Nick Langan - CSC 4797 Capstone

The Florence, NJ & Tabernacle, NJ FM DX website

User Guide & Project Documentation

Table of Contents

Essential URLs

Website URL: http://wnjl.altervista.org/

GitHub code repository for website and python script: https://github.com/W2NJL/CSC-4797-Capstone

Introduction

For my Capstone project, I aimed to produce an output that I could make tangible use out of in my life's endeavors. A hobby that I've poured quite a bit of time into within the last 15 years of my life is DX'ing the FM radio band. In layman's terms, to "DX" is to listen for distant signals. For example, from my home-base in Tabernacle, NJ, I can on a given day hear FM radio stations from Philadelphia (about 30 miles away), Trenton (also about 30 miles away), Atlantic City (about 35 miles away), as well as some New York City (about 70 miles away) and Lehigh Valley (about 65 miles away) FM stations. These signals would not be considered DX. Any station that I can hear daily would constitute as normal conditions. These are in fact the stations that I am often trying to get around and eliminate when conditions open to hearing DX signals! Unfortunately, that has become tougher and tougher as FCC deregulation has led to a packed FM radio dial in the Delaware Valley (as is the case in much of the U.S.). Thus, the empty spots on the dial to hear would-be DX stations have grown thinner and thinner. The number of people who participate in this crazy hobby (a minuscule number to be sure to begin with) has shrunk in recent years with the state of the FM dial being what it is.

Despite the challenges, I've had a degree of success in DX'ing over the years. My DX'ing "career" kicked off at my previous residence in Florence, NJ in April 2005, thanks to the installation of an APS-13 rooftop antenna (which I still use today). In July 2014, I moved to Tabernacle but the DX'ing escapades continued. In total, I have now logged 4700 unique FM radio stations from 42 states and 12 different countries. In an era where it doesn't seem to be "en vogue" anymore to have a hobby or some sort of side interest that doesn't generate the almighty dollar, I get a lot of joy out of all-night tropospheric enhancement down the Atlantic coast or blowout E-Skip openings to the great plains states during the early summer months. There are worse things I could be doing, for certain!

All the while, I've run into two consistent challenges while participating in the hobby. One, I can never keep my log of stations completely up to date. There are multiple reasons for this (laziness is certainly one!). The software-defined radio (SDR) became popular several years ago, and I purchased one and began using it full-time for my DX endeavors in 2015. The SDR is an absolute blessing. Before, I could only be on one frequency at one time listening for DX. I was limited to the amount of FM radio tuners I had running. At one point, I had 9 different FM tuners recording different channels on the FM band, all connected to various computer inputs to record their audio so I could go back and listen to recordings to see if I had received any DX during times of openings. This was obviously completely inefficient. With an SDR, it can record a whole swath (usually 10 MHz. worth) of the FM band at one time. The only cost is storage space (SDR's record in WAV files and use almost 2 GB of space per minute!). But, with a large external storage drive, this is somewhat manageable. As a result, the SDR became an absolute game-changer in recording DX openings and playing back the results later. That said, I went big (I purchased multiple SDRs, they are relatively inexpensive), and as a result, I have an accumulation of FM audio files from 2016 that I still have not reviewed fully. Thus, my log is always a work in progress.

Additionally, when I have the opportunity to work DX events in real-time, I have been posting my logs to a platform called FMList (fmlist.org). This is a wonderful site that has the FCC

FM database (as well as other countries) built-in so that you can log stations in real-time and they will appear on its associated mapping API. The logging feature works so well that I adopted it to use to keep track of stations I hear when I review past events from the SDR recordings I made. Thus, some of the work has already been done for me in keeping track of the stations I've heard. Still, the sheer number of stations meant that it would be a real drag having to relog everything again in my master log Excel FM spreadsheet I had been keeping. Wouldn't it be great if there was a way I could merge my master FM log with the logs I've made at FMList, yet not count all the duplicate logs?

The second issue I've had with the hobby is not having a consistent place to show off the logs I've had and post about my endeavors on the web. I had a website long, long ago (probably 2005-2008), and it looked like it was straight out of the 90s. Needless to say, I did not maintain it and it went into that dark place in cyberspace so many other fruitless dot com ventures ended up landing. Having a good DX website has been something I've always wanted to aspire to do in my "spare time", but for one reason or another, I've always shelved it for other tasks. With what I've been able to learn at my time in Villanova for coding a decent website (and with the myriad of good templates that exist online now), it seemed like this would be the perfect opportunity to get the tens of thousands of daily clicks a DX page for myself would surely garner (this may be somewhat sarcastic).

Executive Summary

My project has two main parts. The first is a script written in Python that takes two CSV files containing DX logs and merges their contents into one Pandas data frame. A series of duplicate checks are made to ensure that the logs from both CSV files are not duplicated in the new data frame. A new

CSV is generated and saved that contains the updated data frame. Then, using Matplotlib with Seaborn, several graphs are generated that perform rudimentary data analysis on the newly created data frame containing the merged DX logs. The graphs include tallies of the most heard U.S. states, the FM frequencies with the most logs, and the months that have been most bountiful for DX logs based on the content of the data frame.

The second part of the project is a website for myself that is built off of the results of the newly scrubbed data from Python. The website, built to be the online home for my DX hobby endeavors, features a MySQL database containing all of the logs that were newly outputted into the CSV file from the Python script. A page featuring several different SQL-based queries (the Log List page) anchors the site. A second page featuring the graphs that were output from the Python script was also created (Log Graphs). A third page, utilizing Google's 'My Maps' tool, features all of the different locations of the logs I've heard displayed on a Google map. A fourth page displays the contents of an article I wrote for the WTFDA (Worldwide TV & FM DXing Association) monthly VUD publication, analyzing 15 years (2005-2019) of E-Skip activity. I also have a working contact form whose results are sent to a separate table in the site's MySQL database.

Use Case Tour

The Python script could be of potential use to any DX'er (not just myself) and has already been utilized successfully by a DX'ing colleague who was in a similar situation to me (had been keeping an Excel log but was also using FMList and wanted a way to merge the logs from the two sources). The benefits of the FM List Merge Python script are as follows:

- Concatenates two separate CSV files into one data frame
- Removes duplicates from the merged data frame by checking for the following criteria:
    - Entries that contain the same call letters

  - Entries that are on the same frequency and also share the same city of license and state of license (this is particularly pertinent for duplicate entries that have changed format over the years and as such have changed call letters)
  - A CSV file is outputted with the newly concatenated and scrubbed Pandas Dataframe data, giving the user an up to date log as to which stations they have heard and what their actual total amount of logged stations is
  - A PDF output is generated containing the following data analysis in graph form:
    - The most productive frequencies for FM logs, sorted greatest to least
    - A bar graph of the number of FM logs based on several mile ranges:
      - 0 to 300
      - 300 to 500
      - 500 to 800
      - 800 to 1200
      - 1200 to 1450
      - 1450 to 1800
      - 1800 and above
    - The most productive states and provinces for FM logs sorted greatest to least
    - The most productive seasons for FM logs
    - The most productive months of all time for FM logs
    - The most productive calendar dates for FM logs

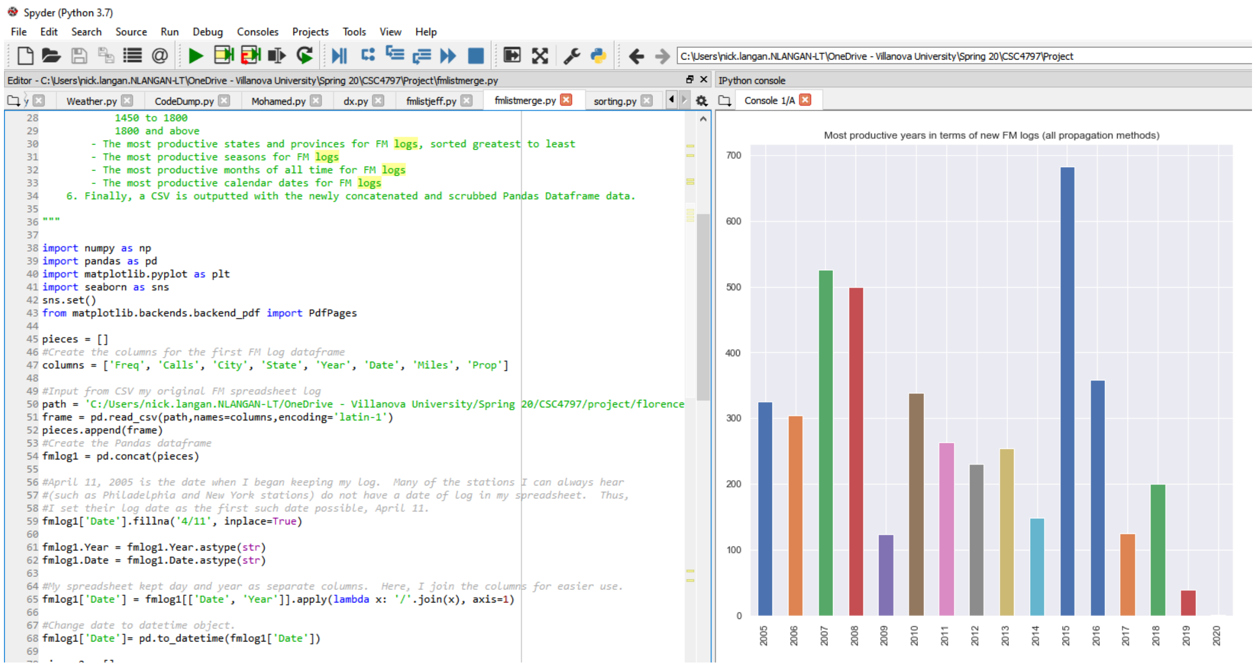
- The website is a display for the output that the Python script helps make possible. Though they do not have strength in numbers, the FM DX community could benefit from having access to a place where I have all of my log data available and displayed in graphical form. I have already gotten the sense (through audio clips I have placed up on YouTube) that such examples of unique FM logs can help draw interest into the hobby and perhaps draw some new participants. I would like to think about the website as a good example of the fun that can still be had by participating in the FM DX hobby. Additionally, I'm hopeful that the results posted on the website can be beneficial to someone in a research capacity (possibly even myself!), able to conclude from over 15 years of dedicated FM monitoring from one particular area. E-Skip, in particular, is a relatively mysterious propagation method, and there is little reliable data in seeing how it behaves from season to season. Though 15 years is a very small sample, I still believe my results could have some benefit. The following pages are part of my website:

- Home page (including a summary about myself and links to the other pages that are part of the website)
- Log List page (provides two drop-down menus to perform queries to the website's MySQL database which contains all 4700 stations I have logged)
- Log Graphs page (the six graphs the Python script generated are posted for data analysis concerning my log)
- Log Map page (the Google 'My Map' of my 4700 logs)
- 15 Years of E-Skip Analysis (an article I wrote analyzing 15 years of E-Skip observation from Burlington County, NJ)
- Link to my latest logs on my FMList account
- A contact page to send me questions or feedback that delivers the results to a table in the MySQL database

Section Detail

Using the Python script

Though any Python compiler will likely be able to run my simplistic script, I recommend running the script in the Spyder IDE, which is where I wrote the script. The script can be run in Spider and its output will appear in its IPython console. Download Spyder at https://www.spyder-ide.org/


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\nick.langan.NLANGAN-LT\OneDrive - Villanova University\Spring 20\CSC4797\Project
Editor - C:\Users\nick.langan.NLANGAN-LT\OneDrive - Villanova University\Spring 20\CSC4797\Project\fmlistmerge.py
Weather.py CodeDump.py Mohamed.py dx.py fmlistjeff.py fmlistmerge.py sorting.py
IPython console
Console 1/A
Most productive years in terms of new FM logs (all propagation methods)
700 600 500 400 300 200 100 0
2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020

```
            1450 to 1800
            1800 and above
        - The most productive states and provinces for FM logs, sorted greatest to least
        - The most productive seasons for FM logs
        - The most productive months of all time for FM logs
        - The most productive calendar dates for FM logs
    6. Finally, a CSV is outputted with the newly concatenated and scrubbed Pandas Dataframe data.

"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from matplotlib.backends.backend_pdf import PdfPages

pieces = []
#Create the columns for the first FM log dataframe
columns = ['Freq', 'Calls', 'City', 'State', 'Year', 'Date', 'Miles', 'Prop']

#Input from CSV my original FM spreadsheet log
path = 'C:/Users/nick.langan.NLANGAN-LT/OneDrive - Villanova University/Spring 20/CSC4797/project/florence
frame = pd.read_csv(path,names=columns,encoding='latin-1')
pieces.append(frame)
#Create the Pandas dataframe
fmlog1 = pd.concat(pieces)

#April 11, 2005 is the date when I began keeping my log.  Many of the stations I can always hear
#(such as Philadelphia and New York stations) do not have a date of log in my spreadsheet.  Thus,
#I set their log date as the first such date possible, April 11.
fmlog1['Date'].fillna('4/11', inplace=True)

fmlog1.Year = fmlog1.Year.astype(str)
fmlog1.Date = fmlog1.Date.astype(str)

#My spreadsheet kept day and year as separate columns.  Here, I join the columns for easier use.
fmlog1['Date'] = fmlog1[['Date', 'Year']].apply(lambda x: '/'.join(x), axis=1)

#Change date to datetime object.
fmlog1['Date']= pd.to_datetime(fmlog1['Date'])

```

Screenshot of working Python script in Spyder

For optimum results, I recommend that the following columns are used in your original "master FM log" (the log that you wish to merge with your FMList log):

- Frequency
- Call Letters (Calls)
- City
- State
- Year of log
- The calendar date of log (i.e. 12/17 for December 17)

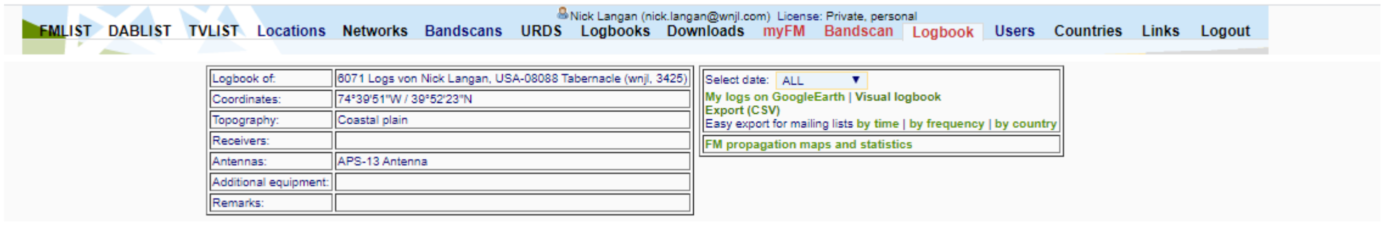
- The distance of log (miles)
- The log's propagation method (Tropo, E-Skip, Meteor Scatter)

Other criteria can potentially be adopted into the script, but these are the columns I have specified in the initial data frame.

The script is designed to work with the essential columns from FMList. To avoid stations being counted multiple times, the script eliminates any portion of entries from FMList that contain "-FM" in the call letters. In my master spreadsheet, I do not note whether the station has "-FM" in its callsign or not (I only track the actual callsign, such as WKXW). A user who does track "-FM" in their callsign may want to note this accordingly. The script also eliminates any portion of entries that contain "-HD" in the call letters (HD stations are typically not noted in FM logs). Finally, FMList tends to at times add a space and add extra content into the callsign of a logged station. This again could cause a station to be counted twice when matched with the original FM log. As such, I have removed any portion of entries from FMList that have an extra space after the callsign.

Exporting a CSV of your logs out of FMList

The number one use case for the Python script is to merge your FMList log database with an existing spreadsheet containing your FM logs. To do this, you need to successfully extract your FM logs out of the FMList platform, which can be tricky. Log into your FMList account at www.fmlist.org. Browse to your 'Logbook' section. Ensure that the date selected to display is 'ALL'. Then, choose 'Export (CSV)'.

