ELE 408 Semester Project: COVID-19 Appointment Tracker and Predictor

Justin Hall
URI, Computer Engineering '21

Brianna MacDonald
URI, Computer Engineering '21

***Abstract*—This document serves to inform the reader on our semester project for ELE 408: our COVID-19 Appointment Tracker & Predictor. It will provide an overview of the project, our achievements, and the work done to achieve our final outcome.**

I. Introduction

When tasked with designing an embedded application, we had desired to create something that was useful and helped solve a relevant problem in today's technology-centric world. With a suggestion provided by Prof. Bin Li, we decided to create an application to assist individuals with the location and registration of COVID-19 vaccination appointments.

II. Project Overview

A. Motivation

At the time of initially having this idea, it had been rather difficult for one to find a vaccination clinic that had open appointments due to the large demand of vaccinations themselves. Our goal was to create an application that obtained appointment availability data and use it to display a more concise representation of clinic availability, making it easier for users to find and make a vaccination appointment. The data would also be used in a machine learning context to predict a forecast of availability for a particular day or time.

B. The Problem

The main difficulty in finding and registering for a vaccination appointment comes from the fact that, because demand is so high, appointments are quickly snatched up by those watching closely for new openings. Not everyone has the time or capability to be watching these vaccination websites with the possibility of securing an appointment soon after it becomes available. Another problem is being able to view all of these appointments in an intuitive and readable format without having to have dozens of tabs open at a single time.

C. Our Proposed Solution

Our aim is to provide users with an automated solution for determining the availability of vaccination appointments using a Raspberry Pi. In doing so, we hope to provide better ease-of-access regarding this highly important form of health care. Ideally, users would be able to see a consolidated view of clinics and the number of appointments they have available, thus eliminating the need to search the many of pages of clinics in hopes of finding one that has openings. Users would also receive a prediction as to when the most optimal time to look for and book an appointment would be depending on a day or time of their choosing.

III. System Overview

The following section will describe the three major functional components of our final product. These three major components include: data collection, data formatting and display, and machine learning using our collected data.

A. Data Collection

The foundation of our project could be considered the data collection functions, as they are necessary in order to make the other functional components possible. Data collection was implemented using the popular Python-based Selenium web scraping library. This library, alongside an instance of the Chromium web driver, is capable of browsing web pages and their respective source HTML codes under the control of a Python script. From source HTML code, we are able to pull data such as clinic names, locations, appointment dates and times, and availability.

Our primary source of data is the site *vaccinateri.org*. We decided to focus on this particular site because it lists many clinics – whether they are available or not – and does not require any forms of authentication to be able to view this data. Many other vaccination sites require some form of account creation in order to even view available appointments, which would have made the automation of data collection significantly more time-consuming and difficult. In the interest of time, we decided to make *vaccinateri.org* our primary source.

The *vaccinateri.org* site lists clinics throughout multiple pages. If a clinic has openings, the user has the ability to go to that clinic's page, which displays appointment time slots and their respective amount of appointment openings. We use Selenium and the Chromium web driver to navigate through each page of clinics, and for each available clinic, pull the times and number of appointments from the table shown on a clinic's page. This data is then exported to comma-separated value files (.CSV) used for its display, as well as the machine learning predictions.

B. Data Formatting and Display

With a data collection and web scraping system working, our next piece of functionality involves displaying it in a way that makes it easy for users of our application to see vaccination clinics and their availability.

One way in which we accomplish this is by creating a Python-based web server that runs on the Raspberry Pi. On this web server, we host two pages: A Welcome page, and an Appointment Tracker page. The Appointment Tracker page, upon being visited by a client, displays the most recently collected appointment data from all available clinics on *vaccinateri.org*. Rather than needing to navigate each search page on *vaccinateri.org* to find a clinic's availability, the Appointment Tracker page displays a table of found clinics, with the total amount of openings for a given date in the future. This makes it easy for a user to then go to an open clinic's registration page to make an appointment faster than they might have been able to otherwise.

Another way in which we display appointment availability is by utilizing the LED matrix on the Raspberry Pi's SenseHAT module. Specifically, the LED Matrix displays a particular number of pixels in either red, yellow, or green colors corresponding to the amount of appointments currently found to be available. If there are between zero and one appointments available, a singular red pixel will be displayed on the matrix. If there are between two and ten total available appointments available, the corresponding number of pixels will be displayed in yellow. If there are eleven or more total available appointments, the corresponding number of pixels will be displayed on the matrix in green. If there are any more than sixty-four total appointments available, the entire LED matrix will display green pixels, as there are only sixty-four pixels in the LED matrix. The total number of appointments is updated after each round of web scraping, which takes anywhere from one to ten minutes to complete depending on the number of clinics with appointment openings.

C. Machine Learning Predictions

The Machine Learning Model utilizes the statsmodel python library, described later in the next section. ARIMA, short for 'Auto Regressive Integrated Moving Average' is a class of models that 'explains' a given time series based on its own past values, that is, its own lags and the lagged forecast errors, so that equation can be used to forecast future values. This kind of model was chosen as it does relatively well with time-series data that is gathered periodically.

To clean and preprocess this data for input into this model, we first had to clean the data by getting rid of any missing or NaN values, and then by making sure that the "Add To Waiting List" row was deleted from each of the appended CSV files that were gathered from the web-scraping program. This was done by looking at the associated row values for any NaN or missing values, and then making sure that any variation of "Add To Waiting List" was successfully removed or replaced from the CSV file (depending on the other values contained within that CSV file).

The model makes predictions as follows:

The model will take data as a time series and be able to predict number of available appointments by forecasting new data on historical time series data that we have gathered beforehand, as well as data that is gathered continuously from the web-scraping program. This can be done by looking at rate-of-change between different timestamps when data was collected, or having the user input a specific time or time-range to calculate predictions of available appointments. In this case, our timeseries is univariate (with only one variable being observed) and is then classified into being unavailable or available. The ARIMA model is then used to forecast new predicted appointment availability given this historical periodic timeseries data. This program then prompts the user for input regarding the day or week they would like to forecast appointments, as well as the desired time or time range that they would like to see the forecasts for.

In the future, we have the goal of implementing a location-services module which will utilize Google Maps API that will prompt the user for their current location or the location in which they want to make their vaccination appointment. Additionally, we would also like to make this model more accurate by having it handle more input data, as well as calculate missing or NaN values using a different more intuitive method than calculating based on the median or replacing it with a zero value.

IV. System Setup

Multiple requirements must be satisfied in order to successfully run our Python application on the Raspberry Pi. These requirements can be broken up into three sections: Python libraries, the Selenium web driver, and application directory structure.

A. Python libraries

The following python libraries must be installed locally in order to make use of the application. We installed them using the built-in pip3 Python package installer within the Raspbian OS.

1) Pandas: A library used for the formatting of collected data, as well as exporting data to comma-separated value files.

Pip3 install pandas

2) Selenium: The library used for automated web navigation and data collection. This library contains the functions necessary to be able to read HTML source code and collect data from fields based on various critera.

Pip3 install selenium

3) SenseHAT library: The library containing functions for interaction with the SenseHAT module. For our use case, it contains the functions necessary to display data on the LED matrix.

Pip3 install sense_hat

4) Flask: A library capable of hosting a web server. This is used to host the pages upon which collected data is displayed.

Pip3 install flask

5) Statsmodels: The statsmodels library provides a suite of functions for working with time series data that we will use as input for our ML model.

Pip3 install statsmodels

6) SKLearn: Scikit-learn (used here as SKLearn) is a free software machine learning library for the Python programming language that contains a variety of machine learning models and algorithms.

Pip3 install sklearn

7) MatPlotLib: A library capable of generating graphs and figures based on collected data and forecasts. For our program, this library will be used to plot future appointment avaiablility, as well as plot additional graphs regarding accuracy and seasonal decomposition trends.

Pip3 install matplotlib

When running the application on a Raspberry Pi in its out-of-the-box configuration, the above libraries should be the only ones that require installation. Any other libraries used by the application should already be installed with the Python installation that comes with the Raspbian OS.

B. Selenium Web Driver

For the Selenium driver to be useful, a web driver executable must be installed on the system. Different web browsers such as Google Chrome, FireFox, etc. make their

own versions of a driver available for use with web scraping utilities. The driver, unlike a normal installation of a web browser, has the capability to be controlled by Python code.

We use the Chromium project's implementation of a web driver in our application. This driver is available for Windows, MacOS, and Linux. One caveat is that the versions of the Chromium web driver hosted on their site are only compiled for x86 architectures. As such, we must utilize a special version compiled for ARM as the Raspberry Pi runs on an ARM-based CPU. This ARM-compatible version of the web driver can be found packaged with our application's source code. Theoretically, if running the application on a system that is not a Raspberry Pi, the corresponding web driver version can be used in place of the one we have provided without any additional configuration.

For a Raspberry Pi implementation of this application, the web driver executable should be located within the application's root directory. The ARM-compatible executable is included in the application's root directory in our project's archive.

C. Application Directory Structure

The successful operation of the application also relies on a proper directory structure within the application's root directory. When running this application, the structure of the files should be as follows:

- The Python application .py files must be present in the root directory. They shall not be nested within other folders.
- "templates" folder: This folder is home to all of the html files required for hosting the web server. It should be located within the root directory.
- "data" folder: This folder is home to the comma-separated value (.CSV) files generated by the web scraping process. Within this folder are more folders corresponding to the date and time in which the web scraping process created them.
- "data_simple" folder: This folder is home to the consolidated appointment data generated by the web scraping process. It is used primarily for the display of appointment data on the web page.
- "static" folder: This folder is home to the cascading style sheet files used for styling the webpage. It should be present within the project's root directory.

The application directory structure is set up and included in our project's submission archive. It may be necessary for the user to update the Python code to reflect the directory names in which the program is going to be stored on their system.

Acknowledgments

We would like to thank our professor, Bin Li, for the initial project idea as well as guidance on the steps we should take when implementing the initial program and its functions.

References

[1] Brownlee, Jason. "How to Create an ARIMA Model for Time Series Forecasting in Python." Machine Learning Mastery, 9 Dec. 2020, machinelearningmastery.com/arima-for-time-series-forecasting-with-python/.

[2] Introduction to ARIMA Models, Duke University, people.duke.edu/~rnau/411arim.htm.

[3] prabhat9. "How to Create an ARIMA Model for Time Series Forecasting in Python?" Analytics Vidhya, 29 Oct. 2020, www.analyticsvidhya.com/blog/2020/10/how-to-create-an-arima-model-for-time-series-forecasting-in-python/.