Project Report

Project Name: New York City's Crime Data in 2024

Contributors: Olivia LaCroix (T1), Swathi Danturi (T2)

Date: 12/10/2024

Authorship Information

- 1.2, 2.1, 2.2, 3.1, 3.2, 4.1, 4.2, 5.1, 5.2 - **Olivia**
- 1.1, 2.3, 2.4, 3.3, 3.4, 4.3, 4.4, 5.3, 4.4 - **Swathi**
- 6 - **Olivia** and **Swathi**

1. The Data

Source of the dataset used in project

```
- NYC Open Data
- https://data.cityofnewyork.us/Public-Safety/NYPD-Arrest-Data-Year-to-Date-/uip8-fykc/about_data
```

Dataset Description

```
- The data from NYC Open Data is about New York Police Department Arrest data in 2024.
- This dataset describes information about arrest records and the details of those arrests including but not limited to the location, level of offense, and perpetrator demographics.
```

1.1 Dataset Overview

General Information

- Dataset Name: NYPD Arrest Date
  - Owner: NYC Open Data
  - Date of last update: October 21, 2024

Dataset content

- Items: Arrest in New York City by the New York Police Department.
- Attributes: Attributes give more details about the arrest(s) records.
- Attributes used in the Dataset:
  - `ARREST_DATE` - exact date of arrest for the reported event

    - `PD_DESC` - description of internal classification corresponding with PD code (more granular than Offense description)
    - `LAW_CAT_CD` - level of offense: felony(F), misdemeanor(M) or violation(V)
    - `ARREST_BORO` - borough of arrest. B(Bronx), S(Staten Island), K(Brooklyn), M(Manhattan), Q(Queens)
    - `AGE_GROUP` - perpetrator's age within a category
    - `PERP_SEX` -perpetrator's sex description
    - `PERP_RACE` - perpetrator's race description
    - Attributes not used in the dataset:
    - `ARREST_KEY` - randomly generated ID number for each arrest
    - `PD_CD` - three digit classification code (more granular than key code)
    - `KY_CD` - three digit internal classification code (more general than PD code)
    - `OFNS_DESC` - description of internal classification corresponding to KY code (more general than PD description)
    - `LAW_CODE` - law code charges corresponding to penal law, and local laws
    - `ARREST_PRECINCT` - precinct where the arrest occurred
    - `JURISDICTION_CODE` - jurisdiction responsible for arrest. Jurisdiction codes 0(Patrol), 1(Transit) and 2(Housing) represent NYPD whilst codes 3 and more represent non NYPD jurisdictions
- Attributes removed from the dataset during preprocessing:
    - `X_COORD_CD` - midblock X-coordinate for New York State Plane Coordinate System, Long Island Zone, NAD 83, units feet (FIPS 3104)
    - `Y_COORD_CD` - midblock Y-coordinate for New York State Plane Coordinate System, Long Island Zone, NAD 83, units feet (FIPS 3104)
    - `Latitude` - latitude coordinate for Global Coordinate System, WGS 1984, decimal degrees (EPSG 4326)
    - `Longitude` - longitude coordinate for Global Coordinate System, WGS 1984, decimal degrees (EPSG 4326)
    - `New Georeferenced Column` - randomly generated geocoded column based on Latitude and Longitude fields
- Number of items: 195K
- Number of attributes: 19
- Timeframe the dataset cover: Jan 2024 - Oct 2024
- Size of the original file: 35.894 MB
- Size of the file used in the project: We used smaller sized files that contained 10, 50 and 100 records which varied by team member.

Intended use/purpose of the dataset

- Explore arrest data from police activity in New York city which helps identify recurring offenses in certain locations by certain demographic perpetrators.

1.2 Dataset Details

Data collection procedures

- The data was collected by The NYPD and reviewed by the Office of Management Analysis and Planning (OPA). It is manually extracted by the NYPD.

Representation

- The representation of the data is very clear and concise. It gives the person reading the data a clear understanding of all fields and why they are used.

Data quality and pre-processing

- There is no missing information that was intended to be used.
- The data is not out of date and is updated every quarterly. Last update was on October 21st 2024.
- Pre-processing is done to take out the following columns from the original dataset:
- X_COORD_CD, Y_COORD_CD, Latitude, Longitude, New Georeferenced Column.
- This was done to make the file smaller and easier to process since none of our investigative questions used these fields.

