```
In [1]: import numpy as np
        import numpy.random as npr
        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline
```

Exact Permutation Tests

Let's start with a simple example. Consider the average course grades for EEE 5544 over the past 6 years, omitting 2017 (when I didn't teach it):

Year	Grade
2013	74.1
2014	74.5
2015	79.4
2016	79.0
2018	78.4

Let's check the means for the first 2013-2014 vs 2015-2018:

```
In [2]: grades1=np.array([74.1,74.5])
        grades2=np.array([79.4,79,78.4])
```

```
In [3]: grades1.mean(), grades2.mean()

Out[3]: (74.3, 78.93333333333334)
```

```
In [4]: diff= grades2.mean()- grades1.mean()
        diff

Out[4]: 4.633333333333334
```

Has the average grade increased over time, or could this just be attributable to the small number of samples?

We could perform bootstrap resampling under the null hypothesis, but the data is so small, we might get repeats of the same samples.

An alternative is to pool the data and try **all** the ways to rsample the data into samples of size 2 and 3:

Exact Permutation Test

In Fisher's exact permutation test (for binary hypothesis testing), we:

1. Pool the data
2. Find every partition of the data into two subsets (with sizes equal to the original samples)
3. Measure the sample statistics for each new sample
4. Determine whether the new sample statistics are as extreme as the original observation

First, we demonstrate how to find all the subsets of a certain size for a list:

In [5]:
```
import itertools
```

In [6]:
```
itertools.combinations([1,2,3,4,5,6],3)
```

Out[6]:
```
<itertools.combinations at 0x11a106c28>
```

In [7]:
```
samples=list(itertools.combinations([1,2,3,4,5,6],3))
samples
```

Out[7]:
```
[(1, 2, 3),
 (1, 2, 4),
 (1, 2, 5),
 (1, 2, 6),
 (1, 3, 4),
 (1, 3, 5),
 (1, 3, 6),
 (1, 4, 5),
 (1, 4, 6),
 (1, 5, 6),
 (2, 3, 4),
 (2, 3, 5),
 (2, 3, 6),
 (2, 4, 5),
 (2, 4, 6),
 (2, 5, 6),
 (3, 4, 5),
 (3, 4, 6),
 (3, 5, 6),
 (4, 5, 6)]
```

In [8]:
```
len(samples)
```

Out[8]:
```
20
```

In [9]:
```
from scipy.special import binom
```

```
In [10]: binom(6,3)
Out[10]: 20.0
```

Thus, we can pull all the samples of size 2 from our data:

```
In [11]: pooled=np.hstack((grades1,grades2))
         pooled
Out[11]: array([74.1, 74.5, 79.4, 79. , 78.4])
```

```
In [12]: samples1=list(itertools.combinations(pooled,2))
         samples1
Out[12]: [(74.1, 74.5),
          (74.1, 79.4),
          (74.1, 79.0),
          (74.1, 78.4),
          (74.5, 79.4),
          (74.5, 79.0),
          (74.5, 78.4),
          (79.4, 79.0),
          (79.4, 78.4),
          (79.0, 78.4)]
```

The trick is that for each sample, we need to find the remaining set to go in the other sample. We can use numpy's `setxor1d` method

```
In [13]: samples1[0],np.setxor1d(pooled,samples1[0])
Out[13]: ((74.1, 74.5), array([78.4, 79. , 79.4]))
```

```
In [14]: for samples1 in itertools.combinations(pooled,2):
             samples2= np.setxor1d(pooled,samples1)
             print(samples1,samples2)

         (74.1, 74.5) [78.4 79.  79.4]
         (74.1, 79.4) [74.5 78.4 79. ]
         (74.1, 79.0) [74.5 78.4 79.4]
         (74.1, 78.4) [74.5 79.  79.4]
         (74.5, 79.4) [74.1 78.4 79. ]
         (74.5, 79.0) [74.1 78.4 79.4]
         (74.5, 78.4) [74.1 79.  79.4]
         (79.4, 79.0) [74.1 74.5 78.4]
         (79.4, 78.4) [74.1 74.5 79. ]
         (79.0, 78.4) [74.1 74.5 79.4]
```

Now we are ready to conduct our exact permutation test:

```
In [15]: perm_count = 0
         event_count = 0
         for samples1 in itertools.combinations(pooled,2):
             samples2= np.setxor1d(pooled,samples1)
             mean1 = np.mean(samples1)
             mean2 = np.mean(samples1)
             sample_diff = mean2-mean1
             perm_count+=1
             if abs(sample_diff) >= diff:
                 event_count+=1

         print("Prob. that mean difference is >=", diff, "=~", event_count/perm_c
         ount)
```

```
Prob. that mean difference is >= 4.633333333333334 =~ 0.0
```

There is not enough data to support that the average score has increased in the last 3 years of the study.

(Since there are only 10 permutations, we can never get a p-value less than 0.1 with data sets of 2 values and 3 values)

Unlike resampling, this approach tries **every** way or redistributing the pooled data. So why not always use it?

Consider the previous example, where the data for the 50 states were split into two clusters of size 42 and 8. How many different combinations of samples would be created in the exact permutation test?

```
In [16]: binom(50,8)
Out[16]: 536878650.0
```

```
In [17]: binom(50,42)
Out[17]: 536878650.0
```

So, how can we handle a case like this?

Monte Carlo Permutation Test

Let's consider the following question that should be of interest to engineers: Do males score higher on standardized high school math and science tests than females?

We will use data from the "High School & Beyond (HS&B)" survey conducted y the National Center for Education Statistics:

https://nces.ed.gov/surveys/hsb/index.asp (https://nces.ed.gov/surveys/hsb/index.asp)

We are using a CSV file with 200 randomly selected observations from that data set. The CSV file is available here:

https://github.com/rpruim/OpenIntro/blob/master/data/hsb2.csv (https://github.com/rpruim/OpenIntro/blob/master/data/hsb2.csv)

A brief discussion of the different fields is available at

http://www.philender.com/courses/762/notes1/about_hsb2.html (http://www.philender.com/courses/762/notes1/about_hsb2.html)

In [18]:
```
df=pd.read_csv("hsb2.csv")
```

In [19]:
```
df;
```

We want to partition this dataframes into two seperate dataframes according to gender. This is easy to do in pandas, but it looks a little strange. First we get a boolean Series that contains True for whichever rows we want to keep:

In [20]:
```
df["gender"]=="male";
```

Now, if we pass that Series as indices to the original dataframe, it will return a new dataframe with only those rows:

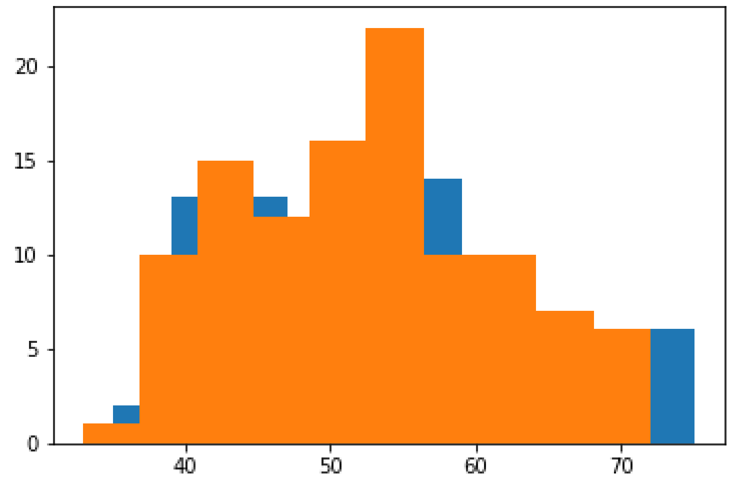
In [21]:
```
males=df[df["gender"]=="male"]
males;
```

In [22]:
```
females=df[df["gender"]=="female"]
females;
```

Let's start with math scores:

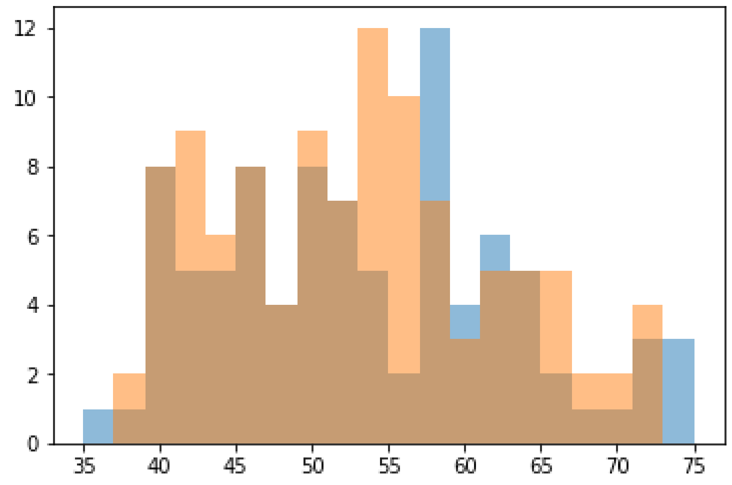
In [23]:
```
plt.hist(males["math"])
plt.hist(females["math"])
```

Out[23]:
```
(array([ 1., 10., 15., 12., 16., 22., 10., 10.,  7.,  6.]),
 array([33. , 36.9, 40.8, 44.7, 48.6, 52.5, 56.4, 60.3, 64.2, 68.1, 72. 
]),
 <a list of 10 Patch objects>)
```



In [24]:
```
counts,my_bins,_ = plt.hist(males["math"], bins=20, alpha=0.5)
plt.hist(females["math"], bins = my_bins, alpha=0.5)
```

Out[24]:
```
(array([ 0.,  2.,  8.,  9.,  6.,  8.,  4.,  9.,  7., 12., 10.,  7., 
3.,
          5.,  5.,  5.,  2.,  2.,  4.,  0.]),
 array([35., 37., 39., 41., 43., 45., 47., 49., 51., 53., 55., 57., 5
9.,
        61., 63., 65., 67., 69., 71., 73., 75.]),
 <a list of 20 Patch objects>)
```



In [25]:
```
np.mean(males["math"]),np.mean(females["math"])
```

Out[25]:
```
(52.94505494505494, 52.39449541284404)
```

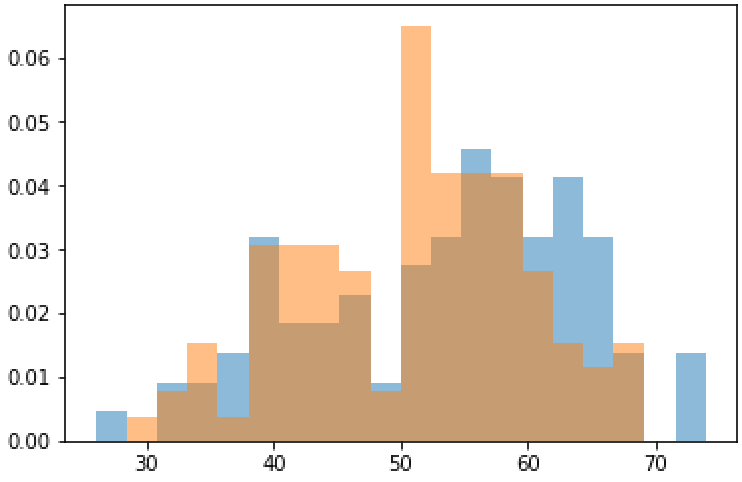
What can you infer from this plot?

I'm going to guess that this is probably not statistically significant, we can test more later

Let's consider science scores:

```
In [26]: counts,my_bins,_ = plt.hist(males["science"], bins=20, alpha=0.5, densit
         y=True)
         plt.hist(females["science"], bins = my_bins, alpha=0.5, density=True)
```

```
Out[26]: (array([0.        , 0.00382263, 0.00764526, 0.01529052, 0.00382263,
                 0.03058104, 0.03058104, 0.03058104, 0.02675841, 0.00764526,
                 0.06498471, 0.04204893, 0.04204893, 0.04204893, 0.02675841,
                 0.01529052, 0.01146789, 0.01529052, 0.        , 0.        ]),
          array([26. , 28.4, 30.8, 33.2, 35.6, 38. , 40.4, 42.8, 45.2, 47.6, 50.
         ,
                 52.4, 54.8, 57.2, 59.6, 62. , 64.4, 66.8, 69.2, 71.6, 74. ]),
          <a list of 20 Patch objects>)
```



```
In [27]: np.mean(males["science"]),np.mean(females["science"])
```

```
Out[27]: (53.23076923076923, 50.69724770642202)
```

```
In [28]: diff = np.mean(males["science"]) - np.mean(females["science"])
         diff
```

```
Out[28]: 2.533521524347215
```

Is this result statistically significant?

How many different sample combinations are there in the exact permutation test?

In [29]:
```
binom(200,91)
```

Out[29]:
```
4.040047748665152e+58
```

Instead, let's sample 10,000 of the permutations. Since there are so many, we will randomly select permutations, so there is some small probability of repeat

To choose the samples, we will permute the pooled data and then subdivide into the appropriate sizes:

In [30]:
```
pooled = df["science"]
```

In [31]:
```
diff = np.mean(males["science"]) - np.mean(females["science"])
pooled = df["science"]
num_sims = 100000
event_count = 0

for sim in range(num_sims):
    perm = npr.permutation(pooled)
    # first 91 and last 109
    male_sample = perm[:len(males)]
    female_sample = perm[len(males):]
    sample_diff = male_sample.mean()-female_sample.mean()
    if abs(sample_diff) >= diff:
        event_count+=1

monte_p_val = event_count/num_sims

print("Prob. of seeing mean diff >=", diff,"under H0 is =~", monte_p_val
)
```

```
Prob. of seeing mean diff >= 2.533521524347215 under H0 is =~ 0.07193
```

What is your conclusion?

The result is not statistically significant at the p < 0.01 level. We cannot be sure that the observed mean difference is caused by a true difference in the underlying populations.

In [32]:
```
# Here are all the average values
tests=["read","write","math","science","socst"]
for test in tests:
    print(test.ljust(8),"--\t", "Males:",males[test].mean(),"\tFemales:"
,females[test].mean())
```

```
read     --      Males: 52.824175824175825      Females: 51.73394495412
844
write    --      Males: 50.120879120879124      Females: 54.99082568807
339
math     --      Males: 52.94505494505494       Females: 52.39449541284
404
science  --      Males: 53.23076923076923       Females: 50.69724770642
202
socst    --      Males: 51.79120879120879       Females: 52.91743119266
055
```

Defn: A collection of events A1,A2,...,An partitions the sample space (s) iff: 1) Ai intersection Aj = null, for all i,j in {1,2,...,N} 2) The union of each event together = S the events make a "tiling" of S

Lecture 13 Assignment

1. Conduct a bootstrap resampling test for statistical significance of the difference in average science scores between the genders. Compare the p-value for the bootstrap test vs. that found via the Monte Carlo permutation test.

In [33]:
```
diff = np.mean(males["science"]) - np.mean(females["science"])
pooled = df["science"]
num_sims=100000
event_count=0
for sim in range(num_sims):
    sample_males=npr.choice(pooled,len(males))
    sample_females=npr.choice(pooled,len(females))
    sample_diff =sample_males.mean()-sample_females.mean()
    if abs(sample_diff) >= diff:
        event_count+=1

bootstrap_p_val = event_count/num_sims

print("Under null hypothesis, observe effect this large with prob. ",boo
tstrap_p_val, ".", sep="")
print("The $p$-value for the bootstrap test is ",bootstrap_p_val,", as show
n above and the $p$-value found via the Monte Carlo Permutation test is ",m
onte_p_val,". As you can see, there is very little difference between th
em and either technique works well for this data.",sep="")
```

```
Under null hypothesis, observe effect this large with prob. 0.07024.
The p-value for the bootstrap test is 0.07024, as shown above and the p
-value found via the Monte Carlo Permutation test is 0.07193. As you ca
n see, there is very little difference between them and either techniqu
e works well for this data.
```

1. Generate a table of the median values for all tests, by gender.

In [34]:
```
females=df[df["gender"]=="female"]
males=df[df["gender"]=="male"]
tests = ["read", "write", "math", "science"]
genders = [males, females]

print("gender\tread\twrite\tmath\tscience",sep="")
for i in range(0,len(genders)):
    gender = "males" if i==0 else "females"
    print(gender,"\t",sep="",end="")
    for test in tests:
        print(np.median(genders[i][test]),"\t",sep="",end="")
    print("")
```

```
gender  read    write   math    science
males   52.0    52.0    52.0    55.0
females 50.0    57.0    53.0    50.0
```

1. What is the median writing score for men vs women? Using a Monte Carlo permutation test (not bootstrap resampling), answer the following questions. Is the difference statistically significant at the p<0.01 level? What is the 99% confidence interval for the difference in medians?

In [35]:
```
np.median(females["write"])
median_diff = np.median(males["write"]) - np.median(females["write"])
print("The median writing score for men vs. women is ",np.median(males[
"write"])," vs. ", np.median(females["write"]),", with a median differen
ce of ", median_diff,".", sep="")
pooled = df["write"]
num_sims = 10000
bs_stats=[]
event_count = 0
for sim in range(num_sims):
    perm = npr.permutation(pooled)
    # first 91 and last 109
    male_sample = perm[:len(males)]
    female_sample = perm[len(males):]
    sample_diff = np.median(male_sample)-np.median(female_sample)
    bs_stats+=[sample_diff]
#     print(sample_diff,end=" ")
    if sample_diff**2 >= median_diff**2:
        event_count+=1

print("Prob. of seeing median diff >= ", abs(median_diff)," under H0 is
 = ~", event_count/num_sims,".", sep="")
print("The difference is NOT statistically significant at the p < 0.01 l
evel.")

# calculate 99% confidence interval
bs_stats.sort()
lower=int(len(bs_stats)*0.01/2)
upper=int(len(bs_stats)-lower-1)
bs_stats[lower],bs_stats[upper]
print('The 99% confidence interval is [',bs_stats[lower],', ',bs_stats[u
pper],'].',sep="")
```

```
The median writing score for men vs. women is 52.0 vs. 57.0, with a med
ian difference of -5.0.
Prob. of seeing median diff >= 5.0 under H0 is = ~0.075.
The difference is NOT statistically significant at the p < 0.01 level.
The 99% confidence interval is [-5.0, 5.0].
```