Python Ecosystem for Machine Learning

LEARNING VOYAGE

Python's use for Machine Learning

1. Python and its rising use for machine learning.
2. SciPy and the functionality it provides with NumPy, Matplotlib and Pandas.
3. scikit-learn that provides all of the machine learning algorithms.
4. How to setup your Python ecosystem for machine learning and what versions to use.

Readability is a key benefit

SciPy

- **NumPy**: A foundation for SciPy that allows you to efficiently work with data in arrays.
- **Matplotlib**: Allows you to create 2D charts and plots from data.
- **Pandas**: Tools and data structures to organize and analyze your data.

scikit-learn

- Helps you to develop and practice machine learning in Python
- Open Source
- The focus of the library is machine learning algorithms for ***classification, regression, clustering*** and more.
- It also provides tools for related tasks such as evaluating models, tuning parameters and pre-processing data.

How to Install Python

https://repl.it/languages/python3

```
python --version
```

```
Python 3.6.3
```

How to install SciPy

- scipy
- numpy
- matplotlib
- pandas

How to install SciPy

```
# scipy import scipy
print('scipy: %s' % scipy.__version__)
# numpy import numpy
print('numpy: %s' % numpy.__version__)

# matplotlib import matplotlib

print('matplotlib: %s' % matplotlib.__version__)

# pandas import pandas

print('pandas: %s' % pandas.__version__)
```

How to install scikit learn

```
#scikit-learn
import sklearn
print('sklearn: %s' % sklearn.__version__)
```

```
 sklearn: 0.20.2
```

How to install the Ecosystem (Alternative)

https://www.continuum.io/downloads

Python and SciPy

Python - Crash Course

- Assignment.
- Flow Control.
- Data Structures.
- Functions.

Python - Crash Course

Strings

```
# Strings

data ='hello world'

print(data[0])

print(len(data))

print(data)

h

11

hello world
```

Python - Crash Course

Numbers

```
# Numbers


value = 123.1

print(value)

value = 10

print(value)
```

```
123.1
10
```

Python - Crash Course

Boolean

```
# Boolean

a = True

b = False

print(a, b)


True, False
```

Python - Crash Course

Multiple Assignment

```
# Multiple Assignment

a, b, c = 1, 2, 3

print(a, b, c)
```

```
1, 2, 3
```

Python - Crash Course

No Value

```
# No value

a = None

print(a)
```

```
None
```

Flow Control

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')
```

```
That is fast
```

Flow Control

For-Loop

```
# For-Loop

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

```
0
1
2
3
4
5
6
7
8
9
```

Flow Control

While-Loop

```
# While-Loop

i = 0

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

Data Structures

- Tuples
- Lists
- Dictionaries

Data Structures

Tuple

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

```
(1, 2, 3)
```

Data Structures

List

```
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)
```

```
Zeroth Value: 1
List Length: 4
1
2
3
4
```

Data Structures

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

```
A value: 1
A value: 11
Keys: ['a','c','b']
Values: [11, 3, 2]
11
3
2
```

Data Structures

Functions

```
# Sum function
def mysum(x, y):
      return x + y


#Test sum function
result = mysum(1, 3)
print(result)
```

```
4
```

NumPy Crash Course

Create Array

```
# define an array

import numpy

mylist = [1, 2, 3]

myarray = numpy.array(mylist)

print(myarray)

print(myarray.shape)
```

```
[1 2 3]


(3,)
```

NumPy Crash Course

Access Data

```
# 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])
```

NumPy Crash Course


