taipei_passerby_prediction

October 6, 2020

1 Taipei Passerby Prediction

In order to know the demand / the market, we would try to predict passerby number as one of the factor (later combined with buying power). In determining the passerby number, we would do:

- Limit the scope of MRT data usage to early hour only, in order to show normal daily hour activities.
- Simulate walking distance simulation on MRT passenger.
- Predict the number of passerby for each village.

[1]:
```
# initial setup, import packages, path, and config
import json
import os

import numpy as np
import pandas as pd
import geopandas as gpd
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
from shapely.geometry import MultiPoint
pd.options.mode.chained_assignment = None  # not show dataframe copy slice␣
↪warning
pio.renderers.default = 'jupyterlab'

import dask
import dask.dataframe as dd
from datetime import datetime
from dask.distributed import Client, LocalCluster

from lib import shared_lib
from shared_lib import data_processor
from data_processor.lib.geocoding import GeoCoder
from data_processor.lib.geolib_helper import get_shp_filepath,␣
↪load_normalize_gov_shp_data

from lib.plotly_helper import add_chart_title, add_chart_annotation
```

```
# dask config
cluster = LocalCluster(
#     n_workers=os.cpu_count()  # this is if you want to setup number of dask␣
↪worker
)
client = Client(cluster)

# setup path
ANALYSIS_NAME = 'taipei_passerby_prediction'

CURRENT_DIR = os.path.dirname(os.path.abspath('__file__'))
BASE_DIR = os.path.dirname(CURRENT_DIR)
ANALYSIS_DIR = os.path.join(BASE_DIR, 'analysis', ANALYSIS_NAME)

plotly_default_config_chart = dict(
    displayModeBar=True,
    responsive=False,
    modeBarButtonsToRemove=['zoomIn2d', 'zoomOut2d', 'select2d', 'lasso2d',␣
↪'toggleSpikelines'],
    displaylogo=False
)

plotly_default_config_geo = dict(
    displayModeBar=True,
    responsive=False,
    scrollZoom=False,
    modeBarButtonsToRemove=['select2d', 'lasso2d'])
```

1.1 Use only specific hours to estimate passerby number

To get the passenger data, we would use the traffic which accounted for the people that would start their work only. So we would exclude the number of people that would use the MRT to end their day (i.e. go home, etc). Therefore we would determine the hour we want to take on.

We would check the hour based on people activities, some of our consideration include: - **Average peak hour for going to work**, we don't want to set the hour befire this time.

The step that we would do to make and support our hypothesis: 1. Get hourly average number of people that use MRT, grouped for weekdays and weekend. 2. Set the hour that represent MRT used for starting daily activities. Make the start day usage peak hour.

[2]:
```
# setup data source:
# - taipei mrt passenger data -> taipei_mrt_info
data_warehouse_dir = os.path.join(BASE_DIR, 'data', 'normalized-data_warehouse')
taipei_mrt_info_dirpath = os.path.join(data_warehouse_dir, 'taipei_mrt_info')
taipei_mrt_info_urlpath = os.path.join(taipei_mrt_info_dirpath,␣
↪'taipei_mrt_passenger_data_*.csv')
```

```python
source_df = dd.read_csv(
    taipei_mrt_info_urlpath,
    parse_dates=['date'],
    infer_datetime_format='%Y-%m-%d'
)
```

1.1.1 Get hourly average number

Aggregate hourly MRT passenger, split by weekdays and weekend.

[3]:
```python
# process the data
df = source_df

agg_df = df\
    .groupby(['date','time_period'])['person_times'].sum()\
    .reset_index()

agg_df['isoweekday_num'] = agg_df['date'].apply(lambda x: x.isoweekday(),␣
 ↪meta=int)
agg_df['mon_to_friday'] = agg_df['isoweekday_num'].apply(lambda x: 1 if x <= 5␣
 ↪else 0, meta=int)

agg_df = agg_df\
    .groupby(['mon_to_friday', 'time_period'])['person_times'].mean()\
    .reset_index()

taipei_mrt_hourly_weekdays_weekend_average = agg_df.compute()

# data sample
print(taipei_mrt_hourly_weekdays_weekend_average.head())
```

```
   mon_to_friday  time_period  person_times
0              0            0  14947.983539
1              0            1    348.930041
2              0            5     63.041152
3              0            6  17964.485597
4              0            7  39963.934156
```

