Modeling Corporate Bond Returns - IPCA

Imports

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
! pip install ipca
import ipca
from ipca import InstrumentedPCA
```

```
Requirement already satisfied: ipca in /opt/miniconda3/envs/py3k/lib/python
3.12/site-packages (0.6.7)
Requirement already satisfied: numpy in /opt/miniconda3/envs/py3k/lib/python
3.12/site-packages (from ipca) (1.26.4)
Requirement already satisfied: progressbar in /opt/miniconda3/envs/py3k/lib/
python3.12/site-packages (from ipca) (2.5)
Requirement already satisfied: numba in /opt/miniconda3/envs/py3k/lib/python
3.12/site-packages (from ipca) (0.61.0)
Requirement already satisfied: scipy in /opt/miniconda3/envs/py3k/lib/python
3.12/site-packages (from ipca) (1.13.1)
Requirement already satisfied: joblib in /opt/miniconda3/envs/py3k/lib/pytho
n3.12/site-packages (from ipca) (1.4.2)
Requirement already satisfied: scikit-learn in /opt/miniconda3/envs/py3k/li
b/python3.12/site-packages (from ipca) (1.6.1)
Requirement already satisfied: llvmlite<0.45,>=0.44.0dev0 in /opt/miniconda
3/envs/py3k/lib/python3.12/site-packages (from numba->ipca) (0.44.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in /opt/miniconda3/envs/
py3k/lib/python3.12/site-packages (from scikit-learn->ipca) (3.5.0)
```

In [ ]:
```python
pd.reset_option('display.max_rows')
pd.reset_option('display.max_columns')
```

In [ ]:
```
# ! pip install numpy==1.22.4
```

In [ ]:
```
os.getcwd()
```

Out[ ]:
```
'/Users/niehuapeng/Desktop/IPCA'
```

Read in Data:

In [ ]:
```
project_path = '/content/drive/My Drive/DSO 585 - Data Driven Consulting/Cor

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

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

Out[ ]:
```
['UST Curve.csv',
 'Bond_Stock_Monthly_Data.parquet',
 'requirements.txt',
 'VIX.csv',
 'LQD.csv',
 'SIC Equity Coverage as of Nov 30 2023 (1).xlsx',
 'REL_VAL',
 'FF_Research_Data_Factors.csv',
 'IPCA_Bond_Factor_Pricing_Model.ipynb',
 'OLD_IPCA_Features.csv',
 'Monthly_Stock_Bond_Data_Creation.ipynb',
 't-bond.csv',
 'IPCA_Features_Monthly.parquet',
 'oos_results_120_25.csv',
 'oos_forecasts_120_25.csv']
```

