In [193]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
import re
import seaborn as sns
sns.set()

from keplergl import KeplerGl

import datetime as dt

from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))

import warnings
warnings.filterwarnings('ignore')
```

This project delves into the dataset to uncover trends, preferences, and insights behind Dave Portnoy's iconic pizza ratings.

With the help of some online tutorials, I was able to find the API being used by the barstool app. The general API is https://one-bite-api.barstoolsports.com (https://one-bite-api.barstoolsports.com) and does not require any form of user authentication. The venue endpoint, accessed by adding /venue after the general API, contains the pizza scores and other useful information.

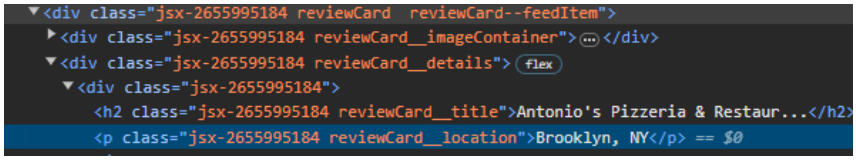
Example API Call:

https://one-bite-api.barstoolsports.com/venue?dave=0&lat=40.73&lon=-73.98&state=NY&zip=11216 (https://one-bite-api.barstoolsports.com/venue?dave=0&lat=40.73&lon=-73.98&state=NY&zip=11216)

Notes for API:

- returns a maximum of 100 items at a time
- has a `dave` boolean parameter, 1 for places Dave has been to, and 0 for places he has not
- has `zipcode` parameter, which gets converted into lat/long when retrieving data
- has `radius` parameter with units in millimeters

Implementing a grid search throughout the states would be inefficient since most of the Dave-reviewed pizza shops are located in high population cities near the east coast such as NYC, Boston, and so on. The barstool website has a list of all reviews including the locations. Each page (https://onebite.app/reviews/dave?page=1&minScore=0&maxScore=10) has 30 reviews and there are a total of 54 pages. By inspecting the HTML elements that contain the city names, I found their respective names and indexes in the page source code.



Then I used Selenium to go through each page and extract the city names.

In [35]:
```
pageNumbers = range(1,55)
cityList = []

from selenium import webdriver
import urllib.request, json

for pageNumber in pageNumbers:
    base_url = 'https://onebite.app/reviews/dave?page=' + str(pageNumber) + '&minScore=0&maxScore=10'
    driver = webdriver.Chrome()
    driver.get(base_url)
    time.sleep(3)
    html_source = driver.page_source

    locations = [location.start() for location in re.finditer('reviewCard__location', html_source)]
    names = [name.start() for name in re.finditer('reviewCard__title', html_source)]
    index = 1
    for locationIndex in locations[2:]:
        locationElement = html_source[locationIndex:locationIndex+55]
        city = re.findall(r'">(.*)</p>', locationElement)[0]
        cityList.append(city)
```

In [36]:
```
print("Dave has done " + str(len(cityList)) + " reviews in " + str(len(np.unique(cityList))) + " different cities.")
city
```

```
Dave has done 1619 reviews in 460 different cities.
```

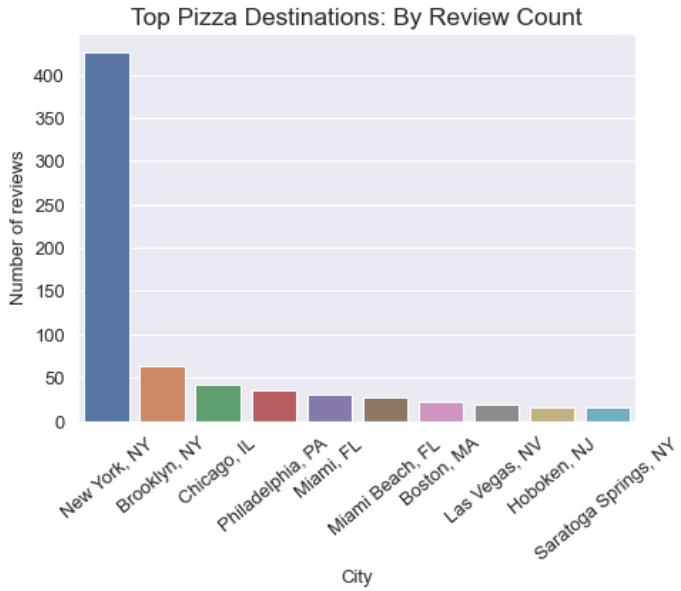
In [37]:
```
pd.DataFrame(data=pd.Series(cityList),columns={'cityName'}).to_csv('data/cities.csv', index=False)
```

In [38]:
```
def splitCityName(cityState):
    '''
    Used to split "City, State" string to two different variables.
    '''
    index = cityState.find(',')
    city = cityState[:index]
    state = cityState[index+2:]
    return pd.Series([city,state])
```

In [39]:
```
city_df = pd.DataFrame(data=pd.Series(cityList),columns={'cityName'}).reset_index().rename(columns={'index':'count'})
city_df[['city', 'state']] = city_df.apply(lambda x: splitCityName(x['cityName']), axis=1)
unique_city_df = city_df.drop_duplicates(['cityName'])
```

In [131]:
```
review_counts = city_df.groupby('cityName').count().sort_values('count',ascending=False).reset_index().rename(columns={'
review_counts_top10 = review_counts.head(10)
```

In [136]:
```
plt.figure(figsize=(7,5))
sns.barplot(x=review_counts_top10['cityName'],y=review_counts_top10['reviewCount'])
plt.xticks(rotation=40, size=13);
plt.yticks(size=13)
plt.xlabel('City',size=13)
plt.ylabel('Number of reviews',size=13)
plt.title("Top Pizza Destinations: By Review Count", size='x-large');
```



Nearly 1/4 of the reviews are based in New York City, where Barstool Sports headquarters is located. The rest of the locations in the histogram are scattered along the US east coast with the exception of Las Vegas, NV. Only two cities have more than 50 reviews.

Now that I have a list of cities Dave has visited, I know exactly where to search using the API.

In [240]:
```
def getAPIurl(Lat,Lon):
    '''
    Returns API url using lat/long search
    '''
    lat = round(Lat,2)
    lon = round(Lon,2)
    return("https://one-bite-api.barstoolsports.com/venue?dave=1&lat=" + str(Lat) + "&lon=" + str(Lon))
```

In [217]:
```
us_city_df = pd.read_csv('data/uscities.csv') # Dataset from https://simplemaps.com/data/us-cities
print(us_city_df.columns)
us_city_df['cityName'] = us_city_df['city'] + ', ' + us_city_df['state_id']
us_city_df = us_city_df[['cityName', 'state_name', 'lat', 'lng', 'zips']]
```

```
Index(['city', 'city_ascii', 'state_id', 'state_name', 'county_fips',
       'county_name', 'lat', 'lng', 'population', 'density', 'source',
       'military', 'incorporated', 'timezone', 'ranking', 'zips', 'id'],
      dtype='object')
```

In [218]:
```
us_city_df = pd.merge(left=unique_city_df, right=us_city_df,on='cityName')
us_city_df.head(5)
```

Out[218]:

	count	cityName	city	state	state_name	lat	lng	zips
0	0	Brooklyn, NY	Brooklyn	NY	New York	40.6501	-73.9496	11229 11226 11225 11224 11223 11221 11220 1124...
1	1	Fort Worth, TX	Fort Worth	TX	Texas	32.7817	-97.3474	76164 76040 76134 76135 76137 76131 76132 7613...
2	2	Hazel Park, MI	Hazel Park	MI	Michigan	42.4619	-83.0977	48030
3	3	Royal Oak, MI	Royal Oak	MI	Michigan	42.5084	-83.1539	48067 48073 48068
4	4	Detroit, MI	Detroit	MI	Michigan	42.3834	-83.1024	48209 48208 48202 48201 48207 48206 48205 4820...

The dataset from simplemaps contains the lat,long coordinates of every city and the corresponding zipcodes. We can use the zipcodes to generate more lat/long coordinates for cities like NYC where the review count is larger than the API limit.

In [247]:
```
df_reviews = pd.DataFrame()
for index, row in us_city_df.iterrows():
    with urllib.request.urlopen(getAPIurl(Lat=row['lat'],Lon=row['lng'])) as url:
        data = json.loads(url.read().decode())
        df = pd.json_normalize(data)
        df_reviews = pd.concat([df_reviews, df], ignore_index=True)
        df_reviews = df_reviews.drop_duplicates(subset=['address1', 'createdAt'])
```

In [263]:
```
print("Retrieved " + str(len(df_reviews)) + " reviews.")
df_reviews.columns
#df_reviews.to_csv('data/pizza-data.csv',index=False)
```

```
Retrieved 1471 reviews.
```

Out[263]:
```
Index(['deleted', 'orderProvider', 'placeId', 'address1', 'address2',
       'categories', 'city', 'country', 'createdAt', 'imageUrl', 'modifiedAt',
       'name', 'phoneNumber', 'priceLevel', 'providerRating',
       'providerReviewCount', 'providerTransactions', 'providerUrl', 'state',
       'type', 'zip', 'openHours', 'photos', 'refreshDate', 'timeZone',
       '_distance', 'id', 'slug', 'reviewStats.all.count',
       'reviewStats.all.totalScore', 'reviewStats.all.averageScore',
       'reviewStats.community.count', 'reviewStats.community.totalScore',
       'reviewStats.community.averageScore', 'reviewStats.critic.count',
       'reviewStats.critic.totalScore', 'reviewStats.critic.averageScore',
       'reviewStats.dave.count', 'reviewStats.dave.totalScore',
       'reviewStats.dave.averageScore', 'loc.type', 'loc.coordinates',
       'featuredMedia.streams.mp4', 'featuredMedia.streams.hls',
       'featuredMedia.thumbnails.small', 'featuredMedia.thumbnails.medium',
       'featuredMedia.thumbnails.large', 'featuredMedia.metadata.aspectRatio',
       'featuredMedia.metadata.duration', 'featuredMedia.metadata.frameRate',
       'featuredMedia.metadata.resolution', 'featuredMedia._id',
       'featuredMedia.deleted', 'featuredMedia.user', 'featuredMedia.status',
       'featuredMedia.type', 'featuredMedia.sourceUrl',
       'featuredMedia.assetId', 'featuredMedia.playbackId',
       'featuredMedia.reviewId', 'featuredMedia.createdAt',
       'featuredMedia.modifiedAt', 'featuredMedia.__v', 'featuredMedia.id',
       'featuredMedia.__t', 'orderProvider.name', 'orderProvider.checkoutUrl',
       'orderProvider.internalId', 'orderProvider.logo', 'orderProvider._id'],
      dtype='object')
```

The API returns a lot of information including name and location of venues, open hours, review video duration, and Dave and the community's review scores.

In [300]:
```
df_reviews.isna().sum().to_string()
```

Out[300]:
```
'deleted                                  0\norderProvider                        1471\nplaceId
6\naddress1                                 1\naddress2                           260\ncategories
0\ncity                                     0\ncountry                              0\ncreatedAt
0\nimageUrl                                 6\nmodifiedAt                          0\nname
0\nphoneNumber                              6\npriceLevel                          6\nproviderRating
5\nproviderReviewCount                      5\nproviderTransactions                 0\nproviderUrl
4\nstate                                    0\ntype                                 0\nzip
0\nopenHours                                0\nphotos                               0\nrefreshDate
8\ntimeZone                                294\n_distance                           0\nid
0\nslug                                     0\nreviewStats.all.count                 0\nreviewStats.all.totalScore
0\nreviewStats.all.averageScore              0\nreviewStats.community.count           0\nreviewStats.community.total
Score         0\nreviewStats.community.averageScore       0\nreviewStats.critic.count                0\nreviewStats.c
ritic.totalScore            0\nreviewStats.critic.averageScore          0\nreviewStats.dave.count                   0
\nreviewStats.dave.totalScore              0\nreviewStats.dave.averageScore            0\nloc.type
0\nloc.coordinates                           0\nfeaturedMedia.streams.mp4              0\nfeaturedMedia.streams.hls
0\nfeaturedMedia.thumbnails.small            0\nfeaturedMedia.thumbnails.medium        0\nfeaturedMedia.thumbnails.la
rge           0\nfeaturedMedia.metadata.aspectRatio       0\nfeaturedMedia.metadata.duration          0\nfeaturedMedi
a.metadata.frameRate         0\nfeaturedMedia.metadata.resolution        0\nfeaturedMedia._id                     1471
\nfeaturedMedia.deleted                    0\nfeaturedMedia.user                   305\nfeaturedMedia.status
0\nfeaturedMedia.type                        0\nfeaturedMedia.sourceUrl                0\nfeaturedMedia.assetId
0\nfeaturedMedia.playbackId                  0\nfeaturedMedia.reviewId                 0\nfeaturedMedia.createdAt
0\nfeaturedMedia.modifiedAt                  0\nfeaturedMedia.__v                    1471\nfeaturedMedia.id
0\nfeaturedMedia.__t                       1471\norderProvider.name                    880\norderProvider.checkoutUrl
880\norderProvider.internalId                880\norderProvider.logo                    880\norderProvider._id
880'
```

Dave uses a simple 0 to 10 scoring system. Here's a more detailed breakdown:

- Anything below 3.0 is virtually inedible
- 3.1 to 5.0: is below "average"
- 5.1 to 7.0: is considered football pizza, good for large gatherings or if you are drunk, but nothing too special
- 7.1 to 8.0: pizzas in this range are considered good to very good, and they can be your regular local pizza spot

- 8.1 to 9.0: pizzas in this range are considered excellent, and are worth travelling at least a few hours to try them
- 9.1 to 10.0: pizzas in this range are considered outstanding and nearly perfect.

In [2]:
```
#pizza_df = pd.read_csv('data/pizza-data.csv')
```

In [74]:
```
print(f"The median review score for all data is {pizza_df['reviewStats.dave.averageScore'].median():.1f} with a standard
```

```
The median review score for all data is 7.3 with a standard deviation of 1.4.
```

Top 10 Pizza shops by score

In [278]:
```
high_scores = df_reviews.sort_values(by='reviewStats.dave.averageScore', ascending=False).reset_index(drop=True)
high_scores[['name', 'city', 'state','reviewStats.dave.averageScore']].head(10)
```

Out[278]:

	name	city	state	reviewStats.dave.averageScore
0	Monte's Restaurant	Lynn	MA	10.0
1	Di Fara Pizza	Brooklyn	NY	9.4
2	DeLucia's Brick Oven Pizza	Raritan	NJ	9.4
3	Frank Pepe Pizzeria Napoletana	Chestnut Hill	MA	9.4
4	Oath Pizza - Nantucket	Nantucket	MA	9.3
5	Lucali	Brooklyn	NY	9.3
6	John's of Bleecker Street	New York	NY	9.3
7	Lazzara's Pizza	New York	NY	9.3
8	Luigi's Pizza	Brooklyn	NY	9.3
9	Angelo's Coal Oven Pizzeria	New York	NY	9.3

In [47]:
```
cities_df = pizza_df.groupby('city')['reviewStats.dave.averageScore'].agg(avg='mean', size='size')
cities_df = cities_df.sort_values(by='avg', ascending=False)
top6_cities = cities_df[cities_df['size'] >= 10].sort_values(by='size', ascending=False).head(6).reset_index().rename(co
top6_cities.head(6)
```

Out[47]:

	city	dave_score	size
0	New York	6.362575	299
1	Brooklyn	7.660317	63
2	Chicago	7.276190	42
3	Philadelphia	7.083333	36
4	Miami	7.216667	30
5	Miami Beach	6.762963	27

In [60]:
```
filtered_pizza = pizza_df[pizza_df['city'].isin(top6_cities['city'])]
filtered_pizza['dave_score'] = filtered_pizza['reviewStats.dave.averageScore']

g = sns.FacetGrid(filtered_pizza, col="city", col_wrap=3)
g = g.map(sns.distplot, "dave_score")
```

The distributions of the top reviewed cities follow similar patterns; left-skewed and centered near 7.

Pizza capital of the world

In [436]:
```
cities_df = pizza_df.groupby('city')['reviewStats.dave.averageScore'].agg(avg='mean', size='size')
cities_df = cities_df.sort_values(by='avg', ascending=False)
pizza_capital_df = cities_df[cities_df['size'] >= 5].sort_values(by='avg', ascending=False).head(100).reset_index().rena
pizza_capital_df.head(5)
```

Out[436]:

	city	avg_dave_score	size
0	New Haven	7.833333	9
1	Staten Island	7.720000	10
2	Brooklyn	7.660317	63
3	Jersey City	7.550000	8
4	Toronto	7.528571	14

It should be no surprise that New Haven is at the top of the list. New Haven pizza, or "apizza", is renowned for its thin, coal-fired crust, and high quality ingredients.

Dave's ratings over time

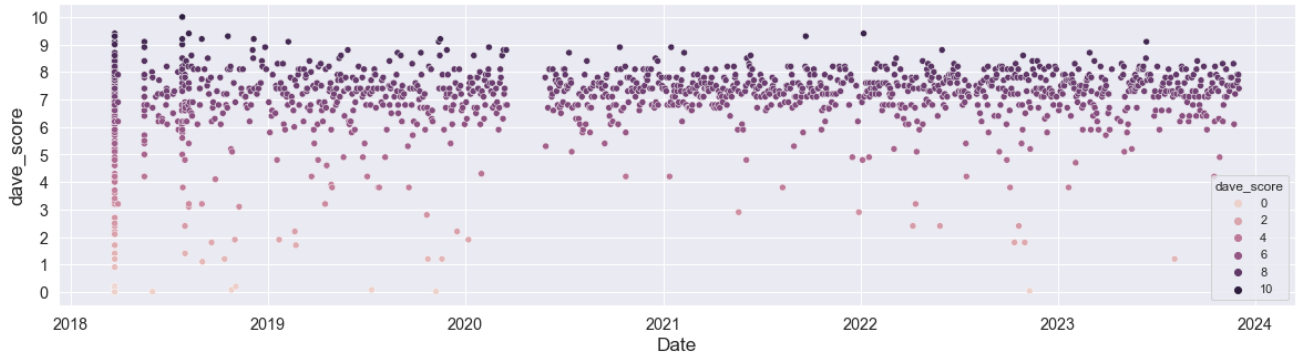
In [166]:
```
pizza_df['createdAt'] = pd.to_datetime(pizza_df['createdAt'])
pizza_df['featuredMedia.createdAt'] = pd.to_datetime(pizza_df['featuredMedia.createdAt'])
timeseries_df = pizza_df[['featuredMedia.createdAt','city','name','reviewStats.dave.averageScore']].rename(columns={'rev
                                                                                                                     'featuredMedia.crea
timeseries_df['dave_score_cumsum'] = timeseries_df['dave_score'].cumsum()
timeseries_df = timeseries_df.reset_index()
timeseries_df['index'] = timeseries_df['index'] + 1
timeseries_df['dave_score_rolling_average'] = np.round(timeseries_df['dave_score_cumsum'] / timeseries_df['index'],2)
timeseries_df.head(5)
```

Out[166]:

	index	date	city	name	dave_score	dave_score_cumsum	dave_score_rolling_average
0	1	2018-03-23 17:38:24.269000+00:00	Bronx	Pugsley's Pizza	7.7	7.7	7.70
1	2	2018-03-23 17:38:26.645000+00:00	Brooklyn	Williamsburg Pizza	8.6	16.3	8.15
2	3	2018-03-23 17:38:31.358000+00:00	New York	Nino's 46	6.9	23.2	7.73
3	4	2018-03-23 17:38:38.259000+00:00	New York	La Gusto Pizza	3.2	26.4	6.60
4	5	2018-03-23 17:38:40.383000+00:00	New York	Cheesy Pizza	6.1	32.5	6.50

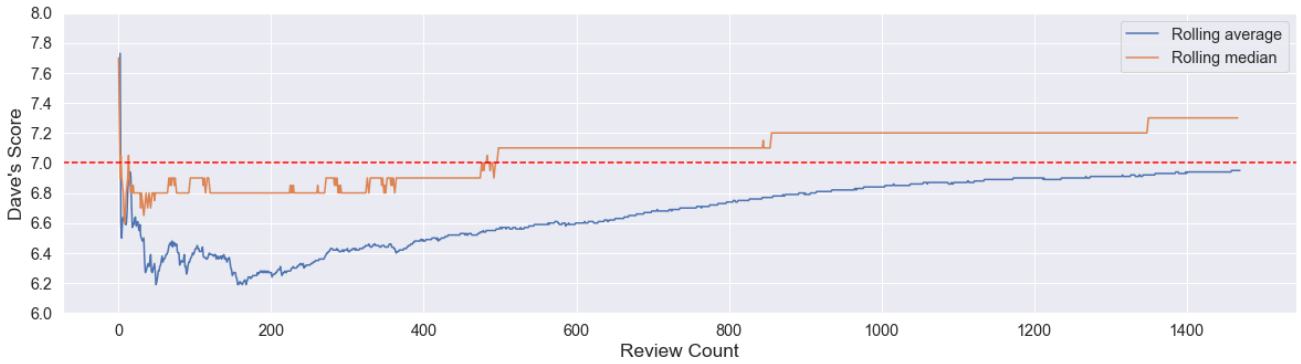
In [167]:
```
median_list = []
for index in range(3,len(timeseries_df)):
    scores = timeseries_df.iloc[:index]['dave_score']
    median_score = scores.median()
    median_list.append(median_score)
```

In [200]:
```
plt.figure(figsize=(20,5))
sns.scatterplot(data=timeseries_df, x="date", y="dave_score", hue="dave_score")
#sns.lineplot(data=timeseries_df, x="date", y="dave_score_rolling_average")
plt.yticks(ticks=np.arange(0,11));
plt.xticks(size='large')
plt.yticks(size='large')
plt.xlabel('Date', size='x-large')
plt.ylabel('dave_score', size='x-large');
```



Overall, Dave seems pretty consistent in his ratings. He has stated in multiple reviews he was particularly generous early on (first 10 to 20 reviews), and there were definitely some ratings higher than deserved. Obviously, the pizza reviews stopped when quarantines measures started and resumed when they were over. By observation, there seem to be less low scores (<5) after the quarantine than before.

In [249]:
```
plt.figure(figsize=(20,5))
plt.ylim([6,8])
plt.plot(timeseries_df['dave_score_rolling_average'].iloc[2:], label='Rolling average')
plt.plot(median_list, label='Rolling median')
plt.axhline(y=7, color='red', linestyle='--')
plt.xticks(size='large')
plt.yticks(np.linspace(6,8,11), size='large')
plt.xlabel("Review Count", size='x-large')
plt.ylabel("Dave's Score", size='x-large')
plt.legend(fontsize='large');
```



As the sample size gets larger, Dave's average score approaches 7, represented by the red dashed line.

Let's take a closer look at Dave's scores before and after the pandemic.

In [206]:
```
timeseries_df_before = timeseries_df[timeseries_df['date'] < '2020-03-15']
timeseries_df_after = timeseries_df[timeseries_df['date'] >= '2020-03-15']
```

In [239]:
```
fig, ax = plt.subplots(figsize=(10,5))

import matplotlib.patches as mpatches
import matplotlib.pyplot as plt

orange_patch = mpatches.Patch(color='orange', label='Before COVID')
blue_patch = mpatches.Patch(color='lightskyblue', label='After COVID')
plt.xlim([0,10])
plt.xticks(ticks=np.arange(0,11));
plt.yticks(ticks=np.linspace(0,1,11));

sns.distplot(timeseries_df_before['dave_score'], ax=ax)
sns.distplot(timeseries_df_after['dave_score'], ax=ax)
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles, labels)
plt.legend(handles=[orange_patch, blue_patch])
plt.xlabel("Pizza Review Scores")
plt.ylabel("Propensity");
```

In [250]:
```
print(f"The mean review score before COVID-19 is {timeseries_df_before['dave_score'].mean():.1f} with a standard deviati
print(f"The mean review score after COVID-19 is {timeseries_df_after['dave_score'].mean():.1f} with a standard deviation
```

```
The mean review score before COVID-19 is 6.8 with a standard deviation of 1.6.
The mean review score after COVID-19 is 7.2 with a standard deviation of 1.0.
```

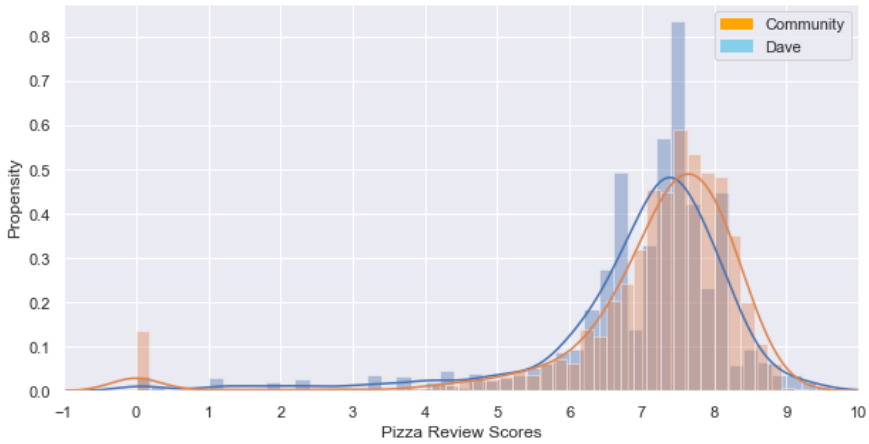
The average score for reviews after the quarantine is 0.4 points higher than before the quarantine.

In [254]:
```
fig, ax = plt.subplots(figsize=(10,5))

orange_patch = mpatches.Patch(color='orange', label='Community')
blue_patch = mpatches.Patch(color='skyblue', label='Dave')
plt.xlim([0,10])
plt.xticks(ticks=np.arange(-1,11));
plt.yticks(ticks=np.linspace(0,1,11));

types = ['reviewStats.dave.averageScore', 'reviewStats.community.averageScore']
for col in types:
    sns.distplot(pizza_df[col], ax=ax)
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles, labels)
plt.legend(handles=[orange_patch, blue_patch])
plt.xlabel("Pizza Review Scores")
plt.ylabel("Propensity")
```

Out[254]:
```
Text(0, 0.5, 'Propensity')
```



The user ratings distribution is near idential to Dave's distribution, but with a lot more 0 ratings and ratings above the median. Users are more polarized with their ratings.

In [278]:
```
import scipy.stats as st
def get_test_results(data):
    test_names = ["levene", "mannwhitneyu", "median_test"]
    test_results = []
    params = {}
    for test_name in test_names:
        test = getattr(st, test_name)
        param = test(data[0], data[1])

        params[test_name] = param
        # Applying the Kolmogorov-Smirnov test
        p = param[1]
        print("p value for "+test_name+" = "+str(p))
        test_results.append((test_name, p))
```

In [279]:
```
get_test_results([pizza_df['reviewStats.dave.averageScore'], pizza_df['reviewStats.community.averageScore']])
```

```
p value for levene = 0.5954199535225673
p value for mannwhitneyu = 7.907770731027693e-12
p value for median_test = 2.4403532524143715e-07
```

In [261]:
```
def get_best_distribution(data):
    dist_names = ["norm", "exponweib", "weibull_max", "weibull_min", "pareto", "genextreme"]
    dist_results = []
    params = {}
    for dist_name in dist_names:
        dist = getattr(st, dist_name)
        param = dist.fit(data)

        params[dist_name] = param
        # Applying the Kolmogorov-Smirnov test
        D, p = st.kstest(data, dist_name, args=param)
        print("p value for "+dist_name+" = "+str(p))
        dist_results.append((dist_name, p))

    # select the best fitted distribution
    best_dist, best_p = (max(dist_results, key=lambda item: item[1]))
    # store the name of the best fit and its p value

    print("Best fitting distribution: "+str(best_dist))
    print("Best p value: "+ str(best_p))
    print("Parameters for the best fit: "+ str(params[best_dist]))

    return best_dist, best_p, params[best_dist]
```

In [262]:
```
pizza_distribution = get_best_distribution(pizza_df['reviewStats.dave.averageScore'])
```

```
p value for norm = 3.6546532599914025e-46
p value for exponweib = 2.3414984902663178e-12
p value for weibull_max = 9.689364027210613e-34
p value for weibull_min = 0.0
p value for pareto = 5.618710950443341e-299
p value for genextreme = 9.721364747406004e-34
Best fitting distribution: exponweib
Best p value: 2.3414984902663178e-12
Parameters for the best fit: (0.6985506680844615, 70210.43454016485, -52487.553236639156, 52495.325509934744)
```

Geomapping

The coordinates from barstool database are not accurate. We will need to use a geocode API to get precise coordinates.

In [332]:
```
pizza_df['address_total'] = pizza_df['address1'] + ", " + pizza_df['city'] + ", " + pizza_df['country']
pizza_df.iloc[0]['address_total']
```

Out[332]: '905 Church Ave, Brooklyn, US'

In [366]:
```
import requests

api_key = ""
def get_google_results(address, api_key=None, return_full_response=False):
    # Set up your Geocoding url
    geocode_url = "https://maps.googleapis.com/maps/api/geocode/json?address={}".format(address)
    if api_key is not None:
        geocode_url = geocode_url + "&key={}".format(api_key)

    # Ping google for the reuslts:
    results = requests.get(geocode_url)
    # Results will be in JSON format - convert to dict using requests functionality
    results = results.json()

    # if there's no results or an error, return empty results.
    if len(results['results']) == 0:
        output = {
            "formatted_address" : None,
            "latitude": None,
            "longitude": None,
            "accuracy": None,
            "google_place_id": None,
            "type": None,
            "postcode": None
        }
    else:
        answer = results['results'][0]
        output = {
            "formatted_address" : answer.get('formatted_address'),
            "latitude": answer.get('geometry').get('location').get('lat'),
            "longitude": answer.get('geometry').get('location').get('lng'),
            "accuracy": answer.get('geometry').get('location_type'),
            "google_place_id": answer.get("place_id"),
            "type": ",".join(answer.get('types')),
            "postcode": ",".join([x['long_name'] for x in answer.get('address_components')
                                  if 'postal_code' in x.get('types')])
        }

    # Append some other details:
    output['input_string'] = address
    output['number_of_results'] = len(results['results'])
    output['status'] = results.get('status')
    if return_full_response is True:
        output['response'] = results

    return output
```

In [ ]:
```
results = []
for address in pizza_df['address_total']:
    geocoded = False
    while geocoded is not True:
        try:
            geocode_result = get_google_results(address, api_key, return_full_response=True)
        except Exception as e:
            print(e)
            print("Major error with {}".format(address))
            print("Skipping!")
            geocoded = True

        if geocode_result['status'] == 'OVER_QUERY_LIMIT':
            print("Hit Query Limit! Backing off for a bit.")
            time.sleep(BACKOFF_TIME * 60) # sleep for 30 minutes
            geocoded = False
        else:
            if geocode_result['status'] != 'OK':
                print("Error geocoding {}: {}".format(address, geocode_result['status']))
            print("Geocoded: {}: {}".format(address, geocode_result['status']))
            results.append(geocode_result)
            geocoded = True
```

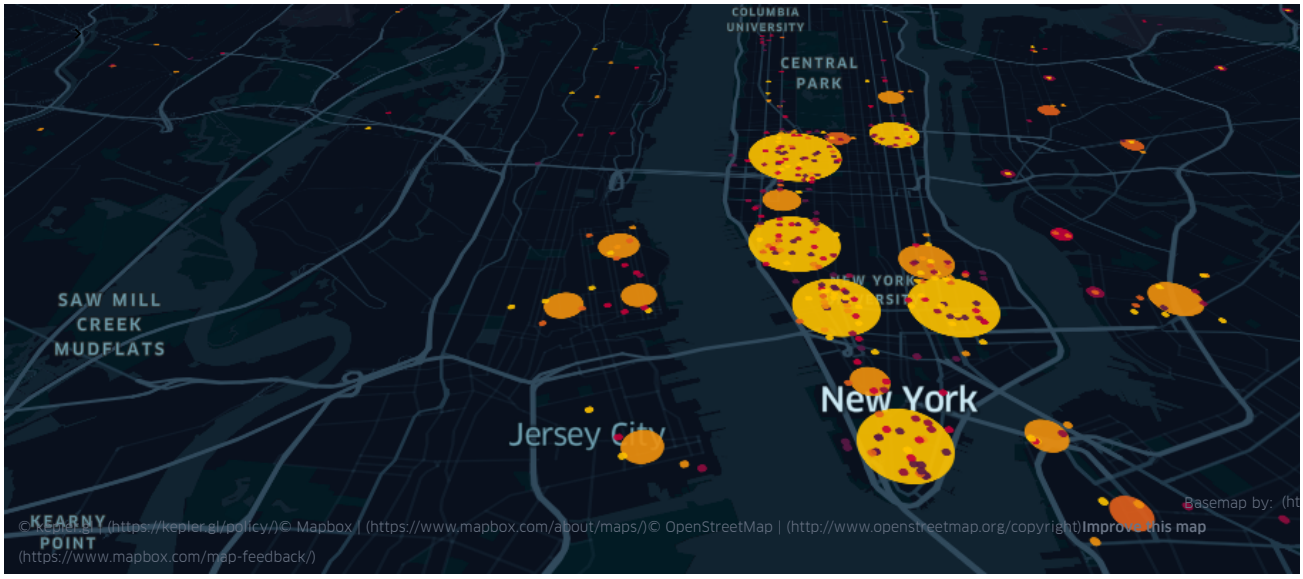
In [375]:
```
pizza_df = pd.concat([pizza_df, pd.json_normalize(results)], axis=1)
#pizza_df.to_csv("data/pizza_data2.csv", index=False)
```

In [377]:
```
pizza_df = pizza_df.rename(columns={'latitude':'Latitude', 'longitude':'Longitude'})
```

In [384]:
```
geomap_df = pizza_df[['name', 'Latitude', 'Longitude', 'reviewStats.dave.averageScore']]
geomap_df['reviewStats.dave.averageScore'] = geomap_df['reviewStats.dave.averageScore'] * 10
```

In [441]:
```
map_1 = KeplerGl(height=400, data={"data_1": geomap_df})
map_1
```

```
User Guide: https://docs.kepler.gl/docs/keplergl-jupyter (https://docs.kepler.gl/docs/keplergl-jupyter)
```



Final notes:

Wrangling this data was like solving a puzzle, and analyzing it felt like uncovering hidden secrets.

Other possible questions to ask:

- Do shops with shorter openHours rank higher than shops with longer openHours?
- Do cash only shops rank higher than shops that accept all forms of payment?
- Using clustering methods to see what attributes are shared among high ranking pizza shops
- Mapping closest pizza shop by transit stop