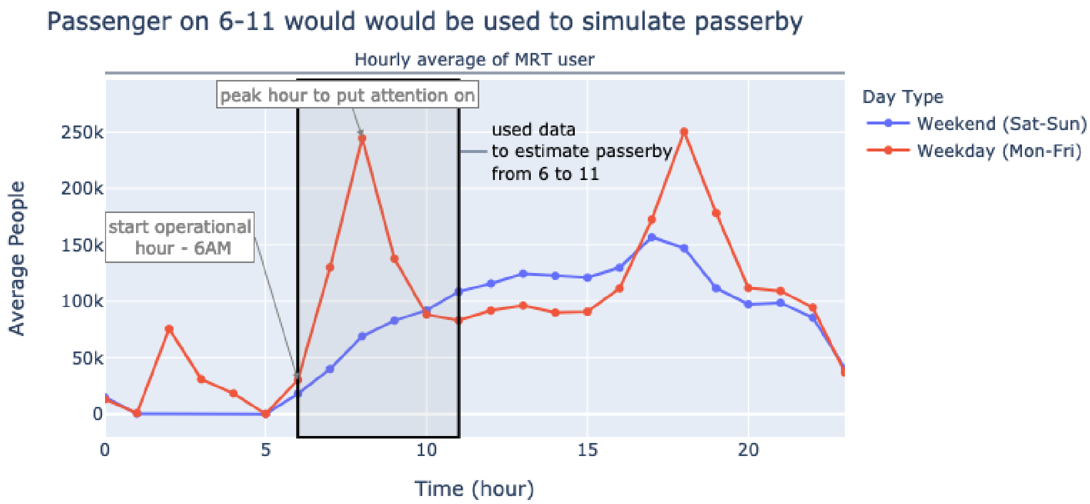
1.1.2 Check hour representation

Here is the data visualization. The data that we would use to represent

[4]:
```python
# plot the graph
fig = px.line(taipei_mrt_hourly_weekdays_weekend_average,
              x='time_period',
              y='person_times',
              color='mon_to_friday')
```

```
fig.update_traces(mode='markers+lines', hovertemplate='%{y}')

fig.for_each_trace(
    lambda data: data.update(
        name={
            '0':'Weekend (Sat-Sun)',
            '1':"Weekday (Mon-Fri)"
        }.get(data.name)
))

fig.update_layout(dict(
                    legend_title_text='Day Type',
                    hovermode='x'))

fig.update_xaxes(title='Time (hour)', fixedrange=True, range=[0,23])
fig.update_yaxes(title='Average People', fixedrange=True)

add_chart_title(fig, 'Hourly average of MRT user')

# set hour and anotation
set_hour = (6,11)

# peak hour annotation
fig.add_annotation(xref='x', x=8, yref='y', y=245500,
                   text='peak hour to put attention on', font_color='grey',
                   bgcolor='white', bordercolor='grey',
                   arrowcolor='grey', arrowhead=2)

# start hour annotation
fig.add_annotation(xref='x', x=6, yref='y', y=30000,
                   text='start operational<br> hour - 6AM', font_color='grey',
                   bgcolor='white', bordercolor='grey',
                   xanchor='right', arrowcolor='grey', arrowhead=2, ay=-100,␣
↪ax=-30)

# used data annotation
fig.add_shape(type='rect',
              xref='x', x0=set_hour[0], x1=set_hour[1], yref='paper', y0=0,␣
↪y1=1,
              fillcolor='grey', opacity=0.1, line_width=0)

fig.add_shape(type='rect',
              xref='x', x0=set_hour[0], x1=set_hour[1], yref='paper', y0=0,␣
↪y1=1,
              line_color='black')

fig.add_annotation(xref='x', x=11, yref='paper', y=0.8,
```

```
                    text='used data<br>to estimate passerby<br>from {} to {}'.
↪format(
                        set_hour[0], set_hour[1]), align='left',␣
↪font_color='black',
                    xanchor='left', ax=20, ay=0)

fig.update_layout(title='Passenger on 6-11 would would be used to simulate␣
↪passerby',
                  width=800, height=400)

fig.show(config=plotly_default_config_chart)
fig.write_image(os.path.join(ANALYSIS_DIR, 'used_hour.png'))
```



1.1.3 Conclusion

To estimate passerby number or other stuff, we would use MRT data from hour 6-11

1.2 Set radius on passenger walk distance

Because we just have the sample data of MRT passenger, we would like to predict population data by simulating walking radius.

