Enhancing the Airbnb Search Experience: Data Analysis, Sentiment Analysis, and Recommendation Optimization

34572
London School of Economics and Political Science
London, United Kingdom

30993
London School of Economics and Political Science
London, United Kingdom

23154
London School of Economics and Political Science
London, United Kingdom

33308
London School of Economics and Political Science
London, United Kingdom

ABSTRACT

In Airbnb, a digital platform connecting accommodations and experiences, guests frequently invest weeks in exploring and comparing various options before finalizing a reservation. The extensive and investigative process of searching, along with the requirement to consider both guest and host preferences, poses distinct challenges for Airbnb's search ranking mechanism [15]. In this research we aim to develop a recommendation system for Airbnb listings based on historical user reviews in major cities, focusing on location and distance parameters. By incorporating user feedback and geographical data, our goal is to create a user-friendly system where users specify a location on a map and a radius, and the system recommends the top 10 Airbnb listings within that specified distance. By using this strategy, we aim to improve the user experience by offering accommodations that are relevant to their needs and preferences for closeness.

1 INTRODUCTION

The advent of online platforms has revolutionized the tourism and hospitality industry, offering travelers a wide array of accommodation options. Among these platforms, Airbnb has emerged as a prominent player, providing users with unique lodging experiences in various cities across the globe. At Airbnb, search is the main way for a guest to discover the right inventory, such as stays, (in-real-life) experiences (i.e. tourism places), online (i.e. virtual) experiences, etc. In this research paper, we delve into the intricate workings of Airbnb data analysis and recommendation systems, focusing on twelve major cities: Paris, London, Toronto, Rome, Amsterdam, Barcelona, Bangkok, Berlin, Bristol, Brussels, Cambridge, and Edinburgh.

Deep learning techniques have demonstrated effectiveness across various domains. Chauhan and Palivela [3] utilized LSTM and Glove word embedding for fake news detection on social media, effectively filtering content. Ensafi et al. [6] proposed LSTM for forecasting seasonal item sales, comparing its efficiency with other methods. Ji et al. (2021) introduced the STARec model for user preference modeling, highlighting its advantages such as the activity frequency feature but noting drawbacks like abnormal training time. Suresh et al. [16] explored future visitor recommendations alongside POI recommendations using KNN and RF. Chen et al. [4] suggested a POI recommendation approach based on user location interest and contextual data, with limitations in handling numerous features. Çakmak et al. [2] analyzed visitation profiles to recommend POIs using link prediction methods. In a reinforcement learning attempt by Massimo and Ricci [12] an IRL-based model was proposed, successfully addressing clustering overheads but struggling with capturing behavioral patterns in large datasets.

Our approach to enhancing the search experience for Airbnb guests is comprehensive, integrating several key components to optimize recommendation systems and facilitate informed decision-making. Central to our methodology is the automation of crawling and pre-processing Airbnb data from InsideAirbnb source, utilizing distributed computing techniques such as Apache Spark DataFrame to handle large-scale datasets efficiently. This curated dataset, comprising millions of observations from diverse cities, serves as the foundation for insightful analysis and recommendation. In tandem with automated data processing, sentiment analysis plays a pivotal role in our methodology, enabling us to understand the sentiment of user reviews towards Airbnb listings. By leveraging tools like NLTK SentimentIntensityAnalyzer, we gain valuable insights into user preferences and satisfaction levels, informing our recommendation strategies. Moreover, our methodology incorporates interactive visualization methods to present analyzed data in a user-friendly manner. Through the development of intuitive interfaces using ipyleaflet and ipywidgets, users can explore Airbnb listings based on their preferences and geographic locations, enhancing user engagement and facilitating informed decision-making. Lastly, optimization strategies are deployed to streamline the recommendation process and enhance search efficiency. Techniques such as geohashing and binary search optimize search algorithms, ensuring a seamless and satisfying user experience, and ultimately driving increased guest conversion rates on Airbnb. Furthermore, we noticed Airbnb users often struggle to plan their trips effectively due to a lack of information on nearby tourist attractions and their distance when choosing to stay on Airbnb. To address such issue, we propose integrating a "Travel Score" into Airbnb's recommendation system. This feature combines clustering algorithms to geographically categorize listings near tourist sites to assess the distance of these locations. By providing a quantifiable Travel Score, our solution aims to transform Airbnb into a more comprehensive travel planning tool, ensuring accommodations align with travelers' sightseeing preferences. Overall, our recommendation system enhances Airbnb's functionality by incorporating a two-dimensional evaluation approach, which includes both a sentiment score and a Travel Score for each stay.

2 METHODOLOGY

2.1 Data

The first part utilises data from Airbnb listings and reviews in Paris, London, Toronto, Rome, Amsterdam, Barcelona, Bangkok, Berlin, Bristol, Brussels, Cambridge, and Edinburgh. After filtering missing values, the merged data set of the 12 cities consists of 8058086 observations with ten features each, taking up 6.6 GB. Furthermore, the data set has diverse data types, including integer, float, and string. Therefore, our data set matches the characteristics of big data, with a volume of 6.6 GB requiring distributed computing. Furthermore, our data ingestion is comprehensive, allowing us to add more cities with a run of a script, and the more cities we have, the more we need distributed computing.

The second part is to accommodate the geo-location requirements of our recommendation system, it's imperative to utilize a comprehensive geolocation dataset. Among the limited options available only OpenStreetMap meets our criteria by providing detailed tourism tags with geographic data. Consequently, we selected OpenStreetMap's osm.pbf files, which can be downloaded from Geofabrik. OpenStreetMap offers data in shp and osm.pbf formats, these formats are not inherently compatible with Apache Spark which present significant challenges. For the twelve cities, the geospatial data has 191259821 rows in total while containing a nested array, necessitating the need for Parquet format and substantial preprocessing.

2.2 Distributed computing methods

In the first part, we use Spark SQL DataFrame API and its operations to view and transform the raw data. The first kind of operations are for dataframe-level transformations: `select` and `withColumn`. The second kind is for column-level transformation and returns column expressions: `alias` and `lit`. The third kind is for set-like transformations: `union`. The fourth kind is for joining between dataframes: `join`. The last is for grouping transformations: `groupBy` and `agg`. The detailed description of these operations is listed in Table 1.

In the second part, initial attempts to load shp and osm.pbf files locally using the Pyrosm parser and GeoDataFrame were unsuccessful, often resulting in memory overflows or extended load times, especially with files larger than 100MB (as their actual size can be 5 times larger). To overcome these limitations, we turned to Google Cloud Platform (GCP) and Apache Spark. Despite several attempts using Apache Sedona(as declared 2-10x faster) and OSM4Scala, compatibility issues with Spark versions (2.4, 3.3, and 3.5) and Scala 2.12 on GCP led to unresolved problems related to shaded jars and the integration of Java with Scala. Alternative methods, such as Fiona and GeoDataFrame for shp files, were able to handle larger files but still encountered overflow issues on GCP.

Moreover, we adopted a conversion method that transforms osm files into Parquet format, as detailed in Adrian Ulbona's guide [1]. This approach significantly enhances osm.pbf file's compatibility with Spark and HDFS, facilitating faster data loading and more efficient processing. Moreover, PySpark DataFrames enabled us to handle large volumes of geospatial information with significantly reduced memory consumption.

Table 1: Spark SQL DataFrame Operations

Operation	Description
`select(*cols)`	Returns a new DataFrame using a series of expressions that could be column names or `Column` objects
`withColumn(colName, col)`	Returns a new DataFrame by adding a new column or replacing an existing column with the given name
`alias (xalias)`	Returns a column with a new name
`lit(value)`	Creates a column with value provided
`union (df)`	Returns the union of elements of two DataFrames (keep duplicates)
`join (other, on, how)`	Joins two DataFrames
`groupBy(*cols)`	Groups the DataFrame using the specified columns for running aggregation on them

`agg(*exprs)`
 	
`...etc`
	Returns the aggregated DataFrame by the given expressions
	...etc

Interestingly, different usage of partitionBy and Repartition methods also impact the performance of processing of big data and the clustering algorithms such as Sklearn DBSCAN(since PySpark doesn't provide DBSCAN) and PySpark KMeans, particularly in distributed computing environments. This is primarily due to the challenges associated with calculating distances when coordinates are distributed across different partitions, complicating the process of performing nested iterations across them. We also compare the performance difference between Sklearn KMeans and PySpark-ml KMeans in Section 4 NUMERICAL EVALUATION.

To enhance performance, we strategically aimed to method that writes close coordinates data within the same partition. Our initial approach was to use the partitionBy("city") and repartition method when writing tourism spot data to HDFS in 'geodata_PreProcessing'. This method groups data based on city names, thereby reducing the distance calculations required between points by ensuring that data points within the same geographic locale are likely to be stored together. This approach not only optimizes the execution efficiency of distance-based clustering algorithms but also minimizes the overhead associated with data shuffling and inter-partition communication. We proved such a method could improve processing speed in Section 4 NUMERICAL EVALUATION. Our next step could be to discover how to stop calculating far distance coordinates from cross partition/worker-node that will further improve the performance[14].

2.3 Algorithms

2.3.1 Machine Learning. The NLTK (Natural Language Toolkit) `SentimentIntensityAnalyzer` is a tool for sentiment analysis [13], which is the process of identifying and quantifying the sentiment or emotional tone expressed in each text. It uses a pre-trained sentiment lexicon, a dictionary that associates words with their respective sentiment scores. When a piece of text is passed to the

method `polarity_scores(text)`, it breaks the text down into individual words, looks up their sentiment scores in the lexicon, and returns a float for sentiment strength ranging from -1 to 1 based on the input text. Negative values indicate negative sentiment (the more negative, the more intense). Positive values indicate positive sentiment (the more positive, the more intense). A score of 0 indicates neutral sentiment. We apply this tool to every comment to generate the sentiment score of every user review to the Airbnb host.

On the other hand, the Travel Score is the distance between each Airbnb listing and many tourism spots, if we directly do the calculation from the Airbnb dataset and geospatial dataset, the calculation size will be 5302613 x 191259821. Therefore, to reduce the size, we employed both DBSCAN and KMeans clustering algorithms. These algorithms are well-suited for managing geospatial data, given their robustness in large-scale distance calculation. The foundational concept of our method is to find the center of clusters of tourist spots, and calculate its distance with Airbnb listing instead of tourism spots. Listings closer to tourism spots cluster centres are deemed higher Travel Scores due to reduced travel distance to multiple spots. Initially, each algorithm was to be deployed independently; however, inspired by insights from Kremers [9], we opted to integrate both methods to three layers. We first applied PySpark KMeans to get initial clusters in large-scale data, but the tourism spots within each cluster are spread unevenly with noise. Then we configured DBSCAN to each cluster to find tourism spots that are gathered in a radius of 3000 meters and a minimum of 3 spots, simulating the average number of tourist spots a traveler might visit from a single location by walk, aligning with typical tourist behaviors. DBSCAN excels in identifying clusters with noisy, irregular clustering shapes—a common scenario in geospatial data representing unevenly distributed spots, which often do not conform to standard geometrical patterns. Thus, any location that doesn't match is an outlier or noise as they are too far away or too few to be representative. The practical application and specific settings can be found in the file geodata_KMeans$_D$*BSCAN*.

Once clusters are identified using DBSCAN, they are processed further with Sklearn KMeans to find the new center within the new DBSCAN cluster. We use the Sklearn package because it processes small data faster, please check section 4 for more details. Use the figure below as an example, this cluster is from Paris, with one layer KMeans the center is at the red dot, while after three layers the centre is at the green dot, closer to the more meaningful tourism spot cluster.

2.3.2 Optimisation. We use a concept called geohash [10] to optimise our search process based on distance. The concept maps longitude and latitude into a string format that indicates the distance range between two locations according to the first several characters. For example, if two locations have the same first eight characters, they are less than 0.019 km from each other; if they have the same first seven characters, they are less than 0.076 km from each other. Using this, we can optimise our process of finding all Airbnb that lies within the distance from the pinned location.

To fully utilising geohash, we employ an algorithm called binary search [7]. This algorithm works by repeatedly comparing the geohash of the pinned location to the middle element of the sorted

Figure 1: Cluster Example

geohash column of the dataframe to find the first and last row where geohash matches our desired pinned geohash's prefixes. For example, to find the first row where the prefix of geohash matches the prefix of the pinned geohash, we compare the prefix of geohash at pinned location and geohash at the middle point of the dataframe. We know that if the geohash at middle point is greater than the prefix of geohash at the pinned location, then the first row where prefixes match the geohash at the pinned location has to be after the middle point. Otherwise, we know that the first row must be before the middle point. Thus, we are able to reduce the search range by half. From there we repeat this process repeatedly until the target value is found. Binary search has a time complexity of O(logn), significantly faster than linear search, making it an excellent search strategy in large datasets to locate the first and last rows that matches our needs.

2.4 Visualisation methods

First, we plot geospatial heatmaps, to visually represent the density and distribution of listings within each city. Using latitude and longitude coordinates, we generate heatmaps that depict the density of listings within specific regions. The geospatial heatmaps offer valuable insights into the spatial distribution of Airbnb listings across the twelve cities. Understanding these patterns can inform various stakeholders, including policymakers, real estate investors, and Airbnb hosts, about market dynamics and potential areas for intervention or investment. Additionally, the visualizations serve as a foundation for further analysis, such as identifying correlations between listing density and factors like tourism hotspots, transportation infrastructure, or socioeconomic indicators.

For the UI, we create an interactive map using `ipyleaflet` and `ipywidgets`, which display a map of the world, along with a pin on the map and a circle surrounding that map. The pin would be our centre position and is initialised at the mean coordinates of the whole dataframe. The circle would be where we want to look and recommend top-ranking Airbnb listings. There is also a slider for the radius, which allows us to alter the search distance. Finally, the centre pin is entirely drag and movable, and when we move it to a new location, we suggest Airbnb listings within the radius of that new location. From there, the top 10 Airbnb listings within that radius will be shown along with their information, URL, and rating.

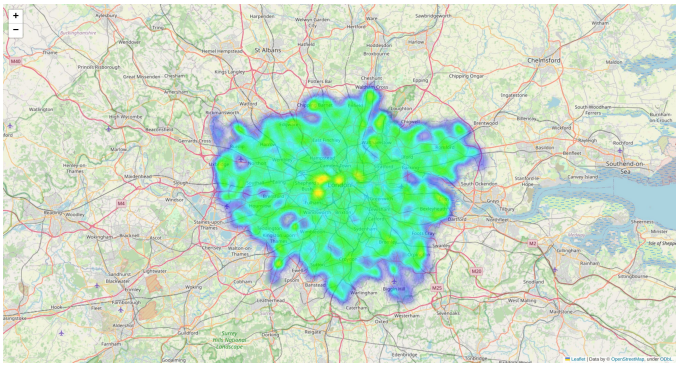

Figure 2: Geospatial Heathmap for London

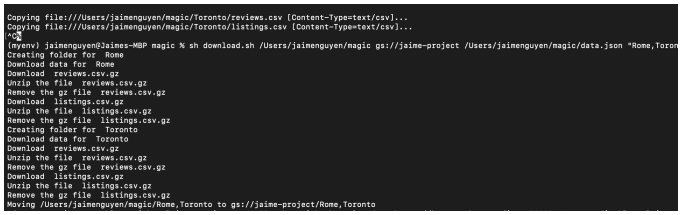

Figure 3: Terminal Expression

3 IMPLEMENTATION

The project is divided into two parts. In the first part, we construct the fundamental recommendation system that would suggest the top 10 Airbnb listings within the specified radius of a given location. We divided this part into three main stages: data crawling automation, data pre-processing and modelling, and visualisation.

In the second part, we aim to give every Airbnb listing a Travel Score that indicates how close is the listing's location to each tourism cluster using geospatial data. This part is fully on HDFS and Spark. We divided this part into Four main stages: data downloading transformation (geodata_download), data pre-processing (geodata_Preprocessing), clustering (geodata_KMeans_DBSCAN), and scoring (geodata_travel_score_recommendation). The travel scoring data will be merged with sentiment score data during visualisation stage.

Overall, we mainly utilized Pyspark SparkSQL and Dataframe API, Pyspark Pandas API, NLTK, and Pyspark ML API, etc. on the project.

3.1 Data Crawling Automation

First, we use data from https://insideairbnb.com/get-the-data/, which provides comprehensive Airbnb user reviews and listing data for most major cities worldwide. For each city, we care about two datasets: listings and reviews. The former contains details about the city's Airbnb properties, while the latter contain user reviews. We intend to automate the whole process of data ingestion, from crawling data directly from the website to unzipping and uploading the data into the city folder in Google Cloud Bucket. To ease the process of crawling data from the website, we create a JSON file, which consists of the key-value pairs of a city and the URL of the data of that city on the website. If we want to get data from another city, we only need to find the link to that city's data and add it to the JSON file. Then, we create a Python script, download.py that looks into the URL associated with the cities in the JSON file and download them to a specified folder. The command to run the script is like:

```
python3 download.py
 --output_folder /path/to/target/folder
 --json_path /path/to/data.json
 --cities London,Toronto,(other cities)
```

It creates, for each city, a folder in the output_folder, downloads the listings and reviews files, unzips them, and removes the zipped files, leaving only the unzipped files. Furthermore, we utilize a bash script, Download.sh to fully automate the process from crawling data to upload them to Google Cloud Bucket. This bash script is first called the above Python script and then goes through all the created folders and uploads them to the Google Cloud Bucket. A sample command to run the script is:

```
sh download.sh /path/to/a/local/folder
 gs://bucket_name /path/to/data.json
 "Rome,Toronto,(other cities)"
```

It downloads the specified cities' data to a designated local folder to store the data and uploads it to the specified bucket. With these files, we only need to add the city and its URL to the JSON file to include new data, making the data crawling process automated and modular.

3.2 Data Pre-processing and Modelling

We perform data pre-processing and NLP modelling on the Google Cloud Platform. We first create a function `find_city` to find all the cities of interest in the bucket and put them in a list. Later, we create a function called `ingest_data` to pre-process the data. Using the above list of cities, we read the datasets for each city and add a column combining the city's name and the Airbnb property's ID to every `pyspark.sql.DataFrame`. Then we select only the columns of our interest: for listings, we select "id", "price", "longitude", "latitude", "name", "listing_url", "review_scores_rating"; for reviews, we select "listing_id", "reviewer_id", "comments". Afterward, we merge them on the Airbnb property's ID ("id" for listings and "listing_id" for reviews) and drop missing values. We further yield the union of different cities' dataframes, convert the "price" column from string to float and drop those where the conversion fails.

Next, in order to modelling the dataset, we removed any entries where the comments column was either null or empty. This is important to ensuring that only valid text data was processed in subsequent stages. We excluded entries that contained comments indicating a cancellation by the host. This helps focus our analysis on guest reviews rather than administrative records.Furthermore, we standardize the text data and improve the analysis's accuracy, we applied a text normalization function that removes numerical characters from the comments. This normalization was implemented through a user-defined function (UDF) in Spark. These pre-processing steps are integral to our analytical pipeline, ensuring that the data is primed for the subsequent sentiment analysis while eliminating potential sources of noise and irrelevance.

listing_id	reviewer_id	comments	id	price	longitude	latitude	name	listing_url	review_scores_rating
paris3109	51636494	Tout s'est bien d...	paris3109	$150.00	2.3187	48.83191	Rental unit in Pa...	https://www.airbn...	5.0
paris3109	4142888	Un petit nid foui...	paris3109	$150.00	2.3187	48.83191	Rental unit in Pa...	https://www.airbn...	5.0
paris3109	7415343	Appartement spaci...	paris3109	$150.00	2.3187	48.83191	Rental unit in Pa...	https://www.airbn...	5.0
paris3109	21159216	Appartement total...	paris3109	$150.00	2.3187	48.83191	Rental unit in Pa...	https://www.airbn...	5.0
paris5396	19995	Perfect location!...	paris5396	$146.00	2.35835	48.85247	Rental unit in Pa...	https://www.airbn...	4.59

Figure 4: First 5 rows of Paris data

To have a sentimental score for each review, we use the NKTL library, specifically `SentimentIntensityAnalyzer` and its method `polarity_scores(text)` and obtain the polarity score for each review. With the score, we group all the datasets based on the `listing_id`. For each `listing_id`, we care about the property's price, average sentiment score, latitude, longitude, name, URL, and review score rating. Then, we calculate the RMSE for each property based on normalised review scores rating and sentiment scores.

The primary challenge in managing geospatial data involves the complexity of the data sources, particularly with formats such as OSM.PBF and SHP files. As outlined in the previous sections, these files are cumbersome to process due to their size and structure. A significant hurdle during data extraction is the immutable nature of data structures in Apache Spark, notably DataFrames which, akin to RDDs, cannot be altered once created. To address these challenges, we focused meticulously on the schema and data types used within our DataFrames. We experimented with various data manipulations to refine our approach gradually. By filtering out unnecessary columns and rows and utilizing groupBy(), array(), agg(), explode(), and when() from pyspark.sql functions, we significantly enhanced dataframe structure, and successfully extracted 8435 rows of data out of 191259821 rows of data. Additionally, when implementing KMeans, DBSCAN, and Travel Scores, the user-defined functions (UDFs) combined with PySpark Pandas have substantially increased the scalability and flexibility of our implementation, especially on handling nested array, enabling more sophisticated analyses(labeling, filtering, calculating) and transformations of data within our distributed computing framework.

Realizing KMeans and DBSCAN not only aids in providing targeted recommendations but also significantly reduces the computational load for the final Travel Score. For instance, the original dataset has 55,382 tourism spots. Post-application of DBSCAN and KMeans, from the figure we can see the spots for further computation was reduced to 143(green sphere), effectively decreasing the distance calculations between Airbnb locations and tourist spots by nearly 500 times.

For Travel Score, we rate each Airbnb listing by the number of tourism spots cluster center within a 1.5km radius, starting from 1 star and ceiling at 5 stars, each cluster centre will add 1 star to the score. After we count the number of listings in each star, it does matches the normal distribution on quality of location. In the figure below, different scored Airbnb's were colored in different colors around the cluster centre, there are some blank spaces because the different city boundaries were used in Airbnb and geodata datasets.

3.3 Data Visualisation

For the visualisation, we utilize local pandas since they are better at visualizing and showing a better and more interactive UI.

First, we read and pre-process the data with pandas. To optimise our search process based on distance, we employ geohash and sort

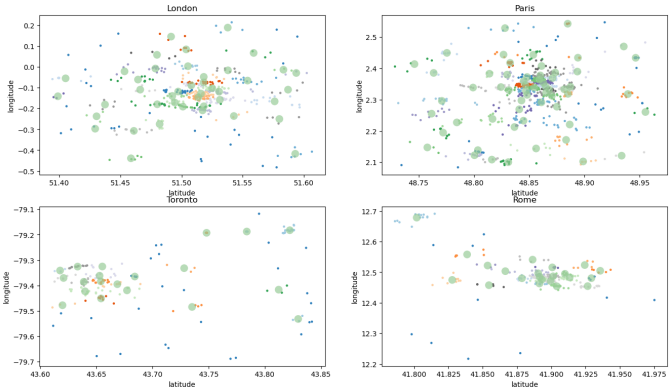

Figure 5: KMeans+DBSCAN Cluster Center(green sphere) vs. Cluster Tourism Spots(colored dots)

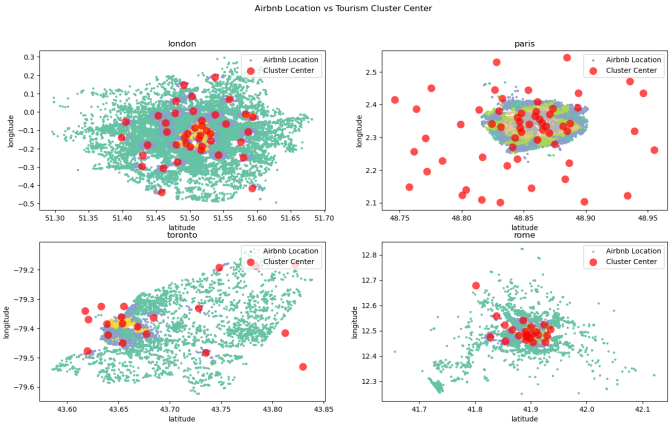

Figure 6: Airbnb Location(colored dots) vs. Tourism Cluster Center(red sphere)

the whole dataset based on it. Then, we create an interactive map UI using `ipyleaflet` and `ipywidgets` whose centre and radius to search are adjustable. Finally, We search for the top 10 Airbnb properties for a given location and present them to the user.

Here, we introduce our search strategy using binary search. Suppose the user has chosen a location and a radius distance (km) from that location; we are required to filter the Airbnb properties that are within the distance from the location by a function `find_start_end_prefix`, which utilises the location's geohash prefix and binary search to optimise search time.

To illustrate this, assuming that the radius is 20 km, we only care about the properties' locations' "geohash" having the same first four characters as the pinned location. First, we create a list containing all geohash values in the dataframe and the pinned location's prefix (the first four geohash characters). Second, we put a midpoint as the middle value of that list and compare the geohash of the middle value and the prefix. If the middle value is larger or equal, we know the first point with four same geohash characters lies within the index 0 to *mid* of the list. On the other hand, if the middle value is smaller, we know that the first point lies

between index *mid*+1 and the final position of the list. By doing this repeatedly, we reduce the search range by half each iteration until we get 1 location, which would be the first location in the list that has the same four geohash characters as the prefix. The same logic goes to finding the end location in the list that has the same four geohash characters as the prefix. Having established the locations of our interest, we sort them by their sentiment scores, recommend the top 10 Airbnb listings, and display them interactively on the map.

Using Pandas DataFrame to plot data directly in PySpark proved inefficient due to performance issues. To address this, we opted for plotting subsets of data, reducing complexity by randomly removing some Airbnb listings. This approach allowed us to maintain visual clarity without sacrificing representativeness. In our analysis, we applied DBSCAN and KMeans to identify and visualize cluster centres among tourism spots. These centres are marked as green circles on the map, providing a clear visual representation of where each cluster's central point is located relative to all tourism spots. In the visualization titled "Airbnb Location vs. Tourism Cluster Center," the proximity of Airbnb listings to these cluster centres visibly correlates with their Travel Scores, highlighting the practical utility of our clustering approach. Additionally, the "Full Tourism Spot vs. Tourism Cluster Center" map effectively showcases the synergy between DBSCAN and KMeans. Notably, this combination captures smaller, potentially significant clusters, such as those representing suburban leisure spots like theme parks, which could be overlooked or treated as noise by KMeans alone. This demonstrates the enhanced capability of our combined approach to discern relevant clusters in geospatial data, offering a nuanced insight into the distribution of tourist attractions.

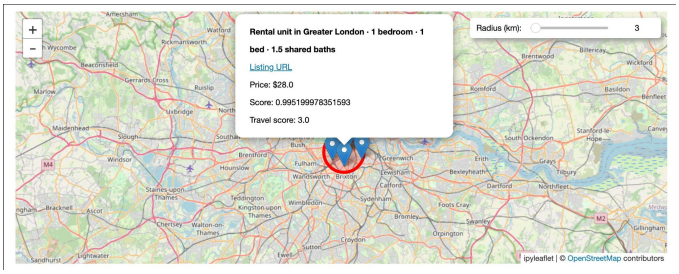

Figure 7: Example of a location in London

4 NUMERICAL EVALUATION

This section evaluates the performance of Apache Spark and Pandas DataFrame in handling data ingestion, and sentiment analysis using the NLTK library.

The dataset consists of listing and reviews of 12 big cities which are London, Paris, Toronto, Rome, Amsterdam, Barcelona, Bangkok, Berlin, Bristol, Brussels, Cambridge, and Edinburgh totaling approximately 6.6GB in csv format. We use a cluster on Google Cloud with 1 master node and 2 worker nodes. The master node uses a n2-standard-2 type with a 200 GB disk size while the 2 worker nodes use a n2-standard-2 and 150GB disk size. For our local machine for comparison between distributed and non-distributed computing, we use a laptop with Ubuntu 22.04, using the processor 11th Gen Intel® Core™ i7-11800H and 16GB RAM.

Along with our implementation of the processing data using Spark on the GCP, we implement a similar pre-processing using pandas dataframe that runs locally on our laptop and time it to compare them together. Firstly, the execution time for Spark and pandas dataframe is similar for preprocessing and cleaning the data (approximately 6 seconds for each time to ingest). Secondly, we evaluate the performance of the sentiment analysis task using NLTK executed directly and compare it with pandas DataFrame. It can be seen from the table that direct NLTK processing was significantly faster than processing through pandas dataframe, which took around 60 minutes on our local machine. Finally, we notice that the computation for `groupBy` function is significantly better for Spark than for Pandas, taking 53 and 90 minutes respectively. The significant difference in the NLTK analysis run times between Spark and Pandas can be attributed to Spark's lazy evaluation mechanism. In Spark, operations are not executed immediately; instead, they are optimized and executed only when an action that requires a result is called. This implies that the NLTK analysis might not have been fully executed during the initial stage. However, when a groupBy operation is performed, Spark is forced to execute all preceding transformations, including the sentiment analysis, to aggregate the data. This delayed execution explains why the groupBy operation takes considerably longer in Spark compared to Pandas, where computations are executed immediately. We can conclude that the running time of groupby (53mins) includes both all NLTK and group by operation

Finally, we also use the actual rating for these listings from AirBnB sites to compare them with the polarity score produced by NLTK. By comparing them, we achieve an RMSE of 0.454666.

Task	Spark	Pandas
Preprocessing	1min14s	70s
NLTK Analysis	14s	60m
Group By	53m	90m

Table 2: Performance: Spark vs. Pandas

For handling geospatial data formats such as SHP and OSM.PBF files present significant challenges, particularly on systems with limited memory capacity. For example, using M1 MacBook with 8GB of RAM as a test case, it took between 3 to 6 minutes to load just a 100MB London OSM file using Pyrosm, the fastest parser available. Moreover, attempting to load files larger than 100MB or performing complex data manipulations after loading led to memory overflows. Loading SHP files was even more problematic; my laptop was unable to load basic building data locally using GeoDataFrame. The conversion of the 1.2GB OSM.PBF files into other formats, such as Parquet and SHP, results in a substantial data expansion to nearly 5GB. Utilizing the n2-standard-2 machine on the Google Cloud Platform (GCP), we efficiently processed the Greater London Area data, with its Parquet file sized at 300MB. The system completed loading and preprocessing in under one minute as documented in the "geodata_Preprocessing" file. However, it struggled with more complex manipulations and was occasionally overwhelmed when loading larger files.

We also faced significant limitations when attempting to process the combined geospatial data for all the cities. The n2-standard-2 machine proved insufficient for a larger task, so we upgraded the master node to the n2-standard-4 machine. Additionally, the MacBook under use lacked the capacity to even load them, complicating efforts to demonstrate the comparative effectiveness of integrating DBSCAN with KMeans versus deploying them separately. Nevertheless, when running files on HDFS, we observed more than a threefold improvement in both file size management and processing times, highlighting the advantages of distributed computing environments for handling extensive geospatial data.

Furthermore, in assessing the impact of data partitioning on processing performance, we analyzed execution times under different scenarios. Our initial setup involved reading data from a Parquet file pre-partitioned by city, where location data were closer (referred to as "Pre-partitioned based on city" in our figures). We then created a copy by repartitioning this data into a similar number of partitions but distributed more evenly across all data (tagged as "Evenly partitioned" in our visualizations). Our findings indicate that the pre-partitioned dataset, with geographically proximate locations, processed faster than the more evenly distributed one. This performance discrepancy can be attributed to more efficient task scheduling and reduced network overhead in the pre-partitioned setup, as discussed in related literature[11].

Conversely, in a smaller scale where a single partition could accommodate all rows, the dynamics shifted. We experimented with one-partition dataframes, system auto-partitioned dataframes, and partition-by-city dataframes within a PySpark KMeans job. Here, the partition by city one still outperformed system auto-partitioned one, likely due to less data skewness (partition skewness of 483:1636) and geographically proximate locations. However, a single-partition setup proved to be the fastest, underscoring the reduced need for inter-partition communication (network I/O) at smaller scales[8]. Combining with large-scale test, we concluded partition based on city utilizing data location proximity does improve the performance of processing distance calculation.

Additionally, our evaluations extended to comparing different libraries. At smaller scales, Scikit-Learn showed superior performance, consistent with findings from Clement's research, which suggests that "PySpark becomes more efficient than Scikit-Learn once the data reaches the 12GB threshold"[5].

This comprehensive analysis helps elucidate the trade-offs between different partitioning strategies and their scalability implications, offering valuable insights for optimizing big data processing tasks in distributed environments.

Partition method/Data scale	Pre-partitioned base on city: 31 partitions	Evenly partitioned: 31~47 partitions	One partition: 2119	System auto partition: 133:1986	Partition by city: 1636:483
Large Scale: UDF	4.78 ms	10.7 ms	-	-	-
Small Scale: PySpark KMeans	-	-	11 ms	14.7 ms	12.9 ms
Small Scale: Sklearn KMeans	-	-	2.78ms	2.82ms	2.82 ms

Figure 8: Time Executive By Partition

5 CONCLUSION

The research provides valuable insights into optimizing Airbnb's recommendation systems through rigorous data analysis and sentiment evaluation. The findings reveal the efficacy of utilizing Apache Spark and Pandas DataFrame for data preprocessing, with Spark demonstrating slightly superior performance. Moreover, the implementation of sentiment analysis using the NLTK library showcased a significant advantage in efficiency when executed directly compared to processing through Pandas DataFrame. Notably, our implementation of sentiment analysis using the NLTK library revealed a substantial advantage in efficiency when executed directly, highlighting the importance of streamlined data processing pipelines. Furthermore, our evaluation of computation efficiency for the groupBy function elucidated Spark's superiority over Pandas DataFrame, emphasizing its scalability for handling large-scale data operations.

Additionally, the comparison between actual ratings from Airbnb sites and polarity scores produced by NLTK yielded a commendable Root Mean Squared Error (RMSE) of 0.45, reflecting a satisfactory level of accuracy in sentiment analysis. These results highlight the robustness of our methodology in enhancing the search experience for Airbnb guests, thereby driving increased guest conversion rates and improving overall user satisfaction.

Looking ahead, future research endeavors could delve deeper into exploring advanced deep learning techniques for sentiment analysis, such as leveraging transformer-based models like BERT or GPT, to enhance the accuracy and granularity of sentiment assessment. Additionally, investigating novel optimization strategies, such as reinforcement learning algorithms, could further refine recommendation systems by dynamically adapting to user preferences and market trends. Moreover, conducting user studies to evaluate the impact of incorporating sentiment and travel scores on user decision-making and satisfaction would provide valuable insights into user behavior and preferences. Qualitative research methodologies, including interviews and focus groups, could uncover nuanced aspects of user experiences and help tailor recommendation systems to better align with user expectations. Furthermore, extending the study to include more cities and diverse datasets would not only broaden the geographical scope of analysis but also enable the examination of different accommodation types and user demographics. This expansion would facilitate the generalization of findings across varied contexts, enhancing the robustness and applicability of our insights. Additionally, incorporating data from emerging tourism destinations and alternative lodging platforms could offer fresh perspectives on the evolving landscape of travel accommodation. Through these, researchers can deepen the understanding of Airbnb's functionality as a comprehensive travel planning tool and contribute to the ongoing advancement of recommendation systems in the tourism and hospitality industry.

6 STATEMENT ABOUT INDIVIDUAL CONTRIBUTIONS

34572 and **30993** contribute mostly on the technical part of this project. **34572** has built an automation of data crawling and data processing from insideAirbnb and upload it automatically on Google Cloud. She designed and implemented the find_city function to

identify cities of interest within the dataset stored in the Google Cloud Platform (GCP) bucket. She integrated the NLTK library's SentimentIntensityAnalyzer to calculate sentiment scores for each review and aggregated the data based on listing IDs, calculating the Root Mean Square Error (RMSE) for properties. She also in charged of reprocessing of data using Pandas for optimal visualization, including employing geohashing and sorting the dataset based on it to facilitate efficient searching based on distance.

30993 has contributed the whole process of applying geospatial data for building Travel Score, from investigating datasets to transforming geospatial data format to developing. Also **30993** has developed whole geodata preprocessing, clustering, and scoring on the Spark and HDFS. During this processing **30993** also developed and researched the DBSCAN and KMeans with PySpark and their performance measurement. In this paper **30993** mainly contributes content writing about Travel Score, geospatial data, DBSCAN, Kmeans, partition, numeric evaluation, and PySpark/HDFS distributed computing mythology.

33308 worked on the Data Visualisation, and plotted the geospatial heatmaps for all 12 cities. She also ran the main Data Visualisation code locally and debugged it. **33308** also assisted **34572** with his Methodology and Implementation of the geospatial data for building traveling score, DBSCAN, Kmeans, partition, and numeric evaluation. She also worked on the Abstract, Introduction, and Conclusion of the report and the plots. Furthermore, **33308** reviewed the paper for final submission.

23154 contributes to checking **34572**'s codes and assisting them in running the codes and exporting the datasets for local visualisation. **23154** also assists **30993** select the datasets to use and perform data cleaning. Besides, **23154** is in charge of constructing the whole structure of the report as well as summarising the Methodology and Implementation of **34572**'s work. Furthermore, **23154** drafted the report by incorporating different parts, crafted the tables and codes.

7 STATEMENT ABOUT THE USE OF GENERATIVE AI

Used the ChatGPT for wording and grammar check

REFERENCES

[1] adrianulbona. 2016. OpenStreetMap in the age of Spark. https://adrianulbona.github.io/2016/12/18/osm-parquetizer.html. Accessed: 2024-04-27.

[2] E. G. Cakmak, B. Kaya, and M. Kaya. 2021. Point-of-Interest Recommendation using Supervised Link Prediction. In *2021 International Conference on Data Analytics for Business and Industry (ICDABI)*. 424–428.

[3] T. Chauhan and H. Palivela. 2021. Optimization and improvement of fake news detection using deep learning approaches for societal benefit. *International Journal of Information Management Data Insights* 1, 2 (2021), 100051. https://doi.org/10.1016/j.jjimei.2021.100051

[4] M. Chen, W.-Z. Li, L. Qian, S.-L. Lu, and D.-X. Chen. 2020. Next POI recommendation based on location interest mining with recurrent neural networks. *Journal of Computer Science and Technology* 35, 3 (2020), 603–616. https://doi.org/10.1007/s11390-020-9107-3

[5] Clément Delteil. 2023. When Should You Use PySpark Over Scikit-Learn ? https://medium.com/geekculture/when-should-you-use-pyspark-over-scikit-learn-b10b91e41252. Accessed: 2024-04-27.

[6] Y. Ensafi, S. H. Amin, G. Zhang, and B. Shah. 2022. Time-series forecasting of seasonal items sales using machine learning – A comparative analysis. *International Journal of Information Management Data Insights* 2, 1 (2022), 100058. https://doi.org/10.1016/j.jjimei.2022.100058

[7] John V. Guttag. 2016. *Introduction to Computation and Programming Using Python: With Application to Understanding Data.* MIT Press, Cambridge, MA, USA.

[8] Halil Ertan. 2021. Apache Spark Performance Boosting. https://towardsdatascience.com/apache-spark-performance-boosting-e072a3ec1179. Accessed: 2024-04-27.

[9] Bart J. J. Kremers, Jonathan Citrin, Aaron Ho, and Karel L. van de Plassche. 2023. Two-step clustering for data reduction combining <scp>DBSCAN</scp> and k-means clustering. *Contributions to Plasma Physics* 63, 5–6 (May 2023). https://doi.org/10.1002/ctpp.202200177

[10] Shivam Kumar. 2021. *Geohash Algorithms: Simplifying Location Data.* Retrieved April 22, 2024 from https://blog.stackademic.com/geohash-algorithms-simplifying-location-data-dd8c3fbaac48

[11] Laurent Leturgez. 2020. On Spark Performance and partitioning strategies. https://medium.com/datalex/on-spark-performance-and-partitioning-strategies-72992bbbf150. Accessed: 2024-04-27.

[12] D. Massimo and F. Ricci. 2019. Clustering users' POIs visit trajectories for next-POI recommendation. In *Information and Communication Technologies in Tourism 2019*. Springer International Publishing, 3–14.

[13] NLTK Project. 2023. *Natural Language Toolkit.* https://www.nltk.org/

[14] Salil Jain. 2020. An Implementation of DBSCAN on PySpark. https://towardsdatascience.com/an-efficient-implementation-of-dbscan-on-pyspark-3e2be646f57d. Accessed: 2024-04-27.

[15] Shivani Goel. [n. d.]. AirBnb Recommender System. https://github.com/goelshivani321/cmpe256-airbnbproject.

[16] G. T. Sri Sai Suresh, K. Pushpak, K. V. Sivanaga Sai, Y. R. Abishake, and C. P. Prathibhamol. 2021. POI and future visitors recommendation. In *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. 1–4.