

# New York City MTA Service Alert

## Term Project Report

Yassine Kadiri (yk1859), Samuel Forman (srf366), Fang Han (fh643)

December 2017

## 1 Introduction

The problem our project seeks to address is the lack of simple notification systems that alert users when there are service issues on their train line (MTA NYCT). We sought to address this problem by producing a program that regularly checks NYCT subway service status feeds online and alerts users before their commute starts if their subway lines are having service problems.

## 2 Business understanding

This project is inspired by an issue many of us have experienced traveling around NYC: we arrive at the subway station expecting a train within a few minutes only to find a major delay waiting for us. Because these experiences are infrequent we do not prepare for them daily by checking the subway service status before we leave our homes. But when the subways are delayed the impacts can be major - and avoidable. That's why we decided to address this problem, because sometimes knowing of subway problems even 10 minutes in advance can help us avoid a transportation mess that can lead to missing work or a host of other negative outcomes.

## 3 Methodology

### 3.1 Service Status Scraping

One foundational part of dealing with the NYCT subway service data is taking the service data, which is shared in a text format on the web that is a combination of XML and HTML text, and processing it to pull out the important elements. To do this we started with two external packages. XML Element Tree for dealing with the XML structure of the service status data and BeautifulSoup for parsing the service

status descriptions, which are provided in HTML. Once we extracted the Service Status information, we decided to keep only the lines that face an altered service. Then, we created a dictionary with those lines, each line being a key for only two possible values: delayed or works. This would help afterwards to easily know what is the status for a given line and therefore call the appropriate functions when looking for details about the delays (`delaystext` function) or the planned works (`Planned_work_Text` function)

### 3.2 Notification System and Data Structures Used

After that, we decided that an appropriate data structure to keep our data would be a DataFrame. This DataFrame has 13 rows (user, commuting time, and all the lines) and for each user, we will create a new column containing all of those informations.

For the user and his/her commuting time, we use strings. The commuting time comes in 24 hour format as a string hh:mm. Regarding the lines, they come as bits. If the user takes a line, then in the row corresponding to that line in the DataFrame, the bit will be set to 1. If not, it will be set to 0. This is why we created functions to encode a list of lines into a list of bits and a function to decode a list of bits into the corresponding list of lines.

Now comes the important part. We decided that we would warn users X minutes before their commuting time if there are any delays. This values X is determined by the variable `time_window` in our code. To get the list of people concerned by the delays or works, we extract all the lines with an altered service, convert them into a list of bits. Then, we do a for loop on the DataFrames columns: in each column, if the user has a commuting time that is X minutes from now, we check whether he takes one of the altered lines (using a logical and on the vectors of bits and summing on all bits), if so, we add him to the list of people to notify. This is what our `listToNotify` function does. Once we have this list, we use it to send personalized emails to the users thanks to the `emailTrigger` and `sendmail` functions.

We rerun this process every minute in order to ensure that every user gets notified.

### 3.3 Web Interface

Our web interface is built in the library CherryPy, which allows for quick website generation using structures like classes, methods, and function to create servers, specific pages, and actions within the pages etc.

We started by building a web form that intakes the necessary information from the user. This form is linked to a function `updtusrstring` that takes the user input and stores it in a dictionary on the

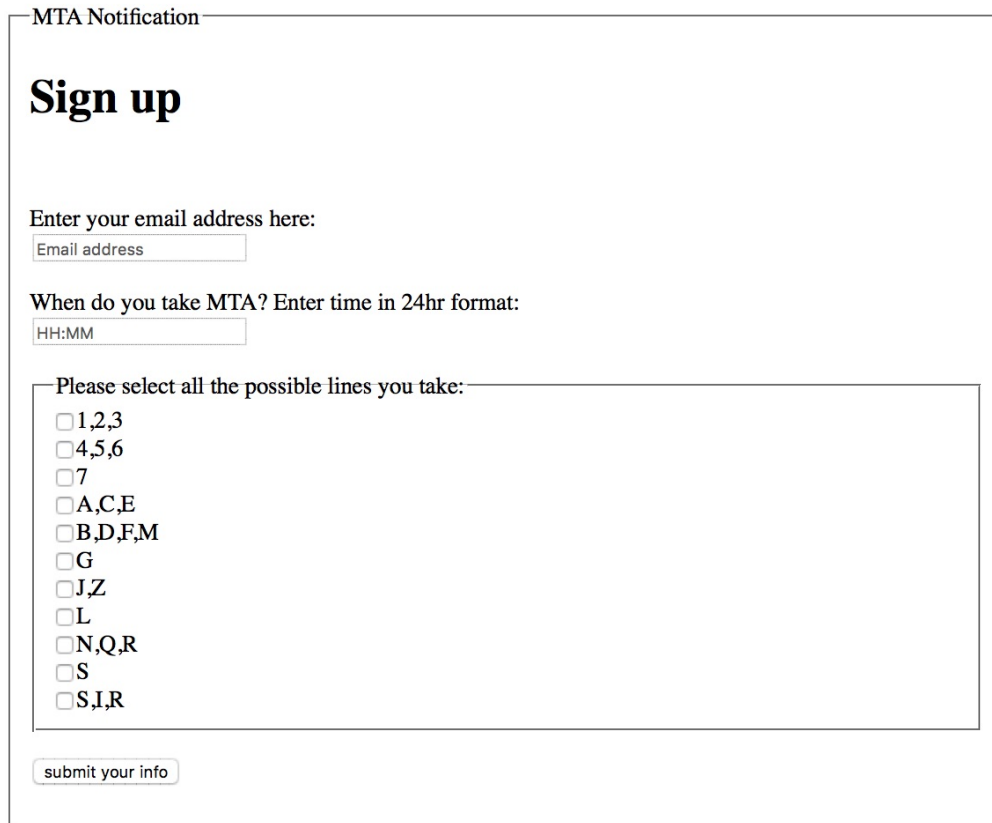
(local) server. This dictionary can be updated with new user information by anyone accessing it while our program is running.

