Testing:

Software testing consists of multiple phases and subphases, including development, release, and user testing. As our ‘customer’ can be considered to be internal (to our organization), we provide minimal specifications and details about user testing, instead, we allow relevant team members to act as the user and provide relevant feedback.

Development Testing:

Development testing will consist of unit tests, component tests whereby we focus on creating tests that can determine if multiple methods/classes can work together as a component as well as a series of systems tests. The system tests involve testing system menus (implemented via the terminal for our project) and we test the menus with incorrect input formatting, length, and type.

The tests and behavior of the system at each stage is given below:

Unit Tests:

Since the system operates on command line inputs, we must define all inputs, then enter them into the terminal and observe the output. The initial setup of tests (inputs and outputs is given below)

Our inputs consist of Location, radius, priority (safety or coolness), display_list (bool)

We can also assert whether the observed behavior of the system, for each of the six test cases, follows the expected behavior. The details are given below, for each test case.

We note two aspects of all of the unit tests, two tests correspond to one location and search radius (the difference between the test cases is the value of the priority variable). We also note that the expected behavior can be difficult to define, as ‘coolness’ or ‘safety’ of a bar is subjective. For our purposes, we will use the specific terminology used in the requirements to define what exactly coolness and safety entail. The breakdown is also dependent on the algorithm used in the implementation of the system.

For the sake of simplicity, we will not detail the specifics of the algorithms (can be found in the code), instead, we will define the main factors and their relation to other factors which are proportional to the safety and coolness values

Coolness_value ~ num_reviews + average_rating + cool_index

safety_value ~ average_rating - cool_index + (min_review / num_reviews)

*Where cool_index = an index belonging to the set [0,3*num_categories], (this value is dependent on the categories the specific bar is listed under)*

We can assert whether the observed output matches the expected output, against these formulae.

The six-unit tests are given below (inputs, behavior, and assertions),

Unit Test One:

Inputs: loc= ‘New Brunswick NJ’ , rad = 1610, priority = “safety”,display_List=False

Observed output:

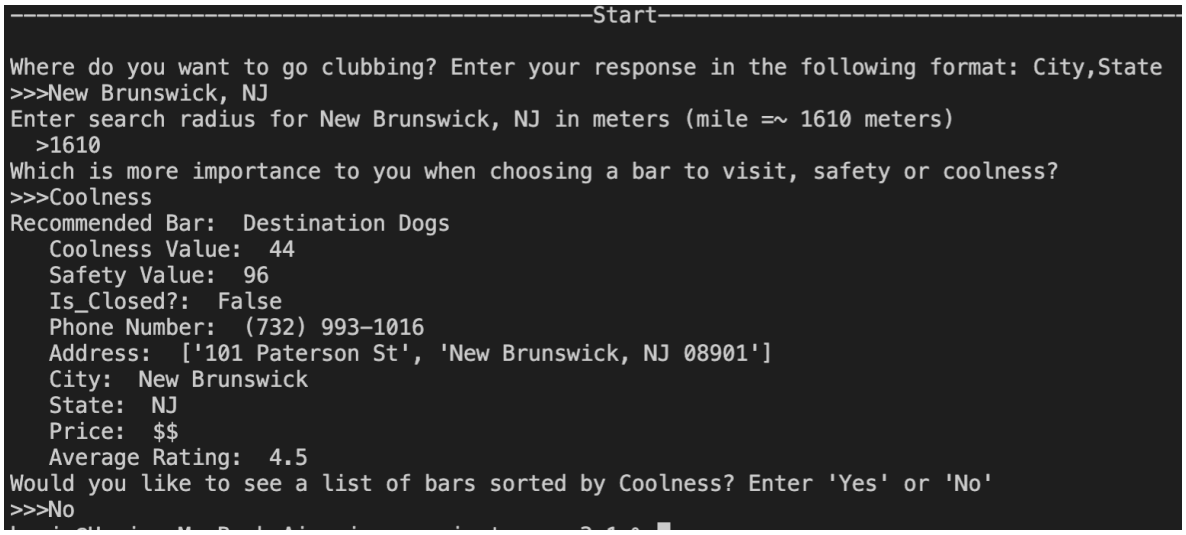
```
-----------------------------------------Start-----------------------------------------

Where do you want to go clubbing? Enter your response in the following format: City,State
>>>New Brunswick, NJ
Enter search radius for New Brunswick, NJ in meters (mile =~ 1610 meters)
  >1610
Which is more importance to you when choosing a bar to visit, safety or coolness?
>>>Safety
Recommended Bar:  Glo Ultra Lounge
   Safety Value:  96
   Coolness Value:  44
   Is_Closed?:  False
   Phone Number:  (732) 261-4044
   Address:  ['367 George St', 'New Brunswick, NJ 08901']
   City:  New Brunswick
   State:  NJ
   Price:  $$
   Average Rating:  3.0
Would you like to see a list of bars sorted by Safety? Enter 'Yes' or 'No'
>>>No
```

For the first test case we know the bar and all of its output data are correct, since when examining the list of bars, the first entry is the same as the bar recommended. So we have been recommended the safest bar, we also cross check the data (phone number, address,ect..) and find that the system (google maps lists the bar as closed but yelp does not). Based on the algorithm we know the recommendation is correct since it has a low cool_index, and low number of reviews (51). By definition safety values are inversely related to coolness values (for general cases)

Unit Test Two:

Inputs: loc= ‘New Brunswick NJ’ , rad = 1610, priority = “coolness”, display_List=False

Observed output:



For this unit test, we have the same specifications as unit test one but with priority defined as coolness instead of safety. The recommended baris in fact the bar with the highest coolness value (tested by inspecting the list of bars), all posted data follows the data provided on the website and yelp page. The bar has 1020 reviews, an average rating of 4.5 and is listed as mediocrely cool (the categories are listed as food and pub ~ moderately cool). Therefore the coolness value is in line with what we expected (45-65 +/- 10).

Unit Test Three:

Input: loc: ‘Jersey City NJ’ , rad = 1610, priority = “safety”,display_List=True

Observed output:

```
-------------------------------------------Start-------------------------------------------

Where do you want to go clubbing? Enter your response in the following format: City,State
>>>Jersey City, NJ
Enter search radius for Jersey City, NJ in meters (mile =~ 1610 meters)
  >1610
Which is more importance to you when choosing a bar to visit, safety or coolness?
>>>Safety
Recommended Bar:  Abbey's Pub & Grill
   Safety Value:  96
   Coolness Value:  88
   Is_Closed?:  False
   Phone Number:  (201) 963-3334
   Address:  ['409 Monmouth St', 'Jersey City, NJ 07302']
   City:  Jersey City
   State:  NJ
   Price:  $
   Average Rating:  3.0
Would you like to see a list of bars sorted by Safety? Enter 'Yes' or 'No'
>>>Yes
```

```
----------Bar 1----------
Name:  Abbey's Pub & Grill
Safety value:  96

----------Bar 2----------
Name:  The Factory
Safety value:  89

----------Bar 3----------
Name:  Ita Italian Kitchen
Safety value:  87

----------Bar 4----------
Name:  Life Pancake Company
Safety value:  83

----------Bar 5----------
Name:  Hard Grove Restaurant
Safety value:  77

----------Bar 6----------
Name:  Low Fidelity
Safety value:  74

----------Bar 7----------
Name:  Sushi By Bou - Jersey City
Safety value:  69

----------Bar 8----------
Name:  Madame Claude Bis
Safety value:  61

----------Bar 9----------
Name:  Pet Shop
Safety value:  58
```

```
----------Bar 10----------
Name:  Carvao BBQ
Safety value:  57

----------Bar 11----------
Name:  The Archer
Safety value:  54

----------Bar 12----------
Name:  The Brightside Tavern
Safety value:  54

----------Bar 13----------
Name:  White Star Bar
Safety value:  49

----------Bar 14----------
Name:  Cellar 335
Safety value:  48

----------Bar 15----------
Name:  Gringo's
Safety value:  47

----------Bar 16----------
Name:  Ani Ramen House JC
Safety value:  46

----------Bar 17----------
Name:  Left Bank Burger Bar
Safety value:  45

----------Bar 18----------
Name:  Würstbar
Safety value:  45

----------Bar 19----------
Name:  Zeppelin Hall
Safety value:  43

----------Bar 20----------
Name:  Lucky 7
Safety value:  39
```

For the fourth unit test, we can confirm that the provided bar is the safest and all supplied data is valid. Secondly the output list is also correct, both in order and actual safeness of the bars. The only issue that has arisen is the lack of bars provided in the list. Some bars are missing from the output list (we know this by observing the output given in the next test case). We expected to see ‘Lucky 7’ (safety_value = 96) as either ‘Bar 1’ or ‘Bar 2’ on the generated list, however it is not included, pointing to a defect in the system.

Unit Test Four:

Input: loc= ‘Jersey City, NJ’ , rad = 1610, priority = “coolness”,display_List=True

Observed output:

```
-------------------------------------------Start-------------------------------------------

Where do you want to go clubbing? Enter your response in the following format: City,State
>>>Jersey City, NJ
Enter search radius for Jersey City, NJ in meters (mile =~ 1610 meters)
  >1610
Which is more importance to you when choosing a bar to visit, safety or coolness?
>>>Coolness
Recommended Bar:  Lucky 7
   Coolness Value:  88
   Safety Value:  96
   Is_Closed?:  False
   Phone Number:  (201) 418-8585
   Address:  ['322 2nd St', 'Jersey City, NJ 07302']
   City:  Jersey City
   State:  NJ
   Price:  $
   Average Rating:  4.0
Would you like to see a list of bars sorted by Coolness? Enter 'Yes' or 'No'
>>>Yes
```

```
----------Bar 1----------
Name:  Lucky 7
Coolness value:  88

----------Bar 2----------
Name:  Würstbar
Coolness value:  39

----------Bar 3----------
Name:  Left Bank Burger Bar
Coolness value:  26

----------Bar 4----------
Name:  Cellar 335
Coolness value:  25

----------Bar 5----------
Name:  Zeppelin Hall
Coolness value:  24

----------Bar 6----------
Name:  Ani Ramen House JC
Coolness value:  24

----------Bar 7----------
Name:  White Star Bar
Coolness value:  22

----------Bar 8----------
Name:  Gringo's
Coolness value:  21

----------Bar 9----------
Name:  The Archer
Coolness value:  20

----------Bar 10----------
Name:  Pet Shop
Coolness value:  20

----------Bar 11----------
Name:  Sushi By Bou - Jersey City
Coolness value:  20
```

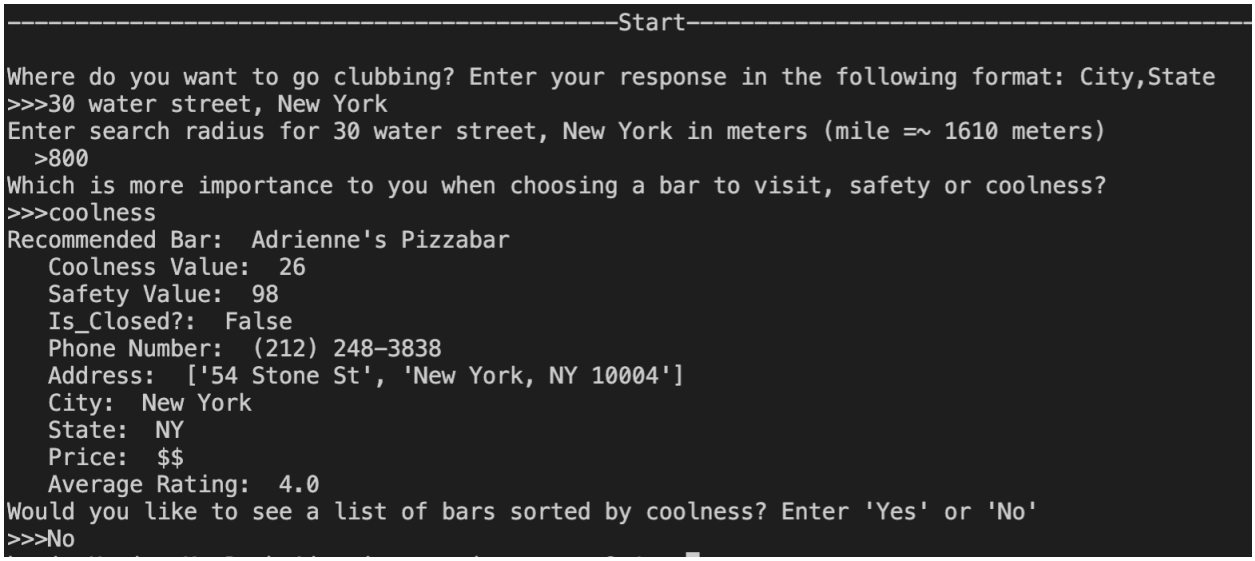
```
----------Bar 12----------
Name:  Low Fidelity
Coolness value:  20

----------Bar 13----------
Name:  Ita Italian Kitchen
Coolness value:  20

----------Bar 14----------
Name:  Madame Claude Bis
Coolness value:  19

----------Bar 15----------
Name:  The Brightside Tavern
Coolness value:  18

----------Bar 16----------
Name:  Carvao BBQ
Coolness value:  17

----------Bar 17----------
Name:  Hard Grove Restaurant
Coolness value:  16

----------Bar 18----------
Name:  Life Pancake Company
Coolness value:  16

----------Bar 19----------
Name:  The Factory
Coolness value:  14

----------Bar 20----------
Name:  Abbey's Pub & Grill
Coolness value:  13
```

We firstly confirm the recommended bar is the coolest bar in the outputted list (which it is), and we can confirm that all details listed below the bar are correct. The number of reviews are high, and it has a somewhat good rating (4.0) which drives its coolness value up. The coolness_index is extremely high (since this bar isnt listed under a resteraunt/food category but instead listed as a sportsbar/pub). We can also observe that the list is ordered correctly (descending order) and all bars listed meet the inputs defined earlier. The only issue that has arisen is that the bar 'Abbey's pub and bar' is ranked with a coolness value of 88 (the same as 'Lucky 7') yet it does not show up in the list. The same can be said for 'Lucky 7' which has a safety rank of 96 (same as Abbeys pub) but does not show up in the list. We can conclude that the system has not passed this unit test (and test four) fully, as the list of bars is not comprehensive (it skips some bar entries). This bug has most likely occurred due to either restrictions defined by the API (can only provide x=20 bars) or some internal defect that only prints twenty bars and for some reason skips over certain entries.

Unit Test Five:

Input: loc= '30 water st New York' , rad = 800, priority = "coolness",display_List=False

Observed output:



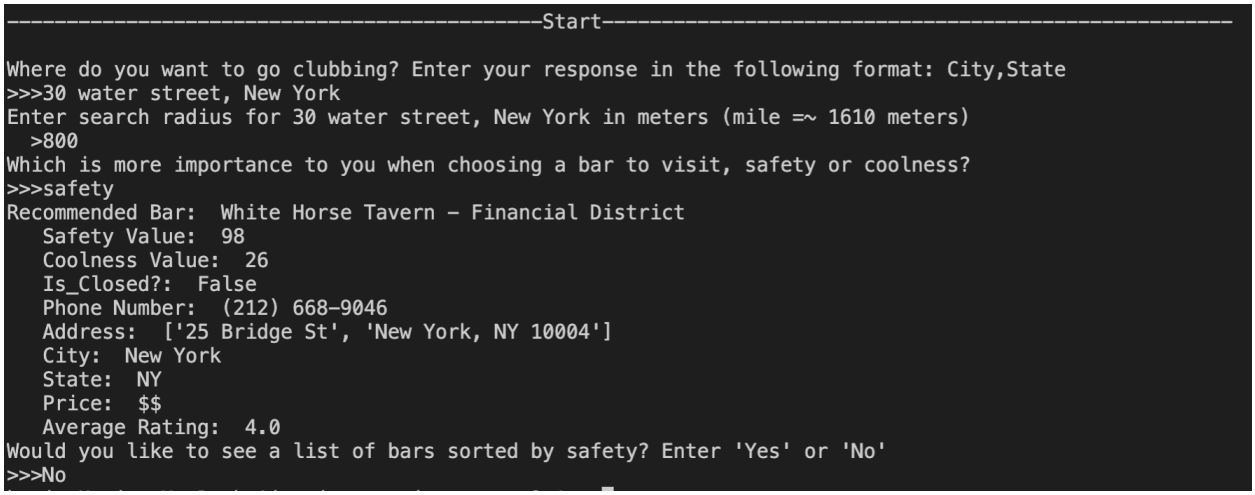
For this test case we alter the location to the address of a seemingly cool bar in the financial district (NYC). The address given is that of a bar named 'The Dead Rat' which we expect to be in the first five entries of the list (potentially the recommended bar). When analyzing the list we note that 'The Dead Rat' is contained in the list, and the given bar 'Adrienne's Pizzabar' is in fact the highest ranked bar (by coolness). All supplied data is also correct. The only point of contention is due to the fact that the coolness value is so low. Considering the location is in New

York, we expected to observe a coolness value above 50 or 60. We can either call this a defect in this system, particularly in the coolness_value algorithm, or we can assume that bars in this particular area of New York City are not cool, which is possible. The financial district isnt known for its bar or night-life. What is interesting and possibly a defect, is that the safety value is extremely high technically can be expected since the coolness value is low and both are inversely related. All in all It would be important to explore this unit test further in later iterations.

Unit Test Six:

Input: loc= '30 water st New York' , rad = 800, priority = "safety",display_List=False

Observed output:



The last test case is also implemented for the same inputs as test five (except priority has changed). We confirm all supplied data is correct and the safety value matches up with the expected value, since the number of reviews of this bar is low relative to the minimum number of reviews of all bars, and since the coolness value is so low. We also note the coolness ans safety values of this bar are the same as the bar outputted in the previous test. When running the test again and observing the generated list, we see the previous bar in the list of bars which means the system has operated correctly in this test case. The only issue observed (with unit tests five and six) relate to the systems ability to handle bars with equivalent coolness/safety values, as well as a lack of precision in the coolness value (caused by the algorithm used).

The system runs automatically, and its class structure is very minimal. As a result, it is difficult to write tests for specific methods and instances of objects. At this iteration of the project, we neglect this portion of the unit tests (as the system does this automatically - methods/classes call other methods/classes by themselves). We can redefine the class hierarchy of the system to incorporate tests for specific classes and methods.

Component Testing:

After creating unit tests and asserting the end behavior of the system, we can begin to test specific components (made up of intertwined methods and objects). We create a series of tests designed to test each of the main components that make up the system. Our tests focus on altering parameters for API calls, providing null or out-of-bounds inputs

The first three tests alter the parameters of the Yelp API (loc,rad) beyond its limits (focus on the API interface component) [display_list = False, and Priority = "Coolness" - for first three tests to maintain space and simplicity]

Test One:

Input: loc = London United Kingdom, rad = 16100 (=16.1km)

Observed Output:

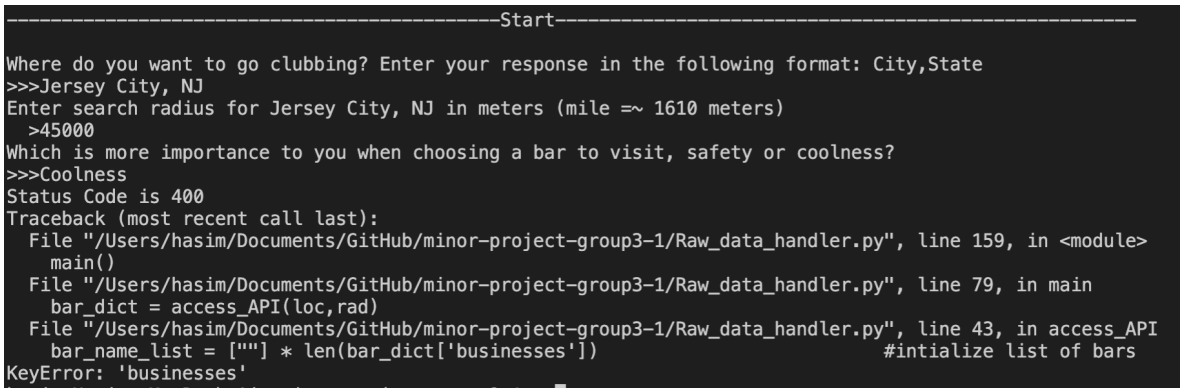
```
-------------------------------------------Start-------------------------------------------

Where do you want to go clubbing? Enter your response in the following format: City,State
>>>London, United Kingdom
Enter search radius for London, United Kingdom in meters (mile =~ 1610 meters)
  >16100
Which is more importance to you when choosing a bar to visit, safety or coolness?
>>>coolness
Recommended Bar:  KOKO
   Coolness Value:  53
   Safety Value:  99
   Is_Closed?:  False
   Phone Number:  +44 870 432 5527
   Address:  ['1A Camden High Street', 'London NW1 7JE', 'United Kingdom']
   City:  London
   State:  XGL
   Price:  ££
   Average Rating:  4.0
Would you like to see a list of bars sorted by coolness? Enter 'Yes' or 'No'
>>>Yes
```

We expected the API to not operate in this case, as the default region is the US. But supplying the system with an international location (London, UK) did not cause any defects in functionality.

Test Two:

Input: loc = Jersey City NJ, rad = 45000 (=~28miles)

Observed Output:



In this case, since the Yelp Fusion API documentation mentions that the maximum search radius is 40,000 meters (~25 miles), we expected the request to not work properly. The system works by generating a request to the Yelp API via its credentials (security key,ect..) and parameters (loc,rad,ect…). The status code of this request does not print, unless the value is not equal to 200 (status_code=200 means that not issues have occurred). In this case the status_code 400 means the request could not be processed due to some client error (that error being our parameter of rad = 45000). This test allows us to define a limit on the search radius, to 40,000 meters = 40km.

Test Three:

Input: loc = Toronto Canada , rad = 40001 (=~25miles)

Observed output:

We expected a status code = 400 to be printed, and some other errors to be outputted, which occurred. This test allowed us to determine if providing an international search request along with a search radius just above the limit (one meter greater than the specified limit). The output matches what we expected. Now we know the API only accepts search radius strictly less than (or equal to) 40,000 meters and that international searches are accepted.

The next three test cases provide null or 'out-of-bounds' inputs (focus on the components that retrieve cmd line inputs and operate on them or alter them) [display_list = False for all remaining tests]

Test Four:

Input: loc=Los Angeles, California, rad = 16100, Priority = "I dont know", display_list = False

Observed Output:

```
----------------------------------------Start----------------------------------------

Where do you want to go clubbing? Enter your response in the following format: City,State
>>>Los Angeles, California
Enter search radius for Los Angeles, California in meters (mile =~ 1610 meters)
  >16100
Which is more importance to you when choosing a bar to visit, safety or coolness?
>>>i dont know
Traceback (most recent call last):
  File "/Users/hasim/Documents/GitHub/minor-project-group3-1/Raw_data_handler.py", line 159, in <module>
    main()
  File "/Users/hasim/Documents/GitHub/minor-project-group3-1/Raw_data_handler.py", line 133, in main
    print("   Is_Closed?: ",bar_dict['businesses'][index]['is_closed'])
UnboundLocalError: local variable 'index' referenced before assignment
```

For this test we expected an error to occur, but we were not sure what error would occur or where the origin of this error would be. Looking more closely at the traceback we can discover the call that prompted the error, which was a line that attempted to access a sorted list with an defined index, where the index was dependent on the values of the variable Priority. Since Priority = "I do't know", an invalid input the defined index was never defined, and as such the program could not access the list with that index. Creating a default value, or using a try-except block would prevent this bug from occurring.

Test Five:

Input: loc = Miami, Florida rad = 5000, Priority = "coolness", display_list = "Im not sure what that means"

Observed output:

```
-----------------------------------------Start-----------------------------------------------------

Where do you want to go clubbing? Enter your response in the following format: City,State
>>>Miami, Florida
Enter search radius for Miami, Florida in meters (mile =~ 1610 meters)
  >5000
Which is more importance to you when choosing a bar to visit, safety or coolness?
>>>coolness
Traceback (most recent call last):
  File "/Users/hasim/Documents/GitHub/minor-project-group3-1/Raw_data_handler.py", line 159, in <module>
    main()
  File "/Users/hasim/Documents/GitHub/minor-project-group3-1/Raw_data_handler.py", line 94, in main
    bar1_obj = YelpHelp.bar(bar_dict['businesses'][i])
  File "/Users/hasim/Documents/GitHub/minor-project-group3-1/YelpHelp.py", line 21, in __init__
    self.price = bar_data['price']
KeyError: 'price'
```

In this test we observe an unexpected output which is related to an edge case. After providing the API with the supplied inputs, one of the internal systems accesses that data and stores it in various attributes and objects. One of those attributes is defined as ‘price’ which is read from the data returned by the API. Since not all bars have ‘price’ data it is possible for the API to not supply any data, and in that case attempting to access a null entry in the database creates the error we see above. We attempt to access an internal dictionary with the key ‘price’, not knowing that this search (for Miami, FL) has provided bars that have no price data.

Test Six:

Input: loc = Chicago, IL, rad = 800, Priority = “”, display_list = “False”

Observed output:

```
-----------------------------------------Start-----------------------------------------------------

Where do you want to go clubbing? Enter your response in the following format: City,State
>>>Chicago, IL
Enter search radius for Chicago, IL in meters (mile =~ 1610 meters)
  >800
Which is more importance to you when choosing a bar to visit, safety or coolness?
>>>
Traceback (most recent call last):
  File "/Users/hasim/Documents/GitHub/minor-project-group3-1/Raw_data_handler.py", line 159, in <module>
    main()
  File "/Users/hasim/Documents/GitHub/minor-project-group3-1/Raw_data_handler.py", line 133, in main
    print("   Is_Closed?: ",bar_dict['businesses'][index]['is_closed'])
UnboundLocalError: local variable 'index' referenced before assignment
```

For this test case, we can note that a bug has occurred, similar to what was present in test number five. In this case, we provided a null value when prompted for a value to be assigned to the variable ‘Priority’. The program then is not able to assign a value to the variable index, causing an error to be returned when the bar list is attempted to accessed using the value index (which is undefined at this point)

Test Seven:

Input: loc = Chicago, Illinois, rad = 800, Priority = “coolness”, display_list = “”

Observed output:

```
-------------------------------------------Start-------------------------------------------

Where do you want to go clubbing? Enter your response in the following format: City,State
>>>Chicago, Illinois
Enter search radius for Chicago, Illinois in meters (mile =~ 1610 meters)
  >800
Which is more importance to you when choosing a bar to visit, safety or coolness?
>>>coolness
Recommended Bar:  Big Star
   Coolness Value:  24
   Safety Value:  96
   Is_Closed?:  False
   Phone Number:  (773) 235-4039
   Address:  ['1531 N Damen Ave', 'Chicago, IL 60622']
   City:  Chicago
   State:  IL
   Price:  $$
   Average Rating:  3.5
Would you like to see a list of bars sorted by coolness? Enter 'Yes' or 'No'
>>>
```

For the last test, we provide valid inputs, all within the defined limits, with the exception of the display_list prompt, where we provide a null value. When doing so, we observe that the system behaves normally, specifically in this case the system infers a null value to be equal to False (does silence imply disagreement?) thereby allowing the system to execute and terminate normally without any issues. Since the user chose to skip the display_list prompt, it is assumed that the user does not assign much importance to viewing a list of bars (or they are limited on time) so the system uses that assumption to infer that the user does not want to see a list of bars.

System Testing:

As unit tests are considered the children of component tests and components are the building blocks of systems, the next logical step is to create high-level system tests to ensure systems work together as they should. At this phase, we begin creating tests to ensure the system works as expected, by altering the inputs. To do this we test the system menu by providing incorrect inputs (that are out of the scope of the documentation) and we alter text formatting (defined in component requirements). We then observe the output and determine if the system behaves as expected.

For system testing we focus on providing incorrectly formatted text inputs which affect the multiple systems (affects API interface which every other system is reliant on - if the API interface cannot call the API the entire system is rendered nonfunctional)

sisplay_list = “False” for all test cases

Test One:

Inputs: loc = “Jersey City” , rad = 2000, Priority = “coolness”

Observed output:

```
-----------------------------------------Start-----------------------------------------

Where do you want to go clubbing? Enter your response in the following format: City,State
>>>Jersey City
Enter search radius for Jersey City in meters (mile =~ 1610 meters)
  >2000
Which is more importance to you when choosing a bar to visit, safety or coolness?
>>>coolness
Recommended Bar:  Lucky 7
   Coolness Value:  87
   Safety Value:  99
   Is_Closed?:  False
   Phone Number:  (201) 418-8585
   Address:  ['322 2nd St', 'Jersey City, NJ 07302']
   City:  Jersey City
   State:  NJ
   Price:  $
   Average Rating:  4.0
Would you like to see a list of bars sorted by coolness? Enter 'Yes' or 'No'
>>>No
```

The system was able to infer that Jersey City referred to Jersey City located in New Jersey, which means State information is not always necessary when calling the API with the ‘loc’ variable. There is a city named Jersey city located in Wisconsin, but the system/API call defaulted to Jersey City, NJ. This provides relevant information related to location inputs, where the system will default to a specific city if the state information is not provided (perhaps it defaults to the most popular city if multiple locations with the same name exist)

Test Two:

Inputs: loc = “ NYC “ , rad = 200, Priority = “coolness”

Observed output:

```
-------------------------------------------Start-------------------------------------------

Where do you want to go clubbing? Enter your response in the following format: City,State
>>>   NYC
Enter search radius for    NYC    in meters (mile =~ 1610 meters)
  >200
Which is more importance to you when choosing a bar to visit, safety or coolness?
>>>coolness
Recommended Bar:  The River Café
   Coolness Value:  26
   Safety Value:  98
   Is_Closed?:  False
   Phone Number:  (718) 522-5200
   Address:  ['1 Water St', 'Brooklyn, NY 11201']
   City:  Brooklyn
   State:  NY
   Price:  $$$$
   Average Rating:  4.0
Would you like to see a list of bars sorted by coolness? Enter 'Yes' or 'No'
>>>No
```

For this test we altered the location input, by adding three empty spaces to the beginning and end of the string, and we used the abbreviation ‘NYC’ instead of following the city,state format. The API handling system as well as the Yelp API was still able to handle this data, by interpreting the input based on its own dataset or database. Attempting to abbreviation another city that isnt often abbreviated (for example entering NB for New Brunswick) would most likely not work as the system may not have encountered that input before (adding ‘NJ’ to the input might alter the systems ability to handle the data)

Test Three:

Inputs: loc = “baltimore MD” , rad = 1600, Priority = “cOOlness”

Observed output:

```
-------------------------------------------Start-------------------------------------------

Where do you want to go clubbing? Enter your response in the following format: City,State
>>>baltimore MD
Enter search radius for baltimore MD in meters (mile =~ 1610 meters)
  >1600
Which is more importance to you when choosing a bar to visit, safety or coolness?
>>>cOOlness
Traceback (most recent call last):
  File "/Users/hasim/Documents/GitHub/minor-project-group3-1/Raw_data_handler.py", line 159, in <module>
    main()
  File "/Users/hasim/Documents/GitHub/minor-project-group3-1/Raw_data_handler.py", line 133, in main
    print("   Is_Closed?: ",bar_dict['businesses'][index]['is_closed'])
UnboundLocalError: local variable 'index' referenced before assignment
```

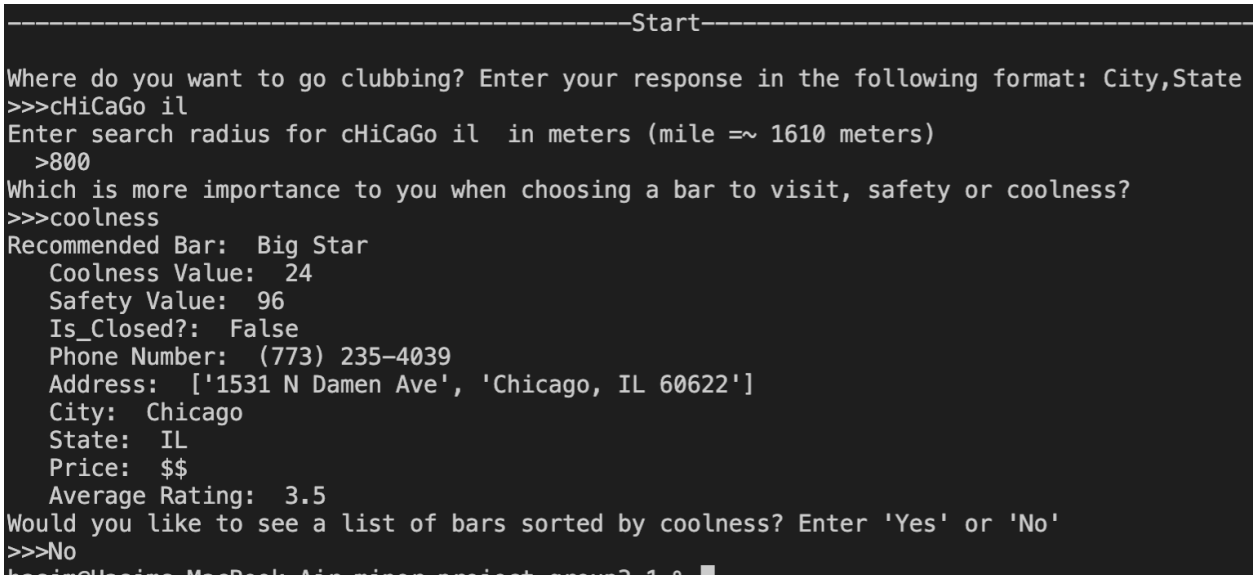
This test is similar to those conducted under component testing portion of development testing, where we observed that entering an input for Priority other than of either ‘safety’, ‘Safety’, ‘coolness’, ‘Coolness’ would prompt an error with an index used to access the bar list. This error was generated here, since the input for Priority was defined as ‘cOOlness’ and the system is using a string comparison statement to determine whether

to assign a value to the variable ‘index’. Altering the process (where we choose a method other than string comparison) would prevent this system defect from occurring

Test Four:

Inputs: loc = “cHiCaGo il” , rad = 800, Priority = “coolness”

Observed output:

