Learnings from the Lecture Materials || Page 117

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

In [2]:
```
myseries = pd.Series([1, -3, 5, -20]) # note capital S in series
myseries
```

Out[2]:
```
0     1
1    -3
2     5
3   -20
dtype: int64
```

In [3]:
```
print(type(myseries))
```

```
<class 'pandas.core.series.Series'>
```

In [4]:
```
myseries.index
```

Out[4]:
```
RangeIndex(start=0, stop=4, step=1)
```

In [5]:
```
udo = pd.Series([1, -3, 5, -20], index = ['K','i','n','g'])
udo
```

Out[5]:
```
K     1
i    -3
n     5
g   -20
dtype: int64
```

In [6]:
```
udo.values
```

Out[6]:
```
array([  1,  -3,   5, -20], dtype=int64)
```

In [7]:
```
udo.index
```

Out[7]:
```
Index(['K', 'i', 'n', 'g'], dtype='object')
```

In [8]:
```
print(udo[2])
print(udo['K'])
print(udo[0:3])
print(myseries[2])
print(udo[['i','n','g']])
```

```
5
1
K    1
i   -3
n    5
dtype: int64
5
i    -3
n     5
g   -20
dtype: int64
```

In [9]:
```
udo['n'] = 10
```

In [10]:
```
udo
```

Out[10]:
```
K     1
i    -3
n    10
g   -20
dtype: int64
```

Converting an ARRAY to a Pandas SERIES

In [12]:
```
myarray = np.array([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15])
print(myarray)
print(myarray[0:10]) # Accessing the first 10 elements in the array
myseries3 = pd.Series(myarray) # Converting the array to pandas Series
myseries3 # Calling the series so it can be displayed
print(myseries3)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
[ 1  2  3  4  5  6  7  8  9 10]
0      1
1      2
2      3
3      4
4      5
5      6
6      7
7      8
```

```
8      9
9     10
10    11
11    12
12    13
13    14
14    15
dtype: int32
```

In [13]:
```
myseries3[myseries3 != 8]
```

Out[13]:
```
0      1
1      2
2      3
3      4
4      5
5      6
6      7
8      9
9     10
10    11
11    12
12    13
13    14
14    15
dtype: int32
```

In [14]:
```
np.log(myseries3)
```

Out[14]:
```
0     0.000000
1     0.693147
2     1.098612
3     1.386294
4     1.609438
5     1.791759
6     1.945910
7     2.079442
8     2.197225
9     2.302585
10    2.397895
11    2.484907
12    2.564949
13    2.639057
14    2.708050
dtype: float64
```

In [15]:
```
mycolors = pd.Series([1,2,3,4,5,4,3,2,1], index = ['white','black','blue',
                                                   'green','green','yellow',
```

```
                                                'black','red','purple'])
mycolors
```

Out[15]:
```
white     1
black     2
blue      3
green     4
green     5
yellow    4
black     3
red       2
purple    1
dtype: int64
```

In [16]:
```
mycolors.unique()
```

Out[16]:
```
array([1, 2, 3, 4, 5], dtype=int64)
```

In [17]:
```
mycolors.value_counts()
```

Out[17]:
```
1    2
2    2
3    2
4    2
5    1
dtype: int64
```

In [18]:
```
mycolors.isin([7,6,1,3])
```

Out[18]:
```
white      True
black     False
blue       True
green     False
green     False
yellow    False
black      True
red       False
purple     True
dtype: bool
```

In [19]:
```
mydict = {'White':1000, 'Black':500,'Red':200,'Green':1000}
myseries = pd.Series(mydict)
myseries
```

Out[19]:
```
White    1000
Black     500
```

```
Red       200
Green    1000
dtype: int64
```

Dealing with Pandas DataFrame

In [20]:
```
mydata = {'Employee Name': ['Jerry','Tom','Jack','John','Alicia'],
          'Specialization': ['Python','Data Science','Data Preparation',
                             'Cloud Computing','Web Development'],
          'Experience (years)': [3,5,8,2,4]}
myframe = pd.DataFrame(mydata) # Note capital D and F in the DataFrame
myframe
```

