Data Analytics

Taxi Pickup Location Recommender System

Dooinn KIM

Feb, 2024

Table of content

Introduction

The urban transportation landscape is experiencing a significant transformation, driven by advancements in technology and data analytics. In this context, taxi services are seeking innovative solutions to enhance operational efficiency and customer satisfaction. The Taxi Pickup Location Recommender System project is a pioneering initiative aimed at addressing one of the most pressing challenges faced by taxi drivers in Chicago: identifying the most probable locations for passenger pickups with minimal effort and time. This system leverages extensive data analysis and machine learning algorithms to recommend optimal pickup points, thereby streamlining the taxi service process and improving the overall experience for both drivers and passengers.

Our primary goal is to harness data from various sources to empower taxi drivers with actionable insights that allow them to efficiently locate high-demand areas for passenger pickups. This, in turn, is expected to reduce idle time, lower fuel consumption, and increase the profitability of their operations. The recommender system utilizes the Nearest Neighbors algorithm with a "Ball Tree" method to analyze current location and time data, providing drivers with a ranked list of the top five pickup locations based on proximity and historical trends.

The development plan for this project was strategically outlined to ensure a thorough and effective execution, encompassing the following key phases:

- **Data Collection**: Aggregating relevant datasets from the Google Cloud Platform, Wikipedia, and the Chicago City Data Portal.
- **Data Cleaning & Exploratory Data Analysis (EDA):** Standardizing and cleaning the collected data to ensure its quality and usability for analysis. Conducting EDA for a deep dive into the dataset using Tableau to uncover patterns, trends, and insights that could inform the system's recommendations.
- **Database Selection & Entities Relationship Diagram(ERD):** Evaluating and selecting MySQL as the preferred database management system for storing and managing the cleaned data. Created ERD to understand the relationships between the data tables.
- **API Development:** Implementing a Flask API to provide seamless access to the processed data, facilitating its integration into the recommender system.
- **Machine Learning Modeling**: Developing the recommendation algorithm using the Nearest Neighbors method, tailored to identify optimal pickup locations based on real-time and historical data.

Data and Data sources

This section outlines the key data sources utilized in the project, encompassing a wide range of information from geographical data to specific details about taxi companies and vehicles.

- **Big Query Google Cloud Platform** (*Available at: https://console.cloud.google.com/marketplace/product/city-of-chicago-public-data/chicago-taxi-trips?project=personal-projects-382818*): Chicago Taxi Trips - This dataset serves as the backbone of the project, providing detailed records of taxi trips in Chicago, including pickup and dropoff locations, fare amounts, and trip durations.
- **Wikipedia - Chicago Community Areas** (*Available at : https://en.wikipedia.org/wiki/Community_areas_in_Chicago*) - This online encyclopedia offers valuable information on Chicago's community areas, facilitating an understanding of the city's geographical and demographic layout, which is crucial for pinpointing high-demand areas for taxi pickups.
- **Nominatim API - OpenStreetMap Data** (*Available at: https://nominatim.org/release-docs/develop/api/Overview/*) - This API provides access to global geographical data, allowing for the retrieval of precise geo-coordinates and addresses for the pickup and dropoff locations identified in the taxi trip records.
- **Flat files from Chicago Data Portal** (*Available at: https://data.cityofchicago.org/browse?q=taxi&sortBy=relevance*) - Sourced from the city's official data portal, this dataset includes detailed information on taxi vehicles and companies, such as vehicle models, fuel types, and company contact information, enriching the analysis with operational insights.

Data Collection

- **Big Query Google Cloud Platform**: Chicago Taxi Trips

From the BigQuery Google Cloud Platform, we specifically queried data timestamped in 2015 for the top five most demanded taxi companies in Chicago: Yellow Cab, American United, Checker Taxi, Blue Diamond, and 5 Star Taxi. This focused dataset was chosen due to the considerable volume and size of the dataset. Furthermore, the year 2015 was selected because it recorded the highest dataset volume compared to other years, suggesting a more reliable data set.

chicago_taxi (*4,972,163 rows x 24 columns*)		
Columns	**Description**	**Data Type**
unique_key	Unique identifier for each record	object
taxi_id	Unique identifier for each taxi	object
trip_year	The year of trips made (e.g 2015)	Int64
trip_start_timestamp	Timestamp marked of trip start	datetime64[us, UTC]
trip_start_date	Date of a trip start (e.g. 2015-04-04)	dbdate
trip_start_time	Hours of a trip made (e.g. 15:00:00)	object
trip_end_timestamp	Timestamp marked of trip end	datetime64[us, UTC]
trip_end_date	Date of a trip end (e.g. 2015-04-04)	dbdate
trip_end_time	Hours of a trip made (e.g. 15:00:00)	object
trip_seconds	Total duration of a trip in seconds	Int64
trip_miles	Total distance of a trip in miles	float64
pickup_community_area	Pickup location of Chicago community area number	Int64
dropoff_community_area	Dropoff location of Chicago community area number	Int64
fare	Taxi fare by distance	Int64
tips	Tips given by passengers	Int64
tolls	Tolls for highways	Int64
extra	Extra charges by taxi drivers	Int64
trip_total	Total amount of trip (fare + tips + tolls + extra)	Int64
payment_type	Passenger's payment method - Card/Cash	object
company	Taxi companies in Chicago	object
pickup_latitude	-	float64
pickup_longitude	-	float64
dropoff_lattitude	-	float64

dropoff_longitude	-	float64

- **Wikipedia - Chicago Community Areas**

In the Chicago Taxi Trips dataset from the big data system, only the Chicago community area numbers are provided, with no specific names corresponding to these numbers. Therefore, we have collected additional information, including the names of the community areas and their populations, which may be useful for future market research purposes. For web scraping, we utilized BeautifulSoup, a library that simplifies the process of extracting information from web pages.

community (*77 rows x 3 columns*)		
Columns	**Description**	**Data Type**
community_number	Chicago Community Area number	Int64
communit_name	Name of Chicago Community	object
Population	Population of community	Int64

- **Nominatim API - OpenStreetMap Data**

We have additionally gathered specific geographic information through the Nominatim API, which utilizes OpenStreetMap data to locate places on Earth by name and address (geocoding). Given the absence of address information in the Chicago taxi trips dataset, the Nominatim API provides address details based on geographic coordinates—latitude and longitude. We compiled the latitude and longitude data for pickup and dropoff locations from the Chicago Taxi Trips dataset to identify unique geolocations. These were then integrated with address information obtained from OpenStreetMap via the Nominatim API.

location_info (*421 rows x 3 columns*)		
Columns	**Description**	**Data Type**
location_coordinates	Location coordinates	object
address	Address of location coordinates	object
type	Type of building	object

- **Flat files from Chicago Data Portal**

We have also collected information on Chicago taxi vehicles and taxi companies in flat file formats (CSV for taxi vehicle info and PDF for taxi company info). Due to the absence of comprehensive information in the dataset and the impossibility of finding detailed information precisely corresponding to the taxi_id in the Chicago Taxi Trips dataset, we have randomly assigned taxi_ids as

identifiers to design a relational database. This approach was taken solely to fulfill the criteria of the RNCP (Répertoire National des Certifications Professionnelles).

Furthermore, regarding the taxi company dataset, it was observed that the available dataset does not encompass the complete list of companies extracted from the Chicago Taxi Trips dataset from the big data system. Despite this, the information from these two sources provides valuable insights for a better understanding of Chicago taxis and their operating companies.

taxi *(386 rows x 6 columns)*		
Column	**Description**	**Data type**
taxi_id	Unique identifier for each taxi	object
Public Vehicle Number	Taxi Licence identifier	int64
Vehicle Make	Name of vehicle brand/company (e.g. Ford, Kia)	object
Vehicle Model Year	Model made year of the vehicle	float64
Vehicle Color	Color of vehicle	object
Vehicle Fuel Source	Fuel source type (e.g. Hybrid, Flex Fuel)	object

company *(19 rows x 9 columns)*		
Column	**Description**	**Data type**
company_id	Unique company identifier of taxi company	int64
company	Name of taxi company	object
taxi_exterior_color	-	object
business_phone	-	object
dispatch_phone	-	object
address	Company Address	object
city_state	State where company located	object
zip	Zipcode of company location	into64
email	-	object

Data Cleaning & Exploratory Data Analysis (EDA)

Data Cleaning

The initial dataset obtained, particularly the Chicago Taxi Trips data from the Big Query Google Cloud Platform, was already in a relatively clean state, which streamlined the process of preparing it for database storage. However, a meticulous approach was adopted to ensure the data was optimally formatted for efficient storage and retrieval. This involved several key steps:

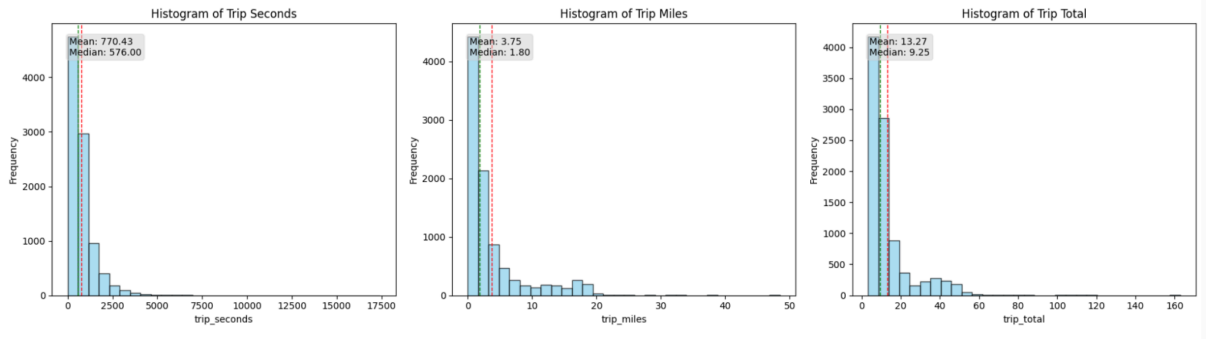
- **Data Type Assignment**: Correct data types were assigned to each column to ensure data integrity and optimize storage efficiency. This step is crucial as it directly impacts the database's performance and the accuracy of data retrieval processes.
- **Standardization of Headers**: Column names were standardized to maintain consistency across the database. This not only aids in easier querying but also prevents potential errors during data analysis.
- **Removal of Unnecessary Columns**: Columns that contained a significant number of null values or were deemed irrelevant to the project's objectives were carefully identified and removed. For instance, the 'tolls' column in the Chicago Taxi Trips dataset, which had a high incidence of null values, was excluded. This decision was made to streamline the dataset and focus on information crucial for the location recommendation system.
- **Handling Null Values:** For columns critical to the recommendation system, such as geographic coordinates (latitude and longitude), records with null values were dropped. This action was taken under the premise that the loss of data (ranging from 3% to 5%) would not significantly impact the overall dataset's utility for the project. The rationale was to maintain a high level of data accuracy, especially concerning location information, which is vital for the recommender system's effectiveness.
- **Assignment of Unique Identifiers:** In the case of the Chicago taxi trips and company datasets, unique numeric identifiers (company IDs) were randomly assigned to each taxi company. This step was crucial for several reasons: it facilitated efficient data storage by reducing the space required for text strings, enhanced the speed of database operations, and simplified the process of data analysis by providing a straightforward method of referencing each company.

Exploratory Data Analysis (EDA)

For our Exploratory Data Analysis (EDA) and machine learning model, we utilized a subset of the 'trips' dataset, specifically 943,890 rows, which constitutes approximately 18% of the entire dataset. This selection was filtered through API endpoints using **GET /trips/4,** as detailed in the section 'Exposing Data via API'. Our focus was on examining taxi trips associated with the American United taxi company.

In our analysis, we employed Python and Tableau for data exploration. The EDA process included several key steps:

1. **Data Inspection**: We used *trips.info()* to check the number of rows and columns and to understand the datatype of each column.
2. **Statistical Summary:** We applied *trips.describe()* to specifically examine the summary statistics for trip_seconds, trip_miles, and trip_total, providing insights into the central tendency and spread of these variables.
3. **Categorical Analysis**: Through *trips[categorical columns].value_counts()*, we investigated the unique values and frequency of each categorical feature, including payment_type, pickup_community_area, dropoff_community_area, and address.
4. **Visualizations**: This comprehensive approach to data analysis, combining Python's data manipulation capabilities with Tableau's visualization strengths, enabled us to derive meaningful insights into the taxi service operations, specifically focusing on optimizing taxi drivers' efficiency and earnings.



Matplotlib Histograms*: We created histograms for trip_seconds, trip_miles, and trip_total to visualize their distributions. This revealed the median values for the duration, distance, and earnings per trip for American United Taxi were 576 seconds (9.6 minutes), 1.80 miles, and $9.25, respectively.*

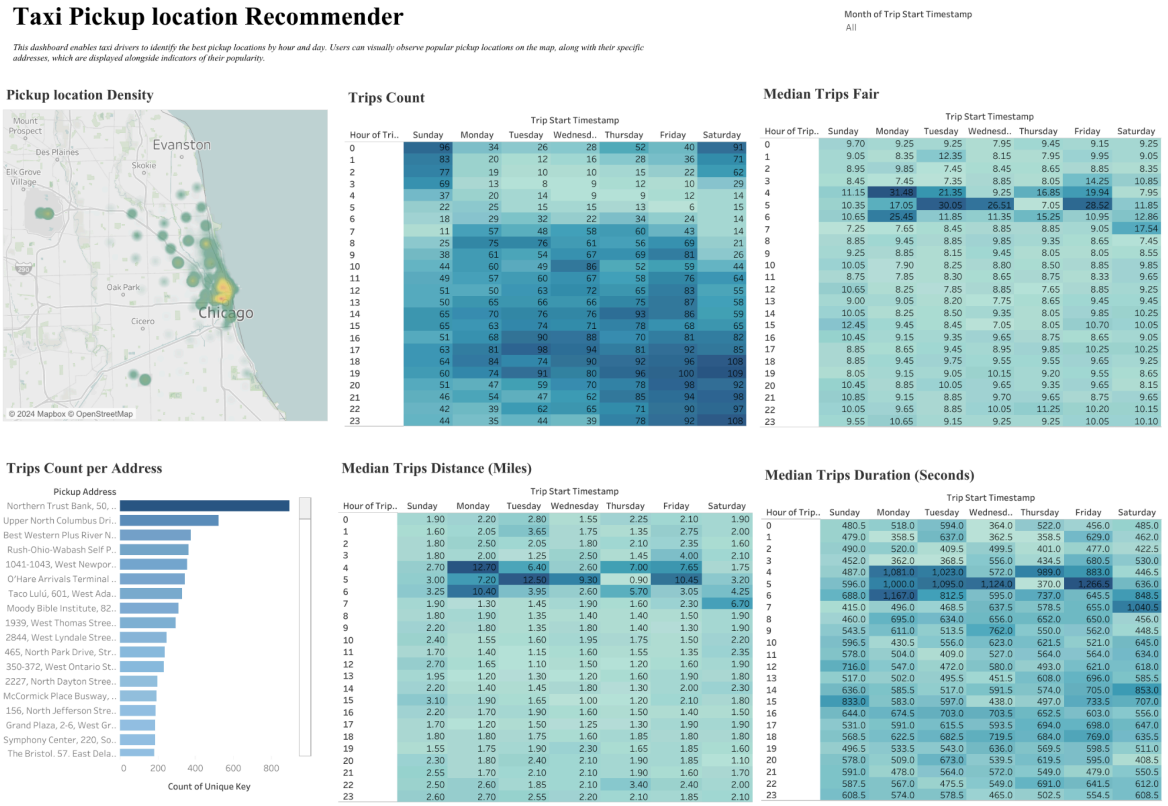


Tableau Dashboards: *We explored the dataset further with Tableau visualizations to uncover interesting insights. A dashboard was developed to allow users to explore various visualizations, including the density of taxi pickup locations on a map, and breakdowns by address, hour, day, and month. This analysis helped identify potential high-revenue pickup locations across different times and days. Notably, we found that from Thursday to Saturday evenings, the demand for taxis increased, suggesting that drivers are more likely to pick up passengers easily during these periods compared to other times or days.*

Database Type Selection

Comparison: Relational Database vs Non-Relational Database

Relational Databases operate on a structured query language (SQL) for defining and manipulating data, which is stored in a tabular form. They are characterized by their strict schema, which requires that all data follow the same structure. This feature makes relational databases highly organized and suitable for complex queries involving multiple tables.

Relational Databases: Pros, Cons & Products	
Pros	- Strong ACID (Atomicity, Consistency, Isolation, Durability) compliance ensures reliability in transactions. - Ideal for complex queries and relationships between different data types. - Scales well vertically.
Cons	- Limited flexibility due to strict schema requirements. - Can become expensive and challenging to scale horizontally.
Products	- MySQL, PostgreSQL, Oracle Database.

Non-Relational Databases, or **NoSQL database**s, are more flexible in terms of data storage. They can store structured, semi-structured, or unstructured data and do not require a fixed schema. This flexibility allows for easier horizontal scaling and adaptation to different data types and structures.

Non-Relational Databases: Pros, Cons & Products	
Pros	- Schema flexibility allows for the storage of varied data types. - Designed to scale out by distributing data across multiple machines. - Can handle large volumes of data at high speeds.

Cons	- May lack the robust transactional integrity of relational databases. - Complex queries can be more challenging to implement and optimize.
Products	- MongoDB, Cassandra, CouchDB.

Database Creation

For the Taxi Pickup Location Recommender System, MySQL was selected as the database platform. A data schema named ‘chicago_taxi’ was created, along with tables named ‘taxi’, ‘community’, ‘company’, ‘location’, and ‘trips’ within MySQL. Data values for each of these tables were populated using Python with the aid of the ‘mysql.connector’ library.

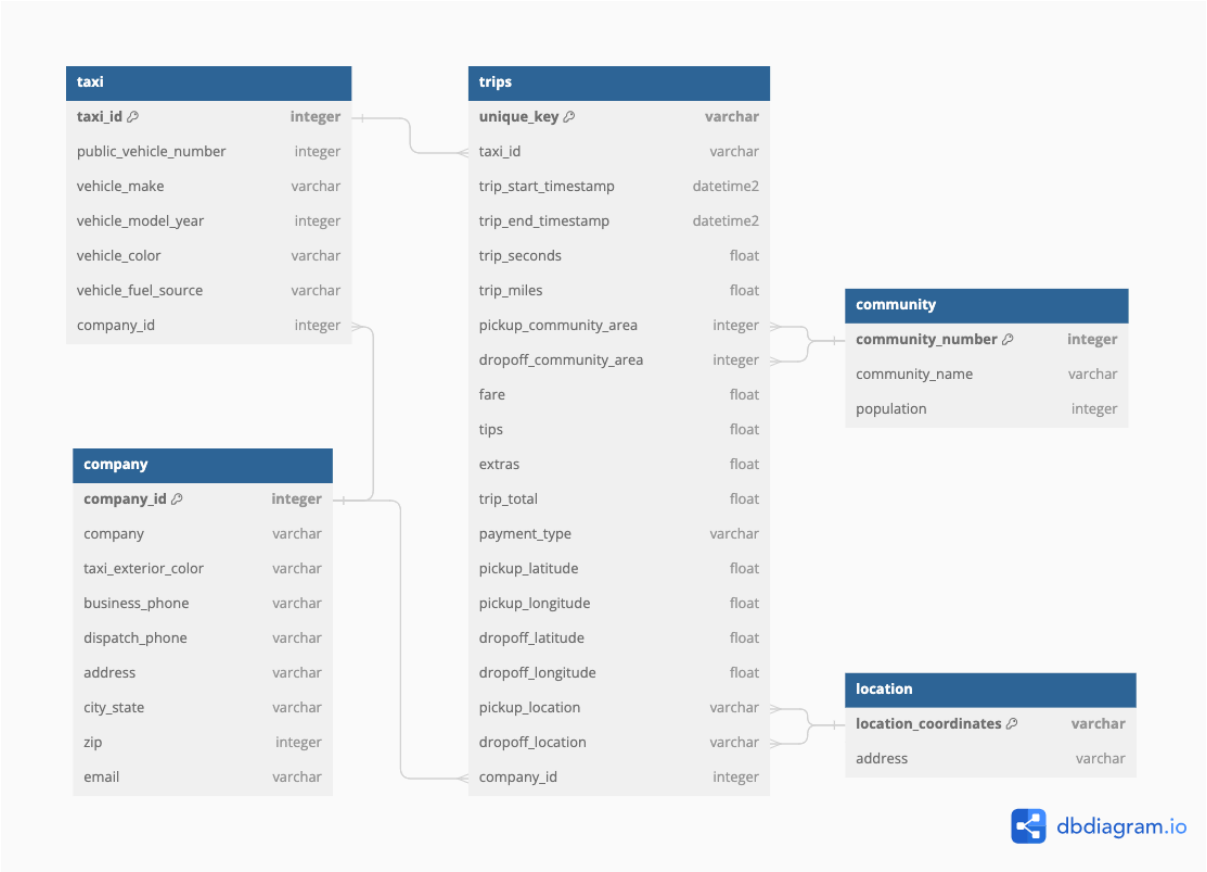
```sql
USE chicago_taxi;

CREATE TABLE IF NOT EXISTS taxi (
    taxi_id VARCHAR(255) PRIMARY KEY,
    public_vehicle_number INT,
    vehicle_make VARCHAR(255),
    vehicle_model_year INT,
    vehicle_color VARCHAR(255),
    vehicle_fuel_source VARCHAR(255),
    company_id INT
);

CREATE TABLE IF NOT EXISTS community (
    community_number INT PRIMARY KEY,
    community_name VARCHAR(255),
    population INT
);

CREATE TABLE IF NOT EXISTS company (
    company_id INT PRIMARY KEY,
    company VARCHAR(255),
    taxi_exterior_color VARCHAR(255),
    business_phone VARCHAR(255),
    dispatch_phone VARCHAR(255),
    address VARCHAR(255),
    city_state VARCHAR(255),
    zip INT,
    email VARCHAR(255)
);

CREATE TABLE IF NOT EXISTS location (
    location_coordinates VARCHAR(255) PRIMARY KEY,
    address VARCHAR(255)
);
```

```sql
CREATE TABLE IF NOT EXISTS trips (
    unique_key VARCHAR(255) PRIMARY KEY,
    taxi_id VARCHAR(255),
    trip_start_timestamp DATETIME,
    trip_end_timestamp DATETIME,
    trip_seconds FLOAT,
    trip_miles FLOAT,
    pickup_community_area INT,
    dropoff_community_area INT,
    fare DECIMAL(10, 2),
    tips DECIMAL(10, 2),
    extras DECIMAL(10, 2),
    trip_total DECIMAL(10, 2),
    payment_type VARCHAR(255),
    pickup_latitude DECIMAL(9, 6),
    pickup_longitude DECIMAL(9, 6),
    dropoff_latitude DECIMAL(9, 6),
    dropoff_longitude DECIMAL(9, 6),
    pickup_location VARCHAR(255),
    dropoff_location VARCHAR(255),
    company_id INT
);
```

Entity Relationship Diagram (ERD) & SQL Queries

Entity Relationship Diagram(ERD)

The following image represents the Entities Relationship Diagram (ERD) of the chicago_taxi database in MySQL. Each table has primary keys, and some of them are linked as foreign keys.



SQL Queries

Example SQL queries for insight extraction.:

```sql
# Daily Average Turnaround per taxi?
with RankedTrips as (
    select taxi_id,trip_start_timestamp,trip_end_timestamp,
        lead(trip_start_timestamp) over (partition by taxi_id order by trip_start_timestamp) as next_pickup_time,
        date(trip_start_timestamp) as pickup_date
    from american_united_trips
),
TurnaroundTimes AS (
    select taxi_id,pickup_date,
        timestampdiff(minute, trip_end_timestamp, next_pickup_time) as turnaround_time
    from RankedTrips
    where next_pickup_time is not null
),
TaxiAVGTR as (
select taxi_id,pickup_date,
    avg(turnaround_time) as average_turnaround_time
from TurnaroundTimes
group by taxi_id, pickup_date
order by taxi_id, pickup_date
)
select taxi_id, round(avg(average_turnaround_time),2) as daily_avg_turnaround from TaxiAVGTR
group by taxi_id;
```

```sql
# Top 10 pickup location?
select location.address, count(*) as total from american_united_trips
left join location
on american_united_trips.pickup_location = location.location_coordinates
group by address
order by total desc
limit 10;
```

```sql
# Payment method - Credit Card vs Cash
select
    payment_type,
    round(avg(trip_total),2) as avg_fare,
    round(avg(trip_miles),2), count(*) as count
from american_united_trips
group by payment_type;
```

```sql
# Daily earnings per taxi?

with EaringDate as (
SELECT taxi_id, trip_total, date(trip_start_timestamp) AS pickup_date
FROM american_united_trips
),
DailyTotal as (
select taxi_id,pickup_date, sum(trip_total) as daily_total from EaringDate
group by taxi_id, pickup_date
)
select taxi_id, avg(daily_total)
from DailyTotal
group by taxi_id;
```

```sql
  # which hours highest demand by location?
with HourlyDemand as (
  select pickup_location, extract(hour from trip_start_timestamp) as trip_hour,
    count(*) as total
  from american_united_trips
  group by pickup_location, extract(hour from trip_start_timestamp)
),
RankedDemand AS (
  select pickup_location, trip_hour, total,
    rank() over (partition by pickup_location order by total desc) as rank_demand
  from HourlyDemand
),
HighDemand as (
select pickup_location, trip_hour as highest_demand_hour, total
from RankedDemand
where rank_demand = 1
) select address, highest_demand_hour, total from HighDemand
left join location
on HighDemand.pickup_location = location.location_coordinates;
```

General Data Protection Regulation (GDPR)

The General Data Protection Regulation (GDPR) is a legal framework established by the European Union (EU) to protect the personal data and privacy of its citizens. It imposes strict requirements on how personal data is collected, processed, stored, and shared, providing individuals with greater control over their personal information. Key principles include data minimization, consent, transparency, and the right to access and rectify information.

Types of Data

Under GDPR, data is categorized into two main types: personal data and non-personal data.

- Personal Data: Information that can directly or indirectly identify a person. This includes:
    - Names
    - Email addresses
    - Location data
    - Identification numbers
    - Online identifiers
- Non-Personal Data: Information that cannot identify a person and relates to objects, places, or anonymized datasets. Examples include:
    - Aggregated data that cannot be de-anonymized
    - Vehicle registration numbers (when not linked to an owner)
    - Locations (when not linked to specific individuals)
    - Company names

API Development & Exposing Data via API

API Development with Flask API

To facilitate access to a dataset stored in a MySQL database, an API was developed using Flask, a popular micro web framework written in Python. This API is designed to serve as a bridge between the database and users wishing to retrieve data, enhancing data accessibility and interaction for various applications and research purposes. Flask's lightweight and extensible nature makes it an excellent choice for creating RESTful APIs, which follow a stateless, client-server architecture where each request from any client contains all the information needed to service the request.

The API provides access to the stored data through five distinct endpoints, each designed to serve a specific segment of the dataset. These endpoints are as follows:

Type	**Description**	**Endpoints**
Community	This endpoint allows users to retrieve information related to community areas. It's useful for applications and analyses that focus on community-based statistics or need to segment data by geographical areas within the city.	**GET** /community
Taxi	Through this endpoint, users can access comprehensive data about taxis, including details about taxi licenses, inspection records, or any other relevant taxi-specific information stored in the database.	**GET** /taxi
Company	This endpoint is dedicated to providing information about the taxi companies operating within the city. Users can find data on company profiles, including their service records, fleet sizes, and operational areas.	**GET** /company
Location	Users can query geographical location data, such as pickup and dropoff points, through this endpoint. This information is crucial for mapping services, spatial analysis, and optimizing route planning for efficiency and service quality improvement.	**GET** /location
Trips	Perhaps the most versatile endpoint, it allows users to filter trip data by a specific taxi company and date range. This endpoint supports additional query parameters for pagination, such as page and page_size, to efficiently manage large datasets and improve the user experience by providing tailored data slices. This feature is particularly beneficial for detailed analyses of trip patterns, revenue estimates, and operational efficiencies over time. (e.g. GET /trips/1/2023-01-01/2023-01-31?page=1&page_size=100)	**GET** /trips/<company_id>/<start_date>/<end_date>

Example of an API result for GET /trips/4/:


```
{
  "current_page": 1,
  "data": [
    {
      "company_id": 4,
      "dropoff_community_area": 32,
      "dropoff_latitude": "41.884987",
      "dropoff_location": "41.884987192, -87.620992913",
      "dropoff_longitude": "-87.620993",
      "extras": "1.50",
      "fare": "5.45",
      "payment_type": "Credit Card",
      "pickup_community_area": 32,
      "pickup_latitude": "41.880994",
      "pickup_location": "41.880994471, -87.632746489",
      "pickup_longitude": "-87.632746",
      "taxi_id":
"b19847c6c45cc89f385408086d905e8de9069bd2ac17152c89b935c7f6bf40644a0ea3608b3e3929cefc79c46487e8439ec47c0bf85d8f98400dd44bc23323aa",
      "tips": "1.00",
      "trip_end_timestamp": "Thu, 19 Mar 2015 15:00:00 GMT",
      "trip_miles": 1.0,
      "trip_seconds": 287.0,
      "trip_start_timestamp": "Thu, 19 Mar 2015 15:00:00 GMT",
      "trip_total": "7.95",
      "unique_key": "000008622a94904e7ba554663f7e286fa38c362e"
    },
    {
      "company_id": 4,
      "dropoff_community_area": 32,
      "dropoff_latitude": "41.877406",
      "dropoff_location": "41.877406123, -87.621971652",
      "dropoff_longitude": "-87.621972",
      "extras": "0.00",
      "fare": "6.65",
      "payment_type": "Credit Card",
      "pickup_community_area": 8,
      "pickup_latitude": "41.895033",
      "pickup_location": "41.89503345, -87.619710672",
      "pickup_longitude": "-87.619711",
      "taxi_id":
"17d9b7691aef7770e5a9e499426b26049100251dbb41dd8717b42954de963b9c7b3abc00cf2928233c830650bd9ad3b4660b4ccb645b0075e054c6ed755bbe91",
      "tips": "1.00",
      "trip_end_timestamp": "Mon, 15 Jun 2015 09:00:00 GMT",
      "trip_miles": 1.2,
      "trip_seconds": 471.0,
      "trip_start_timestamp": "Mon, 15 Jun 2015 09:00:00 GMT",
      "trip_total": "7.65",
      "unique_key": "0000282684214e9e541533eae7b74d315dc4cf04"
    },
```

Exposing Data via API on Jupyter Notebook

The data for the API is presented in a Jupyter notebook as depicted in the accompanying image, setting the stage for subsequent steps in data preprocessing and machine learning modeling. Given the extensive volume of the dataset for trips, we have selectively exposed only the data corresponding to trips where the company id is 4 (representing American United). This subset still comprises a significant size of 943,890 rows across 20 columns.

Python code to fetch data via API call:

```python
def fetch_communiyt_taxi_comapny_location(url):
    response = requests.get(url)
    data = response.json()
    return pd.json_normalize(data)

# Fetch urls for community, taxi, company, location
urls = {
    "community": "http://127.0.0.1:5000/community",
    "taxi": "http://127.0.0.1:5000/taxi",
    "company": "http://127.0.0.1:5000/company",
    "location": "http://127.0.0.1:5000/location"
}


#Fetch trips
def fetch_all_pages(base_url, start_page, end_page):
    all_data_frames = []
    for page in range(start_page, end_page + 1):
        try:
            response = requests.get(f"{base_url}/?page={page}")
            response.raise_for_status()  # Raises an error for non-200 responses
            page_data = response.json()
            page_frame = pd.json_normalize(page_data['data'])
            all_data_frames.append(page_frame)
        except requests.RequestException as e:
            print(f"An error occurred on page {page}: {e}")
            break
    return pd.concat(all_data_frames, ignore_index=True)

# Usage
base_url = "http://127.0.0.1:5000/trips/4"
start_page = 1
end_page = 95
```

Machine Learning: Location Recommender

The Taxi Pickup Location Recommender System is designed to enhance the efficiency of taxi drivers by guiding them to optimal pickup locations. This system minimizes waiting times and reduces fuel consumption by leveraging machine learning to predict where passengers are likely to need rides. Here's a structured overview of how the system was developed and operates:

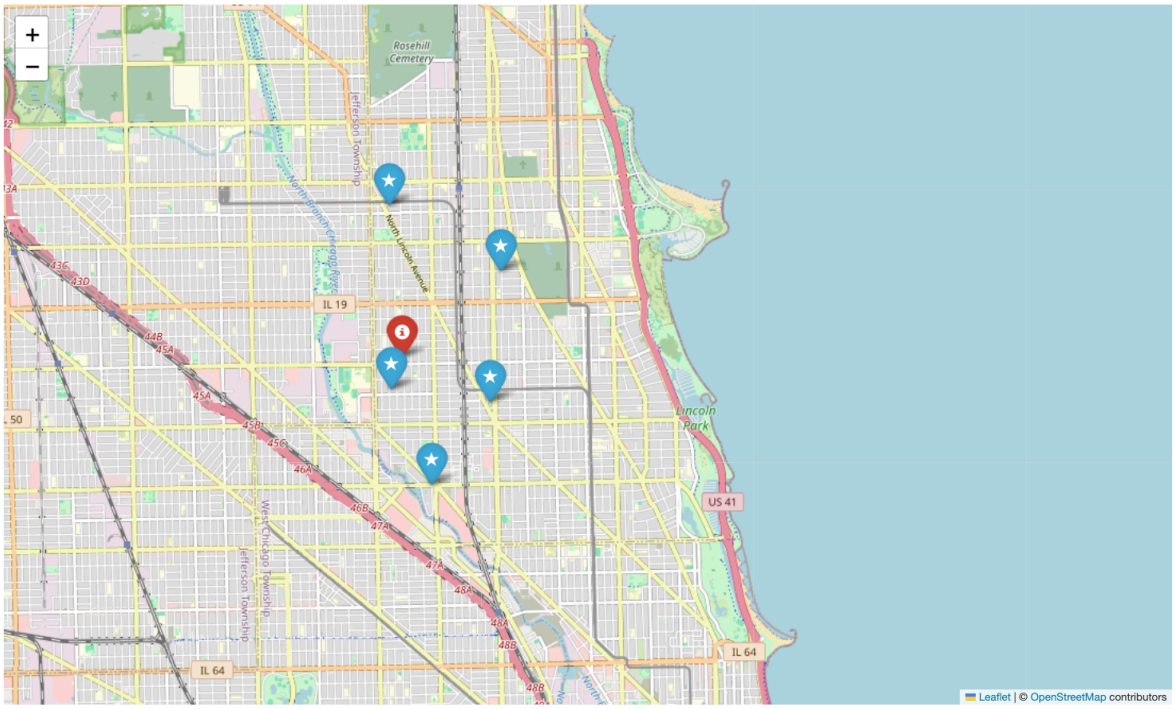
1. **Optimal Cluster Determination Using Silhouette Scores:**
   To improve location recommendations, the system employs silhouette scores to identify the best number of clusters within the geographical data. The silhouette score is a measure of how similar an object is to its own cluster compared to other clusters. The higher the silhouette score, the more appropriately the object is matched to its own cluster and not neighboring clusters. For this system, analysis determined that 80 is the optimal number of clusters. This granularity allows for precise location predictions and efficient resource allocation.
2. **Clustering with K-means:**
   With the optimal cluster number established, the system uses the K-means clustering algorithm, setting the number of clusters to 80. K-means is a popular method for partitioning data into groups based on feature similarity—in this case, pickup latitude and longitude. By clustering pickup locations, the algorithm helps identify hotspots where passengers frequently request taxis.
3. **Nearest Neighbors for Cluster Center Prediction:**
   After defining the clusters, the system calculates their centers. It then employs a Nearest Neighbors model to predict the nearest cluster center to a given location. The Nearest Neighbors algorithm is particularly useful in geolocation contexts as it can efficiently identify the closest pickup hotspots based on the taxi's current position.
4. **Integration with Nominatim API and Rule-Based Recommendations:**
   The system integrates with the Nominatim API to convert recommended geographical coordinates into human-readable addresses. Furthermore, it incorporates a rule-based function that ranks the recommended locations based on historical popularity data, taking into account the time of day and the day of the week. This ensures that the recommendations are not only geographically accurate but also contextually relevant.
5. **Visualization with Folium Maps:**
   For user-friendly visualization, the system utilizes Folium maps to display recommended pickup locations. When a user inputs their current latitude and longitude, along with the current hour and day, the system generates a map marking the top 5 recommended locations. These recommendations include addresses and the distance from the user's current location, enabling drivers to make informed decisions quickly.

Example of output results showing the top 5 recommended locations based on the input data of location, time, and day:

```
# Example usage of the function
example_location = (41.947792, -87.683835)
example_time = 18  # 3 PM
example_date = 4  # Friday


recommended_pickups = recommend_pickup_locations(example_location, example_time, example_date, nearest_neighbors_model, cluster_centers, trips
pickup_map = create_map(example_location, recommended_pickups)

display(recommended_pickups)
display(pickup_map)
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature
names, but NearestNeighbors was fitted with feature names
  warnings.warn(
```

	cluster	count	pickup_latitude	pickup_longitude	address	distance_km
0	34	96	41.943890	-87.685574	3423, North Oakley Avenue, Bricktown, North Ce...	0.456720
1	3	57	41.966028	-87.685844	Jefferson Pumping Station, West Eastwood Avenu...	2.032332
2	60	44	41.942424	-87.669446	Lakeview YMCA, 3333, North Marshfield Avenue, ...	1.333895
3	50	35	41.958116	-87.667727	4200, North Greenview Avenue, Lake View, Chica...	1.760343
4	19	5	41.932406	-87.678993	2826-2838, North Clybourn Avenue, Bricktown, N...	1.755473



Conclusion

In this paper, we examined the development of a Taxi Recommender System, covering the spectrum from data collection and storage, through API development, to the construction of the machine learning model. The development process presented several significant challenges:

- **High Volume of Data:** Managing the large volume of data posed a significant challenge, particularly in establishing criteria for sampling. We chose to focus on data from 2015 and the top 5 highest demand taxi companies, as 2015 contained the most substantial amount of data and was deemed the most representative sample for our analysis. Despite this, the dataset still encompassed over 4 million rows, presenting a considerable task in extraction and storage within a MySQL database.
- **Limited Availability of Relevant Data:** Although we had access to a comprehensive taxi trip dataset from BigQuery, sourcing specific information about taxis and their companies proved difficult. Regulations around personal information, such as driver's licenses and contact details, restricted the use of potentially relevant data. Nevertheless, we managed to gather useful information on taxi vehicles (model and year) and some company details (color, addresses, and phone numbers).
- **Machine Learning Performance Evaluation**: A key limitation of this project was the inability to test our algorithms in real-world scenarios to ascertain their performance. While we employed the Nearest Neighbors algorithm and acknowledged the existence of other spatial distance-based algorithms like DBSCAN, we lacked the means for a comparative performance analysis.
- **Training Dataset Geographical Limitation**: Our reliance on Chicago taxi trip data for training the model inherently limited the scope of our recommendations to Chicago, restricting the applicability of our system to this geographic area.

Despite these challenges, which are not uncommon in data science projects, the methodology and insights gleaned from this work are valuable. This project offers a glimpse into the development of a location-based recommender system and underscores the potential of such systems. To enhance our model, future work will involve:

- Expanding the dataset beyond Chicago taxi trips.
- Including data from years beyond 2015 and from a broader selection of taxi companies.
- Experimenting with and evaluating the recommender system in real-world scenarios to accurately assess its effectiveness and refine it through additional machine learning algorithms.

This approach not only aims to improve the current model but also to broaden its applicability and accuracy in making location recommendations.

Related Links

- **Taxi Pickup Location Recommender System Github Repo:**
  https://github.com/dooinn/taxi_location_recommender
- **Trello for Project Management:**
  https://trello.com/b/oCHDG1CD/rncp-taxi-location-recommender
- **Tableau Taxi Pickup Location Dashboard:**
  https://public.tableau.com/app/profile/dooinn/viz/TaxiRecommendor/Dashboard3
- **General Data Protection Regulation (GDPR)**:
  https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A02016R0679-20160504