jupyter Untitled3 Last Checkpoint: 11 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

In [1]:
```python
import numpy as np
import pandas as pd
```

```
C:\Users\Abhishek\AppData\Local\Temp\ipykernel_22536\1662815981.py:2: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

In [2]:
```python
dict1={
    "name":['harry','rohan','skillf','shubh'],
    "marks":[92,34,24,17],
    "city":['rampur','kolkata','bareilly','antarctica']
}
```

In [3]:
```python
df=pd.DataFrame(dict1)
```

In [4]:
```python
df
```

Out[4]:

	name	marks	city
0	harry	92	rampur
1	rohan	34	kolkata
2	skillf	24	bareilly
3	shubh	17	antarctica

In [5]:
```python
df.to_csv('friends.csv')
```

In [6]:
```python
df.to_csv('friends_index_false.csv',index=False)
```

In [7]:
```python
df.head(2)
```

Out[7]:

	name	marks	city
0	harry	92	rampur
1	rohan	34	kolkata

In [8]:
```python
df.tail(2)
```

Out[8]:

	name	marks	city
2	skillf	24	bareilly
3	shubh	17	antarctica

In [9]:
```python
df.describe()
```

Out[9]:

	marks
count	4.00000
mean	41.75000
std	34.21866
min	17.00000
25%	22.25000
50%	29.00000
75%	48.50000
max	92.00000

In [10]:
```python
harry=pd.read_csv('harry.csv')
```

In [11]:
```python
harry
```

Out[11]:

	Train No	Speed	city
0	12345	100	rampur
1	78945	200	kolkata
2	45678	220	bareilly
3	52478	230	antarctica

In [14]:
```python
harry['Speed'][0]
```

Out[14]:
```
100
```

In [15]:
```python
harry['Speed'][0]=120
```

```
C:\Users\Abhishek\AppData\Local\Temp\ipykernel_22536\4271634116.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  harry['Speed'][0]=120
```

In [16]:
```python
harry['Speed'][0]
```

Out[16]:
```
120
```

In [20]:
```python
harry=pd.read_csv('harry.csv')
```

In [21]:
```python
harry
```

Out[21]:

	Train No	Speed	city

0	12345	100	rampur
1	78945	200	kolkata
2	45678	220	bareilly
3	52478	230	antarctica

In [22]:
```
harry['Speed'][0]=120
```

```
C:\Users\Abhishek\AppData\Local\Temp\ipykernel_22536\4271634116.py:1: FutureWarning: ChainedAssignmentError: behaviour will cha
nge in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which w
ill become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Series, because the in
termediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating th
e original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy

  harry['Speed'][0]=120
C:\Users\Abhishek\AppData\Local\Temp\ipykernel_22536\4271634116.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  harry['Speed'][0]=120
```

In [23]:
```
harry
```

Out[23]:

	Train No	Speed	city
0	12345	120	rampur
1	78945	200	kolkata
2	45678	220	bareilly
3	52478	230	antarctica

In [28]:
```
harry.to_csv('harry.csv')
```

In [29]:
```
harry.index=['first','second','third','fourth']
```

In [30]:
```
harry
```

Out[30]:

	Train No	Speed	city
first	12345	120	rampur
second	78945	200	kolkata
third	45678	220	bareilly
fourth	52478	230	antarctica

In [31]:
```
ser=pd.Series(np.random.rand(34))
```

In [32]:
```
ser
```

Out[32]:
```
0     0.314280
1     0.951706
2     0.903970
3     0.071024
4     0.710963
5     0.161087
6     0.101986
7     0.025757
8     0.243269
9     0.765356
10    0.681155
11    0.269231
12    0.357446
13    0.158161
14    0.290139
15    0.893144
16    0.984160
17    0.728520
18    0.232100
19    0.665366
20    0.274816
21    0.048685
22    0.486903
23    0.511415
24    0.901930
25    0.918581
26    0.382966
27    0.955199
28    0.018173
29    0.943143
30    0.285417
31    0.927803
32    0.391800
33    0.668774
dtype: float64
```

In [33]:
```
type(ser)
```

Out[33]:
```
pandas.core.series.Series
```

In [34]:
```
newdf=pd.DataFrame(np.random.rand(334,5),index=np.arange(334))
```

In [40]:
```
newdf.head()
```

Out[40]:

	0	1	2	3	4
0	0.332980	0.419690	0.860466	0.131381	0.279653
1	0.520138	0.199037	0.058884	0.865166	0.334120
2	0.194305	0.449636	0.146657	0.894711	0.199855
3	0.239741	0.174097	0.191710	0.673072	0.920851
4	0.785714	0.515955	0.164822	0.005365	0.778152

In [41]:
```
type(newdf)
```
Out[41]:
```
pandas.core.frame.DataFrame
```

In [42]:
```
newdf.describe()
```
Out[42]:

	0	1	2	3	4
count	334.000000	334.000000	334.000000	334.000000	334.000000
mean	0.488561	0.477113	0.505140	0.491430	0.483567
std	0.291979	0.297852	0.278224	0.303216	0.281004
min	0.000334	0.001426	0.002549	0.000492	0.006181
25%	0.233336	0.213946	0.296732	0.212411	0.244945
50%	0.484440	0.465773	0.482432	0.466932	0.491775
75%	0.749785	0.739060	0.755414	0.768507	0.731354
max	0.997440	0.989554	0.996093	0.998616	0.994592

In [44]:
```
newdf.dtypes
```
Out[44]:
```
0    float64
1    float64
2    float64
3    float64
4    float64
dtype: object
```

In [45]:
```
newdf[0][0]="harry"
```
```
C:\Users\Abhishek\AppData\Local\Temp\ipykernel_22536\2518634006.py:1: FutureWarning: ChainedAssignmentError: behaviour will cha
nge in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which w
ill become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Series, because the in
termediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating th
e original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy

  newdf[0][0]="harry"
C:\Users\Abhishek\AppData\Local\Temp\ipykernel_22536\2518634006.py:1: FutureWarning: Setting an item of incompatible dtype is d
eprecated and will raise an error in a future version of pandas. Value 'harry' has dtype incompatible with float64, please expl
icitly cast to a compatible dtype first.
  newdf[0][0]="harry"
```

In [46]:
```
newdf.dtypes
```
Out[46]:
```
0     object
1    float64
2    float64
3    float64
4    float64
dtype: object
```

In [47]:
```
newdf.head()
```
Out[47]:

	0	1	2	3	4
0	harry	0.419690	0.860466	0.131381	0.279653
1	0.520138	0.199037	0.058884	0.865166	0.334120
2	0.194305	0.449636	0.146657	0.894711	0.199855
3	0.239741	0.174097	0.191710	0.673072	0.920851
4	0.785714	0.515955	0.164822	0.005365	0.778152

In [48]:
```
newdf.index
```
Out[48]:
```
Index([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
       ...
       324, 325, 326, 327, 328, 329, 330, 331, 332, 333],
      dtype='int32', length=334)
```

In [52]:
```
newdf.columns
```
Out[52]:
```
RangeIndex(start=0, stop=5, step=1)
```

In [53]:
```
newdf.to_numpy()
```
Out[53]:
```
array([['harry', 0.41968975654369955, 0.860466407674581,
        0.13138118214039696, 0.2796528920372212],
       [0.5201375975038302, 0.1990366225961293, 0.05888427473862945,
        0.8651656714140288, 0.3341200061159175],
       [0.19430528484260823, 0.4496364875150092, 0.1466566911945365,
        0.8947110333507035, 0.19985546997128856],
       ...,
       [0.05298353230683206, 0.07258161660256879, 0.37520546123484966,
        0.5557003512747687, 0.6123322953763801],
       [0.20332998660124668, 0.10872677354759497, 0.5268462589793043,
        0.20242414656522367, 0.026534401619536796],
       [0.48192838945860306, 0.9293651304808324, 0.8968708441557225,
        0.664588488620539, 0.7679244803115799]], dtype=object)
```

In [54]:
```
newdf.T
```
Out[54]:

0	1	2	3	4	5	6	7	8	9	...	324	325	326	327	328	329	
y	0.520138	0.194305	0.239741	0.785714	0.447313	0.233082	0.694409	0.611382	0.320119	...	0.943888	0.578999	0.889117	0.045451	0.889843	0.0433	0.71
9	0.199037	0.449636	0.174097	0.515955	0.355038	0.450019	0.012264	0.156608	0.269264	...	0.755422	0.956387	0.058337	0.743503	0.018452	0.392068	0.2
6	0.058884	0.146657	0.19171	0.164822	0.691534	0.325763	0.010695	0.942221	0.751166	...	0.339683	0.081592	0.918847	0.498893	0.370685	0.372763	0.51
1	0.865166	0.894711	0.673072	0.005365	0.36129	0.712781	0.378496	0.900944	0.173854	...	0.270312	0.335601	0.643991	0.409782	0.176526	0.693296	0.87
3	0.33412	0.199855	0.920851	0.778152	0.280756	0.540389	0.656002	0.107002	0.42368	...	0.981254	0.833369	0.552277	0.741208	0.08279	0.209604	0.73

34 columns

In [55]:
```
newdf.head()
```

Out[55]:

	0	1	2	3	4
0	harry	0.419690	0.860466	0.131381	0.279653
1	0.520138	0.199037	0.058884	0.865166	0.334120
2	0.194305	0.449636	0.146657	0.894711	0.199855
3	0.239741	0.174097	0.191710	0.673072	0.920851
4	0.785714	0.515955	0.164822	0.005365	0.778152

In [ ]: