PandaBasics101

September 10, 2018

1 Panda Basics 101 : DataFrames

DataFrames (DF) : Creation class pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)

Two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects. The primary pandas data structure.

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

1.1 Create DF From List

```
In [2]: df = pd.DataFrame([1,2,3,4,5])
        df
```

```
Out[2]:    0
        0  1
        1  2
        2  3
        3  4
        4  5
```

1.2 Create DF From List of Lists

```
In [3]: df = pd.DataFrame([[1,2,3,4,5],[10,20,30,40,50], [100,120,130,140,150]])
        df
```

```
Out[3]:      0    1    2    3    4
        0    1    2    3    4    5
        1   10   20   30   40   50
        2  100  120  130  140  150
```

```
In [4]: # provide column labels- must be enough
        df = pd.DataFrame([[1,2,3,4,5],[10,20,30,40,50],
                           [100,120,130,140,150]],
                            columns=list("ABCDE"))
        df
```

```
Out[4]:      A    B    C    D    E
        0    1    2    3    4    5
        1   10   20   30   40   50
        2  100  120  130  140  150
```

1.3 Create DF From Dict

```
In [5]: #keys = column names  - if val = array, must be same size
        df = pd.DataFrame( {"foo": [1,2,3,4], "bar":[9,8,7,6], 'baz':[12,13,14,15] } )
        df
```

```
Out[5]:    foo  bar  baz
        0    1    9   12
        1    2    8   13
        2    3    7   14
        3    4    6   15
```

1.4 Create DF using random decimals and integers (good for experimenting)

1.5 Generate Random Decimals in range -1 to 1 using random.randn(rows,cols)

```
In [6]: df = pd.DataFrame(np.random.randn(5, 4), columns=list('ABCD'))
        df
```

```
Out[6]:           A         B         C         D
        0  0.875130 -0.567622 -1.184339  0.365110
        1 -0.886746  0.225891 -1.463829  1.227129
        2 -0.615083  1.105010 -0.284913  1.926585
        3  1.421741 -0.020012  0.519023 -0.619867
        4 -0.570243  1.274669 -0.128316 -1.354610
```

1.6 Generate Random Integers using: randint(low,hi,size=(rows,cols))

```
In [7]: df = pd.DataFrame(np.random.randint(low=0, high=10, size=(5, 5)),
                          columns=['a', 'b', 'c', 'd', 'e'])
        df
```

```
Out[7]:    a  b  c  d  e
        0  9  5  0  6  5
        1  2  5  8  2  0
        2  0  0  4  2  4
        3  2  3  6  9  4
        4  9  2  0  4  0
```

1.7 Create DF From CSV File

```
In [8]: df = pd.read_csv('worldcup.csv')
        df
```

```
Out[8]:    WorldCup  year     location      first          second        third  \
0     wc1930  1930      Uruguay    Uruguay       Argentina          USA
1     wc1934  1934        Italy      Italy  Czechoslovakia      Germany
2     wc1938  1938       France      Italy         Hungary       Brazil
3     wc1950  1950       Brazil    Uruguay          Brazil       Sweden
4     wc1954  1954  Switzerland  GermanyFR         Hungary      Austria
5     wc1958  1958       Sweden     Brazil          Sweden       France
6     wc1962  1962        Chile     Brazil  Czechoslovakia        Chile
7     wc1966  1966      England    England       GermanyFR     Portugal
8     wc1970  1970       Mexico     Brazil           Italy    GermanyFR
9     wc1974  1974      Germany  GermanyFR     Netherlands       Poland
10    wc1978  1978    Argentina  Argentina     Netherlands       Brazil
11    wc1982  1982        Spain      Italy       GermanyFR       Poland
12    wc1986  1986       Mexico  Argentina       GermanyFR       France
13    wc1990  1990        Italy  GermanyFR       Argentina        Italy
14    wc1994  1994          USA     Brazil           Italy       Sweden
15    wc1998  1998       France     France          Brazil      Croatia
16    wc2002  2002  Korea_Japan     Brazil         Germany       Turkey
17    wc2006  2006      Germany      Italy          France      Germany
18    wc2010  2010  SouthAfrica      Spain     Netherlands      Germany
19    wc2014  2014       Brazil    Germany       Argentina  Netherlands
20    wc2018  2018       Russia     France         Croatia      Belgium

           fourth  goalsScored  matchesPlayed  attendance
0      Yugoslavia           70             18      590549
1         Austria           70             17      363000
2          Sweden           84             18      375000
3           Spain           88             22     1045246
4         Uruguay           14             26      768607
5       GermanyFR          126             35      819810
6      Yugoslavia           89             32      893172
7    Soviet_Union           89             32     1563135
8         Uruguay           95             32     1603975
9          Brazil           97             38     1865753
10          Italy          102             38     1545791
11         France          146             52     2109723
12        Belgium          132             52     2394031
13        England          115             52     2516215
14       Bulgaria          141             52     3587538
15    Netherlands          171             64     2785100
16  KoreaRepublic          161             64     2705197
17       Portugal          147             64     3359439
18        Uruguay          145             64     3178856
19         Brazil          171             64     3386810
20        England          169             64     3430000
```

2 Creating New Cols/Rows

2.1 New Column Creation based on existing column(s)

```
In [9]: df['goalsPerMatch'] = df['goalsScored']/df['matchesPlayed']
        df
```

```
Out[9]:    WorldCup  year     location      first          second        third  \
        0    wc1930  1930      Uruguay    Uruguay       Argentina          USA
        1    wc1934  1934        Italy      Italy  Czechoslovakia      Germany
        2    wc1938  1938       France      Italy         Hungary       Brazil
        3    wc1950  1950       Brazil    Uruguay          Brazil       Sweden
        4    wc1954  1954  Switzerland  GermanyFR         Hungary      Austria
        5    wc1958  1958       Sweden     Brazil          Sweden       France
        6    wc1962  1962        Chile     Brazil  Czechoslovakia        Chile
        7    wc1966  1966      England    England       GermanyFR     Portugal
        8    wc1970  1970       Mexico     Brazil           Italy    GermanyFR
        9    wc1974  1974      Germany  GermanyFR     Netherlands       Poland
        10   wc1978  1978    Argentina  Argentina     Netherlands       Brazil
        11   wc1982  1982        Spain      Italy       GermanyFR       Poland
        12   wc1986  1986       Mexico  Argentina       GermanyFR       France
        13   wc1990  1990        Italy  GermanyFR       Argentina        Italy
        14   wc1994  1994          USA     Brazil           Italy       Sweden
        15   wc1998  1998       France     France          Brazil      Croatia
        16   wc2002  2002  Korea_Japan     Brazil         Germany       Turkey
        17   wc2006  2006      Germany      Italy          France      Germany
        18   wc2010  2010  SouthAfrica      Spain     Netherlands      Germany
        19   wc2014  2014       Brazil    Germany       Argentina  Netherlands
        20   wc2018  2018       Russia     France         Croatia      Belgium

                  fourth  goalsScored  matchesPlayed  attendance  goalsPerMatch
        0     Yugoslavia           70             18      590549       3.888889
        1        Austria           70             17      363000       4.117647
        2         Sweden           84             18      375000       4.666667
        3          Spain           88             22     1045246       4.000000
        4        Uruguay           14             26      768607       0.538462
        5      GermanyFR          126             35      819810       3.600000
        6     Yugoslavia           89             32      893172       2.781250
        7    Soviet_Union          89             32     1563135       2.781250
        8        Uruguay           95             32     1603975       2.968750
        9         Brazil           97             38     1865753       2.552632
        10         Italy          102             38     1545791       2.684211
        11        France          146             52     2109723       2.807692
        12       Belgium          132             52     2394031       2.538462
        13       England          115             52     2516215       2.211538
        14      Bulgaria          141             52     3587538       2.711538
        15   Netherlands          171             64     2785100       2.671875
        16 KoreaRepublic          161             64     2705197       2.515625
        17      Portugal          147             64     3359439       2.296875
```

```
18          Uruguay          145             64    3178856        2.265625
19           Brazil          171             64    3386810        2.671875
20          England          169             64    3430000        2.640625
```

```
In [10]: # sort on this value - creates a copy
         df2 = df.sort_values(by='goalsPerMatch', ascending=False)
         df2
```

```
Out[10]:    WorldCup  year     location      first          second        third  \
         2     wc1938  1938       France      Italy         Hungary       Brazil
         1     wc1934  1934        Italy      Italy  Czechoslovakia      Germany
         3     wc1950  1950       Brazil    Uruguay          Brazil       Sweden
         0     wc1930  1930      Uruguay    Uruguay       Argentina          USA
         5     wc1958  1958       Sweden     Brazil          Sweden       France
         8     wc1970  1970       Mexico     Brazil           Italy    GermanyFR
         11    wc1982  1982        Spain      Italy       GermanyFR       Poland
         6     wc1962  1962        Chile     Brazil  Czechoslovakia        Chile
         7     wc1966  1966      England    England       GermanyFR     Portugal
         14    wc1994  1994          USA     Brazil           Italy       Sweden
         10    wc1978  1978    Argentina  Argentina     Netherlands       Brazil
         15    wc1998  1998       France     France          Brazil      Croatia
         19    wc2014  2014       Brazil    Germany       Argentina  Netherlands
         20    wc2018  2018       Russia     France         Croatia      Belgium
         9     wc1974  1974      Germany  GermanyFR     Netherlands       Poland
         12    wc1986  1986       Mexico  Argentina       GermanyFR       France
         16    wc2002  2002  Korea_Japan     Brazil         Germany       Turkey
         17    wc2006  2006      Germany      Italy          France      Germany
         18    wc2010  2010  SouthAfrica      Spain     Netherlands      Germany
         13    wc1990  1990        Italy  GermanyFR       Argentina        Italy
         4     wc1954  1954  Switzerland  GermanyFR         Hungary      Austria

                     fourth  goalsScored  matchesPlayed  attendance  goalsPerMatch
         2           Sweden           84             18      375000       4.666667
         1          Austria           70             17      363000       4.117647
         3            Spain           88             22     1045246       4.000000
         0       Yugoslavia           70             18      590549       3.888889
         5        GermanyFR          126             35      819810       3.600000
         8          Uruguay           95             32     1603975       2.968750
         11          France          146             52     2109723       2.807692
         6       Yugoslavia           89             32      893172       2.781250
         7     Soviet_Union           89             32     1563135       2.781250
         14        Bulgaria          141             52     3587538       2.711538
         10           Italy          102             38     1545791       2.684211
         15     Netherlands          171             64     2785100       2.671875
         19          Brazil          171             64     3386810       2.671875
         20         England          169             64     3430000       2.640625
         9           Brazil           97             38     1865753       2.552632
         12         Belgium          132             52     2394031       2.538462
```

```
16  KoreaRepublic          161              64     2705197        2.515625
17       Portugal          147              64     3359439        2.296875
18        Uruguay          145              64     3178856        2.265625
13        England          115              52     2516215        2.211538
4         Uruguay           14              26      768607        0.538462
```

2.2 Applying functions to DataFrames for each row or column

When we df.apply(fn, axis=) a Python function to a DataFrame, each row (axis=0) or col (axis=1) is passed as a parameter to the function and a new DataFrame is created.

axis confusion

- axis=1: All the COLS in each Row
- axis=0: all the ROWS in each COL

```
In [11]: df = pd.DataFrame([[1,1,1,1], [100,200,300,400], [1000,2000,3000,4000]],
                           columns=list('ABCD'))
         df
```

```
Out[11]:       A     B     C     D
         0     1     1     1     1
         1   100   200   300   400
         2  1000  2000  3000  4000
```

```
In [12]: #axis=1: All the COLS in each Row
         #axis=0: all the ROWS in each COL
         def addAll(z):
             #the DF will pass either a row or column (a Series) to this function
             return z.sum()

         df2 = df.apply(addAll, axis=1)
         print (df2)

         #now with axis=0  All ROWS for each COL
         #note: the column names are now the row indexes
         df3 = df.apply(addAll, axis=0)
         print (df3)
```

```
0        4
1     1000
2    10000
dtype: int64
A    1101
B    2201
C    3301
D    4401
dtype: int64
```

2.3 Using Boolean DataFrame to select rows from a DataFrame

Creating a boolean index (a single column of true/false) is useful when you want to select subset of rows from your DataFrame. The strategy is: - create new DF based on logical operators over the orginal DF - use the new boolean DF to select only those rows == True

```
In [13]: df = pd.DataFrame([ [2,4,6,8], [6,12,44,67], [6,6,6,26], [3,4,5,6]],
                           columns=list("abcd"))
         df
```

```
Out[13]:    a   b   c   d
         0  2   4   6   8
         1  6  12  44  67
         2  6   6   6  26
         3  3   4   5   6
```

```
In [14]: #simple boolean expession
         dbool = df['a'] >= 6
         dbool
```

```
Out[14]: 0    False
         1     True
         2     True
         3    False
         Name: a, dtype: bool
```

```
In [15]: #compound boolean condition  - needs parens around each expression
         dbool = (df['a'] >= 6) & (df['d'] > df['a']*2)
         dbool
```

```
Out[15]: 0    False
         1     True
         2     True
         3    False
         dtype: bool
```

```
In [16]: #apply boolean to original DF
         df3 = df[dbool]
         df3
```

```
Out[16]:    a   b   c   d
         1  6  12  44  67
         2  6   6   6  26
```

3 Panda Selection and Slicing from Existing DF

```
In [17]: # set up a dataframe
         df = pd.DataFrame(np.random.randint(0,100,size=(5, 5)), columns=list('abcde'))
         df
```

```
Out[17]:     a   b   c   d   e
         0  64  70  98  63   7
         1  46  88  30  62  11
         2   3  93  44  89  32
         3  38  55  72  54  47
         4  84  22  50  84  74
```

3.1 Select Column or Columns from DF

Index refers to entire column (unlike Python Dict) Note: When selecting a single column or row, a Series is returned

3.1.1 Select Single Column (as Series)

```
In [18]: df2 = df['b']
         print (df2)
         print (type(df2))
```

```
0    70
1    88
2    93
3    55
4    22
Name: b, dtype: int32
<class 'pandas.core.series.Series'>
```

3.1.2 Select Multiple Columns (as DataFrame) - pass List

```
In [19]: df3 = df[ ['a','c','e'] ]
         df3
```

```
Out[19]:     a   c   e
         0  64  98   7
         1  46  30  11
         2   3  44  32
         3  38  72  47
         4  84  50  74
```

3.1.3 Selecting Single row and single data item using .iloc

```
In [20]: # show our DF
         df
```

```
Out[20]:     a   b   c   d   e
         0  64  70  98  63   7
         1  46  88  30  62  11
         2   3  93  44  89  32
         3  38  55  72  54  47
         4  84  22  50  84  74
```

3.1.4 df.iloc expects a numeric index - refers to row

Single integer returns the row

```
In [21]: #single index value -> row
         df.iloc[2]
```

```
Out[21]: a     3
         b    93
         c    44
         d    89
         e    32
         Name: 2, dtype: int32
```

3.1.5 df.iloc with a list of labels -- returns multiple rows

```
In [22]: df.iloc[ [1,3,4]]
```

```
Out[22]:     a   b   c   d   e
         1  46  88  30  62  11
         3  38  55  72  54  47
         4  84  22  50  84  74
```

3.1.6 Single Data item with .iloc [row,col] -> yields single value

```
In [23]: df.iloc[1,2]
```

```
Out[23]: 30
```

3.1.7 Rows and Columns with df.iloc and numeric slice ranges [1:3,2:4]

```
In [24]: # use : for all rows
         df.iloc[:,2:5]
```

```
Out[24]:     c   d   e
         0  98  63   7
         1  30  62  11
         2  44  89  32
         3  72  54  47
         4  50  84  74
```

```
In [25]: # range over rows and cols with integer indexes
         df.iloc[2:4,2:4]
```

```
Out[25]:     c   d
         2  44  89
         3  72  54
```

3.2 Selecting with Labels using df.loc

```
In [26]: # show df
         df

Out[26]:     a   b   c   d   e
         0  64  70  98  63   7
         1  46  88  30  62  11
         2   3  93  44  89  32
         3  38  55  72  54  47
         4  84  22  50  84  74
```

```
In [27]: # one parameter gives us the entire row. (note: row label is an integer)
         # Sanity check:  df[2]       -> column 2 ( if such a label exists)
         #                df.loc[2] -> row  with 'label 2' - not the second in series
         print (df.loc[2])

a     3
b    93
c    44
d    89
e    32
Name: 2, dtype: int32
```

```
In [28]: # two parameters give single element
         df.loc[2,'b']

Out[28]: 93
```

```
In [29]: # range of values
         df.loc[2:4, 'c':'e']

Out[29]:     c   d   e
         2  44  89  32
         3  72  54  47
         4  50  84  74
```

3.3 Select based on column value using in or .isin

Pandas isin() method is used to filter data frames. isin() method helps in selecting rows with having a single or multiple values in a particular column.

```
In [30]: df = pd.read_csv('worldcup.csv')
         df

Out[30]:   WorldCup  year   location    first        second        third  \
         0   wc1930  1930    Uruguay  Uruguay     Argentina          USA
         1   wc1934  1934      Italy    Italy  Czechoslovakia      Germany
         2   wc1938  1938     France    Italy       Hungary       Brazil
```

```
3    wc1950  1950       Brazil    Uruguay          Brazil       Sweden
4    wc1954  1954  Switzerland  GermanyFR         Hungary      Austria
5    wc1958  1958       Sweden     Brazil          Sweden       France
6    wc1962  1962        Chile     Brazil  Czechoslovakia        Chile
7    wc1966  1966      England    England       GermanyFR     Portugal
8    wc1970  1970       Mexico     Brazil           Italy    GermanyFR
9    wc1974  1974      Germany  GermanyFR     Netherlands       Poland
10   wc1978  1978    Argentina  Argentina     Netherlands       Brazil
11   wc1982  1982        Spain      Italy       GermanyFR       Poland
12   wc1986  1986       Mexico  Argentina       GermanyFR       France
13   wc1990  1990        Italy  GermanyFR       Argentina        Italy
14   wc1994  1994          USA     Brazil           Italy       Sweden
15   wc1998  1998       France     France          Brazil      Croatia
16   wc2002  2002  Korea_Japan     Brazil         Germany       Turkey
17   wc2006  2006      Germany      Italy          France      Germany
18   wc2010  2010  SouthAfrica      Spain     Netherlands      Germany
19   wc2014  2014       Brazil    Germany       Argentina  Netherlands
20   wc2018  2018       Russia     France         Croatia      Belgium

           fourth  goalsScored  matchesPlayed  attendance
0      Yugoslavia           70             18      590549
1         Austria           70             17      363000
2          Sweden           84             18      375000
3           Spain           88             22     1045246
4         Uruguay           14             26      768607
5       GermanyFR          126             35      819810
6      Yugoslavia           89             32      893172
7    Soviet_Union           89             32     1563135
8         Uruguay           95             32     1603975
9          Brazil           97             38     1865753
10          Italy          102             38     1545791
11         France          146             52     2109723
12        Belgium          132             52     2394031
13        England          115             52     2516215
14       Bulgaria          141             52     3587538
15    Netherlands          171             64     2785100
16  KoreaRepublic          161             64     2705197
17       Portugal          147             64     3359439
18        Uruguay          145             64     3178856
19         Brazil          171             64     3386810
20        England          169             64     3430000
```

```
In [31]: #create a bool series
         newdf = df['first'].isin(['Brazil'])
         #use to filter
         df[newdf]
```

```
Out[31]:    WorldCup  year     location   first          second      third   \
         5    wc1958  1958       Sweden  Brazil          Sweden     France
```

```
6     wc1962  1962        Chile  Brazil  Czechoslovakia      Chile
8     wc1970  1970       Mexico  Brazil           Italy  GermanyFR
14    wc1994  1994          USA  Brazil           Italy     Sweden
16    wc2002  2002  Korea_Japan  Brazil         Germany     Turkey

            fourth  goalsScored  matchesPlayed  attendance
5        GermanyFR          126             35      819810
6       Yugoslavia           89             32      893172
8          Uruguay           95             32     1603975
14        Bulgaria          141             52     3587538
16  KoreaRepublic          161             64     2705197
```

```
In [32]: newdf = df['first'].isin(['Brazil', 'France'])
         #use to filter
         df[newdf]
```

```
Out[32]:    WorldCup  year     location   first          second      third  \
         5    wc1958  1958       Sweden  Brazil          Sweden     France
         6    wc1962  1962        Chile  Brazil  Czechoslovakia      Chile
         8    wc1970  1970       Mexico  Brazil           Italy  GermanyFR
         14   wc1994  1994          USA  Brazil           Italy     Sweden
         15   wc1998  1998       France  France          Brazil    Croatia
         16   wc2002  2002  Korea_Japan  Brazil         Germany     Turkey
         20   wc2018  2018       Russia  France         Croatia    Belgium

                     fourth  goalsScored  matchesPlayed  attendance
         5        GermanyFR          126             35      819810
         6       Yugoslavia           89             32      893172
         8          Uruguay           95             32     1603975
         14        Bulgaria          141             52     3587538
         15     Netherlands          171             64     2785100
         16   KoreaRepublic          161             64     2705197
         20         England          169             64     3430000
```

4 Get and set single values from DataFrame with .at and .iat

4.1 Get/Set with .at (using labels - not indexes)

```
In [33]: # set up a dataframe
         df = pd.DataFrame(np.random.randint(0,100,size=(5, 5)), columns=list('abcde'))
         df
```

```
Out[33]:     a   b   c   d   e
         0  51  27  17   3  87
         1  88  47  64  66  76
         2  88  42  70  54  95
         3  32  12  55  20   5
         4  77  76  54   8   3
```

```
In [34]: # get value
         df.at[1,'c']

Out[34]: 64

In [35]: # set value
         df.at[1,'c'] = 1002
         df

Out[35]:     a   b     c   d   e
         0  51  27    17   3  87
         1  88  47  1002  66  76
         2  88  42    70  54  95
         3  32  12    55  20   5
         4  77  76    54   8   3
```

4.2 Get/Set with .iat (using integer indexes)

```
In [36]: df.iat[3,3]

Out[36]: 20

In [37]: df.iat[3,3] = 3333
         df

Out[37]:     a   b     c     d   e
         0  51  27    17     3  87
         1  88  47  1002    66  76
         2  88  42    70    54  95
         3  32  12    55  3333   5
         4  77  76    54     8   3
```

5 Sorting

```
In [38]: #set up
         df = pd.DataFrame({
           'col1' : ['A', 'A', 'B', np.nan, 'D', 'C'],
           'col2' : [2, 1, 9, 8, 7, 4],
           'col3': [0, 1, 9, 4, 2, 3],
          })
         df

Out[38]:   col1  col2  col3
         0    A     2     0
         1    A     1     1
         2    B     9     9
         3  NaN     8     4
         4    D     7     2
         5    C     4     3
```

```
In [39]: df.sort_values(by=['col1'])
```

```
Out[39]:   col1  col2  col3
         0    A     2     0
         1    A     1     1
         2    B     9     9
         5    C     4     3
         4    D     7     2
         3  NaN     8     4
```

```
In [40]: #sort multiple columns
         df.sort_values(by=['col1', 'col2'])
```

```
Out[40]:   col1  col2  col3
         1    A     1     1
         0    A     2     0
         2    B     9     9
         5    C     4     3
         4    D     7     2
         3  NaN     8     4
```

```
In [41]: #sort multiple columns -  DESCENDING
         df.sort_values(by=['col1', 'col2'], ascending=False)
```

```
Out[41]:   col1  col2  col3
         4    D     7     2
         5    C     4     3
         2    B     9     9
         0    A     2     0
         1    A     1     1
         3  NaN     8     4
```

```
In [42]: # Put NAs first
         df.sort_values(by=['col1', 'col2'], ascending=False, na_position='first')
```

```
Out[42]:   col1  col2  col3
         3  NaN     8     4
         4    D     7     2
         5    C     4     3
         2    B     9     9
         0    A     2     0
         1    A     1     1
```

6 Dropping rows and cols with Unknown NaN values

DataFrame.dropna(axis=0, how=’any’, thresh=None, subset=None, inplace=False)ű

```
In [43]: #drop rows with NaN
         df.dropna()
```

```
Out[43]:   col1  col2  col3
         0    A     2     0
         1    A     1     1
         2    B     9     9
         4    D     7     2
         5    C     4     3
```

```
In [44]: #drop columns
         df.dropna(axis=1)
```

```
Out[44]:    col2  col3
         0     2     0
         1     1     1
         2     9     9
         3     8     4
         4     7     2
         5     4     3
```

7 GroupBy

When using groupby you get an 'Object' with methods you can use

```
In [45]: # Create dataframe from dict: default is keys are column headers
         raw_data = {'regiment': ['Nighthawks', 'Nighthawks', 'Nighthawks', 'Nighthawks', 'Drag
                 'company': ['1st', '1st', '2nd', '2nd', '1st', '1st', '2nd', '2nd','1st', '1s
                 'name': ['Miller', 'Jacobson', 'Ali', 'Milner', 'Cooze', 'Jacon', 'Ryaner', 'S
                 'preTestScore': [4, 24, 31, 2, 3, 4, 24, 31, 2, 3, 2, 3],
                 'postTestScore': [22, 94, 57, 62, 70, 25, 94, 57, 62, 70, 62, 70]}
         df = pd.DataFrame(raw_data, columns = ['regiment', 'company', 'name', 'preTestScore',
         df
```

```
Out[45]:       regiment company      name  preTestScore  postTestScore
         0   Nighthawks     1st    Miller             4             22
         1   Nighthawks     1st  Jacobson            24             94
         2   Nighthawks     2nd       Ali            31             57
         3   Nighthawks     2nd    Milner             2             62
         4     Dragoons     1st     Cooze             3             70
         5     Dragoons     1st     Jacon             4             25
         6     Dragoons     2nd    Ryaner            24             94
         7     Dragoons     2nd      Sone            31             57
         8       Scouts     1st     Sloan             2             62
         9       Scouts     1st     Piger             3             70
         10      Scouts     2nd     Riani             2             62
         11      Scouts     2nd       Ali             3             70
```

```
In [46]: # Create a groupby variable that groups our DataFrame by regiment
         groupby_regiment = df.groupby(['regiment'])
```

7.1 Get size of your Groups

```
In [47]: groupby_regiment.size()
```

```
Out[47]: regiment
         Dragoons      4
         Nighthawks    4
         Scouts        4
         dtype: int64
```

7.2 Compute sums and mean of groups : use .sum() and .mean()

```
In [48]: groupby_regiment.mean()
```

```
Out[48]:             preTestScore  postTestScore
         regiment
         Dragoons           15.50          61.50
         Nighthawks         15.25          58.75
         Scouts              2.50          66.00
```

7.2.1 add_prefix('string') when you sum() or mean() over columns

```
In [49]: groupby_regiment.mean().add_prefix('mean_')
```

```
Out[49]:             mean_preTestScore  mean_postTestScore
         regiment
         Dragoons                15.50               61.50
         Nighthawks              15.25               58.75
         Scouts                   2.50               66.00
```

7.3 Iterating over Groups

When iterating you can get the name of the group and a dataframe for the group

```
In [50]: for name, group in groupby_regiment:
             print ("Group name = ", name)
             print (group)
```

```
Group name =  Dragoons
   regiment company    name  preTestScore  postTestScore
4  Dragoons     1st   Cooze             3             70
5  Dragoons     1st   Jacon             4             25
6  Dragoons     2nd  Ryaner            24             94
7  Dragoons     2nd    Sone            31             57
Group name =  Nighthawks
     regiment company      name  preTestScore  postTestScore
0  Nighthawks     1st    Miller             4             22
1  Nighthawks     1st  Jacobson            24             94
2  Nighthawks     2nd       Ali            31             57
3  Nighthawks     2nd    Milner             2             62
```

```
Group name =  Scouts
   regiment company   name  preTestScore  postTestScore
8    Scouts      1st  Sloan             2             62
9    Scouts      1st  Piger             3             70
10   Scouts      2nd  Riani             2             62
11   Scouts      2nd    Ali             3             70
```

8 Ploting

Easy to plot one column vs another Include '%matplotlib inline' to get Jupyter to plot

pandas vs matplotlib

Under the hood, pandas plots graphs with the matplotlib library. This is usually pretty convenient since it allows you to just .plot your graphs, but since matplotlib is kind of a train wreck pandas inherits that confusion. Which .plot do I use?

When you use .plot on a dataframe, you sometimes pass things to it and sometimes you don't.

```
.plot plots the index against every column
.plot(x='col1') plots against a single specific column
.plot(x='col1', y='col2') plots one specific column against another specific column
```

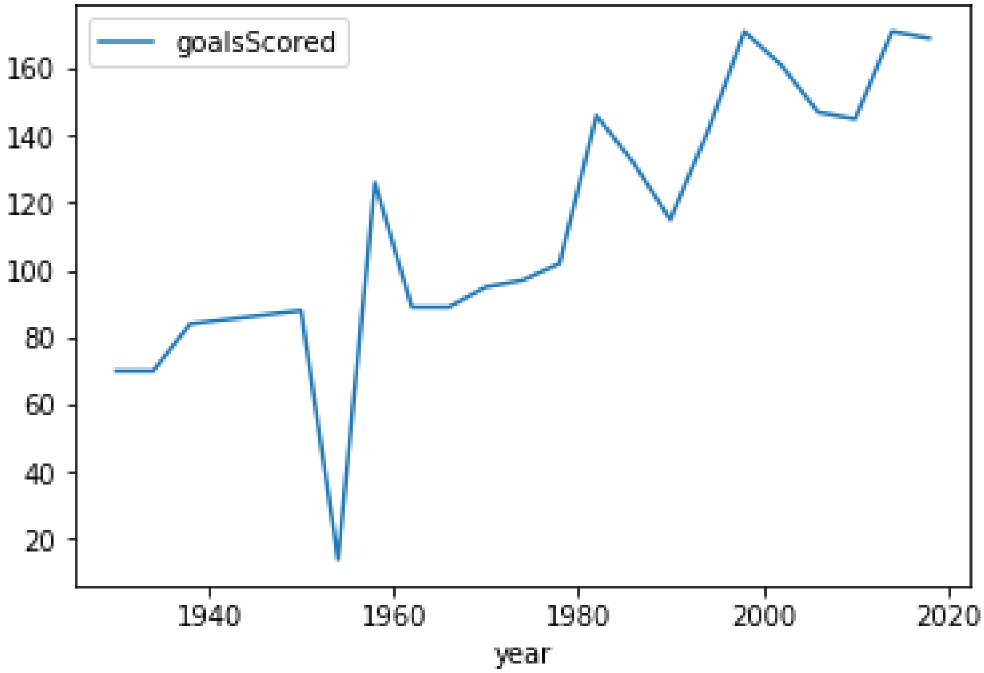
```
In [51]: import matplotlib.pyplot as plt
         %matplotlib inline
         #load data
         df = pd.read_csv('worldcup.csv')
         df.head()
```

```
Out[51]:   WorldCup  year     location      first          second    third      fourth  \
         0   wc1930  1930      Uruguay    Uruguay       Argentina      USA  Yugoslavia
         1   wc1934  1934        Italy      Italy  Czechoslovakia  Germany     Austria
         2   wc1938  1938       France      Italy         Hungary   Brazil      Sweden
         3   wc1950  1950       Brazil    Uruguay          Brazil   Sweden       Spain
         4   wc1954  1954  Switzerland  GermanyFR         Hungary  Austria     Uruguay

            goalsScored  matchesPlayed  attendance
         0           70             18      590549
         1           70             17      363000
         2           84             18      375000
         3           88             22     1045246
         4           14             26      768607
```

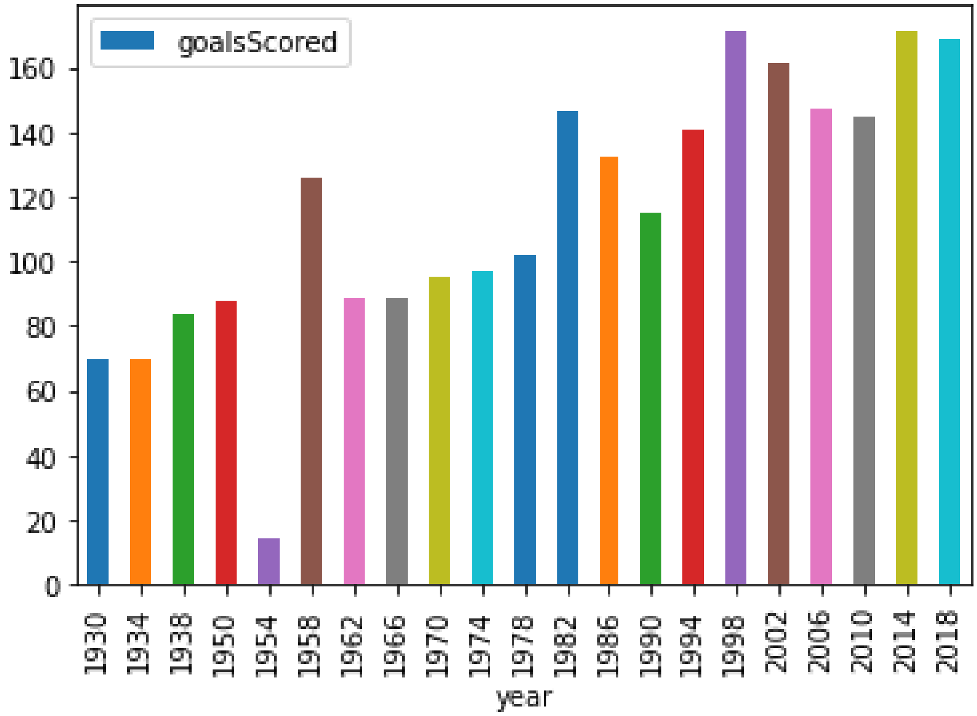
```
In [52]: df.plot(y='goalsScored', x='year')
```

```
Out[52]: <matplotlib.axes._subplots.AxesSubplot at 0x2123ad05780>
```



```
In [53]: df.plot(y='goalsScored', x='year', kind='bar')

Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x2123b015160>
```



8.0.1 Groupby and plots

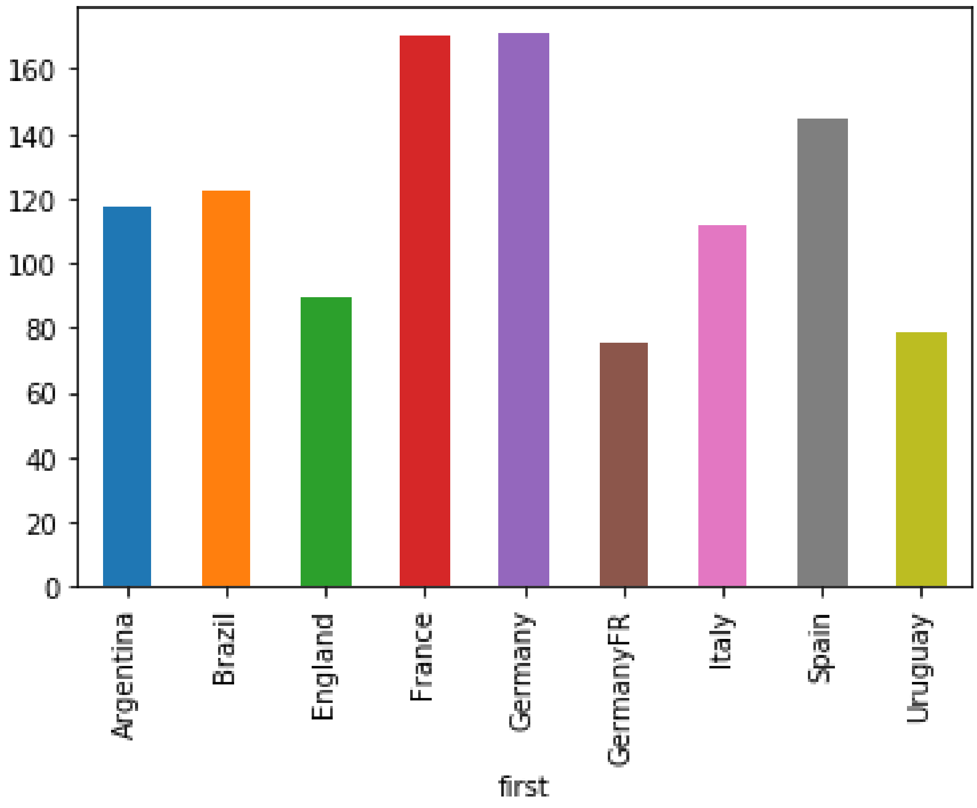
Let's look at worldcup and get a plot of goals scored by winning teams

```
In [54]: df.groupby('first')['goalsScored'].mean()
```

```
Out[54]: first
         Argentina    117.000000
         Brazil       122.400000
         England       89.000000
         France       170.000000
         Germany      171.000000
         GermanyFR     75.333333
         Italy        111.750000
         Spain        145.000000
         Uruguay       79.000000
         Name: goalsScored, dtype: float64
```

```
In [55]: # now add plot to that
         df.groupby('first')['goalsScored'].mean().plot(kind='bar')
```

```
Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x2123b0158d0>
```



9 pandas.DataFrame.to_json

DataFrame.to_json(path_or_buf=None, orient=None, date_format=None, double_precision=10, force_ascii=True, date_unit='ms', default_handler=None, lines=False, compression=None, index=True)

- Convert the object to a JSON string.
- orient defaults to 'columns' : for rows use 'index'
- Note NaN's and None will be converted to null and datetime objects will be converted to UNIX timestamps.

```
In [56]: #set up
         df = pd.DataFrame(np.random.randint(low=0, high=20, size=(5, 5)),
                           columns=['a', 'b', 'c', 'd', 'e'], index=['r1','r2','r3','r4','r5']
                        )
         print (df)
```

```
     a   b   c   d   e
r1  12   4   4   5  12
r2  17   6   0  11   0
r3  16  11   2  16   5
r4  14   1  10   2   5
r5  12  10   0  19  17
```

9.1 Create JSON from columns (default)

```
In [57]: s = df.to_json()
         print (s)
```

```
{"a":{"r1":12,"r2":17,"r3":16,"r4":14,"r5":12},"b":{"r1":4,"r2":6,"r3":11,"r4":1,"r5":10},"c":
```

9.2 Create JSON from rows using orient='index'

```
In [58]: s = df.to_json(orient='index')
         print (s)
```

```
{"r1":{"a":12,"b":4,"c":4,"d":5,"e":12},"r2":{"a":17,"b":6,"c":0,"d":11,"e":0},"r3":{"a":16,"b
```