# Restaurant Selector for the Indecisive

Claire Kolman

December 17, 2024

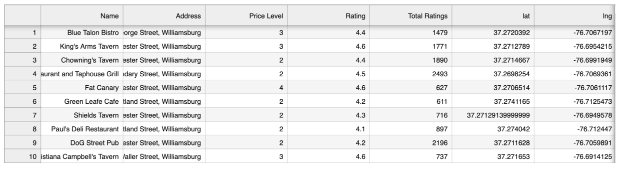
## Project Background and Motivation

The purpose for creating a restaurant picker assistant is to take the choices away when indecision gets in the way of meal time. The age-old question of "where should we go?" plagues not only individuals, but also groups of people when no one wants to be the one to make the decision. Taking the choice out of human hands can reduce possible conflict in this situation and simplify the solution. For an individual who can't decide, possibly having waited too long to eat and becoming overwhelmed with hangriness to the point they can no longer decide where to eat. This selector can help them immensely by giving them an immediate answer and get them to increase their blood sugar as quickly as possible. Another potential result from this project would be exposure to local businesses that people may not know about, but would fit their criteria perfectly. This could be a benefit to these businesses, but the effects would have to be studied post implementation before this conclusion can be accurately drawn.

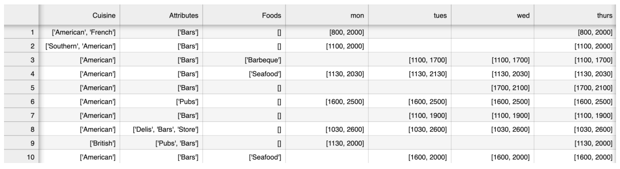
## Data Description

For the purpose of the creation of this program, a temporary incomplete dataset, included on Github, is being used. Further experimentation with google maps scraping through the google api would need to be conducted to formulate a complete dataset for the area. A goal is to add dietary information to the dataset, though it does not currently exist in the dummy data being worked with. The variable information I have at the moment comes from a combination of google maps and yelp listings by using their respective APIs to scrape the sites. Preliminary results for restaurants in the area were analyzed and it was decided to only include the entries that were not missing information for any of the variables to get the code working without having to do a lot of manual data entry at this stage. This data could be replicated either through the usage of google maps and yelp APIs specifically targeting: first the coordinates, address and name of the restaurant from google maps and the cuisine, attributes, price level, rating and hours from yelp, or through manual data entry of restaurants in the area desired including the necessary variables. A significant amount of processing was done on the dataset to get the variables in a working form to be queried from.

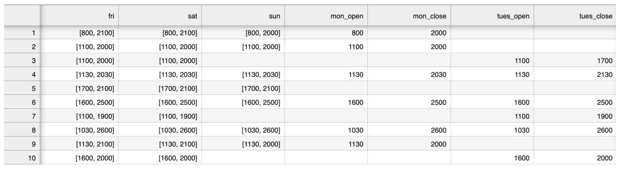
The data necessary columns (shown in images below):
'Name'(str), 'Address'(str), 'Price Level'(int): 1-4, 'Rating'(float): 0-5, 'Total Ratings'(int)[this variable is not required, only displayed in the output for additional information], 'lat'(float): latitude, 'lng'(float): longitude, 'Cuisine'(list[str]), 'Attributes'(list[str]), 'Foods'(list[str]), 'mon'-'sun'(list[int,int]): for monday through sunday list of open and close times in military time format, 'day open' and 'day close'(int) for all of the respective days: the time of open or close respectively in military time format.



