Databootcamp Final Project: Airbnb Listing Prices and Reviews & COVID-19

Haanbi Kim, Christian Sarkis, Jennifer Zhang

Objective

The objective of this project is to take Airbnb data from NYC, Paris, and Sydney (cities with high levels of housing costs) and analyze 3 different scenarios amongst the listings.

1. How Price and Superhost Status vary with listing characteristics
2. How closely the prices of Airbnb listings mirror actual hotel listings
3. How volume of reviews changed with the onset of COVID-19

Instructions

1. Create a repository on GitHub and name it: *data_bootcamp_final_project*

1. Populate the repository on GitHub with the following items

- A readme file (GitHub should populate a blank one for you), that has very specific language specified on the next page. I will not grade your project until this step is complete.
- Your Jupyter Notebook. This should be titled in the following way: *lastname_final_project.ipynb*
- A .pdf file of your Jupyter Notebook. This should be titled in the following way: *lastname_final_project.pdf* How to create this: Click on File/Print Preview/Print/ and then there you should be able to select "Save as PDF"
- A powerpoint file (five slides max) uses your graphics to tell the story your project explores. This should be titled in the following way: *lastname final project presentation.ppt*

1. Upload a .ipynb notebook to NYU classes which contains only the Readme File below and the hyper-link to your repository.

Readme file should contain the following:

This project was completed by insert full name here in partial fulfilment of ECON-UB.0232, Data Bootcamp, Spring 2021. I certify that the NYU Stern Honor Code applies to this project. In particular, I have: Clearly acknowledged the work and efforts of others when submitting written work as our own. The incorporation of the work of others–including but not limited to their ideas, data, creative expression, and direct quotations (which should be designated with quotation marks), or paraphrasing thereof– has been fully and appropriately referenced using notations both in the text and the bibliography. And I understand that: Submitting the same or substantially similar work in multiple courses, either in the same semester or in a different semester, without the express approval of all instructors is strictly forbidden. I acknowledge that a failure to abide by NYU Stern Honor Code will result in a failing grade for the project and course.

And, a one paragraph project description

Description: In this project, we examined the Airbnb listings in New York City, Sydney, and Paris. We compared the the types of listings and their prices across the cities as well as with local renting costs. Additionally, we looked at the impact of the Covid-19 pandemic on these cities in relation to each city's lockdown policy.

Grading:
Projects will be graded on their overall quality. This includes, but is not restricted to, these categories:

- Did you follow directions. Turn in data report on time, name files correctly, etc.
- Quality of the idea. Is the question clearly articulated? Is it interesting?
- Quality of the data. Does the data support the idea? Is it the best data for this question?
- Quality of the code. Is the code readable? Could someone else understand what you were doing and why?
- Degree of difficulty. Some ideas are harder than others to implement. As in Olympic diving, you get credit for taking on a challenge.
- Professional look. Does your project look professional? Are the graphs easy to understand? Are they clearly labeled?

Link to Google Slides Requirement:
https://docs.google.com/presentation/d/1WrobBqffEtvr5vmcedFVKRVN75nZcoheW2GOUoZsgOQ/edit#slide=id.p

Preliminaries

Data is obtained from: http://insideairbnb.com/get-the-data.html
Using listing and review data for the following cities: NYC, Paris, Sydney
Rationale for selected cities: Are known to have high costs of living and housing and are located in entirely different continents

In [1]:
```
# import packages (potential packages to use)
import pandas as pd
from pandas import datetime
import matplotlib.pyplot as plt
```

```
import matplotlib as mpl
import numpy as np
import plotly.express as px
from plotly.subplots import make_subplots
import statsmodels.formula.api as smf
from urllib.request import urlopen

from sklearn.linear_model import LinearRegression as linreg
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR as svr

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

```
<ipython-input-1-70e0718efb56>:3: FutureWarning: The pandas.datetime class is deprecated and will be removed from pandas
in a future version. Import from datetime module instead.
  from pandas import datetime
```

In [7]:
```
# import datasets
df_nyc = pd.read_csv('DatabootcampFinal/nyclistings.csv')
df_par = pd.read_csv('DatabootcampFinal/parislistings.csv')
df_syd = pd.read_csv('DatabootcampFinal/sydneylistings.csv')
```

In [10]:
```
# import detailed datasets
df_nyc_det = pd.read_csv('DatabootcampFinal/listingsnyc_detailed.csv')
df_par_det = pd.read_csv('DatabootcampFinal/listingsparis_detailed.csv')
df_syd_det = pd.read_csv('DatabootcampFinal/listingssydney_detailed.csv')

# import reviews data
df_nyc_rev = pd.read_csv('DatabootcampFinal/reviewsnyc.csv')
df_par_rev = pd.read_csv('DatabootcampFinal/reviewsparis.csv')
df_syd_rev = pd.read_csv('DatabootcampFinal/reviews_sydney.csv')
```

In [11]:
```
# modify price column
df_nyc_det['price'] = df_nyc_det['price'].str.replace('$', '')
df_nyc_det['price'] = pd.to_numeric(df_nyc_det['price'].str.replace(',', ''))

df_par_det['price'] = df_par_det['price'].str.replace('$', '')
df_par_det['price'] = pd.to_numeric(df_par_det['price'].str.replace(',', ''))
```

```
df_syd_det['price'] = df_syd_det['price'].str.replace('$', '')
df_syd_det['price'] = pd.to_numeric(df_syd_det['price'].str.replace(',', ''))
```

In [12]:
```
# add new column 'city
df_nyc['city'] = 'NYC'
df_nyc_det['city'] = 'NYC'
df_nyc_rev['city'] = 'NYC'
```

In [13]:
```
# make distinction for city and change price in terms of usd (assuming prices are in euros)
# currency conversion data is from April 2021 (~ 1.2 USD per Euro)
df_par['city'] = 'PAR'
df_par_det['city'] = 'PAR'
df_par_rev['city'] = 'PAR'
df_par['price'] = df_par_det['price'] * 1.2
df_par_det['price'] = df_par_det['price'] * 1.2
```

In [14]:
```
# make distinction for city and change price in terms of usd (assuming prices are in australian dollars)
# currency conversion data is from April 2021 (~ 0.77 USD per AUD)
df_syd['city'] = 'SYD'
df_syd_det['city'] = 'SYD'
df_syd_rev['city'] = 'SYD'
df_syd['price'] = df_syd_det['price'] * 0.77
df_syd_det['price'] = df_syd_det['price'] * 0.77
```

In [15]:
```
# merged general data
df_comp = pd.concat([df_nyc, df_par, df_syd], ignore_index = True)
```

In [16]:
```
# merged review data
df_comp_rev = pd.concat([df_nyc_rev, df_par_rev, df_syd_rev], ignore_index = True)
```

In [17]:
```
# merged detailed data
df_comp_det = pd.concat([df_nyc_det, df_par_det, df_syd_det], ignore_index = True)
```

In [19]:
```
# style sheet
plt.style.use('ggplot')
```

1. Airbnb Listings: Prices and Hosts

In this following section, we will be conducting data analysis on several different price- and listing-related factors of the dataset, in two parts.

1) Linear regression models predicting price against several listing characteristics
2) Classification models predicting superhost status against several host characteristics

Regressions on Listing Prices with respect to Different Factors

Independent Variables: ['review_scores_rating', 'bedrooms', 'amenities_num']

Dependent Variable: 'price'

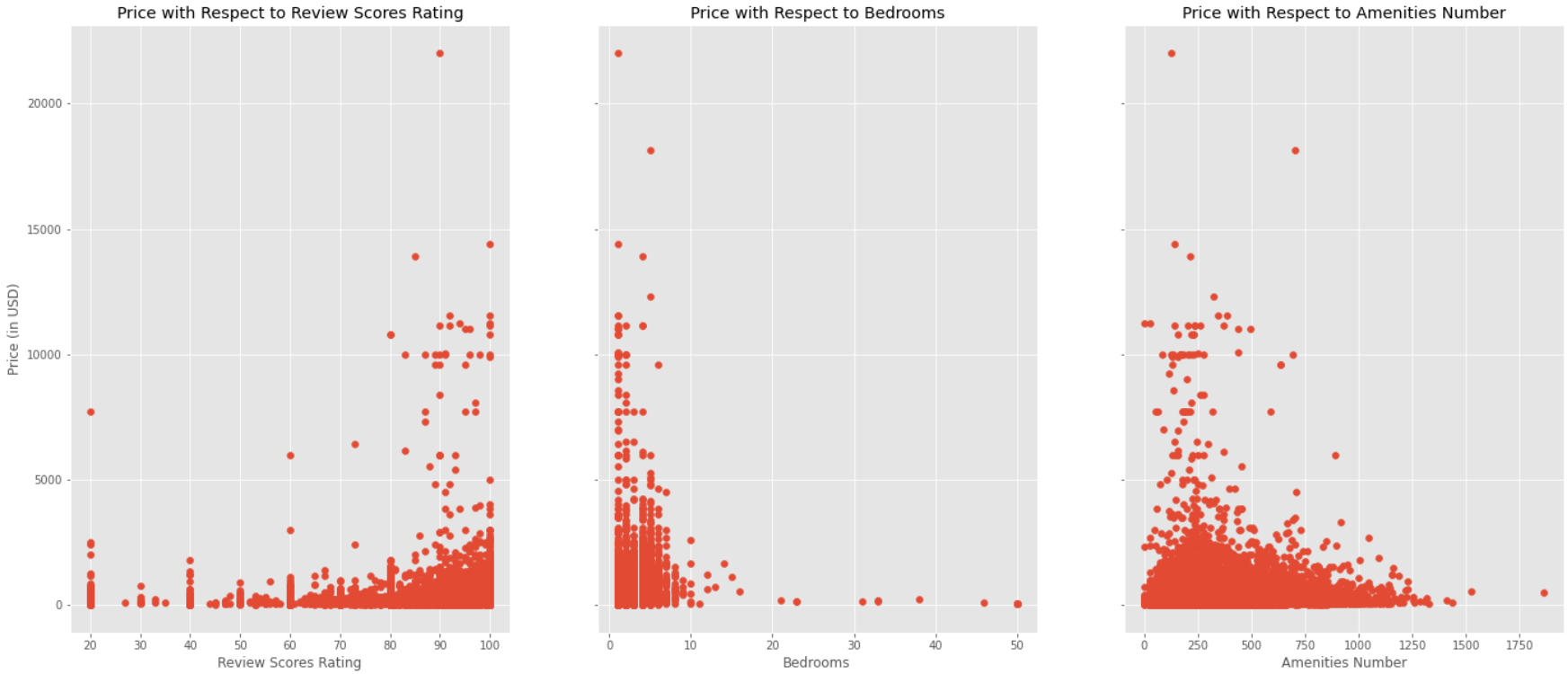
In this section, we will be seeing how listing prices in New York City, Paris, and Sydney vary with the independent variables in separate models. Independent variables are chosen because they intuitively would influence the price of a house or apartment, so we are seeing whether this logic would apply to the data in these models, espeically across these three cities which boast the highest levels of rent in the world. We expect that these variables will have some kind of a positive correlation with prices.

First, we will preview the scatterplots for each variable to see whether any adjustments need to be made and create a new column that represents the number of amenities per listing

In [23]:
```
# track number of amentities per listing by creating blank column and filling values out with for loop
df_comp_det['amenities_number'] = 0
for i in range(len(df_comp_det)):
    df_comp_det['amenities_number'][i] = len(df_comp_det['amenities'][i])
```

