Pandas basics - Numpy, and standard library

This notebook gives examples of basic Pandas constructs and operations.

- Creating a DataFrame
  - specifying Schema
  - specifying column/row index (headers)
- Merge (concat) of DataFrames
- Pivot
- Reduce, GroupBy, MultiIndex
- Iterators
- Comparing to SQL

This notebook is available at https://github.com/cwinsor/pandas_gold.git (https://github.com/cwinsor/pandas_gold.git)

In [1]:
```
import pandas as pd
import numpy as np
```

row reduction (axis=1)

In [2]:
```
arr = np.array([[1,5],[2,5],[3,5]])
df = pd.DataFrame(arr, columns=["col1", "col2"])
print(df)
print()
print(df.sum(axis=1))
```

```
   col1  col2
0     1     5
1     2     5
2     3     5

0    6
1    7
2    8
dtype: int64
```

Reduce and "groupBy"

A table can be split into subset called groups. Reduce can give a summary of each group.

In [3]:
```
df = pd.DataFrame({'Animal': ['Falcon', 'Falcon',
                              'Parrot', 'Parrot'],
                   'Max Speed': [380., 370., 24., 26.]})
df
```

Out[3]:

	Animal	Max Speed
0	Falcon	380.0
1	Falcon	370.0
2	Parrot	24.0
3	Parrot	26.0

In [4]:
```
df.groupby(['Animal']).mean()
```

Out[4]:

	Max Speed
Animal	
Falcon	375.0
Parrot	25.0

using multiIndex ...

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.groupby.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.groupby.html)

In [5]:
```
arrays = [['Falcon', 'Falcon', 'Parrot', 'Parrot'],
          ['Captive', 'Wild', 'Captive', 'Wild']]
index = pd.MultiIndex.from_arrays(arrays, names=('Animal', 'Type'))
df = pd.DataFrame({'Max Speed': [390., 350., 30., 20.]},
                  index=index)
df
```

Out[5]:

		Max Speed
Animal	**Type**	
Falcon	**Captive**	390.0
	Wild	350.0
Parrot	**Captive**	30.0
	Wild	20.0

In [6]:
```
df.groupby(level=0).mean()
```

Out[6]:

	Max Speed
Animal	
Falcon	370.0
Parrot	25.0

In [7]:
```
1 df.groupby(level="Type").mean()
```

Out[7]:

Type	Max Speed
Captive	210.0
Wild	185.0

Pivot (reshape)

https://pandas.pydata.org/pandas-docs/stable/user_guide/reshaping.html (https://pandas.pydata.org/pandas-docs/stable/user_guide/reshaping.html)

The example from that reference is below. But we start with easier examples.

In summary:

1. "and" of values between two nominal columns (col_1 value and col_2 value) -> filter
2. "or" of values between two nominal columns (col_1 value or col_2 value) -> filter
3. "or' of values within one nominal column (col_1 value or col_1 value) -> filter, then pivot

In [8]:
```
1  # example from https://pandas.pydata.org/pandas-docs/stable/user_guide/resha
2  df = pd.DataFrame({'foo': ['one', 'one', 'one', 'two', 'two', 'two'],
3                     'bar': ['A', 'B', 'C', 'A', 'B', 'C'],
4                     'baz': ['1', '2', '3', '4', '5', '6'],
5                    'zoo': ['x', 'y', 'z', 'q', 'w', 't']})
6  print(df)
7  df.pivot(index="foo", columns="bar", values="baz")
8
```

```
   foo bar baz zoo
0  one   A   1   x
1  one   B   2   y
2  one   C   3   z
3  two   A   4   q
4  two   B   5   w
5  two   C   6   t
```

Out[8]:

bar	A	B	C
foo			
one	1	2	3
two	4	5	6

In [9]:
```
df = pd.DataFrame({'Category_1': [100, 100, 100, 101, 101, 102],
                   'Category_2': ["apple", 'pear', 'banana', 'apple', 'pear'
                   'Value': [380., 370., 24., 26., 12., 28.]})
df
```

Out[9]:

	Category_1	Category_2	Value
0	100	apple	380.0
1	100	pear	370.0
2	100	banana	24.0
3	101	apple	26.0
4	101	pear	12.0
5	102	apple	28.0

In [10]:
```
df.pivot(index="Category_1", columns="Category_2", values="Value")
```

Out[10]:

Category_2	apple	banana	pear
Category_1			
100	380.0	24.0	370.0
101	26.0	NaN	12.0
102	28.0	NaN	NaN

Merge/Join of DataFrames (the .concat method)

See https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html (https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html)

Specifying Schema when Creating a DataFrame

In [11]:
```
import pandas as pd
schema = {
    'left': int,
    'center_y': float,
    'original_file_path': str,
    "is_origin": bool}

df_objects = pd.DataFrame(columns=schema.keys()).astype(schema)
df_objects.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 0 entries
Data columns (total 4 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   left                0 non-null      int32  
 1   center_y            0 non-null      float64
 2   original_file_path  0 non-null      object 
 3   is_origin           0 non-null      bool   
dtypes: bool(1), float64(1), int32(1), object(1)
memory usage: 0.0+ bytes
```

Iterating through a DataFrame

In [12]:
```
for file_num, file_info in df_objects.iterrows():
    print("image {} of {}".format(file_num, df_objects.shape[0]))
    #print(file_info)

    if file_info['cassette_has_42_tp']:
```

```
  File "<ipython-input-12-f2ff16f626ba>", line 5
    if file_info['cassette_has_42_tp']:
                                       ^
SyntaxError: unexpected EOF while parsing
```

Appending to DataFrame

In [13]:
```
# the example is from 004b_auto_labeler.ipynb  in "opencv tutorial" git
def df_from_image(image):
    count, labels, stats, centr = cv.connectedComponentsWithStats(image)
    df = pd.DataFrame(stats, columns=['left', 'top', 'width', 'height', 'are
    # convenience add right/bottom and center x/y
    df['bottom'] = df['top'] + df['height'] - 1
    df['right'] = df['left'] + df['width'] - 1
    df['center_y'] = centr[:,1]
    df['center_x'] = centr[:,0]
    df['area_bb'] = df['width'] * df['height']
    df['density'] = df['area_px'] / df['area_bb']
    return df

# Find connected components
objects_this_image = df_from_image(img_masked)

# append to the overall list of objects
df_objects = df_objects.append(objects_this_image, ignore_index=True)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-13-590ad4ce2608> in <module>
     13
     14 # Find connected components
---> 15 objects_this_image = df_from_image(img_masked)
     16
     17 # append to the overall list of objects

NameError: name 'img_masked' is not defined
```

Pandas has amazing functions for data analysis, series, plotting...

- Rolling windows: https://pandas.pydata.org/pandas-docs/stable/reference/window.html (https://pandas.pydata.org/pandas-docs/stable/reference/window.html)
- Series types and functions: https://pandas.pydata.org/pandas-docs/stable/reference/series.html (https://pandas.pydata.org/pandas-docs/stable/reference/series.html)
- General functions (e.g. Pivot, time/date) https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.pivot.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.pivot.html)
- Plotting: https://pandas.pydata.org/pandas-docs/stable/reference/plotting.html (https://pandas.pydata.org/pandas-docs/stable/reference/plotting.html)

Examples:

- df["column"].unique()
- df.isnull().sum()

- df.corr() (de-factors scale)
- df.cov() (preserves units)

- df.kurt() # kurtosis - a measure of 'tailness' of sample, from 1 to infinity. A.normal distribution measures "3".

- df.info()
- df describe()
- df.describe(include=['O']) (include categorical)
- df.hist()

In [14]:
```
# https://towardsdatascience.com/let-us-understand-the-correlation-matrix-an
```

Relational Data in the context of Data Science (including Pandas and SQLite)

https://www.datasciencecourse.org/notes/relational_data/ (https://www.datasciencecourse.org/notes/relational_data/)

Pandas vs SQL (command mappings)

https://pandas.pydata.org/pandas-docs/stable/getting_started/comparison/comparison_with_sql.html (https://pandas.pydata.org/pandas-docs/stable/getting_started/comparison/comparison_with_sql.html)

Pandas basics

In [15]:
```
import pandas as pd
df = pd.DataFrame({'A': [1,2,3,4],
                   'B': [10,20,30,40],
                   'C': [20,40,60,80]
                  })
#                   index=['Row 1', 'Row 2', 'Row 3', 'Row 4'])
df
```

Out[15]:

	A	B	C
0	1	10	20
1	2	20	40
2	3	30	60
3	4	40	80

In [16]:
```
# The primary purpose of the DataFrame indexing operator, [] is to select co
df['B']
```

Out[16]:
```
0    10
1    20
2    30
3    40
Name: B, dtype: int64
```

In [17]:
```
# The DataFrame indexing operator completely changes behavior to select rows
# Strangely, when given a slice, the DataFrame indexing operator selects row
df[1:3]
```

Out[17]:

	A	B	C
1	2	20	40
2	3	30	60

.loc and .iloc

It is recommended use .loc or .iloc for all indexing

- .iloc is position indexing
- .loc is label indexing

these return a dataFrame

In [18]:
```
df.loc[:,['A','C']]
```

Out[18]:

	A	C
0	1	20
1	2	40
2	3	60
3	4	80

In [19]:
```
import numpy as np
```

In [20]:
```
df = pd.DataFrame(np.random.rand(5,2),index=range(0,10,2),columns=list('AB'))

print(df)
print()
print(df.iloc[[2]])
print()
print(df.loc[[2]])
```

```
          A         B
0  0.746499  0.413450
2  0.647250  0.159546
4  0.859715  0.656277
6  0.787922  0.219401
8  0.000393  0.689714

          A         B
4  0.859715  0.656277

         A         B
2  0.64725  0.159546
```

In [21]:
```
# to access the value of a single element - use at and iat
# iat is position indexing
# at is label indexing

print(df)
print()
print(df.iat[2,1])
print()
print(df.at[2,'B'])
```

```
          A         B
0  0.746499  0.413450
2  0.647250  0.159546
4  0.859715  0.656277
6  0.787922  0.219401
8  0.000393  0.689714

0.6562770631378325

0.15954632379274025
```

Make a (numpy) array from DataFrame, or list from Sequence

In [22]:
```
import pandas as pd
df = pd.DataFrame({'A': [1,2,3,4],
                   'B': [10,20,30,40],
                   'C': [20,40,60,80]
                  })
df
```

Out[22]:

	A	B	C
0	1	10	20
1	2	20	40
2	3	30	60
3	4	40	80

In [23]:
```
the_list = df['B'].values
print(type(the_list))
the_list
```

```
<class 'numpy.ndarray'>
```

Out[23]:
```
array([10, 20, 30, 40], dtype=int64)
```

"List comprehensions" and "enumerate"...

This is traditional 'looping' code...

Examples from https://mlwhiz.com/blog/2019/04/22/python_forloops/ (https://mlwhiz.com/blog/2019/04/22/python_forloops/)

In [24]:
```
### Yuk
x = [1,3,5,7,9]
sum_squared = 0
for i in range(len(x)):
    sum_squared+=x[i]**2
```

In [25]:
```
### OK
x = [1,3,5,7,9]
sum_squared = 0
for y in x:
    sum_squared+=y**2
```

In [26]:
```
### Enumerate
L = ['blue', 'yellow', 'orange']
for i, val in enumerate(L):
    print("index is %d and value is %s" % (i, val))
```

```
index is 0 and value is blue
index is 1 and value is yellow
index is 2 and value is orange
```

In [27]:
```
### List Comprehension
x = [1,3,5,7,9]
squared = [y**2 for y in x]
```

In [28]:
```
### List Comprehension with "if"
x = [1,2,3,4,5,6,7,8,9]
even_squared = [y**2 for y in x if y%2==0]
```

In [29]:
```
### List Comprehension with "if/else" and "for" and reduction "sum"
x = [1,2,3,4,5,6,7,8,9]
squared_cubed_sum = sum([y**2 if y%2==0 else y**3 for y in x])
```

In [ ]:
```

```