Data Mining

Preprocessing & Analysis of Data Scraped from Booking.com

Source Code

Collecting Dataset

Introduction

With the help of web scraping methods I successfully collected information regarding 889 hotels based in the Netherlands, Belgium, and Germany. Out of these ~900 hotels we extracted 54,914 rows of data meaning around ~18,300 rows for each country.
The dataset was collected from Booking.com which is one of the largest online travel agencies, coincidentally, headquartered in Amsterdam.

Collecting Hotels from Booking

Certain query parameters were necessary in order to get the data from Booking's servers. Those parameters are ones you would expect when booking an apartment, i.e. destination, dates you would like to stay, how many people are staying, and the number of rooms.
In this experiment we decided to:

- Query multiple destinations - Amsterdam - Netherlands, Antwerp - Belgium, and Dortmund - Germany.
- Searched for single night stays for dates between 01.02.2024 - 31.01.2025.
- Book for only 2 people (adults) and a single room

When sending this query with the help of BeautifulSoup the hotels returned are paginated - showing us items (hotels) with an offset of, at most, 25 per page. In order to get more hotels than this limitation allows us, multiple requests need to be sent to booking.com with an offset prop of the number of the items in all the previous pages.
The name of this script is create_hotels_csv.py and parameters can be set dynamically in order to achieve code reusability for different places and time frames. The resulting csv can be found in data/hotels_daily_final.csv.

Said csv contains these columns: name, location, price, date of stay (date), stars, distance from city center in km (distance_from_center), number of external (user) reviews (num_external_reviews), bookings classified rating based on user reviews (booking_user_ratings), average user ratings (avg_user_ratings), wether a taxi from the airport is available (airport_taxi), sustainable travel options ranked from 0 to 3+ (travel_sustainable_level), wether free cancelation is included in the deal (has_free_cancelation).

NOTE: Although attractions were not used in the final preprocessing and analysis batch I found it was an interesting obstacle for me to overcome so I decided to leave the approach documented below.

Collecting Attractions from Booking

Complications arose when trying to web scrape Booking's attractions data with the same approach as previously done with Booking's hotel data. When requesting a list of hotels from Booking we receive a page that is already fully rendered, i.e. server side rendering, but when requesting a list of attractions we get a page not fully populated with all the data we see when actually opening the website, this implies client side rendering. I also had a problem web scraping more items since the initial load only renders 70 attractions and adding offset as a query parameter was now not possible since this page uses infinite load and a click of a button ("See more") to show more attractions.

These obstacles were overcome with the help of Selenium which ''allows developers to programmatically control a web browser programmatically, meaning they can interact with websites as if they were human users''.

Selenium helped me:

1. Set an implicit wait of 5 seconds to allow time for the client to render all the elements;
2. Scroll to the bottom of the page because attractions were loaded dynamically;
3. Click on the "Show more" button which loads more attractions.

When querying for an attraction there are no options to select multiple visitors, so the prices listed are for one person only. Parameters needed were the destination and a range of dates, we used the same parameters as before.
Like before, we created a csv consisting of these attributes regarding attractions: name, location, description, stars - user rating out of 5, average user ratings, total number of user reviews, whether it has free cancelation, duration, if it is a bestseller (popular), and price.

Data Preprocessing

```
Missing Values:
name                          0
location                      0
price                         0
date                          0
stars                     22224
distance_from_center          0
num_external_reviews        528
booking_user_ratings        528
avg_user_ratings           1088
airport_taxi                  0
travel_sustainable_level  13789
has_free_cancelation          0
```

Cleaning

First and foremost, we need to discover and handle missing values in our dataset. Through a simple analysis we can conclude that:

1. the **price** should be converted into Euros and MKD prefix should be removed from the entries.
2. **stars** column contains Nan values (at least 1), so we should replace them with 0. This would only make sense since nan would represent no stars present, i.e. 0 stars.
3. **distance from center** is represented in string format. We can rename the column and represent the entries in int values, i.e. km from center.
4. **number of external reviews** contains a colon mark and 'reviews' suffix which should both be removed and the column represented as an int column
5. **booking user ratings** also contains dirty data. Nan, "Review score" and user ratings that were incorrectly parsed when scraping. These will be converted to numerical values with the help of a dictionary mapping.
6. **airport taxi** contains values that we are not interested in, remove these and mark them as Nan.
7. Map the **travel sustainable level** column into int values.
8. **avg_user_ratings** is a column of user ratings from (1-10), we can clean this up by applying the average user rating of all the hotels residing in the given area/town.
9. **airport_taxi** is a column defining wether the hotel/apartment has an airport taxi available, we should consider Nan the same as "No airport taxi".
10. **date** column contains a list of 2 dates - arriving and departure dates, we really only need the first date, i.e. the date of stay.
11. Encode **has_free_cancelation**

After cleaning the data, we can divide the dataset by City (‘town_location’ column) and end up with 4 DataFrames - df (for all of the cities), df_antwerp, df_dortmund, and df_amsterdam.

