Jenna Li, Odiso Obiora, Audrey Tang (Team JOA)

21 April 2023

SI 206 Data-Oriented Programming

Project Name: Comparing Art Museum Records

Repository: https://github.com/Jenna9192/Final-Project.git

1. Original Goals:

Originally, we wanted to work on the API's from the Art Institute of Chicago, the MET, the Brooklyn Museum and the Harvard Art Museum. We wanted to find the data on the most popular art medium, most recorded time periods for artwork, the most and least common cultures represented, artwork on display vs not on display, the oldest and newest piece of artwork at each museum, and the most prolific gallery. We wanted to compare each museum to one another but also find the most common types of mediums, periods, cultures, etc.

2. Achieved Goals:

Currently, we have worked on gathering data from the MET API, Harvard Art Museum API, and Wikipedia page for selected artworks in the MET Museum. We gathered information on the medium, culture, and period. To identify each unique artwork, we identified them by their object_id given by their museums and created our own ids for inputting into the database. We also gathered additional information on the year made and title when the information was available.

3. Problems Faced:

We faced multiple problems during this project. One problem we faced when we first began working on this project was that not all APIs contain the same information. Also, the APIs we chose lacked the information that we were looking for (i.e,. Display or not on display,

location or origin, etc.), either entirely or in certain artworks. Some artworks were missing information which we later filed as “N/A.” In addition, some APIs were harder to understand or access than others. The Art Institute of Chicago API had multiple URLs to access before accessing the information on individual artworks that had incomplete information. Therefore, we have changed our sources to only focus on the MET API, Harvard API, and web scraping the MET Wikipedia page for information since these sources were easier to access. We also decided to only get information available across sources which was the medium, culture, and period of each artwork.

We also had difficulty uploading only 25 pieces of data into a table at a time. To solve this problem, we decided that we would first check if a table exists. If it does not exist, we would run the function such that only 25 pieces of data are uploaded to the table (ex. make_met_data). If it does exist, it would check the last index of the current data in the table and start appending an additional 25 pieces of data from the JSON or CSV file starting from that index. If the table has all the data, the next time the file is run, the table would automatically delete all data in the table and restart uploading 25 pieces of data.

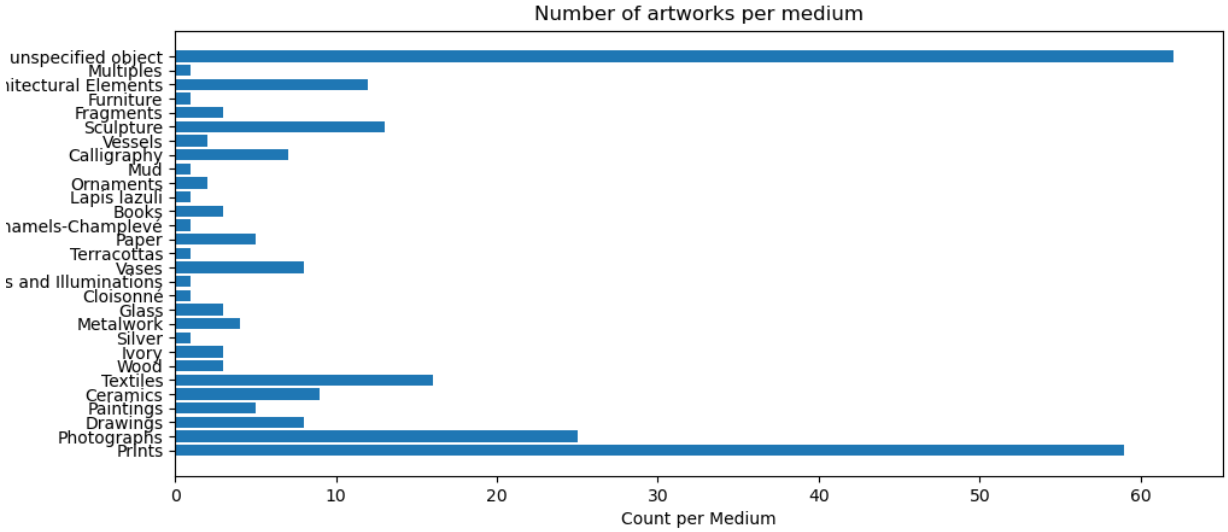
4. Calculations:

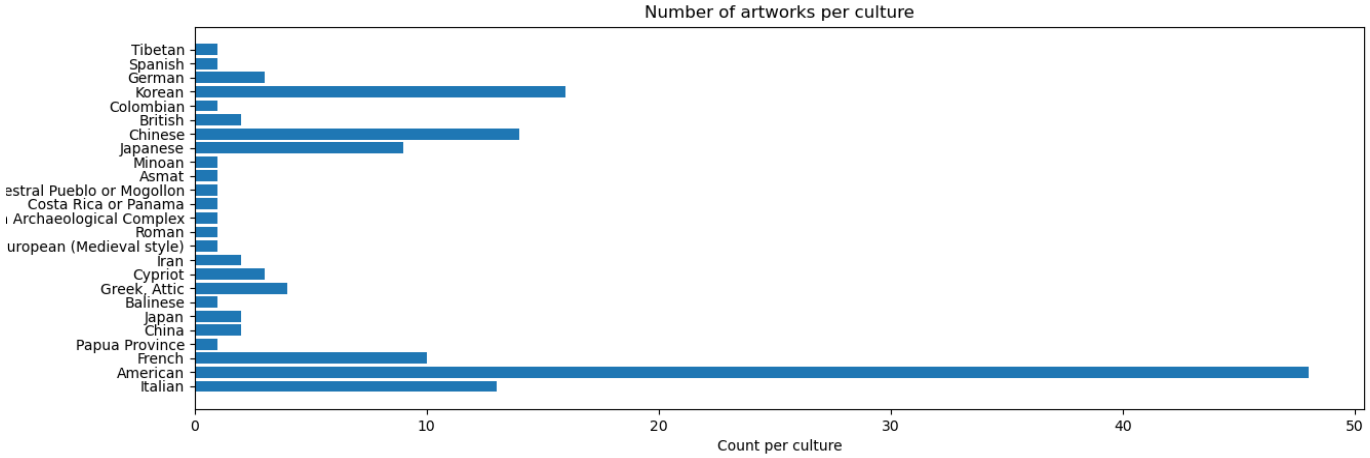
We calculated the sum of artworks in each period, medium, and culture and saved them into a JSON file. Below are screenshots from the file.

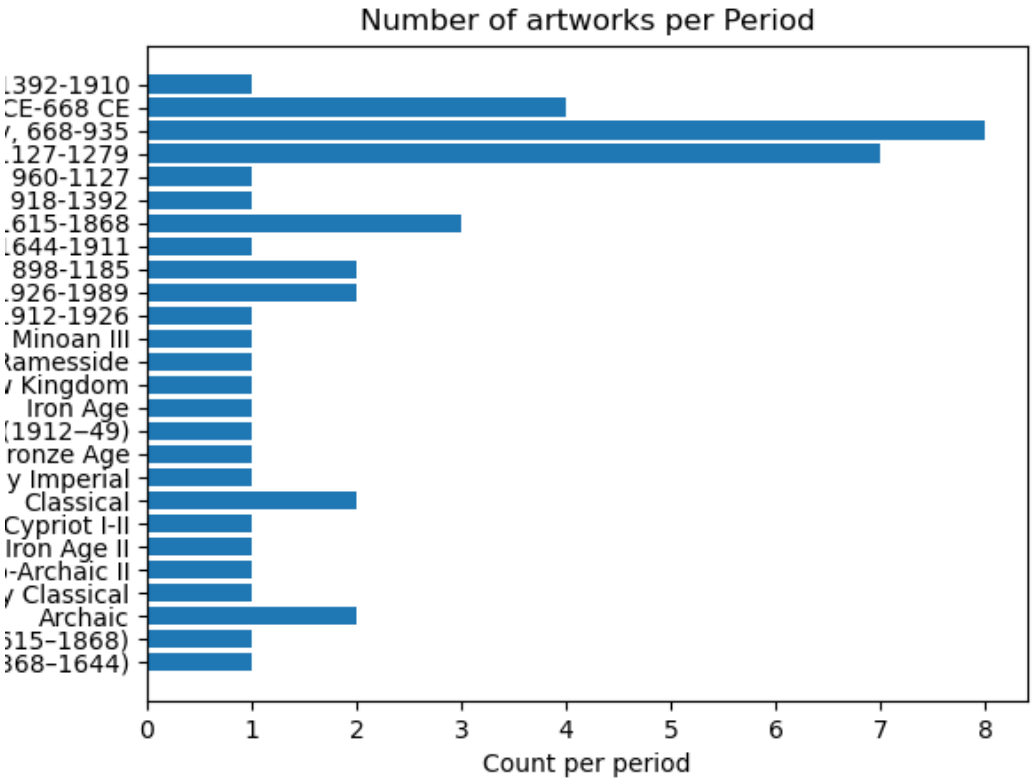
```
{
    "medium_sum_data": {
        "Prints": 32,
        "Photographs": 11,
        "Drawings": 7,
        "Paintings": 3,
        "Ceramics": 9,
        "Textiles": 16,
        "Wood": 3,
        "N/A": 1,
        "Ivory": 3,
        "Silver": 1,
        "Metalwork": 4,
        "Glass": 3,
        "Cloisonn\u00e9": 1,
        "Manuscripts and Illuminations": 1,
        "Vases": 8,
        "Terracottas": 1,
        "Paper": 5,
        "Enamels-Champlev\u00e9": 1,
        "Books": 3,
        "Lapis lazuli": 1,
        "Ornaments": 2,
        "Mud": 1,
        "Calligraphy": 7,
        "Vessels": 1,
        "Sculpture": 0,
        "Fragments": 0,
        "Furniture": 0,
        "Architectural Elements": 0,
        "Multiples": 0,
        "unspecified object": 62
    },
},
"culture_sum_data": {
    "Italian": 2,
    "American": 21,
    "French": 8,
    "N/A": 60,
    "Papua Province": 1,
    "China": 2,
    "Japan": 2,
    "Balinese": 1,
    "Greek, Attic": 4,
    "Cypriot": 3,
    "Iran": 2,
    "European (Medieval style)": 1,
    "Roman": 1,
    "Bactria-Margiana Archaeological Complex": 1,
    "Costa Rica or Panama": 1,
    "Ancestral Pueblo or Mogollon": 1,
    "Asmat": 1,
    "Minoan": 1,
    "Japanese": 4,
    "Chinese": 6,
    "British": 1,
    "Colombian": 1,
    "Korean": 0,
    "German": 0,
    "Spanish": 0,
    "Tibetan": 0,
    "Austrian?": 0,
    "Bohemian": 0,
    "American?": 0,
    "French?": 0,
    "null": 0,
    "Pre-Columbian?": 0,
    "Italian, Lombard, Milanese": 0,
    "Italian, Umbrian": 0,
    "Indian": 0
},
"period_sum_data": {
    "N/A": 83,
    "Ming dynasty (1368\u20131644)": 1,
    "Edo period (1615\u20131868)": 1,
    "Archaic": 2,
    "Early Classical": 1,
    "Cypro-Archaic II": 1,
    "Iron Age II": 1,
    "Late Cypriot I-II": 1,
    "Classical": 2,
    "Early Imperial": 1,
    "Bronze Age": 1,
    "Republic period (1912\u201249)": 1,
    "Iron Age": 1,
    "New Kingdom": 1,
    " Ramesside": 1,
    "Late Minoan III": 1,
    "null": 0,
    "Taish\u014d era, 1912-1926": 1,
    "Sh\u014dwa era, 1926-1989": 1,
    "Heian period, Late, 898-1185": 2,
    "Qing dynasty, 1644-1911": 1,
    "Edo period, 1615-1868": 0,
    "Kory\u014f dynasty, 918-1392": 0,
    "Song dynasty, Northern Song period, 960-1127": 0,
    "Song dynasty, Southern Song period, 1127-1279": 0,
    "Unified Silla dynasty, 668-935": 0,
    "Three Kingdoms period, Silla, 57 BCE-668 CE": 0,
    "Chos\u014fn dynasty, 1392-1910": 0
},
```

5. Visualizations:

We created three bar graph visualizations from the calculations above.







6. Instructions:

    1. Run met_museum.py four times to get 100 pieces of data into met_database

    2. Run Harvard Art Museums.py four times to get 100 pieces of data into Harvard_data

    3. Run met_wikipedia.py four times to get 65 pieces of data for selected_works and 100 pieces of data for citations

    4. Run calculations.py for the calculations and visualizations.

7. Documentation:

met_museum.py

Function Name	Input	Output
`def` [redacted]	`filename - name of JSON file to load`	`Returns dictionary Loads a JSON cache from filename`

`load_json(filename):`		if it exists
`def write_json(filename, dict)`	filename - name of JSON file to write to dict - the data(dictionary) to write into the data	Returns None Encodes dict into JSON format and writes the JSON to filename to save the search results
`def get_api_info(url, params=None):`	url - url of API params - optional dictionary of query string arguments (default value is 'None').	Returns dictionary of decoded JSON from successful requests If the request is not successful, an empty dictionary will return with “exceptions” being printed
`def cache_all_pages(artifact_url, filename):`	artifact_url - url of API filename - name of JSON file to write to	Returns None 1. Finds 100 random object id to collect data from 2. Add data to dictionary and write out dictionary into file(key is index and value is result from request)
`def setUpDatabase(db_name):`	db_name - database file to connect to	Returns cursor and conn Creates cursor and conn for the database
`def make_period_data(data, cur, conn):`	data - JSON data cur - database cursor conn - database connection object	Returns None Creates a separate table for the artwork periods 1. Iterates through the period of each artwork in data 2. If period is not in periods list, appends period to periods list 3. Loads periods into table called met_periods
`def make_medium_data(data, cur, conn):`	data - JSON data cur - database cursor conn - database connection object	Returns None Creates a separate table for the artwork mediums 1. Iterates through the medium of each artwork in data 2. If medium is not in mediums list, appends medium to medium list 3. Loads mediums into table called met_mediums
`def make_culture_data(data, cur, conn):`	data - JSON data cur - database cursor conn - database connection object	Returns None Creates a separate table for the artwork periods 1. Iterates through the culture of each artwork in data 2. If culture is not in cultures list, appends culture to cultures list 3. Loads cultures into table called met_cultures

`def make_location_data(data, cur, conn):`	data - JSON data cur - database cursor conn - database connection object	Returns None Creates a separate table for the artwork periods 1. Iterates through the region, subregion, country, state, city of each artwork in data for artwork origin 2. If location is not in locations list, appends location to locations list 3. Loads locations into table called met_locations
`def make_met_data(data, cur, conn, index):`	data - JSON data cur - database cursor conn - database connection object Index - starting index to iterate from	Returns None Uploads 25 consecutive data from the JSON file into all_database each time starting at the id that matches with the inputted index
`def main():`	None	Returns None Executes all functions 1. Loads JSON file 2. Translate 100 pieces of API data into JSON 3. Set up database 4. Create separate tables for medium, culture, period, and location 5. Creates database for all data in JSON file with only 25 iterations each time

Harvard Art Museums.py

Function Name	**Input**	**Output**
def **load_json**(filename):	**filename** (string)	Returns dictionary Loads a JSON cache from filename if it exists
def **write_json**(filename, dict):	**filename** (string) **dict** (dictionary)	Returns None Encodes dict into JSON format and writes the JSON to filename to save the search results
def **get_api_info**(url):	**url** (string)	Returns dictionary of decoded JSON from successful requests If the request is not successful, an empty dictionary will return with “exceptions” being printed
def **cache_pages**(url, filename):	**url** (string) **filename** (string)	Returns None Retrieves individual pages from url and makes cache where each dictionary k,v is a separate object

def **create_database**(db_name):	**db_name** (string)	Returns cursor and connection objects Establishes connection to database with filename "db_name".
def **create_museum_table**(cur, conn):	**cur** (cursor object) **conn** (connection object)	Returns None Creates museum table from dict: {1:"The Metropolitan Museum of Art", 2: "Harvard Art Museums"}
def **create_medium_data**(data,cur,conn):	**data** (dictionary) **cur** (cursor object) **conn** (connection object)	Returns None Creates Harvard_mediums table with columns "id", "medium"
def **create_culture_data**(data, cur,conn):	**data** (dictionary) **cur** (cursor object) **conn** (connection object)	Returns None Creates Harvard_cultures table with columns "id", "culture"
def **create_period_data**(data,cur,conn):	**data** (dictionary) **cur** (cursor object) **conn** (connection object)	Returns None Creates Harvard_periods table with columns "id", "period"
def **create_harvard_full_data**(data,cur,conn,index):	**data** (dictionary) **cur** (cursor object) **conn** (connection object)	Returns None Creates Harvard_data table featuring "id", "museum id", "object id", "medium", "culture", "period"
def **main**():	None	Returns None Executes all functions 6. Loads JSON file 7. Translates API data into JSON 8. Sets up database 9. Create separate tables for museum id, medium, culture, period, and century 10. Creates database for all data in JSON file with only 25 new rows each time

met_wikipedia.py

Function Name	**Input**	**Output**
make_request(url)	**url**(string)	Retrieves website content. Returns beautifulsoup object Or Returns None and prints error code

get_selected_objects(soup)	**soup** (bs4 object)	Parses through website HTML to find title, artist, year, link information for selected objects. Returns a list of lists for each object: [[title 1, museum_Id 1, artist 1, year 1, object_Type 1, image 1], [title 2, museum_Id 2, artist 2, year 2, object_Type 2, image 2] ...]
get_selected_paintings(soup)	**soup** (bs4 object)	Parses through website HTML to find title, artist, year, link information for selected paintings. Returns a list of lists for each painting: [[title 1, museum_Id 1, artist 1, year 1, object_Type 1, image 1], [title 2, museum_Id 2, artist 2, year 2, object_Type 2, image 2] ...]
get_citations(soup)	**soup** (bs4 object)	Parses through website HTML to find title, website, date, link information for all citations. Returns a list of lists for each citation: [[title 1, website 1, date 1, link 1], [title 2, website 2, date 2, link 2]...]
create_csv(data, heading, filename)	**data** (list of lists) **heading** (list of strings) **filename** (string)	Adds indexing to data and writes csv file with filename containing the list of lists with input header. Returns None
setUpDatabase(db_name)	**db_name** (string)	Establishes connection to database with filename “db_name”. Returns cursor and connection object
extract_data(file)	**file** (string for csv filename)	Takes the data from csv and converts it to an indexing-friendly format. Returns a list of tuples
create_objType_data(cur, conn)	**cur** (cursor object) **conn** (connection object)	Returns None Creates wiki_objType table from dict: {1:"object", 2:

		"paintings"}
create_artist_data(data, cur, conn)	**data** (list of tuples) **cur** (cursor object) **conn** (connection object)	Returns None Creates wiki_artists table with columns "id" and "artist"
create_works_data(file, cur, conn)	**data** (list of tuples) **cur** (cursor object) **conn** (connection object) **index** (int)	Uploads 25 pieces of data from the list of tuples to the selected_works table. Returns None
create_citations_data(file, cur, conn)	**file** (string for csv filename) **cur** (cursor object) **conn** (connection object) **index** (int)	Uploads 25 pieces of data from the list of tuples to the citations table. Returns None
main()	**None**	Executes all functions: 1. Requests content from MET Wikipedia page. 2. Extract selected objects and selected paintings information from website. 3. Combine the two into a master list. 4. Creates a CSV file from the master list. 5. Extract citations information from website 6. Creates a CSV file for the citations. 7. Opens database to work. 8. Converts the csv files to a workable format. 9. Creates table for selected works with only 25 iterations each time. 10. Creates table for citations with only 25 iterations at a time.

calculations.py

Function Name	Input	Output
open_database(db_name)	**db_name** (string)	Establishes connection to database with filename "db_name". Returns cursor and connection object
write_json(filename, dict)	**filename** (string)	Encodes dict into JSON format and

	dict (dictionary)	writes the JSON to filename to save the search results Return None
calc_mediums(db_name, cur, conn)	**db_name** (string) **cur** (cursor object) **conn** (connection object)	Compiles cross-listed mediums in Harvard, MET, and MET wiki databases into a list, then appends unique mediums to list. Calculates sum of each medium. Returns a dictionary with key-value pair: {medium : sum}
calc_period(db_name, cur, conn)	**db_name** (string) **cur** (cursor object) **conn** (connection object)	Compiles cross-listed periods in Harvard and MET databases into a list, then appends unique periods to list. Calculates sum of each period. Returns a dictionary with key-value pair: {period : sum}
calc_culture(db_name, cur, conn)	**db_name** (string) **cur** (cursor object) **conn** (connection object)	Compiles cross-listed cultures in Harvard and MET databases into a list, then appends unique cultures to list. Calculates sum of each culture. Returns a dictionary with key-value pair: {culture : sum}
visual_medium(medium_dict)	**medium_dict** (dictionary)	Calls in dictionary outputed by `calc_mediums()` and creates bar graph from calculations. Returns None
visual_culture(culture_dict)	**culture_dict** (dictionary)	Calls in dictionary outputed by `calc_culture()` and creates bar graph from calculations. Returns None
visual_period(period_dict)	**period_dict** (dictionary)	Calls in dictionary outputed by `calc_period()` and creates bar graph from calculations. Returns None
main()	**None**	Executes all functions: 1. Runs calculations for medium, period, and culture. 2. Saves calculations to a nested dictionary. 3. Writes nested dictionary into a JSON file. 4. Creates 3 bar graphs for medium, culture, and period.

8. Resources:

Date	Issue Description	Location of Resources	Result (did it solve the issue?)
4/2/2023	Finding available APIs to use for the project	https://github.com/public-apis/public-apis	Yes, the resource helped us figure out the topic for our project(museums) and locate potential APIs that we could use. We looked through the available APIs to
4/18/2023	Learning to access the JSON file for each individual object_id	https://chat.openai.com/	Yes, this resource helped me access the information of each individual artwork in the Met museum by changing the params each time I try to access a different file containing a different artwork
4/17/2023	Accessing The Metropolitan Museum of Art Collection API	https://metmuseum.github.io/	This resource taught me how to access the information in the API for the MET museum and informed us about the details and information we can get from each artwork.
4/19/2023	Writing regular expressions to parse for specific HTML information	https://regex101.com/	This resource helped me test and iterate the regular expression needed to get specific information for my web scraping.
4/20/2023	How to create database from csv data	https://chat.openai.com/	This resource helped explain the steps to creating a database from a csv file.

9. References

Source	Base url	Documentation
The Metropolitan Museum of Art	https://collectionapi.metmuseum.org	https://metmuseum.github.io
Harvard Art Museum	https://api.harvardartmuseums.org	https://github.com/harvardartmuseums/api-docs

Wikipedia - “Metropolitan Museum of Art”	https://en.wikipedia.org/wiki/Metropolitan_Museum_of_Art	n/a