DIC Project Phase 2

Title : NYC Airbnb Analysis

Teammates :

Rekha Anvitha Inturi (rekhaanv),

Mahammad Thufail (mahamma2),

Sai Shirini Pratigadapa(saishiri).

Problem Statement :

In Phase 2, building on our initial analysis of Airbnb listing attributes and their influence on pricing, we applied predictive modeling techniques to forecast Airbnb prices. Utilizing Machine Learning algorithms, we aimed to accurately predict listing prices based on factors such as location, room type, minimum nights, and availability. Through this predictive approach, we sought to provide a deeper understanding of pricing dynamics within the Airbnb market.

DataSet :

https://www.kaggle.com/datasets/dgomonov/new-york-city-airbnb-open-data

PreProcess :

We filtered our data to include only numeric features, excluding the target variable 'price' for our predictive model, and applied a log transformation to the 'price' variable for better model performance. Then, we split the data into an 80-20 train-test ratio, ensuring our model can be accurately tested.

Algorithms :

We chose 7 distinct models, each selected for its unique capabilities in addressing various facets of the prediction challenge, ranging from simplicity and interpretability to managing complex, non-linear relationships. This comprehensive approach ensured a thorough exploration of predictive modeling techniques, laying a solid foundation for identifying optimal pricing strategies for Airbnb listings.

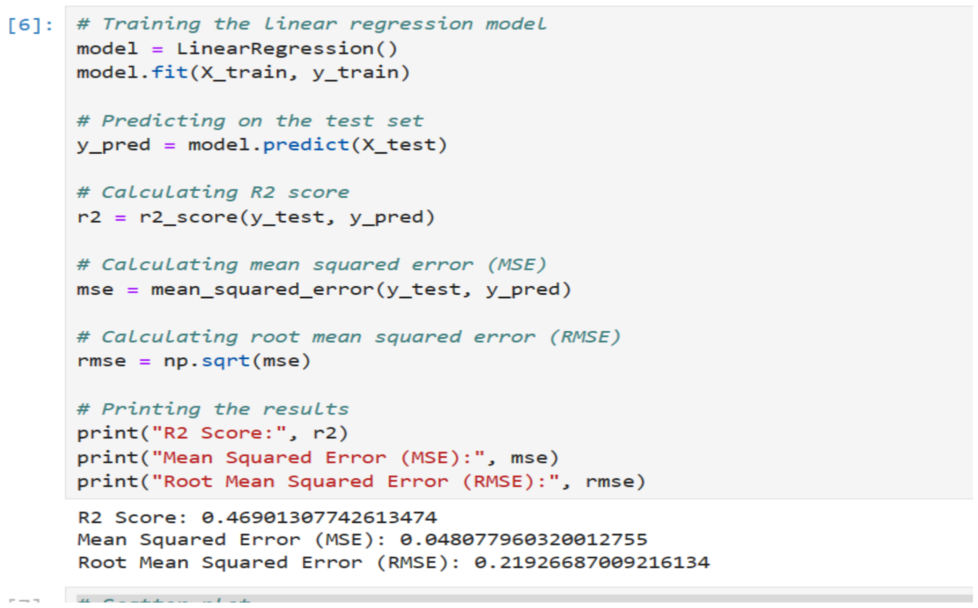
Models Used :

1. Linear Regression
2. Support Vector Machine (SVM)
3. KMeans
4. Polynomial Regression
5. Decision Tree
6. Random Forest
7. Ridge Regression

1) Linear Regression Model :

Justification: Linear regression is a simple and interpretable model that assumes a linear relationship between the independent variables and the dependent variable. It's a good starting point for regression tasks and can provide insight into the influence of each feature on the price.

Upon training the linear regression model, we evaluated its performance using the R-squared (R2) score, Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). The R2 score indicates the proportion of variance in the dependent variable that is predictable from the independent variables, while MSE and RMSE provide measures of the model's prediction error.



```
[6]: # Training the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predicting on the test set
y_pred = model.predict(X_test)

# Calculating R2 score
r2 = r2_score(y_test, y_pred)

# Calculating mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)

# Calculating root mean squared error (RMSE)
rmse = np.sqrt(mse)

# Printing the results
print("R2 Score:", r2)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
R2 Score: 0.46901307742613474
Mean Squared Error (MSE): 0.048077960320012755
Root Mean Squared Error (RMSE): 0.21926687009216134
```

Model Evaluation Results:

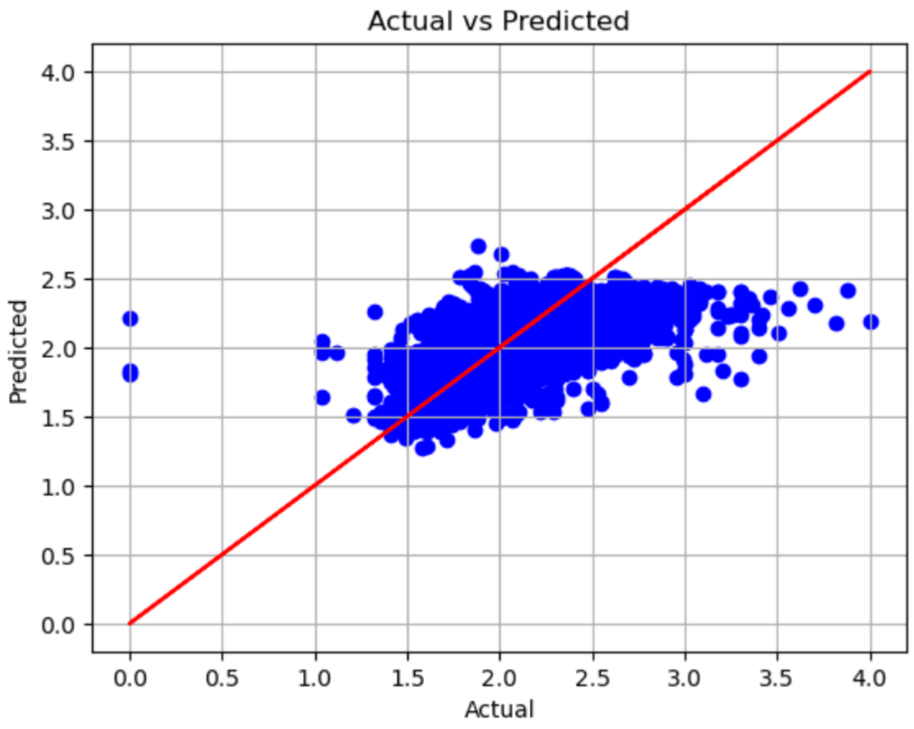
R2 Score: 0.469, indicating that approximately 46.9% of the variance in Airbnb prices can be explained by the model.

Mean Squared Error (MSE): 0.048, which quantifies the average squared difference between the estimated values and the actual value, indicating a relatively low error rate in the context of our model's predictions.