The dictionary with the user data in it is then converted to a JSON and exposed using the function `curr_user_dict`. That data is then accessed in the main program `Core_Code_Final` using a `request.get` function aimed at the `curruserdict` function on the local server. The `request.get` function accesses the current user data dictionary and incorporates it into the main code for processing so users can immediately receive emails with subway service alerts.

## 4 Results

Now that everything was merged, we were able to "launch" the website and try our code to see if indeed they communicated well.

First, we execute the website's notebook which allows us to locally connect to the website. Once we connect, we get the following page:



The image shows a web browser window displaying a sign-up form for MTA notifications. The page has a title "MTA Notification" at the top left. Below it is a large heading "Sign up". The form contains three main sections: 1. "Enter your email address here:" with a text input field labeled "Email address". 2. "When do you take MTA? Enter time in 24hr format:" with a text input field labeled "HH:MM". 3. "Please select all the possible lines you take:" followed by a list of checkboxes and line names: ☐ 1,2,3; ☐ 4,5,6; ☐ 7; ☐ A,C,E; ☐ B,D,F,M; ☐ G; ☐ J,Z; ☐ L; ☐ N,Q,R; ☐ S; ☐ S,I,R. At the bottom of the form is a button labeled "submit your info".

Figure 1: Web Interface

In this page, the user can enter his informations: email, commuting time and which lines he takes in his commute. Once he fills in this information, he gets redirected to the following page:

You just inserted commuter info for yk1859@nyu.edu.

[Click here to go back to main page](#)

[Click here if you are done entering user info.](#)

Figure 2: Interface

Once the user entered his information, we then have his data synchronized with the general code. Then, after setting the `time_window` variable, he will get a notification `time_window` minutes before his commuting time only if he is taking a line with an altered service.

Here is an example of one of the emails we send to a user impacted by delays or planned works:

Our emails provide information on which lines are affected and the nature of the delays or works.

## 5 Possible further directions

One immediate possible further direction would be to propose alternative routes to the user when he is notified. This would allow to provide a better service. Another direction is to work on the website to put it online and create a real database to store the user's information rather than store it locally. Finally, one ultimate step would be to get more granular information as delays start at one station and affect the whole network. Detecting delays per station would allow to quickly warn users not only on delays but on immediate upcoming delays. For now, we have a working website with working notifications so we already have a good baseline to work on those directions.

## 6 Conclusion

In conclusion, we were able to use the knowledge earned in this course to build a working website and notification system. Our aim was to work on a meaningful project that would allow us to discover Python and go even further than the scope of this course as we discovered useful libraries in Python along with pandas, numpy that can be used to build such a notification system.

ALERT: Service Change on line(s) 123, 456, 7, ACE, BDFM, NQR, S and SIR Inbox x



notificationmta@gmail.com

to me ▾

Dear user,

Please be aware of the following service changes that might affect your commute:

Lines 123

Status: Planned Work

TRACK & TRACK PLATE INSTALLATION, REPLACEMENT OF POWER & COMMUNICATION CABLES[2] [3] Trains run at reduced speed through the Clark S

Lines 456

Status: Planned Work

ELECTRICAL IMPROVEMENTS[6D] Brooklyn Bridge-bound trains run local from Parkchester to 3 Av-138 St

ELECTRICAL IMPROVEMENTS[6] The last stop for alternate trains headed toward Pelham Bay Park is 3 Av-138 St

ELECTRICAL IMPROVEMENTS[6] Pelham Bay Park-bound trains run express from 3 Av-138 St to Pelham Bay Park

TRACK REPLACEMENT[4] Woodlawn-bound trains skip 161 St, 167 St, 170 St, Mt Eden Av and 176 St

Lines 7

Status: Planned Work

ELECTRICAL IMPROVEMENTS[7] Flushing-bound trains skip 33, 40, 46, 52, 69, 74, 82, 90, 103 and 111 Sts

Lines ACE

Status: Delayed

[E], [F] and [M] train service has resumed following an earlier incident of a customer struck by a train at 50 St

Lines BDFM

Status: Delayed

[E], [F] and [M] train service has resumed following an earlier incident of a customer struck by a train at 50 St

Lines NQR

Status: Planned Work

TRACK MAINTENANCE[W] No trains between 57 St-7 Av and Ditmars Blvd - Take the [N] instead[W] trains run every 20 minutes in Manhattan

TRACK MAINTENANCE[N] Astoria-bound trains skip 39 Av and Broadway

Lines S

Status: Planned Work

TRACK REPLACEMENT[A] Trains replace the [S] Rockaway Park Shuttle

Lines SIR

Status: Planned Work

SCHEDULED MAINTENANCE[SIR] Trains board at the St. George-bound platform from Grant City to Great Kills

Thanks for using our platform,

The subway helper team

Figure 3: Email sent by our notification system