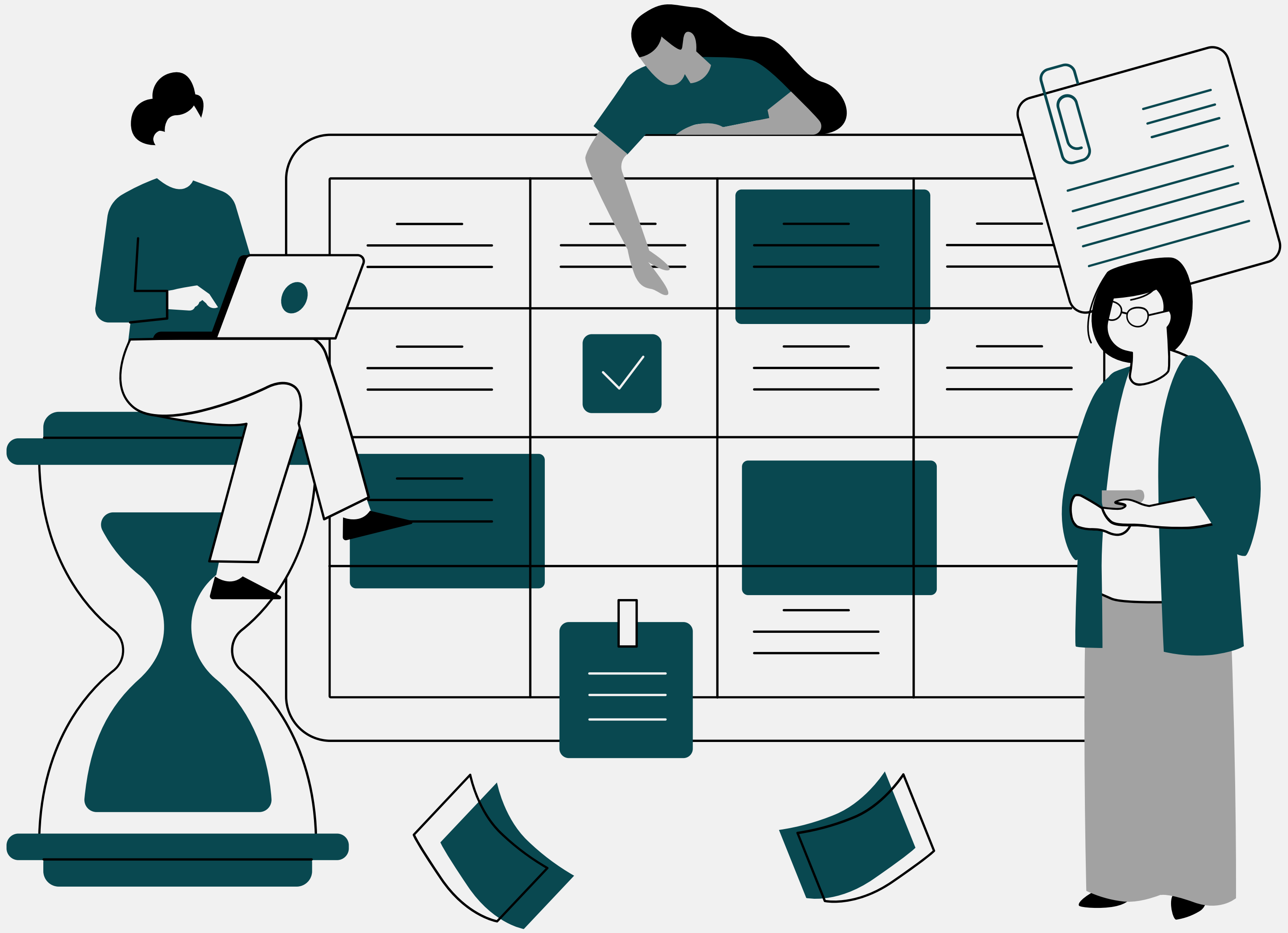
Tripadvisor European Restaurants Analysis and Rating Prediction

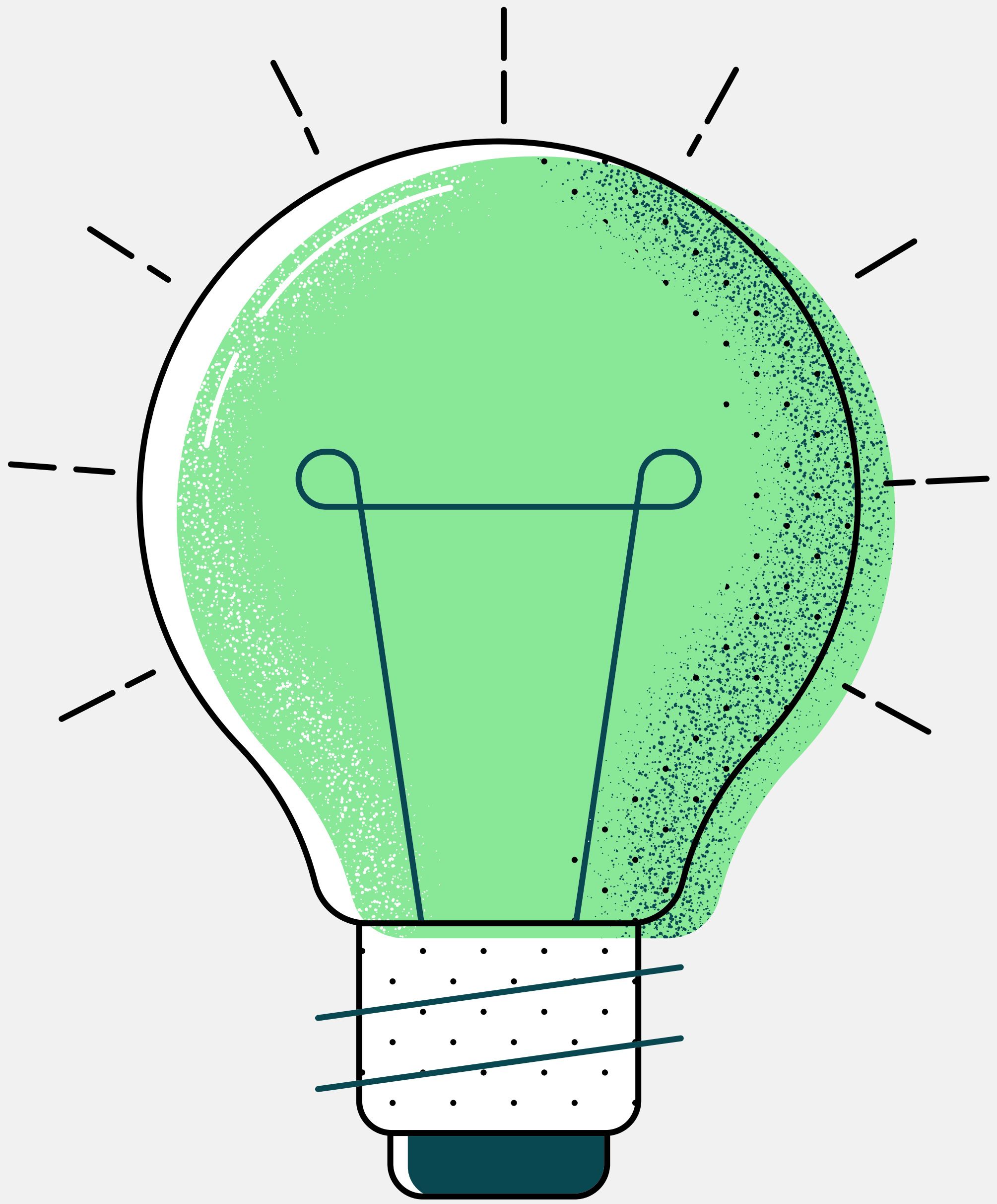
Group 18:

Sushmitha Jogula | 001546751
Gowtham Rajsimha Pulipati | 001572342

Contents

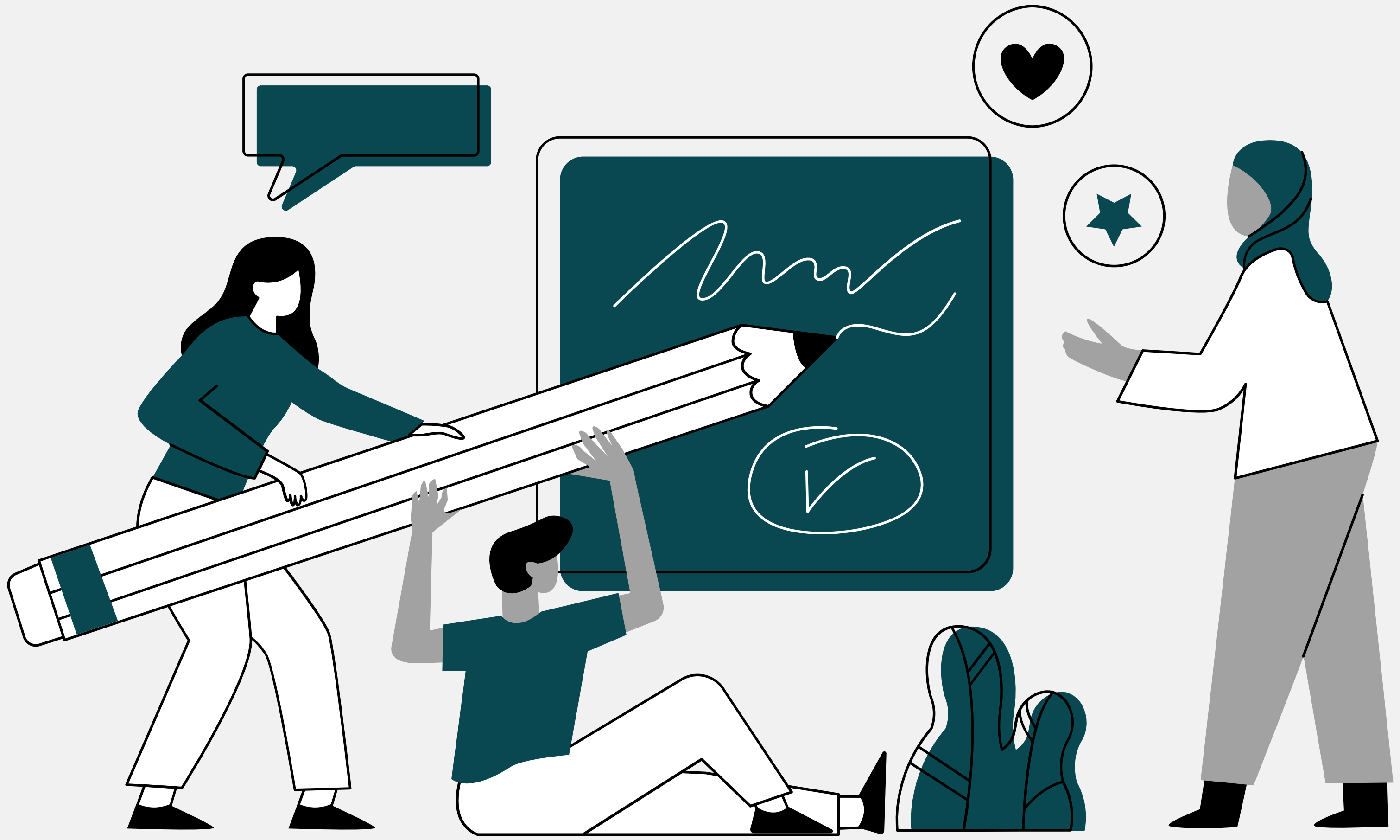


Motivation



- Being avid travelers and frequent users of the Tripadvisor website, we thought this would be the apt dataset to explore for our project.
- To identify and analyze various factors that contribute to making a restaurant successful and appreciated by the users, while performing a comparative analysis of features.
- Every company wants to know if it will be successful or not, with the restaurant rating being one of the most essential factors in deciding that.
- Also, it is imperative to consider some factors here such as an area's demographics, degree of influence of people that live in the area, restaurant's theme.

Goals



- Our goal is to analyze and predict the performance and success of a restaurant based on various factors such as cuisine types, location, price range, rating for food, service, and value, average rating, popularity index, awards etc. by applying several machine learning models.

- Using machine learning models to predict the rating value of the restaurant.

- Compare the predicted rating value with the true rating value and identify the most efficient algorithm with the highest accuracy rate for our dataset.

- Visualize and analyze the performance and accuracy of all the applied models

Dataset Information

- The Tripadvisor European Restaurants dataset has been taken from Kaggle.
- The data is retrieved from the publicly available Tripadvisor website by scraping the data in early May 2021.
- The dataset consists of information related to restaurants in European countries along with their attributes and features such as average rating, cuisine types, awards, popularity ranking, location information, number of reviews, open working hours, etc.
- The dataset has 1083397 unique restaurant links with 42 columns.
- The size of data is 679.68 MB.

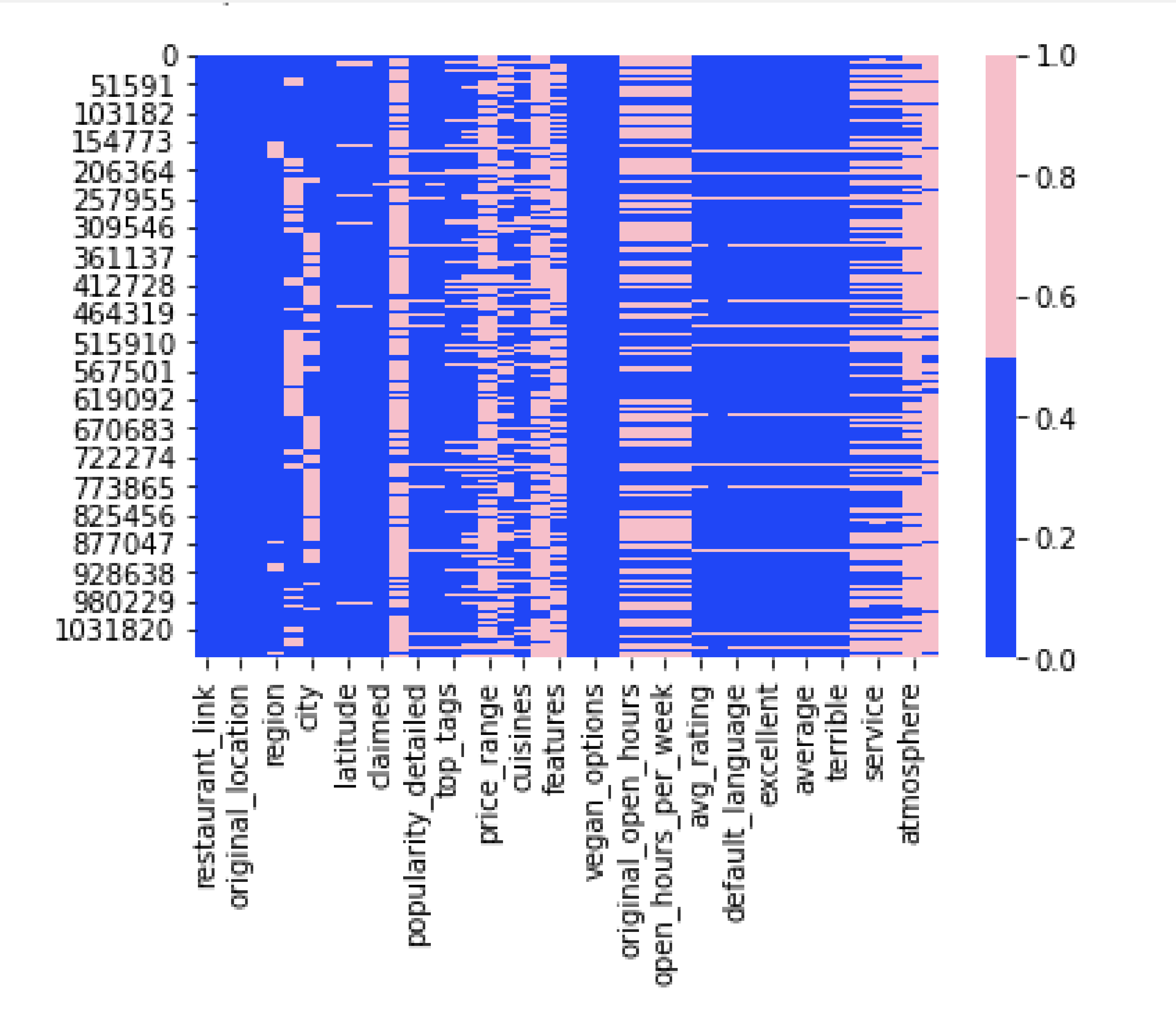
The dataset includes data from all restaurants located across several European countries as recorded on Tripadvisor.

Graphical Exploratory Data Analysis

Visualizing Missing Values

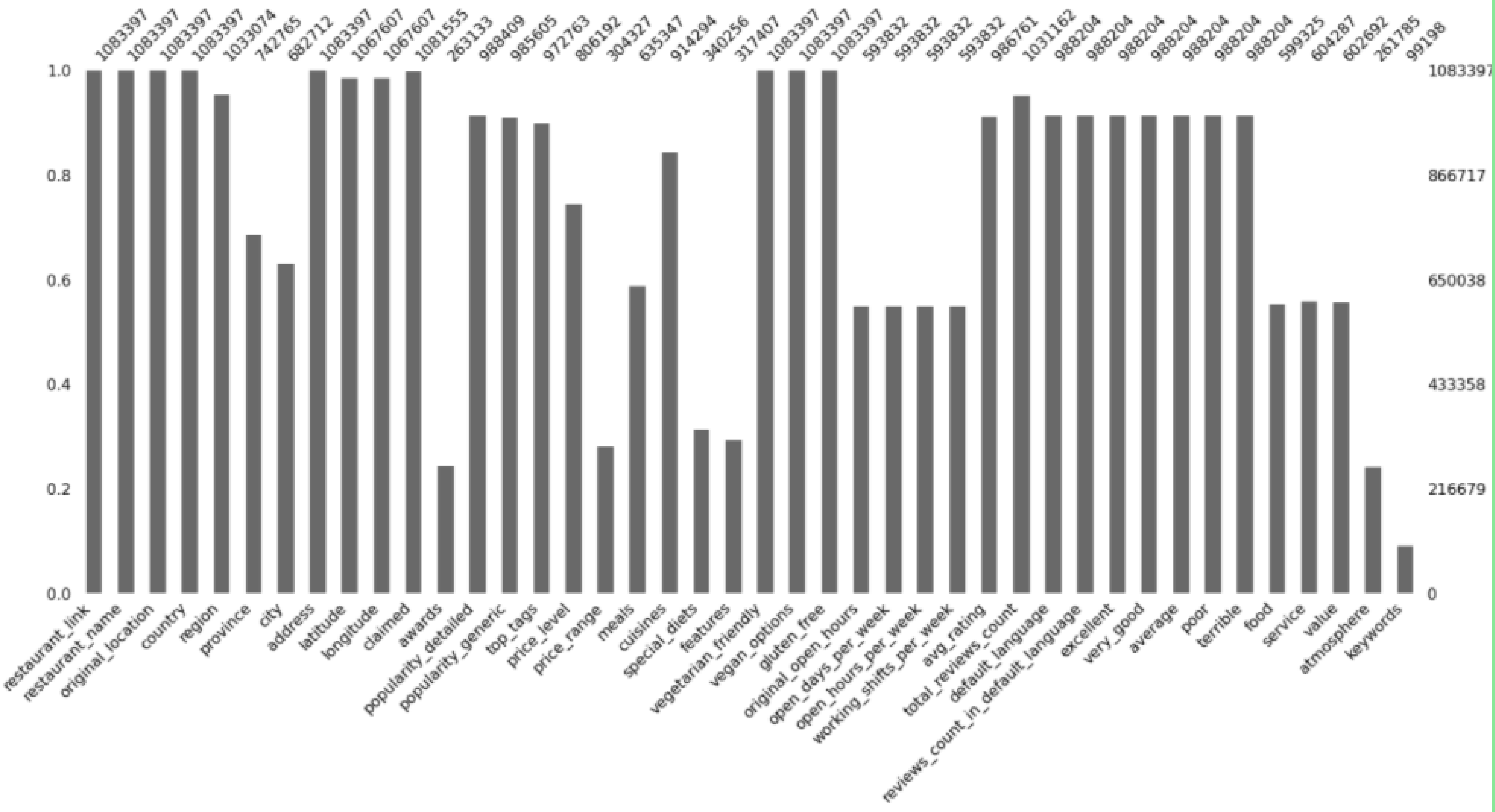
Heatmap:

Pink indicates missing values.



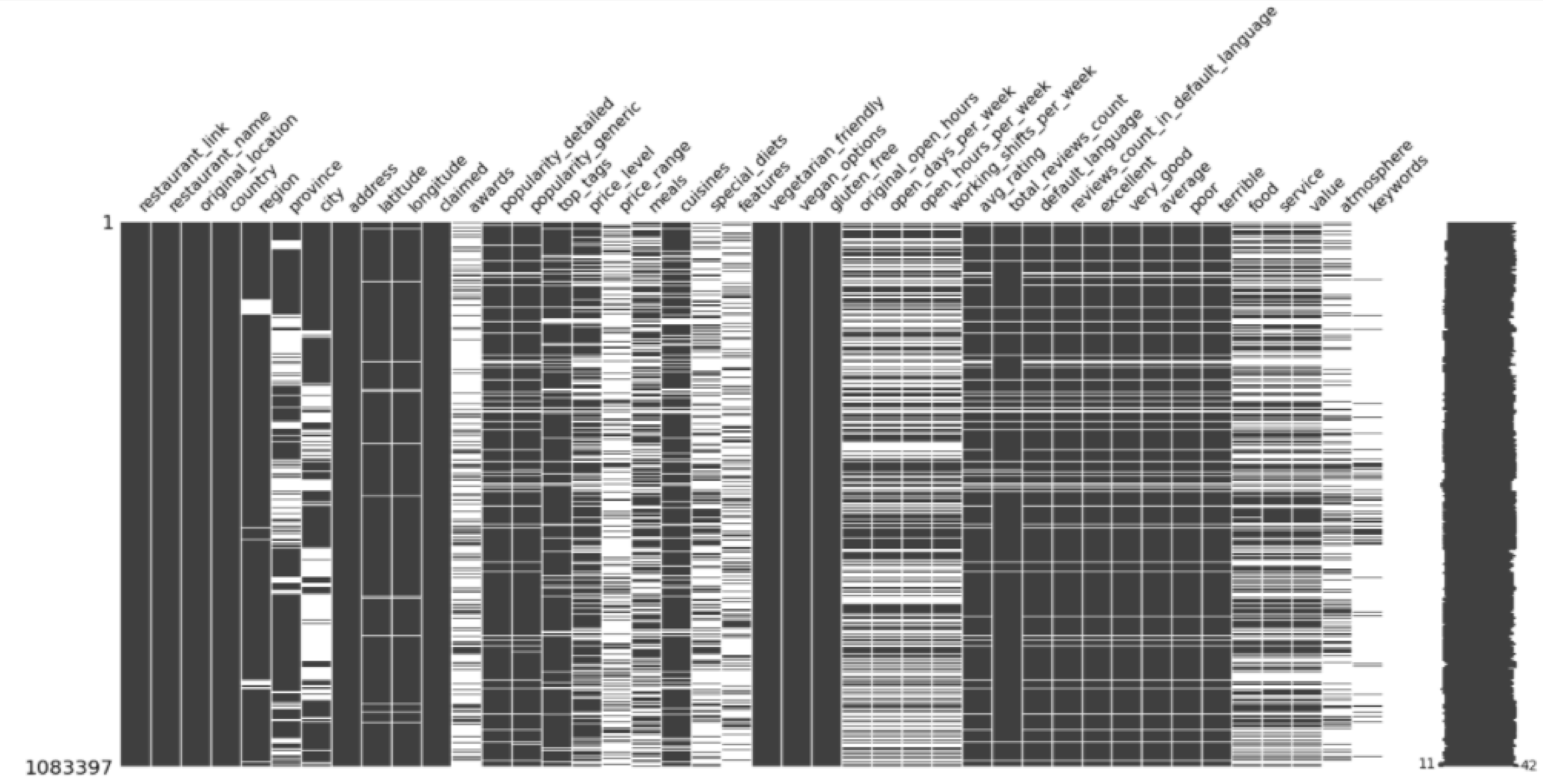
Visualizing Missing Values

Bar Plot:



Visualizing Missing Values

Matrix:



New Measures Added

The following measures are created by manipulating existing features for visualization purposes:

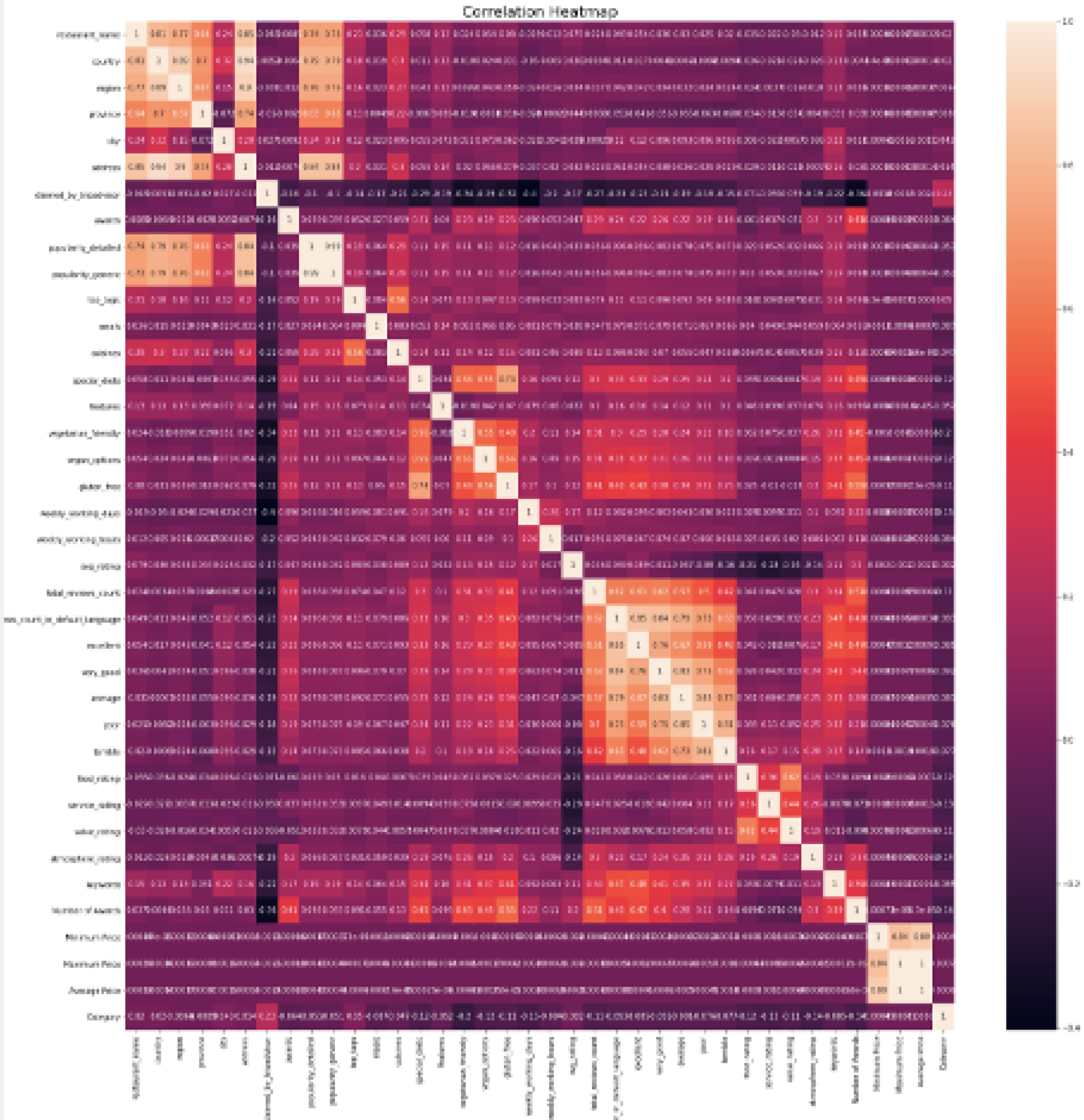
a) Number of Awards - using 'awards' column

b) Average Price - using 'price_range' column

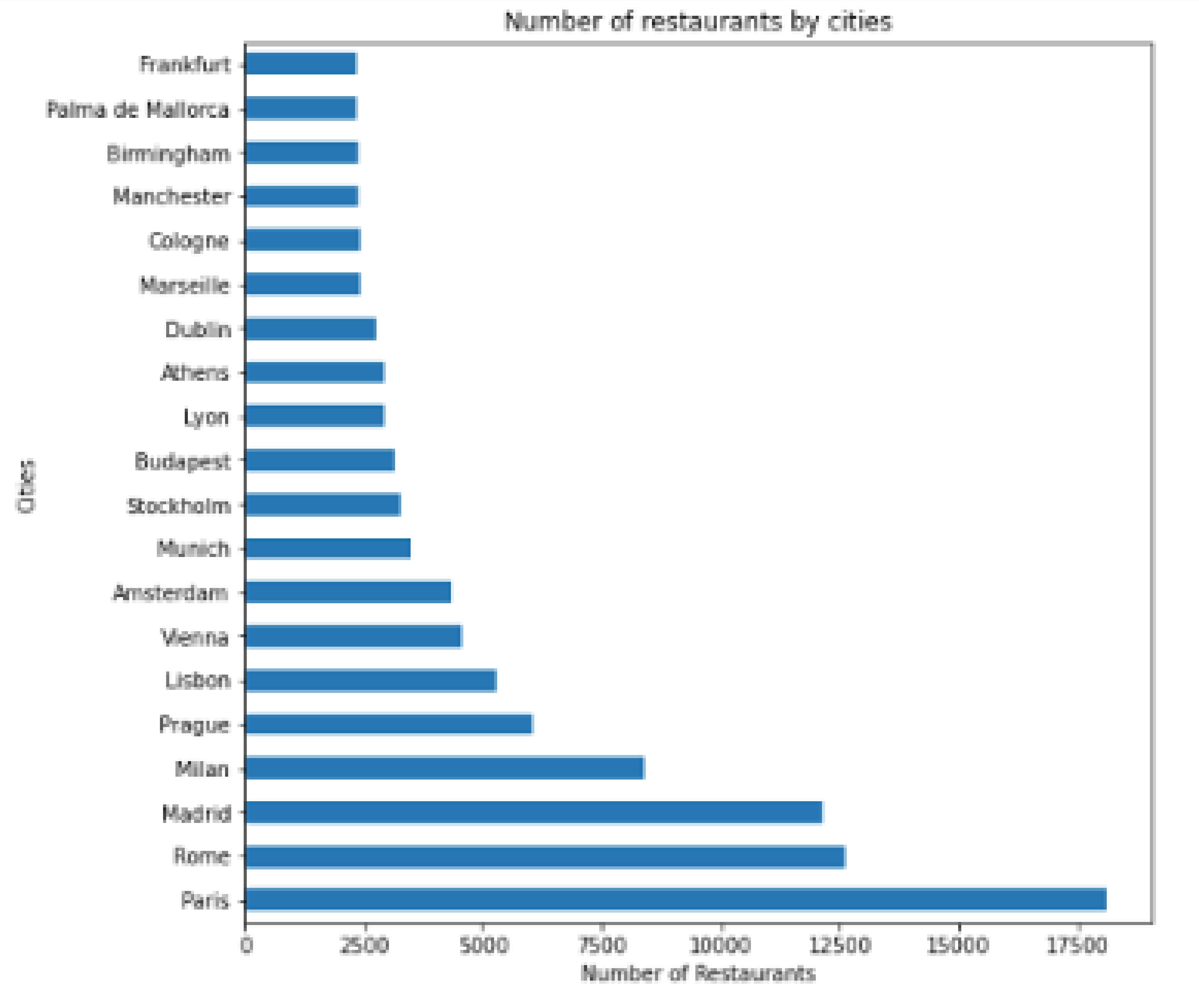
c) Restaurant Category - using 'top-_tags' column

d) Number of Features - using 'features' column

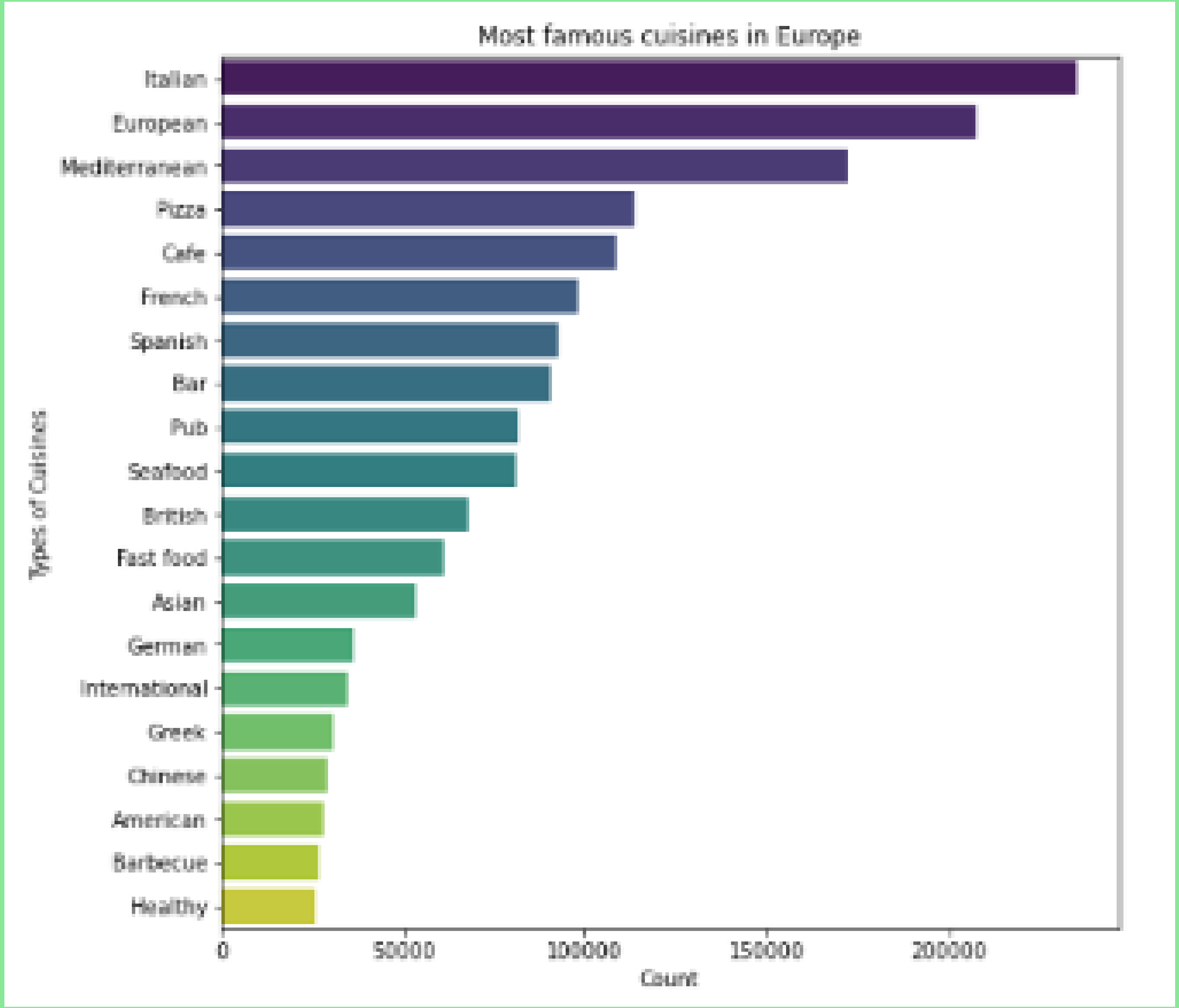
Correlation Matrix



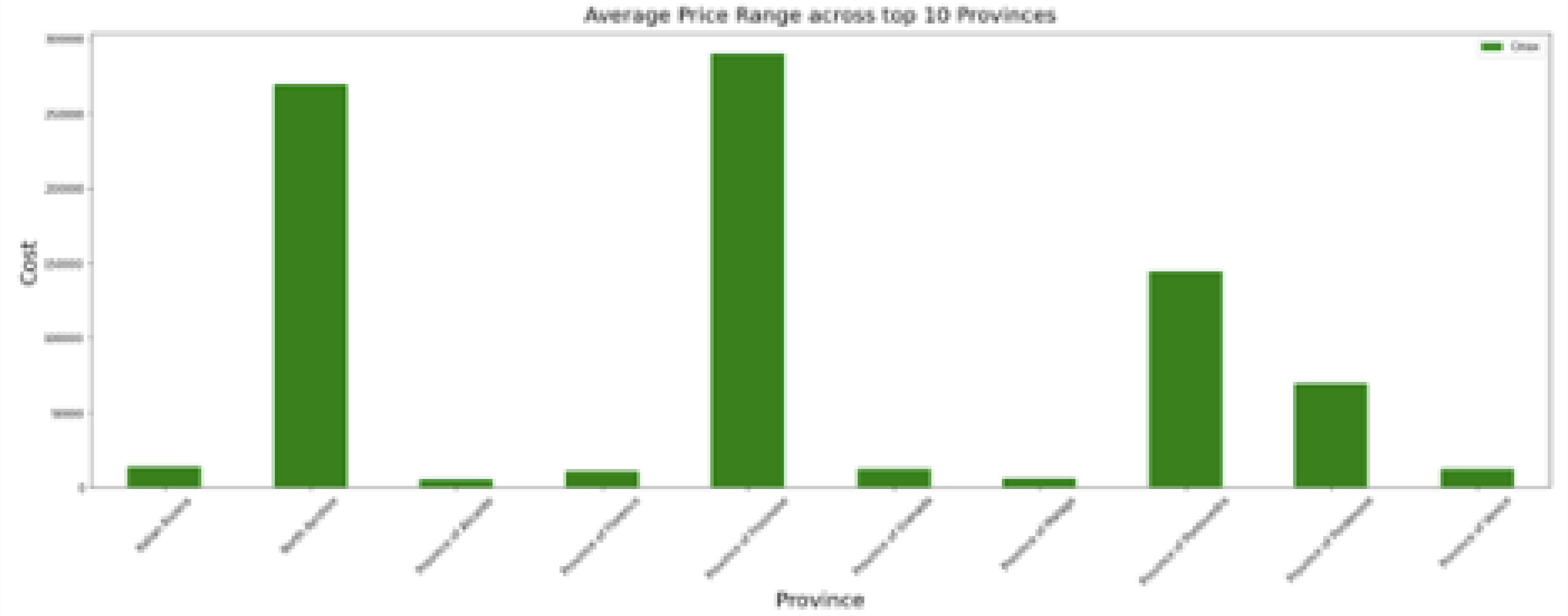
What are the most popular cities in Europe for restaurant business?



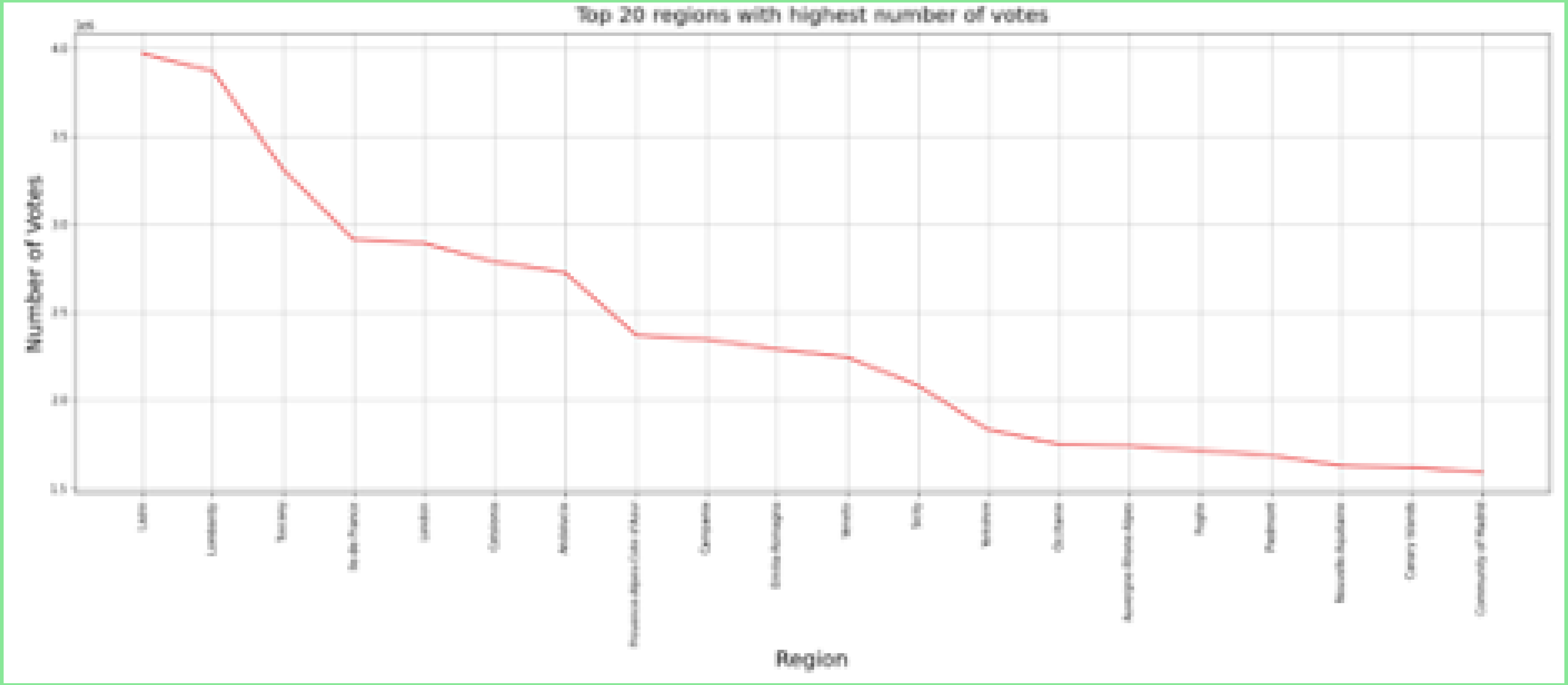
What are the most favorite cuisines liked by Europeans?



What are the top 10 European provinces with most spending in restaurants?

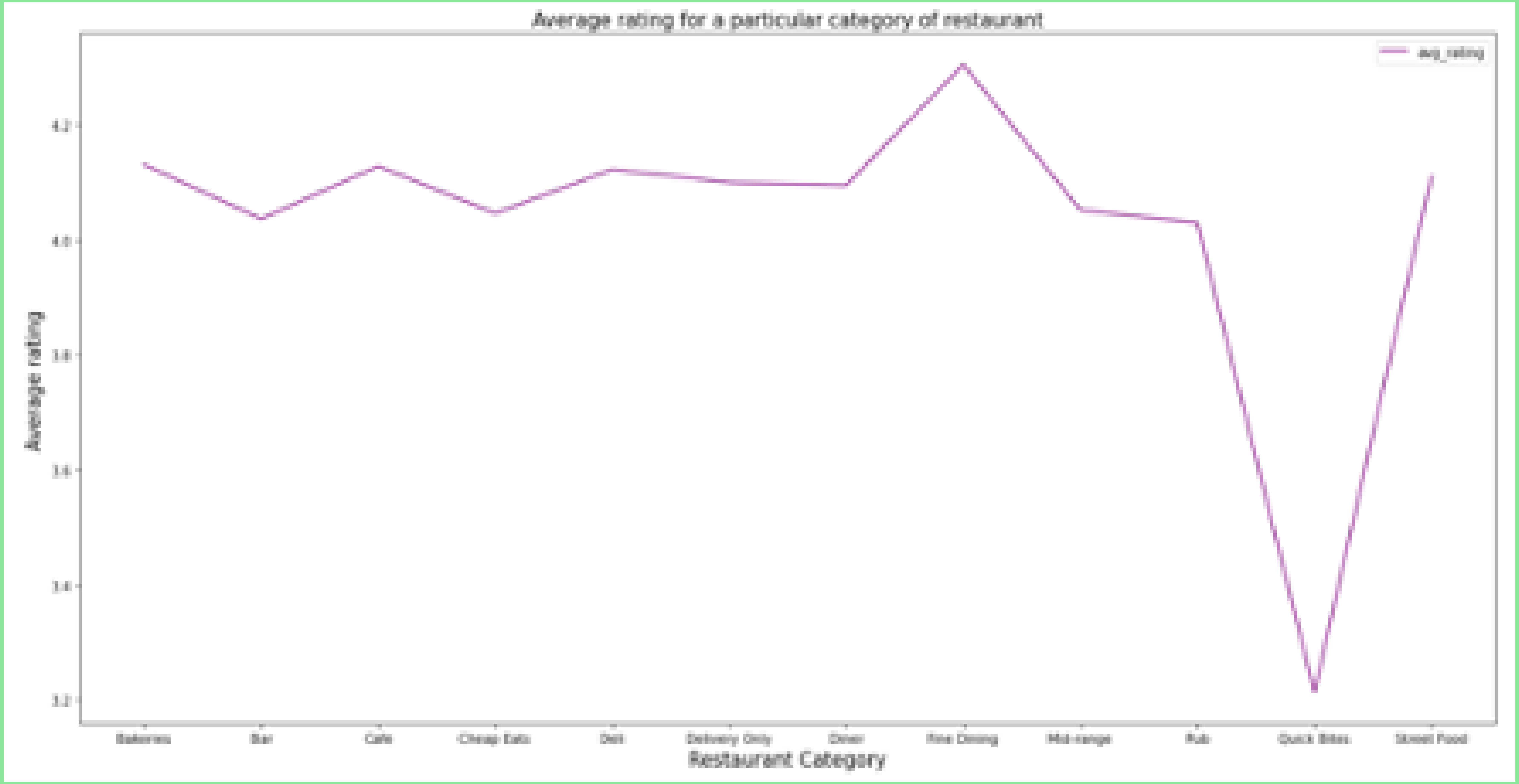


Regions with highest number of active restaurant customers

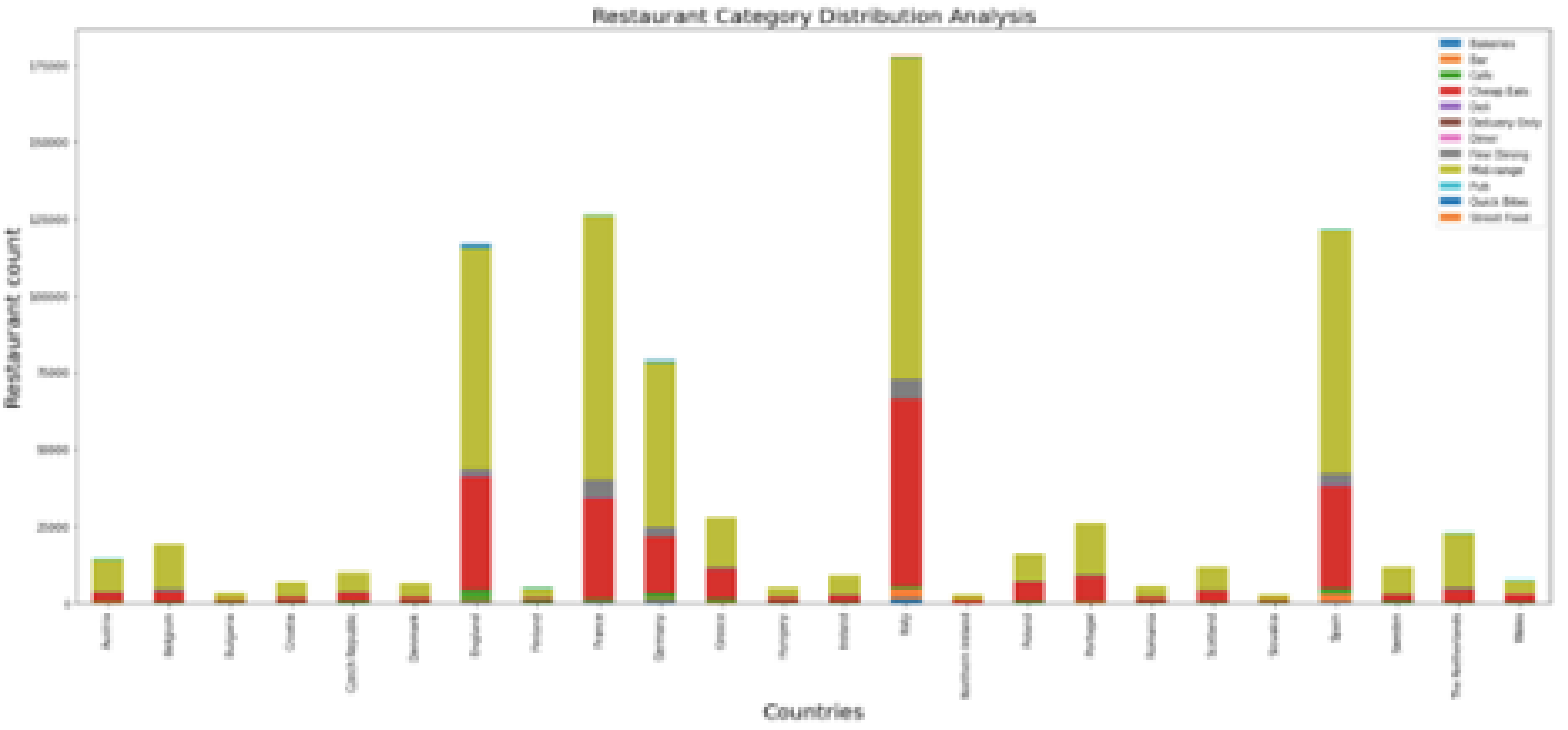


What are the top provinces with most awarded restaurants?

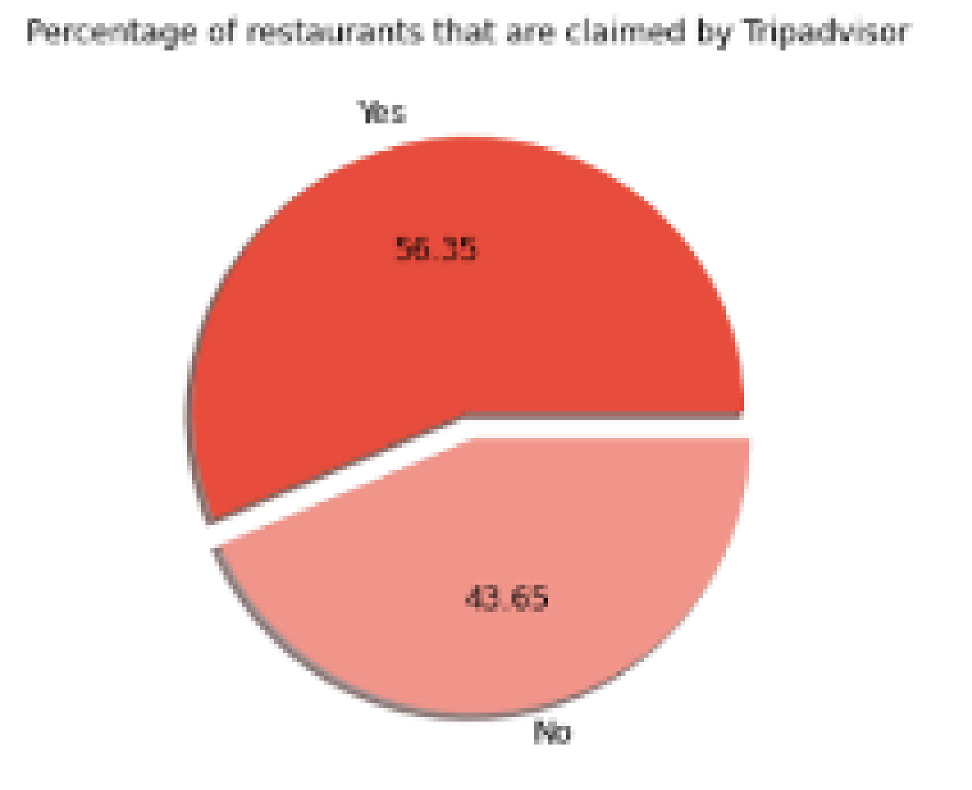
Does restaurant category affect its rating?

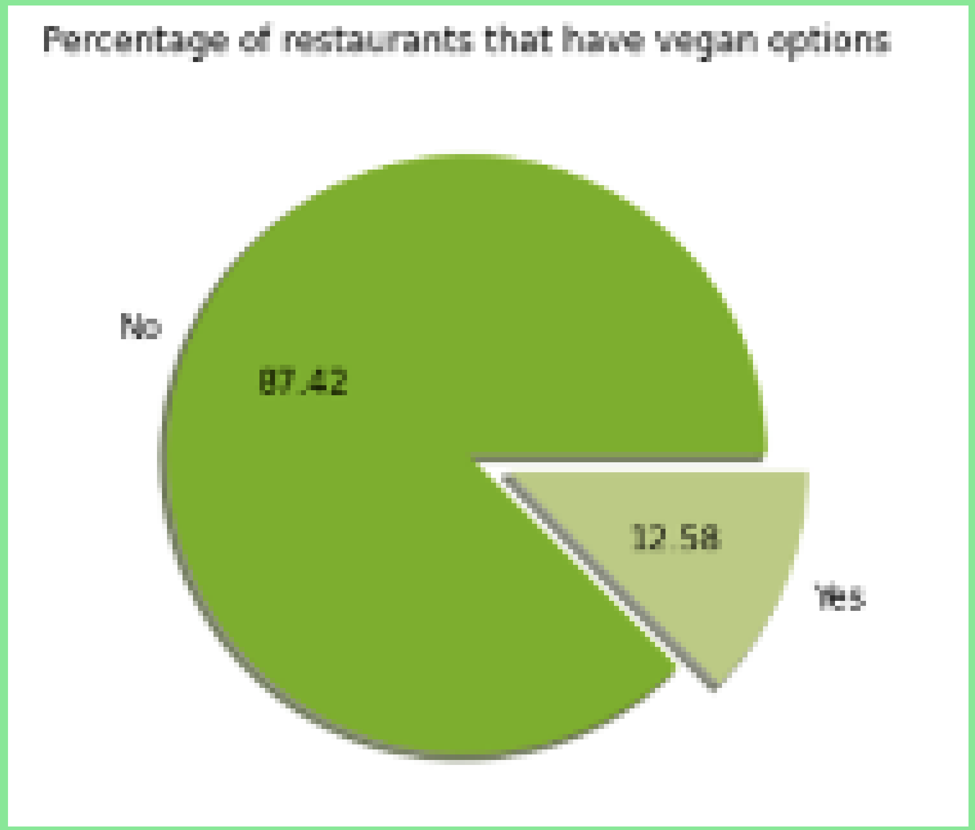


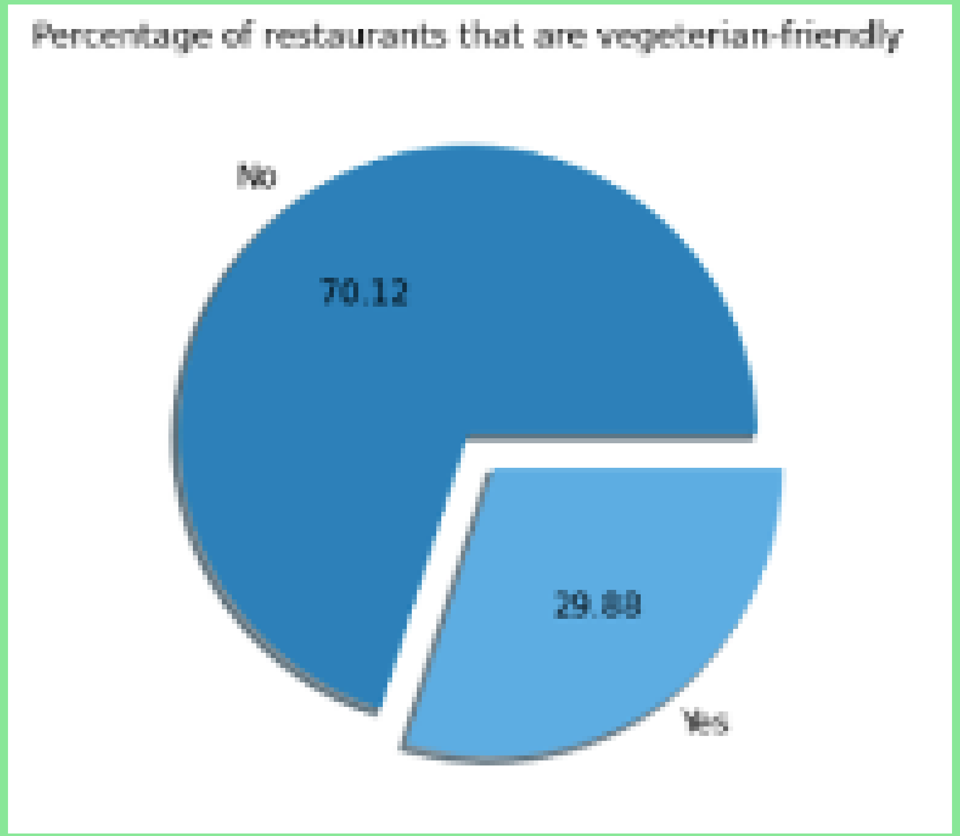
Restaurant Category Distribution

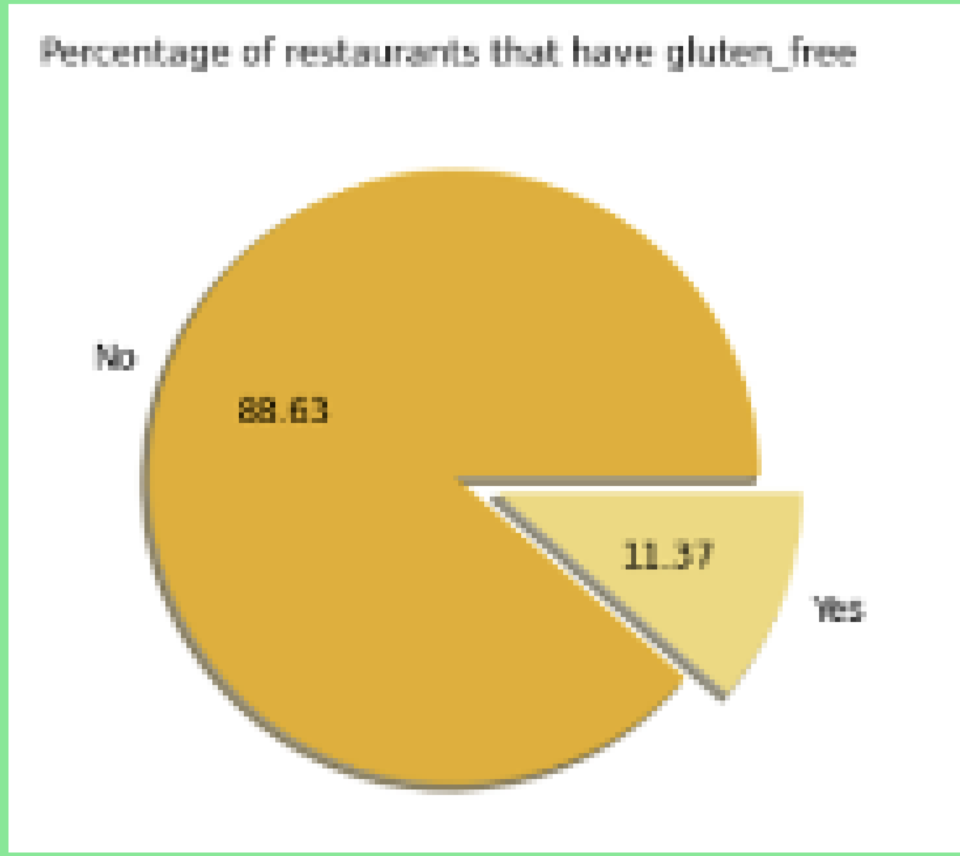


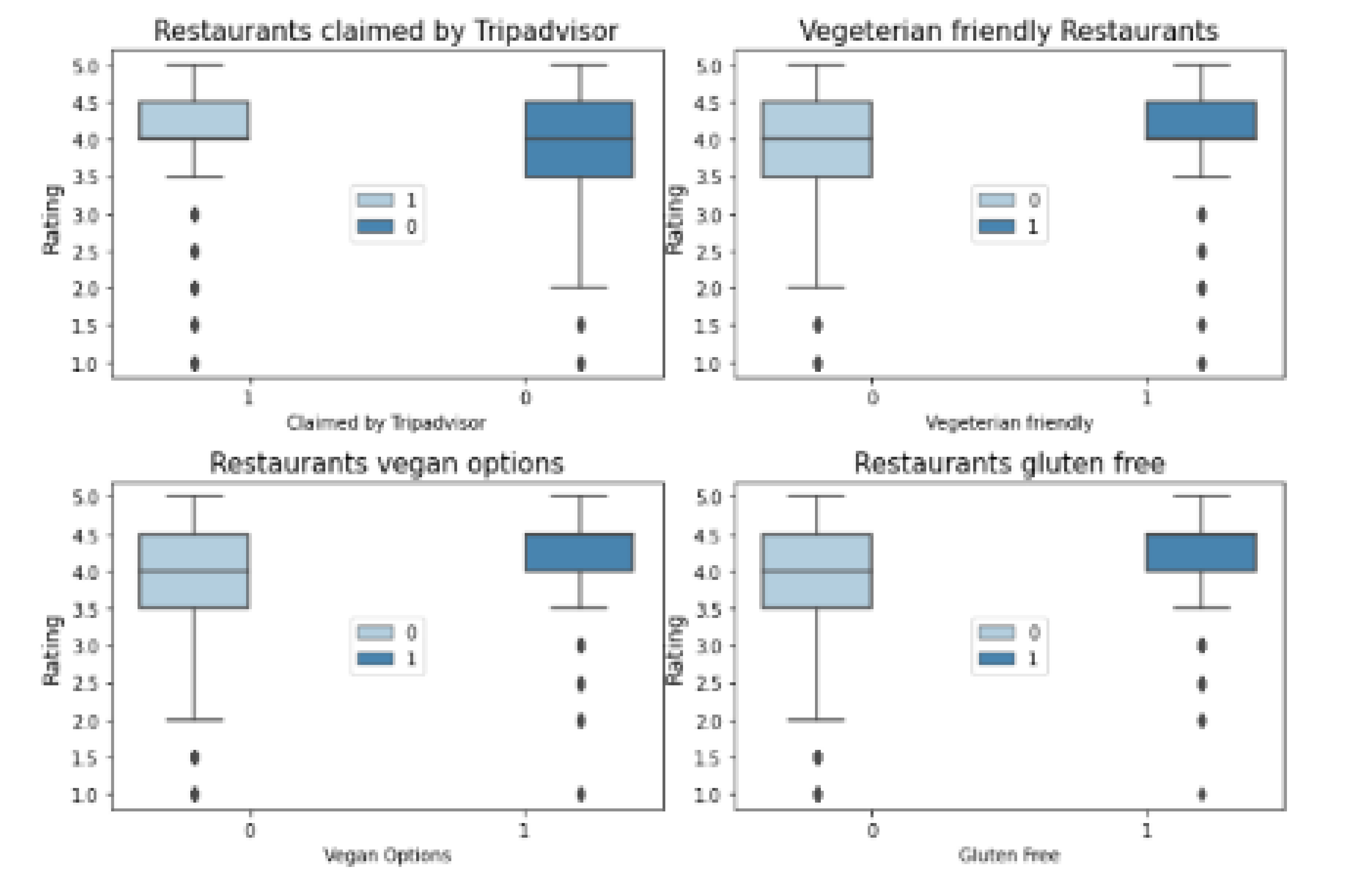
How do claimed_by_tripadvisor, vegan_options, vegeterian_friendly, gluten_free options affect ratings?





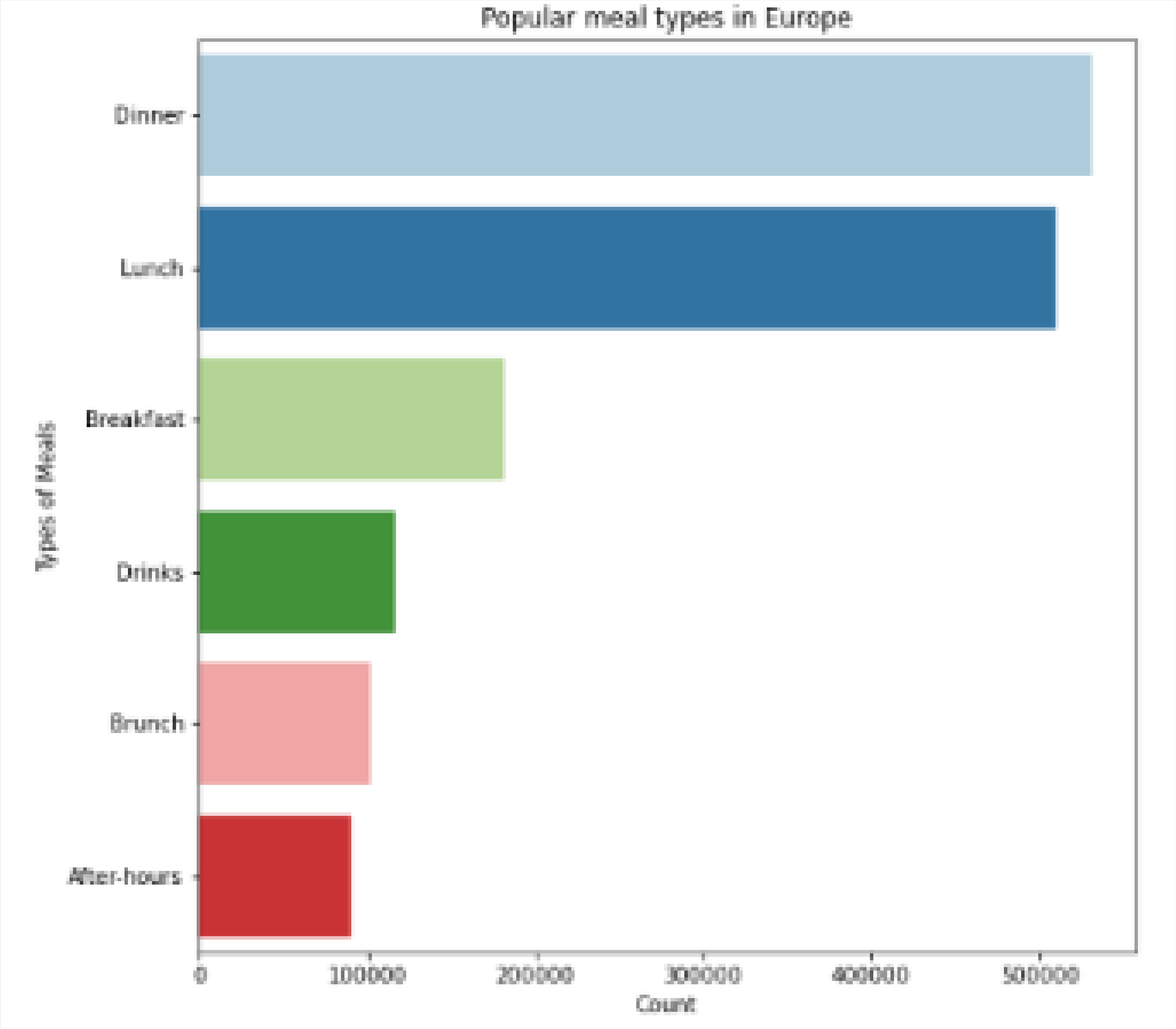




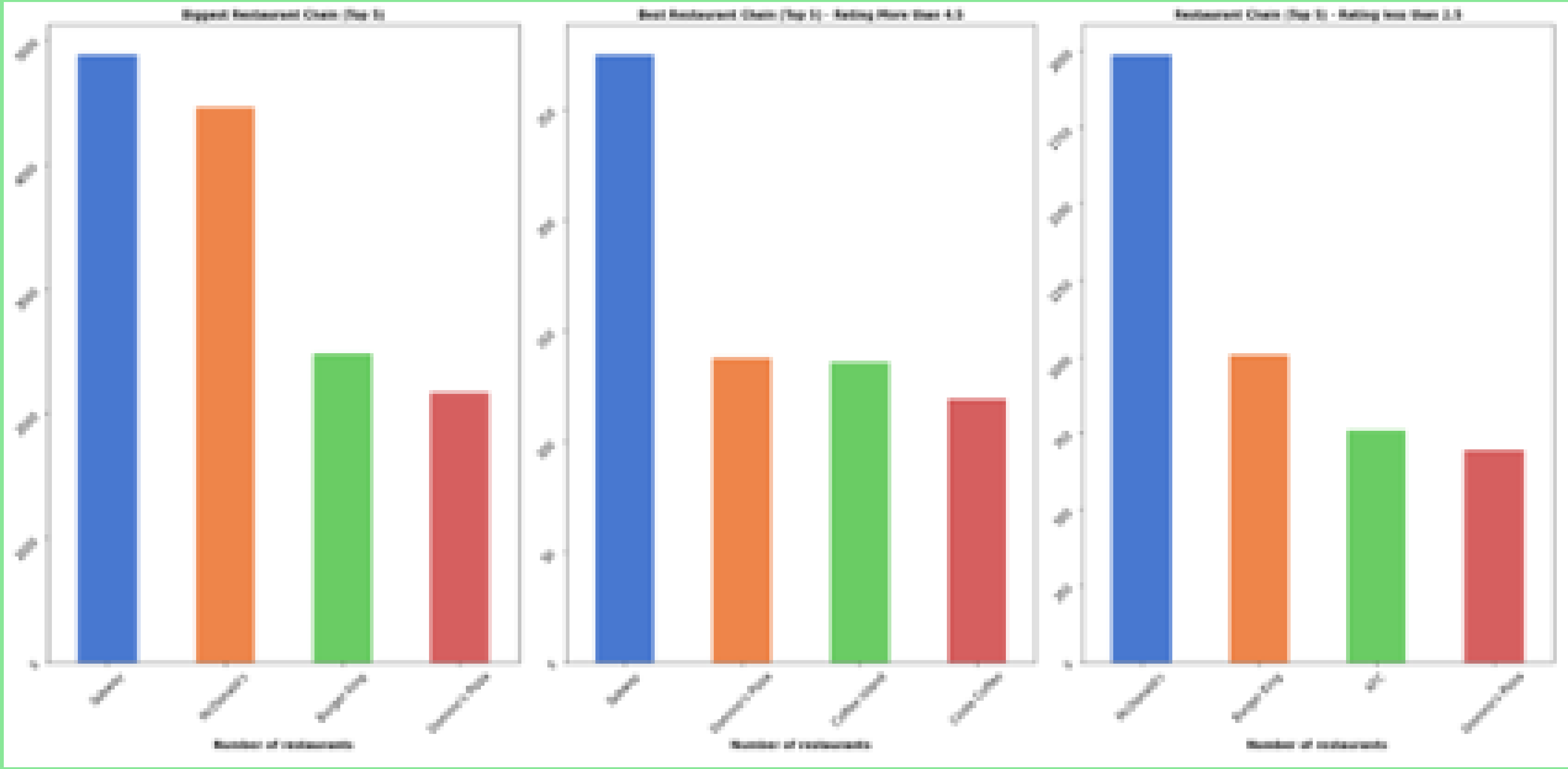


Restaurants claimed by Tripadvisor
Rating
Claimed by Tripadvisor
1
0
Vegeterian friendly Restaurants
Rating
Vegeterian friendly
0
1
Restaurants vegan options
Rating
Vegan Options
0
1
Restaurants gluten free
Rating
Gluten Free
0
1

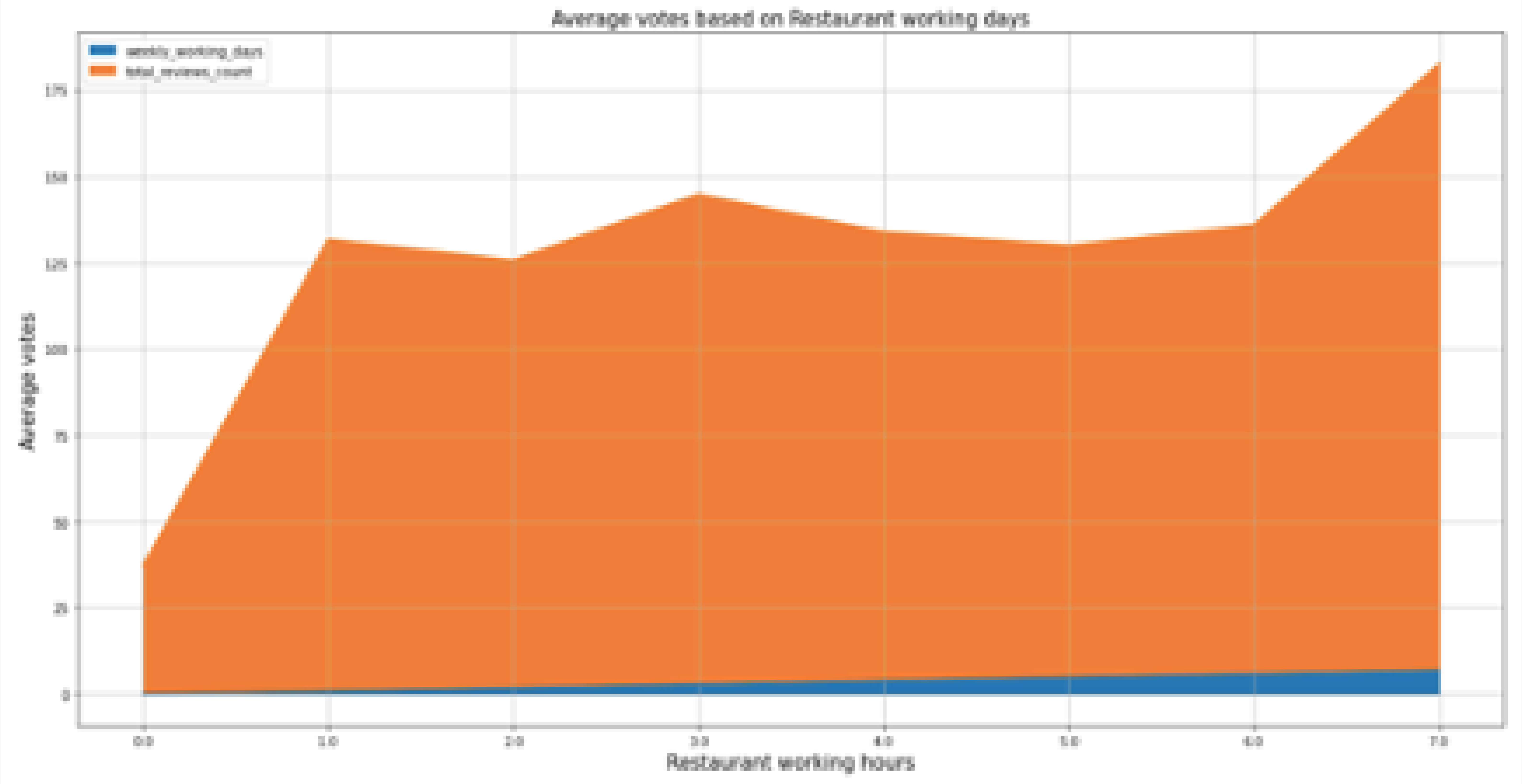
What are the popular meal types in European restaurants?



Do customers prefer quality over quantity?



How does number of working days affects number of reviews?



Machine Learning Models

Data Preprocessing

1. Dropping the columns not required.

```python
#Step 1: removing columns not required
restaurants_df.drop(['restaurant_link','default_language','original_location', 'latitude', 'longitude',
                     'price_level', 'original_open_hours', 'working_shifts_per_week'], axis = 1, inplace = True)
```

2. Renaming columns for ease of understanding.

```python
#Step 2: Rename columns for easy understanding and usage
restaurants_df = restaurants_df.rename(columns={'claimed':'claimed_by_tripadvisor','open_days_per_week':'weekly_working_days',
                                               'open_hours_per_week':'weekly_working_hours', 'food':'food_rating',
                                               'service':'service_rating', 'value':'value_rating', 'atmosphere':'atmosphere_rating'})
```

3. Cleaning data

```python
#Step 3: Cleaning data
restaurants_df['region'] = restaurants_df['region'].replace('',np.NaN)
restaurants_df['province'] = restaurants_df['province'].replace('',np.NaN)
restaurants_df['city'] = restaurants_df['city'].replace('',np.NaN)
restaurants_df['claimed_by_tripadvisor'] = restaurants_df['claimed_by_tripadvisor'].replace('',np.NaN)
#restaurants_df['awards'] = restaurants_df['awards'].replace('',np.NaN)
restaurants_df['popularity_detailed'] = restaurants_df['popularity_detailed'].replace('',np.NaN)
restaurants_df['popularity_generic'] = restaurants_df['popularity_generic'].replace('',np.NaN)
restaurants_df['top_tags'] = restaurants_df['top_tags'].replace('',np.NaN)
#restaurants_df['price_range'] = restaurants_df['price_range'].replace('',np.NaN)
#restaurants_df['meals'] = restaurants_df['meals'].replace('',np.NaN)
#restaurants_df['cuisines'] = restaurants_df['cuisines'].replace('',np.NaN)
#restaurants_df['cuisines'] = restaurants_df['cuisines'].replace('nan',np.NaN)
restaurants_df['special_diets'] = restaurants_df['special_diets'].replace('',np.NaN)
restaurants_df['features'] = restaurants_df['features'].replace('',np.NaN)
#restaurants_df['keywords'] = restaurants_df['keywords'].replace('',np.NaN)
restaurants_df['food_rating'] = restaurants_df['food_rating'].replace(np.NaN,0)
restaurants_df['service_rating'] = restaurants_df['service_rating'].replace(np.NaN,0)
restaurants_df['value_rating'] = restaurants_df['value_rating'].replace(np.NaN,0)
restaurants_df['atmosphere_rating'] = restaurants_df['atmosphere_rating'].replace(np.NaN,0)
restaurants_df['weekly_working_days'] = restaurants_df['weekly_working_days'].replace(np.NaN,0)
restaurants_df['weekly_working_hours'] = restaurants_df['weekly_working_hours'].replace(np.NaN,0)
```

Data Preprocessing

4. Changing categorical features for statistical computation

```
#Step 4: Changing categorical features for statistical computation
restaurants_df.claimed_by_tripadvisor = restaurants_df.claimed_by_tripadvisor.apply(lambda x: '1' if str(x)=='Claimed' else '0')
restaurants_df.vegetarian_friendly = restaurants_df.vegetarian_friendly.apply(lambda x: '0' if str(x)=='N' else '1')
restaurants_df.vegan_options = restaurants_df.vegan_options.apply(lambda x: '0' if str(x)=='N' else '1')
restaurants_df.gluten_free = restaurants_df.gluten_free.apply(lambda x: '0' if str(x)=='N' else '1')
```

Data Cleaning

1.Converting Nan values to fit in Count Vectorizer

```python
# converting nan values to empty strings to fit in Count Vectorizer
restaurants_df['cuisines'] =restaurants_df['cuisines'].apply(lambda x: x if x is not np.nan else '')
restaurants_df['top_tags'] =restaurants_df['top_tags'].apply(lambda x: x if x is not np.nan else '')
restaurants_df['awards'] =restaurants_df['awards'].apply(lambda x: x if x is not np.nan else '')
restaurants_df['keywords'] =restaurants_df['keywords'].apply(lambda x: x if x is not np.nan else '')
restaurants_df['meals'] =restaurants_df['meals'].apply(lambda x: x if x is not np.nan else '')
restaurants_df['Category'] =restaurants_df['Category'].apply(lambda x: x if x is not np.nan else '')
```

2. Converting text into dummies for cuisines, top tags, meals, and category

```python
[61]: # Converting text into dummies for cusines, top tags, meals and category
X = countvec.fit_transform(restaurants_df['cuisines']+' '+ restaurants_df['top_tags']+' '+restaurants_df['meals']+' '+restaurants_df['Category'])
feature_names = countvec.get_feature_names()
dummy_df = pd.DataFrame(X.todense(), columns=countvec.get_feature_names())
dummy_df
```

[61]:

	afghani	african	after	albanian	algerian	american	anhui	apulian	arabic	argentinian	...	vegetarian	venezuelan	vietnamese	welsh	wine	with	xinjiang	yunnan
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1079185	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	

Data Cleaning

3. Removing unwanted columns

```
# Removing unwanted columns
restaurants_df = restaurants_df.drop(['features','address','special_diets','price_range',
                                      'popularity_generic', 'popularity_detailed'], axis = 1)
```

4. Removing dummy columns

```
# Removing columns that were converted into dummies
restaurants_df = restaurants_df.drop(['awards','top_tags','meals', 'cuisines','keywords','Category'], axis = 1)
restaurants_df
```

5. Dropping null rows and duplicate rows

```
# Dropping nan rows
restaurants_df.dropna(how='any',inplace=True)
restaurants_df.drop_duplicates(keep='first',inplace=True)
```

Models Used

- Linear Regression
- Decision Tree
- Support Vector Regressor
- Random Forest Regressor
- Gradient Boosting Regressor
- Gradient Boosting with GridSearch

Linear Regression

```
# Training the data with Linear Regression model

lr = LinearRegression()
lr.fit(X_train.head(50000), y_train.head(50000))
```

```
LinearRegression()
```

```
y_pred_lr = lr.predict(X_test)

# R-squared value of Linear Regressor model
r2_score_lr = r2_score(y_test, y_pred_lr)

# Mean-squared value of Linear Regressor model
mse_score_lr = mean_squared_error(y_test, y_pred_lr)

print("R squared score:", r2_score_lr,
      " Mean Squared Error : ", mse_score_lr)
```

```
R squared score: 0.3112485986809397  Mean Squared Error :  2.59976117938593
```

Results

The accuracy score of the Linear Regression is 31% and not the accurate model for the dataset.

```
top_pred_lr = lr.predict(X_test.iloc[0:10, :])
print("Predictions of the first 10 examples from the test dataset \n")
for idx, v in enumerate(top_pred_lr):
    print("Rating : ",
          y_test.iloc[idx],
          ", Predicted Rating :", v)
```

```
Predictions of the first 10 examples from the test dataset

Rating :  3.5 , Predicted Rating : 3.5349621772766113
Rating :  0.0 , Predicted Rating : 1.3042984008789062
Rating :  1.7142857142857142 , Predicted Rating : 2.0751829147338867
Rating :  4.915254237288136 , Predicted Rating : 3.0514559745788574
Rating :  4.7368421052631575 , Predicted Rating : 4.230720520019531
Rating :  3.75 , Predicted Rating : 2.0357627868652344
Rating :  0.0 , Predicted Rating : 1.396428108215332
Rating :  4.5 , Predicted Rating : 3.191941261291504
Rating :  3.8260869565217392 , Predicted Rating : 2.9474897384643555
Rating :  4.235294117647059 , Predicted Rating : 2.504594326019287
```

Decision Tree

```python
# initializing Grid Search for Decision tree

dt_ds = GridSearchCV(estimator=DecisionTreeRegressor(),
                     param_grid={"criterion":
                                 ["mse", "friedman_mse", "mae"],
                                 "splitter": ["best"]})
```

training the model with 50000 rows as it almost takes 6 hours to train on full data

```python
dt_ds.fit(X_train.head(50000), y_train.head(50000))
```

```
GridSearchCV(estimator=DecisionTreeRegressor(),
             param_grid={'criterion': ['mse', 'friedman_mse', 'mae'],
                         'splitter': ['best']})
```

```python
dt_ds.cv_results_
```

```
{'mean_fit_time': array([  0.46316023,   0.47864733, 461.2406775 ]),
 'std_fit_time': array([3.19795304e-02, 3.50639232e-02, 6.93517611e+01]),
 'mean_score_time': array([0.00428562, 0.01239271, 0.032903  ]),
 'std_score_time': array([0.00111473, 0.01578472, 0.01191598]),
 'param_criterion': masked_array(data=['mse', 'friedman_mse', 'mae'],
              mask=[False, False, False],
        fill_value='?',
             dtype=object),
 'param_splitter': masked_array(data=['best', 'best', 'best'],
              mask=[False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'criterion': 'mse', 'splitter': 'best'},
  {'criterion': 'friedman_mse', 'splitter': 'best'},
  {'criterion': 'mae', 'splitter': 'best'}],
 'split0_test_score': array([0.5643567 , 0.5565077 , 0.48653268]),
 'split1_test_score': array([0.57220305, 0.56426   , 0.46920663]),
 'split2_test_score': array([0.5664681 , 0.58080717, 0.47413967]),
 'split3_test_score': array([0.58270717, 0.59307618, 0.48699693]),
 'split4_test_score': array([0.55130388, 0.55856019, 0.44761837]),
 'mean_test_score': array([0.56742378, 0.57064225, 0.47291486]),
 'std_test_score': array([0.0102374 , 0.01409327, 0.01441059]),
```

```python
dt_ds.best_params_
```

```
{'criterion': 'friedman_mse', 'splitter': 'best'}
```

```python
y_pred_dt = dt_ds.best_estimator_.predict(X_test.iloc[:, :])

# R-squared value of Decision Tree model
r2_score_dt = r2_score(y_test, y_pred_dt)

# R-squared value of Decision Tree model
mse_score_dt = mean_squared_error(y_test, y_pred_dt)

print("R square score:",
      r2_score_dt, " Mean Squared Error: ",
      mse_score_dt)
```

```
R square score: 0.6035227674664361  Mean Squared Error:  0.2135210150674068
```

Results

The accuracy score of the Decision Tree is 60% and this is not the accurate model for the dataset.

looking at the r square value and MSE, we can conclude that this model fits fairly well, and the value is almost 0.81 when trained with full dataset

```
top_pred_dt = dt_ds.predict(X_test.iloc[:10, :])
print("Predictions of the first 10 examples from the test dataset \n")
for idx, v in enumerate(top_pred_dt):
    print("Rating : ", y_test.iloc[idx],
          ", Predicted Rating :", v)
```

```
Predictions of the first 10 examples from the test dataset

Rating :  3.5 , Predicted Rating : 4.5
Rating :  5.0 , Predicted Rating : 5.0
Rating :  1.5 , Predicted Rating : 1.5
Rating :  5.0 , Predicted Rating : 4.5
Rating :  5.0 , Predicted Rating : 4.5
Rating :  5.0 , Predicted Rating : 5.0
Rating :  4.5 , Predicted Rating : 4.0
Rating :  4.5 , Predicted Rating : 4.5
Rating :  4.0 , Predicted Rating : 4.0
Rating :  4.5 , Predicted Rating : 4.5
```

As we did not train on the entire data, some of the predictions have a higher error margin

Support Vector Regression

```
# initializing and training the model

svr = SVR(kernel = 'rbf')
svr.fit(X_train.head(75000), y_train.head(75000))
```

```
SVR()
```

we used 75000 rows instead of the full dataset as the results were almost same and training the entire dataset takes a lot of time

```
y_pred_svr = svr.predict(X_test.iloc[:, :])

# R-squared value of SVR
r2_score_svr = r2_score(y_test, y_pred_svr)

# Mean-squared value of SVR
mse_score_svr = mean_squared_error(y_test, y_pred_svr)

print("R square score:",
      r2_score_svr, " Mean Squared Error: ",
      mse_score_svr)
```

```
R square score: 0.3180946401083809  Mean Squared Error:  0.36723703829158993
```

Results

The accuracy score of the Support Vector Regression is 32% and this is not the accurate model for the dataset.

similar to Linear Regression, we can see that this model does not fit well, but has a better MSE compared to Linear Regression

```
top_pred_svr = svr.predict(X_test.iloc[:10, :])
print("Predictions of the first 10 examples from the test dataset \n")
for idx, v in enumerate(top_pred_svr):
    print("True Rating : ", y_test.iloc[idx],
          ", Predicted Rating :", v)
```

```
Predictions of the first 10 examples from the test dataset

True Rating :  3.5 , Predicted Rating : 3.9526483123299 49
True Rating :  5.0 , Predicted Rating : 4.1004416816223515
True Rating :  1.5 , Predicted Rating : 1.3851788966589793
True Rating :  5.0 , Predicted Rating : 4.500310259509006
True Rating :  5.0 , Predicted Rating : 5.136619268359779
True Rating :  5.0 , Predicted Rating : 4.491788563325329
True Rating :  4.5 , Predicted Rating : 4.230660894329148
True Rating :  4.5 , Predicted Rating : 4.4236017495898785
True Rating :  4.0 , Predicted Rating : 4.027824544952 66
True Rating :  4.5 , Predicted Rating : 4.357434298646382
```

We can see the measure of error above for some of the predictions

Random Forest Regression

```python
X_train, X_test, y_train, y_test=train_test_split(x_select,
                                                  Y_unscaled,
                                                  test_size = 0.85,
                                                  random_state = 42)
```

```python
# initializing and training the model
rf_ds = RandomForestRegressor(random_state=42)
rf_ds.fit(X_train, y_train)
```

```
RandomForestRegressor(random_state=42)
```

We use the full dataset for this model as it gives fast and best results

```python
y_pred_rf = rf_ds.predict(X_test.iloc[:, :])

# R-squared value of Random Forest model
r2_score_rf = r2_score(y_test, y_pred_rf)

# Mean-squared value of Random Forest model
mse_score_rf = mean_squared_error(y_test, y_pred_rf)

print("R square score:",
      r2_score_rf,
      "Mean Squared Error:",
      mse_score_rf)
```

```
R square score: 0.813802756826671 Mean Squared Error: 0.1802756886767116
```

The r squared value of this model proves that this model fits perfectly

Results

The accuracy score of the Random Forest Regression is 81% and as of now, this is the best performing model for the dataset.

```
top_pred_rf = rf_ds.predict(X_test.iloc[:10, :])
print("Predictions of the first 10 examples from the test dataset \n")
for idx, v in enumerate(top_pred_rf):
    print("True Rating : ", y_test.iloc[idx],
          ",Predicted Rating :", v)
```

```
Predictions of the first 10 examples from the test dataset

True Rating :  3.5 ,Predicted Rating : 4.195
True Rating :  5.0 ,Predicted Rating : 5.0
True Rating :  1.5 ,Predicted Rating : 1.48
True Rating :  5.0 ,Predicted Rating : 4.66
True Rating :  5.0 ,Predicted Rating : 4.845
True Rating :  5.0 ,Predicted Rating : 5.0
True Rating :  4.5 ,Predicted Rating : 4.185
True Rating :  4.5 ,Predicted Rating : 4.385
True Rating :  4.0 ,Predicted Rating : 4.145
True Rating :  4.5 ,Predicted Rating : 4.575
```

We can see the measure of error above for some of the predictions which is marginal

Gradient Boosting Regressor

```
# initializing and trainig using gradient boosting
gbr = GradientBoostingRegressor(loss="huber")
gbr.fit(X_train, y_train)
```

```
GradientBoostingRegressor(loss='huber')
```

```
y_pred_gbr = gbr.predict(X_test.iloc[:, :])

# R-squared value of gradient boosting
r2_score_gbr = r2_score(y_test, y_pred_gbr)

# Mean-squared value of gradient boosting
mse_score_gbr = mean_squared_error(y_test, y_pred_gbr)

print("R square score:",
      r2_score_gbr, " Mean Squared Error is", mse_score_gbr)
```

```
R square score: 0.7128986090862084  Mean Squared Error is 0.15461715171750579
```

judging by the r square value above, this model fits good with the dataset but not as good as Random Forest

Results

The accuracy score of the Gradient Boosting Regressor is 71% and this is a good performing model for the dataset.

```
[97]: top_pred_gbr = gbr.predict(X_test.iloc[:10, :])
      print("Predictions of the first 10 examples from the test dataset \n")
      for idx, v in enumerate(top_pred_gbr):
          print("True Rating : ",
                y_test.iloc[idx],
                ",Predicted Rating :", v)
```

```
Predictions of the first 10 examples from the test dataset

True Rating :  3.5 ,Predicted Rating : 4.249036326286708
True Rating :  5.0 ,Predicted Rating : 4.796715276992027
True Rating :  1.5 ,Predicted Rating : 1.779283002532051
True Rating :  5.0 ,Predicted Rating : 4.625801516108199
True Rating :  5.0 ,Predicted Rating : 4.6711137713254995
True Rating :  5.0 ,Predicted Rating : 4.789387960756944
True Rating :  4.5 ,Predicted Rating : 4.256920296707539
True Rating :  4.5 ,Predicted Rating : 4.36493101331207
True Rating :  4.0 ,Predicted Rating : 3.9148926232893158
True Rating :  4.5 ,Predicted Rating : 4.451888907418005
```

We can see the measure of error above for some of the predictions which is marginal

Gradient Boosting Regressor with GridSearch

```
# initializing and trainig using gradient boosting with GridSearch

gb_ds = GridSearchCV(estimator=GradientBoostingRegressor(),
                     param_grid={"loss": ["ls", "lad", "huber"],
                                 "n_estimators": [100, 200]})
gb_ds.fit(X_train.head(50000), y_train.head(50000))
```

```
GridSearchCV(estimator=GradientBoostingRegressor(),
             param_grid={'loss': ['ls', 'lad', 'huber'],
                         'n_estimators': [100, 200]})
```

smaller dataset is used to save time as it gives similar results

```
y_pred_gb_ds = gb_ds.predict(X_test.iloc[:, :])

# R-squared value of gradient boosting with gridsearch
r2_score_gb_ds = r2_score(y_test, y_pred_gb_ds)

# Mean-squared value of gradient boosting with gridsearch
mse_score_gb_ds = mean_squared_error(y_test, y_pred_gb_ds)

print("R square score:", r2_score_gb_ds, "Mean Squared Error: ", mse_score_gb_ds)
```

```
R square score: 0.7754377364408056 Mean Squared Error:  0.12093698837280555
```

Results

The accuracy score of the Gradient Boosting Regressor with GridSearch is 77% which is a significant improvement over traditional Gradient Boosting Regressor method and this is the second best performing model for the dataset.

judging by the r square value above, we can see an improvement from the normal Gradient Boosting model and this fits well with the given dataset

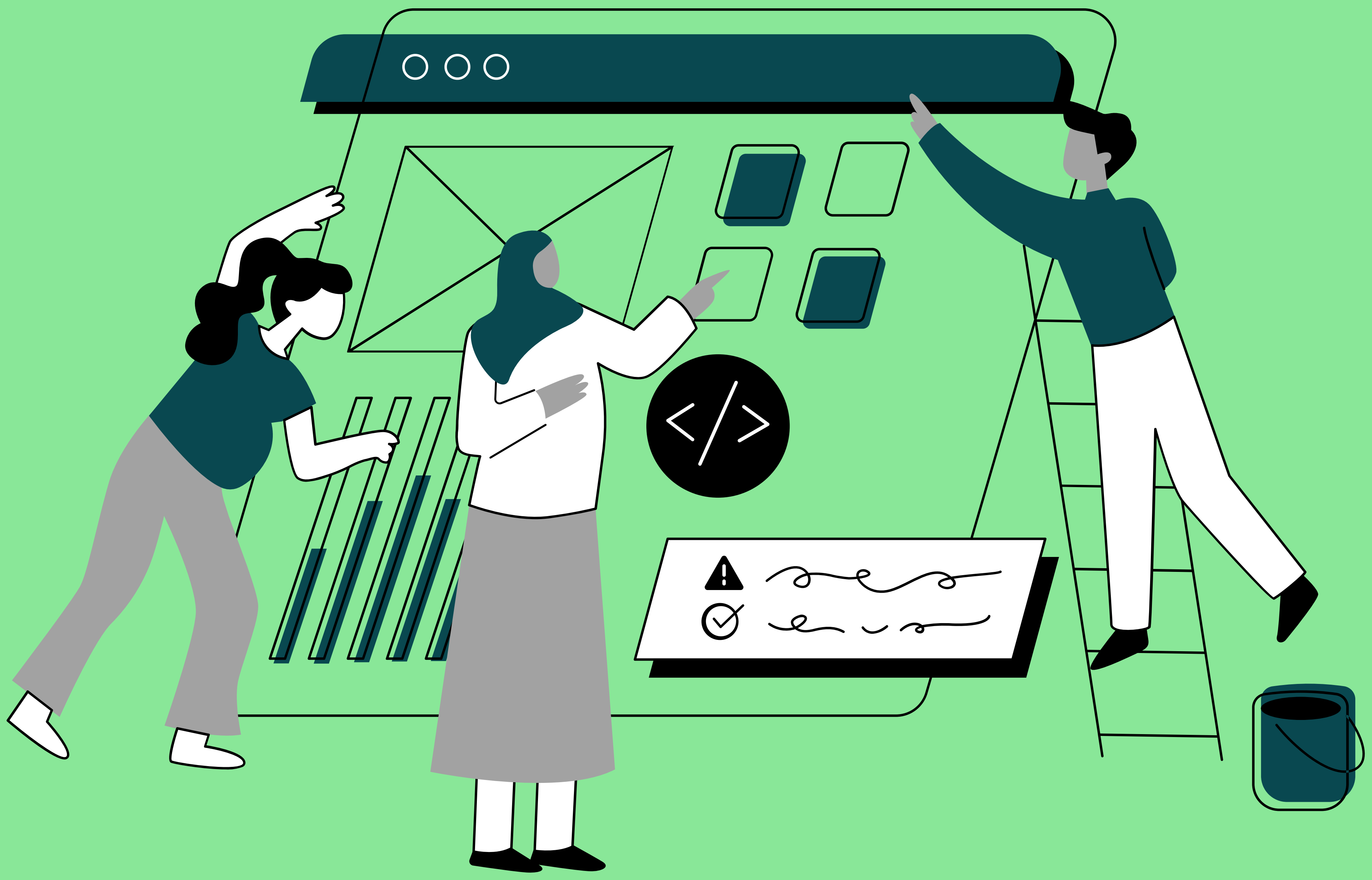
```
top_pred_gb_ds = gb_ds.predict(X_test.iloc[:10, :])
print("Predictions of the first 10 examples from the test dataset \n")
for idx, v in enumerate(top_pred_gb_ds):
    print("True Rating : ", y_test.iloc[idx],
          ", Predicted Rating :", v)
```

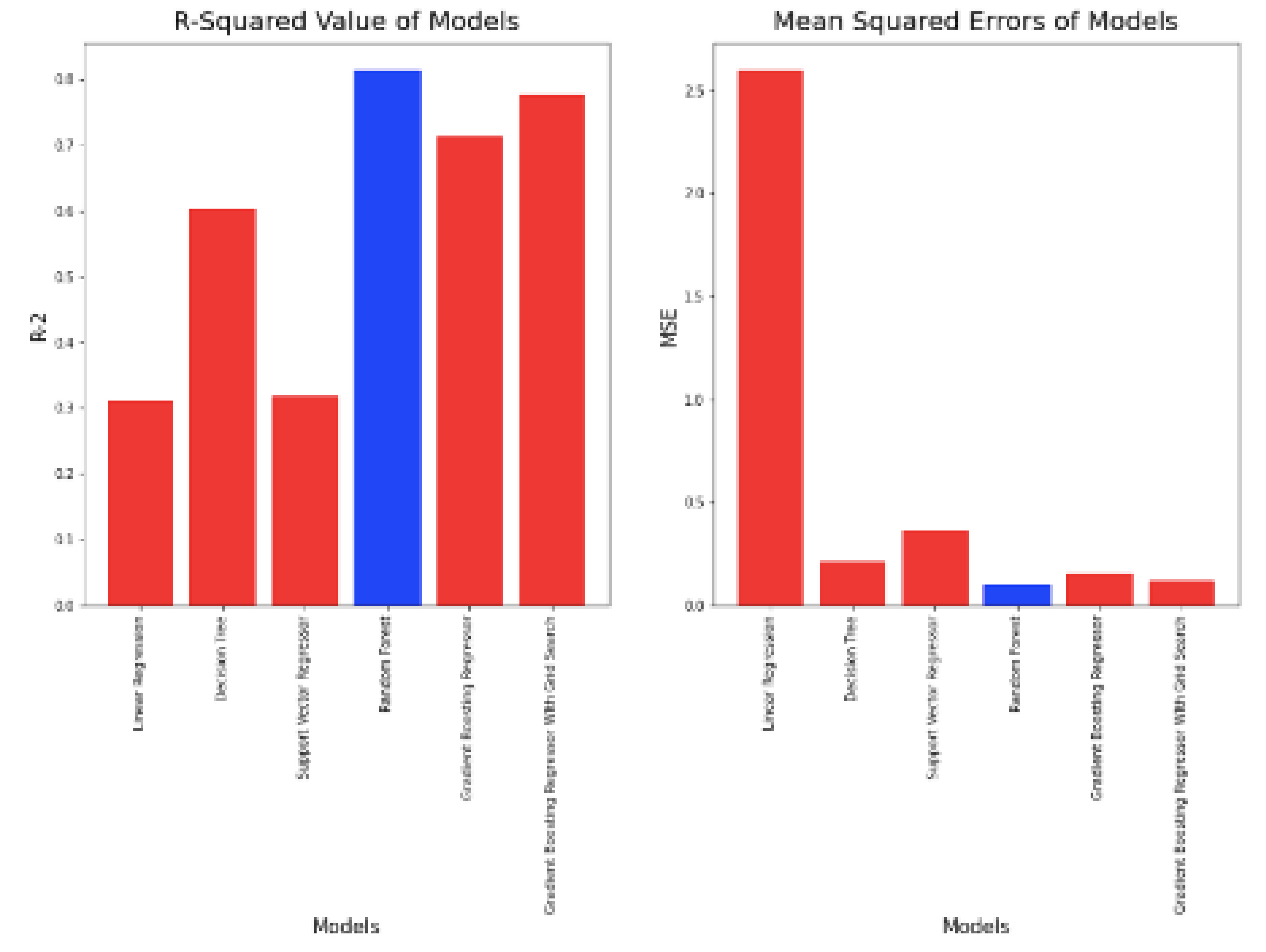
```
Predictions of the first 10 examples from the test dataset

True Rating :  3.5 , Predicted Rating : 4.191750448142788
True Rating :  5.0 , Predicted Rating : 4.798669530962853 5
True Rating :  1.5 , Predicted Rating : 1.6027770797424241
True Rating :  5.0 , Predicted Rating : 4.616324117345453
True Rating :  5.0 , Predicted Rating : 4.6825220746041225
True Rating :  5.0 , Predicted Rating : 4.824986116826986
True Rating :  4.5 , Predicted Rating : 4.274514921463149
True Rating :  4.5 , Predicted Rating : 4.390664751456 66
True Rating :  4.0 , Predicted Rating : 3.983315747465896
True Rating :  4.5 , Predicted Rating : 4.456872790775197
```

We can see the measure of error above for some of the predictions which is marginal

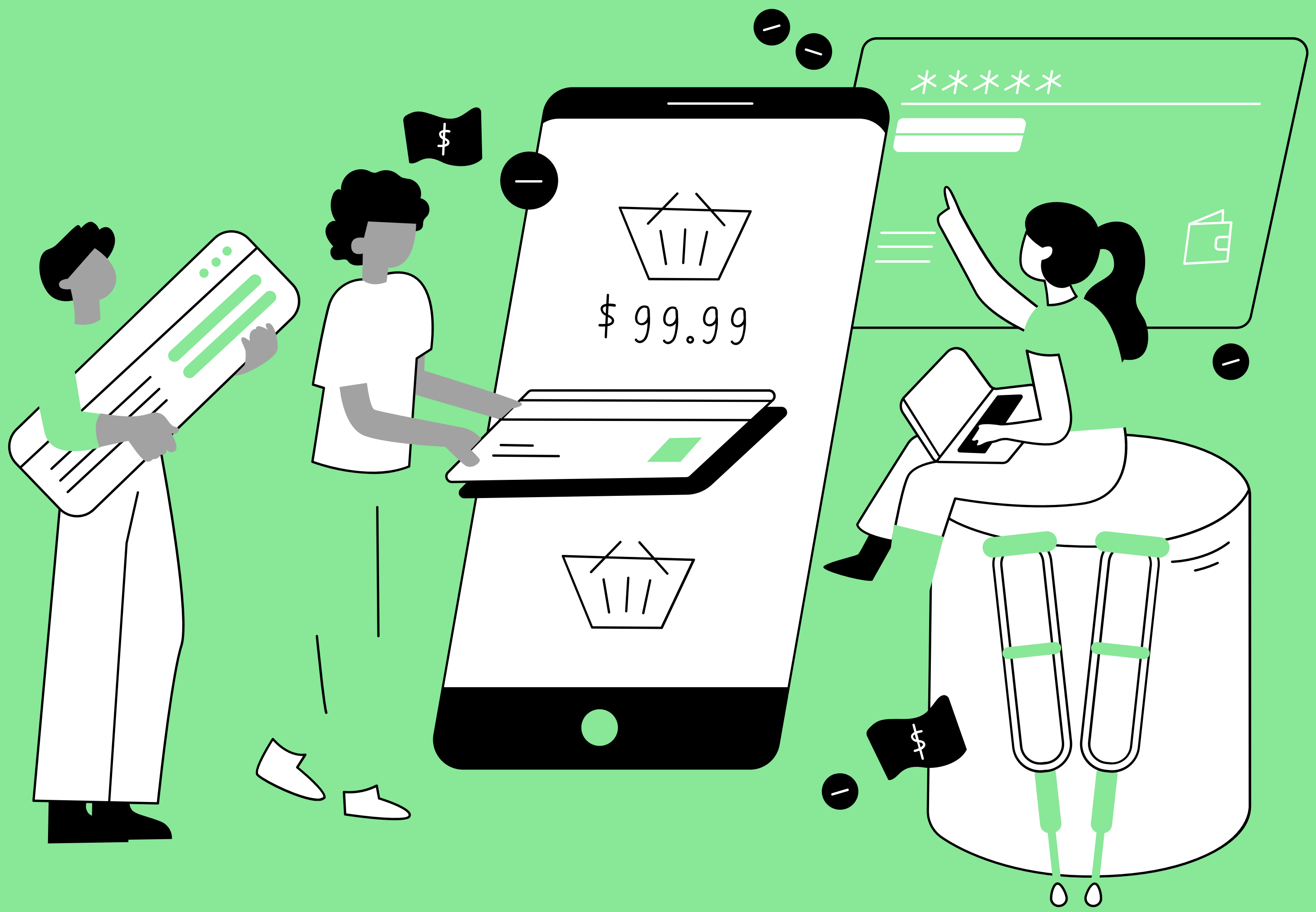
Comparative Analysis of Models




R-Squared Value of Models
R-2
Models
Linear Regression
Decision Tree
Support Vector Regressor
Random Forest
Gradient Boosting Regressor
Gradient Boosting Regressor With Grid Search
Mean Squared Errors of Models
MSE
2.5
2.0
1.5
1.0
0.5
0.0
Models
Linear Regression
Decision Tree
Support Vector Regressor
Random Forest
Gradient Boosting Regressor
Gradient Boosting Regressor With Grid Search

Conclusion

We can see that Random Forest Regression gives the best accuracy rate for the dataset with R2 score of 81%, followed by Gradient Boosting Regressor with GridSearch with an accuracy rate of 77%. The worst performing model in this case is Linear Regression with an accuracy rate of 31% which does not provide an accurate fit for the given dataset.



Thank You