In [ ]:
```
# Set the device to GPU if available
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

In [ ]:
```
bond_stock = pd.read_parquet(os.path.join(data_path, 'Bond_Stock_Monthly_Dat
```

In [ ]:
```
for i in bond_stock.columns:
  print(i)
```

Pricing_Date
Index_Name
Cusip
ISIN
Description
ICE_Ticker
Maturity
Seniority
Coupon
Sector_Level_1
Sector_Level_2
Sector_Level_3
Sector_Level_4
Sector_Code
ParAmount
Yield2Worst
Price_BOM
Price_EOM
oas_BOM
oas_EOM
DUR
RR
PD_DRISK
RR40Spread
Rating
Idx_Wgt
Return
bondRV
BS_Spread
Decile
StarMine_ID
PD_StarMine
sentiment_score
Stock_Ticker
Primary_Exchange
Stock_Price
Stock_Price_Date
REPORTING_CALENDAR_DATE
STARTING_LAG
ENDING_LAG
fiscal_year
fiscal_period
revenuegrowth
nopat
nopatmargin
investedcapital
investedcapitalturnover
investedcapitalincreasedecrease
freecashflow
netnonopex
netnonopobligations
ebit
depreciationandamortization
ebitda
capex
dfcfnwc

dfnwc
nwc
debt
ltdebtandcapleases
netdebt
totalcapital
bookvaluepershare
tangbookvaluepershare
marketcap
enterprisevalue
pricetobook
pricetotangiblebook
pricetorevenue
pricetoearnings
dividendyield
earningsyield
evtoinvestedcapital
evtorevenue
evtoebitda
evtoebit
evtonopat
evtoocf
evtofcff
ebitdagrowth
ebitgrowth
nopatgrowth
netincomegrowth
epsgrowth
ocfgrowth
fcffgrowth
investedcapitalgrowth
revenueqoqgrowth
ebitdaqoqgrowth
ebitqoqgrowth
nopatqoqgrowth
netincomeqoqgrowth
epsqoqgrowth
ocfqoqgrowth
fcffqoqgrowth
investedcapitalqoqgrowth
grossmargin
ebitdamargin
operatingmargin
ebitmargin
profitmargin
costofrevtorevenue
sgaextorevenue
rdextorevenue
opextorevenue
taxburdenpct
interestburdenpct
efftaxrate
assetturnover
arturnover
invturnover
faturnover

apturnover
dso
dio
dpo
ccc
finleverage
leverageratio
compoundleveragefactor
ltdebttoequity
debttoequity
roic
nnep
roicnnepspread
rnnoa
roe
croic
oroa
roa
noncontrollinginterestsharingratio
roce
divpayoutratio
augmentedpayoutratio
ocftocapex
stdebttocap
ltdebttocap
debttototalcapital
preferredtocap
noncontrolinttocap
commontocap
debttoebitda
netdebttoebitda
ltdebttoebitda
debttonopat
netdebttonopat
ltdebttonopat
altmanzscore
ebittointerestex
nopattointerestex
ebitlesscapextointerestex
nopatlesscapextointex
ocftointerestex
ocflesscapextointerestex
fcfftointerestex
currentratio
quickratio
dfcfnwctorev
dfnwctorev
nwctorev
normalizednopat
normalizednopatmargin
pretaxincomemargin
adjweightedavebasicsharesos
adjbasiceps
adjweightedavedilutedsharesos
adjdilutedeps
adjweightedavebasicdilutedsharesos

```
adjbasicdilutedeps
roe_simple
```

In [ ]:
```
data = pd.read_csv(os.path.join(data_path, 'OLD_IPCA_Features.csv'))
```

In [ ]:
```
data
```

Out[ ]:

	Date	Index	Cusip	Company	Industry	Excess_returns	Issue_
0	2010-01-01	C0A0	581557AU	MCKESSON CORP	Healthcare	0.232	2009-(
1	2010-01-01	C0A0	581557AV	MCKESSON CORP	Healthcare	-0.203	2009-(
2	2010-01-01	C0A0	581557AW	MCKESSON CORP	Healthcare	0.439	2009-(
3	2010-01-01	C0A0	581557AX	MCKESSON CORP	Healthcare	-0.660	2009-(
4	2010-01-01	C0A0	58155QAA	MCKESSON CORP	Healthcare	0.641	2009-(
...	...	...	...	...	...	...	
228801	2023-05-01	C0A0	98978VAM	Zoetis Inc.	Healthcare	0.329	2017-:
228802	2023-05-01	C0A0	98978VAN	Zoetis Inc.	Healthcare	0.095	2018-(
228803	2023-05-01	C0A0	98978VAP	Zoetis Inc.	Healthcare	0.081	2018-(
228804	2023-05-01	C0A0	98978VAS	Zoetis Inc.	Healthcare	-0.147	2020-(
228805	2023-05-01	C0A0	98978VAT	Zoetis Inc.	Healthcare	-1.715	2020-(

228806 rows × 56 columns

Applying IPCA

In [ ]:
```
# run on collab
features = pd.read_csv(os.path.join(data_path, 'OLD_IPCA_Features.csv'), par
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[4], line 2
      1 # run on collab
----> 2 features = pd.read_csv(os.path.join(data_path, 'OLD_IPCA_Features.cs
v'), parse_dates=[0,6])

