
    age_group_under_18_2017.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_age_under_18_2017
```

[20]:

TO WARD	1.0	2.0	3.0	4.0	5.0	11.0	12.0	15.0	20.0	24.0	…	\
FROM WARD											…	
1.0	35	3	0	0	0	0	0	0	0	0	…	
2.0	2	23	0	3	0	0	0	0	0	0	…	
3.0	0	0	1	45	0	1	0	0	0	0	…	
4.0	0	2	35	75	8	1	0	0	1	0	…	
5.0	0	0	0	11	87	0	0	0	3	0	…	
7.0	0	0	0	0	1	0	0	0	0	0	…	
11.0	0	0	0	0	0	3	0	0	0	0	…	
12.0	0	0	0	0	0	0	1	2	0	0	…	
16.0	0	0	0	0	0	0	0	0	1	0	…	
20.0	0	0	0	2	0	0	0	0	0	0	…	
24.0	0	0	0	0	0	0	0	0	0	4	…	
25.0	0	0	3	1	0	0	0	0	0	0	…	
26.0	2	0	0	0	0	0	0	0	0	0	…	

27.0	3	3	0	3	0	0	0	0	0	1 …
28.0	0	0	5	18	0	0	0	0	0	2 …
29.0	0	0	0	0	0	0	0	0	0	1 …
30.0	0	0	0	0	0	0	0	0	0	0 …
32.0	8	2	0	0	0	0	0	0	0	0 …
33.0	0	0	0	0	0	0	0	0	0	0 …
34.0	0	4	2	16	0	0	0	0	0	1 …
35.0	1	0	0	0	0	0	0	0	0	0 …
36.0	3	1	0	0	0	0	0	0	0	0 …
37.0	0	0	0	0	0	0	0	0	0	0 …
39.0	0	0	0	0	0	0	0	0	0	0 …
40.0	0	0	0	0	0	0	0	0	0	0 …
42.0	13	68	0	43	0	0	0	0	0	0 …
43.0	3	24	0	1	0	0	0	0	0	0 …
44.0	0	1	0	0	0	0	0	0	0	0 …
45.0	0	0	0	0	0	0	0	0	0	0 …
46.0	0	0	0	0	0	0	0	0	0	0 …
47.0	0	0	0	0	0	0	0	0	0	0 …
48.0	0	0	0	0	0	0	0	0	0	0 …
49.0	0	0	0	0	0	0	0	0	0	0 …
50.0	0	0	0	0	0	0	0	0	0	0 …

TO WARD	40.0	42.0	43.0	44.0	45.0	46.0	47.0	48.0	49.0	50.0
FROM WARD										
1.0	0	6	3	0	0	0	0	0	0	0
2.0	0	59	15	2	0	0	1	0	0	0
3.0	0	0	0	0	0	0	0	0	0	0
4.0	0	21	0	0	0	0	0	0	0	0
5.0	0	0	0	0	0	0	0	0	0	0
7.0	0	0	0	0	0	0	0	0	0	0
11.0	0	0	0	0	0	0	0	0	0	0
12.0	0	0	0	0	0	0	0	0	0	0
16.0	0	0	0	0	0	0	0	0	0	0
20.0	0	0	0	0	0	0	0	0	0	0
24.0	0	0	0	0	0	0	0	0	0	0
25.0	0	10	0	0	0	0	0	0	0	0
26.0	0	0	0	0	0	0	0	0	0	0
27.0	0	15	23	0	0	0	0	0	0	0
28.0	0	42	0	0	0	0	0	0	0	0
29.0	0	0	0	0	0	0	0	0	0	0
30.0	0	0	0	0	0	0	0	0	0	0
32.0	0	0	18	16	0	2	6	0	0	0
33.0	1	0	0	1	0	1	0	0	0	0
34.0	0	40	3	0	0	0	1	0	0	0
35.0	0	0	0	0	0	0	1	0	0	0
36.0	0	1	0	0	0	0	0	0	0	0
37.0	0	0	0	0	0	0	0	0	0	0

```
39.0          3     1     0     0     0     0     0     4     1     0
40.0          0     0     0     0     0     0     1     1     0     1
42.0          0    92    10     1     0     1     0     0     0     0
43.0          0    38    21     9     0     1     4     1     0     0
44.0          0     1    14    52     0     0    40     2     0     0
45.0          0     0     0     0     1     0     0     0     0     0
46.0          0     0     1     0     0    15     3     7     1     0
47.0          2     0     3    51     1     2    22     0     3     0
48.0          2     0     0     2     0     5     1    35     1     0
49.0          0     0     0     0     0     1     0     3    46     1
50.0          0     0     0     0     0     0     0     0     5     0

[34 rows x 31 columns]
```

As it is shown, the number of under 18 years old users are low and the most trips happened inside ward 28.

[21]:
```
age_group_19_30_2017 = df_MM_clean_2017[df_MM_clean_2017['age group'] ==␣
 ↪'19-30']
# OD Matrix 2018 calculation using pivot table
matrix_age_19_30_2017 = (
    age_group_19_30_2017.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_age_19_30_2017
```

[21]:
```
TO WARD     1.0    2.0    3.0    4.0    5.0   6.0   7.0   8.0   11.0  12.0  \
FROM WARD
1.0       17739   2541      4    105      0     0     0     0     3     3
2.0        2803  15176    351   1193      9     0     0     0    27     0
3.0           7    365   2641  10080    410     1     0     0  1761    32
4.0         131   1326  11562  19516   4644     0     0     0  2483     5
5.0           1      9    465   4745  18066     7    36    48    10     0
6.0           0      0      2      0     16   444     5   101     0     0
7.0           0      0      0      0     33     2   361    50     0     0
8.0           0      0      0      1     52    98    55   139     0     0
11.0          7     26   1491   2073      8     0     0     0  5127   190
12.0          0      0     28     10      0     0     0     0   158   140
15.0          1      0      6      1      0     0     0     0    17    21
16.0          7      0      0      2      5    10     0     0     7    14
17.0          0      0      0      0      0     0     0     0     0     0
20.0          0      0    178    231    792    13     1     4    42    24
22.0          0      0      0      0      0     0     0     0     0     0
```

24.0	2	0	0	4	0	0	0	0	1	2
25.0	233	101	208	444	3	0	0	0	506	98
26.0	2600	97	0	8	0	0	0	0	0	1
27.0	7566	10005	77	692	1	0	0	0	105	5
28.0	256	259	439	1482	0	0	0	0	609	3
29.0	0	0	0	1	0	0	0	0	0	0
30.0	24	0	0	1	0	0	0	0	0	0
32.0	6130	1510	1	23	0	0	0	0	3	0
33.0	128	3	0	0	0	0	0	0	0	0
34.0	4457	7042	3367	7522	11	0	0	0	1167	6
35.0	2272	66	0	0	0	0	0	0	0	0
36.0	2578	442	0	20	0	0	0	0	1	0
37.0	0	0	0	0	0	0	0	0	0	0
39.0	18	2	0	0	0	0	0	0	0	0
40.0	37	0	0	0	0	0	0	0	0	0
42.0	7678	27081	2013	9485	93	0	0	0	542	2
43.0	3226	15165	65	419	6	0	0	0	4	0
44.0	1076	2979	5	32	1	0	0	0	0	0
45.0	101	13	0	0	0	0	0	0	0	0
46.0	106	735	4	14	0	0	0	0	1	0
47.0	579	213	0	0	0	0	0	0	0	0
48.0	17	65	0	1	0	0	0	0	0	0
49.0	50	7	0	0	0	0	0	0	0	0
50.0	1	0	0	1	0	0	0	0	0	0

TO WARD	…	40.0	42.0	43.0	44.0	45.0	46.0	47.0	48.0	49.0	\
FROM WARD	…										
1.0	…	37	7552	3402	1082	99	170	832	20	13	
2.0	…	2	24534	17745	3810	6	1058	262	101	4	
3.0	…	0	1856	64	8	0	5	1	1	0	
4.0	…	0	9665	553	52	0	17	1	8	1	
5.0	…	0	60	2	3	0	0	0	0	0	
6.0	…	0	0	0	0	0	0	0	0	0	
7.0	…	0	0	0	0	0	0	0	0	0	
8.0	…	0	0	0	0	0	0	0	0	0	
11.0	…	0	424	2	1	0	0	0	0	0	
12.0	…	0	0	0	0	0	0	0	0	0	
15.0	…	0	0	0	0	0	0	0	0	0	
16.0	…	0	0	0	0	0	0	0	0	0	
17.0	…	0	0	0	0	0	0	0	0	0	
20.0	…	0	0	0	0	0	0	0	0	0	
22.0	…	0	0	0	0	0	0	0	0	0	
24.0	…	0	8	1	0	0	0	0	0	0	
25.0	…	0	800	29	7	0	4	1	0	0	
26.0	…	13	512	138	43	6	6	63	1	0	
27.0	…	4	19756	9516	2185	2	333	186	11	4	
28.0	…	0	2643	183	24	2	1	4	0	0	

29.0	…	0	0	0	0	1	0	0	0	1
30.0	…	1	1	7	11	52	1	23	1	0
32.0	…	120	1371	12316	7030	147	983	2121	126	46
33.0	…	117	1	46	131	170	149	354	53	3
34.0	…	0	31751	3618	704	0	90	16	7	1
35.0	…	39	76	262	253	152	44	211	9	1
36.0	…	0	1031	198	72	3	25	63	2	1
37.0	…	0	0	0	0	1	0	0	0	0
39.0	…	113	0	2	3	64	20	162	80	4
40.0	…	449	2	69	101	8	277	1068	537	468
42.0	…	10	67588	20363	3587	2	780	143	90	5
43.0	…	48	15275	54209	15503	7	5296	1790	984	17
44.0	…	128	2756	14617	20737	22	8896	3152	1285	64
45.0	…	13	1	10	23	80	18	78	20	1
46.0	…	305	616	4705	8642	14	15010	3452	1912	152
47.0	…	1145	92	1639	3218	94	3560	10007	1464	199
48.0	…	597	61	913	1157	25	1598	1408	2830	820
49.0	…	355	12	20	43	0	125	179	671	2914
50.0	…	64	1	0	1	0	1	19	186	38

TO WARD	50.0
FROM WARD	
1.0	1
2.0	0
3.0	0
4.0	0
5.0	0
6.0	0
7.0	0
8.0	0
11.0	0
12.0	0
15.0	0
16.0	0
17.0	0
20.0	0
22.0	0
24.0	0
25.0	0
26.0	1
27.0	1
28.0	0
29.0	0
30.0	0
32.0	0
33.0	1
34.0	0

```
35.0          0
36.0          0
37.0          0
39.0          6
40.0         45
42.0          0
43.0          2
44.0          2
45.0          0
46.0          2
47.0         30
48.0        226
49.0         55
50.0          8

[39 rows x 39 columns]
```

Because the siz of the matrix is too large, it is hard to conclude here, so we will wait untile the next part for further analysis.

[22]:
```
age_group_31_40_2017 = df_MM_clean_2017[df_MM_clean_2017['age group'] ==␣
 ↪'31-40']
# OD Matrix 2018 calculation using pivot table
matrix_age_31_40_2017 = (
    age_group_31_40_2017.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_age_31_40_2017
```

[22]:
```
TO WARD     1.0    2.0   3.0   4.0   5.0   6.0   7.0   8.0   11.0  12.0  … \
FROM WARD                                                              …
1.0        15564   1728    28   223     0     0     0     0    14     1  …
2.0         1940  12849   802  1382     7     0     0     0    49     2  …
3.0           21    719  2382  3234   429     0     0     0   720    17  …
4.0          225   1605  3741  8920  3663     0     1     0  1073     1  …
5.0            2      6   378  3599  9537     5    19    74    17     1  …
6.0            0      0     1     0     1   250     0    39     1     0  …
7.0            0      0     0     2    19     0    45     9     0     0  …
8.0            0      0     0     0    75    45     1    73     0     0  …
11.0          17     16   590   932    18     2     0     0  3712   124  …
12.0           0      0    12     0     0     0     0     0   139    48  …
15.0           0      0     0     1     2     0     0     0    19    24  …
16.0           0      0     0     1     0     4     0     0     1     2  …
```

17.0	0	0	0	0	0	0	0	0	0	0	…
20.0	0	0	386	82	275	10	0	2	5	1	…
22.0	0	0	0	0	0	0	0	0	0	2	…
24.0	2	0	0	0	0	0	0	0	0	4	…
25.0	191	123	469	456	5	0	0	0	591	136	…
26.0	2516	160	2	4	0	0	0	0	3	2	…
27.0	6430	8281	171	618	3	0	0	0	121	13	…
28.0	246	277	560	1138	1	0	0	0	407	9	…
29.0	0	0	0	0	0	0	0	0	0	0	…
30.0	102	1	0	0	0	0	0	0	0	0	…
32.0	7606	1544	3	109	0	0	0	0	2	0	…
33.0	218	8	0	2	0	0	0	0	0	0	…
34.0	4205	9417	6059	7659	13	0	0	0	894	28	…
35.0	2366	74	0	5	0	0	0	0	2	1	…
36.0	3259	264	5	17	0	0	0	0	6	0	…
37.0	0	0	0	0	0	0	0	0	0	0	…
39.0	70	0	0	0	0	0	0	0	0	0	…
40.0	37	15	0	0	0	0	0	0	0	0	…
42.0	8392	28669	4266	9422	54	0	0	0	764	5	…
43.0	1982	10664	41	364	3	0	0	0	8	0	…
44.0	519	2011	5	27	0	0	0	0	1	0	…
45.0	79	0	0	0	0	0	0	0	0	0	…
46.0	106	1399	1	19	0	0	0	0	0	0	…
47.0	532	352	0	38	0	0	0	0	0	0	…
48.0	19	116	3	1	0	0	0	0	0	0	…
49.0	38	6	0	0	0	0	0	0	0	0	…
50.0	0	0	0	0	0	0	0	0	0	0	…

TO WARD	40.0	42.0	43.0	44.0	45.0	46.0	47.0	48.0	49.0	50.0
FROM WARD										
1.0	17	7293	2023	580	58	114	639	21	90	0
2.0	10	24487	12291	2652	1	1438	296	118	9	0
3.0	0	3983	72	8	0	2	0	0	0	0
4.0	2	9837	484	56	1	23	22	8	0	0
5.0	0	51	6	2	0	3	0	1	0	0
6.0	0	0	0	0	0	0	0	0	0	0
7.0	0	0	0	0	0	0	0	0	0	0
8.0	0	0	0	0	0	0	0	0	0	0
11.0	0	687	16	0	0	0	0	0	0	0
12.0	0	1	0	0	0	0	0	0	0	0
15.0	0	0	0	0	0	0	0	0	0	0
16.0	0	0	0	0	0	0	0	0	0	0
17.0	0	0	0	0	0	0	0	0	0	0
20.0	0	1	1	0	0	0	0	0	0	0
22.0	0	0	0	0	0	0	0	0	0	0
24.0	0	2	2	0	0	0	0	0	0	0
25.0	0	1699	36	22	0	0	2	0	0	0

```
26.0          6    294    148     14      0      3    131      0      0      0
27.0          8  24559   5864   1645      2    414    493     18      6      0
28.0          0   3260     58     90      0      7      4      0      0      0
29.0          0      1      0      0      0      0      0      0      0      0
30.0          2      2      2     17    203      2     29      1      0      0
32.0        145   2846   7338   4858     67    853   3846     75      8      1
33.0        310      4    105    153    128    235    950     37      4      3
34.0          5  45858   3456    810      0    102     83     12      7      0
35.0         66    280    509    284    203     55    425     13      0      0
36.0          3   1440    234     56      1     10     97      2      0      0
37.0          0      0      0      0      0      0      0      0      0      0
39.0         89      0      5     12     68     19    206     19      7      3
40.0       1170     19     91    383     36    351   3152   1486    586     99
42.0         15  83880  15447   3061      6    973    412    280     12      2
43.0        112  11406  29066   8679      6   4603   1841   1225     59      2
44.0        324   1993   7293  12520     24   7453   3423   1037     92      5
45.0         25      1      1     16    845     42    219      7      2      0
46.0        333    610   4061   6631     30  14801   4387   2571    193      5
47.0       3333    202   1566   3181    234   4253  16479   2760    187     26
48.0       1371     71    928    918      4   1911   2343   4880    821     44
49.0        684      5     12     41      1    185    218    736   4821     59
50.0         99      0      0      2      0      4     28     55     59      6

[39 rows x 39 columns]
```

[23]:
```
age_group_41_50_2017 = df_MM_clean_2017[df_MM_clean_2017['age group'] ==␣
 ↪'41-50']
# OD Matrix 2018 calculation using pivot table
matrix_age_41_50_2017 = (
    age_group_41_50_2017.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_age_41_50_2017
```

[23]:
```
TO WARD    1.0    2.0   3.0   4.0   5.0   6.0   7.0   8.0   11.0  12.0  … \
FROM WARD                                                              …
1.0       3548    295     4    34     0     0     0     0     2     0  …
2.0        314   3761   141   390     3     0     0     0     1     0  …
3.0         11     72   950  1107   292     0     0     2   209     3  …
4.0         18    380  1375  3834   580     0     0     1   144     0  …
5.0          2      2   196   617  3625     0    16    26    18     0  …
6.0          0      0     0     1     1    18     1    26     0     0  …
7.0          0      0     0     0    17     0    97     1     0     0  …
```

8.0	0	0	2	0	25	10	1	42	0	0	…
11.0	2	8	172	304	4	0	0	0	863	61	…
12.0	1	0	4	2	0	0	0	0	38	32	…
15.0	1	0	3	15	0	0	0	0	149	9	…
16.0	0	0	0	0	0	1	0	0	7	0	…
17.0	0	0	0	0	0	0	0	0	0	0	…
20.0	0	0	47	28	144	5	0	8	10	6	…
22.0	0	0	0	4	0	0	0	0	0	1	…
24.0	0	0	1	0	0	0	0	0	1	0	…
25.0	45	25	158	144	2	0	0	0	225	31	…
26.0	656	14	0	2	0	0	0	0	0	0	…
27.0	1448	2341	91	407	0	0	0	0	26	109	…
28.0	57	68	140	421	1	0	0	0	123	12	…
29.0	0	0	0	0	0	0	0	0	0	0	…
30.0	48	0	0	0	0	0	0	0	0	0	…
32.0	1854	464	0	42	0	0	0	0	0	0	…
33.0	38	0	0	0	0	0	0	0	0	0	…
34.0	1169	4303	2358	4678	5	0	0	0	179	70	…
35.0	603	3	0	0	0	0	0	0	0	0	…
36.0	1179	176	9	15	0	0	0	0	0	0	…
37.0	0	0	0	0	0	0	0	0	0	0	…
39.0	6	0	0	0	0	0	0	0	0	0	…
40.0	7	0	0	14	0	0	0	0	0	0	…
42.0	2235	10162	1619	4169	21	0	1	0	110	2	…
43.0	462	3502	8	310	3	0	0	0	0	1	…
44.0	105	691	3	10	1	0	0	0	0	0	…
45.0	14	1	0	0	0	0	0	0	0	0	…
46.0	27	387	2	4	2	0	0	0	1	0	…
47.0	278	61	0	6	0	0	0	0	0	0	…
48.0	35	17	0	3	1	0	0	0	0	0	…
49.0	0	4	0	0	0	0	0	0	0	0	…
50.0	0	1	0	0	1	0	0	0	0	0	…

TO WARD	40.0	42.0	43.0	44.0	45.0	46.0	47.0	48.0	49.0	50.0
FROM WARD										
1.0	7	2196	353	130	16	38	252	30	1	0
2.0	3	8840	3798	775	0	524	110	30	3	0
3.0	0	1647	14	3	0	0	0	0	0	0
4.0	1	4564	224	19	1	6	2	5	0	1
5.0	0	26	3	2	0	1	0	0	0	0
6.0	0	3	0	0	0	0	0	0	0	0
7.0	0	0	0	0	0	0	0	0	0	0
8.0	0	1	0	0	0	0	0	0	0	0
11.0	0	134	4	2	0	0	0	0	0	0
12.0	0	0	0	0	0	0	0	0	0	0
15.0	0	0	0	0	0	0	0	0	0	0
16.0	0	0	0	0	0	0	0	0	0	0

```
17.0          0      0     0     0     0     0     0     0     0     0
20.0          0      1     0     0     0     0     0     0     0     0
22.0          0      1     0     0     0     0     0     0     0     0
24.0          0      2     0     0     0     0     0     0     0     0
25.0          0    446    20     2     0     0     0     1     0     0
26.0          1    212    54     4     3     2    17     1     0     0
27.0          2  11144  1491   333     1    79    31     6     0     1
28.0          0   1365    15     3     0     0     2     0     0     0
29.0          0      0     0     0     0     0     0     0     0     0
30.0          1      0     2     2    16     0    21     0     0     1
32.0         32   1043  3360  1630    13   343  1555    51     3     0
33.0         51      0    20    15    26    40   241    25     1     1
34.0          0  26390  1742   259    10    42    31     2     3     0
35.0         30     11    60    36    50    11   193    12     3     1
36.0          1    434    31    27     1     6     7     6     0     0
37.0          0      0     0     0     0     0     0     0     0     0
39.0         54      0     0     3    11     9    86    25     2     3
40.0        480     15    64    51    10    69  1586   495   499    28
42.0         11  42411  5617  1038     3   272   135    44     7     1
43.0         20   5604  9199  2727     1  1144   785   480    21     1
44.0        128    883  2363  4375     5  2438  1357   483    95     2
45.0          2      6     2     8    95     7    27     0     0     0
46.0        103    252  1144  2476    13  6197  1236  1036    49     0
47.0       1794     93   680  1330    27  1377  7224   945    70    29
48.0        538     27   357   435     1   802   819  2226   430    31
49.0        488      1    11    15     0    41    59   366  3252    46
50.0         27      1     2     1     0     1    15    13    28     8

[39 rows x 39 columns]
```

[24]:
```
age_group_51_60_2017 = df_MM_clean_2017[df_MM_clean_2017['age group'] ==␣
 ↪'51-60']
# OD Matrix 2018 calculation using pivot table
matrix_age_51_60_2017 = (
    age_group_51_60_2017.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_age_51_60_2017
```

[24]:
```
TO WARD    1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   11.0  12.0  … \
FROM WARD                                                              …
1.0        1018   233     3     6     0     0     0     0     0     0  …
2.0         189  3727   136   407     7     0     0     0     4     0  …
```

3.0	1	106	513	1220	198	0	0	0	200	2	…
4.0	10	362	1151	3193	1362	1	1	0	156	1	…
5.0	0	8	229	1351	2629	3	39	18	2	0	…
6.0	0	0	0	0	2	80	0	38	0	0	…
7.0	0	0	0	0	22	0	375	13	0	0	…
8.0	0	0	0	0	18	47	5	51	0	0	…
11.0	0	3	135	248	6	0	0	0	710	26	…
12.0	0	0	5	1	0	0	0	0	25	36	…
15.0	0	0	5	0	0	0	0	0	7	12	…
16.0	0	0	20	1	0	2	0	0	4	1	…
20.0	0	0	100	50	199	8	0	6	196	2	…
22.0	0	0	0	0	0	0	0	0	0	0	…
24.0	0	0	0	0	0	0	0	0	0	2	…
25.0	6	6	78	33	0	0	0	0	182	20	…
26.0	285	5	0	0	0	0	0	0	0	0	…
27.0	515	1804	26	283	12	0	0	0	7	0	…
28.0	18	23	77	210	0	0	0	0	86	4	…
29.0	0	0	0	0	0	0	0	0	0	0	…
30.0	9	0	0	0	0	0	0	0	0	0	…
32.0	909	560	1	4	26	0	0	0	0	0	…
33.0	34	29	0	0	0	0	0	0	0	0	…
34.0	195	2851	1566	3196	8	0	0	0	276	0	…
35.0	120	5	0	1	0	0	0	0	0	0	…
36.0	294	45	0	6	0	0	0	0	0	0	…
37.0	0	0	0	0	0	0	0	0	0	0	…
39.0	0	0	0	0	0	0	0	0	0	0	…
40.0	6	6	0	0	0	0	0	0	0	0	…
42.0	365	7636	1200	4014	22	0	0	0	124	0	…
43.0	212	2767	4	94	3	0	0	0	4	0	…
44.0	65	499	0	7	0	0	0	0	0	0	…
45.0	0	0	0	0	0	0	0	0	0	0	…
46.0	8	161	0	1	0	0	0	0	0	0	…
47.0	57	74	0	0	0	0	0	0	0	0	…
48.0	4	87	0	0	0	0	0	0	0	0	…
49.0	0	0	0	1	0	0	0	0	0	0	…
50.0	0	0	0	0	0	0	0	0	0	0	…

TO WARD	40.0	42.0	43.0	44.0	45.0	46.0	47.0	48.0	49.0	50.0
FROM WARD										
1.0	2	307	250	59	4	6	43	2	0	0
2.0	5	6801	2992	675	0	187	58	57	0	0
3.0	0	1059	12	2	0	0	0	0	0	0
4.0	0	4162	113	15	0	7	1	0	0	0
5.0	0	29	2	0	0	0	0	0	0	0
6.0	0	0	0	0	0	0	0	0	0	0
7.0	0	0	0	0	0	0	0	0	0	0
8.0	0	0	0	0	0	0	0	0	0	0

11.0	0	156	1	0	0	0	1	0	0	0
12.0	0	2	0	0	0	0	0	0	0	0
15.0	0	0	0	0	0	0	0	0	0	0
16.0	0	0	0	0	0	0	0	0	0	0
20.0	0	0	0	0	0	0	0	0	0	0
22.0	0	0	0	0	0	0	0	0	0	0
24.0	0	0	0	0	0	0	0	0	0	0
25.0	0	73	0	0	0	0	0	0	0	0
26.0	1	30	19	8	0	0	3	2	0	0
27.0	0	6398	1000	285	0	57	48	5	0	0
28.0	1	895	15	2	0	0	0	0	0	0
29.0	0	0	0	0	1	0	0	0	0	0
30.0	0	0	2	3	11	0	2	0	0	0
32.0	42	500	1941	1005	6	205	882	19	2	0
33.0	115	1	11	10	32	14	230	10	0	3
34.0	2	21794	861	87	0	10	32	6	0	0
35.0	50	16	14	14	10	4	51	4	0	0
36.0	0	273	11	3	0	3	10	0	0	0
37.0	0	0	0	0	0	0	0	0	0	0
39.0	36	0	2	0	32	2	35	4	4	0
40.0	1280	0	22	26	6	77	835	392	134	64
42.0	1	34192	3746	605	1	141	74	16	0	0
43.0	15	3391	7254	1902	0	716	473	266	0	0
44.0	24	372	1500	2693	5	1788	1019	323	38	8
45.0	3	0	1	12	44	5	22	0	0	0
46.0	66	138	763	1498	2	3808	1064	800	44	0
47.0	747	34	422	721	19	1024	4159	606	76	8
48.0	580	12	225	261	1	672	470	1797	401	52
49.0	235	0	1	25	0	58	57	415	2575	30
50.0	30	0	0	6	0	2	21	35	28	10

[38 rows x 38 columns]

[25]:
```
age_group_above_61_2017 = df_MM_clean_2017[df_MM_clean_2017['age group'] ==␣
 ↪'61+']
# OD Matrix 2018 calculation using pivot table
matrix_age_above_61_2017 = (
    age_group_above_61_2017.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_age_above_61_2017
```

[25]:

TO WARD	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	11.0	12.0	…	\
FROM WARD											…	
1.0	422	34	0	3	0	0	0	0	0	0	…	
2.0	49	1508	18	98	0	0	0	0	0	0	…	
3.0	1	20	256	645	94	0	0	0	56	3	…	
4.0	4	93	403	1575	442	0	0	0	32	1	…	
5.0	0	0	73	418	1413	1	4	1	0	0	…	
6.0	0	0	0	0	0	2	1	0	0	0	…	
7.0	0	0	0	0	3	0	3	0	0	0	…	
8.0	0	0	0	0	0	0	0	3	0	0	…	
11.0	0	0	71	57	0	0	0	0	231	34	…	
12.0	0	0	84	2	0	0	0	0	47	8	…	
15.0	0	0	0	0	0	0	0	0	2	2	…	
17.0	0	0	0	0	0	0	0	0	0	0	…	
20.0	0	0	9	15	10	0	1	0	1	0	…	
22.0	0	0	0	0	0	0	0	0	0	0	…	
24.0	0	0	0	0	0	0	0	0	1	5	…	
25.0	7	0	29	54	0	0	0	0	76	78	…	
26.0	61	2	0	0	0	0	0	0	0	0	…	
27.0	247	789	6	38	0	0	0	0	4	0	…	
28.0	10	16	39	237	0	0	0	0	23	1	…	
29.0	0	1	0	0	0	0	0	0	0	0	…	
30.0	2	0	0	0	0	0	0	0	0	0	…	
32.0	189	48	0	0	0	0	0	0	0	0	…	
33.0	3	0	0	0	0	0	0	0	0	0	…	
34.0	79	1434	435	1533	0	0	0	0	20	0	…	
35.0	61	2	0	0	0	0	0	0	0	0	…	
36.0	61	4	0	0	0	0	0	0	0	0	…	
37.0	1	0	0	0	0	0	0	0	0	0	…	
39.0	0	0	0	0	0	0	0	0	0	0	…	
40.0	0	0	0	0	0	0	0	0	0	0	…	
42.0	180	3109	455	1317	22	0	0	0	5	0	…	
43.0	39	1057	2	19	0	0	0	0	0	0	…	
44.0	8	222	0	7	0	0	0	0	0	0	…	
45.0	4	0	0	0	0	0	0	0	0	0	…	
46.0	3	40	0	1	0	0	0	0	0	0	…	
47.0	10	74	0	0	0	0	0	0	0	0	…	
48.0	0	3	0	2	0	0	0	0	0	0	…	
49.0	0	0	0	0	0	0	0	0	0	0	…	
50.0	0	0	0	0	0	0	0	0	0	0	…	

TO WARD	40.0	42.0	43.0	44.0	45.0	46.0	47.0	48.0	49.0	50.0
FROM WARD										
1.0	0	150	28	6	4	2	14	0	0	0
2.0	0	3212	1009	232	0	31	64	6	0	0
3.0	0	447	3	0	0	0	0	0	0	0
4.0	0	1555	50	3	0	0	0	1	0	0

5.0	0	18	0	0	0	0	0	0	0	0
6.0	0	0	0	0	0	0	0	0	0	0
7.0	0	0	0	0	0	0	0	0	0	0
8.0	0	0	0	0	0	0	0	0	0	0
11.0	0	7	0	0	0	0	0	0	0	0
12.0	0	0	0	0	0	0	0	0	0	0
15.0	0	0	0	0	0	0	0	0	0	0
17.0	0	0	0	0	0	0	0	0	0	0
20.0	0	0	0	0	0	0	0	0	0	0
22.0	0	0	0	0	0	0	0	0	0	0
24.0	0	0	0	0	0	0	0	0	0	0
25.0	0	18	4	1	0	0	0	0	0	0
26.0	0	5	6	0	0	0	1	0	0	0
27.0	0	2066	366	29	0	64	13	8	0	0
28.0	0	454	0	5	0	0	0	0	0	0
29.0	0	0	0	0	0	0	0	0	0	0
30.0	0	0	2	0	5	0	2	0	0	0
32.0	7	133	723	171	2	31	571	2	0	0
33.0	20	0	1	1	47	5	61	1	1	0
34.0	0	7184	358	66	0	3	0	2	0	0
35.0	1	3	4	4	15	1	62	0	0	0
36.0	0	75	5	0	0	0	0	0	0	0
37.0	0	0	0	0	0	0	0	0	0	0
39.0	10	0	0	0	3	0	8	13	0	1
40.0	196	0	5	14	20	27	172	139	71	36
42.0	0	13327	1263	127	0	31	0	7	0	0
43.0	6	1271	2387	784	0	229	97	125	0	0
44.0	12	119	707	1499	1	882	347	284	0	0
45.0	7	0	0	0	22	2	7	0	0	0
46.0	31	30	309	809	1	1878	537	358	8	0
47.0	265	2	119	337	6	440	1310	110	10	2
48.0	137	4	146	188	0	395	78	464	104	7
49.0	65	0	0	0	0	1	4	128	927	16
50.0	25	0	0	0	0	0	2	3	15	9

[38 rows x 38 columns]

……

[26]:
```
# Drop null values
df_MM_clean_2018 = df_MM_clean_2018.dropna(subset=['BIRTH YEAR'])
```

[27]:
```
# Convert BIRTH YEAR to integer
df_MM_clean_2018['BIRTH YEAR'] = df_MM_clean_2018['BIRTH YEAR'].astype('int')
```

[31]:
```
# Create age groups based on the agr distribution of users
```

```
df_MM_clean_2018['age'] = df_MM_clean_2017['YEAR'] - df_MM_clean_2017['BIRTH␣
 ↪YEAR']
```

[32]:
```
# Define age groups
age_bins = [0, 18, 30, 40, 50, 60, float('inf')]  # Define age bins/ranges
age_labels = ['0-18', '19-30', '31-40', '41-50', '51-60', '61+']  # Define␣
 ↪corresponding labels
```

[33]:
```
# Create 'age group' column using pd.cut
df_MM_clean_2018['age group'] = pd.cut(df_MM_clean_2018['age'], bins=age_bins,␣
 ↪labels=age_labels, right=False)
```

[34]:
```
age_group_under_18_2018 = df_MM_clean_2018[df_MM_clean_2018['age group'] ==␣
 ↪'0-18']
# OD Matrix 2018 calculation using pivot table
matrix_age_under_18_2018 = (
    age_group_under_18_2018.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_age_under_18_2018
```

[34]:

TO WARD	1.0	2.0	3.0	4.0	5.0	8.0	11.0	16.0	20.0	25.0	…	\
FROM WARD											…	
1.0	23	2	1	0	0	0	0	0	0	0	…	
2.0	2	16	1	1	0	0	0	0	0	0	…	
3.0	0	1	3	10	1	0	2	0	0	0	…	
4.0	0	1	10	17	1	0	2	0	0	0	…	
5.0	0	0	0	11	28	0	0	0	0	0	…	
8.0	0	0	0	0	0	0	0	0	1	0	…	
11.0	0	1	1	2	0	0	11	0	0	1	…	
12.0	0	0	0	0	0	0	0	0	0	1	…	
16.0	0	0	0	0	0	1	0	1	0	0	…	
20.0	0	0	0	0	1	0	1	0	3	0	…	
25.0	1	1	1	0	0	0	3	0	0	4	…	
26.0	9	0	0	0	0	0	0	0	0	0	…	
27.0	12	12	1	4	0	0	0	0	0	5	…	
28.0	0	0	1	1	0	0	0	0	0	2	…	
29.0	0	0	0	0	0	0	0	0	0	0	…	
30.0	0	0	0	1	0	0	0	0	0	0	…	
32.0	12	4	0	0	0	0	0	0	0	0	…	
33.0	0	0	0	0	0	0	0	0	0	0	…	
34.0	5	15	8	17	0	0	0	0	0	3	…	
35.0	1	1	0	0	0	0	0	0	0	0	…	

```
36.0          1     1     0     0     0     0     0     0     0     0 …
40.0          0     0     0     0     0     0     0     0     0     0 …
42.0         13    36    12    19     0     0     0     0     0     0 …
43.0          7    22     0     2     0     0     0     0     0     0 …
44.0          0     1     0     1     0     0     0     0     0     0 …
46.0          1     2     0     0     0     0     0     0     0     0 …
47.0          1     2     0     0     0     0     0     0     0     0 …
48.0          0     2     0     0     0     0     0     0     0     0 …
49.0          0     0     0     0     0     0     0     0     0     0 …

TO WARD    36.0  40.0  42.0  43.0  44.0  45.0  46.0  47.0  48.0  49.0
FROM WARD
1.0           2     0     8     1     0     0     0     1     0     0
2.0           0     0    46    21     3     0     1     0     0     0
3.0           0     0     2     0     0     0     0     0     0     0
4.0           0     0    11     1     1     0     0     0     0     0
5.0           0     0     0     1     0     0     0     0     0     0
8.0           0     0     0     0     0     0     0     0     0     0
11.0          0     0     2     0     0     0     0     0     0     0
12.0          0     0     0     0     0     0     0     0     0     0
16.0          0     0     0     0     0     0     0     0     0     0
20.0          0     0     0     0     0     0     0     0     0     0
25.0          0     0     3     0     0     0     0     0     0     0
26.0          1     0     0     0     0     0     0     0     0     0
27.0          3     0    33    11     3     0     3     1     0     0
28.0          0     0     4     0     0     0     0     0     0     0
29.0          0     0     0     0     0     0     0     0     0     0
30.0          0     0     0     0     0     0     0     0     0     0
32.0          0     0     5    14    11     0     1     4     0     0
33.0          0     1     0     0     0     0     0     2     0     0
34.0          1     0    70     5     0     0     0     0     0     0
35.0          0     0     1     0     0     0     0     1     0     0
36.0          3     0     3     0     0     0     0     0     0     0
40.0          0     2     0     0     1     1     1     4     0     1
42.0          1     0   149    33     3     0     2     2     1     2
43.0          0     0    23    53    12     0     3     1     0     0
44.0          1     0     5    12    13     1    17     4     2     0
46.0          0     0     2     4    11     0    16     4     2     0
47.0          0     3     1     2     5     0     5    20     1     0
48.0          0     1     0     1     0     0     2     1     5     0
49.0          0     2     0     1     0     0     0     0     3     6

[29 rows x 28 columns]
```

[35]:
```
age_group_19_30_2018 = df_MM_clean_2018[df_MM_clean_2018['age group'] ==␣
 ↪'19-30']
# OD Matrix 2018 calculation using pivot table
```

```
matrix_age_19_30_2018 = (
    age_group_19_30_2018.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_age_19_30_2018
```

[35]:

TO WARD	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	11.0	12.0	…	\
FROM WARD											…	
1.0	11006	1266	41	138	3	0	0	0	31	36	…	
2.0	1418	10717	378	1134	57	0	0	0	47	3	…	
3.0	35	355	1853	3849	461	0	0	2	913	27	…	
4.0	159	1175	4362	8888	3609	2	2	3	1001	9	…	
5.0	3	26	424	3640	12776	11	48	43	23	0	…	
6.0	0	0	2	0	15	231	7	35	0	0	…	
7.0	0	1	0	2	40	0	319	9	0	0	…	
8.0	0	0	0	5	38	40	20	76	0	0	…	
11.0	23	43	734	940	29	0	0	0	3104	108	…	
12.0	12	1	34	18	0	0	0	0	88	128	…	
15.0	0	0	3	8	6	0	0	0	63	53	…	
16.0	0	0	5	2	2	15	0	2	6	2	…	
17.0	0	0	0	0	0	3	0	0	0	0	…	
20.0	0	0	128	131	464	23	1	16	48	7	…	
22.0	0	3	0	5	0	0	0	0	0	0	…	
24.0	2	0	0	2	0	0	0	0	0	4	…	
25.0	145	125	256	285	23	1	0	2	438	109	…	
26.0	1613	78	0	8	1	0	0	0	0	0	…	
27.0	5354	6613	202	682	13	0	0	0	139	52	…	
28.0	270	219	394	912	12	1	0	0	314	22	…	
29.0	1	2	0	0	0	0	0	0	1	0	…	
30.0	30	2	0	0	0	0	0	0	0	0	…	
32.0	4506	1268	7	50	3	0	0	0	4	1	…	
33.0	114	19	0	0	0	0	0	0	0	0	…	
34.0	3139	6542	4185	7230	50	1	0	0	740	32	…	
35.0	1337	73	0	11	0	0	0	0	0	0	…	
36.0	1756	298	5	65	0	0	0	0	7	0	…	
37.0	3	0	0	0	0	0	0	0	0	0	…	
39.0	7	7	0	0	0	0	0	0	0	0	…	
40.0	31	16	1	1	0	0	0	0	0	0	…	
42.0	5906	21950	2957	8741	222	0	4	0	493	8	…	
43.0	1726	9091	79	516	28	0	0	0	22	0	…	
44.0	602	1940	20	127	10	0	0	0	4	1	…	
45.0	43	4	0	1	0	0	0	0	0	0	…	
46.0	147	921	4	43	0	0	0	0	2	0	…	

47.0	410	317	0	21	0	0	0	0	1	0 …
48.0	32	208	6	21	3	0	0	0	1	0 …
49.0	18	18	1	0	0	0	0	0	0	0 …
50.0	0	2	0	0	0	0	0	0	0	0 …

TO WARD	40.0	42.0	43.0	44.0	45.0	46.0	47.0	48.0	49.0	50.0
FROM WARD										
1.0	37	5189	1689	577	44	158	429	30	21	0
2.0	21	19454	9919	2346	3	1212	376	205	19	1
3.0	0	2770	121	19	0	6	1	6	0	0
4.0	3	9159	669	163	2	69	26	30	2	0
5.0	0	259	18	2	0	0	1	5	0	0
6.0	0	1	0	0	0	0	0	0	0	0
7.0	0	2	0	0	0	0	0	0	0	0
8.0	0	0	1	1	0	0	0	0	0	0
11.0	0	420	18	4	0	1	0	0	0	0
12.0	0	5	0	0	0	0	0	0	0	0
15.0	0	0	0	0	0	0	0	0	0	0
16.0	0	2	0	0	0	0	0	0	0	0
17.0	0	0	0	0	0	0	0	0	0	0
20.0	0	5	0	0	0	0	0	0	0	0
22.0	0	4	0	0	0	0	0	0	0	0
24.0	0	5	1	0	0	0	0	0	0	0
25.0	1	875	44	10	0	3	2	1	0	0
26.0	6	363	144	40	2	9	26	5	0	0
27.0	19	20546	5291	1378	4	405	363	61	3	0
28.0	2	2442	81	21	2	5	9	4	0	0
29.0	0	2	0	0	2	0	0	0	0	0
30.0	4	1	7	19	137	5	13	2	0	1
32.0	109	2175	7320	4086	35	784	2394	119	19	1
33.0	128	29	47	80	81	120	476	42	5	3
34.0	9	38439	3622	842	6	253	170	38	13	0
35.0	38	261	239	181	134	57	296	25	5	0
36.0	6	1061	153	55	3	16	56	0	1	0
37.0	0	0	1	0	1	0	0	0	0	0
39.0	61	2	7	9	48	29	194	40	5	2
40.0	797	43	129	198	20	244	1846	871	533	39
42.0	25	74245	13911	3642	13	1591	589	378	31	0
43.0	109	11763	27243	8180	5	2963	1423	844	72	6
44.0	189	2472	7444	12027	19	5928	2536	839	89	3
45.0	22	4	12	11	343	22	106	16	1	0
46.0	277	1194	2725	5793	15	11584	2806	1807	171	2
47.0	2074	413	1234	2342	97	2737	10360	1524	125	24
48.0	917	207	714	721	16	1664	1318	3658	638	69
49.0	559	10	81	57	0	118	128	554	4065	63
50.0	29	0	5	5	1	3	22	44	46	14

[39 rows x 39 columns]

[36]:
```
age_group_31_40_2018 = df_MM_clean_2018[df_MM_clean_2018['age group'] ==␣
 ↪'31-40']
# OD Matrix 2018 calculation using pivot table
matrix_age_31_40_2018 = (
    age_group_31_40_2018.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_age_31_40_2018
```

[36]:

TO WARD	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	11.0	12.0	…	\
FROM WARD											…	
1.0	10622	1249	36	147	7	0	0	0	24	40	…	
2.0	1365	10389	348	1122	37	0	0	0	50	3	…	
3.0	43	360	1859	3613	399	4	0	0	832	15	…	
4.0	153	1080	4223	8664	3363	2	6	6	1010	9	…	
5.0	7	13	407	3507	12128	13	61	38	30	1	…	
6.0	0	0	1	0	24	226	2	38	0	0	…	
7.0	0	0	0	2	50	2	287	12	0	0	…	
8.0	0	0	0	7	45	44	11	74	1	0	…	
11.0	32	41	727	875	33	0	0	1	2879	97	…	
12.0	6	1	37	19	2	0	0	0	90	98	…	
15.0	0	0	4	11	6	0	0	0	37	68	…	
16.0	0	0	2	1	2	8	0	2	2	3	…	
17.0	0	0	0	0	0	2	0	0	0	0	…	
20.0	0	1	139	95	457	27	2	17	37	5	…	
22.0	0	2	0	12	0	0	0	0	2	0	…	
24.0	5	1	0	9	1	0	0	0	1	3	…	
25.0	162	111	213	288	19	0	0	0	372	91	…	
26.0	1500	90	1	7	0	0	0	0	3	0	…	
27.0	5090	6353	188	693	12	0	0	0	116	41	…	
28.0	256	205	400	871	4	0	0	0	277	20	…	
29.0	3	0	0	1	0	0	0	0	0	0	…	
30.0	40	2	0	0	0	0	0	0	0	0	…	
32.0	4179	1223	5	53	5	0	0	0	6	0	…	
33.0	148	21	0	0	0	0	0	0	0	0	…	
34.0	2955	6286	3966	6825	56	0	0	1	724	40	…	
35.0	1367	61	0	7	0	0	0	0	0	0	…	
36.0	1702	219	4	55	1	0	0	0	12	1	…	
37.0	2	0	0	0	0	0	0	0	0	0	…	
39.0	9	6	0	0	0	0	0	0	0	0	…	
40.0	26	15	0	2	0	0	0	0	0	0	…	

42.0	5548	21153	2821	8176	203	4	1	0	506	13 …
43.0	1661	8553	63	553	11	0	0	0	15	1 …
44.0	619	1963	12	141	3	0	0	0	7	0 …
45.0	30	5	0	1	0	0	0	0	0	0 …
46.0	144	833	5	31	0	0	0	0	0	1 …
47.0	321	284	0	17	0	0	0	0	0	0 …
48.0	32	184	1	18	1	0	0	0	0	0 …
49.0	18	17	0	2	0	0	0	0	0	0 …
50.0	0	1	0	0	0	0	0	0	0	0 …

TO WARD	40.0	42.0	43.0	44.0	45.0	46.0	47.0	48.0	49.0	50.0
FROM WARD										
1.0	31	5187	1571	556	40	162	427	27	24	2
2.0	21	18249	9543	2231	1	1142	353	170	19	0
3.0	0	2819	126	12	0	5	4	1	0	0
4.0	1	8489	682	148	0	38	19	24	2	0
5.0	0	219	16	7	0	2	1	2	0	0
6.0	0	1	0	0	0	0	0	0	0	0
7.0	0	2	0	0	0	0	0	0	0	0
8.0	0	2	0	0	0	0	0	0	0	0
11.0	0	443	16	5	0	0	2	0	0	0
12.0	0	8	1	0	0	0	0	0	0	0
15.0	0	0	0	0	0	0	0	0	0	0
16.0	0	1	0	0	0	0	0	0	0	0
17.0	0	0	0	0	0	0	0	0	0	0
20.0	0	6	0	0	0	0	0	0	0	0
22.0	0	4	0	0	0	0	0	0	0	0
24.0	0	21	0	0	0	0	0	0	0	0
25.0	0	859	39	6	0	3	3	3	0	0
26.0	5	322	117	28	2	8	29	3	0	0
27.0	17	19623	4965	1346	6	325	301	53	6	0
28.0	0	2311	101	21	3	7	5	2	0	0
29.0	0	5	0	0	0	0	0	0	0	0
30.0	4	4	1	9	136	7	18	2	0	0
32.0	112	2038	7091	3976	33	719	2252	88	24	0
33.0	113	27	41	68	79	103	471	44	4	2
34.0	9	36308	3349	816	8	237	172	42	10	0
35.0	37	271	248	159	136	52	281	24	3	0
36.0	4	953	136	46	3	16	44	1	1	0
37.0	0	0	0	0	0	0	0	0	0	0
39.0	70	3	7	11	38	34	161	31	1	5
40.0	771	40	99	161	22	235	1771	848	494	26
42.0	26	71162	13149	3442	4	1421	540	350	35	1
43.0	88	11087	25704	7633	5	2852	1317	845	73	3
44.0	191	2426	6916	11508	15	5601	2521	768	106	2
45.0	23	5	8	19	334	22	96	15	0	0
46.0	282	1132	2649	5436	21	10730	2624	1774	181	2

```
47.0       2085    415   1173   2218    90   2586  10077  1461   108    24
48.0        849    204    690    714    25   1517   1302  3500   607    55
49.0        551     25     76     65     1    124    124   526  3824    60
50.0         36      1      6      1     0      2     24    46    42     6

[39 rows x 39 columns]
```

[37]:
```
age_group_41_50_2018 = df_MM_clean_2018[df_MM_clean_2018['age group'] ==␣
 ↪'41-50']
# OD Matrix 2018 calculation using pivot table
matrix_age_41_50_2018 = (
    age_group_41_50_2018.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_age_41_50_2018
```

[37]:
```
TO WARD    1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   11.0  12.0  … \
FROM WARD                                                              …
1.0       4266   519    14    42     1     0     0     0    11    13  …
2.0        546  4343   130   424    15     0     0     0    17     1  …
3.0         20   140   754  1490   166     0     0     0   343     8  …
4.0         50   435  1752  3598  1424     3     0     0   390     2  …
5.0          0     8   174  1456  5071     7    27    16    10     0  …
6.0          0     0     1     1     4    92     2    16     0     0  …
7.0          0     0     0     2    22     0    98     5     0     0  …
8.0          0     0     2     2    15    14     5    27     0     0  …
11.0         6    18   272   414    12     1     0     0  1154    47  …
12.0         1     0    12     5     0     0     0     0    46    39  …
15.0         1     0     1     4     2     1     0     0    22    24  …
16.0         0     0     1     0     1     8     0     1     2     0  …
17.0         0     0     0     0     0     0     0     0     0     0  …
20.0         0     0    57    42   180    10     1     3    24     3  …
22.0         0     3     0     5     0     0     0     0     0     0  …
24.0         1     0     0     0     0     0     0     0     1     1  …
25.0        58    33    82   113     3     0     0     0   185    37  …
26.0       652    31     1     6     0     0     0     0     1     0  …
27.0      2116  2574    79   272     4     0     0     0    47    12  …
28.0       107    84   164   305     2     0     0     0   141     6  …
29.0         2     2     0     0     0     0     0     0     0     0  …
30.0        12     1     0     0     0     0     0     0     0     0  …
32.0      1775   495     2    19     2     0     0     0     1     0  …
33.0        35     7     0     0     0     0     0     0     0     0  …
34.0      1152  2597  1552  2776    21     1     0     0   294    17  …
```

35.0	518	30	0	2	0	0	0	0	0	0	…
36.0	681	102	1	19	0	0	0	0	4	1	…
37.0	0	0	0	0	0	0	0	0	0	0	…
39.0	4	3	0	0	0	0	0	0	0	0	…
40.0	15	6	1	0	0	0	0	0	0	0	…
42.0	2391	8583	1178	3351	103	0	0	1	191	3	…
43.0	705	3476	27	216	6	0	0	0	4	0	…
44.0	234	801	5	44	0	0	0	0	1	0	…
45.0	10	1	0	1	0	0	0	0	0	0	…
46.0	79	360	3	23	0	0	0	0	0	0	…
47.0	151	113	0	8	1	0	0	0	0	0	…
48.0	13	68	1	12	2	0	0	0	0	0	…
49.0	15	5	1	0	0	0	0	0	1	0	…
50.0	0	0	0	0	0	0	0	0	0	0	…

TO WARD	40.0	42.0	43.0	44.0	45.0	46.0	47.0	48.0	49.0	50.0
FROM WARD										
1.0	25	2149	701	216	15	62	157	14	9	0
2.0	6	7888	3985	930	3	457	131	84	8	0
3.0	0	1128	41	5	0	5	0	1	0	0
4.0	1	3537	280	53	0	22	7	10	0	0
5.0	0	91	2	1	0	0	0	1	0	0
6.0	0	0	0	0	0	0	0	0	0	0
7.0	0	1	0	0	0	0	0	0	0	0
8.0	0	0	0	0	0	0	0	0	0	0
11.0	0	198	7	1	0	1	0	1	0	0
12.0	0	1	0	0	0	0	0	0	0	0
15.0	0	0	0	0	0	0	0	0	0	0
16.0	0	0	0	0	0	0	0	0	0	0
17.0	0	0	0	0	0	0	0	0	0	0
20.0	0	4	0	0	0	0	0	0	0	0
22.0	0	1	1	0	0	0	0	0	0	0
24.0	0	4	0	0	0	0	0	0	0	0
25.0	0	359	20	5	0	0	1	1	1	0
26.0	0	123	52	21	2	2	16	4	0	1
27.0	5	8324	1927	534	3	137	149	17	1	0
28.0	1	927	36	10	0	0	2	0	0	0
29.0	0	0	0	0	0	0	0	0	0	0
30.0	1	2	2	7	59	2	8	0	0	0
32.0	44	851	2860	1637	11	304	922	51	6	0
33.0	52	12	26	37	42	34	183	22	2	0
34.0	3	15200	1326	333	4	96	61	16	5	0
35.0	16	108	97	82	54	21	102	6	1	0
36.0	0	381	51	16	1	2	18	0	0	0
37.0	0	0	0	0	0	0	0	0	0	0
39.0	29	0	3	5	23	13	62	14	1	2
40.0	318	18	40	68	10	96	758	334	208	21

```
42.0          15  29632   5259   1419     4   635   226   131    16     1
43.0          38   4547  10589   3243     2  1160   529   329    24     1
44.0          74    980   2948   4747     8  2270   957   298    41     2
45.0          14      1      3      7   136    13    45    10     0     0
46.0          89    444   1130   2246     8  4549  1140   726    50     5
47.0         812    151    462    917    29  1117  4215   584    54    13
48.0         357     79    250    269    14   579   558  1406   241    24
49.0         212      6     36     27     1    49    52   217  1666    18
50.0          11      0      4      0     0     4     9    18    22     6

[39 rows x 39 columns]
```

[38]:
```
age_group_51_60_2018 = df_MM_clean_2018[df_MM_clean_2018['age group'] ==␣
 ↪'51-60']
# OD Matrix 2018 calculation using pivot table
matrix_age_51_60_2018 = (
    age_group_51_60_2018.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_age_51_60_2018
```

[38]:
```
TO WARD    1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   11.0  12.0  … \
FROM WARD                                                              …
1.0        2826   322     9    40     0     0     0     0     8    10  …
2.0         369  2921   108   292    15     0     0     0     9     0  …
3.0           9    77   501  1083   114     1     0     0   227     3  …
4.0          44   305  1180  2427   951     0     2     4   263     2  …
5.0           0     5    98  1016  3439     3    13    13     5     0  …
6.0           0     0     0     1     4    57     0    16     0     0  …
7.0           0     0     0     0     8     0    69     2     0     0  …
8.0           0     0     0     1    14    11     5    20     0     0  …
11.0          7    14   204   298     8     0     0     0   801    37  …
12.0          6     0     7     8     1     0     0     0    26    38  …
15.0          0     0     0     4     1     0     0     0     9    18  …
16.0          0     0     0     1     1     7     0     0     3     1  …
17.0          0     0     0     0     0     1     0     0     0     0  …
20.0          0     0    27    35   120     1     2     2     6     1  …
22.0          0     2     0     3     0     0     0     0     0     0  …
24.0          0     0     1     3     0     0     0     0     0     0  …
25.0         51    30    57    58     4     0     0     0   108    16  …
26.0        475    17     0     0     0     0     0     0     0     0  …
27.0       1440  1742    54   170     3     0     0     0    43    24  …
28.0         81    52   108   237     2     0     0     0    88     2  …
```

29.0	0	0	0	0	0	0	0	0	0	0	…
30.0	11	1	0	0	0	0	0	0	0	0	…
32.0	1226	372	2	15	2	0	0	0	0	0	…
33.0	52	5	0	0	0	0	0	0	0	0	…
34.0	854	1743	1147	1832	13	0	0	0	210	8	…
35.0	370	18	0	0	0	0	0	0	0	0	…
36.0	484	59	2	7	0	0	0	0	2	0	…
37.0	0	0	0	0	0	0	0	0	0	0	…
39.0	2	0	0	0	0	0	0	0	0	0	…
40.0	10	6	0	0	0	0	0	0	0	0	…
42.0	1558	5762	747	2267	52	1	2	0	120	0	…
43.0	472	2349	18	165	8	0	0	0	8	0	…
44.0	157	527	3	26	0	0	0	0	0	0	…
45.0	10	0	0	0	0	0	0	0	0	0	…
46.0	40	227	0	12	1	0	0	0	1	0	…
47.0	67	86	0	4	0	0	0	0	0	0	…
48.0	8	59	0	2	1	0	0	0	0	0	…
49.0	8	2	0	1	0	0	0	0	0	0	…
50.0	0	1	0	0	0	0	0	0	0	0	…

TO WARD	40.0	42.0	43.0	44.0	45.0	46.0	47.0	48.0	49.0	50.0
FROM WARD										
1.0	10	1355	466	154	11	43	124	10	7	0
2.0	2	5253	2519	599	1	308	120	67	7	0
3.0	0	737	28	4	0	3	2	0	0	0
4.0	0	2329	199	46	0	12	7	5	2	0
5.0	0	78	3	1	0	0	0	0	0	0
6.0	0	0	0	0	0	0	0	0	0	0
7.0	0	0	0	0	0	0	0	0	0	0
8.0	0	0	0	0	0	0	0	0	0	0
11.0	0	130	3	0	0	0	1	0	0	0
12.0	0	1	0	0	0	0	0	0	0	0
15.0	0	0	0	0	0	0	0	0	0	0
16.0	0	1	0	0	0	0	0	0	0	0
17.0	0	0	0	0	0	0	0	0	0	0
20.0	0	0	0	0	0	0	0	0	0	0
22.0	0	1	0	0	0	0	0	0	0	0
24.0	0	5	0	0	0	0	0	0	0	0
25.0	0	227	13	1	0	1	1	0	0	0
26.0	0	91	28	7	1	3	8	1	0	0
27.0	4	5527	1389	388	1	76	88	13	0	0
28.0	0	643	22	3	1	3	0	0	0	0
29.0	0	0	0	0	0	0	0	0	0	0
30.0	2	0	1	2	37	0	3	1	0	0
32.0	29	579	1897	1081	11	218	631	33	5	2
33.0	23	7	8	18	19	23	132	11	3	0
34.0	3	10119	912	210	0	65	39	6	1	0

```
35.0          9     76    54    52    38    21    65     8     0     0
36.0          1    285    33     9     0     6    14     0     0     0
37.0          0      0     0     0     0     0     0     0     0     0
39.0         11      2     1     4    14     7    37    12     2     1
40.0        227      9    36    49    12    59   500   245   135    15
42.0          6  19778  3548   927     1   417   156    95    10     0
43.0         29   3142  6975  2051     3   757   389   224    22     2
44.0         52    649  1859  3156     2  1585   653   197    27     1
45.0          7      3     2     4    98     6    26     7     0     0
46.0         69    336   754  1566     4  3003   717   501    39     0
47.0        517    113   361   658    21   697  2670   421    36     7
48.0        217     54   194   163     5   423   323  1001   169    16
49.0        151      1    22    14     0    23    34   156  1090    17
50.0          9      0     3     1     0     3    10    10    16     3

[39 rows x 39 columns]
```

[39]:
```
age_group_above_61_2018 = df_MM_clean_2018[df_MM_clean_2018['age group'] ==␣
 ↪'61+']
# OD Matrix 2018 calculation using pivot table
matrix_age_above_61_2018 = (
    age_group_above_61_2018.assign(count=1)
    .pivot_table(index='FROM WARD', columns='TO WARD',
                 values="count", aggfunc="count")
    .fillna(0)
    .astype(int)
).sort_values('FROM WARD')

matrix_age_above_61_2018
```

[39]:
```
TO WARD    1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   11.0  12.0  … \
FROM WARD                                                              …
1.0       1102   130     7    20     1     0     0     0     3     5  …
2.0        137  1091    41   121     5     0     1     0     6     0  …
3.0          5    39   191   371    48     0     0     0    84     0  …
4.0         15   124   421   893   347     0     0     1   102     0  …
5.0          1     0    30   364  1265     1     8     5     4     0  …
6.0          0     0     1     0     1    19     0     1     0     0  …
7.0          0     0     1     0     6     0    27     2     0     0  …
8.0          0     0     0     0     6     2     2     5     0     0  …
11.0         3     3    83    86     4     0     0     0   307     6  …
12.0         0     0     7     2     0     0     0     0    10     9  …
15.0         0     0     0     1     0     0     0     0     3     6  …
16.0         0     0     0     0     0     1     0     0     0     0  …
17.0         0     0     0     0     0     1     0     0     0     0  …
20.0         0     0    13    14    54     2     0     2     2     0  …
22.0         0     0     0     1     0     0     0     0     0     0  …
```

24.0	0	1	0	2	0	0	0	0	0	0 …
25.0	17	9	23	34	3	0	0	0	42	9 …
26.0	149	6	0	4	0	0	0	0	0	0 …
27.0	554	696	17	74	2	0	0	0	17	7 …
28.0	25	18	39	97	0	0	0	0	28	4 …
29.0	0	1	0	0	0	0	0	0	0	0 …
30.0	2	1	0	0	0	0	0	0	0	0 …
32.0	498	123	2	5	0	0	0	0	0	0 …
33.0	17	0	0	0	0	0	0	0	0	0 …
34.0	315	628	364	749	9	0	0	0	86	1 …
35.0	148	7	0	1	0	0	0	0	0	0 …
36.0	182	31	0	2	0	0	0	0	0	0 …
37.0	0	0	0	0	0	0	0	0	0	0 …
39.0	1	2	0	0	0	0	0	0	0	0 …
40.0	2	1	0	2	0	0	0	0	0	0 …
42.0	606	2098	298	907	17	0	0	0	50	1 …
43.0	167	919	6	61	2	0	0	0	1	0 …
44.0	66	210	1	13	1	0	0	0	0	0 …
45.0	5	1	0	0	0	0	0	0	0	0 …
46.0	19	102	0	4	0	0	0	0	0	0 …
47.0	45	25	0	3	0	0	0	0	0	0 …
48.0	6	20	0	2	0	0	0	0	0	0 …
49.0	2	2	0	0	0	0	0	0	0	0 …
50.0	0	1	0	0	0	0	0	0	0	0 …

TO WARD	40.0	42.0	43.0	44.0	45.0	46.0	47.0	48.0	49.0	50.0
FROM WARD										
1.0	1	529	162	61	4	14	54	1	3	0
2.0	1	1925	1018	255	0	118	37	18	3	0
3.0	0	287	14	1	0	0	0	0	0	0
4.0	0	867	61	10	1	6	0	2	0	0
5.0	0	20	0	0	0	0	0	0	0	0
6.0	0	0	0	0	0	0	0	0	0	0
7.0	0	0	0	0	0	0	0	0	0	0
8.0	0	0	0	0	0	0	0	0	0	0
11.0	0	39	3	1	0	0	0	0	0	0
12.0	0	0	0	0	0	0	0	0	0	0
15.0	0	0	0	0	0	0	0	0	0	0
16.0	0	0	0	0	0	0	0	0	0	0
17.0	0	0	0	0	0	0	0	0	0	0
20.0	0	0	0	0	0	0	0	0	0	0
22.0	0	0	0	0	0	0	0	0	0	0
24.0	0	1	0	0	0	0	0	0	0	0
25.0	0	94	3	1	0	1	0	0	0	0
26.0	0	35	17	7	0	0	2	0	0	0
27.0	4	2128	516	145	0	41	33	4	0	0
28.0	0	253	8	4	0	1	0	0	0	0

```
29.0          0     0     0     0     0     0     0     0     0     0
30.0          0     0     0     1    11     1     0     0     0     0
32.0         15   249   738   396     5    87   232    14     0     0
33.0         21     4     4    10     6    13    44     9     1     0
34.0          4  3861   352    74     3    21    15     4     2     0
35.0          1    31    27    17    16     9    41     1     0     0
36.0          0   102    18     2     0     0     3     0     0     0
37.0          0     0     0     0     0     0     0     0     0     0
39.0         10     1     1     0     6     2    22     2     1     0
40.0         77     3    15    18     2    34   196    91    52     0
42.0          0  7697  1396   367     0   148    59    38     5     0
43.0         14  1178  2683   804     1   272   133    97     8     0
44.0         14   255   749  1195     1   556   287    87    11     1
45.0          2     0     2     0    32     2     7     0     0     0
46.0         34   116   260   622     2  1162   265   193    20     0
47.0        190    38   118   230     5   279  1040   123    12     1
48.0         90    26    74    69     3   153   139   350    69     5
49.0         51     3    12     9     0     3    15    56   401     6
50.0          5     0     1     1     0     0     0     2     6     2

[39 rows x 39 columns]
```

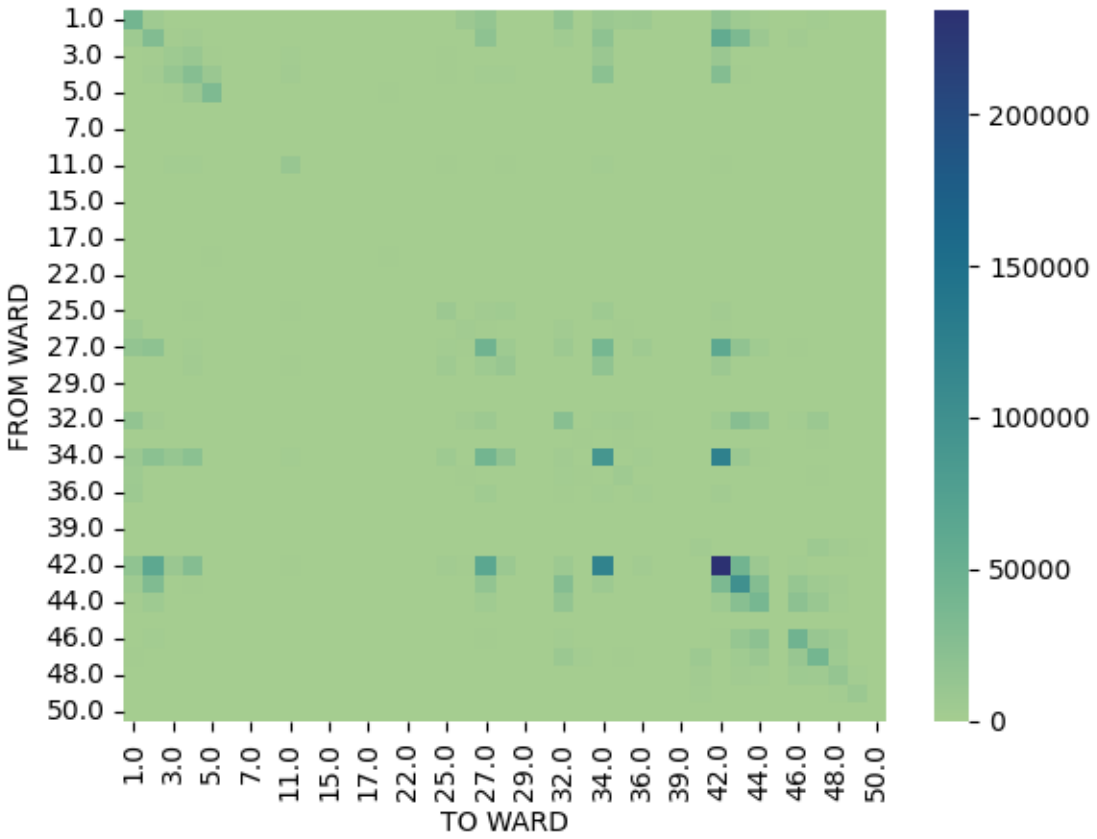
……

- Based on observation, visualise selected OD matrices that show some trends/periodicity on a map.

[40]:
```
import seaborn as sns
```

[43]:
```
# Heatmap general 2016
sns.heatmap(matrix_2016,cmap='crest')
```

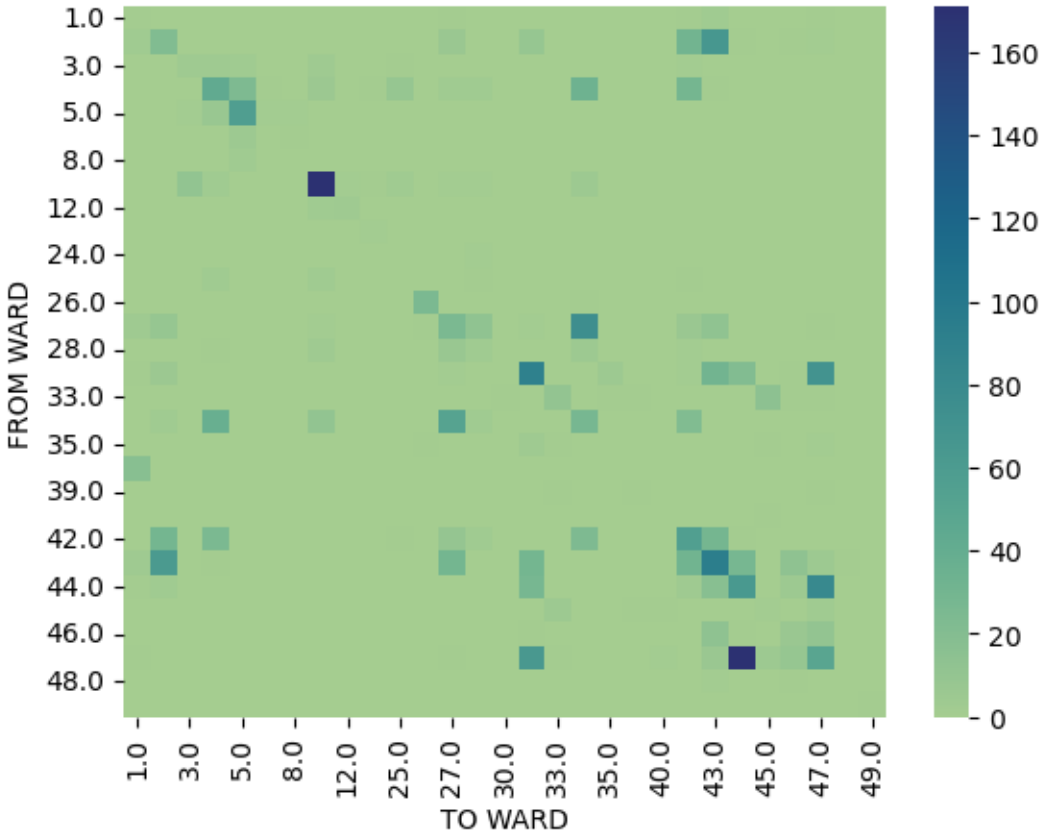
[43]:
```
<AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>
```



It shows that in general in 2016, most of the trips happened in ward 42. After that there are wards 34 and 43 and also the trips among these three wards.

[61]:
```
# Heatmap 0-18 in 2016
sns.heatmap(matrix_age_under_18_2016,cmap='crest')
```

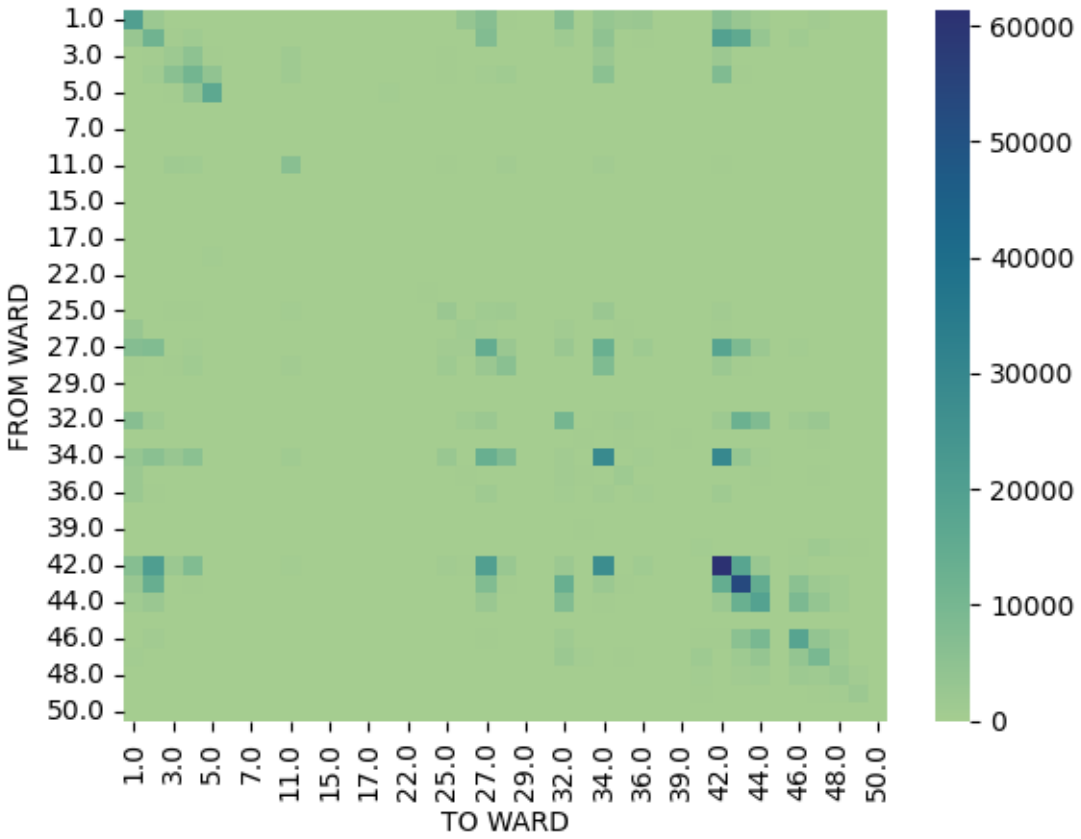
[61]: <AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>



It shows that most of the teenagers' trips (under 18) in 2016 happened in ward 10 and 44. after that there are wards 27, 30, etc.

[44]:
```
# Heatmap 19-30 in 2016
sns.heatmap(matrix_age_19_30_2016,cmap='crest')
```

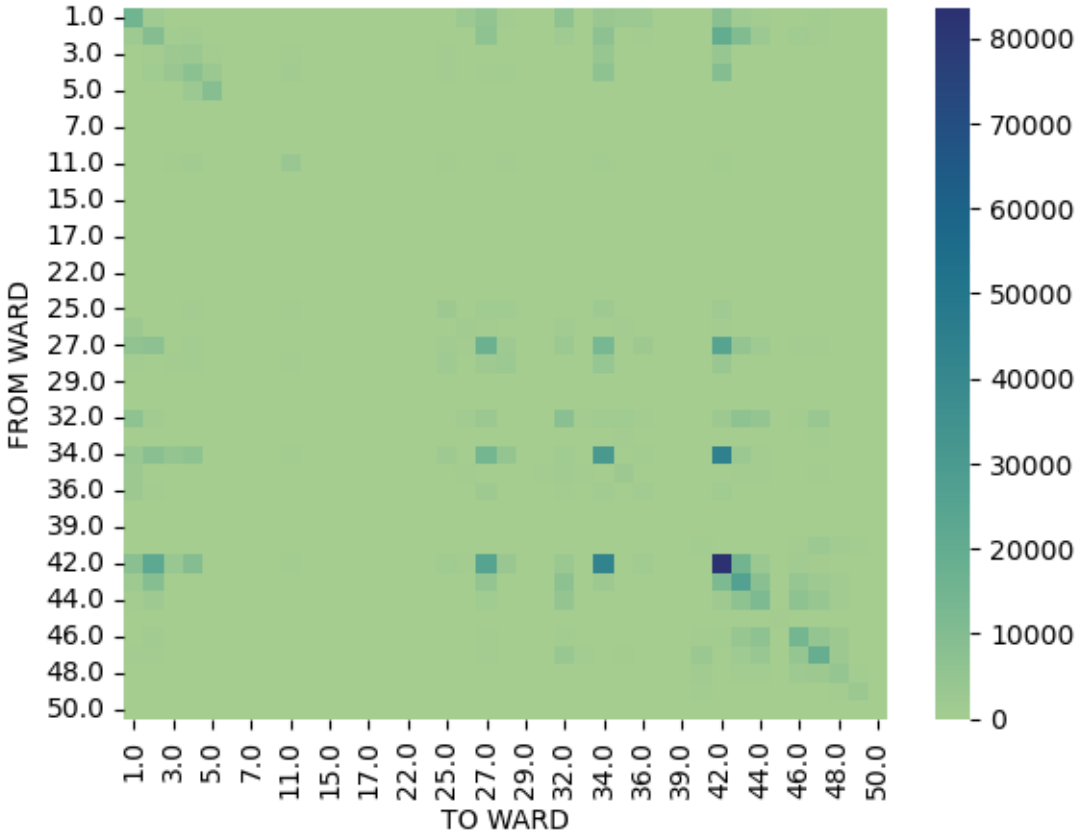
[44]:
```
<AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>
```



It shows that most of the young people's trips (19-30) in 2016 happened in wards 42 and 43.

[45]:
```
# Heatmap 31-40 in 2016
sns.heatmap(matrix_age_31_40_2016,cmap='crest')
```

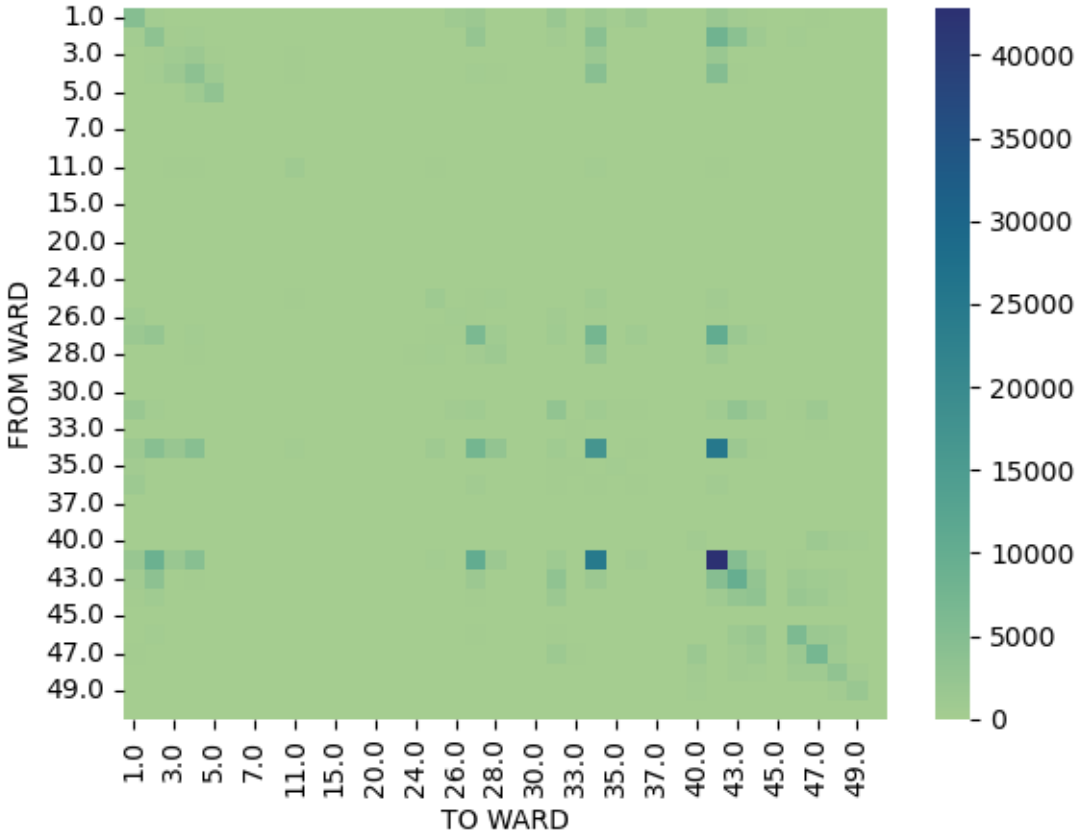
[45]: <AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>



It shows that most of 31-40 people's trips in 2016 happened in ward 42. After that there are wards 43, 34 and 27 and also the trips among these wards.

