Python for Data Science
Day 3

By Craig Sakuma

Schedule - Day 3

Time	Topic
10:00 – 10:30	Review Python, Numpy & Pandas Solutions
10:30 – 12:00	Twitter API
12:00 – 1:00	Lunch
1:00 – 3:00	Data Cleaning
3:00 – 3:15	Break
3:15 – 4:15	Visualization in Matplotlib
4:15 – 4:30	Exception Handling
4:30 – 5:00	List Comprehensions, Map and Reduce

JSON

- JavaScript Object Notation
- Human readable
- Compatible across variety of languages
- JSON package converts data into Python dictionary
- Use ‘pretty print’ parameters in JSON package
- Think of data as giant multi-level outline
- Test validity and view ‘pretty’ format with http://jsonlint.com/

Create Pretty Print of Data

```
import json
```

```
print json.dumps(search_results, sort_keys=True,
                 indent=4)
```

Explore Data

```
print search_results.keys()
```

```
search_results['search_metadata']
```

View Statuses

```
type(search_results['statuses'])
```

```
print json.dumps(search_results['statuses'][0],
                 sort_keys=True, indent=4)
```

Extract Data from First Tweet

```
print search_results['statuses'][0]['text']
print search_results['statuses'][0]['created_at']
print search_results['statuses'][0]['user']['name']
```

Create Counter for Tweets

```
len(search_results['statuses'])
```

```
range(len(search_results['statuses']))
```

View All the Tweet Text

```
for n in range(len(search_results['statuses'])):
    print n, search_results['statuses'][n]['text']
```

Create Lists for Tweet Data

```
texts = []
created = []
name = []

for n in range(len(search_results['statuses'])):
    texts.append(search_results['statuses'][n]['text'])
    created.append(search_results['statuses'][n]['created_at'])
    name.append(search_results['statuses'][n]['user']['name'])
```

Build DataFrame with Tweets

```
import pandas as pd
```

```
df = pd.DataFrame({'text':texts,
                   'created_at':created,
                   'name':name})
print df
```

Data Munging

FOR BIG-DATA SCIENTISTS, 'JANITOR WORK' IS KEY HURDLE TO INSIGHTS

From NYTimes on August 18, 2014:

"Data wrangling is a huge — and surprisingly so — part of the job," said Monica Rogati, vice president for data science at Jawbone, whose sensor-filled wristband and software track activity, sleep and food consumption, and suggest dietary and health tips based on the numbers. "It's something that is not appreciated by data civilians. At times, it feels like everything we do."

Data Cleaning

- Missing Values
  - Identify Missing Values
  - Drop Values
  - Impute Values
    - Zero
    - Mean
- Categorical Data
  - Convert Text to Numbers
  - Encode Labels as Boolean Variables

Open combine.csv File

Import pandas

```
import pandas as pd
```

Read csv file and view data

```
data = pd.read_csv('combine.csv')
print data.head()
print data.info()
print data.describe()
```

pandas.DataFrame.fillna

`DataFrame.fillna`(*value=None, method=None, axis=0, inplace=False, limit=None, downcast=None*) ¶

Fill NA/NaN values using the specified method

Parameters :

method : *{'backfill', 'bfill', 'pad', 'ffill', None}, default None*

Method to use for filling holes in reindexed Series pad / ffill: propagate last valid observation forward to next valid backfill / bfill: use NEXT valid observation to fill gap

value : *scalar, dict, or Series*

Value to use to fill holes (e.g. 0), alternately a dict/Series of values specifying which value to use for each index (for a Series) or column (for a DataFrame). (values not in the dict/Series will not be filled). This value cannot be a list.

axis : *{0, 1}, default 0*

- 0: fill column-by-column
- 1: fill row-by-row

inplace : *boolean, default False*

If True, fill in place. Note: this will modify any other views on this object, (e.g. a no-copy slice for a column in a DataFrame).

limit : *int, default None*

Maximum size gap to forward or backward fill

downcast : *dict, default is None*

a dict of item->dtype of what to downcast if possible, or the string 'infer' which will try to downcast to an appropriate equal type (e.g. float64 to int64 if possible)

Returns :

filled : *same type as caller*

Fill Missing Values

Fill missing values for college with 'No College'

```
print data.info()
data.college.fillna('No College',inplace=True)
print data.info()
```

pandas.DataFrame.dropna

`DataFrame.dropna`(*axis=0, how='any', thresh=None, subset=None, inplace=False*)

Return object with labels on given axis omitted where alternately any or all of the data are missing

Parameters :

axis *: {0, 1}, or tuple/list thereof*

Pass tuple or list to drop on multiple axes

how *: {'any', 'all'}*

- any : if any NA values are present, drop that label
- all : if all values are NA, drop that label

thresh *: int, default None*

int value : require that many non-NA values

subset *: array-like*

Labels along other axis to consider, e.g. if you are dropping rows these would be a list of columns to include

inplace *: boolean, defalt False*

If True, do operation inplace and return None.

