Problem Statement

1) How-to-count-distance-to-the-previous-zero
For each value, count the difference of the distance from the previous zero (or the start of the Series, whichever is closer) and if there are no previous zeros,print the position
Consider a DataFrame df where there is an integer column {'X':[7, 2, 0, 3, 4, 2, 5, 0, 3, 4]}
The values should therefore be [1, 2, 0, 1, 2, 3, 4, 0, 1, 2]. Make this a new column 'Y'.
import pandas as pd
df = pd.DataFrame({'X': [7, 2, 0, 3, 4, 2, 5, 0, 3, 4]})

Solution 1

Source Code:

```
import pandas as pd

df = pd.DataFrame(({'X':[7,2,0,3,4,2,5,0,3,4]}))

df["Y"] = (df.X.groupby(df.X.eq(0).cumsum().mask(df.X.eq(0))).cumcount() + 1).mask(df.X.eq(0), 0).tolist()

print ("The desired output values are as follows :-\n",df)
```

Output Screenshot:

```
In [28]: import pandas as pd
         df = pd.DataFrame(({'X':[7,2,0,3,4,2,5,0,3,4]}))
         df["Y"] = (df.X.groupby(df.X.eq(0).cumsum().mask(df.X.eq(0))).cumcount() + 1).mask(df.X.eq(0), 0).tolist()
         print ("The desired output values are as follows :-\n",df)

         The desired output values are as follows :-
             X  Y
         0   7  1
         1   2  2
         2   0  0
         3   3  1
         4   4  2
         5   2  3
         6   5  4
         7   0  0
         8   3  1
         9   4  2
```

2) Create a DatetimeIndex that contains each business day of 2015 and use it to index a Series of random numbers.

Solution 2

Source Code:

```
import numpy as np

DT_Index = pd.date_range(start='2015-01-01', end='2015-12-31')

s = pd.Series(np.random.rand(len(DT_Index)), index=DT_Index)

print(s)
```

Output Screenshot:

```
In [33]: import numpy as np
         DT_Index = pd.date_range(start='2015-01-01', end='2015-12-31')
         s = pd.Series(np.random.rand(len(DT_Index)), index=DT_Index)
         print(s)
```

```
2015-01-01    0.299866
2015-01-02    0.738543
2015-01-03    0.715940
2015-01-04    0.805331
2015-01-05    0.175179
2015-01-06    0.260948
2015-01-07    0.975421
2015-01-08    0.326205
2015-01-09    0.349786
2015-01-10    0.875593
2015-01-11    0.956047
2015-01-12    0.131267
2015-01-13    0.057889
2015-01-14    0.008165
2015-01-15    0.933267
2015-01-16    0.836292
2015-01-17    0.834956
2015-01-18    0.998091
2015-01-19    0.784264
2015-01-20    0.372680
2015-01-21    0.511243
2015-01-22    0.666210
2015-01-23    0.258733
2015-01-24    0.468249
2015-01-25    0.035018
2015-01-26    0.870964
2015-01-27    0.471613
2015-01-28    0.293496
2015-01-29    0.205685
2015-01-30    0.787943
                ...
                ...
2015-12-02    0.305768
2015-12-03    0.794537
2015-12-04    0.156393
2015-12-05    0.080113
2015-12-06    0.173654
2015-12-07    0.624232
2015-12-08    0.091043
2015-12-09    0.408212
2015-12-10    0.619288
2015-12-11    0.192099
2015-12-12    0.938026
2015-12-13    0.575521
2015-12-14    0.382601
2015-12-15    0.597797
2015-12-16    0.359654
2015-12-17    0.690980
2015-12-18    0.934432
2015-12-19    0.091539
2015-12-20    0.695885
2015-12-21    0.053185
2015-12-22    0.832847
2015-12-23    0.574555
2015-12-24    0.257140
2015-12-25    0.303696
2015-12-26    0.369781
2015-12-27    0.047998
2015-12-28    0.952740
2015-12-29    0.816141
2015-12-30    0.499971
2015-12-31    0.769967
Freq: D, Length: 365, dtype: float64
```

3) Find the sum of the values in s for every Wednesday

Solution 3

Source Code:

s[DT_Index.weekday == 2].sum()

Output Screenshot:

```
In [34]: s[DT_Index.weekday == 2].sum()

Out[34]: 21.75716482773719
```

4) Average For each calendar mont

Solution 4

Source Code:

s.resample('M').mean()

Output Screenshot:

```
In [36]: s.resample('M').mean()

Out[36]: 2015-01-31    0.529040
         2015-02-28    0.530022
         2015-03-31    0.457856
         2015-04-30    0.498447
         2015-05-31    0.504872
         2015-06-30    0.543696
         2015-07-31    0.496984
         2015-08-31    0.447131
         2015-09-30    0.442307
         2015-10-31    0.534838
         2015-11-30    0.459339
         2015-12-31    0.483584
         Freq: M, dtype: float64
```

5) For each group of four consecutive calendar months in s, find the date on which the highest value occurred.

Solution 5:

Source Code:

s.groupby(pd.TimeGrouper('4M')).idxmax()

Output Screenshot:

```
In [37]: s.groupby(pd.TimeGrouper('4M')).idxmax()

C:\Users\Amir\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureWarning: pd.TimeGrouper is deprecated and wi
ll be removed; Please use pd.Grouper(freq=...)
  """Entry point for launching an IPython kernel.

Out[37]: 2015-01-31   2015-01-18
         2015-05-31   2015-02-13
         2015-09-30   2015-06-20
         2016-01-31   2015-11-08
         dtype: datetime64[ns]
```