```
<ipython-input-23-ed5e6718c524>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_comp_det['amenities_number'][i] = len(df_comp_det['amenities'][i])
```

In [22]:
```
# visualize the data for the variables we are examining
fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(24,10), sharey = True)
columns = ['review_scores_rating', 'bedrooms', 'amenities_number']
for i in columns:
  ax[columns.index(i)].scatter(df_comp_det[i], df_comp_det['price'])
  ax[columns.index(i)].set_title('Price with Respect to ' + i.replace('_', ' ').title())
  ax[columns.index(i)].set_xlabel(i.replace('_',' ').title())
  if i == 'review_scores_rating':
    ax[columns.index(i)].set_ylabel('Price (in USD)')
```



All scatter plots show that linear regression may not be the best approach in predicting listing prices. We will predict by modifying price to its log form (i.e. ln('price')) and then create the linear regression models since the data points are shaped in more or less an exponential form, and doing so may help guide a better fit.

Functions

In [21]:
```
# function for regression model
def reg_create(dep, indep, dataset):
  command = '' + dep + ' ~ ' + indep
  reg = smf.ols(command, dataset).fit()
  print(reg.summary())

# function for plotting model
def reg_plot(dep, indep, dataset, y, x):
  command = '' + dep + ' ~ ' + indep
  reg = smf.ols(command, dataset).fit()
  dataset['yhat'] = reg.predict()

  fig,ax = plt.subplots()
  dataset.plot.scatter(x=indep, y=dep, ax=ax, figsize = (12,6), color = 'black')
```

```
    dataset.sort_values(indep).set_index(indep)['yhat'].plot(ax=ax, color = 'red', lw = 4, figsize=(12,6))


    ax.set_title(y + ' Prediction Plot Using ' + x)
    ax.set_ylabel(y)
    ax.set_xlabel(x)
```

In [25]:
```
# exclude null rows
df_comp_det_reg = df_comp_det.loc[df_comp_det['review_scores_rating'].notna()]

# create log column to adjust for the exponential shape
df_comp_det_reg['lnprice'] = np.log(df_comp_det_reg['review_scores_rating'])
```

```
<ipython-input-25-14537c5ba9a4>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_comp_det_reg['lnprice'] = np.log(df_comp_det_reg['review_scores_rating'])
```

Model 1: Review Scores Rating - Price (log)

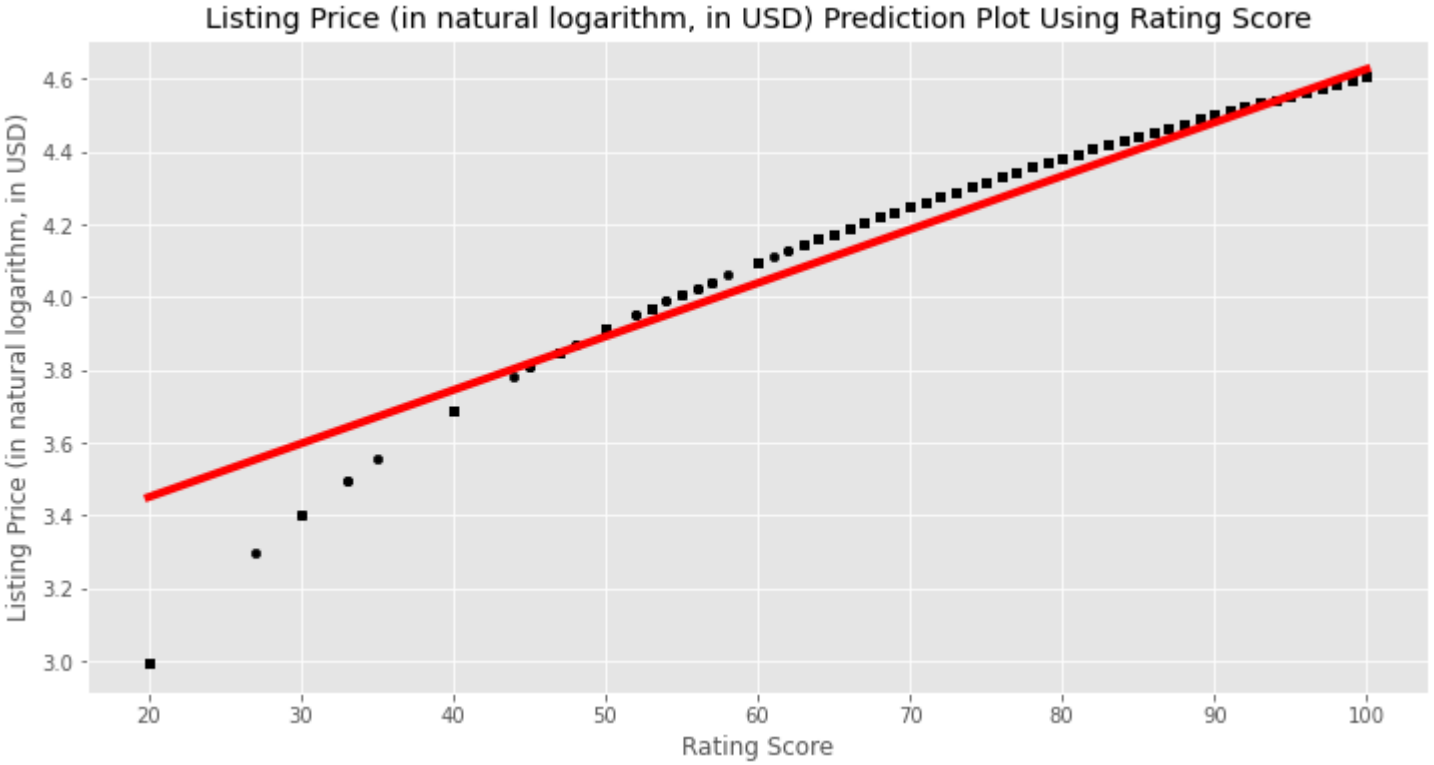
In [26]:
```
reg_create('lnprice', 'review_scores_rating', df_comp_det_reg)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                lnprice   R-squared:                       0.930
Model:                            OLS   Adj. R-squared:                  0.930
Method:                 Least Squares   F-statistic:                 1.283e+06
Date:                Mon, 17 May 2021   Prob (F-statistic):               0.00
Time:                        17:18:53   Log-Likelihood:             1.7781e+05
No. Observations:               97189   AIC:                        -3.556e+05
Df Residuals:                   97187   BIC:                        -3.556e+05
Df Model:                           1
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
Intercept                3.1565      0.001   2594.062      0.000       3.154       3.159
review_scores_rating     0.0147    1.3e-05   1132.720      0.000       0.015       0.015
==============================================================================
Omnibus:                   124864.497   Durbin-Watson:                   1.959
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         28979634.348
Skew:                          -7.116   Prob(JB):                         0.00
Kurtosis:                      86.389   Cond. No.                         916.
==============================================================================
```

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

In [27]:
```
reg_plot('lnprice', 'review_scores_rating', df_comp_det_reg, 'Listing Price (in natural logarithm, in USD)', 'Rating Scor
```

```
<ipython-input-21-c2a9caa57272>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-v
iew-versus-a-copy
  dataset['yhat'] = reg.predict()
```



Explanation and Flaws:

The model shows a high R-squared value, meaning that the model generally has a good fit, as the model explains 93% of the variation in the lnprice variable. Especially for rating scores that are above 40, the model predicts listing prices (in the natural log) quite efficiently. One flaw is that the model doesn't capture price for ratings that score less than 40, as seen in the graph above.

Model 2: Bedrooms - Price (log)

In [28]:
```
# regression 2 set up
df_comp_det_reg1 = df_comp_det.loc[df_comp_det['bedrooms'].notna()]
df_comp_det_reg1['lnprice'] = np.log(df_comp_det_reg1['price'])
df_comp_det_reg1['lnbedrooms'] = np.log(df_comp_det_reg1['bedrooms'])
```

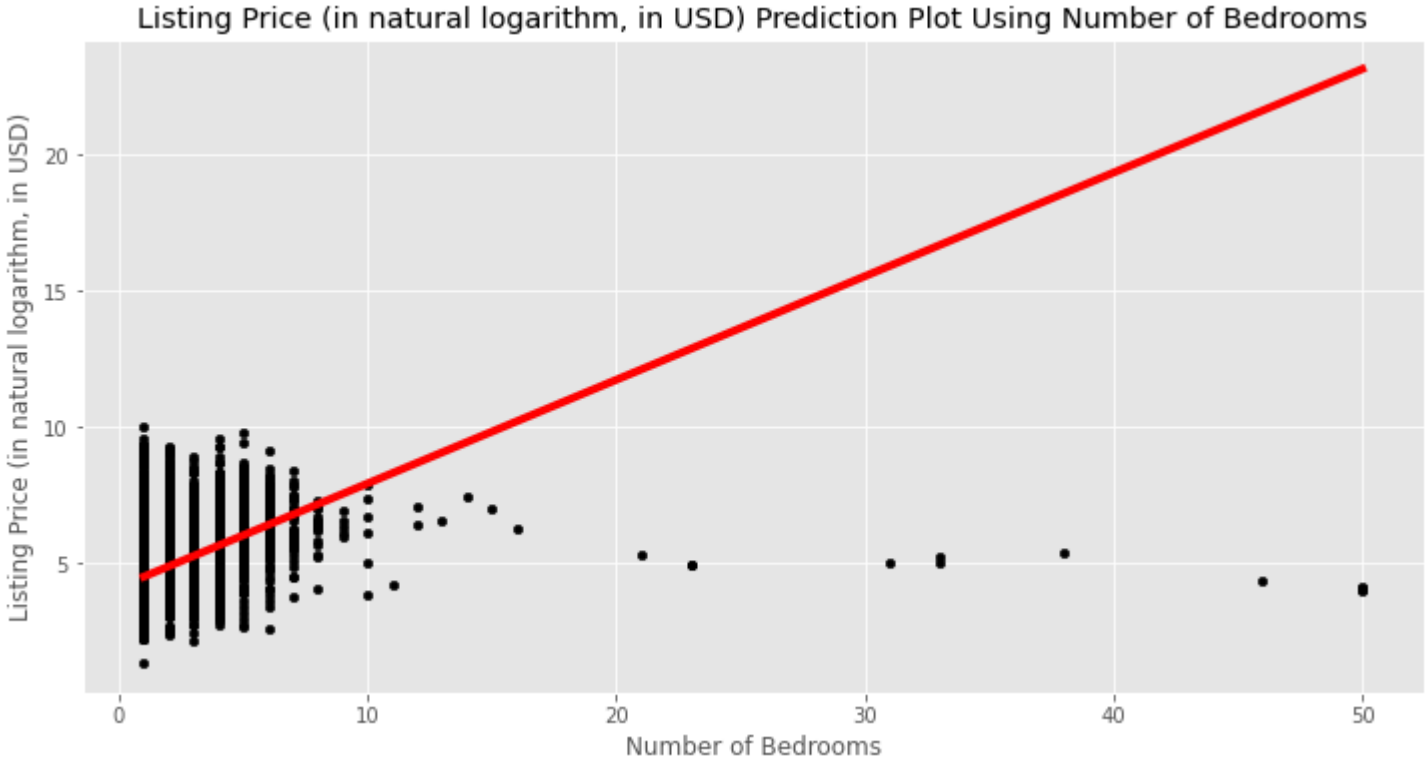
```
<ipython-input-28-c2a323291a35>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_comp_det_reg1['lnprice'] = np.log(df_comp_det_reg1['price'])
<ipython-input-28-c2a323291a35>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_comp_det_reg1['lnbedrooms'] = np.log(df_comp_det_reg1['bedrooms'])
```

In [29]:
```
# regression table
reg_create('lnprice', 'bedrooms', df_comp_det_reg1)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                lnprice   R-squared:                       0.228
Model:                            OLS   Adj. R-squared:                  0.228
Method:                 Least Squares   F-statistic:                 3.425e+04
Date:                Mon, 17 May 2021   Prob (F-statistic):               0.00
Time:                        17:19:06   Log-Likelihood:            -1.1536e+05
No. Observations:              116111   AIC:                         2.307e+05
Df Residuals:                  116109   BIC:                         2.308e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      4.1276      0.004   1173.809      0.000       4.121       4.135
bedrooms       0.3802      0.002    185.064      0.000       0.376       0.384
==============================================================================
Omnibus:                    60549.974   Durbin-Watson:                   1.756
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         14106414.726
Skew:                          -1.367   Prob(JB):                         0.00
Kurtosis:                      56.929   Cond. No.                         3.96
==============================================================================