[46]:
```
# Heatmap 41-50 in 2016
sns.heatmap(matrix_age_41_50_2016,cmap='crest')
```

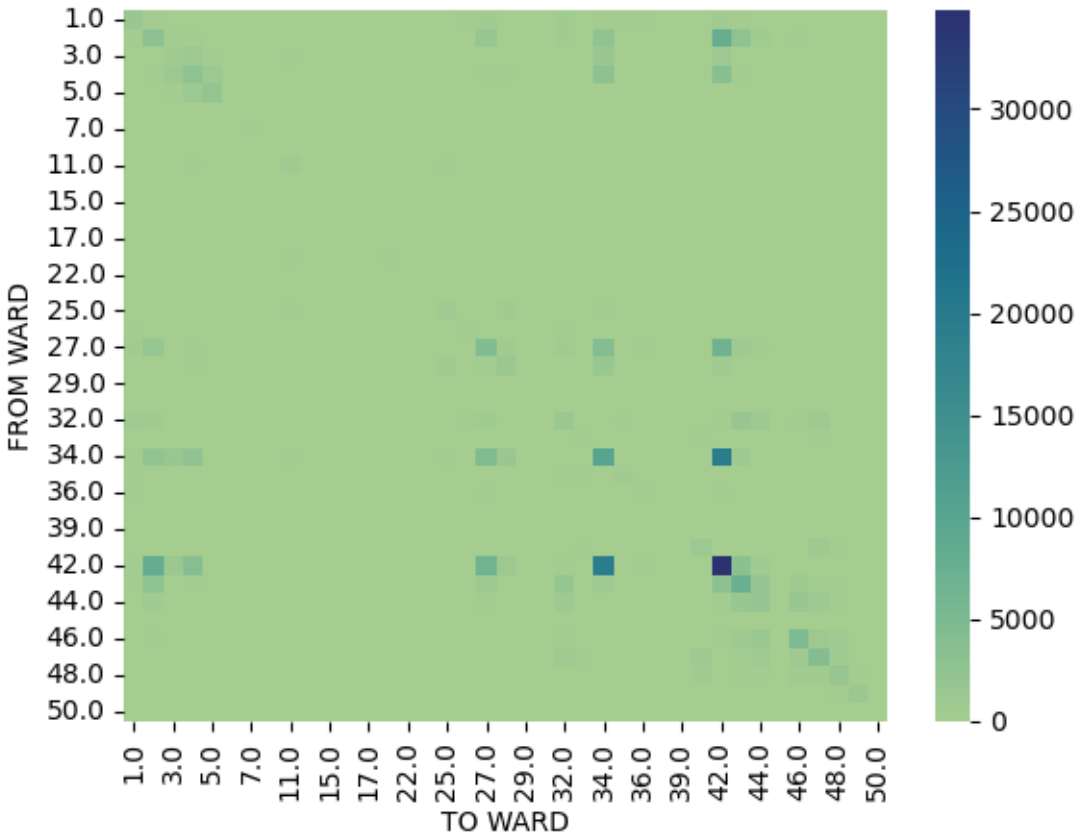
[46]:
```
<AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>
```



Almost the same pattern exists for 41-50 travelers.

[47]:
```
# Heatmap 51-60 in 2016
sns.heatmap(matrix_age_51_60_2016,cmap='crest')
```

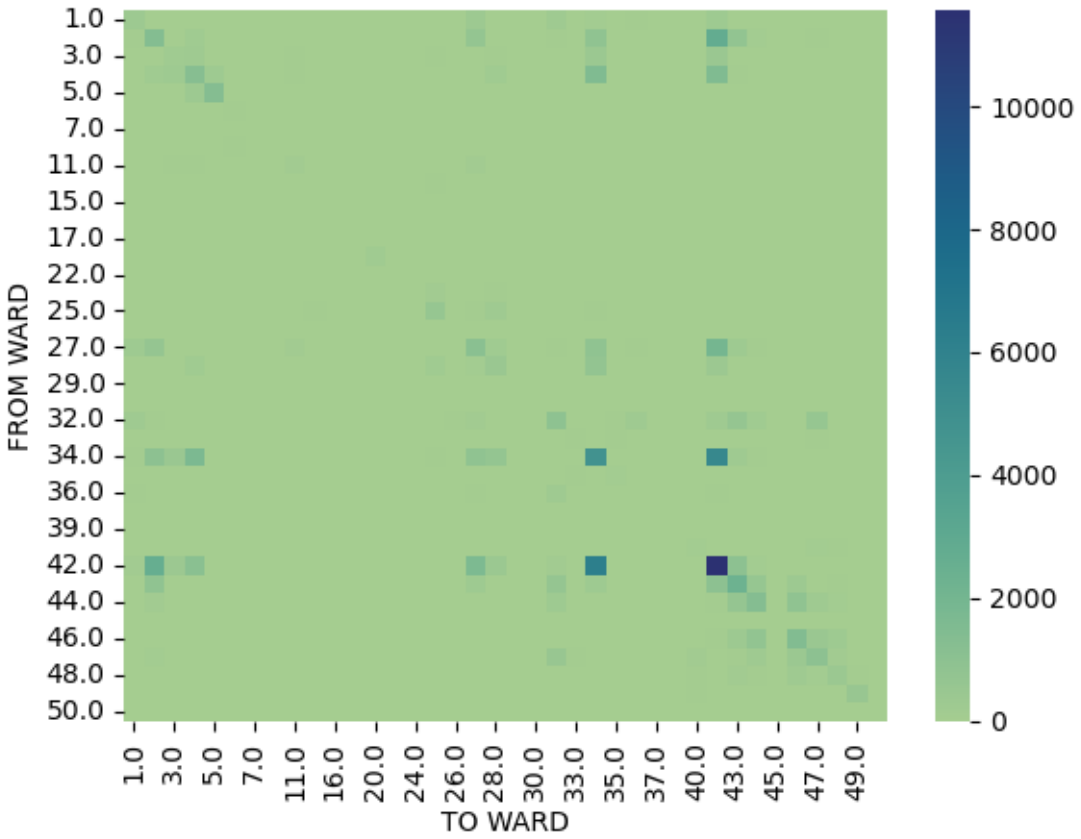
[47]:
```
<AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>
```



Almost the same pattern exists for 51-60 travelers.

[48]:
```
# Heatmap above 61 in 2016
sns.heatmap(matrix_age_above_61_2016,cmap='crest')
```

[48]:
```
<AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>
```



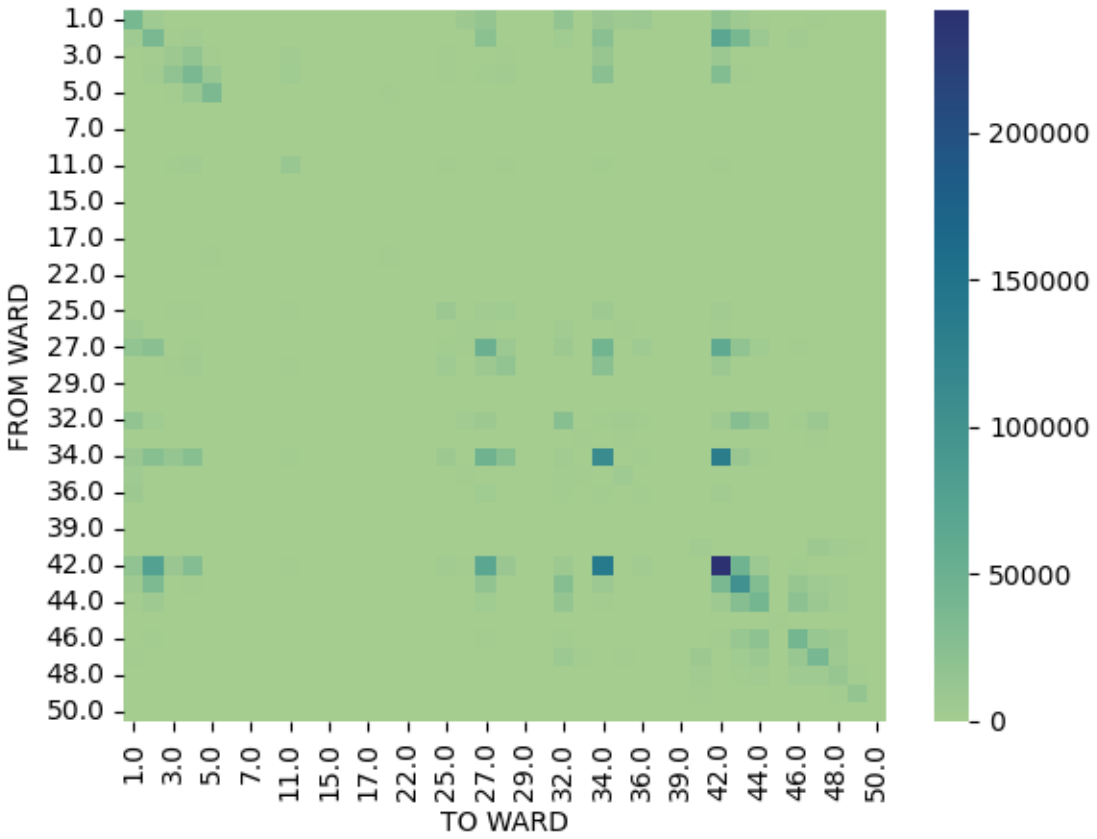
Almost the same pattern exists for above 61 years old travelers.

In conclusion, we can say that wards 42, 34, and 27 are the most important wards for most of the age groups but ward 5 is only attractive for young people and most of their trips happened there.

……

[78]:
```
# Heatmap general 2017
sns.heatmap(matrix_2017,cmap='crest')
```

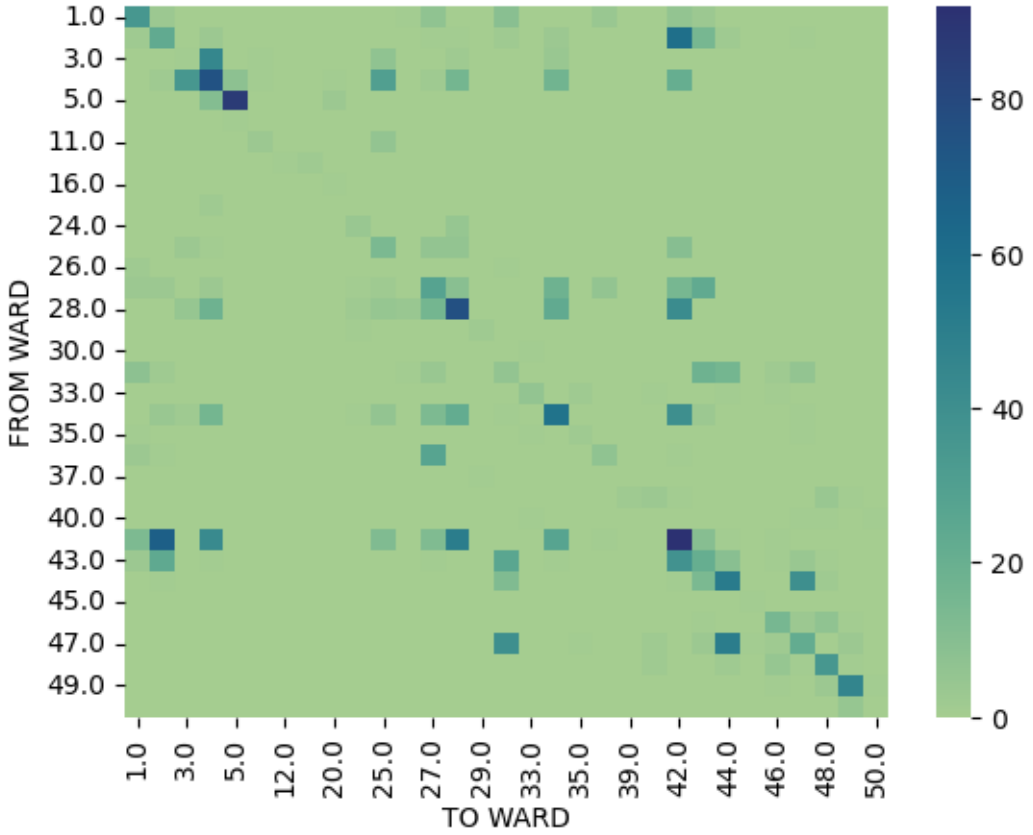
[78]:
```
<AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>
```



In general, for whole 2017, the most trips happened inside ward 42 and after that from ward 42 to ward 34. insode ward 34 and from ward 34 to 42 are comming after them.

[49]:
```
# Heatmap 0-18 in 2017
sns.heatmap(matrix_age_under_18_2017,cmap='crest')
```

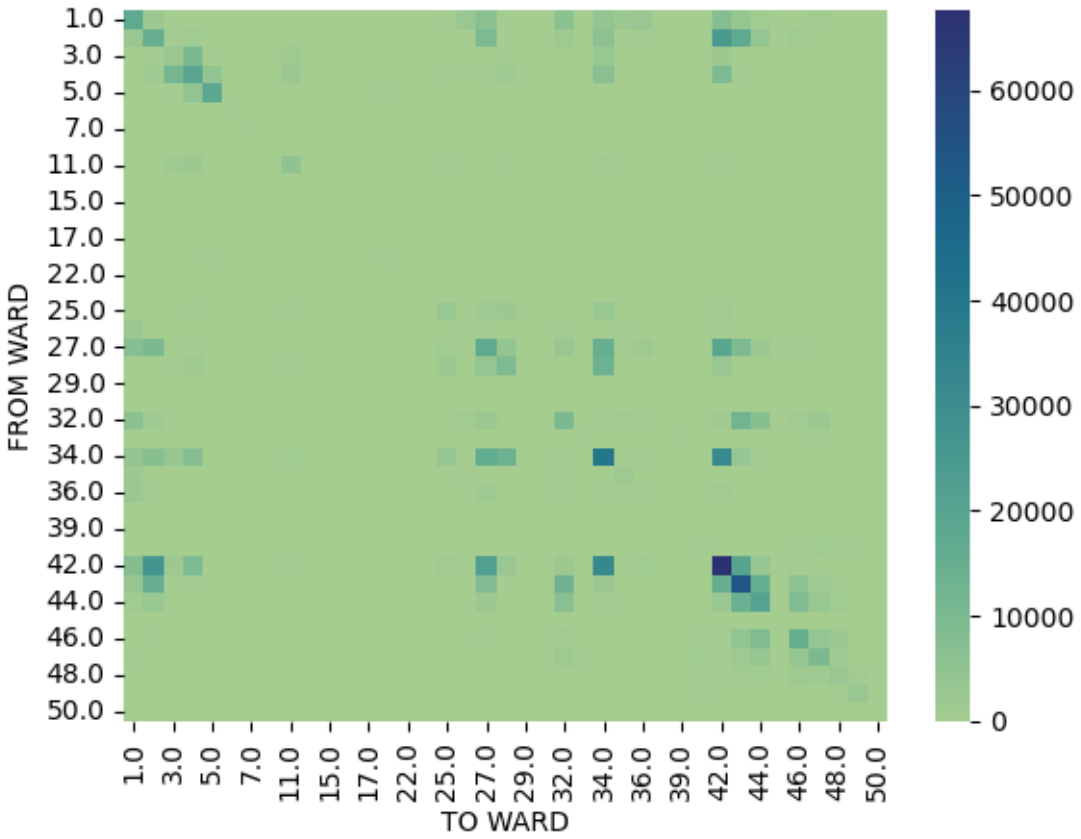
[49]: <AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>



It shows that in 2017, teenagers' trips mostly happened inside the ward 5 and 42. also from ward 42 to 2.

[50]:
```
# Heatmap 19-30 in 2017
sns.heatmap(matrix_age_19_30_2017,cmap='crest')
```

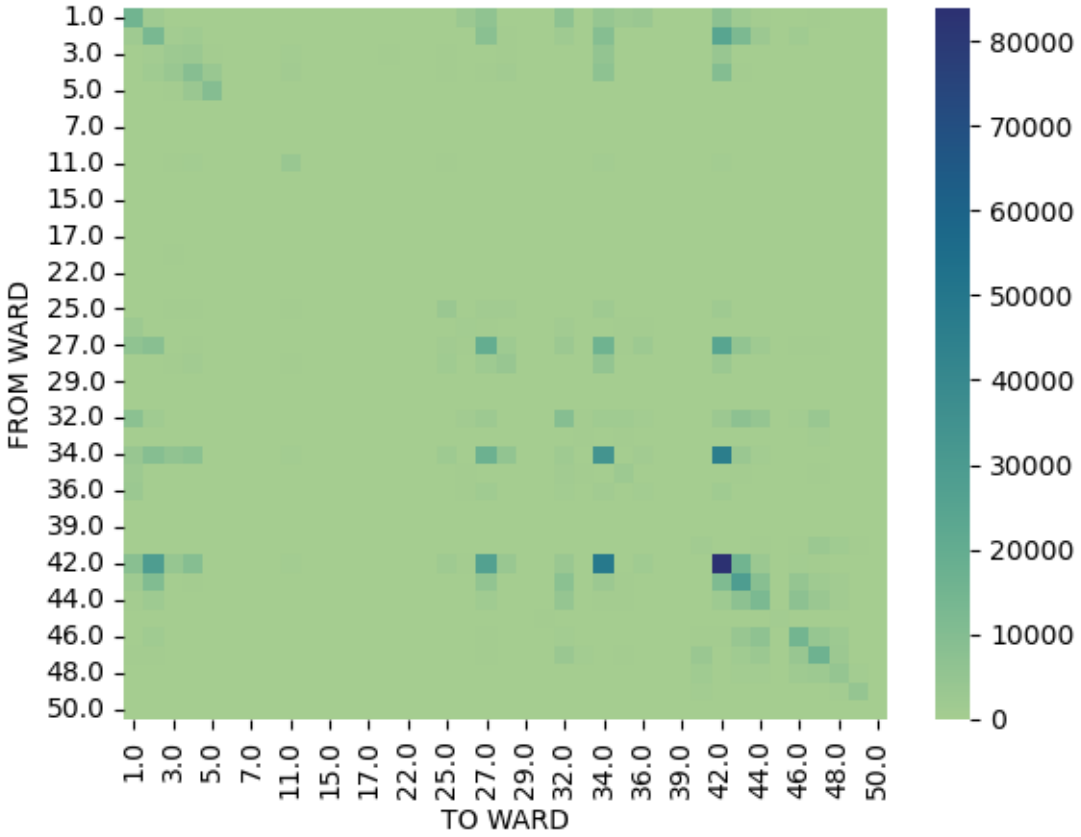
[50]: <AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>



It shows that most of the yung people's trips (19-30) in 2017 happened in ward 5 and after that there are ward 42, 43, and 34.

[51]:
```
# Heatmap 31-40 in 2017
sns.heatmap(matrix_age_31_40_2017,cmap='crest')
```

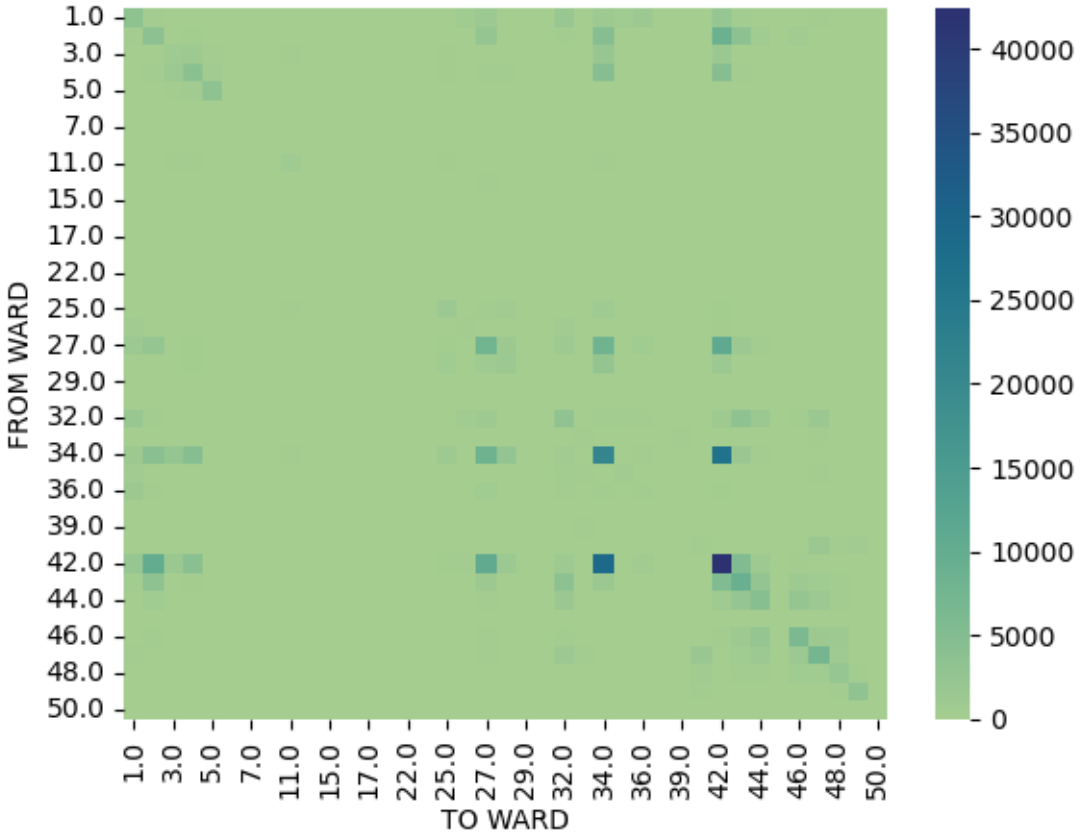
[51]: <AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>



It shows that most of the 31-40 people trips in 2017 happened in ward 42. After that there are 43, 34 and also among these wards.

[52]:
```
# Heatmap 41-50 in 2017
sns.heatmap(matrix_age_41_50_2017,cmap='crest')
```

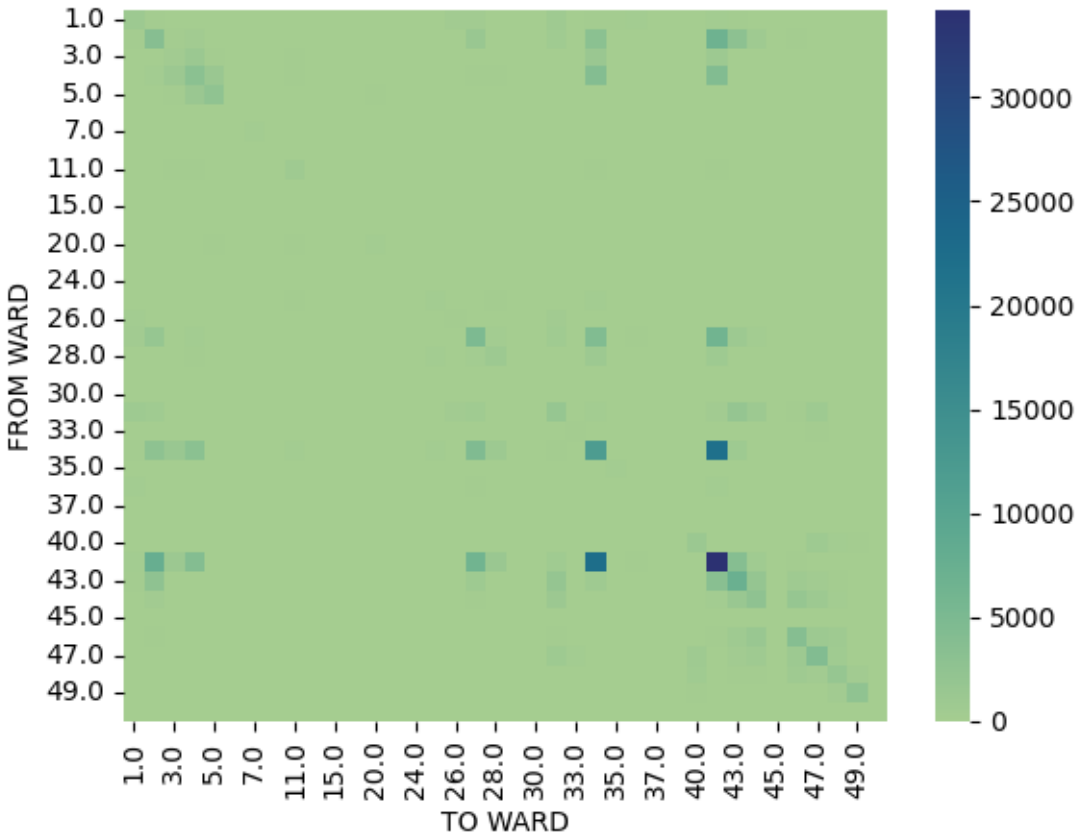
[52]: <AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>



It shows that most of 42-50 people's trips in 2017 happened in wards 42 and 34 and among these two wards.

[53]:
```
# Heatmap 51-60 in 2017
sns.heatmap(matrix_age_51_60_2017,cmap='crest')
```

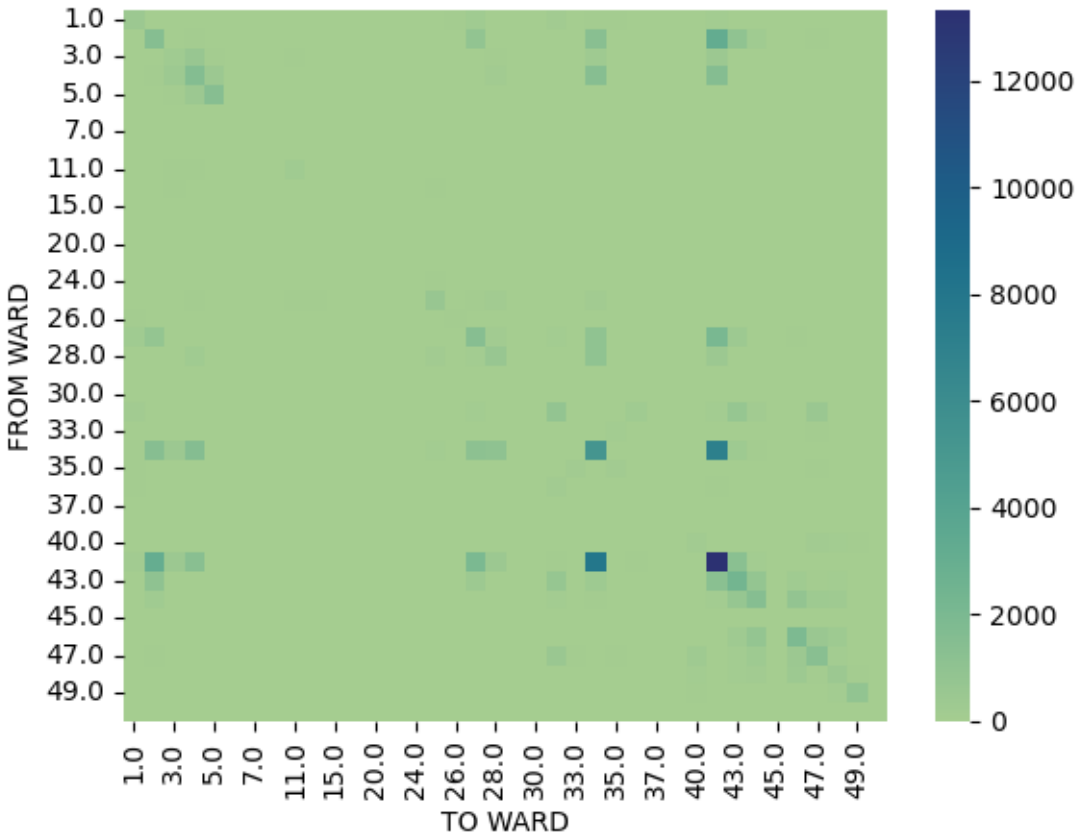
[53]: <AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>



The same pattern exists for 51-60 people.

