K Means Clustering of University Data

The Data

Features:

- Apps: Number of applications received
- Accept: Number of applications accepted
- Enroll: Number of new students enrolled
- Top10perc: % of new students from top 10% of their high school class
- Top25perc: % of new students from top 25% of their high school class
- F.Undergrad: Number of full-time undergraduates
- P.Undergrad: Number of part-time undergraduates
- Outstate: Out-of-state tuition
- Room.Board: Room and board costs
- Books: Estimated book costs
- Personal: Estimated personal spending
- PhD: % of faculty with PhDs
- Terminal: % of faculty with a terminal degree (PhD/JD/MD/MBA/etc)
- S.F.Ratio: Student/faculty ratio
- perc.alumni: % alumni who donate
- Expend: Instructional expenditure per student
- Grad.Rate: Graduation rate

EDA = Exploratory Data Analysis

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

colleges = pd.read_csv('College_Data',index_col=0)
colleges.describe()
```

Out[2]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstat
count	777.000000	777.000000	777.000000	777.000000	777.000000	777.000000	777.000000	777.00000
mean	3001.638353	2018.804376	779.972973	27.558559	55.796654	3699.907336	855.298584	10440.66924
std	3870.201484	2451.113971	929.176190	17.640364	19.804778	4850.420531	1522.431887	4023.01648
min	81.000000	72.000000	35.000000	1.000000	9.000000	139.000000	1.000000	2340.00000
25%	776.000000	604.000000	242.000000	15.000000	41.000000	992.000000	95.000000	7320.00000
50%	1558.000000	1110.000000	434.000000	23.000000	54.000000	1707.000000	353.000000	9990.00000
75%	3624.000000	2424.000000	902.000000	35.000000	69.000000	4005.000000	967.000000	12925.00000
max	48094.000000	26330.000000	6392.000000	96.000000	100.000000	31643.000000	21836.000000	21700.00000

Purely by looking at the max value of some of these columns, a few things stand out:

1) It's common knowledge that top private schools can charge up to $60k/year, butthemaxout-of-statetuitionhereisonly$21,700, so we can assume this tuition is listed on a per semester basis rather than per year.

2) There's a school that has above 100% of faculty with PhDs! This seems fishy, but it can probably be explained by a handful of over-achieving faculty who may have a DUAL PhD that have been double counted.

3) In the same vein, we've also got a school with a graduation rate of 118%! We'll explore this anomaly later...

Feel free to poke around the mean, median, and other statistics to see what sticks out to you.

Let's figure out which college had the highest number of applicants.

In [3]:

```
colleges.loc[colleges['Apps']==np.max(colleges['Apps'])]
```

Out[3]:

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Bo
Rutgers at New Brunswick	No	48094	26330	4520	36	79	21401	3712	7410	4

Turns out Rutgers has the highest number of applicants and acceptances, but not the highest number of enrollments. Let's figure which college had the highest enrollment.

In [4]:

```
colleges.loc[colleges['Enroll']==np.max(colleges['Enroll'])]
```

Out[4]:

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Board
Texas A&M Univ. at College Station	No	14474	10519	6392	49	85	31643	2798	5130	3412

Of the 777 universities in this dataset, Texas A&M takes the cake for highest enrollment. What about the school with the highest percentage of students in the top 10% of their high school class?

In [5]:

```
colleges.loc[colleges['Top10perc']==np.max(colleges['Top10perc'])]
```

Out[5]:

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room
Massachusetts Institute of Technology	Yes	6411	2140	1078	96	99	4481	28	20100	

Another major bragging point for high schools and universities alike is their student-faculty ratio: the lower the better. The winner here is University of Charleston, with a **VERY** low ratio of 2.5, which is practically as intimate a teaching environment as homeschooling or private tutoring. Considering that the 25th percentile of the student-faculty ratio is 11.5, U of C is a major outlier.

In [6]:

```
colleges.loc[colleges['S.F.Ratio']==np.min(colleges['S.F.Ratio'])]
```

Out[6]:

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Bo

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Bo
University of Charleston	Yes	682	535	204	22	43	771	611	9500	3

Now let's talk money: which university has the highest alumni donation rate?

In [7]:
```
colleges.loc[colleges['perc.alumni']==np.max(colleges['perc.alumni'])]
```

Out[7]:

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Boar
Williams College	Yes	4186	1245	526	81	96	1988	29	19629	579

Hardly surprising that Williams takes it away as it's a small liberal arts college (~2000 students total) that caters to largely wealthy families with a strong network. Let's check out the statistics for the percentage of alumni donations.

In [8]:
```
colleges.describe()['perc.alumni']
```

Out[8]:
```
count    777.000000
mean      22.743887
std       12.391801
min        0.000000
25%       13.000000
50%       21.000000
75%       31.000000
max       64.000000
Name: perc.alumni, dtype: float64
```

Hence the abundance of on-campus job opportunities as student callers - the median percentage is a measly 21%.

Now here's the juicy stuff: let's see who's really getting their dollar's worth from their university education by looking at the ratio of student expenditure to out-of-state tuition.

In [9]:
```
colleges['expense ratio'] = colleges['Expend']/(colleges['Outstate'])
colleges.describe()['expense ratio']
```

Out[9]:
```
count    777.000000
mean       0.958618
std        0.359789
min        0.378261
25%        0.741767
50%        0.862842
75%        1.072951
max        3.682883
Name: expense ratio, dtype: float64
```

In [10]:
```
np.percentile(colleges['expense ratio'],69)
```

Out[10]:
```
0.997309962048866
```

This tells me that ~69% of universities are investing more money in their students than they're charging. In other words, ~69% of these universities are LOSING MONEY on their students if we look at tuition alone.

In [11]:
```
colleges.loc[colleges['expense ratio']==np.max(colleges['expense ratio'])]
```

Out[11]:

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.B
University of Alabama at Birmingham	No	1797	1260	938	24	35	6960	4698	4440	

So the best "bang for your buck" award goes to...**drum roll**...University of Alabama at Birmingham! However, a more cynical way of looking at it is if UAB's spending 4x the out-of-state tuition on every student, then it might not be in business much longer...so if you're in the HS class of 2021, get in while they're still afloat, but hold tight to your wallet because the student callers will be working overtime to snag your donation dollars!

Out of curiosity, let's also figure out who the biggest thief in higher education is: cue the college with the lowest expense ratio!

In [13]:

```
colleges.loc[colleges['expense ratio']==np.min(colleges['expense ratio'])]
```

Out[13]:

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.B
Lindenwood College	Yes	810	484	356	6	33	2155	191	9200	

Visualizations

It's time to create some data visualizations!

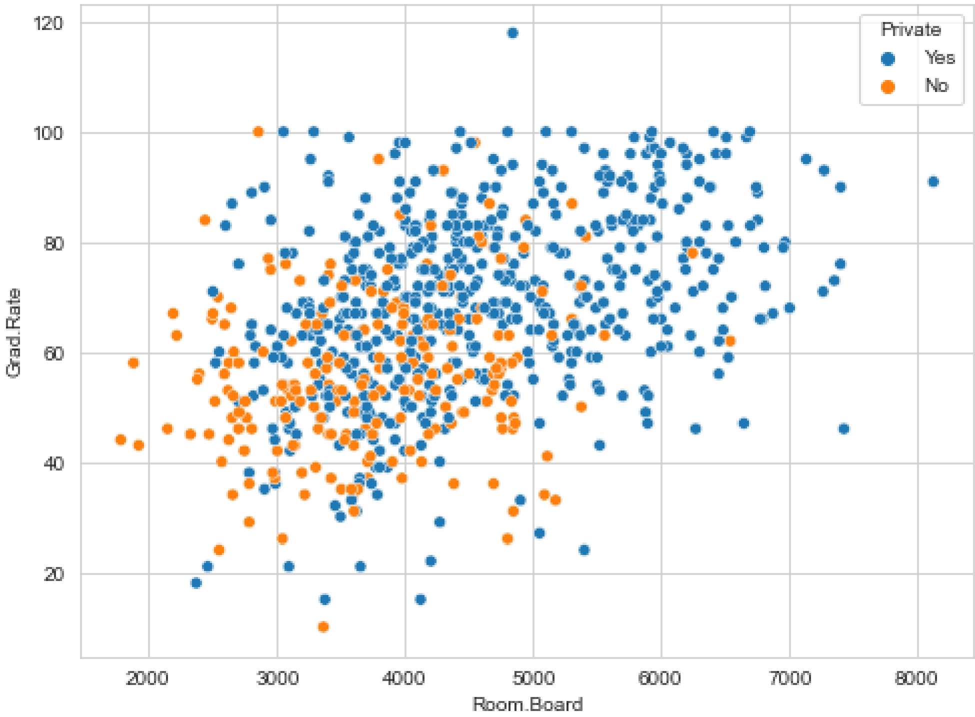
The first hypotheses I'll make is that private universities tend to have higher tuition and higher room and board on account of having generally nicer facilities.

In [14]:

```
plt.figure(figsize=(8,6))
sns.set_style('whitegrid')
sns.scatterplot(colleges['Room.Board'],colleges['Grad.Rate'],hue=colleges['Private'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the f
ollowing variables as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will result in an error
or misinterpretation.
  warnings.warn(
```

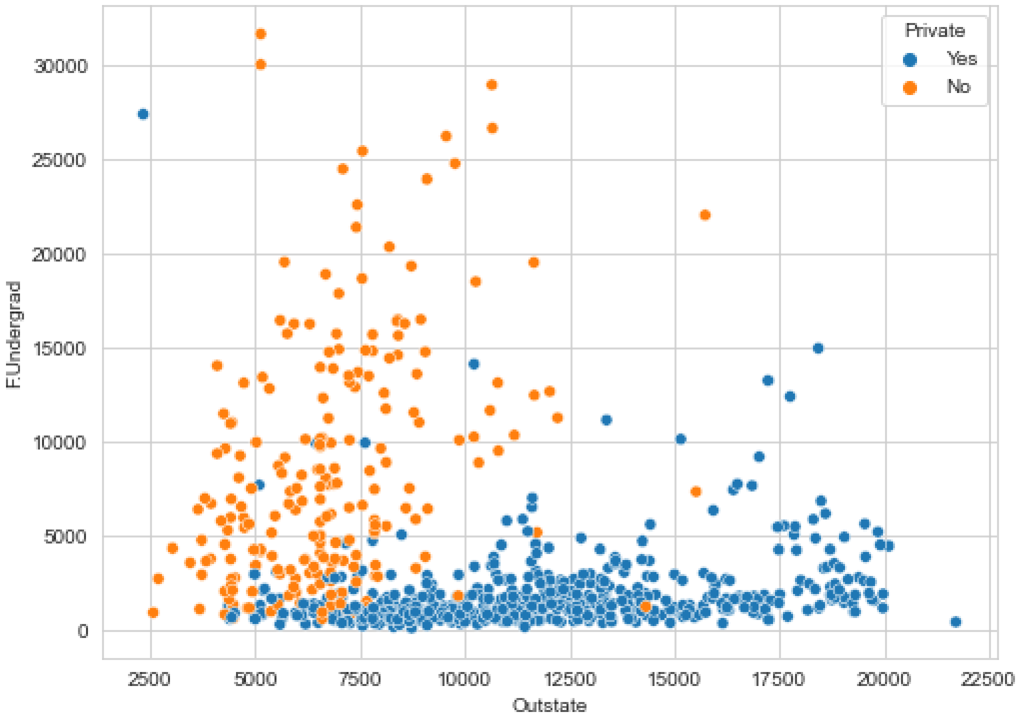
Out[14]: `<AxesSubplot:xlabel='Room.Board', ylabel='Grad.Rate'>`



In [15]:

```
plt.figure(figsize=(8,6))
sns.scatterplot(data=colleges,x='Outstate',y='F.Undergrad',hue='Private')
```

Out[15]: <AxesSubplot:xlabel='Outstate', ylabel='F.Undergrad'>

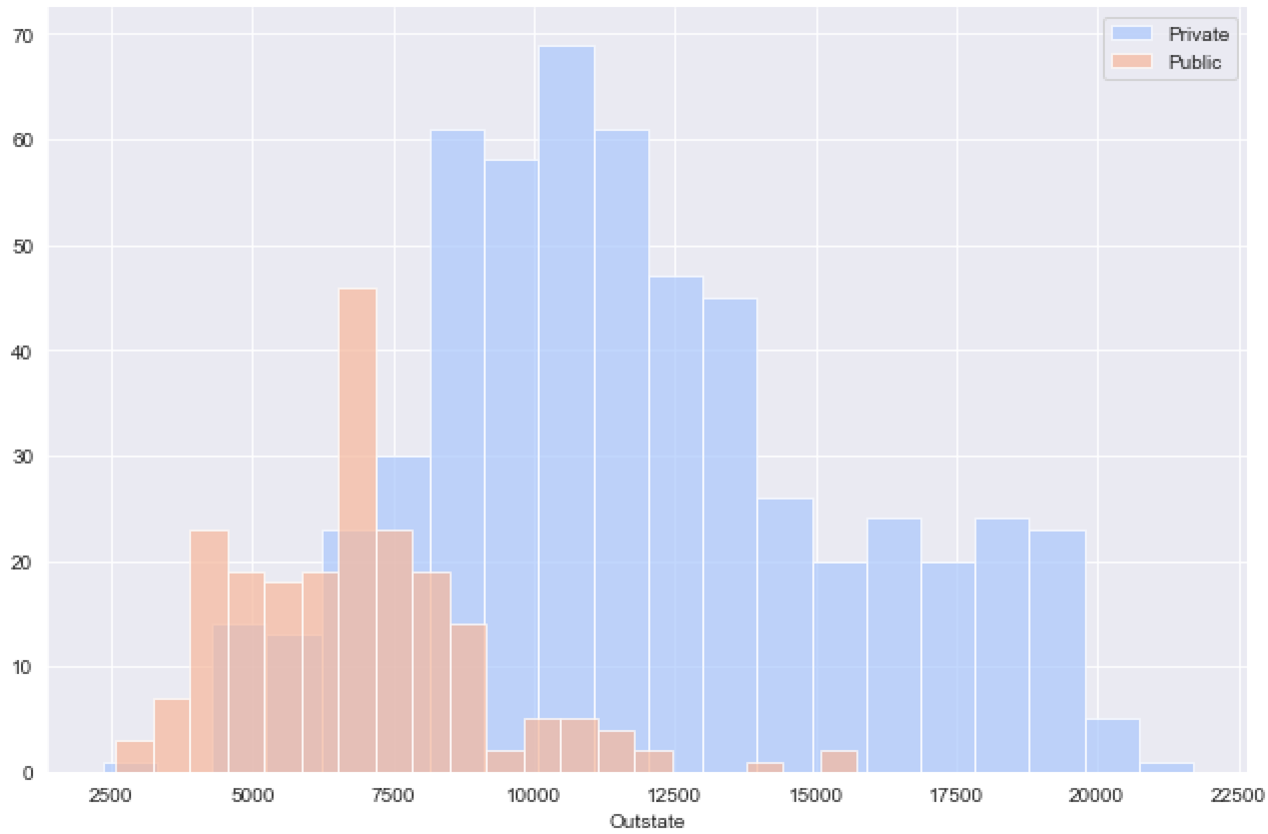


VERY interesting how the school's undergrad population is almost completely uncorrelated with tuition, which begs the question: where is that money going?

In [16]:

```
sns.set_style('darkgrid')
g = sns.FacetGrid(colleges,hue='Private',palette='coolwarm',height=6,aspect=1.5)
g = g.map(plt.hist,'Outstate',bins=20,alpha=0.7)
plt.legend(labels=['Private','Public'])
```

Out[16]: <matplotlib.legend.Legend at 0x20ae8ac88e0>

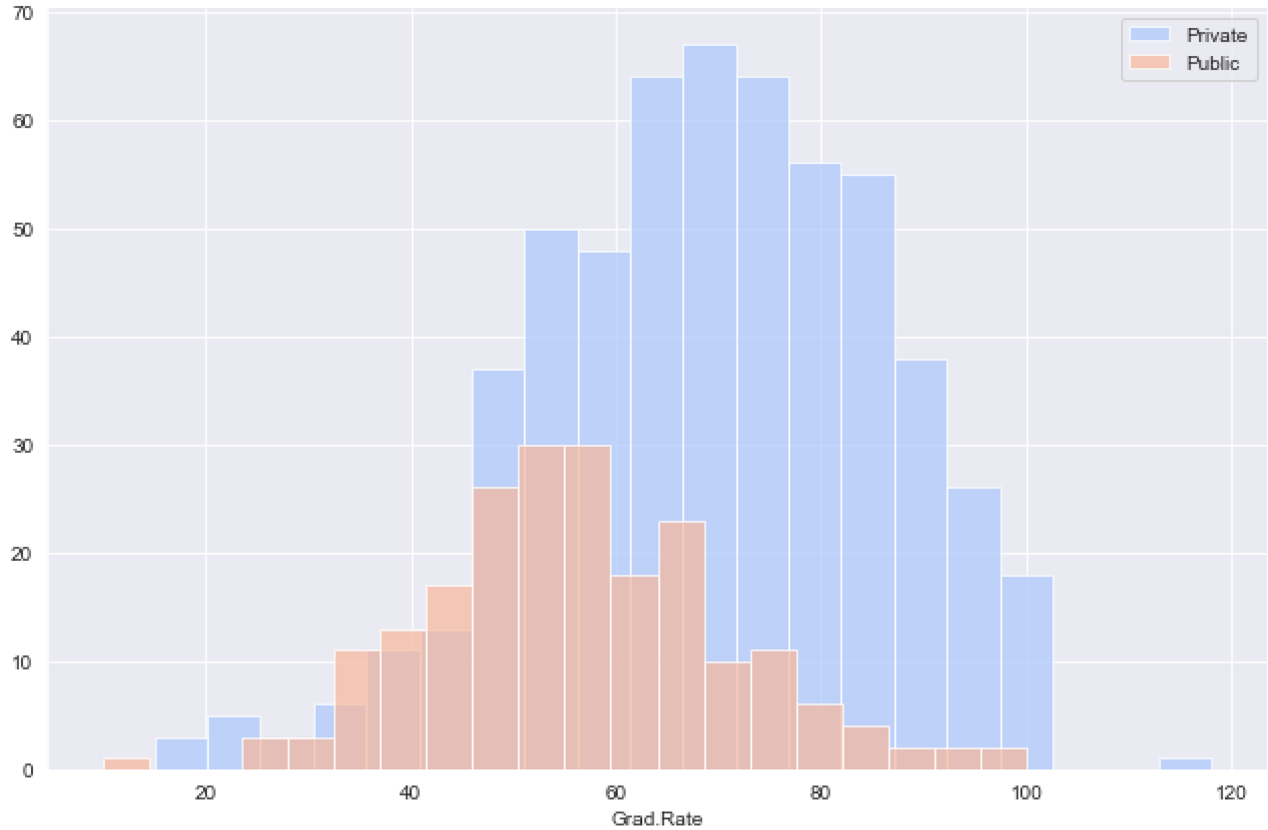


In [17]:

```
grid = sns.FacetGrid(colleges,hue='Private',palette='coolwarm',height=6,aspect=1.5)
grid = grid.map(plt.hist,'Grad.Rate',bins=20,alpha=0.7)
plt.legend(labels=['Private','Public'])
```

Out[17]:

```
<matplotlib.legend.Legend at 0x20ae8e708e0>
```



As expected, out-of-state tuition is higher for private schools, and so is graduation rate - perhaps because if a student is investing that much money into their education, their parents are thinking, "You better finish...or else!"

Now back to that school with a graduation rate of higher than 100%.

In [18]:

```
colleges.loc[colleges['Grad.Rate'] > 100]
```

Out[18]:

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Boa
Cazenovia College	Yes	3847	3433	527	9	35	1010	12	9384	48

Maybe this school pulled a fast one and counted double majors as having graduated twice? In reality, this makes no sense, so I'm going to change this graduation rate to 100%.

In [19]:

```
colleges['Grad.Rate']['Cazenovia College']=100
```

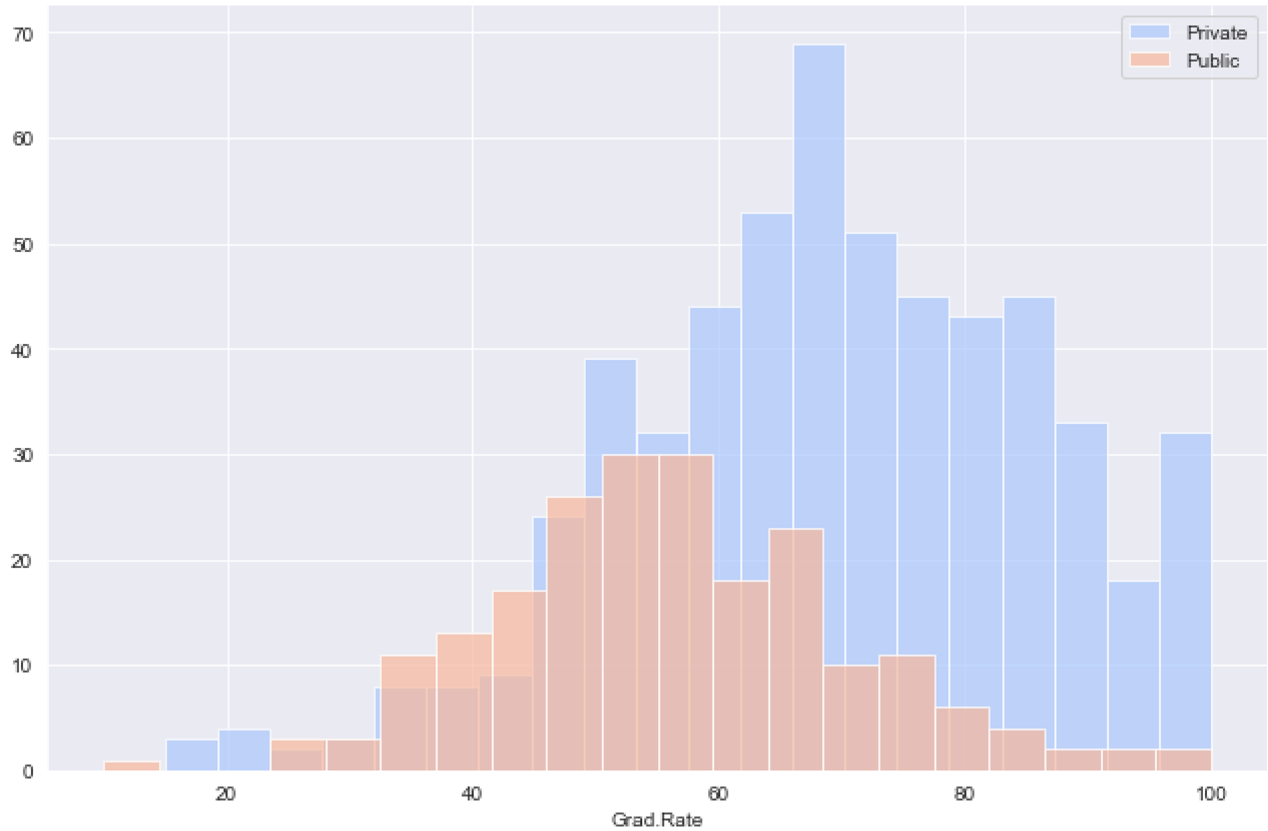
```
<ipython-input-19-37f846747b86>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/i
ndexing.html#returning-a-view-versus-a-copy
  colleges['Grad.Rate']['Cazenovia College']=100
```

In [20]:

```
grid = sns.FacetGrid(colleges,hue='Private',palette='coolwarm',height=6,aspect=1.5)
grid = grid.map(plt.hist,'Grad.Rate',bins=20,alpha=0.7)
plt.legend(labels=['Private','Public'])
```

Out[20]: <matplotlib.legend.Legend at 0x20ae91835b0>



Much better! Now on to the moment you've all been waiting for: K-means clustering!

K-Means Cluster Creation

First we'll import KMeans from Scikit-learn, the most popular package for machine learning in Python. The way K-means works is the following:

1) **Pick the number of clusters.** Whether you choose 2 or 4 or 10 is dictated by the domain of the specific problem. In this case, we're sorting into public and private schools, so we will choose the number of clusters to be 2. If you were analyzing genetic variations in a population and knew beforehand there were 7 known

variants, then you would want to choose 7. 2) **Randomly assign each data point to a category.** 3) **Take the centroid of the data points in each category.** For those of you who think a centroid is some cool-sounding type of asteroid, it's calculus-speak for "the average of all the data in each cluster." You already understand this intuitively for 1-D data: the average price of grocery items gets you a single number like $52. If you calculate the average price AND average quantity of grocery items (2 dimensions), you'll get twor
52 and 2 items. In this dataset, we have 18 features, so each centroid corresponds to a 18-D set of coordinates. And no, we're not going to visualize this. 4) **Re-assign each data point to the category corresponding to the nearest centroid.** 5) **Repeat Steps 3 and 4 until there are no more changes in category.**

Here's a great image to help illustrate: [insert one for notes]

In [21]:

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=2)
km.fit(colleges.drop('Private',axis=1))
```

Out[21]:

```
KMeans(n_clusters=2)
```

One end result of K means clustering works is a mathematical representation (set of coordinates) of the centroid of each final cluster. Calling km.cluster*centers* gives an array of 2 arrays, each one corresponding to the centroid of one cluster.

In [22]:

```
km.cluster_centers_
```

Out[22]:

```
array([[1.03631389e+04, 6.55089815e+03, 2.56972222e+03, 4.14907407e+01,
        7.02037037e+01, 1.30619352e+04, 2.46486111e+03, 1.07191759e+04,
        4.64347222e+03, 5.95212963e+02, 1.71420370e+03, 8.63981481e+01,
        9.13333333e+01, 1.40277778e+01, 2.00740741e+01, 1.41705000e+04,
        6.75925926e+01, 1.30734947e+00],
       [1.81323468e+03, 1.28716592e+03, 4.91044843e+02, 2.53094170e+01,
        5.34708520e+01, 2.18854858e+03, 5.95458894e+02, 1.03957085e+04,
        4.31136472e+03, 5.41982063e+02, 1.28033632e+03, 7.04424514e+01,
        7.78251121e+01, 1.40997010e+01, 2.31748879e+01, 8.93204634e+03,
        6.50926756e+01, 9.02321069e-01]])
```

Evaluation

There is no perfect way to evaluate clustering if you don't have the labels. However, in this case, the colleges data set told us whether each school was public or private, so we can cross-validate our K-means model with these labels to compare the performance of supervised and unsupervised models in general.

First, we need to convert our "Private: Yes or No?" column into 0s and 1s that the K-means model can understand.

In [23]:

```
def convertToCluster(cluster):
    if cluster=='Yes':
        return 1
    else:
        return 0
colleges['Cluster'] = colleges['Private'].apply(convertToCluster)
```

Two quick ways to evaluate the performance of a machine learning model is to look at a confusion matrix and a classification report. I won't go into all the details of these here, so check out the Wikipedia pages to learn more.

In [32]:

```
from sklearn.metrics import classification_report,confusion_matrix
```

```python
from sklearn.metrics import accuracy_score

print('Accuracy: {}'.format(accuracy_score(colleges['Cluster'],km.labels_)))
print('confusion_matrix: \n{}'.format(confusion_matrix(colleges['Cluster'],km.labels_)))
print(classification_report(colleges['Cluster'],km.labels_))
```

```
Accuracy: 0.7786357786357786
confusion_matrix:
[[ 74 138]
 [ 34 531]]
              precision    recall  f1-score   support

           0       0.69      0.35      0.46       212
           1       0.79      0.94      0.86       565

    accuracy                           0.78       777
   macro avg       0.74      0.64      0.66       777
weighted avg       0.76      0.78      0.75       777
```

In [ ]:

```python

```