Pandas and Alternatives

1. Import pandas aliased as 'pd' and numpy aliased as 'np'

In [3]:
```
import pandas as pd
import numpy as np
```

2. Create a DataFrame named 'df' by reading the file 'USCG.Search.Rescue.Stats.csv' using the pandas read_csv method.

In [4]:
```
df = pd.read_csv       ('USCG.Search.Rescue.Stats.csv')
```

3. View the top 5 rows of data using the DataFrame `head()` method.

In [5]:
```
df.head()
```

Out[5]:

	Fiscal Year	Cases	Responses	Sorties	Lives Saved	Lives Lost After CG Notification	Lives Lost Before CG Notification	Total	Lives Unaccounted For
1964	NaN	41525	NaN	2932	NaN	NaN	NaN	NaN	NaN
1965	NaN	38586	NaN	1984	NaN	NaN	NaN	NaN	NaN
1966	NaN	43366	NaN	2629	NaN	NaN	NaN	NaN	NaN
1967	NaN	42225	NaN	3028	NaN	NaN	NaN	NaN	NaN
1968	NaN	46922	NaN	2434	NaN	NaN	NaN	NaN	NaN

4. View the last 5 rows by using the DataFrame `tail()` method.

In [6]:
```
df.tail()
```

Out[6]:

	Fiscal Year	Cases	Responses	Sorties	Lives Saved	Lives Lost After CG Notification	Lives Lost Before CG Notification	Total	Lives Unaccounted For
2011	20512.0	43954	21566.0	3793	259.0	476.0	735.0	392.0	NaN
2012	19787.0	43940	21609.0	4037	284.0	429.0	713.0	440.0	NaN
2013	17803.0	38272	19420.0	3753	226.0	425.0	651.0	252.0	NaN
2014	17508.0	38282	19032.0	3443	170.0	425.0	595.0	308.0	NaN
2015	16456.0	37215	18781.0	3536	169.0	434.0	603.0	330.0	NaN

5. View the values in the 'Cases' column using dot syntax, bracket syntax, `loc[]` or `iloc[]`.

In [7]:
```
df.loc[:,'Cases']
```

Out[7]:
```
1964    41525
1965    38586
1966    43366
1967    42225
1968    46922
1969    48720
1970    52183
1971    56181
1972    60328
1973    64182
1974    67692
1975    70551
1976    75069
1977    82601
1978    86222
1979    79858
1980    81476
1981    78951
1982    75717
1983    72585
1984    66073
1985    70237
1986    68805
1987    66656
1988    63446
1989    64027
1990    64971
1991    66409
1992    69856
1993    69784
1994    70337
1995    63679
1996    55710
1997    52141
1998    46602
1999    50622
2000    48226
2001    49502
2002    46643
2003    51389
2004    59995
```

```
2005    52741
2006    45910
2007    47517
2008    44931
2009    47497
2010    46407
2011    43954
2012    43940
2013    38272
2014    38282
2015    37215
Name: Cases, dtype: int64
```

6. Use `describe()` to view the summary statistics for the DataFrame.

In [8]:
```
df.describe()
```

Out[8]:

	Fiscal Year	Cases	Responses	Sorties	Lives Saved	Lives Lost After CG Notification	Lives Lost Before CG Notification	Total	Lives Unaccounted For
count	46.000000	52.000000	46.000000	52.000000	46.000000	37.000000	46.000000	16.000000	0.0
mean	46296.608696	58013.769231	67666.586957	4339.230769	670.956522	508.486486	1079.956522	468.000000	NaN
std	17438.646933	13480.714228	29300.537271	1334.134847	499.839128	134.761028	394.869765	149.916866	NaN
min	16456.000000	37215.000000	18781.000000	1984.000000	169.000000	180.000000	533.000000	252.000000	NaN
25%	31676.250000	46632.750000	33202.750000	3348.500000	281.750000	425.000000	751.000000	336.750000	NaN
50%	50621.500000	55945.500000	81711.500000	4221.000000	383.500000	492.000000	998.000000	437.500000	NaN
75%	57072.750000	69049.750000	88433.750000	5484.500000	1118.750000	593.000000	1440.750000	584.250000	NaN
max	77954.000000	86222.000000	110267.000000	7889.000000	1783.000000	800.000000	1821.000000	732.000000	NaN

7. You can filter for particular values by comparing a colum to a value within the square bracket syntax. This creates a mask on the fly. Lets look at all of the rows whose case count is higher than the mean. You can get this number from the summary statistics above.