[5]:
```
# setup data source:
# - taipei area data, village detail
village_shp_path = get_shp_filepath(os.path.join(BASE_DIR, 'data',␣
↪'taiwan_twd97_map_data_village'))
village_gpd = load_normalize_gov_shp_data(village_shp_path)
```

```
taipei_village_gpd = village_gpd[village_gpd['county_chinese_name'] == '  ']
taipei_village_gpd.set_index('village_code', drop=False, inplace=True)

# - taipei mrt data coordinate
taipei_mrt_coordinate_data_dir = os.path.join(BASE_DIR, 'data',␣
 ↪'taipei_mrt_map_coordinate')
taipei_mrt_map_coordinate_filepath = os.path.join(
    taipei_mrt_coordinate_data_dir, 'structured','taipei_mrt_map_coordinate.csv'
)
```

1.2.1 Visualize and set the radius in km

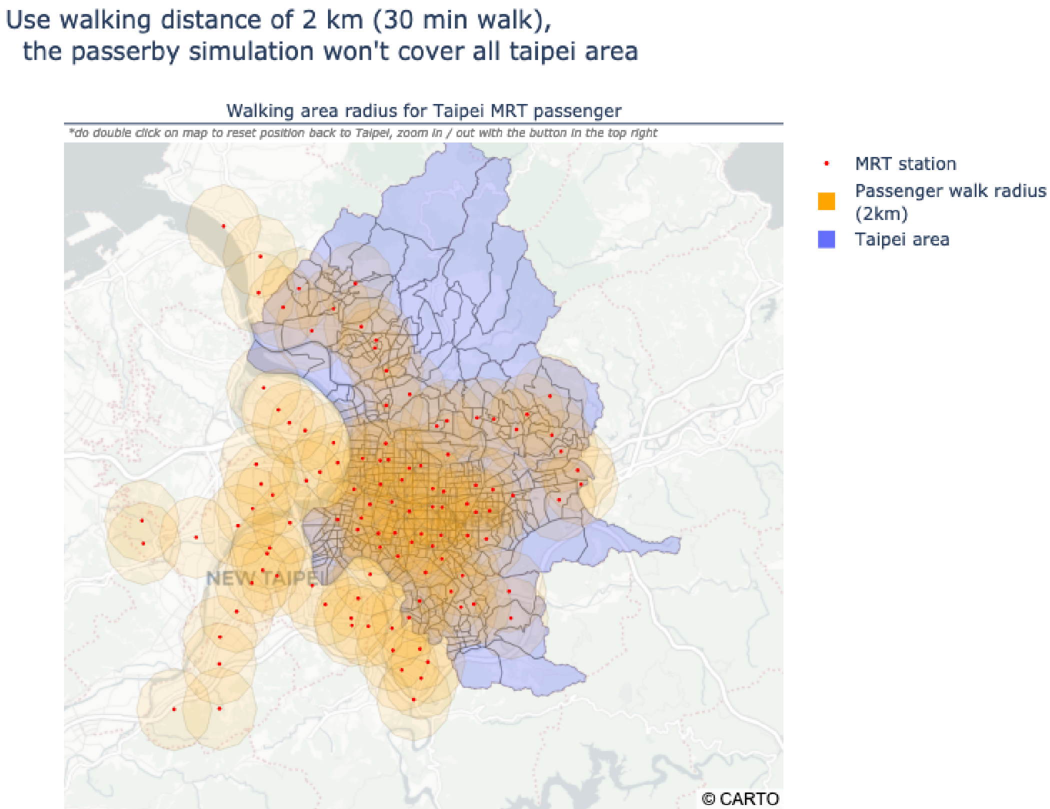
Check the radius throgh data visualization

[6]:
```
# prepare are data
taipei_village_geojson = json.loads(taipei_village_gpd.geometry.to_json())

center_point = MultiPoint(taipei_village_gpd['geometry'].apply(lambda x: x.
 ↪centroid)).centroid

# get location of the MRT station
taipei_mrt_map_df = pd.read_csv(taipei_mrt_map_coordinate_filepath)
taipei_mrt_map_df['village_code'] = taipei_mrt_map_df['village_code'].apply(str)
taipei_mrt_map_df = gpd.GeoDataFrame(
    pd.merge(taipei_mrt_map_df, village_gpd[['village_code', 'geometry']],
    how='left', on='village_code'
))

taipei_mrt_map_df.set_index('village_code', drop=False, inplace=True)

taipei_mrt_map_df['centroid'] = taipei_mrt_map_df['geometry'].apply(lambda x: x.
 ↪centroid)
```

[7]:
```
# plot the main graph
fig = px.choropleth_mapbox(taipei_village_gpd, geojson=taipei_village_geojson,
                           locations='village_code',
                           hover_name='village_english_name',
                           hover_data=['village_chinese_name',␣
 ↪'township_chinese_name'],
                           labels={'village_chinese_name': 'Village Chinese␣
 ↪Name',
                                   'township_chinese_name': 'Township Chinese␣
 ↪Name'},
                           opacity=0.3,
                           mapbox_style='carto-positron',
                           center={'lon':center_point.x, 'lat':center_point.y},
                           zoom=10)
```

```
fig.update_traces(dict(
                    name='Taipei area',
                    hovertemplate=fig['data'][-1]['hovertemplate']\
                        .replace('<br>village_code=%{location}<br>','')\
                        .replace('=', ' = ')))

add_chart_title(fig, 'Walking area radius for Taipei MRT passenger', 1.2)

add_chart_annotation(fig,
                     '<i>*do double click on map to reset position back to␣
↪Taipei, '
                     'zoom in / out with the button in the top right</i>')

radius_in_km = 2

taipei_mrt_map_df['centroid_circle'] = \
    taipei_mrt_map_df['centroid']\
    .apply(lambda x: x.buffer(radius_in_km / 111, resolution=3)
)

taipei_mrt_map_centroid_geojson = json.loads(taipei_mrt_map_df['centroid'].
↪to_json())
taipei_mrt_map_centroid_circle_geojson = json.
↪loads(taipei_mrt_map_df['centroid_circle'].to_json())

fig.add_trace(go.Choroplethmapbox(
                name='Passenger walk radius<br>({}km)'.format(radius_in_km),
                geojson=taipei_mrt_map_centroid_circle_geojson,
                locations=taipei_mrt_map_df['village_code'],
                z=[1]*len(taipei_mrt_map_df),␣
↪colorscale=[[0,'orange'],[1,'orange']],
                marker=dict(
                    opacity=0.1
                ),
                customdata=taipei_mrt_map_df['station_name'],
                hovertemplate='Station Chinese Name = %{customdata}',
                showlegend=True, showscale=False, ))

fig.add_trace(go.Scattermapbox(
                name='MRT station',
                lon=taipei_mrt_map_df['centroid'].apply(lambda x: x.x),
                lat=taipei_mrt_map_df['centroid'].apply(lambda x: x.y),
                marker=dict(
                  color='red',
                  size=3,
```

```
                  sizemode='area'
                ),
                customdata=taipei_mrt_map_df['station_name'],
                hovertemplate='Station Chinese Name = %{customdata}'))

fig.update_layout(dict(
    legend={'traceorder': 'reversed'}
))

fig.update_layout(dict(
    title=dict(
        text="Use walking distance of 2 km (30 min walk),<br>"
        "  the passerby simulation won't cover all taipei area",
        yanchor='top',
        yref='container', y=0.9,
    ),
    margin={'t':150},
    height=700
))


# fig.show(config=plotly_default_config_geo)
fig.show(config=plotly_default_config_geo)
fig.write_image(os.path.join(ANALYSIS_DIR, 'walking_radius.png'))
```



1.2.2 Make distribution formula

We would use the 2 km radius, which still possible (30 mins walks), even it is limited to some area. But we would create a really steep distribution function.