[54]:
```
# Heatmap above 61 in 2017
sns.heatmap(matrix_age_above_61_2017,cmap='crest')
```

[54]: <AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>

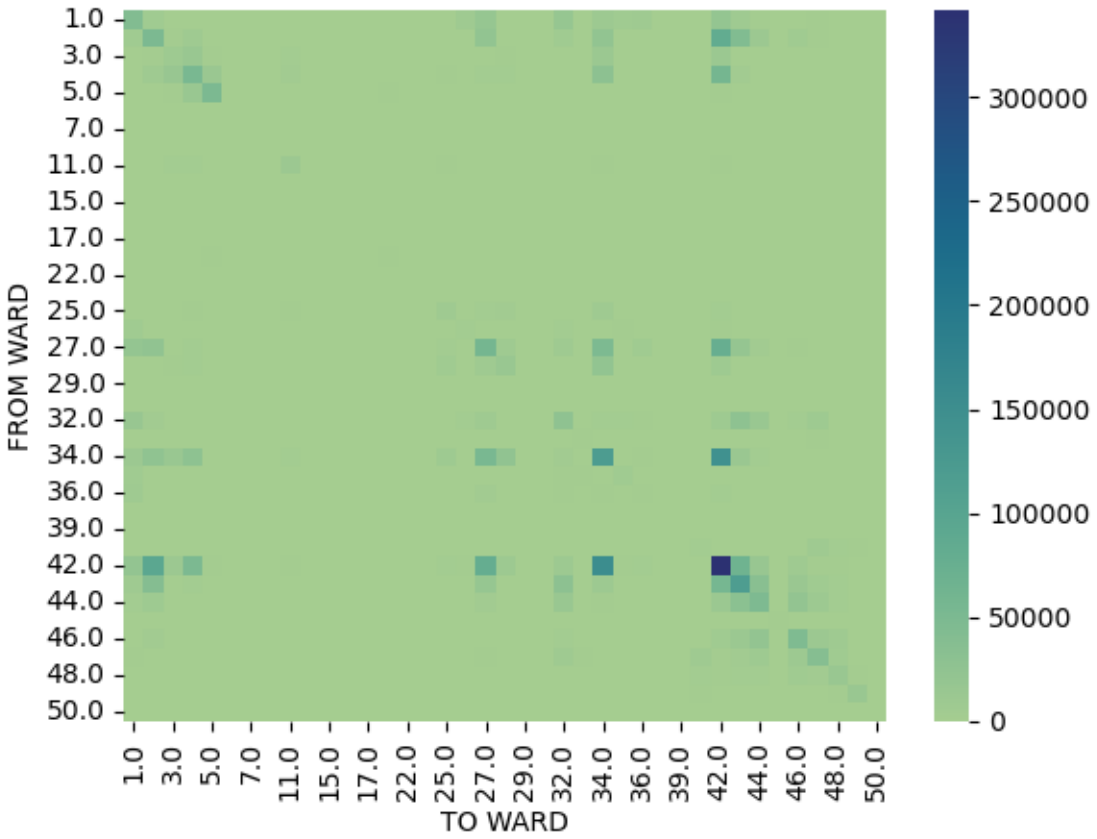


The same pattern exists for old people (above 61).

……

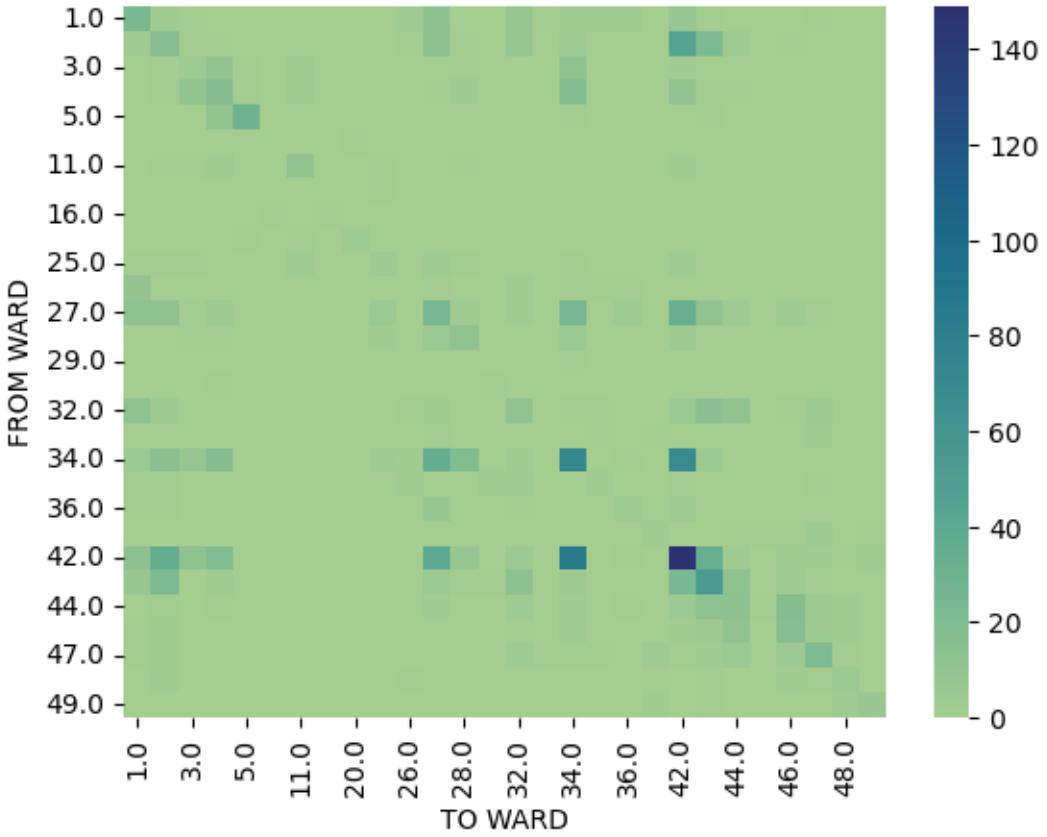
[37]:
```
# Heatmap general 2018
sns.heatmap(matrix_2018,cmap='crest')
```

[37]:
```
<AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>
```



[55]:
```
# Heatmap 0-18 in 2018
sns.heatmap(matrix_age_under_18_2018,cmap='crest')
```

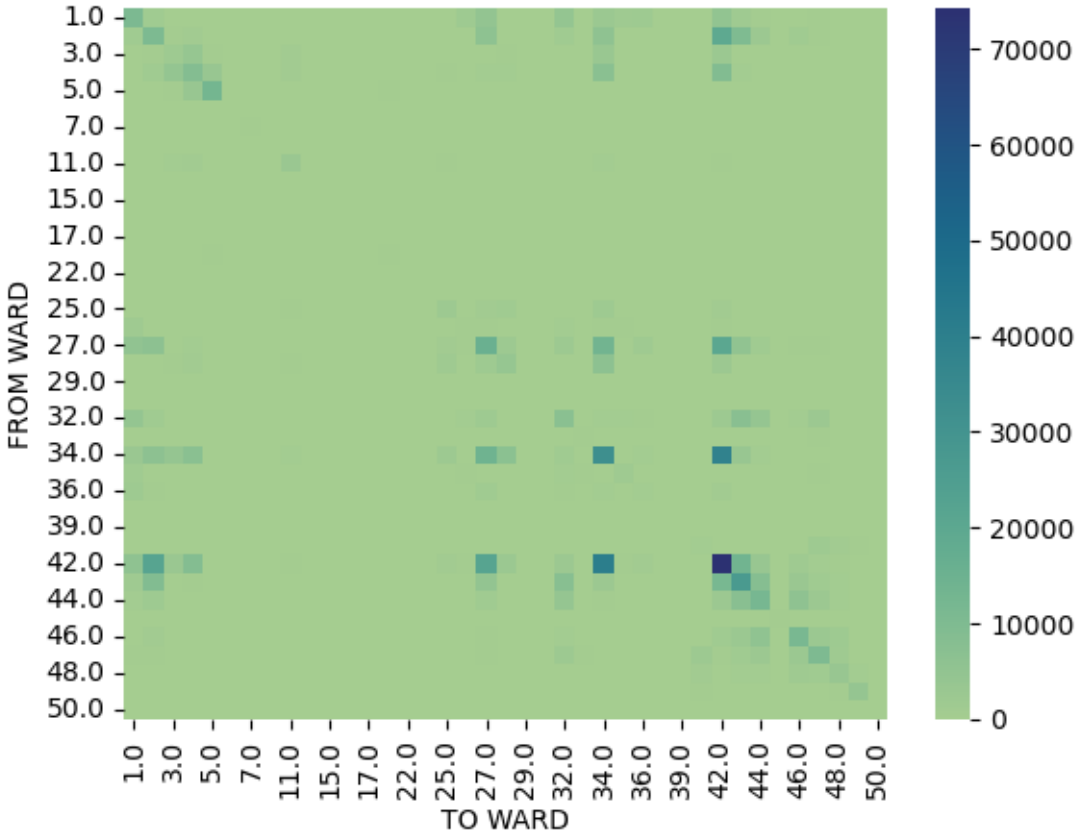
[55]: <AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>



It shows that in 2018, the most of the teenagers' trips happened inside ward 42 and 34 and also between these two wards.

[56]:
```
# Heatmap 19-30 in 2018
sns.heatmap(matrix_age_19_30_2018,cmap='crest')
```

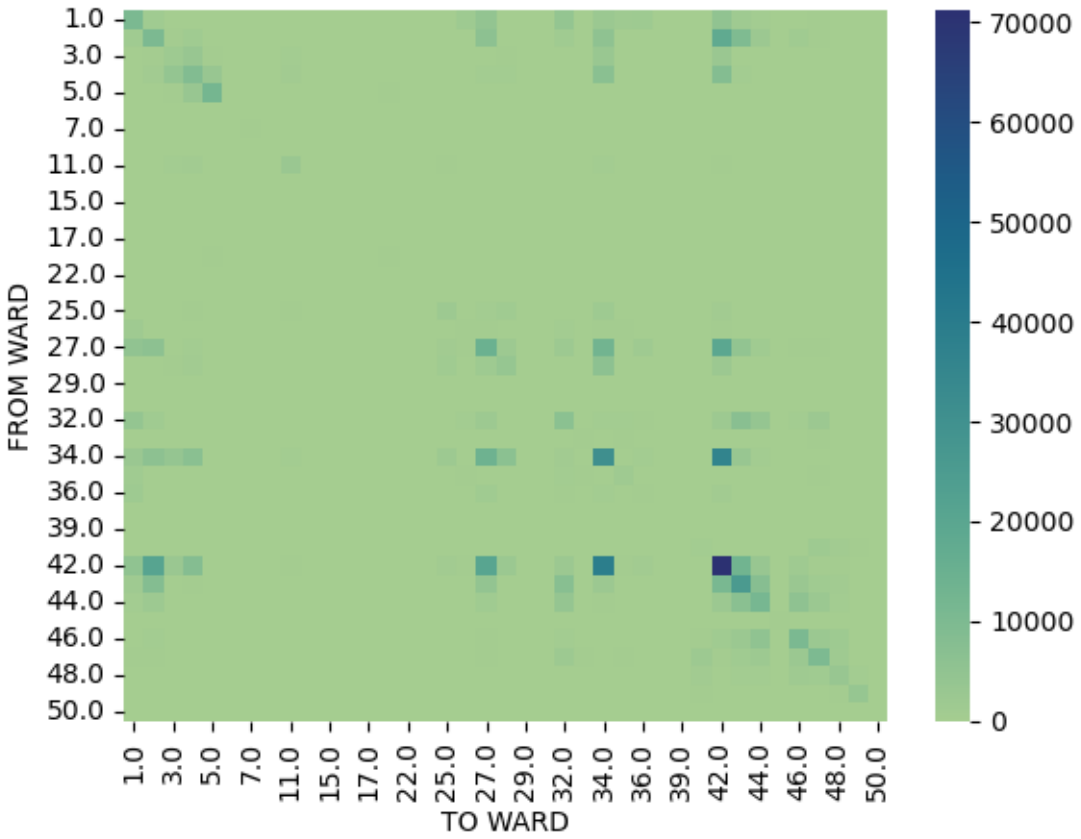
[56]: <AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>



It shows that the most of the young people's trips (19-30) in 2018 happened inside ward 5. The rest of places respectively belong to ward 42, 34, and 44.

[57]:
```
# Heatmap 31-40 in 2018
sns.heatmap(matrix_age_31_40_2018,cmap='crest')
```

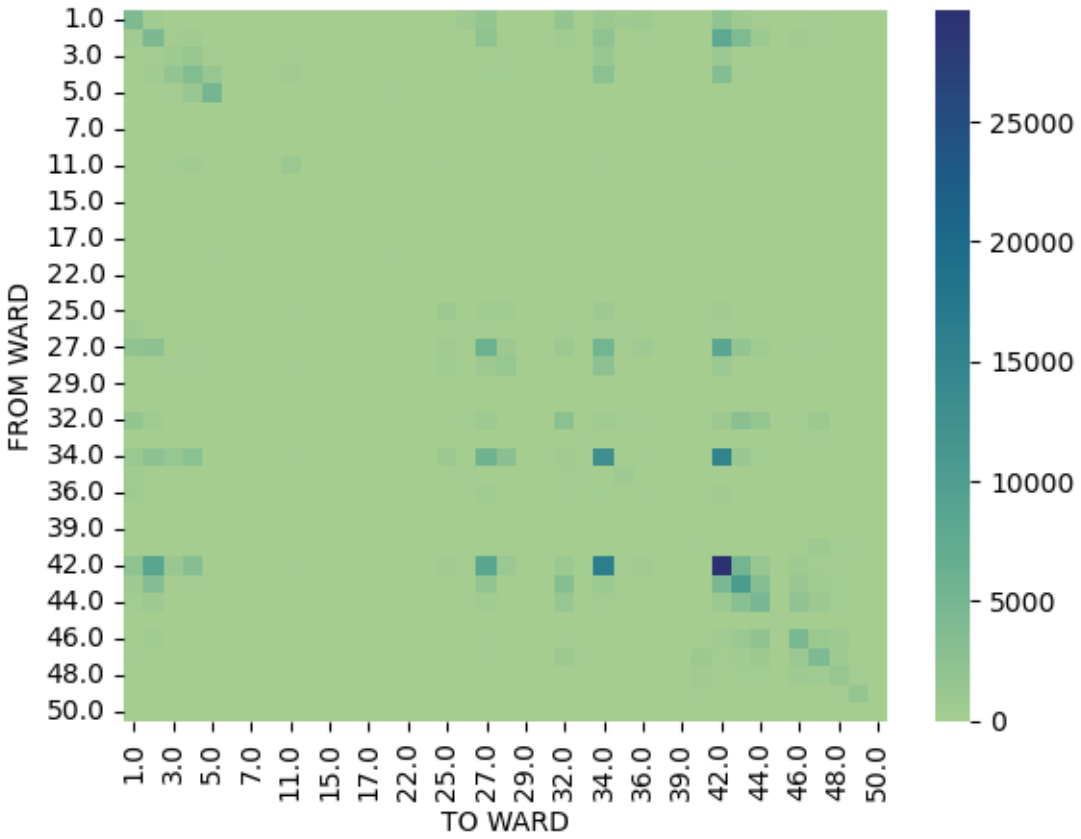
[57]: <AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>



It shows that most of 31-40 travelers' trips in 2018 happened in ward 42. The second place belongs to ward 44. Thus, we can see that ward 5 is not very attractive for this group however it was attractive for young people.

[58]:
```
# Heatmap 41-50 in 2018
sns.heatmap(matrix_age_41_50_2018,cmap='crest')
```

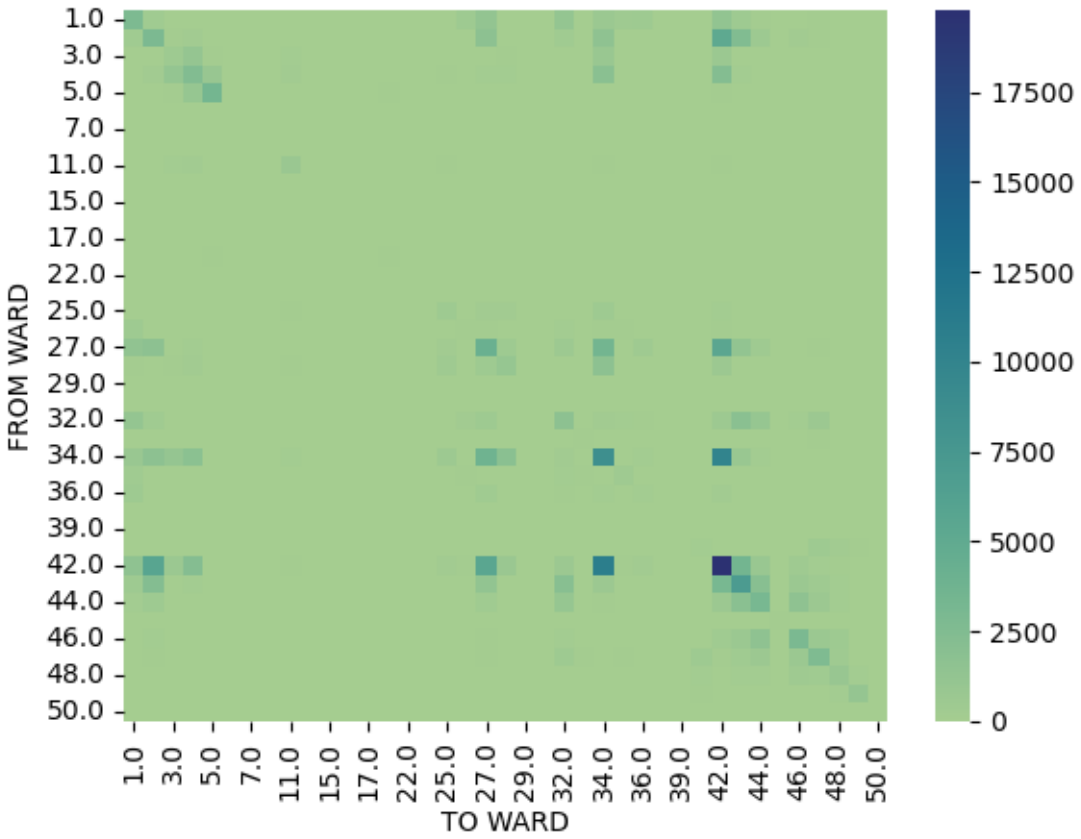
[58]: <AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>



It shows that 41-50 people's trips in 2018 happened in ward 42 and after that inside the ward 34. Additionally, the trips among these two wards (both ways) are also high.

[59]:
```
# Heatmap 51-60 in 2018
sns.heatmap(matrix_age_51_60_2018,cmap='crest')
```

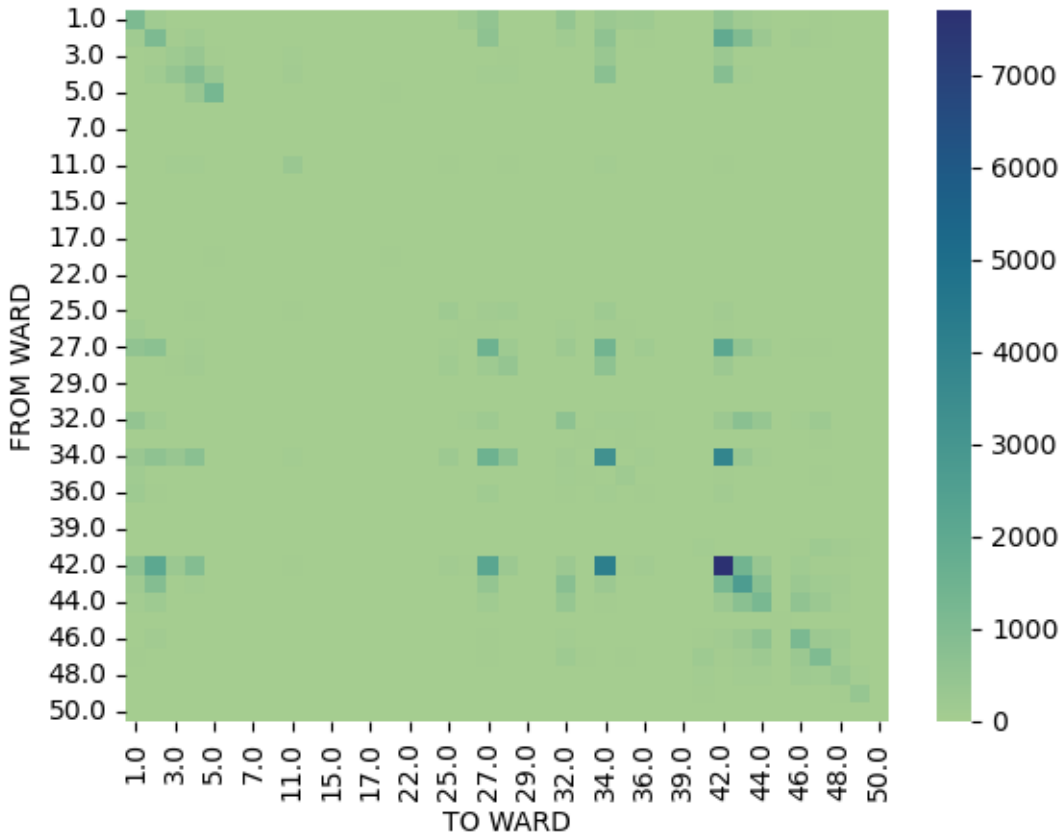
[59]: <AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>



It shows that the same pattern exists for people between 51-60.

[60]:
```
# Heatmap above 61 in 2018
sns.heatmap(matrix_age_above_61_2018,cmap='crest')
```

[60]: <AxesSubplot:xlabel='TO WARD', ylabel='FROM WARD'>



It shows that, traveler above 61 years old, mostly travel inside ward 42 and after that from ward 42 to ward 34.

In conclusion, It can be said that ward 42 is one of the most important zones in micromobility and ward 34 comes after that. Also, most of the young people's mobility happened in ward 5 but this ward is not very bold for other age groups.

……

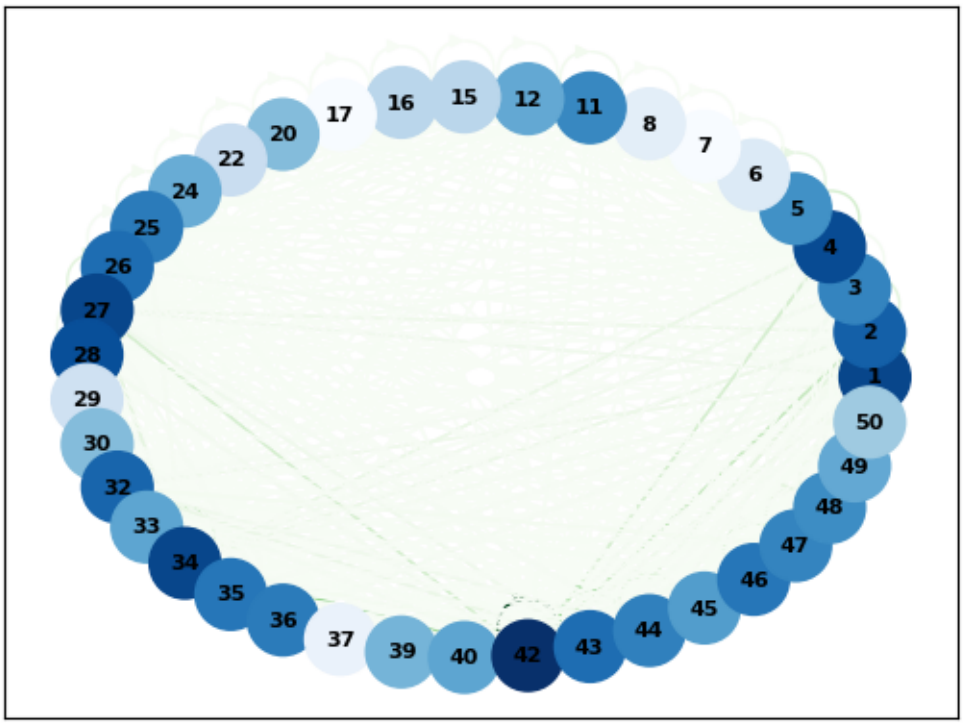
- Create a flowmap for the OD matrices.

[43]:
```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
import seaborn as sns

matrix_2016.index = matrix_2016.index.astype(int)
matrix_2016.columns = matrix_2016.columns.astype(int)

# Create a directed graph from the OD matrix
G = nx.DiGraph()
```

```
# Add nodes
for node in matrix_2016.index:
    G.add_node(node)

# Add edges
for from_node in matrix_2016.index:
    for to_node in matrix_2016.columns:
        weight = matrix_2016.loc[from_node, to_node]
        if weight > 0:
            G.add_edge(from_node, to_node, weight=weight)

# Draw the graph with a circular layout
pos = nx.circular_layout(G)

# Set node colors based on degree (number of connections)
node_colors = [G.degree(node) for node in G.nodes]

# Set edge colors based on weights
edge_colors = [G[from_node][to_node]['weight'] for from_node, to_node in G.
 ↪edges]

# Draw nodes
nx.draw_networkx_nodes(G, pos, node_size=700, node_color=node_colors, cmap=plt.
 ↪cm.Blues)

# Draw edges with weights
nx.draw_networkx_edges(G, pos, width=1, edge_color=edge_colors, edge_cmap=plt.
 ↪cm.Greens, arrowsize=10)

# Add labels
nx.draw_networkx_labels(G, pos, font_size=8, font_color='black',␣
 ↪font_weight='bold')

# Add colorbar for edge weights
edge_weights = nx.get_edge_attributes(G, 'weight')
cbar = plt.colorbar()
cbar.set_label('Flow Counts')

plt.title("Flowmap for OD Matrix 2017")
plt.show()
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
/tmp/ipykernel_5137/1446145611.py in <module>
     45 # Add colorbar for edge weights
     46 edge_weights = nx.get_edge_attributes(G, 'weight')
---> 47 cbar = plt.colorbar()
```

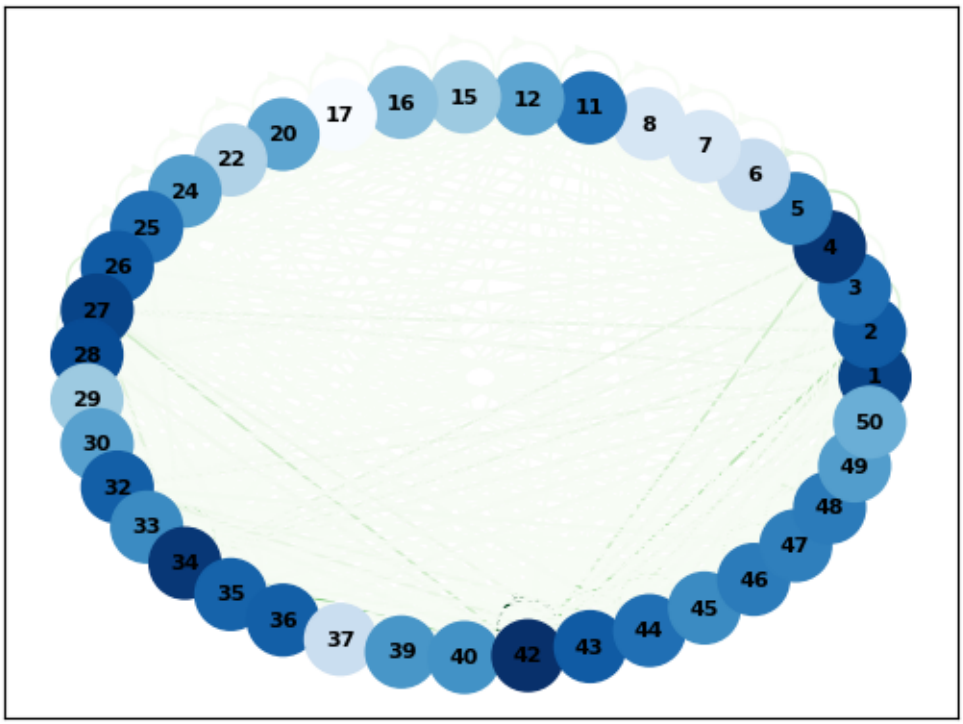
```
     48 cbar.set_label('Flow Counts')
     49

/opt/conda/lib/python3.7/site-packages/matplotlib/pyplot.py in␣
 ↪colorbar(mappable, cax, ax, **kw)
   2101         mappable = gci()
   2102         if mappable is None:
-> 2103             raise RuntimeError('No mappable was found to use for␣
 ↪colorbar '
   2104                                'creation. First define a mappable such␣
 ↪as '
   2105                                'an image (with imshow) or a contour set␣
 ↪('

RuntimeError: No mappable was found to use for colorbar creation. First define ␣
 ↪mappable such as an image (with imshow) or a contour set (with contourf).
```



[44]:
```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
import seaborn as sns

matrix_2017.index = matrix_2017.index.astype(int)
```

```
matrix_2017.columns = matrix_2017.columns.astype(int)

# Create a directed graph from the OD matrix
G = nx.DiGraph()

# Add nodes
for node in matrix_2017.index:
    G.add_node(node)

# Add edges
for from_node in matrix_2017.index:
    for to_node in matrix_2017.columns:
        weight = matrix_2017.loc[from_node, to_node]
        if weight > 0:
            G.add_edge(from_node, to_node, weight=weight)

# Draw the graph with a circular layout
pos = nx.circular_layout(G)

# Set node colors based on degree (number of connections)
node_colors = [G.degree(node) for node in G.nodes]

# Set edge colors based on weights
edge_colors = [G[from_node][to_node]['weight'] for from_node, to_node in G.
 ↪edges]

# Draw nodes
nx.draw_networkx_nodes(G, pos, node_size=700, node_color=node_colors, cmap=plt.
 ↪cm.Blues)

# Draw edges with weights
nx.draw_networkx_edges(G, pos, width=1, edge_color=edge_colors, edge_cmap=plt.
 ↪cm.Greens, arrowsize=10)

# Add labels
nx.draw_networkx_labels(G, pos, font_size=8, font_color='black',␣
 ↪font_weight='bold')

# Add colorbar for edge weights
edge_weights = nx.get_edge_attributes(G, 'weight')
cbar = plt.colorbar()
cbar.set_label('Flow Counts')

plt.title("Flowmap for OD Matrix 2017")
plt.show()
```

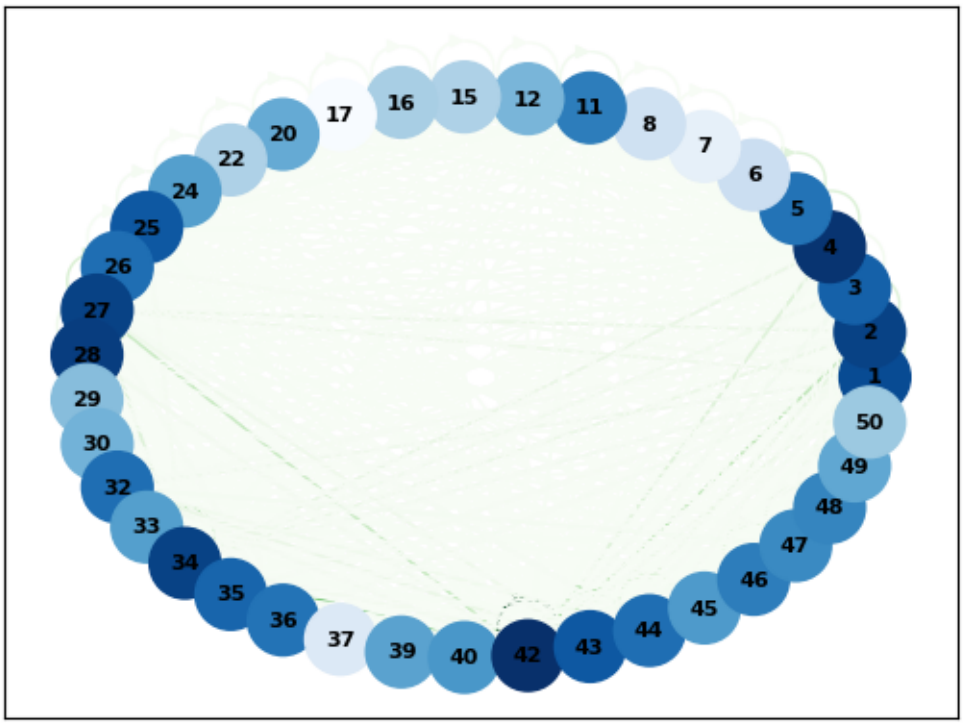
```
RuntimeError                              Traceback (most recent call last)
/tmp/ipykernel_5137/758526037.py in <module>
     41 # Add colorbar for edge weights
     42 edge_weights = nx.get_edge_attributes(G, 'weight')
---> 43 cbar = plt.colorbar()
     44 cbar.set_label('Flow Counts')
     45

/opt/conda/lib/python3.7/site-packages/matplotlib/pyplot.py in␣
 ↪colorbar(mappable, cax, ax, **kw)
   2101         mappable = gci()
   2102         if mappable is None:
-> 2103             raise RuntimeError('No mappable was found to use for␣
 ↪colorbar '
   2104                                'creation. First define a mappable such␣
 ↪as '
   2105                                'an image (with imshow) or a contour set␣
 ↪('

RuntimeError: No mappable was found to use for colorbar creation. First define ␣
 ↪mappable such as an image (with imshow) or a contour set (with contourf).
```



[46]:
```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
import seaborn as sns

matrix_2018.index = matrix_2018.index.astype(int)
matrix_2018.columns = matrix_2018.columns.astype(int)

# Create a directed graph from the OD matrix
G = nx.DiGraph()

# Add nodes
for node in matrix_2018.index:
    G.add_node(node)

# Add edges
for from_node in matrix_2018.index:
    for to_node in matrix_2018.columns:
        weight = matrix_2018.loc[from_node, to_node]
        if weight > 0:
            G.add_edge(from_node, to_node, weight=weight)

# Draw the graph with a circular layout
pos = nx.circular_layout(G)

# Set node colors based on degree (number of connections)
node_colors = [G.degree(node) for node in G.nodes]

# Set edge colors based on weights
edge_colors = [G[from_node][to_node]['weight'] for from_node, to_node in G.
 ↪edges]

# Draw nodes
nx.draw_networkx_nodes(G, pos, node_size=700, node_color=node_colors, cmap=plt.
 ↪cm.Blues)

# Draw edges with weights
nx.draw_networkx_edges(G, pos, width=1, edge_color=edge_colors, edge_cmap=plt.
 ↪cm.Greens, arrowsize=10)

# Add labels
nx.draw_networkx_labels(G, pos, font_size=8, font_color='black',␣
 ↪font_weight='bold')

# Add colorbar for edge weights
edge_weights = nx.get_edge_attributes(G, 'weight')
cbar = plt.colorbar()
```

```
cbar.set_label('Flow Counts')

plt.title("Flowmap for OD Matrix 2017")
plt.show()
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
/tmp/ipykernel_5137/3227053325.py in <module>
     41 # Add colorbar for edge weights
     42 edge_weights = nx.get_edge_attributes(G, 'weight')
---> 43 cbar = plt.colorbar()
     44 cbar.set_label('Flow Counts')
     45

/opt/conda/lib/python3.7/site-packages/matplotlib/pyplot.py in␣
 ↪colorbar(mappable, cax, ax, **kw)
   2101         mappable = gci()
   2102         if mappable is None:
-> 2103             raise RuntimeError('No mappable was found to use for␣
 ↪colorbar '
   2104                                'creation. First define a mappable such␣
 ↪as '
   2105                                'an image (with imshow) or a contour set␣
 ↪('

RuntimeError: No mappable was found to use for colorbar creation. First define ␣
 ↪mappable such as an image (with imshow) or a contour set (with contourf).
```



4 3 - Relation to Public transport line

In this sectoin, I am going to do again some pre-processing on micro-mobility data (Divvy) and extract the csv files with coordinations. Also, I have downloaded the kml files of public transport in Chicago (such as bus stops, bus routes, subway rails, etc.). Thus, I will import the output of this section and those kml files into QGIS software to create some maps for further analysis.

[ ]:
```
# Group data by wards in order to be used into QGIS for further analysis.
```

[43]:
```
import pandas as pd
```

[44]:
```
df_MM_clean_2016 = pd.read_csv('df_MM_clean_2016', on_bad_lines = 'warn',␣
↪sep=',')
df_MM_clean_2017 = pd.read_csv('df_MM_clean_2017', on_bad_lines = 'warn',␣
↪sep=',')
df_MM_clean_2018 = pd.read_csv('df_MM_clean_2018', on_bad_lines = 'warn',␣
↪sep=',')
```

[14]:
```
number_of_start_trips = df_MM_clean_2016.groupby("FROM WARD").size().
↪reset_index(name='number_of_start_trips_2016')
```

[15]:
```
number_of_start_trips.to_csv("number_of_start_trips_2016", index=False)
```

[16]:
```
number_of_start_trips_2017 = df_MM_clean_2017.groupby("FROM WARD").size().
  ↪reset_index(name='number_of_start_trips_2017')
```

[17]:
```
number_of_start_trips_2017.to_csv("number_of_start_trips_2017", index=False)
```

[18]:
```
number_of_start_trips_2018 = df_MM_clean_2018.groupby("FROM WARD").size().
  ↪reset_index(name='number_of_start_trips_2018')
```

[19]:
```
number_of_start_trips_2018.to_csv("number_of_start_trips_2018", index=False)
```

[20]:
```
number_of_end_trips_2016 = df_MM_clean_2016.groupby("TO WARD").size().
  ↪reset_index(name='number_of_end_trips_2016')
```

[21]:
```
number_of_end_trips_2016.to_csv("number_of_end_trips_2016", index=False)
```

[22]:
```
number_of_end_trips_2017 = df_MM_clean_2017.groupby("TO WARD").size().
  ↪reset_index(name='number_of_end_trips_2017')
```

[23]:
```
number_of_end_trips_2017.to_csv("number_of_end_trips_2017", index=False)
```

[24]:
```
number_of_end_trips_2018 = df_MM_clean_2018.groupby("TO WARD").size().
  ↪reset_index(name='number_of_end_trips_2018')
```

[25]:
```
number_of_end_trips_2018.to_csv("number_of_end_trips_2018", index=False)
```

[82]:
```
df_MM_clean_2019 = pd.read_csv('df_2019.csv', on_bad_lines = 'warn', sep=',')
```

[27]:
```
number_of_start_trips_2019 = df_MM_clean_2019.groupby("FROM WARD").size().
  ↪reset_index(name='number_of_start_trips_2019')
```

[28]:
```
number_of_start_trips_2019.to_csv("number_of_start_trips_2019", index=False)
```

[29]:
```
number_of_end_trips_2019 = df_MM_clean_2019.groupby("TO WARD").size().
  ↪reset_index(name='number_of_start_trips_2019')
```

[30]:
```
number_of_end_trips_2019.to_csv("number_of_end_trips_2019", index=False)
```

Here, I am dividing a day into 3 day times (Day, Evening and Night) to do some analysis based on the time of trips in QGIS software.

[62]:
```
# day time
df_MM_clean_2016_time = pd.to_datetime(df_MM_clean_2016['START TIME'],␣
  ↪format="%m/%d/%Y %I:%M:%S %p")
df_MM_clean_2017_time = pd.to_datetime(df_MM_clean_2017['START TIME'],␣
  ↪format="%m/%d/%Y %I:%M:%S %p")
```

