1. The goals for your project (10 points)

In our project, we were interested in collecting data from the Met Museum of Art and the Chicago Institute of Art, specifically about artworks that are related to activism. We were especially interested in the years the most activism artworks were created, when the artworks were most popular, the mediums/object types, and where they originated from.

2. The goals that were achieved (10 points)

We were able to successfully select items according to a search query “activism” and get qualities of these items such as artist, end date, medium, etc. We were able to translate the museum identification number for each item from a multi-digit number to a single-digit number that aligned with our samples of 100 items. We were able to join our two database tables on this identification number and therefore have clearer, more organized data. The data we collected from both APIs were able to help us answer our questions about time periods and origins.

3. The problems that you faced (10 points)

We had to completely change gears on our project because the Apple Music API had a paywall. We attempted to use Chartmetric API, but even after consulting the professor and GSIs we were unable to get past its authentication system. Therefore, we completely started fresh, switching topics from music to art. This led us to lose a bit of time on the project, but we were able to use these APIs with much more ease. We also had a bit of a hiccup when we created a caching system to store our data from the APIs, rather than just inserting it directly into the database. This also resulted in wasted time and having to rework code that worked correctly, but just did not fit the assignment’s expectations. Lastly, we had some trouble coordinating the information between the two different APIs, since they had different abilities. For example, Sarah’s API did not have a “limit” parameter, so it took a bit of time to come up with a system that limited the amount of requests we were taking and putting into the database. Finally, our data included a lot of repeating strings since we had so many categories that we were interested in investigating. We faced challenges in eliminating these repeating strings so the data would be clean.

4. Your file that contains the calculations from the data in the database (10 points)

File name: calculations.txt

The average year with the most highlighted 'activism' pieces was 1569
The total number of highlighted 'activism' items in this sample of 100 'activism' items was 56
The object type id with the most highlighted 'activism' pieces was 3, with 25 highlighted pieces.
Of the object type with the most highlighted 'activism' pieces, the most used medium was Oil on wood.

List of years that activism artwork was made in each century (chicago)
{"1600": ["1659", "1699", "1668", "1690", "1600", "1614", "1600", "1600", "1685", "1647", "1647"], "1500": ["1533", "1505", "1517", "1580", "1510", "1505", "1505", "1533", "1553", "1550", "1535", "1530", "1579", "1563", "1570", "1535"], "1800": ["1890", "1802", "1891", "1891", "1891", "1800",

"1800", "1807", "1861", "1850", "1825", "1820", "1870", "1870"], "1700": ["1767", "1777", "1771", "1719", "1799", "1746", "1744", "1710", "1793", "1779", "1784", "1778", "1719", "1720"], "1900": ["1951", "1931", "1906", "1905", "1959", "1964", "1904", "1921", "1900", "1901", "1900", "1909", "1906", "1909", "1912", "1927", "1922", "1919", "1915", "1921", "1930", "1935", "1910", "1952", "1939", "1929", "1949", "1947", "1952", "1968"], "1400": ["1475", "1487", "1495", "1400", "1425", "1485", "1495", "1433"], "1300": ["1355", "1375"], "1200": ["1265"], "2000": ["2020", "2020", "2020"]}

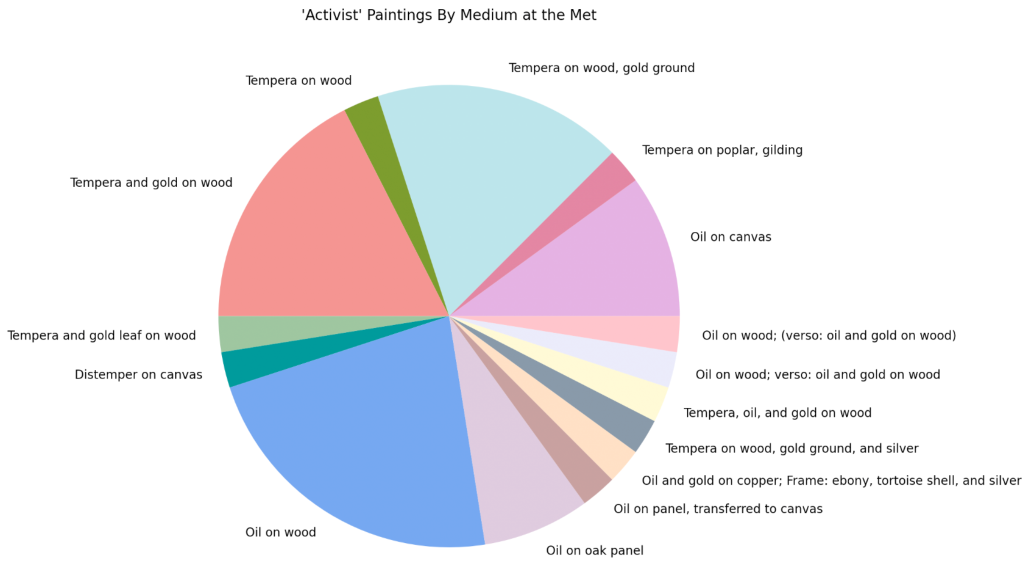
The count of years activism artwork was made in each century (chicago)

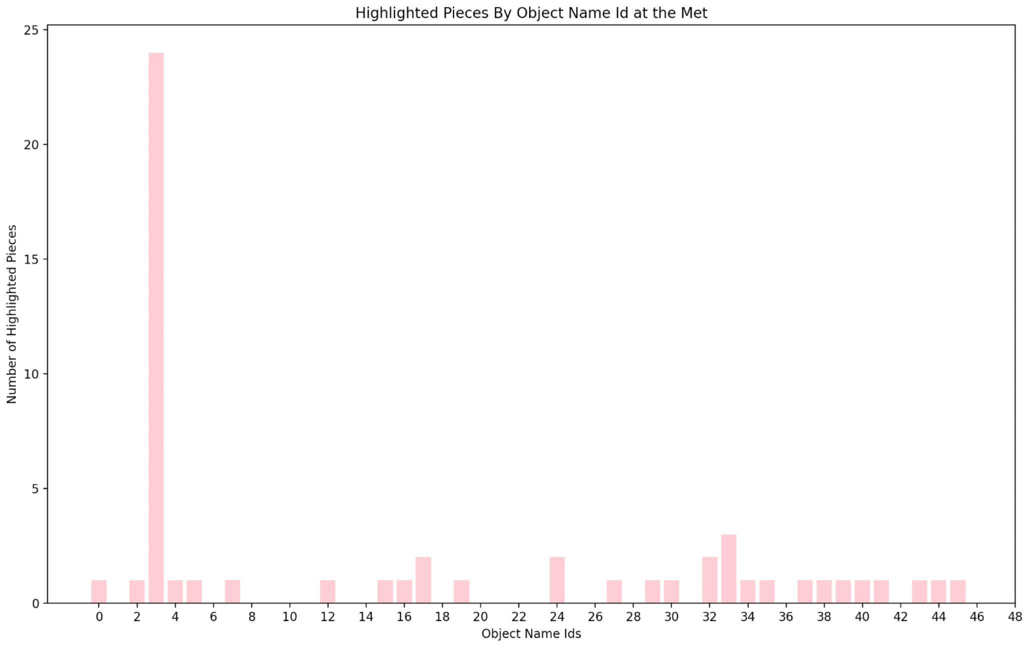
{"1200": 1, "1300": 2, "1400": 8, "1500": 16, "1600": 11, "1700": 14, "1800": 14, "1900": 30, "2000": 3}

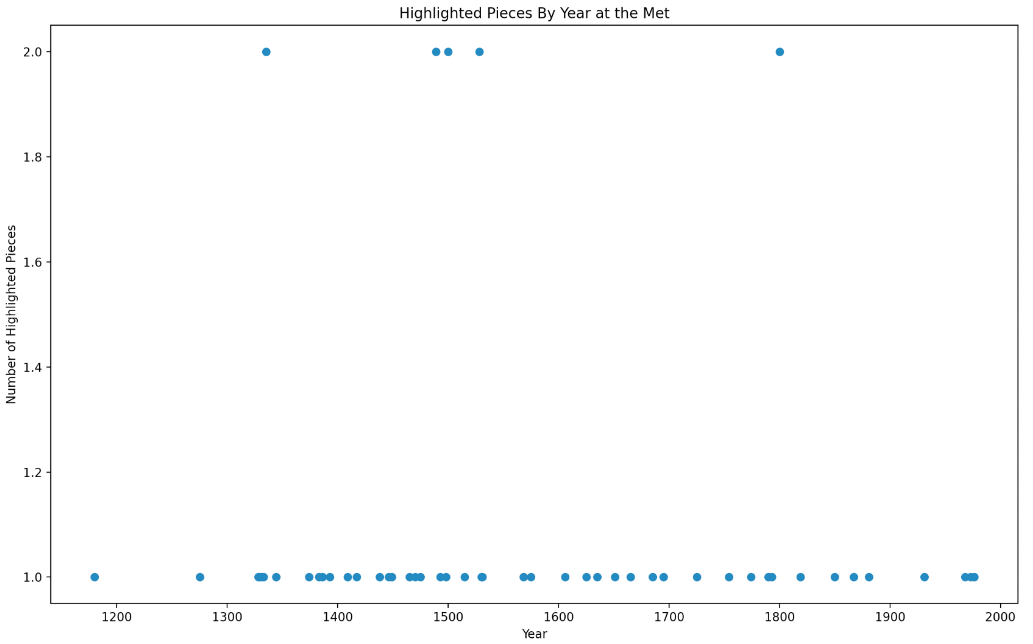
The count of activism artworks that were created in each country (chicago)

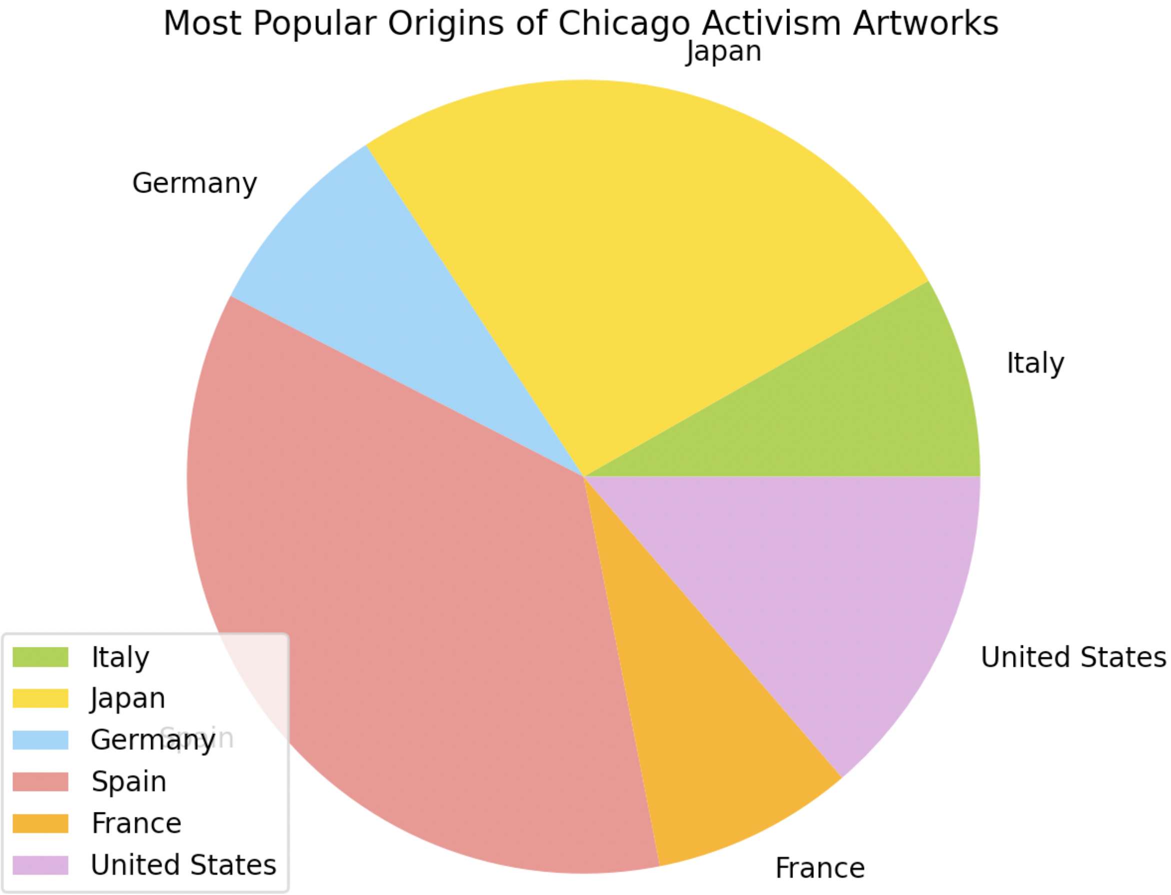
{"Netherlands": 3, "Italy": 6, "Japan": 19, "Denmark": 1, "Germany": 6, "Spain": 26, "France": 6, "Venice": 2, "England": 2, "United States": 10, "Austria": 2, "Bruges": 1, "Switzerland": 1, "India": 1, "United Kingdom": 1, "Burgos": 1, "Philadelphia": 1, "Jouy-en-Josas": 1, "Flanders": 2, "Augsburg": 1, "Unknown Place": 1, "China": 2}

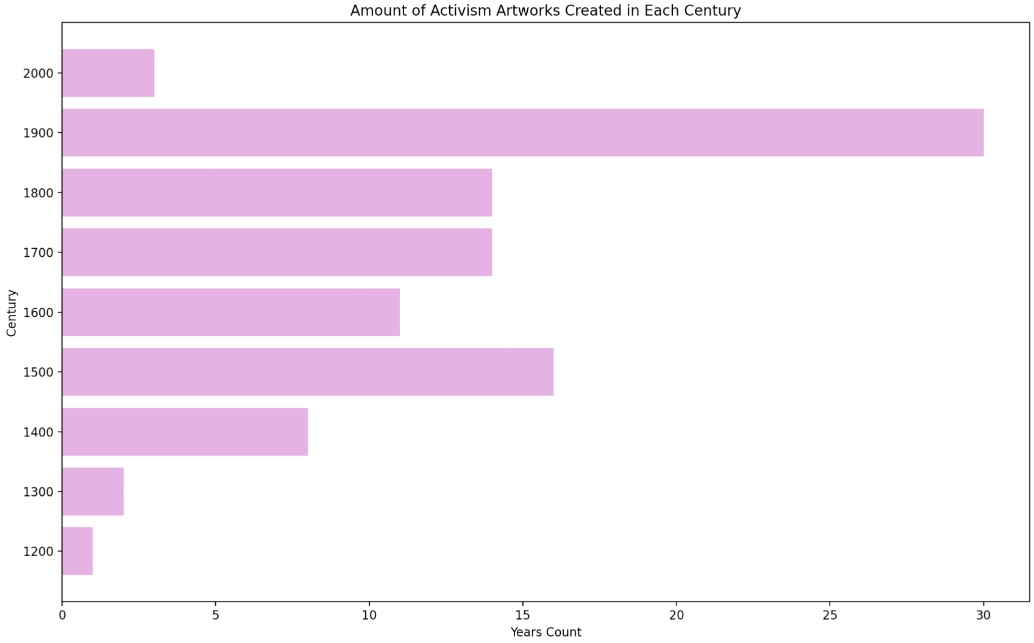
5. The visualization that you created (i.e. screen shot or image file) (10 points)




Highlighted Pieces By Object Name Id at the Met
Number of Highlighted Pieces
0
5
10
15
20
25
Object Name Ids
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48


Highlighted Pieces By Year at the Met
Number of Highlighted Pieces
1.0
1.2
1.4
1.6
1.8
2.0
Year
1200
1300
1400
1500
1600
1700
1800
1900
2000


Most Popular Origins of Chicago Activism Artworks


Amount of Activism Artworks Created in Each Century

6. Instructions for running your code (10 points)
All of our functions run under main(). Our data is in a database in SQLite and our file is a text file. Our plots will show automatically. You should only have to click run one time.

7. Documentation for each function that you wrote. This includes the input and output for each function (20 points)
`met_get_ids(cur, conn, query)`- This function takes in a cursor object, a connection object, and a search query. It gets a list of objectIDs off of the Met website based on the passed query "activism." This function returns the list of object IDs. It also creates a database table called object_ids in which we convert the Met's multi-digit object IDs into single-digit IDs from 1-100 to better organize our data.

`met_add_to_database(cur, conn, query, start, end)`- This function takes in a cursor object, a connection object, a search query,, a starting value, and an ending value indicating where in the list of object IDs to start and stop adding to the database. This function creates the table met_objects with various columns holding information about each item, including the Met's assigned object ID, title, artist name, when the object was completed (end date), object name, medium, and if the object is highlighted at the museum. This function calls the met_get_ids function and loops through the returned list of object IDs, making a request to the API for each ID in the given start to end period indicated. It then selects data from the response to be added to the database in each respective column created in the met_objects table.

`met_create_name_table(cur, conn)` - This function takes in a cursor object and a connection object. It selects the objectname column from the met_objects table and creates a list called no_repeats_names with each unique string in the objectname column. It then creates the table met_names and assigns each unique objectname string to a name_id to eliminate repeating strings.

`met_create_medium_table(cur, conn)` - This function takes in a cursor object and a connection object. It selects the medium column from the met_objects table and creates a list called no_repeats_mediums with each unique string in the medium column. It then creates the table met_mediums and assigns each unique medium string to a medium_id to eliminate repeating strings.

`met_create_artist_table(cur, conn)` - This function takes in a cursor object and a connection object. It selects the artist_name column from the met_objects table and creates a list called no_repeats_artists with each unique string in the artist_name column. It then creates the table met_artists and assigns each unique artist name string to an artist_id to eliminate repeating strings.

`met_dates_and_highlights(cur, conn, file)` - This function takes in a cursor object, a connection object, and a .txt file name for the calculations to be written to. This function selects data from the database, writes calculations to a file, and creates a data visualization. It selects the date the object was created (object_enddate) and if the object is highlighted at the museum (is_highlight) on the condition that the object **is highlighted**. It then creates a dictionary called highlight_counts of the number of highlighted objects for each year. It also counts the number of highlighted pieces selected, as well as the average year with the most highlighted pieces. It then creates a scatter plot to visualize the number of highlighted pieces by year at the Met.

`met_names_and_highlights(cur, conn, file)` - This function takes in a cursor object, a connection object, and a .txt file name for the calculations to be written to. This function selects data from the database, writes calculations to a file, and creates a data visualization. It selects the name id of the object from the met_names table and if the object is highlighted at the museum (is_highlight) on the condition that the object **is highlighted**, using a database join on met_objects and met_names where met_objects.objectname = met_names.name_id. It then creates a dictionary called object_counts of the number of highlighted objects for each object name id. It shows the object name id with the most highlighted pieces. It then creates a bar chart to visualize the number of highlighted pieces by object name at the Met.

`met_extra_credit_viz(cur, conn, file)` - This function takes in a cursor object, a connection object, and a .txt file name for the calculations to be written to. This function selects data from the database, writes calculations to a file, and creates a data visualization. It selects the object name id of the object from the met_objects table and the medium from the met_mediums table, using a database join on met_objects and met_mediums where met_objects.medium = met_mediums.medium_id. It then creates a dictionary called medium_counts of the number of times a medium was used on the most highlighted object name (as calculated from the previous function: paintings, name_id = 3). It shows the most used medium for this object name. It then creates a pie chart to visualize the activist paintings by medium at the Met.

`met_update_table(cur, conn)` - This function takes in a cursor object and a connection object. It updates the database file to eliminate any repeating strings. It does so by selecting the id and string from each respective name, artist, and medium tables. It creates a list of each unique string and updates the main table met_objects with the respective id instead of string to eliminate the repeating strings.

`def chi_get_ids(cur, conn, query, limit = 100)` - This function takes in a cursor object, a connection object, a query string for the api, and a limit (we take a hundred objects so I set the default to 100). The function retrieves 100 api object ids for artworks with the given keyword. The output is a list of 100 object ids.

`chi_add_to_database(cur, conn, query, db_filename, start, end)` - This function takes in a cursor object, a connection object, a query string for the request url, the file name it is added to, and the start and end for indexing the object id list returned by the chi_get_ids function. This function creates the table in the database which includes the following information: object_id, title, artist_name, object_enddate, medium, origin, and popularity. It then uses the query and start/end to index the object id list and use the id to retrieve the prior information about each artwork from the api. It then inserts this information into the table. There is no output.

`chi_no_repeats(cur, conn, query)` - This function takes in a cursor object, a connection object, and a query string. For every item in the object id list, it retrieves the repeated-string information (artist name, medium, and origin) from the api and replaces it with the new keys that are made from the new tables in the following code. There is no output.

`chi_create_name_table(cur, conn)` - This function takes in a cursor object and a connection object. It selects all artist names from the original database and puts them into a list. It creates a new table in the database for artist names and loops through the list to add each name with a new key. There is no output.

`chi_create_medium_table(cur, conn)` - This function takes in a cursor object and a connection object. It selects all mediums from the original database and puts them into a list. It creates a new table in the database for mediums and loops through the list to add each medium with a new key. There is no output.

`chi_create_origin_table(cur, conn)` - This function takes in a cursor object and a connection object. It selects all origins from the original database and puts them into a list. It creates a new table in the database for origins and loops through the list to add each country with a new key. There is no output.

`chi_century_years(cur, conn)` - This function takes in a cursor object and a connection object. It selects all end dates from the chicago objects in the database and puts them in a list. A for loop loops through the list and uses regex to organize each year into a dictionary where the key is the century and the value is a list of the years in that century. It returns this dictionary.

`chi_century_counts(cur, conn)` - This function takes in a cursor object and a connection object. It calls chi_century_years() to retrieve its returned dictionary, then creates a new dictionary where the key is the century and the values are the counts of years. It returns this dictionary sorted by century order.

`chi_plot_century_count(cur, conn) -` This function takes in a cursor object and a connection object. It calls chi_century_counts() to get the dictionary on counts and uses the keys and values to create a horizontal bar chart that displays the Amount of Activism Artworks Created in Each Century. It shows this plot.

`chi_origin_counts(cur, conn) -` This function takes in a cursor object and a connection object. It selects the origin types (string form) from the chicago_origins table and joins it with the new keys from the chicago_objects table. It then loops through this list and creates a dictionary where the keys are the countries and the values are counts of artworks that originated there. It returns this dictionary.

`chi_plot_origin_count(cur, conn) -` This function takes in a cursor object and a connection object. It calls chi_origin_counts() to retrieve that dictionary and makes a new dictionary with countries that have five or more artworks (to find the most popular collection of countries). From this new dictionary, it creates lists of the keys and values. It then plots the data on a pie chart with a legend and shows this.

`chi_write_file(cur, conn) -` This function takes in a cursor object and a connection object. It calls chi_century_years(), chi_century_counts(), chi_origin_counts(). It opens a text file “calculations.txt” and writes this data using strings for explanations and json.dumps for the dictionaries. There is no output.

`main() -` Our main function calls all of these functions in the appropriate order so it only needs to be run once.

8. You must also clearly document all resources you used.

Date Issue Description Location of Resource Result

4-18, Needed help figuring out how to add to the end of a file without erasing the whole thing, https://www.geeksforgeeks.org/python-append-to-a-file/, it helped me successfully use ‘a’ in appending to a file

4-20, Needed help figuring out how to create a scatter plot, https://www.w3schools.com/python/matplotlib_scatter.asp, helped me figure out the parameters successfully as well as how to make different colors!

4-21 needed help with different color names for my pie chart https://www.w3schools.com/colors/colors_names.asp helped me figure out color names to use

4-15 every time I ran into an issue with the chicago api I looked through the documentation website (there is a lot of information here) https://api.artic.edu/docs/#introduction

4-20 writing a dictionary to a file https://www.geeksforgeeks.org/write-a-dictionary-to-a-file-in-python/ helped me with the format of writing files