BACHELOR THESIS

Faculty of Science, Technology and Communication

Bachelor in Telecommunications

Understanding the Popularity of Local Businesses via Popular Times

2018/19

Cedric Figueiredo Sousa

Bachelor's Thesis in Telecommunications

Author:	Cedric Figueiredo Sousa
Supervisor:	Prof. Dr. Ulrich Sorger
Advisor:	Dr. Piergiorgio Vitello
Submission date:	16. August 2019

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Luxembourg, 16. August 2019

Table of Contents

1 Introduction

The design of the urban environment has for an extended period been planned by a more graphical approach to ensure a satisfying distribution of locations.

However, with the rise of sophisticated mobile devices, it has become clear that the information acquired from such mobile devices can produce much more accurate results, to improve urban environments and urban accessibility.

The main characteristics of how local businesses operate are popular times, wait times, and opening hours. That information helps a lot to plan and solve current and future urbanization issues, such as managing smart city services, managing a considerable amount of people during special events, help new businesses to choose an appropriate location and the required staff.

Using Google services as a mobile crowdsensing application allows programmers to accumulate precise information on how local businesses interact with customers and on the other side how long customers reside in a specific location and their experience based on a rating system. With this, local businesses can be categorized, and their popularity calculated.

Google generates these data-sets by cumulating and anonymizing collected data from users that chose to use Google location history. That data offers detailed information about local business names, geographical location, customer visits, popular times, duration of visit, and waiting time to access the service and more.

1.1 Related Work

For several decades there has been realized a notable amount of related work in terms of crowdsensing and mobile crowdsensing applications [1] using sophisticated visit attribution with GPS signals to create highly accurate datasets [9], which leads up to allow the creation of highly-accurate predictions of local business categories and attractiveness [2] on which this thesis is based.

2 State of The Art

2.1 Mobile Crowdsensing

In the day and age where the Internet of Things is slowly but surely taking over. It will enrich data scientists and developers with a large amount of sensor data, compared to the current mobile devices. However, they have similar functionality in terms of sensing, communication, and computing information about the environment. These applications are classed as personal and community sensing [1].

The personal sensing category includes applications of monitoring individual movement patterns such as running walking or exercising to keep record of evolution or health while community sensing, tracks information which cannot be easily measured by single data but requires a more considerable amount of it. For example, traffic congestion monitoring and air pollution monitoring which require more data to be accurately calculated.

Community sensing can then be divided into two categories as well, participatory sensing and opportunistic sensing. Participatory sensing requires the involvement of voluntary people to collect sensor data like business locations, pictures, or roads. On the other hand, opportunistic sensing collects sensor data automatically with small user contributions such as tracking user's movement pattern through location sampling.

2.1.1 Applications

In terms of mobile crowdsensing applications, it can be categorized into three different types, environmental, social and infrastructure

Environmental mobile crowdsensing includes collecting data of more extensive environmental conditions such as pollution levels in cities, water levels, and wildlife habitats. An application example for such data collection is special portable equipment for air quality sensing connected to a mobile device to measure CO2 or mobile devices with microphones to measure noise levels.

Social applications share data between individual users. For example, users can share and compare their exercise data with other individuals within a community and get better results. As an example, there is Komoot [14], which is based on OpenStreetMap to generate bike routes. Users can also take photos of the route share information about it while other members of the community then can follow the same route.

Finally, there are the Infrastructure applications that create sensor data from more significant size events such as traffic congestion, parking availability, public location, and real-time transit location in cities. Popular times falls in this category since mobile devices are used to generate information about public locations and their average time spent in those specific locations.

2.1.2 Characteristics

Nowadays, mobile devices dispose of a large variety of hardware and applications which can be used to create all sort of data. On top of that, with the vast amount of processing power and storage capabilities that such small devices are produced, it is much more convenient than deploying specific sensors to analyze any behavioral patterns.

In addition to that, every person nowadays disposes of a mobile device which they carry every time wherever they go. So, it is very convenient to create any applications for mobile devices since they dispose of all sorts of sensing capabilities to monitor public behavior. The most common sensors found in mobile devices are a microphone, camera, GPS, and accelerometer. The operating system then allows the application to gather sensing data from them. As an example, the location data can be extracted with a combination of GPS, Wi-Fi, and GSM. A typical mobile crowdsensing application is usually divided into two components. The first one is located on the device itself which has the purpose of collecting the sensor data while the second one is located on a cloud server and has the function to create a result of the more significant amount of data cumulated from all the devices as seen in figure 1 [3].

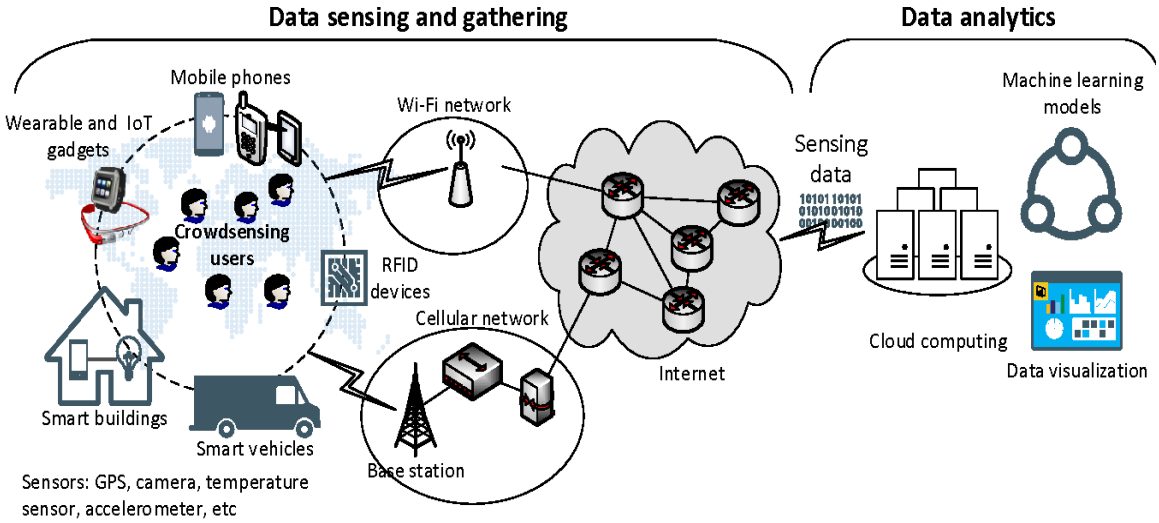

figure 1

As an example, the mobile crowdsensing application could analyze the geographical structure of hotspots or 5g antennas with data collection on how popular a specific area is over different time scales. Such information can help determine how much energy is needed to provide a good quality of connection and how much energy can be saved at different times of the day. Furthermore, it also helps plan future network deployment and expansions.

However, there are also many challenges when using mobile crowdsensing, each data produced by a mobile device is not always correct due to technical mistakes in terms of accuracy and latency since the device is continuously in movement. So, it is of great value to be able to determine the right devices that produced the desired data so that measurements can have a high quality. Another challenge encountered in mobile crowdsensing is that individuals naturally have privacy concerns and personal preferences that are not equal to the demands of mobile crowdsensing applications.

2.1.3 Privacy

Privacy is very user-specific, everyone has another perspective on what privacy is. Users might not want to share private information such as location history or other sensitive information at all while others are willing to share their location history at any given time. This makes an essential task for mobile crowdsensing applications to not only satisfy user demand in terms of privacy but also protect them from actual malicious intents of third parties.

The most common and popular measure to protect the collected data is anonymization. The process of anonymization is that of removing any identifier of the collected data before sharing it. Another measure of protection is the use of cryptographic algorithms to transform the information into an unreadable data chunk which then gets recovered at the backend (or cloud server).

2.2 Google

Google Inc. was founded in 1998 by Larry Page and Sergey Brin [5]. Known by the public for being the most popular search engine, it is also the world's largest internet company. Google offers a diversity of products from mobile devices, laptops to other accessories. Next, to the search engine, they also offer services such as e-mail, application store, interactive mapping technology, and APIs

2.2.1 Google Popular Times

Part of the Google Places Service, Google Popular Times provides data about popular times, wait times, visit duration, live visit data, ratings, and type of location.

The popular times are calculated based on the average customers that visited that location over the last week, while the wait times show how long a customer must wait until he gets attended during different times of the day. The wait times are calculated differently for each type of business [8].

Restaurant wait times return information on how long a customer must wait until he gets seated, on the other hand, a grocery store wait times returns the time on how long a customer has to wait to check out their items. The live visit data is a real-time calculation on how busy a specific location is at that given moment, and the visit duration shows how much time a customer typically spends at that location.

Google generates such data collection with the use of GPS signals and mobile device users that opted for Google's location history. However, GPS is just a part of how popular times are tracked and calculated. Other companies like Apple or Facebook also use GPS data, but what keeps those companies behind is that Google has the best building footprints, also mentioned as polygons in the industry.

These polygons help Google to determine the exact location and shape of a store. Google then uses machine learning and the already large GPS data pool to calculate the popular times.

However, these polygons cannot be found in the Google Places API since Google keeps them as a secret so that enterprises cannot copy them, but there is a similar work on how these polygons work, mentioned later.

2.2.2 Application Software Protocol

Application Software Protocols commonly referred to as APIs in computer programming are software tools used to facilitate the developer's work so that they can focus on the core part of their project.

An API can be anything from an operating system to a software library. In this Thesis, the term API will refer to software libraries. A Software library is added to a software program by the developer. The library consists of a pre-written code with classes, scripts, and data to create an automatic process without having to write all the code for it [4].

An example of such libraries can be one for mathematical functions so that the developer does not need to write a complex function themselves but only need to call it in the program.

2.2.3 Google APIs

The APIs developed by Google allows developers to communicate with Google Services. The APIs allow access to user data, machine learning as service or analytics. Google offers APIs that are implemented in several different fields and sectors, like web development and data science.

The ability to access that information increases a developer's possibility to focus on the core side of his project without having to make every single program or data from scratch. However, some APIs require authentication and authorization.

Developers can access Google APIs within their code in various languages, including Java, JavaScript, Ruby, .NET, Objective-C, PHP, and Python [6].

Google Place ID Finder

The Google Place ID Finder is a free API from Google based on their collected information from Google Places Services, which gives every specific location or business a unique identifier [7]. A user can enter the address of a specific location into the ID Finder Tool which then sets a marker to the location with the identifier in an information window.

2.3 Determining Visits from Location Data

In this section, a possible approach to determine visits based on GPS signals will be explained based on the SafeGraph experience and the implementation of visit attribution [9].

Being able to track down GPS signals and figure out if a mobile device visited a place or type of store can be very valuable information. However, there are accuracy problematics to determine if that place is visited due to having to deal with messy GPS data, incomplete business listing information, and limitations in knowing where places are located exactly.

2.3.1 Nearest centroid wins

The simplest way of determining if a GPS signal visited a location is when the GPS ping makes an entrance in a centroid of a point of interest. If that takes place, it is set as a visit to that location as seen in figure 2.

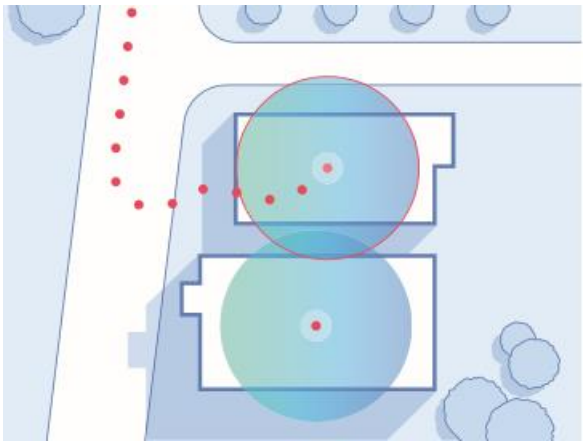

figure 2

However, this approach is not efficient enough. There is likely going to be another nearby location that is closer if a person is near a sizable point of interest. As an example, an airport would not have an obvious centroid that would be of use. Another issue is when smaller stores are located next to larger stores, the centroids of smaller stores will end up colliding with bigger stores for having smaller footprints, which can produce a falsification of the collected data.

That is why using centroids is not the most accurate approach, and determining visits need to consider the actual size of the building by using actual building footprint referred to as polygons.

2.3.2 Polygons

In terms of using polygons, the most straightforward approach is to count the GPS pings that are inside of one of the points of interest, as seen in figure 3.

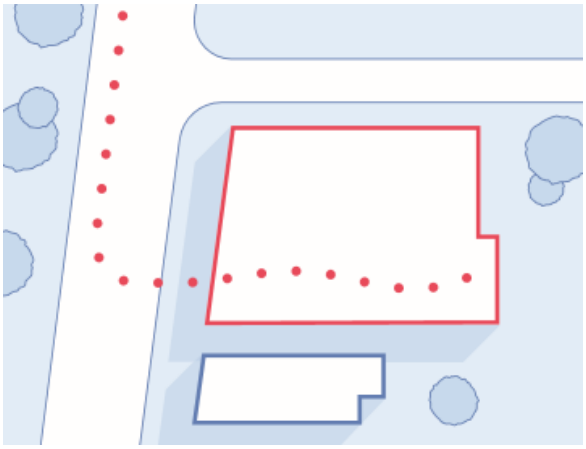

figure 3

However, there is also an issue with this type of approach. Drifting GPS signals can create false visits due to not enter the building polygon, at times the drift can be so powerful that it can enter other building polygons located around. However, there is a positive side to this approach. If the polygon is large enough (airports, golf place) any drift of the GPS is meaningless compared to the size of it.

2.3.3 Resolving Drift issues

A solution for resolving drift issues is to build a custom geofence with padding around the point of interest and then count any GPS pings that enter that area, but there is still the possibility of interfering with other GPS pings that passed by but did not enter. The same goes if there are two stores next to each other.

At this point, an algorithm of prediction needs to be used, one that takes into consideration the time of day, duration spent in the geofence, type of location, and opening hours. These features will then help determinate if the duration was too short to be one or if it is possible to be one by considering the opening hours.

Another solution and the simplest one is using IP-addresses to determine if a mobile device is in the location. The algorithm only needs to check if an IP-address falls inside a point of interest's Wi-Fi hotspot address range, and no prediction is needed. There is a limitation to this since not everyone connects to a location's Wi-Fi hotspot.

3 Limitation

Since there is a considerable number of mobile devices with all sorts of data the challenge remains in terms of how and with which implementation one can collect and analyze the data to find the most effective way to help improve and expand current or future environmental issues.

To do so, one must rely on Google's data collection since it is the current, most accurate worldwide data collection.

However, there are limitations on how far Google can help produce such mobile crowdsensing results. Currently, no specific Google Popular Times API for multiple businesses in an area of choice is released, which would allow a developer to access that information instantly. Even if they decide to, there would be most likely an expensive fee involved with a large amount of data. On top of that, there would be no monetary control over the collected data, and one would most likely need to pay for unusable data.

Nonetheless, there is the possibility to access popular times information through Google Places Services "Find Current Place" but the limitation to that is that such call for data only provides a response for one specific place.

A developer interested in generating an extensive data collection of popular places data needs first to find a way to collect information about every possible point of interest in the desired area before he proceeds with further implementation.

Then, with that information, the actual "Place ID" can be found, and only then the information can be retrieved for one point of interest.

4 Solution

As mentioned above, using Google Services as the mobile crowdsensing application allows programmers to accumulate precise information on how local businesses interact with customers, on the other side how long customers reside in a specific location and how their experience was based on a rating system. With this, local businesses can be categorized, and their popularity calculated for further investigation.

Then, with the help of location-based social networks like Facebook or Twitter and crowdsourced projects like OpenStreetMap a programmer can collect a vast amount of location-based data. The collected data can then be implemented and used in correlation with Google Services to acquire the “Places ID” information for a greater geographical area.

Furthermore, with the help of “Populartimes”, which is a Python library that can be found on the GitHub website, and the “Places ID”, the actual information can be retrieved from the Google Places Service "Find Current Place”.

4.1 OpenStreetMap

Created in 2004 by Steve Coast, OpenStreetMap is a collective project built by a community of volunteers to create a free geographic map of the world and with data about roads, restaurants, cafés, shops, railway stations and much more. This data is distributed for free so people can use them in creative and productive ways without having legal or technical restrictions on their use. Contributors use aerial imagery, GPS devices, and on-field technologies to verify that OSM is accurate and updated [15].

4.1.1 HOT Export Tool

The Humanitarian OpenStreetMap Team (HOT) Export Tool is an open service that uses up-to-date OpenStreetMap data and allows users to more easily obtain access to OSM data. Users can access updated geographical information from OpenStreetMap’s 1.3 million users, and design customized maps with specific tags. Users are also able to upload/update and download maps in different formats such as OSM, SQL, and Shapefile [16].

4.2 Populartimes

“Populartimes” is a Python library that can be found on the website GitHub [17] which was created and is still being updated by a variety of master students. The goal of the library is to enable programmers to use Google Popular Times data.

With the authorization of Google by giving out a Google Maps API-Key, a programmer is able with the help of this library and Google Places Service to retrieve information for a place id. A smaller monthly fee is then priced corresponding to the number of times the service has been used.

The response is formatted as a .json file containing popular times, wait time, time spent, and other popular data like ratings for a given local business or point of interest. The values are a combination of Google searches, Google maps application location data and local traffic data which gives hour and day readings on a scale of 1-100 on how busy the location is and 0 indicates that the business is closed.

4.2.1 JSON

JSON or JavaScript Object Notation is an easy to understand data format to read and write for humans and accessible for machines. The data consists of array data types which can contain strings, boolean, values, and lists.

JSON is a language-independent data format derived from JavaScript. Nowadays, many programming languages use codes to generate and implement JSON-format data.

5 Preparation

This chapter will present a couple of key features needed to have an environment in which the actual implementation of the thesis can be done. There will be a brief mention of the programming language, the editor of choice and the installation procedure, and how to create a path in the system which allows a simple installation of the needed libraries.

Furthermore, the actual intern and extern libraries will be explained and mentioned on how to download and install them through the command prompt.

5.1 Python

Python is one of the most popular programming languages created by Guido van Rossum and released in 1991. Over the past few years, Python has become a first-class programming language in automation, artificial intelligence, building applications, and websites.

The success of Python comes with a robust standard library and easily obtained and ready to use libraries from third-party developers. Furthermore, Python has been enriched by decades of expansion and contribution. Python's design is very readable with heavy use of whitespace, and it has an object-oriented approach to help programmers to write clear and logical code for small or more significant projects.

The programming language can be used to implement scientific and numeric applications, connect to database systems, read and modify files that facilitate data analysis and visualization.

5.1.1 Download and Install Python

To download the latest version of Python, one can go to the main Python website [10], from there by going under downloads. A user can click on the latest version and download the file.

After downloading the file, the user needs to locate the file in their operating system and start to install it by going through the usual Windows application installation procedure. In this thesis, the Python version 3.7.4 is used. After the installation is completed, a code editor is needed.

5.1.2 PyCharm

The code editor is needed to write the actual code, and there are many code editors out there. For this thesis, PyCharm is used, which is one of the most popular code editors for Python. PyCharm is considered an IDE which stands for Integrated Development Environment, which means that it has additional features to write code.

Developed by a Czech Company named “JetBrains”, first released on July 10, PyCharm provides utilities like code analysis, graphical debugger and more. PyCharm is aimed at Python, Django, and Google App Engine developers.

To download PyCharm, one needs to access the JetBrains website [11], by clicking on downloads, the webpage takes us to a page with two different types of PyCharm. The professional version is a paid version that includes further scientific tools while the community version is a free open source edition that became available on October 2013.

In this thesis, the community version is used, after downloading the code editor one needs to go through the usual Windows installation wizard process until PyCharm is installed.

5.1.3 Add Python Path to Environment Variables

A great feature for running Python and PyCharm is that one can perform actions from the Windows or any other operating system command-line, which helps to perform tasks faster like running a program or install extern libraries.

For that, a path variable needs to be created in Windows for the Python framework. First, one needs to locate the path in which Python is installed. If Python was installed with the default settings, the path should look like the following while “NAME” represents the username.

```
C:\Users\NAME\AppData\Local\Programs\Python\Python37-32
```

After this one needs to right-click on “This PC” select “Properties” and click on “Advanced System Settings”. By choosing “NEW” under environment variables, a window should pop up where the variable name and variable value can be created like in the following figure.

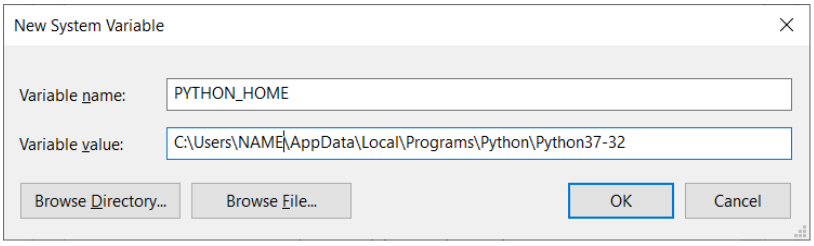

figure 4

Then, one needs to locate and select the “path” variable in environment variables under “System variables” and select “Edit”.

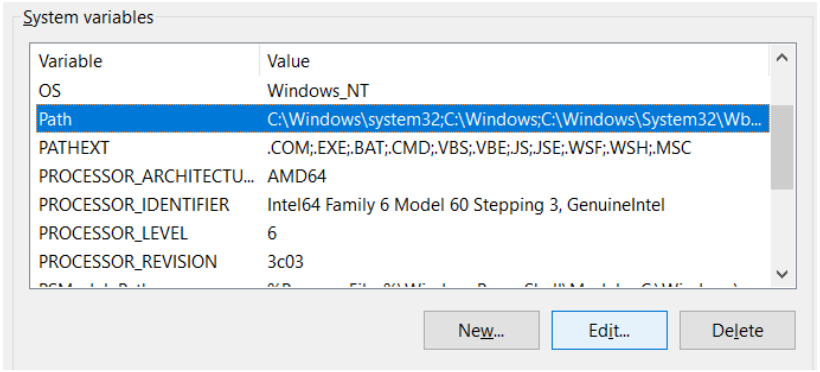

figure 5

In the “Edit environment variable” window select “New” and add the created variable.

%PYTHON_HOME%

Finally, to verify that Python is working with the Windows command prompt one can open it via searching for “cmd” in the search box or by pressing “Windows Key + R” to open the “Run” box and type “cmd”. If everything is set correctly, then the command prompt should give the current version of Python by typing the following.

```
> python
```

To exit the Python shell, one can use the following command.

```
> exit()
```

5.1.4 Installing pip

The best tool used in Python to install packages is called pip [12]. It is a standard package management system used to install and manage software libraries in a more rapid manner. Pip is known for the variety of applications using this tool and enables third-party package installations over the “easy_install” package manager. Usually, pip comes installed as a default with the newest version of Python. If pip is not installed, one needs to Download “get-pip.py” to a folder, open the command prompt, navigate to the folder containing “get-pip.py” and run the following command.

```
> python get-pip.py
```

To check if pip is installed and configured, one can run the following command which should give the current version of pip in the operating system.

```
> pip -V
```

To update pip, one can then use the following simple command.

```
> python -m pip install --upgrade pip
```

5.1.5 Core Library

The Python core library is composed of a variety of elements such as data types numbers, variables, and lists [13]., It also contains built-in functions and exceptions that can be used by the whole Python code without having to implement an import-statement. Some of the essential built-in functions are boolean operations that evaluate the truth of variables. There are also the usual arithmetic operations such as the sum, difference, or product of two or more variables and numbers. In conjunction with the arithmetic operations are the numeric types such as integer (int), float and complex. Next, to that are the text processing services like strings that allow creating a variable that contains a text. That variable can then be manipulated for further use.

There are also sequence types that are used to create lists, tuples, and range objects, which is used to create and accumulate a more significant number of variables or strings into one point of connection which then can be accessed through the program. Built-in exceptions are also used in programs to create a “try” statement with an “exception”. The program tries a chunk of code under the “try” statement, and if the code generates an error, the program ignores that previous chunk of code and continues with the exception to avoid any interruption to the process.

5.2 Extern Libraries

5.2.1 Pandas

Pandas is a free open source and popular software library used in data manipulation and analysis [18]. Pandas stands for “Python Data Analysis Library”. Pandas takes data, as an excel data format, and creates a python data object with rows and columns called “data frame” that looks very similar to the usual excel table. This library enables a much smoother workflow than using standard tools such as lists or dictionaries in conjunction with loops. To be able to use pandas as a library in python, one needs to download and install it first, which can be easily be achieved by using pip in the command prompt by using the following command.

```
> python -m pip install pandas
```

For further use, one can also easily update the library.

```
> python -m pip install --upgrade pandas
```

After the pandas is installed, it is ready to use by Python. One only needs to import the library first, which means loading it into the actual program that is being worked on. To do so, the import statement in conjunction with “pandas” needs to be included in the code.

```
1   import pandas as pd
```

The part “as pd” allows the user to access pandas with a “pd” command rather than having to write “pandas”.

5.2.2 XLWT

The xlwt library is used to write data and formatting information into spreadsheets in Excel format (.xls) [19]. The package itself has no dependencies on extern modules or packages. Pandas writes excel files using the xlwt module, that is why the XLWT library is used to create excel files since it is more efficient in the process while pandas is used to read in already created data files. To install the library, the following command is used in the command prompt.

```
> python -m pip install xlwt
```

The following import statement is then used in the code itself.

```
1   import xlwt
```

5.2.3 Matplotlib

Matplotlib is considered a plotting library designed for Python, which produces 2D figures out of data in a variety of formats across platforms [20]. Matplotlib is able to generate plots, histograms, pie charts, bar charts, and more. The library makes it possible to generate those simple plots with a few lines of code, but also allows the creation of fully customizable plots while keeping the code readable.

Furthermore, the module pyplot is used in conjunction with Matplotlib which provides a MATLAB-like interface that allows the user to have full control of line styles, font properties, and axes properties through a set of functions familiar to MATLAB.

The library gets installed by the following command.

```
> python -m pip install matplotlib
```

Then imported into the code with the following statement.

```
1   import matplotlib.pyplot as plt
```

5.2.4 Numpy

Numpy is a library used to support multidimensional arrays and matrices alongside with mathematical functions. It allows a user to generate arbitrary data-types and create routines [21].

Numpy uses much less memory to store data, comparing a numpy array memory to a generic Python list. It also provides a mechanism of specifying the data types of the contents, which allows further optimization of the code.

To install the library, one can use the following command.

```
> python -m pip install numpy
```

Also, import it with the following statement in the code.

```
1   import numpy as np
```

5.2.5 Populartimes

As mentioned in Section 4.2, the populartimes library is used to retrieve information of a place id as an option to use Google Maps Popular Times, until it is available via Google's API [17].

This API uses the Google Places Web Service with the SKU call "Find Current Place, where each call is priced. To be able to use this library, one needs to demand an API-Key [22], each call, at this current time, costs 0.017 USD [23].

An example of such a call through the populartimes library can be seen below while using an API-Key string and a place id string that is freely retrievable through Google Place ID Finder.

```
1   populartimes.get_id("api-key", "ChIJSYuuSx9awokRyrrOFTGg0GY")
```

The response for that call is formatted as a .json file containing all the information. An example of such a file can be seen in the Appendix.

The library can be installed directly from GitHub using the command prompt.

```
> python -m pip install git+https://github.com/m-wrzr/populartimes
```

Also, imported via the following statement.

```
1   import populartimes
```

6 Implementation

In this chapter, the required steps from choosing a location up to creating programs to analyze the data will be presented, which will consist of downloading the required files, manipulate them, create a program to gather the popular times and visualize them in many different manners.

6.1 Accumulate Area Information

In this thesis, a circle area of the Metropolitan City of Bologna, Italy, has been chosen, as seen in the following figure. The advantage with this area is that is simple to control later, when retrieving the places id, if the businesses reside in the circle area or not since there are similar names that could be located outside of the desired area, an example for that would be a fast-food chain like Burger King which can be located inside and outside the circle. Those local businesses will then be used to determine their popularity and average popularity in the area.

figure 6

The next step is to retrieve all the names of the local businesses in that area. To do so, one needs to use the HOT export tool located on their webpage [16]. To be able to download OpenStreetMap, one needs to sign up for an account. However, that is entirely free, and one can start exporting information right away. When accessing the tool, there is a variety of text boxes that need to be filled and text boxes checked. First, one needs to give a name to the export that is about to be executed and a brief description. Then under the export formats, the checkbox of the Shapefile (.shp) needs to be checked.

Furthermore, the actual desired data needs to be chosen, in this thesis, “Restaurants” will be checked which will output all the restaurants, fast foods, cafes, bars and pubs in the area. However, there are also a lot more that can be examined, such as shops, emergency stations, banks, government offices, healthcare, transportation, sports centers, and more.

Then, one needs to replicate the located area in Google Maps in the HOT Export Tool by using the “Draw”-Tool provided by the webpage in their own world map. For that, one needs to zoom into the Metropolitan City of Bologna and mark the desired area like in the following figure.

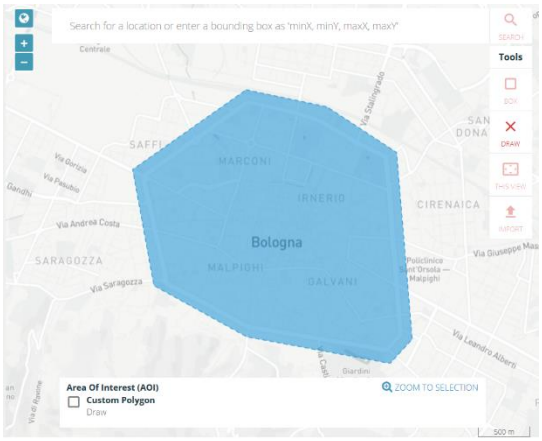

figure 7

Finally, one can start executing the Tool, and after a short time, the download of a compressed file will be downloaded.

That file has a few different types of files compressed. However, the only one that is important for the next steps is the excel file that contains 3 rows named “osm_id”, “amenity” and “name”. That excel file also contains over 600 local businesses for the “Restaurant” category.

6.2 Gather Places ID

With the help of the free Google API “Place ID Finder” mentioned in Section 2.2.3, one can retrieve a unique identifier for every point of interest in the world based on Google’s collected data. This identifier is used to retrieve the actual popular times within their data.

To use the free Google API, one needs to navigate to their “Place ID Finder”-webpage [24]. On their webpage, there is an interactive 2D map which allows a user to enter the desired location. In that textbox is where the retrieved name of the gathered locations from the HOT-Export Tool is written which will set a marker to a specific location followed by an information window with the desired “Place ID” as seen below.

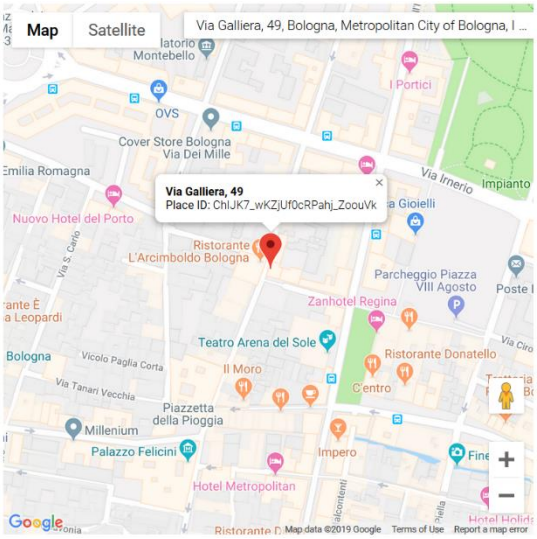

figure 8

That “Place ID” needs then to be copied into the already existing Excel file under a newly created “Google_Id” column in the appropriate row of the local business.

The “Google_Id” column is a crucial part of the following steps since the script will use that information to retrieve popular times.

6.3 Collect the Popular Times

In this section, the previously created and expanded excel file will be used to create a new, improved excel file with all the popular times for every “Place ID” that has been found.

Since there are many identifiers, the best solution is to create a program with the help of the populartimes library which will automatically retrieve the popular times and include them in a new excel file. This will enable a user to have all the information in one file so that it facilitates any future analysis.

The whole program can be found in the Appendix section of the thesis under “app.py File”. However, the crucial parts of the program itself will be presented and explained in this section.

To start with, one needs to create a new project in PyCharm and under “Project Interpreter” check the box to inherit global site-packages since the populartimes library is a third-party library and will not be recognized. With a right-click on the created project, a new python file can be included. After one can start importing the libraries pandas, xlwt and populartimes.

6.3.1 Reading Excel File

The first programming lines will be used to read the whole excel file containing the retrieved identifiers so that it is at disposal for the entire program. That is achieved with the help of the following pandas command which will read the excel file in the current project folder with the string as the file name.

```
1    df = pd.read_excel("Bologna_Centro.xlsx", sheet_name=0)
```

Pandas creates a data frame object which is stored into the variable called df. That data frame looks like a standard excel file but has the advantage that it can easily be used in the program in conjunction with other pandas commands.

6.3.2 Creating a New Excel File Template

The next step consists of creating a workbook for the new, improved excel file that will contain the popular times data. To do so one needs first to create a workbook with the help of the xlwt library with a variable called wb which will contain two sheets ws and ws2 named “Times” and “Times_Wait”.

```
wb = xlwt.Workbook()
ws = wb.add_sheet("Times")
ws2 = wb.add_sheet("Time_Wait")
```

6.3.3 Creating the Column Titles

To classify the .json popular times data that will be retrieved from Google, one needs to have column titles for every chunk of data so that it can be analyzed further. For that, the following xlwt commands are used, wherein the ws sheet, in row 0 and column 0, the string “address” will be written.

```
ws.write(0, 0, "address")
```

This will be done multiple times with the latitude, longitude, identifier, name, and more until it corresponds to the .json file.

However, to avoid repeating the same lines of code too often, the parts from Monday 0-1AM up to Monday 23-24PM can be written in two lines of code with the help of a for-loop. In this exact for-loop, the variable c will go from the range 6 to 30, which corresponds to the column numbers in increments of 1 while the c variable takes the value of the current number.

That variable is then used in the command itself to create the desired result of the loop. Furthermore, the value of 6-c will give the column a unique title from “Monday 0” to “Monday 23” which simplifies further analysis.

```
for c in range(6, 30):
    ws.write(0, c, f"Monday {c-6}")
```

This chunk of code will be repeated for all the week-days and types until the requirements for the .json file are met.

6.3.4 For-Loop for all Indexes

To ease all computation and create an automatic process for collecting all the popular times for all the identifiers located in the excel file and place that information into a new, improved excel file the following command is beneficial.

```
for index, row in df.iterrows():
```

This for-loop has the benefit that it uses the method iterrows() which is a generator that creates a repeated process over all the rows located in the data frame and returns the index which is faster than manually looping over the all the rows.

It means that for every row in all the rows, give index the current index while running through the loop. The following sections will then be inside this exact loop as the commands need to happen for every identifier in each row.

6.3.5 Collecting the Popular Times

In this part, the popular times for one “Places ID” is assembled with the help of the populartimes library. With the implementation of the method “populartimes.get_id” and including the API-Key with the Places ID, one can download the location-specific .json file and store it in the variable called res.

In conjunction with that, try and except statements are used. So that, when errors occur in terms of wrong or non-existent “Places ID”, the process continues while ignoring the rest of the loop and gives out an error message in the console.

If, however, the populartimes method is accomplished, the console is provided with a message of success. The variable i, that was created and given the value of 0 outside of the loop, gets incremented by 1 and the loop carries on.

```
        try:
            res = populartimes.get_id(apik, row["Google_Id"])
            print(f"Index: {index} Google_ID:
{row['Google_Id']}")
            i += 1
        except:
            print(f"Error on Index: {index} Google_ID:
{row['Google_Id']}")
            continue
```

6.3.6 Filling and Saving the New Excel File

Now that the res variable contains the .json file information for the location-specific Places ID, one needs to relocate that data into the right excel columns.

The initial data that gets relocated can be done directly in a simple manner since the “address” category of the .json file only contains one information. This can be done by using the following chunk of code while using the try and except statements for the possibility of the “address” part being idle.

```
1         try:
2             ws.write(i, 0, res["address"])
3         except:
4             ws.write(i, 0, "NA")
```

A similar chunk of code is used for coordinates which, however, contains a list in a list with one information. The information gets written in the appropriate ws sheet while the variable i is used as the index for the sheet and the number relocates the data into the specific column.

```
1         try:
2             ws.write(i, 1, res["coordinates"]["lat"])
3         except:
4             ws.write(i, 1, "NA")
5
6         try:
7             ws.write(i, 2, res["coordinates"]["lng"])
8         except:
9             ws.write(i, 2, "NA")
```

There are more problematic parts in the .json file which contain the actual popular times themselves. For example, the information for Monday is located in a list with arrays containing a list with multiple data. To access that information within the res variable one needs to create a for loop which will gather all the 24-time stamps for that specific day. The following chunk of code represents the implementation for Monday. However, this needs to be done for all the weekdays and types.

```
1         for c in range(6, 30):
2             try:
3                 ws.write(i, c, res["populartimes"][0]["data"][c
- 6])
4             except:
5                 ws.write(i, c, "NA")
```

6.4 Create a Pie Chart

The first script to represent the collected data in a graphical manner will be the pie chart, which will represent all the categories of the location businesses in percentages. With the use of pandas and matplotlib libraries on can get the count of all the types of businesses and save them in a variable corresponding to their type, for example, “restaurants”.

With the count being saved, one proceeds by creating labels and lists for all the categories which then are used to plot the pie chart. The pie chart will use the color scheme “ggplot”. To be able to see the percentages, the string “%.2f %%” needs to be included in the code under “autopct”. For an even better graphical representation, the explode statement will be used to break the graph to an extent.

After creating the title, the plot will be shown and saved in the current folder where the script and all the other files are located. In this thesis the actual pie chart will be presented in the “Results” chapter.

```
1    import pandas as pd
2    import matplotlib.pyplot as plt
3
4    popular = pd.read_excel("popular.xls")
5
6    plt.style.use("ggplot")
7
8    restaurant = popular.loc[popular["types_1"] ==
"restaurant"].count()[0]
9    bar = popular.loc[popular["types_1"] == "bar"].count()[0]
10   meal_takeaway = popular.loc[popular["types_1"] ==
"meal_takeaway"].count()[0]
11   cafe = popular.loc[popular["types_1"] == "cafe"].count()[0]
12
13   labels = ["Restaurants", "Bars", "Meal Takeaways", "Cafes"]
14   businesses = [restaurant, bar, meal_takeaway, cafe]
15   explode = (0, 0, .2, .1)
16
17   plt.pie(businesses, labels=labels, autopct="%.2f %%",
explode=explode)
18   plt.title("Types of Businesses")
19   plt.show()
20   plt.savefig('piechart.png')
21
```

6.5 Average Ratings

Another simple method to represent the attractiveness of the local businesses is their ratings. For that, a simple program that will represent the average rating for different types of local businesses will be described in this section.

First, one needs to create a rating list that will contain the averages for the different categories and variables which will be used to collect the individual rating followed by variables to count the available number of ratings for every category as well as one for the averages.

After that, one needs to create a for loop that will go through all the rows, which will search every type of local business and collect the ratings if they are at disposal. If that is the case, the rest variable will be incremented by the rating value and the n variable incremented by one. Finally, after the loop, one proceeds by dividing the rest variable by the n variable and append it to the rating list. The try and except statements will avoid any conflict if there are no restaurants found in the data.

```
    for index, row in popular.iterrows():
        if popular.types_1[index] == "restaurant":
            if math.isnan(popular.rating[index]) is False:
                rest += popular.rating[index]
                n += 1
    try:
        average = rest / n
        ratings.append(average)
    except:
        average = 0
        ratings.append(average)
```

The program described above is only a part of the program, which has the purpose of collecting and calculating the average rating for the restaurants. The entire program for all the average ratings in the different categories can be visualized in the Appendix and the results in the “Results” chapter. To finish the program, one only needs to call the bar method and provide the bins list which contains the names and the rating list. Creating an improved Y-axis also improves the visualization of the average ratings.

```
    plt.bar(bins, ratings)
    plt.yticks(np.arange(0, 5.5, 0.5))
    plt.title("Average Ratings")
    plt.savefig('ratings.png')
    plt.show()
```

6.6 Popular Times Line Graph

A line graph is needed, to generate a better visualization of the popular times located in the new updated excel file. A simple approach is to gather the data for three restaurants in the data set and plot the popular times and the average. The entire program is located in the Appendix. However, some code chunks will be explained in further detail in this section.

To start with the usual procedure needs to be done, which consists of creating the needed lists, variables and reading the excel file. The next step consists of creating a loop that ranges from all the daily hours from Monday to Saturday followed by a loop which goes through all the rows.

```
1     for c in range(6, 150):
2         n = 0
3         sums = 0
4         average = 0
5
6         for index, row in popular.iterrows():
```

In this, loop in a loop, the restaurants get located and their respective popular times added to the corresponding lists. While using the variable n and if statements. When the variable n arrives at the value of three the average of the previous located popular times gets calculated and added to the average list and the inner loop is stopped. This procedure happens until the outer loop is finished and all popular times are collected.

```
1                 if math.isnan(popular.iloc[index, c]) is False:
2                     sums += popular.iloc[index, c]
3                     if n == 0:
4                         rest_0.append(popular.iloc[index, c])
5                     if n == 1:
6                         rest_1.append(popular.iloc[index, c])
7                     if n == 2:
8                         rest_2.append(popular.iloc[index, c])
9                     if n == 3:
10                        rest_3.append(popular.iloc[index, c])
11                    n += 1
12                    if n == 4:
13                        try:
14                            average = sums / n
15                            rest_avg.append(average)
16                        except:
17                            average = 0
18                            rest_avg.append(average)
19                        break
```

6.7 Average Popular Times Line Graph

This line graph is similar to the previous one, and the entire program is located in the Appendix. The difference here consists of how the different rows are addressed since the goal of this program is to create the average popular times for restaurants or meal takeaways and cafes or bars. For this, an if statement is used to determine if the current row is a restaurant or meal takeaway. If that is the case, the popular times get added to a sum variable. The average of the sum gets calculated and then added to the list, which will happen for all the daily hours from Monday to Saturday. The same applies to cafes and bars.

```
1     for c in range(6, 150):
2         n = 0
3         sums = 0
4         average = 0
5
6         for index, row in popular.iterrows():
7             if popular.types_1[index] == "restaurant" or
popular.types_1[index] == "meal_takeaway":
8                 if math.isnan(popular.iloc[index, c]) is False:
9                     sums += popular.iloc[index, c]
10                    n += 1
11        try:
12            average = sums / n
13            food_average.append(average)
14        except:
15            average = 0
16            food_average.append(average)
```

Before creating the plot, the numpy library is used to create time stamps ranging from 0 to 144 in increments of 1 which will represent all the hours from Monday to Saturday. Then the plots are created with the respective labels, title and the desired ticks.

```
1     time_stamps = np.arange(0, 144, 1)
2
3     plt.plot(time_stamps, food_average, label="Restaurants &
Meal Takeaways")
4     plt.plot(time_stamps, drinks_average, label="Cafes & Bars")
5     plt.title("Average Popularity of Local Businesses")
6     plt.xlabel("Monday to Saturday Weekly Hours")
7     plt.ylabel("Popularity in %")
8     plt.xticks(np.arange(0, 145, 12))
9     plt.yticks(np.arange(0, 101, 5))
10    plt.legend()
11    plt.savefig('linegraph.png')
12    plt.show()
```

6.8 Percentage of Open Local Businesses

This program will allow visualization of the weighted opening hours through the week for different categories. Every same hour of the different days will be analyzed, added together, and the average calculated based upon the given information if the business is open or closed. That average then gets represented in percentage and plotted for every type of business.

First, the intermediate list and final list for the different categories need to be created. The next step consists of counting every business in the excel file, which contains popular times.

```
1     for index, row in popular.iterrows():
2         if math.isnan(popular.Monday_0[index]) is False:
3             count += 1
```

After that count, the process of having a loop in a loop is used for the first 24 hours, and the gathering of the opening hours can be started. For example, in the category of meal takeaways, the opening hours are located on the final list.

```
1     for c in range(6, 30):
2
3         # For Meal_Takeaways
4         n = 0
5         for index, row in popular.iterrows():
6             if popular.types_1[index] == "meal_takeaway":
7                 if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
8                     n += 1
9         # Add Percentage to final  Meal-Takeaway list
10        meal_takeaways_final.append(n / count)
```

In the following next 24 hours representing Tuesday with a different range, the intermediate list gets used to gather the opening hours.

```
1         # Create Percentage of current day as a list
2         meal_takeaways.append(n / count)
```

Finally, at the end of Tuesday the intermediate list is added to the final list of the meal takeaways, divided by 2 and the intermediate list cleared for further use. The above steps are done for all different categories and days which can be seen in the Appendix.

```
    # Add Percentage to final list and divide by 2 to create the
average while maintaining it as a list
    meal_takeaways_final = (np.add(meal_takeaways_final,
meal_takeaways) / 2).tolist()
    cafes_final = (np.add(cafes_final, cafes) / 2).tolist()
    bars_final = (np.add(bars_final, bars) / 2).tolist()
    restaurants_final = (np.add(restaurants_final, restaurants)
/ 2).tolist()
    # Delete intermediate list
    meal_takeaways = []
    cafes = []
    bars = []
    restaurants = []
```

After creating all the lists with the opening hours, one needs to create a weighted bar chart to have a better visualization. To be able to do that one needs to specify at which position the category needs to be plotted during bar chart creation seen below.

```
    # Creating the plot
    time_stamps = np.arange(0, 24, 1)

    plt.bar(time_stamps+.5, restaurants_final,
label="Restaurants")
    plt.bar(time_stamps+.5, bars_final,
bottom=restaurants_final, label="Bars")
    plt.bar(time_stamps+.5, cafes_final, label="Cafes",
bottom=[sum(x) for x in zip(bars_final, restaurants_final)])
    plt.bar(time_stamps+.5, meal_takeaways_final, label="Meal
Takeaways", bottom=[sum(x) for x in zip(bars_final,

restaurants_final,

cafes_final)])
    plt.title("Opening Hours of Different Local Businesses")
    plt.xlabel("Daily Hours")
    plt.ylabel("Open Local Businesses in %")
    plt.legend()
    plt.xticks(np.arange(0, 25, 6))
    plt.yticks(np.arange(0, 101, 10))
    plt.savefig('openlbs.png')
    plt.show()

```

7 Results

7.1 Types and Average Ratings

With the concluded programs and the different visualization programs implemented, one can start illustrating the behavioral pattern of the people of the Metropolitan City of Bologna in Italy. Figure 9 shows that there is a dominant number of restaurants compared to the other local businesses with the minority being meal takeaways.

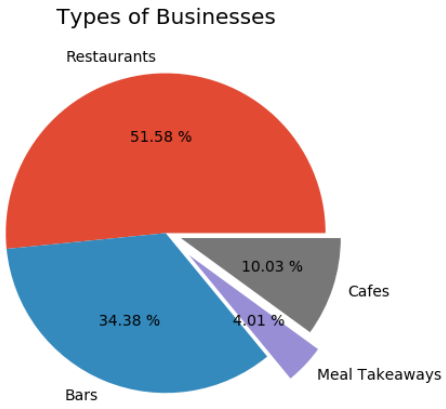

figure 9

The restaurants and cafes are also higher rated compared to the other local businesses with the meal takeaways being the worst-rated category, as seen in the following figure.

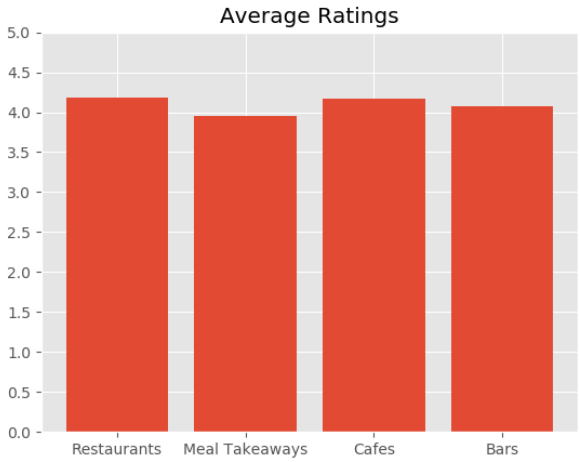

figure 10

7.2 Popular Times and the Different Averages

The actual visual representation of the popular times can be seen in the following figure, which consists of 3 different restaurants and their average popular times which shows their popularity from Monday to Saturday in weekly hours ranging from 0 to 144. Sundays are not considered since most local businesses are closed or not available in the gathered data.

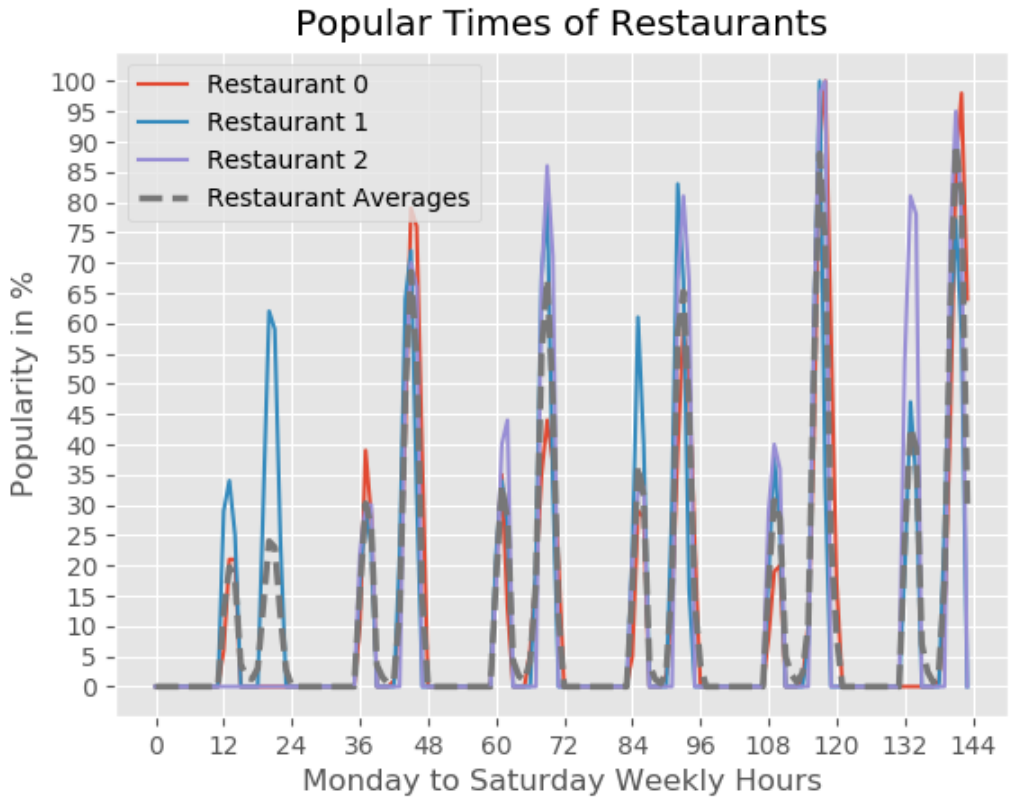

figure 11

In the graph, one can clearly see that there is a pattern in terms of popularity during lunch and dinner time. This graph also shows that there are different opening hours for restaurants. For example, the Restaurant 2 is closed on Monday while Restaurant 0 is closed on Saturday during lunchtime while the Restaurant 1 is open for every lunch and dinner during the week. The average popularity is a good line of representation to determine if a restaurant at given times is performing over or under the average.

Another interesting visualization is one that calculates the average of all the types of local businesses and produces their average popular times, as seen below.

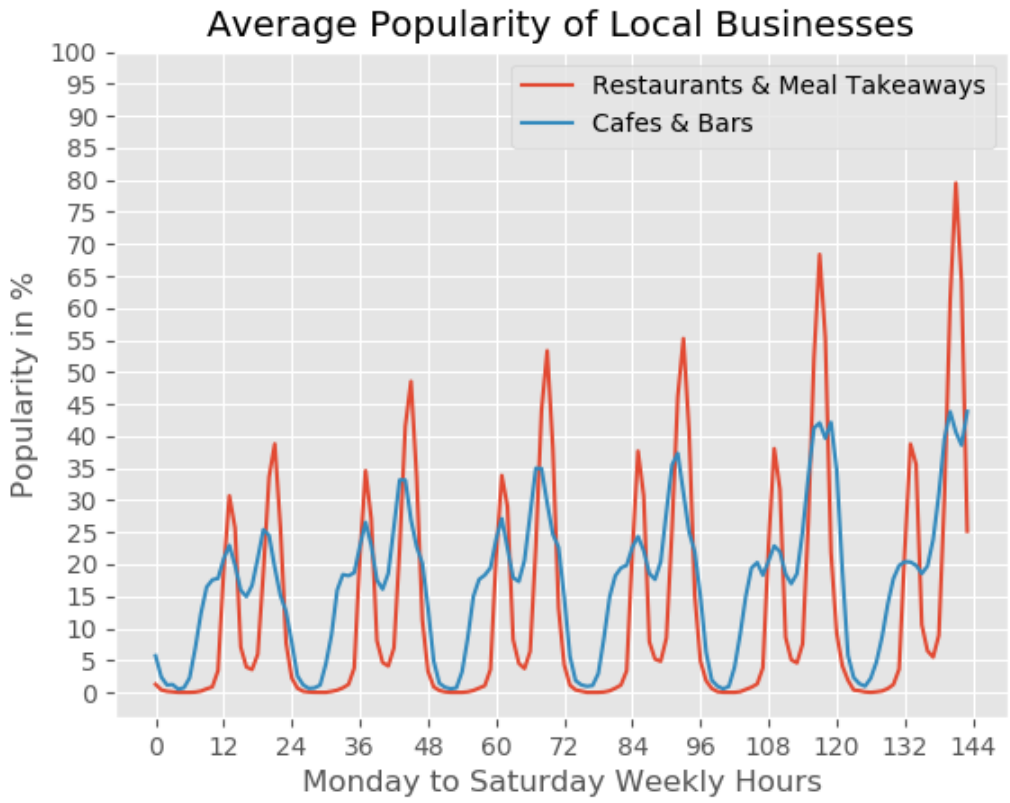

figure 12

In this graph, one can see that cafes and bars are busy in the morning and afternoon while restaurants and meal takeaways are quiet. However, there is an apparent popularity increase at lunchtime in restaurants and meal takeaways. As the popular times approach the end of the week, the spikes grow. The bars and cafe popularity only slightly increases at the weekends but remains busy for a more extended period.

One can also see that people prefer to go to a bar or cafe before having dinner in a restaurant, especially during the week. All those patterns indicate that there is a popular nightlife while maintaining stable popularity at lunchtimes during working days.

7.3 Opening Hours for Different Categories

The opening hours for different categories also represent an interesting visualization that shows how local businesses work during the day. As seen in figure 13, Restaurants have peak opening times during lunch and dinner, similar to the meal takeaways.

By taking a closer look at the bars and cafes, one can see that their opening times remain constant during the day, which indicates that their opening times are between morning and evening. However, the bars seem to stay open until midnight while the cafes close around lunchtime.

The graph also shows that the maximum of open local businesses in that area reside around lunchtime, which also seems to be where the local businesses are the most popular and the businesses want to benefit from it.

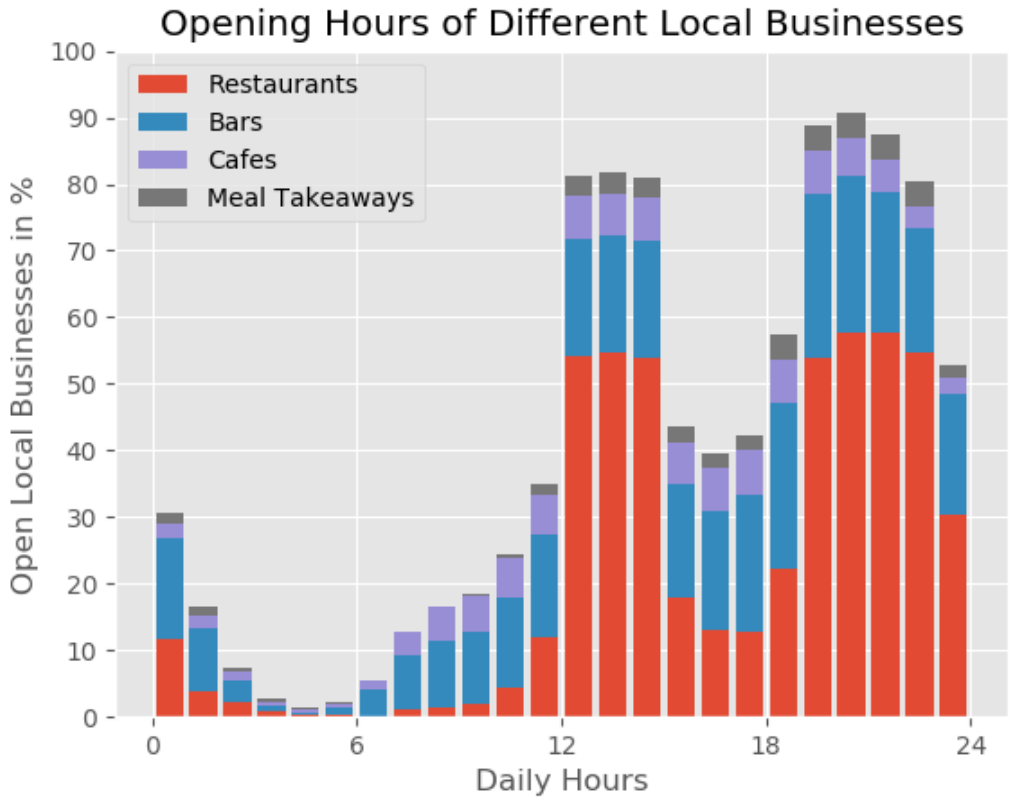

figure 13

8 Conclusion

In this thesis, a mobile crowdsensing application such as Google has been presented, which produces highly accurate popular times. With the help of open-source programs such as OpenStreetMap HOT export Tool, it is possible to generate the first collection of diverse points of interest.

With such large dataset, one uses the populartimes library to access Google Popular Times, which enables a developer or data scientist to design tools which facilitate the design of urban environments.

A developer can illustrate such datasets with the help of Python and specific libraries by creating any visualization programs which illustrate the behavioral pattern of people in the desired areas. On the other hand, a developer can also analyze how local businesses operate in those areas and how the different business categories set their opening hours.

With all this in hand, using mobile crowdsensing applications enables modern data analysis to solve any urban environment issues.

9 References

[1] Raghu K. Ganti, Fan Ye, and Hui Lei. Mobile Crowdsensing: Current State and Future Challenges, pages 1-7.

[2] Andrea Capponi, Piergiorgio Vitello, Claudio Fiandrino, Guido Cantelmo, Dzmitry Kliazovich, Ulrich Sorger, Pascal Bouvry. Mobile CrowdLearning: Highly-Accuracy Prediction of Local Business Category and Attractiveness, pages 1-4.

[3] Mohammad Abu Alsheikh, Yutao Jiao, Dusit Niyato, Ping Wang, Derek Leong, and Zhu Han. The Accuracy-Privacy Tradeoff of Mobile Crowdsensing, page 3.

[4] Software Libraries. https://www.techopedia.com/definition/3828/software-library

[5] Google. https://web.archive.org/web/20150116073513/https://www.google.com/intl/en/about/company/

[6] Google APIs for various languages. https://developers.google.com/api-client-library/

[7] Google Place ID Finder. https://developers.google.com/places/place-id

[8] Google Popular Times. https://support.google.com/business/answer/6263531?hl=en

[9] SafeGraph. Determining Point-of -Interest Visits From Location Data: A Technical Guide To Visit Attribution, pages 3-17.

[10] Python Website. https://www.python.org/

[11] JetBrains for PyCharm. https://www.jetbrains.com/pycharm/download/#section=Windows

[12] Pip. https://bootstrap.pypa.io/get-pip.py

[13] Python Core Library. https://docs.python.org/3/library/

[14] Komoot. https://www.komoot.com/features

[15] OpenStreetMap. https://wiki.openstreetmap.org/wiki/About_OpenStreetMap

[16] HOT Export Tool. https://export.hotosm.org/en/v3/

[17] Populartimes Library. https://github.com/m-wrzr/populartimes

[18] Pandas Library. https://pandas.pydata.org/

[19] Xlwt Library. https://xlwt.readthedocs.io/en/latest/

[20] Matplotlib Library. https://matplotlib.org/

[21] Numpy. https://docs.scipy.org/doc/numpy/reference/

[22] Get API Key. https://developers.google.com/places/web-service/get-api-key

[23] Usage and Billing. https://developers.google.com/places/web-service/usage-and-billing

[24] Google Place ID Finder. https://developers.google.com/maps/documentation/javascript/examples/places-placeid-finder

10 Appendix

10.1 .json File


```
{
  "id": "ChIJSYuuSx9awokRyrrOFTGg0GY",
  "name": "Gran Morsi",
  "address": "22 Warren St, New York, NY 10007, USA",
  "types": [
    "restaurant",
    "food",
    "point_of_interest",
    "establishment"
  ],
  "coordinates": {
    "lat": 40.71431500000001,
    "lng": -74.007766
  },
  "rating": 4.4,
  "rating_n": 129,
  "international_phone_number": "+1 212-577-2725",
    "time_spent": [
    90,
    180
  ],
  "current_popularity": 33,
  "populartimes": [
    {
      "name": "Monday",
      "data": [
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 19, 20, 17, 0,
0, 20, 28, 26, 18, 10, 6, 0
      ]
    },
    {
      "name": "Tuesday",
      "data": [
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 25, 27, 19, 10, 0,
0, 34, 42, 42, 35, 26, 15, 0
      ]
    },
    {
      "name": "Wednesday",
      "data": [
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 28, 34, 23, 13, 0,
0, 36, 46, 47, 39, 26, 13, 0
      ]
    },
    {
      "name": "Thursday",
      "data": [
```

```
45            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 28, 42, 42, 28, 0,
0, 59, 61, 46, 39, 32, 20, 0
46          ]
47        },
48        {
49          "name": "Friday",
50          "data": [
51            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 30, 44, 40, 28, 0,
0, 70, 96, 100, 80, 48, 22, 0
52          ]
53        },
54        {
55          "name": "Saturday",
56          "data": [
57            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
34, 42, 48, 47, 36, 21, 0
58          ]
59        },
60        {
61          "name": "Sunday",
62          "data": [
63            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
27, 34, 34, 28, 21, 10, 0
64          ]
65        }
66      ],
67      "time_wait": [
68        {
69          "name": "Monday",
70          "data": [
71            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 15, 15, 0, 0, 0, 0,
15, 15, 15, 0, 15, 15, 0
72          ]
73        },
74        {
75          "name": "Tuesday",
76          "data": [
77            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 15, 15, 0, 0, 0, 0,
0, 15, 15, 15, 15, 15, 0
78          ]
79        },
80        {
81          "name": "Wednesday",
82          "data": [
83            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 15, 0, 0, 0, 0, 0,
0, 15, 15, 15, 15, 15, 0
84          ]
85        },
86        {
87          "name": "Thursday",
88          "data": [
```

```
89            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 15, 15, 0, 0, 0, 0,
0, 15, 15, 15, 15, 15, 0
90          ]
91        },
92        {
93          "name": "Friday",
94          "data": [
95            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
15, 15, 15, 15, 15, 15, 0
96          ]
97        },
98        {
99          "name": "Saturday",
100         "data": [
101           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
15, 15, 15, 15, 15, 15, 0
102         ]
103       },
104       {
105         "name": "Sunday",
106         "data": [
107           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
15, 15, 15, 15, 0, 0, 0
108         ]
109       }
110     ]
111   }
```

10.2 app.py File

```
#
# Application to retrieve popular times and include them in
# a new excel file
import populartimes

import pandas as pd

import xlwt

apik = "your-api-key"


# Reading/creating excel file/sheet
df = pd.read_excel("Bologna_Centro.xlsx", sheet_name=0)

wb = xlwt.Workbook()
ws = wb.add_sheet("Times")
ws2 = wb.add_sheet("Time_Wait")

# Creating the column titles
ws.write(0, 0, "address")
ws.write(0, 1, "lat")
ws.write(0, 2, "lng")
ws.write(0, 3, "id")
ws.write(0, 4, "international_phone_number")
ws.write(0, 5, "name")

for c in range(6, 30):
    ws.write(0, c, f"Monday {abs(6-c)}")
for c in range(30, 54):
    ws.write(0, c, f"Tuesday {abs(30-c)}")
for c in range(54, 78):
    ws.write(0, c, f"Wednesday {abs(54-c)}")
for c in range(78, 102):
    ws.write(0, c, f"Thursday {abs(78-c)}")
for c in range(102, 126):
    ws.write(0, c, f"Friday {abs(102-c)}")
for c in range(126, 150):
    ws.write(0, c, f"Saturday {abs(126-c)}")
for c in range(150, 174):
    ws.write(0, c, f"Sunday {abs(150-c)}")

ws.write(0, 174, "rating")
ws.write(0, 175, "rating_n")
ws.write(0, 176, "search_term")

for c in range(177, 184):
    ws.write(0, c, f"types {abs(177-c)}")

```

```
50    i = 0
51
52    # For-loop for all the indexes
53    for index, row in df.iterrows():
54
55        # Part for sheet number 1
56        try:
57            res = populartimes.get_id(apik, row["Google_Id"])
58            print(f"Index: {index} Google_ID:
{row['Google_Id']}")
59            i += 1
60        except:
61            print(f"Error on Index: {index} Google_ID:
{row['Google_Id']}")
62            continue
63
64        try:
65            ws.write(i, 0, res["address"])
66        except:
67            ws.write(i, 0, "NA")
68
69        try:
70            ws.write(i, 1, res["coordinates"]["lat"])
71        except:
72            ws.write(i, 1, "NA")
73
74        try:
75            ws.write(i, 2, res["coordinates"]["lng"])
76        except:
77            ws.write(i, 2, "NA")
78
79        try:
80            ws.write(i, 3, res["id"])
81        except:
82            ws.write(i, 3, "NA")
83
84        try:
85            ws.write(i, 4, res["international_phone_number"])
86        except:
87            ws.write(i, 4, "NA")
88
89        try:
90            ws.write(i, 5, res["name"])
91        except:
92            ws.write(i, 5, "NA")
93
94        # Monday
95        for c in range(6, 30):
96            try:
97                ws.write(i, c, res["populartimes"][0]["data"][c
- 6])
98            except:
```

```
99                ws.write(i, c, "NA")
100       # Tuesday
101       for c in range(30, 54):
102           try:
103               ws.write(i, c, res["populartimes"][1]["data"][c
- 30])
104           except:
105               ws.write(i, c, "NA")
106
107       # Wednesday
108       for c in range(54, 78):
109           try:
110               ws.write(i, c, res["populartimes"][2]["data"][c
- 54])
111           except:
112               ws.write(i, c, "NA")
113
114       # Thursday
115       for c in range(78, 102):
116           try:
117               ws.write(i, c, res["populartimes"][3]["data"][c
- 78])
118           except:
119               ws.write(i, c, "NA")
120
121       # Friday
122       for c in range(102, 126):
123           try:
124               ws.write(i, c, res["populartimes"][4]["data"][c
- 102])
125           except:
126               ws.write(i, c, "NA")
127
128       # Saturday
129       for c in range(126, 150):
130           try:
131               ws.write(i, c, res["populartimes"][5]["data"][c
- 126])
132           except:
133               ws.write(i, c, "NA")
134
135       # Sunday
136       for c in range(150, 174):
137           try:
138               ws.write(i, c, res["time_wait"][6]["data"][c -
150])
139           except:
140               ws.write(i, c, "NA")
141
142       try:
143           ws.write(i, 174, res["rating"])
144       except:
```

```
145             ws.write(i, 174, "NA")
146
147         try:
148             ws.write(i, 175, res["rating_n"])
149         except:
150             ws.write(i, 175, "NA")
151
152         try:
153             ws.write(i, 176, res["searchterm"])
154         except:
155             ws.write(i, 176, "NA")
156
157         try:
158             ws.write(i, 177, str(res["time_spent"]))
159         except:
160             ws.write(i, 177, "NA")
161
162         for c in range(178, 178 + len(res["types"])):
163             try:
164                 ws.write(i, c, res["types"][c - 178])
165             except:
166                 ws.write(i, c, 0)
167
168         # Part for Sheet number 2
169         # Monday
170         for c in range(6, 30):
171             try:
172                 ws2.write(i, c, res["time_wait"][0]["data"][c -
6])
173             except:
174                 ws2.write(i, c, "NA")
175         # Tuesday
176         for c in range(30, 54):
177             try:
178                 ws2.write(i, c, res["time_wait"][1]["data"][c -
30])
179             except:
180                 ws2.write(i, c, "NA")
181
182         # Wednesday
183         for c in range(54, 78):
184             try:
185                 ws2.write(i, c, res["time_wait"][2]["data"][c -
54])
186             except:
187                 ws2.write(i, c, "NA")
188
189         # Thursday
190         for c in range(78, 102):
191             try:
192                 ws2.write(i, c, res["time_wait"][3]["data"][c -
78])
```

```
193             except:
194                 ws2.write(i, c, "NA")
195 
196         # Friday
197         for c in range(102, 126):
198             try:
199                 ws2.write(i, c, res["time_wait"][4]["data"][c -
102])
200             except:
201                 ws2.write(i, c, "NA")
202 
203         # Saturday
204         for c in range(126, 150):
205             try:
206                 ws2.write(i, c, res["time_wait"][5]["data"][c -
126])
207             except:
208                 ws2.write(i, c, "NA")
209 
210         # Sunday
211         for c in range(150, 174):
212             try:
213                 ws2.write(i, c, res["time_wait"][6]["data"][c -
150])
214             except:
215                 ws2.write(i, c, "NA")
216 
217         try:
218             ws2.write(i, 175, res["current_popularity"])
219         except:
220             ws2.write(i, 175, "NA")
221 
222     # Save the final excel file
223     wb.save("popular.xls")
224 
```

10.3 ratings.py File

```
import pandas as pd
import matplotlib.pyplot as plt
import math
import numpy as np

popular = pd.read_excel("popular.xls")

plt.style.use("ggplot")

bins = ["Restaurants", "Meal Takeaways", "Cafes", "Bars"]
ratings = []
rest = 0
mt = 0
cafe = 0
bar = 0

n = 0
average = 0
for index, row in popular.iterrows():
    if popular.types_1[index] == "restaurant":
        if math.isnan(popular.rating[index]) is False:
            rest += popular.rating[index]
            n += 1
try:
    average = rest / n
    ratings.append(average)
except:
    average = 0
    ratings.append(average)

n = 0
average = 0
for index, row in popular.iterrows():
    if popular.types_1[index] == "meal_takeaway":
        if math.isnan(popular.rating[index]) is False:
            mt += popular.rating[index]
            n += 1
try:
    average = mt / n
    ratings.append(average)
except:
    average = 0
    ratings.append(average)

n = 0
average = 0
for index, row in popular.iterrows():
    if popular.types_1[index] == "cafe":
        if math.isnan(popular.rating[index]) is False:
```

```
50                cafe += popular.rating[index]
51                n += 1
52    try:
53        average = cafe / n
54        ratings.append(average)
55    except:
56        average = 0
57        ratings.append(average)
58
59    n = 0
60    average = 0
61    for index, row in popular.iterrows():
62        if popular.types_1[index] == "bar":
63            if math.isnan(popular.rating[index]) is False:
64                bar += popular.rating[index]
65                n += 1
66    try:
67        average = bar / n
68        ratings.append(average)
69    except:
70        average = 0
71        ratings.append(average)
72
73    plt.bar(bins, ratings)
74    plt.yticks(np.arange(0, 5.5, 0.5))
75    plt.title("Average Ratings")
76    plt.savefig('ratings.png')
77    plt.show()
78
```

10.4 linegraph2.py File

```
import pandas as pd
import math
import numpy as np
import matplotlib.pyplot as plt

plt.style.use("ggplot")

rest_0 = []
rest_1 = []
rest_2 = []
rest_3 = []
rest_avg = []

popular = pd.read_excel("popular.xls")

# Line for Restaurants
for c in range(6, 150):
    n = 0
    sums = 0
    average = 0

    for index, row in popular.iterrows():
        if popular.types_1[index] == "restaurant":
            if math.isnan(popular.iloc[index, c]) is False:
                sums += popular.iloc[index, c]

                if n == 0:
                    rest_0.append(popular.iloc[index, c])
                if n == 1:
                    rest_1.append(popular.iloc[index, c])
                if n == 2:
                    rest_2.append(popular.iloc[index, c])
                if n == 3:
                    rest_3.append(popular.iloc[index, c])

                n += 1

                if n == 4:
                    try:
                        average = sums / n
                        rest_avg.append(average)
                    except:
                        average = 0
                        rest_avg.append(average)
                    break

# Create plots
time_stamps = np.arange(0, 144, 1)

```

```
50   plt.plot(time_stamps, rest_0, label="Restaurant 0")
51   plt.plot(time_stamps, rest_1, label="Restaurant 1")
52   plt.plot(time_stamps, rest_2, label="Restaurant 2")
53   plt.plot(time_stamps, rest_avg, label="Restaurant Averages",
linestyle="dashed", linewidth=2.5)
54   plt.title("Popular Times of Restaurants")
55   plt.xlabel("Monday to Saturday Weekly Hours")
56   plt.ylabel("Popularity in %")
57   plt.xticks(np.arange(0, 145, 12))
58   plt.yticks(np.arange(0, 101, 5))
59   plt.legend()
60   plt.savefig('linegraph2.png')
61   plt.show()
62
```

10.5 linegraph.py File

```
1    import pandas as pd
2    import math
3    import numpy as np
4    import matplotlib.pyplot as plt
5
6    plt.style.use("ggplot")
7
8    food_average = []
9    drinks_average = []
10
11   popular = pd.read_excel("popular.xls")
12
13   # Line for Restaurants and Meal Takeaways
14   for c in range(6, 150):
15       n = 0
16       sums = 0
17       average = 0
18
19       for index, row in popular.iterrows():
20           if popular.types_1[index] == "restaurant" or
popular.types_1[index] == "meal_takeaway":
21               if math.isnan(popular.iloc[index, c]) is False:
22                   sums += popular.iloc[index, c]
23                   n += 1
24       try:
25           average = sums / n
26           food_average.append(average)
27       except:
28           average = 0
29           food_average.append(average)
30
31   # Line for Bars and Cafes
32   for c in range(6, 150):
33       n = 0
34       sums = 0
35       average = 0
36
37       for index, row in popular.iterrows():
38           if popular.types_1[index] == "cafe" or
popular.types_1[index] == "bar":
39               if math.isnan(popular.iloc[index, c]) is False:
40                   sums += popular.iloc[index, c]
41                   n += 1
42       try:
43           average = sums / n
44           drinks_average.append(average)
45       except:
46           average = 0
47           drinks_average.append(average)
```

```
48
49    # Create plots
50    time_stamps = np.arange(0, 144, 1)
51
52    plt.plot(time_stamps, food_average, label="Restaurants &
Meal Takeaways")
53    plt.plot(time_stamps, drinks_average, label="Cafes & Bars")
54    plt.title("Average Popularity of Local Businesses")
55    plt.xlabel("Monday to Saturday Weekly Hours")
56    plt.ylabel("Popularity in %")
57    plt.xticks(np.arange(0, 145, 12))
58    plt.yticks(np.arange(0, 101, 5))
59    plt.legend()
60    plt.savefig('linegraph.png')
61    plt.show()
62
```

10.6 openlbs.py File

```
1    import pandas as pd
2    import matplotlib.pyplot as plt
3    import math
4    import numpy as np
5
6    plt.style.use("ggplot")
7
8    popular = pd.read_excel("popular.xls")
9
10   restaurants = []
11   restaurants_final = []
12   meal_takeaways = []
13   meal_takeaways_final = []
14   cafes = []
15   cafes_final = []
16   bars = []
17   bars_final = []
18
19   n = 0
20   count = 0
21
22   # Count all the Businesses that have popular times to be
able to calculate the percentages
23   # by ignoring all NA
24   for index, row in popular.iterrows():
25       if math.isnan(popular.Monday_0[index]) is False:
26           count += 1
27
28   # Monday - Count all open LB's for different categories
29   for c in range(6, 30):
30
31       # For Meal_Takeaways
32       n = 0
33       for index, row in popular.iterrows():
34           if popular.types_1[index] == "meal_takeaway":
35               if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
36                   n += 1
37       # Add Percentage to final  Meal-Takeaway list
38       meal_takeaways_final.append(n / count)
39
40       # For Cafes
41       n = 0
42       for index, row in popular.iterrows():
43           if popular.types_1[index] == "cafe":
44               if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
45                   n += 1
46       # Add Percentage to final cafe list
```

```
47        cafes_final.append(n / count)
48
49        # For Bars
50        n = 0
51        for index, row in popular.iterrows():
52            if popular.types_1[index] == "bar":
53                if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
54                    n += 1
55        # Add Percentage to final bars list
56        bars_final.append(n / count)
57
58        # For Restaurants
59        n = 0
60        for index, row in popular.iterrows():
61            if popular.types_1[index] == "restaurant":
62                if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
63                    n += 1
64        # Add Percentage to final restaurants list
65        restaurants_final.append(n / count)
66
67    # Tuesday - Count all open LB's for different categories
68    for c in range(30, 54):
69
70        # For Meal_Takeaways
71        n = 0
72        for index, row in popular.iterrows():
73            if popular.types_1[index] == "meal_takeaway":
74                if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
75                    n += 1
76        # Create Percentage of current day as a list
77        meal_takeaways.append(n / count)
78
79        # For Cafes
80        n = 0
81        for index, row in popular.iterrows():
82            if popular.types_1[index] == "cafe":
83                if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
84                    n += 1
85        # Create Percentage of current day as a list
86        cafes.append(n / count)
87
88        # For Bars
89        n = 0
90        for index, row in popular.iterrows():
91            if popular.types_1[index] == "bar":
92                if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
93                    n += 1
```

```
94        # Create Percentage of current day as a list
95        bars.append(n / count)
96
97        # For Restaurants
98        n = 0
99        for index, row in popular.iterrows():
100           if popular.types_1[index] == "restaurant":
101               if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
102                   n += 1
103       # Create Percentage of current day as a list
104       restaurants.append(n / count)
105
106   # Add Percentage to final list and divide by 2 to create the
average while maintaining it as a list
107   meal_takeaways_final = (np.add(meal_takeaways_final,
meal_takeaways) / 2).tolist()
108   cafes_final = (np.add(cafes_final, cafes) / 2).tolist()
109   bars_final = (np.add(bars_final, bars) / 2).tolist()
110   restaurants_final = (np.add(restaurants_final, restaurants)
/ 2).tolist()
111   # Delete intermediate list
112   meal_takeaways = []
113   cafes = []
114   bars = []
115   restaurants = []
116
117   # Wednesday - Count all open LB's for different categories
118   for c in range(54, 78):
119
120       # For Meal_Takeaways
121       n = 0
122       for index, row in popular.iterrows():
123           if popular.types_1[index] == "meal_takeaway":
124               if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
125                   n += 1
126       # Create Percentage of current day as a list
127       meal_takeaways.append(n / count)
128
129       # For Cafes
130       n = 0
131       for index, row in popular.iterrows():
132           if popular.types_1[index] == "cafe":
133               if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
134                   n += 1
135       # Create Percentage of current day as a list
136       cafes.append(n / count)
137
138       # For Bars
139       n = 0
```

```
140      for index, row in popular.iterrows():
141          if popular.types_1[index] == "bar":
142              if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
143                  n += 1
144      # Create Percentage of current day as a list
145      bars.append(n / count)
146
147      # For Restaurants
148      n = 0
149      for index, row in popular.iterrows():
150          if popular.types_1[index] == "restaurant":
151              if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
152                  n += 1
153      # Create Percentage of current day as a list
154      restaurants.append(n / count)
155
156  # Add Percentage to final list and divide by 2 to create the
average while maintaining it as a list
157  meal_takeaways_final = (np.add(meal_takeaways_final,
meal_takeaways) / 2).tolist()
158  cafes_final = (np.add(cafes_final, cafes) / 2).tolist()
159  bars_final = (np.add(bars_final, bars) / 2).tolist()
160  restaurants_final = (np.add(restaurants_final, restaurants)
/ 2).tolist()
161  # Delete intermediate list
162  meal_takeaways = []
163  cafes = []
164  bars = []
165  restaurants = []
166
167  # Thursday - Count all open LB's for different categories
168  for c in range(78, 102):
169
170      # For Meal_Takeaways
171      n = 0
172      for index, row in popular.iterrows():
173          if popular.types_1[index] == "meal_takeaway":
174              if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
175                  n += 1
176      # Create Percentage of current day as a list
177      meal_takeaways.append(n / count)
178
179      # For Cafes
180      n = 0
181      for index, row in popular.iterrows():
182          if popular.types_1[index] == "cafe":
183              if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
184                  n += 1
```

```
185       # Create Percentage of current day as a list
186       cafes.append(n / count)
187
188       # For Bars
189       n = 0
190       for index, row in popular.iterrows():
191           if popular.types_1[index] == "bar":
192               if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
193                   n += 1
194       # Create Percentage of current day as a list
195       bars.append(n / count)
196
197       # For Restaurants
198       n = 0
199       for index, row in popular.iterrows():
200           if popular.types_1[index] == "restaurant":
201               if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
202                   n += 1
203       # Create Percentage of current day as a list
204       restaurants.append(n / count)
205
206   # Add Percentage to final list and divide by 2 to create the
average while maintaining it as a list
207   meal_takeaways_final = (np.add(meal_takeaways_final,
meal_takeaways) / 2).tolist()
208   cafes_final = (np.add(cafes_final, cafes) / 2).tolist()
209   bars_final = (np.add(bars_final, bars) / 2).tolist()
210   restaurants_final = (np.add(restaurants_final, restaurants)
/ 2).tolist()
211   # Delete intermediate list
212   meal_takeaways = []
213   cafes = []
214   bars = []
215   restaurants = []
216
217   # Friday - Count all open LB's for different categories
218   for c in range(102, 126):
219
220       # For Meal_Takeaways
221       n = 0
222       for index, row in popular.iterrows():
223           if popular.types_1[index] == "meal_takeaway":
224               if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
225                   n += 1
226       # Create Percentage of current day as a list
227       meal_takeaways.append(n / count)
228
229       # For Cafes
230       n = 0
```

```
231      for index, row in popular.iterrows():
232          if popular.types_1[index] == "cafe":
233              if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
234                  n += 1
235      # Create Percentage of current day as a list
236      cafes.append(n / count)
237
238      # For Bars
239      n = 0
240      for index, row in popular.iterrows():
241          if popular.types_1[index] == "bar":
242              if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
243                  n += 1
244      # Create Percentage of current day as a list
245      bars.append(n / count)
246
247      # For Restaurants
248      n = 0
249      for index, row in popular.iterrows():
250          if popular.types_1[index] == "restaurant":
251              if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
252                  n += 1
253      # Create Percentage of current day as a list
254      restaurants.append(n / count)
255
256  # Add Percentage to final list and divide by 2 to create the
average while maintaining it as a list
257  meal_takeaways_final = (np.add(meal_takeaways_final,
meal_takeaways) / 2).tolist()
258  cafes_final = (np.add(cafes_final, cafes) / 2).tolist()
259  bars_final = (np.add(bars_final, bars) / 2).tolist()
260  restaurants_final = (np.add(restaurants_final, restaurants)
/ 2).tolist()
261  # Delete intermediate list
262  meal_takeaways = []
263  cafes = []
264  bars = []
265  restaurants = []
266
267  # Saturday - Count all open LB's for different categories
268  for c in range(126, 150):
269
270      # For Meal_Takeaways
271      n = 0
272      for index, row in popular.iterrows():
273          if popular.types_1[index] == "meal_takeaway":
274              if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
275                  n += 1
```

```
276       # Create Percentage of current day as a list
277       meal_takeaways.append(n / count)
278
279       # For Cafes
280       n = 0
281       for index, row in popular.iterrows():
282           if popular.types_1[index] == "cafe":
283               if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
284                   n += 1
285       # Create Percentage of current day as a list
286       cafes.append(n / count)
287
288       # For Bars
289       n = 0
290       for index, row in popular.iterrows():
291           if popular.types_1[index] == "bar":
292               if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
293                   n += 1
294       # Create Percentage of current day as a list
295       bars.append(n / count)
296
297       # For Restaurants
298       n = 0
299       for index, row in popular.iterrows():
300           if popular.types_1[index] == "restaurant":
301               if math.isnan(popular.iloc[index, c]) is False
and popular.iloc[index, c] != 0:
302                   n += 1
303       # Create Percentage of current day as a list
304       restaurants.append(n / count)
305
306   # Add Percentage to final list and divide by 2 to create the
average while maintaining it as a list
307   # Multiply with 100 to be able to display as percentage
308   meal_takeaways_final = ((np.add(meal_takeaways_final,
meal_takeaways) / 2)*100).tolist()
309   cafes_final = ((np.add(cafes_final, cafes) /
2)*100).tolist()
310   bars_final = ((np.add(bars_final, bars) / 2)*100).tolist()
311   restaurants_final = ((np.add(restaurants_final, restaurants)
/ 2)*100).tolist()
312
313   # Creating the plot
314   time_stamps = np.arange(0, 24, 1)
315
316   plt.bar(time_stamps+.5, restaurants_final,
label="Restaurants")
317   plt.bar(time_stamps+.5, bars_final,
bottom=restaurants_final, label="Bars")
```

```
318  plt.bar(time_stamps+.5, cafes_final, label="Cafes",
bottom=[sum(x) for x in zip(bars_final, restaurants_final)])
319  plt.bar(time_stamps+.5, meal_takeaways_final, label="Meal
Takeaways", bottom=[sum(x) for x in zip(bars_final,
320
restaurants_final,
321
cafes_final)])
322  plt.title("Opening Hours of Different Local Businesses")
323  plt.xlabel("Daily Hours")
324  plt.ylabel("Open Local Businesses in %")
325  plt.legend()
326  plt.xticks(np.arange(0, 25, 6))
327  plt.yticks(np.arange(0, 101, 10))
328  plt.savefig('openlbs.png')
329  plt.show()
330
```