Airbnb Analytical Report

Introduction and Business Question

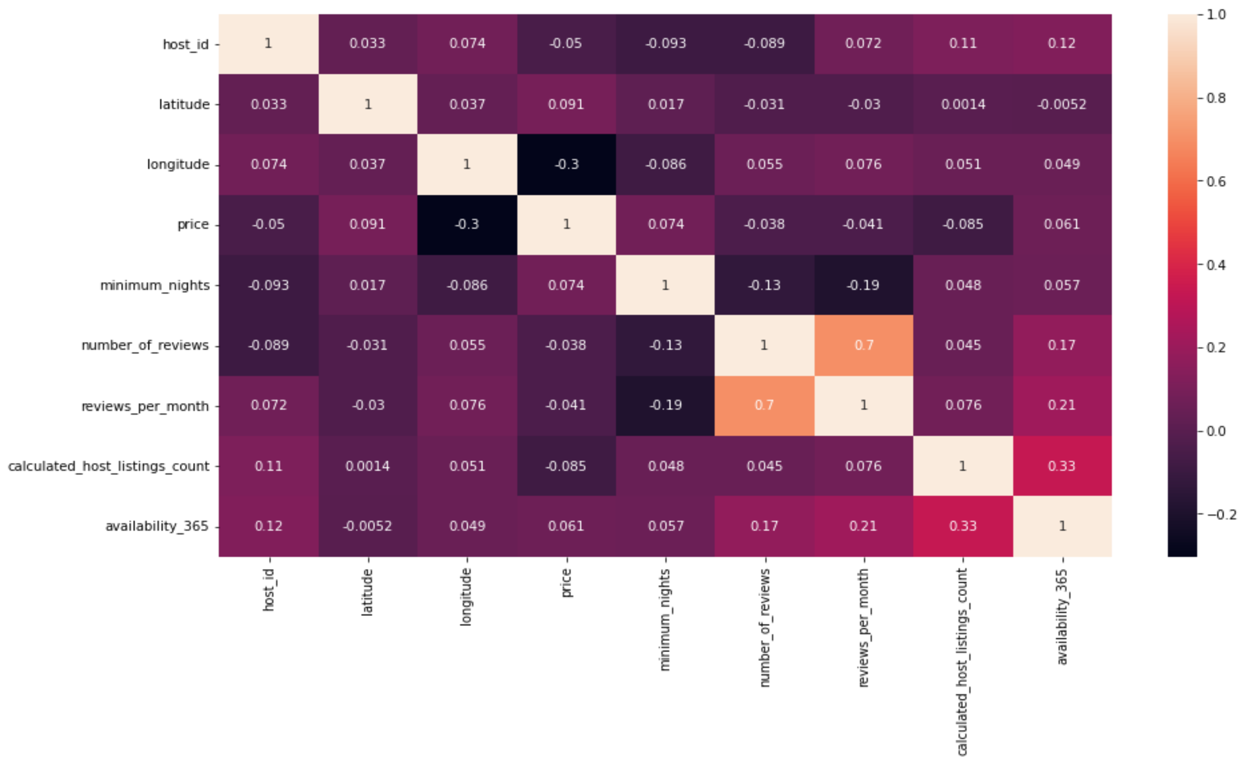
This report poses the business question: what is the biggest determinant of price for listings? This will be answered through an exploratory data analysis of price in relation to other variables, followed by utilisation of the machine learning algorithms Random Forest Regressor and XGBoost Regressor. These will identify features that determine price and thus predict price. Having a model that can predict price can then be used in the app. For example, for each new listing, the model can provide an estimated price for the listing based on its features.

Exploratory Data Analysis

Data Preprocessing

During the data preprocessing step, null values have been filled and some of the unnecessary columns such as 'id', 'name', 'host_name', and 'last_review' have been removed as there is no need to analyse these, especially details of the host as this is personal information. Further, for the Machine Learning models, 'host_id' and 'reviews_per_month' were dropped as they were found not to have feature importance.

We can see from the correlation graph in figure 1, there are no strong correlations between variables other than 'reviews' and 'reviews_per_month' which is to be expected as they are both counting reviews.



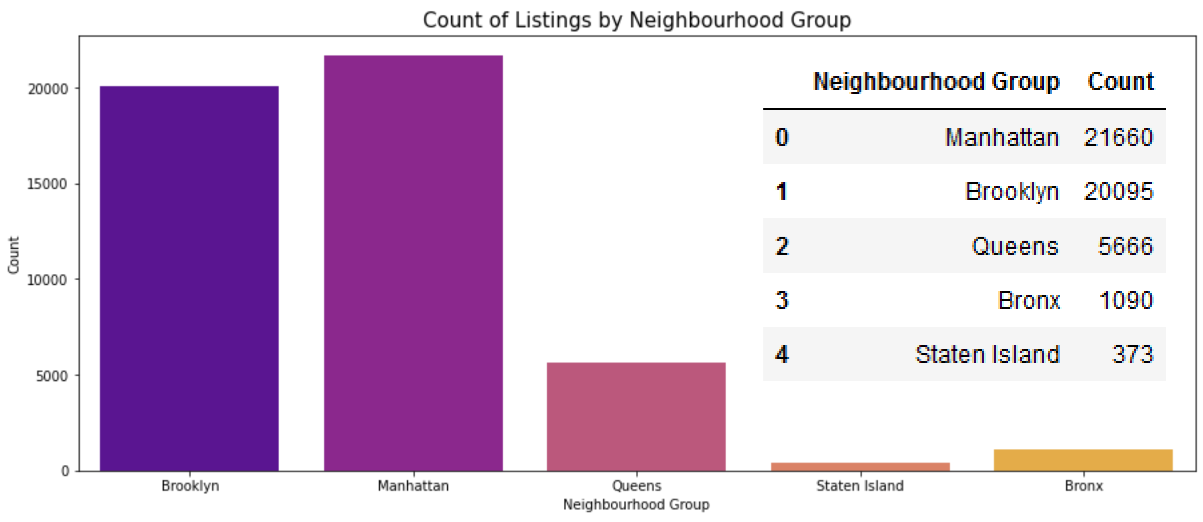
	host_id	latitude	longitude	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host_listings_count	availability_365
host_id	1	0.033	0.074	-0.05	-0.093	-0.089	0.072	0.11	0.12
latitude	0.033	1	0.037	0.091	0.017	-0.031	-0.03	0.0014	-0.0052
longitude	0.074	0.037	1	-0.3	-0.086	0.055	0.076	0.051	0.049
price	-0.05	0.091	-0.3	1	0.074	-0.038	-0.041	-0.085	0.061
minimum_nights	-0.093	0.017	-0.086	0.074	1	-0.13	-0.19	0.048	0.057
number_of_reviews	-0.089	-0.031	0.055	-0.038	-0.13	1	0.7	0.045	0.17
reviews_per_month	0.072	-0.03	0.076	-0.041	-0.19	0.7	1	0.076	0.21
calculated_host_listings_count	0.11	0.0014	0.051	-0.085	0.048	0.045	0.076	1	0.33
availability_365	0.12	-0.0052	0.049	0.061	0.057	0.17	0.21	0.33	1

Figure 1 - Correlation Graph

Variation and Covariation

First, the categorical variables are considered and then their relationship with price and location. For the purposes of this report and based on the results of the machine learning models discussed below, ‘neighbourhood group’, location (‘latitude’ and ‘longitude’ and ‘room_type’ are the most likely to contain features that impact price and so shall be examined below.

Neighbourhood Group and Price



	Neighbourhood Group	Count
0	Manhattan	21660
1	Brooklyn	20095
2	Queens	5666
3	Bronx	1090
4	Staten Island	373

Figure 2 – Neighbourhood Count charts

Manhattan has the most listings, followed closely by Brooklyn and then there is a large drop for the other neighbourhoods (see Figure 2). This distribution could be due to the Queens and Bronx areas being primarily residential and the population of Staten Island being small.

When looking at price, the data has been limited to anything below the 95th percentile to remove extreme values. The histogram in figure 3 displays a positive skewness with most listings around 100 and below. Looking at figure 4, the same observation can be found for each feature, except for Manhattan having a more even distribution of price and thus may have a bigger impact on price.

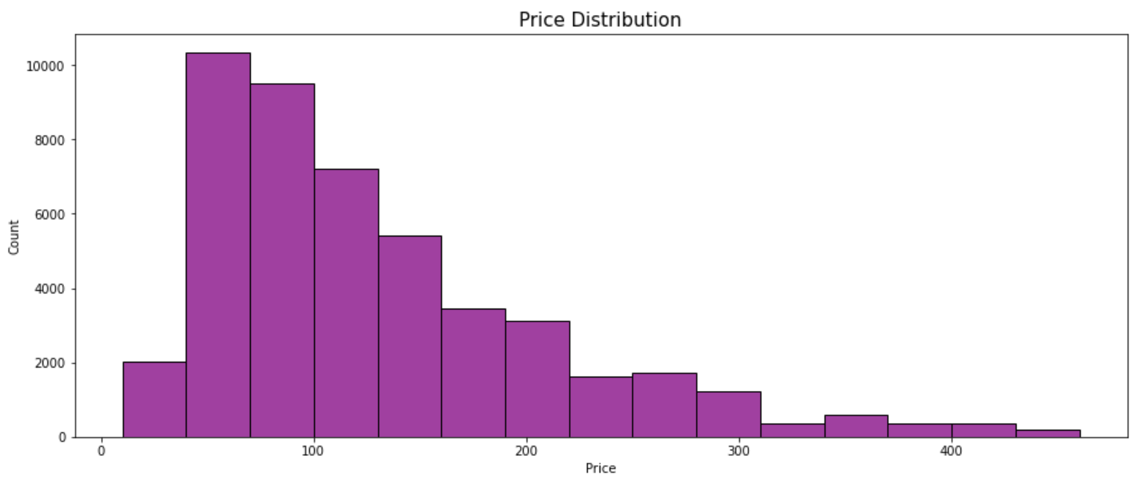


Figure 3 - Price histogram

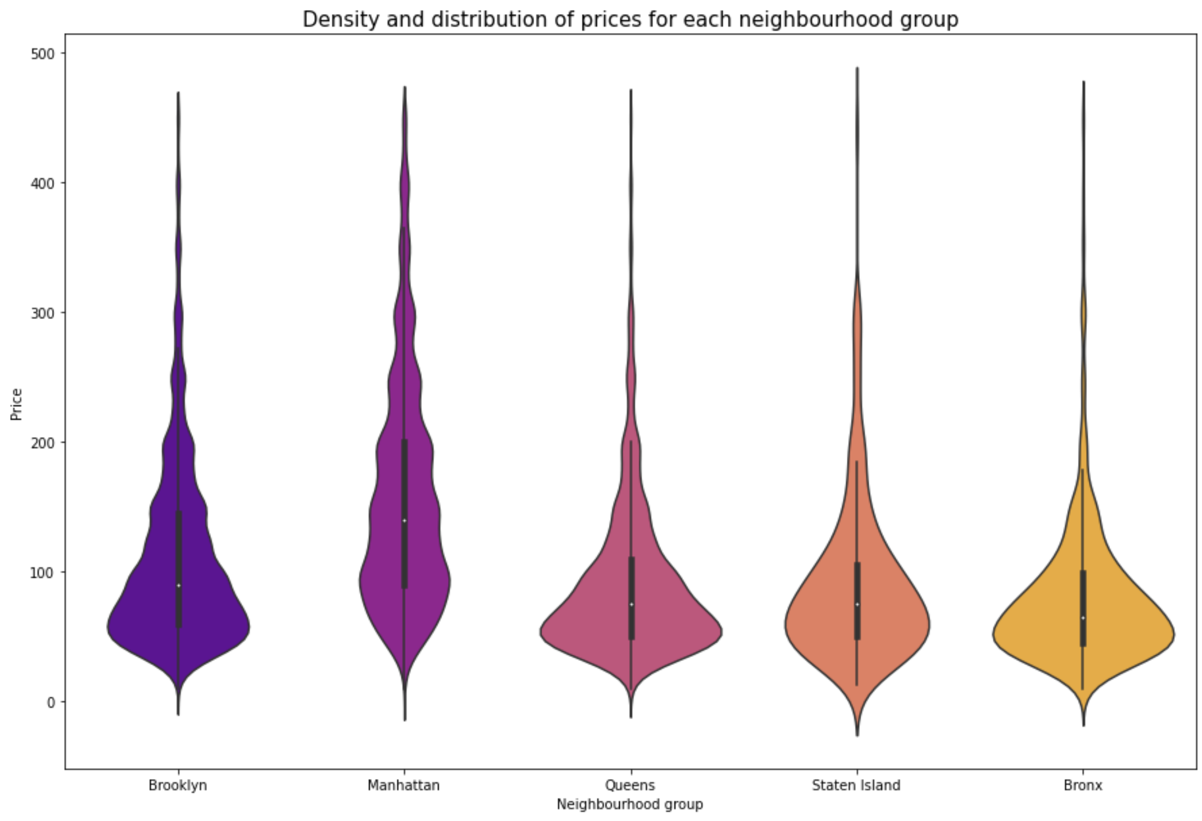


Figure 4 - Violin chart for price and neighbourhood group

Location, Neighbourhood Group, and Price

When looking at location in *figure 5*, this conforms to neighbourhood groups as they are simply groups of locations. The scatter shows the distribution of price by location, and as corroborated above, there is a concentration of high price listings in Manhattan.

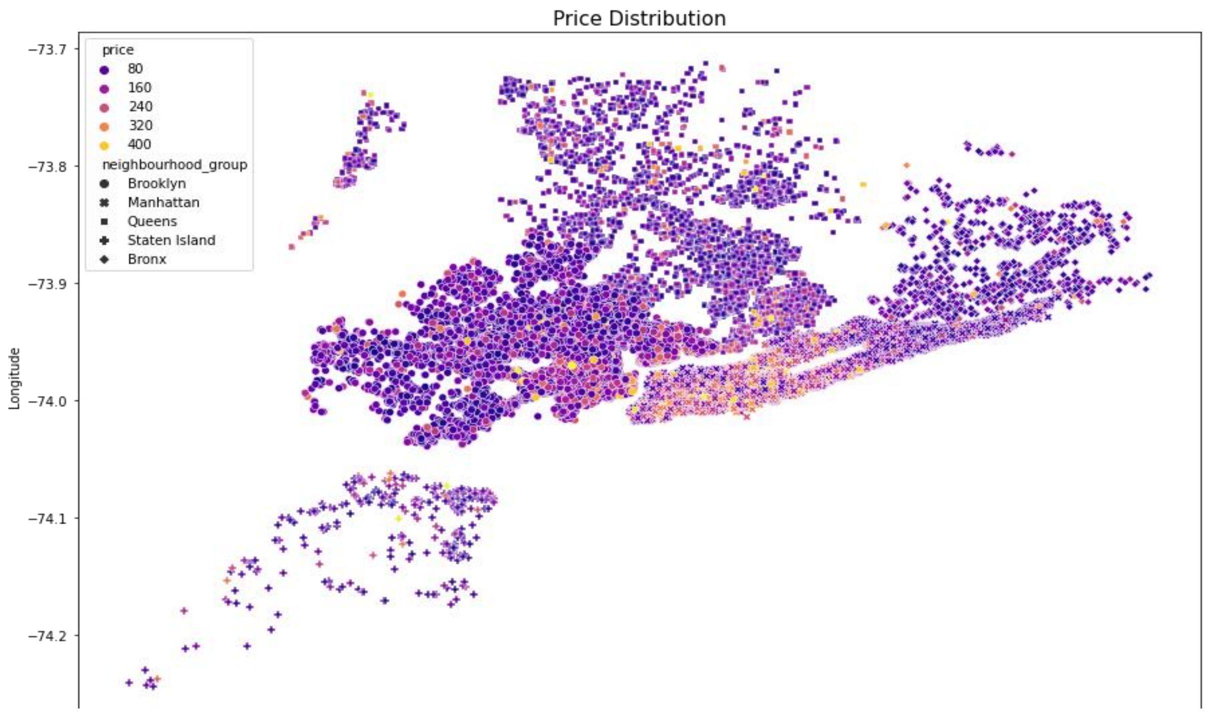


Figure 5 - Scatter Plot showing Location and Neighbourhood Group and Price

Room Type and Price

Figure 6 shows that shared room listings are very low in comparison to entire homes/apartments and private rooms, with entire homes/apartments being the highest.

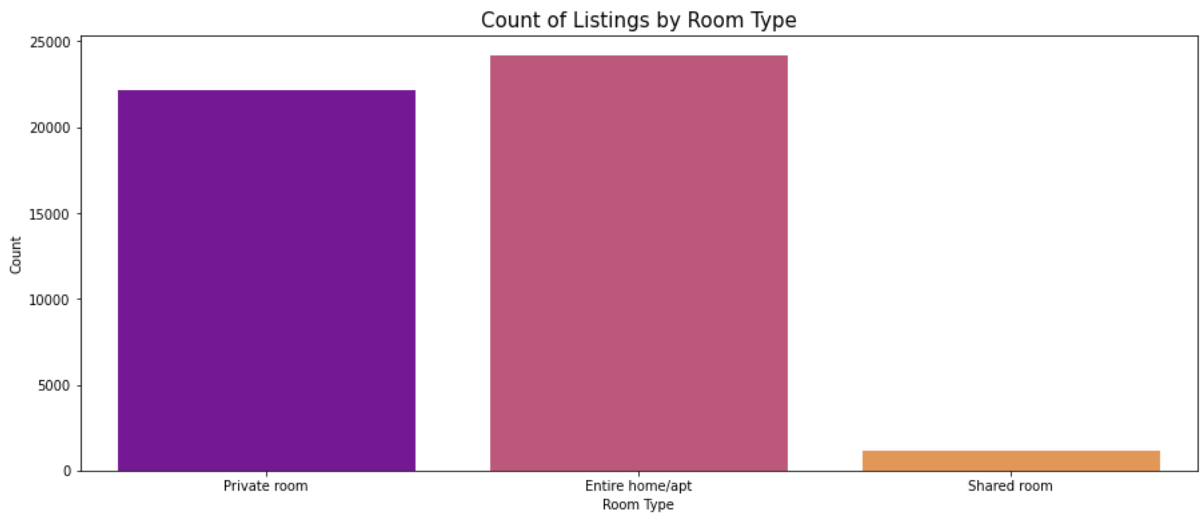


Figure 6 - Room price count plot

In *figure 7*, it is observable that entire homes/apt has a higher median than the other room types, with its first quartile higher than the third quartile of both shared room and private room. This shows that this feature may be a strong determiner of price.

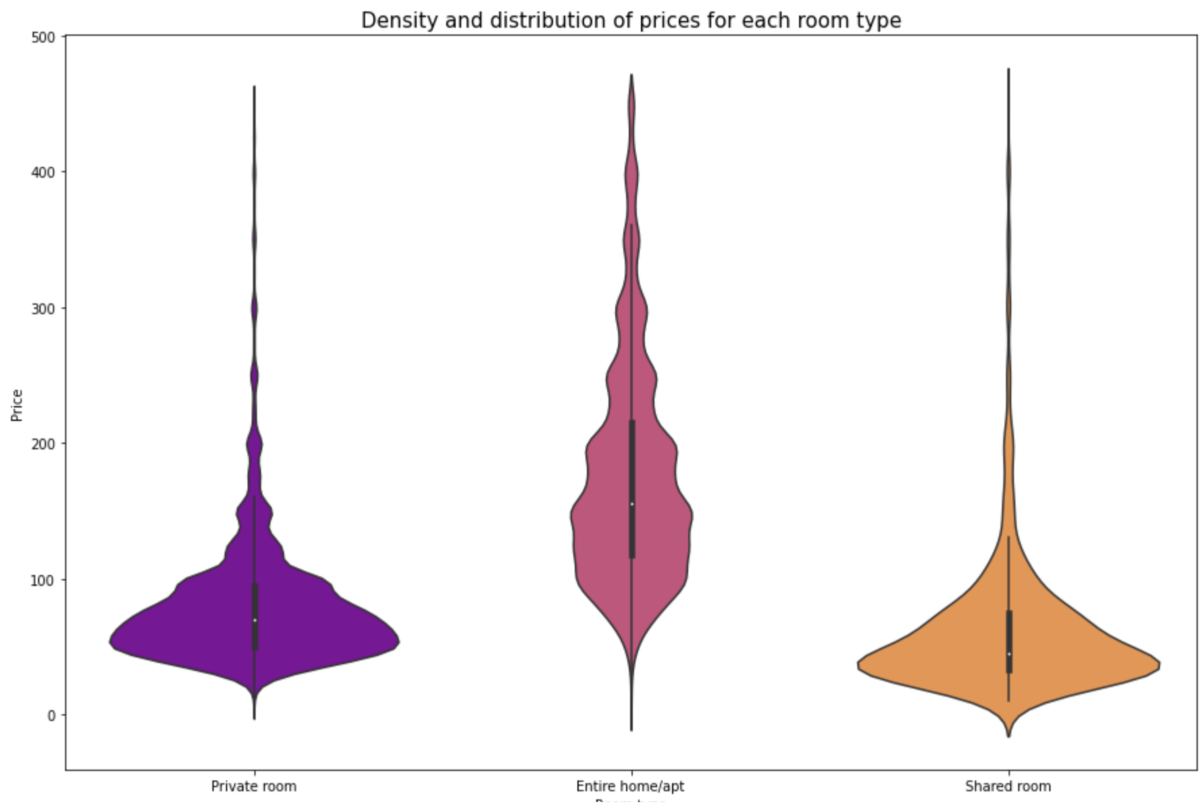


Figure 7 - violin chart of price and room type

Machine Learning

Purpose of the Models

Of the models that were developed, the ones that showed the highest accuracy were XGBoost Regressor and Random Forest.

The intended purpose of these models is to understand the most important features from the dataset that affect price and then predicting price determined by the other variables in the dataset.

Evaluating XGBoost Regressor Model

The initial model has an R^2 score of 0.635 on the test data and thus predicts 64% of the relationship of price and the other variables.

The SHAP summary chart in *Figure 8* lists the most important features in descending order determined by the model. Thus, it shows that room type and location are the most important variables in determining listing price since features from these variables are the highest.

The next steps to iterate on this model is to keep the most important features and remove unimportant features. Additional variables are likely to further enhance the model and increase accuracy. For example, a variable for amenities the listing provides, nearby landmarks, and how many people the listing can accommodate.

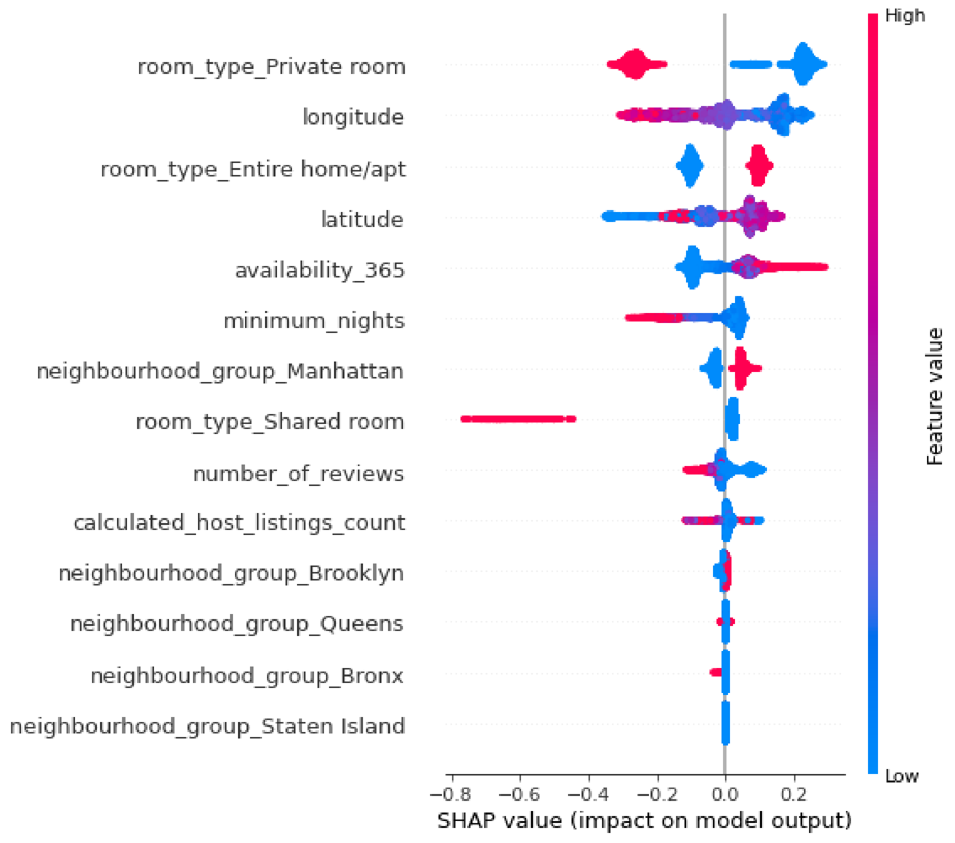


Figure 8 – XGBoost Regressor SHAP Summary Chart

Evaluating Random Forest Regressor Model

This model has a slightly better R^2 than the XGBoost model with a score of 0.640.

The SHAP summary chart in *figure 9* shows a similar outcome to the XGBoost model. The model ranks features in a different order but the conclusion remains the same: room type and location are the most important features in determining price.

As above, this model would also benefit from the additional variables suggested for the XGBoost model.

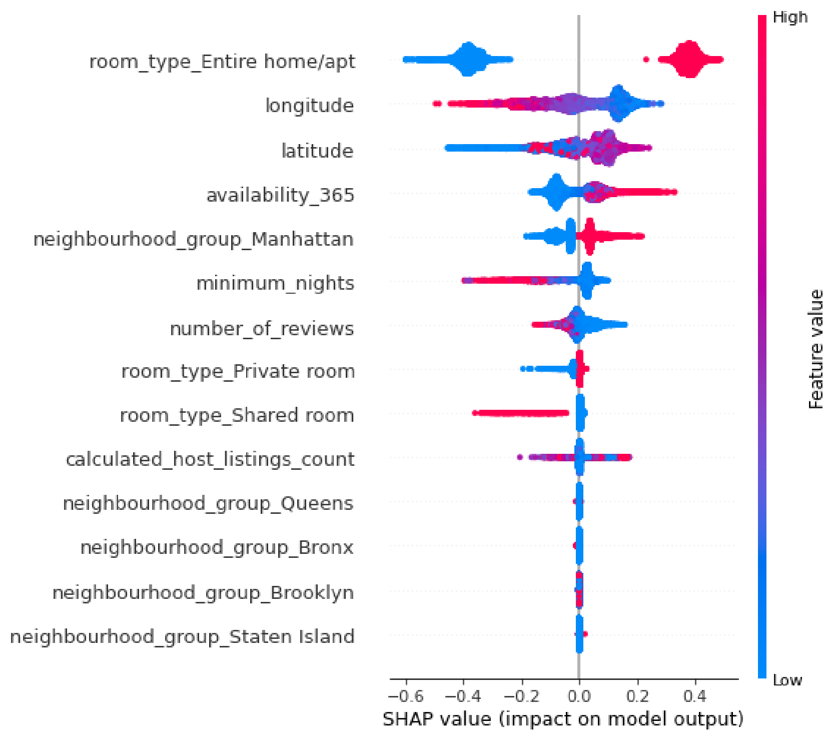


Figure 9 - Random Forest Regressor SHAP Summary Chart

Conclusion

This report has analysed and determined the variables that contain the most important features (room type and location) and provided a model to predict the price of listings which should be iterated on with new variables to improve accuracy. Additionally, by examining the distributions of the most important variables from the exploratory analysis combined with the findings of the models, Airbnb can look to encourage listings from hosts with these features to increase profit.

Appendices

APPENDIX A: EDA and ML for determining most important features and predicting price.

```
# # EDA

### Package Import

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

get_ipython().run_line_magic('matplotlib', 'inline')

import seaborn as sns

import warnings

import xgboost

import scipy.stats as stats

import pylab

warnings.filterwarnings('ignore')

# ## Importing Dataset

#importing dataset

airbnb=pd.read_csv("AB_NYC_2019.csv")

#looking at head

airbnb.head(5)

#looking at number of rise to guage size of dataset - notably some values missing

airbnb.shape

#data types

airbnb.info

# ## Data Preprocessing

#removing duplicates

airbnb.duplicated().sum()

airbnb.drop_duplicates(inplace=True)

#checking how many null values in dataset for each column
```

```
airbnb.isnull().sum()

#drop columns such as id, host name, and last review as there is no need to analyse the names of hosts. The last_review column is a date. This won't exist if thee number of reviews is 0 and will not be relevant to analysis. The name of the listing is also to be dropped as we will not be conducting NLP.

airbnb.drop(['id','name','host_name','last_review'], axis=1, inplace=True)

airbnb.head(5)

#replace null values with 0

airbnb.fillna({'reviews_per_month':0}, inplace=True)

airbnb.reviews_per_month.isnull().sum()

airbnb.isnull().sum()

airbnb.dropna(how='any',inplace=True)

#remove any listings with price = 0

airbnb = airbnb[airbnb.price>0]

airbnb.describe()

# ## Visualisations for Variation and Covariation

# ### Correlation between variables

#correlation between variables

corr = airbnb.corr(method='kendall')

plt.figure(figsize=(15,8))

sns.heatmap(corr, annot=True)

airbnb.columns

# ### Neighbourhood Group and Price

#looking at unique values for neighbourhood group

airbnb['neighbourhood_group'].unique()

#looking at distribution of neighbourhood group

df=pd.DataFrame(airbnb['neighbourhood_group'].value_counts()).reset_index().rename(columns={'index': 'Neighbourhood Group','neighbourhood_group':'Count'})

df

#visualising distribution

plt.figure(figsize=(15,6))

sns.countplot(data=airbnb, x='neighbourhood_group', palette='plasma')
```

```
plt.title('Count of Listings by Neighbourhood Group', fontsize=15)

plt.xlabel('Neighbourhood Group')

plt.ylabel("Count")

#looking at stats for price by neighbourhood group

airbnb_neighbourprice = airbnb[['host_id','neighbourhood_group','price']]

airbnb_neighbourprice =
pd.pivot_table(airbnb_neighbourprice,index=['host_id'],columns='neighbourhood_group',values="pr
ice")

airbnb_neighbourprice = airbnb_neighbourprice[['Brooklyn', 'Manhattan', 'Queens', 'Staten Island',
'Bronx']]

airbnb_neighbourprice.describe()

#Price histogram

plt.figure(figsize=(15,6))

sns.histplot(data=airbnb, x='price', binwidth = 200, color='purple')

plt.title('Price Distribution', fontsize=15)

plt.xlabel('Price')

plt.ylabel("Count")

#finding 95th percentile for each neighbourhood group

airbnb_neighbourprice.quantile(0.95)

#limiting data to remove outliers above 95th percentile to get rid of outliers

airbnb = airbnb[airbnb.price < 451]

#Looking at price distribution after transformation

plt.figure(figsize=(15,6))

sns.histplot(data=airbnb, x='price', binwidth = 30, color='purple')

plt.title('Price Distribution', fontsize=15)

plt.xlabel('Price')

plt.ylabel("Count")

#looking at covariation of price and neighbourhood group

plt.figure(figsize=(15,10))

sns.violinplot(data=airbnb, x='neighbourhood_group', y='price', palette='plasma')

plt.title('Density and distribution of prices for each neighbourhood group', fontsize=15)

plt.xlabel('Neighbourhood group')
plt.ylabel("Price")

# ### Room Type and Price

#room type distribution

plt.figure(figsize=(15,6))

sns.countplot(data=airbnb, x='room_type', palette='plasma')

plt.title('Count of Listings by Room Type', fontsize=15)

plt.xlabel('Room Type')

plt.ylabel("Count")

#covariation room type and price

plt.figure(figsize=(15,10))

sns.violinplot(data=airbnb, x='room_type', y='price', palette='plasma')

plt.title('Density and distribution of prices for each room type', fontsize=15)

plt.xlabel('Room type')

plt.ylabel("Price")

plt.figure(figsize=(15,10))

ax = sns.boxplot(data=airbnb, x='neighbourhood_group', y='availability_365', palette='plasma')

plt.title('Relationship between Neighbourhood Group and Availability', fontsize=15)

plt.xlabel('Neighbourhood Group')

plt.ylabel("Availability")

plt.figure(figsize=(15,10))

sns.scatterplot(data=airbnb, x='latitude', y = 'longitude', palette='plasma', hue = "price", style =
"neighbourhood_group")

plt.title('Price Distribution', fontsize=15)

plt.xlabel('Latitude')

plt.ylabel("Longitude")

# ## Preparing Data for Machine Learning

#function to return plots for the feature

def normality(data,feature):

   plt.figure(figsize=(10,5))

   plt.subplot(1,2,1)

   sns.kdeplot(data[feature])
   plt.subplot(1,2,2)
   stats.probplot(data[feature],plot=pylab)
   plt.show()
#transforming features into normal distribution
airbnb['price_log'] = np.log(airbnb.price)
normality(airbnb,'price_log')
# # Machine Learning
y = airbnb['price_log']
x =
airbnb[['neighbourhood_group','latitude','longitude','room_type','minimum_nights','number_of_revi
ews','calculated_host_listings_count','availability_365']]
x = pd.get_dummies(x, columns=['room_type','neighbourhood_group'])
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.inspection import permutation_importance
#Split into train and test
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=212)
# ## XGBoost
# ### Hyperparameter Tuning
#Hyperparameter tuning to reduce overfitting
import xgboost as xgb
from hyperopt import STATUS_OK, Trials, fmin, hp, tpe
space={'max_depth': hp.quniform("max_depth", 3, 18, 1),
      'gamma': hp.uniform ('gamma', 1,9),
      'reg_alpha' : hp.quniform('reg_alpha', 40,180,1),
      'reg_lambda' : hp.uniform('reg_lambda', 0,1),
      'colsample_bytree' : hp.uniform('colsample_bytree', 0.5,1),
      'min_child_weight' : hp.quniform('min_child_weight', 0, 10, 1),
      'n_estimators': 180,
      'seed': 0
   }
```

```
def objective(space):
   clf=xgb.XGBRegressor(
                n_estimators =space['n_estimators'], max_depth = int(space['max_depth']), gamma = 
space['gamma'],
                reg_alpha = int(space['reg_alpha']),min_child_weight=int(space['min_child_weight']),
                colsample_bytree=int(space['colsample_bytree']))
evaluation = [( x_train, y_train), ( x_test, y_test)]
clf.fit(x_train, y_train)
pred = clf.predict(x_test)
accuracy = r2_score(y_test, pred)
print ("SCORE:", accuracy)
return {'loss': -accuracy, 'status': STATUS_OK }
trials = Trials()
best_hyperparams = fmin(fn = objective,
                   space = space,
                   algo = tpe.suggest,
                   max_evals = 100,
                   trials = trials)
#Get best hyperparameters
print(best_hyperparams)
# ### XGBoost Model
#XGBoost Regressor model
xgb_model = xgb.XGBRegressor(colsample_bytree = 0.5, gamma = 1, max_depth = 15, 
min_child_weight = 7, reg_alpha = 42, reg_lambda = 0.7936246860452796)
xgb_model.fit(x_train, y_train)
#training data score
y_pred=xgb_model.predict(x_train)
r2_score(y_train,y_pred)
#test data score
y_pred=xgb_model.predict(x_test)
r2_score(y_test,y_pred)
# ## Descision Tree
# ### Hyperparameter Tuning

#hyperparameter tuning for regurlarisation to avoid overfitting

from sklearn.tree import DecisionTreeRegressor

from matplotlib import pyplot

train_scores, test_scores = list(), list()

# define the tree depths to evaluate

values = [i for i in range(1, 21)]


# evaluate a decision tree for each depth

for i in values:

    model = DecisionTreeRegressor(max_depth=i)


# fit model on the training dataset

    model.fit(x_train, y_train)


# evaluate on the train dataset

    train_yhat = model.predict(x_train)

    train_acc = r2_score(y_train, train_yhat)

    train_scores.append(train_acc)


# evaluate on the test dataset

    test_yhat = model.predict(x_test)

    test_acc = r2_score(y_test, test_yhat)

    test_scores.append(test_acc)


# summarize progress

    print('>%d, train: %.3f, test: %.3f' % (i, train_acc, test_acc))


# plot of train and test scores vs tree depth

pyplot.plot(values, train_scores, '-o', label='Train')

pyplot.plot(values, test_scores, '-o', label='Test')
pyplot.legend()
pyplot.show()

# ### Decision Tree Model
#best max_depth is 8
dt_model = DecisionTreeRegressor(max_depth = 8)
dt_model.fit(x_train, y_train)
y_pred=dt_model.predict(x_train)
r2_score(y_train,y_pred)
y_pred=dt_model.predict(x_test)
r2_score(y_test,y_pred)
# ## Random Forest
#hyperparameter tuning for regurlarisation to avoid overfitting
from sklearn.ensemble import RandomForestRegressor
from matplotlib import pyplot
train_scores, test_scores = list(), list()

#define max depths to evaluate
values = [i for i in range(1, 11)]

#evaluate a random forest for each depth
for i in values:
   model = RandomForestRegressor(max_depth=i)

#fit model on the training dataset
   model.fit(x_train, y_train)

#evaluate on the train dataset
   train_yhat = model.predict(x_train)
   train_acc = r2_score(y_train, train_yhat)
   train_scores.append(train_acc)
```

```
#evaluate on the test dataset
    test_yhat = model.predict(x_test)
    test_acc = r2_score(y_test, test_yhat)
    test_scores.append(test_acc)

#summarize progress
    print('>%d, train: %.3f, test: %.3f' % (i, train_acc, test_acc))

#plot of train and test scores vs tree depth
pyplot.plot(values, train_scores, '-o', label='Train')
pyplot.plot(values, test_scores, '-o', label='Test')
pyplot.legend()
pyplot.show()
#max depth of 9 is best
rf_model = RandomForestRegressor(max_depth = 9)
rf_model.fit(x_train, y_train)
y_pred=rf_model.predict(x_train)
r2_score(y_train,y_pred)
y_pred=rf_model.predict(x_test)
r2_score(y_test,y_pred)
## Shap
import shap
shap.initjs()
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=212)
x_train_summary = shap.kmeans(x_train, 10)

# ## Shap - XGBoost
shap_xgb_model = xgb.XGBRegressor(colsample_bytree = 0.5, gamma = 1, max_depth = 15,
min_child_weight = 7, reg_alpha = 42, reg_lambda = 0.7936246860452796)
shap_xgb_model.fit(x_train, y_train)
```

```
explainer = shap.TreeExplainer(shap_xgb_model)

shap_values = explainer.shap_values(x_test)

shap.summary_plot(shap_values, x_test)

# ## SHAP Decision Tree

shap_dt_model = DecisionTreeRegressor(max_depth = 8)

shap_dt_model.fit(x_train, y_train)

explainer = shap.TreeExplainer(shap_dt_model)

shap_values = explainer.shap_values(x_test)

shap.summary_plot(shap_values, x_test)

# ## SHAP Random Forest

shap_rf_model = RandomForestRegressor(max_depth = 9)

shap_rf_model.fit(x_train, y_train)

explainer = shap.TreeExplainer(shap_rf_model)

shap_values = explainer.shap_values(x_test)

shap.summary_plot(shap_values, x_test)
```

APPENDIX B: Linear Regression for Availability and Price

```
%matplotlib inline
#imports
from numpy import *
import matplotlib.pyplot as plt
import pandas as pd
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.inspection import permutation_importance
df=pd.read_csv("AB_NYC_2019.csv")
print(df.head(10))
x=df.price
y=df.availability_365
plt.scatter(x, y)
plt.xlabel('price')
plt.ylabel("availability")
plt.title('price x availability')
plt.show()
#hyperparamters
learning_rate = 0.0001
initial_b = 0
initial_m = 0
num_iterations = 10
def compute_cost(b, m):
    total_cost = 0
    N = float(len(df))

    #Compute sum of squared errors
    for i in range(0, len(df)):
        x = (df['price'].values[i])
        y = (df['availability_365'].values[i])
        total_cost += (y - (m * x + b)) ** 24
    #Return average of squared error
    return total_cost/N
def gradient_descent_runner(starting_b, starting_m, learning_rate, num_iterations):
    b = starting_b
    m = starting_m
    cost_graph = []

    #For every iteration, optimize b, m and compute its cost
    for i in range(num_iterations):
        cost_graph.append(compute_cost(b, m))
        b, m = step_gradient(b, m, learning_rate)

    return [b, m, cost_graph]

def step_gradient(b_current, m_current, learning_rate):
    m_gradient = 0
    b_gradient = 0
    N = float(len(df))

    #Calculate Gradient
    for i in range(0, len(df)):
        x = (df['price'].values[i])
        y = (df['availability_365'].values[i])
        m_gradient += - (2/N) * x * (y - (m_current * x + b_current))
        b_gradient += - (2/N) * (y - (m_current * x + b_current))
    #Update current m and b
    m_updated = m_current - learning_rate * m_gradient
    b_updated = b_current - learning_rate * b_gradient
    #Return updated parameters
    return b_updated, m_updated

b, m, cost_graph = gradient_descent_runner(initial_b, initial_m, learning_rate, num_iterations)

#Print optimized parameters
print ('Optimized b:', b)
print ('Optimized m:', m)

#Print error with optimized parameters
print ('Minimized cost:', compute_cost(b, m))
plt.plot(cost_graph)
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.title('Cost per iteration')
plt.show()

#Plot dataset
print(x)
print(y)
plt.scatter(x, y)
#Predict y values
pred = m * x + b
#Plot predictions as line of best fit
plt.plot(x, pred, c='r')
plt.xlabel('price')
plt.ylabel('availability')
plt.title('price x availability')
plt.show()
```

APPENDIX C: Clustering Neighbourhood and Price

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

df = pd.read_csv("AB_NYC_2019.csv")

df.head()

columns = ['neighbourhood','price']

df = df[columns]

df.head()

df.isna().sum()

df.neighbourhood.value_counts()

len(df.neighbourhood.unique())

plt.figure(figsize = (10,5))

plt.hist(df.price,bins = 600)

plt.show()

other_than_0_mask = df.price != 0

df = df[other_than_0_mask]

original_price = df['price']

df['price'] = np.log(df['price'])

plt.figure(figsize = (10,5))

plt.hist(df.price,bins = 50)

plt.show()
```

Like many Unsupervised Learning Algorithms, it's essential to ensure that our input data is entirely numeric. In our dataset, the only categorical data pertains to neighborhoods. To address this, we will employ a technique known as target encoding. Target encoding strives to transform categorical data by incorporating information from the target variable it aims to predict. Although our primary goal may not be predicting price based on neighborhood, this method can effectively encode the data while still capturing any existing relationship between the two variables.

```
data_encoded = pd.DataFrame()

data_encoded['price'] = df['price']


means = df.groupby('neighbourhood')['price'].mean().to_dict()

data_encoded['neighbourhood_encoded'] = df['neighbourhood'].map(means)
```

```
from sklearn.covariance import EllipticEnvelope


pred_eliptic = EllipticEnvelope(contamination=0.1).fit(data_encoded).predict(data_encoded)

outlier_mask = pred_eliptic == -1

data_encoded = data_encoded[~outlier_mask]

len(data_encoded)
```

Standardization is a data preprocessing technique in statistics and machine learning that transforms numerical data to have a mean of zero and a standard deviation of one. It involves subtracting the mean of the data from each data point and then dividing by the standard deviation.

```
from sklearn.preprocessing import StandardScaler


# Standardize the 'price' column

scaler = StandardScaler()

data_standarized = scaler.fit_transform(data_encoded)

def plot_data(data,labels):


  # Plot the clusters in two dimensions

  x1 = data[:,0]

  x2 = data[:,1]


  plt.figure(figsize = (7,7))

  plt.scatter(x1, x2,

            c = labels,

            s = 1)


  plt.title('Pre-processed data and clusters')

  plt.xlabel("x_1",fontsize = 13)

  plt.ylabel("x_2",fontsize = 13)

  plt.show()


plot_data(data_standarized,None)
```

The silhouette score is a metric used to evaluate the performance of clustering algorithms. It measures how well each data point fits into its assigned cluster, based on both the distance between the point and other points in the same cluster (intra-cluster distance) and the distance between the point and points in other clusters (inter-cluster distance)

```
from sklearn.metrics import silhouette_score, calinski_harabasz_score, davies_bouldin_score


def labels_prop(labels):

  return pd.Series(labels).value_counts()


def score(data,labels):

  # Calculate silhouette score

  silhouette_avg = silhouette_score(data, labels)

  print("Silhouette Score: {:.4f}".format(silhouette_avg))


  print(labels_prop(labels))
```

K-Means is a popular unsupervised machine learning algorithm used for clustering similar data points into groups or clusters. Here's a brief overview:

Objective: The main goal of K-Means is to partition a dataset into K distinct, non-overlapping clusters, where each data point belongs to the cluster with the nearest mean (centroid).

```
from sklearn.cluster import KMeans


# Initialize the k-means model with the desired number of clusters (k)

kmeans = KMeans(n_clusters = 3)


# Fit the model to the data in the dataframe

kmeans.fit(data_standarized)


# Get the labels for each row in the dataframe

labels = kmeans.predict(data_standarized)


score(data_standarized,labels)

plot_data(data_standarized,labels)
```

```
from sklearn.cluster import DBSCAN


# Assuming you have your data standardized and combined as X_combined
# If not, please refer to the previous responses for data preprocessing


# Create a DBSCAN model
dbscan = DBSCAN(eps=0.1, min_samples=5)  # You can adjust eps and min_samples as needed


# Fit the model to your data
clusters = dbscan.fit_predict(data_standarized)


# Check the number of clusters found by DBSCAN
n_clusters = len(set(clusters)) - (1 if -1 in clusters else 0)
n_noise = list(clusters).count(-1)


print(f'Number of clusters found by DBSCAN: {n_clusters}')
print(f'Number of noise points: {n_noise}')
score(data_standarized,clusters)
from sklearn.neighbors import NearestNeighbors
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
# Fit a k-nearest neighbors model to your data
k_values = range(1, 100)  # You can adjust the range as needed
distances = []


for k in tqdm(k_values):
    neighbors_model = NearestNeighbors(n_neighbors=k)
    neighbors_model.fit(data_standarized)
    avg_distances, _ = neighbors_model.kneighbors()
    distances.append(np.mean(avg_distances[:, -1]))
```

```
# Plot the average distances for different k values
plt.figure(figsize=(10, 6))
plt.plot(k_values, distances, marker='o', linestyle='-', color='b')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Average Distance')
plt.title('Average Distance to k-Nearest Neighbors')
plt.grid(True)
plt.show()
plot_data(data_standarized,clusters)
```

A Gaussian Mixture Model (GMM) is a probabilistic model used in statistics and machine learning. It represents a dataset as a combination of multiple Gaussian distributions, each associated with a specific cluster or component. GMMs are employed for tasks such as clustering and density estimation. They capture complex data patterns by modeling the underlying probability distribution as a mixture of simpler Gaussian distributions, allowing them to handle datasets with non-uniform shapes and varying densities effectively. GMMs are characterized by parameters like means, variances, and mixing coefficients, which are estimated from the data through methods like the Expectation-Maximization (EM) algorithm.

```
import pandas as pd
import numpy as np
from sklearn.mixture import GaussianMixture

# Assuming 'neighbourhood' and 'price' are your feature columns

# Initialize and fit the GMM model
n_clusters = 7  # You can specify the number of clusters you want
gmm = GaussianMixture(n_components=n_clusters)
gmm.fit(data_standarized)

# Predict cluster assignments
cluster_assignments = gmm.predict(data_standarized)

# You can access the means and covariances of each cluster like this:
```

```
cluster_means = gmm.means_

cluster_covariances = gmm.covariances_


# You can explore the results or visualize the clusters as needed

score(data_standarized,cluster_assignments)

plot_data(data_standarized,cluster_assignments)
```

To determine the optimum number of clusters a good technique is to test for many clusters and storet a score. In this specific case, the BIC score is more appropriate.

The Bayesian Information Criterion (BIC) is a measure used in Gaussian Mixture Models (GMMs) to find the best number of clusters. It balances model fit and complexity by penalizing complex models. In GMMs, lower BIC scores indicate a better trade-off between model fit and simplicity, helping select the right number of clusters for the data.

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.mixture import GaussianMixture

from tqdm import tqdm

# Assuming you have your data in 'features'

# Fit GMM with a range of cluster numbers

num_clusters = range(1, 11)

bic_scores = []


for n in tqdm(num_clusters):

    gmm = GaussianMixture(n_components=n)

    gmm.fit(data_standarized)

    bic_scores.append(gmm.bic(data_standarized))


# Plot the BIC scores

plt.figure(figsize=(8, 4))

plt.plot(num_clusters, bic_scores, marker='o', linestyle='-', color='b')

plt.xlabel('Number of Clusters')

plt.ylabel('BIC Score')
plt.title('BIC Score vs. Number of Clusters')
plt.grid(True)
plt.show()
```

The sudden drop in the BIC score at 7 clusters suggests that the model fits the data much better at 7 clusters than at 6 or 8 clusters . This could be due to the data having a natural clustering at 7 clusters, or it could be due to the model overfitting the data at 6 or 8 clusters .

While the Gaussian Mixture Model may not exhibit a particularly high silhouette score, it stands out as the most intriguing option among all the models. This is primarily due to its unique ability to demonstrate a substantial distinction when varying the number of clusters. This suggests that the model possesses the capability to identify inherent clusters within the data, an assertion supported by the relatively low BIC score associated with seven clusters when compared to other cluster numbers.

Based on these observations, we can confidently conclude that the GMM model, configured with seven clusters, yields the most favorable results among all the models considered.

```
plot_data(data_standarized,cluster_assignments)
import pandas as pd


original_df = pd.read_csv('/content/drive/MyDrive/Fiverr/Client 7 - Lisa/AB_NYC_2019.csv')



# Create a new column 'Cluster' in the original dataset
original_df['Cluster'] = -1  # Initialize all rows with -1


# Assign clusters to the rows present in the final dataset
original_df.loc[data_encoded.index, 'Cluster'] = cluster_assignments


# Now, your original DataFrame has 'Cluster' values assigned, with -1 for deleted lines
original_df
original_df.to_csv('original_dataset_with_clusters.csv')
```

APPENDIX D: Kprototype Neighbourhoods and Availability & Neighbourhoods and Price

```
import numpy as np
import pandas as pd
import sklearn.cluster
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
import matplotlib.cm as cm
from kmodes.kmodes import KModes
from kmodes.kprototypes import KPrototypes
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler



df = pd.read_csv("AB_NYC_2019.csv")


prices = df["price"].to_numpy()
nhbs = df["neighbourhood"].to_numpy()


df2 = df[["price", "neighbourhood"]].to_numpy() #selecting parameters of interest: modular
"""
kproto = KPrototypes(n_clusters = 5, init = 'Cao', n_jobs = -1)
clusters = kproto.fit_predict(df2, categorical = 1)
pd.series(clusters).value_counts()
"""
costs = []
n_clusters = []
clusters_assigned = []


for i in range(1,10): #running for optimal k
   try:
      kproto = KPrototypes(n_clusters = i, init = 'Cao', verbose = 2)
```

```
        clusters = kproto.fit_predict(df2, categorical = 1)
        costs.append(kproto.cost_)
        n_clusters.append(i)
        clusters_assigned.append(clusters)
    except:
        print("Invalid with {i} clusters")

#optimal k determined through elbow method
fig, ax = plt.subplots()
clt = ax.plot(n_clusters, costs, marker = 'o')
plt.show()
import numpy as np
import pandas as pd
import sklearn.cluster
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
import matplotlib.cm as cm
from kmodes.kmodes import KModes
from kmodes.kprototypes import KPrototypes
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler



df = pd.read_csv("AB_NYC_2019.csv")


prices = df["price"].to_numpy()
nhbs = df["neighbourhood"].to_numpy()


df2 = df[["price", "neighbourhood"]].to_numpy() #selecting parameters of interest: modular
"""
kproto = KPrototypes(n_clusters = 5, init = 'Cao', n_jobs = -1)
```

```
clusters = kproto.fit_predict(df2, categorical = 1)
pd.series(clusters).value_counts()
"""
costs = []
n_clusters = []
clusters_assigned = []

for i in range(1,10): #running for optimal k
    try:
        kproto = KPrototypes(n_clusters = i, init = 'Cao', verbose = 2)
        clusters = kproto.fit_predict(df2, categorical = 1)
        costs.append(kproto.cost_)
        n_clusters.append(i)
        clusters_assigned.append(clusters)
    except:
        print("Invalid with {i} clusters")

#optimal k determined through elbow method
fig, ax = plt.subplots()
clt = ax.plot(n_clusters, costs, marker = 'o')
plt.show()
```

FEEDBACK: Overall a good effort to organise your work, however code samples could be further discussed in the report. The report is well organised yet, the structure could be better in terms of discussions e.g. on page 5 (purpose of the models) or in the various sections e.g. analysis of figures.

Knowledge and under- standing of the topic/ issues under consideration

There is a good demonstration of knowledge and understanding, displaying originality and a good understanding of all essential knowledge relevant to the work. The discussion of data is good, but more details could be included, especially in figures, e.g. what do we see in Figure 5? The discussion could be extended.

Application of knowledge and understanding

A very good demonstration of the application of knowledge and understanding to address the learning outcomes assessed by the assignment. Good demonstrations using diagrams. The figures require more text e.g. what do we see in Figure 3? What is the lesson learned?

Some further elaboration in terms of comments or discussions in the Appendix could highlight this contribution.

Criticality

In general you included a good set of discussions, yet there are no references. You could include references to support your discussion and analysis, for example why these visualisation types are chosen? Also why these models have been implemented? You could include references in terms of the Python libraries used.

Structure and Presentation

A very good structure and presentation, however, references should be included. Further conclusions could be more detailed, and discuss limitations of current effort.