In [18]:
```
df[df.Cases >  df.describe()['Cases']['mean']      ]
```

Out[18]:

	Fiscal Year	Cases	Responses	Sorties	Lives Saved	Lives Lost After CG Notification	Lives Lost Before CG Notification	Total	Lives Unaccounted For
1972	51539.0	60328	72306.0	2633	1389.0	NaN	1389.0	NaN	NaN
1973	55107.0	64182	77209.0	2918	1474.0	NaN	1474.0	NaN	NaN
1974	59335.0	67692	79950.0	2751	1509.0	NaN	1509.0	NaN	NaN
1975	62334.0	70551	81561.0	3024	1254.0	NaN	1254.0	NaN	NaN
1976	67179.0	75069	87807.0	2995	1112.0	NaN	1112.0	NaN	NaN
1977	74637.0	82601	96021.0	4121	1458.0	NaN	1458.0	NaN	NaN
1978	77954.0	86222	100262.0	4386	1556.0	NaN	1556.0	NaN	NaN
1979	72517.0	79858	92117.0	5747	949.0	672.0	1621.0	NaN	NaN
1980	73345.0	81476	93726.0	6868	1235.0	586.0	1821.0	NaN	NaN
1981	71781.0	78951	91432.0	6339	1080.0	637.0	1717.0	NaN	NaN
1982	68552.0	75717	87715.0	5675	1359.0	446.0	1805.0	NaN	NaN
1983	63980.0	72585	85796.0	5946	1121.0	640.0	1761.0	NaN	NaN
1984	57431.0	66073	80698.0	5645	1148.0	319.0	1467.0	NaN	NaN
1985	60775.0	70237	88449.0	6497	1076.0	259.0	1335.0	NaN	NaN
1986	51765.0	68805	89318.0	4307	475.0	180.0	655.0	NaN	NaN
1987	55998.0	66656	87211.0	5785	1015.0	576.0	1591.0	NaN	NaN
1988	54199.0	63446	83616.0	4307	583.0	449.0	1032.0	NaN	NaN
1989	52776.0	64027	81862.0	3981	461.0	646.0	1107.0	NaN	NaN
1990	53097.0	64971	84033.0	4407	463.0	622.0	1085.0	NaN	NaN
1991	52782.0	66409	84872.0	5465	368.0	748.0	1116.0	NaN	NaN
1992	53294.0	69856	88388.0	5543	399.0	540.0	939.0	NaN	NaN
1993	53026.0	69784	88147.0	5826	415.0	800.0	1215.0	NaN	NaN
1994	53899.0	70337	108758.0	7889	338.0	593.0	931.0	NaN	NaN
1995	49704.0	63679	110267.0	4453	304.0	468.0	772.0	NaN	NaN
2004	32418.0	59995	33460.0	5557	281.0	502.0	783.0	691.0	NaN

9. Now lets create a NumPy array with the same data. Pandas DataFrames have a `to_numpy()` method. Use this method to create an array named 'np_array'.

In [20]:
```
np_array = df.to_numpy()
np_array
```

Out[20]:
```
array([[    nan,  41525.,     nan,   2932.,     nan,     nan,     nan,
            nan,     nan],
       [    nan,  38586.,     nan,   1984.,     nan,     nan,     nan,
            nan,     nan],
       [    nan,  43366.,     nan,   2629.,     nan,     nan,     nan,
            nan,     nan],
       [    nan,  42225.,     nan,   3028.,     nan,     nan,     nan,
            nan,     nan],
       [    nan,  46922.,     nan,   2434.,     nan,     nan,     nan,
            nan,     nan],
       [    nan,  48720.,     nan,   2050.,     nan,     nan,     nan,
            nan,     nan],
       [ 44975.,  52183.,  62286.,   4135.,   1783.,     nan,   1783.,
            nan,     nan],
       [ 48894.,  56181.,  68251.,   2423.,   1324.,     nan,   1324.,
            nan,     nan],
       [ 51539.,  60328.,  72306.,   2633.,   1389.,     nan,   1389.,
            nan,     nan],
       [ 55107.,  64182.,  77209.,   2918.,   1474.,     nan,   1474.,
```

```
       nan,     nan],
       [ 59335.,  67692.,  79950.,   2751.,   1509.,     nan,   1509.,
            nan,     nan],
       [ 62334.,  70551.,  81561.,   3024.,   1254.,     nan,   1254.,
            nan,     nan],
       [ 67179.,  75069.,  87807.,   2995.,   1112.,     nan,   1112.,
            nan,     nan],
       [ 74637.,  82601.,  96021.,   4121.,   1458.,     nan,   1458.,
            nan,     nan],
       [ 77954.,  86222., 100262.,   4386.,   1556.,     nan,   1556.,
            nan,     nan],
       [ 72517.,  79858.,  92117.,   5747.,    949.,    672.,   1621.,
            nan,     nan],
       [ 73345.,  81476.,  93726.,   6868.,   1235.,    586.,   1821.,
            nan,     nan],
       [ 71781.,  78951.,  91432.,   6339.,   1080.,    637.,   1717.,
            nan,     nan],
       [ 68552.,  75717.,  87715.,   5675.,   1359.,    446.,   1805.,
            nan,     nan],
       [ 63980.,  72585.,  85796.,   5946.,   1121.,    640.,   1761.,
            nan,     nan],
       [ 57431.,  66073.,  80698.,   5645.,   1148.,    319.,   1467.,
            nan,     nan],
       [ 60775.,  70237.,  88449.,   6497.,   1076.,    259.,   1335.,
            nan,     nan],
       [ 51765.,  68805.,  89318.,   4307.,    475.,    180.,    655.,
            nan,     nan],
       [ 55998.,  66656.,  87211.,   5785.,   1015.,    576.,   1591.,
            nan,     nan],
       [ 54199.,  63446.,  83616.,   4307.,    583.,    449.,   1032.,
            nan,     nan],
       [ 52776.,  64027.,  81862.,   3981.,    461.,    646.,   1107.,
            nan,     nan],
       [ 53097.,  64971.,  84033.,   4407.,    463.,    622.,   1085.,
            nan,     nan],
       [ 52782.,  66409.,  84872.,   5465.,    368.,    748.,   1116.,
            nan,     nan],
       [ 53294.,  69856.,  88388.,   5543.,    399.,    540.,    939.,
            nan,     nan],
       [ 53026.,  69784.,  88147.,   5826.,    415.,    800.,   1215.,
            nan,     nan],
       [ 53899.,  70337., 108758.,   7889.,    338.,    593.,    931.,
            nan,     nan],
       [ 49704.,  63679., 110267.,   4453.,    304.,    468.,    772.,
            nan,     nan],
       [ 43553.,  55710.,  98423.,   5047.,    367.,    611.,    978.,
            nan,     nan],
       [ 41096.,  52141.,  91722.,   3897.,    290.,    454.,    744.,
            nan,     nan],
       [ 37218.,  46602.,  83307.,   3194.,    188.,    418.,    606.,
            nan,     nan],
       [ 39844.,  50622.,  89635.,   3743.,    180.,    353.,    533.,
            nan,     nan],
       [ 40214.,  48226.,  57697.,   3400.,    239.,    779.,   1018.,
             304.,     nan],
       [ 39457.,  49502.,  59015.,   4010.,    297.,    413.,    710.,
             515.,     nan],
       [ 36763.,  46643.,  54609.,   3661.,    236.,    399.,    635.,
             339.,     nan],
       [ 31429.,  51389.,  33117.,   5192.,    263.,    409.,    672.,
             496.,     nan],
       [ 32418.,  59995.,  33460.,   5557.,    281.,    502.,    783.,
             691.,     nan],
       [ 29646.,  52741.,  30779.,   5635.,    324.,    521.,    845.,
             603.,     nan],
       [ 28151.,  45910.,  28583.,   5275.,    328.,    452.,    780.,
             664.,     nan],
       [ 26927.,  47517.,  26586.,   5200.,    300.,    492.,    792.,
             732.,     nan],
       [ 24213.,  44931.,  25475.,   4900.,    291.,    534.,    825.,
             435.,     nan],
       [ 23545.,  47497.,  24644.,   4882.,    261.,    555.,    816.,
             578.,     nan],
       [ 22229.,  46407.,  23145.,   4362.,    263.,    552.,    815.,
             409.,     nan],
       [ 20512.,  43954.,  21566.,   3793.,    259.,    476.,    735.,
             392.,     nan],
       [ 19787.,  43940.,  21609.,   4037.,    284.,    429.,    713.,
             440.,     nan],
       [ 17803.,  38272.,  19420.,   3753.,    226.,    425.,    651.,
             252.,     nan],
       [ 17508.,  38282.,  19032.,   3443.,    170.,    425.,    595.,
             308.,     nan],
       [ 16456.,  37215.,  18781.,   3536.,    169.,    434.,    603.,
             330.,     nan]])
```

10. Call the shape attribute on the array.

In [21]:
```
np_array.shape
```

Out[21]:
```
(52, 9)
```

11. Use the array `reshape()` method to return a 4 x 13 x 9 array (the arguments to the method will be these numbers) .

In [22]:
```
np_array.reshape(4,13,9)
```

Out[22]:
```
array([[[    nan,  41525.,     nan,   2932.,     nan,     nan,     nan,
             nan,     nan],
        [    nan,  38586.,     nan,   1984.,     nan,     nan,     nan,
             nan,     nan],
        [    nan,  43366.,     nan,   2629.,     nan,     nan,     nan,
             nan,     nan],
        [    nan,  42225.,     nan,   3028.,     nan,     nan,     nan,
             nan,     nan],
        [    nan,  46922.,     nan,   2434.,     nan,     nan,     nan,
             nan,     nan],
        [    nan,  48720.,     nan,   2050.,     nan,     nan,     nan,
             nan,     nan],
        [ 44975.,  52183.,  62286.,   4135.,   1783.,     nan,   1783.,
```

```
       nan,     nan],
[ 48894.,  56181.,  68251.,   2423.,   1324.,     nan,   1324.,
       nan,     nan],
[ 51539.,  60328.,  72306.,   2633.,   1389.,     nan,   1389.,
       nan,     nan],
[ 55107.,  64182.,  77209.,   2918.,   1474.,     nan,   1474.,
```

12. Import the dask.dataframe module aliased as 'dd'

In [23]:
```
import dask.dataframe as dd
```

13. the `dask.dataframe` module has a `read_csv()` method which works in a similar fasion to the Pandas one. Use this method to read the file 'USCG.Search.Rescue.Stats.csv' into a dask DataFrame named 'ddf'

In [24]:
```
ddf = dd.read_csv ('USCG.Search.Rescue.Stats.csv')
```

14. Call the DataFrames `std()` method.

In [25]:
```
ddf.std()
```

Out[25]:
```
Dask Series Structure:
npartitions=1
Cases    float64
Total        ...
dtype: float64
Dask Name: dataframe-std, 9 tasks
```

15. Notice that this did not calculate the standard deviation due to dask's use of lazy evaluation. add a `.compute()` after the `std()` to compute the result.

In [26]:
```
ddf.std().compute()
```

Out[26]:
```
Fiscal Year                         17438.646933
Cases                               13480.714228
Responses                           29300.537271
Sorties                              1334.134847
Lives Saved                           499.839128
Lives Lost After CG Notification      134.761028
Lives Lost Before CG Notification     394.869765
Total                                 149.916866
Lives Unaccounted For                        NaN
dtype: float64
```