In [ ]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import os

import statsmodels.api as sm
import scipy
from scipy.stats import multivariate_normal
import itertools

import tensorflow as tf
import torch

# from google.colab import drive
# drive.mount('/content/drive')
```

In [ ]:
```python
project_path = '/Users/niehuapeng/Desktop/IPCA'

data_path = os.path.join(project_path, 'Data')
```

In [ ]:
```python
os.listdir(data_path)
```

Out[ ]:
```
['Bond Stock Monthly Data.parquet',
 'US 10-Year Yield.csv',
 'UST Curve Data.csv',
 'LQD.csv',
 'VIX.csv']
```

In [ ]:
```python
data = pd.read_parquet(os.path.join(data_path, 'Bond Stock Monthly Data.parq

data
```

Out[ ]:

	Pricing_Date	Index_Name	Cusip	ISIN	Description	IC
0	2008-07-01	C0A0	00184AAB1	US00184AAB17	TIME WARNER INC	
1	2008-07-01	C0A0	00184AAC9	US00184AAC99	AOL TIME WARNER	
2	2008-07-01	C0A0	00184AAF2	US00184AAF21	TIME WARNER INC	
3	2008-07-01	C0A0	00184AAG0	US00184AAG04	AOL TIME WARNER	
4	2008-07-01	C0A0	00209TAB1	US00209TAB17	COMCAST CABLE CO	
...	...	...	...	...	...	
720217	2024-09-01	C0A0	98978VAU	US98978VAU70	Zoetis Inc.	
720218	2024-09-01	C0A0	98978VAV	US98978VAV53	Zoetis Inc.	
720219	2024-09-01	H0A0	98980BAA	US98980BAA17	ZipRecruiter Inc	
720220	2024-09-01	H0A0	98981BAA	US98981BAA08	Zoominfo Technologies Llc /Zoominfo Financial ...	
720221	2024-09-01	C0A0	00206RJR	XS2091666748	AT&T Inc	

720222 rows × 170 columns

In [ ]:

```
vix = pd.read_csv(os.path.join(data_path, 'VIX.csv'))

vix
```

Out[ ]:

	Price	Adj Close	Close	High	
0	Ticker	^VIX	^VIX	^VIX	
1	Date	NaN	NaN	NaN	
2	2005-01-03	14.079999923706055	14.079999923706055	14.229999542236328	
3	2005-01-04	13.979999542236328	13.979999542236328	14.449999809265137	13
4	2005-01-05	14.09000015258789	14.09000015258789	14.09000015258789	13
...	...	...	...	...	
5030	2024-12-24	14.270000457763672	14.270000457763672	17.040000915527344	14
5031	2024-12-26	14.729999542236328	14.729999542236328	15.930000305175781	14
5032	2024-12-27	15.949999809265137	15.949999809265137	18.450000762939453	15
5033	2024-12-30	17.399999618530273	17.399999618530273	19.219999313354492	16
5034	2024-12-31	17.350000381469727	17.350000381469727	17.809999465942383	1

5035 rows × 7 columns

In [ ]:
```
from scipy.stats.mstats import winsorize

def remove_outliers(df):
    df = df.replace(-99.990000, np.nan).dropna()
    x = winsorize(df.iloc[:,0], limits=[0.001, 0.001])
    y = winsorize(df.iloc[:,1], limits=[0.0001, 0.0001])
    df = pd.DataFrame({df.iloc[:,0].name: x,
                       df.iloc[:,1].name: y})
    return df
```

In [ ]:
```
data.replace(-99.990000, np.nan, inplace=True)
```

In [ ]:
```
date_columns = ['Pricing_Date', 'Maturity', 'Stock_Price_Date', 'REPORTING_C

for col in date_columns:
    data[f'{col}'] = pd.to_datetime(data[f'{col}'], format='mixed')
```

In [ ]:
```
features = data[['Pricing_Date', 'Index_Name', 'ISIN', 'Description', 'Matur
                 'Sector_Level_1', 'Sector_Level_2', 'Sector_Level_3', 'Sect
                 'Return']].copy()

features
```

Out[ ]:

	Pricing_Date	Index_Name	ISIN	Description	Maturity	Sec
0	2008-07-01	C0A0	US00184AAB17	TIME WARNER INC	2011-04-15	
1	2008-07-01	C0A0	US00184AAC99	AOL TIME WARNER	2031-04-15	
2	2008-07-01	C0A0	US00184AAF21	TIME WARNER INC	2012-05-01	
3	2008-07-01	C0A0	US00184AAG04	AOL TIME WARNER	2032-05-01	
4	2008-07-01	C0A0	US00209TAB17	COMCAST CABLE CO	2022-11-15	
...	...	...	...	...	...	
720217	2024-09-01	C0A0	US98978VAU70	Zoetis Inc.	2025-11-14	
720218	2024-09-01	C0A0	US98978VAV53	Zoetis Inc.	2032-11-16	
720219	2024-09-01	H0A0	US98980BAA17	ZipRecruiter Inc	2030-01-15	
720220	2024-09-01	H0A0	US98981BAA08	Zoominfo Technologies Llc /Zoominfo Financial ...	2029-02-01	
720221	2024-09-01	C0A0	XS2091666748	AT&T Inc	2050-03-01	

720222 rows × 10 columns

In [ ]:
```
features.sort_values(by = ['Pricing_Date', 'ISIN'], inplace =True)
```

In [ ]:
```
# compute issue date

features['Issue_Date'] = features.groupby('ISIN')['Pricing_Date'].transform(
```

In [ ]:
```
# compute bond age

features['Bond_Age_Percentage'] = (features['Pricing_Date'] - features['Issu
                                   (features['Maturity'] - features['Issue_Da

features['Bond_Age_Years'] = (features['Pricing_Date'] - features['Issue_Dat

features.tail()
```

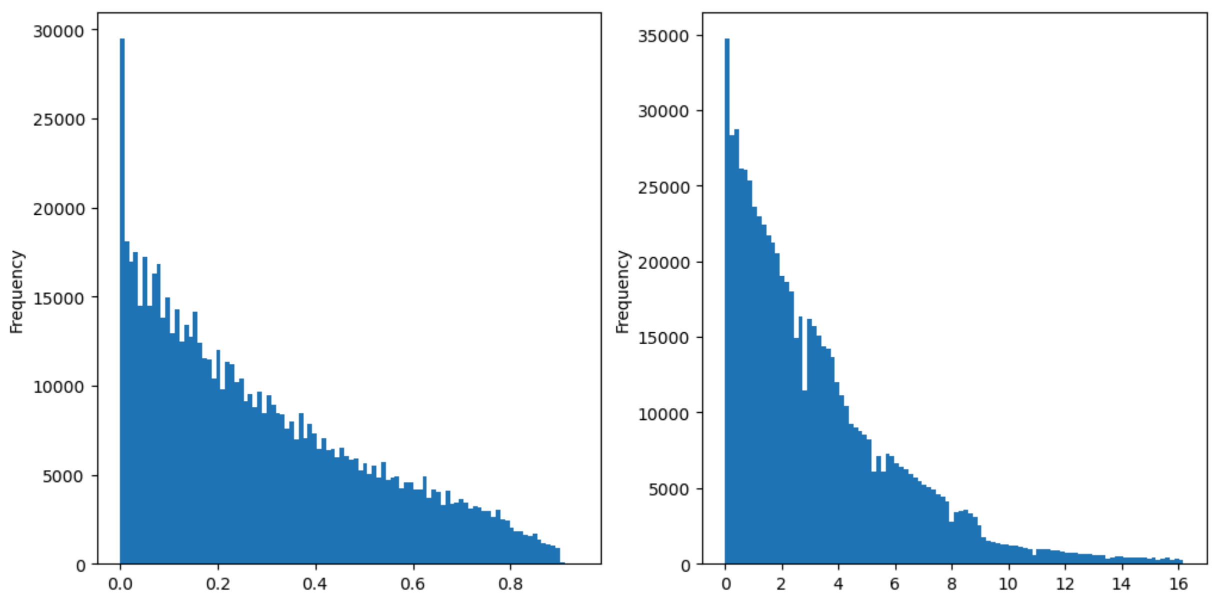
Out[ ]:

	Pricing_Date	Index_Name	ISIN	Description	Maturity	Sect
720217	2024-09-01	C0A0	US98978VAU70	Zoetis Inc.	2025-11-14	
720218	2024-09-01	C0A0	US98978VAV53	Zoetis Inc.	2032-11-16	
720219	2024-09-01	H0A0	US98980BAA17	ZipRecruiter Inc	2030-01-15	
720220	2024-09-01	H0A0	US98981BAA08	Zoominfo Technologies Llc /Zoominfo Financial ...	2029-02-01	
720221	2024-09-01	C0A0	XS2091666748	AT&T Inc	2050-03-01	

In [ ]:

```
fig, axes = plt.subplots(1,2, figsize = (12,6))

features['Bond_Age_Percentage'].plot(kind = 'hist', bins = 100, ax=axes[0])
features['Bond_Age_Years'].plot(kind = 'hist', bins = 100, ax=axes[1])
```

Out[ ]: <Axes: ylabel='Frequency'>



In [ ]:

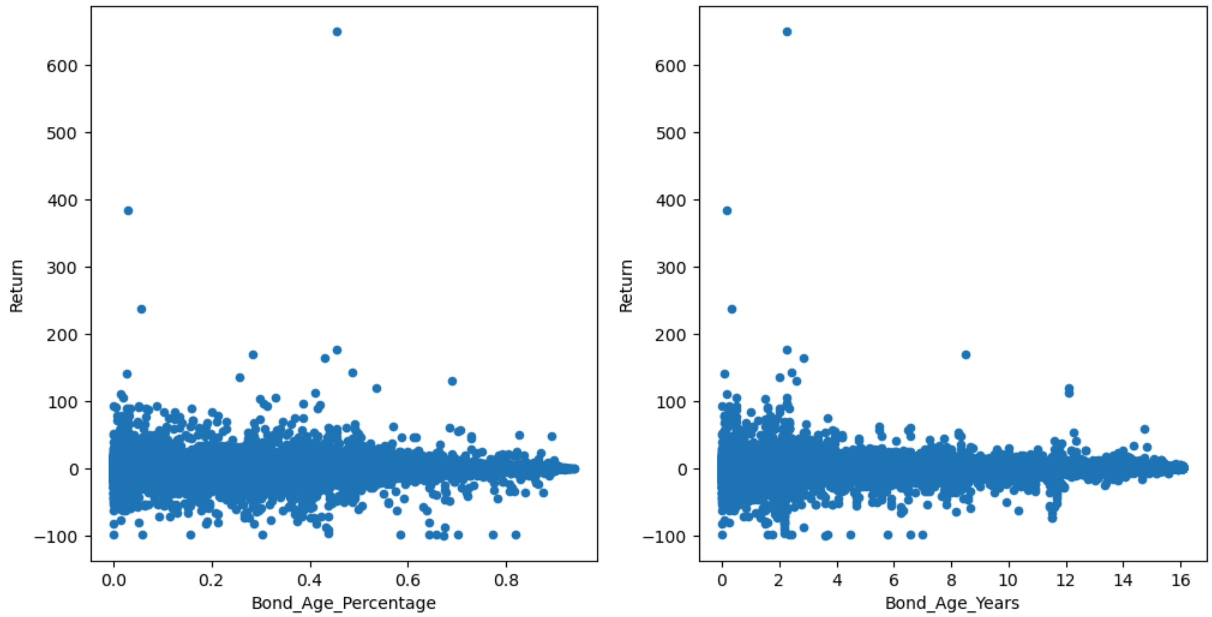
```
features.replace(-99.990000, np.nan)[['Bond_Age_Percentage', 'Bond_Age_Years
```

Out[ ]:

	Bond_Age_Percentage	Bond_Age_Years	Return
Bond_Age_Percentage	1.000000	0.628640	-0.029566
Bond_Age_Years	0.628640	1.000000	-0.009658
Return	-0.029566	-0.009658	1.000000

In [ ]:
```
fig, axes = plt.subplots(1,2, figsize = (12,6))

features.replace(-99.990000, np.nan).dropna().plot(kind = 'scatter', x='Bond

features.replace(-99.990000, np.nan).dropna().plot(kind = 'scatter', x='Bond
```

Out[ ]: <Axes: xlabel='Bond_Age_Years', ylabel='Return'>



In [ ]:
```
features['Coupon'] = data['Coupon']
```

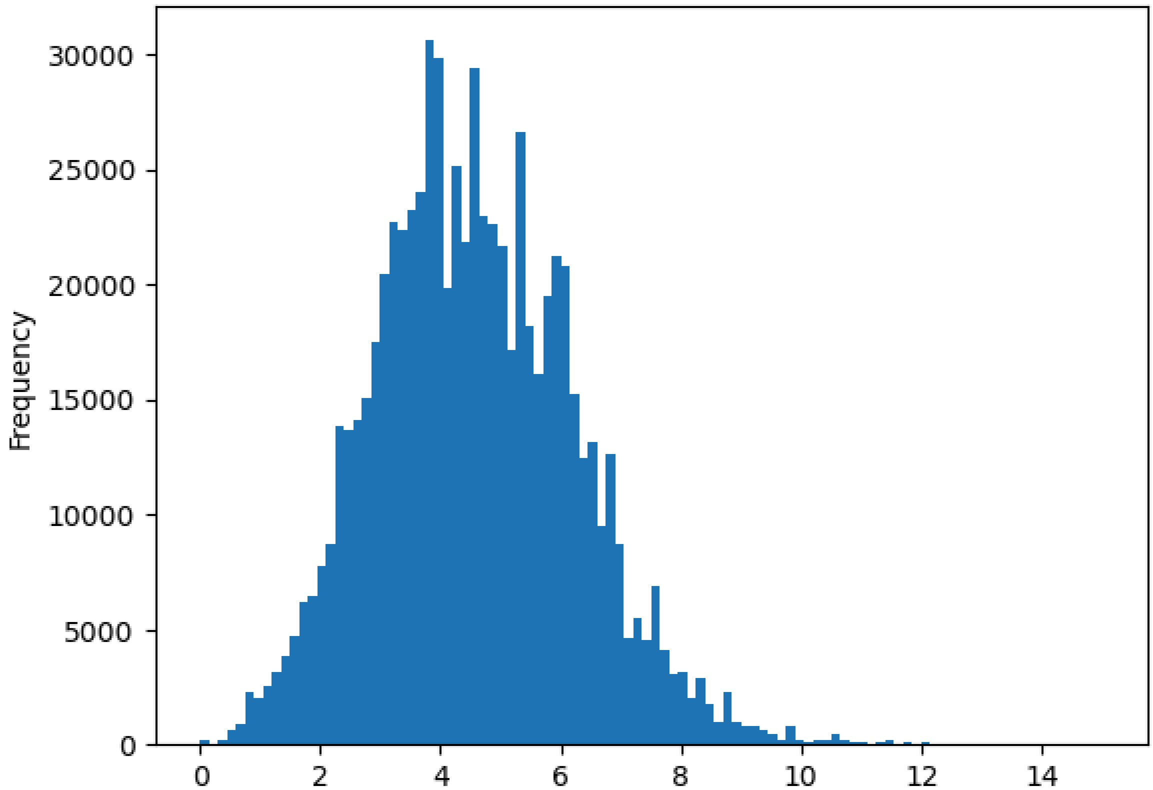
In [ ]:
```
features[['Coupon']].describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
Coupon	720222.0	4.550372	1.668525	0.0	3.375	4.45	5.7	15.0

In [ ]:
```
features['Coupon'].plot(kind = 'hist', bins= 100)
```

Out[ ]: <Axes: ylabel='Frequency'>



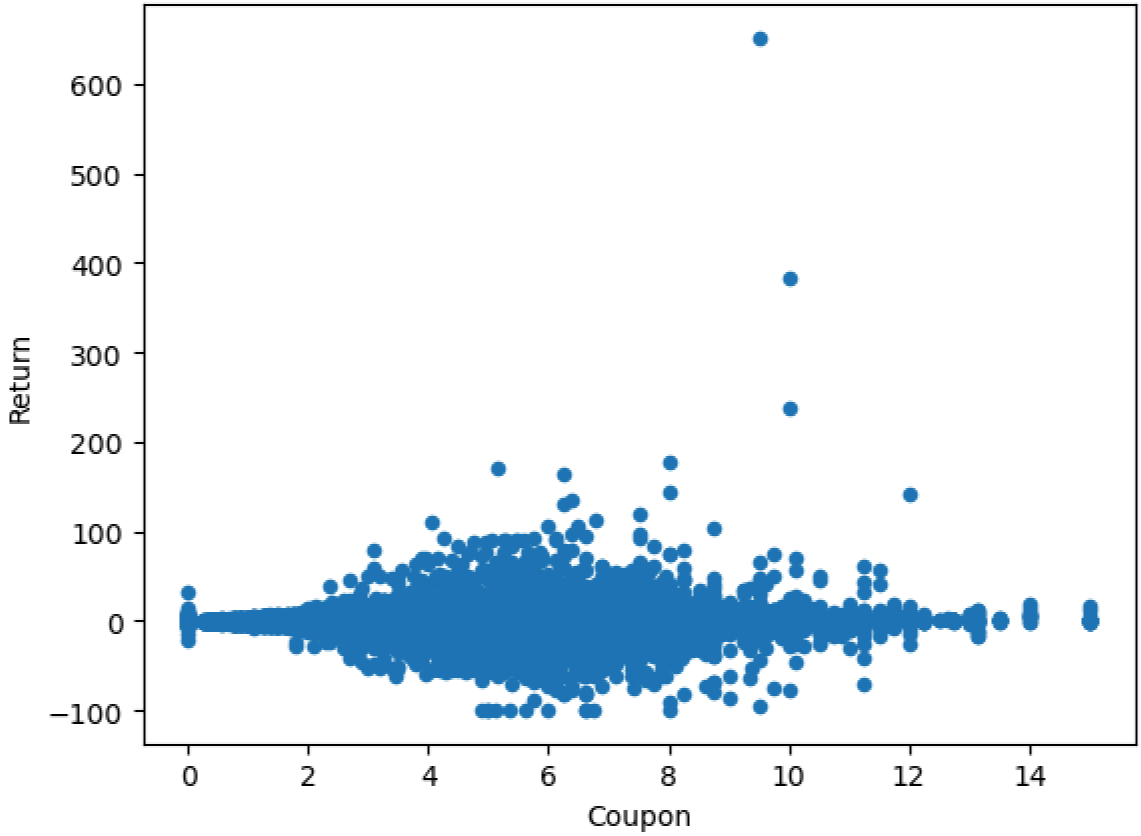
In [ ]:
```
features.replace(-99.990000, np.nan)[['Coupon', 'Return']].corr()
```

Out[ ]:

	Coupon	Return
Coupon	1.000000	0.055895
Return	0.055895	1.000000

In [ ]:
```
features.replace(-99.990000, np.nan).dropna().plot(kind='scatter', x='Coupor
```

Out[ ]:
```
<Axes: xlabel='Coupon', ylabel='Return'>
```



In [ ]:
```
features['Face_Value'] = data['ParAmount']
```

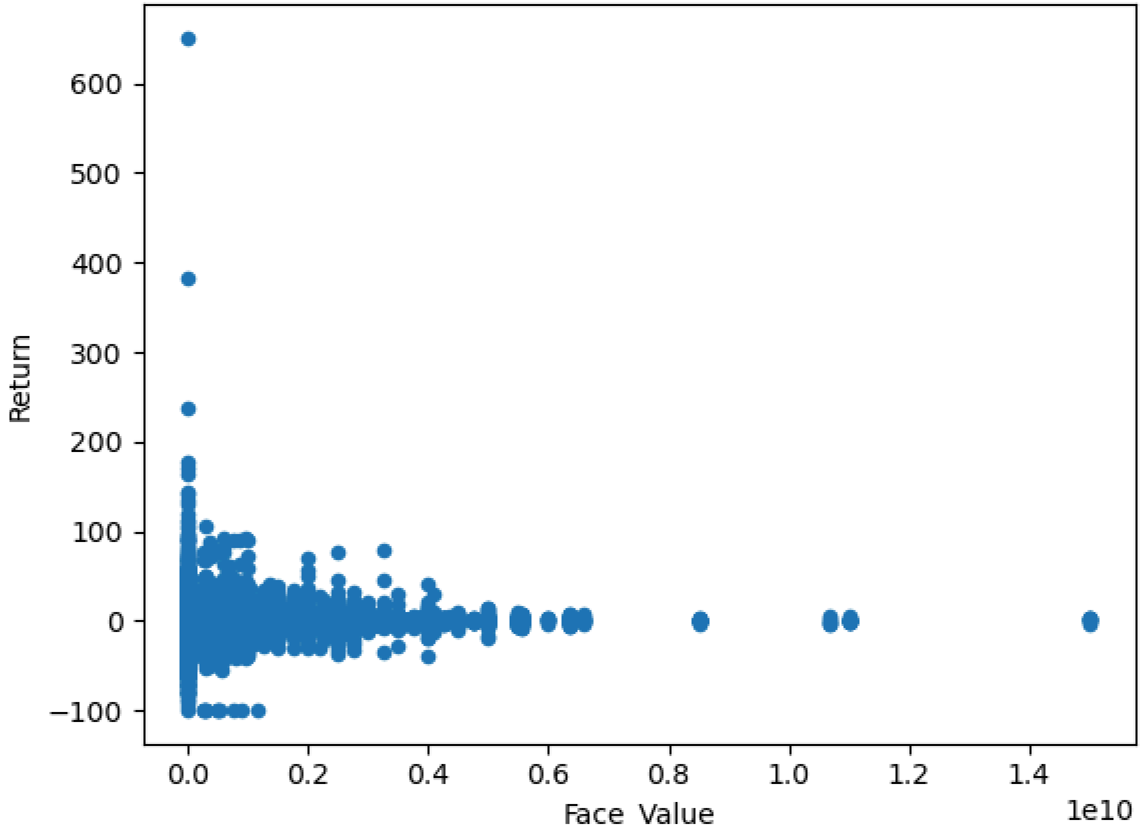
In [ ]:
```
features.replace(-99.990000, np.nan)[['Face_Value', 'Return']].corr()
```

Out[ ]:

	Face_Value	Return
Face_Value	1.000000	0.008955
Return	0.008955	1.000000

In [ ]:
```
features.replace(-99.990000, np.nan).dropna().plot(kind='scatter', x='Face_V
```

Out[ ]:
```
<Axes: xlabel='Face_Value', ylabel='Return'>
```



In [ ]:
```
features['Duration'] = data['DUR']
```

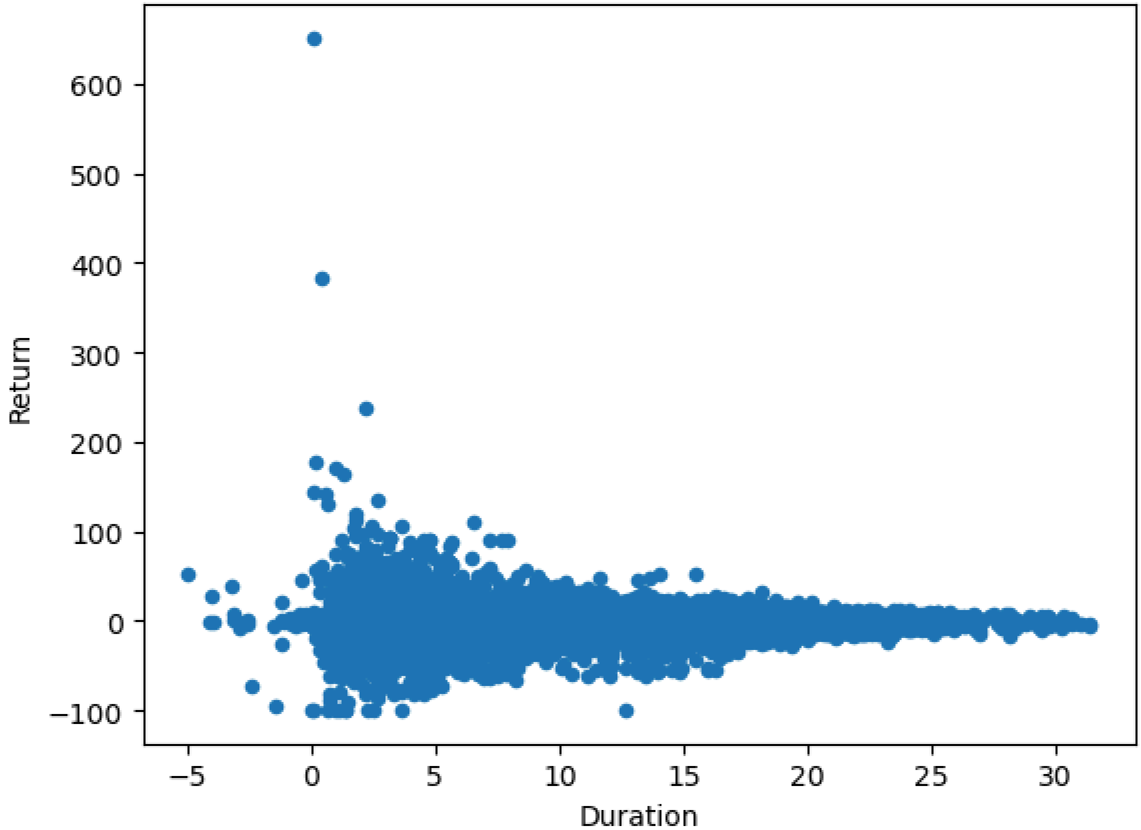
In [ ]:
```
features.replace(-99.990000, np.nan)[['Duration', 'Return']].corr()
```

Out[ ]:

	Duration	Return
Duration	1.00000	0.00503
Return	0.00503	1.00000

In [ ]:
```
features.replace(-99.990000, np.nan).dropna().plot(kind='scatter', x='Durati
```

Out[ ]:
```
<Axes: xlabel='Duration', ylabel='Return'>
```



In [ ]:
```
rating_key = { 'AAA': 1, 'AA1': 2, 'AA2': 3, 'AA3': 4, 'A1': 5, 'A2': 6,
              'A3': 7,'BBB1': 8, 'BBB2': 9,'BBB3': 10, 'BB1': 11,
              'BB2': 12, 'BB3': 13,'B1': 14, 'B2': 15, 'B3': 16,
              'CCC1': 17, 'CCC2': 18, 'CCC3': 19, 'CC': 20, 'C': 21, 'D': 22

features['Rating'] = data['Rating'].map(rating_key)
```

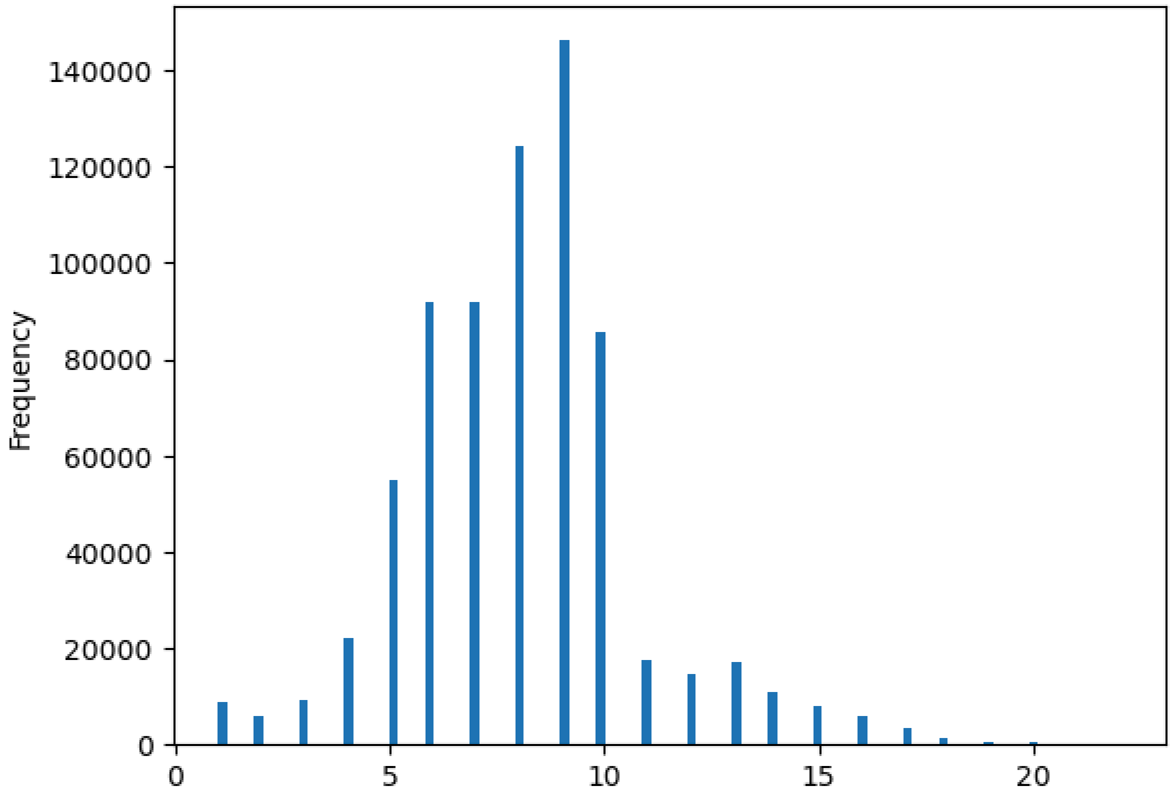
In [ ]:
```
features[['Rating']].describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
Rating	720222.0	8.101027	2.691346	1.0	6.0	8.0	9.0	22.0

In [ ]:
```
features['Rating'].plot(kind = 'hist', bins =100)
```

Out[ ]: <Axes: ylabel='Frequency'>



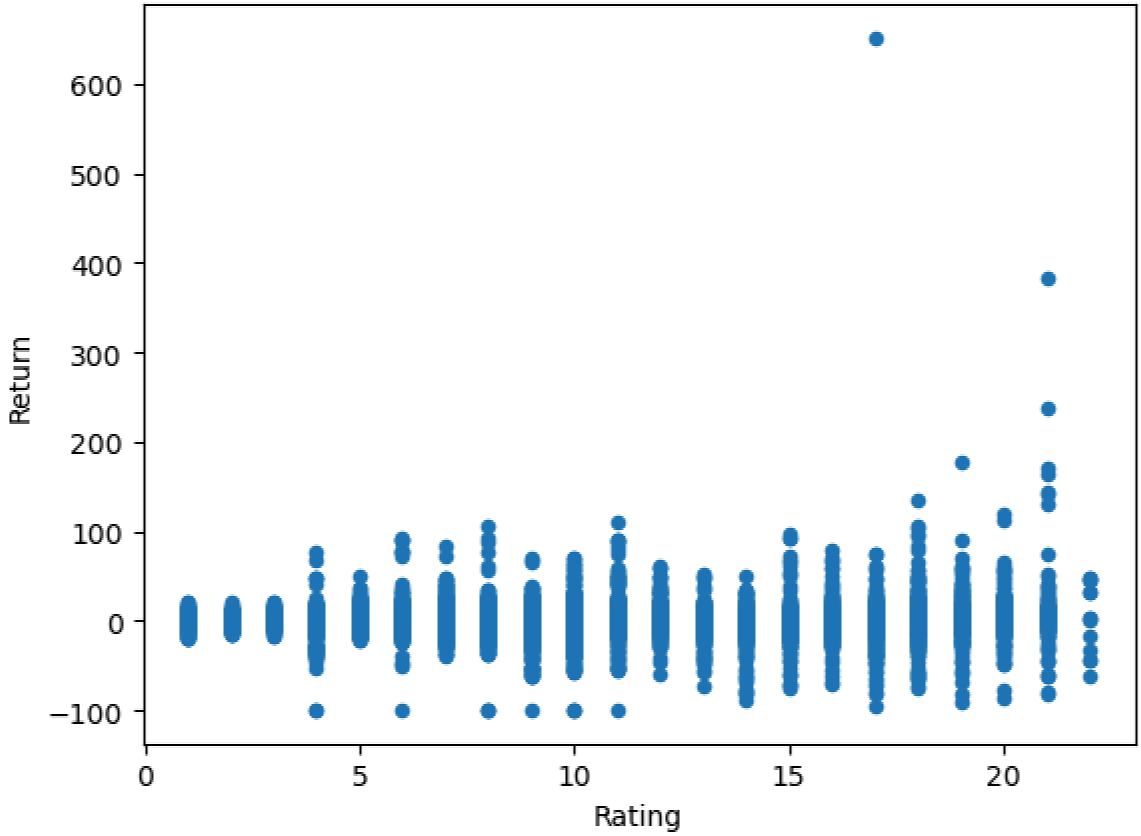
In [ ]:
```
features.replace(-99.990000, np.nan)[['Rating', 'Return']].corr()
```

Out[ ]:

	Rating	Return
Rating	1.00000	0.02099
Return	0.02099	1.00000

In [ ]:
```
features.replace(-99.990000, np.nan).dropna().plot(kind='scatter', x='Rating
```

Out[ ]:
```
<Axes: xlabel='Rating', ylabel='Return'>
```



In [ ]:
```
features['Spread'] = data['oas_BOM']
```

In [ ]:
```
features[['Spread']].describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%
Spread	720222.0	163.228061	217.536294	-1000.0	80.0	125.2854	189.777375

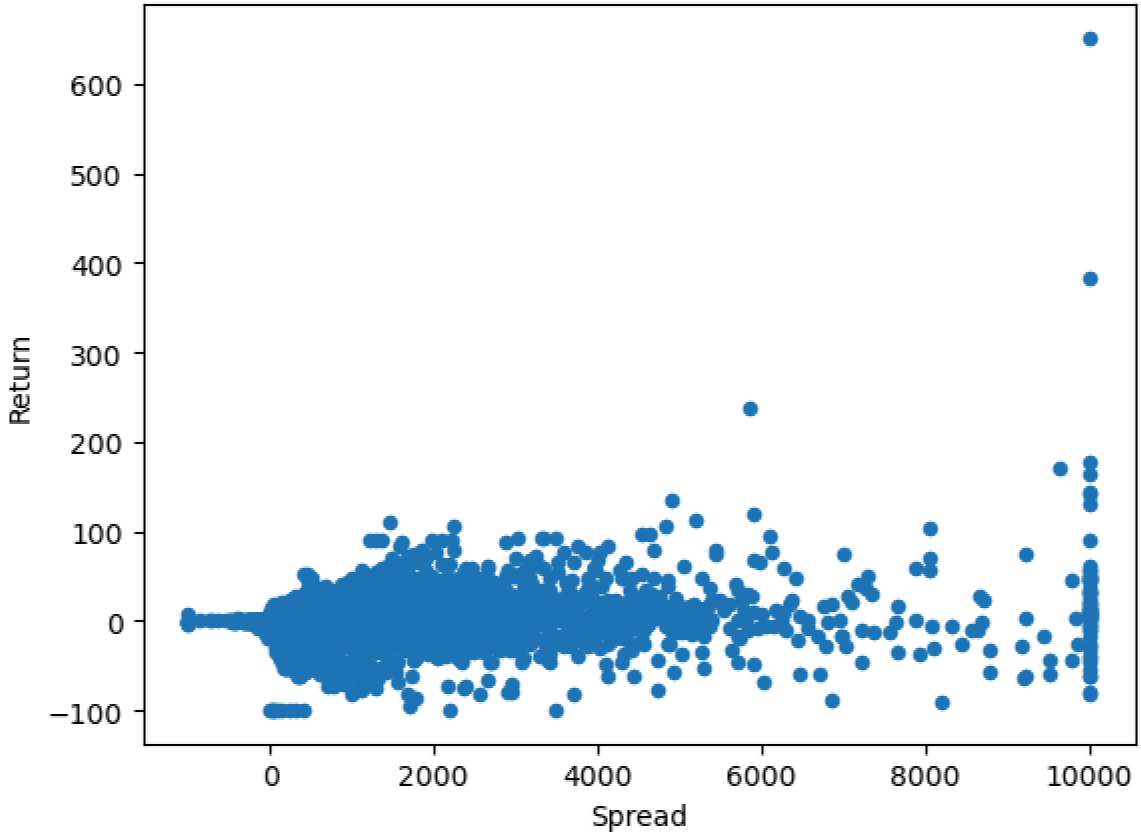
In [ ]:
```
features.replace(-99.990000, np.nan)[['Spread', 'Return']].corr()
```

Out[ ]:

	Spread	Return
Spread	1.00000	0.17072
Return	0.17072	1.00000

In [ ]:
```
features.replace(-99.990000, np.nan).dropna().plot(kind='scatter', x='Spreac
```

