# Loc和iloc

*#-\*- coding: utf-8 -\*-  
# @Time : 2018/10/30 11:46  
# @Author : Z  
# @Email : S  
# @File : 12loc\_iloc.py***import** pandas **as** pd  
**import** numpy **as** np  
dfl = pd.DataFrame(np.random.randn(5,4), columns=list(**'ABCD'**),  
 index=pd.date\_range(**'20130101'**,periods=5))  
  
print(dfl)  
*# A B C D  
# 2013-01-01 1.075770 -0.109050 1.643563 -1.469388  
# 2013-01-02 0.357021 -0.674600 -1.776904 -0.968914  
# 2013-01-03 -1.294524 0.413738 0.276662 -0.472035  
# 2013-01-04 -0.013960 -0.362543 -0.006154 -0.923061  
# 2013-01-05 0.895717 0.805244 -1.206412 2.565646*print(dfl.loc[**'20130102'**:**'20130104'**])  
print(dfl.loc[**'20130102'**:**'20130104'**])  
print(dfl.loc[**'20130102'**:**'20130104'**,**"A"**:**"C"**])  
print(dfl.iloc[2:4])  
print(dfl.iloc[2:4,0:3])  
print(dfl.loc[:, **lambda** df: [**'A'**, **'B'**]])  
  
  
pd.read\_pickle()

# Pickle和其他函数

**import** pandas **as** pd  
original\_df = pd.DataFrame({**"foo"**: range(5), **"bar"**: range(5, 10)})  
print(original\_df)  
pd.to\_pickle(original\_df, **"./dummy.pkl"**)  
unpickled\_df = pd.read\_pickle(**"./dummy.pkl"**)  
print(unpickled\_df)  
  
**import** numpy **as** np  
print(pd.cut(np.array([.2, 1.4, 2.5, 6.2, 9.7, 2.1]), 3, retbins=**True**))  
*# (0.19, 3.367],(3.367, 6.533], (6.533, 9.7]  
# ([(0.191, 3.367], (0.191, 3.367], (0.191, 3.367], (3.367, 6.533],  
# (6.533, 9.7], (0.191, 3.367]]  
# Categories (3, object): [(0.191, 3.367] < (3.367, 6.533] < (6.533, 9.7]],  
# array([ 0.1905 , 3.36666667, 6.53333333, 9.7 ]))*pd.cut(np.array([.2, 1.4, 2.5, 6.2, 9.7, 2.1]), 3,  
 labels=[**"good"**,**"medium"**,**"bad"**])  
*# [good, good, good, medium, bad, good]  
# Categories (3, object): [good < medium < bad]*pd.cut(np.ones(5), 4, labels=**False**)  
*# array([1, 1, 1, 1, 1], dtype=int64)*print(pd.qcut(range(5), 3, labels=[**"good"**, **"medium"**, **"bad"**]))  
  
s1 = pd.Series([**'a'**, **'b'**])  
s2 = pd.Series([**'c'**, **'d'**])  
print(pd.concat([s1, s2],axis=0))  
print(pd.concat([s1, s2],axis=1))  
  
  
df1 = pd.DataFrame([[**'a'**, 1], [**'b'**, 2]],columns = [**'letter'**, **'number'**])  
print(df1)  
df2 = pd.DataFrame([[**'c'**, 3], [**'d'**, 4]],columns = [**'letter'**, **'number'**])  
print(df2)  
print(pd.concat([df1, df2],ignore\_index=**True**))  
print(pd.concat([df1, df2],ignore\_index=**True**,axis=1))  
*#one-hot编码形式*s = pd.Series(list(**'abca'**))  
print(s)  
pd.get\_dummies(s)  
*# a b c  
# 0 1 0 0  
# 1 0 1 0  
# 2 0 0 1  
# 3 1 0 0*df = pd.DataFrame({**'A'**: [**'a'**, **'b'**, **'a'**], **'B'**: [**'b'**, **'a'**, **'c'**],  
 **'C'**: [1, 2, 3]})  
print(df)  
print(pd.get\_dummies(df, prefix=[**'col1'**, **'col2'**]))  
*# prefix前缀:字符串,字符串,列表或dict的字符串,默认没有字符串附  
# 加数据帧通过列名列表长度等于列数当调用数据帧上的。  
# 另外,“前缀”可以是一个字典映射列名称前缀。  
# col2\_a col2\_b col2\_c  
# 0 1 0  
# 1 0 0  
# 0 0 1  
  
# pd.factorize()  
# #实现说明:该方法负责3 #  
# # 1。)强迫数据数组类(ndarray、索引扩展数组)  
# # 2)分解标签和--物品  
# # 3)也许索引中的输出  
  
# pd.eval()***import** pandas **as** pd  
s = pd.Series([**'1.0'**, **'2'**, -3])  
print(pd.to\_numeric(s))  
s = pd.Series([**'apple'**, **'1.0'**, **'2'**, -3])  
*# print(pd.to\_numeric(s))*print(pd.to\_numeric(s, errors=**'ignore'**))  
print(pd.to\_numeric(s, errors=**'coerce'**))

# GroupBy函数

*#homework-学习groupy用法  
# data.groupby(func, axis=0).mean()  
# data.groupby(['col1', 'col2'])['col3'].mean()***import** pandas **as** pd  
data=pd.DataFrame([[1,2,3],[1,3,3],[4,5,6],[4,3,6]],columns=[**"one"**,**"two"**,**"three"**])  
*# one two three  
# 0 1 2 3  
# 0 1 3 3  
# 1 4 5 6  
# 1 4 3 6  
  
# 1 2 3 3 3  
# 4 5 3 6 6*print(data)  
print(data.groupby(by=[**"one"**])[**"two"**])  
print(data.groupby(by=[**"one"**])[**"two"**].mean())  
print(data.groupby(by=[**"one"**])[**"two"**].mean())  
*#等价写法*print(data[**"two"**].groupby(by=data[**"one"**]).mean())  
*# print(data.groupby(['one', 'two']).mean())  
# one  
# 1 2.5  
# 4 4.0  
# Name: two, dtype: float64  
# one  
# 1 2.5  
# 4 4.0  
# Name: two, dtype: float64***import** numpy **as** np  
*#根据one列进行聚合，在对其余列进行求解均值*print(data.groupby([**'one'**]).mean())  
print(**"=="**\*100)  
*#根据one列进行聚合，在对其余列进行求解均值*print(data.groupby([**"one"**]).transform(**lambda** x:np.mean(x)))  
print(data.groupby([**"one"**]).transform(**lambda** x:(x-np.mean(x))/(np.std(x))))  
print(**"=="**\*100)  
*#根据one列进行聚合，在对指定列进行求解均值，如”two“*print(data.groupby([**"one"**]).apply(**lambda** data:np.mean(data[**"two"**])))  
  
  
df = pd.DataFrame(np.random.randn(3, 3),columns=[**"a"**,**"b"**,**"c"**],dtype=**"int32"**)  
*# >>> df  
# 0 1 2  
# 0 -0.029638 1.081563 1.280300  
# 1 0.647747 0.831136 -1.549481  
# 2 0.513416 -0.884417 0.195343  
# df = df.applymap(lambda x: '%.2f' % x)  
# print(df)*df1 = df.applymap(**lambda** x:x.max())  
print(df1)  
print(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**)  
*# df  
# 0 1 2  
# 0 -0.03 1.08 1.28  
# 1 0.65 0.83 -1.55  
# 2 0.51 -0.88 0.20  
  
# df2=df.drop(["a"],axis=1)*df2=df.drop(**"a"**,axis=1)  
print(df2)  
df3=pd.DataFrame(np.random.randn(3, 3),columns=[**"a"**,**"b"**,**"c"**],index=[**"one"**,**"two"**,**"three"**])  
print(df3)  
df4=df3.reindex([**"three"**,**"two"**,**"one"**])  
print(df4)  
  
df4.select(**"two"**)