FIN 580 Homework 1 - Tejas Dhomne UIN:661586178

Import Packages

In [1]:
```
import numpy as np
```

1.1 Use list() and range() to create a list named l1 that stores the following even integers.

In [2]:
```
#1.1  Use list() and range() to create a list named l1 that stores the following
l1=list(range(0,20,2))
l1
```

Out[2]:
```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

1.2 Method 1: perform an element-wise array operation

In [3]:
```
#1.2 Method 1: perform an element-wise array operation
a1=np.array(l1)
(a1**3).tolist()
```

Out[3]:
```
[0, 8, 64, 216, 512, 1000, 1728, 2744, 4096, 5832]
```

1.3 Method 2: use a for loop and append elements to a list named l2

In [4]:
```
#1.3 Method 2: use a for loop and append elements to a list named l2
l2=[]
for i in l1:
    l2.append(i**3)
l2
```

Out[4]:
```
[0, 8, 64, 216, 512, 1000, 1728, 2744, 4096, 5832]
```

2.1 Use list() and range() to create a list named l3 that stores the following integers.

In [5]:
```
#2.1 Use list() and range() to create a list named l3 that stores the following i
l3=list(range(0,20))
l3
```

Out[5]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

2.2 Method 1: use a list comprehension

In [6]:
```
#2.2 Method 1: use a list comprehension
[i**3 for i in l3 if i%2==0]
```

Out[6]:
```
[0, 8, 64, 216, 512, 1000, 1728, 2744, 4096, 5832]
```

2.3 Define a function named even with one parameter x

In [7]:
```
#2.3 Define a function named even with one parameter x
def even(x):
    return x%2==0
list(map(even,l3))
```

Out[7]:
```
[True,
 False,
 True,
 False,
 True,
 False,
 True,
 False,
 True,
 False,
 True,
 False,
 True,
 False,
 True,
 False,
 True,
 False,
 True,
 False,
 True,
 False]
```

2.4 Define a function named even with one parameter x

In [8]:
```
#2.4 Define a function named even with one parameter x
[i for i in l3 if even(i)]
```

Out[8]:
```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

2.5 Use a filter function to select even numbers in l3

In [9]:
```
#2.5 Use a filter function to select even numbers in l3
list(filter(even,l3))
```