Now with the help of Papermill, which is a Python library that allows users to run and parametrize Jupyter notebooks in a way that is easy to maintain and reuse, we can perform Visualization, find Missing Values,

Outliers, and EDA on each of these DFs separately. By defining a single Jupyter notebook to be executed for all of the aforementioned data frames we cut on redundant code. The output of the executed notebooks is kept in new notebooks created in the output_results folder.

Visualization, Missing Values, Outliers, EDA

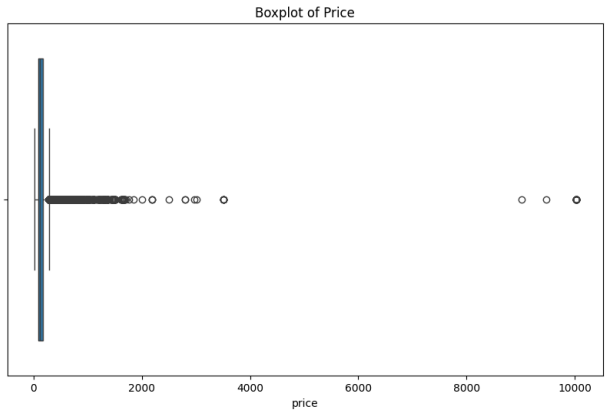
The mean price of a 1 night stay for 1 room for 2 adults for each town can be seen below represented in euros:

Amsterdam → 208.202478

Antwerp → 129.592413

Dortmund → 123.311975

We can conclude that Amsterdam is far more expensive than Antwerp and Dortmund, almost twice the price in fact.



Discovering and removing outliers can be done with boxplots.
By visualizing and calculating the exact values we can remove the upper and lower bounds of the price ranges.In fact, we can calculate that the upper bound is 286.59€, lower bound is 0€ since prices can not be negative and mean is: 154.24€. After removing outliers we got a mean of 123.51€. These values are for all the cities combined, meanwhile for:

Amsterdam:

- Upper bound is: 355.11€, lower bound is: 0€. Mean is: 208.20€
- After removing outliers mean: 165.97€

Antwerp:

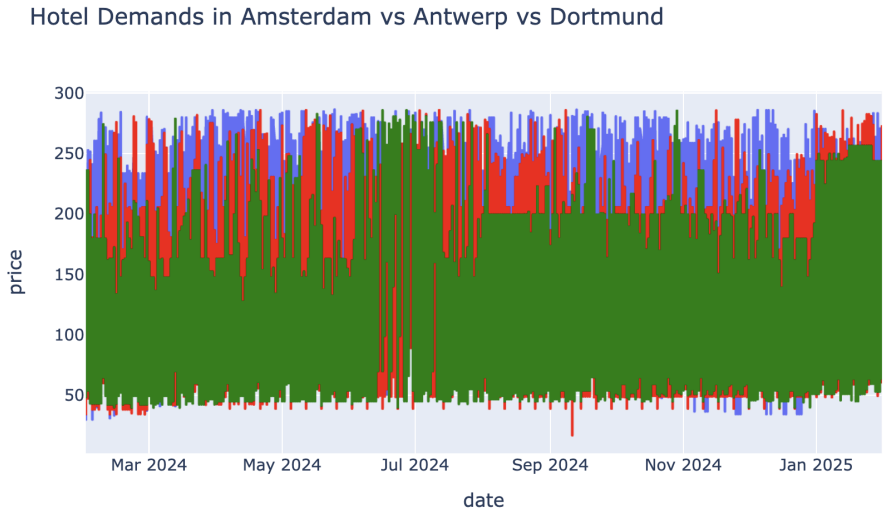
- Upper bound is: 210.43€, lower bound is: 10.02€. Mean is: 129.59€
- After removing outliers mean: 108.84€

Dortmund:

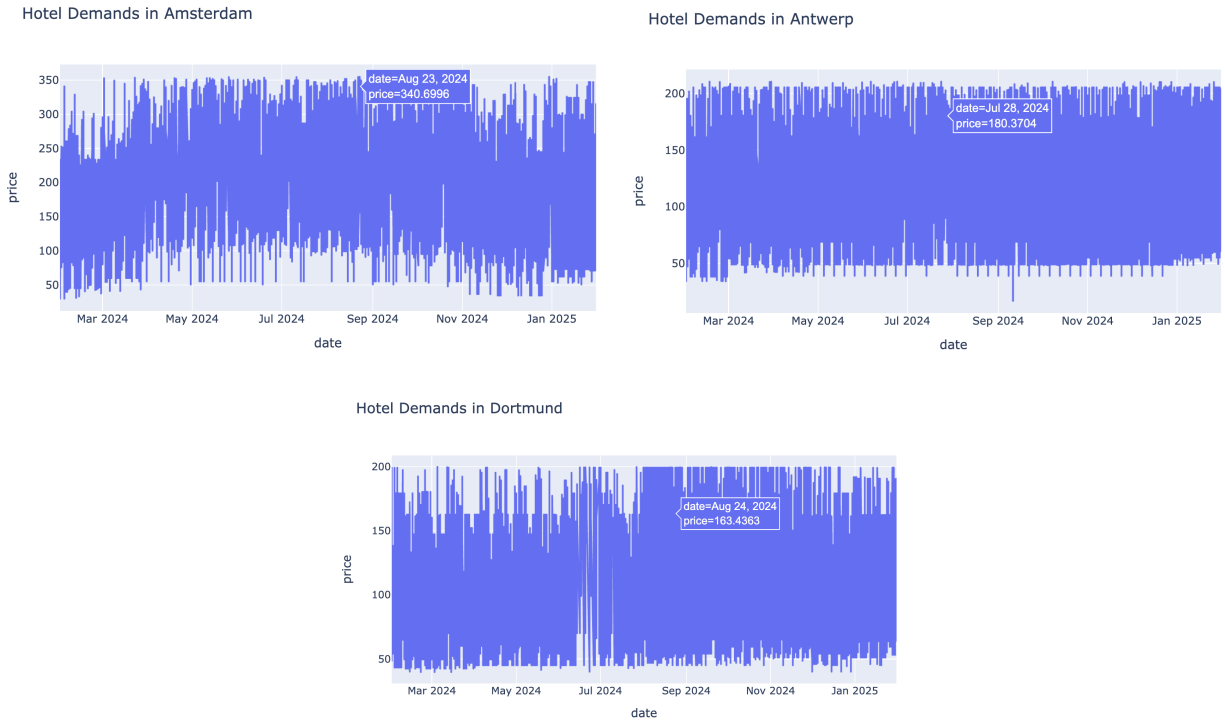
- Upper bound is: 200.21€, lower bound is: 0€. Mean is: 123.31€

- After removing outliers mean: 96.75€

Here we can see hotel demands and prices by time of year and color coded by place, where blue represents Amsterdam, red represents Antwerp, and green represents Dortmund.



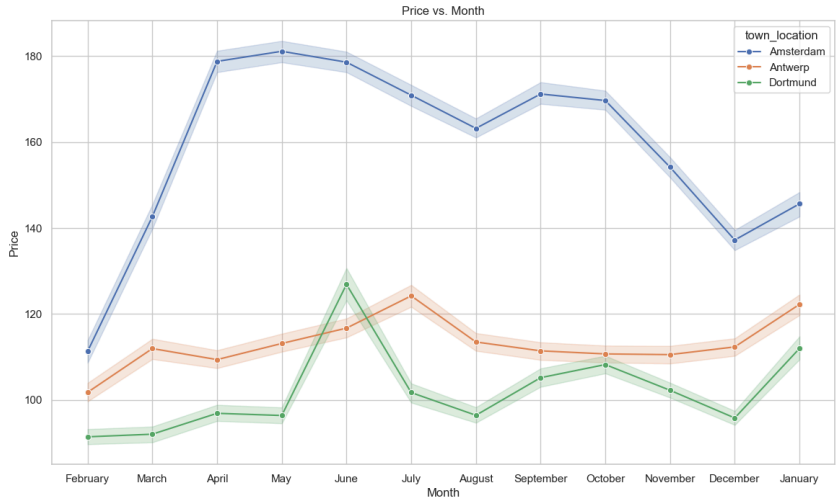
This graph can be separated in order to have a clearer view of the whole picture for each city.



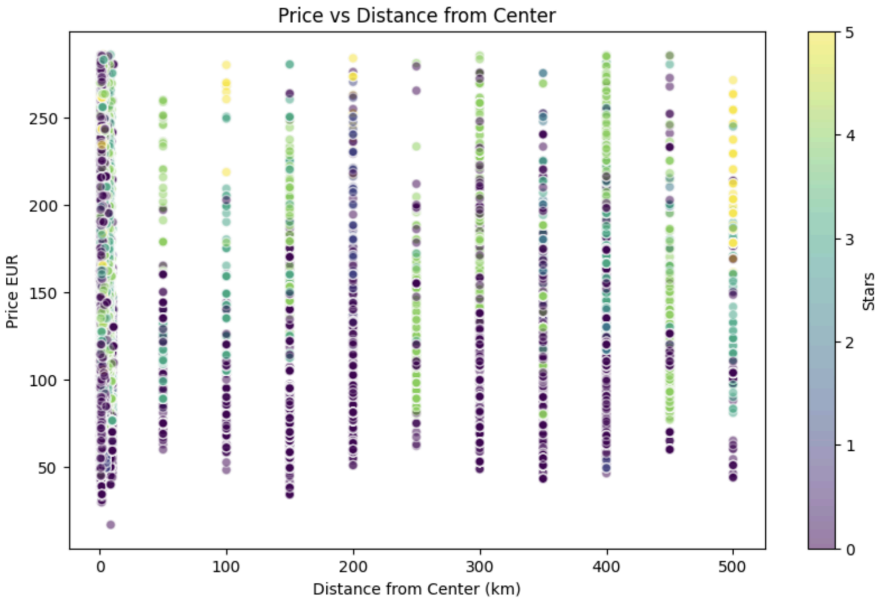
The prices in Amsterdam seem to have high demand throughout the year, apart from February - March where prices seem low. Prices in Antwerp look pretty much the same year round, with both cheap and

expensive accommodations available. Dortmund seems to have a spike in demand during the summer where no hotels can be found with prices less than 100€, but right after this a downward trend can be seen until around August.

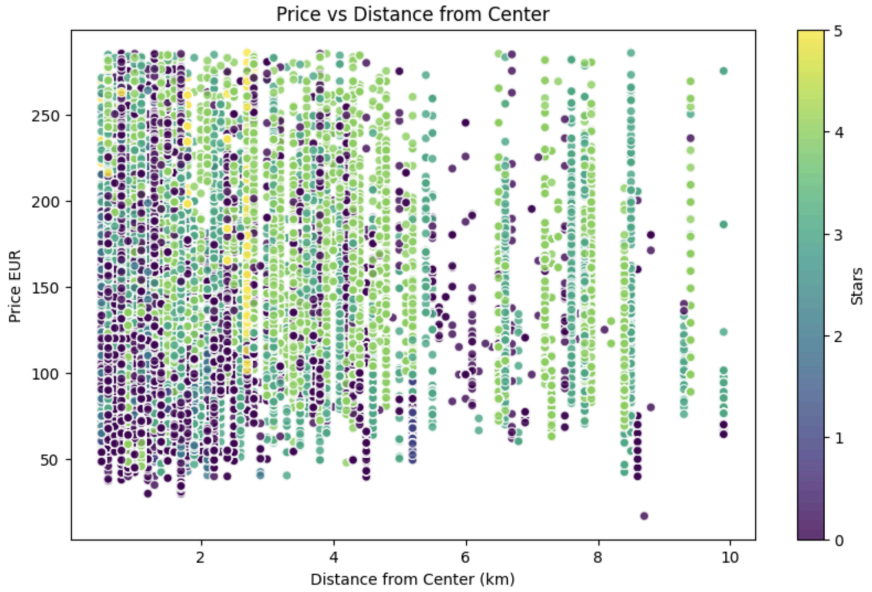
This can more clearly be seen here:



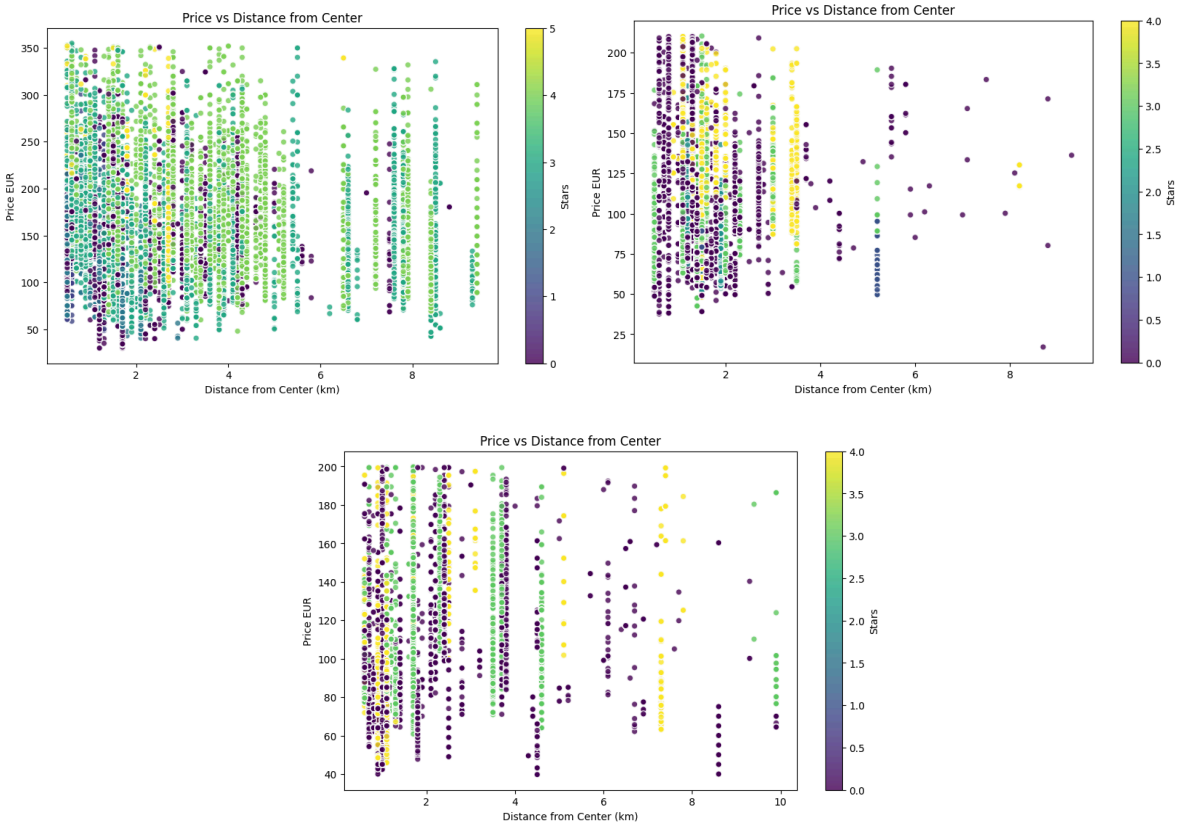
We can gain a lot of information by plotting the prices in regards to how far the hotel is from the city center and the stars it has. We have filled the accommodations with 0 rating stars where no information was present. These types of accommodations are mostly, but not always, apartments and not hotels. Apartments listed on Booking.com can have star ratings, especially if the property has certain amenities or services that align with hotel standards.



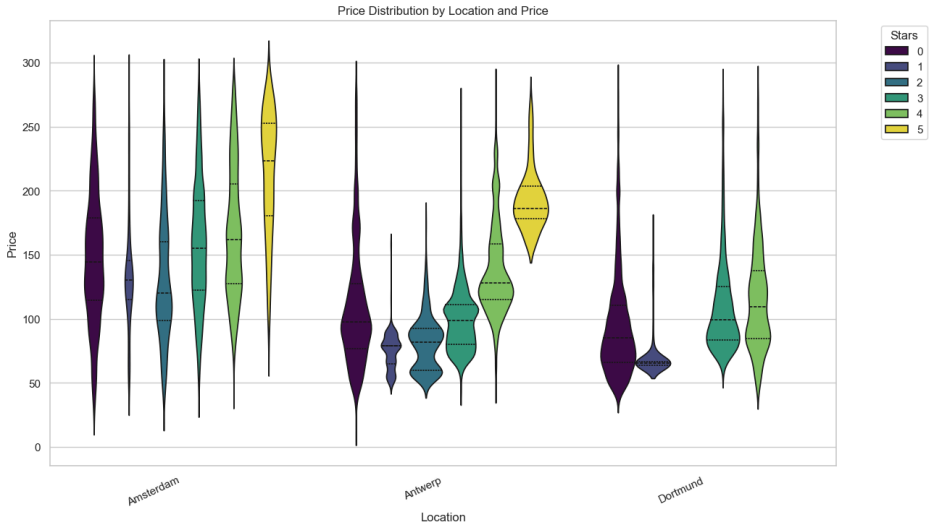
If we inspect hotels that are closest to the center (less than 10 km) we can see that we have an abundance of 0 star rated accommodations with very low prices. This might be potent in one city as opposed to the others, so let's investigate and split this plot by cities yet again.



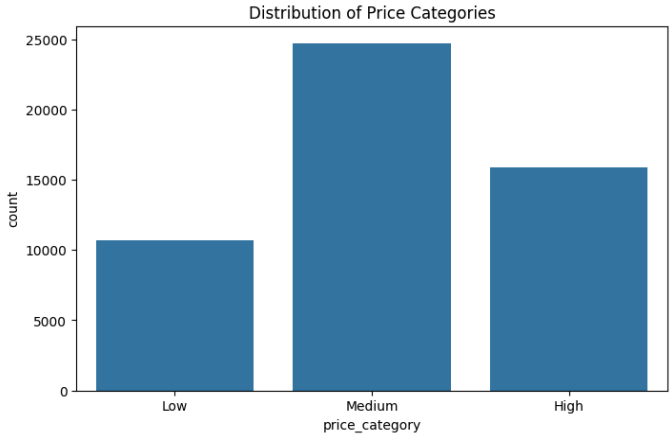
We find that our assumption was correct, most of the 0 star accommodations come from Antwerp, with a few of them from Dortmund. Represented in order Amsterdam, Antwerp, Dortmund.



The plot below might give us a clearer picture of what we are trying to describe above.



We can simplify the continuous data of the **price** column with the help of binning. By splitting the data we create a new column of discrete bins with these categorical values Low [0-80], Medium (80,140], High (140, inf). This type of analysis can help us understand how different price categories relate to average prices, star ratings, and user ratings in the dataset.



Count of Data Points in Each Category:

price_category

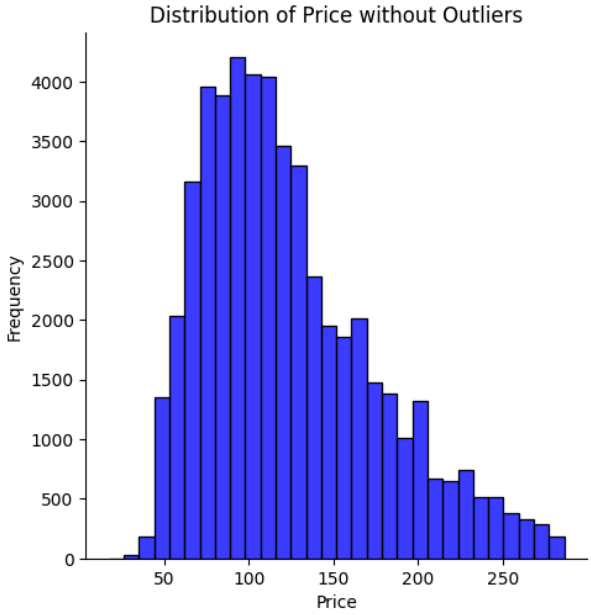
Medium → 24712

High → 15869

Low → 10711

	price		stars	avg_user_ratings
	mean	median	mean	mean
price_category				
Low	65.549390	67.508756	0.923817	7.395042
Medium	107.673661	107.314520	2.031442	7.976866
High	187.324320	178.564725	2.561346	8.257108

We can also visualize the frequency of prices, most of them being in the 100€ range.



Analysis

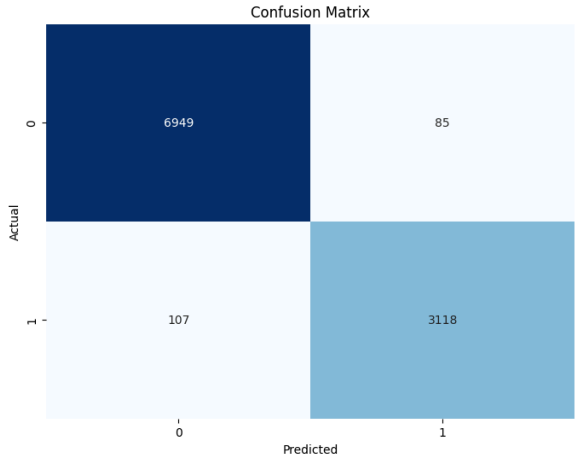
Classification

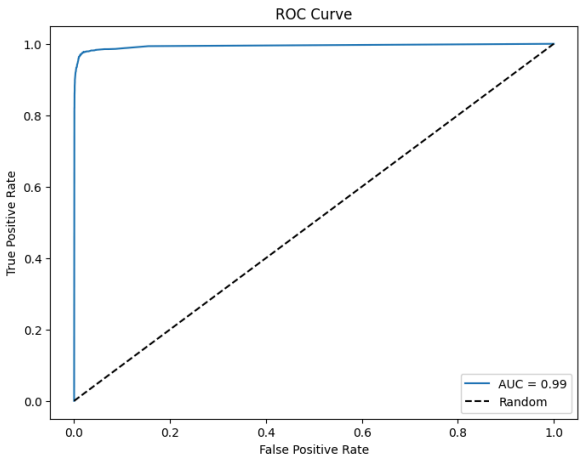
Two types of classifications were performed:

- Binary Random Forest Classification - specifically to predict the presence or absence of free cancellation for accommodations, and
- Multiclass Random Forest Classification - to predict the travel sustainability level of the accommodations.

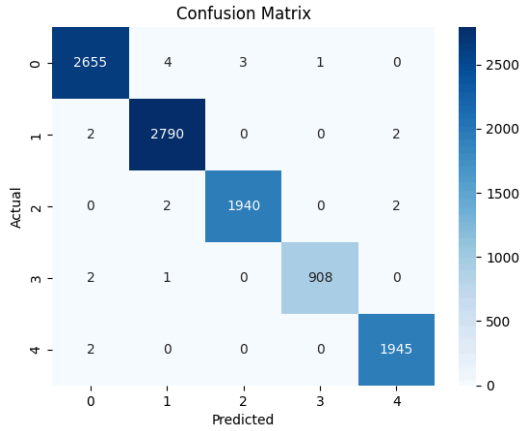
Features selected for the model include price, star rating, distance from the city center, the number of external reviews, booking user ratings, and average user ratings. The dataset is divided into training (80%) and testing (20%) sets, and a Random Forest Classifier with 100 decision trees is trained on the training data. The model is then used to make predictions on the test set.
Binary Random Forest Classification achieved an accuracy of 98.1% in predicting free cancellation, while Multiclass Random Forest Classification achieved an accuracy of 99.7% in predicting travel sustainability levels.

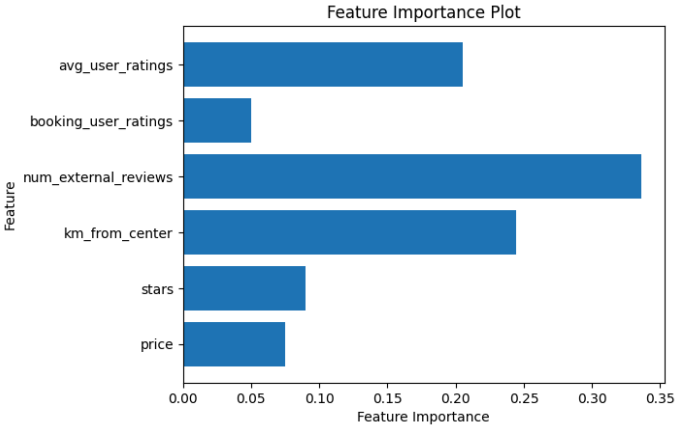
- Binary classification visualization:





- Multiclass classification visualization:





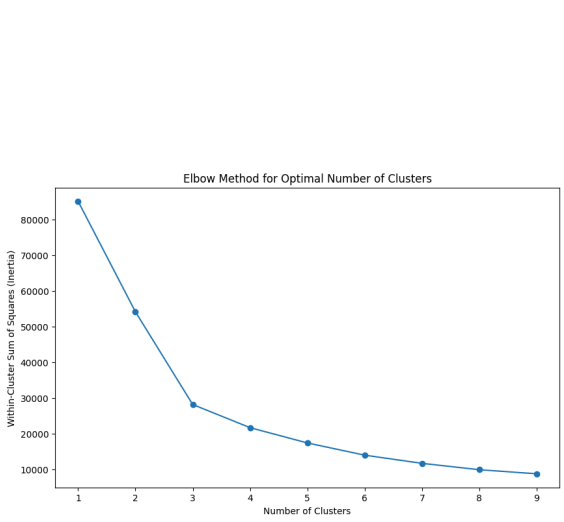
The feature importance plot contains the importance scores assigned to each feature in the dataset. This visualization allows users to identify which features have a more significant impact on the classifier's decision-making process. This can help us understand the relative contribution of different features in the classification model, aiding in feature selection and interpretation.

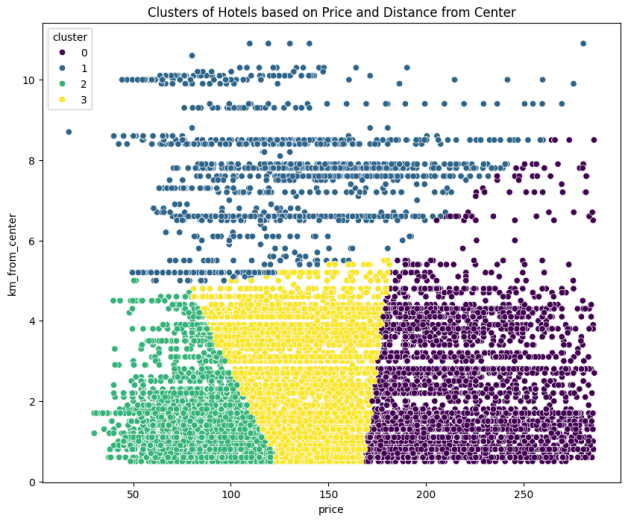
Clustering

The Elbow Method is a technique used in clustering analysis to determine the optimal number of clusters for a dataset. It involves running the clustering algorithm, in our case KMeans, for a range of cluster numbers and plotting the within-cluster-sum-of-squares (WCSS). The "elbow" point on the plot is considered the optimal number of clusters, where the rate of decrease in WCSS starts to diminish. This signifies a balance between minimizing intra-cluster distances and avoiding overfitting.

KMeans partitions the dataset into K clusters (4 to be exact, determined by the Elbow Method), where each data point belongs to the cluster with the nearest mean (centroid). The algorithm iteratively assigns data points to clusters and updates the cluster centroids until convergence. It aims to minimize the sum of squared distances within each cluster.

If the plot exhibits a clear elbow at 3 or 4 clusters, it suggests that adding more clusters may not significantly improve the model's representation of the data. The choice between 3 or 4 clusters can be a trade-off between granularity and interpretability, which is why 4 clusters was the chosen value.

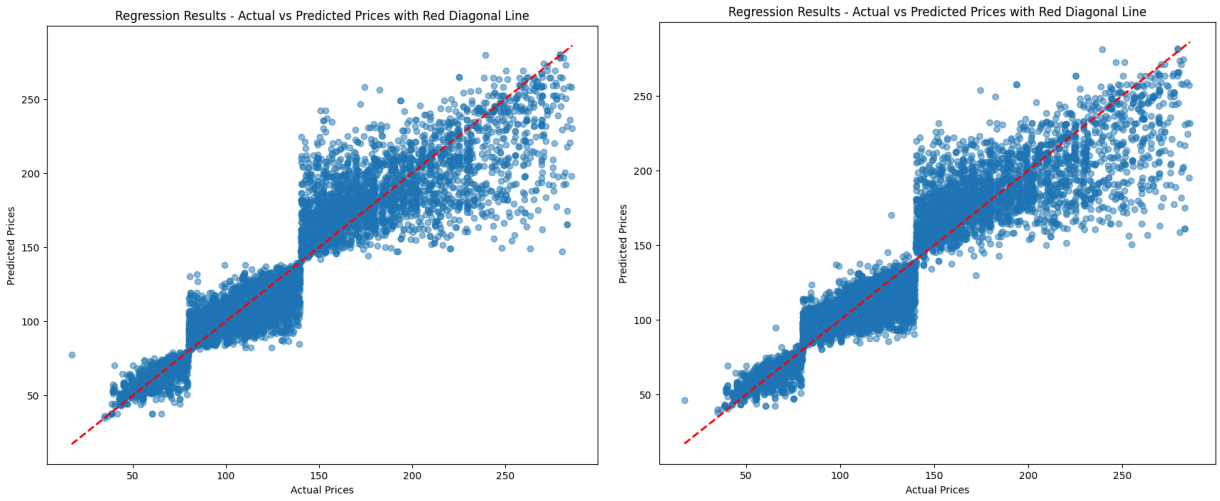




Regression

Categorical features such as 'town_location,' 'price_category,' and 'month' are encoded into numerical representations using a LabelEncoder. Subsequently, the dataset is divided into training and testing sets, and the features are standardized using StandardScaler to ensure uniform scaling across numerical features. The resulting feature matrix (`X`) excludes target-related columns ('price', 'name', 'location', 'date'), while the target variable (`y`) is defined as the 'price' column.

RandomForestRegressor constructs an ensemble of decision trees independently, providing robustness against overfitting. On the other hand, XGBRegressor, an implementation of gradient boosting, builds decision trees sequentially, with each tree refining the errors of the combined ensemble.
RandomForestRegressor and XGBRegressor were compared, with XGBRegressor performing slightly better (mean squared error: 271.34) than RandomForestRegressor (272.21) despite using 9 times more estimators (900 vs. 100). This increase in XGBoost provided better results, while increasing the number of estimators in RandomForestRegressor did nothing.
It's important to note though that that using a very high number of estimators may also increase the risk of overfitting,
XGBoost often offers superior predictive accuracy and computational efficiency, incorporating regularization to control overfitting and is generally favored for its efficiency in practical applications.
We can see the similar results here:



Future Work - Conclusion

Future work involves exploring advanced methodologies for hotel room occupancy forecasting discussed in ‘’FORECASTING HOTEL DEMAND USING MACHINE LEARNING APPROACHES”, a thesis paper from Faculty of the Graduate School of Cornell University seems to be the next step.
The dataset there contains bookings made in the past, something this dataset lacks. The identified methods here present opportunities for improvement and innovation in this project.
This paper introduces several ideas, some of them being: the quadratic relationship between time and Rooms On Hand (ROHs - number of currently available rooms), external factors like local events and competitive set occupancy predictions, differences between the number of booked rooms and the actual

number of final arrivals. Finally, the exploration of advanced machine learning models, Neural Network, might suggest potential for more robust predictions.

These considerations outline promising directions for future research in hotel room occupancy forecasting, aiming to advance predictive capabilities and contribute to the evolving landscape of hospitality management.