Privacy

- From the dataset, it is not possible to identify individuals as no information about the perpetrators is mentioned.
- No sensitive data like personal information, business information about any individual is mentioned in the dataset.
- There are no data confidentiality issues like disclosure of non-public or sensitive data and the data is intended for the public and there is no issue of privacy for this dataset.

2. The Questions

2.1 Question 1:

- **Which day of the year had the highest number of arrests and what was the most common race of the perpetrators on that day? (T1)**
- **Fields used** for method: ARREST_DATE, PERP_RACE
- **Relation** between the fields: The data relates to each other because a certain number of arrests are occurring every year, and to be able to target enforcement in a certain timeframe of the year, you want to identify the highest crime days. From there, you are able to target further by addressing the question of which race is most likely to commit the crimes during those times.
- **Characteristics** that might provide additional insights: Other data that would provide additional insights for this question would be if there was a field for `TIME`. Time would help narrow down the output even more by allowing law enforcement to identify not only the highest day of the year in arrests but also the most popular time to commit a crime.

2.2 Question 2

- **Which borough had the highest number of arrests for felony offenses and therefore what felony offense was most committed in that borough? (T1)**
- **Attributes used** for method: ARREST_BORO, LAW_CAT_CD, PD_DESC
- **Relation** between the fields: This data relates to each other because we want to find out the amount of felony offenses that are taking place in the year of 2024 and out of those felony arrests, we want to know which offense was most common to be able to find patterns in offenses that are occurring.

- **Characteristics** that might provide additional insights: Other data that would provide additional insights for this question would be if the suspect had a `PRIOR ARREST RECORD`. This information gathered would allow police to identify if there are systematic issues with bail reform and recurring crimes with the same suspect.

2.3 Question 3

- **Which age group committed the given crime (of 'Intoxicated and Impaired Driving') the most and what is the ratio of male to female in the crime?** `(T2)`
- **Attributes used** for method: `PD_DESC`, `AGE_GROUP`, `PERP_SEX`
- **Relation** between the fields: These fields are related because they help us understand the crime rate of `Intoxicated and Impaired Driving` committed by different `AGE_GROUP` and how it varies between males and females, `PERP_SEX`.
- **Characteristics** that might provide additional insights: A new field such as `Age` instead of `Age Group` would allow us to determine the crime rate according to a particular age instead of a range.

2.4 Question 4

- **For each borough, what is the average time between arrests in days for the (violence) given crime type according to the age group?** `(T2)`
- **Attributes used** for method: `ARREST_BORO`, `LAW_CAT_CODE`, `AGE_GROUP`, `ARREST_DATE`
- **Relation** between the fields: The fields are connected because they help us see where crime happens, how serious the crime is, the age group of the person arrested, and when the crime happened. This shows the patterns of crime, who commits them, and when and where they happen.
- **Characteristics** that might provide additional insights: Recording `Time` of crime and adding it to the dataset might be helpful to analyze the patterns in the crime if any.

3. The Solutions

3.1 Solution for Question 1

- Create a new method in `analysis_t1` called `most_arrests_day_most_common_race():`
  - `self` is the instance of the class.
  - `self.arrests` is the instance variable of the class.
  - the elements of the list are dictionaries and each dictionary is a row in the csv.
- Initialize an empty dictionary `{}` called `arrests_per_day` to store the arrests on each day.
- Initialize another empty dictionary `{}` called `arrests_by_race_on_max_day` to hold the number of arrests by a race on the day with the highest number of arrests.
- Create a for loop with the loop variable `arrests` and iterate over the `self.arrests` list which each item represents an arrest record.
  - store the `"ARREST DATE"` column information in the variable `arrest_date`
  - if the `arrest_date` exists in the dictionary `arrests_per_day` then increase that key in the `arrests_per_day` dictionary by `1`
  - else, add the `arrest_date` to the dictionary by adding `1`
- Find the day with the highest number of arrests by using the `max()` built-in function with `.get` on `arrests_per_day` which gets the key arrest date with the highest number of arrests. Store it in `max_arrests_per_day`.

- Create another for loop for `arrests` in `self.arrests`
    - if the `arrest_date` is equal to `==` the `max_arrests_per_day`, store it in `perp_race`
    - if `perp_race` is in `arrests_by_race_on_max_day` dictionary, then increase the value by `1`
    - else, add the `perp_race` to the dictionary and initialize the count to `1`.
- Use the `max()` built-in function again to find the arrest counts by race on the highest arrest day with `.get` on `arrests_by_race_on_max_day`
- For this method, use the print statements to give a clear result; print `f"The day with the most arrests was: {max_arrests_per_day}, " f"the most common perpetrator race on that day was: "`
- Then `return` a tuple containing `max_arrests_per_day, most_common_race`
- Example of sample input and output: for the input of `most_arrests_day_most_common_race()` which is a csv file called `NYPD_Arrest_Data_T1.csv`, the output would be a tuple containing the date of the most arrests and the race of the perpetrator on the day with the most arrests; **("1/1/2024", "BLACK")**

3.2 Solution for Question 2

- Create a new method in `analysis_t1` called `highest_felony_offense_in_borough()`
    - `self` is the instance of the class.
    - `self.arrests` is the instance variable of the class.
    - the elements of the list are dictionaries and each dictionary is a row in the csv.
- Initialize an empty dictionary `{}` called `borough_dict` which stores data from each borough with the key being the felony offense and the value being the frequency.
- Create a for loop to iterate over `self.arrests` which contains a list of arrest records data using the loop variable `arrest`.
    - at each step check if the arrest category called `LAW_CAT_CD` is a felony offense by seeing if it is equal to `'F'` in order to filter by felony offenses
    - assign the borough from `"ARREST_BORO"` in the arrest dictionary to `borough`
    - assign the offense from `"PD_DESC"` in the arrest dictionary to `offense`
    - check if the `borough` is already in `borough_dict`
    - if it is not, add the borough by initializing a dictionary with the keys. Since it is not in the dictionary yet, set the`"arrest_count"to0and"offenses"to an empty dictionary{}` to account for the different felony offenses.
    - Then, increase the `"arrest_count"` by `1` in the specific borough every time the type of felony arrest is found.
    - to count the offenses, check if the `offense` is already in the borough `"offenses"` dictionary, and if it is, increase that offense type by `1`.
    - otherwise, if the `offense` is not yet counted, create the `offense` and add to the ' `"offenses"` dictionary by initializing it with `1`.
- Create a variable called `most_felony_arrests_borough` to track the names of the boroughs with the most arrests and set it to `None`.
- Create a variable called `max_arrest_count` to store the borough with the highest number of felony arrests and set it to `0`.
- Create a for loop with two loop variables called `borough` and `data` to iterate through `borough_dict` using the `.items()` method. This will loop through both loop variables at the same time in the dictionary.
- if the `"arrest_count"` for the borough is greater than `>` the `max_arrest_count` then update the following:

    - update `most_felony_arrests_borough` with the new `borough`
    - update `max_arrest_count` with the borough with the most felony arrests using `data['arrest_count']`
- Initialize a new variable to track the `most_committed_offense` in the borough with the most felony offenses and set it to `None`
- Initialize another new variable to track the `max_offense_count` to track how many times that offense occurred and set it to `0`.
- Create a for loop with two loop variables called `offense` and `count` to iterate over the `most_felony_arrests_borough` using `items()` method in the `borough_dict` which is a dictionary where the borough is the key. The `.items()` method will return a tuple which contains the `offense` which is the name of the felony offense and `count` which is the number of times that offense was committed in the borough with the most arrests.
    - if the `count` is greater `>` than the `max_offense_count`
        - update the `most_committed_offense` with the new most committed `offense`
        - update the `max_offense_count` with the offense most committed by `count`
- For this method, use the print statements to give a clear result; print `f"The borough with the highest number of felony arrests was: "{most_felony_arrests_borough}."` and `f"The most committed felony offense being " {most_committed_offense} with {max_offense_count} occurrence(s)."`
- Then `return` a tuple containing `most_felony_arrests_borough, most_committed_offense, max_offense_count`
- Example of sample input and output: for the input of `highest_felony_offense_in_borough()` which is a csv file called `NYPD_Arrest_Data_T1.csv`, the output would be a tuple containing the borough which had the most felony arrests and in that borough which felony offense was most committed with the number of times it occurs; **("M", "LARCENY,GRAND FROM OPEN AREAS, UNATTENDED", 2)**.

