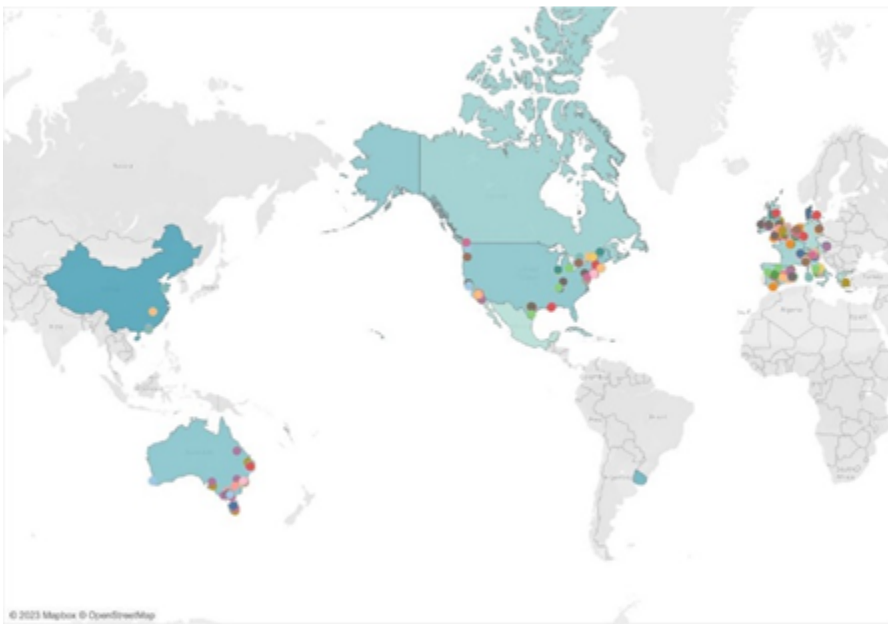
Project Report

Group 53

Richard Wang 509402, Jimeng Yin 516867, Bosan Hsu 519095

Data Overview

The data utilized in this study pertains to Airbnb's global pricing. This dynamic dataset, last updated four months ago, was sourced from Kaggle, courtesy of Joakim Arvidsson. The dataset is structured and sizable, with a volume of 1.94 GB, encompassing 89 columns and over 36 million records.



This map provides a comprehensive view of Airbnb's global presence. A corresponding map illustrates the distribution of Airbnb locations, with colored dots representing cities and large color blocks indicating country regions where Airbnb operates. The color gradient signifies the average price, with lighter shades denoting lower prices.

Significance of the Data

The importance of studying Airbnb's pricing data is multifold. Primarily, it enables the understanding of market trends, as price fluctuations can offer insights into the supply and demand dynamics across various locations and times. Additionally, the data can uncover patterns and correlations, such as the relationship between specific amenities or neighborhoods and price levels.

For policymakers and city planners, comprehending Airbnb's pricing data is crucial. It can provide insights into how short-term rentals are impacting housing markets and neighborhoods, thereby informing policy decisions and urban planning strategies. For researchers and data scientists, Airbnb's pricing data serves as a rich resource for exploratory data analysis, predictive modeling, and machine learning. The insights gleaned from this data can be extrapolated to other domains and industries, underscoring the significance of studying Airbnb's pricing data.

Problem Statement

The objective of this project is to investigate the influence of Airbnb listing ratings and features on their prices. To address this, we aim to predict future Airbnb prices using machine learning methods and statistical modeling. The process entails data collection and cleaning, encompassing information on listing ratings, features, and prices. This data will subsequently be employed to train our models, which will be optimized to enhance the accuracy of our predictions.

Analysis Methods

Data Analysis:

1. Word Count

    Our dataset contained numerous comments and words related to transit and amenities. To determine the most frequently occurring characteristics, we utilized a word count method.

2. SQL's "Group By"

    For response time and property type, as they are categorical variables, we recommend employing SQL's "group by" function to display the distinctions between each group.

Machine Learning:

Linear Regression

To determine which factors have the greatest impact on price, we develop two linear regression models and examine the coefficient and p-value of the independent variables to figure out which variables most affect the price.

Analysis Results

Transit:

Based on our analysis, the three most frequently appearing words were "phone," "email," and "reviews." The term "jumio" only showed up about 60% as frequently as "reviews," so we recommend treating it as less important than the top three words. As our transit data came from customer notes, we advise that customers value these three factors.

Customers highly value being able to directly contact Airbnb hosts by phone or email. Phone calls can offer immediate answers and help establish trust and a personal connection. Similarly, email communication is also important for its convenience and the ability to keep a record of agreements and information exchanged. It allows for detailed inquiries and is less intrusive than a phone call, which could be preferred by some customers. Reviews are an essential element of decision-making for Airbnb clients as they depend on prior guests' experiences to evaluate the accommodation's quality and the host's reliability. Positive reviews can significantly impact prospective guests' perception and selection.

Transit	count	AveragePrice
phone	13248	9.730272202364587
email	13087	9.733401240855635
reviews	12718	9.731131643948403
jumio	7965	9.738031119090365
United States	5892	9.904449741756059
1.0	5706	9.570974576271187
United Kingdom	3762	9.66472602739726
Spain	3389	9.617452440033086
facebook	3288	9.789454545454545
France	3044	10.288540534253647

Amenities:

This result could be very influential for hosts looking to optimize their pricing strategy and determine which amenities to prioritize to increase their listing's appeal and profitability. The analysis clearly shows that certain amenities are associated with a higher average price. In our amenities data, TV has the highest average price among the top 10 frequent words. However, despite this, TVs are only listed in 225,280 out of 494,955 total cases, which is less than half (45.5%). Even if we assume that there is a 50 % chance that the property without a TV in its amenity's description is due to a recording mistake, many properties still do not have a TV. Hence, we suggest that hosts provide TVs to attract more customers, and properties with TV can justify higher rental rates.

Amenity	count	AveragePrice
Wireless Internet	301897	139.894383855269
Kitchen	296850	141.42693088455155
Heating	285280	138.53689722531155
Essentials	272512	140.51520093750932
Washer	235544	142.11130688508692
TV	225280	154.405368044634
Internet	190759	144.2520076196168
Hangers	183868	138.5601776658159
Shampoo	183126	141.3421657196129
Smoke detector	177721	147.89176864200925

Response Time

The "Host Response Time" is categorized into four groups: "within a few hours," “within an hour,” “within a day” and “a few days or more.” Despite the general expectation that faster response times would lead to higher ratings, the data presents a slightly nuanced picture. Specifically, hosts who respond within an hour have an average rating that is slightly lower than those who respond within a few hours.

When a host frequently responds to inquiries within a few hours, guests might create the expectation that response times are prompt but considerate. However, if a host answers within an hour, it creates the impression of an extremely high level of immediate service, which can be challenging to uphold consistently and might result in a reduced rating if that expectation is not fulfilled in other areas of the service. Nonetheless, the faster a host responds to customers, the more costs it generates.

Hence, it would be better for hosts to answer inquiries within a few hours but not within an hour.

Host Response Time	Count_Number_of_Reviews	Avg_Review_Scores_Rating
within a few hours	5144	93.33547257876313
within an hour	18333	92.95301757066463
within a day	2047	92.86112469437653
a few days or more	119	89.8655462184874

Property Types

Apartments are the most common property type among the top neighborhoods in popular cities. The reason might be in urban areas where space is limited, apartments are a practical option for both hosts and travelers. They provide the essential facilities in a confined area and are usually situated conveniently near popular city attractions, business hubs, and public transportation. Also, the high incidence of apartments indicates a demand for lodgings that cater to a diverse set of travelers. Airbnb's user base includes not only travelers seeking luxury, but also individuals in search of a comfortable, home-like lodging option. Thus, we recommend that apartment owners in major cities consider joining the Airbnb industry.

City	Neighbourhood	Property Type	AvgRating	COUNTS
Amsterdam	Oud-West	Apartment	94.33739130434783	1150
Berlin	Neukölln	Apartment	93.6020482809071	1367
Brooklyn	Williamsburg	Apartment	93.89115646258503	1617
Brooklyn	Williamsburg	Loft	94.4014598540146	137
København	Nørrebro	Apartment	94.19538572458544	1387
London	LB of Islington	House	92.35036496350365	137
London	LB of Islington	Apartment	92.46984924623115	398
Los Angeles	Mid-Wilshire	Apartment	92.56801195814649	669
Los Angeles	Mid-Wilshire	House	94.08243727598567	279
New York	Upper West Side	Apartment	93.18666666666667	900
Paris	Montmartre	Apartment	92.30014124293785	1416
Roma	Prati	Bed & Breakfast	91.61578947368422	190
Roma	Prati	Apartment	93.44505494505495	546
Toronto	Downtown Toronto	House	92.3	100
Toronto	Downtown Toronto	Apartment	93.54385964912281	456
Toronto	Downtown Toronto	Condominium	94.73451327433628	226

Machine Learning Results

Linear Regression between price and room features:

For the first regression, we choose accommodates, bathrooms, bedrooms, and beds as independent variables; and we choose price as the dependent variable. From the data we collected, we get the following linear regression:

price = 28.48 + 21.77 accommodates + 13.33 bathrooms + 35.40 bedrooms - 13.65 beds

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.164
Model:                            OLS   Adj. R-squared:                  0.164
Method:                 Least Squares   F-statistic:                 2.374e+04
Date:                Sun, 03 Dec 2023   Prob (F-statistic):               0.00
Time:                        15:57:08   Log-Likelihood:            -3.0714e+06
No. Observations:              484544   AIC:                         6.143e+06
Df Residuals:                  484539   BIC:                         6.143e+06
Df Model:                           4
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const           28.4832      0.484     58.846      0.000      27.535      29.432
Accommodates    21.7748      0.184    118.073      0.000      21.413      22.136
Bathrooms       13.3307      0.429     31.072      0.000      12.490      14.172
Bedrooms        35.4008      0.360     98.417      0.000      34.696      36.106
Beds           -13.6465      0.252    -54.056      0.000     -14.141     -13.152
==============================================================================
Omnibus:                   311633.729   Durbin-Watson:                   0.850
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          3347489.466
Skew:                           3.051   Prob(JB):                         0.00
Kurtosis:                      14.339   Cond. No.                         14.8
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

The analysis reveals that all four variables are statistically significant; however, the R-squared value is not particularly high. After deriving the linear function, the data was split into 70% for training and 30% for testing to apply machine learning techniques. The outcomes yielded a mean squared error of 19,624.42 and an R-squared value of 16.51%. These results are suboptimal, suggesting three potential issues:

1. There may be insufficient feature relevance, indicating that the chosen independent variables are not strong predictors of the dependent variable, and other critical factors may be overlooked.
2. The model could be underfitting, meaning it is too simplistic to encapsulate the data's complexities, lacking the necessary capacity to learn from the data adequately.
3. A lack of data might be hindering the model's ability to discern underlying patterns effectively.

Linear Regression between price and rate:

For the second regression, we choose Review Scores Rating, Review Scores Cleanliness, and Review Scores Location as independent variables; and we choose price as the dependent variable. From the data we collected, we get the following linear regression:

price = -34.57 + 0.58 Review Scores Rating + 0.01 Review Scores Cleanliness + 12.09 Review Scores Location

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.008
Model:                            OLS   Adj. R-squared:                  0.008
Method:                 Least Squares   F-statistic:                     936.1
Date:                Sun, 03 Dec 2023   Prob (F-statistic):               0.00
Time:                        16:40:00   Log-Likelihood:            -2.3070e+06
No. Observations:              361000   AIC:                         4.614e+06
Df Residuals:                  360996   BIC:                         4.614e+06
Df Model:                           3
Covariance Type:            nonrobust
=============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------------
const                      -34.5742      3.251    -10.634      0.000     -40.947     -28.202
Review Scores Rating         0.5834      0.042     13.982      0.000       0.502       0.665
Review Scores Cleanliness    0.0133      0.324      0.041      0.967      -0.621       0.648
Review Scores Location      12.0860      0.335     36.128      0.000      11.430      12.742
==============================================================================
Omnibus:                   227680.671   Durbin-Watson:                   0.948
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          2193573.632
Skew:                           3.020   Prob(JB):                         0.00
Kurtosis:                      13.458   Cond. No.                     1.28e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.28e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

The evaluation indicates that among the variables, only Review Scores Rating and Review Scores Location are statistically significant, yet the R-squared value remains modest. After formulating the linear equation, the dataset was partitioned in the same 70% training and 30% testing manner as before. The derived results produced a mean squared error (MSE) of 75,000.15 and an R-squared value of 34.73%. These figures are somewhat disappointing, pointing to three potential complications:

1. Insufficient or Irrelevant Features: The independent variables selected may not be adequately forecasting the Price. There could be vital determinants impacting the Price that the model has not accounted for.
2. Non-Linear Relationships: Should the connections between the independent variables and the Price be non-linear, a linear model will fail to appropriately capture these complexities, resulting in a diminished R-squared.
3. High Variance in the Price: A substantial range or variability in the Price that the independent variables do not explain could cause the model to falter in making precise predictions, as evidenced by the elevated MSE.

Conclusion

This analysis has brought several significant insights relevant to the Airbnb market. A pivotal aspect is the importance of effective communication; specifically, phone and email interactions and the quality of reviews are critical to customer satisfaction and perception. Additionally, amenities such as televisions are highlighted as valued by customers and should be considered a standard offering.

Responsiveness is another critical factor; hosts who engage with customers promptly, ideally within a few hours, will likely enhance customer experience and encourage positive reviews. For those considering market entry, apartment owners in populous cities may find Airbnb a lucrative platform, primarily if they can ensure high-quality accommodations.

Our statistical analysis identifies several features with significant predictive power for pricing models. These include the number of accommodations, bathrooms, bedrooms, and beds, and the review scores for the overall rating and location. These factors should be carefully considered when setting prices, as they directly correlate with customer expectations and willingness to pay.

However, it is noteworthy that the linear regression models employed in this analysis did not serve as robust predictors within the machine learning framework. This outcome suggests that while the identified variables are relevant, the linear approach may not fully capture the complexity of the market dynamics or may be insufficient to handle the nonlinear relationships inherent in the data.

Future strategies should expand the feature set to encapsulate more relevant variables, explore more sophisticated, non-linear models, and increase the dataset's breadth to capture a more comprehensive array of market behaviors. By doing so, we anticipate a more accurate and reliable predictive model that aligns closer with the multifaceted nature of the Airbnb marketplace.

Appendix

Code for the project

In [1]:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode, split, col, avg


spark = SparkSession.builder.appName("AmenitiesWordCount").getOrCreate()
df = spark.read.csv('airbnb-listings.csv', header=True, inferSchema=True)

# group by Amenity
df_exploded = df.withColumn('Amenity', explode(split(col('Amenities'), ',')))

# wordcount
word_counts = df_exploded.groupBy('Amenity').count().orderBy(col('count').desc

# top 10 frequent word
top_words = word_counts.limit(10)

# top 10's avg price
average_prices_ame = df_exploded.groupBy('Amenity').agg(avg(col('Price')).alias


top_words_with_price = top_words.join(average_prices_ame, 'Amenity').select('Ar
top_words_with_price.orderBy(col('count').desc()).limit(10).show()

# 停止 SparkSession
spark.stop()
```

```
+-----------------+------+------------------+
|          Amenity| count|      AveragePrice|
+-----------------+------+------------------+
|Wireless Internet|301897|  139.894383855269|
|          Kitchen|296850|141.42693088455155|
|          Heating|285280|138.53689722531155|
|       Essentials|272512|140.51520093750932|
|           Washer|235544|142.11130688508692|
|               TV|225280|  154.405368044634|
|         Internet|190759| 144.2520076196168|
|          Hangers|183868| 138.5601776658159|
|          Shampoo|183126| 141.3421657196129|
|   Smoke detector|177721|147.89176864200925|
+-----------------+------+------------------+
```

In [2]:

```
spark = SparkSession.builder.appName("AmenitiesWordCount").getOrCreate()
df = spark.read.csv('airbnb-listings.csv', header=True, inferSchema=True)

# group by Transit
df_exploded = df.withColumn('Transit', explode(split(col('Transit'), ',')))

# wordcount
word_counts = df_exploded.groupBy('Transit').count().orderBy(col('count').desc

# top 10 frequent word
top_words = word_counts.limit(10)

# top 10's avg price
average_prices_transit = df_exploded.groupBy('Transit').agg(avg(col('Price')).a
```

```
top_words_with_price = top_words.join(average_prices_transit, 'Transit').selec
top_words_with_price.orderBy(col('count').desc()).limit(10).show()

# 停止 SparkSession
spark.stop()
```

```
+---------------+-----+------------------+
|        Transit|count|      AveragePrice|
+---------------+-----+------------------+
|          phone|13248| 9.730272202364587|
|          email|13087| 9.733401240855635|
|        reviews|12718| 9.731131643948403|
|          jumio| 7965| 9.738031119090365|
|  United States| 5892| 9.904449741756059|
|            1.0| 5706| 9.570974576271187|
| United Kingdom| 3762|  9.66472602739726|
|          Spain| 3389| 9.617452440033086|
|       facebook| 3288| 9.789454545454545|
|         France| 3044|10.288540534253647|
+---------------+-----+------------------+
```

In [8]:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import to_date
from pyspark.sql.types import IntegerType, FloatType

# Create a SparkSession
spark = SparkSession.builder \
    .appName("AirbnbDataAnalysis") \
    .getOrCreate()

# Read the CSV file into a DataFrame
df = spark.read \
    .option("header", "true") \
    .option("sep", ",") \
    .option("quote", "\"") \
    .option("escape", "\"") \
    .csv("airbnb-listings.csv")

df = df.dropna(subset=["Host Response Time"])
# Cast the 'Number of Reviews' to Integer and 'Last Review' to Date
df = df.withColumn("Number of Reviews", df["Number of Reviews"].cast(IntegerTy
df = df.withColumn("Last Review", to_date(df["Last Review"], "yyyy-MM-dd"))
df = df.withColumn("Review Scores Rating", df["Review Scores Rating"].cast(Floa

# Create a temporary view to run SQL queries
df.createOrReplaceTempView("listings")

# Now run the SQL query
query = """
SELECT
    `Host Response Time`,
    COUNT(`Number of Reviews`) AS Count_Number_of_Reviews,
    AVG(`Review Scores Rating`) AS Avg_Review_Scores_Rating
FROM listings
WHERE `Number of Reviews` > 50 AND `Last Review` > '2016-01-01'
GROUP BY `Host Response Time`
ORDER BY AVG(`Review Scores Rating`) DESC
"""
```

```
result = spark.sql(query)
result.show(4)

# Stop the Spark session
spark.stop()
```

```
+------------------+-----------------------+------------------------+
|Host Response Time|Count_Number_of_Reviews|Avg_Review_Scores_Rating|
+------------------+-----------------------+------------------------+
|within a few hours|                   5144|       93.33547257876313|
|    within an hour|                  18333|       92.95301757066463|
|      within a day|                   2047|       92.86112469437653|
|a few days or more|                    119|        89.8655462184874|
+------------------+-----------------------+------------------------+
only showing top 4 rows
```

In [32]:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, avg
from pyspark.sql.window import Window
from pyspark.sql import functions as F


spark = SparkSession.builder.appName("TopCitiesNeighbourhoodAnalysis").getOrCre

df = spark.read \
    .option("header", "true") \
    .option("sep", ",") \
    .option("quote", "\"") \
    .option("escape", "\"") \
    .csv("airbnb-listings.csv")


df = df.na.drop(subset=["City", "Neighbourhood", "Property Type", "Review Score

df = df.withColumn("Review Scores Rating", col("Review Scores Rating").cast("f

df.createOrReplaceTempView("listings")

query = """
WITH CityFrequency AS (
    SELECT City, COUNT(*) AS Listings
    FROM listings
    GROUP BY City
    ORDER BY Listings DESC
    LIMIT 10
),
NeighbourhoodFrequency AS (
    SELECT City, Neighbourhood, COUNT(*) AS Listings
    FROM listings
    WHERE City IN (SELECT City FROM CityFrequency)
    GROUP BY City, Neighbourhood
),
MaxNeighbourhoodPerCity AS (
    SELECT City, MAX(Listings) AS MaxListings
    FROM NeighbourhoodFrequency
    GROUP BY City
),
TopNeighbourhoods AS (
    SELECT nf.City, nf.Neighbourhood
```

```
    FROM NeighbourhoodFrequency nf
    INNER JOIN MaxNeighbourhoodPerCity mnc
    ON nf.City = mnc.City AND nf.Listings = mnc.MaxListings
)
SELECT tn.City, tn.Neighbourhood, l.`Property Type`, AVG(l.`Review Scores Ratin
FROM TopNeighbourhoods tn
JOIN listings l ON tn.City = l.City AND tn.Neighbourhood = l.Neighbourhood
GROUP BY tn.City, tn.Neighbourhood, l.`Property Type`
HAVING COUNTS >= 100
"""

result = spark.sql(query)
result.orderBy(col('City').asc()).show(100)
```

```
+-----------+----------------+---------------+-----------------+------+
|       City|   Neighbourhood|  Property Type|        AvgRating|COUNTS|
+-----------+----------------+---------------+-----------------+------+
|  Amsterdam|        Oud-West|      Apartment|94.33739130434783|  1150|
|     Berlin|        Neukölln|      Apartment| 93.6020482809071|  1367|
|   Brooklyn|    Williamsburg|      Apartment|93.89115646258503|  1617|
|   Brooklyn|    Williamsburg|           Loft| 94.4014598540146|   137|
|  København|        Nørrebro|      Apartment|94.19538572458544|  1387|
|     London| LB of Islington|          House|92.35036496350365|   137|
|     London| LB of Islington|      Apartment|92.46984924623115|   398|
|Los Angeles|    Mid-Wilshire|      Apartment|92.56801195814649|   669|
|Los Angeles|    Mid-Wilshire|          House|94.08243727598567|   279|
|   New York| Upper West Side|      Apartment|93.18666666666667|   900|
|      Paris|      Montmartre|      Apartment|92.30014124293785|  1416|
|       Roma|           Prati|Bed & Breakfast|91.61578947368422|   190|
|       Roma|           Prati|      Apartment|93.44505494505495|   546|
|    Toronto|Downtown Toronto|          House|             92.3|   100|
|    Toronto|Downtown Toronto|      Apartment|93.54385964912281|   456|
|    Toronto|Downtown Toronto|    Condominium|94.73451327433628|   226|
+-----------+----------------+---------------+-----------------+------+
```

Code for Machine Learning Part 1

In [1]:
```
import findspark
findspark.init()
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel
(R) SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math K
ernel Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel
(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel
(R) SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math K
ernel Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel
(R) AVX) instructions.
```

In [2]:
```
import pyspark
from pyspark.sql import SparkSession
spark=SparkSession.builder.master("local").appName('LinearRegression').getOr
sc=spark.sparkContext
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLog
Level(newLevel).
23/12/03 22:38:18 WARN NativeCodeLoader: Unable to load native-hadoop librar
y for your platform... using builtin-java classes where applicable
```

In [3]:
```
df = spark.read.option("header", "true").option("sep", ";").csv("airbnb-list
```

In [4]:
```
from pyspark.sql.functions import col
df = df.withColumn("Price", col("Price").cast("double")).withColumn("Accommo
df = df.na.drop(subset=["Accommodates", "Bathrooms", "Bedrooms", "Beds", "Pr
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssembler(inputCols=['Accommodates', 'Bathrooms', 'Bedroom
output = assembler.transform(df)
output.select('ML_Features', 'Price').show(5)
```

```
+-----------------+-----+
|      ML_Features|Price|
+-----------------+-----+
|[4.0,1.0,2.0,2.0]|125.0|
|[4.0,1.0,2.0,4.0]|130.0|
|[2.0,1.0,0.0,1.0]| 80.0|
|[2.0,1.0,1.0,1.0]|150.0|
|[3.0,1.5,2.0,2.0]|144.0|
+-----------------+-----+
only showing top 5 rows
```

In [5]:
```
final_data = output.select('ML_Features', 'Price')
train_data,test_data = final_data.randomSplit([0.7,0.3])
train_data.show(5)
```

```
[Stage 3:>                                                          (0 + 1)
/ 1]
```

```
+------------------+-----+
|       ML_Features|Price|
+------------------+-----+
|[1.0,0.0,1.0,1.0]| 18.0|
|[1.0,0.0,1.0,1.0]| 25.0|
|[1.0,0.0,1.0,1.0]| 27.0|
|[1.0,0.0,1.0,1.0]| 35.0|
|[1.0,0.0,1.0,1.0]| 37.0|
+------------------+-----+
only showing top 5 rows
```

In [6]:

```python
from pyspark.ml.regression import LinearRegression
lr=LinearRegression(featuresCol='ML_Features',labelCol='Price')
trained_model=lr.fit(train_data)
results=trained_model.evaluate(train_data)
```

```
23/12/03 22:39:01 WARN Instrumentation: [3f98bbeb] regParam is zero, which m
ight cause numerical instability and overfitting.
23/12/03 22:39:02 WARN InstanceBuilder: Failed to load implementation from:d
ev.ludovic.netlib.blas.JNIBLAS
23/12/03 22:39:16 WARN InstanceBuilder: Failed to load implementation from:d
ev.ludovic.netlib.lapack.JNILAPACK
```

In [7]:

```python
print('Mean Squared Error :',results.meanSquaredError)
print('Rsquared Error :',results.r2)
```

```
Mean Squared Error : 19624.418093429384
Rsquared Error : 0.16508157509431598
```

Code for Machine Learning Part 2

In [1]:
```
import findspark
findspark.init()
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel
(R) SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math K
ernel Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel
(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel
(R) SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math K
ernel Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel
(R) AVX) instructions.
```

In [2]:
```
import pyspark
from pyspark.sql import SparkSession
spark=SparkSession.builder.master("local").appName('LinearRegression').getOr
sc=spark.sparkContext
```

```
23/12/04 12:48:28 WARN Utils: Your hostname, wangruiqideMacBook-Pro.local re
solves to a loopback address: 127.0.0.1; using 172.27.108.50 instead (on int
erface en0)
23/12/04 12:48:28 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to anot
her address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLog
Level(newLevel).
23/12/04 12:48:29 WARN NativeCodeLoader: Unable to load native-hadoop librar
y for your platform... using builtin-java classes where applicable
```

In [3]:
```
df = spark.read.option("header", "true").option("sep", ";").csv("airbnb-list
```

In [4]:
```
from pyspark.sql.functions import col
df = df.withColumn("Price", col("Price").cast("double")).withColumn("Review
df = df.na.drop(subset=["Review Scores Rating", "Review Scores Cleanliness",
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssembler(inputCols=['Review Scores Rating', 'Review Score
output = assembler.transform(df)
output.select('ML_Features', 'Price').show(5)
```

```
[Stage 2:>                                                          (0 + 1)
/ 1]
+-----------------+-----+
|      ML_Features|Price|
+-----------------+-----+
|[100.0,10.0,10.0]|125.0|
|   [97.0,9.0,9.0]|130.0|
|   [78.0,8.0,9.0]| 80.0|
|  [97.0,10.0,9.0]|150.0|
| [97.0,10.0,10.0]|144.0|
+-----------------+-----+
only showing top 5 rows
```

In [5]:
```python
final_data = output.select('ML_Features', 'Price')
train_data,test_data = final_data.randomSplit([0.7,0.3])
train_data.show(5)
```

```
[Stage 3:>                                                          (0 + 1)
/ 1]
```

```
+--------------+-----+
|   ML_Features|Price|
+--------------+-----+
|[20.0,2.0,2.0]| 25.0|
|[20.0,2.0,2.0]| 46.0|
|[20.0,2.0,2.0]| 47.0|
|[20.0,2.0,2.0]| 83.0|
|[20.0,2.0,2.0]| 89.0|
+--------------+-----+
only showing top 5 rows
```

In [6]:
```python
from pyspark.ml.regression import LinearRegression
lr=LinearRegression(featuresCol='ML_Features',labelCol='Price')
trained_model=lr.fit(train_data)
results=trained_model.evaluate(train_data)
```

```
23/12/04 12:52:19 WARN Instrumentation: [bd510a7d] regParam is zero, which m
ight cause numerical instability and overfitting.
23/12/04 12:52:21 WARN InstanceBuilder: Failed to load implementation from:d
ev.ludovic.netlib.blas.JNIBLAS
23/12/04 12:52:40 WARN InstanceBuilder: Failed to load implementation from:d
ev.ludovic.netlib.lapack.JNILAPACK
```

In [7]:
```python
print('Mean Squared Error :',results.meanSquaredError)
print('Rsquared Error :',results.r2)
```

```
Mean Squared Error : 75000.14931835789
Rsquared Error : 0.34734566029177305
```