NameError: name 'data_path' is not defined
```

In [ ]:
```python
# run on local
features = pd.read_csv('Data/IPCA Features.csv', parse_dates=[0,6])
```

Note that this dataframe only contains investment-grade bonds

In [ ]:
```python
features
```

Out[ ]:

	Date	Index	Cusip	Company	Industry	Excess_returns	Issue_
0	2010-01-01	C0A0	581557AU	MCKESSON CORP	Healthcare	0.232	2009-(
1	2010-01-01	C0A0	581557AV	MCKESSON CORP	Healthcare	-0.203	2009-(
2	2010-01-01	C0A0	581557AW	MCKESSON CORP	Healthcare	0.439	2009-(
3	2010-01-01	C0A0	581557AX	MCKESSON CORP	Healthcare	-0.660	2009-(
4	2010-01-01	C0A0	58155QAA	MCKESSON CORP	Healthcare	0.641	2009-(
...	...	...	...	...	...	...	
228801	2023-05-01	C0A0	98978VAM	Zoetis Inc.	Healthcare	0.329	2017-:
228802	2023-05-01	C0A0	98978VAN	Zoetis Inc.	Healthcare	0.095	2018-(
228803	2023-05-01	C0A0	98978VAP	Zoetis Inc.	Healthcare	0.081	2018-(
228804	2023-05-01	C0A0	98978VAS	Zoetis Inc.	Healthcare	-0.147	2020-(
228805	2023-05-01	C0A0	98978VAT	Zoetis Inc.	Healthcare	-1.715	2020-(

228806 rows × 56 columns

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

Bond Excess Returns Construction R_t:

In [ ]:
```python
returns = features[['Excess_returns']].copy()
returns_unstacked = returns.unstack(level='Cusip')

returns_unstacked
```

Out[ ]:

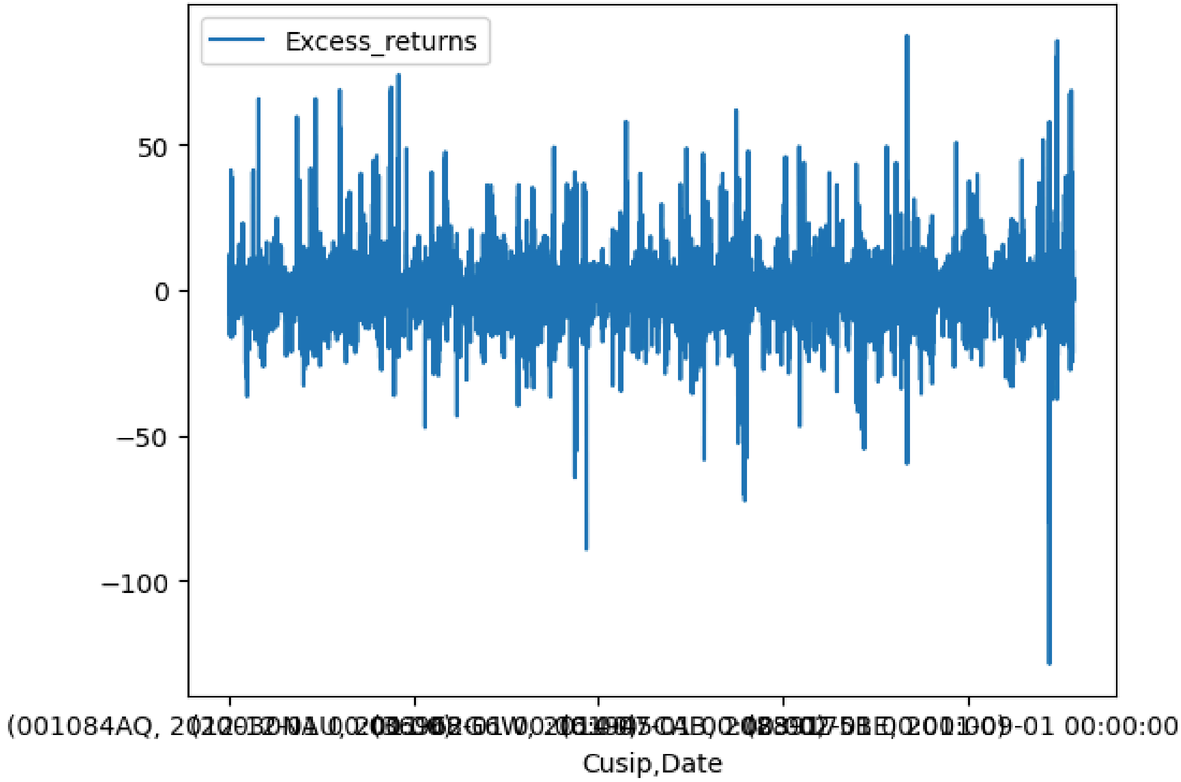
Cusip Date	001084AQ	00108WAD	00108WAF	00108WAH	00108WAJ	00108WAK
2010-01-01	NaN	NaN	NaN	NaN	NaN	NaN
2010-02-01	NaN	NaN	NaN	NaN	NaN	NaN
2010-03-01	NaN	NaN	NaN	NaN	NaN	NaN
2010-04-01	NaN	NaN	NaN	NaN	NaN	NaN
2010-05-01	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...
2023-01-01	NaN	NaN	NaN	NaN	NaN	NaN
2023-02-01	NaN	NaN	NaN	NaN	NaN	NaN
2023-03-01	NaN	NaN	NaN	NaN	NaN	NaN
2023-04-01	NaN	NaN	NaN	NaN	NaN	NaN
2023-05-01	NaN	NaN	NaN	NaN	NaN	NaN

161 rows × 6704 columns

In [ ]:

```
returns.plot()
```

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



Lagged Characteristics Construction X_{t-1}:

In [ ]:
```
features = features.drop(columns = ['Excess_returns', 'Issue_date', 'Index',

features_lag = features.groupby('Cusip').shift().dropna()

features_lag
```

Out[ ]:

Cusip	Date	Bond_age	Face_value	Coupon	Duration	Spread	Ratin
001084AQ	**2013-01-01**	0.501370	280000000.0	5.875	7.037407	345.2777	11.
	2013-02-01	0.586301	280000000.0	5.875	6.971084	320.7451	11.
	2013-03-01	0.671233	280000000.0	5.875	6.903765	333.3739	11.
	2013-04-01	0.747945	280000000.0	5.875	6.810539	335.8488	11.
	2013-05-01	0.832877	280000000.0	5.875	6.690296	380.7369	11.
...	**...**	...	...	...	...	...	.
98978VAT	**2023-01-01**	2.501370	46700.0	3.000	16.500000	112.0000	9.
	2023-02-01	2.586301	46100.0	3.000	16.277000	113.0000	9.
	2023-03-01	2.671233	47100.0	3.000	16.586000	112.0000	9.
	2023-04-01	2.747945	45500.0	3.000	16.002000	120.0000	9.
	2023-05-01	2.832877	46700.0	3.000	16.359000	115.0000	9.

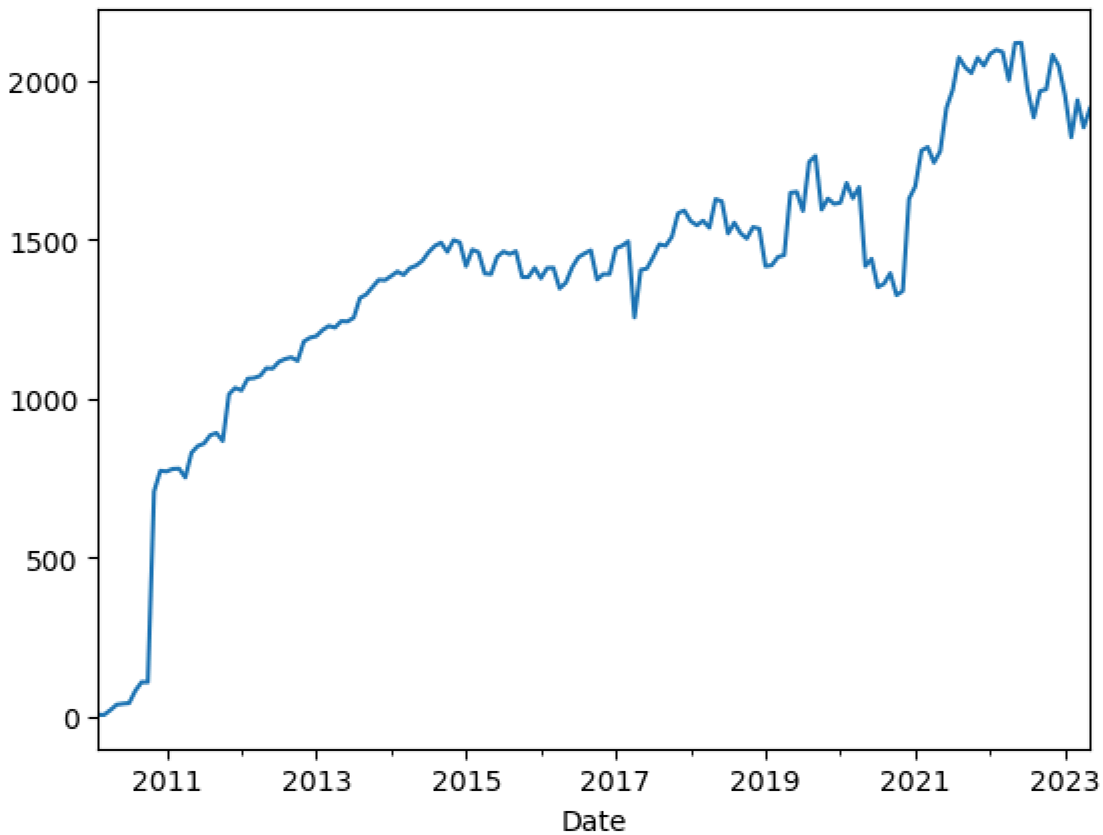
222102 rows × 49 columns

In [ ]:
```
# Intersect characteristics index with excess returns index
returns = returns.loc[features_lag.index, :]
```

In [ ]:
```
num_bonds_ts = returns.groupby('Date')['Excess_returns'].apply(len)

print(num_bonds_ts.describe().T)

num_bonds_ts.plot()
```

```
count     160.000000
mean     1388.137500
std       456.052359
min         5.000000
25%      1243.750000
50%      1442.500000
75%      1618.000000
max      2120.000000
Name: Excess_returns, dtype: float64
```

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



IPCA:

In [ ]:
```
%%time

regr = ipca.InstrumentedPCA(n_factors=4, intercept=False)
regr = regr.fit(X=features_lag, y=returns.squeeze())
# Gamma, Factors = regr.get_factors(label_ind=True)


ret_hat_package = pd.DataFrame(regr.predict(features_lag),
                               index=features_lag.index,
                               columns=['Returns_forecast'])

ret_ret_hat_package = pd.concat([returns, ret_hat_package], axis=1)

ret_ret_hat_package.head()
```

```
[=====                                                                    ]
7%
The panel dimensions are:
n_samples: 6492 , L: 49 , T: 160
[=========================================================================]
100%
```

```
Step 1 - Aggregate Update: 813804725295.03
Step 2 - Aggregate Update: 781.5505568797184
Step 3 - Aggregate Update: 29.336925545207606
Step 4 - Aggregate Update: 55.33930135423009
Step 5 - Aggregate Update: 33.01404888816989
Step 6 - Aggregate Update: 4.216090430455509
Step 7 - Aggregate Update: 4.2753194563758825
Step 8 - Aggregate Update: 3.8896141737522454
Step 9 - Aggregate Update: 3.1236680885060553
Step 10 - Aggregate Update: 2.7613263133624377
Step 11 - Aggregate Update: 2.266610263662481
Step 12 - Aggregate Update: 1.8906318788547232
Step 13 - Aggregate Update: 1.6128801042830272
Step 14 - Aggregate Update: 1.4109528339302795
Step 15 - Aggregate Update: 1.264892707101044
Step 16 - Aggregate Update: 1.1550748210520592
Step 17 - Aggregate Update: 1.0652731706221328
Step 18 - Aggregate Update: 0.9876094810747347
Step 19 - Aggregate Update: 0.920435638970762
Step 20 - Aggregate Update: 0.8639960414477059
Step 21 - Aggregate Update: 0.8311967184595481
Step 22 - Aggregate Update: 0.8611230422673479
Step 23 - Aggregate Update: 0.9009930563773025
Step 24 - Aggregate Update: 0.9494058664104408
Step 25 - Aggregate Update: 1.004620056336556
Step 26 - Aggregate Update: 1.0645562476913533
Step 27 - Aggregate Update: 1.126778663320522
Step 28 - Aggregate Update: 1.1884752904063518
Step 29 - Aggregate Update: 1.2464507198714045
Step 30 - Aggregate Update: 1.2971679533191107
Step 31 - Aggregate Update: 1.3368740356782673
Step 32 - Aggregate Update: 1.3910290627890376
Step 33 - Aggregate Update: 1.4774897854475881
Step 34 - Aggregate Update: 1.5526283014326268
Step 35 - Aggregate Update: 1.6120815846087666
