Lecture 17

- **Goodness-of-fit measures**
- **Contingency tables**: test of independence

Exam 2

Wednesday, November 2 @ 8:20PM - 10:20PM in FLG 0260 and FLG 0270

Students whose last name begins with A-M should go to FLG 0260, and those whose last name begins N-Z should go to FLG 0270.

Exam 2 Review Lecture: Tuesday, November 1 (first half)

Two **practice exams** (from last semester) is available in the Assignment-Solutions repo.

- Coverage: Lectures 8 to 17 (modules 6 and 7).

Exam Details

All exams will have 2 parts:

Part 1 - Analytical.

- You are allowed 1-page letter-sized front and back of formulas (handwritten or typed).
- You are allowed a scientific calculator.
- **TOTAL TIME:** 1 hour

Part 2 - Simulation.

- Bring the computer you have been using with Anaconda installed.
- This part is open-book. You are allowed access to the textbooks and lecture materials (including assignments).
- You are allowed to use the internet, if needed.
- *Recommended*: create a python "cheat sheet" where you will add useful functions, simulations and other Python implementations.
- **TOTAL TIME:** 1 hour

Communications between students or anyone else during the exam is considered cheating. Turn off all Slack notifications and other communications channels!

Exam 2 Coverage

Exam 2 will cover all materials from Lectures 8-17. These include:

1. **Random Variables**

    - What are random variables?
    - Defining discrete and continuous RVs
    - Important discrete RVs:
        - Bernoulli, Binomial, Geometric, Poisson, Uniform
    - Probability Mass Functions (PMFs)
    - Important continuous RVs:
        - Uniform, Exponential, Gaussian
    - Probability Density Functions (PDFs)
    - Cumulative Distribution Functions (CDFs)
    - Expected Value
    - Moments of RV
        - Mean, Variance, Skewness, Kurtosis
    - Central Limit Theorem (CLT)
    - Kernel Density Estimation (KDE)
        - types of kernels (Gaussian, exponential, etc.)
        - bandwidth
    - Point Conditioning
        - Conditional Probability for continuous RVs
        - Law of Total Probability for continuous RVs
        - Bayes' Rule version for continuous RVs
    - Optimal Decisions
2. **Experimental Design, Decision and Analysis**

    - Sum of Independent Gaussian RVs
    - Hypothesis test for the difference of the means -- Gaussian, from CLT
    - Binary T-test
    - Hypothesis test errors
        - Type I error: False Positives
        - Type II error: False Negatives
    - Performance Trade-offs
        - Finding threshold for probability of false rejection, false acceptance
        - ROC curve
        - AUC of the ROC curve
    - Goodness-of-fit measures
        - For discrete RVs: chi-squared test
        - For continuous RVs: r^2 of the Q-Q plot
        - Q-Q plot vs probability plot
    - Contingency Tables
        - `pandas` dataframe manipulation
        - Expected contingency table
        - Degrees of freedom
        - Test of independence: Chi-Squared statistic

How to prepare for exam

This is a suggestion only.

1. Review/read all Notebooks.

2. Create your formula sheet for part 1 and "common functions" Notebook for part 2.
   - You will be given a Q-function table.
3. Review/redo exercises from HW3.
4. Review/redo exercises from SA2 and SA3.
5. Review/redo exercises from Lecture 12 Recitation and Lecture 18 Recitation (upcoming).
6. Solve practice exams 2.

```
In [ ]:
```

Last class

- We saw the **T-test** (unknown variance)
- **Tradeoffs in hypothesis testing** : how do our results depend on the level of alpha (confidence)?

T-test

Unknown variance: we have to approximate the variance.

If we use our *unbiased* estimator for the variance, then the distribution of

$$\frac{\hat{\mu} - \mu}{S_{N-1}/\sqrt{N}}$$

has a **Student's t-distribution with $N-1$ degrees of freedom (dof).**

- The density and distribution functions for the **Student's t-distribution** are shown on its Wikipedia page (https://en.wikipedia.org/wiki/Student's_t-distribution).
- Unlike the Gaussian distribution, the distribution function for Student's t-distribution is in closed form for several values of ν (degrees of freedom or dof).

Errors and Performance Tradeoffs in Hypothesis Testing

- In binary hypothesis testing, there are two types of errors:
  1. **False Alarm** (Type I Error, also called *False Positive*)
     - occurs if we accept the alternate hypothesis (H_1) when it is not true; or reject the null hypothesis (H_0) when it is in fact true.
     - we will use the notation
       $$P_{fa} = P(\text{false alarm})$$
     - $P_{fa} = \alpha$
  2. **Miss** (Type II Error, also called *False Negative*)
     - occurs if we reject an alternative hypothesis (H_1) when it is actually true; or accept the null hypothesis (H_0) when it is in fact false.
     - we will use the notation
       $$P_m = P(\text{miss})$$
- When performing a hypothesis test, there is always a tradeoff between these two types of errors

- The tradeoff is controlled by choosing the significance level, α, to which the p-value is compared with
  - the value α is the probability that we will reject the null hypothesis, H_0 when it is in fact true
  - equivalently, it is the probability of accepting the alternative hypothesis, H_1, when H_1 is false

Visualizing Tradeoffs in Hypothesis Testing: ROC Curves

- We can visualize the relation between these types of errors using a ROC curve
  - ROC stands for *receiver operating characteristic*
  - ROC curves were developed for RADAR systems but are widely used in fields of statistical tests, such as biomedicine
- In ROC curves, we do not plot P_{fa} vs P_m
- Instead:
  - the x-axis is **FPR (false positive rate)**
$$\text{FPR} = P_{fa}$$
  - the y-axis is **TPR (true positive rate)**
$$\text{TPR} = 1 - P_m$$

In [2]:
```python
import numpy as np
import numpy.random as npr
import scipy.stats as stats
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')
```

Today

- **Goodness-of-fit measures**
- **Contingency tables**: test of independence

Testing Goodness-of-Fit

The goodness of fit of a statistical model describes how well it fits a set of observations. Measures of goodness of fit typically summarize the discrepancy between observed values and the values expected under the model in question.

Testing Whether Data Comes from a Distribution: Discrete Distributions

- Given a set of random data and a proposed model, how could we determine if the data could have reasonably come from that model?
- For example, given values from a six-sided die, how could you tell if the die is fair?

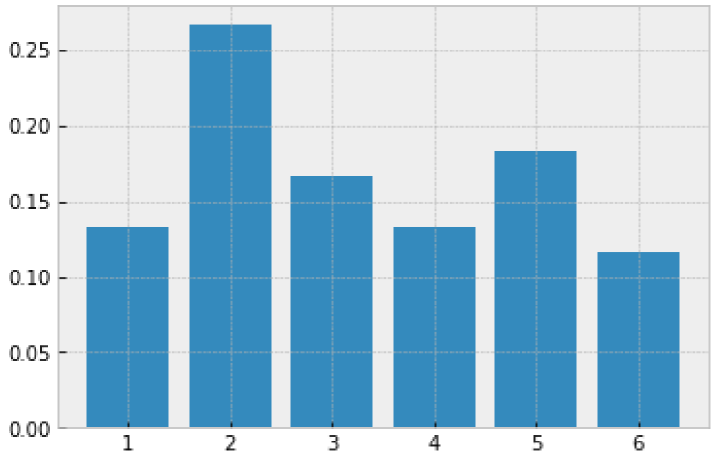
Example 1

In [51]:
```python
# Draw 60 values from a fair 6-sided die:

dice = npr.randint(1,7,size=60)
dice
```

Out[51]:
```
array([3, 4, 5, 5, 1, 4, 2, 2, 2, 3, 4, 3, 2, 2, 3, 1, 6, 3, 1, 3, 3, 6,
       5, 5, 3, 3, 1, 5, 5, 2, 2, 5, 1, 6, 4, 2, 6, 6, 2, 2, 2, 5, 2, 1,
       2, 6, 4, 5, 1, 4, 5, 2, 2, 4, 4, 6, 5, 1, 3, 2])
```

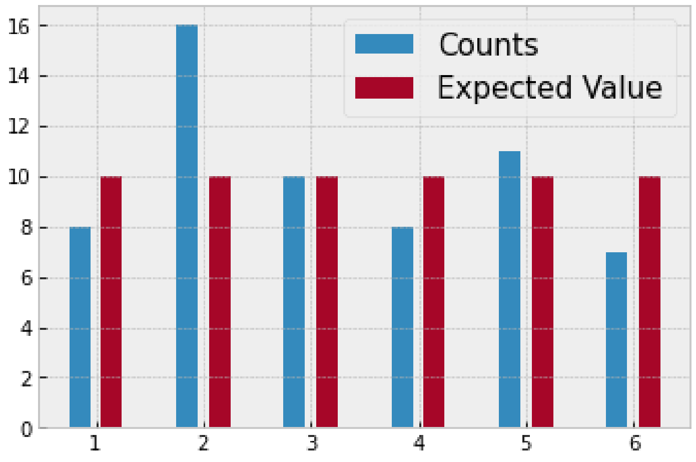
In [52]:
```python
vals, counts = np.unique(dice, return_counts=True)
```

In [53]:
```python
plt.bar(vals, counts/60);
```



- Even after 60 rolls, the numbers still vary significantly
- We could compare them to the expected values:

In [54]:
```python
plt.bar(vals-0.15, counts, width=0.2, label='Counts')
plt.bar(vals+0.15, [60*1/6], width=0.2, label='Expected Value')
plt.legend(fontsize=15);
```



Example 2

The file "baseball.pickle" contains the birth months of major league baseball players.

> `pickle` [(https://docs.python.org/3/library/pickle.html)](https://docs.python.org/3/library/pickle.html) is a Python object serialization library.

In [3]:
```
import pickle
```

In [4]:
```
pf = open('baseball.pickle','rb')
```

In [5]:
```
df = pickle.load(pf)
pf.close()
```

In [6]:
```
df
```

Out[6]:

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	
num_players	1398.0	1213.0	1350.0	1221.0	1229.0	1157.0	1317.0	1558.0	1382.0	1526.0	1406.0	13

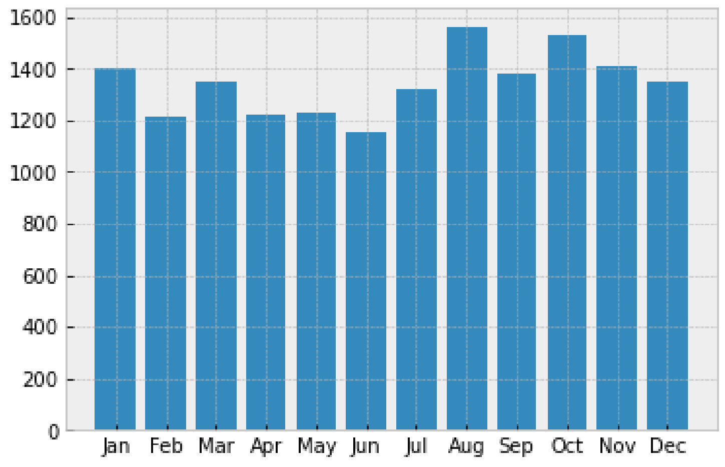
In [8]:
```
baseball = df.loc['num_players'].to_numpy()
baseball
```

Out[8]:
```
array([1398., 1213., 1350., 1221., 1229., 1157., 1317., 1558., 1382.,
       1526., 1406., 1350.])
```

In [9]:
```
months = df.columns.to_numpy()
print(months)
```

```
['Jan' 'Feb' 'Mar' 'Apr' 'May' 'Jun' 'Jul' 'Aug' 'Sep' 'Oct' 'Nov' 'Dec']
```

In [10]:
```
plt.bar(months, baseball);
```



Observations and Comments:

- Note that more MLB players are born in August than any other month

- Some people claim that this is because in little league baseball through 2006, the cutoff for determing a player's age eligibility was July 31st
- That is, a player who was turning 9 that year would not be eligible to play in the 8 & Under league if their birthday was before August 1st.
- Thus, players with July birthdays were the youngest (and, on average, smallest) in their leagues, and player with August birthdays were the oldest (and, on average, largest)

Let's start by comparing the data values in the cells to the expected values for those cells

- **We assume a uniform distribution of birthdays over the year**

In [13]:
```
total_players = int(sum(baseball))

total_players
```

Out[13]:
```
16107
```

Then we can get the expected number of birthdays in a month as the probability a player is born in a month (which is just the number of days in the month divided by 365) times the total number of players in the table:

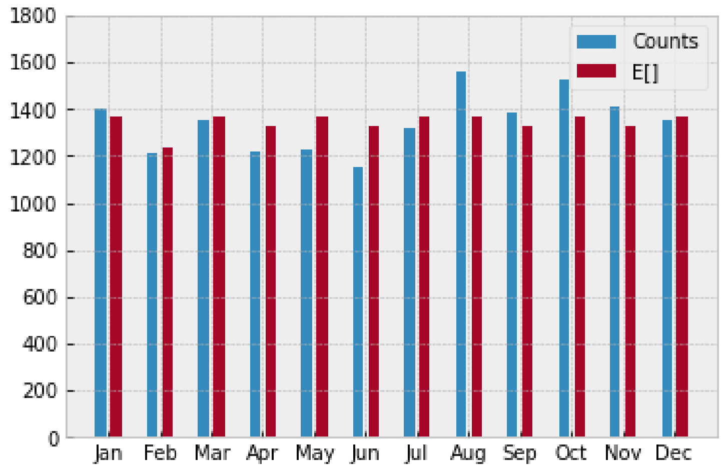
In [14]:
```
days_in_month=np.array([31,28,31,30,31,30,31,31,30,31,30,31])
```

