In [138]:

```
import seaborn as sns
sns.set_style("whitegrid")
```

Introduction to the Course

MGMT 638: Data-Driven Investments: Equity

Kerry Back, Rice University

- Idea + data $\rightarrow$ backtest
- Look-ahead bias: can we reasonably backtest the strategy "buy electric car companies whose name starts with T?"
- parameters to this strategy: type of company, first letter of name
- We can backtest in a loop, updating once per year for example:
    - Find the type of company and the first letter of name that did best in the past n years
    - Buy that company and hold for a year
    - Update each year: find the new best company/first letter and hold it for a year

- We have data on past prices and company financial statements (also more, but we won't get to it)
- We can code (ChatGPT can maybe help) to write backtest routines
- Create visualizations to compare strategies and write reports (ChatGPT can help)

Elements of tests

- Past average return
- Sharpe ratio
- CAPM alpha and information ratio
- Fama-French alpha and information ratio
- Maximum drawdown
- Transactions costs (including shorting fees)
- Correlation with other strategies
- Tracking error relative to a benchmark

Universe of stocks

- Large cap or small cap or mid cap or some of all?
- Industries: do we want to bet on industries or match industry weights to a benchmark?
- Value vs growth, etc.
- Our goal could be to find the best possible strategy without any constraints or we might be constrained to find the best strategy within mid-cap energy, for example.
- Different strategies may work better or worse depending on the universe of stocks we can consider.

Example for today

- Do moving average strategies work?
- Get data from Yahoo Finance for Tesla
- Get data from a JGSB SQL database for all NYSE and Nasdaq tickers

Get data from Yahoo Finance

In [139]:

```python
import yfinance as yf
import pandas as pd


data = yf.download('TSLA', start="1970-01-01")["Adj Close"]
data = pd.DataFrame(data)
data.columns = ["price"]
data.head()
```

```
[*********************100%***********************]  1 of 1 completed
```

Out[139]:

	price
Date	
2010-06-29 00:00:00-04:00	1.592667
2010-06-30 00:00:00-04:00	1.588667
2010-07-01 00:00:00-04:00	1.464000
2010-07-02 00:00:00-04:00	1.280000
2010-07-06 00:00:00-04:00	1.074000

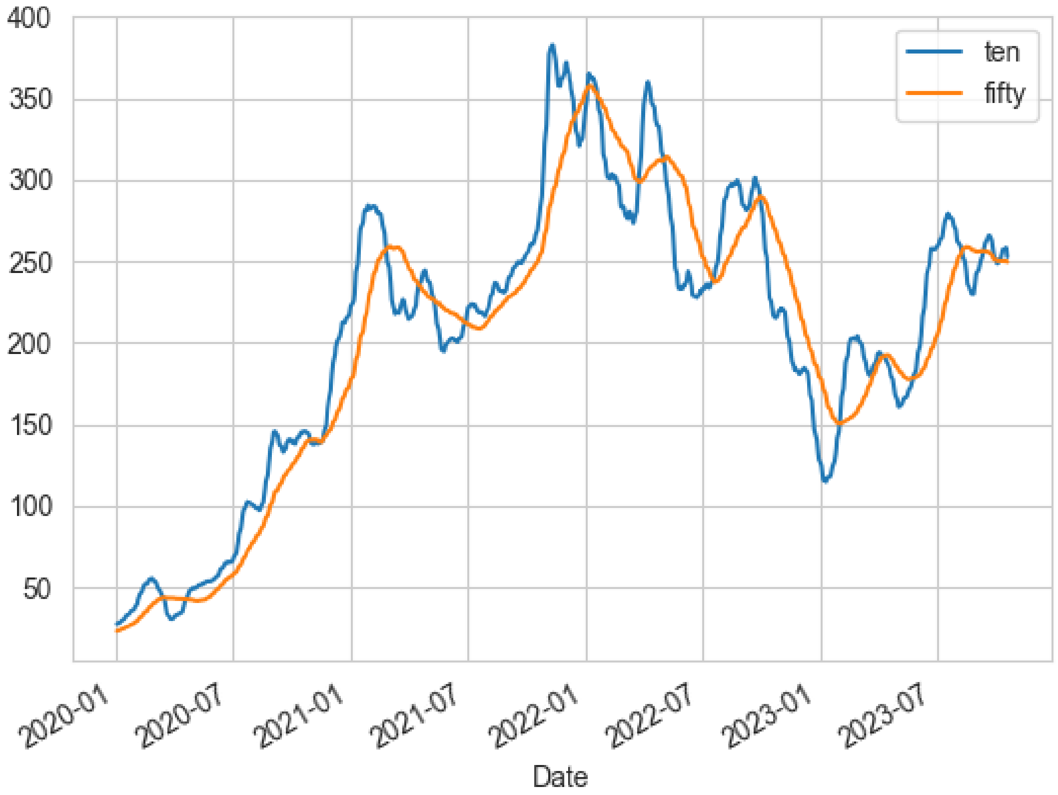
Compute moving averages

In [140]:

```
data["ten"] = data.price.rolling(10).mean()
data["fifty"] = data.price.rolling(50).mean()
data[["ten", "fifty"]].loc["2020-01-01":].plot()
```

Out[140]:

```
<AxesSubplot: xlabel='Date'>
```



Compute returns and lag moving averages

In [141]:

```python
data[["ten", "fifty"]] = data[["ten", "fifty"]].shift()
data["ret"] = data.price.pct_change()
data = data.dropna()
data.head()
```

Out[141]:

	price	ten	fifty	ret
Date				
2010-09-09 00:00:00-04:00	1.380667	1.351333	1.322240	-0.009090
2010-09-10 00:00:00-04:00	1.344667	1.356733	1.318000	-0.026074
2010-09-13 00:00:00-04:00	1.381333	1.359533	1.313120	0.027268
2010-09-14 00:00:00-04:00	1.408000	1.366333	1.311467	0.019305
2010-09-15 00:00:00-04:00	1.465333	1.374667	1.314027	0.040719

Compute trade indicator

In [142]:

```python
data["long"] = data.ten < data.fifty
data.head()
```

Out[142]:

	price	ten	fifty	ret	long
Date					
2010-09-09 00:00:00-04:00	1.380667	1.351333	1.322240	-0.009090	False
2010-09-10 00:00:00-04:00	1.344667	1.356733	1.318000	-0.026074	False
2010-09-13 00:00:00-04:00	1.381333	1.359533	1.313120	0.027268	False
2010-09-14 00:00:00-04:00	1.408000	1.366333	1.311467	0.019305	False
2010-09-15 00:00:00-04:00	1.465333	1.374667	1.314027	0.040719	False

Compute returns of moving average strategy

In [143]:

```python
data["ma_ret"] = data.long * data.ret
data.head()
```

Out[143]:

	price	ten	fifty	ret	long	ma_ret
Date						
2010-09-09 00:00:00-04:00	1.380667	1.351333	1.322240	-0.009090	False	-0.0
2010-09-10 00:00:00-04:00	1.344667	1.356733	1.318000	-0.026074	False	-0.0
2010-09-13 00:00:00-04:00	1.381333	1.359533	1.313120	0.027268	False	0.0
2010-09-14 00:00:00-04:00	1.408000	1.366333	1.311467	0.019305	False	0.0
2010-09-15 00:00:00-04:00	1.465333	1.374667	1.314027	0.040719	False	0.0

What tests do we want to do?

- Start by calculating average returns - multiply by 252 to annualize
- Then look at plot of compound returns - log scale works better for long time period
- Compute Sharpe ratios
- CAPM alphas, ...

Mean returns

In [144]:

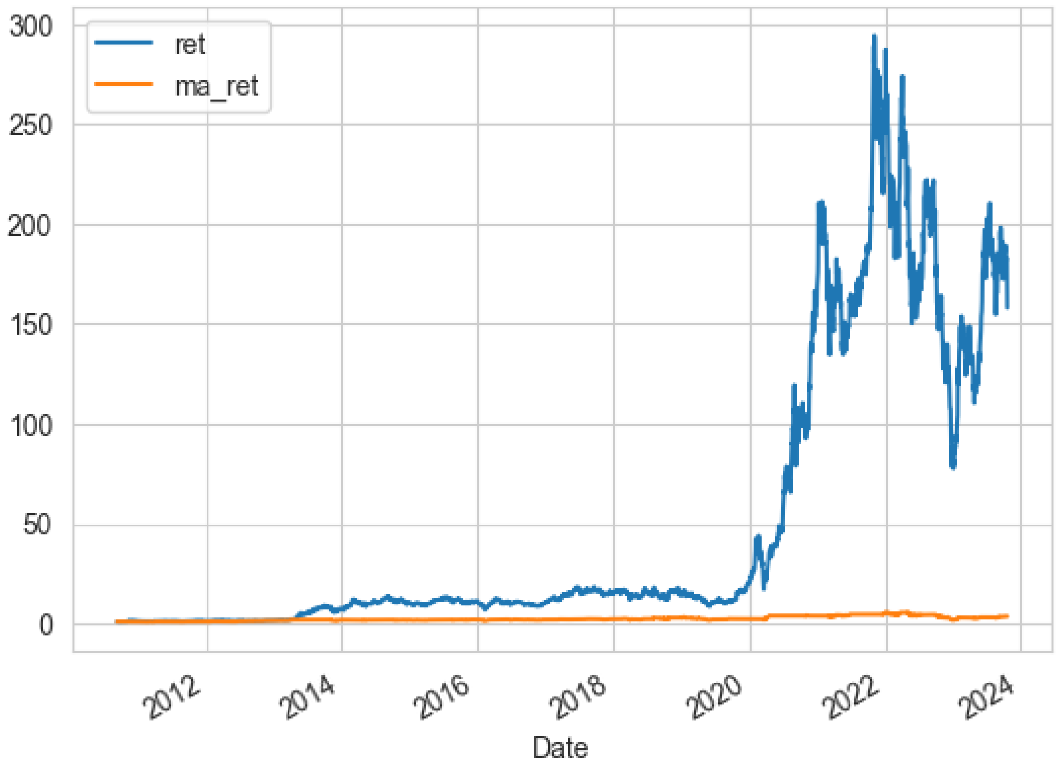
```
print(f"buy and hold mean return is {252*data.ret.mean():.2%} annualized")
print(f"moving average mean return is {252*data.ma_ret.mean():.2%} annualized
```

```
buy and hold mean return is 54.68% annualized
moving average mean return is 16.94% annualized
```

Compound return plots

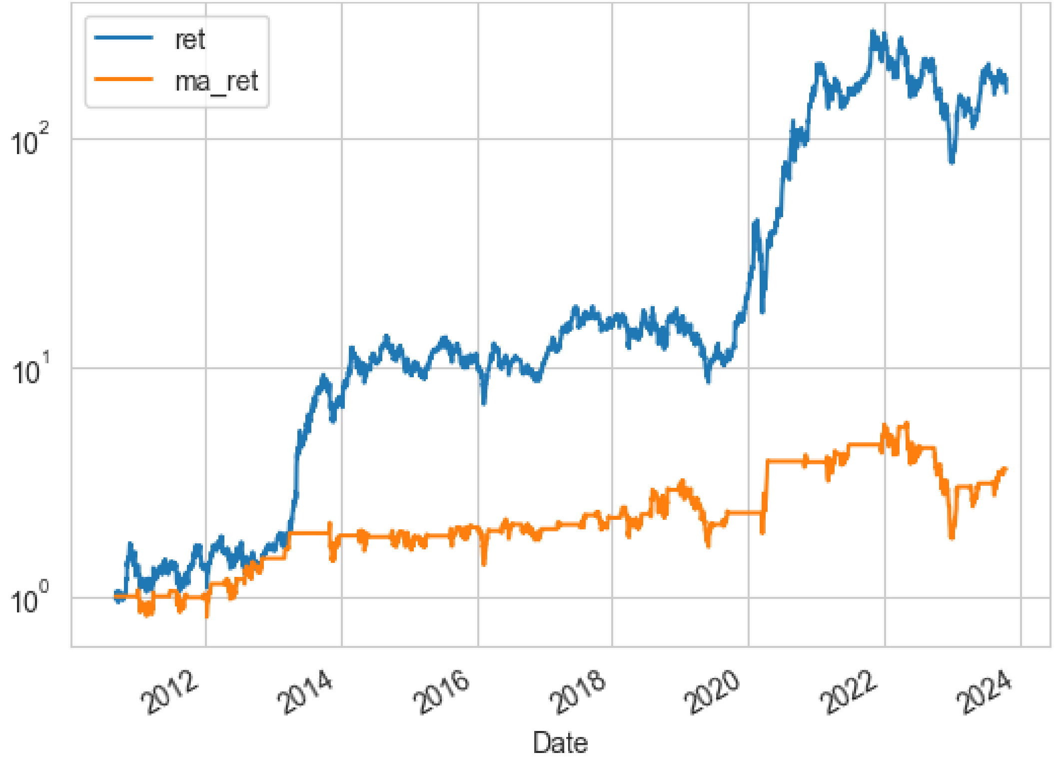
In [145]:
```
(1+data[["ret", "ma_ret"]]).cumprod().plot()
```

Out[145]: <AxesSubplot: xlabel='Date'>



In [146]:
```
(1+data[["ret", "ma_ret"]]).cumprod().plot(logy=True)
```

Out[146]: <AxesSubplot: xlabel='Date'>



Sharpe ratios

- Sharpe ratio is expected return minus risk-free rate / standard deviation
- We'll skip the risk-free rate
- Annualize mean return by multiplying by 252
- Annualize variance by multiplying by 252
- $\Rightarrow$ annualize standard deviation by multiplying by $\sqrt{252}$
- $\Rightarrow$ annualize Sharpe ratio by multiplying by $\sqrt{252}$

In [147]:

```
import numpy as np
sharpe = np.sqrt(252) * (data.ret.mean() / data.ret.std())
ma_sharpe = np.sqrt(252) * (data.ma_ret.mean() / data.ma_ret.std())
print(f"Sharpe ratio of buy and hold strategy is {sharpe:.2%} annualized")
print(f"Sharpe ratio of moving average strategy is {ma_sharpe:.2%} annualized
```

```
Sharpe ratio of buy and hold strategy is 96.35% annualized
Sharpe ratio of moving average strategy is 44.55% annualized
```

Multiple stocks

In [148]:

```python
tickers = ["PG", "WMT", "CVX", "F", "MSFT"]
data = yf.download(tickers, start="2000-01-01")["Adj Close"]
data = pd.DataFrame(data.stack())
data.columns = ["price"]
data.index.names = ["date", "ticker"]
data.head()
```

```
[*********************100%***********************]  5 of 5 completed
```

Out[148]:

date	ticker	price
2000-01-03 00:00:00-05:00	**CVX**	17.508459
	F	13.405164
	MSFT	36.205597
	PG	28.608192
	WMT	43.717712

In [149]:

```python
data["ten"] = data.groupby("ticker", group_keys=False).price.apply(
    lambda x: x.rolling(10).mean()
)
data["fifty"] = data.groupby("ticker", group_keys=False).price.apply(
    lambda x: x.rolling(50).mean()
)
data["ten"] = data.groupby("ticker", group_keys=False).ten.shift()
data["fifty"] = data.groupby("ticker", group_keys=False).fifty.shift()
```

In [150]:

```
data["ret"] = data.groupby("ticker", group_keys=False).price.pct_change()
data["long"] = data.ten > data.fifty
data["ma_ret"] = data.long * data.ret
data = data.dropna()
data.head()
```

Out[150]:

		price	ten	fifty	ret	long	ma_ret
date	**ticker**						
2000-03-15 00:00:00-05:00	**CVX**	17.722534	16.635165	17.040949	0.012821	False	0.0
	F	11.391672	10.794621	12.190062	0.074695	False	0.0
	MSFT	29.624529	29.616764	31.569343	0.002628	False	0.0
	PG	15.656404	18.598676	25.318128	0.049327	False	0.0
	WMT	33.740608	32.119635	36.882368	0.079891	False	0.0

Equally weighted returns

- Equally weighted strategies are never buy and hold, because with buy and hold weights increase on stocks that did relatively better.
- Value weighted is buy and hold.

In [151]:

```
rets = data.groupby("date").ret.mean()
ma_rets = data.groupby("date").ma_ret.mean()

print(f"equally weighted mean return is {252*rets.mean():.2%} annualized")
print(f"equally weighted moving average mean return is {252*ma_rets.mean():.2
```

```
equally weighted mean return is 11.78% annualized
equally weighted moving average mean return is 5.72% annualized
```

Exercise

Look at different sets of stocks and different moving averages and test strategies.