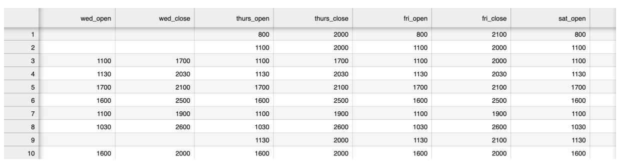
| | Name | Address | Price Level | Rating | Total Ratings | lat | lng |
|---|---|---|---|---|---|---|---|
| 1 | Blue Talon Bistro | orge Street, Williamsburg | 3 | 4.4 | 1479 | 37.2720392 | -76.7067197 |
| 2 | King's Arms Tavern | ester Street, Williamsburg | 3 | 4.6 | 1771 | 37.2712789 | -76.6954215 |
| 3 | Chowning's Tavern | ester Street, Williamsburg | 2 | 4.4 | 1890 | 37.2714667 | -76.6991949 |
| 4 | aurant and Taphouse Grill | ıdary Street, Williamsburg | 2 | 4.5 | 2493 | 37.2698254 | -76.7069361 |
| 5 | Fat Canary | ester Street, Williamsburg | 4 | 4.6 | 627 | 37.2706514 | -76.7061117 |
| 6 | Green Leafe Cafe | tland Street, Williamsburg | 2 | 4.2 | 611 | 37.2741165 | -76.7125473 |
| 7 | Shields Tavern | ester Street, Williamsburg | 2 | 4.3 | 716 | 37.27129139999999 | -76.6949578 |
| 8 | Paul's Deli Restaurant | tland Street, Williamsburg | 2 | 4.1 | 897 | 37.274042 | -76.712447 |
| 9 | DoG Street Pub | ester Street, Williamsburg | 2 | 4.2 | 2196 | 37.2711628 | -76.7059891 |
| 10 | istiana Campbell's Tavern | /aller Street, Williamsburg | 3 | 4.6 | 737 | 37.271653 | -76.6914125 |



| | Cuisine | Attributes | Foods | mon | tues | wed | thurs |
|---|---|---|---|---|---|---|---|
| 1 | ['American', 'French'] | ['Bars'] | [] | [800, 2000] | | | [800, 2000] |
| 2 | ['Southern', 'American'] | ['Bars'] | [] | [1100, 2000] | | | [1100, 2000] |
| 3 | ['American'] | ['Bars'] | ['Barbeque'] | | [1100, 1700] | [1100, 1700] | [1100, 1700] |
| 4 | ['American'] | ['Bars'] | ['Seafood'] | [1130, 2030] | [1130, 2130] | [1130, 2030] | [1130, 2030] |
| 5 | ['American'] | ['Bars'] | [] | | | [1700, 2100] | [1700, 2100] |
| 6 | ['American'] | ['Pubs'] | [] | [1600, 2500] | [1600, 2500] | [1600, 2500] | [1600, 2500] |
| 7 | ['American'] | ['Bars'] | [] | | [1100, 1900] | [1100, 1900] | [1100, 1900] |
| 8 | ['American'] | ['Delis', 'Bars', 'Store'] | [] | [1030, 2600] | [1030, 2600] | [1030, 2600] | [1030, 2600] |
| 9 | ['British'] | ['Pubs', 'Bars'] | [] | [1130, 2000] | | | [1130, 2000] |
| 10 | ['American'] | ['Bars'] | ['Seafood'] | | [1600, 2000] | [1600, 2000] | [1600, 2000] |



| | fri | sat | sun | mon_open | mon_close | tues_open | tues_close |
|---|---|---|---|---|---|---|---|
| 1 | [800, 2100] | [800, 2100] | [800, 2000] | 800 | 2000 | | |
| 2 | [1100, 2000] | [1100, 2000] | [1100, 2000] | 1100 | 2000 | | |
| 3 | [1100, 2000] | [1100, 2000] | | | | 1100 | 1700 |
| 4 | [1130, 2030] | [1130, 2030] | [1130, 2030] | 1130 | 2030 | 1130 | 2130 |
| 5 | [1700, 2100] | [1700, 2100] | [1700, 2100] | | | | |
| 6 | [1600, 2500] | [1600, 2500] | [1600, 2500] | 1600 | 2500 | 1600 | 2500 |
| 7 | [1100, 1900] | [1100, 1900] | | | | 1100 | 1900 |
| 8 | [1030, 2600] | [1030, 2600] | [1030, 2600] | 1030 | 2600 | 1030 | 2600 |
| 9 | [1130, 2100] | [1130, 2100] | [1130, 2000] | 1130 | 2000 | | |
| 10 | [1600, 2000] | [1600, 2000] | | | | 1600 | 2000 |



| | wed_open | wed_close | thurs_open | thurs_close | fri_open | fri_close | sat_open |
|---|---|---|---|---|---|---|---|
| 1 | | | 800 | 2000 | 800 | 2100 | 800 |
| 2 | | | 1100 | 2000 | 1100 | 2000 | 1100 |
| 3 | 1100 | 1700 | 1100 | 1700 | 1100 | 2000 | 1100 |
| 4 | 1130 | 2030 | 1130 | 2030 | 1130 | 2030 | 1130 |
| 5 | 1700 | 2100 | 1700 | 2100 | 1700 | 2100 | 1700 |
| 6 | 1600 | 2500 | 1600 | 2500 | 1600 | 2500 | 1600 |
| 7 | 1100 | 1900 | 1100 | 1900 | 1100 | 1900 | 1100 |
| 8 | 1030 | 2600 | 1030 | 2600 | 1030 | 2600 | 1030 |
| 9 | | | 1130 | 2000 | 1130 | 2100 | 1130 |
| 10 | 1600 | 2000 | 1600 | 2000 | 1600 | 2000 | 1600 |

## Methods

The main function called **choose-rest()**, selects either 1 random restaurant if the lucky argument (explained below) is left as the default True, or selects at most 5 restaurants that most closely match the criteria inputted. These choices are displayed on a map and in a dataframe including some additional useful information about the establishments.
Inputs:
- *data*(pd.DataFrame): A dataframe containing restaurant data. The dataframe read from the file 'update-data.csv' is the default value for this argument, so it is not necessary to specify one's own data.
- *day*(str, optional): Day of the week. Valid inputs include: ('mon','tues','wed','thurs','fri','sat','sun'). Without a specified day of the week the subsequent code will be run on the entire data set and could result in a closed establishment.
- *time*(int, optional): Time of day. A valid input is an integer between 0-2400 in military time format. ex. 2:30pm would be entered as 1430. Including a time without a day will have no impact on the results.
- *lucky*(bool): Default set to True, no other specifications are used and a restaurant is selected at random. Must be set to False if optional arguments are desired.
- *price-level*(int, optional): Integer between 1-4 indicating the price level they are willing to spend. rating(float, optional): Float between 0-5 indicating the ideal reviews rating.
- *cuisine*(str, optional): Desired cuisine type. Valid inputs include: ('American', 'French', 'Southern', 'British', 'Japanese', 'Greek', 'Chinese', 'Spanish', 'Italian', 'Thai', 'Mexican')

- *type*(str, optional): Desired atmosphere or trait. Valid inputs include: ('Pubs', 'Bars', 'Delis', 'Store', 'Delivery', 'Fast Food', 'Cafes','Bakeries', 'Tapas/Small Plates', 'Diners', 'Halal')
- *food*(str, optional): Desired food. Valid inputs include: ('Barbeque', 'Seafood', 'Pizza', 'Chicken', 'Sandwiches', 'Sushi', 'Breakfast, Brunch, Coffee', 'Salad', 'Burgers', 'Desserts', 'Wraps', 'Soup')

First the **choose-rest()** function uses the **sub-open()** function with the inputs: *data,day,time* to find a subset of the open restaurants. If day and time are not specified, or if only one or the other are specified, the **sub-open()** function will not get run and the subsequent code will run on the whole dataset.

- **sub-open()** takes the inputs: *data, day*, and *time*(same as above). This function works by checking if the time input is within the open and close hours for the desired day. It then creates a subset with all the rows where this condition is met.

After establishing the subset of open restaurants, (this is used whenever 'data' is referred to from now on) for each optional argument (*cuisine, price-level, type, rating, food*) that is specified, a new subset is created from the data where the entry matches the criteria for that variable's input. Five variables are specified at this stage of the function: *df-cuisine, df-type, df-price, df-rate*, and *df-food*. If the respective argument was specified, the function to check for it (explained below) is used and the variable is set to the subsetted dataframe. If an argument is not specified, the variable is set to None. The result will be some combination of dataframes and Nones depending on the arguments specified.

- **check-price()** takes the inputs: *price-level* and *data*. It checks the 'Price Level' column for entries where the price is less than or equal to the price-level input and returns a subset of the data where that condition is met.

- **check-rating()** takes the inputs: *rating* and *data*. It checks the 'Rating' column for the entries where the rating is greater than or equal to the rating input and returns a subset of the data where that condition is met.

- **checker()** takes the inputs: *column(variable)*, *type(input attribute)*, and *data*. This is a helper function for the following check functions. It finds the rows where the desired input is included in the search column. The data is then subset to only include these rows.

- **check-cuisine()** takes the inputs: *type(input attribute)* and *data* and uses **checker()** with the column argument specified as 'Cuisine' to return a subset of the data that matches the desired attribute.

- **check-type()** takes the inputs: *type(input attribute)* and *data* and uses **checker()** with the column argument specified as 'Attributes' to return a subset of the data that matches the desired attribute.

- **check-for-food()** takes the inputs: *type(input attribute)* and *data* and uses **checker()** with the column argument specified as ‘Foods’ to return a subset of the data that matches the desired attribute.

Before anything else occurs, the **choose-rest()** function checks if *lucky = True.* If this condition is satisfied, a random number is picked and the variable *target (this is the final chosen entries)* is set to a subset of the data at the index of that random number.
If *lucky = False* and at least one of [*df-cuisine, df-type, df-price, df-rate, df-food*] is not None, the **get-max()** function is used with the inputs: *df-cuisine, df-type, df-price, df-rate, df-food* and this provides a list of the restaurants that satisfied the most conditions.

- **get-max()** takes in all of the dataframe and none objects as arguments and returns a list of strings which are the restaurant names. For each of the respective arguments, if it is not None, the function **add-to-dict()** is used. A dictionary is returned from the **add-to-dict()** function and gets added to with each argument that is not None. After the dictionary is filled (**add-to-dict()** on each of the arguments that are not None), the next step in the function is determining the max value that exists within the dictionary. A list is then created and returned with each key that has that max value in it.

- **add-to-dict()** takes in one of [*df-cuisine, df-type, df-price, df-rate, df-food*], for which argument it is run on and also takes in a dictionary. This function iterates through the passed in dataframe and determines if that ‘Name’ already exists as a key in the dictionary. If it does, the value += 1. If not, the key is added with the value 1. This dictionary is a count of the times a restaurant is recognized in one of the dataframes passed into the **get-max()** function which represents everytime it satisfies a condition.