In [16]:
```
# Expected value (assuming a uniform distribution)

ref = days_in_month/365 * total_players

ref
```

Out[16]:
```
array([1367.99178082, 1235.60547945, 1367.99178082, 1323.8630137 ,
       1367.99178082, 1323.8630137 , 1367.99178082, 1367.99178082,
       1323.8630137 , 1367.99178082, 1323.8630137 , 1367.99178082])
```

Now we can compare the data to the expected values:

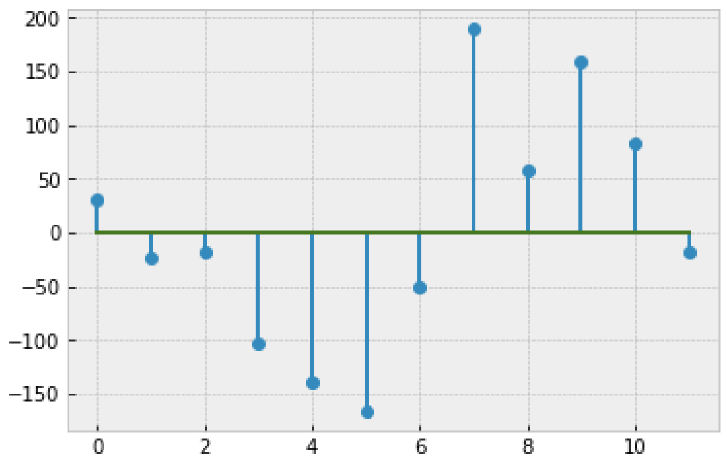
In [17]:
```
plt.bar(np.arange(12)-0.15, baseball, width=0.2, label='Counts')
plt.bar(np.arange(12)+0.15, ref, width=0.2, label='E[]')
plt.legend()
plt.ylim(0,1800)
plt.xticks(np.arange(12), months);
```



Observations:

- August through November seem to be overrepresented, but how can we test it, with 12 different values?
- Let's start by computing the errors:

In [19]:
```
# Errors = observed- reference

errors = baseball - ref

# Stem plot of errors
plt.stem(errors);
```



- Now we need to turn the errors into a single test statistic
- Note that the errors are both positive and negative
- We solve this in the same way we have before, let's start by looking at the total squared error (also called the total deviation):

```
In [20]: E2_sum = sum(errors**2)
         E2_sum
```

```
Out[20]: 133597.9654269094
```

- Then we can carry out our statistical test in the usual way
- We draw examples from the distribution under H_0 and then see how often we get such a large total deviation:

```
In [21]: # RV under the H0 - assuming data is Uniformly-distributed

         bballRV = stats.rv_discrete(values=(np.arange(12), days_in_month/365))

         # This RV models the number of baseball players born in each month of the year
```

```
In [29]: players = bballRV.rvs()
         print(players)
```

```
5
```

```
In [31]: players = bballRV.rvs(size=20)
         print(players)

         vals, counts = np.unique(players,return_counts=True)
         print(vals, counts)
```

```
[ 5  9  7  6  1  9 10  5  6  5  8  2  9 11  2 11  8  5  2  7]
[ 1  2  5  6  7  8  9 10 11] [1 3 4 2 2 2 3 1 2]
```

```
In [32]: ref = days_in_month/365 * total_players

         num_sims=1000
         count=0
         for sim in range(num_sims):
             players = bballRV.rvs(size=total_players) # sample size should be the same
             vals, counts = np.unique(players,return_counts=True)

             test_errors = counts - ref # computes elementwise errors
             test_E2sum = sum(test_errors**2) # sum of the squared errors
             if test_E2sum >= E2_sum:
                 count+=1

         print('Prob of seeing a result this extreme is', count/num_sims)
```

```
Prob of seeing a result this extreme is 0.0
```

Conclusion: The result is statistically significant. MLB players' birthdays are not uniformly distributed throughout the year.

- Note that some months have more days than others. Those months will naturally have more variation than months with more days because the expected counts will be smaller

- To compensate for this effect, it is instead common to normalize the cell deviations by dividing by the expected value of that cell:

```
In [33]: # squared errors divided by the expected values

         Cvals = errors**2/ref

         Cvals
```

```
Out[33]: array([ 0.65825923,  0.41356866,  0.23662728,  7.9923674 , 14.12195264,
                21.03183264,  1.90071443, 26.39133061,  2.55306564, 18.2505463 ,
                 5.09605937,  0.23662728])
```

After normalizing, we can again calculate a statistic that is a sum of the normalized deviations:

```
In [34]: Chisq = sum(Cvals)

         Chisq
```

```
Out[34]: 98.88295146884933
```

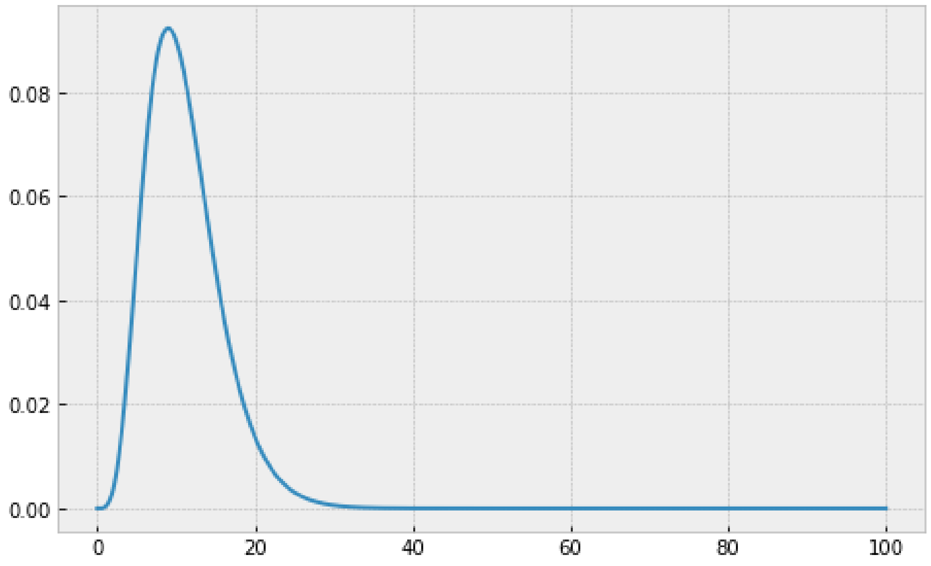
For reasons we will discuss later, this is called the **chi-squared statistic** with $N - 1$ degrees of freedom (dof), i.e.

$$\sum_{i=1}^{N} \frac{(O_i - E_i)^2}{E_i} \sim \chi^2(\text{dof} = N - 1)$$

where O_i is the observed value, E_i is the expected value and N is the total number of random values.

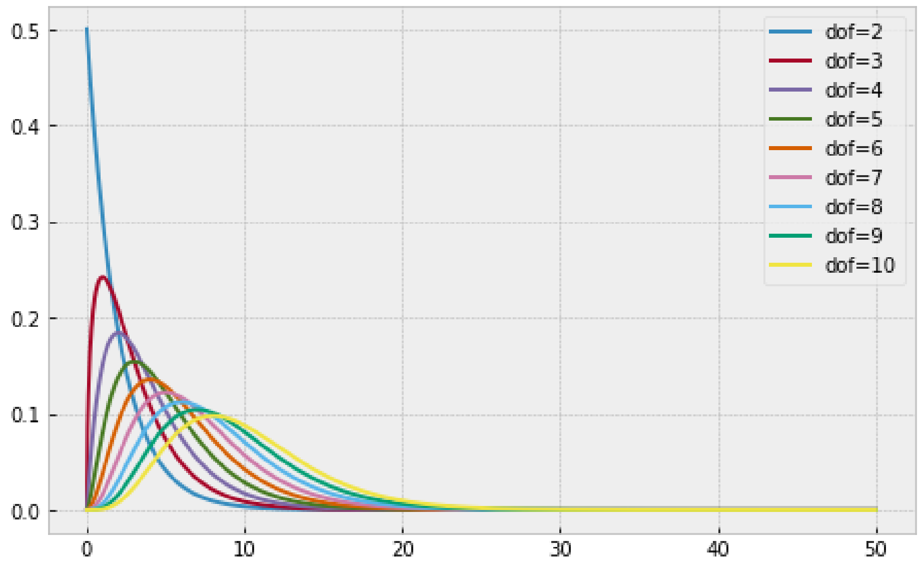
In [38]:
```
x = np.linspace(0,100,1000)

plt.figure(figsize=(8,5))
Chi2_RV = stats.chi2(12-1)
plt.plot(x, Chi2_RV.pdf(x))
print('p-value = ', Chi2_RV.sf(Chisq))
```

```
p-value =  2.9710916764507454e-16
```



```
In [37]: x = np.linspace(0,50,1000)
         plt.figure(figsize=(8,5))
         for dof in range(2,11):
             C = stats.chi2(dof)
             plt.plot(x, C.pdf(x), label='dof='+str(dof))
             print('Chi-square with dof = ',dof,', Moments: ', C.stats('mvsk'))
         plt.legend();
```

```
Chi-square with dof =  2 , Moments:  (array(2.), array(4.), array(2.), array
(6.))
Chi-square with dof =  3 , Moments:  (array(3.), array(6.), array(1.6329931
6), array(4.))
Chi-square with dof =  4 , Moments:  (array(4.), array(8.), array(1.4142135
6), array(3.))
Chi-square with dof =  5 , Moments:  (array(5.), array(10.), array(1.2649110
6), array(2.4))
Chi-square with dof =  6 , Moments:  (array(6.), array(12.), array(1.1547005
4), array(2.))
Chi-square with dof =  7 , Moments:  (array(7.), array(14.), array(1.0690449
7), array(1.71428571))
Chi-square with dof =  8 , Moments:  (array(8.), array(16.), array(1.), arra
y(1.5))
Chi-square with dof =  9 , Moments:  (array(9.), array(18.), array(0.9428090
4), array(1.33333333))
Chi-square with dof =  10 , Moments:  (array(10.), array(20.), array(0.89442
719), array(1.2))
```



```
In [ ]:
```

We can carry out a similar simulation test as above.

In [40]:
```
num_sims=1000
count=0
for sim in range(num_sims):
    players = bballRV.rvs(size=total_players)
    vals, counts = np.unique(players,return_counts=True)
    test_E2= (counts-ref)**2
    test_chi2 = sum(test_E2/ref)
    if test_chi2 >= Chisq:
        count+=1

print('Prob of seeing a result this extreme is',count/num_sims)
```

```
Prob of seeing a result this extreme is 0.0
```

Conclusion: The result is statistically significant. MLB players' birthdays are not uniformly distributed throughout the year.

Example 3: try at home

Instead, let's try another baseball example:

(From *Mathematical Statistics with Resampling and R* By Laura M. Chihara, Tim C. Hesterberg)

The file "homeruns.pickle" contains the homerun data for the Philadelphia Phillies in 2009.

Each entry is the number of games with the corresponding index number of homeruns:

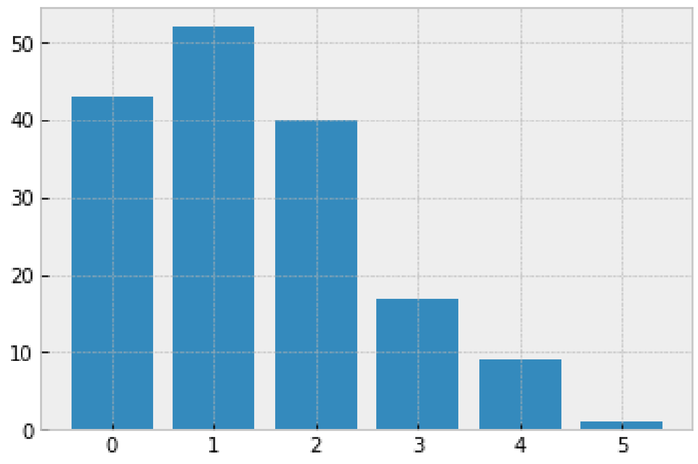
In [84]:
```
pf = open('homeruns.pickle','rb')
```

In [85]:
```
homeruns = pickle.load(pf)

pf.close()
```

In [86]:
```
homeruns
```

Out[86]:
```
array([43, 52, 40, 17,  9,  1])
```

```
In [88]: plt.bar(range(len(homeruns)),homeruns);
```



- What distribution might this come from??
  - Poisson?
- What do we need to specify that distribution?
  - Need the average number of HRs/game (i.e., the mean of the distribution)

```
In [89]: # Total number of games
         num_games = homeruns.sum()

         num_games
Out[89]: 162
```

```
In [90]: homeruns
Out[90]: array([43, 52, 40, 17,  9,  1])
```

```
In [92]: total_hr = 0*43 + 1*52 + 2*40 + 3*17 + 4*9 + 5*1

         total_hr
Out[92]: 224
```

```
In [94]: alpha = total_hr/num_games

         alpha
```

```
Out[94]: 1.382716049382716
```

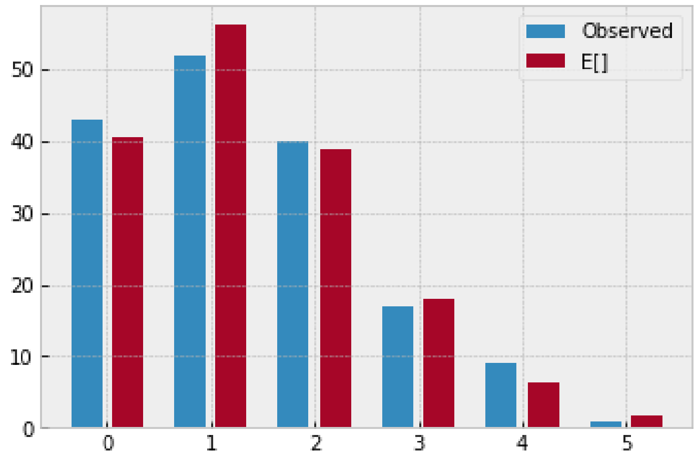
```
In [95]: poiss = stats.poisson(alpha)
```

```
In [99]: # Expected value for the number of HRs using this RV

         Ehr = poiss.pmf(range(0,6))*num_games

         Ehr
```

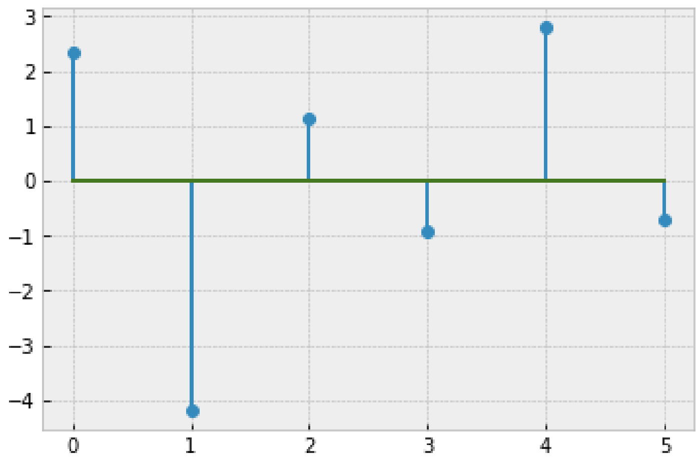
```
Out[99]: array([40.64518122, 56.20074441, 38.85483564, 17.90840161,  6.19055858,
                 1.71195694])
```

It looks like a good fit. BUT we should test it:

```
In [102]: plt.bar(np.arange(len(homeruns))-0.2, homeruns, width = 0.3, label = 'Observed
          plt.bar(np.arange(len(homeruns))+0.2, Ehr, width = 0.3, label = 'E[]')
          plt.legend();
```



In [105]:
```python
errors = homeruns - Ehr

plt.stem(errors);
```



In [107]:
```python
C2 = sum(errors**2/Ehr)

C2

# Chi squared statistic with dof = num_games-1 = 5
```

Out[107]:
```
2.101328348553322
```

We think that this probably comes from this Poisson distribution. If so, the simulation should produce a p-value >> 0.055

In [108]:
```python
chi = stats.chi2(5)

chi.sf(C2)
```

Out[108]:
```
0.8349541601119141
```

In [109]:
```python
num_sims=1000
count=0
for sim in range(num_sims):
    simulated_hrs = poiss.rvs(size=num_games)
    vals, counts = np.unique(simulated_hrs, return_counts = True)
    Ehr = poiss.pmf(vals)*num_games
    test_C2 = sum((counts-Ehr)**2/Ehr)
    if test_C2 >= C2:
        count+=1

print('Prob of seeing a result this extreme is',count/num_sims)
```

```
Prob of seeing a result this extreme is 0.843
```

Conclusion:

It is likely that this data matches a Poisson distribution.

Testing Whether Data Comes from a Distribution: Continuous Distributions

Example 4

Consider the data in "lightbulb.pickle".

```
In [41]: file = open('lightbulb.pickle', 'rb')
```

```
In [42]: lb = pickle.load(file)

         file.close()
```

How should we determine what distribution this data comes from?

- Let's look at what sorts of values we got:

```
In [43]: print(lb[:50])
```

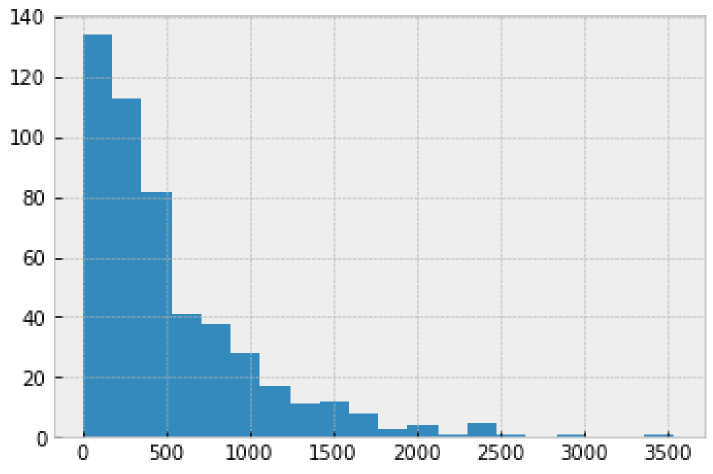
```
[ 887.7415503   208.72771492  389.00714683  183.18045766  920.22754213
 2461.6626774   607.78389789  414.15694351  263.98779539  176.9330967
  299.93308142  443.2713806   573.42681808   46.23002197   74.8567485
   47.26357526  895.60807601  294.04566125  388.85985157  129.06611905
  448.9831565   191.58687295  388.62848586  819.18336506  382.7912885
 1453.46975      39.43299978  199.0261991   261.29036441  378.52594954
  720.35231601   18.90299026  960.07827717 1714.80328959   36.80597999
  240.34430536 1315.28561533  214.39605876  144.81915044  706.99767947
  192.29612863  206.80354989  177.36177955 1450.0942653   279.46545748
  447.63800719  471.88379668   71.77093065  357.1771056   743.68617784]
```

The values seem to be coming from throughout the positive real line -- this data is from a continous distribution.

Now we need to try to determine which continuous distribution is a good fit for the data.

1. Start by plotting a histogram of the data. Adjust the number of bins to provide an appropriate amount of resolution to help infer what distribution this might be from.

In [45]:
```
plt.hist(lb, bins=20);
```



Clearly the data is not Gaussian/Normal or Uniform.

- Of the distributions we have considered, this seems to match the exponential random variable.

Assuming that this is from an exponential distribution, we can compare the histogram of the data with that from the theoretical model. The exponential distribution is characterized by a single parameter, either λ or $\mu = 1/\lambda$, which is the mean.

2. Let's estimate the mean of the reference distribution. We know that the sample mean estimator is an unbiased estimate of the mean:

In [46]:
```
# Estimate for the sample mean

mu_hat = lb.mean()

mu_hat
```

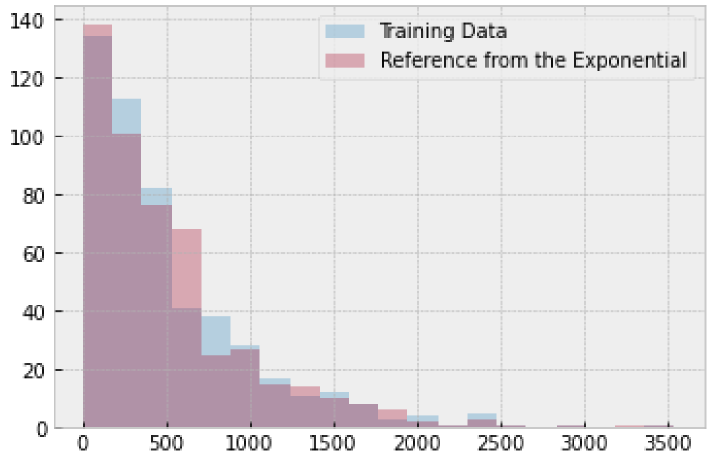
Out[46]:
```
522.7296432789975
```

3. Now create an exponential random variable object and draw data from this reference distribution:

In [47]:
```
E = stats.expon(scale = mu_hat)
```

In [48]:
```
# randomly sampling values from this exponential RV

ref = E.rvs(size=len(lb))
```

In [57]:
```

_, mybins, _ = plt.hist(lb, bins=20, alpha=0.3, label='Training Data')
plt.hist(ref, bins = mybins, alpha=0.3, label='Reference from the Exponential'
plt.legend();
```



In [55]:
```
```

The match is not perfect, but they are similar.

- Since this data comes from a continuous distribution, kernel density estimation (KDE) may be better than a histogram!

Quantile-Quantile (Q-Q) Plot

Let's investigate other ways that we can visually compare these data. We will first generate a **quantile-quantile (Q-Q) plot** for the data.

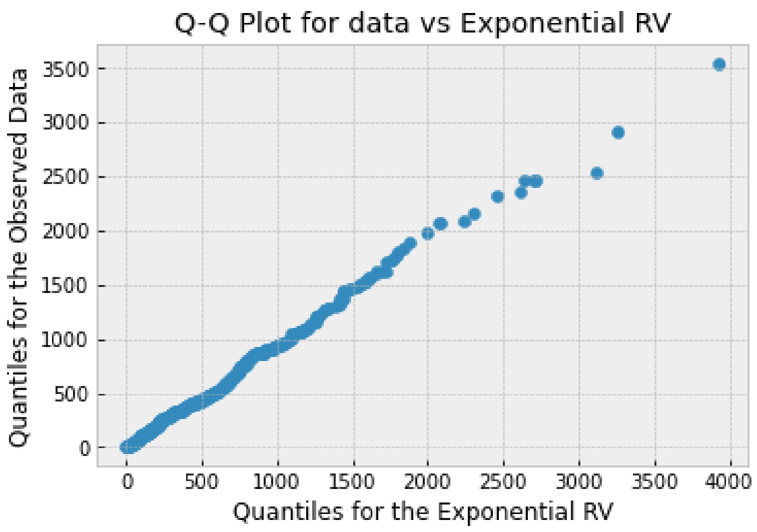
- The kth **quantile** from a data set of length n is the data point that is k/nth of the way through the ordered set.
- In a Q-Q plot, we plot the data at a particular quantile in one data set vs the data at a particular quantile in another data set.

Read the wikipedia page on Q-Q plots: https://en.wikipedia.org/wiki/Q–Q_plot (https://en.wikipedia.org/wiki/Q%E2%80%93Q_plot)

We will only consider the easiest case, which is when the data sets are of the same size. In that case, we can just plot the sorted values with respect to each other:

In [58]:
```python
# random sample from the exponential

ref = E.rvs(size=len(lb))
```

In [59]:
```python
# Plot against each other
plt.scatter(np.sort(ref),np.sort(lb))
plt.title('Q-Q Plot for data vs Exponential RV')
plt.xlabel('Quantiles for the Exponential RV')
plt.ylabel('Quantiles for the Observed Data');
```

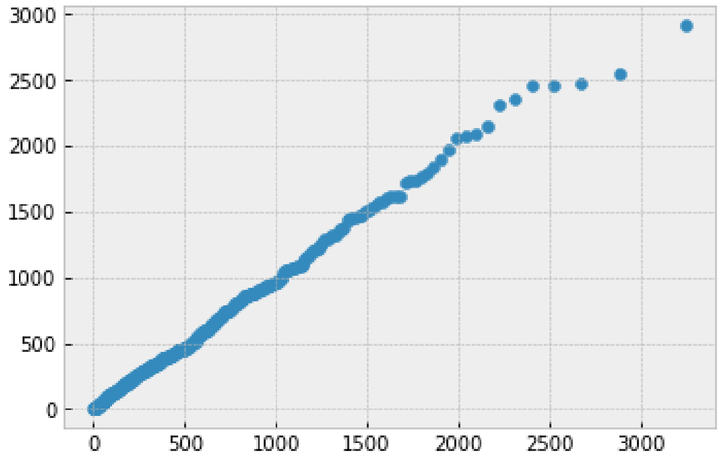


If the data is from the same distribution, the plot should be approximately linear!

When we are comparing data to a known distribution, we can get the exact quantiles from the distribution of the random variable, rather than using samples from the random variable.

Sometimes this type of plot is called a **probability plot**. More generally, the term probability plot is sometimes used to refer to a broader class of plots including the Q-Q plot. **We can get the quantiles from a distribution using the inverse CDF (in `scipy.stats` , this is called the percent point function (ppf):**

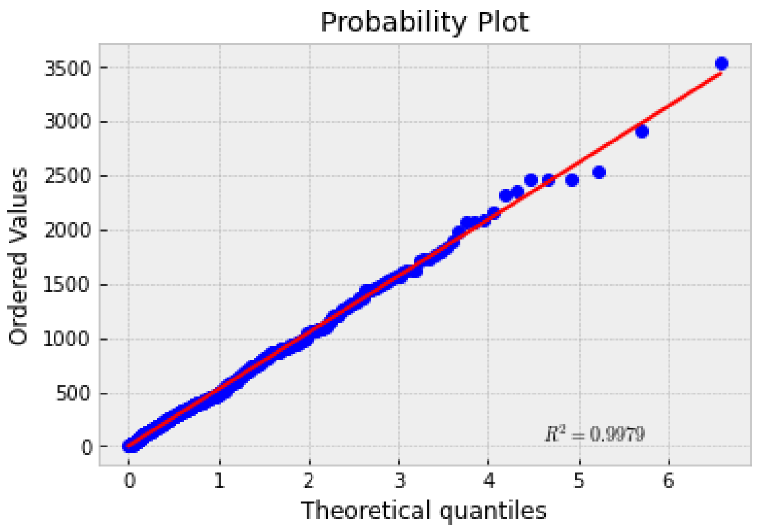
```
In [62]: cumulative_prob = np.linspace(0,1,len(lb))

         ## Scatter plot
         plt.scatter(E.ppf(cumulative_prob), np.sort(lb));
```



We observe that the function is even more linear. It has a little less variation because we have eliminated one of the sources of randomness in the Q-Q plot.

Finally, we leverage the `scipy.stats` `probplot` method to generate the same plot directly:

```
In [64]: stats.probplot(lb, dist='expon', plot=plt, rvalue=True);
```



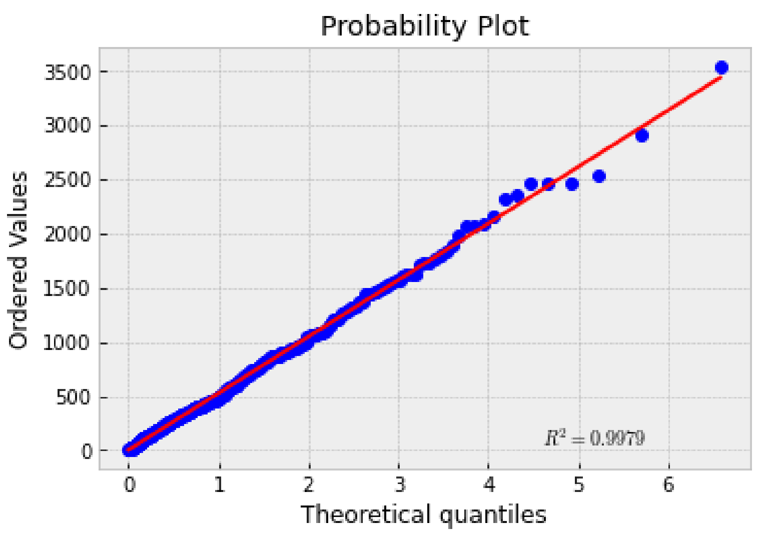
Read the docstring for the `stats.probplot` method:

```
In [65]: ?stats.probplot
```

Note that we can only use `stats.probplot` for distributions that `scipy.stats` knows (but that is A LOT).

After you have read the docstring and understand the outputs, let's store those and look at them:

In [66]:
```
quantiles,regress_info = stats.probplot(lb, dist='expon', plot=plt, rvalue=Tru
```



We are not yet ready to talk in detail about linear regression, but basically it is finding the best line to fit a set of data (when the error is mean-squared error).

The regression parameters are (from the docstring): (slope, intercept, r)

You all should be familiar with the slope and intercept of a line. The parameter `r` (usually written r in text) measures how close the data fits the line. We will work with r^2 instead. The closer r^2 is to 1, then the better the line fits the quantiles (and the better our reference distribution fits the data). We will consider the reference distribution to be a good match for the data if $r^2 \geq 0.9$.

Let's check how well the exponential distribution fits our data:

In [67]:
```
regress_info
```

Out[67]:
```
(522.2937228908745, 1.9203056142599735, 0.9989686323478805)
```

In [68]:
```
r = regress_info[2]

print('R^2 or Coefficient of Determination: ', r**2)
```

```
R^2 or Coefficient of Determination:  0.9979383284149947
```

Conclusion:

Since $r^2 \approx 0.9979 > 0.9$, the exponential distribution is an excellent fit to this data set.

What is a Contingency Table?

Contingency Table

A **contingency table**, sometimes called *cross-tabulation* or *two-way table*, is a type of table in a matrix format that displays (multivariate) categorical data in terms of frequency counts.

More precisely, an $r \times c$ contingency table shows the observed frequency of two variables, the observed frequencies of which are arranged into r rows and c columns. The intersection of a row and a column of a contingency table is called a cell.

- Contingency tables are great to summarize (large) data sets
- Contingency tables are used for organizing categorical variables and testing hypothesis with the chi-squared test for independence

For example, the contingency table below has two rows and five columns (not counting header rows/columns) and shows the results of a random sample of 2200 adults classified by two variables, namely gender and favorite way to eat ice cream.

	cup	**cone**	**sundae**	**sandwich**	**other**
male	592	300	204	24	80
female	410	335	180	20	55

One benefit of having data presented in a contingency table is that it allows one to more easily perform basic probability calculations, a feat made easier still by augmenting a summary row and column to the table.

	cup	**cone**	**sundae**	**sandwich**	**other**	**total**
male	592	300	204	24	80	1200
female	410	335	180	20	55	1000
total	1002	635	384	44	135	2200

The above table is an extended version of the first table obtained by adding a summary row and column. These summaries allow easier computation of several different probability-related quantities.

Marginal Total

The numbers in every cell are called **marginal totals**. The grand total (the total number of individuals represented in the contingency table) is the number in the bottom right corner.

The table allows users to see at a glance that the proportion of men who like to eat their ice cream in a cone is about the same as the proportion of women who like to eat their ice cream in a cone although the proportions are not identical.

Conditional Probability

If the proportions of individuals in the different columns vary significantly between rows (or vice versa), it is said that there is a *contingency* between the two variables. In other words, the two variables are **not independent**. If there is no contingency, it is said that the two variables are independent.

Expected Frequency

One useful value to know is the **expected frequency** $E_{r,c}$ of the cell at the intersection of column c and row r, the formula for which is given by

$$E_{c,r} = \frac{(\text{sum of row } r) \times (\text{sum of column } c)}{\text{sample size}}$$

Example 5

From the contingency table below, compute:

	cup	cone	sundae	sandwich	other	total
male	592	300	204	24	80	1200
female	410	335	180	20	55	1000
total	1002	635	384	44	135	2200

1. Probability that a random participant prefers their ice cream in a cup?

$$\frac{1002}{2200}$$

2. Probability that a random participant prefers their ice cream in a sandwich?

$$\frac{44}{2200}$$

3. Probability that a random participant is female?

$$\frac{1000}{2200}$$

4. Probability that a person prefers ice cream sandwiches given that the person is male?

$$P(\text{sandwich}|\text{male}) = \frac{P(\text{sandwich} \cap \text{male})}{P(\text{male})} = \frac{24}{1200}$$

5. Probability that a person is male given that ice cream sandwiches are preferred?

$$P(\text{male}|\text{sandwich}) = \frac{24}{44}$$

6. Expected value of men who prefer to eat ice cream from a cup?

$$\frac{1200 \times 1002}{2200} \approx 546.54$$

7. Expected value of women who prefer to eat ice cream from a sundae?

$$\frac{1000 \times 384}{2200} \approx 174.54$$

8. What are the variables of study?

Gender and preferred way to eat ice cream.

9. How many degrees of freedom does this contingency table have?

$$(2-1)\times(5-1)=4$$

Chi-Squared Test & Independence

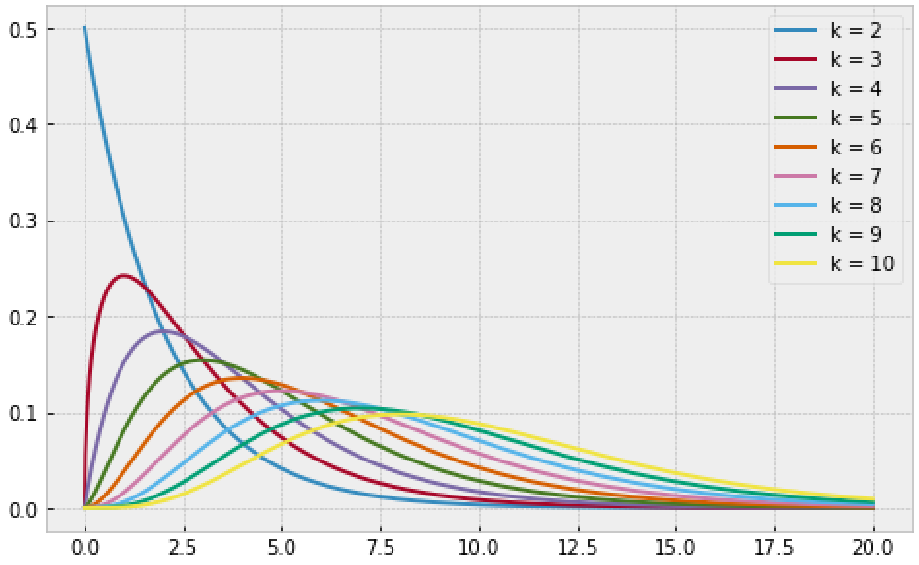
One of the major benefits of computing expected frequencies is the ability to test whether the two variables are actually *independent*. This is done by computing, for each cell (c,r), the expected frequency $E_{c,r}$, comparing it to the observed frequency $O_{c,r}$, and then performing the **chi-squared test**.

$$\chi^2=\sum_{\text{all cells}}\frac{(O_{c,r}-E_{c,r})^2}{E_{c,r}}=\sum_c\sum_r\frac{(O_{c,r}-E_{c,r})^2}{E_{c,r}}$$

This statistic is called a χ^2 as it follows a χ^2 distribution with k degrees of freedom (https://en.wikipedia.org/wiki/Chi-squared_distribution).

- The degrees of freedom can be computed as $(\#\text{ rows}-1)\times(\#\text{ columns}-1)$
- In the example above, there are 4 degrees of freedom

```
In [69]: x = np.linspace(0,20,1000)

         plt.figure(figsize=(8,5))
         for i in range(2,11):
             C = stats.chi2(i)
             plt.plot(x,C.pdf(x),label='k = '+str(i))

         plt.legend();
```



Chi-Squared (χ^2) RV and Gaussian RV

If $Y_1, \ldots, Y_k$ independent identically distributed (i.i.d.), standard normal random variables, that is, $Y_i \sim G(0, 1)$, then

$$Z = \sum_{i=1}^{k} Y_i^2$$

Z is distributed as $\chi^2$ with k degrees of freedom, $Z \sim \chi^2(k)$.

In other words, the chi-square distribution with k degrees of freedom is the distribution of a sum of the squares of k independent Normal random variables.

```
In [ ]: N = 10_000
        x = np.linspace(0,20,N)

        dof = 5
        C = stats.chi2(dof)
        G = stats.norm()

        sum_kG = 0
        for i in range(dof):
            #
            #

        plt.plot(x,C.pdf(x),label='$\chi^2$(dof='+str(dof)+')')
        plt.plot(x, stats.gaussian_kde(sum_kG)(x), label='KDE estimate for $\chi^2=(dc
        plt.hist(sum_kG,density='True', label='Sum of '+str(dof)+' Normal dist.')
        plt.legend();
```

Example 6

Carry the independence test for the two variables in the following contingency tables:

	cup	cone	sundae	sandwich	other	total
male	592	300	204	24	80	1200
female	410	335	180	20	55	1000
total	1002	635	384	44	135	2200

```
In [70]: # H0: the two variables are independent
         # H1: the two variables are not independent

         # Statistic: chi-squared statistic
```

```
In [73]: observed = np.array([[592, 300, 204, 24, 80],
                              [410, 335, 180, 20, 55]])

         observed
```

```
Out[73]: array([[592, 300, 204,  24,  80],
                [410, 335, 180,  20,  55]])
```

```
In [75]: E = stats.contingency.expected_freq(observed)
         print(E)
```

```
[[546.54545455 346.36363636 209.45454545  24.          73.63636364]
 [455.45454545 288.63636364 174.54545455  20.          61.36363636]]
```

```
In [76]: print(observed-E)
```

```
[[ 45.45454545 -46.36363636  -5.45454545   0.           6.36363636]
 [-45.45454545  46.36363636   5.45454545   0.          -6.36363636]]
```

Observations:

- The columns and rows sum to 0
- Cannot use the sum of the error (defined as the different between observed and expected) as it always sums to 0. Instead let's consider the squared of the error

```
In [77]: print((observed-E)**2)
```

```
[[2066.11570248 2149.58677686   29.75206612    0.          40.49586777]
 [2066.11570248 2149.58677686   29.75206612    0.          40.49586777]]
```

Observations:

- Some cells have larger expected values and more observations than others
- A difference of 10 in a cell of expected value of 5 is more significant than a difference of 10 in a cell of expected value of 100
- We can take into account the expected cell size

```
In [78]: print((observed-E)**2/E)
```

```
[[3.78031815 6.20615605 0.14204545 0.         0.54994388]
 [4.53638178 7.44738726 0.17045455 0.         0.65993266]]
```

```
In [79]: C = np.sum((observed-E)**2/E)
         C

Out[79]: 23.4926197837629
```

Given this statistic, how can we determine if this result is statistically significant with $\alpha = 0.01$? We need to find the p-value under the null Hypothesis

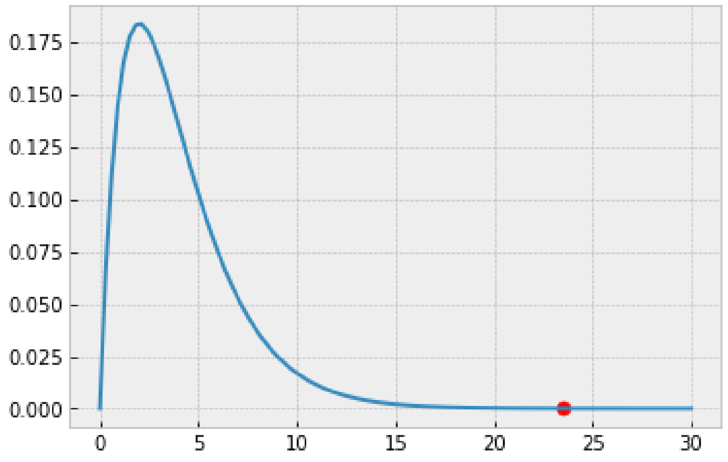
$$H_0 : \text{variables gender and favorite way to eat ice cream are independent}$$

There are two approaches:

1. Resampling (permutation)
2. Analytical Solution (Chi-Squared test): let's focus on this one.

```
In [81]: # Analytical Solution

         x = np.linspace(0,30,100)
         chi = stats.chi2(4)

         plt.plot(x, chi.pdf(x))
         plt.scatter(C, chi.pdf(C), s=50, c='r')

         print(' The probability of observing an error this extreme is equal to ',
              chi.sf(C))
```

```
 The probability of observing a table this extreme is equal to  0.0001009315
4104146287
```



Conclusion:

We reject the null hypothesis. The two variables are **not** independent.

```
In [ ]:
```