Expanding MTA Accessibility

Binh, Eliza, Meehir, Roman, Tolu

Introduction

A Mother's Fatal Fall on Subway Stairs Rouses New Yorkers to Demand Accessibility

- **Problem:** in Manhattan alone, 102/152 stations are not accessible
  - Dangerous for parents with young children and the disabled/elderly

- **Goal:** recommend MTA where to install next 5 station elevators in Manhattan

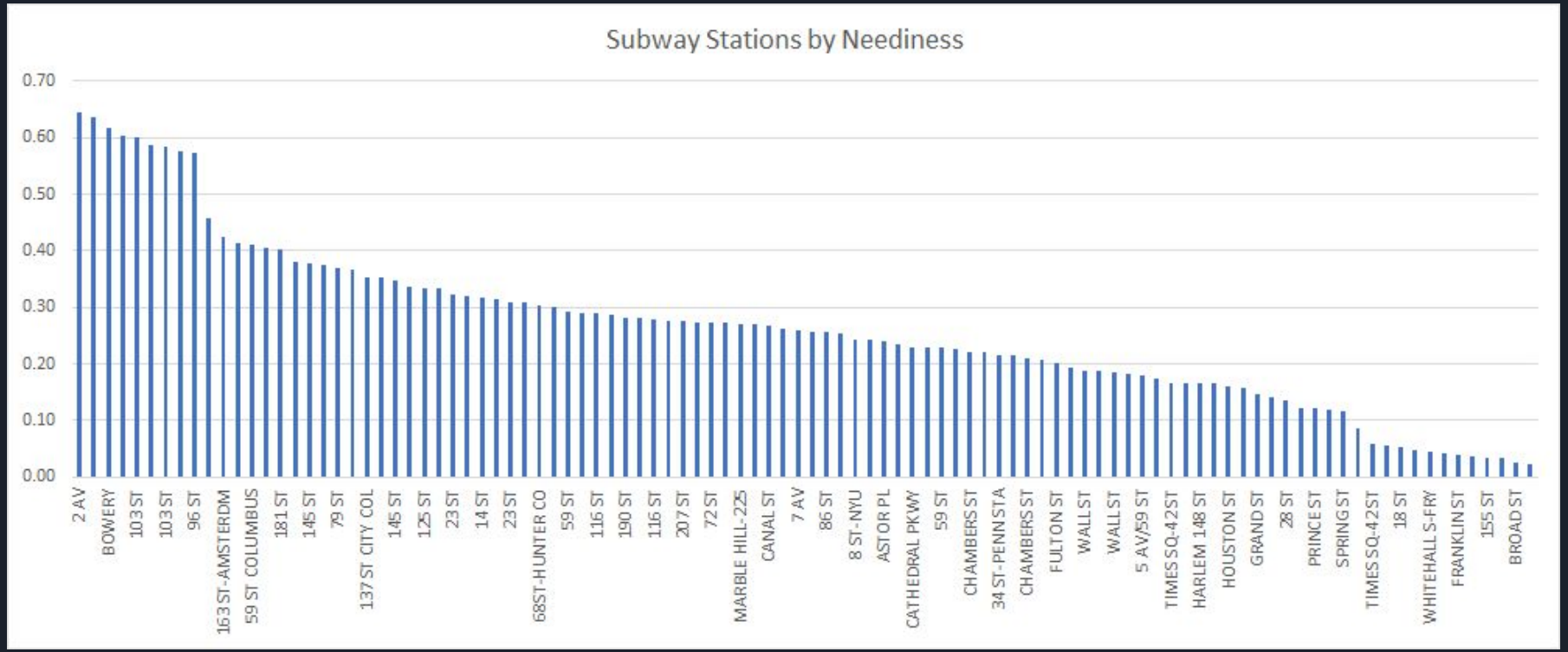
Methodology

- Data sources
  - MTA turnstile data (Sep - Nov 2019): station usage (entries)
  - Google API: station zip codes, longitude/latitude
  - Census Bureau American Community Survey (ACS): demographics

- Key metrics
  - Station entries (usage) - each station mapped to zip code
  - Population of disabled people by zip code
  - Population of young children and seniors by zip code

- Created composite “neediness” metric
  - Normalized each key metric by dividing individual values by max value (creates normalized scores between 0 and 1)
  - Took average of normalized scores for each metric to arrive at “neediness” score between 0 and 1

Subway stations ranked by neediness (composite score)

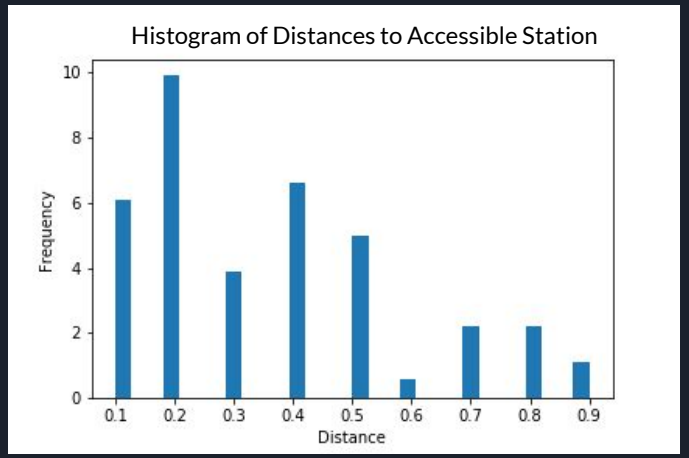


Results: radar charts

Conclusions

- ★ Five stations to install elevator
  1. 2 Avenue on F line
  2. East Broadway on F line
  3. Bowery Street on JZ line
  4. 103 Street on 6 line
  5. 103 Street on 1 line

- ★ The distance between a non-accessible station to an accessible station can be as great as .9 miles



Future work

- Standardize MTA station name without manual adjustments
- Further investigation:
  - Day of week/hourly trends
  - Use entries AND exits
  - Estimate percent of people who use station AND reside in that station's zip code
- Set different weights for disability, age, and subway entries scores

Future work: Geospatial Analysis

Appendix

Parsing through inconsistent naming in dataframe columns

```
no_elevators=pd.read_csv('Stations With No Elevator_updated.csv')
```

```
count=0
lst=[]
for index, row in no_elevators.iterrows():
    for i, r in stations_total.iterrows():
        if row["Stop Name"].upper()==r["STATION"] and row['Daytime Routes'] in r['LINENAME']:
            lst.append(r)
            count+=1

print(count)
dat=pd.DataFrame(lst)
dat.head()
```

118

	index	STATION	LINENAME	ENTRIES	PREV_ENTRIES	DAILY_ENTRIES
117	0	1 AV	L	108210509971	1.082092e+11	1298171.0
347	3	103 ST	BC	2747657700	2.747271e+09	386482.0
171	1	103 ST	1	5901511168	5.900530e+09	981372.0
160	2	103 ST	6	4361693836	4.360641e+09	1052643.0
192	7	110 ST	6	3525627318	3.524766e+09	861425.0

```
#dat.to_csv("dat.csv")
```

```
duplicateDFRow = dat[dat.duplicated()]
print(duplicateDFRow)
```

```
     index          STATION      LINENAME       ENTRIES  PREV_ENTRIES  \
55      31            145 ST          ABCD   10532589724  1.053063e+10
24     269     DELANCEY/ESSEX          FJMZ    6832262457  6.830297e+09
8      301         FULTON ST      2345ACJZ  141641492745  1.416370e+11
37     302         FULTON ST      ACJZ2345  285343602498  2.853413e+11
8      301         FULTON ST      2345ACJZ  141641492745  1.416370e+11
37     302         FULTON ST      ACJZ2345  285343602498  2.853413e+11
0      311   GRD CNTRL-42 ST         4567S  226865326108  2.268583e+11
0      311   GRD CNTRL-42 ST         4567S  226865326108  2.268583e+11
9      447    TIMES SQ-42 ST  1237ACENQRSW  506048173890  5.060434e+11
71     446    TIMES SQ-42 ST   1237ACENQRS   10503475931  1.050172e+10
137    448    TIMES SQ-42 ST  ACENQRS1237W    2493827302  2.492658e+09
9      447    TIMES SQ-42 ST  1237ACENQRSW  506048173890  5.060434e+11
71     446    TIMES SQ-42 ST   1237ACENQRS   10503475931  1.050172e+10
137    448    TIMES SQ-42 ST  ACENQRS1237W    2493827302  2.492658e+09
```

Renaming columns for clarity:

```
stations_zips = pd.read_csv("sub_st_zip.csv")
no_elevator_stations_total =pd.read_csv("no_elevators_station_total.csv")
```

```
stations_zips.rename(columns={'Stop Name':'STATION', "Zip Code":"ZipCode"}, inplace = True)
```

Combining dataframes of zip codes and stations using zip code dictionary:

```
zip_dict  = pd.Series(stations_zips.ZipCode.values,index=stations_zips.STATION).to_dict()
```

```
no_elevator_stations_total["ZipCode"] = no_elevator_stations_total["STATION"].map(zip_dict)
```

```
manh_stations_total=no_elevator_stations_total[["STATION","LINENAME","ENTRIES","PREV_ENTRIES","DAILY_ENTRIES","ZipCode"]]
```

Mapping:

```
aged_dict  = pd.Series(age_disability.aged.values,index=age_disability.ZipCode).to_dict()
```

```
manh_stations_total["aged"] = manh_stations_total["ZipCode"].map(aged_dict)
```

Normalize by max value and create composite score called “Neediness” based on average normalized scores:

```
manh_stations_total["disabled"] = manh_stations_total["ZipCode"].map(disabled_dict)
```

```
manh_stations_total["aged_scaled"] = manh_stations_total["aged"]/max(manh_stations_total.aged)
```

```
manh_stations_total["disabled_scaled"] = manh_stations_total["disabled"]/max(manh_stations_total.disabled)
```

```
manh_stations_total["entries_scaled"] = manh_stations_total["DAILY_ENTRIES"]/max(manh_stations_total.DAILY_ENTRIES)
```

```
manh_stations_total["Neediness"] = (manh_stations_total.aged_scaled + manh_stations_total.disabled_scaled + manh_stations_t
otal.entries_scaled)/3
```

Sorting neediness and dropping na values:

```
#Sort by neediest stations
manh_stations_total=manh_stations_total.sort_values("Neediness", ascending = False).dropna()
```

Google Maps API use

Using findplace API for zip codes

```
#quering google geomaps api to determine distance from each station w/o elevator to every station w/ ele
for i in range(len(no_elev_st_coor)):
    url4 = "https://maps.googleapis.com/maps/api/distancematrix/json?units=imperial&origins="
    url5 = "&destinations="
    url6 = "&mode=walking&key=AIzaSyDN9Bns2x7oLoXlf6lDYtMaWNCrtXkIaAU"

    #makes list of distance to every station w/ elevator, selecs min value and adds it to "dist to elev"
    mindist= list()
    for j in range(len(elev_st_coor)):
        url_dist = urllib.request.urlopen(url4+str(no_elev_st_coor[i])+url5+str(elev_st_coor[j])+url6)
        data = url_dist.read()
        i1 = str(data).rfind(' mi",')
        mindist.append(str(data)[i1-3:i1])
    min_dis = float(min(mindist))
    min_dist_lst.append(min_dis)
    sub_st_df["Dist to Elev"][index_no_elev[i]] = min_dis

    #used for live tracking
    print("i= {}, min dist {}".format(i, min_dis))
    print(sub_st_df.loc[index_no_elev[i]])
```

Using distancematrix API for distances

```
#Taking column Stop Name and exporting it as a list
mta_stations = list(sub_st_df["Stop Name"])

# link to Google Maps API is broken up into 3 parts, part 1 and part 3 are static hence declared before iteration

# Part 1 up to input
url1 = 'https://maps.googleapis.com/maps/api/place/findplacefromtext/json?input='

# Part 2, since we only have name of the station in our list we add "Subway Station New York" to the end of each input
# Our only query is "formatted_address" field and last part is the API key
url2 = 'Subway%Station%New%York&inputtype=textquery&fields=formatted_address&key=AIzaSyDN9Bns2x7oLoXlf6lDYtMaWNCrtXkIaAU'

# Indexing through every row of our df
for i in sub_st_df.index:

    #since API will return errors must use "try" function
    try:

        #combining all parts into one url
        ur = urllib.request.urlopen(url1+str(mta_stations[i]).replace(' ',"%")+url2)

        #reading response from API
        data = ur.read()

        #since feedback is a string we use find function to locate zipcode in it
        i1 = str(data).rfind(', United States')

        #insert zipcode into our df
        sub_st_df["Zip Code"][i] = (str(data)[i1-5:i1])
```