State Tourism

Final Report

Aditya Agarwal, Sparsh Bansal

CS 4624: Multimedia, Hypertext, and Information Access

Virginia Tech

Blacksburg, VA 24061

December 15, 2021

Instructor: Dr. Edward A. Fox

Client: Dr. Florian Zach

Table of Contents

Abstract - Executive Summary 5
1. Introduction 6
1.1 Teams Roles 7
2. Requirements 8
2.1 Data Extraction 8
2.2 Data Visualizations 8
3. Design 9
3.1 Extractor 9
3.2 Keyword JSON File 9
3.3 Processing JSON data 9
3.4 Data Visualization 10
4. Implementation 11
4.1 Coding Environment 11
4.2 Language + Tools 11
4.3 Prerequisites 11
4.4 Loading All Data + Config File 16
4.5 Extracting Meta Tags 16
4.6 Amazon Alexa API 16
4.7 Visualization 18
5. User's Manual 19
6. Developer's Manual 20
6.1 Library Installation 20
6.2 Loading files, Extraction, and CSV output 20
6.3 Handling Visualization 21
7. Methodology 22
7.1 Goals of our Users 22
7.2 Subtasks of our goals 22
7.3 Workflows 22
8. Results 23
9. Scripts 29
10. Lessons Learned 37
10.1 Timeline/schedule 37
10.2 Problems 37
10.3 Solutions 38
10.4 Future Work 39
11. Acknowledgements 40
12. References 41

Table of Figures

Figure 1: Format of the created JSON file 9
Figure 2: Anaconda website 11
Figure 3: Anaconda Setup Wizard 12
Figure 4: Execution of pip install dash 13
Figure 5: Execution of pip install dash 14
Figure 6: Execution of pip install pandas 14
Figure 7: Execution of pip install requests 15
Figure 8: Execution of pip install BeautifulSoup4 15
Figure 9: Prices of Alexa Web API 17
Figure 10: Our API implementation 18
Figure 11: Command to run python scripts 19
Figure 12: Developer points in the extraction script 20
Figure 13: Developer points in the visualization script 21
Figure 14: Code output by running the Colorado's JSON file .. 23
Figure 15: Code output by running the California's JSON file .. 25
Figure 16: Code output by running the Virginia's JSON file .. 27
Figure 17: Script 1: script_1_extract.py 29
Figure 18: Script 2: script_2_data_processing.py 31
Figure 19: Script 3: script_3_plotly_script.py 33
Figure 20: Script 4: test 1.py 36

Table of Tables

Table 1: Graph of Colorado 26
Table 2: Graph of California 28
Table 3: Graph of Virginia 30

Abstract - Executive Summary:

The project is about analyzing and visualizing metadata of tourism websites of three states (Virginia, Colorado, and California) from 1998 to 2018.

Each state in the United States has its own state website that is used as a resource to attract new tourists to this location. Each of these sites usually includes great attractions in this state, travel tips and facts about this place, blog posts, and reviews from other people who have there. Suggestions regarding what might attract potential customers could emerge from examining past tourism websites and looking for any patterns amongst them that would determine what worked and what didn't. These patterns can then be used to determine what was successful and use that information to make better informed decisions on the future of state tourism.

We will use the historical analysis of past government tourism websites to further support research on content and traffic trends on these websites. The various iterations of each state's tourism website are saved as snapshots in the Internet Archive. Our team was given the Parquet files having the snapshots of data containing the information recording tourism for California, Colorado, and Virginia dating back to 1998.

We used a combination of Python's Pandas library and Beautiful Soup to examine and extract relevant pieces of data from the given Parquet files. This data was scraped to extract the meta tags used for the website as of that date. With this data we plotted the presence of all the various on a state's tourism website in a chronological order. This made it possible for us to analyze the addition and removal of keywords and to see other changes that were made like using phrases, capitalizations, keywords in languages other than English, and updating of keywords based on internet trends. This led us to conclude that meta tags play a very important role in a website's search engine ranking and a lot of analysis needs to be done keeping in mind the primary user base of the website.

1. Introduction

Our goal for this project was to extract and analyze data from past and present iterations of the California, Colorado and Virginia state tourism websites -- www.visitcalifornia.com, www.colorado.com, and www.virginia.org, respectively -- so that researchers are able to learn from that data. Our client was Dr. Florian Zach of the Howard Feiertag Department of Hospitality and Tourism.

Dr. Zach asked us to produce a system that is able to extract as much raw information as possible from the previous iterations of the state website -- such as meta tags -- and store them in a location where he can then use that information to look for patterns in the data that would lead to more informed decisions regarding the construction of future iterations of the state website. We also were to create a visualization model that can be presented to the class. We chose to focus on the keywords that are present within each iteration of the website to determine the keywords used in generating website traffic for a particular state at a given point of time, building on a prior CS4624 project [14].

For both goals it was crucial that we be able to extract the data first. We were given Parquet files of each iteration of the Colorado website to extract data from. These Parquet files contained snapshots of each version of the Colorado website from random years to the present [1]. After becoming familiar with what types of information were present in these Parquet files, we determined that we were able to extract the data that Dr. Zach requested. We created a parser using Python's Pandas library and Beautiful Soup, which would sift through all of the Parquet files and pull out any relevant information we desired. We organized each of our desired forms of data into their own JSON files for each Parquet file analyzed and placed them within their own respective folders (all raw text for each Parquet file went in the raw text folder). We did this not only so that Dr. Zach is able to easily gather the data that he needs but also so that other teams could continue, as with the prior team [14].

We then created visualization code that took all the created keyword CSV files and then plotted a bar graph to show how much of each keyword was used within a particular time span. Our hope is that with the work we have completed, in being able to correctly parse and store the data in an easy to read format, future teams are able to easily pick up our work and expand upon it, as with the prior team [14].

1.1 Teams Roles

Our team had a person assigned to each of two different roles: Project Lead and Data Extractor/Visualizer. The Project Lead worked as a liaison between Dr. Zach, Dr. Fox and our team, relaying any new or pertinent information to the team in between scheduled meetings. The Data Extractor produced the bulk of our code which was able to parse through the Parquet files and create designated JSON files for the desired data and place them into their own designated folder. The Data Visualizer produced the code that then took the extracted data and created easy to read graphs to help visualize the data. While each individual on the team had their own specific role, all team members helped out with each portion of the project and worked together to produce reports and presentations.

Aditya Agarwal, Project Lead

- Serve as the main point of contact for the project, responsible for documenting the processes.
- Correspond with the client and document all meetings and progress.
- Implement the keyword analysis API

Person 2, Sparsh Bansal, Data Extractor and Visualizer

- Develop how the data will be visualized to the users
- Research on SEO tools and meta tags
- Find any kind of a relationship between existing data and the data we are going to collect

2. Requirements

2.2 Data Extraction:

Parquet files, as mentioned above, were given to us by the previous year's team. Refactoring of this data is necessary to take the information from complicated and lengthy Parquet files and make it easy to understand and analyze. Through these methods, four Parquet files, containing over 50,000 snapshots each, were condensed into one file with meaningful pieces of data in it. We were then tasked to extract relevant information, pointed out by Dr. Florian Zach. Main extraction elements included meta tags and raw text of the files [2]. We extracted specific items from these elements, like the timestamp, as well as the tags and website keywords mentioned above. The data was extracted using Python and its external libraries. Python's Pandas helped read these Parquet files for the extraction. Beautiful Soup was used to extract specific elements from the Parquet files.