```
df_MM_clean_2018_time = pd.to_datetime(df_MM_clean_2018['START TIME'],␣
↪format="%m/%d/%Y %I:%M:%S %p")
```

[63]:
```
import datetime as dt
df_MM_clean_2016["HOUR"] = df_MM_clean_2016_time.dt.strftime("%H")
df_MM_clean_2017["HOUR"] = df_MM_clean_2017_time.dt.strftime("%H")
df_MM_clean_2018["HOUR"] = df_MM_clean_2018_time.dt.strftime("%H")
```

[64]:
```
# Function for day time
def day_time(hour):
    if  hour=='00' or hour=='01' or hour=='02' or hour=='03' or hour=='04' or␣
↪hour=='05' or hour=='06' or hour=='07':
        return 'Night'
    elif hour=='08' or hour=='09' or hour=='10' or hour=='11' or hour=='12' or␣
↪hour=='13' or hour=='14' or hour=='15':
        return 'Day'
    elif hour=='16' or hour=='17' or hour=='18' or hour=='19' or hour=='20' or␣
↪hour=='21' or hour=='22' or hour=='23':
        return 'Evening'
```

[65]:
```
df_MM_clean_2016["DAY_TIME"] = df_MM_clean_2016["HOUR"].apply(day_time)
df_MM_clean_2017["DAY_TIME"] = df_MM_clean_2017["HOUR"].apply(day_time)
df_MM_clean_2018["DAY_TIME"] = df_MM_clean_2018["HOUR"].apply(day_time)
```

[67]:
```
df_MM_clean_2016.head()
```

[67]:

	TRIP ID	START TIME	STOP TIME	BIKE ID	\
0	10000000	06/09/2016 06:24:00 PM	06/09/2016 06:33:00 PM	5	
1	10000001	06/09/2016 06:24:00 PM	06/09/2016 06:27:00 PM	4201	
2	10000002	06/09/2016 06:25:00 PM	06/09/2016 06:41:00 PM	1976	
3	10000003	06/09/2016 06:25:00 PM	06/09/2016 06:33:00 PM	5354	
4	10000005	06/09/2016 06:25:00 PM	06/09/2016 06:38:00 PM	4365	

	TRIP DURATION	FROM STATION ID	FROM STATION NAME	\
0	516	26	McClurg Ct & Illinois St	
1	163	238	Ravenswood Ave & Montrose Ave (*)	
2	987	91	Clinton St & Washington Blvd	
3	518	310	Damen Ave & Charleston St	
4	813	20	Sheffield Ave & Kingsbury St	

	TO STATION ID	TO STATION NAME	USER TYPE	…	SEASON	\
0	90	Millennium Park	Subscriber	…	Summer	
1	234	Clark St & Montrose Ave	Subscriber	…	Summer	
2	289	Wells St & Concord Ln	Subscriber	…	Summer	
3	327	Sheffield Ave & Webster Ave	Subscriber	…	Summer	
4	153	Southport Ave & Wellington Ave	Subscriber	…	Summer	

```
   DAY OF WEEK                  from_point                      to_point  \
0     Thursday    POINT (-87.6173 41.89102)  POINT (-87.624084 41.881032)
1     Thursday  POINT (-87.674365 41.961615)  POINT (-87.666036 41.961588)
2     Thursday   POINT (-87.64117 41.88338)  POINT (-87.634656 41.912133)
3     Thursday  POINT (-87.677855 41.920082)   POINT (-87.653818 41.92154)
4     Thursday  POINT (-87.653106 41.910522)  POINT (-87.663576 41.935733)

  FROM WARD  TO WARD  age age group  HOUR  DAY_TIME
0      42.0     42.0   28     19-30    18   Evening
1      47.0     47.0   32     31-40    18   Evening
2      42.0      2.0   27     19-30    18   Evening
3      32.0     43.0   29     19-30    18   Evening
4      27.0     32.0   35     31-40    18   Evening

[5 rows x 30 columns]
```

[69]:
```
group_day_time_2016 = df_MM_clean_2016.groupby(["FROM WARD", "DAY_TIME"]).
↪size().reset_index(name='group_day_time_2016')
```

[71]:
```
group_day_time_2017 = df_MM_clean_2017.groupby(["FROM WARD", "DAY_TIME"]).
↪size().reset_index(name='group_day_time_2017')
```

[72]:
```
group_day_time_2018 = df_MM_clean_2018.groupby(["FROM WARD", "DAY_TIME"]).
↪size().reset_index(name='group_day_time_2018')
```

[73]:
```
group_day_time_2016.to_csv("group_day_time_2016", index=False)
```

[74]:
```
group_day_time_2017.to_csv("group_day_time_2017", index=False)
```

[75]:
```
group_day_time_2018.to_csv("group_day_time_2018", index=False)
```

---

5 4 – Utilization of bicycles and costs

In this section, I am going to calculate the utilization percentage of bicycles and also assess the cost and benefit of the bike-sharing company. For this purpose, I have made a rough estimation of bicycle maintenance costs and the revenue from each bike based on the trip duration. This is just a simple simulation of this kind of analysis.

[76]:
```
# Working on the last year of data 2019
```

[1]:
```
import pandas as pd
df_MM_clean_2019 = pd.read_csv('dataframe_2019 (1).csv', on_bad_lines = 'warn',␣
↪sep=',')
```

[2]:
```
# Create datetime from "START TIME", "STOP TIME"
df_MM_clean_2019['START TIME'] = pd.to_datetime(df_MM_clean_2019['START TIME'])
df_MM_clean_2019['STOP TIME'] = pd.to_datetime(df_MM_clean_2019['STOP TIME'])
```

[3]:
```
# Geoup by "BIKE ID"
group_bikes = df_MM_clean_2019.groupby("BIKE ID").agg({"TRIP DURATION": "sum",
                                                       "START TIME": "min",
                                                       "STOP TIME": "max"})
```

[4]:
```
# Convert from seconds to minutes
group_bikes["TRIP DURATION (MINUTES)"] = group_bikes["TRIP DURATION"]/60
```

[6]:
```
# Classic bike price of divvy = 0.17$/min
```

[5]:
```
# Calculate the revenue
group_bikes["REVENUE (USD)"] = group_bikes["TRIP DURATION (MINUTES)"] * 0.17
```

[6]:
```
# Calculate the total time that each bicycle exists on the streets
group_bikes["ACTIVE TIME"] = (group_bikes["STOP TIME"] - group_bikes["START␣
 ↪TIME"]).dt.total_seconds() / 60
```

[7]:
```
# Calculate the utilization percentage of each bike
group_bikes["utilization_percentage"] = ((group_bikes["TRIP DURATION␣
 ↪(MINUTES)"] / group_bikes["ACTIVE TIME"]) * 100)
```

[10]:
```
group_bikes.head()
```

[10]:

BIKE ID	TRIP DURATION	START TIME	STOP TIME	TRIP DURATION (MINUTES)	REVENUE (USD)	ACTIVE TIME	utilization_percentage
1	674786	2019-01-14 07:54:08	2019-12-28 22:32:06	11246.433333	1911.893667	501997.966667	2.240334
2	6618706	2019-02-25 00:59:06	2019-12-31 07:36:02	110311.766667	18753.000333	445356.933333	24.769294
3	834957	2019-02-22 07:04:14	2019-11-09 21:42:37	13915.950000	2365.711500	375278.383333	
4	939373	2019-04-11 17:25:27	2019-12-29 16:14:28	15656.216667	2661.556833	377209.016667	
5	391873	2019-01-08 13:13:01	2019-12-30 14:47:56	6531.216667	1110.306833	512734.916667	

```
3                      3.708167
4                      4.150541
5                      1.273800
```

[11]:
```
group_bikes["utilization_percentage"].mean()
```

[11]:
```
3.843668012819399
```

The average of utilization percentage of bikes is about 3.8%.

[12]:
```
group_bikes["utilization_percentage"].median()
```

[12]:
```
3.055378640027452
```

[8]:
```
group_bikes["utilization_percentage"].max()
```

[8]:
```
100.0
```

[9]:
```
group_bikes["utilization_percentage"].min()
```

[9]:
```
0.0031177561471445667
```

[22]:
```
group_bikes.sort_values("REVENUE (USD)", ascending = False).head()
```

[22]:
```
          TRIP DURATION          START TIME           STOP TIME  \
BIKE ID
3846           11124484 2019-01-14 08:01:04 2019-10-09 14:29:43
1229            9550031 2019-01-02 09:57:50 2019-12-29 19:20:44
6232            9287412 2019-03-06 11:50:03 2020-01-13 21:55:24
1083            9098417 2019-01-01 12:47:38 2020-01-21 13:54:35
773             8538863 2019-01-05 18:27:43 2019-10-25 10:56:54

          TRIP DURATION (MINUTES)  REVENUE (USD)    ACTIVE TIME  \
BIKE ID
3846                185408.066667   31519.371333  386308.650000
1229                159167.183333   27058.421167  520402.900000
6232                154790.200000   26314.334000  451325.350000
1083                151640.283333   25778.848167  554466.950000
773                 142314.383333   24193.445167  421469.183333

          utilization_percentage
BIKE ID
3846                   47.994801
1229                   30.585376
6232                   34.296810
1083                   27.348841
773                    33.766261
```

We can see that Bike 3846 with 48% utilization percentage, made about 31,519 dolar with 185,408 minutes trip.

[21]:
```
total_revenue = group_bikes["REVENUE (USD)"].sum()
```

[22]:
```
print(f'the total revenue is {total_revenue}')
```

```
the total revenue is 15687834.081500001
```

The revenue that we hav calculated, shows that in 2019, divvy made at least 15.68 M$.

[15]:
```
# Costs for classic bikes per ye
bikes_price = 300
maintenance = 50
infrastructure = 800
technology = 30
insurance = 10
operations = 60
marketing = 20
unexpected_costs = 20
```

[16]:
```
# Calculate the total cost for divvy per each bike
cost = bikes_price + maintenance + infrastructure + technology + insurance +␣
 ↪operations + marketing + unexpected_costs
```

[23]:
```
print(f'the cost for each bike in 2019 is {cost}')
```

```
the cost for each bike in 2019 is 1290
```

[18]:
```
group_bikes.count()
```

[18]:
```
TRIP DURATION              6017
START TIME                 6017
STOP TIME                  6017
TRIP DURATION (MINUTES)    6017
REVENUE (USD)              6017
ACTIVE TIME                6017
utilization_percentage     6017
dtype: int64
```

[19]:
```
total_cost = 6017 * cost
```

[24]:
```
print(f'the total cost in 2019 is {total_cost}')
```

```
the total cost in 2019 is 7761930
```

[25]:
```
interest = total_revenue - total_cost
```

[26]:
```
print(f'the interest in 2019 is {interest}')
```

```
the interest in 2019 is 7925904.081500001
```

I have estimated that there was about 8 M$ benefit for Divvy in 2019.

……

[99]:
```
# During different day of week
```

[10]:
```
group_bikes_week = df_MM_clean_2019.groupby(["BIKE ID", "Day_of_Week"]).
↪agg({"TRIP DURATION": "sum",
                                                                    ␣
↪"START TIME": "min",
                                                                     "STOP␣
↪TIME": "max"})
```

[11]:
```
# Convert from seconds to minutes
group_bikes_week["TRIP DURATION (MINUTES)"] = group_bikes_week["TRIP DURATION"]/
↪60
```

[12]:
```
# Calculate the total time that each bicycle exists on the streets in each day␣
↪of week
group_bikes_week["ACTIVE TIME"] = (group_bikes_week["STOP TIME"] -␣
↪group_bikes_week["START TIME"]).dt.total_seconds() / 60
```

[13]:
```
# Calculate the utilization percentage of each bike in each day of week
group_bikes_week["utilization_percentage"] = ((group_bikes_week["TRIP DURATION␣
↪(MINUTES)"] / group_bikes_week["ACTIVE TIME"]) * 100)
```

[42]:
```
group_bikes_week.head(14)
```

[42]:

		TRIP DURATION	START TIME	STOP TIME	\
BIKE ID	Day_of_Week				
1	Friday	131254	2019-02-22 14:34:53	2019-12-27 06:02:59	
	Monday	89813	2019-01-14 07:54:08	2019-09-09 18:53:30	
	Saturday	131591	2019-03-16 17:03:42	2019-12-28 22:32:06	
	Sunday	83157	2019-03-10 19:59:49	2019-09-08 12:07:21	
	Thursday	64400	2019-01-17 16:40:10	2019-09-12 18:36:49	
	Tuesday	105962	2019-02-26 09:06:35	2019-10-15 23:47:58	
	Wednesday	68609	2019-02-27 17:55:27	2019-10-09 18:52:46	
2	Friday	43249	2019-03-15 11:36:32	2019-12-06 09:07:14	
	Monday	55960	2019-02-25 00:59:06	2019-12-30 18:30:04	
	Saturday	107622	2019-03-02 16:02:50	2019-12-07 12:11:52	
	Sunday	97991	2019-03-03 14:14:48	2019-12-29 13:54:59	
	Thursday	37374	2019-03-07 17:30:22	2019-12-12 11:27:22	
	Tuesday	296018	2019-03-05 04:51:18	2019-12-31 07:36:02	
	Wednesday	5980492	2019-02-27 21:10:52	2019-12-04 17:31:56	

	TRIP DURATION (MINUTES)	ACTIVE TIME	\

```
BIKE ID Day_of_Week
1       Friday                   2187.566667  443008.100000
        Monday                   1496.883333  343379.366667
        Saturday                 2193.183333  413608.400000
        Sunday                   1385.950000  261607.533333
        Thursday                 1073.333333  342836.650000
        Tuesday                  1766.033333  333521.383333
        Wednesday                1143.483333  322617.316667
2       Friday                    720.816667  382890.700000
        Monday                    932.666667  444570.966667
        Saturday                 1793.700000  402969.033333
        Sunday                   1633.183333  433420.183333
        Thursday                  622.900000  402837.000000
        Tuesday                  4933.633333  433604.733333
        Wednesday               99674.866667  402981.066667

                        utilization_percentage
BIKE ID Day_of_Week
1       Friday                        0.493798
        Monday                        0.435927
        Saturday                      0.530256
        Sunday                        0.529782
        Thursday                      0.313074
        Tuesday                       0.529511
        Wednesday                     0.354440
2       Friday                        0.188257
        Monday                        0.209790
        Saturday                      0.445121
        Sunday                        0.376813
        Thursday                      0.154628
        Tuesday                       1.137818
        Wednesday                    24.734380
```

[20]:
```
group_bikes_week.sort_values('utilization_percentage', ascending=False).head()
```

[20]:
```
                        TRIP DURATION          START TIME           STOP TIME  \
BIKE ID Day_of_Week
56      Wednesday                 439 2019-05-01 14:16:17 2019-05-01 14:23:36
3973    Monday                    578 2019-03-18 18:28:14 2019-03-18 18:37:52
6206    Monday                    150 2019-06-17 13:54:49 2019-06-17 13:57:19
2581    Monday                 212030 2019-07-15 09:03:03 2019-07-17 19:56:53
3973    Thursday                27338 2019-04-11 19:09:46 2019-04-12 02:45:24

                        TRIP DURATION (MINUTES)  ACTIVE TIME  \
BIKE ID Day_of_Week
56      Wednesday                      7.316667     7.316667
3973    Monday                         9.633333     9.633333
```

```
6206    Monday                    2.500000     2.500000
2581    Monday                 3533.833333  3533.833333
3973    Thursday                455.633333   455.633333
```

```
                     utilization_percentage
BIKE ID Day_of_Week
56      Wednesday                     100.0
3973    Monday                        100.0
6206    Monday                        100.0
2581    Monday                        100.0
3973    Thursday                      100.0
```

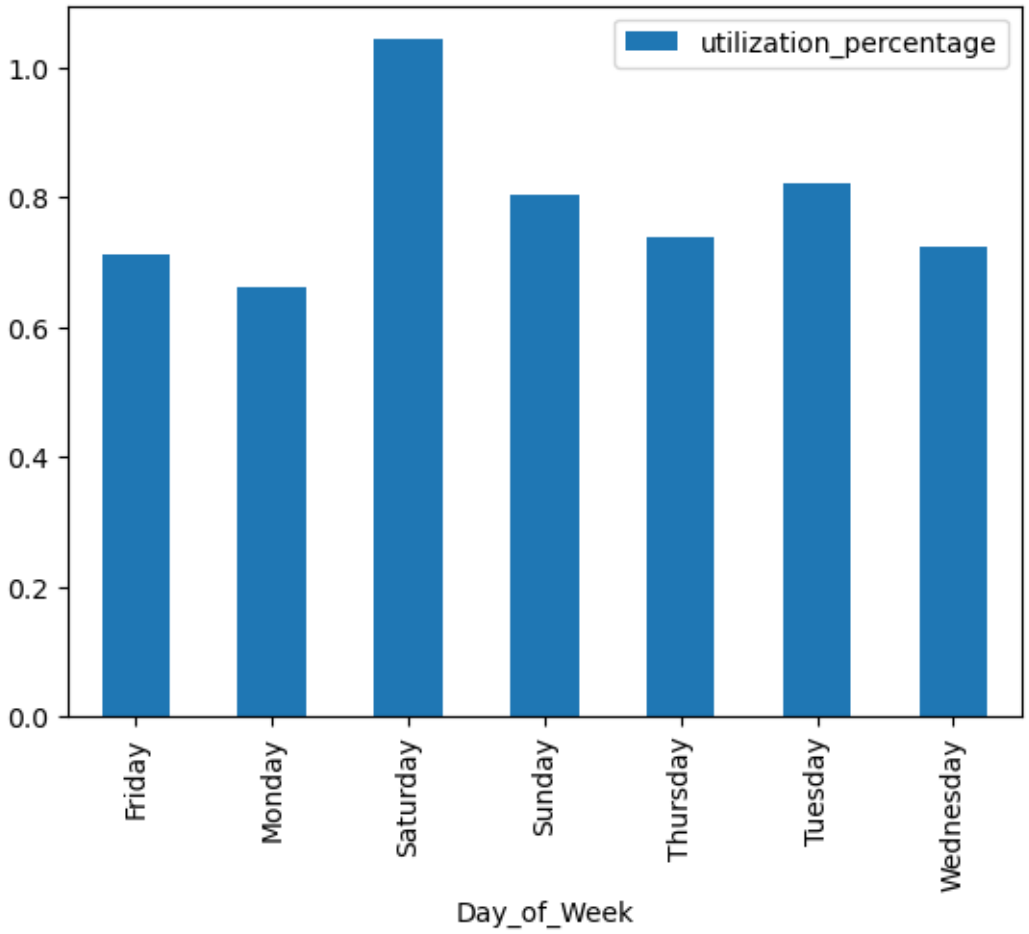
[15]:
```
week_days = group_bikes_week.groupby("Day_of_Week").
 ↪agg({"utilization_percentage": "mean"})
```

[36]:
```
week_days.head(7)
```

[36]:
```
             utilization_percentage
Day_of_Week
Friday                     0.713160
Monday                     0.662425
Saturday                   1.043443
Sunday                     0.803796
Thursday                   0.739848
Tuesday                    0.821461
Wednesday                  0.724842
```

It shows that the average of utilization of bikes was more on weekends.

[17]:
```
# The bar plot of average of utilization percentage within each day of week:
bp_week_days = week_days.plot(kind = "bar")
```



[37]:
```
week_days_median = group_bikes_week.groupby("Day_of_Week").
↪agg({"utilization_percentage": "median"})
```

[39]:
```
week_days_median.head(7)
```

[39]:

Day_of_Week	utilization_percentage
Friday	0.429317
Monday	0.408524
Saturday	0.553881
Sunday	0.462633
Thursday	0.408802
Tuesday	0.397998
Wednesday	0.397965

[40]:
```
group_bikes.to_csv("group_bikes_2019_2", index=False)
```

[41]:
```
group_bikes_week.to_csv("group_bikes_week_2019_2", index=False)
```

This is the end of this Analysis. More analysis can be done in further efforts.