Returns :

dropped *: DataFrame*

Drop Missing Values

Remove the players that have null values for the pick column

```
print data.info()
data_dropped = data.dropna(subset=['pick'])
print data_dropped.info()
```

pandas.DataFrame.replace

`DataFrame.replace`(*to_replace=None*, *value=None*, *inplace=False*, *limit=None*, *regex=False*, *method='pad'*, *axis=None*)

Replace values given in 'to_replace' with 'value'.

Parameters :

to_replace *: str, regex, list, dict, Series, numeric, or None*

- str or regex:
  - str: string exactly matching *to_replace* will be replaced with *value*
  - regex: regexs matching *to_replace* will be replaced with *value*
- list of str, regex, or numeric:
  - First, if *to_replace* and *value* are both lists, they **must** be the same length.
  - Second, if `regex=True` then all of the strings in **both** lists will be interpreted as regexs otherwise they will match directly. This doesn't matter much for *value* since there are only a few possible substitution regexes you can use.
  - str and regex rules apply as above.
- dict:
  - Nested dictionaries, e.g., {'a': {'b': nan}}, are read as follows: look in column 'a' for the value 'b' and replace it with nan. You can nest regular expressions as well. Note that column names (the top-level dictionary keys in a nested dictionary) **cannot** be regular expressions.
  - Keys map to column names and values map to substitution values. You can treat this as a special case of passing two lists except that you are specifying the column to search in.

Replace RB and QB

Investigate the unique values in the position column

```
data.position.unique()
```

Replace the abbreviations for RB and QB

```
data.position.replace(['RB','QB'],['Running Back',
                                   'Quarterback'],inplace=True)
data.head(20)
```

pandas.get_dummies

`pandas.get_dummies`(*data*, *prefix=None*, *prefix_sep='_'*, *dummy_na=False*)

Convert categorical variable into dummy/indicator variables

Parameters :

data *: array-like or Series*

prefix *: string, default None*

String to append DataFrame column names

prefix_sep *: string, default '_'*

If appending prefix, separator/delimiter to use

dummy_na *: bool, default False*

Add a column to indicate NaNs, if False NaNs are ignored.

Returns :

dummies *: DataFrame*

Create Dummy Values

Convert position column into dummy values

```
data_positions = pd.get_dummies(data.position,prefix='Pos')
data_positions.head()
```

pandas.DataFrame.merge

`DataFrame.merge`(*right*, *how='inner'*, *on=None*, *left_on=None*, *right_on=None*, *left_index=False*, *right_index=False*, *sort=False*, *suffixes=('_x', '_y')*, *copy=True*)

Merge DataFrame objects by performing a database-style join operation by columns or indexes.

If joining columns on columns, the DataFrame indexes *will be ignored*. Otherwise if joining indexes on indexes or indexes on a column or columns, the index will be passed on.

Parameters :

right *: DataFrame*

how *: {'left', 'right', 'outer', 'inner'}, default 'inner'*

- left: use only keys from left frame (SQL: left outer join)
- right: use only keys from right frame (SQL: right outer join)
- outer: use union of keys from both frames (SQL: full outer join)
- inner: use intersection of keys from both frames (SQL: inner join)

on *: label or list*

Field names to join on. Must be found in both DataFrames. If on is None and not merging on indexes, then it merges on the intersection of the columns by default.

left_on *: label or list, or array-like*

Field names to join on in left DataFrame. Can be a vector or list of vectors of the length of the DataFrame to use a particular vector as the join key instead of columns

right_on *: label or list, or array-like*

Field names to join on in right DataFrame or vector/list of vectors per left_on docs

Merge Data

Merge dummy values to original data

```
data = pd.merge(data, data_positions, left_index=True,
                right_index=True)
data.head()
```

Convert Weight

Create new column with weight in kgs

```
data['weight_kg'] = data.weight/2.2
data.head()
```

Lambda Function

- Python technique for creating compact functions
- One time use only
- Syntax:

    lambda <variable>: <actions with variable>

Example:

```
lambda x: x**2    =    def square(x):
                           return x**2
```

pandas.DataFrame.apply

`DataFrame.apply`(*func*, *axis=0*, *broadcast=False*, *raw=False*, *reduce=None*, *args=()*, ***kwds*)

Applies function along input axis of DataFrame.

Objects passed to functions are Series objects having index either the DataFrame's index (axis=0) or the columns (axis=1). Return type depends on whether passed function aggregates, or the reduce argument if the DataFrame is empty.

Parameters:

func *: function*

Function to apply to each column/row

axis *: {0 or 'index', 1 or 'columns'}, default 0*

- 0 or 'index': apply function to each column
- 1 or 'columns': apply function to each row

Convert Name

Capitalize name

```
data.name.apply(str.upper).head()
```

Reverse order of first and last name

```
data.apply(lambda x:'{0}, {1}'.format(
                x['lastname'], x['firstname']),
                axis=1).head()
```

Exercise

- Create new notebook and open train.csv file
- Replace male / female with boolean values
- Fill missing Age values with average
- Use get_dummies function to convert Pclass into 3 separate boolean variables
- Merge the results from get_dummies to the original DataFrame
- Save the DataFrame as clean_data.csv

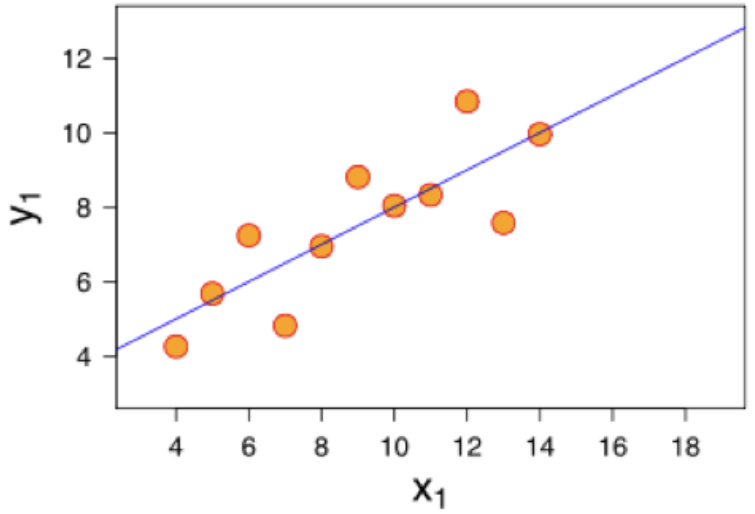
Data Visualization

- Visualization is important step in data exploration
- Gain insight into data that will guide analysis approaches
- Great technique to help explain patterns

Summary Statistics Are Valuable

Consider the following dataset:

- *eleven (x, y) points*
- *mean of x = 9, mean of y = 7.5*
- *variance of x = 11, variance of y = 4.1*
- *correlation of x, y = 0.8*
- *line of best fit: y = 3.00 + 0.500x*