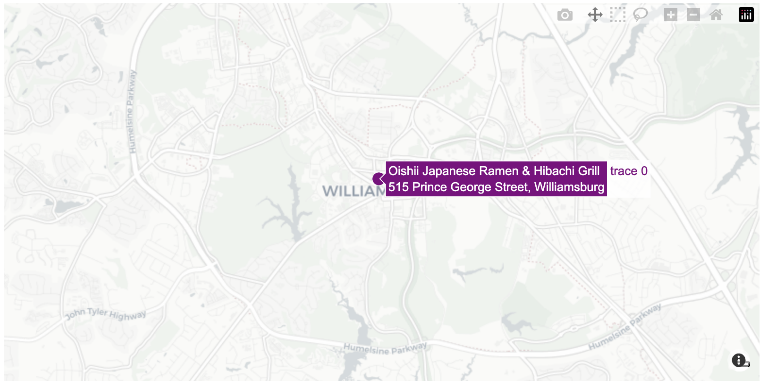
If the length of the returned list is longer than 5, (which was decided as the max number of restaurants to return no matter what) 5 random values are chosen and a new list is created to only include the names at the indices of those random values in the initial list.
Now that the list of names is 5 or shorter, the *target* variable is assigned a subset of the data where the ‘Name’ is in the list of names. This dataframe *target* is then used in the **make-viz()** function and *target[['Name','Address','Price Level','Rating','Total Ratings', 'Cuisine', 'Attributes', 'Foods',day+'-open',day+'-close']]* is returned as useful information.

- **make-viz()** This function generates and displays a figure with the selected restaurant(s). The only argument that **make-viz()** takes is *data*(dataframe). The map is created with the specifications: (width = 800, height = 400, margin = 'r':0, 't': 0, 'l': 0, 'b': 0, and showledgend = False). Additional settings include: customdata = data['Address'], hovertext displays text = ['Name'] and customdata, marker settings: size = 14, color = purple, opacity = 1, symbol = circle, mapbox style = carto-positron, and the map is centered at 'lat': 37.2707, 'lon': -76.7075 with zoom=12. The coordinates for the center are the middle of Williamsburg, VA. If one wanted to redo this project with data for a different city, these coordinates would need to be changed accordingly.

## Output

The output of my project is a table and map of the top 5 (or less) restaurants that align with the criteria inputted, or randomly selected restaurant if there is no criteria. The output if only the required day and time are input such as: (day = ‘thurs’, time = 1400), will be one restaurant from the open restaurant subset plotted on a map of Williamsburg and information is also returned below it with relevant information.

```
[156]: choose_rest(rest, 'thurs', 1400)
```

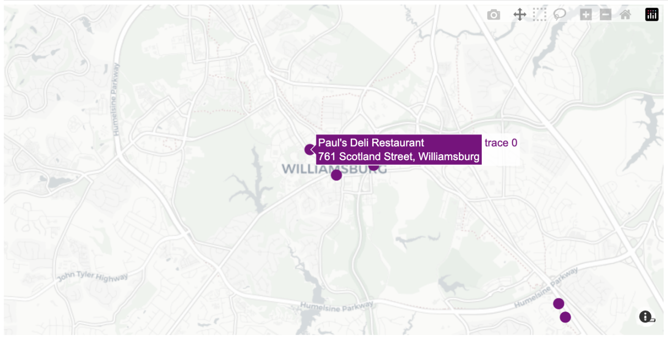


[156]:

| | Name | Address | Price Level | Rating | Total Ratings | Cuisine | Attributes | Foods | thurs_open | thurs_close |
|---|---|---|---|---|---|---|---|---|---|---|
| 8 | Oishii Japanese Ramen & Hibachi Grill | 515 Prince George Street, Williamsburg | 1 | 4.4 | 404 | ['Japanese'] | ['Bars'] | ['Sushi'] | 1200.0 | 2000.0 |

The output for at least one optional argument such as the following: (day = ‘fri’, time = 2000, lucky = False, type = ‘Bars’) will result in a subset appearing on the map along with the corresponding information appearing in the following dataframe.

```
[177]: choose_rest(df,'fri',2000, lucky = False, type = 'Bars')
```



