```
In [26]: import numpy as np
         import pandas as pd
         from sqlalchemy import create_engine
         from joblib import load
         import yfinance as yf
         from datetime import datetime
         import os.path

         from alpaca.trading.client import TradingClient
         from alpaca.trading.requests import MarketOrderRequest, GetAssetsRequest, Ass
         from alpaca.trading.enums import OrderSide, TimeInForce
```

Build Feature Dataset

- Don't need much history. Start here in 2022.
- And don't need weekly returns (after computing momentum).

```
In [2]: server = 'fs.rice.edu'
        database = 'stocks'
        username = 'stocks'
        password = '6LAZH1'
        driver = 'SQL+Server'
        string = f"mssql+pyodbc://{username}:{password}@{server}/{database}"
        try:
            conn = create_engine(string + "?driver='SQL+Server'").connect()
        except:
            try:
                conn = create_engine(string + "?driver='ODBC+Driver+18+for+SQL+Server
            except:
                import pymssql
                string = f"mssql+pymssql://{username}:{password}@{server}/{database}"
                conn = create_engine(string).connect()
```

In [3]:
```
sep_weekly = pd.read_sql(
    """
    select date, ticker, closeadj, closeunadj, volume, lastupdated from sep_we
    where date >= '2022-01-01'
    order by ticker, date, lastupdated
    """,
    conn,
)
sep_weekly = sep_weekly.groupby(["ticker", "date"]).last()
sep_weekly = sep_weekly.drop(columns=["lastupdated"])

ret = sep_weekly.groupby("ticker", group_keys=False).closeadj.pct_change()
ret.name = "ret"

price = sep_weekly.closeunadj
price.name = "price"

volume = sep_weekly.volume
volume.name = "volume"
```

```
In [4]: ret_annual = sep_weekly.groupby("ticker", group_keys=False).closeadj.pct_chan
        ret_monthly = sep_weekly.groupby("ticker", group_keys=False).closeadj.pct_cha
        mom = (1 + ret_annual) / (1 + ret_monthly) - 1
        mom.name = "mom"
```

```
In [5]: weekly = pd.read_sql(
            """
            select date, ticker, pb, marketcap, lastupdated from weekly
            where date>='2022-01-01'
            order by ticker, date, lastupdated
            """,
            conn,
        )
        weekly = weekly.groupby(["ticker", "date"]).last()
        weekly = weekly.drop(columns=["lastupdated"])

        pb = weekly.pb
        pb.name = "pb"
        marketcap = weekly.marketcap
        marketcap.name = "marketcap"
```

In [6]:
```python
sf1 = pd.read_sql(
    """
    select datekey as date, ticker, assets, netinc, equity, lastupdated from
    where datekey>='2022-01-01' and dimension='ARY' and assets>0 and equity>0
    order by ticker, datekey, lastupdated
    """,
    conn,
)
sf1 = sf1.groupby(["ticker", "date"]).last()
sf1 = sf1.drop(columns=["lastupdated"])

# change dates to Fridays
from datetime import timedelta
sf1 = sf1.reset_index()
sf1.date =sf1.date.map(
    lambda x: x + timedelta(4 - x.weekday())
)
sf1 = sf1.set_index(["ticker", "date"])
sf1 = sf1[~sf1.index.duplicated()]

assets = sf1.assets
assets.name = "assets"
netinc = sf1.netinc
netinc.name = "netinc"
equity = sf1.equity
equity.name = "equity"

equity = equity.groupby("ticker", group_keys=False).shift()
roe = netinc / equity
```

In [7]:

```
df = pd.concat(
    (
        mom,
        volume,
        price,
        pb,
        marketcap,
        roe,
        assetgr
        ),
        axis=1
    )
df["roe"] = df.groupby("ticker", group_keys=False).roe.ffill()
df["assetgr"] = df.groupby("ticker", group_keys=False).assetgr.ffill()

df = df.reset_index()
df.date = df.date.astype(str)
df = df[df.date==df.date.max()]
df = df[df.price >= 5]
df = df.dropna()

features = [
    "mom",
    "volume",
    "pb",
    "marketcap",
    "roe",
    "assetgr"
]
```

```
In [8]: industries = pd.read_sql(
            """
            select ticker, famaindustry as industry from tickers
            """,
            conn,
        )
        industries["industry"] = industries.industry.fillna("Almost Nothing")
        df = df.merge(industries, on="ticker", how="left")
        df = df.dropna()
```

```
In [9]: for x in features:
            df[f"{x}_industry"] = df.groupby(
                ["industry"],
                group_keys=False
            )[x].apply(
                lambda x: x - x.median()
            )

        features += [f"{x}_industry" for x in features]
```

```
In [10]: for f in features:
             df[f] = df[f].rank(pct=True)
```

Load Model and Predict

In [11]:

```python
model = load("mymodel.joblib")
df["predict"] = model.predict(df[features])
```

Best and worst stocks

- Best stocks must be tradable
- Worst stocks must be tradable and shortable

```
In [12]: with open("keys.txt", "r") as f:
             keys = f.readlines()

         key, secret_key = [x.strip() for x in keys]
         trading_client = TradingClient(key, secret_key, paper=True)

         search_params = GetAssetsRequest(asset_class=AssetClass.US_EQUITY)
         assets = trading_client.get_all_assets(search_params)
         tradable = [x.symbol for x in assets if x.tradable]
         shortable = [x.symbol for x in assets if x.shortable]
```

In [13]:

```
numstocks = 50

df = df.sort_values(by="predict", ascending=False)
best = df[["ticker", "predict"]].copy().reset_index(drop=True)
best = best[best.ticker.isin(tradable)].iloc[:numstocks]

df = df.sort_values(by="predict", ascending=True)
worst = df[["ticker", "predict"]].copy().reset_index(drop=True)
worst = worst[worst.ticker.isin(shortable)].iloc[:numstocks]
```

In [14]:

```
best
```

Out[14]:

	ticker	predict
0	SMCI	51.394069
1	QSR	51.306917
2	MPWR	51.278958
3	FERG	51.278958
4	ODFL	51.278958
5	FAST	51.272102
6	TT	51.268591
7	SNPS	51.268591
8	SCCO	51.268591
9	LULU	51.268591
10	WST	51.268043
11	CCI	51.266315
12	A	51.265236
13	MCHP	51.265236
14	VEEV	51.265101
15	BR	51.264289

In [15]:

```
worst
```

Out[15]:

	ticker	predict
3	EIGR	39.167009
8	PRPO	42.381558
14	BODY	44.111180
16	KPLT	44.205348
18	CALC	44.351845
19	XOS	44.567180
22	AIRT	44.969343
23	ONCT	45.006040
24	ECOR	45.024591
25	SKLZ	45.068362
29	AEYE	45.264590
32	NVNO	45.458468
33	IPWR	45.482758
34	LEE	45.507330
37	CRVO	45.558953
39	TSE	45.651877

Close unwanted positions

In [20]:

```python
positions = trading_client.get_all_positions()
positions = {x.symbol: float(x.qty) for x in positions}
positions_to_close = [
    symbol for symbol in positions
    if (symbol not in best.ticker.to_list())
    and (symbol not in worst.ticker.to_list())
]

for symbol in positions_to_close:
    qty = positions[symbol]
    order=MarketOrderRequest(
        symbol=symbol,
        qty=abs(qty),
        side=OrderSide.BUY if qty<0 else OrderSide.SELL,
        time_in_force=TimeInForce.DAY
    )
    _ = trading_client.submit_order(order)
```

Rebalance SPY

In [21]:

```python
price = yf.download("SPY", start=2024, progress=False)["Close"].iloc[-1].item

account = trading_client.get_account()
equity = float(account.equity)
qty = int(equity / price)
qty -= positions["SPY"] if "SPY" in positions else 0

if qty != 0:
    order = MarketOrderRequest(
        symbol="SPY",
        qty=abs(qty),
        side=OrderSide.BUY if qty>0 else OrderSide.SELL,
        time_in_force=TimeInForce.DAY
    )
    _ = trading_client.submit_order(order)
```

Trade best stocks

```
In [22]: symbols = best.ticker.to_list()
         prices = yf.download(symbols, start=2024)["Close"].iloc[-1]
         symbols = [s for s in symbols if not np.isnan(prices[s])]
         dollars = 0.4 * equity / numstocks
         for symbol in symbols:
             price = prices[symbol]
             qty = int(dollars / price)
             qty -= positions[symbol] if symbol in positions else 0
             if qty != 0:
                 try:
                     order = MarketOrderRequest(
                         symbol=symbol,
                         qty=abs(qty),
                         side=OrderSide.BUY if qty>0 else OrderSide.SELL,
                         time_in_force=TimeInForce.DAY
                     )
                     _ = trading_client.submit_order(order)
                 except Exception as error:
                     print("An error occurred:", error)
```

```
[*********************100%%**********************]  50 of 50 complete
d
```

Trade worst stocks

```
In [24]: symbols = worst.ticker.to_list()
         prices = yf.download(symbols, start=2024)["Close"].iloc[-1]
         symbols = [s for s in symbols if not np.isnan(prices[s])]
         for symbol in symbols:
             price = prices[symbol]
             qty = - int(dollars / price)
             qty -= positions[symbol] if symbol in positions else 0
             if qty != 0:
                 try:
                     order = MarketOrderRequest(
                         symbol=symbol,
                         qty=abs(qty),
                         side=OrderSide.BUY if qty>0 else OrderSide.SELL,
                         time_in_force=TimeInForce.DAY
                     )
                     _ = trading_client.submit_order(order)
                 except Exception as error:
                     print("An error occurred:", error)
```

```
[*********************100%%**********************]  50 of 50 complete
d
```

Save data

In [27]:

```python
today = datetime.strftime(datetime.today(), "%Y-%m-%d")
account = trading_client.get_account()
equity = float(account.equity)
if os.path.isfile("equity.csv"):
    d = pd.read_csv("equity.csv", index_col="date")
    d.loc[today] = equity
else:
    d = pd.Series({today: equity})
    d.name = "equity"
    d.index.name = "date"
d.to_csv("equity.csv")
```

```
In [ ]: positions = trading_client.get_all_positions()
        d = pd.DataFrame([x.qty for x in positions], index=[x.symbol for x in positio
        d["date"] = today
        d.index.name = "symbol"
        d = d.reset_index()
        if os.path.isfile("positions.csv"):
            d0 = pd.read_csv("equity.csv")
            d = pd.concat((d0, d))
        d.to_csv("positions.csv", index=False)
```