My Pandas DataFrame Hands on Codes

. . . by swarnadeep

Importing Libraries

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

Making a Student database `studentData` with a 'dict' `data`

In [2]:
```
data = {
    'name' : ['amal', 'bimal', 'suneet', 'swarna', 'jeetu'],
    'course' : ['ece', 'cs', 'it', 'ece', 'me'],
    'marks' : [70, 80, 90, 100, 60]
}
```

In [3]:
```
studentData = pd.DataFrame(data)
studentData
```

Out[3]:

	name	course	marks
0	amal	ece	70
1	bimal	cs	80
2	suneet	it	90
3	swarna	ece	100
4	jeetu	me	60

Specifying an Index and make another database `studentDataWithIndex`

In [4]:
```
studentDataWithIndex = pd.DataFrame(data, index=['a','b','c','d','e'])
studentDataWithIndex
```

Out[4]:

	name	course	marks
a	amal	ece	70
b	bimal	cs	80
c	suneet	it	90
d	swarna	ece	100
e	jeetu	me	60

`.info()` and `.describe()` methods produce some important information about the data

In [5]:
```
studentData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   name    5 non-null      object
 1   course  5 non-null      object
 2   marks   5 non-null      int64
dtypes: int64(1), object(2)
memory usage: 248.0+ bytes
```

In [6]:
```
studentData.describe()
```

Out[6]:

	marks
count	5.000000
mean	80.000000
std	15.811388
min	60.000000
25%	70.000000
50%	80.000000
75%	90.000000
max	100.000000

Extracting Information by Attributes

Using `.shape` , `.ndim` , `.elements` to extract data

In [7]:
```
# We print some information about shopping_carts
print('Shape:', studentData.shape)
print('Dimension:', studentData.ndim)
print('Total Size : ',studentData.size,' elements')
```

```
Shape: (5, 3)
Dimension: 2
Total Size :  15  elements
```

Getting `Values` , `Indexes` & `Columns` from DataFrame

In [8]:
```
print('The data in studentDataWithIndex is:\n', studentDataWithIndex.values)
print()
print('The row index in studentDataWithIndex is:', studentDataWithIndex.index)
print()
print('The column index in studentDataWithIndex is:', studentDataWithIndex.columns)
```

```
The data in studentDataWithIndex is:
 [['amal' 'ece' 70]
 ['bimal' 'cs' 80]
 ['suneet' 'it' 90]
 ['swarna' 'ece' 100]
 ['jeetu' 'me' 60]]

The row index in studentDataWithIndex is: Index(['a', 'b', 'c', 'd', 'e'], dtype='object')

The column index in studentDataWithIndex is: Index(['name', 'course', 'marks'], dtype='object')
```

Exporting data to a CSV File by `.to_csv()` and Reading CSV by `.read_csv()`

In [9]:
```
studentData.to_csv('studentData.csv', index = False)
```

In [10]:
```
studentDataWithIndex.to_csv('studentDataWithIndex.csv')
```

In [11]:
```
pd.read_csv('studentData.csv')
```

Out[11]:

	name	course	marks
0	amal	ece	70
1	bimal	cs	80
2	suneet	it	90
3	swarna	ece	100
4	jeetu	me	60

In [12]:
```
pd.read_csv('studentData.csv',index_col = 'name')
```

Out[12]:

	course	marks
name		
amal	ece	70
bimal	cs	80
suneet	it	90
swarna	ece	100
jeetu	me	60

`Random` DataFrame and `Transpose` in Pandas

In [13]:
```
newdf = pd.DataFrame(np.random.rand(5,10))
```

In [14]:
```
newdf
```

Out[14]:

	0	1	2	3	4	5	6	7	8	9
0	0.059922	0.000491	0.449921	0.324782	0.507193	0.240135	0.341536	0.052719	0.017614	0.933931
1	0.150177	0.811267	0.138901	0.324032	0.581915	0.992713	0.303574	0.429948	0.388371	0.197264
2	0.715923	0.821184	0.569175	0.196939	0.030910	0.619481	0.757978	0.574092	0.289384	0.279372
3	0.433589	0.695339	0.446048	0.026728	0.665372	0.096266	0.556915	0.048511	0.200502	0.098683
4	0.816158	0.296369	0.083093	0.860637	0.497140	0.923827	0.683001	0.902646	0.975896	0.334106

In [15]:
```
# Transpose of a DataFrame
newdf.T
```

Out[15]:

	0	1	2	3	4
0	0.059922	0.150177	0.715923	0.433589	0.816158
1	0.000491	0.811267	0.821184	0.695339	0.296369
2	0.449921	0.138901	0.569175	0.446048	0.083093
3	0.324782	0.324032	0.196939	0.026728	0.860637
4	0.507193	0.581915	0.030910	0.665372	0.497140
5	0.240135	0.992713	0.619481	0.096266	0.923827
6	0.341536	0.303574	0.757978	0.556915	0.683001
7	0.052719	0.429948	0.574092	0.048511	0.902646
8	0.017614	0.388371	0.289384	0.200502	0.975896
9	0.933931	0.197264	0.279372	0.098683	0.334106

Sorting Row(`axis=0`) and Column(`axis=1`) by `sort_index`

In [16]:
```
newdf.sort_index(axis=1, ascending = False)
```

Out[16]:

	9	8	7	6	5	4	3	2	1	0
0	0.933931	0.017614	0.052719	0.341536	0.240135	0.507193	0.324782	0.449921	0.000491	0.059922
1	0.197264	0.388371	0.429948	0.303574	0.992713	0.581915	0.324032	0.138901	0.811267	0.150177
2	0.279372	0.289384	0.574092	0.757978	0.619481	0.030910	0.196939	0.569175	0.821184	0.715923
3	0.098683	0.200502	0.048511	0.556915	0.096266	0.665372	0.026728	0.446048	0.695339	0.433589
4	0.334106	0.975896	0.902646	0.683001	0.923827	0.497140	0.860637	0.083093	0.296369	0.816158

Another Exciting Example from Udacity

Creating a new DataFrame `shopping_carts`

In [17]:
```
items = {'Bob' : pd.Series(data = [245, 25, 55], index = ['bike', 'pants', 'watch']),
         'Alice' : pd.Series(data = [40, 110, 500, 45], index = ['book', 'glasses', 'bike', 'pants'])}
# We create a Pandas DataFrame by passing it a dictionary of Pandas Series
shopping_carts = pd.DataFrame(items)

# We display the DataFrame
shopping_carts
```

Out[17]:

	Bob	Alice
bike	245.0	500.0
book	NaN	40.0
glasses	NaN	110.0
pants	25.0	45.0
watch	55.0	NaN

In [18]:
```
# Creating a dictionary of Pandas Series without indexes
datas = {'Bob' : pd.Series([245, 25, 55]),
        'Alice' : pd.Series([40, 110, 500, 45])}

# We create a DataFrame
df = pd.DataFrame(datas)

# We display the DataFrame
df
```

Out[18]:

	Bob	Alice
0	245.0	40
1	25.0	110
2	55.0	500
3	NaN	45

Selecting elements as per `index` and `column`

In [19]:
```
pd.DataFrame(shopping_carts, index = ['glasses', 'bike'], columns = ['Alice'])
```

Out[19]:

	Alice
glasses	110.0
bike	500.0

Creating a DataFrame from list of Python Dictionaries

In [20]:
```
# We create a list of Python dictionaries
items2 = [{'bikes': 20, 'pants': 30, 'watches': 35},
          {'watches': 10, 'glasses': 50, 'bikes': 15, 'pants':5}]

# We create a DataFrame  and provide the row index
store_items = pd.DataFrame(items2, index = ['store 1', 'store 2'])

# We display the DataFrame
store_items
```

Out[20]:

	bikes	pants	watches	glasses
store 1	20	30	35	NaN
store 2	15	5	10	50.0

Accessing Elements in pandas DataFrames

In [21]:
```
# How many bikes and pants are in each store:
store_items[['bikes', 'pants']]
```

Out[21]:

	bikes	pants
store 1	20	30
store 2	15	5

In [22]:
```
# What items are in Store 1:
store_items.loc[['store 1']]
```

Out[22]:

	bikes	pants	watches	glasses
store 1	20	30	35	NaN

In [23]:
```
# How many bikes are in Store 2 :
# Format : dataframe[column][row]
store_items['bikes']['store 2']
```

Out[23]: 15

Adding a external Column named `shirts` in Dataframe

Can be done using `.insert()` function (Demonstrated Later)

In [24]:
```
# Adding a Shirt Column and put 15 shirts in store 1 and 2 shirts in store 2
store_items['shirts'] = [15,2]
store_items
```

Out[24]:

	bikes	pants	watches	glasses	shirts
store 1	20	30	35	NaN	15
store 2	15	5	10	50.0	2

Adding a arithmetic Column named `suits` = (`pants` + `shirts`)

In [25]:
```
store_items['suits'] = store_items['pants'] + store_items['shirts']
store_items
```

Out[25]:

	bikes	pants	watches	glasses	shirts	suits
store 1	20	30	35	NaN	15	45
store 2	15	5	10	50.0	2	7

Creating a Row named `store 3` , which is to be appended to `store_items`

In [26]:
```
new_items = [{'bikes': 20, 'pants': 30, 'watches': 35, 'glasses': 4}]
new_store = pd.DataFrame(new_items, index = ['store 3'])
new_store
```

Out[26]:

	bikes	pants	watches	glasses
store 3	20	30	35	4

Appending `store 3` to `store_items`

In [27]:
```
store_items = store_items.append(new_store)
store_items
```

Out[27]:

	bikes	pants	watches	glasses	shirts	suits
store 1	20	30	35	NaN	15.0	45.0
store 2	15	5	10	50.0	2.0	7.0
store 3	20	30	35	4.0	NaN	NaN

We can also add new columns by using data of existing columns

In [28]:
```
store_items['new watches'] = store_items['watches'][1:]
store_items
```

Out[28]:

	bikes	pants	watches	glasses	shirts	suits	new watches
store 1	20	30	35	NaN	15.0	45.0	NaN
store 2	15	5	10	50.0	2.0	7.0	10.0
store 3	20	30	35	4.0	NaN	NaN	35.0

Some DataFrame Functions

`.insert()` , `.pop()` , `.drop()` , `.rename()` , `.set_index()`

`df.insert(loc,label,data)` : Used to insert a new column

In [29]:
```
store_items.insert(4, 'shoes', [8,5,0])
store_items
```

Out[29]:

	bikes	pants	watches	glasses	shoes	shirts	suits	new watches
store 1	20	30	35	NaN	8	15.0	45.0	NaN
store 2	15	5	10	50.0	5	2.0	7.0	10.0
store 3	20	30	35	4.0	0	NaN	NaN	35.0

`df.pop('column_name')` : Used to delete a column(only)

In [30]:
```
store_items.pop('new watches')
store_items
```

Out[30]:

	bikes	pants	watches	glasses	shoes	shirts	suits
store 1	20	30	35	NaN	8	15.0	45.0
store 2	15	5	10	50.0	5	2.0	7.0
store 3	20	30	35	4.0	0	NaN	NaN

`df.drop('column_name' , axis = 0 or 1)` : deletes both rows and columns by `axis` keyword

In [31]:
```
# Deleting 2 Columns using axis = 1
store_items = store_items.drop(['watches', 'shoes'], axis = 1)
store_items
```

Out[31]:

	bikes	pants	glasses	shirts	suits
store 1	20	30	NaN	15.0	45.0
store 2	15	5	50.0	2.0	7.0
store 3	20	30	4.0	NaN	NaN

In [32]:
```
# Deleting 1 Row using axis = 0
store_items = store_items.drop('store 2', axis = 0)
store_items
```

Out[32]:

	bikes	pants	glasses	shirts	suits
store 1	20	30	NaN	15.0	45.0
store 3	20	30	4.0	NaN	NaN

`df.rename(columns = {Before_Col_Name : After_Col_Name})` : Used to rename a Column Label

In [33]:
```
# Changing the column label 'bikes' to 'hats'
store_items = store_items.rename(columns = {'bikes': 'hats'})
store_items
```

Out[33]:

	hats	pants	glasses	shirts	suits
store 1	20	30	NaN	15.0	45.0
store 3	20	30	4.0	NaN	NaN

`df.rename(index = {Before_Row_Name : After_Row_Name})` : Used to rename a Row Label

In [34]:
```
# Changing a Row label 'store 3' to 'last store'
store_items = store_items.rename(index = {'store 3': 'last store'})
store_items
```

Out[34]:

	hats	pants	glasses	shirts	suits
store 1	20	30	NaN	15.0	45.0
last store	20	30	4.0	NaN	NaN

`df.set_index('Column_name')` : Used to rename a Row Label

In [35]:
```
# Changing the row index to be the data in the pants column
store_items.set_index('pants')
```

Out[35]:

	hats	glasses	shirts	suits
pants				
30	20	NaN	15.0	45.0
30	20	4.0	NaN	NaN

`.groupby()` : Splits the data into groups based on some criteria

Syntax : `.groupby('Column-name')['Column-name'].aggregate_function()`

In [36]:
```
# print value counts for each user type
# user_types = df.groupby('User Type')['User Type'].count()
```