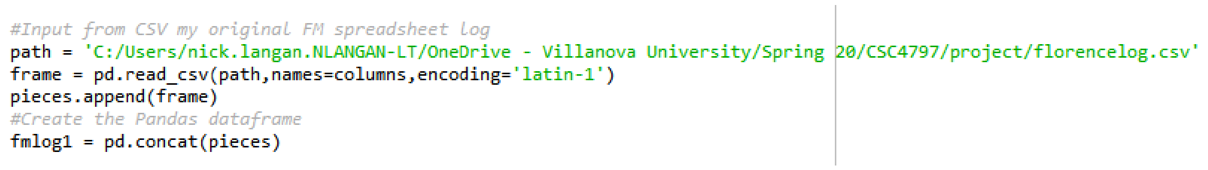

Nick Langan (nick.langan@wnjl.com) License: Private, personal
FMLIST DABLIST TVLIST Locations Networks Bandscans URDS Logbooks Downloads myFM Bandscan Logbook Users Countries Links Logout

Logbook of:	6071 Logs von Nick Langan, USA-08088 Tabernacle (wnjl, 3425)
Coordinates:	74°39'51"W / 39°52'23"N
Topography:	Coastal plain
Receivers:	
Antennas:	APS-13 Antenna
Additional equipment:	
Remarks:	

Select date: ALL

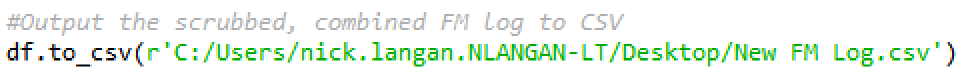
My logs on GoogleEarth | Visual logbook
Export (CSV)
Easy export for mailing lists by time | by frequency | by country
FM propagation maps and statistics

Unfortunately, the FMList exported CSV file itself needs some minor scrubbing of its own. Launch Microsoft Excel and open up the CSV file you just exported, and ensure that any text that is placed at the top of the CSV file (above the columns) is removed. This will interfere with the Python script. It is recommended that you change the separator in the CSV file from a semicolon to a comma (again, this seems to work better with the Python script). Finally, remove any column headers from the FMList CSV file. Save the file and make sure to take note of the locations where the FMList CSV file and your original CSV file are located, as the Python script will search a specified location for each:



```
#Input from CSV my original FM spreadsheet log
path = 'C:/Users/nick.langan.NLANGAN-LT/OneDrive - Villanova University/Spring 20/CSC4797/project/florencelog.csv'
frame = pd.read_csv(path,names=columns,encoding='latin-1')
pieces.append(frame)
#Create the Pandas dataframe
fmlog1 = pd.concat(pieces)
```

The same care should be given to the file output location for the CSV file that the Python script generates.



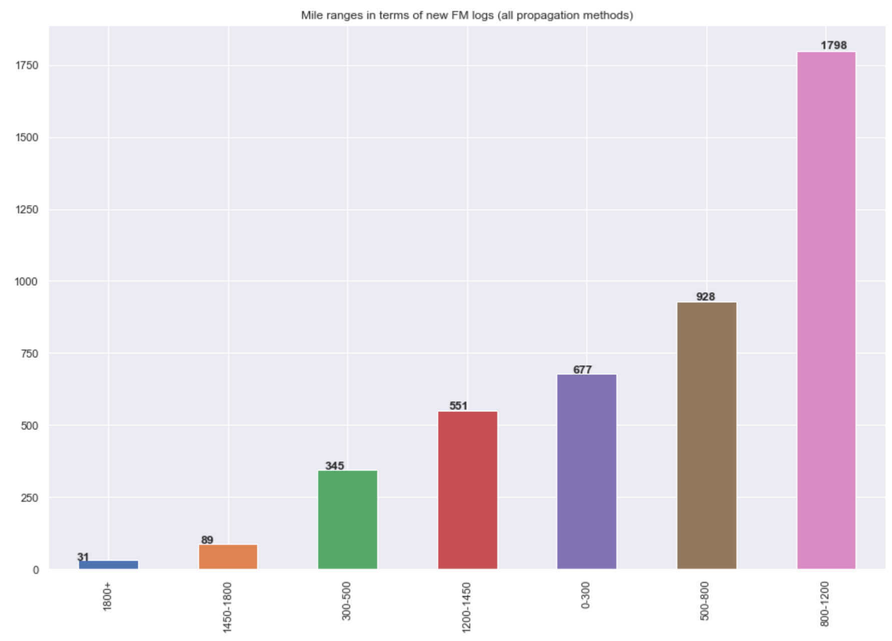
```
#Output the scrubbed, combined FM log to CSV
df.to_csv(r'C:/Users/nick.langan.NLANGAN-LT/Desktop/New FM Log.csv')
```

Note that the PDF file the Python script outputs of its Matplotlib data analysis will be stored in whatever location is selected as default for your Spyder IDE.

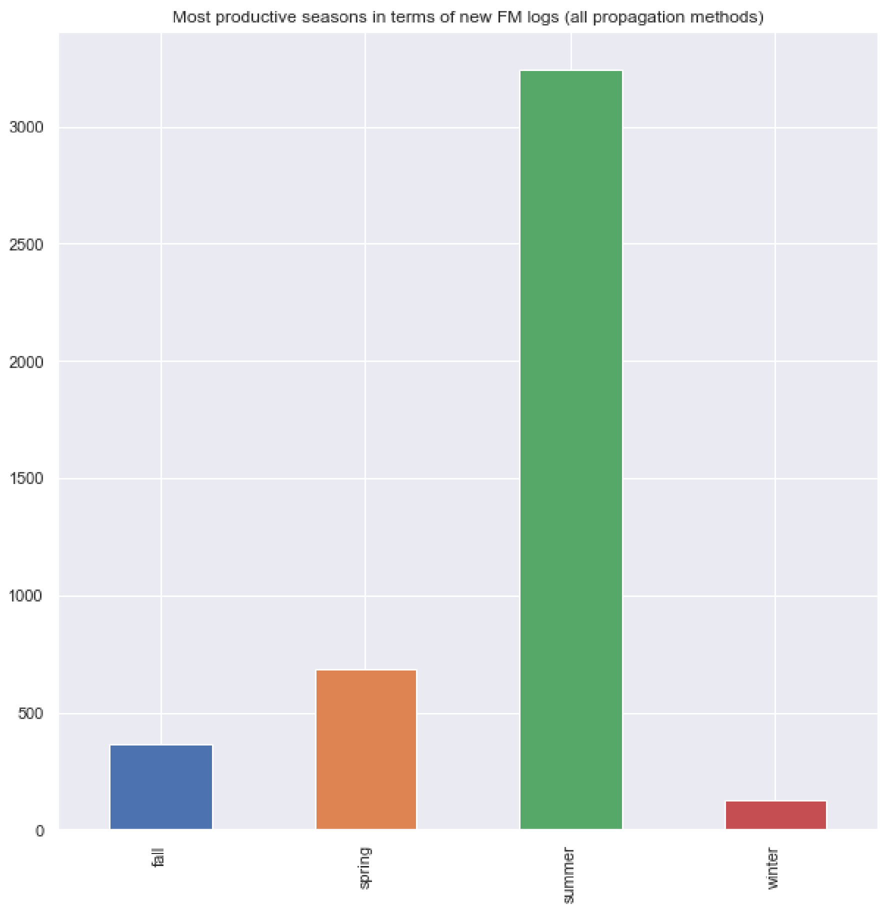
Understanding the Python output

The CSV file that the Python script outputs is fairly self-explanatory. It features eight columns, including call letters, date of reception, propagation method, country of reception, city of reception, state of reception, and its distance in miles. The data should be sorted by frequency, from lowest (88.1) to highest (107.9).

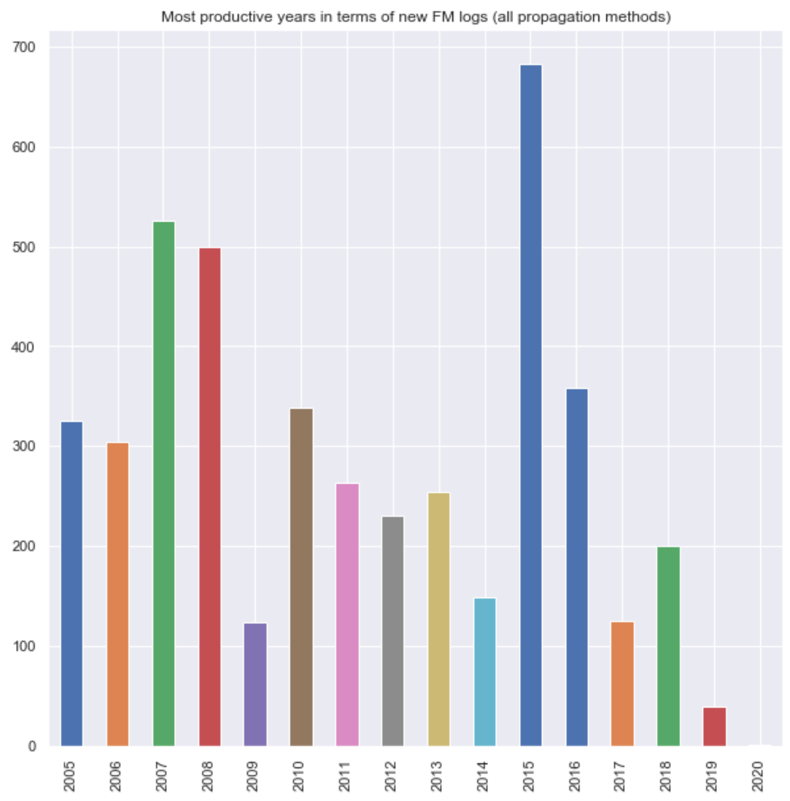
The PDF file that the Python script outputs features six graphs generated by Matplotlib. For an understanding of each graph:



