Data Exploration

Exploratory Data Analysis on Omdena_Repzone_v1.xlsx data

This document outlines an exploratory data analysis (EDA) performed on a dataset related to sales representatives and their customers in Turkey.

The goal is to gain insights into representative workload, customer distribution, and potential optimization opportunities within the sales network.

The analysis examines two datasets: one detailing representative information like working hours and customer capacity, and another containing customer details such as visit frequency and geographic location.

The EDA involves data cleaning, feature engineering (creating new columns from existing ones), and visualisation using libraries like Pandas, Matplotlib, Seaborn, and Folium.

1. Data Overview

The dataset "Omdena_Repzone_v1.xlsx" contains information about sales representatives ("Representative" sheet) and customers ("Customer" sheet) and existing route (‘Route” sheet).

Representative Data: Contains 246 entries with 9 columns (expanded to 16 through feature engineering):

Representative data		
Column Names	**Description**	**Datatype**
RepresentativeId	Unique identifier for each representative	64 bit Integer
DailyMinCustomerCount	Minimum number of customers a representative can visit daily (always 1).	64 bit Integer
DailyMaxCustomerCount	Maximum number of customers a representative can visit daily (always 15).	64 bit Integer

WorkStartTime(hh:mm)	Representative's work start time.	String
WorkEndTime(hh:mm)	Representative's work end time.	String
WorkingDays	Days the representative works.	String
NoonBreakDuration (hh:mm)	Duration of the noon break.	String
BreakTimeBeforeNoon(hh:mm)	Duration of break time before noon.	String
BreakTimeAfterNoon(hh:mm)	Duration of break time after noon.	String

Customer Data: Contains 2654 entries, reduced to 2590 after cleaning, with 12 columns:(expanded to 17 with feature engineering):

Customer Data		
Columns	**Description**	**Datatype**
CustomerId	Unique identifier for each customer	64 bit Integer
VisitDuration	Duration of a visit	String
VisitFreq (1: Every Week, 2: Every 2 Weeks, 3: Every 3 Weeks, 4: Every 4 Weeks)	Visit frequency code.	String
MustVisitDays	Specific days a customer must be visited.	String
EligibilityBeginTime (hh:mm)	Start time for customer visit eligibility.	String
EligibilityEndTime (hh:mm)	End time for customer visit eligibility.	String
Country	Customer's country	String
City	Customer's city.	String
District	Customer's district	String

AddressText	Customer's address.	String
Latitude	Customer's latitude.	String
Longitude	Customer's longitude.	String

2. Data Cleaning and Feature Engineering

Representatives:

1. Conversion of time strings (e.g., "08:30") to minutes since midnight to facilitate calculations using the function time_str_to_minutes(time_str).
2. Creation of new features like WorkStartMinutes, WorkEndMinutes, NoonBreakMinutes, BreakBeforeNoonMinutes, BreakAfterNoonMinutes, TotalWorkMinutes, and TotalWorkHours.
3. Calculation of TotalWorkHours reveals that all representatives work 7.5 hours per day after accounting for breaks: "Working hours statistics: count 246.0 mean 7.5".
4. A WorkingDaysCount column is created.

Customers:

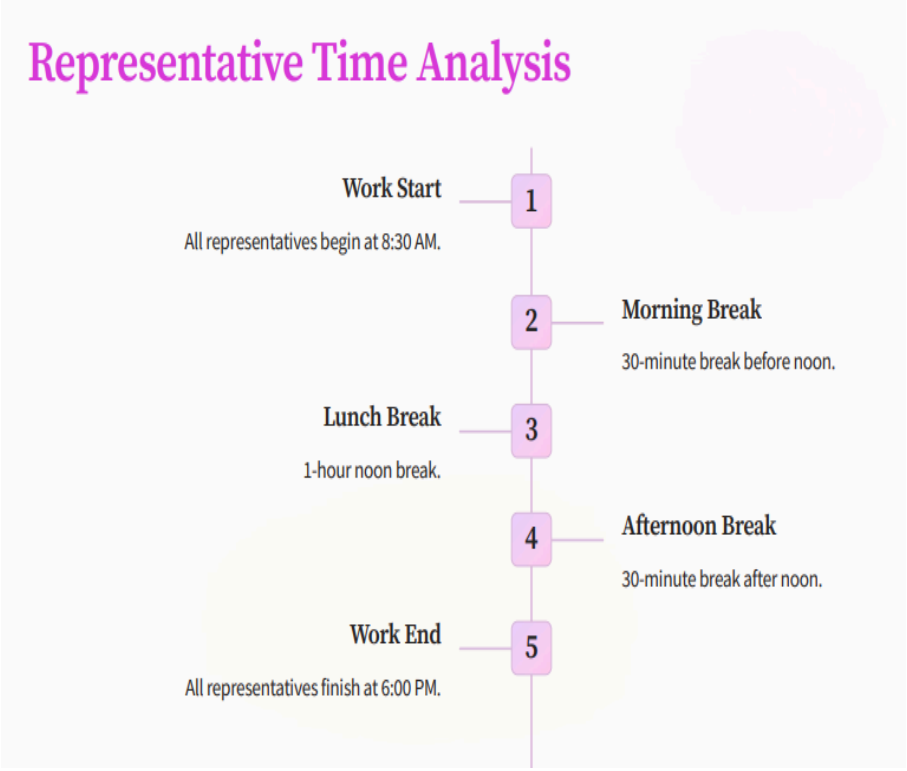
1. Renaming the 'VisitFreq' column for clarity.
2. Mapping numerical visit frequencies to descriptive labels (e.g., 1 -> "Weekly", 4 -> "Monthly"). This creates the VisitFreqDescription column.
3. Calculating 'EligibilityBeginMinutes' and 'EligibilityWindowHours'.
4. Handling missing 'Latitude' and 'Longitude' values by initially creating a subset customers_with_missing_coordinates. Rows *are* dropped to remove the NaNs for some of the visualisation, indicating that data completeness is less of a priority in this notebook.
5. Latitude and longitude coordinates with commas are replaced with periods, converting them to numeric types using: customers_df['Latitude'] = customers_df['Latitude'].str.replace(',', '.', regex=False) and customers_df['Latitude'] = pd.to_numeric(customers_df['Latitude'], errors='coerce').

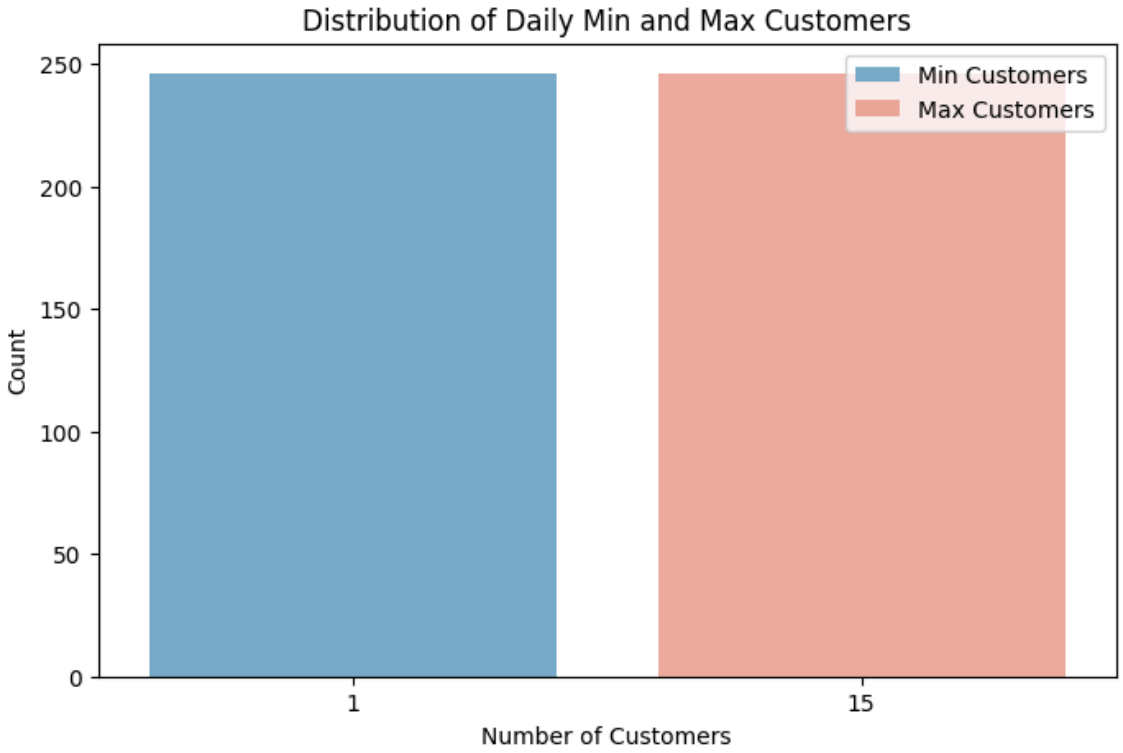
3. Insights

The exploratory data analysis (EDA) reveals several insights that are crucial for the subsequent data preprocessing steps. These insights are derived from the "Omdena_Repzone_v1_EDA.ipynb - Colab.pdf" source document.

Representatives Data Insights:

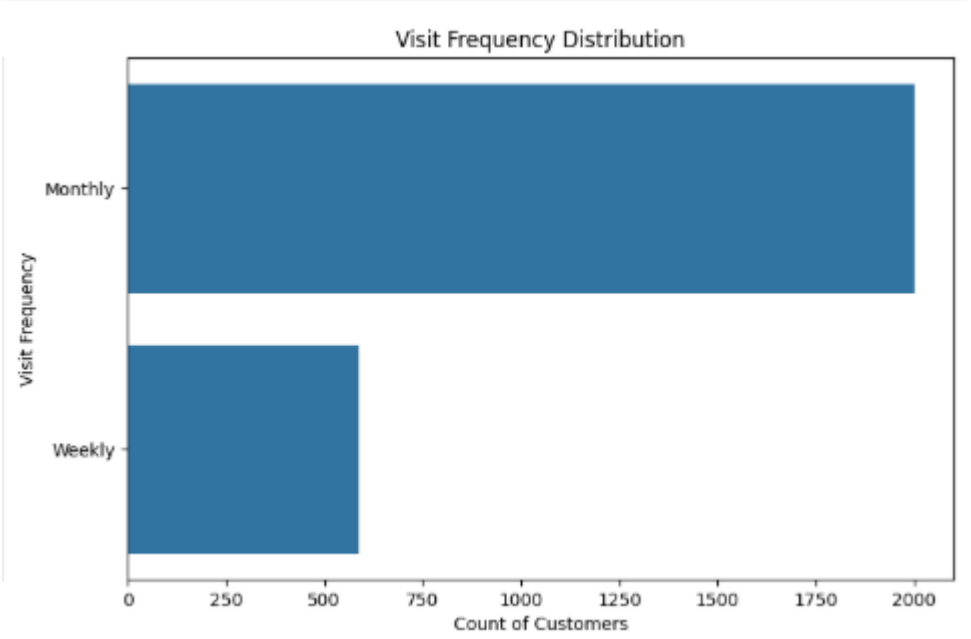
- **Data Types:** The representative data includes integer and object data types. Specifically, there are 3 integer columns and 6 object columns.
- **Customer Count Capacity:** Each representative has a minimum daily customer count of 1 and a maximum of 15.
- **Working Hours:** Representatives work for a total of 7.5 hours per day, after accounting for breaks.
- **Working Days:** Representatives work 6 days a week.
- **Consistent Schedules:** All representatives have the same working days ('Mon, Tue, Wed, Thu, Fri, Sat') and the same start and end times, as well as break durations.
- **Distributions:** Visualisations show the distribution of daily customer capacity, working hours, work start times, and work end times.

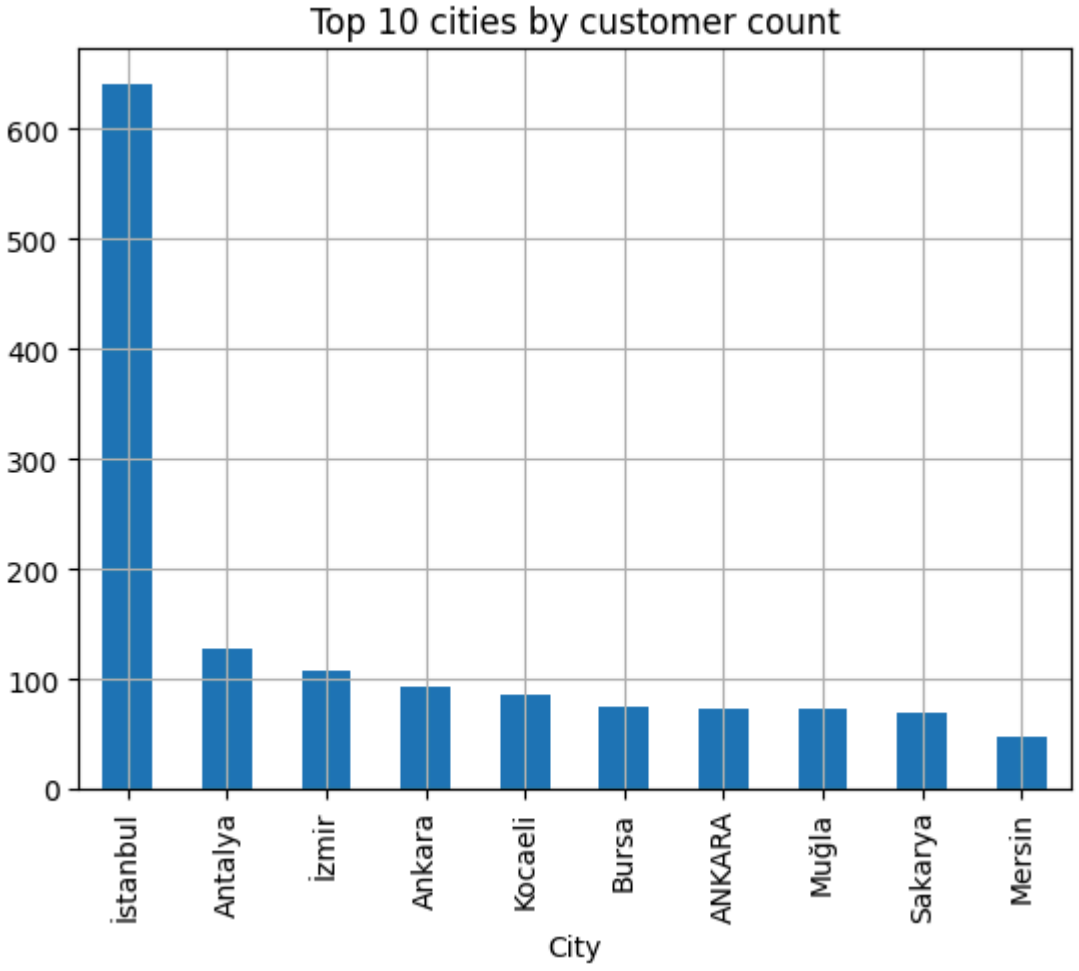




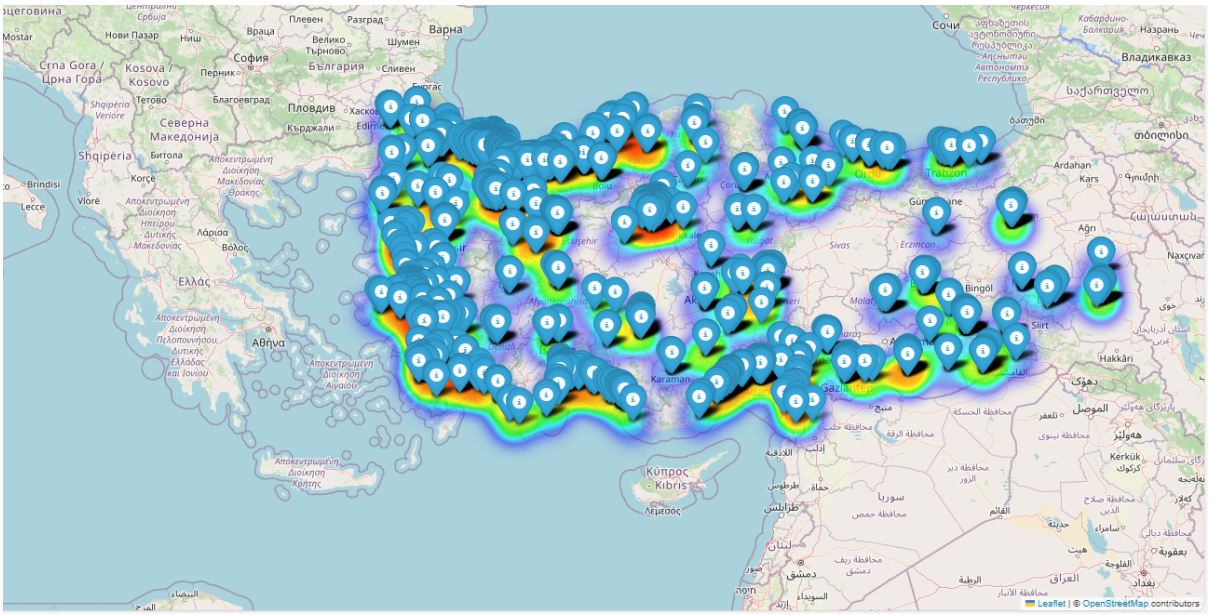
Customers Data Insights:

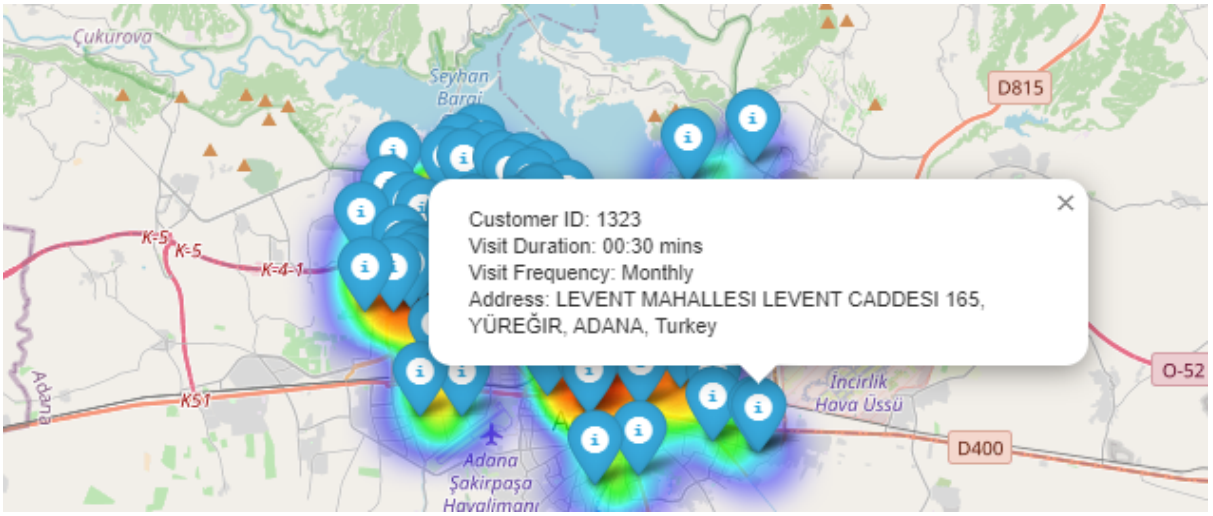
- **Total Customers:** There are 2654 customers in the dataset.
- **Visit Duration:** All visit durations are 00:30.
- **Visit Frequency:** Most customers (2048) are visited monthly, while 606 are visited weekly.
- **Eligibility Windows:** Customers have eligibility windows for visits, with most having a 9.5-hour window. Some have a 3.5-hour window.
- **Geographic Distribution:** The majority of customers are located in Istanbul being the city with the highest customer count of 641 followed by Antalya with a count of 128.
- **Missing Coordinates:** There are 64 missing latitude and longitude values in the dataset.
- **Distributions:** Visualisations show the distribution of visit duration, visit frequency, eligibility window size and eligibility start time.


Visit Frequency Distribution
Monthly
Weekly
Visit Frequency
0
250
500
750
1000
1250
1500
1750
2000
Count of Customers


Top 10 cities by customer count
600
500
400
300
200
100
0
İstanbul
Antalya
İzmir
Ankara
Kocaeli
Bursa
ANKARA
Muğla
Sakarya
Mersin
City

Map showing customer locations:





Cross-Dataset Insights:

- **Time Window Overlap:** The average representative working hours (8.50 - 18.00) align with the average customer eligibility hours (8.50 - 18.00).
- **Capacity Utilisation:** The capacity utilisation is 5.05%, indicating under-utilisation of the representatives.

Calculation of Capacity Utilization:

Weekly Representatives Capacity : Calculated the total weekly capacity of all representatives by:

1. Taking the sum of **DailyMaxCustomerCount** across all representatives - this gives the maximum number of customers that can be served per day by all representatives combined
2. Multiplying by the average number of working days per representative (**WorkingDaysCount**.mean())
3. The result is an estimate of how many customer visits the representatives can handle in a typical week.

Total weekly customer demand: Calculated the total weekly customer visit demand by:

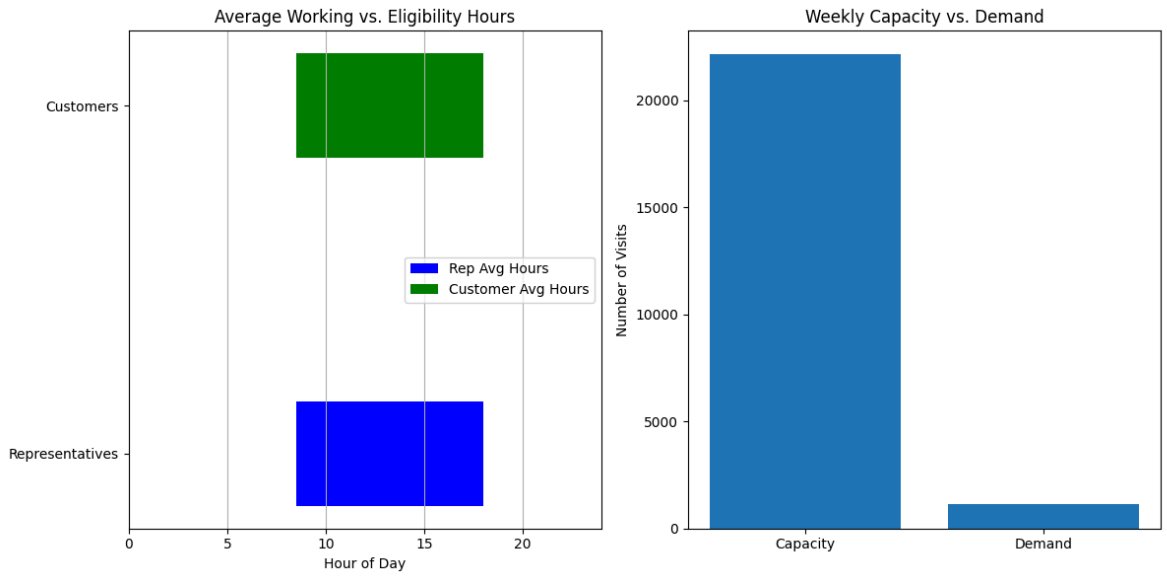
Converting each customer's visit frequency into a weekly equivalent:

If VisitFreq = 1 (weekly visits), then WeeklyVisitEquivalent = 1.0 visits per week If VisitFreq = 2 (bi-weekly visits), then WeeklyVisitEquivalent = 0.5 visits per week If VisitFreq = 4 (monthly visits), then WeeklyVisitEquivalent = 0.25 visits per week

Summing these weekly equivalents to get the total number of customer visits needed per week.

