# PANDAS

In [ ]:
```
import numpy as np

import pandas as pd
```

Series: a one-dimensional labeled array holding data of any type such as integers, strings, Python objects etc. DataFrame: a two-dimensional data structure that holds data like a two-dimension array or a table with rows and columns.

In [ ]:
```
a=pd.Series([1,8,90,5.3,"hello"])
a
```

Out[ ]:
```
0        1
1        8
2       90
3      5.3
4    hello
dtype: object
```

In [ ]:
```
de=pd.DataFrame(
{
    "A":1.6,
    "B":pd.Timestamp("2004-08-15"),
    "C":pd.Series(1,index=list(range(4)),dtype="float32"),
    "D":np.array([3]*4,dtype="int32"),
    "E":pd.Categorical(["A","B","C","D"]),
    "F":"hi",
})

de
```

Out[ ]:

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| **0** | 1.6 | 2004-08-15 | 1.0 | 3 | A | hi |
| **1** | 1.6 | 2004-08-15 | 1.0 | 3 | B | hi |
| **2** | 1.6 | 2004-08-15 | 1.0 | 3 | C | hi |
| **3** | 1.6 | 2004-08-15 | 1.0 | 3 | D | hi |

In [ ]:
```
de.dtypes
```

Out[ ]:
```
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

# Viewing data

In [ ]:
```
de.head()
```

Out[ ]:

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| **0** | 1.6 | 2004-08-15 | 1.0 | 3 | A | hi |
| **1** | 1.6 | 2004-08-15 | 1.0 | 3 | B | hi |
| **2** | 1.6 | 2004-08-15 | 1.0 | 3 | C | hi |
| **3** | 1.6 | 2004-08-15 | 1.0 | 3 | D | hi |

In [ ]:
```
de.head(2)   #first 2 row
```

Out[ ]:

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| **0** | 1.6 | 2004-08-15 | 1.0 | 3 | A | hi |
| **1** | 1.6 | 2004-08-15 | 1.0 | 3 | B | hi |

In [ ]:
```
de.tail()  #form last
```

Out[ ]:

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| **0** | 1.6 | 2004-08-15 | 1.0 | 3 | A | hi |
| **1** | 1.6 | 2004-08-15 | 1.0 | 3 | B | hi |
| **2** | 1.6 | 2004-08-15 | 1.0 | 3 | C | hi |
| **3** | 1.6 | 2004-08-15 | 1.0 | 3 | D | hi |

In [ ]:
```
de.index
```

Out[ ]:
```
Int64Index([0, 1, 2, 3], dtype='int64')
```

In [ ]:
```
de.columns
```

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

In [ ]:
```
# without index and columns
de.to_numpy()
```

Out[ ]:
```
array([[1.6, Timestamp('2004-08-15 00:00:00'), 1.0, 3, 'A', 'hi'],
       [1.6, Timestamp('2004-08-15 00:00:00'), 1.0, 3, 'B', 'hi'],
       [1.6, Timestamp('2004-08-15 00:00:00'), 1.0, 3, 'C', 'hi'],
       [1.6, Timestamp('2004-08-15 00:00:00'), 1.0, 3, 'D', 'hi']],
      dtype=object)
```

NumPy arrays have one dtype for the entire array while pandas DataFrames have one dtype per column. When you call de.to_numpy(), pandas will find the NumPy dtype that can hold all of the dtypes in the DataFrame. If the common data type is object, de.to_numpy() will require copying data.

In [ ]:
```
# quick statistic summary of your data
de.describe()
```

Out[ ]:

| | A | C | D |
|---|---|---|---|
| **count** | 4.0 | 4.0 | 4.0 |
| **mean** | 1.6 | 1.0 | 3.0 |
| **std** | 0.0 | 0.0 | 0.0 |
| **min** | 1.6 | 1.0 | 3.0 |
| **25%** | 1.6 | 1.0 | 3.0 |
| **50%** | 1.6 | 1.0 | 3.0 |
| **75%** | 1.6 | 1.0 | 3.0 |
| **max** | 1.6 | 1.0 | 3.0 |

In [ ]:
```
#transposing
de.T
```

Out[ ]:

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| **A** | 1.6 | 1.6 | 1.6 | 1.6 |
| **B** | 2004-08-15 00:00:00 | 2004-08-15 00:00:00 | 2004-08-15 00:00:00 | 2004-08-15 00:00:00 |
| **C** | 1.0 | 1.0 | 1.0 | 1.0 |
| **D** | 3 | 3 | 3 | 3 |
| **E** | A | B | C | D |
| **F** | hi | hi | hi | hi |

# Sorting

In [ ]:
```
# sort_index() and sort_values()
```

In [ ]:
```
de.sort_index(axis=1,ascending=False)
#column index (axis=1)
```

Out[ ]:

| | F | E | D | C | B | A |
|---|---|---|---|---|---|---|
| **0** | hi | A | 3 | 1.0 | 2004-08-15 | 1.6 |
| **1** | hi | B | 3 | 1.0 | 2004-08-15 | 1.6 |
| **2** | hi | C | 3 | 1.0 | 2004-08-15 | 1.6 |
| **3** | hi | D | 3 | 1.0 | 2004-08-15 | 1.6 |

In [ ]:
```
de.sort_index(axis=0,ascending=False)
#row based sorting
```

Out[ ]:

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| **3** | 1.6 | 2004-08-15 | 1.0 | 3 | D | hi |
| **2** | 1.6 | 2004-08-15 | 1.0 | 3 | C | hi |
| **1** | 1.6 | 2004-08-15 | 1.0 | 3 | B | hi |
| **0** | 1.6 | 2004-08-15 | 1.0 | 3 | A | hi |

In [ ]:
```
de.sort_values(by="E",ascending=True)
```

Out[ ]:

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| **0** | 1.6 | 2004-08-15 | 1.0 | 3 | A | hi |
| **1** | 1.6 | 2004-08-15 | 1.0 | 3 | B | hi |
| **2** | 1.6 | 2004-08-15 | 1.0 | 3 | C | hi |
| **3** | 1.6 | 2004-08-15 | 1.0 | 3 | D | hi |

Column => axis=1 row=> axis=0 column&row=>label

# Selection

.at .iat .loc .ilocDataFrame.at Access a single value for a row/column pair by label. DataFrame.iat Access a single value for a row/column pair by integer position. DataFrame.loc Access a group of rows and columns by label(s). DataFrame.iloc Access a group of rows and columns by integer position(s).

In [ ]:
```
de["A"]
```

Out[ ]:
```
0    1.6
1    1.6
2    1.6
3    1.6
Name: A, dtype: float64
```

In [ ]:
```
de[1:3]
```

Out[ ]:

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| **1** | 1.6 | 2004-08-15 | 1.0 | 3 | B | hi |
| **2** | 1.6 | 2004-08-15 | 1.0 | 3 | C | hi |

Selection by label .at() & .loc()

In [ ]:
```
de.loc[1]
```

Out[ ]:
```
A                   1.6
B    2004-08-15 00:00:00
C                   1.0
D                     3
E                     B
F                    hi
Name: 1, dtype: object
```

In [ ]:
```
#all rows (:) with a select column
```

```
de.loc[:,["A"]]
```

Out[ ]:

| | A |
|---|---|
| **0** | 1.6 |
| **1** | 1.6 |
| **2** | 1.6 |
| **3** | 1.6 |

In [ ]:
```
de.loc[:,:] #all rows and columns
```

Out[ ]:

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| **0** | 1.6 | 2004-08-15 | 1.0 | 3 | A | hi |
| **1** | 1.6 | 2004-08-15 | 1.0 | 3 | B | hi |
| **2** | 1.6 | 2004-08-15 | 1.0 | 3 | C | hi |
| **3** | 1.6 | 2004-08-15 | 1.0 | 3 | D | hi |

In [ ]:
```
print(de.loc[3,"C"])#Selecting a single row and column label returns a scala
print(de.at[3,"C"])#For getting fast access to a scalar (equivalent to the prior
```

```
1.0
1.0
```

# Selection by position

.iloc() & iat()

In [ ]:
```
de.iloc[3]
```

Out[ ]:
```
A                    1.6
B    2004-08-15 00:00:00
C                    1.0
D                      3
E                      D
F                     hi
Name: 3, dtype: object
```

In [ ]:
```
de.iloc[2:5, 0:2]
```

Out[ ]:

| | A | B |
|---|---|---|
| **2** | 1.6 | 2004-08-15 |
| **3** | 1.6 | 2004-08-15 |

In [ ]:
```
de.iloc[[1, 2, 0], [0, 2]]
```

Out[ ]:

| | A | C |
|---|---|---|
| **1** | 1.6 | 1.0 |
| **2** | 1.6 | 1.0 |
| **0** | 1.6 | 1.0 |

In [ ]:
```
print(de.iloc[:,0:2])#all row
print(de.iloc[0:2,:])#all column
```

```
     A          B
0  1.6 2004-08-15
1  1.6 2004-08-15
2  1.6 2004-08-15
3  1.6 2004-08-15
     A          B    C  D  E   F
0  1.6 2004-08-15  1.0  3  A  hi
1  1.6 2004-08-15  1.0  3  B  hi
```

In [ ]:
```
print(de.iloc[1,1])#value explicitly:
print(de.iat[1,1])#fast access to scalar
```

```
2004-08-15 00:00:00
2004-08-15 00:00:00
```

# PANDAS PART 2

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

In [ ]:
```
df=pd.DataFrame({
    "A":5.6,
    "B":pd.date_range("2004-07-18",periods=5),
    "C":pd.Series([1,2,8,4,5],index=list("12345")),
    "D":pd.Categorical(["test","train","train","testing","training"])

},index=list("12345"),columns=list("ABCD"))
df
```

Out[ ]:

| | A | B | C | D |
|---|---|---|---|---|
| **1** | 5.6 | 2004-07-18 | 1 | test |
| **2** | 5.6 | 2004-07-19 | 2 | train |
| **3** | 5.6 | 2004-07-20 | 8 | train |
| **4** | 5.6 | 2004-07-21 | 4 | testing |
| **5** | 5.6 | 2004-07-22 | 5 | training |

# Boolean indexing

column=> axis=1,column=list("") row=> axis=0,index= axis{0 or 'index', 1 or 'columns'}, default 0

In [ ]:
```
print(df["C"]>4)
print(df[df["C"]>4])
```

```
1    False
2    False
3     True
4    False
5     True
Name: C, dtype: bool
     A          B  C         D
3  5.6 2004-07-20  8     train
5  5.6 2004-07-22  5  training
```

In [ ]:
```
#.copy()
df1=df.copy()
df1["E"]=["one", "two", "three", "four","five"]
#new column is created
print(df1)
```

```
     A          B  C         D      E
1  5.6 2004-07-18  1      test    one
2  5.6 2004-07-19  2     train    two
3  5.6 2004-07-20  8     train  three
4  5.6 2004-07-21  4   testing   four
5  5.6 2004-07-22  5  training   five
```

In [ ]:
```
#isin()
df1[df1["E"].isin(["one"])]
```

Out[ ]:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **1** | 5.6 | 2004-07-18 | 1 | test | one |

# Setting

new column automatically aligns the data by the index

In [ ]:
```
s1 = pd.Series([1, 24, 32, 46,75], index=list("12345"))

df["E"]=s1
df
```

Out[ ]:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **1** | 5.6 | 2004-07-18 | 1 | test | 1 |
| **2** | 5.6 | 2004-07-19 | 2 | train | 24 |
| **3** | 5.6 | 2004-07-20 | 8 | train | 32 |
| **4** | 5.6 | 2004-07-21 | 4 | testing | 46 |
| **5** | 5.6 | 2004-07-22 | 5 | training | 75 |

In [ ]:
```
# Setting values by label
df.at[1,"A"]=5.5
df.at[10,"A"]=5
```

```
df.at[2,"A"]=5.8
df
```

Out[ ]:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **1** | 5.6 | 2004-07-18 | 1.0 | test | 1.0 |
| **2** | 5.6 | 2004-07-19 | 2.0 | train | 24.0 |
| **3** | 5.6 | 2004-07-20 | 8.0 | train | 32.0 |
| **4** | 5.6 | 2004-07-21 | 4.0 | testing | 46.0 |
| **5** | 5.6 | 2004-07-22 | 5.0 | training | 75.0 |
| **1** | 5.5 | NaT | NaN | NaN | NaN |
| **10** | 5.0 | NaT | NaN | NaN | NaN |
| **2** | 5.8 | NaT | NaN | NaN | NaN |

In [ ]:
```
# Setting values by position
df.iat[3,1]=8.4
df.iat[3,4]=8.4
df
```

Out[ ]:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **1** | 5.6 | 2004-07-18 00:00:00 | 1.0 | test | 1.0 |
| **2** | 5.6 | 2004-07-19 00:00:00 | 2.0 | train | 24.0 |
| **3** | 5.6 | 2004-07-20 00:00:00 | 8.0 | train | 32.0 |
| **4** | 5.6 | 8.4 | 4.0 | testing | 8.4 |
| **5** | 5.6 | 2004-07-22 00:00:00 | 5.0 | training | 75.0 |
| **1** | 5.5 | NaT | NaN | NaN | NaN |
| **10** | 5.0 | NaT | NaN | NaN | NaN |
| **2** | 5.8 | NaT | NaN | NaN | NaN |

In [ ]:
```
df.loc[:,"E"]=np.array([5]*len(df))
df
```

```
C:\Users\heman\AppData\Local\Temp\ipykernel_25372\2153832009.py:1: FutureWarning:
In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inp
lace instead of always setting a new array. To retain the old behavior, use eithe
r `df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, ne
wvals)`
  df.loc[:,"E"]=np.array([5]*len(df))
```

Out[ ]:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | 5.6 | 2004-07-18 00:00:00 | 1.0 | test | 5 |
| 2 | 5.6 | 2004-07-19 00:00:00 | 2.0 | train | 5 |
| 3 | 5.6 | 2004-07-20 00:00:00 | 8.0 | train | 5 |
| 4 | 5.6 | 8.4 | 4.0 | testing | 5 |
| 5 | 5.6 | 2004-07-22 00:00:00 | 5.0 | training | 5 |
| 1 | 5.5 | NaT | NaN | NaN | 5 |
| 10 | 5.0 | NaT | NaN | NaN | 5 |
| 2 | 5.8 | NaT | NaN | NaN | 5 |

# Missing data

In [ ]:
```
#fillna() =>fill null with mean...
#dropna() =>remove null
#isna() =>check is null
```

In [ ]:
```
pd.isna(df)
```

Out[ ]:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | False | False | False | False | False |
| 2 | False | False | False | False | False |
| 3 | False | False | False | False | False |
| 4 | False | False | False | False | False |
| 5 | False | False | False | False | False |
| 1 | False | True | True | True | False |
| 10 | False | True | True | True | False |
| 2 | False | True | True | True | False |

In [ ]:
```
df.sum().isna()
```

```
C:\Users\heman\AppData\Local\Temp\ipykernel_25372\1949428304.py:1: FutureWarning:
The default value of numeric_only in DataFrame.sum is deprecated. In a future ver
sion, it will default to False. In addition, specifying 'numeric_only=None' is de
precated. Select only valid columns or specify the value of numeric_only to silen
ce this warning.
  df.sum().isna()
```

Out[ ]:
```
A    False
C    False
E    False
dtype: bool
```

In [ ]:
```
df["C"].dropna()
```

Out[ ]:
```
1    1.0
2    2.0
3    8.0
4    4.0
5    5.0
Name: C, dtype: float64
```

In [ ]:
```
df["C"].fillna(0)
df
```

Out[ ]:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **1** | 5.6 | 2004-07-18 00:00:00 | 1.0 | test | 5 |
| **2** | 5.6 | 2004-07-19 00:00:00 | 2.0 | train | 5 |
| **3** | 5.6 | 2004-07-20 00:00:00 | 8.0 | train | 5 |
| **4** | 5.6 | 8.4 | 4.0 | testing | 5 |
| **5** | 5.6 | 2004-07-22 00:00:00 | 5.0 | training | 5 |
| **1** | 5.5 | NaT | NaN | NaN | 5 |
| **10** | 5.0 | NaT | NaN | NaN | 5 |
| **2** | 5.8 | NaT | NaN | NaN | 5 |

inplacebool, default False If True, fill in-place. Note: this will modify any other views on this object (e.g., a no-copy slice for a column in a DataFrame)

# Operations

Stats

In [ ]:
```
df["A"].mean()
```

Out[ ]:
```
5.5375
```

In [ ]:
```
# df.mean(axis=1)
```

In [ ]:
```
df["C"].median()
```

Out[ ]:
```
4.0
```

In [ ]:
```
#shift =>move element in series
print(pd.Series([1, 3, 5, np.nan, 6, 8]))
print(pd.Series([1, 3, 5, np.nan, 6, 8]).shift(2))
s=pd.Series([1, 3, 5, np.nan, 6, 8]).shift(2)
```

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

In [ ]:
```
# sub => subract (row or column)
```

In [ ]:
```
df["E"].sub(s, axis="index")
```

Out[ ]:
```
0     NaN
1     NaN
2     4.0
3     NaN
4     NaN
5     NaN
10    NaN
1     NaN
2     NaN
3     NaN
4     NaN
5     NaN
dtype: float64
```

In [ ]:
```
#agg is an alias for aggregate
```

In [ ]:
```
df.agg('sum',axis="columns")
```

```
C:\Users\heman\AppData\Local\Temp\ipykernel_25372\1423552456.py:1: FutureWarning:
The default value of numeric_only in DataFrame.sum is deprecated. In a future ver
sion, it will default to False. In addition, specifying 'numeric_only=None' is de
precated. Select only valid columns or specify the value of numeric_only to silen
ce this warning.
  df.agg('sum',axis="columns")
```

Out[ ]:
```
1     10.5
10    10.0
2     10.8
dtype: object
```

In [ ]:
```
df["C"].agg(['sum', 'min','max'])
```

Out[ ]:
```
sum    20.0
min     1.0
max     8.0
Name: C, dtype: float64
```

In [ ]:
```
df.agg({'A' : ['sum', 'min'], 'C' : ['min', 'max']})
```

Out[ ]:

| | A | C |
|---|---|---|
| **sum** | 44.3 | NaN |
| **min** | 5.0 | 1.0 |
| **max** | NaN | 8.0 |

In [ ]:
```
# Applies a function to each value (element) in a DataFrame.
# Often used for group-wise transformations, but can also be applied to the whol
```

In [ ]:
```
df["C"].transform(lambda x: x + 1)
```

Out[ ]:
```
1     2.0
2     3.0
3     9.0
4     5.0
5     6.0
1     NaN
10    NaN
2     NaN
Name: C, dtype: float64
```

In [ ]:
```
s1=pd.Series(np.random.randint(0,8,size=12))
print(s1)
s1.value_counts()
# count number of occurrence
```

```
0     7
1     6
2     0
3     3
4     2
5     2
6     7
7     4
8     2
9     5
10    1
11    4
dtype: int32
```

Out[ ]:
```
2    3
7    2
4    2
6    1
0    1
3    1
5    1
1    1
dtype: int64
```

In [ ]:
```
df.groupby("D").sum()
df.groupby("D")[["A","C"]].sum()
```

```
C:\Users\heman\AppData\Local\Temp\ipykernel_25372\1267160968.py:1: FutureWarning:
The default value of numeric_only in DataFrameGroupBy.sum is deprecated. In a fut
ure version, numeric_only will default to False. Either specify numeric_only or s
elect only columns which should be valid for the function.
  df.groupby("D").sum()
```

Out[ ]:

| D | A | C |
|---|---|---|
| **test** | 5.6 | 1.0 |
| **testing** | 5.6 | 4.0 |
| **train** | 11.2 | 10.0 |
| **training** | 5.6 | 5.0 |

pivot_table() pivots a DataFrame specifying the values, index and columns

In [ ]:
```
pd.pivot_table(df,values="D",index=['C'])
```

```
C:\Users\heman\AppData\Local\Temp\ipykernel_25372\3834935483.py:1: FutureWarning:
The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a fu
ture version, numeric_only will default to False. Either specify numeric_only or
select only columns which should be valid for the function.
  pd.pivot_table(df,values="D",index=['C'])
```

Out[ ]:

| C |
|---|
| **1.0** |
| **2.0** |
| **4.0** |
| **5.0** |
| **8.0** |

In [ ]:
```
df3=pd.DataFrame({
    "id":[1,2,4,5,6,7],"raw":["A","E","W","E","A","D"]
})
df3
```

Out[ ]:

| | id | raw |
|---|---|---|
| **0** | 1 | A |
| **1** | 2 | E |
| **2** | 4 | W |
| **3** | 5 | E |
| **4** | 6 | A |
| **5** | 7 | D |

In [ ]:
```
df3["grade"]=df3["raw"].astype("category")
print(df3)
df3.dtypes
```

```
   id raw grade
0   1   A     A
1   2   E     E
2   4   W     W
3   5   E     E
4   6   A     A
5   7   D     D
```

Out[ ]:
```
id           int64
raw         object
grade     category
dtype: object
```

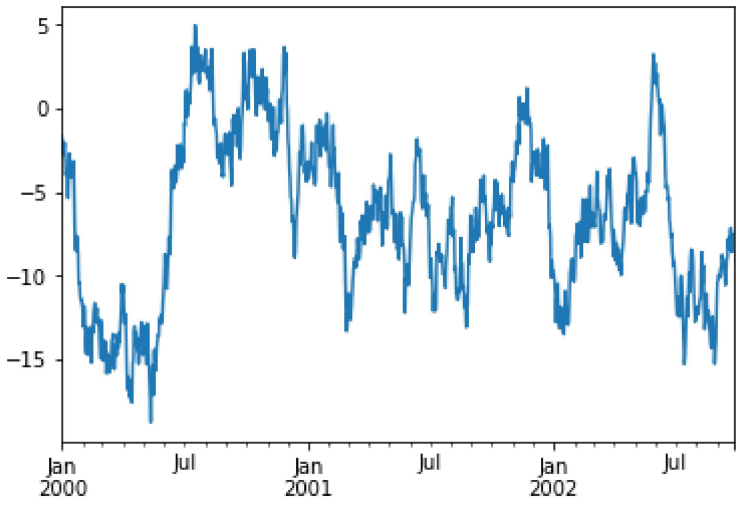
In [ ]:
```
df3.groupby("grade", observed=False).size()
```

Out[ ]:
```
grade
A    2
D    1
E    2
W    1
dtype: int64
```

In [ ]:
```
ts = pd.Series(np.random.randn(1000), index=pd.date_range("1/1/2000", periods=10

ts = ts.cumsum()

ts.plot();
```




In [ ]: