Statistics Fundamentals

Statistics is primarily about analyzing data samples, and that starts with understanding the distribution of data in a sample.

Analyzing Data Distribution

A great deal of statistical analysis is based on the way that data values are distributed within the dataset. In this section, we'll explore some statistics that you can use to tell you about the values in a dataset.

Measures of Central Tendency

The term *measures of central tendency* sounds a bit grand, but really it's just a fancy way of saying that we're interested in knowing where the middle value in our data is. For example, suppose decide to conduct a study into the comparative salaries of people who graduated from the same school. You might record the results like this:

Name	Salary
Dan	50,000
Joann	54,000
Pedro	50,000
Rosie	189,000
Ethan	55,000
Vicky	40,000
Frederic	59,000

Now, some of the former-students may earn a lot, and others may earn less; but what's the salary in the middle of the range of all salaries?

Mean

A common way to define the central value is to use the *mean*, often called the *average*. This is calculated as the sum of the values in the dataset, divided by the number of observations in the dataset. When the dataset consists of the full population, the mean is represented by the Greek symbol μ (*mu*), and the formula is written like this:

$$\mu = \frac{\sum i = 1^N Xi}{N}$$

More commonly, when working with a sample, the mean is represented by $\bar{x}$ (*x-bar*), and the formula is written like this (note the lower case letters used to indicate values from a sample):

$$\bar{x} = \frac{\sum i = 1^n xi}{n}$$

In the case of our list of heights, this can be calculated as:

$$\bar{x} = \frac{50000 + 54000 + 50000 + 189000 + 55000 + 40000 + 59000}{7}$$

Which is **71,000**.

> In technical terminology, $\bar{x}$ is a *statistic* (an estimate based on a sample of data) and μ is a *parameter* (a true value based on the entire population). A lot of the time, the parameters for the full population will be impossible (or at the very least, impractical) to measure; so we use statistics obtained from a representative sample to approximate them. In this case, we can use the sample mean of salary for our selection of surveyed students to try to estimate the actual average salary of all students who graduate from our school.

In Python, when working with data in a *pandas.dataframe*, you can use the **mean** function, like this:

```
import pandas as pd


df = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan', 'Vicky', 'Frederic'],
                   'Salary':[50000,54000,50000,189000,55000,40000,59000]})

print (df['Salary'].mean())
```

```
71000.0
```

So, is **71,000** really the central value? Or put another way, would it be reasonable for a graduate of this school to expect to earn $71,000? After all, that's the average salary of a graduate from this school.

If you look closely at the salaries, you can see that out of the seven former students, six earn less than the mean salary. The data is *skewed* by the fact that Rosie has clearly managed to find a much higher-paid job than her classmates.

Median

OK, let's see if we can find another definition for the central value that more closely reflects the expected earning potential of students attending our school. Another measure of central tendancy we can use is the *median*. To calculate the median, we need to sort the values into ascending order and then find the middle-most value. When there are an odd number of observations, you can find the position of the median value using this formula (where *n* is the number of observations):

$$\frac{n+1}{2}$$

Remember that this formula returns the *position* of the median value in the sorted list; not the value itself.

If the number of observations is even, then things are a little (but not much) more complicated. In this case you calculate the median as the average of the two middle-most values, which are found like this:

$$\frac{n}{2} \text{ and } \frac{n}{2} + 1$$

So, for our graduate salaries; first lets sort the dataset:

Salary
40,000
50,000
50,000
54,000
55,000
59,000
189,000

There's an odd number of observation (7), so the median value is at position (7 + 1) ÷ 2; in other words, position 4:

Salary
40,000
50,000
50,000
> 54,000
55,000
59,000
189,000

So the median salary is **54,000**.

The *pandas.dataframe* class in Python has a ***median*** function to find the median:

```
1  import pandas as pd
2
3
4  df = pd.DataFrame({'Name': ['Dan','Joann','Pedro','Rosie','Ethan',
   'Vicky','Frederic'],
5                    'Salary':[50000,54000,50000,189000,55000,40000,59000]})
6
7  print (df['Salary'].median())
```

```
1   54000.0
```

Mode

Another related statistic is the *mode*, which indicates the most frequently occurring value. If you think about it, this is potentially a good indicator of how much a student might expect to earn when they graduate from the school; out of all the salaries that are being earned by former students, the mode is earned by more than any other.

Looking at our list of salaries, there are two instances of former students earning **50,000**, but only one instance each for all other salaries:

Salary
40,000
> 50,000
> 50,000
54,000
55,000
59,000
189,000

The mode is therefore **50,000**.

As you might expect, the *pandas.dataframe* class has a ***mode*** function to return the mode:

```
1   import pandas as pd
2
3
4   df = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan', 'Vicky',
    'Frederic'],
5                     'Salary':[50000,54000,50000,189000,55000,40000,59000]})
6
7   print (df['Salary'].mode())
```

```
1   0    50000
2   dtype: int64
```

Multimodal Data

It's not uncommon for a set of data to have more than one value as the mode. For example, suppose Ethan receives a raise that takes his salary to **59,000**:

Salary
40,000
> 50,000
> 50,000
54,000
> 59,000
> 59,000
189,000

Now there are two values with the highest frequency. This dataset is *bimodal*. More generally, when there is more than one mode value, the data is considered *multimodal*.

The *pandas.dataframe.**mode*** function returns all of the modes:

```
1  import pandas as pd
2
3
4  df = pd.DataFrame({'Name':
   ['Dan','Joann','Pedro','Rosie','Ethan','Vicky','Frederic'],
5                'Salary':[50000,54000,50000,189000,59000,40000,59000]})
6
7  print (df['Salary'].mode())
```

```
1  0    50000
2  1    59000
3  dtype: int64
```

Distribution and Density

Now we know something about finding the center, we can start to explore how the data is distributed around it. What we're interested in here is understanding the general "shape" of the data distribution so that we can begin to get a feel for what a 'typical' value might be expected to be.

We can start by finding the extremes - the minimum and maximum. In the case of our salary data, the lowest paid graduate from our school is Vicky, with a salary of **40,000**; and the highest-paid graduate is Rosie, with **189,000**.

The *pandas.dataframe* class has ***min*** and ***max*** functions to return these values.

Run the following code to compare the minimum and maximum salaries to the central measures we calculated previously:

```
1   import pandas as pd
2
3   df = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan', 'Vicky',
    'Frederic'],
4                     'Salary':[50000,54000,50000,189000,55000,40000,59000]})
5
6   print ('Min: ' + str(df['Salary'].min()))
7   print ('Mode: ' + str(df['Salary'].mode()[0]))
8   print ('Median: ' + str(df['Salary'].median()))
9   print ('Mean: ' + str(df['Salary'].mean()))
10  print ('Max: ' + str(df['Salary'].max()))
```

```
1   Min: 40000
2   Mode: 50000
3   Median: 54000.0
4   Mean: 71000.0
5   Max: 189000
```

We can examine these values, and get a sense for how the data is distributed - for example, we can see that the *mean* is closer to the max than the *median*, and that both are closer to the *min* than to the *max*.

However, it's generally easier to get a sense of the distribution by visualizing the data. Let's start by creating a histogram of the salaries, highlighting the *mean* and *median* salaries (the *min*, *max* are fairly self-evident, and the *mode* is wherever the highest bar is):

```
1   %matplotlib inline
2   import pandas as pd
3   import matplotlib.pyplot as plt
4
5   df = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan', 'Vicky',
    'Frederic'],
6                     'Salary':[50000,54000,50000,189000,55000,40000,59000]})
7
8   salary = df['Salary']
9   salary.plot.hist(title='Salary Distribution', color='lightblue', bins=25)
10  plt.axvline(salary.mean(), color='magenta', linestyle='dashed', linewidth=2)
11  plt.axvline(salary.median(), color='green', linestyle='dashed', linewidth=2)
12  plt.show()
```

Salary Distribution
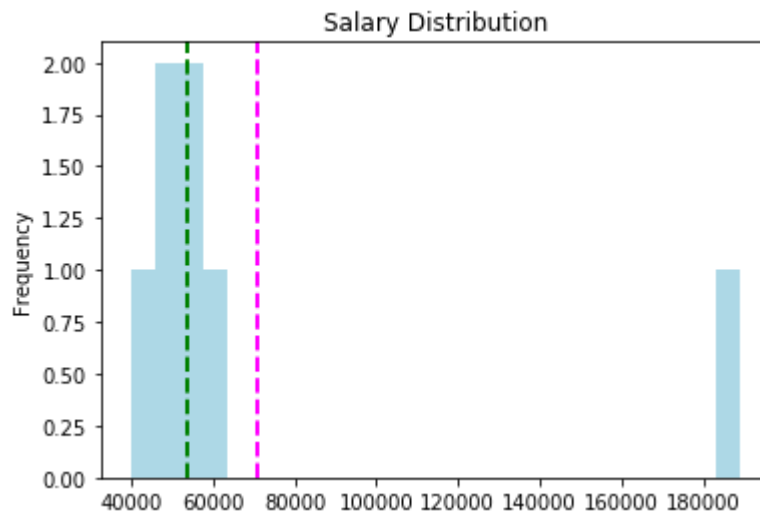
The **mean** and **median** are shown as dashed lines. Note the following:

- *Salary* is a continuous data value - graduates could potentially earn any value along the scale, even down to a fraction of cent.
- The number of bins in the histogram determines the size of each salary band for which we're counting frequencies. Fewer bins means merging more individual salaries together to be counted as a group.
- The majority of the data is on the left side of the histogram, reflecting the fact that most graduates earn between 40,000 and 55,000
- The mean is a higher value than the median and mode.
- There are gaps in the histogram for salary bands that nobody earns.

The histogram shows the relative frequency of each salary band, based on the number of bins. It also gives us a sense of the *density* of the data for each point on the salary scale. With enough data points, and small enough bins, we could view this density as a line that shows the shape of the data distribution.

Run the following cell to show the density of the salary data as a line on top of the histogram:

```
1   %matplotlib inline
2   import pandas as pd
3   import matplotlib.pyplot as plt
4   import numpy as np
5   import scipy.stats as stats
6
7   df = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan', 'Vicky',
    'Frederic'],
8                      'Salary':[50000,54000,50000,189000,55000,40000,59000]})
9
10  salary = df['Salary']
11  density = stats.gaussian_kde(salary)
12  n, x, _ = plt.hist(salary, histtype='step', normed=True, bins=25)
13  plt.plot(x, density(x)*5)
14  plt.axvline(salary.mean(), color='magenta', linestyle='dashed', linewidth=2)
15  plt.axvline(salary.median(), color='green', linestyle='dashed', linewidth=2)
16  plt.show()
17
```
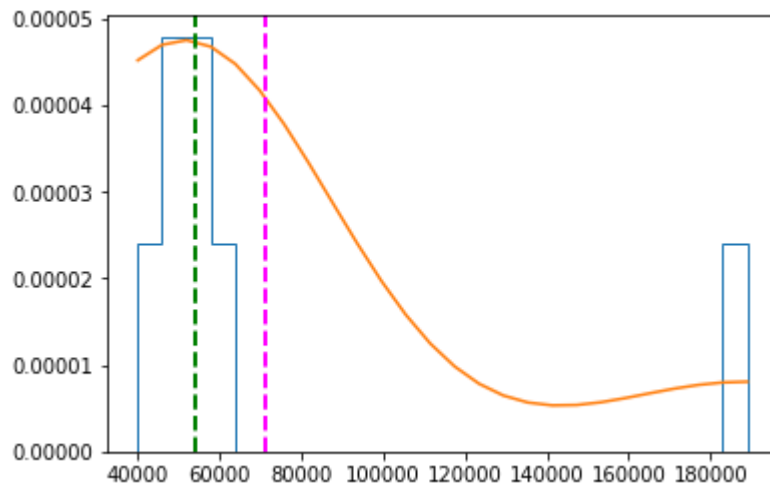
Note that the density line takes the form of an asymmetric curve that has a "peak" on the left and a long tail on the right. We describe this sort of data distribution as being *skewed*; that is, the data is not distributed symmetrically but "bunched together" on one side. In this case, the data is bunched together on the left, creating a long tail on the right; and is described as being *right-skewed* because some infrequently occurring high values are pulling the *mean* to the right.

Let's take a look at another set of data. We know how much money our graduates make, but how many hours per week do they need to work to earn their salaries? Here's the data:

Name	Hours
Dan	41
Joann	40
Pedro	36
Rosie	30
Ethan	35
Vicky	39
Frederic	40

Run the following code to show the distribution of the hours worked:

```
1    %matplotlib inline
2    import pandas as pd
3    import matplotlib.pyplot as plt
4    import numpy as np
5    import scipy.stats as stats
6
7    df = pd.DataFrame({'Name': ['Dan','Joann','Pedro','Rosie',
     'Ethan','Vicky','Frederic'],
8                      'Hours':[41,40,36,30,35,39,40]})
9
10   hours = df['Hours']
11   density = stats.gaussian_kde(hours)
```
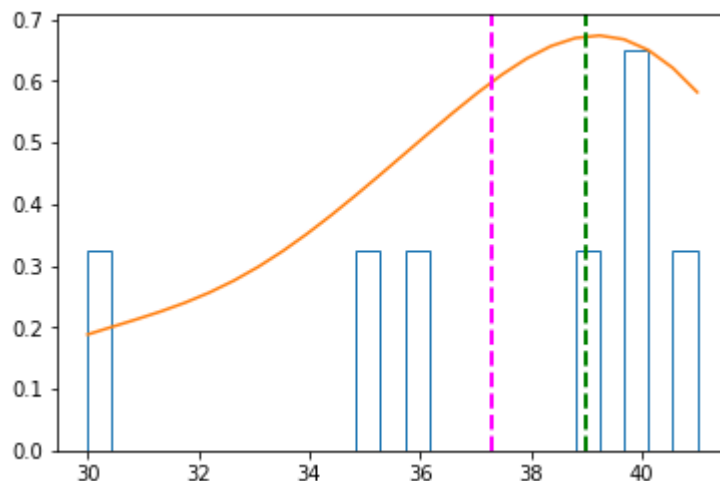
```
12   n, x, _ = plt.hist(hours, histtype='step', normed=True, bins=25)
13   plt.plot(x, density(x)*7)
14   plt.axvline(hours.mean(), color='magenta', linestyle='dashed', linewidth=2)
15   plt.axvline(hours.median(), color='green', linestyle='dashed', linewidth=2)
16   plt.show()
```


Once again, the distribution is skewed, but this time it's **left-skewed**. Note that the curve is asymmetric with the *mean* to the left of the *median* and the *mode*; and the average weekly working hours skewed to the lower end.

Once again, Rosie seems to be getting the better of the deal. She earns more than her former classmates for working fewer hours. Maybe a look at the test scores the students achieved on their final grade at school might help explain her success:

Name	Grade
Dan	50
Joann	50
Pedro	46
Rosie	95
Ethan	50
Vicky	5
Frederic	57

Let's take a look at the distribution of these grades:

```
1   %matplotlib inline
2   import pandas as pd
3   import matplotlib.pyplot as plt
4   import numpy as np
5   import scipy.stats as stats
```

```
  6
  7    df = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan', 'Vicky',
       'Frederic'],
  8                       'Grade':[50,50,46,95,50,5,57]})
  9
 10    grade = df['Grade']
 11    density = stats.gaussian_kde(grade)
 12    n, x, _ = plt.hist(grade, histtype='step', normed=True, bins=25)
 13    plt.plot(x, density(x)*7.5)
 14    plt.axvline(grade.mean(), color='magenta', linestyle='dashed', linewidth=2)
 15    plt.axvline(grade.median(), color='green', linestyle='dashed', linewidth=2)
 16    plt.show()
```
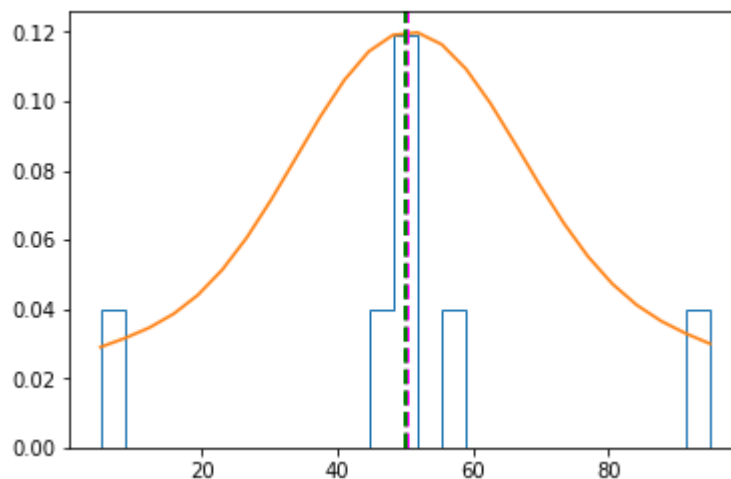

This time, the distribution is symmetric, forming a "bell-shaped" curve. The **_mean_**, **_median_**, and **_mode_** are at the same location, and the data tails off evenly on both sides from a central peak.

Statisticians call this a **normal** distribution (or sometimes a **Gaussian** distribution), and it occurs quite commonly in many scenarios due to something called the *Central Limit Theorem*, which reflects the way continuous probability works - more about that later.

Skewness and Kurtosis

You can measure *skewness* (in which direction the data is skewed and to what degree) and kurtosis (how "peaked" the data is) to get an idea of the shape of the data distribution. In Python, you can use the ***skew*** and ***kurt*** functions to find this:

```
  1    %matplotlib inline
  2    import pandas as pd
  3    import numpy as np
  4    from matplotlib import pyplot as plt
  5    import scipy.stats as stats
  6
  7    df = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan', 'Vicky',
       'Frederic'],
  8                       'Salary':[50000,54000,50000,189000,55000,40000,59000],
  9                       'Hours':[41,40,36,30,35,39,40],
```

```
10                        'Grade':[50,50,46,95,50,5,57]})
11
12  numcols = ['Salary', 'Hours', 'Grade']
13  for col in numcols:
14      print(df[col].name + ' skewness: ' + str(df[col].skew()))
15      print(df[col].name + ' kurtosis: ' + str(df[col].kurt()))
16      density = stats.gaussian_kde(df[col])
17      n, x, _ = plt.hist(df[col], histtype='step', normed=True, bins=25)
18      plt.plot(x, density(x)*6)
19      plt.show()
20      print('\n')
```
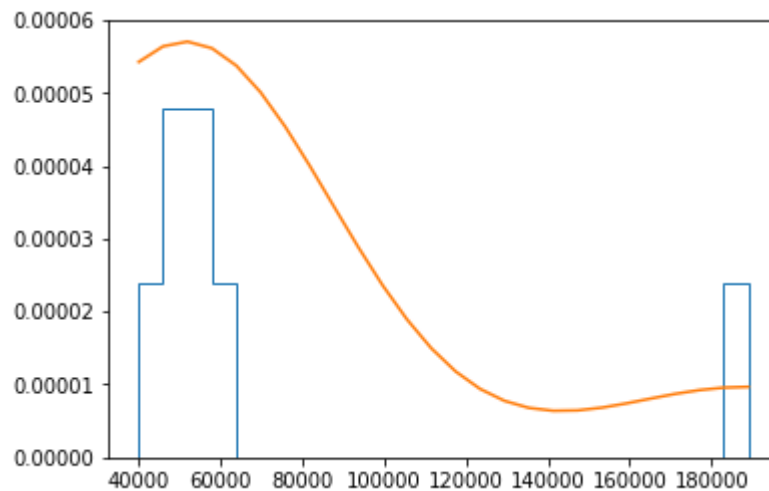
```
1  Salary skewness: 2.57316410755049
2  Salary kurtosis: 6.719828837773431
```
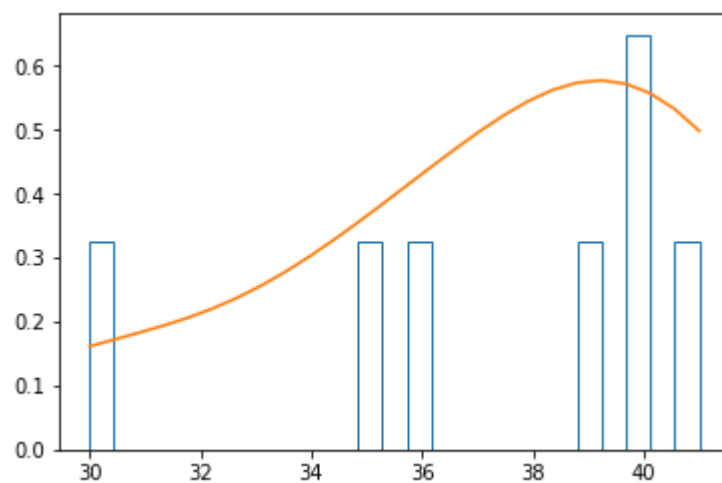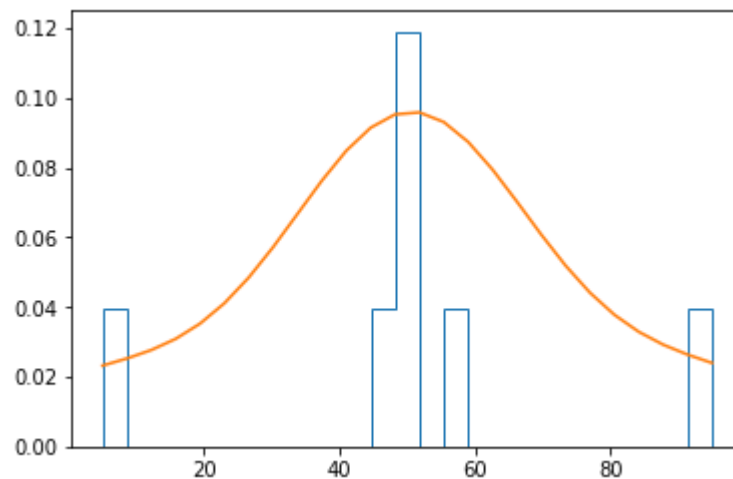


```
1  Hours skewness: -1.194570307262883
2  Hours kurtosis: 0.9412265624999989
```

```
1   Grade skewness: -0.06512433009682762
2   Grade kurtosis: 2.7484764913773034
```


Now let's look at the distribution of a real dataset - let's see how the heights of the father's measured in Galton's study of parent and child heights are distributed:

```
1    %matplotlib inline
2    import pandas as pd
3    import matplotlib.pyplot as plt
4    import numpy as np
5    import scipy.stats as stats
6
7    import statsmodels.api as sm
8
9    df = sm.datasets.get_rdataset('GaltonFamilies', package='HistData').data
10
11
12   fathers = df['father']
13   density = stats.gaussian_kde(fathers)
14   n, x, _ = plt.hist(fathers, histtype='step', normed=True, bins=50)
15   plt.plot(x, density(x)*2.5)
16   plt.axvline(fathers.mean(), color='magenta', linestyle='dashed', linewidth=2)
17   plt.axvline(fathers.median(), color='green', linestyle='dashed', linewidth=2)
18   plt.show()
19
```
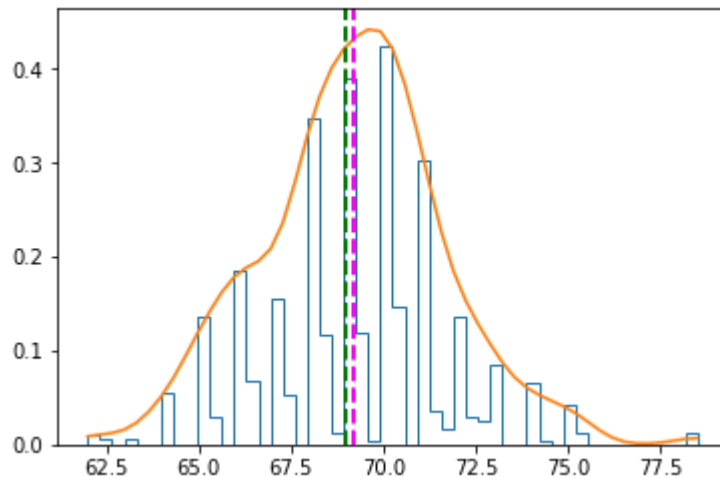
As you can see, the father's height measurements are approximately normally distributed - in other words, they form a more or less *normal* distribution that is symmetric around the mean.

Measures of Variance

We can see from the distribution plots of our data that the values in our dataset can vary quite widely. We can use various measures to quantify this variance.

Range

A simple way to quantify the variance in a dataset is to identify the difference between the lowest and highest values. This is called the *range*, and is calculated by subtracting the minimim value from the maximum value.

The following Python code creates a single Pandas dataframe for our school graduate data, and calculates the *range* for each of the numeric features:

```python
import pandas as pd

df = pd.DataFrame({'Name':
['Dan','Joann','Pedro','Rosie','Ethan','Vicky','Frederic'],
                   'Salary':[50000,54000,50000,189000,55000,40000,59000],
                   'Hours':[41,40,36,30,35,39,40],
                   'Grade':[50,50,46,95,50,5,57]})

numcols = ['Salary', 'Hours', 'Grade']
for col in numcols:
    print(df[col].name + ' range: ' + str(df[col].max() - df[col].min()))
```

```
Salary range: 149000
Hours range: 11
Grade range: 90
```

Percentiles and Quartiles

The range is easy to calculate, but it's not a particularly useful statistic. For example, a range of 149,000 between the lowest and highest salary does not tell us which value within that range a graduate is most likely to earn - it doesn't tell us nothing about how the salaries are distributed around the mean within that range. The range tells us very little about the comparative position of an individual value within the distribution - for example, Frederic scored 57 in his final grade at school; which is a pretty good score (it's more than all but one of his classmates); but this isn't immediately apparent from a score of 57 and range of 90.

Percentiles

A percentile tells us where a given value is ranked in the overall distribution. For example, 25% of the data in a distribution has a value lower than the 25th percentile; 75% of the data has a value lower than the 75th percentile, and so on. Note that half of the data has a value lower than the 50th percentile - so the 50th percentile is also the median!

Let's examine Frederic's grade using this approach. We know he scored 57, but how does he rank compared to his fellow students?

Well, there are seven students in total, and five of them scored less than Frederic; so we can calculate the percentile for Frederic's grade like this:

$$\frac{5}{7} \times 100 \approx 71.4$$

So Frederic's score puts him at the 71.4th percentile in his class.

In Python, you can use the ***percentileofscore*** function in the *scipy.stats* package to calculate the percentile for a given value in a set of values:

```python
import pandas as pd
from scipy import stats

df = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan', 'Vicky', 'Frederic'],
                   'Salary':[50000,54000,50000,189000,55000,40000,59000],
                   'Hours':[41,40,36,30,35,39,40],
                   'Grade':[50,50,46,95,50,5,57]})

print(stats.percentileofscore(df['Grade'], 57, 'strict'))
```

```
71.42857142857143
```

We've used the strict definition of percentile; but sometimes it's calculated as being the percentage of values that are less than *or equal to* the value you're comparing. In this case, the calculation for Frederic's percentile would include his own score:

$$\frac{6}{7} \times 100 \approx 85.7$$

You can calculate this way in Python by using the ***weak*** mode of the ***percentileofscore*** function:

```
1   import pandas as pd
2   from scipy import stats
3
4   df = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan', 'Vicky',
    'Frederic'],
5                        'Salary':[50000,54000,50000,189000,55000,40000,59000],
6                        'Hours':[41,40,36,30,35,39,40],
7                        'Grade':[50,50,46,95,50,5,57]})
8
9   print(stats.percentileofscore(df['Grade'], 57, 'weak'))
```

```
1   85.71428571428571
```

We've considered the percentile of Frederic's grade, and used it to rank him compared to his fellow students. So what about Dan, Joann, and Ethan? How do they compare to the rest of the class? They scored the same grade (50), so in a sense they share a percentile.

To deal with this *grouped* scenario, we can average the percentage rankings for the matching scores. We treat half of the scores matching the one we're ranking as if they are below it, and half as if they are above it. In this case, there were three matching scores of 50, and for each of these we calculate the percentile as if 1 was below and 1 was above. So the calculation for a percentile for Joann based on scores being less than or equal to 50 is:

$$(\frac{4}{7}) \times 100 \approx 57.14$$

The value of **4** consists of the two scores that are below Joann's score of 50, Joann's own score, and half of the scores that are the same as Joann's (of which there are two, so we count one).

In Python, the ***percentileofscore*** function has a ***rank*** function that calculates grouped percentiles like this:

```
1   import pandas as pd
2   from scipy import stats
3
4   df = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan', 'Vicky',
    'Frederic'],
5                        'Salary':[50000,54000,50000,189000,55000,40000,59000],
6                        'Hours':[41,40,36,30,35,39,40],
7                        'Grade':[50,50,46,95,50,5,57]})
8
9   print(stats.percentileofscore(df['Grade'], 50, 'rank'))
```

```
1   57.142857142857146
```

Quartiles

Rather than using individual percentiles to compare data, we can consider the overall spread of the data by dividing those percentiles into four *quartiles*. The first quartile contains the values from the minimum to the 25th percentile, the second from the 25th percentile to the 50th percentile (which is the median), the third from the 50th percentile to the 75th percentile, and the fourth from the 75th percentile to the maximum.

In Python, you can use the **quantile** function of the *pandas.dataframe* class to find the threshold values at the 25th, 50th, and 75th percentiles (*quantile* is a generic term for a ranked position, such as a percentile or quartile).

Run the following code to find the quartile thresholds for the weekly hours worked by our former students:

```
1   # Quartiles
2   import pandas as pd
3
4   df = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan', 'Vicky',
    'Frederic'],
5                       'Salary':[50000,54000,50000,189000,55000,40000,59000],
6                       'Hours':[41,40,36,17,35,39,40],
7                       'Grade':[50,50,46,95,50,5,57]})
8   print(df['Hours'].quantile([0.25, 0.5, 0.75]))
```

```
1   0.25    35.5
2   0.50    39.0
3   0.75    40.0
4   Name: Hours, dtype: float64
```

Its usually easier to understand how data is distributed across the quartiles by visualizing it. You can use a histogram, but many data scientists use a kind of visualization called a *box plot* (or a *box and whiskers* plot).

Let's create a box plot for the weekly hours:

```
1    %matplotlib inline
2    import pandas as pd
3    from matplotlib import pyplot as plt
4
5    df = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan', 'Vicky',
     'Frederic'],
6                        'Salary':[50000,54000,50000,189000,55000,40000,59000],
7                        'Hours':[41,40,36,30,35,39,40],
8                        'Grade':[50,50,46,95,50,5,57]})
9
10   # Plot a box-whisker chart
11   df['Hours'].plot(kind='box', title='Weekly Hours Distribution', figsize=(10,8))
12   plt.show()
```

Weekly Hours Distribution
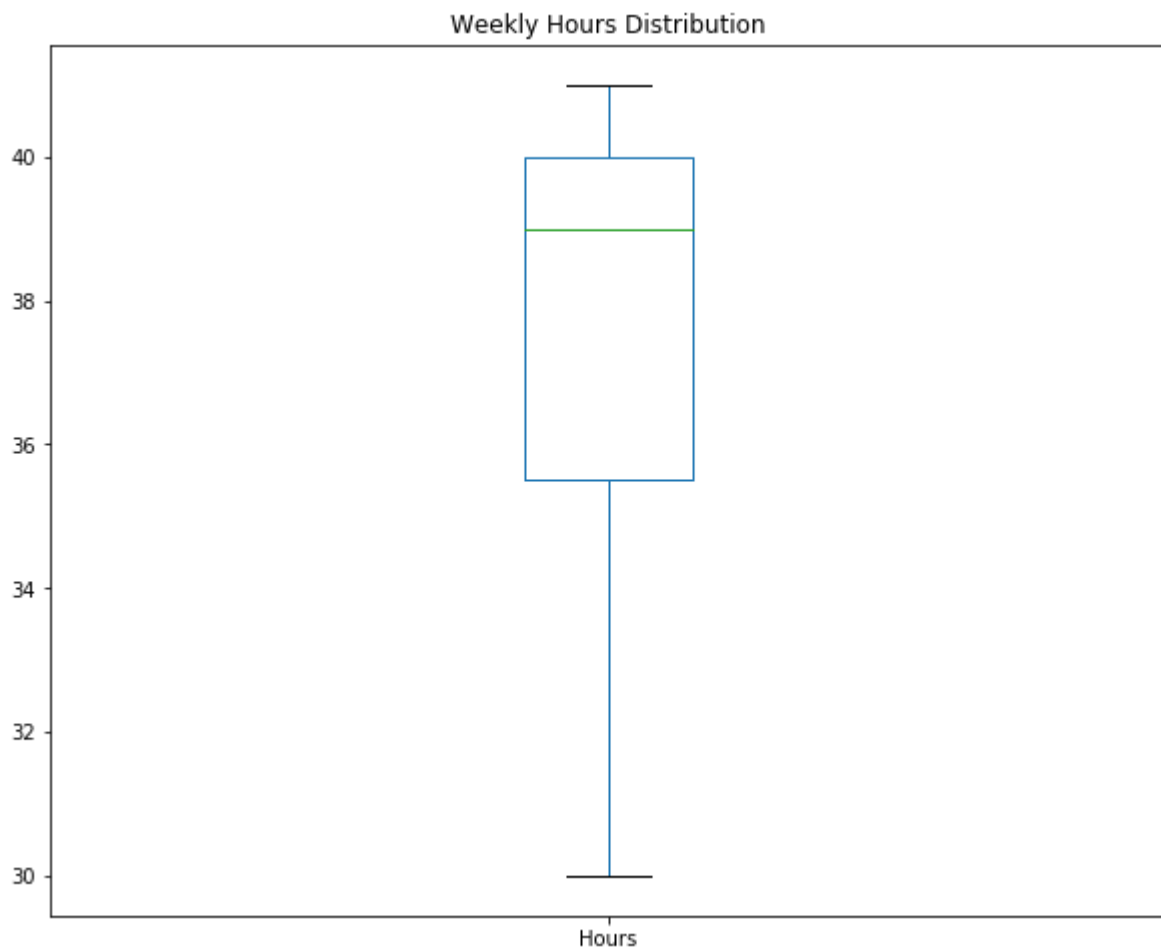
The box plot consists of:

- A rectangular *box* that shows where the data between the 25th and 75th percentile (the second and third quartile) lie. This part of the distribution is often referred to as the *interquartile range* - it contains the middle 50 data values.
- *Whiskers* that extend from the box to the bottom of the first quartile and the top of the fourth quartile to show the full range of the data.
- A line in the box that shows that location of the median (the 50th percentile, which is also the threshold between the second and third quartile)

In this case, you can see that the interquartile range is between 35 and 40, with the median nearer the top of that range. The range of the first quartile is from around 30 to 35, and the fourth quartile is from 40 to 41.
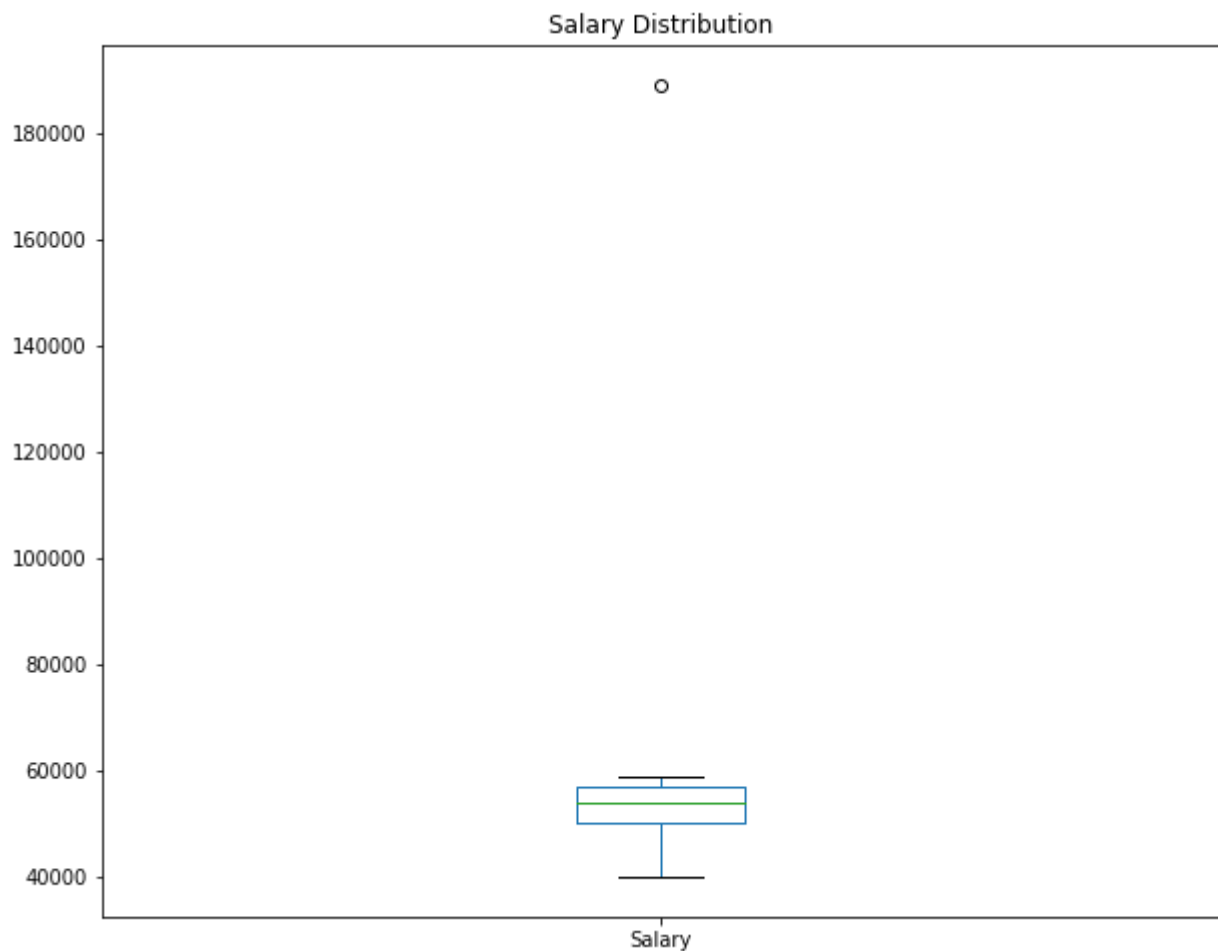
Outliers

Let's take a look at another box plot - this time showing the distribution of the salaries earned by our former classmates:

```
 1   %matplotlib inline
 2   import pandas as pd
 3   from matplotlib import pyplot as plt
 4
 5   df = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan', 'Vicky',
     'Frederic'],
 6                      'Salary':[50000,54000,50000,189000,55000,40000,59000],
 7                      'Hours':[41,40,36,30,35,39,40],
 8                      'Grade':[50,50,46,95,50,5,57]})
 9
10   # Plot a box-whisker chart
11   df['Salary'].plot(kind='box', title='Salary Distribution', figsize=(10,8))
12   plt.show()
```
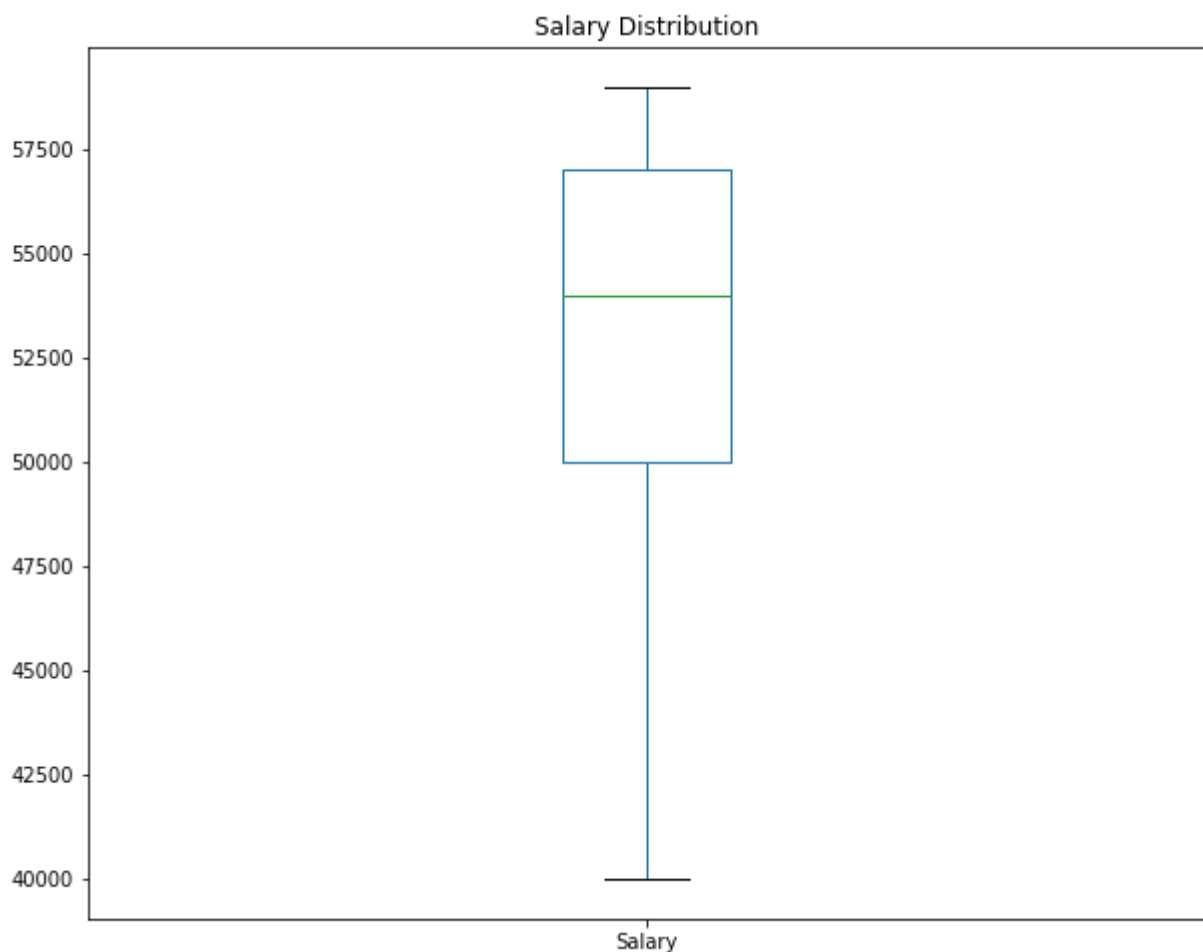

Salary Distribution

So what's going on here?

Well, as we've already noticed, Rosie earns significantly more than her former classmates. So much more in fact, that her salary has been identifed as an *outlier*. An outlier is a value that is so far from the center of the distribution compared to other values that it skews the distribution by affecting the mean. There are all sorts of reasons that you might have outliers in your data, including data entry errors, failures in sensors or data-generating equipment, or genuinely anomalous values.

So what should we do about it?

This really depends on the data, and what you're trying to use it for. In this case, let's assume we're trying to figure out what's a reasonable expectation of salary for a graduate of our school to earn. Ignoring for the moment that we have an extremly small dataset on which to base our judgement, it looks as if Rosie's salary could be either an error (maybe she mis-typed it in the form used to collect data) or a genuine anomaly (maybe she became a professional athelete or some other extremely highly paid job). Either way, it doesn't seem to represent a salary that a typical graduate might earn.

Let's see what the distribution of the data looks like without the outlier:

```
%matplotlib inline
import pandas as pd
from matplotlib import pyplot as plt

df = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan', 'Vicky',
  'Frederic'],
                   'Salary':[50000,54000,50000,189000,55000,40000,59000],
                   'Hours':[41,40,36,17,35,39,40],
                   'Grade':[50,50,46,95,50,5,57]})

# Plot a box-whisker chart
df['Salary'].plot(kind='box', title='Salary Distribution', figsize=(10,8),
  showfliers=False)
plt.show()
```
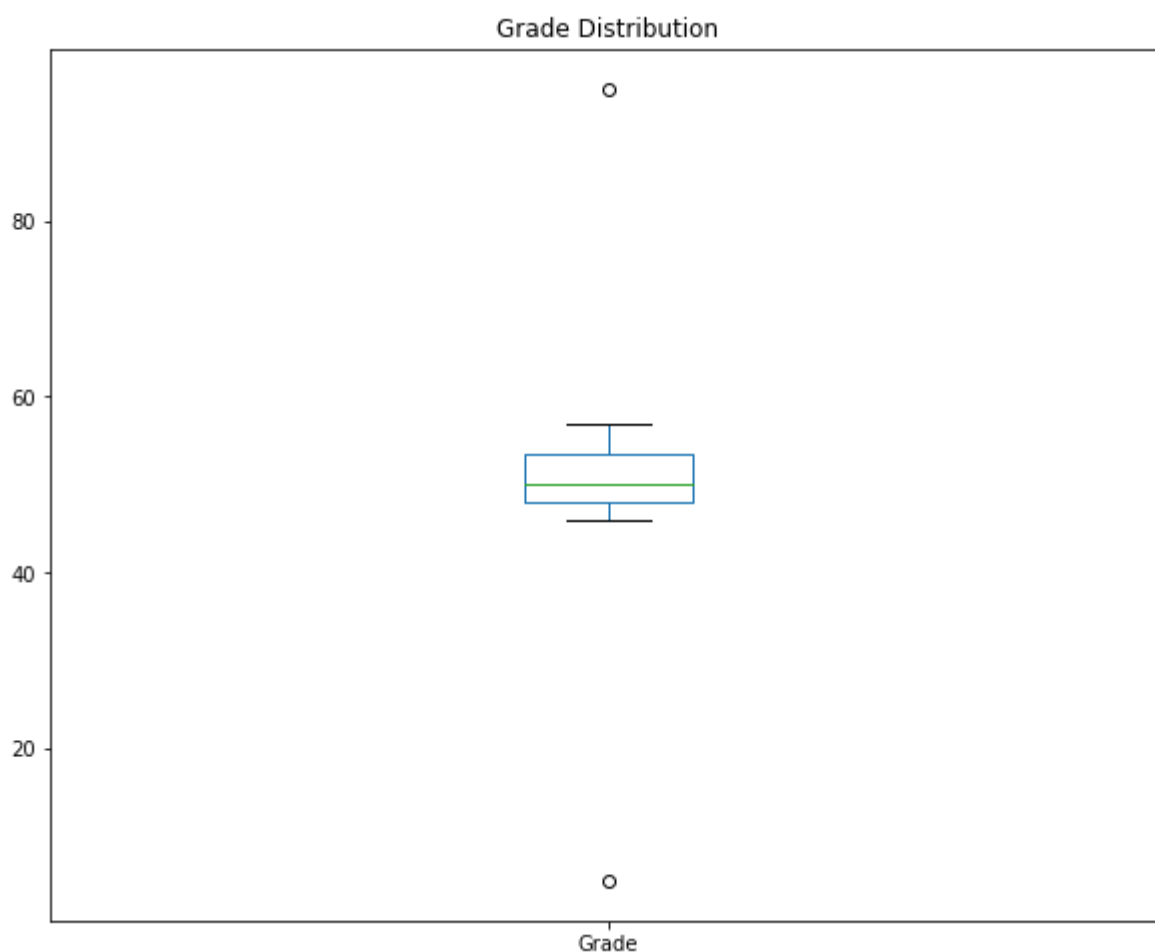

Salary Distribution

Now it looks like there's a more even distribution of salaries. It's still not quite symmetrical, but there's much less overall variance. There's potentially some cause here to disregard Rosie's salary data when we compare the salaries, as it is tending to skew the analysis.

So is that OK? Can we really just ignore a data value we don't like?

Again, it depends on what you're analyzing. Let's take a look at the distribution of final grades:

```python
%matplotlib inline
import pandas as pd
from matplotlib import pyplot as plt

df = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan', 'Vicky',
    'Frederic'],
                   'Salary':[50000,54000,50000,189000,55000,40000,59000],
                   'Hours':[41,40,36,17,35,39,40],
                   'Grade':[50,50,46,95,50,5,57]})

# Plot a box-whisker chart
df['Grade'].plot(kind='box', title='Grade Distribution', figsize=(10,8))
plt.show()
```


Grade Distribution

Once again there are outliers, this time at both ends of the distribution. However, think about what this data represents. If we assume that the grade for the final test is based on a score out of 100, it seems reasonable to expect that some students will score very low (maybe even 0) and some will score very well (maybe even 100); but most will get a score somewhere in the middle. The reason that the low and high scores here look like outliers might just be because we have so few data points. Let's see what happens if we include a few more students in our data:

```python
%matplotlib inline
import pandas as pd
from matplotlib import pyplot as plt

df = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan', 'Vicky',
    'Frederic', 'Jimmie', 'Rhonda', 'Giovanni', 'Francesca', 'Rajab', 'Naiyana',
    'Kian', 'Jenny'],
                    'Grade':[50,50,46,95,50,5,57,42,26,72,78,60,40,17,85]})

# Plot a box-whisker chart
df['Grade'].plot(kind='box', title='Grade Distribution', figsize=(10,8))
plt.show()
```
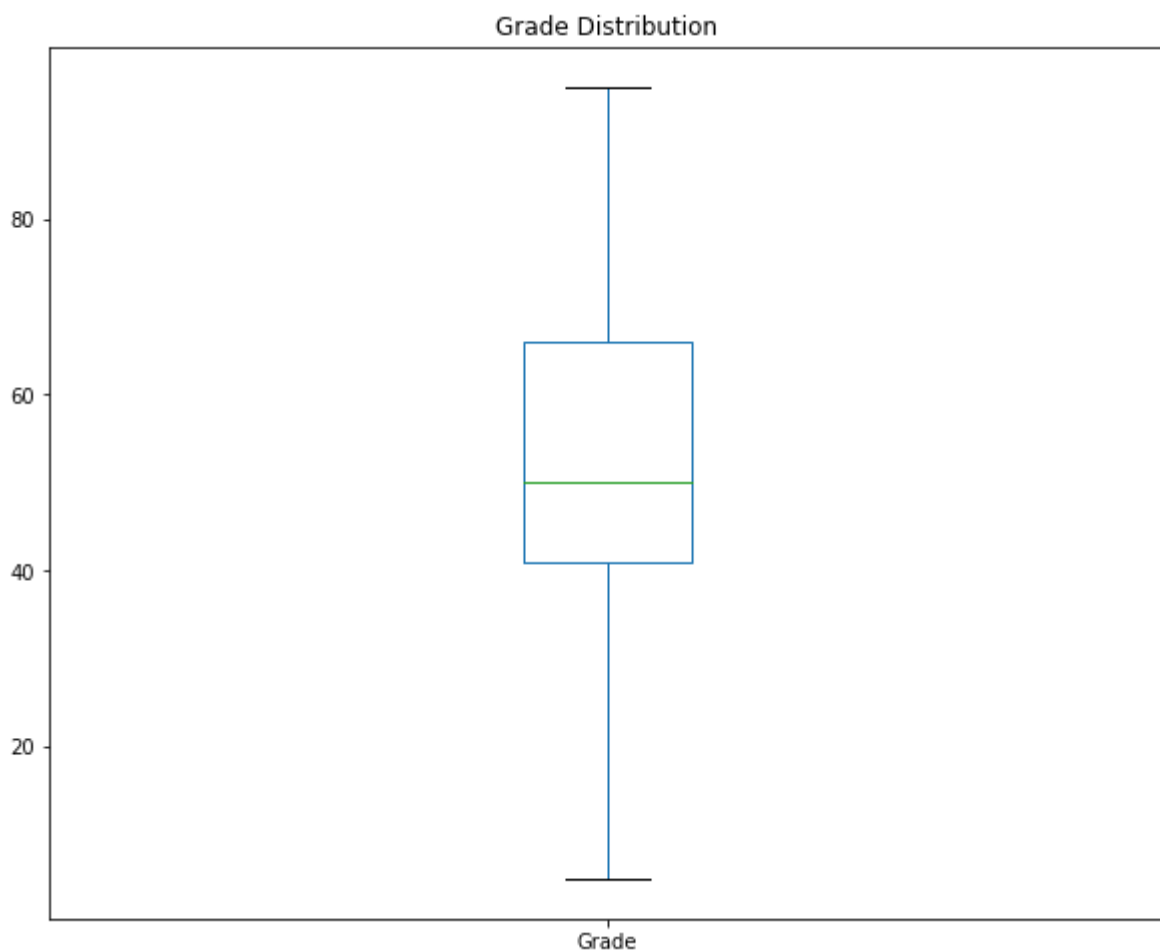

With more data, there are some more high and low scores; so we no longer consider the isolated cases to be outliers.

The key point to take away here is that you need to really understand the data and what you're trying to do with it, and you need to ensure that you have a reasonable sample size, before determining what to do with outlier values.

Variance and Standard Deviation

We've seen how to understand the *spread* of our data distribution using the range, percentiles, and quartiles; and we've seen the effect of outliers on the distribution. Now it's time to look at how to measure the amount of variance in the data.

Variance

Variance is measured as the average of the squared difference from the mean. For a full population, it's indicated by a squared Greek letter *sigma* (**σ²**) and calculated like this:

$$\sigma^2 = \frac{\sum i = 1^N (Xi - \mu)^2}{N}$$

For a sample, it's indicated as **s²** calculated like this:

$$s^2 = \frac{\sum i = 1^n (xi - \bar{x})^2}{n - 1}$$

In both cases, we sum the difference between the individual data values and the mean and square the result. Then, for a full population we just divide by the number of data items to get the average. When using a sample, we divide by the total number of items **minus 1** to correct for sample bias.

Let's work this out for our student grades (assuming our data is a sample from the larger student population).

First, we need to calculate the mean grade:

$$\bar{x} = \frac{50 + 50 + 46 + 95 + 50 + 5 + 57}{7} \approx 50.43$$

Then we can plug that into our formula for the variance:

$$s^2 = \frac{(50 - 50.43)2 + (50 - 50.43)2 + (46 - 50.43)2 + (95 - 50.43)2 + (50 - 50.43)2 + (5 - 50.43)2 + (57 - 50.43)^2}{7 - 1}$$

So:

$$s^2 = \frac{0.185 + 0.185 + 19.625 + 1986.485 + 0.185 + 2063.885 + 43.165}{6}$$

Which simplifies to:

$$s^2 = \frac{4113.715}{6}$$

Giving the result:

$$s^2 \approx 685.619$$

The higher the variance, the more spread your data is around the mean.

In Python, you can use the **var** function of the *pandas.dataframe* class to calculate the variance of a column in a dataframe:

```python
import pandas as pd

df = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan', 'Vicky',
'Frederic'],
                   'Salary':[50000,54000,50000,189000,55000,40000,59000],
                   'Hours':[41,40,36,17,35,39,40],
                   'Grade':[50,50,46,95,50,5,57]})

print(df['Grade'].var())
```

```
685.6190476190476
```

Standard Deviation

To calculate the variance, we squared the difference of each value from the mean. If we hadn't done this, the numerator of our fraction would always end up being zero (because the mean is at the center of our values). However, this means that the variance is not in the same unit of measurement as our data - in our case, since we're calculating the variance for grade points, it's in grade points squared; which is not very helpful.

To get the measure of variance back into the same unit of measurement, we need to find its square root:

$$s = \sqrt{685.619} \approx 26.184$$

So what does this value represent?

It's the *standard deviation* for our grades data. More formally, it's calculated like this for a full population:

$$\sigma = \sqrt{\frac{\sum i = 1^N (Xi - \mu)^2}{N}}$$

Or like this for a sample:

$$s = \sqrt{\frac{\sum i = 1^n (xi - \bar{x})^2}{n - 1}}$$

Note that in both cases, it's just the square root of the corresponding variance forumla!

In Python, you can calculate it using the **std** function:

```
1  import pandas as pd
2
3  df = pd.DataFrame({'Name': ['Dan', 'Joann', 'Pedro', 'Rosie', 'Ethan', 'Vicky',
   'Frederic'],
4                      'Salary':[50000,54000,50000,189000,55000,40000,59000],
5                      'Hours':[41,40,36,17,35,39,40],
6                      'Grade':[50,50,46,95,50,5,57]})
7
8  print(df['Grade'].std())
```

```
1  26.184328282754315
```

Standard Deviation in a Normal Distribution

In statistics and data science, we spend a lot of time considering *normal* distributions; because they occur so frequently. The standard deviation has an important relationship to play in a normal distribution.

Run the following cell to show a histogram of a *standard normal* distribution (which is a distribution with a mean of 0 and a standard deviation of 1):

```
1   %matplotlib inline
2   import pandas as pd
3   import matplotlib.pyplot as plt
4   import numpy as np
5   import scipy.stats as stats
6
7   # Create a random standard normal distribution
8   df = pd.DataFrame(np.random.randn(100000, 1), columns=['Grade'])
9
10  # Plot the distribution as a histogram with a density curve
11  grade = df['Grade']
12  density = stats.gaussian_kde(grade)
13  n, x, _ = plt.hist(grade, color='lightgrey', normed=True, bins=100)
14  plt.plot(x, density(x))
15
16  # Get the mean and standard deviation
17  s = df['Grade'].std()
18  m = df['Grade'].mean()
19
20  # Annotate 1 stdev
21  x1 = [m-s, m+s]
22  y1 = [0.25, 0.25]
23  plt.plot(x1,y1, color='magenta')
24  plt.annotate('1s (68.26%)', (x1[1],y1[1]))
25
26  # Annotate 2 stdevs
```

```
27  x2 = [m-(s*2), m+(s*2)]
28  y2 = [0.05, 0.05]
29  plt.plot(x2,y2, color='green')
30  plt.annotate('2s (95.45%)', (x2[1],y2[1]))
31
32  # Annotate 3 stdevs
33  x3 = [m-(s*3), m+(s*3)]
34  y3 = [0.005, 0.005]
35  plt.plot(x3,y3, color='orange')
36  plt.annotate('3s (99.73%)', (x3[1],y3[1]))
37
38  # Show the location of the mean
39  plt.axvline(grade.mean(), color='grey', linestyle='dashed', linewidth=1)
40
41  plt.show()
```
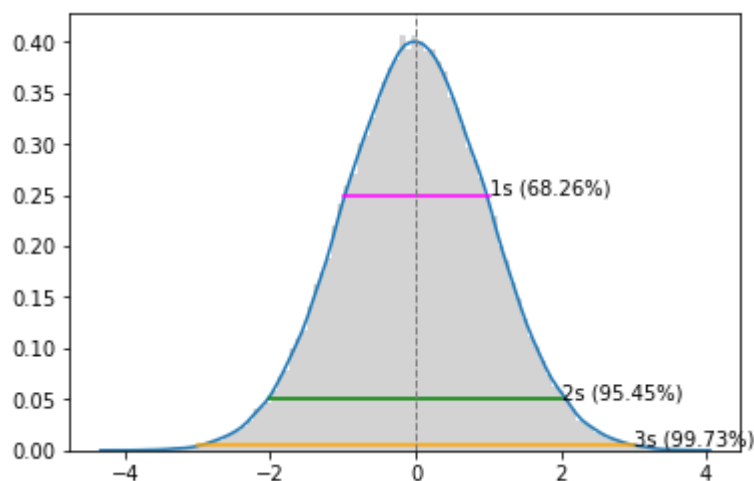

The horizontal colored lines show the percentage of data within 1, 2, and 3 standard deviations of the mean (plus or minus).

In any normal distribution:

- Approximately 68.26% of values fall within one standard deviation from the mean.
- Approximately 95.45% of values fall within two standard deviations from the mean.
- Approximately 99.73% of values fall within three standard deviations from the mean.

Z Score

So in a normal (or close to normal) distribution, standard deviation provides a way to evaluate how far from a mean a given range of values falls, allowing us to compare where a particular value lies within the distribution. For example, suppose Rosie tells you she was the highest scoring student among her friends - that doesn't really help us assess how well she scored. She may have scored only a fraction of a point above the second-highest scoring student. Even if we know she was in the top quartile; if we don't know how the rest of the grades are distributed it's still not clear how well she performed compared to her friends.

However, if she tells you how many standard deviations higher than the mean her score was, this will help you compare her score to that of her classmates.

So how do we know how many standard deviations above or below the mean a particular value is? We call this a *Z Score*, and it's calculated like this for a full population:

$$Z = \frac{x - \mu}{\sigma}$$

or like this for a sample:

$$Z = \frac{x - \bar{x}}{s}$$

So, let's examine Rosie's grade of 95. Now that we know the *mean* grade is 50.43 and the *standard deviation* is 26.184, we can calculate the Z Score for this grade like this:

$$Z = \frac{95 - 50.43}{26.184} = 1.702$$

So Rosie's grade is 1.702 standard deviations above the mean.

Summarizing Data Distribution in Python

We've seen how to obtain individual statistics in Python, but you can also use the ***describe*** function to retrieve summary statistics for all numeric columns in a dataframe. These summary statistics include many of the statistics we've examined so far (though it's worth noting that the *median* is not included):

```python
import pandas as pd

df = pd.DataFrame({'Name':
['Dan','Joann','Pedro','Rosie','Ethan','Vicky','Frederic'],
                   'Salary':[50000,54000,50000,189000,55000,40000,59000],
                   'Hours':[41,40,36,17,35,39,40],
                   'Grade':[50,50,46,95,50,5,57]})
print(df.describe())
```

```
              Salary       Hours       Grade
count       7.000000    7.000000    7.000000
mean    71000.000000   35.428571   50.428571
std     52370.475143    8.423324   26.184328
min     40000.000000   17.000000    5.000000
25%     50000.000000   35.500000   48.000000
50%     54000.000000   39.000000   50.000000
75%     57000.000000   40.000000   53.500000
max    189000.000000   41.000000   95.000000
```