Capacity Utilization: Calculated what percentage of your total capacity is being utilized using the formula:

(total weekly demand) / (total weekly capacity) * 100



These insights provide a foundation for data preprocessing steps, including:

- Handling missing values in the customer data, particularly latitude and longitude.
- Encoding categorical variables such as city and district.
- Converting time-based columns to appropriate formats for analysis.
- Addressing the imbalance in visit frequency.
- Using the geocoded customer locations for spatial analysis and visualisation.

4. Key Themes & Observations

1. **Data Uniformity:** The representative data exhibits significant uniformity in terms of working hours, customer capacity, and working days. This suggests a standardized operational model.
2. **Under-Utilisation:** The capacity utilisation calculation indicates that the representatives have a significantly large unused capacity.
3. **Geographic Focus:** The customer base is heavily concentrated in Turkey, particularly in İstanbul.
4. **Data Quality:** Missing latitude and longitude data requires geocoding, indicating potential data quality issues.

5. Potential Next Steps

1. Explore strategies for improving capacity utilisation, such as optimizing routes, targeting specific customer segments, or adjusting visit frequencies.
2. Prioritize improving the completeness and accuracy of the customer location data.
3. Analyse the 'Route' sheet of the Excel file.
4. Further investigate the customers with a 3.5-hour eligibility window, as they are outliers.

Exploratory Data Analysis on Customer_route.csv data

1. **Data Overview**:

The dataset contains 2654 entries with information about customer routes, including customer details, visit schedules, geographical coordinates (latitude and longitude), regional and city information, and route specifics like the representative assigned, day of the week, and visit order.

Region column: The region names are fetched for a given city in Turkey by querying the Nominatim API from OpenStreetMap. It does this by using the city name to retrieve detailed location information, and then it extracts the region (e.g., province) from the address details.

Geocoding missing addresses: Used Google Maps and Selenium for automating the retrieval of latitude and longitude coordinates for a list of addresses.

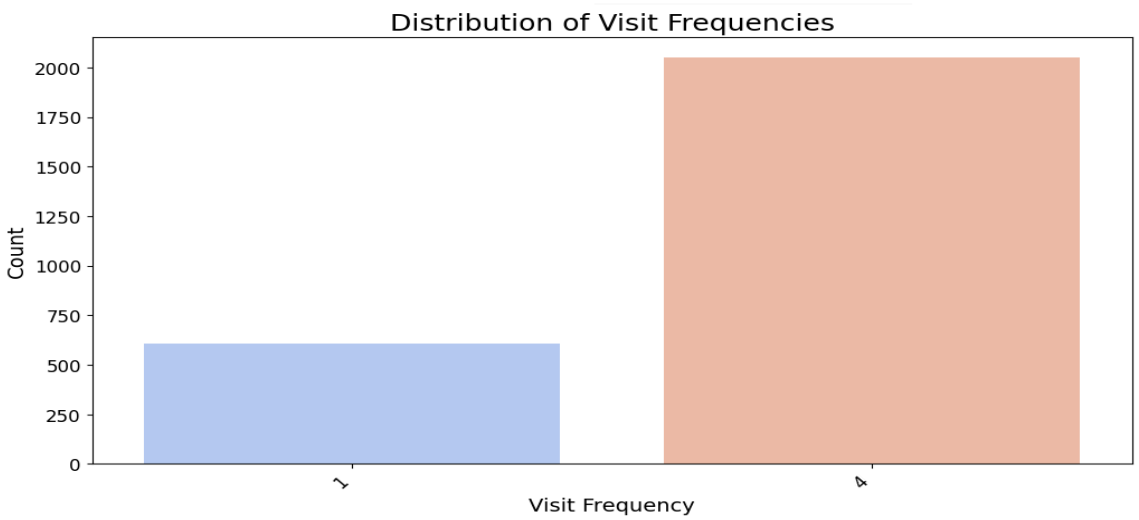
Data Analysis:

Redundant Information: The column 'RouteFrequency\n(1: Every Week, \n2: Every 2 Weeks, \n3: Every 3 Weeks, \n4: Every 4 Weeks)' was found to be redundant with 'VisitFreq' and was removed from the DataFrame.

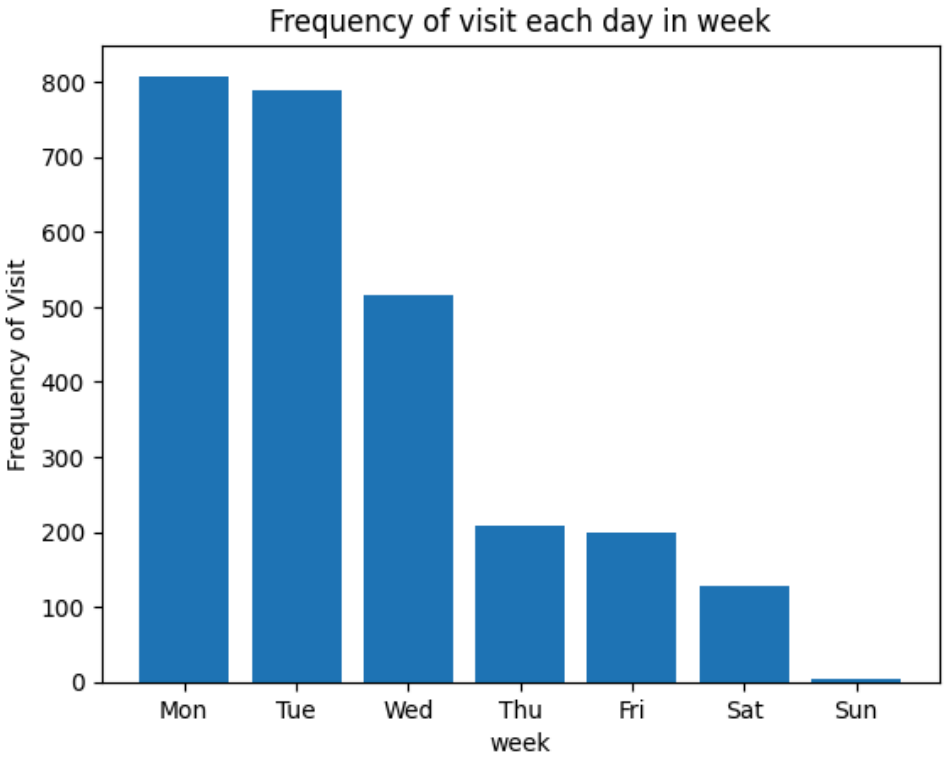
Visit Frequency Distribution: The analysis of 'VisitFreq' shows the distribution of different visit frequencies across the customer base.

Popular Visit Week:

Analyzing the frequency of visits on each day of the week can help in **optimizing routes and allocating resources effectively** by identifying days with higher visit loads. The visualization shows the distribution of visits across the week, highlighting potential peak days.We can observe that mostly **visits are on monthly basis**.



Frequency of Visit Each day in week:



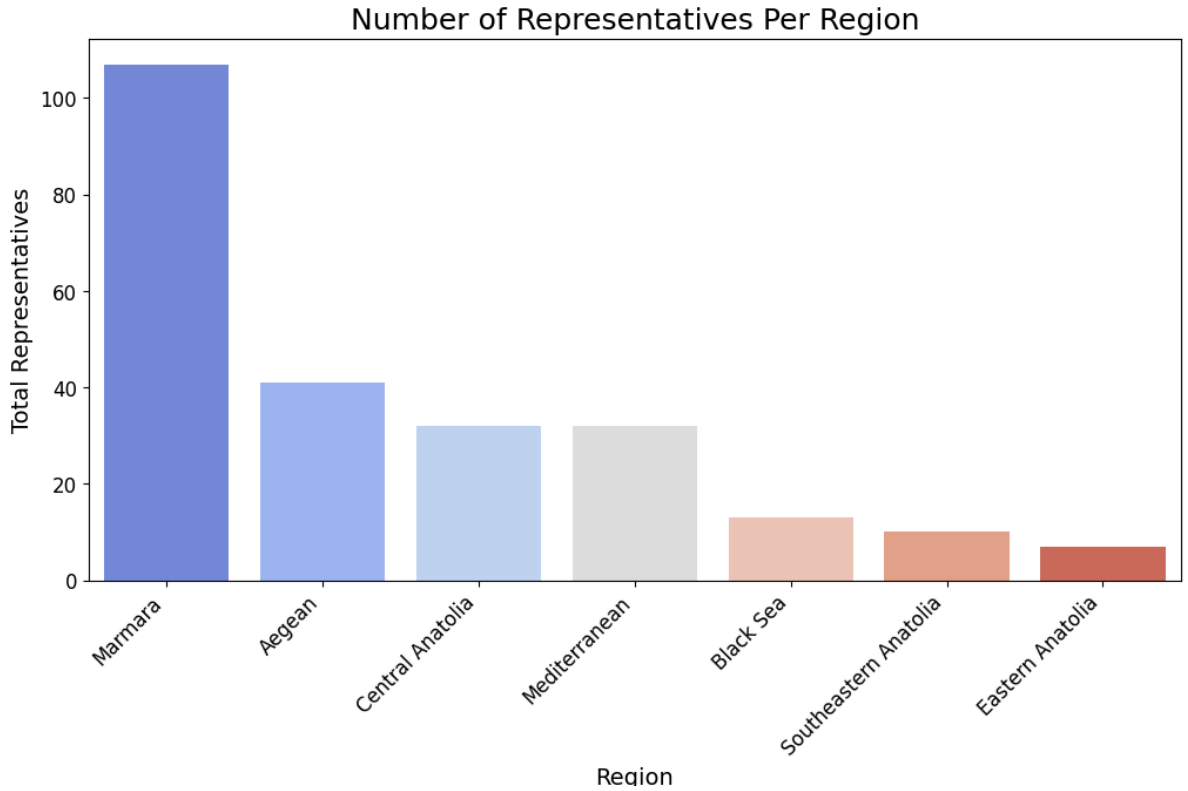
Monday has the highest route count (808), indicating it is likely a **peak business day**.
Tuesday (789) and Wednesday (516) still have a significant number of routes, though slightly lower than Monday.
Thursday (209) and Friday (199) see a sharp decline, suggesting a gradual decrease in demand as the weekend approaches.

Saturday (129) and Sunday (4) have the lowest counts, implying that very few routes operate on weekends .

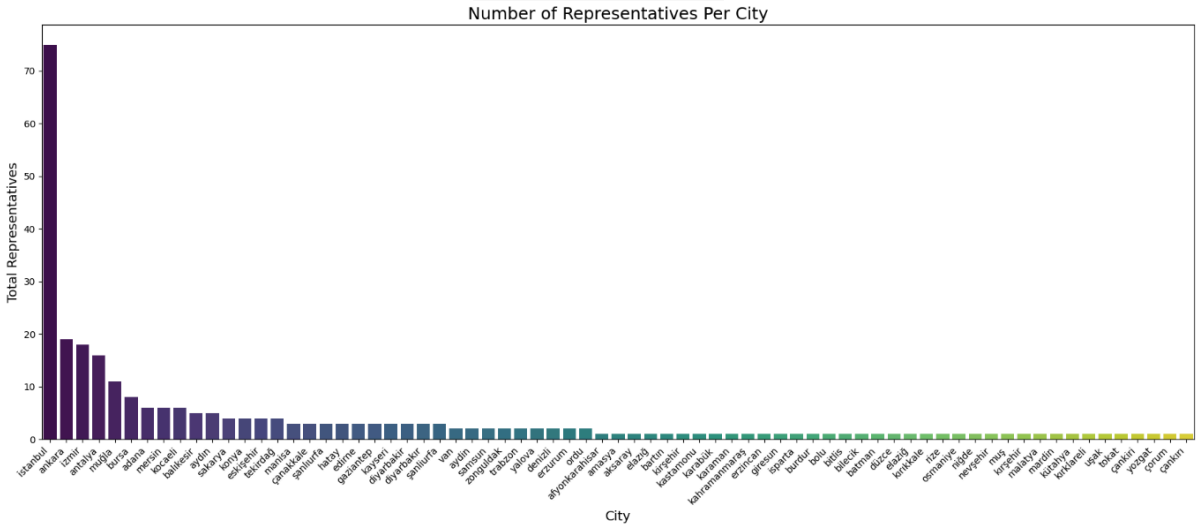
Representative Allocation by Region::

	region	TotalRepresentatives
4	Marmara	107
0	Aegean	41
2	Central Anatolia	32
5	Mediterranean	32
1	Black Sea	13
6	Southeastern Anatolia	10
3	Eastern Anatolia	7

There is a varying number of representatives assigned to different regions. **Marmara has the highest number of representatives (107)**, significantly more than other regions like Aegean (41), Central Anatolia (32), and Mediterranean (32). This suggests a potentially higher customer density or strategic focus in the Marmara region.

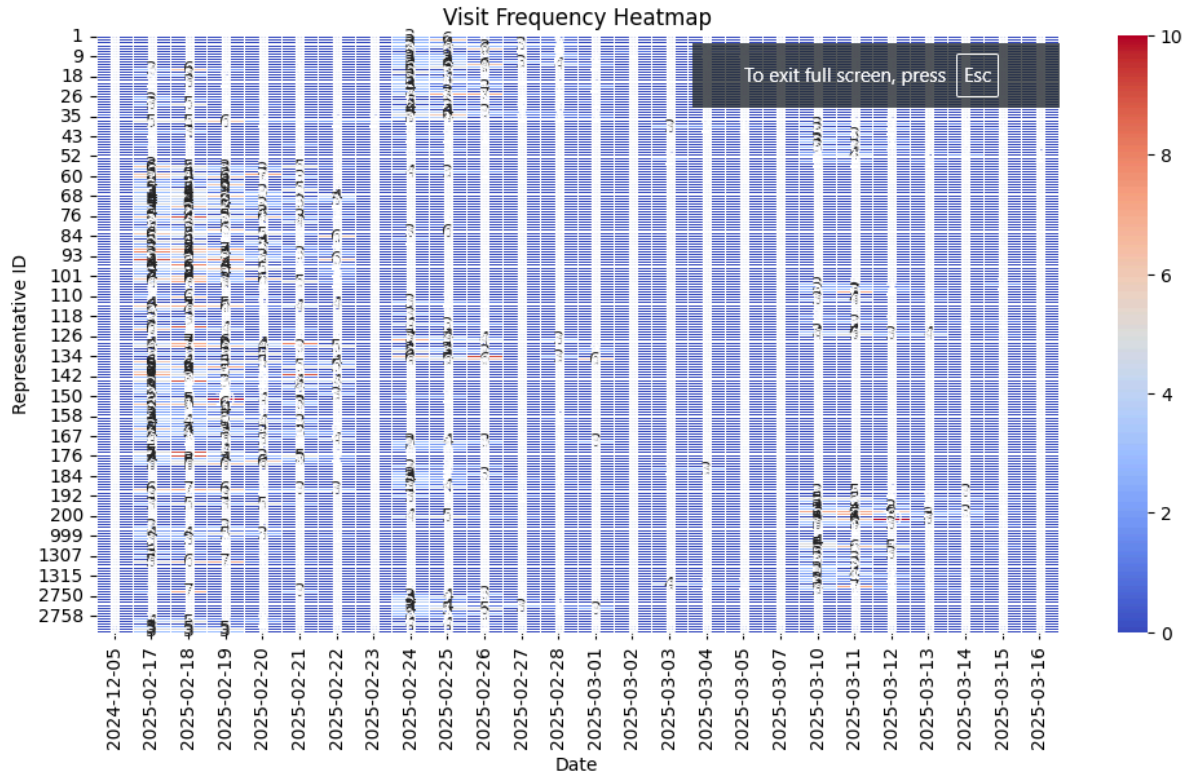


Representative Allocation by City:



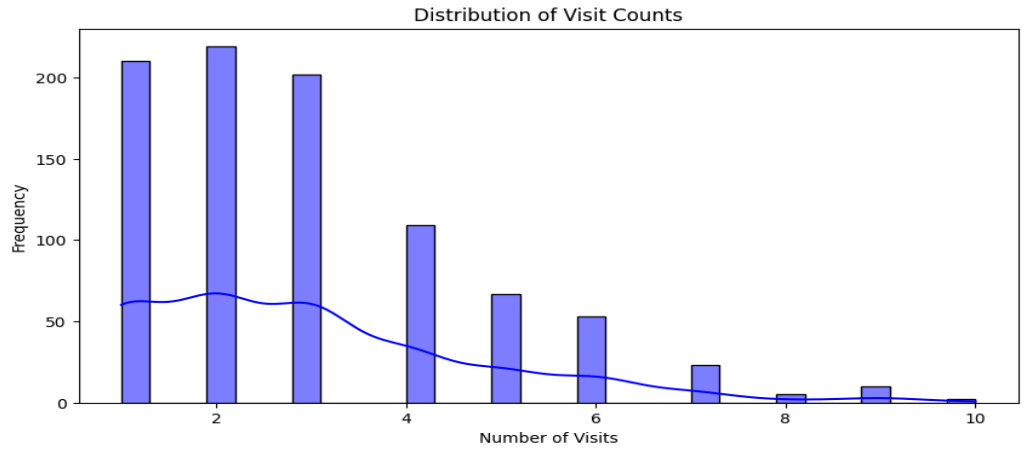
The analysis of representatives per city, after normalizing city names, helps identify cities with a higher concentration of sales representatives, indicating potentially **high-demand areas : Istanbul , Ankara, Izmir.**

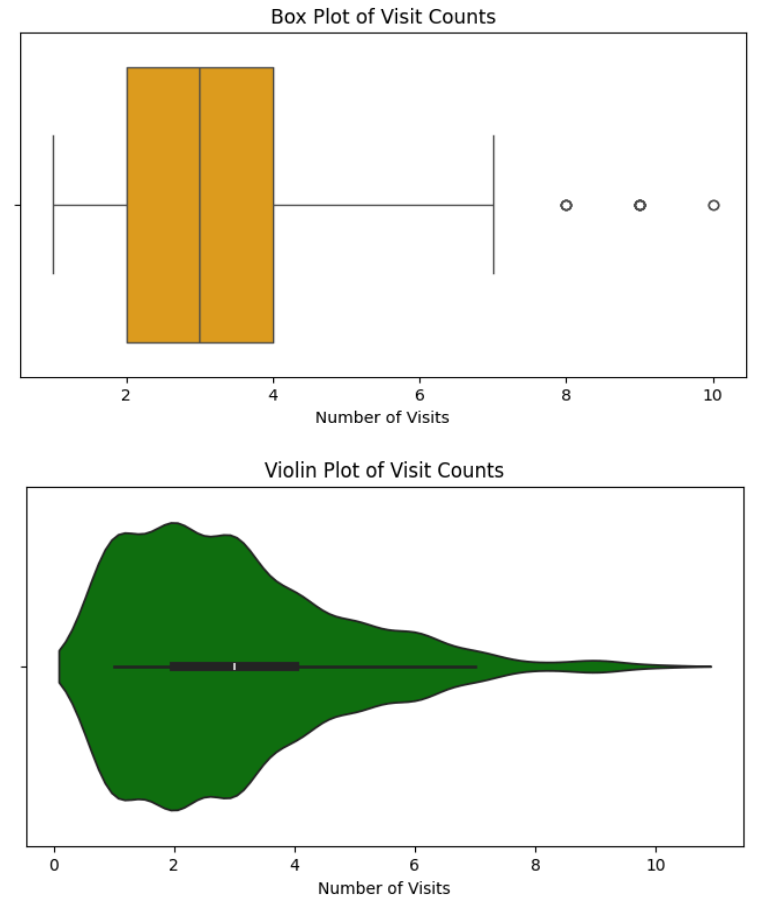
Visit Frequency per Representative per Day:



Examining the number of visits made by each representative on different days reveals That most of the working days the **Representative visits 2 to 3 Customers**. The presence of outliers with higher visit counts (e.g., up to 10 visits) on certain days for some representatives is noted.

Distribution of Visit Counts Vs Frequency:

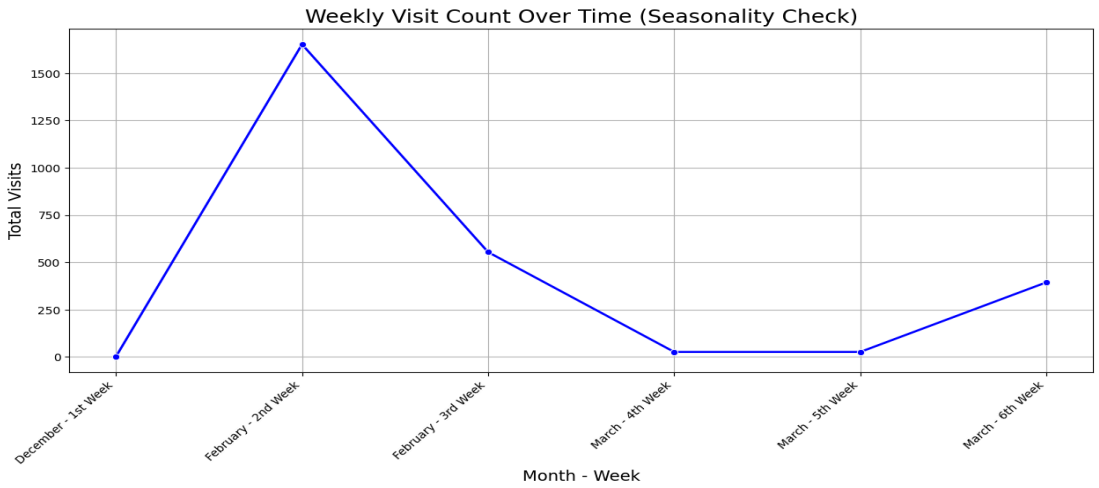




The distribution of the number of visits per representative per day is **right-skewed**. This means that while most representatives make a 2-3 number of visits,with median as 3, there are some who make a considerably higher number of visits (8,9,10 visits)in a day. This skewness might be relevant if the data were used for training deep learning models, potentially leading to issues like the vanishing gradient problem.

`Right-Skewness: 1.09 which is considered very high.`

Seasonality Check:



The analysis includes an attempt to check for seasonality in visit frequency by examining weekly visit counts over time. The visualization of weekly visits can help identify any recurring patterns or trends in customer visits throughout the year,Which can be used to forecast the visit frequency and manage resource allocation.

- Peak in February (2nd Week): A sharp surge above **1,500 visits, likely due to a seasonal event** .
- Rapid Decline After Peak: Visits drop significantly from the 3rd week of February, reaching a low in March.
- March Low Activity: 4th & 5th weeks show minimal visits, suggesting an off-season.
- Slight Recovery (6th Week of March): A small increase, possibly signaling the start of a new cycle.

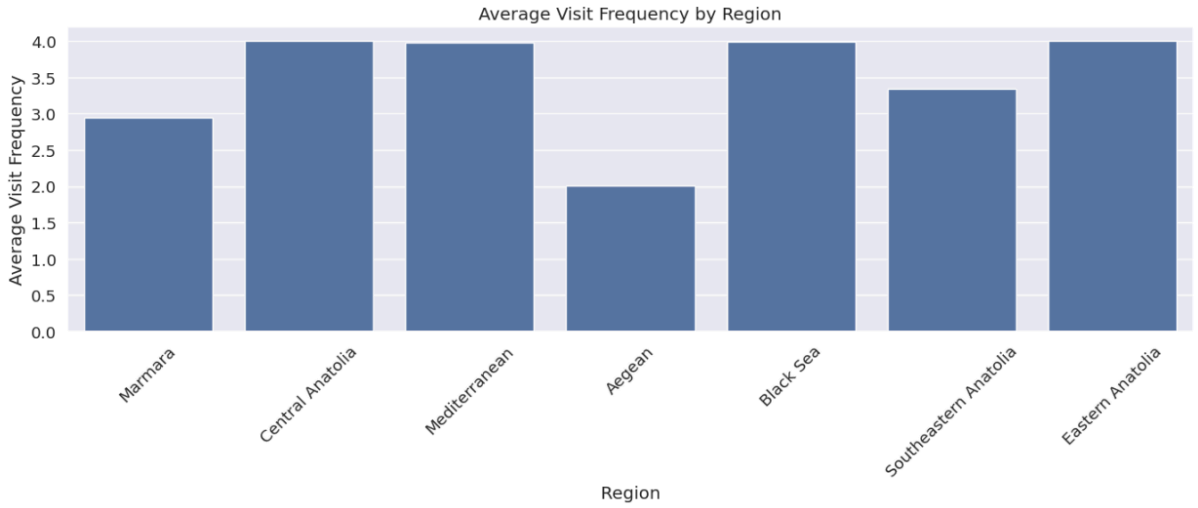
Average visit frequency by Region

1. The visit frequency varies across regions.

2. Central Anatolia, Mediterranean, Black Sea, and Eastern Anatolia have the highest average visit frequencies (around 4).

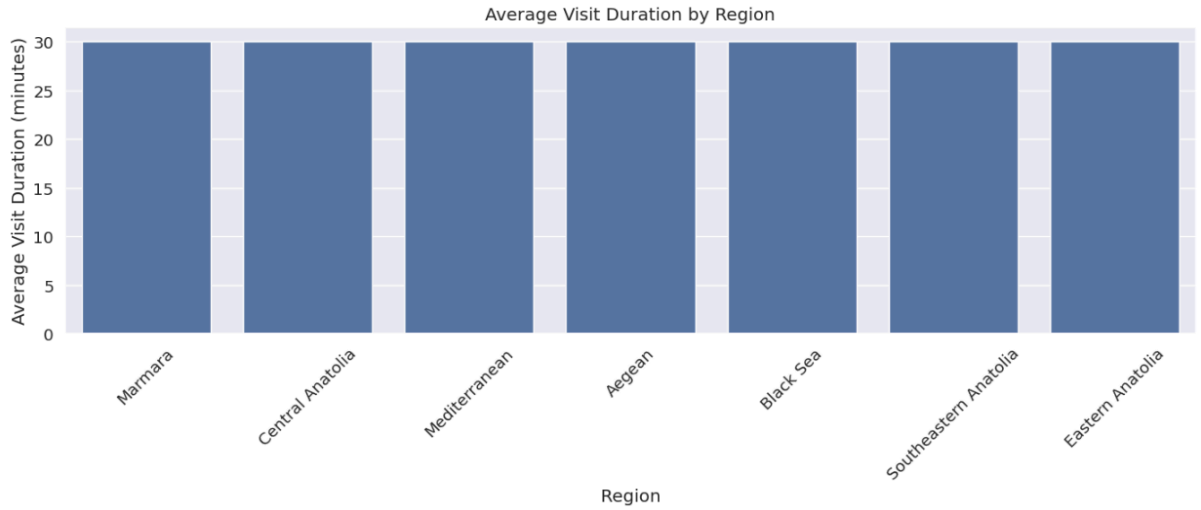
3. Marmara and Southeastern Anatolia have slightly lower visit frequencies.

4. Aegean has the lowest average visit frequency among all regions



Average visit duration by region

All regions show similar average visit durations (30 minutes), indicating that visit length does not vary significantly by region.

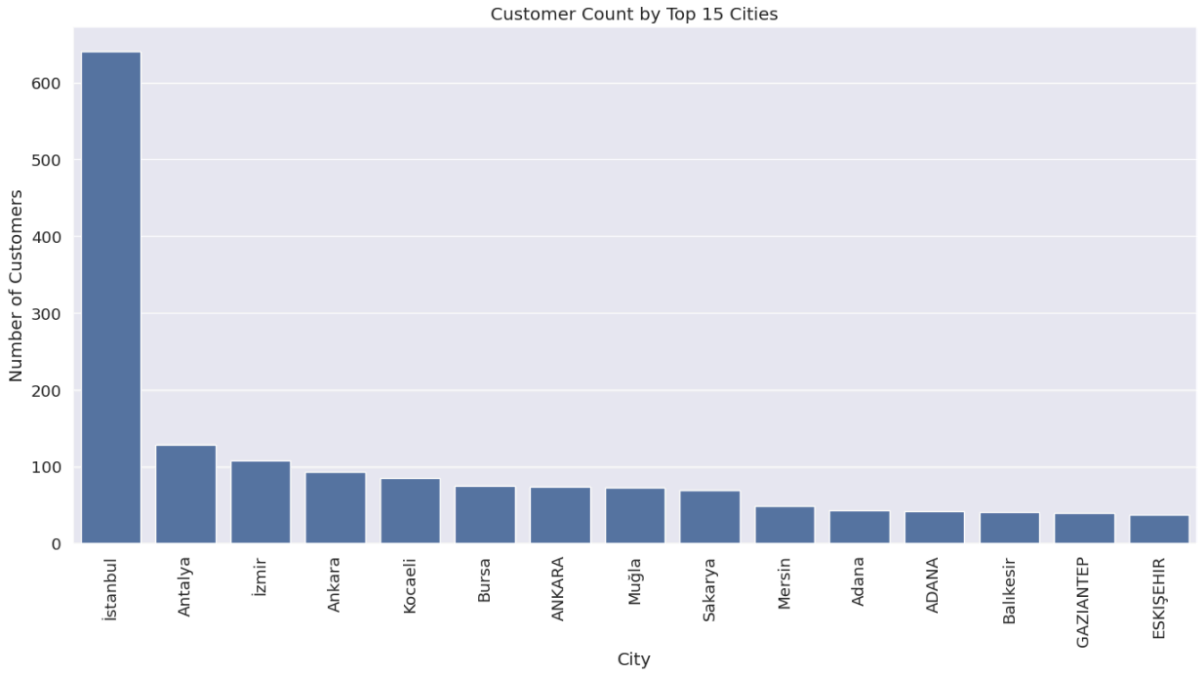


Customer Count by Top 15 Cities

1. Istanbul dominates, having the highest customer count by a large margin (over 600 customers).

2. Other major cities such as Antalya, İzmir, and Ankara have significantly lower but still notable customer counts.

3. The distribution shows a steep drop-off after the top cities, indicating that customer presence is highly concentrated in a few key urban areas.

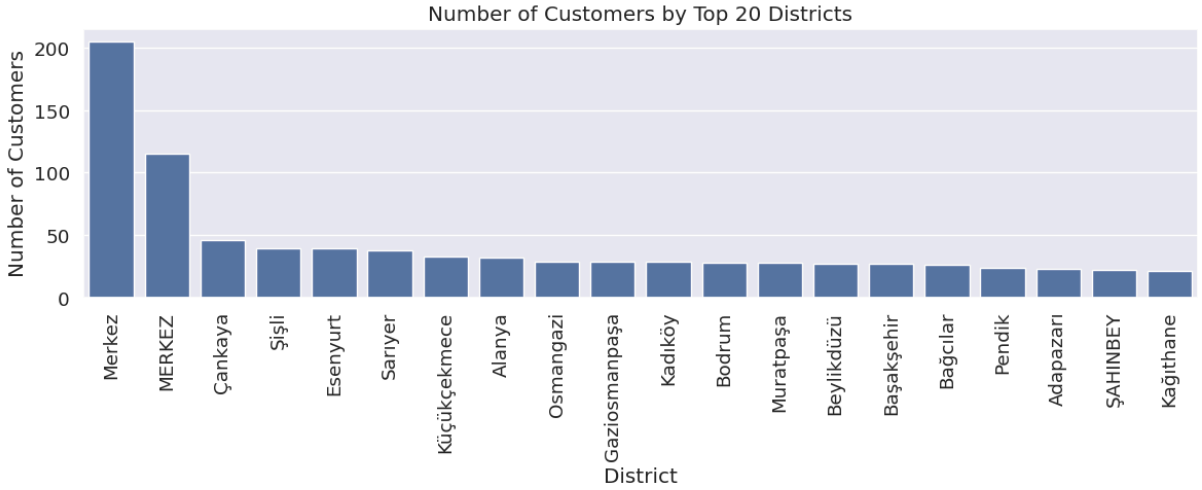


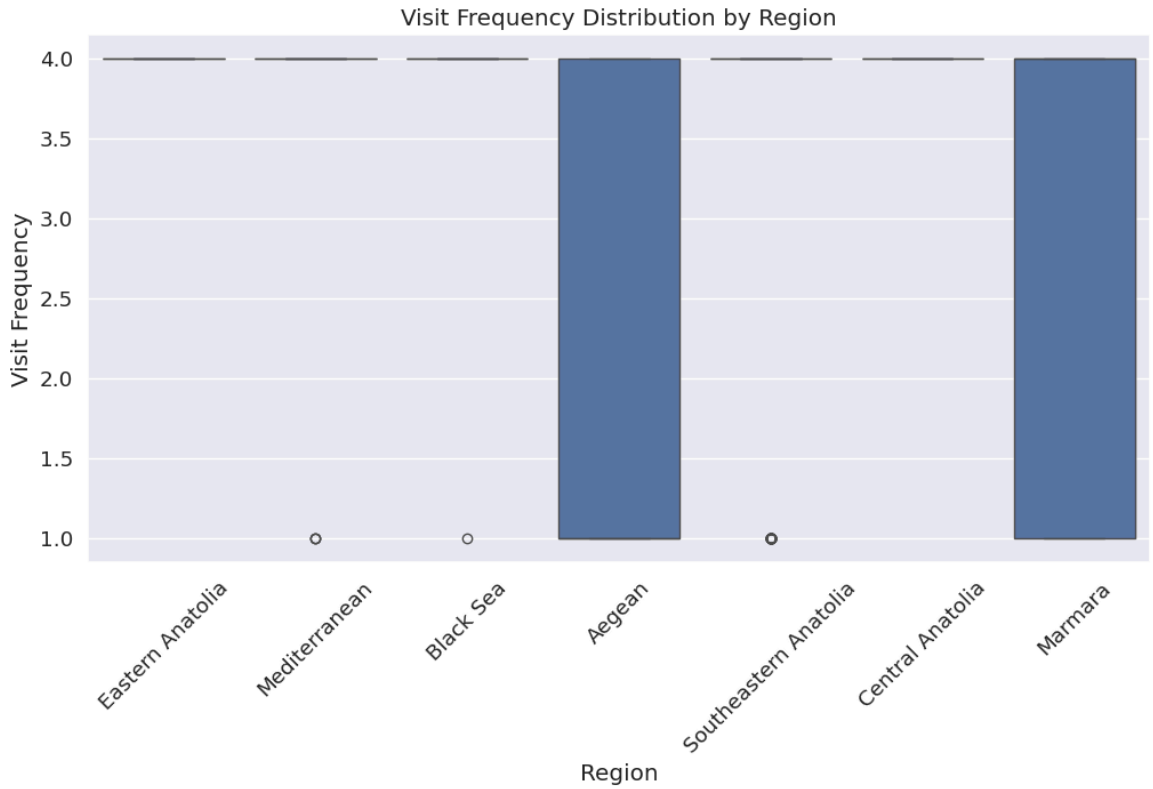
Number of Customers by Top 20 Districts

1. "Merkez" and "MERKEZ" (potential duplicate entries) have the highest number of customers by a large margin.

