CSE 519: Data Science

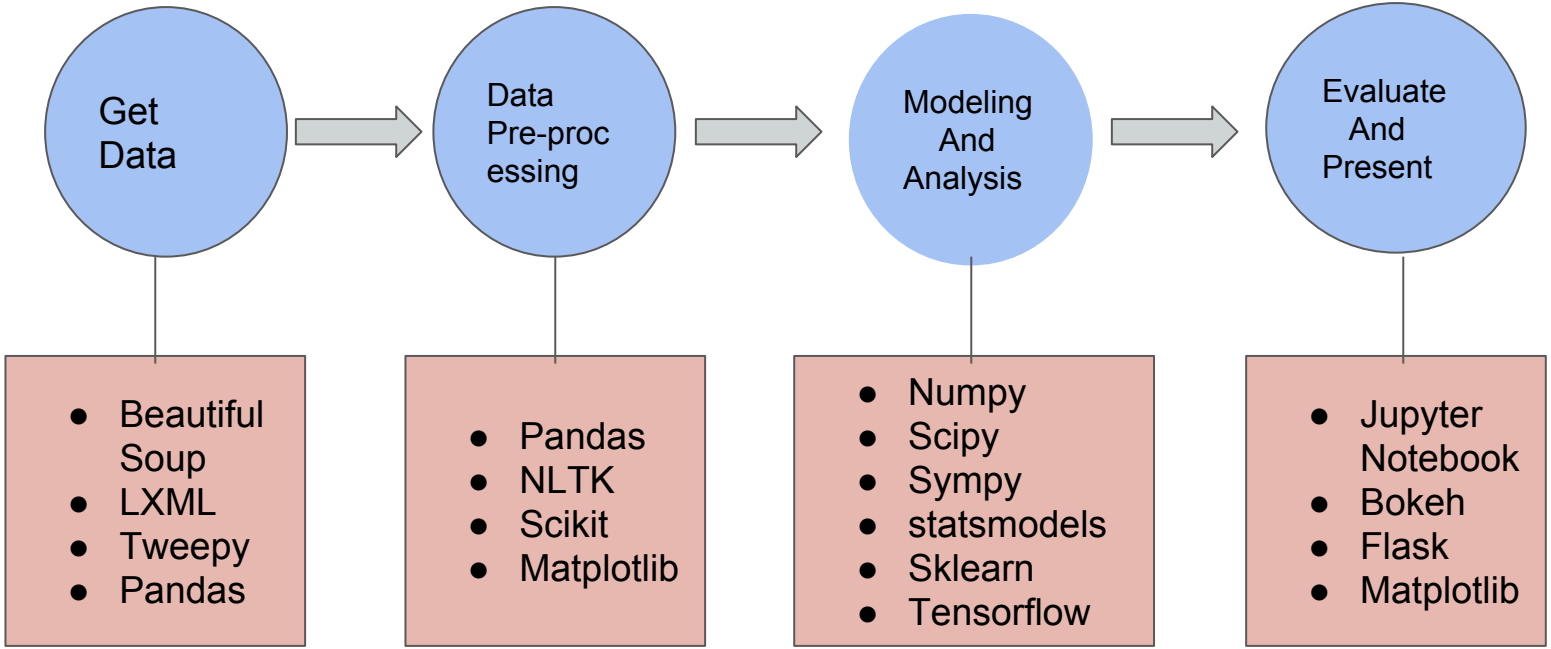
Steven Skiena
Stony Brook University

Lecture 3: Python for Data Science I

Lecture Goals

- Overview of how to use Python for Data Science.
- Not a Python 101
  - Assume you already know Python or are willing to learn.
  - See http://www.learnpython.org/
- Learn by example
  - Demonstrate by solving actual problems.

Data Science with Python



Step 1: Get Data

- Several Python packages to easily scrape and download data
  - HTML and XML: Beautiful Soup
  - Twitter: Tweepy
  - Reddit: PRAW
  - Wikipedia Processing: wikipedia
  - Stackoverflow - PyStackExchange

Example

- Scrape IMDB and get actor names and characters in Shawshank Redemption

Cast

Edit

Cast overview, first billed only:

Tim Robbins	...	Andy Dufresne
Morgan Freeman	...	Ellis Boyd 'Red' Redding
Bob Gunton	...	Warden Norton
William Sadler	...	Heywood
Clancy Brown	...	Captain Hadley
Gil Bellows	...	Tommy
Mark Rolston	...	Bogs Diamond
James Whitmore	...	Brooks Hatlen

Sample Code using Beautiful Soup

```
link = 'http://www.imdb.com/title/tt0111161/?ref_=nv_sr_1'
movie_page = requests.get(link)

# Strain the cast_list table from the movie_page
soup = BeautifulSoup(movie_page.content)

# Iterate through rows and extract the name and character
# Remember that some rows might not be a row of interest (e.g., a blank
# row for spacing the layout). Therefore, we need to use a try-except
# block to make sure we capture only the rows we want, without python
# complaining.
for row in soup.find_all('tr'):
    try:
        actor = clean_text(row.find(itemprop='name').text)
        character = clean_text(row.find(class_='character').text)

        print '\t'.join([actor, character])

    except AttributeError:
        pass
```

See https://raw.githubusercontent.com/5harad/datascience/master/webscraping/01-bs/get_cast_from_movie.py for full code

Using Pandas to load CSV or Tables

- Pandas is spreadsheet software for Python
  - A table is called a DataFrame
  - A 1-D array of numbers is called a Series
- Important Features
  - Easily load CSV, TSV files
  - Can easily load data in chunks if needed.
  - Support group-by, indexing, selection, merge operations
  - Data Analysis Functions like mean, median

Example

- Load data containing height in centimeters of boys and girls through ages 2, 9,18 years.

```
import pandas as pd
df = pd.read_csv('children_heights.csv', '\t')
```

	Boys_2	Boys_9	Boys_18	Girls_2	Girls_9	Girls_18
0	90.2	139.4	179.0	83.8	136.5	169.6
1	91.4	144.3	195.1	86.2	137.0	166.8
2	86.4	136.5	183.7	85.1	129.0	157.1
3	87.6	135.4	178.7	88.6	139.4	181.1
4	86.7	128.9	171.5	83.0	125.6	158.4
5	88.1	136.0	181.8	88.9	137.1	165.6

Step 2: Preprocessing

- Raw data might need to be pre-processed
- Specialized packages might need to be used based on type of data
  - Numeric Data: numpy, pandas
  - Text Data: NLTK, spaCy
  - Image Data: scikit-image
- Preprocess the data only once! Don’t waste CPU cycles doing it each time!

Example: Textual Data

- Split text into sentences

```
import nltk
sents = 'What is this life if full of care we have no time to stand and stare! A thing of beauty is a joy forever.'
nltk.sent_tokenize(sents)
```

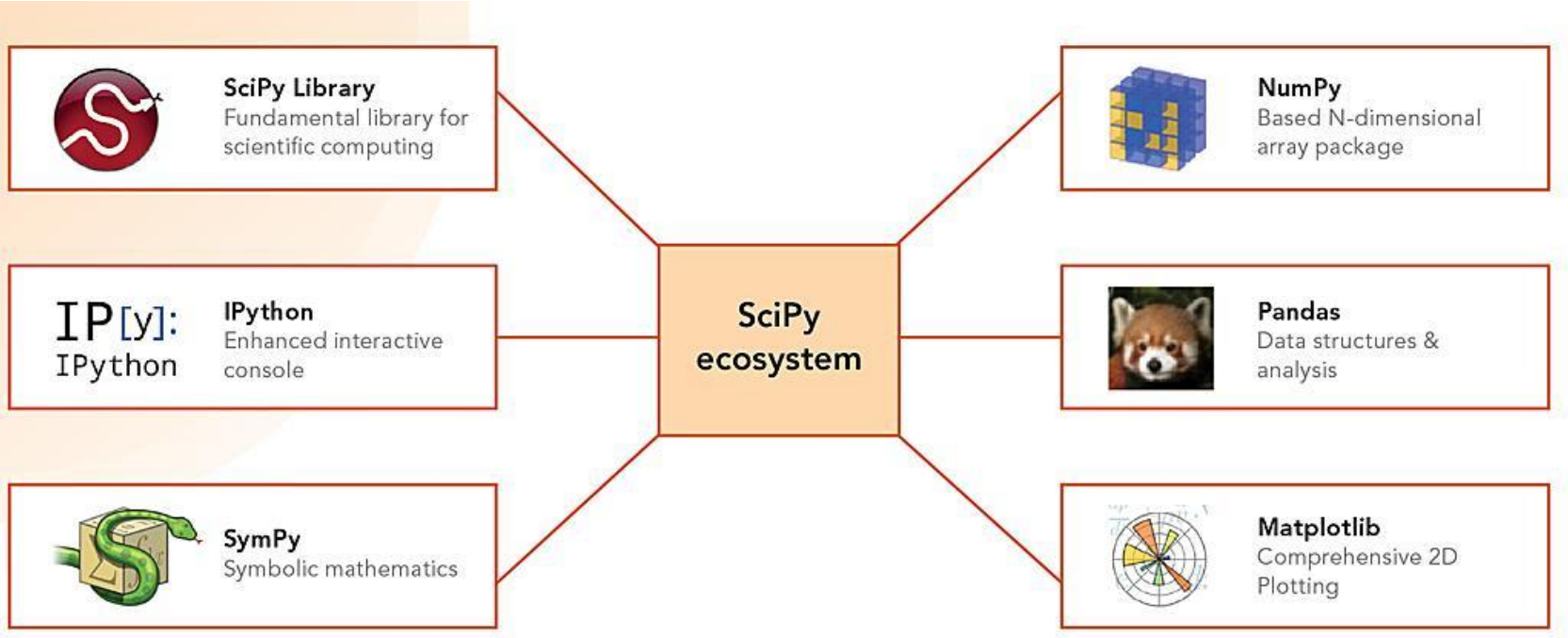
- Output

```
['What is this life if full of care we have no time to stand and stare!',
 'A thing of beauty is a joy forever.']
```

Step 3: Modeling and Analysis

- Build or infer a mathematical model for the problem
- The Scientific Python (Scipy) stack is most useful in this step
- Several Distributions (pre-packaged) available:
  - Enthought
  - Anaconda

Scientific Python Ecosystem



http://www.esri.com/~/media/Images/Content/news/arcuser/0115/scipy_2-lg.jpg

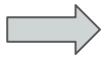
Numpy overview

- Provides a fast, efficient implementation of N-d array (ndarray)
- Several statistical operations supported:np.mean, np.std, np.median
- Supports linear algebra operations: dot product, cross product
- Fast Fourier Transforms, Signal Processing operations also supported

Using Numpy Example

- Invert the matrix $\begin{pmatrix} 2 & 3 \\ 2 & 2 \end{pmatrix}$
- Sample code using Numpy

```
import numpy as np
# Create the matrix we want to invert
A = np.array([[2,3],[2,2]])
# Invert the matrix using linalg.inv
AI = np.linalg.inv(A)
# Print the inverse out
```



$$\begin{pmatrix} -1 & \frac{3}{2} \\ 1 & -1 \end{pmatrix}$$

Scipy overview

- Package containing extensive functionality for use by scientists
  - Linear Algebra (scipy.linalg)
  - Optimization (scipy.optimize)
  - Statistics (scipy.stats)
  - Signal Processing: (scipy.signal)
  - Special functions (like Gamma): (scipy.special)

Using SciPy example

- A car's velocity in (mph) at time t is given by: 25 + 10t. Find the distance in miles covered by the car in 3 hours.
- Solution: 120 miles

```
import scipy

# Velocity of car
def velocity(t):
    return 25 + 10.0*t

# Integrate velocity from from 0 to 3
distance = scipy.integrate.quad(velocity, 0, 3)

print "Distance", distance
```

scikit-learn Overview

- Scikit-learn is the definitive low-level toolkit for “classical” machine learning in Python
- Supports regression, classification, clustering and dimensionality reduction
- Many models: SVM, Linear Regression, Logistic Regression
- Catch: Understand how these algorithms work before you apply them

Low-level Machine Learning Library

- Libraries for low-level computation on data flow graphs: Tensorflow / PyTorch / Chainer
- More flexible: develop your own machine learning models (especially neural network models)

Step 4: Evaluate and Present

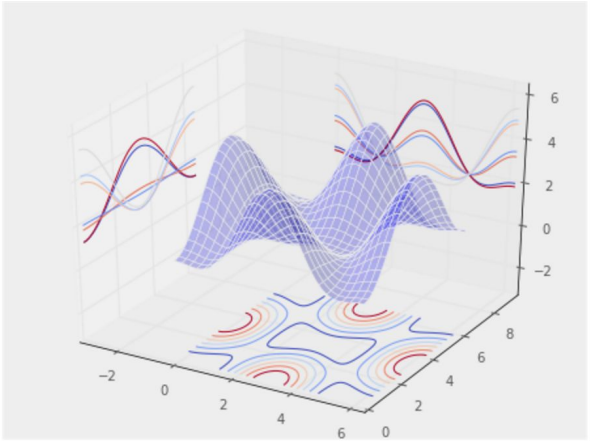
- Jupyter notebook: interactive Python console runs in a web browser
  - Highly recommended!
- Matplotlib and Seaborn for data visualization
- Flask for lightweight Web framework

Visualization in Python: Matplotlib

- Matplotlib is basic plotting library in Python
- Can easily create figures and manipulate them
- Support for:
  - Scatter plots
  - Charts
  - Bar Charts, Pie Charts
  - Box and Whisker Plots
  - Lines

Example Visualization

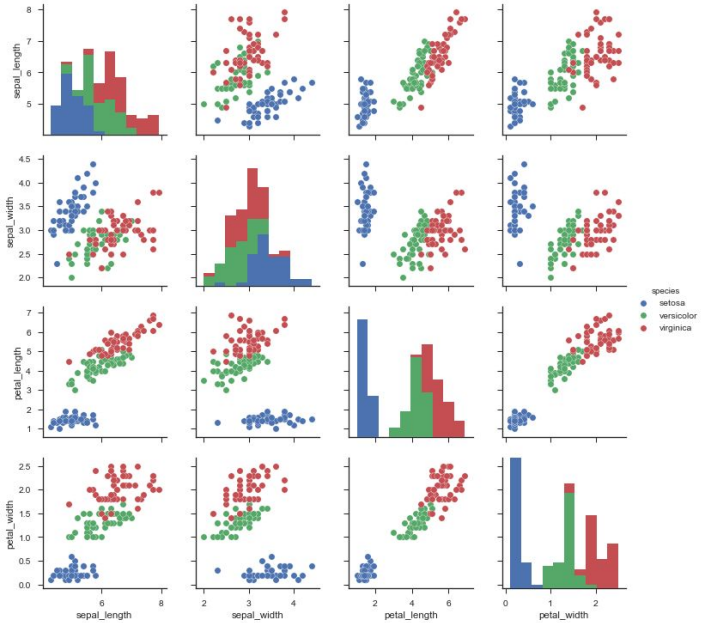
```
from mpl_toolkits.mplot3d.axes3d import Axes3D

alpha = 0.7
phi_ext = 2 * np.pi * 0.5

def flux_qubit_potential(phi_m, phi_p):
    return 2 + alpha - 2 * np.cos(phi_p)*np.cos(phi_m) - alpha * np.cos(phi_ext - 2*phi_p)

phi_m = np.linspace(0, 2*np.pi, 100)
phi_p = np.linspace(0, 2*np.pi, 100)
X,Y = np.meshgrid(phi_p, phi_m)
Z = flux_qubit_potential(X, Y).T

fig = plt.figure(figsize=(8,6))

ax = fig.add_subplot(1,1,1, projection='3d')

ax.plot_surface(X, Y, Z, rstride=4, cstride=4, alpha=0.25)
cset = ax.contour(X, Y, Z, zdir='z', offset=-np.pi, cmap=plt.cm.coolwarm)
cset = ax.contour(X, Y, Z, zdir='x', offset=-np.pi, cmap=plt.cm.coolwarm)
cset = ax.contour(X, Y, Z, zdir='y', offset=3*np.pi, cmap=plt.cm.coolwarm)

ax.set_xlim3d(-np.pi, 2*np.pi);
ax.set_ylim3d(0, 3*np.pi);
ax.set_zlim3d(-np.pi, 2*np.pi);
```



Advanced Visualization: Seaborn

- Built upon Matplotlib with a high-level interface
- With a single line of code

```
import seaborn as sns
sns.set(style="ticks")

df = sns.load_dataset("iris")
sns.pairplot(df, hue="species")
```

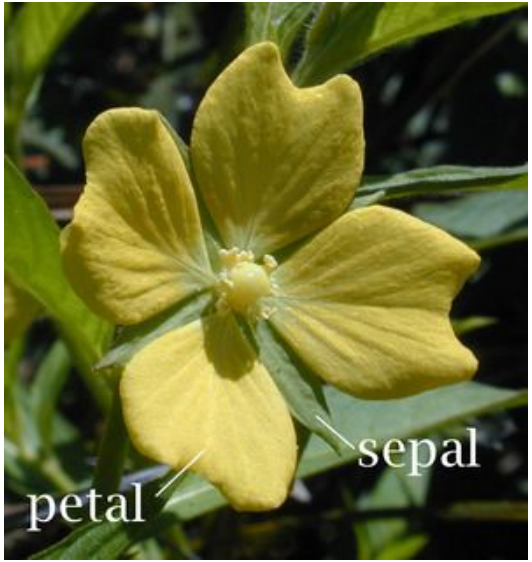


An Example: Classify Iris Flowers

- Derived from an example given by Randal S. Olson: http://www.randalolson.com/, licensed under CC BY 4.0
- Goal: take four measurements of the flowers and identifies the species based on those measurement
- The measurements (features): sepal length, sepal width, petal length, and petal width

An Example: Classify Iris Flowers

- These measurements come from hand-measurements by field researchers



Goal: Identify Flower Type

Iris setosa

Iris versicolor

Iris virginica

Metric for success: classification accuracy

Step 1: Checking the Data

- Read the data into a pandas dataframe

```
import pandas as pd

iris_data = pd.read_csv('iris-data.csv')
iris_data.head()
```

	sepal_length_cm	sepal_width_cm	petal_length_cm	petal_width_cm	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Describe Data

- Identify missing values

```
iris_data = pd.read_csv('iris-data.csv', na_values=['NA'])
```

- Get summary statistics

```
iris_data.describe()
```

	sepal_length_cm	sepal_width_cm	petal_length_cm	petal_width_cm
count	150.000000	150.000000	150.000000	145.000000
mean	5.644627	3.054667	3.758667	1.236552
std	1.312781	0.433123	1.764420	0.755058
min	0.055000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.400000
50%	5.700000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Create a Scatterplot Matrix

- Use the Seaborn library for plotting

```
import seaborn as sb
```

- Use pairplot to plot the distribution of each feature and correlation between different features

Create a Scatterplot Matrix

```
sb.pairplot(iris_data.dropna(), hue='class')
```

```
<seaborn.axisgrid.PairGrid at 0x109668cf8>
```

Learn from the Plot

- There are five classes when there should only be three
- There are some outliers that may be erroneous: several *sepal_length_cm* entries for *Iris-versicolor* are near-zero for some reason

Step 2: Tidying the Data

- Merge classes:

```
iris_data.loc[iris_data['class'] == 'versicolor', 'class'] = 'Iris-versicolor'
iris_data.loc[iris_data['class'] == 'Iris-setossa', 'class'] = 'Iris-setosa'

iris_data['class'].unique()
```

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

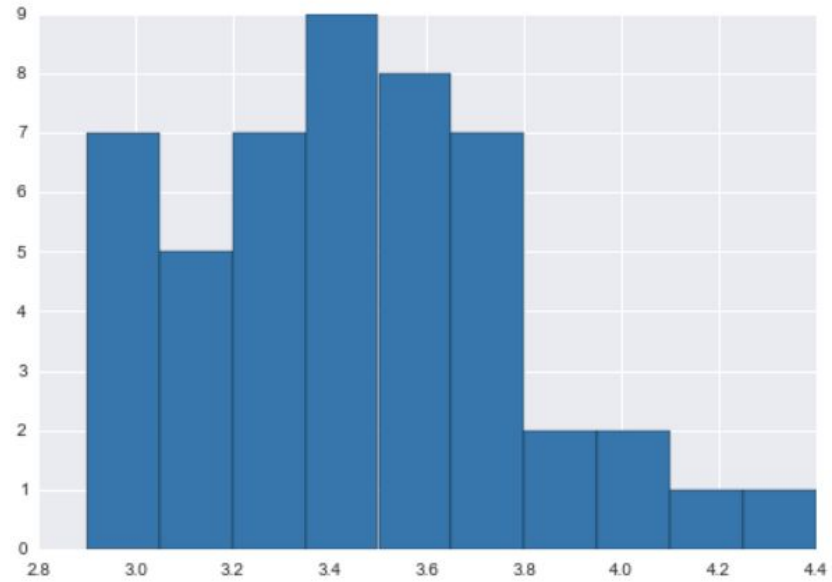
Remove Erroneous Outliers

- Let’s say our field researchers know that it's impossible for *Iris-setosa* to have a sepal width below 2.5 cm...

Remove Erroneous Outliers

```
# This line drops any 'Iris-setosa' rows with a separal width less than 2.5 cm
iris_data = iris_data.loc[(iris_data['class'] != 'Iris-setosa') | (iris_data['sepal_width_cm'] >= 2.5)]
iris_data.loc[iris_data['class'] == 'Iris-setosa', 'sepal_width_cm'].hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10dac0ef0>
```



Save the tidied data

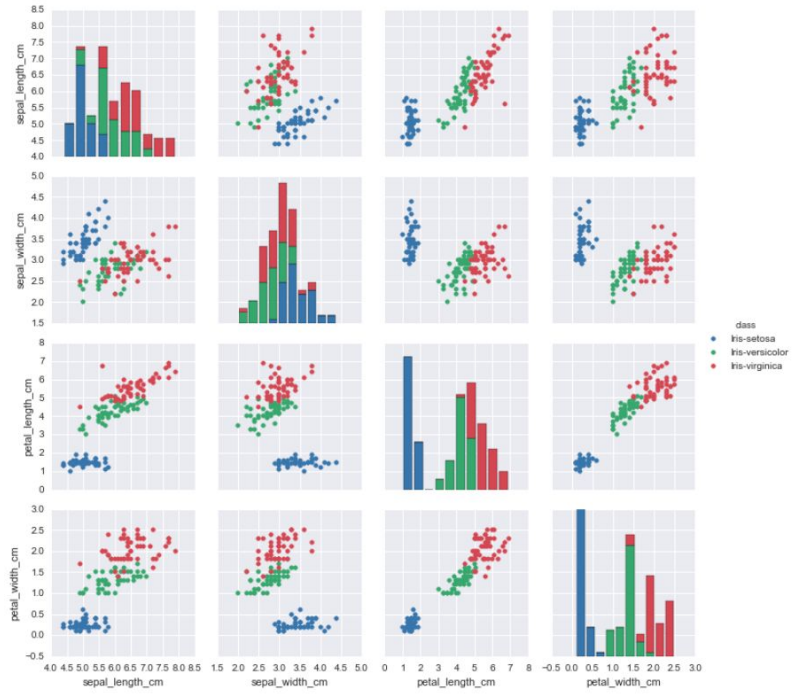
- Important, since we do not want to repeat this tidying process every time we work with this dataset

```
iris_data.to_csv('iris-data-clean.csv', index=False)

iris_data_clean = pd.read_csv('iris-data-clean.csv')
```

Scatterplot matrix for the clean data

```
sb.pairplot(iris_data_clean, hue='class')
```

```
<seaborn.axisgrid.PairGrid at 0x10ea45630>
```



Step 3: Classification

- Get all input features and classes from the clean dataset

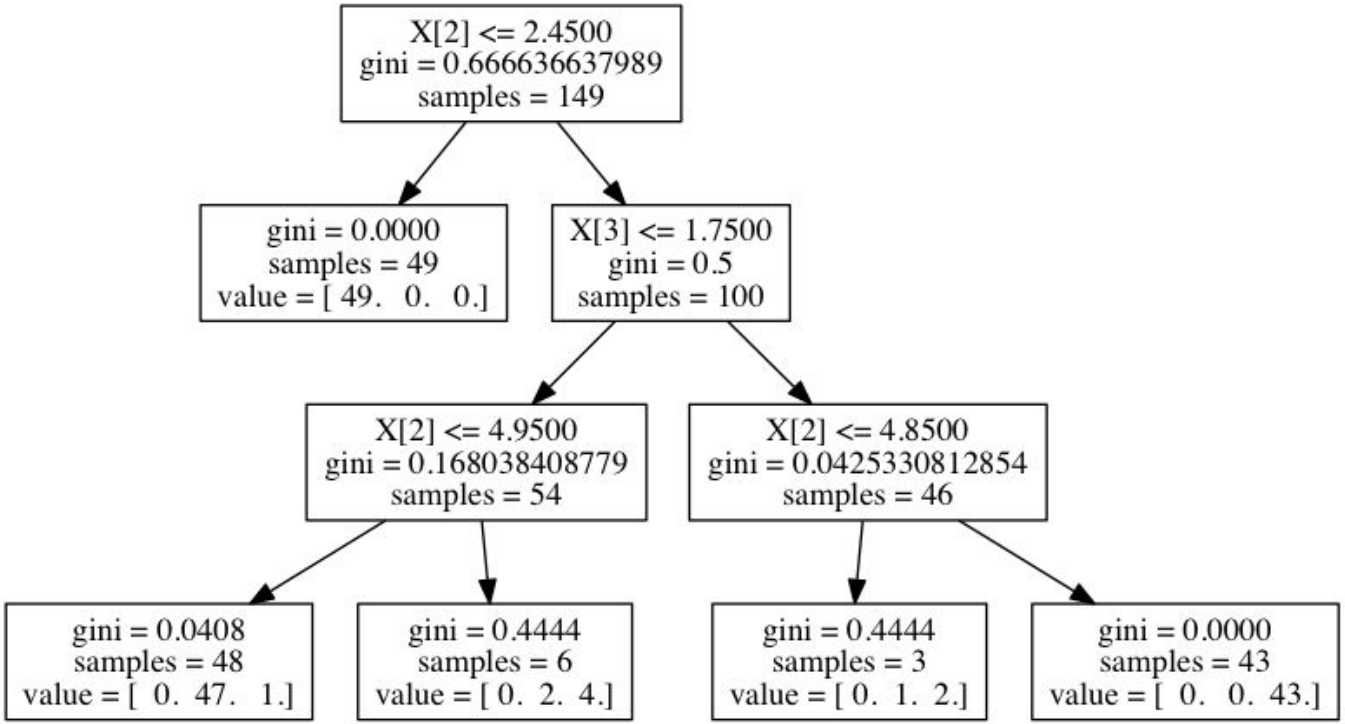
```
iris_data_clean = pd.read_csv('iris-data-clean.csv')

# We're using all four measurements as inputs
# Note that scikit-learn expects each entry to be a list of values, e.g.,
# [ [val1, val2, val3],
#   [val1, val2, val3],
#   ... ]
# such that our input data set is represented as a list of lists

# We can extract the data in this format from pandas like this:
all_inputs = iris_data_clean[['sepal_length_cm', 'sepal_width_cm',
                              'petal_length_cm', 'petal_width_cm']].values

# Similarly, we can extract the classes
all_classes = iris_data_clean['class'].values

# Make sure that you don't mix up the order of the entries
# all_inputs[5] inputs should correspond to the class in all_classes[5]

# Here's what a subset of our inputs looks like:
all_inputs[:5]
```

```
array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2],
       [ 4.7,  3.2,  1.3,  0.2],
       [ 4.6,  3.1,  1.5,  0.2],
       [ 5. ,  3.6,  1.4,  0.2]])
```

Split into training and testing sets

```
from sklearn.cross_validation import train_test_split

(training_inputs,
 testing_inputs,
 training_classes,
 testing_classes) = train_test_split(all_inputs, all_classes, train_size=0.75, random_state=1)
```

Build a Decision Tree Classifier



Build a Decision Tree Classifier

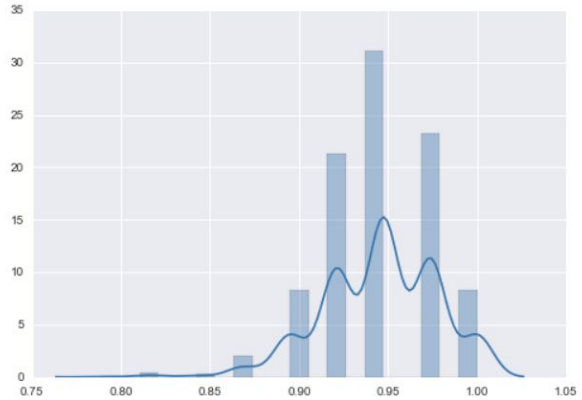
```
from sklearn.tree import DecisionTreeClassifier

# Create the classifier
decision_tree_classifier = DecisionTreeClassifier()

# Train the classifier on the training set
decision_tree_classifier.fit(training_inputs, training_classes)

# Validate the classifier on the testing set using classification accuracy
decision_tree_classifier.score(testing_inputs, testing_classes)
```

```
0.9736842105263158
```

Stability of the result

```
model_accuracies = []

for repetition in range(1000):
    (training_inputs,
     testing_inputs,
     training_classes,
     testing_classes) = train_test_split(all_inputs, all_classes, train_size=0.75)

    decision_tree_classifier = DecisionTreeClassifier()
    decision_tree_classifier.fit(training_inputs, training_classes)
    classifier_accuracy = decision_tree_classifier.score(testing_inputs, testing_classes)
    model_accuracies.append(classifier_accuracy)

sb.distplot(model_accuracies)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11164c128>
```



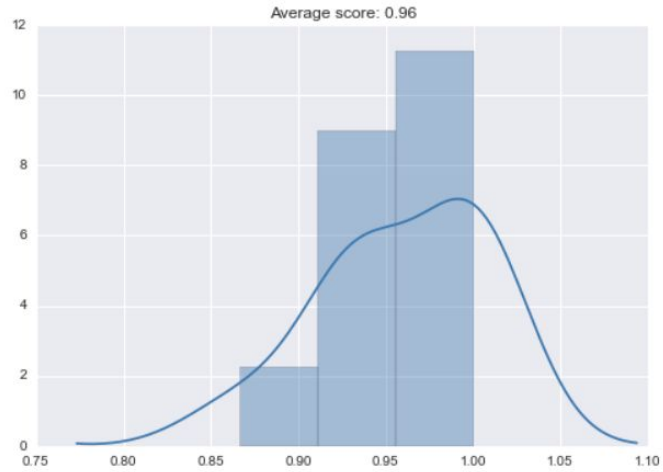
K-fold cross-validation

- Split the original data set into k subsets, use one of the subsets for test and the rest for training
- Repeat this process for k times such that each subset is used as the testing set once

K-fold cross-validation

```
from sklearn.cross_validation import cross_val_score

decision_tree_classifier = DecisionTreeClassifier()

# cross_val_score returns a list of the scores, which we can visualize
# to get a reasonable estimate of our classifier's performance
cv_scores = cross_val_score(decision_tree_classifier, all_inputs, all_classes, cv=10)
sb.distplot(cv_scores)
plt.title('Average score: {}'.format(np.mean(cv_scores)))
```

```
<matplotlib.text.Text at 0x1138e2278>
```



The resulting classifier

```
import sklearn.tree as tree
from sklearn.externals.six import StringIO

with open('iris_dtc.dot', 'w') as out_file:
    out_file = tree.export_graphviz(decision_tree_classifier, out_file=out_file)
```