Guest Satisfaction Prediction

Team ID: SC_9

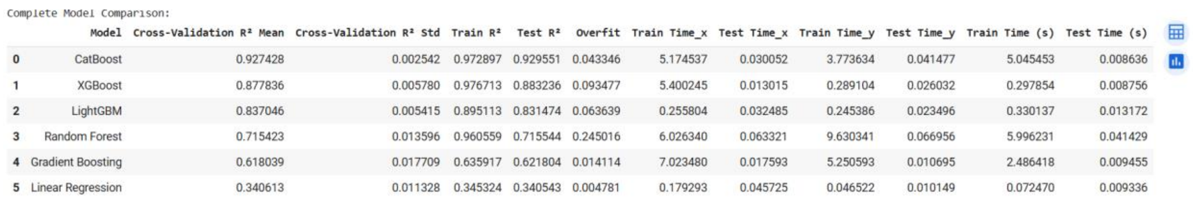
Student ID	Student Name
2022170629	ميسون حلمي عبدالرحمن علي شعبان
2022170162	رؤي محمد لطفي محمد
2022170447	منة ياسر حامد مصطفى
2022170340	مايا سامح احمد سمير
2022170051	أروى ابوزليمة محمد نور

- ➢ **Feature Engineering** ---------- 12
- 1) Explanation for some features: ---------- 12
- ➢ **Filling Missing Values** ---------- 15
- 1) Normal Cases: ---------- 15
- ➢ **Encoding :** ---------- 17
- ➢ **Milestone 1 Feature selection:** ---------- 19
- ➢ **Visualizations** ---------- 22
  - • host_response_rate ---------- 22
  - • host_listings_count ---------- 22
  - ---------- 22
  - • accommodates ---------- 23
  - • bathrooms ---------- 23
  - • bedrooms ---------- 24
  - ---------- 24
  - • beds ---------- 24
  - • nightly_price ---------- 25
  - • price_per_stay ---------- 25
  - • security_deposit ---------- 26
  - - cleaning_fee ---------- 26
  - - guests_included ---------- 27
  - • extra_people ---------- 27
  - • minimuma_nights ---------- 28
  - • maximum_nights ---------- 28
  - • number_of_stays ---------- 29
  - • number_of_reviews ---------- 29
  - • review_scores_rating ---------- 30
- ➢ Handling outliers ---------- 33
- ➢ **Text Analysis & Preprocessing** ---------- 35
- ➢ **Regression Model Trials with Different Selected Features** ---------- 37
- ➢ **Some Graphs for Each Model Performance** ---------- 39
- ---------- 39
- ➢ **Model Performance Conclusion** ---------- 40
- ➢ **Classification Model Trial with Different Selected Features** ---------- 42

➢ **Some Graphs for Each Model Performance** --- 45
➢ **Conclusion**--- 46

➢ Observations

According to our EDA We have observed the following:

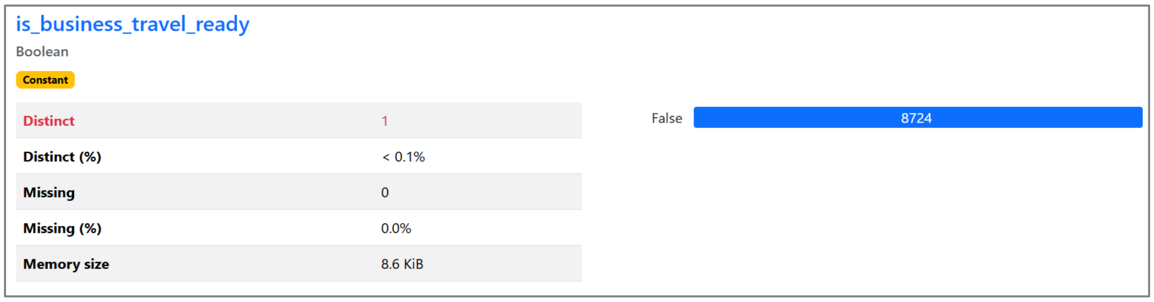


```
Complete Model Comparison:
```

	Model	Cross-Validation R² Mean	Cross-Validation R² Std	Train R²	Test R²	Overfit	Train Time_x	Test Time_x	Train Time_y	Test Time_y	Train Time (s)	Test Time (s)
0	CatBoost	0.927428	0.002542	0.972897	0.929551	0.043346	5.174537	0.030052	3.773634	0.041477	5.045453	0.008636
1	XGBoost	0.877836	0.005780	0.976713	0.883236	0.093477	5.400245	0.013015	0.289104	0.026032	0.297854	0.008756
2	LightGBM	0.837046	0.005415	0.895113	0.831474	0.063639	0.255804	0.032485	0.245386	0.023496	0.330137	0.013172
3	Random Forest	0.715423	0.013596	0.960559	0.715544	0.245016	6.026340	0.063321	9.630341	0.066956	5.996231	0.041429
4	Gradient Boosting	0.618039	0.017709	0.635917	0.621804	0.014114	7.023480	0.017593	5.250593	0.010695	2.486418	0.009455
5	Linear Regression	0.340613	0.011328	0.345324	0.340543	0.004781	0.179293	0.045725	0.046522	0.010149	0.072470	0.009336

- Column name: is_business_travel_ready

  Observation: all false values so it's not a differentiating feature



is_business_travel_ready

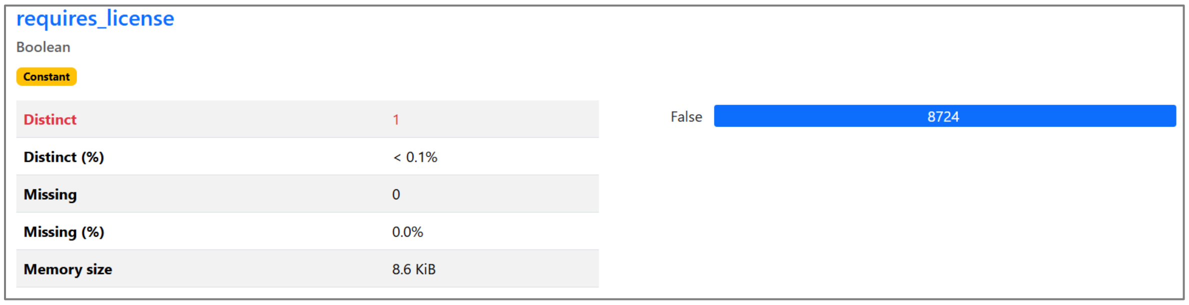
Boolean

Constant

Distinct	1
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	8.6 KiB

- Column name: requires_license

  Observation: all false values so it's not a differentiating feature



requires_license

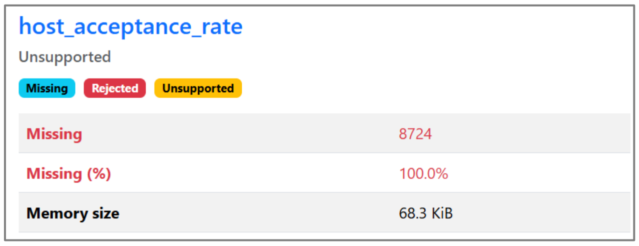
Boolean

Constant

Distinct	1
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	8.6 KiB

- Column name: host_acceptance_rate

  Observation: empty



host_acceptance_rate

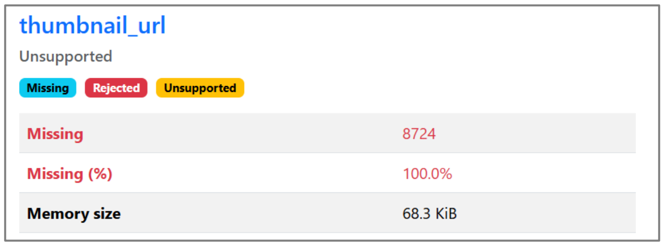
Unsupported

Missing Rejected Unsupported

Missing	8724
Missing (%)	100.0%
Memory size	68.3 KiB

- Column name: thumbnail_url

  Observation: empty



thumbnail_url

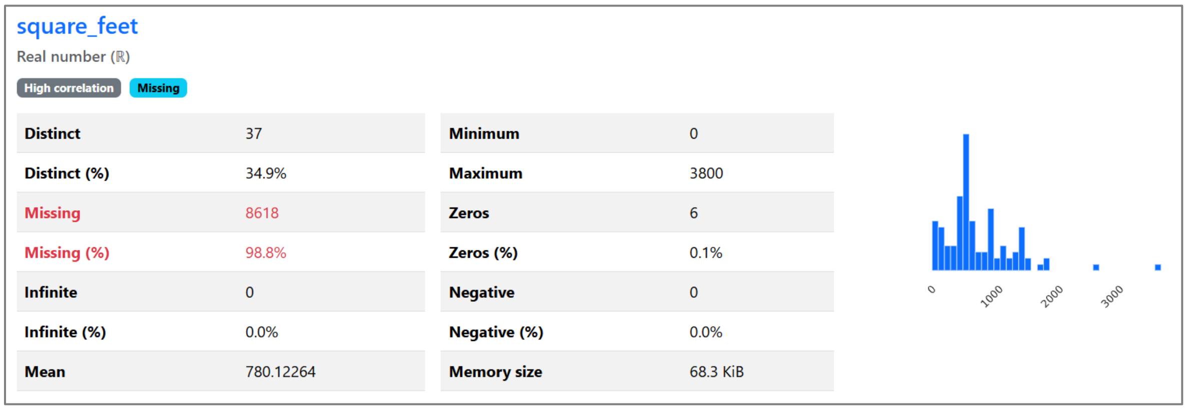
Unsupported

Missing Rejected Unsupported

Missing	8724
Missing (%)	100.0%
Memory size	68.3 KiB

- Column name: square_feet

  Observation: 98% empty



square_feet

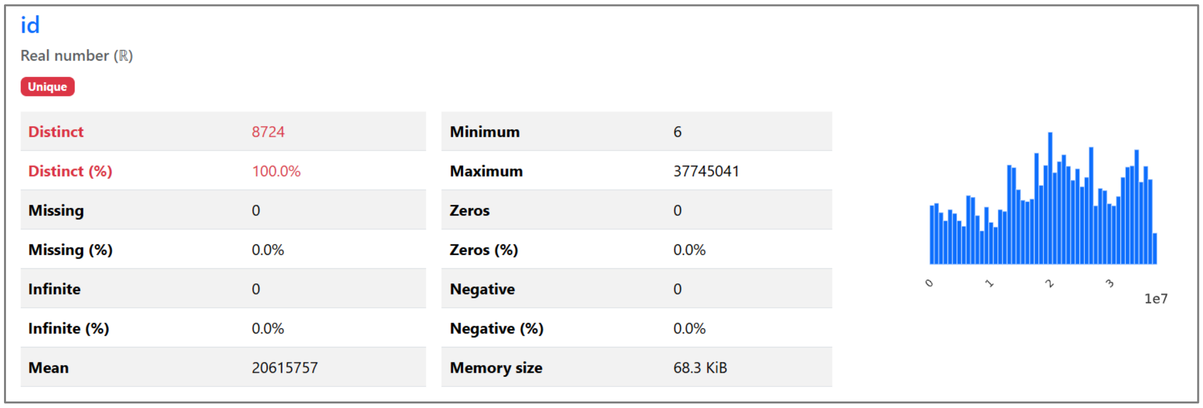
Real number (ℝ)

High correlation Missing

Distinct	37	Minimum	0
Distinct (%)	34.9%	Maximum	3800
Missing	8618	Zeros	6
Missing (%)	98.8%	Zeros (%)	0.1%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	780.12264	Memory size	68.3 KiB

- Column name: ID

  Observation: distinct



id

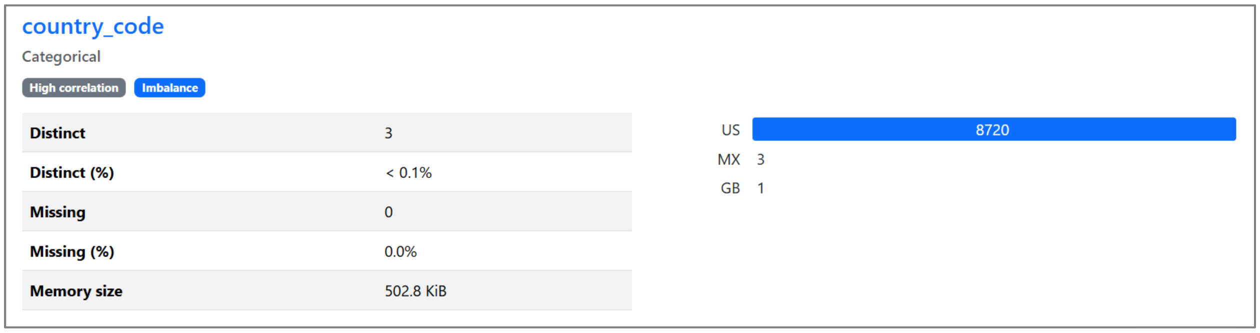
Real number (ℝ)

Unique

Distinct	8724	Minimum	6
Distinct (%)	100.0%	Maximum	37745041
Missing	0	Zeros	0
Missing (%)	0.0%	Zeros (%)	0.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	20615757	Memory size	68.3 KiB

- Column name: country_code

  Observation: all values are the same (US)



country_code

Categorical

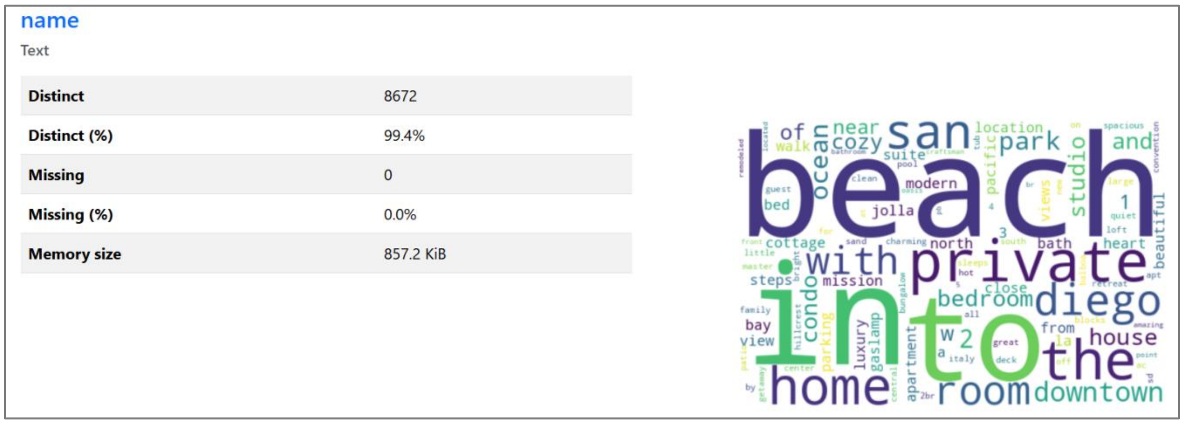
High correlation Imbalance

Distinct	3
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	502.8 KiB

US	8720
MX	3
GB	1

- Column name: name

  Observation: almost all values are distinct values



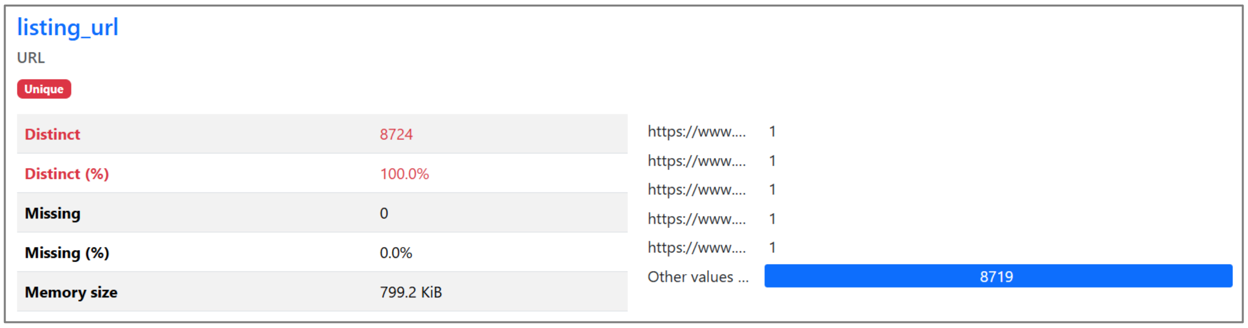
name

Text

Distinct	8672
Distinct (%)	99.4%
Missing	0
Missing (%)	0.0%
Memory size	857.2 KiB

