Concatenate a Series

Concatenate a Series

- A python Series is created using the pandas library as shown.

```
import pandas as pd
series = pd.Series([1,2,3,4,5])
type(series)
```

```
pandas.core.series.Series
```

```
# Create two series using linspace() and concatenate them
x = np.linspace(0,20, 11)
y = np.linspace(1,21, 11)
# x and y are numpy arrays
# first convert them to pandas series
x_series = pd.Series(x)
y_series = pd.Series(y)
print("The type of x_series is :", type(x_series), "\nAnd the type of y_series is : ", type(y_series))
```

```
The type of x_series is : <class 'pandas.core.series.Series'>
And the type of y_series is :  <class 'pandas.core.series.Series'>
```

- Concatenating a Series is appending a Series to another to give one Series
- Using the function pd.concat() and Series.append()
- More than two Series can be concatenated

Concatenate a Series using concat()

The concat() function concatenates pandas Series in the order they are passed in the function

```
# Concatenate using concat()
pd.concat([x_series,y_series])
```

```
0      0.0
1      2.0
2      4.0
3      6.0
4      8.0
5     10.0
6     12.0
7     14.0
8     16.0
9     18.0
10    20.0
0      1.0
1      3.0
2      5.0
3      7.0
4      9.0
5     11.0
6     13.0
7     15.0
8     17.0
9     19.0
10    21.0
dtype: float64
```

Adding hierarchical indexing and labelling the index

```
# adding a hierarchical index
# labeling the Index
pd.concat([x_series, y_series], keys = ['Even', 'Odd'],names=['Category', 'Index'])
```

```
Category  Index
Even      0          0.0
          1          2.0
          2          4.0
          3          6.0
          4          8.0
          5         10.0
          6         12.0
          7         14.0
          8         16.0
          9         18.0
          10        20.0
Odd       0          1.0
          1          3.0
          2          5.0
          3          7.0
          4          9.0
          5         11.0
          6         13.0
          7         15.0
          8         17.0
          9         19.0
          10        21.0
dtype: float64
```

Concatenate a Series using append()

```
# append x_series by y_series
x_series.append(y_series)
```

```
0       0.0
1       2.0
2       4.0
3       6.0
4       8.0
5      10.0
6      12.0
7      14.0
8      16.0
9      18.0
10     20.0
0       1.0
1       3.0
2       5.0
3       7.0
4       9.0
5      11.0
6      13.0
7      15.0
8      17.0
9      19.0
10     21.0
dtype: float64
```

```
# append y_series by x_series
y_series.append(x_series)
```

```
0       1.0
1       3.0
2       5.0
3       7.0
4       9.0
5      11.0
6      13.0
7      15.0
8      17.0
9      19.0
10     21.0
0       0.0
1       2.0
2       4.0
3       6.0
4       8.0
5      10.0
6      12.0
7      14.0
8      16.0
9      18.0
10     20.0
dtype: float64
```

Concatenate the DataFrames

Concatenating DataFrames

- A Pandas DataFrame is a two dimensional size-mutable, heterogeneous data structure with labeled rows and columns
- DataFrames can be concatenated vertically (column-wise) and horizontally (row-wise)
- The functions used to concatenate Data Frames are pd.concat() and pd.append()

Concatenating Dataframes

To concatenate DataFrames let us first load the DataFrames

```
# load the data from 'Sheet1' of the xlsx file and print it
df1 = pd.read_excel('example.xlsx',  sheet_name=0 )
print(type(df1))
df1
```

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

	Age	Gender	Salary	City_Residence
0	45.0	Male	40000	NaN
1	12.0	Male	0	Banglore
2	NaN	Female	150000	Banglore
3	26.0	Male	30000	Chennai
4	64.0	Female	15000	Chennai

```
# load the data from 'Sheet2' of the xlsx file and print it
df2 = pd.read_excel('example.xlsx',  sheet_name=1 )
print(type(df2))
df2
```

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

	Age	Gender	Salary	City_Residence
0	45	Male	88900	NaN
1	23	NaN	18000	Mumbai
2	67	Male	92000	Mumbai
3	34	Male	180000	Delhi
4	67	Male	92000	Mumbai
5	34	Male	180000	Delhi

Concatenating DataFrames using concat()

- The two Data Frames are concatenated using concat()
- They are concatenated along the axis = 0
- The concatenation is in the order they are passed in the function
- The index numbers of the concatenated Data Frame are of the actual data frames df1 and df2

```
pd.concat([df1,df2])
```

	Age	Gender	Salary	City_Residence
0	45.0	Male	40000	NaN
1	12.0	Male	0	Banglore
2	NaN	Female	150000	Banglore
3	26.0	Male	30000	Chennai
4	64.0	Female	15000	Chennai
0	45.0	Male	88900	NaN
1	23.0	NaN	18000	Mumbai
2	67.0	Male	92000	Mumbai
3	34.0	Male	180000	Delhi
4	67.0	Male	92000	Mumbai
5	34.0	Male	180000	Delhi

Concatenating DataFrames using concat()

Using the function concat(), it is possible to concatenate more than two Data Frames

```
pd.concat([df1,df2,df1])
```

	Age	Gender	Salary	City_Residence
0	45.0	Male	40000	NaN
1	12.0	Male	0	Banglore
2	NaN	Female	150000	Banglore
3	26.0	Male	30000	Chennai
4	64.0	Female	15000	Chennai
0	45.0	Male	88900	NaN
1	23.0	NaN	18000	Mumbai
2	67.0	Male	92000	Mumbai
3	34.0	Male	180000	Delhi
4	67.0	Male	92000	Mumbai
5	34.0	Male	180000	Delhi
0	45.0	Male	40000	NaN
1	12.0	Male	0	Banglore
2	NaN	Female	150000	Banglore
3	26.0	Male	30000	Chennai
4	64.0	Female	15000	Chennai

Concatenating DataFrames using concat()

- They are concatenated along the axis = 1
- The concatenation is in the order they are passed in the function
- The lengths of the two data frames are different. It is seen that NaN's are displayed for the index numbers where data is not available
- The concatenation is in the order they are passed in the function

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

	Age	Gender	Salary	City_Residence	Age	Gender	Salary	City_Residence
0	45.0	Male	40000.0	NaN	45	Male	88900	NaN
1	12.0	Male	0.0	Banglore	23	NaN	18000	Mumbai
2	NaN	Female	150000.0	Banglore	67	Male	92000	Mumbai
3	26.0	Male	30000.0	Chennai	34	Male	180000	Delhi
4	64.0	Female	15000.0	Chennai	67	Male	92000	Mumbai
5	NaN	NaN	NaN	NaN	34	Male	180000	Delhi

Concatenating DataFrames using concat()

- They are concatenated along the axis = 0
- The concatenation is in the order they are passed in the function
- The argument 'ignore_index' is set to 'True' to assign the index numbers beginning with 0

```
pd.concat([df1,df2],axis =0)
```

	Age	Gender	Salary	City_Residence
0	45.0	Male	40000	NaN
1	12.0	Male	0	Banglore
2	NaN	Female	150000	Banglore
3	26.0	Male	30000	Chennai
4	64.0	Female	15000	Chennai
0	45.0	Male	88900	NaN
1	23.0	NaN	18000	Mumbai
2	67.0	Male	92000	Mumbai
3	34.0	Male	180000	Delhi
4	67.0	Male	92000	Mumbai
5	34.0	Male	180000	Delhi

Concatenating DataFrames using concat()

- Multiple DataFrames can be concatenated using the concat function
- Here we have concatenated df1, df2 and then df1 again

```
pd.concat([df1,df2,df1])
```

	Age	Gender	Salary	City_Residence
0	45.0	Male	40000	NaN
1	12.0	Male	0	Banglore
2	NaN	Female	150000	Banglore
3	26.0	Male	30000	Chennai
4	64.0	Female	15000	Chennai
0	45.0	Male	88900	NaN
1	23.0	NaN	18000	Mumbai
2	67.0	Male	92000	Mumbai
3	34.0	Male	180000	Delhi
4	67.0	Male	92000	Mumbai
5	34.0	Male	180000	Delhi
0	45.0	Male	40000	NaN
1	12.0	Male	0	Banglore
2	NaN	Female	150000	Banglore
3	26.0	Male	30000	Chennai
4	64.0	Female	15000	Chennai

Concatenating DataFrames using append()

```
# append df1 to df2
df1.append(df2)
```

	Age	Gender	Salary	City_Residence
0	45.0	Male	40000	NaN
1	12.0	Male	0	Banglore
2	NaN	Female	150000	Banglore
3	26.0	Male	30000	Chennai
4	64.0	Female	15000	Chennai
0	45.0	Male	88900	NaN
1	23.0	NaN	18000	Mumbai
2	67.0	Male	92000	Mumbai
3	34.0	Male	180000	Delhi
4	67.0	Male	92000	Mumbai
5	34.0	Male	180000	Delhi

```
# append df1 to df2
df2.append(df1)
```

	Age	Gender	Salary	City_Residence
0	45.0	Male	88900	NaN
1	23.0	NaN	18000	Mumbai
2	67.0	Male	92000	Mumbai
3	34.0	Male	180000	Delhi
4	67.0	Male	92000	Mumbai
5	34.0	Male	180000	Delhi
0	45.0	Male	40000	NaN
1	12.0	Male	0	Banglore
2	NaN	Female	150000	Banglore
3	26.0	Male	30000	Chennai
4	64.0	Female	15000	Chennai

Concatenating multiple Data Frames using append()

The append function can not append more than two Data Frames

```
# concatenation of multiple DataFrames using .append()
# append df1 and df2 to df2
df2.append(df1,df2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-22-234a24432a1c> in <module>
      1 # concatenation of multiple DataFrames using .append()
      2 # append df1 and df2 to df2
----> 3 df2.append(df1,df2)

/opt/anaconda3/lib/python3.7/site-packages/pandas/core/frame.py in append(self, other, ignore_index, verify_integrit
y, sort)
   7121             ignore_index=ignore_index,
   7122             verify_integrity=verify_integrity,
-> 7123             sort=sort,
   7124         )
   7125

/opt/anaconda3/lib/python3.7/site-packages/pandas/core/reshape/concat.py in concat(objs, axis, join, join_axes, ignor
e_index, keys, levels, names, verify_integrity, sort, copy)
    253         verify_integrity=verify_integrity,
    254         copy=copy,
--> 255         sort=sort,
    256     )
    257

/opt/anaconda3/lib/python3.7/site-packages/pandas/core/reshape/concat.py in __init__(self, objs, axis, join, join_axe
s, keys, levels, names, ignore_index, verify_integrity, copy, sort)
    426         self.copy = copy
    427
--> 428         self.new_axes = self._get_new_axes()
    429
    430     def get_result(self):

/opt/anaconda3/lib/python3.7/site-packages/pandas/core/reshape/concat.py in _get_new_axes(self)
    520                 new_axes[i] = ax
    521
--> 522         new_axes[self.axis] = self._get_concat_axis()
    523         return new_axes
    524

/opt/anaconda3/lib/python3.7/site-packages/pandas/core/reshape/concat.py in _get_concat_axis(self)
    568             indexes = [x._data.axes[self.axis] for x in self.objs]
    569
--> 570         if self.ignore_index:
    571             idx = ibase.default_index(sum(len(i) for i in indexes))
    572             return idx

/opt/anaconda3/lib/python3.7/site-packages/pandas/core/generic.py in __nonzero__(self)
   1553             "The truth value of a {0} is ambiguous. "
   1554             "Use a.empty, a.bool(), a.item(), a.any() or a.all().".format(
-> 1555                 self.__class__.__name__
   1556             )
   1557

ValueError: The truth value of a DataFrame is ambiguous. Use a.empty, a.bool(), a.item(), a.any() or a.all().
```

Concatenating using merge()

Concatenating using merge()

- .merge() connects rows in Data Frames based on one or more keys
- If the column on which to join is not specified, merge uses the overlapping column names as the keys

We shall use the following data frames:

```
# load the data from 'Sheet1' of the xlsx file and print it
df1 = pd.read_excel('example_merge.xlsx',  sheet_name=0 )
print(type(df1))
df1
```

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

	Age	Gender	Salary	City_Residence
0	45	Male	40000	Mumbai
1	12	Male	0	Banglore
2	54	Female	150000	Banglore
3	26	Male	30000	Delhi
4	64	Female	15000	Chennai

```
# load the data from 'Sheet2' of the xlsx file and print it
df2 = pd.read_excel('example_merge.xlsx',  sheet_name=1 )
print(type(df2))
df2
```

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

	Age	Gender	Salary	City_Residence
0	45	Male	88900	Delhi
1	23	Female	18000	Mumbai
2	67	Male	92000	Mumbai
3	34	Male	180000	Banglore

Concatenating using merge()

- The merge function concatenates the two DataFrames on the variable 'City_Residence' using the parameter 'on'
- The suffix '_x' denotes the information of 'df1' and the suffix '_y' denotes the information of 'df2'

```
pd.merge(df1, df2, on = 'City_Residence')
```

	Age_x	Gender_x	Salary_x	City_Residence	Age_y	Gender_y	Salary_y
0	45	Male	40000	Mumbai	23	Female	18000
1	45	Male	40000	Mumbai	67	Male	92000
2	12	Male	0	Banglore	34	Male	180000
3	54	Female	150000	Banglore	34	Male	180000
4	26	Male	30000	Delhi	45	Male	88900

Type of merge

The type of merge can be specified using the parameter how = 'type'

how = 'Type'	Description
'outer'	Use all key combinations observed in both tables DataFrames
'inner'	Use only the key combinations observed in both DataFrames
'right'	Use all key combinations found in the right DataFrames
'left'	Use all key combinations found in the left DataFrames

NOTE: If the type is not specified, by default it is 'inner'

Merge using inner join

Concatenating by inner join

The merging takes place on common cities in both the DataFrames

```
pd.merge(df1, df2, on = 'City_Residence', how = 'inner')
```

	Age_x	Gender_x	Salary_x	City_Residence	Age_y	Gender_y	Salary_y
0	45	Male	40000	Mumbai	23	Female	18000
1	45	Male	40000	Mumbai	67	Male	92000
2	12	Male	0	Banglore	34	Male	180000
3	54	Female	150000	Banglore	34	Male	180000
4	26	Male	30000	Delhi	45	Male	88900

Merge using outer join

Concatenating by outer join

All the cities are included if the chosen type is 'outer' in the DataFrame. The city 'Chennai' is not in the second data frame so NaNs are printed corresponding to it.

```
pd.merge(df1, df2, on = 'City_Residence', how = 'outer')
```

	Age_x	Gender_x	Salary_x	City_Residence	Age_y	Gender_y	Salary_y
0	45	Male	40000	Mumbai	23.0	Female	18000.0
1	45	Male	40000	Mumbai	67.0	Male	92000.0
2	12	Male	0	Banglore	34.0	Male	180000.0
3	54	Female	150000	Banglore	34.0	Male	180000.0
4	26	Male	30000	Delhi	45.0	Male	88900.0
5	64	Female	15000	Chennai	NaN	NaN	NaN

Merge using right join

Concatenating by right join

- The resultant merged Data Frame includes all the cities in the right DataFrame.
- The DataFrame df1 has only one observation on Mumbai and df2 has only one observation on Bangalore. So the observations are repeated on their respective sides
- The city 'Chennai' is not in the right Data Frame so it is excluded from the resultant merge output.

```
pd.merge(df1, df2, on = 'City_Residence', how = 'right')
```

	Age_x	Gender_x	Salary_x	City_Residence	Age_y	Gender_y	Salary_y
0	45	Male	40000	Mumbai	23	Female	18000
1	45	Male	40000	Mumbai	67	Male	92000
2	12	Male	0	Banglore	34	Male	180000
3	54	Female	150000	Banglore	34	Male	180000
4	26	Male	30000	Delhi	45	Male	88900

Merge using left join

Concatenating using left join

- The resultant merged Data Frame includes all the cities in the right DataFrame.
- The DataFrame df1 has only one observation on Mumbai and df2 has only one observation on Bangalore. So the observations are repeated on their respective sides
- The city 'Chennai' is not in the right Data Frame so it is excluded from the resultant merge output.

```
pd.merge(df1, df2, on = 'City_Residence', how = 'left')
```

	Age_x	Gender_x	Salary_x	City_Residence	Age_y	Gender_y	Salary_y
0	45	Male	40000	Mumbai	23.0	Female	18000.0
1	45	Male	40000	Mumbai	67.0	Male	92000.0
2	12	Male	0	Banglore	34.0	Male	180000.0
3	54	Female	150000	Banglore	34.0	Male	180000.0
4	26	Male	30000	Delhi	45.0	Male	88900.0
5	64	Female	15000	Chennai	NaN	NaN	NaN

Merge based on Index

Merge based on index

The resultant merged Data Frame has number of rows equal to that of the minimum DataFrame. includes rows from both Data Frames having minimum index

```
pd.merge(df1, df2, right_index=True, left_index=True)
```

	Age_x	Gender_x	Salary_x	City_Residence_x	Age_y	Gender_y	Salary_y	City_Residence_y
0	45	Male	40000	Mumbai	45	Male	88900	Delhi
1	12	Male	0	Banglore	23	Female	18000	Mumbai
2	54	Female	150000	Banglore	67	Male	92000	Mumbai
3	26	Male	30000	Delhi	34	Male	180000	Banglore

Concatenating multiple Data Frames using merge()

The merge function can not merge more than two Data Frames.

```
pd.merge(df1,df2,df3, on = 'City_Residence')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-235-09c1e1da2a96> in <module>
----> 1 pd.merge(df1,df2,df3, on = 'City_Residence')

/opt/anaconda3/lib/python3.7/site-packages/pandas/core/reshape/merge.py in merge(left, right, how, on, left_on, right
_on, left_index, right_index, sort, suffixes, copy, indicator, validate)
     81         validate=validate,
     82     )
---> 83     return op.get_result()
     84
     85

/opt/anaconda3/lib/python3.7/site-packages/pandas/core/reshape/merge.py in get_result(self)
    640             self.left, self.right = self._indicator_pre_merge(self.left, self.right)
    641
--> 642         join_index, left_indexer, right_indexer = self._get_join_info()
    643
    644         ldata, rdata = self.left._data, self.right._data

/opt/anaconda3/lib/python3.7/site-packages/pandas/core/reshape/merge.py in _get_join_info(self)
    857                 )
    858         else:
--> 859             (left_indexer, right_indexer) = self._get_join_indexers()
    860
    861             if self.right_index:

/opt/anaconda3/lib/python3.7/site-packages/pandas/core/reshape/merge.py in _get_join_indexers(self)
    836         """ return the join indexers """
    837         return _get_join_indexers(
--> 838             self.left_join_keys, self.right_join_keys, sort=self.sort, how=self.how
    839         )
    840

/opt/anaconda3/lib/python3.7/site-packages/pandas/core/reshape/merge.py in _get_join_indexers(left_keys, right_keys,
sort, how, **kwargs)
   1314     # preserve left frame order if how == 'left' and sort == False
   1315     kwargs = copy.copy(kwargs)
-> 1316     if how == "left":
   1317         kwargs["sort"] = sort
   1318     join_func = _join_functions[how]

/opt/anaconda3/lib/python3.7/site-packages/pandas/core/generic.py in __nonzero__(self)
   1553             "The truth value of a {0} is ambiguous. "
   1554             "Use a.empty, a.bool(), a.item(), a.any() or a.all().".format(
-> 1555                 self.__class__.__name__
   1556             )
   1557         )
ValueError: The truth value of a DataFrame is ambiguous. Use a.empty, a.bool(), a.item(), a.any() or a.all().
```

Stack and unstack the dataframe

Unstacking pandas.Series

Unstacking can be used to rearrange the Series in a DataFrame.

```
# Create two series using linspace() and concatenate them
x = np.linspace(0,20, 11)
y = np.linspace(1,21, 11)
# x and y are numpy arrays
# first convert them to pandas series
x_series = pd.Series(x)
y_series = pd.Series(y)
print("The type of x_series is :", type(x_series), "\nAnd the type of y_series is : ", type(y_series))
# adding a hierarchical index
# labeling the Index
data = pd.concat([x_series, y_series], keys = ['Even', 'Odd'],names=['Category', 'Index'])
data.unstack()
```

```
The type of x_series is : <class 'pandas.core.series.Series'>
And the type of y_series is :  <class 'pandas.core.series.Series'>
```

Index	0	1	2	3	4	5	6	7	8	9	10
Category											
Even	0.0	2.0	4.0	6.0	8.0	10.0	12.0	14.0	16.0	18.0	20.0
Odd	1.0	3.0	5.0	7.0	9.0	11.0	13.0	15.0	17.0	19.0	21.0

Stacking pandas.Series

- Unstacking can be used to rearrange the Series in a DataFrame
- Stacking a Series can writes the data in the long format

```
# unstack the Series
data = data.unstack()
data
```

Index	0	1	2	3	4	5	6	7	8	9	10
Category											
Even	0.0	2.0	4.0	6.0	8.0	10.0	12.0	14.0	16.0	18.0	20.0
Odd	1.0	3.0	5.0	7.0	9.0	11.0	13.0	15.0	17.0	19.0	21.0

```
# stack the Series
data = data.stack()
data
```

```
Category  Index
Even      0          0.0
          1          2.0
          2          4.0
          3          6.0
          4          8.0
          5         10.0
          6         12.0
          7         14.0
          8         16.0
          9         18.0
          10        20.0
Odd       0          1.0
          1          3.0
          2          5.0
          3          7.0
          4          9.0
          5         11.0
          6         13.0
          7         15.0
          8         17.0
          9         19.0
          10        21.0
dtype: float64
```

Hierarchical Indexing in Data Frames

Creating a DataFrame

```
dataframe = pd.DataFrame(np.arange(12).reshape((4, 3)),
                         index=[['Class_Test', 'Class_Test', 'Sem_Exam', 'Sem_Exam'], [1, 2, 1, 2]],
                            columns=[['Aria', 'Aria', 'John'],
                                     ['Maths', 'English', 'Maths']])

# adding index names
dataframe.index.names = ['key1', 'key2']

# adding column names
dataframe.columns.names = ['Name', 'Subject']

dataframe
```

Name		Aria		John
Subject		Maths	English	Maths
key1	key2			
Class_Test	1	0	1	2
	2	3	4	5
Sem_Exam	1	6	7	8
	2	9	10	11

Hierarchical indexing

```
dataframe = pd.DataFrame(np.arange(12).reshape((4, 3)),
                         index=[['Class_Test', 'Class_Test', 'Sem_Exam', 'Sem_Exam'], [1, 2, 1, 2]],
                            columns=[['Aria', 'Aria', 'John'],
                                     ['Maths', 'English', 'Maths']])

# adding index names
dataframe.index.names = ['key1', 'key2']

# adding column names
dataframe.columns.names = ['Name', 'Subject']

dataframe
```

Name		Aria		John
Subject		Maths	English	Maths
key1	key2			
Class_Test	1	0	1	2
	2	3	4	5
Sem_Exam	1	6	7	8
	2	9	10	11

Unstacking pandas.DataFrame

Unstacking a DataFrame returns a DataFrame having a new level of column labels whose inner most level consists of the the pivoted index labels.

```
dataframe.unstack()
```

Name	Aria				John	
Subject	Maths		English		Maths	
key2	1	2	1	2	1	2
key1						
Class_Test	0	3	1	4	2	5
Sem_Exam	6	9	7	10	8	11

Stacking pandas.DataFrame

Stacking a DataFrame returns a DataFrame having a new level of innermost level (index) consists of the the pivoted column labels.

```
dataframe.stack()
```

			Name	Aria	John
key1	key2	Subject			
Class_Test	1	English	1	NaN
		Maths	0	2.0
	2	English	4	NaN
		Maths	3	5.0
Sem_Exam	1	English	7	NaN
		Maths	6	8.0
	2	English	10	NaN
		Maths	9	11.0

Reshape using melt() function

Reshaping using melt()

- Stacking a DataFrame returns a DataFrame from wide format to long format
- It uses 'id_vars['Col_names']" for reshaping

```
df_melt = df1.melt(id_vars =['Gender', 'City_Residence'])
df_melt
```

	Gender	City_Residence	variable	value
0	Male	Mumbai	Age	45
1	Male	Banglore	Age	12
2	Female	Banglore	Age	54
3	Male	Delhi	Age	26
4	Female	Chennai	Age	64
5	Male	Mumbai	Salary	40000
6	Male	Banglore	Salary	0
7	Female	Banglore	Salary	150000
8	Male	Delhi	Salary	30000
9	Female	Chennai	Salary	15000

Pivot tables

Data for used creating pivot tables

```
yields = {'Months': ['Jan','Feb','Mar','Apr','May','June','July','Aug','Sep','Oct','Nov','Dec'],
        'Yield': [22000,27000,25000,29000,35000,67000,78000,67000,56000,56000,89000,60000],
         'Seasons': ['Winter','Winter','Summer','Summer','Summer','Summer','Summer','Summer',
                     'Rainy','Rainy','Rainy','Winter']
        }
df_yield = pd.DataFrame(yields, columns= ['Months', 'Yield','Seasons'])
df_yield
```

	Months	Yield	Seasons
0	Jan	22000	Winter
1	Feb	27000	Winter
2	Mar	25000	Summer
3	Apr	29000	Summer
4	May	35000	Summer
5	June	67000	Summer
6	July	78000	Summer
7	Aug	67000	Summer
8	Sep	56000	Rainy
9	Oct	56000	Rainy
10	Nov	89000	Rainy
11	Dec	60000	Winter

Pivot table

- Pivot table is a Data Frame like structure

- The pivot tables display data for the specified columns and index

Pivot table

```
pd.pivot_table(df_yield,index=["Months"])
```

	Yield
Months	
Apr	29000
Aug	67000
Dec	60000
Feb	27000
Jan	22000
July	78000
June	67000
Mar	25000
May	35000
Nov	89000
Oct	56000
Sep	56000

```
pd.pivot_table(df_yield,index=["Months", "Seasons"])
```

		Yield
Months	**Seasons**	
Apr	Summer	29000
Aug	Summer	67000
Dec	Winter	60000
Feb	Winter	27000
Jan	Winter	22000
July	Summer	78000
June	Summer	67000
Mar	Summer	25000
May	Summer	35000
Nov	Rainy	89000
Oct	Rainy	56000
Sep	Rainy	56000

Pivot table

```
# specifying the index and values
# yield column sums the data using the 'aggfunc' parameter
pd.pivot_table(df_yield,index=["Seasons"],values=["Yield"],aggfunc=np.sum)
```

Seasons	Yield
Rainy	67000.000000
Summer	50166.666667
Winter	36333.333333

```
# specifying the index and values
# by default 'aggfunc' parameter takes the average
pd.pivot_table(df_yield,index=["Seasons"],values=["Yield"])
```

Seasons	Yield
Rainy	67000.000000
Summer	50166.666667
Winter	36333.333333

Check for duplicates

Check for duplicates

- To check for duplicates, we use the '.duplicated()'.
- To check for duplicates, we use the data from 'example.xlsx' file
- It can be seen last row is the duplicate of second last row, i.e. all the entries in each column are the same.

```
# to check for duplicates
data = pd.concat([df1, df2])
data
```

	Age	Gender	Salary	City_Residence
0	45.0	Male	40000	NaN
1	12.0	Male	0	Banglore
2	NaN	Female	150000	Banglore
3	26.0	Male	30000	Chennai
4	64.0	Female	15000	Chennai
0	45.0	Male	88900	NaN
1	23.0	NaN	18000	Mumbai
2	67.0	Male	92000	Mumbai
3	34.0	Male	180000	Delhi
4	67.0	Male	92000	Mumbai
5	34.0	Male	180000	Delhi

Check for duplicates

- To check for duplicates, we use the '.duplicated()'
- It can be seen rows with index numbers 4 and 5 are duplicates in the second DataFrame

```
# to find the duplicates
data.duplicated()
```

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

Check for duplicates

- To get the count of duplicates, we use the '.duplicated().value_counts()'
- The value 'True' gives the number of duplicate rows in the DataFrame

```
# to get the counts of duplicates
data.duplicated().value_counts()
```

```
False    9
True     2
dtype: int64
```

Drop duplicates

To drop duplicates, we use the '.drop_duplicates()'

```
data.drop_duplicates()
```

	Age	Gender	Salary	City_Residence
0	45.0	Male	40000	NaN
1	12.0	Male	0	Banglore
2	NaN	Female	150000	Banglore
3	26.0	Male	30000	Chennai
4	64.0	Female	15000	Chennai
0	45.0	Male	88900	NaN
1	23.0	NaN	18000	Mumbai
2	67.0	Male	92000	Mumbai
3	34.0	Male	180000	Delhi

Mapping and Replacing

Map

The function map() is used to create a new column by mapping the data frame column values with the dictionary Key

```
# create a data frame
df= pd.DataFrame({'Product':['Milk','Cornflakes','Prunes','Bread','Jam'],
                  'Price':[15,134,322,16,165]})
df
```

	Product	Price
0	Milk	15
1	Cornflakes	134
2	Prunes	322
3	Bread	16
4	Jam	165

```
# create a dictionary
brand = {
'Milk':'MilkMan',
'Prunes':'DryFruits',
'Jam':'DailyEats',
'Butter':'Amul',
'PeanutSpread':'Spreadz',
'Cornflakes':'Kellogs',
'Bread':'Bakes&More',
'Ketchup': 'Kissan',
'Sausage':'ColdChicken'
}
```

Create a new column

```
df['Brand'] = df['Product'].map(brand)
df
```

	Product	Price	Brand
0	Milk	15	MilkMan
1	Cornflakes	134	Kellogs
2	Prunes	322	DryFruits
3	Bread	16	Bakes&More
4	Jam	165	DailyEats

Reduce

The reduce() is used to replace any value in the DataFrame

```
dict_price = {15: '30', 322: '324'}

# The values 15 is replaced by 30 and 322 is replaced by 324
df['Price'].replace(dict_price, inplace=True)

print(df)
```

```
      Product Price       Brand
0        Milk    30     MilkMan
1  Cornflakes   134     Kellogs
2      Prunes   324   DryFruits
3       Bread    16  Bakes&More
4         Jam   165   DailyEats
```

Dataframe Manipulation using groupby()

Groupby()

```
# create a data frame
customer =pd.DataFrame({
 'Customer_ID':['191A','132D','113D','414P','532L','980J'],
 'Mode_of_Payment':['Cash','Online','Cash','Online','Card','Online'],
 'Delivery':['Self','Self','Delivered','Delivered','Self','Pick-up'],
 'Amount': [450, 834, 734, 564,454, 90],
 'Discount': [10, 67, 83, 72, 32, 2]
})
print(customer)
```

```
  Customer_ID Mode_of_Payment    Delivery  Amount  Discount
0        191A            Cash        Self     450        10
1        132D          Online        Self     834        67
2        113D            Cash   Delivered     734        83
3        414P          Online   Delivered     564        72
4        532L            Card        Self     454        32
5        980J          Online     Pick-up      90         2
```

```
# Number of unique values per group
customer.groupby("Mode_of_Payment")["Delivery"].nunique().to_frame()
```

Mode_of_Payment	Delivery
Card	1
Cash	2
Online	3

Groupby()

```
customer.groupby('Mode_of_Payment')['Amount'].sum().to_frame().reset_index()
```

	Mode_of_Payment	Amount
0	Card	454
1	Cash	1184
2	Online	1488

Summary Statistics

Summary statistics

- Summary Statistics of the DataFrame is obtained using the function .describe()
- The function DataFrame.describe() gives the summary output of numeric values
- Summary Statistics of the DataFrame is obtained using the function .describe(include =['object'])
- The function DataFrame.describe() gives the summary output of categorical values

Summary statistics

The output gives the :
count: total rows
mean: average of the observations
std: standard deviation of the observations
min: minimum value of the observations
25%: first quartile of the observations
50%: second quartile (median) of the observations
75%: third quartile of the observations
max: maximum value of the observations

```
data = pd.concat([df2,df1], axis = 0, ignore_index= True)
data.describe()
```

	Age	Salary
count	16.000000	16.00000
mean	34.625000	52287.50000
std	17.549454	52229.19522
min	12.000000	0.00000
25%	22.500000	17250.00000
50%	24.500000	33375.00000
75%	47.250000	80725.00000
max	67.000000	180000.00000

Summary statistics

The output gives the :

count: total rows
unique: the number of unique values
top: the value which has the highest frequency
freq: the frequency of the 'top' value

```
data = pd.concat([df2,df1], axis = 0, ignore_index= True)
data.describe(include=['object'])
```

	Gender	City_Residence
count	16	16
unique	2	4
top	Male	Delhi
freq	9	7

Skewness and Kurtosis

Skewness

- Skewness is a lack of symmetry or departure from symmetry
- If the distribution of the data is elongated on either sides then the data is said to be skewed
- If the distribution of the data is elongated on the left side then the data is said to be left skewed
- If the distribution of the data is elongated on the right side then the data is said to be right skewed

Skewness

```
# load the data from 'Sheet1' of the xlsx file and print it
df1 = pd.read_excel('example_merge.xlsx',  sheet_name=0 )
# load the data from 'Sheet2' of the xlsx file and print it
df2 = pd.read_excel('example_merge.xlsx',  sheet_name=1 )
data = df1.append(df2)
```

```
data.skew()
```

```
Age      -0.067567
Salary    0.805063
dtype: float64
```

Kurtosis

- Kurtosis is a statistical measure that defines how heavily the tails of the distribution differ from the normal distribution
- Kurtosis identifies whether the tails of a given distribution contain extreme values
- Thicker the tails are more the extreme values in the data

Kurtosis

```
# load the data from 'Sheet1' of the xlsx file and print it
df1 = pd.read_excel('example_merge.xlsx',  sheet_name=0 )
# load the data from 'Sheet2' of the xlsx file and print it
df2 = pd.read_excel('example_merge.xlsx',  sheet_name=1 )
data = df1.append(df2)
```

```
data.kurt()
```

```
Age      -1.114185
Salary   -0.658787
dtype: float64
```

Thank You