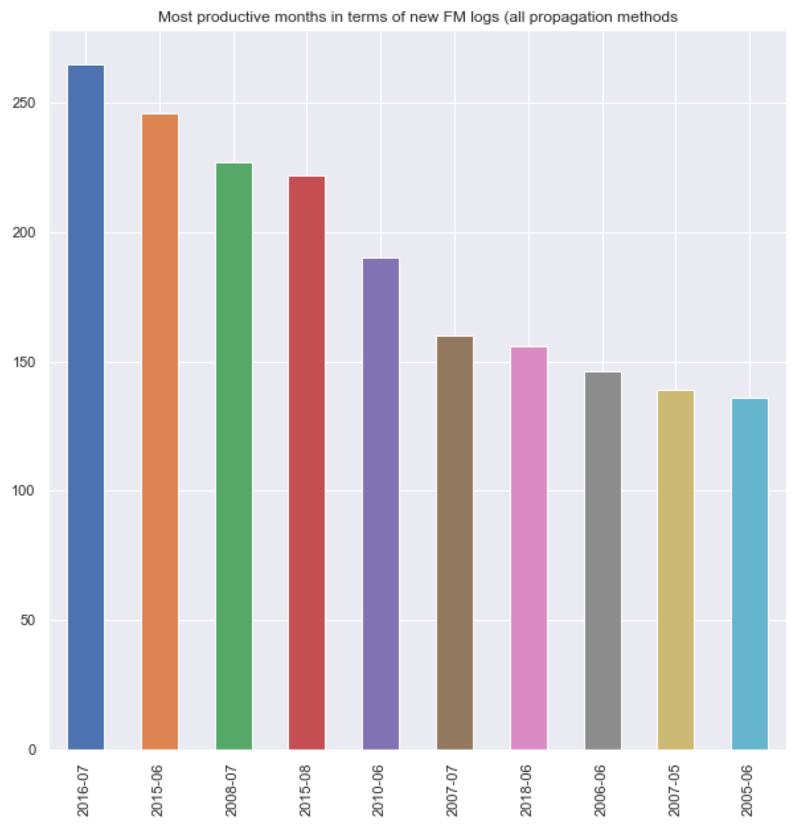
The Python script placed all logs into seven different categories based on the distance of the log. The bar graph helps illustrate which range of mileage is most prevalent amongst your FM logs.



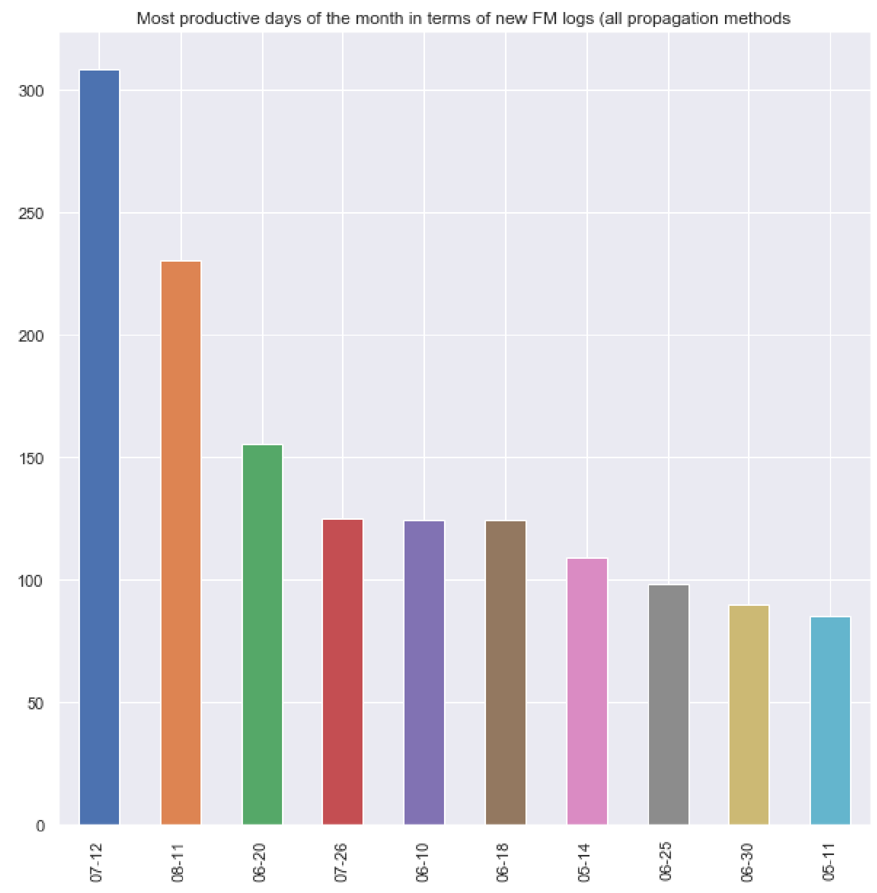
The Python script also shows the most prevalent seasons by which your DX logs have occurred. Note, meteorological seasons are tracked here (December to February, March to May, and June to August).



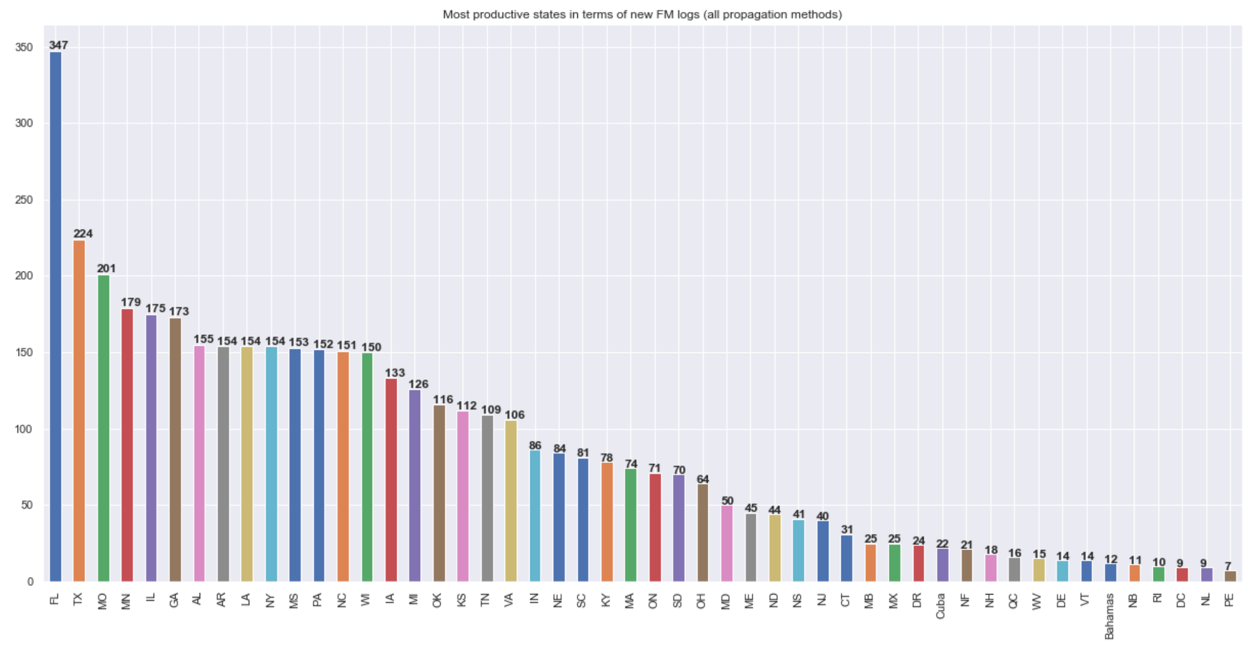
The script outputs a graph tallying the counts of logs by year (from whenever the log initially commenced through present-day).