Out[9]:
```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

2.6 Define a function named cube with one parameter x

In [10]:
```
#2.6 Define a function named cube with one parameter x
def cube(x):
    return x**3
```

2.7 Method 3: use map and filter functions

In [11]:
```
#2.7 Method 3: use map and filter functions
list(filter(even,map(cube,13)))
```

Out[11]:
```
[0, 8, 64, 216, 512, 1000, 1728, 2744, 4096, 5832]
```

2.8 Method 4: use map, filter, and lambda functions

In [12]:
```
#2.8 Method 4: use map, filter, and lambda functions
list(filter(lambda x: x%2==0,map(lambda x: x**3,13)))
```

Out[12]:
```
[0, 8, 64, 216, 512, 1000, 1728, 2744, 4096, 5832]
```

3 Import pandas_datareader as pdr. Use pdr.get_data_yahoo to obtain Google's stock information (High, Low, Open, Close, Volume, and Adj Close) from 01/01/2016 to 12/31/2020. The Ticker of Google's common stock is GOOGL. Save the stock information in a pandas DataFrame named df1.

In [13]:
```
#3
import pandas as pd
import pandas_datareader as pdr
df1 = pdr.get_data_yahoo('GOOGL', start='2016-01-01',end='2020-12-31')
df1
```

Out[13]:

Date	High	Low	Open	Close	Volume	Adj Close
2016-01-04	38.110001	37.376999	38.110001	37.972000	67382000	37.972000
2016-01-05	38.459999	37.782501	38.205002	38.076500	45216000	38.076500
2016-01-06	38.286499	37.400002	37.518501	37.966499	48206000	37.966499
2016-01-07	37.765499	36.764000	37.324501	37.049999	63132000	37.049999
2016-01-08	37.506001	36.445999	37.389999	36.545502	47506000	36.545502
...	...	...	...	...	...	...
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

1259 rows × 6 columns

3.1 Use drop(labels=, axis=, inplace=) to drop the Volume column from df1.

In [14]:
```
#3.1 Use drop(labels=, axis=, inplace=) to drop the Volume column from df1.
df1.drop("Volume",axis="columns",inplace=True)
df1
```

Out[14]:

Date	High	Low	Open	Close	Adj Close
2016-01-04	38.110001	37.376999	38.110001	37.972000	37.972000
2016-01-05	38.459999	37.782501	38.205002	38.076500	38.076500
2016-01-06	38.286499	37.400002	37.518501	37.966499	37.966499
2016-01-07	37.765499	36.764000	37.324501	37.049999	37.049999
2016-01-08	37.506001	36.445999	37.389999	36.545502	36.545502
...	...	...	...	...	...
2020-12-24	87.120499	86.217499	86.449997	86.708000	86.708000
2020-12-28	89.349998	87.091003	87.245499	88.697998	88.697998
2020-12-29	89.423500	87.755501	89.361504	87.888000	87.888000
2020-12-30	88.388000	86.400002	88.250000	86.812500	86.812500
2020-12-31	87.875000	86.804497	86.863503	87.632004	87.632004

1259 rows × 5 columns

3.2 Create three new columns in df1 that extract year, month, and day information from the index.

In [15]:
```
#3.2 Create three new columns in df1 that extract year, month, and day informatic
df1['year']=df1.index.year
df1['month']=df1.index.month
df1['day']=df1.index.day
df1
```

Out[15]:

Date	High	Low	Open	Close	Adj Close	year	month	day
2016-01-04	38.110001	37.376999	38.110001	37.972000	37.972000	2016	1	4
2016-01-05	38.459999	37.782501	38.205002	38.076500	38.076500	2016	1	5
2016-01-06	38.286499	37.400002	37.518501	37.966499	37.966499	2016	1	6
2016-01-07	37.765499	36.764000	37.324501	37.049999	37.049999	2016	1	7
2016-01-08	37.506001	36.445999	37.389999	36.545502	36.545502	2016	1	8
...	...	...	...	...	...	...	...	...
2020-12-24	87.120499	86.217499	86.449997	86.708000	86.708000	2020	12	24
2020-12-28	89.349998	87.091003	87.245499	88.697998	88.697998	2020	12	28
2020-12-29	89.423500	87.755501	89.361504	87.888000	87.888000	2020	12	29
2020-12-30	88.388000	86.400002	88.250000	86.812500	86.812500	2020	12	30
2020-12-31	87.875000	86.804497	86.863503	87.632004	87.632004	2020	12	31

1259 rows × 8 columns

3.3 Use df1.set_index() to create a new dataframe named df2 that has three index levels.

In [16]:
```
#3.3 Use df1.set_index() to create a new dataframe named df2 that has three index
df2=df1.set_index(keys=["year","month","day"])
df2
```

Out[16]:

year	month	day	High	Low	Open	Close	Adj Close
2016	1	4	38.110001	37.376999	38.110001	37.972000	37.972000
		5	38.459999	37.782501	38.205002	38.076500	38.076500
		6	38.286499	37.400002	37.518501	37.966499	37.966499
		7	37.765499	36.764000	37.324501	37.049999	37.049999
		8	37.506001	36.445999	37.389999	36.545502	36.545502
...	...	...	...	...	...	...	...
2020	12	24	87.120499	86.217499	86.449997	86.708000	86.708000
		28	89.349998	87.091003	87.245499	88.697998	88.697998
		29	89.423500	87.755501	89.361504	87.888000	87.888000
		30	88.388000	86.400002	88.250000	86.812500	86.812500
		31	87.875000	86.804497	86.863503	87.632004	87.632004

1259 rows × 5 columns

3.4 Use sort_index() to sort df2 by year, month, and day in an ascending order and modify df2 in place.

In [17]:
```
#3.4 Use sort_index() to sort df2 by year, month, and day in an ascending order a
df2.sort_index(ascending=[True,True,True], inplace=True)
df2
```

year	month	day	High	Low	Open	Close	Adj Close
2016	1	4	38.110001	37.376999	38.110001	37.972000	37.972000
		5	38.459999	37.782501	38.205002	38.076500	38.076500
		6	38.286499	37.400002	37.518501	37.966499	37.966499
		7	37.765499	36.764000	37.324501	37.049999	37.049999
		8	37.506001	36.445999	37.389999	36.545502	36.545502
...	...	...	...	...	...	...	...
2020	12	24	87.120499	86.217499	86.449997	86.708000	86.708000
		28	89.349998	87.091003	87.245499	88.697998	88.697998
		29	89.423500	87.755501	89.361504	87.888000	87.888000
		30	88.388000	86.400002	88.250000	86.812500	86.812500
		31	87.875000	86.804497	86.863503	87.632004	87.632004

1259 rows × 5 columns

3.5 Use sort_index() to sort df2 by year, month, and day in an ascending order and modify df2 in place.

In [18]:
```
#3.5 Use sort_index() to sort df2 by year, month, and day in an ascending order a
df2.loc[(slice(None),slice(1,3),slice(1,15)),:]
```

Out[18]:

			High	Low	Open	Close	Adj Close
year	month	day					
2016	1	4	38.110001	37.376999	38.110001	37.972000	37.972000
		5	38.459999	37.782501	38.205002	38.076500	38.076500
		6	38.286499	37.400002	37.518501	37.966499	37.966499
		7	37.765499	36.764000	37.324501	37.049999	37.049999
		8	37.506001	36.445999	37.389999	36.545502	36.545502
...	...	...	...	...	...	...	...
2020	3	9	62.636501	59.902000	60.248001	60.789501	60.789501
		10	63.792500	60.725498	62.719501	63.758499	63.758499
		11	62.846500	59.556999	62.413502	60.544998	60.544998
		12	59.437500	55.423500	56.131001	55.577499	55.577499
		13	60.720001	55.614498	58.749500	60.713501	60.713501

155 rows × 5 columns

3.6 From df2, select rows where month is from 1 to 3 and day is from 1 to 15, return the third index level, count the unique values in the third index level, and sort the series of value counts by the index.

In [19]:
```
eturn the third index level, count the unique values in the third index level, and
counts().sort_index(ascending=True)
```

Out[19]:
```
1      8
2     10
3     10
4     10
5     10
6     11
7     11
8     12
9     11
10    10
11    10
12    10
13    11
14    11
15    10
Name: day, dtype: int64
```

3.7 From df2, select rows where month is from 1 to 3 and day is from 1 to 15, return the third index level, count the unique values in the third index level, and sort the series of value counts by the index.

In [20]:
```
#3.7 From df2, select rows where month is from 1 to 3 and day is from 1 to 15, re
df2.loc[(slice(None),slice(1,3),slice(1,15)),"Adj Close"].sort_values(ascending=F
```

Out[20]:
```
year  month  day
2020  2      14     75.936501
             12     75.931503
             13     75.669502
Name: Adj Close, dtype: float64
```

3.7 From df2, select rows where month is from 1 to 3 and day is from 1 to 15, select the Adj Close column, and return the 3 largest adjusted closing prices in a descending order. Use both the sort_values() and the nlargest() methods. The output looks like the following series.

In [21]:
```
#3.7  From df2, select rows where month is from 1 to 3 and day is from 1 to 15, s
df2.loc[(slice(None),slice(1,3),slice(1,15)),"Adj Close"].nlargest(3)
```

Out[21]:
```
year  month  day
2020  2      14     75.936501
             12     75.931503
             13     75.669502
Name: Adj Close, dtype: float64
```

3.8 Group df2 by year and month and calculate the median value of the numeric variables for each group

In [22]:
```
#3.8 Group df2 by year and month and calculate the median value of the numeric va
df2.groupby(["year","month"]).median()
```

Out[22]:

year	month	High	Low	Open	Close	Adj Close
2016	1	37.428001	36.445999	37.037498	36.681000	36.681000
	2	36.502748	35.719500	36.210751	35.963499	35.963499
	3	37.795500	37.254000	37.376749	37.596251	37.596251
	4	38.421001	37.814999	38.265999	38.216000	38.216000
	5	36.633999	35.982498	36.216000	36.258999	36.258999
	6	36.659750	36.051001	36.431749	36.357750	36.357750
	7	37.403500	36.778749	36.972750	37.225000	37.225000
	8	40.116001	39.799500	39.989498	40.006001	40.006001
	9	40.295502	39.801498	40.055500	40.132000	40.132000
	10	40.993000	40.400002	40.708500	40.588501	40.588501
	11	39.564499	38.904999	39.358501	39.109501	39.109501
	12	40.553501	40.224998	40.400501	40.395000	40.395000
2017	1	41.514000	41.158501	41.452250	41.390751	41.390751
	2	42.088501	41.811001	41.924999	41.948002	41.948002
	3	42.824501	42.384499	42.598999	42.507000	42.507000
	4	42.950001	42.376499	42.626999	42.756500	42.756500
	5	48.049500	47.601500	47.842999	47.814999	47.814999
	6	49.005001	48.410000	48.770250	48.564751	48.564751
	7	48.589499	47.931751	48.497000	48.324249	48.324249
	8	47.190498	46.695999	46.960499	47.004002	47.004002
	9	47.512501	46.897249	47.243752	47.163750	47.163750
	10	50.345751	49.549250	50.057751	49.886999	49.886999
	11	52.543999	51.900002	52.193501	52.207500	52.207500
	12	53.183500	52.651999	52.774502	52.733749	52.733749
2018	1	57.029499	56.223000	56.901501	56.798500	56.798500
	2	55.749500	54.500000	54.668999	54.775002	54.775002
	3	55.435001	54.360500	54.919998	54.790001	54.790001
	4	52.335499	51.022499	51.721001	51.632000	51.632000
	5	54.439251	53.480499	53.875000	54.131750	54.131750
	6	57.782001	56.849998	57.610500	57.347500	57.347500
	7	60.538502	59.669998	59.962002	60.063000	60.063000
	8	62.598999	61.471001	62.214001	61.908001	61.908001

year	month	High	Low	Open	Close	Adj Close
	9	59.547001	58.530499	59.209999	59.106998	59.106998
	10	56.291500	54.851501	55.900002	55.568501	55.568501
	11	53.705002	52.147499	53.169498	53.413502	53.413502
	12	53.211498	51.652000	52.364498	52.392502	52.392502
2019	1	54.582001	53.417500	54.125500	54.082500	54.082500
	2	56.546501	55.705502	55.950001	56.100498	56.100498
	3	59.747501	59.074001	59.484501	59.427502	59.427502
	4	61.152500	60.664501	60.965000	61.136501	61.136501
	5	58.446249	57.242001	57.687000	58.059000	58.059000
	6	54.749001	53.885500	54.396252	54.139000	54.139000
	7	57.391750	56.602501	57.123249	57.124750	57.124750
	8	59.841751	58.649752	59.204250	59.068499	59.068499
	9	61.881500	60.715500	61.250999	61.394751	61.394751
	10	62.436501	61.926498	62.010502	62.112000	62.112000
	11	65.538250	64.704498	65.019249	65.148750	65.148750
	12	67.728996	67.055000	67.394997	67.221497	67.221497
2020	1	72.131500	71.388496	71.904999	71.959999	71.959999
	2	74.315002	73.270500	73.369499	74.129997	74.129997
	3	58.622000	55.311750	56.588499	57.706499	57.706499
	4	63.755501	61.519001	62.135502	62.865002	62.865002
	5	69.735001	68.644253	68.965252	68.988003	68.988003
	6	72.382500	71.028748	71.794498	71.853001	71.853001
	7	76.378750	74.835503	75.373249	75.840000	75.840000
	8	76.848503	75.410500	75.798500	75.832497	75.832497
	9	76.934998	74.651001	76.250000	75.441498	75.441498
	10	79.038998	76.305500	77.362751	77.808750	77.808750
	11	88.421501	86.866249	87.493500	87.824249	87.824249
	12	88.807499	87.272003	88.223251	87.971001	87.971001