Multiple Linear Regression on Airbnb Data

By Alexander Vaillant

Research Question

"To what extent do the independent variables of Airbnb rentals predict the rental price in the Los Angeles Market?"

Setup Environment

Import necessary libraries

In [1]:

```
import pandas as pd
import numpy as np
from numpy import random
import gzip
import shutil
from sklearn.impute import KNNImputer
import re
import seaborn as sns
from sklearn.preprocessing import normalize
import statsmodels.api as sm
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

Turn off unnecessary warnings and set seed for consistent results

In [2]:

```
# Shut off SettingWithCopyWarning; Unnecessary
pd.options.mode.chained_assignment = None
```

In [3]:

```
# Set Random Seed for consistency in results
random.seed(94)
```

Data Extraction

Extract data from original .gz file

In [4]:

```
# Extract data from the .gz file
url_in = "C:/Users/tedda/Desktop/Data Science Portfolio/Machine Learning/Supervised Lea
url_out = "C:/Users/tedda/Desktop/Data Science Portfolio/Machine Learning/Supervised Le
with gzip.open(url_in, "rb") as file_in:
    with open(url_out, "wb") as file_out:
        shutil.copyfileobj(file_in, file_out)
```

Load raw data into a pandas dataframe for EDA and Cleaning

In [5]:
```
data_url = "C:/Users/tedda/Desktop/Data Science Portfolio/Machine Learning/Supervised L
Airbnb_raw = pd.read_csv(data_url, header = 0)
print("Airbnb Raw Data Shape:", Airbnb_raw.shape)
```

```
Airbnb Raw Data Shape: (32240, 74)
```

Currently, there are 74 features with several containing host PII. Avoided .head() to not show PII.

Select desired features without host PII

In [6]:
```
#Remove PII by grabbing only desired columns
AirbnbDesiredColumns = Airbnb_raw[['host_response_time', 'host_response_rate',
       'host_acceptance_rate', 'host_is_superhost', 'host_listings_count',
       'host_has_profile_pic', 'host_identity_verified',
       'neighbourhood_group_cleansed', 'room_type', 'accommodates',
       'bathrooms_text', 'bedrooms', 'beds', 'price', 'minimum_nights',
       'maximum_nights', 'has_availability', 'availability_30',
       'availability_60', 'availability_90', 'availability_365',
       'number_of_reviews', 'review_scores_rating', 'review_scores_accuracy',
       'review_scores_cleanliness', 'review_scores_checkin',
       'review_scores_communication', 'review_scores_location',
       'review_scores_value', 'instant_bookable']]
print("Airbnb Data Shape:", AirbnbDesiredColumns.shape)
AirbnbDesiredColumns.head()
```

```
Airbnb Data Shape: (32240, 30)
```

Out[6]:

	host_response_time	host_response_rate	host_acceptance_rate	host_is_superhost	host_listings_count
0	NaN	NaN	NaN	f	1.0
1	within an hour	100%	100%	t	2.0
2	within an hour	100%	36%	t	2.0
3	NaN	NaN	NaN	f	1.0
4	within a few hours	100%	25%	f	6.0

5 rows × 30 columns

The "id" column can be used to identify a host, which qualifies as PII. Decided to remove the "id" column as well. There are 30 variables left.

Exploratory Data Analysis

Check initial Data Sparsity

In [7]:

```
# Get the initial data sparsity of each column.
# Goal: remove NaN values in all categorical variables so KNNImputer can be used on con
data_sparsity = AirbnbDesiredColumns.isnull().sum()/ len(AirbnbDesiredColumns)
data_sparsity
```

Out[7]:

```
host_response_time              0.266811
host_response_rate              0.266811
host_acceptance_rate            0.260577
host_is_superhost               0.001272
host_listings_count             0.001272
host_has_profile_pic            0.001272
host_identity_verified          0.001272
neighbourhood_group_cleansed    0.000000
room_type                       0.000000
accommodates                    0.000000
bathrooms_text                  0.001799
bedrooms                        0.115043
beds                            0.019789
price                           0.000000
minimum_nights                  0.000000
maximum_nights                  0.000000
has_availability                0.000000
availability_30                 0.000000
availability_60                 0.000000
availability_90                 0.000000
availability_365                0.000000
number_of_reviews               0.000000
review_scores_rating            0.247177
review_scores_accuracy          0.258065
review_scores_cleanliness       0.258033
review_scores_checkin           0.258344
review_scores_communication     0.258096
review_scores_location          0.258437
review_scores_value             0.258499
instant_bookable                0.000000
dtype: float64
```

Descriptive Summary of the Data

In [8]:

```
AirbnbDesiredColumns.describe()
```

Out[8]:

	host_listings_count	accommodates	bedrooms	beds	minimum_nights	maximum_nigh
count	32199.000000	32240.000000	28531.000000	31602.000000	32240.000000	32240.00000
mean	36.371626	3.601427	1.665627	1.964401	20.317246	664.48362
std	200.998227	2.539507	1.085385	1.552321	34.120765	506.68502
min	0.000000	0.000000	1.000000	0.000000	1.000000	1.00000
25%	1.000000	2.000000	1.000000	1.000000	2.000000	90.00000
50%	2.000000	3.000000	1.000000	1.000000	30.000000	1125.00000
75%	7.000000	4.000000	2.000000	2.000000	30.000000	1125.00000
max	2232.000000	16.000000	15.000000	26.000000	1125.000000	10000.00000

Data Preparation

Limit data to only the City of Los Angeles

In [9]:
```
# Delimitation 1: Limit the data to only the City of Los Angeles
AirbnbLA = AirbnbDesiredColumns[(AirbnbDesiredColumns['neighbourhood_group_cleansed'] =
print('New data shape:',AirbnbLA.shape)
```

```
New data shape: (17872, 30)
```

Check data sparsity after removal of two neighborhood groups

In [10]:
```
#Check current data sparsity with the removal of two neighborhood groups.
AirbnbLASparsity = AirbnbLA.isnull().sum()/len(AirbnbLA)
AirbnbLASparsity
```

Out[10]:
```
host_response_time              0.282285
host_response_rate              0.282285
host_acceptance_rate            0.285307
host_is_superhost               0.000112
host_listings_count             0.000112
host_has_profile_pic            0.000112
host_identity_verified          0.000112
neighbourhood_group_cleansed    0.000000
room_type                       0.000000
accommodates                    0.000000
bathrooms_text                  0.001846
bedrooms                        0.131770
beds                            0.023500
price                           0.000000
minimum_nights                  0.000000
maximum_nights                  0.000000
has_availability                0.000000
availability_30                 0.000000
availability_60                 0.000000
availability_90                 0.000000
availability_365                0.000000
number_of_reviews               0.000000
review_scores_rating            0.284188
review_scores_accuracy          0.295546
review_scores_cleanliness       0.295490
review_scores_checkin           0.295826
review_scores_communication     0.295490
review_scores_location          0.295882
review_scores_value             0.295938
instant_bookable                0.000000
dtype: float64
```

Host_response_rate and host_acceptance_rates must be changed from percentage strings to floats. Price must be changed to float.

Convert percentages and currency to Floats

In [11]:
```
AirbnbLA['host_response_rate']=AirbnbLA['host_response_rate'].replace('%','', regex = T
AirbnbLA['host_acceptance_rate']=AirbnbLA['host_acceptance_rate'].replace('%','', regex
AirbnbLA['price']=AirbnbLA['price'].replace('[\$,]','', regex = True).astype(float)
AirbnbLA.head()
```

Out[11]:

	host_response_time	host_response_rate	host_acceptance_rate	host_is_superhost	host_listings_count	h

	host_response_time	host_response_rate	host_acceptance_rate	host_is_superhost	host_listings_count
1	within an hour	1.0	1.00	t	2.0
4	within a few hours	1.0	0.25	f	6.0
5	within a few hours	1.0	0.80	t	8.0
6	within a few hours	1.0	0.80	t	8.0
7	within a few hours	1.0	0.80	t	8.0

5 rows × 30 columns

Bathrooms_text needs to be parsed.

Since we cannot know whether each individual bath is either private or shared, remove the text.

In [12]:
```
# Current Value Counts
AirbnbLA['bathrooms_text'].value_counts()
```

Out[12]:
```
1 bath               8548
2 baths              2579
1 shared bath        1619
1 private bath       1448
2.5 baths             645
1.5 baths             575
3 baths               533
1.5 shared baths      282
2 shared baths        263
3.5 baths             246
4 baths               191
4.5 baths             160
8 shared baths         97
5 baths                95
5.5 baths              91
2.5 shared baths       68
3 shared baths         53
6 baths                39
6.5 baths              36
4 shared baths         34
3.5 shared baths       30
0 shared baths         22
8 baths                22
11 shared baths        22
0 baths                17
7 baths                16
Private half-bath      14
Half-bath              13
7.5 baths              12
10 baths               10
4.5 shared baths        9
8.5 baths               9
Shared half-bath        7
9 baths                 7
```

```
8.5 shared baths        6
5 shared baths          4
11 baths                3
11.5 baths              2
13 baths                2
10.5 baths              2
12.5 baths              2
9.5 baths               2
25 baths                1
6 shared baths          1
12 baths                1
11.5 shared baths       1
Name: bathrooms_text, dtype: int64
```

Note that there are three full text values: Half-bath, Shared half-bath, and Private half-bath. Convert each to 0.5

Parse Bathrooms_text column and convert to Float

In [13]:

```
AirbnbLA['bathrooms'] = AirbnbLA['bathrooms_text'].replace(
    ("Half-bath","Shared half-bath","Private half-bath"),"0.5 baths", regex = True).str
AirbnbLA.drop('bathrooms_text', axis=1,inplace=True)

# New Value Counts
AirbnbLA['bathrooms'].value_counts()
```

Out[13]:

```
1.0     11615
2.0      2842
1.5       857
2.5       713
3.0       586
3.5       276
4.0       225
4.5       169
8.0       119
5.0        99
5.5        91
6.0        40
0.0        39
6.5        36
0.5        34
11.0       25
7.0        16
8.5        15
7.5        12
10.0       10
9.0         7
11.5        3
9.5         2
12.5        2
10.5        2
13.0        2
25.0        1
12.0        1
Name: bathrooms, dtype: int64
```

Prepare data for KNN Imputation

Converting Categorical Variables into Binary Dummy Variables

In [14]:

```
# Transform the categorical variables into binary dummy varibles; drop_first removes 1s
```

```
AirbnbLA = pd.get_dummies(AirbnbLA, drop_first = True)
AirbnbLA.head()
```

Out[14]:

	host_response_rate	host_acceptance_rate	host_listings_count	accommodates	bedrooms	beds	price
1	1.0	1.00	2.0	1	1.0	1.0	74.0
4	1.0	0.25	6.0	2	1.0	2.0	118.0
5	1.0	0.80	8.0	2	1.0	1.0	50.0
6	1.0	0.80	8.0	2	1.0	1.0	65.0
7	1.0	0.80	8.0	4	2.0	2.0	130.0

5 rows × 33 columns

Perform KNN Imputation

In [15]:

```
# Instantiate the KNNImputer.
# Fill in missing values by fit_transform
AirbnbLAColumns = AirbnbLA.columns
imputer = KNNImputer(n_neighbors = 5, weights = 'uniform', metric = 'nan_euclidean')
AirbnbLA_imputed = imputer.fit_transform(AirbnbLA)
Airbnb = pd.DataFrame(AirbnbLA_imputed, columns = AirbnbLAColumns)
print(Airbnb.shape)
Airbnb.head()
```

```
(17872, 33)
```

Out[15]:

	host_response_rate	host_acceptance_rate	host_listings_count	accommodates	bedrooms	beds	price
0	1.0	1.00	2.0	1.0	1.0	1.0	74.0
1	1.0	0.25	6.0	2.0	1.0	2.0	118.0
2	1.0	0.80	8.0	2.0	1.0	1.0	50.0
3	1.0	0.80	8.0	2.0	1.0	1.0	65.0
4	1.0	0.80	8.0	4.0	2.0	2.0	130.0

5 rows × 33 columns

Check final data sparsity after KNN Imputation

In [16]:

```
final_data_sparsity = Airbnb.isnull().sum()/len(Airbnb)
final_data_sparsity
```

Out[16]:

```
host_response_rate                         0.0
host_acceptance_rate                       0.0
host_listings_count                        0.0
accommodates                               0.0
bedrooms                                   0.0
beds                                       0.0
```

```
price                                   0.0
minimum_nights                          0.0
maximum_nights                          0.0
availability_30                         0.0
availability_60                         0.0
availability_90                         0.0
availability_365                        0.0
number_of_reviews                       0.0
review_scores_rating                    0.0
review_scores_accuracy                  0.0
review_scores_cleanliness               0.0
review_scores_checkin                   0.0
review_scores_communication             0.0
review_scores_location                  0.0
review_scores_value                     0.0
bathrooms                               0.0
host_response_time_within a day         0.0
host_response_time_within a few hours   0.0
host_response_time_within an hour       0.0
host_is_superhost_t                     0.0
host_has_profile_pic_t                  0.0
host_identity_verified_t                0.0
room_type_Hotel room                    0.0
room_type_Private room                  0.0
room_type_Shared room                   0.0
has_availability_t                      0.0
instant_bookable_t                      0.0
dtype: float64
```

There are now no null values in the dataset. The null values in both continuous and categorical variables were imputed using KNNImputer.

Normalization before KNN Imputation on this dataset reduced accuracy. ***Normalize dataset after KNN Imputation***

Normalize data after KNN Imputation (Normality Assumption of Linear Regression Models)

In [17]:

```
AirbnbLANormalized = pd.DataFrame(normalize(Airbnb), columns = AirbnbLAColumns)
AirbnbLANormalized.head()
```

Out[17]:

	host_response_rate	host_acceptance_rate	host_listings_count	accommodates	bedrooms	beds	
0	0.002046	0.002046	0.004092	0.002046	0.002046	0.002046	0.
1	0.001310	0.000327	0.007858	0.002619	0.001310	0.002619	0.
2	0.000852	0.000682	0.006819	0.001705	0.000852	0.000852	0.(
3	0.000867	0.000694	0.006940	0.001735	0.000867	0.000867	0.(
4	0.004849	0.003879	0.038789	0.019395	0.009697	0.009697	0.(

5 rows × 33 columns

Export Cleansed, Imputed, and Normalized Dataset

In [18]:
```
# Export the complete cleansed, imputed dataset

AirbnbLANormalized.to_csv("C:/Users/tedda/Desktop/Data Science Portfolio/Machine Learni
```

Exploratory Data Analysis Pt. 2

Correlation Matrix (Identify Potential Multicollinearity):

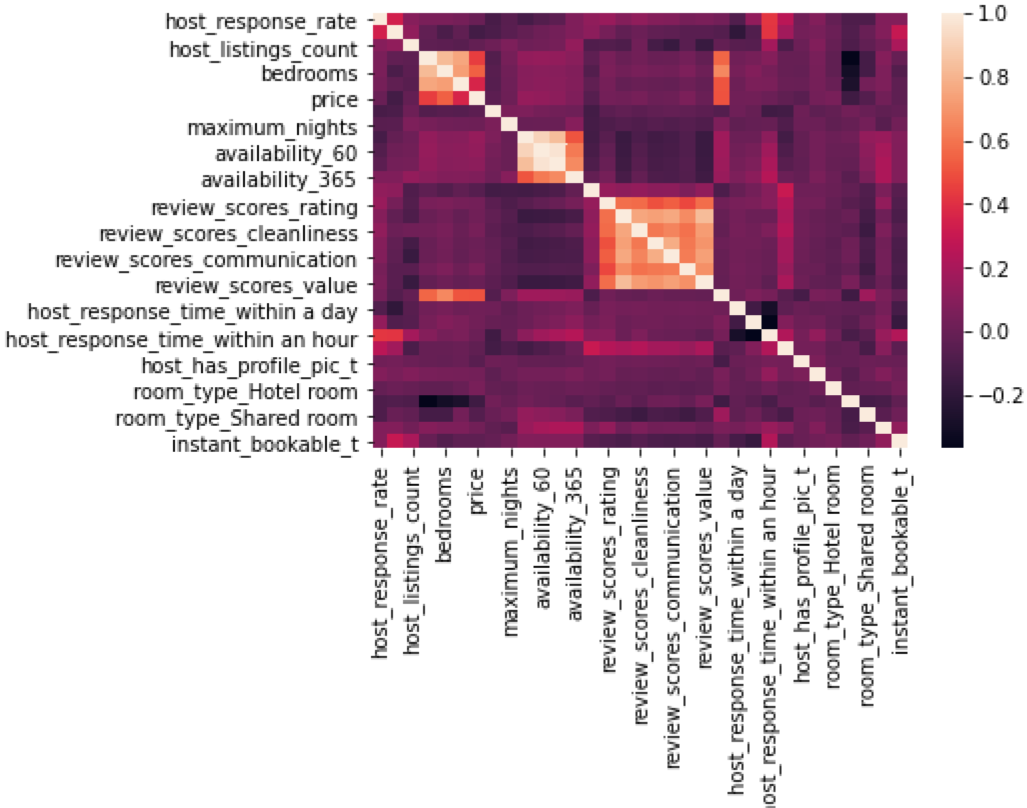
Create Correlation Matrix of Features

In [19]:
```
AirbnbCorrelationMatrix = Airbnb.corr()
AirbnbCorrelationMatrix.to_csv("C:/Users/tedda/Desktop/Data Science Portfolio/Machine L
```

Plot Correlation Matrix

In [20]:
```
sns.heatmap(AirbnbCorrelationMatrix)
```

Out[20]: <AxesSubplot:>



There are several features with high correlation to each other. VIF needed.

VIF Check (Remove Multicollinearity):

Define function to calculate VIF and remove features above set threshold of 10

In [21]:
```
# Define a function that calculates the VIF of each feature
# Remove any feature with a VIF higher than the threshold (Good thresholds are between
# Export the dataframe of features without high VIF
```

```
def calculate_vif_(df, thresh=10):
    global X_NoMulti
    const = sm.add_constant(df)
    cols = const.columns
    variables = np.arange(const.shape[1])
    vif_df = pd.Series([variance_inflation_factor(const.values, i)
               for i in range(const.shape[1])],
              index=const.columns).to_frame()

    vif_df = vif_df.sort_values(by=0, ascending=False).rename(columns={0: 'VIF'})
    vif_df = vif_df.drop('const')
    vif_df = vif_df[vif_df['VIF'] > thresh]

    print('Features above VIF threshold:\n')
    print(vif_df[vif_df['VIF'] > thresh])

    col_to_drop = list(vif_df.index)

    for i in col_to_drop:
        print('Dropping: {}'.format(i))
        df = df.drop(columns=i)

    X_NoMulti = df
    return X_NoMulti
```

Split dataframe into X and y values for VIF check

In [22]:

```
y = pd.DataFrame(AirbnbLANormalized['price'])
X = AirbnbLANormalized.drop(['price'],axis=1)
```

Calculate VIF and remove features above threshold

In [23]:

```
calculate_vif_(X, thresh = 10)
```

```
Features above VIF threshold:

                                    VIF
review_scores_accuracy       441.642173
review_scores_checkin        367.390774
review_scores_value          338.563331
review_scores_communication  327.573744
review_scores_location       229.937998
review_scores_cleanliness    132.343336
host_has_profile_pic_t       104.349085
availability_60               34.543010
review_scores_rating          24.662996
availability_90               22.464315
host_response_rate            18.682910
Dropping: review_scores_accuracy
Dropping: review_scores_checkin
Dropping: review_scores_value
Dropping: review_scores_communication
Dropping: review_scores_location
Dropping: review_scores_cleanliness
Dropping: host_has_profile_pic_t
Dropping: availability_60
Dropping: review_scores_rating
Dropping: availability_90
Dropping: host_response_rate
```

Out[23]:

	host_acceptance_rate	host_listings_count	accommodates	bedrooms	beds	minimum_nights
0	0.002046	0.004092	0.002046	0.002046	0.002046	0.061387
1	0.000327	0.007858	0.002619	0.001310	0.002619	0.040597
2	0.000682	0.006819	0.001705	0.000852	0.000852	0.025570
3	0.000694	0.006940	0.001735	0.000867	0.000867	0.026024
4	0.003879	0.038789	0.019395	0.009697	0.009697	0.145460
...	...	...	...	...	...	...
17867	0.000504	0.004068	0.006509	0.003255	0.003255	0.024411
17868	0.000504	0.004068	0.006509	0.003255	0.003255	0.024411
17869	0.000727	0.009192	0.005014	0.001671	0.003343	0.002507
17870	0.000832	0.012613	0.005045	0.001682	0.001682	0.001682
17871	0.000846	0.012826	0.003420	0.000855	0.000855	0.001710

17872 rows × 21 columns

Data Cleansing Pt. 2

Split Training and Testing Datasets (For Model Evaluation on Unseen Data)

Split Data into Training (80%) and Testing (20%) datasets

In [24]:

```
x_train, x_test, y_train, y_test = train_test_split(X_NoMulti, y, test_size = 0.2, rand
```

Print shapes of Training and Testing datasets

In [25]:

```
print("x_train shape:", x_train.shape)
print("x_test shape:", x_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
x_train shape: (14297, 21)
x_test shape: (3575, 21)
y_train shape: (14297, 1)
y_test shape: (3575, 1)
```

Export Training and Testing datasets before Stepwise Regression

In [26]:

```
x_train.to_csv("C:/Users/tedda/Desktop/Data Science Portfolio/Machine Learning/Supervis
x_test.to_csv("C:/Users/tedda/Desktop/Data Science Portfolio/Machine Learning/Supervise
y_train.to_csv("C:/Users/tedda/Desktop/Data Science Portfolio/Machine Learning/Supervis
y_test.to_csv("C:/Users/tedda/Desktop/Data Science Portfolio/Machine Learning/Supervise
```

Perform Stepwise Regression (For Feature Selection of most impactful features on explained variance)

In [27]:

```
# Instantiate the Linear Regression algorithm
LR = LinearRegression()

# Build step forward feature selection
sfs = SequentialFeatureSelector(
    estimator = LR
    ,n_features_to_select = None
    ,direction = 'forward'
    ,scoring = 'explained_variance'
    ,cv = 10)

# Perform SFFS
sfs = sfs.fit(x_train, y_train)
x_variables = sfs.transform(x_train)

print("Selected x_variables shape:", x_variables.shape)
```

```
Selected x_variables shape: (14297, 10)
```

Show Support Values of each Feature

In [28]:

```
SupportClassification = sfs.get_support().reshape(1,x_train.shape[1])
XColumns = X_NoMulti.columns
XSupport = pd.DataFrame(SupportClassification, columns = (XColumns)).transpose().rename
XSupport.head(x_train.shape[1])
```

Out[28]:

	Support
host_acceptance_rate	False
host_listings_count	True
accommodates	True
bedrooms	True
beds	False
minimum_nights	True
maximum_nights	True
availability_30	False
availability_365	True
number_of_reviews	True
bathrooms	False
host_response_time_within a day	False
host_response_time_within a few hours	False
host_response_time_within an hour	False
host_is_superhost_t	True
host_identity_verified_t	False

	Support
room_type_Hotel room	False
room_type_Private room	True
room_type_Shared room	True
has_availability_t	False
instant_bookable_t	False

Remove non-supported columns from training and testing datasets

In [29]:

```
# Get list of columns with Support of True
XVariables = XSupport[(XSupport['Support'] == True)].transpose().columns
x_train_stepwise = x_train[XVariables]
x_test_stepwise = x_test[XVariables]

# Print shape of new dataframes
print('Post-Stepwise Regression x_train shape:', x_train_stepwise.shape)
print('Post-Stepwise Regression x_test shape:', x_test_stepwise.shape)
```

```
Post-Stepwise Regression x_train shape: (14297, 10)
Post-Stepwise Regression x_test shape: (3575, 10)
```

Export Post-Stepwise Regression Training and Testing datasets

In [30]:

```
x_train_stepwise.to_csv("C:/Users/tedda/Desktop/Data Science Portfolio/Machine Learning
x_test_stepwise.to_csv("C:/Users/tedda/Desktop/Data Science Portfolio/Machine Learning/
```

Create and Fit the Linear Regression Model

In [31]:

```
# Add constant to x_train values
x_const = sm.add_constant(x_train_stepwise)

# Build and fit model to training data
MLR = sm.OLS(y_train, x_const)
MLR_Fitted = MLR.fit()

# Print Model Summary
print(MLR_Fitted.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                  price   R-squared:                       0.819
Model:                            OLS   Adj. R-squared:                  0.819
Method:                 Least Squares   F-statistic:                     6448.
Date:                Mon, 06 Sep 2021   Prob (F-statistic):               0.00
Time:                        11:18:16   Log-Likelihood:                 9594.5
No. Observations:               14297   AIC:                        -1.917e+04
Df Residuals:                   14286   BIC:                        -1.908e+04
Df Model:                          10                                         
Covariance Type:            nonrobust                                         
================================================================================
==
                             coef    std err          t      P>|t|      [0.025      0.97
5]
```

```
------------------------------------------------------------------------------------------
--
const                      0.9648      0.005    182.956      0.000       0.955       0.9
75
host_listings_count       -0.4453      0.008    -55.972      0.000      -0.461      -0.4
30
accommodates               4.2460      0.307     13.852      0.000       3.645       4.8
47
bedrooms                  19.5615      0.779     25.106      0.000      18.034      21.0
89
minimum_nights            -0.1545      0.020     -7.865      0.000      -0.193      -0.1
16
maximum_nights            -0.7789      0.005   -164.089      0.000      -0.788      -0.7
70
availability_365          -0.4642      0.005    -95.914      0.000      -0.474      -0.4
55
number_of_reviews         -0.4610      0.008    -58.484      0.000      -0.476      -0.4
46
host_is_superhost_t        8.5394      0.802     10.647      0.000       6.967      10.1
12
room_type_Private room   -15.2066      0.640    -23.744      0.000     -16.462     -13.9
51
room_type_Shared room    -29.9452      1.331    -22.498      0.000     -32.554     -27.3
36
==============================================================================
Omnibus:                      189.975   Durbin-Watson:                   2.038
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              294.020
Skew:                           0.134   Prob(JB):                     1.43e-64
Kurtosis:                       3.649   Cond. No.                     1.63e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[2] The condition number is large, 1.63e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Coefficients of Model

In [32]:

```
# Print coefficients of each feature and constant
print(MLR_Fitted.params)
```

```
const                      0.964846
host_listings_count       -0.445316
accommodates               4.245991
bedrooms                  19.561517
minimum_nights            -0.154465
maximum_nights            -0.778914
availability_365          -0.464200
number_of_reviews         -0.460954
host_is_superhost_t        8.539390
room_type_Private room   -15.206620
room_type_Shared room    -29.945156
dtype: float64
```

Residual Plot of Training Set

In [33]:

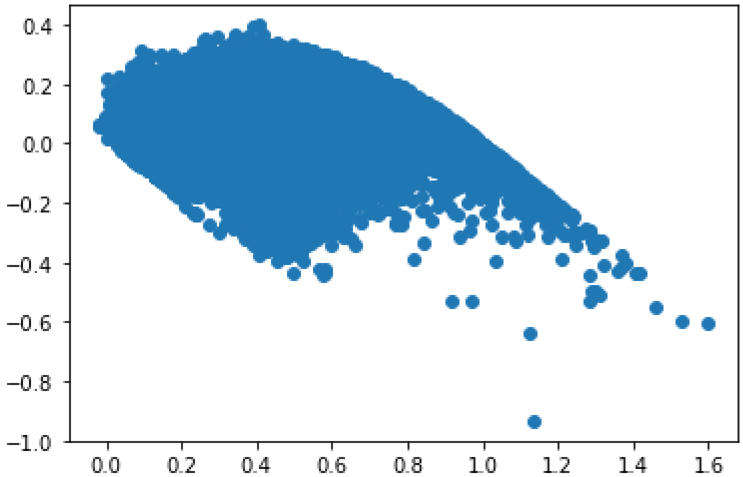
```
# Get predictions of y_train values
TrainPredictions = MLR_Fitted.predict(x_const)

# Reshape y_test and predictions
y_train_array = np.array(y_train).reshape(x_const.shape[0],1)
```

```
TrainPredictions_array = np.array(TrainPredictions).reshape(x_const.shape[0],1)

# Calculate residuals
TrainResiduals = y_train_array - TrainPredictions_array

# Create a scatter plot of the residuals vs. predictions
plt.scatter(TrainPredictions_array, TrainResiduals)
plt.show()
```



Save and Load Model

In [34]:

```
# Import Joblib Library to Save and Load Model
import joblib

# Export as .pkl file to Save the Trained Model
joblib_url = "C:/Users/tedda/Desktop/Data Science Portfolio/Machine Learning/Supervised
joblib.dump(MLR_Fitted, joblib_url)

# Load in the Saved Model
MLR_model = joblib.load(joblib_url)
```

Model Evaluation

Evaluation Metrics: MSE and R-Squared on Unseen Data

In [35]:

```
# Add constant to x_test values
x_test_const = sm.add_constant(x_test_stepwise)

# Get predictions of y_test values
predictions = MLR_model.predict(x_test_const)

# The mean squared error
print('Unseen Data Mean squared error:', mean_squared_error(y_test, predictions), '\n')

# The R-squared/Explained Variance Score (Coefficient of Determination): 1 is perfect p
print('Unseen Data R-Squared/Explained Variance Score:',  r2_score(y_test, predictions)
```

```
Unseen Data Mean squared error: 0.01521526606581391
```

```
Unseen Data R-Squared/Explained Variance Score: 0.8200367463286045
```

Residual Plot of Testing Set

In [36]:

```
# Reshape y_test and predictions
y_test_array = np.array(y_test).reshape(3575,1)
predictions_array = np.array(predictions).reshape(3575,1)

# Calculate residuals
residuals = y_test_array - predictions_array

# Create a scatter plot of the residuals vs. predictions
plt.scatter(predictions_array, residuals)
plt.show()
```