# Statistical Methods in Pandas

(https://github.com/learn-co-curriculum/dsc-statistical-methods-in-pandas) (https://github.com/learn-co-curriculum/dsc-statistical-methods-in-pandas/issues/new/choose)

## Introduction

In this lesson, you'll learn how to use some of the key summary statistics methods in Pandas.

## Objectives:

You will be able to:

- Calculate summary statistics for a series and DataFrame
- Use the `.apply()` or `.applymap()` methods to apply a function to a pandas series or DataFrame

## Getting DataFrame-Level Summary Statistics

When working with a new dataset, the first step is always to begin to understand what makes up that dataset. The Pandas DataFrame class contains two built-in methods that make this very easy for us.

### Using `df.info()`

The `df.info()` method provides us with summary ***metadata*** about our DataFrame -- that is, it gives us data about our dataset, such as how many rows and columns it contains, and what data types they are stored as. Let's demonstrate this by reading in the Titanic dataset and calling the `.info()` method on the DataFrame.

```
import pandas as pd
df = pd.read_csv('titanic.csv', index_col=0)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null object
Name           891 non-null object
Sex            891 non-null object
               714 non-null float64
               891 non-null int64
```

Help

```
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(4), object(6)
memory usage: 90.5+ KB
```

As we can see from the output above, the `.info()` method provides us with great information about the characteristics of the DataFrame, without telling us anything about the data it actually contains.

Examine the output above, and take note of the important things it tells us about the DataFrame, such as:

- The number of columns and rows in the DataFrame
- The data type of the data each column contains
- How many values each column contains (NaNs are not counted)
- The memory footprint of the DataFrame

This sort of information about a dataset is called ***metadata***, since it's data about our data.

# Using `.describe()`

The next step in Exploratory Data Analysis (EDA) is usually to dig into the summary statistics of the dataset, and get a feel for the data each column contains. Rather than force us to deal with the tedium of doing this individually for every column, Pandas DataFrames provide the handy `df.describe()` method which calculates the basic summary statistics for each column for us automatically.

See the example in the cell below.

```
df.describe()
```

| | PassengerId | Survived | Age | SibSp | Parch | Fare |
|---|---|---|---|---|---|---|
| **count** | 891.000000 | 891.000000 | 714.000000 | 891.000000 | 891.000000 | 891.000000 |
| **mean** | 446.000000 | 0.383838 | 29.699118 | 0.523008 | 0.381594 | 32.204208 |
| **std** | 257.353842 | 0.486592 | 14.526497 | 1.102743 | 0.806057 | 49.693429 |
| **min** | 1.000000 | 0.000000 | 0.420000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 223.500000 | 0.000000 | 20.125000 | 0.000000 | 0.000000 | 7.910400 |
| **50%** | 446.000000 | 0.000000 | 28.000000 | 0.000000 | 0.000000 | 14.454200 |
| **75%** | 668.500000 | 1.000000 | 38.000000 | 1.000000 | 0.000000 | 31.000000 |
| **max** | 891.000000 | 1.000000 | 80.000000 | 8.000000 | 6.000000 | 512.329200 |

Help

As we can see, the output of the `.describe()` method is very handy, and gives us relevant information such as:

- a `count` of the number of values in each column, making it identify columns with missing values
- The mean and standard deviation of each column
- The minimum and maximum values found in each column
- The median (50%) and quartile values (25% & 75%) for each column

Use the `.describe()` method to quickly help you get a feel for your datasets when you start the Exploratory Data Analysis process.

# Calculating Individual Column Statistics

If we need to calculate individual statistics about a column, we can also do this easily. Pandas DataFrames and Series objects come with a plethora of built-in methods to instantly calculate summary statistics for us.

See the code blocks below for examples:

```
df.mean()
```

```
PassengerId    446.000000
Survived         0.383838
Age             29.699118
SibSp            0.523008
Parch            0.381594
Fare            32.204208
dtype: float64
```

```
df['Fare'].mean()
```

```
32.2042079685746
```

```
df['Age'].quantile(.9)
```

```
50.0
```

```
df['Age'].median()
```

```
28.0
```

**Help**

There are many different statistical methods built into Pandas DataFrames -- these are just a few! We will not list all of them, but here are some common ones you'll probably make use of early and often:

- `.mode()` -- the mode of the column
- `.count()` -- the count of the total number of entries in a column
- `.std()` -- the standard deviation for the column
- `.var()` -- the variance for the column
- `.sum()` -- the sum of all values in the column
- `.cumsum()` -- the cumulative sum, where each cell index contains the sum of all indices lower than, and including, itself.

# Summary Statistics for Categorical Columns

Obviously, we cannot calculate most summary statistics on columns that contain non-numeric data -- there's no way for us to find the mean of the letters in the `Embarked` column, for instance. However, there are some summary statistics we can use to help us better understand our categorical columns.

See the examples in the cell below:

```
df['Embarked'].unique()
```

```
array(['S', 'C', 'Q', nan], dtype=object)
```

```
df['Embarked'].value_counts()
```

```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

These methods are extremely useful when dealing with categorical data!

`.unique()` shows us all the unique values contained in the column.

`.value_counts()` shows us a count for how many times each unique value is present in a dataset, giving us a feel for the distribution of values in the column.

# Calculating on the Fly with `.apply()` and `.applymap()`

Sometimes, we'll need to make changes to our dataset, or to compute functions on our data that aren't built-in to Pandas. We can do this by passing lambda values into the `apply()` method when w ? Help ndas series, and the `.applymap()` method when working with Pandas DataFrames.

Note that both of these do not mutate the original dataset -- instead, they return a copy of the Series or DataFrame containing the result.

See the example in the cell below:

```
string_df = df.applymap(lambda x: str(x))
string_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null object
Survived       891 non-null object
Pclass         891 non-null object
Name           891 non-null object
Sex            891 non-null object
Age            891 non-null object
SibSp          891 non-null object
Parch          891 non-null object
Ticket         891 non-null object
Fare           891 non-null object
Cabin          891 non-null object
Embarked       891 non-null object
dtypes: object(12)
memory usage: 90.5+ KB
```

```
display(df['Age'].apply(lambda x: x**2).head())

df['Age'].head()
```

```
0     484.0
1    1444.0
2     676.0
3    1225.0
4    1225.0
Name: Age, dtype: float64
```

```
0    22.0
```

Help

```
2
```

```
3    35.0
4    35.0
Name: Age, dtype: float64
```

# Summary

In this lesson, you learned how to:

- Understand and use the `df.describe()` and `df.info()` summary statistics methods
- Use built-in Pandas methods for calculating summary statistics
- Apply a function to every element in a Series or DataFrame using `s.apply()` and `df.applymap()`

How do you feel about this lesson?

Have specific feedback?

**Tell us here! (https://github.com/learn-co-curriculum/dsc-statistical-methods-in-pandas/issues/new/choose)**

Help