In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
df = pd.read_csv("resturant.csv")

df.head()
```

Out[3]:

| | Restaurant ID | Restaurant Name | Country Code | City | Address | Locality | Locality Verbose | Longitude | Latitude | Cuisines | ... | Currency | Has Table booking | Has Online delivery | Is delivering now | Switch to order menu | Price range |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 6317637 | Le Petit Souffle | 162 | Makati City | Third Floor, Century City Mall, Kalayaan Avenu... | Century City Mall, Poblacion, Makati City | Century City Mall, Poblacion, Makati City, Mak... | 121.027535 | 14.565443 | French, Japanese, Desserts | ... | Botswana Pula(P) | Yes | No | No | No | 3 |
| **1** | 6304287 | Izakaya Kikufuji | 162 | Makati City | Little Tokyo, 2277 Chino Roces Avenue, Legaspi... | Little Tokyo, Legaspi Village, Makati City | Little Tokyo, Legaspi Village, Makati City, Ma... | 121.014101 | 14.553708 | Japanese | ... | Botswana Pula(P) | Yes | No | No | No | 3 |
| **2** | 6300002 | Heat - Edsa Shangri-La | 162 | Mandaluyong City | Edsa Shangri-La, 1 Garden Way, Ortigas, Mandal... | Edsa Shangri-La, Ortigas, Mandaluyong City | Edsa Shangri-La, Ortigas, Mandaluyong City, Ma... | 121.056831 | 14.581404 | Seafood, Asian, Filipino, Indian | ... | Botswana Pula(P) | Yes | No | No | No | 4 |
| **3** | 6318506 | Ooma | 162 | Mandaluyong City | Third Floor, Mega Fashion Hall, SM Megamall, O... | SM Megamall, Ortigas, Mandaluyong City | SM Megamall, Ortigas, Mandaluyong City, Mandal... | 121.056475 | 14.585318 | Japanese, Sushi | ... | Botswana Pula(P) | No | No | No | No | 4 |
| **4** | 6314302 | Sambo Kojin | 162 | Mandaluyong City | Third Floor, Mega Atrium, SM Megamall, Ortigas... | SM Megamall, Ortigas, Mandaluyong City | SM Megamall, Ortigas, Mandaluyong City, Mandal... | 121.057508 | 14.584450 | Japanese, Korean | ... | Botswana Pula(P) | Yes | No | No | No | 4 |

5 rows × 21 columns

In [5]:
```
# removing features that will inhibit model training
df.drop('Restaurant ID', axis=1, inplace=True)
df.drop('Country Code', axis=1, inplace=True)
df.drop('City', axis=1, inplace=True)
df.drop('Address', axis=1, inplace=True)
df.drop('Locality', axis=1, inplace=True)
df.drop('Locality Verbose', axis=1, inplace=True)
df.drop('Longitude', axis=1, inplace=True)
df.drop('Latitude', axis=1, inplace=True)
df.drop('Currency', axis=1, inplace=True)
df.drop('Has Table booking', axis=1, inplace=True)
df.drop('Has Online delivery', axis=1, inplace=True)
df.drop('Is delivering now', axis=1, inplace=True)
df.drop('Switch to order menu', axis=1, inplace=True)
df.drop('Price range', axis=1, inplace=True)
df.drop('Aggregate rating', axis=1, inplace=True)
df.drop('Rating color', axis=1, inplace=True)
df.drop('Rating text', axis=1, inplace=True)
df.drop('Votes', axis=1, inplace=True)
```

In [7]:
```
df.isna().sum()
```

Out[7]:
```
Restaurant Name         0
Cuisines                9
Average Cost for two    0
dtype: int64
```

In [9]:
```
df.dropna(inplace=True)
df.shape
```

Out[9]:
```
(9542, 3)
```

In [11]:
```
df.describe(include="all")
```

Out[11]:

| | Restaurant Name | Cuisines | Average Cost for two |
|---|---|---|---|
| **count** | 9542 | 9542 | 9542.000000 |
| **unique** | 7437 | 1825 | NaN |
| **top** | Cafe Coffee Day | North Indian | NaN |
| **freq** | 83 | 936 | NaN |
| **mean** | NaN | NaN | 1200.326137 |
| **std** | NaN | NaN | 16128.743876 |
| **min** | NaN | NaN | 0.000000 |
| **25%** | NaN | NaN | 250.000000 |
| **50%** | NaN | NaN | 400.000000 |
| **75%** | NaN | NaN | 700.000000 |
| **max** | NaN | NaN | 800000.000000 |

In [13]:
```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
df['Restaurant Name'] = label_encoder.fit_transform(df['Restaurant Name'])
df['Cuisines'] = label_encoder.fit_transform(df['Cuisines'])
df
```

Out[13]:

| | Restaurant Name | Cuisines | Average Cost for two |
|---|---|---|---|
| **0** | 3743 | 920 | 1100 |
| **1** | 3168 | 1111 | 1200 |
| **2** | 2894 | 1671 | 4000 |
| **3** | 4701 | 1126 | 1500 |
| **4** | 5516 | 1122 | 1500 |
| **...** | ... | ... | ... |
| **9546** | 4437 | 1813 | 80 |
| **9547** | 1312 | 1824 | 105 |
| **9548** | 3065 | 1110 | 170 |
| **9549** | 86 | 1657 | 120 |
| **9550** | 7232 | 331 | 55 |

9542 rows × 3 columns

In [15]:
```
x = df.drop('Cuisines',axis=1)
y = df['Cuisines']
```

In [17]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
x=scaler.fit_transform(x)
```

In [19]:
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.1,random_state=15)
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression

classifier_logreg = LogisticRegression(solver="newton-cg")
classifier_logreg.fit(x_train, y_train)
logreg_pred = classifier_logreg.predict(x_test)
```

In [33]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
accuracy = accuracy_score(y_test, logreg_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.10
```

In [34]:
```
# Precision, recall, F1-score
precision = precision_score(y_test, logreg_pred, average='micro')
recall = recall_score(y_test, logreg_pred, average='micro')
f1 = f1_score(y_test, logreg_pred, average='micro')
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")
```

```
Precision: 0.10
Recall: 0.10
F1-score: 0.10
```

In [35]:
```
from sklearn.ensemble import RandomForestClassifier
model_rfc = RandomForestClassifier(n_estimators=100, random_state=42)

model_rfc.fit(x_train, y_train)

rfc_pred = model_rfc.predict(x_test)
```

In [36]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

accuracy = accuracy_score(y_test, rfc_pred)
print(f"Accuracy: {accuracy:.2f}")

# Precision, recall, F1-score
precision = precision_score(y_test, rfc_pred, average='micro')
recall = recall_score(y_test, rfc_pred, average='micro')
f1 = f1_score(y_test, rfc_pred, average='micro')
print(f"Precision: {precision:.2f}")
```

```
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")
```

```
Accuracy: 0.23
Precision: 0.23
Recall: 0.23
F1-score: 0.23
```

In [ ]:

```
Conclusion:

On comparison, we can conclude that Random Forest performs better on our model than logistic regression.

Despite repeatedly trying my best on preprocessing and model selection, model performance could not be elevated beyond the current accuracy score.

This might be because of some underlying biases either in the model training or the dataset itself.
```