hw12

August 6, 2021

[1]:
```
# Initialize Otter
import otter
grader = otter.Notebook("hw12.ipynb")
```

1 Homework 12: Principal Component Analysis

In lecture we discussed how PCA can be used for dimensionality reduction. Specifically, given a high dimensional dataset, PCA allows us to: 1. Understand the rank of the data. If k principal components capture almost all of the variance, then the data is roughly rank k. 2. Create 2D scatterplots of the data. Such plots are a rank 2 representation of our data, and allow us to visually identify clusters of similar observations.

A solid geometric understanding of PCA will help you understand why PCA is able to do these two things. In this homework, we'll build that geometric intuition and look at PCA on two datasets: one where PCA works poorly, and the other where it works pretty well.

1.1 Due Date

This assignment is due **Monday, August 9th at 11:59 PM PDT**.

Collaboration Policy

Data science is a collaborative activity. While you may talk with others about the homework, we ask that you **write your solutions individually**. If you do discuss the assignments with others please **include their names** in the cell below.

Collaborators: …

1.2 Score Breakdown

Question	Points
Question 1a	1
Question 1b	1
Question 1c	1
Question 1d	1
Question 1e	1
Question 2a	2
Question 2b	1
Question 2c	1

Question	Points
Question 2d	3
Question 2e	2
Question 3a	1
Question 3b	1
Question 3c	1
Question 3d	2
Question 3e	2
Question 3f	2
Question 3g	1
Question 3h	2
Question 3i	2
Total	28

[2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import plotly.express as px

# Note: If you're having problems with the 3d scatter plots, uncomment the two␣
↪lines below, and you should see a version that
#       number that is at least 4.1.1.
# import plotly
# plotly.__version__
```

1.3 Question 1: PCA on 3D Data

In question 1, our goal is to see visually how PCA is simply the process of rotating the coordinate axes of our data.

The code below reads in a 3D dataset. We have named the DataFrame `surfboard` because the data resembles a surfboard when plotted in 3D space.

[3]:
```
surfboard = pd.read_csv("data3d.csv")
surfboard.head(5)
```

[3]:

	x	y	z
0	0.005605	2.298191	1.746604
1	-1.093255	2.457522	0.170309
2	0.060946	0.473669	-0.003543
3	-1.761945	2.151108	3.132426
4	1.950637	-0.194469	-2.101949

The cell below will allow you to view the data as a 3D scatterplot. Rotate the data around and zoom in and out using your trackpad or the controls at the top right of the figure.

You should see that the data is an ellipsoid that looks roughly like a surfboard or a hashbrown patty. That is, it is pretty long in one direction, pretty wide in another direction, and relatively thin along its third dimension. We can think of these as the "length", "width", and "thickness" of the surfboard data.

Observe that the surfboard is not aligned with the x/y/z axes.

If you get an error that your browser does not support webgl, you may need to restart your kernel and/or browser.

[4]:
```
fig = px.scatter_3d(surfboard, x='x', y='y', z='z', range_x = [-10, 10],␣
↪range_y = [-10, 10], range_z = [-10, 10])
fig.show()
```

To give the figure a little more visual pop, the following cell does the same plot, but also assigns a pre-determined color value (that we've arbitrarily chosen) to each point. These colors do not mean anything important, they're simply there as a visual aid.

You might find it useful to use `colorize_surfboard_data` later in this assignment.

[5]:
```
def colorize_surfboard_data(df):
    colors = pd.read_csv("surfboard_colors.csv", header = None).values
    df_copy = df.copy()
    df_copy.insert(loc = 3, column = "color", value = colors)
    return df_copy

fig = px.scatter_3d(colorize_surfboard_data(surfboard), x='x', y='y', z='z',␣
↪range_x = [-10, 10], range_y = [-10, 10], range_z = [-10, 10], color =␣
↪"color", color_continuous_scale = 'RdBu')
fig.show()
```

1.4 Question 1a

Now that we've understood the data, let's work on understanding what PCA will do when applied to this data.

To properly perform PCA, we will first need to "center" the data so that the mean of each feature is 0.

Compute the columnwise mean of `surfboard` in the cell below, and store the result in `surfboard_mean`. You can choose to make `surfboard_mean` a numpy array or a series, whichever is more convenient for you. Regardless of what data type you use, `surfboard_mean` should have 3 means, 1 for each attribute, with the x coordinate first, then y, then z.

Then, subtract `surfboard_mean` from `surfboard`, and save the result in `surfboard_centered`. The order of the columns in `surfboard_centered` should be x, then y, then z.

[6]:
```
surfboard_mean = np.mean(surfboard, axis = 0)
surfboard_centered = surfboard - surfboard_mean
```

[7]:
```
grader.check("q1a")
```

[7]:
```
q1a results: All test cases passed!
```

1.5 Question 1b

As you may recall from lecture, PCA is a specific application of the singular value decomposition (SVD) for matrices. If we have a data matrix X, we can decompose it into U, Σ and V^T such that $X = U\Sigma V^T$.

In the following cell, use the `np.linalg.svd` function to compute the SVD of `surfboard_centered`. Store the U, Σ, and V^T matrices in `u`, `s`, and `vt` respectively. This is one line of simple code, exactly like what we saw in lecture.

Hint: Set the `full_matrices` argument of `np.linalg.svd` to `False`.

[8]:
```
u, s, vt = np.linalg.svd(surfboard_centered, full_matrices = False)
u, s, vt
```

[8]:
```
(array([[-0.02551985, -0.02108339, -0.03408865],
        [-0.02103979, -0.0259219 ,  0.05432967],
        [-0.00283413, -0.00809889,  0.00204459],
        …,
        [ 0.01536972, -0.00483066,  0.05673824],
        [-0.00917593,  0.0345672 ,  0.03491181],
        [-0.01701236,  0.02743128, -0.01966704]]),
 array([103.76854043,  40.38357469,  21.04757518]),
 array([[ 0.38544534, -0.67267377, -0.63161847],
        [-0.5457216 , -0.7181477 ,  0.43180066],
        [-0.74405633,  0.17825229, -0.64389929]]))
```

[9]:
```
grader.check("q1b")
```

[9]:
```
q1b results: All test cases passed!
```

1.6 Question 1c: Total Variance

Let's now consider the relationship between the singular values `s` and the variance of our data. Recall that the total variance is the sum of the variances of each column of our data. Below, we provide code that computes the variance for each column of the data.

Note: The variances are the same for both `surfboard_centered` and `surfboard`, so we show only one to avoid redundancy.

[10]:
```
np.var(surfboard, axis=0)
```

[10]:
```
x    2.330704
y    5.727527
z    4.783513
dtype: float64
```

The total variance of our dataset is given by the sum of these numbers.

[11]:
```
total_variance_computed_from_data = sum(np.var(surfboard, axis=0))
total_variance_computed_from_data
```

[11]:
```
12.841743509780109
```

As discussed in lecture, the total variance of the data is also equal to the sum of the squares of the singular values divided by the number of data points, that is:

$$Var(X) = \frac{\sum_{i=1}^{d} \sigma_i^2}{N}$$

where σ_i is the singular value corresponding to the i^{th} principal component, N is the total number of data points, and $Var(X)$ is the total variance of the data.

In the cell below, compute the total variance using the the formula above and store the result in the variable `total_variance_computed_from_singular_values`. Your result should be very close to `total_variance_computed_from_data`.

[12]:
```
total_variance_computed_from_singular_values = np.sum(s**2)/surfboard.shape[0]
total_variance_computed_from_singular_values
```

[12]:
```
12.841743509780104
```

[13]:
```
grader.check("q1c")
```

[13]:
```
q1c results: All test cases passed!
```

1.7 Question 1d: Explained Variance and Scree Plots

In the cell below, set `variance_explained_by_1st_pc` to the proportion of the total variance explained by the 1st principal component. Your answer should be a number between 0 and 1.

Note: This topic was discussed in this section of the PCA lecture slides.

[14]:
```
variance_explained_by_1st_pc = (s[0]**2/surfboard.shape[0])/␣
 ↪total_variance_computed_from_data
variance_explained_by_1st_pc
```

[14]:
```
0.8385084140449129
```

[15]:
```
grader.check("q1d")
```

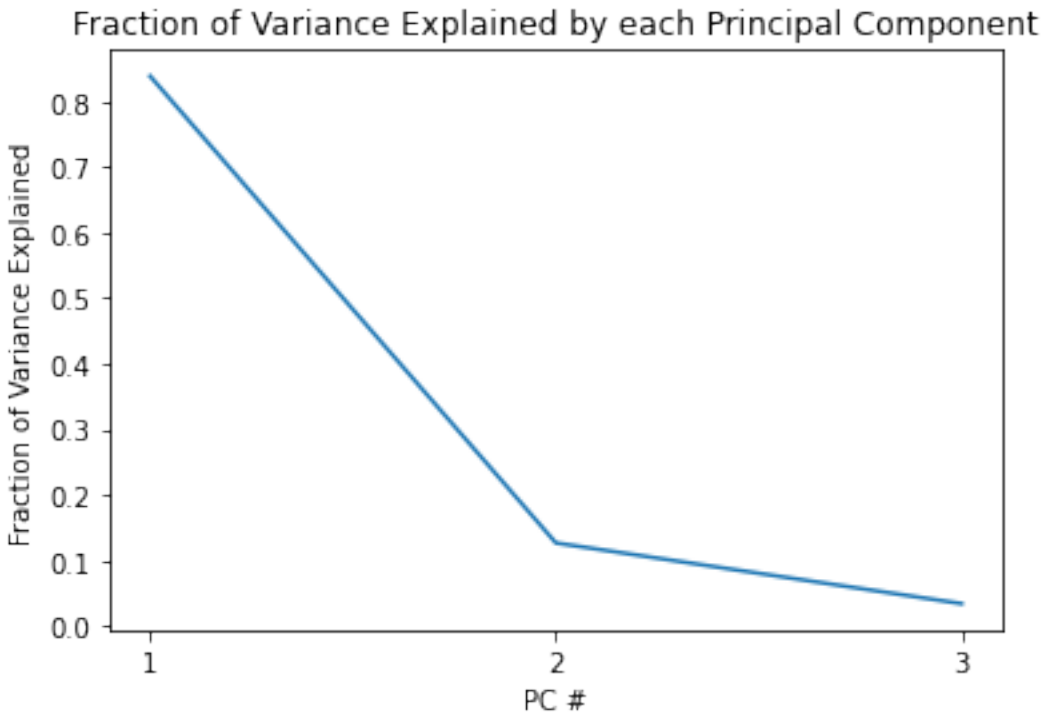
[15]:
```
q1d results: All test cases passed!
```

We can also create a scree plot that shows the proportion of variance explained by all of our principal components, ordered from most to least. An example scree plot is given below. Note that the variance explained by the first principal component matches the value we calculated above for `variance_explained_by_1st_pc`.

Note: If you're wondering where `len(surfboard_centered)` went, it got canceled out when we divided the variance of a given PC by the total variance.

[16]:
```
plt.plot([1, 2, 3], s**2 / sum(s**2));
plt.xticks([1, 2, 3], [1, 2, 3]);
plt.xlabel('PC #');
plt.ylabel('Fraction of Variance Explained');
plt.title('Fraction of Variance Explained by each Principal Component')
```

[16]:
```
Text(0.5, 1.0, 'Fraction of Variance Explained by each Principal Component')
```



For this small toy problem, the scree plot is not particularly useful. We'll see why they are useful in practice later in this homework.

1.8 Question 1e: V as a Rotation Matrix

In lecture, we saw that the first column of XV contained the first principal component values for each observation, the second column of XV contained the second principal component values for

each observation, and so forth.

Let's give this matrix a name: $P = XV$ is sometimes known as the "principal component matrix".

Compute the P matrix for the surfboard dataset and store it in the variable `surfboard_pcs`.

[17]:
```
surfboard_pcs = u @ np.diag(s)
```

[18]:
```
grader.check("q1e")
```

[18]:
```
q1e results:
    Trying:
        all(np.isclose(surfboard_pcs.loc[0], [-2.648, -0.851, -0.717],
atol=1e-3))
    Expecting:
        True
    **********************************************************************
    Line 1, in q1e 2
    Failed example:
        all(np.isclose(surfboard_pcs.loc[0], [-2.648, -0.851, -0.717],
atol=1e-3))
    Exception raised:
        Traceback (most recent call last):
          File "/opt/conda/lib/python3.8/doctest.py", line 1336, in __run
            exec(compile(example.source, filename, "single",
          File "<doctest q1e 2[0]>", line 1, in <module>
            all(np.isclose(surfboard_pcs.loc[0], [-2.648, -0.851, -0.717],
atol=1e-3))
        AttributeError: 'numpy.ndarray' object has no attribute 'loc'
```

1.9 Visualizing the Principal Component Matrix

In some sense, we can think of P as an output of the PCA procedure.

It is simply a rotation of the data such that the data will now appear "axis aligned". Specifically, for a 3d dataset, if we plot PC1, PC2, and PC3 along the x, y, and z axes of our plot, then the greatest amount of variation happens along the x-axis, the second greatest amount along the y-axis, and the smallest amount along the z-axis.

To visualize this, run the cell below, which will show our data now projected onto the principal component space. Compare with your original figure, and observe that the data is exactly the same, only it is now rotated.

[19]:
```
surfboard_pcs = surfboard_pcs.rename(columns = {0: "pc1", 1: "pc2", 2: "pc3"})
fig = px.scatter_3d(colorize_surfboard_data(surfboard_pcs),
                    x='pc1', y='pc2', z='pc3', range_x = [-10, 10], range_y =␣
↪[-10, 10], range_z = [-10, 10], color = 'color', color_continuous_scale =␣
↪'RdBu');
fig.show();
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/tmp/ipykernel_22/437898750.py in <module>
----> 1 surfboard_pcs = surfboard_pcs.rename(columns = {0: "pc1", 1: "pc2", 2:␣
 ↪"pc3"})
      2 fig = px.scatter_3d(colorize_surfboard_data(surfboard_pcs),
      3                     x='pc1', y='pc2', z='pc3', range_x = [-10, 10],␣
 ↪range_y = [-10, 10], range_z = [-10, 10], color = 'color',␣
 ↪color_continuous_scale = 'RdBu');
      4 fig.show();

AttributeError: 'numpy.ndarray' object has no attribute 'rename'
```

We can also create a 2D scatterplot of our surfboard data as well. Note that the resulting is just the 3D plot as viewed from directly "overhead".

[20]:
```
ax = sns.scatterplot(data = colorize_surfboard_data(surfboard_pcs), x = 'pc1',␣
 ↪y = 'pc2', hue = "color", palette = "RdBu", legend = False)
ax.set_xlim(-10, 10);
ax.set_ylim(-10, 10);
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/tmp/ipykernel_22/745272516.py in <module>
----> 1 ax = sns.scatterplot(data = colorize_surfboard_data(surfboard_pcs), x =
 ↪'pc1', y = 'pc2', hue = "color", palette = "RdBu", legend = False)
      2 ax.set_xlim(-10, 10);
      3 ax.set_ylim(-10, 10);

/tmp/ipykernel_22/1396351968.py in colorize_surfboard_data(df)
      2     colors = pd.read_csv("surfboard_colors.csv", header = None).values
      3     df_copy = df.copy()
----> 4     df_copy.insert(loc = 3, column = "color", value = colors)
      5     return df_copy
      6

AttributeError: 'numpy.ndarray' object has no attribute 'insert'
```

1.10 Question 1 Summary

Above, we saw that the principal component matrix P is simply the original data rotated in space so that it appears axis-aligned.

Whenever we do a 2D scatter plot of only the first 2 columns of P, we are simply looking at the data from "above", i.e. so that the 3rd (or higher) PC is invisible to us.

1.11 Question 2

Let's try out PCA on a higher dimensional dataset.

In this question, we'll look at a sheet containing Midterm 1 grades from Data 100 in Fall 2019. As it turns out, PCA scatterplots won't be particularly useful on this dataset, but we'll learn some important things along the way.

Specifically, we'll have 3 primary goals in question 2: we will see what a 2D scatter plot of high dimensional data looks like, we will try to interpret the meaning of our principal components, and we will see why standardizing columns can be important.

Note: These grades are a snapshot of the grades before the TAs were done grading. Any problem which hadn't been graded at the time was given a score of 0.

[21]:
```
mid1_grades = pd.read_csv("fa19mid1.csv")
mid1_grades.head()
```

[21]:

	1.1: 1a (1.0 pts)	1.2: 1b (3.0 pts)	1.3: 1c i (1.0 pts)	\
0	0	3.0	0.5	
1	0	3.0	1.0	
2	1	3.0	1.0	
3	0	2.0	1.0	
4	1	3.0	1.0	

	1.4: 1c ii (1.0 pts)	1.5: 1c iii (1.0 pts)	1.6: 1d (6.0 pts)	\
0	0	0.5	6	
1	0	0.0	6	
2	1	1.0	6	
3	0	0.5	6	
4	1	0.5	6	

	1.7: 1e (8.0 pts)	1.8: 1f (2.0 pts)	1.9: 1g (2.0 pts)	2.1: 2a (2.0 pts)	\
0	3	2	2	2	
1	2	2	0	2	
2	7	2	2	2	
3	8	2	2	2	
4	6	1	0	2	

	…	5.2: 5a ii (3.0 pts)	5.3: 5b i (3.0 pts)	5.4: 5b ii (3.0 pts)	\
0	…	1	0.0	0	
1	…	0	3.0	3	
2	…	1	2.0	3	
3	…	1	3.0	0	
4	…	3	2.0	3	

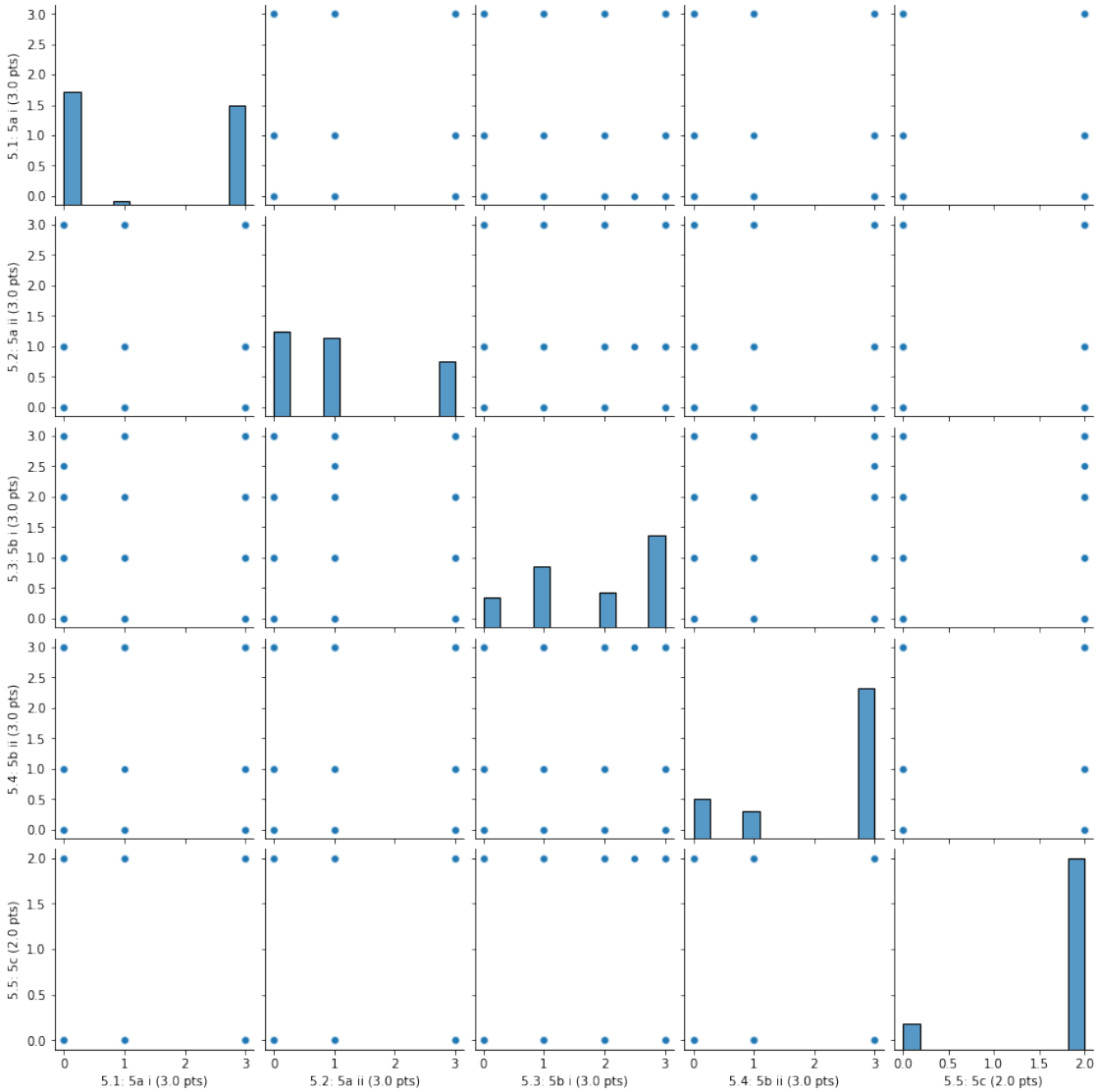
	5.5: 5c (2.0 pts)	6.1: 6a i (2.0 pts)	6.2: 6a ii (2.0 pts)	\
0	2	1	2	
1	0	2	2	

```
2                   2                   2                   2
3                   2                   1                   2
4                   0                   2                   2

   6.3: 6b i (3.0 pts)  6.4: 6b ii (4.0 pts)  6.5: 6b iii (3.0 pts)  \
0                    3                    1                     2
1                    3                    0                     2
2                    3                    4                     3
3                    3                    1                     0
4                    3                    1                     2

   6.6: 6c (4.0 pts)
0                  0
1                  0
2                  0
3                  0
4                  4

[5 rows x 31 columns]
```

This is 31 dimensional data, with each row of the table representing the scores of a given student. This data has far too many dimensions for us to be able to plot all of the data at once.

One approach is to create a bunch of 2D scatterplots, one for each pair of variables. For example, the cell below generates scatterplots for each of the 5 problems about visualizations.

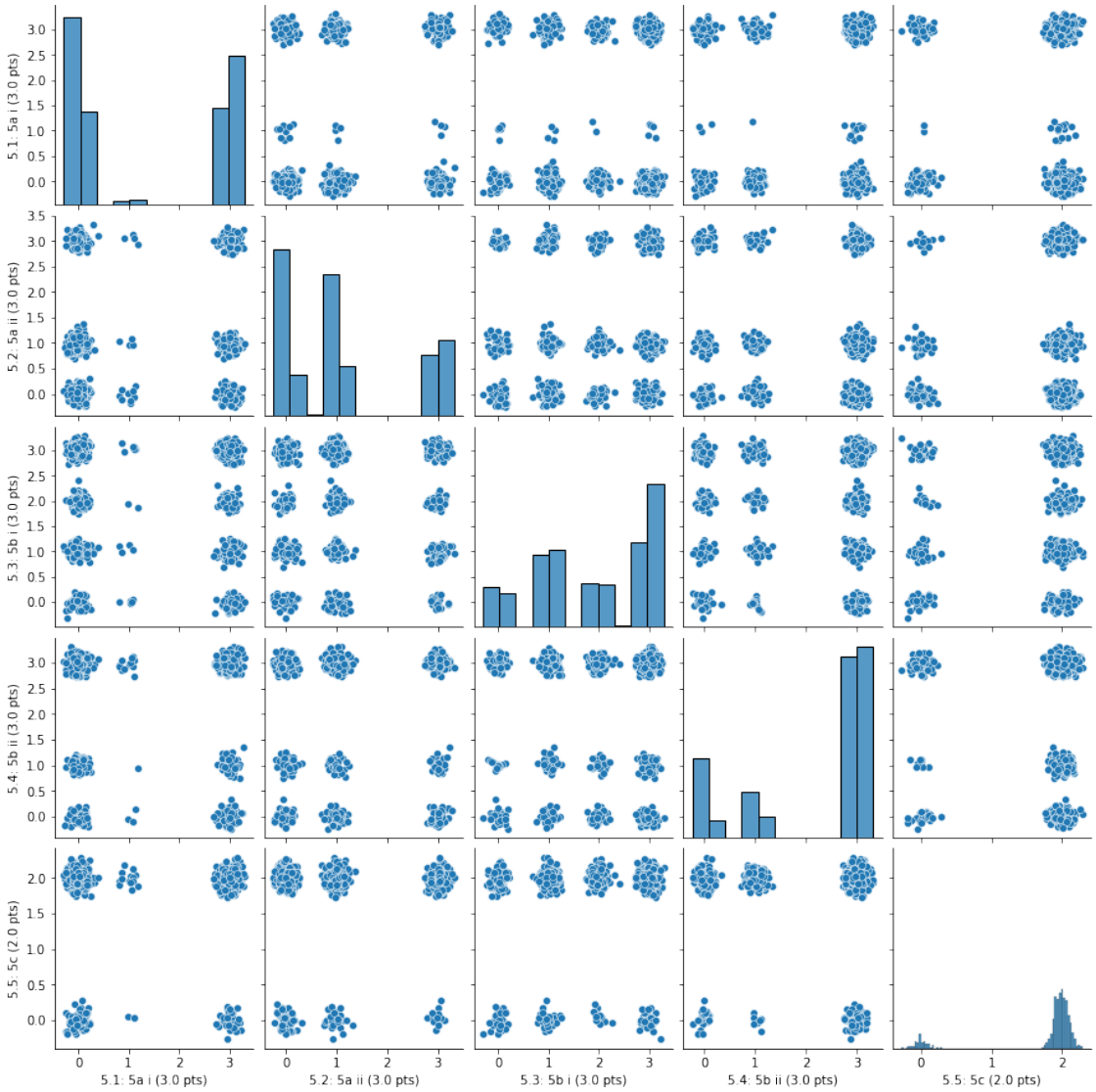
[22]:
```
sns.pairplot(mid1_grades.loc[:, '5.1: 5a i (3.0 pts)':'5.5: 5c (2.0 pts)']);
```



Unfortunately, due to overplotting, the plot above isn't very informative.

To address overplotting, we can add a little bit of noise. The resulting visualization still isn't great, but we can at least learn a few things, e.g. among students who got 1 point on problem 5b.ii, only one of those students also got 1 point on problem 5a.i.

[23]:
```
sns.pairplot(mid1_grades.loc[:, '5.1: 5a i (3.0 pts)':'5.5: 5c (2.0 pts)'] + np.
↪random.normal(0, 0.1, size = (len(mid1_grades), 5)));
```



1.12 Question 2a

Using PCA, we can try to visualize student performance on ALL questions simultaneously. In the cell below, create a **DataFrame** called `mid1_1st_2_pcs` that has 992 rows and 2 columns, where the first column is named `pc1` and represents the first principal component, and the second column is named `pc2` and represents the second principal component. The columns of your dataframe should be named `pc1` and `pc2`.

Reminder: make sure to center your data first!

[24]:
```
mid_mean = np.mean(mid1_grades, axis = 0)
mid1_grades_centered = mid1_grades - mid_mean
u_2a, s_2a, vt_2a = np.linalg.svd(mid1_grades_centered, full_matrices = False)
mid1_1st_2_pcs = pd.DataFrame(data= (u_2a*s_2a)[:, 0:2], columns=['pc1', 'pc2'])
```

[25]:
```
grader.check("q2a")
```

[25]:
```
q2a results: All test cases passed!
```

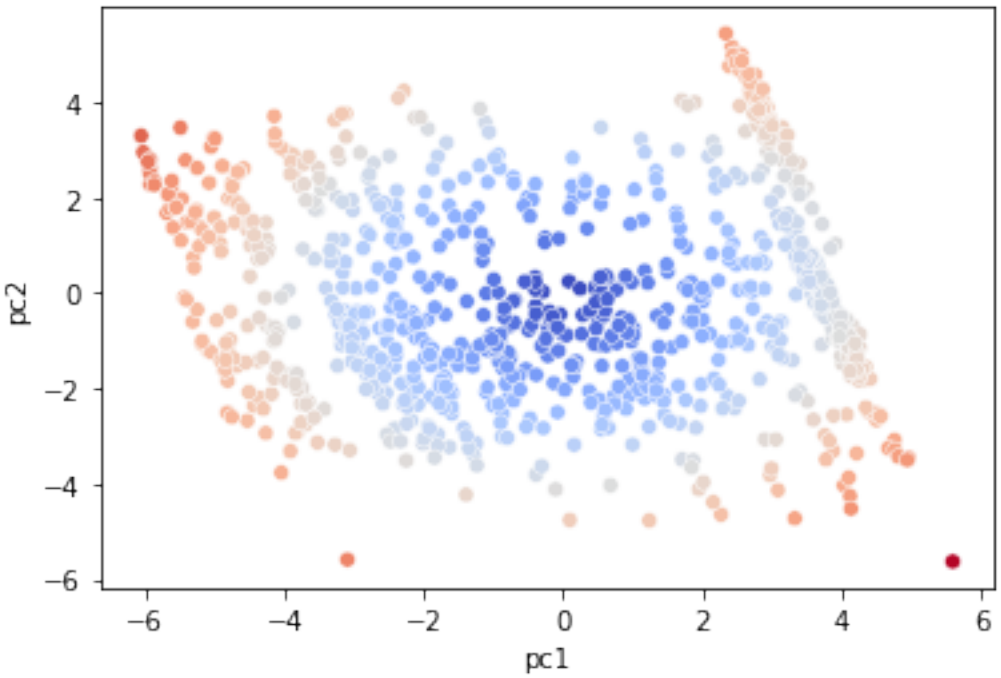
1.13 Question 2b

In the cell below, we create a 2d scatterplot of the first two principal components of the data. Observe that the plot appears to have some sort of structure: The data shows diagonal bands. We will not explore the reasons for these diagonal bands, but we leave this as an optional exercise (Q4) at the very end of the homework if you're curious.

As with the surfboard data, we have assigned arbitrary colors to each student as a visual aid. There is no special meaning to the colors.

[26]:
```
def colorize_midterm_data(df):
    """Adds a color column to the given midterm data."""
    colors = pd.read_csv("mid1_colors.csv", header = None).values
    df_copy = df.copy()
    df_copy.insert(loc = 0, column = "color", value = colors)
    return df_copy
```

[27]:
```
sns.scatterplot(data = colorize_midterm_data(mid1_1st_2_pcs), x = "pc1", y =␣
↪"pc2", hue = "color", palette = "coolwarm", legend = False);
```



This is a 2D plot of 31 dimensional data. To get a sense of how much of the story we're capturing, in the cell below, give the proportion of the variance accounted for by the first two principal

components. Give your answer exactly. It should be around 39%.

[28]:
```
ds100_mt1_1st_2_pc_variance_fraction = ((np.sum(s[0:2]**2))/(992))/(np.
↪sum(s**2)/(992))
ds100_mt1_1st_2_pc_variance_fraction
```

[28]:
```
0.9655030938220892
```

[29]:
```
grader.check("q2b")
```

[29]:
```
q2b results: All test cases passed!
```

1.14 Question 2c

While the plot above captures around 39% of the variance, it's hard to interpret. One approach is to do something similar to what we did during lecture where we investigated how much each column of our data contributes to each principal component.

The function defined in the cell below plots and labels the rows of V^T.

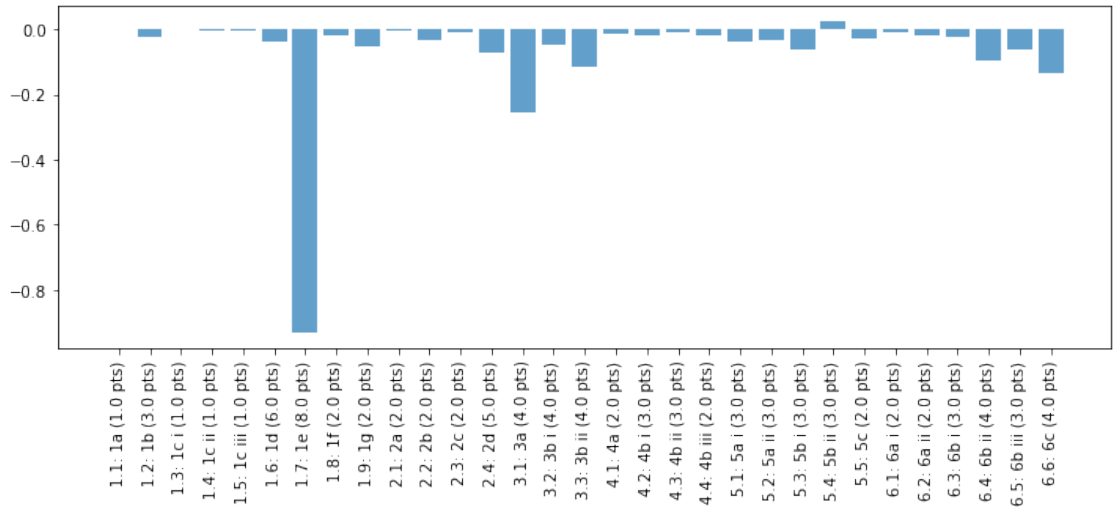
[30]:
```
def plot_pc(col_names, vt, k):
    plt.bar(col_names, vt[k, :], alpha=0.7)
    plt.xticks(col_names, rotation=90);
```

The cell below plots the first row of V^T.

This gives us a chance to try to interpret what PC1 actually means. We see that the first principal component is largely dominated by how a student did on problem 1.7.

Naively, we'd assume that means then that problem 1.7 is the best indicator of student success on the exam. That is, we might expect that 1.7 is a really good problem that deeply tests Data 100 insights.

[31]:
```
mid1_col_names = mid1_grades.columns

plt.figure(figsize=(12, 4))
plot_pc(mid1_col_names, vt_2a, 0);
```



However, if we look at the variance of each individual problem, we see that `problem 1.7: 1e` has a much higher variance than the other problems. After all, PCA is all about identifying the combination of features that best captures variance across observations.

And in case you're curious, this large variance is partially because this problem is worth a lot of points, but also because this snapshot of the grades was at a time when we had only graded a small number of the 1.7 submissions so lots of people still have a zero.

[32]:
```
np.var(mid1_grades)
```

[32]:
```
1.1: 1a (1.0 pts)          0.246582
1.2: 1b (3.0 pts)          0.321965
1.3: 1c i (1.0 pts)        0.026728
1.4: 1c ii (1.0 pts)       0.080010
1.5: 1c iii (1.0 pts)      0.089778
1.6: 1d (6.0 pts)          1.528688
1.7: 1e (8.0 pts)          8.491249
1.8: 1f (2.0 pts)          0.243952
1.9: 1g (2.0 pts)          0.966340
2.1: 2a (2.0 pts)          0.035961
2.2: 2b (2.0 pts)          0.452519
2.3: 2c (2.0 pts)          0.251915
2.4: 2d (5.0 pts)          1.582213
3.1: 3a (4.0 pts)          2.351464
3.2: 3b i (4.0 pts)        0.710237
3.3: 3b ii (4.0 pts)       1.498929
4.1: 4a (2.0 pts)          0.471467
4.2: 4b i (3.0 pts)        0.735016
4.3: 4b ii (3.0 pts)       0.374919
4.4: 4b iii (2.0 pts)      0.313374
```

```
5.1: 5a i (3.0 pts)      2.204308
5.2: 5a ii (3.0 pts)     1.388593
5.3: 5b i (3.0 pts)      1.228950
5.4: 5b ii (3.0 pts)     1.497788
5.5: 5c (2.0 pts)        0.422277
6.1: 6a i (2.0 pts)      0.364395
6.2: 6a ii (2.0 pts)     0.230278
6.3: 6b i (3.0 pts)      0.389746
6.4: 6b ii (4.0 pts)     1.732163
6.5: 6b iii (3.0 pts)    1.173012
6.6: 6c (4.0 pts)        3.513381
dtype: float64
```

One way to prevent having high variance variables from dominating the first principal component is to rescale our data so that each column has unit variance.

Create a data frame `mid1_grades_centered_scaled` such that the means of each column in `mid1_grades_centered` are 0 and their variances are 1.

Hint: Consider how dividing a column by c affects the sample variance of that column. Each column will need to be divided by a different number in order to achieve unit variance in that column.

[33]:
```
midtsds = np.std(mid1_grades, axis = 0)
mid1_grades_centered_scaled = mid1_grades_centered/midtsds
mid1_grades_centered_scaled
```

[33]:

	1.1: 1a (1.0 pts)	1.2: 1b (3.0 pts)	1.3: 1c i (1.0 pts)	\
0	-1.124651	0.421937	-2.802474	
1	-1.124651	0.421937	0.255891	
2	0.889165	0.421937	0.255891	
3	-1.124651	-1.340428	0.255891	
4	0.889165	0.421937	0.255891	
..	…	…	…	
987	-1.124651	0.421937	0.255891	
988	0.889165	-0.459245	0.255891	
989	0.889165	0.421937	0.255891	
990	0.889165	0.421937	0.255891	
991	-1.124651	0.421937	0.255891	

	1.4: 1c ii (1.0 pts)	1.5: 1c iii (1.0 pts)	1.6: 1d (6.0 pts)	\
0	-3.225260	-0.203544	0.569910	
1	-3.225260	-1.872265	0.569910	
2	0.310053	1.465178	0.569910	
3	-3.225260	-0.203544	0.569910	
4	0.310053	-0.203544	0.569910	
..	…	…	…	
987	0.310053	1.465178	-1.856487	

```
988              0.310053              -0.203544          -1.047688
989              0.310053              -0.203544          -0.238889
990              0.310053               1.465178          -1.047688
991              0.310053              -0.203544          -2.665286

     1.7: 1e (8.0 pts)  1.8: 1f (2.0 pts)  1.9: 1g (2.0 pts)  \
0            -0.162592           0.506160           0.830632
1            -0.505766           0.506160          -1.203903
2             1.210103           0.506160           0.830632
3             1.553277           0.506160           0.830632
4             0.866929          -1.518481          -1.203903
..                 …                  …                  …
987           1.553277           0.506160           0.830632
988          -0.848940           0.506160           0.830632
989          -0.505766          -1.518481           0.830632
990           0.180581           0.506160           0.830632
991           0.180581           0.506160           0.830632

     2.1: 2a (2.0 pts)  …  5.2: 5a ii (3.0 pts)  5.3: 5b i (3.0 pts)  \
0             0.095685  …             -0.102655           -1.689079
1             0.095685  …             -0.951274            1.017085
2             0.095685  …             -0.102655            0.115030
3             0.095685  …             -0.102655            1.017085
4             0.095685  …              1.594582            0.115030
..                 …  …                   …                  …
987           0.095685  …             -0.951274           -0.787024
988           0.095685  …              1.594582           -1.689079
989           0.095685  …             -0.102655           -0.787024
990           0.095685  …             -0.102655            0.115030
991           0.095685  …             -0.951274           -0.787024

     5.4: 5b ii (3.0 pts)  5.5: 5c (2.0 pts)  6.1: 6a i (2.0 pts)  \
0              -1.804702           0.369204          -0.911790
1               0.646596          -2.708530           0.744795
2               0.646596           0.369204           0.744795
3              -1.804702           0.369204          -0.911790
4               0.646596          -2.708530           0.744795
..                   …                  …                  …
987            -1.804702           0.369204           0.744795
988             0.646596           0.369204          -0.911790
989             0.646596           0.369204           0.744795
990             0.646596           0.369204          -0.911790
991             0.646596           0.369204           0.744795

     6.2: 6a ii (2.0 pts)  6.3: 6b i (3.0 pts)  6.4: 6b ii (4.0 pts)  \
0               0.277291            0.217987           -0.411309
1               0.277291            0.217987           -1.171120
```

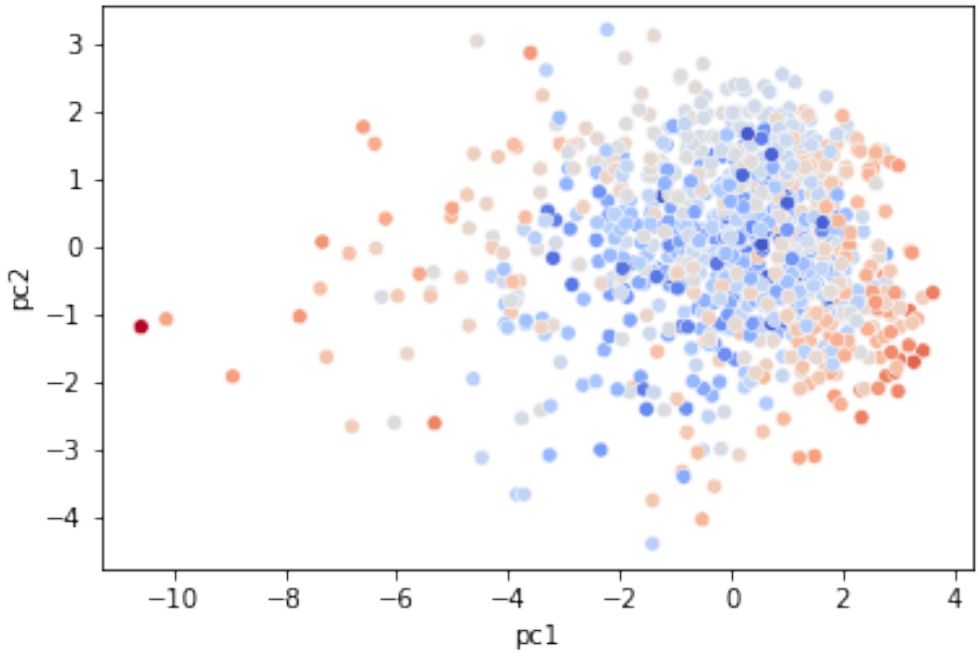
```
2                0.277291             0.217987              1.868124
3                0.277291             0.217987             -0.411309
4                0.277291             0.217987             -0.411309
..                    …                    …                     …
987              0.277291             0.217987             -0.411309
988              0.277291             0.217987             -0.411309
989              0.277291             0.217987             -0.411309
990              0.277291             0.217987             -0.411309
991              0.277291            -4.587422             -1.171120

     6.5: 6b iii (3.0 pts)  6.6: 6c (4.0 pts)
0                 0.067945           -0.694845
1                 0.067945           -0.694845
2                 0.991258           -0.694845
3                -1.778680           -0.694845
4                 0.067945            1.439169
..                     …                  …
987               0.991258           -0.694845
988               0.067945           -0.694845
989               0.991258           -0.694845
990               0.067945            1.439169
991              -1.778680           -0.694845

[992 rows x 31 columns]
```

[34]:
```
grader.check("q2c")
```

[34]:
```
q2c results: All test cases passed!
```

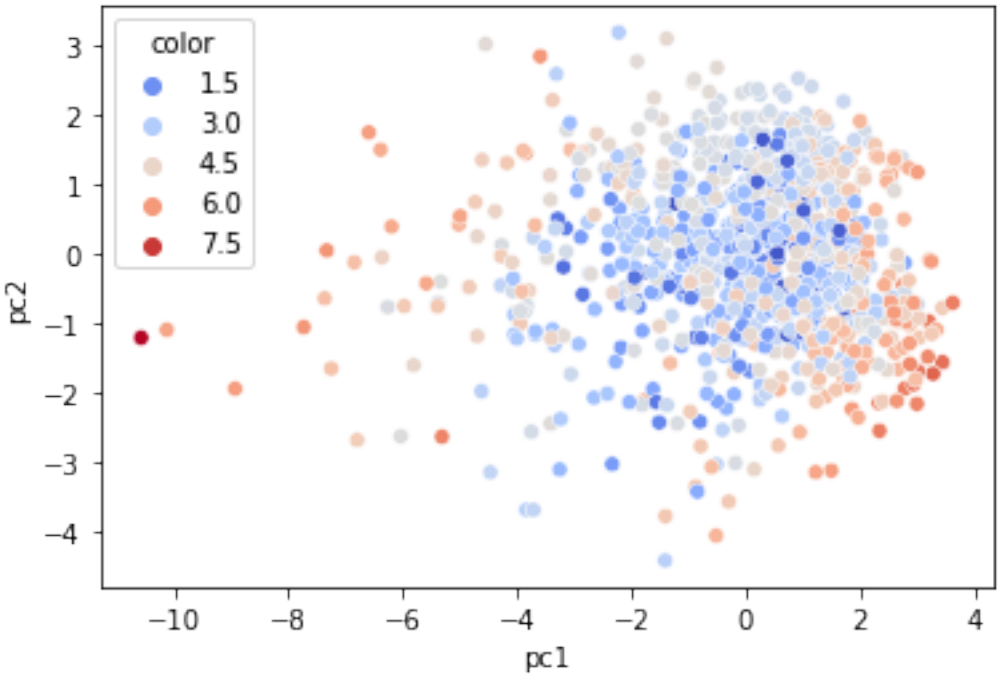
1.15 Question 2d

Create a 2D scatterplot of the first two principal components of `mid1_grades_centered_scaled`. Use `colorize_midterm_data` to add a `color` column to `mid1_1st_2_pcs`. Your code will be very similar to the code from problems 2a and 2b. Your result should look like this



[35]:
```
u, s, vt = np.linalg.svd(mid1_grades_centered_scaled, full_matrices = False)
mid1_1st_2_pcs = pd.DataFrame(data= (u*s)[:, 0:2], columns=['pc1', 'pc2'])
mid1_1st_2_pcs
sns.scatterplot(data = colorize_midterm_data(mid1_1st_2_pcs), x = "pc1", y =␣
↪"pc2", hue = "color", palette = "coolwarm")
```

[35]:
```
<AxesSubplot:xlabel='pc1', ylabel='pc2'>
```



This scatterplot is quite different. The diagonal banding we saw before is gone.

By looking at the colors, we can also get some sense of how our centering process altered the locations of each student in the 2D PCA scatterplot. Observe that the blue students are still mostly close to each other, and the red points are still largely at the margins.

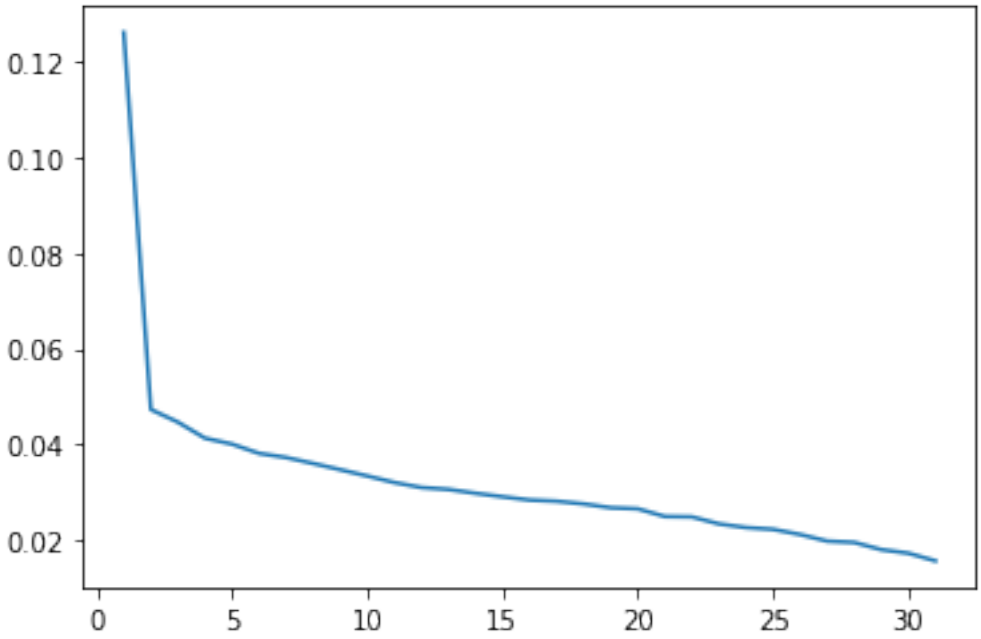
This plot shows relatively little structure and has no clusters. There does not appear to be much to learn here.

1.16 Question 2e

If you compute the fraction of the variance captured by this 2D scatter plot, you'll see it's only 17%, roughly 12% by the 1st PC, and roughly 5% by the 2nd PC. **In the cell below, create a scree plot showing the fraction of the variance explained by each principal componant using the data from 2d.**

Informally, we can say that our midterm scores matrix has a high rank. More formally, we can say that 2 principal components only capture a small fraction of the variance, and thus the data are not particularly amenable to 2D PCA scatterplotting.

[36]:
```
sns.lineplot(x = np.arange(1,32), y= ((s**2)/992)/(np.sum(s**2)/(992)))
plt.show()
```



1.17 Question 3

PCA really shines on data where you have reason to believe that the data is relatively low in rank.

In this final question of the homework, we'll look at how states voted in presidential elections between 1972 and 2016. **Our ultimate goal in question 3 is to show how 2D PCA scatterplots can allow us to identify clusters in a high dimensional dataset.** For this example, that means finding groups of states that vote similarly by plotting their 1st and 2nd principal components.

[37]:
```
df = pd.read_csv("presidential_elections.csv")
df.head(5)
```

[37]:

	State	1789	1792	1796	1800 †	Unnamed: 5	1804	1808	1812	1816	…	1992	\
0	Alabama	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	…	R	
1	Alaska	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	…	R	
2	Arizona	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	…	R	
3	Arkansas	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	…	D	
4	California	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	…	D	

	1996	2000 ‡	Unnamed: 60	2004	2008	2012	2016 ‡	2020	State.1
0	R	R	NaN	R	R	R	R	R	Alabama
1	R	R	NaN	R	R	R	R	R	Alaska
2	D	R	NaN	R	R	R	R	D	Arizona
3	D	R	NaN	R	R	R	R	R	Arkansas
4	D	D	NaN	D	D	D	D	D	California

```
[5 rows x 67 columns]
```

The data in this table is pretty messy, so let's create a clean version. The clean table should contain exactly 51 rows (corresponding to the 50 states plus Washington DC) and 13 columns (one for each of the election years from 1972 to 2020). The index of this dataframe should be the state name.

Note: In your personal projects, it is sometimes more convenient to manually do your data cleaning using Excel or Google Sheets. The downside of doing this is that you have no record of what you did, and if you have to redownload the data, you have to redo the manual data cleaning process.

[38]:
```
df_clean = (
        df.iloc[:, -15:]
        .drop(['Unnamed: 60'], axis = 1)
        .rename(columns = {"2000 ‡": "2000", "2016 ‡": "2016", "State.1":␣
↪"State"})
        .drop([51])
        .set_index("State")
)
df_clean.head(5)
```

[38]:

State	1972	1976	1980	1984	1988	1992	1996	2000	2004	2008	2012	2016	2020
Alabama	R	D	R	R	R	R	R	R	R	R	R	R	R
Alaska	R	R	R	R	R	R	R	R	R	R	R	R	R
Arizona	R	R	R	R	R	R	D	R	R	R	R	R	D
Arkansas	R	D	R	R	R	D	D	R	R	R	R	R	R
California	R	R	R	R	R	D	D	D	D	D	D	D	D

1.18 Question 3a

What does each row in `df_clean` represent?

Type your answer here, replacing this text.

1.19 Question 3b

To perform PCA, we need to convert our data into numerical values. To do this, replace all of the "D" characters with the number 0, and all of the "R" characters with the number 1.

Hint: Use `df.replace`.

[39]:
```
df_numerical = df_clean.replace({'D': 0, 'R': 1} )
```

[40]:
```
grader.check("q3b")
```

[40]:
```
q3b results: All test cases passed!
```

1.20 Question 3c

Now center the data so that the mean of each column is 0 and scale the data so that the variance of each column is 1. Store your result in `df_standardized`.

[43]:
```
elections_mean = pd.DataFrame.mean(df_numerical, axis = 0)
centerd_elections = df_numerical - elections_mean
elections_std = np.std(df_numerical, axis = 0)
elections_centered_scaled = centerd_elections/elections_std
df_standardized = elections_centered_scaled
```

[44]:
```
grader.check("q3c")
```

[44]:
```
q3c results: All test cases passed!
```

1.21 Question 3d

We now have our data in a nice and tidy centered and scaled format, phew. We are now ready to do PCA.

Create a new dataframe `first_2_pcs` that contains exactly the first two columns of the principal components matrix. The first column should be labeled `pc1` and the second column should be labeled `pc2`. Store your result in `first_2_pcs`.

[49]:
```
u_q3, s_q3, vt_q3 = np.linalg.sv, full_matrices = False)
first_2_pcs = pd.DataFrame(data= (u_q3*s_q3)[:, 0:2], columns=['pc1', 'pc2'])
first_2_pcs.head()
```

[49]:

	pc1	pc2
0	-2.938635	0.796415
1	-3.019941	-0.166841
2	-1.723192	-0.487691
3	-1.698397	0.807836
4	2.376794	-1.807335

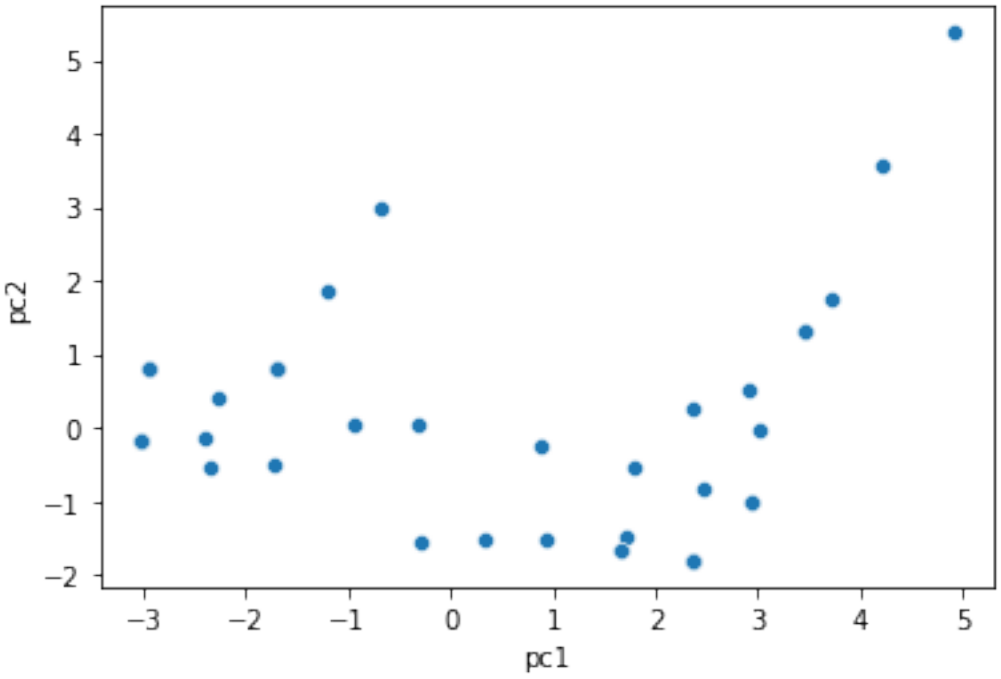
[50]:
```
grader.check("q3d")
```

[50]:
```
q3d results: All test cases passed!
```

1.22 Question 3e

The cell below plots the 1st and 2nd principal components of our 50 states + Washington DC.

[51]:
```
sns.scatterplot(data = first_2_pcs, x = "pc1", y = "pc2");
```



Unfortunately, we have two problems:

1. There is a lot of overplotting, with only 28 distinct dots. This means that at least some states voted exactly alike in these elections.
2. We don't know which state is which because the points are unlabeled.

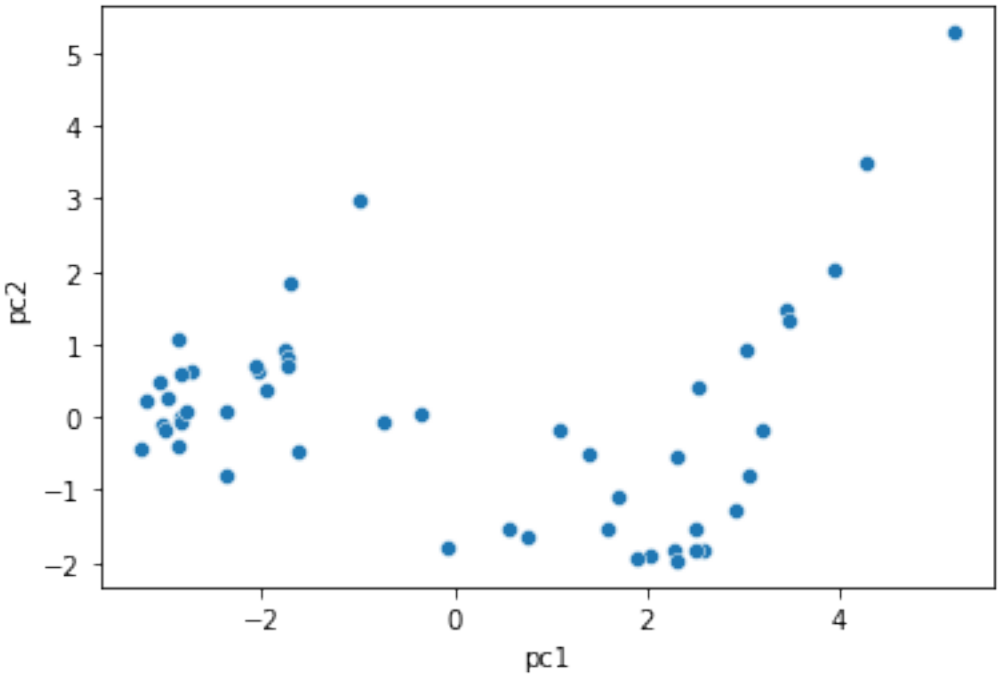
Let's start by addressing problem 1.

In the cell below, create a new dataframe `first_2_pcs_jittered` with a small amount of random noise added to each principal component. In this same cell, create a scatterplot.

The amount of noise you add should not significantly affect the appearance of the plot, it should simply serve to separate overlapping observations. Don't get caught up on the exact details of your noise generation, it's fine as long as your plot looks roughly the same as the original scatterplot.

Hint: See the pairplot from the intro to question 2 for an example of how to introduce noise.

[53]:
```
first_2_pcs_jittered = first_2_pcs+np.random.normal(0, 0.2, size␣
↪=(len(first_2_pcs), 2))
sns.scatterplot(data = first_2_pcs_jittered, x = "pc1", y = "pc2");
```



1.23 Question 3f

To label our points, the best option is to use the `plotly` library that we used earlier in this homework. `plotly` is an incredibly powerful plotting library that will automatically add axis labels, and will also provide controls so that you can zoom and pan around to look at the data.

One important skill as a user of modern tools is using existing documentation and examples to get the plot you want.

Using the example given on this page as a guide, we can create a scatter plot of the **jittered data** from 3e.

[54]:
```
import plotly.express as px

state_names = list(df_standardized.index)
first_2_pcs_jittered['state'] = state_names

fig = px.scatter(first_2_pcs_jittered, x="pc1", y="pc2", text="state");

fig.update_traces(textposition='top center');

fig.show();
```

1.23.1 Part i

Give an example of a cluster of states that vote a similar way. Does the composition of this cluster surprise you? If you're not familiar with U.S. politics, it's fine to just say 'No, I'm not surprised because I don't know anything about U.S. politics.'.

Although I'm not very familiar with U.S.politics, it is not suprising to see a large amount of ht blue and red states are grouped together–for example california and Conneticut, and Lousiana.

1.23.2 Part ii

In the cell below, write down anything interesting that you observe by looking at this plot. You will get credit for this as long as you write something reasonable that you can take away from the plot.

Fristly, some typical swing state vote differently from others such as Florida. It is also interesting to see that D.C is placed in the top right corner of the plot.

1.24 Question 3g

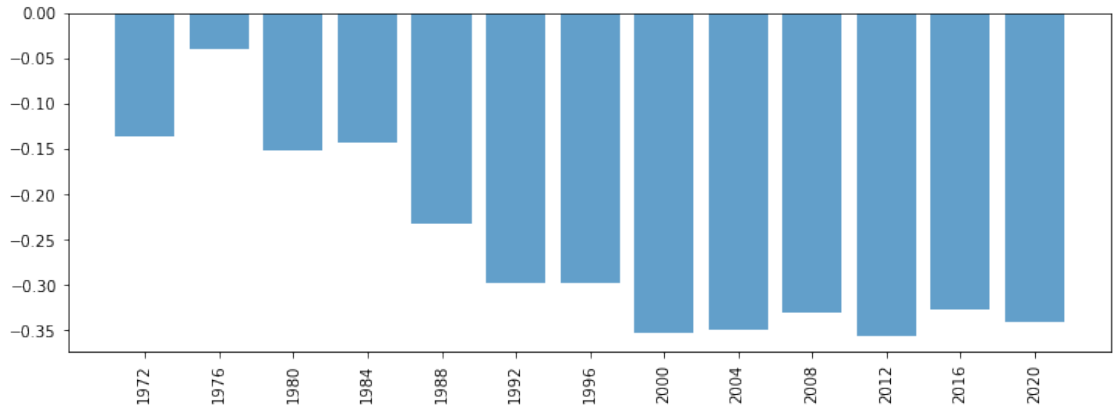
We can also look at the contributions of each year's elections results on the values for our principal components. Below, we use the `plot_pc` function from question 3c to plot the 1st row of V^T in the cell below.

Here by "1st row" we mean the row that is used to generate `pc1`, and by "2nd row" we mean the row that is used to generate `pc2`.

Note: If you want to adjust the size of your plot for a single figure, you can set `plt.figure(figsize=(x, y))` before creating the figure.

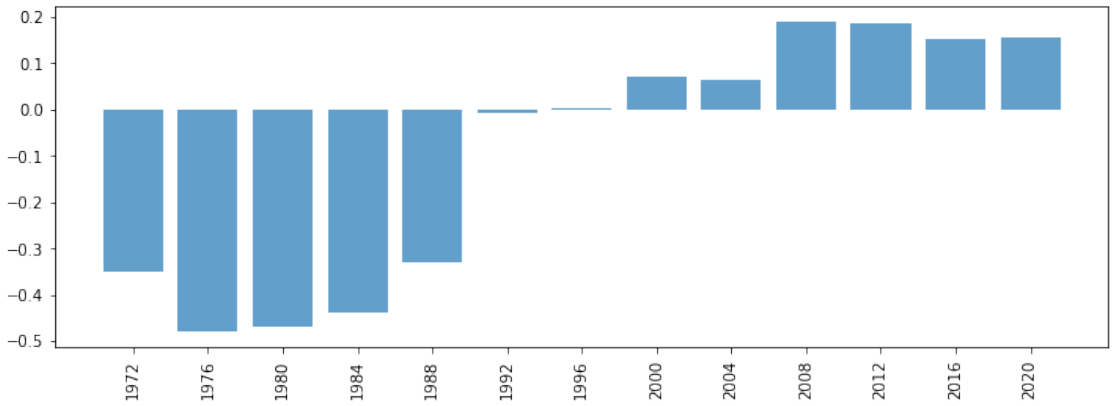
Note: If you get an error when running this cell, make sure you are properly assigning the `vt_q3` variable in Question 3d.

[55]:
```
plt.figure(figsize=(12, 4))
plot_pc(list(df_standardized.columns), vt_q3, 0);
```



In the cell below, plot the the 2nd row of V^T.

[58]:
```
with plt.rc_context({"figure.figsize": (12, 4)}):
    plot_pc(list(df_standardized.columns), vt_q3, 1);
```



1.25 Question 3h

Using your plots from question 3g as well as the original table, give a description of what it means to have a relatively large positive value for `pc1` (right side of the 2D scatter plot), and what it means to have a relatively large positive value for `pc2` (top side of the 2D scatter plot).

In other words, what is generally true about a state with relatively large positive value for `pc1`? For a large positive value for `pc2`?

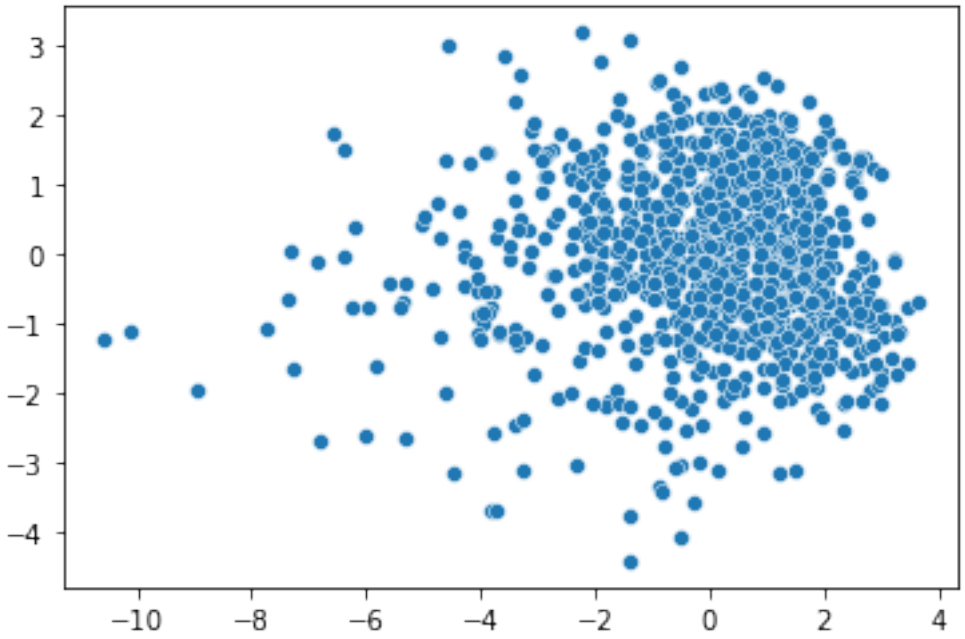
Note: `pc2` is pretty hard to interpret, and the staff doesn't really have a consensus on what it means either. We'll be nice when grading.

Note: Principal components beyond the first are often hard to interpret (but not always; see question 1 earlier in this homework).

To interpret pc1, it can be simply found out that the states with larger pc1 are less likely is going to vote for the Republican party. The state will tend to vote Republican with small pc1. It could be told by the negative principle component values in the graph, showing a negative correlation. By interpreting PC2, I found out that state with the larger the PC2 value tend to support a single party during most of the elections and it rawly changes. But for state with smaller PC2 values. things go the opposite way. Furthermore, I noticed that the elections have two separate term(first half and second half), and these two terms have contrasting relations.

[66]:
```
# feel free to use this cell for scratch work. If you need more scratch space,␣
↪add cells *below* this one.
pcs = u*s
sns.scatterplot(x=pcs[:, 0], y = pcs[:, 1])
# Make sure to put your actual answer in the cell above where it says "Type␣
↪your answer here, replacing this text"
```

[66]:
```
<AxesSubplot:>
```

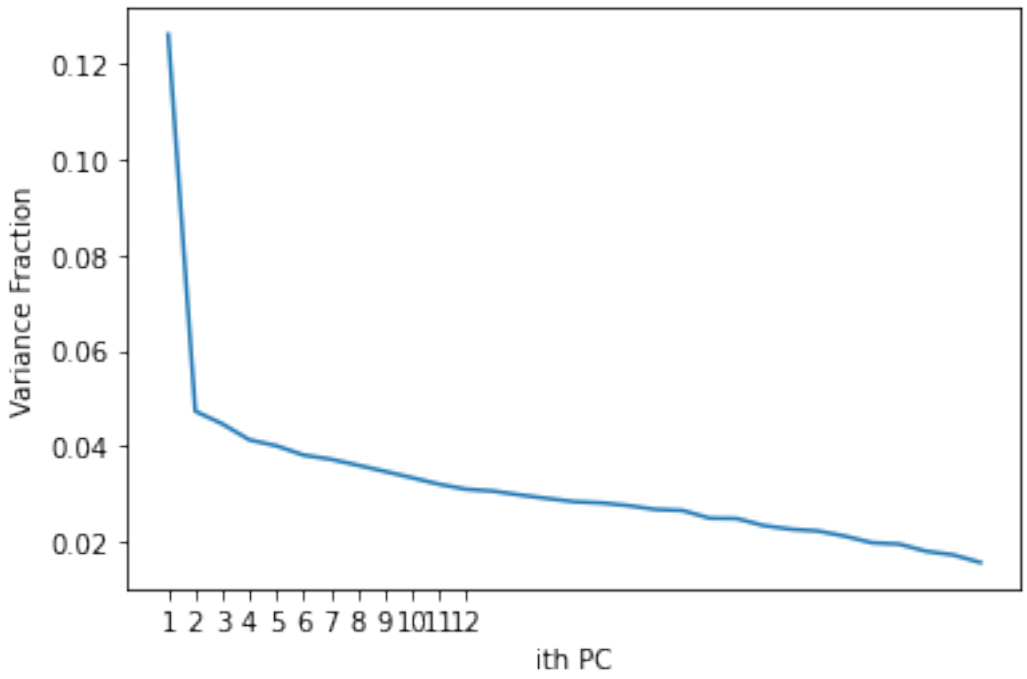


1.26 Question 3i

To get a better sense of whether our 2D scatterplot captures the whole story, create a scree plot for this data. On the y-axis plot the fraction of the total variance captured by the ith principal component. You should see that the first two principal components capture much more of the variance than we were able to capture when using the Data 100 Midterm 1 data. It is partially for this reason that the 2D scatter plot was so much more useful for this dataset.

Hint: Your code will be very similar to the scree plot from problem 1d. Be sure to label your axes appropriately!

[70]:
```
plt.plot(np.arange(1,32), s**2/sum(s**2));
plt.xticks(np.arange(1,13), np.arange(1,13));
plt.xlabel('ith PC');
plt.ylabel('Variance Fraction');
```



1.27 Question 4: Interpreting Diagonal Banding (Optional)

As an optional exercise, try to figure out why we saw the diagonal bands in the scatterplot from question 2b. One approach is to use `plot_pc` on the 1st and 2nd pc to understand how they're created from the data. The answer is somewhat lame, but you might find the exercise useful.

Type your answer here, replacing this text.

[74]:
```
### Feel free to use these cells for scratch work
```

[75]:
```

```

To double-check your work, the cell below will rerun all of the autograder tests.

[ ]:
```
grader.check_all()
```

1.28 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

[ ]:
```
# Save your notebook first, then run this cell to export your submission.
grader.export()
```