Out[ ]:
```
<Axes: xlabel='Spread', ylabel='Return'>
```



In [ ]:
```
features['Distance_to_Default'] = data['PD_DRISK']
```

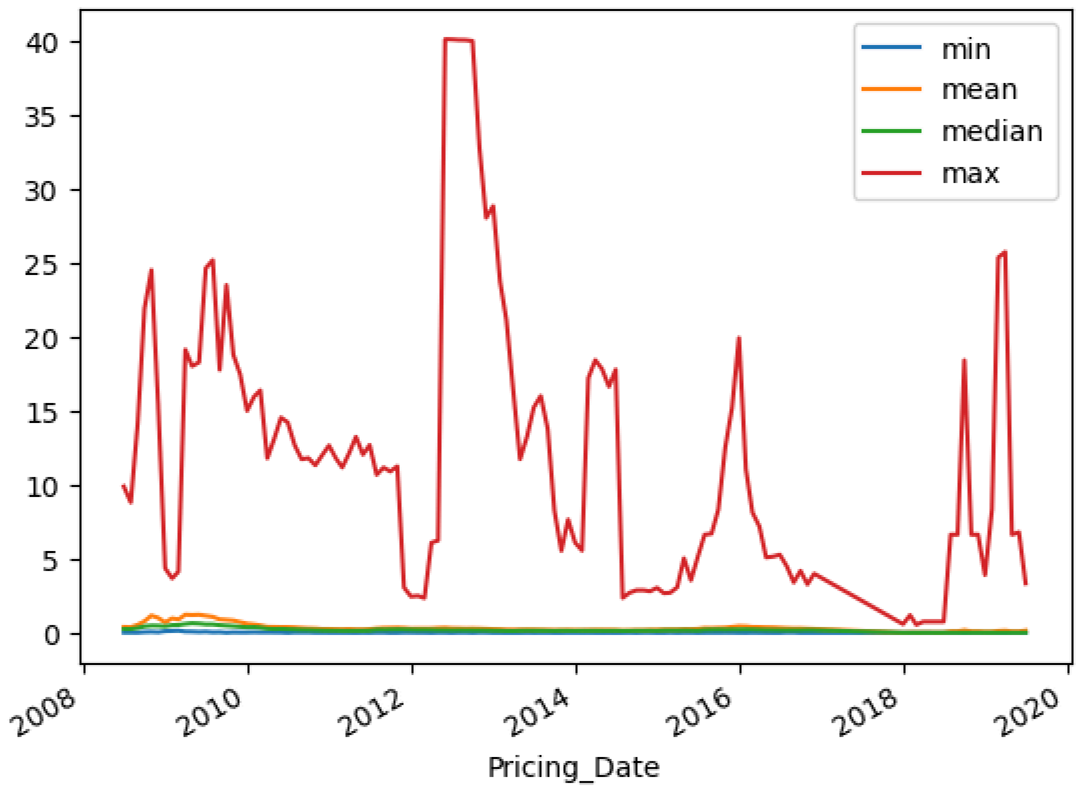
In [ ]:
```
features.loc[features['Distance_to_Default']>-99, ['Distance_to_Default']].c
```

Out[ ]:

	count	mean	std	min	25%	50%
Distance_to_Default	337776.0	0.275754	0.748569	3.000000e-09	0.068483	0.1455

In [ ]:
```
features.loc[features['Distance_to_Default']>-99, :].groupby('Pricing_Date')
```

Out[ ]: <Axes: xlabel='Pricing_Date'>



In [ ]:
```
features.loc[features['Distance_to_Default']>-99, :].replace(-99.990000, np.
```

Out[ ]:

	Distance_to_Default	Return
Distance_to_Default	1.00000	0.10878
Return	0.10878	1.00000

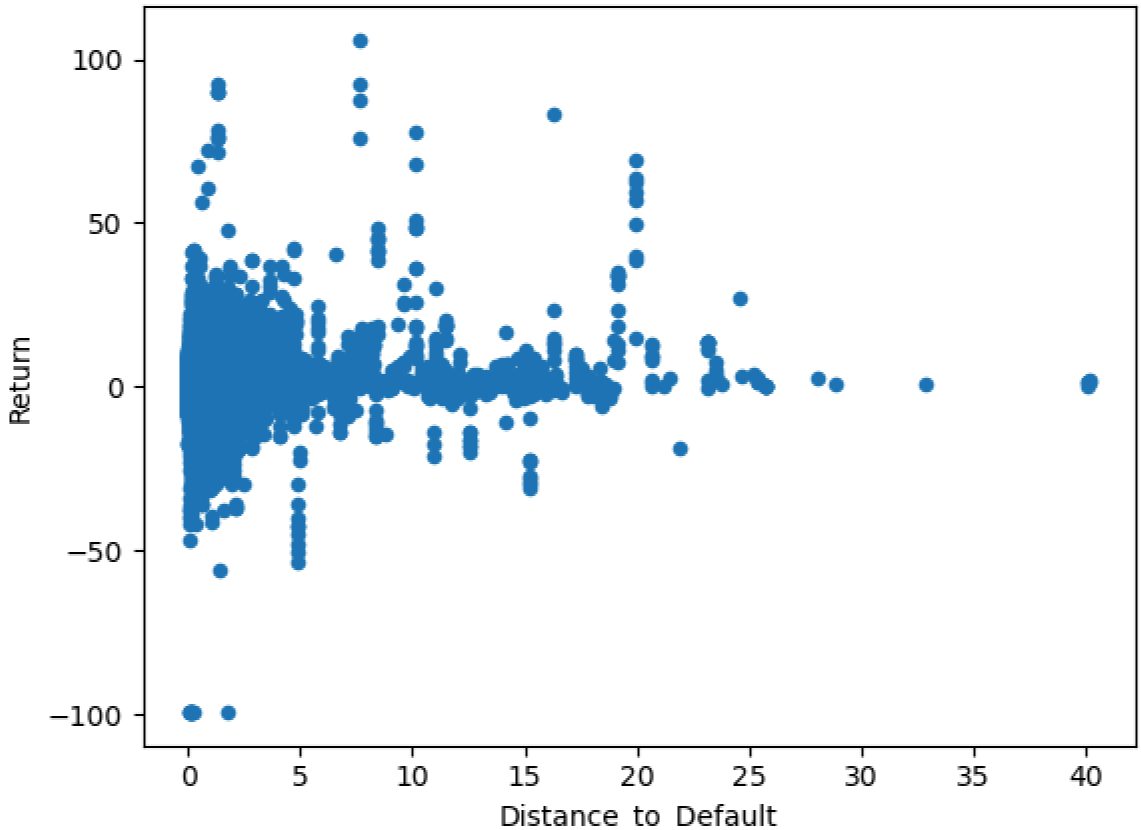
In [ ]:
```
features.replace(-99.990000, np.nan)[['Distance_to_Default', 'Return']].corr
```

Out[ ]:

	Distance_to_Default	Return
Distance_to_Default	1.00000	0.01402
Return	0.01402	1.00000

In [ ]:
```
features.loc[features['Distance_to_Default']>-99, :].replace(-99.990000, np.
```

Out[ ]:
```
<Axes: xlabel='Distance_to_Default', ylabel='Return'>
```



In [ ]:
```
features['Spread_to_D2D'] = data['oas_BOM'] / data['PD_DRISK']
```

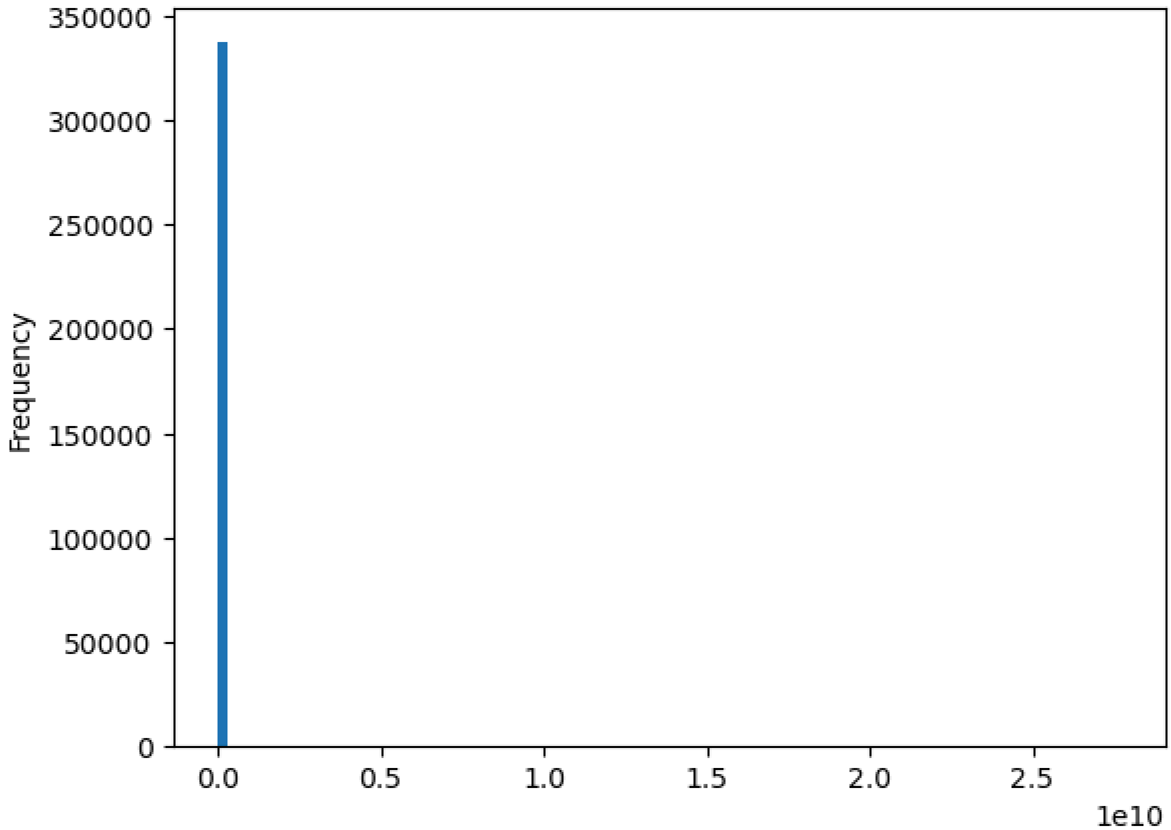
In [ ]:
```
features[['Spread_to_D2D']].describe().T
```

Out[ ]:

	count	mean	std	min	2
Spread_to_D2D	338308.0	3.264080e+06	1.085960e+08	-2.928870e+06	456.373(

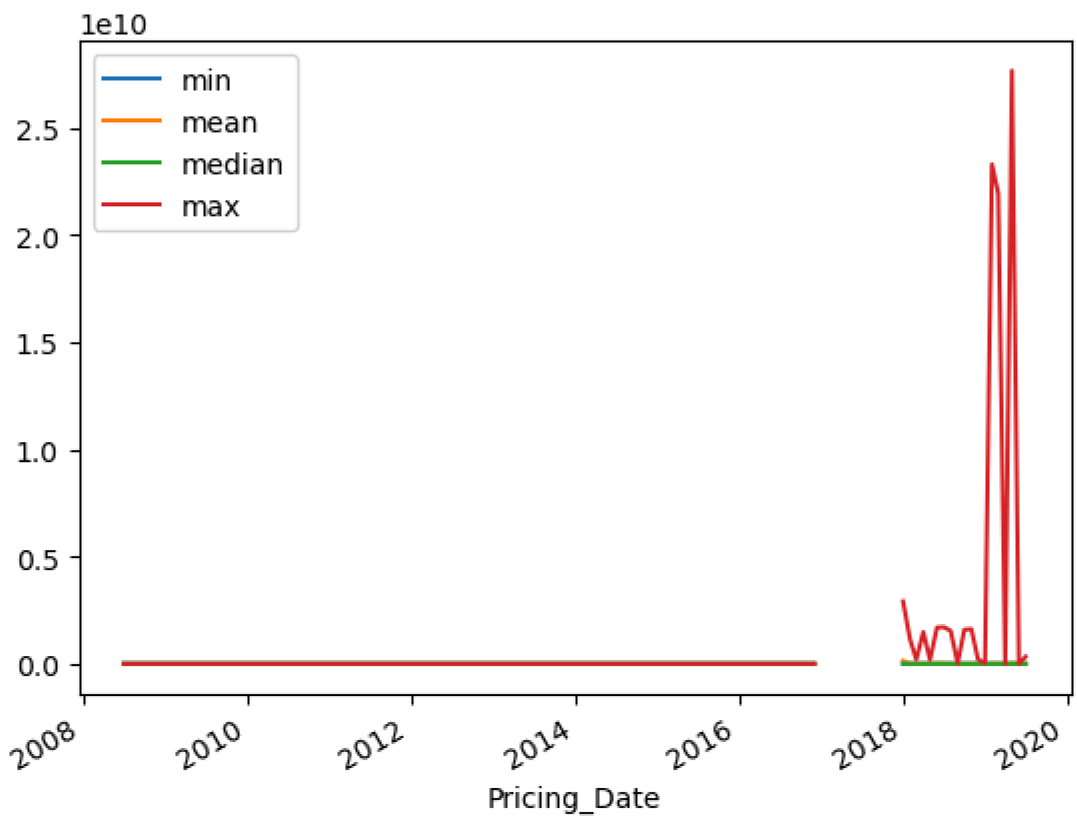
In [ ]:
```
features['Spread_to_D2D'].plot(kind='hist', bins = 100)
```

Out[ ]: <Axes: ylabel='Frequency'>



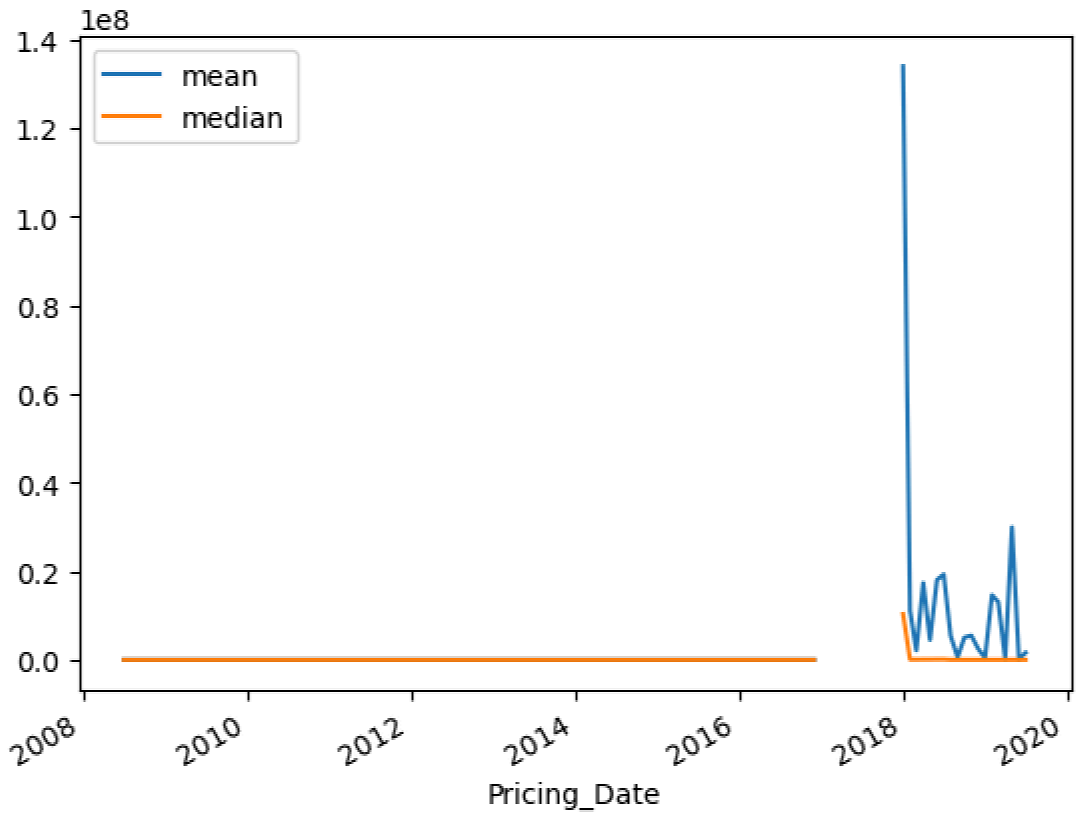
In [ ]:
```
features.groupby('Pricing_Date')['Spread_to_D2D'].agg(['min', 'mean', 'media
```

Out[ ]:
```
<Axes: xlabel='Pricing_Date'>
```



In [ ]:
```
features.groupby('Pricing_Date')['Spread_to_D2D'].agg(['mean', 'median']).pl
```

Out[ ]: <Axes: xlabel='Pricing_Date'>



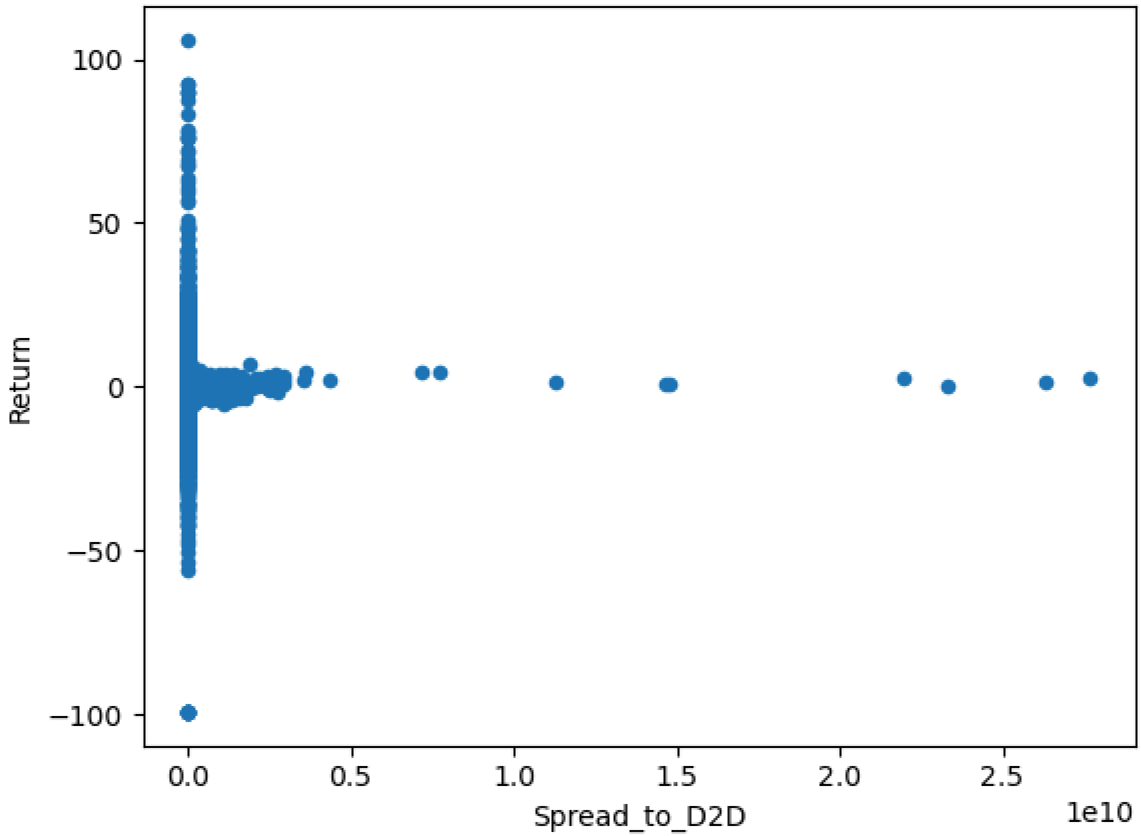
In [ ]:
```
features.replace(-99.990000, np.nan)[['Spread_to_D2D', 'Return']].corr()
```

