Learning_Pandas_Part_3_Joins_Append

June 20, 2021

0.0.1 Prepared by Abhishek Kumar

0.0.2 https://www.linkedin.com/in/abhishekkumar-0311/

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
# To get multiple outputs in the same cell

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

%matplotlib inline
```

[3]:
```
# Setup : DataFrame creation

salary = [['1','Abhishek Kumar','AIML', 'Machine Learning Engineer','M', 'Y',␣
 ↪'04051990', 1121000],
          ['2','Arjun Kumar','DM', 'Tech Lead','M', 'Y', '09031992', 109000],
          ['3','Vivek Raj','DM', 'Devops Engineer','M', 'N', np.NaN , 827000],
          ['4','Mika Singh','DM', 'Data Analyst','F', 'Y', '15101991',  np.NaN],
          ['5','Anusha Yenduri','AIML', 'Data Scientist','M', 'Y', '01011989', ␣
 ↪921000],
          ['6','Ritesh Srivastava','AIML', 'Data Engineer','M', 'Y', np.NaN,␣
 ↪785000]]

columns_name=['Emp_Id','Emp_Name','Department','Role','Gender', 'WFH Status',␣
 ↪'DOB', 'Salary']

emp_df = pd.DataFrame(salary,columns=columns_name)
emp_df
```

[3]:

	Emp_Id	Emp_Name	Department	Role	Gender	\
0	1	Abhishek Kumar	AIML	Machine Learning Engineer	M	
1	2	Arjun Kumar	DM	Tech Lead	M	
2	3	Vivek Raj	DM	Devops Engineer	M	
3	4	Mika Singh	DM	Data Analyst	F	

```
4      5     Anusha Yenduri       AIML             Data Scientist      M
5      6  Ritesh Srivastava       AIML              Data Engineer      M

  WFH Status       DOB     Salary
0          Y  04051990  1121000.0
1          Y  09031992   109000.0
2          N       NaN   827000.0
3          Y  15101991        NaN
4          Y  01011989   921000.0
5          Y       NaN   785000.0
```

1 1. Concatenating dataframes vertically / Appending dataframes

I ll take up 2 ways to do this.

i. df.append()
ii. pd.concat() - With concatenation, your datasets are just stitched together along an axis - either the row axis or column axis.

1.0.1 i. df.append()

[4]:
```
detail_1 = ({'Id'    : [1,2,3,4],
             'Name' : ['A','B','C','D'],
             'Age'  : [21,22,20,24] })
detail_2 = ({'Id'    : [2,8,5],
             'Name' : ['Again','H','E'],
             'Age'  : [25,18,28],
             'City' : ['Pune','Panaji','Patna']})
detail_3 = ({'Id'    : [7,6],
             'Name' : ['G','F'],
             'Age'  : [34,30] })
df1 = pd.DataFrame(detail_1)
df2 = pd.DataFrame(detail_2)
df3 = pd.DataFrame(detail_3)
df1
df2
df3