Out[20]:

	Employee Name	Specialization	Experience (years)
0	Jerry	Python	3
1	Tom	Data Science	5
2	Jack	Data Preparation	8
3	John	Cloud Computing	2
4	Alicia	Web Development	4

In [21]:
```
print(type(myframe))
```

```
<class 'pandas.core.frame.DataFrame'>
```

__There is an alternative way to create a DataFrame.__ > And that is to ___pass input arguments to the DataFrame Constructor___ in the following order: 1. a data matrix 2. an array of the labels for the indices (index option) 3. an array containing the column names (columns option) We use the $np.arange()$ to create the array. And use the $reshape()$ function to convert the array to a $matrix$

In [22]:
```
# Note that the range requires it multiples as the reshape values
matrix5by8 = np.arange(40).reshape((5,8))
matrix5by8
```

Out[22]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29, 30, 31],
       [32, 33, 34, 35, 36, 37, 38, 39]])
```

In [23]:
```
arrayA = np.arange(15) # This creates the array
```

```
matA = arrayA.reshape((3,5)) # This reshapes the array to a matrix
matA
```

Out[23]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

In [24]:
```
arrayA = np.arange(15) # This creates the array
matA = arrayA.reshape((3,5)) # This reshapes the array to a matrix
print(arrayA)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
```

__We have created a 2 dimensional array or a $matrix$ of size $FiveRows$ and $EightColumns$__ and __now, we can pass the $matrix$ into a $DataFrame$ constructor__

In [25]:
```
newFrame = pd.DataFrame(np.arange(40).reshape((5,8)),
                        index = ['Record-1','Record-2','Record-3','Record-4',
                                 'Record-5'],
                        columns = ['Field-1','Field-2','Field-3',
                                   'Field-4','Field-5','Field-6','Field-7',
                                   'Field-8'])
newFrame
```

Out[25]:

	Field-1	Field-2	Field-3	Field-4	Field-5	Field-6	Field-7	Field-8
Record-1	0	1	2	3	4	5	6	7
Record-2	8	9	10	11	12	13	14	15
Record-3	16	17	18	19	20	21	22	23
Record-4	24	25	26	27	28	29	30	31
Record-5	32	33	34	35	36	37	38	39

In [26]:
```
newFrame2 = pd.DataFrame(matrix5by8)
newFrame2
```

Out[26]:

	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	8	9	10	11	12	13	14	15

	0	1	2	3	4	5	6	7
2	16	17	18	19	20	21	22	23
3	24	25	26	27	28	29	30	31
4	32	33	34	35	36	37	38	39

Having our data in a $DataFrame$ makes it ready for __manipulation__ and __preparation__ so that the data can be effortlessly $analyzed$ and $manipulated$. We could carry out the following operations on a $DataFrame$ 1. Concatenation _using_ $pandas.concat()$ to concatenate the objects along the $x-axis$ 2. Merging _using_ $pandas.merge()$ function to connect the rows in a DataFrame __based on one or more keys__ by implementing join operations. 3. Combination _using_ $pandas.DataFrame.combine\_first()$ function to connect overlapped data to fill in missing values in a data structure.

Concatenating Data

Concatenation is linking together two or more separate data structures (DataFrame, Array etc) and placing them next to each to make single entity.

> NOTE: $Numpy$ has a concatenate function and $Pandas$ also has a concatenate function

In [27]:
```
myseriesA = pd.Series(np.random.rand(5), index = [0,1,2,3,4])
# myseries = pd.Series(np.random.rand(5))
myseriesA
```

Out[27]:
```
0    0.198460
1    0.375178
2    0.742061
3    0.323173
4    0.213468
dtype: float64
```

In [28]:
```
myseriesB = pd.Series(np.random.rand(5), index = [4,6,7,8,9])
myseriesB
```

Out[28]:
```
4    0.322362
6    0.199016
7    0.167997
8    0.343535
9    0.288352
dtype: float64
```

In [29]:
```
# Concatenating the two Series that we have generated above.
pd.concat([myseriesA,myseriesB])
```

Out[29]:
```
0    0.198460
1    0.375178
2    0.742061
3    0.323173
4    0.213468
4    0.322362
6    0.199016
7    0.167997
8    0.343535
9    0.288352
dtype: float64
```

__NOTE:__ the $concat()$ functions _by default_ works on $axis = 0$, which means rows to return a series. **We can set $axis = 1$ *which* denotes $columns$ for the pandas to concatenate as a DataFrame.** An example of this is illustrated in the code below.

In [30]:
```
# Setting axis = 1 (columns) to realize a dataFrame
pd.concat([myseriesA,myseriesB], axis = 1)
```

Out[30]:

	0	1
0	0.198460	NaN
1	0.375178	NaN
2	0.742061	NaN
3	0.323173	NaN
4	0.213468	0.322362
6	NaN	0.199016
7	NaN	0.167997
8	NaN	0.343535
9	NaN	0.288352

We can use the $keys$ option along the axis = 1, to set the column names for the DataFrame. This is illustrated in the code below.

In [31]:
```
df = pd.concat([myseriesA,myseriesB], axis = 1, keys = ['RowOne', 'RowTwo'])
df
```

Out[31]:

	RowOne	RowTwo

	RowOne	RowTwo
0	0.198460	NaN
1	0.375178	NaN
2	0.742061	NaN
3	0.323173	NaN
4	0.213468	0.322362
6	NaN	0.199016
7	NaN	0.167997
8	NaN	0.343535
9	NaN	0.288352

__Removing the NaN or null values from the $DataFrame$ using the $dropna()$ function.__ __We can also use the $df = df[df['RowOne'].notna()]$ to remove null values from a specific column__

In [32]:
```
df_notNull = df.dropna(subset = ['RowOne', 'RowTwo'])
df_notNull
```

Out[32]:

	RowOne	RowTwo
4	0.213468	0.322362

In [33]:
```
df_one = df.dropna(subset = ['RowOne'])
df_one
```

Out[33]:

	RowOne	RowTwo
0	0.198460	NaN
1	0.375178	NaN
2	0.742061	NaN
3	0.323173	NaN
4	0.213468	0.322362

In [34]:
```
df_two = df.dropna(subset = ['RowTwo'])
df_two
```

Out[34]:

	RowOne	RowTwo
4	0.213468	0.322362
6	NaN	0.199016
7	NaN	0.167997
8	NaN	0.343535
9	NaN	0.288352

__Using the $pandas.concat()$ function on a $DataFrame$__

In [35]:
```
myframeA = pd.DataFrame({'Student Name': ['I\'mKing Udonyah',
                                          'Precious Kingsley','Esther Xttle'],
                         'Sex':['Male','Female','Female'], 'Age': [30,7,22],
                         'School':['Post Graduate','Primary','College']})
myframeA
```

Out[35]:

	Student Name	Sex	Age	School
0	I'mKing Udonyah	Male	30	Post Graduate
1	Precious Kingsley	Female	7	Primary
2	Esther Xttle	Female	22	College

In [36]:
```
myframeB = pd.DataFrame({'Student Name': ['Favour Christopher','Joshua Pius','Steve Inem'],
                         'Sex':['Female','Male','Male'], 'Age': [21,33,25],
                         'School':['College','Vocational','Post Graduate'],
                        'Hostel': ['Green','Red','White']})
myframeB
```

Out[36]:

	Student Name	Sex	Age	School	Hostel
0	Favour Christopher	Female	21	College	Green
1	Joshua Pius	Male	33	Vocational	Red
2	Steve Inem	Male	25	Post Graduate	White

__Concatenating the two $DataFrames$ using the $pandas.concat()$ function__

In [37]:
```
concated_df = pd.concat([myframeA, myframeB])
concated_df
```

Out[37]:

	Student Name	Sex	Age	School	Hostel
0	I'mKing Udonyah	Male	30	Post Graduate	NaN
1	Precious Kingsley	Female	7	Primary	NaN
2	Esther Xttle	Female	22	College	NaN
0	Favour Christopher	Female	21	College	Green
1	Joshua Pius	Male	33	Vocational	Red
2	Steve Inem	Male	25	Post Graduate	White

__Replacing the $nullvalues$ with specified values using the $fillna()$ function__

In [38]:
```
newConcated_df = concated_df.fillna('Off Campus')
newConcated_df
```

Out[38]:

	Student Name	Sex	Age	School	Hostel
0	I'mKing Udonyah	Male	30	Post Graduate	Off Campus
1	Precious Kingsley	Female	7	Primary	Off Campus
2	Esther Xttle	Female	22	College	Off Campus
0	Favour Christopher	Female	21	College	Green
1	Joshua Pius	Male	33	Vocational	Red
2	Steve Inem	Male	25	Post Graduate	White

5.4.2 Merging Data

Merging data consists of combining data through the connection of rows using one or more keys.
The keys are **common columns** in the DataFrame to be merged.

Based on this **keys**, it is possible to obtain new data in a tabular form
The $merge()$ function to perform this kind of operation.

Let us merge $myframeA$ and $myframeB$ using the *codes line* illustrated below

In [39]:
```
bookFrame1 = pd.DataFrame({'Student Name': ['A','B','C'],
                           'Sex':['M','F','M'], 'Age': [10,16,17],
                           'School':['Primary','High','High']})
bookFrame1
```

Out[39]:

	Student Name	Sex	Age	School
0	A	M	10	Primary
1	B	F	16	High
2	C	M	17	High

In [40]:
```
bookFrame2 = pd.DataFrame({'Student Name': ['D','E','A'],
                           'Class':['9','10','5'],
                             'School':['Primary','High','High']})
bookFrame2
```

Out[40]:

	Student Name	Class	School
0	D	9	Primary
1	E	10	High
2	A	5	High

In [41]:
```
mergedFrame = pd.merge(bookFrame1,bookFrame2) # Do not include square brackets
mergedFrame
```

Out[41]:

Student Name	Sex	Age	School	Class

In [42]:
```
mergeFrame = pd.merge(bookFrame1,bookFrame2, how = 'right')
# Try using 'left', 'right','inner' in how
mergeFrame
```

Out[42]:

Student Name	Sex	Age	School	Class

	Student Name	Sex	Age	School	Class
0	D	NaN	NaN	Primary	9
1	E	NaN	NaN	High	10
2	A	NaN	NaN	High	5

__The merge operation merges only those columns together for which the _key entries_ $School$ are the same.__ __In a situation where we have multiple columns, we can merge on the basis of one particular column.__ __We use the on option to specify the key for merging the data.__

In [43]:
```
mergedSchool = pd.merge(bookFrame1,bookFrame2, on = 'School')
# mergedSchool = pd.merge(myframeA,myframeB, on = 'School')
mergedSchool.head(10)
```

Out[43]:

	Student Name_x	Sex	Age	School	Student Name_y	Class
0	A	M	10	Primary	D	9
1	B	F	16	High	E	10
2	B	F	16	High	A	5
3	C	M	17	High	E	10
4	C	M	17	High	A	5

__Merging using a different key $StudentName$__

In [44]:
```
mergedSchool = pd.merge(bookFrame1,bookFrame2, on = 'Student Name')
mergedSchool
```

Out[44]:

	Student Name	Sex	Age	School_x	Class	School_y
0	A	M	10	Primary	5	High

__We can merge $DataFrames$ based on indices using $join$.__ For this to be possible, __neither of the DataFrames should have the same column names__. __Therefore, let us rename the column names of one of the DataFrames we created earlier from our Textbook__ And then merge it with the other as illustrated below

In [45]:
```
# Renaming the columns of the DataFrame
bookFrame2.columns = ['StudentNames','Class','Institution'] #Renames bookFrame2
joinedFrame = bookFrame1.join(bookFrame2)
joinedFrame
```

Out[45]:

	Student Name	Sex	Age	School	StudentNames	Class	Institution
0	A	M	10	Primary	D	9	Primary
1	B	F	16	High	E	10	High
2	C	M	17	High	A	5	High

5.4.3 Combining Data Let's say we have two datasets with __overlapping indices__ and we want to keep values from one of the datasets if an overlapping index comes during combining these. If the index is not overlapping, then its value is kept. __This cannot be obtained by either merging or with concatenation.__ We use the $combine\_first()$ function provided by the $Pandas$ library to perform this kind of operation. The code cells below illustrates this:

In [45]:
```
seriesAA = pd.Series([50,40,30,20,10], index = [1,2,3,4,5])
seriesAA
```

Out[45]:
```
1    50
2    40
3    30
4    20
5    10
dtype: int64
```

In [46]:
```
seriesBB = pd.Series([100,200,300,400], index = [3,4,5,6])
seriesBB
```

Out[46]:
```
3    100
4    200
5    300
6    400
dtype: int64
```

In [47]:
```
# To keep the values from seriesAA, we combine both series
seriesAA.combine_first(seriesBB)
```

Out[47]:
```
1     50.0
2     40.0
3     30.0
```

```
4     20.0
5     10.0
6    400.0
dtype: float64
```

In [48]:
```
# To keep the values from seriesBB, we combine both series
seriesBB.combine_first(seriesAA)
```

Out[48]:
```
1     50.0
2     40.0
3    100.0
4    200.0
5    300.0
6    400.0
dtype: float64
```

5.5 Data Transformation

This simply involves the removal or replacement of duplicate or invalid values respectively.

The aim here is to handle **outliers** and **missing values**

In the following subsections, we shall be discussing and working on *data transformation techniques*

<div = class = "alert alert-block alert-success">

REMOVING UNWANTED DATA AND DUPLICATES

In [69]:
```
# Deleting a field using the del command
del joinedFrame['School']
joinedFrame
```

Out[69]:

	Student Name	Sex	Age	StudentNames	Class	Institution
0	A	M	10	D	9	Primary
1	B	F	16	E	10	High
2	C	M	17	A	5	High

In [50]:
```
# Removing a row using an index
joinedFrame.drop(1)
```

Out[50]:

	Student Name	Sex	Age	School	StudentNames	Institution
0	A	M	10	Primary	D	Primary
2	C	M	17	High	A	High

REMOVING DUPLICATES __Dupicate rows in a dataset do not convey extra information.__ These extra rows consume extra memory and are termed as $redundant$. Also, processing these extra records adds to the cost of computations. Therefore, it is desirable to remove duplicates rows from the data. Below, we create a DataFrame with duplicate row

In [51]:
```
item_frame = pd.DataFrame({'Items':['Ball','Bat','Hockey','Football','Ball'],
                           'Color':['White','Gray','White','Red','White'],
                            'Price':[100,500,700,200,100]})
item_frame
```

Out[51]:

	Items	Color	Price
0	Ball	White	100
1	Bat	Gray	500
2	Hockey	White	700
3	Football	Red	200
4	Ball	White	100

__We see from the DataFrame above, that the rows indexed 0 and 4 are duplicates__ __To find duplicates rows, we use the $duplicated()$ function__

In [52]:
```
item_frame.duplicated()
```

Out[52]:
```
0    False
1    False
2    False
3    False
4     True
dtype: bool
```

__We can also display the duplicated entries only__ __To achieve this, we use the $item\_frame.duplicated()$ as the index to the DataFrame__

In [53]:
```
item_frame[item_frame.duplicated()]
```

Out[53]:

	Items	Color	Price
4	Ball	White	100

__We can remove the duplicate entries, we can use the following commands__

In [54]:

```
item_frame.drop_duplicates()
```

Out[54]:

	Items	Color	Price
0	Ball	White	100
1	Bat	Gray	500
2	Hockey	White	700
3	Football	Red	200

HANDLING OUTLIERS __Outliers are values that are outside the expected range of a feature. Outliers are commonly caused by:__ 1. Human errors during data entry; 2. Measurement (instrument) errors; 3. Experimental errors during data extraction or manipulation; 4. Intentional errors to test the accuracy of the outlier detection methods. __It is the responsibity of the Data Scientist, during $DataPreparation$ and $Analysis$ to detect the presence of unexpected values within a data structure.__

In [62]:

```
student_frame = pd.DataFrame({'Student Name': ['A','B','C','D','E','F','G'],
                              'Sex':['M','F','M','F','F','M','M'],
                                   'Age': [10,14,60,15,16,15,11],
                              'School':['Primary','High','High','High','High','High','Primary']})
student_frame
```

Out[62]:

	Student Name	Sex	Age	School
0	A	M	10	Primary
1	B	F	14	High
2	C	M	60	High
3	D	F	15	High
4	E	F	16	High
5	F	M	15	High

	Student Name	Sex	Age	School
6	G	M	11	Primary

__From the above DataFrame, the age of student C is unexpected hence it is regarded as an $outlier$.__ __We use the $describe()$ function to surmise the important statistical values of our DataFrame.__ __NOTE: it is only the statistical summary of fields with numerical values that are calculated.__

In [63]:
```
statSummary = student_frame.describe()
statSummary
```

Out[63]:

	Age
count	7.000000
mean	20.142857
std	17.714670
min	10.000000
25%	12.500000
50%	15.000000
75%	15.500000
max	60.000000

NOTE

1. The **statistical count** gives the number of elements in the fields.
2. The **mean** gives the average value
3. The **std** provides the *standard deviation*. The standard deviation is the average deviation of data points from the mean of the data.
4. The **min** is the minimum value.
5. The **max** is maximum value.
6. The 25% , 50% and 75% gives the **25th percentile (the first Quartile - Q1) and the 50th percentile(the median - Q2), and the 75th percentile (the third Quartile - Q3)** of the values

Determination of Outliers in a DataSet

We can find the outliers in a dataset using the InterQuartile Range (IQR)

> $IQR = Q3 - Q1$ and that is, $15.5 - 12.5 = 3$

Using an interquartile multiplier value of $k = 1.5$, we find the lower & upper values beyond which data points can be considered as outliers.

> $IQR * 1.5 = 4.5$

Now, we subtract 4.5 from Q1 to find the lower limit and add 4.5 to Q3 to find the upper limit. Thus,

> **Lower Limit** = $Q1 - 4.5 = 8$
> **Upper Limit** = $Q3 + 4.5 = 20$

Any value less than 8 or greater than 20 is an outlier

In [64]:
```
Q1 = student_frame.quantile(0.25) # 25% (quartile 1)
Q3 = student_frame.quantile(0.75) # 75% (quartile 3)

IQR = Q3 - Q1
IQR_mult = IQR * 1.5

lowerL = Q1 - IQR_mult
upperL = Q3 + IQR_mult

print("The lower limit is ", lowerL)
print("The upper limit is ", upperL)
```

```
The lower limit is  Age    8.0
dtype: float64
The upper limit is  Age    20.0
dtype: float64
```

In [65]:
```
student_frame = student_frame[student_frame['Age'] > int(lowerL)]
student_frame = student_frame[student_frame['Age'] < int(upperL)]
student_frame
```

Out[65]:

	Student Name	Sex	Age	School
0	A	M	10	Primary
1	B	F	14	High
3	D	F	15	High
4	E	F	16	High
5	F	M	15	High

	Student Name	Sex	Age	School
6		G	M	11	Primary

Handling Missing or Invalid Data

In [66]:
```
myseries4 = pd.Series([10,20,30, None, 40, 50, np.NaN],
                      index = [0,1,2,3,4,5,6])
print(myseries4.isnull())
myseries4
```

```
0    False
1    False
2    False
3     True
4    False
5    False
6     True
dtype: bool
```

Out[66]:
```
0    10.0
1    20.0
2    30.0
3     NaN
4    40.0
5    50.0
6     NaN
dtype: float64
```

In [67]:
```
# To get the row or indices containing the NULL VALUES
myseries4[myseries4.isnull()]
```

Out[67]:
```
3   NaN
6   NaN
dtype: float64
```

In [68]:
```
# Dropping the NULL VALUES
myseries4_clean = myseries4.dropna()
myseries4_clean
```

Out[68]:
```
0    10.0
1    20.0
2    30.0
4    40.0
5    50.0
dtype: float64
```

In [ ]: