FIN 580 Homework 2 - Tejas Dhomne | UIN - 661586178

Import pandas_datareader as pdr

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

Question 1

1.1) Create a new column in df1 named month1 that extracts month information from the index.

In [2]:
```
#1.1
df1 = pdr.get_data_yahoo('GOOGL', start='2020-01-01',end='2020-12-31')
df1
```

Out[2]:

	High	Low	Open	Close	Volume	Adj Close
Date						
2020-01-02	68.433998	67.324501	67.420502	68.433998	27278000	68.433998
2020-01-03	68.687500	67.365997	67.400002	68.075996	23408000	68.075996
2020-01-06	69.916000	67.550003	67.581497	69.890503	46768000	69.890503
2020-01-07	70.175003	69.578003	70.023003	69.755501	34330000	69.755501
2020-01-08	70.592499	69.631500	69.740997	70.251999	35314000	70.251999
...	...	...	...	...	...	...
2020-12-24	87.120499	86.217499	86.449997	86.708000	9312000	86.708000
2020-12-28	89.349998	87.091003	87.245499	88.697998	27650000	88.697998
2020-12-29	89.423500	87.755501	89.361504	87.888000	19726000	87.888000
2020-12-30	88.388000	86.400002	88.250000	86.812500	21026000	86.812500
2020-12-31	87.875000	86.804497	86.863503	87.632004	21070000	87.632004

253 rows × 6 columns

In [3]:
```
df1['month1']=df1.index.month
df1
```

Out[3]:

	High	Low	Open	Close	Volume	Adj Close	month1
Date							
2020-01-02	68.433998	67.324501	67.420502	68.433998	27278000	68.433998	1
2020-01-03	68.687500	67.365997	67.400002	68.075996	23408000	68.075996	1
2020-01-06	69.916000	67.550003	67.581497	69.890503	46768000	69.890503	1
2020-01-07	70.175003	69.578003	70.023003	69.755501	34330000	69.755501	1
2020-01-08	70.592499	69.631500	69.740997	70.251999	35314000	70.251999	1
...	...	...	...	...	...	...	...
2020-12-24	87.120499	86.217499	86.449997	86.708000	9312000	86.708000	12
2020-12-28	89.349998	87.091003	87.245499	88.697998	27650000	88.697998	12
2020-12-29	89.423500	87.755501	89.361504	87.888000	19726000	87.888000	12
2020-12-30	88.388000	86.400002	88.250000	86.812500	21026000	86.812500	12

1.2) Use replace() to create a new column in df1 named month2 that replaces integers in month1 with abbreviations of the names of the months. For example, replace 1 with Jan, 2 with Feb, 3 with Mar, 4 with Apr, 5 with May, 6 with Jun, 7 with Jul, 8 with Aug, 9 with Sep, 10 with Oct, 11 with Nov, and 12 with Dec. Do not modify the numeric values in month1.

In [4]:
```
:h1"].replace({1:"Jan",2:"Feb",3:"Mar",4:"Apr",5:"May",6:"Jun",7:"Jul",8:"Aug",9:'
```

Out[4]:

	High	Low	Open	Close	Volume	Adj Close	month1	month2
Date								
2020-01-02	68.433998	67.324501	67.420502	68.433998	27278000	68.433998	1	Jan
2020-01-03	68.687500	67.365997	67.400002	68.075996	23408000	68.075996	1	Jan
2020-01-06	69.916000	67.550003	67.581497	69.890503	46768000	69.890503	1	Jan
2020-01-07	70.175003	69.578003	70.023003	69.755501	34330000	69.755501	1	Jan
2020-01-08	70.592499	69.631500	69.740997	70.251999	35314000	70.251999	1	Jan
...	...	...	...	...	...	...	...	...
2020-12-24	87.120499	86.217499	86.449997	86.708000	9312000	86.708000	12	Dec
2020-12-28	89.349998	87.091003	87.245499	88.697998	27650000	88.697998	12	Dec
2020-12-29	89.423500	87.755501	89.361504	87.888000	19726000	87.888000	12	Dec
2020-12-30	88.388000	86.400002	88.250000	86.812500	21026000	86.812500	12	Dec
2020-12-31	87.875000	86.804497	86.863503	87.632004	21070000	87.632004	12	Dec

253 rows × 8 columns

1.3) Return the counts of unique values in month2 and sort the series of value counts by the counts in an ascending order.

In [5]:
```
#1.3
df1["month2"].value_counts().sort_values()
```

Out[5]:
```
Feb    19
May    20
Nov    20
Jan    21
Apr    21
Aug    21
Sep    21
Mar    22
Jun    22
Jul    22
Oct    22
Dec    22
Name: month2, dtype: int64
```

1.4) Create a new column in df1 named high_minus_low1 that represents the difference between High and Low.

In [6]:
```
df1['high_minus_low1']=df1["High"]-df1["Low"]
df1
```

Date								
2020-01-02	68.433998	67.324501	67.420502	68.433998	27278000	68.433998	1	Jan
2020-01-03	68.687500	67.365997	67.400002	68.075996	23408000	68.075996	1	Jan
2020-01-06	69.916000	67.550003	67.581497	69.890503	46768000	69.890503	1	Jan
2020-01-07	70.175003	69.578003	70.023003	69.755501	34330000	69.755501	1	Jan
2020-01-08	70.592499	69.631500	69.740997	70.251999	35314000	70.251999	1	Jan
...	...	...	...	...	...	...	...	...
2020-12-24	87.120499	86.217499	86.449997	86.708000	9312000	86.708000	12	Dec
2020-12-28	89.349998	87.091003	87.245499	88.697998	27650000	88.697998	12	Dec

1.5) Create a function named dollars with one parameter. The function rounds the argument to two decimal places, converts the number to a string, and adds the string to “ dollars”. After creating the function, dollars(22.190063) should return ‘22.19 dollars’.

In [7]:
```
#1.5
def dollars(x):
    return str(round(x,2))+" dollars"
```

In [8]:
```
dollars(22.190063)
```

Out[8]:
```
'22.19 dollars'
```

1.6) Apply the dollars function to all the values in the high_minus_low1 column. Create a new column in df1 named high_minus_low2 to store the result. Do not modify the values in high_minus_low1.

In [9]:
```
#1.6
df1['high_minus_low2']=df1["high_minus_low1"].apply(dollars)
df1
```

Date								
2020-01-02	68.433998	67.324501	67.420502	68.433998	27278000	68.433998	1	Jan
2020-01-03	68.687500	67.365997	67.400002	68.075996	23408000	68.075996	1	Jan
2020-01-06	69.916000	67.550003	67.581497	69.890503	46768000	69.890503	1	Jan
2020-01-07	70.175003	69.578003	70.023003	69.755501	34330000	69.755501	1	Jan
2020-01-08	70.592499	69.631500	69.740997	70.251999	35314000	70.251999	1	Jan
...	...	...	...	...	...	...	...	...
2020-12-24	87.120499	86.217499	86.449997	86.708000	9312000	86.708000	12	Dec
2020-12-28	89.349998	87.091003	87.245499	88.697998	27650000	88.697998	12	Dec

1.7) Use a lambda function to combine steps used in questions 1.5 and 1.6 into one step. Create a column in df1 named high_minus_low3 to store the result. Do not modify the values in high_minus_low1.

In [10]:
```
#1.7
df1['high_minus_low3']=df1["high_minus_low1"].apply(lambda x: str(round(x,2)) +"
df1
```

Out[10]:

	High	Low	Open	Close	Volume	Adj Close	month1	month2	high_
Date									
2020-01-02	68.433998	67.324501	67.420502	68.433998	27278000	68.433998	1	Jan	
2020-01-03	68.687500	67.365997	67.400002	68.075996	23408000	68.075996	1	Jan	
2020-01-06	69.916000	67.550003	67.581497	69.890503	46768000	69.890503	1	Jan	
2020-01-07	70.175003	69.578003	70.023003	69.755501	34330000	69.755501	1	Jan	
2020-01-08	70.592499	69.631500	69.740997	70.251999	35314000	70.251999	1	Jan	
...	...	...	...	...	...	...	...	...	
2020-	87.120499	86.217499	86.449997	86.708000	9312000	86.708000	12	Dec	

1.8) Return a row from df1 if high_minus_low2 is not equal to high_minus_low3. Since the values in high_minus_low2 and high_minus_low3 are the same, it should return an empty dataframe.

In [11]:
```
#1.8
df1[df1["high_minus_low2"]!=df1["high_minus_low3"]]
```

Out[11]:

	High	Low	Open	Close	Volume	Adj Close	month1	month2	high_minus_low1	high_minus_lo
Date										

1.9) Use drop(labels=, axis=, inplace=) to drop the "High", "Low", "high_minus_low1", "high_minus_low2", "high_minus_low3" columns from df1 and modify df1 in place. Specify the column names in a list using the labels parameter. Use axis=”columns” to drop columns and inplace=True to modify df1 in place.

In [12]:
```
#1.9
df1.drop(labels=["High","Low","high_minus_low1","high_minus_low2","high_minus_lo
df1
```

Out[12]:

	Open	Close	Volume	Adj Close	month1	month2
Date						
2020-01-02	67.420502	68.433998	27278000	68.433998	1	Jan
2020-01-03	67.400002	68.075996	23408000	68.075996	1	Jan
2020-01-06	67.581497	69.890503	46768000	69.890503	1	Jan
2020-01-07	70.023003	69.755501	34330000	69.755501	1	Jan
2020-01-08	69.740997	70.251999	35314000	70.251999	1	Jan
...	...	...	...	...	...	...
2020-12-24	86.449997	86.708000	9312000	86.708000	12	Dec
2020-12-28	87.245499	88.697998	27650000	88.697998	12	Dec
2020-12-29	89.361504	87.888000	19726000	87.888000	12	Dec
2020-12-30	88.250000	86.812500	21026000	86.812500	12	Dec

1.10) Use cut() to create a new column in df1 named price_bins to convert continuous values of the Adj Close column to four bins with an equal number of observations. Use the labels parameter to specify the following labels in order: low, moderate, high, and very high. Use include_lowest=True to specify that the first interval should be left-inclusive.

In [13]:
```
#1.10
l1=[0, 0.25, 0.5, 0.75,1]
l2=[]
```

In [14]:
```
#1.10
for i in l1:
    l2.append(df1["Adj Close"].quantile(i))
l2
```

Out[14]:
```
[52.70650100708008,
 69.8584976196289,
 73.95549774169922,
 78.22949981689453,
 91.24849700927734]
```

In [15]:
```
#1.10
df1['price_bins']=pd.cut(df1["Adj Close"],l2,labels=["low","moderate","high","ver
df1
```

Date							
2020-01-02	67.420502	68.433998	27278000	68.433998	1	Jan	low
2020-01-03	67.400002	68.075996	23408000	68.075996	1	Jan	low
2020-01-06	67.581497	69.890503	46768000	69.890503	1	Jan	moderate
2020-01-07	70.023003	69.755501	34330000	69.755501	1	Jan	low
2020-01-08	69.740997	70.251999	35314000	70.251999	1	Jan	moderate
...	...	...	...	...	...	...	...
2020-12-24	86.449997	86.708000	9312000	86.708000	12	Dec	very high
2020-12-28	87.245499	88.697998	27650000	88.697998	12	Dec	very high
2020-12-29	89.361504	87.888000	19726000	87.888000	12	Dec	very high
2020-12-30	88.250000	86.812500	21026000	86.812500	12	Dec	very high
2020-12-31	86.863503	87.632004	21070000	87.632004	12	Dec	very high

253 rows × 7 columns

1.11) Return the counts of unique values in price_bins and sort the series of value counts by the index.

In [16]:
```
#1.11
df1["price_bins"].value_counts().sort_index()
```

Out[16]:
```
low          64
moderate     63
high         63
very high    63
Name: price_bins, dtype: int64
```

1.12) Return a row from df1 if the value in the price_bins column is null. It should return an empty dataframe.

In [17]:
```
#1.12
df1[df1["price_bins"].isnull()]
```

Out[17]:

	Open	Close	Volume	Adj Close	month1	month2	price_bins
Date							

1.13) Use crosstab() to return two-way frequency counts of month1 and price_bins. Use margins=True to add row and column subtotals.

In [18]:
```
#1.13
pd.crosstab(df1["month1"],df1["price_bins"],margins=True)
```

Out[18]:

price_bins	low	moderate	high	very high	All
month1					
1	3	14	4	0	21
2	4	5	10	0	19
3	22	0	0	0	22
4	21	0	0	0	21
5	12	8	0	0	20
6	2	20	0	0	22
7	0	2	19	1	22
8	0	2	11	8	21
9	0	9	8	4	21
10	0	3	11	8	22
11	0	0	0	20	20
12	0	0	0	22	22
All	64	63	63	63	253

#1.13 - December month in 2020 has the greatest number of adjusted closing prices that fall in the very high category.

Question 2

2.1) Set the random seed value to 0 and use np.random.randint() to generate 100 numbers from a uniform distribution over [-50, 50). Convert the data type to float and save the result in a NumPy array named a1.

In [19]:
```
#2.1
np.random.seed(0)
a1=np.random.randint(-50,50,100).astype('float')
a1
```

Out[19]:
```
array([ -6.,  -3.,  14.,  17.,  17., -41.,  33., -29., -14.,  37.,  20.,
        38.,  38., -38.,   8.,  15., -11.,  37.,  -4.,  38.,  31., -13.,
       -25.,  27.,  22., -41., -30.,  30.,  19.,  29.,  -3.,  14.,  32.,
        49.,  38.,  -1., -21., -31., -31., -36., -11., -18.,  15., -41.,
         7., -18., -19.,  24., -27., -15.,  25.,   5., -22., -16., -50.,
       -50., -14.,   3., -45., -12., -33.,  29., -46.,  -8.,   8., -19.,
       -49.,  15.,  -9.,   7., -15., -39.,  -4.,  32.,  41., -50., -36.,
        49.,   3., -38.,  -8.,  34.,  25.,  18., -44.,  18.,  -3., -47.,
        26.,   2.,  28., -35., -30.,  49.,   8., -27.,  29., -37.,  35.,
        -2.])
```

2.2) Set the random seed value to 0 and use np.ramdom.choice() to create a NumPy array named a2 and populate it with 20 integers randomly selected from [0, 100) without replacement. Use the replace=False parameter to sample without replacement.

In [20]:
```
#2.2
np.random.seed(0)
a2=np.random.choice(np.arange(0,100), size=20, replace=False)
a2
```

Out[20]:
```
array([26, 86,  2, 55, 75, 93, 16, 73, 54, 95, 53, 92, 78, 13,  7, 30, 22,
       24, 33,  8])
```

2.3) Replace numbers in a1 at indices a2 with missing values.

In [21]:
```
#2.3
a1[a2]=np.nan
a1
```

Out[21]:
```
array([ -6.,  -3.,  nan,  17.,  17., -41.,  33.,  nan,  nan,  37.,  20.,
        38.,  38.,  nan,   8.,  15.,  nan,  37.,  -4.,  38.,  31., -13.,
        nan,  27.,  nan, -41.,  nan,  30.,  19.,  29.,  nan,  14.,  32.,
        nan,  38.,  -1., -21., -31., -31., -36., -11., -18.,  15., -41.,
         7., -18., -19.,  24., -27., -15.,  25.,   5., -22.,  nan,  nan,
        nan, -14.,   3., -45., -12., -33.,  29., -46.,  -8.,   8., -19.,
       -49.,  15.,  -9.,   7., -15., -39.,  -4.,  nan,  41.,  nan, -36.,
        49.,  nan, -38.,  -8.,  34.,  25.,  18., -44.,  18.,  nan, -47.,
        26.,   2.,  28., -35.,  nan,  nan,   8.,  nan,  29., -37.,  35.,
        -2.])
```

2.4) Use reshape() to convert a1 from a 100-element vector to a 25*4 matrix.

```
In [22]: #2.4
         a1=a1.reshape(25,4)
         a1
```

```
       [ nan, -41.,  nan,  30.],
       [ 19.,  29.,  nan,  14.],
       [ 32.,  nan,  38.,  -1.],
       [-21., -31., -31., -36.],
       [-11., -18.,  15., -41.],
       [  7., -18., -19.,  24.],
       [-27., -15.,  25.,   5.],
       [-22.,  nan,  nan,  nan],
       [-14.,   3., -45., -12.],
       [-33.,  29., -46.,  -8.],
       [  8., -19., -49.,  15.],
       [ -9.,   7., -15., -39.],
       [ -4.,  nan,  41.,  nan],
       [-36.,  49.,  nan, -38.],
       [ -8.,  34.,  25.,  18.],
       [-44.,  18.,  nan, -47.],
       [ 26.,   2.,  28., -35.],
       [ nan,  nan,   8.,  nan],
       [ 29., -37.,  35.,  -2.]])
```

2.5) Convert a1 to a dataframe named df2 and name the columns a, b, c, and d.

```
In [23]: #2.5
         df2=pd.DataFrame(data=a1,columns=["a","b","c","d"])
         df2
```

```
12  -27.0  -15.0   25.0    5.0
13  -22.0    NaN    NaN    NaN
14  -14.0    3.0  -45.0  -12.0


15  -33.0   29.0  -46.0   -8.0
16    8.0  -19.0  -49.0   15.0
17   -9.0    7.0  -15.0  -39.0
18   -4.0    NaN   41.0    NaN
19  -36.0   49.0    NaN  -38.0
20   -8.0   34.0   25.0   18.0
21  -44.0   18.0    NaN  -47.0
22   26.0    2.0   28.0  -35.0
23    NaN    NaN    8.0    NaN
```

2.6) Use mean() to return the mean value of each row. Use the axis=”columns” parameter to calculate the mean value across columns.

In [24]:
```
#2.6
df2.mean(axis="columns")
```

Out[24]:
```
0      2.666667
1      3.000000
2     31.666667
3     20.333333
4     23.666667
5     15.000000
6     -5.500000
7     20.666667
8     23.000000
9    -29.750000
10   -13.750000
11    -1.500000
12    -3.000000
13   -22.000000
14   -17.000000
15   -14.500000
16   -11.250000
17   -14.000000
18    18.500000
```

2.7) Create a new column in df2 named group that takes on a value of 1 if the row mean is positive, and it takes on a value of 2 if the row mean is not positive.

In [25]:
```
#2.7
df2['group']=np.where(df2.mean(axis="columns")>0,"1","2")
df2
```

Out[25]:

	a	b	c	d	group
0	-6.0	-3.0	NaN	17.0	1
1	17.0	-41.0	33.0	NaN	1
2	NaN	37.0	20.0	38.0	1
3	38.0	NaN	8.0	15.0	1
4	NaN	37.0	-4.0	38.0	1
5	31.0	-13.0	NaN	27.0	1
6	NaN	-41.0	NaN	30.0	2
7	19.0	29.0	NaN	14.0	1
8	32.0	NaN	38.0	-1.0	1
9	-21.0	-31.0	-31.0	-36.0	2
10	-11.0	-18.0	15.0	-41.0	2

2.8) Use groupby() to group df2 by the group column and use size() to calculate the group siz

In [26]:
```
#2.8
df2.groupby("group").size()
```

Out[26]:
```
group
1    13
2    12
dtype: int64
```

2.9) Use notnull() and sum() to calculate the number of non-missing values in each column in df2.

In [27]:
```
#2.9
df2.notnull().sum()
```

Out[27]:
```
a        21
b        20
c        18
d        21
group    25
dtype: int64
```

2.10) Filter and return rows in df2 that have missing values in column c.

In [28]:
```
#2.10
df2[df2["c"].isnull()]
```

Out[28]:

	a	b	c	d	group
0	-6.0	-3.0	NaN	17.0	1
5	31.0	-13.0	NaN	27.0	1
6	NaN	-41.0	NaN	30.0	2
7	19.0	29.0	NaN	14.0	1
13	-22.0	NaN	NaN	NaN	2
19	-36.0	49.0	NaN	-38.0	2
21	-44.0	18.0	NaN	-47.0	2

2.11) Use dropna() and the subset parameter to remove rows in df2 that have missing values in column c. Keep inplace=False by default to not modify df2 in place.

In [29]:
```
#2.11
df2.dropna(subset=["c"])
```

```
 9  -21.0  -31.0  -31.0  -36.0     2
10  -11.0  -18.0   15.0  -41.0     2
11    7.0  -18.0  -19.0   24.0     2
12  -27.0  -15.0   25.0    5.0     2
14  -14.0    3.0  -45.0  -12.0     2
15  -33.0   29.0  -46.0   -8.0     2
16    8.0  -19.0  -49.0   15.0     2
17   -9.0    7.0  -15.0  -39.0     2
18   -4.0    NaN   41.0    NaN     1
20   -8.0   34.0   25.0   18.0     1
22   26.0    2.0   28.0  -35.0     1
23    NaN    NaN    8.0    NaN     1
24   29.0  -37.0   35.0   -2.0     1
```

2.12) Use dropna() and the thresh parameter to keep rowsin df2 that have at least 4 non-missing values. Keep inplace=False by default to not modify df2 in place.

In [30]:
```
#2.12
df2.dropna(thresh=4, inplace=False)
```

```
 9  -21.0  -31.0  -31.0  -36.0     2
10  -11.0  -18.0   15.0  -41.0     2
11    7.0  -18.0  -19.0   24.0     2
12  -27.0  -15.0   25.0    5.0     2
14  -14.0    3.0  -45.0  -12.0     2
15  -33.0   29.0  -46.0   -8.0     2
16    8.0  -19.0  -49.0   15.0     2
17   -9.0    7.0  -15.0  -39.0     2


19  -36.0   49.0    NaN  -38.0     2
20   -8.0   34.0   25.0   18.0     1
21  -44.0   18.0    NaN  -47.0     2
22   26.0    2.0   28.0  -35.0     1
```

2.13) Calculate the median value of each column in df2. Use fillna() to fill missing values in df2 with the column median. Keep inplace=False by default to not modify df2 in place.

In [31]:
```
#2.13
df2.fillna(df2.median(),inplace=False)
```

Out[31]:

	a	b	c	d	group
0	-6.0	-3.0	11.5	17.0	1
1	17.0	-41.0	33.0	5.0	1
2	-6.0	37.0	20.0	38.0	1
3	38.0	-0.5	8.0	15.0	1
4	-6.0	37.0	-4.0	38.0	1
5	31.0	-13.0	11.5	27.0	1
6	-6.0	-41.0	11.5	30.0	2
7	19.0	29.0	11.5	14.0	1
8	32.0	-0.5	38.0	-1.0	1
9	-21.0	-31.0	-31.0	-36.0	2
10	-11.0	-18.0	15.0	-41.0	2

2.14) Calculate the median value of each column in df2. Use fillna() to fill missing values in df2 with the column median. Keep inplace=False by default to not modify df2 in place.

In [32]:
```
#2.14
df2.groupby("group").median()
```

Out[32]:

	a	b	c	d
group				
1	22.5	2.0	26.5	16.0
2	-21.0	-15.0	-25.0	-12.0

2.15) This question aims to fill missing values in df2 with the group median. Use groupby() and apply() to group df2 by the group column and apply a lambda function to all the groups. Use fillna() and median() in the lambda function to replace missing values with the group median. Take column a in df2 as an example, we want to replace missing values in group 1 with 22.5, and missing values in group 2 with -21. Keep inplace=False by default to not modify df2 in place

In [33]:
```
#2.15
df2.groupby("group").apply(lambda x:x.fillna(x.median(),inplace=False))
```

Out[33]:

	a	b	c	d	group
0	-6.0	-3.0	26.5	17.0	1
1	17.0	-41.0	33.0	16.0	1
2	22.5	37.0	20.0	38.0	1
3	38.0	2.0	8.0	15.0	1
4	22.5	37.0	-4.0	38.0	1
5	31.0	-13.0	26.5	27.0	1
6	-21.0	-41.0	-25.0	30.0	2
7	19.0	29.0	26.5	14.0	1
8	32.0	2.0	38.0	-1.0	1
9	-21.0	-31.0	-31.0	-36.0	2
10	-11.0	-18.0	15.0	-41.0	2
11	7.0	-18.0	-19.0	24.0	2
12	-27.0	-15.0	25.0	5.0	2
13	-22.0	-15.0	-25.0	-12.0	2
14	-14.0	3.0	-45.0	-12.0	2
15	-33.0	29.0	-46.0	-8.0	2
16	8.0	-19.0	-49.0	15.0	2
17	-9.0	7.0	-15.0	-39.0	2
18	-4.0	2.0	41.0	16.0	1
19	-36.0	49.0	-25.0	-38.0	2
20	-8.0	34.0	25.0	18.0	1
21	-44.0	18.0	-25.0	-47.0	2
22	26.0	2.0	28.0	-35.0	1
23	22.5	2.0	8.0	16.0	1
24	29.0	-37.0	35.0	-2.0	1