DAILY GROUND

Finding Your Ideal Cafe In Manhattan

Team Members:
Elina Shirolkar, Joy Shyu, Joy Tay, Meenu Selvakesari and Po-Wen Hsu

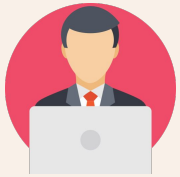
Background and Problem Statement

Problem Statement:

- People often face difficulties in finding a cafe, that is in close proximity and meets their specific requirements, such as price, type, and rating.
- Challenge of not having access to all the necessary information in one place, which can be time-consuming and frustrating.
- Need for a solution that simplifies the process of finding a perfect study spot.

Potential Use Cases:

Students looking for study spots



Remote workers looking for workspace

Tourists looking for local cafes

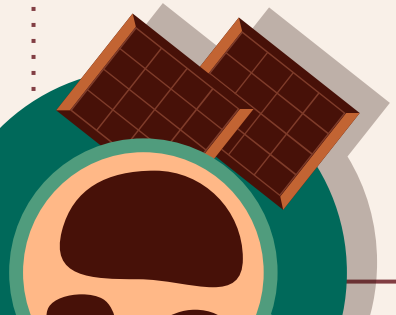
Social gatherings

Solution and Outcome of the Project

- **Solution:**
  Our goal is to develop a search engine, to help users find the most suitable cafe spot in Manhattan based on their preferences. For now, our search engine will allow users to filter for certain criteria such as types, neighborhood, price level, rating, and reviews.

  In the future, our target is to connect real-time data and offer more attributes such as ideal study place, wifi availability, power outlets, crowd level, and loudness to the users.

- **Outcome:**
  This will improve the productivity and studying experience of students who need a conducive study environment outside their homes.



DAILY GR ND

Data Source Specification and Procurement Details

1. **Data source:** Extracting cafe data by using Google Maps Places API

2. **Procurement Details:** Google offers a free tier for the Places API that includes up to 100,000 API calls per day. We note that it only allows 20 query results per call. Currently, we will not exceed this tier's limits but if we plan to scale this project in the future we may start incurring costs.

MONTHLY VOLUME RANGE (Price per CALL)		
0–100,000	100,001–500,000	500,000+
Places request cost + 0.003 USD per each	Places request cost + 0.0024 USD per each	Contact Sales for volume pricing

Found on Google Paid API Website

DAILY GRIND

Google Places API

Install googlemaps

```
!pip install googlemaps
```

We query cafe data using latitude and longitude

As we were only allowed 20 query results per call, we manually selected 10 locations that covered majority of Manhattan and queried them individually before concatenating all the data points.

The code shown retrieves cafes from the Upper West Side.

```
# collect 20 cafes from the 10 locations we have set in Manhattan
# Upper West Side 20 cafes
import googlemaps
from datetime import datetime

# Set up the Google Maps API client
gmaps = googlemaps.Client(key='AIzaSyAa81xKqSxG-Rh83POFL8Y-TcdXaLGi27k')

location = "40.7870, -73.9754"

# Set up the search parameters
uws_params = {
    'location': location,
    'radius': 300,
    'type': 'cafe'
}

# Use the Places API to search for cafes near the location
uws_results = gmaps.places_nearby(**uws_params)

#Print the name, address, and amenities of each cafe
for place in uws_results['results']:
    print(place['name'])
    if 'types' in place:
        print('Amenities:', ', '.join(place['types']))
    if 'opening_hours' in place:
        print('Open:', ', '.join(place['opening_hours']))
    print()
```

ETL Pipeline and Rationale

1. Python
get/clean data from API

- Efficient tool to retrieve data from the API
- Explore and clean the data before converting it into a pandas dataframe.

2. PostgreSQL
store and manage data

- Provides robust support for a wide range of data types
- Ideal for storing and establishing relationships between multiple datasets
- Provides strong data integrity and consistency, ensuring the data stored is accurate and reliable

3. Spark
query the attributes

- Handles large-scale data processing
- Has algorithms to build our machine learning models and perform query matching
- It supports real-time processing, allowing us to improve our search engine in the future

4. Flask
display the result

- Display the output, which connects to the result of our search engine

DAILY GROUND

Database Schema

- Our database schema is created in two places: the PostgreSQL database and the PySpark Dataframe.
- Our data was cleaned in python before inserting it into our schema

Upload CSV File and Creating Spark Dataframe

```
from pyspark.sql import SparkSession
import pandas as pd
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, DoubleType
filecontents = pd.read_csv('cafes1.csv')

schema = StructType([
    StructField("Name", StringType(), True),
    StructField("Address", StringType(), True),
    StructField("Latitude", DoubleType(), True),
    StructField("Longitude", DoubleType(), True),
    StructField("Types", StringType(), True),
    StructField("Price_level", IntegerType(), True),
    StructField("Rating", IntegerType(), True),
    StructField("Reviews", IntegerType(), True),
    StructField("Neighborhood", StringType(), True),
])

sparkDF=spark.createDataFrame(filecontents, schema)
sparkDF.printSchema()
sparkDF.show()
```

Create table in PostgreSQL

```
createCmd = """CREATE TABLE cafe8 (
                    Name varchar(255),
                    Address varchar(255),
                    Latitude numeric,
                    Longitude numeric,
                    Types varchar(255),
                    Price_level integer,
                    Rating float,
                    Reviews integer,
                    Neighborhood varchar(255),
                    PRIMARY KEY (Address)
                );"""

cur.execute(createCmd)
conn.commit()
```

Our Schema

Cafe	
Name	varchar
Address	varchar
Latitude	numeric
Longitude	numeric
Types	integer
Price Level	integer
Rating	float
Review	integer
Neighborhood	varchar

Designed System Interface

Jupyter Notebook to PostgreSQL

```
[ ]  1 #connect to local host in postgreSQL
     2 import psycopg, os
     3
     4 print('Connecting to the PostgreSQL database...')
     5 conn = psycopg.connect(
     6     host="localhost",
     7     port='5432',
     8     dbname="postgres",
     9     user="postgres",
    10     password="123")

    Connecting to the PostgreSQL database...
```

```
[ ]  1 # create a cursor
     2 cur = conn.cursor()
```

```
     1 #create cafes table in postgreSQL
     2
     3 createCmd = """CREATE TABLE cafe8 (
     4                     Name varchar(255),
     5                     Address varchar(255),
     6                     Latitude numeric,
     7                     Longitude numeric,
     8                     Types varchar(255),
     9                     Price_level integer,
    10                     Rating float,
    11                     Reviews integer,
    12                     Neighborhood varchar(255),
    13                     PRIMARY KEY (Address)
    14                 );"""
    15
    16 #NOTE: we will prob want to change the primary key to (name, places_id) since not every place has an address listed.
    17
    18 cur.execute(createCmd)
    19
    20 #insert data row by row (normal insert df.iterrows)
    21 #save api into file form and import into sql
    22
    23 conn.commit()
```

- The code shown takes the dataframe created from Jupyter Notebook and stores it in PostgreSQL

```
# create a list of all the records
records2 = []
for index, row in df_clean.iterrows():
    name = row['Name']
    address = row['Address']
    latitude = row['Latitude']
    longitude = row['Longitude']
    types = row['Types']
    price_level = row['Price_level']
    rating = row['Rating']
    reviews = row['Reviews']
    neighborhood = row['Neighborhood']

    records2.append((name, address, latitude, longitude, types,
                     price_level, rating, reviews, neighborhood))
```

```
#insert values into the cafe table
insertCmd = "INSERT INTO cafe8 VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s)"


cur.executemany(insertCmd, records2)


conn.commit()
```

Designed System Interface

PostgreSQL to Spark

```
import csv
# execute a SELECT query to retrieve data from a table
cur.execute("SELECT * FROM cafe8")

# fetch all rows as a list of tuples
rows = cur.fetchall()

# close the cursor and connection
cur.close()
conn.close()

# write the rows to a CSV file
with open('cafe_data.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(['Name', 'Address', 'Latitude', 'Longitude', 'Types',
                     'Price_level', 'Rating', 'Reviews', 'Neighborhood'])
    writer.writerows(rows)
```

```
import os
import sys
os.environ['PYSPARK_PYTHON'] = sys.executable
os.environ['PYSPARK_DRIVER_PYTHON'] = sys.executable


from pyspark.sql import SparkSession
from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext

spark = SparkSession.builder.getOrCreate()
sc = spark.sparkContext

print("Using Apache Spark Version", spark.version)
```

- We save the data stored in SQL as a csv before importing it into Spark.
- The code shown displays how we import the data into Spark after creating a schema.

```
from pyspark.sql import SparkSession
import pandas as pd
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, DoubleType
filecontents = pd.read_csv('cafes1.csv')

schema = StructType([
    StructField("Name", StringType(), True),
    StructField("Address", StringType(), True),
    StructField("Latitude", DoubleType(), True),
    StructField("Longitude", DoubleType(), True),
    StructField("Types", StringType(), True),
    StructField("Price_level", IntegerType(), True),
    StructField("Rating", IntegerType(), True),
    StructField("Reviews", IntegerType(), True),
    StructField("Neighborhood", StringType(), True),
])

sparkDF=spark.createDataFrame(filecontents, schema)
sparkDF.printSchema()
sparkDF.show()
```

Designed System Interface

Spark to Flask to HTML

```
from flask import Flask, request, jsonify, redirect, url_for, render_template
import numpy as np
import time

start_time = time.time()

app = Flask("JSON_OUTPUT")

@app.route('/')
def form():
    return render_template('moreedits.html')

@app.route('/submit', methods=['GET','POST'])
def submit():
    if request.method == 'POST':
        types_user = request.form['q']
        price_level_user = request.form['price_level']
        neighborhood_user = request.form['neighborhoods']
        rating_user = request.form['rating']

        def cossim(v1, v2):
            dot_product = np.sum(v1 * v2)
            mag_v1 = np.sqrt(np.sum(np.power(v1, 2)))
            mag_v2 = np.sqrt(np.sum(np.power(v2, 2)))
            return dot_product / (mag_v1 * mag_v2 + 0.1)

        query_txt = types_user
        query_df = sc.parallelize([(1,query_txt)]).toDF(['index','Types'])
        query_tok = regexTokFilter.transform(query_df)
        query_vec = model_type.transform(query_tok)
        query_vec = query_vec.select('wordvectors_type').collect()[0][0]
```

- The code shown takes the inputs from the flask, processes it using code from Spark and connects to the html.

```
        sim_rdd = sc.parallelize((i[0],i[1], i[2],i[3],i[4],i[5],i[6],\
                                  float(cossim(query_vec, i[7])), i[8]) for i in sparkDF_wv_final)
        sim_df = spark.createDataFrame(sim_rdd).\
            withColumnRenamed('_1', 'Name').\
            withColumnRenamed('_2', 'Address').\
            withColumnRenamed('_3', 'Latitude').\
            withColumnRenamed('_4', 'Longitude').\
            withColumnRenamed('_5', 'Types').\
            withColumnRenamed('_6', 'Price_level').\
            withColumnRenamed('_7', 'Rating').\
            withColumnRenamed('_8', 'Similarity').\
            withColumnRenamed('_9', 'Neighborhood').\
            orderBy("Similarity", ascending=False)


    pandas_df = sim_df.toPandas()
    df_filtered = pandas_df[pandas_df['Price_level'] == int(price_level_user)]
    df_filtered1 = df_filtered[df_filtered['Rating'] == int(rating_user)]
    df_filtered2 = df_filtered1[df_filtered1['Neighborhood'] == neighborhood_user]
    html_table = df_filtered2.head(10).to_html(classes='table')
    return render_template('onlytable.html', table=html_table)

@app.route('/output')
def output():
    # render the output HTML page
    return render_template('onlytable.html',table=processed_data)

end_time = time.time()
process_time = end_time - start_time
print("Process time:", process_time)

app.run(host='localhost', port=7033)
```

Queries

- Users can select the price level, rating level, neighbourhood and key in the type of place they intend to go to, such as “cafe” or “restaurant”.
- We filtered our stored data to match each of their queries, except for place types
- For place types, we tokenized the “types” column in Spark and created a word vector to allow for similarity querying. This allowed us to provide the most similar output for each user input.

Tokenize “types” column to a word vector

```
sparkDF_wv['Types','tokens_types','wordvectors_type'].show()
```

```
+--------------------+--------------------+--------------------+
|               Types|        tokens_types|    wordvectors_type|
+--------------------+--------------------+--------------------+
|['cafe', 'point_o...|[cafe, point_of_i...|[0.01981889468152...|
|['book_store', 'c...|[book_store, cafe...|[0.01589654921554...|
|['cafe', 'bakery'...|[cafe, bakery, st...|[0.01682056446692...|
|['cafe', 'store',...|[cafe, store, res...|[0.01883810704263...|
|['cafe', 'store',...|[cafe, store, res...|[0.01883810704263...|
|['cafe', 'store',...|[cafe, store, poi...|[0.01907585905864...|
|['cafe', 'store',...|[cafe, store, res...|[0.01883810704263...|
|['cafe', 'store',...|[cafe, store, poi...|[0.01907585905864...|
|['florist', 'cafe...|[florist, cafe, s...|[0.01589654921554...|
|['cafe', 'restaur...|[cafe, restaurant...|[0.01938498513773...|
|['cafe', 'bakery'...|[cafe, bakery, me...|[0.01475093688350...|
|['cafe', 'restaur...|[cafe, restaurant...|[0.01938498513773...|
|['cafe', 'point_o...|[cafe, point_of_i...|[0.01981889468152...|
|['cafe', 'meal_ta...|[cafe, meal_takea...|[0.01618459800790...|
|['cafe', 'bakery'...|[cafe, bakery, st...|[0.01668243405098...|
|['restaurant', 'c...|[restaurant, cafe...|[0.01682056446692...|
|['cafe', 'store',...|[cafe, store, poi...|[0.01907585905864...|
|['bakery', 'cafe'...|[bakery, cafe, st...|[0.01668243405098...|
|['cafe', 'meal_ta...|[cafe, meal_takea...|[0.01618459800790...|
|['cafe', 'store',...|[cafe, store, foo...|[0.01907585905864...|
+--------------------+--------------------+--------------------+
only showing top 20 rows
```

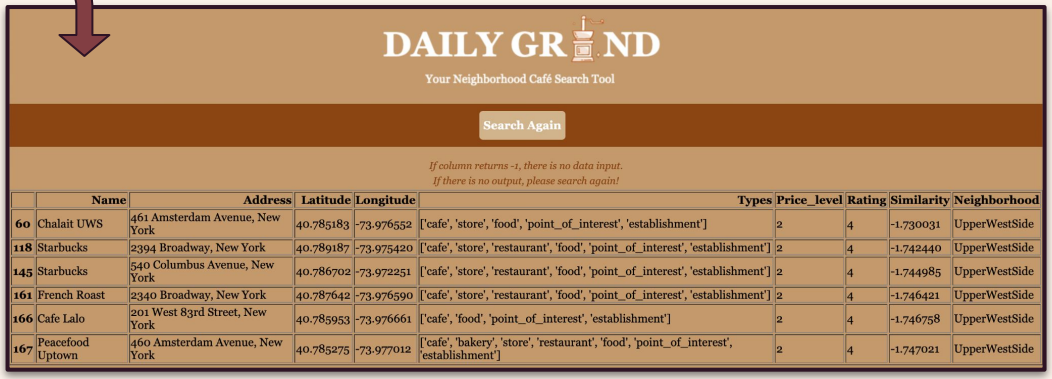
Similarity querying

```
synonyms = model_type.findSynonyms("bakery",5)
synonyms.show()
```

```
+----------------+------------------+
|            word|        similarity|
+----------------+------------------+
|      restaurant|0.9693372845649719|
|             bar|0.9635136723518372|
|point_of_interest|0.9554662108421326|
|           store|0.9458102583885193|
|   establishment|0.9133652448654175|
+----------------+------------------+
```

Interface Output

Our interface output shows a set of attributes that the user can filter for to find the ideal cafe for them in the 10 neighborhoods we have selected. Users can interact with the platform by selecting from the check boxes, drop boxes or directly type in keywords and our system will give an output of the top 10 cafes that are the most similar to the user's preferences.



	Name	Address	Latitude	Longitude	Types	Price_level	Rating	Similarity	Neighborhood
60	Chalait UWS	461 Amsterdam Avenue, New York	40.785183	-73.976552	['cafe', 'store', 'food', 'point_of_interest', 'establishment']	2	4	-1.730031	UpperWestSide
118	Starbucks	2394 Broadway, New York	40.789187	-73.975420	['cafe', 'store', 'restaurant', 'food', 'point_of_interest', 'establishment']	2	4	-1.742440	UpperWestSide
145	Starbucks	540 Columbus Avenue, New York	40.786702	-73.972251	['cafe', 'store', 'restaurant', 'food', 'point_of_interest', 'establishment']	2	4	-1.744985	UpperWestSide
161	French Roast	2340 Broadway, New York	40.787642	-73.976590	['cafe', 'store', 'restaurant', 'food', 'point_of_interest', 'establishment']	2	4	-1.746421	UpperWestSide
166	Cafe Lalo	201 West 83rd Street, New York	40.785953	-73.976661	['cafe', 'food', 'point_of_interest', 'establishment']	2	4	-1.746758	UpperWestSide
167	Peacefood Uptown	460 Amsterdam Avenue, New York	40.785275	-73.977012	['cafe', 'bakery', 'store', 'restaurant', 'food', 'point_of_interest', 'establishment']	2	4	-1.747021	UpperWestSide

Licensing, Data Quality Dimensions, Scalability & Cost Implications

Licensing:

- Google Places API is a paid service that requires a billing account with Google Cloud.
- The free tier API includes up to 100,000 API calls per day, but it limits the default search result to only 20 per query.

Data Quality Dimensions:

- **Accuracy & Timeliness**: Retrieved data from Google Places API, giving us direct access to accurate and regularly updated data, although there may be instances where the data is incomplete.
- **Concise & Consistent:** Google Places API data is generally comprehensive and consistent. We only pulled the necessary data, minimizing data storage and combined it into a uniform dataframe to ease data manipulation.
- **Completeness & Accessibility:** To combat the constraints of the free tier API, we manually selected 10 different locations in Manhattan and queried them individually before concatenating all the data points.

Scalability (requirements):

- In the future, we could use 'next_page_token' to encode more data points into our system.
- Use cloud storage which around $12 for 2TB per month depending on the service provide.
- Include more filters by pulling more variables from the API

Cost:

Google's pay-as-you-go plan has a recurring

$200 monthly credit charge

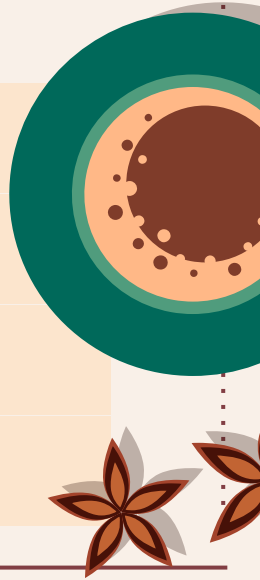
on top of a **$300 trial charge.** To scale, we would need to pay the monthly fee and additional costs based on the number of calls. (API Pricing)

Performance Evaluation Setup and Results

Optimization of data storage and query performance

- Removed duplicates to avoid unnecessary redundancies and maintain data accuracy.
- Selected appropriate data types, using smaller data types where possible to optimize disk space.
- Utilize RDD caching in Spark to improve query performance, as Spark can access the data more quickly in memory than if it had to read it from disk each time.

EVALUATION CRITERIA		METRIC
01	Accuracy in retrieving relevant search results	• Precision and recall
02	Velocity and Scalability	• The number seconds per query, stress/capacity test results
03	Satisfaction of users	• User satisfaction score from surveys/application reviews
04	Coverage and Relevance	• Number of cafes and reviews in the database, the frequency of updates



Storage Costs & Performance Evaluation

Stored Data Size (on SQL): 27KB

- Relatively small due to Google's limitations on API calls
- Easy to scale after purchasing Google's pay-as-you-go API plan
- Plan to update it regularly

To evaluate velocity performance, we used the processing time code chunk to record and minimize the time taken for each process to load in the system

- **API Query Process time: 0.383 seconds (3sf)**
- **Flask Process time: 0.00753 seconds (3sf)**

In the future, we plan to conduct stress/capacity tests to determine scalability, surveys to assess customer satisfaction and improve the accuracy of our machine learning algorithm.

DAILY GROUND

Conclusion

- Using Google Places API, Python, SQL, Spark, and Flask, we built a search engine that allows people to easily find the perfect cafe based on their specific needs and preferences.

Recommendations

- Increase the size of database
  - Get a paid plan of Google Places API
- Enable real-time data processing
  - Stream real-time data in Spark to provide up-to-date results to users quickly and accurately, making for a more seamless and efficient user experience.
- Provide more options for users to narrow their preference
  - Add more attributes for the users to select from
- Enable group search
  - Get multiple users to select their preferences and give back results that fit majority
- Allow users to prioritize their features of their requirement
  - Incorporate a ranking system for users to set preferences that are the most important or non-negotiable

Team Member Roles and Contributions

Roles	Name	Contributions
Team Leader	Elina Shirolkar	API, Flask & Powerpoint Content
Team Member	Powen Hsu	Debugging Code, Powerpoint Content
Team Member	Meenu Selvakesari	Spark & Flask, Powerpoint Content
Team Member	Joy Tay	Data Wrangling, PostgreSQL, Flask & Powerpoint Content
Team Member	Joy Shyu	API, PostgreSQL, Flask & Powerpoint Content

Thank You!

Have a great summer :)

References

- Brown, S. (2023, February 15). *Best Cloud Storage Software Options for 2023*. CNET. Retrieved from https://www.cnet.com/tech/services-and-software/best-cloud-storage-software-options/
- Google. (2023, March 29). Google Maps Platform. Retrieved from https://developers.google.com/maps/documentation/places/web-service/usage-and-billing
- MongoDB. (n.d.). Retrieved from https://www.mongodb.com/pricing