import pandas as pd#1

import numpy as np#2

import matplotlib.pyplot as plt#3

#creating a Series by passing a list of values,letting pandas create a default integer index

s=pd.Series([1,3,5,np.nan,6,8])#4

print (s)#5

print('-'\*20)

#creating a DataFrame by passing a NumPy array, with a datetime index and labelled columns.

dates=pd.date\_range('20130101',periods=6)#6

print (dates)#7

print('-'\*20)

#8

df=pd.DataFrame(np.random.randn(6,4),index=dates,columns=list('ABCD'))

print (df)#9

print('-'\*20)

#creating a DataFrame by passing a dict of objects that can be converted to series-like.

#10

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'

})

print("df2:\n",df2)#11

print('-'\*20)

#The columns of the resulting DataFrame have different dtypes.

print ("df2.dtypes :\n",df2.dtypes )#12

print('-'\*20)

#df2.<TAB> #Notice E #13

#Viewing Data

#view the top and bottom rows of the frame.

print ("df.head():\n",df.head())#14

print('-'\*20)

print ("df.tail(3):\n",df.tail(3))#15

print('-'\*20)

#Display the index,columns, and the underlying NumPy data.

print ("df.index\n",df.index)#16

print('-'\*20)

print ("df.columns:\n",df.columns)#17

print('-'\*20)

print ("df.values :\n",df.values)#18

print('-'\*20)

#describe() shows a quick statistic summery of your data.

print ("df.describe() :\n",df.describe())#19

print('-'\*20)

#Transposing your data

print ("df.T :\n",df.T)#20

print('-'\*20)

#Sorting by an axis

print ("df.sort\_index :\n",df.sort\_index(axis=1,ascending=False))#21

print('-'\*20)

#Sorting by values

print ("df.sort\_values :\n:",df.sort\_values(by='B'))#22

print('-'\*20)

#Selection

#Selecting a single column, which yields a Series, equivalent to df.A

print ("df['A']:\n",df['A'])#23

print('-'\*20)

#Selecting via [], which slices the rows.

print ("df[0:3]:\n",df[0:3])#24

print('-'\*20)

print ("df['20130102':'20130104'] :\n",df['20130102':'20130104'])#25

print('-'\*20)

#Selection by Label

#Get a cross section using a label

print("df.loc[dates[0]]:\n",df.loc[dates[0]])#26

print('-'\*20)

#selecting on a multi-axis by label

print("df.loc[:,['A','B']]:\n",df.loc[:,['A','B']])#27

print('-'\*20)

#Showing label slicing , both endpoints are included

print("df.loc :\n",df.loc['20130102':'20130104',['A','B']])#28

print('-'\*20)

#reduction in the dimensions of the returned object.

print ("df.loc['20130102',['A','B']]:\n",df.loc['20130102',['A','B']])#29

print('-'\*20)

#for getting a scalar value

print ("df.loc[dates[0],'A'] :\n",df.loc[dates[0],'A'])#30

print('-'\*20)

#for getting fast access to a scalar

print ("df.at[dates[0],'A']:\n",df.at[dates[0],'A'])#31

print('-'\*20)

#Selection by position

#Select via the position of the passed integers.

print ("df.iloc[3]:\n",df.iloc[3])#32

print('-'\*20)

#by integer slices, acting similar to numpy/python.

print ("df.iloc[3:5,0:2]:\n",df.iloc[3:5,0:2])#33

print('-'\*20)

#By lists of integer position locations,similar to numpy/python.

print ("df.iloc[[1,2,4],[0,2]]:\n",df.iloc[[1,2,4],[0,2]])#34

print('-'\*20)

#For slicing rows explicitely

print ('df.iloc[1:3,:]:\n',df.iloc[1:3,:])#35

print('-'\*20)

#For slicing columns explicitely.

print ("df.iloc[:,1:3]:\n",df.iloc[:,1:3])#36

print('-'\*20)

#For getting a value explicitely

print("df.iloc[1,1]:\n",df.iloc[1,1])#37

print('-'\*20)

#For getting fast access to a scalar

print("df.iat[1,1] :\n",df.iat[1,1])#38

print('-'\*20)

#Boolean Indexing

#Using a single column's values to select data.

print("df[df.A>0]:\n",df[df.A>0])#39

print('-'\*20)

#Selecting values from a DataFrame where a boolean condition is met.

print("df[df>0] :\n",df[df>0])#40

print('-'\*20)

#Using the isin() method for filtering

print('-'\*20)

df2=df.copy()#41

print("df:\n",df)

print('-'\*20)

df2['E']=['one','one','two','three','four','three']#42

print("df2:\n",df2)#43

print('-'\*20)

print ("df2[df2['E'].isin(['two','four']) :\n",df2[df2['E'].isin(['two','four'])])#44

#Setting

#Setting a new column automatically aligns the data by the indexes.

s1=pd.Series([1,2,3,4,5,6],index=pd.date\_range('20130102',periods=6))#45

print(s1)#46

print('-'\*20)

df['F']=s1 #47

print("df:\n",df)

print('-'\*20)

#Setting values by label

df.at[dates[0],'A']=0 #48

print("df:\n",df)

print('-'\*20)

#Setting values by position

df.iat[0,1]=0 #49

print("df:\n",df)

print('-'\*20)

#Setting by assigning with a numpy array

df.loc[:,'D']=np.array([5]\*len(df))#50

print("df:\n",df)#51

print('-'\*20)

#A where operation with setting.

df2=df.copy()#52

df2[df2>0]=-df2#53

print("df2:\n",df2)#54

print('-'\*20)

#Missing Data

#pandas uses the value np.nan to represent missing data.

#It is by default not included in computations.

#Reindexing allows you to change/add/delete the index

#on a specified axis.This returns a copy of the data.

df1=df.reindex(index=dates[0:4],columns=list(df.columns)+['E'])#55

df1.loc[dates[0]:dates[1],'E']=1 #56

print("df1:\n",df1)#57

print('-'\*20)

#To drop any rows that have missing data.

print("After dropping missing Data :\n",df1.dropna(how='any'))#58

print('-'\*20)

#Filling missing data

print("After Filling missing Data :\n",df1.fillna(value=5))#59

print('-'\*20)

#To get the boolean mask where values are nan

print("Boolean mask:\n",pd.isna(df1))#60

print('-'\*20)

#Operations

#Stats

#Operations in general exclude missing data

#Performing a descriptive statistics

print(df.mean()) #61

print('-'\*20)

#Same operation on the other axis

print(df.mean(1)) #62

print('-'\*20)

#Operating with objects that have different dimensionality

#and need alignment. In addition, pandas automatically

#broadcasts along the specified dimension.

s = pd.Series([1,3,5,np.nan,6,8], index=dates).shift(2) #63

print("s :\n",s) #64

print('-'\*20)

print("df.sub(s, axis='index') :\n",df.sub(s, axis='index'))#65

print('-'\*20)

#Applying functions to the data:

print("df.apply(np.cumsum) :\n",df.apply(np.cumsum) )#66

print('-'\*20)

print(df.apply(lambda x: x.max() - x.min()) )#67

s=pd.Series(np.random.randint(0,7,size=10)) #68

print(s)#69

print('-'\*20)

print(s.value\_counts())#70

lst=['A','B','C','Aaba','Baca',np.nan,'CABA','dog','cat']

print(lst)

print('-'\*20)

#s=pd.Series(['A','B','C','Aaba','Baca',np.nan,'CABA','dog','cat'])

s=pd.Series(lst)#71

print(s.str.lower())#72

#Concatenating pandas objects together with concat()

df=pd.DataFrame(np.random.randn(10,4))#73

print("df:\n",df)#74

print('-'\*20)

#break it into pieces

pieces=[df[:3],df[3:7],df[7:]]#75

print("pieces:")

print(pieces[0])

print('-'\*20)

print(pieces[1])

print('-'\*20)

print(pieces[2])

print('-'\*20)

print("pd.concat:\n",pd.concat(pieces)) #76

print('-'\*20)

#Join SQL style merges

left=pd.DataFrame({'key':['foo','foo'],'lval':[1,2]})#77

right=pd.DataFrame({'key':['foo','foo'],'rval':[4,5]})#78

print("left:\n",left)#79

print('-'\*20)

print("right:\n",right)#80

print('-'\*20)

print("Merging:\n",pd.merge(left,right,on='key'))#81

print('-'\*20)

#Another example of merging

left=pd.DataFrame({'key':['foo','bar'],'lval':[1,2]})#82

right=pd.DataFrame({'key':['foo','bar'],'rval':[4,5]})#83

print("left:\n",left)#84

print('-'\*20)

print("right:\n",right)#85

print('-'\*20)

print("Merging:\n",pd.merge(left,right,on='key'))#86

print('-'\*20)

#Append: Append rows to a dataframe

df=pd.DataFrame(np.random.randn(8,4),columns=['A','B','C','D'])#87

print("df:\n",df)#88

print('-'\*20)

s=df.iloc[3] #89

print("s: \n",s)

print("df.append:\n",df.append(s,ignore\_index=True))#90

print('-'\*20)

df = pd.DataFrame(

{'A' : ['foo', 'bar', 'foo', 'bar','foo', 'bar', 'foo', 'foo'],

'B' : ['one', 'one', 'two', 'three','two', 'two', 'one', 'three'],

'C' : np.random.randn(8),

'D' : np.random.randn(8)})#91

print(df)#92

print('-'\*20)

#Grouping and then applying the sum() function to the resulting groups.

print(df.groupby('A').sum())#93

print('-'\*20)

#Grouping by multiple columns forms a hierarchical index,

#and again we can apply the sum function.

print(df.groupby(['A','B']).sum())#94

print('-'\*20)