538'S STAR WARS SURVEY

Load the data. You'll need to put the data file in your current working directory, or equivalent to this load() function, you can double-click it. Use the str() function to take a peek at the types of variables in the data.

The survey (and it's results) are described here:
https://fivethirtyeight.com/features/americas-favorite-star-wars-movies-and-least-favorite-characters/

```
> load('StarWarsSurvey.RData')
> str(starwars)
```

```
'data.frame':       1186 obs. of  38 variables:
 $ RespondentID                 : num  3.29e+09 3.29e+09 3.29e+09 3.29e+09 3.29e+09 ...
 $ SeenAnyMovie                 : Factor w/ 3 levels "No","Yes",NA: 2 1 2 2 2 2 2 2 2 2 ...
 $ AreYouAFan                   : Factor w/ 3 levels "No","Yes",NA: 2 3 1 2 2 2 2 2 2 1 ...
 $ SeenEpisodeI                 : num  1 0 1 1 1 1 1 1 1 0 ...
 $ SeenEpisodeII                : num  1 0 1 1 1 1 1 1 1 1 ...
 $ SeenEpisodeIII               : num  1 0 1 1 1 1 1 1 1 0 ...
 $ SeenEpisodeIV                : num  1 0 0 1 1 1 1 1 1 0 ...
 $ SeenEpisodeV                 : num  1 0 0 1 1 1 1 1 1 0 ...
 $ SeenEpisodeVI                : num  1 0 0 1 1 1 1 1 1 0 ...
 $ RankEpisodeI                 : num  3 0 1 5 5 1 6 4 5 1 ...
 $ RankEpisodeII                : num  2 0 2 6 4 4 5 5 4 2 ...
 $ RankEpisodeIII               : num  1 0 3 1 6 3 4 6 6 3 ...
 $ RankEpisodeIV                : num  4 0 4 2 2 6 3 3 2 4 ...
 $ RankEpisodeV                 : num  5 0 5 4 1 5 1 2 1 5 ...
 $ RankEpisodeVI                : num  6 0 6 3 3 2 2 1 3 6 ...
 $ ViewHanSolo                  : num  5 0 4 5 5 5 5 5 5 3 ...
 $ ViewLukeSkywalker            : num  5 0 4 5 4 5 5 4 2 5 ...
 $ ViewPrincessLeiaOrgana       : num  5 0 4 5 4 5 4 5 4 5 ...
 $ ViewAnakinSkywalker          : num  5 0 4 5 2 5 4 3 4 5 ...
 $ ViewObiWanKenobi             : num  5 0 4 5 5 5 5 5 4 5 ...
 $ ViewEmperorPalpatine         : num  5 0 0 4 1 3 5 1 5 2 ...
 $ ViewDarthVader               : num  5 0 0 5 4 5 5 2 5 5 ...
 $ ViewLandoCalrissian          : num  0 0 0 4 3 3 5 3 5 2 ...
 $ ViewBobaFett                 : num  0 0 0 2 5 4 5 4 5 2 ...
 $ ViewC3P0                     : num  5 0 0 5 4 4 4 4 3 5 ...
 $ ViewR2D2                     : num  5 0 0 5 4 4 5 4 4 5 ...
 $ ViewJarJarBinks              : num  5 0 0 5 1 4 2 1 1 5 ...
 $ ViewPadmeAmidala             : num  5 0 0 5 4 3 4 2 2 2 ...
 $ ViewYoda                     : num  5 0 0 5 4 5 5 5 4 5 ...
 $ WhichCharacterShotFirst      : Factor w/ 4 levels "Greedo","Han",..: 3 4 3 3 1 2 2 2 2 3 ...
 $ FamiliarWithExpandedUniverse: Factor w/ 3 levels "No","Yes",NA: 2 3 1 1 2 2 2 1 1 1 ...
 $ FanOfExpandedUniverse        : Factor w/ 3 levels "No","Yes",NA: 1 3 3 3 1 1 1 3 3 3 ...
 $ FanOfStarTrek                : Factor w/ 3 levels "No","Yes",NA: 1 2 1 2 1 2 1 2 1 1 ...
 $ Gender                       : Factor w/ 3 levels "Female","Male",..: 2 2 2 2 2 2 2 2 2 2 ...
 $ Age                          : Factor w/ 5 levels "> 60","18-29",..: 2 2 2 2 2 2 2 2 2 2 ...
 $ HouseholdIncome              : Factor w/ 6 levels "$0 - $24,999",..: 6 1 1 2 2 4 6 6 1 4 ...
 $ Education                    : Factor w/ 6 levels "Bachelor degree",..: 3 1 3 5 5 1 3 3 5 5 ...
 $ Location                     : Factor w/ 10 levels "East North Central",..: 7 9 8 8 8 3 1 7 7 6 ...
```

There are lots of factor variables which are not suited for PCA input (PCA functions will undoubtedly want you to input numeric data, regardless of software platform). While it's not the *most* principled thing to use dummy variables as input to PCA (we prefer to think about PCA in terms of elliptically distributed numeric data), it's common and often works. The math holds just the same - we still get a projection of our data that is of maximal variance. We're going to proceed with that approach here - though you are welcome to further exploration without the binary and factor columns.

To create the dummy variables for all factors, we make a model matrix with **one-hot encoded** factors (this means we don't leave out a reference column). The contrasts.arg= option in the following model.matrix() function will do the one-hot encoding.

```
> starwars.x = model.matrix(RespondentID~. ,
+             contrasts.arg = lapply(starwars[,sapply(starwars,is.factor) ],
+                                    contrasts, contrasts=FALSE),
+             data = starwars)
```

Print the dimensions of the final matrix (with dummy variables enumerated):

```
> dim(starwars.x)
```

```
[1] 1186   76
```

This is the *true dimensionality* of your data! If you were asked "how many columns are in your dataset?" you would be wise to answer 76 rather than 38!

Computing the PCA

The prcomp() function is the one I most often recommend for reasonably sized principal component calculations in R. This function returns a list with class "prcomp" containing the following components (from help prcomp):

1. sdev: the standard deviations of the principal components (i.e., the square roots of the eigenvalues of the covariance/correlation matrix, though the calculation is actually done with the singular values of the data matrix).

2. rotation: the matrix of *variable loadings* (i.e., a matrix whose columns contain the eigenvectors). The function princomp returns this in the element loadings.

3. x: if retx is true *the value of the rotated data (i.e. the scores)* (the centred (and scaled if requested) data multiplied by the rotation matrix) is returned. Hence, cov(x) is the diagonal matrix $diag(sdev^2)$. For the formula method, napredict() is applied to handle the treatment of values omitted by the na.action.

4. center, scale: the centering and scaling used, or FALSE.

The option scale = TRUE inside the prcomp() function instructs the program to use **correlation PCA**. The **default is covariance PCA**.

```
> pca = prcomp(starwars.x, scale =T)
```

An error message! Cannot rescale a constant/zero column to unit variance. Solution: check for columns with zero variance and remove them. Recheck dimensions of the matrix to see how many columns we lost.

```
> starwars.x = starwars.x[,apply(starwars.x, 2, sd)>0 ]
> dim(starwars.x)
```

```
[1] 1186   74
```

We can now compute the principal components of the matrix.

Questions to Explore for this Case Study

1. Project this data onto the first two principal components. *Note: It is common for points to have the same or very similar scores along PCs, particularly when each variable has a limited range of values (i.e. binary or likert scale). This means that many points can be overlapping on a plot and that can be misleading. A common solution is to "jitter" the plot which merely adds some random error to the placement of the points so that you can see overlapping groups of points with more clarity. The `jitter()` function can help with this. Try `plot(jitter(data,1))` for instance*

2. How would you describe what the first principal component is telling you about the data? It may be helpful here to explore groups of interest from the plot in terms of their original data. Consider subsetting the rows of the original dataset, textitstarwars, based upon their coordinate on the first principal component, just so that you can view the subsetted table to see what it looks like.

3. Subset the data according to the first principal component, taking what you might call "your observations of interest" as a new dataset. For this, I mean subset the rows of starwars.x according to their scores on the first PC.

4. Repeat the procedure of finding the first principal components of this new subsetted data.

5. How much variance can be explained by the first 2 principal components? Create a screeplot of the eigenvalues. Is there any noticeable elbow in it?

6. Explore the biplot and note anything of interest as it pertains to the Star Wars Saga.