2. Other districts such as Çankaya, Şişli, and Esenyurt have moderate numbers of customers.

3. The remaining districts have relatively lower customer counts.





1. The distribution of visit frequencies shows variations in visit patterns.

2. Marmara and Aegean have visit frequencies concentrated at 4, indicating they are consistently visited at this frequency.

3. Other regions show a mix of high and low visit frequencies, with some outliers present.

4. Some regions, like the Mediterranean, Black Sea, and Southeastern Anatolia, have lower outliers, suggesting that while they have high visit frequencies on average, there are instances of low visit counts.

Overall Takeaways:

- Visit frequency and duration are region-dependent, with Central Anatolia and the Mediterranean seeing frequent visits, but the duration of visits remains consistent across regions.

- Istanbul has a significantly higher number of customers compared to other cities, emphasizing its importance as a key hub.

- Certain districts (Merkez, Çankaya, Şişli) are customer hotspots.

- Data cleaning might be needed for district names (e.g., "Merkez" vs. "MERKEZ").

2. Potential Next Steps:

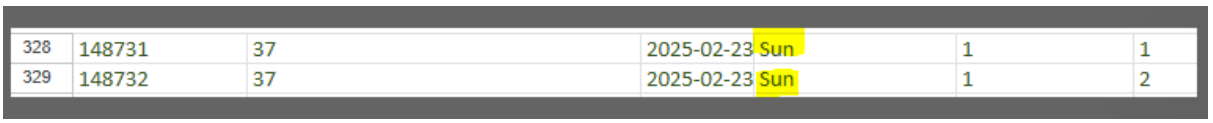
The insights about the distribution of visit frequencies, the number of representatives per region and city, and the visit frequency per representative per day could be further enriched by incorporating travel distance calculations. For example:

1. Understanding the number of representatives in high-demand cities and their daily visit counts could be combined with estimated travel distances to assess the efficiency of representative allocation.
2. The right-skewed distribution of visit counts might correlate with the estimated travel distances, suggesting some representatives might be covering larger areas or having more densely located customers.

Having the start location of representatives, would further enhance these insights by allowing for the calculation of the total route travel distance, segment-wise travel distances, and more refined travel efficiency metrics.

Queries to be answered:

- In the 'Representative' data, the working days is mentioned as Mon, Tue, Wed, Thu, Fri, Sat. Whereas in the 'Route' data, there are deliveries on Sun as seen in the image below:

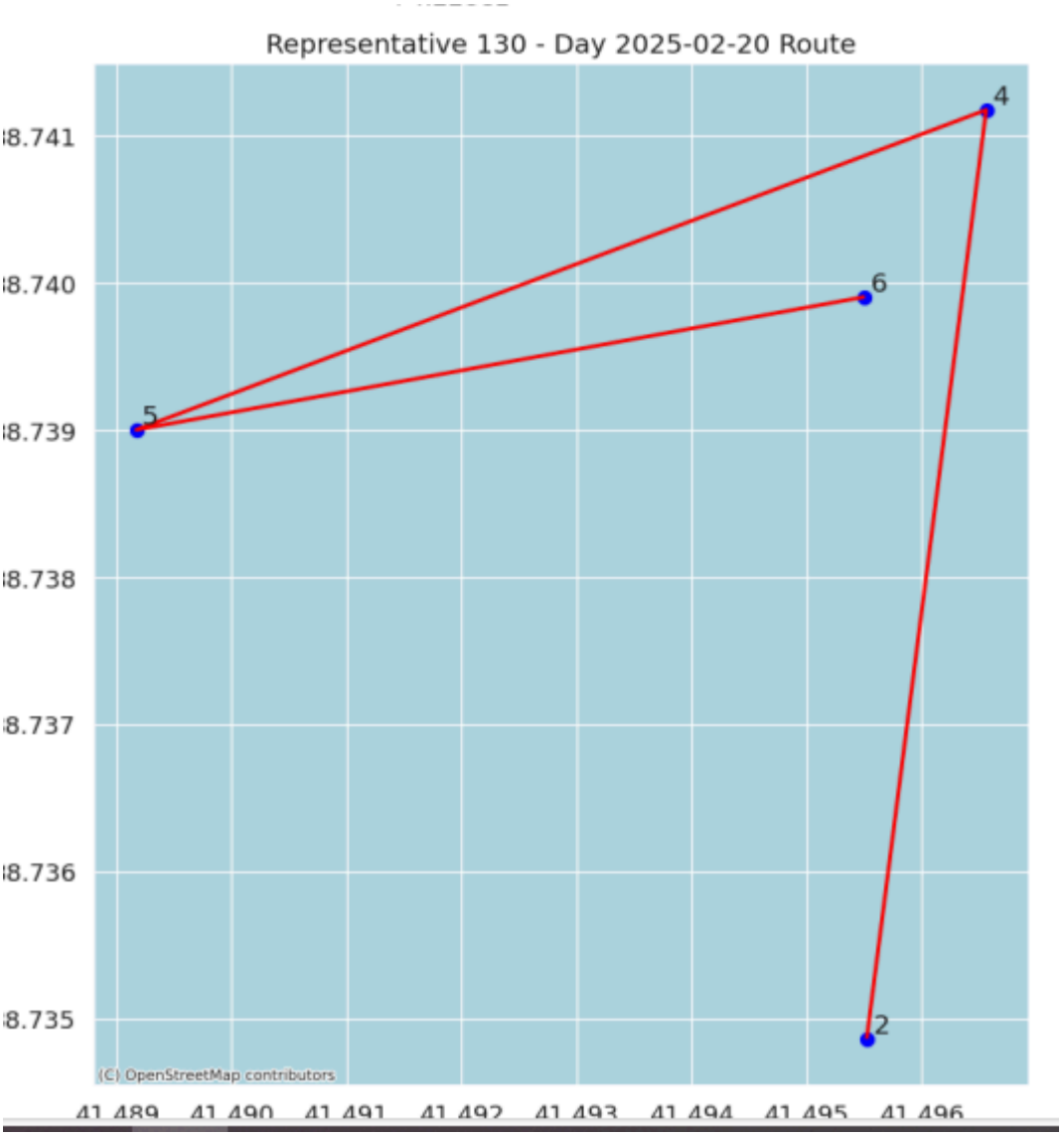


328	148731	37	2025-02-23	Sun	1	1
329	148732	37	2025-02-23	Sun	1	2

- The **Must Visit** column has only four instances: **2 Saturday and 2 Monday**. Including other days as "Other" may reduce the weightage of Saturday and Monday. How can we ensure that Must Visit days retain their significance?

- In the Route Visit order, consider representative id 1, on 2025-02-24, the visit order is 1-->2-->3, on 2025-02-25, the visit order is 1-->3, on 2025-02-26, there is one delivery with order as 2, and on 2025-02-28, there is one delivery with order as 2. Is this right?

1	RouteCustomerId	RouteRepresentativeId	RouteDay	RouteDayOfTheWeek (Mon, Tue, Wed, Thu, Fri, Sat, Sun)	RouteFrequency (1: Every Week, 2: Every 2 Weeks, 3: Everey 3	RouteVisitOrder
2	9830	1	2025-02-24	Mon	1	1
3	4234	1	2025-02-24	Mon	1	2
4	13297	1	2025-02-24	Mon	1	3
5	4281	1	2025-02-25	Tue	1	1
6	13298	1	2025-02-25	Tue	1	3
7	9189	1	2025-02-26	Wed	1	2
8	9257	1	2025-02-28	Fri	1	2



- There is no location for the Representatives. If we have this data, we can calculate the distances from the start point.
- While checking the seasonality in the number of visits over time, from the below plot we suspect that there might be a seasonality trend. To confirm this, we need more data may be historical data. This will help to forecast the demand and do the route planning accordingly.

Data cleaning:

Removing duplicates (**"Aydin" vs. "Aydın"**, **"Diyarbakir" vs. "Diyarbakır"**, etc.) and inconsistent capitalization issues (e.g., `İZMIR` vs. `İzmir`).

Defined a function which does the following and applied this function on the ‘City’ and ‘District’ columns on the dataset “ customers_with_region_and_priority.csv”.

Comprehensive Character Normalization

1. Uses Unicode normalization to strip accents,
2. Converts to ASCII representation,
3. Handles multiple variations of similar-looking characters.

Regex-based Replacements

1. Specifically targets variations like Şişli/Sisli,
2. Standardizes different representations of characters like 'S', 'Ş', 'I', 'İ',
3. Case-insensitive matching.

Flexible Preprocessing

1. Handles various input formats,
2. Supports uppercase, lowercase, or no case transformation,
3. Optional duplicate removal.

Result: **Number of unique cities: 65**

Number of unique districts: 274

Final dataset: customers_with_region_and_priority_V2.csv Link
Link to notebook:

Geocoding

Geocoding is the process of converting an **address, place name, or location description** into **latitude and longitude coordinates**.

For example:

- **Input Address** → `"1600 Amphitheatre Parkway, Mountain View, CA"`
- **Geocoded Output** → `(37.4221, -122.0841)`

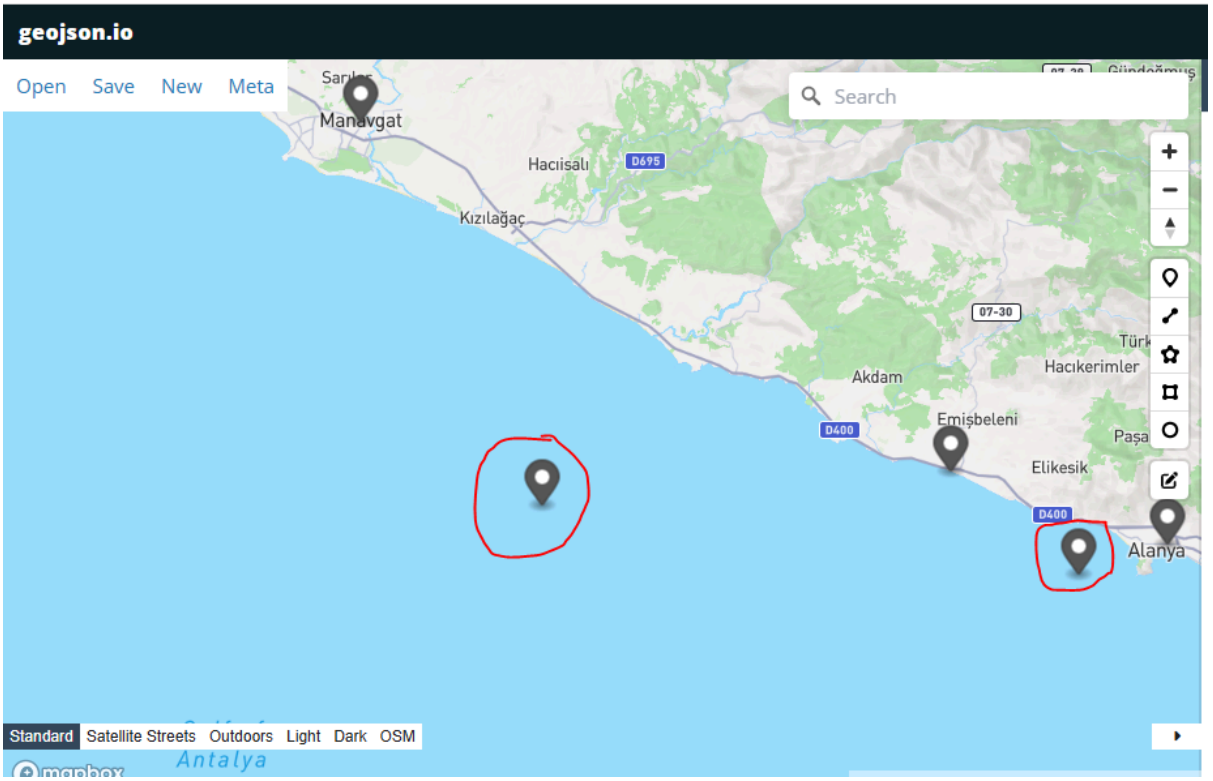
Issues faced:

Geocoding the missing addresses:

There are 64 missing locations for 64 Customers in the ‘Customer’ dataset.

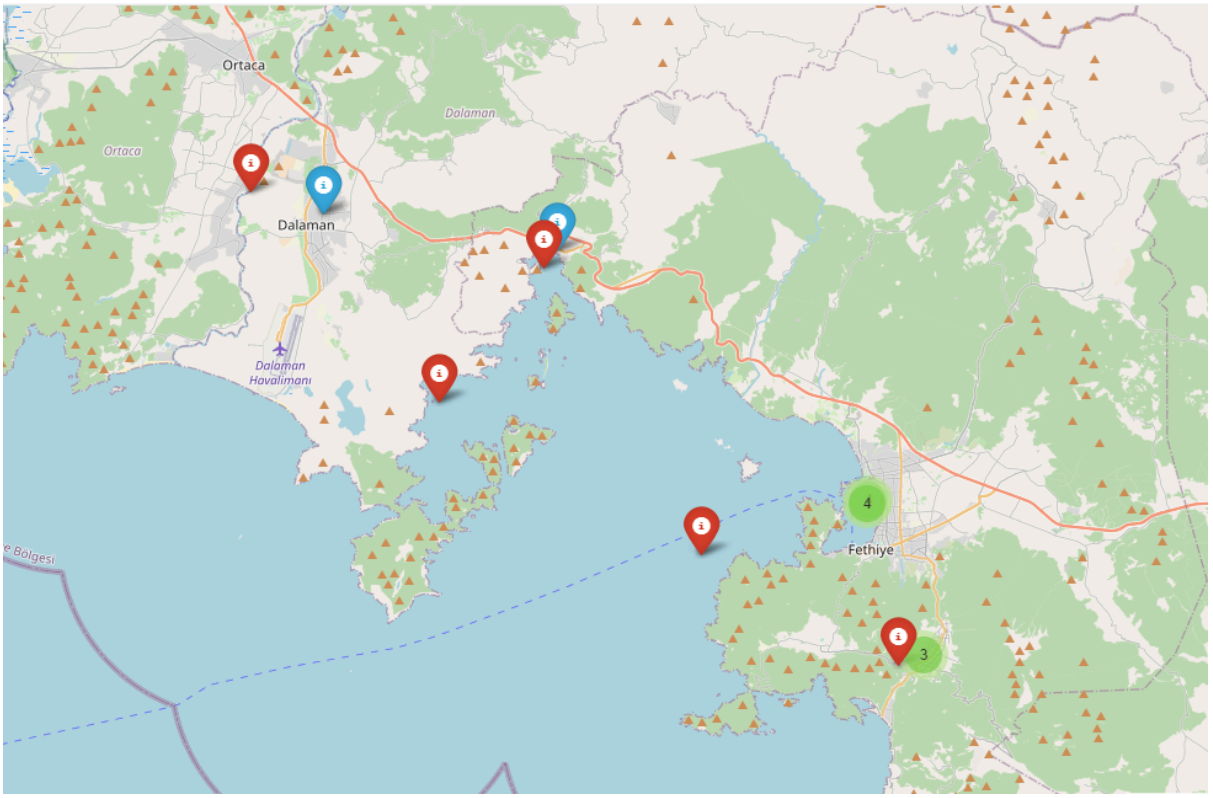
Approaches used to Geocode:

1) Used scraping method to scrape the google maps using selenium.
    a) Issue: Some of the coordinates were located on the sea which is not correct as in the figure below:
    Filename: geocoded_addresses.csv
    Link



2) Used ArcGIS API from geopandas.tools.
    All except one address was not decoded by the API. And all the locations geocoded are located on the land.
    Address not geocoded: Tosmur Mah. 8. Sok. No:1 A/A,A/B,A/C,A/D Alanya/Antalya ,Alanya,Antalya,Turkey

Plotted a map showing the locations obtained as a result of using both approaches.
Link to the map.



Those marked in red correspond to locations obtained using approach 1 (scrapping)
Those marked in blue correspond to locations obtained using approach 2

Link to the geocoded addresses using ArcGIS API: Link

ArcGIS ensures the accuracy of its geocoding data through a combination of high-quality reference datasets, advanced geocoding methods, and tools for precision assessment. Below are the key mechanisms: (Source: perplexity.ai)

1. Use of Authoritative Reference Data

- ArcGIS relies on authoritative datasets, including commercial street data (e.g., HERE, TomTom), government data (e.g., Census TIGER), and Esri's own curated databases. These datasets are regularly updated to ensure current and reliable address information36.

- The choice of dataset can significantly affect accuracy. Commercial datasets typically provide higher precision (e.g., rooftop-level geocoding), while public datasets may be less detailed35.

2. Cascading Geocoding Functionality

- ArcGIS employs cascading functionality to match input addresses to the highest possible precision level. If a rooftop-level match is unavailable, it cascades to the next best level, such as street segment or postal centroid[3][6].

- This ensures that even when exact matches are not possible, the best available alternative is provided.

3. Match Levels and Precision Indicators

- The geocoding service outputs a field called `Addr_type`, which indicates the precision level of a match (e.g., PointAddress for rooftop-level accuracy or Postal for centroid-level accuracy)[1][2].

- Match scores (ranging from 0 to 100) assess confidence in the geocoded result. A higher score indicates greater reliability[5].

4. Support for Global Coverage and Local Formats

- Esri World Geocoder supports address-level geocoding in over 130 countries and place-level coverage globally. It accommodates local languages, alphabets, and formats to improve accuracy in diverse regions[6].

5. Customizable Locators

- Users can adjust locators to meet specific needs, such as prioritizing rooftop matches or ensuring results align with routing requirements (e.g., placing points on street sides)[5].

- Features like "Match out of range" allow flexibility in handling incomplete or slightly outdated address data[5].

6. High-Accuracy GPS Integration

- For field data collection, ArcGIS supports high-accuracy GPS receivers that provide submeter to centimeter-level precision. This is especially useful for applications requiring extreme accuracy, such as infrastructure mapping[4].

7. Error Mitigation Through Datum Transformations

- ArcGIS minimizes errors introduced by coordinate system mismatches through datum transformations, ensuring spatial consistency between GPS data and map layers4.

By combining these features with robust tools for analysis and visualization, ArcGIS delivers geocoding solutions that are both accurate and adaptable to various use cases.

Citations:

1. https://community.esri.com/t5/addressing-questions/geocoding-precision-how-to-determine/td-p/832622
2. https://developers.arcgis.com/rest/geocode/api-reference/geocoding-service-output.htm
3. https://www.esri.com/arcgis-blog/products/analytics/analytics/geocoding-delivering-high-location-accuracy/
4. https://doc.arcgis.com/en/field-maps/latest/prepare-maps/high-accuracy-data-collection.htm
5. https://pro.arcgis.com/en/pro-app/latest/help/data/geocoding/tips-for-improving-geocoding-quality.htm
6. https://www.esri.com/arcgis-blog/products/analytics/analytics/location-is-everything-geocode-to-success-with-arcgis/
7. https://desktop.arcgis.com/en/arcmap/latest/manage-data/geocoding/what-is-geocoding.htm
8. https://www.esri.com/about/newsroom/arcuser/efficient-geocoding-with-arcgis-pro/

Distance Matrix calculations

We have identified two libraries which can be used for this purpose.

1) Using OSRM
2) Using OSMnx
3) Using ORS

OSRM (Open Source Routing Machine) and OSMnx are both libraries for working with routing and OpenStreetMap data, but they serve different purposes and have distinct functionalities.

Links →

1. https://blog.afi.io/blog/osrm-table-api-free-and-open-source-distance-matrix-api/
2. https://neocarto.github.io/rspatial/osrm/osrm.html

OpenRouteService (ORS) and Open Source Routing Machine (OSRM) are not the same, though both are routing engines that use OpenStreetMap (OSM) data. Here are the differences:

Key Differences

1. **Purpose and Features**:
    - **OSRM**: A high-performance routing engine optimized for speed, focusing on shortest path calculations. It supports car, bicycle, and pedestrian routing modes but lacks features like isochrone generation or advanced customization[1][2][4].
    - **OpenRouteService (ORS)**: Offers broader functionality, including isochrone generation, geocoding, and matrix routing. It supports more transportation modes (e.g., car, bike, foot, wheelchair) and provides advanced features like elevation profiles and turn-by-turn navigation[7].
2. **Customization**:
    - **OSRM**: Limited customization options for profiles; primarily designed for fast routing with predefined modes[4].
    - **ORS**: Allows more user-defined profile modifications and supports diverse applications like accessibility analysis[7].

3. **Performance**:
    - **OSRM**: Known for its speed, handling continental-sized networks within milliseconds using contraction hierarchies[1][2].
    - **ORS**: While slower than OSRM for large-scale routing tasks, it compensates with versatility in features[7].
4. **Real-Time Data Integration**:
    - **OSRM**: Does not support real-time traffic data integration[3].
    - **ORS**: Can incorporate real-time data for dynamic routing scenarios.
5. **Use Cases**:
    - **OSRM**: Ideal for applications requiring rapid route calculations over large networks.
    - **ORS**: Better suited for diverse analytical tasks, accessibility studies, and applications needing detailed route information.

In summary, OSRM is optimized for speed and simplicity in routing tasks, while OpenRouteService offers a broader range of features and customization options for varied use cases.

Citations:

1. https://stackshare.io/stackups/openstreetmap-vs-osrm
2. https://stackshare.io/osrm
3. https://stackshare.io/osrm/alternatives
4. https://www.geofabrik.de/data/routing.html
5. https://neis-one.org/2011/07/comparison-routing/
6. https://stackoverflow.com/questions/63469745/reasons-for-using-osmbonuspack-versus-osrm-open-source-routing-machine
7. https://www.libhunt.com/compare-openrouteservice-vs-osrm-backend

---

Key Differences Between OSRM and OSMnx

Source: Perplexity.ai

Feature/Aspect	OSRM	OSMnx
Purpose	Designed for high-performance routing and navigation. It calculates routes, distances, travel times, and isochrones using OpenStreetMap data3.	Focuses on creating, analyzing, and visualizing street networks from OpenStreetMap data. It is more suitable for urban planning and network analysis2.
Primary Use Case	Routing services like shortest path computation between locations, distance matrices, and trip optimization3.	Graph-based operations, such as extracting street networks as graphs, analyzing their properties, and visualizing them2.
Programming Language	Primarily used via APIs (HTTP interface) or as a C++ library (libosrm). It also has bindings for languages like R3.	Python library designed for easy integration with data science workflows2.
Data Representation	Focuses on routing-specific data like waypoints, routes, travel times, and distances13.	Represents street networks as graph objects (nodes and edges) for spatial analysis2.
Performance	Optimized for speed and scalability in routing computations; often deployed as a server application for real-time use cases13.	Suitable for offline analysis and visualization; not optimized for real-time routing2.
Visualization	Limited visualization capabilities; primarily returns JSON responses for routing queries3.	Offers rich visualization tools for street networks and spatial data2.
Dependencies	Requires pre-processing of OpenStreetMap data into `.osrm` files for efficient querying3.	Directly interacts with OpenStreetMap API to fetch raw data without pre-processing2.

Summary

- **OSRM** is ideal for applications requiring fast routing computations (e.g., navigation systems).
- **OSMnx** is better suited for spatial network analysis and visualization tasks in urban planning or research contexts.

Citations:

1. https://pub.dev/documentation/osrm/latest/osrm/
2. https://github-wiki-see.page/m/Project-OSRM/osrm-backend/wiki/Library-API
3. https://cran.r-project.org/web/packages/osrm/osrm.pdf
4. https://geoffboeing.com/2018/03/osmnx-features-roundup/
5. https://www.codeproject.com/Articles/5328791/Using-OSMnx-for-Graph-Analysis-of-Street-Networks
6. https://stackshare.io/stackups/openstreetmap-vs-osrm
7. https://talks.osgeo.org/foss4g-2023-academic-track/speaker/9C8HQM/
8. https://osmnx.readthedocs.io/_/downloads/en/latest/pdf/
9. https://talks.osgeo.org/foss4g-2023/talk/P7D3SC/
10. https://talks.osgeo.org/foss4g-2023-academic-track/talk/PXY33K/
11. https://pypi.org/project/osrm-py/
12. https://help.openstreetmap.org/questions/30272/how-the-routing-osrm-algorithm-works
13. https://gis-ops.com/osrm-nodejs-bindings/
14. https://stackoverflow.com/questions/44759497/which-osm-attributes-are-used-for-routing-osrm/44769063
15. https://www.npmjs.com/package/osrm
16. https://talkpython.fm/episodes/show/495/osmnx-python-and-openstreetmap
17. https://pypi.org/project/osmnx/
18. https://geoffboeing.com/share/osmnx-paper.pdf
19. https://github.com/rajesvariparasa/spatial-routing-libraries-and-services
20. https://stackoverflow.com/tags/openstreetmap
21. https://osmnx.readthedocs.io/en/stable/user-reference.html
22. https://www.rdocumentation.org/packages/osrm/versions/3.2.0
23. https://en.wikipedia.org/wiki/Open_Source_Routing_Machine
24. https://arxiv.org/ftp/arxiv/papers/1611/1611.01890.pdf
25. https://geoffboeing.com/2016/11/osmnx-python-street-networks/
26. https://escholarship.org/content/qt4720r36p/qt4720r36p.pdf
27. https://osmnx.readthedocs.io/en/stable/getting-started.html
28. https://arxiv.org/pdf/1611.01890v3.pdf

The performance of OSRM (Open Source Routing Machine) and OSMnx differs significantly due to their design and intended use cases:

Performance Comparison

Source: Perplexity.ai

1. **Routing Efficiency**:

    - **OSRM**: Extremely efficient and optimized for large-scale routing tasks. It can handle thousands of routing requests in seconds. For example, in a study involving 32 million origin-destination pairs, OSRM processed all calculations in just 12 minutes when run on a high-performance cluster, demonstrating its suitability for high-throughput applications[1][2].

    - **OSMnx**: Less efficient for large-scale routing due to limitations in handling the size of road networks. When analyzing large datasets, such as hospital ZIP code pairs, the data had to be divided into smaller subsets, which made the process labor-intensive and slower[1][2].

2. **Scalability**:

    - **OSRM**: Designed for scalability and real-time applications. It is capable of handling extensive datasets and high request volumes efficiently, making it ideal for production environments like navigation systems or logistics platforms[1][2].

    - **OSMnx**: Better suited for smaller-scale analyses or research purposes. It struggles with very large datasets due to memory limitations and the computational overhead of its graph-based approach[1][2].

3. **Accuracy of Travel Time Estimation**:

    - **OSRM**: Produces slightly longer travel time estimates (about 10% higher) compared to services like Google Maps, as it uses specific parameters for traffic modeling but does not incorporate real-time traffic conditions[1][2].

    - **OSMnx**: Tends to underestimate travel times because it calculates routes based on the shortest distance using maximum velocity without accounting for traffic or road-specific speed limits[1][2].

4. **Setup and Preprocessing**:

    - **OSRM**: Requires preprocessing OpenStreetMap data into `.osrm` files, which takes time but enables fast query responses during runtime[1][2].

    - **OSMnx**: Directly fetches data from OpenStreetMap APIs without preprocessing but at the cost of slower computations during runtime[3].

Summary

- OSRM excels in speed, scalability, and handling large datasets, making it ideal for production-level routing tasks.

- OSMnx is more suitable for network analysis and visualization but lacks the performance needed for large-scale or real-time routing applications.

Citations:

1. https://talks.osgeo.org/foss4g-2023/talk/P7D3SC/
2. https://talks.osgeo.org/foss4g-2023-academic-track/talk/PXY33K/
3. https://osmnx.readthedocs.io/en/stable/getting-started.html
4. https://community.openstreetmap.org/t/osrm-in-a-system-container/107772
5. https://stackoverflow.com/questions/tagged/osrm?tab=Votes
6. https://stackoverflow.com/questions/tagged/osrm?tab=newest&page=5
7. https://gis.stackexchange.com/questions/430030/distance-between-two-points-on-osm-without-routing-information
8. https://stackoverflow.com/questions/63469745/reasons-for-using-osmbonuspack-versus-osrm-open-source-routing-machine

---

For handling large road networks, **hierarchical graph partitioning methods** and **optimized algorithms like those used in OSRM** are generally better suited compared to libraries like OSMnx. Here's why:

Advantages of Hierarchical Approaches (e.g., OSRM)

1. **Efficient Search Space Reduction**:
    - Hierarchical graph models, such as those described in1, use region pruning strategies to significantly reduce the search space without compromising accuracy. This approach is ideal for large urban road networks where computational efficiency is critical.
    - Partitioning techniques, as highlighted in3, divide road networks into smaller, well-connected components, minimizing computation time for routing queries.
2. **Scalability**:
    - OSRM excels in scalability, processing millions of routing requests efficiently due to its preprocessing step that structures data into `.osrm` files optimized for traversal. This makes it highly suitable for large-scale applications like continent-wide routing3.
3. **Real-Time Applications**:
    - Hierarchical path computation algorithms are designed to facilitate real-time routing by leveraging techniques such as beacon-based

partitioning and graph traversal optimization3.

Limitations of OSMnx for Large Networks

1. **Memory Constraints**:
    - OSMnx struggles with very large datasets due to memory limitations and computational overhead when extracting and analyzing street networks as graphs13.
2. **Slower Computation**:
    - While OSMnx is excellent for visualization and analysis, its graph-based approach is less efficient for large-scale routing tasks compared to hierarchical models optimized for speed13.

Conclusion

For handling large road networks, hierarchical methods like those implemented in OSRM are better suited due to their efficiency, scalability, and ability to support real-time applications. OSMnx is more appropriate for smaller-scale network analysis and visualization tasks.

Citations:

1. https://pmc.ncbi.nlm.nih.gov/articles/PMC5812589/
2. https://www.mdpi.com/2220-9964/6/12/400
3. https://research.google/blog/efficient-partitioning-of-road-networks/
4. https://pmc.ncbi.nlm.nih.gov/articles/PMC8887733/
5. https://www.nature.org/content/dam/tnc/nature/en/documents/latin-america/Friendlyroads.pdf
6. https://www.mckinsey.com/industries/travel-logistics-and-infrastructure/our-insights/finding-the-right-of-way-to-efficient-road-operation-and-maintenance
7. https://pmc.ncbi.nlm.nih.gov/articles/PMC11002554/
8. https://pmc.ncbi.nlm.nih.gov/articles/PMC5666254/

Cost Comparison Between OSRM and OSMnx

1. **OSRM**:
    - **Open Source**: OSRM is free to use as an open-source tool, but costs arise when hosting it in production environments.

- **Hosting Costs**: If self-hosted, OSRM requires infrastructure like cloud servers (e.g., AWS). Typical costs include:
  - **Setup Costs**: One-time expenses for computing and storage (e.g., EBS, S3).
  - **Monthly Maintenance**: $50–$200 depending on server size, storage needs, and data update frequency1.
- **Usage-Based Costs**: For hosted solutions, OSRM pricing depends on API request volumes:
  - Free tier: 2,500 requests/month.
  - Paid: $0.50 per 1,000 requests (up to 100K); $0.20 per 1,000 requests beyond 100K1.
- **Additional Efforts**: Regular updates of OpenStreetMap data are necessary for accurate routing.

2. **OSMnx**:
   - **Completely Free**: OSMnx is open-source and does not involve hosting or API usage costs.
   - **No Hosting Required**: It operates locally, fetching data directly from OpenStreetMap APIs without needing preprocessed files or external servers5.
   - **Indirect Costs**:
     - Computational overhead for large datasets may require powerful local hardware.
     - Time and labor costs for dividing large datasets into manageable subsets for analysis3.

Summary

- **OSRM** may incur hosting and usage costs if deployed in production environments but offers high efficiency for routing tasks.
- **OSMnx** is entirely free but less suitable for large-scale or real-time applications due to its computational limitations.

Citations:

1. https://www.c-sharpcorner.com/article/understanding-osrm-routing-with-aws/
2. https://arxiv.org/ftp/arxiv/papers/1611/1611.01890.pdf
3. https://talks.osgeo.org/foss4g-2023/talk/P7D3SC/
4. https://gis.stackexchange.com/questions/139630/what-are-the-osm-class-cost-settings-for-osrm
5. https://osmnx.readthedocs.io/en/stable/getting-started.html
6. https://stackshare.io/stackups/mapbox-vs-osrm
7. https://osmnx.readthedocs.io/_/downloads/en/latest/pdf/
8. https://stackshare.io/stackups/openstreetmap-vs-osrm
9. https://help.openstreetmap.org/questions/23715/how-to-use-osrm
10. https://www.cnet.com/tech/tech-industry/start-up-launches-linux-legal-protection/
