Machine Learning: Data Foundations + Algorithms & Applications

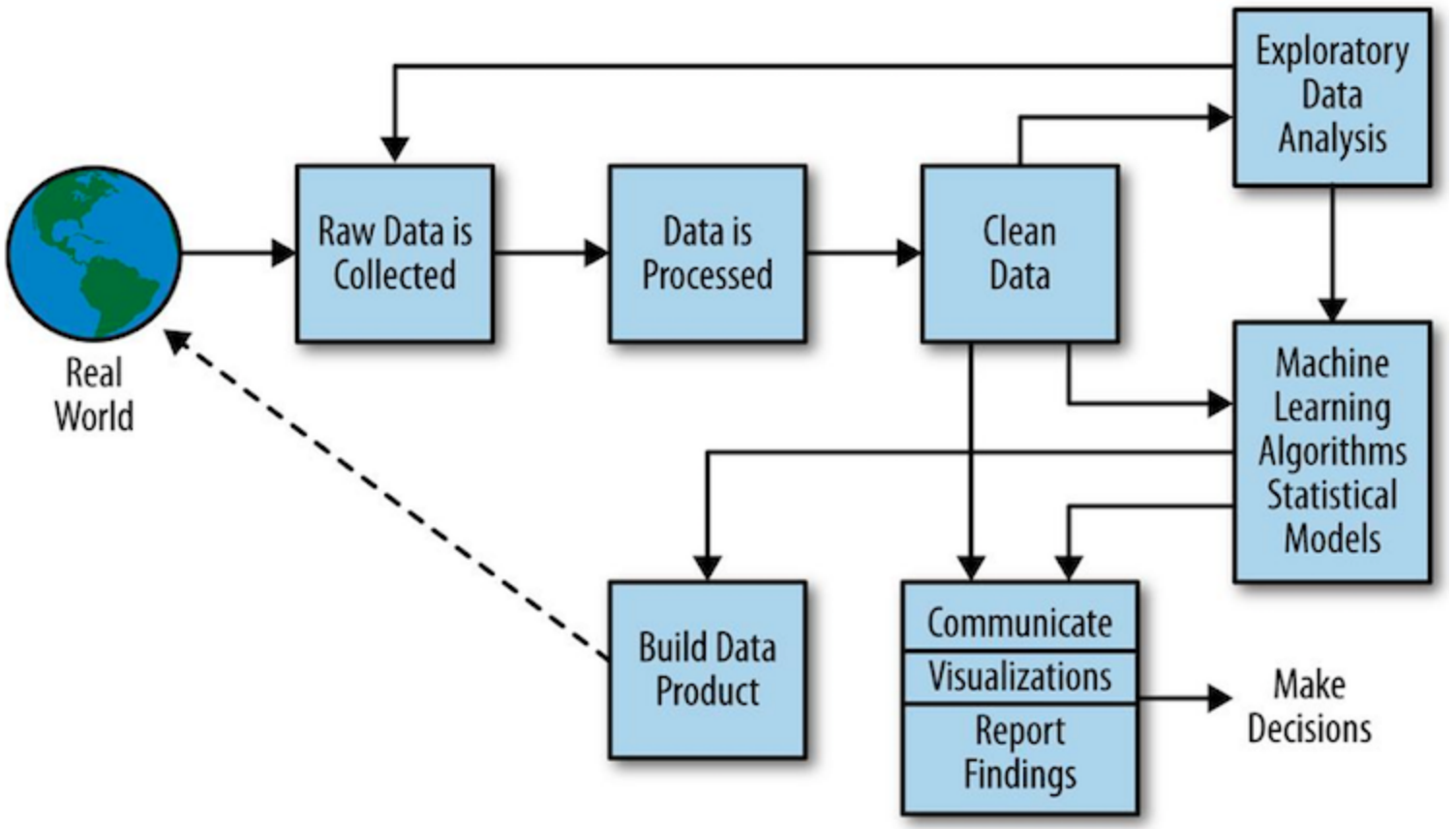
Day 4

Agenda - Day 2

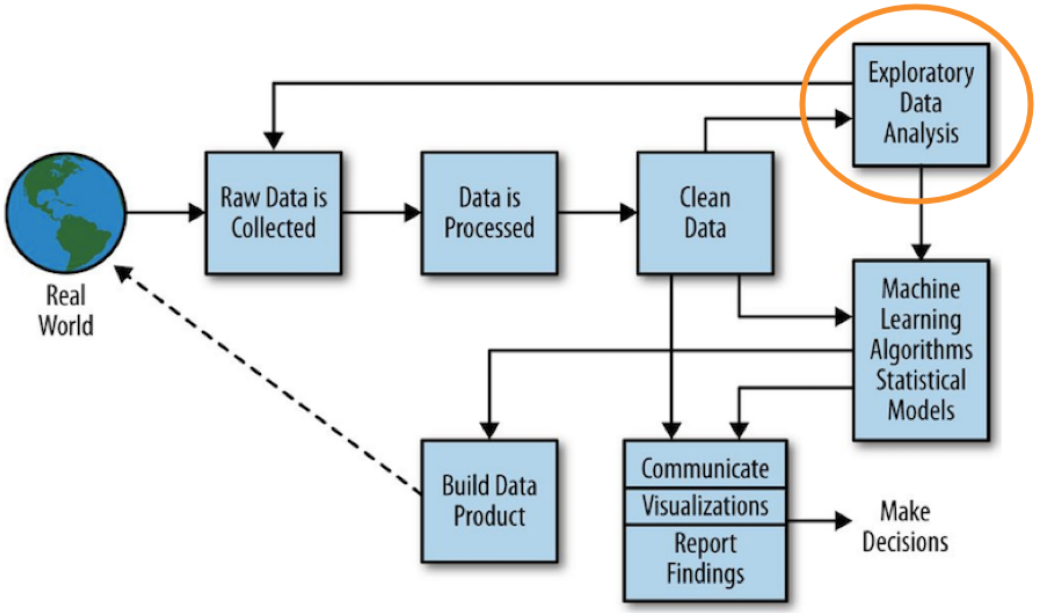
- Data Science
- Data Cleansing
- Data Visualization
- Data Exploration
- Model Deeper Dives

Data Science

Data Science Pipeline



"Doing Data Science"



"Doing Data Science"

Machine Learning Checklist

1. Frame the problem and look at the big picture.
2. Get the data.
3. Explore the data to gain insights.
4. Prepare the data to better expose the underlying data patterns to Machine Learning algorithms.
5. Explore many different models and short-list the best ones.
6. Fine-tune your models and combine them into a great solution.
7. Present your solution.
8. Launch

Frame the problem and look at the big picture

1. Define the objective in business terms.
2. How will your solution be used?
3. What are the current solutions/workarounds (if any)?
4. How should you frame this problem (supervised/unsupervised, online/offline, etc.)?
5. How should performance be measured?
6. Is the performance measure aligned with the business objective?
7. What would be the minimum performance needed to reach the business objective?
8. What are comparable problems? Can you reuse experience or tools?
9. Is human expertise available?
10. How would you solve the problem manually?
11. List the assumptions you (or others) have made so far.
12. Verify the assumptions if possible.

Get the Data

1. List the data you need and how much you need.
2. Find and document where you can get that data.
3. Check how much space it will take.
4. Check legal obligations, and get authorization if necessary.
5. Get access authorizations.
6. Create a workspace (with enough storage space).
7. Get the data.
8. Convert the data to a format you can easily manipulate (without changing the data itself).
9. Ensure sensitive information is deleted or protected (e.g. anonymized).
10. Check the size and type of data (time series, sample, geographical, etc.).
11. Sample a test set, put it aside, and never look at it (no data snooping!).

Explore the Data

1. Create a copy of the data for exploration (sampling it down to a manageable size if necessary).
2. Create a Jupyter notebook to keep a record of your data exploration.
3. Study each attribute and its characteristics.
4. For supervised learning tasks, identify the target attribute(s).
5. Visualize the data.
6. Study the correlations between attributes.
7. Study how you would solve the problem manually.
8. Identify the promising transformation you may want to apply.
9. Identify extra data that would be useful.
10. Document what you have learned.

Prepare the Data

1. Clean the data.
2. Feature selection (if needed).
3. Feature engineering (where appropriate).
4. Feature scaling.

Short-List Promising Models

1. Train many quick and dirty models from different categories (e.g., linear, naive Bayes, SVM, Random Forests, etc.) using standard parameters.
2. Measure and compare their performance.
3. Analyze the most significant variables for each algorithm.
4. Analyze the types of errors the models make.
5. Have a quick round of feature selection and engineering.
6. Repeat the steps 1-5 a few times.
7. Short-list the top 3-5 most promising models, preferring models that make different types of errors.

Fine-Tune the System

1. Fine-tune the hyperparameters using cross-validation.
2. Try Ensemble methods. Combining your best models will often perform better than running them individually.
3. Once you are confident about your final model, measure its performance on the test set to estimate the generalization error.

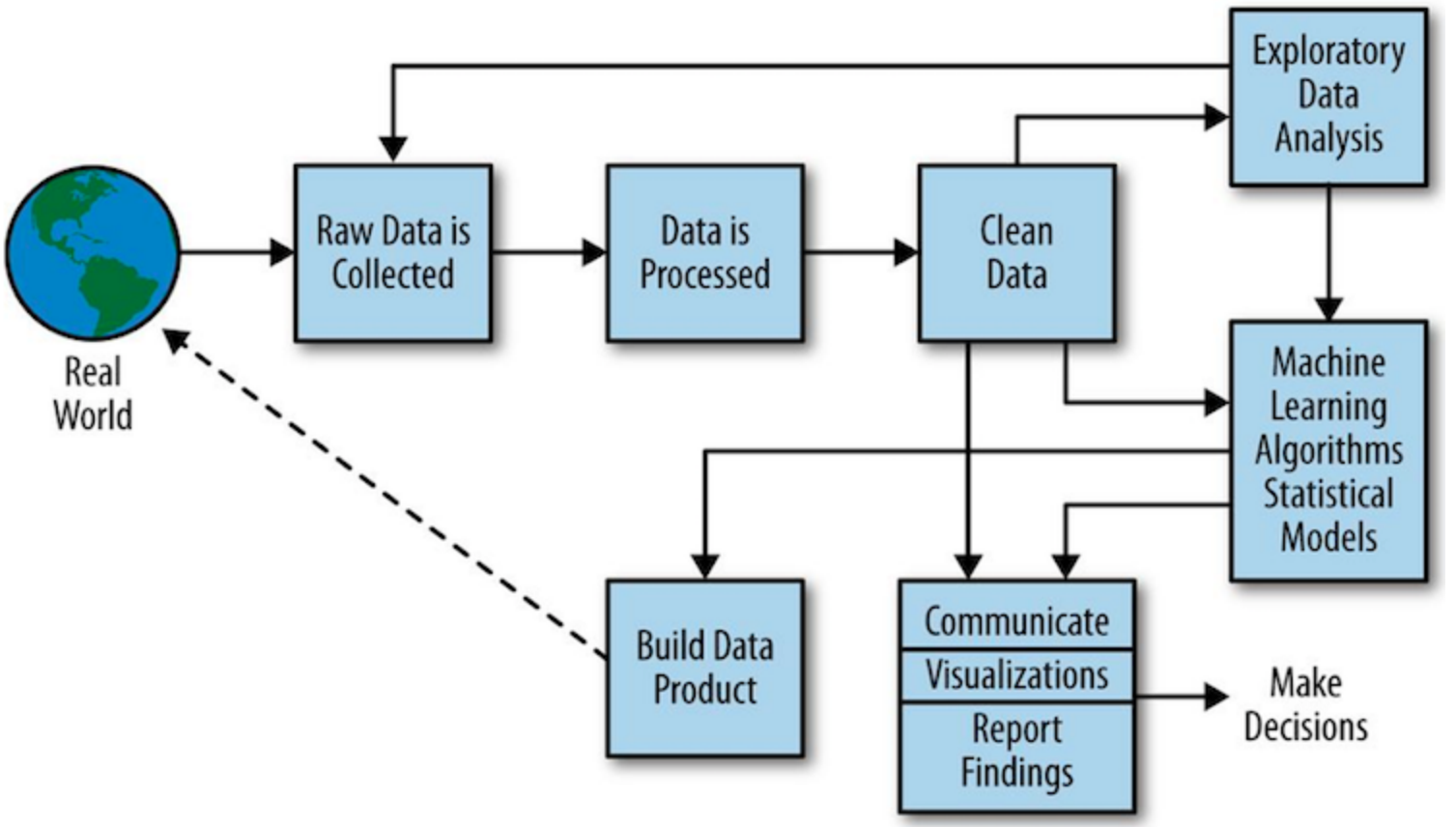
Present Your Solution

1. Document what you have done.
2. Create a nice presentation.
3. Explain why your solution achieves the business objective.

Launch

1. Get your solution ready for production (plug into production data inputs, write unit tests, etc.).
2. Write monitoring code to check your system's live performance at regular intervals and trigger alerts when it drops.
3. Retrain your models on a regular basis on fresh data (automate as much as possible).

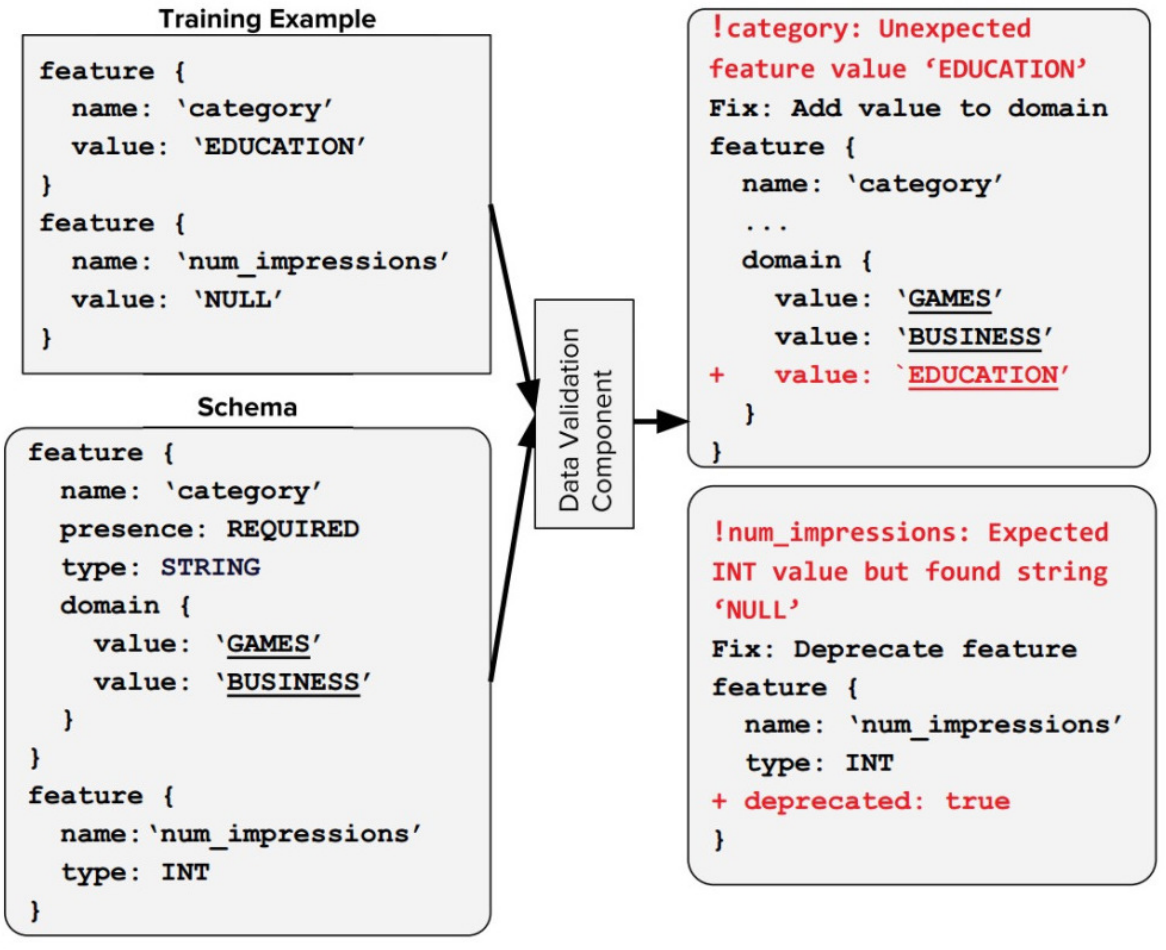
Data Science Pipeline



Data Cleansing

"We want the user to treat data errors with the same rigor and care that they deal with bugs in code. To promote this practice, we allow anomalies to be filed just like any software bug where they are documented, tracked, and eventually resolved."

TFX: A TensorFlow-based production scale machine learning platform, Baylor et al., KDD'17


Training Example
feature {
name: 'category'
value: 'EDUCATION'
}
feature {
name: 'num_impressions'
value: 'NULL'
}
Schema
feature {
name: 'category'
presence: REQUIRED
type: STRING
domain {
value: 'GAMES'
value: 'BUSINESS'
}
}
feature {
name: 'num_impressions'
type: INT
}
Data Validation Component
!category: Unexpected feature value 'EDUCATION'
Fix: Add value to domain
feature {
name: 'category'
...
domain {
value: 'GAMES'
value: 'BUSINESS'
+ value: `EDUCATION'
}
}
!num_impressions: Expected INT value but found string 'NULL'
Fix: Deprecate feature
feature {
name: 'num_impressions'
type: INT
+ deprecated: true
}

Field Validation

- Field delineation (boundaries/lengths)
- Verify presence of values
- Consider units

Value Validation

- Valid types
- Patterns (regex)
- Enumerations/factors/categories
- Verify range of values
- Text encoding of strings

Demo: How to Handle Missing Data

How to Handle Missing Data

- there are screenshots in this presentation to maintain continuity, but let's open the notebook named **`Demo - How to Handle Missing Data.ipynb`** and go through it together
- when done, click here to skip screenshots

Missing Data

```
>>> states = pd.Series(['alabama', 'california', np.nan, 'minnesota', 'virginia'])
```

```
>>> states
0       alabama
1    california
2           NaN
3     minnesota
4      virginia
dtype: object
```

```
>>> states.isnull()
0    False
1    False
2     True
3    False
4    False
dtype: bool
```

```
>>> states.isnull().any()
True
>>> states.isnull().sum()
1
```

```
>>> states[2] = None
```

```
>>> states.isnull()
0    False
1    False
2     True
3    False
4    False
dtype: bool
```

```
>>> states.dropna()
0       alabama
1    california
3     minnesota
4      virginia
dtype: object
```

```
>>> states.notnull()
0     True
1     True
2    False
3     True
4     True
dtype: bool
```

```
>>> states[states.notnull()]
0       alabama
1    california
3     minnesota
4      virginia
dtype: object
```

```
>>> data = pd.DataFrame([
['alabama', np.NAN, np.NAN],
['california', 39536653, 1246],
['minnesota', 5576606, 169],
[np.NAN, np.NAN, np.NAN],
['virginia', 8470020, 222]
])
```

```
>>> data
            0           1       2
0     alabama         NaN     NaN
1  california  39536653.0  1246.0
2   minnesota   5576606.0   169.0
3         NaN         NaN     NaN
4    virginia   8470020.0   222.0
```

```
>>> data.dropna()
            0           1       2
1  california  39536653.0  1246.0
2   minnesota   5576606.0   169.0
4    virginia   8470020.0   222.0
```

```
>>> data.dropna(how='all')
            0           1       2
0     alabama         NaN     NaN
1  california  39536653.0  1246.0
2   minnesota   5576606.0   169.0
4    virginia   8470020.0   222.0
```

```
>>> data.dropna(axis=1, how='all')
            0           1       2
0     alabama         NaN     NaN
1  california  39536653.0  1246.0
2   minnesota   5576606.0   169.0
3         NaN         NaN     NaN
4    virginia   8470020.0   222.0
```

```
>>> data.iloc[2,2] = np.NAN
>>> data
            0           1       2
0     alabama         NaN     NaN
1  california  39536653.0  1246.0
2   minnesota   5576606.0     NaN
3         NaN         NaN     NaN
4    virginia   8470020.0   222.0
```

```
>>> data.dropna(thresh=2)
            0           1       2
1  california  39536653.0  1246.0
2   minnesota   5576606.0     NaN
4    virginia   8470020.0   222.0
```

```
>>> data.fillna(0)
            0           1       2
0     alabama         0.0     0.0
1  california  39536653.0  1246.0
2   minnesota   5576606.0     0.0
3           0         0.0     0.0
4    virginia   8470020.0   222.0
```

```
>>> data.fillna({1:1000000, 2:50})
            0           1       2
0     alabama   1000000.0    50.0
1  california  39536653.0  1246.0
2   minnesota   5576606.0    50.0
3         NaN   1000000.0    50.0
4    virginia   8470020.0   222.0
```

```
>>> data.fillna(data.mean())
            0           1       2
0     alabama  17861093.0   734.0
1  california  39536653.0  1246.0
2   minnesota   5576606.0   734.0
3         NaN  17861093.0   734.0
4    virginia   8470020.0   222.0
```

```
>>> data = pd.DataFrame([
['alabama', np.NAN, np.NAN],
['california', 39536653, 1246],
['virginia', 8470020, 222],
['california', 39536653, 1246],
['minnesota', 5576606, 169],
[np.NAN, np.NAN, np.NAN],
['virginia', 8470020, 222]
])
```

```
>>> data.duplicated()
0    False
1    False
2    False
3     True
4    False
5    False
6     True
dtype: bool
```

```
>>> data.drop_duplicates()
            0           1       2
0     alabama         NaN     NaN
1  california  39536653.0  1246.0
2    virginia   8470020.0   222.0
4   minnesota   5576606.0   169.0
5         NaN         NaN     NaN
```

Replacing Text

```
>>> dollars = pd.Series(['12', '-$10', '$10,000'])
>>> dollars
0         12
1       -$10
2    $10,000
dtype: object
```

```
>>> dollars.str.replace('$', '')
0        12
1       -10
2    10,000
dtype: object
```

```
>>> dollars
0         12
1       -$10
2    $10,000
dtype: object
```

```
>>> dollars = dollars.str.replace('$', '')
```

Regular Expression Replace

```
>>> dollars.apply(lambda s: re.sub('\$', '', s))
0        12
1       -10
2    10,000
dtype: object
```

```
>>> states = pd.Series(['Caaliforni.', 'Maaryl.nd', 'Al.baama', 'Virginiaa'])
```

```
>>> states.apply(lambda s: re.sub('aa|\.', 'a', s))
0    California
1      Maryland
2       Alabama
3      Virginia
dtype: object
```

```
>>> data = pd.read_csv('states.csv')
>>> data
                            State Abbrev  Count  Population
0                         Alabama     AL    129   4874747.0
1                          Alaska     AK     35    739795.0
2                  American Samoa     AS      1     51504.0
3                         Arizona     AZ    155   7016270.0
4                        Arkansas     AR    108   3004279.0
5                      California     CA   1246  39536653.0
6                        Colorado     CO    171   5607154.0
7                     Connecticut     CT    114   3588184.0
8                        Delaware     DE     23    961939.0
9            District of Columbia     DC     33    693972.0
10  Federated States of Micronesia     FM      4         NaN
11                         Florida     FL    439  20984400.0
12                         Georgia     GA    210  10429379.0
13                            Guam     GU      2    167358.0
14                          Hawaii     HI     43   1427538.0
15                           Idaho     ID     33   1716943.0
16                        Illinois     IL    391  12802023.0
17                         Indiana     IN    175   6666818.0
18                            Iowa     IA    107   3145711.0
19                          Kansas     KS     99   2913123.0
20                        Kentucky     KY    165   4454189.0
21                       Louisiana     LA    173   4684333.0
22                           Maine     ME     60   1335907.0
23                 Marshall Islands     MH      2         NaN
24                        Maryland     MD    148   6052177.0
25                   Massachusetts     MA    261   6859819.0
26                        Michigan     MI    302   9962311.0
27                       Minnesota     MN    169   5576606.0
...
```

```
>>> bins = [0, 10, 100, 500, 1000, 2000]
```

```
>>> schools = pd.cut(data['Count'], bins)
>>> schools
0       (100, 500]
1        (10, 100]
2          (0, 10]
3       (100, 500]
4       (100, 500]
5     (1000, 2000]
6       (100, 500]
7       (100, 500]
8        (10, 100]
9        (10, 100]
10         (0, 10]
11      (100, 500]
12      (100, 500]
13         (0, 10]
14       (10, 100]
15       (10, 100]
...
```

```
>>> pd.value_counts(schools)
(100, 500]      27
(10, 100]       21
(0, 10]          7
(500, 1000]      3
(1000, 2000]     1
Name: Count, dtype: int64
```

```
>>> data.describe()
              Count    Population
count     59.000000  5.400000e+01
mean     159.949153  6.097689e+06
std      200.886005  7.216798e+06
min        1.000000  5.150400e+04
25%       36.000000  1.499889e+06
50%      108.000000  4.036820e+06
75%      181.500000  6.977157e+06
max     1246.000000  3.953665e+07
```

```
>>> pd.qcut(data['Count'], 4)
0      (108.0, 181.5]
1       (0.999, 36.0]
2       (0.999, 36.0]
3      (108.0, 181.5]
4       (36.0, 108.0]
5     (181.5, 1246.0]
6      (108.0, 181.5]
7      (108.0, 181.5]
8       (0.999, 36.0]
9       (0.999, 36.0]
10      (0.999, 36.0]
11    (181.5, 1246.0]
12    (181.5, 1246.0]
13      (0.999, 36.0]
14      (36.0, 108.0]
...
Name: Count, dtype: category
Categories (4, interval[float64]):
   [(0.999, 36.0] < (36.0, 108.0] < (108.0, 181.5] < (181.5, 1246.0]]
```

```
>>> cats = pd.qcut(data['Count'], 4)
>>> pd.value_counts(cats)
(181.5, 1246.0]    15
(36.0, 108.0]      15
(0.999, 36.0]      15
(108.0, 181.5]     14
  Name: Count, dtype: int64
```

```
>>> pop = data['Population']
>>> pop[pop > 5000000]
3      7016270.0
5     39536653.0
6      5607154.0
11    20984400.0
12    10429379.0
16    12802023.0
17     6666818.0
24     6052177.0
25     6859819.0
26     9962311.0
27     5576606.0
29     6113532.0
34     9005644.0
36    19849399.0
37    10273419.0
40    11658609.0
44    12805537.0
47     5024369.0
49     6715984.0
50    28304596.0
54     8470020.0
55     7405743.0
57     5795483.0
```

Exercise: Data Cleansing

(open the notebook named **`Exercise 11 - Data Cleansing.ipynb`**)

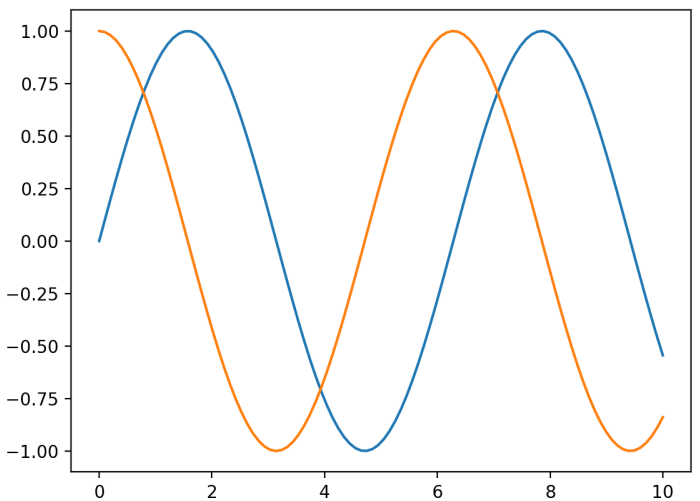
Demo: Data Visualization

Data Visualization

- there are screenshots in this presentation to maintain continuity, but they are actually the same as what we did in the Intro class
- feel free to open the notebook named **`Demo - Data Visualization.ipynb`** if you need it for the exercise
- click here to skip screenshots and jump to exercise

```
# show.py

import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))

plt.show()
```

```
> python show.py
```



localhost

PB store | Pinboard | Emergent | Images | Squirt | Hue Bridges | Coins | #Who%20is%2...n%20Phish? | DI | Bin2Text | TinyURL!

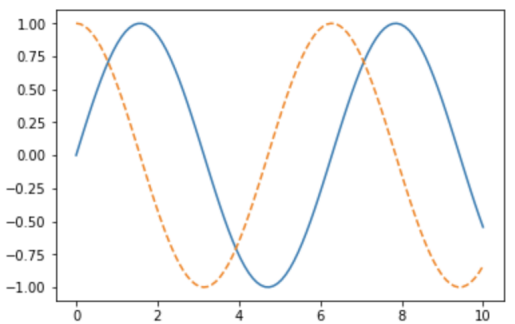
jupyter **Data Visualization** (autosaved) Logout

File Edit View Insert Cell Kernel Help Notebook saved Trusted Python 2

Code

In [1]:
```
%matplotlib inline
```

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 100)

fig = plt.figure()
plt.plot(x, np.sin(x), '-')
plt.plot(x, np.cos(x), '--');
```



In [ ]:

Saving Image Files

In [5]:
```
fig.savefig('my_figure.png')
```

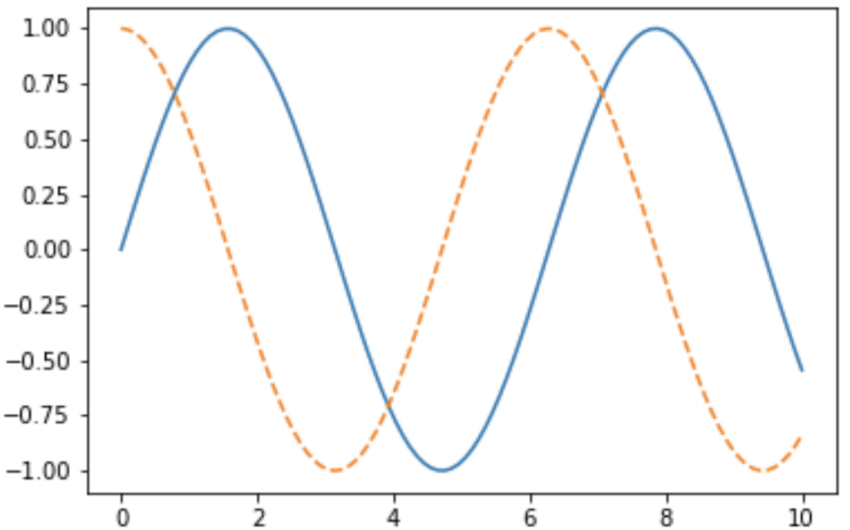
In [6]:
```
!ls -lh my_figure.png
```

```
-rw-r--r--  1 brian  staff    22K Jan 24 06:47 my_figure.png
```

Show Image Files

In [7]:
```
from IPython.display import Image
Image('my_figure.png')
```

Out[7]:



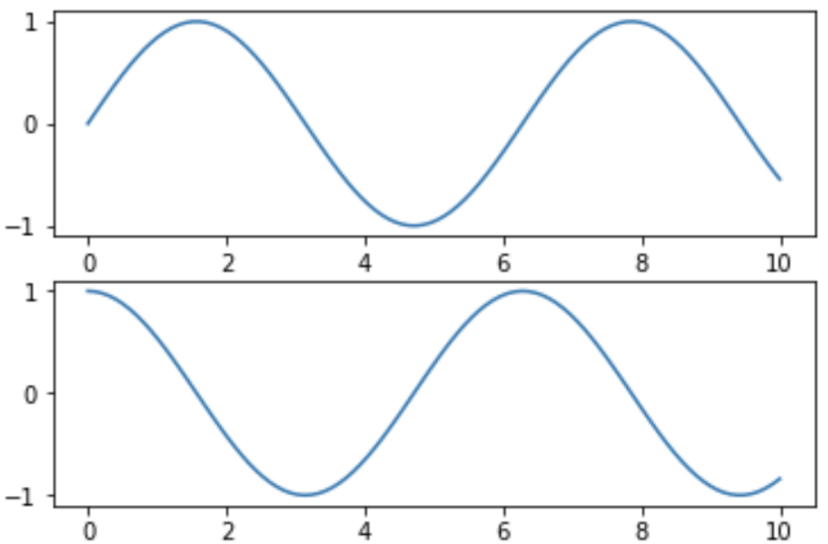
Show Available File Support

```
In [8]: fig.canvas.get_supported_filetypes()
```

```
Out[8]: {u'eps': u'Encapsulated Postscript',
 u'jpeg': u'Joint Photographic Experts Group',
 u'jpg': u'Joint Photographic Experts Group',
 u'pdf': u'Portable Document Format',
 u'pgf': u'PGF code for LaTeX',
 u'png': u'Portable Network Graphics',
 u'ps': u'Postscript',
 u'raw': u'Raw RGBA bitmap',
 u'rgba': u'Raw RGBA bitmap',
 u'svg': u'Scalable Vector Graphics',
 u'svgz': u'Scalable Vector Graphics',
 u'tif': u'Tagged Image File Format',
 u'tiff': u'Tagged Image File Format'}
```

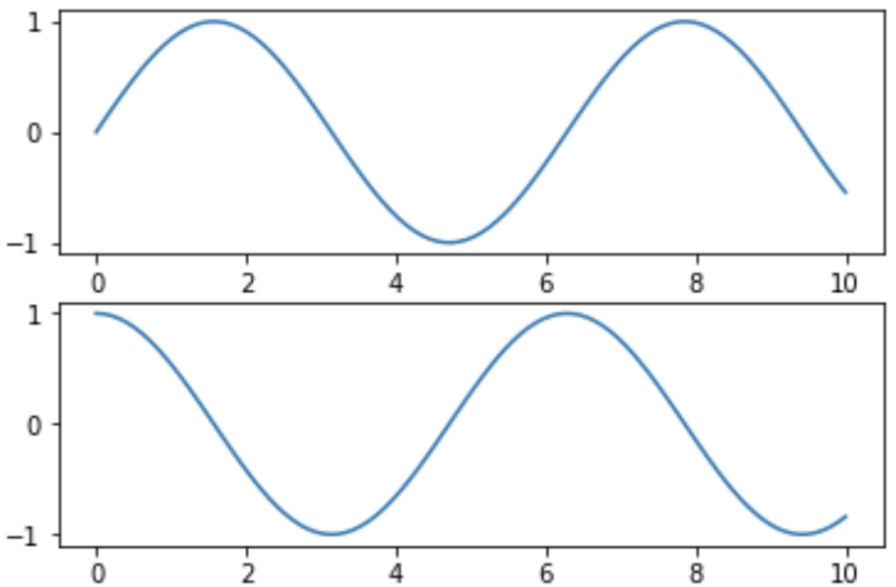
MATLAB-Style Interface

In [9]:
```
plt.figure()  # create a plot figure

# create the first of two panels and set current axis
plt.subplot(2, 1, 1) # (rows, columns, panel number)
plt.plot(x, np.sin(x))

# create the second panel and set current axis
plt.subplot(2, 1, 2)
plt.plot(x, np.cos(x));
```



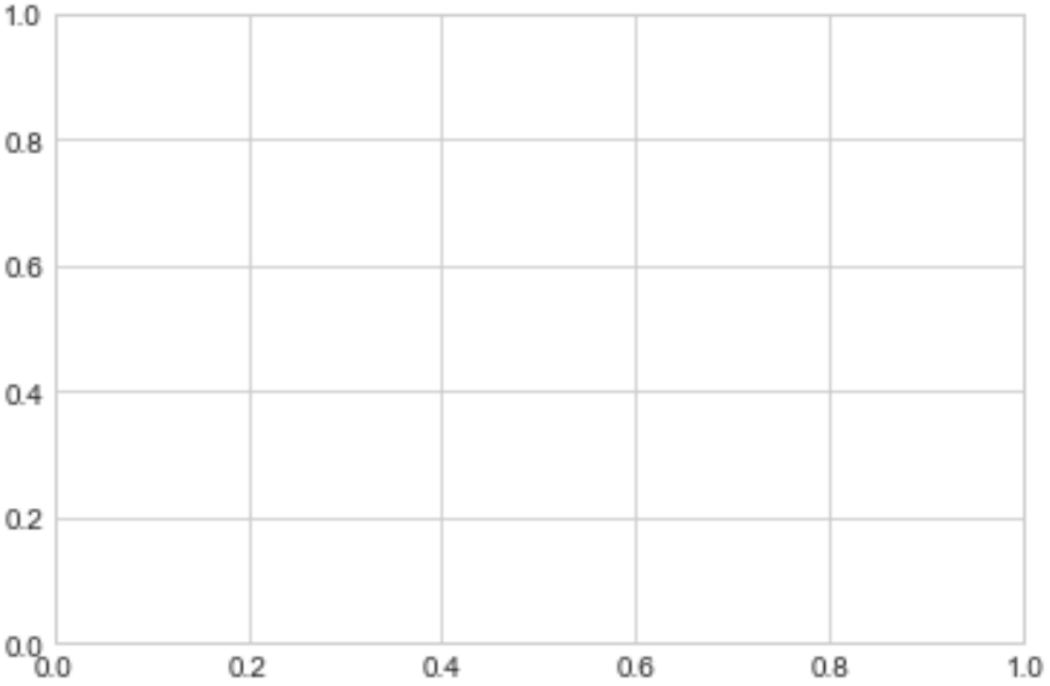
Object-Oriented Interface

In [10]:
```
# First create a grid of plots
# ax will be an array of two Axes objects
fig, ax = plt.subplots(2)

# Call plot() method on the appropriate object
ax[0].plot(x, np.sin(x))
ax[1].plot(x, np.cos(x));
```



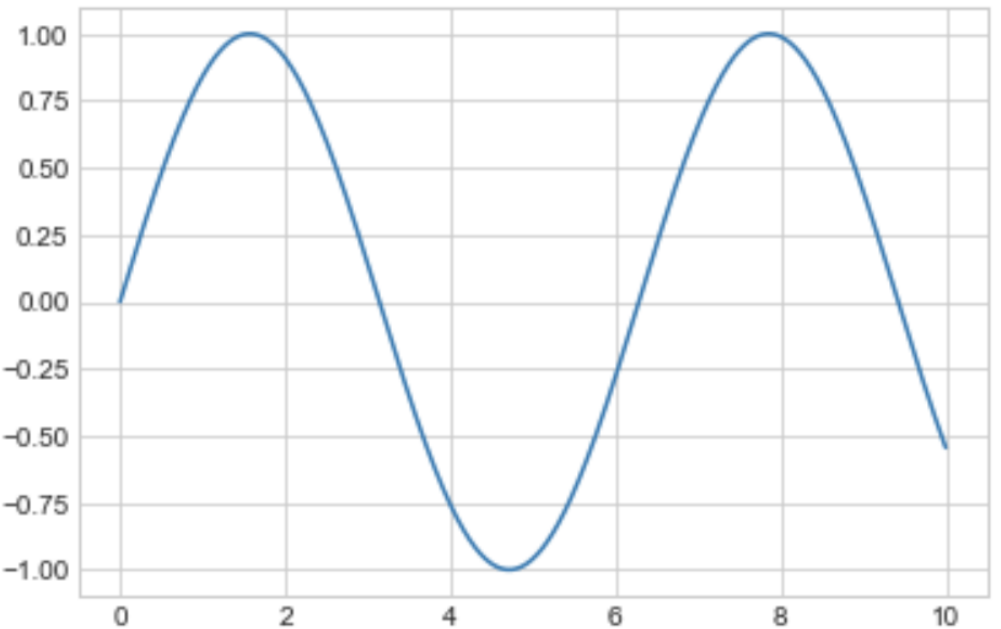
Set up a Grid

In [11]:
```
plt.style.use('seaborn-whitegrid')
fig = plt.figure()
ax = plt.axes()
```



Draw a Function

In [14]:
```
plt.style.use('seaborn-whitegrid')
fig = plt.figure()
ax = plt.axes()
x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x));
```

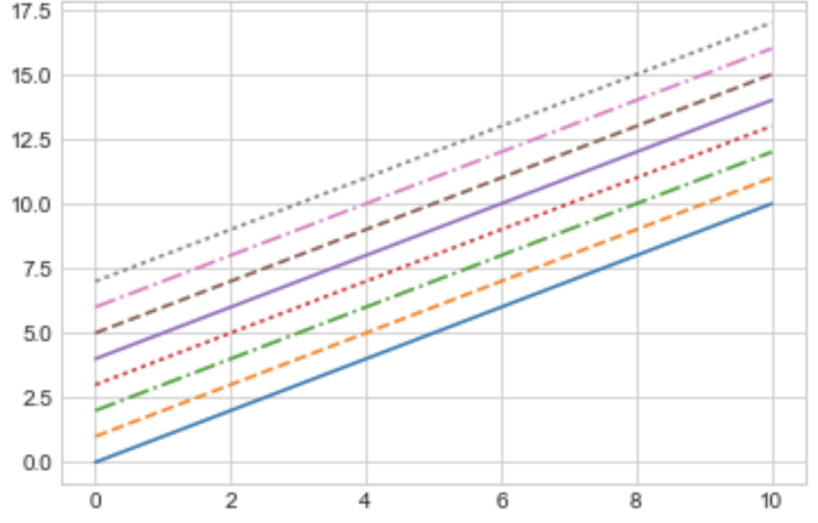


Ways to Specify Color

In [15]:
```
plt.plot(x, np.sin(x - 0), color='blue')        # specify color by name
plt.plot(x, np.sin(x - 1), color='g')           # short color code (rgbcmyk)
plt.plot(x, np.sin(x - 2), color='0.75')        # Grayscale between 0 and 1
plt.plot(x, np.sin(x - 3), color='#FFDD44')     # Hex code (RRGGBB from 00 to FF)
plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3)) # RGB tuple, values 0 to 1
plt.plot(x, np.sin(x - 5), color='chartreuse'); # all HTML color names supported
```

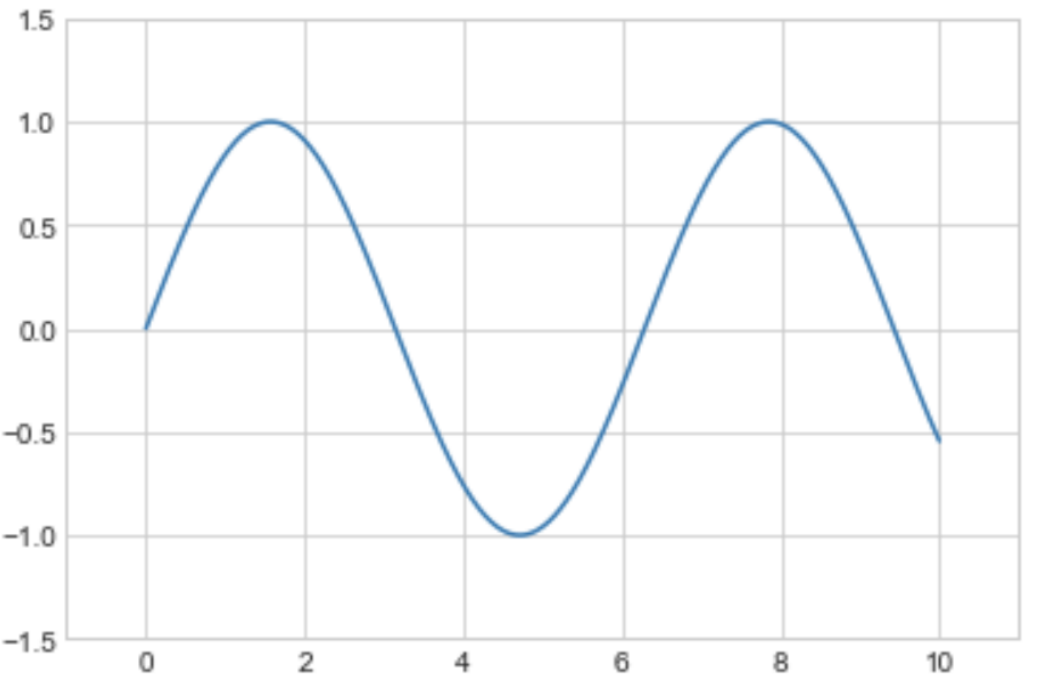
Ways to Specify Line Style

In [16]:
```
plt.plot(x, x + 0, linestyle='solid')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
plt.plot(x, x + 3, linestyle='dotted');

# For short, you can use the following codes:
plt.plot(x, x + 4, linestyle='-')  # solid
plt.plot(x, x + 5, linestyle='--') # dashed
plt.plot(x, x + 6, linestyle='-.') # dashdot
plt.plot(x, x + 7, linestyle=':');  # dotted
```



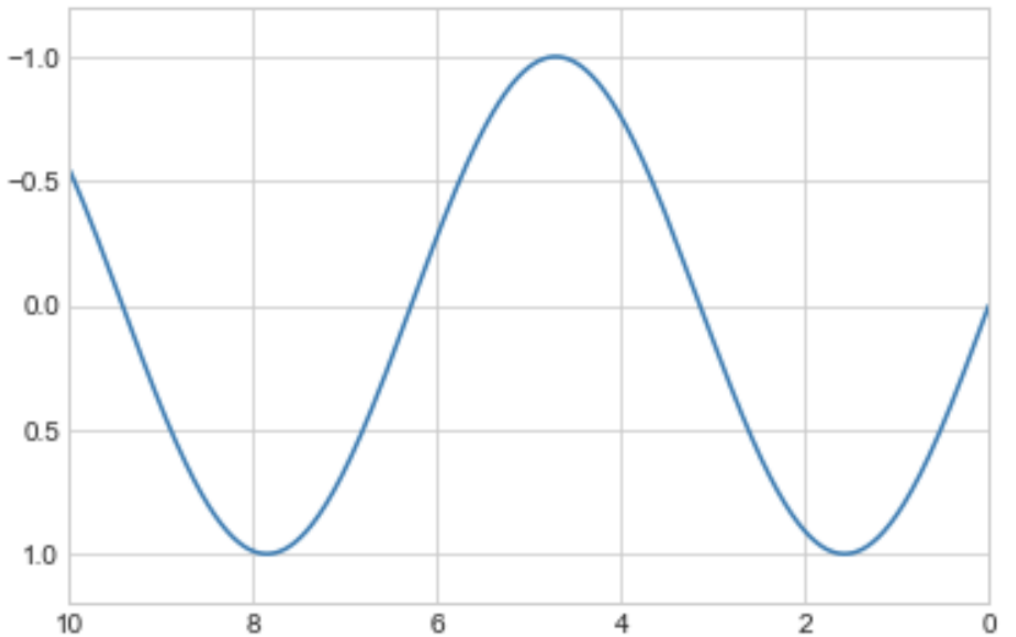
Setting Axes Limits

In [17]:
```
plt.plot(x, np.sin(x))

plt.xlim(-1, 11)
plt.ylim(-1.5, 1.5);
```



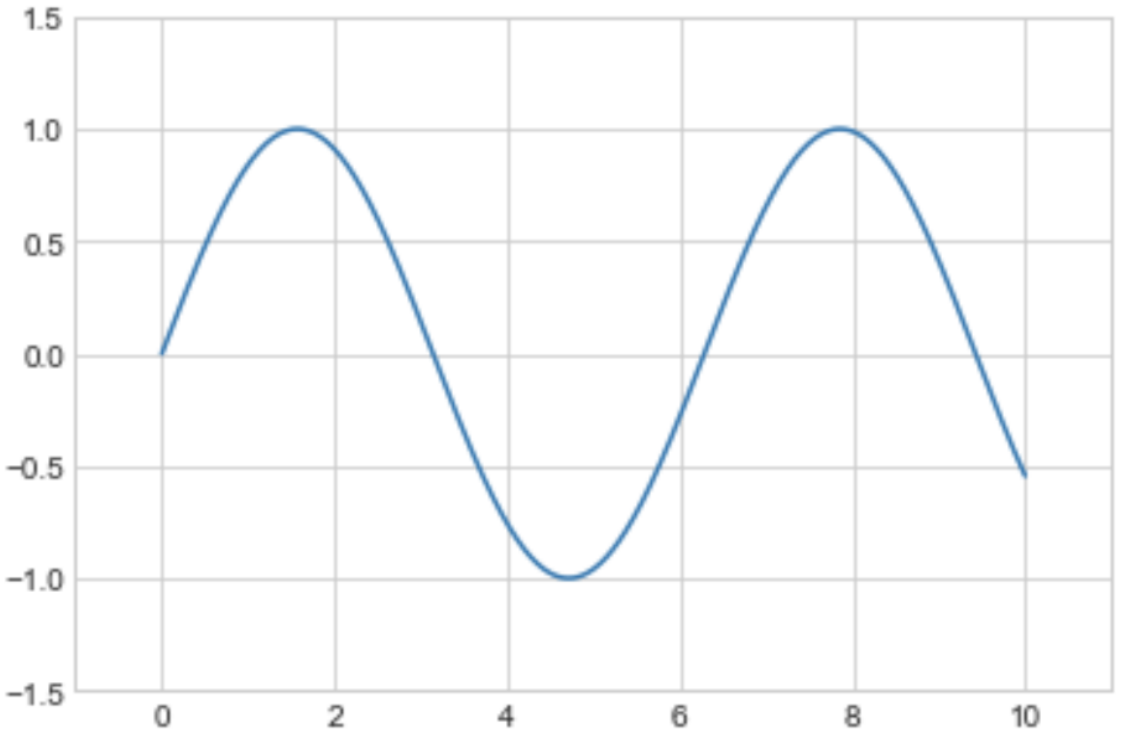
Flipping the Axes Limits

In [18]:
```
plt.plot(x, np.sin(x))

plt.xlim(10, 0)
plt.ylim(1.2, -1.2);
```



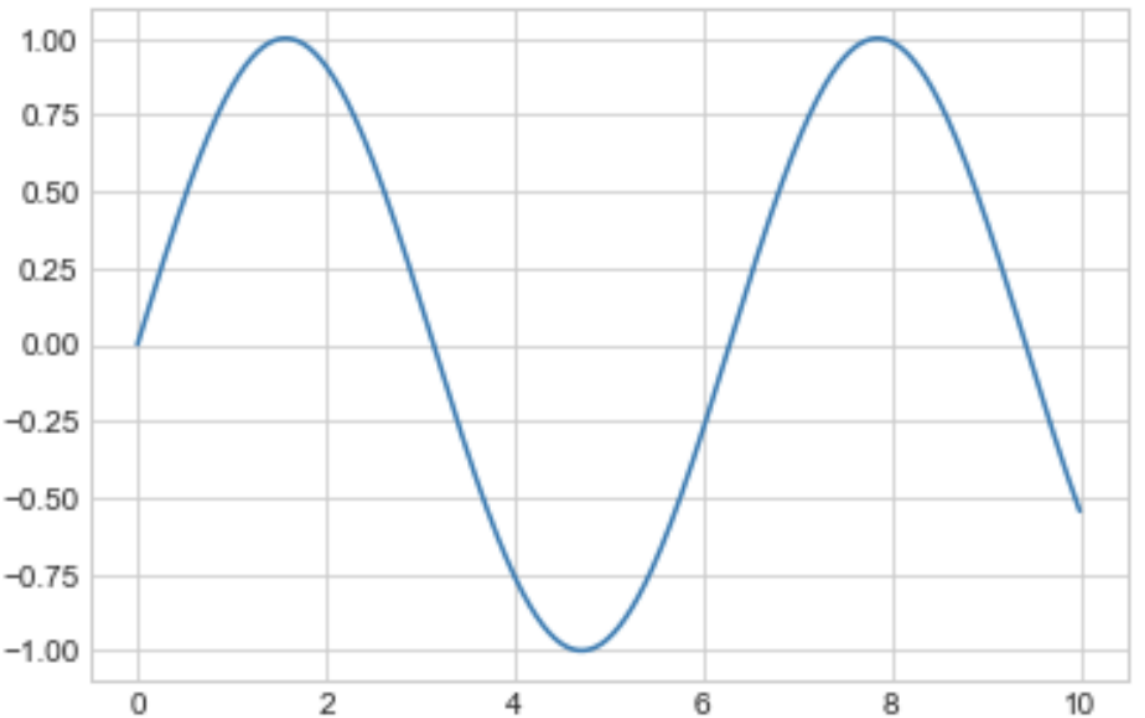
Axis

In [19]:
```
plt.plot(x, np.sin(x))
plt.axis([-1, 11, -1.5, 1.5]);
```



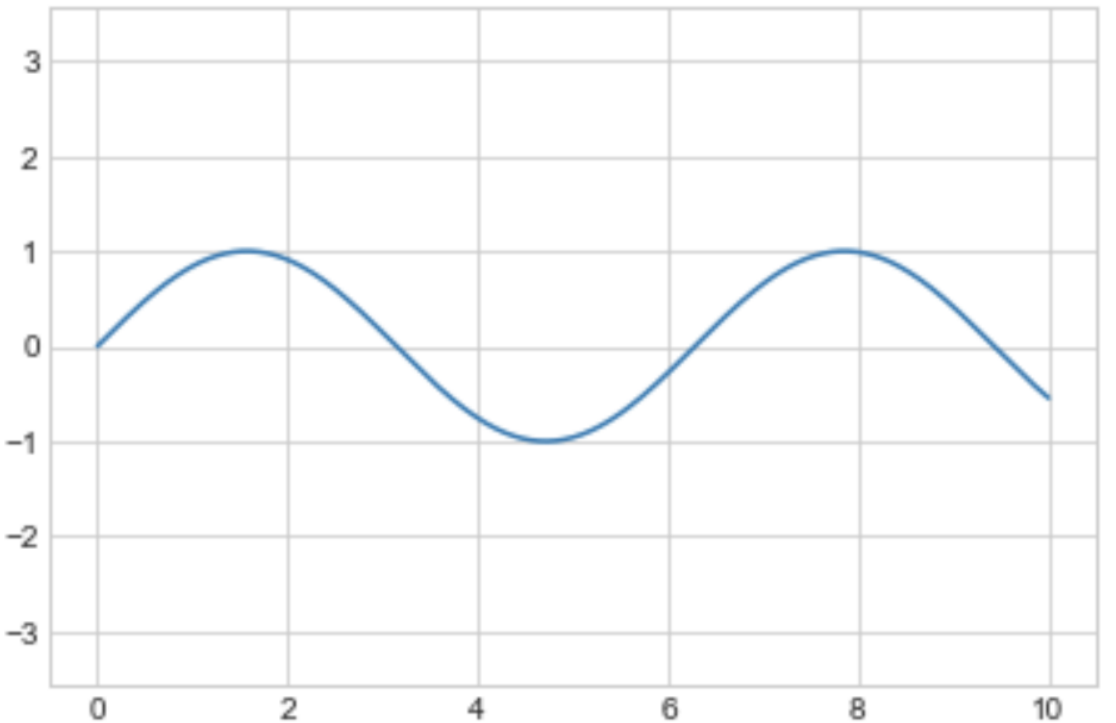
Tight Fit

In [20]:
```
plt.plot(x, np.sin(x))
plt.axis('tight');
```



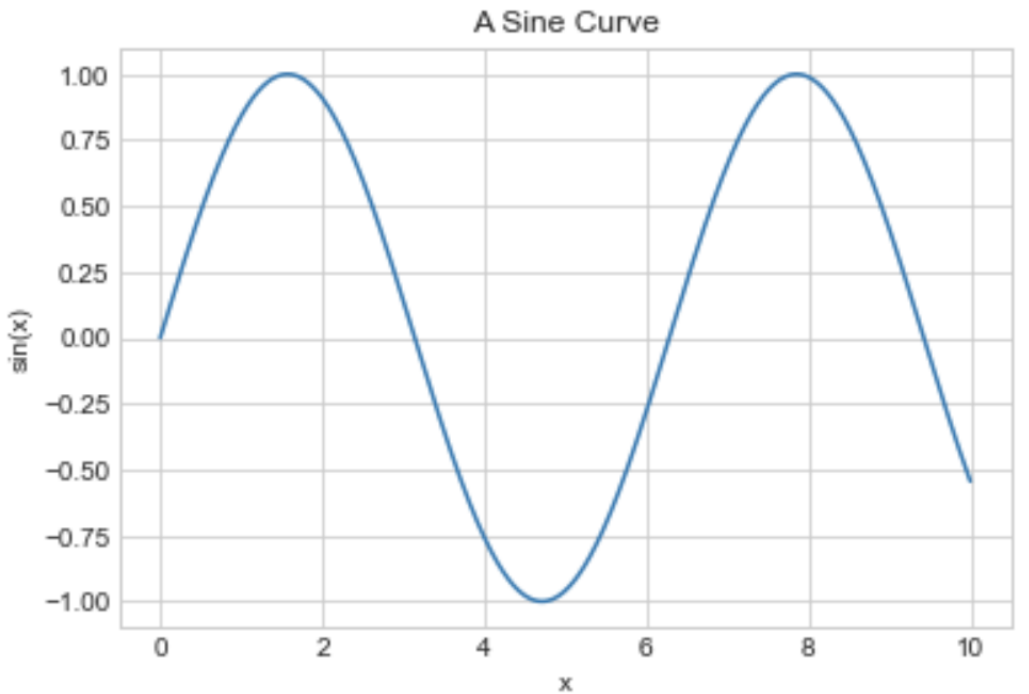
Equal Aspect Ratio

In [21]:
```
plt.plot(x, np.sin(x))
plt.axis('equal');
```



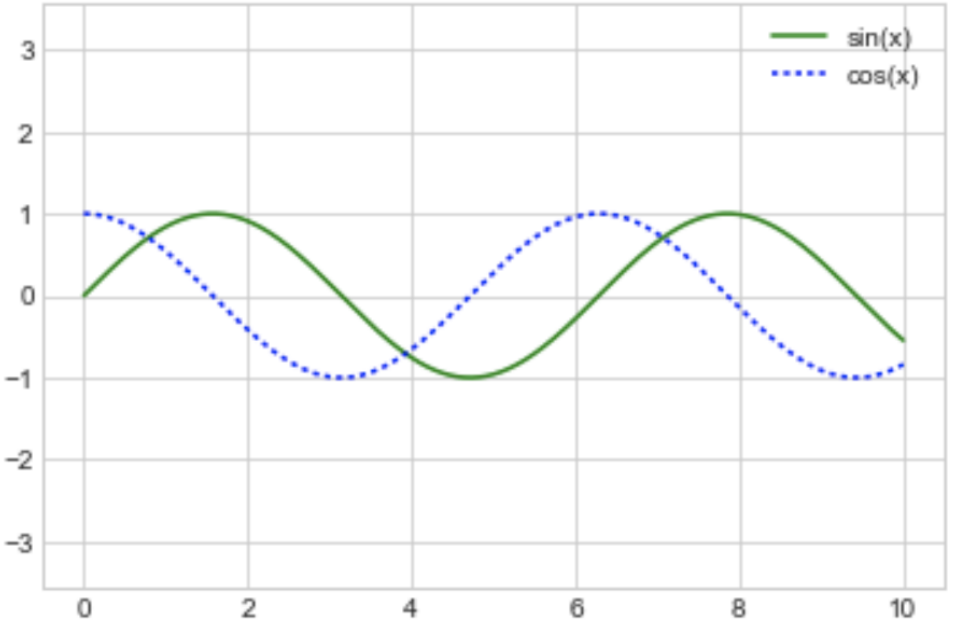
Labels

In [22]:
```
plt.plot(x, np.sin(x))
plt.title("A Sine Curve")
plt.xlabel("x")
plt.ylabel("sin(x)");
```



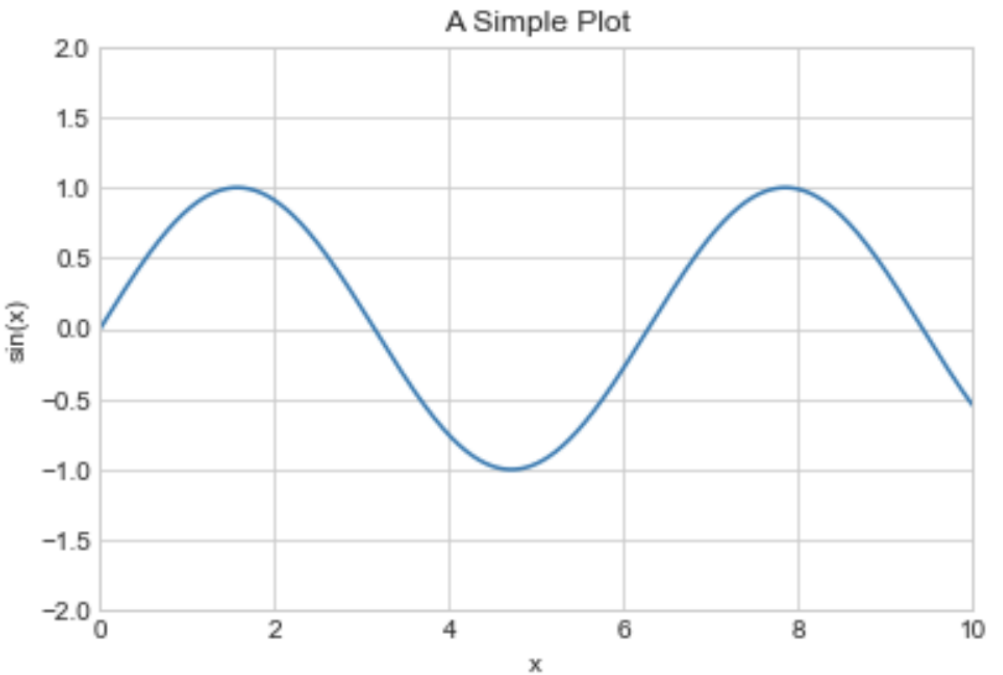
Legends

In [23]:
```
plt.plot(x, np.sin(x), '-g', label='sin(x)')
plt.plot(x, np.cos(x), ':b', label='cos(x)')
plt.axis('equal')

plt.legend();
```



Object-Oriented Interface

In [24]:
```
ax = plt.axes()
ax.plot(x, np.sin(x))
ax.set(xlim=(0, 10), ylim=(-2, 2),
       xlabel='x', ylabel='sin(x)',
       title='A Simple Plot');
```

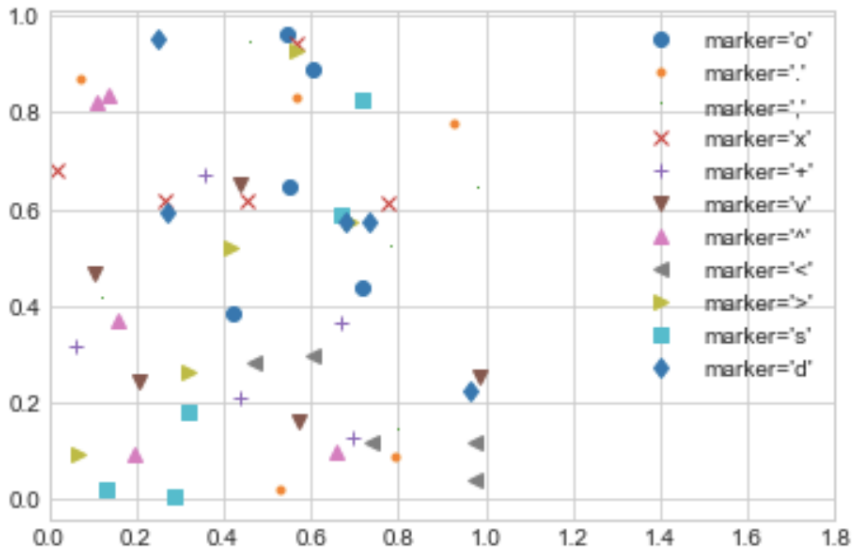


Interface Differences

MATLAB-Style	OO Style
plt.xlabel()	ax.set_xlabel()
plt.ylabel()	ax.set_ylabel()
plt.xlim()	ax.set_xlim()
plt.ylim()	ax.set_ylim()
plt.title()	ax.set_title()

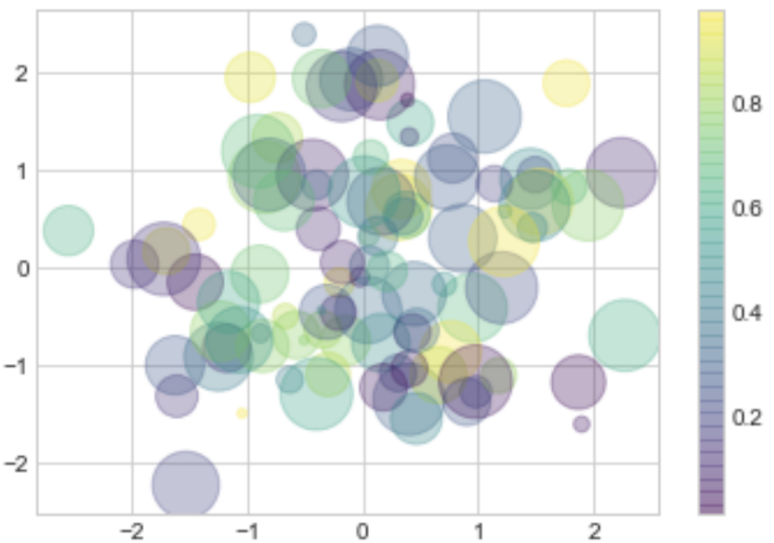
Specifying Markers

In [25]:
```
rng = np.random.RandomState(0)
for marker in ['o', '.', ',', 'x', '+', 'v', '^', '<', '>', 's', 'd']:
    plt.plot(rng.rand(5), rng.rand(5), marker,
             label="marker='{0}'".format(marker))
plt.legend(numpoints=1)
plt.xlim(0, 1.8);
```



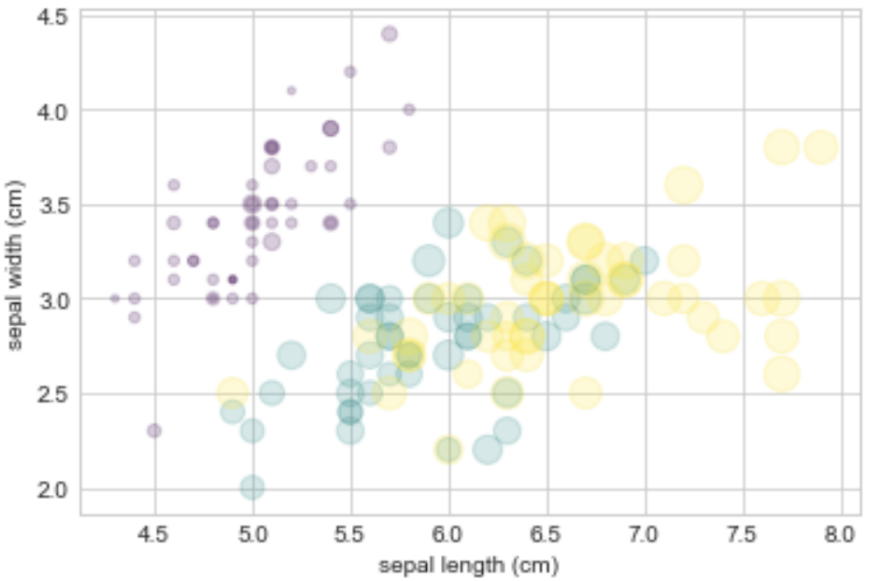
Scatterplot with Colors and Sizes

In [26]:
```
rng = np.random.RandomState(0)
x = rng.randn(100)
y = rng.randn(100)
colors = rng.rand(100)
sizes = 1000 * rng.rand(100)

plt.scatter(x, y, c=colors, s=sizes, alpha=0.3,
            cmap='viridis')
plt.colorbar();  # show color scale
```



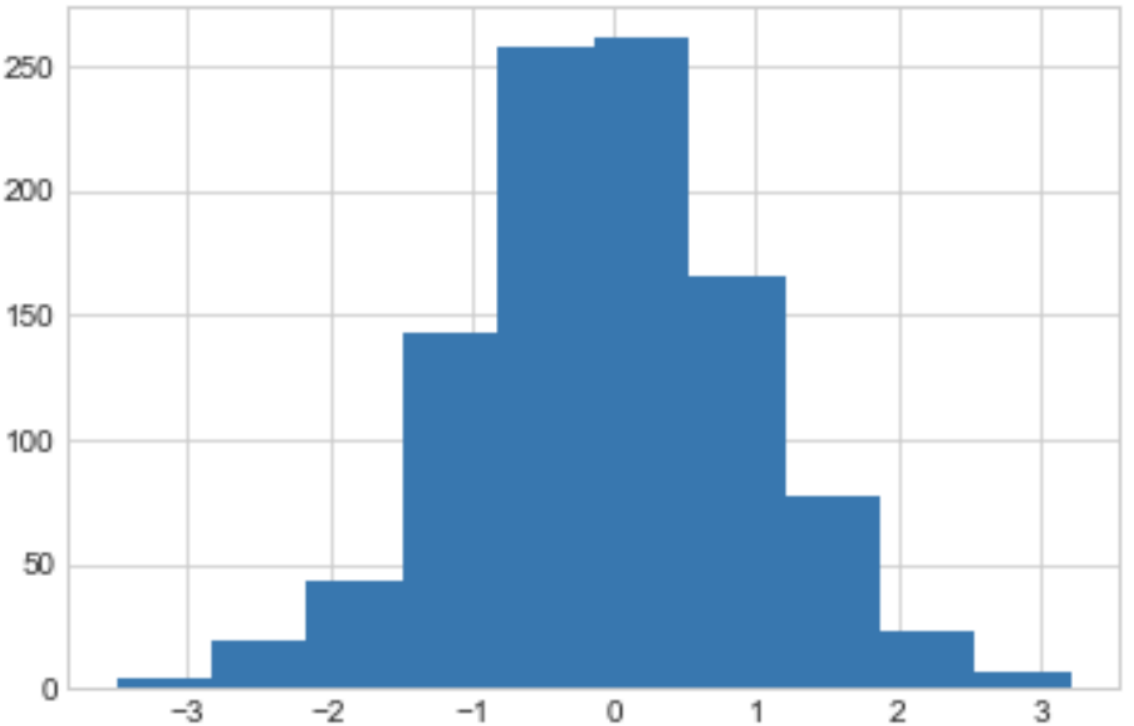
Visualizing Multiple Dimensions

In [27]:
```
from sklearn.datasets import load_iris
iris = load_iris()
features = iris.data.T

plt.scatter(features[0], features[1], alpha=0.2,
            s=100*features[3], c=iris.target, cmap='viridis')
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1]);
```



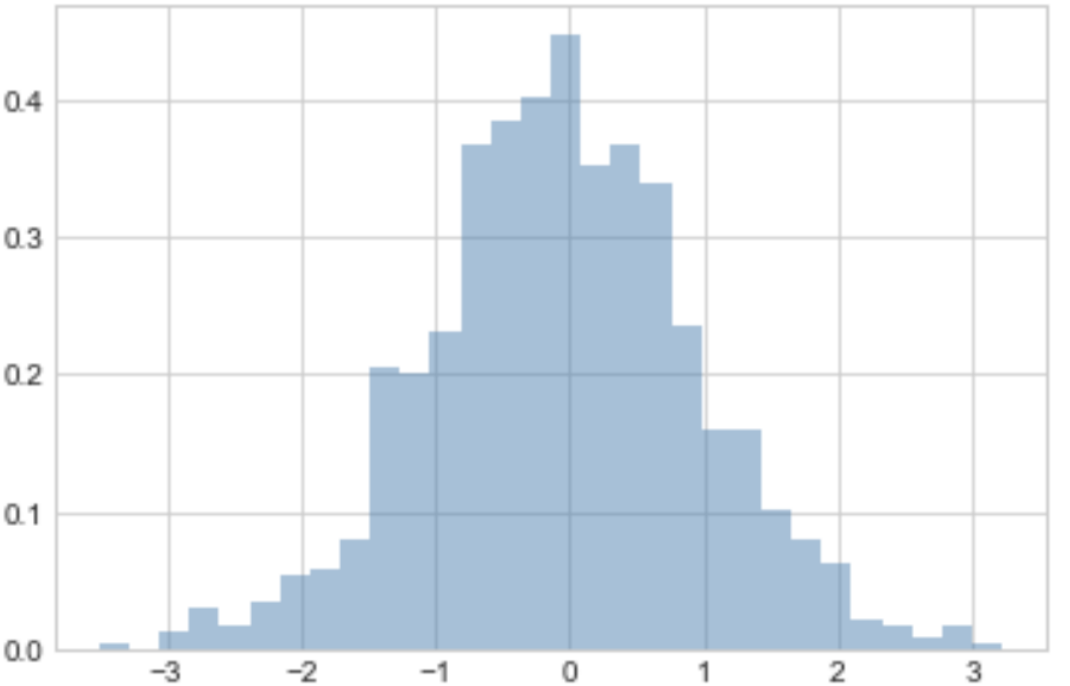
Histograms

In [28]:
```
data = np.random.randn(1000)
plt.hist(data);
```



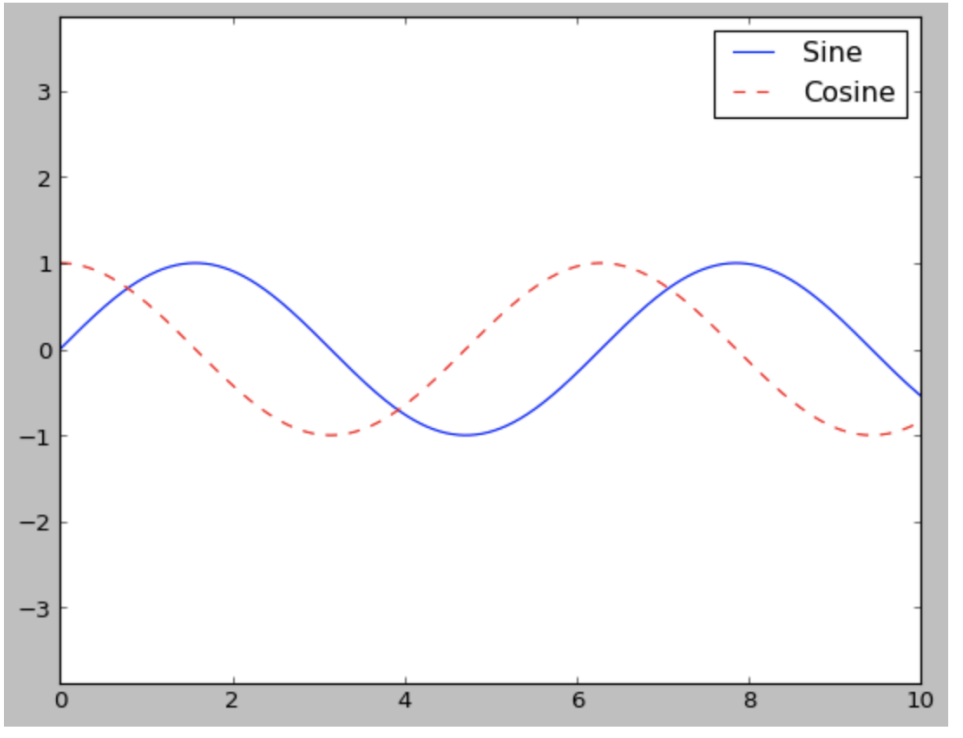
Customizing Histograms

In [29]:
```
plt.hist(data, bins=30, normed=True, alpha=0.5,
         histtype='stepfilled', color='steelblue',
         edgecolor='none');
```



Customizing Legends

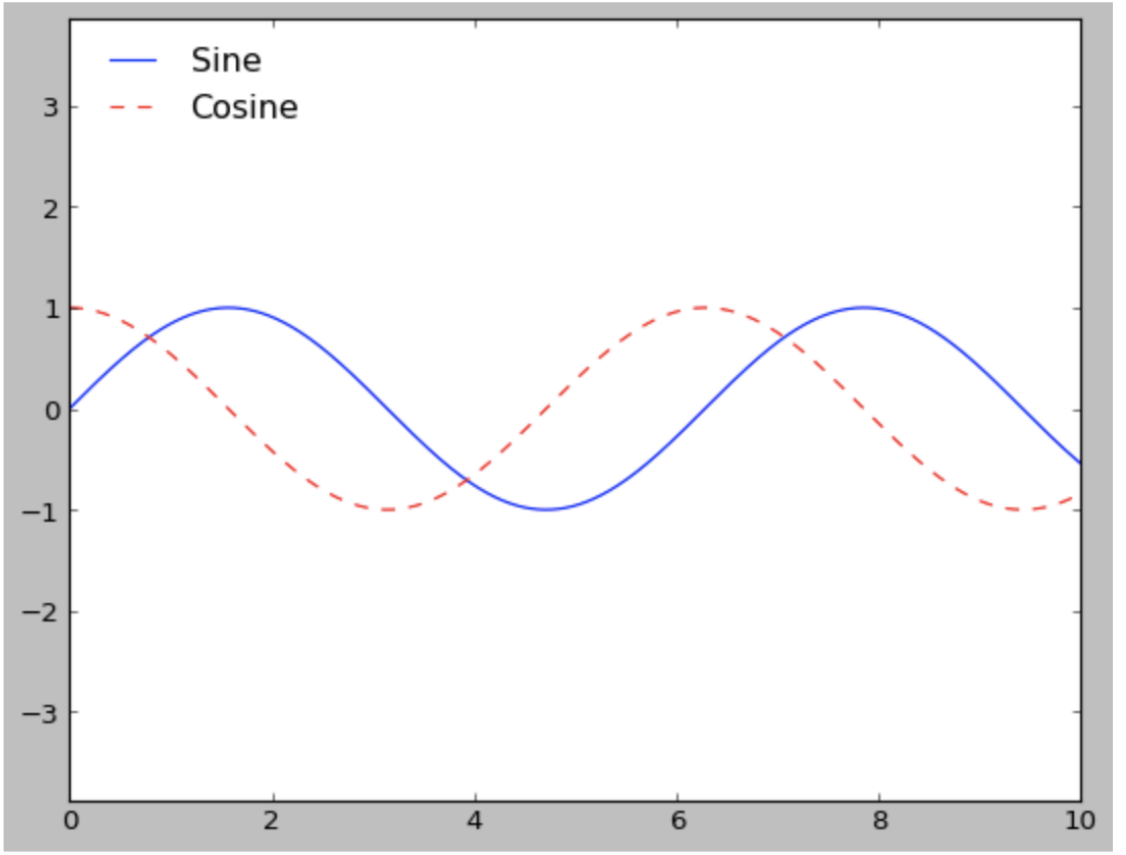
In [33]:
```
plt.style.use('classic')
x = np.linspace(0, 10, 1000)
fig, ax = plt.subplots()
ax.plot(x, np.sin(x), '-b', label='Sine')
ax.plot(x, np.cos(x), '--r', label='Cosine')
ax.axis('equal')
leg = ax.legend();
```



Customizing Legends

In [34]:
```
ax.legend(loc='upper left', frameon=False)
fig
```

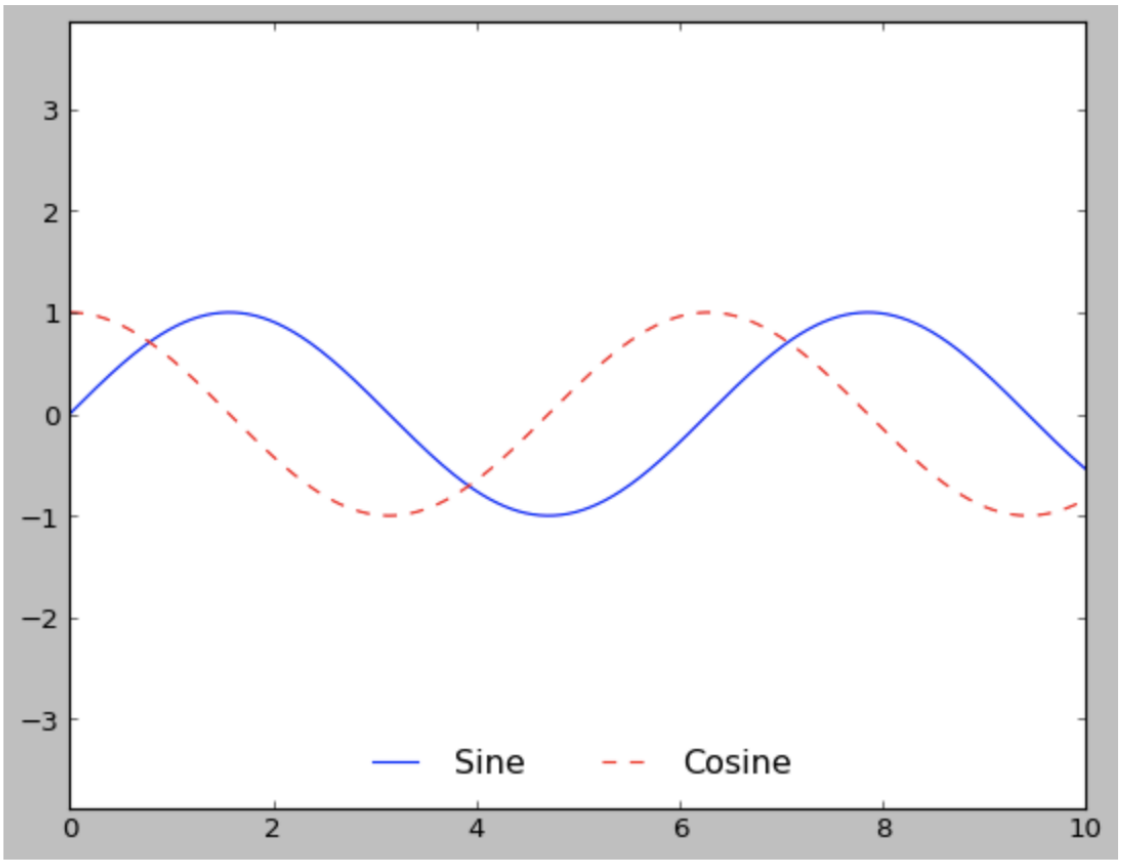
Out[34]:



Customizing Legends

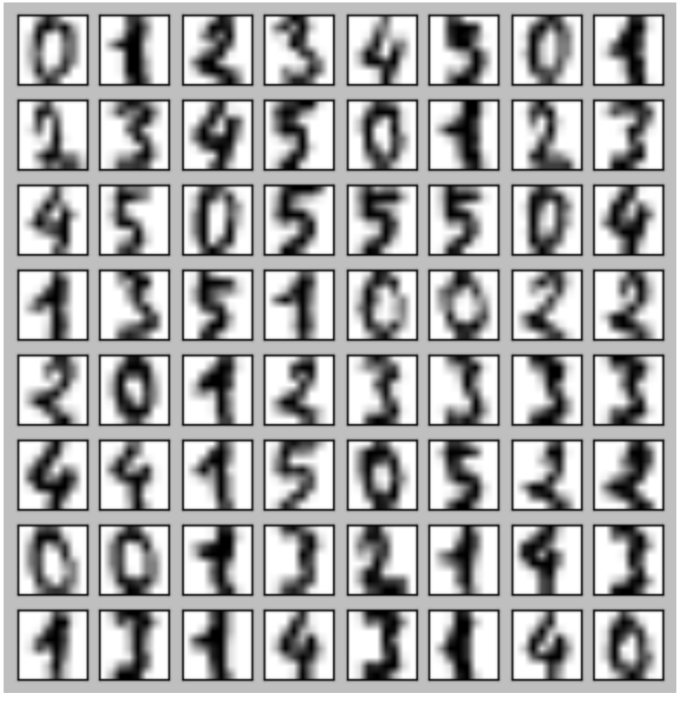
In [35]:
```
ax.legend(frameon=False, loc='lower center', ncol=2)
fig
```

Out[35]:



Visualizing Handwriting Samples

In [37]:
```
# load images of the digits 0 through 5 and visualize several of them
from sklearn.datasets import load_digits
digits = load_digits(n_class=6)

fig, ax = plt.subplots(8, 8, figsize=(6, 6))
for i, axi in enumerate(ax.flat):
    axi.imshow(digits.images[i], cmap='binary')
    axi.set(xticks=[], yticks=[])
```



Exercise: Data Visualization

(open the notebook named **`Exercise 12 - Data Visualization.ipynb`**)

Demo: Data Exploration

Data Exploration

- let's open the notebook named **`Demo - Boston Housing Data.ipynb`** and go through it together
- http://www.neural.cz/dataset-exploration-boston-house-pricing.html

Demo: Titanic Survivors

Titanic Survivors

- let's open the notebook named **`Demo - Titanic Survivors .ipynb`** and go through it together

Demo: Titanic Logistic Regression

Titanic Logistic Regression

- let's open the notebook named **`Demo - Titanic Logistic Regression .ipynb`** and go through it together
- http://www.data-mania.com/blog/logistic-regression-example-in-python/

Demo: Model Deeper Dive

Model Deeper Dive

- there are screenshots in this presentation to maintain continuity, but let's open the notebook named **`Demo - Model Deeper Dive.ipynb`** and go through it together
- when done, click here to skip screenshots

Iris Data

```
#Modified from http://www.agcross.com/2015/02/random-forests-scikit-learn/
#which was modified from the example written by yhat that can be found here:
#http://blog.yhathq.com/posts/random-forests-in-python.html

from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['is_train'] = np.random.uniform(0, 1, len(df)) <= .75
df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)

train, test = df[df['is_train']==True], df[df['is_train']==False]
features = df.columns[0:4]

y, _ = pd.factorize(train['species'])
```

Decision Tree

```
tree = DecisionTreeClassifier(max_depth=2)
tree.fit(train[features], y)

preds = iris.target_names[tree.predict(test[features])]
print(pd.crosstab(index=test['species'], columns=preds, rownames=['actual'],
      colnames=['preds']))
```

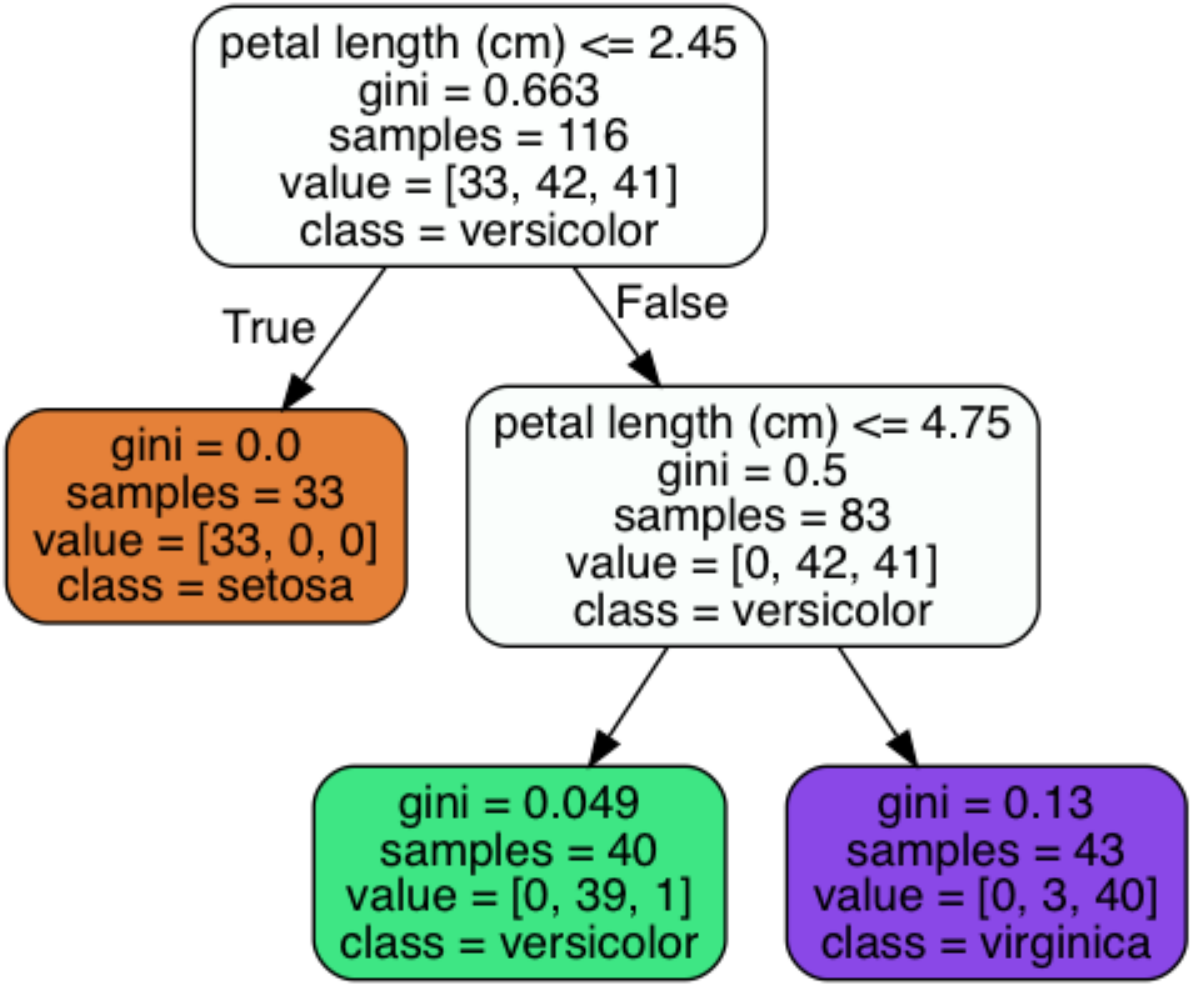
```
preds       setosa  versicolor  virginica
actual
setosa          17           0          0
versicolor       0           5          3
virginica        0           0          9
```

Visualizing the Tree

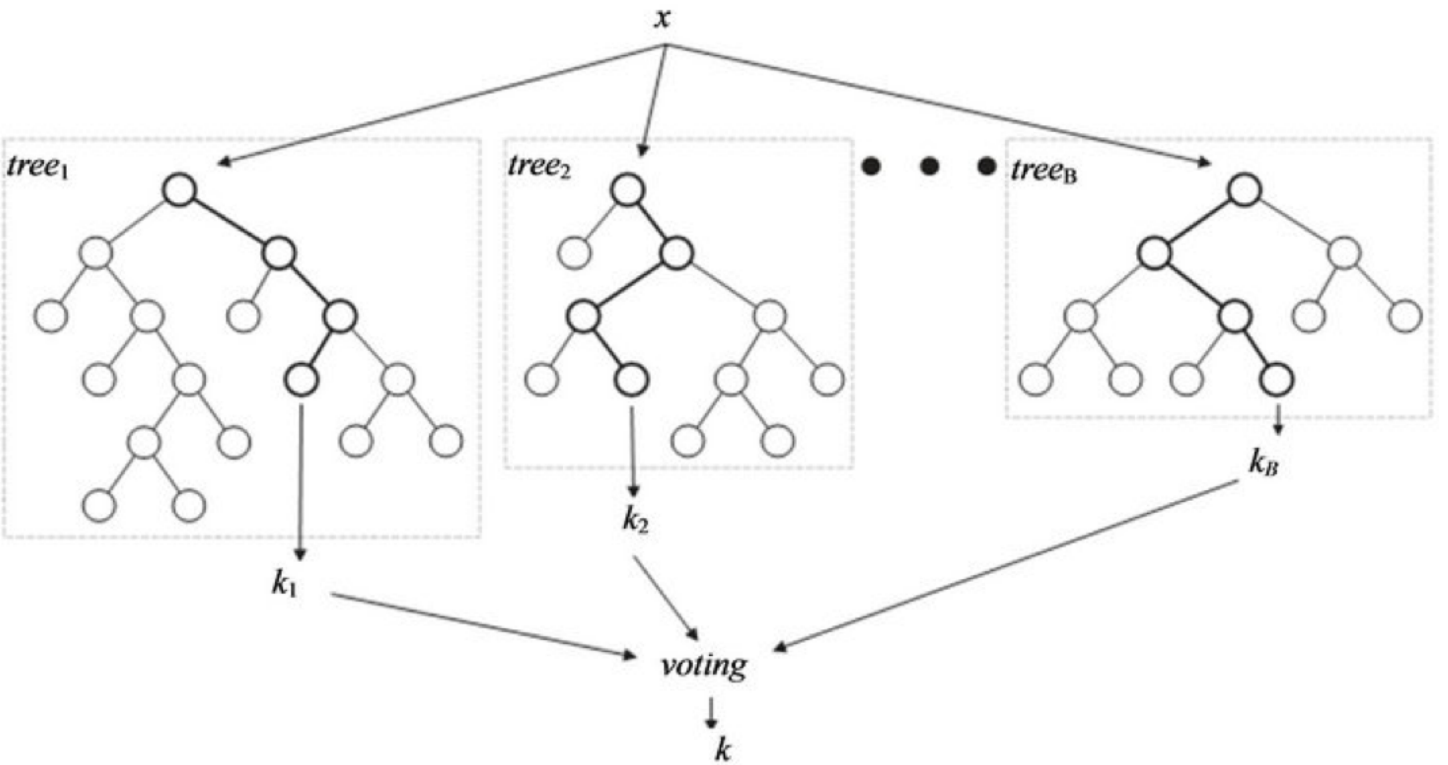
```
from sklearn.tree import export_graphviz
export_graphviz(tree, out_file="iris_tree.dot",
                feature_names=iris.feature_names,
                class_names=iris.target_names,
                rounded=True,
                filled=True)
```

```
!dot -Tpng iris_tree.dot -o iris_tree.png
```

```
from IPython.display import Image
Image('iris_tree.png')
```


petal length (cm) <= 2.45
gini = 0.663
samples = 116
value = [33, 42, 41]
class = versicolor
True
False
gini = 0.0
samples = 33
value = [33, 0, 0]
class = setosa
petal length (cm) <= 4.75
gini = 0.5
samples = 83
value = [0, 42, 41]
class = versicolor
gini = 0.049
samples = 40
value = [0, 39, 1]
class = versicolor
gini = 0.13
samples = 43
value = [0, 3, 40]
class = virginica

Random Forest



http://file.scirp.org/Html/6-9101686_31887.htm

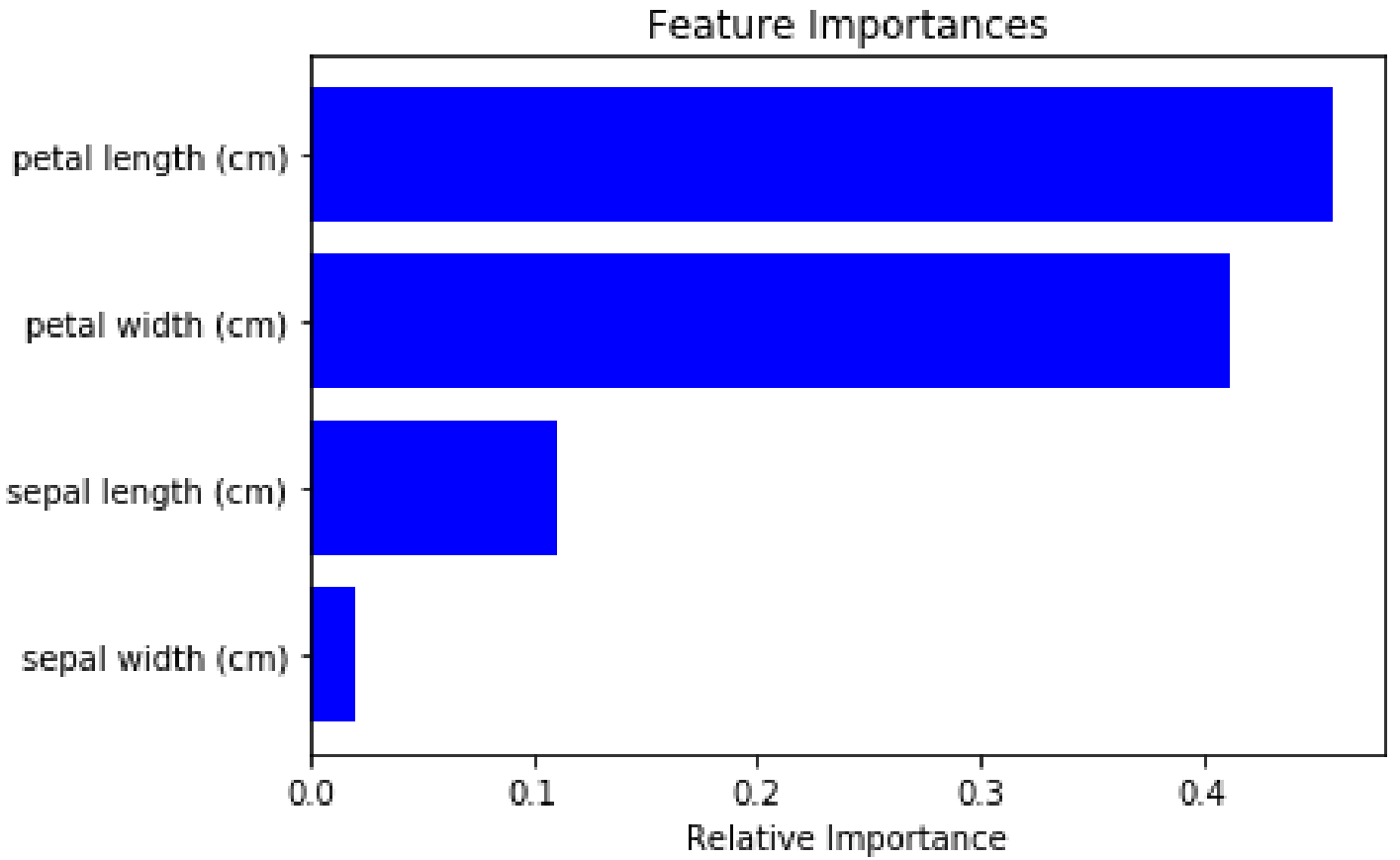
Random Forest

```
forest = RandomForestClassifier(n_jobs=2,n_estimators=50)
forest.fit(train[features], y)

preds = iris.target_names[forest.predict(test[features])]
print(pd.crosstab(index=test['species'], columns=preds, rownames=['actual'],
      colnames=['preds']))
```

```
preds       setosa  versicolor  virginica
actual
setosa          14           0          0
versicolor       0           9          1
virginica        0           1          9
```

Visualizing the Forest

```
importances = forest.feature_importances_
indices = np.argsort(importances)

plt.figure(1)
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), features[indices])
plt.xlabel('Relative Importance')
```


Feature Importances
petal length (cm)
petal width (cm)
sepal length (cm)
sepal width (cm)
0.0
0.1
0.2
0.3
0.4
Relative Importance

Exercise: Deep Dive - DecisionTree and RandomForest

open the notebook named **`Exercise 13 - Deep Dive (DecisionTree and RandomForest).ipynb`**