- Column name: listing_url

  Observation: distinct and we couldn’t find new data to fetch



listing_url

URL

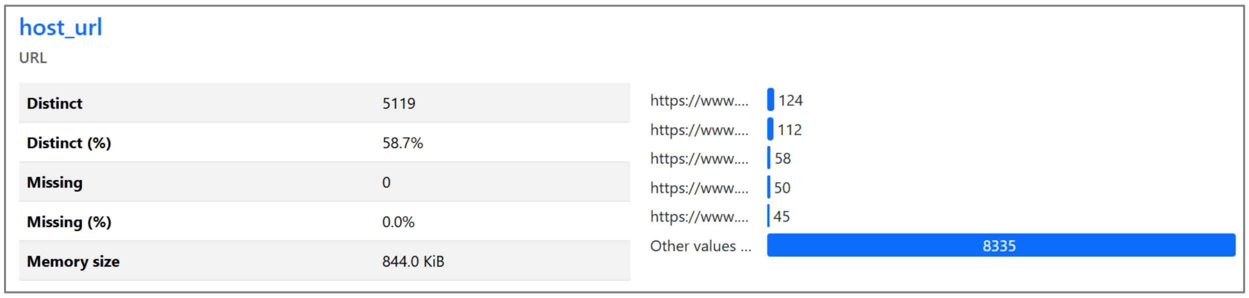
Unique

Distinct	8724
Distinct (%)	100.0%
Missing	0
Missing (%)	0.0%
Memory size	799.2 KiB

https://www....	1
https://www....	1
https://www....	1
https://www....	1
https://www....	1
Other values ...	8719

- Column name: host_url

  Observation: distinct and we couldn’t find new data to fetch



host_url

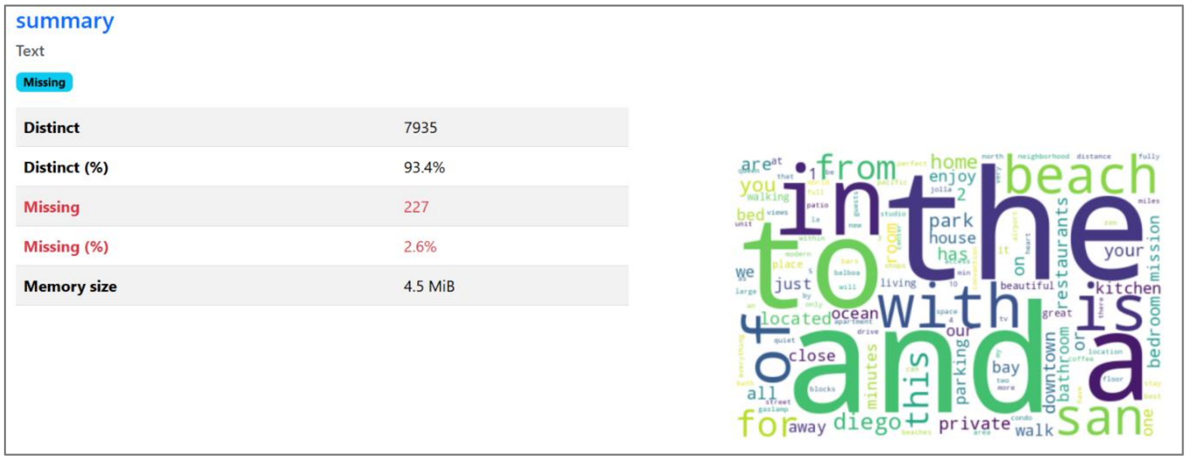
URL

Distinct	5119
Distinct (%)	58.7%
Missing	0
Missing (%)	0.0%
Memory size	844.0 KiB

https://www....	124
https://www....	112
https://www....	58
https://www....	50
https://www....	45
Other values ...	8335

- Column name: summary

  Observation: has 93% distinct values & 2.6% missing values, Also, it almost has the same data as description and space



summary

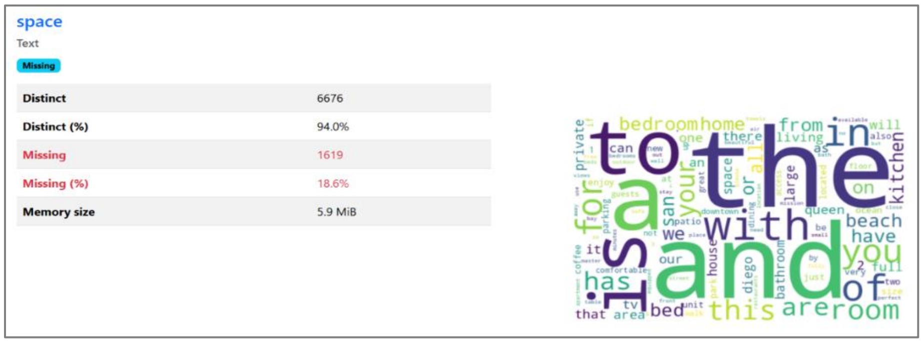
Text

Missing

Distinct	7935
Distinct (%)	93.4%
Missing	227
Missing (%)	2.6%
Memory size	4.5 MiB

- Column name: space

➢ Observation: has 94% distinct values & 18% missing values, it Also, it almost has the same data as description and summary



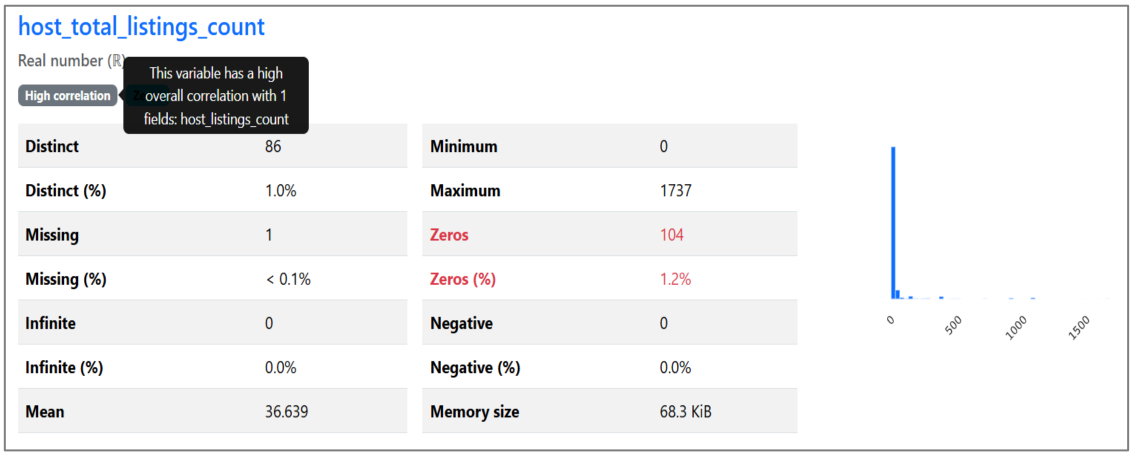
space

Text

Missing

Distinct	6676
Distinct (%)	94.0%
Missing	1619
Missing (%)	18.6%
Memory size	5.9 MiB

- Column name: host_total_listings_count
  Observation: correlation between it and host_listings_count is 1



host_total_listings_count

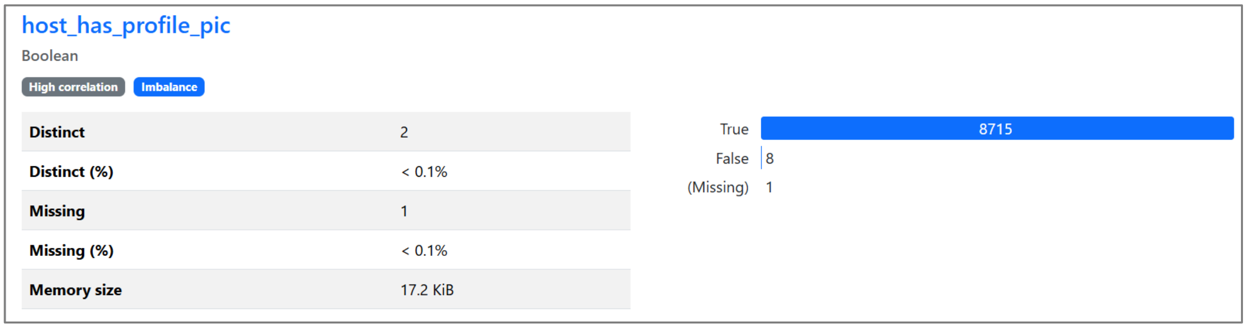
Real number (ℝ)

High correlation

This variable has a high overall correlation with 1 fields: host_listings_count

Distinct	86	**Minimum**	0
Distinct (%)	1.0%	**Maximum**	1737
Missing	1	**Zeros**	104
Missing (%)	< 0.1%	**Zeros (%)**	1.2%
Infinite	0	**Negative**	0
Infinite (%)	0.0%	**Negative (%)**	0.0%
Mean	36.639	**Memory size**	68.3 KiB

- Column name: host_has_profile_pic
  Observation: all true values



host_has_profile_pic

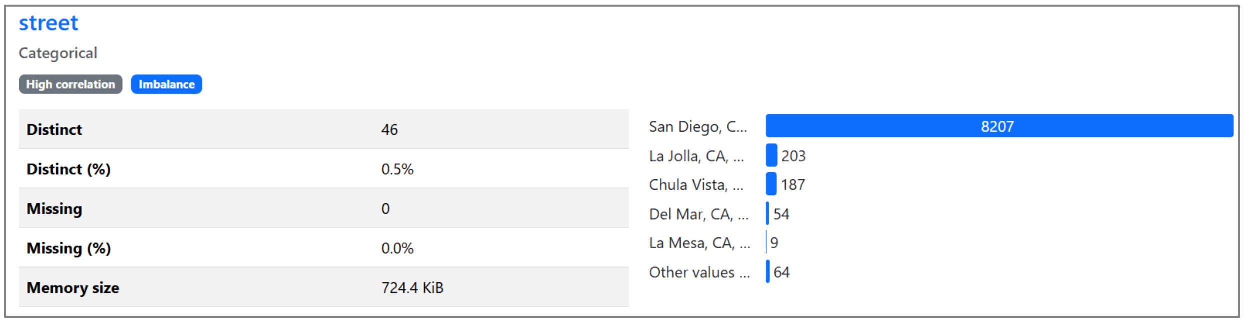
Boolean

High correlation Imbalance

Distinct	2
Distinct (%)	< 0.1%
Missing	1
Missing (%)	< 0.1%
Memory size	17.2 KiB

True	8715
False	8
(Missing)	1

- Column name: street
  Observation: 99% is San Diego, CA, United States



street

Categorical

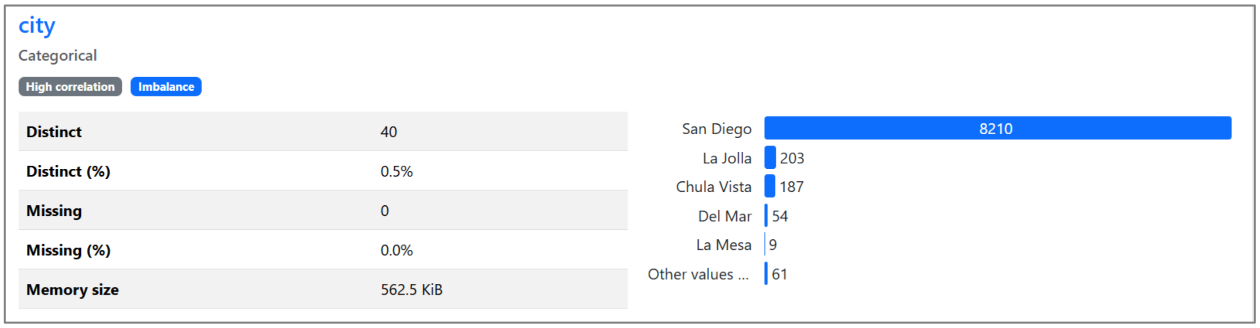
High correlation Imbalance

Distinct	46
Distinct (%)	0.5%
Missing	0
Missing (%)	0.0%
Memory size	724.4 KiB

San Diego, C...	8207
La Jolla, CA, ...	203
Chula Vista, ...	187
Del Mar, CA, ...	54
La Mesa, CA, ...	9
Other values ...	64

- Column name: city

  Observation: 98% is San Diego



city

Categorical

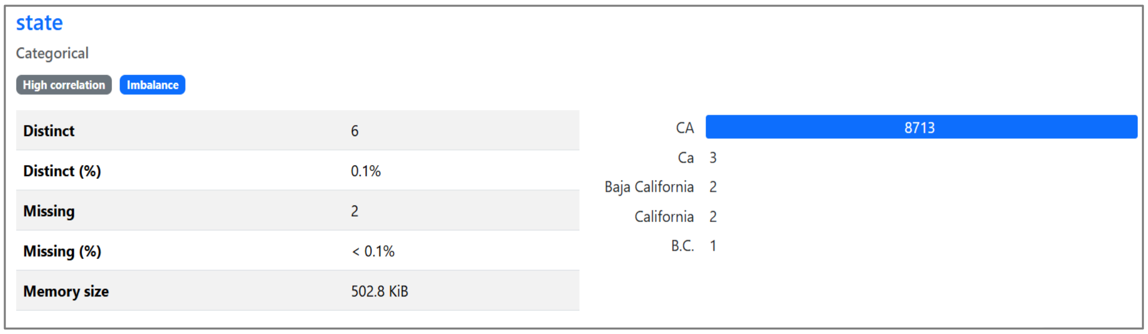
High correlation | Imbalance

Distinct	40
Distinct (%)	0.5%
Missing	0
Missing (%)	0.0%
Memory size	562.5 KiB

San Diego	8210
La Jolla	203
Chula Vista	187
Del Mar	54
La Mesa	9
Other values ...	61

- Column name: state

  Observation: 99% is CA



state

Categorical

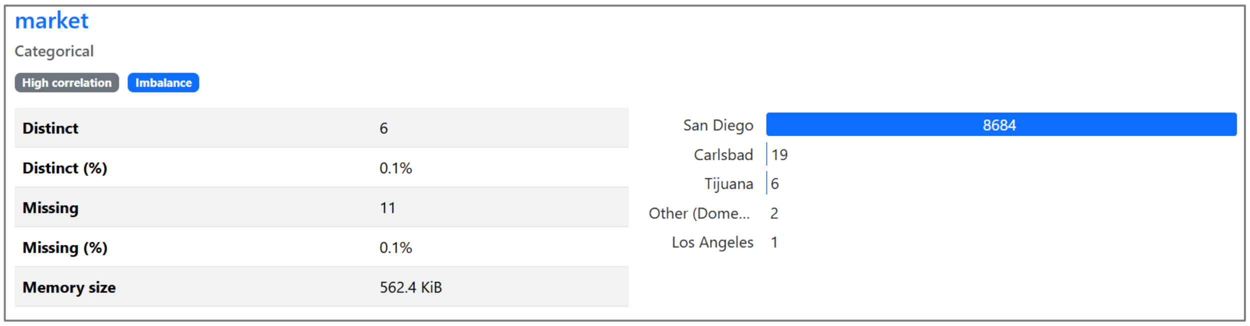
High correlation | Imbalance

Distinct	6
Distinct (%)	0.1%
Missing	2
Missing (%)	< 0.1%
Memory size	502.8 KiB

CA	8713
Ca	3
Baja California	2
California	2
B.C.	1

- Column name: market

  Observation: 99% is San Diego



market

Categorical

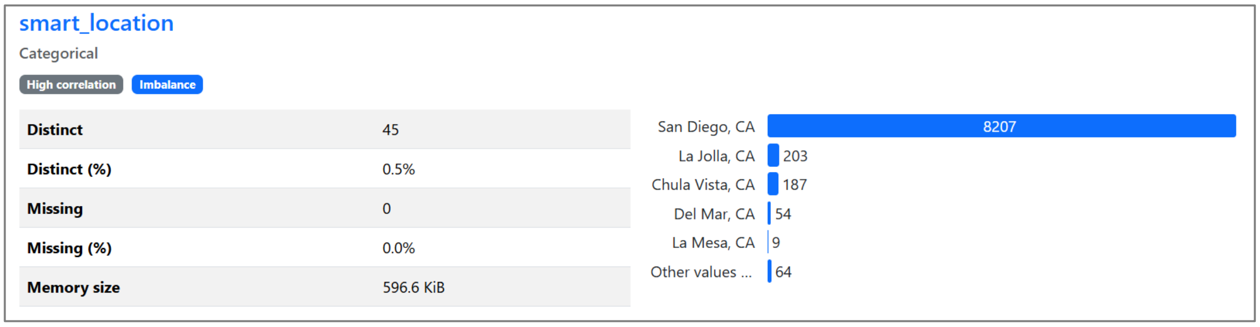
High correlation | Imbalance

Distinct	6
Distinct (%)	0.1%
Missing	11
Missing (%)	0.1%
Memory size	562.4 KiB

San Diego	8684
Carlsbad	19
Tijuana	6
Other (Dome...	2
Los Angeles	1

- Column name: smart_location

  Observation: 99% is San Diego, CA



smart_location

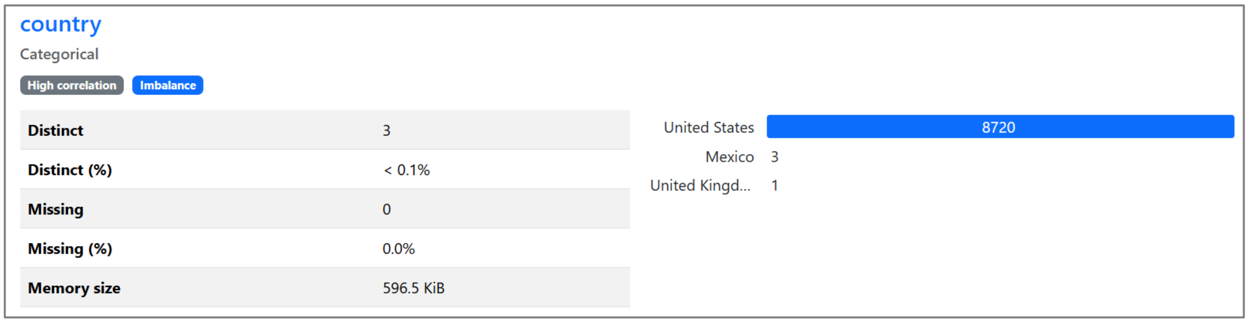
Categorical

Distinct	45
Distinct (%)	0.5%
Missing	0
Missing (%)	0.0%
Memory size	596.6 KiB

San Diego, CA	8207
La Jolla, CA	203
Chula Vista, CA	187
Del Mar, CA	54
La Mesa, CA	9
Other values ...	64

- Column name: country

  Observation: 99% is United states



country

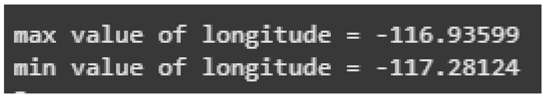
Categorical

Distinct	3
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	596.5 KiB

United States	8720
Mexico	3
United Kingd...	1

- Column name: longitude

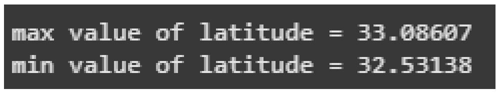
  Observation: they are almost identical as the difference is in very small decimal



```
max value of longitude = -116.93599
min value of longitude = -117.28124
```

- Column name: latitude

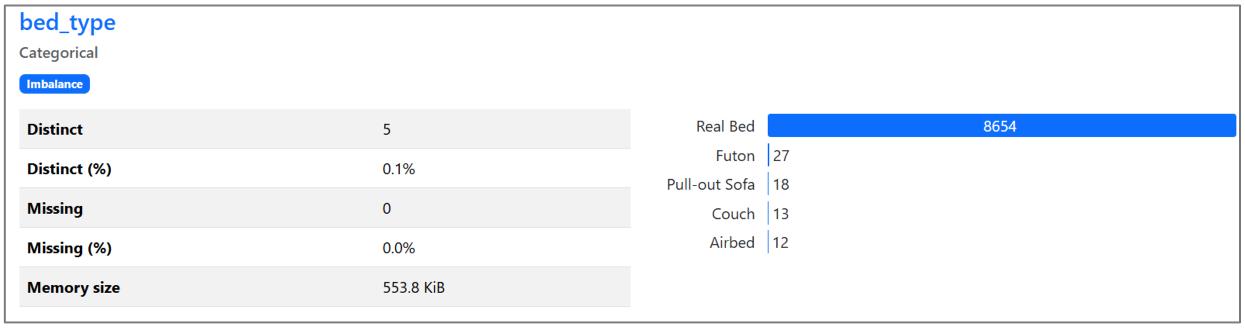
  Observation: they are almost identical as the difference is in very small decimal



```
max value of latitude = 33.08607
min value of latitude = 32.53138
```

- Column name: bed_type

  Observation: 99% is Real bed



bed_type

Categorical

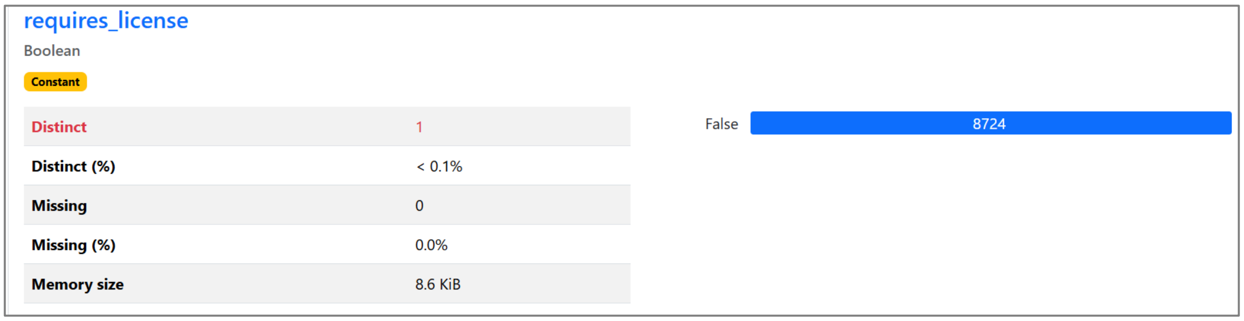
Imbalance

Distinct	5
Distinct (%)	0.1%
Missing	0
Missing (%)	0.0%
Memory size	553.8 KiB

Real Bed	8654
Futon	27
Pull-out Sofa	18
Couch	13
Airbed	12

- Column name: requires_license

  Observation: 100% is false values



requires_license

Boolean

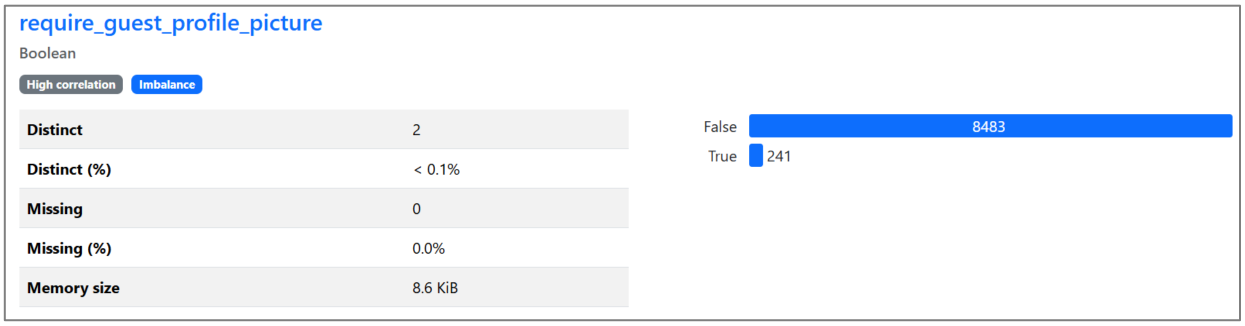
Constant

Distinct	1
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	8.6 KiB

False	8724

- Column name: require_guest_profile_picture

  Observation: 97% is false



require_guest_profile_picture

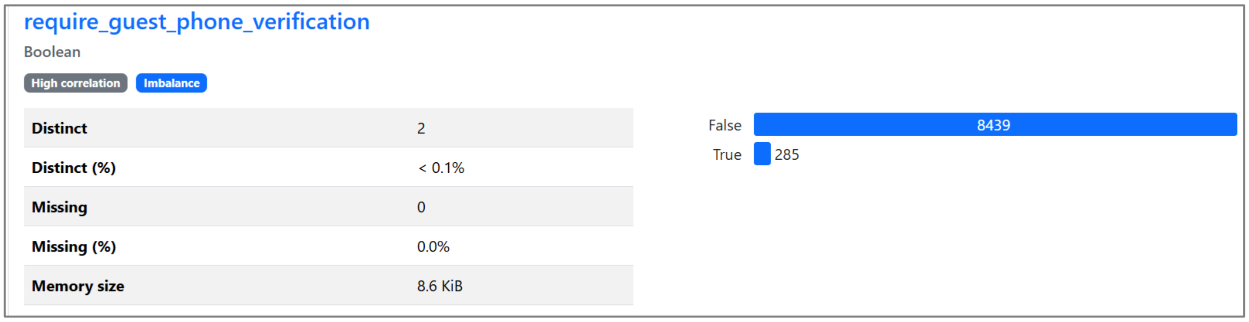
Boolean

High correlation Imbalance

Distinct	2
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	8.6 KiB

False	8483
True	241

- Column name: require_guest_phone_verification
  Observation: 97% is false


require_guest_phone_verification

Boolean

High correlation Imbalance

Distinct	2
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	8.6 KiB

False	8439
True	285

- Column name: neighborhood
  Observation: we used neighborhood cleansed instead, as it has no missing values

- Column name: is_business_travel_ready
  Observation: all false values

- Column name: host_name
  Observation: we can use the host_Id instead

➢ Feature Engineering

1) Explanation for some features:

- **Host Response Power***:* We combine the host's response rate with the number of listings they manage. A host who responds quickly and operates multiple listings is typically more professional and experienced.

- **Host Commitment**: This feature measures a host's dedication over time. A host who has been active for a long period but has relatively few stays may indicate lower engagement.

- **Bedroom Quality**: This reflects the balance between the number of bedrooms and bathrooms. A property with too few bathrooms compared to bedrooms might make guests uncomfortable.

- **Space per Guest**: By looking at the ratio of accommodation to the number of beds, we estimate how much personal space each guest has.

- **Essential Amenities**: We tracked the availability of crucial amenities such as Wi-Fi, air conditioning, and a kitchen. Missing any of these essentials can seriously impact a guest’s stay and, in turn, their review score.

- **Review Consistency**: A property with many reviews but fluctuating scores signals inconsistency. Stable, high-quality performance is key to maintaining strong reviews, and this feature captures that trend.

- **Price Value**: Guests naturally assess whether the experience matches the price they paid. Listings that seem overpriced relative to what they offer are more likely to receive lower scores.

2) Basic information about selected features:

- From the first_review and last_review we extracted the duration in days

  ```
  df['review_duration_days'] =
  (df['last_review'] - df['first_review']).dt.days
  ```

  We have also extracted the duration in months, so the numbers are not confusing for the model.

  ```
  df['review_duration_months'] = df['review_duration_days'] / 30
  ```

  From this we observed that the **review_duration_months** is the most suitable to use, as its average is reasonable.

- From the host_Since column we have extracted the host duration in days, months, and years from today.

  ```
  df['host_duration_days'] =
  (pd.to_datetime("today") - df['host_since']).dt.days
  df['host_duration_months'] = df['host_duration_days'] / 30
  df['host_duration_years'] = df['host_duration_days'] / 365.25
  ```

  From this we observed that the **host_duration_years** is the most suitable to use as its average is reasonable.

- Also, from the host_since column we have extracted the host_duration relative to the oldest host.

  ```
  oldest_host_date = df['host_since'].min()
  df['host_relative_age_days'] =
  (df['host_since'] - oldest_host_date).dt.days
  df['host_relative_age_months'] = df['host_relative_age_days'] / 30
  df['host_relative_age_years'] =
  df['host_relative_age_days'] / 365.25
  ```

  In this case as well, **host_relative_age_years** offered the most meaningful and interpretable representation.

- From host_response_rate and host_listings_count we have extracted host_response_power

```
df['host_response_power'] =
df['host_response_rate'] * np.log1p(df['host_listings_count'])
```

- From bedrooms and bathrooms, we have extracted bedroom_quality

```
df['bedroom_quality'] = df['bedrooms'] / (df['bathrooms'] + 0.5)
```

- From accommodates and beds we have extracted space_per_guest

```
df['space_per_guest'] = df['accommodates'] / (df['beds'] + 0.1)
```

- From host_duration_days we have extracted seasonal_demand

```
df['seasonal_demand'] =
np.sin(2 * np.pi * (df['host_duration_days'] % 365) / 365)
```

- From review_duration_days we have extracted recent_review_boost

```
df['recent_review_boost'] =
np.where(df['review_duration_days'] < 30, 1, 0.5)
```

- From has_wifi, has_air_conditioning and has_kitchen we have extracted essential_amenities.

```
df['essential_amenities'] =
df[['has_wifi', 'has_air_conditioning', 'has_kitchen']].sum(axis=1)
```

- From has_hot_tub, has_hot_tub, has_pool, has_gym we have extracted luxury_amenities.

```
df['luxury_amenities'] =
df[['has_hot_tub', 'has_pool', 'has_gym']].sum(axis=1)
```

- From number_of_reviews and review_scores_rating we have extracted review_consistency.

```
df['review_consistency'] =
df['number_of_reviews'] / (df['review_scores_rating'] + 1)
```

- From review_scores_rating and number_of_reviews we have extracted positive_momentum.

```
df['positive_momentum'] =
df['review_scores_rating'] * np.log1p(df['number_of_reviews'])
```

- From nightly_price and accommodates we have extracted price_value

```
df['price_value'] = df['nightly_price'] / df['accommodates']
```

➢ Filling Missing Values

1) Normal Cases:

- Filling text columns like **('description', 'transit', 'access', 'interaction', 'house_rules')** with ‘No information provided’

- Filling categorical columns like (**'host_location', 'host_is_superhost', 'host_identity_verified')** with mode.

2) Special Cases:

- Fill host_neighbourhood using neighbourhood_cleansed using group by as they have a higher percentage of missing values.

```
df['host_neighbourhood'] =
df.groupby('neighbourhood_cleansed')['host_neighbourhood']
   .transform(lambda x: x.fillna(x.mode()[0] if not x.mode().empty
else df['host_neighbourhood'].mode()[0]))
```

- Fill host_response_time using host_is_superhost and host_response_rate using group by as they have a higher percentage of missing values.

```
df['host_response_time'] = df.groupby(['host_is_superhost',
'host_response_rate'])['host_response_time']
   .transform(lambda x: x.fillna(x.mode()[0] if not x.mode().empty
else df['host_response_time'].mode()[0]))
```

- Fill neighborhood_overview with neighbourhood_cleansed using group by as they have a higher percentage of missing values.

```
df['neighborhood_overview'] = df.groupby('neighbourhood_cleansed')
['neighborhood_overview'].transform( lambda x:x.fillna(x.mode()[0]
if not x.mode().empty
else "No overviewavailable"))
```

- If any of them are still missing fill with overall Mode
- Fill 'host_about' with this formula.

```
df['host_about'] = df.apply(
    lambda row: (
        "is a host offering a {row['room_type']} in
{row['neighbourhood_cleansed']}. "
        f"Guests usually enjoy a {row['property_type']} with great
hospitality."
        if pd.isnull(row['host_about']) else row['host_about']
    ), axis=1
)
```

- Fill notes with this formula.

```
df['notes'] = df.apply(
    lambda row: (
        f"This {row['room_type']} in {row['neighbourhood_cleansed']}
offers a comfortable stay. "
        f"Arrival instructions and local tips will be shared after booking."
        if pd.isnull(row['notes']) else row['notes']
    ), axis=1
)
```

- provides a contextual default based on the existing data, helping to maintain meaningful descriptions and improve the quality of the dataset.

➢ Encoding :

1) Binary Replaced Columns:

```
df = df.replace({
    'require_guest_phone_verification': {'t': 1, 'f': 0},
    'require_guest_profile_picture': {'t': 1, 'f': 0},
    'instant_bookable': {'t': 1, 'f': 0},
    'is_location_exact': {'t': 1, 'f': 0},
    'host_identity_verified': {'t': 1, 'f': 0},
    'host_has_profile_pic': {'t': 1, 'f': 0},
    'is_business_travel_ready': {'t': 1, 'f': 0},
    'host_is_superhost': {'t': 1, 'f': 0},
    'requires_license': {'t': 1, 'f': 0}
})
```

This step replaces boolean-like string values ('t' for true and 'f' for false) with numerical equivalents (1 for true and 0 for false).

2) Label Encoded Categorical Columns:

```
label_cols = ['cancellation_policy_cleaned', 'room_type_cleaned',
'property_type_cleaned', 'host_neighbourhood_cleaned']
```

Label encoding is applied to categorical columns to convert their unique categories into integer labels.
For example, if room_type_cleaned contains values like "Entire home/apt", "Private room", and "Shared room", they will be encoded as 0, 1, and 2, respectively.
This transformation is necessary to apply **Nominal relationship** as machine learning algorithms cannot process text directly.

3) Embedding Layer for Host Neighbourhood:

An embedding layer is created to represent the host neighborhoods in a dense vector space. This is particularly useful in deep learning models where categorical variables need to be represented in a meaningful way.

- Parameters :
  input_dim: Total number of unique neighborhoods (num_neighborhoods).
  output_dim: Dimensionality of the embedding vectors (set to 8 in this case).

The host_neighbourhood_cleaned column is converted to a TensorFlow tensor (neighbourhood_tensor) and passed through the embedding layer to generate dense embeddings.

4) Mapping Ordinal Categories:

```
mapping = {
    'within an hour': 1,
    'within a few hours': 2,
    'within a day': 3,
    'a few days or more': 4
}
df['host_response_time'] = df['host_response_time'].map(mapping)
```

This mapping preserves the **ordinal relationship** between the categories, ensuring that the numerical representation reflects their logical order.

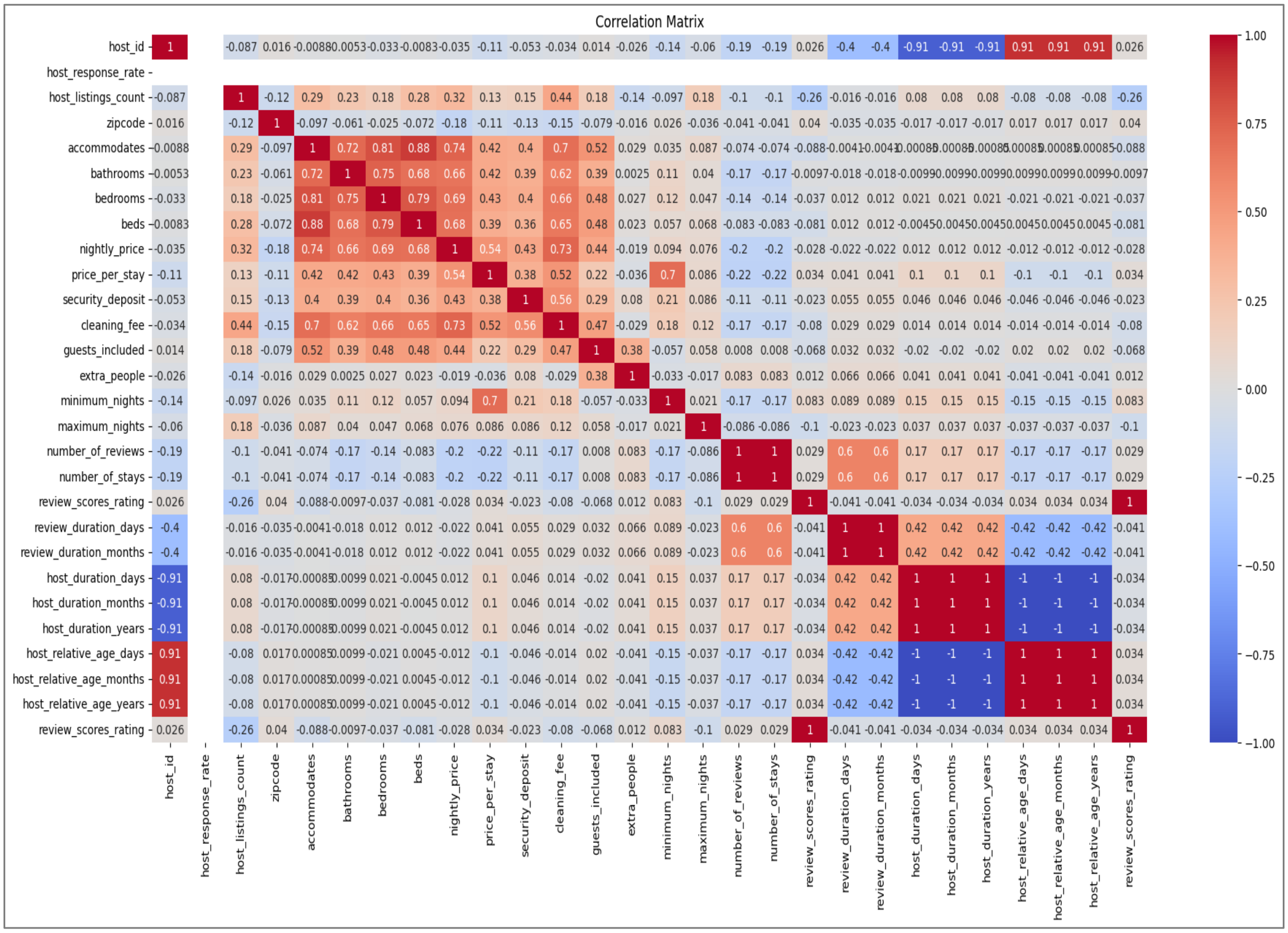
➢ Milestone 1 Feature selection:

1) Correlation Analysis :

- To identify relationships between numerical features and the target variable (review_scores_rating)

- Features with a correlation greater than 0.1 with the target variable are selected for further analysis.

- **Selected columns: ['host_listings_count', 'accommodates', 'beds', 'cleaning_fee', 'minimum_nights', 'maximum_nights', 'review_scores_rating'].**



2) Missing Value Imputation (ANOVA):

Missing values in numerical columns are imputed using the KNN algorithm, which estimates missing values based on the nearest neighbors.

3) primary selection:

```
X = df[Numerical_cols]
y = df['review_scores_rating']
selector = SelectKBest(score_func=mutual_info_regression, k=10)
X_new = selector.fit_transform(X, y)
selected_features = X.columns[selector.get_support()]
print("Selected Top 10 Features:")
for feature in selected_features:
    print("-", feature)
```

SelectKBest with mutual_info_regression is used to select the top ten features based on their relevance to the target variable.

4) Text Feature Extraction:

Textual columns are vectorized using TfidfVectorizer to convert them into numerical representations.
The resulting sparse matrices are combined into a single feature matrix.

5) Feature Selection for Text Features:

Identify the most important text-based features using ANOVA F-test.
SelectKBest with f_classif is applied to select the top 100 text features.
The importance of each feature is evaluated using F-scores and p-values.
Output :
Source columns contributing the most selected features are identified.
Example: host_about_cleaned, house_rules_cleaned, etc.

6) Interaction Features:

Create new features by combining existing numerical and binary features to capture interactions.

```
listing_features_df['accommodates_per_bed'] =
listing_features_df['accommodates'] / (listing_features_df['beds'] + 1e-9)

listing_features_df['has_kitchen_and_iron'] =
(listing_features_df['has_kitchen'] == 1) &
(listing_features_df['has_iron'] == 1)

listing_features_df['has_kitchen_and_laptop_workspace'] =
(listing_features_df['has_kitchen'] == 1) &
(listing_features_df['has_laptop_friendly_workspace'] == 1)
```

7) Polynomial Features:

Generate polynomial combinations of numerical features to capture non-linear relationships.

Explanation :

Polynomial features up to degree 2 are generated for numerical columns. These features are added to the main DataFrame for further modeling.

8) Encoding Categorical Variables:

Categorical columns are encoded using LabelEncoder to prepare them for machine learning models.

9) Aggregating Features:

{ Group-level statistics (e.g., average cleaning fee per property type) are calculated and added as new features. }

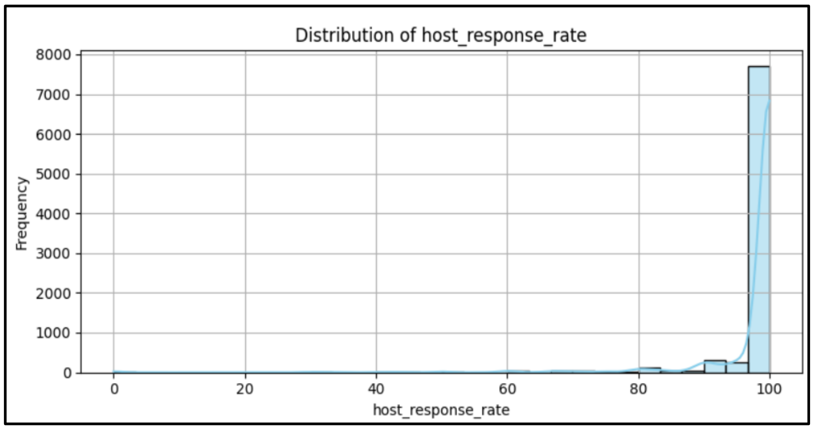
```
avg_cleaning_fee_by_property_type =
listing_features_df.groupby('property_type_cleaned')
['cleaning_fee'].transform('mean')
listing_features_df['avg_cleaning_fee_per_property_type'] =
avg_cleaning_fee_by_property_type.iloc[:, 0]
```

➢ Visualizations

1) Distributions:

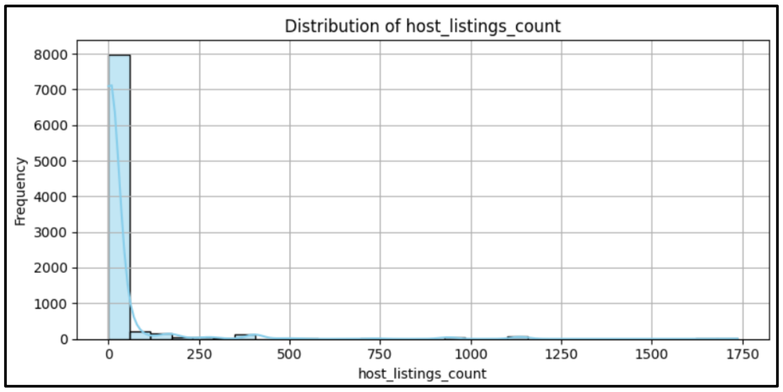
- **host_response_rate**

Distribution: Mostly centered near 100%.
Outliers: Some hosts have lower response rates (~0–40%).
Skewness: Left-skewed (more high values).
Interpretation: Most hosts respond quickly.



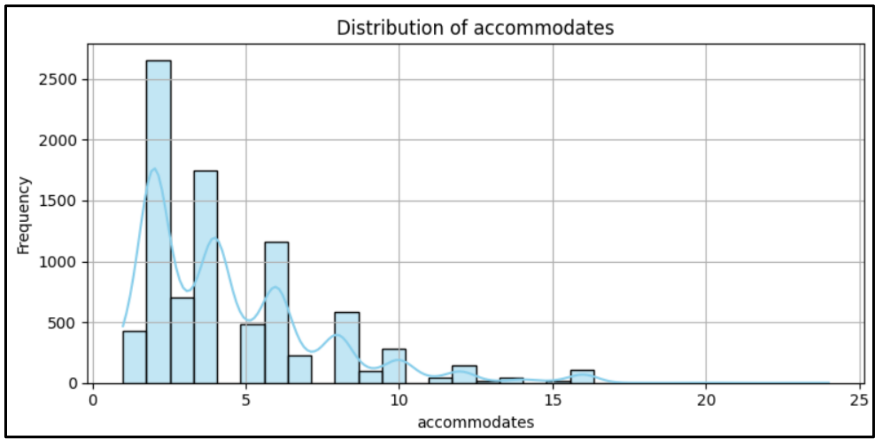
- **host_listings_count**

Distribution: Right-skewed (most hosts have few listings).
Outliers: Hosts with a very high number of listings (superhosts)
Skewness: Right-skewed.
Interpretation: Majority of hosts manage few listings, but a small number manage many.



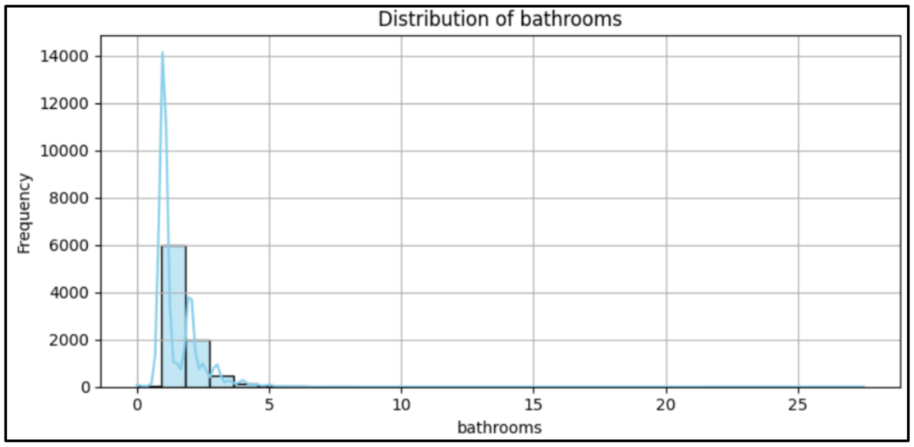
- **accommodates**

Distribution: Peaks around 2–4 guests.
Outliers: Listings for very large groups (>10 guests).
Skewness: Right-skewed.
Interpretation: Most listings accommodate small groups; large group accommodations are rare but could be premium offering



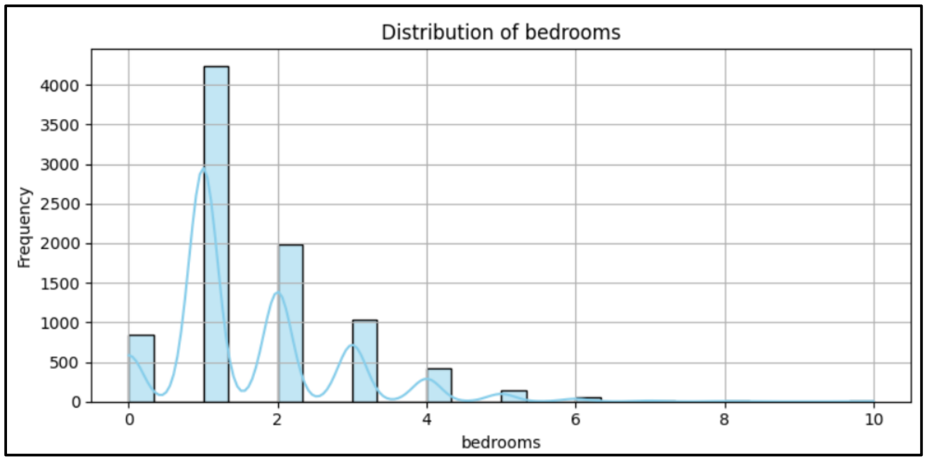
- **bathrooms**

Distribution: Peaks at 1 bathroom.
Outliers: Listings with many bathrooms (> 4).
Skewness: Right-skewed.
Interpretation: Typical listings have 1–2 bathrooms; luxury listings have more.



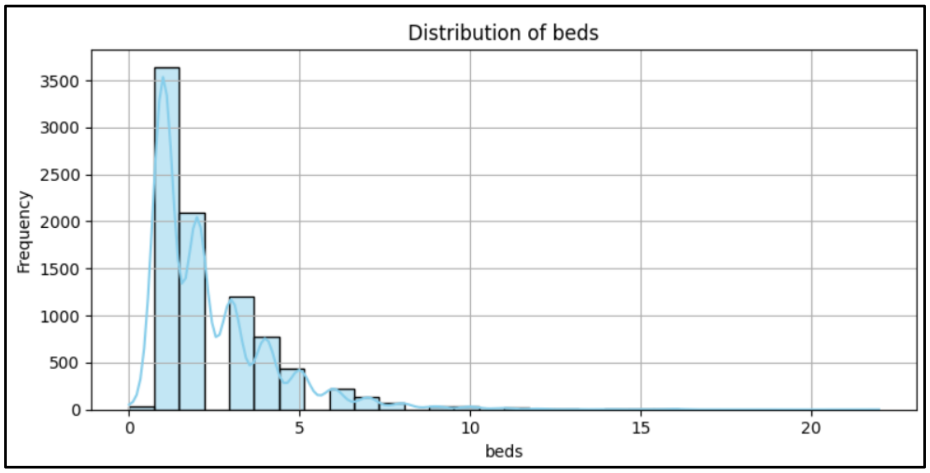
- **bedrooms**

Distribution: Peaks at 1–2 bedrooms.
Outliers: Listings with >5 bedrooms.
Skewness: Right-skewed.
Interpretation: Most properties are small apartments or houses; big listings are rare and more likely to be luxury.



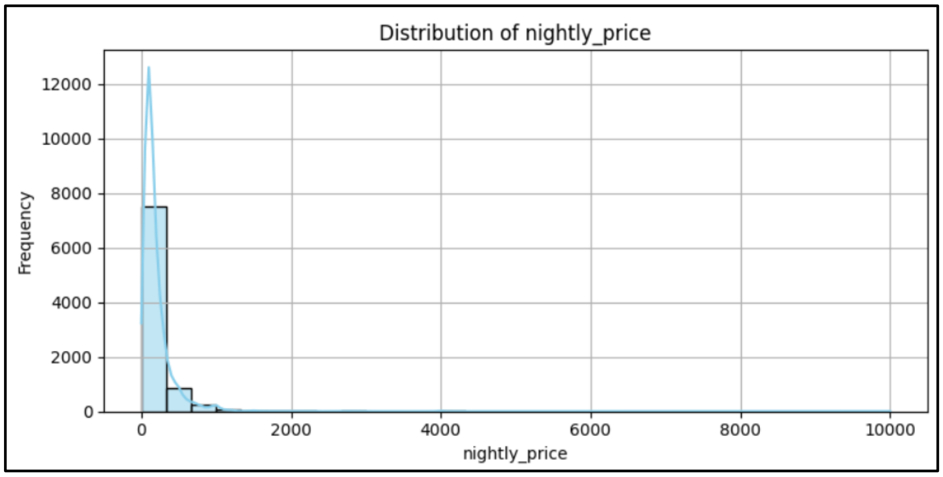
- **beds**

Distribution: Centered around 1–3 beds.
Outliers: Listings with > 5 beds.
Skewness: Right-skewed.
Interpretation: Listings generally offer 1–3 beds; large-capacity listings are less frequent.



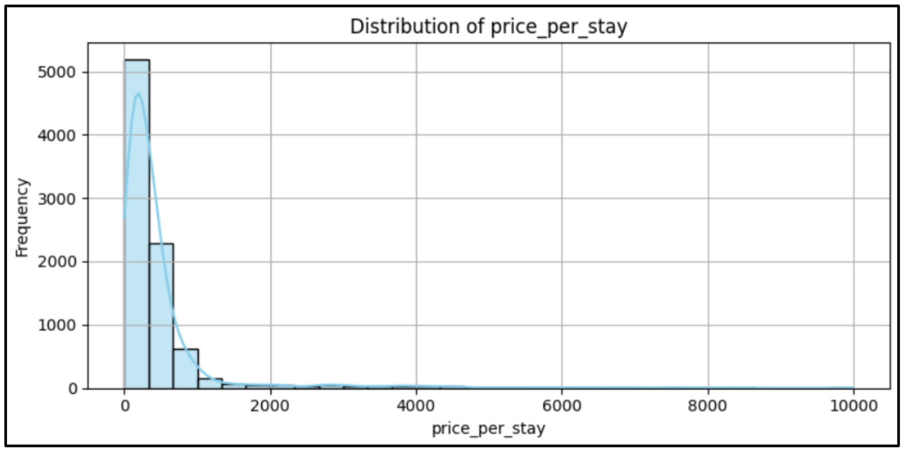
- **nightly_price**

Distribution: Highly right-skewed.
Outliers: Listings priced extremely high per night.
Skewness: Right-skewed.
Interpretation: Most properties are affordable, but luxury properties (outliers) are priced much higher.



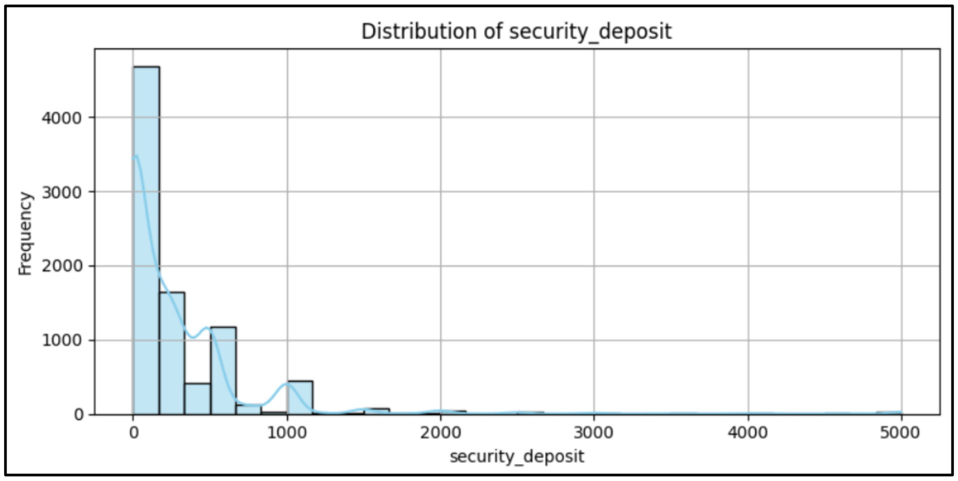
- **price_per_stay**

Distribution: Right-skewed.
Outliers: Extremely high total stay prices.
Skewness: Right-skewed.
Interpretation: Reflects nightly price multiplied by the stay duration. Highlights how longer stays or luxury accommodations inflate total prices.



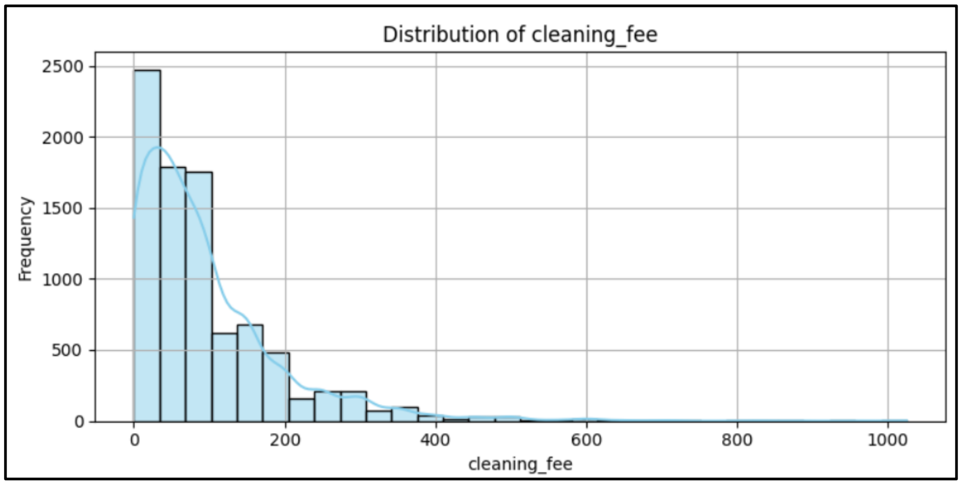
- **security_deposit**

Distribution: Mostly near zero, with some higher values.
Outliers: Listings requiring very high deposits.
Skewness: Strong right-skewed
Interpretation: Most listings require little or no deposit; some high-end listings ask for larger security deposits.



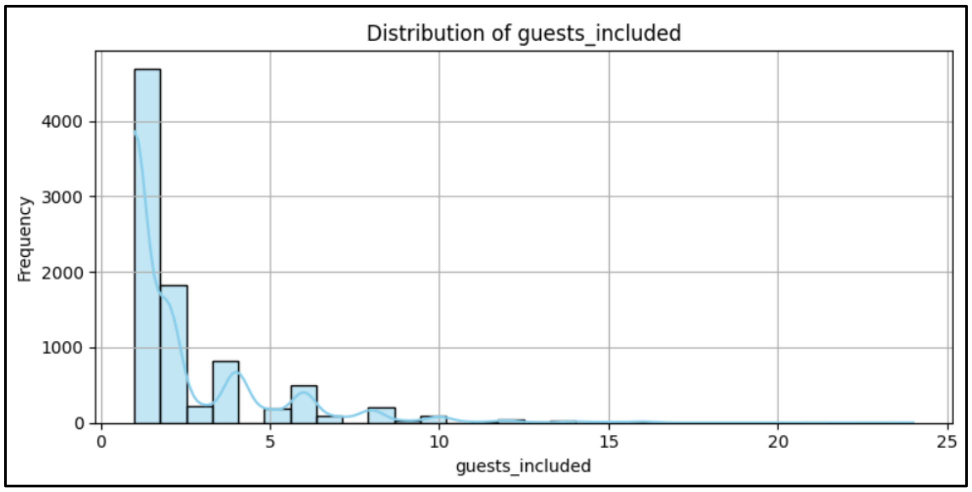
- **cleaning_fee**

Distribution: Right-skewed.
Outliers: Some listings have very high cleaning fees.
Skewness: Right-skewed.
Interpretation: Most cleaning fees are modest; luxury listings or those with complex properties may charge high fees.



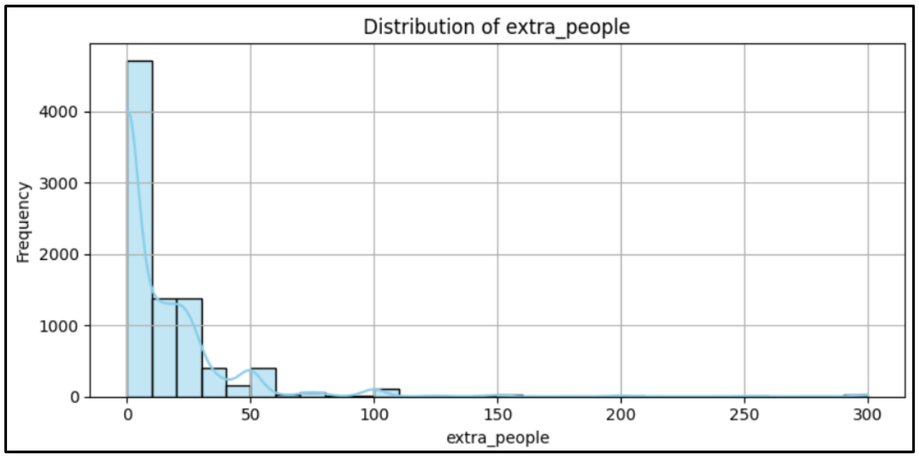
- **guests_included**

  Distribution: Peak at 1–2 guests.
  Outliers: Listings allowing many guests included at base price.
  Skewness: Right-skewed.
  Interpretation: Most listings include a few guests before additional charges apply



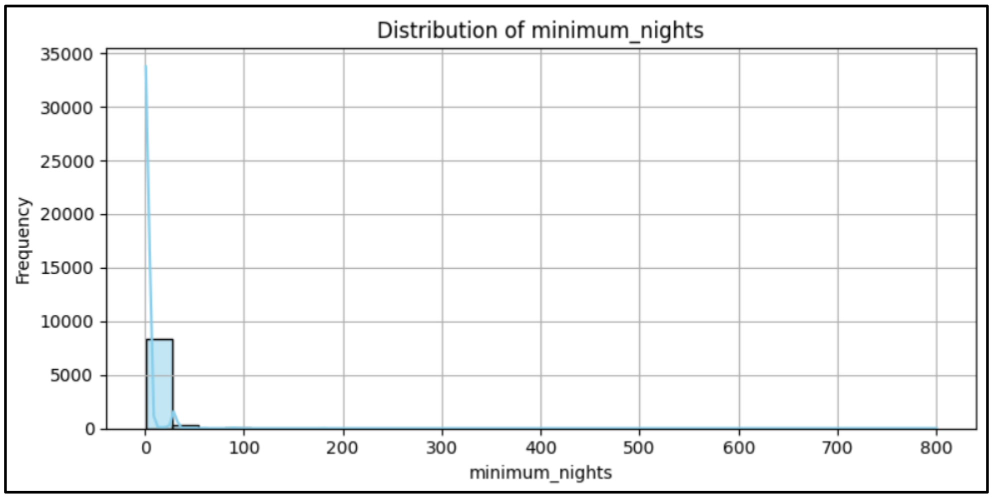
- **extra_people**

  Distribution: Most listings charge little to nothing extra; right-skewed.
  Outliers: Some listings charge very high extra fees per guest.
  Skewness: Right-skewed.
  Interpretation: Extra people fees are generally low but can be significant for larger groups.



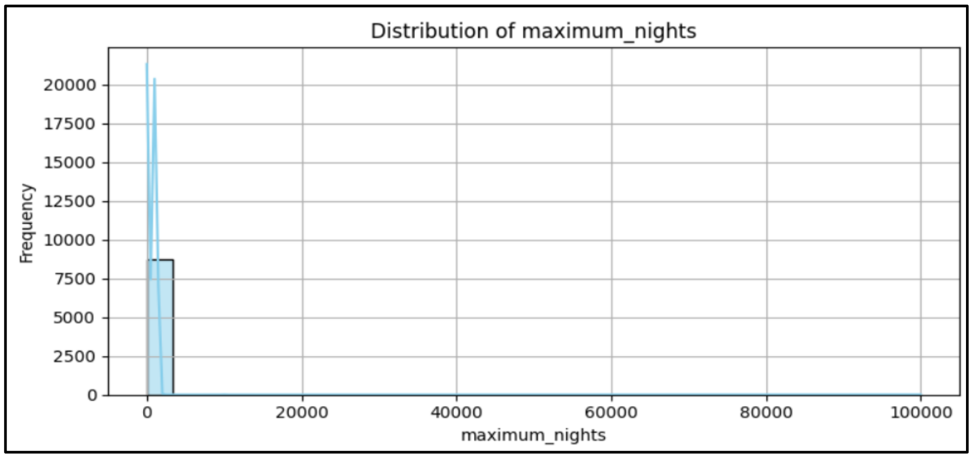
- **minimum_nights**

Distribution: Peaks at 1 night minimum.
Outliers: Peaks at 1 night minimum.
Skewness: Right-skewed.
Interpretation: Most listings allow short stays; some require longer minimums, often for regulatory or preference reasons.



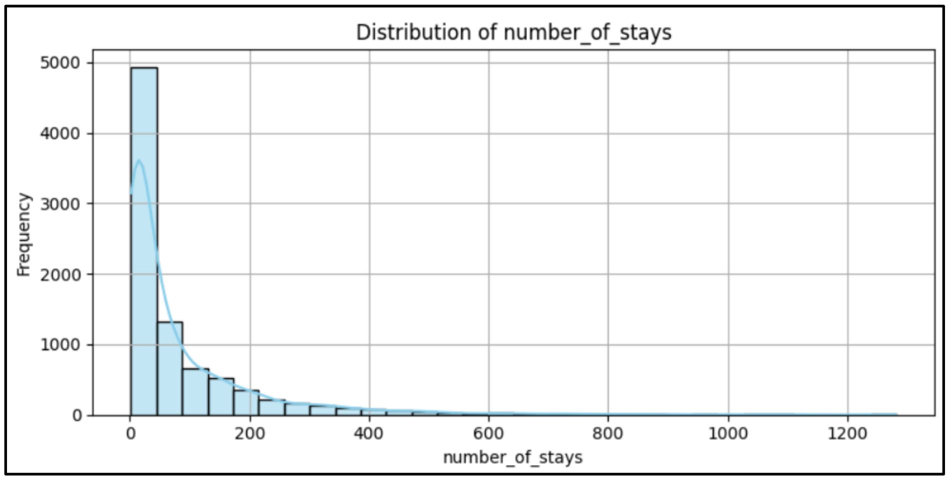
- **maximum_nights**

Distribution: Very wide spread; many very high maximum values.
Outliers: Maximum nights set to 365+, 9999+, etc.
Skewness: Extremely right-skewed.
Interpretation: Some listings set very high maximums as defaults; usually not meaningful.



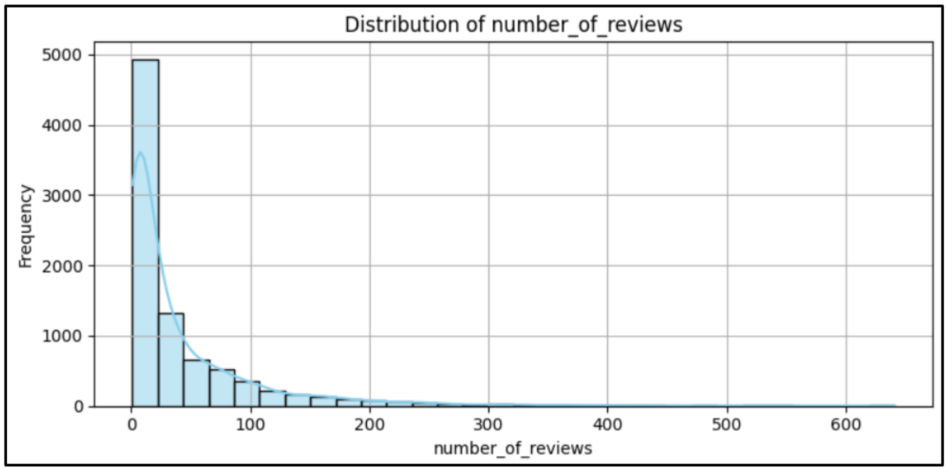
- **number_of_stays**

Distribution: Similar to reviews, peaks at lower numbers.
Outliers: Listings with very high numbers of stays.
Skewness: Right-skewed.
Interpretation: Only a few listings have very high turnover.



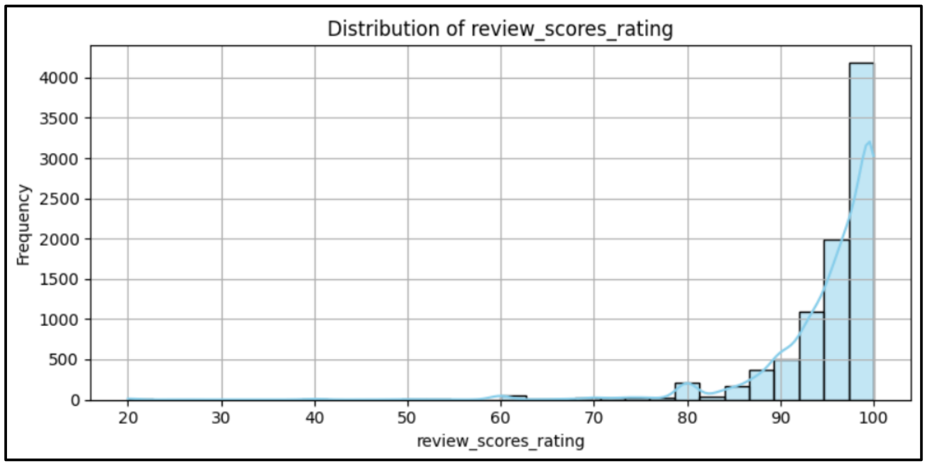
- **number_of_reviews**

Distribution: Peaks at low review counts.
Outliers: Listings with hundreds of reviews.
Skewness: Right-skewed.
Interpretation: Most listings have few reviews; some very popular listings have accumulated hundreds.



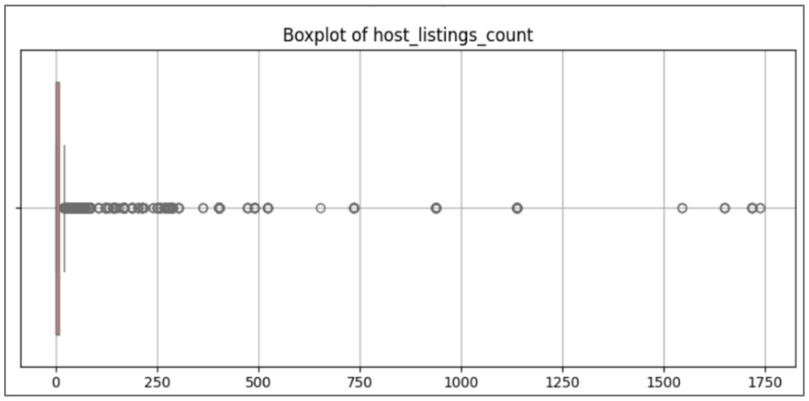
- **review_scores_rating**

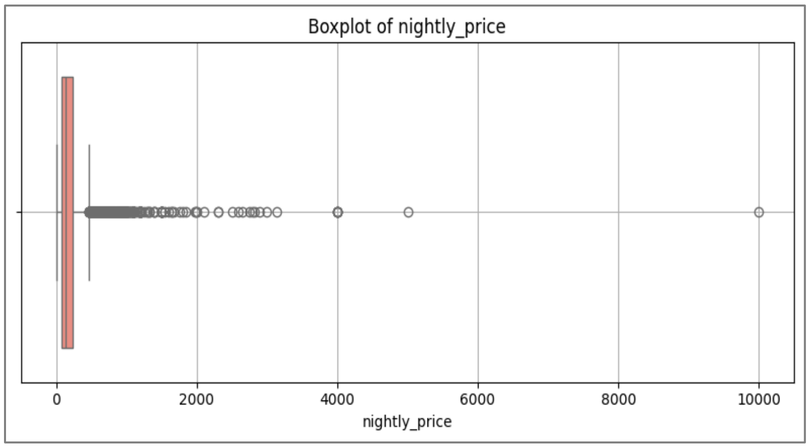
Distribution: Heavily clustered around 4–5 stars (80–100).
Outliers: Very low scores (rare).
Skewness: Left-skewed (most ratings are high).
Interpretation: Airbnb reviews are biased positively; users mostly leave high ratings.

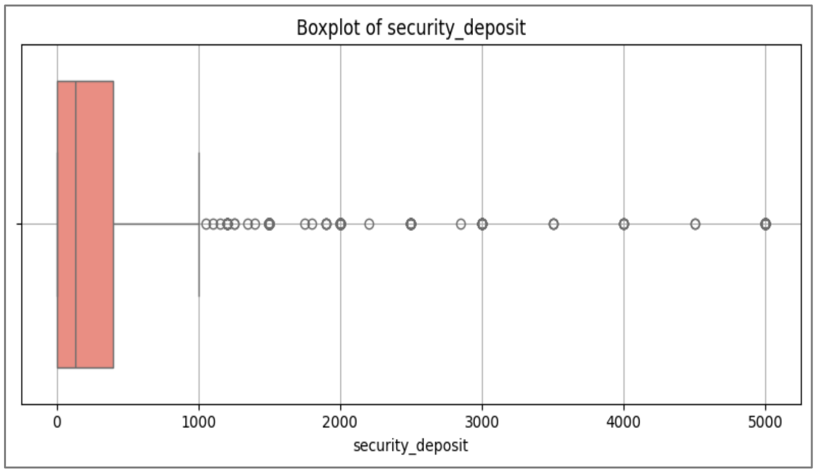


2) Boxplots :

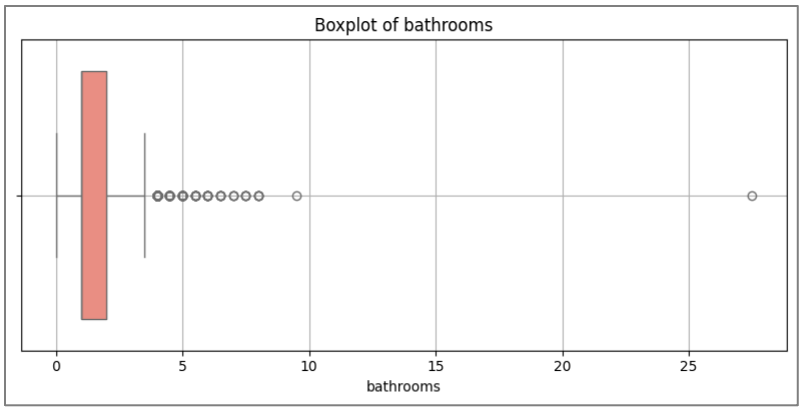
- Features such as **host_listings_count, nightly_price,** and **security_deposit** showed significant presence of outliers, reflecting natural variability in hosts and listing types.

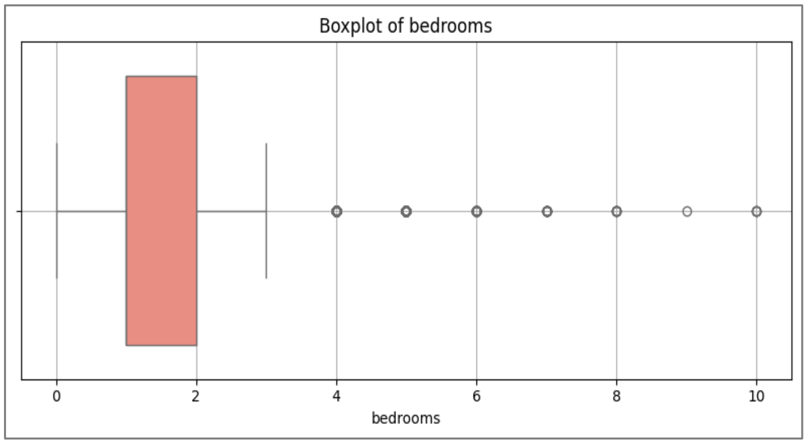


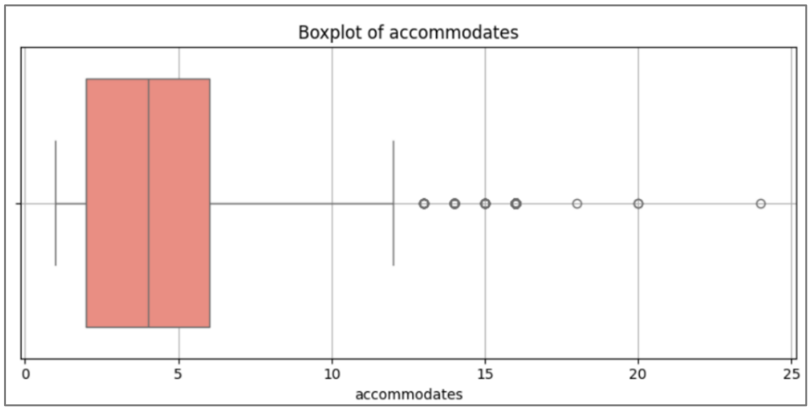




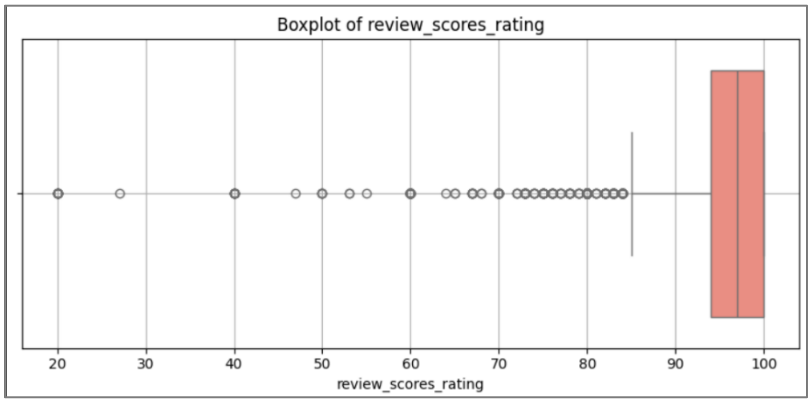
- Several features, including **accommodates, bedrooms,** and **bathrooms**, exhibited slight right-skewness, indicating a concentration of smaller properties with a few larger ones.







- Fields like **review_scores_rating** showed high clustering toward positive values, confirming user bias toward higher ratings.



➢ Handling outliers

- From the previous visualizations we can see that there are outliers in a lot of columns.

- We used the Interquartile Range (IQR) method to detect and cap outliers across numerical features. Capping outliers, instead of removing them, allowed us to maintain the integrity of the dataset while reducing the skewing effect of extreme values.

Column	Type of Outliers	Comments
host_response_rate	Low values (close to 0%)	Some hosts barely respond.
host_listings_count	Very high values (100+ listings)	Power users like companies.
accommodates	Very large values (>10 guests)	Big properties are rare.
bathrooms	Listings with 4+ bathrooms	Luxury/villa properties.
bedrooms	Listings with many bedrooms (>5)	Large homes or villas.
beds	Listings offering many beds	Dormitory-style listings.
nightly_price	Very expensive nightly rates	Luxury or very expensive listings.
price_per_stay	Extremely high total prices	Long stays + high price.
security_deposit	Very large deposits	Luxury properties.
cleaning_fee	Very high cleaning fees	Big villas or luxury properties.
minimum_nights	Listings requiring long minimum stays	30 days+, very rare.
maximum_nights	Extremely high values (9999 days etc.)	Defaults or rare configurations.
number_of_reviews	Properties with hundreds of reviews	Very popular listings.
number_of_stays	Same as above, high booking counts.	

➢ Text Analysis & Preprocessing

1) Amenities Parsing and Categorization:

- Each listing had a messy amenities field.
- We cleaned this field by removing curly brackets and quotes, splitting it into a simple list.
- Then, we mapped each amenity to logical categories like "Basic Necessities," "Comfort/Convenience," "Technology," etc.
- For example, if a listing had "Wi-Fi" and "Heating," we categorized them under "Technology" and "Basic Necessities."

2) Creating Binary Features for Amenities:

- We have created new features that indicate whether each amenity is available or not.
- If a listing has "Wi-Fi," a new column, has_wifi will have a 1. Otherwise, it will be 0.
- This turned the messy text data into clean, machine-readable columns.

3) Text Column Identification and Cleaning:

- We scanned the entire dataset to find real text columns, not numbers.

4) We filtered based on:

- Enough text length
- Enough variation
- Not being numeric

5) Then cleaned these text fields:

- Lowercased all words
- Removed punctuation
- Removed stop words like "the" and "is"
- Lemmatized words (e.g., "running" becomes "run")

6) Dropping Unnecessary Columns:

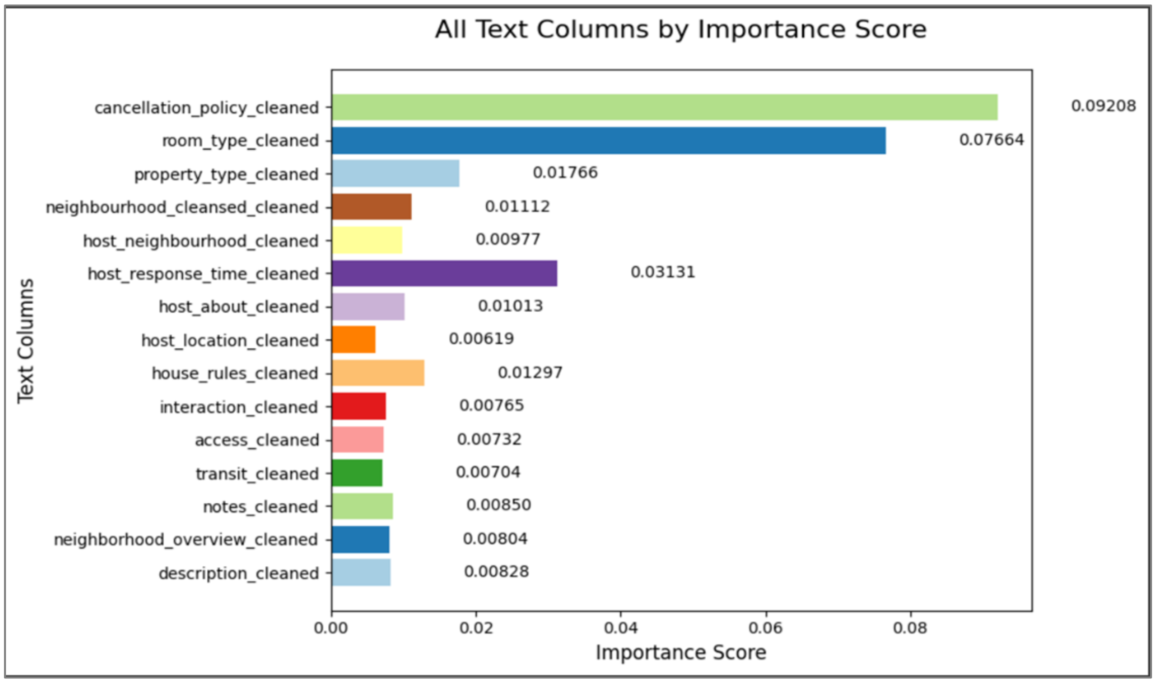
- We removed the original messy text columns after cleaning them.
- Exception: host_response_time was preserved to introducing null values after encoding

7) Analysis of the Importance of Text Fields:

- We calculated how important each cleaned text field is for predicting review_scores_rating.

- We used:
  1. TF-IDF to vectorize words
  2. Variance and correlation with the target to measure influence
  3. Higher scores mean the text is more informative for predicting good or bad reviews.

8) Visualizing Importance Scores:



9) Host Location Analysis

- We checked how many hosts exactly listed their location as "San Diego, California, United States."
- We cleaned the location text first.

➢ Regression Model Trials with Different Selected Features

- For this set of features, we selected variables such as **[host_response_rate, host_is_superhost, host_listings_count, accommodates, bathrooms, bedrooms, beds, number_of_reviews, number_of_stays, review_duration_days, host_duration_days, instant_bookable, and host_identity_verified]**.These features were chosen based on their strong correlation with the target variable (review_scores_rating).

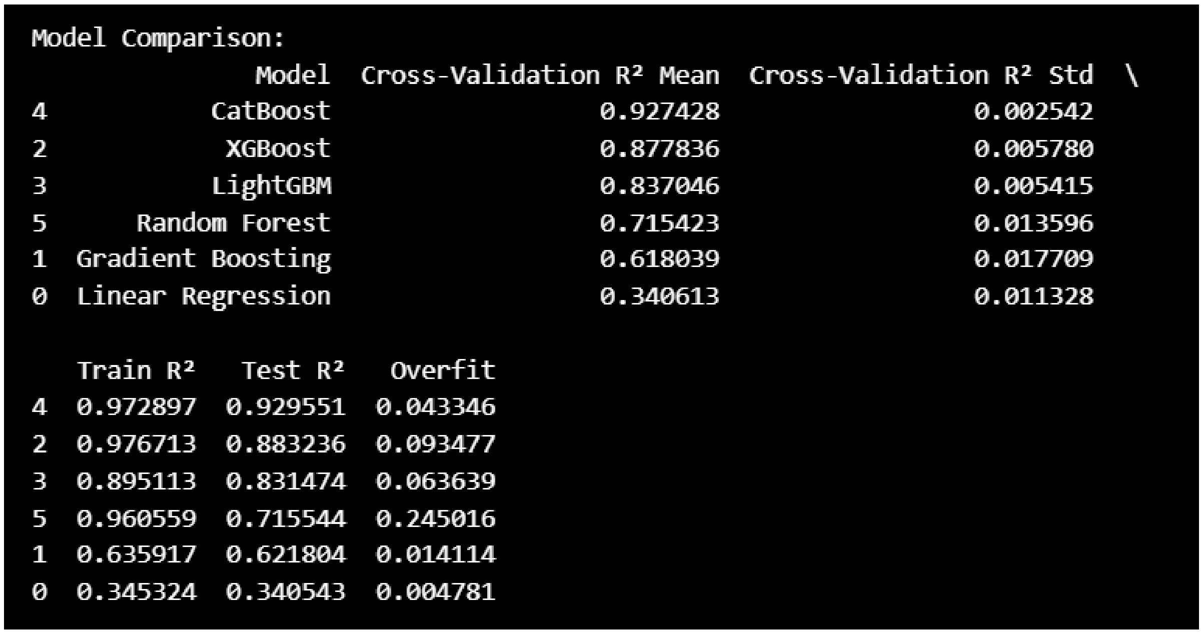
- We prioritized variables that demonstrated the clearest and most consistent relationships with guest satisfaction, aiming to capture the key factors that directly impact review scores.

- We use these features **[host_response_power, host_commitment, bedroom_quality, space_per_guest, essential_amenities, review_consistency]** from feature engineering to make model train better and prevent overfitting.

- The features **room_type_cleaned** and **cancellation_policy_cleaned** were selected based on **ANOVA** tests, as they demonstrated the most significant impact on review scores.

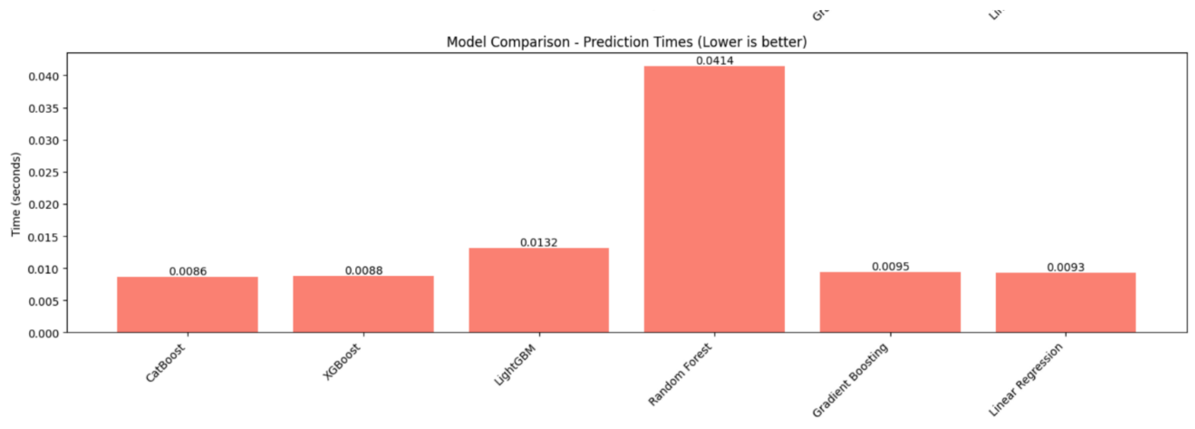
- We also used a Pipeline to automate preprocessing and modeling. Numerical features were imputed with the median and scaled, while categorical features were imputed with the most frequent value and one-hot encoded. We trained a Gradient Boosting model with early stopping to prevent overfitting. The model's performance was evaluated using 5-fold cross-validation and a train-test split.We used that because the train-test R^2 was too low and by using Pipeline we improved the model accuracy.

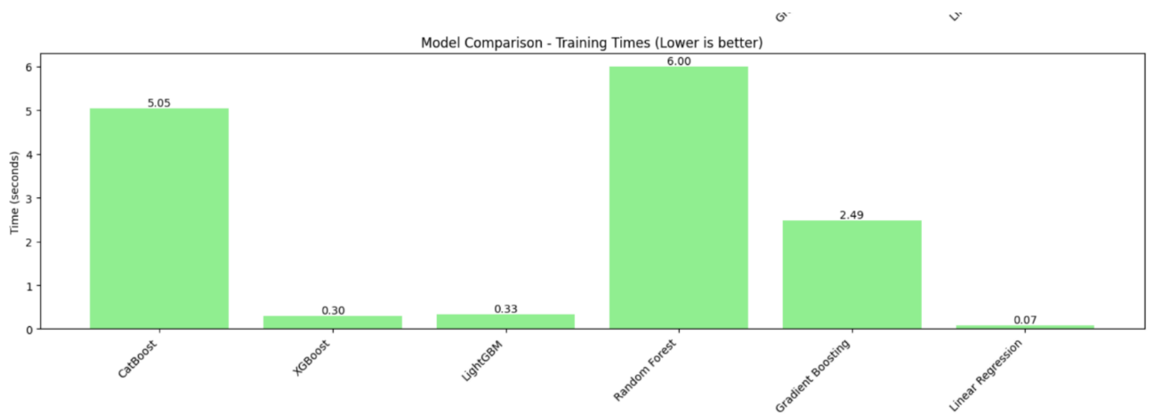
- The following results were obtained by training and evaluating the selected features using various models, such as **Linear Regression, Gradient Boosting, XGBoost, LightGBM, CatBoost, and Random Forest**.

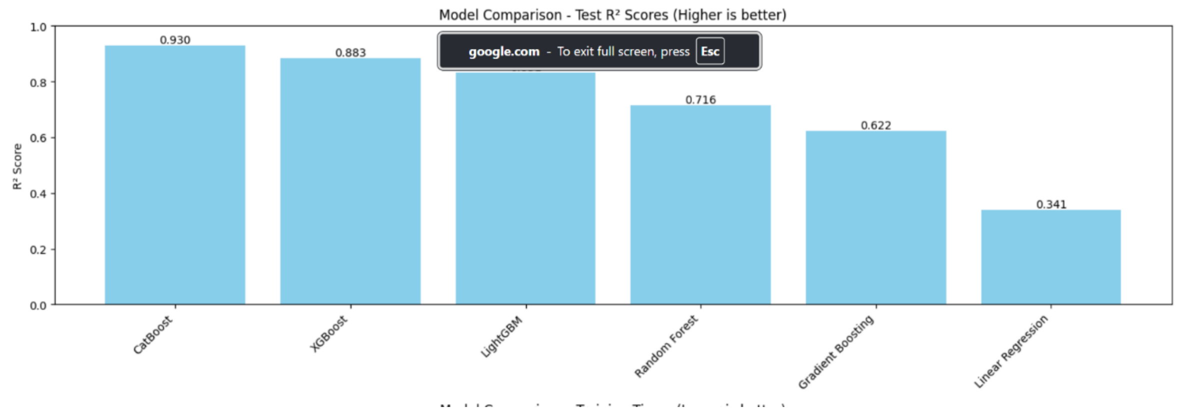


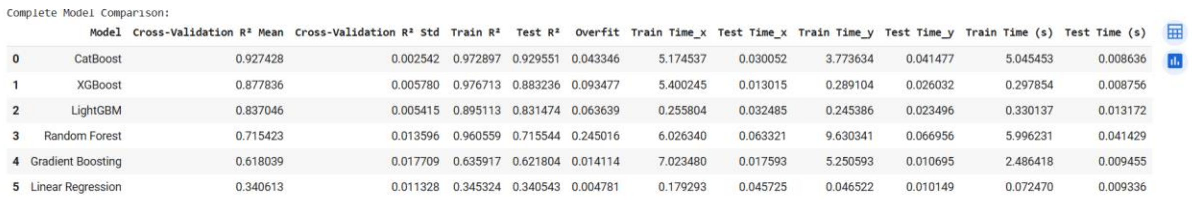
```
Model Comparison:
               Model  Cross-Validation R² Mean  Cross-Validation R² Std  \
4           CatBoost                  0.927428                 0.002542
2            XGBoost                  0.877836                 0.005780
3           LightGBM                  0.837046                 0.005415
5      Random Forest                  0.715423                 0.013596
1  Gradient Boosting                  0.618039                 0.017709
0  Linear Regression                  0.340613                 0.011328

   Train R²   Test R²   Overfit
4  0.972897  0.929551  0.043346
2  0.976713  0.883236  0.093477
3  0.895113  0.831474  0.063639
5  0.960559  0.715544  0.245016
1  0.635917  0.621804  0.014114
0  0.345324  0.340543  0.004781
```

➢ Some Graphs for Each Regression Model Performance








Complete Model Comparison:

	Model	Cross-Validation R² Mean	Cross-Validation R² Std	Train R²	Test R²	Overfit	Train Time_x	Test Time_x	Train Time_y	Test Time_y	Train Time (s)	Test Time (s)
0	CatBoost	0.927428	0.002542	0.972897	0.929551	0.043346	5.174537	0.030052	3.773634	0.041477	5.045453	0.008636
1	XGBoost	0.877836	0.005780	0.976713	0.883236	0.093477	5.400245	0.013015	0.289104	0.026032	0.297854	0.008756
2	LightGBM	0.837046	0.005415	0.895113	0.831474	0.063639	0.255804	0.032485	0.245386	0.023496	0.330137	0.013172
3	Random Forest	0.715423	0.013596	0.960559	0.715544	0.245016	6.026340	0.063321	9.630341	0.066956	5.996231	0.041429
4	Gradient Boosting	0.618039	0.017709	0.635917	0.621804	0.014114	7.023480	0.017593	5.250593	0.010695	2.486418	0.009455
5	Linear Regression	0.340613	0.011328	0.345324	0.340543	0.004781	0.179293	0.045725	0.046522	0.010149	0.072470	0.009336

➢ Model Performance Conclusion

After evaluating all models through cross-validation and comparing their training and testing R² scores, a few key insights emerged:

- **CatBoost** delivered the best overall performance, achieving the highest cross-validation R² score (0.8867) and the strongest test R² (0.8922) with minimal overfitting. This suggests it generalizes very well to unseen data.

- **XGBoost** and **LightGBM** also performed strongly, with high R² values on both training and testing sets.

- **Random Forest** performed decently, but the overfitting was much higher (Train R²: 0.9605 vs. Test R²: 0.7159), meaning it captured the training patterns very well but struggled more on new data.

- **Linear Regression** had moderate performances, with low overfitting but relatively low R² scores, meaning they may not be capturing the complexity of the relationships well enough.

➢ In summary:

CatBoost was the best model, closely followed by **XGBoost** and **LightGBM**. These tree-based ensemble models handle the complexity of the dataset better than linear or simpler models, while maintaining a good balance between bias and variance.

➢ Classification Feature Selection

- **Chi-square**
  We selected categorical features and applied the Chi-square test to evaluate their dependency with the target variable. After encoding both features (via one-hot encoding) and the target (via label encoding), we ranked the features by their Chi-square scores. The top features were selected for further analysis/modeling based on their statistical significance.

  Selected features = ['cancellation_policy', 'host_about', 'host_is_superhost', 'house_rules', 'neighbourhood_cleansed', 'property_type', 'transit']

- **ANOVA**
  We applied the ANOVA F-test to identify numerical features that have a statistically significant relationship with the target variable. After cleaning the data, imputing missing values, and removing constant columns, we computed F-statistics and p-values for each feature. Features were then ranked, and the top 10 were selected. Those with p-values less than 0.05 were considered statistically significant and are prioritized for inclusion in the final model.

  Selected features = ['host_listings_count', 'number_of_stays', 'number_of_reviews', 'has_lock_on_bedroom_door', 'has_hot_water', 'maximum_nights','minimum_nights','has_laptop_friendly_workspace', 'accommodates', 'has_air_conditioning']

➢ Classification Model Trial with Different Selected Features

- **Selected Features:**

```
numeric_features = [
    'host_response_rate', 'host_is_superhost', 'accommodates',
    'bathrooms', 'bedrooms', 'beds', 'nightly_price',
    'cleaning_fee', 'number_of_reviews', 'minimum_nights',
    'host_listings_count', 'number_of_stays', 'security_deposit',
    'extra_people', 'maximum_nights', 'guests_included'
]
categorical_features = [
    'neighbourhood_cleansed', 'instant_bookable',
    'cancellation_policy', 'property_type', 'room_type'
]
amenity_features = [
    'has_wifi', 'has_air_conditioning', 'has_kitchen',
    'has_heating', 'has_tv', 'has_free_parking_on_premises',
    'has_iron', 'has_laptop_friendly_workspace'
]
```

• Models and Hyper parameters

Model	Hyperparameters	Train Acc	CV Acc	Test Acc	Train Time (s)	Test Time (s)
Random Forest	max_depth=20	0.9943	0.5971	0.6126	5.50	0.07
	min_samples_split=10	0.9069	0.5979	0.6166	4.12	0.08
	n_estimators=300	0.9990	0.5958	0.6080	22.09	0.21
XGBoost	learning_rate=0.1	0.7444	0.5908	0.6201	0.79	0.04
	max_depth=3	0.6694	0.5903	0.6189	0.38	0.03
	learning_rate=0.01	0.6429	0.5928	0.6069	1.01	0.04
Gradient Boosting	n_estimators=100	0.6474	0.5985	0.6086	5.52	0.03
	max_depth=3	0.6474	0.5985	0.6086	6.23	0.05
	n_estimators=200	0.6756	0.5923	0.6052	11.42	0.07
Logistic Regression	C=0.1	0.5872	0.5743	0.5782	0.51	0.02
	max_iter=100	0.5934	0.5727	0.5765	0.49	0.02
	solver=lbfgs	0.5934	0.5727	0.5765	1.13	0.02
SVM	C=1.0	0.6275	0.5918	0.5960	39.45	1.51
	kernel=rbf	0.6275	0.5918	0.5960	39.91	1.52
	gamma=scale	0.6275	0.5918	0.5960	39.51	1.90

• Observations

Among the evaluated classification models**, Random Forest, XGBoost, and Gradient Boosting** consistently delivered the highest cross-validation and test accuracies, demonstrating their effectiveness on this dataset.
Random Forest achieved **the best balance of accuracy and training time**, with hyperparameters such as max_depth=20 and min_samples_split=10 improving generalization without overfitting.

XGBoost showed competitive accuracy, benefiting from careful tuning of learning_rate and max_depth, though it had generally faster training times than Random Forest.

Gradient Boosting also performed well, with optimal n_estimators and max_depth values enhancing predictive performance.

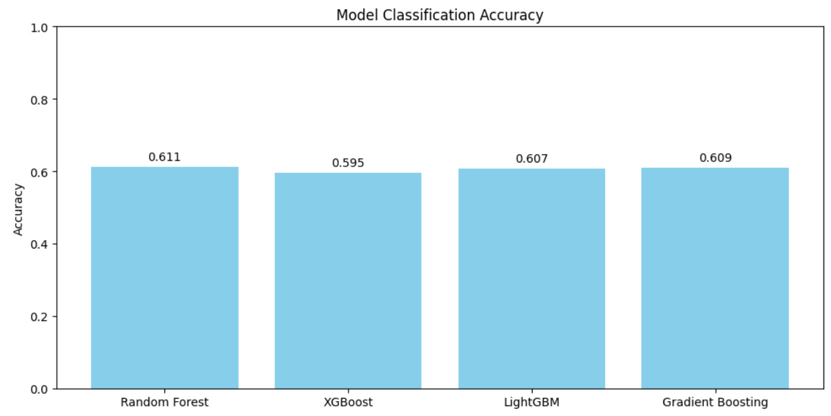
On the other hand**, Logistic Regression and SVM models**, while faster to train and test, **lagged in accuracy**, indicating that their simpler decision boundaries might not capture the complexity of the data as effectively.

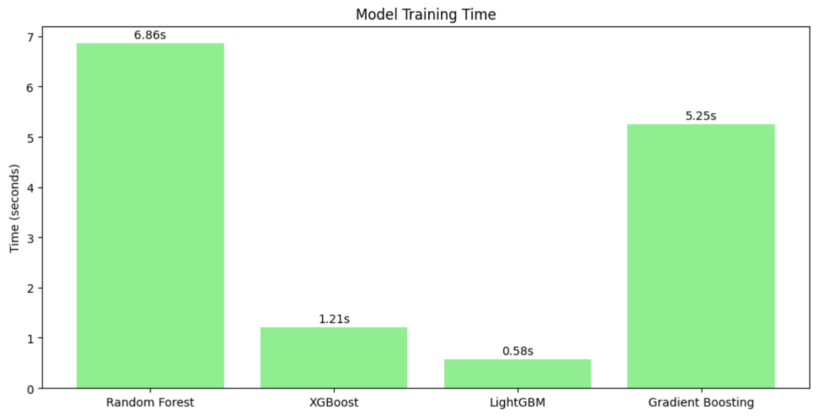
- **Conclusion**
  Overall, ensemble tree-based methods are preferred for this task due to their superior accuracy and reasonable computational cost. Fine-tuning hyperparameters notably improved performance across models, underscoring the importance of systematic hyperparameter search in classification problems.

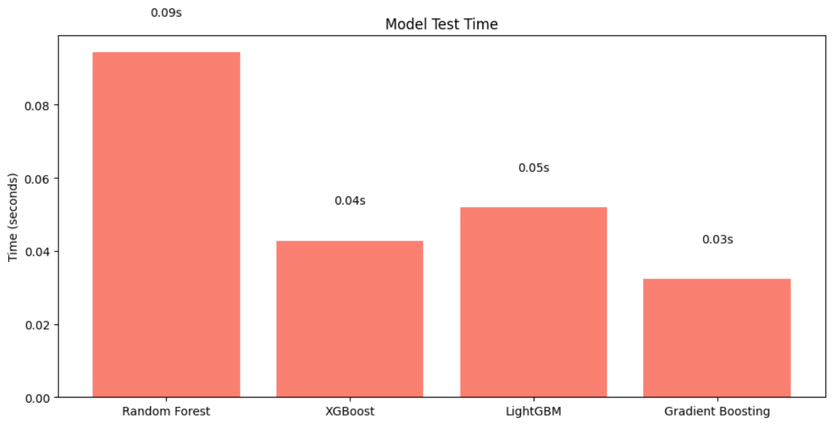
  The best model is XGBoost.

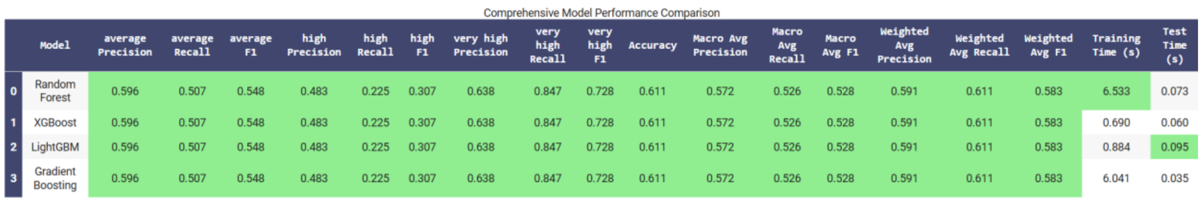
  Why XGBoost?
- High predictive accuracy
- Efficient handling of missing data
- Robustness to overfitting due to regularization
- Fast training with parallel processing
- Works well on structured/tabular data

➢ Some Graphs for Each Classification Model Performance








Comprehensive Model Performance Comparison

	Model	average Precision	average Recall	average F1	high Precision	high Recall	high F1	very high Precision	very high Recall	very high F1	Accuracy	Macro Avg Precision	Macro Avg Recall	Macro Avg F1	Weighted Avg Precision	Weighted Avg Recall	Weighted Avg F1	Training Time (s)	Test Time (s)
0	Random Forest	0.596	0.507	0.548	0.483	0.225	0.307	0.638	0.847	0.728	0.611	0.572	0.526	0.528	0.591	0.611	0.583	6.533	0.073
1	XGBoost	0.596	0.507	0.548	0.483	0.225	0.307	0.638	0.847	0.728	0.611	0.572	0.526	0.528	0.591	0.611	0.583	0.690	0.060
2	LightGBM	0.596	0.507	0.548	0.483	0.225	0.307	0.638	0.847	0.728	0.611	0.572	0.526	0.528	0.591	0.611	0.583	0.884	0.095
3	Gradient Boosting	0.596	0.507	0.548	0.483	0.225	0.307	0.638	0.847	0.728	0.611	0.572	0.526	0.528	0.591	0.611	0.583	6.041	0.035

➢ Conclusion

Ensemble tree-based models outperformed simpler models in both classification and regression tasks.

- **XGBoost** was the best Classification model
- Balanced accuracy, speed, and overfitting control
- **CatBoost** was the best Regression model
- Models like Logistic Regression, SVM, and Linear Regression performed faster but could not match the predictive power of ensemble methods.
  .