The distribution formula is based on: - previous maximum determined radius, 2 km - use function of mirror quadratic from $f(0) = 1$, $f(2) = 0$, $f(0.3) = 0.7$, $f(1) = 0.2$ that limit x from 0-2 with minimum value of 0 and maximum value of 1:

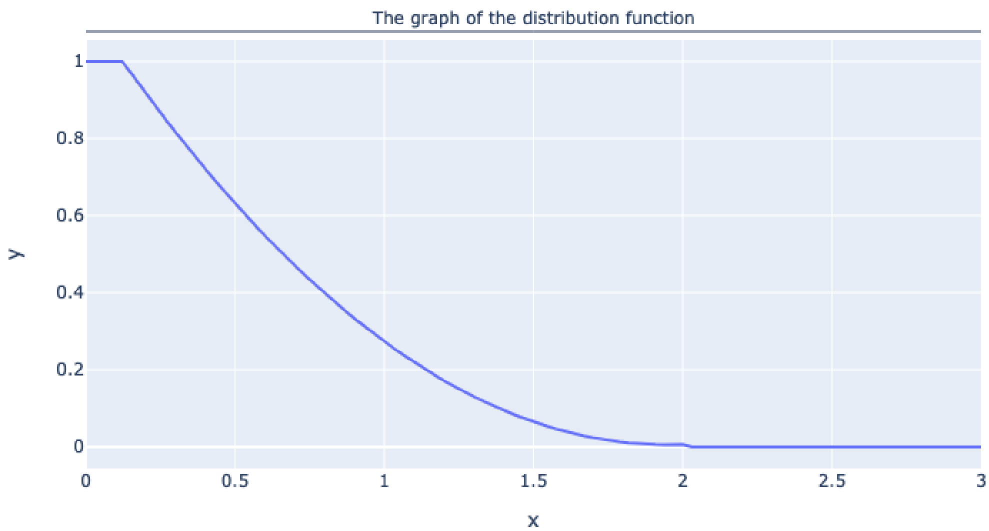
Therefore, the distribution value that we get is:

$$f(x) = \frac{3x^2}{10} - \frac{17x}{6} + 1 \tag{1}$$

with result no more than 1 or less than 0

[8]:
```
# simulating the graph
def distribution_function(x):
    if x > 2:
```

```
        return 0
    elif x <= 0:
        return 1
    else:
        fx = (3*(pow(x,2))/10) - ((7*x)/6) + 1.14
        if fx > 1:
            return 1
        elif fx < 0:
            return 0
        else:
            return fx

graph_x = np.linspace(0,3,100)
graph_y = [distribution_function(x) for x in graph_x]

fig = px.line(x=graph_x, y=graph_y)

add_chart_title(fig, 'The graph of the distribution function')

fig.show(config=plotly_default_config_chart)
fig.write_image(os.path.join(ANALYSIS_DIR, 'distribution_function.png'))
```



1.3 Predicting population passerby number

With the hour / sample data decided, ditribution formula decided, we would start predicting the population passerby number

[9]:
```python
# setup save path
data_mart_dir = os.path.join(BASE_DIR, 'data', 'aggregated-data_mart')

save_taipei_passerby_prediction_filepath = os.path.join(data_mart_dir,␣
↪'taipei_passerby_prediction.csv')

# setup the data source:
data_warehouse_dir = os.path.join(BASE_DIR, 'data', 'normalized-data_warehouse')

# - area dimension table
area_dimension_table = pd.read_csv('../data/normalized-data_warehouse/
↪area_dimension_table.csv')
area_dimension_table = area_dimension_table.astype({'village_code':str})
area_dimension_table.set_index('village_code', inplace=True)

# - taipei area data, village detail
village_shp_path = get_shp_filepath(os.path.join(BASE_DIR, 'data',␣
↪'taiwan_twd97_map_data_village'))
village_gpd = load_normalize_gov_shp_data(village_shp_path)

taipei_village_gpd = village_gpd[village_gpd['county_chinese_name'] == '  ']
taipei_village_gpd.set_index('village_code', drop=False, inplace=True)

taipei_village_gpd = pd.merge(
    taipei_village_gpd, area_dimension_table[['township_english_name']],
    left_index=True, right_index=True
)

# - taipei mrt passenger data -> taipei_mrt_info
taipei_mrt_info_dirpath = os.path.join(data_warehouse_dir, 'taipei_mrt_info')
taipei_mrt_info_urlpath = os.path.join(taipei_mrt_info_dirpath,␣
↪'taipei_mrt_passenger_data_*.csv')

source_df = dd.read_csv(taipei_mrt_info_urlpath)

# - taipei village distance matrix
taipei_village_distance_matrix_filepath =\
    os.path.join(data_warehouse_dir,␣
↪'taipei_village_centroid_distance_km_matrix.csv')
taipei_village_distance_matrix_df = pd.
↪read_csv(taipei_village_distance_matrix_filepath)
taipei_village_distance_matrix_df.set_index('village_code', inplace=True)
```

1.3.1 Predict sample-to-population multiplier value

We use this fact: - Goverment data in show that in 2019, the public transportation usage in Taipei was 49.4% - Goverment data show that Taipei population in 2016 was 2,695,704 people - Goverment data show that Taipei aging population (above 65) in 2016 was 419,130

Therefore, we would use the current sample (taipei MRT data) to predict the population using the fact. We would use this assumption: - Current MRT data distribution represent at least 80% overall Taipei people activities - Ignore the 20% unrepresented population passerby - Aging population would not having any activities at all (use for reducing overall passerby number)

$$\textit{sample-to-population multiplier} = \frac{(\textit{Taipei population} * 0.8) - \textit{Taipei elderly population}}{\textit{Total sample data}} \tag{2}$$

$$\tag{3}$$

The calculation on cell below would show the result is 2.75. Therefore we would use the multiplier 2.75 to predict passerby (population) data.

[10]:
```
df = source_df

df = source_df
df = df[(df['time_period'] >= 6) & (df['time_period'] <= 11)]

df_agg = df.groupby(['station_out_village_code', 'date'])['person_times'].sum()\
    .reset_index()

df_agg = df_agg.groupby('station_out_village_code')['person_times'].mean()\
    .reset_index()

taipei_daily_passerby_per_village = df_agg.compute()
taipei_daily_passerby_per_village.set_index('station_out_village_code',␣
↪inplace=True)

total_daily_average_mrt_passenger =␣
↪taipei_daily_passerby_per_village['person_times'].sum()

sample_to_population_multiplier = 2.75
```

1.3.2 Simulate walking distribution number

Based on previous part, we would use this distribution formula: the distribution value that we get is:

$$f(x) = \frac{3x^2}{10} - \frac{17x}{6} + 1 \tag{4}$$

with result no more than 1 or less than 0

[11]:
```
# calculate simulation using the formula
taipei_village_distance_matrix_df = taipei_village_distance_matrix_df\
    .applymap(lambda x: distribution_function(x))
```

```
taipei_daily_passerby_per_village_dict =\
    taipei_daily_passerby_per_village['person_times'].to_dict()

simulated_passerby_number_dict = {}
for index, row in taipei_village_distance_matrix_df.iterrows():
    if index in taipei_daily_passerby_per_village_dict:
        _distributed_passenger = \
            (row * float(taipei_daily_passerby_per_village_dict.get(index))).
 ↪to_dict()
        simulated_passerby_number_dict = {**simulated_passerby_number_dict,
                                          **{index: _distributed_passenger}}

taipei_simulated_passerby_df = \
    pd.DataFrame.from_dict(simulated_passerby_number_dict, orient='index')
taipei_simulated_passerby_df = taipei_simulated_passerby_df.apply(sum)

taipei_simulated_passerby_dict = taipei_simulated_passerby_df.to_dict()

# predict population data with sample to population multiplier
taipei_simulated_passerby_dict = \
    {k:sample_to_population_multiplier * v for k, v in␣
 ↪taipei_simulated_passerby_dict.items()}

taipei_village_gpd['simulated_passerby'] = taipei_village_gpd['village_code']\
    .apply(lambda x: taipei_simulated_passerby_dict.get(x))
```

1.3.3 Save and visualize data

After all of the data, we compute the prediction of the population data and visualize the data.

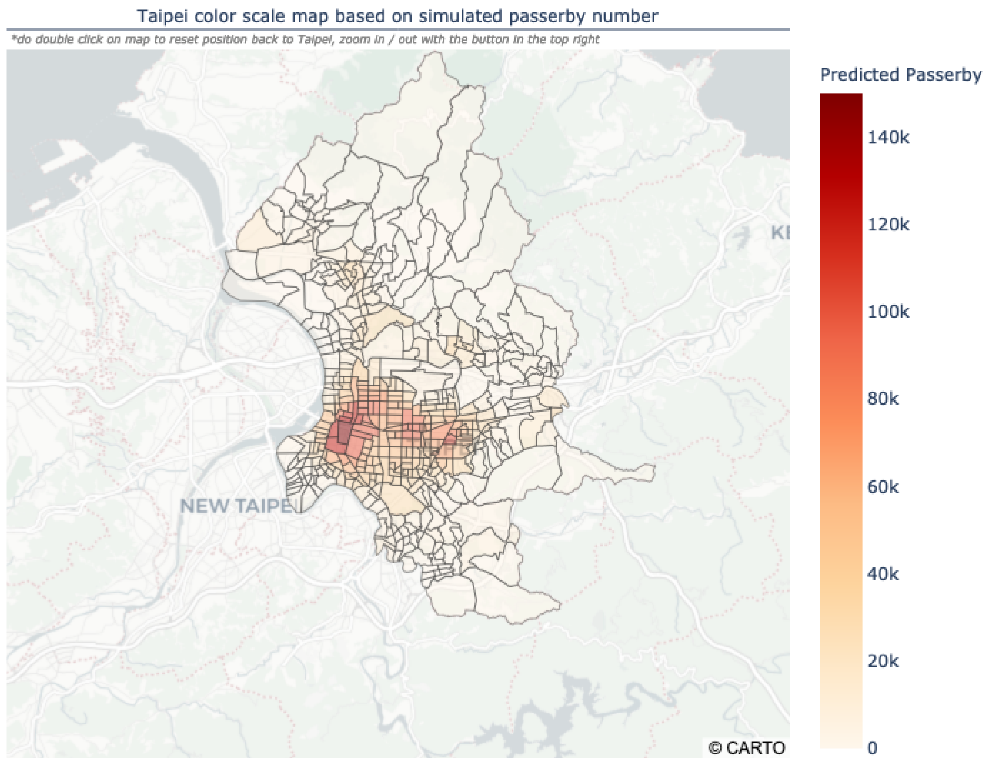
[12]:
```
# prepare the geojson data
taipei_village_geojson = json.loads(taipei_village_gpd.geometry.to_json())
center_point = MultiPoint(taipei_village_gpd['geometry'].apply(lambda x: x.
 ↪centroid)).centroid

taipei_township_passerby_agg = taipei_village_gpd.groupby(['township_code',␣
 ↪'township_english_name'])['simulated_passerby'].sum().reset_index()

taipei_township_passerby_agg.sort_values('simulated_passerby', ascending=False,␣
 ↪inplace=True)

save_df = taipei_village_gpd.loc[:, taipei_village_gpd.columns != 'geometry']
save_df.to_csv(save_taipei_passerby_prediction_filepath, index=False)
```

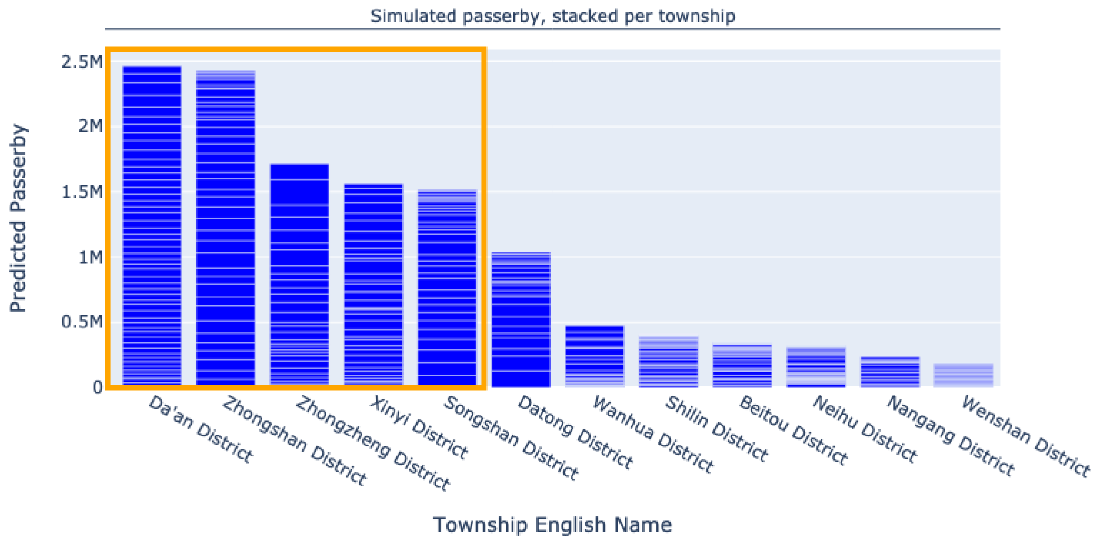
[13]:
```
# draw first chart the map
fig = px.choropleth_mapbox(taipei_village_gpd, geojson=taipei_village_geojson,
                           locations='village_code',
```

```
                            color='simulated_passerby',
                            hover_name='village_english_name',
                            hover_data=['township_english_name'],
                            labels={'township_english_name': 'Township English␣
↪Name',
                                    'simulated_passerby': 'Predicted Passerby'},
                            color_continuous_scale='OrRd',
                            range_color=(0,150000),
                            opacity=0.5,
                            mapbox_style='carto-positron',
                            center={'lon':center_point.x, 'lat':center_point.y},
                            zoom=10)

fig.update_traces(hovertemplate=fig['data'][-1]['hovertemplate']\
                  .replace('village_code=%{location}<br>','')\
                  .replace('=',' = ')\
                  .replace('{z}','{z:,.2r}')
                 )

add_chart_title(fig, "Taipei color scale map based on simulated passerby␣
↪number", 1.2)

add_chart_annotation(fig,
                     '<i>*do double click on map to reset position back to␣
↪Taipei, '
                     'zoom in / out with the button in the top right</i>')

fig.update_layout(
    title='Most of passerby are in Taipei mid-west area',
    margin={'t':120},
    height=700
)

fig.show(config=plotly_default_config_geo)
fig.write_image(os.path.join(ANALYSIS_DIR, 'predicted_passerby_number-1.png'))

# draw second chart, top 5 bar chart
fig = px.bar(taipei_village_gpd,
             x='township_english_name',
             y='simulated_passerby',
             labels={
                 'township_english_name': 'Township English Name',
                 'simulated_passerby': 'Predicted Passerby',
                 'village_chinese_name': 'Village Chinese Name'
             },
             color='village_chinese_name')
```

```
fig.update_traces(hovertemplate=fig['data'][-1]['hovertemplate']\
                  .replace('=',' = ')\
                 )

fig.update_traces(marker={'color': 'blue'})

fig.update_xaxes(categoryorder='array',
                ␣
 ↪categoryarray=taipei_township_passerby_agg['township_english_name'])

fig.update_layout(showlegend=False)
fig.update_xaxes(fixedrange=True)
fig.update_yaxes(fixedrange=True)

add_chart_title(fig, "Simulated passerby, stacked per township", 2)

fig.add_shape(type='rect',
              xref='x', x0=-0.6, x1=4.5, yref='paper', y0=0, y1=1,
              line=dict(
                  color='orange',
                  width=4
              ))

fig.show(config=plotly_default_config_chart)
fig.write_image(os.path.join(ANALYSIS_DIR, 'predicted_passerby_number-2.png'))
```

Most of passerby are in Taipei mid-west area




Simulated passerby, stacked per township
Predicted Passerby
2.5M
2M
1.5M
1M
0.5M
0
Da'an District
Zhongshan District
Zhongzheng District
Xinyi District
Songshan District
Datong District
Wanhua District
Shilin District
Beitou District
Neihu District
Nangang District
Wenshan District
Township English Name