6240 Parallel Data Processing Final Project Report

Team: Unlocking Airbnb Insights with MapReduce

Team members: Harjis Ahuja, Sumer Bal

Problem Statement: To conduct a comprehensive analysis of Airbnb data with the objective of identifying top listings based on specific conditions and computing the average listing prices in designated areas

Introduction to problem:

One of the most common challenges for every family is planning for a vacation. Not only do you have to take care of kids, packing, transport, etc.. We aim to set out the goal of simplifying this challenge for those choosing to stay at AirBnbs, a solution for those seeking long and short term homestays. As of September 2023, there were over 7 million active listing on the website and we can only assume it has grown since then. This can be daunting to many people as we face decision paralysis. So many places to go and so many homes to choose from, so we have set out with the goal to make this process easier by allowing users to filter and search through regions to find the perfect vacation for them.

As a college student when it comes to trips I struggle with identifying places within my budget. So we have set out with the goal to group and segment data by region to find more information about homes separating them by geographical area, such as average home cost within a region. We set out with the goal to let user's effectively input filter and quickly find results relating to AirBNB data. We work with a dataset of ~ 2 Gb of data allowing us to effectively filter and find valuable data insights through the MapReduce programming model and software framework. This means taking the large amount of data and distributing it amongst servers working parallelly to each other and letting them find pieces of data that fit the filter's put in place by the user.

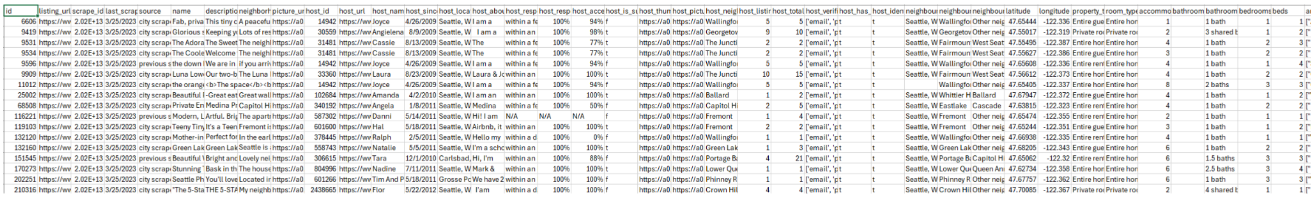
Data:

Our chosen dataset for analysis is the Airbnb data from the United States, last updated eight months ago, available at https://www.kaggle.com/datasets/konradb/inside-airbnb-usa. In this dataset, we will focus on exploring one main file: "listings_details.csv." The "listings_details.csv" file includes the average rating for each listing, contributing to a more nuanced understanding of the data. The data was originally structured in a partitioned format which allowed for individual city information to be stored separately but for the class we also combined the data into a singular csv file for more in depth analytics for states across the country.

Columns

id listing_url scrape_id last_scraped source name description neighborhood_overview picture_url host_id host_url host_name host_since host_location host_about host_response_time host_response_rate host_acceptance_rate host_is_superhost host_thumbnail_url host_picture_url host_neighbourhood host_listings_count host_total_listings_count host_verifications host_has_profile_pic host_identity_verified neighbourhood neighbourhood_cleansed neighbourhood_group_cleansed latitude longitude property_type room_type accommodates bathrooms bathrooms_text bedrooms beds amenities price minimum_nights maximum_nights minimum_minimum_nights maximum_minimum_nights minimum_maximum_nights maximum_maximum_nights minimum_nights_avg_ntm maximum_nights_avg_ntm calendar_updated has_availability availability_30 availability_60 availability_90 availability_365 calendar_last_scraped number_of_reviews number_of_reviews_ltm number_of_reviews_l30d first_review last_review review_scores_rating review_scores_accuracy review_scores_cleanliness review_scores_checkin review_scores_communication review_scores_location review_scores_value license instant_bookable calculated_host_listings_count calculated_host_listings_count_entire_homes calculated_host_listings_count_private_rooms calculated_host_listings_count_shared_rooms reviews_per_month

Image of CSV



Technical Discussion

Purpose of the Task

Through the data obtained from AirBNB we intended to analyze and create insights to allow customers/users to make more informed decisions about their AirBNB vacation rentals.

Explain in a few sentences the main idea for solving the problem.

We set out with an idea for 3 separate spark scripts to find meaningful insights to the AirBNB data. The first being to find the local mean of different neighborhoods, this feature can help users help find locations within budgets to better plan your vacations. Our Second Script allows us to implement search filters to look for specific feature counts such as bedrooms, bathrooms and price range. Finally our third and last script is aimed at getting an analysis if the ratings are affected by the pricing of AirBnbs. To do this we implement the search filter from the previous and from the top 10 AirBNB outputs by rating we label them as High Value if they are also above their neighborhood mean price.

Combining CSV

We achieved our goal by breaking the problem into multiple steps, this involves aggregating the data into one csv file, this was done through the command line which combined and outputted all the csv files together.

```
cat *.csv >combined.csv
```

AirBNB 0.ipyn -> Cleansing the Data

After this we cleansed the data, this involves the handling of null values, missing values or misplaced values. Occasionally the numeric count for a feature was missing but was written in the text description of the feature so we had to parse the text for related feature and write the attribute into the list. We also had certain columns with irrelevant information that was removed from the text so we dropped said columns.

Pseudocode for cleansing

```
# Read CSV
df  = spark.read.csv(input_path)

df.drop(List of columns to drop)

# Cleanse text cells:
For each cell in text column:
    If cell is null:
        Fill cell with "No description"

# Cleanse feature counts:
For each feature:
    # Filter rows where count is null
    null_count_rows = filter_rows_where_count_is_null(feature)

    # Fill null counts with NaN
    null_count_rows.fill_null_counts_with_nan()

    # Parse feature description to extract count if available
    parsed_counts = parse_counts_from_description(feature)

    # Fill null counts with parsed counts

null_count_rows.fill_null_counts_with_parsed_counts(parsed_counts)
```

Output

Columns

id name description neighborhood_overview picture_url neighbourhood neighbourhood_cleansed latitude longitude property_type accommodates bedrooms beds price number_of_reviews number_of_reviews_l30d review_scores_rating bathrooms_numeric

Image of Output CS

id	name	description	neighborhood_overview	picture_ur	neighbourhood	neighbourhood_	latitude	longitude	property_t	accommo	bedrooms	beds	price	number_o	number_o	review_sc	bathroom
177	Tiny Home	160 sq ft + 80 sq ft loft fo	Quiet neighborhood next to pa	https://a0	Denver, Colorado, U	Virginia Village	39.69551	-104.925	Entire gue	2	1	1	79	120	0	4.85	1
1223612	Suite of ro	I have a suite of rooms i	The house is located in the hea	https://a0	Denver, Colorado, U	Cheesman Park	39.73165	-104.971	Private roo	3	1	2	58	184	0	4.71	1
1313699	Platt Park	Beautiful 3-story Bungal	The Platt Park neighborhood is	https://a0	Denver, Colorado, U	Platt Park	39.68332	-104.974	Entire hon	10	5	5	218	24	0	4.71	2
1327856	Downtowr	Welcome to our home! \	Right in the heart of downtown	https://a0	Denver, Colorado, U	Capitol Hill	39.73484	-104.982	Entire tow	6	2	2	139	174	0	4.9	2
1402409	Spacious	This 1928 bungalow has	Our home is on the west side c	https://a0	Denver, Colorado, U	West Colfax	39.73995	-105.041	Entire hon	10	5	6	256	142	2	4.96	2
1529804	Bright & C	Bright, stylish, and priva	The historic and desireable Be	https://a0	Denver, Colorado, U	Berkeley	39.77863	-105.026	Entire gue	2	1	1	55	365	2	4.74	1
1581384	2BR - Wes	Our recently updated W	We love the diversity of the nei	https://a0	Denver, Colorado, U	West Highland	39.76779	-105.05	Entire rent	2	2	2	135	82	1	4.62	1
1583103	W. Wash F	920 sq. ft. W. Wash Park	This neighborhood has everyth	https://a0	Denver, Colorado, U	Speer	39.72088	-104.984	Entire bun	2	1	1	108	39	1	4.85	1
1637744	Park Hill D	Visitors say my home is	No neighborhood overview	https://a0.muscache.com/pic		South Park Hill	39.74446	-104.911	Private roo	3	1	1	80	39	0	4.84	1
1641476	Cozy Creol	Enjoy your stay in this e	This location is ideal! You can	https://a0	Denver, Colorado, U	Sloan Lake	39.75459	-105.05	Entire hon	6	3	3	216	47	2	4.89	2
1733052	Beautiful I	Beautifully remodeled b	Many historical homes with a	https://a0	Denver, Colorado, U	Cheesman Park	39.73799	-104.972	Private roo	4	1	5	101	1367	10	4.93	1
1737365	Stylish & H	Quiet, spacious, renova	The apartment is a five-minute	https://a0	Denver, Colorado, U	Highland	39.75901	-105.015	Entire loft	2	1	1	100	74	0	4.93	1
1758820	Denver Wa	Beautifully renovated 2,	Washington Park is where you	https://a0	Denver, Colorado, U	Washington Par	39.70187	-104.967	Entire hon	6	2	2	265	1	0	5	2
1785877	Cozy and (	Comfortably furnished,	The house is located in the hea	https://a0	Denver, Colorado, U	Cheesman Park	39.7388	-104.97	Private roo	2	1	1	60	170	0	4.7	1
1792152	Peaceful F	Make yourself comforta	Congress Park is a quaint little	https://a0	Denver, Colorado, U	Congress Park	39.73377	-104.949	Private roo	2	2	1	78	254	0	4.93	1.5
1801950	Best Locat	You won't find any one b	Being right on the edge of the "	https://a0	Denver, Colorado, U	CBD	39.74569	-104.992	Entire loft	2	1	1	173	289	1	4.91	1.5
1901266	PrivateBdr	Located near the Univer	Search Cherry Hills Vista neigh	https://a0	Denver, Colorado, U	University	39.66641	-104.966	Private roo	1	1	1	30	100	1	4.93	1
1905596	Charming	The space 	What don't I love? The Highlan	https://a0	Denver, Colorado, U	Berkeley	39.77954	-105.04	Entire gue	4	2	2	110	74	0	4.58	1
1915301	Modern Lc	4 story Modern townhor	No neighborhood overview	https://a0.muscache.com/pic		Highland	39.76875	-105.011	Entire tow	6	2	2	244	53	1	4.75	2.5
1955800	High-Rise	1blk - Convention Cente	THE NEIGHBORHOOD The	https://a0	Denver, Colorado, U	CBD	39.74578	-104.996	Entire rent	3	1	1	200	15	0	5	1
1959836	Artist Cott	Great size private bedro	The neighborhood is filled with	https://a0	Denver, Colorado, U	Clayton	39.76476	-104.958	Private roo	1	1	1	39	297	0	4.74	1
1966103	Wash Parl	Our 2000 sqft brick hom	Location and space is perfect	https://a0	Denver, Colorado, U	Platt Park	39.67937	-104.978	Entire hon	6	3	4	150	8	0	5	2
2104774	Skylight,N	This is a long term furni	THIS IS THE BEST LOCATION IN	https://a0	Denver, Colorado, U	City Park West	39.74245	-104.968	Entire rent	2	1	1	61	52	0	4.75	1
2119667	Architect (	Step into this quirky 2-b	This unique and welcoming br	https://a0	Denver, Colorado, U	Lincoln Park	39.73499	-105.001	Entire hon	4	2	2	258	414	0	4.89	1
2139342	Designer I	This listing is available	Quick access to all kinds of re	https://a0	Denver, Colorado, U	Hilltop	39.72112	-104.913	Entire rent	4	2	2	125	158	0	4.96	1
2216969	Modern St	This listing is available	Quick access to all kinds of re	https://a0	Denver, Colorado, U	Hilltop	39.7212	-104.913	Entire rent	2	1	1	70	156	0	4.92	1
2232323	Cozy Cotta	You will enjoy my cozy h	Friendly with families of all ag	https://a0	Denver, Colorado, U	Country Club	39.72367	-104.966	Entire hon	5	3	3	260	93	0	4.72	3
2239191	Spacious '	Quiet, spacious Victoria	Spacious house on a huge priv	https://a0	Denver, Colorado, U	Clayton	39.76768	-104.955	Entire hon	6	3	4	169	78	0	4.73	1

AirBNB 1.py -> Neighborhood Mean Price

Then to further analyze the data we also computed the mean for each neighborhood in each city, this was done using the combined and cleansed CSV file generated from the previous steps. With this information we were able to read the CSV and groupBy() method which allows us to select columns and essentially creates a unique key for those said columns. Once all the neighborhoods are grouped together we are able to aggregate all the neighborhoods average prices and put it into 1 table and write to a CSV in the format

Pseudocode for Neighborhood Mean Price

```
# Initialize Spark Session
spark = GetOrCreateSparkSession()

# Read CSV file into DataFrame
df = spark.read.csv(input_path)

# Group DataFrame by neighborhood and calculate average price
```

```
group_avg = df.groupBy("neighborhood",
"neighborhood_cleansed").agg(avg("price"))

# Display the computed average prices
group_avg.show()

# Write the computed average prices to CSV file
group_avg.write.mode("overwrite").csv(output_path)

# Stop SparkSession
spark.stop()
```

Output for resulting code

Sherman Oaks, Los Angeles, California, United States	Sherman Oaks	68.5
Harbor City, California, United States	Signal Hill	43
Los Angeles, Hollywood Hills, California, United States	Hollywood Hills	156
Brooklyn, New York, United States	Fort Hamilton	102.6521739
Corona, New York, United States	Jackson Heights	65
Portland, Oregon, United States	Creston-Kenilworth	80.5
Pompano Beach, Florida, United States	Lighthouse Point	658
Hawaiian Paradise Park , Hawaii, United States	Puna	211
Burleson, Texas, United States		35
Beverly Hills, California, United States	Carthay	154.75
Staten Island, New York, United States	Graniteville	99
United States	Creston-Kenilworth	127
Burlingame, California, United States	Burlingame	207.6530612
Washington, District of Columbia, United States	Eastland Gardens, Ke	124.5
Pompano Beach, Florida, United States	Pompano Beach	282.7311385
	Northwest Antelope '	41.5
Redondo Beach,, California, United States	Redondo Beach	138
Pacifica, California, United States	Unincorporated Area	415
	Scott	268.0666667
Wyoming, Minnesota, United States	Chisago	55
	District 33	72.58823529
Las Vegas, Nevada, United States	Nellis AFB	138
Nashville, Tennessee, United States	District 3	209.3076923
	Gentilly Terrace	192.15
New York, United States	Concourse	215.3333333

AirBNB 2.py -> Search Filter Top 10

After this we intended to create a search feature so we could list multiple features that would be interesting to us. We can list requirements such as the number of people we want to accommodate or the number of bathrooms or bedrooms we would like to have.

For example, say I want a home with 3 bedrooms and 2 bathrooms between $ 50 - $ 200 a night. We could request this from the EMR Cluster by performing this operation

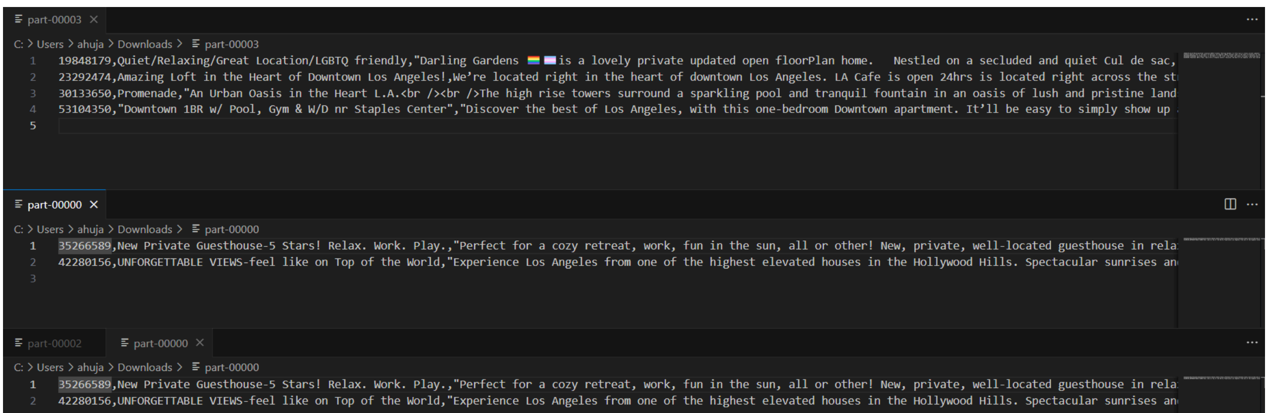
Input_path output_path bedrooms 3 bathrooms 2 50 200

Which would allow us to map those arguments into the filter to better parse the data.

Pseudocode for Search Filter Top 10

```
# Get input arguments from command line
input_path = sys.argv[1]
output_path = sys.argv[2]
filter_values = sys.argv[3]

# Initialize Spark Session
spark = GetOrCreateSparkSession()

# Read CSV file into DataFrame
df = spark.read.csv(input_path)

# Filter records
df_filtered = df.filter(filter_records)

# Map records
mapped_records = df_filtered.map(emit_key_value)

# Group records
grouped_records = mapped_records.groupByKey()

# Get top 10 records by rating
top_10_records =
grouped_records.parallelize(grouped_records.top(10, rating))

# Save top 10 records as text files
top_10_records.saveAsTextFile(output_path)

# Define function to filter records
def filter_records(record):
    if fields match filter and price within range:
        return True
    return False
```

Image of Output



AirBNB 3.py -> Analysis of Top 10

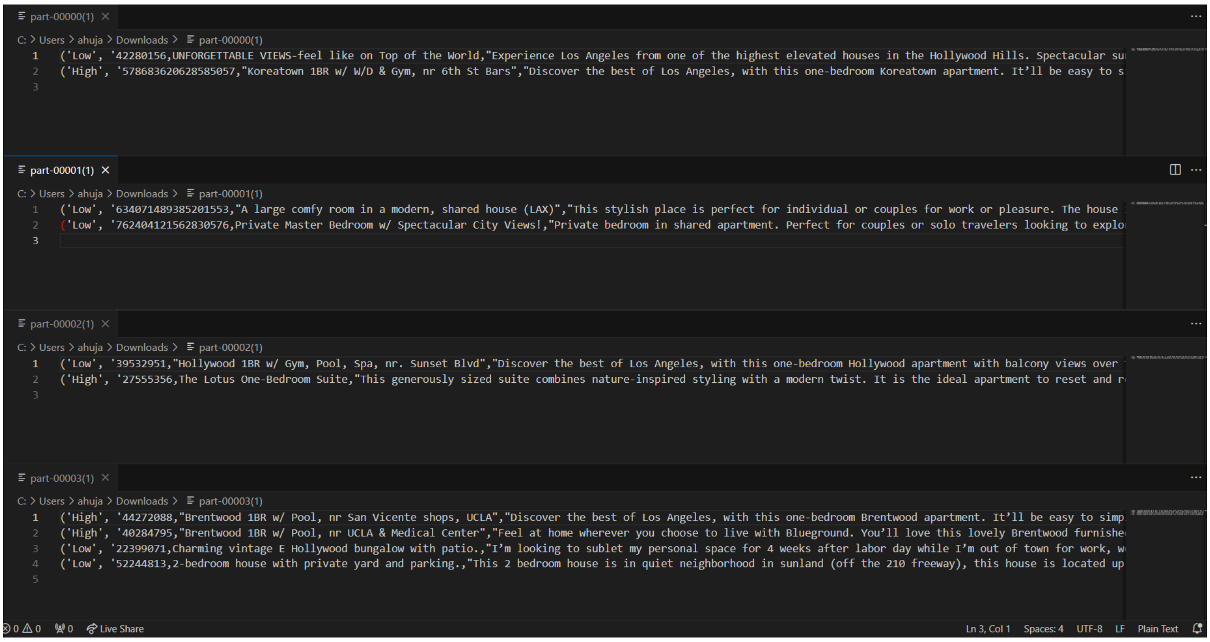
Finally say we wanted to find homes for the best value for my dollar, in this case we decided to implement a script which would take in arguments as before and pull the 10 highest reviewed AirBnb rentals and look at the mean neighborhood price of the neighborhood. We would then evaluate whether they fall above or below said mean neighborhood price. This creates a list of rentals that meet our requirements, are highly rated and are below the mean price in their neighborhood.

PseudoCode for Top 10 Value Finder

```
# Get input arguments from command line
input_path = sys.argv[1]
output_path = sys.argv[2]
filter_values = sys.argv[3]

# Initialize Spark Session
spark = GetOrCreateSparkSession()

# Read CSV file into DataFrame
df = spark.read.csv(input_path)

# Filter records
df_filtered = df.filter(filter_records)

# Map records
mapped_records = df_filtered.map(emit_key_value)

# Group records
grouped_records = mapped_records.groupByKey()

# Get top 10 records by rating
top_10_records =
grouped_records.parallelize(grouped_records.top(10, rating))

# Read neighborhood information from CSV
neighborhood_info = spark.read.csv(neighbor_hood_path.csv)

# Label top 10 records
label_records(top_10_records, neighborhood_info)

# Save labeled top 10 records as text files
top_10_records.saveAsTextFile(output_path)

# Define function to label records
def label_records(records, neighborhood_info):
```

```
    for record in records:
        if neighborhood_mean_info.neighborhood ==
record.neighborhood and neighborhood_mean_info.price <
record.price:
            record.value = "High"
        else:
            record.value = "Low"

# Define function to filter records
def filter_records(record):
    if fields match filter and price within range:
        return True
    return False
```

Output for AirBNB Value



SCRIPTS Submitted

AirBNB 0.ipynb -> Cleansing Data

AirBNB 1.py -> Neighborhood Mean Data

AirBNB 2.py -> Search Filter Top 10

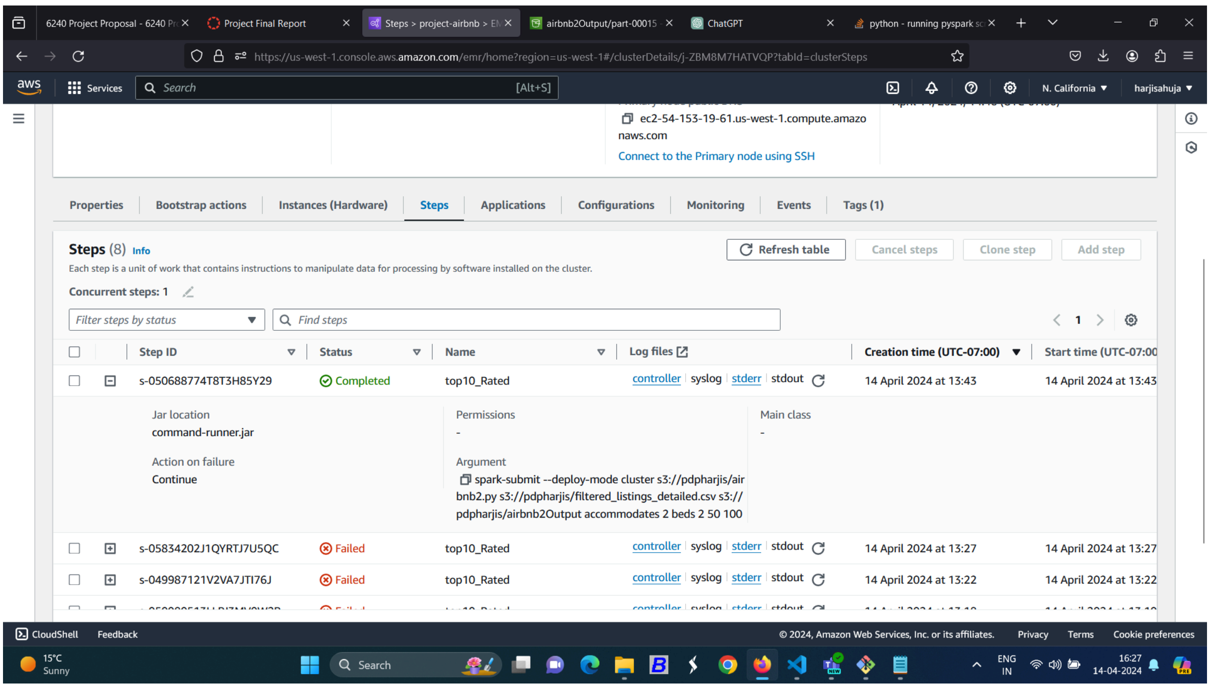
AirBNB 3.py -> Analysis of Top 10

Performance analytics

We computed our performance at 2 different settings, one with 2 m5x.large Worker Nodes and one with 5 m5.xlarge Worker Nodes. After we looked at the results and performed the preprocessing our original Dataset went from 10Gb -> 1 Gb which was an issue as our dataset wasn't as large as we actually preferred. After this when comparing the results we see generally equal performance across all 3 spark applications.

	Airbnb1	Airbnb2	Airbnb3
2 Core	42s	32s	32s
5 Core	66s	32s	34s

This is generally true except for AirBNB 1's runtime which was found to be faster on 2 Worker Nodes than 5, this we believe is caused by low processing times but high data transportation times. Transferring the data might take excessive amounts of time making the one with 5 worker nodes operate 24 seconds slower. Also our entire project was done using Spark Applications so there might be a configuration challenge. Throughout our application no syslog files were generated however we did get controller and stderr logs from all the runs which are included in the ZIP.



Challenges:

There were numerous challenges faced throughout this entire process but the greatest one was data pre-processing to get the data into a manageable format. The CSV file was extremely challenging to load and process and took 95% of our time. Initially we set out to run this project using Java Hadoop but as the CSV file was so challenging to load and run we decided to pivot to using PySpark, not only does this allow us to learn a new technology but when dealing with the CSV we are able to use all of Python's built in features to simplify this process extensively. Now

Cleansing of csv input files

Our solution to combining and cleansing the data came from some discoveries and command line features. Initially we were stuck on combining the CSV but we soon realized if we copied all the listing_details.csv files for each city into a single directory we were able to combine the data using a single command line prompt as seen above. This allowed us to combine the CSV. As for the actual cleansing of this data, we ended up having to go through the CSV line by line, and parse carefully. The amenities column of the original CSV was formatted in an array with double quotes which made it difficult to parse so we ended up dropping the entire column to simplify our pre-processing as any CSV loader we tried to use would get confused on what the individual cells were.

Steps

AWS EMR Cluster Settings
EMR Version 7.xx, Default Spark Settings
While adding a step to run our script we used the generic way of using a custom jar that is an inbuilt command-runner.jar and provided the Argument as spark-submit –deploy-mode cluster script.py arg

We were not required to copy data around but for our third script we used the entire aggregated csv path (generated by running our first script) as an input argument.

Conclusion

Throughout this project, our team delved deeply into MapReduce problems and gained valuable experience with PySpark, a powerful tool for parallel data processing. We seamlessly combined simple linear processing techniques with parallel processing methods as needed, optimizing our approach for efficiency.

Harjis Ahuja worked on developing essential components such as the cleansing process, search filter, and the Spark application for identifying high/low value properties.

Sumer Bal worked on analyzing neighborhood mean price values and further refining the cleansing process as well as troubleshooting various aspects of the project, ensuring smooth execution and resolving any challenges that arose along the way.

One of the major challenges we encountered during the project was the learning curve associated with implementing Spark applications, both locally and on AWS as there were no system logs generated. Initially attempting to use Java for processing proved cumbersome, particularly when handling CSV files. However, transitioning to PySpark proved to be a game-changer. Setting up Spark was notably simpler, and the concise syntax of PySpark significantly reduced the lines of code required for application development, ultimately streamlining our workflow and enhancing productivity.

Future Advancements

We were thinking about training a ML model on the parameters of AirBNB rentals to better adjust price for new AirBNB Home Renters. That way they could make more informed decisions about renting their own properties and for how much they should put their home up for rent based on data from other rentals.

We can enhance our existing filtering script by incorporating machine learning models. These models will do more than just filter data; they'll also offer personalized suggestions tailored to our preferences, saving us time by reducing the need to sift through numerous listings.

Additionally, we can conduct deeper analyses to draw multiple insights, such as determining the factors that influence pricing or ratings, or identifying if specific patterns are unique to certain regions.

References:

Dataset - https://www.kaggle.com/datasets/konradb/inside-airbnb-usa