Dependancy imports

In [1]:

```
from bs4 import BeautifulSoup
import glob
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import requests
import seaborn as sns
import sklearn.covariance as skcov
from typing import List, Tuple

MIN_VOLUME = 100_000
benchmark = '^GSPC' # add S&P 500 as benchmark
```

Gather all tickers

In [2]:

```
wiki_page = requests.get('https://en.wikipedia.org/wiki/List_of_American_exchange-trade
soup = BeautifulSoup(wiki_page, 'lxml')

list_items = soup.select('li:contains("|")')
tickers = []

for list_item in list_items:
    li_text: str = list_item.text
    start_index: int = li_text.find('|')
    end_index: int = li_text.find(')',start_index)
    tickers.append(li_text[start_index +1:end_index].strip())

print(tickers)
```

```
['DIA', 'RSP', 'IOO', 'IVV', 'SPY', 'SHE', 'VOO', 'IWM', 'OEF', 'QQQ', 'CVY', 'RPG', 'RP
V', 'IWB', 'IWF', 'IWD', 'IVV', 'IVW', 'IVE', 'PKW', 'PRF', 'SPLV', 'SCHX', 'SCHG', 'SCH
V', 'SCHD', 'FNDX', 'SDY', 'VOO', 'VOOG', 'VOOV', 'VV', 'VUG', 'VTV', 'MGC', 'MGK', 'MG
V', 'VONE', 'VONG', 'VONV', 'VIG', 'VYM', 'DTN', 'DLN', 'MDY', 'DVY', 'IWR', 'IWP', 'IW
S', 'IJH', 'IJK', 'IJJ', 'PDP', 'SCHM', 'IVOO', 'IVOG', 'IVOV', 'VO', 'VOT', 'VOE', 'VX
F', 'DON', 'IWC', 'IWM', 'IWO', 'IWN', 'IJR', 'IJT', 'IJS', 'SCHA', 'FNDA', 'VIOO', 'VIO
G', 'VIOV', 'VB', 'VBK', 'VBR', 'VTWO', 'VTWG', 'VTWV', 'EEB', 'ECON', 'IDV', 'ACWX', 'B
KF', 'EFA', 'EFG', 'EFV', 'SCZ', 'EEM', 'PID', 'SCHC', 'SCHE', 'SCHF', 'FNDF', 'FNDC',
'FNDE', 'DWX', 'VEA', 'VWO', 'VXUS', 'VEU', 'VSS', 'DEM', 'DGS', 'AAXJ', 'EZU', 'EPP',
'IEV', 'ILF', 'FEZ', 'VGK', 'VPL', 'HEDJ', 'DFE', 'AND', 'GXF', 'EWA', 'EWC', 'EWG', 'EI
S', 'EWI', 'EWJ', 'EWY', 'EWD', 'EWL', 'EWP', 'EWU', 'DXJ', 'NORW', 'INDF', 'EWZ', 'FX
I', 'EWH', 'EWW', 'EPHE', 'RSX', 'EWS', 'EWM', 'EWT', 'EPI', 'ARGT', 'BRAF', 'BRAQ', 'BR
AZ', 'GXG', 'XLY', 'IYC', 'ITB', 'XHB', 'VCR', 'XLP', 'IYK', 'VDC', 'AMLP', 'XLE', 'IY
E', 'IGE', 'OIH', 'XOP', 'VDE', 'QCLN', 'NERD', 'ESPO', 'XLF', 'IYF', 'KBE', 'KRE', 'VF
H', 'FXH', 'FBT', 'XLV', 'IYH', 'IBB', 'PJP', 'XBI', 'VHT', 'XLI', 'IYJ', 'VIS', 'XLB',
'IYM', 'GDX', 'GDXJ', 'VAW', 'FDN', 'XLK', 'IYW', 'IGV', 'VGT', 'IYZ', 'VOX', 'XLU', 'ID
U', 'VPU', 'IPD', 'RXI', 'IPS', 'KXI', 'IPW', 'IXC', 'IPF', 'IXG', 'IRY', 'IXJ', 'IPN',
'EXI', 'GUNR', 'IRV', 'MXI', 'IPK', 'IXN', 'IST', 'IXP', 'IPU', 'JXI', 'HYLD', 'TDTT',
'CSJ', 'IEI', 'AGG', 'SHY', 'TIP', 'HYG', 'LQD', 'IEF', 'TLT', 'FLOT', 'CIU', 'GVI', 'EM
B', 'MBB', 'MUB', 'SHV', 'HYD', 'HYS', 'STPZ', 'MINT', 'BOND', 'PCY', 'BKLN', 'SCHZ', 'S
CHP', 'SCHO', 'SCHR', 'JNK', 'BIL', 'CWB', 'JNK', 'SCPB', 'BWX', 'SJNK', 'TFI', 'SHM',
'EDV', 'BIV', 'VCIT', 'VGIT', 'BLV', 'VCLT', 'VGLT', 'VMBS', 'BSV', 'VCSH', 'VGSH', 'VTI
P', 'BND', 'BNDX', 'RJI', 'DJP', 'GSG', 'DBC', 'RJA', 'JJA', 'DBA', 'RJN', 'OIL', 'GAZ',
'UNG', 'USO', 'RJZ', 'JJM', 'JJC', 'DBB', 'SGOL', 'IAU', 'GLD', 'SIVR', 'SLV', 'PALL',
'PPLT', 'ICF', 'IFAS', 'IFEU', 'IYR', 'REM', 'SCHH', 'RWO', 'RWX', 'RWR', 'WREI', 'VNQ',
'VNQI', 'HDGE', 'HDGI', 'DOG', 'SH', 'MYY', 'SBB', 'PSQ', 'RWM', 'EFZ', 'FBGX', 'FLGE',
'MIDU', 'SPUU', 'SPXL', 'ERX', 'FAS', 'BGU', 'TNA', 'DDM', 'QLD', 'UWM', 'SSO', 'UPRO',
'SDS', 'SPXU', 'TZA', 'SQQQ', 'QID', 'SKF', 'TWM', 'DXD', 'SRS', 'MZZ', 'DUG', 'BGZ', 'E
```

```
RY', 'FAZ', 'AADR', 'ACCU', 'DBIZ', 'EPRO', 'FWDB', 'FWDD', 'FWDI', 'GEUR', 'GGBP', 'GIV
E', 'GLDE', 'GYEN', 'GTAA', 'HDGE', 'HDGI', 'HOLD', 'HYLD', 'MATH', 'MINC', 'QEH', 'TTF
S', 'VEGA', 'YPRO', 'RIGS', 'ARKG', 'ARKQ', 'ARKW', 'ARKK', 'SYLD', 'GMMB', 'GMTB', 'GV
T', 'RPX', 'RWG', 'EMLP', 'FMB', 'FMF', 'FPE', 'FTGS', 'FTHI', 'FTLB', 'FTSL', 'HYLS',
'RAVI', 'FTSD', 'GSY', 'HECO', 'HUSE', 'ICSH', 'IEIL', 'IEIS', 'IELG', 'IESM', 'NEAR',
'BABZ', 'BOND', 'DI', 'FORX', 'ILB', 'LDUR', 'MINT', 'MUNI', 'SMMU', 'CHNA', 'LALT', 'PH
DG', 'PSR', 'ONEF', 'GAL', 'INKM', 'RLY', 'SYE', 'SYG', 'SYV', 'SRLN', 'ULST', 'ALD', 'A
UNZ', 'BZF', 'CCX', 'CEW', 'CRDT', 'CYB', 'ELD', 'EMCB', 'EU', 'ICB', 'RRF', 'USDU', 'WD
TI']
```