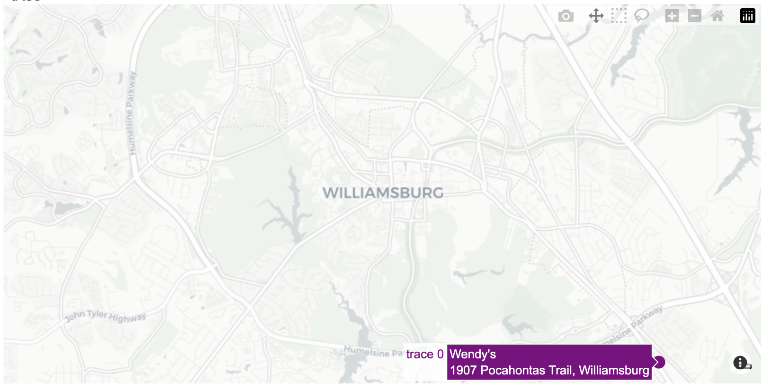
[177]:

| | Name | Address | Price Level | Rating | Total Ratings | Cuisine | Attributes | Foods | fri_open | fri_close |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | Chowning's Tavern | 109 East Duke of Gloucester Street, Williamsburg | 2 | 4.4 | 1890 | ['American'] | ['Bars'] | ['Barbeque'] | 1100.0 | 2000.0 |
| 3 | Berret's Seafood Restaurant and Taphouse Grill | 199 South Boundary Street, Williamsburg | 2 | 4.5 | 2493 | ['American'] | ['Bars'] | ['Seafood'] | 1130.0 | 2030.0 |
| 6 | Paul's Deli Restaurant | 761 Scotland Street, Williamsburg | 2 | 4.1 | 897 | ['American'] | ['Delis', 'Bars', 'Store'] | [] | 1030.0 | 2600.0 |
| 27 | The Whaling Company | 494 McLaws Circle, Williamsburg | 2 | 4.3 | 1359 | [] | ['Bars'] | ['Seafood'] | 1600.0 | 2100.0 |
| 35 | Thai Tara Sushi & Bar | 240 McLaws Circle #117, Williamsburg | 2 | 4.7 | 231 | ['Japanese', 'Thai'] | ['Bars'] | ['Sushi'] | 1200.0 | 2200.0 |

The following output would result from the situation where someone entered an optional argument, but didn’t specify lucky = False. The function will return the same output for a random restaurant, but will print a message incase that was not the intent of the user.

```
[178]: choose_rest(rest, 'fri', 2000, type = 'Bars')

Arguments were specified, but random choice is provided (default lucky = True). If specifications are desired, set lucky = F
alse
```



[178]:

| | Name | Address | Price Level | Rating | Total Ratings | Cuisine | Attributes | Foods | fri_open | fri_close |
|---|---|---|---|---|---|---|---|---|---|---|
| 36 | Wendy's | 1907 Pocahontas Trail, Williamsburg | 1 | 3.9 | 1249 | [] | ['Fast Food'] | ['Burgers'] | 1000.0 | 2300.0 |

If one set lucky = False, but didn't specify any arguments and or they didn't enter a valid input such as a misspelling or an option that doesn't exist in the data, they would see the following message and would not get a map output.

```
[179]: choose_rest(df,'fri',2000, lucky = False)

No valid arguments were provided, set lucky = True, or specify a valid argument for at least one of: price_level, cuisine, rating,
type, food. Check README.md for valid arguments.
```

## Future Work

I believe there are some steps/ideas that would be beneficial to this project that I either didn't have the technical skills and or time to implement. One of which would be the creation of an interface. I imagine it may have drop down menus of valid inputs which could reduce human error of spelling and or typing in an input that is not in the data set. In future exploration of this project I would like to form a more comprehensive data set either by expanding information of the entries I already have or including more entries. One particular variable I was interested in exploring initially was dietary restrictions, but this would have required manual research and input of that data which was just not something I had time for in this project timeline. The data research and input would be the most extensive part of this task, as writing a function would look very similar to the checker functions I already have with the difference being the valid inputs. Another expansion of the data set I would ideally have been able to get would be more thoroughly scraping google maps because I found that with the scope I was using there were still missing results due to a result cap from each section in the grid. Expanding the data in this way would require experimenting with the search radius and grid size to potentially improve results.