# Multiple dataframe objects can be passed as a list
df_appended_1 = df1.append([df2,df3], sort=True) # sort=True/False - sorts the␣
 ↪column names in Alphabetical order.
df_appended_1
df_appended_2 = df1.append([df2,df3], sort=False, ignore_index=True) #␣
 ↪ignore_index= True creates a new index for the dataframe
df_appended_2
```

[4]:

	Id	Name	Age
0	1	A	21
1	2	B	22
2	3	C	20
3	4	D	24

[4]:

	Id	Name	Age	City
0	2	Again	25	Pune
1	8	H	18	Panaji
2	5	E	28	Patna

[4]:

	Id	Name	Age
0	7	G	34
1	6	F	30

[4]:

	Age	City	Id	Name
0	21	NaN	1	A
1	22	NaN	2	B
2	20	NaN	3	C
3	24	NaN	4	D
0	25	Pune	2	Again
1	18	Panaji	8	H
2	28	Patna	5	E
0	34	NaN	7	G
1	30	NaN	6	F

[4]:

	Id	Name	Age	City
0	1	A	21	NaN
1	2	B	22	NaN
2	3	C	20	NaN
3	4	D	24	NaN
4	2	Again	25	Pune
5	8	H	18	Panaji
6	5	E	28	Patna
7	7	G	34	NaN
8	6	F	30	NaN

[5]:
```
df_appended_1 is df1
```

[5]: False

1.0.2 Note :

1. join= and keys= parameters are not available in df.append.
2. So, By default, ALL the columns are selected and index can be either retained or newly creat
3. It makes a full copy of the data, and that constantly reusing this function can create a sig

1.0.3 ii. pd.concat(objs, axis=0, join='outer', sort='False', ignore_index=False, keys=None, levels=None, names=None, verify_integrity=False, copy=True)

pd.concat() is capable of concatenating dataframes either way longitudinal as well latitudinal.

[6]:
```
# Implementing the pd.concat() to behave similar to df.append()
# Default action of concat is vertical join/ append, as axis=0, by default.

df_appended_3 = pd.concat([df1,df2,df3], axis = 0, join = 'outer', sort= False)
df_appended_3
```

[6]:

	Id	Name	Age	City
0	1	A	21	NaN
1	2	B	22	NaN
2	3	C	20	NaN
3	4	D	24	NaN
0	2	Again	25	Pune
1	8	H	18	Panaji
2	5	E	28	Patna
0	7	G	34	NaN
1	6	F	30	NaN

[7]:
```
df_appended_4 = pd.concat([df1,df2,df3], axis = 0, join = 'outer', sort = False␣
↪, keys = ['a','b','c'], ignore_index = True, copy = True)
df_appended_4

# Note : Keys=a,b,c is passed, but still the keys are not assigned because the␣
↪INDEXES are IGNORED. To create the keys, we need to retain the indexes.
```

[7]:

	Id	Name	Age	City
0	1	A	21	NaN
1	2	B	22	NaN
2	3	C	20	NaN
3	4	D	24	NaN
4	2	Again	25	Pune
5	8	H	18	Panaji
6	5	E	28	Patna
7	7	G	34	NaN
8	6	F	30	NaN

[8]:
```
# The distinct keys are created, with keys= parameter and ignore_index=False
# join=inner considers only the common columns of all dataframes

df_appended_5 = pd.concat([df1,df2,df3], axis = 0, join = 'inner', sort = True␣
↪, keys = ['a','b','c'], ignore_index = False, copy = True)
df_appended_5
```

[8]:
```
      Age  Id   Name
a 0    21   1      A
  1    22   2      B
  2    20   3      C
  3    24   4      D
b 0    25   2  Again
  1    18   8      H
  2    28   5      E
c 0    34   7      G
  1    30   6      F
```

[9]:
```
# The keys created above, can be used as filters.

df_appended_5.loc['b']
```

[9]:
```
   Age  Id   Name
0   25   2  Again
1   18   8      H
2   28   5      E
```

2 1.1 Appending Rows (as Series)

[10]:
```
# Attention : Does not Work

s = pd.Series(['11', 'Eleven', 21])
s
df1
# appended_row = df1.append(s)
# appended_row = df1.append(s, ignore_index = True) # ignore_index = True fixes␣
 ↪the error, but the output Dataframe is NOT DESIRED.
# appended_row
```

[10]:
```
0        11
1    Eleven
2        21
dtype: object
```

[10]:
```
   Id Name  Age
0   1    A   21
1   2    B   22
2   3    C   20
3   4    D   24
```

[11]:
```
s = pd.Series(['11', 'Eleven', 21], index = ['Id','Name', 'Age'])
s
df1
```

```
appended_row = df1.append(s, ignore_index = True)
appended_row
```

[11]:
```
Id          11
Name    Eleven
Age         21
dtype: object
```

[11]:

	Id	Name	Age
0	1	A	21
1	2	B	22
2	3	C	20
3	4	D	24

[11]:

	Id	Name	Age
0	1	A	21
1	2	B	22
2	3	C	20
3	4	D	24
4	11	Eleven	21

[12]:
```
# Set up

detail_1 = ({'Id'   : [1,2,3,4],
             'Name' : ['A','B','C','D'],
             'Age'  : [21,22,20,24] })
detail_2 = ({'Id'   : [1,2,5],
             'Sal'  : [100,200,500],
             'City' : ['Pune','Panaji',np.NaN]})

dfv_1 = pd.DataFrame(detail_1)
dfv_2 = pd.DataFrame(detail_2)
dfv_1
dfv_2
```

[12]:

	Id	Name	Age
0	1	A	21
1	2	B	22
2	3	C	20
3	4	D	24

[12]:

	Id	Sal	City
0	1	100	Pune
1	2	200	Panaji
2	5	500	NaN

[13]:
```python
# Does not make sense to me, as of now.

s2 = pd.Series(['_0', '_1', '_2', '_3'])

dfh_concat_s = pd.concat([dfv_1,dfv_2,s2,s2,s2], axis=0)
dfh_concat_s

# Uncomment and see the result , with rows have index name
# s3 = pd.Series(['_0', '_1', '_2', '_3'], index = ['Id','Name', 'Age','Sal'])

# dfh_concat_s = pd.concat([dfv_1,dfv_2,s3,s3,s3], axis=0, ignore_index = True)
# dfh_concat_s
```

[13]:

	0	Age	City	Id	Name	Sal
0	NaN	21.0	NaN	1.0	A	NaN
1	NaN	22.0	NaN	2.0	B	NaN
2	NaN	20.0	NaN	3.0	C	NaN
3	NaN	24.0	NaN	4.0	D	NaN
0	NaN	NaN	Pune	1.0	NaN	100.0
1	NaN	NaN	Panaji	2.0	NaN	200.0
2	NaN	NaN	NaN	5.0	NaN	500.0
0	_0	NaN	NaN	NaN	NaN	NaN
1	_1	NaN	NaN	NaN	NaN	NaN
2	_2	NaN	NaN	NaN	NaN	NaN
3	_3	NaN	NaN	NaN	NaN	NaN
0	_0	NaN	NaN	NaN	NaN	NaN
1	_1	NaN	NaN	NaN	NaN	NaN
2	_2	NaN	NaN	NaN	NaN	NaN
3	_3	NaN	NaN	NaN	NaN	NaN
0	_0	NaN	NaN	NaN	NaN	NaN
1	_1	NaN	NaN	NaN	NaN	NaN
2	_2	NaN	NaN	NaN	NaN	NaN
3	_3	NaN	NaN	NaN	NaN	NaN

3 2. Concatenating Dataframes horizontally

Using pd.concat(axis=1) - With concatenation, your datasets are just stitched together along an either the row axis or column axis.

3.0.1 pd.concat(objs, axis=1, join='outer', sort='False', ignore_index=False, keys=None, levels=None, names=None, verify_integrity=False, copy=True)

Parameter :

1. objs: This parameter takes any sequence (typically a list) of Series or DataFrame objects to

2. axis: Like in the other techniques, this represents the axis you will concatenate along. The

3. join: This is similar to the how parameter in the other techniques, but it only accepts the

4. ignore_index: This parameter takes a boolean (True or False) and defaults to False. If True

5. keys: This parameter allows you to construct a hierarchical index. One common use case is t

6. copy: This parameter specifies whether you want to copy the source data. The default value

[14]:
```
# Set up

detail_1 = ({'Id'   : [1,2,3,4],
             'Name' : ['A','B','C','D'],
             'Age'  : [21,22,20,24] })
detail_2 = ({'Id'   : [1,2,5],
             'Sal'  : [100,200,500],
             'City' : ['Pune','Panaji',np.NaN]})

dfh_1 = pd.DataFrame(detail_1, index = [5,6,7,1])
# dfh_1 = pd.DataFrame(detail_1)
dfh_2 = pd.DataFrame(detail_2)
dfh_1
dfh_2
```

[14]:

	Id	Name	Age
5	1	A	21
6	2	B	22
7	3	C	20
1	4	D	24

[14]:

	Id	Sal	City
0	1	100	Pune
1	2	200	Panaji
2	5	500	NaN

[15]:
```
# The dataframes are concatenated horizontally based on INDEX

dfh_concat = pd.concat([dfh_1,dfh_2], axis=1, join = 'outer',␣
 ↪ignore_index=False)
dfh_concat
```

[15]:

	Id	Name	Age	Id	Sal	City
0	NaN	NaN	NaN	1.0	100.0	Pune
1	4.0	D	24.0	2.0	200.0	Panaji
2	NaN	NaN	NaN	5.0	500.0	NaN
5	1.0	A	21.0	NaN	NaN	NaN
6	2.0	B	22.0	NaN	NaN	NaN
7	3.0	C	20.0	NaN	NaN	NaN

[16]:
```
# With keys=a,b, the columns are labelled as 'a' and 'b' with ignore_index=False
# With join= parameter - is used to Set logic on the other axes
# With join= inner, the common records are selected

dfh_concat = pd.concat([dfh_1,dfh_2], axis=1, join = 'inner',␣
 ↪ignore_index=False ,keys = ['a','b'] , sort = False )
dfh_concat
```

[16]:

	a			b		
	Id	Name	Age	Id	Sal	City
1	4	D	24	2	200	Panaji

[17]:
```
# ignore_index= True, column labels are Re-named.
# sort= True, sorts the records in ascending order

dfh_concat = pd.concat([dfh_1,dfh_2], axis=1, join = 'outer', ignore_index=␣
 ↪True,sort = True )
dfh_concat
```

[17]:

	0	1	2	3	4	5
0	NaN	NaN	NaN	1.0	100.0	Pune
1	4.0	D	24.0	2.0	200.0	Panaji
2	NaN	NaN	NaN	5.0	500.0	NaN
5	1.0	A	21.0	NaN	NaN	NaN
6	2.0	B	22.0	NaN	NaN	NaN
7	3.0	C	20.0	NaN	NaN	NaN

Note : With concatenation, your datasets are just stitched together along an axis — either the row axis or column axis.

4 2.1 Appending Columns (as Series)

[18]:
```
# Set up

detail_1 = ({'Id'   : [1,2,3,4],
             'Name' : ['A','B','C','D'],
             'Age'  : [21,22,20,24] })
detail_2 = ({'Id'   : [1,2,5],
             'Sal'  : [100,200,500],
             'City' : ['Pune','Panaji',np.NaN]})

dfh_1 = pd.DataFrame(detail_1)
dfh_2 = pd.DataFrame(detail_2)
dfh_1
dfh_2
```

[18]:
	Id	Name	Age
0	1	A	21
1	2	B	22
2	3	C	20
3	4	D	24

[18]:
	Id	Sal	City
0	1	100	Pune
1	2	200	Panaji
2	5	500	NaN

[19]:
```
s2 = pd.Series(['_0', '_1', '_2', '_3'])

dfh_concat_s = pd.concat([dfh_1,dfh_2,s2,s2,s2], axis=1)
dfh_concat_s
```

[19]:
	Id	Name	Age	Id	Sal	City	0	1	2
0	1	A	21	1.0	100.0	Pune	_0	_0	_0
1	2	B	22	2.0	200.0	Panaji	_1	_1	_1
2	3	C	20	5.0	500.0	NaN	_2	_2	_2
3	4	D	24	NaN	NaN	NaN	_3	_3	_3

5 3. Database-style DataFrame or named Series joining/merging

i. pd.merge()
ii. pd.join()

5.0.1 i. pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=True, suffixes=('_x', '_y'), copy=True, indicator=False,validate=None)

1. how: This defines what kind of merge to make. It defaults to 'inner', but other possible opt

2. on: Use this to tell merge() which columns or indices (also called key columns or key indice

3. left_on and right_on: Use either of these to specify a column or index that is present only

4. left_index and right_index: Set these to True to use the index of the left or right objects

5. suffixes: This is a tuple of strings to append to identical column names that are not merge

6. indicator: If True, a Categorical-type column called _merge will be added to the output obje

[20]:
```
# Set up

detail_1 = ({'Id'   : [1,2,3,4],
             'Name' : ['A','B','C','D'],
```

```
            'Age'  : [21,22,20,24] })
detail_2 = ({'Id'   : [1,2,5],
             'Sal'  : [100,200,500],
             'City' : ['Pune','Panaji',np.NaN]})

df_mrg_1 = pd.DataFrame(detail_1)
df_mrg_2 = pd.DataFrame(detail_2)
df_mrg_1
df_mrg_2
```

[20]:

	Id	Name	Age
0	1	A	21
1	2	B	22
2	3	C	20
3	4	D	24

[20]:

	Id	Sal	City
0	1	100	Pune
1	2	200	Panaji
2	5	500	NaN

[21]:
```
# how= inner, which tells the type of join
# if left_index and right_index are False, then columns from the two DataFrames␣
 ↪that share names will be used as join keys
# So here, on = Id

data_merged = pd.merge(df_mrg_1,df_mrg_2)
data_merged
```

[21]:

	Id	Name	Age	Sal	City
0	1	A	21	100	Pune
1	2	B	22	200	Panaji

[22]:
```
# Set up

detail_1 = ({'Id'   : [1,2,3,4],
             'Name' : ['A','B','C','D'],
             'Age'  : [21,np.NaN,25,24] })
detail_2 = ({'Id'   : [1,2,5],
             'Age'  : [21,38,20],
             'Sal'  : [100,200,500],
             'City' : ['Pune','Panaji',np.NaN]})

df_mrg_1 = pd.DataFrame(detail_1)
df_mrg_2 = pd.DataFrame(detail_2)
df_mrg_1
df_mrg_2
```

[22]:
```
   Id Name   Age
0   1    A  21.0
1   2    B   NaN
2   3    C  25.0
3   4    D  24.0
```

[22]:
```
   Id  Age  Sal    City
0   1   21  100    Pune
1   2   38  200  Panaji
2   5   20  500     NaN
```

[23]:
```
# There are Multiple things to observe here.
# 1. type of join, how= parameter states, here it is left join
# 2. on= parameter, defines the columns on which dataframes need to be joined,␣
 ↪here ['Id', 'Age']
# 3a. suffixes= parameter, allows to differentiate the columns of different␣
 ↪dataframes.
# 3b. Note: this won't add suffixes for columns which are passed in the 'on'␣
 ↪parameter.
# 4. indicator = columns_name, states the presence of 'on' columns in␣
 ↪dataframes being merged.

data_merged = pd.merge(df_mrg_1,df_mrg_2, how = 'left', on = ['Id'],␣
 ↪suffixes=['_l', '_r'], indicator= 'Presence')
data_merged
```

[23]:
```
   Id Name  Age_l  Age_r    Sal    City   Presence
0   1    A   21.0   21.0  100.0    Pune       both
1   2    B    NaN   38.0  200.0  Panaji       both
2   3    C   25.0    NaN    NaN     NaN  left_only
3   4    D   24.0    NaN    NaN     NaN  left_only
```

[24]:
```
# Set up

detail_1 = ({'Id'   : [1,2,3,4],
             'Name' : ['A','B','C','D'],
             'Age'  : [21,np.NaN,25,24] })
detail_2 = ({'Id'   : [1,2,5],
             'Age'  : [21,38,20],
             'Sal'  : [100,200,500],
             'City' : ['Pune','Panaji',np.NaN]})

df_mrg_1 = pd.DataFrame(detail_1)
df_mrg_2 = pd.DataFrame(detail_2)
df_mrg_1
df_mrg_2
```

[24]:
```
   Id Name   Age
0   1    A  21.0
1   2    B   NaN
2   3    C  25.0
3   4    D  24.0
```

[24]:
```
   Id  Age  Sal    City
0   1   21  100    Pune
1   2   38  200  Panaji
2   5   20  500     NaN
```

[25]:
```
# Now, the dataframes are merged based on INDEXes.
# Left dataframe Index is merged with Right dataframe Index

data_merged = pd.merge(df_mrg_1, df_mrg_2, left_index=True, right_index=True,␣
↪how='outer')
data_merged
```

[25]:
```
   Id_x Name  Age_x  Id_y  Age_y    Sal    City
0     1    A   21.0   1.0   21.0  100.0    Pune
1     2    B    NaN   2.0   38.0  200.0  Panaji
2     3    C   25.0   5.0   20.0  500.0     NaN
3     4    D   24.0   NaN    NaN    NaN     NaN
```

[26]:
```
# Now, the dataframes are merged by Joining key columns on an index.
# Left dataframe Index is merged with 'Id' from right dataframe.

data_merged = pd.merge(df_mrg_1, df_mrg_2, left_index=True, right_on='Id',␣
↪how='inner')
data_merged
```

[26]:
```
   Id  Id_x Name  Age_x  Id_y  Age_y  Sal    City
0   1     2    B    NaN     1     21  100    Pune
1   2     3    C   25.0     2     38  200  Panaji
```

Note: There are more parameters of merge(). Please find them at Pandas Documentation.

5.0.2 ii. DataFrame.join(self, other, on=None, how='left', lsuffix='', rsuffix='', sort=False)

Parameters:

1. other: This is the only required parameter. It defines the other DataFrame to join. You can

2. on: This parameter specifies an optional column or index name for the left DataFrame to joi

3. how: This has the same options as how from merge(). The difference is that it is index-base

4. lsuffix and rsuffix: These are similar to suffixes in merge(). They specify a suffix to add

5. sort: Enable this to sort the resulting DataFrame by the join key.

5.0.3 Note :

While merge() is a module function, .join() is an object function that lives on the DataFrame.

[27]:
```
# Set up

detail_1 = ({'Id'   : [1,2,3,4],
             'Name' : ['A','B','C','D'],
             'Age'  : [21,np.NaN,25,24] })
detail_2 = ({'Id'   : [1,2,5],
             'Age'  : [21,38,20],
             'Sal'  : [100,200,500],
             'City' : ['Pune','Panaji',np.NaN]})

df_mrg_1 = pd.DataFrame(detail_1)
df_mrg_2 = pd.DataFrame(detail_2)

df_mrg_1
df_mrg_2
```

[27]:

	Id	Name	Age
0	1	A	21.0
1	2	B	NaN
2	3	C	25.0
3	4	D	24.0

[27]:

	Id	Age	Sal	City
0	1	21	100	Pune
1	2	38	200	Panaji
2	5	20	500	NaN

[28]:
```
# Since the columns overlap, suffixes are required.Uncomment to see the error.
# df_mrg_1.join(df_mrg_2, how = 'inner', on = 'Id')

df_mrg_1.join(df_mrg_2, how = 'inner', on = 'Id', lsuffix = '_l', rsuffix =␣
 ↪'_r' )
```

[28]:

	Id	Id_l	Name	Age_l	Id_r	Age_r	Sal	City
0	1	1	A	21.0	2	38	200	Panaji
1	2	2	B	NaN	5	20	500	NaN

5.0.4 References :

1. **Refer PDF**

2. **Pandas Documentation**

3. **Real Python**

[ ]: