AI Boot Camp

Exploring Data with Pandas

Module 4 Day 2

Class Objectives

By the end of class, you will be able to:

1. Sort values in a DataFrame.
2. View Summary Statistics using Describe.
3. Utilize mean, sum, nunique, value_counts and more where appropriate.
4. Utilize foundational Python concepts to explore data with Pandas.

Activity:
DataFrame Review

In this activity, you will use Pandas to find various pieces of information from a dataset on temperature readings from the LAX airport.

Suggested Time:
10 Minutes

Time’s up!

Let’s review

Instructor **Demonstration**

Sorting Values

Sorting Made Easy

To sort a DataFrame based on the values within a column, use the `df.sort_values()` method and pass in the column name to sort by as a parameter.

The `ascending` parameter is always marked as True by default. Therefore, the `sort_values()` method will always sort from lowest to highest unless the parameter of `ascending=False` is also passed into the `sort_values()` method.

Sorting Values

```
# Sorting the DataFrame based on "Meals" column

# Will sort from lowest to highest if no other parameter is passed

meals_taxes_df = taxes_df.sort_values("Meals")

meals_taxes_df.head()

# To sort from highest to lowest, ascending=False must be passed in

meals_taxes_df = taxes_df.sort_values("Meals", ascending=False)

meals_taxes_df.head()
```

Sorting Values

```
# It is possible to sort based upon multiple columns

meals_and_rent_count_df = taxes_df.sort_values(["Meals Count", "Rent Count"], ascending=False)

meals_and_rent_count_df.head(15)
```

	Town	Meals	Meals Count	Rent	Rent Count	Alcohol	Alcohol Count	Past Meals	Past Meals count	Past Rent	Past Rent Count	Past Alcohol	Past Alchohol Count
17	BURLINGTON	74507552.54	219	18230026.80	26	18324508.20	122	1.276183e+08	236	53634054.09	44	44233463.37	129
81	SOUTH BURLINGTON	64445667.13	111	13750969.61	19	4138460.85	40	8.953598e+07	117	38211751.51	25	10313786.70	44
12	BRATTLEBORO	33966669.55	102	4868408.74	26	2840765.10	41	4.144862e+07	100	9867296.43	27	6096085.57	42
77	RUTLAND	38005509.10	98	1508769.29	14	2973734.52	38	4.199332e+07	98	3822279.43	14	5316214.36	38
32	ESSEX	36429036.93	91	0.00	0	2359611.62	29	4.203358e+07	104	0.00	0	4129281.23	31
7	BENNINGTON	26317917.62	81	3296492.96	23	2225916.88	32	3.214152e+07	94	7243933.44	27	4199857.36	33
87	STOWE	33678629.46	80	40772303.07	96	10993675.86	54	5.218909e+07	84	67794549.41	156	18101140.22	58
55	MANCHESTER	21537627.26	65	13410916.83	41	4124721.26	40	3.084579e+07	69	28037091.09	59	7650316.61	40
59	MONTPELIER	15480173.01	61	0.00	0	1893772.30	28	2.591748e+07	66	3458227.45	17	4959620.16	29
102	WILLISTON	27712613.17	59	0.00	0	2190208.80	20	3.976950e+07	58	0.00	0	4164070.87	20
56	MIDDLEBURY	18797796.05	58	3438513.74	15	1669289.12	27	2.591859e+07	60	7438483.99	13	3943914.16	28
42	HARTFORD	17734685.75	56	6920252.34	27	2250788.12	20	2.623356e+07	51	14940795.47	31	4519017.53	21
23	COLCHESTER	23323491.04	54	5876613.24	22	2041842.99	16	2.775314e+07	58	13882595.77	37	3329310.39	19
86	ST JOHNSBURY	11550000.84	54	0.00	0	614945.83	25	1.342922e+07	55	3748551.33	10	1312342.04	22
4	BARRE	14101058.17	46	0.00	0	1420668.11	19	1.648034e+07	49	0.00	0	2809362.08	21

Sorting Values

```
# DataFrame with a second column sort on "Alcohol Count"

# (Compare the order of the two "54" value Rent Count rows)

meals_and_alcohol_count_df = taxes_df.sort_values(["Meals Count", "Alcohol Count"], ascending=False)

meals_and_alcohol_count_df.head(15)
```

	Town	Meals	Meals Count	Rent	Rent Count	Alcohol	Alcohol Count	Past Meals	Past Meals count	Past Rent	Past Rent Count	Past Alcohol	Past Alchohol Count
17	BURLINGTON	74507552.54	219	18230026.80	26	18324508.20	122	1.276183e+08	236	53634054.09	44	44233463.37	129
81	SOUTH BURLINGTON	64445667.13	111	13750969.61	19	4138460.85	40	8.953598e+07	117	38211751.51	25	10313786.70	44
12	BRATTLEBORO	33966669.55	102	4868408.74	26	2840765.10	41	4.144862e+07	100	9867296.43	27	6096085.57	42
77	RUTLAND	38005509.10	98	1508769.29	14	2973734.52	38	4.199332e+07	98	3822279.43	14	5316214.36	38
32	ESSEX	36429036.93	91	0.00	0	2359611.62	29	4.203358e+07	104	0.00	0	4129281.23	31
7	BENNINGTON	26317917.62	81	3296492.96	23	2225916.88	32	3.214152e+07	94	7243933.44	27	4199857.36	33
87	STOWE	33678629.46	80	40772303.07	96	10993675.86	54	5.218909e+07	84	67794549.41	156	18101140.22	58
55	MANCHESTER	21537627.26	65	13410916.83	41	4124721.26	40	3.084579e+07	69	28037091.09	59	7650316.61	40
59	MONTPELIER	15480173.01	61	0.00	0	1893772.30	28	2.591748e+07	66	3458227.45	17	4959620.16	29
102	WILLISTON	27712613.17	59	0.00	0	2190208.80	20	3.976950e+07	58	0.00	0	4164070.87	20
56	MIDDLEBURY	18797796.05	58	3438513.74	15	1669289.12	27	2.591859e+07	60	7438483.99	13	3943914.16	28
42	HARTFORD	17734685.75	56	6920252.34	27	2250788.12	20	2.623356e+07	51	14940795.47	31	4519017.53	21
86	ST JOHNSBURY	11550000.84	54	0.00	0	614945.83	25	1.342922e+07	55	3748551.33	10	1312342.04	22
23	COLCHESTER	23323491.04	54	5876613.24	22	2041842.99	16	2.775314e+07	58	13882595.77	37	3329310.39	19
4	BARRE	14101058.17	46	0.00	0	1420668.11	19	1.648034e+07	49	0.00	0	2809362.08	21

Sorting Values

```
# The index can be reset to provide index numbers based on the new rankings.

new_index_df = meals_and_alcohol_count_df.reset_index(drop=True)

new_index_df.head()
```

Instructor **Demonstration**

Exploring Data

Exploring data

To delete a column of extraneous information from a DataFrame, use `del <DataFrame>[<Column>]`.

To figure out if any rows are missing data, simply run the `count()` method on the DataFrame and check that all columns contain equal values.

To drop rows with missing information from a DataFrame, use `<DataFrame>.dropna(how="any")`. In the image, count is used to show the total number of non-null values present in each column both before and after using dropna to remove rows with missing values.

```
# Identify incomplete rows
df.count()
```

```
Name          2000
Employer      1820
City          1999
State         1999
Zip           1996
Amount        2000
dtype: int64
```

```
# Drop all rows with missing information
df = df.dropna(how='any')
```

```
# Verify dropped rows
df.count()
```

```
Name          1818
Employer      1818
City          1818
State         1818
Zip           1818
Amount        1818
dtype: int64
```

Exploring data

- Sometimes, the rows containing `NaN` values should not be removed but should instead be filled with another value. In such cases, simply use the `<DataFrame>.fillna(value=<Value>)` method and pass the desired value into the parentheses.
- To find similar or misspelled values, run the `value_counts()` method on the column in question and check the returned values.
- To replace similar or misspelled values, run the `replace()` method on the column in question, and pass in a dictionary with the keys as the values to replace and the values as the replacements, as in the image on the following slide.

