day_5_lecture

January 15, 2020

1 Day 5: Pandas

Probably the most important tool for a data scientist in python is pandas. Pandas is built on top of numpy, therefore there will be many similarites and mechanics you already know. Where numpy made heavy use of the `ndarray`, most magic happens in the pandas `DataFrame`. Like other data frames like in R, the pandas `DataFrame` stores data in a rectangular grid that can be easily overviewed. Numpy is mostly used for numerical data, while pandas can be used for any tabular data. Pandas also has many useful functions! (really, a lot).

pandas, numpy and matplotlib are the holy trio for data science with python.

Documentation can be found here: https://pandas.pydata.org/pandas-docs/stable/index.html

```
In [1]: import numpy as np
        import pandas as pd # common way to import pandas
```

We will start with `Series` we can build one column of a `DataFrame`. A `Series` can also be seen as one feature of a dataset. They can easily be created from a list and are similar to 1-dimensional numpy arrays.

1.1 Series and Index

```
In [2]: s = pd.Series([1, 3, 5, np.nan, 6, 8])
        s

Out[2]: 0    1.0
        1    3.0
        2    5.0
        3    NaN
        4    6.0
        5    8.0
        dtype: float64
```

`Series` can contain also strings or any other type of value.

```
In [3]: t = pd.Series(["red", "green", "blue", "yellow", "purple", "black"])
        t
```

```
Out[3]: 0       red
        1     green
        2      blue
        3    yellow
        4    purple
        5     black
        dtype: object
```

When we print out the `Series` we see that we get two columns of values. The right one is the one we speciefied, and the left one is the index. Default, the index is just the integer index. We can also give it another index.

```
In [4]: u = pd.Series(np.arange(5), index=list("ABCDE")) #create a series from numpy array and c
        u.index

Out[4]: Index(['A', 'B', 'C', 'D', 'E'], dtype='object')
```

We can access the elements with the new specfied index

```
In [5]: s[2], t[4], u["B"]

Out[5]: (5.0, 'purple', 1)
```

Next to creating your own index, pandas also offers multiple ways to create an `Index`. Some can be found here: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Index.html Some examples: - `DatetimeIndex` for dates - `TimedeltaIndex` for time steps - `CategeorialIndex` for defined categories

```
In [6]: dates = pd.date_range('20200101', periods=6) # index for 6 days starting with 2020-01-01
        dates

Out[6]: DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04',
                       '2020-01-05', '2020-01-06'],
                      dtype='datetime64[ns]', freq='D')

In [7]: times = pd.timedelta_range(start=0, periods=6, freq="3s")
        times

Out[7]: TimedeltaIndex(['00:00:00', '00:00:03', '00:00:06', '00:00:09', '00:00:12',
                        '00:00:15'],
                       dtype='timedelta64[ns]', freq='3S')

In [8]: times_6H = pd.timedelta_range(start=0, periods=6, freq="6H")
        times_6H

Out[8]: TimedeltaIndex(['0 days 00:00:00', '0 days 06:00:00', '0 days 12:00:00',
                        '0 days 18:00:00', '1 days 00:00:00', '1 days 06:00:00'],
                       dtype='timedelta64[ns]', freq='6H')

In [9]: c = pd.CategoricalIndex(['a', 'b', 'c', 'a', 'b', 'c'])
        c
```

```
Out[9]: CategoricalIndex(['a', 'b', 'c', 'a', 'b', 'c'], categories=['a', 'b', 'c'], ordered=Fal

In [10]: c_ord = pd.CategoricalIndex(['a', 'b', 'c', 'a', 'b', 'c'], ordered=True)
         c_ord

Out[10]: CategoricalIndex(['a', 'b', 'c', 'a', 'b', 'c'], categories=['a', 'b', 'c'], ordered=Tr

In [11]: c_ord.min(), c_ord.max() # if ordered, can have min max values

Out[11]: ('a', 'c')

In [12]: v = pd.Series(np.arange(6), index=c_ord)
         v["a"]

Out[12]: a    0
         a    3
         dtype: int64
```

To be honest, mostly the normal `RangeIndex` (default integer index) is used, and values such as time can be stored as a feature in another `Series` itself. But it is useful to know that we can use different indexes.

We can also use categories when creating series

```
In [113]: s = pd.Series(["a", "b", "c", "a"], dtype="category")
          s

Out[113]: 0    a
          1    b
          2    c
          3    a
          dtype: category
          Categories (3, object): [a, b, c]
```

1.2 DataFrames

Next is the key element `DataFrame`, which is similar two a 2-dimensional numpy array, storing data in a grid. There are multiple ways to create a `DataFrame`.

```
In [13]: df = pd.DataFrame() # empty dataframe
         df.dtypes

Out[13]: Series([], dtype: object)
```

Like we have seen, a `DataFrame` consists of one or more `Series`. We can create them by joining them together.

```
In [14]: s1 = pd.Series(np.random.rand(5))
         s2 = pd.Series(np.random.rand(5))
         print(s1)
```

```
0    0.498109
1    0.614942
2    0.315786
3    0.063064
4    0.991918
dtype: float64
```

```
In [15]: print(s2)
```

```
0    0.563196
1    0.898871
2    0.220011
3    0.330872
4    0.880307
dtype: float64
```

```
In [16]: df = pd.concat([s1,s2])
         print(df)
         print(type(df)) # actually still a series
```

```
0    0.498109
1    0.614942
2    0.315786
3    0.063064
4    0.991918
0    0.563196
1    0.898871
2    0.220011
3    0.330872
4    0.880307
dtype: float64
<class 'pandas.core.series.Series'>
```

```
In [17]: df = pd.concat([s1,s2], axis=1)
         df
```

Out[17]:

	0	1
0	0.498109	0.563196
1	0.614942	0.898871
2	0.315786	0.220011
3	0.063064	0.330872
4	0.991918	0.880307

```
In [ ]:
```

```
In [18]: df = pd.DataFrame(np.random.randn(6, 4)) # from numpy array
         df
```

```
Out[18]:           0         1         2         3
         0 -0.201963  0.212811  0.444819  0.818569
         1 -1.459040 -0.275027  0.296985  0.043006
         2 -0.161683  0.382867 -0.240930  1.993062
         3 -2.042379  0.959030 -1.084232 -1.140696
         4 -0.235343  0.092286 -0.606111 -1.326658
         5 -0.802292 -0.346570 -1.669577 -0.804868
```

When we print out a `DataFrame`, we see that now we have two indices, one for the rows and one for the columns. As with `Series`, we can specify those in the creation.

```
In [19]: df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list('ABCD')) #row index
         df

Out[19]:                    A         B         C         D
         2020-01-01  0.126044 -0.107740  1.451376  0.961694
         2020-01-02 -2.675785  2.056661  0.675453  1.053569
         2020-01-03 -0.352593  1.612962 -0.003553  0.232848
         2020-01-04 -0.429911 -1.152860  0.474587 -0.859570
         2020-01-05 -1.009845  0.517491  0.157385  1.159045
         2020-01-06  2.907520 -1.236553  1.613119  0.940631
```

Unlike numpy arrays, `DataFrame` can have multiple types. For each columns entry, we have one type.

```
In [20]: df.dtypes

Out[20]: A    float64
         B    float64
         C    float64
         D    float64
         dtype: object
```

We can also create `DataFrames` from python dictionaries.

```
In [21]: df2 = pd.DataFrame({'A': 1,
                             'B': pd.Timestamp('20130102'),
                             'C': pd.Series(1, index=list(range(4)), dtype='float32'),
                             'D': np.array([3] * 4, dtype='int32'),
                             'E': pd.Categorical(["test", "train", "test", "train"]),
                             'F': 'foo'})
         df2

Out[21]:    A          B    C  D      E    F
         0  1 2013-01-02  1.0  3   test  foo
         1  1 2013-01-02  1.0  3  train  foo
         2  1 2013-01-02  1.0  3   test  foo
         3  1 2013-01-02  1.0  3  train  foo

In [22]: df2.dtypes # each column or Series has a different type
```

```
Out[22]: A             int64
         B    datetime64[ns]
         C           float32
         D             int32
         E          category
         F            object
         dtype: object
```

We also see, that broadcasting is applied if a value is not a list. Otherwise all list for each `Series` must have the same length.

```
In [23]: df2 = pd.DataFrame({'A': 1,
                             'B': pd.Timestamp('20130102'),
                             'C': pd.Series(1, index=list(range(4)), dtype='float32'),
                             'D': np.array([3] * 2, dtype='int32'),
                             'E': pd.Categorical(["test", "train", "test", "train"]),
                             'F': 'foo'})
         ## will create error, because list are not the same length


        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-23-b6829310fc21> in <module>
          4                     'D': np.array([3] * 2, dtype='int32'),
          5                     'E': pd.Categorical(["test", "train", "test", "train"]),
    ----> 6                     'F': 'foo'})
          7 ## will create error, because list are not the same length


        ~/.local/lib/python3.5/site-packages/pandas/core/frame.py in __init__(self, data, index,
        390                                 dtype=dtype, copy=copy)
        391         elif isinstance(data, dict):
    --> 392             mgr = init_dict(data, index, columns, dtype=dtype)
        393         elif isinstance(data, ma.MaskedArray):
        394             import numpy.ma.mrecords as mrecords


        ~/.local/lib/python3.5/site-packages/pandas/core/internals/construction.py in init_dict(
        210         arrays = [data[k] for k in keys]
        211
    --> 212     return arrays_to_mgr(arrays, data_names, index, columns, dtype=dtype)
        213
        214


        ~/.local/lib/python3.5/site-packages/pandas/core/internals/construction.py in arrays_to_
```

```
     49     # figure out the index, if necessary
     50     if index is None:
---> 51         index = extract_index(arrays)
     52     else:
     53         index = ensure_index(index)


~/.local/lib/python3.5/site-packages/pandas/core/internals/construction.py in extract_in
315             lengths = list(set(raw_lengths))
316             if len(lengths) > 1:
--> 317                 raise ValueError('arrays must all be same length')
318
319             if have_dicts:


ValueError: arrays must all be same length
```

Once we have build a DataFrame, we can access its columns also over function call. (Built in function from IPython).

```
In [24]: df2.A

Out[24]: 0    1
         1    1
         2    1
         3    1
         Name: A, dtype: int64

In [25]: df2.E

Out[25]: 0     test
         1    train
         2     test
         3    train
         Name: E, dtype: category
         Categories (2, object): [test, train]
```

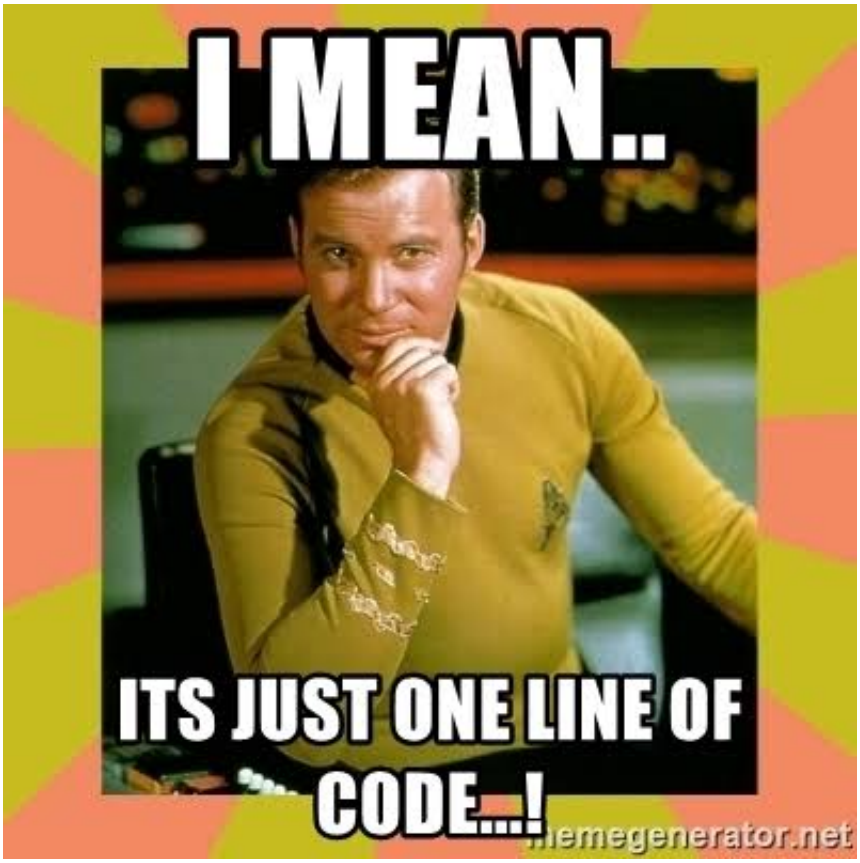
1.2.1 DataFrames from files

Since our data is usually stored in some file, pandas allow us to read many file types directly into a panda `DataFrame`. Very convienient! We also see, that pandas takes the headers as column index directly

```
In [32]: student_performance_df = pd.read_csv('NewStudentPerformance.csv')
         #student_performance_df
```

Just one line of code! wuhu!!

Pandas can also read and write to .xlsx (MS Excel) or .h5 (from the HDF group: https://www.hdfgroup.org/). You might need some extra software installed for that!


I MEAN..
ITS JUST ONE LINE OF CODE...!
memegenerator.net

```
In [27]: df_excel = pd.read_excel('excel_example.xlsx', 'Sheet1', index_col=None, na_values=['NA
         df_excel
```

Out[27]:

	Unnamed: 0	First Name	Last Name	Gender	Country	Age	\
0	1	Dulce	Abril	Female	United States	32	
1	2	Mara	Hashimoto	Female	Great Britain	25	
2	3	Philip	Gent	Male	France	36	
3	4	Kathleen	Hanner	Female	United States	25	
4	5	Nereida	Magwood	Female	United States	58	
5	6	Gaston	Brumm	Male	United States	24	
6	7	Etta	Hurn	Female	Great Britain	56	
7	8	Earlean	Melgar	Female	United States	27	
8	9	Vincenza	Weiland	Female	United States	40	
9	10	Fallon	Winward	Female	Great Britain	28	
10	11	Arcelia	Bouska	Female	Great Britain	39	
11	12	Franklyn	Unknow	Male	France	38	
12	13	Sherron	Ascencio	Female	Great Britain	32	
13	14	Marcel	Zabriskie	Male	Great Britain	26	
14	15	Kina	Hazelton	Female	Great Britain	31	
15	16	Shavonne	Pia	Female	France	24	
16	17	Shavon	Benito	Female	France	39	
17	18	Lauralee	Perrine	Female	Great Britain	28	
18	19	Loreta	Curren	Female	France	26	
19	20	Teresa	Strawn	Female	France	46	
20	21	Belinda	Partain	Female	United States	37	
21	22	Holly	Eudy	Female	United States	52	
22	23	Many	Cuccia	Female	Great Britain	46	
23	24	Libbie	Dalby	Female	France	42	
24	25	Lester	Prothro	Male	France	21	
25	26	Marvel	Hail	Female	Great Britain	28	
26	27	Angelyn	Vong	Female	United States	29	
27	28	Francesca	Beaudreau	Female	France	23	
28	29	Garth	Gangi	Male	United States	41	
29	30	Carla	Trumbull	Female	Great Britain	28	
..	...	...	...	...	...	...	
970	971	Belinda	Partain	Female	United States	37	
971	972	Holly	Eudy	Female	United States	52	
972	973	Many	Cuccia	Female	Great Britain	46	
973	974	Libbie	Dalby	Female	France	42	
974	975	Lester	Prothro	Male	France	21	
975	976	Marvel	Hail	Female	Great Britain	28	
976	977	Angelyn	Vong	Female	United States	29	
977	978	Francesca	Beaudreau	Female	France	23	
978	979	Garth	Gangi	Male	United States	41	
979	980	Carla	Trumbull	Female	Great Britain	28	
980	981	Veta	Muntz	Female	Great Britain	37	
981	982	Stasia	Becker	Female	Great Britain	34	
982	983	Jona	Grindle	Female	Great Britain	26	

983	984	Judie	Claywell	Female	France	35
984	985	Dewitt	Borger	Male	United States	36
985	986	Nena	Hacker	Female	United States	29
986	987	Kelsie	Wachtel	Female	France	27
987	988	Sau	Pfau	Female	United States	25
988	989	Shanice	Mccrystal	Female	United States	36
989	990	Chase	Karner	Male	United States	37
990	991	Tommie	Underdahl	Male	United States	26
991	992	Dorcas	Darity	Female	United States	37
992	993	Angel	Sanor	Male	France	24
993	994	Willodean	Harn	Female	United States	39
994	995	Weston	Martina	Male	United States	26
995	996	Roma	Lafollette	Female	United States	34
996	997	Felisa	Cail	Female	United States	28
997	998	Demetria	Abbey	Female	United States	32
998	999	Jeromy	Danz	Male	United States	39
999	1000	Rasheeda	Alkire	Female	United States	29

	Date	Id
0	15/10/2017	1562
1	16/08/2016	1582
2	21/05/2015	2587
3	15/10/2017	3549
4	16/08/2016	2468
5	21/05/2015	2554
6	15/10/2017	3598
7	16/08/2016	2456
8	21/05/2015	6548
9	16/08/2016	5486
10	21/05/2015	1258
11	15/10/2017	2579
12	16/08/2016	3256
13	21/05/2015	2587
14	16/08/2016	3259
15	21/05/2015	1546
16	15/10/2017	3579
17	16/08/2016	6597
18	21/05/2015	9654
19	21/05/2015	3569
20	15/10/2017	2564
21	16/08/2016	8561
22	21/05/2015	5489
23	21/05/2015	5489
24	15/10/2017	6574
25	16/08/2016	5555
26	21/05/2015	6125
27	15/10/2017	5412
28	16/08/2016	3256

```
29   21/05/2015  3264
..          ...   ...
970  15/10/2017  2564
971  16/08/2016  8561
972  21/05/2015  5489
973  21/05/2015  5489
974  15/10/2017  6574
975  16/08/2016  5555
976  21/05/2015  6125
977  15/10/2017  5412
978  16/08/2016  3256
979  21/05/2015  3264
980  15/10/2017  4569
981  16/08/2016  7521
982  21/05/2015  6458
983  16/08/2016  7569
984  21/05/2015  8514
985  15/10/2017  8563
986  16/08/2016  8642
987  21/05/2015  9536
988  21/05/2015  2567
989  15/10/2017  2154
990  16/08/2016  3265
991  21/05/2015  8765
992  15/10/2017  3259
993  16/08/2016  3567
994  21/05/2015  6540
995  15/10/2017  2654
996  16/08/2016  6525
997  21/05/2015  3265
998  15/10/2017  3265
999  16/08/2016  6125

[1000 rows x 8 columns]
```

In [31]:
```
#Store the former csv files into excel
df.to_excel('NewStudentPerformance.xlsx',sheet_name="Sheet1")
df
```

Out[31]:

	A	B	C	D
2020-01-01	0.126044	-0.107740	1.451376	0.961694
2020-01-02	-2.675785	2.056661	0.675453	1.053569
2020-01-03	-0.352593	1.612962	-0.003553	0.232848
2020-01-04	-0.429911	-1.152860	0.474587	-0.859570
2020-01-05	-1.009845	0.517491	0.157385	1.159045
2020-01-06	2.907520	-1.236553	1.613119	0.940631

In [29]:
```
df_hdf = pd.read_hdf("hdf_example.h5",'df')
df_hdf
```

Out[29]:

	Segment	Country	Product	Discount Band
0	Government	Canada	Carretera	None
1	Government	Germany	Carretera	None
2	Midmarket	France	Carretera	None
3	Midmarket	Germany	Carretera	None
4	Midmarket	Mexico	Carretera	None
5	Government	Germany	Carretera	None
6	Midmarket	Germany	Montana	None
7	Channel Partners	Canada	Montana	None
8	Government	France	Montana	None
9	Channel Partners	Germany	Montana	None
10	Midmarket	Mexico	Montana	None
11	Enterprise	Canada	Montana	None
12	Small Business	Mexico	Montana	None
13	Government	Germany	Montana	None
14	Enterprise	Canada	Montana	None
15	Midmarket	United States of America	Montana	None
16	Government	Canada	Paseo	None
17	Midmarket	Mexico	Paseo	None
18	Channel Partners	Canada	Paseo	None
19	Government	Germany	Paseo	None
20	Channel Partners	Germany	Paseo	None
21	Government	Mexico	Paseo	None
22	Midmarket	France	Paseo	None
23	Small Business	Mexico	Paseo	None
24	Midmarket	Mexico	Paseo	None
25	Government	United States of America	Paseo	None
26	Government	Canada	Paseo	None
27	Channel Partners	United States of America	Paseo	None
28	Midmarket	Canada	Paseo	None
29	Government	Canada	Paseo	None
..	...	...	...	...
670	Government	Germany	Paseo	High
671	Midmarket	Canada	Paseo	High
672	Government	Mexico	Paseo	High
673	Government	Mexico	Paseo	High
674	Midmarket	Canada	Paseo	High
675	Government	United States of America	Paseo	High
676	Enterprise	Germany	Paseo	High
677	Midmarket	Germany	Paseo	High
678	Government	United States of America	Paseo	High
679	Government	Mexico	Paseo	High
680	Channel Partners	United States of America	Paseo	High
681	Government	France	Paseo	High
682	Channel Partners	Mexico	Velo	High
683	Midmarket	France	Velo	High
684	Enterprise	France	Velo	High
685	Small Business	United States of America	Velo	High

686	Enterprise	United States of America	Velo	High	
687	Channel Partners	United States of America	Velo	High	
688	Government	Canada	VTT	High	
689	Midmarket	Germany	VTT	High	
690	Government	United States of America	VTT	High	
691	Midmarket	Germany	VTT	High	
692	Enterprise	Canada	VTT	High	
693	Enterprise	Germany	VTT	High	
694	Government	France	VTT	High	
695	Small Business	France	Amarilla	High	
696	Small Business	Mexico	Amarilla	High	
697	Government	Mexico	Montana	High	
698	Government	Canada	Paseo	High	
699	Channel Partners	United States of America	VTT	High	

	Units Sold	Manufacturing Price	Sale Price	Gross Sales	Discounts	\
0	1618.5	3	20	32370.0	0.00	
1	1321.0	3	20	26420.0	0.00	
2	2178.0	3	15	32670.0	0.00	
3	888.0	3	15	13320.0	0.00	
4	2470.0	3	15	37050.0	0.00	
5	1513.0	3	350	529550.0	0.00	
6	921.0	5	15	13815.0	0.00	
7	2518.0	5	12	30216.0	0.00	
8	1899.0	5	20	37980.0	0.00	
9	1545.0	5	12	18540.0	0.00	
10	2470.0	5	15	37050.0	0.00	
11	2665.5	5	125	333187.5	0.00	
12	958.0	5	300	287400.0	0.00	
13	2146.0	5	7	15022.0	0.00	
14	345.0	5	125	43125.0	0.00	
15	615.0	5	15	9225.0	0.00	
16	292.0	10	20	5840.0	0.00	
17	974.0	10	15	14610.0	0.00	
18	2518.0	10	12	30216.0	0.00	
19	1006.0	10	350	352100.0	0.00	
20	367.0	10	12	4404.0	0.00	
21	883.0	10	7	6181.0	0.00	
22	549.0	10	15	8235.0	0.00	
23	788.0	10	300	236400.0	0.00	
24	2472.0	10	15	37080.0	0.00	
25	1143.0	10	7	8001.0	0.00	
26	1725.0	10	350	603750.0	0.00	
27	912.0	10	12	10944.0	0.00	
28	2152.0	10	15	32280.0	0.00	
29	1817.0	10	20	36340.0	0.00	
..	...	...	...	...	...	
670	1158.0	10	20	23160.0	3474.00	

671	1614.0		10	15	24210.0	3631.50
672	2535.0		10	7	17745.0	2661.75
673	2851.0		10	350	997850.0	149677.50
674	2559.0		10	15	38385.0	5757.75
675	267.0		10	20	5340.0	801.00
676	1085.0		10	125	135625.0	20343.75
677	1175.0		10	15	17625.0	2643.75
678	2007.0		10	350	702450.0	105367.50
679	2151.0		10	350	752850.0	112927.50
680	914.0		10	12	10968.0	1645.20
681	293.0		10	20	5860.0	879.00
682	500.0		120	12	6000.0	900.00
683	2826.0		120	15	42390.0	6358.50
684	663.0		120	125	82875.0	12431.25
685	2574.0		120	300	772200.0	115830.00
686	2438.0		120	125	304750.0	45712.50
687	914.0		120	12	10968.0	1645.20
688	865.5		250	20	17310.0	2596.50
689	492.0		250	15	7380.0	1107.00
690	267.0		250	20	5340.0	801.00
691	1175.0		250	15	17625.0	2643.75
692	2954.0		250	125	369250.0	55387.50
693	552.0		250	125	69000.0	10350.00
694	293.0		250	20	5860.0	879.00
695	2475.0		260	300	742500.0	111375.00
696	546.0		260	300	163800.0	24570.00
697	1368.0		5	7	9576.0	1436.40
698	723.0		10	7	5061.0	759.15
699	1806.0		250	12	21672.0	3250.80

	Sales	COGS	Profit	Date	Month Number	Month Name	Year
0	32370.00	16185.0	16185.00	2014-01-01	1	January	2014
1	26420.00	13210.0	13210.00	2014-01-01	1	January	2014
2	32670.00	21780.0	10890.00	2014-06-01	6	June	2014
3	13320.00	8880.0	4440.00	2014-06-01	6	June	2014
4	37050.00	24700.0	12350.00	2014-06-01	6	June	2014
5	529550.00	393380.0	136170.00	2014-12-01	12	December	2014
6	13815.00	9210.0	4605.00	2014-03-01	3	March	2014
7	30216.00	7554.0	22662.00	2014-06-01	6	June	2014
8	37980.00	18990.0	18990.00	2014-06-01	6	June	2014
9	18540.00	4635.0	13905.00	2014-06-01	6	June	2014
10	37050.00	24700.0	12350.00	2014-06-01	6	June	2014
11	333187.50	319860.0	13327.50	2014-07-01	7	July	2014
12	287400.00	239500.0	47900.00	2014-08-01	8	August	2014
13	15022.00	10730.0	4292.00	2014-09-01	9	September	2014
14	43125.00	41400.0	1725.00	2013-10-01	10	October	2013
15	9225.00	6150.0	3075.00	2014-12-01	12	December	2014
16	5840.00	2920.0	2920.00	2014-02-01	2	February	2014

17	14610.00	9740.0	4870.00	2014-02-01	2	February	2014
18	30216.00	7554.0	22662.00	2014-06-01	6	June	2014
19	352100.00	261560.0	90540.00	2014-06-01	6	June	2014
20	4404.00	1101.0	3303.00	2014-07-01	7	July	2014
21	6181.00	4415.0	1766.00	2014-08-01	8	August	2014
22	8235.00	5490.0	2745.00	2013-09-01	9	September	2013
23	236400.00	197000.0	39400.00	2013-09-01	9	September	2013
24	37080.00	24720.0	12360.00	2014-09-01	9	September	2014
25	8001.00	5715.0	2286.00	2014-10-01	10	October	2014
26	603750.00	448500.0	155250.00	2013-11-01	11	November	2013
27	10944.00	2736.0	8208.00	2013-11-01	11	November	2013
28	32280.00	21520.0	10760.00	2013-12-01	12	December	2013
29	36340.00	18170.0	18170.00	2014-12-01	12	December	2014
..	...	...	...	...	...	...	...
670	19686.00	11580.0	8106.00	2014-03-01	3	March	2014
671	20578.50	16140.0	4438.50	2014-04-01	4	April	2014
672	15083.25	12675.0	2408.25	2014-04-01	4	April	2014
673	848172.50	741260.0	106912.50	2014-05-01	5	May	2014
674	32627.25	25590.0	7037.25	2014-08-01	8	August	2014
675	4539.00	2670.0	1869.00	2013-10-01	10	October	2013
676	115281.25	130200.0	-14918.75	2014-10-01	10	October	2014
677	14981.25	11750.0	3231.25	2014-10-01	10	October	2014
678	597082.50	521820.0	75262.50	2013-11-01	11	November	2013
679	639922.50	559260.0	80662.50	2013-11-01	11	November	2013
680	9322.80	2742.0	6580.80	2014-12-01	12	December	2014
681	4981.00	2930.0	2051.00	2014-12-01	12	December	2014
682	5100.00	1500.0	3600.00	2014-03-01	3	March	2014
683	36031.50	28260.0	7771.50	2014-05-01	5	May	2014
684	70443.75	79560.0	-9116.25	2014-09-01	9	September	2014
685	656370.00	643500.0	12870.00	2013-11-01	11	November	2013
686	259037.50	292560.0	-33522.50	2013-12-01	12	December	2013
687	9322.80	2742.0	6580.80	2014-12-01	12	December	2014
688	14713.50	8655.0	6058.50	2014-07-01	7	July	2014
689	6273.00	4920.0	1353.00	2014-07-01	7	July	2014
690	4539.00	2670.0	1869.00	2013-10-01	10	October	2013
691	14981.25	11750.0	3231.25	2014-10-01	10	October	2014
692	313862.50	354480.0	-40617.50	2013-11-01	11	November	2013
693	58650.00	66240.0	-7590.00	2014-11-01	11	November	2014
694	4981.00	2930.0	2051.00	2014-12-01	12	December	2014
695	631125.00	618750.0	12375.00	2014-03-01	3	March	2014
696	139230.00	136500.0	2730.00	2014-10-01	10	October	2014
697	8139.60	6840.0	1299.60	2014-02-01	2	February	2014
698	4301.85	3615.0	686.85	2014-04-01	4	April	2014
699	18421.20	5418.0	13003.20	2014-05-01	5	May	2014

[700 rows x 16 columns]

1.3 DataFrame Basic Functions

Some basic functions for viewing the data.

```
In [33]: df.head(2) # see 2 first items rows in df
```

```
Out[33]:                    A         B         C         D
         2020-01-01  0.126044 -0.107740  1.451376  0.961694
         2020-01-02 -2.675785  2.056661  0.675453  1.053569
```

```
In [34]: df.tail(2) # see last two rows in df
```

```
Out[34]:                    A         B         C         D
         2020-01-05 -1.009845  0.517491  0.157385  1.159045
         2020-01-06  2.907520 -1.236553  1.613119  0.940631
```

```
In [35]: df.columns, df.index
```

```
Out[35]: (Index(['A', 'B', 'C', 'D'], dtype='object'),
          DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04',
                         '2020-01-05', '2020-01-06'],
                        dtype='datetime64[ns]', freq='D'))
```

We can display some quick statistics with the function `describe()`.

```
In [36]: df.describe()
```

```
Out[36]:                A         B         C         D
         count  6.000000  6.000000  6.000000  6.000000
         mean  -0.239095  0.281660  0.728061  0.581369
         std    1.822927  1.377901  0.668500  0.778225
         min   -2.675785 -1.236553 -0.003553 -0.859570
         25%   -0.864862 -0.891580  0.236685  0.409793
         50%   -0.391252  0.204876  0.575020  0.951163
         75%    0.006385  1.339094  1.257396  1.030600
         max    2.907520  2.056661  1.613119  1.159045
```

`describe()` only works for numerical dtypes.

```
In [37]: df2.describe()
```

```
Out[37]:          A    C    D
         count  4.0  4.0  4.0
         mean   1.0  1.0  3.0
         std    0.0  0.0  0.0
         min    1.0  1.0  3.0
         25%    1.0  1.0  3.0
         50%    1.0  1.0  3.0
         75%    1.0  1.0  3.0
         max    1.0  1.0  3.0
```

We can also rearange and sort our DataFrames quickly.

```
In [38]: df.T # Transprosing, just like in numpy
```

```
Out[38]:    2020-01-01  2020-01-02  2020-01-03  2020-01-04  2020-01-05  2020-01-06
         A    0.126044   -2.675785   -0.352593   -0.429911   -1.009845    2.907520
         B   -0.107740    2.056661    1.612962   -1.152860    0.517491   -1.236553
         C    1.451376    0.675453   -0.003553    0.474587    0.157385    1.613119
         D    0.961694    1.053569    0.232848   -0.859570    1.159045    0.940631
```

```
In [39]: df.sort_index(axis=1, ascending=False) # Sorting by an axis:
```

```
Out[39]:                    D         C         B         A
         2020-01-01  0.961694  1.451376 -0.107740  0.126044
         2020-01-02  1.053569  0.675453  2.056661 -2.675785
         2020-01-03  0.232848 -0.003553  1.612962 -0.352593
         2020-01-04 -0.859570  0.474587 -1.152860 -0.429911
         2020-01-05  1.159045  0.157385  0.517491 -1.009845
         2020-01-06  0.940631  1.613119 -1.236553  2.907520
```

```
In [40]: df.sort_values(by='B') #sort by values in a column
```

```
Out[40]:                    A         B         C         D
         2020-01-06  2.907520 -1.236553  1.613119  0.940631
         2020-01-04 -0.429911 -1.152860  0.474587 -0.859570
         2020-01-01  0.126044 -0.107740  1.451376  0.961694
         2020-01-05 -1.009845  0.517491  0.157385  1.159045
         2020-01-03 -0.352593  1.612962 -0.003553  0.232848
         2020-01-02 -2.675785  2.056661  0.675453  1.053569
```

We can find unique elements in a Series with pd.unique()

```
In [58]: pd.unique(df_excel.Country)
```

```
Out[58]: array(['United States', 'Great Britain', 'France'], dtype=object)
```

1.3.1 Selection

We can use the known indexing methods from python and numpy, but pandas also offer optimized function to select data.

```
In [41]: df['A']
```

```
Out[41]: 2020-01-01    0.126044
         2020-01-02   -2.675785
         2020-01-03   -0.352593
         2020-01-04   -0.429911
         2020-01-05   -1.009845
         2020-01-06    2.907520
         Freq: D, Name: A, dtype: float64
```

```
In [42]: df.A

Out[42]: 2020-01-01    0.126044
         2020-01-02   -2.675785
         2020-01-03   -0.352593
         2020-01-04   -0.429911
         2020-01-05   -1.009845
         2020-01-06    2.907520
         Freq: D, Name: A, dtype: float64

In [43]: df[0:3] # slice rows

Out[43]:                    A         B         C         D
         2020-01-01  0.126044 -0.107740  1.451376  0.961694
         2020-01-02 -2.675785  2.056661  0.675453  1.053569
         2020-01-03 -0.352593  1.612962 -0.003553  0.232848

In [44]: df['20200101':'20200104'] # slice rows with custom index

Out[44]:                    A         B         C         D
         2020-01-01  0.126044 -0.107740  1.451376  0.961694
         2020-01-02 -2.675785  2.056661  0.675453  1.053569
         2020-01-03 -0.352593  1.612962 -0.003553  0.232848
         2020-01-04 -0.429911 -1.152860  0.474587 -0.859570
```

We can also use selection with the function `loc()`

```
In [46]: df_excel.columns

Out[46]: Index(['Unnamed: 0', 'First Name', 'Last Name', 'Gender', 'Country', 'Age',
                'Date', 'Id'],
               dtype='object')

In [47]: df_excel.index

Out[47]: RangeIndex(start=0, stop=1000, step=1)

In [49]: df_excel.loc[0:5]

Out[49]:    Unnamed: 0 First Name  Last Name  Gender        Country  Age        Date  \
         0           1      Dulce      Abril  Female  United States   32  15/10/2017
         1           2       Mara  Hashimoto  Female  Great Britain   25  16/08/2016
         2           3     Philip       Gent    Male         France   36  21/05/2015
         3           4   Kathleen     Hanner  Female  United States   25  15/10/2017
         4           5    Nereida    Magwood  Female  United States   58  16/08/2016
         5           6     Gaston      Brumm    Male  United States   24  21/05/2015

              Id
         0  1562
         1  1582
```

```
2  2587
3  3549
4  2468
5  2554
```