In [3]:

```python
leveraged_page = requests.get('https://etfdb.com/etfs/leveraged/equity/').text
soup = BeautifulSoup(leveraged_page, 'lxml')

list_items = soup.select('td[data-th="Symbol"] > a')
for list_item in list_items:
    tickers.append(list_item.text)

print(tickers)
```

```
['DIA', 'RSP', 'IOO', 'IVV', 'SPY', 'SHE', 'VOO', 'IWM', 'OEF', 'QQQ', 'CVY', 'RPG', 'RP
V', 'IWB', 'IWF', 'IWD', 'IVV', 'IVW', 'IVE', 'PKW', 'PRF', 'SPLV', 'SCHX', 'SCHG', 'SCH
V', 'SCHD', 'FNDX', 'SDY', 'VOO', 'VOOG', 'VOOV', 'VV', 'VUG', 'VTV', 'MGC', 'MGK', 'MG
V', 'VONE', 'VONG', 'VONV', 'VIG', 'VYM', 'DTN', 'DLN', 'MDY', 'DVY', 'IWR', 'IWP', 'IW
S', 'IJH', 'IJK', 'IJJ', 'PDP', 'SCHM', 'IVOO', 'IVOG', 'IVOV', 'VO', 'VOT', 'VOE', 'VX
F', 'DON', 'IWC', 'IWM', 'IWO', 'IWN', 'IJR', 'IJT', 'IJS', 'SCHA', 'FNDA', 'VIOO', 'VIO
G', 'VIOV', 'VB', 'VBK', 'VBR', 'VTWO', 'VTWG', 'VTWV', 'EEB', 'ECON', 'IDV', 'ACWX', 'B
KF', 'EFA', 'EFG', 'EFV', 'SCZ', 'EEM', 'PID', 'SCHC', 'SCHE', 'SCHF', 'FNDF', 'FNDC',
'FNDE', 'DWX', 'VEA', 'VWO', 'VXUS', 'VEU', 'VSS', 'DEM', 'DGS', 'AAXJ', 'EZU', 'EPP',
'IEV', 'ILF', 'FEZ', 'VGK', 'VPL', 'HEDJ', 'DFE', 'AND', 'GXF', 'EWA', 'EWC', 'EWG', 'EI
S', 'EWI', 'EWJ', 'EWY', 'EWD', 'EWL', 'EWP', 'EWU', 'DXJ', 'NORW', 'INDF', 'EWZ', 'FX
I', 'EWH', 'EWW', 'EPHE', 'RSX', 'EWS', 'EWM', 'EWT', 'EPI', 'ARGT', 'BRAF', 'BRAQ', 'BR
AZ', 'GXG', 'XLY', 'IYC', 'ITB', 'XHB', 'VCR', 'XLP', 'IYK', 'VDC', 'AMLP', 'XLE', 'IY
E', 'IGE', 'OIH', 'XOP', 'VDE', 'QCLN', 'NERD', 'ESPO', 'XLF', 'IYF', 'KBE', 'KRE', 'VF
H', 'FXH', 'FBT', 'XLV', 'IYH', 'IBB', 'PJP', 'XBI', 'VHT', 'XLI', 'IYJ', 'VIS', 'XLB',
'IYM', 'GDX', 'GDXJ', 'VAW', 'FDN', 'XLK', 'IYW', 'IGV', 'VGT', 'IYZ', 'VOX', 'XLU', 'ID
U', 'VPU', 'IPD', 'RXI', 'IPS', 'KXI', 'IPW', 'IXC', 'IPF', 'IXG', 'IRY', 'IXJ', 'IPN',
'EXI', 'GUNR', 'IRV', 'MXI', 'IPK', 'IXN', 'IST', 'IXP', 'IPU', 'JXI', 'HYLD', 'TDTT',
'CSJ', 'IEI', 'AGG', 'SHY', 'TIP', 'HYG', 'LQD', 'IEF', 'TLT', 'FLOT', 'CIU', 'GVI', 'EM
B', 'MBB', 'MUB', 'SHV', 'HYD', 'HYS', 'STPZ', 'MINT', 'BOND', 'PCY', 'BKLN', 'SCHZ', 'S
CHP', 'SCHO', 'SCHR', 'JNK', 'BIL', 'CWB', 'JNK', 'SCPB', 'BWX', 'SJNK', 'TFI', 'SHM',
'EDV', 'BIV', 'VCIT', 'VGIT', 'BLV', 'VCLT', 'VGLT', 'VMBS', 'BSV', 'VCSH', 'VGSH', 'VTI
P', 'BND', 'BNDX', 'RJI', 'DJP', 'GSG', 'DBC', 'RJA', 'JJA', 'DBA', 'RJN', 'OIL', 'GAZ',
'UNG', 'USO', 'RJZ', 'JJM', 'JJC', 'DBB', 'SGOL', 'IAU', 'GLD', 'SIVR', 'SLV', 'PALL',
'PPLT', 'ICF', 'IFAS', 'IFEU', 'IYR', 'REM', 'SCHH', 'RWO', 'RWX', 'RWR', 'WREI', 'VNQ',
'VNQI', 'HDGE', 'HDGI', 'DOG', 'SH', 'MYY', 'SBB', 'PSQ', 'RWM', 'EFZ', 'FBGX', 'FLGE',
'MIDU', 'SPUU', 'SPXL', 'ERX', 'FAS', 'BGU', 'TNA', 'DDM', 'QLD', 'UWM', 'SSO', 'UPRO',
'SDS', 'SPXU', 'TZA', 'SQQQ', 'QID', 'SKF', 'TWM', 'DXD', 'SRS', 'MZZ', 'DUG', 'BGZ', 'E
RY', 'FAZ', 'AADR', 'ACCU', 'DBIZ', 'EPRO', 'FWDB', 'FWDD', 'FWDI', 'GEUR', 'GGBP', 'GIV
E', 'GLDE', 'GYEN', 'GTAA', 'HDGE', 'HDGI', 'HOLD', 'HYLD', 'MATH', 'MINC', 'QEH', 'TTF
S', 'VEGA', 'YPRO', 'RIGS', 'ARKG', 'ARKQ', 'ARKW', 'ARKK', 'SYLD', 'GMMB', 'GMTB', 'GV
T', 'RPX', 'RWG', 'EMLP', 'FMB', 'FMF', 'FPE', 'FTGS', 'FTHI', 'FTLB', 'FTSL', 'HYLS',
'RAVI', 'FTSD', 'GSY', 'HECO', 'HUSE', 'ICSH', 'IEIL', 'IEIS', 'IELG', 'IESM', 'NEAR',
'BABZ', 'BOND', 'DI', 'FORX', 'ILB', 'LDUR', 'MINT', 'MUNI', 'SMMU', 'CHNA', 'LALT', 'PH
DG', 'PSR', 'ONEF', 'GAL', 'INKM', 'RLY', 'SYE', 'SYG', 'SYV', 'SRLN', 'ULST', 'ALD', 'A
UNZ', 'BZF', 'CCX', 'CEW', 'CRDT', 'CYB', 'ELD', 'EMCB', 'EU', 'ICB', 'RRF', 'USDU', 'WD
TI', 'TQQQ', 'SOXL', 'QLD', 'SSO', 'FAS', 'UPRO', 'TECL', 'SPXL', 'TNA', 'FNGU', 'SQQQ',
'NUGT', 'UDOW', 'UYG', 'ROM', 'GUSH', 'UWM', 'LABU', 'JNUG', 'ERX', 'SDS', 'SPXU', 'DD
M', 'URTY', 'NRGU']
```