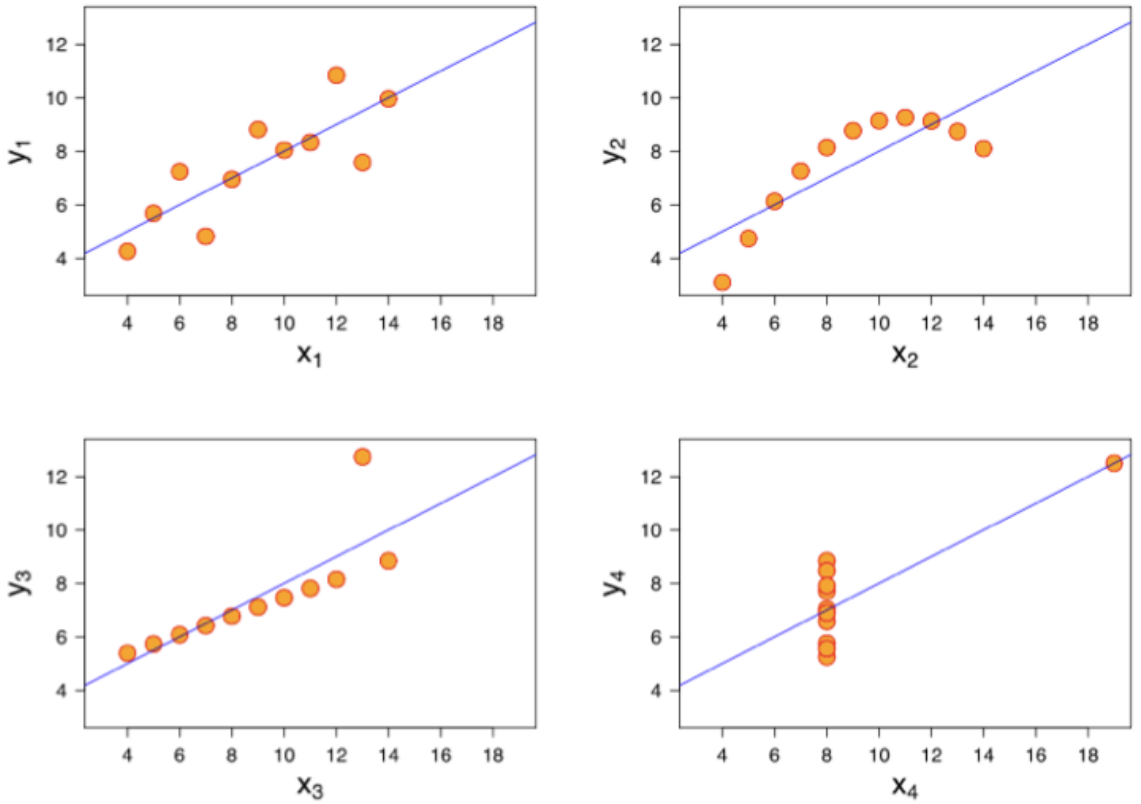


But They Don't Always Tell the Whole Story

Now, suppose I give you three more datasets with exactly the same characteristics.

Q: how similar are these datasets?

A: not very!

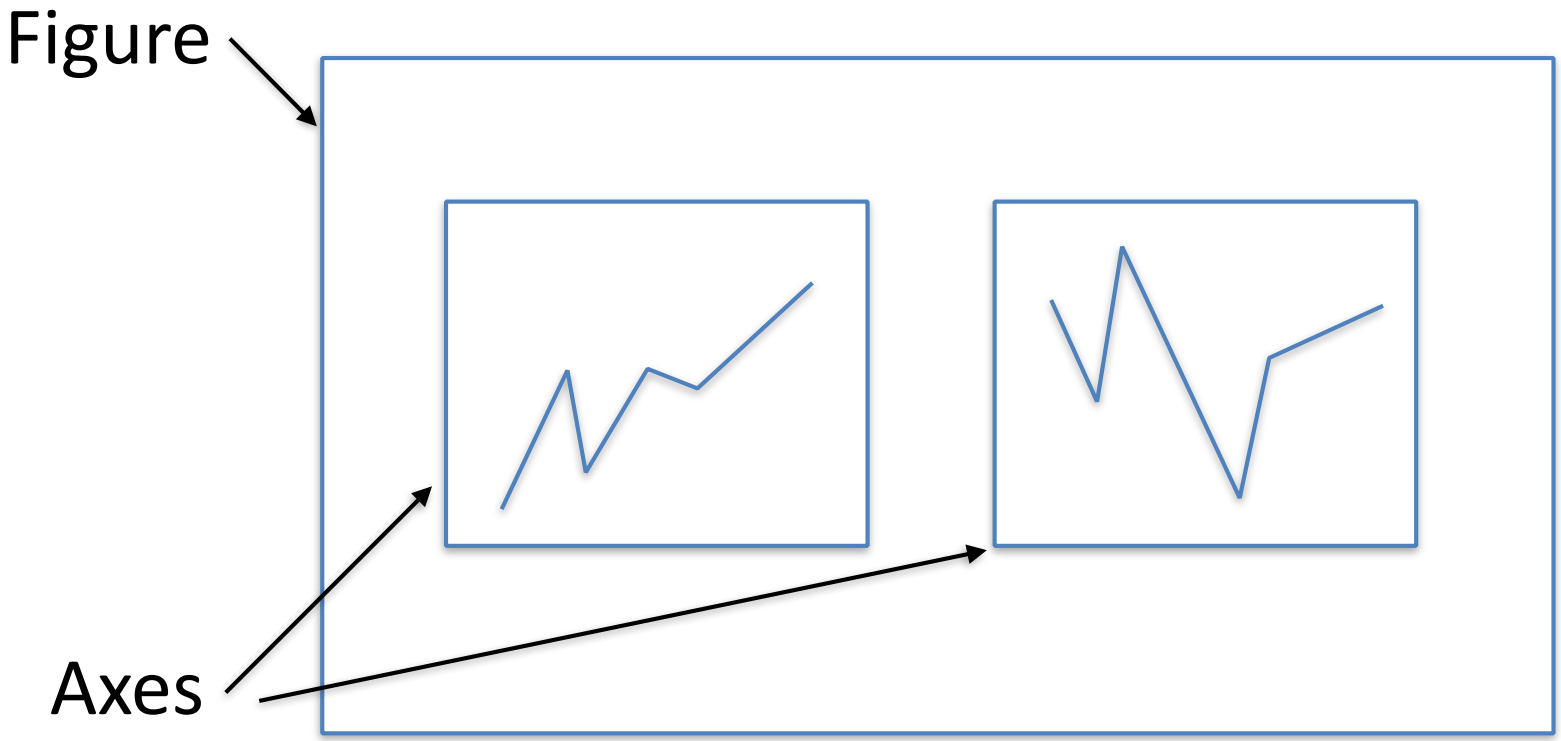


Matplotlib

- Most popular graphing package in python
- Visualization is important step in exploratory analysis and communicating findings
- We'll be creating four types of graphs
  - Line Plots
  - Scatter Plots
  - Histograms
  - Box plots

Matpotlib

- Figures are the “canvas” of your plot
- Axes are individual plots or charts



Set Up Your Packages

Import your packages

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

Add “magic command” for iPython Notebook to display plots in the notebook

```
%matplotlib inline
```

Data for Plots

Create mock data for plots

```
x = np.arange(20)
y = np.random.normal(10, 1, 20)
z = np.random.normal(10, 2, 20)
```

Create Your First Plot

Create your first plot

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(x,y)
```

Subplot has 3 values:

- Number of rows
- Number of columns
- Chart number

Short-hand notation is three digits with no commas

Lines and Scatter Plots

Create two plots with line and scatterplot

```
fig = plt.figure()
ax1 = fig.add_subplot(121)
ax1.plot(x,y)
ax2 = fig.add_subplot(122)
ax2.scatter(y,z)
```

Histograms and Boxplots

Create two plots with histogram and boxplot

```
fig = plt.figure()
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
ax1.hist(y)
ax2.boxplot([y,z])
```

Labels and Legends

Plot multiple data sets on same axes with labels and a legend

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(x, y, label = 'Thing #1')
ax.plot(x, z, label = 'Thing #2')
ax.legend()
```

Titles and Suptitles

Add titles and suptitle to figure

```
fig = plt.figure()
ax1 = fig.add_subplot(121)
ax1.plot(x,y)
ax1.set_title("Line")
ax2 = fig.add_subplot(122)
ax2.scatter(y,z)
ax2.set_title("Scatter")
fig.suptitle('Two Charts')
```

Axis Labels and Saving Image

Add x and y axis labels with font sizes

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(x, y)
ax.set_ylabel('Y-Axis', fontsize=10)
ax.set_xlabel('X-Axis', fontsize=20)
```

Save figure as png file

```
fig.savefig('figure.png')
```

Seaborn

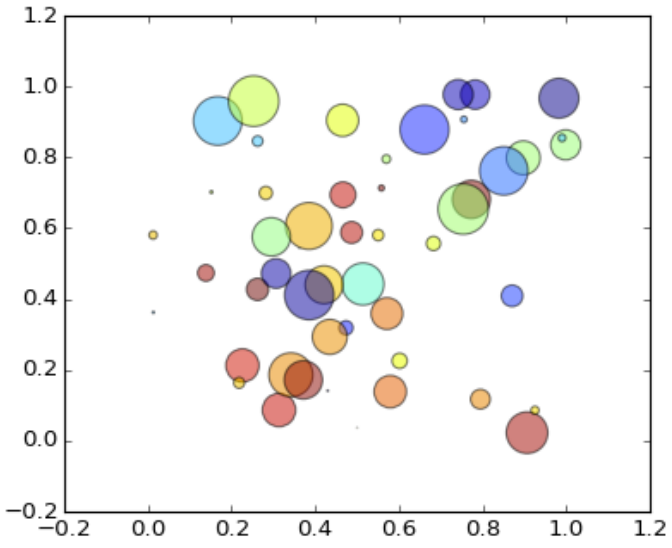
- Seaborn is a package that sits on top of Matplotlib
- Updates default formats to be more visually appealing (similar to R default chart formats)
- Update your iPython Notebook for Matplotlib and add this to the beginning of notebook:

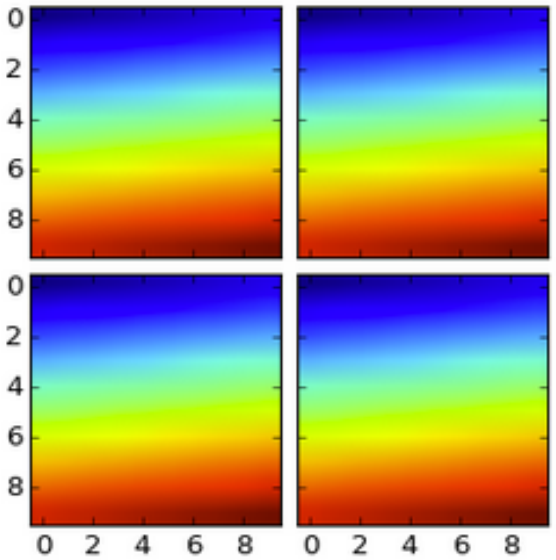
```
import seaborn
```

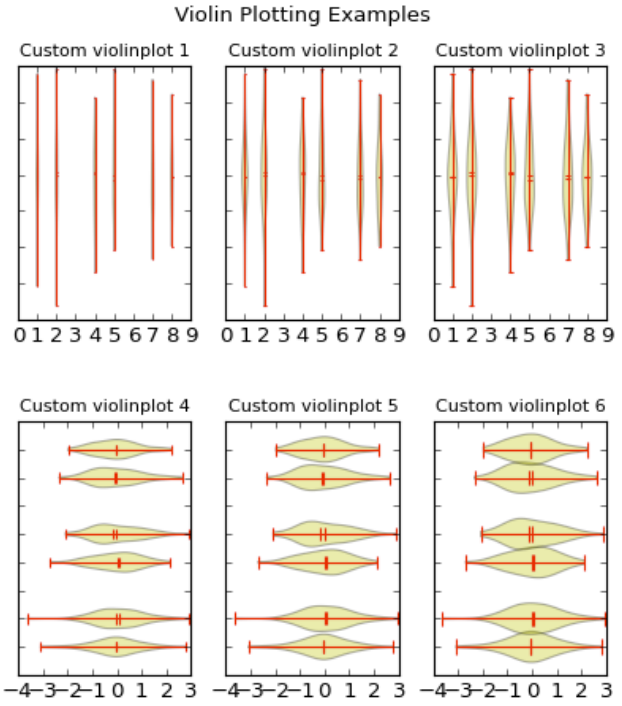
- Reset kernel and run all

Matplotlib Gallery

- http://matplotlib.org/gallery.html







Exercise

- Create a figure with 4 subplots
  - Histogram of Titanic age distribution
  - Scatterplot of Age and Ticket Fares
  - Boxplot of Ages for Survivors vs. Deceased
  - Boxplot of Ages for Men vs. Women
  - Save figure as a png file
- Pick a chart from the Matplotlib Gallery and recreate it in your ipython notebook

Exception Handling

- Useful technique for handling errors without breaking your code

- Basic exception handling contains two parts:
  - Code to try and execute
  - Separate instructions to execute if the initial code can’t complete successfully

Create a Function to Test

Create a function that takes a list of numbers and returns a list of squared numbers

```
def squared(values):
    results = []
    for x in values:
        sq = x **2
        results.append(sq)
    return results
```

Test Function

Try the function on a list of numbers

```
print squared([2,4,6,8,10])
```

What happens when you pass a string in the list of values?

```
print squared([2,4,6,8,'cat'])
```

Add Exception Handling

Add try and except clauses

```
def squared_2(values):
    results = []
    for x in values:
        try:
            sq = x **2
            results.append(sq)
        except Exception as e:
            error_msg = "Error on {0}-{1}".format(x,e)
            results.append(error_msg)
    return results
```

Test the Exception Handling

Try the new function with the same data set

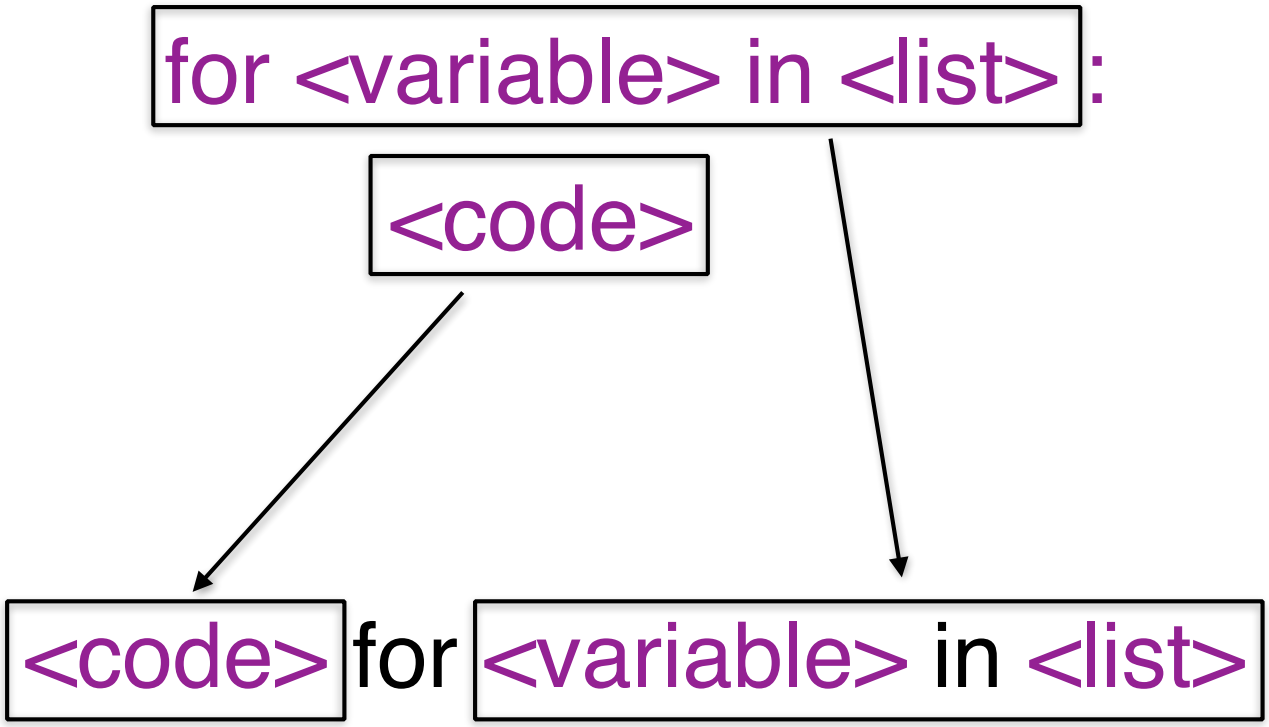
```
print squared_2([2,4,6,8,'cat'])
```

List Comprehensions

- Compact code to reduce nesting and accomplish tasks with fewer commands
- Leverages single line commands for:
  - For Loops
  - If Statements
- Considered Pythonic technique
- Complicated to understand at first, but becomes really useful as your coding skills improve

For Loop Compressed

Original For Loop



Compressed For Loop

List Comprehension Example #1

Long Format

```
example_1 =[]
for x in [1,2,3,4]:
    calculation = x ** 2
    example_1.append(calculation)
print example_1
```

List Comprehension

```
[x ** 2 for x in [1,2,3,4]]
```

List Comprehension Example #2

Long Format

```
example_2 = []
for x in [1,2,3,4]:
    if x%2 == 0:
        example_2.append(x)
print example_2
```

List Comprehension

```
[x for x in [1,2,3,4] if x%2==0]
```

List Comprehension Example #3

Long Format

```
example_3 = []
for x in [1,2,3,4]:
    if x%2==0:
        example_3.append('even')
    else:
        example_3.append('odd')
print example_3
```

List Comprehension

```
['even' if x%2==0 else 'odd' for x in [1,2,3,4]]
```

Exercise

- Create a list comprehension that converts temperatures from Celsius to Fahrenheit for 0 to 100 in increments of 10
- Update your list comprehension to check the data type and return an error message if the data type isn't numeric