Unlike python or numpy slicing, both start and end are included!!

```
In [ ]: df_excel.loc[0:5,'First Name':'Last Name']
```

We can still use normal slicing by integers with `iloc`.

```
In [51]: df_excel.iloc[0:5,1:6]
```

```
Out[51]:   First Name  Last Name  Gender        Country  Age
         0      Dulce      Abril  Female  United States   32
         1       Mara  Hashimoto  Female  Great Britain   25
         2     Philip       Gent    Male         France   36
         3   Kathleen     Hanner  Female  United States   25
         4    Nereida    Magwood  Female  United States   58
```

1.3.2 Boolean Indexing

Just like numpy, we can use boolean indexing to select data.

```
In [61]: df_excel[(df_excel.Age == 38)][:10] # first 10 people with age 38
```

```
Out[61]:      Unnamed: 0 First Name Last Name Gender Country  Age        Date    Id
         11           12   Franklyn    Unknow   Male  France   38  15/10/2017  2579
         61           62   Franklyn    Unknow   Male  France   38  15/10/2017  2579
         111         112   Franklyn    Unknow   Male  France   38  15/10/2017  2579
         161         162   Franklyn    Unknow   Male  France   38  15/10/2017  2579
         211         212   Franklyn    Unknow   Male  France   38  15/10/2017  2579
         261         262   Franklyn    Unknow   Male  France   38  15/10/2017  2579
         311         312   Franklyn    Unknow   Male  France   38  15/10/2017  2579
         361         362   Franklyn    Unknow   Male  France   38  15/10/2017  2579
         411         412   Franklyn    Unknow   Male  France   38  15/10/2017  2579
         461         462   Franklyn    Unknow   Male  France   38  15/10/2017  2579
```

use `isin()` to filter stuff out

```
In [67]: df1 = df.copy()
         df1['E'] = list("TUVXYZ")
         df1

         df1[df1['E'].isin(['U', 'Z'])]
```

```
Out[67]:                    A         B         C         D  E
         2020-01-02 -2.675785  2.056661  0.675453  1.053569  U
         2020-01-06  2.907520 -1.236553  1.613119  0.940631  Z
```

1.3.3 Assigning Values

To set values, we can use the `at()` function to set values by label, or `iat()` when setting values by position.

```
In [68]: df.at[dates[0], 'A'] = 0
         df
```

```
Out[68]:                    A         B         C         D
         2020-01-01  0.000000 -0.107740  1.451376  0.961694
         2020-01-02 -2.675785  2.056661  0.675453  1.053569
         2020-01-03 -0.352593  1.612962 -0.003553  0.232848
         2020-01-04 -0.429911 -1.152860  0.474587 -0.859570
         2020-01-05 -1.009845  0.517491  0.157385  1.159045
         2020-01-06  2.907520 -1.236553  1.613119  0.940631
```

We can also use numpy arrays to set values

```
In [70]: df.loc[:,'D'] = np.array([5] * len(df))
         df
```

```
Out[70]:                    A         B         C  D
         2020-01-01  0.000000 -0.107740  1.451376  5
         2020-01-02 -2.675785  2.056661  0.675453  5
         2020-01-03 -0.352593  1.612962 -0.003553  5
         2020-01-04 -0.429911 -1.152860  0.474587  5
         2020-01-05 -1.009845  0.517491  0.157385  5
         2020-01-06  2.907520 -1.236553  1.613119  5
```

1.3.4 Missing data

Sometimes, your dataset is not fully filled, and values are missing. Panda has some functions for that.

```
In [75]: df1 = df.reindex(index=dates[0:4], columns=list(df.columns) + ['E']) #reindex datafram
         df1
```

```
Out[75]:                    A         B         C  D   E
         2020-01-01  0.000000 -0.107740  1.451376  5 NaN
         2020-01-02 -2.675785  2.056661  0.675453  5 NaN
         2020-01-03 -0.352593  1.612962 -0.003553  5 NaN
         2020-01-04 -0.429911 -1.152860  0.474587  5 NaN
```

```
In [76]: df1.loc[dates[0]:dates[1], 'E'] = 1 # set some values to 1 in E
```

```
In [77]: df1.dropna(how='any') # drop all rows with nan
```

```
Out[77]:                    A         B         C  D    E
         2020-01-01  0.000000 -0.107740  1.451376  5  1.0
         2020-01-02 -2.675785  2.056661  0.675453  5  1.0
```

```
In [82]: df1 = df.reindex(index=dates[0:4], columns=list(df.columns) + ['E']) #reindex datafram
         df1.loc[dates[0]:dates[1], 'E'] = 1,2 # set some values to 1 in E
         df1
```

```
Out[82]:                    A         B         C  D    E
         2020-01-01  0.000000 -0.107740  1.451376  5  1.0
         2020-01-02 -2.675785  2.056661  0.675453  5  2.0
         2020-01-03 -0.352593  1.612962 -0.003553  5  NaN
         2020-01-04 -0.429911 -1.152860  0.474587  5  NaN
```

```
In [83]: df1.fillna(df1.mean())
```

```
Out[83]:                    A         B         C  D    E
         2020-01-01  0.000000 -0.107740  1.451376  5  1.0
         2020-01-02 -2.675785  2.056661  0.675453  5  2.0
         2020-01-03 -0.352593  1.612962 -0.003553  5  1.5
         2020-01-04 -0.429911 -1.152860  0.474587  5  1.5
```

1.3.5 Statistics

Very similar to numpy, we can use mean(), max(), min() etc.

```
In [84]: df.mean()
```

```
Out[84]: A   -0.260102
         B    0.281660
         C    0.728061
         D    5.000000
         dtype: float64
```

```
In [85]: df.std()
```

```
Out[85]: A    1.818599
         B    1.377901
         C    0.668500
         D    0.000000
         dtype: float64
```

1.3.6 Grouping

With `groupby` we can quickly split and combine data on some criteria. We ca also apply a function the each group independently.

```
In [87]: df = pd.DataFrame({'A': ['x', 'x', 'y', 'x',
                                  'x', 'y', 'y', 'x'],
                            'B': ['one', 'one', 'two', 'three',
                                  'two', 'two', 'one', 'three'],
                            'C': np.random.randn(8),
                            'D': np.random.randn(8)})
         df
```

```
Out[87]:    A      B         C         D
         0  x    one -0.157515 -0.853246
         1  x    one -0.395700  1.002586
         2  y    two  0.333713  0.694014
         3  x  three -0.398128 -0.427000
         4  x    two  1.003186  1.348453
         5  y    two  0.506414  0.316184
         6  y    one -1.493659  2.122196
         7  x  three  0.024755  1.263005

In [88]: df.groupby('A').sum() # group them together and applying sumation on grous # B is dropp

Out[88]:           C         D
         A
         x  0.076599  2.333798
         y -0.653532  3.132394
```

Grouping by multiple columns forms a hierarchical index, and again we can apply the sum function.

```
In [89]: df.groupby(['A', 'B']).sum()

Out[89]:                 C         D
         A B
         x one   -0.553215  0.149340
           three -0.373373  0.836004
           two    1.003186  1.348453
         y one   -1.493659  2.122196
           two    0.840128  1.010198
```

1.4 Plotting with Pandas

Matplotlib and Panda go hand in hand, and makes plotting really easy.

```
In [90]: import matplotlib.pyplot as plt

In [105]: ts = pd.Series(np.random.randn(1000),index=pd.date_range('1/1/2000', periods=1000))
          ts = ts.cumsum()

          df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index, columns=['A', 'B', 'C', 'D
          df

Out[105]:                    A         B         C         D
          2000-01-01 -0.639264  0.425656 -0.836404  0.155747
          2000-01-02 -0.465900 -0.387340 -0.184180  0.463308
          2000-01-03 -0.755542 -2.153248 -0.079528 -1.437335
          2000-01-04 -1.161744  1.783604  1.118597  0.543278
          2000-01-05 -1.851973  0.371926  0.152015 -0.156182
          2000-01-06 -0.530612  1.262829  0.301695  1.980780
```

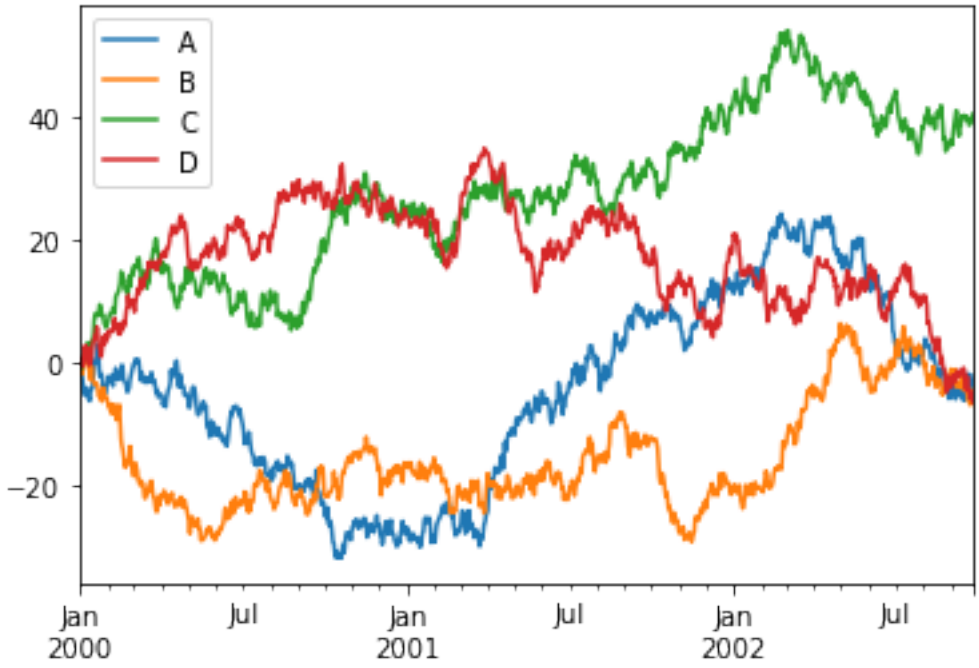
```
2000-01-07  1.202687 -0.863100  0.750397  1.051778
2000-01-08 -0.106650 -0.436423  0.361572 -1.572775
2000-01-09 -1.278419 -0.136594 -0.648674  1.333183
2000-01-10  0.063678 -0.820896  2.192116 -2.214769
2000-01-11 -0.666013 -0.027303 -0.645106 -0.710475
2000-01-12 -0.216189 -1.387731 -0.608319  0.016361
2000-01-13  2.069558 -0.585140  0.641569  0.287719
2000-01-14 -0.082015  2.195458  0.524207  0.829117
2000-01-15  0.032198 -2.108679  0.384916  2.007404
2000-01-16  1.742204 -0.385409  0.744326  0.794600
2000-01-17  1.959855 -0.332313  2.666998  0.295812
2000-01-18  0.895630 -0.123801  0.865512  0.672287
2000-01-19 -0.359173  0.167084  0.189377  0.385947
2000-01-20  3.009687 -0.054279 -0.115099  0.925764
2000-01-21 -1.741877 -0.665828 -0.353724 -1.134266
2000-01-22 -0.487541 -1.392878  0.497941 -2.011035
2000-01-23 -2.816089  0.087574  1.347424 -0.685257
2000-01-24 -0.287465  0.516768 -0.245602 -1.029799
2000-01-25  0.456266 -1.352964 -1.104378  0.067129
2000-01-26 -0.453316 -0.070614 -0.427572  1.336032
2000-01-27 -2.776330  2.191786  0.106109  1.371789
2000-01-28  1.032707 -0.479041 -1.348573  0.611911
2000-01-29 -1.057011 -0.956418 -0.157670 -1.257776
2000-01-30  0.736496 -0.087014 -0.568348  1.432001
...              ...       ...       ...       ...
2002-08-28 -0.740279  0.308551  0.519891  2.653240
2002-08-29  0.005273  0.460606  0.161770 -0.118340
2002-08-30 -0.832470  0.107505  0.332726 -0.947334
2002-08-31  0.313373  1.549692 -0.763913  0.337092
2002-09-01  0.404982  0.451437  0.814548 -1.187296
2002-09-02  1.279578 -0.695180 -0.655086  0.697402
2002-09-03  0.403510  0.027965  1.472849  1.389981
2002-09-04 -0.547555  0.115536  1.244077 -0.216948
2002-09-05 -0.828857  0.350606  1.052804 -0.889273
2002-09-06 -1.097685  0.151246  0.725545  0.076159
2002-09-07  2.581701 -2.731286  1.099541  0.583776
2002-09-08 -0.493511 -0.178400 -1.095296  0.988065
2002-09-09 -1.611279  0.343331 -2.111611  0.117517
2002-09-10  0.810478 -0.090628 -0.963900 -0.293053
2002-09-11 -0.025558  0.327968  1.567379 -0.613342
2002-09-12  0.951309 -0.248479  0.077689  0.554184
2002-09-13 -1.331006  1.624029  1.431111  0.106546
2002-09-14 -0.656639 -0.359900 -0.212400  0.465833
2002-09-15 -0.439610 -2.021497  0.521118  0.236919
2002-09-16  0.466854  1.403691 -0.343846 -0.328927
2002-09-17 -0.372737 -1.571209 -0.096702 -1.916425
2002-09-18  2.140338 -0.058420 -0.258618 -1.318598
2002-09-19  0.767253  0.538179  0.530686  2.165291
```

```
2002-09-20  1.387975 -1.893316 -0.450098 -1.962234
2002-09-21 -0.818661  0.000919 -0.928917 -0.621670
2002-09-22 -0.181008 -0.746455 -0.018580 -0.778426
2002-09-23 -3.711092 -0.117219  0.382648 -0.262681
2002-09-24  1.461237  0.843078  0.076132  1.185654
2002-09-25  1.233075  0.956962  0.909757 -1.122409
2002-09-26  1.725378  0.636183  0.444906 -0.648733

[1000 rows x 4 columns]
```

In [106]:
```
df = df.cumsum()
df.plot()
```

Out[106]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc3f4ce51d0>



In [107]:
```
df_excel
```

Out[107]:

	Unnamed: 0	First Name	Last Name	Gender	Country	Age	\
0	1	Dulce	Abril	Female	United States	32	
1	2	Mara	Hashimoto	Female	Great Britain	25	
2	3	Philip	Gent	Male	France	36	
3	4	Kathleen	Hanner	Female	United States	25	
4	5	Nereida	Magwood	Female	United States	58	
5	6	Gaston	Brumm	Male	United States	24	
6	7	Etta	Hurn	Female	Great Britain	56	

7	8	Earlean	Melgar	Female	United States	27
8	9	Vincenza	Weiland	Female	United States	40
9	10	Fallon	Winward	Female	Great Britain	28
10	11	Arcelia	Bouska	Female	Great Britain	39
11	12	Franklyn	Unknow	Male	France	38
12	13	Sherron	Ascencio	Female	Great Britain	32
13	14	Marcel	Zabriskie	Male	Great Britain	26
14	15	Kina	Hazelton	Female	Great Britain	31
15	16	Shavonne	Pia	Female	France	24
16	17	Shavon	Benito	Female	France	39
17	18	Lauralee	Perrine	Female	Great Britain	28
18	19	Loreta	Curren	Female	France	26
19	20	Teresa	Strawn	Female	France	46
20	21	Belinda	Partain	Female	United States	37
21	22	Holly	Eudy	Female	United States	52
22	23	Many	Cuccia	Female	Great Britain	46
23	24	Libbie	Dalby	Female	France	42
24	25	Lester	Prothro	Male	France	21
25	26	Marvel	Hail	Female	Great Britain	28
26	27	Angelyn	Vong	Female	United States	29
27	28	Francesca	Beaudreau	Female	France	23
28	29	Garth	Gangi	Male	United States	41
29	30	Carla	Trumbull	Female	Great Britain	28
..	...	...	...	...	...	...
970	971	Belinda	Partain	Female	United States	37
971	972	Holly	Eudy	Female	United States	52
972	973	Many	Cuccia	Female	Great Britain	46
973	974	Libbie	Dalby	Female	France	42
974	975	Lester	Prothro	Male	France	21
975	976	Marvel	Hail	Female	Great Britain	28
976	977	Angelyn	Vong	Female	United States	29
977	978	Francesca	Beaudreau	Female	France	23
978	979	Garth	Gangi	Male	United States	41
979	980	Carla	Trumbull	Female	Great Britain	28
980	981	Veta	Muntz	Female	Great Britain	37
981	982	Stasia	Becker	Female	Great Britain	34
982	983	Jona	Grindle	Female	Great Britain	26
983	984	Judie	Claywell	Female	France	35
984	985	Dewitt	Borger	Male	United States	36
985	986	Nena	Hacker	Female	United States	29
986	987	Kelsie	Wachtel	Female	France	27
987	988	Sau	Pfau	Female	United States	25
988	989	Shanice	Mccrystal	Female	United States	36
989	990	Chase	Karner	Male	United States	37
990	991	Tommie	Underdahl	Male	United States	26
991	992	Dorcas	Darity	Female	United States	37
992	993	Angel	Sanor	Male	France	24
993	994	Willodean	Harn	Female	United States	39

994	995	Weston	Martina	Male	United States	26
995	996	Roma	Lafollette	Female	United States	34
996	997	Felisa	Cail	Female	United States	28
997	998	Demetria	Abbey	Female	United States	32
998	999	Jeromy	Danz	Male	United States	39
999	1000	Rasheeda	Alkire	Female	United States	29

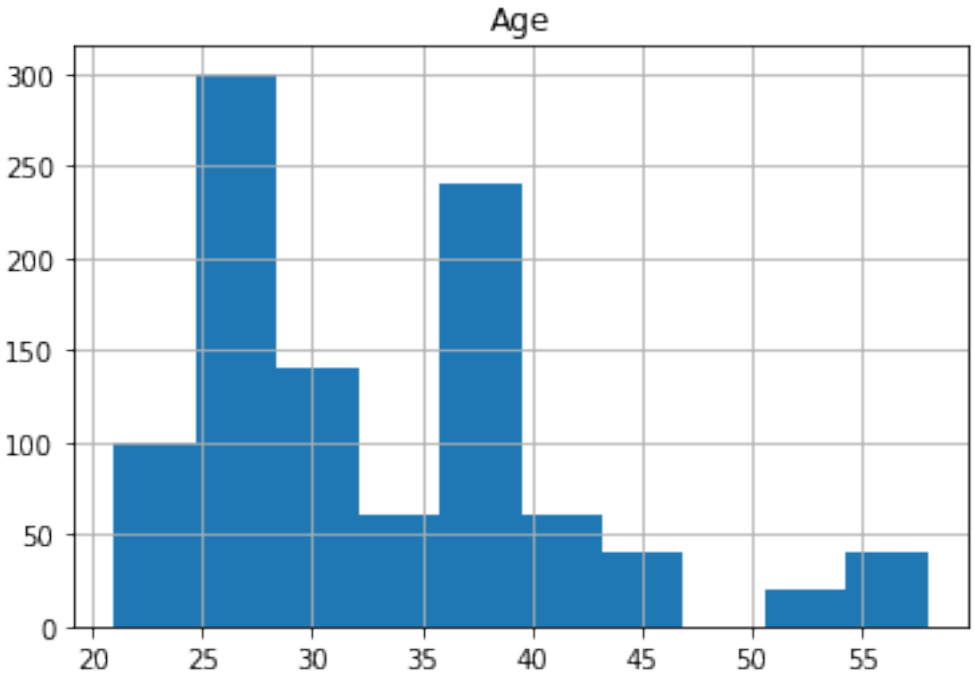
	Date	Id
0	15/10/2017	1562
1	16/08/2016	1582
2	21/05/2015	2587
3	15/10/2017	3549
4	16/08/2016	2468
5	21/05/2015	2554
6	15/10/2017	3598
7	16/08/2016	2456
8	21/05/2015	6548
9	16/08/2016	5486
10	21/05/2015	1258
11	15/10/2017	2579
12	16/08/2016	3256
13	21/05/2015	2587
14	16/08/2016	3259
15	21/05/2015	1546
16	15/10/2017	3579
17	16/08/2016	6597
18	21/05/2015	9654
19	21/05/2015	3569
20	15/10/2017	2564
21	16/08/2016	8561
22	21/05/2015	5489
23	21/05/2015	5489
24	15/10/2017	6574
25	16/08/2016	5555
26	21/05/2015	6125
27	15/10/2017	5412
28	16/08/2016	3256
29	21/05/2015	3264
..	...	...
970	15/10/2017	2564
971	16/08/2016	8561
972	21/05/2015	5489
973	21/05/2015	5489
974	15/10/2017	6574
975	16/08/2016	5555
976	21/05/2015	6125
977	15/10/2017	5412
978	16/08/2016	3256

```
979  21/05/2015  3264
980  15/10/2017  4569
981  16/08/2016  7521
982  21/05/2015  6458
983  16/08/2016  7569
984  21/05/2015  8514
985  15/10/2017  8563
986  16/08/2016  8642
987  21/05/2015  9536
988  21/05/2015  2567
989  15/10/2017  2154
990  16/08/2016  3265
991  21/05/2015  8765
992  15/10/2017  3259
993  16/08/2016  3567
994  21/05/2015  6540
995  15/10/2017  2654
996  16/08/2016  6525
997  21/05/2015  3265
998  15/10/2017  3265
999  16/08/2016  6125

[1000 rows x 8 columns]
```

```
In [111]: df_excel.hist(column="Age")
```

```
Out[111]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fc3f4bab668>]],
          dtype=object)
```



```
In [112]: np.NaN
```

```
Out[112]: nan
```