The script also outputs a bar graph showing which months have been most bountiful, throughout the log, for garnering the most FM logs (numeric values are used, such that 2016-07 refers to July 2016).

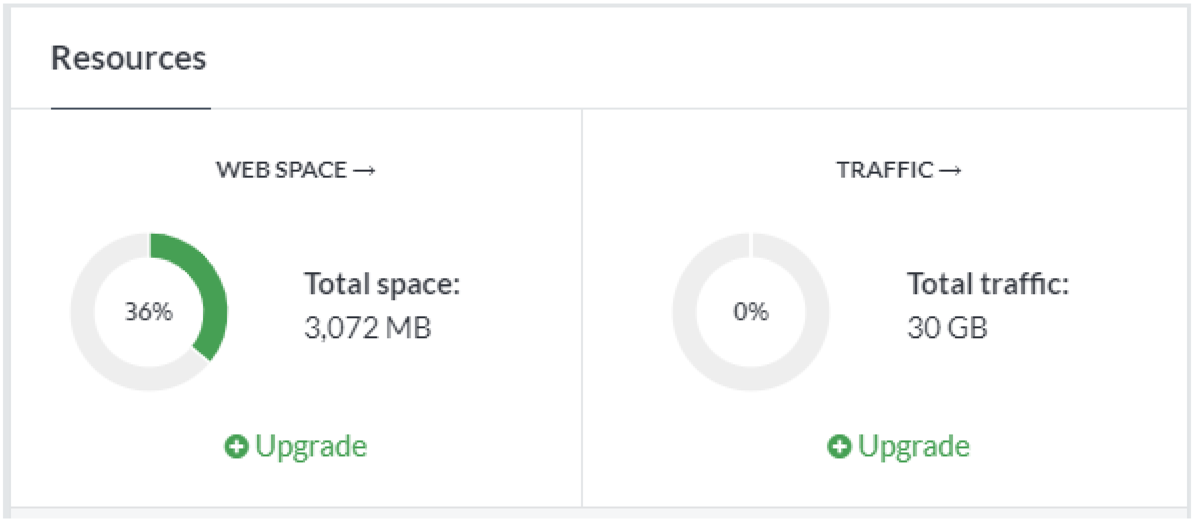


Longtime DX'ers have noted certain calendar dates, for whatever reason, seem to favor certain propagation activity (particularly E-Skip). The script outputs a graph showing which calendar days (in numeric form) have been most bountiful for FM logs. Finally, an output of the most logged state and political units is shown below.



Using the website

Detail explaining how the website was built will be given in the Detail Description. From a user perspective, the website is designed to be fairly simple to access. The website is hosted at Altervista, a free website host based out of Italy. The URL to the website is: http://wnjl.altervista.org/



A look at the site's resources at Altervista

All logs available from the drop-down menu in the "Log List" page are generated from the website's MySQL database, which features the contents of the CSV file generated from the Python script. The CSV file was imported into the database via its PHPMyAdmin control panel. More detail on how to import the database is given in the Detail Description section.

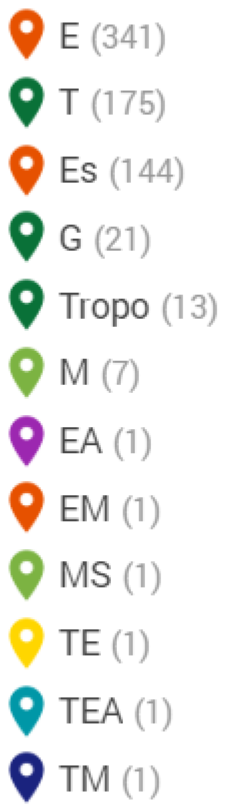
The contact form is linked to the bottom of each of the site's webpages to make it easy to reach out to yours truly. All results from the contact form are stored in the "contactform" table in the site's MySQL database.

Using the Google 'My Map'

The Google My Map displaying all 4700 of my FM logs on a Google map was generated in Google's My Map tool and is shared publicly so that any user can access the map (not just with a Google account). This is the "crown jewel" of the website. Note, because of the sheer amount of icons shown on the

Google map, it can take some time to load, depending on the specifications of the system viewing the map. Details on how the map was compiled can also be found in the Detail Description section.

Each of the icons on the Google map reflects the locations of logged stations. Those icons are color-coded to represent the propagation method of the log. Those are the following:



Red map icon/marker: E-Skip

Green map icon/marker: Tropo or groundwave propagation

Light green map icon/marker: Meteor scatter

Yellow map icon/marker: Station has been logged by both E-Skip and tropo

Dark purple map icon/marker: Station has been logged by both tropo and meteor scatter

Light blue map icon/marker: Station has been logged by both E-Skip, tropo, and meteor scatter (the 'ol "hat trick")

Purple map icon/marker: Station has been logged by both E-Skip and auroral scatter

Teal map icon/marker: Station has been logged by both E-Skip, tropo, and auroral scatter

Audio clips on the website

I have over 2000 recorded audio clips of the FM stations I have heard in my time DX'ing (note, since the adoption of an SDR, I have switched to producing YouTube captures of SDR logs). An important part of the website would be to feature these audio clips amongst the "Log List" SQL queries so that a user could hear a clip of a station, if available. Links are generated to applicable clips only if one exists for the station in question in the website's "audio" folder. More detail on the PHP code that allows the URL to be generated for a station's audio clip if it has one available can be found in the Detail Description section.

Most audio clips are in MP3 format and should play right in the user's browser, except for those hyperlinks which are PHP file redirects to the applicable YouTube clip for the particular station and reception.

Project Documentation

EDP

Name of Project:__Nick Langan's Florence & Tabernacle, NJ DX Website____

Leader: _____________Nick Langan____

Phone:___________(216)410-0513____

E-mail:___nlanga01@villanova.edu____

Type of project (Software development, Hardware Installation, Integration, Other) Other (Website/Database)____________

Stake Holders: (Here please fill out who will receive the deliverables) Prof. Mark Anderson, WTFDA (The DX Association I am a member of)

Estimated Start Date: (When work, including EDP begins) February 17, 2020

Estimated End Date: (When all requirements are delivered) May 5, 2020

Project Requirements:

This website must be linked to a MySQL database containing each of the 4700 FM stations I have logged as part of my FM DXing hobby from Tabernacle, NJ, and Florence, NJ. Using Google's Mapping API, I will have charts and graphs displaying the stations logged based on state, distance, and year. PHP will be used on the website, and Python will be used to analyze my FM log data before importing it into the MySQL database.

Project development technique stated:

Will the project use a waterfall methodology? - No

Will the project use a spiral method? - Yes

Will the project use an agile approach? - Yes

Will you use “sprints”?   Ideally, yes

Risk Analysis (STATE HERE: What could de-rail the project and how will you mitigate these risks)

Struggle to adopt or grasp Google’s Graph API (a backup plan would be to generate all graphics through Python, which I have some experience with). **Note (5/1): Graphs were generated in Python, but the Google My Maps feature worked well for mapping the logs**

Designing/locating a proper CSS template for the website (there should be many possibilities for this).

My master FM DX log is not currently up to date (currently in an Excel spreadsheet format). Time must be taken during the study phase to make sure this is up to date.

Update (5/1): A newfound use for the Python script was realized to merge the FMList CSV file into the existing master FM DX log, which ensured that my log was in fact up to date.

Intermediate Milestones so that we know we are on track for the final delivery date. Each Milestone needs a deliverable, either report or prototype. See the syllabus for the required deliverables and estimated dates.

