Milestone1_4. Exploratory Data Analysis

Identify and engineer influential Features

Team #30

In [1]:
```
import pandas as pd
import numpy as np
import altair as alt
from vega_datasets import data
import geopandas as gpd
import json
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
from scipy.stats import ttest_ind
from scipy.stats import f_oneway
from scipy.stats import mannwhitneyu
from scipy import linalg
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import statsmodels.api as sm
from statsmodels.stats.multicomp import pairwise_tukeyhsd
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R) SSE4.2) enabled only processors has been deprecated. Intel oneAPI Mat
h Kernel Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R) SSE4.2) enabled only processors has been deprecated. Intel oneAPI Mat
h Kernel Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(R) AVX) instructions.
```

In [2]:
```
# Set the maximum number of displayed columns to a higher value
pd.set_option('display.max_columns', None)

# Disable the max rows limit
alt.data_transformers.disable_max_rows()
```

Out[2]:
```
DataTransformerRegistry.enable('default')
```

In [3]:
```
# Yelp color palette
# https://www.behance.net/gallery/26422079/Yelp-Rebrand-Concept
# https://www.flerlagetwins.com/2021/06/datafam-colors-color-palette.html

# colors = ['#D84465','#B04C75','#B4ACA6','#FF6F4C','#F15060','#B04C75','#8F648C','#746CAF','#96B6E5','#3Ad4A4']
colors = ['#D84465','#B04C75','#3369dd', '#B4ACA6', '#0097a7','#FF6F4C','#F15060', '#B04C75']
```

Context

Our analysis of Yelp restaurant data focuses on 4 prominent cuisines: Chinese, Japanese, Italian, and Mexican. These cuisines are among the top cuisines in the U.S. that are not of American origin.

These cuisines have gained widespread popularity and acceptance in the United States and are often considered staples of international cuisine. Each brings a unique set of flavors, cooking techniques, and cultural influences, contributing to the diversity of the American culinary landscape.

We compare the ratings of each of the four cuisines and aim to understand the factors that contribute to the disparities in ratings across these culinary categories.

4.1. Check Data Coverage

Insights

the Yelp data is evidently not randomly sampled at the zip code level, whereas our secondary data source - demographics - is structured at the zip code level. Therefore, our analysis is likely biased due to the data limitations, and our findings may not be applicable to the entire country.

In [4]:
```
master3 = pd.read_csv('data/master3.csv')
print(master3.shape)
```

```
(49369, 51)
```

In [5]:
```
master4 = pd.read_csv('data/master4.csv')
print(master4.shape)
print(master4.dtypes)
```

```
(12005, 64)
business_id     object
name            object
city            object
state           object
zip_code       float64
                 ...
gluten_free      int64
fast_food        int64
breakfast        int64
nightlife        int64
ctgy_count       int64
Length: 64, dtype: object
```

In [6]:
```
master3.zip_code = master3.zip_code.astype(int)
master4.zip_code = master4.zip_code.astype(int)
```

In [7]:
```
print(f'The subset of Yelp academic data, focusing solely on restaurants, includes {master3.zip_code.nunique()} unique zip codes.')
print(f'The subset of Yelp academic data, focusing solely on four cuisines, includes {master4.zip_code.nunique()} unique zip codes.')
```

```
The subset of Yelp academic data, focusing solely on restaurants, includes 772 unique zip codes.
The subset of Yelp academic data, focusing solely on four cuisines, includes 688 unique zip codes.
```

In [8]:
```
# zip county mapping: https://simplemaps.com/data/us-zips

zip_county = pd.read_csv('data/uszips.csv')

zip_county.dtypes
```

Out[8]:
```
zip                   int64
lat                 float64
lng                 float64
city                 object
state_id             object
state_name           object
zcta                   bool
parent_zcta         float64
population          float64
density             float64
county_fips           int64
county_name          object
county_weights       object
county_names_all     object
county_fips_all      object
imprecise              bool
military               bool
timezone             object
dtype: object
```

In [9]:
```
print(f'There are {zip_county.zip.nunique()} unique zip codes in the U.S.')
```

```
There are 33788 unique zip codes in the U.S.
```

In [10]:
```
pcnt = round(master3.zip_code.nunique() / zip_county.zip.nunique() * 100, 1)
print(f'''The Yelp Academic data is probably a limited subset of actual data,
given that restaurant information is available for only {pcnt}% of U.S. zip codes.''')
```

```
The Yelp Academic data is probably a limited subset of actual data,
given that restaurant information is available for only 2.3% of U.S. zip codes.
```

In [11]:
```
# Yelp academic data only covers a few states

master3.groupby('state').business_id.count().sort_values(ascending=False)
```

Out[11]:
```
state
PA    12443
FL     8698
TN     4317
MO     4205
IN     4135
LA     3610
NJ     3306
AZ     2648
NV     1641
ID     1296
CA     1129
IL      981
DE      957
CO        1
MT        1
NC        1
Name: business_id, dtype: int64
```

In [12]:
```
a = master4.zip_code.unique()
b = zip_county.zip.unique()
print(len(a))
print(len(b))
print(len(set(a).intersection(set(b))))
```

```
688
33788
688
```

In [13]:
```
df1 = (master3.merge(zip_county[['zip','county_fips']].rename(columns={'zip':'zip_code'}),
                         how='left', on='zip_code'))
df2 = (master4.merge(zip_county[['zip','county_fips']].rename(columns={'zip':'zip_code'}),
                         how='left', on='zip_code'))
```

In [14]:
```
restaurant_count_byCounty = (zip_county[['state_name','county_name','county_fips']].drop_duplicates()
            .merge(df1.groupby('county_fips')['business_id'].count().reset_index()
                   .rename(columns={'business_id':'restaurant_count'})
                   , how='left', on='county_fips'))
```

In [15]:
```
restaurant_count_byCounty.isna().sum()
```

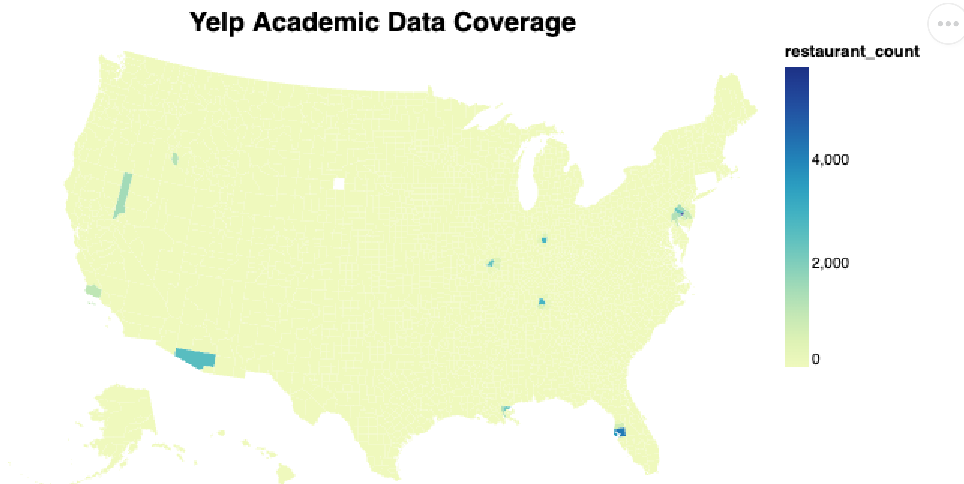
Out[15]:
```
state_name             0
county_name            0
county_fips            0
restaurant_count    3143
dtype: int64
```

In [16]:
```
restaurant_count_byCounty.fillna(0, inplace=True)
```

In [17]:
```
counties = alt.topo_feature(data.us_10m.url, 'counties')


alt.Chart(counties).mark_geoshape().encode(
    color='restaurant_count:Q',
    tooltip=['state_name:N','county_name:N','restaurant_count:Q']
```

```
).transform_lookup(
    lookup='id',
    from_=alt.LookupData(restaurant_count_byCounty, 'county_fips', ['restaurant_count','county_name','state_name'])
).project(
    type='albersUsa'
).properties(
    title={
        'text': 'Yelp Academic Data Coverage',
        'fontSize': 18  # Adjust the font size as needed
    },
    width=500,
    height=300
)
```

Out[17]:



In [18]:

```
# Restaurant count by state for subset of Yelp data

master4.groupby('state').business_id.count().sort_values(ascending=False)
```

Out[18]:

```
state
PA    3012
FL    1907
MO    1016
TN     989
IN     976
NJ     972
AZ     887
LA     640
NV     469
CA     333
ID     303
IL     262
DE     239
Name: business_id, dtype: int64
```

4.2. Variables

explain

We categorized features into numerical and categorical, which facilitates the application of different methods in exploring the relationship between restaurant stars and numerical versus categorical features.

In [19]:
```
master4.columns
```

Out[19]:
```
Index(['business_id', 'name', 'city', 'state', 'zip_code', 'latitude',
       'longitude', 'stars', 'review_count', 'is_open', 'categories',
       'RestaurantsDelivery', 'OutdoorSeating', 'BusinessAcceptsCreditCards',
       'BikeParking', 'RestaurantsTakeOut', 'Alcohol', 'Caters',
       'RestaurantsReservations', 'GoodForKids', 'RestaurantsGoodForGroups',
       'HasTV', 'NoiseLevel', 'RestaurantsPriceRange', 'expensive',
       'free_WiFi', 'attire_dressy', 'noise_loud', 'median_household_income',
       'population', 'household_cnt', 'median_age',
       'population_hispanic_latino', 'population_white', 'population_asian',
       'bachelors_pcnt', 'education_pcnt', 'restaurant_count',
       'population_perRestaurant', 'household_perRestaurant',
       'hispanic_latino_pcnt', 'white_pcnt', 'asian_pcnt',
       'useful_review_count', 'funny_review_count', 'cool_review_count',
       'review_sentiment_score', 'avg_tip_compliment', 'tip_sentiment_score',
       'tip_count', 'has_tip', 'Chinese', 'Japanese', 'Italian', 'Mexican',
       'MECE_check', 'cuisine', 'plant_based', 'seafood', 'gluten_free',
       'fast_food', 'breakfast', 'nightlife', 'ctgy_count'],
      dtype='object')
```

In [20]:
```
numerical_vars = ['median_household_income','household_cnt',
                  'median_age', 'bachelors_pcnt', 'education_pcnt',
                  'hispanic_latino_pcnt', 'white_pcnt', 'asian_pcnt',
                  'population_perRestaurant', 'household_perRestaurant',
                  'review_count', 'review_sentiment_score', 'useful_review_count', 'funny_review_count', 'cool_review_count',
                  'avg_tip_compliment', 'tip_sentiment_score', 'tip_count','ctgy_count']

categorical_vars_binary = ['RestaurantsDelivery', 'OutdoorSeating', 'BusinessAcceptsCreditCards',
                    'BikeParking', 'RestaurantsTakeOut', 'Alcohol', 'Caters',
                    'RestaurantsReservations', 'GoodForKids',
                    'RestaurantsGoodForGroups', 'HasTV',
                    'free_WiFi', 'noise_loud', 'attire_dressy','expensive',
                    'plant_based', 'seafood', 'gluten_free', 'fast_food', 'breakfast', 'nightlife']

categorical_vars_nominal = ['cuisine']


nonUsed_vars = ['business_id', 'name', 'city', 'state', 'zip_code', 'latitude',
                'longitude', 'stars','is_open', 'categories',
                'NoiseLevel',  'RestaurantsPriceRange',
                'restaurant_count', 'population',  'has_tip',
                'Chinese', 'Japanese','Italian', 'Mexican', 'MECE_check',
                'population_hispanic_latino', 'population_white', 'population_asian']
```

```
assert (len(numerical_vars) + len(categorical_vars_binary)
        + len(categorical_vars_nominal) + len(nonUsed_vars)
        == master4.shape[1])
```

In [21]:
```
master4[categorical_vars_binary].nunique()
```

Out[21]:
```
RestaurantsDelivery           2
OutdoorSeating                2
BusinessAcceptsCreditCards    2
BikeParking                   2
RestaurantsTakeOut            2
Alcohol                       2
Caters                        2
RestaurantsReservations       2
GoodForKids                   2
RestaurantsGoodForGroups      2
HasTV                         2
free_WiFi                     2
noise_loud                    2
attire_dressy                 2
expensive                     1
plant_based                   2
seafood                       2
gluten_free                   2
fast_food                     2
breakfast                     2
nightlife                     2
dtype: int64
```

In [22]:
```
categorical_vars_binary.remove('attire_dressy')
categorical_vars_binary.remove('expensive')
```

In [23]:
```
nonUsed_vars.append('attire_dressy')
nonUsed_vars.append('expensive')

assert (len(numerical_vars) + len(categorical_vars_binary)
        + len(categorical_vars_nominal) + len(nonUsed_vars)
        == master4.shape[1])
```

4.3. Restaurant Ratings by Cuisine

insights

Japanese cuisine generally receives higher ratings compared to the other three cuisines, with the highest mean and median stars among all categories. On the contrary, Chinese cuisine has the lowest average ratings among the four, with a statistically significant difference observed.

In [24]:
```
master4.groupby('cuisine').stars.mean().sort_values()
```

Out[24]:
```
cuisine
Chinese     3.343501
Italian     3.505867
Mexican     3.511181
Japanese    3.761137
Name: stars, dtype: float64
```

In [25]:
```
master4.groupby('cuisine')['stars'].median().sort_values()
```

Out[25]:
```
cuisine
Chinese     3.5
Italian     3.5
Mexican     3.5
Japanese    4.0
Name: stars, dtype: float64
```

In [26]:
```
# Perform one-way ANOVA
f_statistic, p_value = f_oneway(
    master4['stars'][master4['cuisine'] == 'Chinese'],
    master4['stars'][master4['cuisine'] == 'Japanese'],
    master4['stars'][master4['cuisine'] == 'Italian'],
    master4['stars'][master4['cuisine'] == 'Mexican']
)

# Print the results
print("F-statistic:", f_statistic)
print("P-value:", p_value)

# Interpret the results
alpha = 0.05  # Set significance level
if p_value < alpha:
    print("There is a statistically significant difference in average ratings among the cuisines.")
else:
    print("There is no statistically significant difference in average ratings among the cuisines.")
```

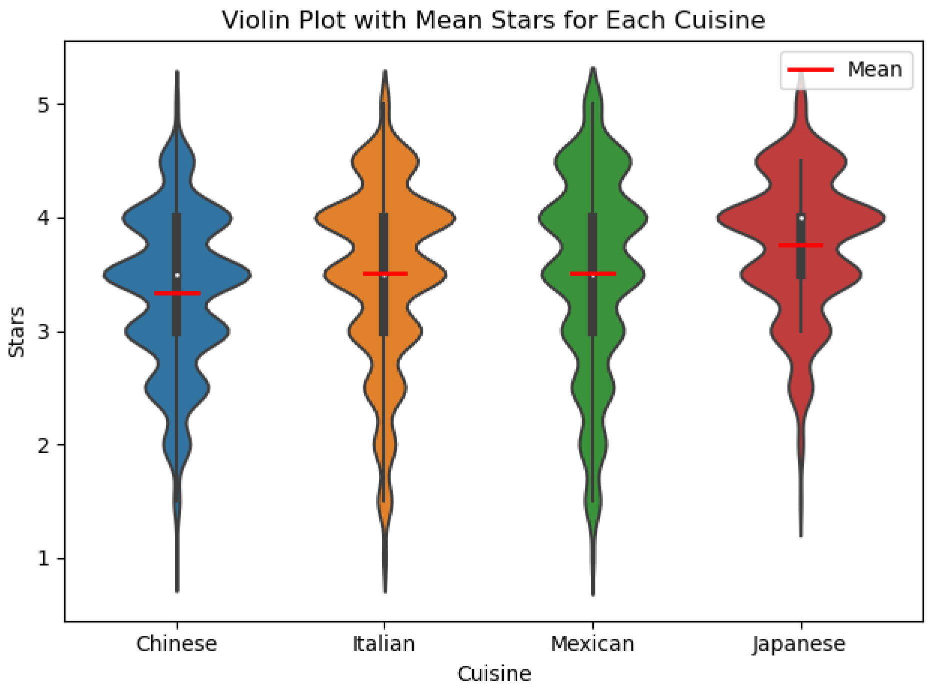
```
F-statistic: 69.2793149191495
P-value: 2.0610420104320917e-44
There is a statistically significant difference in average ratings among the cuisines.
```

In [27]:
```
# We perform post hoc tests following an ANOVA to compare
# pairwise differences in average ratings among the four cuisines.
# One commonly used post hoc test is the Tukey-Kramer test.
# This test will help us determine which specific pairs of cuisines
# have statistically significant differences in their average ratings.

# Perform one-way ANOVA
f_statistic, p_value = f_oneway(
    master4['stars'][master4['cuisine'] == 'Chinese'],
    master4['stars'][master4['cuisine'] == 'Japanese'],
    master4['stars'][master4['cuisine'] == 'Italian'],
    master4['stars'][master4['cuisine'] == 'Mexican']
)

# Perform Tukey-Kramer post hoc test
tukey_results = pairwise_tukeyhsd(master4['stars'], master4['cuisine'], alpha=0.05)
```

```
# Print the Tukey-Kramer results
print(tukey_results)
```

```
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=======================================================
 group1   group2  meandiff p-adj  lower   upper  reject
-------------------------------------------------------
 Chinese  Italian   0.1624    0.0 0.1099  0.2148   True
 Chinese Japanese   0.4176    0.0 0.3424  0.4928   True
 Chinese  Mexican   0.1677    0.0 0.1156  0.2198   True
 Italian Japanese   0.2553    0.0 0.1859  0.3246   True
 Italian  Mexican   0.0053 0.9891 -0.038  0.0486  False
Japanese  Mexican    -0.25    0.0 -0.319 -0.1809   True
-------------------------------------------------------
```

In [28]:
```
# Calculate mean stars for each cuisine
mean_stars = master4.groupby('cuisine')['stars'].mean().sort_values()

# Cuisine order
cuisineOrder = master4.groupby('cuisine').stars.mean().sort_values().index.tolist()

# Create a violin plot
sns.violinplot(x='cuisine', y='stars', data=master4, order=cuisineOrder)


# Plot mean stars as short lines
for i, cuisine in enumerate(mean_stars.index):
    plt.plot([i - 0.1, i + 0.1], [mean_stars[cuisine], mean_stars[cuisine]],
             color='red', linewidth=2, zorder=3, label='Mean' if i == 0 else "")


plt.legend()
plt.xlabel('Cuisine')
plt.ylabel('Stars')
plt.title('Violin Plot with Mean Stars for Each Cuisine')
plt.xticks(ticks=range(len(mean_stars.index)), labels=mean_stars.index)
plt.tight_layout()
plt.show()
```



4.4. Examine Relationships Between Stars and Numerical Variables

4.4.1. Correlation Matrix

insights

Upon examining the correlation matrix and heatmap, we observed a strong correlation between restaurant ratings and review sentiment, tip sentiment, as well as associated features related to reviews and tips. In contrast, there is almost no correlation between restaurant ratings and demographic factors such as education, ethnicity, age, income, population density, etc.

Additionally, we observed significant correlations between certain features. For instance, the useful review count and cool review count exhibit a high positive correlation, and similarly, the review count and cool review count also display a strong positive correlation. Moreover, the percentage of Hispanic Latino population is negatively correlated with the percentage of bachelor's degree.

It is not surprising that restaurant ratings correlate strongly with review sentiment, tip sentiment, review count, and associated features. After all, the rating serves as a reflection of the overall customer sentiment and the restaurant's popularity. However, the lack of correlation with demographic factors was unexpected.

In [29]:
```
correlation_matrix = master4[['stars'] + numerical_vars].corr()

corr_order = correlation_matrix.abs().sort_values(by='stars', ascending=False).index.tolist()
correlation_matrix = correlation_matrix.sort_values(by='stars', ascending=False)[corr_order]
correlation_matrix
```

Out[29]:

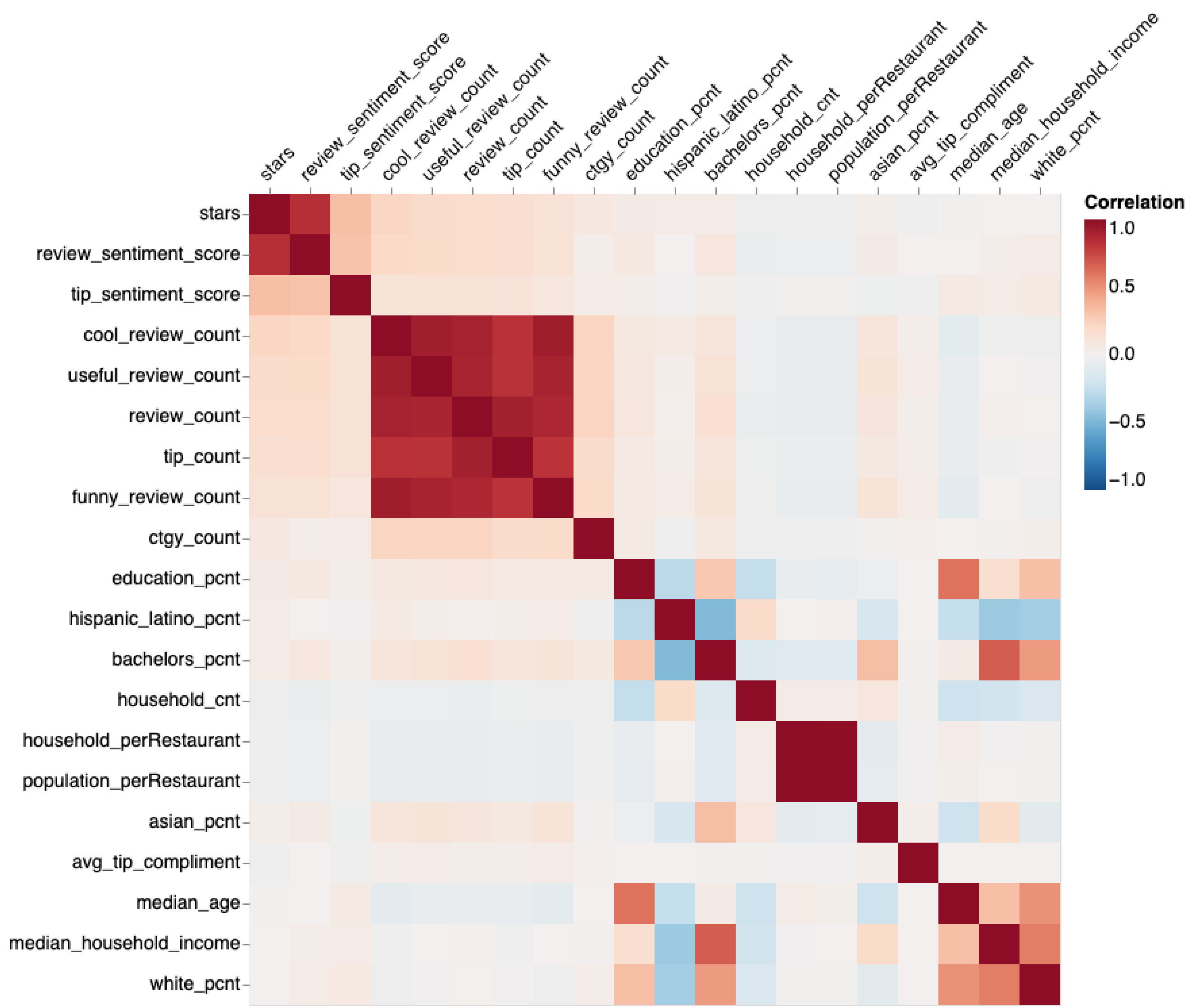
	stars	review_sentiment_score	tip_sentiment_score	cool_review_count	useful_review_count	review_count	tip_count	funny_review_c
stars	1.000000	0.836599	0.326402	0.216546	0.185717	0.175694	0.155739	0.125
review_sentiment_score	0.836599	1.000000	0.310836	0.204683	0.192749	0.180372	0.168935	0.137
tip_sentiment_score	0.326402	0.310836	1.000000	0.118354	0.125535	0.132007	0.126845	0.086
cool_review_count	0.216546	0.204683	0.118354	1.000000	0.918731	0.896107	0.824232	0.92
useful_review_count	0.185717	0.192749	0.125535	0.918731	1.000000	0.889395	0.821745	0.891
review_count	0.175694	0.180372	0.132007	0.896107	0.889395	1.000000	0.909416	0.870
tip_count	0.155739	0.168935	0.126845	0.824232	0.821745	0.909416	1.000000	0.814
funny_review_count	0.125708	0.137107	0.086478	0.921110	0.891684	0.870785	0.814187	1.000
ctgy_count	0.078273	0.029353	0.042468	0.227301	0.223208	0.224051	0.185594	0.195
education_pcnt	0.046246	0.068781	0.033549	0.072714	0.084316	0.091492	0.059723	0.058
hispanic_latino_pcnt	0.037845	0.001812	-0.015131	0.055689	0.028188	0.015899	0.034428	0.040
bachelors_pcnt	0.037020	0.090948	0.024609	0.108535	0.127813	0.152421	0.105763	0.114
asian_pcnt	0.020155	0.046164	-0.029898	0.107391	0.113702	0.099364	0.074963	0.119
median_household_income	0.003697	0.032793	0.043097	-0.018677	0.001048	0.012289	-0.017546	-0.001
white_pcnt	0.002492	0.038254	0.064613	-0.033783	-0.013865	0.003305	-0.015322	-0.03
median_age	-0.004898	0.001279	0.061384	-0.087243	-0.065048	-0.061831	-0.061716	-0.084
avg_tip_compliment	-0.019360	-0.001857	-0.023885	0.029394	0.035830	0.021233	0.027036	0.036
population_perRestaurant	-0.020454	-0.037300	-0.003631	-0.058084	-0.061531	-0.060908	-0.047454	-0.054
household_perRestaurant	-0.021153	-0.037418	-0.003204	-0.059779	-0.062656	-0.062508	-0.048641	-0.056
household_cnt	-0.031107	-0.049486	-0.025324	-0.036869	-0.042999	-0.048754	-0.028169	-0.031

4.4.2. Correlation Visualizations

In [30]:
```
# Heatmap

# Melt the correlation matrix into a long format so Altair can work with it
correlation_melted = correlation_matrix.reset_index().melt('index', var_name='Column', value_name='Correlation')


# Create a heatmap using Altair
heatmap = alt.Chart(correlation_melted).mark_rect().encode(
```

```
    x=alt.X('index:O', title=None, sort=corr_order),  # Specify the custom order for X-axis
    y=alt.Y('Column:O', title=None, sort=corr_order), # Specify the custom order for Y-axis
    color=alt.Color('Correlation:Q',
                    scale=alt.Scale(domain=[-1, 0, 1], scheme="redblue", reverse=True)
                    ),
#                     scale=alt.Scale(domain=[-1, 0, 1],
#                                     range=[colors[2],'white', colors[0]])),  # Set scale domain for color
    tooltip=[
        alt.Tooltip('index', title='Variable 1'),
        alt.Tooltip('Column', title='Variable 2'),
        alt.Tooltip('Correlation', title='Correlation')
    ]
).properties(
#     title='Correlation Heatmap',
    width=600,
    height=600
).configure_axisX(
    orient='top',  # Move x-axis labels to the top
    labelFontSize=14,
    labelAngle=-45,  # Rotate x-axis labels by 45 degrees
    domain=False  # Remove axis lines
).configure_axisY(
    labelFontSize=14,
    domain=False  # Remove axis lines
).configure_legend(
    titleFontSize=14,  # Set the font size for the legend title
    labelFontSize=14  # Set the font size for the legend
)

# Display the heatmap
heatmap
```

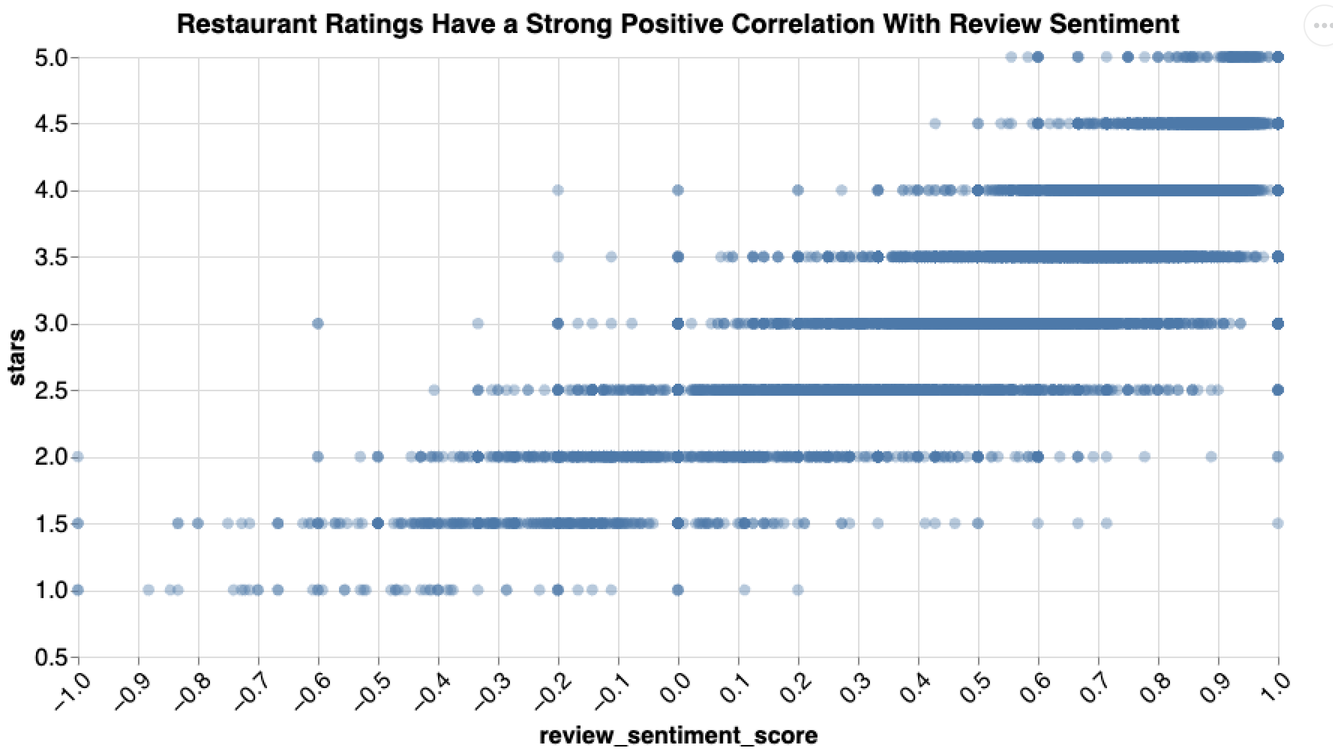
Out[30]:



In [31]:
```
# Scatter plot

alt.Chart(master4[['stars', 'review_sentiment_score']]).mark_circle(size=60, opacity=0.4).encode(
    x=alt.X('review_sentiment_score'),
    y=alt.Y('stars', scale=alt.Scale(domain=[0.5, 5]))
).properties(
    title='Restaurant Ratings Have a Strong Positive Correlation With Review Sentiment',
    width=800,
    height=400
).configure_title(
    fontSize=18,
```

```
).configure_axisX(
    titleFontSize=16,
    labelFontSize=16,
    labelAngle=-45,  # Rotate x-axis labels by 45 degrees
    domain=False  # Remove axis lines
).configure_axisY(
    titleFontSize=16,
    labelFontSize=16,
    domain=False,  # Remove axis lines
    tickMinStep=0.5
).configure_legend(
    titleFontSize=16,  # Set the font size for the legend title
    labelFontSize=16,  # Set the font size for the legend
)
```

Out[31]:



In [32]:
```
important_numerics = ['review_sentiment_score','tip_sentiment_score',
                      'review_count','tip_count']
```

4.4.3. Aggregating Data at Zip Code Level Reveals Higher Correlations Between Demographics and Ratings

Insights

After aggregating data at the zip code level and revisiting the correlation matrix, we discovered stronger correlations between certain demographic features and restaurant ratings/reviews. This is particularly notable for the feature “percentage of White population”. This could be attributed to the increased variability in demographic features when aggregated at the zip code level.

In [33]:
```
data_zipCode = (master4.groupby('zip_code')[['stars'] + numerical_vars]
 .mean().reset_index().set_index('zip_code'))

data_zipCode.head()
```

Out[33]:

zip_code	stars	median_household_income	household_cnt	median_age	bachelors_pcnt	education_pcnt	hispanic_latino_pcnt	white_pcnt	asian_pcnt	population
8002	3.457143	103031.0	9304.0	39.8	0.2430	0.7254	0.110	0.690	0.122	
8003	3.617647	145590.0	11485.0	46.0	0.3243	0.7213	0.058	0.729	0.169	
8004	3.416667	97821.0	4014.0	44.9	0.1905	0.7809	0.079	0.795	0.025	
8007	4.071429	87761.0	2528.0	46.3	0.2503	0.7667	0.025	0.889	0.034	
8009	3.769231	94351.0	5523.0	42.9	0.2262	0.6898	0.103	0.704	0.023	

In [34]:
```
correlation_matrix_zipCode = data_zipCode.corr()

corr_order_zipCode = correlation_matrix_zipCode.sort_values(by='stars', ascending=False).index.tolist()
correlation_matrix_zipCode = correlation_matrix_zipCode.sort_values(by='stars', ascending=False)[corr_order_zipCode]
correlation_matrix_zipCode
```

Out[34]:

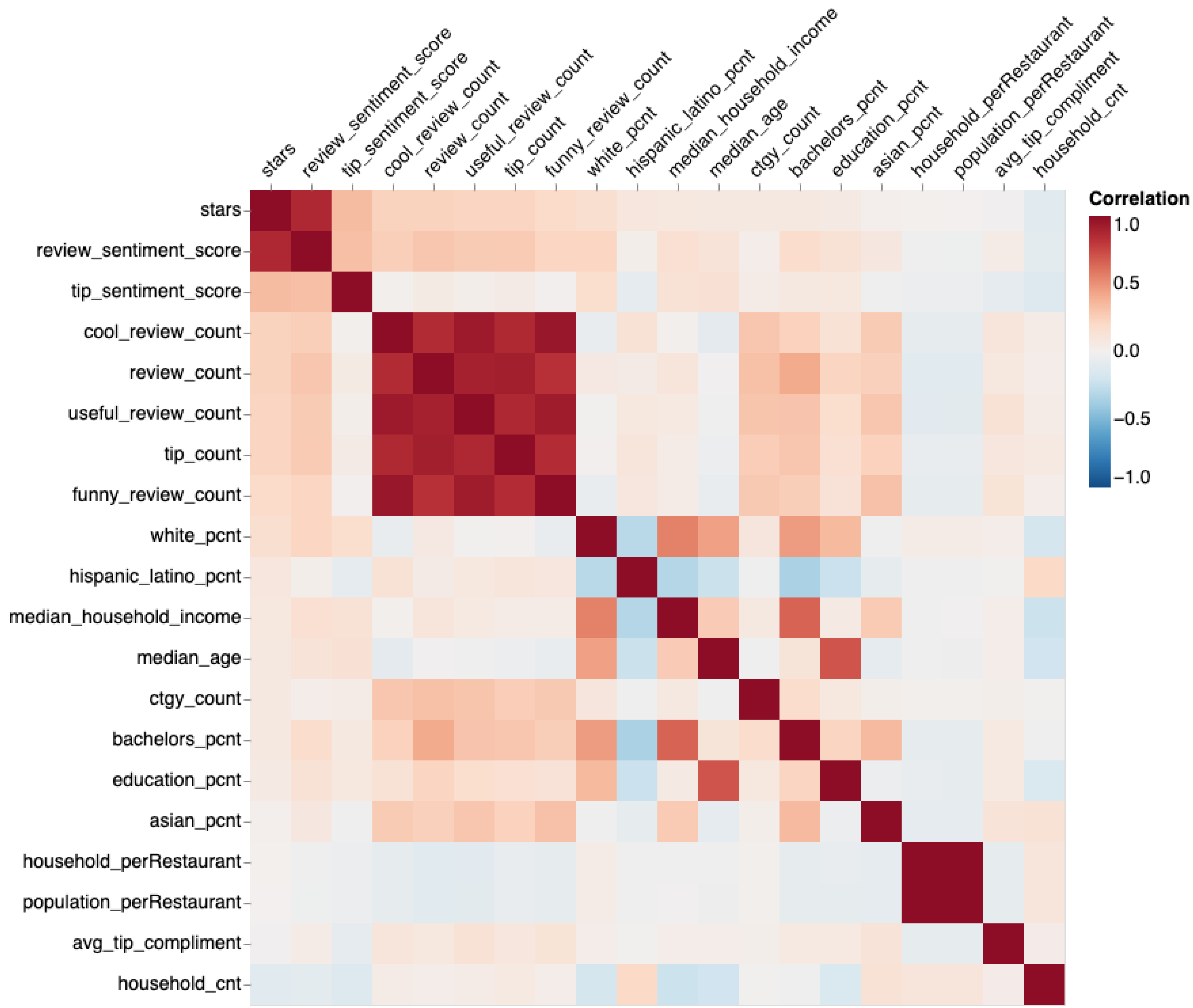
	stars	review_sentiment_score	tip_sentiment_score	cool_review_count	review_count	useful_review_count	tip_count	funny_review_co
stars	1.000000	0.867524	0.337616	0.236091	0.234006	0.226347	0.224825	0.194
review_sentiment_score	0.867524	1.000000	0.326468	0.253912	0.295113	0.272154	0.270805	0.22
tip_sentiment_score	0.337616	0.326468	1.000000	0.009465	0.052898	0.023540	0.046681	0.005
cool_review_count	0.236091	0.253912	0.009465	1.000000	0.855161	0.935045	0.864044	0.950
review_count	0.234006	0.295113	0.052898	0.855161	1.000000	0.900033	0.912606	0.832
useful_review_count	0.226347	0.272154	0.023540	0.935045	0.900033	1.000000	0.867364	0.926
tip_count	0.224825	0.270805	0.046681	0.864044	0.912606	0.867364	1.000000	0.846
funny_review_count	0.194240	0.221617	0.005109	0.950301	0.832542	0.926687	0.846521	1.000
white_pcnt	0.160488	0.221185	0.174306	-0.060675	0.065915	-0.004355	0.005607	-0.065
hispanic_latino_pcnt	0.086603	0.021281	-0.079878	0.134003	0.048144	0.080906	0.104060	0.092
median_household_income	0.079310	0.153261	0.129856	0.008730	0.106681	0.063558	0.040204	0.039
median_age	0.074137	0.119724	0.138485	-0.090060	-0.013760	-0.026247	-0.036517	-0.061
ctgy_count	0.069595	0.027198	0.040981	0.297720	0.317743	0.303662	0.262704	0.281
bachelors_pcnt	0.069465	0.181456	0.070125	0.239788	0.409961	0.304936	0.295911	0.255
education_pcnt	0.060416	0.130671	0.082640	0.127656	0.227444	0.172364	0.152203	0.125
asian_pcnt	0.018601	0.075538	-0.016580	0.271795	0.246838	0.291333	0.240134	0.315
household_perRestaurant	0.003572	-0.024391	-0.036245	-0.075292	-0.107146	-0.100516	-0.062040	-0.078
population_perRestaurant	0.002695	-0.028328	-0.036330	-0.073446	-0.104931	-0.098957	-0.058059	-0.076
avg_tip_compliment	-0.015191	0.045089	-0.080980	0.109261	0.080647	0.128734	0.094587	0.123
household_cnt	-0.108637	-0.087775	-0.132102	0.045738	0.029724	0.043955	0.064546	0.027

In [35]:

```
# Heatmap

# Melt the correlation matrix into a long format so Altair can work with it
correlation_melted = correlation_matrix_zipCode.reset_index().melt('index', var_name='Column', value_name='Correlation')


# Create a heatmap using Altair
heatmap = alt.Chart(correlation_melted).mark_rect().encode(
    x=alt.X('index:O', title=None, sort=corr_order_zipCode),  # Specify the custom order for X-axis
    y=alt.Y('Column:O', title=None, sort=corr_order_zipCode), # Specify the custom order for Y-axis
    color=alt.Color('Correlation:Q',
                    scale=alt.Scale(domain=[-1, 0, 1], scheme="redblue", reverse=True)
                    ),
#                     scale=alt.Scale(
#                                     range=[colors[2],'white', colors[0]])),  # Set scale domain for color
    tooltip=[
        alt.Tooltip('index', title='Variable 1'),
        alt.Tooltip('Column', title='Variable 2'),
```

```
        alt.Tooltip('Correlation', title='Correlation')
    ]
).properties(
#     title='Correlation Heatmap',
    width=600,
    height=600
).configure_axisX(
    orient='top',  # Move x-axis labels to the top
    labelFontSize=14,
    labelAngle=-45,  # Rotate x-axis labels by 45 degrees
    domain=False  # Remove axis lines
).configure_axisY(
    labelFontSize=14,
    domain=False  # Remove axis lines
).configure_legend(
    titleFontSize=14,  # Set the font size for the legend title
    labelFontSize=14  # Set the font size for the legend
)

# Display the heatmap
heatmap
```

Out[35]:



4.5. Examine Relationships Between Stars and Binary Categorical Variables

insights

By examining the differences in means and boxplots, we found 12 binary features that significantly influence restaurant ratings. For instance, fast-food restaurants tend to receive lower ratings, while restaurants offering plant-based or seafood dishes tend to receive higher ratings.

- Below binaries have strong predictive power for stars:

`BusinessAcceptsCreditCards` ,

`RestaurantsTakeOut` ,

`RestaurantsReservations` ,

`GoodForKids` ,

`plant_based` ,

`seafood` ,

`gluten_free` ,

`fast_food` ,

`breakfast` ,

`nightlife`

- Below binaries have some predictive power for stars:

`RestaurantsGoodForGroups` ,

`noise_loud`

4.5.1. Statistical examination of the relationship between Binary Variables and Restaurant Ratings

4.5.1.1. statistical summary

In [36]:
```
# Check average stars by each binary feature

binary_meanStars = pd.DataFrame(columns=['feature', 0, 1])

for var in categorical_vars_binary:

    df = master4.pivot_table(values='stars', columns=f'{var}', aggfunc='mean').reset_index(drop=True)
    df['feature'] = f'{var}'
    df[[df.columns[-1]]+ df.columns[:-1].tolist()]

    # Add the results to the result DataFrame
    binary_meanStars = pd.concat([binary_meanStars, df], ignore_index=True)

binary_meanStars['diff_in_mean'] = binary_meanStars[1] - binary_meanStars[0]

binary_meanStars.sort_values(by='diff_in_mean')
```

Out[36]:

	feature	0	1	diff_in_mean
16	fast_food	3.614092	2.500000	-1.114092
17	breakfast	3.531828	3.086505	-0.445323
12	noise_loud	3.515266	3.189024	-0.326241
0	RestaurantsDelivery	3.654309	3.389665	-0.264644
2	BusinessAcceptsCreditCards	3.684536	3.474171	-0.210365
8	GoodForKids	3.643289	3.452244	-0.191045
10	HasTV	3.553286	3.473462	-0.079824
9	RestaurantsGoodForGroups	3.540047	3.483901	-0.056147
4	RestaurantsTakeOut	3.517979	3.497693	-0.020286
5	Alcohol	3.431049	3.595800	0.164751
11	free_WiFi	3.444326	3.629852	0.185526
1	OutdoorSeating	3.431015	3.628657	0.197642
7	RestaurantsReservations	3.433642	3.631651	0.198010
18	nightlife	3.476459	3.680718	0.204259
3	BikeParking	3.397763	3.605746	0.207984
6	Caters	3.407312	3.620200	0.212888
14	seafood	3.486141	3.702937	0.216797
15	gluten_free	3.496125	3.815789	0.319664
13	plant_based	3.493134	3.871981	0.378846

In [37]:

```
# Check average stars by each binary feature

binary_medianStars = pd.DataFrame(columns=['feature', 0, 1])

for var in categorical_vars_binary:

    df = master4.pivot_table(values='stars', columns=f'{var}', aggfunc='median').reset_index(drop=True)
    df['feature'] = f'{var}'
    df[[df.columns[-1]]+ df.columns[:-1].tolist()]

    # Add the results to the result DataFrame
    binary_medianStars = pd.concat([binary_medianStars, df], ignore_index=True)

binary_medianStars['diff_in_median'] = binary_medianStars[1] - binary_medianStars[0]

binary_medianStars.sort_values(by='diff_in_median')
```

Out[37]:

	feature	0	1	diff_in_median
16	fast_food	3.5	2.5	-1.0
2	BusinessAcceptsCreditCards	4.0	3.5	-0.5
17	breakfast	3.5	3.0	-0.5
8	GoodForKids	4.0	3.5	-0.5
0	RestaurantsDelivery	3.5	3.5	0.0
12	noise_loud	3.5	3.5	0.0
11	free_WiFi	3.5	3.5	0.0
10	HasTV	3.5	3.5	0.0
9	RestaurantsGoodForGroups	3.5	3.5	0.0
6	Caters	3.5	3.5	0.0
5	Alcohol	3.5	3.5	0.0
4	RestaurantsTakeOut	3.5	3.5	0.0
3	BikeParking	3.5	3.5	0.0
1	OutdoorSeating	3.5	3.5	0.0
7	RestaurantsReservations	3.5	3.5	0.0
18	nightlife	3.5	3.5	0.0
13	plant_based	3.5	4.0	0.5
14	seafood	3.5	4.0	0.5
15	gluten_free	3.5	4.0	0.5

4.5.1.2. statistical testing

We wanted to examine the relationship between a binary categorical variable and restaurant ratings.

We first employed statistical methods, considering both an independent samples t-test and a Mann-Whitney U test.

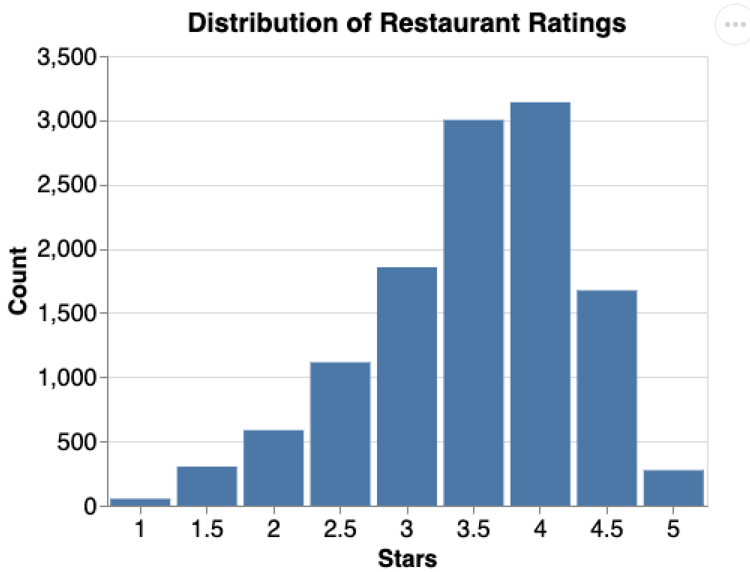
The former assumes a normal distribution of the data, while the latter is a non-parametric test that does not assume normal distribution.

In [38]:

```
# Check normality of ratings

alt.Chart(master4).mark_bar().encode(
    x=alt.X('stars:O', title='Stars'),
    y=alt.Y('count():Q', title='Count')
).properties(
    title='Distribution of Restaurant Ratings',
    width=400
).configure_title(
    fontSize=18,
).configure_axis(
```

```
    titleFontSize=16,
    labelFontSize=16,
    labelAngle=0
).configure_axisY(
    titleFontSize=16,
    labelFontSize=16,
)
```

Out[38]:



In [39]:

```
# independent samples t-test

def binary_vs_stars(data, binary_vars, target='stars'):
    results = []

    for var in binary_vars:
        group_1 = data[data[var] == 1][target]
        group_0 = data[data[var] == 0][target]

        t_stat, p_value = ttest_ind(group_1, group_0, equal_var=False)
        results.append({'variable': var, 't_stat': t_stat, 'p_value': p_value})

    return pd.DataFrame(results)
```

In [40]:

```
results = binary_vs_stars(master4, categorical_vars_binary)

results[results.p_value<=0.05].sort_values(by='p_value', ascending=True).reset_index(drop=True)
```

Out[40]:

	variable	t_stat	p_value
0	fast_food	-45.506297	3.768222e-281
1	RestaurantsDelivery	-18.689142	8.363966e-77
2	Caters	15.216661	8.339550e-52
3	BikeParking	14.586364	9.117752e-48
4	RestaurantsReservations	14.575403	1.149347e-47
5	OutdoorSeating	14.007231	3.613509e-44
6	free_WiFi	12.822435	2.795784e-37
7	Alcohol	12.000374	5.464874e-33
8	breakfast	-12.295862	2.526524e-32
9	nightlife	11.636195	2.218260e-30
10	GoodForKids	-10.389417	5.403124e-25
11	noise_loud	-9.757889	4.753880e-21
12	seafood	9.215842	2.013606e-19
13	BusinessAcceptsCreditCards	-8.298450	2.086887e-16
14	plant_based	8.766887	5.249518e-16
15	gluten_free	6.561609	9.901693e-10
16	HasTV	-4.903519	9.633139e-07
17	RestaurantsGoodForGroups	-3.114910	1.850450e-03

In [41]:

```
# Mann-Whitney U test

def u_test(data, binary_vars, target='stars'):

    results = []

    for var in binary_vars:
        group_0 = data[data[var] == 0][target]
        group_1 = data[data[var] == 1][target]

        u_stat, p_value = mannwhitneyu(group_0, group_1)
        results.append({'variable': var, 'u_stat': u_stat, 'p_value': p_value})

    return pd.DataFrame(results)
```

In [42]:

```
results = u_test(master4, categorical_vars_binary)
results[results.p_value<=0.05].sort_values(by='p_value', ascending=True).reset_index(drop=True)
```

Out[42]:

	variable	u_stat	p_value
0	fast_food	11157099.0	0.000000e+00
1	RestaurantsDelivery	20590455.0	1.067951e-63
2	GoodForKids	15784681.0	5.841818e-48
3	BikeParking	15678696.5	5.233242e-36
4	Caters	15455982.0	4.371283e-34
5	breakfast	5940796.5	6.832547e-31
6	BusinessAcceptsCreditCards	9075320.0	8.604730e-31
7	OutdoorSeating	14362249.5	7.413388e-29
8	free_WiFi	13305461.5	1.488185e-25
9	RestaurantsReservations	14217531.5	1.088285e-24
10	noise_loud	4068552.5	3.114894e-23
11	nightlife	6366792.0	4.542137e-14
12	Alcohol	16132592.0	5.195740e-14
13	HasTV	17166957.5	2.638882e-13
14	plant_based	884539.0	3.597155e-12
15	RestaurantsGoodForGroups	15672191.0	2.094831e-11
16	seafood	3627010.5	6.097189e-11
17	gluten_free	609691.0	3.855741e-06
18	RestaurantsTakeOut	6716372.5	4.371120e-04

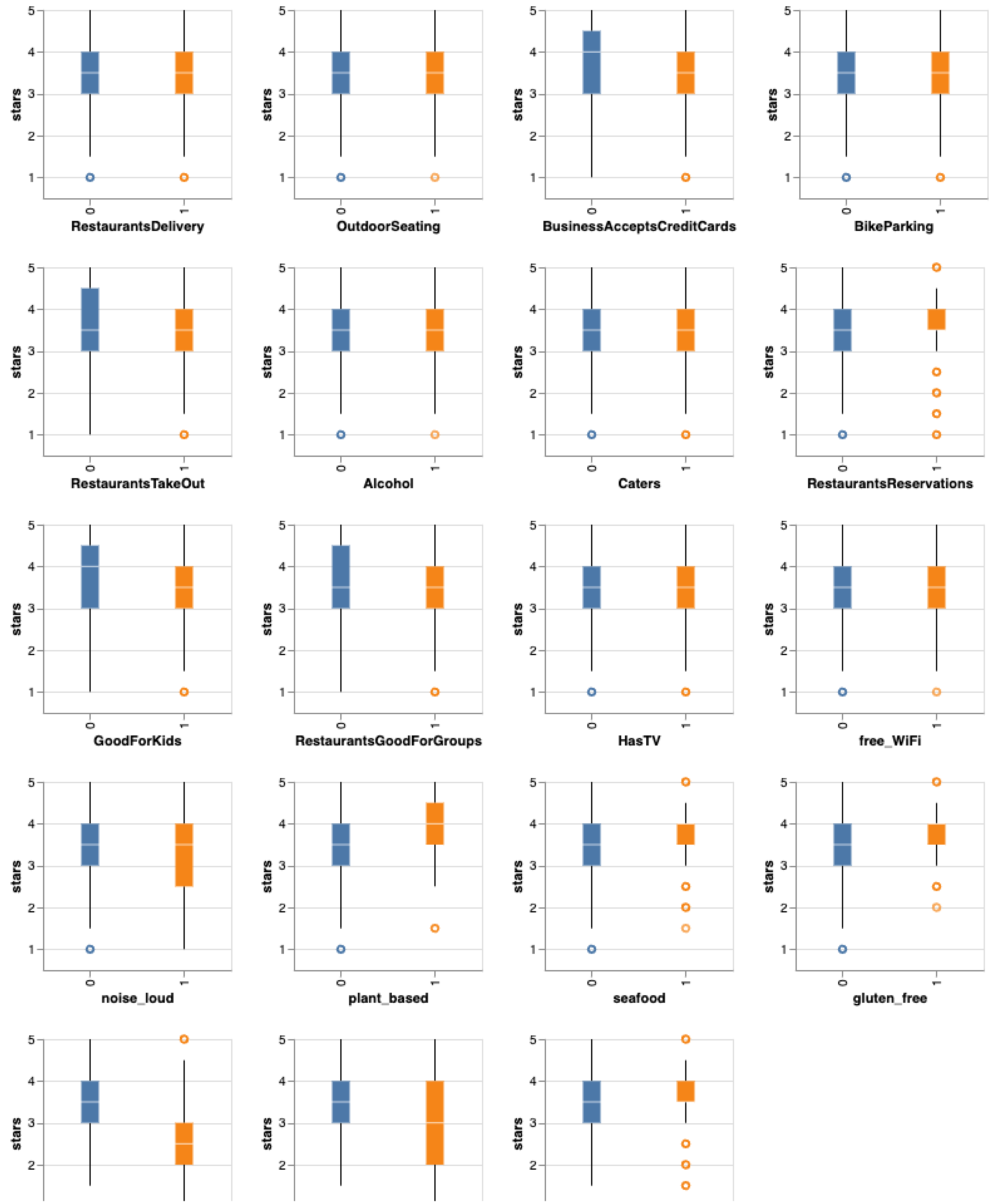
4.5.2. Visualizations

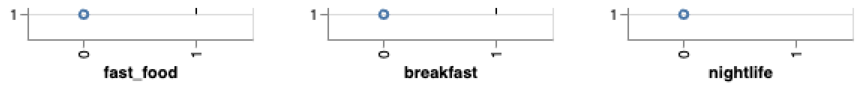
In [43]:
```
def box_plot(df, var):

    chart = alt.Chart(df).mark_boxplot().encode(
        x=alt.X(f'{var}:N'),
        y=alt.Y('stars:Q', scale=alt.Scale(domain=[0.5, 5])),
        color=alt.Color(f'{var}:N', legend=None)
    ).properties(
        width=150,
        height=150
    )

    return chart
```

In [44]:
```
# Boxplot

charts = []
```

```
for var in categorical_vars_binary:
    charts.append(box_plot(master4, var))


((charts[0] | charts[1] | charts[2] | charts[3])
& (charts[4] | charts[5] | charts[6] | charts[7])
& (charts[8] | charts[9] | charts[10] | charts[11])
& (charts[12] | charts[13] | charts[14] | charts[15])
& (charts[16] | charts[17] | charts[18]))
```

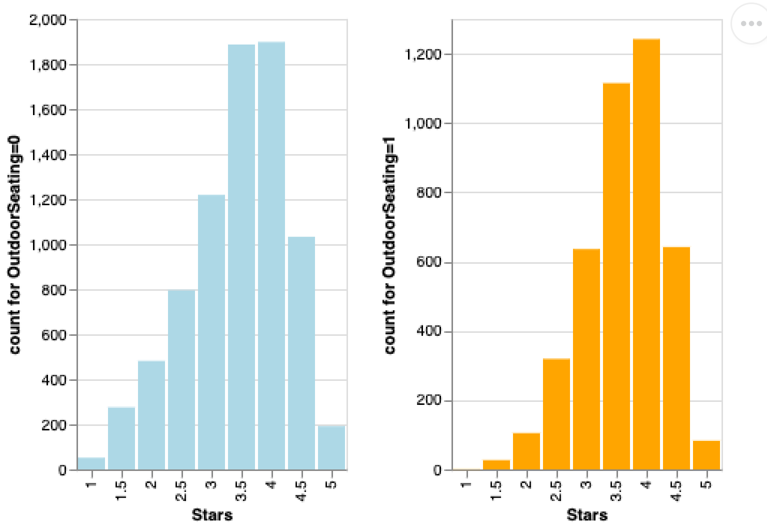
Out[44]:


RestaurantsDelivery
OutdoorSeating
BusinessAcceptsCreditCards
BikeParking
RestaurantsTakeOut
Alcohol
Caters
RestaurantsReservations
GoodForKids
RestaurantsGoodForGroups
HasTV
free_WiFi
noise_loud
plant_based
seafood
gluten_free
stars



In [45]:
```
# Double check distributions for the first binary variable

var = categorical_vars_binary[1]

chart1 = alt.Chart(master4).mark_bar(opacity=1).encode(
        x=alt.X('stars:O', title='Stars'),
        y=alt.Y('count()', title=f'count for {var}=0'),
        color=alt.value('lightblue')  # Set color for the first bar
    ).transform_filter(
        f'datum.{var} == 0'
    )

chart2 = alt.Chart(master4).mark_bar(opacity=1).encode(
        x=alt.X('stars:O', title='Stars'),
        y=alt.Y('count()', title=f'count for {var}=1'),
        color=alt.value('orange')  # Set color for the second bar
    ).transform_filter(
        f'datum.{var} == 1'
    )


chart1 | chart2
```

Out[45]:



In [46]:
```
index_toSelect = [2,4,7,8,13,14,15,16,17,18]
important_binaries = [categorical_vars_binary[x] for x in index_toSelect]
print(f"""Below binaries have strong predictive power for stars:\n {important_binaries}""")

print(f"""\n Below binaries have some predictive power for stars:
{categorical_vars_binary[9]},
```

```
{categorical_vars_binary[12]}""")

important_binaries.append(categorical_vars_binary[9])
important_binaries.append(categorical_vars_binary[12])
```

```
Below binaries have strong predictive power for stars:
 ['BusinessAcceptsCreditCards', 'RestaurantsTakeOut', 'RestaurantsReservations', 'GoodForKids', 'plant_based', 'seafood', 'gluten_free', 'fast_foo
d', 'breakfast', 'nightlife']

 Below binaries have some predictive power for stars:
RestaurantsGoodForGroups,
noise_loud
```

In [47]:

```
important_binaries
```

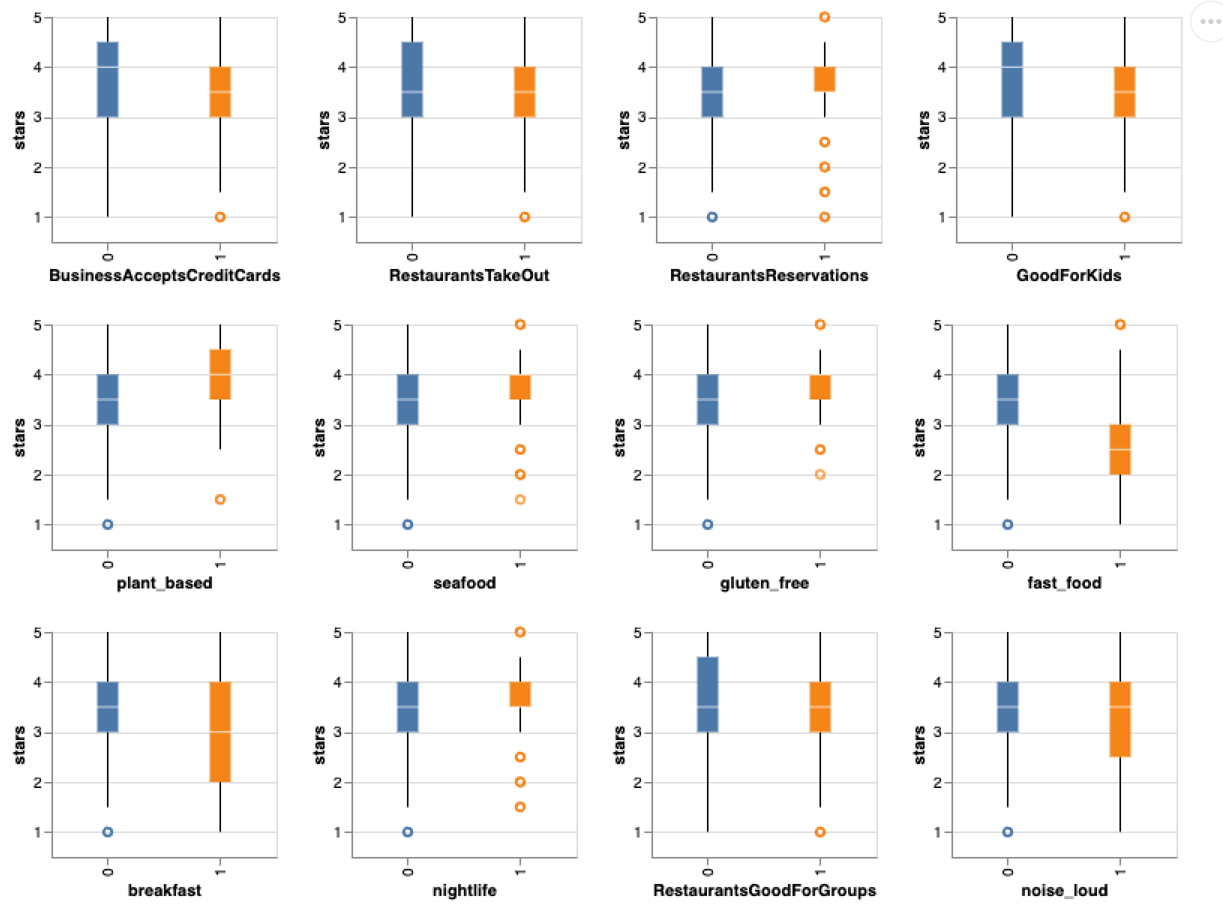
Out[47]:

```
['BusinessAcceptsCreditCards',
 'RestaurantsTakeOut',
 'RestaurantsReservations',
 'GoodForKids',
 'plant_based',
 'seafood',
 'gluten_free',
 'fast_food',
 'breakfast',
 'nightlife',
 'RestaurantsGoodForGroups',
 'noise_loud']
```

In [48]:

```
# Boxplot for only important binaries

charts = []

for var in important_binaries:
    charts.append(box_plot(master4, var))


((charts[0] | charts[1] | charts[2] | charts[3])
& (charts[4] | charts[5] | charts[6] | charts[7])
& (charts[8] | charts[9] | charts[10] | charts[11]))
```

Out[48]:



4.6. Examine Relationships Between Cuisine and Key Numerical / Binary Variables

explain & insights

We aggregated the key features identified by cuisine, standardized the data for each feature, and visualized the results in a heatmap, as depicted in Figure 6 below. This allows us to easily observe the discrepancies in key features across different cuisines. The greater difference in shading indicates more significant disparities.

For instance, Mexican cuisine tends to focus more on plant-based options, Italian cuisine leans towards being more gluten-free, Chinese cuisine is often perceived as kid-friendly and offers seafood dishes, and Japanese cuisine is less likely to have a noisy environment.

Additionally, we can observe that Mexican restaurants are typically located in areas with higher Hispanic-Latino populations and are less likely to be found in areas with higher Asian populations. On the other hand, both Chinese and Japanese restaurants are prevalent in communities with higher Asian populations, but Chinese restaurants are less commonly found in areas with higher White populations compared to Japanese restaurants.

In [49]:
```
# master4.groupby('cuisine')[numerical_vars+categorical_vars_binary].mean().round(2).T
vars = important_numerics+important_binaries + ['asian_pcnt', 'white_pcnt','hispanic_latino_pcnt']

cuisine_impVar = master4.groupby('cuisine')[vars].mean().round(2).T

cuisine_impVar
```

Out[49]:

cuisine	Chinese	Italian	Japanese	Mexican
review_sentiment_score	0.57	0.64	0.73	0.61
tip_sentiment_score	0.44	0.50	0.55	0.46
review_count	58.27	95.10	125.24	91.05
tip_count	8.97	11.93	18.43	13.43
BusinessAcceptsCreditCards	0.86	0.92	0.92	0.84
RestaurantsTakeOut	0.93	0.92	0.93	0.87
RestaurantsReservations	0.23	0.45	0.59	0.21
GoodForKids	0.79	0.74	0.73	0.75
plant_based	0.01	0.01	0.01	0.02
seafood	0.08	0.07	0.05	0.05
gluten_free	0.00	0.02	0.00	0.00
fast_food	0.06	0.06	0.01	0.19
breakfast	0.01	0.04	0.00	0.15
nightlife	0.01	0.15	0.08	0.14
RestaurantsGoodForGroups	0.62	0.76	0.77	0.72
noise_loud	0.05	0.05	0.04	0.05
asian_pcnt	0.06	0.06	0.06	0.05
white_pcnt	0.64	0.71	0.69	0.66
hispanic_latino_pcnt	0.13	0.11	0.12	0.18

In [50]:
```
cuisine_impVar2 = StandardScaler().fit_transform(cuisine_impVar.T).T
cuisine_impVar2
```

Out[50]:
```
array([[-1.14608617,  0.04244764,  1.57056253, -0.466924  ],
       [-1.12943277,  0.29721915,  1.48609575, -0.65388213],
       [-1.4389015 ,  0.11314835,  1.3832755 , -0.05752235],
       [-1.23226518, -0.36792752,  1.53011126,  0.07008143],
       [-0.70014004,  0.98019606,  0.98019606, -1.26025208],
       [ 0.70352647,  0.30151134,  0.70352647, -1.70856429],
       [-0.88543774,  0.50596443,  1.39140217, -1.01192885],
       [ 1.6464639 , -0.5488213 , -0.98787834, -0.10976426],
       [-0.57735027, -0.57735027, -0.57735027,  1.73205081],
       [ 1.34715063,  0.57735027, -0.96225045, -0.96225045],
       [-0.57735027,  1.73205081, -0.57735027, -0.57735027],
       [-0.29981268, -0.29981268, -1.04934436,  1.64896972],
       [-0.67134509, -0.16783627, -0.83918136,  1.67836272],
       [-1.52052622,  0.98386991, -0.26832816,  0.80498447],
       [-1.64365403,  0.71646458,  0.88504448,  0.04214498],
       [ 0.57735027,  0.57735027, -1.73205081,  0.57735027],
       [ 0.57735027,  0.57735027,  0.57735027, -1.73205081],
       [-1.29986737,  1.29986737,  0.55708601, -0.55708601],
       [-0.18569534, -0.92847669, -0.55708601,  1.67125804]])
```

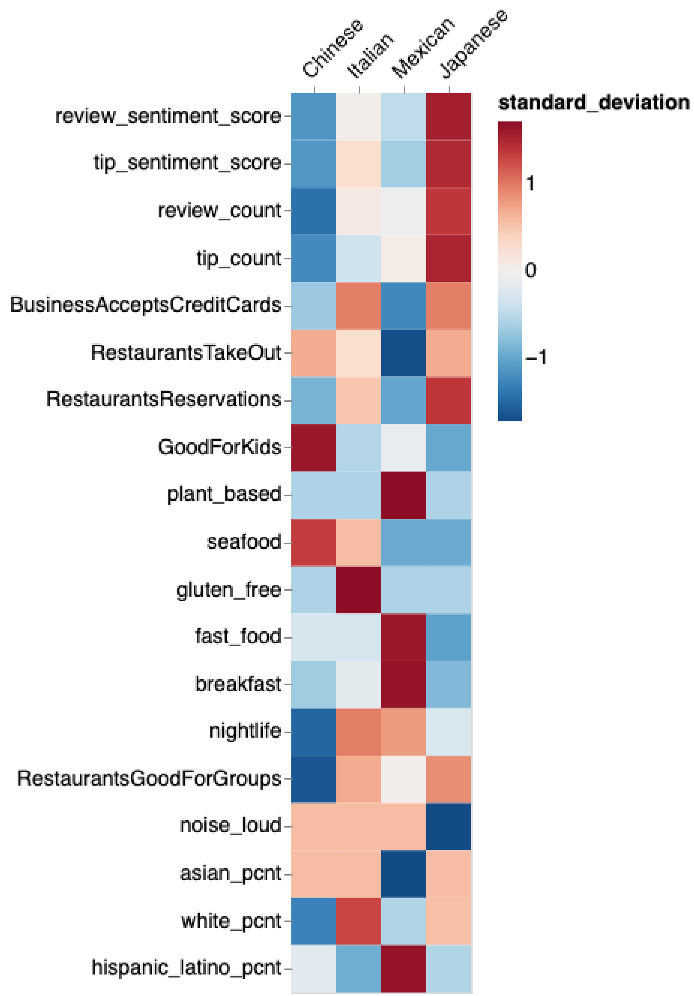
In [51]:
```
a = np.array([0.57, 0.64, 0.73, 0.61])
(a - np.mean(a) )/ np.std(a)
```

Out[51]:
```
array([-1.14608617,  0.04244764,  1.57056253, -0.466924  ])
```

In [52]:
```
# Standardize values by row
scaler = StandardScaler()
standardized_cuisine_impVar = pd.DataFrame(scaler.fit_transform(cuisine_impVar.T).T,
                               columns=cuisine_impVar.columns, index=cuisine_impVar.index)

# Reset the index for Altair compatibility
standardized_cuisine_impVar = standardized_cuisine_impVar.reset_index()

# Melt the DataFrame for Altair heatmap plotting
melted_df = pd.melt(standardized_cuisine_impVar,
                    id_vars=['index'],value_vars=['Chinese','Italian','Japanese','Mexican'], var_name='cuisine', value_name='standardized_value')


# Create a heatmap using Altair
varOrder = important_numerics+important_binaries
cuisineOrder = master4.groupby('cuisine').stars.mean().sort_values().index.tolist()

heatmap = alt.Chart(melted_df).mark_rect().encode(
    x=alt.X('cuisine:O', title=None, sort=cuisineOrder),  # Specify the custom order
    y=alt.Y('index:O', title=None, sort=varOrder, # Specify the custom order
            axis=alt.Axis(labelLimit=0, bandPosition=0.5)),
    color=alt.Color('standardized_value:Q',
                    title='standard_deviation',
                    scale=alt.Scale(scheme="redblue", reverse=True),
#                     legend=alt.Legend(orient='top')
                   ),
    tooltip=[
        alt.Tooltip('index', title='Variable 1'),
        alt.Tooltip('cuisine', title='Variable 2'),
        alt.Tooltip('standardized_value', title='Value')
```

```
    ]
).properties(
#     title='',
    width=120,
    height=600
).configure_axisX(
    orient='top',  # Move x-axis labels to the top
    labelFontSize=14,
    labelAngle=-45,  # Rotate x-axis labels by 45 degrees
    domain=False  # Remove axis lines
).configure_axisY(
    labelFontSize=14,
    domain=False  # Remove axis lines
).configure_legend(
    titleFontSize=14,  # Set the font size for the legend title
    labelFontSize=14  # Set the font size for the legend
)

# Display the heatmap
heatmap
```

Out[52]:



In [53]:

```
# master4.groupby('cuisine')[numerical_vars+categorical_vars_binary].mean().round(2).T
cuisine_demoVar = master4.groupby('cuisine')[numerical_vars[:10]].mean().round(2).T

cuisine_demoVar
```

Out[53]:

cuisine	Chinese	Italian	Japanese	Mexican
median_household_income	78783.17	90289.47	86559.39	77189.93
household_cnt	13192.51	12165.99	12937.20	13507.33
median_age	38.59	40.29	38.83	38.37
bachelors_pcnt	0.24	0.27	0.28	0.25
education_pcnt	0.70	0.72	0.71	0.70
hispanic_latino_pcnt	0.13	0.11	0.12	0.18
white_pcnt	0.64	0.71	0.69	0.66
asian_pcnt	0.06	0.06	0.06	0.05
population_perRestaurant	408.04	371.76	289.47	389.89
household_perRestaurant	160.93	147.60	116.30	152.85

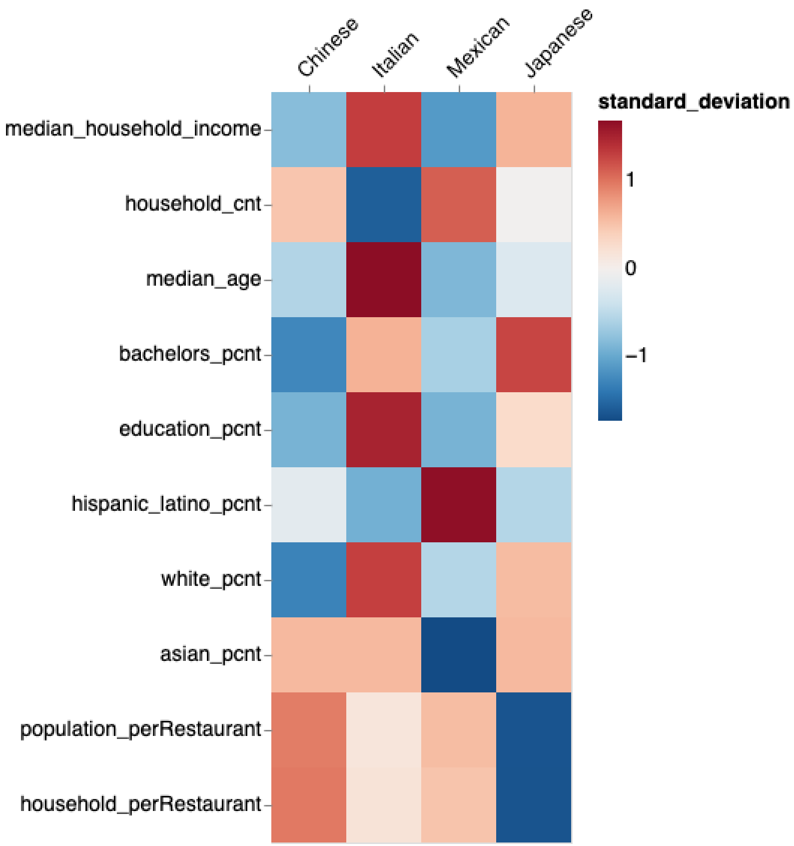
4.7. Examine Differences in Demographics Across Cuisines

In [54]:

```
# Standardize values by row
scaler = StandardScaler()
standardized_cuisine_demoVar = pd.DataFrame(scaler.fit_transform(cuisine_demoVar.T).T,
                               columns=cuisine_demoVar.columns, index=cuisine_demoVar.index)

# Reset the index for Altair compatibility
standardized_cuisine_demoVar = standardized_cuisine_demoVar.reset_index()

# Melt the DataFrame for Altair heatmap plotting
melted_df2 = pd.melt(standardized_cuisine_demoVar,
                    id_vars=['index'],value_vars=['Chinese','Italian','Japanese','Mexican'],
                    var_name='cuisine', value_name='standardized_value')


# Create a heatmap using Altair
varOrder = numerical_vars[:10]
cuisineOrder = master4.groupby('cuisine').stars.mean().sort_values().index.tolist()

heatmap = alt.Chart(melted_df2).mark_rect().encode(
    x=alt.X('cuisine:O', title=None, sort=cuisineOrder),  # Specify the custom order
    y=alt.Y('index:O', title=None, sort=varOrder, # Specify the custom order
            axis=alt.Axis(labelLimit=0, bandPosition=0.5)),
    color=alt.Color('standardized_value:Q', title='standard_deviation',
                    scale=alt.Scale(scheme="redblue", reverse=True),
#                     legend=alt.Legend(orient='top')
                   ),
    tooltip=[
        alt.Tooltip('index', title='Variable 1'),
        alt.Tooltip('cuisine', title='Variable 2'),
        alt.Tooltip('standardized_value', title='Value')
```

```
    ]
).properties(
#     title='',
    width=200,
    height=500
).configure_axisX(
    orient='top',  # Move x-axis labels to the top
    labelFontSize=14,
    labelAngle=-45,  # Rotate x-axis labels by 45 degrees
    domain=False  # Remove axis lines
).configure_axisY(
    labelFontSize=14,
    domain=False  # Remove axis lines
).configure_legend(
    titleFontSize=14,  # Set the font size for the legend title
    labelFontSize=14  # Set the font size for the legend
)

# Display the heatmap
heatmap
```

Out[54]:



In [55]:
```
(master4.groupby('zip_code')
 .agg({'hispanic_latino_pcnt':np.max, 'white_pcnt':np.max,
       'asian_pcnt':np.max, 'Chinese':np.mean, 'Japanese':np.mean,
       'Italian':np.mean, 'Mexican':np.mean, 'review_sentiment_score':np.mean, 'review_count':np.mean})
 .reset_index().set_index('zip_code').corr()).iloc[:3,-6:]
```

Out[55]:

	Chinese	Japanese	Italian	Mexican	review_sentiment_score	review_count
hispanic_latino_pcnt	-0.081893	-0.052701	-0.253963	0.332123	0.021281	0.048144
white_pcnt	-0.323098	0.092293	0.296251	-0.060697	0.221185	0.065915
asian_pcnt	0.037668	0.230532	0.038515	-0.144308	0.075538	0.246838

4.8. Principal Component Analysis (PCA)

PCA enabled us to identify underlying patterns, reduce noise, and extract the most influential features that contribute to explaining restaurant ratings. By transforming the high-dimensional data into a lower-dimensional space, PCA helped us visualize the structure of the data and identify the principal components that explain the majority of the variance. This process facilitated a clearer understanding of the factors driving customer satisfaction and allowed us to focus on the most relevant features in our analysis.

4.8.1. Conduct PCA

In [56]:
```
# Prepare data for PCA
X = master4[['business_id'] + numerical_vars + categorical_vars_binary].copy().set_index('business_id')

# Use sklearn "StandardScaler" package and its fit_transform method to standardize X
X_scaled = StandardScaler().fit_transform(X)

# Perform PCA
pca = PCA()
restaurant_pca = pca.fit_transform(X_scaled)

# Eigenvalues
eigenvalues = pca.explained_variance_

# Eigenvectors
# principal components are essentially eigenvectors of the covariance matrix of the centered data matrix
eigenvectors = pca.components_

print(X.shape)
print(X_scaled.shape)
print(restaurant_pca.shape)

print(eigenvalues.shape)
print(eigenvectors.shape)
```

```
(12005, 38)
(12005, 38)
(12005, 38)
(38,)
(38, 38)
```

In [57]:
```
principal_components = pca.components_

# calculates the projection of the scaled data onto the principal components
# and transposes the result to obtain the transformed data matrix in the principal component space
m = np.dot(principal_components, X_scaled.T).T

print(m.shape)
print(restaurant_pca.shape)

# Check if the matrices are the same
result = np.array_equal(restaurant_pca, m)
print("Are the matrices the same?", result)

# Check if the matrices are very close
result2 = np.allclose(restaurant_pca, m, atol=1e-10)
print("Are the matrices very close?", result2)
```

```
(12005, 38)
(12005, 38)
Are the matrices the same? False
Are the matrices very close? True
```

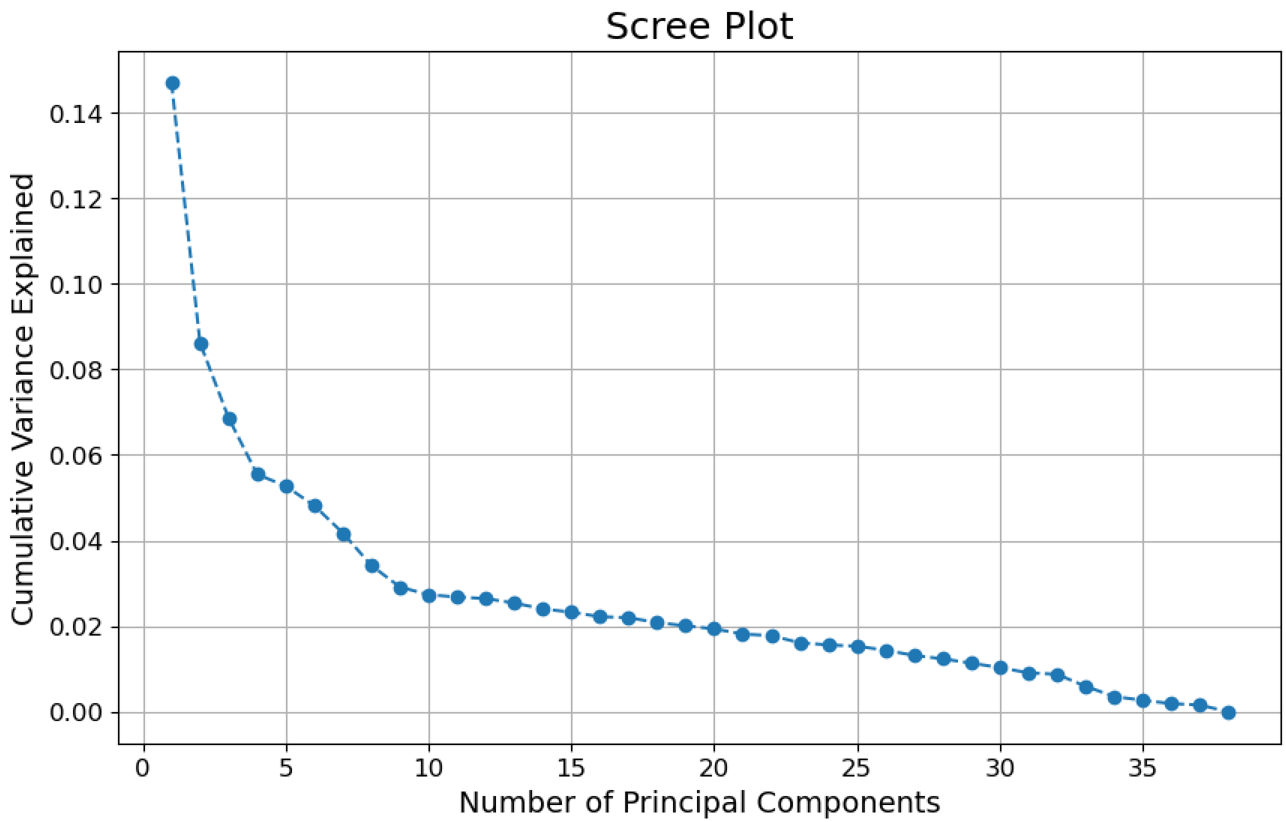
4.8.2. Scree Plot

explain

The Scree plot displays the eigenvalues of each principal component in PCA. Essentially, it provides insights into the amount of variance explained by each component and thus helps decide how many components to retain based on the amount of variance they explain.

By examining the Scree plot, it becomes apparent that the first two principal components do not explain much of the variance.

In [58]:
```
# Scree Plot
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance_ratio)

# Plotting Scree Plot
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(explained_variance_ratio) + 1), explained_variance_ratio, marker='o', linestyle='--')
plt.title('Scree Plot', fontsize=18)
plt.xlabel('Number of Principal Components', fontsize=14)
plt.ylabel('Cumulative Variance Explained', fontsize=14)
plt.tick_params(axis='both', which='major', labelsize=12)
plt.grid(True)
plt.show()
```



4.8.3. Check Loadings

In PCA, loadings refer to the coefficients that define the projection of the original variables onto the principal components.

When performing PCA on a dataset, we transform the original variables into a new set of variables called principal components. Each principal component is a linear combination of the original variables, where the coefficients of this combination are known as loadings. These loadings represent the contributions of each original variable to the principal component.

The loadings are crucial because they indicate how strongly each original variable influences the principal component. High loadings (positive or negative) suggest a strong influence, while low loadings suggest a weak influence or no influence at all. By examining the loadings, we can interpret the structure and relationships within the data and understand which variables contribute the most to each principal component.

In [59]:
```
# Extract the loadings for PC1 to PC4
loadings_pc1to4 = pca.components_[:4, :]

# Create a DataFrame with loadings
loadings_df = pd.DataFrame(loadings_pc1to4.T, columns=[f'PC{i}' for i in range(1, 5)], index=X.columns)

loadings_df
```

Out[59]:

	PC1	PC2	PC3	PC4
median_household_income	0.058513	-0.365948	-0.196184	0.121508
household_cnt	-0.043690	0.164305	0.129471	-0.005714
median_age	0.009868	-0.317268	-0.108177	0.052997
bachelors_pcnt	0.120092	-0.311411	-0.238592	0.107830
education_pcnt	0.071474	-0.258707	-0.206494	0.048091
hispanic_latino_pcnt	-0.031045	0.312654	0.175957	-0.129180
white_pcnt	0.054411	-0.382334	-0.144464	0.118261
asian_pcnt	0.055641	-0.009051	-0.107967	0.055963
population_perRestaurant	-0.049667	0.028995	0.120893	0.013940
household_perRestaurant	-0.050074	0.025850	0.120255	0.014554
review_count	0.371055	0.161923	-0.105504	0.095076
review_sentiment_score	0.130739	-0.036549	-0.083236	-0.449217
useful_review_count	0.364396	0.172111	-0.115794	0.074988
funny_review_count	0.346988	0.195868	-0.125368	0.114698
cool_review_count	0.355064	0.200703	-0.134429	0.089021
avg_tip_compliment	0.021455	-0.003394	0.021507	0.017460
tip_sentiment_score	0.102958	-0.066257	0.068757	-0.213196
tip_count	0.351377	0.167229	-0.066880	0.093435
ctgy_count	0.145142	0.002083	-0.004728	0.113594
RestaurantsDelivery	0.030402	-0.045542	0.145451	0.331362
OutdoorSeating	0.154703	-0.097965	0.053121	-0.104401
BusinessAcceptsCreditCards	0.129304	-0.159910	0.336219	0.061944
BikeParking	0.181312	-0.081563	0.194830	0.003610
RestaurantsTakeOut	0.075219	-0.127801	0.303011	0.050032
Alcohol	0.208864	-0.092236	0.130039	-0.242735
Caters	0.149923	-0.145442	0.193338	-0.049429
RestaurantsReservations	0.161429	-0.129744	0.055446	-0.227515
GoodForKids	0.082535	-0.096688	0.359198	0.117390
RestaurantsGoodForGroups	0.165103	-0.115113	0.305331	-0.008865
HasTV	0.086681	-0.081201	0.301822	0.073529
free_WiFi	0.145460	-0.130626	0.136779	-0.058816
noise_loud	0.016425	0.018292	0.016782	0.074345

	PC1	PC2	PC3	PC4
plant_based	0.048394	0.002893	-0.007888	0.008915
seafood	0.054514	-0.020839	-0.009270	-0.082999
gluten_free	0.055934	-0.019835	-0.007401	-0.005161
fast_food	-0.079629	0.054460	0.081996	0.484891
breakfast	0.005487	0.048795	0.038356	0.326147
nightlife	0.152732	-0.036179	-0.032900	-0.111334

In [60]:

```
# Sort loadings of PC1
PC1_sorted_loadings = loadings_df[['PC1']].apply(lambda x: abs(x)).sort_values(by=['PC1'], ascending=False)

# Display the sorted DataFrame
PC1_sorted_loadings.iloc[:6]
```

Out[60]:

	PC1
review_count	0.371055
useful_review_count	0.364396
cool_review_count	0.355064
tip_count	0.351377
funny_review_count	0.346988
Alcohol	0.208864

In [61]:

```
# Sort loadings of PC2
PC2_sorted_loadings = loadings_df[['PC2']].apply(lambda x: abs(x)).sort_values(by=['PC2'], ascending=False)

# Display the sorted DataFrame
PC2_sorted_loadings.iloc[:6]
```

Out[61]:

	PC2
white_pcnt	0.382334
median_household_income	0.365948
median_age	0.317268
hispanic_latino_pcnt	0.312654
bachelors_pcnt	0.311411
education_pcnt	0.258707

In [62]:

```
# Sort loadings of PC3
PC3_sorted_loadings = loadings_df[['PC3']].apply(lambda x: abs(x)).sort_values(by=['PC3'], ascending=False)
```

```
# Display the sorted DataFrame
PC3_sorted_loadings.iloc[:6]
```

Out[62]:

	PC3
GoodForKids	0.359198
BusinessAcceptsCreditCards	0.336219
RestaurantsGoodForGroups	0.305331
RestaurantsTakeOut	0.303011
HasTV	0.301822
bachelors_pcnt	0.238592

In [63]:
```
# Sort loadings of PC4
PC4_sorted_loadings = loadings_df[['PC4']].apply(lambda x: abs(x)).sort_values(by=['PC4'], ascending=False)

# Display the sorted DataFrame
PC4_sorted_loadings.iloc[:6]
```

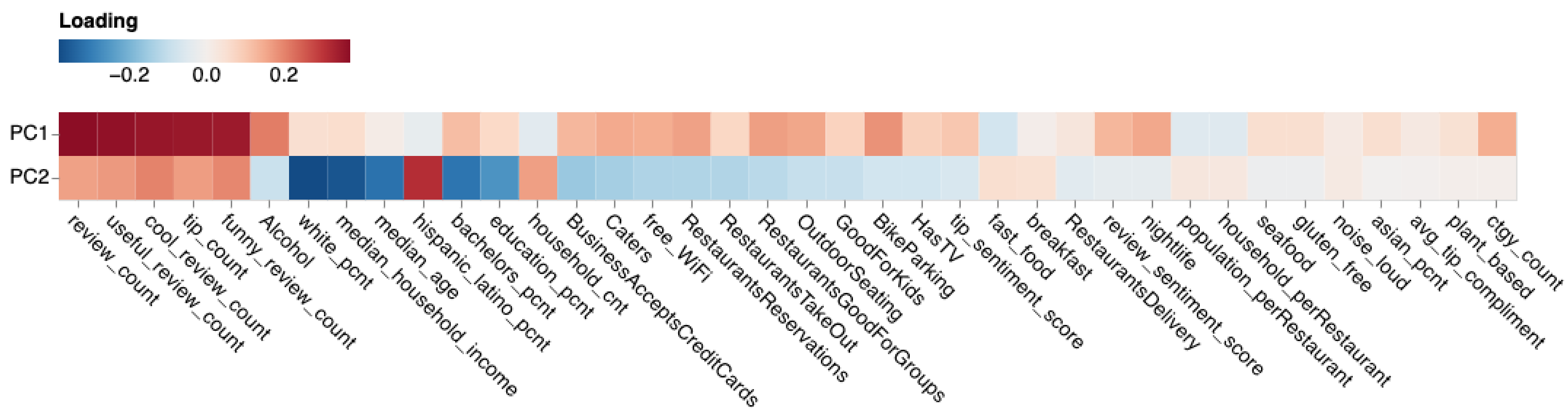
Out[63]:

	PC4
fast_food	0.484891
review_sentiment_score	0.449217
RestaurantsDelivery	0.331362
breakfast	0.326147
Alcohol	0.242735
RestaurantsReservations	0.227515

In [64]:
```
# Reshape the DataFrame for Altair

# Sort the DataFrame by the absolute values of PC1, PC2, PC3, and PC4
# sorted_loadings_df = loadings_df.apply(lambda x: abs(x)).sort_values(by=['PC1', 'PC2', 'PC3', 'PC4'], ascending=False)
# loadingOrder = sorted_loadings_df.index.tolist()

loadingOrder = (PC1_sorted_loadings.iloc[:6].index.tolist()
                + PC2_sorted_loadings.index.tolist())

loadings_df_melted = loadings_df.iloc[:,:2].reset_index().melt(id_vars='index', var_name='Principal Component', value_name='Loading')

# Create the heatmap using Altair
heatmap = alt.Chart(loadings_df_melted).mark_rect().encode(
    x=alt.X('index:O', title=None,
            axis=alt.Axis(labelLimit=0, bandPosition=0.5),
            sort=loadingOrder),
    y=alt.Y('Principal Component:O', title=None),
    color=alt.Color('Loading:Q',
                    scale=alt.Scale(scheme="redblue", reverse=True),
                    legend=alt.Legend(orient='top')),
```

```python
).properties(
    width=1000,
    height=60,
#     title='Loadings Heatmap'
).configure_axisX(
#     orient='top',  # Move x-axis labels to the top
    labelFontSize=14,
    labelAngle=45,  # Rotate x-axis labels by 45 degrees
    domain=False  # Remove axis lines
).configure_axisY(
    labelFontSize=14,
    domain=False  # Remove axis lines
).configure_legend(
    titleFontSize=14,  # Set the font size for the legend title
    labelFontSize=14  # Set the font size for the legend
)

# Display the heatmap
heatmap
```

Out[64]:



The heatmap above illustrates the loadings for the first two principal components (PC1 and PC2). It is evident that PC1 emphasizes customer interactions, such as reviews and tips, while PC2 emphasizes demographic features.

4.8.4. Project restaurant data onto PC1 and PC2 space

when visualizing all restaurants in PC1 and PC2 space, we observe that the restaurants are not distinguishable by cuisine.

In [65]:

```python
# project the average data by cuisine onto the principal components learned

cuisine_loadings = pd.DataFrame(columns=['PC1_loading','PC2_loading','PC3_loading','PC4_loading'],
                                index=cuisineOrder)

for cuisine in cuisineOrder:

    df = X.reset_index()
```

```
    idx = df[df.business_id.isin(master4[master4.cuisine==cuisine].business_id.unique())].index

    cuisine_scaled = np.mean(X_scaled[idx], axis=0).reshape(1, -1)

    # Transform the standardized data using PCA
    cuisine_pca = pca.transform(cuisine_scaled)

    # Loadings of the cuisine on PC1 through PC4
    cuisine_loadings.loc[cuisine,'PC1_loading'] = cuisine_pca[0, 0]
    cuisine_loadings.loc[cuisine,'PC2_loading'] = cuisine_pca[0, 1]
    cuisine_loadings.loc[cuisine,'PC3_loading'] = cuisine_pca[0, 2]
    cuisine_loadings.loc[cuisine,'PC4_loading'] = cuisine_pca[0, 3]

cuisine_loadings
```

Out[65]:

	PC1_loading	PC2_loading	PC3_loading	PC4_loading
Chinese	-0.838905	0.366151	-0.085617	0.09999
Italian	0.408616	-0.584672	-0.046201	-0.093956
Mexican	-0.119069	0.408831	0.153348	0.142428
Japanese	0.647971	-0.139186	-0.273309	-0.432566

In [66]:
```
# Extract PC1 and PC2 values for all records

pc1_values = restaurant_pca[:, 0]
pc2_values = restaurant_pca[:, 1]

PC1PC2_df = pd.DataFrame({'PC1':pc1_values, 'PC2':pc2_values})

PC1PC2_df.shape
```

Out[66]: (12005, 2)

In [67]:
```
PC1PC2_df = PC1PC2_df.merge(master4[['cuisine']], how='left', left_index=True, right_index=True)
PC1PC2_df.head()
```

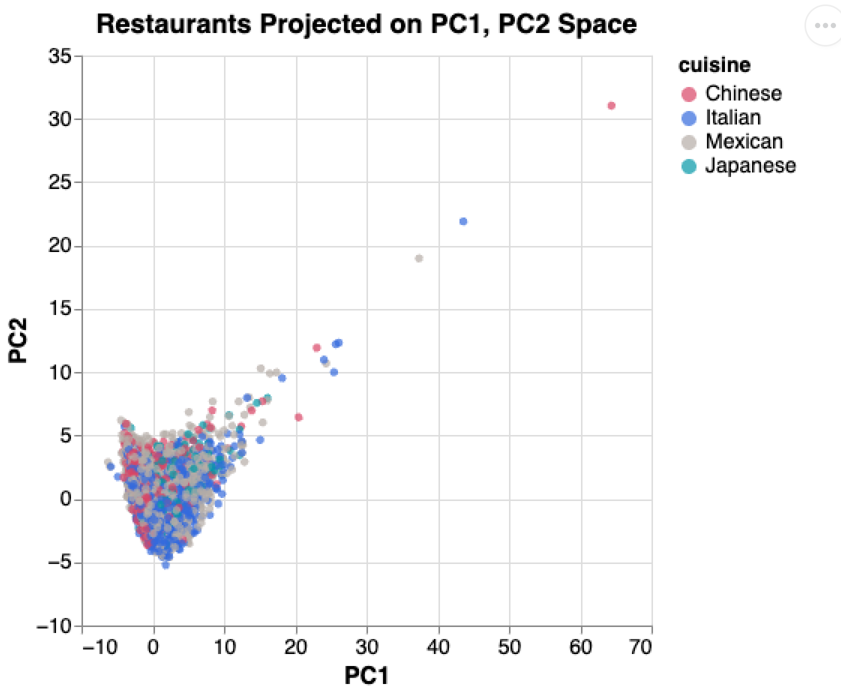
Out[67]:

	PC1	PC2	cuisine
0	1.294531	-1.059477	Japanese
1	1.084277	-0.417732	Japanese
2	-0.337177	-0.370560	Chinese
3	-2.580549	1.479077	Chinese
4	0.007207	0.100586	Japanese

In [68]:
```
# PC1 and PC2 alone cannot seperate restaurants with different cuisines

alt.Chart(PC1PC2_df).mark_circle().encode(
    x=alt.X('PC1:Q'),
```

```
    y=alt.Y('PC2:Q'),
    color=alt.Color('cuisine:N', scale =alt.Scale(domain=cuisineOrder,
                        range=[colors[0], colors[2], colors[3], colors[4]]))
).properties(
    title={
        'text': ['Restaurants Projected on PC1, PC2 Space'],
        'fontSize': 18  # Adjust the font size as needed
    },
    width=380,
    height=380,
).configure_axisX(
    labelFontSize=14,
    titleFontSize=16
).configure_axisY(
    labelFontSize=14,
    titleFontSize=16
).configure_legend(
    titleFontSize=14,  # Set the font size for the legend title
    labelFontSize=14  # Set the font size for the legend
)
```

Out[68]:

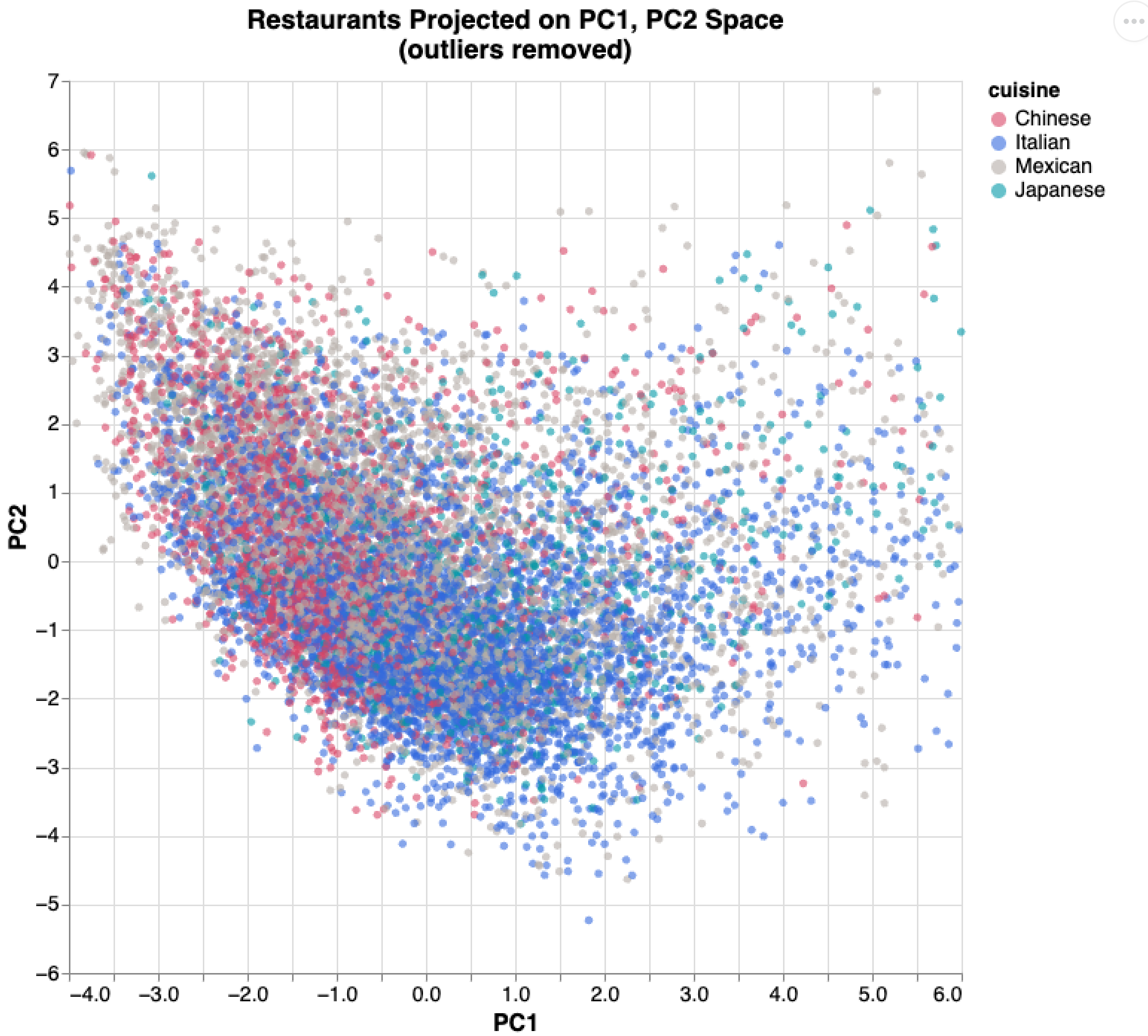


In [69]:

```
# Zooming in - still - PC1 and PC2 alone cannot seperate restaurants with different cuisines

alt.Chart(PC1PC2_df[PC1PC2_df.PC1.between(-8,10) & PC1PC2_df.PC1.between(-4,6)]).mark_circle(opacity=0.6).encode(
    x=alt.X('PC1:Q'),
    y=alt.Y('PC2:Q'),
    color=alt.Color('cuisine:N', scale =alt.Scale(domain=cuisineOrder,
                        range=[colors[0], colors[2], colors[3], colors[4]]))
```

```
).properties(
    title={
        'text': ['Restaurants Projected on PC1, PC2 Space','(outliers removed)'],
        'fontSize': 18  # Adjust the font size as needed
    },
    width=600,
    height=600,
).configure_axisX(
    labelFontSize=14,
    titleFontSize=16
).configure_axisY(
    labelFontSize=14,
    titleFontSize=16
).configure_legend(
    titleFontSize=14,  # Set the font size for the legend title
    labelFontSize=14  # Set the font size for the legend
)
```

Out[69]:



4.8.5. Project cuisine onto PC1 and PC2 space

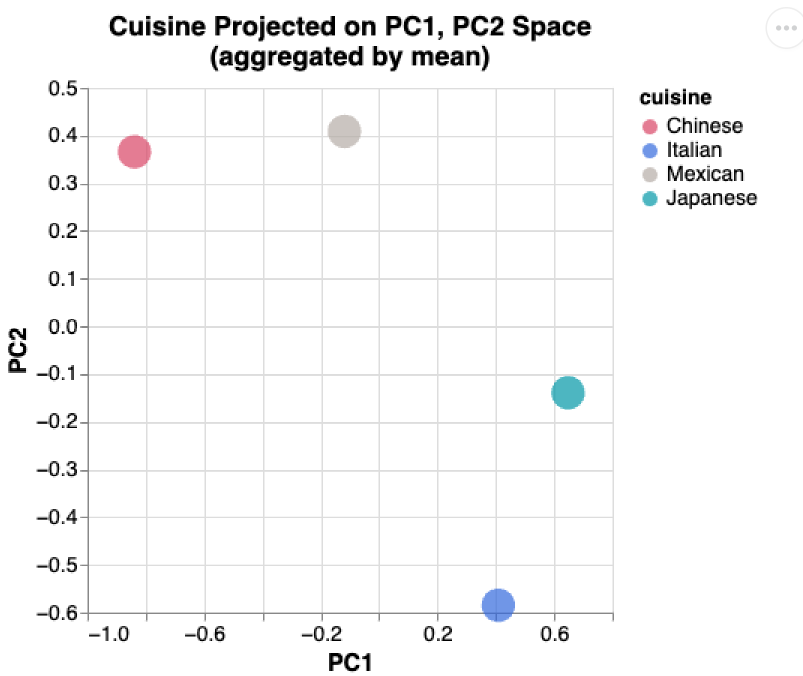
If we aggregate restaurants to the cuisine level by computing the average of all features, and then visualize the cuisines in PC1 and PC2 space while considering the loadings, we can better understand the main differences between cuisines.

Positive PC2 values suggest that Chinese and Mexican cuisines are predominantly situated in neighborhoods characterized by lower income, education levels, average age, and a higher Hispanic-Latino population. Additionally, negative PC1 values indicate that both cuisines tend to display lower levels of customer engagement, with Chinese cuisine showing a more pronounced low level.

Conversely, negative PC2 values suggest that both Italian and Japanese cuisines are typically found in communities with higher income, education, average age, and a higher White population, but a lower Hispanic-Latino population, with Italian cuisine being more extreme in this regard. Additionally, positive PC1 values indicate that they both demonstrate higher levels of customer engagement, although Japanese cuisine tends to exhibit even higher levels.

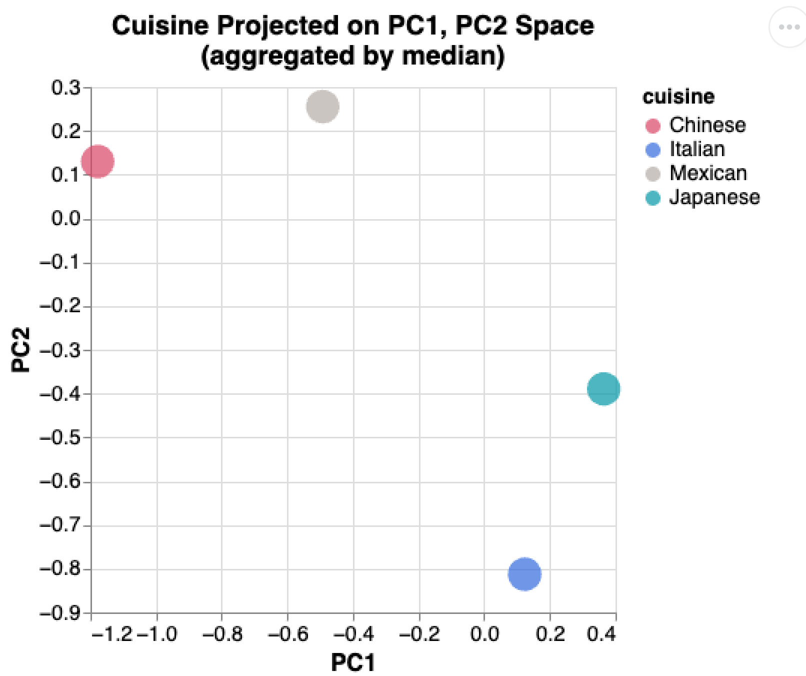
In [70]:
```
# Project each Cuisine (using mean to aggregate data) on two-dimention (PC1, PC2) space

_df = PC1PC2_df.groupby('cuisine').mean().reset_index()

alt.Chart(_df).mark_circle(size=500).encode(
    x=alt.X('PC1:Q'),
    y=alt.Y('PC2:Q'),
    color=alt.Color('cuisine:N', scale =alt.Scale(domain=cuisineOrder,
                        range=[colors[0], colors[2], colors[3], colors[4]]))
).properties(
    title={
        'text': ['Cuisine Projected on PC1, PC2 Space','(aggregated by mean)'],
        'fontSize': 18  # Adjust the font size as needed
    },
    width=350,
    height=350,
).configure_axisX(
    labelFontSize=14,
    titleFontSize=16
).configure_axisY(
    labelFontSize=14,
    titleFontSize=16
).configure_legend(
    titleFontSize=14,  # Set the font size for the legend title
    labelFontSize=14  # Set the font size for the legend
)
```

Out[70]:



In [71]:
```
# Project each Cuisine (using median to aggregate data) on two-dimention (PC1 and PC2) space

_df = PC1PC2_df.groupby('cuisine').median().reset_index()

alt.Chart(_df).mark_circle(size=500).encode(
    x=alt.X('PC1:Q'),
    y=alt.Y('PC2:Q'),
    color=alt.Color('cuisine:N', scale =alt.Scale(domain=cuisineOrder,
                    range=[colors[0], colors[2], colors[3], colors[4]]))
).properties(
    title={
        'text': ['Cuisine Projected on PC1, PC2 Space','(aggregated by median)'],
        'fontSize': 18  # Adjust the font size as needed
    },
    width=350,
    height=350,
).configure_axisX(
    labelFontSize=14,
    titleFontSize=16
).configure_axisY(
    labelFontSize=14,
    titleFontSize=16
).configure_legend(
    titleFontSize=14,  # Set the font size for the legend title
    labelFontSize=14  # Set the font size for the legend
)
```

Out[71]:



Appendix

In [72]:
```
# # Reshape the DataFrame for Altair

# # Sort the DataFrame by the absolute values of PC1, PC2, PC3, and PC4
# # sorted_loadings_df = loadings_df.apply(lambda x: abs(x)).sort_values(by=['PC1', 'PC2', 'PC3', 'PC4'], ascending=False)
# # loadingOrder = sorted_loadings_df.index.tolist()

# loadingOrder = (PC1_sorted_loadings.iloc[:6].index.tolist()
#                 + PC2_sorted_loadings.iloc[:6].index.tolist()
#                 + PC3_sorted_loadings.iloc[:6].index.tolist()
#                 + PC4_sorted_loadings.iloc[:6].index.tolist())

# loadings_df_melted = loadings_df.reset_index().melt(id_vars='index', var_name='Principal Component', value_name='Loading')

# # Create the heatmap using Altair
# heatmap = alt.Chart(loadings_df_melted).mark_rect().encode(
#     x=alt.X('index:O', title=None,
#             axis=alt.Axis(labelLimit=0, bandPosition=0.5),
#             sort=loadingOrder),
#     y=alt.Y('Principal Component:O', title=None),
#     color=alt.Color('Loading:Q',
#                     scale=alt.Scale(scheme="redblue", reverse=True),
#                     legend=alt.Legend(orient='top')),
# ).properties(
#     width=800,
#     height=100,
# #     title='Loadings Heatmap'
# ).configure_axisX(
# #     orient='top',  # Move x-axis labels to the top
#     labelFontSize=14,
#     labelAngle=45,  # Rotate x-axis labels by 45 degrees
#     domain=False  # Remove axis lines
# ).configure_axisY(
#     labelFontSize=14,
#     domain=False  # Remove axis lines
# ).configure_legend(
#     titleFontSize=14,  # Set the font size for the legend title
#     labelFontSize=14  # Set the font size for the legend
# )

# # Display the heatmap
# heatmap
```

In [73]:
```
# Biplot
def biplot(score, coeff, labels=None):
    plt.figure(figsize=(10, 10))
    xs = score[:, 0]
    ys = score[:, 1]
    n = coeff.shape[0]

    for i in range(n):
        plt.arrow(0, 0, coeff[i, 0], coeff[i, 1], color='r', alpha=0.8)
```

```
        if labels is not None:
            plt.text(coeff[i, 0] * 1.15, coeff[i, 1] * 1.15, labels[i], color='g', ha='center', va='center')

    plt.scatter(xs, ys, c='b', marker='o', alpha=0.5)
    plt.xlabel("PC1")
    plt.ylabel("PC2")
    plt.grid(True)
    plt.show()
```

In [74]:
```
# biplot(restaurant_pca[:, :2], np.transpose(pca.components_[:2, :]), labels=X.columns)
```

In [75]:
```
# # too slow to render at zip code level

# # Zip code boundaries: https://hub.arcgis.com/datasets/d6f7ee6129e241cc9b6f75978e47128b/explore


# # Load GeoJSON data from a local file (replace 'path/file.geojson' with the actual file path)

# with open('data/USA_ZIP_Code_Boundaries.geojson', 'r') as f:
#     geojson_data = json.load(f)

# # Create a GeoDataFrame from the GeoJSON data
# gdf = gpd.GeoDataFrame.from_features(geojson_data['features'])


# # Create the choropleth map
# chart = alt.Chart(gdf).mark_geoshape().encode(
#     alt.Color('average_rating:Q', title='Pcnt Chinese'),
# #     tooltip=['properties.ZIP:O', 'average_rating:Q', 'average_sentiment:Q']
# ).transform_lookup(
#     lookup='properties.ZIP',
#     from_=alt.LookupData(df, 'zip_code', ['average_rating'])
# ).project(
#     'identity'
# ).properties(
#     width=800,
#     height=500
# )

# chart
```

In [76]:
```
# # layer them before faceting
# alt.Chart(master4[['cuisine', 'stars']]).transform_density(
#     'stars',
#     as_=['stars', 'density'],
# #     extent=[5, 50],
#     groupby=['cuisine']
# ).mark_area(orient='horizontal').encode(
#     y='stars:Q',
#     color='cuisine:N',
#     x=alt.X(
#         'density:Q',
#         stack='center',
#         impute=None,
#         title=None,
```

```
#         axis=alt.Axis(labels=False, values=[0],grid=False, ticks=True),
#     ),
#     column=alt.Column(
#         'cuisine:N',
#         header=alt.Header(
#             titleOrient='bottom',
#             labelOrient='bottom',
#             labelPadding=0,
#         ),
#     )
# ).properties(
#     width=100
# ).configure_facet(
#     spacing=0
# ).configure_view(
#     stroke=None
# )
```