Capstone

Task 2

Nathan Hefner

Research Question

Using the Airbnb pricing dataset, the research question will be, can a multiple linear regression model be constructed based on the dataset? Multiple Linear Regression will be used because the hypothesis is that many variables affect the price of an Airbnb. We are also using Linear Regression because the subject variable, the price of Airbnb for a night, is a continuous numeric variable. The null hypothesis is that a predictive Multiple Linear Regression model cannot be made from the research data, and the alternative hypothesis is that a predictive Multiple Linear Regression model can be constructed from the dataset with a model accuracy greater than 65%. This will give us a benchmark to aim for using R^2. This study will utilize a general linear regression model to help determine which features affect the price the most and least. We hypothesize that location and size will be the biggest contributing factors.

Data Collection

I retrieved this data from Kaggle, an open-source library of different kinds of databases. This dataset had over 74000 rows of data and 29 columns of data, including both categorical and continuous data. This dataset had more than enough data, as one challenge was finding a large enough dataset that would be sufficient for the study. I also needed the dataset to have a large variety of features to help hypothesize the price; luckily, this dataset met both of these requirements. An advantage of collecting the data this way is that we already know what features the dataset will provide, along with seeing other uses of this dataset for their own studies, showing the dataset is properly set and in usable condition. A disadvantage is that we don't know why some cells are null and will need to be imputed or dropped entirely, possibly causing feature loss, and with this dataset in particular, the price is already logged, which is helpful for normality, but may be difficult to reverse log and find the true price.

Data Extraction and Preparation

Data preparation is a pillar piece and integral for the data analytics journey, with many different steps. First step is to check the data and find out what columns to keep or drop, depending on what type of analysis and if the data type is possible to encode.

Below is a snapshot of my data.

```
column_headers = df.columns.tolist()
print(column_headers)
```

```
['id', 'log_price', 'property_type', 'room_type', 'amenities', 'accommodates', 'bathrooms', 'bed_type', 'cancellation_policy', 'cleaning_fee', 'city', 'd
escription', 'first_review', 'host_has_profile_pic', 'host_identity_verified', 'host_response_rate', 'host_since', 'instant_bookable', 'last_review', 'la
titude', 'longitude', 'name', 'neighbourhood', 'number_of_reviews', 'review_scores_rating', 'thumbnail_url', 'zipcode', 'bedrooms', 'beds']
```

Below are statistics for my numeric data; the main one we will be focusing on is the dependent variable, log_price.

```
print("\n--- Shape of the DataFrame (Rows, Columns) ---")
print(df.shape)

print("\n--- Descriptive Statistics for Numerical Columns ---")
print(df.describe())
```

```
--- Shape of the DataFrame (Rows, Columns) ---
(74111, 29)

--- Descriptive Statistics for Numerical Columns ---
                 id     log_price  accommodates     bathrooms      latitude
count  7.411100e+04  74111.000000  74111.000000  73911.000000  74111.000000
mean   1.126662e+07      4.782069      3.155146      1.235263     38.445958
std    6.081735e+06      0.717394      2.153589      0.582044      3.080167
min    3.440000e+02      0.000000      1.000000      0.000000     33.338905
25%    6.261964e+06      4.317488      2.000000      1.000000     34.127908
50%    1.225415e+07      4.709530      2.000000      1.000000     40.662138
75%    1.640226e+07      5.220356      4.000000      1.000000     40.746096
max    2.123090e+07      7.600402     16.000000      8.000000     42.390437

          longitude  number_of_reviews  review_scores_rating      bedrooms
count  74111.000000       74111.000000          57389.000000  74020.000000
mean     -92.397525          20.900568             94.067365      1.265793
std       21.705322          37.828641              7.836556      0.852143
min     -122.511500           0.000000             20.000000      0.000000
25%     -118.342374           1.000000             92.000000      1.000000
50%      -76.996965           6.000000             96.000000      1.000000
75%      -73.954660          23.000000            100.000000      1.000000
max      -70.985047         605.000000            100.000000     10.000000

              beds
count  73980.000000
mean       1.710868
std        1.254142
min        0.000000
25%        1.000000
50%        1.000000
75%        2.000000
max       18.000000
```