Project Timeline

PROJECT TASK	ESTIMATED START	ESTIMATED END	RESPONSIBLE
Update FM DX log spreadsheet	1/27/20	2/27/20	Nick Langan
Locate website CSS template	2/4/20	2/11/20	Nick Langan
Run Python analysis on FM DX data	2/28/20	3/10/20	Nick Langan
Explore Google Graphics API	2/11/20	3/25/20	Nick Langan
Website CSS/HTML coding	3/10/20	4/10/20	Nick Langan
MySQL database deployment	3/15/20	4/15/20	Nick Langan
PHP/Javascript link to MySQL database	4/1/20	4/20/20	Nick Langan
Website completion	4/20/20	5/1/20	Nick Langan

Intermediate Milestones

2/11/20 – Sufficient CSS template located

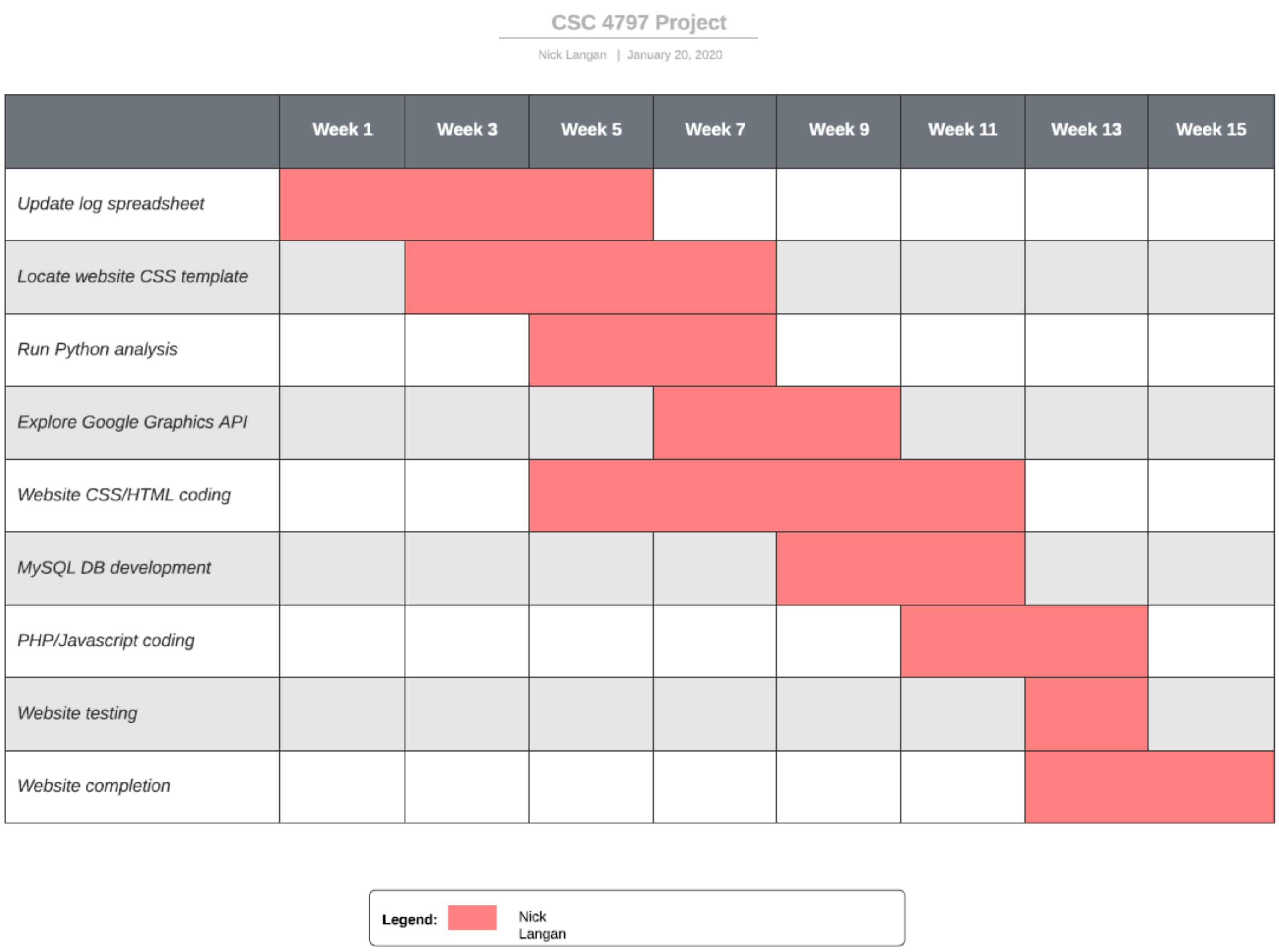
2/27/20 – FM DX log database is up to date

3/25/20 – All Google API graphics are generated

4/5/20 – A working website with everything but MySQL/DB queries

4/20/20 – MySQL DB tested and working

Gantt Chart



CSC 4797 Project

Nick Langan | January 20, 2020

	Week 1	Week 3	Week 5	Week 7	Week 9	Week 11	Week 13	Week 15
Update log spreadsheet	🟥	🟥	🟥					
Locate website CSS template		🟥	🟥	🟥				
Run Python analysis			🟥	🟥				
Explore Google Graphics API				🟥	🟥			
Website CSS/HTML coding			🟥	🟥	🟥	🟥		
MySQL DB development					🟥	🟥		
PHP/Javascript coding						🟥	🟥	
Website testing							🟥	
Website completion							🟥	🟥

Legend: 🟥 Nick Langan

Cost Document

Hourly rate: $100.00/hour for website development and Python coding

Estimated man-hours

Phase 1 – Gathering requirements or "user stories", drafting concept and initial design

6 hours

No hardware needed

Software utilized: Microsoft Excel, Microsoft Word (previously owned)

Phase 2 – Study phase (analyzing existing FM logs, determining if they should be updated manually, beginning initial website plan, risk mitigation)

12 hours

No hardware needed

Software utilized: Microsoft Excel, Microsoft Word, Radio-Locator.com (Free, but has a daily search limit which will impact progress)

Phase 3 – Project design

18 hours

No hardware needed

Software utilized: Microsoft Excel, Microsoft Word, Lucidchart, Microsoft DevOps for task planning

Phase 4 – Implementation

50 hours

No hardware needed

Software utilized: Spyder for Python, Microsoft Excel, DevOps, Google MyMaps, Google Sheets

Requirements:

- Index page with the appropriate template (4 hours)
- Logs page that can be queried by type (10 hours)
- Working MySQL database for logs (10 hours)
- Google Maps API to display logs (10 hours)
- Clips page with embedded YouTube vids (10 hours)
- Contact page to contact me with form (6 hours)

Phase 5 - Delivery

Prepare documentation and presentation

15 hours

Total hours: 101 hours

Total cost: USD 10,100 payable to Nick Langan (in my dreams!)

Design Description

Index page with an appropriate template

The CSS template I utilized for the design can be found here: https://templated.co/horizons

Logs page that can be queried by type

The logs page should have a drop-down menu where visitors can view all my 4700 DX logs in a table format. Users should be able to sort the logs by date, by propagation method, by frequency, and by distance. This logs page should be directly linked to query the MySQL database via PHP. This page should use the template found for the index page and be linked at the top of the website.

Working MySQL database for logs

Import updated master FM log spreadsheet into MySQL database table. The database connection must be made from the PHP webpages as part of the website. Xampp will be used for initial testing and then the database will be exported to the permanent web host.

Google Maps API to display logs

From the logs page, a link to the Google "My Maps" page where all my FM DX logs are overlayed will be available. The spreadsheet with all logs must be imported into Google's API and testing must be accomplished to ensure all the data is plotted properly (most importantly, the city of license of the FM radio stations I've logged so they appear correctly on the map).

Contact page to contact me with form

Working contact form where all resorts are stored in a separate "contactform" table in the MySQL database. Linked at the top of the website.

Log Graphs page with data analysis of logs

PNG output of graphs generated with Python script that are displayed as a supplement to the logs available on the website.

15 years of E-Skip data page

Features an article I wrote for the WTFDA September 2019 VUD, analyzing 15 years of E-Skip observations from my locations in Burlington County, NJ.

Detail Description

Writing the Python script

I credit any knowledge I have of Python from my CSC 5930 'Computing for Data Science' special topics class I took with Dr. Cassel in the Spring of 2019. My Python experience helped produce powerful data visualization for a climate/meteorology project I presented at the end of the semester. The trial and error experience I gained putting that project together made me want to continue to sharpen my Python skills. When looking at the challenge of both updating my master log and importing data from my FMList log into my master log, it dawned on me that Python was the perfect tool to perform this with.

Algorithm used

The very simple algorithm that was relied upon to guide the composition of the FM List Merge Python script was:

```
Read Main Log CSV file – Create dataframe

Read FMList CSV file – Create dataframe

Concatenate the two dataframes

If there are entries containing the same call letters

Keep only the first entry (by date)

If there are entries containing the same frequency, state, and city of license

Keep only the first entry (by date)

Sort entries by frequency

Output to CSV
```

Techniques of note applied in the Python script

April 11, 2005 is the date when I began keeping my log. Many of the stations I can always hear (such as Philadelphia and New York stations) do not have a date of log in my spreadsheet. Thus, I set their log date as the first such date possible, April 11.

