**Numpy notes –** NumPy is great at storing and manipulating numerical data in arrays.

Let's take a look at an example. Twice Charred in a fictional (mostly) movie review site where four good friends and movie reviewers, Lorie, Marty, Tori, and Kurtz watch movies and give them ratings on a scale of 0 to 100.

In [1]:

*# Before we do anything, we need to import NumPy*

**import** **numpy** **as** **np**

When the gang rates a movie, we can store their ratings in a NumPy array movie\_ratings:

In [2]:

movie\_ratings = np.array([63.0, 54.0, 70.0, 50.0])

But they see more than one movie, so we have to create a 2-dimensional array where each row is their ratings for a specific movie.

In [3]:

movie\_ratings = np.array([[63.0, 54.0, 70.0, 50.0],

[94.0, 85.0, 89.0, 95.0],

[64.0, 90.0, 73.0, 85.0]])

Some fans prefer to have the movies rated on a five star scale, so we can use NumPy to easily divide each element by 20.

In [4]:

movie\_ratings\_stars = movie\_ratings / 20

Now let's say the ratings are always in the same order (Lorie, Marty, Tori, Kurtz) if we wanted to create an array that only had Tori's ratings, we could select that from our movie\_ratings array.

In [5]:

tori\_ratings = movie\_ratings[:, 2]

tori\_ratings

Out[5]:

array([ 70., 89., 73.])

Now, say we find that we have very similar taste to Marty, so we only want to see movies that he gives a good rating to, we can use logic to select those movies.

Let's select all of Marty's ratings that are over 80:

In [6]:

marty\_ratings = movie\_ratings[:, 1]

marty\_ratings[marty\_ratings > 80]

Out[6]:

array([ 85., 90.])

## Introduction to Statistics with NumPy

After the river in your town flooded during a recent hurricane, you've become interested in collecting data about the its height. Every day for the past month, you walk to the river, measure the height of the water, and enter this information into a notebook.

Let's look at how you can use NumPy functions to analyze your dataset.

First, we'll import the NumPy module, so we can use its statistical calculation functions.

In [10]:

**import** **numpy** **as** **np**

water\_height = np.array([4.01, 4.03, 4.27, 4.29, 4.19,

4.15, 4.16, 4.23, 4.29, 4.19,

4.00, 4.22, 4.25, 4.19, 4.10,

4.14, 4.03, 4.23, 4.08, 14.20,

14.03, 11.20, 8.19, 6.18, 4.04,

4.08, 4.11, 4.23, 3.99, 4.23])

Let's use the function np.mean() to find the average water height:

In [11]:

np.mean(water\_height)

Out[11]:

5.2510000000000003

But wait! We should sort our data to see if there could be any measurements to throw our data off, or represent a deviation from the mean:

In [12]:

np.sort(water\_height)

Out[12]:

array([ 3.99, 4. , 4.01, 4.03, 4.03, 4.04, 4.08, 4.08,

4.1 , 4.11, 4.14, 4.15, 4.16, 4.19, 4.19, 4.19,

4.22, 4.23, 4.23, 4.23, 4.23, 4.25, 4.27, 4.29,

4.29, 6.18, 8.19, 11.2 , 14.03, 14.2 ])

Looks like that thunderstorm might have impacted the average height! Let's measure the median to see if its more representative of the dataset:

In [13]:

np.median(water\_height)

Out[13]:

4.1900000000000004

While the median tells us where half of our data lies, let's look at a value closer to the end of the dataset. We can use percentiles to use a data points position and get its value:

In [14]:

np.percentile(water\_height, 75)

Out[14]:

4.2649999999999997

So far, we've gotten a good idea about specific values. But what about the spread of our data? Let's calculate the standard deviation to understand how similar or how different each data point is:

In [15]:

np.std(water\_height)

Out[15]:

2.784585367099861

Great! Just using a few simple functions we've been able to quickly calculate several important measurements and can begin analyzing our dataset.

**Statistical Distributions with NumPy**

Imagine that you work as an admissions officer at a university. Part of your job is to collect, analyze, and visualize data that's relevant to interested applicants.

Recently, you've become interested in how histograms can show different distributions of populations and even occurences. You think that histograms would be useful in visualizing different trends, such as changes in department numbers and participation in extracurriculars. You also want to learn more about how you can use randomly generated distributions to make statistical calculations and predict the probability of future events, such as the sucess of your ultimate frisbee team.

For this lesson, we'll be using NumPy to calculate distributions and Matplotlib to graph our calculations.

In [2]:

**import** **numpy** **as** **np**

**from** **matplotlib** **import** pyplot **as** plt

One set of data you want to analyze is enrollment in different degree programs. By looking at histograms of the number of years students are enrolled in a program, you can identify what programs are becoming more popular, which are falling out of favor, and which have steady, continual enrollment.

First, let's look at how many hundreds of students decide to enroll in Codecademy University and how many years they've been enrolled.

In [3]:

total\_enrollment = [1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5]

plt.hist(total\_enrollment, bins=5, range=(1, 6))

plt.title('Student Enrollment (Codecademy University)')

plt.xlabel('Years Enrolled')

plt.ylabel('Students Enrolled (Hundreds)')

plt.show()

The histogram above shows the University's total enrollment, which is fairly consistent. This is a *uniform distribution* and is what the University wants to see. Total enrollment is staying at a good level.

Now let's take a look at the enrollment specifically for students seeking a degree in History:

In [4]:

history\_enrollment = [1, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5]

plt.hist(history\_enrollment, bins=5, range=(1, 6))

plt.title('Student Enrollment (History Department)')

plt.xlabel('Years Enrolled')

plt.ylabel('Students Enrolled (Tens)')

plt.show()

What does this histogram tell us? Well this is somewhat *skewed left* dataset, we can see that there are a lot more students who have been enrolled for 3 or 4 years over 1 and 2 years. This indicates that the History program is becoming less popular. Where are all the students going then?

The school recently invested a lot of money in a new building for the Computer Science Department. Let's take a look at enrollment and see if the investment is paying off.

In [5]:

cs\_enrollment = [1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 4, 4]

plt.hist(cs\_enrollment, bins=5, range=(1, 6))

plt.title('Student Enrollment (Computer Science Department)')

plt.xlabel('Years Enrolled')

plt.ylabel('Students Enrolled (Tens)')

plt.show()

It looks like enrollment has skyrocketed for the Computer Science department in recent years. This could be because the University invested in the department, or it could be a sign that the sought after job skills in the real world are changing. Whatever the reason, the histograms let us clearly see the trends.

Interested applicants would like to know what kinds of SAT scores accepted students had. You've previously calculated that the mean score is 1250, with a standard deviation of 50.

Rather than gather every students score, you take what you know about the data and use a random number generator to generate a model.

In [6]:

sat\_scores = np.random.normal(1250, 50, size=100000)

plt.hist(sat\_scores, bins=1000, range=(800,1600))

plt.title('Admitted Student SAT Scores')

plt.xlabel('SAT Score')

plt.ylabel('Students')

plt.show()

95% of Students score within two standard deviations of the mean. An interested student scores an 1130 and wants to know if they are within that range.

In [7]:

mean = 1250

one\_std = 50

two\_below = (mean - 2\*one\_std)

**print** two\_below

1150

Looks like they're just below it! Better re-take that test.

One of the big draws to your school is your excellent ultimate frisbee team. The team wins about 70% of their 50 games each season, or 35 games. An interested applicant wants to know what the chance is that they could improve their record to 40 games. You use what you know about binomial distributions to calculate the probability of such an occurence:

In [8]:

ultimate = np.random.binomial(50, 0.70, size=10000)

plt.hist(ultimate, range=(0, 50), bins=50, normed=True)

plt.xlabel('Number of Games')

plt.ylabel('Frequency')

plt.show()

Since it's a little hard to see from the graph, let's calculate exactly what chance they have of winning 40 games:

In [10]:

ultimate = np.random.binomial(50, 0.70, size=10000)

np.mean(ultimate == 40)

Out[10]:

0.041000000000000002

Hmm, looks like it might be tough for the team to reach that number of wins, given the current data - but even more of a reason for this applicant to sign up and help the team improve!