2

Explain whether each scenario is a classification or regression problem, and indicate whether we are most interested in inference or prediction. Finally, provide n and p.

(a) We collect a set of data on the top 500 firms in the US. For each firm we record profit, number of employees, industry and the CEO salary. We are interested in understanding which factors affect CEO salary.

- This is a regression problem with n = 500 and p = 3. (target var is salary)

(b) We are considering launching a new product and wish to know whether it will be a success or a failure. We collect data on 20 similar products that were previously launched. For each prod- uct we have recorded whether it was a success or failure, price charged for the product, marketing budget, competition price, and ten other variables.

- This is a classification problem with n = 20 and p = 13. (target var is success/failure)

(c) We are interested in predicting the % change in the USD/Euro exchange rate in relation to the weekly changes in the world stock markets. Hence we collect weekly data for all of 2012. For each week we record the % change in the USD/Euro, the % change in the US market, the % change in the British market, and the % change in the German market.

- This is a regression problem with n = 52 and p = 3. (target var is % change in USD/Euro)

5

What are the advantages and disadvantages of a very flexible (versus a less flexible) approach for regression or classification? Under what circumstances might a more flexible approach be preferred to a less flexible approach? When might a less flexible approach be preferred?

- Advantages of a flexible approach are that it can fit a wider range of possible shapes of the data. Disadvantages are that it can overfit the data and be more computationally expensive. A more flexible approach might be preferred when the data is more complex and a less flexible approach might be preferred when the data is more simple (i.e. non-linear vs linear relationships).

6

Describe the differences between a parametric and a non-parametric statistical learning approach. What are the advantages of a parametric approach to regression or classification (as opposed to a non-parametric approach)? What are its disadvantages?

- Parametric statistical learning approaches assume a functional form for f(X) and estimate the parameters of that function. Non-parametric statistical learning approaches do not assume a functional form for f(X) and instead estimate f(X) directly from the data. The advantages of a parametric approach are that it is simpler and more interpretable. The disadvantages are that it can lead to a poor fit if the functional form is not correct.
- Example of parametric approach: linear regression
- Example of non-parametric approach: k-nearest neighbors or SVM

8

This exercise relates to the College data set, which can be found in the file College.csv on the book website. It contains a number of variables for 777 different universities and colleges in the US.

(a) Use the pd.read_csv() function to read the data into Python. Call the loaded data college. Make sure that you have the directory set to the correct location for the data.

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
college_df = pd.read_csv('data/College.csv')
college_df
```

Out[ ]:

	Unnamed: 0	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.
0	Abilene Christian University	Yes	1660	1232	721	23	52	2885	
1	Adelphi University	Yes	2186	1924	512	16	29	2683	
2	Adrian College	Yes	1428	1097	336	22	50	1036	
3	Agnes Scott College	Yes	417	349	137	60	89	510	
4	Alaska Pacific University	Yes	193	146	55	16	44	249	
...	...	...	...	...	...	...	...	...	
772	Worcester State College	No	2197	1515	543	4	26	3089	
773	Xavier University	Yes	1959	1805	695	24	47	2849	
774	Xavier University of Louisiana	Yes	2097	1915	695	34	61	2793	
775	Yale University	Yes	10705	2453	1317	95	99	5217	
776	York College of Pennsylvania	Yes	2989	1855	691	28	63	2988	

777 rows × 19 columns

(b) Look at the data used in the notebook by creating and running a new cell with just the code college in it. You should notice that the first column is just the name of each university in a column named something like Unnamed: 0. We don't really want pandas to treat this as data. However, it may be handy to have these names for later. Try the following commands and similarly look at the resulting data frames:

```
college2 = pd.read_csv('College.csv', index_col =0)
college3 = college.rename ({'Unnamed: 0': 'College '}, axis
=1)
college3 = college3.set_index('College ')
```

In [ ]:

```
college2 = pd.read_csv('data/College.csv', index_col =0)
college3 = college2.rename ({'Unnamed: 0': 'College'}, axis =1)
# college3 = college3.set_index('College')
```

This has used the first column in the file as an index for the data frame. This means that pandas has given each row a name corresponding to the appropriate university. Now you should see that the first data column is Private. Note that the names of the colleges appear on the left of the table. We also introduced a new python object above: a dictionary, which is specified by (key, value) pairs. Keep your modified version of the data with the following:

```
college = college3
```

In [ ]:
```
college = college3
college
```

Out[ ]:

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Unde
Abilene Christian University	Yes	1660	1232	721	23	52	2885	
Adelphi University	Yes	2186	1924	512	16	29	2683	
Adrian College	Yes	1428	1097	336	22	50	1036	
Agnes Scott College	Yes	417	349	137	60	89	510	
Alaska Pacific University	Yes	193	146	55	16	44	249	
...	...	...	...	...	...	...	...	
Worcester State College	No	2197	1515	543	4	26	3089	
Xavier University	Yes	1959	1805	695	24	47	2849	
Xavier University of Louisiana	Yes	2097	1915	695	34	61	2793	
Yale University	Yes	10705	2453	1317	95	99	5217	
York College of Pennsylvania	Yes	2989	1855	691	28	63	2988	

777 rows × 18 columns

(c) Use the describe() method of to produce a numerical summary of the variables in the data set.

In [ ]:
```
college.describe()
```

Out[ ]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad
count	777.000000	777.000000	777.000000	777.000000	777.000000	777.000000
mean	3001.638353	2018.804376	779.972973	27.558559	55.796654	3699.907336
std	3870.201484	2451.113971	929.176190	17.640364	19.804778	4850.420531
min	81.000000	72.000000	35.000000	1.000000	9.000000	139.000000
25%	776.000000	604.000000	242.000000	15.000000	41.000000	992.000000
50%	1558.000000	1110.000000	434.000000	23.000000	54.000000	1707.000000
75%	3624.000000	2424.000000	902.000000	35.000000	69.000000	4005.000000
max	48094.000000	26330.000000	6392.000000	96.000000	100.000000	31643.000000

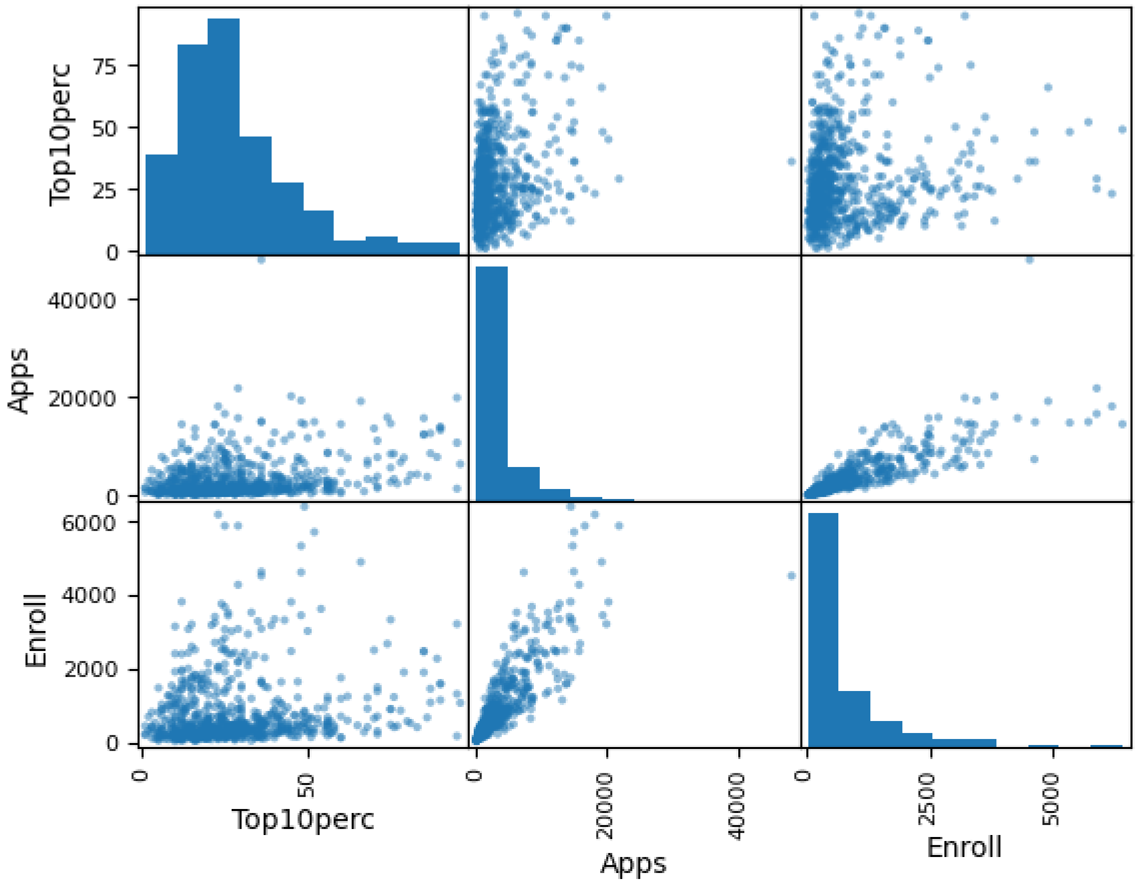
(d) Use the pd.plotting.scatter_matrix() function to produce a scatterplot matrix of the first columns [Top10perc, Apps, Enroll]. Recall that you can reference a list C of columns of a data frame A using A[C].

In [ ]:

```
pd.plotting.scatter_matrix(college[['Top10perc', 'Apps', 'Enroll']])
```

Out[ ]:

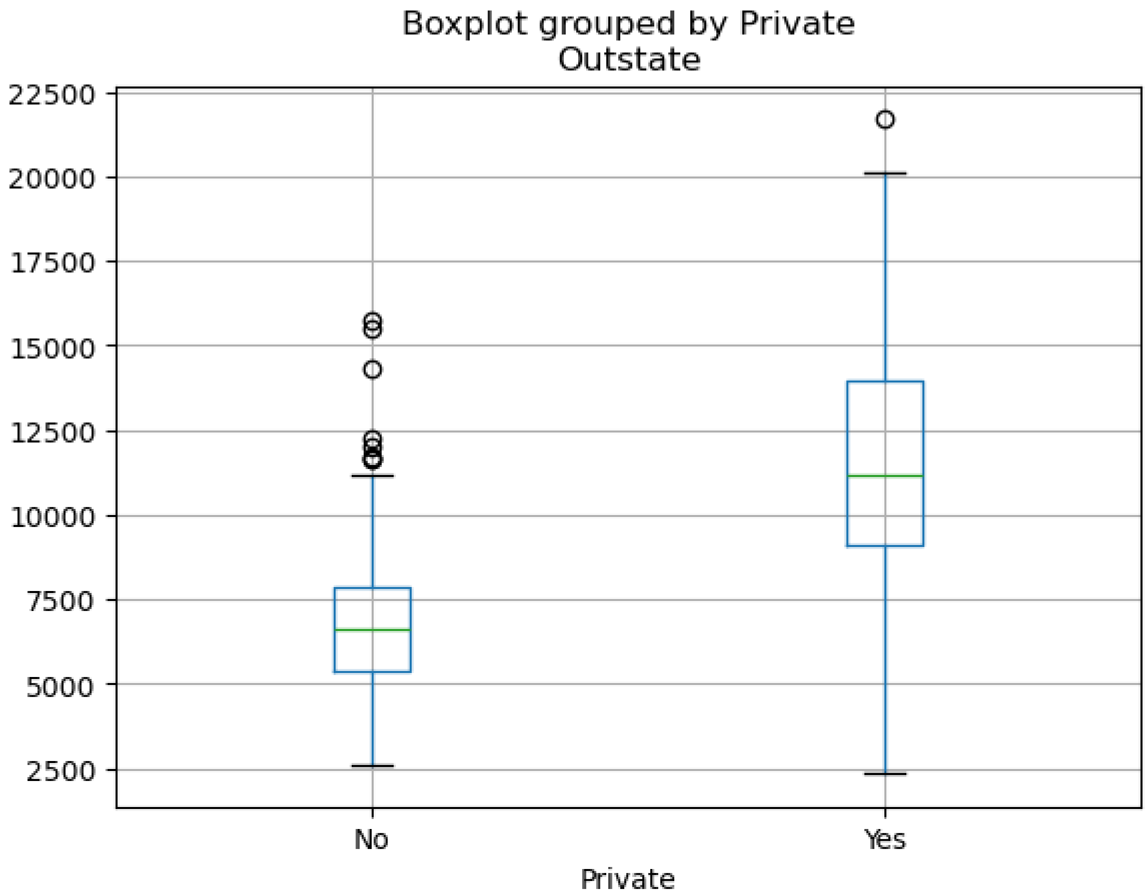
```
array([[<Axes: xlabel='Top10perc', ylabel='Top10perc'>,
        <Axes: xlabel='Apps', ylabel='Top10perc'>,
        <Axes: xlabel='Enroll', ylabel='Top10perc'>],
       [<Axes: xlabel='Top10perc', ylabel='Apps'>,
        <Axes: xlabel='Apps', ylabel='Apps'>,
        <Axes: xlabel='Enroll', ylabel='Apps'>],
       [<Axes: xlabel='Top10perc', ylabel='Enroll'>,
        <Axes: xlabel='Apps', ylabel='Enroll'>,
        <Axes: xlabel='Enroll', ylabel='Enroll'>]], dtype=object)
```



(e) Use the boxplot() method of college to produce side-by-side boxplots of Outstate versus Private.

In [ ]:
```
college.boxplot(column = 'Outstate', by = 'Private')
```

Out[ ]:
```
<Axes: title={'center': 'Outstate'}, xlabel='Private'>
```



(f) Create a new qualitative variable, called Elite, by binning the Top10perc variable into two groups based on whether or not the proportion of students coming from the top 10% of their high school classes exceeds 50%.

```
college['Elite '] = pd.cut(college['Top10perc '],
[0,0.5,1],
labels =['No', 'Yes'])
```

Use the value_counts() method of college['Elite'] to see how many elite universities there are. Finally, use the boxplot() method again to produce side-by-side boxplots of Outstate versus Elite.

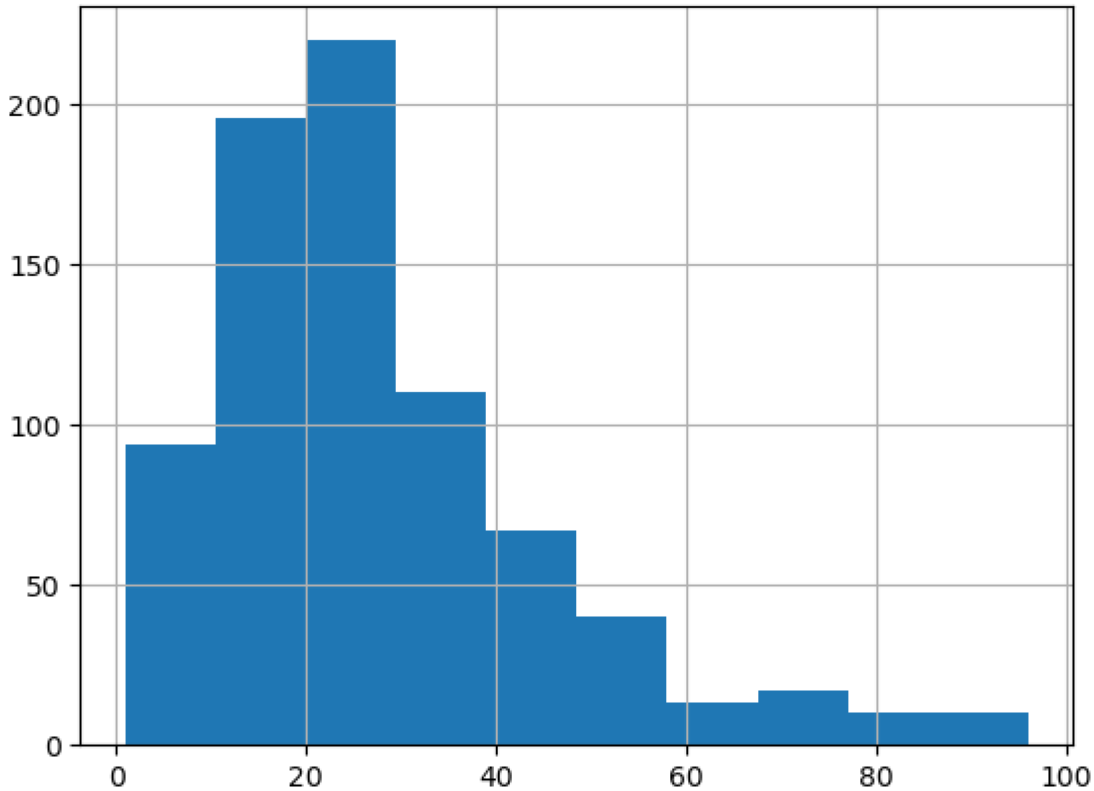
In [ ]:
```
# college['Top10perc_ratio'] = college['Top10perc'] / college['Enroll']
# college['Elite'] = college['Top10perc_ratio'] > 0.5
# # change to yes or no
# college['Elite'] = college['Elite'].replace({True: 'Yes', False: 'No'})
```

In [ ]:
```
college['Elite'].value_counts()
```

Out[ ]:
```
No     775
Yes      2
Name: Elite, dtype: int64
```

In [ ]:
```
college['Top10perc'].hist()
```

Out[ ]: <Axes: >

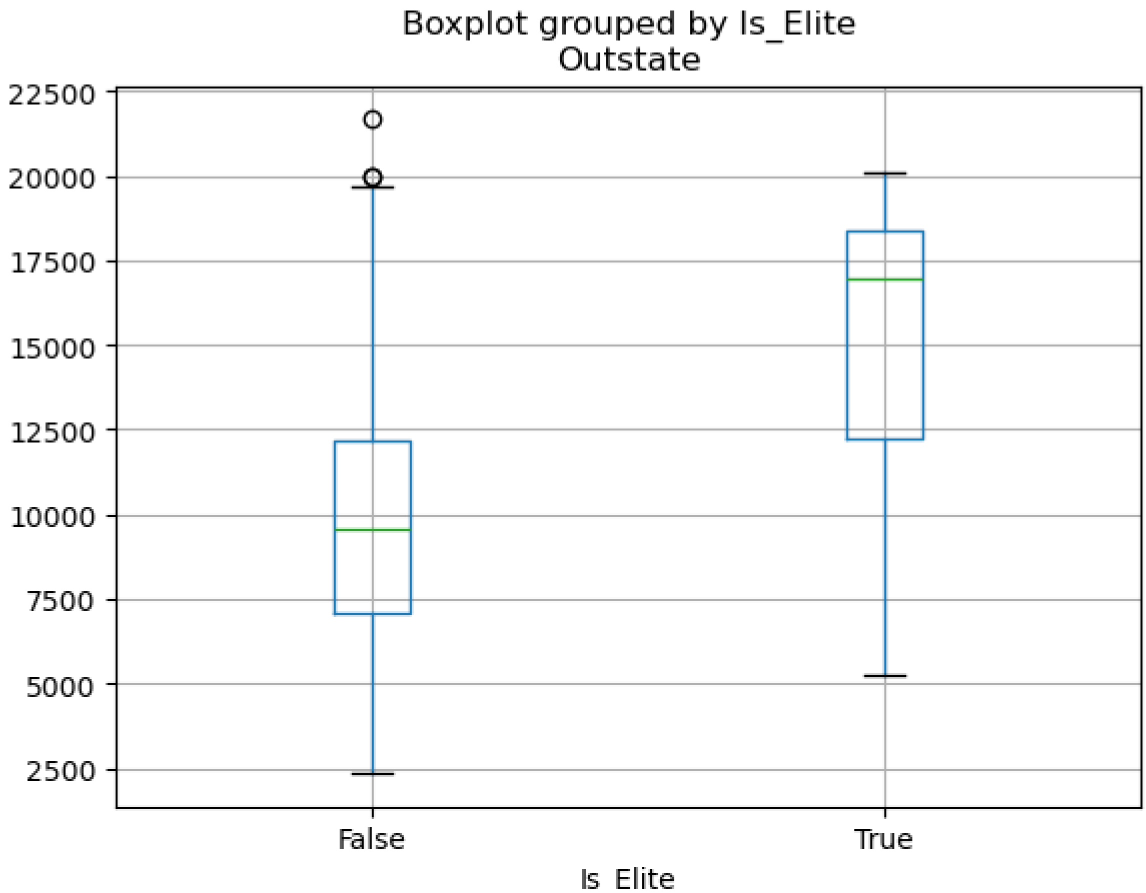


In [ ]:
```
# use pandas cut to create a binary variable, Elite, which is True if the pr
college['Is_Elite'] = pd.cut(college['Top10perc'], [0,50,100], labels = [Fal
college['Is_Elite'].value_counts()
```

Out[ ]:
```
False    699
True      78
Name: Is_Elite, dtype: int64
```

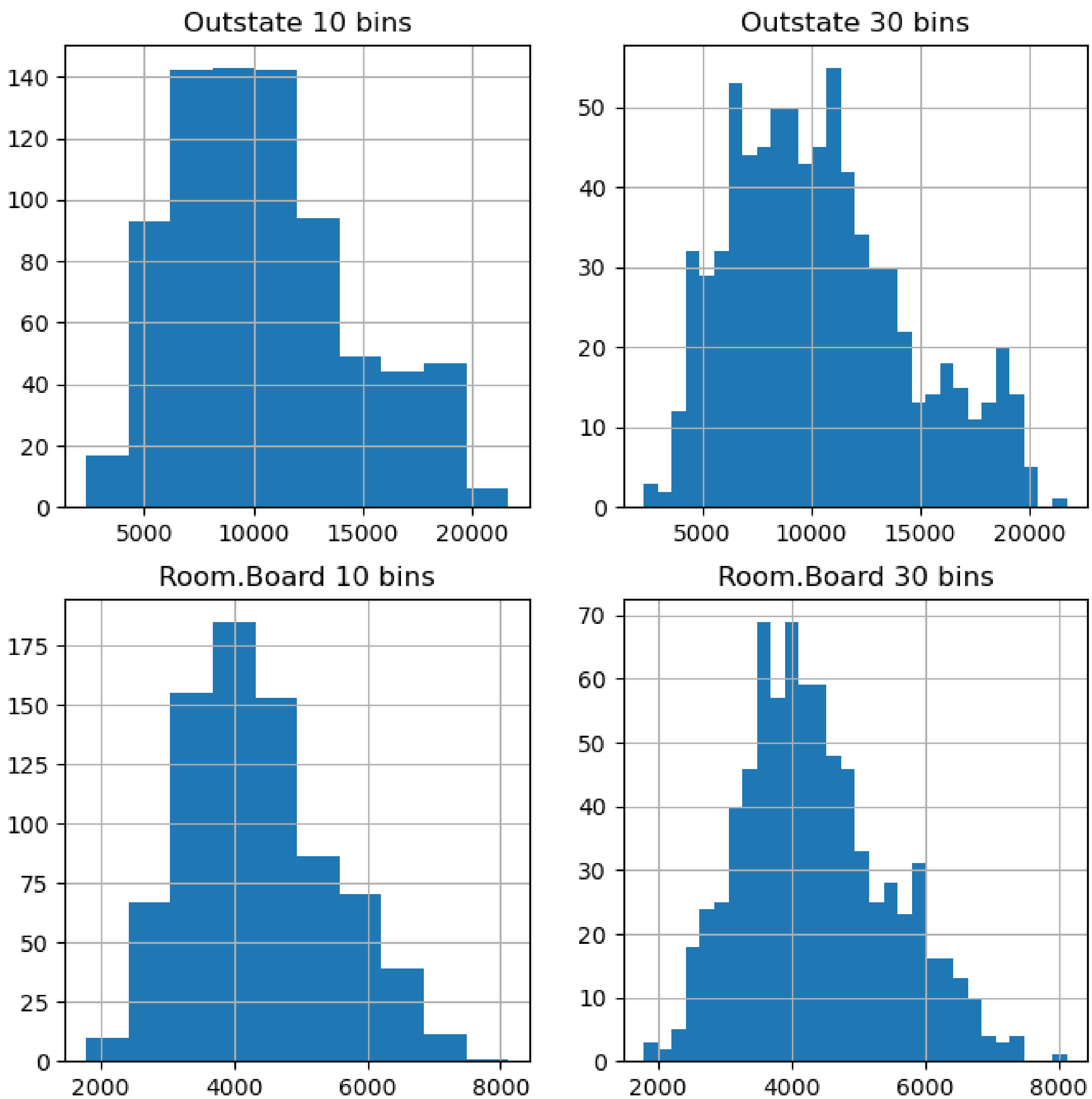
In [ ]:
```
college.boxplot(column = 'Outstate', by = 'Is_Elite')
```

Out[ ]: <Axes: title={'center': 'Outstate'}, xlabel='Is_Elite'>



(g) Use the plot.hist() method of college to produce some histograms with differing numbers of bins for a few of the quantitative variables. The command plt.subplots(2, 2) may be useful: it will divide the plot window into four regions so that four plots can be made simultaneously. By changing the arguments you can divide the screen up in other combinations.

In [ ]:
```
fig, ax = plt.subplots(2,2)
# change size
fig.set_size_inches(8,8)
college['Outstate'].hist(ax = ax[0,0], bins = 10)
college['Outstate'].hist(ax = ax[0,1], bins = 30)
college['Room.Board'].hist(ax = ax[1,0], bins = 10)
college['Room.Board'].hist(ax = ax[1,1], bins = 30)
# set titles
ax[0,0].set_title('Outstate 10 bins')
ax[0,1].set_title('Outstate 30 bins')
ax[1,0].set_title('Room.Board 10 bins')
ax[1,1].set_title('Room.Board 30 bins')
plt.show()
```

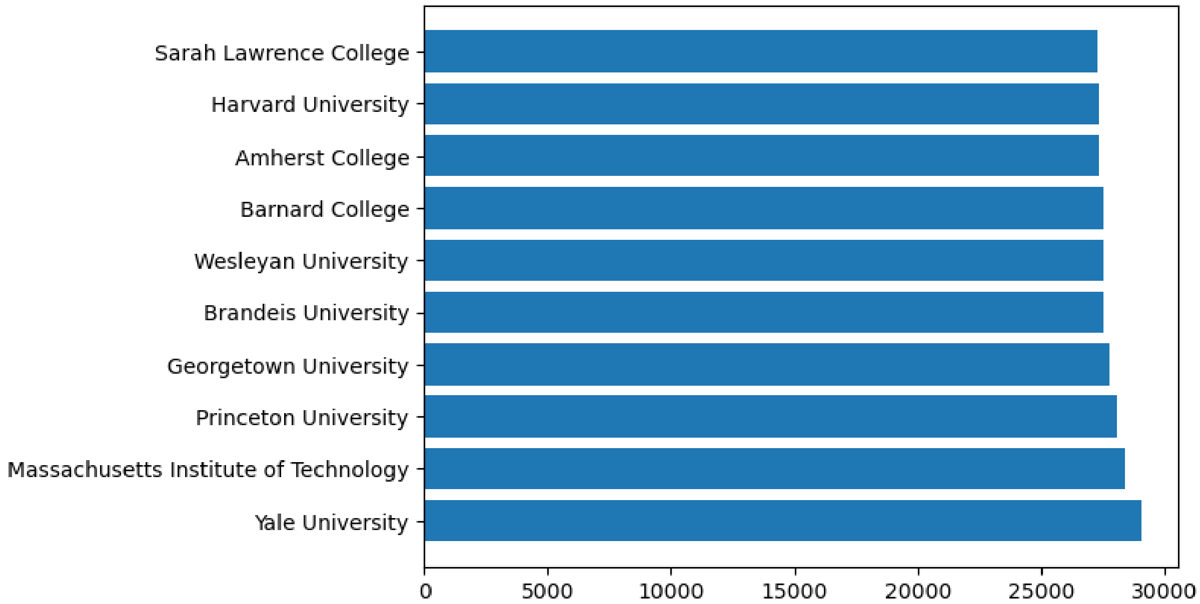


(h) Continue exploring the data, and provide a brief summary of what you discover.

In [ ]:
```
college['Estimated_Total_Cost'] = college['Outstate'] + college['Room.Board'
college['Acceptance_Rate'] = college['Accept'] / college['Apps']
college['Spend_to_Cost_Ratio'] = college['Expend'] / college['Estimated_Tota
```

In [ ]:
```
# show top 10 colleges with highest Estimated Total Cost
highest_cost_top10 = college.sort_values('Estimated_Total_Cost', ascending =
# plot top 10 colleges with highest Estimated Total Cost, name and Estimatec
plt.barh(highest_cost_top10.index, highest_cost_top10['Estimated_Total_Cost'
```

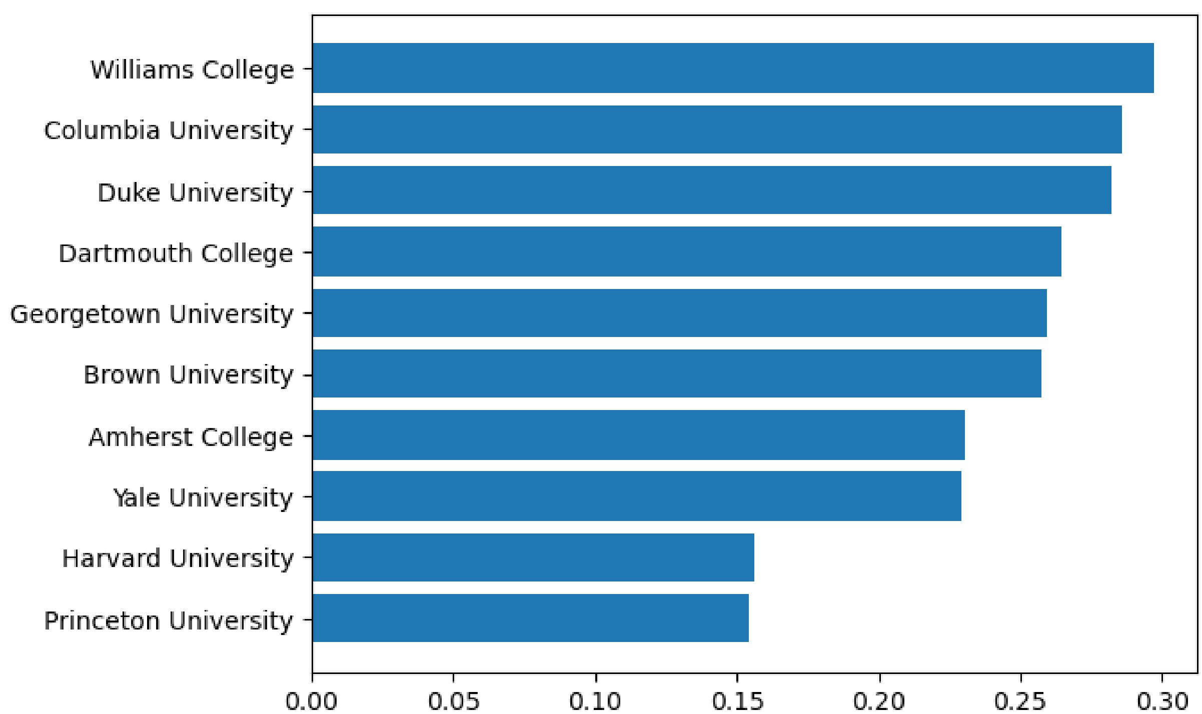
Out[ ]:
```
<BarContainer object of 10 artists>
```



The above table shows the top 10 most expensive total cost colleges in the dataset. Unsurprisingly the top 10 has multiple Ivy League schools.

In [ ]:
```
# show colleges with lowest Acceptance Rate
acceptance_bottom10 = college.sort_values('Acceptance_Rate').head(10)
# plot colleges with lowest Acceptance Rate, name and Acceptance Rate, horiz
plt.barh(acceptance_bottom10.index, acceptance_bottom10['Acceptance_Rate'])
```

Out[ ]: <BarContainer object of 10 artists>



Acceptance rates sort of line up with highest cost.

In [ ]:

```
# show colleges with highest Spend to Cost Ratio
spend_to_cost_top10 = college.sort_values('Spend_to_Cost_Ratio', ascending =
# plot colleges with highest Spend to Cost Ratio, name and Spend to Cost Rat
plt.barh(spend_to_cost_top10.index, spend_to_cost_top10['Spend_to_Cost_Ratic
```

Out[ ]:

```
<BarContainer object of 10 artists>
```

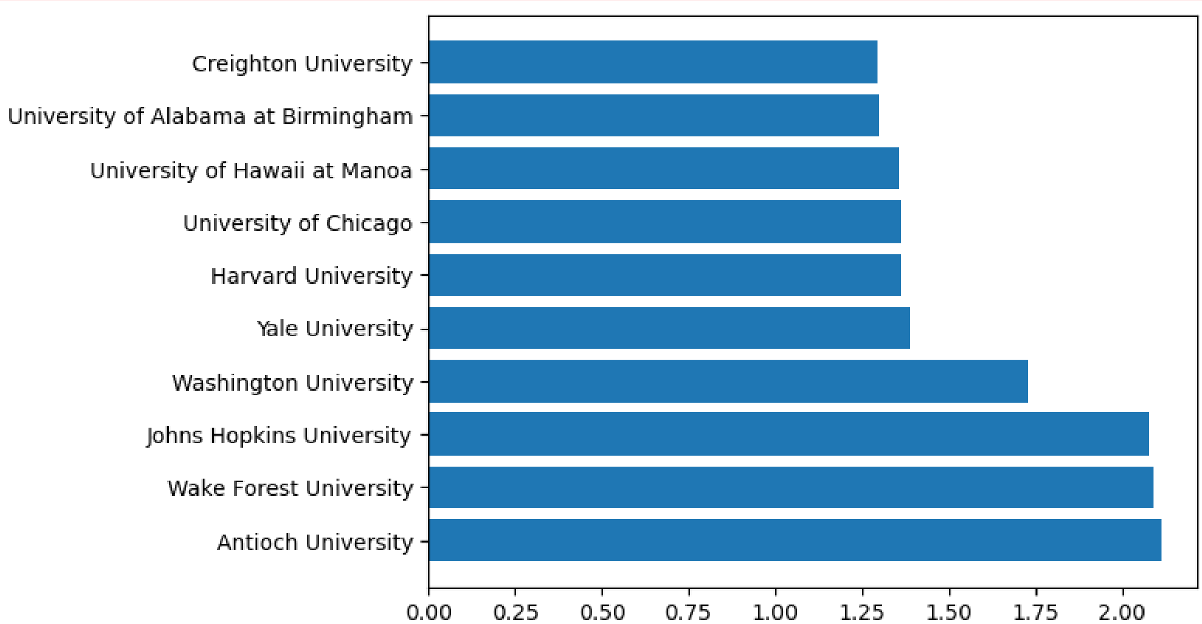
```
Error in callback <function _draw_all_if_interactive at 0x7f26aeba6290> (for
post_execute):
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/pyplo
t.py:197, in _draw_all_if_interactive()
    195 def _draw_all_if_interactive() -> None:
    196     if matplotlib.is_interactive():
--> 197         draw_all()

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/_pyla
b_helpers.py:132, in Gcf.draw_all(cls, force)
    130 for manager in cls.get_all_fig_managers():
    131     if force or manager.canvas.figure.stale:
--> 132         manager.canvas.draw_idle()

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/backe
nd_bases.py:1893, in FigureCanvasBase.draw_idle(self, *args, **kwargs)
   1891 if not self._is_idle_drawing:
   1892     with self._idle_draw_cntx():
-> 1893         self.draw(*args, **kwargs)

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/backe
nds/backend_agg.py:388, in FigureCanvasAgg.draw(self)
    385 # Acquire a lock on the shared font cache.
    386 with (self.toolbar._wait_cursor_for_draw_cm() if self.toolbar
    387       else nullcontext()):
--> 388     self.figure.draw(self.renderer)
    389     # A GUI class may be need to update a window using this draw, so
    390     # don't forget to call the superclass.
    391     super().draw()

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/artis
t.py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer,
*args, **kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/artis
t.py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/figur
e.py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)
```

```
File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/imag
e.py:132, in _draw_list_compositing_images(renderer, parent, artists, suppre
ss_composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/artis
t.py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/axes/
_base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/imag
e.py:132, in _draw_list_compositing_images(renderer, parent, artists, suppre
ss_composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/artis
t.py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/axis.
py:1387, in Axis.draw(self, renderer, *args, **kwargs)
   1384     return
   1385 renderer.open_group(__name__, gid=self.get_gid())
-> 1387 ticks_to_draw = self._update_ticks()
   1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/axis.
py:1277, in Axis._update_ticks(self)
   1275 major_locs = self.get_majorticklocs()
```

```
   1276 major_labels = self.major.formatter.format_ticks(major_locs)
-> 1277 major_ticks = self.get_major_ticks(len(major_locs))
   1278 for tick, loc, label in zip(major_ticks, major_locs, major_labels):
   1279     tick.update_position(loc)

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/axis.
py:1626, in Axis.get_major_ticks(self, numticks)
   1622     numticks = len(self.get_majorticklocs())
   1624 while len(self.majorTicks) < numticks:
   1625     # Update the new tick label properties from the old.
-> 1626     tick = self._get_tick(major=True)
   1627     self.majorTicks.append(tick)
   1628     self._copy_tick_props(self.majorTicks[0], tick)

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/axis.
py:1562, in Axis._get_tick(self, major)
   1558     raise NotImplementedError(
   1559         f"The Axis subclass {self.__class__.__name__} must define "
   1560         "_tick_class or reimplement _get_tick()")
   1561 tick_kw = self._major_tick_kw if major else self._minor_tick_kw
-> 1562 return self._tick_class(self.axes, 0, major=major, **tick_kw)

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/axis.
py:470, in YTick.__init__(self, *args, **kwargs)
    469 def __init__(self, *args, **kwargs):
--> 470     super().__init__(*args, **kwargs)
    471     # x in axes coords, y in data coords
    472     ax = self.axes

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/axis.
py:182, in Tick.__init__(self, axes, loc, size, width, color, tickdir, pad,
labelsize, labelcolor, labelfontfamily, zorder, gridOn, tick1On, tick2On, la
bel1On, label2On, major, labelrotation, grid_color, grid_linestyle, grid_lin
ewidth, grid_alpha, **kwargs)
    176 self.gridline.get_path()._interpolation_steps = \
    177     GRIDLINE_INTERPOLATION_STEPS
    178 self.label1 = mtext.Text(
    179     np.nan, np.nan,
    180     fontsize=labelsize, color=labelcolor, visible=label1On,
    181     fontfamily=labelfontfamily, rotation=self._labelrotation[1])
--> 182 self.label2 = mtext.Text(
    183     np.nan, np.nan,
    184     fontsize=labelsize, color=labelcolor, visible=label2On,
    185     fontfamily=labelfontfamily, rotation=self._labelrotation[1])
    187 self._apply_tickdir(tickdir)
    189 for artist in [self.tick1line, self.tick2line, self.gridline,
    190                self.label1, self.label2]:

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/text.
py:136, in Text.__init__(self, x, y, text, color, verticalalignment, horizon
talalignment, multialignment, fontproperties, rotation, linespacing, rotatio
n_mode, usetex, wrap, transform_rotates_text, parse_math, antialiased, **kwa
rgs)
    104 def __init__(self,
    105              x=0, y=0, text='', *,
    106              color=None,           # defaults to rc params
```

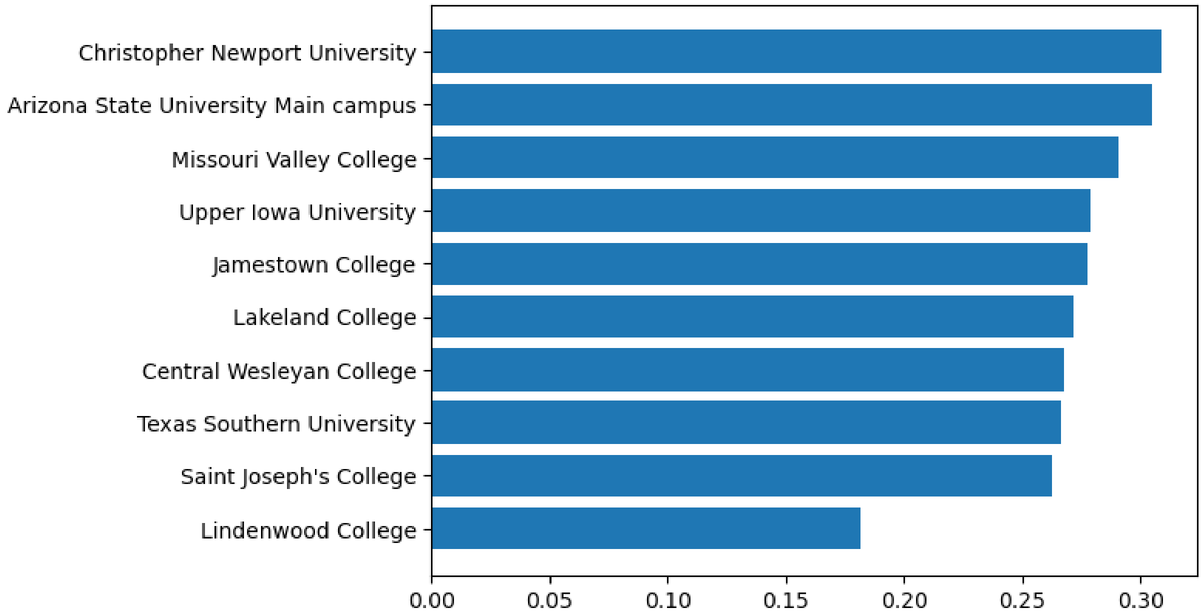
```
   (...)
    119             **kwargs
    120             ):
    121     """
    122     Create a `.Text` instance at *x*, *y* with string *text*.
    123 
   (...)
    134     %(Text:kwdoc)s
    135     """
--> 136     super().__init__()
    137     self._x, self._y = x, y
    138     self._text = ''

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/artis
t.py:201, in Artist.__init__(self)
    198 # Normally, artist classes need to be queried for mouseover info if
and
    199 # only if they override get_cursor_data.
    200 self._mouseover = type(self).get_cursor_data != Artist.get_cursor_da
ta
--> 201 self._callbacks = cbook.CallbackRegistry(signals=["pchanged"])
    202 try:
    203     self.axes = None

File ~/miniconda3/envs/mathenv/lib/python3.10/site-packages/matplotlib/cboo
k.py:185, in CallbackRegistry.__init__(self, exception_handler, signals)
    183 self.exception_handler = exception_handler
    184 self.callbacks = {}
--> 185 self._cid_gen = itertools.count()
    186 self._func_cid_map = {}
    187 # A hidden variable that marks cids that need to be pickled.

KeyboardInterrupt: 
```



A higher spend to cost ratio indicates that a college is spending more money per student. These would be the schools you would probably want to go to if you want to get the most bang for your buck.

In [ ]:
```
# show colleges with lowest Spend to Cost Ratio
spend_to_cost_bottom10 = college.sort_values('Spend_to_Cost_Ratio').head(10)
# plot colleges with lowest Spend to Cost Ratio, name and Spend to Cost Rati
plt.barh(spend_to_cost_bottom10.index, spend_to_cost_bottom10['Spend_to_Cost
```

Out[ ]:
```
<BarContainer object of 10 artists>
```



The bottom 10 schools in terms of spend to cost ratio would be the schools you would want to avoid if you want to have a worthwhile investment.

9

This exercise involves the Auto data set studied in the lab. Make sure that the missing values have been removed from the data.

In [ ]:
```
auto_data = pd.read_csv('data/Auto.csv')
auto_data['horsepower'] = pd.to_numeric(auto_data['horsepower'], errors = 'c
auto_data.isna().sum()
```

Out[ ]:
```
mpg             0
cylinders       0
displacement    0
horsepower      5
weight          0
acceleration    0
year            0
origin          0
name            0
dtype: int64
```

In [ ]:
```
auto_data.dropna(inplace = True)
```

(a) Which of the predictors are quantitative, and which are quali- tative?

In [ ]:
```
auto_data.dtypes
```

Out[ ]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower      float64
weight            int64
acceleration    float64
year              int64
origin            int64
name             object
dtype: object
```

In [ ]:
```
auto_data.describe()
```

Out[ ]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	
count	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000	39
mean	23.445918	5.471939	194.411990	104.469388	2977.584184	15.541327	7
std	7.805007	1.705783	104.644004	38.491160	849.402560	2.758864	
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	7
25%	17.000000	4.000000	105.000000	75.000000	2225.250000	13.775000	7
50%	22.750000	4.000000	151.000000	93.500000	2803.500000	15.500000	7
75%	29.000000	8.000000	275.750000	126.000000	3614.750000	17.025000	7
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	8

It appears that all of the predictors are quantitative except for name. An argument could be made that year is qualitative as well. I am unsure about the origin variable.

(b) What is the range of each quantitative predictor? You can an- swer this using the min() and max() methods in numpy.

In [ ]:
```
print('mpg range: {}'.format(auto_data['mpg'].max() - auto_data['mpg'].min()
print('cylinders range: {}'.format(auto_data['cylinders'].max() - auto_data[
print('displacement range: {}'.format(auto_data['displacement'].max() - auto
print('horsepower range: {}'.format(auto_data['horsepower'].max() - auto_dat
print('weight range: {}'.format(auto_data['weight'].max() - auto_data['weigh
print('acceleration range: {}'.format(auto_data['acceleration'].max() - auto
print('year range: {}'.format(auto_data['year'].max() - auto_data['year'].mi
```

```
mpg range: 37.6
cylinders range: 5
displacement range: 387.0
horsepower range: 184.0
weight range: 3527
acceleration range: 16.8
year range: 12
```

(c) What is the mean and standard deviation of each quantitative predictor?

In [ ]:
```
print('mpg mean: {:.2f}, std: {:.2f}'.format(auto_data['mpg'].mean(), auto_d
print('cylinders mean: {:.2f}, std: {:.2f}'.format(auto_data['cylinders'].me
print('displacement mean: {:.2f}, std: {:.2f}'.format(auto_data['displacemen
print('horsepower mean: {:.2f}, std: {:.2f}'.format(auto_data['horsepower'].
print('weight mean: {:.2f}, std: {:.2f}'.format(auto_data['weight'].mean(),
print('acceleration mean: {:.2f}, std: {:.2f}'.format(auto_data['acceleratio
print('year mean: {:.2f}, std: {:.2f}'.format(auto_data['year'].mean(), auto
```

```
mpg mean: 23.45, std: 7.81
cylinders mean: 5.47, std: 1.71
displacement mean: 194.41, std: 104.64
horsepower mean: 104.47, std: 38.49
weight mean: 2977.58, std: 849.40
acceleration mean: 15.54, std: 2.76
year mean: 75.98, std: 3.68
```

(d) Now remove the 10th through 85th observations. What is the range, mean, and standard deviation of each predictor in the subset of the data that remains?

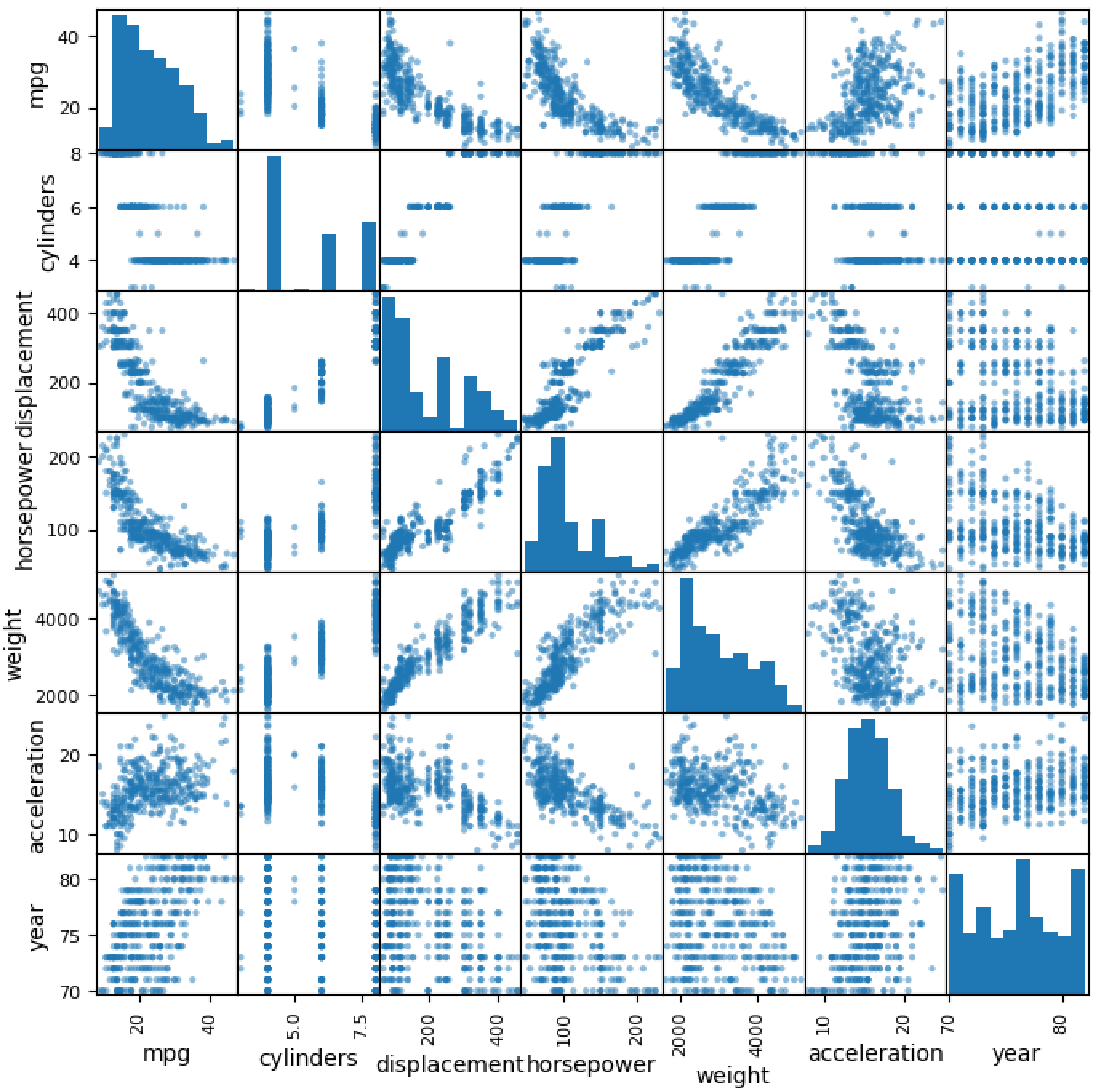
In [ ]:
```
# remove the 10th through 85th observations
auto_data.drop(auto_data.index[9:85], inplace = True)
print('mpg range: {}, mean: {:.2f}, std: {:.2f}'.format(auto_data['mpg'].max
print('cylinders range: {}, mean: {:.2f}, std: {:.2f}'.format(auto_data['cyl
print('displacement range: {}, mean: {:.2f}, std: {:.2f}'.format(auto_data['
print('horsepower range: {}, mean: {:.2f}, std: {:.2f}'.format(auto_data['ho
print('weight range: {}, mean: {:.2f}, std: {:.2f}'.format(auto_data['weight
print('acceleration range: {}, mean: {:.2f}, std: {:.2f}'.format(auto_data['
print('year range: {}, mean: {:.2f}, std: {:.2f}'.format(auto_data['year'].m
```

```
mpg range: 35.6, mean: 24.40, std: 7.87
cylinders range: 5, mean: 5.37, std: 1.65
displacement range: 387.0, mean: 187.24, std: 99.68
horsepower range: 184.0, mean: 100.72, std: 35.71
weight range: 3348, mean: 2935.97, std: 811.30
acceleration range: 16.3, mean: 15.73, std: 2.69
year range: 12, mean: 77.15, std: 3.11
```

(e) Using the full data set, investigate the predictors graphically, using scatterplots or other tools of your choice. Create some plots highlighting the relationships among the predictors. Comment on your findings.

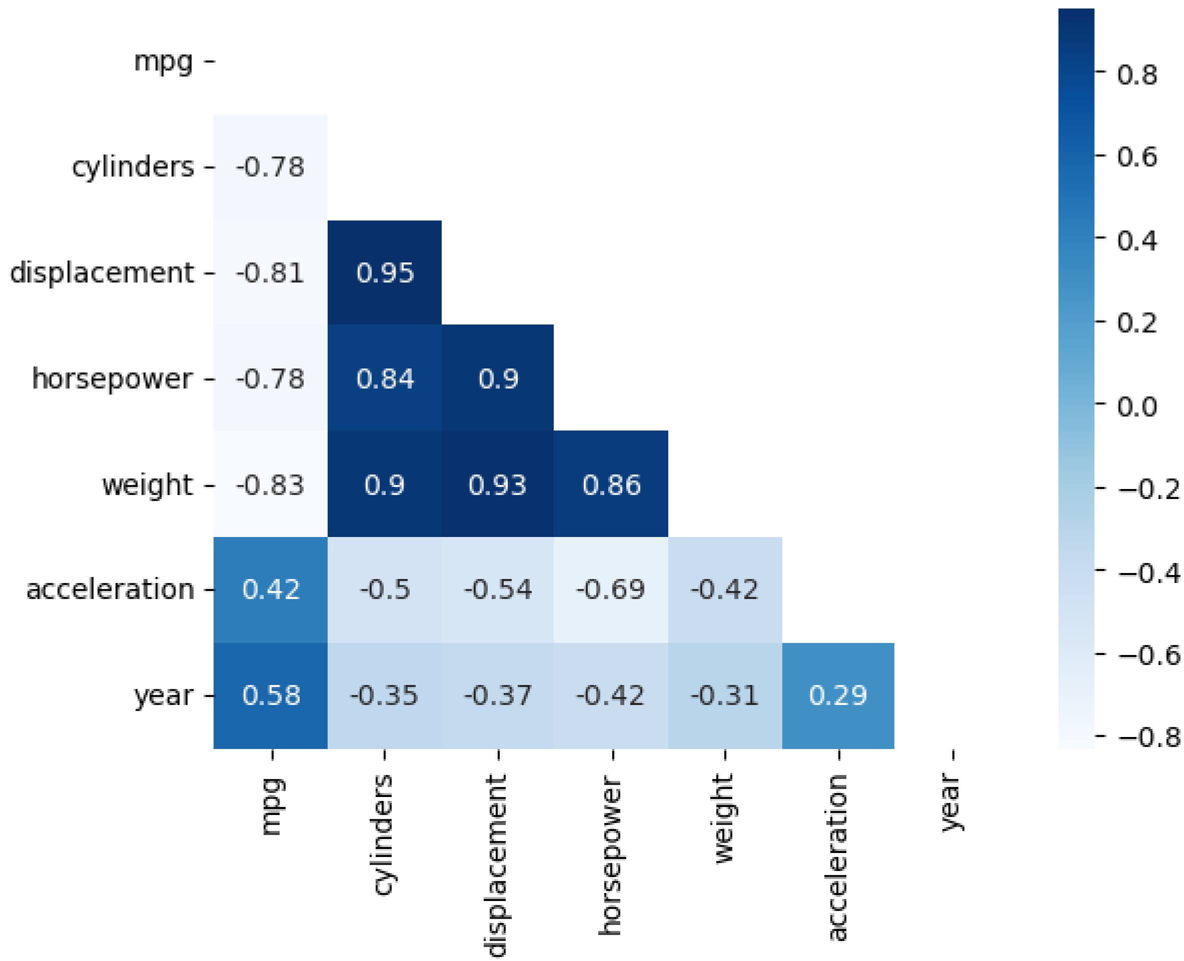
In [ ]:
```
auto_data = pd.read_csv('data/Auto.csv')
auto_data['horsepower'] = pd.to_numeric(auto_data['horsepower'], errors = 'c
auto_data.dropna(inplace = True)
# correlation matrix
```

```
from pandas.plotting import scatter_matrix
scatter_matrix(auto_data[['mpg', 'cylinders', 'displacement', 'horsepower',
```



In [ ]:

```
# heatmap
# %pip install seaborn uncomment to install seaborn
import seaborn as sns
# use absolute value of correlation
corr = auto_data[['mpg', 'cylinders', 'displacement', 'horsepower', 'weight'
# mask upper triangle
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
# plot heatmap
sns.heatmap(corr, mask = mask, annot = True, cmap = 'Blues');
```

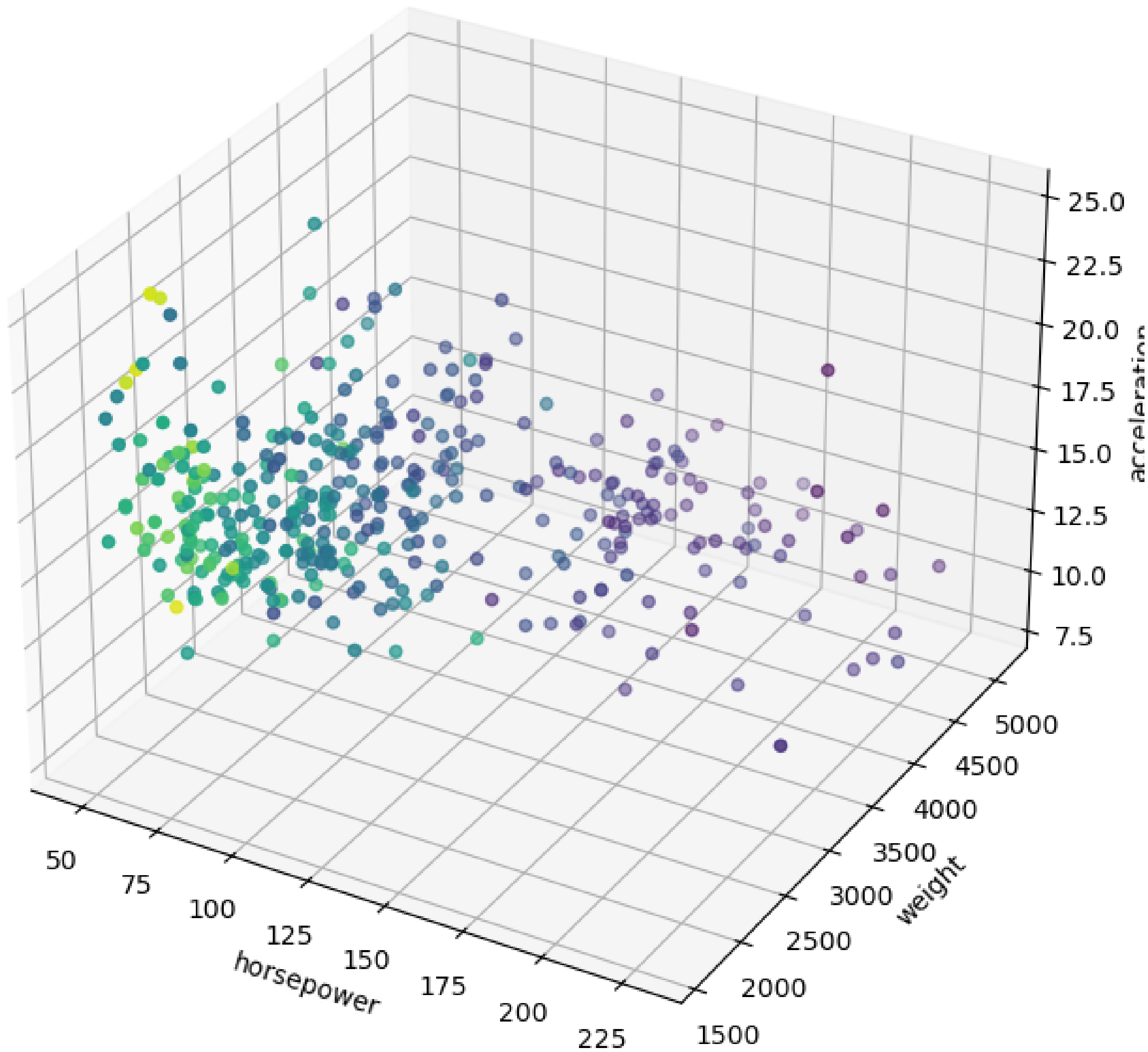


- The above plots show the correlations between the predictors. The strongest correlations are between mpg and displacement, mpg and horsepower, and mpg and weight. This makes sense because the more a car weighs, the more power it needs to move. The more power it needs to move, the more gas it will use.

(f) Suppose that we wish to predict gas mileage (mpg) on the basis of the other variables. Do your plots suggest that any of the other variables might be useful in predicting mpg? Justify your answer.

- The most useful variables for predicting mpg would be weight, displacement, and horsepower. These variables have the strongest correlations with mpg.

In [ ]:
```
#  4d plot of mpg, horsepower, weight, acceleration
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(111, projection = '3d')
ax.scatter(auto_data['horsepower'], auto_data['weight'], auto_data['accelera
ax.set_xlabel('horsepower')
ax.set_ylabel('weight')
ax.set_zlabel('acceleration')
plt.show()
```



In [ ]:

```
# %pip install plotly_express uncomment this to install plotly_express
import plotly_express as px

# 3d scatter of horsepower, weight, acceleration, colored by mpg

fig = px.scatter_3d(auto_data, x = 'horsepower', y = 'weight', z = 'accelera
# change to 800x800 pixels
fig.update_layout(width = 800, height = 800)
```