Introduction to Big Data

- Developed by Dr. Keungoui KIM
- https://awekim.github.io/portfolio/

Lecture 7. Relational Database

Join

- merging or integrating two Data Frame

Pandas functions; merge, concat, append

- (row reference) join by column name: merge
- (row reference) join by index name: merge, concat
- (column reference) join by row: concat, append

Methods

- Inner: Intersect
- Left: Join to left DataFrame
- Right: Join to right DataFrame
- Outer: Join all

pd.merge

pd.merge('left', 'right', 'how', 'on', 'left_on', 'right_on', 'left_index', 'right_index')

- left : DataFrame positioned on the left
- right : DataFrame positioned on the right

- how : inner, left, right, outer
- on : Name of reference column for Join
- left_on : Name of reference column of left - DataFrame
- right_on : Name of reference column of right DataFrame
- left_index : True if join withi left DataFrame index
- right_index : True if join withi right DataFrame index

pd.concat

pd.concat([left,right], axis, join,...)

- [left,right] : a list of left DataFrame and right DataFrame
- axis = 0 if join by raw reference, 1 if join by column reference

Pandas Exercise

Create DataFrame

```
import pandas as pd

df1 = pd.DataFrame(
    {   "A": ["A0", "A1", "A2", "A3"],
        "B": ["B0", "B1", "B2", "B3"],
        "C": ["C0", "C1", "C2", "C3"],
        "D": ["D0", "D1", "D2", "D3"]},
     index=[0, 1, 2, 3])
df11 = pd.DataFrame(
    {   "A": ["A0", "A1", "A21", "A31"],
        "E": ["E0", "E11", "E21", "E31"],
        "F": ["F0", "F11", "F21", "F31"],
        "G": ["G0", "G11", "G21", "G31"]},
     index=[0, 1, 2, 3])
df2 = pd.DataFrame(
     {  "A": ["A4", "A5", "A6", "A7"],
        "B": ["B4", "B5", "B6", "B7"],
        "C": ["C4", "C5", "C6", "C7"],
        "D": ["D4", "D5", "D6", "D7"]},
     index=[4, 5, 6, 7])
df3 = pd.DataFrame(
    {   "A": ["A8", "A9", "A10", "A11"],
        "B": ["B8", "B9", "B10", "B11"],
        "C": ["C8", "C9", "C10", "C11"],
        "D": ["D8", "D9", "D10", "D11"]},
     index=[8, 9, 10, 11])
```

```
df1
```

```
df11
```

```
df2
```

```
df3
```

.merge

```
pd.merge(df1, df11, how='left', on='A')
```

```
pd.merge(df1,df11,how='left')
```

```
pd.merge(df1,df11,how='left',on='B')
```

```
pd.merge(df1,df11,how='right',on='A')
```

```
pd.merge(df1,df11,how='inner',on='A')
```

```
pd.merge(df1,df11,how='outer',on='A')
```

.concat

- .concat: The concat() function (in the main pandas namespace) does all of the heavy lifting of performing concatenation operations along an axis while performing optional set logic (union or intersection) of the indexes (if any) on the other axes.

```
frames = [df1, df2, df3]
frames
```

```
type(frames)
```

```
len(frames)
```

```
pd.concat([df1, df2, df3])
```

```
pd.concat([df1, df2, df3], axis=0)
```

```
pd.concat([df1, df2, df3], axis=1)
```

```
pd.concat([df1, df2, df3], join='inner', axis=1)
```

```
pd.concat([df1, df2], keys=['df1','df2'])
```

```
pd.concat([df1, df2],
          keys=['df1','df2'],
          names=['dfname','index'])
```

```
(pd.concat([df1, df2],
          keys=['df1','df2'],
          names=['dfname','index']).
          reset_index())
```

Relational Database Practice

```
bb_pp_df = pd.read_csv('/content/drive/MyDrive/[Lecture]/IntBigData/BigData_Python/07_RelationalDataba
bb_ap_df = pd.read_csv('/content/drive/MyDrive/[Lecture]/IntBigData/BigData_Python/07_RelationalDataba
bb_sal_df = pd.read_csv('/content/drive/MyDrive/[Lecture]/IntBigData/BigData_Python/07_RelationalDatab
```

```
bb_pp_df.head(2)
```

```
bb_ap_df.head(2)
```

```
bb_sal_df.head(2)
```

```
bb_ap_df['yearID'].min()
```

```
bb_ap_df['yearID'].max()
```

```
minyear, maxyear = [bb_ap_df['yearID'].min(), bb_ap_df['yearID'].max()]
print(minyear)
print(maxyear)
```

What is the average salary of award winning player in 2016?

```
bb_ap_df['yearID']==2016
```

```
bb_ap_df[bb_ap_df['yearID']==2016].head()
```

```
bb_ap_sal_df = (
    pd.merge(bb_ap_df[bb_ap_df['yearID']==2016],
             bb_sal_df,
             how='inner'#,on=['playerID','yearID','lgID']
             ))
bb_ap_sal_df
```

```
bb_ap_sal_df['salary'].mean()
```

```
bb_ap_sal_df.value_counts('playerID').head()
```

```
# Option 1
bb_ap_sal_df_idgr = (
    bb_ap_sal_df.groupby('playerID')
)
```

```
(bb_ap_sal_df_idgr['salary'].
 mean().head())
```

```
bb_ap_sal_df_idgr['salary'].mean().mean()
```

```
# Option 2
(bb_ap_sal_df.
 drop_duplicates(subset='playerID').
 salary.mean())
```

Compare the average salary of award winning players between AL and NL in 2016

```
bb_ap_sal_df.groupby('lgID').playerID.nunique()
```

```
(bb_ap_sal_df.groupby('lgID').
 salary.mean().
 reset_index())
```

```
# Option 1
bb_ap_sal_df_idlggr = (
    bb_ap_sal_df.groupby(['playerID','lgID'])
)
```

```
bb_ap_sal_df_idlggr_m = (
    bb_ap_sal_df_idlggr['salary'].mean().reset_index()
)
bb_ap_sal_df_idlggr_m.head()
```

```
(
    bb_ap_sal_df_idlggr_m.groupby('lgID').
 salary.mean().reset_index()
)
```

In 2016, is top salary player the award winning player ?

```
(pd.merge(bb_sal_df[bb_sal_df['yearID']==2016],
          bb_ap_df[bb_ap_df['yearID']==2016], how='left').
 sort_values('salary', ascending=False))
```

What is the name of player with the top salary in 2016?

```
bb_pp_sal_df = pd.merge(bb_pp_df, bb_sal_df[bb_sal_df['yearID']==2016],
                        how='left', on='playerID')
bb_pp_sal_df.sort_values('salary', ascending=False)[['nameLast','nameFirst']]
```

```
pd.concat([df1, df2, df3], axis=1, keys=['x','y','z'])
```

```
result = pd.concat([df1, df2, df3], axis=0, keys=['x','y','z'])
result
```

```
# check first 5 rows
result.head()
```

```
# check first 10 rows
result.head(10)
```

```
# check last 5 rows
result.tail()
```

```
# Filter specific column
result['A']
```

```
# Filter specific key
result.loc["x"]
```

```
# inner join
# result is different because concat matches by index
# It is not often recommended because it returns the duplicated data.
inner_result = pd.concat([df1, df4], axis=1, join = 'inner') # column reference
inner_result
```

```
# outer join
# result is different because concat matches by index
# It is not often recommended because it returns the duplicated data.
outer_result = pd.concat([df1, df4], axis=1, join = 'outer') # column reference
outer_result
```

Vertical join

```
df4 = pd.DataFrame(
    {"B": ["B2", "B3", "B6", "B7"],
     "D": ["D2", "D3", "D6", "D7"],
     "F": ["F2", "F3", "F6", "F7"],
     },
     index=[2, 3, 6, 7]
)
df4
```

```
# outer join
outer_result = pd.concat([df1, df4], axis=0, join='outer') # raw reference
outer_result
```

```
df1.append(df4)
```

```
# outer join
inner_result = pd.concat([df1, df4], axis=0, join='inner') # raw reference
inner_result
```

Horizontal join

```
# left join
left_result = pd.merge(df1, df4, 'left')
left_result
```

```
# right join
right_result = pd.merge(df1, df4, 'right')
```

```
# inner join
inner_result = pd.merge(df1, df4, 'inner')
inner_result
```

```
# outer join
outer_result = pd.merge(df1, df4, 'outer')
outer_result
```