Airbnb Price Predictor

Group - 5

Team Members:

1. **Divya Kota-11708036**
2. **Manyam Viswa bhavitha-11708239**
3. **Vandana Kosuri – 11703470**
4. **Vadapalli Sasikumar – 11703034**
5. **Harshini Sai Sangadi – 11637776**

GitHub: https://github.com/DivyaKota22/Airbnb-Price-Predictor-Group5

Project Description:

The Airbnb Price Predictor project is a creative effort that aims to create an advanced prediction model for estimating Airbnb listing prices. This project aims to identify the complex elements affecting rental costs on the Airbnb platform by employing a large and varied dataset. It delivers precise and dependable price forecasts for Airbnb rentals by incorporating a variety of features, such as location attributes, property categories, host information, and historical booking data.

Goals and Objectives:

Motivation:

The increasing demand for a precise instrument that can precisely forecast rental costs has been brought about by the rise in demand of Airbnb as a substitute for conventional accommodation alternatives. In a market where prices vary according to a multitude of criteria, such as property attributes and geographical specifics, such a tool is important. The "Airbnb Price Predictor" project's goal is to use machine learning and data analytics to make Airbnb pricing more transparent and predictable for the advantage of hosts and guests inside the network's ecosystem.

Significance:

The project is highly valuable in a number of important areas:

- For Visitors: It assists in selecting economical and well-informed hotel options.
- Hosts: It helps them choose affordable and trustworthy marketing prices.
- For Industry Researchers: It offers valuable information about factors influencing lease expenses and pricing trends, which is essential for prediction and evaluating the market.
- Regarding the Collective Economy: It advances our knowledge of the mechanics of pricing in collaborative activity networks.

Objectives:

To create a trustworthy predictive model based on a variety of factors that can reliably anticipate Airbnb rental pricing. Examine the ways in which various elements—like property type, location, host details, and booking history—affect rental pricing. Offer a user-friendly tool that hosts can use to optimize prices and prospective tenants may use to estimate prices. Add to the corpus of knowledge about statistical analysis and modeling predictions in a collaborative market.

Features:

1. Based on data Insights: Deriving valuable insights regarding pricing from an extensive dataset.
2. Multi-Factor Analysis: Taking into account a wide range of factors, such as neighborhood, host verification status, property type, and geographic location.
3. Advanced Predictive Models: Applying and assessing different machine learning models for price estimation, such as decision trees Regression and linear regression.

4. Focused on users Approach: Emphasizing the model's useful implementation for Airbnb guests as well as hosts.
5. Industry Pattern Evaluation: This helps those in the real estate and hospitality sectors make strategic decisions by giving a broad picture of pricing patterns in the Airbnb industry.

Related Work (Background):

The research and advancements in the domains of housing market evaluation, machine learning, and modeling for prediction serve as a solid foundation for this project. This background section examines pertinent research, developments in technology, and the growth approaches that have shaped this initiative's evolution.

Estimation of Prices Problems in Machine Learning: With significant benefits, machine learning has been used more and more in price prediction tasks. Prices in different industries have been forecasted using a variety of techniques, from straightforward linear regression to intricate ensemble methods. With regard to Airbnb, these techniques are especially useful because of the platform's pricing system, which is impacted by a variety of variable fluctuations.

Research of Airbnb Market Behavior: A number of investigations have examined the variables, including location popularity, seasonal trends, and host credibility, that influence Airbnb price. These observations are beneficial to this project since they can be included into a more comprehensive model that takes into consideration both the peculiarities of the Airbnb industry and conventional real estate aspects.

Past Prediction Pricing Methods for Airbnb Listing: Earlier attempts were made to develop predictive pricing algorithms especially for Airbnb listings. By finding important predictive characteristics and proving that machine learning can be used to anticipate prices in this market, these models have set the foundation for future research. With the use of a large dataset and cutting-edge methodologies, the "Airbnb Price Predictor" project seeks to improve and expand these models' reliability and usability.

Dataset:

This project makes use of an extensive Airbnb dataset that was obtained from Kaggle. The dataset offers extensive knowledge into the variables affecting rental costs and is specifically made to support the creation of predictive models for Airbnb pricing.

The following are the main characteristics of the dataset:

- ID : An individual identifying number that is given to every dataset element.
- NAME: The name or title assigned to the real estate listing.
- Host ID: The property's host's unique identification.
- Host identification Verified: Returns a "Yes" or "No" depending on whether the host's identification has been confirmed.
- Name of the person or organization that is hosting the property is the host name.
- Neighborhood Group: The neighborhood of the property is categorized or classified.
- Neighborhood: The precise location of the property inside a locale.
- Latitude: The property's geographical latitude coordinate.
- Longitude: The property's geographical longitude coordinate.

- Country: The nation in which the asset is located.
- Country Code: The country code that corresponds to the location of the property.

Detail design of Features

ID acts as a distinct identifier for every dataset listing. mostly employed to maintain and track data, guaranteeing its integrity. used to identify listings apart in a unique way, guaranteeing correct reference and the absence of duplicates in the dataset.

NAME is a textual descriptor that gives the listing a title and a succinct explanation. Using text analysis, this feature can extract sentiment or theme information that could be related to pricing strategies.

Another distinct numerical identity, the host ID refers to the individual hosting the listing. enables the inclusion of host-specific characteristics in the model, such as a host's possible influence.

A category variable called "Host Identity Verified" indicates whether or not the host's identification has been confirmed. Pricing may be impacted by this binary characteristic (Yes/No), since verified hosts may charge more because of a greater trust factor.

The host's name may be utilized for secondary analysis or to personalize data, but it is not directly used for price prediction.

The neighborhood feature improves the dataset's based on location resolution and makes hyper-local market price analysis possible.

Analysis:

The goal of the Airbnb Price Predictor project is to find patterns, trends, and connections between different attributes and listing prices by methodically going over the dataset. This is accomplished by combining inferential statistics, visual research, and statistical analysis, all of which provide information for the ensuing modeling procedure.

Summary Evaluation: Determine the mean, median, mode, standard deviation, and variation for numerical characteristics such as cost, minimum number of nights, and reviews.

Distributed Analysis: Examine the statistical patterns of significant factors to determine the skewness, spread, and existence of outliers. For instance, looking at the price range might show whether there are extreme values or whether the majority of listings are concentrated inside a particular price range.

Heatmaps for Correlation: Make heatmaps to illustrate the relationship among numerical features. This makes it easier to determine whether variables are strongly or weakly correlated with the price.

Geographic Maps: To find geographic trends and pricing hotspots, plot items on a map utilizing latitude and longitude information.

Regression Analyses: While accounting for a variety of factors, use regression models to measure the link between price and other variables.

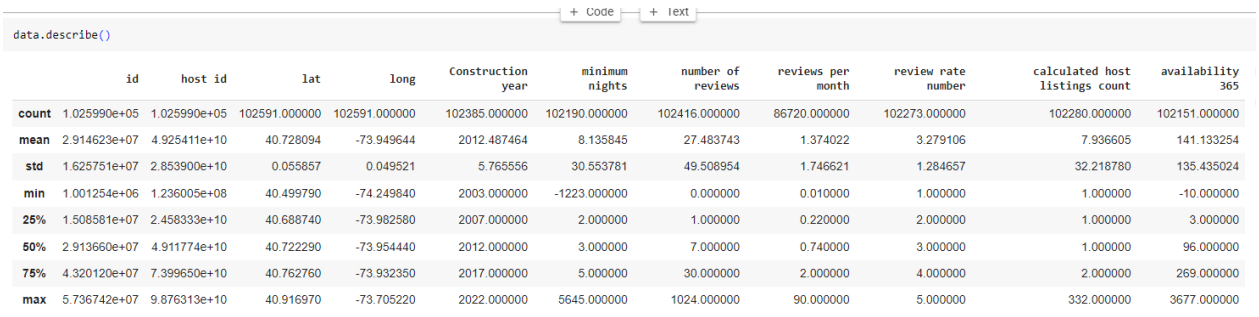
Model Efficiency Metrics: R-squared, RMSE, and MAE are a few examples of metrics that can be used to assess how well different predictive models perform. To choose the model that predicts prices most accurately, compare them.

Implementation:

The portion of the Airbnb Price Prediction project when statistical findings are converted into a useful model for prediction is called the Implementation phase. Preparing the data, choosing the model, training, and testing are all involved.

Exploratory Data Analysis:

Below is statistical analysis of mean, std, min and max by using describe() function.



+ Code + Text

```
data.describe()
```

	id	host id	lat	long	Construction year	minimum nights	number of reviews	reviews per month	review rate number	calculated host listings count	availability 365
count	1.025990e+05	1.025990e+05	102591.000000	102591.000000	102385.000000	102190.000000	102416.000000	86720.000000	102273.000000	102280.000000	102151.000000
mean	2.914623e+07	4.925411e+10	40.728094	-73.949644	2012.487464	8.135845	27.483743	1.374022	3.279106	7.936605	141.133254
std	1.625751e+07	2.853900e+10	0.055857	0.049521	5.765556	30.553781	49.508954	1.746621	1.284657	32.218780	135.435024
min	1.001254e+06	1.236005e+08	40.499790	-74.249840	2003.000000	-1223.000000	0.000000	0.010000	1.000000	1.000000	-10.000000
25%	1.508581e+07	2.458333e+10	40.688740	-73.982580	2007.000000	2.000000	1.000000	0.220000	2.000000	1.000000	3.000000
50%	2.913660e+07	4.911774e+10	40.722290	-73.954440	2012.000000	3.000000	7.000000	0.740000	3.000000	1.000000	96.000000
75%	4.320120e+07	7.399650e+10	40.762760	-73.932350	2017.000000	5.000000	30.000000	2.000000	4.000000	2.000000	269.000000
max	5.736742e+07	9.876313e+10	40.916970	-73.705220	2022.000000	5645.000000	1024.000000	90.000000	5.000000	332.000000	3677.000000

The diagram shows a summary of a pandas Data Frame used for Airbnb listing analysis, with 102,599 items and 26 columns. Each column's non-null values and data types are displayed.

```
[ ] data.shape

    (102599, 26)

    data.info()

    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 102599 entries, 0 to 102598
    Data columns (total 26 columns):
     #   Column                          Non-Null Count   Dtype
    ---  ------                          --------------   -----
     0   id                              102599 non-null  int64
     1   NAME                            102349 non-null  object
     2   host id                         102599 non-null  int64
     3   host_identity_verified          102310 non-null  object
     4   host name                       102193 non-null  object
     5   neighbourhood group             102570 non-null  object
     6   neighbourhood                   102583 non-null  object
     7   lat                             102591 non-null  float64
     8   long                            102591 non-null  float64
     9   country                         102067 non-null  object
     10  country code                    102468 non-null  object
     11  instant_bookable                102494 non-null  object
     12  cancellation_policy             102523 non-null  object
     13  room type                       102599 non-null  object
     14  Construction year               102385 non-null  float64
     15  price                           102352 non-null  object
     16  service fee                     102326 non-null  object
     17  minimum nights                  102190 non-null  float64
     18  number of reviews               102416 non-null  float64
     19  last review                     86706 non-null   object
     20  reviews per month               86720 non-null   float64
     21  review rate number              102273 non-null  float64
     22  calculated host listings count  102280 non-null  float64
     23  availability 365                102151 non-null  float64
     24  house_rules                     50468 non-null   object
     25  license                         2 non-null       object
    dtypes: float64(9), int64(2), object(15)
    memory usage: 20.4+ MB
```

This image shows the result of applying the isnull().sum() method to a pandas DataFrame; it shows how many values in every column of the Airbnb listings dataset are missing.

```
data.isnull().sum()

id                                  0
NAME                              250
host id                             0
host_identity_verified            289
host name                         406
neighbourhood group                29
neighbourhood                      16
lat                                 8
long                                8
country                           532
country code                      131
instant_bookable                  105
cancellation_policy                76
room type                           0
Construction year                 214
price                             247
service fee                       273
minimum nights                    409
number of reviews                 183
last review                     15893
reviews per month               15879
review rate number                326
calculated host listings count    319
availability 365                  448
house_rules                     52131
license                        102597
dtype: int64
```

Below code that removes particular columns from a panda DataFrame, bringing the total number of columns down to 20, and fills in the blanks in the 'house_rules' column with the text "Not stated."

Dropping the specified columns

+ Code + Text

```
columns_to_drop = ["host name", "license", "last review", "reviews per month", "country", "country code"]

data.drop(columns=columns_to_drop, axis=1, inplace=True)
data.shape
```

```
(102599, 20)
```

The program below transforms the columns labeled "price" and "service fee" into strings, eliminates leading individuals, eliminates whitespace, removes commas, and then converts it back to numerical values. It handles mistakes by forcing erroneous parsing to result in NaN.

Convert 'price' and 'service fee' columns to string and convert back to numeric

```
price = data['price'].astype(str).str[1:]
price = price.str.strip()

price = price.str.replace(",", "")

svc = data['service fee'].astype(str).str[1:]
svc = svc.str.strip()

svc = svc.str.replace(",", "")
price = pd.to_numeric(price, errors='coerce')
svc = pd.to_numeric(svc, errors='coerce')
```

Data Cleaning - The section of code cleans the data by assigning zeros to NaN values in a number of columns and transforming them to the proper data types.

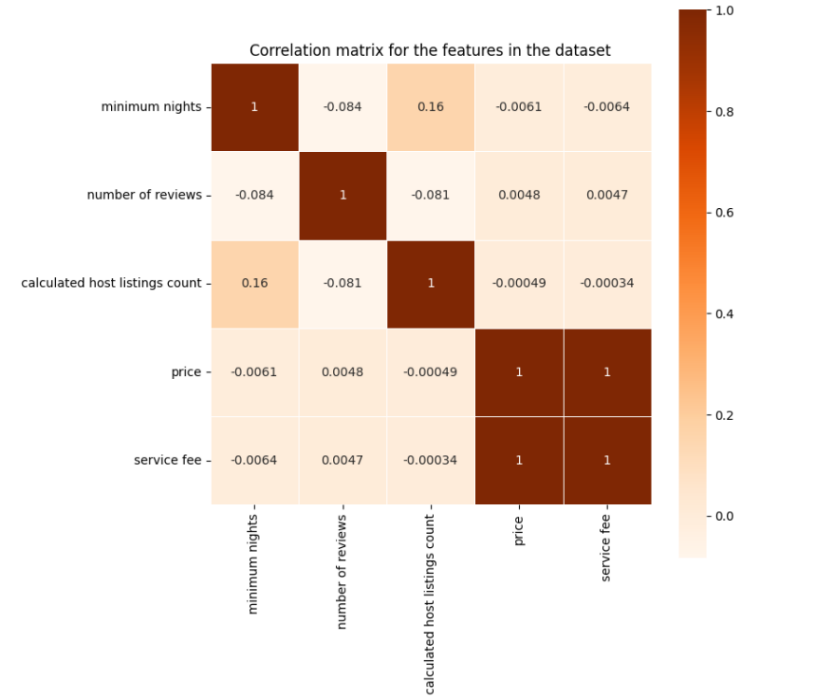
Data Cleaning

Handle NaNs before conversion

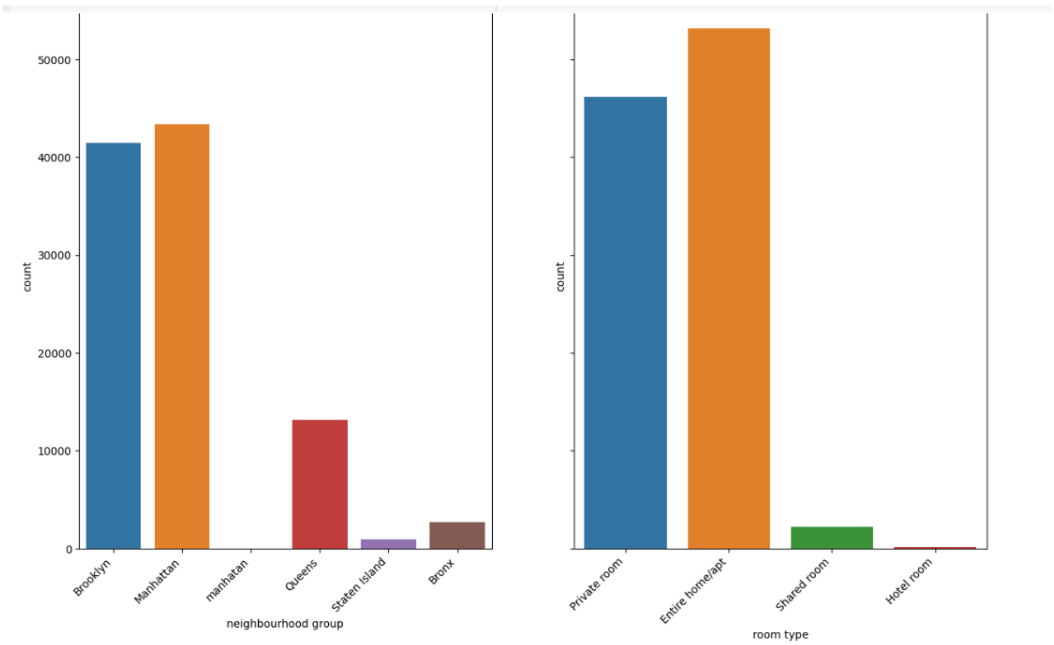
```
data["Construction year"].fillna(0, inplace=True)
data["minimum nights"].fillna(0, inplace=True)
data["number of reviews"].fillna(0, inplace=True)
data["review rate number"].fillna(0, inplace=True)
data["calculated host listings count"].fillna(0, inplace=True)
data["availability 365"].fillna(0, inplace=True)

# Convert to integers
data["Construction year"] = data["Construction year"].astype(int)
data["minimum nights"] = data["minimum nights"].astype(int)
data["number of reviews"] = data["number of reviews"].astype(int)
data["review rate number"] = data["review rate number"].astype(int)
data["calculated host listings count"] = data["calculated host listings count"].astype(int)
data["availability 365"] = data["availability 365"].astype(int)

data["instant_bookable"] = data["instant_bookable"].astype(bool)
```

Data Visualization:

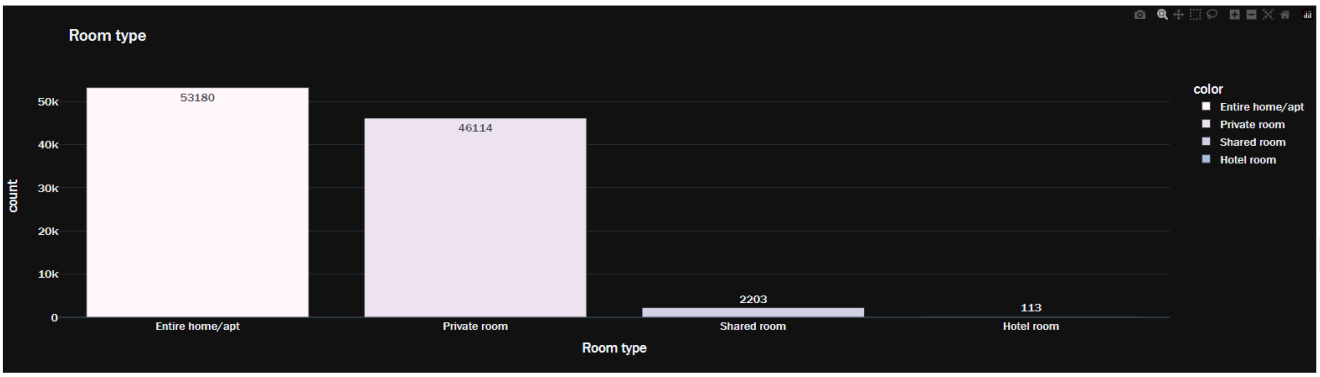
The below heatmap shows the link strengths between pairs of features from an Airbnb dataset, such as "minimum nights," "number of reviews," "estimated host listings count," "price," and "service fee." It is a heatmap of a relationship matrix.



The below two bar graphs are displayed as one demonstrates the breakdown of Airbnb listings by neighborhood group, with Brooklyn and Manhattan getting the greatest counts; the other demonstrates the number of different room types accessible, with the most prevalent types being "Entire home/apt" and "Private room."



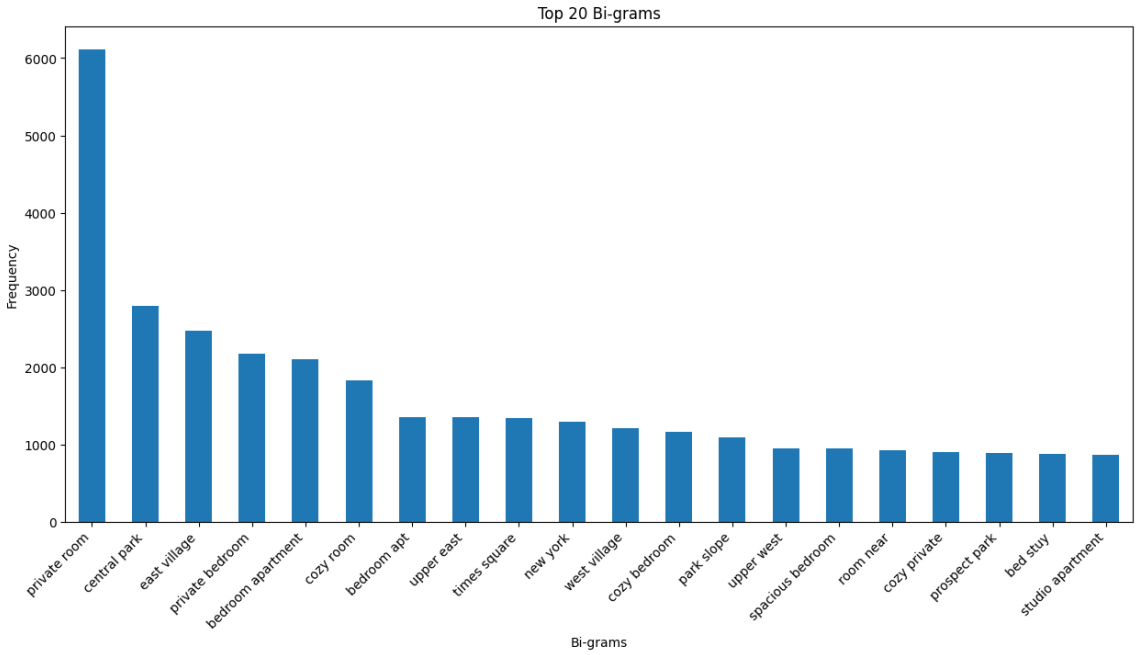
Another way of representation, the graph, which is a bar chart, displays the quantity of Airbnb listings broken down by kind of room. The most popular category is "Entire home/apt," which follows by "Private room," "Shared room," & "Hotel room."



The image below creates a word cloud using a dataset's 'NAME' column to show the most frequently occurring terms in Airbnb listing names.

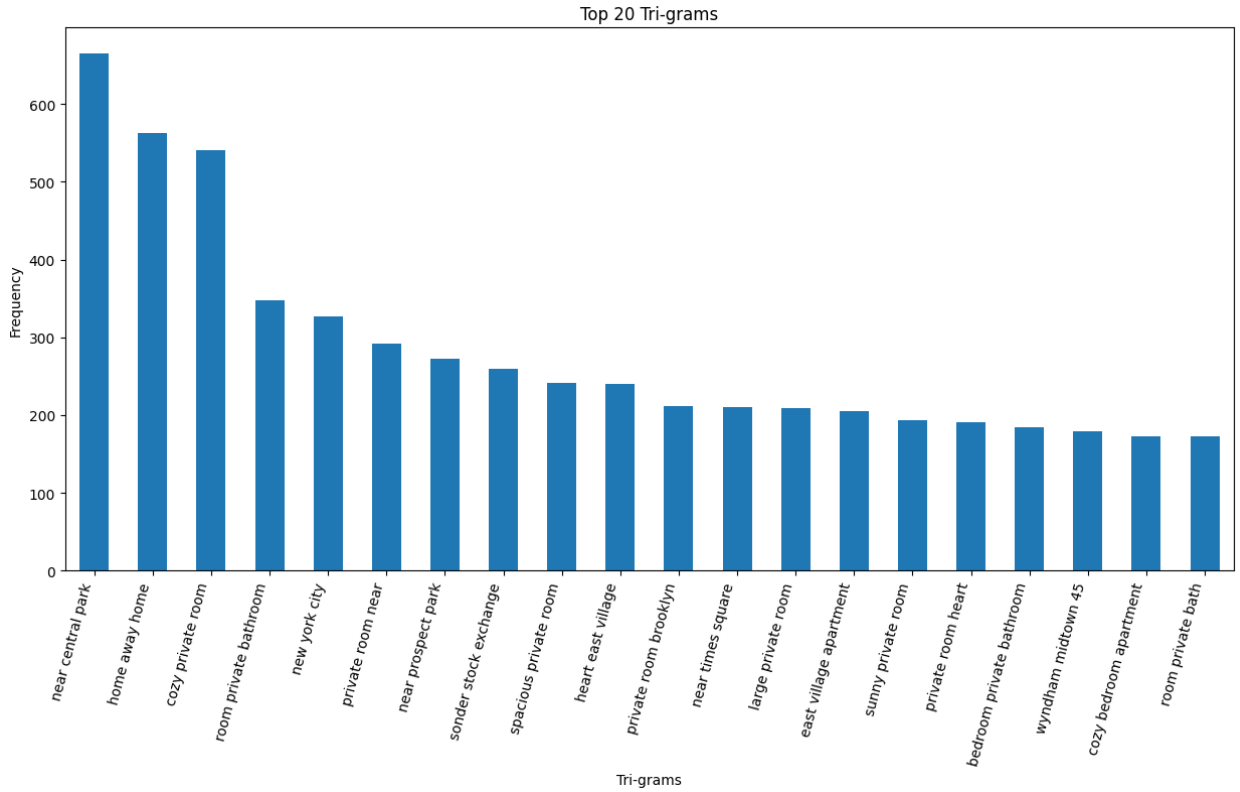
Bi-gram

The picture is a "Top 20 Bi-grams" bar chart. A collection of bi-grams are shown along the x-axis, and the y-axis shows the frequencies that correspond to them. Word pairings are represented by the bi-grams, and the height of each bar shows the frequency of each bi-gram in the dataset. The bi-grams are arranged with the most frequent bi-gram on the left and the least frequent of the top 20 on the right, in descending order of frequency. The frequency of the first bi-gram, which seems to be the most common, is noticeably higher than that of the other bi-grams, with the other bi-grams exhibiting a progressive decrease in frequency from left to right. Each bi-gram is listed on the "Bi-grams" x-axis, while the "Frequency" y-axis has numerical values that represent the frequency of each bi-gram. The precise text of the bi-grams and their precise frequencies are not easily readable due to the image's size and resolution, but it is clear that there is a wide variety of frequencies, and the chart offers a brief visual comparison of these.



Tri-gram

A bar chart labeled "Top 20 Tri-grams" is displayed in the picture. The most frequent three-word phrases (tri-grams) are represented by their respective frequency values along the y-axis, and their occurrences are visualized along the x-axis in this graphic. The tri-grams are displayed in the chart in decreasing order of occurrence. The tallest bar represents the most prevalent tri-gram, with a steep drop to the next one and an increasingly steady frequency reduction from left to right for the bars that follow.



Feature Engineering

By assigning numerical values to categorical variables (neighborhood group, room type, and cancelation policy) in a DataFrame, which illustrates feature engineering.

Feature Engineering

```
datainmodel["neighbourhood group_mapping"] = datainmodel["neighbourhood group"].map({"Brooklyn": 1,
                                                                                     "Manhattan": 2,
                                                                                     "Staten Island": 3,
                                                                                     "Queens": 4,
                                                                                     "Bronx": 5})
datainmodel["room type_mapping"] = datainmodel["room type"].map({"Private room":1,
                                                                 "Entire home/apt":2,
                                                                 "Shared room":3,})
datainmodel["cancellation_policy_mapping"] = datainmodel["cancellation_policy"].map({"strict": 1,
                                                                                     "moderate": 2,
                                                                                     "flexible": 3})
```

Splitting the data

The dataset was divided into training and test sets utilizing the scikit-learn train test split function, with 20% set aside for testing and a random state specified for repeatability.

```
y = pd.DataFrame(datainmodel["price"])
X = datainmodel.drop(["price", "neighbourhood group","cancellation_policy", "room type"], axis= 1)
```

Splitting the data

```
[ ]
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size= 0.20, random_state= 42)
```

Splitting the data

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size= 0.20, random_state= 42)
```

Model Building

Linear Regression:

The statistical technique known as linear regression involves fitting an equation with observed data in order to model the connection between the dependent variable and one or more variables that are independent. The main objective is to identify the line of equality that fits the data point the best and minimizes the sum of the squared discrepancies between the line's anticipated values and the actual values.

Uses the scikit-learn module to initialize a Linear Regression model, fit it to the training set of data, and provide predictions for the test set of data.

Linear Regression

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

Decision Tree:

A non-linear predictive modeling method called a decision tree model divides the information into groups recursively according to the feature input values. The information is divided into branches at decision nodes, which stand for options. Leaf nodes with anticipated output values are located at the terminus of these branches. Although this model is simple to understand and visualize, if it is not adjusted properly, it may become overfit.

Decision Tree is imported the scikit-learning regression model trains on training data, produces prediction on test data, and then resets to a random state for consistency.

Decision Tree

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score

dt_model = DecisionTreeRegressor(random_state=42)

dt_model.fit(X_train, y_train)

y_pred_dt = dt_model.predict(X_test)
```

Random Forest:

The Random Forest Regressor begins with a random_state parameter. As soon as the information and parameter values are the same, the random_state option acts similarly to setting a seed in a random function, enabling the method to yield consistent results throughout multiple runs. R2 is the R-squared score. Regression model accuracy is commonly assessed using these metrics. The process of bringing the error measurement back to the identical scale as the target variable involves first calculating the mean square error (MSE) and then deriving the root mean squared error (RMSE) by taking the square root of MSE.

```
from sklearn.ensemble import RandomForestRegressor

# Initialize the Random Forest Regressor
rf_model = RandomForestRegressor(random_state=42)

# Train the model
rf_model.fit(X_train, y_train.values.ravel())

# Predicting on the test set
y_pred_rf = rf_model.predict(X_test)

# Evaluate the model
mse_rf = mean_squared_error(y_test, y_pred_rf)
rmse_rf = np.sqrt(mse_rf)
r2_rf = r2_score(y_test, y_pred_rf)
```

Neural Network:

Google created the robust free machine learning package TensorFlow, and on top of it is the Keras API, which is used to create and train neural networks. A Dense layer is the initial layer to be inserted (keras.layers.Dense). Every neuron in this layer is linked to every other neuron in the layer above, indicating that this layer is completely connected. Utilizing the activation function of ReLU, the first Dense layer comprises 128 units (neurons). The structure of the model is

summarized using the model.summary() function, which also displays the overall number of parameters and the total amount of parameters in each layer.

Neural Network Model

```
from sklearn.neural_network import MLPRegressor

# Initialize the MLP Regressor
mlp_model = MLPRegressor(random_state=42)

# Train the model
mlp_model.fit(X_train, y_train.values.ravel())

# Predicting on the test set
y_pred_mlp = mlp_model.predict(X_test)

# Evaluate the model
mse_mlp = mean_squared_error(y_test, y_pred_mlp)
rmse_mlp = np.sqrt(mse_mlp)
r2_mlp = r2_score(y_test, y_pred_mlp)
```

Preliminary Results:

Model Evaluation

A linear regression model's performance measures are displayed in its assessment output. Measuring the variation between that the model predicts and the values, the Mean Squared Error (MSE) is a metric that takes the average of the squares of the mistakes. The mean square error (MSE) in this instance is around 2.02, indicating that the model's predictions are, on average, 2.02 units off from the true values.

A more concrete understanding of the model's prediction error is provided by the RMSE 1.42, which indicates that the model's predictions are, on average, 1.42 units off from the actual values. Since it shows greater model performance with less variance from the observed data, a

lower RMSE is usually preferred. A model's ability to explain almost all of the variance in the dependent variable is shown by an R-squared value that is extremely close to 1, precisely around 0.9999. This indicates an excellent fit between the model and the data. But with an R2 value so near to ideal, it's equally critical to exercise caution to avoid overfitting.

```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")
```

```
Mean Squared Error (MSE): 2.0180597163816425
Root Mean Squared Error (RMSE): 1.420584286968444
R-squared (R2): 0.9999817802892063
```

Here the preliminary results for the decision tree model, The mean squared variance between the actual results that were observed and the outcomes that the model anticipated is indicated by the Mean Squared Error (MSE), which is roughly 2.79. Root Mean Squared Error (RMSE) is the average difference between the expected and actual values is represented by the square root of the mean square error (MSE), which is around 1.67. The statistical measure of how well the data match the fitted regression line is called R-squared, with 1.0 denoting a perfect fit. R^2 is roughly 0.9999. According to these measures, the Decision Tree model's predictive accuracy on the provided test data is extremely high. On the other hand, an R2 value this close to 1 may suggest possible overfitting to the training set, much like in the evaluation of the linear regression model. This ought to be further examined using extra verification methods like cross-validation.

```
# Evaluate the model
mse_dt = mean_squared_error(y_test, y_pred_dt)
rmse_dt = np.sqrt(mse_dt)
r2_dt = r2_score(y_test, y_pred_dt)

print(f"Decision Tree - Mean Squared Error (MSE): {mse_dt}")
print(f"Decision Tree - Root Mean Squared Error (RMSE): {rmse_dt}")
print(f"Decision Tree - R-squared (R2): {r2_dt}")
```

```
Decision Tree - Mean Squared Error (MSE): 2.7941943288413267
Decision Tree - Root Mean Squared Error (RMSE): 1.6715843768237746
Decision Tree - R-squared (R2): 0.9999747730891412
```

Hyper Parameter Tuning:

A grid search across the specified hyperparameters is carried out by a GridSearchCV object that has been put up. It assesses each set of hyperparameters using 5-fold cross-validation and chooses the set of parameters that has the lowest negative mean square error. The optimal hyperparameters are found and printed once the grid search has been fitted to the training set. These optimal parameters are then used to build a new Decision Tree model, which is then trained using the training set.

Hyper Parameter Tuning

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score

# Initialize the Decision Tree Regressor
dt_model = DecisionTreeRegressor(random_state=42)

# Define the grid of hyperparameters to search
param_grid = {
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 4, 10],
    'max_features': ['auto', 'sqrt', 'log2', None],
    'criterion': ['mse', 'friedman_mse', 'mae']
}

# Set up the grid search
grid_search = GridSearchCV(estimator=dt_model, param_grid=param_grid,
                           cv=5, n_jobs=-1, scoring='neg_mean_squared_error', verbose=2)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_
print("Best parameters:", best_params)

# Train the model with the best parameters
best_dt_model = DecisionTreeRegressor(**best_params, random_state=42)
best_dt_model.fit(X_train, y_train)
```

By choosing the best hyperparameters for the job, the algorithm effectively optimizes a Decision Tree Regressor using grid search, improving predicted accuracy on the test data.

Cross- Validation:

Five folds are used for cross-validation (n_folds = 5). This implies that there are five subgroups of the initial training data, and that the model is trained and assessed five times, with a distinct subset being used as the validation set and the rest used for learning. determines the average RMSE and the RMSE standard error for every folds. The standard deviation shows the amount of success fluctuation across several folds, whereas the mean the RMSE estimates the algorithm's overall prediction accuracy.

Cross Validation

```python
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
import numpy as np

model = LinearRegression()

# Define the number of folds for cross-validation
n_folds = 5
scores = cross_val_score(model, X_train, y_train, scoring='neg_mean_squared_error', cv=n_folds)
mse_scores = -scores
rmse_scores = np.sqrt(mse_scores)
mean_rmse = np.mean(rmse_scores)
std_rmse = np.std(rmse_scores)

print(f"Cross-validated RMSE scores: {rmse_scores}")
print(f"Mean RMSE: {mean_rmse}")
print(f"Standard Deviation of RMSE: {std_rmse}")

```

Project Management:

Work Completed:

References:

Airbnb dataset refer from,

https://www.kaggle.com/datasets/arianazmoudeh/airbnbopendata/data

James, G., Tibshirani, R., Witten, D., and Hastie, T. (2013). An Overview of Statistical Education. Sprout.

Proserpio, D., Zervas, G., and Byers, J. W. (2017). The Sharing Economy is Growing: Measuring Airbnb's Effect on the Hotel Sector.

Pedregosa and associates (2011). Scikit-learn: Python for Machine Learning, JMLR 12