3.3 Solution for Question 3

- define a new method in `analysis_t2.py` called `crime_most_committed_agegroup`.
- `self` is the current instance of the class.
- `self.arrests` is the instance variable of the class.
- the elements of the list are dictionaries.
- each dictionary corresponds to a row in the text file. `crime_type` is passed as an argument, to find the age group that has committed it the most.
- declare an accumulator `age_group_crime_count` and initialize it to an empty dictionary, to store the crime count of the age groups.
- declare one more accumulator `gender`, to hold the crime count by gender and initialize the values of keys `M` and `F` of it to `0`.
- using `for` loop iterate through each `arrest` record of `self.arrests` and at each iteration:
    - check whether `crime_type` is in `arrest['PD_DESC']` using `if`
    - now, `if arrest['AGE_GROUP']` is `not in` the dictionary `age_group_crime_count`, then:
        - initialize the value of `age_group_crime_count` for the key `arrest['AGE_GROUP']` to zero.
    - increment the value of `age_group_crime_count` for the key `arrest['AGE_GROUP']` by 1.
    - also increase the count of the value `gender` for the key `arrest[PERP_SEX]` by 1.
- assign the max value from the `age_group_crime_count` to the variable `highest_crime_count` using the `max()' function with default value as zero.

- `highest_crime_count` holds the maximum count of crime committed.
- to the list `highest_count_age_groups` assign all the `keys` of `age_group_crime_count` whose count equals `highest_crime_count` using `.items()`.
- `highest_count_age_groups` list has the list of age groups which have committed the most.
- assign the ratio of `gender['M'] : gender['F']` obtained using string concatenation to `ratio`.
- return the tuple `(highest_count_age_groups, highest_crime_count, ratio)`.
- Example of sample input and output:
    - for the input of crime type "Intoxicated and Impaired Driving", output will be similar to this **(["25-44", "18-24"], 2, "3:1")** which is a tuple containing a list of age groups which committed the crime the most number of times, count of crime committed, ratio between male and female.

3.4 Solution for Question 4

- define a new method in `analysis_t2.py` called `crime_most_committed_agegroup`.
- `self` is the current instance of the class.
- `self.arrests` is the instance variable of the class.
    - the elements of the list are dictionaries.
    - each dictionary corresponds to a row in the text file.
- `crime_type` is passed as an argument, to calculate the average time difference of arrests of that `crime_type` for each `BOROUGH` and each `AGE_GROUP`.
- define an accumulator `time_diff`and initialize it to an empty dictionary.
- `time_diff` is used to store the time differences between all the consecutive arrest dates.
- define one more empty dictionary `arrest_days`, an accumulator to store the arrest dates of each borough according to age groups.
- using `for` loop, iterate through each `arrest` record of `self.arrests` and at each iteration:
    - check whether `crime_type` is in `arrest['LAW_CAT_CD']` using `if`:
        - if yes, then assign the value of the borough available at `arrest["ARREST_BORO"]` to the variable `boro`.
        - and also assign the value of the age group available at `arrest["AGE_GROUP"]` to the variable `age_group`.
        - now, split the arrest date with `"/"` as a delimiter at `arrest["ARREST_DATE"]` using `split()` method.
        - convert the result of the above step to a tuple of integers using `map()` and then packing them into a tuple.
        - assign this tuple to a local variable called `arrest_date`.
        - check `if boro` is in `arrest_days`:
            - if not, initialize the value of `arrest_days` for the key `boro` to an empty dictionary.
        - check `if age_group` is in `arrest_days[boro]`:
            - if not, initialize the value of `arrest_days[boro]` for the key `age_group` to an empty list.
        - append the `arrest_date` to the list `arrest_days[boro][age_group]`.
- iterate through `arrest_days.items` with `boro, age_groups` as iterators and at each iteration:
    - `if boro` is not in `time_diff`, then initialize `time_diff[boro]` to an empty dictionary.
    - using `for` loop, iterate through the items of `age_groups.items` with `age_groups, dates` as iterators and at each iteration:
        - assign the sorted list of `dates` obtained using the `sorted()` function to a variable called `dates`.

        - define an empty list called `diffs` to store the list of differences of each consecutive arrest date.
        - now iterate through `dates` using `range()` function on `length-1` of `dates` with `i` as an iterator.
            - call the `date_diff` instance method by passing `dates[i] and dates[i+1]` as arguments.
            - append the `return` value of the above method to `diffs`.
        - if the list `diffs` is not empty then:
            - find the average time difference by dividing the sum of `diffs` by length of `diffs` obtained using `sum()` and `len()` respectively.
            - typecast the average value to float and assign it to a local variable called `avg_time_diff`.
            - round off the average to 2 decimal values using `round()` and assign it to `time_diff[boro]` at the key `age_group`.
        - else,
            - assign `zero` to `time_diff[boro]` at the key `age_group`.
- declare an empty dictionary called `result` to store the average time difference of `crime_type` for each `AGE_GROUP` of each `ARREST_BORO` in a readable format.
- iterate through `time_diff.items()` using the iterators `boro, age_group`:
    - initialize `result[boro]` to an empty dictionary.
    - iterate through `age_group.items()` using the iterators `age, time`:
        - at `result[boro][age]` store the value of `time`
- return the dictionary `result`.
- Example of sample input and output:
    - for the input of crime type Violence "V", output will be similar to this **{ "Q": {"25-44": 4.0, "18-24": 17.0, "65+": 33.0}, "M": {"25-44": 1.0, "18-24": 0.0, "65+": 20.0}, "K": {"25-44": 0.0, "18-24": 7.0, "45-64": 2.0, "65+": 111.0}, "B": {"25-44": 6.0, "18-24": 7.0, "45-64": 5.0, "65+": 108.0}, "S": {"18-24": 78.0, "65+": 0}, }**
    - which is a dictionary with boroughs as keys and value of each borough is a dictionary with age groups as keys and average number of days between arrests for that age group as value.

4. Test Cases

4.1 Test Case Information for Question 1

- Tests the `most_arrests_day_most_common_race()` method of `AnalysisT1` class.
- Create a testing file called `test_most_arrests_day_most_common_race.py`
- In the testing file create 3 separate test cases for 3 different csv files named as `NYPD_Arrest_Data_#_T1.csv`.
- Use the naming convention `test_most_arrests_day_most_common_race_#()`
- For each test case, create a doc string that talks about the test.
- The method is to find the date with the most arrests, so the `Arrest date` field from the data set is used.
- Expected output is a tuple of 2 elements as ('Arrest_date', 'Race').
- Alternative way of getting and testing the results is to filter necessary field(s) in the data sets and count the number of rows manually.
- See the file `test_most_arrests_day_most_common_race.py` for more details.

4.2 Test Case Information for Question 2

- Tests the `highest_felony_offense_in_borough` method of `AnalysisT1` class.
- Create a testing file called `test_highest_felony_offense_in_borough.py`
- In the testing file create 3 separate test methods for 3 different csv files named as `NYPD_Arrest_Data_#_T2.csv`.
- Use the naming convention `test_highest_felony_offense_in_borough_#()` to write the test methods.
- For each test case, create a doc string that talks about the test case details such as method name, class name, file used, inputs if any.
- The method is based on felony crime and boroughs, so test cases are written accordingly.
- Expected output is a tuple of 3 elements: ('Borough', 'Offense', 'Occurrences')
- Alternative way of getting and testing the results is to filter necessary field(s) in the data sets and count the number of rows manually.
- See the file `test_highest_felony_offense_in_borough.py` for more details.

4.3 Test Case Information for Question 3

- Tests the `crime_most_committed_agegroup()` method of `AnalysisT2` class.
- Create a testing file called `test_crime_most_committed_agegroup.py`
- In the testing file create 3 separate test methods for 3 different csv files named as `NYPD_Arrest_Data_#_T2.csv`.
- Use the naming convention `test_crime_most_committed_agegroup_#()` to write the test methods.
- For each test case, create a doc string that talks about the test case details such as method name, class name, file used, inputs if any.
- The method expects crime type as parameter, so test cases are written accordingly.
- Expected output is a tuple with 3 elements: `(['list of age groups'], crime count, male:female ratio)`.
- Alternative way of getting and testing the results is to filter necessary field(s) in the data sets and count the number of rows manually.
- See the file `test_crime_most_committed_agegroup.py` for more details.

4.4 Test Case Information for Question 4

- Tests the `avg_time_diff_between_arrests()` method of `AnalysisT2` class.
- Create a testing file called `test_avg_time_diff_between_arrests.py`
- In the testing file create 3 separate test methods for 3 different csv files named as `NYPD_Arrest_Data_#_T2.csv`.
- Use the naming convention `test_avg_time_diff_between_arrests_#()` to write the test methods.
- For each test case, create a doc string that talks about the test case details such as method name, class name, file used, inputs if any.
- The method expects crime type as parameter, so test cases are written accordingly.
- Expected output is a dictionary with `boroughs` as keys and its values as a dictionary with `age groups` as keys and `average time difference for arrests` as values, `{'Borough':{'Age Group':Count}}`.
- Alternative way of getting and testing the results is to filter necessary field(s) in the data sets and count the number of rows manually.

- See the file `test_avg_time_diff_between_arrests.py` for more details.

5. Results and Discussion

5.1 For Question 1

Results

- The `most_arrests_day_most_common_race` method returns the expected output for the datasets containing 10, 50, 100 records.
- The expected output is a tuple with 2 elements:
- The date with the most arrests.
- Most common race on the day with the most arrests.

Interpretation

- The method correctly identifies the day on which the most number of arrests took place and it also gives what is the most common race. This shows that the logic is functioning as intended.

Insights

- Analyzing the day and race for the crimes can help in adapting preventive measures.

5.2 For Question 2

Results

- The `highest_felony_offense_in_borough` method returns the expected output for the datasets containing 10, 50, 100 records.
- The expected output is a tuple with 3 elements:
- The borough with the highest felony offense.
- The second element is the type of the felony offense.
- Last element is the count.

Interpretation

- The method correctly identifies the borough with the highest felony offense and its count along with the type of the offense involved.

Insights

- Finding which offense is common in a borough will help in finding patterns of the offenses that are occurring and necessary measures can be identified from it.

5.3 For Question 3

Results

- The `crime_most_committed_agegroup` method returns the expected output for the datasets containing 10, 50, 100 records.
- The expected output is a tuple with 3 elements:
- The first element is the list of age group(s) that have committed the given crime type the most number of times.
- The second element is the highest crime count.
- Last element is the crime ratio between Male and Female.

Interpretation

- The method correctly identifies the age group that commits the most crimes for crime types passed as parameter. This shows that the age group logic is functioning as intended.

Insights

- Understanding the age groups most involved in certain crimes can help in developing targeted intervention and prevention strategies.

5.4 For Question 4

Results

- The `avg_time_diff_between_arrests` method returns the expected output for the datasets containing 10, 50, 100 records.
- The expected output is a dictionary of dictionary:
- Dictionary with `boroughs` as keys and its values as a dictionary with `age groups` as keys and `average time difference for arrests` as values

Interpretation

- The function accurately calculates the average time difference between arrests for different crime types and age groups. This indicates that the time difference calculation logic is correctly implemented.

Insights

- The results can be used to analyze patterns in arrest timings for different crime types and age groups, which can be valuable for law enforcement agencies to understand and address crime trends.

6. Reflection

6.1 Challenges

- What was the most challenging part of the project?
    - Using github to collaborate for the first time during the project (T1)
    - Analyzing the dataset and finding relative records that are expected by my methods.(T2)

6.2 Successes

- Which part of the project went really well?
    - My partner was easy to work with and we communicated very well which successfully helped us not have any merging issues in github. (T1)
    - Collaboration with the team mate. (T2)

6.3 Lessons learned

- What is the most valuable lesson learned through the project?
    - Understanding the real world value of finding answers to important questions based on data that you are interested in. (T1)
    - Even though I had prior experience with Github, I learnt the essence of github is to help coders bring version control into their projects and workspaces.(T2)