2.3 Data Visualization:

The main prerequisite for running the visualization program is to have previously run the data extractor on the Parquet files. This is needed so that we can accurately pull the data needed to populate the graph. We used a stacked bar chart to view our data extraction. The data was structured by year, for all tourism websites. The previous group structured their graphs to include seasonal visualization, to see change by months. We however decided to go with the approach of making graphs for each state in a chronological order, so that we can utilize the existing data to its fullest extent. To make the graphs, we used a tool called Plotly, which is a part of the Dash library in Python. Plotly lets you set up the graphs as per your requirements. For our project, we specified characteristics for the data point, the x axis, the y axis, and the grid itself. Plotly displays these graphs on the running system's local host directory which is set to https://127.0.0.1:8050/ by default on Windows machines.

3. Design

3.1 Extractor

Data extraction from the multiple .snappy.parquet files to a cumulative JSON file is done by the data extractor script (script_1_extract.py) which needs no arguments to run. The extracted data is stored in a JSON file with a name that is specified in the program.
This script makes use of BeautifulSoup and the Python dictionary data type to extract the date (MMYYYY format) as well as the corresponding keywords for that date, and to store them as a JSON object in the output file.
The script also makes sure that the dates are not being repeated and that even the characters outside the Unicode-8 range are being taken into consideration.

3.2 Keyword JSON file

An example of the JSON file used to store the keywords is shown in Figure 1.


```
[{"Sep 2015": ["visit", "cottages", "seafood", "travel", "civil war", "nature", "tourist", "dining", "river", "guide", "tourism",
"vacations", "Destinations", "map", "camping", "travel information", "Culinary", "virginia", "foliage", "theme parks",
"attractions", "history", "travel guide", "mountains", "tourism board", "golf", "beach", "museums", "fun", "Summer", "Food", "bed
breakfast", "family fun", "virginia is for lovers", "heritage"]}, {"May 2007": []}, {"Jan 2009": []}, {"Jul 2009": []}, {"May 2007":
```


(Figure 1: Format of the created JSON file)

This file enables us to use the other scripts directly and multiple times rather than having to run the base script again and again. This way, we can ensure that the most time-consuming operation only has to take place once for all the data and then we can process it anyway we want.
The JSON file also makes debugging a lot easier since we have an intermediate stage in the data extraction process that we can refer to and make sure things are running smoothly.

3.3 Processing JSON data

After the JSON file is created and saved, we run this file through the data processing script (script_2_data_processing_from_json_file.py) and get a more refined version of the data in the JSON file. This script is never called directly and has been added as a helper script for the final visualization process.

The data processing includes rearranging the JSON data in a chronological order, and creating data tables for the x-axis and the y-axis. These data tables are returned to the calling script and used to plot the final graphs.

3.4 Data visualization

Data visualization in a graphical form is done using the Dash Python library; the tool specifically is called plotly.dash [10]. The script used for this is script_3_plotly_script.py. It takes the JSON file for the state, runs script_2 on it, gets data tables for the x and y axes, and then goes on to generate the various graph components.
This script sets the following for our graph:

- The tracing tool is set to a “Marker” with a defined width.
- Layout follows the preset colors, size, and the visibility of the grid, the labels, and the lines.
- The margins around the graph are set for its placement, and to ensure sufficient white space.
- The properties of the graph are set, like width, height, background color, etc.

Once this is done, we are ready to run the local server to display this graph [11].

4. Implementation

4.1 Coding Environment

- Operating System used: Windows 10
  - Undocumented support may exist for macOS
- Development Environment: Microsoft Visual Studio Code
- Python version 3.7

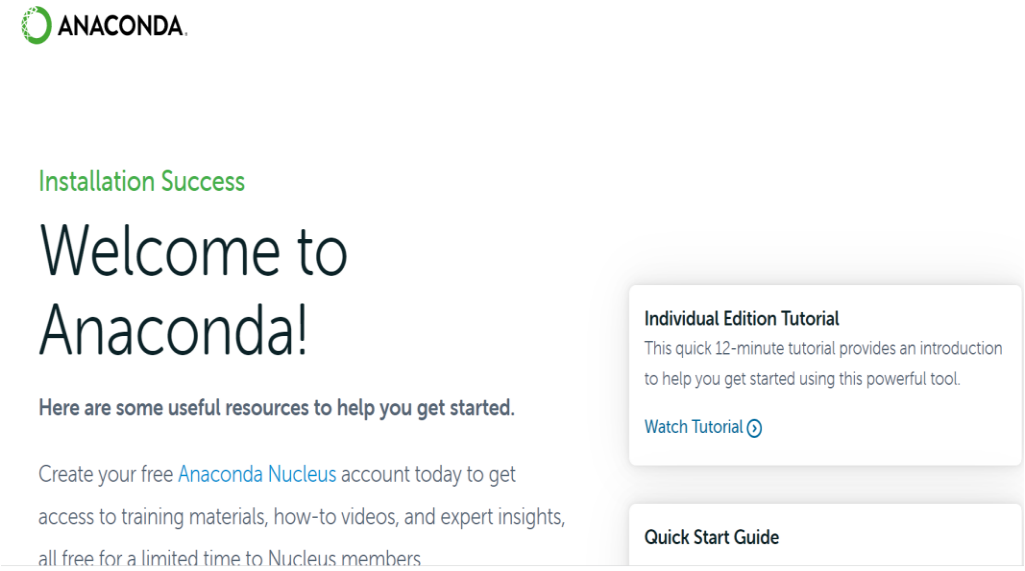
4.2 Language + Tools

- Python version 3.7
- Google Drive: Sharing data and Backup
- GitHub: Manage our code and version control
- pip: Download Python libraries

4.3 Pre-requisites

The following Python libraries need to be installed to run the code:

1. **Anaconda:** Anaconda is an open-source distribution for Python and R [8]. Download (see Figures 2 and 3) from: https://docs.anaconda.com/anaconda/install/windows/

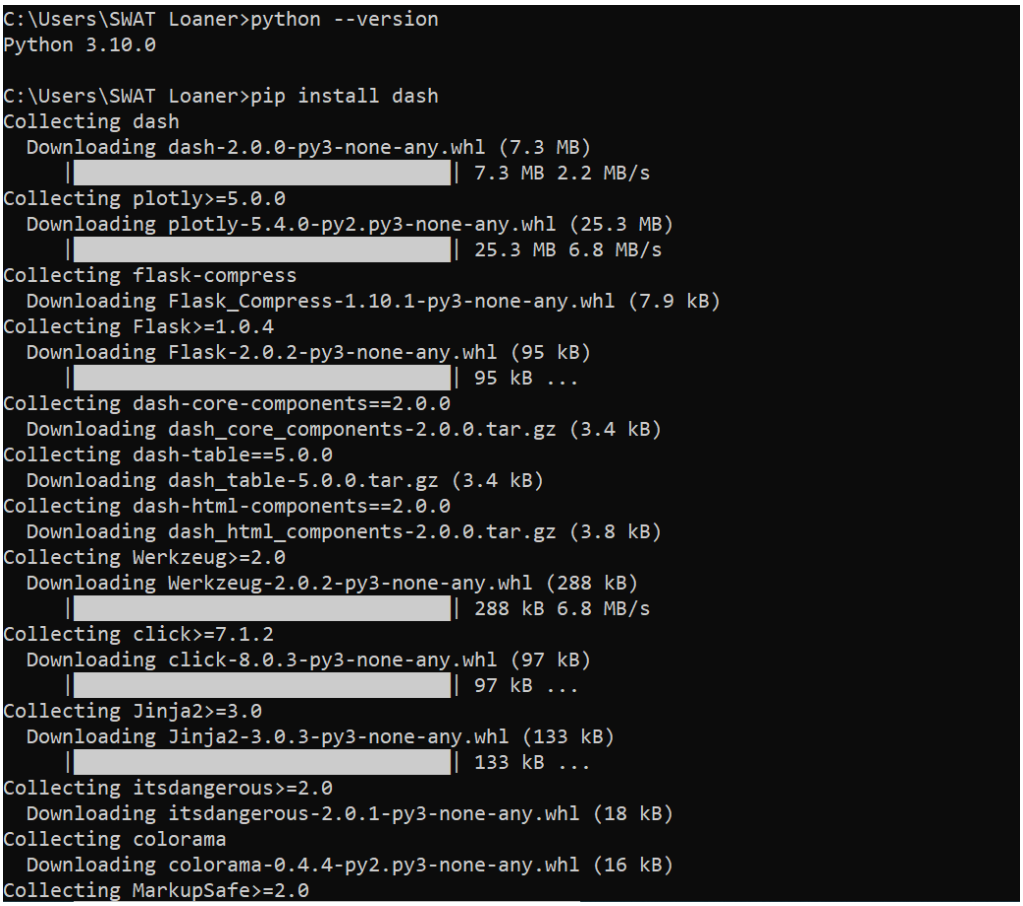


(Figure 2: Anaconda website)

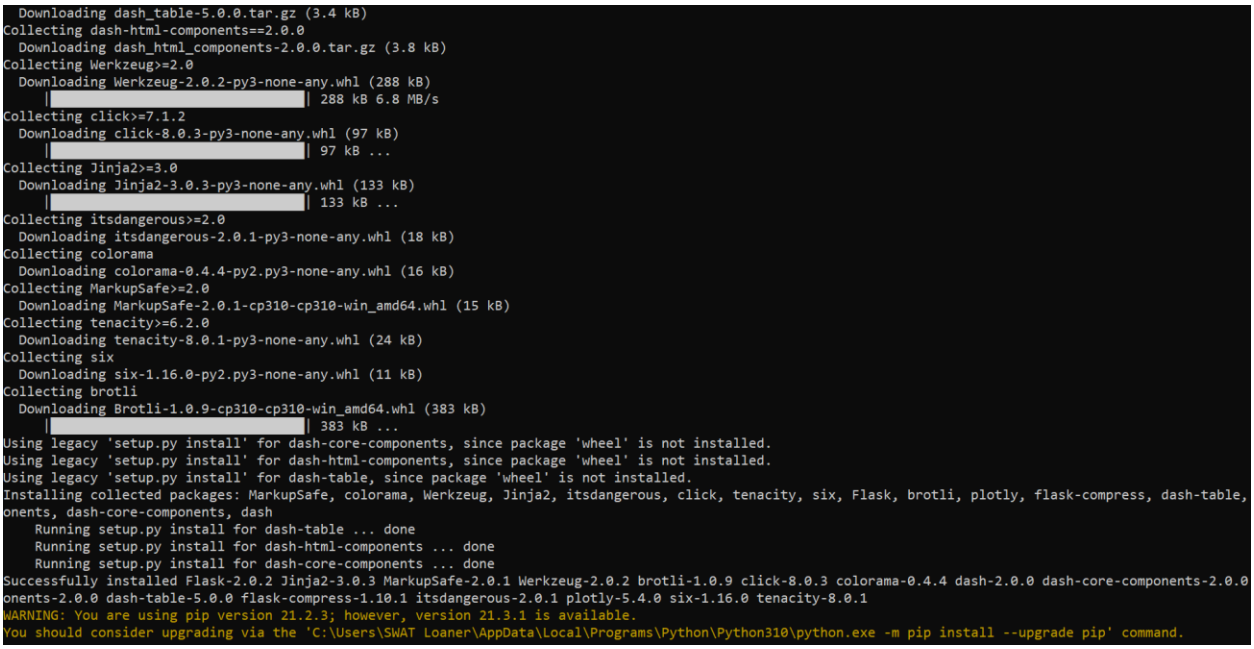
(Figure 3: Anaconda Setup Wizard)

2. **Dash:** Dash is a Python framework created by Plotly for creating interactive web applications. Dash is open source and the applications built using this framework are viewed on the web browser [5].

   Command: “pip install dash”



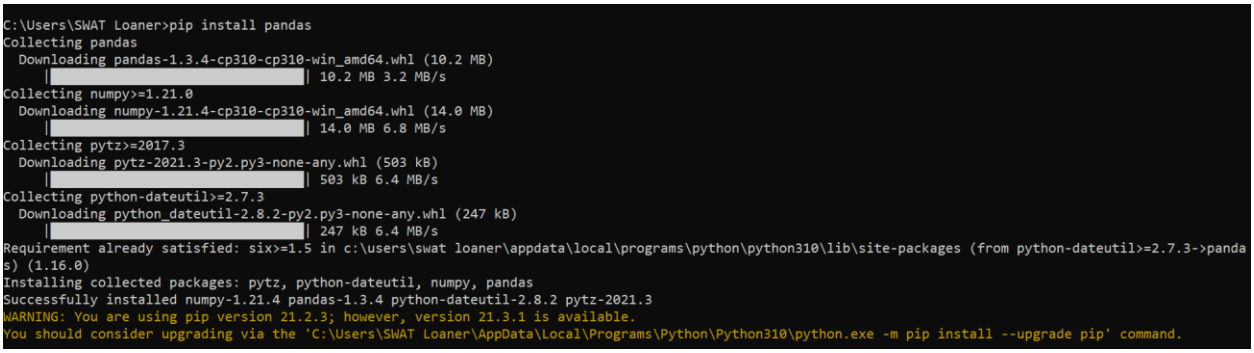
(Figure 4: Execution of pip install dash)



(Figure 5: Execution of pip install dash)

3. **Pandas:** Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

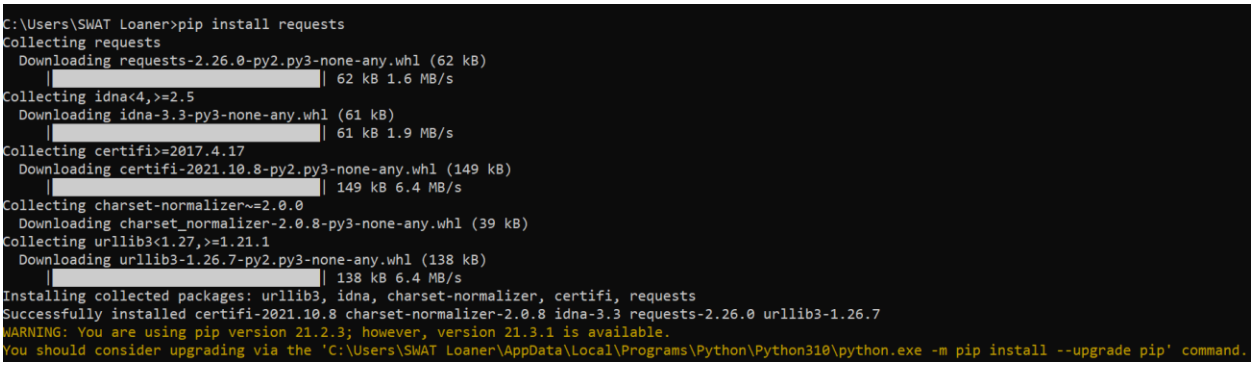
   Command: “pip install pandas”



(Figure 6: Execution of pip install pandas)

4. **Requests:** Requests is an HTTP library for the Python programming language. It helps to make HTTP requests simpler and easier.

   Command: “pip install requests”



(Figure 7: Execution of pip install requests)

5. **Beautiful Soup:** Beautiful Soup is a Python package for parsing HTML and XML documents. It creates a parse tree for parsed pages that can be used to extract data from HTML, which is useful for web scraping [9].

   Command: “pip install BeautifulSoup4”

(Figure 8: Execution of pip install BeautifulSoup4)

6. **PyArrow:** Apache Arrow is a development platform for in-memory analytics. The Arrow Python bindings (also named “PyArrow”) have first-class integration with NumPy, Pandas, and built-in Python objects**.**

4.4 Loading All Data + Config File

All data for each state was saved in a folder; the location of this folder (path) needs to be given to the script.
This makes a JSON file for the given state that extracts all the keywords from the given archive files.
This is done so that we don't have to use the parser again and again; it saves a lot of time.

4.5 Extracting Meta Tags

The meta tags are extracted using "script_1_extract.py". This script makes a dictionary of the date and all the keywords from each archive file and adds it to the state's JSON file.
This script does not take any arguments.
The folder path, and the name of the final file, will have to be changed manually every time we run the script.

4.6 Amazon Alexa API

Alexa Web Information Service (AWIS) is an API so developers, researchers, and website owners can access website traffic data, related links, etc. [4].

AWIS supports the following actions:

UrlInfo - offers access to Alexa's information about websites, including Traffic Rank and site statistics [7].

TrafficHistory - returns historical Alexa Traffic Rank, Reach per Million, and Pageviews per Million metrics for websites -- updated daily [7].

SitesLinkingIn - returns a list of websites linking to a given website -- updated weekly [7].

Charges for this tool (see Figure 9) go to the user's AWS account, according to usage. Recent requests are cached locally to reduce costs for repeated requests [6].

Pricing Information

This software is priced along a consumption dimension.
Your bill will be determined by the number of units you use. Additional taxes or fees may apply.

Alexa Web Information Service

Units	Cost
0 to 100 units (1 unit = 10 API requests)	$0 / unit
101 to 10,000 units (1 unit = 10 API requests)	$0.036 / unit
10,001 to 100,000 units (1 unit = 10 API requests)	$0.018 / unit
Greater than 100,000 units (1 unit = 10 API requests)	$0.003 / unit

(Figure 9: Prices of Alexa Web API)

To use the API in Python, we use the requests (or re) Python library. The following script components are required to access the information properly [6]:

- API-key: The key that you receive when you subscribe to the API.
- Params: The list of parameters that you will be required to change based on the kind of request you are trying to make to the server.

```
import requests
import pandas as pd


headers = {
    'x-api-key': 'lDnD9oe3FuoG31A2eoAZ5Jt2eR8F4TX2mSb4sVBd',
}

params = (
    ('Action', 'UrlInfo'),
    ('Count', '10'),
    ('ResponseGroup', 'Rank'),
    ('Start', '1'),
    ('Url', 'https://www.facebook.com'),
)
```

(Figure 10: Our API implementation)

4.7 Visualization - Tables with data trends

Plotly dash is a very powerful visualization tool.
The tool can be run using “script_1_plotly_script.py”. This script takes in a JSON file and uses a data fetching function from “script_2_data_processing_from_json_file.py” to process the data into a Plotly ready format.
We then define the layout and all the visual requirements for the figure. These requirements can be changed as per the user’s requirements and choices.
The Plotly script returns a list of all the dates that were present for the given state and then makes a graph at the Dash default local IP address:
https://127.0.0.1:8050/
This will only work on the host machine till the Python script is not updated. The graph can be downloaded in a PDF format and used as required [12] [13].

5. User's Manual

In order to start running the program, please download all the libraries that are listed in the Developer's Manual. Open the command prompt in your local host and check the version of Python the device is running. If the device doesn't have Python installed, please install it from the internet. After completing this process, proceed and open the command prompt again and install all the libraries. All the commands to install a particular library are listed in the Developer's Manual. Just type the command in the prompt and libraries will automatically get installed.

After completing these prerequisites, the machine has all the necessary libraries installed so the code will run smoothly. The first step that the user needs to do is to download all the data files and save them in one folder. The user needs to copy the path where the data folder is stored because the user needs to copy this path in the Python script and the output JSON file will be made in the same directory. After running the Python script, the user will again go to that folder and see a JSON file there. After getting the JSON file, the user needs to update the name of the JSON file in the Python script, which is making the Plotly representation. In order to do that, open the Python script and in it add just the name of the JSON file. Then run the Python script. After running the Python script, the user will see the output in which all the years will be mentioned for which the user has the data. Moreover, there will be a Dash link with format: http://127.0.0.1:8050/ . If the user opens this link, the graph will appear which contains the keywords and the time they were used in. Above this graph, the user could see a lot of different functions such as zoom in, zoom out, download as PNG, autoscale, etc. These are some of the built-in functions provided by the Plotly Dash library. The user can download the graph using those icons. Please read all the instructions carefully and use it as required.

Using this command, the user can run the Python files (see Figure 11).

```
PS C:\Users\SWAT Loaner\Desktop\code> python .\script_3_plotly_script.py
```

(Figure 11: Command to run Python scripts)

Command: python *name_of_the_file.py*

6. Developer's Manual

6.1 Library Installation

Install the following Python modules before running the Python scripts

- Pyarrow
- Pandas
- Parquet
- Dash
- Beautiful Soup

6.2 Loading files

Extraction, and JSON output

- Run the “script_1_plotly_script.py” file to make JSON files for all the states, or simply use the created JSON file.
- We can change the input path, and the name of the final file depending on what we need. This information can be changed as shown in Figure 12.
    - The first marking indicates the path to the files, relative to the current running directory.
    - The second marking indicates the name of the file that the script is going to create.

```
directory_list = list()
complete_list = []
for root, dirs, files in os.walk('./California/California', topdown=False):
    for name in files:
        directory_list.append(os.path.join(root, name))
for i in directory_list:
    print(i)
    complete_list.append(extract(i))
    print(complete_list)

print(complete_list)

#json.dump( complete_list, open( 'california.json', 'w') ,ensure_ascii=False)


json_file = 'california.json'
with open(json_file, 'w', encoding='utf8') as json_file:
    json.dump(complete_list, json_file, ensure_ascii=False)
```

(Figure 12: Developer points in the extraction script)

6.3 Handling Visualization

- Run the script script_3_plotly_script.py to generate the graphs on your machine's localhost server.
- We can change the name of the input file, RGB color keys for each component, the label for each of the axes, the visibility of the components, and much more. This information can be changed as shown in Figure 13.
    - The first marker indicates the file name which can be changed to get the desired JSON input.
    - The second marker indicates the existing RGB matrix that is being used as the color code for the graph.

```
file_name = 'california.json'
months, keywords =
script_2_data_processing_from_json_file.data_fetching(file_name)

fig = go.Figure()


fig.add_trace(go.Scatter(
    x=months, y=keywords,
    name='Keywords wise data',
    marker=dict(
        color='rgba(0, 0, 255, 0.95)',
        line_color='rgba(0, 0, 255, 1.0)'
    )
))
```

(Figure 13: Developer points in the visualization script)

7. Methodology

7.1 Goals of our Users

To analyze the traffic trends for three states

7.2 Subtasks of our goals

- Extract Meta Tags
- Consume RestAPI
- Generate Visualize tables

7.3 Workflows

These descriptions should help the users to understand and interpret the actual working of our project.

Workflow #1

User → Goal 1 → Workflow 1

Workflow 1 = Service 1A + Service 1B + Service 1C

Service 1A: Extract Data: Go through the Parquet file and grab all dates that are present in the data.

Service 1B: Extract Data: Go through Parquet files and grab Meta Tags.
From the meta tags, filter out the content associated with
name="keywords"

Service 1C: Export the data into one file

b. Workflow #2

User → Goal 2 → Workflow 2

Workflow 2 = Service 2A + Service 2B + Service 2C

Service 2A: Visualization: Output JSON object from the collected data

Service 2B: Visualization: Change the Plotly Dash Python script and add the JSON file of the state for which one needs to visualize the graph.

Service 2C: Visualization: Output a graph showing keywords used in that particular year to search that state.

8. Results:

Figure 14 illustrates the results of running the Python script on the State of Colorado's data, using shell commands.

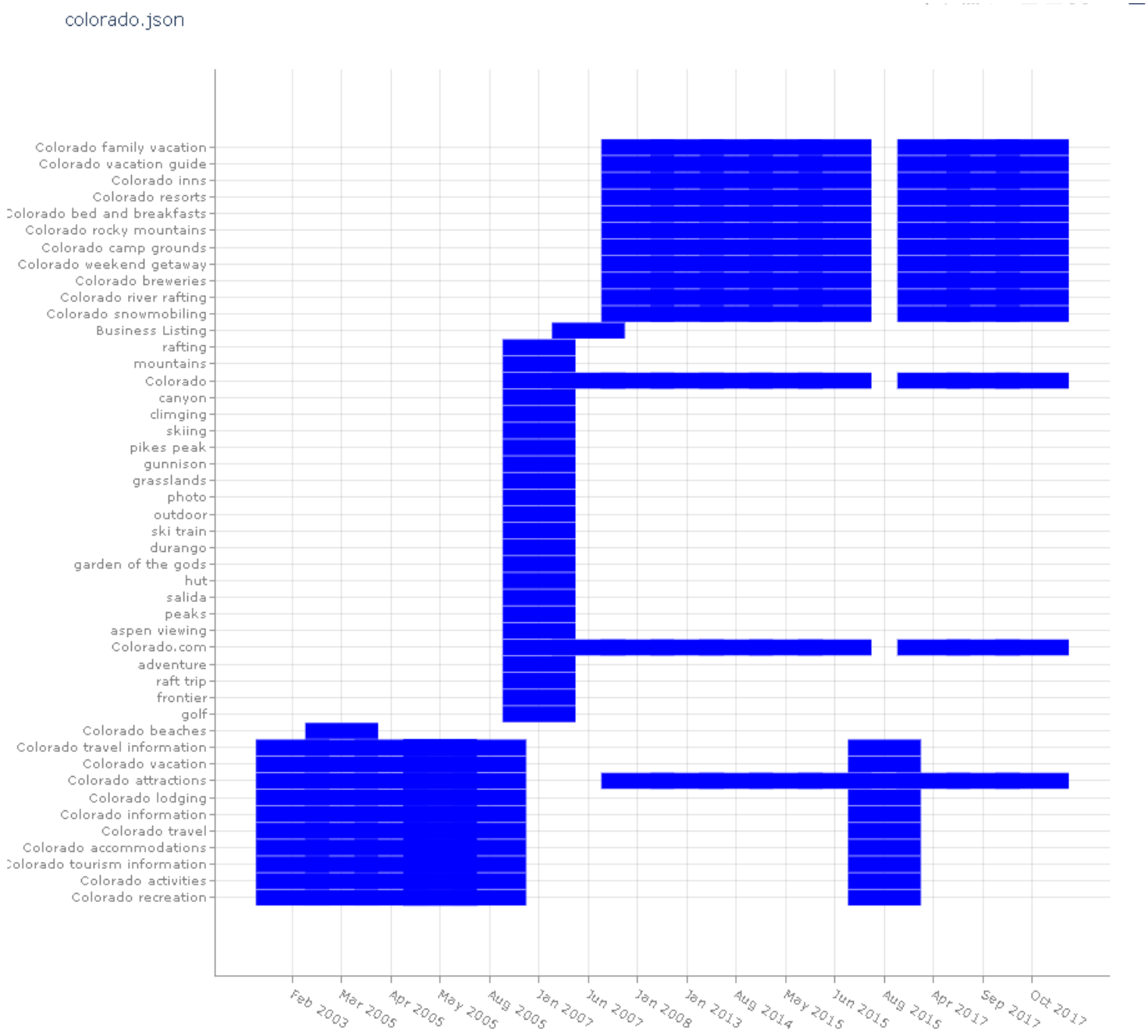
```
PS C:\Users\SWAT Loaner> cd .\Desktop\
PS C:\Users\SWAT Loaner\Desktop> cd .\code\
PS C:\Users\SWAT Loaner\Desktop\code> python .\script_3_plotly_script.py
['Jan 2008', 'Apr 2005', 'Mar 2005', 'May 2005', 'Jun 2007', 'Jan 2007']
['Mar 2005', 'Apr 2005', 'May 2005', 'Jan 2007', 'Jun 2007', 'Jan 2008']
111
111
Dash is running on http://127.0.0.1:8050/

 * Serving Flask app 'script_3_plotly_script' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
['Jan 2008', 'Apr 2005', 'Mar 2005', 'May 2005', 'Jun 2007', 'Jan 2007']
['Mar 2005', 'Apr 2005', 'May 2005', 'Jan 2007', 'Jun 2007', 'Jan 2008']
111
111
```

(Figure 14: Code output by running Colorado's JSON file)

GRAPH 1 (COLORADO):

Table 1 illustrates the output from running the data files for Colorado.



(Table 1: Graph of Colorado)

Figure 15 illustrates the results of running the Python script on the State of California's data, using shell commands.

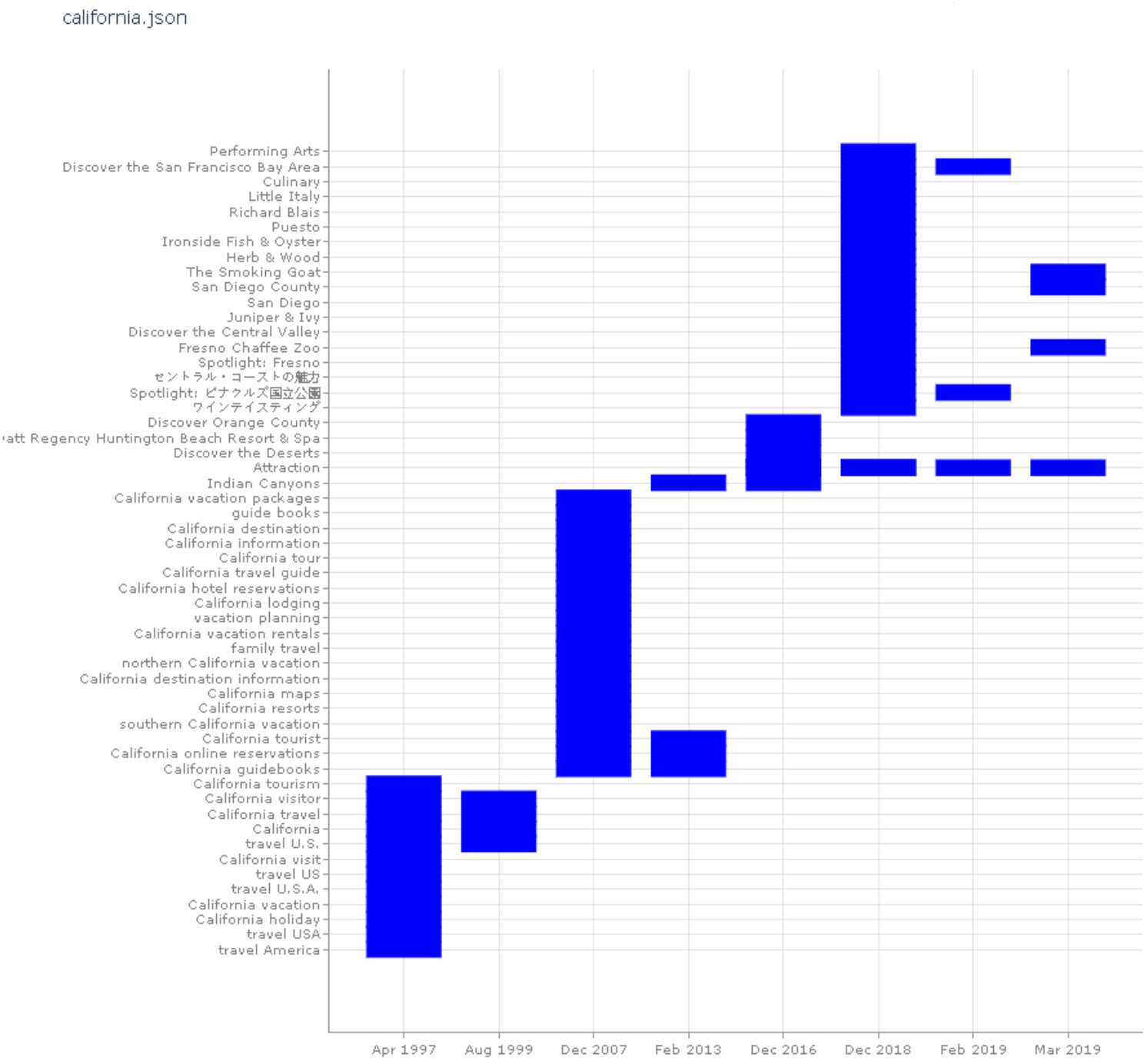
```
PS C:\Users\SWAT Loaner\Desktop\code> python .\script_3_plotly_script.p
['Dec 2018', 'Dec 2016', 'Dec 2007', 'Apr 1997']
['Apr 1997', 'Dec 2007', 'Dec 2016', 'Dec 2018']
58
58
Dash is running on http://127.0.0.1:8050/

 * Serving Flask app 'script_3_plotly_script' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production
   Use a production WSGI server instead.
 * Debug mode: on
['Dec 2018', 'Dec 2016', 'Dec 2007', 'Apr 1997']
['Apr 1997', 'Dec 2007', 'Dec 2016', 'Dec 2018']
58
58
```

(Figure 15: Code output by running the California's JSON file)

GRAPH 2 (CALIFORNIA):

Table 2 illustrates the output from running the data files for California.



(Table 2: Graph of California)

Figure 16 illustrates the results of running the Python script on the Commonwealth of Virginia's data, using shell commands.

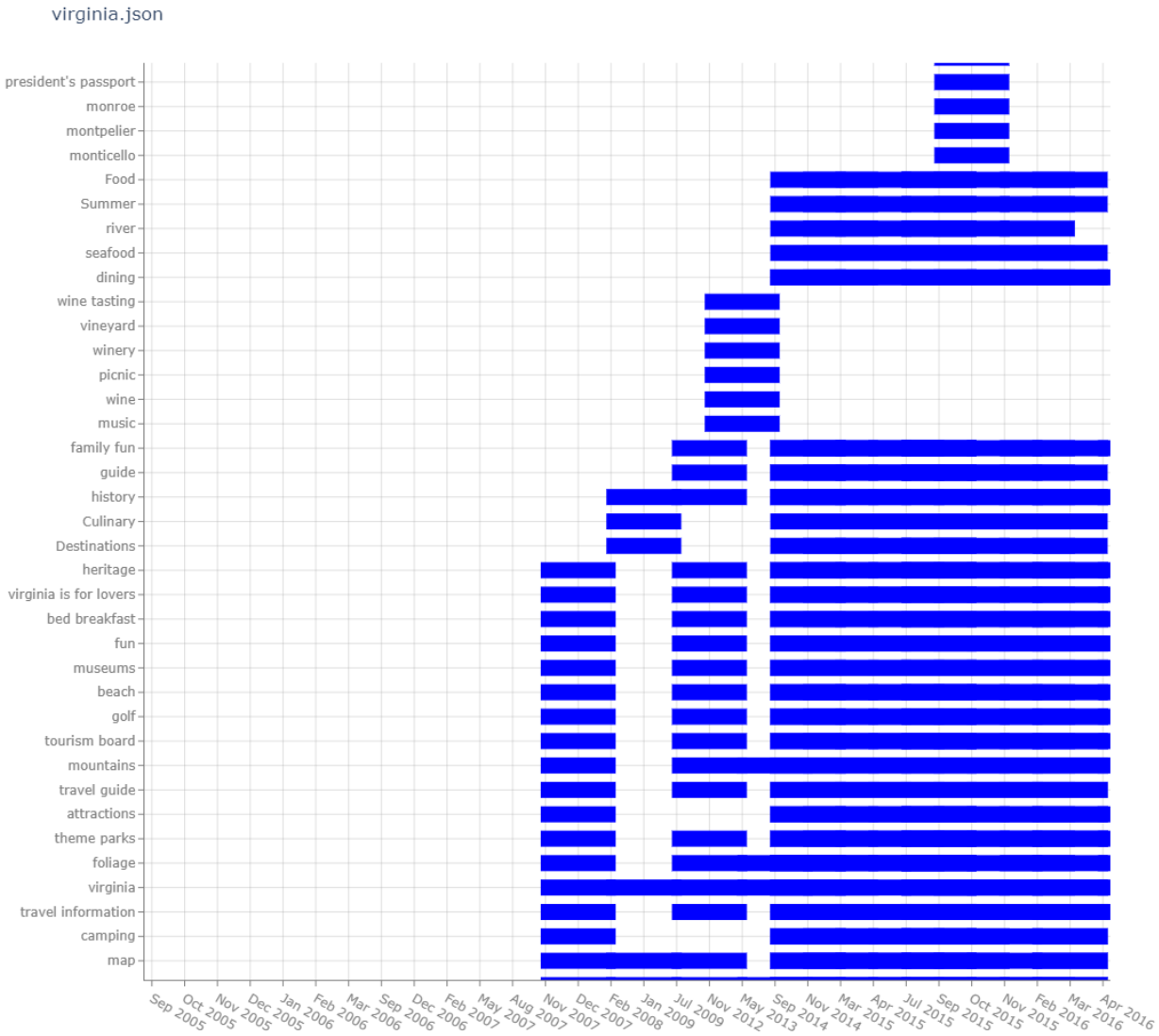
```
PS C:\Users\SWAT Loaner> cd .\Desktop\
PS C:\Users\SWAT Loaner\Desktop> cd .\code\
PS C:\Users\SWAT Loaner\Desktop\code> python .\script_3_plotly_script.py
['Sep 2015', 'Feb 2008', 'Nov 2012', 'May 2013', 'Sep 2014', 'Nov 2015', 'Dec 2007', 'Mar 2016', 'Jul 2015', 'Jan 2019', 'Apr 2016', 'Oct 2015']
['Dec 2007', 'Feb 2008', 'Nov 2012', 'May 2013', 'Sep 2014', 'Jul 2015', 'Sep 2015', 'Oct 2015', 'Nov 2015', 'Mar 2016', 'Apr 2016', 'Jan 2019']
242
242
Dash is running on http://127.0.0.1:8050/

 * Serving Flask app 'script_3_plotly_script' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
['Sep 2015', 'Feb 2008', 'Nov 2012', 'May 2013', 'Sep 2014', 'Nov 2015', 'Dec 2007', 'Mar 2016', 'Jul 2015', 'Jan 2019', 'Apr 2016', 'Oct 2015']
['Dec 2007', 'Feb 2008', 'Nov 2012', 'May 2013', 'Sep 2014', 'Jul 2015', 'Sep 2015', 'Oct 2015', 'Nov 2015', 'Mar 2016', 'Apr 2016', 'Jan 2019']
242
242
```

(Figure 16: Code output by running the Virginia JSON file)

GRAPH 3 (VIRGINIA: Zoomed In)

Table 3 illustrates the output from running the data files for Virginia.



(Table 3: Graph of Virginia)

9. Scripts

Script 1: script_1_extract.py

```
import pyarrow.parquet as pq
from bs4 import BeautifulSoup as bs
import re
import os
import json

def extract(file_name):
    masterDict = {}

    df = pq.read_table(source=file_name).to_pandas()

    #print(df)
    htmlContent = df.payload[1]

    soup = bs(htmlContent, 'lxml')

    dateStr = re.search("Date:\s*.+", soup.text).group(0)

    monthAndYear = re.search("[a-zA-Z]{3} [0-9]{4}", dateStr).group(0)

    metaTags = soup.find_all('meta')

    keywords = []

    for tag in metaTags:
        attrs = tag.attrs

        if 'name' in attrs:

            if attrs['name']=="keywords":
                keywordStr = tag.get('content')
                keywordArr = keywordStr.split(",")

                for kwd in keywordArr:
                    kwd = kwd.strip()
                    keywords.append(kwd)
```

```
    if monthAndYear not in masterDict:
            masterDict[monthAndYear] = list(set(keywords)) # converting it to set and
back to list to remove duplicates
            masterDict[monthAndYear] = [kwd for kwd in masterDict[monthAndYear]
if kwd != '']

    else:
            masterDict[monthAndYear].append(keywords)
            masterDict[monthAndYear] = list(set(masterDict[monthAndYear]))
            masterDict[monthAndYear] = [kwd for kwd in masterDict[monthAndYear]
if kwd != '']

    print(masterDict)
    return masterDict


directory_list = list()
complete_list = []
for root, dirs, files in os.walk('./California/California', topdown=False):
    for name in files:
        directory_list.append(os.path.join(root, name))
for i in directory_list:
    print(i)
    complete_list.append(extract(i))
    print(complete_list)

print(complete_list)

#json.dump( complete_list, open( 'california.json', 'w') ,ensure_ascii=False)


json_file = 'california.json'
with open(json_file, 'w', encoding='utf8') as json_file:
    json.dump(complete_list, json_file, ensure_ascii=False)
```

(Figure 17: Script 1: script_1_extract.py)

Script 2: script_2_data_processing.py

```
import json
from datetime import datetime
import time

def data_fetching(json_file):


    data = json.load( open( json_file, encoding="utf8" ) )

    month_list_unsorted = []
    new_data = []
    for i in data:
        for j in i.keys():
            if len(i[j]) == 0:
                pass
            else:
                new_data.append(i)

                if j in month_list_unsorted:
                    pass
                else:
                    month_list_unsorted.append(j)


    print(month_list_unsorted)
    unsorted_dates = [datetime.strptime(value, '%b %Y') for value in
month_list_unsorted]
    sorted_dates = sorted(unsorted_dates)
    month_list_sorted = [value.strftime('%b %Y') for value in sorted_dates]

    print(month_list_sorted)

    ################ Data arrays ############################
    x_axis = []
    y_axis = []
    for i in month_list_sorted:
        for j in data:
            for k in j.keys():
                if i == k :
```

```
            for m in j[k]:
                x_axis.append(k)
                y_axis.append(m)


print(len(x_axis))
print(len(y_axis))
return x_axis,y_axis
```

(Figure 18: Script 2: script_2_data_processing.py)

Script 3: script_3_plotly_script.py

```
import dash
from dash import dcc
from dash import html
import plotly.express as px
import pandas as pd
import plotly.graph_objects as go
app = dash.Dash(__name__)

import script_2_data_processing_from_json_file

file_name = 'california.json'
months, keywords =
script_2_data_processing_from_json_file.data_fetching(file_name)

fig = go.Figure()


fig.add_trace(go.Scatter(
    x=months, y=keywords,
    name='Keywords wise data',
    marker=dict(
        color='rgba(0, 0, 255, 0.95)',
        line_color='rgba(0, 0, 255, 1.0)'
    )
))

fig.update_traces(mode='markers', marker=dict(line_width=15, symbol='line-ew',
size=50))

fig.update_layout(
    title= file_name,
    xaxis=dict(
        showgrid=True,
        gridcolor='rgba(166, 166, 166, 0.35)',
        gridwidth=1,  # tried different values, same issue
        showline=True,
        linecolor='rgb(128, 128, 128)',
        tickfont_color='rgb(128, 128, 128)',
        showticklabels=True,
```

```
        dtick=1,
        ticks='outside',
        tickcolor='rgb(128, 128, 128)',
    ),

    yaxis=dict(
        showgrid=True,
        gridcolor='rgba(166, 166, 166, 0.35)',
        gridwidth=1,  # tried different values, same issue
        showline=True,
        linecolor='rgb(128, 128, 128)',
        tickfont_color='rgb(128, 128, 128)',
        showticklabels=True,
        dtick=1,
        ticks='outside',
        tickcolor='rgb(128, 128, 128)',
    ),

    margin=dict(l=140, r=40, b=50, t=80),
    legend=dict(
        font_size=10,
        yanchor='middle',
        xanchor='right',
    ),

    width=1100,
    height=1000,
    paper_bgcolor='white',
    plot_bgcolor='white',
    hovermode='closest',
)


app.layout = html.Div(children=[
    html.H1(children='Hello Dash'),

    html.Div(children='''
        Dash: A web application framework for your data.
    '''),
```

```
    dcc.Graph(

        id='example-graph',
        figure=fig
    )
])

if __name__ == '__main__':
    app.run_server(debug=True)
```

(Figure 19: Script 3: script_3_plotly_script.py)

SCRIPT 4

```
import requests
import pandas as pd


headers = {
    'x-api-key': 'lDnD9oe3FuoG31A2eoAZ5Jt2eR8F4TX2mSb4sVBd',
}

params = (
    ('Action', 'UrlInfo'),
    ('Count', '10'),
    ('ResponseGroup', 'Rank'),
    ('Start', '1'),
    ('Url', 'https://www.facebook.com'),
)

response = requests.get('https://awis.api.alexa.com/api', headers=headers,
params=params)
print(response.content)
list1 = []
for i in response.content:
    #print(i)
    list1.append(i)

print(list1)
print(len(list1))
```

(Figure 20: Script 4: test 1.py)

10. Lessons Learned

10.1 Timeline / Schedule

- Sept-7: Meet the client and discuss the various objectives for this project. Use cookies and tracking pixels to understand the data collected by websites and how it influences their activities.
- Sept-14: Decide on the aspect of the project we will be working on and get approval from the client.
- Sept-21: Presentation1
- Sept-28: Write Python code and add API support to extract meta tags from each website
- Oct- 5: Document meta tags collected from each of the websites.
- Oct-12: Use Alexa website tool for analyzing the website traffic
- Oct-19: Plot increasing or decreasing activity for each of the websites
- Oct-26: Arrange all keywords and deliverables from the collected data
- Nov-2: Start Visualization as instructed by the client
- Nov-9: Make the box plots for each keyword based on usage
- Nov-16: Run all the keywords through ALEXA and collect required data
- Dec-7: Deliver complete Report and all the collected data

10.2 Problems

One of the problems we faced was the large learning curve of understanding all our tools. None of our team members had a very good grasp on Python and its libraries. That made parsing HTML from a Parquet file using BeautifulSoup a little harder task than we thought. Just getting to open up a Parquet file was a challenge in itself. We learned from the previous group's work. But since our way of arranging files and extracting data was different, we had to largely start from scratch, but keep in mind future extension of this script and the data.
Another challenge was working with such a large volume of data. When downloading the data from the shared drive in very large batches, it would often get corrupted, which led to some of the files being incomplete, and it would take us running the whole script to realize there is something wrong with the data. This took a lot of our working time and we had to download every file individually to make sure we had all the data.
Then we had the challenge of exploring the Dash library. Its scale and complexity led to a lot of design changes in the final graph design.

We also faced the problem that the Alexa API that we wanted to use to get website rankings and then compare the change in ranking with the addition / removal of keywords had a limitation that they only accepted dates within the past 4 years. This did not match with the larger range of dates in our data set. We tried to use other tools in the same API but they were all in real time and so were not of as much significance to this project.

10.3 Solutions

The large learning curve of all the technical tools was a relatively easy problem to solve. Throughout our collegiate years at Tech, we have developed a fair idea of how different programming languages work. Overall, our group had a very low understanding of Python, but we did some smaller projects and tried keeping it fairly simple. It also took us a while to understand the .snappy.parquet files and to explore all the components they have. This gave us a good idea of the Internet Archive and also of why the previous teams chose to work with these file formats. After working on the project and finishing up, we have a better understanding of Parquet files and how to utilize them. Beautiful Soup is a very useful tool for web scraping with Python. We found Beautiful Soup to help us with reading Parquet files and extracting needed data from them.

To fix the problem of data corruption on download, we had to manually download each file individually and then set up the directories for the scripts to run. Our client also helped us here by giving us access to the data over another platform but it turned out to be some network issue, wherein very high file sizes would time out after some time.

To find the useful components from the Dash library, we followed YouTube tutorials and other projects. This library had a lot of components and labels which took us a lot of time and reading to keep track of.

The API component of the project could not be integrated very well with the keyword aspect. Although we have the script and the collection ready to be used, the lack of data proved it not effective and we were not able to get any results from the API. It was a really important and big part of this project and we were really excited to use RESTful API’s and experiment with the various tools the Alexa API has to offer. This can be a potential direction for teams to work towards in the future.

Overall, the project gave us a good perspective of collecting and working with historical data and we now have a much better knowledge and understanding of how we can run analysis on data.

10.4 Future Work

Future teams can work on our existing code and modify it as per their needs. Moreover, the user can analyze the keyword usage for any website. The only thing the user needs is the data files and after running the Python script, the graphs can be generated. Also, the user should be able to take our codebase and apply that to extracted information from other state's Parquet files in order to help with researching state tourism efforts. The other thing is that, by changing the Python script and setting the corresponding field that needs to be extracted in the dictionary, the user can extract images, website description, social media links, and analysis tools the particular website is using and visualize the data. Basically, these Python scripts could be used to extract anything from a given website and then display the results in the graphical manner. If the user doesn't want graphs, then he can make some change in our Plotly-Dash Python script and can add any other library of his own choice and extract the results. The future groups could use this to their advantage and can help with researching state tourism efforts.

11. Acknowledgements

Florian Zach, Ph.D. - Assistant Professor in the Howard Feiertag Department of Hospitality and Tourism Management
Email: florian@vt.edu
Edward Fox, Professor of class CS 4624- Multimedia/Hypertext
Email: fox@vt.edu

12. References

1. Patel, Dr. Birajkumar V., and Dr. Raina D. Gaharwar. *Search Engine Optimization (SEO) Using HTML Meta-Tags.* International Journal of Scientific Research in Science and Technology, 2018, vol. 4, pp. 1–6.
2. Maria Cristina Enache, 2014. "Optimization Methods And Seo Tools," Risk in Contemporary Economy, Dunarea de Jos University of Galati, Faculty of Economics and Business Administration, pages 98-103, https://EconPapers.repec.org/RePEc:ddj:fserec:y:2014:p:98-103.
3. Hardwick, Joshua. *Meta Tags for SEO: A Simple Guide for Beginners*. (29 September, 2020). Retrieved December 10, 2021, from ahrefs.com/blog/seo-meta-tags/.
4. *Keyword Research, competitor analysis, & website ranking: Alexa*. Alexa.com. (2020, October 7). Retrieved November 9, 2021, from https://www.alexa.com/.
5. Dash.com, *Dash documentation & user guide*. Plotly. (2021). Retrieved November 9, 2021, from https://dash.plotly.com/.
6. Catania, P. J., & Keefer, N. (1987). *The Marketplace*. Amazon. Retrieved December 10, 2021, from https://aws.amazon.com/marketplace/pp/prodview-w6qmxismbbs7u.
7. Amazon.com, Developer portal. (2021). Retrieved December 10, 2021, from https://awis.alexa.com/developer-guide.
8. Anaconda.com, *Individual edition*. Anaconda. (2021). Retrieved December 10, 2021, from https://www.anaconda.com/products/individual.
9. Beautiful Soap 4.9.0 Documentation, *Beautiful Soup documentation*. Beautiful Soup Documentation - Beautiful Soup 4.9.0 documentation. (2021). Retrieved December 10, 2021, from https://www.crummy.com/software/BeautifulSoup/bs4/doc/.
10. Dash enterprise. (2021). Retrieved December 10, 2021, from https://dash.gallery/Portal/.
11. Charming Data. *Introduction to dash plotly: Data Visualisation in Python - YouTube*. (2020). Retrieved December 10, 2021, from https://www.youtube.com/watch?v=hSPmj7mK6ng.
12. Awesome Open Source, The top 236 plotly dash open source projects on GitHub. (2018). Retrieved December 10, 2021, from https://awesomeopensource.com/projects/plotly-dash.
13. Ucg8j: Awesome-dash. (2017). *UCG8J/awesome-dash: A curated list of Awesome Dash (plotly) resources*. GitHub. Retrieved December 10, 2021, from https://github.com/ucg8j/awesome-dash.
14. Abhinav Verelly, Ashutosh Bhattarai, Shane Grishaw, David Gruhn. (2021). *US State Tourism,* Virginia Tech Department of Computer Science, Blacksburg, VA, CS4624 Final Report (Spring 2021). Retrieved December 10, 2021, from

http://hdl.handle.net/10919/103269