In [4]:

```python
tickers.append(benchmark) # append benchmark
```

Fetch ticker info from Yahoo

In [5]:

```python
import yfinance as yf

data: pd.DataFrame = yf.download(tickers=" ".join(tickers), period="5y", interval="1d",
print(data)
```

```
[*********************100%***********************]  435 of 435 completed

23 Failed downloads:
- CRDT: No data found for this date range, symbol may be delisted
- IFAS: No data found for this date range, symbol may be delisted
- RWG: No data found, symbol may be delisted
- FTGS: No data found, symbol may be delisted
- BABZ: No data found for this date range, symbol may be delisted
- DBIZ: No data found for this date range, symbol may be delisted
- RRF: No data found, symbol may be delisted
- BGU: No data found for this date range, symbol may be delisted
- BRAF: No data found for this date range, symbol may be delisted
- GGBP: No data found for this date range, symbol may be delisted
- ACCU: No data found for this date range, symbol may be delisted
- IRV: No data found for this date range, symbol may be delisted
- AND: No data found, symbol may be delisted
- QEH: No data found, symbol may be delisted
- IELG: No data found for this date range, symbol may be delisted
- ONEF: No data found for this date range, symbol may be delisted
- YPRO: No data found, symbol may be delisted
- HDGI: No data found for this date range, symbol may be delisted
- GVT: No data found for this date range, symbol may be delisted
- GLDE: No data found for this date range, symbol may be delisted
- RPX: No data found, symbol may be delisted
- FORX: No data found for this date range, symbol may be delisted
- WDTI: No data found, symbol may be delisted
           CRDT                                   IFAS                  ...  \
           Open High Low Close Adj Close Volume Open High Low Close  ...
Date                                                                 ...
2016-04-18  NaN  NaN NaN   NaN       NaN    NaN  NaN  NaN NaN   NaN  ...
2016-04-19  NaN  NaN NaN   NaN       NaN    NaN  NaN  NaN NaN   NaN  ...
2016-04-20  NaN  NaN NaN   NaN       NaN    NaN  NaN  NaN NaN   NaN  ...
2016-04-21  NaN  NaN NaN   NaN       NaN    NaN  NaN  NaN NaN   NaN  ...
2016-04-22  NaN  NaN NaN   NaN       NaN    NaN  NaN  NaN NaN   NaN  ...
...         ...  ...  ..   ...       ...    ...  ...  ...  ..   ...  ...
2021-04-12  NaN  NaN NaN   NaN       NaN    NaN  NaN  NaN NaN   NaN  ...
2021-04-13  NaN  NaN NaN   NaN       NaN    NaN  NaN  NaN NaN   NaN  ...
2021-04-14  NaN  NaN NaN   NaN       NaN    NaN  NaN  NaN NaN   NaN  ...
2021-04-15  NaN  NaN NaN   NaN       NaN    NaN  NaN  NaN NaN   NaN  ...
2021-04-16  NaN  NaN NaN   NaN       NaN    NaN  NaN  NaN NaN   NaN  ...

                   EDV                                          ACWX             \
                   Low       Close   Adj Close  Volume        Open       High
Date
2016-04-18  127.070000  127.930000  104.986168   46600  40.180000  40.590000
2016-04-19  126.639999  127.339996  104.501976   81700  41.130001  41.290001
2016-04-20  125.339996  125.459999  102.959145  170200  41.139999  41.400002
2016-04-21  123.940002  124.339996  102.040009  103100  41.160000  41.220001
2016-04-22  123.669998  123.889999  101.670708  113100  40.950001  41.070000
...                ...         ...         ...     ...        ...        ...
2021-04-12  126.629997  126.940002  126.940002  110200  56.139999  56.139999
2021-04-13  126.699997  128.059998  128.059998   85200  56.160000  56.389999
2021-04-14  127.050003  127.550003  127.550003   85200  56.520000  56.689999
2021-04-15  129.339996  130.389999  130.389999  121300  56.830002  56.939999
2021-04-16  128.660004  129.179993  129.179993  115000  57.070000  57.250000


                  Low      Close  Adj Close   Volume
```

```
Date
2016-04-18  40.090000  40.560001  35.551880   418900
2016-04-19  41.049999  41.240002  36.147919   248400
2016-04-20  41.070000  41.200001  36.112854   418500
2016-04-21  40.869999  40.959999  35.902485   830100
2016-04-22  40.790001  40.930000  35.876190   960100
...               ...        ...        ...      ...
2021-04-12  55.959999  56.080002  56.080002   551700
2021-04-13  56.080002  56.369999  56.369999   844100
2021-04-14  56.410000  56.459999  56.459999  1046500
2021-04-15  56.770000  56.919998  56.919998   594100
2021-04-16  56.970001  57.240002  57.240002  2282500

[1259 rows x 2610 columns]
```

Delete existing file cache

In [6]:

```
files = glob.glob(os.path.join("data", '*'))

for file in files:
    os.remove(file)
```

Save output to file to prevent further network requests.

In [7]:

```
found_tickers: List[str] = data.columns.get_level_values(0).unique().to_list()

for found_ticker in found_tickers:
    data[found_ticker].to_csv(os.path.join("data", found_ticker + '.csv'))
```

Read files back from directory.

In [8]:

```
csv_paths = glob.glob(os.path.join("data", '*.csv'))
prices_df = None
vol_df = None

for csv_path in csv_paths:
    (ticker_id, extension) = csv_path.split(".", 1)
    df = pd.read_csv(csv_path, index_col='Date', usecols=['Date', 'Adj Close', 'Volume'

    if prices_df is not None:
        prices_df = prices_df.join(df[['Adj Close']])
        vol_df = vol_df.join(df[['Volume']])
    else:
        prices_df = df[['Adj Close']]
        vol_df = df[['Volume']]

    prices_df = prices_df.rename(columns={'Adj Close': os.path.split(ticker_id)[1]})
    vol_df = vol_df.rename(columns={'Volume': os.path.split(ticker_id)[1]})

prices_df = prices_df.sort_values(by='Date', axis=0)
vol_df = vol_df.sort_values(by='Date', axis=0)

print(prices_df)
print(vol_df)
```

```
                 AADR       AAXJ  ACCU       ACWX        AGG        ALD  \
Date
2016-04-18  38.013577  50.725719   NaN  35.551880  97.996193  42.828297
2016-04-19  38.354286  51.283649   NaN  36.147919  97.987366  43.017387
```

```
2016-04-20  38.480835  50.808037   NaN  36.112854   97.748909  42.913380
2016-04-21  37.974636  50.460476   NaN  35.902485   97.616371  42.771580
2016-04-22  37.964901  50.259254   NaN  35.876190   97.625206  42.516312
...               ...        ...   ...        ...         ...        ...
2021-04-12  62.130001  92.410004   NaN  56.080002  114.150002        NaN
2021-04-13  63.020000  92.879997   NaN  56.369999  114.480003        NaN
2021-04-14  63.360001  93.320000   NaN  56.459999  114.389999        NaN
2021-04-15  63.779999  93.900002   NaN  56.919998  114.839996        NaN
2021-04-16  64.669998  94.099998   NaN  57.240002  114.540001        NaN

                 AMLP  AND       ARGT       ARKG  ...        XLF         XLI  \
Date                                              ...
2016-04-18  37.004353  NaN  19.658564  17.574158  ...  13.032992   50.818710
2016-04-19  38.496212  NaN  20.334116  16.921885  ...  13.196821   51.181065
2016-04-20  39.209705  NaN  20.420973  17.080292  ...  13.298507   51.099537
2016-04-21  39.242134  NaN  20.372719  17.238703  ...  13.179873   50.972710
2016-04-22  39.598881  NaN  20.170053  17.271315  ...  13.304155   51.099537
...               ...  ...        ...        ...  ...        ...         ...
2021-04-12  30.930000  NaN  29.719999  86.620003  ...  35.299999  100.879997
2021-04-13  30.940001  NaN  29.840000  89.320000  ...  34.970001  100.389999
2021-04-14  31.490000  NaN  29.610001  89.470001  ...  35.180000  100.519997
2021-04-15  31.730000  NaN  30.150000  90.339996  ...  35.150002  100.919998
2021-04-16  31.330000  NaN  30.410000  88.699997  ...  35.400002  101.139999

                   XLK        XLP        XLU         XLV         XLY  \
Date
2016-04-18   41.492729  46.561714  41.557678   64.814850   75.200920
2016-04-19   41.259777  46.535530  41.625397   65.044128   74.788551
2016-04-20   41.343639  46.011772  40.583916   65.392647   74.882271
2016-04-21   41.194553  45.191231  39.737194   65.777840   74.619865
2016-04-22   40.477074  45.357082  40.092819   65.915428   74.469917
...                ...        ...        ...         ...         ...
2021-04-12  141.089996  69.290001  64.879997  118.070000  177.110001
2021-04-13  142.419998  68.919998  65.650002  118.559998  178.979996
2021-04-14  140.910004  68.860001  65.940002  118.550003  177.220001
2021-04-15  143.330002  69.410004  66.669998  120.580002  178.490005
2021-04-16  143.300003  69.809998  67.209999  121.480003  179.869995

                   XOP  YPRO        ^GSPC
Date
2016-04-18  123.840950   NaN  2094.340088
2016-04-19  126.972359   NaN  2100.800049
2016-04-20  128.724457   NaN  2102.399902
2016-04-21  127.382439   NaN  2091.479980
2016-04-22  131.855927   NaN  2091.580078
...                ...   ...          ...
2021-04-12   76.589996   NaN  4127.990234
2021-04-13   76.800003   NaN  4141.589844
2021-04-14   80.040001   NaN  4124.660156
2021-04-15   78.809998   NaN  4170.419922
2021-04-16   77.510002   NaN  4185.470215

[1259 rows x 435 columns]
             AADR     AAXJ  ACCU     ACWX      AGG     ALD     AMLP  AND  \
Date
2016-04-18   5200   401300   NaN   418900  2146600  9500.0  2527620  NaN
2016-04-19   1000   566900   NaN   248400  2019200  3700.0  2538420  NaN
2016-04-20    300   396300   NaN   418500  2376000  2000.0  2862520  NaN
2016-04-21    800   548400   NaN   830100  2692800   300.0  2578700  NaN
2016-04-22   1700   602800   NaN   960100  3795600   600.0  3200320  NaN
...           ...      ...   ...      ...      ...     ...      ...  ...
2021-04-12   1100  2127300   NaN   551700  3844900     NaN  1925300  NaN
2021-04-13  36800  1985800   NaN   844100  4040500     NaN  1855400  NaN
2021-04-14   3000   693500   NaN  1046500  3620500     NaN  2188200  NaN
2021-04-15   2300   610600   NaN   594100  6753200     NaN  1352900  NaN
```

```
2021-04-16   5500   696800   NaN  2282500  3663600     NaN  1952300  NaN

                ARGT     ARKG  ...       XLF       XLI       XLK       XLP  \
Date                           ...
2016-04-18  52000     1200  ...  52434000   8619000   7112900   8476800
2016-04-19  83900      900  ...  57169300   9318800   6471000   5980100
2016-04-20  86200    10100  ...  74743100  13632600   6877800  20207300
2016-04-21  59100      100  ...  63436700  11941800  11277900  30667400
2016-04-22  91100      400  ...  46728100  14749400  14924400  22004200
...           ...      ...  ...       ...       ...       ...       ...
2021-04-12  13500  2136600  ...  44435500   7704800   7953500  10117800
2021-04-13   9500  2444400  ...  39950900  10612000   5657900  11982600
2021-04-14   1300  2877900  ...  44065300   9331900   5608100  12087500
2021-04-15  30600  1853800  ...  55483500  12631200   5514400   7714900
2021-04-16   2100  2497600  ...  39098200  10221900   5847800   7807800

                 XLU       XLV      XLY       XOP  YPRO       ^GSPC
Date
2016-04-18   9124000  11505400  4463000   5437275   NaN  3316880000
2016-04-19  10495400   6655000  4272800   4820500   NaN  3896830000
2016-04-20  26068700  10763400  4785500   5829525   NaN  4184880000
2016-04-21  32065700  12463100  4142700   4683525   NaN  4175290000
2016-04-22  12309000  12052800  4730000   4123875   NaN  3790580000
...              ...       ...      ...       ...   ...         ...
2021-04-12   7100700   8279500  3142800   6357400   NaN  3578500000
2021-04-13   9125400   6519600  2675600   5761600   NaN  3728440000
2021-04-14   7562700   6226600  3338200  10005200   NaN  3976540000
2021-04-15  11931700   8438300  2515600   5706700   NaN  4027680000
2021-04-16  11187600   7447400  2638000   4232300   NaN  4157430000

[1259 rows x 435 columns]
```

Calculate the average volume and only bother running further analysis on securities with sufficient liquidity.

```
avg_vol = vol_df.mean()
liquid_tickers = avg_vol.index[avg_vol >= MIN_VOLUME]
prices_df = prices_df[liquid_tickers]
```

In [10]:

```
most_recent_prices_df = prices_df.iloc[-1]
tickers_to_drop = most_recent_prices_df[most_recent_prices_df.isnull()].index.values
tickers_to_drop
```

Out[10]:

```
array(['CIU', 'CSJ', 'EU', 'IPF', 'IRY'], dtype=object)
```

Calculate price returns

In [11]:

```
returns_df = prices_df.pct_change()
returns_df = returns_df.drop(columns=tickers_to_drop)

found_tickers = returns_df.columns
returns_df
```

Out[11]:

	AAXJ	ACWI	AGG	AMLP	ARKG	ARKK	ARKQ	ARKW	BIL
Date									
2016-04-19	0.010999	0.016765	0.000058	0.040316	-0.037115	-0.021296	-0.001029	0.007588	0.000000
2016-04-20	0.002271	0.000578	0.002414	0.018534	0.009361	0.008817	0.000243	0.005246	0.000000
2016-04-21	-0.006841	-0.005825	-0.001356	0.000827	0.009274	0.009813	0.023377	0.008832	-0.000219
2016-04-22	-0.003988	-0.000732	0.000091	0.005091	0.001892	-0.002379	-0.006599	0.002353	0.000219
...	...	...	...	...	...	...	...	...	...
2021-04-12	-0.009008	-0.005497	-0.000263	-0.001936	-0.012765	-0.006409	-0.006797	-0.000583	0.000000
2021-04-13	0.005086	0.005171	0.002851	0.000323	0.031171	0.041561	0.014444	0.048142	0.000000
2021-04-14	0.004747	0.001597	0.000785	0.017776	0.001679	-0.021088	0.005484	0.029868	0.000000
2021-04-15	0.006215	0.008147	0.003544	0.007621	0.005724	0.014015	-0.003281	0.015132	0.000109
2021-04-16	0.002130	0.005822	-0.002612	-0.012606	-0.018154	-0.017691	-0.002489	-0.014209	0.000000

1259 rows × 275 columns

Calucate expected return using geomean from price return

In [12]:

```
from scipy.stats import gmean

exp_return_df = pd.DataFrame()
for found_ticker in found_tickers:
    returns_sr = returns_df[pd.notnull(returns_df[found_ticker])][found_ticker]
    if exp_return_df.empty:
        exp_return_df = pd.DataFrame(data={
            'ticker': found_ticker,
            'exp_return': [0] if returns_sr.empty else [gmean(returns_sr + 1) - 1]
        })
    else:
        exp_return_df = pd.concat([
            exp_return_df,
            pd.DataFrame(data={
                'ticker': found_ticker,
                'exp_return': [0] if returns_sr.empty else [gmean(returns_sr + 1) - 1]
            })
        ])

exp_return_df = exp_return_df.set_index('ticker').T
exp_return_df
```

Out[12]:

[illegible]	AAXJ	ACWX	AGG	AMLF	ARKG	ARKK	ARKQ	ARKW	BIL
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible] 0.

1 rows × 275 columns

Calculate expected covariance using price return

In [13]:

```
lw = skcov.LedoitWolf() # use Ledoit and Wolf Shrinkage covariance
lw.fit(returns_df.fillna(0))
covar_df = pd.DataFrame(lw.covariance_).set_index(pd.Index(returns_df.columns))
covar_df.columns = returns_df.columns
#covar_df = returns_df.cov()
covar_df
```

Out[13]:

	AAXJ	ACWX	AGG	AMLF	ARKG	ARKK	ARKQ	ARKW	BIL
AAXJ	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
ACWX	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
AGG	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
AMLF	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
ARKG	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
...	...	...	...	...	...	...	...	...	...
XLU	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
XLV	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
XLY	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
^GSPC	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

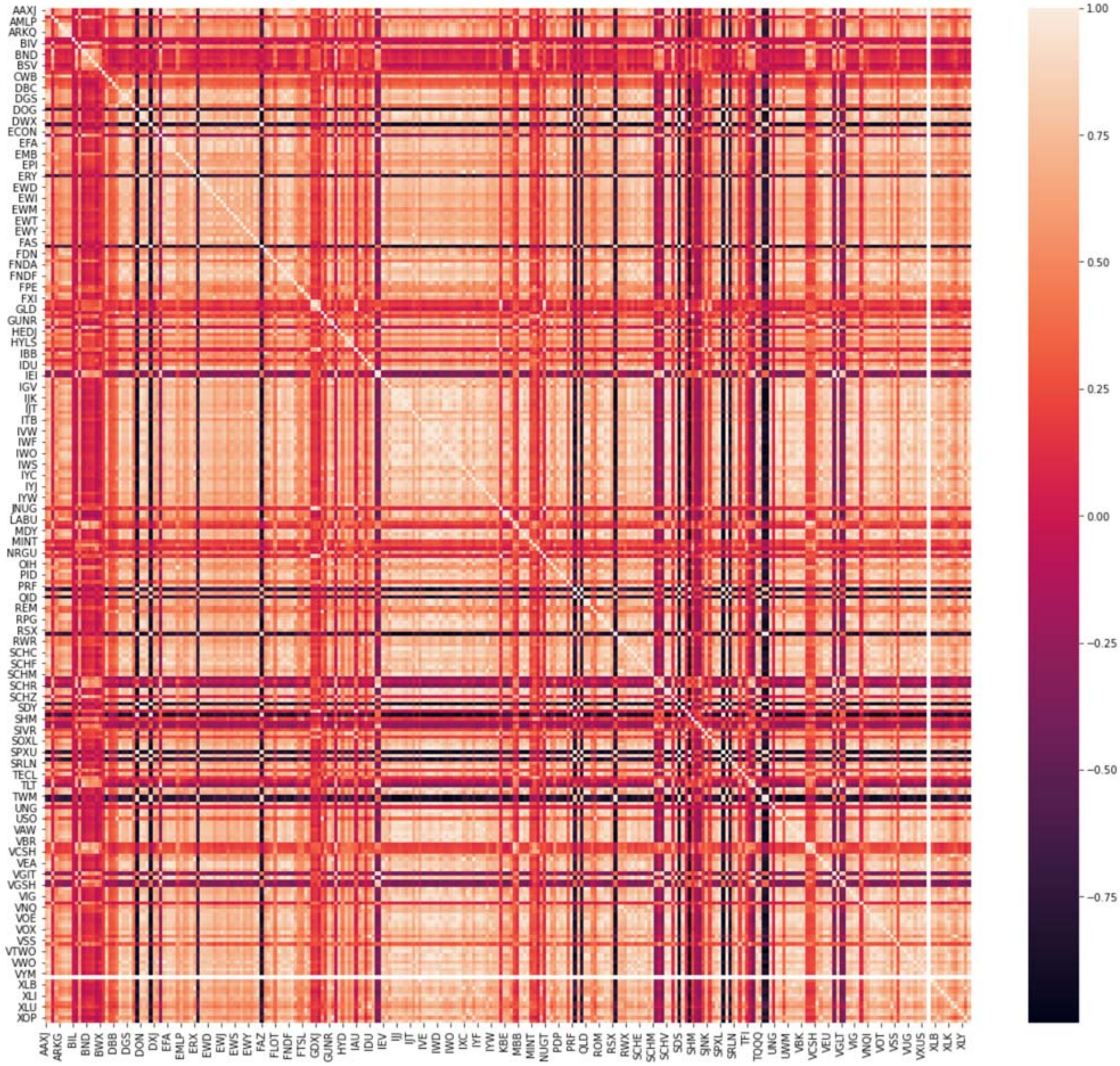
275 rows × 275 columns

In [14]:

```
plt.figure(figsize=(20, 18))

sns.heatmap(returns_df.corr())
```

Out[14]: <AxesSubplot:>



Reorder expected return to line up with covar matrix

In [15]:

```
exp_return_df = exp_return_df[covar_df.columns]
exp_return_df
```

Out[15]:

0.

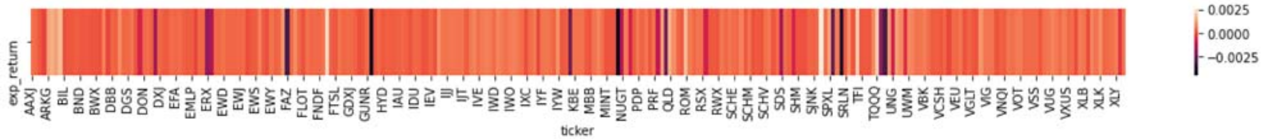
1 rows × 275 columns

In [16]:

```
plt.figure(figsize=(20, 1))

sns.heatmap(exp_return_df)
```

Out[16]: <AxesSubplot:xlabel='ticker'>



Optimization of portfolio

$$ \frac{\mu - r_{f}}{\sigma} $$

In [17]:

```
alpha_df = exp_return_df.T - exp_return_df[benchmark] # subtract market return
alpha_df = alpha_df[alpha_df['exp_return'] > 0] # filter to only positive alpha
alpha_sr = alpha_df['exp_return'].to_numpy()

covar_df = covar_df[covar_df.index.isin(alpha_df.index)][alpha_df.index] # filter to on
covar = covar_df.round(8).to_numpy()
alpha_df
```

Out[17]:

	exp_return
ticker	
ARKG	0.000737
ARKK	0.000952
ARKQ	0.000663
ARKW	0.001161
EWB	0.000151
...	...
VXF	0.000122
XHB	0.000095
XLF	0.000244
XLK	0.000435
XLY	0.000143

72 rows × 1 columns

In [18]:

```
from scipy.optimize import minimize

def sharpe_ratio(weights: np.ndarray, covar_matrix: np.ndarray, alpha_returns: np.ndarr
    # we are minimizing the negative to get a maximum
    objective = float(-weights.dot(alpha_returns) / np.sqrt(weights.dot(covar_matrix).d
    return objective

weights = np.ones_like(covar_df.columns)
constraints = ({'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1})
# bounds are in the form of (lower, upper)
bounds = [(0, None,) for i in range(len(weights))] # bounded by zero (no shorting)
portfolio = minimize(sharpe_ratio, weights, args=(covar, alpha_sr), bounds=bounds, cons
print(portfolio)
pd.concat([alpha_df.reset_index()['ticker'], pd.Series(portfolio.x)], axis=1)
```

```
     fun: -0.06085265524922186
```

```
     jac: array([1.38812675e-02, 9.41461464e-03, 1.18948054e-02, 9.31322575e-10,
       1.84806739e-02, 3.09494161e-02, 2.33212644e-02, 1.42520363e-02,
       4.73267585e-02, 2.08172752e-02, 2.94610788e-02, 1.70958620e-02,
       3.01936371e-02, 2.99345274e-02, 2.84514716e-02, 2.99250744e-02,
       2.35722531e-02, 2.19883663e-02, 2.41147727e-02, 2.19591996e-02,
       2.41578575e-02, 2.08603810e-02, 3.01806331e-02, 3.09032821e-02,
       2.72197351e-02, 2.87748617e-02, 2.39033229e-02, 2.51514842e-02,
       1.52493087e-02, 2.06028861e-02, 2.33625807e-02, 3.09900688e-02,
       2.63923644e-02, 1.45833902e-02, 2.12463276e-02, 1.73239671e-02,
       1.69470175e-02, 3.13429581e-02, 2.61251670e-02, 3.18776825e-02,
       1.76023240e-02, 2.08501280e-02, 2.99337823e-02, 2.39883917e-02,
       3.17256311e-02, 4.63670460e-02, 2.37985351e-02, 3.21435994e-02,
       3.43405479e-02, 7.79867782e-02, 3.28131714e-02, 4.52143834e-02,
       4.67967018e-02, 8.01876094e-02, 4.86461944e-02, 3.94068714e-02,
       2.96128448e-02, 2.80407281e-02, 2.50792508e-02, 1.38210640e-02,
       2.16491106e-02, 2.91270097e-02, 2.40033250e-02, 2.79429387e-02,
       3.01800743e-02, 2.14716694e-02, 2.36441591e-02, 2.82831155e-02,
       2.81788954e-02, 1.31751131e-02, 1.59034296e-02, 2.38143718e-02])
 message: 'Optimization terminated successfully'
    nfev: 1168
     nit: 16
    njev: 16
  status: 0
 success: True
       x: array([0.00000000e+00, 3.05311332e-16, 0.00000000e+00, 1.00000000e+00,
       7.33585235e-18, 3.20636402e-17, 4.38918638e-17, 1.69440454e-16,
       0.00000000e+00, 8.12026006e-17, 0.00000000e+00, 0.00000000e+00,
       4.02079279e-17, 1.14971339e-18, 0.00000000e+00, 3.38134489e-18,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.05153882e-16,
       0.00000000e+00, 4.37450519e-17, 0.00000000e+00, 2.58486405e-17,
       4.10355995e-17, 1.08435060e-17, 2.03011462e-17, 4.17870927e-17,
       0.00000000e+00, 1.51304497e-17, 2.25090783e-17, 2.58555820e-17,
       9.47335091e-17, 0.00000000e+00, 0.00000000e+00, 5.78758282e-17,
       0.00000000e+00, 4.50709033e-17, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 7.69083669e-18, 5.49711821e-17,
       0.00000000e+00, 0.00000000e+00, 1.97694851e-17, 1.88952973e-17,
       6.76026387e-17, 0.00000000e+00, 0.00000000e+00, 5.50358194e-17,
       8.29622597e-17, 0.00000000e+00, 2.21389864e-18, 3.36523914e-17,
       2.40062262e-17, 1.33753146e-17, 0.00000000e+00, 0.00000000e+00,
       9.81861957e-17, 4.14620416e-17, 0.00000000e+00, 5.06312859e-17,
       1.81451712e-17, 6.06454160e-18, 3.74122106e-17, 3.52789854e-17,
       0.00000000e+00, 9.88136046e-17, 2.20019920e-16, 6.56103123e-17])
```

Out[18]:

	ticker	0
0	ARKG	0.000000e+00
1	ARKK	3.053113e-16
2	ARKQ	0.000000e+00
3	ARKW	1.000000e+00
4	EYVB	7.335852e-18
...	...	...
67	VXF	3.527899e-17
68	XHB	0.000000e+00
69	ILF	9.881360e-17
70	XLK	2.200199e-16
71	XLY	6.561031e-17

72 rows × 2 columns

```python
import cvxpy as cv
# math basis for reformation to a standard convex quadratic program here:
# https://people.stat.sc.edu/sshen/events/backtesting/reference/maximizing%20the%20shar
# https://coral.ise.lehigh.edu/~ted/files/ie447/lectures/Lecture9.pdf

mu = alpha_df.to_numpy().T # asset expected return (we use alpha in this case)
w = cv.Variable((len(alpha_df), 1)) # optimization weights
k = cv.Variable((1, 1))
ret = mu @ w # port return = assets return * asset weight
rf0 = 0 # this could be substituted for benchmark return instead of calculating alpha b
sigma = covar_df.to_numpy()

g = cv.Variable(nonneg=True) # non-negative scalar

try:
    G = np.linalg.cholesky(sigma) # cholesky decomposition of covariance matrix
except:
    G = sqrtm(sigma)

risk = g ** 2
devconstraints = [cv.SOC(g, G.T @ w)]
constraints = [
    cv.sum(w) == 1 * k,
    k >= 0,
    mu @ w - rf0 * k == 1,
    w <= 1 * k, w * 1000 >= 0 # these 2 constraints = no shorting
]
constraints += devconstraints
objective = cv.Minimize(risk * 1000)
prob = cv.Problem(objective, constraints)
prob.solve(solver='ECOS') # options are ['ECOS', 'SCS', 'OSQP', 'CVXOPT']
weights = np.array(w.value / k.value, ndmin=2).T
weights = np.abs(weights) / np.sum(np.abs(weights)) # use absolute value since no short
weights_df = pd.concat([alpha_df.reset_index()['ticker'], pd.Series(weights[0])], axis=
weights_df
```

Out[19]:

	ticker	weight
0	ARKG	[illegible]
1	ARKK	[illegible]
2	ARKQ	[illegible]
3	ARKW	[illegible]
4	CWB	[illegible]
...	...	...
67	VXF	[illegible]
68	XHB	[illegible]
69	XLF	[illegible]
70	XLK	[illegible]
71	XLY	[illegible]

72 rows × 2 columns

In [20]:
```
# send weights to clipboard
pd.Series(portfolio.x).to_clipboard()
```

In [21]:
```
# send expected alphas to clipboard
alpha_df.to_clipboard()
```

In [22]:
```
# send covars to clipboard
covar_df.to_clipboard()
```

In [23]:
```
exp_yearly_alpha =  (1 + alpha_sr.dot(pd.Series(portfolio.x).to_numpy())) ** 250 - 1 #
exp_yearly_alpha
```

Out[23]: 0.33642835850736974

In [24]:
```
from math import sqrt
exp_yearly_risk = sqrt(pd.Series(portfolio.x).to_numpy().dot(covar_df.to_numpy()).dot(p
exp_yearly_risk
```

Out[24]: 0.3015812100862168

In [25]:
```
exp_yearly_alpha / exp_yearly_risk
```

Out[25]: 1.1155481417797573