Synopsis

This notebook will explain the following topics and concepts:

Ranking Data

- Ascending and Descending

Concatenation

- Rows and Columns

Merging Data

- left
- right
- inner
- outer

Joining Data

- left
- right
- inner
- outer

Grouping Data

- by time
- by columns

Importing libraries & Load Data

We'll use the same csv files as we used in chapter 3.

In [2]:

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

# use matplotlib inline jupyter magic
%matplotlib inline


# format for floats
pd.options.display.float_format = '{:,.2f}'.format
```

Import a number of DataFrames from different sheets in the same excel workbook

In [3]:

```
df_GOOGL = pd.read_excel('https://s3.eu-west-1.amazonaws.com/neueda.conygre.com/pydat
df_IBM = pd.read_excel('https://s3.eu-west-1.amazonaws.com/neueda.conygre.com/pydata/
df_MSFT = pd.read_excel('https://s3.eu-west-1.amazonaws.com/neueda.conygre.com/pydata
```

Concatenation

- Glues together DataFrames, without much intelligence.
- Dimensions should match along the axis you are concatenating on.
- Use **pd.concat** and pass in a list of DataFrames to concatenate together

Create a few simple DataFrames

In [4]:

```
# To demonstrate concatenation and merging, we'll first select some small subsets of
# This is just to give us Data that more suited to explaining things!!

date_range = pd.date_range(start='2017', freq='BQ', periods=4)
cols   = ['Open', 'Close']

df1 = df_IBM.reindex(date_range)[cols]
df2 = df_GOOGL.reindex(date_range)[cols]
df3 = df_MSFT.reindex(date_range)[cols]
```

Concatenate

The default is to concatenate the rows

In [5]:

```
# concatenate all three of df1, df2 and df3 along the default axis
df_all = pd.concat([df1, df2, df3])
df_all
```

Out[5]:

	Open	Close
2017-03-31	173.98	174.14
2017-06-30	154.28	153.83
2017-09-29	145.45	145.08
2017-12-29	154.17	153.42
2017-03-31	846.83	847.80
2017-06-30	943.99	929.68
2017-09-29	966.00	973.72
2017-12-29	1,055.49	1,053.40
2017-03-31	65.65	65.86
2017-06-30	68.78	68.93
2017-09-29	73.94	74.49
2017-12-29	85.63	85.54

Concatenate the columns

Pass the `axis = 1` parameter to pd.concat

In [11]:

```
df_all = pd.concat([df1, df2, df3], axis=1)
df_all
```

Out[11]:

	Open	Close	Open	Close	Open	Close
2017-03-31	173.98	174.14	846.83	847.80	65.65	65.86
2017-06-30	154.28	153.83	943.99	929.68	68.78	68.93
2017-09-29	145.45	145.08	966.00	973.72	73.94	74.49
2017-12-29	154.17	153.42	1,055.49	1,053.40	85.63	85.54

Merging

Pandas has two important functions for joining DataFrames together which intelligently try to align values from selected columns of each DataFrame. These functions are called **merge** and **join**. These functions use a similar logic to joins in SQL.

First we will look at merge.

There are 4 Different types of merge

- **Inner Merge** – The default Pandas behaviour, only keep rows where the merge “on” value exists in both the left and right DataFrames.
- **Left Merge** – (aka left merge or left join) Keep every row in the left DataFrame. Where there are missing values of the “on” variable in the right DataFrame, add empty / NaN values in the result.
- **Right Merge** – (aka right merge or right join) Keep every row in the right DataFrame. Where there are missing values of the “on” variable in the left column, add empty / NaN values in the result.
- **Outer Merge** – A full outer join returns all the rows from the left DataFrame, all the rows from the right DataFrame, and matches up rows where possible, with NaNs elsewhere.

Create some sample DataFrames

Just a few days worth of Data from Google and IBM

Note the difference in date ranges

In [5]:

```
cols = ['High', 'Low']

df1 = df_IBM[cols]['2017-Jan-01':'2017-Jan-06'].sort_index()
df2 = df_GOOGL[cols]['2017-Jan-05':'2017-Jan-10'].sort_index()

# show both dataframes
print("== IBM ==")
display(df1)
print("== GOOGLE ==")
display(df2)
```

```
== IBM ==
```

	High	Low
Date		
2017-01-03	167.87	166.01
2017-01-04	169.87	167.36
2017-01-05	169.39	167.26
2017-01-06	169.92	167.52

```
== GOOGLE ==
```

	High	Low
Date		
2017-01-05	813.74	805.92
2017-01-06	828.96	811.50
2017-01-09	830.43	821.62
2017-01-10	829.41	823.14

Inner Merge

Only keep values for Dates found in both left (df1) and right (df2)

In [ ]:

```
# This version of merge works in latest Pandas
pd.merge(df1, df2, how='inner', on='Date')

# This version of merge works in older Pandas
pd.merge(df1, df2, how='inner', left_index=True, right_index=True)
```

Left Merge

- Keep everything in the left DataFrame.
- Where nothing exists in the right DataFrame, fill with NaN ("Not a Number" - these are empty values).
- Use the suffixes parameter to override the x *and y* defaults

In [ ]:

```
srcs = ['_IBM','_GOOGL']

pd.merge(df1, df2, how='left', on='Date', suffixes=srcs)

pd.merge(df1, df2, how='left', left_index=True, right_index=True,  suffixes=srcs)
```

Right Merge

- Keep everything in the right DataFrame
- Where nothing exists in the left DataFrame, fill with NaN

In [ ]:

```
pd.merge(df1, df2, how='right', on='Date', suffixes=srcs)

pd.merge(df1, df2, how='right', left_index=True, right_index=True, suffixes=srcs)
```

Outer Merge

Keep everything in both left and right DataFrames, fill with NaN where no data present

In [ ]:

```
pd.merge(df1, df2, how='outer', on='Date', suffixes=srcs)

pd.merge(df1, df2, how='outer', on='Date', left_index=True, right_index=True, suffixe
```

Joining

- The second pandas function for intelligently combining DataFrames is called **join**.
- Join is **very** similar to merge.
- As with merge, the **how** parameter takes inner, outer, left or right.
- As with merge, the **on** parameter is the name of a column to join on.
- However there is one major difference:
  - When using join the "on" **must** be the index in at least one of the DataFrames.
  - Merge will allow the "on" to be a regular column in **both** DataFrames.

The syntax for calling the two functions is also slightly different:

- **join** : df1.join(df2, how="inner", on="Date")
- **merge** : pd.merge(df1, df2, how="inner", on="Date")

In [7]:

```
# If we don't specify an "on" then we will join "on" the index of both DataFrames
df1.join(df2, lsuffix='_df1')
```

Out[7]:

	Open_df1	Close_df1	Open	Close
2017-03-31	173.98	174.14	846.83	847.80
2017-06-30	154.28	153.83	943.99	929.68
2017-09-29	145.45	145.08	966.00	973.72
2017-12-29	154.17	153.42	1,055.49	1,053.40

In [8]:

```
# right join
df1.join(df2, how='right', lsuffix='_df1')
```

Out[8]:

	Open_df1	Close_df1	Open	Close
2017-03-31	173.98	174.14	846.83	847.80
2017-06-30	154.28	153.83	943.99	929.68
2017-09-29	145.45	145.08	966.00	973.72
2017-12-29	154.17	153.42	1,055.49	1,053.40

In [9]:

```
# inner join
df1.join(df2, how='inner', lsuffix='_df1')
```

Out[9]:

	Open_df1	Close_df1	Open	Close
2017-03-31	173.98	174.14	846.83	847.80
2017-06-30	154.28	153.83	943.99	929.68
2017-09-29	145.45	145.08	966.00	973.72
2017-12-29	154.17	153.42	1,055.49	1,053.40

In [10]:

```
# outer join
df1.join(df2, how='outer', lsuffix='_df1')
```

Out[10]:

	Open_df1	Close_df1	Open	Close
2017-03-31	173.98	174.14	846.83	847.80
2017-06-30	154.28	153.83	943.99	929.68
2017-09-29	145.45	145.08	966.00	973.72
2017-12-29	154.17	153.42	1,055.49	1,053.40

Grouping Data

Pandas provides functions that allow us to group rows of data together and call aggregate functions on them as a unit e.g. mean, max, min, std, etc.

To create a group we call the **groupby** method on a DataFrame.

e.g. Create groups where the 'Industry' column is the same:

```
- df1.groupby('Industry')
```

Group by Columns

Use the **by** parameter and supply a column name

In [12]:

```
df = pd.read_excel(io='https://s3.eu-west-1.amazonaws.com/neueda.conygre.com/pydata/s
                   sheet_name='Groups', index_col='Date', parse_dates=True)
```

In [13]:

```
# Grouping is a convenient way to get the mean for each sector of each column
df.groupby(by='Sector').mean()

# Grouping is a convenient way to get the mean for each sector of each column
df.groupby(by='Rep').mean()

# Grouping is a convenient way to get the mean for each sector of each column
df.groupby(by='Portfolio').mean()

# Or we could create a variable to store the name of the function
func = 'std'
df.groupby(by='Sector').agg(func)

# Or we can create a list of functions to aggregate with
funcs = ['median', 'std']
df.groupby(by='Rep').agg(funcs)
```

Out[13]:

	High		Low	
	median	std	median	std
Rep				
George	85.10	158.88	61.96	89.81
John	81.29	213.04	55.50	154.95
Paul	90.88	101.06	65.15	62.97
Ringo	87.32	121.71	61.09	81.32

Group by Time Period

Often we want to group according to a frequency e.g. a group of all values in a single business quarter.

We can then call mean for all of the groups, e.g. get the mean Volume per business quarter.

A convenient way to group by a frequency is to use the Grouper object (in the pandas namespace).

- pd.Grouper
- Most commonly used to pass in a frequency

- Group by frequencies: B (business day), BQ (business quarter), M (month), Y (year), etc.
- It's also possible to group by specific time frequencies e.g. 15d, 1h30min etc.
    - See the list of frequency aliases: http://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#offset-aliases

In [ ]:

```
df.groupby(pd.Grouper(freq='BQ')).mean()

# Or we could create a variable to store the grouper
by_BMonth = pd.Grouper(freq='BM')
df.groupby(by=by_BMonth).mean()

# Or use teh agg and a list of functions
funcs = ['median', 'std']
df.groupby(by=by_BMonth).agg(funcs)


# Finally we can supply a list of groupers
by_BMonth = pd.Grouper(freq='BM')
by_Rep = pd.Grouper(key='Rep')
by_Sector = pd.Grouper(key='Sector')

# And perform a variety of slice and dice operations
groups1 = [by_Rep, by_BMonth, by_Sector]
df.groupby(by=groups1).agg(funcs)

groups2 = [by_Sector, by_Rep, by_BMonth]
df.groupby(by=groups2).agg(funcs)

groups3 = [by_BMonth, by_Sector, by_Rep]
df.groupby(by=groups3).agg(funcs)
```

In [ ]:

```
# try some more complex grouping with plotting after e.g. group by Quarter, get the m
df_GOOGL['Adj. Open'].groupby(pd.Grouper(freq='Q')).mean()
df_GOOGL['Adj. Open'].groupby(pd.Grouper(freq='Q')).mean().plot()

df_GOOGL['2017']['Adj. Open'].groupby(pd.Grouper(freq='d')).mean()
df_GOOGL['2017']['Adj. Open'].rolling(30).mean()
df_GOOGL['2017']['Adj. Open'].rolling(30, min_periods=3).mean()

df_GOOGL['2017']['Adj. Open'].groupby(pd.Grouper(freq='15d')).mean()
df_GOOGL['Adj. Open'].groupby(pd.Grouper(freq='BQ')).mean()

df_GOOGL['Adj. Open'].groupby(pd.Grouper(freq='BQ')).mean()
```