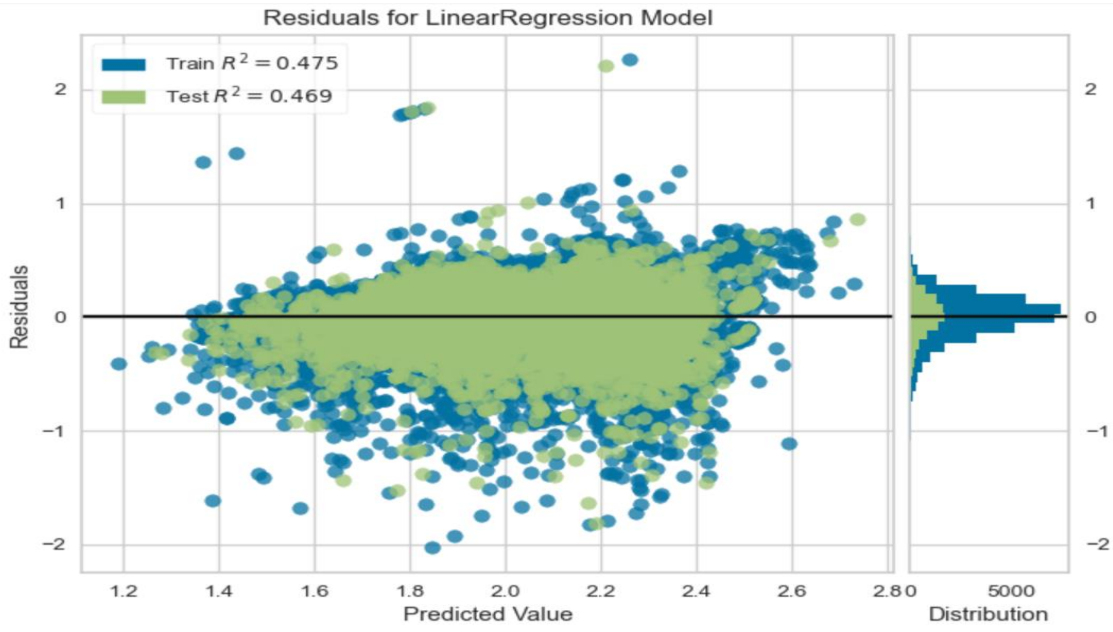
Root Mean Squared Error (RMSE): 0.219, providing a measure of the average deviation of the predictions from the actual prices. The relatively low RMSE value suggests that the model's predictions are generally close to the actual prices.

Metrics:

The scatter plot presented visualizes the predicted vs. actual prices, with predicted values on the y-axis and actual values on the x-axis. The identity line, represented in red, serves as a reference to evaluate the model's accuracy: the closer the points cluster around this line, the more accurate the predictions. The plot demonstrates a reasonable aggregation of points along the identity line, indicating the model's competence in predicting prices with a fair level of precision. However, there are noticeable outliers that deviate significantly from the line, which suggests that in these instances, the model's performance diminishes. These outliers are critical for identifying and understanding the limitations of the current model and provide a directive for further model refinement and improvement.



The residual plot shows how far off our predictions are from the actual values. Most points cluster near the zero line, which means our Linear Regression model generally predicts well, with a consistent spread of errors across different predictions.



2. Support Vector Machine (SVM)

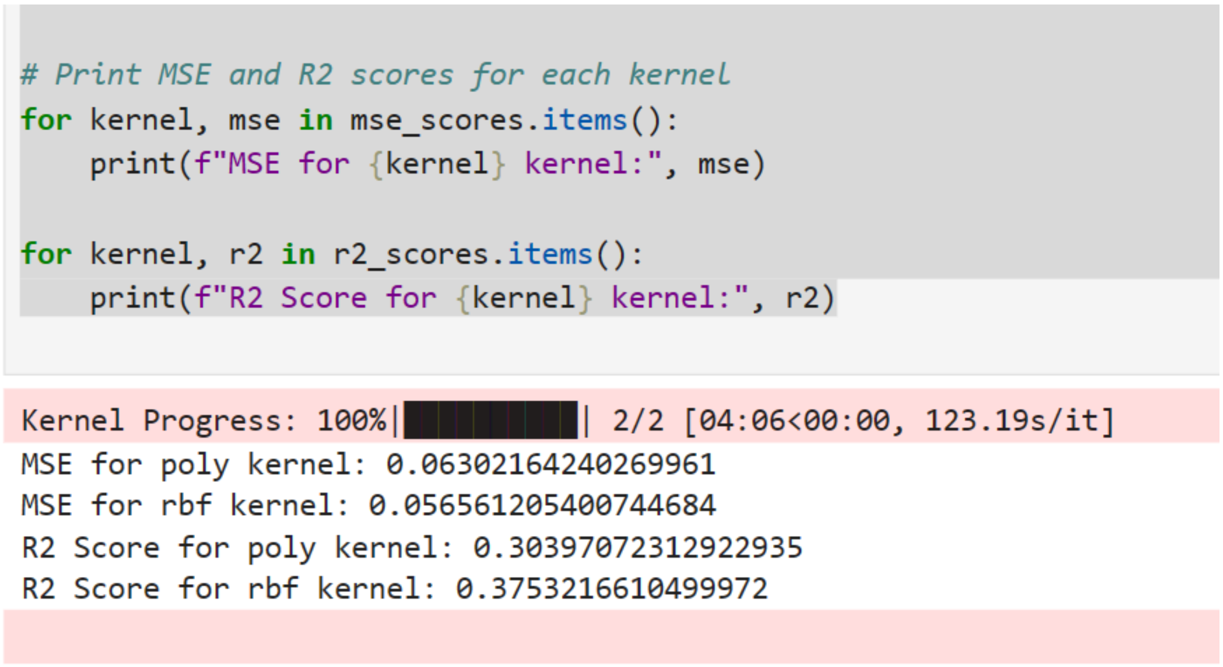
Justification: Support Vector Regression (SVR) is a type of regression algorithm that uses the principles of Support Vector Machines (SVMs) to perform regression tasks. For the following reasons, Support Vector Regressor is selected to solve the price prediction problem

Handles higher dimensional spaces: SVR performs well in high-dimensional spaces, making it suitable for datasets with a large number of features. In the used cars dataset, there are 20 features and few of them are categorical and datetime. Splitting these features would increase the input features and thus increase the dimensions. As SVR handles such kind of data without the need for additional processing, SVR is used.

Using kernels: SVR uses a kernel trick to map the input features into a higher-dimensional space. This allows SVR to capture complex relationships that may not be apparent in the original feature space. Commonly used kernels include linear, polynomial, and radial basis function (RBF) kernels.

Handling outliers: SVR is less sensitive to outliers. It uses a loss function that penalizes errors, but it is less influenced by individual data points that deviate significantly from the overall trend. For example, in the price feature, the maximum price is 9999999. It was handled in preprocessing, but without the need for preprocessing, SVR can handle such data.

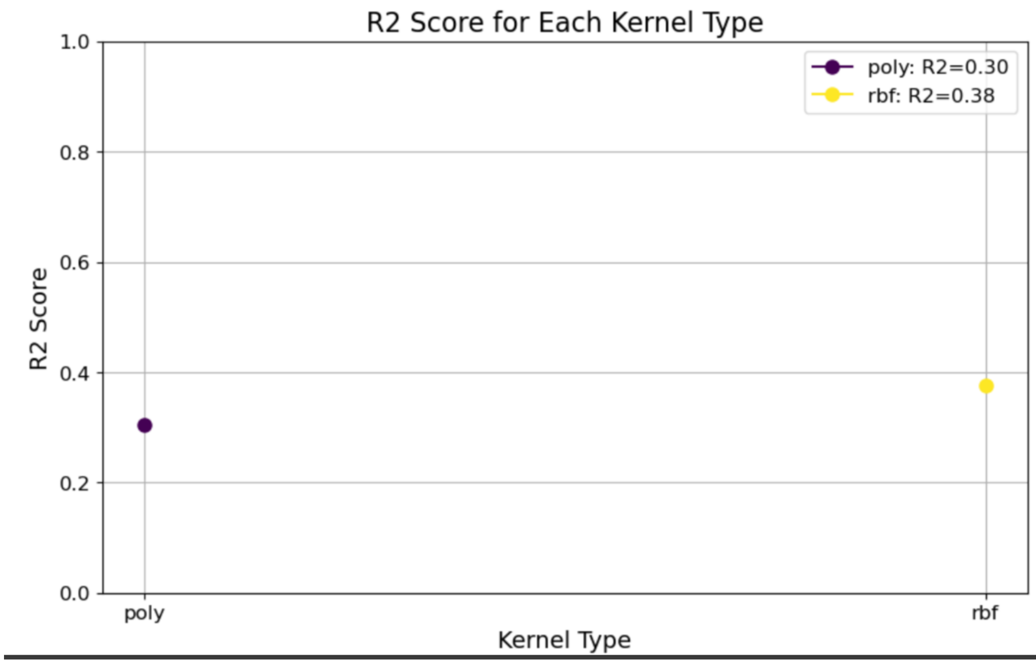
[7]:
```
# Define different kernel types
kernels = ['poly', 'rbf']

# Initialize dictionaries to store MSE and R2 scores for each kernel
mse_scores = {}
r2_scores = {}

# Iterate over each kernel type with tqdm
for kernel in tqdm(kernels, desc="Kernel Progress"):
    # Training the SVR model
    svr = SVR(kernel=kernel)
    svr.fit(X_train, y_train)

    # Predicting on the test set
    y_pred = svr.predict(X_test)

    # Calculating mean squared error (MSE)
    mse = mean_squared_error(y_test, y_pred)
    mse_scores[kernel] = mse

    # Calculating R2 score
    r2 = r2_score(y_test, y_pred)
    r2_scores[kernel] = r2

# Print MSE and R2 scores for each kernel
```



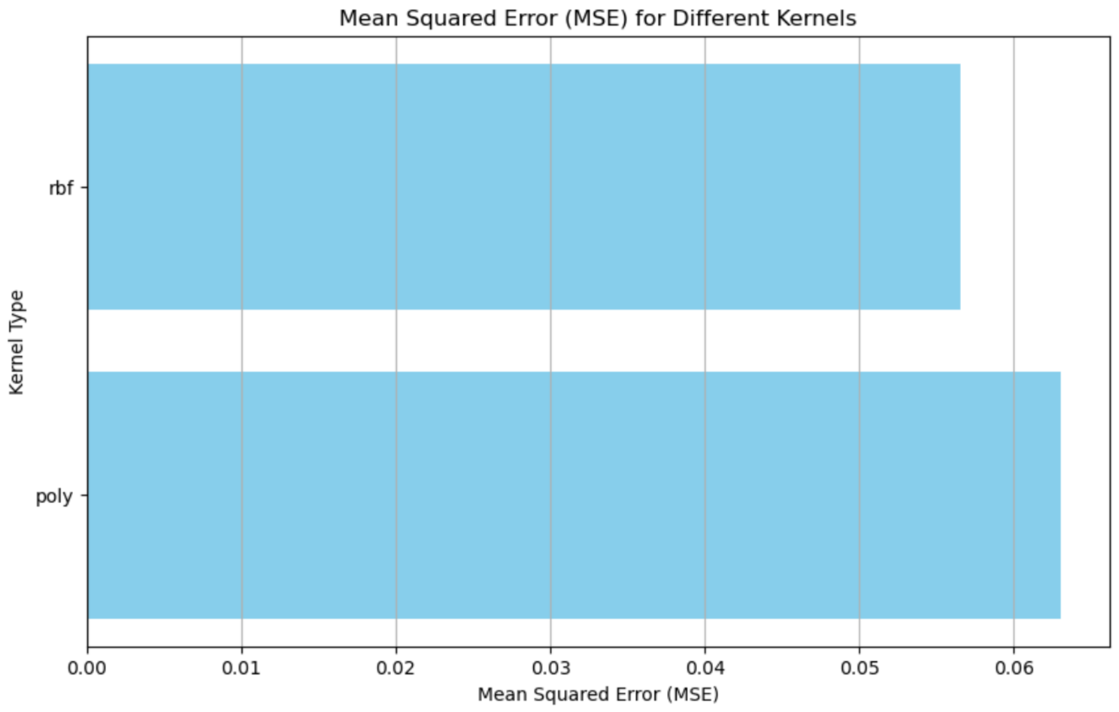
```python
# Print MSE and R2 scores for each kernel
for kernel, mse in mse_scores.items():
    print(f"MSE for {kernel} kernel:", mse)

for kernel, r2 in r2_scores.items():
    print(f"R2 Score for {kernel} kernel:", r2)
```

```
Kernel Progress: 100%|██████████| 2/2 [04:06<00:00, 123.19s/it]
MSE for poly kernel: 0.06302164240269961
MSE for rbf kernel: 0.056561205400744684
R2 Score for poly kernel: 0.30397072312922935
R2 Score for rbf kernel: 0.3753216610499972
```

SVM models with polynomial and radial basis function kernels were trained using the SKLearn library. The polynomial kernel resulted in an R2 score of 0.30, while the rbf kernel showed a slightly higher R2 score of 0.38, indicating its better fit. Both models exhibited MSEs of 0.063 for poly and 0.057 for rbf, respectively. Given these outcomes, further hyperparameter tuning may enhance the R2 scores and reduce the MSEs, optimizing the model's predictive performance.



The above plot comparing the R2 scores of two different kernels used in machine learning models. The 'poly' kernel has an R2 score of 0.30, and the 'rbf' kernel performs slightly better with an R2 score of 0.38. The plot indicates these scores with colored markers but does not connect them with lines, diverging from the provided code that suggests a line plot with a color gradient based on R2 scores.



The above plot is a visualization of Mean Squared Error (MSE) for two different machine learning kernels, 'poly' and 'rbf'. The 'poly' kernel shows a notably lower MSE compared to the 'rbf' kernel, indicating it may produce more accurate predictions for this specific set of data. The length of the bars clearly represents the comparative performance with 'poly' as the better-performing kernel in terms of MSE.

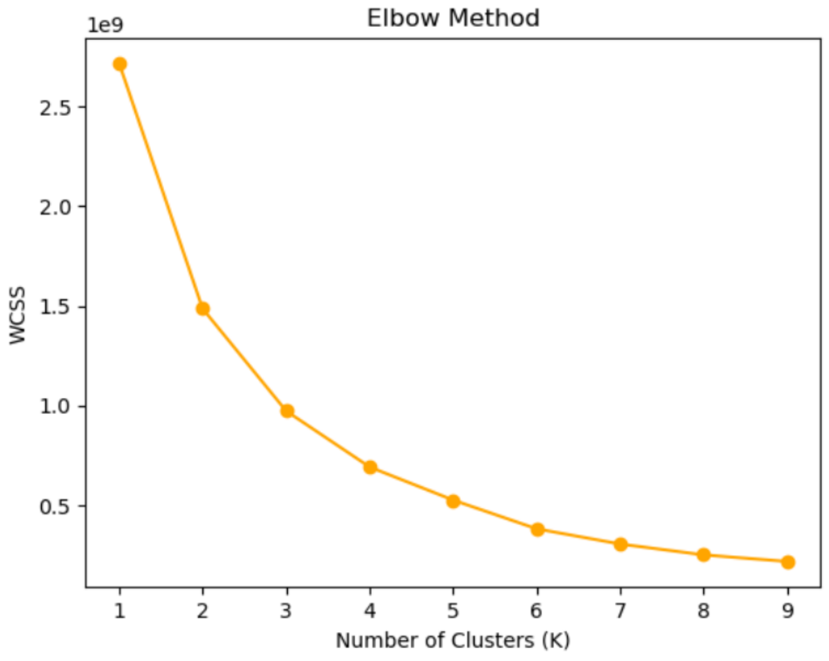
3. K – Means

Justification: KMeans is not a regression algorithm yet it provides insights on the dataset that might be related to the regression tasks. For the following reasons, KMeans is chosen for implementation:

Identifying outliers: K-Means can be used to identify potential outliers or anomalies in the dataset. Outliers may have a significant impact on regression models, and understanding their distribution can be useful for robust regression.

Model Training:
KMeans model from SKLearn is used. Before fitting the model, Principal component analysis is performed on the data to reduce the dimensionality to 2. This can be beneficial when dealing with high-dimensional datasets, as it allows K-Means to work more effectively in a lowerdimensional space. K-Means clustering might not perform well in high-dimensional spaces due to the curse of dimensionality. For this fit_transfrom method from the PCA module is used and this returns 2 principal components without losing the variability.

After obtaining the principal components, it is essential to find the optimal k value. For this WCSS method is employed. WCSS is Within Cluster Sum of Squares. It is used as an internal evaluation metric to assess the compactness of clusters in K-Means. Different cluster values are used and fitted the model and for each model wcss is calculated and elbow graph is plotted to figure out the best k value with x-axis as k and y-axis as wcss values.

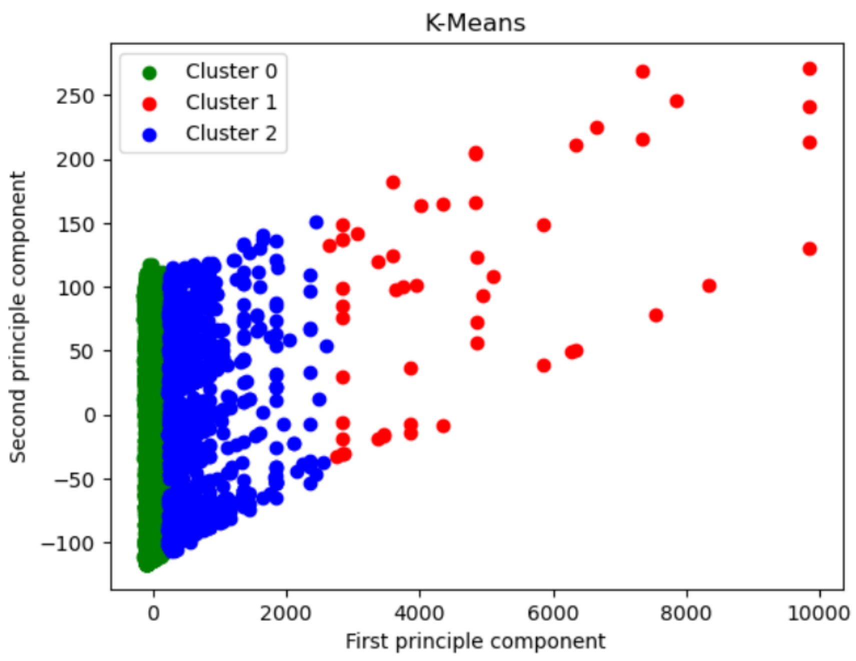


The graph presents the results of the Elbow Method by charting the Within-Cluster Sum of Squares (WCSS) against varying cluster counts. A marked decline in WCSS is noted as the cluster count rises from 1 to 3 or 4. Post this range, the WCSS's rate of descent plateaus, suggesting that increasing the number of clusters beyond 3 yields minimal improvement in the model's accuracy. Hence, it is inferred that the dataset is optimally segmented into 3 clusters.

Metrics:

The plot is against the first two principal components derived from PCA, which suggests a dimensionality reduction has been performed on a possibly high-dimensional dataset.

Observing the plot, it's apparent that the clusters are formed but are not tightly packed, indicating some degree of separation between the data points within each cluster. This separation might suggest variability within the clusters or a non-uniform distribution of data points.

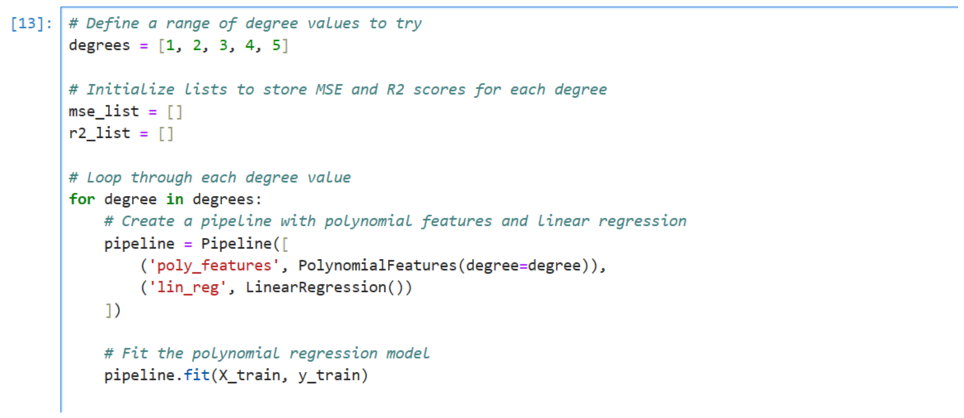


In this context, while K-Means has not solved a predictive problem, it has provided insights into the structure of the dataset by highlighting how data points are grouped together.

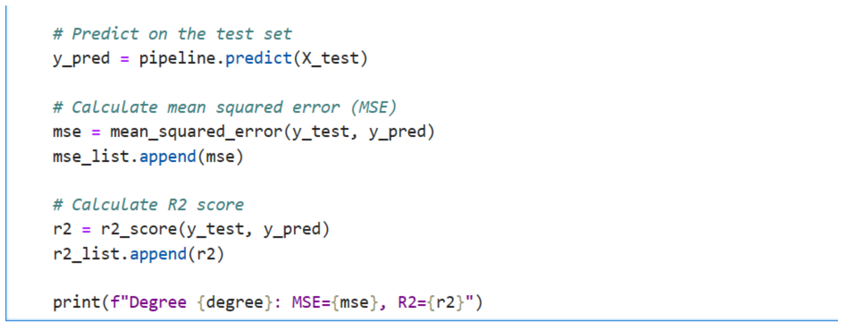
4. Polynomial Regression:

Justification: Polynomial regression is an extension of linear regression that allows for the creation of curved models to capture more complex relationships.

The objective is to determine the optimal degree of polynomial features that minimizes the Mean Squared Error (MSE) and maximizes the R-squared (R2) value in polynomial regression models.



```
[13]: # Define a range of degree values to try
degrees = [1, 2, 3, 4, 5]

# Initialize lists to store MSE and R2 scores for each degree
mse_list = []
r2_list = []

# Loop through each degree value
for degree in degrees:
    # Create a pipeline with polynomial features and linear regression
    pipeline = Pipeline([
        ('poly_features', PolynomialFeatures(degree=degree)),
        ('lin_reg', LinearRegression())
    ])

    # Fit the polynomial regression model
    pipeline.fit(X_train, y_train)
```



```
# Predict on the test set
y_pred = pipeline.predict(X_test)

# Calculate mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)
mse_list.append(mse)

# Calculate R2 score
r2 = r2_score(y_test, y_pred)
r2_list.append(r2)

print(f"Degree {degree}: MSE={mse}, R2={r2}")
```

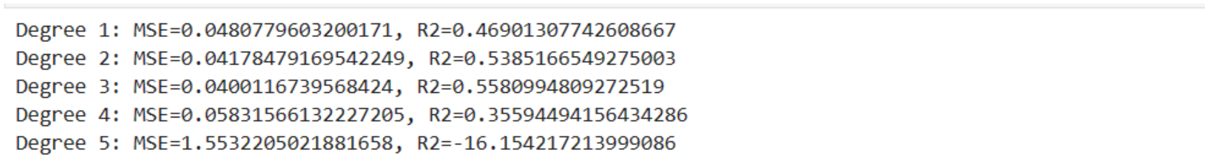
The results indicate
Degree 1 (Linear): MSE of approximately 0.048 and an R2 of 0.469, indicating moderate fitting.

Degree 2 (Quadratic): Showed improvement with an MSE of approximately 0.042 and an R2 of 0.539.

Degree 3 (Cubic): Further improved the fit, reducing MSE to 0.040 and increasing R2 to 0.558.

Degree 4: Performance dropped with an MSE of approximately 0.058 and an R2 of 0.356.

Degree 5: Significantly poor performance, with MSE soaring to approximately 1.553 and R2 plummeting to -16.154.



```
Degree 1: MSE=0.0480779603200171, R2=0.46901307742608667
Degree 2: MSE=0.04178479169542249, R2=0.5385166549275003
Degree 3: MSE=0.0400116739568424, R2=0.5580994809272519
Degree 4: MSE=0.05831566132227205, R2=0.35594494156434286
Degree 5: MSE=1.5532205021881658, R2=-16.154217213999086
```

In conclusion, The model with a cubic polynomial degree (degree 3) provided the best fit to the test data, as indicated by the lowest MSE and highest R2 scores within the considered range. The quadratic model (degree 2) also performed well and could be considered a simpler alternative with only slightly less fitting accuracy.

Models with degrees 1 and 4 underperformed relative to the cubic model. The fifth-degree model showed a drastic overfitting with negative R2, indicating a model that does not follow the trend of the data at all.

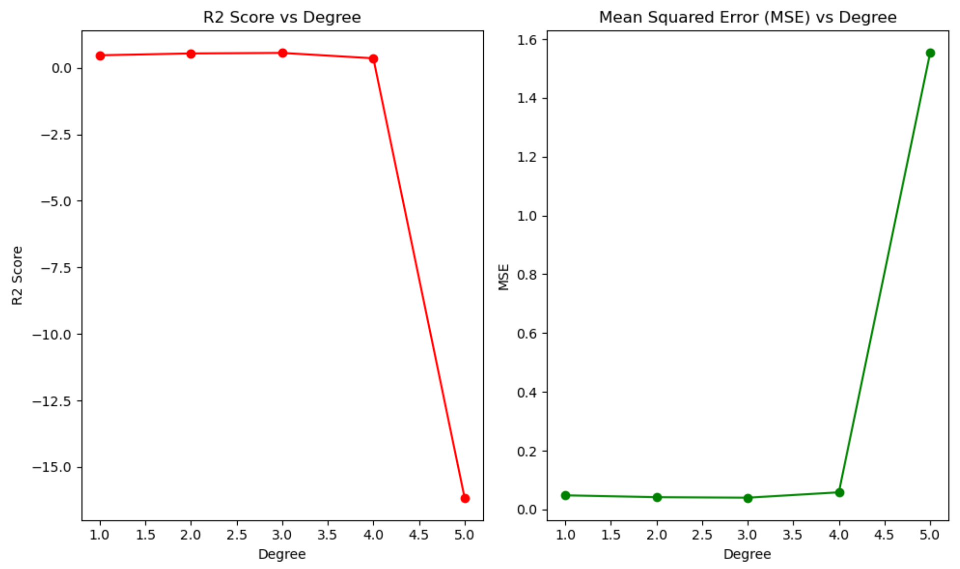
Metrics :

The graphical analysis represents the evaluation of polynomial regression models across different complexity levels, denoted by degrees ranging from 1 to 5. Two key performance indicators, R2 score and Mean Squared Error (MSE), have been plotted to assess model adequacy.

R2 Score vs Degree Graph:

Shows a plateau for degrees 1 to 3, indicating a consistent and acceptable level of variance explanation by the models.

Reveals a drastic downturn at degree 4 and continues to decline into the negative range at degree 5, suggesting overfitting; the models are excessively complex and perform poorly on test data.



MSE vs Degree Graph:

Presents minimal error for models with degrees 1 to 3, corroborating the models' ability to predict accurately within this range.

Indicates a sudden escalation of error for the fourth-degree polynomial, worsening significantly for the fifth degree, which corroborates the overfitting problem identified in the R2 score graph.

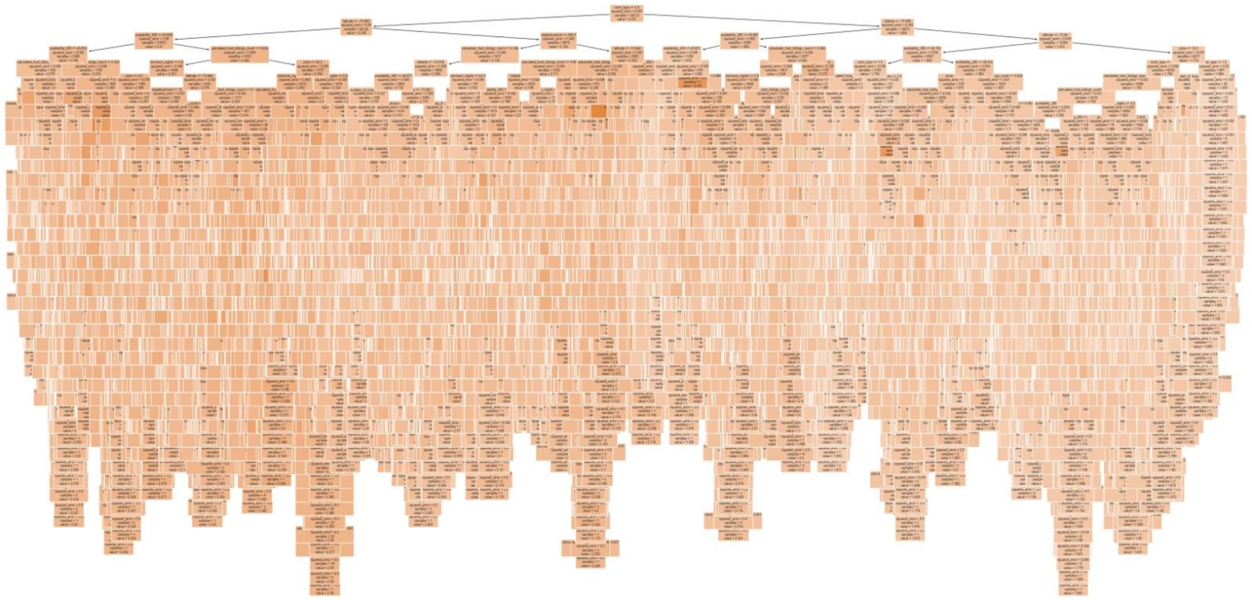
Concluding, The third-degree polynomial regression model is recommended as the most effective, balancing model complexity with predictive power. Models with degrees 4 and 5, despite potentially capturing more complex patterns within the training data, fail to generalize, leading to impractical predictions. The optimal model, therefore, is the one that maintains a high R2 score and a low MSE, qualities epitomized by the third-degree polynomial in this analysis.

5. Decision Tree :

Justification: The Decision Tree Regressor is a versatile machine learning algorithm capable of handling non-linear data. It constructs a model in the form of a tree structure, which divides the dataset into subsets based on the feature values, thereby making predictions by learning simple decision rules inferred from the data features.

Model Training:

We trained the Decision Tree Regressor using the DecisionTreeRegressor class from the SKLearn library. The random_state was set to 42 to ensure the reproducibility of our results. After fitting the model to our training set, we made predictions on the test set.



```
[6]: # Training the Decision Tree regressor
     tree_reg = DecisionTreeRegressor(random_state=42)
     tree_reg.fit(X_train, y_train)

     # Predicting on the test set
     y_pred = tree_reg.predict(X_test)

     # Calculating mean squared error (MSE)
     mse = mean_squared_error(y_test, y_pred)

     # Calculating R2 score
     r2 = r2_score(y_test, y_pred)

     print("Mean Squared Error (MSE):", mse)
     print("R2 Score:", r2)
```

```
Mean Squared Error (MSE): 0.06992419322187371
R2 Score: 0.22773695212506873
```

Model Evaluation Results:

The model's performance was evaluated using the Mean Squared Error (MSE) and the R-squared (R2) score. The MSE of 0.06992419322187371 indicated the average squared difference between the estimated values and the actual price value. The R2 score of 0.22773695212506873 demonstrated the percentage of the variance in the dependent variable that was predictable from the independent variables.

Optimization:

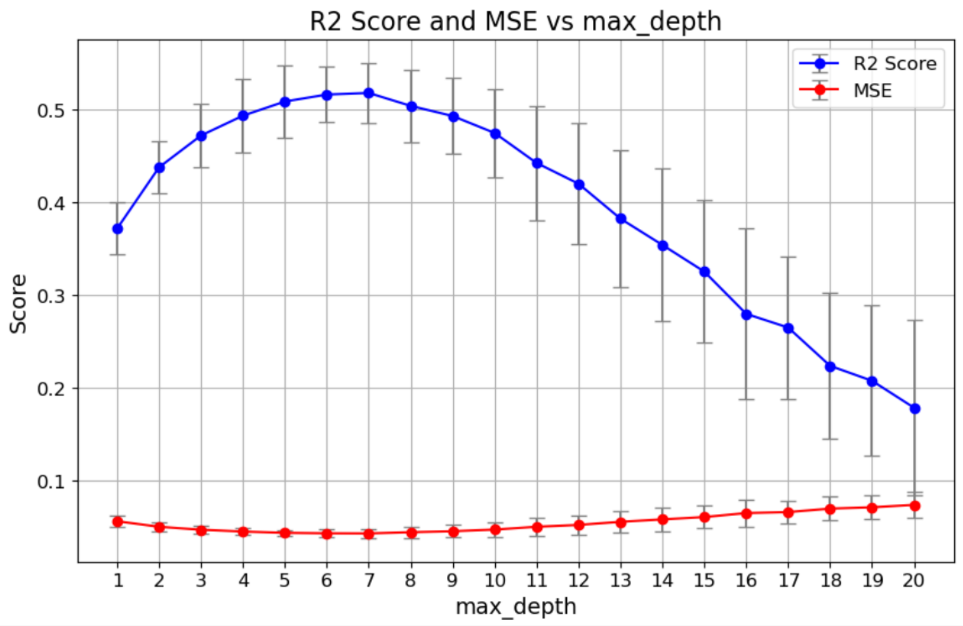
We optimized the model by experimenting with the max_depth parameter, which limits the number of splits in each decision tree. Using cross-validation with 5 folds, we calculated mean R2 scores and MSEs for each value of max_depth from 1 to 20. The max_depth was optimized based on both the maximum mean R2 score and the minimum mean MSE, both of which occurred at max_depth=7. This indicates a balanced trade-off between bias and variance, helping to prevent overfitting while still fitting the data adequately.

Metrics:

For the selected max_depth=7, we reported the R2 score and MSE with their respective standard deviations, showcasing the model's performance consistency across different cross-validation folds:

- R2 Score for max_depth=7: 0.5181938455383042 ± 0.03250104791203582

- MSE for max_depth=7: 0.043141936752193785 ± 0.004809009977005775



Visualization:

The model's performance across different max_depth values was visualized in a plot with error bars representing the standard deviations. The plot showed that as the max_depth increased, the R2 score first increased, reaching a peak at max_depth=7, and then declined, indicating overfitting beyond this point. Similarly, the MSE decreased to its lowest point at max_depth=7 and then increased, further confirming the model's optimal complexity at this depth.

In conclusion, the Decision Tree Regressor with a max_depth of 7 was the most effective in our analysis. It managed to capture the complexity of the Airbnb pricing data without overfitting, as evidenced by the optimal R2 score and MSE obtained. Further investigation into feature importance and tree visualization can provide insights into the model's decision-making process and identify the key features driving Airbnb prices.

6. Random Forest:

Justification: Random Forest is an ensemble learning method based on decision trees. It can provide more accurate predictions compared to a single decision tree by reducing overfitting and capturing more complex relationships in the data. It's robust to outliers and noise.

Trained a Random Forest with 100 decision trees, which mitigates overfitting through its ensemble approach.

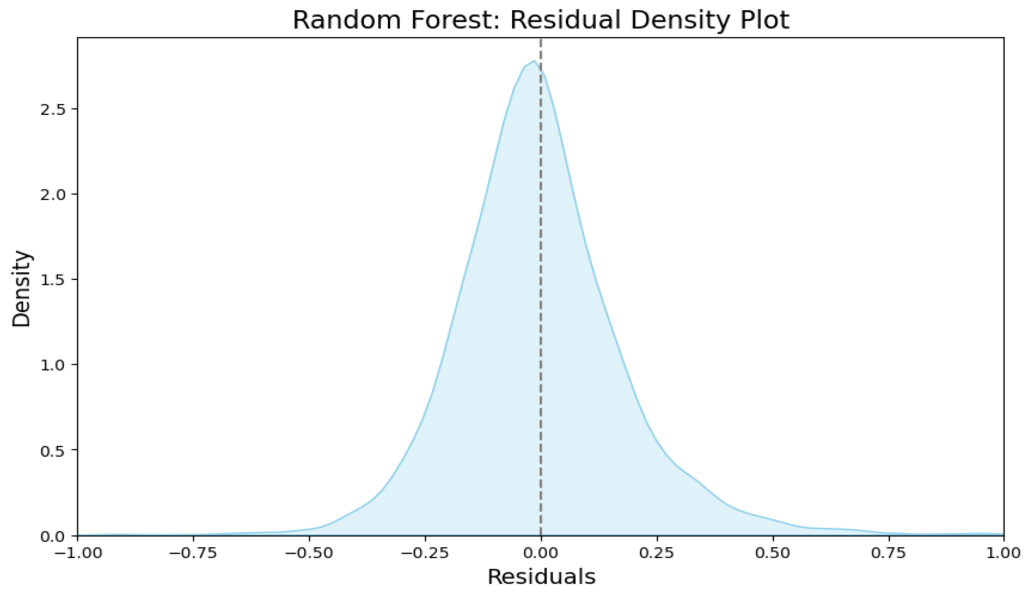
Applied the model to the test data to forecast Airbnb rental prices.

```
[18]: # Training the Random Forest regressor
forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(X_train, y_train)

# Predicting on the test set
y_pred = forest_reg.predict(X_test)

# Calculating mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)

# Calculating R2 score
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error (MSE):", mse)
print("R2 Score:", r2)
```

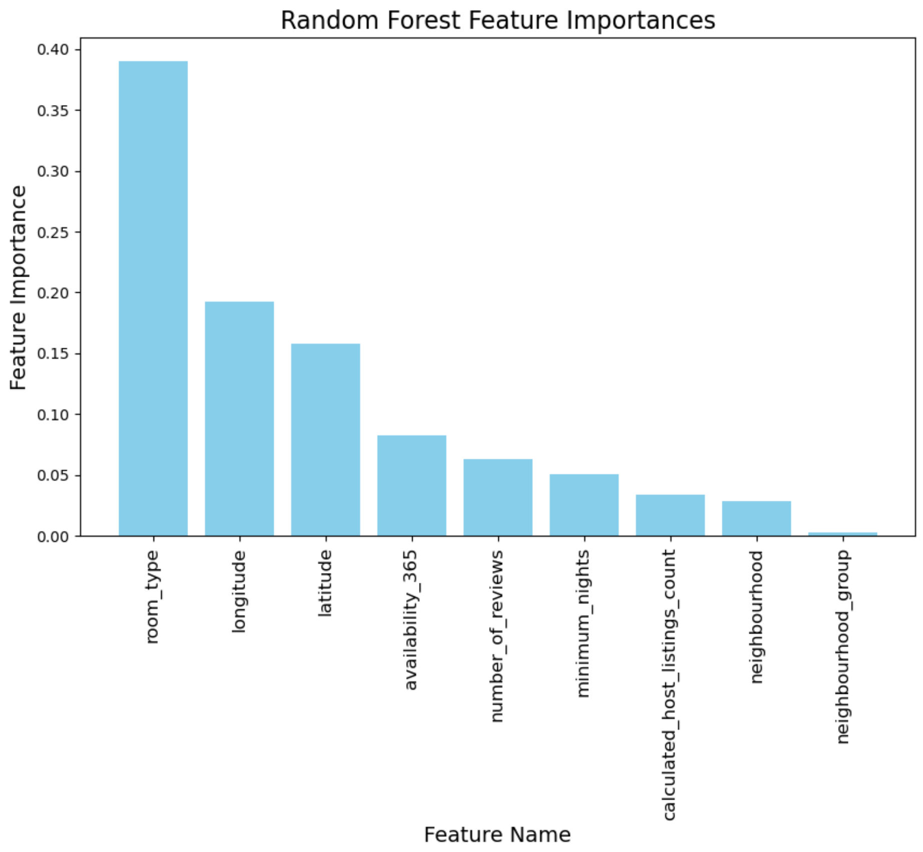
```
Mean Squared Error (MSE): 0.03631839115640159
R2 Score: 0.5988891661665556
```

The model achieved a Mean Squared Error (MSE) of approximately 0.0363, indicating the predictions were close to the true rental prices on average.

The R2 score was around 0.599, meaning that about 60% of the variance in Airbnb rental prices was explained by the model.



The residual density plot for the Random Forest model illustrates the distribution of residuals (the differences between the predicted and actual values). The density of the residuals is centered around zero and exhibits a bell-shaped curve, indicating that the model predictions are generally close to the actual values with a symmetrical distribution of errors. This symmetry suggests the model does not consistently overestimate or underestimate the Airbnb rental prices. The concentration of the density near the zero line also points to the accuracy of the Random Forest model. However, the tails of the distribution slightly suggest the presence of some larger errors, which could be outliers or instances where the model's predictions deviate more from the actual values.



The graph provides a quantified depiction of feature importances derived from a Random Forest model, which was developed to predict accommodation prices. These feature importance values represent the relative contribution of each input variable to the predictive power of the model.

The 'room_type' feature emerges as the most influential, with an importance score of approximately 0.40. This indicates a substantial impact on the accommodation price, implying that the type of room is a primary factor customers consider when evaluating price.

Following 'room_type', the geographical coordinates, 'longitude' and 'latitude', hold significant weight, with importance scores in the range of 0.20 and 0.15 respectively. The model suggests that location is nearly as critical as the room type in affecting price.

The 'availability_365' feature, which likely represents the number of days an accommodation is available for booking throughout the year, also plays a notable role with an importance score just above 0.10. It is clear that the more available an accommodation is, the more it may influence its price, potentially due to demand and seasonality effects.

Lesser but still meaningful contributions come from 'number_of_reviews', 'minimum_nights', 'calculated_host_listings_count', and 'neighbourhood', each with importance values ranging from just above 0.05 to just below 0.10. These features collectively suggest that customer feedback, minimum stay requirements, host activity, and the specific neighborhood play nuanced roles in price determination.

In conclusion, the Random Forest model indicates that both intrinsic property characteristics (like room type) and location details (specifically, the precise geographical positioning) are pivotal in forecasting accommodation prices. As a recommendation, stakeholders should prioritize these factors in their pricing strategy and in improving the model's predictive accuracy by ensuring these features are accurately measured and incorporated.

7. Ridge Regression:

Purpose of Regularization in Regression Models:

Regularization serves as a safeguard against overfitting in regression models. Overfitting occurs when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This is particularly a problem in regression models that have many features which may lead to a model that is too complex. Regularization addresses this by introducing a penalty term to the loss function used to train the model. The alpha parameter in Ridge Regression controls the strength of this penalty and, consequently, the degree of regularization applied.

[23]:
```python
# Define alpha values
alphas = [0.1, 0.5, 1.0 , 3.0]

# Initialize lists to store R2 scores for each alpha value
alpha_r2_scores = []

# Loop through each alpha value
for alpha in alphas:
    # Initialize Ridge Regression model with the current alpha value
    ridge = Ridge(alpha=alpha)

    # Perform cross-validation with 5 folds
    r2_scores = cross_val_score(ridge, X_train, y_train, cv=5, scoring='r2')

    # Store mean R2 score for the current alpha value
    alpha_r2_scores.append(np.mean(r2_scores))

# Find the best alpha value based on the maximum R2 score
best_alpha_index = np.argmax(alpha_r2_scores)
best_alpha = alphas[best_alpha_index]
best_r2_score = alpha_r2_scores[best_alpha_index]
print("Best Alpha:", best_alpha)
print("R2 Score for Best Alpha:", best_r2_score)
```

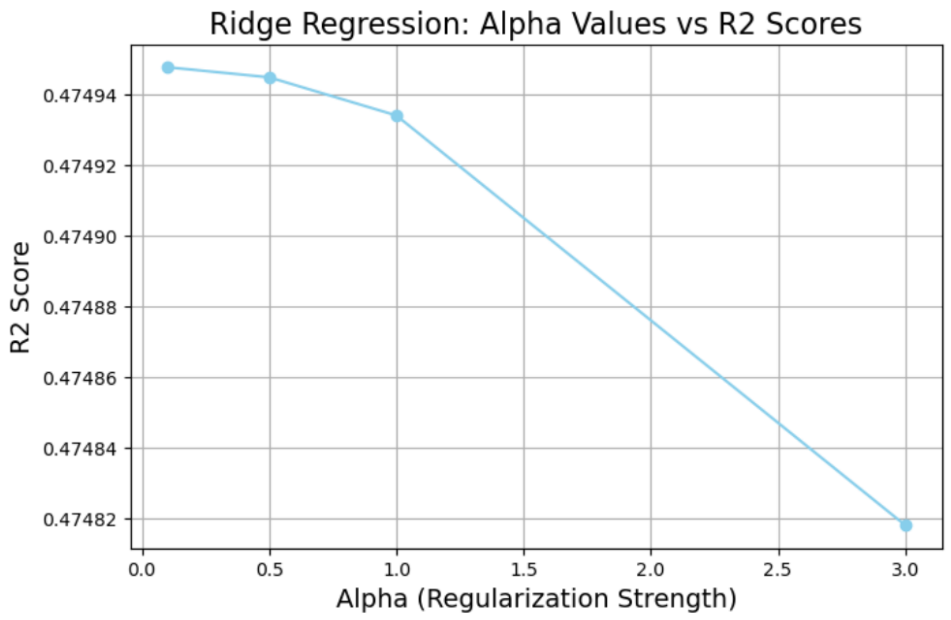
```
Best Alpha: 0.1
R2 Score for Best Alpha: 0.47494768218383027
```

Alpha Parameter Tuning:

We explored different values of alpha to understand its effect on the performance of our Ridge Regression model. The chosen values were 0.1, 0.5, 1.0, and 3.0. The rationale behind varying alpha is to determine a value that allows the model to maintain a balance between fitting the training data accurately and maintaining the ability to generalize well to unseen data.

The alpha value that achieved the highest mean R2 score was 0.1, with a score of approximately 0.475. This suggests that a small amount of regularization, as controlled by an alpha of 0.1, is the most beneficial for our Ridge Regression model. It provides a model that is complex enough to accurately predict the dependent variable but simple enough to avoid overfitting.

Based on these results, we recommend using an alpha value of 0.1 for the Ridge Regression model in production. This value is shown to optimize the predictive performance of the model according to the R2 score obtained from cross-validation.



The graph illustrates the effect of the alpha parameter on the R2 score within a Ridge Regression model, showing a discernible trend where the R2 score reaches its peak at the lowest alpha value tested, 0.1. As the alpha value incrementally increases to 3.0, there is a steady decline in the R2 score, moving from slightly below 0.47494 down to approximately 0.47482. This pattern underscores the impact of regularization strength on the model's predictive accuracy, with higher levels of regularization corresponding to a decrease in R2 score.

The visual data analysis suggests that when using Ridge Regression for this specific dataset, a lower alpha value (0.1) is preferable. This level of regularization helps to maintain the predictive accuracy of the model without overcomplicating it, which aligns with the principle that simpler models generalize better.