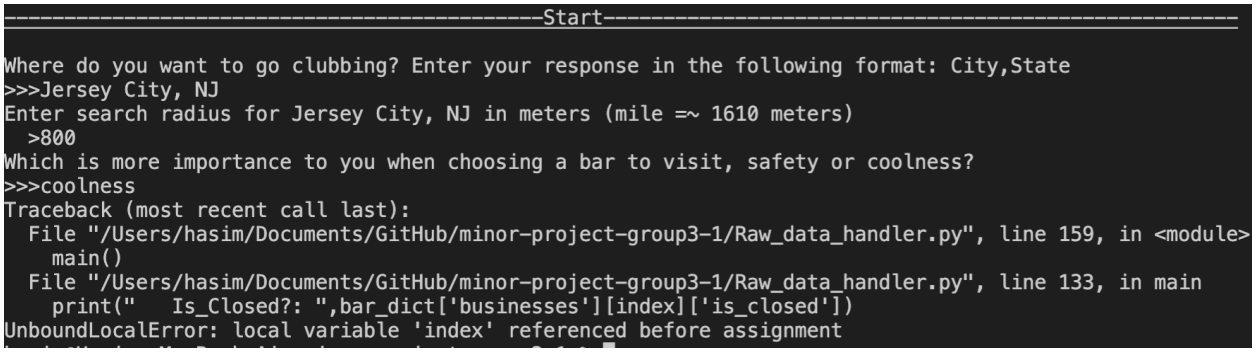


We Provided the system with valid inputs aside from the variable ‘location’ (type:string). For ‘location’ we provided a valid city, state in the correct format but with the inclusion of upper and lower case letters (allowing us to determine if strict string comparison is used). The system was able to handle the input, pointing to some different internal process used to parse inputs. The general trend is that for location and radius (variables used by the API) are able to take on a variety of values and strings with few exceptions. The variable ‘priority’ on the other hand requires one of two inputs otherwise the entire system fails and exits execution.

Test Five:

Inputs: loc = “Jersey City, NJ” , rad = 800, Priority = “coolness ”

Observed output:



Finally, we test the formatting of the weakest link - the priority variable. Appending a few spaces (three) to the end of the string creates the error we have become used to seeing (issue assigning a value to index due to an invalid input). Thus the Priority variable requires a strict string input (one of four exact inputs) otherwise the system will crash. Because of this steep risk, it is important to alter the use of this variable and input processing used by the system in order to prevent this test case from failing in the future.

Release Testing:

Requirements based Testing:

We design tests that ensure all requirements are met by the system and should serve as requirements checks. The details are given below:

Initially, we defined a set of requirements (user/system and functional/non-functional), we can create a set of tests corresponding to all of these requirements in order to ensure all requirements have been satisfied by the system

Test One:[Requirements]

We defined a user requirement to recommend a bar that is safe and cool, essentially optimizing safety and coolness. This has been expanded upon in previous tests where specified locations and radi were tested with Priority = coolness and safety. Since safety and coolness are inversely related the user requirement can only be satisfied be either recommending a cool or safe bar or by providing a recommendation that finds a bar where the safety and coolness thereof are maximized.

We can implement a test to determine if safety/coolness can be maximized (together) and if each can be maximized irrespective of the other.

Inputs: loc = Hoboken, NJ, rad = 1600, Priority = “coolness”, display_List = True

Observed Output:

```
----------------------------------------------Start----------------------------------------------

Where do you want to go clubbing? Enter your response in the following format: City,State
>>>Hoboken, NJ
Enter search radius for Hoboken, NJ in meters (mile =~ 1610 meters)
  >1600
Which is more importance to you when choosing a bar to visit, safety or coolness?
>>>coolness
Recommended Bar:  The Cuban Restaurant and Bar
   Coolness Value:  26
   Safety Value:  96
   Is_Closed?:  False
   Phone Number:  (201) 795-9899
   Address:  ['333 Washington St', 'Hoboken, NJ 07030']
   City:  Hoboken
   State:  NJ
   Price:  $$$
   Average Rating:  4.0
Would you like to see a list of bars sorted by coolness? Enter 'Yes' or 'No'
>>>Yes
```

```
----------Bar 1----------
Name:  The Cuban Restaurant and Bar
Coolness value:  26

----------Bar 2----------
Name:  Pilsener Haus & Biergarten
Coolness value:  24

----------Bar 3----------
Name:  Zack's Oak Bar & Restaurant
Coolness value:  22

----------Bar 4----------
Name:  Antique Bar & Bakery
Coolness value:  22

----------Bar 5----------
Name:  Grand Vin
Coolness value:  21

----------Bar 6----------
Name:  Elysian Cafe
Coolness value:  20

----------Bar 7----------
Name:  Bin 14
Coolness value:  20

----------Bar 8----------
Name:  Onieals
Coolness value:  20

----------Bar 9----------
Name:  The Brass Rail Hoboken
Coolness value:  19

----------Bar 10----------
Name:  House of ‘Que
Coolness value:  19
```

```
----------Bar 11----------
Name:  Del Frisco's Grille
Coolness value:  19

----------Bar 12----------
Name:  The Madison Bar & Grill
Coolness value:  19

----------Bar 13----------
Name:  Halifax
Coolness value:  19

----------Bar 14----------
Name:  City Bistro
Coolness value:  18

----------Bar 15----------
Name:  The Ainsworth Hoboken
Coolness value:  17

----------Bar 16----------
Name:  East LA
Coolness value:  17

----------Bar 17----------
Name:  Wicked Wolf Tavern
Coolness value:  16

----------Bar 18----------
Name:  Black Bear Bar & Grill
Coolness value:  16

----------Bar 19----------
Name:  Madd Hatter
Coolness value:  16

----------Bar 20----------
Name:  10th & Willow Bar & Grill
Coolness value:  16
```

User Requirements:

Based on the output, we observe that the bar with the highest coolness value has been recommended but its safety value is also maximized as well. Even though both are inversely related, it is possible for some cases that both the safety and coolness values will be greater than 80 (see Unit test for loc = Jersey City). The system is also able to isolate recommendations

and generate a list based on either safety or coolness, but by default the other value is maximized as much as possible.

This test case also satisfies every test that can be derived from the five system requirements detailed earlier. The generic tests (not inputs just test statements) and explanations of each test related to this particular set of inputs (for test one) are given below:

System Requirements Tests

1)Given a location and radius determine if the system compiles a list of bars following the defined constraints.

- The test case given above provides the compiled list of bars. All bars in the list follow the constraints (they are all within 1600 meters of Hoboken)

2) Determine if each bar contains coolness and safety values assigned by the system

- The recommended bar for the test case given provides safety and coolness values, and every list generated provides either safety or coolness values depending on the users preference. Although the system does not output both coolness and safety values for the list, the requirement is satisfied since both values ‘exist’ internally (even if they are not printed to the terminal)

3) Is the list of bars sorted by coolness in descending order?

- The list generated by test one prints a list of bars (20 bars) in descending order ranked by coolness where the first entry is the recommended bar

4) Does the system provide a bar recommendation based on safety/coolness values?

- The test case provides a recommendation (for a bar named ‘The cuban Restaurant and Bar’) which is based on the coolness value (since that was defined by the user)

5)Does the system automatically recommend a bar when constraints are not provided?

- The system does not recommend a bar automatically when constraints are not provided. This test assumes that the constraints are specifics about each bar, but we will infer constraints refers to the priority value. The system is not able to operate when a priority value is not provided, therefore failing this test, similar to the test given in earlier parts of the system.

Functional & Non-Functional tests:

For the sake of simplicity and to avoid redundancy we do not provide specific test cases for the functional & non-functional requirements as many were given earlier. We can summarize all remaining requirements into a few test prompts/questions which we will evaluate based on all of the relevant test cases given throughout the testing document.

1)Can the user search (obtain via API search) for a specific bar given custom constraints?

- All test cases allude to this test being passed. Each set of unique constraints given in the inputs created a specific bar recommendation (found via the API search)

2)Does the bar recommendation fit the user constraints?

- Every bar recommended in the test cases detailed earlier provides bars recommended based on the users inputs (if coolness is desired a cool bar is given,ect…)

3)Does each set of unique inputs correspond to a set of unique outputs?

- Changing the inputs in the unit tests (six unit tests at beginning of the document) reflected large changes in the output. Changing locations caused new bar recommendations/lists to be provided, and altering priority of safety or coolness caused changes in the output as well

4)Is the output provided within a reasonable amount of time?

- It cannot be displayed by test cases, but when the system is called with any inputs it is able to return the output within 5-10 seconds (measured across all test cases so far)

5)Is the storage requirement minimal?

- Some storage is necessary for the data gathered by the API, but since memory isn't efficiently created repetitively and since the data returned by the API is sifted by the program and limited to an output length of 20 (20 bars), little memory is used.

6)Can the environment allow the system to handle real-time data?

- Utilizing python along with VS Code allows for easy access to the Yelp API, thereby allowing the system to handle real time data (coming from the Yelp API)

Does the system avoid gathering personal information from the user?

- No personal information is requested or accessed by the system. The only instance of that is the use of a unique API key used to access the Yelp API but that is local and only accessible via source code.

Scenario Testing:

For this part of the release testing process, we design a typical scenario that is expected to be undertaken by a user. We design tests to mimic this use case and observe the behavior of the system.

The tests are implemented as apart of the overall typical use case, detailed below:

1. The user is planning a trip to a new city and wants to find cool bars to visit in the city within some distance of the user's hotel.
2. The user installs all packages for the system if running locally
3. The user then enters their desired location and search radius
4. Next the user enters whether they would like to go to a safe or cool bar
5. The user is provided with a bar recommendation and relevant data
6. Finally the user enters whether they would like to view a list of all bars meeting their criteria, sorted by their preference (coolness/safety)

We can refer to test one: requirements for the response of the system for this particular use case (set of tests). For that test the location is defined as Hoboken, NJ the radius is set to 1600, Priority set to coolness and a list of bars is displayed. The breakdown of this set of scenario tests and the specifics of the systems output for each part of the scenario is depicted in that test.

Performance Testing:

The system has few performance limitations, which only arise from accessing data via the Yelp API and storing all of this data locally. The Yelp API is limited to returning a list of 20 bars and only some attributes are provided for each bar. The internal system takes data from the dictionary provided by the API (it uses keys to access data) and stores it in instance attributes and objects defined by the driver file (Raw_Data_handling). Since we use a dictionary the time complexity of an average data access is O(1), some loops in the system contribute to the runtime but they are limited by the number of bars (20) which is a constant number (O(1)). All in all the system, internally performs well, the only potential for problems (access to API) is also limited since not all of the data returned by the API is sifted through.

For the sake of avoiding redundancy, we do not create specific performance tests, as the component test cases were able to test the peak of the systems performance. When the system is given a high-traffic location (like Jersey City, New York City or Los Angeles) and a large search radius the performance of the system decreases as more data must be handled, but runtime is still less than 10 seconds (details can be found in earlier test cases under component testing)

User Testing:

Our system has been implemented internally and our customer can be considered internal (we are both the developers and customers of this software). As a result user testing is extremely minimal for this project, but still important to consider.

To summarize, the system has passed most of the requirements and tests with two major exceptions. The system is not able to handle invalid inputs, it cannot operate when certain defined data is missing and the algorithms used to generate safety/coolness values are not as precise as they could be (although this isn't a requirement it can be considered important). The safety and coolness values generated are based on little data which means they are heavily influenced by a few factors, causing a decrease in overall precision.

Aside from what was mentioned, almost all unit tests and test cases have been passed, and most of the requirements are fulfilled by the system. I would recommend the system is released as an initial version, as it fulfills most of the requirements and is still functional and able to provide relevant recommendations which are extremely useful to users, especially those looking for a quick bar to visit. Existing changes that need to be made can be implemented via software updated (in an application setting updates would be pushed to users remotely), for our local setting we would define updates to add to the source code and continue with testing. Lastly automated testing should be implemented in the future to cover more edge cases.