Recommendation System Project Group # 4

1. Shravan Kumar Singh
2. Gurjeet Singh
3. Deepika Sinha

(Website link:https://shravanksingh.github.io/CS109a_Recommendation_System_Project_Group_4/ (https://shravanksingh.github.io/CS109a_Recommendation_System_Project_Group_4/)

Loading Libraries

In [21]:
```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegressionCV
import sklearn.metrics as metrics
from sklearn.preprocessing import PolynomialFeatures
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn import tree
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc, roc_auc_score
import json
from sklearn.tree import export_graphviz
from IPython.display import Image
from IPython.display import display
from IPython.display import display, Math, Latex
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')

pd.set_option('display.width', 450)
pd.set_option('display.max_columns', 100)
pd.set_option('display.notebook_repr_html', True)
import seaborn.apionly as sns
sns.set_style("whitegrid")

c0=sns.color_palette()[0]
c1=sns.color_palette()[1]
c2=sns.color_palette()[2]
```

Loading Data via function line by line

As we have large amounf data so we are loading data line by line in dataframe business_df, review_df, user_df

In [22]:

```
import json

def readjson(filepath):
    data = []
    i=0
    with open(filepath,encoding="utf8") as f:
            for line in f:
                 if i<100000:
                    data.append(json.loads(line))
                    #print(i)
                    i +=1
    return pd.DataFrame(data)

business_df = readjson('./dataset/business.json')
review_df = readjson('./dataset/review.json')
user_df = readjson('./dataset/user.json')
```

Filtering data

Getting reaturants out of business dataframe based on Food category

In [23]:

```
business_df['categories'] = business_df['categories'].astype(str)
restaurant_df = business_df[business_df['categories'].str.contains('Food')==True]

complete_df = restaurant_df.merge(review_df,on='business_id').merge(user_df,on='user_id')
```

```
In [24]: complete_df.head(2)
```

Out[24]:

	address	attributes	business_id	categories	city	hours	is_open	latitude	longitude	r
0	1203 E Charleston Blvd, Ste 140	{'BusinessParking': {'validated': False, 'gara...	YTqtM2WFhcMZGeAGA08Cfg	['Seafood', 'Restaurants', 'Specialty Food', '...	Las Vegas	{'Sunday': '10:15-21:00', 'Wednesday': '10:30-...	1	36.159363	-115.135949	Ma Pla Esc
1	1203 E Charleston Blvd, Ste 140	{'BusinessParking': {'validated': False, 'gara...	YTqtM2WFhcMZGeAGA08Cfg	['Seafood', 'Restaurants', 'Specialty Food', '...	Las Vegas	{'Sunday': '10:15-21:00', 'Wednesday': '10:30-...	1	36.159363	-115.135949	Ma Pla Esc

```
In [25]: restaurant_df.describe()
```

Out[25]:

	is_open	latitude	longitude	review_count	stars
count	18503.00000	18503.000000	18503.000000	18503.000000	18503.000000
mean	0.83073	39.702568	-87.807760	34.804464	3.546857
std	0.37500	5.747548	27.691971	82.946472	0.889710
min	0.00000	-34.520401	-119.551325	3.000000	1.000000
25%	1.00000	35.135615	-112.013439	5.000000	3.000000
50%	1.00000	40.440368	-81.357777	11.000000	3.500000
75%	1.00000	43.665419	-79.414244	31.000000	4.000000
max	1.00000	59.438181	11.769500	3439.000000	5.000000

In [26]:
```
user_df.describe()
```

Out[26]:

	average_stars	compliment_cool	compliment_cute	compliment_funny	compliment_hot	compliment_list	compliment_more	c
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	10
mean	3.729684	16.342210	0.950070	16.342210	12.015470	0.416970	1.465460	6.
std	0.835715	197.424646	16.639768	197.424646	175.458886	7.165452	15.762362	70
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	3.350000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
50%	3.810000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
75%	4.240000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.
max	5.000000	16710.000000	2146.000000	16710.000000	19988.000000	1265.000000	1576.000000	63

In [27]:
```
review_df.describe()
```

Out[27]:

	cool	funny	stars	useful
count	100000.000000	100000.000000	100000.000000	100000.00000
mean	0.532470	0.411740	3.730530	1.01213
std	1.992121	1.655608	1.418456	2.46252
min	0.000000	0.000000	1.000000	0.00000
25%	0.000000	0.000000	3.000000	0.00000
50%	0.000000	0.000000	4.000000	0.00000
75%	0.000000	0.000000	5.000000	1.00000
max	104.000000	114.000000	5.000000	113.00000

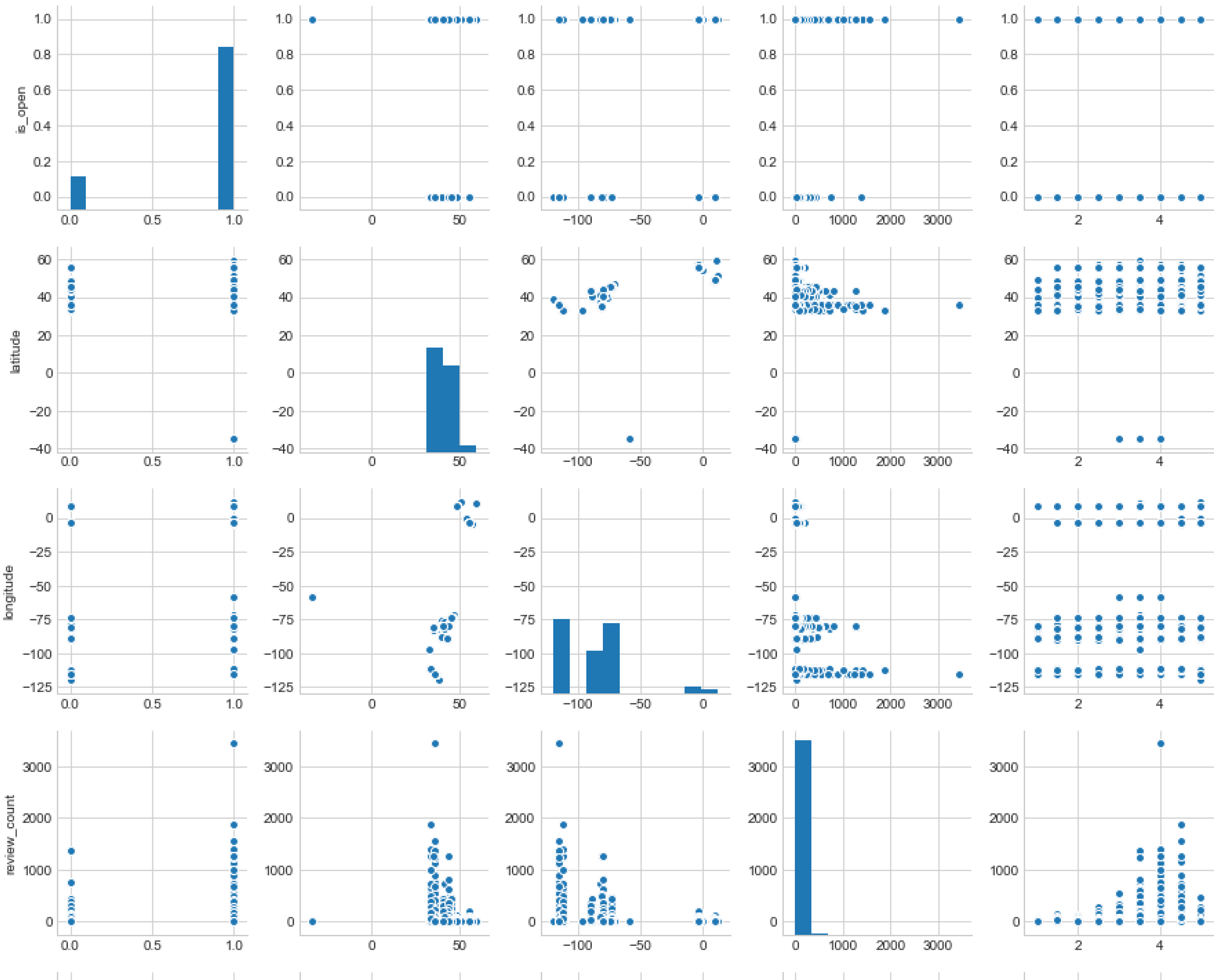
In [28]:
```
review_df.head(2)
```
Out[28]:

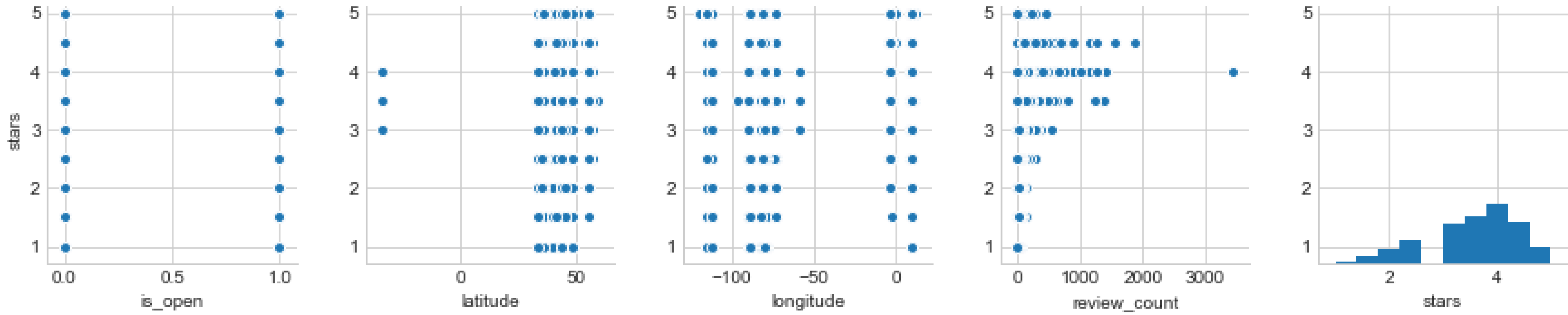
	business_id	cool	date	funny	review_id	stars	text	useful	user_id
0	uYHaNptLzDLoV_JZ_MuzUA	0	2016-07-12	0	VfBHSwC5Vz_pbFluy07i9Q	5	My girlfriend and I stayed here for 3 nights a...	0	cjpdDjZyprfyDG3RlkVG3w
1	uYHaNptLzDLoV_JZ_MuzUA	0	2016-10-02	0	3zRpneRKDsOPq92tq7ybAA	3	If you need an inexpensive place to stay for a...	0	bjTcT8Ty4cJZhEOEo01FGA

EDA

Performing Exploratory data analysis

In [30]:
```
sns.pairplot(restaurant_df.iloc[0:10000,:]);
```

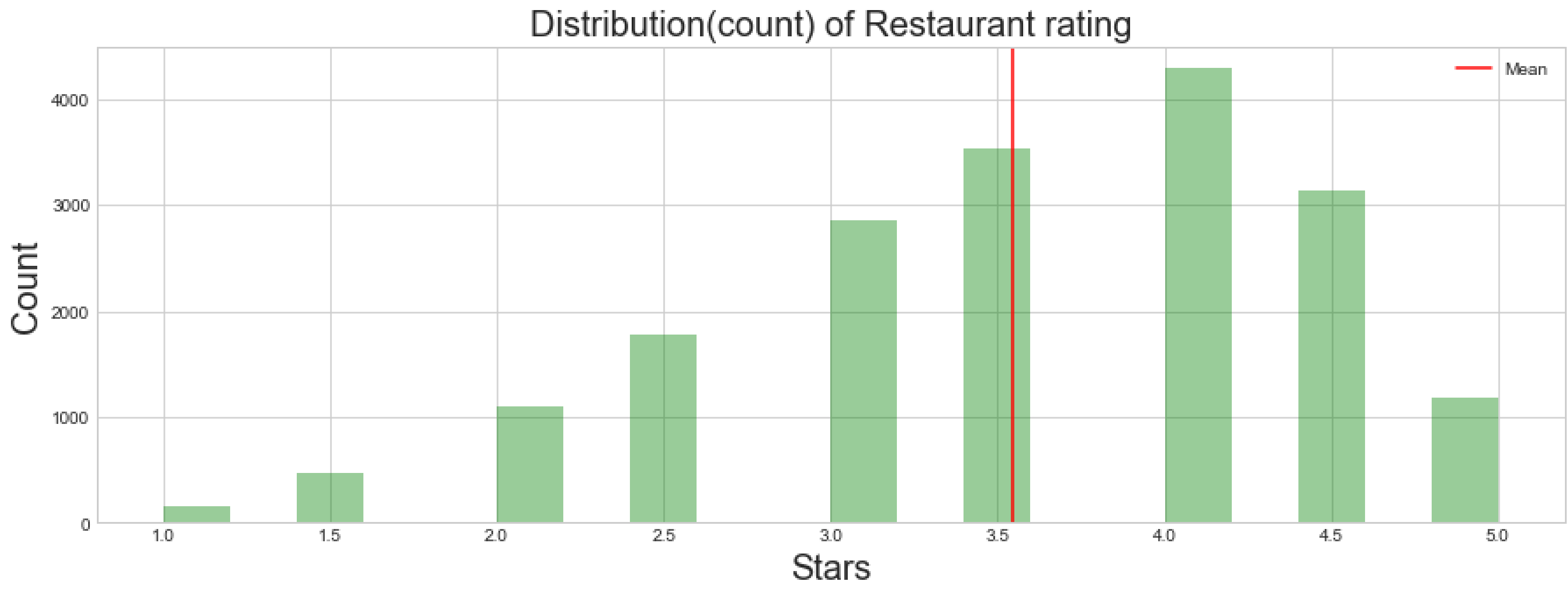

is_open
latitude
longitude
review_count



Distribution count of Restaurant rating

We can see below more restaurants get 4 rating than other ratings

In [31]:
```
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(15, 5))

sns.distplot(restaurant_df.stars,kde=False,color = 'g',ax =ax,bins=20);
ax.axvline(restaurant_df.stars.mean(), 0, 1, color='r', label='Mean')
ax.legend();
ax.set_ylabel('Count',size=20)
ax.set_xlabel('Stars',size=20)
ax.set_title('Distribution(count) of Restaurant rating',size=20);
```



Distribution count of Reviews rating for restaurants

We can see below more reviews have 5 rating than other ratings

In [32]:
```
#review just for business which are restautrant
review_df_filter_df = review_df.merge(restaurant_df,how='inner',on='business_id')

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(15, 5))
sns.distplot(review_df_filter_df.stars_x,kde=False,color = 'g',ax =ax,bins=20);
ax.axvline(review_df_filter_df.stars_x.mean(), 0, 1, color='r', label='Mean')
ax.legend();
ax.set_ylabel('Count',size=20)
ax.set_xlabel('Stars',size=20)
ax.set_title('Distribution(count) of different Reviews rating',size=20)
```

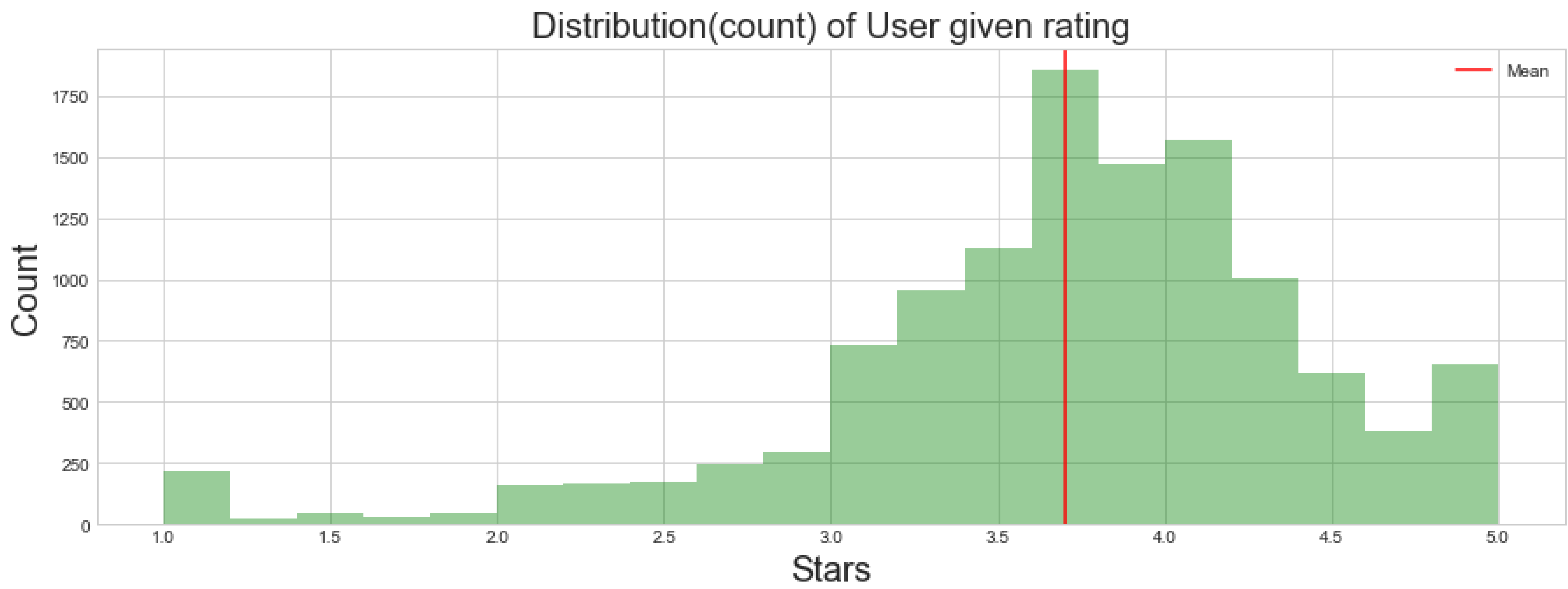
Out[32]:
```
Text(0.5,1,'Distribution(count) of different Reviews rating')
```

Distribution count of user rating for restaurants

We can see below users have around mean of 3.7 rating

In [33]:
```
#user just for business which are restautrant
user_df_filter_df = complete_df.groupby(['user_id'],as_index=False).mean()

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(15, 5))
sns.distplot(user_df_filter_df.average_stars,kde=False,color = 'g',ax =ax,bins=20);
ax.axvline(user_df_filter_df.average_stars.mean(), 0, 1, color='r', label='Mean')
ax.legend();
ax.set_ylabel('Count',size=20)
ax.set_xlabel('Stars',size=20)
ax.set_title('Distribution(count) of User given rating',size=20)

#fig.tight_layout()
```

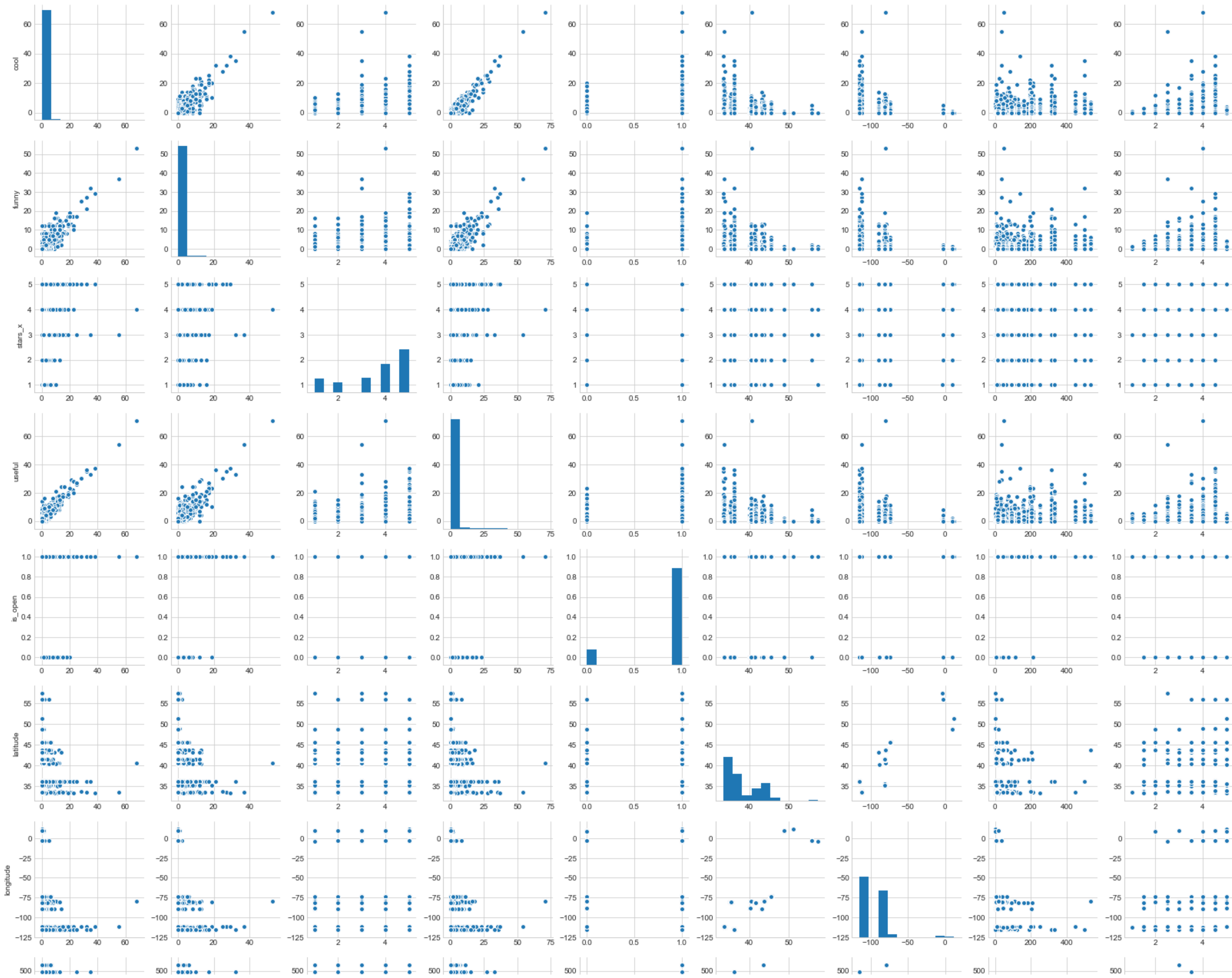
Out[33]: Text(0.5,1,'Distribution(count) of User given rating')

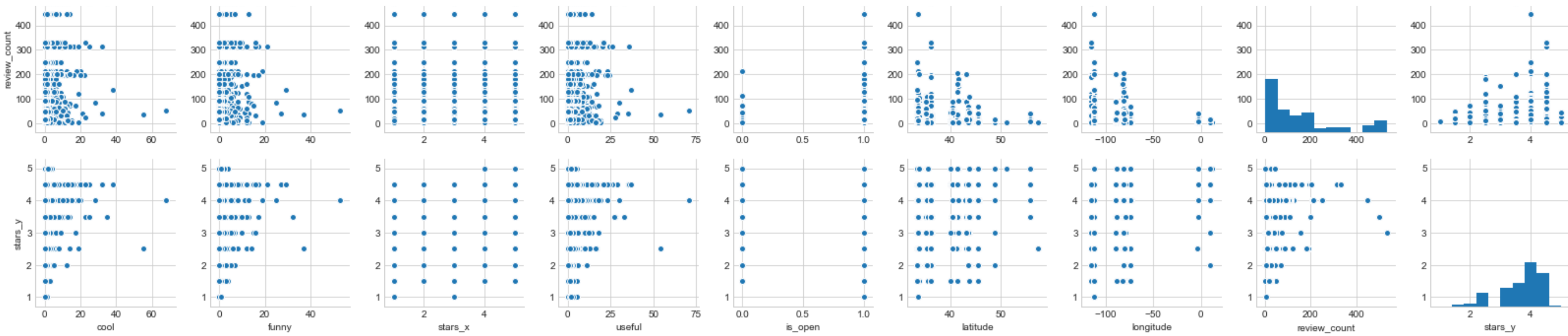


Scatter plot various features

We can see that useful, funny and cool are correlated

In [34]:
```
sns.pairplot(review_df_filter_df.iloc[0:10000,:]);
```

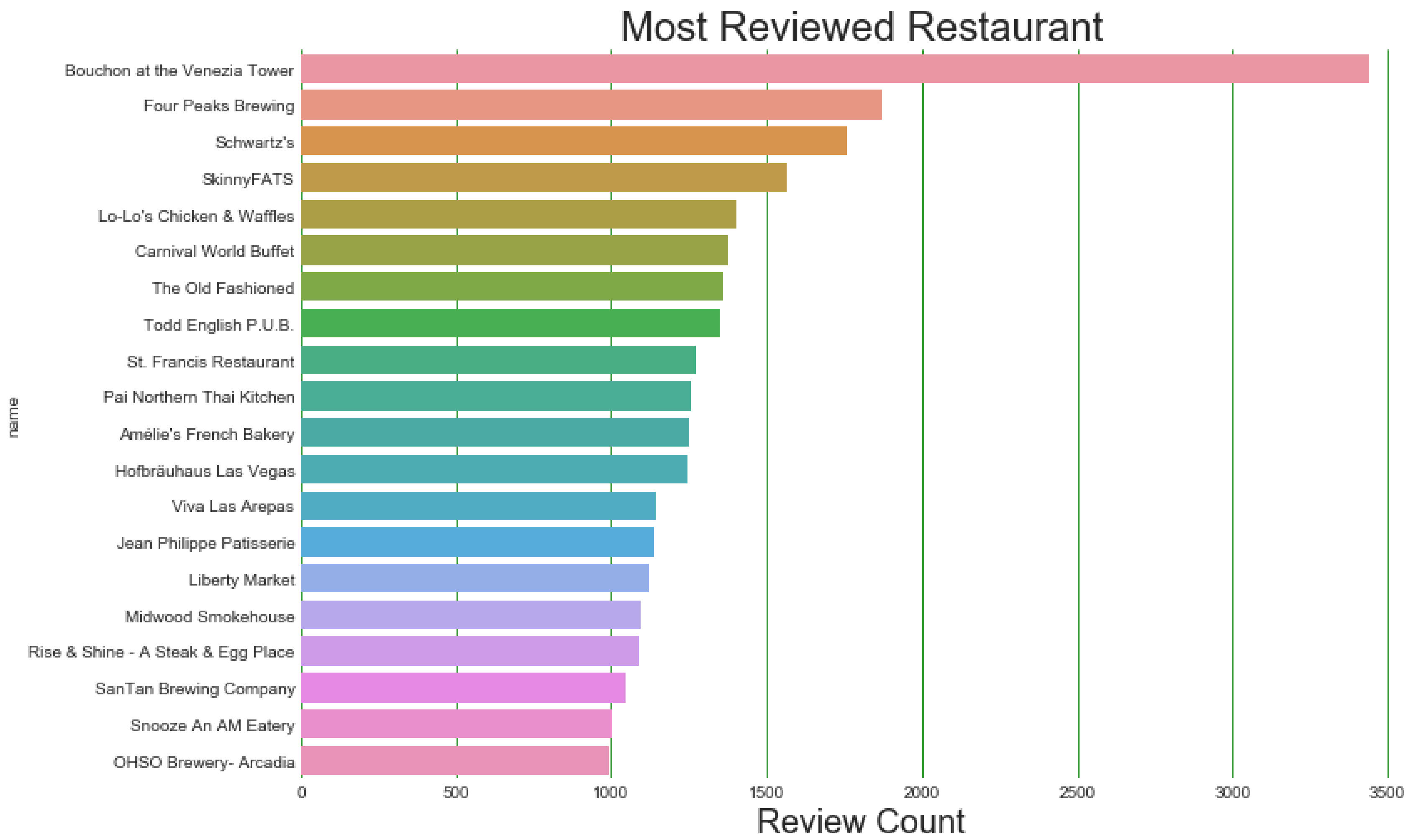

cool
funny
stars_x
useful
is_open
latitude
longitude



Most Reviewed Restaurant

Bouchon at the Venezia Tower is reviewed almost double as compared to others

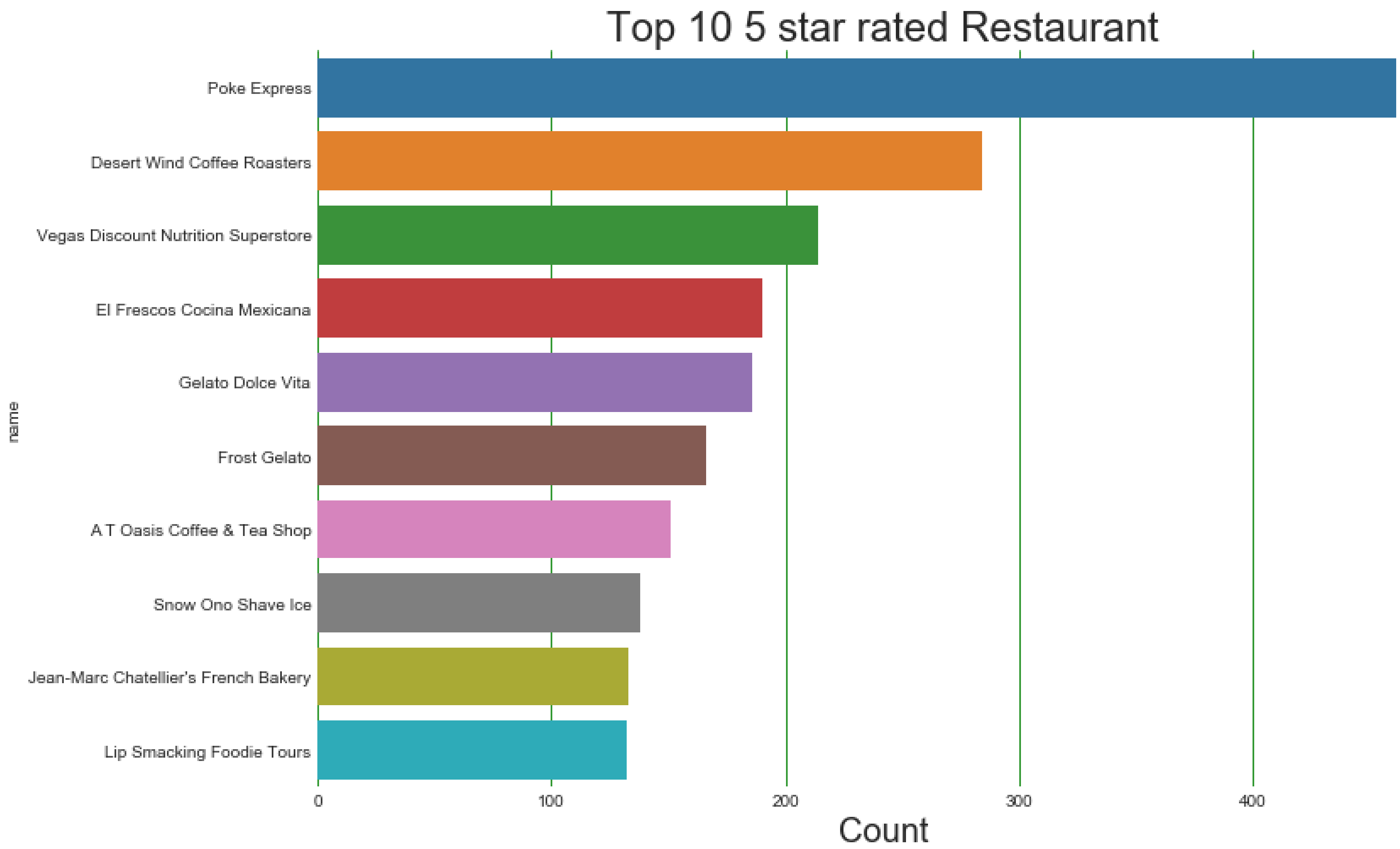
In [35]:
```
#get top 20 most reviewed restaurants
n_top =20
most_reviewed_restaurant = restaurant_df.nlargest(n_top, 'review_count')
fig, ax = plt.subplots()
ax = sns.barplot(y="name", x="review_count", data=most_reviewed_restaurant)
ax.set_xlabel('Review Count',size=20)
fig.set_size_inches(12, 8)
plt.title("Most Reviewed Restaurant",fontsize=24);
ax.grid(axis = 'x', color ='green', linestyle='-')
ax.tick_params(axis='both', which='both',length=0)
sns.despine(left=True, bottom=True)
```



Top 10 5 star rated Restaurant

Poke Express is the top 5 star rated restaurant

In [36]:
```
top_rated_restaurant = restaurant_df.sort_values(by=['stars','review_count'],
                                                 ascending=False)[['name','business_id','review_count','star
s']]
#get top 10 5 star rated restaurant
n_top =10
top_rated_restaurant = top_rated_restaurant.nlargest(n_top, 'stars')
fig, ax = plt.subplots()
ax = sns.barplot(y="name", x="review_count", data=top_rated_restaurant)
ax.set_xlabel('Count',size=20)
fig.set_size_inches(12, 8)
plt.title("Top 10 5 star rated Restaurant",fontsize=24);
ax.grid(axis = 'x', color ='green', linestyle='-')
ax.tick_params(axis='both', which='both',length=0)
sns.despine(left=True, bottom=True)
```



Getting different food categories from the restaurant dataframe

In [37]:
```
top_rated_restaurant = restaurant_df.sort_values(by=['stars','review_count'],
                                                  ascending=False)[['name','business_id','review_count','star
s']]
#top_rated_restaurant
```

In [38]:
```
def get_food_type_count(category):
    count = restaurant_df[restaurant_df['categories'].str.contains(category)==True]['business_id'].count()
    return count
```

In [39]:
```
food_dict = {}
food_categories = ['American','Italian','Mexican','Chinese','Thai','Indian','Japan','French']
for food_category in food_categories:
    food_dict[food_category] = get_food_type_count(food_category)
```

Distribution of review count with respect to Food Categories

We can see American restaurant have higher count of reviews followed by Mexican

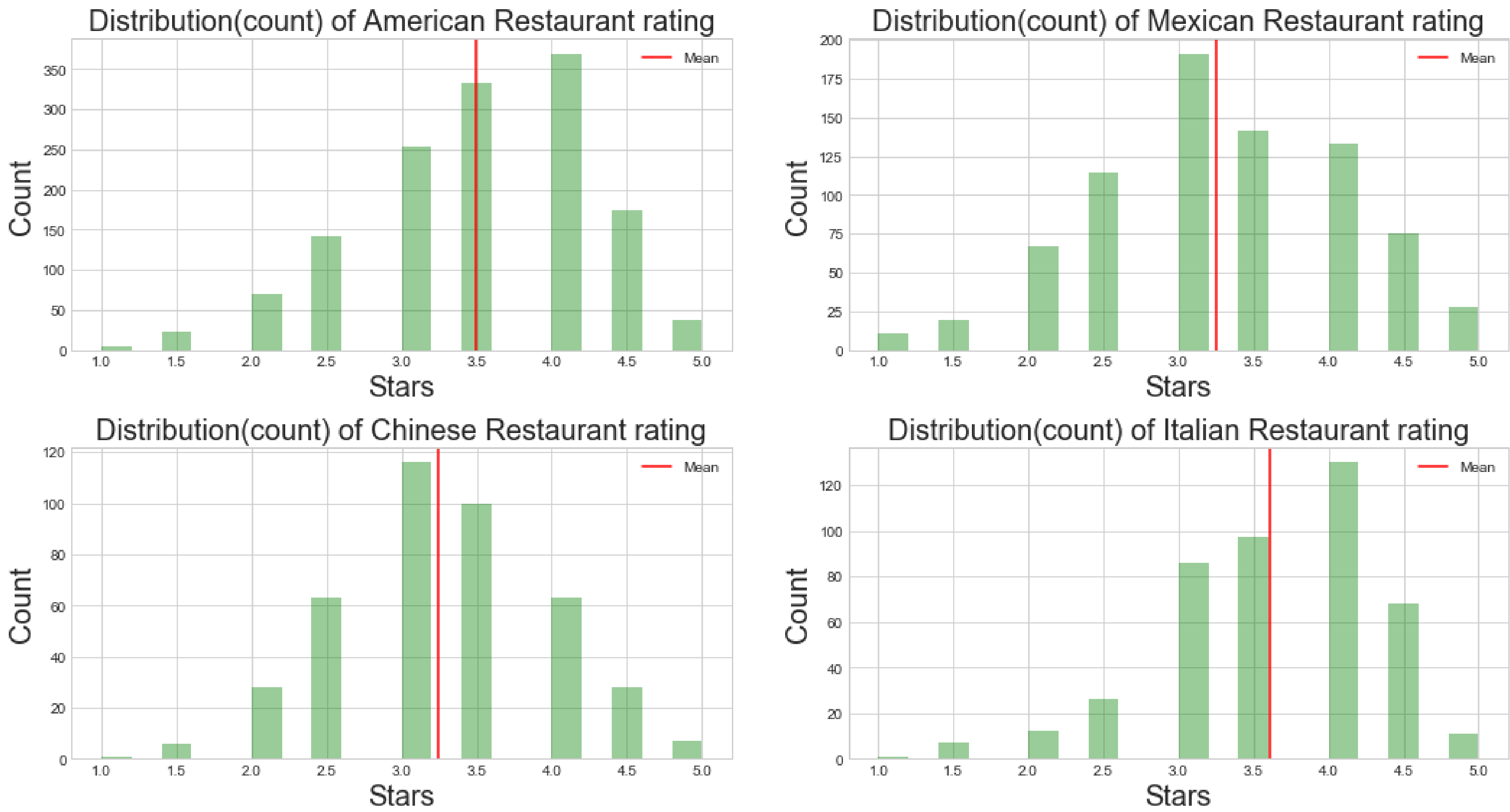
In [40]:
```
plt.figure(figsize=(20,10))
plt.bar(range(len(food_dict)), food_dict.values(), align='center',color='forestgreen')
plt.xticks(range(len(food_dict)), list(food_dict.keys()),fontsize = 15);
plt.title('Distribution of review count with respect to Food Categories',fontsize=18)
plt.xlabel('Food Category',fontsize=18)
plt.ylabel('Count',fontsize=18)
```

Out[40]: Text(0,0.5,'Count')

Distribution(count) of American, Mexican, Italian, Chinese Restaurant rating

We can see American and Italian restaurants are rated higher than other restaurants

In [41]:
```
American_restaurant_rating_df = restaurant_df[restaurant_df['categories'].str.contains('American')==True][['b
usiness_id','stars','categories','name','review_count']]
Mexican_restaurant_rating_df = restaurant_df[restaurant_df['categories'].str.contains('Mexican')==True][['bus
iness_id','stars','categories','name','review_count']]
Chinese_restaurant_rating_df = restaurant_df[restaurant_df['categories'].str.contains('Chinese')==True][['bus
iness_id','stars','categories','name','review_count']]
Italian_restaurant_rating_df = restaurant_df[restaurant_df['categories'].str.contains('Italian')==True][['bus
iness_id','stars','categories','name','review_count']]

fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(15, 8))
ax = ax.ravel()

def restaurant_category(df, title, ax):

    sns.distplot(df.stars,kde=False,color = 'g',ax =ax,bins=20);
    ax.axvline(df.stars.mean(), 0, 1, color='r', label='Mean')
    ax.legend();
    ax.set_ylabel('Count',size=20)
    ax.set_xlabel('Stars',size=20)
    ax.set_title('Distribution(count) of '+ title + ' Restaurant rating',size=20);

restaurant_category(American_restaurant_rating_df, 'American', ax[0])
restaurant_category(Mexican_restaurant_rating_df, 'Mexican', ax[1])
restaurant_category(Chinese_restaurant_rating_df, 'Chinese', ax[2])
restaurant_category(Italian_restaurant_rating_df, 'Italian', ax[3])

plt.tight_layout()
```



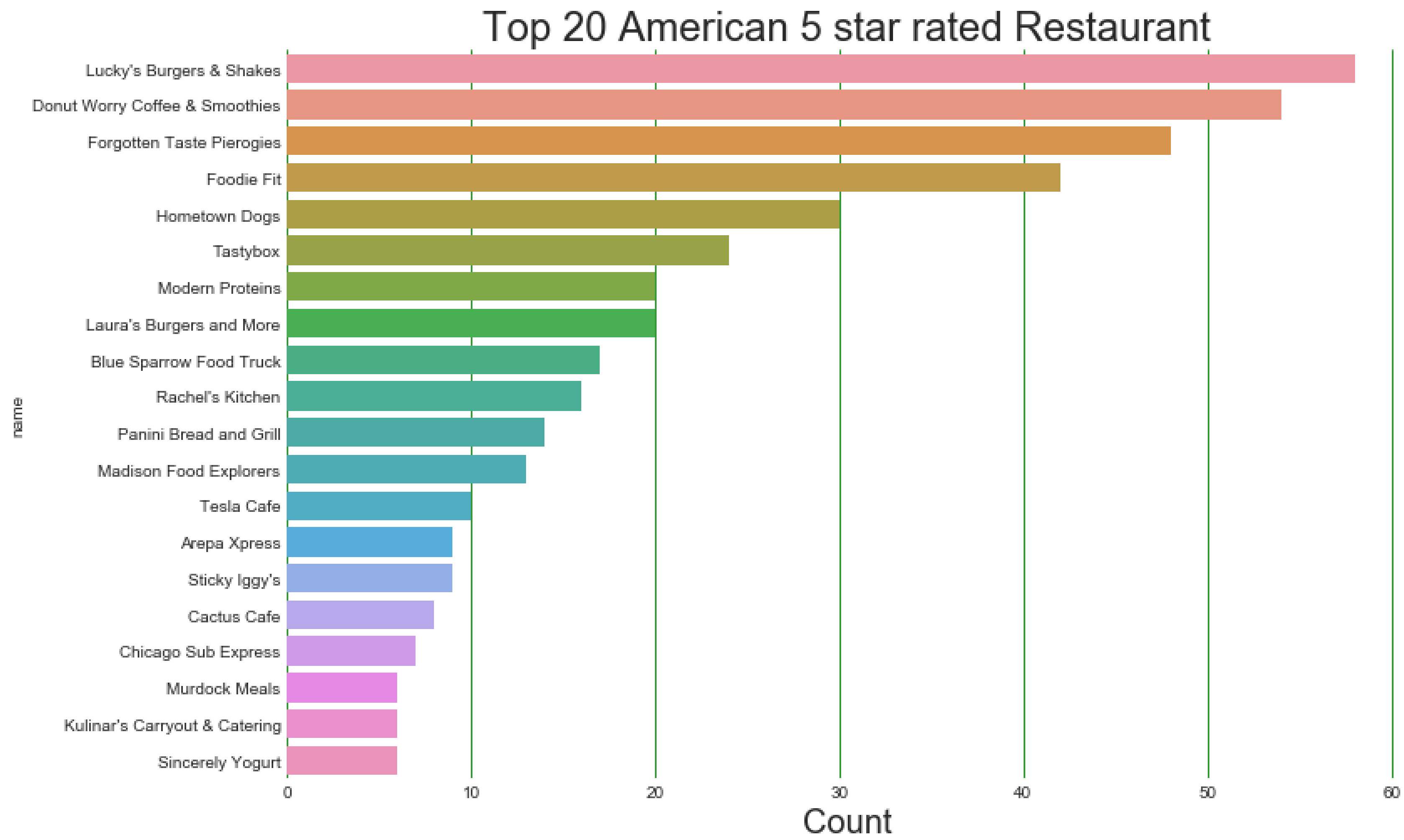
In [42]:
```
American_restaurant_rating_df.head(2)
```

Out[42]:

	business_id	stars	categories	name	review_count
34	reWc1g65PNZnKz_Ub9QKOQ	2.5	['Comfort Food', 'Canadian (New)', 'Restaurant...	Milestones Restaurants	51
55	Z1r6b30Tg0n0ME4-Zj2wQQ	3.0	['American (Traditional)', 'Restaurants', 'Bar...	Boardwalk Place	13

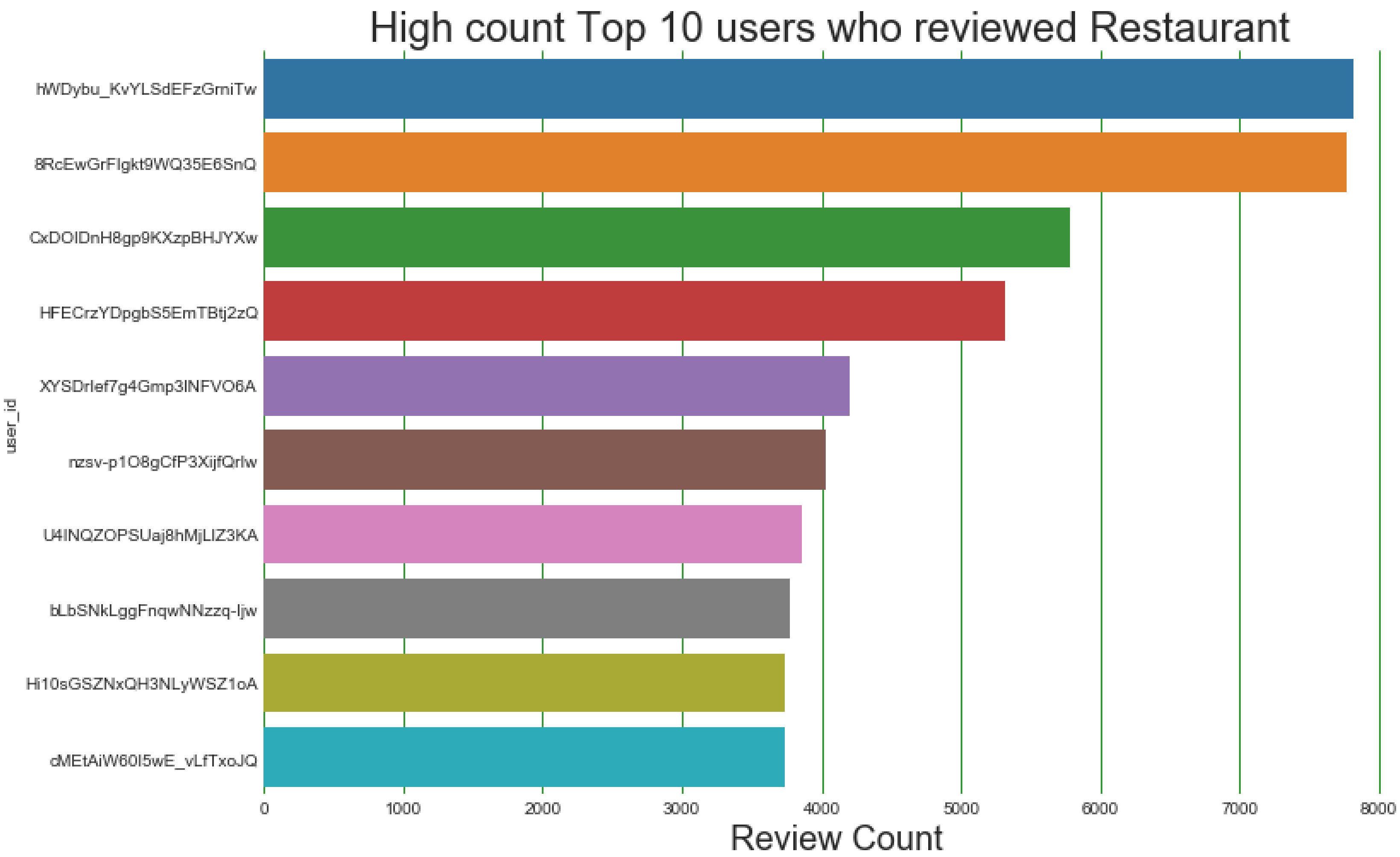
Top 20 American 5 star rated Restaurant

In [43]:
```
American_top_rated_restaurant = American_restaurant_rating_df.sort_values(by=['stars','review_count'],
                                                                ascending=False)[['name','business_id','review_count','star
s']]
#get top 20 5 star rated restaurant
n_top =20
American_top_rated_restaurant = American_top_rated_restaurant.nlargest(n_top, 'stars')
fig, ax = plt.subplots()
ax = sns.barplot(y="name", x="review_count", data=American_top_rated_restaurant)
ax.set_xlabel('Count',size=20)
fig.set_size_inches(12, 8)
plt.title("Top 20 American 5 star rated Restaurant",fontsize=24);
ax.grid(axis = 'x', color ='green', linestyle='-')
ax.tick_params(axis='both', which='both',length=0)
sns.despine(left=True, bottom=True)
```



High-count Top 10 users who reviewed Restaurant

In [44]:
```
#get top 10 most reviewing users
n_top =10
most_review_user = user_df_filter_df.nlargest(n_top, 'review_count_y').reindex()
fig, ax = plt.subplots()
ax = sns.barplot(y="user_id", x="review_count_y", data=most_review_user)
ax.set_xlabel('Review Count',size=20)
fig.set_size_inches(12, 8)
plt.title("High count Top 10 users who reviewed Restaurant ",fontsize=24);
ax.grid(axis = 'x', color ='green', linestyle='-')
ax.tick_params(axis='both', which='both',length=0)
sns.despine(left=True, bottom=True)
```

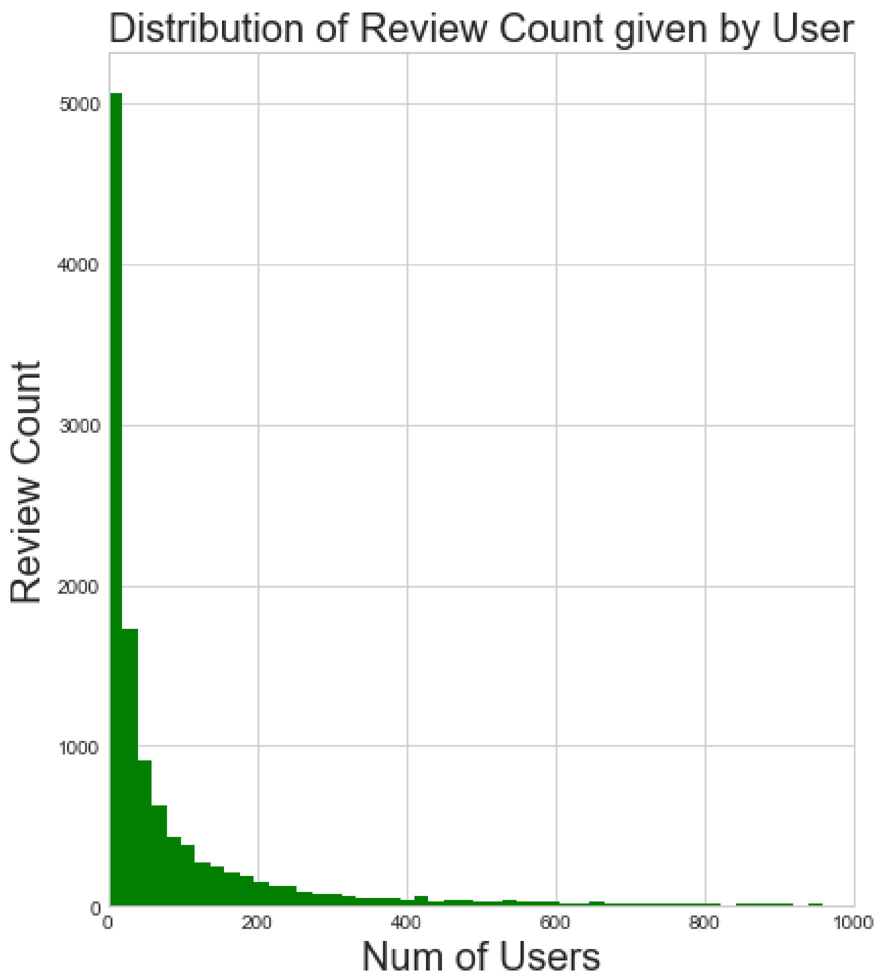


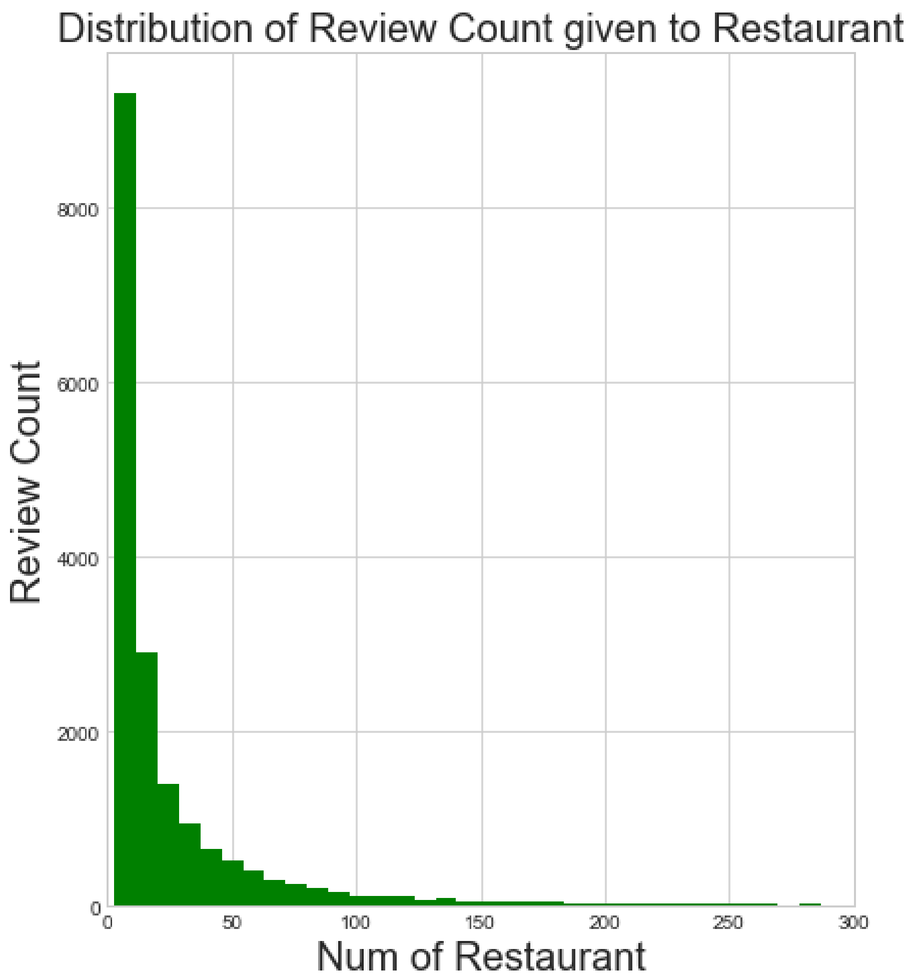
Distribution of Review Count given by users and given to Restaurant

We can see that most review count is with less number of users and restaurants

In [45]:

```
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(15, 8))
user_df_filter_df.review_count_y.hist(bins=400,ax=ax[0],color = 'g')
#plt.xlim([0,1000])
ax[0].legend();
ax[0].set_xlim([0,1000])
ax[0].set_ylabel('Review Count',size=20)
ax[0].set_xlabel('Num of Users',size=20)
ax[0].set_title('Distribution of Review Count given by User',size=20);

restaurant_df.review_count.hist(bins=400,ax=ax[1],color = 'g')
ax[1].set_xlim([0,300])
ax[1].legend();
ax[1].set_ylabel('Review Count',size=20)
ax[1].set_xlabel('Num of Restaurant',size=20)
ax[1].set_title('Distribution of Review Count given to Restaurant',size=20);
```





Models

Creating Baseline Model

In [46]:
```
complete_df.head(2)
```

Out[46]:

	address	attributes	business_id	categories	city	hours	is_open	latitude	longitude	r
0	1203 E Charleston Blvd, Ste 140	{'BusinessParking': {'validated': False, 'gara...	YTqtM2WFhcMZGeAGA08Cfg	['Seafood', 'Restaurants', 'Specialty Food', '...	Las Vegas	{'Sunday': '10:15-21:00', 'Wednesday': '10:30-...	1	36.159363	-115.135949	Ma Pla Esc
1	1203 E Charleston Blvd, Ste 140	{'BusinessParking': {'validated': False, 'gara...	YTqtM2WFhcMZGeAGA08Cfg	['Seafood', 'Restaurants', 'Specialty Food', '...	Las Vegas	{'Sunday': '10:15-21:00', 'Wednesday': '10:30-...	1	36.159363	-115.135949	Ma Pla Esc

Taking only user_id, business_id, stars_y and using the surprise library(https://pypi.python.org/pypi/scikit-surprise (https://pypi.python.org/pypi/scikit-surprise)) Algorithm predicting the baseline estimate for given user and item.

In [47]:
```
display(Math('r^ui=bui=μ+bu+bi'))
```

$$r^{u}i = bui = \mu + bu + bi$$

In [48]:
```
baseline_df = complete_df[['user_id','business_id','stars_y']]
```

In [49]:
```
from surprise import SVD,BaselineOnly, Reader,KNNBaseline
from surprise import Dataset
from surprise import Reader
from surprise import evaluate, print_perf

reader = Reader(rating_scale=(1, 5))
# Load the dataset
# and split it into 3 folds for cross-validation.
data = Dataset.load_from_df(baseline_df,reader)
data.split(n_folds=3)
```

BaselineOnly Model

We used Surprise library for Baseline models. Surprise is a Python scikit for building, and analyzing (collaborative-filtering) recommender systems. Various algorithms are built-in, with a focus on rating prediction. BaselineOnly is an algorithm predicting the baseline estimate for given user and item Ym = μ + su + sm where the unknown parameters su and sm indicate the deviations, or biases, of user u and item m respectively from some intercept parameter.

KNNBaseline is a basic collaborative filtering algorithm taking into account a baseline rating.

In [99]:
```
# Baselineonly model
algo = BaselineOnly()
# Performance
perf_baseline = evaluate(algo, data, measures=['RMSE', 'MAE'])
print_perf(perf_baseline)
```

```
Evaluating RMSE, MAE of algorithm BaselineOnly.

------------
Fold 1
Estimating biases using als...
RMSE: 1.2468
MAE:  1.0153
------------
Fold 2
Estimating biases using als...
RMSE: 1.2374
MAE:  1.0051
------------
Fold 3
Estimating biases using als...
RMSE: 1.2583
MAE:  1.0204
------------
------------
Mean RMSE: 1.2475
Mean MAE : 1.0136
------------
------------
        Fold 1  Fold 2  Fold 3  Mean
RMSE    1.2468  1.2374  1.2583  1.2475
MAE     1.0153  1.0051  1.0204  1.0136
```

KNNBaseline Model

KNN Based on user restaurant rating

In [52]:
```
display(Math(r'\hat{r}_{ui} = \mu_u + \sigma_u \frac{ \sum\limits_{v \in N^k_i(u)}\text{sim}(u, v) \cdot (r_{vi} - \mu_v) / \sigma_v} {\sum\limits_{v\in N^k_i(u)} \text{sim}(u, v)}'))
```

$$\hat{r}_{ui} = \mu_u + \sigma_u \frac{ \sum\limits_{v \in N^k_i(u)}\text{sim}(u, v) \cdot (r_{vi} - \mu_v) / \sigma_v} {\sum\limits_{v\in N^k_i(u)} \text{sim}(u, v)}$$

In [100]:
```python
# KNNBaseline model
algo = KNNBaseline()

# Performance
perf_knn_baseline = evaluate(algo, data, measures=['RMSE', 'MAE'])
print_perf(perf_knn_baseline)
```

```
Evaluating RMSE, MAE of algorithm KNNBaseline.

------------
Fold 1
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 1.2541
MAE:  1.0201
------------
Fold 2
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 1.2429
MAE:  1.0096
------------
Fold 3
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 1.2687
MAE:  1.0287
------------
------------
Mean RMSE: 1.2552
Mean MAE : 1.0195
------------
------------
        Fold 1  Fold 2  Fold 3  Mean
RMSE    1.2541  1.2429  1.2687  1.2552
MAE     1.0201  1.0096  1.0287  1.0195
```

Memory Based Collaborative filtering

We used Collaborative filtering. The two primary areas of collaborative filtering are the neighborhood methods and latent factor models.

Neighborhood methods are centered on computing the relationships between items or, alternatively, between users. The item oriented approach evaluates a user's preference for an item based on ratings of "neighboring" items by the same user. A product's neighbors are other products that tend to get similar ratings when rated by the same user.

In [54]:
```
# Number of unique users nad restaurants
n_users = complete_df['user_id'].nunique()
n_restaurants = complete_df['business_id'].nunique()

print('Number of Unique Users: ', n_users)
print('Number of Restaurant: ',n_restaurants)
```

```
Number of Unique Users:  11749
Number of Restaurant:  482
```

Making user_id and business_id as nominal variable

In [56]:
```
# Creating the nominal variable for  user_id
unique_user_id = pd.DataFrame(complete_df['user_id'].unique(),columns =['user_id']).reset_index()
unique_user_id['new_user_id'] =unique_user_id['index']
del unique_user_id['index']

# Creating the nominal variable for  restaurant_id
unique_business_id = pd.DataFrame(complete_df['business_id'].unique(),columns =['business_id']).reset_index()
unique_business_id['new_business_id'] =unique_business_id['index']
del unique_business_id['index']
```

In [57]:
```
# Joining the nominal user_id and restaurant_id main dataframe with all the data
new_complete_df = complete_df.merge(unique_user_id,on='user_id',how ='left')
new_complete_df = new_complete_df.merge(unique_business_id,on='business_id',how ='left')
```

In [58]:

```
new_complete_df.head(2)
```

Out[58]:

	address	attributes	business_id	categories	city	hours	is_open	latitude	longitude	r
0	1203 E Charleston Blvd, Ste 140	{'BusinessParking': {'validated': False, 'gara...	YTqtM2WFhcMZGeAGA08Cfg	['Seafood', 'Restaurants', 'Specialty Food', '...	Las Vegas	{'Sunday': '10:15-21:00', 'Wednesday': '10:30-...	1	36.159363	-115.135949	Ma Pla Esc
1	1203 E Charleston Blvd, Ste 140	{'BusinessParking': {'validated': False, 'gara...	YTqtM2WFhcMZGeAGA08Cfg	['Seafood', 'Restaurants', 'Specialty Food', '...	Las Vegas	{'Sunday': '10:15-21:00', 'Wednesday': '10:30-...	1	36.159363	-115.135949	Ma Pla Esc

Train Test Split

In [61]:

```
from sklearn.cross_validation import train_test_split
train_data, test_data = train_test_split(new_complete_df, test_size=0.25)
```

In [62]:

```
#Creating two,  user and restaurant matrices, one for training and another for testing
train_data_matrix = np.zeros((n_users, n_restaurants))
for row in train_data.itertuples():
    # selecting new_user_id, new_restaurant_id, and rating star
    train_data_matrix[row[45]-1, row[46]-1] = row[20]

test_data_matrix = np.zeros((n_users, n_restaurants))
for line in test_data.itertuples():
    test_data_matrix[row[45]-1, row[46]-1] = row[20]
```

In [66]:
```
# Calculating the pairwise distances using the cosine metric
from sklearn.metrics.pairwise import pairwise_distances
user_similarity = pairwise_distances(train_data_matrix, metric='cosine')
restaurant_similarity = pairwise_distances(train_data_matrix.T, metric='cosine')
```

In [67]:
```
# Function for predicting rating with argument as number of rating for users and restaurant, similarity betwe
en them and type: user or restaurant

def predict_rating(num_rating, sim, type='user'):
    if type == 'user':
        user_rating_avg = num_rating.mean(axis=1)
        ratings_difference = (num_rating - user_rating_avg[:, np.newaxis])
        prediction = user_rating_avg[:, np.newaxis] + sim.dot(ratings_difference) / np.array([np.abs(sim).sum
(axis=1)]).T
    elif type == 'restaurant':
        prediction = num_rating.dot(sim) / np.array([np.abs(sim).sum(axis=1)])
    return prediction
```

In [71]:
```
# Training prediction
restaurant_prediction = predict_rating(train_data_matrix, restaurant_similarity, type='restaurant')
user_prediction = predict_rating(train_data_matrix, user_similarity, type='user')

# Testing prediction
restaurant_prediction_test = predict_rating(test_data_matrix, restaurant_similarity, type='restaurant')
user_prediction_test = predict_rating(test_data_matrix, user_similarity, type='user')
```

In [72]:
```
model_memory_based_pred_res = restaurant_prediction
model_memory_based_pred_user = user_prediction

model_memory_based_pred_res_test = restaurant_prediction_test
model_memory_based_pred_user_test = user_prediction_test
```

Evaluation using RMSE

In [73]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt
def rmse(prediction, true_value):
    prediction = prediction[true_value.nonzero()].flatten()
    true_value = true_value[true_value.nonzero()].flatten()
    return sqrt(mean_squared_error(prediction, true_value))
```

In [80]:
```
print('RMSE for training  User based Collaborative filtering:', (rmse(user_prediction, train_data_matrix)))
print('RMSE for training Restaurant based Collaborative filtering: ', (rmse(restaurant_prediction, train_data
_matrix)))
print('RMSE for testing  User based Collaborative filtering:', (rmse(user_prediction_test, test_data_matrix
)))
print('RMSE for testing Restaurant based Collaborative filtering: ', (rmse(restaurant_prediction_test, test_d
ata_matrix)))
```

```
RMSE for training  User based Collaborative filtering: 3.927746327283826
RMSE for training User based Collaborative filtering:  3.931752523947338
RMSE for testing  User based Collaborative filtering: 4.9896265560165975
RMSE for testing User based Collaborative filtering:  5.0
```

SVD

Latent factor models (aka SVD) are an alternative approach that tries to explain the ratings by characterizing both items and users on number of factors inferred from the ratings patterns. Latent factor models are based on matrix factorization which characterizes both items and users by vectors of factors inferred from item rating patterns. High correspondence between item and user factors leads to a recommendation. From the results, we can see that prediction accuracy has improved by considering also implicit feedback, which provides an additional indication of user preferences.

In [87]:
```
#Using libraries
import scipy.sparse as sp
from scipy.sparse.linalg import svds

#get SVD components from train matrix. Choose k.
u, s, vt = svds(train_data_matrix, k =10)
s_diag_matrix=np.diag(s)
X_pred = np.dot(np.dot(u, s_diag_matrix), vt)

u_test, s_test, vt_test = svds(test_data_matrix, k =10)
X_pred_test = np.dot(np.dot(u_test, s_diag_matrix), vt)
```

In [88]:
```
print('RMSE for training User based SVD Collaborative filtering: ', (rmse(X_pred, train_data_matrix)))
print('RMSE for testing User based SVD Collaborative filtering: ', (rmse(X_pred_test, test_data_matrix)))
```

```
RMSE for training User based SVD Collaborative filtering:  3.3661688897431503
RMSE for testing User based SVD Collaborative filtering:  5.000000000000065
```

Meta Classifier

We have used multiple models (neighborhoods & SVD) whose individual predictions are combined to classify new examples. Integration should improve predictive accuracy. Each of the models has a mediocre accuracy rate. We would have to increase the importance of the model with high accuracy, and reduce the importance of the models with lower accuracy. To do this in Python, one may use the predicted values as the predictors in a Logistic Regression model, and the corresponding y as the response. Logistic Regression can take the "importance" of each model into account: the "predictors" or models that do well most of the time will have the more significant coefficients.

In [90]:
```
model_svd_based_pred = X_pred
model_svd_based_pred_test = X_pred_test

# flattening the results from each model above for training
model_memory_based_pred_res_flat = model_memory_based_pred_res.ravel()
model_memory_based_pred_user_flat = model_memory_based_pred_user.ravel()
model_svd_based_pred_flat = model_svd_based_pred.ravel()

# flattening the results from each model above for testing
model_memory_based_pred_res_test_flat = model_memory_based_pred_res_test.ravel()
model_memory_based_pred_user_test_flat = model_memory_based_pred_user_test.ravel()
model_svd_based_pred_test_flat = model_svd_based_pred_test.ravel()

# creating a 3-columns  array for 3 models
pred_model_array_train =  np.zeros((model_memory_based_pred_res_flat.size,3))
pred_model_array_test =  np.zeros((model_memory_based_pred_res_test_flat.size,3))

# for training
pred_model_array_train[:,0] = model_memory_based_pred_res_flat
pred_model_array_train[:,1] = model_memory_based_pred_user_flat
pred_model_array_train[:,2] = model_svd_based_pred_flat

# for testing
pred_model_array_test[:,0] = model_memory_based_pred_res_test_flat
pred_model_array_test[:,1] = model_memory_based_pred_user_test_flat
pred_model_array_test[:,2] = model_svd_based_pred_test_flat

# True response values from train and test
y_train_data_matrix_flat = train_data_matrix.ravel()
y_test_data_matrix_flat = test_data_matrix.ravel()
```

In [108]:
```
# function for error calculation
def rmse_new(prediction, true_value):
    return sqrt(mean_squared_error(prediction, true_value))
```

In [113]:
```
from sklearn.metrics import mean_squared_error
logreg = LogisticRegressionCV()
y_hat_train = logreg.fit(pred_model_array_train[0:100000], y_train_data_matrix_flat[0:100000]).predict(pred_m
odel_array_train)
y_hat_test = logreg.fit(pred_model_array_train[0:100000], y_train_data_matrix_flat[0:100000]).predict(pred_mo
del_array_test)

print("Test LogReg RMSE: ", rmse_new(y_test_data_matrix_flat, y_hat_test))
print("Train LogReg RMSE: ", rmse_new(y_train_data_matrix_flat, y_hat_train))
```

```
Test LogReg RMSE:  0.07446305550471391
Train LogReg RMSE:  0.14115554579033043
```

In [104]:
```
print_perf(perf_baseline)
```

```
        Fold 1  Fold 2  Fold 3  Mean
RMSE    1.2468  1.2374  1.2583  1.2475
MAE     1.0153  1.0051  1.0204  1.0136
```

In [105]:
```
print_perf(perf_knn_baseline)
```

```
        Fold 1  Fold 2  Fold 3  Mean
RMSE    1.2541  1.2429  1.2687  1.2552
MAE     1.0201  1.0096  1.0287  1.0195
```

In [152]:
```
dict = {'Meta Classifer Training': meta_clf_scores_tr,
                         'SVD Collaborative Filtetering Training': SVD_cf_scores_tr,
                         'Memory Based User Collaborative Filetering Training': memory_user_based_cf_scores_t
r,
                        'Memory Based Restaurant Collaborative Filtering Training': memory_restaurant_based_c
f_scores_tr}

pd.DataFrame.from_items(dict.items(),
                            orient='index',
                            columns=[1,2,3,4])
```


```
Test LogReg RMSE:  0.07446305550471391
Train LogReg RMSE:  0.14115554579033043
```

Model comparison via RMSE

In [163]:
```
score = [meta_clf_scores_tr,SVD_cf_scores_tr,memory_user_based_cf_scores_tr,memory_restaurant_based_cf_scores_tr,
        meta_clf_scores_ts,SVD_cf_scores_ts,memory_user_based_cf_scores_ts,memory_restaurant_based_cf_scores_ts]


pd.DataFrame(np.array(score).reshape(2,4), columns = ['Meta Classifer','SVD Collaborative Filtetering','Memory Based User Collaborative Filering',
                      'Memory Based Restaurant Collaborative Filtering'], index = ['RMSE in Training','RMSE in Testing'])
```

Out[163]:

	Meta Classifer	SVD Collaborative Filtetering	Memory Based User Collaborative Filering	Memory Based Restaurant Collaborative Filtering
RMSE in Training	0.074463	3.366169	3.927746	3.931753
RMSE in Testing	0.141156	5.000000	4.989627	5.000000

We can see above that meta Classifier is working better than other models

References:

1. How the Netflix prize was won, http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/ (http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/)
2. Matrix factorization for recommender systems, https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf (https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf)
3. Ensembling for the Netflix Prize, http://web.stanford.edu/~lmackey/papers/netflix_story-nas11-slides.pdf (http://web.stanford.edu/~lmackey/papers/netflix_story-nas11-slides.pdf)
4. Reviews on methods for netflix prize, http://arxiv.org/abs/1202.1112andhttp://www.grouplens.org/system/files/FnT%20CF%20Recsys%20Survey.pdf (http://arxiv.org/abs/1202.1112andhttp://www.grouplens.org/system/files/FnT%20CF%20Recsys%20Survey.pdf)
5. Advances in Collaborative Filtering from the Netflix prize, https://datajobs.com/data-science-repo/Collaborative-Filtering-%5BKoren-and-Bell%5D.pdf (https://datajobs.com/data-science-repo/Collaborative-Filtering-%5BKoren-and-Bell%5D.pdf)
6. Python Surprise library for models, https://pypi.python.org/pypi/scikit-surprise (https://pypi.python.org/pypi/scikit-surprise).
7. Library for SVD, https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.svd.html (https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.svd.html)
8. Library for accessign Similarity, http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.pairwise_distances.html (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.pairwise_distances.html)