Out[ ]:

	Spread_to_D2D	Return
Spread_to_D2D	1.000000	-0.006199
Return	-0.006199	1.000000

In [ ]:
```
features.replace(-99.990000, np.nan).dropna().plot(kind='scatter', x='Spreac
```

Out[ ]: <Axes: xlabel='Spread_to_D2D', ylabel='Return'>



In [ ]:
```
# Equity Vars
(data['pricetobook']==0).sum()
```

Out[ ]: 34532

In [ ]:
```
data.loc[data['pricetobook']==0, 'pricetobook'] = np.nan
```

In [ ]:
```
features['Book_to_Price'] = 1/ data['pricetobook']
```

In [ ]:
```
features[['Book_to_Price']].describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	
Book_to_Price	654415.0	0.681529	8.167072	-0.110675	0.233503	0.450106	0.7

In [ ]:
```
# debt to ebitda

features['Debt_to_EBITDA'] = data['debt']/ data['ebitda']
```

In [ ]:
```
(data['ebitda']==0).sum()
```

Out[ ]: 2180

In [ ]:
```
data.loc[data['ebitda']==0, 'ebitda'] = np.nan
```

In [ ]:
```
features['Debt_to_EBITDA'] = data['debt']/ data['ebitda']
```

In [ ]:

```
features[['Debt_to_EBITDA']].describe().T
```

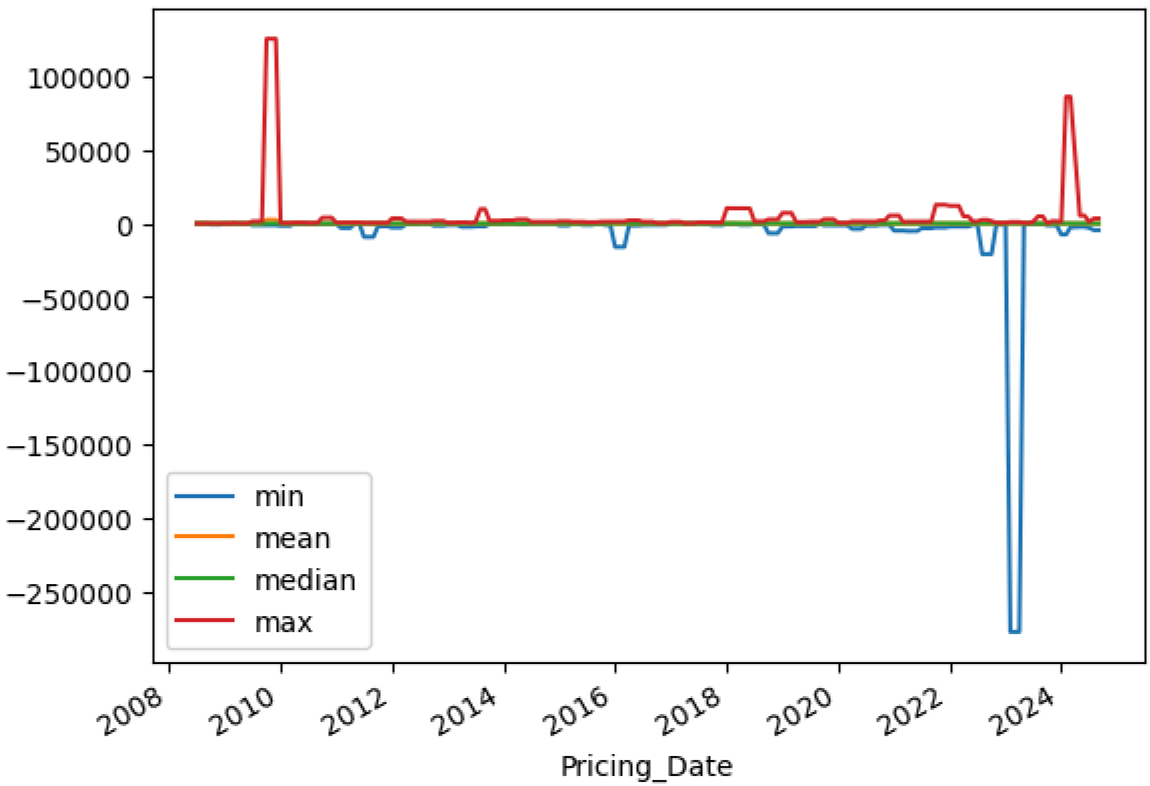
Out[ ]:

	count	mean	std	min	25%	
Debt_to_EBITDA	705523.0	34.248926	1640.895226	-277369.666667	6.674487	1

In [ ]:

```
features.groupby('Pricing_Date')['Debt_to_EBITDA'].agg(['min', 'mean', 'medi
```

Out[ ]: <Axes: xlabel='Pricing_Date'>



In [ ]:

```
data.loc[data['pricetoearnings']==0, 'pricetoearnings'] = np.nan
```

In [ ]:

```
features['Earnings_to_Price'] = 1/ data['pricetoearnings']
```

In [ ]:

```
features[['Earnings_to_Price']].describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%
Earnings_to_Price	613369.0	1.375541	112.89555	-0.734268	0.038073	0.056112

In [ ]:

```
features['Marketcap'] = data['marketcap']
```

In [ ]:

```
features['Debt'] = data['debt']
```

In [ ]:

```
# Profitability
# asset = debt + marketcap/pricetobook
# gross profit = ebit / ebitmargin * grossmargin
```

```
revenue = data['ebit'] / data['ebitmargin']
gross_profit = revenue * data['grossmargin']
asset = data['debt'] + data['marketcap'] / data['pricetobook']

features['Profitability'] = gross_profit / asset
```

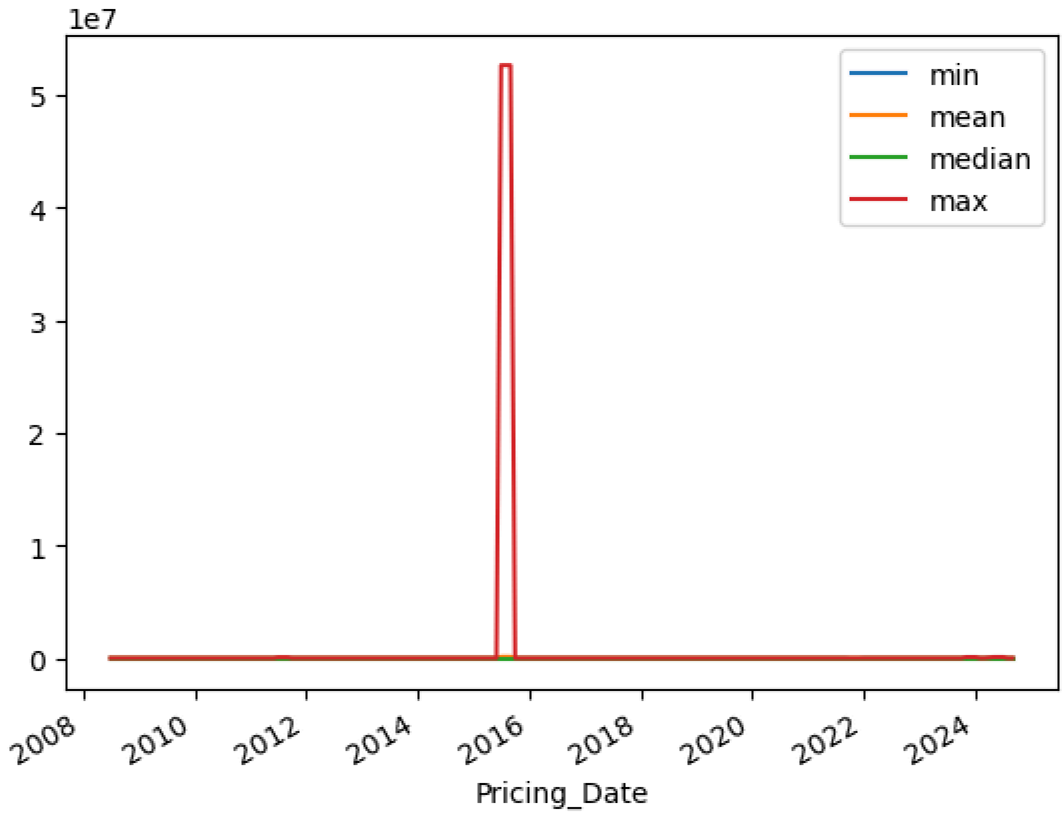
In [ ]:
```
features[['Profitability']].describe().T
```

Out[ ]:

	count	mean	std	min	25%	
Profitability	505125.0	1881.864561	314169.420943	-50169.975762	0.046991	0.

In [ ]:
```
features.groupby('Pricing_Date')['Profitability'].agg(['min', 'mean', 'media

# i dont think this looks right
```

Out[ ]: <Axes: xlabel='Pricing_Date'>



In [ ]:
```
# profiutability change

features.sort_values(by = ['Pricing_Date', 'ISIN'], inplace=True)

features['Profitability_Change'] = 100 * features.groupby('ISIN')['Profitabi
                                      .diff(periods = 60)
```

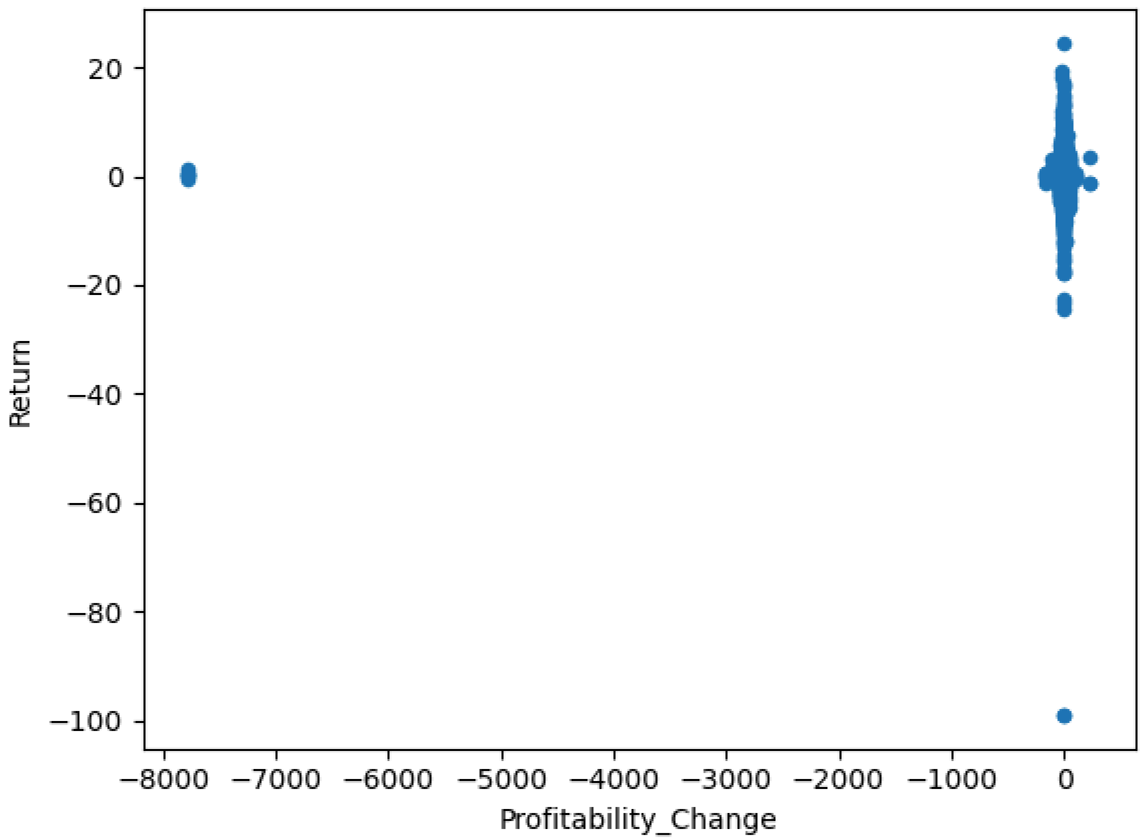
In [ ]:
```
features[['Profitability_Change']].describe().T
```

Out[ ]:

	count	mean	std	min	25%
Profitability_Change	99581.0	-0.735317	60.910189	-7782.223417	-2.619893	-0.

In [ ]:
```
features.replace(-99.990000, np.nan).dropna().plot(kind = 'scatter', x = 'Pr
```

Out[ ]: <Axes: xlabel='Profitability_Change', ylabel='Return'>



In [ ]:
```
# skip industry dummies

# the data frame might be too large
```

In [ ]:
```
# leveraged-based vars
# this is degree of operating leversage?
features['Operating_Leverage'] = data['ebitgrowth'] / data['revenuegrowth']
```

In [ ]:
```
(data['revenuegrowth']==0).sum()
```

Out[ ]: 11475

In [ ]:
```
data.loc[data['revenuegrowth']==0, 'revenuegrowth'] = np.nan

features['Operating_Leverage'] = data['ebitgrowth'] / data['revenuegrowth']
```

In [ ]:
```
features[['Operating_Leverage']].describe().T
```

Out[ ]:

	count	mean	std	min	25%
Operating_Leverage	685921.0	0.829706	3018.526682	-997492.483166	-0.69022

In [ ]:
```
features['Book_Leverage'] = data['leverageratio']
```

In [ ]:
```
features[['Book_Leverage']].describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	
Book_Leverage	707991.0	10.557819	899.272731	-3213.5313	2.2445	3.1084	5.

In [ ]:
```
# market leverage

features['Market_Leverage'] = data['debt'] / data['marketcap']
```

In [ ]:
```
(data['marketcap'] == 0).sum()
```

Out[ ]: 4183

In [ ]:
```
data.loc[data['marketcap'] == 0, 'marketcap'] = np.nan
```

In [ ]:
```
features['Market_Leverage'] = data['debt'] / data['marketcap']
```

In [ ]:
```
features[['Market_Leverage']].describe().T
```

Out[ ]:

	count	mean	std	min	25%	
Market_Leverage	698462.0	911.72098	381175.105625	-26.956919	0.202685	0.4

In [ ]:
```
# moments-based measures

# turnover volatility

data['Asset_Turnover'] = revenue/asset
```

In [ ]:
```
features['Turnover_Volatility'] = data.groupby('ISIN')['Asset_Turnover']\
                                      .rolling(window = 12, min_periods = 6)\
                                      .std().reset_index(level = 0, drop = True)
```

In [ ]:
```
features[['Turnover_Volatility']].describe().T
```

Out[ ]:

	count	mean	std	min	25%	
Turnover_Volatility	580646.0	10978.132305	1.152556e+06	0.0	0.005241	0.012

In [ ]:
```
# equity volatility

data['Stock_Return'] = data.groupby('ISIN')['Stock_Price'].pct_change()
```

```python
features['Stock_Volatility'] = data.groupby('ISIN')['Stock_Return'].rolling(
                               .std().reset_index(level =0, drop =True)
```

In [ ]:
```python
features[['Stock_Volatility']].describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%
Stock_Volatility	624032.0	1.043157	86.489802	0.000965	0.048274	0.067323	0

In [ ]:
```python
# bond volatility

features['Bond_Volatility'] = data.groupby('ISIN')['Return'].rolling(window=
                              .std().reset_index(level=0, drop =True)
```

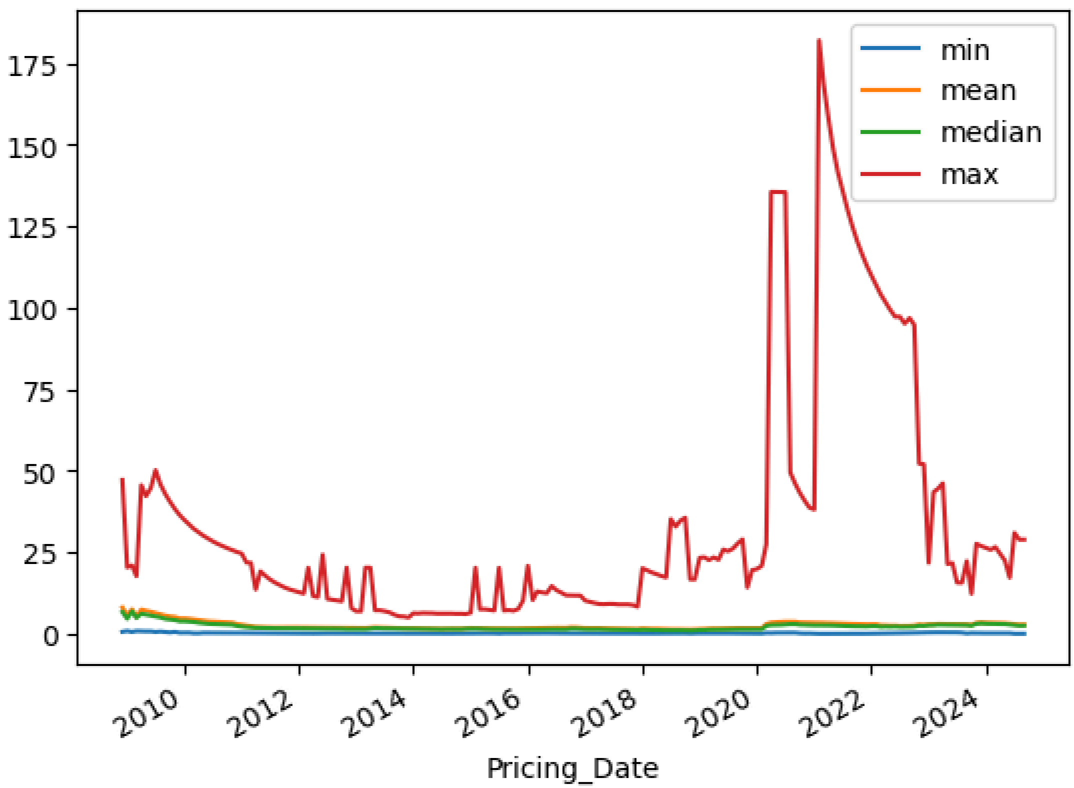
In [ ]:
```python
features[['Bond_Volatility']].describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%
Bond_Volatility	638264.0	2.395712	2.31653	0.012596	1.079623	1.864304	3.07

In [ ]:
```python
features.groupby('Pricing_Date')['Bond_Volatility'].agg(['min', 'mean', 'mec
```

Out[ ]: <Axes: xlabel='Pricing_Date'>



In [ ]:
```python
features['Bond_Skew'] = data.groupby('ISIN')['Return'].rolling(window = 60,
                        .skew().reset_index(level = 0, drop =True)
```

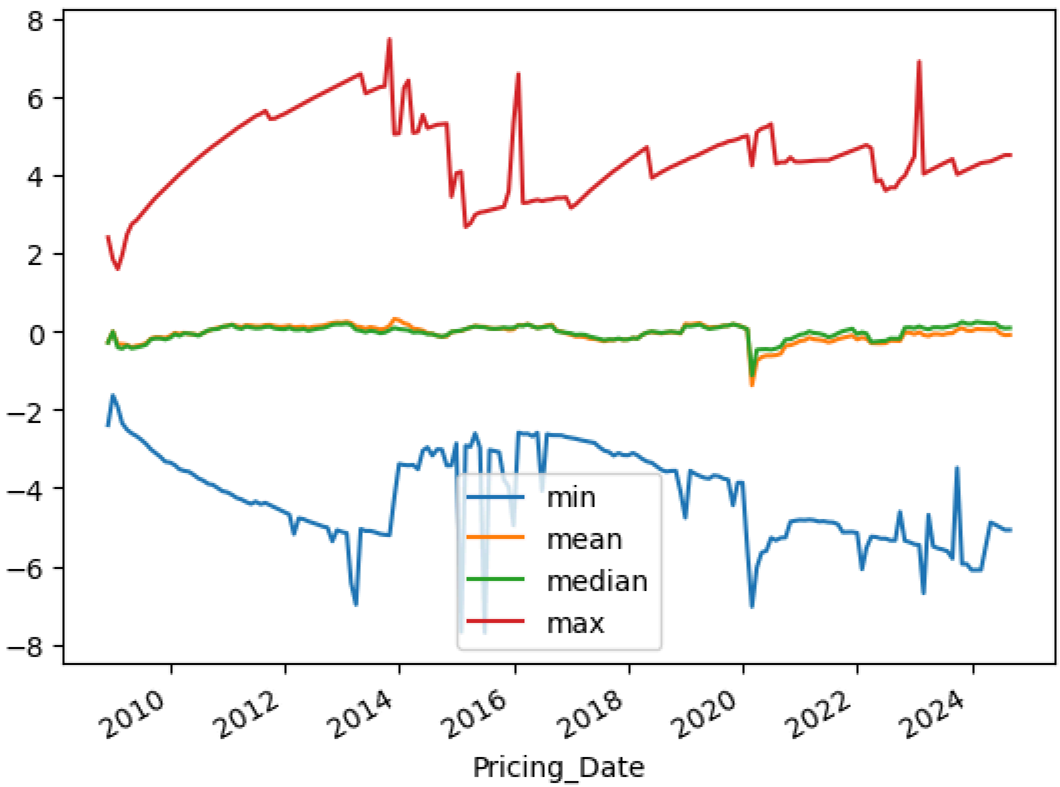
In [ ]:
```python
features[['Bond_Skew']].describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	7
Bond_Skew	638264.0	-0.058572	0.908185	-7.69809	-0.469035	0.003808	0.4057

In [ ]:
```
features.groupby('Pricing_Date')['Bond_Skew'].agg(['min', 'mean', 'median',
```

Out[ ]: <Axes: xlabel='Pricing_Date'>



In [ ]:
```
# VaR
# 5% VaR of each bond over the past 36 months
features['VaR'] = data.groupby('ISIN')['Return'].rolling(window = 36, min_pe
                     .apply(lambda x: np.percentile(x,5), raw=True).reset_index
```

In [ ]:
```
features[['VaR']].describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%
VaR	539383.0	-2.542746	2.503426	-63.564257	-3.437242	-1.88835	-1.018107

In [ ]:
```
shifted_2 = data.groupby('ISIN')['Stock_Price'].shift(2)
shifted_6 = data.groupby('ISIN')['Stock_Price'].shift(6)

features['Mom_6m_Equity'] = 100* (shifted_2 - shifted_6) / shifted_6
```

In [ ]:
```
features[['Mom_6m_Equity']].describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%
Mom_6m_Equity	624032.0	48.876028	8405.660379	-99.9299	-5.989897	3.08003

In [ ]:
```
shifted_2 = data.groupby('ISIN')['Price_BOM'].shift(2)
shifted_6 = data.groupby('ISIN')['Price_BOM'].shift(6)

features['Mom_6m_Bond'] = 100* (shifted_2 - shifted_6) / shifted_6
```

In [ ]:
```
features[['Mom_6m_Bond']].describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%
Mom_6m_Bond	624032.0	-0.09418	6.490818	-97.410811	-2.174783	-0.117957

In [ ]:
```
features['Mom_6m_BondxRating'] = features['Mom_6m_Bond'] * features['Rating'
```

In [ ]:
```
features[['Mom_6m_BondxRating']].describe().T
```

Out[ ]:

	count	mean	std	min	25%	
Mom_6m_BondxRating	624032.0	-0.680163	76.495838	-1974.0	-16.628765	-0.7

In [ ]:
```
shifted_2 = features.groupby('ISIN')['Spread'].shift(2)
shifted_6 = features.groupby('ISIN')['Spread'].shift(6)

features['Mom_6m_Log_Spread'] = 100 * (np.log(shifted_2) - np.log(shifted_6)
```

```
/opt/miniconda3/envs/py3k/lib/python3.12/site-packages/pandas/core/arraylik
e.py:399: RuntimeWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/miniconda3/envs/py3k/lib/python3.12/site-packages/pandas/core/arraylik
e.py:399: RuntimeWarning: invalid value encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

In [ ]:
```
features.loc[features['Mom_6m_Log_Spread']==-np.inf, 'Mom_6m_Log_Spread'] =
features.loc[features['Mom_6m_Log_Spread']==np.inf, 'Mom_6m_Log_Spread'] = f
```

In [ ]:
```
features[['Mom_6m_Log_Spread']].describe().T
```

Out[ ]:

	count	mean	std	min	25%
Mom_6m_Log_Spread	622959.0	-4.848219	37.874423	-894.018614	-21.455981

In [ ]:
```
curve_UST_df = pd.read_csv(os.path.join(data_path, 'UST Curve Data.csv'), pa

curve_UST_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 215 entries, 0 to 214
Data columns (total 26 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Price       215 non-null    object
 1   1 Mo        215 non-null    float64
 2   2 Mo        215 non-null    float64
 3   3 Mo        215 non-null    float64
 4   6 Mo        214 non-null    float64
 5   1 Yr        215 non-null    float64
 6   2 Yr        215 non-null    float64
 7   3 Yr        215 non-null    float64
 8   5 Yr        215 non-null    float64
 9   7 Yr        215 non-null    float64
 10  10 Yr       215 non-null    float64
 11  20 Yr       215 non-null    float64
 12  30 Yr       215 non-null    float64
 13  Price Date  215 non-null    datetime64[ns]
 14  1 M         215 non-null    float64
 15  2 M         215 non-null    float64
 16  3 M         215 non-null    float64
 17  6 M         215 non-null    float64
 18  1 Y         215 non-null    float64
 19  2 Y         215 non-null    float64
 20  3 Y         215 non-null    float64
 21  5 Y         215 non-null    float64
 22  7 Y         215 non-null    float64
 23  10 Y        215 non-null    float64
 24  20 Y        215 non-null    float64
 25  30 Y        215 non-null    float64
dtypes: datetime64[ns](1), float64(24), object(1)
memory usage: 43.8+ KB
```

In [ ]:

```
curve_UST_df['Price Date'] = curve_UST_df['Price Date'] + pd.DateOffset(mont
```

In [ ]:

```
curve_UST = curve_UST_df.iloc[:, 1:14].copy()
curve_UST.set_index('Price Date', inplace=True)
curve_UST.sort_index(inplace=True)
```

In [ ]:

```
(curve_UST==0).sum()
```

Out[ ]:

```
1 Mo     3
2 Mo     1
3 Mo     1
6 Mo     0
1 Yr     0
2 Yr     0
3 Yr     0
5 Yr     0
7 Yr     0
10 Yr    0
20 Yr    0
30 Yr    0
dtype: int64
```

In [ ]:
```python
curve_UST_returns = curve_UST_df.iloc[:, 13:]

curve_UST_returns.set_index('Price Date', inplace =True)
curve_UST_returns.sort_index(inplace = True)
```

In [ ]:
```python
curve_UST_returns
```

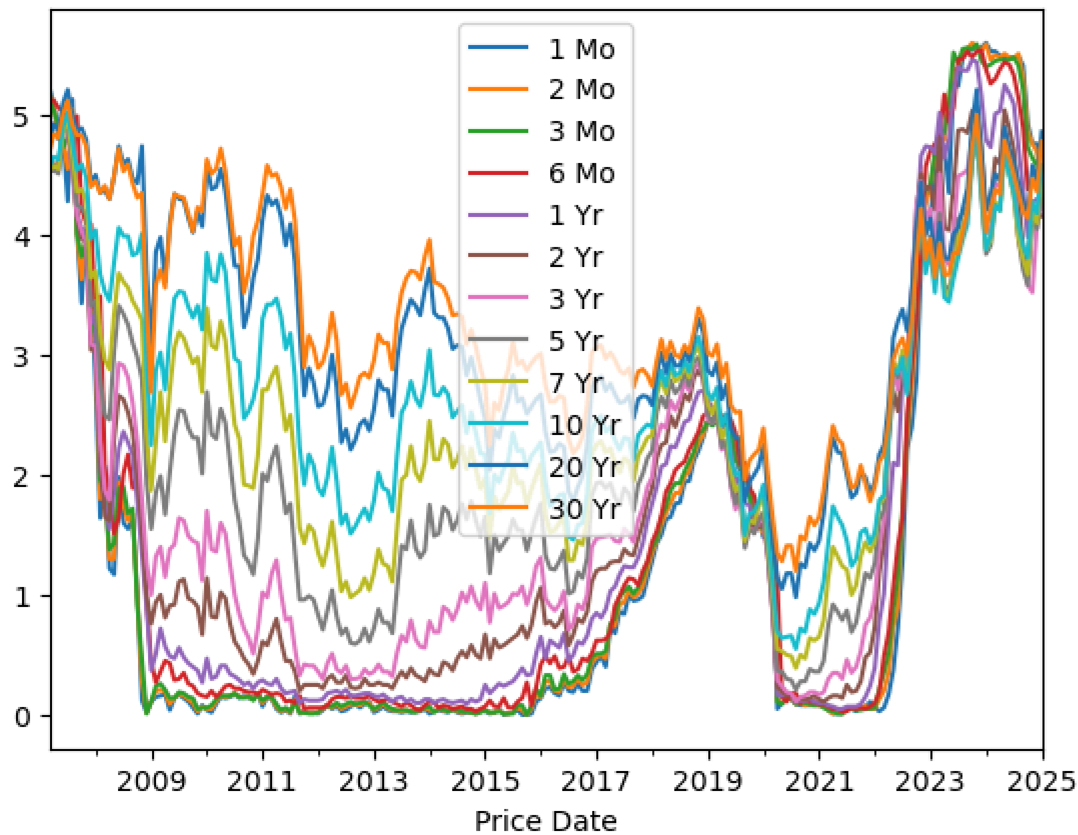
Out[ ]:

	1 M	2 M	3 M	6 M	1 Y	2 Y	3 Y	5 Y	7 Y	10 Y	20 Y	30
Price Date												
2007-03-01	0.24	0.140	0.04	-0.04	-0.13	-0.29	-0.30	-0.30	-0.29	-0.27	-0.24	-0.
2007-04-01	-0.17	-0.145	-0.12	-0.06	-0.06	-0.07	-0.01	0.02	0.05	0.09	0.14	0.
2007-05-01	-0.27	-0.200	-0.13	-0.03	-0.01	0.02	0.00	-0.03	-0.03	-0.02	-0.04	-0.
2007-06-01	-0.02	-0.100	-0.18	-0.07	0.06	0.32	0.34	0.35	0.32	0.27	0.22	0.
2007-07-01	-0.50	-0.205	0.09	-0.03	-0.04	-0.05	0.01	0.06	0.09	0.13	0.11	0.
...	...	...	...	...	...	...	...	...	...	...	...	
2024-09-01	-0.08	-0.190	-0.20	-0.25	-0.35	-0.38	-0.17	-0.26	-0.17	-0.18	-0.16	-0.
2024-10-01	-0.48	-0.450	-0.48	-0.51	-0.40	-0.25	-0.21	-0.13	-0.13	-0.10	-0.09	-0.
2024-11-01	-0.17	-0.110	-0.09	0.05	0.29	0.50	-0.06	0.57	0.54	0.47	0.39	0.
2024-12-01	0.00	-0.070	-0.06	-0.01	0.03	-0.03	0.58	-0.10	-0.11	-0.10	-0.13	-0.
2025-01-01	-0.36	-0.300	-0.21	-0.10	-0.06	0.03	0.15	0.33	0.38	0.40	0.41	0.

215 rows × 12 columns

In [ ]:
```python
curve_UST.plot()
```

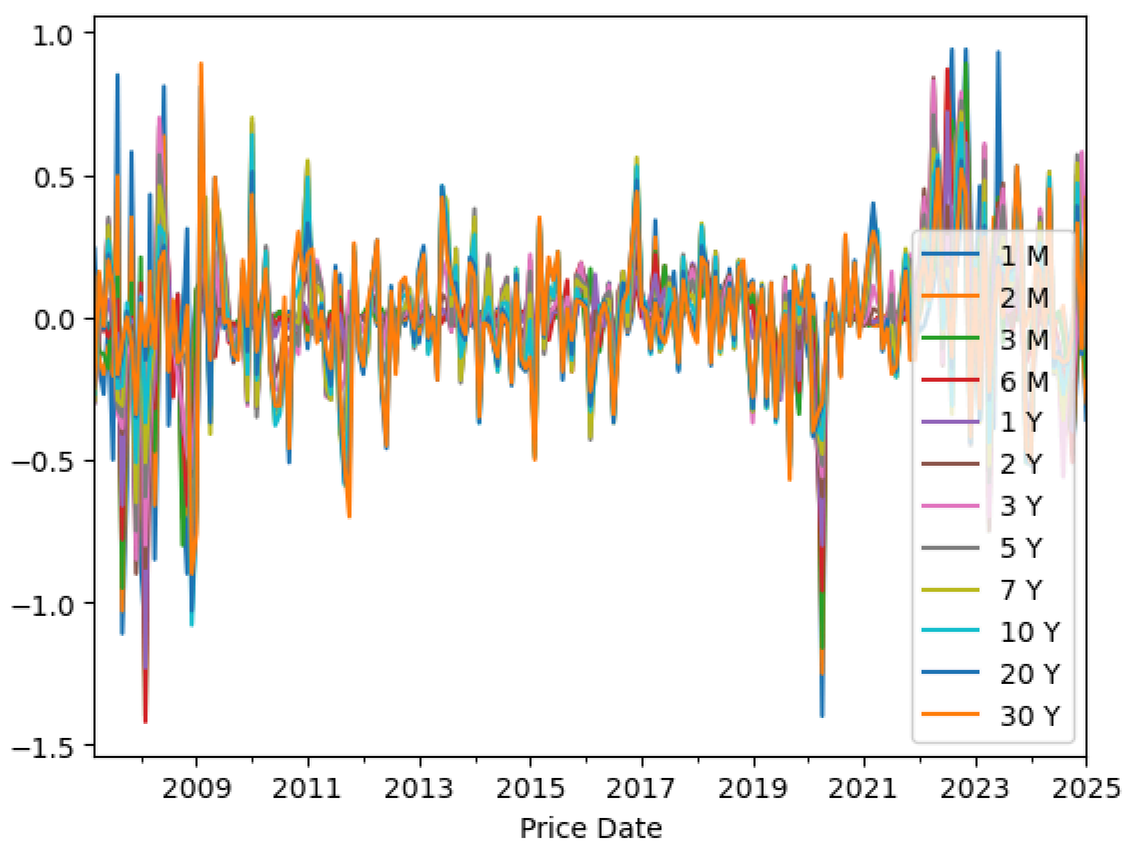
Out[ ]: <Axes: xlabel='Price Date'>



In [ ]:

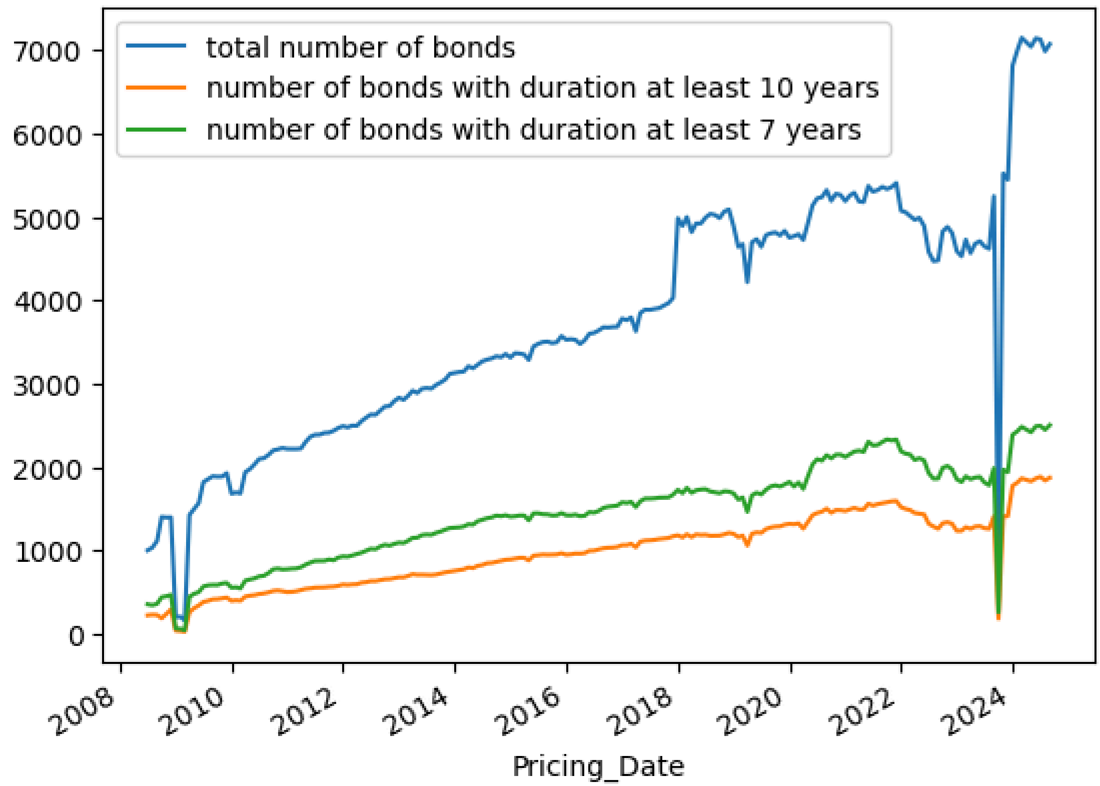
```
curve_UST_returns.plot()
```

Out[ ]: <Axes: xlabel='Price Date'>



In [ ]:
```
features.groupby('Pricing_Date')['Duration'].count().plot(label='total numbe
features.groupby('Pricing_Date')['Duration'].apply(lambda x: (x>10).sum()).p
features.groupby('Pricing_Date')['Duration'].apply(lambda x: (x>7).sum()).pl

plt.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x30f8c3470>

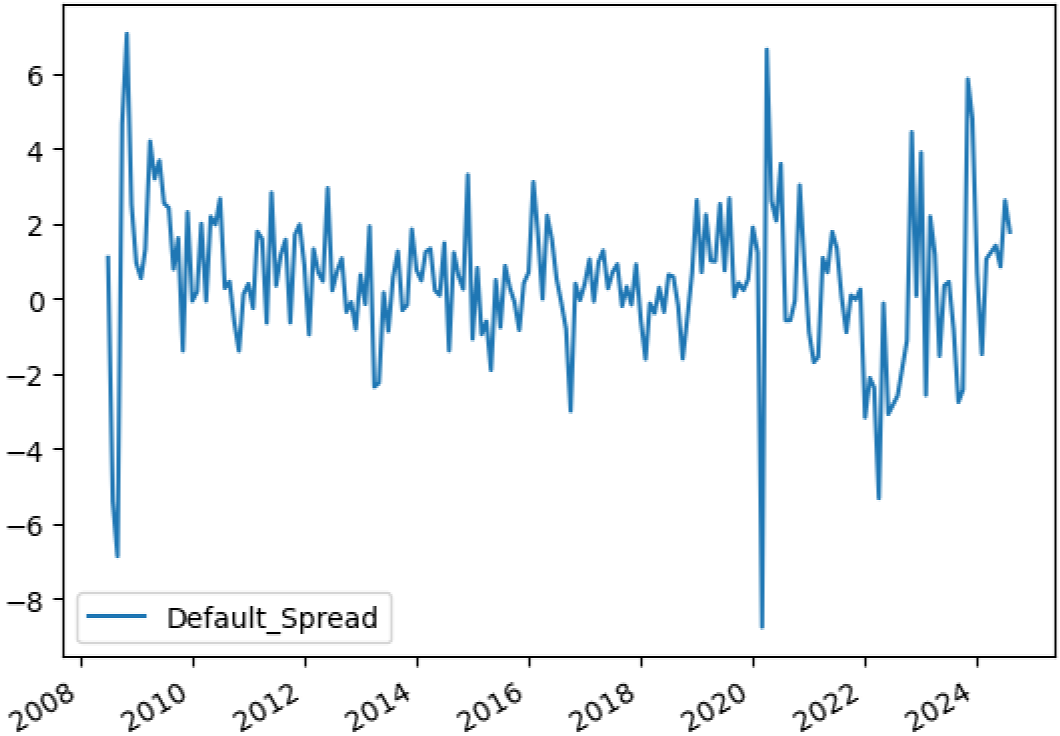


In [ ]:
```
curve_UST_returns.loc[:, '10 Y':].mean(axis = 1)
```

Out[ ]:
```
Price Date
2007-03-01   -0.253333
2007-04-01    0.130000
2007-05-01   -0.030000
2007-06-01    0.230000
2007-07-01    0.116667
                ...
2024-09-01   -0.163333
2024-10-01   -0.083333
2024-11-01    0.396667
2024-12-01   -0.113333
2025-01-01    0.410000
Length: 215, dtype: float64
```

In [ ]:
```
default_spread = (data.groupby('Pricing_Date')['Return'].apply('mean') - cur

default_spread.plot()
```

Out[ ]: <Axes: >



In [ ]:
```
term_spread = curve_UST_returns['10 Y'] - curve_UST_returns['1 M']

term_spread = term_spread.to_frame(name='Term_Spread')
```

In [ ]:
```
term_spread
```

Out[ ]:

	Term_Spread
Price Date	
2007-03-01	-0.51
2007-04-01	0.26
2007-05-01	0.25
2007-06-01	0.29
2007-07-01	0.63
...	...
2024-09-01	-0.10
2024-10-01	0.38
2024-11-01	0.64
2024-12-01	-0.10
2025-01-01	0.76

215 rows × 1 columns

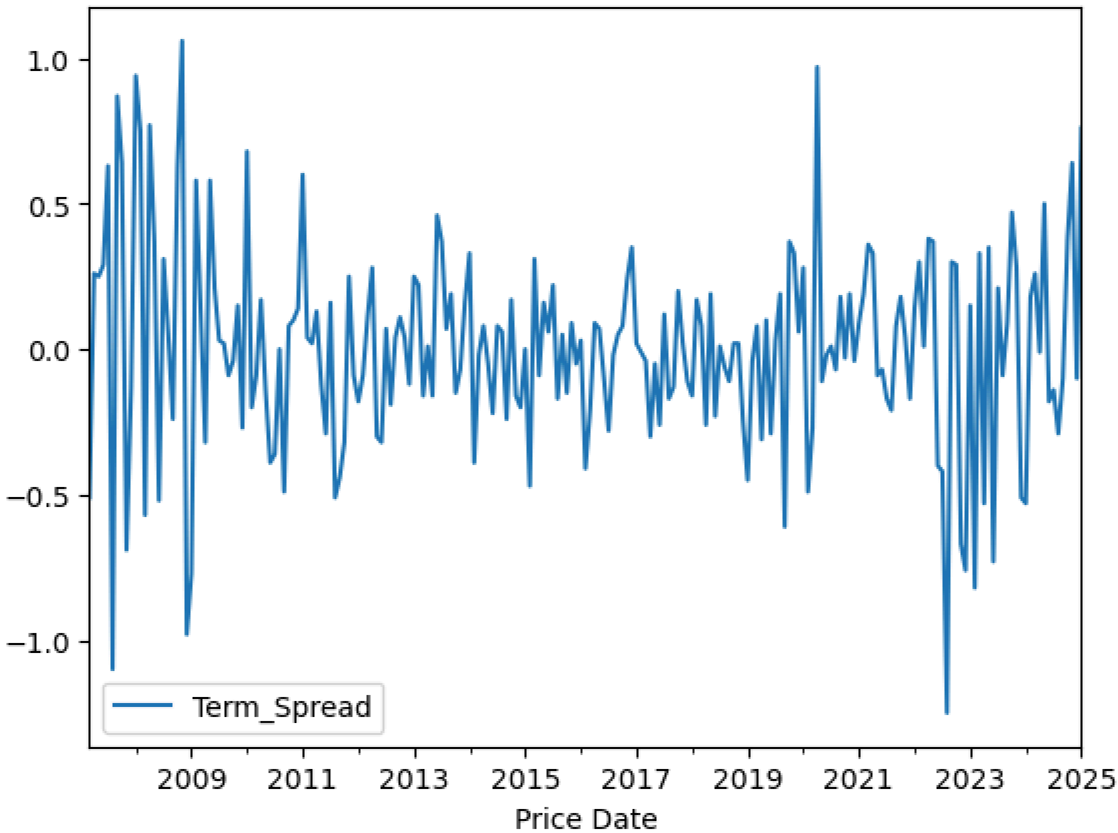
In [ ]:
```
term_spread.describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
Term_Spread	215.0	0.001628	0.352841	-1.25	-0.175	0.02	0.19	1.06

In [ ]:
```
term_spread.plot()
```

Out[ ]: <Axes: xlabel='Price Date'>



In [ ]:
```
factors = pd.read_csv(os.path.join(data_path, 'FF Research Data Factors.csv'
```

In [ ]:
```
factors.set_index('Date', inplace=True)
factors.sort_index(inplace=True)

factors
```

Out[ ]:

Date	Mkt-RF	SMB	HML	RF
1926-07-01	2.96	-2.56	-2.43	0.22
1926-08-01	2.64	-1.17	3.82	0.25
1926-09-01	0.36	-1.40	0.13	0.23
1926-10-01	-3.24	-0.09	0.70	0.32
1926-11-01	2.53	-0.10	-0.51	0.31
...	...	...	...	...
2024-08-01	1.61	-3.55	-1.13	0.48
2024-09-01	1.74	-0.17	-2.59	0.40
2024-10-01	-0.97	-1.01	0.89	0.39
2024-11-01	6.51	4.63	-0.05	0.40
2024-12-01	-3.17	-2.73	-2.95	0.37

1182 rows × 4 columns

In [ ]:
```
factors['Mkt'] = factors['Mkt-RF'] - factors['RF']
```

In [ ]:
```
def cross_section_market_cap_weighting(df):
     df_val_weight = (df['Return'] * df['marketcap'] / df['marketcap'].sum()
     return df_val_weight


# Fix a month, fix sector (so all bonds with the associated sector will be s
# Take their returns and value weight them
data.groupby(['Pricing_Date', 'Sector_Level_3'])[['Return', 'marketcap']].ap
plt.legend(bbox_to_anchor=(1, 1))
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x30f53bf80>
```

In [ ]:
```
df = data.groupby(['Pricing_Date', 'Sector_Level_3'])[['Return', 'marketcap'

df
```

Out[ ]:

	Pricing_Date	Sector_Level_3	Industry_Return_value_weighted
0	2008-07-01	Automotive	1.981068
1	2008-07-01	Banking	0.844094
2	2008-07-01	Basic Industry	1.513041
3	2008-07-01	Capital Goods	0.945575
4	2008-07-01	Consumer Cyclical	1.022311
...	...	...	...
3748	2024-09-01	Services	0.000000
3749	2024-09-01	Technology & Electronics	0.000000
3750	2024-09-01	Telecommunications	0.000000
3751	2024-09-01	Transportation	0.000000
3752	2024-09-01	Utility	0.000000

3753 rows × 3 columns

In [ ]:
```
data = data.merge(df, on=['Pricing_Date', 'Sector_Level_3'], how= 'outer')

data
```

Out[ ]:

	Pricing_Date	Index_Name	Cusip	ISIN	Description	IC
0	2008-07-01	C0A0	231021AJ5	US231021AJ54	CUMMINS ENGINE	
1	2008-07-01	C0A0	278058AX0	US278058AX04	EATON CORP	
2	2008-07-01	C0A0	478366AM9	US478366AM91	JOHNSON CONTROLS	
3	2008-07-01	C0A0	478366AN7	US478366AN74	JOHNSON CONTROLS	
4	2008-07-01	C0A0	478366AQ0	US478366AQ06	JOHNSON CONTROLS	
...	...	...	...	...	...	
720217	2024-09-01	C0A0	98389BAW	US98389BAW00	Xcel Energy Inc	
720218	2024-09-01	C0A0	98389BAX	US98389BAX82	Xcel Energy Inc	
720219	2024-09-01	C0A0	98389BAY	US98389BAY65	Xcel Energy Inc	
720220	2024-09-01	C0A0	98389BBA	US98389BBA70	Xcel Energy Inc	
720221	2024-09-01	C0A0	98389BBB	US98389BBB53	Xcel Energy Inc	

720222 rows × 173 columns

In [ ]:

```
value_weighted_industry_returns = data[['Pricing_Date', 'ISIN', 'Return', 'I

value_weighted_industry_returns
```

Out[ ]:

	Pricing_Date	ISIN	Return	Industry_Return_value_weighte
0	2008-07-01	US231021AJ54	3.661463	1.9810
1	2008-07-01	US278058AX04	0.967297	1.9810
2	2008-07-01	US478366AM91	1.161691	1.9810
3	2008-07-01	US478366AN74	3.640262	1.9810
4	2008-07-01	US478366AQ06	0.424859	1.9810
...	...	...	...	
720217	2024-09-01	US98389BAW00	NaN	0.0000
720218	2024-09-01	US98389BAX82	NaN	0.0000
720219	2024-09-01	US98389BAY65	NaN	0.0000
720220	2024-09-01	US98389BBA70	NaN	0.0000
720221	2024-09-01	US98389BBB53	NaN	0.0000

720222 rows × 4 columns

In [ ]:

```
reg_vars = factors.merge(
    value_weighted_industry_returns,
    left_index=True,
    right_on='Pricing_Date',
    how='inner'
).drop(columns=['Mkt-RF', 'SMB', 'HML', 'RF'])

reg_vars.set_index(['ISIN', 'Pricing_Date'], inplace= True)
reg_vars.sort_index(inplace = True)

reg_vars
```

Out[ ]:

		Mkt	Return	Industry_Return_value_weighted
ISIN	**Pricing_Date**			
US00081TAJ79	**2018-01-01**	5.45	-0.182482	-1.168591
	2018-02-01	-3.76	-1.162791	-1.326191
	2018-03-01	-2.46	0.435323	0.321425
	2018-04-01	0.15	-0.062189	-0.856887
	2018-05-01	2.51	0.437500	0.581216
...	**...**	...	...	...
XS2091666748	**2024-05-01**	3.90	2.613745	2.322924
	2024-06-01	2.36	1.974114	0.680991
	2024-07-01	0.79	2.901941	2.650981
	2024-08-01	1.13	2.227949	1.643104
	2024-09-01	1.34	NaN	0.000000

720222 rows × 3 columns

In [ ]:

```

```

In [ ]:

```
reg_vars.dropna(inplace = True)
```

In [ ]:

```
X = reg_vars[['Mkt', 'Industry_Return_value_weighted']].copy()
Y = reg_vars[['Return']].copy()
```

In [ ]:

```
# why we doing this?

Y.loc[:, 'Return_Industry_Adjusted'] = np.nan
```

In [ ]:

```
idx = pd.IndexSlice
```

In [ ]:

```
%%time

from sklearn.linear_model import LinearRegression

min_training_window = 24
training_window = 60

# We cannot let any leakage slip into the estimate of the industry return be
# therefore, the betas have to be calculated on a walkforward out-of-sample

bonds = sorted(reg_vars.index.get_level_values('ISIN').unique())

for bond in bonds:

    dates = reg_vars.loc[idx[bond, :], :].index.get_level_values('Pricing_Da
```

```
    for t in dates[min_training_window:]:
        X_train = X.loc[idx[bond, t - pd.DateOffset(months=training_window):
        Y_train = Y.loc[idx[bond, t - pd.DateOffset(months=training_window):

        model = LinearRegression()
        model.fit(X_train, Y_train)

        # X_test = vix[['VIX_lag']].iloc[vix.index.get_loc(t): vix.index.get
        Y_test = Y.loc[idx[bond, t: t + pd.DateOffset(months=1)], 'Return']

        # Save industry regression adjusted return
        industry_return = X.loc[idx[bond, t: t + pd.DateOffset(months=1)], '
        Y.loc[idx[bond, t: t + pd.DateOffset(months=1)], 'Return_Industry_Ac
```

```
CPU times: user 14min 44s, sys: 12.3 s, total: 14min 56s
Wall time: 15min 1s
```

In [ ]:

```
%%time

mom_6m_industry_adj = Y.groupby('ISIN', group_keys=False)['Return_Industry_A

mom_6m_industry_adj
```

```
CPU times: user 8.78 s, sys: 199 ms, total: 8.98 s
Wall time: 9.07 s
```

Out[ ]:

	ISIN	Pricing_Date	Mom_6m_Industry_Adj
0	US00081TAJ79	2018-01-01	NaN
1	US00081TAJ79	2018-02-01	NaN
2	US00081TAJ79	2018-03-01	NaN
3	US00081TAJ79	2018-04-01	NaN
4	US00081TAJ79	2018-05-01	NaN
...	...	...	...
704094	XS2091666748	2024-03-01	48.150180
704095	XS2091666748	2024-05-01	-503.698353
704096	XS2091666748	2024-06-01	482.753278
704097	XS2091666748	2024-07-01	21.556429
704098	XS2091666748	2024-08-01	-23.306881

704099 rows × 3 columns

In [ ]:

```
features = features.merge(mom_6m_industry_adj, on = ['Pricing_Date', 'ISIN']

features
```

Out[ ]:

	Pricing_Date	Index_Name	ISIN	Description	Maturity	Sec
0	2008-07-01	C0A0	US00184AAB17	TIME WARNER INC	2011-04-15	
1	2008-07-01	C0A0	US00184AAC99	AOL TIME WARNER	2031-04-15	
2	2008-07-01	C0A0	US00184AAF21	TIME WARNER INC	2012-05-01	
3	2008-07-01	C0A0	US00184AAG04	AOL TIME WARNER	2032-05-01	
4	2008-07-01	C0A0	US00209TAB17	COMCAST CABLE CO	2022-11-15	
...	...	...	...	...	...	
720217	2024-09-01	C0A0	US98978VAU70	Zoetis Inc.	2025-11-14	
720218	2024-09-01	C0A0	US98978VAV53	Zoetis Inc.	2032-11-16	
720219	2024-09-01	H0A0	US98980BAA17	ZipRecruiter Inc	2030-01-15	
720220	2024-09-01	H0A0	US98981BAA08	Zoominfo Technologies Llc /Zoominfo Financial ...	2029-02-01	
720221	2024-09-01	C0A0	XS2091666748	AT&T Inc	2050-03-01	

720222 rows × 40 columns

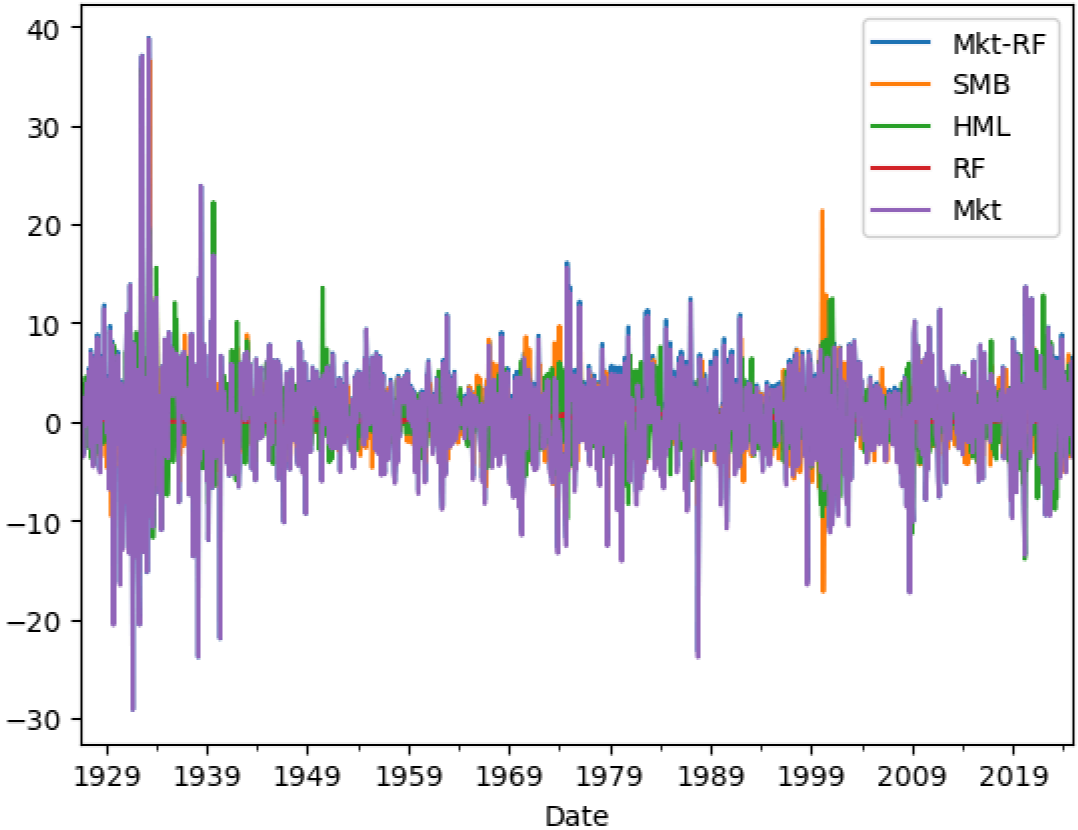
In [ ]:

```
# vix innovation

vix = pd.read_csv(os.path.join(data_path, 'VIX.csv'))
```

In [ ]:

```
factors.plot()
```

Out[ ]: <Axes: xlabel='Date'>

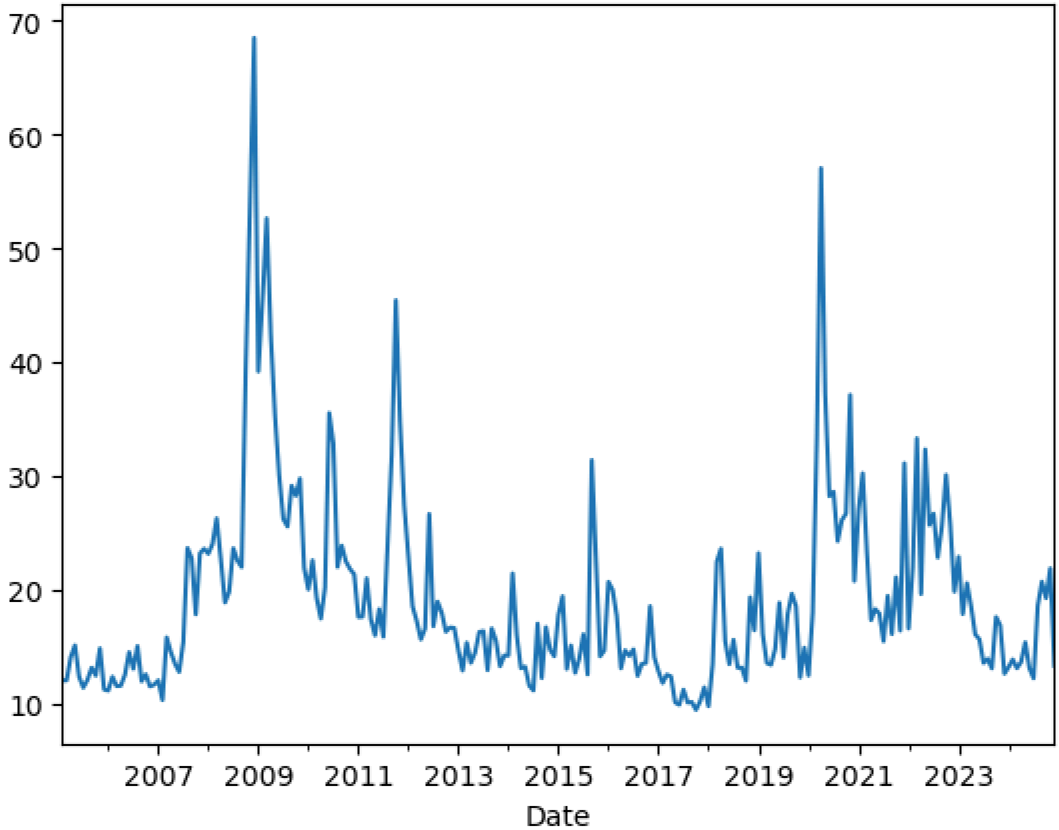


In [ ]:
```
factors = factors.merge(default_spread, left_index = True, right_index=True,
```

In [ ]:
```
factors = factors.merge(term_spread, left_index=True, right_index=True, how=
```

In [ ]:
```
vix = vix[['Price', 'Adj Close']].iloc[2:, :].rename(columns={'Price':'Date'
vix['Date'] = pd.to_datetime(vix['Date'])
vix['VIX'] = pd.to_numeric(vix['VIX'])
vix.set_index('Date', inplace=True)
vix.sort_index(inplace=True)

vix = vix.resample('MS').first()
vix['VIX_lag'] = vix['VIX'].shift()
vix.dropna(inplace=True)

vix['VIX'].plot()
```

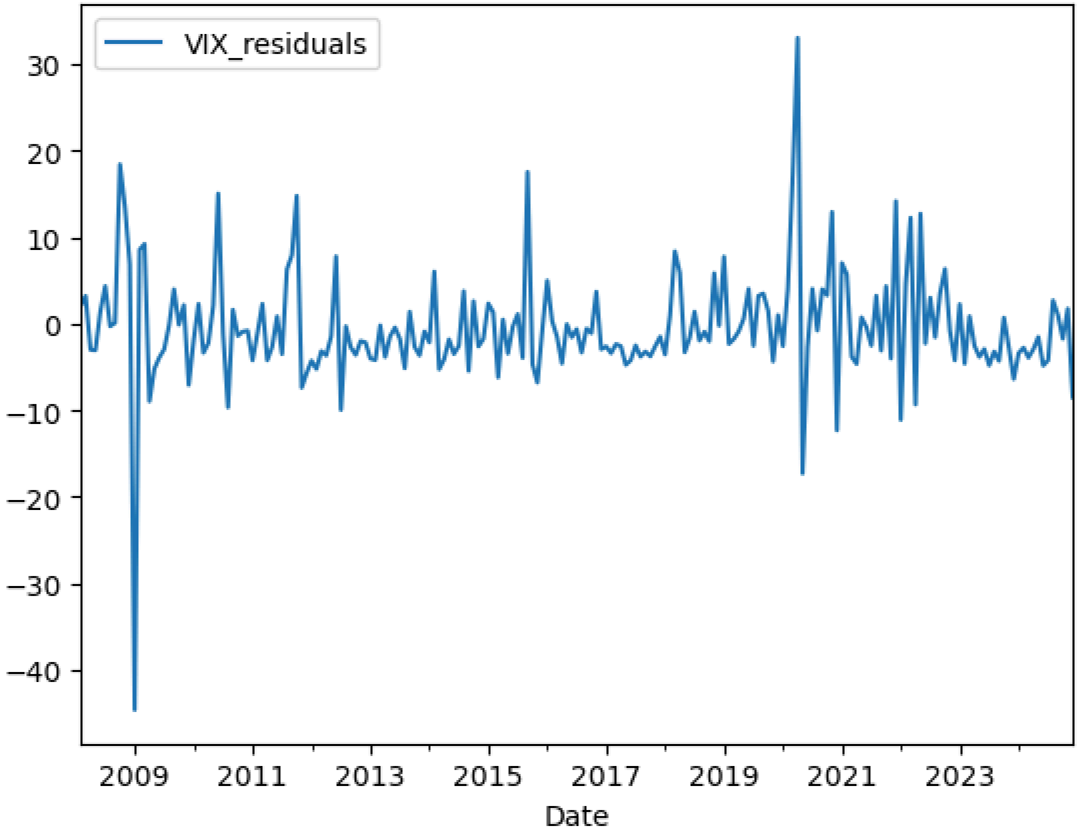
Out[ ]: <Axes: xlabel='Date'>



In [ ]:
```
from sklearn.linear_model import LinearRegression

vix['VIX_residuals'] = np.nan
min_training_window = 36
training_window = 60

# We cannot let any leakage slip into the estimate of the VIX beta,
# therefore, the residuals have to be calculated on a walkforward out-of-sam
for t in vix.index[min_training_window:]:

    X_train = vix[['VIX_lag']].iloc[max(0, vix.index.get_loc(t)-training_wir
    Y_train = vix[['VIX']].iloc[max(0, vix.index.get_loc(t)-training_window)
    model = LinearRegression()
    model.fit(X_train, Y_train)

    X_test = vix[['VIX_lag']].iloc[vix.index.get_loc(t): vix.index.get_loc(t
    Y_test = vix[['VIX']].iloc[vix.index.get_loc(t): vix.index.get_loc(t)+1]

    Y_test_hat = model.predict(X_test)
    vix.iloc[vix.index.get_loc(t): vix.index.get_loc(t)+1, -1] = Y_test - Y_
```

In [ ]:
```
vix.dropna()[['VIX_residuals']].plot()
```

Out[ ]: <Axes: xlabel='Date'>



In [ ]:
```
vix['VIX_residuals_lag'] = vix[['VIX_residuals']].shift()

vix.dropna(inplace = True)
```

In [ ]:
```
factors = factors.merge(vix[['VIX_residuals', 'VIX_residuals_lag']], left_ir
```

In [ ]:
```
factors
```

Out[ ]:

	Mkt-RF	SMB	HML	RF	Mkt	Default_Spread	Term_Spread	VIX_resi
2008-07-01	-0.77	2.60	5.42	0.15	-0.92	1.095309	0.31	4.3
2008-08-01	1.53	3.60	1.59	0.13	1.40	-5.412796	0.05	-0.2
2008-09-01	-9.24	-1.23	5.91	0.15	-9.39	-6.882607	-0.24	0.1
2008-10-01	-17.23	-2.60	-2.30	0.08	-17.31	4.667415	0.63	18.4
2008-11-01	-7.86	-2.85	-6.31	0.03	-7.89	7.074458	1.06	13.6
...	...	...	...	...	...	...	...	
2024-03-01	2.83	-2.51	4.22	0.43	2.40	1.058493	0.26	-3.9
2024-05-01	4.34	0.78	-1.67	0.44	3.90	1.417857	0.50	-1.5
2024-06-01	2.77	-3.06	-3.31	0.41	2.36	0.852650	-0.18	-4.8
2024-07-01	1.24	6.80	5.74	0.45	0.79	2.619706	-0.14	-4.2
2024-08-01	1.61	-3.55	-1.13	0.48	1.13	1.784745	-0.29	2.7

191 rows × 9 columns

In [ ]:

```
returns_over_riskfree = data[['Pricing_Date', 'ISIN', 'Return']].dropna()

returns_over_riskfree
```

Out[ ]:

	Pricing_Date	ISIN	Return
0	2008-07-01	US231021AJ54	3.661463
1	2008-07-01	US278058AX04	0.967297
2	2008-07-01	US478366AM91	1.161691
3	2008-07-01	US478366AN74	3.640262
4	2008-07-01	US478366AQ06	0.424859
...	...	...	...
713148	2024-08-01	US98389BAW00	1.571422
713149	2024-08-01	US98389BAX82	2.150305
713150	2024-08-01	US98389BAY65	1.534003
713151	2024-08-01	US98389BBA70	1.303814
713152	2024-08-01	US98389BBB53	1.567865

704099 rows × 3 columns

In [ ]:
```
returns_over_riskfree = returns_over_riskfree.merge(factors[['RF']], left_or
```

In [ ]:
```
returns_over_riskfree['Return_minus_Rf'] = returns_over_riskfree['Return'] -
```

In [ ]:
```
bonds = sorted(returns_over_riskfree['ISIN'].unique())
```

In [ ]:
```
reg_vars = factors.merge(returns_over_riskfree, left_index=True, right_on='P

reg_vars.set_index(['ISIN', 'Pricing_Date'], inplace=True)
reg_vars.sort_index(inplace=True)

reg_vars
```

Out[ ]:

ISIN	Pricing_Date	Mkt-RF	SMB	HML	Mkt	Default_Spread	Term_S
US00081TAJ79	2018-01-01	5.57	-3.12	-1.28	5.45	-0.570960	
	2018-02-01	-3.65	0.26	-1.04	-3.76	-1.616755	
	2018-03-01	-2.35	4.06	-0.20	-2.46	-0.128126	
	2018-04-01	0.29	1.13	0.54	0.15	-0.390668	
	2018-05-01	2.65	5.26	-3.22	2.51	0.296874	
...	...	...	...	...	...	...	
XS2091666748	2024-03-01	2.83	-2.51	4.22	2.40	1.058493	
	2024-05-01	4.34	0.78	-1.67	3.90	1.417857	
	2024-06-01	2.77	-3.06	-3.31	2.36	0.852650	
	2024-07-01	1.24	6.80	5.74	0.79	2.619706	
	2024-08-01	1.61	-3.55	-1.13	1.13	1.784745	

704099 rows × 9 columns

In [ ]:

```
X = reg_vars[['Mkt-RF', 'SMB', 'HML', 'Default_Spread', 'Term_Spread', 'VIX_
Y = reg_vars[['Return_minus_Rf']].copy()
```

In [ ]:

```
Y.loc[:, 'VIX_beta'] = np.nan
```

In [ ]:

```
%%time

from sklearn.linear_model import LinearRegression

min_training_window = 12
training_window_num_months = 60
training_window = pd.DateOffset(months=training_window_num_months)


# We cannot let any leakage slip into the estimate of the VIX beta,
# therefore, the betas have to be calculated on a walkforward out-of-sample

bonds = sorted(returns_over_riskfree['ISIN'].unique())

for bond in bonds:

    dates = reg_vars.loc[idx[bond, :], :].index.get_level_values('Pricing_Da

    for t in dates[min_training_window:]:
        X_train = X.loc[idx[bond, t - training_window: t-pd.DateOffset(month
        Y_train = Y.loc[idx[bond, t - training_window: t-pd.DateOffset(month

        if len(X_train) > 0:
            model = LinearRegression()
```

```python
            model.fit(X_train, Y_train)

            # X_test = X.iloc[idx[bond, t: t+1], :]
            # Y_test = Y.iloc[idx[bond, t: t+1], :]
            # Y_test_hat = model.predict(X_test)

            # Save VIX betas
            Y.loc[idx[bond, t], 'VIX_beta'] = model.coef_[-2:].sum()
```

```
CPU times: user 9min 33s, sys: 2.25 s, total: 9min 35s
Wall time: 9min 39s
```

In [ ]:
```python
Y.isna().sum() / Y.shape[0]
```

Out[ ]:
```
Return_minus_Rf    0.000000
VIX_beta           0.249025
dtype: float64
```

In [ ]:
```python
Y.head()
```

Out[ ]:

		Return_minus_Rf	VIX_beta
ISIN	**Pricing_Date**		
US00081TAJ79	**2018-01-01**	-0.302482	NaN
	2018-02-01	-1.272791	NaN
	2018-03-01	0.325323	NaN
	2018-04-01	-0.202189	NaN
	2018-05-01	0.297500	NaN

In [ ]:
```python
Y.groupby('Pricing_Date')['VIX_beta'].mean().plot()
```

Out[ ]:
```
<Axes: xlabel='Pricing_Date'>
```

In [ ]:
```
vix_beta = Y[['VIX_beta']].reset_index()

vix_beta
```

Out[ ]:

	ISIN	Pricing_Date	VIX_beta
0	US00081TAJ79	2018-01-01	NaN
1	US00081TAJ79	2018-02-01	NaN
2	US00081TAJ79	2018-03-01	NaN
3	US00081TAJ79	2018-04-01	NaN
4	US00081TAJ79	2018-05-01	NaN
...	...	...	...
704094	XS2091666748	2024-03-01	0.012102
704095	XS2091666748	2024-05-01	0.016489
704096	XS2091666748	2024-06-01	0.014594
704097	XS2091666748	2024-07-01	0.013028
704098	XS2091666748	2024-08-01	0.013026

704099 rows × 3 columns

In [ ]:
```
features = features.merge(vix_beta, on=['Pricing_Date', 'ISIN'], how='outer'

features
```

Out[ ]:

	Pricing_Date	Index_Name	ISIN	Description	Maturity	Sec
0	2008-07-01	C0A0	US00184AAB17	TIME WARNER INC	2011-04-15	
1	2008-07-01	C0A0	US00184AAC99	AOL TIME WARNER	2031-04-15	
2	2008-07-01	C0A0	US00184AAF21	TIME WARNER INC	2012-05-01	
3	2008-07-01	C0A0	US00184AAG04	AOL TIME WARNER	2032-05-01	
4	2008-07-01	C0A0	US00209TAB17	COMCAST CABLE CO	2022-11-15	
...	...	...	...	...	...	
720217	2024-09-01	C0A0	US98978VAU70	Zoetis Inc.	2025-11-14	
720218	2024-09-01	C0A0	US98978VAV53	Zoetis Inc.	2032-11-16	
720219	2024-09-01	H0A0	US98980BAA17	ZipRecruiter Inc	2030-01-15	
720220	2024-09-01	H0A0	US98981BAA08	Zoominfo Technologies Llc /Zoominfo Financial ...	2029-02-01	
720221	2024-09-01	C0A0	XS2091666748	AT&T Inc	2050-03-01	

720222 rows × 41 columns

In [ ]:

```
features[['VIX_beta']].describe().T
```

Out[ ]:

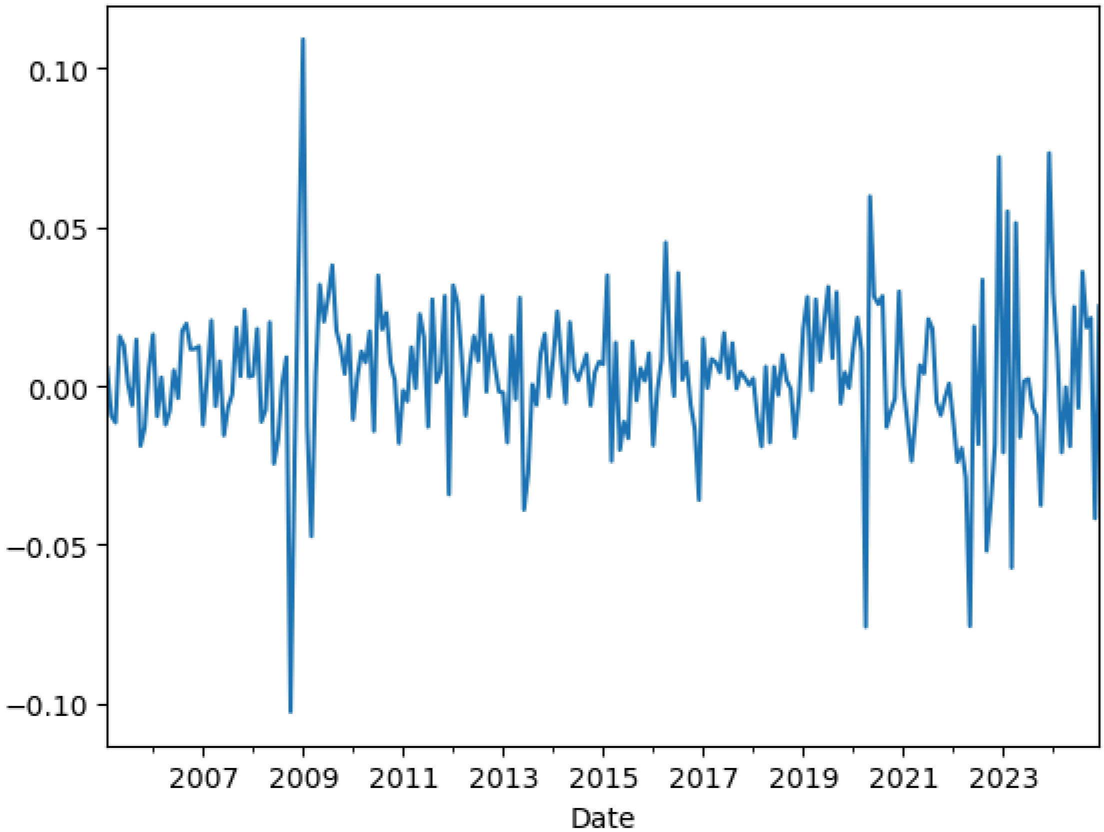
	count	mean	std	min	25%	50%	75
VIX_beta	528761.0	-0.006244	0.16075	-18.367795	-0.056468	-0.005209	0.0408

In [ ]:

```
LQD = pd.read_csv(os.path.join(data_path, 'LQD.csv'))


LQD = LQD[['Price', 'Adj Close']].iloc[2:, :].rename(columns={'Price':'Date'
LQD['Date'] = pd.to_datetime(LQD['Date'])
LQD['LQD'] = pd.to_numeric(LQD['LQD'])
LQD.set_index('Date', inplace=True)
LQD.sort_index(inplace=True)

LQD = LQD.resample('MS').first()
LQD['LQD_Return'] = LQD['LQD'].pct_change()
LQD.dropna(inplace=True)
```

```
LQD['LQD_Return'].plot()
```

Out[ ]: <Axes: xlabel='Date'>



In [ ]:
```
reg_vars = curve_UST_returns.merge(LQD, left_index=True, right_index=True, h

reg_vars.sort_index(inplace=True)

reg_vars
```

Out[ ]:

	1 M	2 M	3 M	6 M	1 Y	2 Y	3 Y	5 Y	7 Y	10 Y	20 Y	30
2007-03-01	0.24	0.140	0.04	-0.04	-0.13	-0.29	-0.30	-0.30	-0.29	-0.27	-0.24	-0.
2007-04-01	-0.17	-0.145	-0.12	-0.06	-0.06	-0.07	-0.01	0.02	0.05	0.09	0.14	0.
2007-05-01	-0.27	-0.200	-0.13	-0.03	-0.01	0.02	0.00	-0.03	-0.03	-0.02	-0.04	-0.
2007-06-01	-0.02	-0.100	-0.18	-0.07	0.06	0.32	0.34	0.35	0.32	0.27	0.22	0.
2007-07-01	-0.50	-0.205	0.09	-0.03	-0.04	-0.05	0.01	0.06	0.09	0.13	0.11	0.
...	...	...	...	...	...	...	...	...	...	...	...	
2024-08-01	0.02	0.040	-0.07	-0.19	-0.36	-0.42	-0.56	-0.23	-0.36	-0.27	-0.17	-0.
2024-09-01	-0.08	-0.190	-0.20	-0.25	-0.35	-0.38	-0.17	-0.26	-0.17	-0.18	-0.16	-0.
2024-10-01	-0.48	-0.450	-0.48	-0.51	-0.40	-0.25	-0.21	-0.13	-0.13	-0.10	-0.09	-0.
2024-11-01	-0.17	-0.110	-0.09	0.05	0.29	0.50	-0.06	0.57	0.54	0.47	0.39	0.
2024-12-01	0.00	-0.070	-0.06	-0.01	0.03	-0.03	0.58	-0.10	-0.11	-0.10	-0.13	-0.

214 rows × 13 columns

In [ ]:

```
# X = reg_vars[['1 M', '2 M', '3 M', '6 M', '1 Y', '2 Y', '3 Y', '5 Y', '7 Y
X = reg_vars[['1 M', '2 Y', '10 Y', '30 Y']].copy()
Y = reg_vars[['LQD_Return']].copy()
```

In [ ]:

```
Y.loc[:, 'Bond_market_excess_return'] = np.nan
```

In [ ]:

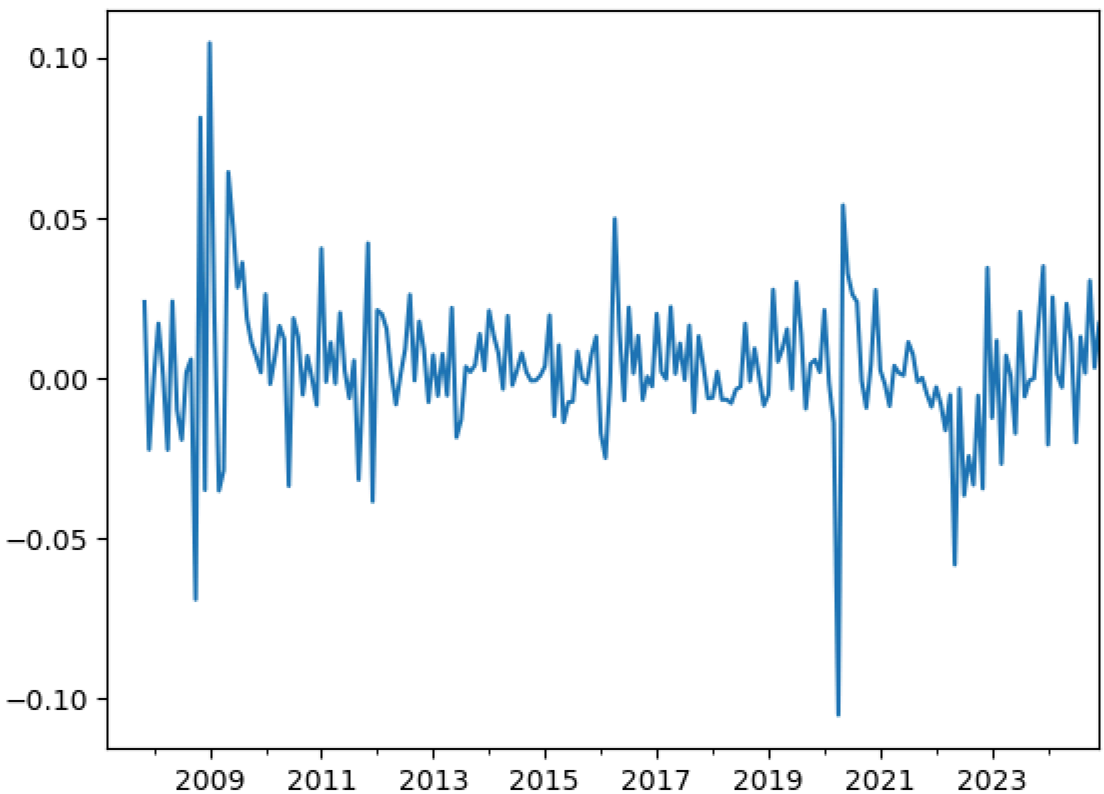
```
%%time

from sklearn.linear_model import LinearRegression

min_training_window = 8
training_window = 60

# We cannot let any leakage slip into the estimate of the VIX beta,
# therefore, the residuals have to be calculated on a walkforward out-of-sam
for t in reg_vars.index[min_training_window:]:

    X_train = X.iloc[max(0, X.index.get_loc(t)-training_window): X.index.get
    Y_train = Y[['LQD_Return']].iloc[max(0, Y.index.get_loc(t)-training_winc

    model = LinearRegression(fit_intercept=False)
    model.fit(X_train, Y_train)
```

```
    X_test = X.iloc[X.index.get_loc(t): X.index.get_loc(t)+1]
    Y_test = Y[['LQD_Return']].iloc[Y.index.get_loc(t): Y.index.get_loc(t)+1

    Y_test_hat = model.predict(X_test)

    Y.iloc[Y.index.get_loc(t): Y.index.get_loc(t)+1, -1] = Y_test - Y_test_h
```

```
CPU times: user 217 ms, sys: 8.29 ms, total: 225 ms
Wall time: 229 ms
```

In [ ]:
```
Y.loc[:, 'Bond_market_excess_return'].plot()
```

Out[ ]: <Axes: >



In [ ]:
```
reg_vars = Y.reset_index(names='Date').merge(returns_over_riskfree, left_on=

reg_vars.set_index(['ISIN', 'Pricing_Date'], inplace=True)
reg_vars.sort_index(inplace=True)

reg_vars
```

Out[ ]:

ISIN	Pricing_Date	Bond_market_excess_return	RF	Return_minus_
US00081TAJ79	**2018-01-01**	-0.006134	0.12	-0.3024
	2018-02-01	0.001982	0.11	-1.2727
	2018-03-01	-0.006762	0.11	0.3253
	2018-04-01	-0.006740	0.14	-0.2021
	2018-05-01	-0.007839	0.14	0.2975
...	**...**	...	...	
XS2091666748	**2024-03-01**	0.001238	0.43	0.3446
	2024-05-01	0.023151	0.44	2.1737
	2024-06-01	0.011337	0.41	1.5641
	2024-07-01	-0.020069	0.45	2.4519
	2024-08-01	0.012830	0.48	1.7479

704099 rows × 3 columns

In [ ]:
```
X = reg_vars[['Bond_market_excess_return']].copy()
Y = reg_vars[['Return_minus_Rf']].copy()
```

In [ ]:
```
Y.loc[:, 'Bond_beta'] = np.nan
```

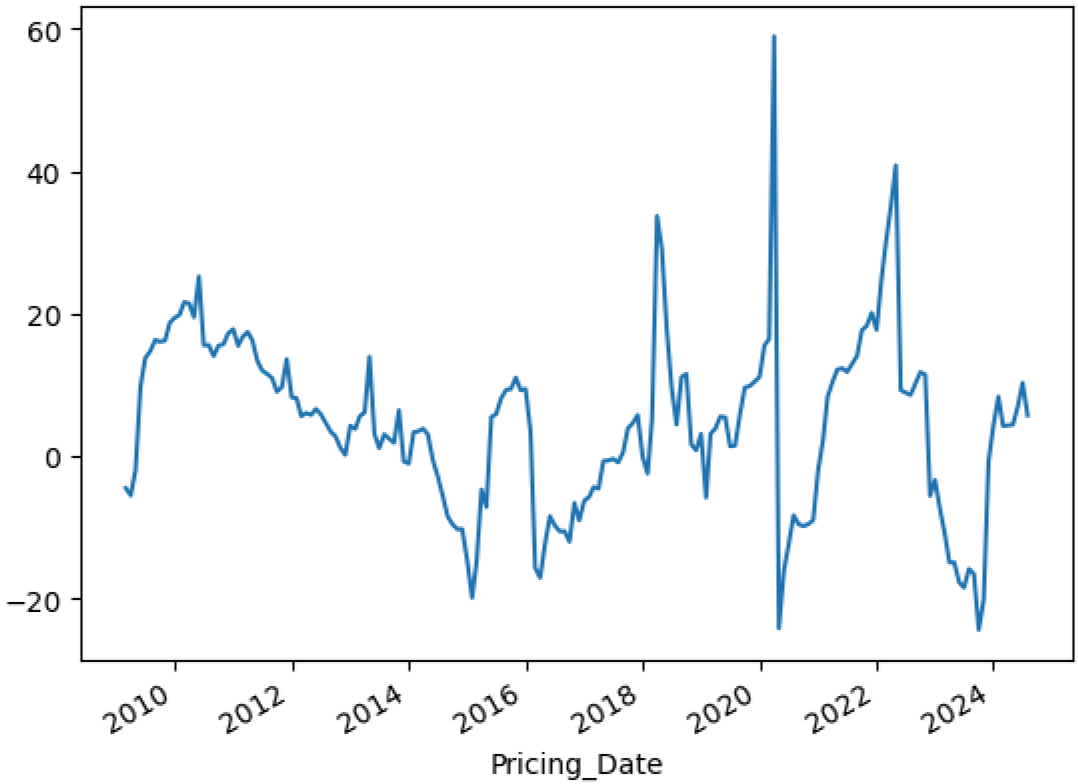
In [ ]:
```
%%time

from sklearn.linear_model import LinearRegression

min_training_window = 8
training_window = 60

# We cannot let any leakage slip into the estimate of the VIX beta,
# therefore, the betas have to be calculated on a walkforward out-of-sample

bonds = sorted(returns_over_riskfree['ISIN'].unique())

for bond in bonds:

    dates = reg_vars.loc[idx[bond, :], :].index.get_level_values('Pricing_Da

    for t in dates[min_training_window:]:

        X_train = X.loc[idx[bond, t - pd.DateOffset(months=training_window):
        Y_train = Y.loc[idx[bond, t - pd.DateOffset(months=training_window):

        if len(X_train) > 0:
            model = LinearRegression()
            model.fit(X_train, Y_train)
```

```
            # Save Bond betas
            Y.loc[idx[bond, t], 'Bond_beta'] = model.coef_
```

```
CPU times: user 10min 34s, sys: 3.53 s, total: 10min 38s
Wall time: 10min 44s
```

In [ ]:
```
Y.groupby('Pricing_Date')['Bond_beta'].mean().plot()
```

Out[ ]: <Axes: xlabel='Pricing_Date'>



In [ ]:
```
bond_beta = Y[['Bond_beta']].reset_index()

bond_beta
```

Out[ ]:

	ISIN	Pricing_Date	Bond_beta
0	US00081TAJ79	2018-01-01	NaN
1	US00081TAJ79	2018-02-01	NaN
2	US00081TAJ79	2018-03-01	NaN
3	US00081TAJ79	2018-04-01	NaN
4	US00081TAJ79	2018-05-01	NaN
...	...	...	...
704094	XS2091666748	2024-03-01	-9.725471
704095	XS2091666748	2024-05-01	-9.701470
704096	XS2091666748	2024-06-01	-7.815373
704097	XS2091666748	2024-07-01	-7.206721
704098	XS2091666748	2024-08-01	-8.603905

704099 rows × 3 columns

In [ ]:

```
features = features.merge(bond_beta, on=['Pricing_Date', 'ISIN'], how='outer

features
```

Out[ ]:

	Pricing_Date	Index_Name	ISIN	Description	Maturity	Sec
0	2008-07-01	C0A0	US00184AAB17	TIME WARNER INC	2011-04-15	
1	2008-07-01	C0A0	US00184AAC99	AOL TIME WARNER	2031-04-15	
2	2008-07-01	C0A0	US00184AAF21	TIME WARNER INC	2012-05-01	
3	2008-07-01	C0A0	US00184AAG04	AOL TIME WARNER	2032-05-01	
4	2008-07-01	C0A0	US00209TAB17	COMCAST CABLE CO	2022-11-15	
...	...	...	...	...	...	
720217	2024-09-01	C0A0	US98978VAU70	Zoetis Inc.	2025-11-14	
720218	2024-09-01	C0A0	US98978VAV53	Zoetis Inc.	2032-11-16	
720219	2024-09-01	H0A0	US98980BAA17	ZipRecruiter Inc	2030-01-15	
720220	2024-09-01	H0A0	US98981BAA08	Zoominfo Technologies Llc /Zoominfo Financial ...	2029-02-01	
720221	2024-09-01	C0A0	XS2091666748	AT&T Inc	2050-03-01	

720222 rows × 42 columns

In [ ]:

```
features.to_parquet(os.path.join(data_path, 'IPCA_Features_Monthly.parquet')
```