2_2_high_lelvel_exploration_of_dataset

September 21, 2023

0.0.1 Explore the whole Dataset in a general way to understand it more deeply.

[1]:
```
# import libraries
import pandas as pd
import os
import glob
from pathlib import Path
```

[2]:
```
# change into Dataset't directory
os.chdir("../dataset")
```

[3]:
```
# Create dictionaries to hold ETFs and Stocks information in memory
ETFs, Stocks = {}, {}
```

[4]:
```
# load all ETFs
for file in glob.glob("etfs/**.csv"):
    df = pd.read_csv(file)
    df.set_index(pd.DatetimeIndex(df["Date"]), inplace=True)
    key = Path(file).stem
    ETFs[key] = df
```

[5]:
```
# load all Stocks
for file in glob.glob("stocks/**.csv"):
    df = pd.read_csv(file)
    df.set_index(pd.DatetimeIndex(df["Date"]), inplace=True)
    key = Path(file).stem
    Stocks[key] = df
```

0.0.2 Learn what is inside the dataset

[6]:
```
# Get the first symbol from ETFs
df = ETFs[list(ETFs.keys())[0]]
```

[7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1334 entries, 2018-05-16 to 2023-09-01
Data columns (total 7 columns):
```

```
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       1334 non-null   object
 1   Open       1334 non-null   float64
 2   High       1334 non-null   float64
 3   Low        1334 non-null   float64
 4   Close      1334 non-null   float64
 5   Adj Close  1334 non-null   float64
 6   Volume     1334 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 83.4+ KB
```

[8]:
```
df.head()
```

[8]:

	Date	Open	High	Low	Close	Adj Close	Volume
Date							
2018-05-16	2018-05-16	25.000000	25.010000	24.950001	24.950001	18.586744	4500
2018-05-17	2018-05-17	24.990000	25.000000	24.990000	25.000000	18.623989	3600
2018-05-18	2018-05-18	25.039000	25.059999	25.039000	25.059999	18.668688	3500
2018-05-21	2018-05-21	25.084999	25.090000	25.040001	25.040001	18.653793	9100
2018-05-22	2018-05-22	25.028999	25.030001	25.000000	25.000000	18.623989	9200

0.0.3 Learn some basic information about the data

- ETF and Stock with the earliest date / most datarows
- ETF and Stock with the least datarows
- ETF and Stock with the highest and lowest ever Adjusted closing price

[9]:
```
# Create temporary DataFrame with base interesting information about every␣
 ↪symbol to help further exploration

tmp_list = []
for key, df in {**ETFs, **Stocks}.items():
    etf = True if key in ETFs else False
    tmp_list.append([key,
                     df.index.min(),
                     df["Adj Close"].idxmin(),
                     df["Adj Close"].min(),
                     df["Adj Close"].idxmax(),
```

```
                      df["Adj Close"].max(),
                      df["Volume"].idxmin(),
                      df["Volume"].min(),
                      df["Volume"].idxmax(),
                      df["Volume"].max(),
                      etf])

symbols_info = pd.DataFrame(tmp_list, columns=["Symbol", "First Seen",
                                    "Adj Close Min Id", "Adj Close Min",
                                    "Adj Close Max Id", "Adj Close Max",
                                    "Volume Min Id", "Volume Min",
                                    "Volume Max Id", "Volume Max",
                                    "ETF"])
```

[10]:
```
symbols_info.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10476 entries, 0 to 10475
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Symbol            10476 non-null  object
 1   First Seen        10476 non-null  datetime64[ns]
 2   Adj Close Min Id  10476 non-null  datetime64[ns]
 3   Adj Close Min     10476 non-null  float64
 4   Adj Close Max Id  10476 non-null  datetime64[ns]
 5   Adj Close Max     10476 non-null  float64
 6   Volume Min Id     10476 non-null  datetime64[ns]
 7   Volume Min        10476 non-null  int64
 8   Volume Max Id     10476 non-null  datetime64[ns]
 9   Volume Max        10476 non-null  int64
 10  ETF               10476 non-null  bool
dtypes: bool(1), datetime64[ns](5), float64(2), int64(2), object(1)
memory usage: 828.8+ KB
```

[11]:
```
symbols_info.describe()
```

[11]:

	First Seen	Adj Close Min Id	\
count	10476	10476	
mean	2011-09-22 17:07:12.989690624	2015-03-01 03:16:25.567010304	
min	1962-01-02 00:00:00	1962-05-28 00:00:00	
25%	2005-10-26 00:00:00	2009-03-09 00:00:00	
50%	2016-12-17 12:00:00	2020-03-24 00:00:00	
75%	2021-05-26 00:00:00	2022-12-31 00:00:00	
max	2023-09-01 00:00:00	2023-09-01 00:00:00	
std	NaN	NaN	

```
       Adj Close Min               Adj Close Max Id  Adj Close Max  \
count   10476.000000                          10476   1.047600e+04
mean       10.166979  2020-01-01 05:46:07.010309376   8.304749e+09
min      -689.452881            1980-03-18 00:00:00   2.100000e-03
25%         1.036377            2020-12-10 00:00:00   2.004387e+01
50%         5.595000            2021-11-05 00:00:00   3.505500e+01
75%        16.419993            2022-11-15 00:00:00   7.806053e+01
max       356.965698            2023-09-01 00:00:00   8.675100e+13
std        14.839491                            NaN   8.475735e+11

                     Volume Min Id    Volume Min  \
count                        10476  1.047600e+04
mean   2012-10-29 11:20:41.237113344  2.673253e+04
min              1962-01-02 00:00:00  0.000000e+00
25%              2007-01-29 00:00:00  0.000000e+00
50%              2018-02-21 12:00:00  0.000000e+00
75%              2022-01-04 06:00:00  2.900000e+03
max              2023-09-01 00:00:00  9.744000e+06
std                              NaN  2.285973e+05

                     Volume Max Id    Volume Max
count                        10476  1.047600e+04
mean   2018-12-01 01:00:20.618556416  2.565835e+07
min              1973-03-26 00:00:00  0.000000e+00
25%              2018-02-08 00:00:00  8.350750e+05
50%              2021-07-09 00:00:00  4.413550e+06
75%              2022-12-22 00:00:00  1.831219e+07
max              2023-09-01 00:00:00  7.421641e+09
std                              NaN  1.157440e+08
```

[12]:
```
symbols_info.head()
```

[12]:
```
  Symbol First Seen Adj Close Min Id  Adj Close Min Adj Close Max Id  \
0   PFFA 2018-05-16       2020-03-18       6.911489       2021-08-06
1   SCHV 2009-12-15       2010-07-06      17.392866       2022-01-04
2   AMND 2020-07-23       2020-10-01      20.643538       2022-06-07
3    IDU 2000-06-20       2002-10-09      10.409277       2022-09-12
4   PULS 2018-04-10       2020-03-20      46.116730       2021-07-29

   Adj Close Max Volume Min Id  Volume Min Volume Max Id  Volume Max   ETF
0      25.430000    2018-06-13           0    2021-05-12     1266800  True
1      74.190002    2011-11-25        6700    2020-06-02     5856700  True
2      42.729000    2020-07-28           0    2020-11-11       34300  True
3      95.889999    2000-06-23           0    2014-12-29    19570000  True
4      49.740002    2018-04-19           0    2022-03-07    12885500  True
```

[13]:
```
symbols_info.tail()
```

[13]:
```
       Symbol First Seen Adj Close Min Id  Adj Close Min Adj Close Max Id  \
10471   WWAC 2021-12-17       2022-01-19       9.710000       2023-08-15
10472  ENERW 2023-09-01       2023-09-01       0.037500       2023-09-01
10473   SXTC 2019-01-04       2022-05-18       0.113000       2019-01-09
10474   IVCA 2022-06-30       2022-06-30       9.950000       2023-08-28
10475   POOL 1995-10-13       1995-10-18       0.675367       2021-11-18

       Adj Close Max Volume Min Id  Volume Min Volume Max Id  Volume Max  \
10471      10.490000    2022-01-04           0    2023-04-03     7540400
10472       0.037500    2023-09-01         165    2023-09-01         165
10473      47.799999    2019-06-11         125    2022-03-29   217753500
10474      10.750000    2022-07-01           0    2023-06-13     4054000
10475     577.849976    1996-05-16           0    2007-09-21    10618900

         ETF
10471  False
10472  False
10473  False
10474  False
10475  False
```

[14]:
```
# ETF and Stock with the earliest date / most datarows
print("The ETF with the most datarow is {0}, first seen on {1}.".format(
    *symbols_info.loc[symbols_info[symbols_info["ETF"] == True]["First Seen"].
 ↪idxmin()][["Symbol", "First Seen"]].values
))

print("The Stock with the most datarow is {0}, first seen on {1}.".format(
    *symbols_info.loc[symbols_info[symbols_info["ETF"] == False]["First Seen"].
 ↪idxmin()][["Symbol", "First Seen"]].values
))
```

```
The ETF with the most datarow is CEF, first seen on 1986-04-03 00:00:00.
The Stock with the most datarow is FL, first seen on 1962-01-02 00:00:00.
```

[15]:
```
# ETF and Stock with the least datarows
print("The ETF with the least datarow is {0}, first seen on {1}.".format(
    *symbols_info.loc[symbols_info[symbols_info["ETF"] == True]["First Seen"].
 ↪idxmax()][["Symbol", "First Seen"]].values
))

print("The Stock with the least datarow is {0}, first seen on {1}.".format(
    *symbols_info.loc[symbols_info[symbols_info["ETF"] == False]["First Seen"].
 ↪idxmax()][["Symbol", "First Seen"]].values
))
```

```
The ETF with the least datarow is FTWO, first seen on 2023-08-31 00:00:00.
```

```
The Stock with the least datarow is INDIW, first seen on 2023-09-01 00:00:00.
```

[16]:
```
# ETF with the highest and lowest ever Adjusted closing price
print("The ETF with the highest ever adjusted closing price is {0}, with the␣
 ↪price of {1}USD.".format(
    *symbols_info.loc[symbols_info["ETF"] == True]["Adj Close␣
 ↪Max"].idxmax()][["Symbol", "Adj Close Max"]].values
))

print("The ETF with the lowest ever adjusted closing price is {0}, with the␣
 ↪price of {1}USD.".format(
    *symbols_info.loc[symbols_info["ETF"] == True]["Adj Close␣
 ↪Min"].idxmin()][["Symbol", "Adj Close Min"]].values
))
```

```
The ETF with the highest ever adjusted closing price is UVXY, with the price of
20579999744.0USD.
The ETF with the lowest ever adjusted closing price is TECL, with the price of
0.15114666521549 22USD.
```

[17]:
```
# Stock with the highest and lowest ever Adjusted closing price
print("The Stock with the highest ever adjusted closing price is {0}, with the␣
 ↪price of {1}USD.".format(
    *symbols_info.loc[symbols_info["ETF"] == False]["Adj Close␣
 ↪Max"].idxmax()][["Symbol", "Adj Close Max"]].values
))

print("The Stock with the lowest ever adjusted closing price is {0}, with the␣
 ↪price of {1}USD.".format(
    *symbols_info.loc[symbols_info["ETF"] == False]["Adj Close␣
 ↪Min"].idxmin()][["Symbol", "Adj Close Min"]].values
))
```

```
The Stock with the highest ever adjusted closing price is TOPS, with the price
of 86750999347200.0USD.
The Stock with the lowest ever adjusted closing price is VHI, with the price of
-689.452880859375USD.
```

0.0.4 Get all ETFs and Stocks that have data at least from 2000 Jan 1st

[18]:
```
# Filter the symbols_info to get ETFs and Stocks that have engough data that it␣
 ↪will be meaningful for training models
ETFs_pre2000 = symbols_info.loc[(symbols_info["First Seen"] <= "2000-01-01") &␣
 ↪(symbols_info["ETF"] == True)]
Stocks_pre2000 = symbols_info.loc[(symbols_info["First Seen"] <= "2000-01-01")␣
 ↪& (symbols_info["ETF"] == False)]
```

[19]:
```
ETFs_pre2000.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 32 entries, 38 to 3164
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Symbol            32 non-null     object
 1   First Seen        32 non-null     datetime64[ns]
 2   Adj Close Min Id  32 non-null     datetime64[ns]
 3   Adj Close Min     32 non-null     float64
 4   Adj Close Max Id  32 non-null     datetime64[ns]
 5   Adj Close Max     32 non-null     float64
 6   Volume Min Id     32 non-null     datetime64[ns]
 7   Volume Min        32 non-null     int64
 8   Volume Max Id     32 non-null     datetime64[ns]
 9   Volume Max        32 non-null     int64
 10  ETF               32 non-null     bool
dtypes: bool(1), datetime64[ns](5), float64(2), int64(2), object(1)
memory usage: 2.8+ KB
```

[20]:
```
ETFs_pre2000
```

[20]:

	Symbol	First Seen	Adj Close Min Id	Adj Close Min	Adj Close Max Id	\
38	EWJ	1996-03-18	2003-04-25	19.765661	2021-09-15	
70	EWM	1996-03-18	1998-10-07	2.368347	2014-08-27	
117	EWC	1996-03-18	1996-07-16	4.781969	2022-03-30	
190	EWD	1996-03-18	2002-10-09	4.161988	2021-08-13	
361	EWQ	1996-03-18	1996-03-21	6.174470	2021-11-08	
461	XLY	1998-12-22	2009-03-09	13.498015	2021-11-19	
540	XLP	1998-12-22	2003-03-10	11.246113	2022-04-20	
622	XLK	1998-12-22	2002-10-09	8.930201	2023-07-18	
679	XLB	1998-12-22	2000-09-27	10.671762	2021-12-31	
766	DIA	1998-01-20	1998-08-31	45.462040	2022-01-04	
772	XLE	1998-12-22	2002-07-23	12.327965	2022-11-15	
787	MDY	1995-05-04	1995-05-05	26.305365	2021-11-16	
805	EWP	1996-03-18	1996-03-18	5.208922	2007-12-10	
875	EWW	1996-03-18	1998-09-10	4.544022	2023-07-28	
1079	EWL	1996-03-18	2003-03-12	6.658945	2021-12-29	
1151	EWK	1996-03-18	2003-03-12	4.025899	2021-06-15	
1361	BBH	1999-11-23	2002-07-10	8.318973	2021-08-09	
1417	XLV	1998-12-22	2001-09-20	15.964645	2022-04-08	
1666	XLI	1998-12-22	2009-03-09	11.870877	2023-08-01	
1777	XLU	1998-12-22	2002-10-09	7.786070	2022-09-12	
1997	EWS	1996-03-18	1998-09-03	2.866456	2018-01-24	
2052	SPY	1993-01-29	1993-02-18	25.487831	2022-01-03	
2153	EWA	1996-03-18	1998-08-27	2.950543	2021-06-14	

2291	EWH	1996-03-18	1998-08-31	2.924842	2021-05-28
2316	EWO	1996-03-18	2000-10-18	4.365574	2007-06-04
2670	QQQ	1999-03-10	2002-10-09	17.417257	2021-11-19
2704	XLF	1998-12-22	2009-03-06	3.068984	2022-01-12
2794	EWN	1996-03-18	2003-03-12	6.302496	2021-09-07
2863	EWI	1996-03-18	1996-03-28	9.111309	2007-05-07
2881	CEF	1986-04-03	2000-10-31	2.817608	2011-08-22
2904	EWG	1996-03-18	2003-03-12	5.553696	2021-06-07
3164	EWU	1996-03-18	1996-07-16	9.411281	2022-02-09

	Adj Close Max	Volume Min Id	Volume Min	Volume Max Id	Volume Max	ETF
38	74.120003	1996-06-11	225	2011-03-15	99041775	True
70	36.642010	1996-03-21	0	2018-05-09	4399800	True
117	40.660000	1996-03-20	0	2022-01-18	18747200	True
190	48.970001	1996-03-20	0	2011-02-03	5970300	True
361	40.450001	1996-03-22	0	2013-06-11	11532400	True
461	211.419998	2000-08-14	400	2014-02-03	32477700	True
540	80.570000	2000-10-04	6500	2015-08-24	68723800	True
622	180.259995	2000-07-03	39400	2018-10-11	68269500	True
679	90.610001	1998-12-29	100	2011-08-04	39007900	True
766	367.869995	1998-11-27	71000	2008-11-21	91695200	True
772	94.080002	1999-11-26	7400	2020-11-09	99356700	True
787	531.030029	1995-06-07	300	2008-10-10	27593700	True
805	36.686699	1996-03-19	0	2017-10-05	13154500	True
875	65.360001	1996-06-10	0	2018-11-09	18100000	True
1079	53.020000	1996-03-20	0	2018-11-14	14710000	True
1151	23.209999	1996-03-20	0	2016-05-11	11977500	True
1361	220.750000	2022-11-25	1100	2002-07-11	13257600	True
1417	142.830002	2001-10-10	900	2015-09-28	66470200	True
1666	110.750000	1999-01-26	0	2020-02-28	79118200	True
1777	78.120003	1999-09-28	900	2020-02-28	90263100	True
1997	24.832661	1996-03-21	0	2008-03-28	12505100	True
2052	477.709991	1994-09-30	5200	2008-10-10	871026300	True
2153	27.049999	1996-03-26	0	2019-12-17	20564600	True
2291	27.707525	1996-04-24	0	2013-06-07	64668600	True
2316	28.348576	1996-03-20	0	2007-04-04	2815100	True
2670	403.989990	1999-07-02	3302000	2008-09-15	616772300	True
2704	41.419998	1999-07-16	24497	2008-09-18	1050592272	True
2794	53.779999	1996-03-22	0	2009-08-06	2158500	True
2863	45.699600	1996-03-19	0	2014-08-07	15287400	True
2881	26.175383	1986-12-10	0	2011-05-05	11600000	True
2904	35.870651	1996-03-20	0	2016-06-24	46597900	True
3164	34.759998	1996-03-21	0	2018-03-06	21639600	True

[21]:
```
Stocks_pre2000.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Index: 1856 entries, 3169 to 10475
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Symbol            1856 non-null   object
 1   First Seen        1856 non-null   datetime64[ns]
 2   Adj Close Min Id  1856 non-null   datetime64[ns]
 3   Adj Close Min     1856 non-null   float64
 4   Adj Close Max Id  1856 non-null   datetime64[ns]
 5   Adj Close Max     1856 non-null   float64
 6   Volume Min Id     1856 non-null   datetime64[ns]
 7   Volume Min        1856 non-null   int64
 8   Volume Max Id     1856 non-null   datetime64[ns]
 9   Volume Max        1856 non-null   int64
 10  ETF               1856 non-null   bool
dtypes: bool(1), datetime64[ns](5), float64(2), int64(2), object(1)
memory usage: 161.3+ KB
```

[22]:
```
Stocks_pre2000.head()
```

[22]:

	Symbol	First Seen	Adj Close Min Id	Adj Close Min	Adj Close Max Id	\
3169	SGU	1995-12-15	2005-11-17	0.486986	2023-06-09	
3173	OFC	1991-12-31	1995-06-06	1.113526	2022-03-29	
3176	ADSK	1985-06-28	1985-06-28	0.436705	2021-08-25	
3177	WAT	1995-11-17	1995-11-29	3.417857	2021-09-08	
3180	DLTR	1995-03-07	1995-03-08	1.086420	2022-04-20	

	Adj Close Max	Volume Min Id	Volume Min	Volume Max Id	Volume Max	ETF
3169	15.100000	1997-11-26	0	2004-10-18	12340900	False
3173	29.520000	1992-01-03	0	2021-06-25	13551500	False
3176	342.269989	1985-12-26	12000	1991-01-25	49794400	False
3177	424.700012	1996-05-16	21200	2001-06-19	20194400	False
3180	174.080002	1995-08-15	4556	2000-12-19	96293700	False

0.0.5 Save the temporary generated data objects for later use

[23]:
```
# import libraries
import lzma
import gzip
import pickle
```

[24]:
```
# dump objects
with open("2_2_symbols_info.pckl", "wb") as f:
    %time pickle.dump(symbols_info, f)
```

```
CPU times: user 2.86 ms, sys: 409 µs, total: 3.27 ms
Wall time: 3.23 ms
```

[25]:
```
with open("2_2_etfs.pckl", "wb") as f:
    %time pickle.dump(ETFs, f)
```

```
CPU times: user 1.2 s, sys: 308 ms, total: 1.51 s
Wall time: 1.53 s
```

[26]:
```
with open("2_2_stocks.pckl", "wb") as f:
    %time pickle.dump(Stocks, f)
```

```
CPU times: user 6 s, sys: 1.19 s, total: 7.19 s
Wall time: 7.25 s
```

[27]:
```
with gzip.open("2_2_symbols_info.pckl.gz", "wb") as f:
    %time pickle.dump(symbols_info, f)
```

```
CPU times: user 229 ms, sys: 1.94 ms, total: 231 ms
Wall time: 232 ms
```

[28]:
```
with gzip.open("2_2_etfs.pckl.gz", "wb") as f:
    %time pickle.dump(ETFs, f)
```

```
CPU times: user 2min 2s, sys: 140 ms, total: 2min 2s
Wall time: 2min 3s
```

[29]:
```
with gzip.open("2_2_stocks.pckl.gz", "wb") as f:
    %time pickle.dump(Stocks, f)
```

```
CPU times: user 10min 56s, sys: 1.02 s, total: 10min 57s
Wall time: 11min 35s
```

[30]:
```
with lzma.open("2_2_symbols_info.pckl.xz", "wb") as f:
    %time pickle.dump(symbols_info, f)
```

```
CPU times: user 220 ms, sys: 3.96 ms, total: 224 ms
Wall time: 230 ms
```

[31]:
```
with lzma.open("2_2_etfs.pckl.xz", "wb") as f:
    %time pickle.dump(ETFs, f)
```

```
CPU times: user 2min 20s, sys: 168 ms, total: 2min 20s
Wall time: 2min 33s
```

[32]:
```
with lzma.open("2_2_stocks.pckl.xz", "wb") as f:
    %time pickle.dump(Stocks, f)
```

```
CPU times: user 9min 35s, sys: 1.04 s, total: 9min 36s
Wall time: 10min 13s
```

Though compression really helps to save a lot of storage the time it takes to compress the objects are not worth it. -rw-r–r– 1 user grp 376M Sep 2 17:15 2_2_etfs.pckl

-rw-r–r– 1 user grp 127M Sep 2 17:18 2_2_etfs.pckl.gz

-rw-r–r– 1 user grp 54M Sep 2 17:35 2_2_etfs.pckl.xz

-rw-r–r– 1 user grp 1.7G Sep 2 17:16 2_2_stocks.pckl

-rw-r–r– 1 user grp 536M Sep 2 17:31 2_2_stocks.pckl.gz

-rw-r–r– 1 user grp 215M Sep 2 17:54 2_2_stocks.pckl.xz

-rw-r–r– 1 user grp 818K Sep 2 17:15 2_2_symbols_info.pckl

-rw-r–r– 1 user grp 345K Sep 2 17:16 2_2_symbols_info.pckl.gz

-rw-r–r– 1 user grp 250K Sep 2 17:31 2_2_symbols_info.pckl.xz