Now equipped with this knowledge and knowing we will be utilizing linear regression to run a model, we can drop the following columns, as they either cannot be encoded or will not help achieve our goal and just become noise within our data analysis.

```
columns_to_drop = [
    'name',
    'description',
    'first_review',
    'last_review',
    'id',
    'amenities',
    'neighbourhood']
df.drop(columns=columns_to_drop, axis=1, inplace=True)
```

With the columns dropped, we will now check for empty data cells, as linear regression cannot handle missing values.

```
empty_cells = df.isnull().sum()
print(empty_cells)
```

```
log_price                0
property_type            0
room_type                0
accommodates             0
bathrooms              196
bed_type                 0
cancellation_policy      0
cleaning_fee             0
city                     0
host_has_profile_pic   186
host_identity_verified 186
host_response_rate     17993
host_since             186
instant_bookable         0
latitude                 0
longitude                0
number_of_reviews        0
review_scores_rating   16433
thumbnail_url           8097
zipcode                  0
bedrooms                85
beds                   118
dtype: int64
```

Each of these columns will need to be handled differently to correct the missing data and to keep the integrity intact as much as possible. The first data cleaning method will be to impute the mode of the column into the missing cells. The columns we will be applying this to are: ‘host_has_profile_pic’, ‘host_identity_verified’, ‘host_since’, ‘bathrooms’, and ‘beds’. We can impute the mode into these missing values because the amount of missing values is less than 1% for each of these columns, meaning imputing them with the mode will not harm the integrity of the dataset significantly.

```
mode_cols = ['host_has_profile_pic', 'host_identity_verified', 'host_since', 'bathrooms', 'bedrooms', 'beds']
for col in mode_cols:
    mode_value = df[col].mode()[0]
    df[col].fillna(mode_value, inplace=True)
```

Next column was zipcode, and this data type is more integral and complex for our goal; henceforth, it was best to drop all the rows without a zipcode to keep the dataset as clean as possible.

```
rows_before = len(df)
print(f"DataFrame size BEFORE dropping rows: {rows_before} rows")
df.dropna(subset=['zipcode'], inplace=True)
rows_after = len(df)
rows_dropped = rows_before - rows_after
print(f"Dropped {rows_dropped} rows where 'zipcode' was missing.")
print(f"--- DataFrame size AFTER dropping rows: {rows_after} rows ---")
```

```
DataFrame size BEFORE dropping rows: 74111 rows
Dropped 966 rows where 'zipcode' was missing.
--- DataFrame size AFTER dropping rows: 73145 rows ---
```

For thumbnail_url, review_scores_rating, and host_response_rate, we had to wrangle these data columns to fit our particular needs. We dropped thumbnail_url to instead have has_thumbnail for the column, which is now a boolean column, either true or false. For review_scores_rating and host_response_rate, we did the same thing, creating a column to show if they have a response or rating, and imputed the missing data with the median statistical value. This solves both our problems of eliminating empty cells and keeps the integrity by having another column purely to show if the value was supposed to be empty or not. We do this because for these data categories, there are legitimate reasons for them to be empty, and we want the model to capture that feature.

```
df['has_thumbnail'] = df['thumbnail_url'].notnull().astype(int)
df.drop('thumbnail_url', axis=1, inplace=True)
print(df['has_thumbnail'].value_counts())
```

```
has_thumbnail
1    65048
0     8097
Name: count, dtype: int64
```

```
print(df['review_scores_rating'].describe())
```

```
count    56712.000000
mean        94.063179
std          7.814818
min         20.000000
25%         92.000000
50%         96.000000
75%        100.000000
max        100.000000
Name: review_scores_rating, dtype: float64
```

```
rating_col = 'review_scores_rating'
df['has_rating'] = df[rating_col].notnull().astype(int)
median_rating = df[rating_col].median()
df[rating_col].fillna(median_rating, inplace=True)
```

```
C:\Users\Nathan\AppData\Local\Temp\ipykernel_12132\1404176835.py:4: FutureWarnin
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work beca
as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.met
ad, to perform the operation inplace on the original object.


  df[rating_col].fillna(median_rating, inplace=True)
```

```
host_response_col = 'host_response_rate'
df[host_response_col] = df[host_response_col].astype(str)
df[host_response_col] = df[host_response_col].str.replace('%', '', regex=False)
df[host_response_col] = pd.to_numeric(df[host_response_col], errors='coerce')
```

```
df['has_response_rate'] = df[host_response_col].notnull().astype(int)
df[host_response_col].fillna(0.0, inplace=True)
```

With that, we complete our data cleaning process.

```
empty_cells = df.isnull().sum()
print(empty_cells)
```

```
log_price                0
property_type            0
room_type                0
accommodates             0
bathrooms                0
bed_type                 0
cancellation_policy      0
cleaning_fee             0
city                     0
host_has_profile_pic     0
host_identity_verified   0
host_response_rate       0
host_since               0
instant_bookable         0
latitude                 0
longitude                0
number_of_reviews        0
review_scores_rating     0
zipcode                  0
bedrooms                 0
beds                     0
has_thumbnail            0
has_rating               0
has_response_rate        0
dtype: int64
```

Analysis

The goal is to perform Multiple Linear Regression, and to do so, we need to continue wrangling the data so that it is readable and digestible for the model. This means we will need to encode the data, essentially transforming any categorical data into numeric data. We will have multiple methods to fully complete this task. The first method will be One-Hot Encoding, which turns binary values (Like True/False and Yes/No) into numeric values consisting of 1s and 0s. This is advantageous for us as it's a straightforward method that guarantees no data loss. However, we must be careful with this method because a disadvantage is that it increases dimensionality, which could lead to multicollinearity and longer computational times (Micheal).

Below is my One-Hot Encoding

```
ohe_cols = ['room_type', 'bed_type', 'cancellation_policy', 'property_type', 'city']

df = pd.get_dummies(df, columns=ohe_cols, drop_first=True)

print("Applied One-Hot Encoding to low-cardinality features.")
print(df.head())
print(f"\nNew number of columns: {df.shape[1]}")
```

```
Applied One-Hot Encoding to low-cardinality features.
   log_price  accommodates  bathrooms  cleaning_fee host_has_profile_pic  \
0   5.010635             3        1.0          True                    t
1   5.129899             7        1.0          True                    t
2   4.976734             5        1.0          True                    t
3   6.620073             4        1.0          True                    t
4   4.744932             2        1.0          True                    t

  host_identity_verified  host_response_rate  host_since instant_bookable  \
0                      t                 0.0  2012-03-26                f
1                      f               100.0  2017-06-19                t
2                      t               100.0  2016-10-25                t
3                      t                 0.0  2015-04-19                f
4                      t               100.0  2015-03-01                t

    latitude  ...  property_type_Train  property_type_Treehouse  \
0  40.696524  ...                False                    False
1  40.766115  ...                False                    False
2  40.808110  ...                False                    False
3  37.772004  ...                False                    False
4  38.925627  ...                False                    False

   property_type_Vacation home  property_type_Villa  property_type_Yurt  \
0                        False                False               False
1                        False                False               False
2                        False                False               False
3                        False                False               False
4                        False                False               False

   city_Chicago  city_DC  city_LA  city_NYC  city_SF
0         False    False    False      True    False
1         False    False    False      True    False
2         False    False    False      True    False
3         False    False    False     False     True
4         False     True    False     False    False

[5 rows x 68 columns]
```

```
binary_cols_to_map = [
    'host_has_profile_pic',
    'host_identity_verified',
    'instant_bookable'
]
binary_mapping = {'t': 1, 'f': 0}

for col in binary_cols_to_map:

    df[col] = df[col].replace(binary_mapping).fillna(0).astype(int)

print("Successfully mapped 't' to 1 and 'f' to 0 for binary features.")
```

```
Successfully mapped 't' to 1 and 'f' to 0 for binary features.
```

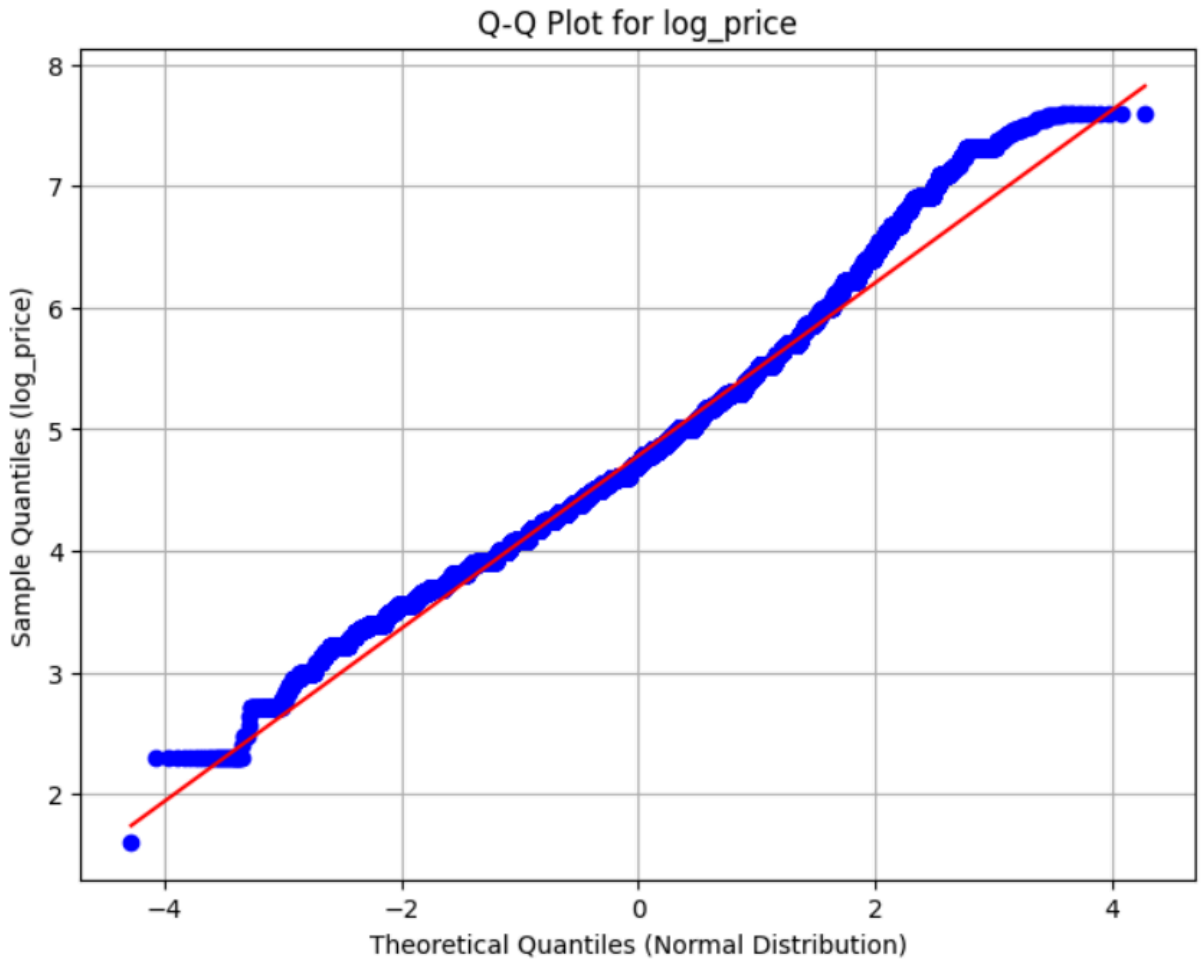
Lastly, we will need to encode zipcode, as even though it is a numeric valu,e the data type is categorical, and since the number of different values is so vas,t we will need a new method, Target Encoding. This encoding method takes the average of all log prices that share the zip code and replaces the zip code with that mean value. This will help the model learn that Airbnbs prices are affected by zip codes. An advantage of this method is that it allows us to use a high cardinality categorical feature in our model, but the disadvantage is that we lose specific zipcodes, and it can be difficult to regain them.

```
cat_col_zip = 'zipcode'
target_col = 'log_price'

zipcode_mapping = df.groupby(cat_col_zip)[target_col].mean().reset_index()

zipcode_mapping.columns = [cat_col_zip, f'{cat_col_zip}_encoded']

df = df.merge(zipcode_mapping, on=cat_col_zip, how='left')

df.drop(cat_col_zip, axis=1, inplace=True)

print(f"Target Encoding fix: Created '{cat_col_zip}_encoded' using a robust merge approach.")
print("Original 'zipcode' column dropped.")

print("\n--- Verification of New Encoded Feature ---")
print(df[f'{cat_col_zip}_encoded'].head())
```

```
Target Encoding fix: Created 'zipcode_encoded' using a robust merge approach.
Original 'zipcode' column dropped.

--- Verification of New Encoded Feature ---
0    5.009599
1    5.184469
2    4.656553
3    5.132011
4    5.030781
Name: zipcode_encoded, dtype: float64
```

After the data is fully encoded, it will now be time to split the data into a train/test split for model validation. Once done, we will run the General Multiple Linear Regression Model. Before that, we must know if the assumption of linearity is met. Usually, with a variable like price, it wouldn't be met, because it is not evenly distributed, but you can log the price to meet this assumption, and luckily for us, the price already is logged. Below is a Q-Q plot to further show linearity.

```
import scipy.stats as stats
target_col = 'log_price'

plt.figure(figsize=(8, 6))
stats.probplot(df[target_col], dist="norm", plot=plt)

plt.title(f'Q-Q Plot for {target_col}')
plt.xlabel('Theoretical Quantiles (Normal Distribution)')
plt.ylabel('Sample Quantiles (log_price)')
plt.grid(True)
plt.show()
```



Below is the model and the results.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

linear_model = LinearRegression()

linear_model.fit(X_train, y_train)
```

```
LinearRegression
LinearRegression()
```

```
y_pred = linear_model.predict(X_test)

r2 = r2_score(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)

print("Linear Regression Performance (on Test Set)")
print(f"R-squared (R²): {r2:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
```

```
Linear Regression Performance (on Test Set)
R-squared (R²): 0.5791
Root Mean Squared Error (RMSE): 0.4631
```

This is advantageous for us because MLR models can strongly predict the influence of predictive variables. Meaning it can tell if certain features have a bigger impact on others. This is exactly what we are looking for, as we hypothesize that location and size will be the biggest predictors for the price. A disadvantage, although, is the linear assumption, which will not capture real-world complex data, instead just forecasting past trends and sensitivity to outliers, giving them more weight than they need within the prediction model (Weedmark).

Data Summary and Implications

After running the model, the R^2 value is 58%, which is less than our hypothesis goal of reaching an accuracy percentage of 65. Which means we reject our alternative hypothesis and accept the null hypothesis, stating that a general Multiple Linear Regression model cannot accurately predict Airbnb price with this dataset. Although the accuracy wasn't high enough, there are two approaches we can take to improve future studies.

1. I wouldn't use just a general multiple linear regression model; I would use that as a baseline, then use a different model, such as Random Forest, to further improve the model accuracy. This also acts like a quick fix because no extra work will need to be done to the data or the data cleaning process.
2. I would collect more data that fits into what we are trying to study to eliminate data noise from the model, along with possibly changing the ways we encode the data to avoid multicollinearity. Along with this, I would focus on specific cities at once to narrow down the study and hyper-focus on smaller features that may have been missed with the large study.

One note I wanted to add as an overall fix for this study is that even though these prices are listed, we don't know how often they are booked. That would be an extra column I would add in a future study because the owner can set the price to be whatever they want, but the value is only worth as much as someone is willing to pay for it. Meaning the prices aren't truly as reflective of reality as we may believe, because what matters most is how often the short-term rental is being booked.

Overall, this is not a good model to predict pricing, but by implementing a clustering technique, such as k-means, we can find which features had the biggest impact on price, which shows our prediction came true that size and location are the biggest factors because zipcode and beds were the biggest impacting features.

Sources

Micheal, Oyebamiji. “A Comprehensive Comparison between One-Hot and Ordinal Encoding | by Oyebamiji Micheal | Medium.” *A Comprehensive Comparison between One-Hot and Ordinal Encoding*, medium.com/@oyebamijimicheal10/a-comprehensive-comparison-between-one-hot-and-ordinal-encoding-6f899c4f08b3. Accessed 6 Dec. 2025.

Weedmark, David. “The Advantages & Disadvantages of a Multiple Regression Model.” *The Advantages & Disadvantages Of A Multiple Regression Model*, Sciencing, 24 Mar. 2022, www.sciencing.com/advantages-disadvantages-multiple-regression-model-12070171/. Accessed 6 Dec. 2025.