```
[[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 Crash Course

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))
```

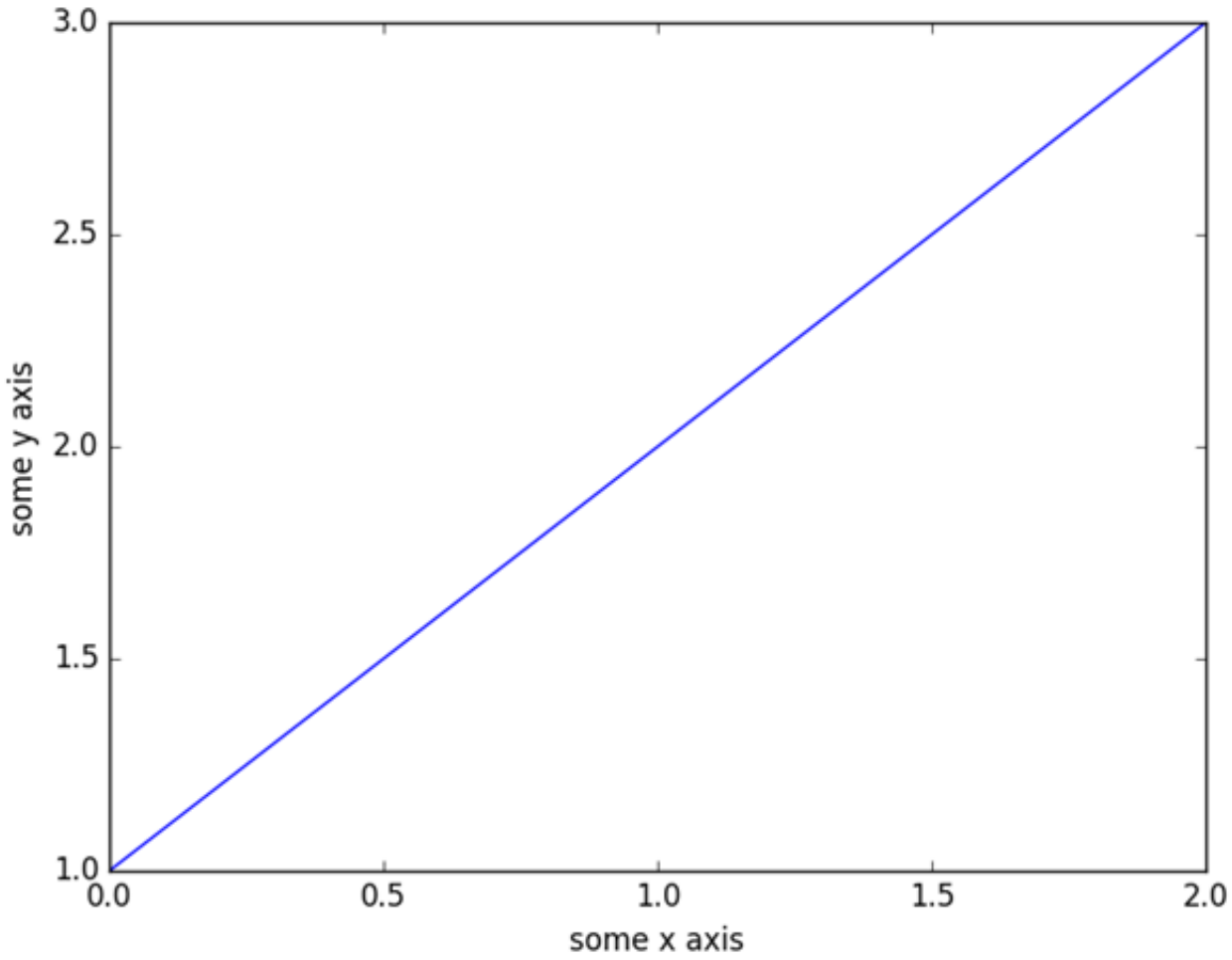
```
Addition: [5 5 5]

Multiplication: [6 6 6]
```

Matplotlib Crash Course

Line Plot

```
# 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 Crash Course

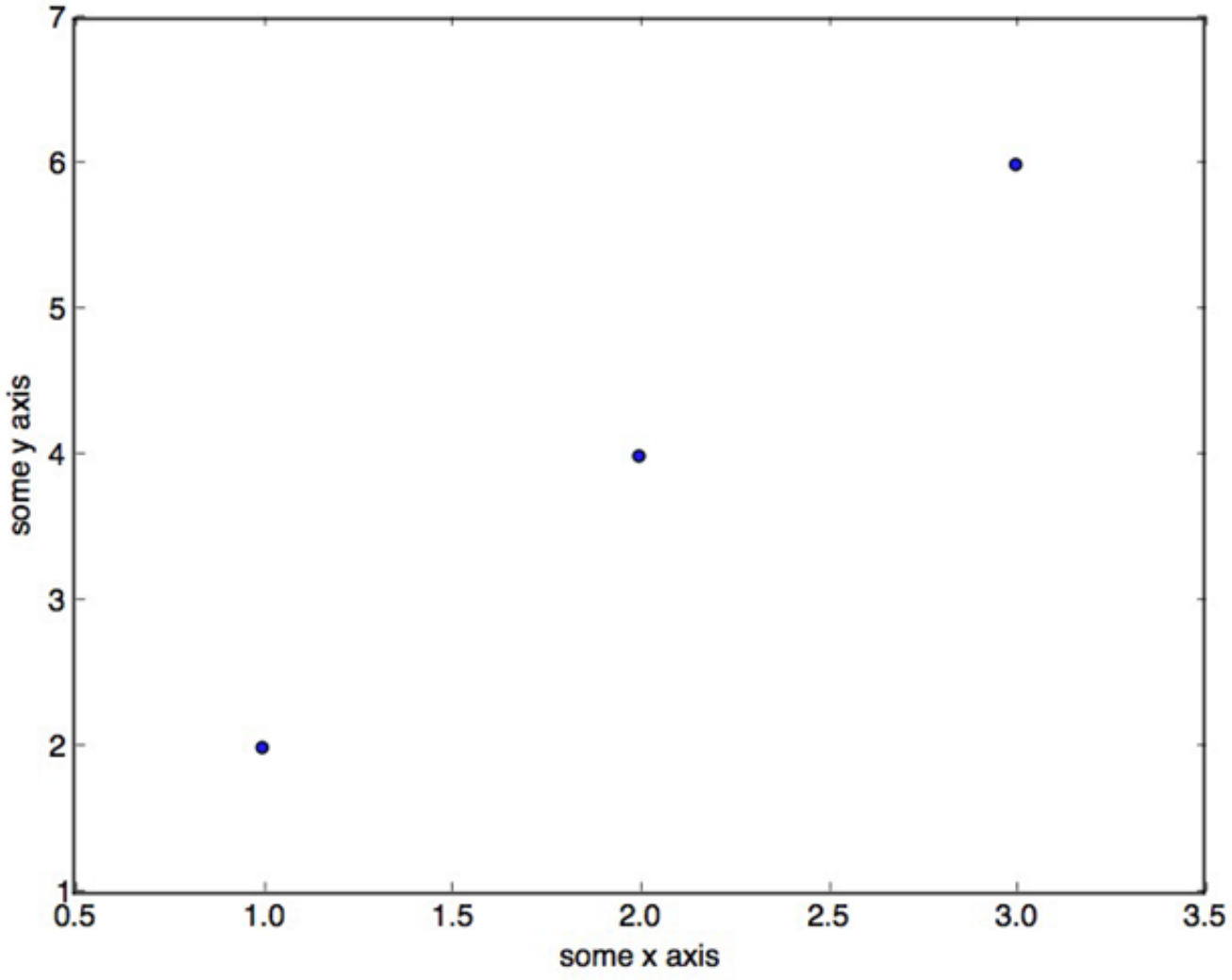


Matplotlib Crash Course

Scatter plot

```
# 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()
```

Matplotlib Crash Course



Pandas Crash Course

```
# 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 Crash Course


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

```
1
1
```

Pandas Crash Course

Data frame

```
# 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)
```

Pandas Crash Course


```
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

DataFrame.
```

Data Loading

Loading Data

- How to load a CSV file.
- How to convert strings from a file to floating point numbers.
- How to convert class values from a file to integers.

CSV

- The standard file format for small datasets is Comma Separated Values or CSV.
- In its simplest form, CSV files are comprised of rows of data.
- Each row is divided into columns using a comma
- In this lab, we are going to practice loading two different, standard machine learning in CSV format.

The Pima Indians Dataset

- In this Section we will use the Pima Indians Diabetes Dataset.
- This dataset involves the prediction of the onset of diabetes within 5 years.
  - The baseline performance on the problem is approximately 65%.
- Download the dataset and add it to your current working directory with the name pima-indians-diabetes.csv.
  - Download: https://goo.gl/2tMFne
- Open and evaluate the data

Iris Flower Species Dataset

- In this section we will also use the Iris Flower Species Dataset.
- This dataset involves the prediction of iris flower species.
- The baseline performance on the problem is approximately 26%.
- Download the dataset and save it into your current working directory with the filename iris.csv.
  - Download: https://goo.gl/uqoqh7

Loading the Datasets

- This Section is divided into 3 parts:

1. Load a file.
2. Load a file and convert Strings to Floats.
3. Load a file and convert Strings to Integers.

- These steps will provide the foundations you need to handle loading your own data.

Load the CSV File

```
# Load a CSV file
def load_csv(filename):
    file = open(filename, "r")
    lines = reader(file)
    dataset = list(lines)
    return dataset
```

- 6,148,72,35,0,33.6,0.627,50,1
- 1,85,66,29,0,26.6,0.351,31,0
- 8,183,64,0,0,23.3,0.672,32,1
- 1,89,66,23,94,28.1,0.167,21,0
- 0,137,40,35,168,43.1,2.288,33,1

Putting it all together

```
# Example of loading Pima Indians CSV dataset
from csv import reader
# Load a CSV file
def load_csv(filename):
	file = open(filename, "r")
	lines = reader(file)
	dataset = list(lines)
	return dataset
# Load dataset
filename = 'pima-indians-diabetes.csv'
dataset = load_csv(filename)
print('Loaded data file {0} with {1} rows and {2} columns'.format(filename,
len(dataset),
	len(dataset[0])))
```

```
# Example of loading Pima Indians CSV dataset
from csv import reader
# Load a CSV file
def load_csv(filename):
	dataset = list()
	with open(filename, 'r') as file:
		csv_reader = reader(file)
		for row in csv_reader:
			if not row:
				continue
			dataset.append(row)
	return dataset
# Load dataset
filename = 'pima-indians-diabetes.csv'
dataset = load_csv(filename)
print('Loaded data file {0} with {1} rows and {2} columns'.format(filename, len(dataset),
	len(dataset[0])))
```

Convert Strings to Floats

- print(dataset[0])

```
def str_column_to_float(dataset, column):
    for row in dataset:
        row[column] = float(row[column].strip())
```

convert_string_to_float.py (reference file)

```
from csv import reader

def load_csv(filename):
        dataset = list()
        with open(filename, 'r') as file:
                csv_reader = reader(file)
                for row in csv_reader:
                        if not row:
                                continue
                        dataset.append(row)
        return dataset

def str_column_to_float(dataset, column):
        for row in dataset:
                row[column] = float(row[column].strip())
```

Test the Load

```
# Load pima-indians-diabetes dataset
filename = 'pima-indians-diabetes.csv'
dataset = load_csv(filename)
print('Loaded data file {0} with {1} rows and {2}
columns'.format(filename, len(dataset),
    len(dataset[0])))
print(dataset[0])
# convert string columns to float
for i in range(len(dataset[0])):
    str_column_to_float(dataset, i)
print(dataset[0])
```

Output

- Loaded data file pima-indians-diabetes.csv with 768 rows and 9 columns
- ['6', '148', '72', '35', '0', '33.6', '0.627', '50', '1']
- [6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0, 1.0]

Converting Strings to Integers in a Dataset

- The iris flowers dataset is like the Pima Indians dataset, in that the columns contain numeric data.
- The difference is the final column, traditionally used to hold the outcome or value to be predicted for a given row.
- The final column in the iris flowers data is the iris flower species as a string.

Flower Dataset

5.1,3.5,1.4,0.2,Iris-setosa

4.9,3.0,1.4,0.2,Iris-setosa

4.7,3.2,1.3,0.2,Iris-setosa

4.6,3.1,1.5,0.2,Iris-setosa

5.0,3.6,1.4,0.2,Iris-setosa

1. First, we locate all of the unique class values, which happen to be: Iris-setosa,

Iris-versicolor and Iris-virginica.

2. Next, we assign an integer value to each, such as: 0, 1 and 2.

3. Finally, we replace all occurrences of class string values with their corresponding integer values.

```
# Convert string column to integer
def str_column_to_int(dataset, column):
    class_values = [row[column] for row in dataset]
    unique = set(class_values)
    lookup = dict()
    for i, value in enumerate(unique):
        lookup[value] = i
    for row in dataset:
        row[column] = lookup[row[column]]
    return lookup
```

convert_string_to_int.py (reference)

Loaded data file iris.csv with 150 rows and 5 columns

['5.1', '3.5', '1.4', '0.2', 'Iris-setosa']

[5.1, 3.5, 1.4, 0.2, 1]

{'Iris-virginica': 0, 'Iris-setosa': 1, 'Iris-versicolor': 2}

Extensions

- . Detect and remove empty lines at the top or bottom of the le.
- . Detect and handle missing values in a column.
- . Detect and handle rows that do not match expectations for the rest of the le.
- . Support for other delimiters such as pipe (|) or white space.
- . Support more ecient data structures such as arrays.

Review

- In this Section, you discovered how you can load your machine learning data from scratch in Python.
  - . How to load a CSV le into memory.
  - . How to convert string values to oating point values.
  - . How to convert a string class value into an integer encoding.

Further Reading

- . Section 13.1, CSV File Reading and Writing, The Python Standard Library
- https://docs.python.org/2/library/csv.html
- . CSV le format in RFC 4180: Common Format and MIME Type for Comma-Separated Values (CSV)
- https://tools.ietf.org/html/rfc4180

Interpreting Data with Descriptive Statistics

LEARNING VOYAGE

Interpreting Data

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.

Review raw 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)
```

Review raw data

```
   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 data

```
# 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)


(768, 9)
```

Data Type by Attribute

```
# 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)
```

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

Descriptive Statistics

```
# 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)
```


```
       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
768.000 768.000 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


```
# 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)
```

```
class 0
500 1 268
```

Correlations between attributes

```
# 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)
```


```
      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 distribution

```
# 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)
```


```
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
```

Understand Your Data with Visualization

LEARNING VOYAGE

Histograms

```
# Univariate Histograms from matplotlib import pyplot
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)
data.hist()
pyplot.show()
```

Density plots

```
# Univariate Density Plots from matplotlib import pyplot
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)
data.plot(kind='density', subplots=True, layout=(3,3),
sharex=False)
pyplot.show()
```

Box and Whisker Plots

```
# Box and Whisker Plots from matplotlib import pyplot
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)
data.plot(kind='box', subplots=True, layout=(3,3),
sharex=False, sharey=False)

pyplot.show()
```

Multivariate Plots

Correlation Matrix Plot

```
# Correlation Matrix Plot
from matplotlib import pyplot from pandas import read_csv
import numpy
filename ='pima-indians-diabetes.data.csv'
names =
['preg','plas','pres','skin','test','mass','pedi','age','cla
ss']
data = read_csv(filename, names=names)
correlations = data.corr() # plot correlation matrix
fig = pyplot.figure()
ax = fig.add_subplot(111)
```

```
# Correlation Matrix Plot (generic) from matplotlib
import pyplot 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)
correlations = data.corr() # plot correlation matrix
fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
```

Scatter plot matrix

```
# Scatterplot Matrix from matplotlib import pyplot from
pandas import read_csv from pandas.plotting import
scatter_matrix

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

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

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

pyplot.show()
```

Scale Machine Learning Data

What we will cover

- . How to normalize your data from scratch.
- . How to standardize your data from scratch.
- . When to normalize as opposed to standardize data.

Why?

- Many machine learning algorithms expect the scale of the input and even the output data to be equivalent.
- It can help in methods that weight inputs in order to make a prediction, such as in linear regression and logistic regression.
- It is practically required in methods that combine weighted inputs in complex ways such as in artificial neural networks and deep learning.

Dataset

- Pima Indians Dataset

1. Normalize Data.
2. Standardize Data.
3. When to Normalize and Standardize.

Normalizing Data

- Normalization can refer to different techniques depending on context.
- Here, I am using normalization to refer to rescaling an input variable to the range between 0 and 1.
- Normalization requires that you know the minimum and maximum values for each attribute.

```
# Find the min and max values for each column
def dataset_minmax(dataset):
    minmax = list()
    for i in range(len(dataset[0])):
        col_values = [row[i] for row in dataset]
        value_min = min(col_values)
        value_max = max(col_values)
        minmax.append([value_min, value_max])
    return minmax
```

Test with a small dataset

x1 x2

50 30

20 90

```
# Find the min and max values for each column
def dataset_minmax(dataset):
    minmax = list()
    for i in range(len(dataset[0])):
        col_values = [row[i] for row in dataset]
        value_min = min(col_values)
        value_max = max(col_values)
        minmax.append([value_min, value_max])
    return minmax
```

- `# Contrive small dataset`
- `dataset = [[50, 30], [20, 90]]`
- `print(dataset)`
- `# Calculate min and max for each column`
- `minmax = dataset_minmax(dataset)`
- `print(minmax)`

Verify

[[50, 30], [20, 90]]

[[20, 50], [30, 90]]

- Once we have estimates of the maximum and minimum allowed values for each column, we can now normalize the raw data to the range 0 and 1.
- The calculation to normalize a single value for a column is:

$$\text{scaled value} = \frac{value - min}{max - min}$$

```
# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
    for row in dataset:
        for i in range(len(row)):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] -
            minmax[i][0])
```

```
# Contrive small dataset
dataset = [[50, 30], [20, 90]]
print(dataset)
# Calculate min and max for each column
minmax = dataset_minmax(dataset)
print(minmax)
# Normalize columns
normalize_dataset(dataset, minmax)
print(dataset)
```

Validation

- Running this example prints the output below, including the normalized dataset.

[[50, 30], [20, 90]]

[[20, 50], [30, 90]]

[[1, 0], [0, 1]]

normalize_diabetes.py

- We can combine this code with code for loading a CSV dataset and load and normalize the Pima Indians Diabetes dataset.
- The example first loads the dataset and converts the values for each column from string to floating point values.
- The minimum and maximum values for each column are estimated from the dataset, and finally, the values in the dataset are normalized.

- Loaded data file pima-indians-diabetes.csv with 768 rows and 9 columns
- BEFORE NORMALIZATION

[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0, 1.0]

- AFTER NORMALIZATION

[0.35294117647058826, 0.7437185929648241, 0.5901639344262295, 0.35353535353535354, 0.0, 0.5007451564828614, 0.2344150298889837, 0.48333333333333334, 1.0]

Standardize Data

- Standardization is a rescaling technique that refers to centering the distribution of the data on the value 0 and the standard deviation to the value 1.
- Together, the mean and the standard deviation can be used to summarize a normal distribution, also called the Gaussian distribution or bell curve.

$$\text{mean} = \frac{\sum_{i=1}^{n} values_i}{count(values)}$$

Define the Mean

```
# calculate column means
def column_means(dataset):
    means = [0 for i in range(len(dataset[0]))]
    for i in range(len(dataset[0])):
        col_values = [row[i] for row in dataset]
        means[i] = sum(col_values) / float(len(dataset))
    return means
```

Standard Deviation

- The standard deviation describes the average spread of values from the mean. It can be calculated as the square root of the sum of the squared difference between each value and the mean and dividing by the number of values minus 1.

$$\text{standard deviation} = \sqrt{\frac{\sum_{i=1}^{n}(value_i - mean)^2}{count(values) - 1}}$$

```
# calculate column standard deviations
def column_stdevs(dataset, means):
    stdevs = [0 for i in range(len(dataset[0]))]
    for i in range(len(dataset[0])):
        variance = [pow(row[i]-means[i], 2) for row in dataset]
        stdevs[i] = sum(variance)
    stdevs = [sqrt(x/(float(len(dataset)-1))) for x in stdevs]
    return stdevs
```

Validate

x1 x2

50 30

20 90

30 50

statistics_contrived_dataset.py

```
[[50, 30], [20, 90], [30, 50]]
[33.333333333333336, 56.666666666666664]
[15.275252316519467, 30.550504633038933]
```

That leads us to here…

- Once the summary statistics are calculated, we can easily standardize the values in each column.
- The calculation to standardize a given value is as follows:

$$\text{standardized\_value}_i = \frac{value_i - mean}{stdev}$$

standardize_contrived_dataset.py

```
# standardize dataset
def standardize_dataset(dataset, means, stdevs):
    for row in dataset:
        for i in range(len(row)):
            row[i] = (row[i] - means[i]) / stdevs[i]
```

Validate

[[50, 30], [20, 90], [30, 50]]

[33.333333333333336, 56.666666666666664]

[15.275252316519467, 30.550504633038933]

[[1.0910894511799618, -0.8728715609439694], [-0.8728715609439697, 1.0910894511799962],

[-0.2182178902359253, -0.2182178902359923]]

standardize_diabetes.py

- Loaded data file pima-indians-diabetes.csv with 768 rows and 9 columns

[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0, 1.0]

[0.6395304921176576, 0.8477713205896718, 0.1495432985295429 6, 0.9066790623472505,

-0.6924393247241 29, 0.2038799072674717, 0.4681868702297 98, 1.4250667195933604,

1.3650063669598067]

When to Standardize and When to Normalize

- Standardization is a scaling technique that assumes your data conforms to a normal distribution.
- If a given data attribute is normal or close to normal, this is probably the scaling method to use.
- It is good practice to record the summary statistics used in the standardization process so that you can apply them when standardizing data in the future that you may want to use with your model.
- Normalization is a scaling technique that does not assume any specific distribution.

Other Things

- . Normalization that permits a congurable range, such as -1 to 1 and more.
- . Standardization that permits a congurable spread, such as 1, 2 or more standard deviations from the mean.
- . Exponential transforms such as logarithm, square root and exponents.
- . Power transforms such as Box-Cox for xing the skew in normally distributed data.

Summary

- . How to normalize data from scratch.
- . How to standardize data from scratch.
- . When to use normalization or standardization on your data.

- . Chapter 3 Data Pre-processing, page 27, Applied Predictive Modeling, 2013
- http://amzn.to/2e3lNXF