

# Lesson 3

## Overview of Python for Machine Learning

# Python Review

- When getting started in Python you need to know a few key details about the language syntax.
- This includes:
  - Assignment.
  - Flow Control.
  - Data Structures.
  - Functions.

# Python Review

- Assignment
- As a programmer, assignment and types should not be surprising to you.
- Strings

# Strings

```
data = 'hello world'
```

```
print(data[0])
```

```
print(len(data))
```

```
print(data)
```

# Python Review

- Notice how you can access characters in the string using array syntax.  
Running the example prints:

h

11

hello world

# Python Review

- Numbers

# Numbers

```
value = 123.1
```

```
print(value)
```

```
value = 10
```

```
print(value)
```

# Python Review

- Boolean
- # Boolean
- a = True
- b = False
- print(a, b)

# Python Review

- **Multiple Assignment**

# Multiple Assignment

```
a, b, c = 1, 2, 3
```

```
print(a, b, c)
```

- **No Value**

# No value

```
a = None
```

```
print(a)
```

# Python Review

- **Flow Control**

- There are three main types of flow control that you need to learn:

- If-Then-Else conditions,
- For-Loops and While-Loops.
- If-Then-Else Conditional

```
value = 99
```

```
if value == 99:
```

```
    print('That is fast')
```

```
elif value > 200:
```

```
    print('That is too fast')
```

```
else:
```

```
    print('That is safe')
```



# Python Review

- **For-Loop**

# For-Loop

```
for i in range(10):  
    print(i)
```

- **While-Loop**

# While-Loop

```
i = 0
```

```
while i < 10:  
    print(i)  
    i += 1
```

# Data Structures

- There are three data structures in Python that you will find the most used and useful.
- They are tuples, lists and dictionaries.

- **Tuple:-** Tuples are read-only collections of items.

```
a = (1, 2, 3)
```

```
print(a)
```

# Data Structures

- **List**

- Lists use the square bracket notation and can be index using array notation.

```
mylist = [1, 2, 3]
```

```
print("Zeroth Value: %d" % mylist[0])
```

```
mylist.append(4)
```

```
print("List Length: %d" % len(mylist))
```

```
for value in mylist:
```

```
    print(value)
```

- Notice that we are using some simple printf-like functionality to combine strings and variables when printing.

# Data Structures

- **Dictionary**
- Dictionaries are mappings of names to values, like key-value pairs.
- Note the use of the curly bracket and colon notations when defining the dictionary.

```
mydict = {'a': 1, 'b': 2, 'c': 3}
print("A value: %d" % mydict['a'])
mydict['a'] = 11
print("A value: %d" % mydict['a'])
print("Keys: %s" % mydict.keys())
print("Values: %s" % mydict.values())
for key in mydict.keys():
    print(mydict[key])
```

# Data Structures

- **Functions**

- The biggest gotcha with Python is the whitespace.
- Ensure that you have an empty new line after indented code.
- The example below defines a new function to calculate the sum of two values and calls the function with two arguments.

*# Sum function*

```
def mysum(x, y):
```

```
    return x + y
```

*# Test sum function*

```
result = mysum(1, 3)
```

```
print(result)
```

# NumPy overview

- NumPy provides the foundation data structures and operations for SciPy. These are arrays (ndarrays) that are efficient to define and manipulate.

- **Create Array**

*# define an array*

*import numpy*

*mylist = [1, 2, 3]*

*myarray = numpy.array(mylist)*

*print(myarray)*

*print(myarray.shape)*

- Notice how we easily converted a Python list to a NumPy array.

# NumPy overview

- **Access Data**

- Array notation and ranges can be used to efficiently access data in a NumPy array.

*# access values*

```
import numpy
```

```
mylist = [[1, 2, 3], [3, 4, 5]]
```

```
myarray = numpy.array(mylist)
```

```
print(myarray)
```

```
print(myarray.shape)
```

```
print("First row: %s" % myarray[0])
```

```
print("Last row: %s" % myarray[-1])
```

```
print("Specific row and col: %s" % myarray[0, 2])
```

```
print("Whole col: %s" % myarray[:, 2])
```

```
[[1 2 3]
```

```
 [3 4 5]]
```

```
(2, 3)
```

```
First row: [1 2 3]
```

```
Last row: [3 4 5]
```

```
Specific row and col: 3
```

```
Whole col: [3 5]
```

# NumPy overview

- **Arithmetic**

- NumPy arrays can be used directly in arithmetic.

*# arithmetic*

*import numpy*

*myarray1 = numpy.array([2, 2, 2])*

*myarray2 = numpy.array([3, 3, 3])*

*print("Addition: %s" % (myarray1 + myarray2))*

*print("Multiplication: %s" % (myarray1 \* myarray2))*

```
Addition: [5 5 5]  
Multiplication: [6 6 6]
```



# Matplotlib overview

- Matplotlib can be used for creating plots and charts.
- The library is generally used as follows:
  - Call a plotting function with some data (e.g. `.plot()`).
  - Call many functions to setup the properties of the plot (e.g. labels and colors).
  - Make the plot visible (e.g. `.show()`).

# Matplotlib overview

- **Line Plot**

- The example below creates a simple line plot from one dimensional data.

*# basic line plot*

*import matplotlib.pyplot as plt*

*import numpy*

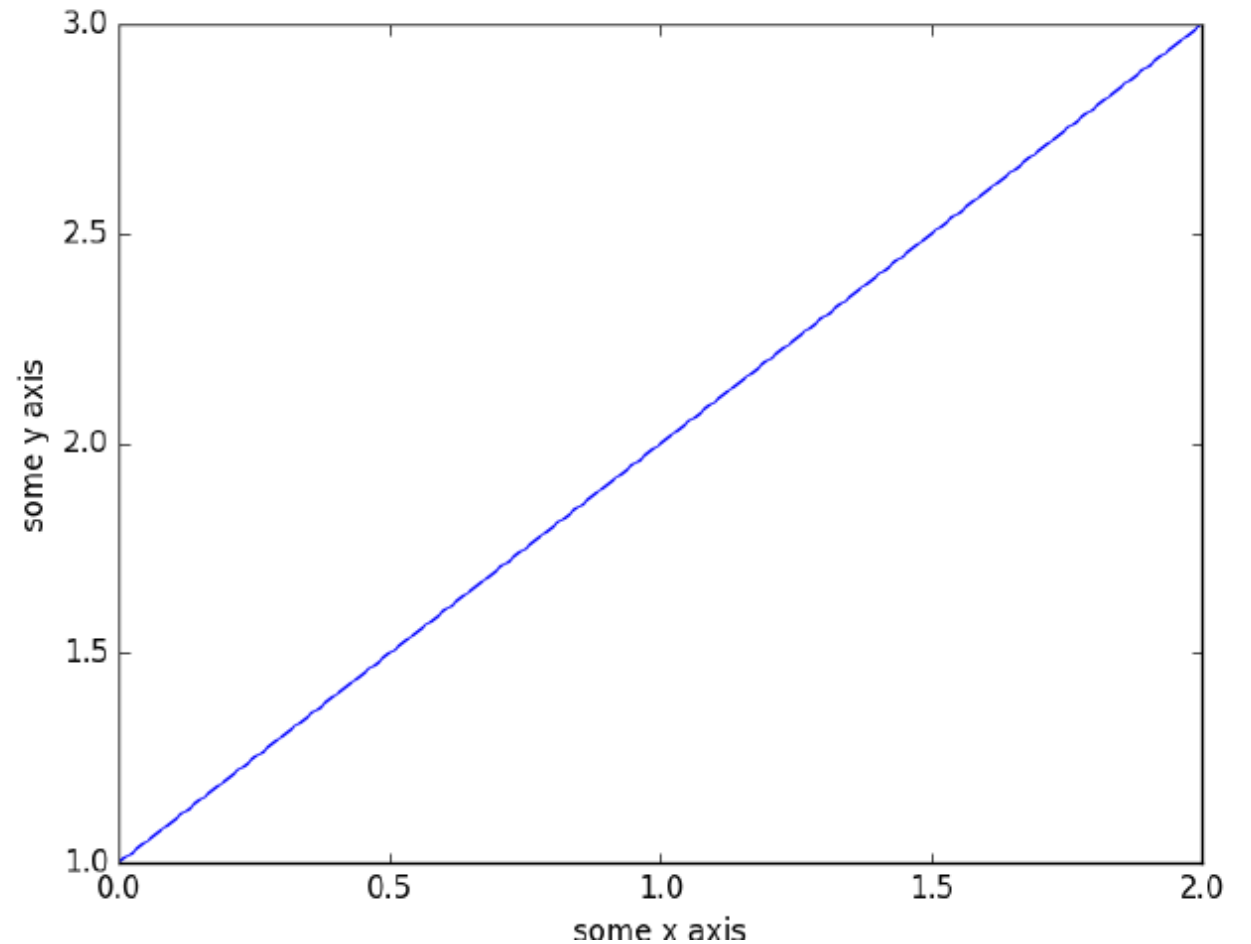
*myarray = numpy.array([1, 2, 3])*

*plt.plot(myarray)*

*plt.xlabel('some x axis')*

*plt.ylabel('some y axis')*

*plt.show()*



# Matplotlib overview

- **Scatter Plot**

- Below is a simple example of
- creating a scatter plot from two dimensional data.

*# basic scatter plot*

*import matplotlib.pyplot as plt*

*import numpy*

*x = numpy.array([1, 2, 3])*

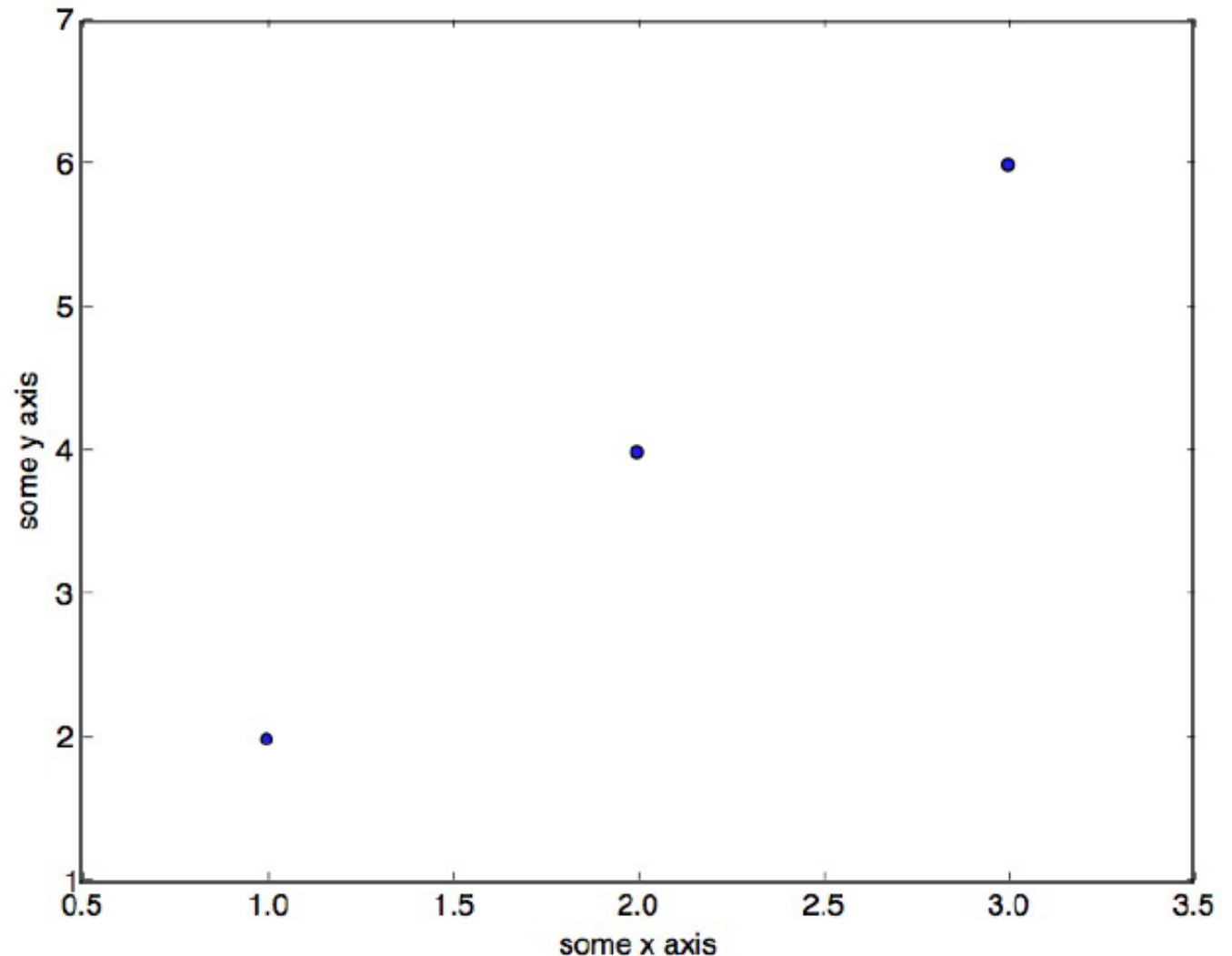
*y = numpy.array([2, 4, 6])*

*plt.scatter(x,y)*

*plt.xlabel('some x axis')*

*plt.ylabel('some y axis')*

*plt.show()*



# Pandas overview

- Pandas provides data structures and functionality to quickly manipulate and analyze data.
- The key to understanding Pandas for machine learning is understanding the Series and DataFrame data structures.

# Pandas overview

- **Series**

- A series is a one dimensional array of data where the rows are labeled using a time axis.

*# series*

*import numpy*


*import pandas*

*myarray = numpy.array([1, 2, 3])*

*rownames = ['a', 'b', 'c']*

*myseries = pandas.Series(myarray, index=rownames)*

*print(myseries)*



a	1
b	2
c	3

# Pandas overview

- You can access the data in a series like a NumPy array and like a dictionary, for example:

```
print(myseries[0])
```

```
print(myseries['a'])
```



# Pandas overview

- **DataFrame**

- A data frame is a multi-dimensional array where the rows and the columns can be labeled.

*# dataframe*

*import numpy*

*import pandas*

*myarray = numpy.array([[1, 2, 3], [4, 5, 6]])*

*rownames = ['a', 'b']*

*colnames = ['one', 'two', 'three']*

*mydataframe = pandas.DataFrame(myarray, index=rownames, columns=colnames)*

*print(mydataframe)*

	one	two	three
a	1	2	3
b	4	5	6

# Pandas overview

- Data can be indexed using column names.

*print("method 1:")*

*print("one column:\n%s" % mydataframe['one'])*

*print("method 2:")*

*print("one column:\n%s" % mydataframe.one)*

```
method 1:
one column:
a      1
b      4
method 2:
one column:
a      1
b      4
```



# Loading Machine Learning Data

- You must be able to load your data before you can start your machine learning project.
- The most common format for machine learning data is CSV files.
- There are a number of ways to load a CSV file in Python.
- We look at three ways that you can use to load your CSV data in Python:
  - 1. Load CSV Files with the Python Standard Library.
  - 2. Load CSV Files with NumPy.
  - 3. Load CSV Files with Pandas.

# Considerations When Loading CSV Data

- There are a number of considerations when loading your machine learning data from CSV files.
- For reference, you can learn a lot about the expectations for CSV files by reviewing the CSV request for comment titled Common Format and MIME Type for Comma-Separated Values (CSV) Files (See this link <https://tools.ietf.org/html/rfc4180>).

# File Header

- Does your data have a file header? If so this can help in automatically assigning names to each column of data.
- If not, you may need to name your attributes manually.
- Either way, you should explicitly specify whether or not your CSV file had a file header when loading your data.

# Comments

- Does your data have comments? Comments in a CSV file are indicated by a hash (#) at the start of a line.
- If you have comments in your file, depending on the method used to load your data, you may need to indicate whether or not to expect comments and the character to expect to signify a comment line.

# Delimiter

- The standard delimiter that separates values in fields is the comma (,) character.
- Your file could use a different delimiter like tab or white space in which case you must specify it explicitly.

# Quotes

- Sometimes field values can have spaces. In these CSV files the values are often quoted.
- The default quote character is the double quotation marks character.
- Other characters can be used, and you must specify the quote character used in your file.

# Pima Indians Dataset

- The Pima Indians dataset is used to demonstrate data loading in this lesson. It will also be used in many of the lessons to come.
- This dataset describes the medical records for Pima Indians and whether or not each patient will have an onset of diabetes within five years.
- As such it is a classification problem. It is a good dataset for demonstration because all of the input attributes are numeric and the output variable to be predicted is binary (0 or 1).
- The data is freely available from the UCI Machine Learning Repository (<https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes>).

# Load CSV Files with the Python Standard Library

- The Python API provides the module `CSV` and the function `reader()` that can be used to load CSV files.
- Once loaded, you can convert the CSV data to a NumPy array and use it for machine learning.
- For example, you can download (<https://goo.gl/vhm1eU>) the Pima Indians dataset into your local directory with the filename `pima-indians-diabetes.data.csv`.
- All fields in this dataset are numeric and there is no header line.



# Example of loading a CSV file using the Python standard library.

*# Load CSV Using Python Standard Library*

```
import csv
```

```
import numpy
```

```
filename = 'pima-indians-diabetes.data.csv'
```

```
raw_data = open(filename, 'rt')
```

```
reader = csv.reader(raw_data, delimiter=',', quoting=csv.QUOTE_NONE)
```

```
x = list(reader)
```

```
data = numpy.array(x).astype('float')
```

```
print(data.shape)
```

- The example loads an object that can iterate over each row of the data and can easily be converted into a NumPy array. Running the example prints the shape of the array.

```
(768, 9)
```

- For more information on the `csv.reader()` function, see CSV File Reading and Writing in the Python API documentation (<https://docs.python.org/2/library/csv.html>).

# Load CSV Files with NumPy

- You can load your CSV data using NumPy and the `numpy.loadtxt()` function.
- This function assumes no header row and all data has the same format. The example below assumes that the file `pima-indians-diabetes.data.csv` is in your current working directory.

```
# Load CSV using NumPy  
from numpy import loadtxt  
filename = 'pima-indians-diabetes.data.csv'  
raw_data = open(filename, 'rt')  
data = loadtxt(raw_data, delimiter=",")  
print(data.shape)
```

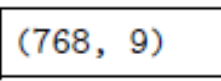
Running the example will load the data as a `numpy.ndarray` (<http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.ndarray.html>) and print the shape of the data:

```
(768, 9)
```

# This example can be modified to load the same dataset directly from a URL as follows:

```
# Load CSV from URL using NumPy  
from numpy import loadtxt  
from urllib.request import urlopen  
url = 'https://goo.gl/vhm1eU'  
raw_data = urlopen(url)  
dataset = loadtxt(raw_data, delimiter=',')  
print(dataset.shape)
```

*Same results are produced when this example is executed*



(768, 9)

For more information on the `numpy.loadtxt()` function see the API documentation (<http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.loadtxt.html> ).

# Load CSV Files with Pandas

- You can load your CSV data using Pandas and the `pandas.read_csv()` function.
- This function is very flexible and is perhaps my recommended approach for loading your machine learning data.
- The function returns a `pandas.DataFrame` (<http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html>) that you can immediately start summarizing and plotting.
- The example below assumes that the `pima-indians-diabetes.data.csv` file is in the current working directory.

# Example of loading a CSV file using Pandas.

```
from urllib.request import urlopen# Load CSV using Pandas
from pandas import read_csv
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
print(data.shape)
```

- Note that in this example we explicitly specify the names of each attribute to the DataFrame.
- Running the example displays the shape of the data:

```
(768, 9)
```

We can also modify this example to load CSV data directly from a URL.

*# Load CSV using Pandas from URL*

*from pandas import read\_csv*

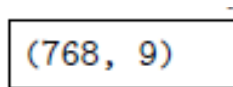
*url = 'https://goo.gl/vhm1eU'*

*names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']*

*data = read\_csv(url, names=names)*

*print(data.shape)*

- Again, running the example downloads the CSV file, parses it and displays the shape of the loaded DataFrame.



(768, 9)

- To learn more about the `pandas.read_csv()` function you can refer to the API documentation ([http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read\\_csv.html](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html)).
- Generally I recommend that you load your data with Pandas in practice and all subsequent examples in this course will use this method.

# Understand Your Data With Descriptive Statistics

- You must understand your data in order to get the best results. In this session you will discover 7 sample code snippets that you can use in Python to better understand your machine learning data.
- After this section you will know how to:
  - 1. Take a peek at your raw data.
  - 2. Review the dimensions of your dataset.
  - 3. Review the data types of attributes in your data.
  - 4. Summarize the distribution of instances across classes in your dataset.
  - 5. Summarize your data using descriptive statistics.
  - 6. Understand the relationships in your data using correlations.
  - 7. Review the skew of the distributions of each attribute.
- Each point is demonstrated by loading the Pima Indians Diabetes classification dataset from the UCI Machine Learning repository.
- Open your Python interactive environment and try each example out in turn.

# Peek at Your Data

- There is no substitute for looking at the raw data. Looking at the raw data can reveal insights that you cannot get any other way.
- It can also plant seeds that may later grow into ideas on how to better pre-process and handle the data for machine learning tasks.
- You can review the first 20 rows of your data using the `head()` function on the Pandas DataFrame.



# Example of reviewing the first few rows of data.

*# View first 20 rows*

*from pandas import read\_csv*

*filename = "pima-indians-diabetes.data.csv"*

*names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']*

*data = read\_csv(filename, names=names)*

*peek = data.head(20)*

*print(peek)*

You can see that the first column lists the row number, which is handy for referencing a specific observation.

	preg	plas	pres	skin	test	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1
10	4	110	92	0	0	37.6	0.191	30	0
11	10	168	74	0	0	38.0	0.537	34	1
12	10	139	80	0	0	27.1	1.441	57	0
13	1	189	60	23	846	30.1	0.398	59	1
14	5	166	72	19	175	25.8	0.587	51	1
15	7	100	0	0	0	30.0	0.484	32	1
16	0	118	84	47	230	45.8	0.551	31	1
17	7	107	74	0	0	29.6	0.254	31	1
18	1	103	30	38	83	43.3	0.183	33	0
19	1	115	70	30	96	34.6	0.529	32	1

# Dimensions of Your Data

- You must have a very good handle on how much data you have, both in terms of rows and columns.
  - Too many rows and algorithms may take too long to train. Too few and perhaps you do not have enough data to train the algorithms.
  - Too many features and some algorithms can be distracted or suffer poor performance due to the curse of dimensionality.
- You can review the shape and size of your dataset by printing the shape property on the Pandas DataFrame.

*# Dimensions of your data*

*from pandas import read\_csv*

*filename = "pima-indians-diabetes.data.csv"*

*names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']*

*data = read\_csv(filename, names=names)*

*shape = data.shape*

*print(shape)*

- The results are listed in rows then columns. You can see that the dataset has 768 rows and 9 columns.

```
(768, 9)
```

# Data Type For Each Attribute

- The type of each attribute is important. Strings may need to be converted to floating point values or integers to represent categorical or ordinal values.
- You can get an idea of the types of attributes by peeking at the raw data, as previously done.
- You can also list the data types used by the DataFrame to characterize each attribute using the dtypes property.

# Example of reviewing the data types of the data.

*# Data Types for Each Attribute*

*from pandas import read\_csv*

*filename = "pima-indians-diabetes.data.csv"*

*names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']*

*data = read\_csv(filename, names=names)*

*types = data.dtypes*

*print(types)*

- You can see that most of the attributes are integers and that mass and pedi are floating point types.

```
preg      int64
plas      int64
pres      int64
skin      int64
test      int64
mass      float64
pedi      float64
age       int64
class     int64
dtype: object
```

# Descriptive Statistics

- Descriptive statistics can give you great insight into the shape of each attribute.
- Often you can create more summaries than you have time to review.
- The `describe()` function on the Pandas DataFrame lists 8 statistical properties of each attribute. They are:
  - Count.
  - Mean.
  - Standard Deviation.
  - Minimum Value.
  - 25th Percentile.
  - 50th Percentile (Median).
  - 75th Percentile.
  - Maximum Value.

# Example of reviewing a statistical summary of the data.

```
# Statistical Summary
```

```
from pandas import read_csv
```

```
from pandas import set_option
```

```
filename = "pima-indians-diabetes.data.csv"
```

```
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
```

```
data = read_csv(filename, names=names)
```

```
set_option('display.width', 100)
```

```
set_option('precision', 3)
```

```
description = data.describe()
```

```
print(description)
```

## Example of reviewing a statistical summary of the data.

- You can see that you do get a lot of data. You will note some calls to `pandas.set_option()` in the sample code to change the precision of the numbers and the preferred width of the output.
- This is to make it more readable for this example.
- When describing your data this way, it is worth taking some time and reviewing observations from the results.
- This might include the presence of NA values for missing data or surprising distributions for attributes.



## Output of reviewing a statistical summary of the data.

	preg	plas	pres	skin	test	mass	pedi	age	class
count	768.000	768.000	768.000	768.000	768.000	768.000	768.000	768.000	768.000
mean	3.845	120.895	69.105	20.536	79.799	31.993	0.472	33.241	0.349
std	3.370	31.973	19.356	15.952	115.244	7.884	0.331	11.760	0.477
min	0.000	0.000	0.000	0.000	0.000	0.000	0.078	21.000	0.000
25%	1.000	99.000	62.000	0.000	0.000	27.300	0.244	24.000	0.000
50%	3.000	117.000	72.000	23.000	30.500	32.000	0.372	29.000	0.000
75%	6.000	140.250	80.000	32.000	127.250	36.600	0.626	41.000	1.000
max	17.000	199.000	122.000	99.000	846.000	67.100	2.420	81.000	1.000

# Class Distribution (Classification Only)

- On classification problems you need to know how balanced the class values are.
- Highly imbalanced problems (a lot more observations for one class than another) are common and may need special handling in the data preparation stage of your project.
- You can quickly get an idea of the distribution of the class attribute in Pandas.

# Example of reviewing a class breakdown of the data.

*# Class Distribution*

*from pandas import read\_csv*

*filename = "pima-indians-diabetes.data.csv"*

*names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']*

*data = read\_csv(filename, names=names)*

*class\_counts = data.groupby('class').size()*

*print(class\_counts)*

- You can see that there are nearly double the number of observations with class 0 (no onset of diabetes) than there are with class 1 (onset of diabetes).

class	
0	500
1	268

# Correlations Between Attributes

- Correlation refers to the relationship between two variables and how they may or may not change together.
- The most common method for calculating correlation is Pearson's Correlation Coefficient, that assumes a normal distribution of the attributes involved.
- A correlation of -1 or 1 shows a full negative or positive correlation respectively. Whereas a value of 0 shows no correlation at all.
- Some machine learning algorithms like linear and logistic regression can suffer poor performance if there are highly correlated attributes in your dataset.
- As such, it is a good idea to review all of the pairwise correlations of the attributes in your dataset.
- You can use the `corr()` function on the Pandas DataFrame to calculate a correlation matrix.

## Example of reviewing correlations of attributes in the data.

```
# Pairwise Pearson correlations  
from pandas import read_csv  
from pandas import set_option  
filename = "pima-indians-diabetes.data.csv"  
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']  
data = read_csv(filename, names=names)  
set_option('display.width', 100)  
set_option('precision', 3)  
correlations = data.corr(method='pearson')  
print(correlations)
```

## Example of reviewing correlations of attributes in the data.

- The matrix lists all attributes across the top and down the side, to give correlation between all pairs of attributes (twice, because the matrix is symmetrical).
- You can see the diagonal line through the matrix from the top left to bottom right corners of the matrix shows perfect correlation of each attribute with itself.

---

	preg	plas	pres	skin	test	mass	pedi	age	class
preg	1.000	0.129	0.141	-0.082	-0.074	0.018	-0.034	0.544	0.222
plas	0.129	1.000	0.153	0.057	0.331	0.221	0.137	0.264	0.467
pres	0.141	0.153	1.000	0.207	0.089	0.282	0.041	0.240	0.065
skin	-0.082	0.057	0.207	1.000	0.437	0.393	0.184	-0.114	0.075
test	-0.074	0.331	0.089	0.437	1.000	0.198	0.185	-0.042	0.131
mass	0.018	0.221	0.282	0.393	0.198	1.000	0.141	0.036	0.293
pedi	-0.034	0.137	0.041	0.184	0.185	0.141	1.000	0.034	0.174
age	0.544	0.264	0.240	-0.114	-0.042	0.036	0.034	1.000	0.238
class	0.222	0.467	0.065	0.075	0.131	0.293	0.174	0.238	1.000

# Skew of Univariate Distributions

- Skew refers to a distribution that is assumed Gaussian (normal or bell curve) that is shifted or squashed in one direction or another.
- Many machine learning algorithms assume a Gaussian distribution. Knowing that an attribute has a skew may allow you to perform data preparation to correct the skew and later improve the accuracy of your models.
- You can calculate the skew of each attribute using the `skew()` function on the Pandas DataFrame.

## Example of reviewing skew of attribute distributions in the data.

*# Skew for each attribute*

*from pandas import read\_csv*

*filename = "pima-indians-diabetes.data.csv"*

*names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']*

*data = read\_csv(filename, names=names)*

*skew = data.skew()*

*print(skew)*

- The skew result show a positive (right) or negative (left) skew. Values closer to zero show less skew.

preg	0.901674
plas	0.173754
pres	-1.843608
skin	0.109372
test	2.272251
mass	-0.428982
pedi	1.919911
age	1.129597
class	0.635017



# Tips To Remember

- This section gives you some tips to remember when reviewing your data using summary statistics.
  - Review the numbers. Generating the summary statistics is not enough. Take a moment to pause, read and really think about the numbers you are seeing.
  - Ask why. Review your numbers and ask a lot of questions. How and why are you seeing specific numbers. Think about how the numbers relate to the problem domain in general and specific entities that observations relate to.
  - Write down ideas. Write down your observations and ideas. Keep a small text file or note pad and jot down all of the ideas for how variables may relate, for what numbers mean, and ideas for techniques to try later. The things you write down now while the data is fresh will be very valuable later when you are trying to think up new things to try.

END