02032021 - Advanced Pandas Functionalities

February 27, 2021

1 Pandas - Data Frame Operations

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

[2]:
```
# create a simple dataframe
data = pd.DataFrame({
    'age' :     [ 10, 22, 13, 21, 12, 11, 17],
    'section' : [ 'A', 'B', 'C', 'B', 'B', 'A', 'A'],
    'city' :    [ 'Gurgaon', 'Delhi', 'Mumbai', 'Delhi', 'Mumbai', 'Delhi',␣
 ↪'Mumbai'],
    'gender' :  [ 'M', 'F', 'F', 'M', 'M', 'M', 'F'],
    'favourite_color' : [ 'red', 'blue', 'yellow', 'blue', 'black', 'green',␣
 ↪'red']
})

# view the data
data
```

[2]:
```
   age section     city gender favourite_color
0   10       A  Gurgaon      M             red
1   22       B    Delhi      F            blue
2   13       C   Mumbai      F          yellow
3   21       B    Delhi      M            blue
4   12       B   Mumbai      M           black
5   11       A    Delhi      M           green
6   17       A   Mumbai      F             red
```

1.1 loc and iloc

- **loc** gets rows (or columns) with particular **labels** from the index.
- **iloc** gets rows (or columns) at particular positions in the **index** (so it only takes integers).

1.1.1 Find all the rows based on any condition in a column

We can solve types of queries with a simple line of code using pandas.DataFrame.loc[]. We just need to pass the condition within the loc statement.

[3]:
```
data.loc[data.age >= 15]
# data[data.age >= 15]
```

[3]:
```
   age section    city gender favourite_color
1   22       B   Delhi      F            blue
3   21       B   Delhi      M            blue
6   17       A  Mumbai      F             red
```

1.1.2 Find all the rows with more than one condition

[4]:
```
data.loc[(data.age >= 12) & (data.gender == 'M')]
# data[(data.age >= 12) & (data.gender == 'M')]
```

[4]:
```
   age section    city gender favourite_color
3   21       B   Delhi      M            blue
4   12       B  Mumbai      M           black
```

1.1.3 Select only required columns with a condition

[5]:
```
data.loc[(data.age >= 12), ['city', 'gender', 'age']]
# data[(data.age >= 12), ['city', 'gender', 'age']] # Name: age, dtype: bool,␣
↪['city', 'gender', 'age'])' is an invalid key
```

[5]:
```
     city gender  age
1   Delhi      F   22
2  Mumbai      F   13
3   Delhi      M   21
4  Mumbai      M   12
6  Mumbai      F   17
```

Update the values of a particular column on selected rows We can do this by running a for loop as well but if our dataset is big in size, then it would take forever to complete the task. Using loc in Pandas, we can do this within seconds, even on bigger datasets!

We just need to specify the condition followed by the target column and then assign the value with which we want to update

[6]:
```
data
```

[6]:
```
   age section     city gender favourite_color
0   10       A  Gurgaon      M             red
1   22       B    Delhi      F            blue
2   13       C   Mumbai      F          yellow
3   21       B    Delhi      M            blue
4   12       B   Mumbai      M           black
5   11       A    Delhi      M           green
6   17       A   Mumbai      F             red
```

```
[7]: data.loc[(data.age >= 12),['section','age']]
```

```
[7]:   section  age
     1       B   22
     2       C   13
     3       B   21
     4       B   12
     6       A   17
```

```
[8]: data.loc[(data.age >= 12),['section']] = 'M'
     display (data)
```

```
   age section     city gender favourite_color
0   10       A  Gurgaon      M             red
1   22       M    Delhi      F            blue
2   13       M   Mumbai      F          yellow
3   21       M    Delhi      M            blue
4   12       M   Mumbai      M           black
5   11       A    Delhi      M           green
6   17       M   Mumbai      F             red
```

1.1.4 Update the values of multiple columns on selected rows

```
[9]: # update multiple columns with condition
     display (data.loc[(data.age >= 20), ['section', 'city', 'age']])
     data.loc[(data.age >= 20), ['section', 'city']] = ['S', 'Pune']
     display (data.loc[(data.age >= 20), ['section', 'city', 'age']])
```

```
  section   city  age
1       M  Delhi   22
3       M  Delhi   21

  section  city  age
1       S  Pune   22
3       S  Pune   21
```

1.1.5 Select rows with indices using iloc

```
[10]: data
```

```
[10]:   age section     city gender favourite_color
      0   10       A  Gurgaon      M             red
      1   22       S     Pune      F            blue
      2   13       M   Mumbai      F          yellow
      3   21       S     Pune      M            blue
```

```
4   12        M   Mumbai      M            black
5   11        A    Delhi      M            green
6   17        M   Mumbai      F              red
```

[11]:
```
# select rows with indexes
data.iloc[[0,2]]
# 1st the row and the 3rd row
```

[11]:
```
   age section     city gender favourite_color
0   10       A  Gurgaon      M             red
2   13       M   Mumbai      F          yellow
```

1.1.6 Select rows with particular indices and particular columns == Slicing

[12]:
```
# select rows with particular indexes and particular columns
data.iloc[[0,2],[1,3]]
```

[12]:
```
  section gender
0       A      M
2       M      F
```

Select a range of rows and columns using iloc

[13]:
```
display (data)
data.iloc[1:3, 2:4]
```

```
   age section     city gender favourite_color
0   10       A  Gurgaon      M             red
1   22       S     Pune      F            blue
2   13       M   Mumbai      F          yellow
3   21       S     Pune      M            blue
4   12       M   Mumbai      M           black
5   11       A    Delhi      M           green
6   17       M   Mumbai      F             red
```

[13]:
```
     city gender
1    Pune      F
2  Mumbai      F
```

1.1.7 Operation 1: Use of Relational,logical and comparison operations

[14]:
```
datasetExample = pd.read_csv('FinalOutput.csv')
display(datasetExample)
```

```
   eid     ename  esal  yearlySalary department  UpdatedYearlySalary
0    1  Prashant  1000         12000         HR              13200.0
```

```
1    2       Amar  2000         24000        Ops             25200.0
2    3     Chitra  2000         24000      Admin             25200.0
3    3     Chitra  2000         24000      Admin             25200.0
4    4   Utkarsha  9878        118536        Ops            124462.8
5    5       Ajit  9999        119988         HR            125987.4
```

[15]:
```
# 1. Extract the records of employees whose esal is greater than 5000
datasetExample.loc[datasetExample.esal > 5000]
```

[15]:
```
   eid     ename  esal  yearlySalary department  UpdatedYearlySalary
4    4  Utkarsha  9878        118536        Ops             124462.8
5    5      Ajit  9999        119988         HR             125987.4
```

[17]:
```
# 2. Extract only ename and department of employees whose esal is greater than␣
↪5000
datasetExample.loc[datasetExample.esal > 5000, ['ename', 'department']]
```

[17]:
```
      ename department
4  Utkarsha        Ops
5      Ajit         HR
```

[18]:
```
# 3. Extract those records whose sal is greater than 6500 and belongs to Ops␣
↪dept
# datasetExample.esal > 6500 # sal is greater than 6500
# datasetExample.department == 'Ops' # belongs to Ops dept

# (datasetExample.esal > 6500) & (datasetExample.department == 'Ops') # sal is␣
↪greater than 6500 and belongs to Ops dept

datasetExample.loc[(datasetExample.esal > 6500) & (datasetExample.department ==␣
↪'Ops')]
```

[18]:
```
   eid     ename  esal  yearlySalary department  UpdatedYearlySalary
4    4  Utkarsha  9878        118536        Ops             124462.8
```

[22]:
```
# 4. Display the name of employees whose salary is greater than 5000 and␣
↪belongs to HR dept
datasetExample.loc[(datasetExample.esal > 5000) & (datasetExample.department ==␣
↪'HR'), ['ename']]#, 'esal', 'department']]
```

[22]:
```
  ename
5  Ajit
```

[28]:
```
datasetExample.loc[(datasetExample.ename == 'Prashant') | (datasetExample.ename␣
↪== 'Chitra')] # or |
[28]:    eid     ename  esal  yearlySalary department  UpdatedYearlySalary
      0    1  Prashant  1000         12000         HR              13200.0
      2    3    Chitra  2000         24000      Admin              25200.0
      3    3    Chitra  2000         24000      Admin              25200.0
```

2 Assignment

- **Replace the salary of the employee as 7000 if the current salary is less than 1500**

2.0.1 Operation 2 - Dealing with Duplicate Rows

```
[29]: display (datasetExample)
```

```
   eid     ename  esal  yearlySalary department  UpdatedYearlySalary
0    1  Prashant  1000         12000         HR              13200.0
1    2      Amar  2000         24000        Ops              25200.0
2    3    Chitra  2000         24000      Admin              25200.0
3    3    Chitra  2000         24000      Admin              25200.0
4    4  Utkarsha  9878        118536        Ops             124462.8
5    5      Ajit  9999        119988         HR             125987.4
```

```
[30]: datasetExample.drop_duplicates(inplace = True)
      display (datasetExample)

      # Designed only for duplicate rows
      # This will update the Dataframe
```

```
   eid     ename  esal  yearlySalary department  UpdatedYearlySalary
0    1  Prashant  1000         12000         HR              13200.0
1    2      Amar  2000         24000        Ops              25200.0
2    3    Chitra  2000         24000      Admin              25200.0
4    4  Utkarsha  9878        118536        Ops             124462.8
5    5      Ajit  9999        119988         HR             125987.4
```

The major demerit of DropDuplicate is the index is never reset automatically which may impact the fetch cycle of the data when performing EDA or Statistical Modelling.

2.0.2 Operation 3 - Groupby in Pandas

[31]:
```
data = {'Company':['GOOG','GOOG','MSFT','MSFT','FB','FB'],
       'Person':['Sam','Charlie','Amy','Vanessa','Carl','Sarah'],
       'Sales':[200,120,340,124,243,350]}

df = pd.DataFrame(data)

display(df)
```

```
  Company   Person  Sales
0    GOOG      Sam    200
1    GOOG  Charlie    120
2    MSFT      Amy    340
3    MSFT  Vanessa    124
4      FB     Carl    243
5      FB    Sarah    350
```

[33]:
```
# We want to look at average sales companywise
df.groupby('Company').mean()
```

[33]:
```
         Sales
Company
FB       296.5
GOOG     160.0
MSFT     232.0
```

[35]:
```
# We want to look at std sales companywise
df.groupby('Company').std()
```

[35]:
```
              Sales
Company
FB        75.660426
GOOG      56.568542
MSFT     152.735065
```

[36]:
```
df.groupby('Company').count()
```

[36]:
```
         Person  Sales
Company
FB            2      2
GOOG          2      2
MSFT          2      2
```

[37]:
```
df.describe()
```

[37]:
```
            Sales
count    6.000000
mean   229.500000
```

```
std     100.899455
min     120.000000
25%     143.000000
50%     221.500000
75%     315.750000
max     350.000000
```

[38]:
```
by_comp = df.groupby('Company')
```

[39]:
```
by_comp.describe()
```

[39]:
```
        Sales
        count   mean         std    min     25%    50%     75%    max
Company
FB        2.0  296.5   75.660426  243.0  269.75  296.5  323.25  350.0
GOOG      2.0  160.0   56.568542  120.0  140.00  160.0  180.00  200.0
MSFT      2.0  232.0  152.735065  124.0  178.00  232.0  286.00  340.0
```

[40]:
```
by_comp.describe().transpose()
```

[40]:
```
Company              FB        GOOG        MSFT
Sales count    2.000000    2.000000    2.000000
      mean   296.500000  160.000000  232.000000
      std     75.660426   56.568542  152.735065
      min    243.000000  120.000000  124.000000
      25%    269.750000  140.000000  178.000000
      50%    296.500000  160.000000  232.000000
      75%    323.250000  180.000000  286.000000
      max    350.000000  200.000000  340.000000
```

[41]:
```
by_comp.describe().transpose()['GOOG']
```

[41]:
```
Sales  count      2.000000
       mean     160.000000
       std       56.568542
       min      120.000000
       25%      140.000000
       50%      160.000000
       75%      180.000000
       max      200.000000
Name: GOOG, dtype: float64
```

2.0.3 Operation 4: Performing Merge Operations in Pandas

Merging is the Pandas operation that performs database joins on objects

```
[42]: dfExample1 = pd.DataFrame([[4,'QA'],[1,'HR'],[3,'Dev'],[2,'Ops']] ,␣
       ↪columns=['eid','dept'])
      display (dfExample1)
      print ()
      dfExample2 = pd.DataFrame([[1,'Prashant'],[2,'Gokul'],[3,'Guna']] ,␣
       ↪columns=['eid','ename'])
      display (dfExample2)
```

```
   eid dept
0    4   QA
1    1   HR
2    3  Dev
3    2  Ops



   eid     ename
0    1  Prashant
1    2     Gokul
2    3      Guna
```

```
[43]: resultDF = pd.merge(dfExample1, dfExample2)
      display (resultDF)

      # The merge worked in this case because both dataframe have the common column␣
       ↪eid
      # that too with the same name
```

```
   eid dept     ename
0    1   HR  Prashant
1    3  Dev      Guna
2    2  Ops     Gokul
```

```
[44]: resultDF = pd.merge(dfExample2, dfExample1)
      display (resultDF)
```

```
   eid     ename dept
0    1  Prashant   HR
1    2     Gokul  Ops
2    3      Guna  Dev
```

```
[45]: dept = pd.DataFrame([[4,'QA'],[1,'HR'],[3,'Dev'],[2,'Ops']] ,␣
       ↪columns=['eid','dept'])
      emp = pd.DataFrame([[1,'Prashant'],[2,'Gokul'],[3,'Guna']] ,␣
       ↪columns=['empid','ename'])
```

```
display(dept)
print ()
display(emp)
```

```
   eid dept
0    4   QA
1    1   HR
2    3  Dev
3    2  Ops
```

```
   empid     ename
0      1  Prashant
1      2     Gokul
2      3      Guna
```

[46]:
```
resultDF2 = pd.merge(dept, emp)
display (resultDF2)

# MergeError: No common columns to perform merge on.
```

```
      ␣
↪---------------------------------------------------------------------------

       MergeError                                Traceback (most recent call␣
↪last)

       <ipython-input-46-d0f7545afc11> in <module>
   ----> 1 resultDF2 = pd.merge(dept, emp)
         2 display (resultDF2)


       /usr/local/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)
        84         copy=copy,
        85         indicator=indicator,
   ---> 86         validate=validate,
        87     )
        88     return op.get_result()
```

```
    /usr/local/lib/python3.7/site-packages/pandas/core/reshape/merge.py in␣
↪__init__(self, left, right, how, on, left_on, right_on, axis, left_index,␣
↪right_index, sort, suffixes, copy, indicator, validate)
    618             warnings.warn(msg, UserWarning)
    619
--> 620         self._validate_specification()
    621
    622         # note this function has side effects


    /usr/local/lib/python3.7/site-packages/pandas/core/reshape/merge.py in␣
↪_validate_specification(self)
   1196                                 ron=self.right_on,
   1197                                 lidx=self.left_index,
-> 1198                                 ridx=self.right_index,
   1199                             )
   1200                         )


    MergeError: No common columns to perform merge on. Merge options:␣
↪left_on=None, right_on=None, left_index=False, right_index=False
```

[48]:
```
resultDF2 = pd.merge(dept, emp, left_on = 'eid', right_on = 'empid')
display(resultDF2)
```

```
   eid dept  empid     ename
0    1   HR      1  Prashant
1    3  Dev      3      Guna
2    2  Ops      2     Gokul
```

Left Join : Returns all rows from the left table, even if there are no matches in the right table

[51]:
```
world_champions={'Team':['India','Australia','West Indies','Pakistan','Sri␣
 ↪Lanka'], 'ICC_rank':[2,3,7,8,4],
                 'World_champions_Year':[2011,2015,1979,1992,1996], 'Points':
 ↪[874,787,753,673,855]}

chokers={'Team':['South Africa','New Zealand','Pakistan'], 'ICC_rank':
 ↪[1,5,9],'Points':[895,764,656]}

df1=pd.DataFrame(world_champions)
df2=pd.DataFrame(chokers)

display(df1)
print ()
```

```
display(df2)
```

```
          Team  ICC_rank  World_champions_Year  Points
0        India         2                  2011     874
1    Australia         3                  2015     787
2  West Indies         7                  1979     753
3     Pakistan         8                  1992     673
4    Sri Lanka         4                  1996     855
```

```
           Team  ICC_rank  Points
0  South Africa         1     895
1   New Zealand         5     764
2      Pakistan         9     656
```

[52]:
```
display (pd.merge(df1, df2, on = 'Team', how = "left"))
```

```
          Team  ICC_rank_x  World_champions_Year  Points_x  ICC_rank_y  \
0        India           2                  2011       874         NaN
1    Australia           3                  2015       787         NaN
2  West Indies           7                  1979       753         NaN
3     Pakistan           8                  1992       673         9.0
4    Sri Lanka           4                  1996       855         NaN

   Points_y
0       NaN
1       NaN
2       NaN
3     656.0
4       NaN
```

Right Join : Preserves the unmatched rows from the second (right) table, joining them with a NULL in the shape of the first (left) table

[55]:
```
world_champions={'Team':['India','Australia','West Indies','Pakistan','Sri␣
↪Lanka'], 'ICC_rank':[2,3,7,8,4],
                 'World_champions_Year':[2011,2015,1979,1992,1996], 'Points':
↪[874,787,753,673,855]}
chokers={'Team':['South Africa','New Zealand','India'],'ICC_rank':
↪[1,5,9],'Points':[895,764,656]}

df1=pd.DataFrame(world_champions)
df2=pd.DataFrame(chokers)

display(df1)
```

```
print ()
display(df2)
print ()

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

```
          Team  ICC_rank  World_champions_Year  Points
0        India         2                  2011     874
1    Australia         3                  2015     787
2  West Indies         7                  1979     753
3     Pakistan         8                  1992     673
4    Sri Lanka         4                  1996     855



           Team  ICC_rank  Points
0  South Africa         1     895
1   New Zealand         5     764
2         India         9     656



           Team  ICC_rank_x  World_champions_Year  Points_x  ICC_rank_y  \
0         India         2.0                2011.0     874.0           9
1  South Africa         NaN                   NaN       NaN           1
2   New Zealand         NaN                   NaN       NaN           5

   Points_y
0       656
1       895
2       764
```

Full Outer Join : Returns all records when there is a match in either left (table1) or right (table2) table records

[57]:
```
world_champions={'Team':['India','Australia','West Indies','Pakistan','Sri␣
 ↪Lanka'], 'ICC_rank':[2,3,7,8,4],
                  'World_champions_Year':[2011,2015,1979,1992,1996], 'Points':
 ↪[874,787,753,673,855]}
chokers={'Team':['South Africa','New Zealand','Zimbabwe'],'ICC_rank':
 ↪[1,5,9],'Points':[895,764,656]}

df1=pd.DataFrame(world_champions)
df2=pd.DataFrame(chokers)

display(df1)
```

```
print ()
display(df2)
print ()

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

```
          Team  ICC_rank  World_champions_Year  Points
0        India         2                  2011     874
1    Australia         3                  2015     787
2  West Indies         7                  1979     753
3     Pakistan         8                  1992     673
4    Sri Lanka         4                  1996     855



          Team  ICC_rank  Points
0  South Africa         1     895
1   New Zealand         5     764
2      Zimbabwe         9     656



          Team  ICC_rank_x  World_champions_Year  Points_x  ICC_rank_y  \
0        India         2.0                2011.0     874.0         NaN
1    Australia         3.0                2015.0     787.0         NaN
2  West Indies         7.0                1979.0     753.0         NaN
3     Pakistan         8.0                1992.0     673.0         NaN
4    Sri Lanka         4.0                1996.0     855.0         NaN
5 South Africa         NaN                   NaN       NaN         1.0
6  New Zealand         NaN                   NaN       NaN         5.0
7     Zimbabwe         NaN                   NaN       NaN         9.0

   Points_y
0       NaN
1       NaN
2       NaN
3       NaN
4       NaN
5     895.0
6     764.0
7     656.0
```

Inner Join : Selects all rows from both participating tables if there is a match between the columns

[59]:
```
world_champions={'Team':['India','Australia','West Indies','Pakistan','Sri␣
↪Lanka'], 'ICC_rank':[2,3,7,8,4],
                  'World_champions_Year':[2011,2015,1979,1992,1996], 'Points':
 ↪[874,787,753,673,855]}
chokers={'Team':['South Africa','New Zealand','India'],'ICC_rank':
 ↪[1,5,9],'Points':[895,764,656]}

df1=pd.DataFrame(world_champions)
df2=pd.DataFrame(chokers)

display(df1)
print ()
display(df2)
print ()
display(pd.merge(df1, df2, on='Team', how='inner'))
```

```
          Team  ICC_rank  World_champions_Year  Points
0        India         2                  2011     874
1    Australia         3                  2015     787
2  West Indies         7                  1979     753
3     Pakistan         8                  1992     673
4    Sri Lanka         4                  1996     855
```

```
           Team  ICC_rank  Points
0  South Africa         1     895
1   New Zealand         5     764
2         India         9     656
```

```
    Team  ICC_rank_x  World_champions_Year  Points_x  ICC_rank_y  Points_y
0  India           2                  2011       874           9       656
```

2.0.4 Operation 5: Concat Operation

```
[60]: empExample = pd.DataFrame([[1,'Prashant'],[2,'Gokul'],[3,'Guna']] ,␣
       ↪columns=['empid','ename'])

      empExample2 = pd.DataFrame([[4,'Nik'],[5,'Ashish'],[6,'Asha']] ,␣
       ↪columns=['empid','ename'])

      display (empExample)
      display (empExample2)
```

```
   empid     ename
0      1  Prashant
```

```
1      2     Gokul
2      3      Guna
```

```
   empid   ename
0      4     Nik
1      5  Ashish
2      6    Asha
```

[61]:
```
# Rowwise Concatenation --- Ensure the column names are same in all DFs.

resultEmp = pd.concat( [empExample,empExample2])
resultEmp
```

[61]:
```
   empid     ename
0      1  Prashant
1      2     Gokul
2      3      Guna
0      4       Nik
1      5    Ashish
2      6      Asha
```

[62]:
```
resultEmp = pd.concat( [empExample,empExample2] , axis = 1)
resultEmp
```

[62]:
```
   empid     ename  empid   ename
0      1  Prashant      4     Nik
1      2     Gokul      5  Ashish
2      3      Guna      6    Asha
```

[63]:
```
empExample3 = pd.DataFrame([[4,'Nik'],[5,'Ashish'],[6,'Asha']] ,␣
 ↪columns=['empid','empname'])
display (empExample3)
```

```
   empid empname
0      4     Nik
1      5  Ashish
2      6    Asha
```

[64]:
```
resultEmp2 = pd.concat( [empExample,empExample3] , axis = 0)
resultEmp2
```

[64]:
```
   empid     ename empname
0      1  Prashant     NaN
1      2     Gokul     NaN
2      3      Guna     NaN
0      4       NaN     Nik
```

```
1      5       NaN  Ashish
2      6       NaN    Asha
```

[65]:
```
empExample3.columns = ['empid','ename']
display (empExample3)
resultEmp2 = pd.concat( [empExample,empExample3] , axis = 0)
resultEmp2
```

```
   empid   ename
0      4     Nik
1      5  Ashish
2      6    Asha
```

[65]:
```
   empid     ename
0      1  Prashant
1      2     Gokul
2      3      Guna
0      4       Nik
1      5    Ashish
2      6      Asha
```

[ ]:
```

```