Exploring Airbnb Listing Prices in Europe

Motivation

Travel has skyrocketed as the tourism industry has been recovering post-pandemic. As a result, demand for vacation accommodations has increased. Airbnb offers a wide range of flexible accommodations, and travelers often use the service when planning their trips. As a result, Airbnb has become a staple in local economies worldwide. Pricing plays a crucial role in shaping travel patterns and destination choices, and it's important to understand the factors that influence Airbnb listing prices.

This project aims to identify variables impact Airbnb rates in well-known European locations. By using machine learning approaches, this project can serve as a tool to predict Airbnb prices in different locations, which can be used by hosts when pricing their listings or tourism stakeholders for understanding local trends.

Exploratory Data Analysis

Reading in Airbnb Price Data

The data set is from Gyódi & Łukasz (2021). A total of 20 csv are provided for popular cities in Europe. Each city has 2 csv files with information on Airbnb prices, one file for weekday prices and one file for weekend prices. I'll start by reading in all 20 csv files. All of the data sets have the same schema, which is as follows:

- **realSum:** the full price of accommodation for two people and two nights in EUR
- **room_type**: the type of the accommodation
- **room_shared:** dummy variable for shared rooms
- **room_private:** dummy variable for private rooms
- **person_capacity:** the maximum number of guests

- **host_is_superhost:** dummy variable for superhost status
- **multi:** dummy variable if the listing belongs to hosts with 2-4 offers
- **biz:** dummy variable if the listing belongs to hosts with more than 4 offers
- **cleanliness_rating:** cleanliness rating
- **guest_satisfaction_overall:** overall rating of the listing
- **bedrooms:** number of bedrooms (0 for studios)
- **dist:** distance from city center in km
- **metro_dist:** distance from nearest metro station in km
- **attr_index:** attraction index of the listing location
- **attr_index_norm:** normalized attraction index (0-100)
- **rest_index:** restaurant index of the listing location
- **attr_index_norm:** normalized restaurant index (0-100)
- **lng:** longitude of the listing location
- **lat:** latitude of the listing location

realSum is the price of each listing, which is what I am interested in predicting. I'll begin by reading the data and previewing one of the data frames.

```
library(reticulate)
```

```
Warning: package 'reticulate' was built under R version 4.1.2
```

```
# create a new environment
virtualenv_create("r-reticulate")
```

```
virtualenv: r-reticulate
```

```
# indicate that we want to use a specific virtualenv
use_virtualenv("r-reticulate")

py_install("scipy", envname = "r-reticulate")
```

```
Using virtual environment 'r-reticulate' ...
```

```
+ '/Users/kirstenjohnson/.virtualenvs/r-reticulate/bin/python' -m pip install --upgrade --no
```

```python
# Import libraries
import numpy as np
import re
import pandas as pd
import os
import seaborn as sns
from statistics import median
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
```

```python
# Import data
amsterdam_weekdays = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/amsterdam_weekdays.csv
amsterdam_weekends = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/amsterdam_weekends.csv
athens_weekdays = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/athens_weekdays.csv')
athens_weekends = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/athens_weekends.csv')
barcelona_weekdays = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/barcelona_weekdays.csv
barcelona_weekends = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/barcelona_weekends.csv
berlin_weekdays = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/berlin_weekdays.csv')
berlin_weekends = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/berlin_weekends.csv')
budapest_weekdays = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/budapest_weekdays.csv')
budapest_weekends = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/budapest_weekends.csv')
lisbon_weekdays = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/lisbon_weekdays.csv')
lisbon_weekends = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/lisbon_weekends.csv')
london_weekdays = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/london_weekdays.csv')
london_weekends = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/london_weekends.csv')
paris_weekdays = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/paris_weekdays.csv')
paris_weekends = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/paris_weekends.csv')
rome_weekdays = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/rome_weekdays.csv')
rome_weekends = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/rome_weekends.csv')
vienna_weekdays = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/vienna_weekdays.csv')
vienna_weekends = pd.read_csv('/Volumes/Extreme Pro/AirbnbPrices/vienna_weekends.csv')
amsterdam_weekdays.head
```

```
<bound method NDFrame.head of       Unnamed: 0      realSum  ...      lng       lat
0              0   194.033698  ...  4.90569  52.41772
1              1   344.245776  ...  4.90005  52.37432
2              2   264.101422  ...  4.97512  52.36103
```

```
3              3   433.529398  ...  4.89417  52.37663
4              4   485.552926  ...  4.90051  52.37508
...          ...          ...  ...      ...       ...
1098        1098  2486.115342  ...  4.85869  52.37677
1099        1099   233.637194  ...  4.83611  52.34910
1100        1100   317.062311  ...  4.88897  52.37798
1101        1101  1812.855904  ...  4.90688  52.35794
1102        1102   258.008577  ...  4.89295  52.37575

[1103 rows x 20 columns]>
```

Combining Data Frames

Since each city data frame has the same schema, I can combine them all into one data frame. But first, I will add a column to each city data frame that denotes the European city and which part of the week the prices reflect.

```
# Add column for city name and day part for each city data frame
amsterdam_weekdays['city'] = "amsterdam"
amsterdam_weekdays['week_part'] = "weekday"
amsterdam_weekends['city'] = "amsterdam"
amsterdam_weekends['week_part'] = "weekend"

athens_weekdays['city'] = "athens"
athens_weekdays['week_part'] = "weekday"
athens_weekends['city'] = "athens"
athens_weekends['week_part'] = "weekend"

barcelona_weekdays['city'] = "barcelona"
barcelona_weekdays['week_part'] = "weekday"
barcelona_weekends['city'] = "barcelona"
barcelona_weekends['week_part'] = "weekend"

berlin_weekdays['city'] = "berlin"
berlin_weekdays['week_part'] = "weekday"
berlin_weekends['city'] = "berlin"
berlin_weekends['week_part'] = "weekend"

berlin_weekdays['city'] = "berlin"
berlin_weekdays['week_part'] = "weekday"
berlin_weekends['city'] = "berlin"
```

```
berlin_weekends['week_part'] = "weekend"

budapest_weekdays['city'] = "budapest"
budapest_weekdays['week_part'] = "weekday"
budapest_weekends['city'] = "budapest"
budapest_weekends['week_part'] = "weekend"

lisbon_weekdays['city'] = "lisbon"
lisbon_weekdays['week_part'] = "weekday"
lisbon_weekends['city'] = "lisbon"
lisbon_weekends['week_part'] = "weekend"

london_weekdays['city'] = "london"
london_weekdays['week_part'] = "weekday"
london_weekends['city'] = "london"
london_weekends['week_part'] = "weekend"

paris_weekdays['city'] = "paris"
paris_weekdays['week_part'] = "weekday"
paris_weekends['city'] = "paris"
paris_weekends['week_part'] = "weekend"

rome_weekdays['city'] = "rome"
rome_weekdays['week_part'] = "weekday"
rome_weekends['city'] = "rome"
rome_weekends['week_part'] = "weekend"

vienna_weekdays['city'] = "vienna"
vienna_weekdays['week_part'] = "weekday"
vienna_weekends['city'] = "vienna"
vienna_weekends['week_part'] = "weekend"
```

Now I can combine all of the columns into one data frame called *airbnb_prices*.

```
# Combine all files into one dataframe
airbnb_prices = pd.concat([amsterdam_weekdays,amsterdam_weekends,athens_weekdays,athens_we
vienna_weekends], ignore_index=True)

# Rename ID column
airbnb_prices.rename( columns={'Unnamed: 0':'id'}, inplace=True )
airbnb_prices.columns
```

```
Index(['id', 'realSum', 'room_type', 'room_shared', 'room_private',
       'person_capacity', 'host_is_superhost', 'multi', 'biz',
       'cleanliness_rating', 'guest_satisfaction_overall', 'bedrooms', 'dist',
       'metro_dist', 'attr_index', 'attr_index_norm', 'rest_index',
       'rest_index_norm', 'lng', 'lat', 'city', 'week_part'],
      dtype='object')
```

Summarizing and Visualizing

To get a quick glance of the data structure, I want to view the column names, data types, and number of rows and columns in my new data frame *airbnb_prices*.

```
airbnb_prices.columns
```

```
Index(['id', 'realSum', 'room_type', 'room_shared', 'room_private',
       'person_capacity', 'host_is_superhost', 'multi', 'biz',
       'cleanliness_rating', 'guest_satisfaction_overall', 'bedrooms', 'dist',
       'metro_dist', 'attr_index', 'attr_index_norm', 'rest_index',
       'rest_index_norm', 'lng', 'lat', 'city', 'week_part'],
      dtype='object')
```

```
df_rows = airbnb_prices.shape[0]
df_columns = len(airbnb_prices.columns)
print("Number of columns:", df_columns)
```

```
Number of columns: 22
```

```
print("Number of rows:", df_rows)
```

```
Number of rows: 51707
```

```
# List data frame column names and their data type
airbnb_prices.dtypes
```

```
id                            int64
realSum                     float64
room_type                    object
```

```
room_shared                       bool
room_private                      bool
person_capacity                float64
host_is_superhost                 bool
multi                            int64
biz                              int64
cleanliness_rating             float64
guest_satisfaction_overall     float64
bedrooms                         int64
dist                           float64
metro_dist                     float64
attr_index                     float64
attr_index_norm                float64
rest_index                     float64
rest_index_norm                float64
lng                            float64
lat                            float64
city                            object
week_part                       object
dtype: object
```

I'm curious what the spatial distribution of the listings looks like at a glance, so I'll view them on a map. I'll add a clustering feature to be able to get a numerical sense of the number of listings in a particular location.

```
airbnb_prices.to_csv('/Volumes/Extreme Pro/AirbnbPrices/airbnb_prices.csv')
```

```
### Map European Airbnb Listings
library(tidyverse)
```

```
Warning: package 'tidyverse' was built under R version 4.1.2
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --
v ggplot2 3.4.0      v purrr   0.3.4
v tibble  3.1.8      v dplyr   1.0.10
v tidyr   1.2.1      v stringr 1.5.0
v readr   2.1.2      v forcats 0.5.1
```

```
Warning: package 'ggplot2' was built under R version 4.1.2
```

```
Warning: package 'tibble' was built under R version 4.1.2
```

```
Warning: package 'tidyr' was built under R version 4.1.2

Warning: package 'readr' was built under R version 4.1.2

Warning: package 'dplyr' was built under R version 4.1.2

Warning: package 'stringr' was built under R version 4.1.2

-- Conflicts --------------------------------------- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

```
library(dbplyr)
```

```
Attaching package: 'dbplyr'

The following objects are masked from 'package:dplyr':

    ident, sql
```

```
library(devtools)
```

```
Warning: package 'devtools' was built under R version 4.1.2

Loading required package: usethis

Warning: package 'usethis' was built under R version 4.1.2
```

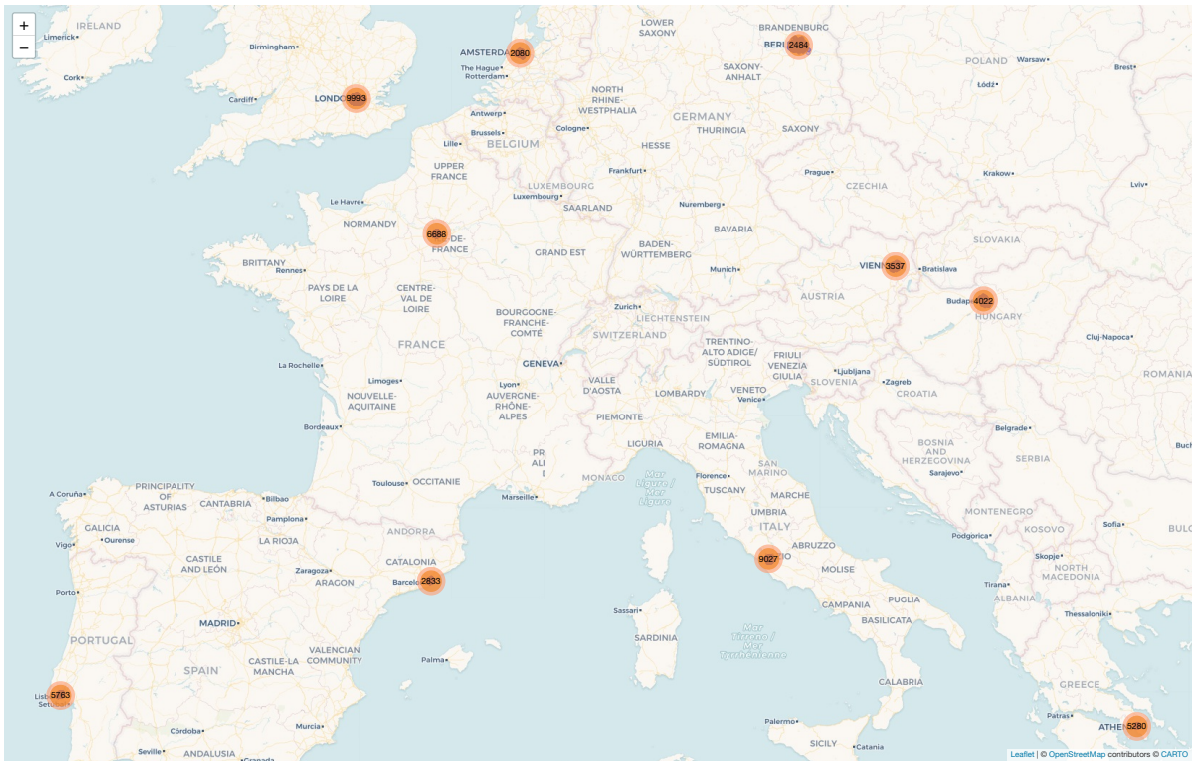
```
library(ggplot2)
library(rmarkdown)
```

```
Warning: package 'rmarkdown' was built under R version 4.1.2
```

```
library(leaflet)
library(ggthemes)
library(leaflet)
airbnb_locations.gps<-read_csv("/Volumes/Extreme Pro/AirbnbPrices/airbnb_prices.csv")
```

```
New names:
Rows: 51707 Columns: 23
-- Column specification
-------------------------------------------------------- Delimiter: "," chr
(3): room_type, city, week_part dbl (17): ...1, id, realSum, person_capacity,
multi, biz, cleanliness_rating... lgl (3): room_shared, room_private,
host_is_superhost
i Use `spec()` to retrieve the full column specification for this data. i
Specify the column types or set `show_col_types = FALSE` to quiet this message.
* `` -> `...1`
```

```
airbnb_locations.gps <- airbnb_locations.gps %>%
  select(lng, lat)

leaflet(airbnb_locations.gps) %>%
  addProviderTiles(providers$CartoDB.Voyager) %>%
  addCircles()%>%
  addMarkers(
  clusterOptions = markerClusterOptions())
```



Now it's time to do visualize the individual columns using different charts. The id field is just a numeric index of the listings and has no predictive value, so I will remove that from my data frame.

```
airbnb_prices = airbnb_prices[['realSum', 'room_type', 'room_shared', 'room_private',
       'person_capacity', 'host_is_superhost', 'multi', 'biz',
       'cleanliness_rating', 'guest_satisfaction_overall', 'bedrooms', 'dist',
       'metro_dist', 'attr_index', 'attr_index_norm', 'rest_index',
       'rest_index_norm', 'lng', 'lat', 'city', 'week_part']]
airbnb_prices.columns
```

```
Index(['realSum', 'room_type', 'room_shared', 'room_private',
       'person_capacity', 'host_is_superhost', 'multi', 'biz',
       'cleanliness_rating', 'guest_satisfaction_overall', 'bedrooms', 'dist',
       'metro_dist', 'attr_index', 'attr_index_norm', 'rest_index',
       'rest_index_norm', 'lng', 'lat', 'city', 'week_part'],
      dtype='object')
```

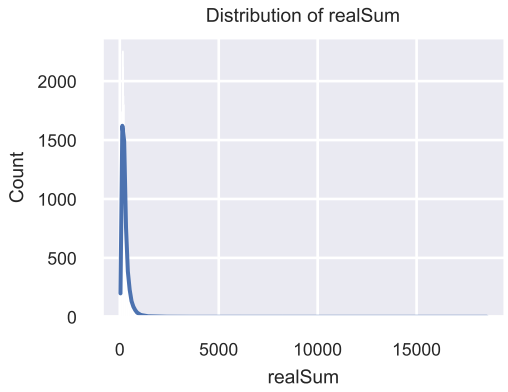
```
import matplotlib.pyplot as plt
import seaborn as sns

airbnb_prices = airbnb_prices[['realSum', 'room_type', 'room_shared', 'room_private',
       'person_capacity', 'host_is_superhost', 'multi', 'biz',
       'cleanliness_rating', 'guest_satisfaction_overall', 'bedrooms', 'dist',
       'metro_dist', 'attr_index', 'attr_index_norm', 'rest_index',
       'rest_index_norm', 'lng', 'lat', 'city', 'week_part']]
# Visualizing the individual distribution of each variable in 'airbnb_prices'


for column in airbnb_prices.columns:
    plt.figure(figsize=(2.8, 2.2))
    sns.set(font_scale=.6)
    # Check the data type of the column
    if airbnb_prices[column].dtype == 'object':
        # Create bar chart for categorical variable
        sns.countplot(data=airbnb_prices, x=column)
        plt.title(f'Distribution of {column}')
        plt.xlabel(column)
        plt.ylabel('Count')
        plt.xticks(rotation=45)

    else:
        # Create Histogram for numeric variable
        sns.histplot(data=airbnb_prices, x=column, kde=True)
        plt.title(f'Distribution of {column}')
        plt.xlabel(column)
        plt.ylabel('Count')
    plt.tight_layout()

    plt.show()
```

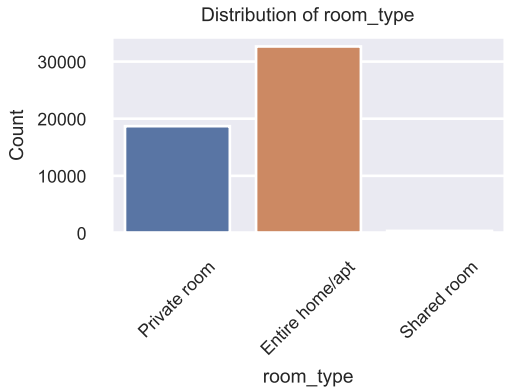
```
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='realSum', ylabel='Count'>
Text(0.5, 1.0, 'Distribution of realSum')
Text(0.5, 0, 'realSum')
Text(0, 0.5, 'Count')
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='room_type', ylabel='count'>
Text(0.5, 1.0, 'Distribution of room_type')
```

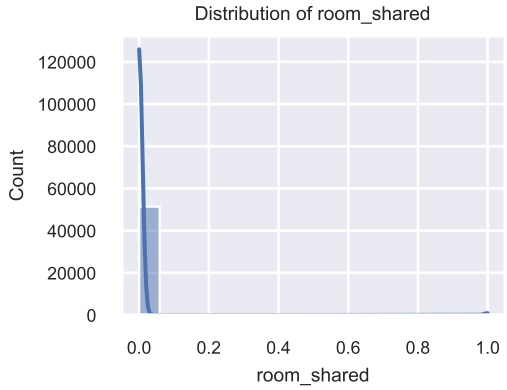
```
Text(0.5, 0, 'room_type')
Text(0, 0.5, 'Count')
(array([0, 1, 2]), [Text(0, 0, 'Private room'), Text(1, 0, 'Entire home/apt'), Text(2, 0, 'Sh
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='room_shared', ylabel='Count'>
Text(0.5, 1.0, 'Distribution of room_shared')
Text(0.5, 0, 'room_shared')
Text(0, 0.5, 'Count')
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='room_private', ylabel='Count'>
Text(0.5, 1.0, 'Distribution of room_private')
Text(0.5, 0, 'room_private')
Text(0, 0.5, 'Count')
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='person_capacity', ylabel='Count'>
Text(0.5, 1.0, 'Distribution of person_capacity')
Text(0.5, 0, 'person_capacity')
Text(0, 0.5, 'Count')
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='host_is_superhost', ylabel='Count'>
Text(0.5, 1.0, 'Distribution of host_is_superhost')
Text(0.5, 0, 'host_is_superhost')
Text(0, 0.5, 'Count')
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='multi', ylabel='Count'>
Text(0.5, 1.0, 'Distribution of multi')
Text(0.5, 0, 'multi')
Text(0, 0.5, 'Count')
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='biz', ylabel='Count'>
Text(0.5, 1.0, 'Distribution of biz')
Text(0.5, 0, 'biz')
Text(0, 0.5, 'Count')
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='cleanliness_rating', ylabel='Count'>
Text(0.5, 1.0, 'Distribution of cleanliness_rating')
Text(0.5, 0, 'cleanliness_rating')
Text(0, 0.5, 'Count')
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='guest_satisfaction_overall', ylabel='Count'>
Text(0.5, 1.0, 'Distribution of guest_satisfaction_overall')
Text(0.5, 0, 'guest_satisfaction_overall')
Text(0, 0.5, 'Count')
```

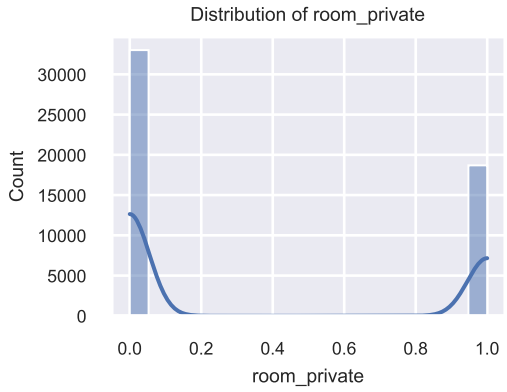
```
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='bedrooms', ylabel='Count'>
Text(0.5, 1.0, 'Distribution of bedrooms')
Text(0.5, 0, 'bedrooms')
Text(0, 0.5, 'Count')
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='dist', ylabel='Count'>
Text(0.5, 1.0, 'Distribution of dist')
Text(0.5, 0, 'dist')
Text(0, 0.5, 'Count')
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='metro_dist', ylabel='Count'>
Text(0.5, 1.0, 'Distribution of metro_dist')
Text(0.5, 0, 'metro_dist')
Text(0, 0.5, 'Count')
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='attr_index', ylabel='Count'>
Text(0.5, 1.0, 'Distribution of attr_index')
Text(0.5, 0, 'attr_index')
Text(0, 0.5, 'Count')
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='attr_index_norm', ylabel='Count'>
Text(0.5, 1.0, 'Distribution of attr_index_norm')
Text(0.5, 0, 'attr_index_norm')
Text(0, 0.5, 'Count')
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='rest_index', ylabel='Count'>
Text(0.5, 1.0, 'Distribution of rest_index')
Text(0.5, 0, 'rest_index')
Text(0, 0.5, 'Count')
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='rest_index_norm', ylabel='Count'>
Text(0.5, 1.0, 'Distribution of rest_index_norm')
Text(0.5, 0, 'rest_index_norm')
Text(0, 0.5, 'Count')
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='lng', ylabel='Count'>
Text(0.5, 1.0, 'Distribution of lng')
Text(0.5, 0, 'lng')
Text(0, 0.5, 'Count')
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='lat', ylabel='Count'>
Text(0.5, 1.0, 'Distribution of lat')
```

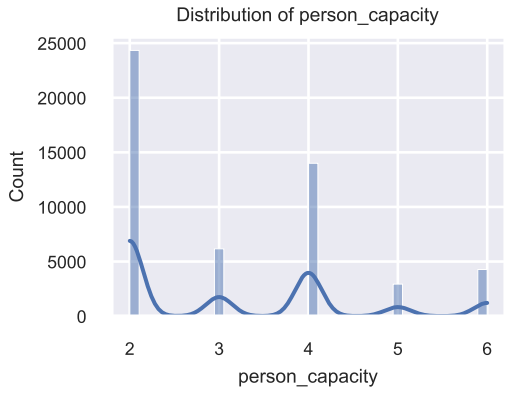
```
Text(0.5, 0, 'lat')
Text(0, 0.5, 'Count')
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='city', ylabel='count'>
Text(0.5, 1.0, 'Distribution of city')
Text(0.5, 0, 'city')
Text(0, 0.5, 'Count')
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), [Text(0, 0, 'amsterdam'), Text(1, 0, 'athens'), Text
<Figure size 560x440 with 0 Axes>
<Axes: xlabel='week_part', ylabel='count'>
Text(0.5, 1.0, 'Distribution of week_part')
Text(0.5, 0, 'week_part')
Text(0, 0.5, 'Count')
(array([0, 1]), [Text(0, 0, 'weekday'), Text(1, 0, 'weekend')])

<__array_function__ internals>:200: RuntimeWarning: Converting input from bool to <class 'nu
<__array_function__ internals>:200: RuntimeWarning: Converting input from bool to <class 'nu
<__array_function__ internals>:200: RuntimeWarning: Converting input from bool to <class 'nu
<string>:2: RuntimeWarning: More than 20 figures have been opened. Figures created through t
```

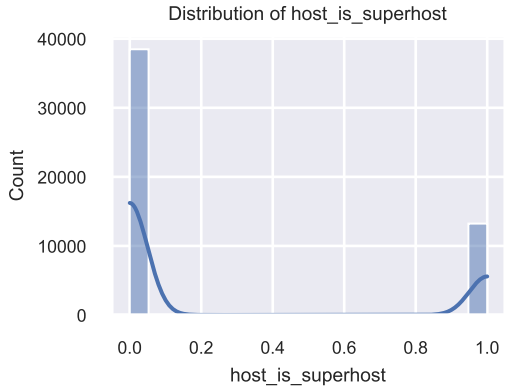


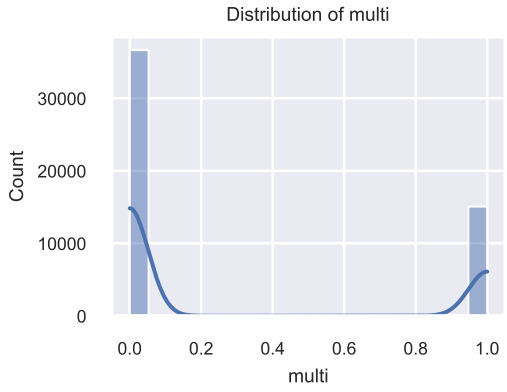

Distribution of room_type
Count
30000
20000
10000
0
Private room
Entire home/apt
Shared room
room_type

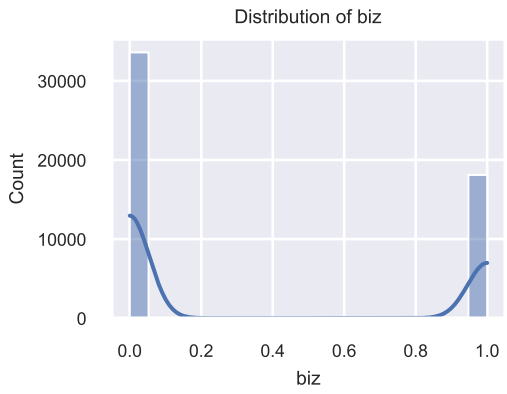

Distribution of room_shared
Count
120000
100000
80000
60000
40000
20000
0
0.0
0.2
0.4
0.6
0.8
1.0
room_shared

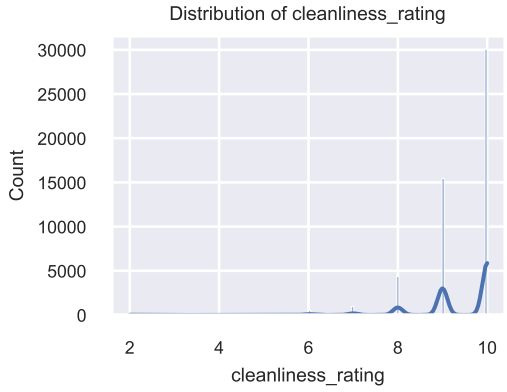

Distribution of room_private
Count
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0
room_private

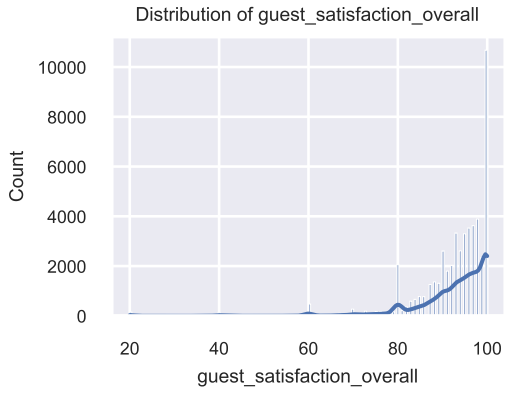

Distribution of person_capacity
Count
25000
20000
15000
10000
5000
0
2
3
4
5
6
person_capacity

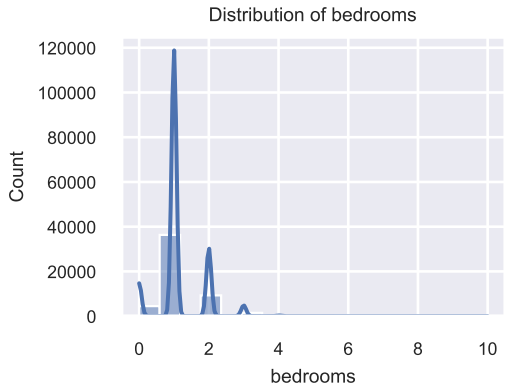

Distribution of host_is_superhost
Count
40000
30000
20000
10000
0
0.0
0.2
0.4
0.6
0.8
1.0
host_is_superhost

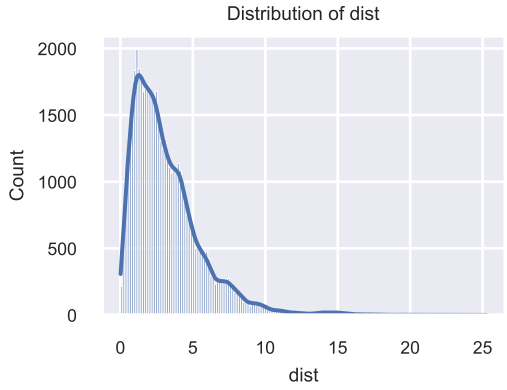

Distribution of multi
Count
30000
20000
10000
0
0.0
0.2
0.4
0.6
0.8
1.0
multi

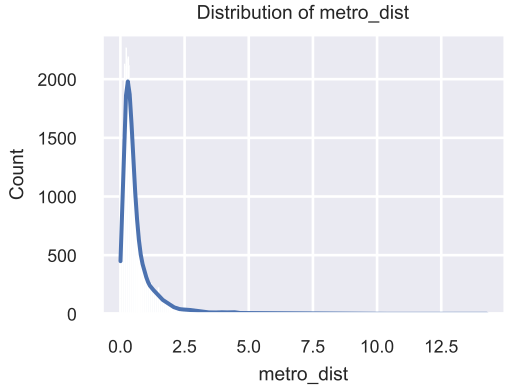

Distribution of biz
Count
30000
20000
10000
0
0.0
0.2
0.4
0.6
0.8
1.0
biz

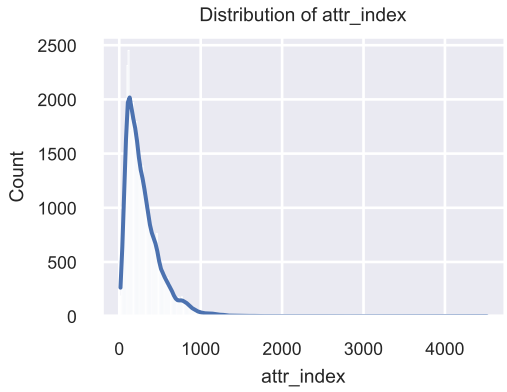

Distribution of cleanliness_rating
Count
30000
25000
20000
15000
10000
5000
0
2
4
6
8
10
cleanliness_rating

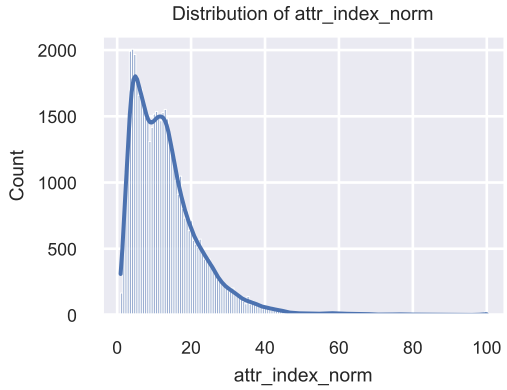

Distribution of guest_satisfaction_overall
Count
10000
8000
6000
4000
2000
0
20
40
60
80
100
guest_satisfaction_overall

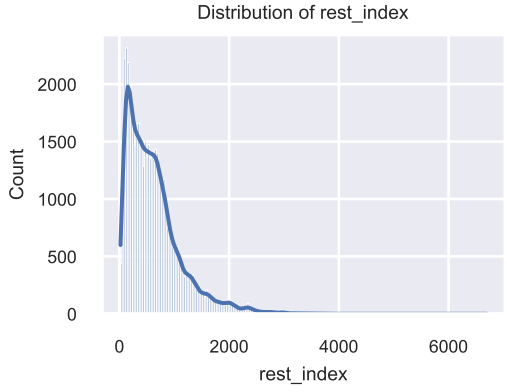

Distribution of bedrooms
120000
100000
80000
60000
40000
20000
0
Count
0
2
4
6
8
10
bedrooms

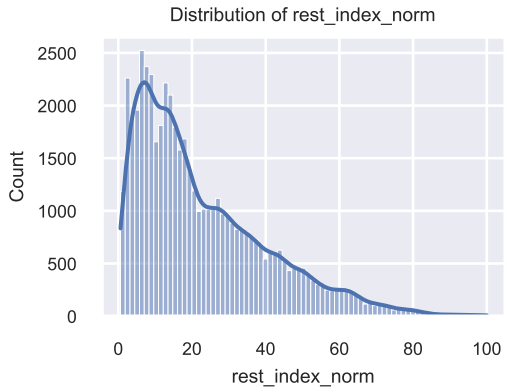

Distribution of dist
2000
1500
1000
500
0
Count
0
5
10
15
20
25
dist

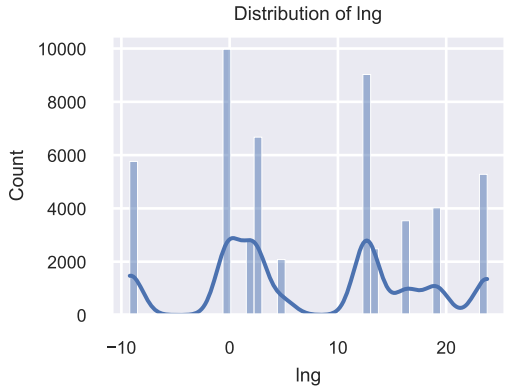

Distribution of metro_dist
2000
1500
1000
500
0
Count
0.0
2.5
5.0
7.5
10.0
12.5
metro_dist

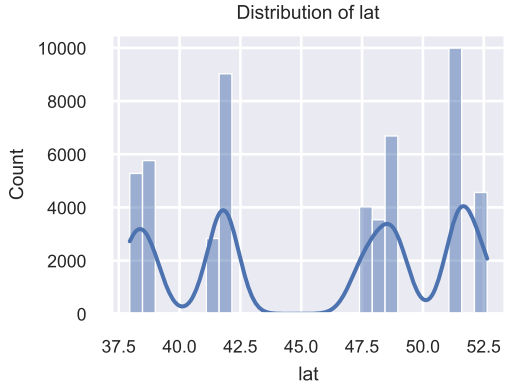

Distribution of attr_index
Count
2500
2000
1500
1000
500
0
0
1000
2000
3000
4000
attr_index

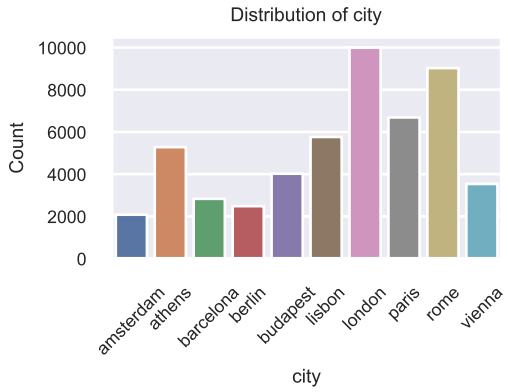

Distribution of attr_index_norm
Count
2000
1500
1000
500
0
0
20
40
60
80
100
attr_index_norm

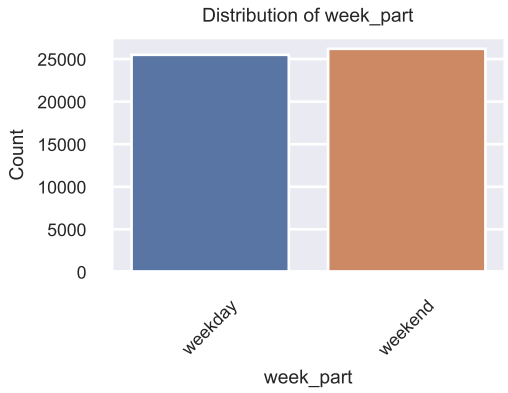

Distribution of rest_index
Count
2000
1500
1000
500
0
0
2000
4000
6000
rest_index


Distribution of rest_index_norm
Count
2500
2000
1500
1000
500
0
0
20
40
60
80
100
rest_index_norm


Distribution of lng
Count
10000
8000
6000
4000
2000
0
−10
0
10
20
lng


Distribution of lat
Count
10000
8000
6000
4000
2000
0
37.5
40.0
42.5
45.0
47.5
50.0
52.5
lat





Many of the variables are skewed either negatively or positively. Since few of my variables are normally distributed, I will need to keep skewness in mind when selecting models, as skewed distributions violate some model assumptions.

Recoding Variables

Some of the variables are categorical, such as city, week_part, and room_type. I will use label encoder to transform all categorical columns to numeric.

```
from sklearn.preprocessing import LabelEncoder

# Create variable for LabelEncoder
encoder = LabelEncoder()

# If a column is categorical, make a numeric by using LabelEcoder
```

```
for column in airbnb_prices.columns:
    if airbnb_prices[column].dtype == 'object':
        airbnb_prices[column] = encoder.fit_transform(airbnb_prices[column])

airbnb_prices.describe(include='all')
```

```
             realSum     room_type  ...          city     week_part
count   51707.000000  51707.000000  ...  51707.000000  51707.000000
unique           NaN           NaN  ...           NaN           NaN
top              NaN           NaN  ...           NaN           NaN
freq             NaN           NaN  ...           NaN           NaN
mean      279.879591      0.375674  ...      5.301487      0.506837
std       327.948386      0.498703  ...      2.586137      0.499958
min        34.779339      0.000000  ...      0.000000      0.000000
25%       148.752174      0.000000  ...      4.000000      0.000000
50%       211.343089      0.000000  ...      6.000000      1.000000
75%       319.694287      1.000000  ...      7.000000      1.000000
max     18545.450280      2.000000  ...      9.000000      1.000000

[11 rows x 21 columns]
```

Train Test Split

Before moving forward with additional EDA, I will split my data in order to avoid potential data leakage. I will reserve 70% for training and 30% for testing.

```
from sklearn.model_selection import train_test_split
# Create data frames for predictors and outcome variables


# Perform train-validation-test split
# Set aside 20% of all data for test set
training_val, test_df = train_test_split(airbnb_prices, test_size=0.2, random_state=25)
# Set aside 20% of training data for validation
training_df, val_df = train_test_split(training_val, test_size=0.2, random_state=25)

# Print the shapes of the resulting datasets
print("Count training records:", training_df.shape[0])
```

```
Count training records: 33092
```

```
print("Count validation records:", val_df.shape[0])
```

```
Count validation records: 8273
```

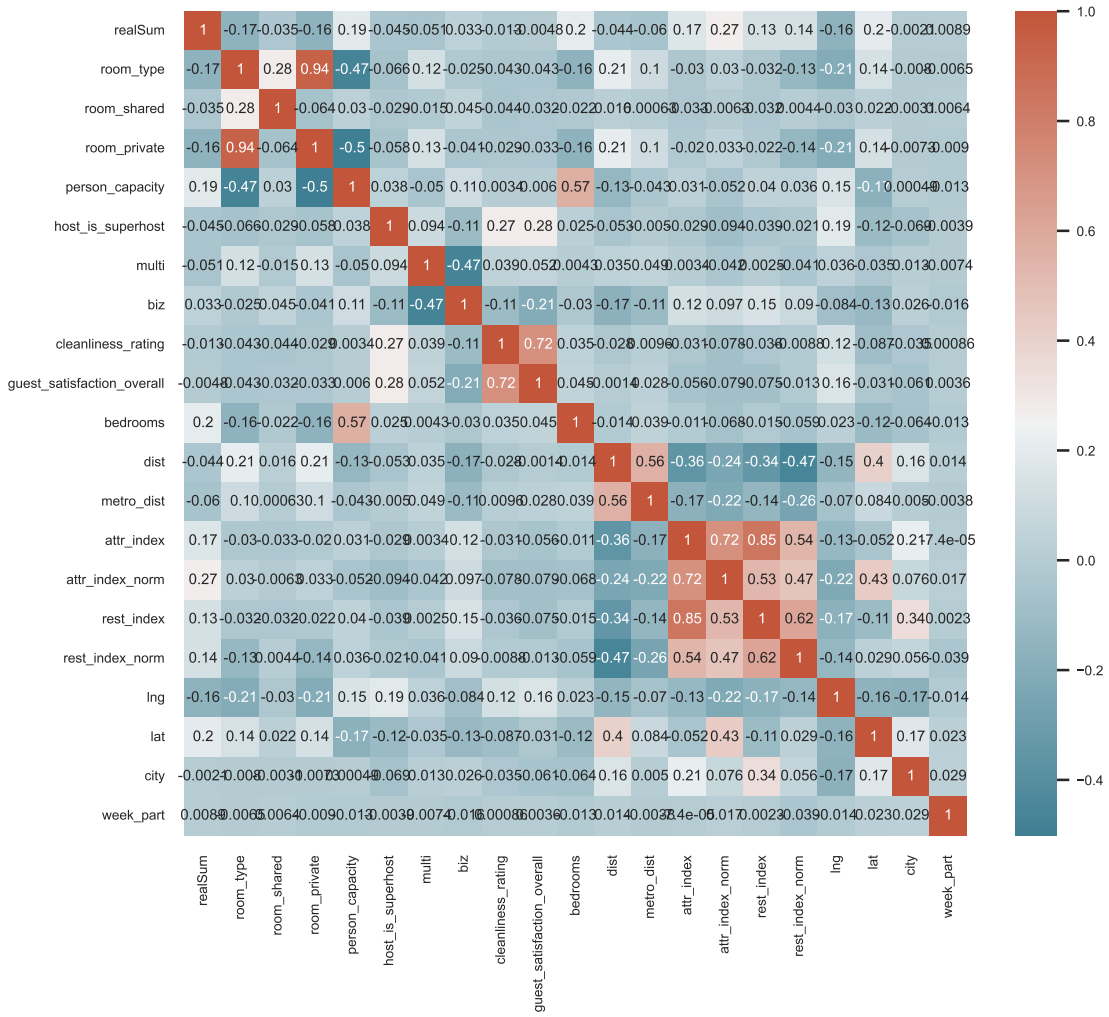
```
print("Count of test records:", test_df.shape[0])
```

```
Count of test records: 10342
```

Correlation Matrix

Now that my data is split up, I'll create a correlation matrix of my variables using the training set.

```
# Create correlation matrix using training set
corr = training_df.corr()
plt.figure(figsize=(8, 7))
sns.set(font_scale=.6)
heatmap = sns.heatmap(corr,annot=True, cmap=sns.diverging_palette(220, 20, as_cmap=True))

plt.tight_layout()
plt.show()
```



A few things to note from this matrix:

- Most variables look to have negative correlation with realSum, the outcome variable.
- There is high correlation between a few variables:
  - *room_type* and *room_private*
  - *metro_distance, attr_index, atr_index_norm,* and *rest_index* are highly correlated.
  - *metro_distance* and *distance*

I will remove the *room_type* variable and *attr_index* from the data frame. When exploring their definitions with other variables, both of these attributes look to be accounted for in other attributes.

Evaluation Metric

As mentioned, my goal is to predict the price attribute *realSum* in order to give model users insight into Airbnb price trends in the area. I am less concerned with my model being precise and more concerned with reducing large mistakes. Due to its relevance to the research objective, the root mean squared error (RMSE) is what I will use for the model evaluation metric. RMSE measures the average prediction error and penalizes large errors between the expected and actual values by giving greater errors a higher weight.

Data Preprocessing

Before I begin fitting my models, there are a few additional steps that need to be taken. First, I will check for missing values in my data frame.

```
# Display the count of missing values in each column
print(training_df.isnull().sum())
```

```
realSum                     0
room_type                   0
room_shared                 0
room_private                0
person_capacity             0
host_is_superhost           0
multi                       0
biz                         0
cleanliness_rating          0
guest_satisfaction_overall  0
bedrooms                    0
dist                        0
metro_dist                  0
attr_index                  0
attr_index_norm             0
rest_index                  0
rest_index_norm             0
lng                         0
```

```
lat                           0
city                          0
week_part                     0
dtype: int64
```

Scaling Variables

Since I am aiming to predict prices, a continuous variable, I have regression problem on my hands. I plan to use Ridge regression for one of my models, so I will scale the data. I will use MinMax Scaler, as my data are not normally distributed and has a significant amount of skewing. In the case of Airbnb prices across Europe, the skewing provides important context, so I want to keep the distribution true to the training data to fit a better model.

```
# Create X and y variables for training, validation, and test sets
# Remove room_type and attr_index

X_train = training_df[['room_shared', 'room_private',
       'person_capacity', 'host_is_superhost', 'multi', 'biz',
       'cleanliness_rating', 'guest_satisfaction_overall', 'bedrooms', 'dist',
       'metro_dist', 'attr_index_norm', 'rest_index',
       'rest_index_norm', 'lng', 'lat', 'city', 'week_part']]
y_train =training_df[['realSum']]

X_val = val_df[['room_shared', 'room_private',
       'person_capacity', 'host_is_superhost', 'multi', 'biz',
       'cleanliness_rating', 'guest_satisfaction_overall', 'bedrooms', 'dist',
       'metro_dist', 'attr_index_norm', 'rest_index',
       'rest_index_norm', 'lng', 'lat', 'city', 'week_part']]
y_val=val_df[['realSum']]

X_test = test_df[[ 'room_shared', 'room_private',
       'person_capacity', 'host_is_superhost', 'multi', 'biz',
       'cleanliness_rating', 'guest_satisfaction_overall', 'bedrooms', 'dist',
       'metro_dist', 'attr_index_norm', 'rest_index',
       'rest_index_norm', 'lng', 'lat', 'city', 'week_part']]
y_test=test_df[['realSum']]

from sklearn.preprocessing import MinMaxScaler

# Scaling Parameters
scaler_x = MinMaxScaler()
```

```
scaler_y = MinMaxScaler()

X_train = pd.DataFrame(scaler_x.fit_transform(X_train), columns=X_train.columns)
y_train = pd.DataFrame(scaler_y.fit_transform(y_train), columns=y_train.columns)
X_val = pd.DataFrame(scaler_x.transform(X_val), columns=X_val.columns)
y_val = pd.DataFrame(scaler_y.transform(y_val), columns=y_val.columns)
X_test = pd.DataFrame(scaler_x.transform(X_test), columns=X_test.columns)
y_test = pd.DataFrame(scaler_y.transform(y_test), columns=y_test.columns)
```

Multiple Linear Regression

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Fit multiple regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

```
LinearRegression()
```

```
y_pred = model.predict(X_val)
y_pred_train = model.predict(X_train)


# Calculate RMSE on the validation set and training set
rmse_val = mean_squared_error(y_val, y_pred, squared=False)
rmse_train = mean_squared_error(y_train, y_pred_train, squared=False)
print("Multiple Linear Regression RMSE Training Set:", rmse_train)
```

```
Multiple Linear Regression RMSE Training Set: 0.0165018274578235
```

```
print("Multiple Linear Regression RMSE Validation Set:", rmse_val)
```

```
Multiple Linear Regression RMSE Validation Set: 0.017267288081732432
```

Ridge Regression Model

```
# Grid for ridge regression
param_grid = {'alpha': [0.1, 1.0, 10.0]}

# Create the Ridge regression model
ridge = Ridge()

# Create the grid search
# Since RMSE is a measure of error and I want to minimize error,
#I will use the negated version
ridge_grid_search = GridSearchCV(estimator=ridge, param_grid=param_grid,
scoring='neg_root_mean_squared_error', cv=5)

# Fit the grid search to the training set
ridge_grid_search.fit(X_train, y_train)

# Get the best Ridge model and best hyper parameter values
```

```
GridSearchCV(cv=5, estimator=Ridge(), param_grid={'alpha': [0.1, 1.0, 10.0]},
             scoring='neg_root_mean_squared_error')
```

```
best_model = ridge_grid_search.best_estimator_
best_alpha = ridge_grid_search.best_params_['alpha']

# Predict using the best model on the validation set and training set
y_pred = best_model.predict(X_val)
y_pred_train = best_model.predict(X_train)
```

```
# Calculate RMSE on the validation set and training set
rmse_val = mean_squared_error(y_val, y_pred, squared=False)
rmse_train = mean_squared_error(y_train, y_pred_train, squared=False)


# Print the best hyperparameters and RMSE
print("Best Ridge Alpha:", best_alpha)
```

```
Best Ridge Alpha: 1.0
```

```
print("Training RMSE:", rmse_train)
```

```
Training RMSE: 0.016501860061887856
```

```
print("Validation RMSE:", rmse_val)
```

```
Validation RMSE: 0.01726866982436714
```

Random Forest

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, mean_squared_error

# Create variable for random forest model
rf = RandomForestRegressor()

# Subset 10% of training data for random forest hyperparameter tuning to save time
other_X, X_rf, other_y, Y_rf = train_test_split(X_train, y_train,
test_size=0.1, random_state=10)

# Create search grid for number of trees
rf_param_grid = {'n_estimators': [50, 100, 200, 300, 400, 500]}
# Create RMSE variable
rmse = make_scorer(mean_squared_error, squared=False)
# Do grid search
rf_grid_search = GridSearchCV(rf, rf_param_grid, cv=5, scoring=rmse)
# Fit the grid search to the training set
rf_grid_search.fit(X_rf, Y_rf.values.ravel())

# Get the best random forest model and best hyperparameter values
```

```
GridSearchCV(cv=5, estimator=RandomForestRegressor(),
             param_grid={'n_estimators': [50, 100, 200, 300, 400, 500]},
             scoring=make_scorer(mean_squared_error, squared=False))
```

```
best_model = rf_grid_search.best_estimator_
best_params = rf_grid_search.best_params_

# Predict using the best random forest model on the validation set and training set
y_pred = best_model.predict(X_val)
y_pred_train = best_model.predict(X_train)

# Calculate RMSE on the validation set and training set
rmse_val = mean_squared_error(y_val, y_pred, squared=False)
rmse_train = mean_squared_error(y_train, y_pred_train, squared=False)


# Print the best hyperparameters and RMSE
print("Best Random Forest Number of Trees:", best_params)
```

```
Best Random Forest Number of Trees: {'n_estimators': 50}
```

```
print("Random Forest Training RMSE:", rmse_train)
```

```
Random Forest Training RMSE: 0.015602127908859731
```

```
print("Random Forest Validation RMSE:", rmse_val)
```

```
Random Forest Validation RMSE: 0.01619280030366702
```

Generating Test Set Predictions

I will use all of the training data to fit a multiple linear regression model, a ridge model with an aplha of 10, and random forest model with 50 trees, as these were chosen as the optimal hyperparameters.

```
# Fit Random Forest model with optimal hyperparameters
rf = RandomForestRegressor(n_estimators= 50)
ridge = Ridge(alpha=10)
mlr = LinearRegression()
```

```
# Combine training and validation set into one training set
X_train_all = pd.concat([X_train, X_val], axis=0, ignore_index=True)
y_train_all = pd.concat([y_train, y_val], axis=0, ignore_index=True)

#Fit models to test set
mlr.fit(X_train_all, y_train_all)
```

```
LinearRegression()
```

```
ridge.fit(X_train_all, y_train_all)
```

```
Ridge(alpha=10)
```

```
rf.fit(X_train_all, y_train_all.values.ravel())
```

```
RandomForestRegressor(n_estimators=50)
```

```
y_pred_mlr = mlr.predict(X_test)
y_pred_ridge = ridge.predict(X_test)
y_pred_rf = rf.predict(X_test)

# Calculate RMSE on the validation set and training set
rmse_mlr = mean_squared_error(y_test, y_pred_mlr, squared=False)
rmse_ridge =  mean_squared_error(y_test, y_pred_ridge, squared=False)
rmse_rf =  mean_squared_error(y_test, y_pred_rf, squared=False)
# Print RMSE
print("Multiple Linear Regression Test RMSE:", rmse_mlr)
```

```
Multiple Linear Regression Test RMSE: 0.01094896576583313
```

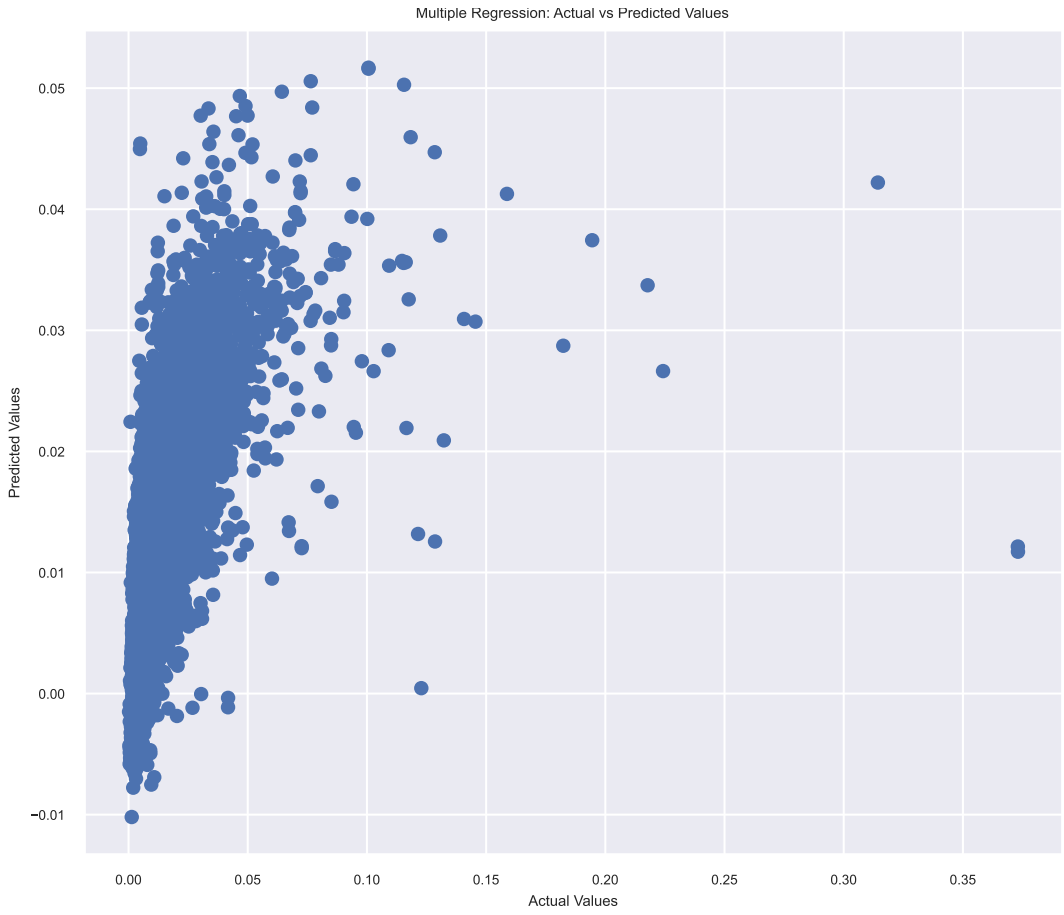
```
print("Ridge Test RMSE:", rmse_ridge)
```

```
Ridge Test RMSE: 0.01094759085436802
```

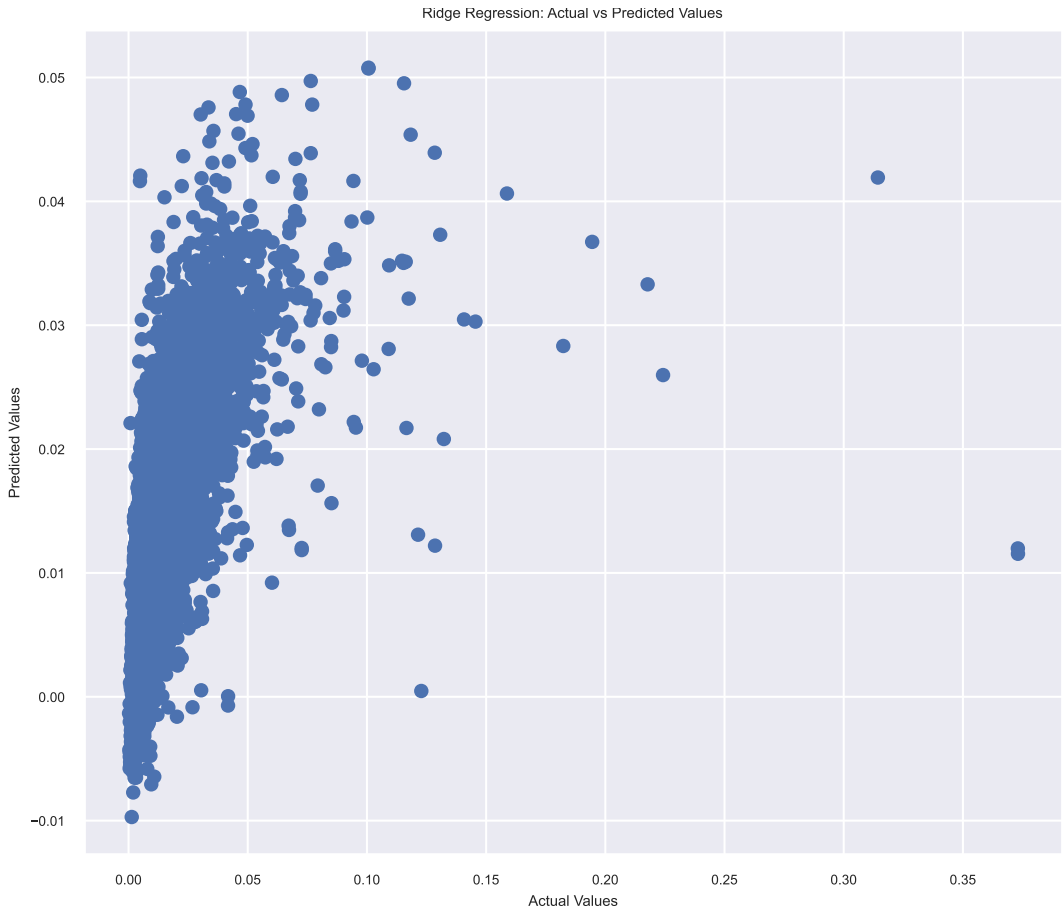
```
print("Random Forest Test RMSE:", rmse_rf)
```

```
Random Forest Test RMSE: 0.009517659906510888
```

```
# create a scatter plot of test set multiple regression predicted vs actual
plt.scatter(y_test, y_pred_mlr)

# add axis labels and a title
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Multiple Regression: Actual vs Predicted Values')
plt.show()
```




```
# create a scatter plot of test set ridge predicted vs actual
plt.scatter(y_test, y_pred_ridge)

# add axis labels and a title
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Ridge Regression: Actual vs Predicted Values')
plt.show()
```




```
# create a scatter plot of test set random forest predicted vs actual
plt.scatter(y_test, y_pred_rf)

# add axis labels and a title
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Random Forest: Actual vs Predicted Values')
plt.show()
```

Discussion

Multiple linear regression is the most general of the three models and has high bias. This model did result in under fitting when looking at the scatter plot of actual values vs. predicted values, and it performed the worst out of all of the models when comparing the training set RMSE and validation set RMSE. Interaction terms and polynomials weren't included, leading to less complexity but lower model variance.

Ridge regression performed slightly better than MLR based on the RMSE. The ridge regression and multiple linear regression scatter plots look nearly identical. However,the ridge model has more variance than the MLR model due to the inclusion of an alpha term. The ridge model is also more complex with the alpha term included, but it is still interpretable overall, as it is a

version of the linear model. The ridge model still underfit the data compared to the Random Forest model.

Random forest had the lowest RMSE on the validation set compared to multiple linear regression and ridge regression. This is likely due to random forest models being able to capture more complex, non-linear relationships between variables, resulting in a model with lower bias. While a model with low bias typically has higher variance, the number of trees decision trees can help control the trade off between bias and variance. Utilizing the grid search for hyperparameter tuning can prevent over fitting the model, and when I used it in this case, the lowest number of trees in the search grid (50) was chosen as the optimal parameter. The random forest model being the most complex does compromise it's intepretability, with ridge and multiple regression being the most interpretable of the models.

In the context of potential tourism stakeholders using this model, there are are ethical concerns to consider. There is a risk of amplifying biases that may be present in the data. For example, some of the listing prices in this Airbnb data may be due to present price discrimination in the area, leading Airbnb hosts to inflate their prices. While there may have not been any mal intent on behalf of the hosts, the inflated prices may contribute to gentrification, housing shortages, and displacement of local residents. Local goods and services may also increase, and discourage future travelers from visiting, which would negatively impact the local economy.

Someone using this model to base their listing price or to predict future prices of Airbnb's should be aware of potential inherent biases in the data before using the model to make decisions.To address these concerns, this model would be best to be reviewed by multiple local stakeholders on a region by region basis before being deployed. People who are familiar with the area should be able to notice potential harm or bias in the model results. Additionally, this should be made available to a range of stakeholders, from local Airbnb hosts, larger hotel managers, tourism government officials, or local citizens. This ensures equity and fairness by increasing visibility by different people who have different interests.

Works Cited

Gyódi, Kristóf, & Nawaro, Łukasz. (2021). Determinants of Airbnb prices in European cities: A spatial econometrics approach (Supplementary Material) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.4446043