Jolana Babjak, Jordana Brief, Samuel Slater
Professor Ericson
SI 206
30 April 2024

Holiday Hoarder: Final Report

Goals

Our group was curious to see different holidays across different countries and cultures. We collected information from 3 different APIs and websites, one relating to European holidays, one relating to USA holidays, and a website relating to fun holidays. We were successful in extracting information and creating visuals to understand when the world celebrates various events.

Problems

One problem that came across was for the creation of the month table referenced as a foreign key for the month ids in the Fun_Holidays table. At first, the months by full name, for example, “January”, were put into the table to be referenced and joined with the full Fun Holidays Table. The weekdays table worked just fine, but for some reason, the months weren’t connecting the ids correctly. After some close analysis, I noticed that on the website that the data was being extracted from, the months weren’t written out in full, and rather only the first three letters of each month were included. This was why the tables weren’t properly connecting, so I went back and changed the formatting for the months corresponding to the month ids; for example, changing “January” to “Jan”. And, that properly resolved the issue.

Another problem we encountered was limiting the files to loading 25 items per run. At first, we were attempting to learn new imports, such as time, but the logic was confusing to use and also did not work. Eventually, we discovered that we could simply use a loop that breaks after a self-set number of iterations. This was both much simpler and also more efficient.

Instructions

To ensure all of the data is properly formatted in the database, it is crucial that the fun_holiday_table.py file is run prior to the US23_Dates_db.py file because both reference the id table, Weekday. And, it is created in fun_holiday_table.py. With this in mind, the European_Holidays.py file can be run before, in the middle, or after these files are run in that order. Finally, the visualizations.py file should be run last, after all of the data is loaded in, because it executes calculations and visualizations based off of the data present in the database.

Code Documentation

fun_holidays_table.py file:

Imported Beautiful Soup, requests, sqlite3, re, and os

```
def set_up_database(db_name):
```

The first one is set_up_database, which takes in the parameter “db_name”, which is the name of the current database being used. It then returns the variables, conn, which is the SQLite connection to the database being created, and cur, which is the connection cursor to execute the SQL commands.

```
def day_of_week_table(cur, conn):
```

```
def month_table(cur, conn):
```

The next two functions, day_of_week_table and month_table, both take cur and conn as their parameters and create the foreign keys for the overall database. They both create their respective tables if they don’t exist, counting the items in the table. And, if the count is equal to 0, meaning there is nothing in the table yet, each day of the week or month is added to their respective table with an ID number. This is to ensure that there is no duplicate string data in the overall fun holidays table when the day of the week or month is referenced for each holiday. The functions don’t return anything, and the SQLite connection is committed to the database.

```
def fun_holiday_info (soup):
```

The next function is fun_holiday_info, which takes in a Beautiful Soup object as a parameter. This is because the information is extracted via web scraping and, therefore, Beautiful Soup. The information is extracted from a table on the website, listing the date, weekday, and the fun holiday’s name. For each component, I searched for its specific classification within the HTML file. However, for the date, there was more information provided by the search than desired. So, I had to use regex to extract only the date for each holiday to ensure there was no unnecessary information was included. Additionally, each goes through a for loop, adding the information to its own list. After that, there is one last for loop that adds all of the fun holiday information to a list of dictionaries, along with an incremented ID number. The function then returns the final list.

```
def create_fun_holiday_table(data, cur, conn):
```

Lastly, the function create_fun_holiday_table takes in the parameters data, cur, and conn. Data is a variable that takes in a list of information to create the table in the database, and cur and conn remain consistent with its other references. This function creates the Fun_Holidays table, referencing the foreign keys for the month and weekday IDs. A variable, total_entries, is defaulted to 0, and for loop is then executed iterating through each holiday in the data variable. If total_entries is greater than or equal to 25, the loop breaks. Otherwise, the weekday and month IDs are selected. Then, a variable existing_entry is created, selecting the count at which

a specific holiday is selected. If the existing_entry equals 0, meaning it is not currently found in the table, the item is then inserted in. And, the variable total entries is incremented by 1. In the end, the connection is committed, and the function doesn't return anything.

```
def main():
```

In the main, I created a variable for the URL and sent a request to use Beautiful Soup with an HTML parser. I then initialized and called all of the functions so that they are needed to properly add the tables and subsequent items to the database.

European_Holidays.py file:
Imported Sqlite, requests, and datetime

```
def get_country_holidays(country_iso_code):
```

This takes in a country_iso_code, which is a two-letter internationally recognized short-code that represents a country. Using the ISO code, it fetches the specific information from the API about each respective country and returns them in a json file called country_holidays.

```
def european_holiday_table_maker():
```

This function creates a table to map through ISO codes and then iterates through each country's json file, checking to see if the country's holiday has already been added or not to the a table named european_holidays. If holiday has not been listed, it assigns a unique id to the holiday in european_holiday_id. This function uses the same logic as the fun_holiday_table file to limit to 25 entries per run and also calls get_country_holiday.

Holiday_api_US23.py File:
Imported the Sqlite module, which allows for interaction with sqlite databases
Imported the requests module, which allows for sending HTTP requests to the holiday api.

```
def fetch_holiday_data(api_key, country, year):
```

Function that obtained data from Holiday Api. Took in api_key, country code, and year as parameters. It returned a list of holiday data from the api

```
def create_tables():
```

This function created tables in the database. It created tables "Weekday" and 'Holidays' if they did not exist in the database. Renamed the Holidays table to US holidays.

Created the Holiday table with foreign key references to the Weekday table.

```
def store_data_in_database(data):
```

To store the data in the database, set up a count to accumulate the amount of items stored in the database at a time. Then iterated through the holiday data with a for loop, and set count to be less than or equal to zero, and then once it reached 25 it would break, so it would not store more than 25 items into the database at a time.
Then the function checks if there are duplicates in the database, and if the count for existing holidays is greater than 0, it continues.
Then this function gets the weekday id from the weekday table and inserts it into the US Holiday table.

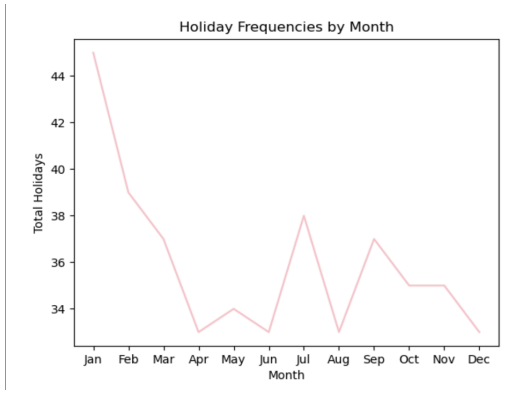
```
def main():
```

This function sets up variables api_key, country, and year.
Then it calls the above functions.

Visualizations and Calculations

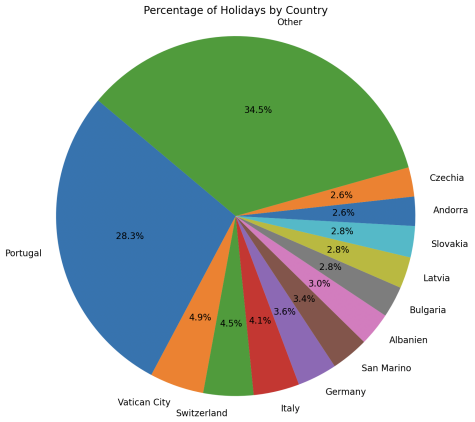
`def create_csv_file(filename):`
`  with open(filename, 'w', newline='') as csvfile:`
`  pass`
	"From the Fun Holidays API, 45 holidays are in Jan"
	"From the Fun Holidays API, 39 holidays are in Feb"
	"From the Fun Holidays API, 37 holidays are in Mar"
	"From the Fun Holidays API, 33 holidays are in Apr"
	"From the Fun Holidays API, 34 holidays are in May"
	"From the Fun Holidays API, 33 holidays are in Jun"
	"From the Fun Holidays API, 38 holidays are in Jul"
	"From the Fun Holidays API, 33 holidays are in Aug"
	"From the Fun Holidays API, 37 holidays are in Sep"
	"From the Fun Holidays API, 35 holidays are in Oct"
	"From the Fun Holidays API, 35 holidays are in Nov"
	"From the Fun Holidays API, 33 holidays are in Dec"
	"From the European Holidays API, 28.3% of holidays are in Portugal"
	"From the European Holidays API, 4.9% of holidays are in Vatican City"
	"From the European Holidays API, 4.5% of holidays are in Switzerland"
	"From the European Holidays API, 4.1% of holidays are in Italy"
	"From the European Holidays API, 3.6% of holidays are in Germany"
	"From the European Holidays API, 3.4% of holidays are in San Marino"
	"From the European Holidays API, 3.0% of holidays are in Albanien"
	"From the European Holidays API, 2.8% of holidays are in Bulgaria"
	"From the European Holidays API, 2.8% of holidays are in Latvia"
	"From the European Holidays API, 2.8% of holidays are in Slovakia"
	"From the European Holidays API, 2.6% of holidays are in Andorra"
	"From the European Holidays API, 2.6% of holidays are in Czechia"
	"From the European Holidays API, 34.5% of holidays are in Other"
	"From the European Holidays API, 8 countries are in Northern Europe"
	"From the European Holidays API, 7 countries are in Western Europe"
	"From the European Holidays API, 9 countries are in Southern Europe"
	"From the European Holidays API, 9 countries are in Central Europe"
	"From the European Holidays API, 12 countries are in Eastern Europe"
	"From the US Holidays API, 31 holidays are on Monday"
	"From the US Holidays API, 12 holidays are on Tuesday"
	"From the US Holidays API, 15 holidays are on Wednesday"
	"From the US Holidays API, 17 holidays are on Thursday"
	"From the US Holidays API, 27 holidays are on Friday"
	"From the US Holidays API, 19 holidays are on Saturday"
	"From the US Holidays API, 24 holidays are on Sunday"
	"From the US Holidays API, 11 public holidays"
	"From the US Holidays API, 134 non-public holidays"
		Opens a new CSV file and writes the data to it.

```python
def get_holiday_frequencies_by_month(db):
    conn = sqlite3.connect(db)
    cur = conn.cursor()
    holiday_freq_dict = {}
    info = cur.execute('SELECT M.month, COUNT(H.id) FROM Fun_Holidays H JOIN Month M ON H.month_id = M.id GROUP BY M.id')
    for holiday in info:
        holiday_freq_dict[holiday[0]] = holiday[1]

    with open('holiday_data.csv', 'a', newline='') as csvfile:
        writer = csv.writer(csvfile)
        for month, count in holiday_freq_dict.items():
            writer.writerow([f"From the Fun Holidays API, {count} holidays are in {month}"])

    plt.plot(list(holiday_freq_dict.keys()), list(holiday_freq_dict.values()), color='pink')
    plt.xlabel('Month')
    plt.ylabel('Total Holidays')
    plt.title('Holiday Frequencies by Month')
    plt.show()

    conn.close()
```



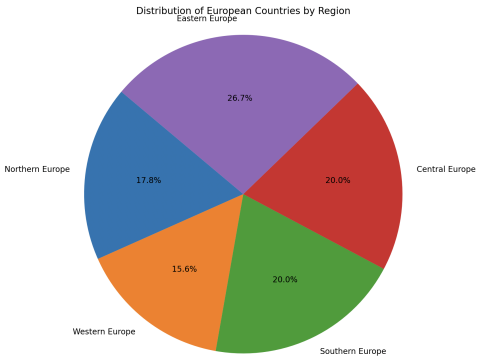
This function takes data from the Fun_Holiday table and joins it with the month_id key. Then it counts which holidays occur in each month and creates a plot showing the frequencies of holidays per month. It writes the amount of holidays occurring each month to the CSV file.

```python
def get_holiday_percentage_by_country(db):
    conn = sqlite3.connect(db)
    c = conn.cursor()
    c.execute('''SELECT c.country, COUNT(h.country_id) AS holiday_count
                 FROM european_country_ids c
                 LEFT JOIN european_holidays h ON c.id = h.country_id
                 GROUP BY c.country''')
    holiday_counts = c.fetchall()
    conn.close()
    holiday_counts.sort(key=lambda x: x[1], reverse=True)
    top_countries = holiday_counts[:12]
    other_count = sum(row[1] for row in holiday_counts[12:])
    top_countries.append(('Other', other_count))
    total_holidays = sum(count for country, count in top_countries)

    with open('holiday_data.csv', 'a', newline='') as csvfile:
        writer = csv.writer(csvfile)
        for country, count in top_countries:
            percentage = (count / total_holidays) * 100
            writer.writerow([f"From the European Holidays API, {percentage:.1f}% of holidays are in {country}"])

    countries = [country for country, count in top_countries]
    holiday_counts = [count for country, count in top_countries]
    plt.figure(figsize=(8, 8))
    plt.pie(holiday_counts, labels=countries, autopct='%1.1f%%', startangle=140)
    plt.title('Percentage of Holidays by Country')
    plt.axis('equal')
    plt.show()
```



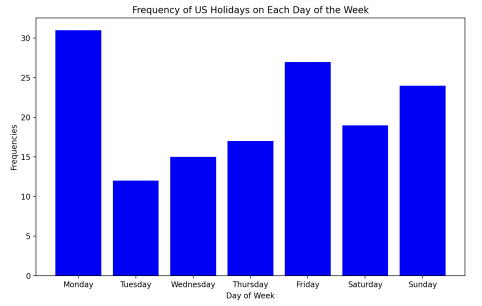
This function pulls data from the european_holidays and european_countrys tables and creates a pie chart to show the 12 most frequented countries in the database, as well as a section that includes all other countries.

```
def get_countries_by_region(db):
    regions = {
        'Northern Europe': ['Norway', 'Sweden', 'Denmark', 'Finland', 'Iceland', 'Estonia', 'Latvia', 'Lithuania'],
        'Western Europe': ['United Kingdom', 'Ireland', 'Netherlands', 'Belgium', 'Luxembourg', 'France', 'Germany'],
        'Southern Europe': ['Spain', 'Portugal', 'Italy', 'Greece', 'Malta', 'Andorra', 'San Marino', 'Vatican City', 'Monaco'],
        'Central Europe': ['Austria', 'Switzerland', 'Liechtenstein', 'Czechia', 'Slovakia', 'Poland', 'Hungary', 'Slovenia', 'Croatia'],
        'Eastern Europe': ['Russia', 'Belarus', 'Ukraine', 'Moldova', 'Romania', 'Bulgaria', 'Serbia', 'Montenegro', 'Bosnia and Herzegovina', 'Nort
    }

    region_counts = {region: len(countries) for region, countries in regions.items()}

    with open('holiday_data.csv', 'a', newline='') as csvfile:
        writer = csv.writer(csvfile)
        for region, count in region_counts.items():
            writer.writerow([f"From the European Holidays API, {count} countries are in {region}"])

    plt.figure(figsize=(10, 8))
    plt.pie(region_counts.values(), labels=region_counts.keys(), autopct='%1.1f%%', startangle=140)
    plt.title('Distribution of European Countries by Region')
    plt.axis('equal')
    plt.show()
```

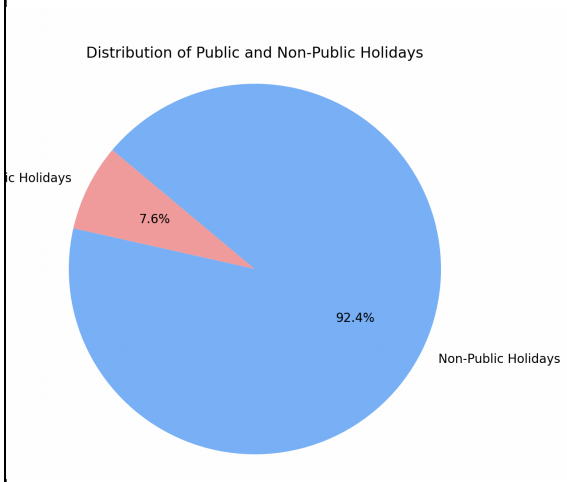
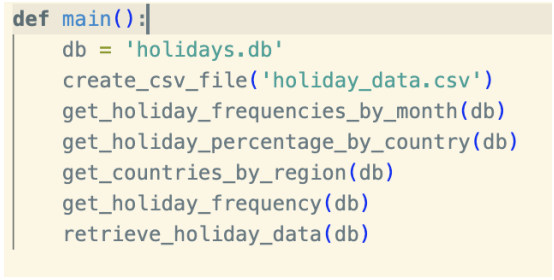


Assuming no new countries will be added to the database, this visualization categorizes the existing countries into subregions in Europe and then creates a pie chart to show how represented each European region is.

```
def get_holiday_frequency(db):
    conn = sqlite3.connect(db)
    c = conn.cursor()
    holiday_counts = {}
    info = c.execute('''SELECT W.weekday, COUNT(USH.id) as ush_count
                        FROM Weekday W
                        JOIN US_Holidays USH ON W.id = USH.weekday_id
                        GROUP BY W.id''').fetchall()
    for weekday, count in info:
        holiday_counts[weekday] = count
    with open('holiday_data.csv', 'a', newline='') as csvfile:
        writer = csv.writer(csvfile)
        for weekday, count in holiday_counts.items():
            writer.writerow([f"From the US Holidays API, {count} holidays are on {weekday}"])
    plt.figure(figsize=(10, 6))
    plt.bar(list(holiday_counts.keys()), list(holiday_counts.values()), color='blue')
    plt.xlabel('Day of Week')
    plt.ylabel('Frequencies')
    plt.title('Frequency of US Holidays on Each Day of the Week')
    plt.show()
```



This function connects to the database, executes an SQL query to retrieve the count of US holidays per weekday. It joins the Weekday table and US holiday table on the weekday_id column.
It also writes the data to a csv file, and calculates the amount of holidays that happen on each weekday.
It then generates a bar plot using Matplotlib to visualize the frequency of US holidays.

		This function connects to the database and executes a SQL query to count the number of public US holidays in the US holiday table. Then it calculates the proportion of ‘public’ to ‘non-public’ holidays and writes the data to a CSV file. Then the function creates a pie chart using Matplotlib, with the sectors corresponding to public and non-public holidays.
		Calls the csv_file to create and calls all the visualizations.

Resources Used

Date	Issue Description	Location of Resource	Result (Did it solve the issue?)
4/22	We were trying to join two tables that both reference the days of the week to create an overall visualization of the frequency of holidays based on the day of the week.	https://www.w3schools.com/sql/sql_join.asp	Yes, it helped me better understand how a database JOIN works. However, I wasn't able to properly
4/22	Struggled to understand the 25 item limit, eventually found an example in the slides that used the break function.	**Slides from lecture**	Yes, it taught me to problem solve in a simple way rather than trying to find a new import.