Exploring data

```
# Display an overview of the Employers column
df['Employer'].value_counts()
```

```
NOT EMPLOYED            609
NONE                    321
SELF-EMPLOYED           132
SELF                     33
RETIRED                  32
                       ...
INTEL CORPORATION         1
SLOCUM & SONS             1
OCPS                      1
HEALTHCARE PARTNERS       1
CARBON FIVE               1
Name: Employer, Length: 519, dtype: int64
```

```
# Clean up Employer category. Replace 'SELF' and 'SELF EMPLOYED' with 'SELF-EMPLOYED'
df['Employer'] = df['Employer'].replace({'SELF': 'SELF-EMPLOYED', 'SELF EMPLOYED': 'SELF-EMPLOYED'})
```

```
# Verify clean-up.
df['Employer'].value_counts()
```

```
NOT EMPLOYED            609
NONE                    321
SELF-EMPLOYED           180
RETIRED                  32
```

Activity:
Search for the Worst

In this activity, you will take a dataset on San Francisco Airport's utility consumption and answer questions based on the dataset

Suggested Time:
20 Minutes

Time’s up!

Let’s review

Break

15 mins

NumPy: Numerical Python library for working with arrays and matrices.

SciPy: Scientific Python library for working with optimization, interpolation, algebraic equations and statistics.

Instructor **Demonstration**

Central Tendency

What's a measure of **central tendency**?

Measures of central tendency
try to identify the center of a dataset.

Measures of Central Tendency

The three most common measures are the mean, the median, and the mode.

Mean

The mean is the sum of all the values divided by the number of elements in the dataset.

Median

The median is the middle value in a sorted dataset.

Mode

The mode is the value that occurs the most frequently in a dataset.

Measures of Central Tendency in Python

When calculating statistics, remember two packages: NumPy and SciPy.

Mean

We calculate the mean by using NumPy.

Median

We calculate the median by using NumPy.

Mode

We calculate the mode by using SciPy.

SciPy has functions for the mean, the median, and the mode and was built as an extension to the NumPy codebase. But, NumPy is significantly faster than SciPy and more compatible with other libraries, like Pandas. So, we prefer it when multiple options for the same function exist.

Mean is calculated using **NumPy**.

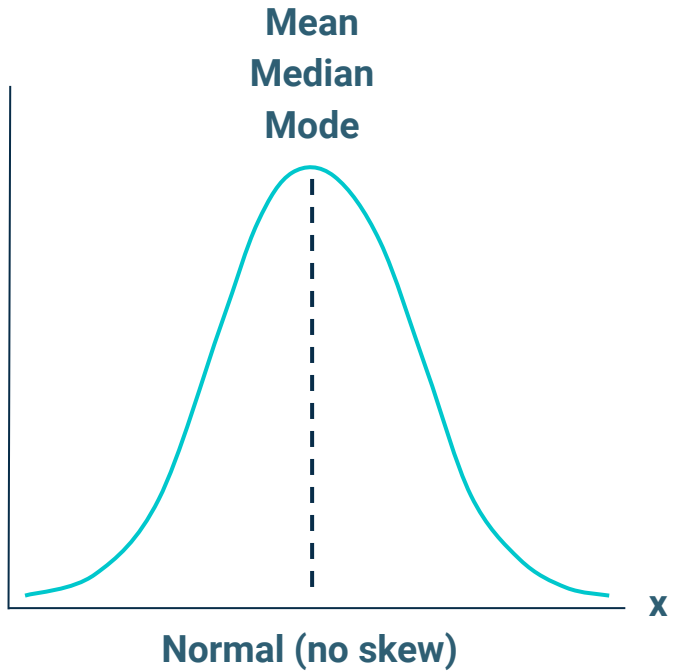
Median is calculated using **NumPy**.

Mode is calculated using **SciPy**.

Measures of Central Tendency: First Example

In this example, all three measures of central tendency have about the same value.

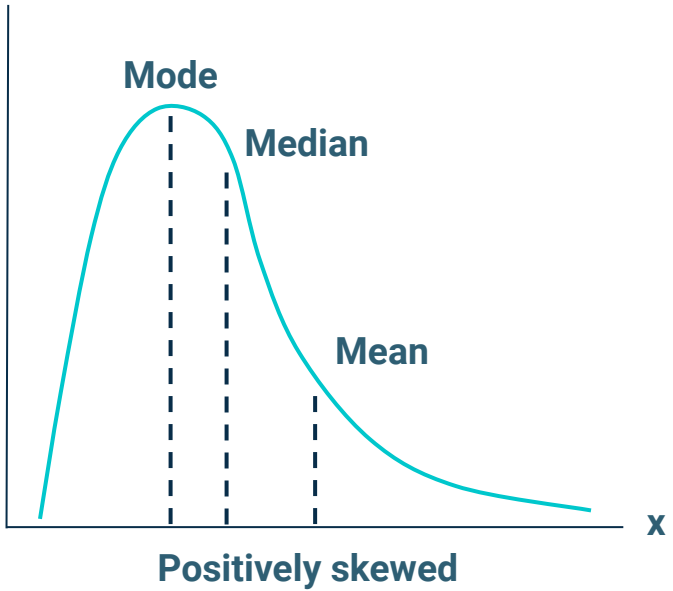
Any of the three measures of central tendency effectively describes the center of the data.



Measures of Central Tendency: Second Example

In this example, the mean of the dataset no longer effectively describes the center of the data.

The distribution of data can affect which measure of central tendency is best for a particular use case.



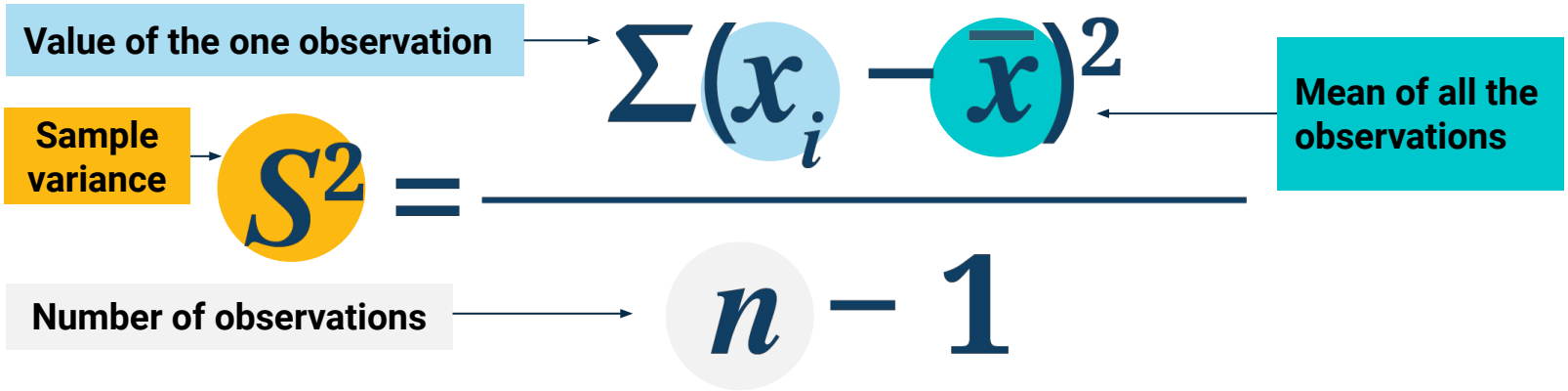
What are **variance, standard deviation,** and **z-score**?

Variance describes how far the values in the dataset are from the mean overall. That is, the variance describes how much variation exists within the data.

Variance Explained

Variance:

- Describes how far values in the dataset are from the mean.
- Describes how much variation exists in the data.
- Considers the distance of each value in the dataset from the center of the data.



Standard deviation: The square root of the variance, which converts the variance back into the units of the original data.

Standard Deviation Explained

Standard deviation:

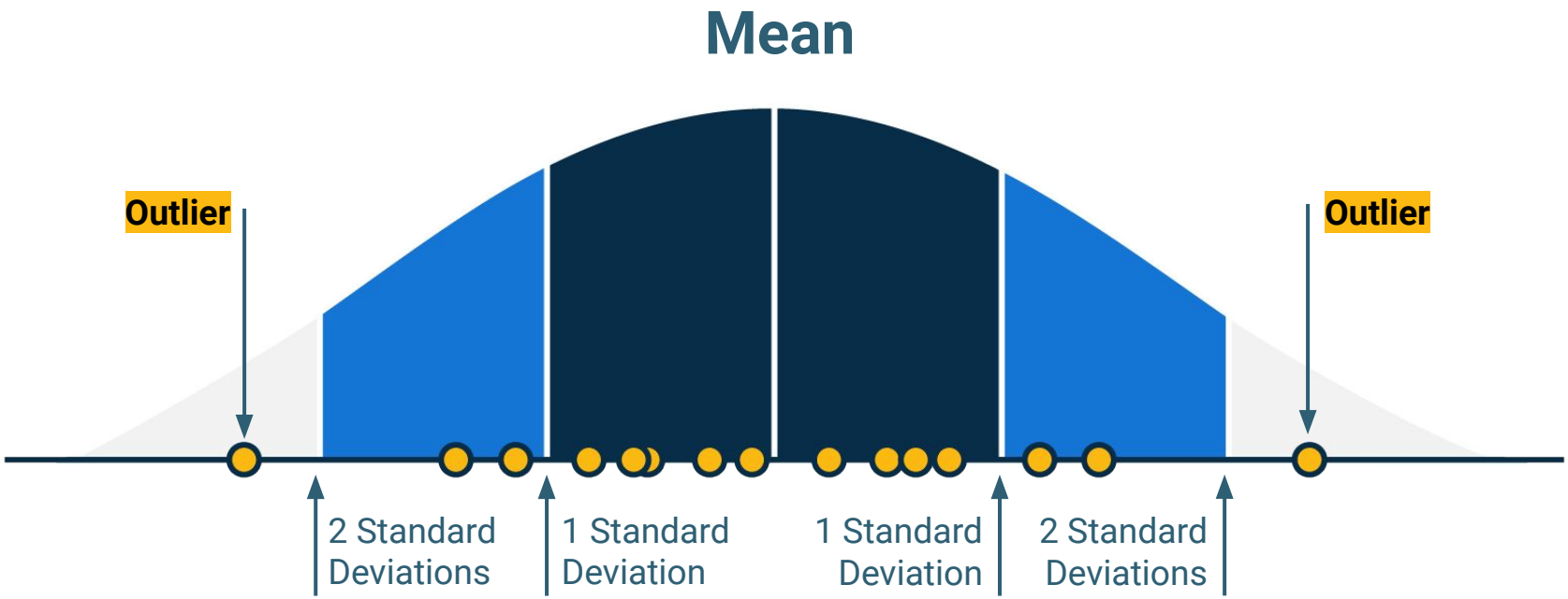
- Describes how spread out the data is from the mean.
- Gets calculated as the square root of the variance.
- Exists In the same unit of measurement as the mean (and the data).

$$\sigma = \sqrt{S^2}$$

Standard deviation **Variance**

Standard Deviation Explained

The **standard deviation** is the square root of the variance, which is a measure that's used to quantify the dispersion of a set of observations.

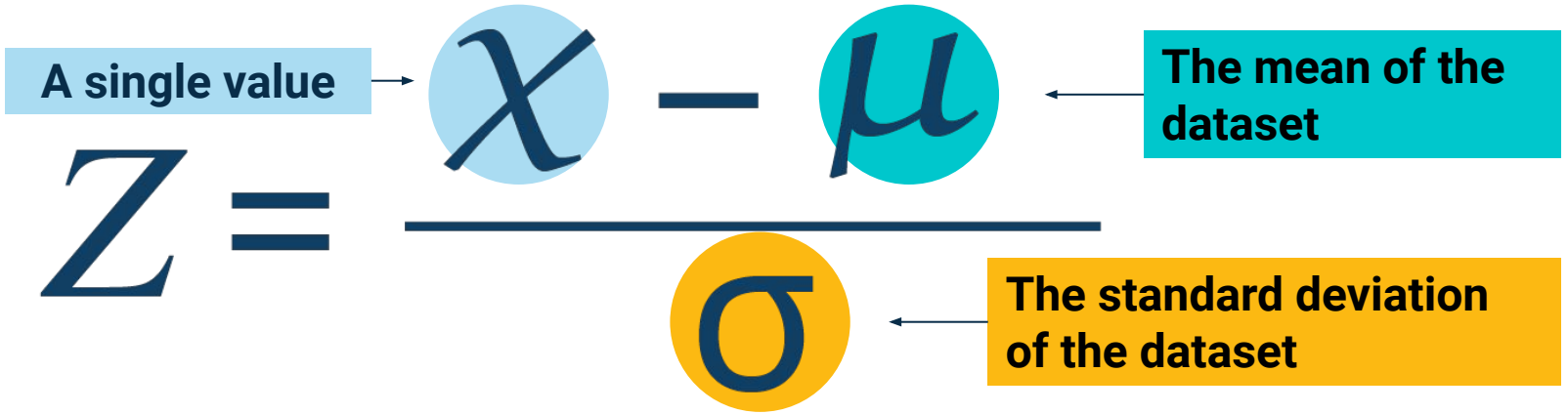


Z-score: A single data point's distance from the mean in terms of the standard deviation. The z-score is also called the standard score.

Z-Score Explained

The distance is in terms of standard deviations and can be positive or negative.

- If the z-score is negative, the value of the data point is less than the mean.
- If the z-score is positive, the value of the data point is greater than the mean.



The smaller the **z-score**, the closer the value is to the mean.

Variability of Data in Python

NumPy

In Python, we'll calculate the **variance** and the **standard deviation** by using the NumPy module.

SciPy

We'll calculate the mode and the **z-score** by using the SciPy module.

Instructor **Demonstration**

Summary Statistics

What are **quantiles, quartiles, percentiles,** and **outliers**?

Quantiles:

These are values that divide sorted data into well-defined bins based on the position of each point. The two most commonly used quantiles are quartiles and percentiles.

Quartiles:

These are the three values that divide the sorted data into four equally sized groups. Thus, 25% of the data values are less than the first quartile, 50% are less than the second quartile, and 75% are less than the third quartile. The second quartile is also the median.

Percentiles:

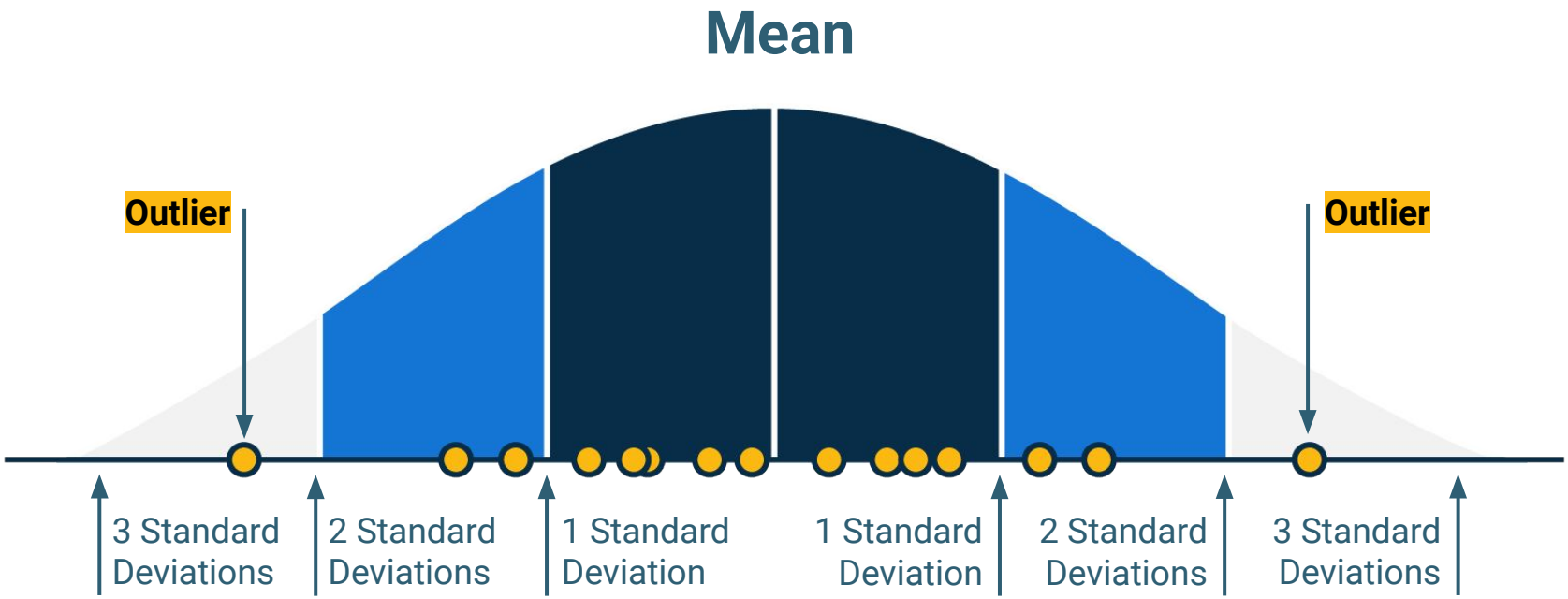
These divide the sorted data into 100 equally sized groups. Each percentile is named for the percentage of data values that are less than that percentile. For example, 57% of the data values are less than the 57th percentile.

Outliers:

Multiple mathematical calculations exist to find potential outliers, but in general outliers are extreme values in a dataset.

Outliers

We typically identify an outlier as a value that's 1.5 × IQR beyond the first or the third quartile.



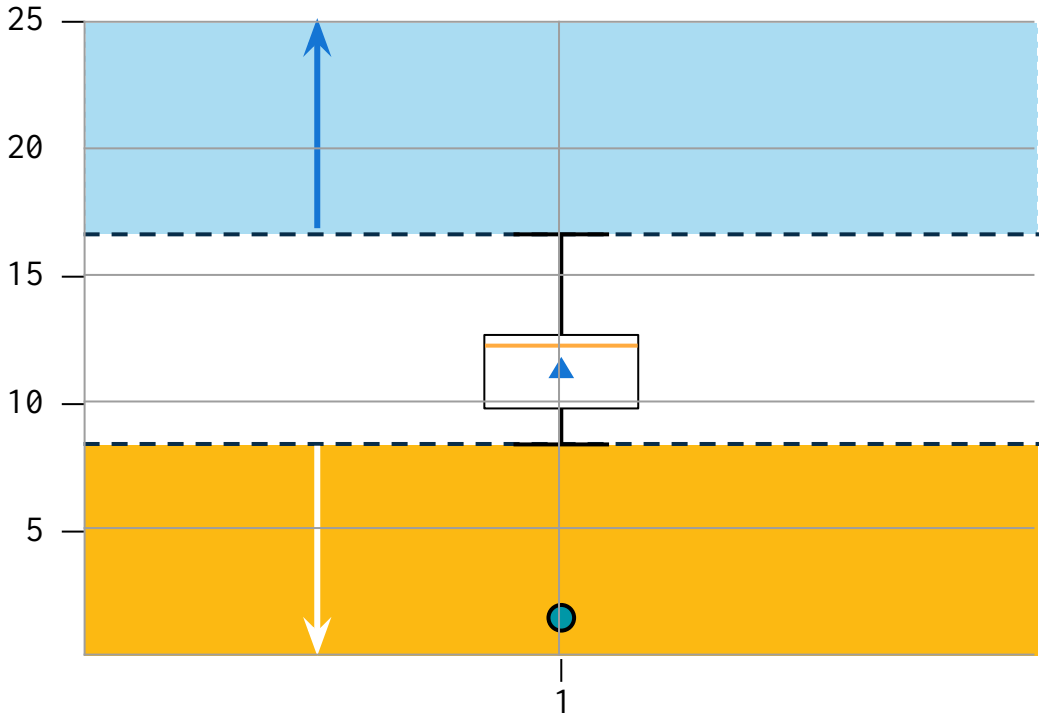
Manually Calculating the IQR

Determine the outlier boundaries in a dataset by using the **1.5 × IQR rule.**

The IQR is the range between the first and the third quartile.

Anything less than, or below, Quartile 1 − (1.5 × IQR) might be an outlier.

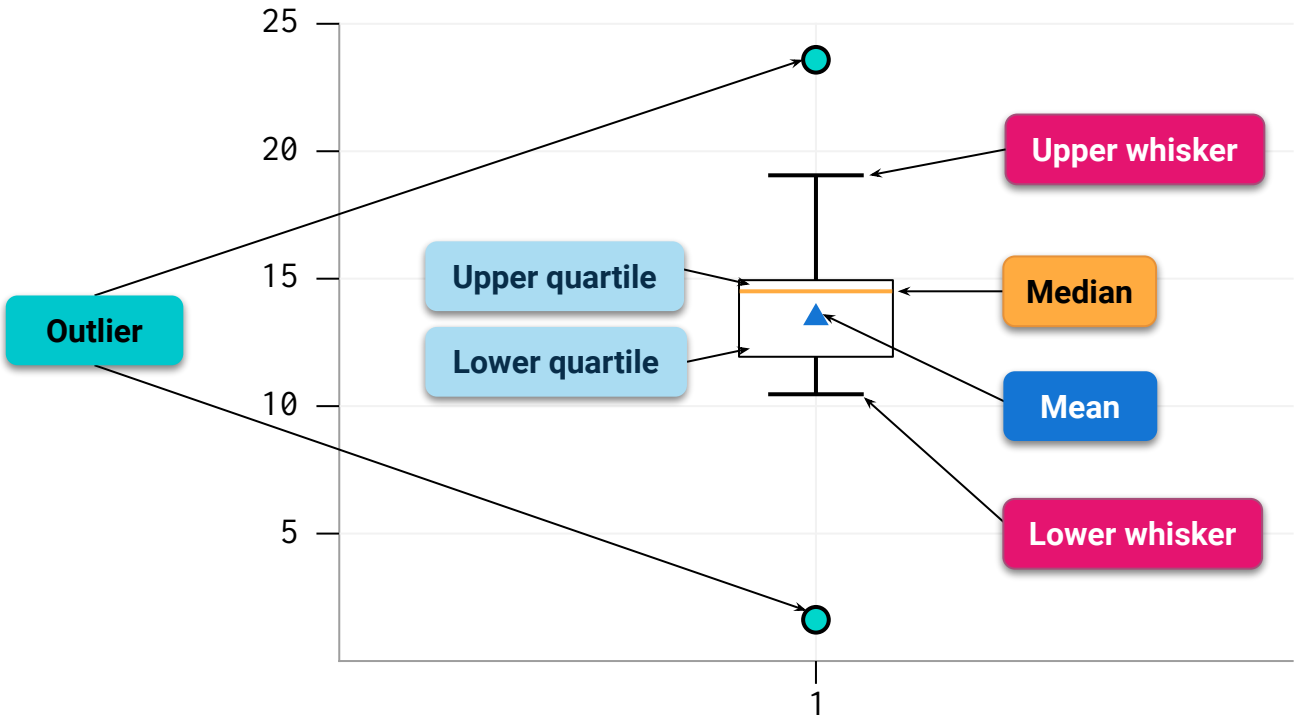
Anything greater than, or above, Quartile 3 + (1.5 × IQR) might be an outlier.



Instructor **Demonstration**

Quartiles and Outliers

How to Identify Potential Outliers in Python



Activity:
Summary Statistics

In this activity you'll practice calculation of the mean, median, and mode. You'll import a CSV file, determine central tendency of a population, identify any outliers through code, create a DataFrame of the outliers, and find maximum and minimum values.

Suggested Time:
10 Minutes

Time’s up!

Let’s review

Instructor **Demonstration**

Correlation

Pandas **Recap**

Suggested Time:
30 Minutes

Activity:
Pandas Recap

Together, we'll review what we have learned about Pandas up to this point.

Open the unsolved file in your Jupyter notebook. Go through the cells, and follow the instructions in the comments.

Suggested Time:
30 Minutes

Questions?

Next

In the lesson that follows, you will learn how to perform mathematical operations on columns within a DataFrame, format and replace text, create new columns, and use the `apply()` feature to transform columns.

The End