In [ ]:
```
import pandas as pd
import pickle
import matplotlib.pyplot as plt
import shap

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
import xgboost as xgb
import lightgbm as lgb
from catboost import CatBoostRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# from evidently.report import Report
# from evidently.metrics import RegressionQualityMetric, RegressionErrorPlot, Re
# from evidently.metric_preset import DataDriftPreset, RegressionPreset
```

In [ ]:
```
df = pd.read_csv("/kaggle/input/training-data/training_data (2).csv")
sentiment_data = pd.read_csv("/kaggle/input/senti-data/senti_data.csv")
```

In [ ]:
```
# Initialize a dictionary to store label encoders
label_encoders = {}

# Create a copy of the original DataFrame to preserve the original data
encoded_df = df.copy()

# List of categorical columns to be label encoded
categorical_columns = ['property_type', 'room_type', 'bed_type',
                       'cancellation_policy', 'city',
                       'host_has_profile_pic', 'host_identity_verified',
                       'instant_bookable', 'cleaning_fee']

# Apply label encoding to each categorical column
for column in categorical_columns:
    # Initialize LabelEncoder
    label_encoder = LabelEncoder()

    # Apply label encoding to the column
    encoded_df[column] = label_encoder.fit_transform(encoded_df[column])

    # Store the label encoder object in the dictionary
    label_encoders[column] = label_encoder

# Save the label encoders to a file using pickle
with open('label_encoders.pkl', 'wb') as f:
    pickle.dump(label_encoders, f)
```

In [ ]:
```
# Extracting month from date columns and adding them as new columns
df['first_review_m'] = pd.to_datetime(df['first_review']).dt.month
df['host_since_m'] = pd.to_datetime(df['host_since']).dt.month
df['last_review_m'] = pd.to_datetime(df['last_review']).dt.month

# Selecting columns of interest
int_df = df[['id', 'log_price', 'accommodates', 'bathrooms', 'zipcode','neighbou
             'host_response_rate', 'number_of_reviews', 'review_scores_rating',
```

```
             'bedrooms', 'latitude', 'longitude','beds', 'review_duration', 'tim
             'host_tenure', 'average_review_score', 'amenities_count',
             'has_wireless_internet', 'has_kitchen', 'has_heating', 'has_essenti
             'has_smoke_detector','first_review_m', 'host_since_m', 'last_review
```

In [ ]:
```
label_encoder_senti = LabelEncoder()
sentiment_data['sentiment_category'] = label_encoder_senti.fit_transform(sentime

# Save this to labelencoders dict
label_encoders['sentiment_category'] = label_encoder_senti
```

In [ ]:
```
Model_training_df = pd.concat([int_df, encoded_df[categorical_columns],sentiment
```

In [ ]:
```
X = Model_training_df.drop("log_price", axis=1)

y = Model_training_df['log_price']
```

In [ ]:
```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Initialize regression models
models = {
    'linear_regression': LinearRegression(),
    'ridge_regression': Ridge(),
    'lasso_regression': Lasso(),
    'elastic_net_regression': ElasticNet(),
    'decision_tree_regression': DecisionTreeRegressor(),
    'random_forest_regression': RandomForestRegressor(),
    'gradient_boosting_regression': GradientBoostingRegressor(),
    'xgboost_regression': xgb.XGBRegressor(),
    'lightgbm_regression': lgb.LGBMRegressor(),
    'catboost_regression': CatBoostRegressor(silent=True)
}

# Train each model
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Calculate Mean Squared Error (MSE)
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    # Print the evaluation metric
    print(f"{name} MSE: {mse}")
    print(f"{name} MSE: {mae}")

    print("-"*30)


# Save the models dictionary to a pickle file
with open('trained_models.pkl', 'wb') as f:
    pickle.dump(models, f)
```

```
linear_regression MSE: 0.2345703679069435
linear_regression MSE: 0.36350034347637267
------------------------------
ridge_regression MSE: 0.23457080577368813
ridge_regression MSE: 0.3635002848955916
------------------------------
lasso_regression MSE: 0.5059901469503172
lasso_regression MSE: 0.5550222372563071
------------------------------
elastic_net_regression MSE: 0.3999987876578443
elastic_net_regression MSE: 0.48878860969443066
------------------------------
decision_tree_regression MSE: 0.318172686448424
decision_tree_regression MSE: 0.40878492734756794
------------------------------
```

```
KeyboardInterrupt
```

In [ ]:

```
# Load the trained models from the pickle file
with open('trained_models.pkl', 'rb') as f:
    models = pickle.load(f)

# Initialize dictionaries to store evaluation metrics for each model
evaluation_metrics = {
    'Model': [],
    'MSE': [],
    'MAE': [],
    'R2': []
}

# Evaluate the performance of each model
for name, model in models.items():
    # Predict on the test data
    y_pred = model.predict(X_test)

    # Calculate evaluation metrics
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Store evaluation metrics in the dictionary
    evaluation_metrics['Model'].append(name)
    evaluation_metrics['MSE'].append(mse)
    evaluation_metrics['MAE'].append(mae)
    evaluation_metrics['R2'].append(r2)
```

From evaluvation we came to know XGBoost, LightBoost, Catboost performed well hence hyperparameter tuning those

In [ ]:

```
# Initialize XGBoost Regression with hyperparameters
xgb_reg = xgb.XGBRegressor(
    n_estimators=100,
    max_depth=3,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
```

```
    objective='reg:squarederror',
    random_state=42
)

# Initialize LightGBM Regression with hyperparameters
lgb_reg = lgb.LGBMRegressor(
    n_estimators=100,
    max_depth=3,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    objective='regression',
    random_state=42
)

# Initialize CatBoost Regression with hyperparameters
catboost_reg = CatBoostRegressor(
    n_estimators=100,
    max_depth=6,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bylevel=0.8,
    objective='MAE',
    random_seed=42
)

# Train XGBoost Regression
xgb_reg.fit(X_train, y_train)

# Train LightGBM Regression
lgb_reg.fit(X_train, y_train)

# Train CatBoost Regression
catboost_reg.fit(X_train, y_train)

# Initialize a dictionary to store models, evaluation metrics, and feature impor
trained_models = {
    'XGBoost': {'model': xgb_reg, 'evaluation_metrics': {}, 'feature_importance'
    'LightGBM': {'model': lgb_reg, 'evaluation_metrics': {}, 'feature_importance
    'CatBoost': {'model': catboost_reg, 'evaluation_metrics': {}, 'feature_impor
}

# Evaluate each model and store evaluation metrics
for model_name, model_data in trained_models.items():
    model = model_data['model']
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    model_data['evaluation_metrics'] = {'MSE': mse, 'MAE': mae, 'R2': r2}

# Store feature importance for XGBoost and LightGBM models
trained_models['XGBoost']['feature_importance'] = xgb_reg.feature_importances_
trained_models['LightGBM']['feature_importance'] = lgb_reg.feature_importances_

# CatBoost provides feature importance as part of its model object
trained_models['CatBoost']['feature_importance'] = catboost_reg.get_feature_impo

# Save the trained models dictionary to a pickle file
```

```
with open('trained_models_with_metrics.pkl', 'wb') as f:
    pickle.dump(trained_models, f)
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.057669 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2823
[LightGBM] [Info] Number of data points in the train set: 59288, number of used f
eatures: 35
[LightGBM] [Info] Start training from score 4.780538
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
0:      learn: 0.5283770        total: 19.4ms   remaining: 1.92s
1:      learn: 0.5005708        total: 38.6ms   remaining: 1.89s
2:      learn: 0.4777701        total: 54.9ms   remaining: 1.77s
3:      learn: 0.4573508        total: 71ms     remaining: 1.71s
4:      learn: 0.4394896        total: 87.8ms   remaining: 1.67s
5:      learn: 0.4254715        total: 105ms    remaining: 1.64s
6:      learn: 0.4120376        total: 121ms    remaining: 1.61s
7:      learn: 0.4006176        total: 138ms    remaining: 1.58s
8:      learn: 0.3912509        total: 155ms    remaining: 1.57s
9:      learn: 0.3837547        total: 172ms    remaining: 1.54s
10:     learn: 0.3763976        total: 187ms    remaining: 1.51s
11:     learn: 0.3699939        total: 204ms    remaining: 1.49s
12:     learn: 0.3645251        total: 221ms    remaining: 1.48s
13:     learn: 0.3592366        total: 238ms    remaining: 1.46s
14:     learn: 0.3551481        total: 254ms    remaining: 1.44s
15:     learn: 0.3514148        total: 270ms    remaining: 1.42s
16:     learn: 0.3474932        total: 286ms    remaining: 1.4s
17:     learn: 0.3448925        total: 302ms    remaining: 1.37s
18:     learn: 0.3426493        total: 318ms    remaining: 1.35s
19:     learn: 0.3398243        total: 334ms    remaining: 1.34s
20:     learn: 0.3372497        total: 351ms    remaining: 1.32s
21:     learn: 0.3350543        total: 367ms    remaining: 1.3s
22:     learn: 0.3326743        total: 383ms    remaining: 1.28s
23:     learn: 0.3305303        total: 400ms    remaining: 1.26s
24:     learn: 0.3289850        total: 415ms    remaining: 1.25s
25:     learn: 0.3274024        total: 434ms    remaining: 1.23s
26:     learn: 0.3260330        total: 450ms    remaining: 1.22s
27:     learn: 0.3245224        total: 466ms    remaining: 1.2s
28:     learn: 0.3233772        total: 482ms    remaining: 1.18s
29:     learn: 0.3220635        total: 498ms    remaining: 1.16s
30:     learn: 0.3211448        total: 514ms    remaining: 1.14s
31:     learn: 0.3199108        total: 531ms    remaining: 1.13s
32:     learn: 0.3190772        total: 548ms    remaining: 1.11s
33:     learn: 0.3182923        total: 565ms    remaining: 1.1s
34:     learn: 0.3172507        total: 581ms    remaining: 1.08s
35:     learn: 0.3164421        total: 597ms    remaining: 1.06s
36:     learn: 0.3154776        total: 614ms    remaining: 1.04s
37:     learn: 0.3149131        total: 630ms    remaining: 1.03s
38:     learn: 0.3140155        total: 648ms    remaining: 1.01s
39:     learn: 0.3135432        total: 663ms    remaining: 995ms
40:     learn: 0.3131326        total: 679ms    remaining: 976ms
41:     learn: 0.3123354        total: 694ms    remaining: 959ms
42:     learn: 0.3117445        total: 710ms    remaining: 941ms
43:     learn: 0.3112021        total: 725ms    remaining: 922ms
44:     learn: 0.3103399        total: 742ms    remaining: 906ms
45:     learn: 0.3096059        total: 759ms    remaining: 891ms
46:     learn: 0.3091153        total: 774ms    remaining: 873ms
47:     learn: 0.3084194        total: 790ms    remaining: 856ms
48:     learn: 0.3080107        total: 806ms    remaining: 839ms
49:     learn: 0.3070531        total: 823ms    remaining: 823ms
```

```
50:	learn: 0.3067287	total: 839ms	remaining: 806ms
51:	learn: 0.3061652	total: 859ms	remaining: 793ms
52:	learn: 0.3058116	total: 875ms	remaining: 776ms
53:	learn: 0.3054287	total: 891ms	remaining: 759ms
54:	learn: 0.3050366	total: 906ms	remaining: 742ms
55:	learn: 0.3046971	total: 923ms	remaining: 725ms
56:	learn: 0.3043352	total: 938ms	remaining: 708ms
57:	learn: 0.3039915	total: 954ms	remaining: 691ms
58:	learn: 0.3036730	total: 972ms	remaining: 675ms
59:	learn: 0.3029704	total: 994ms	remaining: 663ms
60:	learn: 0.3027043	total: 1.01s	remaining: 647ms
61:	learn: 0.3024723	total: 1.03s	remaining: 631ms
62:	learn: 0.3020133	total: 1.04s	remaining: 614ms
63:	learn: 0.3016240	total: 1.06s	remaining: 598ms
64:	learn: 0.3013868	total: 1.08s	remaining: 580ms
65:	learn: 0.3010828	total: 1.09s	remaining: 564ms
66:	learn: 0.3008134	total: 1.11s	remaining: 547ms
67:	learn: 0.3004791	total: 1.13s	remaining: 530ms
68:	learn: 0.3002619	total: 1.14s	remaining: 514ms
69:	learn: 0.3000789	total: 1.16s	remaining: 497ms
70:	learn: 0.2997407	total: 1.18s	remaining: 481ms
71:	learn: 0.2991211	total: 1.21s	remaining: 469ms
72:	learn: 0.2987850	total: 1.23s	remaining: 454ms
73:	learn: 0.2985691	total: 1.25s	remaining: 438ms
74:	learn: 0.2983258	total: 1.27s	remaining: 424ms
75:	learn: 0.2981386	total: 1.29s	remaining: 409ms
76:	learn: 0.2978486	total: 1.31s	remaining: 392ms
77:	learn: 0.2976093	total: 1.33s	remaining: 375ms
78:	learn: 0.2974317	total: 1.34s	remaining: 357ms
79:	learn: 0.2971941	total: 1.36s	remaining: 340ms
80:	learn: 0.2969903	total: 1.37s	remaining: 322ms
81:	learn: 0.2968263	total: 1.39s	remaining: 305ms
82:	learn: 0.2965747	total: 1.4s	remaining: 288ms
83:	learn: 0.2960388	total: 1.42s	remaining: 271ms
84:	learn: 0.2959037	total: 1.44s	remaining: 253ms
85:	learn: 0.2957173	total: 1.45s	remaining: 236ms
86:	learn: 0.2956088	total: 1.47s	remaining: 219ms
87:	learn: 0.2951233	total: 1.49s	remaining: 203ms
88:	learn: 0.2950080	total: 1.5s	remaining: 186ms
89:	learn: 0.2948812	total: 1.52s	remaining: 168ms
90:	learn: 0.2946939	total: 1.53s	remaining: 151ms
91:	learn: 0.2945682	total: 1.55s	remaining: 134ms
92:	learn: 0.2942810	total: 1.56s	remaining: 118ms
93:	learn: 0.2940411	total: 1.58s	remaining: 101ms
94:	learn: 0.2936845	total: 1.59s	remaining: 83.9ms
95:	learn: 0.2934826	total: 1.61s	remaining: 67.1ms
96:	learn: 0.2933020	total: 1.63s	remaining: 50.3ms
97:	learn: 0.2931379	total: 1.64s	remaining: 33.5ms
98:	learn: 0.2930214	total: 1.66s	remaining: 16.7ms
99:	learn: 0.2929389	total: 1.67s	remaining: 0us
```

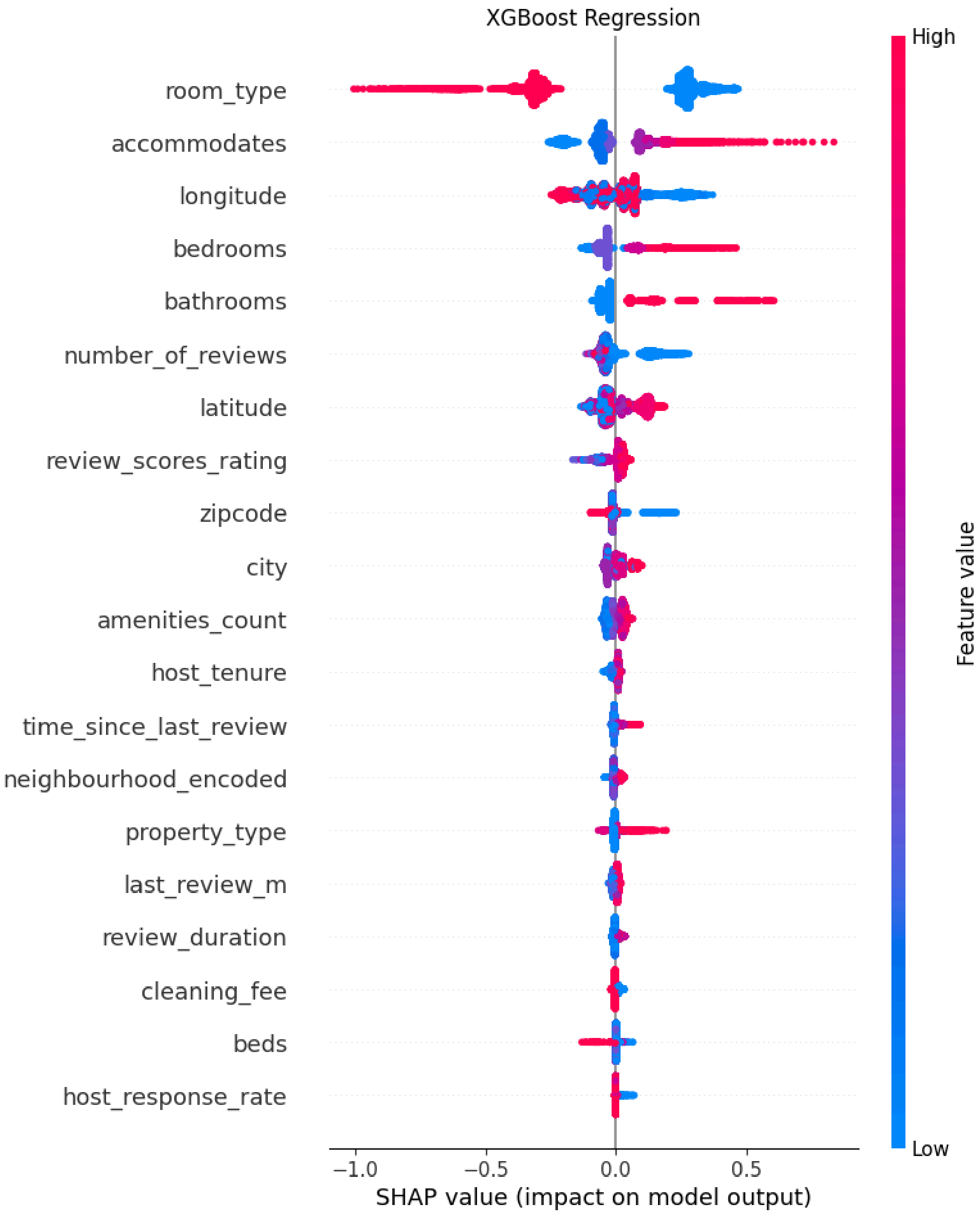
In [ ]:

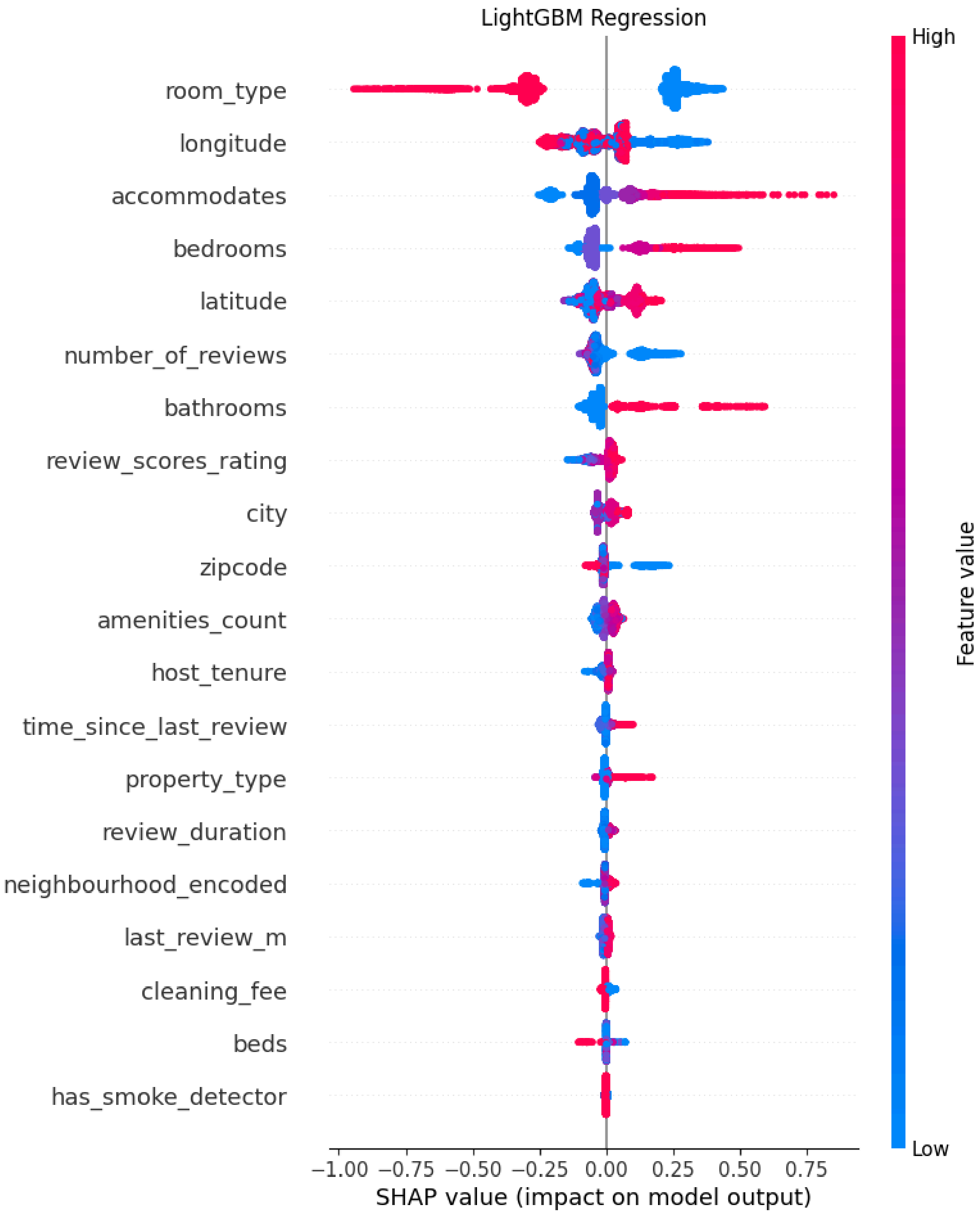
```
# Initialize the SHAP explainer for each model
xgb_explainer = shap.Explainer(xgb_reg)
lgb_explainer = shap.Explainer(lgb_reg)
catboost_explainer = shap.Explainer(catboost_reg)

# Compute SHAP values for each model
xgb_shap_values = xgb_explainer.shap_values(X_test)
lgb_shap_values = lgb_explainer.shap_values(X_test)
catboost_shap_values = catboost_explainer.shap_values(X_test)
```

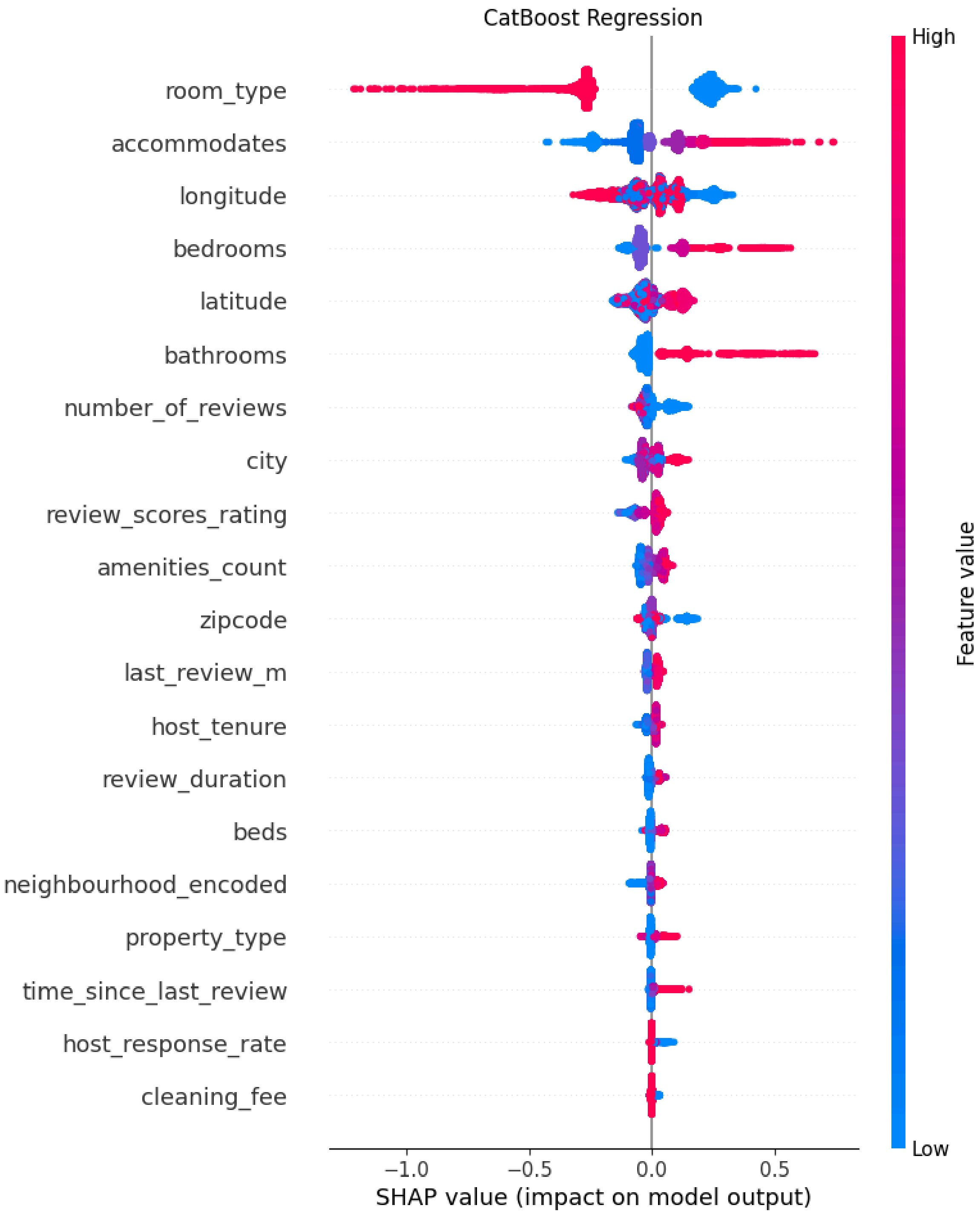
```
# Store SHAP values in the trained_models dictionary
trained_models['XGBoost']['shap_values'] = xgb_shap_values
trained_models['LightGBM']['shap_values'] = lgb_shap_values
trained_models['CatBoost']['shap_values'] = catboost_shap_values

# Save the trained models dictionary to a pickle file
with open('trained_models_with_metrics_and_shap.pkl', 'wb') as f:
    pickle.dump(trained_models, f)
```

In [ ]:

```
# Plot SHAP summary plots for each model
shap.summary_plot(xgb_shap_values, X_test, feature_names=X_test.columns, show=Fa
plt.title('XGBoost Regression')
plt.savefig('xgboost_shap_summary_plot.png')
plt.show()

shap.summary_plot(lgb_shap_values, X_test, feature_names=X_test.columns, show=Fa
plt.title('LightGBM Regression')
plt.savefig('lightgbm_shap_summary_plot.png')
plt.show()

shap.summary_plot(catboost_shap_values, X_test, feature_names=X_test.columns, sh
plt.title('CatBoost Regression')
plt.savefig('catboost_shap_summary_plot.png')
plt.show()
```

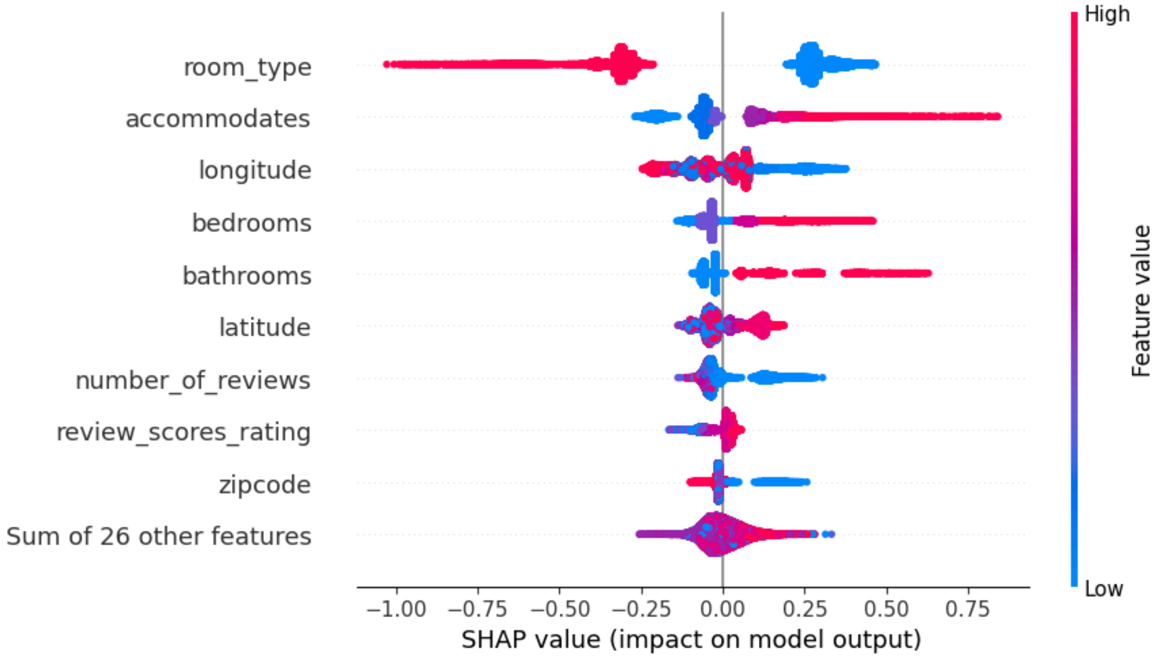

XGBoost Regression
room_type
accommodates
longitude
bedrooms
bathrooms
number_of_reviews
latitude
review_scores_rating
zipcode
city
amenities_count
host_tenure
time_since_last_review
neighbourhood_encoded
property_type
last_review_m
review_duration
cleaning_fee
beds
host_response_rate
−1.0
−0.5
0.0
0.5
SHAP value (impact on model output)
High
Feature value
Low


LightGBM Regression
room_type
longitude
accommodates
bedrooms
latitude
number_of_reviews
bathrooms
review_scores_rating
city
zipcode
amenities_count
host_tenure
time_since_last_review
property_type
review_duration
neighbourhood_encoded
last_review_m
cleaning_fee
beds
has_smoke_detector
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75
SHAP value (impact on model output)
High
Low
Feature value



XGBoost

In [ ]:
```
explainer = shap.TreeExplainer(trained_models["XGBoost"]["model"])
explanation = explainer(X=X_train,y=y_train)

shap.plots.beeswarm(explanation)
```



In [ ]:
```
shap_values = explainer(X_train,y_train)
```

In [ ]:
```
# visualize the first prediction's explanation
shap.plots.force(shap_values[0, ...])
```

Out[ ]:

Visualization omitted, Javascript library not loaded!

Have you run `initjs()` in this notebook? If this notebook was from another user you must also trust this notebook (File -> Trust notebook). If you are viewing this notebook on github the Javascript has been stripped for security. If you are using JupyterLab this error is because a JupyterLab extension has not yet been written.

In [ ]:
```
shap.force_plot(
    explainer.expected_value, shap_values.values[:1000, :], X_train.iloc[:1000,
)
```

Out[ ]:

Visualization omitted, Javascript library not loaded!

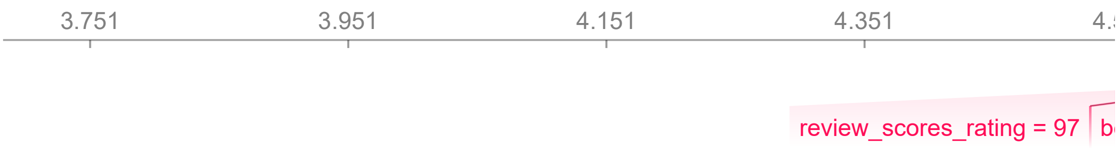
Have you run `initjs()` in this notebook? If this notebook was from another user you must also trust this notebook (File -> Trust notebook). If you are viewing this notebook on github the Javascript has been stripped for security. If you are using JupyterLab this error is because a JupyterLab extension has not yet been written.

CatBoost

In [ ]:
```
shap.initjs()
```

In [ ]:
```
explainer = shap.TreeExplainer(trained_models["CatBoost"]["model"])
shap_values = explainer(X_train,y_train)

# visualize the first prediction's explanation
shap.plots.force(shap_values[0, ...])
```

Out[ ]:

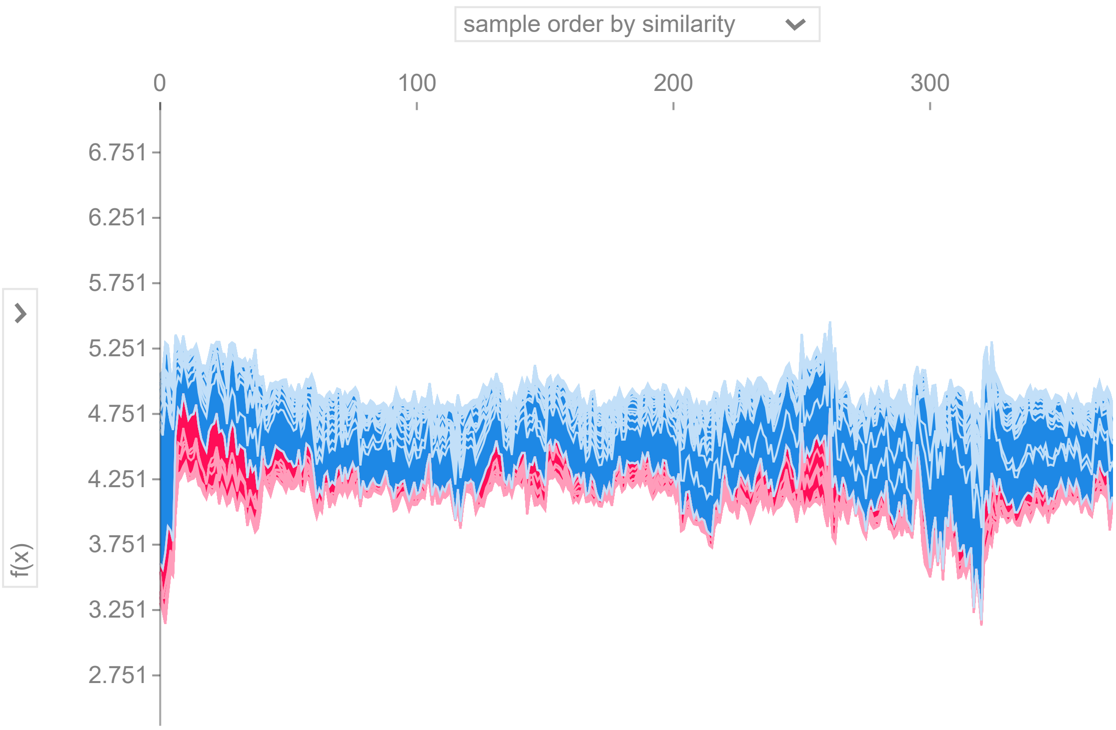


The above explanation shows features each contributing to push the model output from the base value (the average model output over the training dataset we passed) to the model output. Features pushing the prediction higher are shown in red, those pushing the prediction lower are in blue.

If we take many explanations such as the one shown above, rotate them 90 degrees, and then stack them horizontally,

In [ ]:
```
shap.force_plot(
    explainer.expected_value, shap_values.values[:1000, :], X_train.iloc[:1000,
)
```

Out[ ]:



To understand how a single feature affects the output of the model, we can plot the SHAP value of that feature vs. the value of the feature for all the examples in a dataset. Since SHAP values represent a feature's responsibility for a change in the model output,

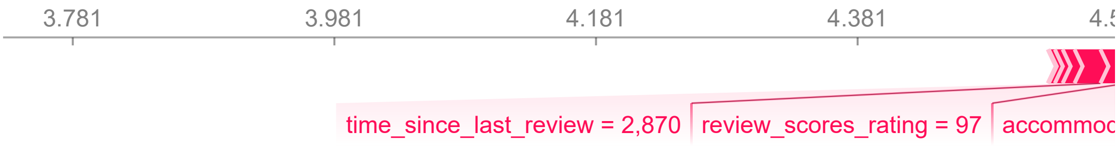
---

LightBoost

In [ ]:
```
explainer = shap.TreeExplainer(trained_models["LightGBM"]["model"])
shap_values = explainer(X_train, y_train)
```

```python
# visualize the first prediction's explanation
shap.plots.force(shap_values[0, ...])
```

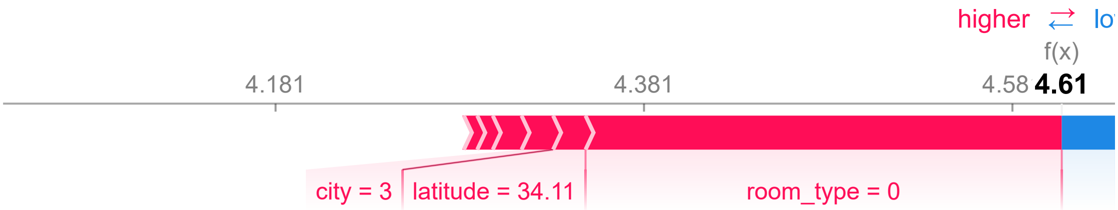
Out[ ]:



In [ ]:

```python
shap.force_plot(
    explainer.expected_value, shap_values.values[1, :], X_train.iloc[0, :]
)
```

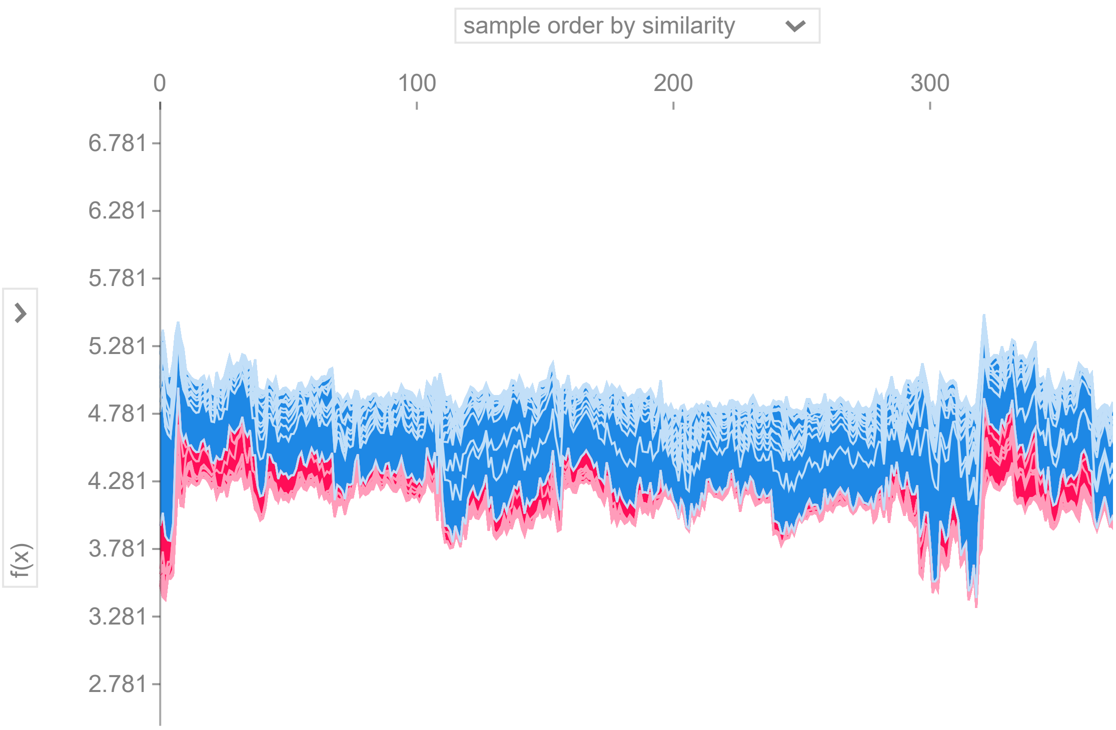
Out[ ]:



In [ ]:

```python
shap.force_plot(
    explainer.expected_value, shap_values.values[:1000, :], X_train.iloc[:1000,
)
```

Out[ ]:



Saving the shap values and use them in the front page

Model Card

In [ ]:
```
model_details = """
  # Model Details

  ## Description
  * Model name: XgBoost
  * Model version
  * Model author: Darshan kumar
  * Model type: Regression Task
  * Model architecture: Include any relevant information about algorithms, param
  * Date
  * contact: Darshankumar@gmail.com

  ## Intended use
   * Primary use case: building a predective model for predicting 'log_price' of
"""
```

In [ ]:
```
training_dataset = """
  # Training dataset

  * Training dataset: The training dataset comprises data collected from a varie
  * Training period: The training dataset spans from a recent timeframe, capturi
  * Sub-groups: The dataset includes relevant sub-groups such as demographic inf
  * Limitations: While efforts were made to ensure data accuracy, there are know
  * Pre-processing: Prior to analysis, the data underwent various pre-processing

"""
```

In [ ]:
```
model_evaluation = """
  # Model evaluation

 * Evaluation process: The model was evaluated using standard regression evaluat
 * Evaluation dataset: The evaluation dataset was created by partitioning the or
 * Metrics: Key model quality metrics for regression tasks include mean squared
 * Decision threshold: In regression tasks, there isn't a decision threshold in

"""
```

In [ ]:
```
# !pip install evidently
```

In [ ]:
```
train_data['prediction'] = xgb_reg.predict(X)
```

In [ ]:
```
train_data.columns = cols = ['id',
 'target',
 'accommodates',
 'bathrooms',
 'zipcode',
 'neighbourhood_encoded',
 'host_response_rate',
 'number_of_reviews',
 'review_scores_rating',
 'bedrooms',
 'latitude',
 'longitude',
 'beds',
```

```
 'review_duration',
 'time_since_last_review',
 'host_tenure',
 'average_review_score',
 'amenities_count',
 'has_wireless_internet',
 'has_kitchen',
 'has_heating',
 'has_essentials',
 'has_smoke_detector',
 'first_review_m',
 'host_since_m',
 'last_review_m',
 'property_type',
 'room_type',
 'bed_type',
 'cancellation_policy',
 'city',
 'host_has_profile_pic',
 'host_identity_verified',
 'instant_bookable',
 'cleaning_fee',
 'sentiment_category',
 'prediction']
```

In [ ]:

```
model_card = Report(metrics=[
    Comment(model_details),
    Comment(training_dataset),
    DatasetSummaryMetric(),
    Comment(model_evaluation),
    RegressionQualityMetric(),
    RegressionErrorDistribution(),
])

model_card.run(current_data=train_data[:60000], reference_data=train_data[60000:
model_card
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_regression.py:918: Undef
inedMetricWarning:

R^2 score is not well-defined with less than two samples.

/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_regression.py:918: Undef
inedMetricWarning:

R^2 score is not well-defined with less than two samples.
```

Out[ ]:

Model Details

Description

- Model name: XgBoost
- Model version
- Model author: Darshan kumar
- Model type: Regression Task
- Model architecture: Include any relevant information about algorithms, parameters, etc.
- Date
- contact: Darshankumar@gmail.com

Intended use

- Primary use case: building a predective model for predicting 'log_price' of home

Training dataset

- Training dataset: The training dataset comprises data collected from a variety of sources, primarily focusing on listings from an online accommodation platform. These listings include details such as 'id', 'accommodates', 'bathrooms', 'zipcode', 'neighbourhood_encoded', 'host_response_rate', 'number_of_reviews', 'review_scores_rating', 'bedrooms', 'latitude', 'longitude', 'beds', 'review_duration', 'time_since_last_review', 'host_tenure', 'average_review_score', 'amenities_count', 'has_wireless_internet', 'has_kitchen', 'has_heating', 'has_essentials', 'has_smoke_detector', 'first_review_m', 'host_since_m', 'last_review_m', 'property_type', 'room_type', 'bed_type', 'cancellation_policy', 'city', 'host_has_profile_pic', 'host_identity_verified', 'instant_bookable', 'cleaning_fee', 'sentiment_category'.
- Training period: The training dataset spans from a recent timeframe, capturing data from the past few years up to the present.
- Sub-groups: The dataset includes relevant sub-groups such as demographic information, property characteristics, and host-related attributes.
- Limitations: While efforts were made to ensure data accuracy, there are known limitations inherent to the dataset. These may include missing values, inconsistencies, or biases in the data collection process.
- Pre-processing: Prior to analysis, the data underwent various pre-processing steps including cleaning, normalization, and feature engineering to ensure compatibility and optimize predictive modeling performance.

Dataset Summary

Metric	Current
id column	None
target column	target
prediction column	prediction
date column	None
number of columns	37
number of rows	60000
missing values	0
categorical columns	0
numeric columns	35
text columns	0
datetime columns	0
empty columns	0
constant columns	0
almost constant features	3
duplicated columns	0
almost duplicated features	1

Model evaluation

- Evaluation process: The model was evaluated using standard regression evaluation techniques, such as mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), and R-squared (R2) score. These metrics provide insight into the model's ability to accurately predict numerical outcomes.
- Evaluation dataset: The evaluation dataset was created by

partitioning the original dataset into separate training and evaluation sets. The evaluation set consists of a subset of the data that was not used during model training, ensuring unbiased assessment of the model's predictive performance on unseen data.
- Metrics: Key model quality metrics for regression tasks include mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), and R-squared (R2) score. These metrics quantify the difference between predicted and actual values, providing an indication of the model's accuracy and goodness of fit.
- Decision threshold: In regression tasks, there isn't a decision threshold in the same sense as classification tasks. Instead, the model's predictions are continuous numerical values, and evaluation metrics are used to assess the closeness of these predictions to the actual target values.

Regression Model Performance. Target: 'target'

Current: Model Quality (+/- std)

0.0 (0.41)	0.3 (0.28)
ME	MAE

26361550053491.62 (64572346459877.41)

MAPE

Reference: Model Quality (+/- std)

-0.01 (0.42)	0.3 (0.29)	6.39 (0.06)
ME	MAE	MAPE