Homework 1: Feature Engineering

Feature understanding and feature improvement

Applications of Machine Learning

Vanessa Gómez Verdejo vanessag@ing.uc3m.es

The content of this lab session is inspired by a previous notebook of Emilio Parrado (eparrado@ing.uc3m.es) on the Yelp database.

In [1]:
```
from IPython.core.display import Image, display
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import numpy as np # to get access to math functionalities
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
```

Objectives of this homework

In this homework we are going to work with the tools we have seen during the first sessions for:

- **Feature understanding**
- **Feature improvement**
- **Data cleaning**

For this purpose, we are going to work with the Yelp dataset.

Throughout this notebook, we will present different exercises with which you will have to extract or transform data from dataframes, representing some of these variables with `pyplot` or `seaborn` to analyze how they are distributed and how they are related to the output variable. Besides, we will have to clean the data, to impute missing values and, of course, train learning models that allow us to evaluate the advantages of this whole process.

It should be noted that the problem we are proposing to solve is really difficult, at least, with the information we are going to work with in this first notebook; so, we

do not expect to obtain accurate results, but rather to analyze how the inclusion of all the previous steps will provide us improvements in the final model performance.

Loading the dataset

Yelp_academic_dataset is a database containing reviews of businesses and establishments in 11 metropolitan areas in 4 countries collected from users of the Yelp service.

Its main motivations are:

- **sentiment analysis** with **natural language processing**, i.e., to find out whether reviews are positive or negative.
- **recommender systems**: suggesting to users new businesses (restaurants, activities, ...) to visit.

In this *notebook* we are not going to be so ambitious and we are going to focus on a task of predicting the *star rating* that a certain business will receive based on some numerical variables.

Let's start loading the database through a `json` file. Next cell does it for you!

In [2]:
```
import json # read the json file with data
from zipfile import ZipFile
from io import BytesIO
import urllib.request

url = urllib.request.urlopen("http://www.tsc.uc3m.es/~vanessa/data_not
contained_file = 'yelp_academic_dataset_business.json'
with ZipFile(BytesIO(url.read())) as my_zip_file:
    data_yelp_file = my_zip_file.open(contained_file)
    data_yelp_df = pd.DataFrame([json.loads(x) for x in data_yelp_file
```

In [3]:
```
data_yelp_df.head()
```

Out[3]:

	business_id	name	address	city	state	postal_cod
0	f9NumwFMBDn751xgFiRbNA	The Range At Lake Norman	10913 Bailey Rd	Cornelius	NC	2803
1	Yzvjg0SayhoZgCljUJRF9Q	Carlos Santo, NMD	8880 E Via Linda, Ste 107	Scottsdale	AZ	8525
2	XNoUzKckATkOD1hP6vghZg	Felinus	3554 Rue Notre-Dame O	Montreal	QC	H4C 1P
3	6OAZjbxqM5ol29BuHsil3w	Nevada House of Hose	1015 Sharp Cir	North Las Vegas	NV	8903
4	51M2Kk903DFYI6gnB5I6SQ	USE MY GUY SERVICES LLC	4827 E Downing Cir	Mesa	AZ	8520

Exercise: Filtering content

As you can see, this database contains quite a lot of information about each business, some more structured (or more accessible than others). To start working with it, let's start by simplifying the problem and limit ourselves to using the entries (rows) that correspond to category `Restaurants` and are open (`is_open` ==1).

In a real application, if we had to work with all the categories, we may had to decide whether to use a joint model for all the categories or a different model for each category, what do you think that it is going to work better?

So, start analyzing the content of the variable `categories` and, later, select those businesses labeled `Restaurants` and with the field `is_open` ==1.

SOLUTION

In [4]:
```
def get_category(df, category='Restaurants'):
    categories_ = df["categories"].dropna()
    is_member = categories_[categories_.apply(lambda x:category in x.s
    return df.loc[is_member.index].copy()

data_yelp_df = get_category(data_yelp_df)
```

In [5]:
```
data_yelp_df = data_yelp_df.loc[data_yelp_df['is_open'] == 1]
```

In [6]:
```
data_yelp_df.head(10)
```

Out[6]:

	business_id	name	address	city	state	postal
8	pQeaRpvuhoEqudo3uymHIQ	The Empanadas House	404 E Green St	Champaign	IL	
24	eBEfgOPG7pvFhb2wcG9I7w	Philthy Phillys	15480 Bayview Avenue, unit D0110	Aurora	ON	L
25	lu7vtrp_bE9PnxWfA8g4Pg	Banzai Sushi	300 John Street	Thornhill	ON	L3
30	9sRGfSVEfLhN_km60YruTA	Apadana Restaurant	13071 Yonge Street	Richmond Hill	ON	L
33	vjTVxnsQEZ34XjYNS-XUpA	Wetzel's Pretzels	4550 East Cactus Rd, #KSFC-4	Phoenix	AZ	
41	98hyK2QEUeI8v2y0AghfZA	Pho Lee's Vietnamese Restaurant	1541 E 38th St, Ste 101	Cleveland	OH	
43	LoRef3ChgZKbxUio-sHgQg	Amir	5252 Rue Jean Talon O	Montréal	QC	H
49	tLpkSwdtqqoXwU0JAGnApw	Wendy's	4602 Northfield Road	Cleveland	OH	
54	lK-wuiq8b1TuU7bfbQZgsg	Hingetown		Cleveland	OH	
59	LAoSegVNU4wx4GTA8reB6A	Tzikii Food Truck	7510 S Priest Dr	Tempe	AZ	

2. Predicting the average *rating*

The problem we are going to try to solve with this database is to **predict** the average *rating* of each establishment in the test set by building predictive models (supervised learning) using the data from the training set.

The first question to solve is whether, according to the *ratings* values, a classification model or a regression model is more suitable for predicting these values. What do you think?

SOLUTION

In [7]:
```
np.unique(data_yelp_df['stars'])
```

Out[7]:
```
array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])
```

Collin Martin

Due to the output value bein incrementally increasing with values 1-5, then it is reasonable to use a regression model to predict the value of the ratings returned within this problem

During next sections, we are going to use different variables, different versions of them, combinations and difference learning models. So, use the next variable (dictionary) to save the different results for later comparison purposes. In this sense, we are going to use the R^2 score as default performance measure.

In [8]:
```
global_results ={} # to store the results of the different models for
```

Besides, to assess the generalization of this model, we are going to divide the database into two partitions, training and test, at 50% each. Next cell includes the code of this partition for you, so that you use the same data partitions along this notebook.

In [9]:
```
from sklearn.model_selection import train_test_split

train_df, test_df = train_test_split(data_yelp_df, test_size=0.5, rand
```

In [10]:
```
# Create log10_review_count column and persist it in both dataframes
train_df['log10_review_count'] = np.log10(train_df['review_count'] + 1
test_df['log10_review_count'] = np.log10(test_df['review_count'] + 1)

print("Added log10_review_count column to both train and test datafram
print(f"Train df shape: {train_df.shape}")
```

```
print(f"Test df shape: {test_df.shape}")
print(f"Train df columns: {list(train_df.columns)}")
```

```
Added log10_review_count column to both train and test dataframes
Train df shape: (21982, 15)
Test df shape: (21983, 15)
Train df columns: ['business_id', 'name', 'address', 'city', 'state', '
postal_code', 'latitude', 'longitude', 'stars', 'review_count', 'is_ope
n', 'attributes', 'categories', 'hours', 'log10_review_count']
```

2.1 A first analysis of variables `review_count`

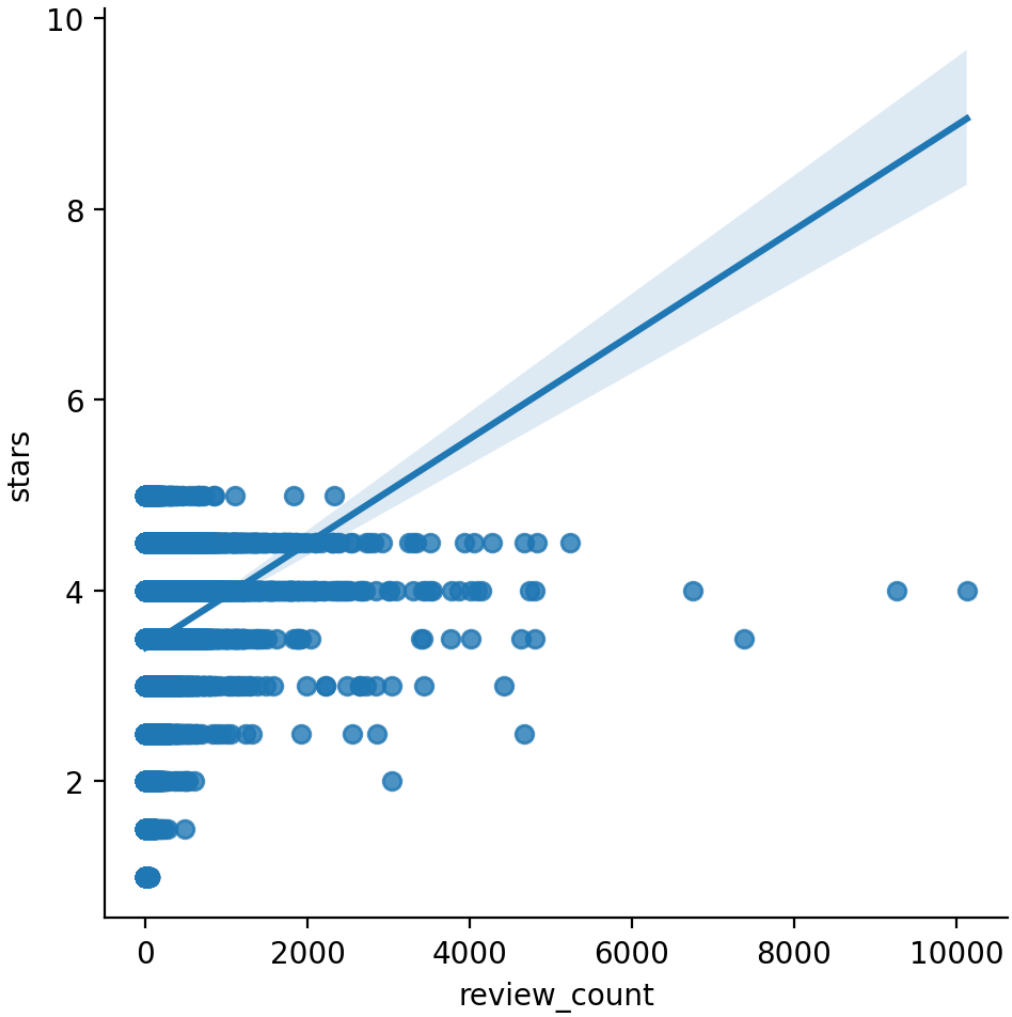
With the above information it appears that the easiest (or most immediate) variable to use is `review_count`.

Start by visualizing the variable to be predicted against this variable. Choose the visualization that you think might be most useful to later apply a linear regression model.

SOLUTION

In [11]:
```
sns.lmplot(data = data_yelp_df, x = 'review_count', y = 'stars')
```

Out[11]:
```
<seaborn.axisgrid.FacetGrid at 0x78668f468470>
```



2.2 Using the `review_count` information

A rather simplified version of the problem is to predict the *rating* from the number of reviews a facility has. The intuition behind this model is that in general better businesses will receive more customers and therefore more reviews.

So, we are going to consider the simplest regression model that can capture this pattern, that it is a linear regression. y=w0+w1x Where y is the rating, x is the number of revisions (so far we are going to consider a single input feature) and w0 and w1 are two constants to be learned from the training data (they are the parameters of the model).

Complete next code cells to train a linear regression model to predict the average rating using as single input the number of reviews. You can use the `LinearRegression` implementation of *scikit learn*.

Once the model is trained, analyze the weights of the regression model and, if you want, visualize the resulting model. What do you think that it is happening???

SOLUTION

In [12]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
linreg = LinearRegression()
x_train = train_df['review_count']
y_train = train_df['stars']
x_test = test_df['review_count']
y_test = test_df['stars']

linreg.fit(x_train.values.reshape(-1,1), y_train.values)
y_pred_linreg = linreg.predict(x_test.values.reshape(-1,1))
weights = linreg.coef_
intercept = linreg.intercept_
mse = mean_squared_error(y_test, y_pred_linreg)
r2 = r2_score(y_test, y_pred_linreg)

global_results['linear_regression'] = {'weights': weights, 'intercept'

print("Linear Regression")
print('\n', global_results['linear_regression'])
```

```
Linear Regression

 {'weights': array([0.00058963]), 'intercept': np.float64(3.39521765359
0921), 'mse': 0.6843743355525993, 'r2': 0.019667929028433506}
```

2.3 Improving the results with adequate feature transformations

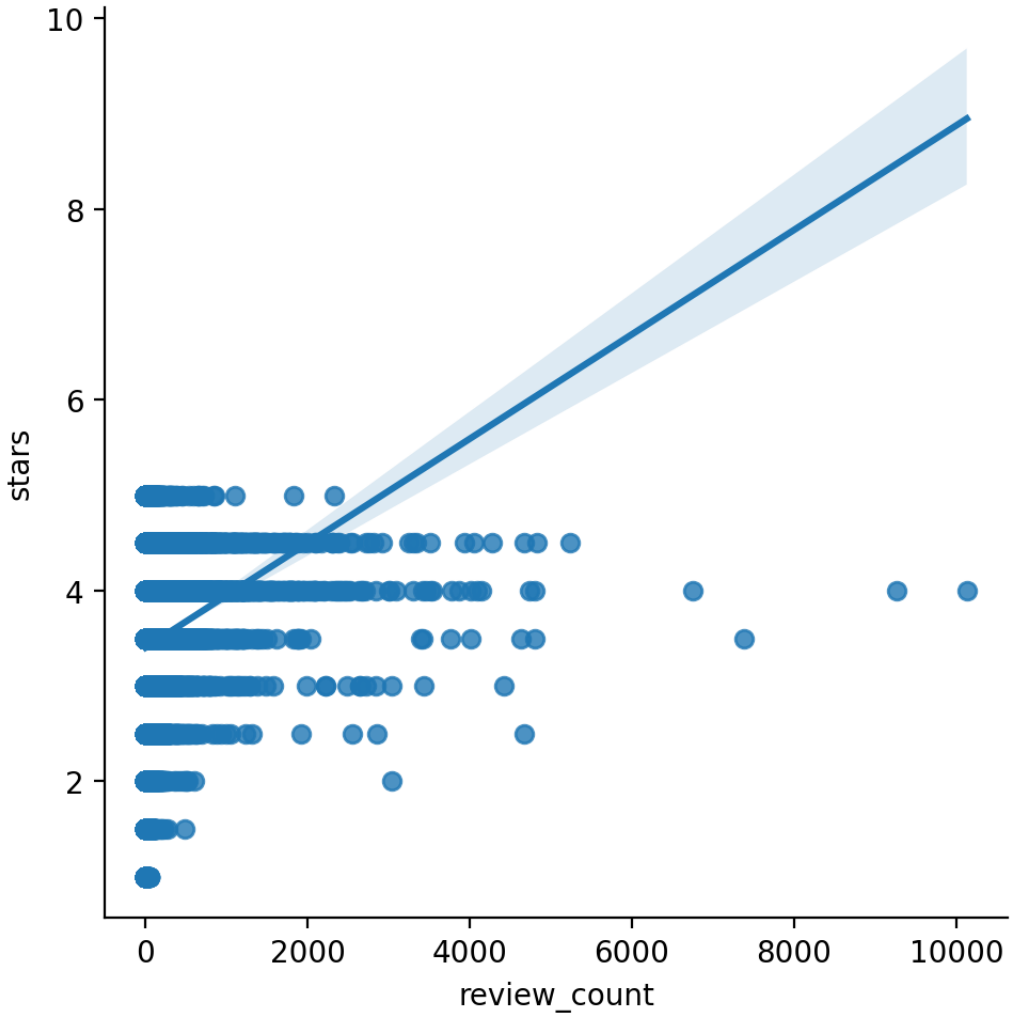
The above result shows that the regressor only estimates values between 3.5 and 5, although it is seen that with few reviews there are very bad ratings. However, we are not able to capture that behavior because there are too many review count values in a very small range.

Analyze the distribution of the `reviews_count` variable to see if we can solve this problem and propose any transformation of the `review_count` variable that you think may help the behavior of the linear regressor. Try to motivate or justify this transformation based on the knowledge of the data and not on the final result...

SOLUTION

In [13]:
```
sns.lmplot(data = data_yelp_df, x = "review_count", y = "stars")
```

Out[13]:
```
<seaborn.axisgrid.FacetGrid at 0x78668f3d4080>
```



2.4 Using other observations: proximity between businesses

Restaurants sometimes appear in the same neighborhoods, and restaurants that are close to each other may have similar *ratings*. In the database we have the **latitude** and **longitude** of each establishment, and we can use these two variables to find the restaurants that are closest to each other.

However, in order to include these two variables, we must consider whether the linear regression model is still the most appropriate. With the number of reviews, it seemed that something linear made sense, a site with more reviews should tend to have better ratings; but, can this model capture the proximity relationships? It seems clear that the answer to this question is **no**. So, we will change the model to one based on nearest neighbors.

Train a K-NN regression model and select by CV the number of neighbors (you can explore this parameter in the range $[1, 5, 10, 25, 50, 100, 250, 500]$). Firstly, evaluate the performance of the model using only the geographical information

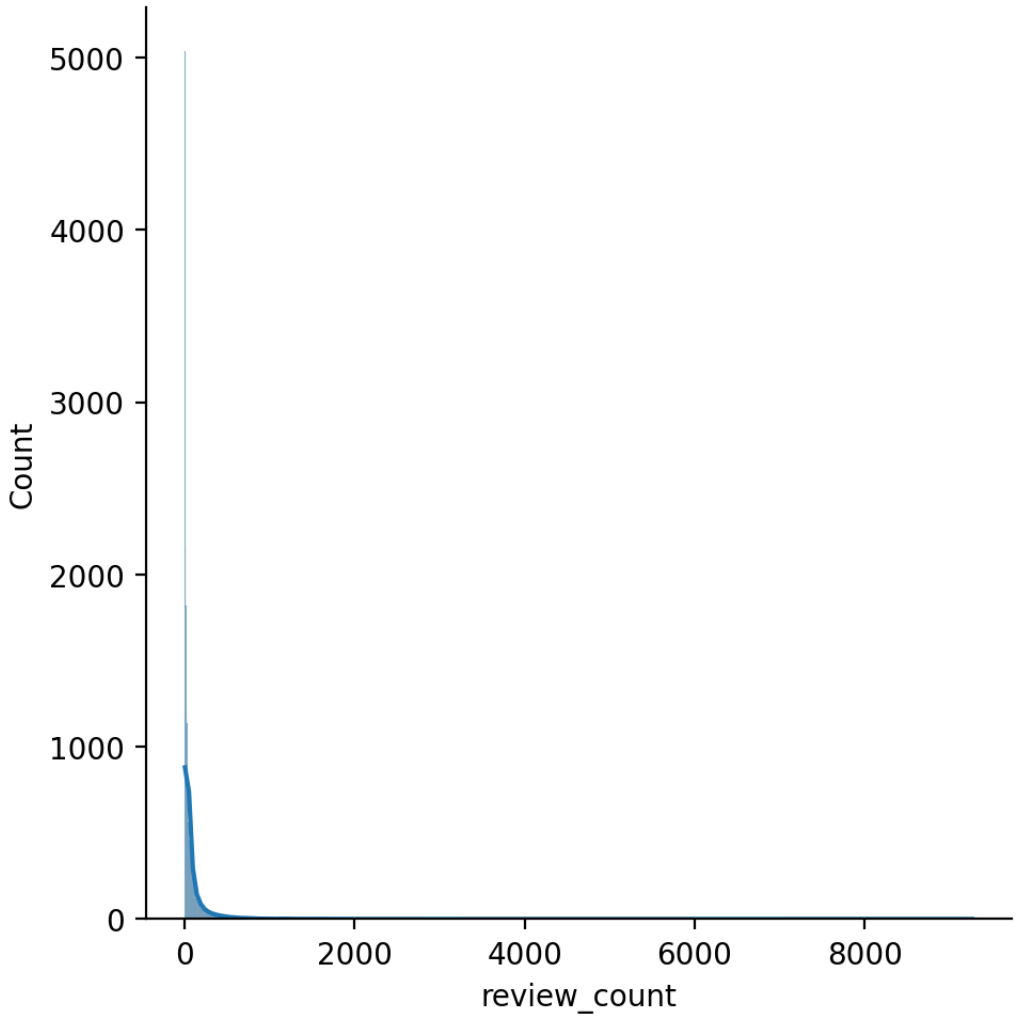
and, later, use it in combination with the number of reviews. In this last case, consider the case of using the raw `number_reviews` variable and its transformed version.

Finally, compare the different results, as well as the number of neighbors used by each approach. What do you think accounts for the large differences in the number of neighbors used by each model?

SOLUTION: using only log and lat information

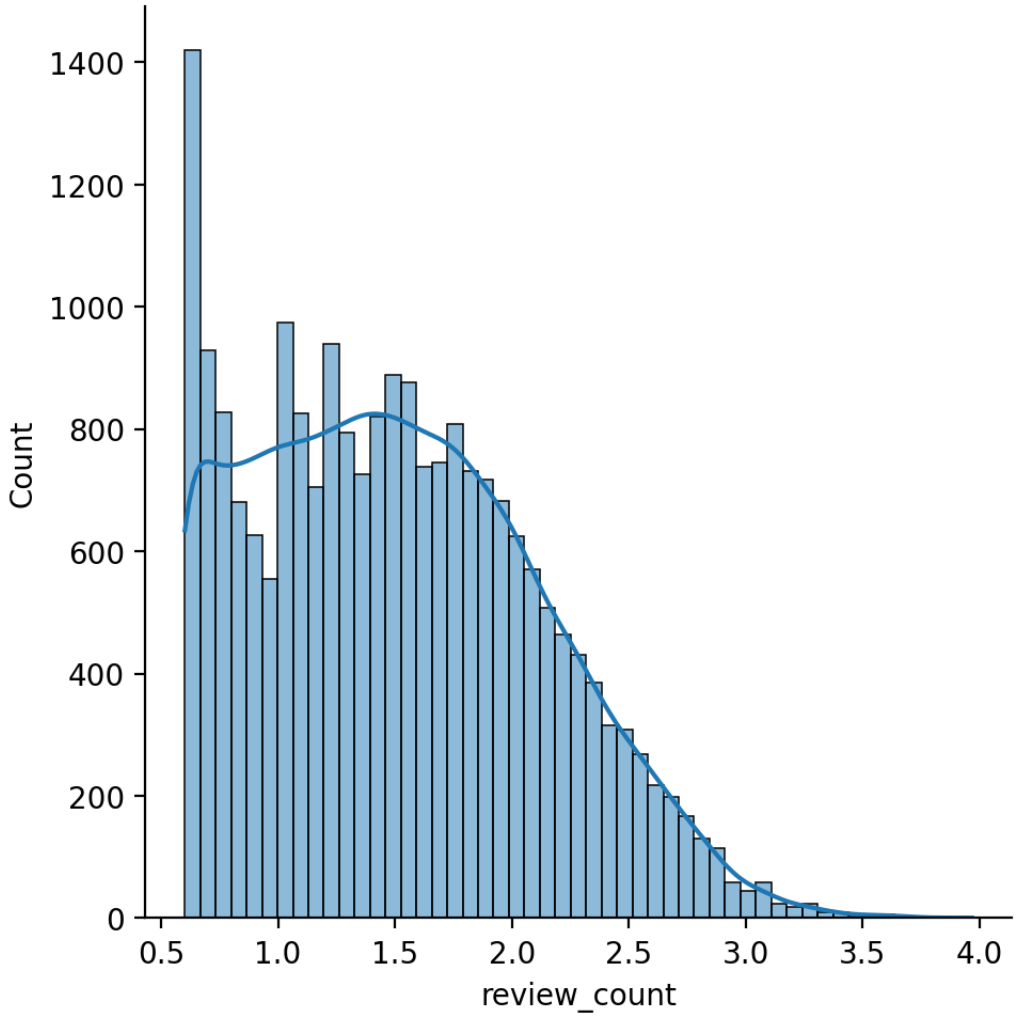
In [14]:
```
sns.displot(data=train_df, x="review_count", kde=True)
```

Out[14]:
```
<seaborn.axisgrid.FacetGrid at 0x78668f283530>
```



In [15]:
```
sns.displot(data=train_df[["review_count"]].apply(lambda x: np.log10(x
```

Out[15]:
```
<seaborn.axisgrid.FacetGrid at 0x78668e9930b0>
```



In [16]:
```
from sklearn.neighbors import KNeighborsRegressor
train_df_geo = train_df[train_df['latitude'].notnull() & train_df['lon
test_df_geo = test_df[test_df['latitude'].notnull() & test_df['longitu
x_train_geo = train_df[['latitude', 'longitude']]
y_train_geo = train_df_geo['stars']
x_test_geo = test_df_geo[['latitude', 'longitude']]
y_test_geo = test_df_geo['stars']

potential_k = [1,5,10,25,50,100,250,500]
best_k = None
best_mse = float('inf')
best_r2 = float('-inf')

for i in potential_k:
    knn_reg = KNeighborsRegressor(n_neighbors=i)
    knn_reg.fit(x_train_geo, y_train_geo)
    y_pred_knn_geo = knn_reg.predict(x_test_geo)

    mse_knn = mean_squared_error(y_test_geo, y_pred_knn_geo)
    r2_knn = r2_score(y_test_geo, y_pred_knn_geo)


    if mse_knn < best_mse:
```

```
        best_mse = mse_knn
        best_r2 = r2_knn
        best_k = i

global_results['knn_regressor'] = {'best_k ': best_k, 'mse': best_mse,
print("KNN Regressor")
print('\n', global_results['knn_regressor'])
```

```
KNN Regressor

{'best_k ': 50, 'mse': 0.6644327480325707, 'r2': 0.048233257645242666}
```

SOLUTION: combining log and lat information with `number_ratings`

SOLUTION: combining log and lat information with transformed `number_ratings`

In [17]:
```
from sklearn.neighbors import KNeighborsRegressor
train_df_geo = train_df[train_df['latitude'].notnull() & train_df['lon
test_df_geo = test_df[test_df['latitude'].notnull() & test_df['longitu
x_train_geo = train_df_geo[['latitude', 'longitude', 'review_count']]
y_train_geo = train_df_geo['stars']
x_test_geo = test_df_geo[['latitude', 'longitude', 'review_count']]
y_test_geo = test_df_geo['stars']

potential_k = [1,5,10,25,50,100,250,500]
best_k = None
best_mse = float('inf')
best_r2 = float('-inf')

for i in potential_k:
    knn_reg = KNeighborsRegressor(n_neighbors=i)
    knn_reg.fit(x_train_geo, y_train_geo)
    y_pred_knn_geo = knn_reg.predict(x_test_geo)

    mse_knn = mean_squared_error(y_test_geo, y_pred_knn_geo)
    r2_knn = r2_score(y_test_geo, y_pred_knn_geo)


    if mse_knn < best_mse:
        best_mse = mse_knn
        best_r2 = r2_knn
        best_k = i

global_results['knn_regressor_with_review_count'] = {'best_k ': best_k
print("KNN Regressor")
print('\n', global_results['knn_regressor_with_review_count'])
```

KNN Regressor

```
{'best_k ': 250, 'mse': 0.6355372988218169, 'r2': 0.08962454614754933}
```

For this next one we can apply the log to the review_count

In [18]:
```
train_df_geo = train_df[train_df['latitude'].notnull() & train_df['lon
test_df_geo = test_df[test_df['latitude'].notnull() & test_df['longitu

# log10_review_count already exists, so just use it directly
x_train_geo = train_df_geo[['latitude', 'longitude', 'log10_review_cou
y_train_geo = train_df_geo['stars']
x_test_geo = test_df_geo[['latitude', 'longitude', 'log10_review_count
y_test_geo = test_df_geo['stars']

potential_k = [1,5,10,25,50,100,250,500]
best_k = None
best_mse = float('inf')
best_r2 = float('-inf')

for i in potential_k:
    knn_reg = KNeighborsRegressor(n_neighbors=i)
    knn_reg.fit(x_train_geo, y_train_geo)
    y_pred_knn_geo = knn_reg.predict(x_test_geo)

    mse_knn = mean_squared_error(y_test_geo, y_pred_knn_geo)
    r2_knn = r2_score(y_test_geo, y_pred_knn_geo)


    if mse_knn < best_mse:
        best_mse = mse_knn
        best_r2 = r2_knn
        best_k = i

global_results['knn_regressor_log_review_counts'] = {'best_k': best_k,
print("KNN Regressor with log10_review_count")
print('\n', global_results['knn_regressor_log_review_counts'])
```

KNN Regressor with log10_review_count

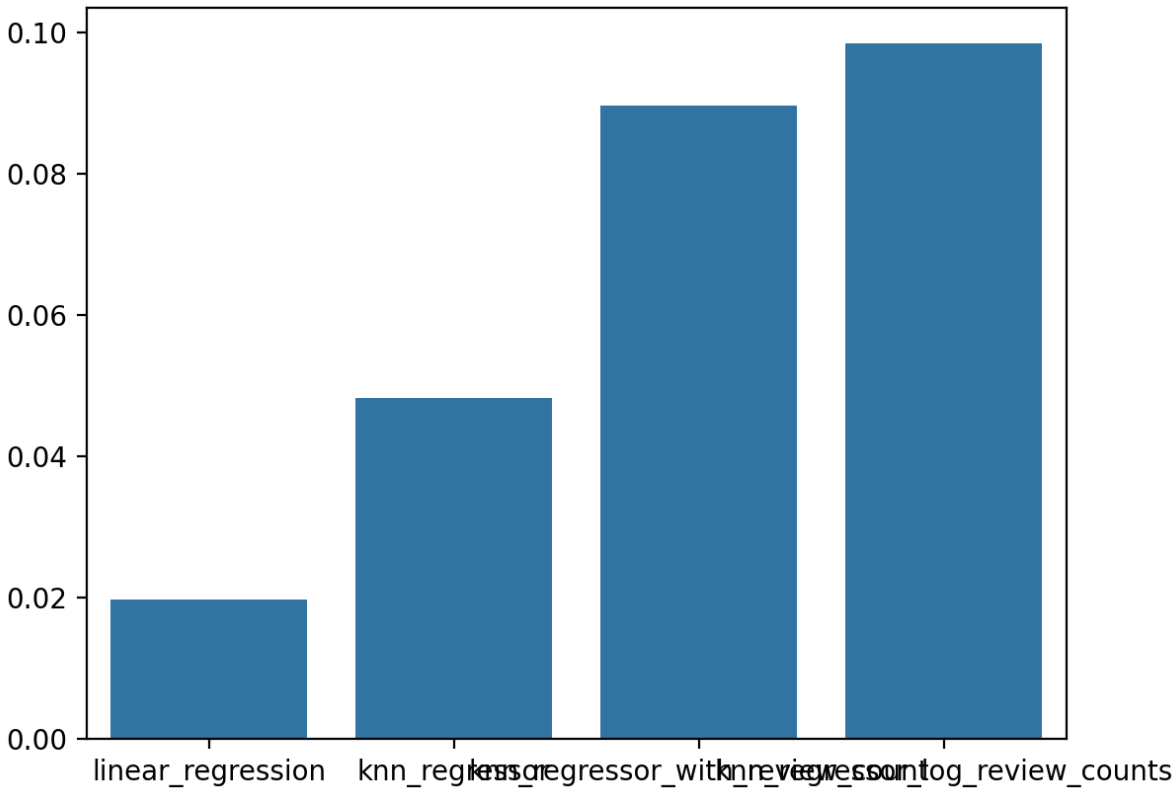
```
{'best_k': 100, 'mse': 0.6293704009916754, 'r2': 0.0984583194310128}
```

In [19]:
```
print(global_results)
```

```
{'linear_regression': {'weights': array([0.00058963]), 'intercept': np.
float64(3.395217653590921), 'mse': 0.6843743355525993, 'r2': 0.01966792
9028433506}, 'knn_regressor': {'best_k ': 50, 'mse': 0.664432748032570
7, 'r2': 0.048233257645242666}, 'knn_regressor_with_review_count': {'be
st_k ': 250, 'mse': 0.6355372988218169, 'r2': 0.08962454614754933}, 'kn
n_regressor_log_review_counts': {'best_k': 100, 'mse': 0.62937040099167
54, 'r2': 0.0984583194310128}}
```

In [20]:
```
sns.barplot(x=list(global_results.keys()), y=[global_results[k]['r2']
```

Out[20]: <Axes: >



2.5 Scaling of variables

Although combining variables provides a clear advantage, we have seen that using the raw or transformed number of ratings, changes the number of neighbors the model uses... This is basically because the different variables have different scales and as we know this significantly affects the performance of the K-NN. Include now a normalization of the data, to zero mean and unit standard deviation, to correct for this effect. For this analysis, you can directly use the transformed version of number of ratings.

Note: It is possible that for this step the effect of the scaling has not an impact of the results, but including the normalization with be useful for further steps where we include additional features.

Remember that within *scikit learn* we can automate scaling using the functionality `Pipeline`.

SOLUTION

In [21]:
```
X_train = train_df.loc[:,['latitude','longitude','review_count']].valu
Y_train = train_df['stars'].values
X_test = test_df.loc[:,['latitude','longitude','review_count']].values
```

```
Y_test = test_df['stars'].values

# Use the existing log10_review_count column that was created earlier
X_train_log = train_df.loc[:,['latitude','longitude','log10_review_cou
X_test_log = test_df.loc[:,['latitude','longitude','log10_review_count

print(f"X_train_log shape: {X_train_log.shape}")
print(f"X_test_log shape: {X_test_log.shape}")
print("Using existing log10_review_count column")
```

```
X_train_log shape: (21982, 3)
X_test_log shape: (21983, 3)
Using existing log10_review_count column
```

In [22]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

# Pipeline definition. List with all the stages of the pipeline
knn_pipe = Pipeline([('scaler', StandardScaler()), ('regressor',KNeigh
# Dictionary with hyperparameters for all the stages of the pipeline,
# that will be tuned with crossvalidation
params_pipe = {'regressor__n_neighbors':[1,5,10,25,50,100,250,500],
               'scaler':[StandardScaler(),'passthrough']}
# 'passthrough' means skip this stage.
# This way we can estimate with crossvalidation if scaling was a good
# set up the grid to optimize the hyperparameters and train the pipeli
grid_pipe_log = GridSearchCV(knn_pipe, param_grid= params_pipe, cv=5)
grid_pipe_log.fit(X_train_log, Y_train)
print("Score with the training data R^2={0:.4f}".format(grid_pipe_log.
print("Score with the test data R^2={0:.4f}".format(grid_pipe_log.scor
print("Hyperparameters choosen with cross-validation")
print(grid_pipe_log.best_params_)
global_results['knn + scale geo + log review'] = grid_pipe_log.score(X
```

```
Score with the training data R^2=0.1170
Score with the test data R^2=0.0985
Hyperparameters choosen with cross-validation
{'regressor__n_neighbors': 100, 'scaler': 'passthrough'}
```

2.6 Removing isolated restaurants

In this analysis we are including all the restaurants that are available, but if we analyze how they are distributed by cities, we find that there are cities with very few restaurants. By using the geographical information of the data, the restaurants in these cities can be considered a kind of **outliers** (atypical data) that do not follow the main data distribution and whose distribution is difficult to learn.

Although we could directly use the `city` field to remove these restaurants, in

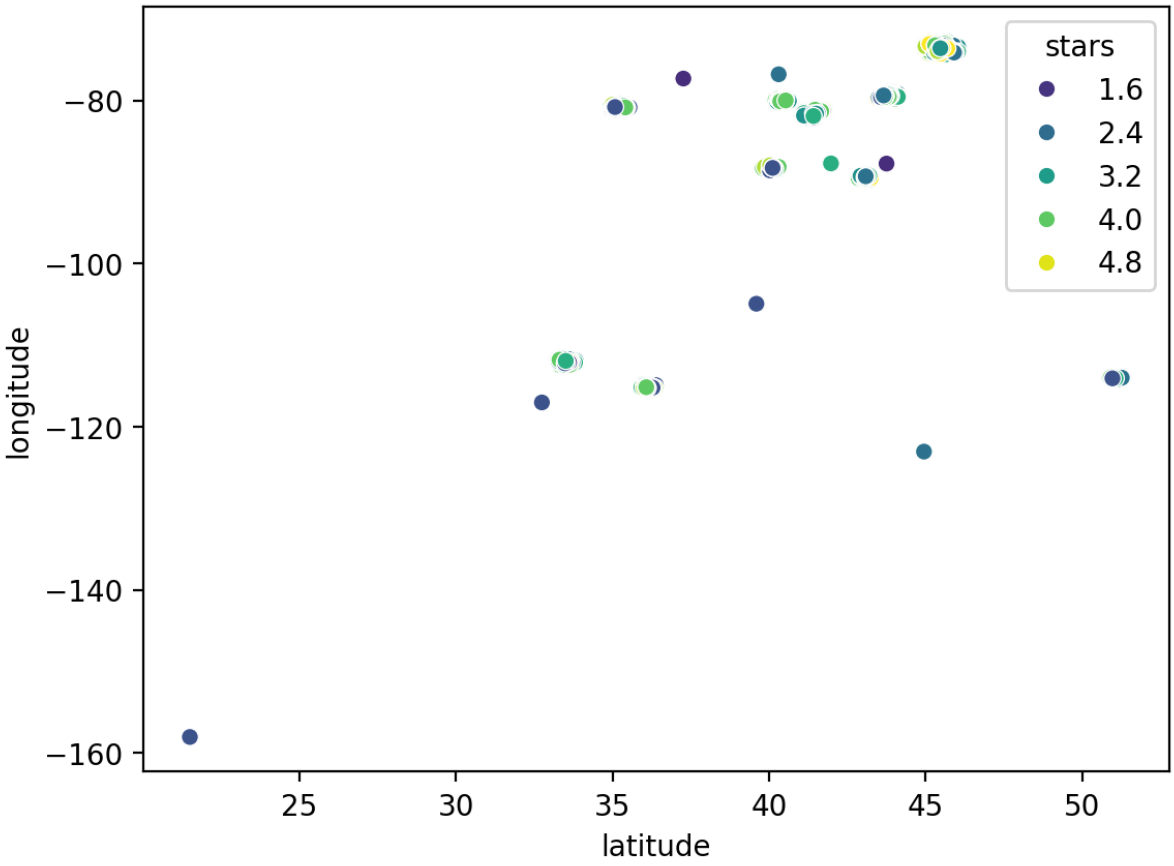
this section we are going to use the geographical coordinates as input to a novelty detection model and remove these outliers. To do this, start by visualizing in a two-dimensional space the positions of the restaurants to see how they are distributed in the space and, based on this distribution, indicate which novelty detection model is most appropriate to eliminate the most isolated restaurants. Then, learn the model with the training data (you can visually adjust the percentage of contamination or outliers and other model parameters if are needed) and then remove the outliers from both the training and test set.

Finally, obtain the performance of the K-NN classifier, using the geographic information and the (transformed) number of ratings, on this *clean* data set.

SOLUTION

In [23]:
```
sns.scatterplot(data = data_yelp_df, x = 'latitude', y = 'longitude',
```

Out[23]:
```
<Axes: xlabel='latitude', ylabel='longitude'>
```

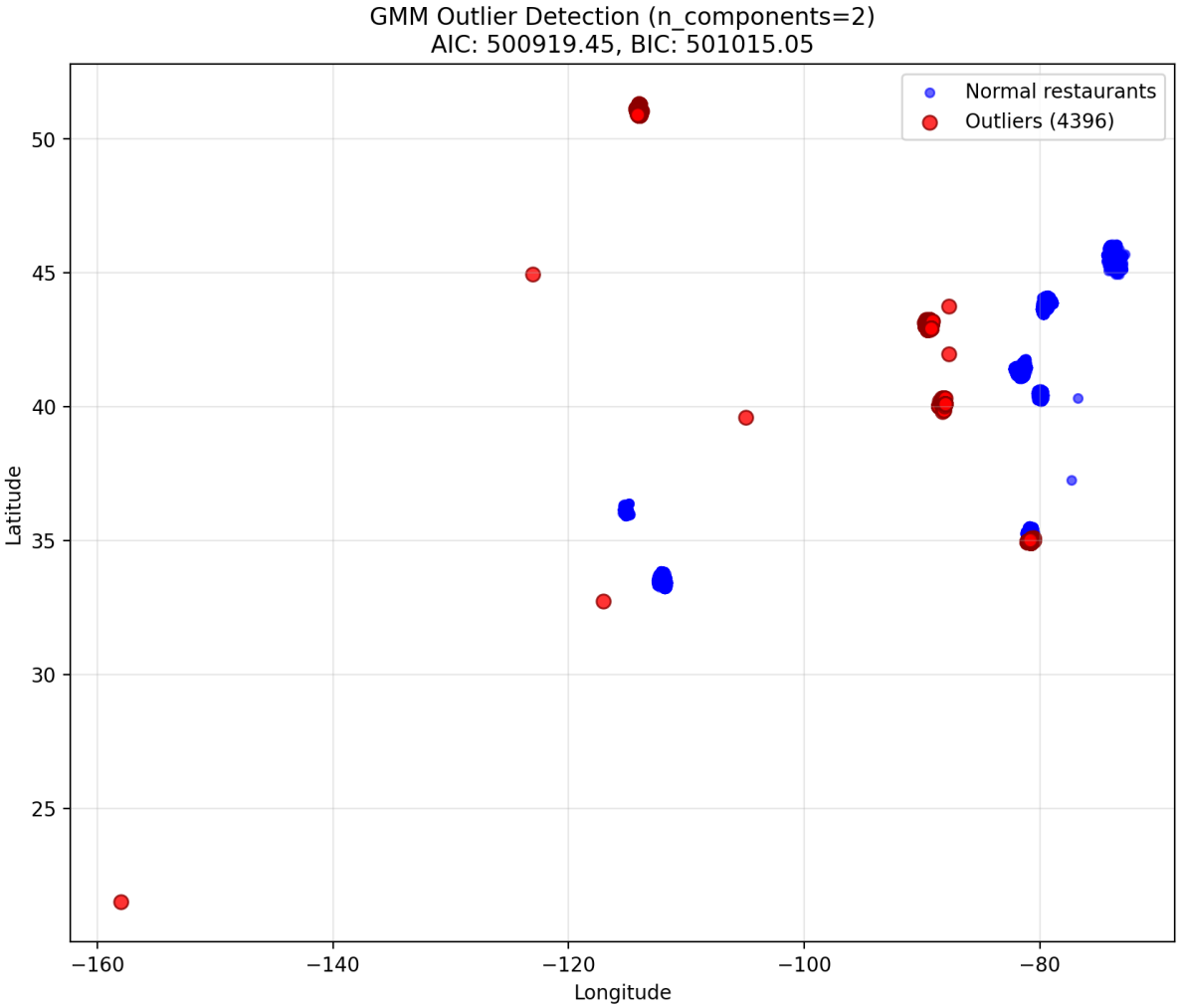


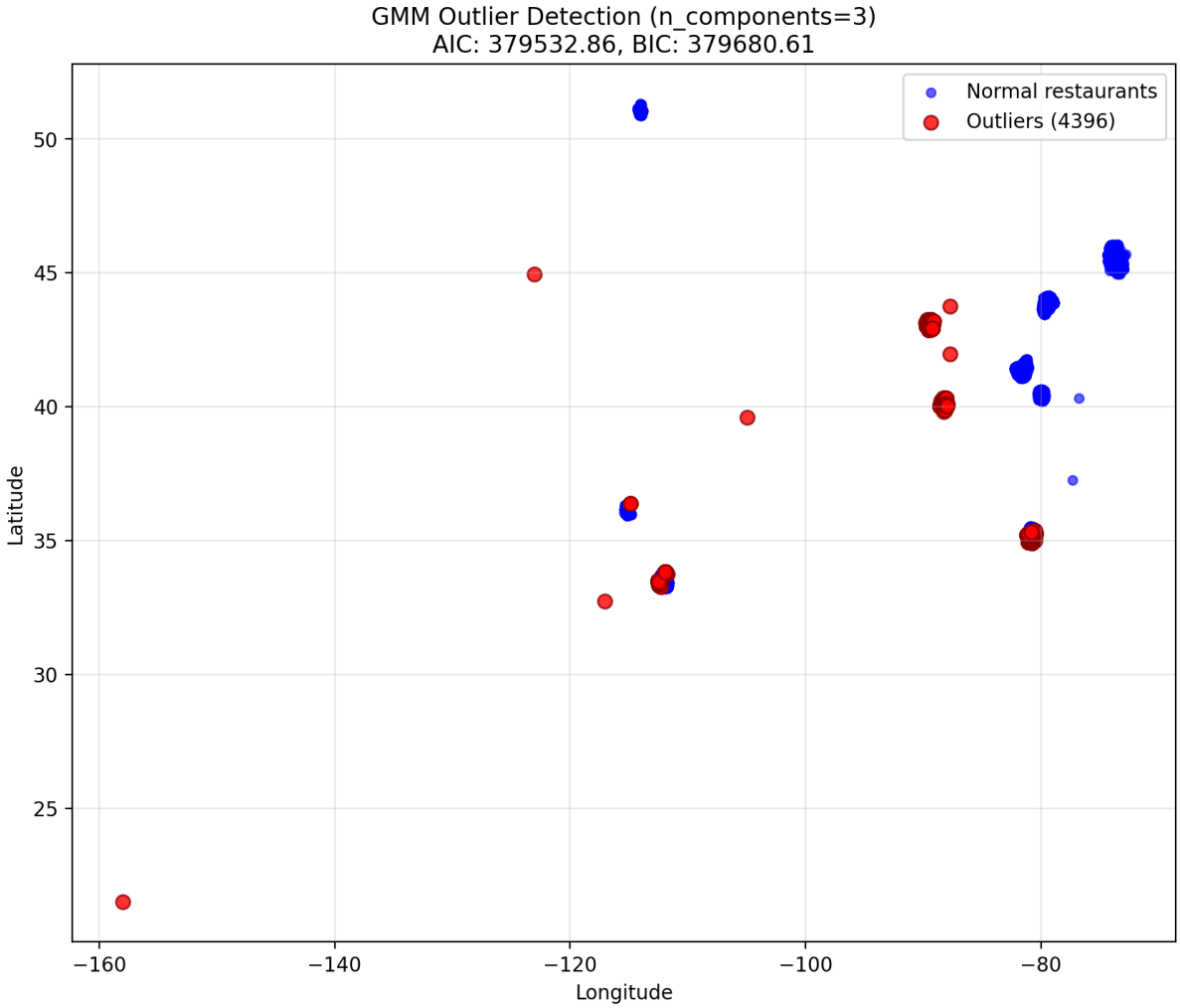
In [24]:
```
from sklearn.mixture import GaussianMixture

# Get geographic coordinates correctly
geo_coords = data_yelp_df[['latitude', 'longitude']].values

# Store results for different numbers of components
```

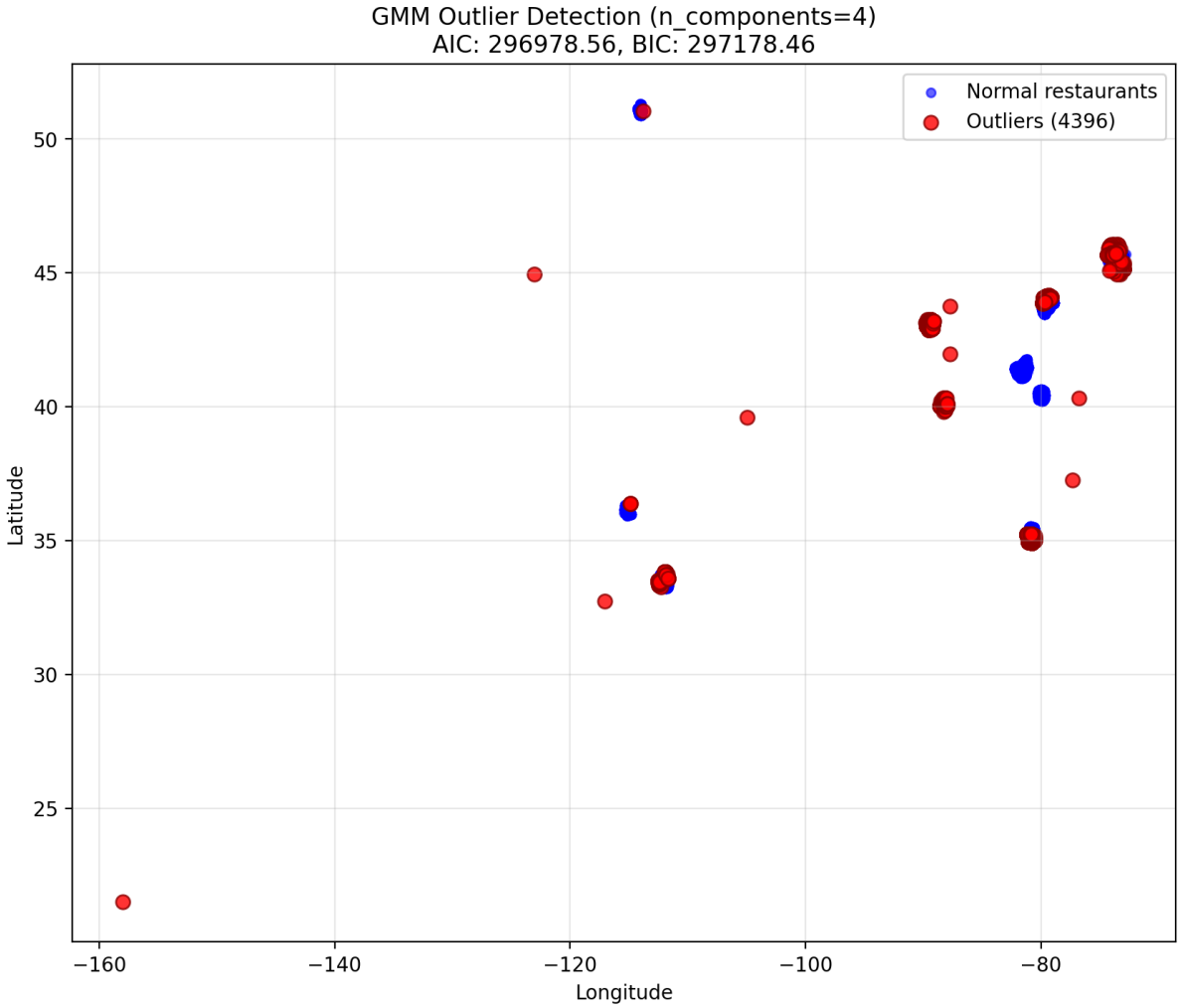
```
gmm_results = {}

for i in [2, 3, 4, 5, 6, 7, 8, 9, 10]:
    gmm = GaussianMixture(n_components=i, covariance_type='full', n_in

    # Fit on geographic coordinates
    gmm.fit(geo_coords)

    # Get log probabilities for each point
    log_probs = gmm.score_samples(geo_coords)

    # Set contamination fraction
    frac = 0.1
    idx_outliers = np.argsort(log_probs)[:int(np.round(len(geo_coords)

    # Create binary outlier labels
    outlier_labels = np.zeros(len(geo_coords))
    outlier_labels[idx_outliers] = 1

    # Store results
    gmm_results[i] = {
        'log_probs': log_probs,
        'outliers': idx_outliers,
        'outlier_labels': outlier_labels,
        'aic': gmm.aic(geo_coords),
        'bic': gmm.bic(geo_coords)
    }

    # Create visualization
    fig, ax = plt.subplots(figsize=(10, 8))

    # Plot all points (normal ones)
    normal_mask = outlier_labels == 0
    ax.scatter(geo_coords[normal_mask, 1], geo_coords[normal_mask, 0],
               c='blue', s=20, alpha=0.6, label='Normal restaurants')

    # Plot outliers
    ax.scatter(geo_coords[idx_outliers, 1], geo_coords[idx_outliers, 0
               marker="o", facecolor="red", edgecolor="darkred", s=50,
               alpha=0.8, label=f'Outliers ({len(idx_outliers)})')

    plt.xlabel('Longitude')
    plt.ylabel('Latitude')
    plt.title(f'GMM Outlier Detection (n_components={i})\nAIC: {gmm.ai
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.show()

# Find best number of components using BIC
best_n_components = min(gmm_results.keys(), key=lambda k: gmm_results[
print(f"\nBest number of components (lowest BIC): {best_n_components}"
```

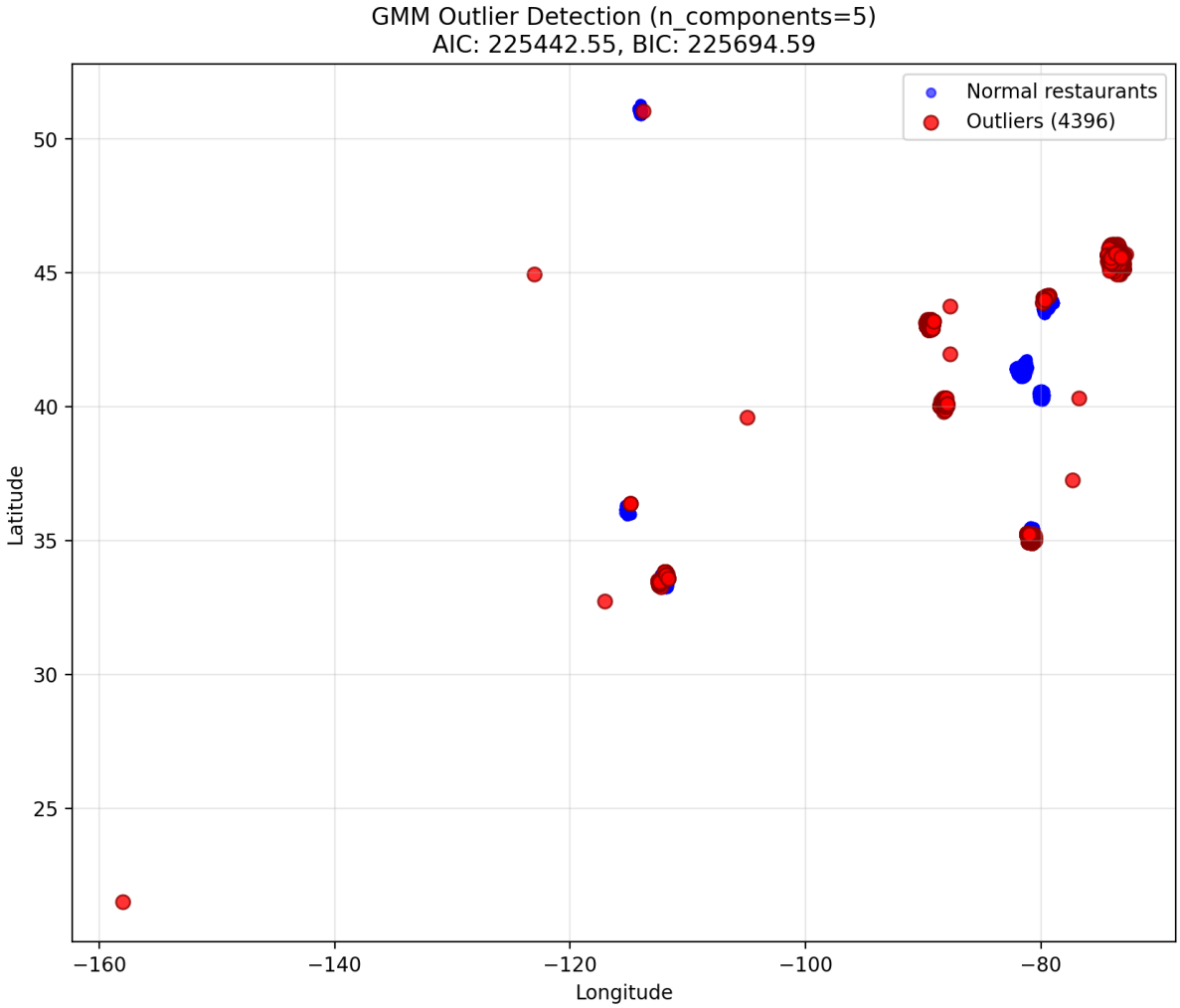
```
# Apply outlier removal to train/test splits and evaluate K-NN perform
print("\nApplying outlier removal to training and test data...")

# Get the best GMM model
best_gmm = GaussianMixture(n_components=best_n_components, covariance_
                           n_init=20, random_state=42)

# Fit on training data geographic coordinates only
train_geo = train_df[['latitude', 'longitude']].values
best_gmm.fit(train_geo)

# Detect outliers in training data
train_log_probs = best_gmm.score_samples(train_geo)
train_outlier_threshold = np.percentile(train_log_probs, frac * 100)
train_outliers = train_log_probs < train_outlier_threshold

# Detect outliers in test data
test_geo = test_df[['latitude', 'longitude']].values
test_log_probs = best_gmm.score_samples(test_geo)
test_outliers = test_log_probs < train_outlier_threshold

# Remove outliers from datasets
train_clean = train_df[~train_outliers].copy()
test_clean = test_df[~test_outliers].copy()

print(f"Original training size: {len(train_df)}, After outlier removal
print(f"Original test size: {len(test_df)}, After outlier removal: {le

# Prepare features for K-NN (geographic + log-transformed review count
if 'log_review_count' not in train_clean.columns:
    train_clean['log_review_count'] = np.log10(train_clean['review_cou
    test_clean['log_review_count'] = np.log10(test_clean['review_count

X_train_clean = train_clean[['latitude', 'longitude', 'log_review_coun
y_train_clean = train_clean['stars'].values
X_test_clean = test_clean[['latitude', 'longitude', 'log_review_count'
y_test_clean = test_clean['stars'].values

# Train K-NN with scaling on clean data
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

knn_pipe_clean = Pipeline([
    ('scaler', StandardScaler()),
    ('regressor', KNeighborsRegressor())
])

params_pipe_clean = {
    'regressor__n_neighbors': [1, 5, 10, 25, 50, 100, 250, 500],
    'scaler': [StandardScaler(), 'passthrough']
}
```

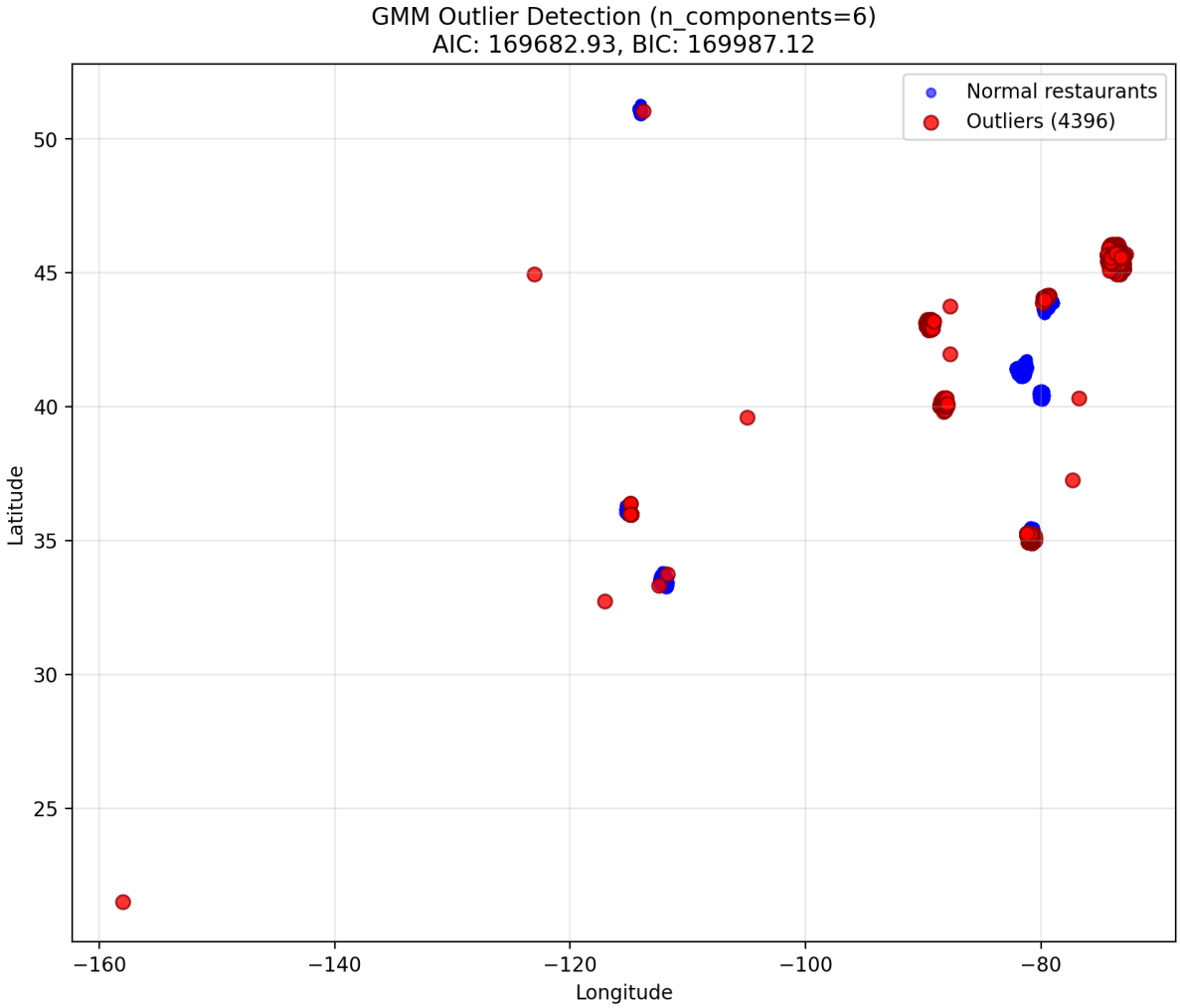
```
grid_clean = GridSearchCV(knn_pipe_clean, param_grid=params_pipe_clean
grid_clean.fit(X_train_clean, y_train_clean)

print(f"\nK-NN Performance on Clean Data:")
print(f"Training R²: {grid_clean.score(X_train_clean, y_train_clean):.
print(f"Test R²: {grid_clean.score(X_test_clean, y_test_clean):.4f}")
print(f"Best parameters: {grid_clean.best_params_}")

# Store results
global_results['K-NN Clean Data (GMM)'] = {
    'R2': grid_clean.score(X_test_clean, y_test_clean),
    'n_components': best_n_components,
    'data_retained': len(train_clean) / len(train_df)
}
```

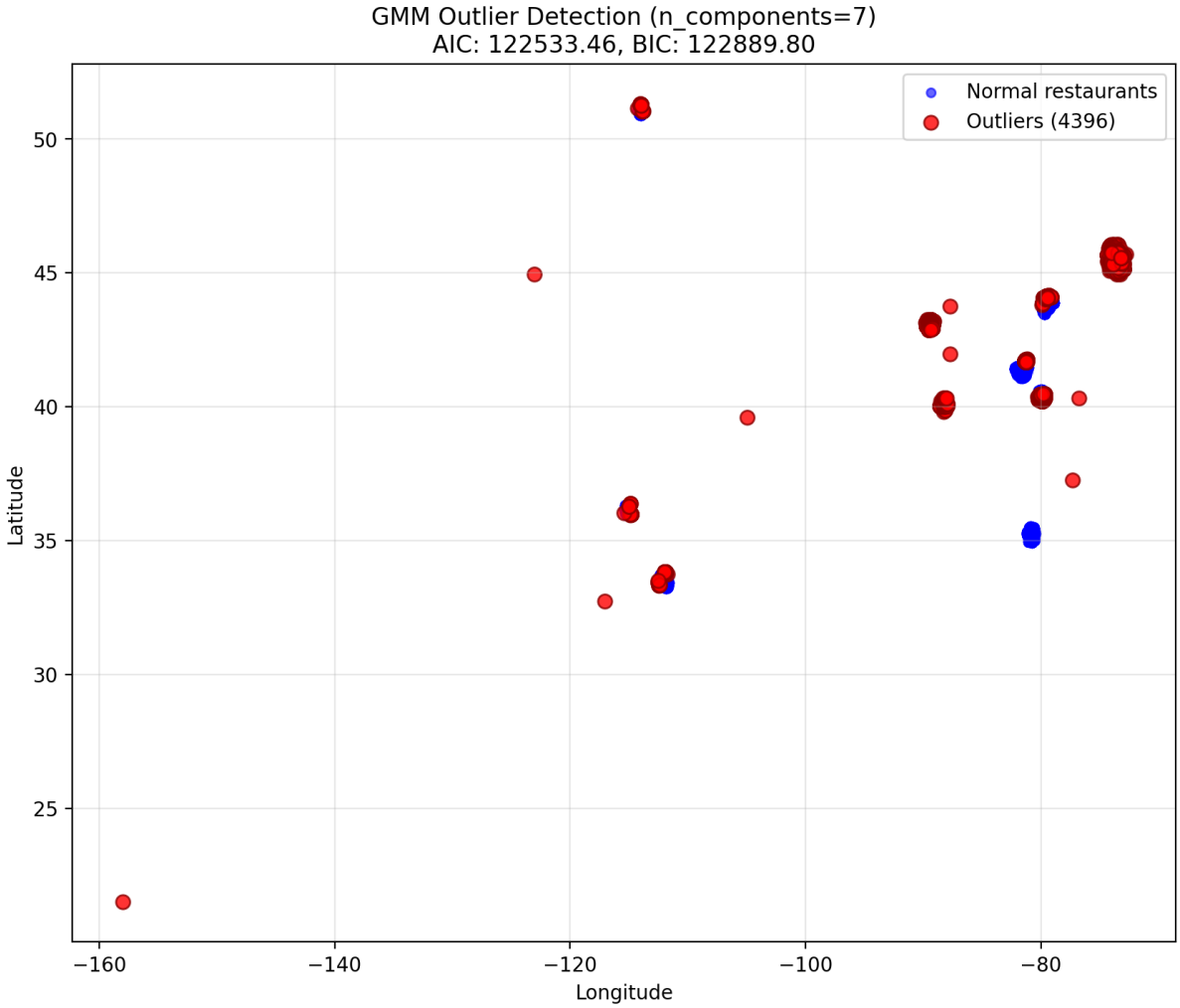


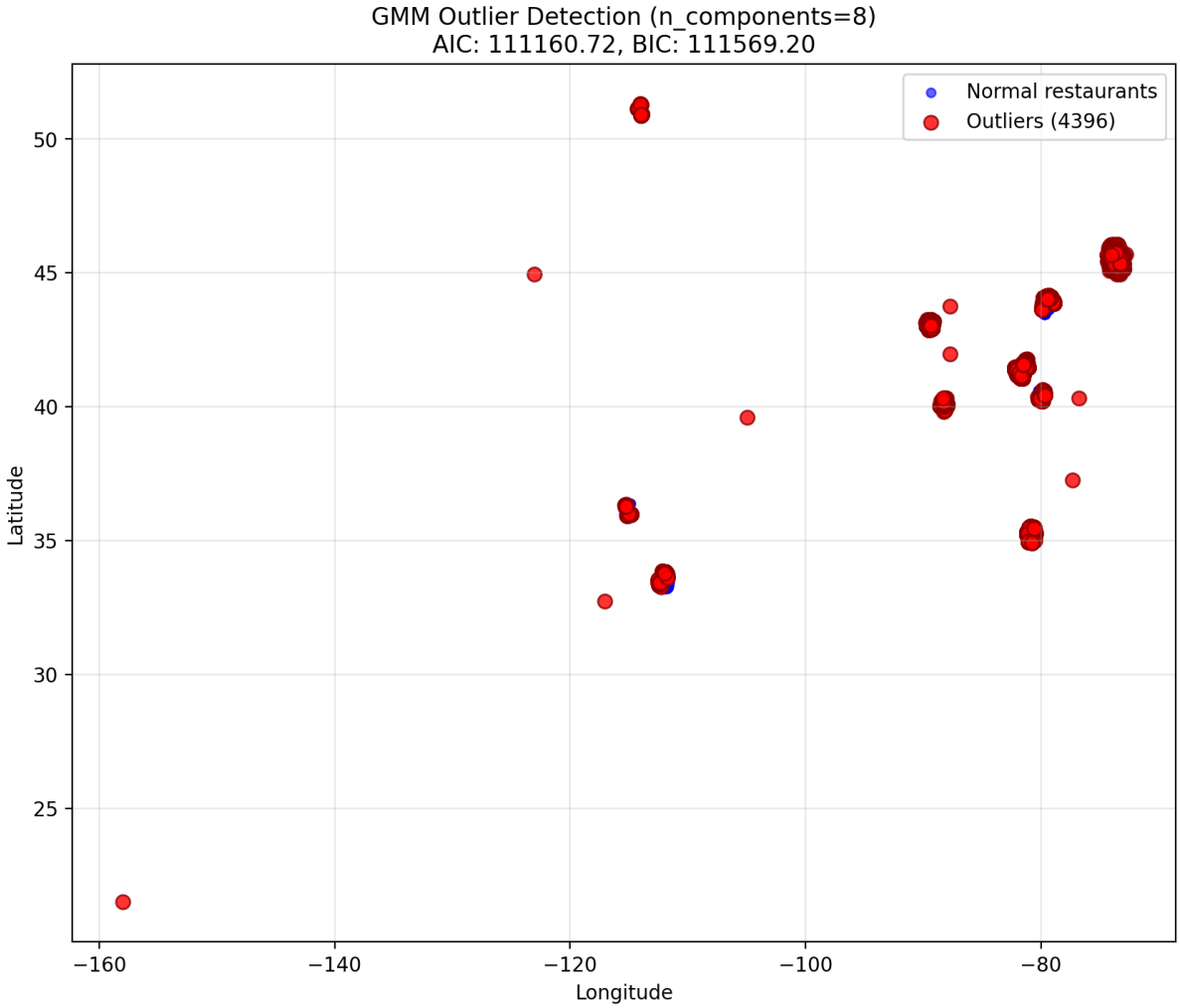

GMM Outlier Detection (n_components=3)
AIC: 379532.86, BIC: 379680.61
Normal restaurants
Outliers (4396)
Latitude
Longitude
50
45
40
35
30
25
−160
−140
−120
−100
−80

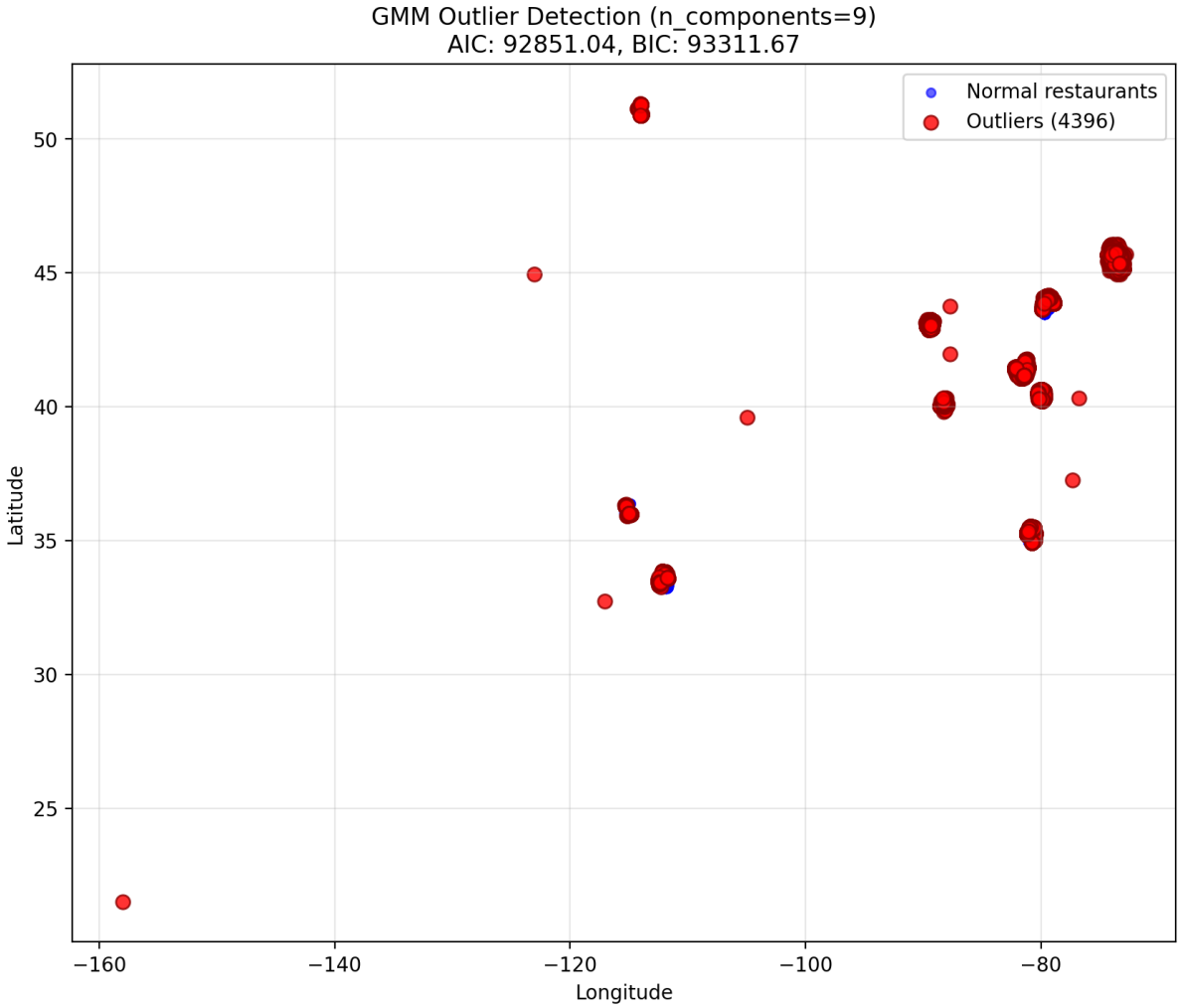

GMM Outlier Detection (n_components=4)
AIC: 296978.56, BIC: 297178.46
Normal restaurants
Outliers (4396)
Latitude
Longitude
50
45
40
35
30
25
−160
−140
−120
−100
−80

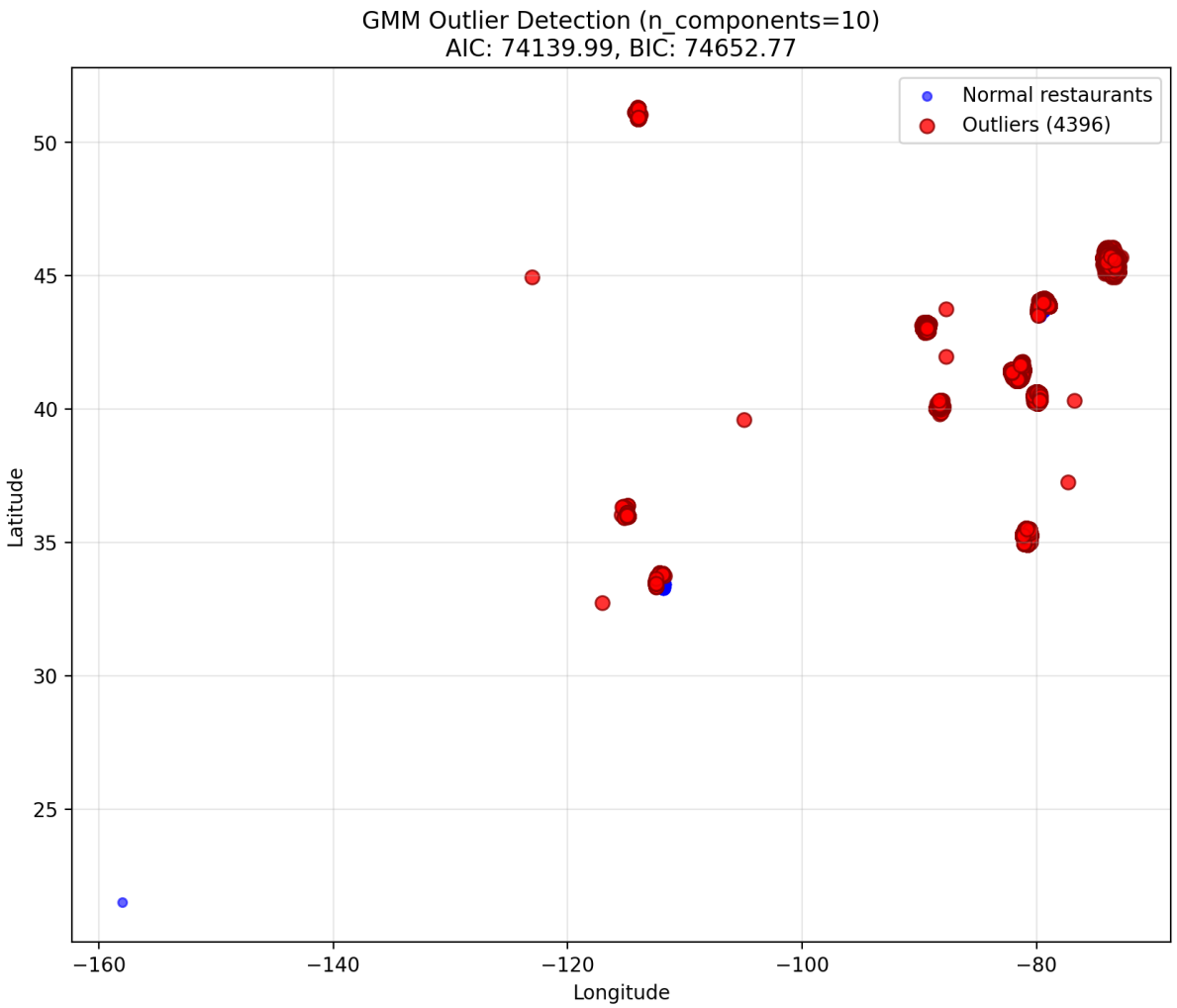

GMM Outlier Detection (n_components=5)
AIC: 225442.55, BIC: 225694.59
Normal restaurants
Outliers (4396)
Latitude
Longitude
50
45
40
35
30
25
−160
−140
−120
−100
−80


GMM Outlier Detection (n_components=6)
AIC: 169682.93, BIC: 169987.12
Normal restaurants
Outliers (4396)
Latitude
Longitude
50
45
40
35
30
25
−160
−140
−120
−100
−80


GMM Outlier Detection (n_components=7)
AIC: 122533.46, BIC: 122889.80
Normal restaurants
Outliers (4396)
Latitude
Longitude
50
45
40
35
30
25
−160
−140
−120
−100
−80


GMM Outlier Detection (n_components=8)
AIC: 111160.72, BIC: 111569.20
Normal restaurants
Outliers (4396)
Latitude
Longitude
50
45
40
35
30
25
−160
−140
−120
−100
−80


GMM Outlier Detection (n_components=9)
AIC: 92851.04, BIC: 93311.67
Normal restaurants
Outliers (4396)
Latitude
Longitude
50
45
40
35
30
25
−160
−140
−120
−100
−80




```
Best number of components (lowest BIC): 10

Applying outlier removal to training and test data...
Original training size: 21982, After outlier removal: 19783
Original test size: 21983, After outlier removal: 19911

K-NN Performance on Clean Data:
Training R²: 0.1104
Test R²: 0.1022
Best parameters: {'regressor__n_neighbors': 250, 'scaler': 'passthroug
h'}
```

In [25]:
```
global_results
```

Out[25]:
```
{'linear_regression': {'weights': array([0.00058963]),
  'intercept': np.float64(3.395217653590921),
  'mse': 0.6843743355525993,
  'r2': 0.019667929028433506},
 'knn_regressor': {'best_k ': 50,
  'mse': 0.6644327480325707,
  'r2': 0.048233257645242666},
 'knn_regressor_with_review_count': {'best_k ': 250,
  'mse': 0.6355372988218169,
  'r2': 0.08962454614754933},
 'knn_regressor_log_review_counts': {'best_k': 100,
  'mse': 0.6293704009916754,
  'r2': 0.0984583194310128},
 'knn + scale geo + log review': 0.0984583194310128,
 'K-NN Clean Data (GMM)': {'R2': 0.10215167032061201,
  'n_components': 10,
  'data_retained': 0.8999636065872078}}
```

2.7 Other sources of information: extracting attributes.

We can complete the model with another variables. The field `attributes` of the dataframe is a dictionary with information about different characteristics of the business.

Extracting new fields from attributes

Let's start checking the values of field `attributes`, counting the number of occurrences of each field and selecting the ten fields that are more frequent along the different restaurants (use the train dataframe for this analysis). For this selection, take into account that some fields (such as `BusinessParking`) have a inner dictionary; to avoid more complex preprocessing ignore these nested fields and select the ten fields consisting in a single dictionary.

Besides, check the values of these new fields and **homogenize** and adequately codify them for its later use.

Note: the method `Counter` can be useful to complete this task.

SOLUTION

In [26]:
```
train_df['attributes']
```

Out[26]:

	attributes
64514	{'GoodForKids': 'True', 'RestaurantsReservatio...
201206	None
202245	{'BYOBCorkage': ''no'', 'RestaurantsAttire': '...
128024	{'RestaurantsGoodForGroups': 'True', 'GoodForK...
88564	{'DogsAllowed': 'False', 'Ambience': '{'touris...
...	...
30271	{'RestaurantsPriceRange2': '2', 'OutdoorSeatin...
54419	{'Alcohol': 'u'full_bar'', 'HappyHour': 'True'...
181417	{'Smoking': 'u'outdoor'', 'BusinessParking': '...
4145	{'Alcohol': 'u'full_bar'', 'RestaurantsPriceRa...
75278	{'RestaurantsAttire': ''casual'', 'GoodForKids...

21982 rows × 1 columns

dtype: object

Data visualization

Using the seaborn facilities, represent these new 10 variables (binary, categorical or ordinals), and/or combinations of them, and visually try to analyze whether or not they will be useful for the prediction of the number of ratings.

Note: before plotting them, you should remove `NaN` entries (you can easily use `.'dropna()` method of pandas dataframes).

SOLUTION

In [27]:
```
def extract_keys(my_att):
  if my_att is not None:
    return list(my_att.keys())

keys_attributes= train_df['attributes'].apply(lambda my_att: extract_k
# Extract list with all the fields
list_keys = []
for i in range(len(keys_attributes)):
  my_keys =keys_attributes.iloc[i]
  if my_keys is not None:
    list_keys += my_keys
```

```
  # Count the number
from collections import Counter
my_count = Counter(list_keys)
print(my_count)
```

```
Counter({'RestaurantsTakeOut': 19204, 'BusinessParking': 18546, 'Restau
rantsDelivery': 18470, 'RestaurantsReservations': 18331, 'RestaurantsPr
iceRange2': 18156, 'Ambience': 18004, 'RestaurantsGoodForGroups': 1764
7, 'HasTV': 17646, 'GoodForKids': 17563, 'OutdoorSeating': 17230, 'Rest
aurantsAttire': 16239, 'Alcohol': 15709, 'WiFi': 15603, 'BikeParking':
15447, 'GoodForMeal': 14445, 'NoiseLevel': 14387, 'Caters': 13619, 'Bus
inessAcceptsCreditCards': 12273, 'RestaurantsTableService': 8084, 'Happ
yHour': 5139, 'WheelchairAccessible': 4643, 'DogsAllowed': 4515, 'Musi
c': 2373, 'BestNights': 1797, 'DriveThru': 1772, 'BusinessAcceptsBitcoi
n': 1735, 'ByAppointmentOnly': 1619, 'GoodForDancing': 1541, 'CoatChec
k': 1409, 'Smoking': 1154, 'BYOBCorkage': 434, 'Corkage': 414, 'BYOB':
332, 'DietaryRestrictions': 23, 'AgesAllowed': 15, 'AcceptsInsurance':
7, 'RestaurantsCounterService': 4, 'HairSpecializesIn': 3, 'Open24Hour
s': 2})
```

In [28]:
```
select_att = my_count.most_common(12)
list_select_att =list(list(zip(*select_att))[0])

# Remove nested fields
list_select_att.remove('BusinessParking')
list_select_att.remove('Ambience')

print(list_select_att)
```

```
['RestaurantsTakeOut', 'RestaurantsDelivery', 'RestaurantsReservation
s', 'RestaurantsPriceRange2', 'RestaurantsGoodForGroups', 'HasTV', 'Goo
dForKids', 'OutdoorSeating', 'RestaurantsAttire', 'Alcohol']
```

In [29]:
```
# Generate variables
def extract_att_variable(my_att, key_name):
  if my_att is not None:
    if key_name in my_att.keys():
      return my_att[key_name]

for feat in list_select_att:
  train_df[feat]= train_df['attributes'].apply(lambda my_att: extract_
  test_df[feat]= test_df['attributes'].apply(lambda my_att: extract_at
```

In [30]:
```
# Unify some names
train_df.loc[:,'RestaurantsAttire'] = train_df['RestaurantsAttire'].re
test_df.loc[:,'RestaurantsAttire'] = test_df['RestaurantsAttire'].repl

train_df.loc[:,'Alcohol'] = train_df['Alcohol'].replace(["u'none'","'n
test_df.loc[:,'Alcohol'] = test_df['Alcohol'].replace(["u'none'","'non
```

In [31]:
```
train_df.head()
```

Out[31]:

	business_id	name	address	city	state	postal_c
64514	fiVH82y41YtGdndjT_wpzQ	Popeyes Louisiana Kitchen	9910 W Flamingo Rd	Las Vegas	NV	8
201206	ku3b6kxAmb05-rLblyQEnw	Tuk Tuk Thai	5763 Signal Hill Centre SW	Calgary	AB	T3H
202245	d_L-rfS1vT3JMzgCUGtiow	Border Grill	3950 S Las Vegas Blvd	Las Vegas	NV	8
128024	yRBE98DheftGKDwZG39P-w	Latazza Cafe	2200 Yonge Street	Toronto	ON	M4S
88564	d4P7boUqiA2pR59jOlzYLA	Assembly Chef's Hall	111 Richmond Street W	Toronto	ON	M5H

5 rows × 25 columns

Using these features with our learning models

To use these new features, we have two problems. First, there are missing values, so we have to impute their values; and, second, they are codified as strings, we have to convert to a numeric format to be used in sklearn models, even, to be used by the imputation methods.

To overcome these limitations, let's proceed as follows:

- Transform these fields into numeric features. So far, do not worry about one-hot encoding or other specific categorical codification, we want this first approach for a missing value imputation. Besides, `None` entries can be converted to `np.nan` values for a better processing.
- As we can see, there are missing values in some positions... the simplest strategy to use this variable (in case there are few missing values) may be to eliminate the data where it is not present. However, let's try to analyze the advantages that other imputation methods can give us. In particular, try to impute these values with the following strategies:
  - A simple imputation based in either the mean value, the median value or the most frequent value (mode)
  - An imputation based on K-NN, you can fix K=100 for this task (to reduce

the CV burden).

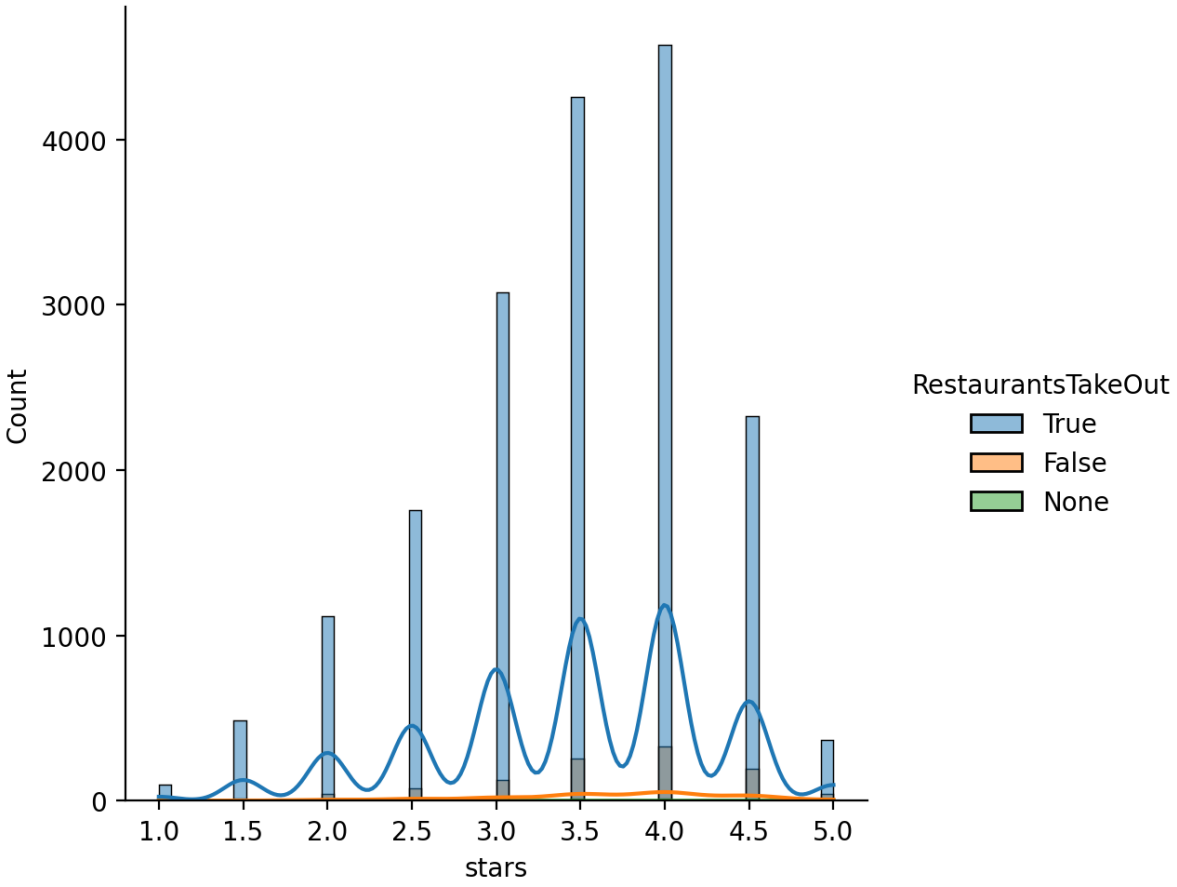
- Finally, codify adequately the categorical data and train a final K-NN using all the available information.

In case any of these fields is category and you have to convert it to one-hot-encoding, we advise you to use the `pd.get_dummies()` method of pandas since it easily handle the `NaN` positions.

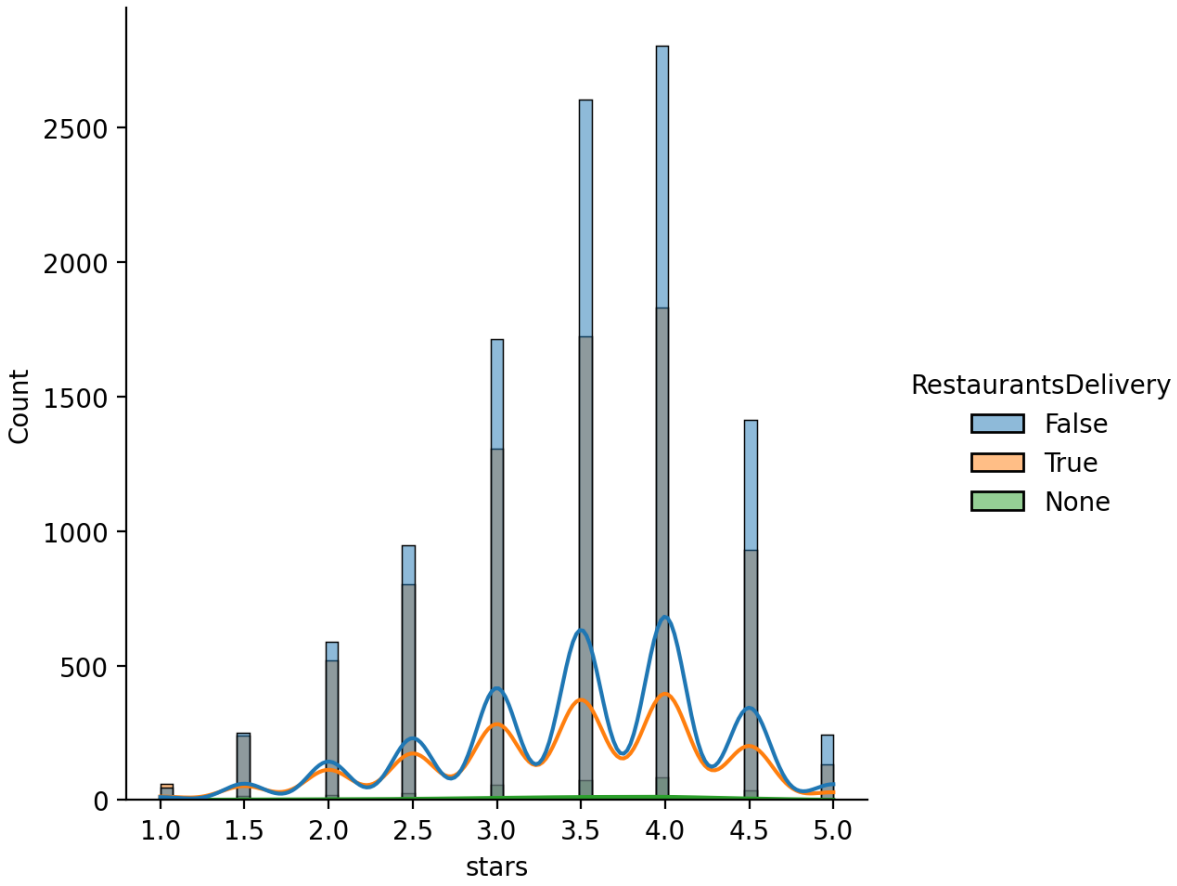
SOLUTION: numeric data trasformation

In [32]:
```
new_cols = ['RestaurantsTakeOut', 'RestaurantsDelivery', 'RestaurantsR
       'RestaurantsPriceRange2', 'RestaurantsGoodForGroups', 'HasTV',
       'GoodForKids', 'OutdoorSeating', 'RestaurantsAttire', 'Alcohol'
for col_plot in new_cols:
  plt.figure()
  sns.displot(data=train_df[["stars", col_plot]], x="stars", hue= col_
```

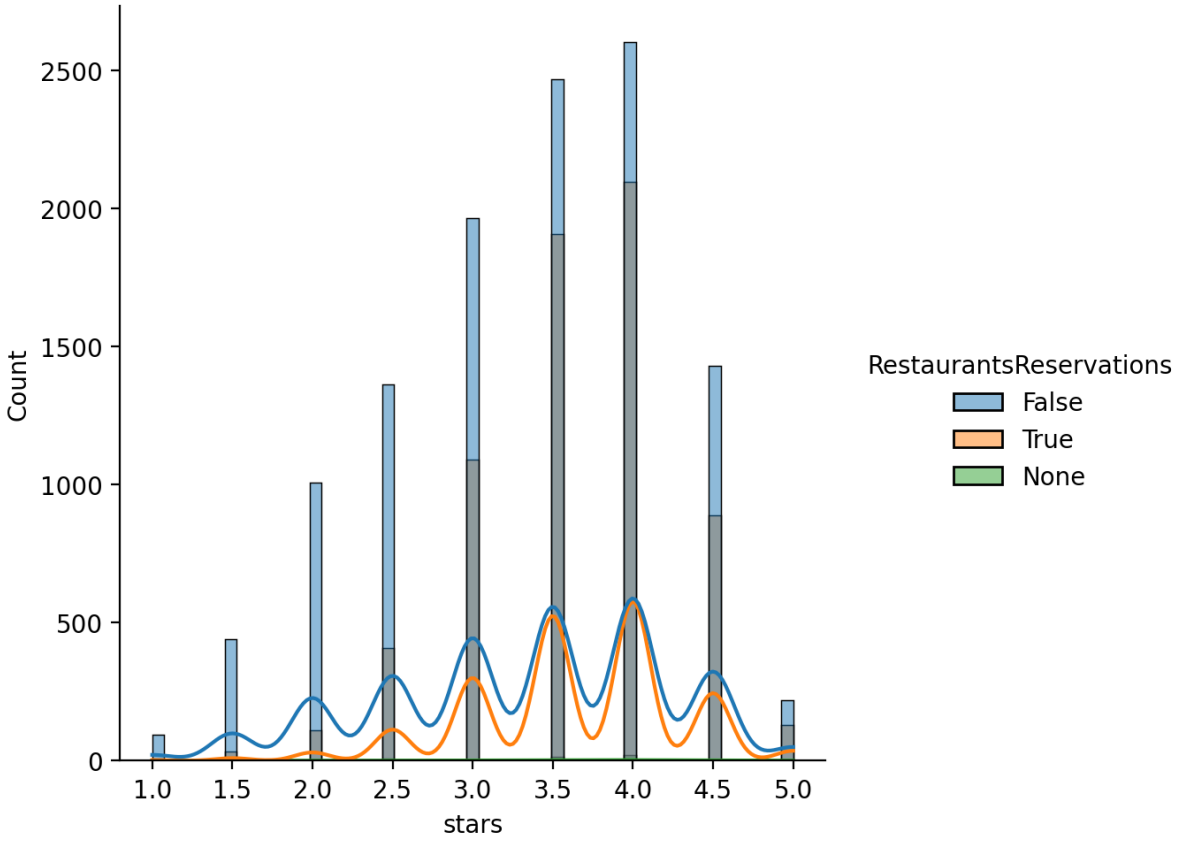
```
<Figure size 640x480 with 0 Axes>
```



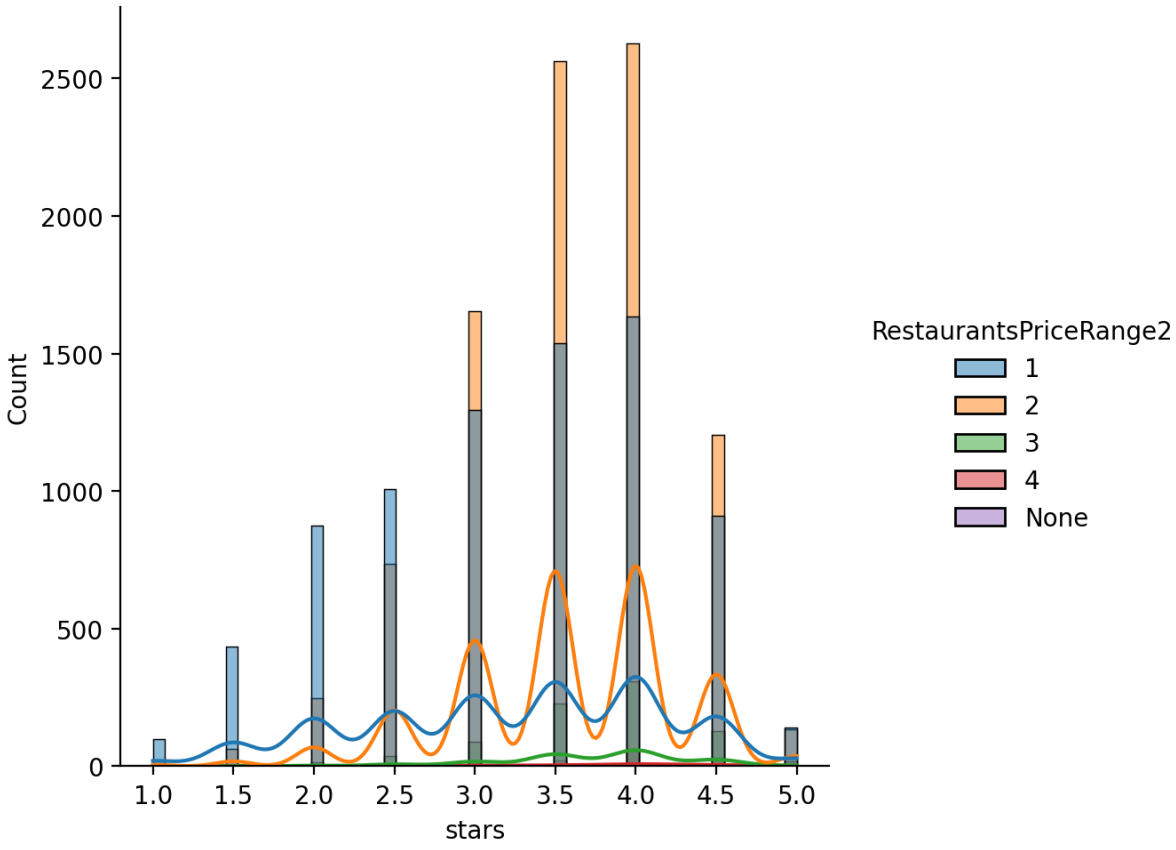

```
<Figure size 640x480 with 0 Axes>
```



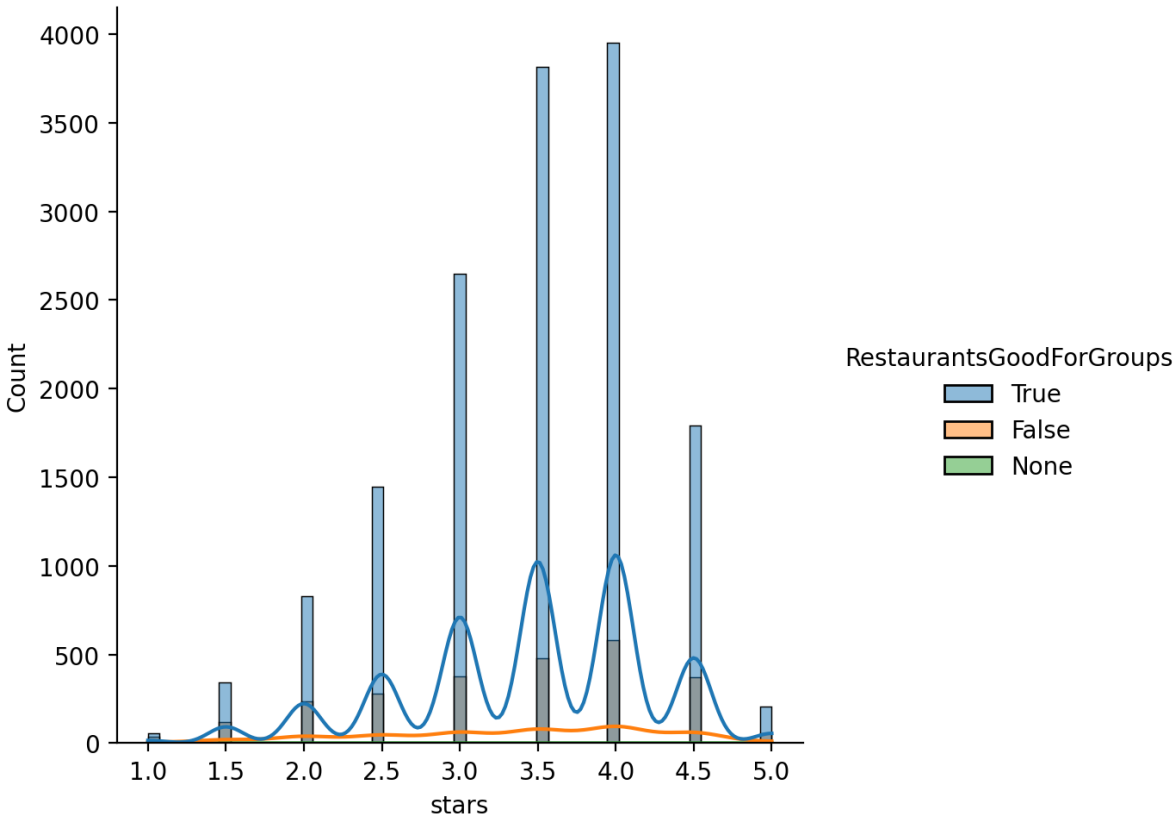

```
<Figure size 640x480 with 0 Axes>
```



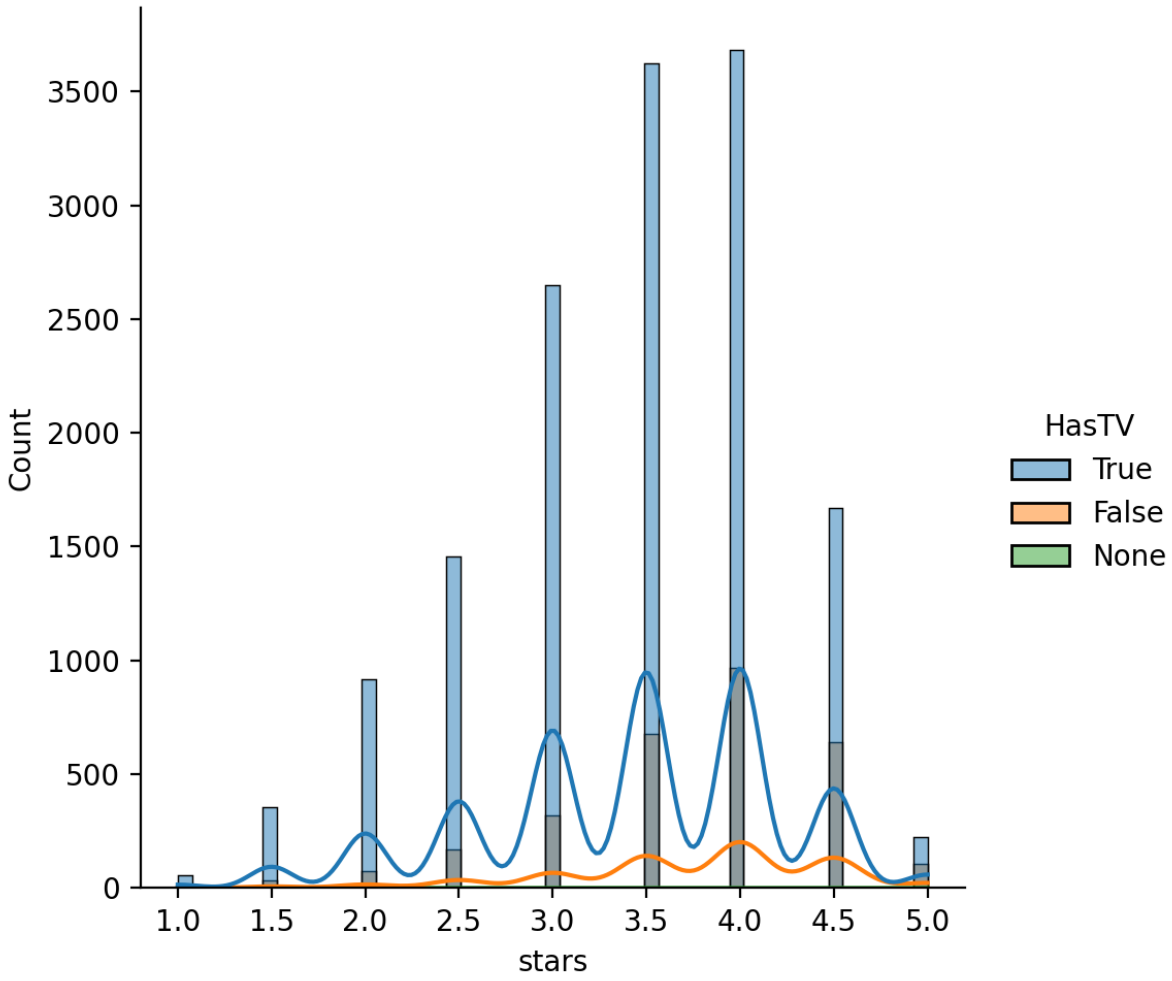

```
<Figure size 640x480 with 0 Axes>
```



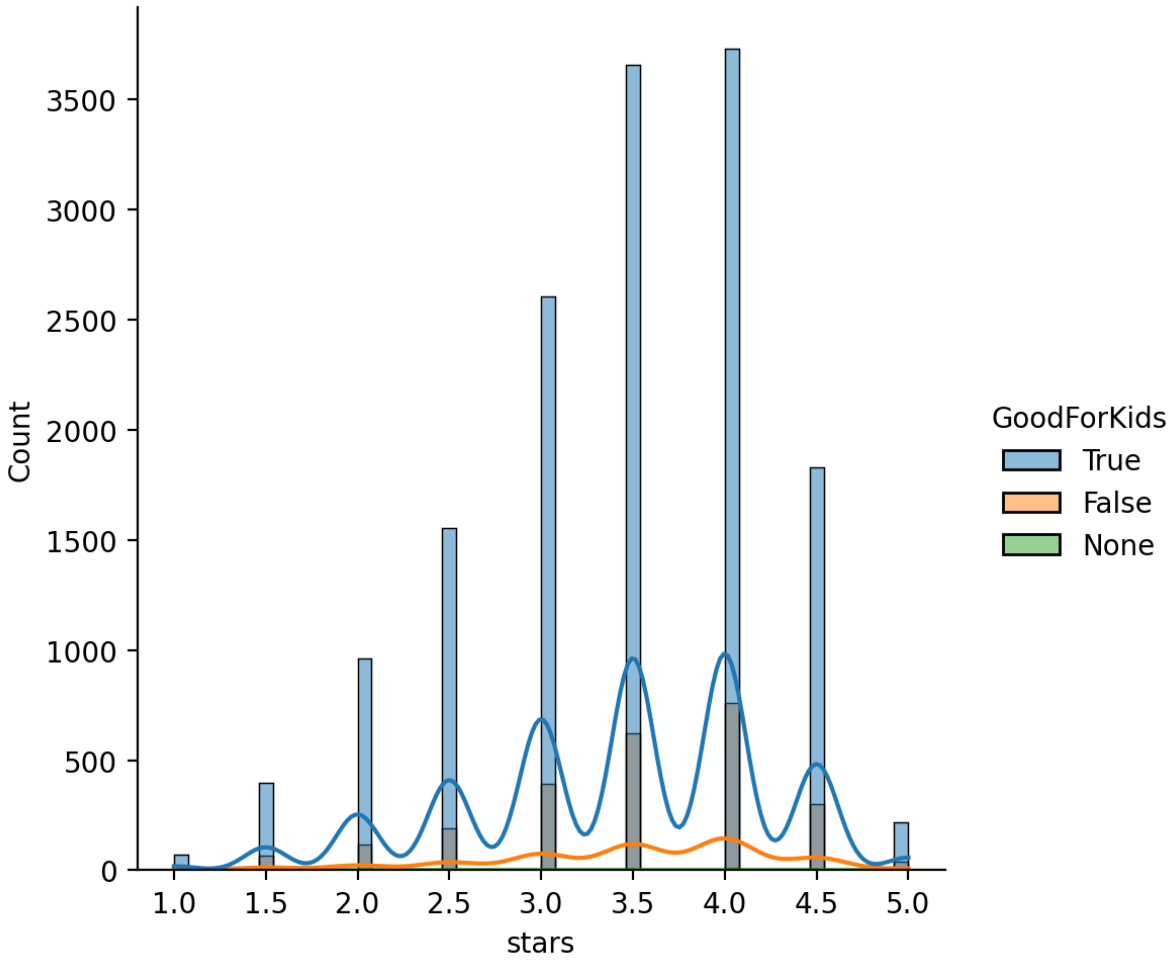
<Figure size 640x480 with 0 Axes>



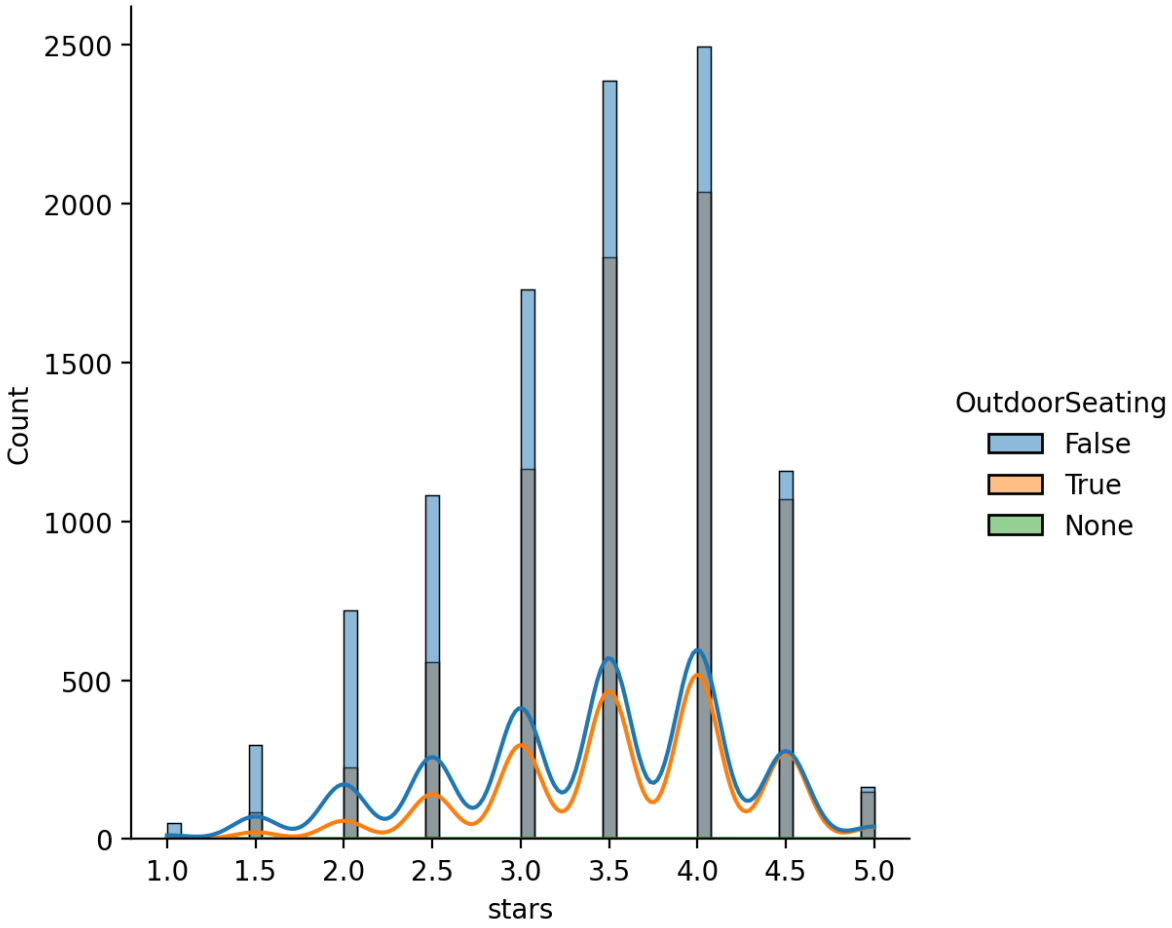
<Figure size 640x480 with 0 Axes>



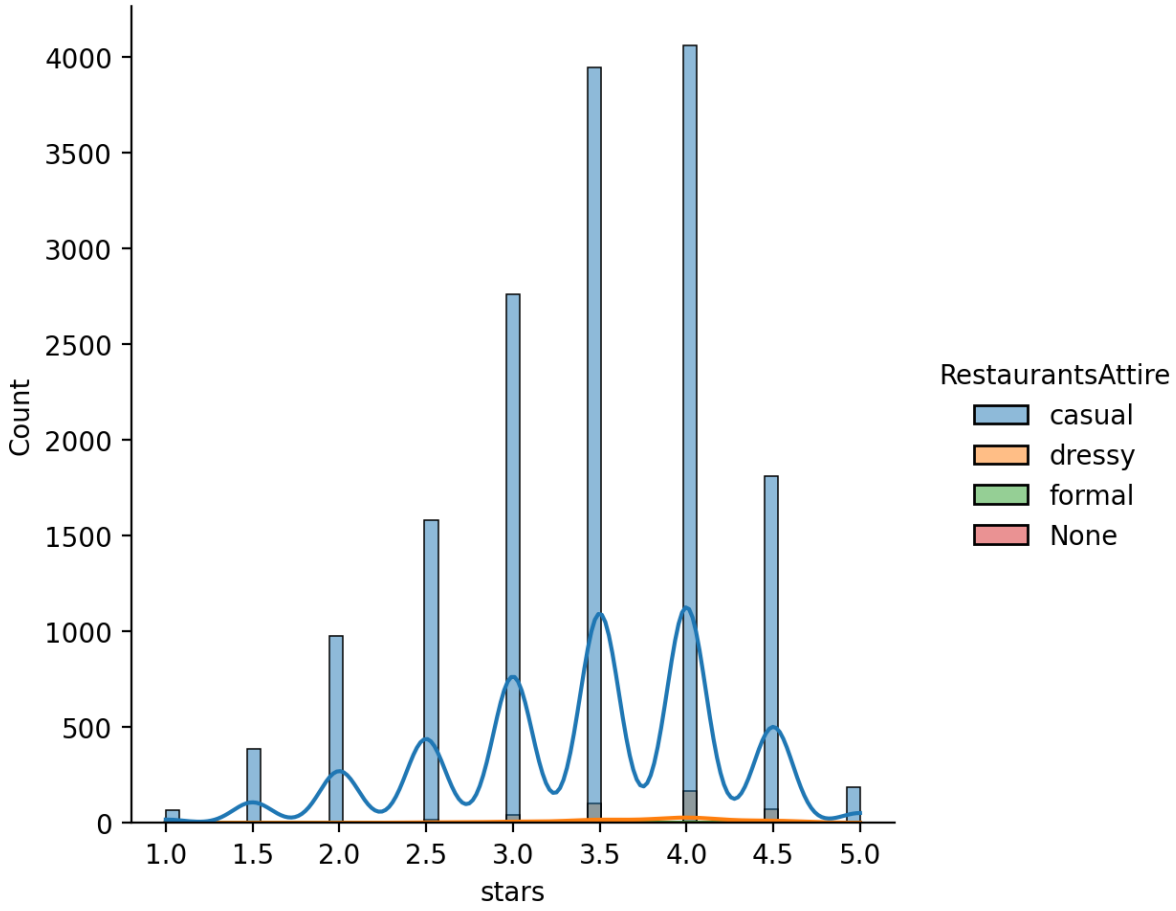

```
<Figure size 640x480 with 0 Axes>
```



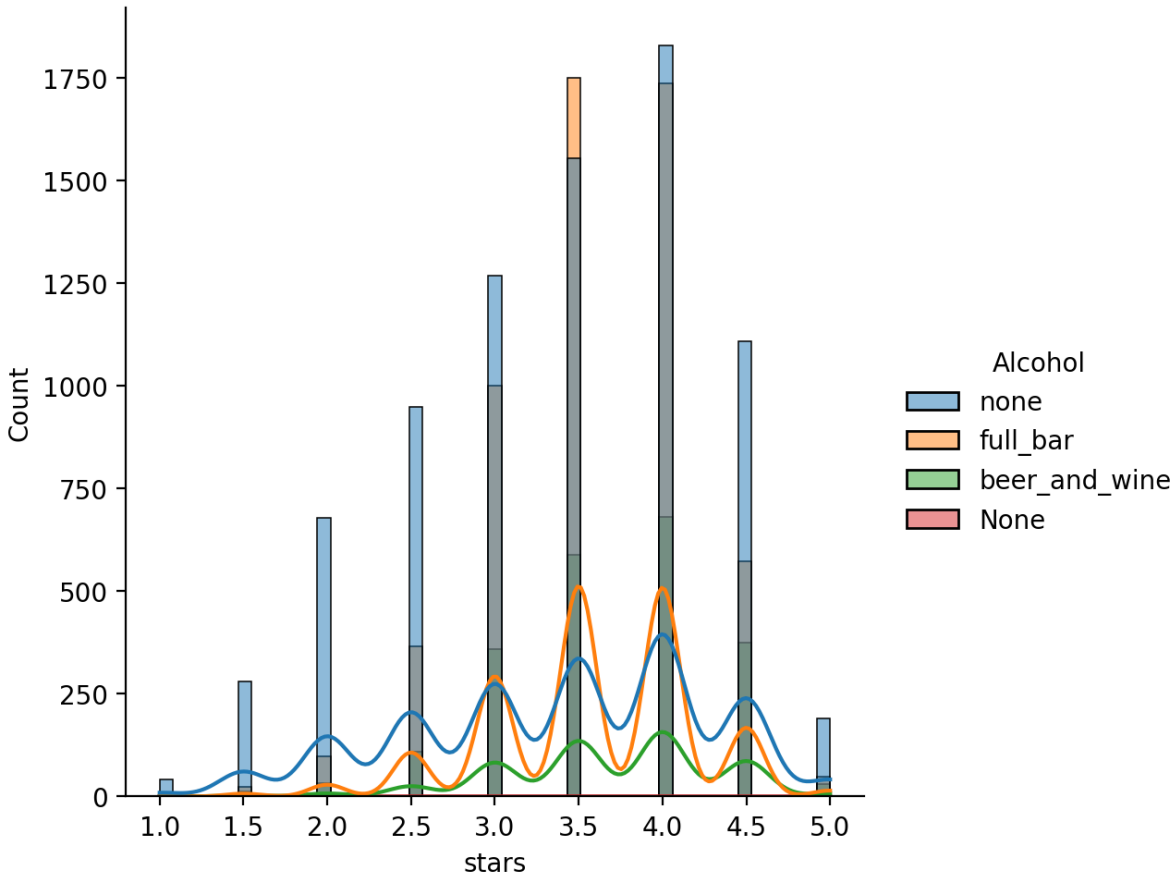

```
<Figure size 640x480 with 0 Axes>
```




```
<Figure size 640x480 with 0 Axes>
```




```
<Figure size 640x480 with 0 Axes>
```



In [33]:
```
# STEP 1
cols1 = ['RestaurantsTakeOut','RestaurantsReservations', 'RestaurantsD
train_df.loc[:,cols1] = train_df[cols1].replace(['True','False','None'
test_df.loc[:,cols1] = test_df[cols1].replace(['True','False','None'],

train_df.loc[:,'RestaurantsPriceRange2'] = train_df['RestaurantsPriceR
test_df.loc[:,'RestaurantsPriceRange2'] = test_df['RestaurantsPriceRan

train_df.loc[:,'RestaurantsAttire'] = train_df['RestaurantsAttire'].re
test_df.loc[:,'RestaurantsAttire'] = test_df['RestaurantsAttire'].repl

train_df.loc[:,'Alcohol'] = train_df['Alcohol'].replace(['none','full_
test_df.loc[:,'Alcohol'] = test_df['Alcohol'].replace(['none','full_ba
train_df.head()
```

```
/tmp/ipython-input-684228891.py:3: FutureWarning: Downcasting behavior
in `replace` is deprecated and will be removed in a future version. To
retain the old behavior, explicitly call `result.infer_objects(copy=Fal
se)`. To opt-in to the future behavior, set `pd.set_option('future.no_s
ilent_downcasting', True)`
  train_df.loc[:,cols1] = train_df[cols1].replace(['True','False','Non
e'],[1,0,np.nan])
/tmp/ipython-input-684228891.py:4: FutureWarning: Downcasting behavior
in `replace` is deprecated and will be removed in a future version. To
retain the old behavior, explicitly call `result.infer_objects(copy=Fal
se)`. To opt-in to the future behavior, set `pd.set_option('future.no_s
ilent_downcasting', True)`
  test_df.loc[:,cols1] = test_df[cols1].replace(['True','False','Non
e'],[1,0,np.nan])
/tmp/ipython-input-684228891.py:6: FutureWarning: Downcasting behavior
in `replace` is deprecated and will be removed in a future version. To
retain the old behavior, explicitly call `result.infer_objects(copy=Fal
se)`. To opt-in to the future behavior, set `pd.set_option('future.no_s
ilent_downcasting', True)`
  train_df.loc[:,'RestaurantsPriceRange2'] = train_df['RestaurantsPrice
Range2'].replace(['1','2','3','4','None'],[1,2,3,4,np.nan])
/tmp/ipython-input-684228891.py:7: FutureWarning: Downcasting behavior
in `replace` is deprecated and will be removed in a future version. To
retain the old behavior, explicitly call `result.infer_objects(copy=Fal
se)`. To opt-in to the future behavior, set `pd.set_option('future.no_s
ilent_downcasting', True)`
  test_df.loc[:,'RestaurantsPriceRange2'] = test_df['RestaurantsPriceRa
nge2'].replace(['1','2','3','4','None'],[1,2,3,4,np.nan])
/tmp/ipython-input-684228891.py:9: FutureWarning: Downcasting behavior
in `replace` is deprecated and will be removed in a future version. To
retain the old behavior, explicitly call `result.infer_objects(copy=Fal
se)`. To opt-in to the future behavior, set `pd.set_option('future.no_s
ilent_downcasting', True)`
  train_df.loc[:,'RestaurantsAttire'] = train_df['RestaurantsAttire'].r
eplace(['casual','dressy','formal','None'], [1,2,3,np.nan])
/tmp/ipython-input-684228891.py:10: FutureWarning: Downcasting behavior
in `replace` is deprecated and will be removed in a future version. To
```

```
retain the old behavior, explicitly call `result.infer_objects(copy=Fal
se)`. To opt-in to the future behavior, set `pd.set_option('future.no_s
ilent_downcasting', True)`
  test_df.loc[:,'RestaurantsAttire'] = test_df['RestaurantsAttire'].rep
lace(['casual','dressy','formal','None'], [1,2,3,np.nan])
/tmp/ipython-input-684228891.py:12: FutureWarning: Downcasting behavior
in `replace` is deprecated and will be removed in a future version. To
retain the old behavior, explicitly call `result.infer_objects(copy=Fal
se)`. To opt-in to the future behavior, set `pd.set_option('future.no_s
ilent_downcasting', True)`
  train_df.loc[:,'Alcohol'] = train_df['Alcohol'].replace(['none','full
_bar','beer_and_wine','None'], [1,2,3,np.nan])
/tmp/ipython-input-684228891.py:13: FutureWarning: Downcasting behavior
in `replace` is deprecated and will be removed in a future version. To
retain the old behavior, explicitly call `result.infer_objects(copy=Fal
se)`. To opt-in to the future behavior, set `pd.set_option('future.no_s
ilent_downcasting', True)`
  test_df.loc[:,'Alcohol'] = test_df['Alcohol'].replace(['none','full_b
ar','beer_and_wine','None'], [1,2,3,np.nan])
```

Out[33]:

	business_id	name	address	city	state	postal_c
64514	fiVH82y41YtGdndjT_wpzQ	Popeyes Louisiana Kitchen	9910 W Flamingo Rd	Las Vegas	NV	89
201206	ku3b6kxAmb05-rLblyQEnw	Tuk Tuk Thai	5763 Signal Hill Centre SW	Calgary	AB	T3H
202245	d_L-rfS1vT3JMzgCUGtiow	Border Grill	3950 S Las Vegas Blvd	Las Vegas	NV	8
128024	yRBE98DheftGKDwZG39P-w	Latazza Cafe	2200 Yonge Street	Toronto	ON	M4S
88564	d4P7boUqiA2pR59jOlzYLA	Assembly Chef's Hall	111 Richmond Street W	Toronto	ON	M5H

5 rows × 25 columns

SOLUTION: data imputation

In [34]:

```
all_cols = ['latitude','longitude','log10_review_count',
       'RestaurantsTakeOut', 'RestaurantsDelivery', 'RestaurantsReserv
       'RestaurantsPriceRange2', 'RestaurantsGoodForGroups', 'HasTV',
       'GoodForKids', 'OutdoorSeating', 'RestaurantsAttire', 'Alcohol'
```

```
# Now these will work correctly since log10_review_count exists in the
X_train = train_df.loc[:,all_cols].values
X_test = test_df.loc[:,all_cols].values
Y_train = train_df['stars'].values  # Add this line that was missing
Y_test = test_df['stars'].values    # Add this line that was missing

print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"Y_train shape: {Y_train.shape}")
print(f"Y_test shape: {Y_test.shape}")
```

```
X_train shape: (21982, 13)
X_test shape: (21983, 13)
Y_train shape: (21982,)
Y_test shape: (21983,)
```

In [36]:
```
from sklearn.impute import SimpleImputer
v_k = [1,5,10,25,50,100,250,500]
v_weights = ['uniform', 'distance']

param_grid = {
    'imp__strategy' : [ 'most_frequent'], #['mean', 'most_frequent', '
    'kNN__n_neighbors': v_k,
    'kNN__weights':v_weights,
}
pipe = Pipeline([('imp', SimpleImputer(missing_values=np.nan)), ('scal
grid_pipe = GridSearchCV(pipe, param_grid, cv=5)
grid_pipe.fit(X_train, Y_train)
print("Score with the training data R^2={0:.4f}".format(grid_pipe.scor
print("Score with the test data R^2={0:.4f}".format(grid_pipe.score(X_
print("Hyperparameters choosen with cross-validation")
print(grid_pipe.best_params_)

global_results['SimpleImputer  +knn + scale + geo + review + attribute
```

```
Score with the training data R^2=0.2163
Score with the test data R^2=0.1552
Hyperparameters choosen with cross-validation
{'imp__strategy': 'most_frequent', 'kNN__n_neighbors': 25, 'kNN__weight
s': 'uniform'}
```

In [37]:
```
from sklearn.impute import KNNImputer
from sklearn.impute import SimpleImputer

imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')

imp.fit(X_train)
X_train_imp = imp.transform(X_train)
X_test_imp = imp.transform(X_test)
```

SOLUTION OHE

In [38]:
```
# Codify as ohe
from sklearn.preprocessing import OneHotEncoder
onehot = OneHotEncoder()
onehot.fit(np.vstack((X_train_imp[:,3:],X_test_imp[:,3:])))
X_train_cat = onehot.transform(X_train_imp[:,3:]).toarray()
X_test_cat = onehot.transform(X_test_imp[:,3:]).toarray()
print(onehot.categories_)

X_train = np.hstack((X_train_imp[:,:3], X_train_cat))
X_test = np.hstack((X_test_imp[:,:3], X_test_cat))
```

```
[array([0.0, 1.0], dtype=object), array([0.0, 1.0], dtype=object), arra
y([0.0, 1.0], dtype=object), array([1.0, 2.0, 3.0, 4.0], dtype=object),
array([0.0, 1.0], dtype=object), array([0.0, 1.0], dtype=object), arra
y([0.0, 1.0], dtype=object), array([0.0, 1.0], dtype=object), array([1.
0, 2.0, 3.0], dtype=object), array([1.0, 2.0, 3.0], dtype=object)]
```

In [39]:
```
# Pipeline definition. List with all the stages of the pipeline
knn_pipe = Pipeline([('scaler', StandardScaler()), ('regressor',KNeigh
# Dictionary with hyperparameters for all the stages of the pipeline,
# that will be tuned with crossvalidation
params_pipe = {'regressor__n_neighbors':v_k,
               'regressor__weights':v_weights,
                'scaler':[StandardScaler(),'passthrough']}
# 'passthrough' means skip this stage.
# This way we can estimate with crossvalidation if scaling was a good
# set up the grid to optimize the hyperparameters and train the pipeli
grid_pipe = GridSearchCV(knn_pipe, param_grid= params_pipe, cv=5)
grid_pipe.fit(X_train, Y_train)
print("Score with the training data R^2={0:.4f}".format(grid_pipe.scor
print("Score with the test data R^2={0:.4f}".format(grid_pipe.score(X_
print("Hyperparameters choosen with cross-validation")
print(grid_pipe.best_params_)

global_results['Imputer +knn + scale + geo + log review + attributes']
```

```
Score with the training data R^2=0.9996
Score with the test data R^2=0.1584
Hyperparameters choosen with cross-validation
{'regressor__n_neighbors': 100, 'regressor__weights': 'distance', 'scal
er': 'passthrough'}
```

In [40]:
```
global_results
```

Out[40]:

```
{'linear_regression': {'weights': array([0.00058963]),
  'intercept': np.float64(3.395217653590921),
  'mse': 0.6843743355525993,
  'r2': 0.019667929028433506},
 'knn_regressor': {'best_k ': 50,
  'mse': 0.6644327480325707,
  'r2': 0.048233257645242666},
 'knn_regressor_with_review_count': {'best_k ': 250,
  'mse': 0.6355372988218169,
  'r2': 0.08962454614754933},
 'knn_regressor_log_review_counts': {'best_k': 100,
  'mse': 0.6293704009916754,
  'r2': 0.0984583194310128},
 'knn + scale geo + log review': 0.0984583194310128,
 'K-NN Clean Data (GMM)': {'R2': 0.10215167032061201,
  'n_components': 10,
  'data_retained': 0.8999636065872078},
 'SimpleImputer  +knn + scale + geo + review + attributes': 0.15517434535173902,
 'Imputer +knn + scale + geo + log review + attributes': 0.1583939376155189}
```