Q1

Object creation

In [1]:

```
import pandas as pd
import numpy as np
```

In [2]:

```
var_series = pd.Series([1,2,3,np.nan,5.52,6])
print(var_series)
```

```
0    1.00
1    2.00
2    3.00
3     NaN
4    5.52
5    6.00
dtype: float64
```

In [3]:

```
dates = pd.date_range("20200101", periods=6)
dates
df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list("ABCD"))
df
```

Out[3]:

	A	B	C	D
2020-01-01	-0.526724	1.202290	-0.126403	-1.844612
2020-01-02	0.578715	-0.651473	0.752117	-0.463860
2020-01-03	0.295356	-0.119894	1.015159	-0.907958
2020-01-04	-1.229944	1.464354	-0.002804	-0.244041
2020-01-05	-1.037098	-0.885968	-0.280248	0.920097
2020-01-06	1.666171	0.260483	0.128384	0.866503

In [4]:

```
df2 = pd.DataFrame({
        "A": 1.0,
        "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[4]:

	A	B	C	D	E	F
0	1.0	2013-01-02	1.0	3	test	foo
1	1.0	2013-01-02	1.0	3	train	foo
2	1.0	2013-01-02	1.0	3	test	foo
3	1.0	2013-01-02	1.0	3	train	foo

In [5]:

```
df2.dtypes
```

Out[5]:

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

In [6]:

```
# df2.<TAB>
```

Viewing data

In [7]:

```
df2.head()
```

Out[7]:

	A	B	C	D	E	F
0	1.0	2013-01-02	1.0	3	test	foo
1	1.0	2013-01-02	1.0	3	train	foo
2	1.0	2013-01-02	1.0	3	test	foo
3	1.0	2013-01-02	1.0	3	train	foo

In [8]:

```
df2.tail(2)
```

Out[8]:

	A	B	C	D	E	F
2	1.0	2013-01-02	1.0	3	test	foo
3	1.0	2013-01-02	1.0	3	train	foo

In [9]:

```
df2.index
```

Out[9]:

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

In [10]:

```
df.index
```

Out[10]:

```
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 [11]:

```
df.to_numpy()
```

Out[11]:

Out[11]:

```
array([[-0.52672397,  1.20228978, -0.12640307, -1.84461231],
       [ 0.57871481, -0.65147285,  0.75211707, -0.46386039],
       [ 0.29535621, -0.11989426,  1.01515909, -0.90795843],
       [-1.22994383,  1.46435437, -0.00280384, -0.24404129],
       [-1.03709789, -0.88596758, -0.2802479 ,  0.92009653],
       [ 1.66617055,  0.26048304,  0.12838446,  0.86650279]])
```

In [12]:

```
df.describe()
```

Out[12]:

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	-0.042254	0.211632	0.247701	-0.278979
std	1.100303	0.960488	0.517513	1.061333
min	-1.229944	-0.885968	-0.280248	-1.844612
25%	-0.909504	-0.518578	-0.095503	-0.796934
50%	-0.115684	0.070294	0.062790	-0.353951
75%	0.507875	0.966838	0.596184	0.588867
max	1.666171	1.464354	1.015159	0.920097

In [13]:

```
df.T
```

Out[13]:

	2020-01-01	2020-01-02	2020-01-03	2020-01-04	2020-01-05	2020-01-06
A	-0.526724	0.578715	0.295356	-1.229944	-1.037098	1.666171
B	1.202290	-0.651473	-0.119894	1.464354	-0.885968	0.260483
C	-0.126403	0.752117	1.015159	-0.002804	-0.280248	0.128384
D	-1.844612	-0.463860	-0.907958	-0.244041	0.920097	0.866503

In [14]:

```
df.sort_index(axis=1, ascending=False)
```

Out[14]:

	D	C	B	A
2020-01-01	-1.844612	-0.126403	1.202290	-0.526724
2020-01-02	-0.463860	0.752117	-0.651473	0.578715
2020-01-03	-0.907958	1.015159	-0.119894	0.295356
2020-01-04	-0.244041	-0.002804	1.464354	-1.229944
2020-01-05	0.920097	-0.280248	-0.885968	-1.037098
2020-01-06	0.866503	0.128384	0.260483	1.666171

In [15]:

```
df.sort_values(by="B")
```

Out[15]:

	A	B	C	D

	A	B	C	D
2020-01-05	-1.037098	-0.885968	-0.280248	0.920097
2020-01-02	0.578715	-0.651473	0.752117	-0.463860
2020-01-03	0.295356	-0.119894	1.015159	-0.907958
2020-01-06	1.666171	0.260483	0.128384	0.866503
2020-01-01	-0.526724	1.202290	-0.126403	-1.844612
2020-01-04	-1.229944	1.464354	-0.002804	-0.244041

Selection

In [16]:

```
df["A"]
```

Out[16]:

```
2020-01-01   -0.526724
2020-01-02    0.578715
2020-01-03    0.295356
2020-01-04   -1.229944
2020-01-05   -1.037098
2020-01-06    1.666171
Freq: D, Name: A, dtype: float64
```

In [17]:

```
df[0:3]
```

Out[17]:

	A	B	C	D
2020-01-01	-0.526724	1.202290	-0.126403	-1.844612
2020-01-02	0.578715	-0.651473	0.752117	-0.463860
2020-01-03	0.295356	-0.119894	1.015159	-0.907958

In [18]:

```
df.loc["20200102":"20200104"]
```

Out[18]:

	A	B	C	D
2020-01-02	0.578715	-0.651473	0.752117	-0.463860
2020-01-03	0.295356	-0.119894	1.015159	-0.907958
2020-01-04	-1.229944	1.464354	-0.002804	-0.244041

Selection by label

In [19]:

```
df.loc[dates[0]]
```

Out[19]:

```
A   -0.526724
B    1.202290
C   -0.126403
D   -1.844612
Name: 2020-01-01 00:00:00, dtype: float64
```

In [20]:

```
df.loc["20200102":"20200104", ["A", "B"]]
```

Out[20]:

	A	B
2020-01-02	0.578715	-0.651473
2020-01-03	0.295356	-0.119894
2020-01-04	-1.229944	1.464354

In [21]:

```
df.loc["20200102", ["A", "B"]]
```

Out[21]:

```
A    0.578715
B   -0.651473
Name: 2020-01-02 00:00:00, dtype: float64
```

In [22]:

```
df.at[dates[0], "A"]
```

Out[22]:

```
-0.526723965502238
```

In [23]:

```
df.at[dates[0], "A"]
```

Out[23]:

```
-0.526723965502238
```

Selection by position

In [24]:

```
df.iloc[3]
```

Out[24]:

```
A   -1.229944
B    1.464354
C   -0.002804
D   -0.244041
Name: 2020-01-04 00:00:00, dtype: float64
```

In [25]:

```
df.iloc[3:5, 0:2]
```

Out[25]:

	A	B
2020-01-04	-1.229944	1.464354
2020-01-05	-1.037098	-0.885968

In [26]:

```
df.iloc[[1, 2, 4], [0, 2]]
```

Out[26]:

	A	C

	A	C
2020-01-02	0.578715	0.752117
2020-01-03	0.295356	1.015159
2020-01-05	-1.037098	-0.280248

In [27]:

```
df.iloc[1:3, :]
```

Out[27]:

	A	B	C	D
2020-01-02	0.578715	-0.651473	0.752117	-0.463860
2020-01-03	0.295356	-0.119894	1.015159	-0.907958

In [28]:

```
df.iloc[:, 1:3]
```

Out[28]:

	B	C
2020-01-01	1.202290	-0.126403
2020-01-02	-0.651473	0.752117
2020-01-03	-0.119894	1.015159
2020-01-04	1.464354	-0.002804
2020-01-05	-0.885968	-0.280248
2020-01-06	0.260483	0.128384

In [29]:

```
df.iloc[1, 1]
```

Out[29]:

```
-0.6514728548193711
```

Boolean indexing

In [30]:

```
df[df["A"] > 0]
```

Out[30]:

	A	B	C	D
2020-01-02	0.578715	-0.651473	0.752117	-0.463860
2020-01-03	0.295356	-0.119894	1.015159	-0.907958
2020-01-06	1.666171	0.260483	0.128384	0.866503

In [31]:

```
df > 0
```

Out[31]:

	A	B	C	D
2020-01-01	False	True	False	False

	A	B	C	D
2020-01-02	True	False	True	False
2020-01-03	True	False	True	False
2020-01-04	False	True	False	False
2020-01-05	False	False	False	True
2020-01-06	True	True	True	True

In [32]:

```
df[df > 0]
```

Out[32]:

	A	B	C	D
2020-01-01	NaN	1.202290	NaN	NaN
2020-01-02	0.578715	NaN	0.752117	NaN
2020-01-03	0.295356	NaN	1.015159	NaN
2020-01-04	NaN	1.464354	NaN	NaN
2020-01-05	NaN	NaN	NaN	0.920097
2020-01-06	1.666171	0.260483	0.128384	0.866503

In [33]:

```
df2 = df.copy()
df2
```

Out[33]:

	A	B	C	D
2020-01-01	-0.526724	1.202290	-0.126403	-1.844612
2020-01-02	0.578715	-0.651473	0.752117	-0.463860
2020-01-03	0.295356	-0.119894	1.015159	-0.907958
2020-01-04	-1.229944	1.464354	-0.002804	-0.244041
2020-01-05	-1.037098	-0.885968	-0.280248	0.920097
2020-01-06	1.666171	0.260483	0.128384	0.866503

In [34]:

```
df2["E"] = ["one", "one", "two", "three", "four", "three"]
df2
```

Out[34]:

	A	B	C	D	E
2020-01-01	-0.526724	1.202290	-0.126403	-1.844612	one
2020-01-02	0.578715	-0.651473	0.752117	-0.463860	one
2020-01-03	0.295356	-0.119894	1.015159	-0.907958	two
2020-01-04	-1.229944	1.464354	-0.002804	-0.244041	three
2020-01-05	-1.037098	-0.885968	-0.280248	0.920097	four
2020-01-06	1.666171	0.260483	0.128384	0.866503	three

In [35]:

```
df2[df2["E"].isin(["two", "four"])]
```

Out[35]:

	A	B	C	D	E
2020-01-03	0.295356	-0.119894	1.015159	-0.907958	two
2020-01-05	-1.037098	-0.885968	-0.280248	0.920097	four

In [36]:

```
df2["E"].isin(["two", "four"])
```

Out[36]:

```
2020-01-01    False
2020-01-02    False
2020-01-03     True
2020-01-04    False
2020-01-05     True
2020-01-06    False
Freq: D, Name: E, dtype: bool
```

Setting

In [37]:

```
s1 = pd.Series([1, 2, 3, 4, 5, 6], index=pd.date_range("20210606", periods=6))
s1
```

Out[37]:

```
2021-06-06    1
2021-06-07    2
2021-06-08    3
2021-06-09    4
2021-06-10    5
2021-06-11    6
Freq: D, dtype: int64
```

In [38]:

```
df["F"] = s1
df
```

Out[38]:

	A	B	C	D	F
2020-01-01	-0.526724	1.202290	-0.126403	-1.844612	NaN
2020-01-02	0.578715	-0.651473	0.752117	-0.463860	NaN
2020-01-03	0.295356	-0.119894	1.015159	-0.907958	NaN
2020-01-04	-1.229944	1.464354	-0.002804	-0.244041	NaN
2020-01-05	-1.037098	-0.885968	-0.280248	0.920097	NaN
2020-01-06	1.666171	0.260483	0.128384	0.866503	NaN

In [39]:

```
df.at[dates[0], "A"] = 0
df
```

Out[39]:

	A	B	C	D	F
2020-01-01	0.000000	1.202290	-0.126403	-1.844612	NaN
2020-01-02	0.578715	-0.651473	0.752117	-0.463860	NaN
2020-01-03	0.295356	-0.119894	1.015159	-0.907958	NaN

	A	B	C	D	F
2020-01-04	-1.229944	1.464354	-0.002804	-0.244041	NaN
2020-01-05	-1.037098	-0.885968	-0.280248	0.920097	NaN
2020-01-06	1.666171	0.260483	0.128384	0.866503	NaN

In [40]:

```
df.iat[0, 1] = 0
df
```

Out[40]:

	A	B	C	D	F
2020-01-01	0.000000	0.000000	-0.126403	-1.844612	NaN
2020-01-02	0.578715	-0.651473	0.752117	-0.463860	NaN
2020-01-03	0.295356	-0.119894	1.015159	-0.907958	NaN
2020-01-04	-1.229944	1.464354	-0.002804	-0.244041	NaN
2020-01-05	-1.037098	-0.885968	-0.280248	0.920097	NaN
2020-01-06	1.666171	0.260483	0.128384	0.866503	NaN

In [41]:

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

Out[41]:

	A	B	C	D	F
2020-01-01	0.000000	0.000000	-0.126403	5	NaN
2020-01-02	0.578715	-0.651473	0.752117	5	NaN
2020-01-03	0.295356	-0.119894	1.015159	5	NaN
2020-01-04	-1.229944	1.464354	-0.002804	5	NaN
2020-01-05	-1.037098	-0.885968	-0.280248	5	NaN
2020-01-06	1.666171	0.260483	0.128384	5	NaN

In [42]:

```
df2 = df.copy()
df2[df2 > 0] = -df2
df2
```

Out[42]:

	A	B	C	D	F
2020-01-01	0.000000	0.000000	-0.126403	-5	NaN
2020-01-02	-0.578715	-0.651473	-0.752117	-5	NaN
2020-01-03	-0.295356	-0.119894	-1.015159	-5	NaN
2020-01-04	-1.229944	-1.464354	-0.002804	-5	NaN
2020-01-05	-1.037098	-0.885968	-0.280248	-5	NaN
2020-01-06	-1.666171	-0.260483	-0.128384	-5	NaN

Missing data

In [43]:

```
df1 = df.reindex(index=dates[0:4], columns=list(df.columns) + ["E"])
```

```
df1.loc[dates[0] : dates[1], "E"] = 1
df1
```

Out[43]:

	A	B	C	D	F	E
2020-01-01	0.000000	0.000000	-0.126403	5	NaN	1.0
2020-01-02	0.578715	-0.651473	0.752117	5	NaN	1.0
2020-01-03	0.295356	-0.119894	1.015159	5	NaN	NaN
2020-01-04	-1.229944	1.464354	-0.002804	5	NaN	NaN

In [44]:

```
df1.dropna(how="any") #to drop any row that have missing data
```

Out[44]:

A	B	C	D	F	E

In [45]:

```
df1.fillna(value=5)
```

Out[45]:

	A	B	C	D	F	E
2020-01-01	0.000000	0.000000	-0.126403	5	5.0	1.0
2020-01-02	0.578715	-0.651473	0.752117	5	5.0	1.0
2020-01-03	0.295356	-0.119894	1.015159	5	5.0	5.0
2020-01-04	-1.229944	1.464354	-0.002804	5	5.0	5.0

In [46]:

```
pd.isna(df1) # To get the boolean mask where values are nan.
```

Out[46]:

	A	B	C	D	F	E
2020-01-01	False	False	False	False	True	False
2020-01-02	False	False	False	False	True	False
2020-01-03	False	False	False	False	True	True
2020-01-04	False	False	False	False	True	True

Operations

In [47]:

```
df.mean()
```

Out[47]:

```
A    0.045533
B    0.011250
C    0.247701
D    5.000000
F         NaN
dtype: float64
```

In [48]:

```
df.mean(1) #same operation but on rows
```

Out[48]:

```
2020-01-01    1.218399
2020-01-02    1.419840
2020-01-03    1.547655
2020-01-04    1.307902
2020-01-05    0.699172
2020-01-06    1.763760
Freq: D, dtype: float64
```

In [49]:

```
s = pd.Series([1, 3, 5, np.nan, 6, 8], index=dates).shift(2) #shift will shift elements by 2 indexes
s
```

Out[49]:

```
2020-01-01    NaN
2020-01-02    NaN
2020-01-03    1.0
2020-01-04    3.0
2020-01-05    5.0
2020-01-06    NaN
Freq: D, dtype: float64
```

In [50]:

```
df.sub(s, axis="index")
```

Out[50]:

	A	B	C	D	F
2020-01-01	NaN	NaN	NaN	NaN	NaN
2020-01-02	NaN	NaN	NaN	NaN	NaN
2020-01-03	-0.704644	-1.119894	0.015159	4.0	NaN
2020-01-04	-4.229944	-1.535646	-3.002804	2.0	NaN
2020-01-05	-6.037098	-5.885968	-5.280248	0.0	NaN
2020-01-06	NaN	NaN	NaN	NaN	NaN

Apply

In [51]:

```
df.apply(np.cumsum)
```

Out[51]:

	A	B	C	D	F
2020-01-01	0.000000	0.000000	-0.126403	5	NaN
2020-01-02	0.578715	-0.651473	0.625714	10	NaN
2020-01-03	0.874071	-0.771367	1.640873	15	NaN
2020-01-04	-0.355873	0.692987	1.638069	20	NaN
2020-01-05	-1.392971	-0.192980	1.357821	25	NaN
2020-01-06	0.273200	0.067503	1.486206	30	NaN

In [52]:

```
df.apply(lambda x: x.max() - x.min())
```

Out[52]:

```
A    2.896114
B    2.350322
C    1.295407
D    0.000000
F         NaN
dtype: float64
```

Histogramming

In [53]:

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

Out[53]:

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

In [54]:

```
s.value_counts()
```

Out[54]:

```
1    4
0    2
6    1
4    1
3    1
2    1
dtype: int64
```

String Methods

In [55]:

```
s = pd.Series(["A", "B", "C", "Aaba", "Baca", np.nan, "CABA", "dog", "cat"])
s.str.lower()
```

Out[55]:

```
0       a
1       b
2       c
3    aaba
4    baca
5     NaN
6    caba
7     dog
8     cat
dtype: object
```

Merge

>concat

->concat

In [56]:

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

Out[56]:

	0	1	2	3
0	-1.292982	-1.281205	0.582134	-0.485533
1	-0.139062	-1.017671	-1.495254	0.000375
2	0.283964	-0.031561	0.693738	0.247965
3	0.703311	-1.485781	-1.379576	0.139319
4	-0.741293	-1.916044	-0.825335	1.485869
5	0.706437	-0.345567	-0.595017	1.078053
6	-0.382067	2.581499	0.431233	1.630471
7	0.929631	0.637642	-1.125079	0.472400
8	-0.726499	0.581470	0.564752	-0.369868
9	1.557468	0.469367	-1.593651	0.258787

In [57]:

```
pieces = [df[:3], df[3:7], df[7:]]
pieces
```

Out[57]:

```
[          0         1         2         3
 0 -1.292982 -1.281205  0.582134 -0.485533
 1 -0.139062 -1.017671 -1.495254  0.000375
 2  0.283964 -0.031561  0.693738  0.247965,
           0         1         2         3
 3  0.703311 -1.485781 -1.379576  0.139319
 4 -0.741293 -1.916044 -0.825335  1.485869
 5  0.706437 -0.345567 -0.595017  1.078053
 6 -0.382067  2.581499  0.431233  1.630471,
           0         1         2         3
 7  0.929631  0.637642 -1.125079  0.472400
 8 -0.726499  0.581470  0.564752 -0.369868
 9  1.557468  0.469367 -1.593651  0.258787]
```

In [58]:

```
pd.concat(pieces)
```

Out[58]:

	0	1	2	3
0	-1.292982	-1.281205	0.582134	-0.485533
1	-0.139062	-1.017671	-1.495254	0.000375
2	0.283964	-0.031561	0.693738	0.247965
3	0.703311	-1.485781	-1.379576	0.139319
4	-0.741293	-1.916044	-0.825335	1.485869
5	0.706437	-0.345567	-0.595017	1.078053
6	-0.382067	2.581499	0.431233	1.630471
7	0.929631	0.637642	-1.125079	0.472400
8	-0.726499	0.581470	0.564752	-0.369868
9	1.557468	0.469367	-1.593651	0.258787

->Join

In [59]:

```
left = pd.DataFrame({"key": ["foo", "foo"], "lval": [1, 2]})
left
```

Out[59]:

	key	lval
0	foo	1
1	foo	2

In [60]:

```
right = pd.DataFrame({"key": ["foo", "foo"], "rval": [4, 5]})
right
```

Out[60]:

	key	rval
0	foo	4
1	foo	5

In [61]:

```
pd.merge(left, right, on="key")
```

Out[61]:

	key	lval	rval
0	foo	1	4
1	foo	1	5
2	foo	2	4
3	foo	2	5

In [62]:

```
left = pd.DataFrame({"key": ["foo", "bar"], "lval": [1, 2]})
left
```

Out[62]:

	key	lval
0	foo	1
1	bar	2

In [63]:

```
right = pd.DataFrame({"key": ["foo", "bar"], "rval": [4, 5]})
right
```

Out[63]:

	key	rval
0	foo	4
1	bar	5

In [64]:

```
pd.merge(left, right, on="key")
```

Out[64]:

	key	lval	rval
0	foo	1	4
1	bar	2	5

Grouping

In [65]:

```
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),
      })
df
```

Out[65]:

	A	B	C	D
0	foo	one	-1.696411	-0.054643
1	bar	one	-0.442058	2.073638
2	foo	two	1.769926	0.786099
3	bar	three	-1.621764	-0.774194
4	foo	two	0.824082	0.557521
5	bar	two	0.598641	0.951191
6	foo	one	0.038138	-0.450416
7	foo	three	-1.441256	-1.021644

In [66]:

```
df.groupby("A").sum()
```

Out[66]:

A	C	D
bar	-1.465181	2.250635
foo	-0.505521	-0.183082

In [67]:

```
df.groupby(["A","B"]).sum()
```

Out[67]:

		C	D
A	B		
bar	one	-0.442058	2.073638
	three	-1.621764	-0.774194
	two	0.598641	0.951191

foo	one	C1.658273	D0.505058
A	three	B -1.441256	-1.021644
	two	2.594008	1.343620

Reshaping

->Stack

In [68]:

```
tuples = list(
     zip(*[
               ["bar", "bar", "baz", "baz", "foo", "foo", "qux", "qux"],
               ["one", "two", "one", "two", "one", "two", "one", "two"],
          ])
             )
tuples
```

Out[68]:

```
[('bar', 'one'),
 ('bar', 'two'),
 ('baz', 'one'),
 ('baz', 'two'),
 ('foo', 'one'),
 ('foo', 'two'),
 ('qux', 'one'),
 ('qux', 'two')]
```

In [69]:

```
index = pd.MultiIndex.from_tuples(tuples, names=["first", "second"])
index
```

Out[69]:

```
MultiIndex([('bar', 'one'),
            ('bar', 'two'),
            ('baz', 'one'),
            ('baz', 'two'),
            ('foo', 'one'),
            ('foo', 'two'),
            ('qux', 'one'),
            ('qux', 'two')],
           names=['first', 'second'])
```

In [70]:

```
df = pd.DataFrame(np.random.randn(8, 2), index=index, columns=["A", "B"])
df
```

Out[70]:

		A	B
first	second		
bar	one	0.856205	-0.542738
	two	-0.461640	-1.091291
baz	one	-0.172701	-1.480335
	two	0.274829	-0.084666
foo	one	-1.200299	0.750257
	two	0.475111	0.448757
qux	one	-0.969304	1.591720

		A	B
	two	-1.133011	-0.067266

In [71]:

```
df2 = df[:4]
df2
```

Out[71]:

		A	B
first	second		
bar	one	0.856205	-0.542738
	two	-0.461640	-1.091291
baz	one	-0.172701	-1.480335
	two	0.274829	-0.084666

In [72]:

```
stacked = df2.stack()
stacked
```

Out[72]:

```
first  second
bar    one     A    0.856205
               B   -0.542738
       two     A   -0.461640
               B   -1.091291
baz    one     A   -0.172701
               B   -1.480335
       two     A    0.274829
               B   -0.084666
dtype: float64
```

In [73]:

```
stacked.unstack()
```

Out[73]:

		A	B
first	second		
bar	one	0.856205	-0.542738
	two	-0.461640	-1.091291
baz	one	-0.172701	-1.480335
	two	0.274829	-0.084666

In [74]:

```
stacked.unstack(1)
```

Out[74]:

	second	one	two
first			
bar	A	0.856205	-0.461640
	B	-0.542738	-1.091291
baz	A	-0.172701	0.274829
	B	-1.480335	-0.084666

In [75]:

```
stacked.unstack(0)
```

Out[75]:

		first	bar	baz
second				
one	A	0.856205	-0.172701
	B	-0.542738	-1.480335
two	A	-0.461640	0.274829
	B	-1.091291	-0.084666

Pivot tables

In [76]:

```
df = pd.DataFrame({
          "A": ["one", "one", "two", "three"] * 3,
          "B": ["A", "B", "C"] * 4,
          "C": ["foo", "foo", "foo", "bar", "bar", "bar"] * 2,
          "D": np.random.randn(12),
          "E": np.random.randn(12),
     })
df
```

Out[76]:

	A	B	C	D	E
0	one	A	foo	0.212288	0.712113
1	one	B	foo	0.799675	-1.800744
2	two	C	foo	1.438741	0.521157
3	three	A	bar	-2.467354	-1.829592
4	one	B	bar	-1.665885	-0.067439
5	one	C	bar	1.400246	0.341556
6	two	A	foo	1.728261	0.785517
7	three	B	foo	0.103216	-0.977905
8	one	C	foo	1.150187	1.512769
9	one	A	bar	-0.555344	0.642513
10	two	B	bar	-0.936792	1.540388
11	three	C	bar	-0.315495	-0.989765

In [77]:

```
pd.pivot_table(df, values="D", index=["A", "B"], columns=["C"])
```

Out[77]:

		C	bar	foo
A	B		
one	A	-0.555344	0.212288
	B	-1.665885	0.799675
	C	1.400246	1.150187
three	A	-2.467354	NaN

	B	bar	NaN
	C	foo	0.403216
A	B	-0.315495	NaN
two	A	NaN	1.728261
	B	-0.936792	NaN
	C	NaN	1.438741

Time series

In [78]:

```
rng = pd.date_range("1/1/2012", periods=100, freq="S")
rng
```

Out[78]:

```
DatetimeIndex(['2012-01-01 00:00:00', '2012-01-01 00:00:01',
               '2012-01-01 00:00:02', '2012-01-01 00:00:03',
               '2012-01-01 00:00:04', '2012-01-01 00:00:05',
               '2012-01-01 00:00:06', '2012-01-01 00:00:07',
               '2012-01-01 00:00:08', '2012-01-01 00:00:09',
               '2012-01-01 00:00:10', '2012-01-01 00:00:11',
               '2012-01-01 00:00:12', '2012-01-01 00:00:13',
               '2012-01-01 00:00:14', '2012-01-01 00:00:15',
               '2012-01-01 00:00:16', '2012-01-01 00:00:17',
               '2012-01-01 00:00:18', '2012-01-01 00:00:19',
               '2012-01-01 00:00:20', '2012-01-01 00:00:21',
               '2012-01-01 00:00:22', '2012-01-01 00:00:23',
               '2012-01-01 00:00:24', '2012-01-01 00:00:25',
               '2012-01-01 00:00:26', '2012-01-01 00:00:27',
               '2012-01-01 00:00:28', '2012-01-01 00:00:29',
               '2012-01-01 00:00:30', '2012-01-01 00:00:31',
               '2012-01-01 00:00:32', '2012-01-01 00:00:33',
               '2012-01-01 00:00:34', '2012-01-01 00:00:35',
               '2012-01-01 00:00:36', '2012-01-01 00:00:37',
               '2012-01-01 00:00:38', '2012-01-01 00:00:39',
               '2012-01-01 00:00:40', '2012-01-01 00:00:41',
               '2012-01-01 00:00:42', '2012-01-01 00:00:43',
               '2012-01-01 00:00:44', '2012-01-01 00:00:45',
               '2012-01-01 00:00:46', '2012-01-01 00:00:47',
               '2012-01-01 00:00:48', '2012-01-01 00:00:49',
               '2012-01-01 00:00:50', '2012-01-01 00:00:51',
               '2012-01-01 00:00:52', '2012-01-01 00:00:53',
               '2012-01-01 00:00:54', '2012-01-01 00:00:55',
               '2012-01-01 00:00:56', '2012-01-01 00:00:57',
               '2012-01-01 00:00:58', '2012-01-01 00:00:59',
               '2012-01-01 00:01:00', '2012-01-01 00:01:01',
               '2012-01-01 00:01:02', '2012-01-01 00:01:03',
               '2012-01-01 00:01:04', '2012-01-01 00:01:05',
               '2012-01-01 00:01:06', '2012-01-01 00:01:07',
               '2012-01-01 00:01:08', '2012-01-01 00:01:09',
               '2012-01-01 00:01:10', '2012-01-01 00:01:11',
               '2012-01-01 00:01:12', '2012-01-01 00:01:13',
               '2012-01-01 00:01:14', '2012-01-01 00:01:15',
               '2012-01-01 00:01:16', '2012-01-01 00:01:17',
               '2012-01-01 00:01:18', '2012-01-01 00:01:19',
               '2012-01-01 00:01:20', '2012-01-01 00:01:21',
               '2012-01-01 00:01:22', '2012-01-01 00:01:23',
               '2012-01-01 00:01:24', '2012-01-01 00:01:25',
               '2012-01-01 00:01:26', '2012-01-01 00:01:27',
               '2012-01-01 00:01:28', '2012-01-01 00:01:29',
               '2012-01-01 00:01:30', '2012-01-01 00:01:31',
               '2012-01-01 00:01:32', '2012-01-01 00:01:33',
               '2012-01-01 00:01:34', '2012-01-01 00:01:35',
               '2012-01-01 00:01:36', '2012-01-01 00:01:37',
               '2012-01-01 00:01:38', '2012-01-01 00:01:39'],
              dtype='datetime64[ns]', freq='S')
```

In [79]:

```
ts = pd.Series(np.random.randint(0, 500, len(rng)), index=rng)
ts
```

Out[79]:

```
2012-01-01 00:00:00     22
2012-01-01 00:00:01    392
2012-01-01 00:00:02     89
2012-01-01 00:00:03    313
2012-01-01 00:00:04     56
                      ...
2012-01-01 00:01:35    242
2012-01-01 00:01:36     92
2012-01-01 00:01:37     19
2012-01-01 00:01:38    371
2012-01-01 00:01:39    101
Freq: S, Length: 100, dtype: int32
```

In [80]:

```
ts.resample("5Min").sum()
```

Out[80]:

```
2012-01-01    25580
Freq: 5T, dtype: int32
```

In [81]:

```
rng = pd.date_range("3/6/2012 00:00", periods=5, freq="D") #freq="D" for day
rng
```

Out[81]:

```
DatetimeIndex(['2012-03-06', '2012-03-07', '2012-03-08', '2012-03-09',
               '2012-03-10'],
              dtype='datetime64[ns]', freq='D')
```

In [82]:

```
ts = pd.Series(np.random.randn(len(rng)), index=rng)
ts
```

Out[82]:

```
2012-03-06    0.681524
2012-03-07    0.865551
2012-03-08   -0.107253
2012-03-09    0.248697
2012-03-10    0.012707
Freq: D, dtype: float64
```

In [83]:

```
ts_utc = ts.tz_localize("UTC")
ts_utc
```

Out[83]:

```
2012-03-06 00:00:00+00:00    0.681524
2012-03-07 00:00:00+00:00    0.865551
2012-03-08 00:00:00+00:00   -0.107253
2012-03-09 00:00:00+00:00    0.248697
2012-03-10 00:00:00+00:00    0.012707
Freq: D, dtype: float64
```

In [84]:

```
ts_utc.tz_convert("US/Eastern")
```

Out[84]:

```
2012-03-05 19:00:00-05:00    0.681524
```

```
2012-03-06 19:00:00-05:00    0.865551
2012-03-07 19:00:00-05:00   -0.107253
2012-03-08 19:00:00-05:00    0.248697
2012-03-09 19:00:00-05:00    0.012707
Freq: D, dtype: float64
```

In [85]:

```
rng = pd.date_range("1/1/2012", periods=5, freq="M")
rng
```

Out[85]:

```
DatetimeIndex(['2012-01-31', '2012-02-29', '2012-03-31', '2012-04-30',
               '2012-05-31'],
              dtype='datetime64[ns]', freq='M')
```

In [86]:

```
ts = pd.Series(np.random.randn(len(rng)), index=rng)
ts
```

Out[86]:

```
2012-01-31   -1.985442
2012-02-29    0.383260
2012-03-31    0.566726
2012-04-30    1.765962
2012-05-31    2.221671
Freq: M, dtype: float64
```

In [87]:

```
ps = ts.to_period()
ps
```

Out[87]:

```
2012-01   -1.985442
2012-02    0.383260
2012-03    0.566726
2012-04    1.765962
2012-05    2.221671
Freq: M, dtype: float64
```

In [88]:

```
ps.to_timestamp()
```

Out[88]:

```
2012-01-01   -1.985442
2012-02-01    0.383260
2012-03-01    0.566726
2012-04-01    1.765962
2012-05-01    2.221671
Freq: MS, dtype: float64
```

In [89]:

```
prng = pd.period_range("1990Q1", "2000Q4", freq="Q-NOV")
prng
```

Out[89]:

```
PeriodIndex(['1990Q1', '1990Q2', '1990Q3', '1990Q4', '1991Q1', '1991Q2',
             '1991Q3', '1991Q4', '1992Q1', '1992Q2', '1992Q3', '1992Q4',
             '1993Q1', '1993Q2', '1993Q3', '1993Q4', '1994Q1', '1994Q2',
             '1994Q3', '1994Q4', '1995Q1', '1995Q2', '1995Q3', '1995Q4',
             '1996Q1', '1996Q2', '1996Q3', '1996Q4', '1997Q1', '1997Q2',
             '1997Q3', '1997Q4', '1998Q1', '1998Q2', '1998Q3', '1998Q4',
             '1999Q1', '1999Q2', '1999Q3', '1999Q4', '2000Q1', '2000Q2',
             '2000Q3', '2000Q4'],
            dtype='period[Q-NOV]', freq='Q-NOV')
```

dtype='period[Q-NOV]', freq='Q-NOV')

In [90]:

```
ts = pd.Series(np.random.randn(len(prng)), index=prng)
ts
```

Out[90]:

```
1990Q1   -0.126930
1990Q2    0.778093
1990Q3   -0.599313
1990Q4    0.948907
1991Q1   -0.001504
1991Q2   -0.456994
1991Q3    0.747272
1991Q4    0.232985
1992Q1   -0.632427
1992Q2    0.783387
1992Q3   -0.516375
1992Q4    0.768408
1993Q1    1.557369
1993Q2   -1.074621
1993Q3   -0.936080
1993Q4    0.709034
1994Q1    0.848268
1994Q2   -0.054661
1994Q3   -0.322075
1994Q4    0.592467
1995Q1   -0.326467
1995Q2    0.017041
1995Q3   -0.506050
1995Q4   -0.933511
1996Q1    0.248558
1996Q2   -1.135420
1996Q3   -1.044964
1996Q4   -0.633714
1997Q1   -0.971606
1997Q2    0.368383
1997Q3    0.079161
1997Q4   -0.265583
1998Q1    0.992794
1998Q2   -1.090968
1998Q3   -1.258131
1998Q4   -0.367030
1999Q1    0.068287
1999Q2    0.071022
1999Q3    0.921033
1999Q4    0.775699
2000Q1    0.022633
2000Q2    0.553150
2000Q3    0.626131
2000Q4    0.268843
Freq: Q-NOV, dtype: float64
```

In [91]:

```
ts.index = (prng.asfreq("M", "e") + 1).asfreq("H", "s") + 9
ts
```

Out[91]:

```
1990-03-01 09:00   -0.126930
1990-06-01 09:00    0.778093
1990-09-01 09:00   -0.599313
1990-12-01 09:00    0.948907
1991-03-01 09:00   -0.001504
1991-06-01 09:00   -0.456994
1991-09-01 09:00    0.747272
1991-12-01 09:00    0.232985
1992-03-01 09:00   -0.632427
1992-06-01 09:00    0.783387
1992-09-01 09:00   -0.516375
```

```
1992-12-01 09:00    0.768408
1993-03-01 09:00    1.557369
1993-06-01 09:00   -1.074621
1993-09-01 09:00   -0.936080
1993-12-01 09:00    0.709034
1994-03-01 09:00    0.848268
1994-06-01 09:00   -0.054661
1994-09-01 09:00   -0.322075
1994-12-01 09:00    0.592467
1995-03-01 09:00   -0.326467
1995-06-01 09:00    0.017041
1995-09-01 09:00   -0.506050
1995-12-01 09:00   -0.933511
1996-03-01 09:00    0.248558
1996-06-01 09:00   -1.135420
1996-09-01 09:00   -1.044964
1996-12-01 09:00   -0.633714
1997-03-01 09:00   -0.971606
1997-06-01 09:00    0.368383
1997-09-01 09:00    0.079161
1997-12-01 09:00   -0.265583
1998-03-01 09:00    0.992794
1998-06-01 09:00   -1.090968
1998-09-01 09:00   -1.258131
1998-12-01 09:00   -0.367030
1999-03-01 09:00    0.068287
1999-06-01 09:00    0.071022
1999-09-01 09:00    0.921033
1999-12-01 09:00    0.775699
2000-03-01 09:00    0.022633
2000-06-01 09:00    0.553150
2000-09-01 09:00    0.626131
2000-12-01 09:00    0.268843
Freq: H, dtype: float64
```

Categoricals

In [92]:

```
df = pd.DataFrame(
    {"id": [1, 2, 3, 4, 5, 6], "raw_grade": ["a", "b", "b", "a", "a", "e"]}
)
df
```

Out[92]:

	id	raw_grade
0	1	a
1	2	b
2	3	b
3	4	a
4	5	a
5	6	e

In [93]:

```
df["grade"] = df["raw_grade"].astype("category")
df["grade"]
```

Out[93]:

```
0    a
1    b
2    b
3    a
4    a
```

```
5    e
Name: grade, dtype: category
Categories (3, object): ['a', 'b', 'e']
```

In [94]:

```
df["grade"].cat.categories = ["very good", "good", "very bad"]
df["grade"]
```

Out[94]:

```
0    very good
1         good
2         good
3    very good
4    very good
5     very bad
Name: grade, dtype: category
Categories (3, object): ['very good', 'good', 'very bad']
```

In [95]:

```
df
```

Out[95]:

	id	raw_grade	grade
0	1	a	very good
1	2	b	good
2	3	b	good
3	4	a	very good
4	5	a	very good
5	6	e	very bad

In [96]:

```
df["grade"] = df["grade"].cat.set_categories(
    ["very bad", "bad", "medium", "good", "very good"]
)
df
```

Out[96]:

	id	raw_grade	grade
0	1	a	very good
1	2	b	good
2	3	b	good
3	4	a	very good
4	5	a	very good
5	6	e	very bad

In [97]:

```
df.sort_values(by="grade")
```

Out[97]:

	id	raw_grade	grade
5	6	e	very bad
1	2	b	good
2	3	b	good

	id	raw_grade	grade
0	1	a	very good
3	4	a	very good
4	5	a	very good

In [98]:

```
df.groupby("grade").size()
```

Out[98]:

```
grade
very bad      1
bad           0
medium        0
good          2
very good     3
dtype: int64
```

Plotting

In [99]:

```
import matplotlib.pyplot as plt
```

In [100]:

```
plt.close("all")
```

In [101]:

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

In [102]:

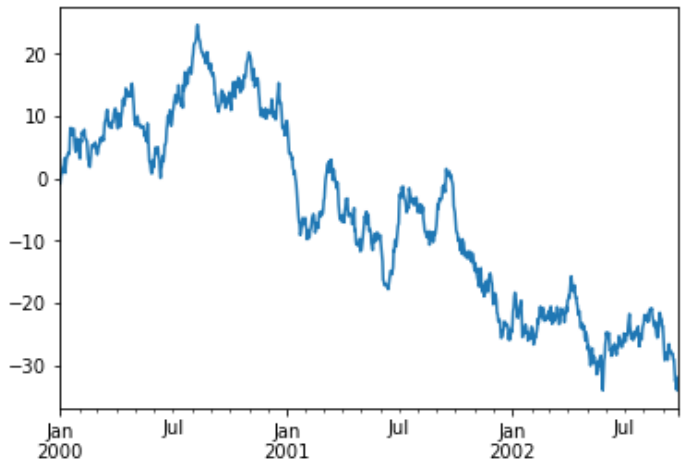
```
ts = ts.cumsum()
```

In [103]:

```
ts.plot()
```

Out[103]:

```
<AxesSubplot:>
```



In [104]:

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

In [105]:

```
df = df.cumsum()
```

```
df = df.cumsum()
```

In [106]:

```
plt.figure()
```

Out[106]:

```
<Figure size 432x288 with 0 Axes>
```

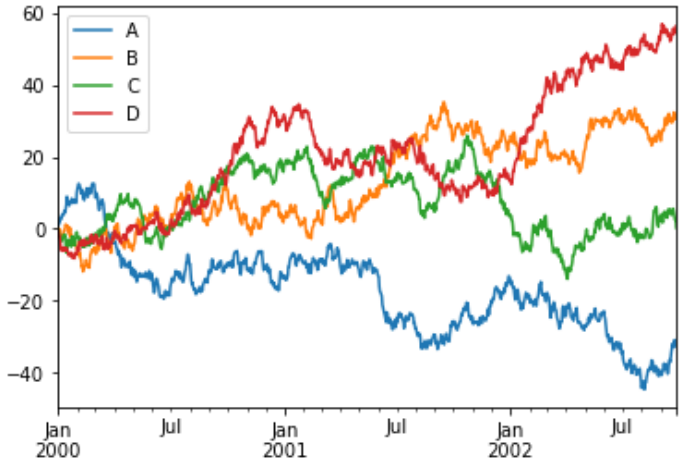
```
<Figure size 432x288 with 0 Axes>
```

In [107]:

```
df.plot()
```

Out[107]:

```
<AxesSubplot:>
```



Getting data in/out

In [108]:

```
df.to_csv("foo.csv")
```

HDF5

In [109]:

```
df.to_hdf("foo.h5", "df")
```

Excel

In [110]:

```
df.to_excel("foo.xlsx", sheet_name="Sheet1")
```

Gotchas

>>> if pd.Series([False, True, False]): ... print("I was true") Traceback ... ValueError: The truth value of an array is ambiguous. Use a.empty, a.any() or a.all().

Q2

In [111]:

```
import pandas as pd
```

```
data = {'cities' : ['lahore','karachi',], 'provinces' : ['punjab','sindh']}

# store data as DataFrame object. Assign object name as frame1
frame1 = pd.DataFrame(data)
print(frame1)

data2 = {"cities": ["islamabad","karachi","peshawar","quetta"],
"provinces": ["capital","sindh", "KPK","Balochistan"]}

# store data as DataFrame object. Assign object name as frame2
frame2 = pd.DataFrame(data2)
print(frame2)
print()

frame3 = pd.concat([frame1,frame2])
print(frame3)
print()

frame3.drop_duplicates(inplace=True)
print(frame3)
print()

frame3.sort_values('provinces', inplace=True)
print(frame3)
print()

frame3 = frame3.reset_index(drop=True)
print(frame3)
```

```
    cities provinces
0   lahore    punjab
1  karachi     sindh
      cities    provinces
0  islamabad      capital
1    karachi        sindh
2   peshawar          KPK
3     quetta  Balochistan

      cities    provinces
0     lahore       punjab
1    karachi        sindh
0  islamabad      capital
1    karachi        sindh
2   peshawar          KPK
3     quetta  Balochistan

      cities    provinces
0     lahore       punjab
1    karachi        sindh
0  islamabad      capital
2   peshawar          KPK
3     quetta  Balochistan

      cities    provinces
3     quetta  Balochistan
2   peshawar          KPK
0  islamabad      capital
0     lahore       punjab
1    karachi        sindh

      cities    provinces
0     quetta  Balochistan
1   peshawar          KPK
2  islamabad      capital
3     lahore       punjab
4    karachi        sindh
```

Q3

In [112]:

```
import numpy as np
dataset = pd.read_excel("data.xlsx")
dataset = dataset.drop(['Age'], axis=1)
dataset['Name'] = dataset['Name'].replace(np.nan, '--')

def assign_values(x):
    if x == 'C':
        return 0
    else:
        return 1

dataset['Field'] = dataset['Field'].apply(assign_values)
dataset['Marks'] = dataset['Marks'].where(dataset['Marks']>0, dataset['Marks'].mean()) # query to ask how it is working, when cond is wrong?
dataset.head()
```

Out[112]:

	Name	Field	Marks
0	--	0	21.0
1	Ali	1	60.0
2	Ahmed	1	21.0
3	Nida	0	70.0
4	--	0	75.0

Q4

In [135]:

```
#pandas and numpy already imported as pd and np
telecom = pd.read_csv('telecom_churn.csv')
print('Shape:')
print(telecom.shape)
print('\nGroup data by:')
print(telecom.groupby('churn').size())
print('\nColumns:')
print(telecom.columns)
print('\nInfo:')
print(telecom.info())
```

```
Shape:
(3333, 21)

Group data by:
churn
False    2850
True      483
dtype: int64

Columns:
Index(['state', 'account length', 'area code', 'phone number',
       'international plan', 'voice mail plan', 'number vmail messages',
       'total day minutes', 'total day calls', 'total day charge',
       'total eve minutes', 'total eve calls', 'total eve charge',
       'total night minutes', 'total night calls', 'total night charge',
       'total intl minutes', 'total intl calls', 'total intl charge',
       'customer service calls', 'churn'],
      dtype='object')

Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   state                   3333 non-null   object
```

```
 1   account length          3333 non-null   int64
 2   area code               3333 non-null   int64
 3   phone number            3333 non-null   object
 4   international plan      3333 non-null   object
 5   voice mail plan         3333 non-null   object
 6   number vmail messages   3333 non-null   int64
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64
 20  churn                   3333 non-null   bool
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
None
```

In [136]:

```
telecom['churn'] = telecom['churn'].astype('int64')
telecom.describe()
```

Out[136]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.980348	100.114311	17.083540
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713844	19.922625	4.310668
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600000	87.000000	14.160000
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400000	100.000000	17.120000
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300000	114.000000	20.000000
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700000	170.000000	30.910000

In [137]:

```
telecom.describe(include=['object', 'bool'])
```

Out[137]:

	state	phone number	international plan	voice mail plan
count	3333	3333	3333	3333
unique	51	3333	2	2
top	WV	379-8805	no	no
freq	106	1	3010	2411

In [138]:

```
telecom['churn'].value_counts()
```

Out[138]:

```
0    2850
```

```
1     483
Name: churn, dtype: int64
```

In [139]:

```
telecom['churn'].value_counts(normalize=True)
```

Out[139]:

```
0    0.855086
1    0.144914
Name: churn, dtype: float64
```

In [140]:

```
telecom.sort_values(by='total day charge', ascending=False).head()
```

Out[140]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls
365	CO	154	415	343-5709	no	no	0	350.8	75	59.64	...	94	18.40	253.9	100
985	NY	64	415	345-9140	yes	no	0	346.8	55	58.96	...	79	21.21	275.4	102
2594	OH	115	510	348-1163	yes	no	0	345.3	81	58.70	...	106	17.29	217.5	107
156	OH	83	415	370-9116	no	no	0	337.4	120	57.36	...	116	19.33	153.9	114
605	MO	112	415	373-2053	no	no	0	335.5	77	57.04	...	109	18.06	265.0	132

5 rows × 21 columns

In [141]:

```
telecom.sort_values(by=['churn', 'total day charge'], ascending=[True, False]).head()
```

Out[141]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls
688	MN	13	510	338-7120	no	yes	21	315.6	105	53.65	...	71	17.76	260.1	123
2259	NC	210	415	363-7802	no	yes	31	313.8	87	53.35	...	103	12.55	192.7	97
534	LA	67	510	373-6784	no	no	0	310.4	97	52.77	...	123	5.65	246.5	99
575	SD	114	415	351-7369	no	yes	36	309.9	90	52.68	...	89	17.03	183.5	105
2858	AL	141	510	388-8583	no	yes	28	308.0	123	52.36	...	128	21.06	152.9	103

5 rows × 21 columns

In [142]:

```
telecom['churn'].mean()
```

Out[142]:

```
0.14491449144914492
```

In [143]:

```
telecom[ telecom['churn']==1 ].mean()
```

Out[143]:

```
account length            102.664596
area code                 437.817805
number vmail messages       5.115942
total day minutes         206.914079
total day calls           101.335404
total day charge           35.175921
total eve minutes         212.410145
total eve calls           100.561077
total eve charge           18.054969
total night minutes       205.231677
total night calls         100.399586
total night charge          9.235528
total intl minutes         10.700000
total intl calls            4.163561
total intl charge           2.889545
customer service calls      2.229814
churn                       1.000000
dtype: float64
```

In [144]:

```
telecom[ telecom['churn']==1 ]['total day minutes'].mean()
```

Out[144]:

```
206.91407867494814
```

In [145]:

```
telecom[ (telecom['churn']==0) & (telecom['international plan']=='No') ]['total intl cha
rge'].max()
```

Out[145]:

```
nan
```

In [146]:

```
telecom.loc[0:5, 'state':'area code']
```

Out[146]:

	state	account length	area code
0	KS	128	415
1	OH	107	415
2	NJ	137	415
3	OH	84	408
4	OK	75	415
5	AL	118	510

In [147]:

```
telecom.iloc[0:5, 0:3]
```

Out[147]:

	state	account length	area code
0	KS	128	415
1	OH	107	415

	state	account length	area code
2	NJ	137	415
3	OH	84	408
4	OK	75	415

In [148]:

```
telecom[-1:] #last row
```

Out[148]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85	...	82	22.6	241.4	77

1 rows × 21 columns

In [149]:

```
telecom.apply(np.max)
```

Out[149]:

```
state                          WY
account length                243
area code                     510
phone number              422-9964
international plan             yes
voice mail plan                yes
number vmail messages           51
total day minutes            350.8
total day calls                165
total day charge             59.64
total eve minutes            363.7
total eve calls                170
total eve charge             30.91
total night minutes            395
total night calls              175
total night charge           17.77
total intl minutes              20
total intl calls                20
total intl charge              5.4
customer service calls            9
churn                             1
dtype: object
```

In [150]:

```
telecom[ telecom['state'].apply(lambda state: state[0] == 'W') ].head()
```

Out[150]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	c
9	WV	141	415	330-8173	yes	yes	37	258.6	84	43.96	...	111	18.87	326.4	97	
26	WY	57	408	357-3817	no	yes	39	213.0	115	36.21	...	112	16.24	182.7	115	
44	WI	64	510	352-1237	no	no	0	154.0	67	26.18	...	118	19.19	265.3	86	
49	WY	97	415	405-7146	no	yes	24	133.2	135	22.64	...	58	18.46	70.6	79	
54	WY	87	415	353-	no	no	0	151.0	83	25.67	...	116	18.67	203.9	127	

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	c
3759																

5 rows × 21 columns

In [152]:

```
d = { 'no': False, 'yes': True }
telecom['international plan'] = telecom['international plan'].map(d)
telecom.head()
```

Out[152]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	ch
0	KS	128	415	382-4657	False	yes	25	265.1	110	45.07	...	99	16.78	244.7	91	
1	OH	107	415	371-7191	False	yes	26	161.6	123	27.47	...	103	16.62	254.4	103	
2	NJ	137	415	358-1921	False	no	0	243.4	114	41.38	...	110	10.30	162.6	104	
3	OH	84	408	375-9999	True	no	0	299.4	71	50.90	...	88	5.26	196.9	89	
4	OK	75	415	330-6626	True	no	0	166.7	113	28.34	...	122	12.61	186.9	121	

5 rows × 21 columns

In [155]:

```
telecom = telecom.replace({'Voice mail plan': d}) #does the same thing according to kaggel tutorial however it is not working now
telecom.head()
```

Out[155]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	ch
0	KS	128	415	382-4657	False	yes	25	265.1	110	45.07	...	99	16.78	244.7	91	
1	OH	107	415	371-7191	False	yes	26	161.6	123	27.47	...	103	16.62	254.4	103	
2	NJ	137	415	358-1921	False	no	0	243.4	114	41.38	...	110	10.30	162.6	104	
3	OH	84	408	375-9999	True	no	0	299.4	71	50.90	...	88	5.26	196.9	89	
4	OK	75	415	330-6626	True	no	0	166.7	113	28.34	...	122	12.61	186.9	121	

5 rows × 21 columns

In [157]:

```
columns_to_show = ['total day minutes', 'total eve minutes',
                   'total night minutes']

telecom.groupby(['churn'])[columns_to_show].describe(percentiles=[])
```

Out[157]:

	total day minutes	total eve minutes	total night minutes

	total day minutes						total eve minutes						total night minutes	
	count	mean	std	min	50%	max	count	mean	std	min	50%	max	count	mean
churn														
0	2850.0	175.175754	50.181655	0.0	177.2	315.6	2850.0	199.043298	50.292175	0.0	199.6	361.8	2850.0	200.133193
1	483.0	206.914079	68.997792	0.0	217.6	350.8	483.0	212.410145	51.728910	70.9	211.3	363.7	483.0	205.231677

In [ ]: