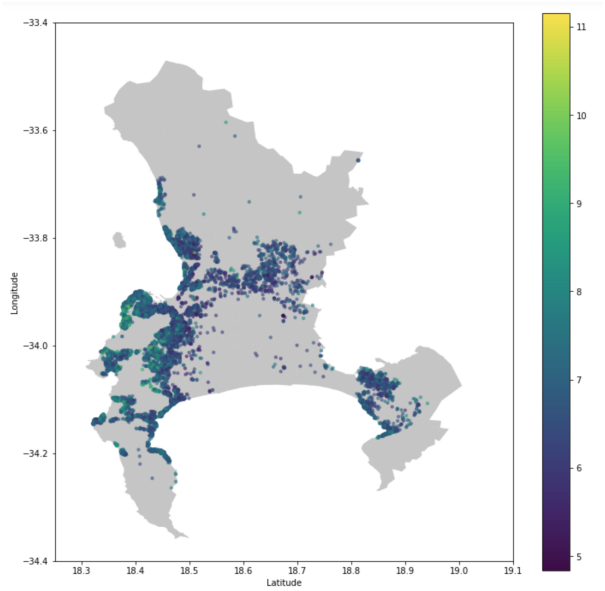
COMS7063A Statistics for Data Science

WITS UNIVERSITY

Term Project

Modelling AirBnB Price in Cape Town



Samantha Ball 1603701

1. Executive Summary

This notebook provides an investigation into the factors influencing AirBnB price in Cape Town. AirBnB price is modelled using a linear regression model under OLS assumptions. The various predictors are analysed to determine which variables have the greatest impact on the price of an AirBnB unit, and whether location or the inherent characteristics of the unit is more significant when determining AirBnB price. Several models obtained through forward subset selection and LASSO procedures are compared with the final model selected according to model selection methods and domain knowledge. Inference is then performed on final model and hypothesis tests are performed to determine whether the selected variables are significant.

From the findings we observe that the number of guests accommodated plays a dominant role in determining the price of an AirBnB unit, accounting for the majority of the variance. While variables pertaining to location such as longitude were included in the final model, these variables displayed a much weaker relationship with

the target price variable. This suggests that the characteristics of the AirBnB unit itself has a greater influence on the final price than the location of the unit. It can be noted that this study was limited to only one month of AirBnB data and could be extended to analyse additional months, as well as time series modelling to capture information such as seasonality. Moreover, additional cities could be analysed to ascertain whether the observed relationships are a general trend or specific to the Cape Town regions. Lastly, a significant amount of variation remains unexplained by the available predictors, warranting further investigation.

2. Introduction

2.1 Background to the Study

Modelling AirBnB prices presents a challenging problem due to seasonal changes in the market and differing price distributions across different cities. The advent of AirBnB has disrupted the hospitality industry by introducing lower rates, a wide variety of venues and an online marketplace. The seasonal and complex nature of the AirBnB market places hosts in a difficult position when determining the optimal price for a venue. The numerous factors that influence the price a customer will be willing to pay for a unit presents a unique modelling challenge.

2.2. Research Question

Accordingly, this investigation aims to determine the core factors that influence the AirBnB price distribution in Cape Town, and the extent to which the variation in AirBnB price can be explained by these factors. Therefore the investigation seeks to find a set of variables to best explain AirBnB pricing across Cape Town and hence understand the relationship of each predictor variable to the target variable *price*. More specifcally, the role of spatial features such as the location of the AirBnB is constrasted with the inherent characteristics such as the size and amenities available at the AirBnB in order to determine which factors plays a larger role in the resulting price.

2.3 Methodology

To provide a suitable dataset for the research question, the *InsideAirBnB* dataset was utilised, containing up-to-date information scraped from the AirBnB website. Since the dataset contained a large number of potential predictors, data exploration and initial model fitting was undertaken to gain an understanding of the relationships between the features and the target. Subsequently, in order to select a suitable subset of variables from the available predictors, model selection was performed using LASSO and forward stepwise regression techniques. The final model was then selected according to the Akaike's Information Criterion criterion together with insights extracted through the data exploration procedure. Lastly, the selected model was refitted on the inference set and the statistical significance of each included variable was determined. To account for the effects of multiple testing, Bonferroni's Procedure was utilised to control the amount of false positives. The final model after exclusion of insignificant variables was then refitted on the inference set to obtain the final coefficient estimates.

2.4 Section Contents Overview

The following section provides details about the dataset used for the investigation, including any limitations or biases that may be introduced from the given dataset. Section 4 then describes the data wrangling and cleaning process to convert the scraped data into workable format for further analysis. Following data cleaning and transformation, the data is split into a selection and inference set in order to prevent data leakage. Exploratory analyses are then performed on the selection dataset in Section 5 to gain insight into predictor-target relationships. Section 6 describes the model fitting and selection process used to obtain the final linear

model. In order to estimate the performance of the final model, the selected model is then refitted on the inference set and evaluated by inspecting the significance of the p-values. Lastly, Section 8 interprets the results in light of the broader investigation context.

3. Data Description

3.1 Origin and Contents

The Cape Town AirBnB dataset was obtained from *Inside AirBnB* [1], a compiled set of publicly available and verified AirBnB data spanning multiple cities. The data is collected by scraping the AirBnB website for the relevant prices and features. The dataset includes details about the listing such as number of guests, amenities, location, neighbourhood and property type as well as characteristics of the host such as response rate, number of listings and identity verification. Review scores and policies such as the maximum or minimum number of nights allowed provide additional information about the listing.

3.2 Data Structure

The Cape Town data set encompasses all AirBnB listings per month from 2019 to 2021. The focus of the investigation has been narrowed to the month of September 2021 as this is the most recently released dataset. The September 2021 dataset was downloaded on the 1st of October from the *Inside AirBnB* website. The data has been made publicly available in CSV format. The September 2021 dataset comprises 17016 samples and 74 features including textual, continuous, categorical and spatial attributes. Narrowing the focus of the investigation to a single month allows for analysis of the contribution of individual variables without the influence of temporal and seasonal changes.

3.3 Data Quality

From a data quality perspective, the data contains some variables with a fairly large proportion of missing data which will need to be handled appropriately. In addition, several outliers are present in the dataset which require further investigation to determine their validity. In terms of the suitability of the data for the research question, the data is both up-to-date and contains a significant number of features that are visible to potential guests. Therefore the features should provide a good indication of the aspects that potential guests look for when selecting an AirBnB and therefore what factors influence the price of the AirBnB.

3.4 Data Validations

In order to ensure the validity of the data, several checks were performed. Firstly, the overall number of observations and features in the dataset were compared to previous months in order to observe whether the number of observations and number of features fluctuated in a reasonable manner. It was determined that the total number of listings stayed fairly constant in short-term, while the number of features remained consistent at 74 features. From a long-term perspective, the total number of listings experienced decline between April 2021 and July 2021. This phenomenon could not be investigated further due to a lack of data available for the months of May and June 2021. However, no specific pattern is observed in the removed listings and therefore no identifiable bias is expected to have been introduced by the observed decline in AirBnB listings.

From inspection of the two months surrounding the chosen dataset, August and October 2021, the number of listings and features remained relatively constant. Therefore this provides a strong indication that the size of the September 2021 dataset is valid with respect to expectations from the rest of the data. Furthermore, the November 2020 dataset was included in the validation process as it represents the closest month to the

September dataset that is available from the previous year. Lastly, the April 2021 dataset was included in the validation process to further understand how the number of listings decreased over time, and whether the observed decrease was gradual or sudden.

Using these selected datasets, averages across features were compared to ensure that any differences were reasonable over the given time period. Lastly, the ranges of the variables, especially the target price variable, were compared across the different months to ensure reasonable consistency. The dataset was also examined for duplicates.

A summary of these validations is included below:

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```

Comparing Current Dataset with Historical Dataset

The characteristics of the main dataset utilised for the analysis, September 2021, were compared to the surrounding months and comparable months from previous years in order to perform validations on the dataset.

Number of Listings and Features over Time

As mentioned, the number of listings remained relatively constant in the short-term, with an observed decline in the number of listings occurring from April 2021 to July 2021. The number of features remained constant at 74.

In [2]:

```
##--- Import Data ---##

df_oct = pd.read_csv('Data/listings_October_2021.csv', low_memory=False)
df = pd.read_csv('Data/listings_September_2021.csv', low_memory=False)
df_aug = pd.read_csv('Data/listings_August_2021.csv', low_memory=False)
df_july = pd.read_csv('Data/listings_July_2021.csv', low_memory=False)
df_april = pd.read_csv('Data/listings_April_2021.csv', low_memory=False)
df_november = pd.read_csv('Data/listings_November_2020.csv', low_memory=False)
pd.set_option("display.max_columns", None)

##--- Dataset Shape ---##

no_listings = [df_november.shape[0], df_april.shape[0], df_july.shape[0],  df_aug.sh
no_features = [df_november.shape[1], df_april.shape[1], df_july.shape[1],  df_aug.sh
index_list = ['November 2020', 'April 2021', 'July 2021', 'August 2021', 'September
df_size = pd.DataFrame({'No. of Listings': no_listings, 'No. of Features': no_featur
df_size.head(10)
```

Out[2]:

	No. of Listings	No. of Features
November 2020	19041	74
April 2021	18999	74
July 2021	16923	74
August 2021	16936	74
September 2021	17016	74
October 2021	16891	74

Summary Statistics over Time

We observe below that the quantiles of the target price variable remain fairly consistent over the months, therefore providing confidence in the validity of the September 2021 dataset in comparison to the dataset as a whole.

In [3]:

```
##--- Range and Average of Target Variable ---##

# change formatting of monetary amounts
def fix_price_format(df):

    df["price"] = df["price"].str.replace('$', '', regex = False)
    df["price"] = df["price"].str.replace(',', '', regex = False)
    df["price"] = df["price"].astype(float)

fix_price_format(df_oct)
fix_price_format(df)
fix_price_format(df_aug)
fix_price_format(df_july)
fix_price_format(df_april)
fix_price_format(df_november)

##--- Summary Statistics over Time ---##

df_compare = pd.DataFrame({'November 2020': df_november['price'].describe(), 'April
df_compare.head(10)
```

Out[3]:

	November 2020	April 2021	July 2021	August 2021	September 2021	October 20
count	19041.000000	18999.000000	16923.000000	16936.000000	17016.000000	16891.0000
mean	2427.257182	2270.162377	2209.711399	2247.907593	2320.457687	2357.3498
std	5877.043331	6077.877475	5784.973918	5723.017268	5675.639654	5361.3071
min	115.000000	122.000000	130.000000	131.000000	126.000000	120.0000
25%	650.000000	610.000000	600.000000	620.000000	650.000000	679.0000
50%	1100.000000	1000.000000	999.000000	1000.000000	1074.000000	1114.0000
75%	2200.000000	2000.000000	2000.000000	2000.000000	2107.000000	2200.0000
max	177743.000000	210000.000000	210000.000000	175500.000000	175500.000000	175500.0000

Checking for Duplicates

No duplicates were found in the September 2021 dataset.

In [4]:

```
##--- Check for Duplicates ---##

print("Number of Duplicate Rows: {}".format(np.sum(df.duplicated())))
```

```
Number of Duplicate Rows: 0
```

4. Data Wrangling

In order to transform the data into a format suitable for further analysis, several preprocessing steps were undertaken such as data cleaning, handling of missing values, data transformation, data aggregation, error checking and outlier removal.

In [5]:

```
##--- Import Data ---##

df = pd.read_csv('Data/listings_September_2021.csv', low_memory=False)
pd.set_option("display.max_columns", None)
df.head(3)
```

Out[5]:

	id	listing_url	scrape_id	last_scraped	name	descript
0	3191	https://www.airbnb.com/rooms/3191	20210929043437	2021-09-29	Malleson Garden Cottage	This lov separ s cate co
1	15007	https://www.airbnb.com/rooms/15007	20210929043437	2021-09-29	Blaauwberg House	Welcom our s cate beach-fr fam
2	15068	https://www.airbnb.com/rooms/15068	20210929043437	2021-09-29	Grande Bay	Mod spaci apartm Th bedroc tw

4.1 Data Tidyness

Since the dataset does not require any joins or merges between separate tables, the data is already in a tidy format. Therefore the first step performed in the data cleaning pipeline is the checking of data types and any necessary type correction which is performed in the next section.

4.2 Data Formatting and Type Correction

Firstly, the default data types are explored and corrected to transform any useful predictors into numerical predictors, or conversely to ensure that features such as IDs are not interpreted as numerical values. In addition, any formatting issues such as added characters that require pre-processing are handled.

a) Object to Numerical

Firstly, any variables that have been read in as objects but that contain numerical values are formatted and converted to float or integer data types where appropriate.

From an inspection of the features read in as objects, the following is ascertained:

- We identify the potentially numerical variables *host_response_rate, host_acceptance_rate* and our target variable *price*.

- We also identify categorical variables *neighbourhood, neighbourhood_cleansed, property_type, room_type, host_neighbourhood, host_response_time* and *host_name* that require suitable processing and encoding and will be further explored further in the notebook.

Additionally, the following features are reformatted:

- The target variable *price* is reformatted by removing the leading dollar symbol and any separating commas
- The predictor variable *neighbourhood_cleansed* is transformed to indicate only the ward number by removing the prefix 'Ward'
- Any variables represented as a rate are stripped of the suffix '%'
- The feature *bathrooms_text* is processed into purely numerical format by removing several textual variants

In [6]:

```
##--- Check for numerical features stored as objects ---##

print(df.dtypes[df.dtypes =='object'])
```

```
listing_url               object
last_scraped              object
name                      object
description               object
neighborhood_overview     object
picture_url               object
host_url                  object
host_name                 object
host_since                object
host_location             object
host_about                object
host_response_time        object
host_response_rate        object
host_acceptance_rate      object
host_is_superhost         object
host_thumbnail_url        object
host_picture_url          object
host_neighbourhood        object
host_verifications        object
host_has_profile_pic      object
host_identity_verified    object
neighbourhood             object
neighbourhood_cleansed    object
property_type             object
room_type                 object
bathrooms_text            object
amenities                 object
price                     object
has_availability          object
calendar_last_scraped     object
first_review              object
last_review               object
instant_bookable          object
dtype: object
```

In [7]:

```
##--- Correct Data Formats and Types ---##

# Initial format of potentially numerical variables

df[["id", "price", "bathrooms_text", "neighbourhood_cleansed", "host_response_rate",

# change formatting of monetary amounts
df["price"] = df["price"].str.replace('$', '', regex = False)
df["price"] = df["price"].str.replace(',', '', regex = False)


# reformat neighbourhood from string to number
df["neighbourhood_cleansed"] = df["neighbourhood_cleansed"].str.replace('Ward ', '',


# reformat bathrooms from string to number
df["bathrooms_text"] = df["bathrooms_text"].str.replace('half-bath', '0.5', regex =
df["bathrooms_text"] = df["bathrooms_text"].str.replace('Half-bath', '0.5', regex =
df["bathrooms_text"] = df["bathrooms_text"].str.replace(' private bath', '', regex =
df["bathrooms_text"] = df["bathrooms_text"].str.replace(' baths', '', regex = False)
df["bathrooms_text"] = df["bathrooms_text"].str.replace(' bath', '', regex = False)
df["bathrooms_text"] = df["bathrooms_text"].str.replace('shared', '', regex = False)
df["bathrooms_text"] = df["bathrooms_text"].str.replace('Shared', '', regex = False)
df["bathrooms_text"] = df["bathrooms_text"].str.replace('Private', '', regex = False

# remove symbols (%, $, . )
df["host_response_rate"] = df["host_response_rate"].str.replace('%', '')
df["host_acceptance_rate"] = df["host_acceptance_rate"].str.replace('%', '')


# change from string to float
df[["price", "bathrooms_text", "host_response_rate", "host_acceptance_rate"]] = df[

# change from string to int
df["neighbourhood_cleansed"] = df["neighbourhood_cleansed"].astype("int")

# check changes are as expected
df[["id", "price", "bathrooms_text", "neighbourhood_cleansed", "host_response_rate",
```

Out[7]:

	id	price	bathrooms_text	neighbourhood_cleansed	host_response_rate	host_acceptance
0	3191	452.0	1.0	57	NaN	
1	15007	2458.0	3.0	23	100.0	
2	15068	2800.0	2.0	23	60.0	
3	15077	968.0	1.5	4	100.0	
4	15199	2500.0	1.0	115	100.0	

b) Numerical to Object

In contrast to object to numeric conversion where the goal is to ensure predictor variables are in a valid numerical format for further analysis, some numerical variables are not suitable for mathematical analysis and should thus be represented as an object. These variables include *ids* such as *id, scrape_id and host_id* where it would be inappropriate to report summary statistics such as the mean or percentiles for these *id* values.

For certain variables, it would be incorrect to interpret them as numerical values subject to further operations such as averaging or summing. These are identified below.

In [8]:

```
##--- Check for non-numerical features stored as continuous numerical variables ---#

print(df.dtypes[df.dtypes =='float'])
```

```
host_response_rate              float64
host_acceptance_rate            float64
host_listings_count             float64
host_total_listings_count       float64
neighbourhood_group_cleansed    float64
latitude                        float64
longitude                       float64
bathrooms                       float64
bathrooms_text                  float64
bedrooms                        float64
beds                            float64
price                           float64
minimum_nights_avg_ntm          float64
maximum_nights_avg_ntm          float64
calendar_updated                float64
review_scores_rating            float64
review_scores_accuracy          float64
review_scores_cleanliness       float64
review_scores_checkin           float64
review_scores_communication     float64
review_scores_location          float64
review_scores_value             float64
license                         float64
reviews_per_month               float64
dtype: object
```

In [9]:

```
##--- Check for non-numerical features stored as discrete numerical variables ---##

print(df.dtypes[df.dtypes =='int'])
```

```
id                                              int64
scrape_id                                       int64
host_id                                         int64
neighbourhood_cleansed                          int64
accommodates                                    int64
minimum_nights                                  int64
maximum_nights                                  int64
minimum_minimum_nights                          int64
maximum_minimum_nights                          int64
minimum_maximum_nights                          int64
maximum_maximum_nights                          int64
availability_30                                 int64
availability_60                                 int64
availability_90                                 int64
availability_365                                int64
number_of_reviews                               int64
number_of_reviews_ltm                           int64
number_of_reviews_l30d                          int64
calculated_host_listings_count                  int64
calculated_host_listings_count_entire_homes     int64
calculated_host_listings_count_private_rooms    int64
calculated_host_listings_count_shared_rooms     int64
dtype: object
```

In [10]:

```
##--- Ensure IDs are interpreted as objects ---#

# do not want an average of IDs
df[["id", "scrape_id", "host_id"]] = df[["id", "scrape_id", "host_id"]].astype("obje
```

c) Drop Irrelevant Features

Textual features such as *description, neighborhood_overview, name* and *host_name* are removed either due to irrelevance or the difficulty of encoding the features numerically without sacrificing interpretability. In addition, irrelevant features such as IDs, urls and scraping dates which have no explanatory relationship with the target variable *price* are removed.

In [11]:

```
##----Drop uninformative columns----##

uncorr_columns = ['id', 'listing_url', 'scrape_id', 'last_scraped', 'picture_url',
df = df.drop(uncorr_columns, axis=1)

##----Drop textual columns----##

textual_columns = ['description', 'neighborhood_overview', 'name', 'host_name']
df = df.drop(textual_columns, axis=1)
```

4.3 Missing Values

After the data types have been corrected, missing values are identified and handled. Data types are corrected before the missing value strategy is implemented in order to allow for imputation of values where appropriate. The proportion of missing values in each variable is explored in order to inform how best to handle the missing values. Missing values are then handled according to the following strategy:

- Firstly, any features with a proportion of missing values above a specified threshold are dropped.
- The remaining features then undergo imputation according to data type.

Imputation was chosen as the preferred missing data strategy due to its simplicity, flexibility and low computational expense. Median imputation was chosen in favour of mean imputation for numerical features in order to minimize the effects of outliers.

In [12]:

```
##--- All missing values denoted as NaN ---##

df.replace("?", np.nan, inplace = True) # replace "?" with NaN
df.replace(".", np.nan, inplace = True) # replace "." with NaN
df.replace("NaN", np.nan, inplace = True)
df.replace("nan", np.nan, inplace = True)
df.replace("", np.nan, inplace = True)

##--- Find number of missing data entries in each column ---##

#Find number of missing data entries in each column
missing_data = df.isnull()

for column in missing_data.columns.values.tolist():
    print(column)
    print (missing_data[column].value_counts())
    print("")
```

```
host_about
False    9134
True     7882
Name: host_about, dtype: int64

host_response_time
False    11271
True      5745
Name: host_response_time, dtype: int64

host_response_rate
False    11271
True      5745
Name: host_response_rate, dtype: int64

host_acceptance_rate
False    11928
True      5088
Name: host_acceptance_rate, dtype: int64
```

In [13]:

```
##--- Missing values in target variable ---##

missing_data["price"].value_counts()
```

Out[13]:

```
False    17016
Name: price, dtype: int64
```

We observe no missing values for our target variable *price*. The remaining predictors are categorised according to whether the proportion of missing data is greater than a certain threshold.

In [14]:

```
##--- Determine which features have missing values exceeding threshold ---##

above_thresh = [] #need to remove feature
below_thresh = [] #use imputations
not_missing = []
threshold = 0.3
for column in missing_data.columns.values.tolist():
    missing_data_counts = missing_data[column].value_counts()
    if(len(missing_data_counts) > 1):
        if((missing_data_counts[1]/(missing_data_counts[0]+missing_data_counts[1]))
            above_thresh.append(column)
        else:
            below_thresh.append(column)

    # 100% of data is missing
    elif(missing_data_counts.index[0] == True):
        above_thresh.append(column)

    else:
        not_missing.append(column)
print("Proportion of Missing Values Exceeds {}%:\n {}".format(threshold*100, above_t
print("Total: {}".format(len(above_thresh)))
print("\n")
print("Proportion of Missing Values Below {}%:\n {}".format(threshold*100, below_thr
print("Total: {}".format(len(below_thresh)))
print("\n")
print("No Missing Values: \n", not_missing)
print("Total: {}".format(len(not_missing)))
```

```
Proportion of Missing Values Exceeds 30.0%:
 ['host_about', 'host_response_time', 'host_response_rate', 'host_neig
hbourhood', 'neighbourhood', 'neighbourhood_group_cleansed', 'bathroom
s', 'license']
Total: 8


Proportion of Missing Values Below 30.0%:
 ['host_acceptance_rate', 'host_is_superhost', 'host_listings_count',
'host_total_listings_count', 'host_has_profile_pic', 'host_identity_ve
rified', 'bathrooms_text', 'bedrooms', 'beds', 'review_scores_rating',
'review_scores_accuracy', 'review_scores_cleanliness', 'review_scores_
checkin', 'review_scores_communication', 'review_scores_location', 're
view_scores_value', 'reviews_per_month']
Total: 17


No Missing Values:
 ['neighbourhood_cleansed', 'latitude', 'longitude', 'property_type',
'room_type', 'accommodates', 'amenities', 'price', 'minimum_nights',
'maximum_nights', 'minimum_minimum_nights', 'maximum_minimum_nights',
'minimum_maximum_nights', 'maximum_maximum_nights', 'minimum_nights_av
g_ntm', 'maximum_nights_avg_ntm', 'has_availability', 'availability_3
0', 'availability_60', 'availability_90', 'availability_365', 'number_
of_reviews', 'number_of_reviews_ltm', 'number_of_reviews_l30d', 'insta
nt_bookable', 'calculated_host_listings_count', 'calculated_host_listi
ngs_count_entire_homes', 'calculated_host_listings_count_private_room
s', 'calculated_host_listings_count_shared_rooms']
Total: 29
```

Missing Value Strategy

The strategy for dealing with missing values is outlined as follows:

- We observe that for our target variable, AirBnB price, there are no missing values and therefore no further processing is necessary.
- Any missing numerical variables were imputed using *median imputation*
- In contrast, any missing categorical variables such as binary variables were imputed using *mode imputation*

In [15]:

```
##----Imputation----##

# Drop columns with missing values above the threshold
df = df.drop(above_thresh, axis=1) # make sure this is the right axis

# For features below threshold = separate into numerical and binary
numerical = ['bathrooms_text', 'bedrooms', 'beds', 'host_listings_count', 'host_tota
category = ['host_is_superhost', 'host_has_profile_pic', 'host_identity_verified',

numerical = list(numerical)
category = list(category)

#Replace numerical variables with median
for column in numerical:
    med = df[column].astype("float").median(axis=0)
    df[column] = df[column].replace(np.nan, med)

#Replace binary/categorical variables with mode
for column in category:
    mode = df[column].mode()
    df[column] = df[column].replace(np.nan, mode[0])
```

Sense Checks

In [16]:

```
##--- Find number of missing data entries in each column ---##

#Find number of missing data entries in each column
missing_data = df.isnull()

for column in missing_data.columns.values.tolist():
    missing_data_counts = missing_data[column].value_counts()
    if(len(missing_data_counts) > 1):
        print(column)
        print(missing_data_counts)
        print("{}%".format(missing_data_counts[1]/(missing_data_counts[0]+missing_da
        print("")
```

4.4 Data Transformation

In order to convert all predictors into appropriate numerical input for the later regression models, data transformation was performed to encode categorical features and binary features. In addition, data aggregation was performed on features containing highly granular information in order to increase predictive ability.

a) Categorical Features

In order to numerically encode categorical variables such as *room_type*, one hot encoding was utilised. One hot encoding was selected in preference to categorical encoding since the categories have no natural order and therefore should not have ordinal relationships between them in their representation. However, a disadvantage of one hot encoding is the resulting sparsity, especially in the presence of a large number of categories. Although some methods suggest the use of PCA to reduce sparsity, this would compromise explainability of the model.

In [17]:

```
#--- One Hot Encoding ---#

#room type
dummy_variable_1=pd.get_dummies(df["room_type"])
dummy_variable_1.rename(columns={'Entire home/apt':'entire', "Hotel room":'hotel',
df = pd.concat([df, dummy_variable_1], axis=1)
df[["room_type", "entire", "hotel", "private", "shared"]].head() # verify new varial
```

Out[17]:

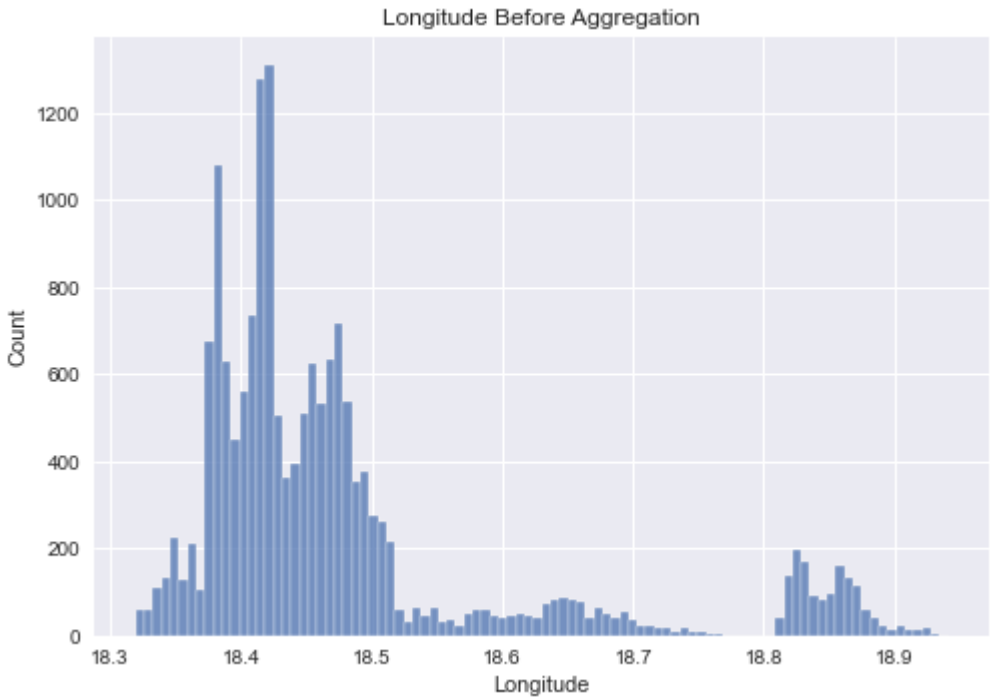
	room_type	entire	hotel	private	shared
0	Entire home/apt	1	0	0	0
1	Entire home/apt	1	0	0	0
2	Entire home/apt	1	0	0	0
3	Private room	0	0	1	0
4	Entire home/apt	1	0	0	0

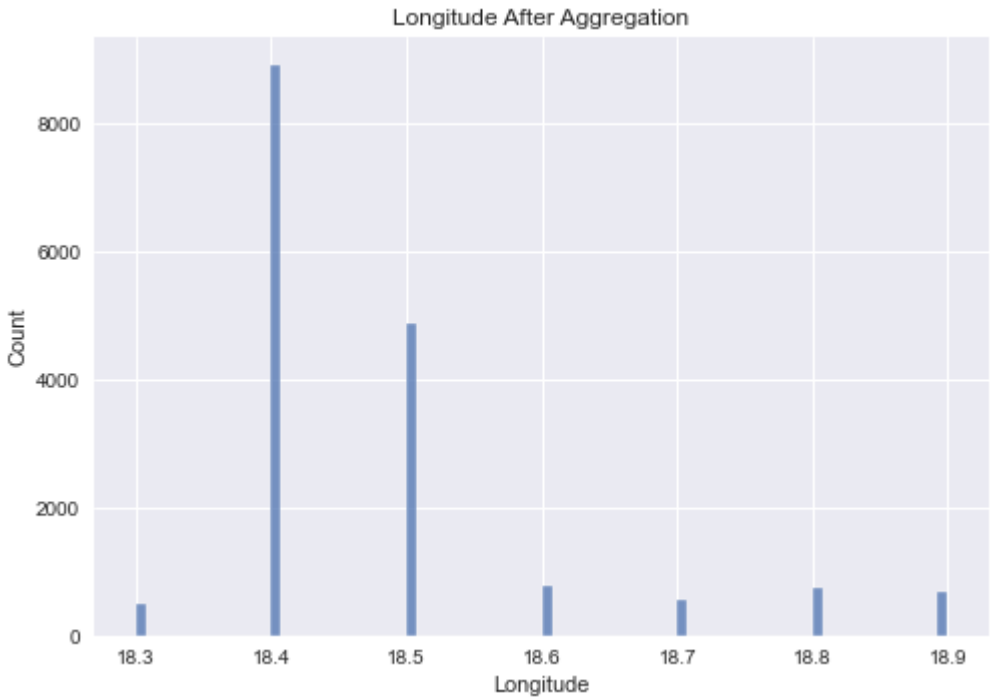
b) Data Aggregation

Due to the granularity of the variables pertaining to location such as *latitude* and *longitude*, data aggregation is performed to allow for coarser granularity of these features.

In [18]:

```
#--- Data Aggregation ---#

sns.set(rc = {'figure.figsize':(11,6)})

# longitude
plt.style.use('seaborn')
sns_plot=sns.histplot(df["longitude"]);
sns_plot.set(xlabel = "Longitude", ylabel = "Count")
plt.title("Longitude Before Aggregation")
plt.show()


# Perform aggregation by rounding off the longitude and latitude

df["longitude"] = np.around(df["longitude"], 1)
df["latitude"] = np.around(df["latitude"], 1)

sns_plot=sns.histplot(df["longitude"]);
sns_plot.set(xlabel = "Longitude", ylabel = "Count")
plt.title("Longitude After Aggregation")
plt.show()
```





c) Binary Features

Binary features are re-coded from true and false values to a corresponding 1 and 0 mapping.

In [19]:

```
##----Process Binary Features----##

#change binary t/f columns to 0 and 1
columns =  ['has_availability', 'host_is_superhost', 'host_identity_verified', 'host
                  'instant_bookable']
for c in columns:
    df[c] = df[c].replace('f',0,regex=False)
    df[c] = df[c].replace('t',1,regex=False)
```

d) Validate Changes

In [20]:

```
# check correct data types by calculating summary statistic
df.describe()
```

Out[20]:

	host_acceptance_rate	host_is_superhost	host_listings_count	host_total_listings_count	hos
count	17016.000000	17016.000000	17016.000000	17016.000000	
mean	83.475846	0.238070	9.118535	9.118535	
std	26.439498	0.425915	40.201709	40.201709	
min	0.000000	0.000000	0.000000	0.000000	
25%	85.000000	0.000000	1.000000	1.000000	
50%	94.000000	0.000000	2.000000	2.000000	
75%	99.000000	0.000000	5.000000	5.000000	
max	100.000000	1.000000	2063.000000	2063.000000	

4.5 Error Identification

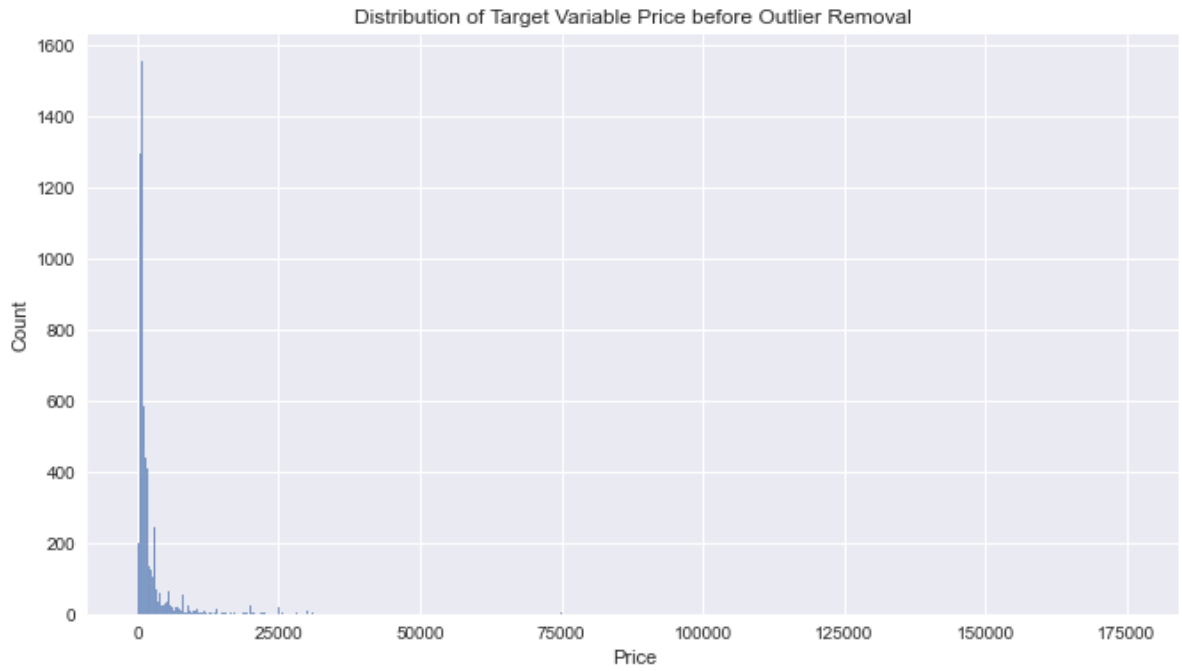
The distributions of the target and predictor variables were examined to check for any possible errors in the dataset.

Target Variable Exploration and Visualisation

The distribution of the target variable *price* was visualised in order to identify any potential sources of error.

In [21]:

```
# Visualise target variable

plt.style.use('seaborn')
plt.rcParams["figure.figsize"] = (11,6)

sns_plot=sns.histplot(df["price"]);
sns_plot.set(xlabel = "Price", ylabel = "Count")
plt.title("Distribution of Target Variable Price before Outlier Removal")
plt.show()
```



Handling Outliers

Several large outliers were identified in the target *price* variable as shown above, resulting in a highly skewed distribution.

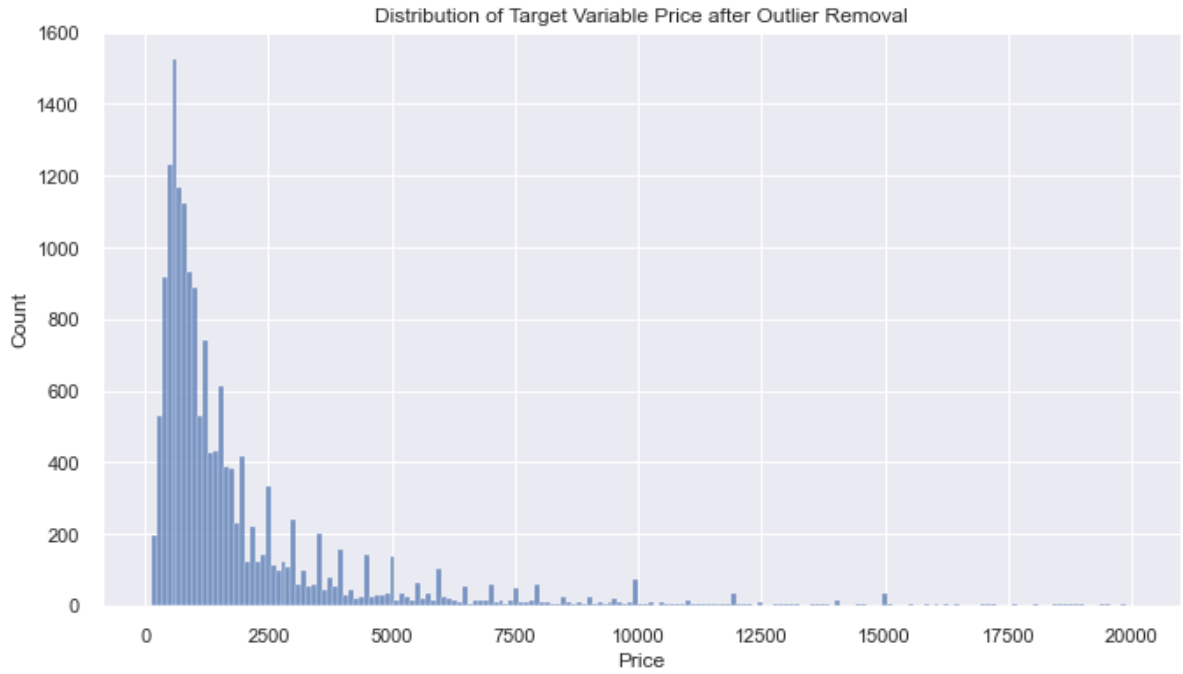
In [22]:

```
##--Handling Outliers--##

#1. Std. dev.
avg_price = df['price'].astype("float").mean(axis=0)
std_dev = df['price'].astype("float").std(axis= 0)
outliers = []
remainder_std = []
threshold = avg_price + 4*std_dev #originally 3 std dev
print("Threshold for Outliers: R",threshold)
for entry in df["price"]:
    if(entry > threshold):
        outliers.append(entry)
    else:
        remainder_std.append(entry)
print("Number of outliers: ", len(outliers))

#2. 99th percentile
percentile_price = df['price'].quantile(.99)
print("Threshold for Outliers: R",percentile_price)
outliers_percentile = []
remainder = []
for entry in df["price"]:
    if(entry > percentile_price):
        outliers_percentile.append(entry)
    else:
        remainder.append(entry)
print("Number of outliers: ", len(outliers_percentile))
```

```
Threshold for Outliers: R 25023.01630168873
Number of outliers:  101
Threshold for Outliers: R 20000.0
Number of outliers:  163
```

In [23]:

```
##----Remove/Retain Outliers----##

df = df.loc[~(df['price'] == 0)] #Remove rows with price value of zero (target)
remove_outliers = 1

#If remove_outliers is set at top of notebook
if(remove_outliers):
    df = df.loc[(df['price'] < percentile_price)] #remove outliers according to 99tl


# visualise target variable without outliers
sns.set(rc = {'figure.figsize':(11,6)})

sns_plot=sns.histplot(df["price"])
sns_plot.set(xlabel = "Price", ylabel = "Count")
plt.title("Distribution of Target Variable Price after Outlier Removal")
plt.show()
```

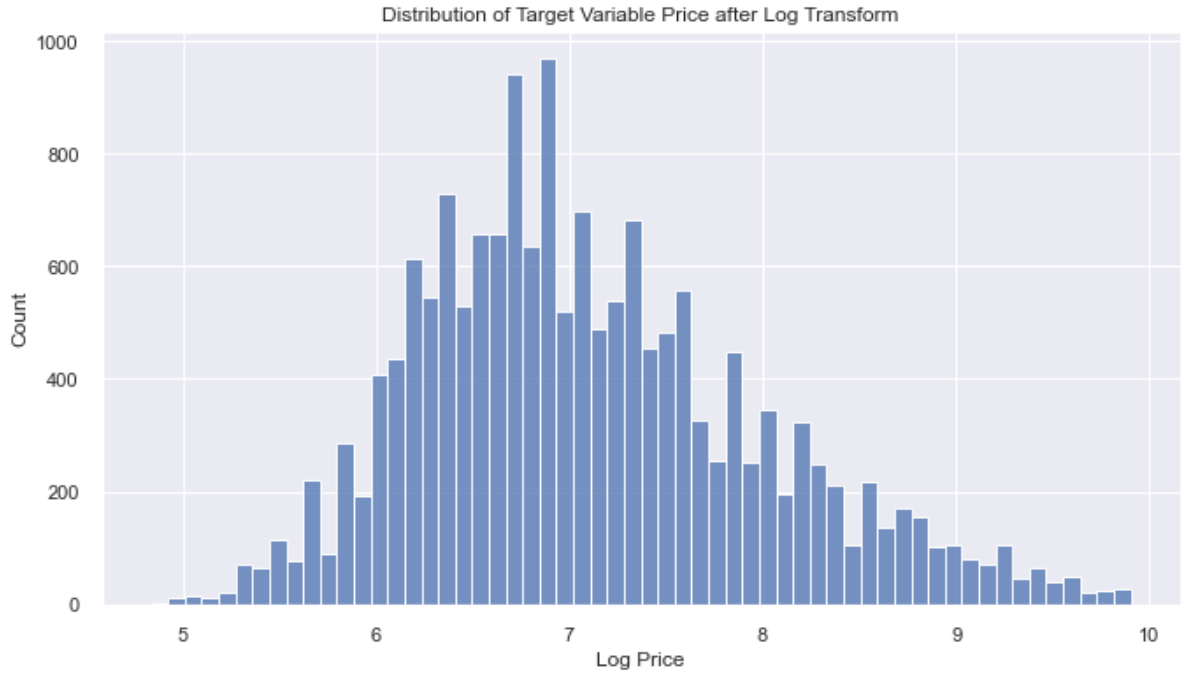


Reducing Skew

In addition, a log transform was applied to the target variable to reduce skew, as shown below. During modelling, the resulting *log price* was used to determine whether there are any further benefits to utilising the transformed *log price* rather than original *price* as the target variable.

In [24]:

```
##---- Reducing Skewness----##

#Log Transform
df["log_price"] = np.log(df["price"])
sns_plot=sns.histplot(df["log_price"])
sns_plot.set(xlabel = "Log Price", ylabel = "Count")
plt.title("Distribution of Target Variable Price after Log Transform")
plt.show()
```



In [25]:

```
##----Probability Plot----##

ax1 =  plt.subplot(221)
s1 = stats.probplot(df["price"], dist=stats.norm, plot=ax1)
ax1.set_title("Probability Plot of Price")
ax2 =  plt.subplot(222)
s2 = stats.probplot(df["log_price"], dist=stats.norm, plot=ax2)
ax2.set_title("Probability Plot of Log Price")
plt.show()
```

Predictor Variable Exploration and Visualisation

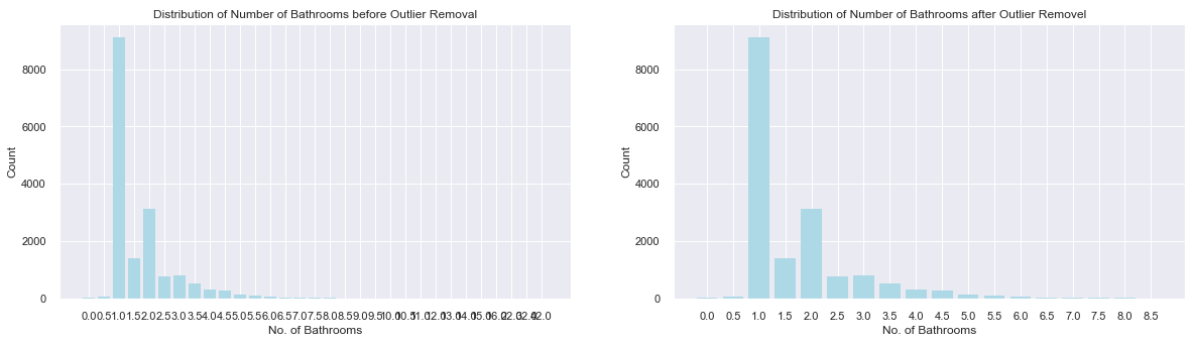
In addition to the target variable, any outliers found in the predictor variables were removed to prevent distortion of the variable standardisation process required before model selection.

Number of Bathrooms

In [26]:

```
# --- Distribution before outlier handling ---##


plt.rcParams["figure.figsize"] = (20,5)
fig, ax1 = plt.subplots(1,2)

acc = df["bathrooms_text"].value_counts().sort_index()
acc.index = acc.index.map(str)
ax1[0].bar(acc.index, acc.values, color = "lightblue",  linewidth=0,  width=0.8, ali
ax1[0].set_xlabel("No. of Bathrooms")
ax1[0].set_ylabel("Count")
ax1[0].set_title("Distribution of Number of Bathrooms before Outlier Removal")
#ax1.show()

##--- Outlier handling ---##

# Using standard deviation
avg_price = df['bathrooms_text'].astype("float").mean(axis=0)
std_dev = df['bathrooms_text'].astype("float").std(axis= 0)
outliers = []
remainder_std = []
threshold = avg_price + 6*std_dev #originally 3 std dev
print("Threshold for Outliers: ",threshold)
for entry in df["bathrooms_text"]:
    if(entry > threshold):
        outliers.append(entry)
    else:
        remainder_std.append(entry)
print("Number of outliers: ", len(outliers))
df = df.loc[(df['bathrooms_text'] < threshold)] #remove outliers

##--- Distribution after outlier handling ---##

acc = df["bathrooms_text"].value_counts().sort_index()
acc.index = acc.index.map(str)

ax1[1].bar(acc.index, acc.values, color = "lightblue",  linewidth=0,  width=0.8, ali
ax1[1].set_xlabel("No. of Bathrooms")
ax1[1].set_ylabel("Count")
ax1[1].set_title("Distribution of Number of Bathrooms after Outlier Removel")
plt.show()
```

```
Threshold for Outliers:  8.921986691076722
Number of outliers:  27
```



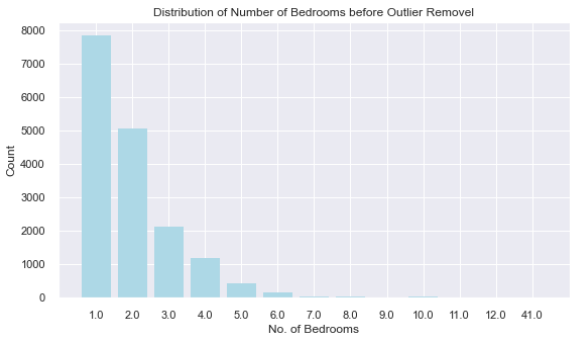
Number of Bedrooms

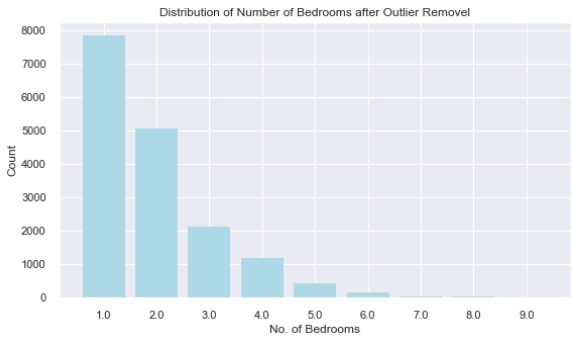
Similarly, we observe an outlier listing with 41 bedrooms which deviates significantly from the rest of the distribution.

In [27]:

```
##--- Distribution before outlier handling ---##

plt.rcParams["figure.figsize"] = (20,5)
fig, ax1 = plt.subplots(1,2)

acc = df["bedrooms"].value_counts().sort_index()
acc.index = acc.index.map(str)
ax1[0].bar(acc.index, acc.values, color = "lightblue",  linewidth=0,  width=0.8, ali
ax1[0].set_xlabel("No. of Bedrooms")
ax1[0].set_ylabel("Count")
ax1[0].set_title("Distribution of Number of Bedrooms before Outlier Removel")
#plt.show()

##--- Outlier handling ---##

# Using standard deviation
avg_price = df['bedrooms'].astype("float").mean(axis=0)
std_dev = df['bedrooms'].astype("float").std(axis= 0)
outliers = []
remainder_std = []
threshold = avg_price + 6*std_dev #originally 3 std dev
print("Threshold for Outliers: ",threshold)
for entry in df["bedrooms"]:
    if(entry > threshold):
        outliers.append(entry)
    else:
        remainder_std.append(entry)
print("Number of outliers: ", len(outliers))
df = df.loc[(df['bedrooms'] < threshold)] #remove outliers

##--- Distribution after outlier handling ---##

acc = df["bedrooms"].value_counts().sort_index()
acc.index = acc.index.map(str)

ax1[1].bar(acc.index, acc.values, color = "lightblue",  linewidth=0,  width=0.8, ali
ax1[1].set_xlabel("No. of Bedrooms")
ax1[1].set_ylabel("Count")
ax1[1].set_title("Distribution of Number of Bedrooms after Outlier Removel")
plt.show()
```

```
Threshold for Outliers:  9.283941083866257
Number of outliers:  20
```





4.6 Sense Checks

The data transformations and manipulations performed were then validated through a number of checks. Firstly, each feature was checked to ensure the final data types was as expected, as well as whether there were any missing values still present. Both of these tasks were accomplished below.

In [28]:

```
# this should indicate correct types, new variables created AND no missing values by

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16786 entries, 0 to 17015
Data columns (total 51 columns):
 #   Column                                      Non-Null Count  Dty
pe
---  ------                                      --------------  ---
--
 0   host_acceptance_rate                        16786 non-null  flo
at64
 1   host_is_superhost                           16786 non-null  int
64
 2   host_listings_count                         16786 non-null  flo
at64
 3   host_total_listings_count                   16786 non-null  flo
at64
 4   host_has_profile_pic                        16786 non-null  int
64
 5   host_identity_verified                      16786 non-null  int
64
 6   neighbourhood_cleansed                      16786 non-null  int
64
 7   latitude                                    16786 non-null  flo
at64
 8   longitude                                   16786 non-null  flo
at64
 9   property_type                               16786 non-null  obj
ect
 10  room_type                                   16786 non-null  obj
ect
 11  accommodates                                16786 non-null  int
64
 12  bathrooms_text                              16786 non-null  flo
at64
 13  bedrooms                                    16786 non-null  flo
at64
 14  beds                                        16786 non-null  flo
at64
 15  amenities                                   16786 non-null  obj
ect
 16  price                                       16786 non-null  flo
at64
 17  minimum_nights                              16786 non-null  int
64
 18  maximum_nights                              16786 non-null  int
64
 19  minimum_minimum_nights                      16786 non-null  int
64
 20  maximum_minimum_nights                      16786 non-null  int
64
 21  minimum_maximum_nights                      16786 non-null  int
64
 22  maximum_maximum_nights                      16786 non-null  int
64
 23  minimum_nights_avg_ntm                      16786 non-null  flo
at64
```

```
 24  maximum_nights_avg_ntm                          16786 non-null  flo
at64
 25  has_availability                                16786 non-null  int
64
 26  availability_30                                 16786 non-null  int
64
 27  availability_60                                 16786 non-null  int
64
 28  availability_90                                 16786 non-null  int
64
 29  availability_365                                16786 non-null  int
64
 30  number_of_reviews                               16786 non-null  int
64
 31  number_of_reviews_ltm                           16786 non-null  int
64
 32  number_of_reviews_l30d                          16786 non-null  int
64
 33  review_scores_rating                            16786 non-null  flo
at64
 34  review_scores_accuracy                          16786 non-null  flo
at64
 35  review_scores_cleanliness                       16786 non-null  flo
at64
 36  review_scores_checkin                           16786 non-null  flo
at64
 37  review_scores_communication                     16786 non-null  flo
at64
 38  review_scores_location                          16786 non-null  flo
at64
 39  review_scores_value                             16786 non-null  flo
at64
 40  instant_bookable                                16786 non-null  int
64
 41  calculated_host_listings_count                  16786 non-null  int
64
 42  calculated_host_listings_count_entire_homes     16786 non-null  int
64
 43  calculated_host_listings_count_private_rooms    16786 non-null  int
64
 44  calculated_host_listings_count_shared_rooms     16786 non-null  int
64
 45  reviews_per_month                               16786 non-null  flo
at64
 46  entire                                          16786 non-null  uin
t8
 47  hotel                                           16786 non-null  uin
t8
 48  private                                         16786 non-null  uin
t8
 49  shared                                          16786 non-null  uin
t8
 50  log_price                                       16786 non-null  flo
at64
dtypes: float64(20), int64(24), object(3), uint8(4)
memory usage: 6.2+ MB
```

Furthermore, summary statistics before and after the data cleaning process were calculated and compared. This was particurly helpful in determining the effects of missing data handling. In addition, the presence of newly created variables such as those created through one hot encoding was verified.

In [29]:

```
# compare summary statistics before and after transformations
df.describe()
```

Out[29]:

	host_acceptance_rate	host_is_superhost	host_listings_count	host_total_listings_count	hos
count	16786.000000	16786.000000	16786.000000	16786.000000	
mean	83.705290	0.238234	9.048195	9.048195	
std	26.155051	0.426016	39.894102	39.894102	
min	0.000000	0.000000	0.000000	0.000000	
25%	85.000000	0.000000	1.000000	1.000000	
50%	94.000000	0.000000	2.000000	2.000000	
75%	99.000000	0.000000	5.000000	5.000000	
max	100.000000	1.000000	2063.000000	2063.000000	

We observe that some means differ between the summary statistics before and after data wrangling as expected due to the removal of outliers.

4.7 Data Splitting

Before data exploration was performed, the dataset was split into training and test sets according to a $70/30$ ratio to prevent any data leakage.

Justification

Splitting the data into selection and inference sets is essential to eliminate the dependency that occurs when performing model selection and inference on the same dataset. Therefore data splitting allows us to imitate the classical paradigm by first selecting a single model through model selection procedures and then fitting the model on the inference set. Moreover, testing a single model rather than multiple models significantly reduces the number of alternative hypothesis test performed and therefore limits the effects of multiple testing. Lastly, although data splitting can have a detrimental effect on the quality of the selected model and also reduce the power of the statistical tests, since the number of observations in the dataset is large compared to the number of variables, data splitting remains appropriate in our scenario. We have 11750 observations and 5036 observations in our selection and inference sets respectively, with 51 variables following data wrangling.

In [30]:

```
from sklearn.model_selection import train_test_split

df_selection, df_inference = train_test_split(df, test_size=0.3, random_state=42)
# Size of train and test set
print("Size of the selection set {}".format(df_selection.shape))
print("Size of the inference set {}".format(df_inference.shape))
df_selection.head(3)
```

```
Size of the selection set (11750, 51)
Size of the inference set (5036, 51)
```

Out[30]:

	host_acceptance_rate	host_is_superhost	host_listings_count	host_total_listings_count	host
965	88.0	1	0.0	0.0	
476	0.0	0	1.0	1.0	
2310	0.0	0	1.0	1.0	

5. Exploratory Analyses

In the following section, we explore the pairwise relationships between the predictors and the target *price* variable. A summary of some of the most pertinent visualisations is included below. Each pairwise relationship is examined in light of how the predictor affects *price* as well as the actual distribution of the predictor, indicating the amount of supporting evidence at each value.

Analysis is split into the following sections:

- Intrinsic Characteristics of the Listing
- Reviews
- Location

Intrinsic Characteristics of the Listing

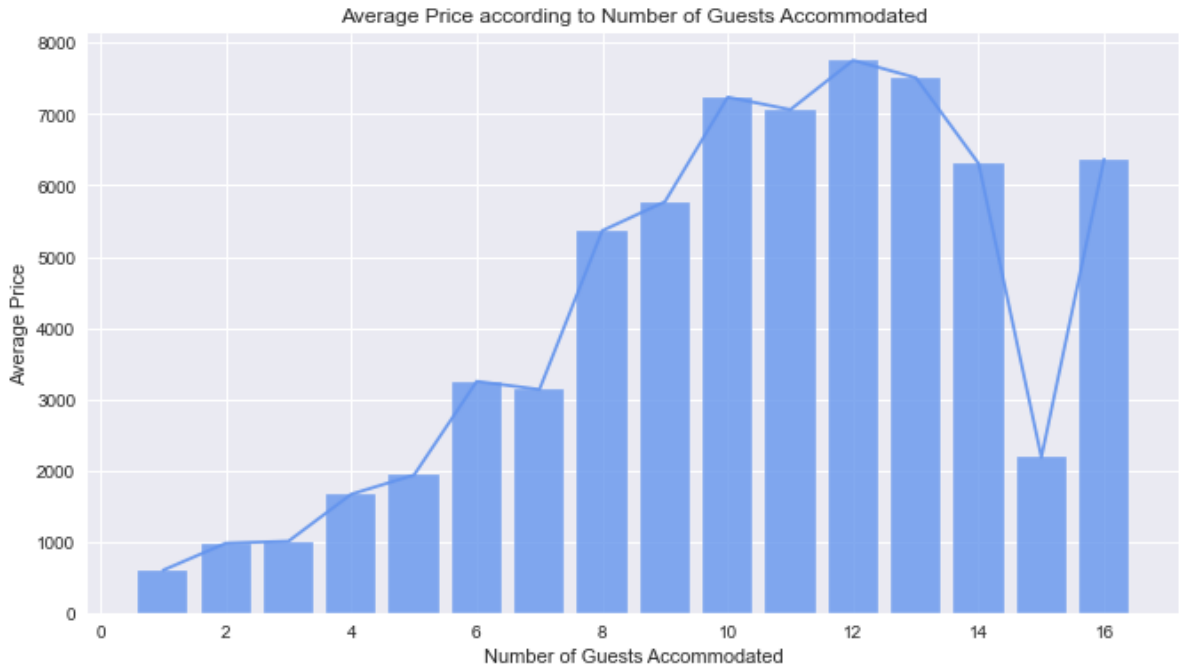
Number of Guests Accommodated

The number of guests accommodated is shown to have a strong positive correlation with price. This is indicated in the graph below which shows the average price according to the number of guests accommodated. The trend is no longer as certain after 12 guests is exceeded. However, as indicated in the following figure which includes the supporting evidence, the number of AirBnBs with guest numbers exceeding

12 drops off rapidly. Therefore the supporting evidence is weaker above 12 guests and therefore the trend for guest numbers above 12 is less certain. This suggests that a positive linear relationship is still appropriate. This observation also makes sense from an intuitive point of view because larger properties able to house a high number of guests are more rare. Therefore *accommodates* is anticipated to have a strong relationship with the target *price* variable.

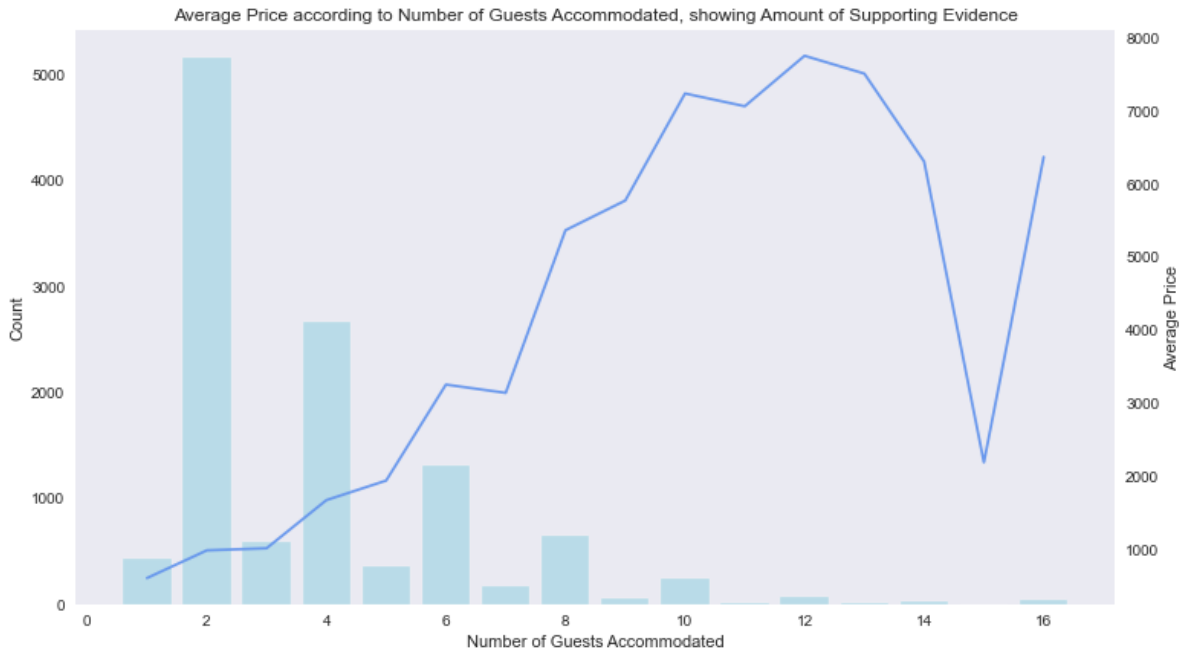
In [31]:

```
##--- Average price according to number of guests accomodated ---##

plt.style.use('seaborn')
plt.rcParams["figure.figsize"] = (11,6)
price_by_accommodates = df_selection.groupby("accommodates")[["price"]].mean()
plt.plot(price_by_accommodates, color = "cornflowerblue", alpha = 0.95)
plt.bar(price_by_accommodates.index.values, price_by_accommodates["price"].values, c
plt.xlabel("Number of Guests Accommodated")
plt.ylabel("Average Price")
plt.title("Average Price according to Number of Guests Accommodated")
plt.show()
```



In [32]:

```
##--- Average price and distribution of number of guests accomodated ---##

plt.style.use('seaborn')
plt.rcParams["figure.figsize"] = (11,6)
fig, ax1 = plt.subplots()
acc = df_selection["accommodates"].value_counts()
color = 'lightsalmon'
ax1.set_xlabel('Number of Guests Accommodated')
ax1.set_ylabel('Count')
ax1.bar(acc.index, acc.values, color = "lightblue", alpha = 0.8)
ax1.tick_params(axis='y')

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis
color = 'lightblue'
ax2.set_ylabel('Average Price')  # we already handled the x-label with ax1
ax2.plot(price_by_accommodates, color = "cornflowerblue", alpha = 0.95)
ax2.tick_params(axis='y')
ax1.grid(False)
ax2.grid(False)
fig.tight_layout()  # otherwise the right y-label is slightly clipped
plt.title("Average Price according to Number of Guests Accommodated, showing Amount
plt.show()
```

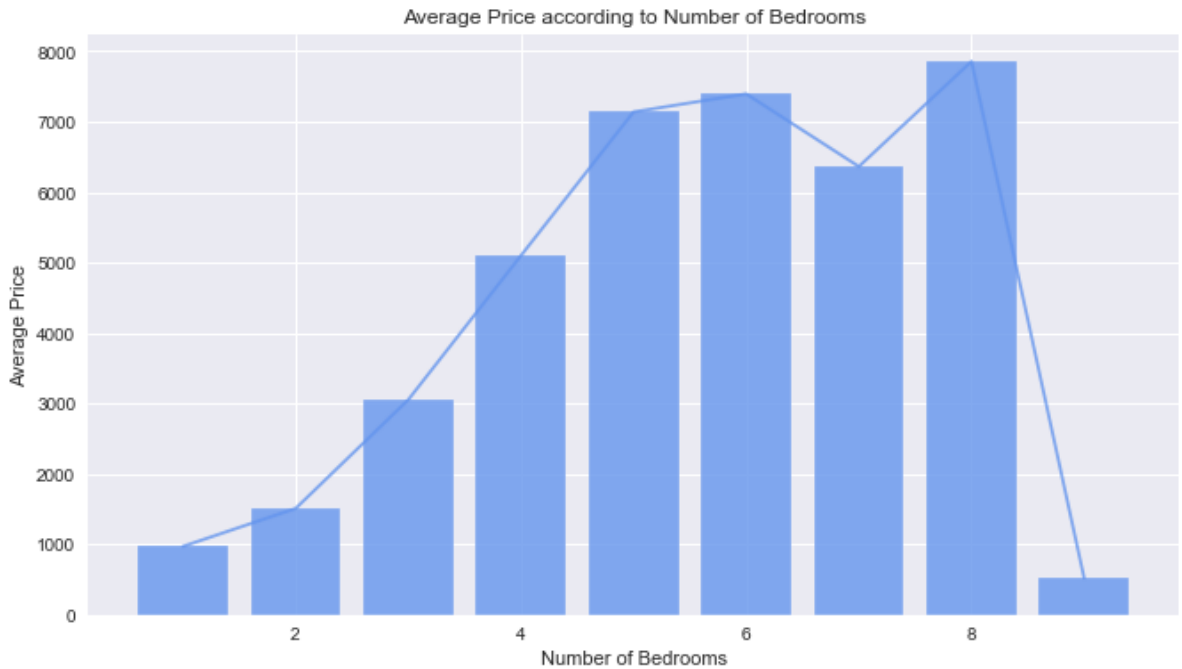


Number of Bedrooms

It is observed that the number of bedrooms has a similar strong positive relationship with price. Since both the number of bedrooms and the number of guests accommodated relate to the size of the property, this further reinforces the idea that size will be a strong predictor. Moreover, in a similar manner, the amount of supporting evidence for the average price decreases as the number of bedrooms increases. Therefore we have less confidence in the downward trend in price observed after 6 bedrooms. Overall, this indicates a strong linear relationship between number of bedrooms and price.

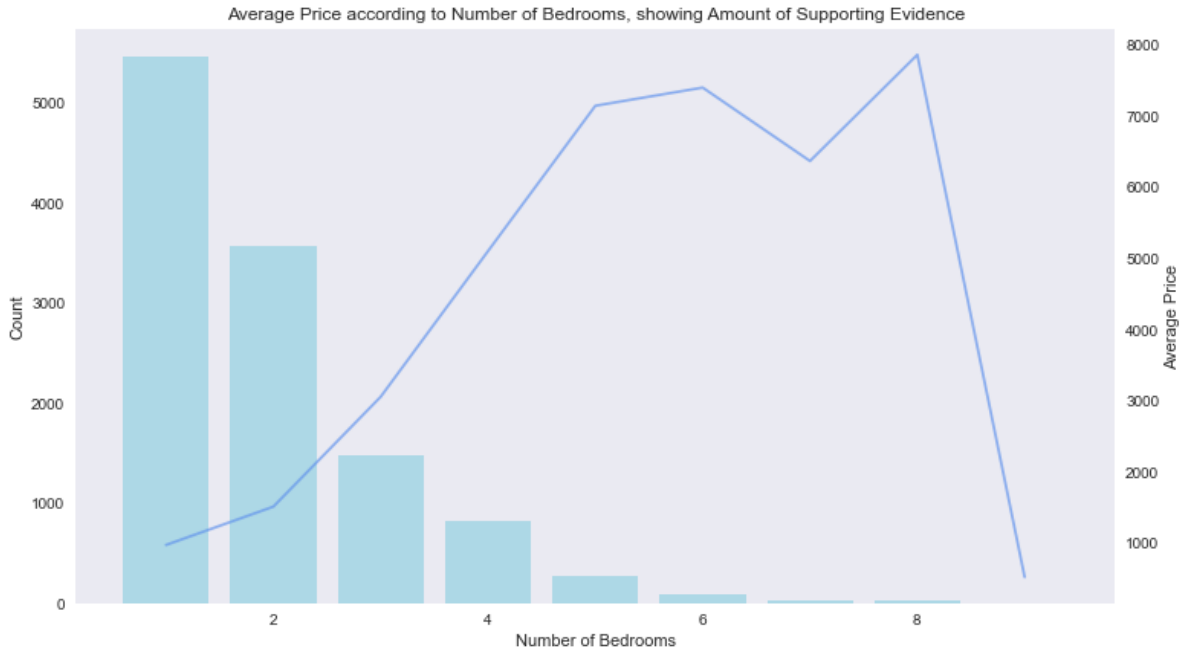
In [33]:

```
plt.style.use('seaborn')
plt.rcParams["figure.figsize"] = (11,6)

price_by_bedrooms = df_selection.groupby("bedrooms")[["price"]].mean()
plt.plot(price_by_bedrooms.index.values, price_by_bedrooms["price"].values, color =
plt.bar(price_by_bedrooms.index.values, price_by_bedrooms["price"].values, color = '
plt.xlabel("Number of Bedrooms")
plt.ylabel("Average Price")
plt.title("Average Price according to Number of Bedrooms")
plt.show()
```



In [34]:

```
fig, ax1 = plt.subplots()
acc = df_selection["bedrooms"].value_counts()
color = 'lightsalmon'
ax1.set_xlabel('Number of Bedrooms')
ax1.set_ylabel('Count')
ax1.bar(acc.index, acc.values, color = "lightblue",  linewidth=0,  width=0.8, align=
ax1.tick_params(axis='y')

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

color = 'lightblue'
ax2.set_ylabel('Average Price')  # we already handled the x-label with ax1
ax2.plot(price_by_bedrooms.index.values, price_by_bedrooms["price"].values, color =
ax2.tick_params(axis='y')
ax1.grid(False)
ax2.grid(False)
fig.tight_layout()  # otherwise the right y-label is slightly clipped
plt.title("Average Price according to Number of Bedrooms, showing Amount of Supporti
plt.show()
```

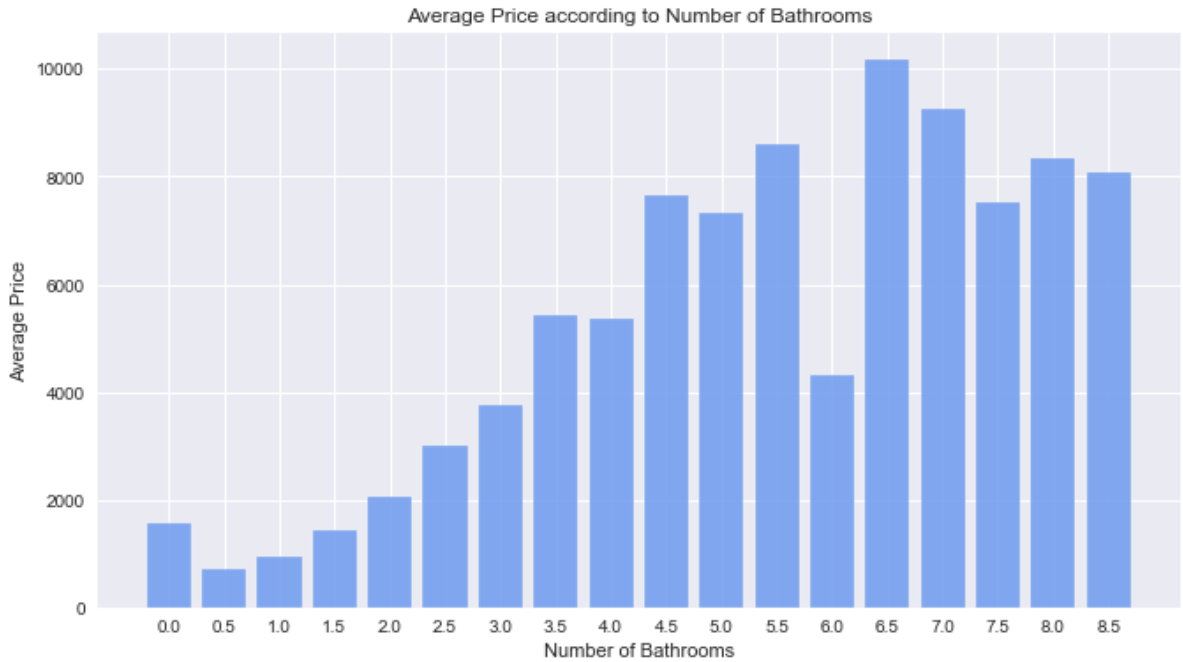


Number of Bathrooms

The number of bathrooms once again follows a similar pattern to guests accommodated and number of bedrooms. Since these three intrinsic characteristics of the are highly linked, complexities stemming from collinearity may affect the estimated coefficient values. These complexities are explored further on in the study.

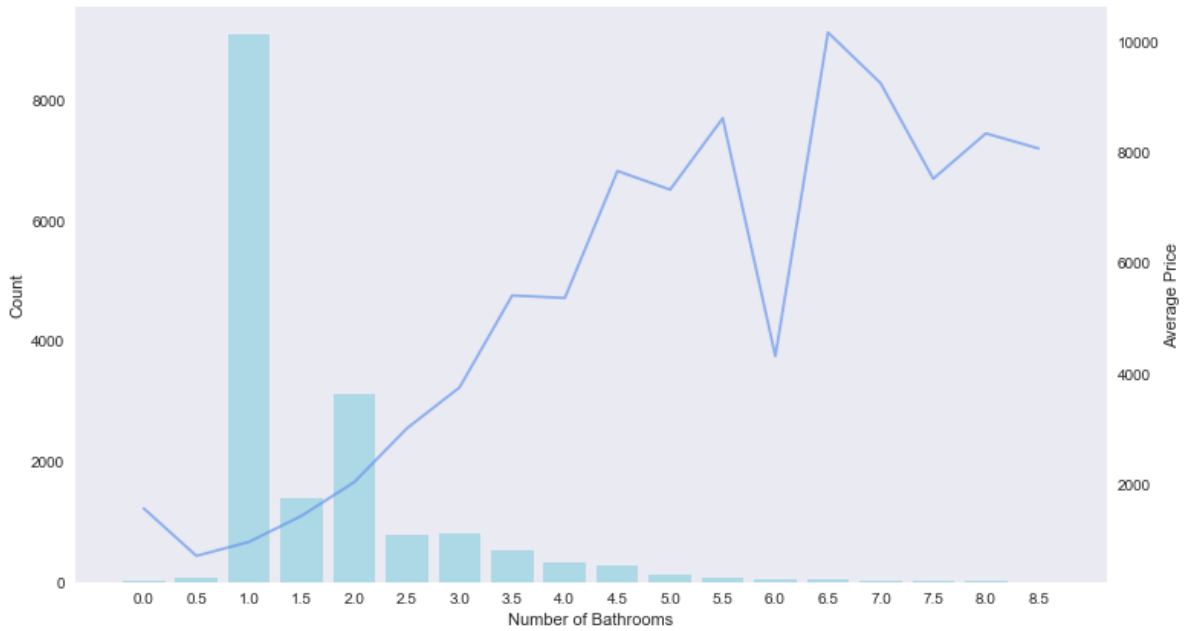
In [35]:

```
price_by_bathrooms = df_selection.groupby("bathrooms_text")[["price"]].mean()


price_by_bathrooms.index = price_by_bathrooms.index.map(str)

plt.bar(price_by_bathrooms.index.values, price_by_bathrooms["price"].values, color =
plt.xlabel("Number of Bathrooms")
plt.ylabel("Average Price")
plt.title("Average Price according to Number of Bathrooms")
plt.show()
```



In [36]:

```
fig, ax1 = plt.subplots()
acc = df["bathrooms_text"].value_counts().sort_index()
acc.index = acc.index.map(str)
ax1.set_xlabel('Number of Bathrooms')
ax1.set_ylabel('Count')
ax1.bar(acc.index, acc.values, color = "lightblue",  linewidth=0,  width=0.8, align=
ax1.tick_params(axis='y')

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis
color = 'lightblue'
ax2.set_ylabel('Average Price')  # we already handled the x-label with ax1
ax2.plot(price_by_bathrooms.index.values, price_by_bathrooms["price"].values, color
ax2.tick_params(axis='y')
ax1.grid(False)
ax2.grid(False)
fig.tight_layout()
plt.show()
```



Room Type

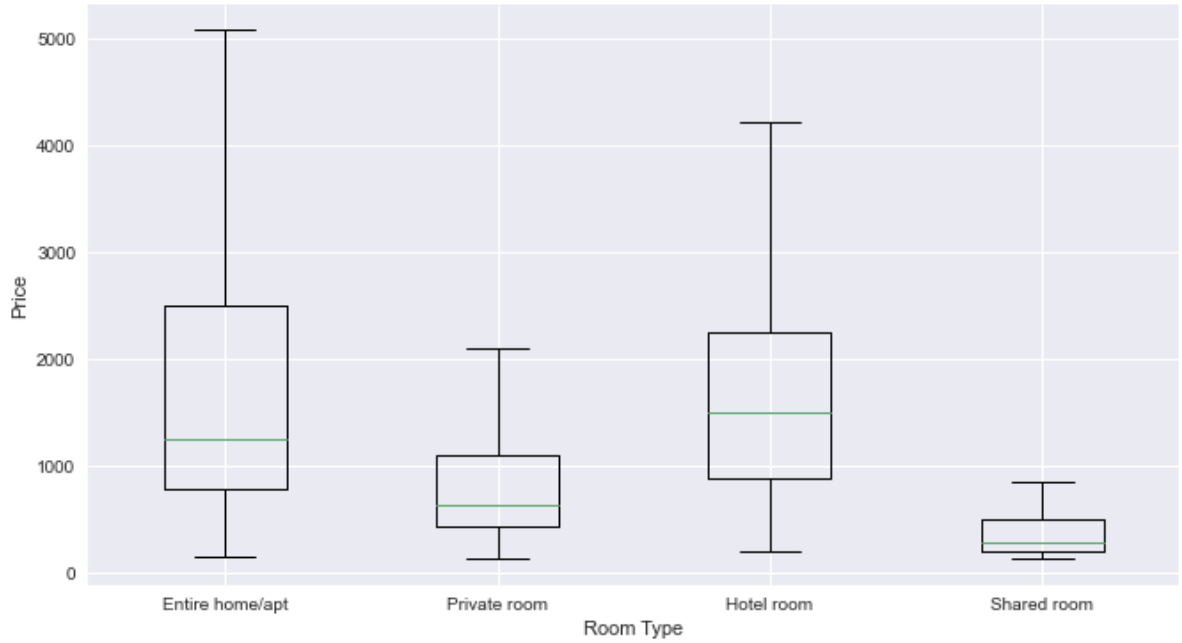
In contrast to numerical variables focusing on property size, the categorical variable *room_type* can be explored through the use of a box plot indicating the range of prices per room type, as shown in graph below. From analysis of the box plot, \textit{room_type} would appear to be a reasonably strong predictor since the distributions vary across the room types.

However, through observation of the number of samples for each room type in the dataset, it is evident that the predictor will be less helpful since there are so few hotel and shared rooms. This can be seen in the second figure below. Therefore, although the price differs according to *room_type*, the dataset is mainly dominated by two room types and therefore the averages for the hotel and shared room categories are based on a very small number of samples. Once more, it is intuitive that there would be less hotel rooms present in the dataset due to the AirBnB model of hosting unique and residential spaces as an alternative to hotels.

Therefore it is observed that both the relationship between the predictor and the target and the predictor distribution itself need to be considered when ascertaining whether a certain predictor may be valuable in explaining the distribution of the target *price* variable.

In [37]:

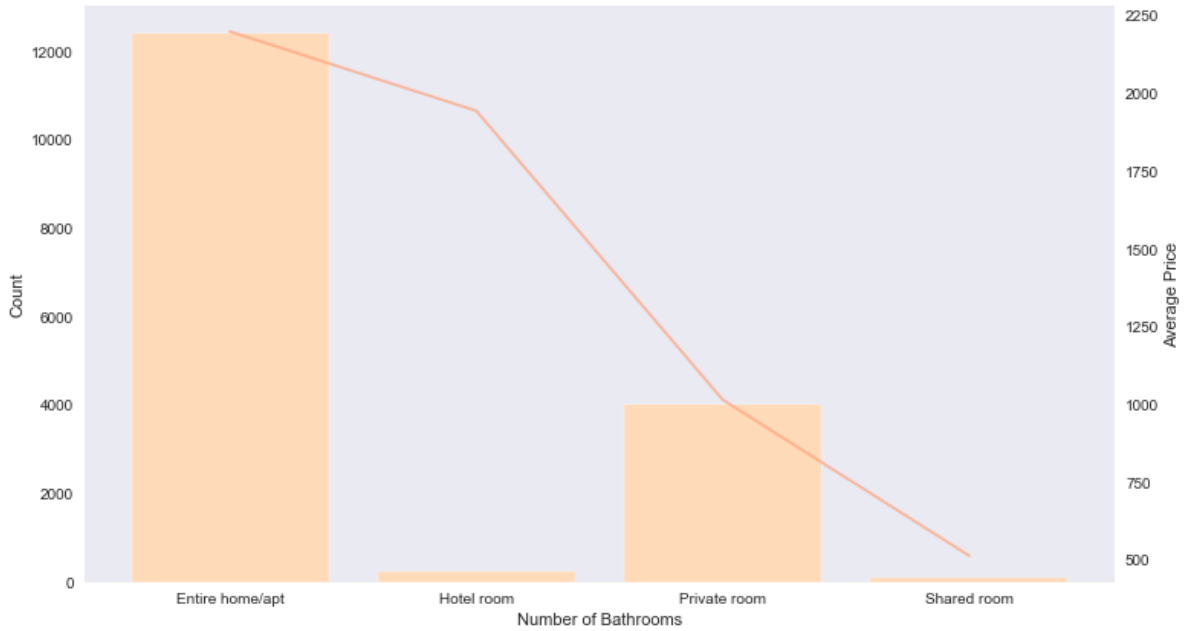
```
price_by_roomtype_series = [df_selection[df_selection["room_type"]=="Entire home/apt
plt.boxplot(price_by_roomtype_series, showfliers=False) # NB fliers/outliers removed
plt.xticks([1, 2, 3, 4], ['Entire home/apt', 'Private room', 'Hotel room', 'Shared r
plt.xlabel("Room Type")
plt.ylabel("Price")
plt.show()
```



In [38]:

```
fig, ax1 = plt.subplots()

price_by_roomtype_series = [df_selection[df_selection["room_type"]=="Entire home/apt
price_by_roomtype = df_selection.groupby("room_type")[["price"]].mean()

price_by_roomtype.index = price_by_roomtype.index.map(str)
print(price_by_roomtype.index)

acc = df["room_type"].value_counts().sort_index()
print(acc.index)

ax1.set_xlabel('Number of Bathrooms')
ax1.set_ylabel('Count')
ax1.bar(acc.index, acc.values, color = "peachpuff")
ax1.tick_params(axis='y')

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

color = 'lightblue'
ax2.set_ylabel('Average Price')  # we already handled the x-label with ax1
ax2.plot(price_by_roomtype.index.values, price_by_roomtype["price"].values, color =
ax2.tick_params(axis='y')

ax1.grid(False)
ax2.grid(False)
fig.tight_layout()  # otherwise the right y-label is slightly clipped
plt.show()
```

```
Index(['Entire home/apt', 'Hotel room', 'Private room', 'Shared roo
m'], dtype='object', name='room_type')
Index(['Entire home/apt', 'Hotel room', 'Private room', 'Shared roo
m'], dtype='object')
```



Reviews

- The review score indicates higher average prices for better review ratings which matches expectations.
- Notably, several variables in the dataset follow a similar pattern [review_scores_location, review_scores_communication, review_scores_accuracy]

- Therefore the collinearity between review variables also warrants further investigation.

In [39]:

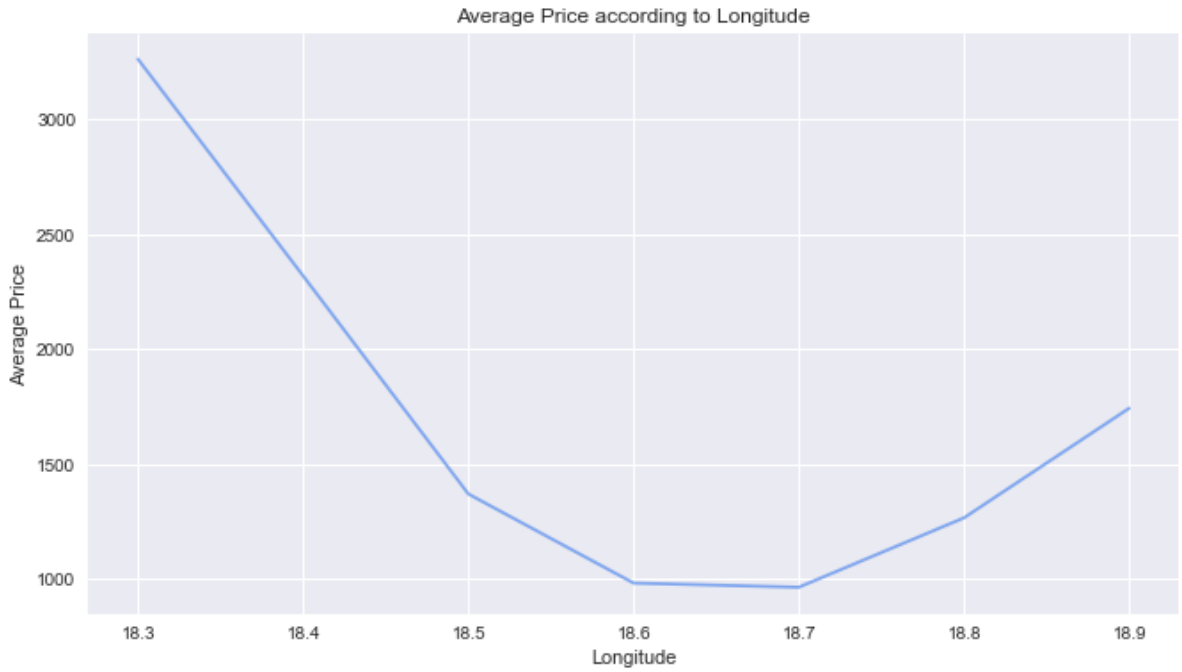
```
price_by_roomtype_series = [df_selection[df_selection["review_scores_value"]==0]["pr


plt.boxplot(price_by_roomtype_series, showfliers=False) # NB fliers/outliers removed
plt.xticks([1, 2, 3, 4, 5, 6], [0, 1, 2, 3, 4, 5])
plt.xlabel("Review Scores Value")
plt.ylabel("Price")
plt.title("Box Plot of Price Distribution per Review Scores Value")
plt.show()
```

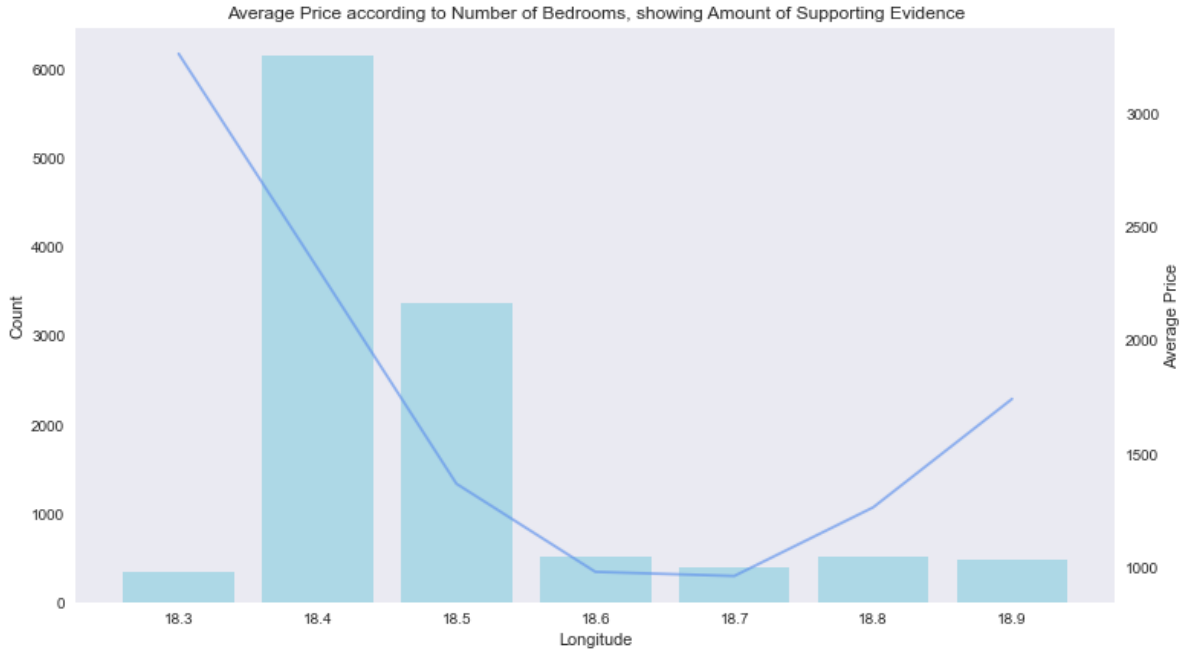
Location

- Following data aggregation, longitude indicates a roughly decreasing relationship with *price*.
- We observe no clear relationship between latitude and the target variable *price*.

In [40]:

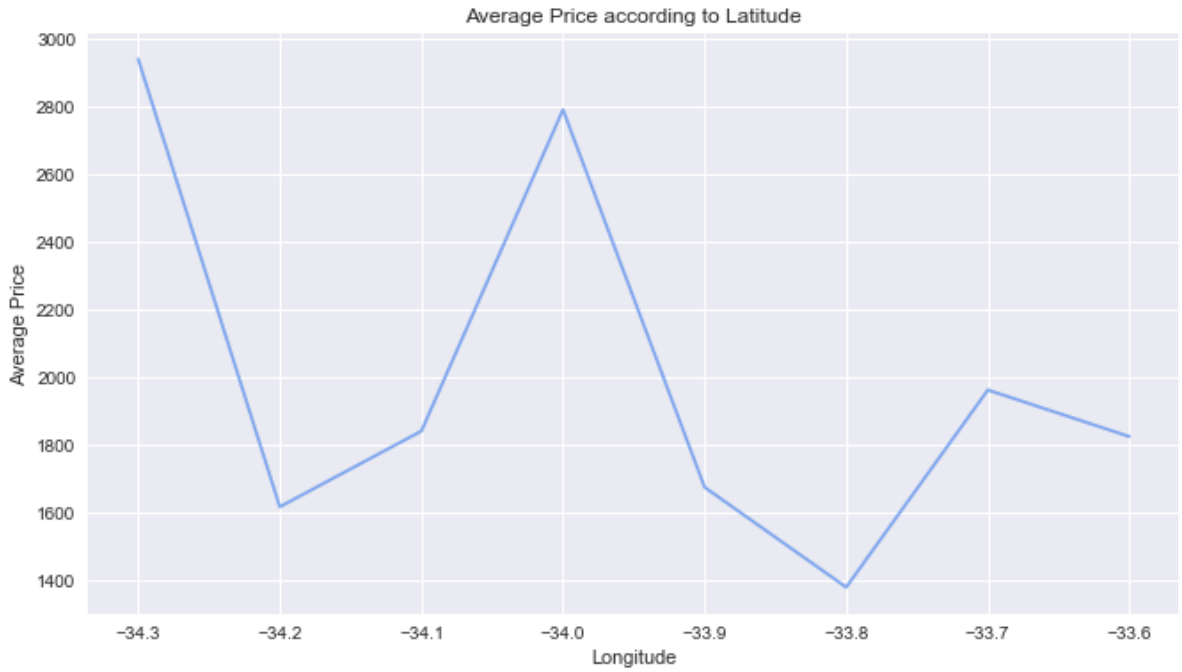
```
price_by_longitude = df_selection.groupby("longitude")[["price"]].mean()

plt.plot(price_by_longitude.index.values, price_by_longitude["price"].values, color
plt.xlabel("Longitude")
plt.ylabel("Average Price")
plt.title("Average Price according to Longitude")
plt.show()

fig, ax1 = plt.subplots()
price_by_longitude.index = price_by_longitude.index.map(str)
acc = df_selection["longitude"].value_counts().sort_index()
acc.index = acc.index.map(str)
ax1.set_xlabel('Longitude')
ax1.set_ylabel('Count')
ax1.bar(acc.index, acc.values, color = "lightblue",  linewidth=0,  width=0.8, align=
ax1.tick_params(axis='y')

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

color = 'lightblue'
ax2.set_ylabel('Average Price')  # we already handled the x-label with ax1
ax2.plot(price_by_longitude.index.values, price_by_longitude["price"].values, color
ax2.tick_params(axis='y')
ax1.grid(False)
ax2.grid(False)
fig.tight_layout()  # otherwise the right y-label is slightly clipped
plt.title("Average Price according to Number of Bedrooms, showing Amount of Supporti
plt.show()
```

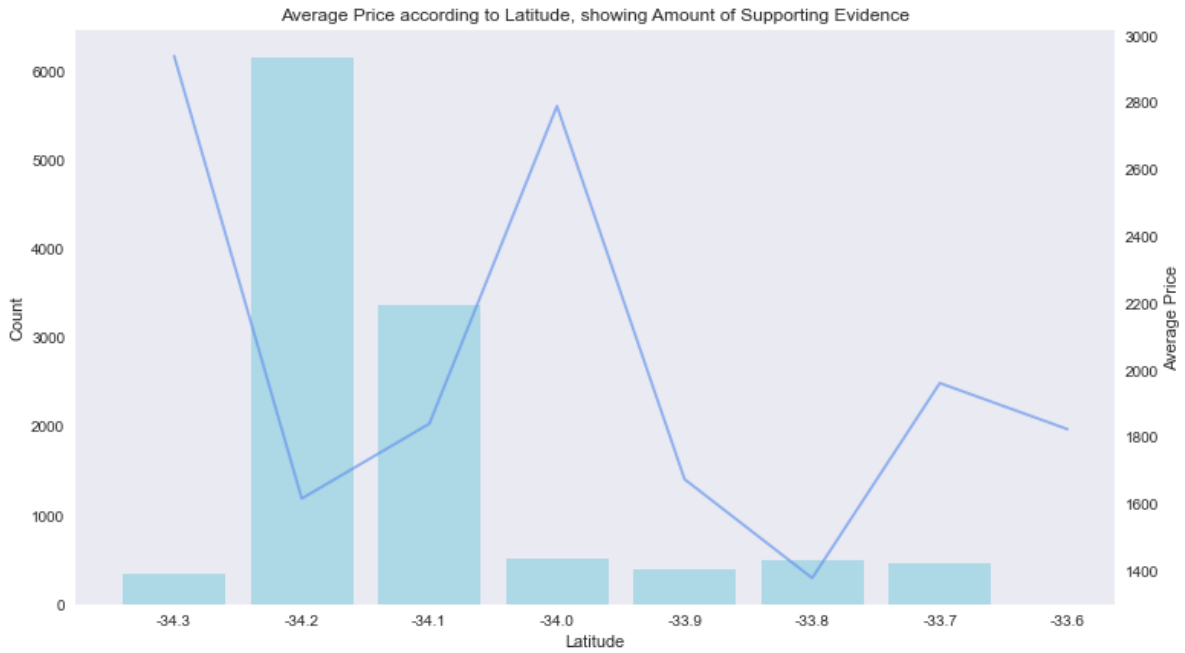



Average Price according to Number of Bedrooms, showing Amount of Supporting Evidence
Count
Average Price
Longitude
0
1000
2000
3000
4000
5000
6000
1000
1500
2000
2500
3000
18.3
18.4
18.5
18.6
18.7
18.8
18.9

In [41]:

```
# latitude

price_by_latitude = df_selection.groupby("latitude")[["price"]].mean()

plt.plot(price_by_latitude.index.values, price_by_latitude["price"].values, color =
plt.xlabel("Longitude")
plt.ylabel("Average Price")
plt.title("Average Price according to Latitude")
plt.show()

fig, ax1 = plt.subplots()
price_by_latitude.index = price_by_latitude.index.map(str)
acc = df_selection["longitude"].value_counts().sort_index()
acc.index = acc.index.map(str)
ax1.set_xlabel('Latitude')
ax1.set_ylabel('Count')
ax1.bar(acc.index, acc.values, color = "lightblue",  linewidth=0,  width=0.8, align=
ax1.tick_params(axis='y')

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

color = 'lightblue'
ax2.set_ylabel('Average Price')  # we already handled the x-label with ax1
ax2.plot(price_by_latitude.index.values, price_by_latitude["price"].values, color =
ax2.tick_params(axis='y')
ax1.grid(False)
ax2.grid(False)
fig.tight_layout()  # otherwise the right y-label is slightly clipped
plt.title("Average Price according to Latitude, showing Amount of Supporting Evidenc
plt.show()
```





Correlation Heatmap

A heatmap showing pairwise correlation is plotted below, indicating which variables have a strong correlation with the target *log_price* variable. In addition, clusters of variables with high collinearity can be identified. Specifically, high collinearity is observed between the following features:

- *accommodates, beds, bedrooms* and *bathrooms*
- *review_scores_accuracy, review_scores_location, review_scores_cleanliness, review_scores_checkin, review_scores_communication* and *review_scores_value*

These observed collinear relationships match intuition which suggests that larger properties will gave a higher number of bedrooms and bathrooms, while a good review score is likely to have a high score in each of the corresponding categories. Variables with high degrees of collinearity will be handled further on in the investigation.

Since all the variables pertaining to the size of the house show the highest correlation with the target variable, we expect size to be the most predictive feature.

In [42]:

```
##----HEATMAP DEFINITION----##
def multi_collinearity_heatmap(df, figsize=(11,9)):

    """
    Creates a heatmap of correlations between features in the df. A figure size can c
    """

    # Set the style of the visualization
    #sns.set(style="white")
    sns.set(style="white", font_scale=1.4)

    # Create a covariance matrix
    corr = df.corr()

    # Generate a mask the size of our covariance matrix
    mask = np.zeros_like(corr, dtype=np.bool)
    mask[np.triu_indices_from(mask)] = True

    # Set up the matplotlib figure
    f, ax = plt.subplots(figsize=figsize)

    # Generate a custom diverging colormap
    cmap = sns.diverging_palette(220, 10, as_cmap=True)

    # Draw the heatmap with the mask and correct aspect ratio
    sns.heatmap(corr, mask=mask, cmap=cmap, center=0, square=True, linewidths=.5, cba
    plt.title("Heatmap of Pairwise Correlations")
    return f
```

In [43]:

```
m = multi_collinearity_heatmap(df_selection, figsize=(20,20))
```

Standardize Variables

Before modelling, the original categorical columns which were transformed through one hot encoding were dropped from the dataset. All variables were then standardized to ensure comparable feature ranges and fulfil the assumption of later model selection methods such as LASSO. The target variable was left unscaled.

In [44]:

```
##---Drop original categorical columns ---##

# drop original columns that were one hot encoded
categorical_columns = ['property_type', 'room_type', 'amenities']
df_selection = df_selection.drop(categorical_columns, axis=1)
```

In [45]:

```
##--- Standardize data ---##
from sklearn.preprocessing import StandardScaler

columns_no_price = df_selection.columns.values
columns_no_price = np.delete(columns_no_price,  np.where(columns_no_price == "price"
columns_no_price = np.delete(columns_no_price,  np.where(columns_no_price == "log_pr
scaler = StandardScaler()
scaled = scaler.fit_transform(df_selection[columns_no_price])
df_selection_scaled = pd.DataFrame(data=scaled, columns=columns_no_price)
df_selection_scaled.head()
```

Out[45]:

	host_acceptance_rate	host_is_superhost	host_listings_count	host_total_listings_count	host_ha
0	0.172114	1.788695	-0.209000	-0.209000	
1	-3.159964	-0.559067	-0.187085	-0.187085	
2	-3.159964	-0.559067	-0.187085	-0.187085	
3	0.399302	-0.559067	-0.187085	-0.187085	
4	0.626489	-0.559067	-0.143255	-0.143255	

Initial Model Fitting

Several initial models were investigated and fitted to forming expectations about the relationships between the variables. The findings are summarised as follows:

- The *accommodates* feature is observed to account for a large proportion of the variance. This poses an interesting problem of how many more variables it is worth including in the fitted model.
- An improvement in R^2 is observed when utilising *log price* rather than *price*. Accordingly, *log price* is utilised for all model selection and inference in the remainder of the investigation.
- It can be noted that despite prior assumptions, location does not appear to have a major contribution in the model in the current format. Most selected features focus on the attributes of the AirBnB listing such as the number of *bedrooms* and *bathrooms* as well as information from previous reviews of the property.

The lack of emphasis on location in the models is likely due to the variables corresponding to location, such as *latitude* and *neighbourhood_cleansed* being too granular. For example, *neighbourhood_cleansed* is separated into 84 different wards. Although simple methods to aggregate these variables have been implemented, alternative methods to aggregate features pertaining to location should be investigated to further explore the effects of location on AirBnB price.

Initial Model with All Variables

In [46]:

```
import numpy as np
from sklearn.linear_model import LinearRegression

# all variables

X = df_selection_scaled
y = df_selection["price"]
reg = LinearRegression().fit(X, y)
print("R^2: {}".format(reg.score(X, y)))
```

```
R^2: 0.5148950015750501
```

In [47]:

```
# all variables

X = df_selection_scaled
y = df_selection["log_price"]
reg = LinearRegression().fit(X, y)
print("R^2: {}".format(reg.score(X, y)))
```

```
R^2: 0.5830040112641308
```

Initial Model with Most Likely Predictors

In [48]:

```
# linear regression model

X = df_selection[["neighbourhood_cleansed", "accommodates", "bathrooms_text", "bedro
y = df_selection["price"]

reg = LinearRegression().fit(X, y)
print("R^2: {}".format(reg.score(X, y)))
print(reg.coef_)
```

```
R^2: 0.48468854724943067
[-1.27330132e+00  2.23419613e+02  9.85996610e+02  8.23109850e+01
 -4.86503992e+02 -2.04262455e+03  1.05084942e+02  9.93547856e+01
 -1.03856700e+03  2.30676635e+01  9.16144552e+02]
```

In [49]:

```
X = df_selection[["neighbourhood_cleansed", "accommodates", "bathrooms_text", "bedro
y = df_selection["log_price"]

reg = LinearRegression().fit(X, y)
print("R^2: {}".format(reg.score(X, y)))
print(reg.coef_)
```

```
R^2: 0.5027186746036902
[ 4.04950186e-04  1.38316467e-01  2.36405243e-01  5.06053312e-02
 -4.15338330e-01 -9.34521660e-01  4.71219333e-02]
```

Initial Model without Collinear Variables

In [50]:

```
X = df_selection_scaled[["latitude", "longitude", "accommodates", "review_scores_rat
y = df_selection["price"]

reg = LinearRegression().fit(X, y)
print("R^2: {}".format(reg.score(X, y)))
print(reg.coef_)
```

```
R^2: 0.4092508387241526
[ -59.8738741  -334.13979351 1458.07635982   84.35845491  -15.38058931
  -90.96196078    8.48455866   85.78637275]
```

In [51]:

```
X = df_selection_scaled[["latitude", "longitude", "accommodates", "review_scores_rat
y = df_selection["log_price"]

reg = LinearRegression().fit(X, y)
print("R^2: {}".format(reg.score(X, y)))
print(reg.coef_)
```

```
R^2: 0.49344231609583067
[-0.0418969  -0.1289868   0.54281855  0.0291032   0.05173289 -0.083083
73
 -0.05397103  0.05728554]
```

Initial Model with Strongest Predictor Only

In [52]:

```
# gets reasonably far with just accommodates
X = df_selection_scaled[["accommodates"]]
y = df_selection["price"]

reg = LinearRegression().fit(X, y)
print("R^2: {}".format(reg.score(X, y)))
print(reg.coef_)
```

```
R^2: 0.3862050290734774
[1467.36433222]
```

In [53]:

```
# gets reasonably far with just accommodates
X = df_selection_scaled[["accommodates"]]
y = df_selection["log_price"]

reg = LinearRegression().fit(X, y)
print("R^2: {}".format(reg.score(X, y)))
print(reg.coef_)
```

```
R^2: 0.44120858156770937
[0.58532122]
```

6. Model Fitting

6.1 The Linear Regression Model

Linear Regression

The linear regression model assumes that the response variable Y is a linear combination of the explanatory variables X as shown below,

$$y_i = \beta_0 + \beta_1 x_{1,i} + \ldots + \beta_m x_{m,i} + \epsilon_i$$

where β_j represents the relationship between the explanatory variable x_j and the response variable Y, and ϵ represents the random error or unexplained variation from factors such as measurement error \cite{coursenotes}.

In the case of the linear regression model, it is assumed that the true relationship between X and Y is linear such that the true data generating distribution is a linear combination of the explanatory variables. The linear regression model assumes that the random errors for each observation are independent and normally distributed with the same variance ($\epsilon_i \sim N(0, \sigma^2)$. It also assumes that the errors are not related to the explanatory variables X and are independent from X (ϵ and X are independent) [3].

6.2 Methodology and Assumptions

When a linear model is used in the context of *inference*, the emphasis is on explaining and understanding the relationships between the explanatory variables X and the response variable y. Therefore in inference the goal is to try to uncover the true data generating process in order to better understand the relationships between the variables.

In order to answer the overarching research questions, the goal of the investigation is to estimate the coefficients β which indicate the relationship between the predictors and the target. In order to estimate the coefficients in a sound manner, a subset of significant features must be identified. Several methods of determining an appropriate variable subset are explored and detailed below.

Forward Stepwise Selection

Forward stepwise regression is a greedy approach which starts with the most minimal model and adds variables one by one according to which variable best improves the model fit. Forward stepwise regression was chosen in favour of backward stepwise regression due to the large number of variables and therefore increased computational cost associated with backward stepwise regression. The number of features selected through forward stepwise regression is a parameter that can be pre-selected.

Through our data exploration process, we have formed expectations regarding which variables are likely to have strong relationships with the target variable *price*. In particular, the number of guests accommodated plays a dominant role. Accordingly, we perform forward stepwise selection with a small selection of 5 variables, and a slightly large selection of 10 variables. A limited number of variables were selected due to prior expectations of only a few variables having a significant relationship with *price*.

LASSO

LASSO provides a method of regularization which shrinks coefficients to zero. As the regularization parameter increases, the LASSO procedure retains only the most significant variables. Therefore it is valuable for model selection. LASSO aims to solve the following optimization problem:

$$min_{\beta_0,\beta} \sum_{i=1}^{N} (y_i - \beta_0 - x_i^T \beta)^2 \ subject\ to \ \sum_{j=1}^{p} |\beta_j| \leq \alpha$$

where α determines the strength of the regularization.

Ridge Regression

Ridge regression does not shrink coefficients to exactly zero and therefore is not used for variable selection in this case. Ridge regression is included in the analysis in order to provide a sense check for the coefficients obtained through the LASSO procedure. Ridge regression aims to solve the following optimization problem:

$$min_{\beta_0,\beta} \sum_{i=1}^{N}(y_i - \beta_0 - x_i^T\beta)^2 \ subject\ to \ \sum_{j=1}^{p}\beta_j^2 \leq \alpha$$

where α determines the strength of the regularization.

6.3 Model Selection

a) Forward Stepwise Regression

Removing Collinear Features

Collinear variables must be removed before performing forward stepwise selection to avoid distortion of the standard errors which will in turn affect the significance tests later on.

In [54]:

```
# take out collinear vars

collinear = ["bathrooms_text"]
columns_no_price = df_selection_scaled.columns.values
columns_no_collinear = np.delete(columns_no_price,  np.where(columns_no_price == "ba
columns_no_collinear = np.delete(columns_no_collinear,  np.where(columns_no_collinea
columns_no_collinear = np.delete(columns_no_collinear,  np.where(columns_no_collinea

# also review scores
columns_no_collinear = np.delete(columns_no_collinear,  np.where(columns_no_collinea
columns_no_collinear = np.delete(columns_no_collinear,  np.where(columns_no_collinea
columns_no_collinear = np.delete(columns_no_collinear,  np.where(columns_no_collinea
columns_no_collinear = np.delete(columns_no_collinear,  np.where(columns_no_collinea
columns_no_collinear = np.delete(columns_no_collinear,  np.where(columns_no_collinea

# number of reviews
columns_no_collinear = np.delete(columns_no_collinear,  np.where(columns_no_collinea
columns_no_collinear = np.delete(columns_no_collinear,  np.where(columns_no_collinea

# minimum maximum nights
columns_no_collinear = np.delete(columns_no_collinear,  np.where(columns_no_collinea
columns_no_collinear = np.delete(columns_no_collinear,  np.where(columns_no_collinea
columns_no_collinear = np.delete(columns_no_collinear,  np.where(columns_no_collinea

df_selection_scaled_no_collinear = df_selection_scaled[columns_no_collinear]
```

(i) Forward Stepwise Regression: 5 Variables

Forward stepwise regression was implemented such that the top 5 variables were selected. A linear regression model was then fitted using the selected variables.

In [55]:

```
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.neighbors import KNeighborsClassifier

#X = df_selection_scaled
X = df_selection_scaled_no_collinear
y = df_selection["log_price"]

# forward stepwise is much less computationally expensive given large number of tota
sfs = SequentialFeatureSelector(reg, direction="forward", scoring = "neg_mean_square
sfs.fit(X, y)

print("5 Features chosen through Forward Stepwise Regression\n")
fsr_5_chosen_features = columns_no_collinear[sfs.get_support()]
print(fsr_5_chosen_features)


# fit a linear model with the chosen features

X = df_selection_scaled_no_collinear[fsr_5_chosen_features]
y = df_selection["log_price"]
reg = LinearRegression().fit(X, y)
print("Fitted Linear Regression Model\n")
print("R^2 Score: {}".format(reg.score(X, y)))
print("Coefficient Values: {}".format(reg.coef_))
```

```
5 Features chosen through Forward Stepwise Regression

['longitude' 'accommodates' 'number_of_reviews' 'private' 'shared']
Fitted Linear Regression Model

R^2 Score: 0.5015756417949434
Coefficient Values: [-0.12968688  0.52996975 -0.10629131 -0.12851629 -
0.09922016]
```

(ii) Forward Stepwise Regression: 10 Variables

In addition, forward stepwise regression was implemented such that the top 10 variables were selected.

In [56]:

```
##--- Forward Stepwise Regression - 15 Features ---##

X = df_selection_scaled_no_collinear
y = df_selection["log_price"]

# forward stepwise is much less computationally expensive given large number of tota
sfs = SequentialFeatureSelector(reg, direction="forward", scoring = "neg_mean_square
sfs.fit(X, y)

fsr_10_chosen_features = columns_no_collinear[sfs.get_support()]
print("10 Features chosen through Forward Stepwise Regression\n")
print(fsr_10_chosen_features)
```

```
10 Features chosen through Forward Stepwise Regression

['host_identity_verified' 'latitude' 'longitude' 'accommodates'
 'availability_30' 'number_of_reviews' 'review_scores_rating'
 'calculated_host_listings_count_shared_rooms' 'private' 'shared']
```

In [57]:

```
# fit a linear model with the chosen features

X = df_selection_scaled_no_collinear[fsr_10_chosen_features]
y = df_selection["log_price"]
reg = LinearRegression().fit(X, y)
print("R^2 Score: {}".format(reg.score(X, y)))
print(reg.coef_)
```

```
R^2 Score: 0.5193000173124416
[ 0.04709111 -0.0463638  -0.14276521  0.52052565  0.09308534 -0.105248
07
  0.03914258 -0.04293828 -0.13984809 -0.07999034]
```

b) LASSO with CV

LASSO was then performed using cross validation to determine the value of the regularisation parameter α.

In [58]:

```
##--- LASSO with optimal regularization strength chosen using CV ---##

from sklearn import linear_model

X = df_selection_scaled
y = df_selection["log_price"] # also log price

lasso = linear_model.LassoCV(max_iter=3000, tol=0.01).fit(X, y)
print("R^2 Score: {}".format(lasso.score(X, y)))
print("Alpha selected through CV: {}".format(lasso.alpha_))
```

```
R^2 Score: 0.5825522055796137
Alpha selected through CV: 0.000721611361923685
```

In [59]:

```
lasso_cv_var_values = lasso.coef_[np.where(lasso.coef_ !=0)]
lasso_cv_var_names = columns_no_price[np.where(lasso.coef_ !=0)]
print(lasso_cv_var_names)
print("Number of chosen variables: {}".format(len(lasso_cv_var_names)))
```

```
['host_acceptance_rate' 'host_is_superhost' 'host_listings_count'
 'host_total_listings_count' 'host_has_profile_pic'
 'host_identity_verified' 'neighbourhood_cleansed' 'latitude' 'longitu
de'
 'accommodates' 'bathrooms_text' 'bedrooms' 'beds' 'maximum_nights'
 'maximum_minimum_nights' 'minimum_maximum_nights'
 'maximum_maximum_nights' 'minimum_nights_avg_ntm'
 'maximum_nights_avg_ntm' 'has_availability' 'availability_30'
 'availability_60' 'availability_90' 'availability_365'
 'number_of_reviews' 'number_of_reviews_ltm' 'number_of_reviews_l30d'
 'review_scores_rating' 'review_scores_accuracy'
 'review_scores_cleanliness' 'review_scores_checkin'
 'review_scores_communication' 'review_scores_location'
 'review_scores_value' 'instant_bookable' 'calculated_host_listings_co
unt'
 'calculated_host_listings_count_entire_homes'
 'calculated_host_listings_count_private_rooms'
 'calculated_host_listings_count_shared_rooms' 'reviews_per_month'
 'entire' 'hotel' 'private' 'shared']
Number of chosen variables: 44
```

In [60]:

```
# fit a linear model with all values chosen from LASSO CV

X = df_selection_scaled[lasso_cv_var_names]
y = df_selection["log_price"] # also log price?
reg = LinearRegression().fit(X, y)
print("R^2 Score: {}".format(reg.score(X, y)))
print(reg.coef_)
```

```
R^2 Score: 0.5829139503719206
[ 2.90145252e-02 -9.47492788e-03 -5.60463000e+11  5.60463000e+11
 -2.21903405e-03  4.00668435e-02  8.49219277e-03 -4.21462978e-02
 -1.17597294e-01  3.29740906e-01  2.72492746e-01  2.30418052e-02
 -5.85802990e-02  5.43893097e-02  1.40190347e-01  3.63428107e-02
  6.64815078e-02 -1.74550184e-01 -1.45868599e-01 -2.14890554e-02
  1.48599596e-01 -1.21706810e-01  4.35766670e-02  3.62929827e-02
 -5.50332503e-02 -2.66537617e-02 -3.24582757e-02  1.48775139e-02
  3.12530335e-02  6.04488991e-02 -8.98592604e-03  2.12053942e-02
  7.87962317e-02 -1.03144405e-01  2.41925000e-02  5.11967784e-02
 -3.68642952e-02 -3.05695378e-02 -4.02865472e-02 -1.37658680e-02
  2.98879596e+11  8.16258812e+10  2.90676676e+11  5.35576325e+10]
```

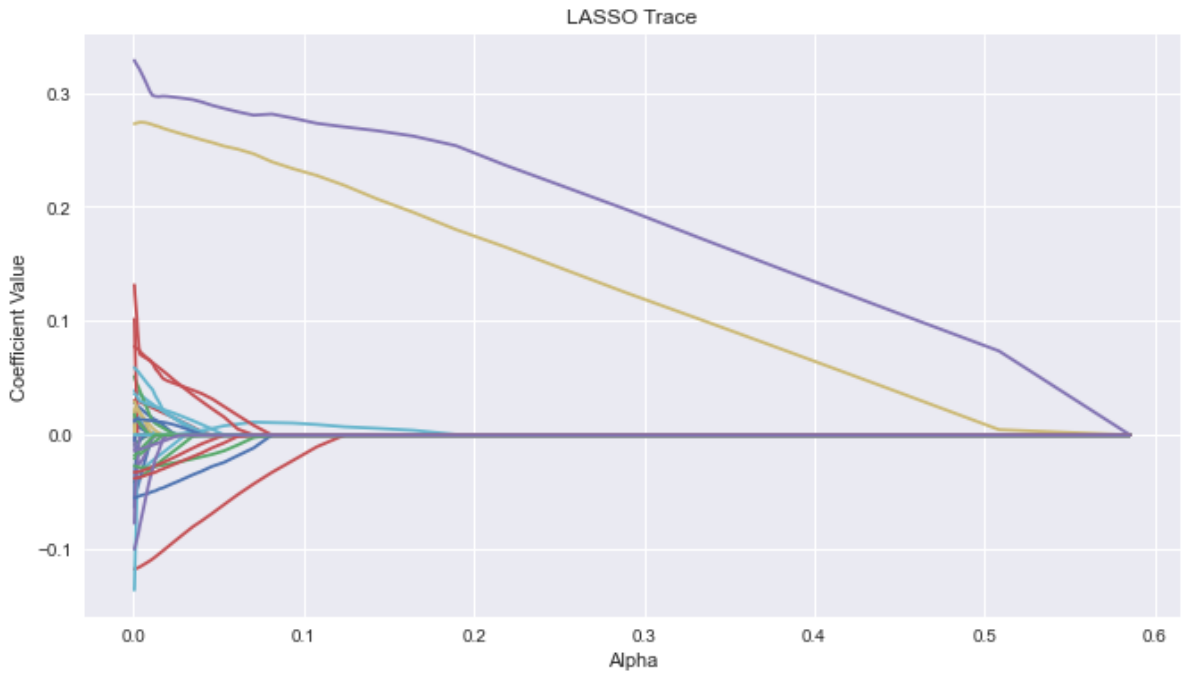
LASSO Trace

From the LASSO trace we observe two dominant variables that retain large positive coefficient values as regularization strength increases, shown by the purple and yellow lines respectively. These two lines correspond to the variables *accommodates* and *bathrooms_text* which both pertain to the size of the property. These variables remain stable and signficant as the regularization parameter α increases. However, these variables are highly correlated with a score of 0.80.

Working backwards we observe that the red line indicates another stable variable with a significant negative coefficient. This corresponds to the *longitude* variable. Notably, out of all potential 51 variables, only a small number remain significant as regularization strength increases. This informs the next model where we perform LASSO with strong regularization to attain a smaller feature set.

In [61]:

```
from sklearn.linear_model import lasso_path

plt.style.use('seaborn')
plt.rcParams["figure.figsize"] = (11,6)

# chosen alphas
chosen_alphas = [0.1, 1, 10]

# Use lasso_path to compute a coefficient path
alphas, coefs_lasso, _  = lasso_path(X, y, n_alphas = 50, max_iter=3000, tol=0.01)

for i in range(40):

    plt.plot(alphas, coefs_lasso[i])

plt.xlabel("Alpha")
plt.ylabel("Coefficient Value")
plt.title("LASSO Trace")
plt.show()
```



In [62]:

```
# variables with largest coefficients are highly correlated

lasso_cv_var_names_big = columns_no_price[np.where(np.abs(lasso.coef_) > 0.2)]
print(lasso_cv_var_names_big)

print(df_selection['accommodates'].corr(df_selection['bathrooms_text']))
```

```
['accommodates' 'bathrooms_text']
0.7971033935057292
```

c) LASSO with Strong Regularization

LASSO with strong regularization ($\alpha = 0.1$) was then performed to obtain a small set of features.

After fitting a linear model with the features chosen through strong regularization, we observe a decrease in R^2 score. This is expected since as we increase the regularization parameter λ, the squared bias will steadily increase as more regularization is being applied and therefore the coefficient estimates are more restricted and are further from the least squares estimates which have the lowest bias. This occurs because as the model becomes less flexible, the bias increases and the variance decreases. Although the R^2 score decreases, insights from the exploratory process and the LASSO trace indicate that a smaller set of features are significant to the research question. This will become even more crucial when selecting the final model.

In [63]:

```
from sklearn import linear_model

X = df_selection_scaled
y = df_selection["log_price"]

lasso = linear_model.Lasso(alpha=0.1, max_iter=3000, tol=0.01).fit(X, y)
print("R^2 Score: {}".format(lasso.score(X, y)))
print(lasso.coef_)
```

```
R^2 Score: 0.4844489520196107
[ 0.         -0.          0.          0.          0.          0.
  0.         -0.         -0.01689115  0.27764955  0.23340652  0.005939
33
  0.         -0.          0.         -0.         -0.         -0.
 -0.         -0.         -0.         -0.          0.          0.
  0.          0.         -0.         -0.         -0.          0.
  0.          0.          0.          0.          0.          0.
 -0.          0.          0.         -0.         -0.         -0.
  0.02873057  0.         -0.02600208 -0.        ]
```

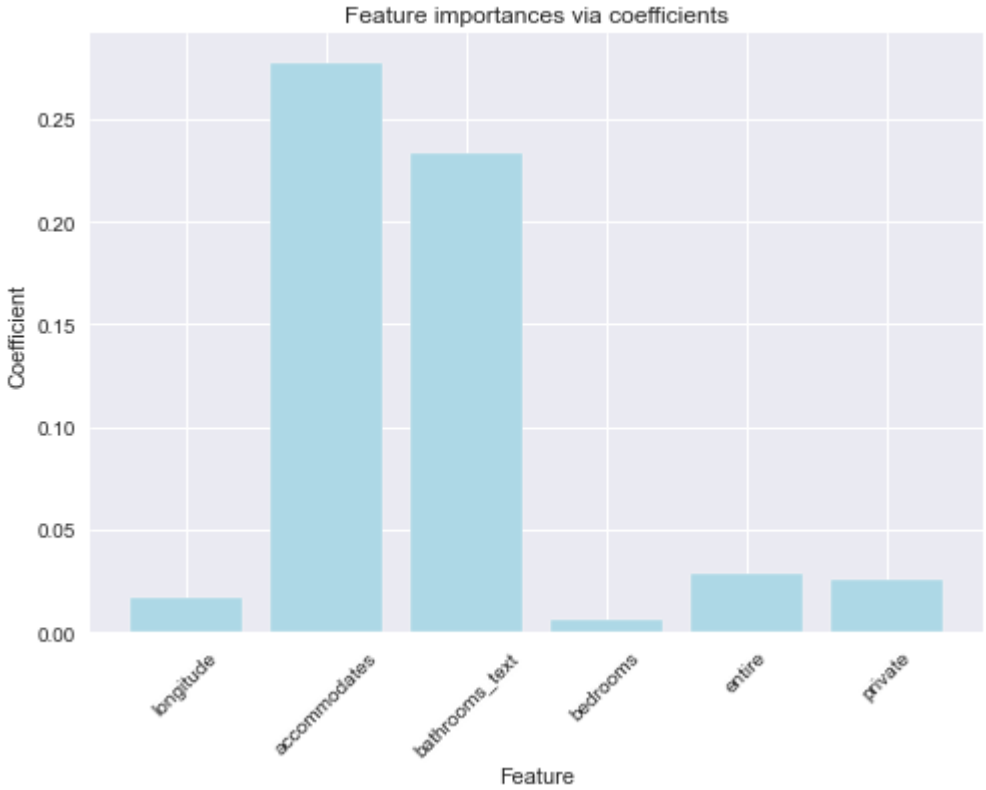
In [64]:

```
lasso_var_values = lasso.coef_[np.where(lasso.coef_ !=0)]
lasso_var_names = columns_no_price[np.where(lasso.coef_ !=0)]
print(lasso_var_names)
print(lasso_var_values)
print("Number of chosen variables: {}".format(len(lasso_var_names)))
```

```
['longitude' 'accommodates' 'bathrooms_text' 'bedrooms' 'entire' 'priv
ate']
[-0.01689115  0.27764955  0.23340652  0.00593933  0.02873057 -0.026002
08]
Number of chosen variables: 6
```

In [65]:

```
sns.set(rc = {'figure.figsize':(11,6)})
plt.style.use('seaborn')
importance = np.abs(lasso_var_values)
plt.bar(height=importance, x=lasso_var_names, color = "lightblue")
plt.title("Feature importances via coefficients")
plt.xticks(rotation=45)
plt.xlabel("Feature")
plt.ylabel("Coefficient")
plt.show()
```



In [66]:

```
# fit a linear model with the chosen features

X = df_selection_scaled[lasso_var_names]
y = df_selection["log_price"]
reg = LinearRegression().fit(X, y)
print("R^2 Score: {}".format(reg.score(X, y)))
print(reg.coef_)
```

```
R^2 Score: 0.5146617512452065
[-0.10191632  0.29869545  0.27827314  0.02400164  0.04109928 -0.080605
74]
```

d) Ridge Regression

Lastly, we utilise Ridge to compare the coefficients obtained through Ridge Regression to those obtained through LASSO.

In [67]:

```
from sklearn.linear_model import Ridge

X = df_selection_scaled[lasso_var_names]
y = df_selection["log_price"]

ridge = Ridge(alpha=0.01).fit(X, y)
print(ridge.score(X, y))
print(ridge.coef_)
print(ridge.intercept_)
```

```
0.514661751244785
[-0.10191625  0.29869457  0.27827266  0.0240028   0.04109943 -0.080605
56]
7.104816126188863
```

Comparing LASSO and Ridge Coefficients

In [68]:

```
comp_df = pd.DataFrame({"Feature": lasso_var_names, "LASSO": np.around(reg.coef_, 3)
comp_df.head(11)
```

Out[68]:

	Feature	LASSO	Ridge
0	longitude	-0.102	-0.102
1	accommodates	0.299	0.299
2	bathrooms_text	0.278	0.278
3	bedrooms	0.024	0.024
4	entire	0.041	0.041
5	private	-0.081	-0.081

The coefficients obtained from Ridge and LASSO regression are observed to be very similar.

6.5 Selection of Final Model

Several metrics were considered to inform the model selection process. The exactness of the error estimate given by each metric is less important for model selection where the order between models is more crucial. The appropriateness of each metric for the given context is discussed below.

a) Akaike Information Criterion (AIC)

The AIC is appropriate in a model selection problem where it is assumed that there are multiple incompletely specified or infinite parameter models. Therefore it is assumed that the process being modelled is infinitely complex and can never be modelled in its entirety, and instead a useful approximation is found. This provides an applicable metric in our scenario since the distribution of AirBnB prices is particularly complex and dependent on many factors. The models are incompletely specified as the distribution of AirBnB price may also depend on variables that are not included in the dataset. Therefore the AIC is a suitable metric in this context.

b) Bayesian Information Criterion (BIC)

In contrast to AIC, the Bayesian Information Criterion (BIC) assumes that at least one model is correct. The BIC is appropriate in a model selection problem where it is assumed that there is a small number of completely specified models and that one of the models is true. Therefore the data generating process is assumed to be more simple, and as the amount of data increases, gets closer to the single best model. Therefore the BIC is less appropriate in our context since the assumption that one of the model is true cannot be fulfilled.

c) Cross Validation

In contrast to AIC and BIC, cross-validation provides an estimate of the out-of-sample error. Cross-validation is appropriate in situations where the dataset size is not too limited and can therefore be split in training and validation sets that are large enough to provide good estimates. This is significant since the properties of the cross validation estimator depend on the amount of data in the training and validation sets. If the dataset size is too small such that the quality of the estimator and error metric will be compromised then AIC and BIC are more appropriate as they utilise the full dataset. However since the number of observations in the dataset is

large, cross validation provides a valid metric to compare the models in this case. In addition, while cross-validation can also be more computationally expensive since the computational expense increases with the number of splits B, this does not pose a problem in this scenario as calculations remain computationally feasible.

K-Fold Cross Validation

K-fold cross validation is performed by splitting the dataset into k partitions where k is specified beforehand. The k partitions, known as folds, are of roughly equal size. The model is then fitted on $k-1$ folds with the k^{th} fold being reserved for validation, allowing for evaluation on the k^{th} fold. The fitting and validation procedure is then repeated, treating each of the k folds as the validation set in turn. This yields k evaluation measures which can be averaged to give a final evaluation score.

Justification

K-fold cross validation with 10 folds was chosen in favour of Leave One Out CV (LOOCV) or the simpler validation set approach for the following reasons. In the validation set approach, the dataset is only split into one training set and one validation set. The model is fitted on the training data and evaluated on the validation or hold-out data. Therefore the evaluation metric represents only the performance on a singular validation set. This could be problematic if the validation set is not representative of the entire dataset and can lead to high variability of the test error estimate depending on which observations have been included in the validation set [2].

Leave One Out CV (LOOCV) is a special case of k-fold CV with $k = n$. This requires the model to be fitted n times. Since k-fold CV is usually implemented with a much smaller number of folds, such as $k = 5$ or $k = 10$, it has a much lower computational cost and is therefore beneficial from a computational efficiency standpoint. In addition the the potential computational advantages, a further benefit can arise with respect to the bias-variance trade-off, resulting in $k - fold$ validation producing more accurate estimates of the test error than the estimates produced by LOOCV. While LOOCV produces approximately unbiased estimates of the test error, LOOCV has higher variance since the n fitted models are trained on very similar datasets and therefore produce highly correlated outputs.\cite{hastie} [1].

K-fold cross validation with $k = 10$ therefore provides a good balance since it results in estimates of the test error rate that do not have very high bias or very high variance.

(i) Forward Stepwise Regression with 5 Features

In [69]:

```
import statsmodels.api as sm

# init results lists
aics = []
bics = []
cv_scores_mean = []
cv_scores_dev = []

# 5 chosen features from stepwise regression
X = df_selection_scaled[fsr_5_chosen_features]
y = list(df_selection["log_price"])
X = sm.add_constant(X)

#fit regression model
model = sm.OLS(y, X).fit()
aics.append(model.aic)
bics.append(model.bic)
print("AIC: {}".format(model.aic))
print("BIC: {}".format(model.bic))
```

```
AIC: 22203.310884066166
BIC: 22247.5405351836
```

In [70]:

```
from sklearn.model_selection import cross_val_score

X = df_selection_scaled[fsr_5_chosen_features]
y = df_selection["log_price"]
reg = LinearRegression().fit(X, y)
scores = cross_val_score(reg, X, y, cv=10)
print(scores)
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.s
cv_scores_mean.append(scores.mean())
cv_scores_dev.append(scores.std())
```

```
[0.55185324 0.43357471 0.4829069  0.49542588 0.54091641 0.52793146
 0.4907588  0.52112408 0.47518898 0.48171996]
0.50 accuracy with a standard deviation of 0.03
```

(ii) Forward Stepwise Regression with 10 Features

In [71]:

```
##--- AIC and BIC ---##

# 10 chosen features from stepwise regression
X = df_selection_scaled[fsr_10_chosen_features]
y = list(df_selection["log_price"])
X = sm.add_constant(X)

#fit regression model
model = sm.OLS(y, X).fit()
aics.append(model.aic)
bics.append(model.bic)
print("AIC: {}".format(model.aic))
print("BIC: {}".format(model.bic))
```

```
AIC: 21787.861006748597
BIC: 21868.948700463894
```

In [72]:

```
##--- 10-Fold CV ---##

X = df_selection_scaled[fsr_10_chosen_features]
y = df_selection["log_price"]
reg = LinearRegression().fit(X, y)
scores = cross_val_score(reg, X, y, cv=10)
print(scores)
print("%0.3f accuracy with a standard deviation of %0.3f" % (scores.mean(), scores.s
cv_scores_mean.append(scores.mean())
cv_scores_dev.append(scores.std())
```

```
[0.56366202 0.45839552 0.49881785 0.51232894 0.5577624  0.54968074
 0.50783316 0.54554721 0.48561494 0.49510321]
0.517 accuracy with a standard deviation of 0.033
```

(iii) Lasso with α = 0.1

In [73]:

```
##--- AIC and BIC ---##

# LASSO
X = df_selection_scaled[lasso_var_names]
y = list(df_selection["log_price"])
X = sm.add_constant(X)

#fit regression model
model = sm.OLS(y, X).fit()
aics.append(model.aic)
bics.append(model.bic)
print("AIC: {}".format(model.aic))
print("BIC: {}".format(model.bic))
```

```
AIC: 21892.6930734529
BIC: 21944.294333089907
```

In [74]:

```
##--- 10-Fold CV ---##

X = df_selection_scaled[lasso_var_names]
y = df_selection["log_price"]
reg = LinearRegression().fit(X, y)
scores = cross_val_score(reg, X, y, cv=10)
print(scores)
print("%0.3f accuracy with a standard deviation of %0.3f" % (scores.mean(), scores.s
cv_scores_mean.append(scores.mean())
cv_scores_dev.append(scores.std())
```

```
[0.56226563 0.44762756 0.52207948 0.48949406 0.56002402 0.5346188
 0.49990269 0.52476695 0.48247754 0.51042325]
0.513 accuracy with a standard deviation of 0.034
```

(iv) Lasso with α chosen using CV

In [75]:

```
##--- AIC and BIC ---##

# LASSO
X = df_selection_scaled[lasso_cv_var_names]
y = list(df_selection["log_price"])
X = sm.add_constant(X)

#fit regression model
model = sm.OLS(y, X).fit()
aics.append(model.aic)
bics.append(model.bic)
print("AIC: {}".format(model.aic))
print("BIC: {}".format(model.bic))
```

```
AIC: 20183.930763562996
BIC: 20500.909929904607
```

In [76]:

```
##--- 10-Fold CV ---##

X = df_selection_scaled[lasso_cv_var_names]
y = df_selection["log_price"]
reg = LinearRegression().fit(X, y)
scores = cross_val_score(reg, X, y, cv=10)
print(scores)
print("%0.3f accuracy with a standard deviation of %0.3f" % (scores.mean(), scores.s
cv_scores_mean.append(scores.mean())
cv_scores_dev.append(scores.std())
```

```
[0.6106927  0.52292349 0.57078697 0.57799308 0.60938747 0.59882199
 0.57576611 0.61550734 0.54228057 0.5632994 ]
0.579 accuracy with a standard deviation of 0.029
```

Model Selection Results

In [77]:

```
##--- Tabulate model selection results ---##

method_index = ["Forward Stepwise Regression with 5 Features", "Forward Stepwise Reg
results_df = pd.DataFrame({"Method": method_index, "AIC": aics, "BIC": bics, "10-Fol
results_df.head(11)
```

Out[77]:

	Method	AIC	BIC	10-Fold CV Mean	10-Fold CV Std. Dev.
0	Forward Stepwise Regression with 5 Features	22203.310884	22247.540535	0.500140	0.033689
1	Forward Stepwise Regression with 10 Features	21787.861007	21868.948700	0.517475	0.033249
2	LASSO	21892.693073	21944.294333	0.513368	0.033558
3	LASSO + CV	20183.930764	20500.909930	0.578746	0.029105

Final Model Selection and Justification

The final model was chosen as the feature set obtained using forward stepwise regression with 10 features. Although the feature set obtained using LASSO with the α paramater chosen through cross validation obtained the best AIC and cross validation scores, the improved scores were mainly due to the much higher number of variables in the model. While the LASSO + CV approach selected 44 variables, the other models compared contained 5, 10 and 6 variables respectively. While a larger number of variables may inflate the R^2 score, insights gained through data exploration and the model selection procedure suggest that most of the variables are not significant and will therefore not generalise well.

Justification

Therefore the feature set obtained using forward stepwise regression with 10 features was selected as it performed second best for all metrics (AIC, BIC and 10-fold CV). The inclusion of ten significant features is also in line with expectations since a small subset of features is anticipated due to:

- The dominance of the *accommodates* variable
- High number of collinear variables present in the dataset
- Interpretation of the LASSO trace

Choosing a model selected through forward stepwise regression also helps to minimize the presence of collinearity in the feature set which could distort the parameter estimates. This is significant since our investigation seeks to understand the relationships between the predictors and the target rather than obtaining the best prediction performance.

Hence the final model was selected taking into consideration the following:

- The task being performed: Inference rather than prediction - correctness of the parameter estimates is more important than predictive performance.
- Estimates of the error: The performance according to the AIC, BIC and 10-fold CV metrics informed the final decision.

- Intuition gained from data exploration: Many variables lacked a substantive relationship with *price* or provided similar information.

Details of the Final Model

The set of 10 selected features includes:

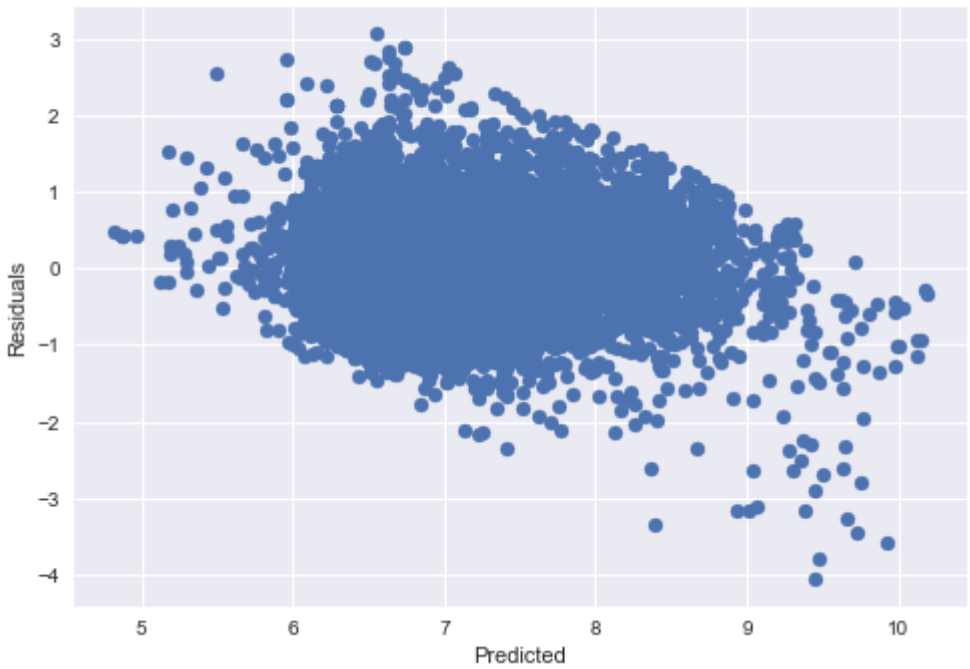
- ['host_identity_verified' 'latitude' 'longitude' 'accommodates' 'availability_30' 'number_of_reviews' 'review_scores_rating' 'calculated_host_listings_count_shared_rooms' 'private' 'shared']

6.6 Residual Analysis

The residuals were then plotted to determine whether any patterns could be identified.

In [78]:

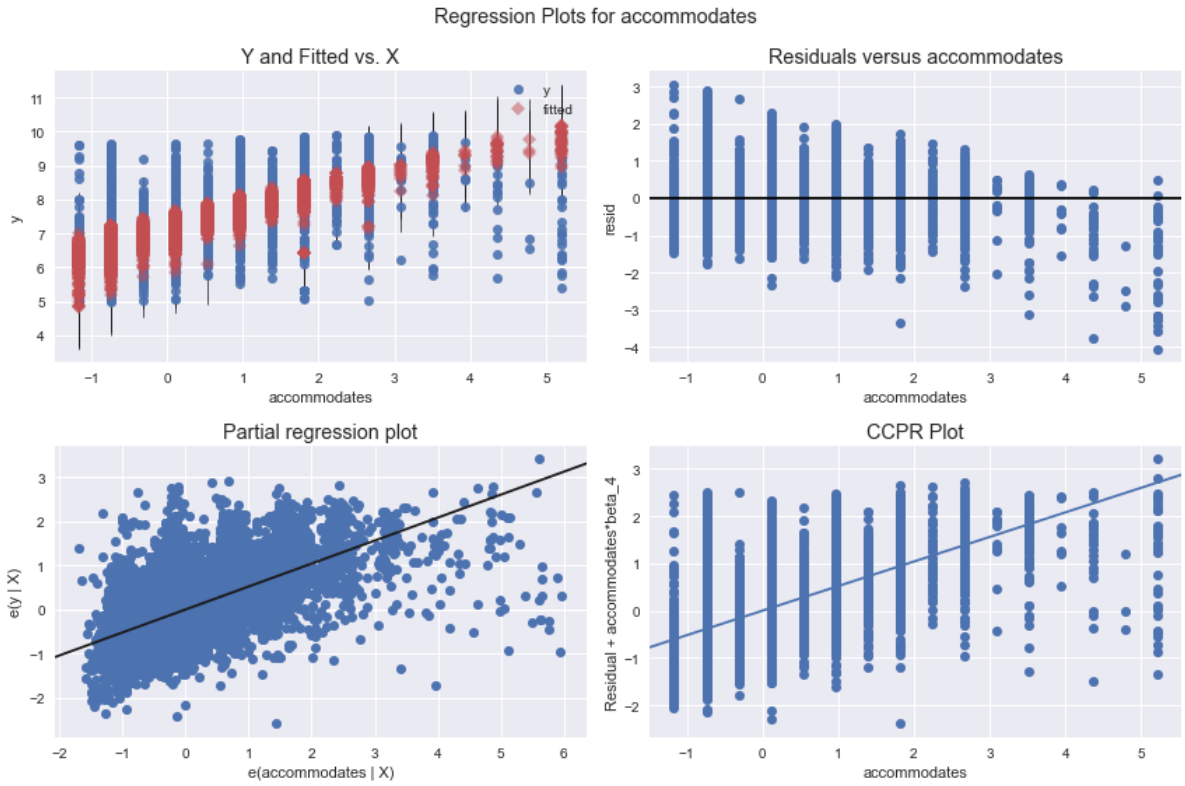
```
# Selected Model

sns.set(rc = {'figure.figsize':(11,6)})
plt.style.use('seaborn')
X = df_selection_scaled[fsr_10_chosen_features]
y = df_selection["log_price"] # also log price?
reg = LinearRegression().fit(X, y)
y_pred = reg.predict(X)
y_true = df_selection["log_price"].values
residuals = y_true-y_pred
plt.scatter(y_pred, residuals, )
plt.ylabel("Residuals")
plt.xlabel("Predicted")
plt.show()
```



In [79]:

```
X = df_selection_scaled[fsr_10_chosen_features]
y = list(df_selection["log_price"])
X = sm.add_constant(X)

#fit regression model
model = sm.OLS(y, X).fit()

#define figure size
fig = plt.figure(figsize=(12,8))

# change font size

#produce regression plots
fig = sm.graphics.plot_regress_exog(model, 'accommodates', fig=fig)
```

```
eval_env: 1
```



7. Hypothesis Testing

7.1 Testing Framework

We perform a t-test to test the significance of each regression coefficient as follows,

$$H_0 : \beta_i = 0$$
$$H_a : \beta_i \neq 0$$

where the null hypothesis states that the given variable has no effect and should not be included in the model, and the alternate hypothesis states that the variable is significant. Since a regression coefficient can be positive or negative, a two-sided test is performed.

7.2 False Positives

During hypothesis testing procedures it is important to control the possible amount of Type I errors or false positives. A Type I error occurs when the null hypothesis H_0 is incorrectly rejected, and therefore a false positive is obtained. The Type I error rate is defined as the probability of making a Type I error given that the null hypothesis H_0 is true. Type I errors are usually viewed as more serious than Type II errors since they put forward an incorrect scientific finding. Therefore the Type I error rate is controlled to be a small number by specifying the significance level of a hypothesis test α. By only rejecting the null hypothesis H_0 when the p-value is below α, we can ensure that the Type I error rate will be less than or equal to α. However, due to the effects of multiple testing, additional considerations are taken into account when managing the rate of false positives.

7.3 Correcting for the Effects of Multiple Testing

Since we are testing multiple variables for significance, complexities associated with multiple testing apply to our problem. Therefore additional measures are necessary to control the amount of false positives. To accomplish this we utilise *Bonferroni's Procedure*.

The Family-Wise Error Rate (FWER) is the probability of at least one false positive.

$$FWER = P(reject\ any\ true\ null\ hypothesis\ H_0)$$

Bonferroni's Procedure is used to set the significance of individual hypothesis tests in order to limit the Family-Wise Error Rate (FWER) to α. Therefore if the desired probability of at least one type I error is α, we set the significance level for each individual test as follows, (check)

$$\alpha_{ind} = \frac{\alpha}{n}$$

which will then give,

$$Pr(At\ least\ one\ Type\ I\ error\ in\ n\ tests) = 1 - (1 - \frac{\alpha}{n})^n \approx n\frac{\alpha}{n} = \alpha$$

Justification

Since the number of variables tested is reasonably small, a single false positive in the final model will be highly problematic. Therefore controlling the Family Wise Error Rate (FWER) using Bonferroni's procedure is appropriate in our case since it allows us to control the probability of at least one false positive. Although Bonferroni's procedure is known to reduce the power of the test, this is often only problematic for a larger number of variables.

Standardization of the Inference Set

The inference set was first standardized to mirror the preprocessing of the selection set.

In [80]:

```
# remove categorical columns that have been one hot encoded
categorical_columns = ['property_type', 'room_type', 'amenities']
df_inference = df_inference.drop(categorical_columns, axis=1)

# transform data
# don't standardize the target
columns_no_price = df_inference.columns.values
columns_no_price = np.delete(columns_no_price,  np.where(columns_no_price == "price'
columns_no_price = np.delete(columns_no_price,  np.where(columns_no_price == "log_pr
scaler = StandardScaler()
scaled_test = scaler.fit_transform(df_inference[columns_no_price])
df_inference_scaled = pd.DataFrame(data=scaled_test, columns=columns_no_price)
df_inference_scaled.head()
```

Out[80]:

	host_acceptance_rate	host_is_superhost	host_listings_count	host_total_listings_count	host_ha
0	-2.165010	-0.559614	-0.327585	-0.327585	
1	-0.285474	1.786945	-0.327585	-0.327585	
2	-0.677044	-0.559614	-0.375005	-0.375005	
3	0.380194	-0.559614	-0.375005	-0.375005	
4	-3.300562	1.786945	-0.327585	-0.327585	

Calculating the P-Values

As discussed above, we define the t-test for a regression coefficient as follows,

$$H_0 : \beta_1 = 0$$
$$H_a : \beta_1 \neq 0$$

Since this hypothesis test is performed for each variable, we can obtain the associated p-values. The p-value represents the probability of observing a value at least as extreme as the one observed from the data if the null hypothesis H_0 is true. The smaller the p-value, the stronger the evidence against the null hypothesis.

In [81]:

```
# chosen model from AIC/CV

X = df_inference_scaled[fsr_10_chosen_features]
y = df_inference["log_price"]

reg = LinearRegression().fit(X, y)
print(reg.score(X, y))
print(reg.intercept_)
```

```
0.5376176530105613
7.09735334305396
```

In [82]:

```
# hypothesis test for each variable

X = df_inference_scaled[fsr_10_chosen_features]
y = list(df_inference["log_price"])
X = sm.add_constant(X)

#fit regression model on the inference data
model = sm.OLS(y, X).fit()

print(model.pvalues)
p_values = model.pvalues.values
```

```
const                                          0.000000e+00
host_identity_verified                         3.316068e-08
latitude                                       7.292099e-08
longitude                                      8.221225e-48
accommodates                                   0.000000e+00
availability_30                                4.704772e-23
number_of_reviews                              7.881090e-30
review_scores_rating                           4.626527e-07
calculated_host_listings_count_shared_rooms    1.809874e-04
private                                        1.300926e-30
shared                                         6.372807e-16
dtype: float64
```

Bonferroni's Correction

As presented in the methodology above, if the desired probability of at least one type I error is α, we set the significance level for each individual test as follows,

$$\alpha_{ind} = \frac{\alpha}{n}$$

This can be performed equivalently by adjusting the p-values found in the original testing procedure. From these adjusted p-values, it can be ascertained which of the variables are significant and should be retained in the model.

In [84]:

```
# adjust them using Bonferroni

from statsmodels.stats.multitest import multipletests
p_vals = p_values

# Create a list of the adjusted p-values
# alpha = FWER, family-wise error rate, e.g. 0.1

p_adjusted = multipletests(p_vals, alpha= 0.1, method='bonferroni')

# Print the resulting conclusions
print("Conclusions")
print(p_adjusted[0])

# Print the adjusted p-values themselves
print("\nAdjusted P Values")
print(p_adjusted[1])
```

```
Conclusions
[ True  True  True  True  True  True  True  True  True  True  True]

Adjusted P Values
[0.00000000e+00 3.64767533e-07 8.02130909e-07 9.04334704e-47
 0.00000000e+00 5.17524929e-22 8.66919924e-29 5.08917938e-06
 1.99086113e-03 1.43101880e-29 7.01008733e-15]
```

In [85]:

```
significant_features = fsr_10_chosen_features[p_adjusted[0][1:] == True]
print(significant_features)
```

```
['host_identity_verified' 'latitude' 'longitude' 'accommodates'
 'availability_30' 'number_of_reviews' 'review_scores_rating'
 'calculated_host_listings_count_shared_rooms' 'private' 'shared']
```

Refitting the Model

Lastly, we refit the model with only the significant variables. In addition, we can compare the coefficient values before and after the removal of insignificant variables to determine whether any sudden changes have occurred. However, since all variables were found to be significant, this does not provide additional insight in this case.

In [86]:

```
# refit model

X = df_inference_scaled[significant_features]
y = df_inference["log_price"]

final_reg = LinearRegression().fit(X, y)
print(final_reg.score(X, y))

print(final_reg.coef_)
print(final_reg.intercept_)
```

```
0.5376176530105613
[ 0.04923393 -0.04815427 -0.13352328  0.55494923  0.08971337 -0.102695
91
  0.04327761 -0.03778976 -0.10892666 -0.08202703]
7.09735334305396
```

In [87]:

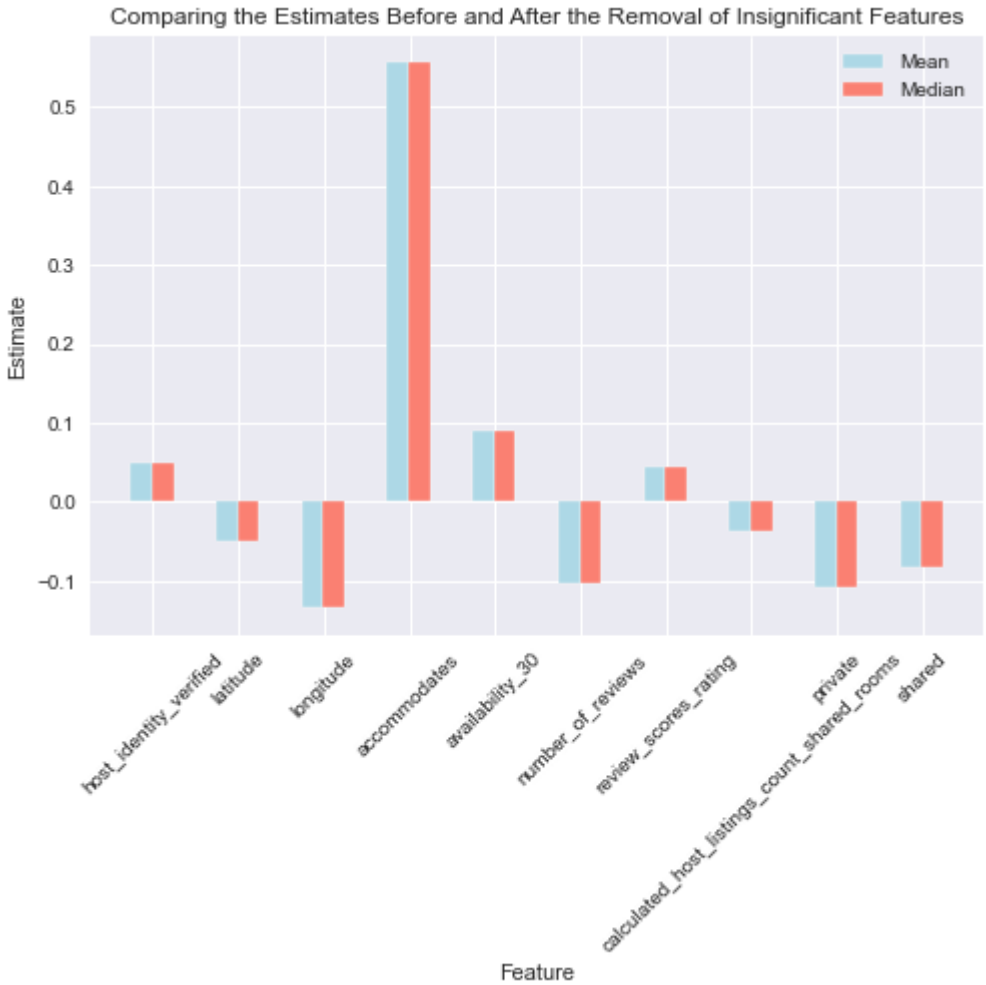
```
estimate_df = pd.DataFrame({"Feature": significant_features, "Before": reg.coef_[p_a
estimate_df.head(11)
```

Out[87]:

	Feature	Before	After
0	host_identity_verified	0.049234	0.049234
1	latitude	-0.048154	-0.048154
2	longitude	-0.133523	-0.133523
3	accommodates	0.554949	0.554949
4	availability_30	0.089713	0.089713
5	number_of_reviews	-0.102696	-0.102696
6	review_scores_rating	0.043278	0.043278
7	calculated_host_listings_count_shared_rooms	-0.037790	-0.037790
8	private	-0.108927	-0.108927
9	shared	-0.082027	-0.082027

In [88]:

```
n=(len(estimate_df["Feature"]))
r = np.arange(n)
width = 0.25


plt.bar(r - 0.5*width, estimate_df["Before"], color = 'lightblue',
        width = width,
        label='Mean')
plt.bar(r + 0.5*width, estimate_df["After"], color = 'salmon',
        width = width,
        label='Median')

plt.xlabel("Feature")
plt.ylabel("Estimate")
plt.xticks(r + width/25, estimate_df["Feature"], rotation = 45)
plt.legend()
plt.title("Comparing the Estimates Before and After the Removal of Insignificant Fea
plt.show()
```



8. Interpreting Results

With respect to the overarching research questions, the main factors influencing AirBnB price in Cape Town were found to be the number of guests accommodated, longitude, the number of reviews and whether the property type was private or shared. Notably, all variables in the final model selected were found to be significant due to the extensive data exploration and model selection procedure. While all variables were verified as significant, the plot comparing coefficient magnitudes indicates the dominance of the *accommodates* variable over the other included variables. The magnitudes of the coefficient values are further explored below.

Interpretation of Coefficient Values

The selected variables and corresponding coefficient values are in line with intuition and the expectations formed through the data exploration process, with *accommodates* having a large positive coefficient value of 0.555. Additionally, longitude has a negative coefficient of -0.134 while *private* and *shared* also indicate a negative relationship with price, with coefficients -0.109 and -0.082 respectively. The final coefficient values match expectations from the data exploration process which suggested a strong positive relationship between the number of guests accommodated and price as well as a negative relationship with *longitude*, as displayed in the corresponding visualisations in Section 5. The remaining variables have small contributions and provide complementary information such as the *review_scores_rating* or whether the host's identity is verified. However, these effects are very small in comparison to the main variables identified.

In addition the observed results match with intuition since *shared* accommodation or *private* room type would be less expensive than the alternative property types, an *entire* property or *hotel* room would be pricier options. The strong positive relationship between the number of guests accommodated and price is intuitive as a higher number of guests corresponds to a larger property which are more expensive. Notably, the negative relationship with *longitude* can be interpreted in light of the geographical layout of Cape Town, where more expensive areas such as those along the Atlantic Seaboard correspond to a lower longitude value, while more inland and less expensive suburbs correspond to a higher longitude value. Therefore we can conclude that location does influence AirBnB price although not to the same extent as property size.

The Role of Property Size vs Location

A slightly unexpected result is the dominance of the number of guests accommodated as the main variable influencing variation in AirBnB price. While the number of guests intuitively corresponds to the size of the property, additional variables such as proximity to major tourist attractions and beaches depending on the location of the property was expected to play a stronger role. Determining the cause behind the discrepency in the relative importance between property size and location is recommended for further study.

Assumptions and Limitations

It can be noted that a large proportion of the variance is left unexplained by the model, with a final R^2 score of 0.538 on the inference set. Notably, information contained in the textual features in the *InsideAirBnB* dataset was not utilised in this analysis and may contribute towards the unexplained variance. Moreover, the dataset is limited to the 74 variables scraped from the AirBnB website and may also lack certain information that contributes towards the unexplained variance.

To obtain an even better idea of the generalisation ability of the model, the model could be tested on another dataset, or another month from the same dataset. This could provide further insight into whether the identified variables continue to exhibit a significant relationship with *price* across datasets. Moreover, an improved method of handling collinear variables may provide great value due to the high presence of groups of collinear variables in the dataset.

9. Conclusion

In this investigation, the relationships between the factors influencing AirBnB price in Cape Town were investigated through a linear regression model using the *Inside AirBnB* dataset. Through model selection, ten significant features were identified and verified through a significance test, taking into account the effects of multiple testing. Due to the presence of multiple groups of highly correlated variables, collinear features were removed to prevent distortion of the variable coefficients. It was determined that the size of the property has the greatest impact on the corresponding price with factors such as the number of guests accommodated and closely related variables such as the number of bedrooms and bathrooms having a strong influence on the associated price. While factors relating to the location of the AirBnB listing were found to have an effect on the price, the overall impact of location was much lower than expected, suggesting that the granularity of the location predictors considered was inadequate. Moreover, a large amount of variance was left unexplained by the model, warranting further exploration. Overall, through careful model selection, the most significant variables present in the dataset were identified and the relationships between each feature and AirBnB price were analysed.

Recommendations for Future Research

Several recommendations are made for future investigation:

1. The relationship between AirBnB location and price can be further analysed by examining new methods to aggregate the location variables present in the dataset or by augmenting the dataset with additional information pertaining to location such as distance to tourist attractions or average income per ward.

2. The scope of the investigation can be broadened to include more months. This could allow for a comparative study between months or time series modelling to understand the effects of seasonality on AirBnB price.

3. Different cities could be compared to discover general trends across the distribution of AirBnB price and whether individual cities display different relationships between the predictors and the price variable.

4. Further processing of the textual features available in the *Inside AirBnB* dataset is recommended to determine their relationship with price.

5. Lastly, further investigation into possible features that could clarify the remaining unexplained variance is warranted due to the notable amount of variation left unaccounted for by the variables in the *InsideAirBnB* dataset.

References

[1] Murray Cox. Inside airbnb. http://insideairbnb.com/get-the-data.html (http://insideairbnb.com/get-the-data.html), 2021. Accessed: 2021-10-01.

[2] Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. "The elements of statistical learning.", 2009. Accessed: 2021-10-01.

[3] Andrew Paskaramoorthy and Terence van Zyl. Data exploration and analysis.https://courses.ms.wits.ac.za/andrew/Notes/ (https://courses.ms.wits.ac.za/andrew/Notes/), 2020. Accessed: 2021-08-17