Group assignment, Introduction to Data Science
Master's Programme in Data Science

Airbnb Price Prediction System

Muhammad Rafay, Yu Wang, Ari Kauppi

October 28, 2024

Supervisor(s): Teemu Roos

Examiner(s):

University of Helsinki
Faculty of Science

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki

Contents

1. Introduction

The **Airbnb Price Prediction System**[†] was developed to equip both hosts and guests with intelligent, data-driven pricing insights that support optimal profitability and informed decision-making. By leveraging key property and location attributes, the system provides an essential tool for Airbnb **users** to identify competitive booking rates and for **hosts** to establish the ideal listing prices. This approach addresses a key market gap by enabling users to secure the best available prices based on specific criteria while empowering hosts to maximize their earnings.

However, pricing in the short-term rental market is shaped by fluctuating demand, seasonal trends, and diverse property attributes, making it challenging to establish optimal rates. To tackle these complexities, this system offers data-driven recommendations that:

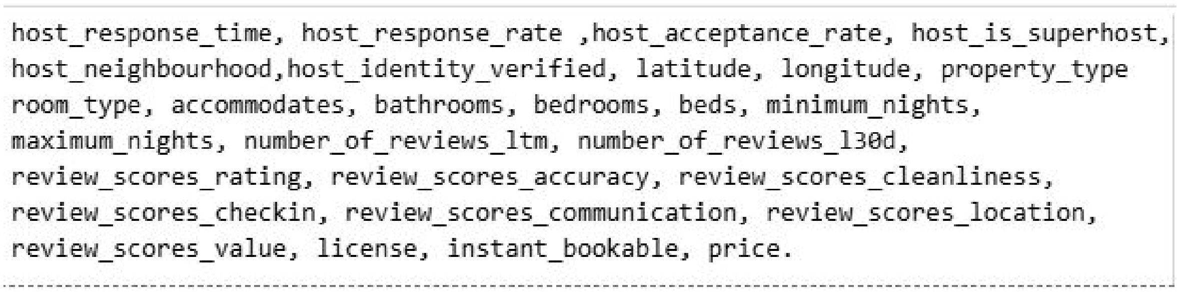
1. **Enable Hosts** to set competitive, well-informed prices that maximize listing appeal and profitability.
2. **Empower Users** to find bookings that align with their budget and preferences.

To effectively deliver these insights, the scope of this system encompasses the **complete development pipeline**, from initial **data collection** and **preprocessing** through to **model selection**, **data visualization**, and **deployment**. Each stage has been meticulously structured to ensure that the final product provides accurate and accessible pricing guidance for Airbnb stakeholders.

[†]Model development Github

2. Data Collection

For our project, we utilized data sourced from Inside Airbnb, which provides datasets in CSV format accessible at Inside Airbnb. Our focus was specifically on the **Boston, Massachusetts** area, allowing us to analyze localized market dynamics. The dataset consists of over 75 variables; however, after thorough research and consideration, we selected a refined set of variables essential for our analysis. The chosen variables are shown in figure 2.1.



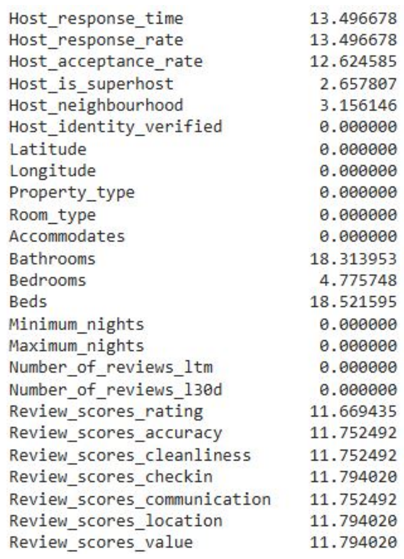
```
host_response_time, host_response_rate ,host_acceptance_rate, host_is_superhost,
host_neighbourhood,host_identity_verified, latitude, longitude, property_type
room_type, accommodates, bathrooms, bedrooms, beds, minimum_nights,
maximum_nights, number_of_reviews_ltm, number_of_reviews_l30d,
review_scores_rating, review_scores_accuracy, review_scores_cleanliness,
review_scores_checkin, review_scores_communication, review_scores_location,
review_scores_value, license, instant_bookable, price.
```

Figure 2.1: Chosen variables

During the data collection process, we encountered various challenges, particularly in identifying relevant data that aligned with our requirements. After exploring different sources, we focused on the Inside Airbnb platform, where we identified key features crucial for determining optimal pricing. Through thorough research and analysis, we matched our input and target variables to develop a robust price prediction algorithm. From this dataset, we selected a subset of variables that demonstrated a strong correlation with the pricing data, ensuring the effectiveness of our predictive model.

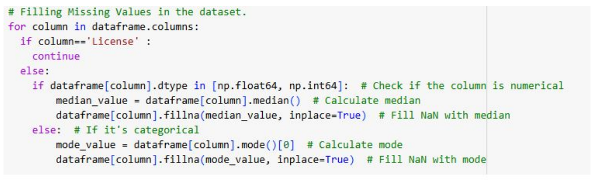
3. Data Preparation

Our data preparation process involved two key components: data preprocessing and feature engineering. During data preprocessing, we first standardized column names and values, as many entries were inconsistently formatted. The dataset, consisting of 2,400 rows and 28 columns, had missing values ranging from 10 to 26 entries across various columns. To handle these missing values, we employed median and mode imputation techniques to preserve data integrity, avoiding mean imputation due to the presence of outliers, which could skew results. The approach used to address missing values is illustrated in Figure 3.1.



```
Host_response_time              13.496678
Host_response_rate              13.496678
Host_acceptance_rate            12.624585
Host_is_superhost                2.657807
Host_neighbourhood               3.156146
Host_identity_verified           0.000000
Latitude                         0.000000
Longitude                        0.000000
Property_type                    0.000000
Room_type                        0.000000
Accommodates                     0.000000
Bathrooms                       18.313953
Bedrooms                         4.775748
Beds                            18.521595
Minimum_nights                   0.000000
Maximum_nights                   0.000000
Number_of_reviews_ltm            0.000000
Number_of_reviews_l30d           0.000000
Review_scores_rating            11.669435
Review_scores_accuracy          11.752492
Review_scores_cleanliness       11.752492
Review_scores_checkin           11.794020
Review_scores_communication     11.752492
Review_scores_location          11.794020
Review_scores_value             11.794020
```

(a) Null values

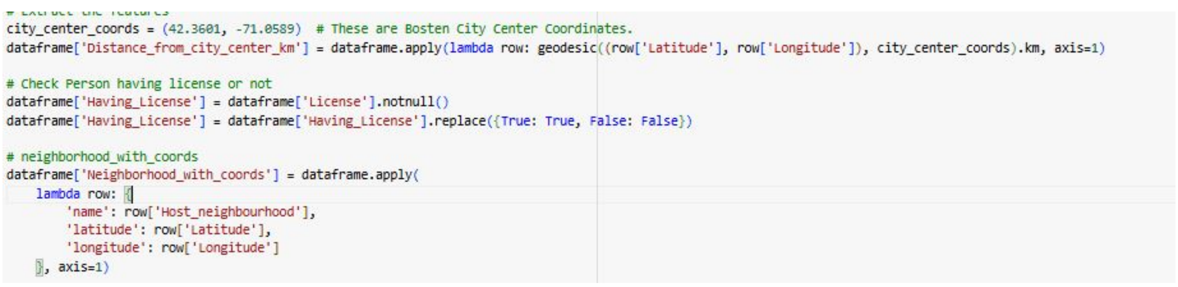


```
# Filling Missing Values in the dataset.
for column in dataframe.columns:
  if column=='License' :
    continue
  else:
    if dataframe[column].dtype in [np.float64, np.int64]:  # Check if the column is numerical
        median_value = dataframe[column].median()  # Calculate median
        dataframe[column].fillna(median_value, inplace=True)  # Fill NaN with median
    else:  # If it's categorical
        mode_value = dataframe[column].mode()[0]  # Calculate mode
        dataframe[column].fillna(mode_value, inplace=True)  # Fill NaN with mode
```

(b) Filling missing values

Figure 3.1: Null values found and imputating values

In addition to data cleaning, we conducted feature engineering to enrich the dataset with new variables. For example, we derived features such as `Distance_from_City_Center_km`, `Having_License`, and `Neighborhood_with_Coords`. Afterward, we applied encoding techniques and standard scaling to the data to normalize the feature distributions and mitigate the impact of outliers. Details on the feature extraction process can be seen in Figure 3.2:

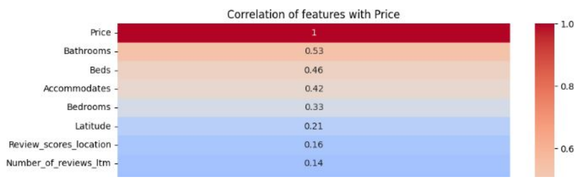


```
city_center_coords = (42.3601, -71.0589)  # These are Bosten City Center Coordinates.
dataframe['Distance_from_city_center_km'] = dataframe.apply(lambda row: geodesic((row['Latitude'], row['Longitude']), city_center_coords).km, axis=1)

# Check Person having license or not
dataframe['Having_License'] = dataframe['License'].notnull()
dataframe['Having_License'] = dataframe['Having_License'].replace({True: True, False: False})

# neighborhood_with_coords
dataframe['Neighborhood_with_coords'] = dataframe.apply(
    lambda row: {
        'name': row['Host_neighbourhood'],
        'latitude': row['Latitude'],
        'longitude': row['Longitude']
    }, axis=1)
```

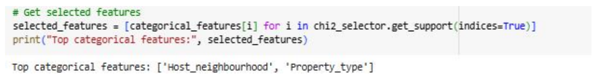
Figure 3.2: Feature extraction

To better understand the relationships in our data, we performed exploratory data analysis (EDA). The correlation matrix (Figure 3.3a) revealed that features such as the number of bathrooms and beds have a moderate correlation with price, while variables like the number of reviews and review scores showed weaker relationships. Additionally, scatter plots were used to explore patterns between host-related attributes (e.g., response rate, acceptance rate, and superhost status) and price. These plots indicate that higher host response and acceptance rates tend to be associated with higher prices (Figure 3.4).

Furthermore, we examined the relationship between occupancy rate and price (Figure 4.1b), which showed that while there is a wide distribution of prices across different occupancy levels, some higher prices are observed when the occupancy rate is close to 100%. Visualization techniques such as these helped us identify trends, spot outliers, and understand the impact of various features on price.



(a) Correlation matrix



```
# Get selected features
selected_features = [categorical_features[i] for i in chi2_selector.get_support(indices=True)]
print("Top categorical features:", selected_features)
```

```
Top categorical features: ['Host_neighbourhood', 'Property_type']
```

(b) Chi-square test

Figure 3.3: Statistical tests

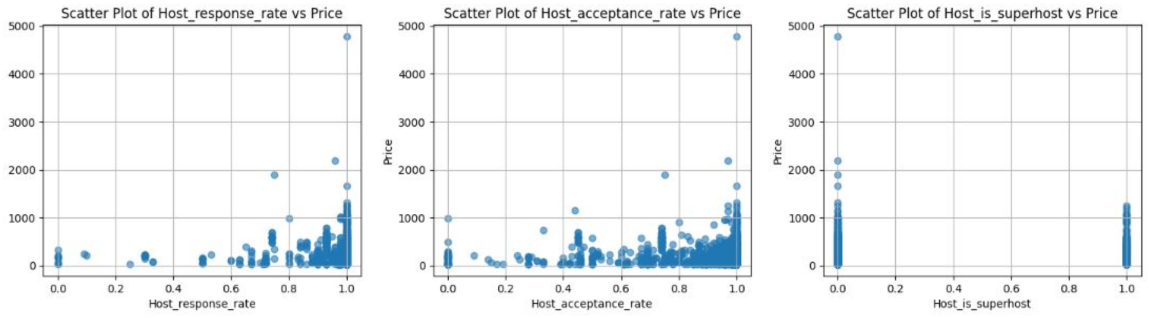


Figure 3.4: Scatter plots of host response rate, acceptance rate and is_superhost vs Price

During the exploratory data analysis (EDA) phase, we generated a correlation matrix and performed a chi-square test to examine the relationships between variables (see figure 3.3). Visualization techniques, including box plots and scatter plots, helped us identify outliers and further analyze correlations (more graphs can be found in appendices). This thorough approach improved our understanding of the data and guided the selection of an effective algorithm for our price prediction model.

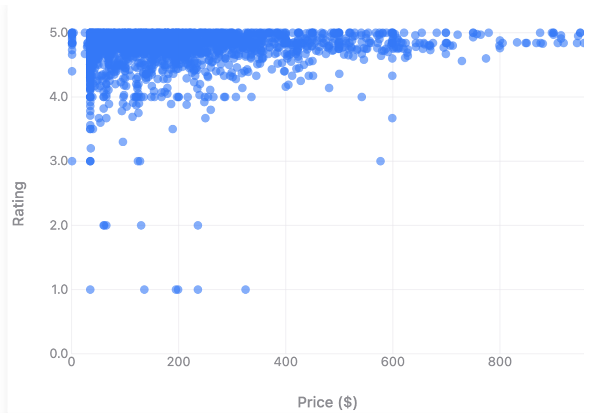
4. Data Visualization and User Interface

To create a user-friendly platform for visualizing Airbnb data, we used Next.js for server-side rendering and site structure, enhancing performance and SEO, and React.js for building modular, reusable UI components. For data visualization, we leveraged D3.js to develop interactive charts, such as pie charts showing host response times, scatter plots of price vs. rating and occupancy rate, and a correlation heatmap to reveal relationships between review factors. These visualizations provided insights into market dynamics and user preferences. We incorporated the Google Maps API for an interactive map, using the @vis.gl/react-google-maps library to integrate it directly into React. This allowed users to explore listing locations, view details, and interact with map markers. Additionally, we used Folium to plot listings by neighborhood, enabling users to browse listings, view prices, and see the number of listings per neighborhood. These are shown below in Figure 4.3.

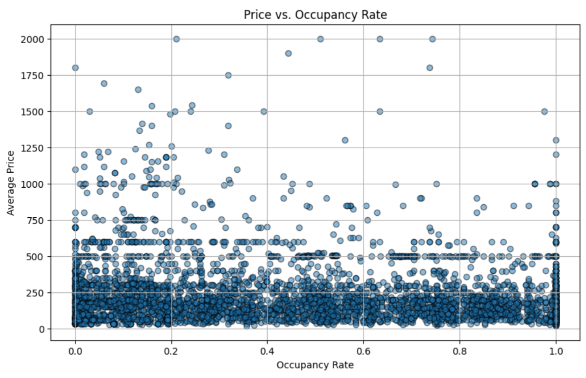
We incorporated into the user interface visualizations such as figure 4.2a depicting host response times, which offers a clear view of how promptly hosts tend to reply to guest inquiries, providing an understanding of the market's service quality in terms of responsiveness. Scatter plots (see figure 4.1) illustrated key relationships between numerical variables. A price versus rating plot suggested that higher-rated listings might command higher prices, while a price versus occupancy rate plot revealed a clustering of budget-friendly listings. Notably, higher prices did not always align with higher occupancy, implying that factors like location, amenities, or host reputation influence bookings. To further analyze review scores, we implemented a correlation heatmap (see figure 4.2b). High correlations, like between cleanliness and accuracy, indicate areas where focused improvements could enhance guest satisfaction, offering actionable insights for hosts.

Throughout the development process, several challenges arose that required thoughtful solutions and optimizations. **Data formatting** was a significant challenge, as ensuring that the data was consistently structured for compatibility with D3.js involved extensive preprocessing. **Performance optimization** became essential, especially when rendering large datasets; techniques such as simplifying visual elements and reducing the number of data points displayed were used to maintain responsiveness. **Enhancing interactivity** was another focus, with plans to introduce features like tooltips, zooming, and filtering in future iterations to allow users to explore the visualizations in greater depth. Lastly, **optimizing map loading times** and securely managing API keys were critical for ensuring a smooth user experience and maintaining data security.

Similarly, the scatter plot of **price versus occupancy rate** illustrated that while there is a higher concentration of budget-friendly listings, the frequency with which a property is booked does not necessarily increase with price. This finding points to the complexity of the market, where factors other than price likely influence booking decisions. Lastly, the **correlation heat map** provided clarity on which review factors tend to be rated similarly by guests. Understanding these associations can help hosts prioritize improvements in areas that could potentially boost multiple aspects of guest satisfaction.

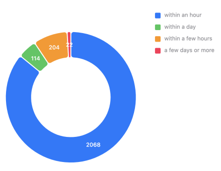


(a) Price vs. Rating

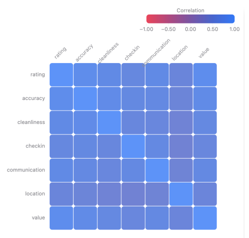


(b) Price vs. Occupancy

Figure 4.1: Price vs. Rating and Price vs. Occupancy

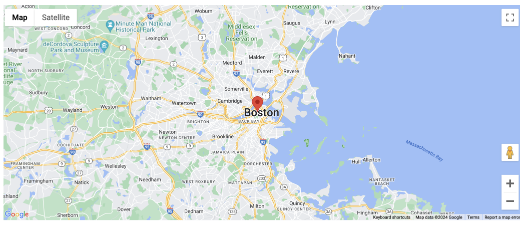


(a) Host response rates

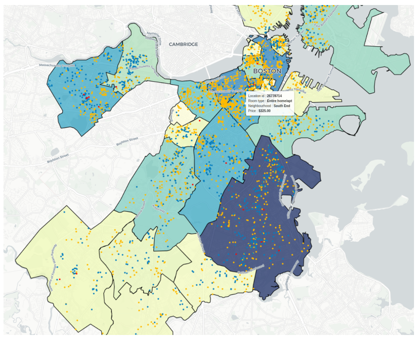


(b) Correlation heat map

Figure 4.2: Price vs. Rating and Price vs. Occupancy



(a) Google maps

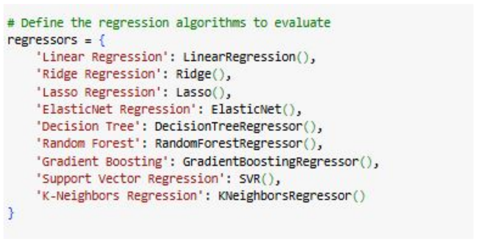


(b) Folium map

Figure 4.3: Map API and listings mapped

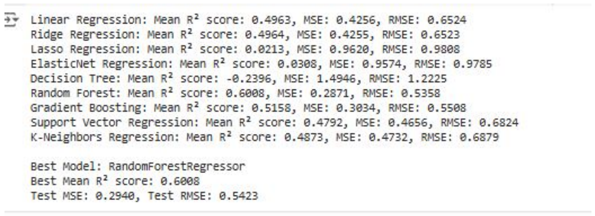
5. Data Modeling

In the data modeling phase, we tested various machine learning algorithms to determine the most effective model for Airbnb price prediction. The algorithms used included Linear Regression, Ridge Regression, Lasso Regression, Elastic Net, Decision Tree, Random Forest, Gradient Boosting, Support Vector Regression, and K-Neighbors Regression (Figure 5.1a).



```python
# Define the regression algorithms to evaluate
regressors = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(),
    'Lasso Regression': Lasso(),
    'ElasticNet Regression': ElasticNet(),
    'Decision Tree': DecisionTreeRegressor(),
    'Random Forest': RandomForestRegressor(),
    'Gradient Boosting': GradientBoostingRegressor(),
    'Support Vector Regression': SVR(),
    'K-Neighbors Regression': KNeighborsRegressor()
}
```

(a) Models used



```
Linear Regression: Mean R² score: 0.4963, MSE: 0.4256, RMSE: 0.6524
Ridge Regression: Mean R² score: 0.4964, MSE: 0.4255, RMSE: 0.6523
Lasso Regression: Mean R² score: 0.0213, MSE: 0.9620, RMSE: 0.9808
ElasticNet Regression: Mean R² score: 0.0308, MSE: 0.9574, RMSE: 0.9785
Decision Tree: Mean R² score: -0.2396, MSE: 1.4946, RMSE: 1.2225
Random Forest: Mean R² score: 0.6008, MSE: 0.2871, RMSE: 0.5358
Gradient Boosting: Mean R² score: 0.5158, MSE: 0.3034, RMSE: 0.5508
Support Vector Regression: Mean R² score: 0.4792, MSE: 0.4656, RMSE: 0.6824
K-Neighbors Regression: Mean R² score: 0.4873, MSE: 0.4732, RMSE: 0.6879

Best Model: RandomForestRegressor
Best Mean R² score: 0.6008
Test MSE: 0.2940, Test RMSE: 0.5423
```

(b) Modeling results

Figure 5.1: Models used and modeling results

To evaluate and compare model performance, we employed cross-validation with the R^2 metric to measure each model's predictive accuracy. Models were implemented using the Scikit-learn library, and the one with the highest R^2 score was selected as the best performer. Among the models tested, the Random Forest Regressor demonstrated the best R^2 value, showcasing strong performance even in the presence of outliers. This robustness made it a promising choice for accurately predicting prices across a diverse range of listings.

In addition to R^2, we calculated Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) to further validate the effectiveness of each model. These metrics provided insights into the average prediction error and helped us better understand the model's precision. The Random Forest Regressor consistently outperformed other models across all metrics, achieving an MSE of 0.29 and an RMSE of 0.54, indicating solid predictive reliability.

The primary challenge was achieving optimal accuracy; however, we managed to attain a R^2 score of 60 %, indicating a reasonably good fit. For the random forest regression, our model produced an MSE of 0.29 and an RMSE of 0.54, reflecting solid predictive performance. Further feature engineering and fine-tuning of hyperparameters could have helped enhance the model's accuracy and reliability. The results of our different models are presented in Figure 5.1b.

6. Deployment

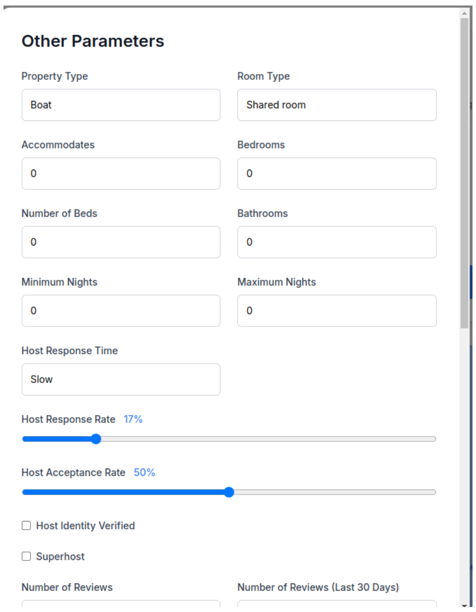
We deployed the front end on Vercel and the backend on Render to leverage their specific strengths. Vercel was ideal for the frontend due to its fast deployment and GitHub integration, while Render provided a flexible environment for our Flask-based backend, supporting a variety of programming languages and frameworks.

For the frontend, we prepared and tested the React application locally, connected the GitHub repository through Vercel, and enabled Continuous Integration/Continuous Deployment (CI/CD) to automate updates with each push. Testing on Vercel ensured that all components functioned correctly, including a user input form that enables the "Calculate Price" button after all fields are completed.

The backend deployment on Render followed a similar process. We configured environment settings, set up build and start commands (gunicorn app:app), and verified all endpoints through automated CI/CD testing. Render's monitoring tools facilitated efficient debugging and performance adjustments as needed.

Challenges included dependency management, securing environment variables, and optimizing performance. Both platforms' logging features proved invaluable for resolving issues quickly. This deployment setup enabled a smooth, automated workflow, ensuring that the latest code changes were consistently and securely deployed.

The current front-end implementation requires users to fill in all text fields before the "Calculate Price" button is enabled. This approach was chosen to simplify development, given that none of us have extensive front-end experience. Once the form is completed, there may be a brief delay before the "Calculate Price" button becomes active. This delay occurs because the backend server, which is suspended when not in use, takes some time to wake up. After approximately a minute, the button should activate, allowing the user to proceed with price predictions. The data input form and the results are shown in Figure 6.1



Other Parameters

Property Type	Room Type
Boat	Shared room
Accommodates	Bedrooms
0	0
Number of Beds	Bathrooms
0	0
Minimum Nights	Maximum Nights
0	0
Host Response Time	
Slow	

Host Response Rate 17%

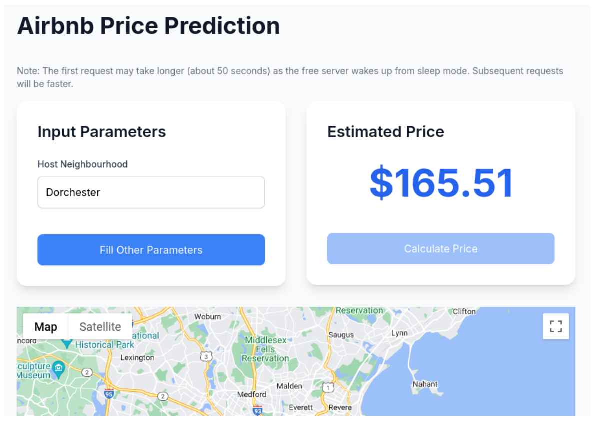
Host Acceptance Rate 50%

☐ Host Identity Verified

☐ Superhost

Number of Reviews | Number of Reviews (Last 30 Days)

(a) Inputs to the model



Airbnb Price Prediction

Note: The first request may take longer (about 50 seconds) as the free server wakes up from sleep mode. Subsequent requests will be faster.

Input Parameters

Host Neighbourhood: Dorchester

Fill Other Parameters

Estimated Price

$165.51

Calculate Price

(b) Model prediction

Figure 6.1: User interface

7. Project Outcomes

The project successfully achieved the deployment of a fully functional Airbnb price prediction platform, consisting of a React-based frontend and a Flask-based backend. Model development and training were performed using Google Colab, leveraging cloud-based computational resources for efficient experimentation with different machine learning algorithms. The frontend was enhanced with interactive visualizations and mapping tools, providing users with an intuitive interface to explore Airbnb listings and predict optimal pricing based on various property attributes.

The deployment was carried out using Vercel for the frontend and Render for the backend, with Continuous Integration and Continuous Deployment (CI/CD) processes ensuring seamless updates. The integration of interactive maps and visual elements, such as scatter plots and heatmaps, improved the accessibility and understanding of data trends, making the platform not just a predictive tool but also an insightful resource for market analysis.

Looking forward, there is significant potential to further improve model accuracy by incorporating additional features or refining the data preprocessing steps. Such enhancements could push predictive reliability to around 80-85%, making the model robust enough for real-time pricing suggestions. This would provide greater value to users and hosts alike, allowing for more informed decision-making and optimized profitability in the Airbnb market. Further incorporating the seasonality of demand and occupancy rates could help create an even better dynamic pricing model.

Bibliography

Appendix A. Feature extraction

```
# Extract the features
# These are Boston City Center Coordinates.
city_center_coords = (42.3601, -71.0589)
dataframe['Distance_from_city_center_km'] = dataframe.apply(
    lambda row: geodesic(
        (row['Latitude'], row['Longitude']), city_center_coords).km, axis=1
    )

# Check Person having license or not
dataframe['Having_License'] = dataframe['License'].notnull()
dataframe['Having_License'] = dataframe['Having_License'].replace(
    {True: True, False: False}
  )

# neighborhood_with_coords
dataframe['Neighborhood_with_coords'] = dataframe.apply(
    lambda row: {
        'name': row['Host_neighbourhood'],
        'latitude': row['Latitude'],
        'longitude': row['Longitude']
    }, axis=1)

neighborhood_coords = {
    row['Host_neighbourhood']: {
        'latitude': row['Latitude'],
        'longitude': row['Longitude']
    } for index, row in dataframe.iterrows()
}
```

Appendix B. Filling missing values

```
# Filling Missing Values in the dataset.
for column in dataframe.columns:
  if column=='License' :
    continue
  else:
  # Check if the column is numerical
    if dataframe[column].dtype in [np.float64, np.int64]:
        # Calculate median
        median_value = dataframe[column].median()
        # Fill NaN with median
        dataframe[column].fillna(median_value, inplace=True)
    # If it's categorical
    else:
        # Calculate mode
        mode_value = dataframe[column].mode()[0]
        # Fill NaN with mode
        dataframe[column].fillna(mode_value, inplace=True)
```

Appendix C. Scatter plots of various features vs. price

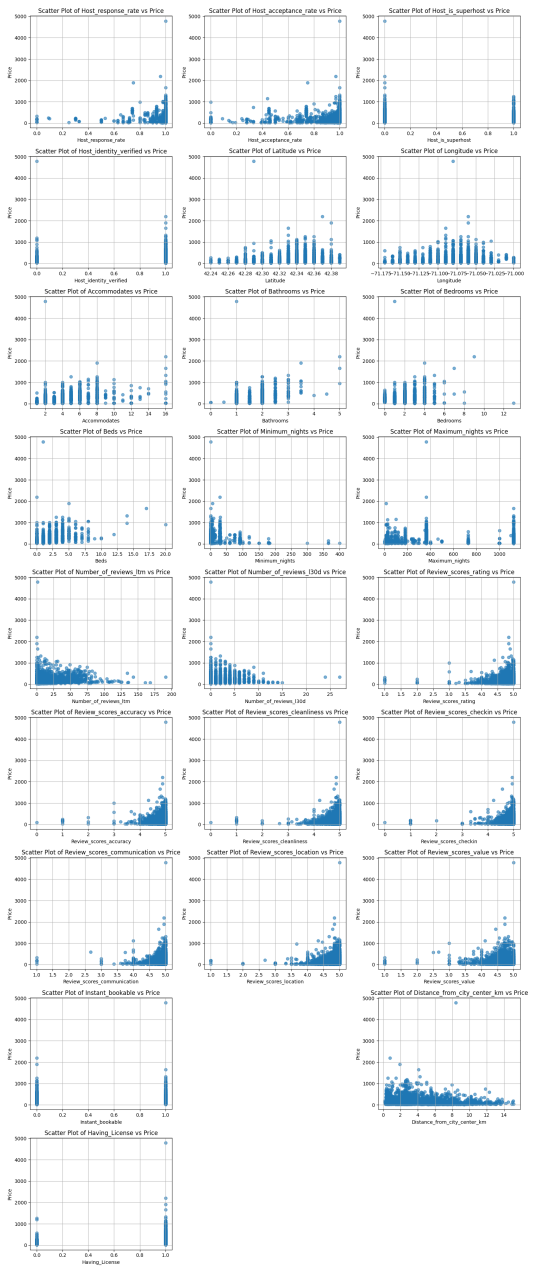
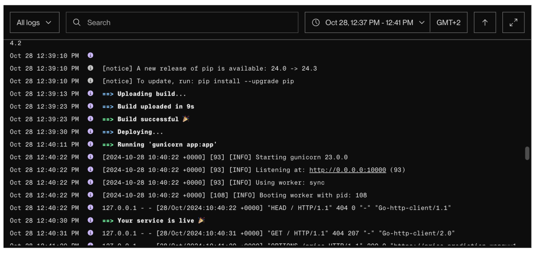


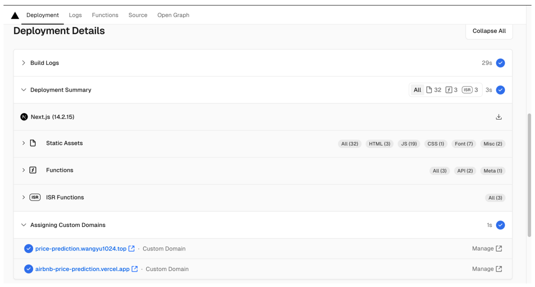
Figure C.1: Scatter plots of various features vs. price

Appendix D. Deployment testing and details figures



```
4.2
Oct 28 12:39:10 PM
Oct 28 12:39:10 PM  [notice] A new release of pip is available: 24.0 -> 24.3
Oct 28 12:39:10 PM  [notice] To update, run: pip install --upgrade pip
Oct 28 12:39:13 PM  ==> Uploading build...
Oct 28 12:39:23 PM  ==> Build uploaded in 9s
Oct 28 12:39:23 PM  ==> Build successful 🎉
Oct 28 12:39:30 PM  ==> Deploying...
Oct 28 12:40:11 PM  ==> Running 'gunicorn app:app'
Oct 28 12:40:22 PM  [2024-10-28 10:40:22 +0000] [93] [INFO] Starting gunicorn 23.0.0
Oct 28 12:40:22 PM  [2024-10-28 10:40:22 +0000] [93] [INFO] Listening at: http://0.0.0.0:10000 (93)
Oct 28 12:40:22 PM  [2024-10-28 10:40:22 +0000] [93] [INFO] Using worker: sync
Oct 28 12:40:22 PM  [2024-10-28 10:40:22 +0000] [108] [INFO] Booting worker with pid: 108
Oct 28 12:40:22 PM  127.0.0.1 - - [28/Oct/2024:10:40:22 +0000] "HEAD / HTTP/1.1" 404 0 "-" "Go-http-client/1.1"
Oct 28 12:40:30 PM  ==> Your service is live 🎉
Oct 28 12:40:31 PM  127.0.0.1 - - [28/Oct/2024:10:40:31 +0000] "GET / HTTP/1.1" 404 207 "-" "Go-http-client/2.0"
```

Figure D.1: Testing deployment



Deployment Details

Build Logs 29s

Deployment Summary All 32 3 3 3s

Next.js (14.2.15)

Static Assets All (32) HTML (3) JS (19) CSS (1) Font (7) Misc (2)

Functions All (3) API (2) Meta (1)

ISR Functions All (3)

Assigning Custom Domains 1s

price-prediction.wangyu1024.top · Custom Domain Manage

airbnb-price-prediction.vercel.app · Custom Domain Manage

Figure D.2: Deployment details