The FMList spreadsheet needed some extra scrubbing. First, I removed any part of the call letters that contain the character r: and anything after that character sequence. Then, I changed the data in the date column to have a '-' for the date signifier (i.e. 10-17-05) instead of a '.' (i.e. 10.17.05). I also removed any extraneous information from the call letter fields (such as '-FM', -'HD', or anything that came after a space).

Using Matplotlib

In my initial draft of my design, I had envisioned producing graphs for my project with the Google Graphing API. I performed some tests with it, and it had promise, but there was a wealth of nuances that had to be understood to take full advantage of it. As I did not see a major advantage using Google Graphs vs. Python's Matplotlib output, I decided to focus my efforts on producing the graphs with Python.

Annotating graphs generated with Matplotlib is not as simplistic as it may seem. Particularly for the graphs showing the count for states and frequencies logged, being able to see the exact count for each state or frequency at the top of the bar for each entry is important for proper context. "Eyeballing" graphs never ends well. A for loop is needed to produce annotation on a given graph axis. This Stack Overflow discussion, in particular, was most helpful for specifying the dimensions to produce annotations on a graph: https://stackoverflow.com/questions/35972380/annotate-axis-with-text-in-matplotlib

The PdfPages package for matplotlib was used to produce the PDF output that displays the multiple graphs generated by the script. This technique was followed to produce multiple graphs into one PDF file: https://stackoverflow.com/questions/21364405/saving-plots-to-pdf-files-using-matplotlib

The Seaborn data visualization library is used with Matplotlib. In my experience, this generates much more professional looking graphs (similar to GGPlot outputs in the R language).

Designing the website

Initially, several avenues were considered for building the website, including Node.js and Wordpress, but a CSS template, of which I've had prior experience (including in a website project for my CSC 2053 class in Fall 2019) was selected which seemed appropriate for the venture. Often, in major E-Skip openings, stations from the heartland of America are received, so it seemed only appropriate a typical farm/plains scene would the backdrop to the website, as is shown in the theme I selected.

The website was initially deployed locally in a Xampp Apache environment. This made testing simple. Xampp also has a full-fledged MySQL database of which supports PHPMyAdmin. All website editing was done in Visual Studio Code.

Upon successful tests on Xampp, all website code and the MySQL database were successfully migrated to Altervista. A brief tutorial on importing data into the Altervista MySQL database will be given below.

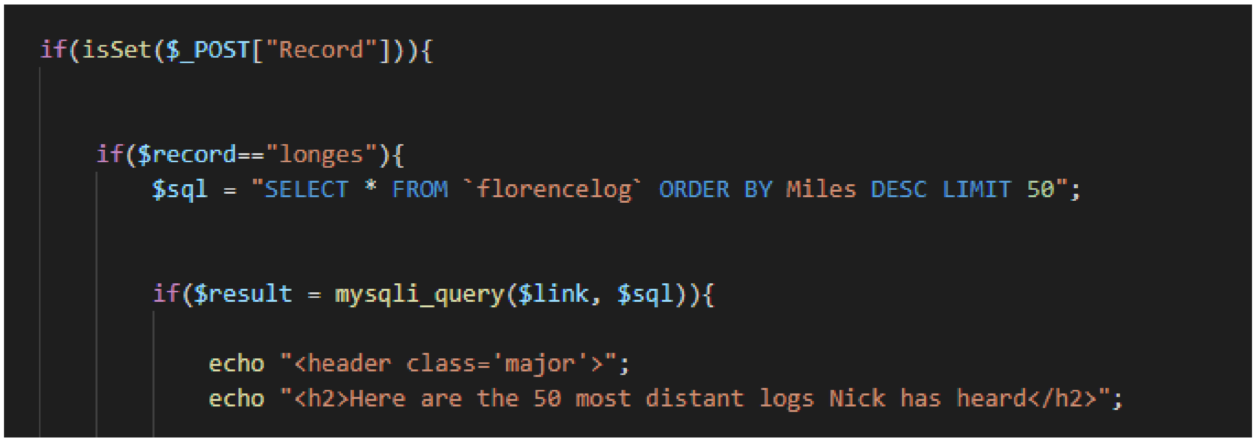
The driving force for the website was PHP code. I wrestled with the tech stack choice. I could use more experience in Javascript, and maybe the website would have been a good experience for me to get more comfortable with Javascript. Javascript seems to be the more en vogue language while PHP in some circles is seen as dead. Still, my experience in CSC 2053 made me learn toward PHP, as it is, for better or worse, easy to learn and versatile for connecting with a SQL database.

Building the index page

There is nothing elaborate about the home page. An appropriate title for the page was given, and a picture of myself was provided that I am hopeful should not drive traffic numbers into negative values. Finally, two appropriate YouTube videos for the page are embedded to show users examples of DX catches I have made.

Building the 'Log List' page

Two drop-down menus with several options are given to query the MySQL database. When a user selects an option, such as 'Top 50 Longest FM logs', they are redirected to 'logquerynew.php'. Making a connection to the MySQL database is conducted through a PHP include to 'database.php'. Variables are set in logquerynew.php based on the 'POST' method provided from the form information on the prior page. If the user selected 'Top 50 Longest FM logs', first the POST method is set to 'Record'. The type of Record is 'longes'. Upon successful connection to the database, 50 records are queried in descending order via the following code:

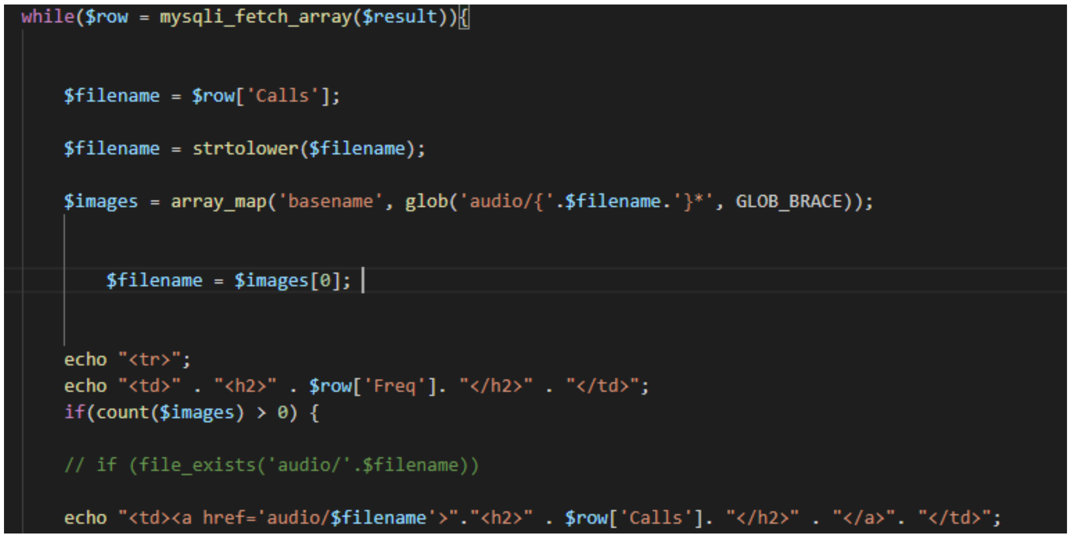


```
if(isSet($_POST["Record"])){


    if($record=="longes"){
        $sql = "SELECT * FROM `florencelog` ORDER BY Miles DESC LIMIT 50";


        if($result = mysqli_query($link, $sql)){

            echo "<header class='major'>";
            echo "<h2>Here are the 50 most distant logs Nick has heard</h2>";
```

The table is then provided to the user via a while loop (if there are remaining results).

An important part of the log list output is the ability for a user to listen to an audio clip of one of the stations in the table if there is an available audio clip. I wanted to find a way to generate a hyperlink to an applicable audio clip, but ONLY if an audio clip existed. If there was no applicable audio click, I did not want a hyperlink to be generated.

This could be done in theory because all of the filenames for the audio clips I have made (in MP3 format) include the call letters of the station received. So, if I could I find a way to do a check by call letters to see if a filename containing it appeared in the website 'Audio' folder, then I could place a statement in the PHP code to generate a hyperlink for the given entry in the while loop.



```
while($row = mysqli_fetch_array($result)){


    $filename = $row['Calls'];

    $filename = strtolower($filename);

    $images = array_map('basename', glob('audio/{'.$filename.'}*', GLOB_BRACE));


        $filename = $images[0];


    echo "<tr>";
    echo "<td>" . "<h2>" . $row['Freq']. "</h2>" . "</td>";
    if(count($images) > 0) {

    // if (file_exists('audio/'.$filename))

    echo "<td><a href='audio/$filename'>"."<h2>" . $row['Calls']. "</h2>" . "</a>". "</td>";
```

The challenge I ran into was the name of the audio files I had in the 'Audio' folder of my DX audio clips. Some (particularly early on in my DX career) simply contained the call letters of the station, i.e. 'wqcd.mp3'. In that case, all I had to do was change the $filename variable to lower case, and then I would be able to match if such a file existed. However, as I went on in my career, I began inserting the date and time into the filename for a DX catch. So, if I logged WROR on August 10 at 8:30 PM, the filename would be 'wror_0810_2030.mp3'. The initial file search PHP function I had would skip that entry, as it was not an exact match to 'wror'. However, the PHP 'glob' function solved this issue, as it can be told to simply look for a partial match to a filename. After some troubleshooting, I was able to get 'glob' to work properly.

Thus, any query on the 'Log List' page will provide working hyperlinks to an audio clip for all of the stations generated, provided there is an associated audio clip.

Building the 'Log Graphs' page

The heavy lifting for this page was done in the Python script. I simply saved the image outputs in Spyder as PNG files, uploaded them to the webserver, and then inserted them appropriately in the HTML code. Some adjustments in Spyder had to be made to ensure the images were sized correctly for the page. Note, this is not perfect in the mobile version of the website. However, I feel it is unlikely I will be getting many mobile viewers.

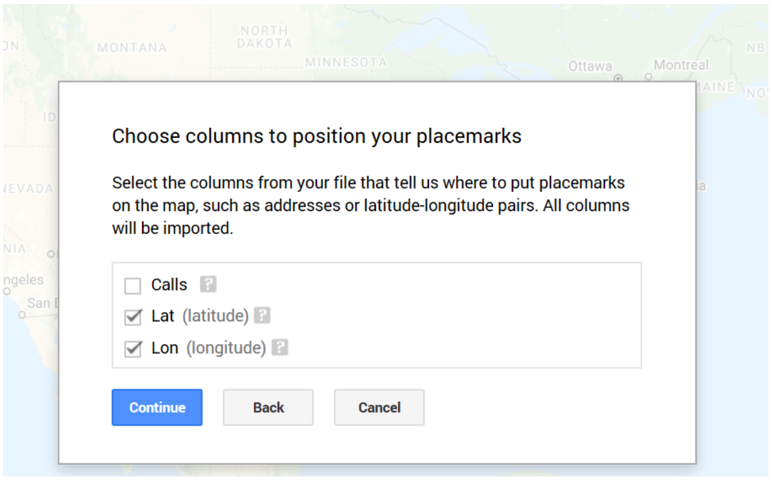
Building the Contact form page

A template was relied upon from my CSC 2053 website project from Fall 2019. I was able to successfully create a separate 'contactform' table in the MySQL database, and all submissions from this form will be sent to the database.

Building the Google ‘My Maps’

The Google ‘My Maps’ interface is surprisingly easy to use (the more data Google can collect, the better after all!) and has no obvious limitations as far as data usage. Creating a map in ‘My Maps’ can be done at https://www.google.com/maps/about/mymaps/

The key to managing the My Map is creating layers. In a layer is where you can import data from a CSV file and use appropriate location data for each entry in the CSV file to plot locations on a map. When you select a file to import into your layer, you will be immediately asked to choose the columns that tell Google where to put placemarks on the map.



Choose columns to position your placemarks

Select the columns from your file that tell us where to put placemarks on the map, such as addresses or latitude-longitude pairs. All columns will be imported.

- ☐ Calls
- ☑ Lat (latitude)
- ☑ Lon (longitude)

After making this choice, every row in your CSV file will be plotted with a location on the map. Clicking on an icon will provide the information each row’s column has in the spreadsheet. MyMaps provides the option to use another column as a label for each icon. In my case, I used the call letters as the label.

The major limitation I ran into with My Maps is that it only allows 2000 rows PER layer. With 4700 logs, this certainly would be a factor. Thus, I have three different layers and had to break my CSV file up into three different iterations. This is inconvenient but I see no other way around it.

MyMaps can be private or shared publicly. In this case, I have it publicly shared.

Building the MySQL database

Importing data into Altervista's MySQL database is no different from Xampp. Conveniently, Altervista also supports PHPMyAdmin. After creating a database and a table, the first step is that appropriate columns must be set up in the table that corresponds to the CSV file you will be importing. Then using the Import function, you can select your CSV file and import its contents into the database. Your columns must correspond to the type of columns you set in PHPMyAdmin. For example, if you select 'date' as the type for the 'Date' column, all dates must be entered in YYYY-MM-DD format. I had to edit my CSV file first to support this. Additionally, if you have set your 'Miles' column as an integer, and then have data such as a '?' in one of the rows, the import process will error.

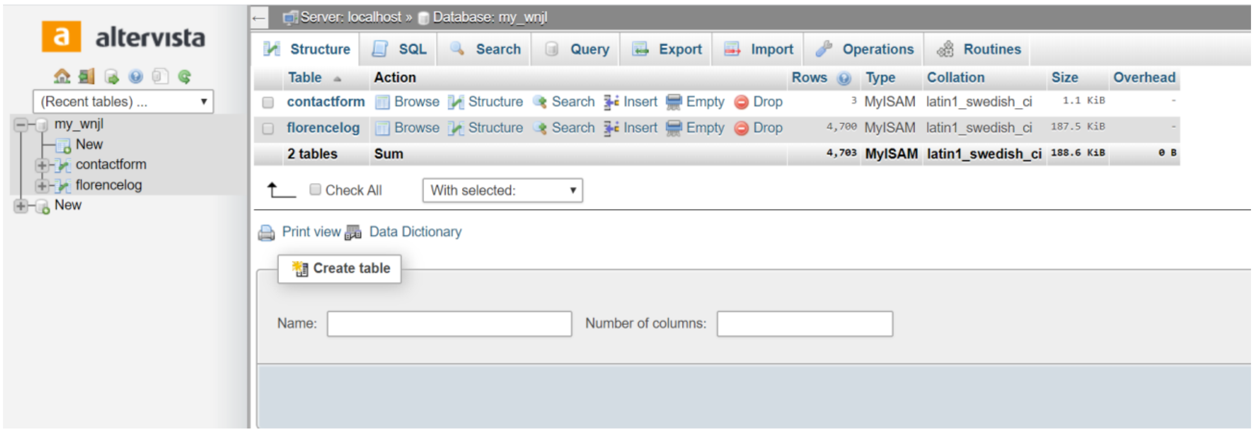


Table	Action	Rows	Type	Collation	Size	Overhead
contactform	Browse Structure Search Insert Empty Drop	3	MyISAM	latin1_swedish_ci	1.1 KiB	-
florencelog	Browse Structure Search Insert Empty Drop	4,700	MyISAM	latin1_swedish_ci	187.5 KiB	-
2 tables	Sum	4,703	MyISAM	latin1_swedish_ci	188.6 KiB	0 B

Overall the database performance in PHPMyAdmin seems quite responsive and I am pleased with it.

Testing

Requirement 1: Working index page with an appropriate template

All links and images on the index page are working properly. The template loads without any errors. The display is sufficiently vibrant on different browsers (Chrome, Firefox, Microsoft Edge)

Requirement 2: Logs page that can be queried by type

All SQL queries on the logs page are provided in the drop-down menu are functional. All states and provinces in the 'View log by state/province' section have been tested. Output was scrutinized and ensured that the appropriate choice is reflected in the SQL output. The hyperlinks that are generated for audio clips have been tested and verified as functional. The MySQL connection has been seamless at Altavista.

Requirement 3: Google Maps API to display the logs

The Google Maps 'My API' was shaped to work with all 4700 logs and are displayed by their location. It is accessible both logged into my Google account and logged out, to ensure that it is, in fact, a public map and not private.

Requirement 4: Contact page with contact form

The contact form was tested with multiple queries and multiple forms of input (including invalid input). It handles input as requested and stores entries successfully in the 'contactform' MySQL table.

Requirement 5: Log Graphs page with data analysis of logs

All Python generated PNG files are displaying correctly on the logs page and in an appropriate resolution.

Requirement 6: 15 years of E-Skip data page

I procured the column I wrote in the WTFDA VUD and successfully transcribed it in Visual Studio Code. The article appears to be formatted correctly on multiple browsers.