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

In [30]:
```
# plotting regression model
reg_plot('lnprice', 'bedrooms', df_comp_det_reg1, 'Listing Price (in natural logarithm, in USD)', 'Number of Bedrooms')
```

```
<ipython-input-21-c2a9caa57272>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataset['yhat'] = reg.predict()
```



Explanation and Flaws:

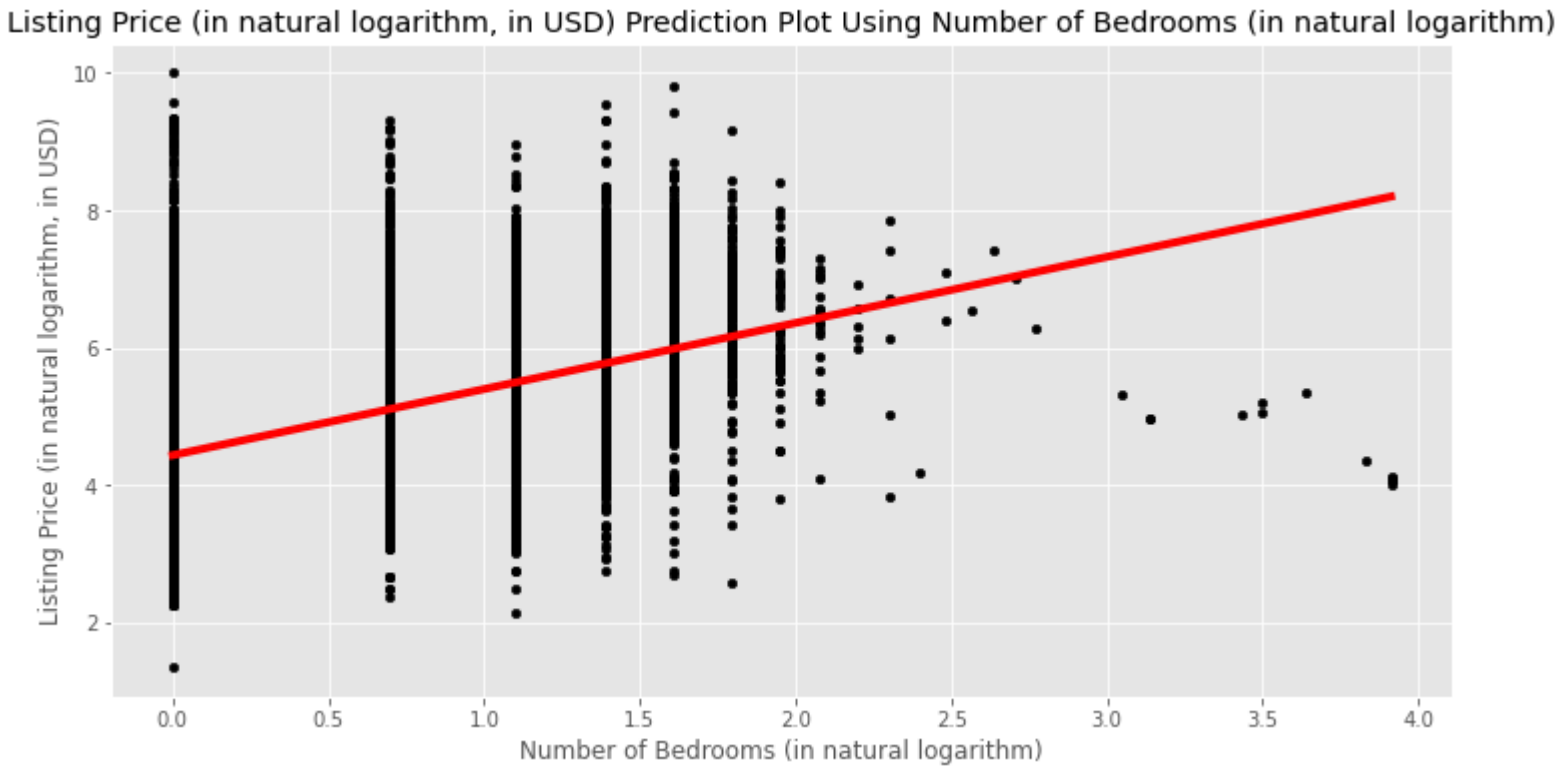
The model shows a lower R-squared value than that of the first regression model's. The trendline doesn't seem to fit the datapoints too accurately, though, and for listings with more than 10 bedrooms the fit becomes worse. Since the current model doesn't seem to show the rationale in the poorer fit all too well, we will try to scale the bedrooms axis so that the differences are more stable by taking log of the bedrooms column. This will help us focus on the more concentrated datapoints (i.e. listings with less than 10 bedrooms).

In [31]:
```
reg_create('lnprice', 'lnbedrooms', df_comp_det_reg1)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 lnprice   R-squared:                       0.301
Model:                             OLS   Adj. R-squared:                  0.301
Method:                  Least Squares   F-statistic:                 5.001e+04
Date:                 Mon, 17 May 2021   Prob (F-statistic):               0.00
Time:                         17:19:18   Log-Likelihood:            -1.0958e+05
No. Observations:               116111   AIC:                         2.192e+05
Df Residuals:                   116109   BIC:                         2.192e+05
Df Model:                            1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      4.4317      0.002   2090.502      0.000       4.428       4.436
lnbedrooms     0.9634      0.004    223.629      0.000       0.955       0.972
==============================================================================
Omnibus:                    13894.036   Durbin-Watson:                   1.723
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            47638.915
Skew:                           0.600   Prob(JB):                         0.00
Kurtosis:                       5.900   Cond. No.                         2.54
==============================================================================

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

In [32]:
```
reg_plot('lnprice', 'lnbedrooms', df_comp_det_reg1, 'Listing Price (in natural logarithm, in USD)', 'Number of Bedrooms (
```

```
<ipython-input-21-c2a9caa57272>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-v
iew-versus-a-copy
  dataset['yhat'] = reg.predict()
```



Additional Explanation and Flaws:

In this model, we can get a better understanding of the bulk of the data. The data are arranged in (generally) distinct category based on the number of bedrooms. We can see that the trendline seems to fit the data a little better with the adjustment, but the issue here is that because bedroom count is a discrete variable, and since more listings (regardless of price) have less bedrooms than more, the dispersion of listings for listings with less bedrooms is greater, thus the model has difficulty in capturing an accurate prediction for those sections.

TL;DR - The variance for listings where lnbedrooms = 0 (1 bedroom) is larger than the variance for listings where lnbedrooms = 2, so it is easier to predict the latter than the former, explaining the poorer fit.

Model 3: Number of Amenities - Price (log)

In [33]:
```
# regression 3 - set up
df_comp_det_reg2 = df_comp_det.loc[(df_comp_det['amenities_number'].notna()) & (df_comp_det['price'] > 0)]
df_comp_det_reg2['lnprice'] = np.log(df_comp_det_reg2['price'])
df_comp_det_reg2['lnamen'] = np.log(df_comp_det_reg2['amenities_number'])
```

```
<ipython-input-33-095b929455ae>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_comp_det_reg2['lnprice'] = np.log(df_comp_det_reg2['price'])
<ipython-input-33-095b929455ae>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_comp_det_reg2['lnamen'] = np.log(df_comp_det_reg2['amenities_number'])
```

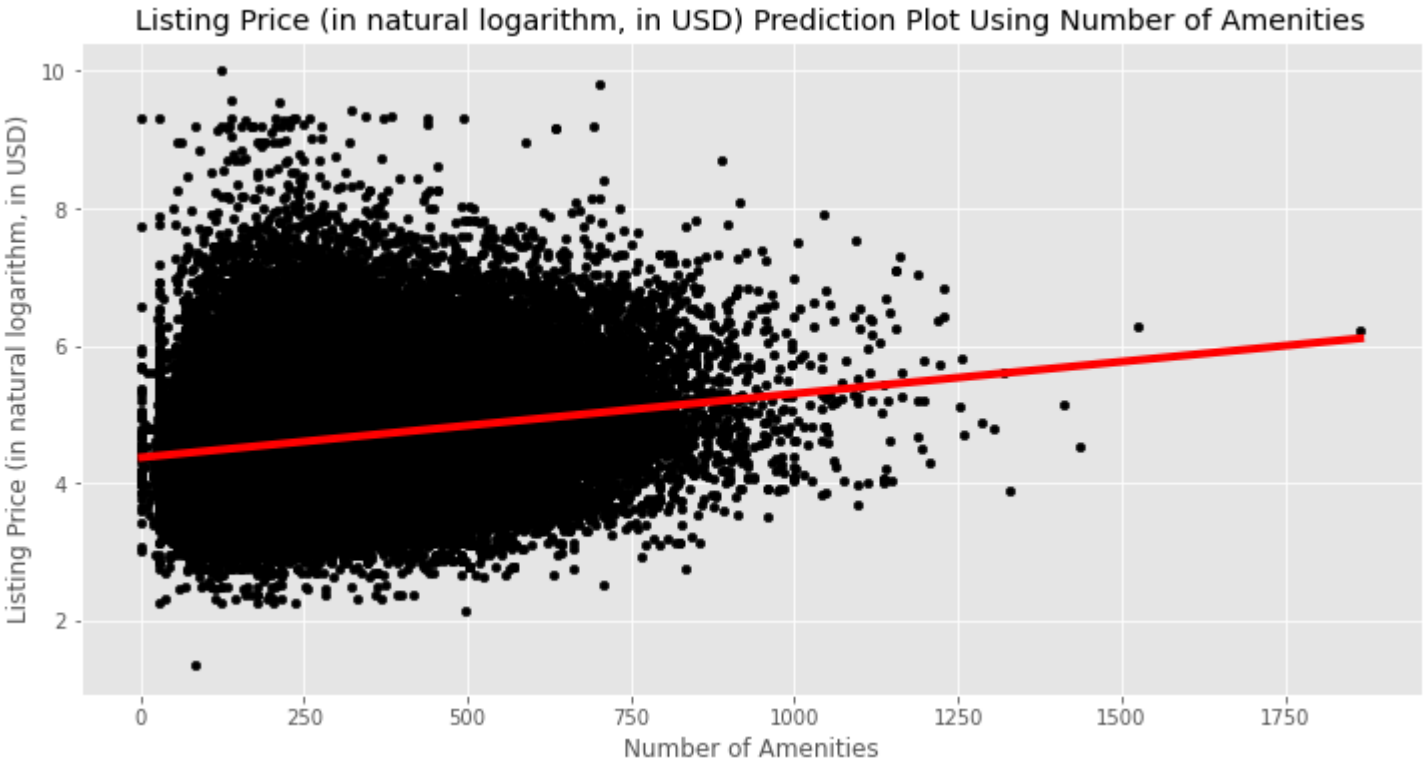
In [34]:
```
# regression table
reg_create('lnprice', 'amenities_number', df_comp_det_reg2)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                lnprice   R-squared:                       0.040
Model:                            OLS   Adj. R-squared:                  0.040
Method:                 Least Squares   F-statistic:                     5596.
Date:                Mon, 17 May 2021   Prob (F-statistic):               0.00
Time:                        17:19:23   Log-Likelihood:            -1.4509e+05
No. Observations:              135242   AIC:                         2.902e+05
Df Residuals:                  135240   BIC:                         2.902e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept            4.3693      0.004   1080.462      0.000       4.361       4.377
amenities_number     0.0009   1.24e-05     74.808      0.000       0.001       0.001
==============================================================================
Omnibus:                    19245.470   Durbin-Watson:                   1.793
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            44475.074
Skew:                           0.835   Prob(JB):                         0.00
Kurtosis:                       5.259   Cond. No.                         683.
==============================================================================

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

In [35]:
```
# plotting regression model
reg_plot('lnprice', 'amenities_number', df_comp_det_reg2, 'Listing Price (in natural logarithm, in USD)', 'Number of Amer
```

```
<ipython-input-21-c2a9caa57272>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-v
```

```
iew-versus-a-copy
  dataset['yhat'] = reg.predict()
```



Explanation and Flaws:

The R-squared for this model is the lowest compared to the other two models, with the regression model explaining only 4% of the variation in the lnprice variable. As seen in the visualization of the regression model, the data points are dispersed greatly, more than the bedrooms variable. As a result, it would be difficult for a linear regression model to capture all of the variation. However, what the model is able to show is an overall trend, which is what we are looking for as well. The number of amenities at an Airbnb stay generally correlates with a moderate positive increase in prices, which is similar to our initial hypothesis.

Conclusion:

Given the past three regression models, it can be said that our intuition of what ideally raises the prices of a listing price comes slightly in line with the results. Listing prices have a higher correlation and fit with respect to review scores, and lower correlation and fit with respect to the number of bedrooms and amenities. This hints at how pricing of listings will depend on explicit reviews of the experiences of previous people who have lived at the stay, rather than more physical qualities of the listing.

Classification on Superhosts and Non-Superhosts

In this section, we will be creating a classification model to see how much influence the number of reviews, review ratings, and response rates separately have on whether a host is a superhost or not.
Independent variable is determined based on the intuitive correlation one would draw between the potential independent variable and superhost status.
Classification models are initially attempted using a Logistic Regression model, and any accommodations will be made later if we see any issues with Logistic Regression.

Independent Variables Used: ['number_of_reviews', 'host_response_rate']
Dependent Variable: 'host_is_superhost'

Functions

In [37]:
```
# general function for confusion matrix
def class_confusion_matrixpx(confuse_matrix):
  import plotly.figure_factory as ff
  confuse_matrix = (confuse_matrix/confuse_matrix.sum()).round(5)
  x = ['not_superhost', 'superhost']
  y = ['not_superhost', 'superhost']
  fig = ff.create_annotated_heatmap(confuse_matrix, x=x, y=y, colorscale='Viridis')
  fig.update_layout(title_text='<i><b>Confusion matrix: Number of Reviews and Superhost Status</b></i>',
                    #xaxis = dict(title='x'),
                    #yaxis = dict(title='x')
                   )

  fig.add_annotation(dict(font=dict(color="black",size=14),
                          x=-0.15,
                          y=0.5,
                          showarrow=False,
                          text="Actual value",
                          textangle=-90,
                          xref="paper",
                          yref="paper"))

  fig.add_annotation(dict(font=dict(color="black",size=14),
                          x=0.5,
                          y=-0.15,
                          showarrow=False,
                          text="Predicted value",
                          xref="paper",
                          yref="paper"))
```

```
  fig.update_layout(margin=dict(t=50, l=200))
  fig.show()
```

In [38]:
```
# function for creating model and splitting test and train data
def class_model_create(indep, dep, dataframe, model_type, n):
  global X_train, X_test, y_train, y_test
  X_train, X_test, y_train, y_test = train_test_split(dataframe[[indep]], dataframe[dep], test_size=.2)
  if model_type == 'logreg':
    global logis
    logis = LogisticRegression().fit(X_train, y_train)
  elif model_type == 'knn':
    global knn_class
    knn_class = KNeighborsClassifier(n_neighbors=n).fit(X_train, y_train)
```

In [39]:
```
# function for normal confusion matrix
def class_confusion_matrixnorm(classvar, indep, dep, dataframe):
  from sklearn.metrics import confusion_matrix
  global pred_val
  pred_val = classvar.predict(dataframe[[indep]])
  global cfm
  cfm = confusion_matrix(dataframe[[dep]], pred_val)
  print(cfm)
```

In [40]:
```
# function for scoring
def class_score(classvar, indep, dep, dataframe):
  print("Score from testing data: ", classvar.score(X_test, y_test),
        '\n', '\nScore from total data: ', classvar.score(dataframe[[indep]], dataframe[dep]))
```

In [41]:
```
# function displaying scores, classification report, and confusion matrix of the model
def class_model_summary(dataframe, classvar, indep, dep):
  print("Score from testing data: ", classvar.score(X_test, y_test),
        '\n', '\nScore from total data: ', classvar.score(dataframe[[indep]], dataframe[dep]))
  print()

  from sklearn.metrics import confusion_matrix
  global pred_val
  pred_val = classvar.predict(dataframe[[indep]])
  global cfm
  cfm = confusion_matrix(dataframe[[dep]], pred_val)
  print(cfm)

  print()
  print(classification_report(dataframe[[dep]], pred_val))
```

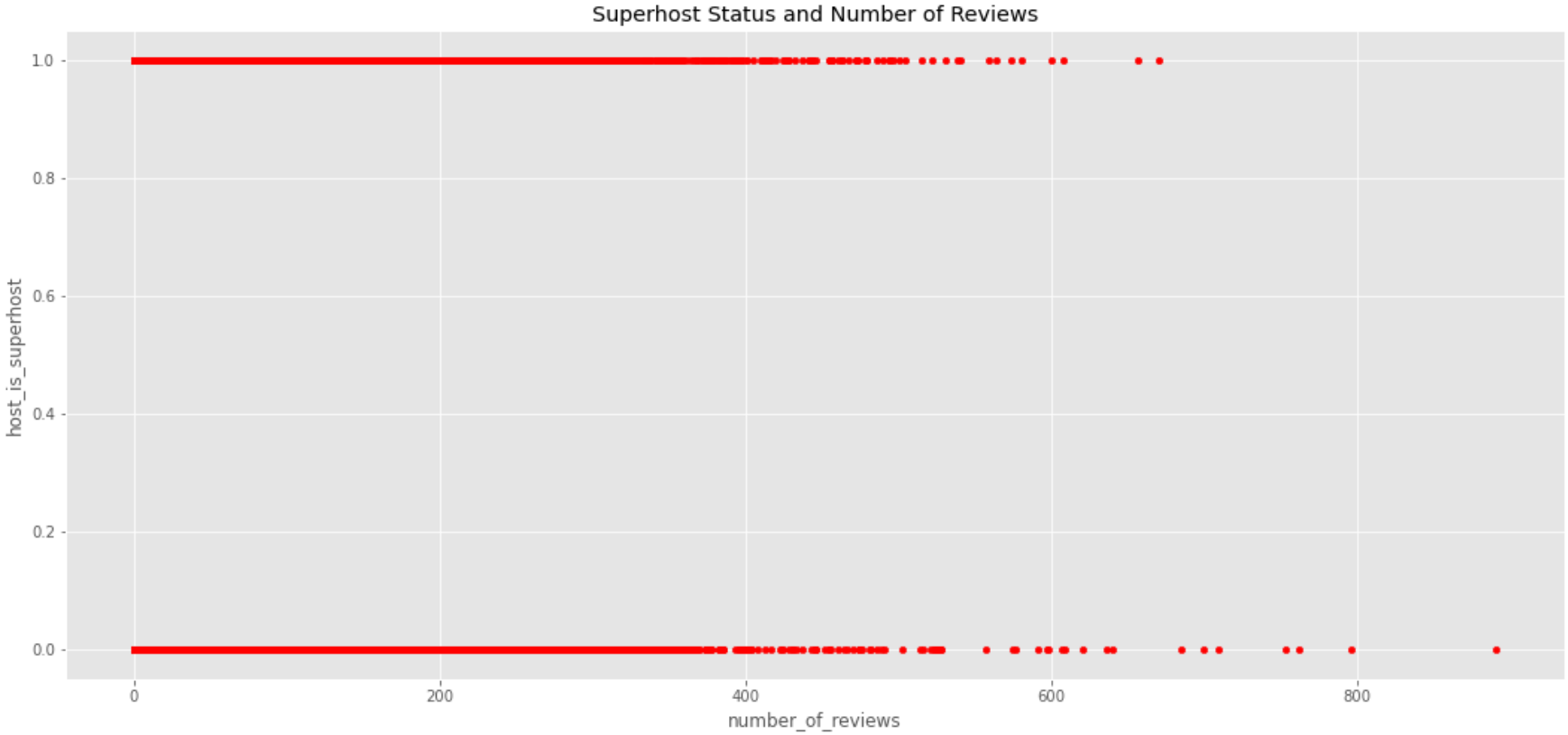
Classification 1: Number of Reviews - Host is Superhost

In [42]:
```
# make superhost column categorical and encode it so that True = 1 and False = 0
df_comp_det = df_comp_det.replace({'host_is_superhost': {'t': 1, 'f': 0}})
```

In [43]:
```
# exclude observations that are null
df_comp_detml = df_comp_det.loc[df_comp_det['host_is_superhost'].notna()]
```

In [44]:
```
# visualization of datapoints
df_comp_detml.plot.scatter(x='number_of_reviews', y='host_is_superhost', figsize = (18,8), title = 'Superhost Status and
```

Out[44]:
```
<AxesSubplot:title={'center':'Superhost Status and Number of Reviews'}, xlabel='number_of_reviews', ylabel='host_is_super
host'>
```



In [45]:
```
class_model_create('number_of_reviews', 'host_is_superhost', df_comp_detml, 'logreg', 5)
```

In [46]:
```
class_model_summary(df_comp_detml, logis, 'number_of_reviews', 'host_is_superhost')
```

```
Score from testing data:  0.8566358595194085
```

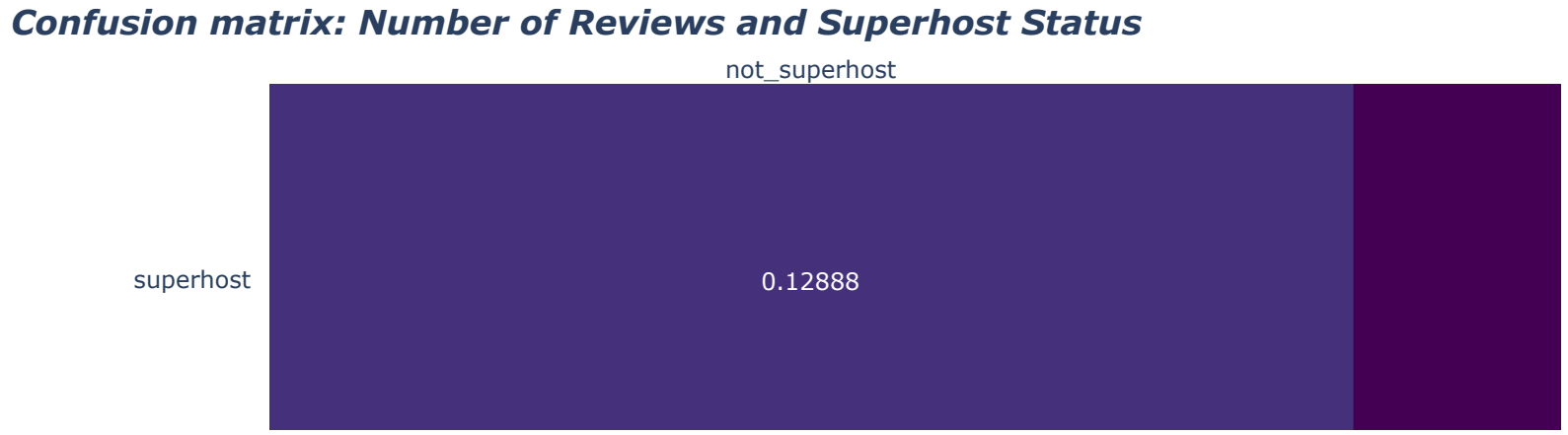
```
Score from total data:  0.8580966675785785

[[114324   1761]
 [ 17431   1731]]

              precision    recall  f1-score   support

         0.0       0.87      0.98      0.92    116085
         1.0       0.50      0.09      0.15     19162

    accuracy                           0.86    135247
   macro avg       0.68      0.54      0.54    135247
weighted avg       0.81      0.86      0.81    135247
```

Above are some of the summary statistics from this classification model, in order to specificity. At the top is the R-scored value for testing and total data (i.e. how well the model explains the variations in the host_is_superhost column). After that is a confusion matrix, which specifically shows how the model fares (top left: True Negative, top right: False Positive, bottom left: False Negative, bottom right: True Positive).

Finally, at the bottom is a classification report. The scores for "precision," "recall," and "f1-score" represent the accuracy of predictions from the model in each respective potential outcome (0 and 1). For simplicity of interpretation, the closer these values are to 1, the better. As seen in the report above, the accuracy of predictions for 1, observation where the host is a superhost, is worse (this concept will come into play later).

The following is a representation of a better confusion matrix visualization based on the proportion of responses.

In [47]:
```
class_confusion_matrixpx(cfm)
```



Explanation and flaws:
Given all of the information we have seen so far, we can see that the model does a good job of predicting that a host is not a superhost correctly for 85% of its total predictions. But, it also does a poor job in predicting that a host is a superhost, only predicting it correctly for 1% of its total predictions when superhost observations make up 14% of the total data. In this sense, its higher R-squared value may be misleading when it predominantly favors only one part of the data.

Classification 2: Host Response Rate - Host is Superhost

In [48]:
```
# exclude null values for independent variable -> the number of observations may be different from the number of observat
# due to the fact that there may be more null values for host response rates
df_comp_detml1 = df_comp_det.loc[df_comp_det['host_response_rate'].notna()]
```

In [49]:
```
# make datatype for host_response_rate compatible
df_comp_detml1['host_response_rate'] = df_comp_detml1['host_response_rate'].str.replace('%','').astype('float')
```

In [50]:
```
# create logistic regression model
class_model_create('host_response_rate', 'host_is_superhost', df_comp_detml1, 'logreg', 5)
```

In [51]:
```
class_model_summary(df_comp_detml1, logis, 'host_response_rate', 'host_is_superhost')
```

```
Score from testing data:  0.7574606619641888

Score from total data:  0.7571893651654911
```

```
[[41865     0]
 [13425     0]]

              precision    recall  f1-score   support

         0.0       0.76      1.00      0.86     41865
         1.0       0.00      0.00      0.00     13425

    accuracy                           0.76     55290
   macro avg       0.38      0.50      0.43     55290
weighted avg       0.57      0.76      0.65     55290

C:\Users\haanb\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1221: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` param
eter to control this behavior.
```

Bad classifying model using logistic regression as every observation is predicted as only one value, let's find a better version for a classification model.

In [52]:
```
# create knn classifier model
class_model_create('host_response_rate', 'host_is_superhost', df_comp_detml1, 'knn', 5)
```

In [53]:
```
class_model_summary(df_comp_detml1, knn_class, 'host_response_rate', 'host_is_superhost')
```

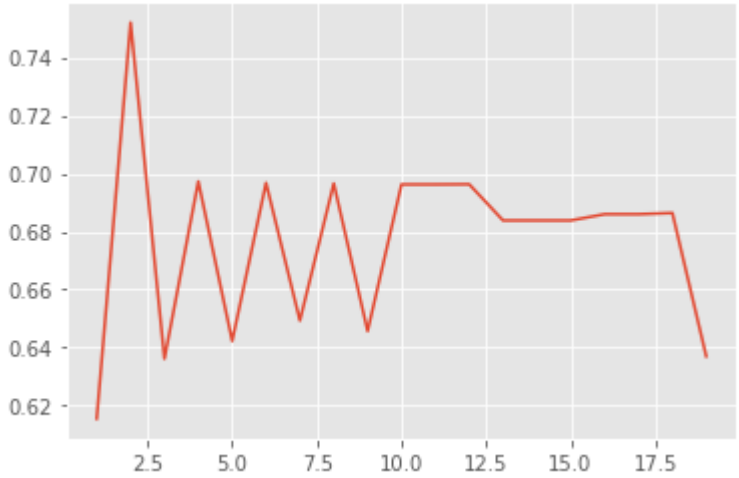
```
Score from testing data:  0.7495930548019534

Score from total data:  0.7573521432447097

[[41837    28]
 [13388    37]]

              precision    recall  f1-score   support

         0.0       0.76      1.00      0.86     41865
         1.0       0.57      0.00      0.01     13425

    accuracy                           0.76     55290
   macro avg       0.66      0.50      0.43     55290
weighted avg       0.71      0.76      0.65     55290
```

That looks a bit better as there is a large number of true positives, but there is also a larger number of false positives and lower true negatives than before (a tradeoff, in a way). Since this is a model that changes with the value that represents n_neighbors, we will see how the score of the model changes with different values of n_neighbhors that will be used to predict superhost status.

In [54]:
```
scores = pd.Series()
for i in range(1,20):
    scores.loc[i] = cross_val_score(KNeighborsClassifier(n_neighbors=i), X=df_comp_detml1[['host_response_rate']], y=df_c
scores.plot()
scores.idxmax()
```

```
<ipython-input-54-7bd640b9c544>:1: DeprecationWarning:

The default dtype for empty Series will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly
to silence this warning.
```

Out[54]: 2



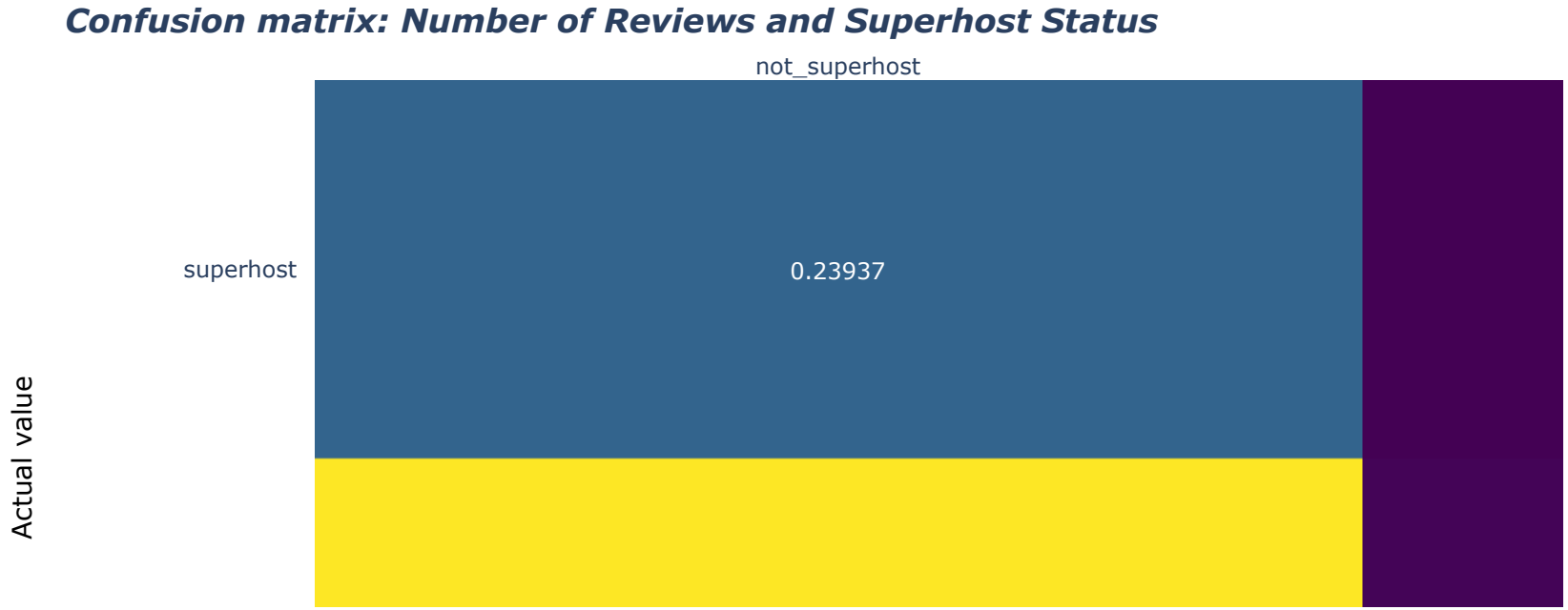
In [55]:
```
# creating new knn classifier model based on max number stipulated in the previous figure
class_model_create('host_response_rate', 'host_is_superhost', df_comp_detml1, 'knn', 2)

class_model_summary(df_comp_detml1, knn_class, 'host_response_rate', 'host_is_superhost')
```

```
Score from testing data:  0.761620546210888

Score from total data:  0.7570989328992584

[[41850    15]
 [13415    10]]

              precision    recall  f1-score   support

         0.0       0.76      1.00      0.86     41865
         1.0       0.40      0.00      0.00     13425

    accuracy                           0.76     55290
```

```
   macro avg       0.58      0.50      0.43     55290
weighted avg       0.67      0.76      0.65     55290
```

The score that is shown in the plots is misleading, as we get a similar problem as in the Classification 1 section where there is a smaller number of true positives (bottom right). Since the default value for n_neighbors brought better results in terms of predicting true positives, we will create a confusion matrix and classification report based on the default value for n_neighbors (5).

In [56]:
```
# readjust for the misleading results
class_model_create('host_response_rate', 'host_is_superhost', df_comp_detml1, 'knn', 5)
from sklearn.metrics import confusion_matrix

pred_val = knn_class.predict(df_comp_detml1[['host_response_rate']])
cfm = confusion_matrix(df_comp_detml1['host_is_superhost'], pred_val)

class_confusion_matrixpx(cfm)
```

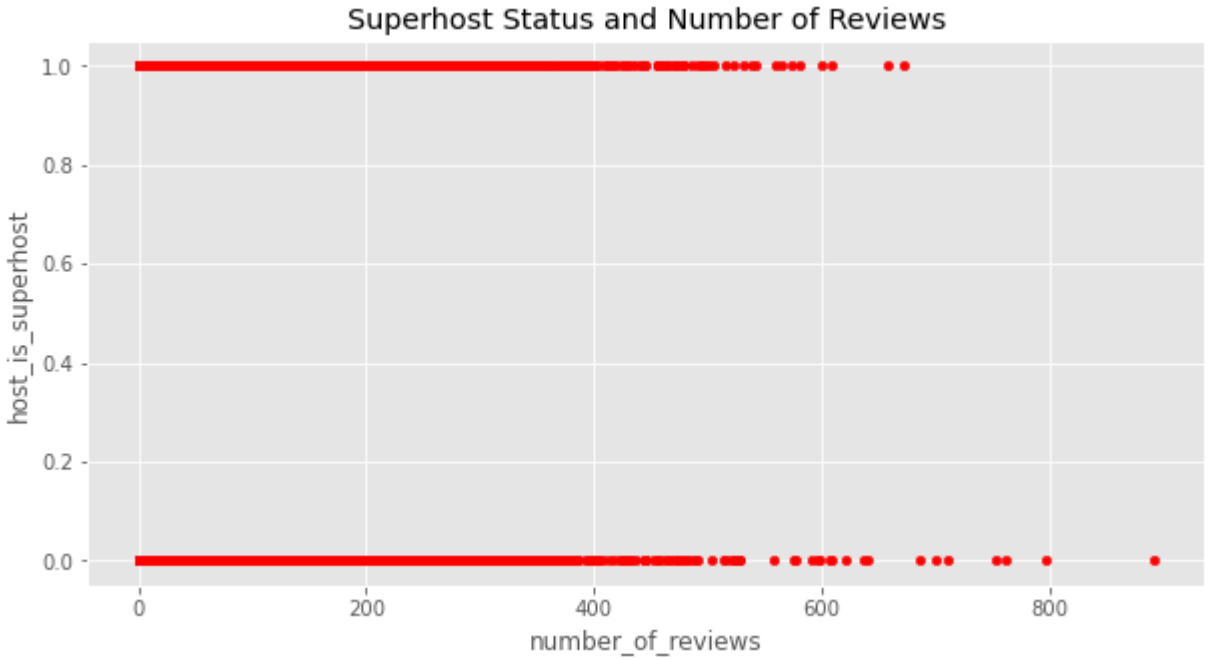


Explanation and flaws:
In this set of classification models, similar to the classification models, we have a trend where the rate of predicting hosts who aren't superhosts is higher than the rate of predicting hosts who are superhosts. This may suggest that the following two independent variables may not do the best job at predicting that the host is a superhost, but better at predicting that a host is not a superhost (better prediction on one side than the other).

Conclusion:
The classification models predict only one part of the data more efficiently than the other part overall, where both models predict that the host is a not a superhost with greater accuracy, while the predictions that the host is a superhost falls in accuracy. When visualizing the datapoints, we can get a possible explanation into this phenomenon. As seen below, there is more overlap in the sense that a potential host can be a superhost or not a superhost regardless of the number of reviews or its response rate. In other words, at 200 reviews, there are many instances where a host is a superhost and there are many instances where a host is not a superhost. This can suggest as to why the classification models perform poorly when predicting the observations that are superhosts.

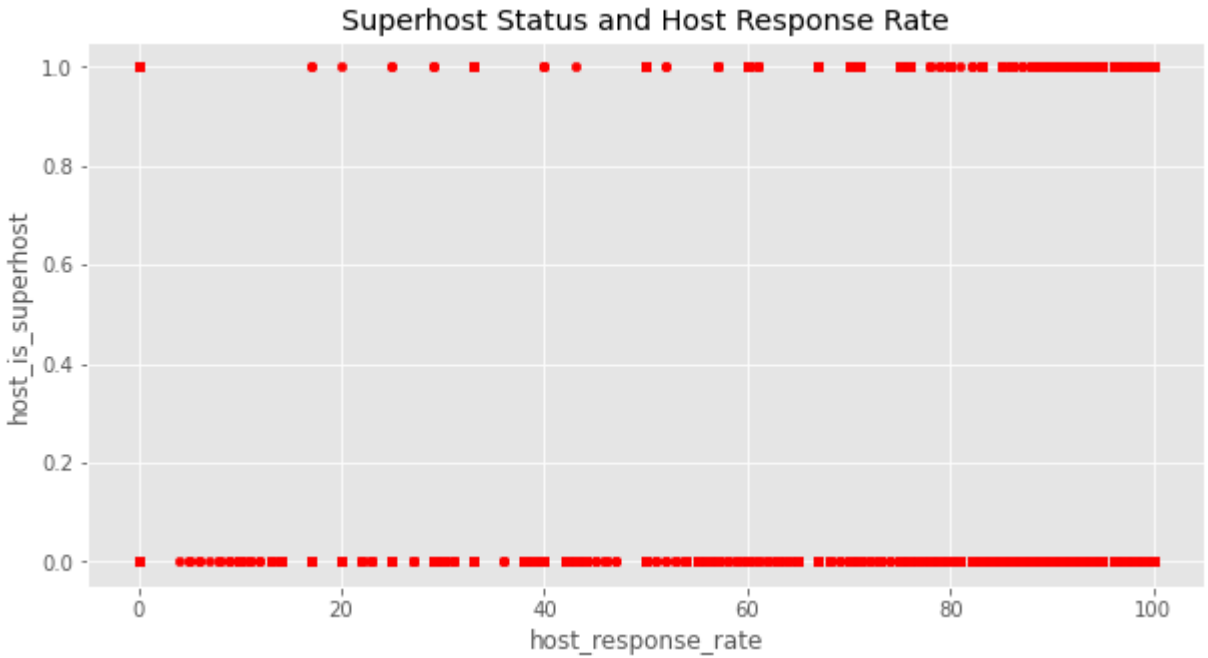
In [57]:
```
df_comp_detml.plot.scatter(x='number_of_reviews', y='host_is_superhost', figsize = (10,5), title = 'Superhost Status and
```

Out[57]:
```
<AxesSubplot:title={'center':'Superhost Status and Number of Reviews'}, xlabel='number_of_reviews', ylabel='host_is_super
host'>
```



In [58]:
```
df_comp_detml1.plot.scatter(x = 'host_response_rate', y = 'host_is_superhost', figsize = (10,5), title = 'Superhost Statu
```

Out[58]:
```
<AxesSubplot:title={'center':'Superhost Status and Host Response Rate'}, xlabel='host_response_rate', ylabel='host_is_sup
erhost'>
```



2. Airbnb Listings vs. Hotel Prices

Airbnb advertises itself as a cost effective alternative to traditional hotels. In order to test this, we decided to compare 2 bed private room Airbnb prices to the current average double hotel room rate (Trivago Hotel Price Index) in each city. The choice to use 2 bed private room listings was made to control for some of the many differences between hotels and Airbnbs.

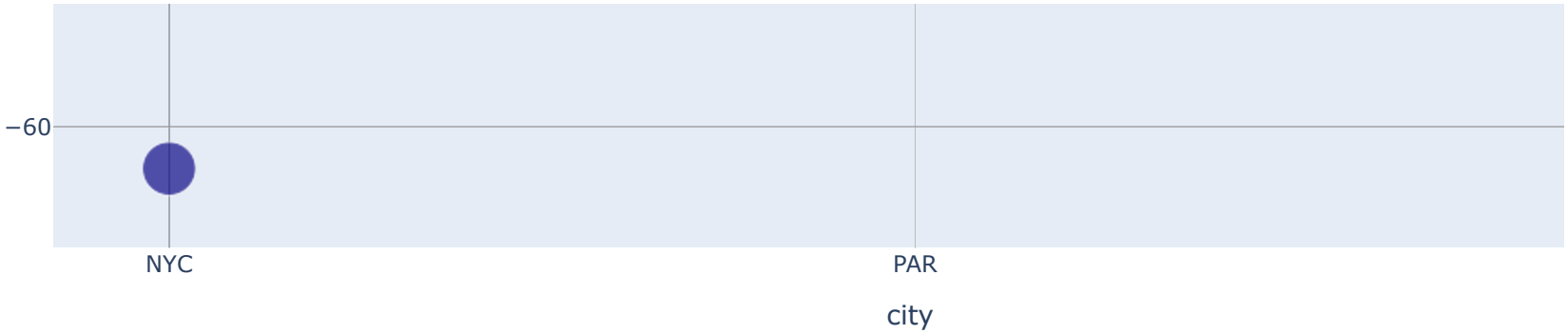
In [59]:
```
#Select the data we will need
HousingDF = pd.DataFrame(df_comp_det[['city','beds','price','minimum_nights','maximum_nights','room_type', 'latitude', 'l
HousingDF = HousingDF.loc[(HousingDF['beds']==2)]
HousingDF = HousingDF.loc[(HousingDF['room_type']=='Private room')]
```

In [60]:
```
#Average double room hotel rate (USD)
NYChotelavg=221
PARhotelavg=193
SYDhotelavg=156

#Computing and storing deal ratings for each city
for i in HousingDF['city']:
    if i=='NYC':
        HousingDF['Deal_Score']=(NYChotelavg-HousingDF['price'])/NYChotelavg
    elif i=='SYD':
```

```
        HousingDF['Deal_Score']= (SYDhotelavg - HousingDF['price'])/SYDhotelavg
    else:
        HousingDF['Deal_Score']= (PARhotelavg - HousingDF['price'])/PARhotelavg
```

In [61]:
```
#Function to plot deal ratings based on city
def GlobalDeals(dataset, variable1, variable2):
  fig = px.scatter(dataset, x='city', y='Deal_Score', color=variable2, size=variable1,
                hover_data = [variable1, variable2],
                width = 1100, height = 700,
                title = 'Deal Ratings for 2 Bed Airbnb Private Rooms in NYC, Sydney, and Paris (Compared to Median Doubl
  fig.show()
```

In [62]:
```
#Plotting our data
GlobalDeals(HousingDF,'price','Deal_Score')
```



Our results here are interesting. Firstly, outside of a few outliers, there do not seem to be major differences in the distributions of deal ratings between cities (With the exception of Sydney having more extreme negative deals; possibly due to the availability of beach homes).

Zooming into the graph reveals that a large number of Airbnb rentals are actually more expensive than the average hotel rate for their city. This is pretty novel considering a hotel will usually have more amenities than a simple private room for Airbnb.

There are also clearly some deals on Airbnb listings in these cities, even if they are outnumbered by the bad; how do we go about finding these deals? Let's try and find out by mapping our positive deals and seeing if we can notice any trends.

In [63]:
```
#Removing negative deals and inputting a mapbox access token
HousingDF = HousingDF.loc[(HousingDF['Deal_Score']>0)]
px.set_mapbox_access_token('pk.eyJ1IjoiY3NhcmtpczEzIiwiYSI6ImNrb3N6dTR0azA2bHYyd3J4dnoxMDg0eHkifQ.SE6flSGfY7grvHqakXZLKw'
```

In [64]:
```
#Plotting a geomap of deal ratings for each city
fig = px.scatter_mapbox(HousingDF, lat='latitude', lon="longitude",color="Deal_Score", zoom=1,
                        title = 'Deal Rating Geomap (Interactive! Scroll to Zoom)')
fig.show()
```

Deal Rating Geomap (Interactive! Scroll to Zoom)

Explanations:
Zooming into any of our three cities reveals a consistent trend: better deals from the average are usually found farther away from central city locations.

For example, in NYC, we see much higher deal scores in the other boroughs when compared to Manhattan, in Paris, we see more deals farther away from the Seine River, and in Sydney we see better deals as we move farther away from the coast line.

While it could be argued that hotel rates in these areas may also be generally lower, reducing the significancce of the deal. the fact still stands that the best Airbnb deals for these cities are going to be located in non-traditional vacation areas. One explaination for this may be that real estate is cheaper in these areas, paving the way for more amateur hosts that are willing to bargain more on their prices.

Ultimately there are deals over hotels on Airbnb in these major cities, but due diligence must be taken to make sure that you are actually getting a deal.

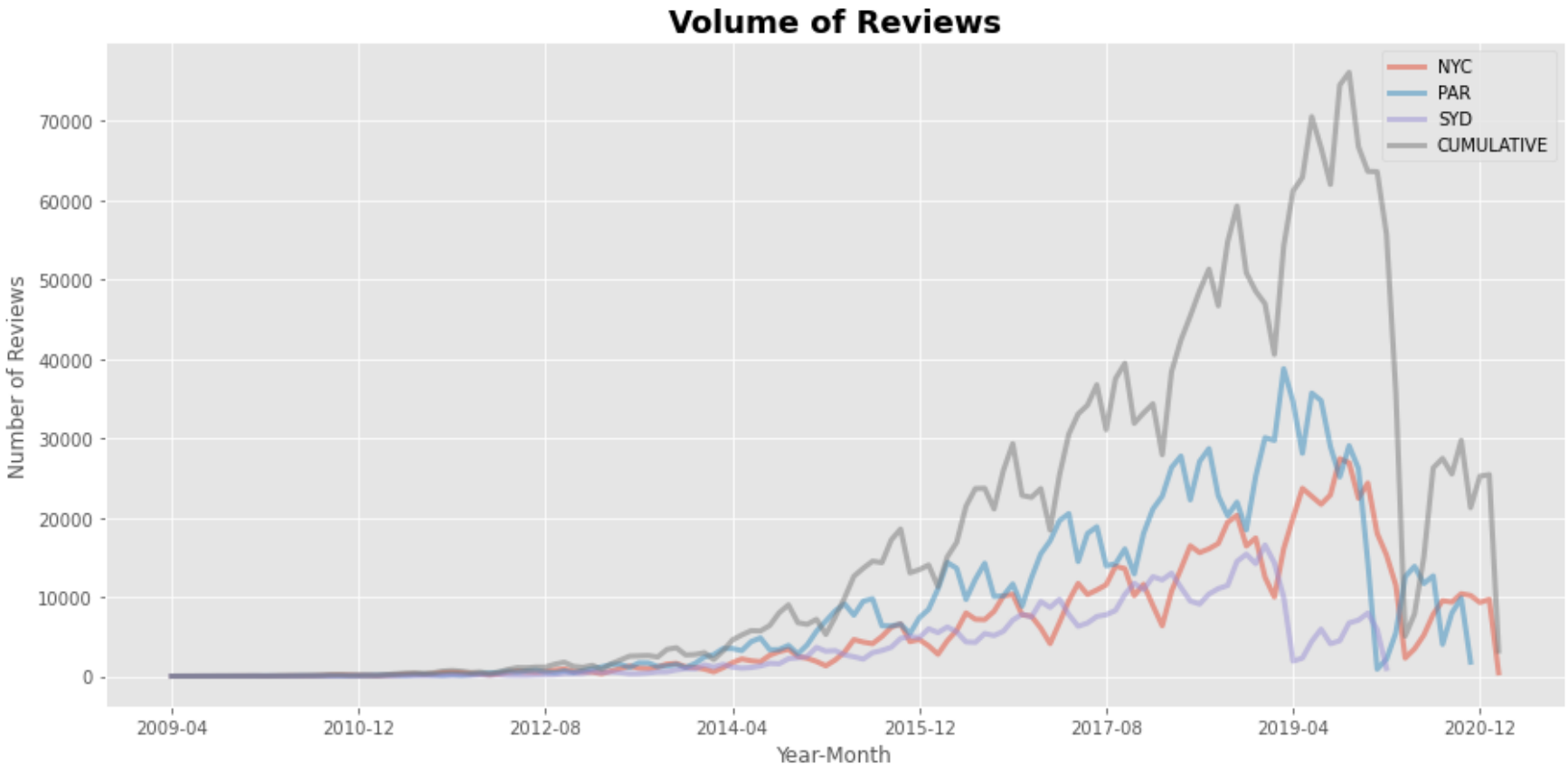
3. Airbnb Listings in Relation to COVID-19 Lockdown Policies

When looking at available data for Airbnb, we were unable to find detailed information covering prices going back further than April 2020. The data is unavailable to those who don't make a special request and pay to access. So in order to observe Airbnb's performance in Paris, Sydney, and New York City, we've opted to look at the volume of comments. While not every renter will leave a comment, there is a direct relationship between the reviews left and the number of people using Airbnb in the given cities.

In [65]:
```
#change format of data
df_comp_rev['date'] = df_comp_rev['date'].astype('str')
for i in [df_nyc_rev, df_par_rev, df_syd_rev, df_comp_rev]:
    i['Year-Month'] = i['date'].str.slice(0,7)
```

In [66]:
```
#plot volume of reviews over time
fig,ax = plt.subplots()
for i in [df_nyc_rev, df_par_rev, df_syd_rev, df_comp_rev]:
    pd.DataFrame(i['Year-Month'].value_counts()).reset_index().sort_values(
      'index').plot.line(x='index', y='Year-Month', figsize=(15,7), ax=ax, lw=3, alpha = .5)
ax.set_title ('Volume of Reviews', size=18, fontweight='bold')
ax.set_xlabel('Year-Month', size = 12)
ax.set_ylabel('Number of Reviews', size = 12)
ax.legend(['NYC','PAR','SYD','CUMULATIVE'])
```

Out[66]:
```
<matplotlib.legend.Legend at 0x134c5665d60>
```



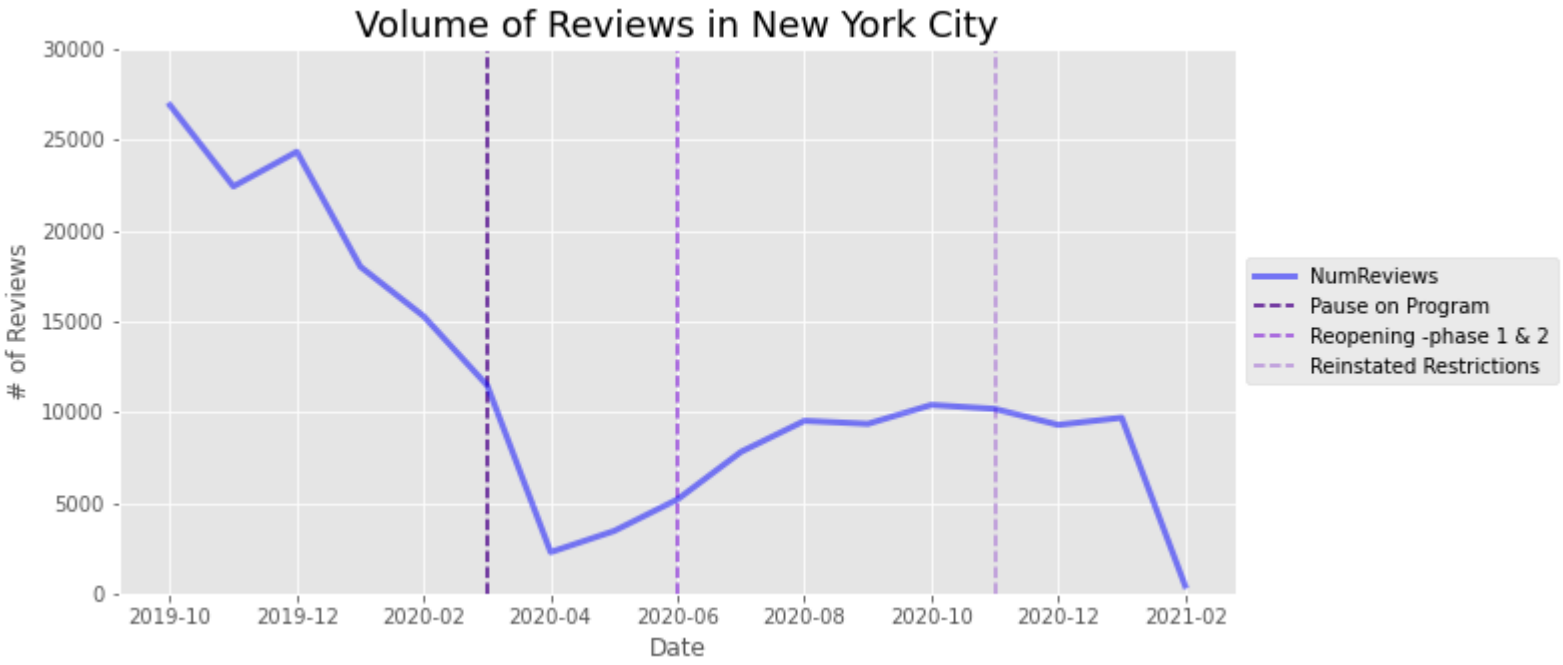
While there is a direct relationship between number of reviews and the number of consumers on Airbnb in the forementioned 3 cities, examining the change in number of reviews when studying the increases and decreases in Airbnb's customers would be more accurate. Consequently, we chose to look at the changes in number of reviews for each city throughout the pandemic in relation to the governments' policy changes.

In [67]:
```
#function to graph volume of reivews
def graph_covidreviews(dataframe, ylim, color, line1, line2, line3, text1, text2, text3, city):
  #create new dataframe to count the number of reviews per month
  df_Count = dataframe['Year-Month'].value_counts().to_frame()
  df_Count = df_Count.rename(columns = {"Year-Month" : "NumReviews"})
  df_Count = df_Count.sort_index()

  #data frame for number of reviews from 07-2019 onwards
  df_Covid = df_Count.truncate(before = '2019-10')

  #plotting
  fig,ax = plt.subplots()
```

```
    df_Covid.plot(figsize=(10,5), ax=ax, lw=3, color = color, alpha = .5)
    ax.vlines(line1,0, ylim, colors = '#4d038a', linestyles = 'dashed', label = text1)
    ax.vlines(line2,0, ylim, colors = '#9644db', linestyles = 'dashed', label = text2)
    ax.vlines(line3,0, ylim, colors = '#b68dd9', linestyles = 'dashed', label = text3)
    ax.legend(loc = 'center left', bbox_to_anchor = (1,0.5))
    ax.set_ylim([0,ylim])

    title = 'Volume of Reviews in ' + city
    ax.set_title(title ,size=18)
    ax.set_ylabel('# of Reviews', size =12)
    ax.set_xlabel('Date', size =12)
```

New York City

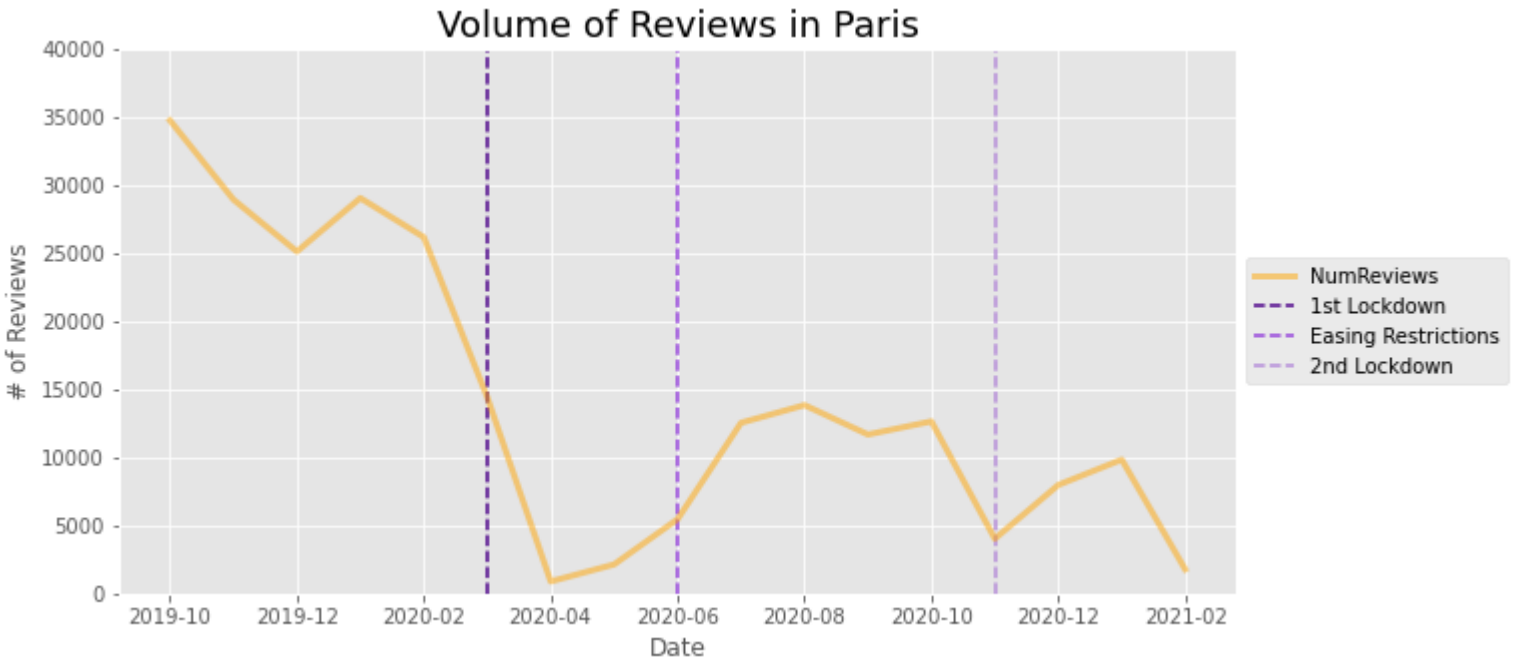
In [69]:
```
graph_covidreviews(df_nyc_rev, 30000, 'blue', 5, 8, 13, 'Pause on Program', 'Reopening -phase 1 & 2', 'Reinstated Restric
```



1. March 2020 - Pause on Program begins requiring all non-essential workers to stay home
2. June 2020 - NYC begins reopening (phase 1 and phase 2)
3. November 2020 - new restrictions are reinstated (curfews, indoor dining restrictions, limited gatherings in private homes)

Paris

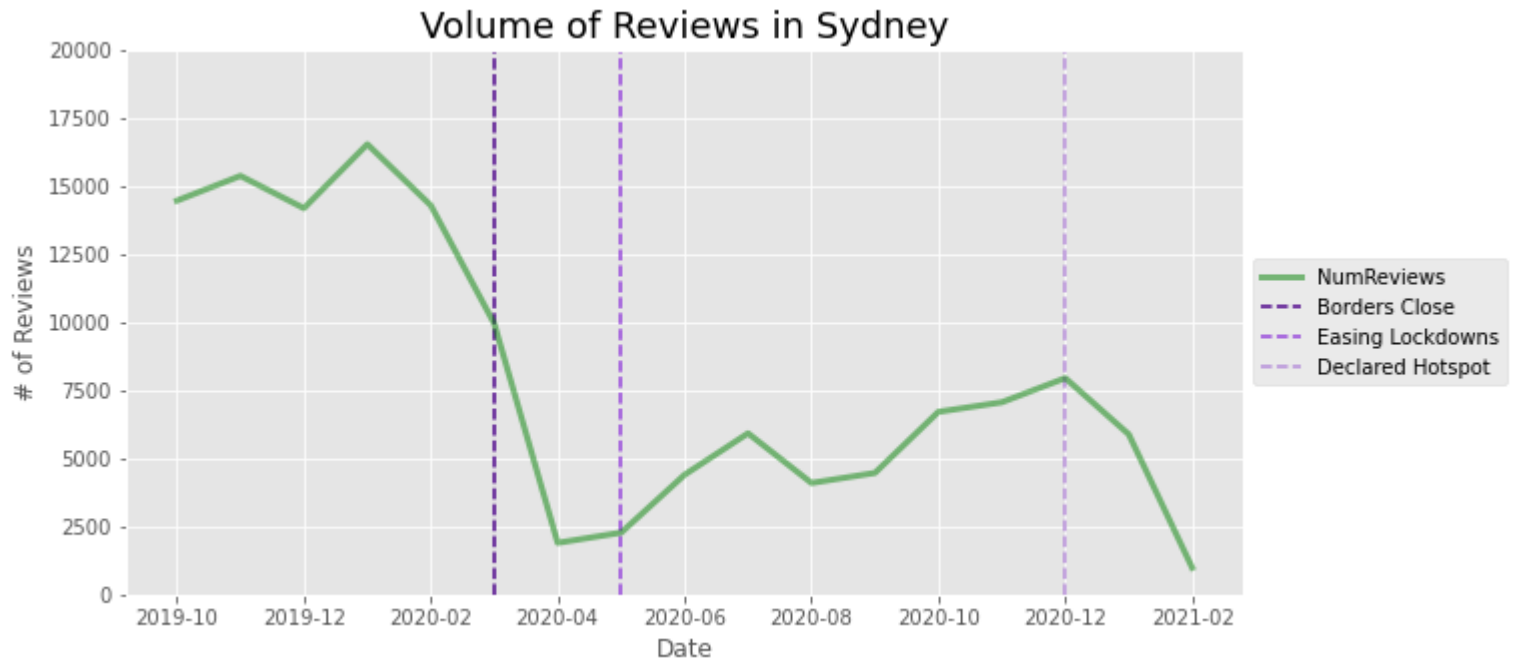
In [68]:
```
graph_covidreviews(df_par_rev, 40000, 'orange', 5, 8, 13, '1st Lockdown', 'Easing Restrictions', '2nd Lockdown', 'Paris')
```



1. March 2020 - strict lockdown imposed (no non-essential movement allowed and need travel pass to move)
2. June 2020 - easing of restrictions; travel to other EU countries with open borders is allowed
3. November 2020 - 2nd strict lockdown with additional restrictions in Paris

Sydney

In [70]:
```
graph_covidreviews(df_syd_rev, 20000, 'green', 5, 7, 14, 'Borders Close', 'Easing Lockdowns', 'Declared Hotspot', 'Sydney
```



1. March - Borders close to non-residents and residents are required to quarantine
2. May- Easing of lockdowns accross Australia
3. December - Northern part of Sydney declared a hotspot and restrictions increase

Conclusion

Governments in all 3 cities implemented policies of various degrees in March, following the spread of Covid-19 outside of Asia. Paris, which had the strictest lockdown policy of the three cities, restricted the movement of individuals coming into the city and also within the city. Consequently, the graphs show that Paris saw the greatest drop of guests at Airbnbs. On the other hand, New York City, which did not close its borders and did not legally restrict travel in the city, did not see as large of a percent decrease. Around June for Paris and New York City and May for Sydney, Covid-19 number seemed to be improving and local governments opted to ease quarantine and social distancing policies. All three cities saw an increase in guests but no where near pre-Covid levels. Torwards the end of the the end of 2020, Covid-19 cases where on a rise again accross the globe. In response to to fears of a 2nd wave, the local governments of the 3 cities re-implemented restrictions. Interestingly, the usage of Airbnb and travel only decreased in Sydney and Paris: New York City was seemingly unaffected.This could be due to a variety of reasons, such as a differece in attitudes torward Covid-19 or conflicting information from the state and federal government.

Note: While we believe that observing the reviews left per month reflects the number of guests using Airbnb, it is important to note that review response can be delayed.