Step 36 - Aggregate Update: 1.6518180643736997
Step 37 - Aggregate Update: 1.668780173681359
Step 38 - Aggregate Update: 1.6614440874041492
Step 39 - Aggregate Update: 1.630114620258123
Step 40 - Aggregate Update: 1.576861020719413
Step 41 - Aggregate Update: 1.50512700526248
Step 42 - Aggregate Update: 1.419154475562319
Step 43 - Aggregate Update: 1.3233883638981823
Step 44 - Aggregate Update: 1.2219930134546395
Step 45 - Aggregate Update: 1.1185420356340465
Step 46 - Aggregate Update: 1.0158805806907232
Step 47 - Aggregate Update: 0.9161180623690761
Step 48 - Aggregate Update: 0.8207028257277891
Step 49 - Aggregate Update: 0.7305387305601805
Step 50 - Aggregate Update: 0.6461070248169598
Step 51 - Aggregate Update: 0.5675814343887033
Step 52 - Aggregate Update: 0.49492522648934845
Step 53 - Aggregate Update: 0.42796740690085855
Step 54 - Aggregate Update: 0.3664603952850243
Step 55 - Aggregate Update: 0.31012078851922453
Step 56 - Aggregate Update: 0.2586559500147896
```

```
Step 57 - Aggregate Update: 0.2117785667478067
Step 58 - Aggregate Update: 0.16921510638949044
Step 59 - Aggregate Update: 0.130710224428908
Step 60 - Aggregate Update: 0.0960217566716679
Step 61 - Aggregate Update: 0.06492039013577156
Step 62 - Aggregate Update: 0.04610776705633368
Step 63 - Aggregate Update: 0.04373414128338515
Step 64 - Aggregate Update: 0.04140772463300546
Step 65 - Aggregate Update: 0.039145268725160554
Step 66 - Aggregate Update: 0.04416245337688984
Step 67 - Aggregate Update: 0.05809869820356006
Step 68 - Aggregate Update: 0.06985509487036978
Step 69 - Aggregate Update: 0.0796136309031521
Step 70 - Aggregate Update: 0.08755192359576291
Step 71 - Aggregate Update: 0.09383900636858833
Step 72 - Aggregate Update: 0.09863596965547394
Step 73 - Aggregate Update: 0.1020962140625219
Step 74 - Aggregate Update: 0.10436127048237331
Step 75 - Aggregate Update: 0.10556850682911012
Step 76 - Aggregate Update: 0.10584047084575232
Step 77 - Aggregate Update: 0.10529421155954033
Step 78 - Aggregate Update: 0.10403642262293644
Step 79 - Aggregate Update: 0.10216362948851554
Step 80 - Aggregate Update: 0.09976288892783103
Step 81 - Aggregate Update: 0.0969139553746281
Step 82 - Aggregate Update: 0.0936890444813514
Step 83 - Aggregate Update: 0.09015163645464774
Step 84 - Aggregate Update: 0.08635891612605562
Step 85 - Aggregate Update: 0.08236272152549162
Step 86 - Aggregate Update: 0.07820815866401176
Step 87 - Aggregate Update: 0.07393405973384226
Step 88 - Aggregate Update: 0.06957520764396996
Step 89 - Aggregate Update: 0.06516066596040915
Step 90 - Aggregate Update: 0.06071813666736503
Step 91 - Aggregate Update: 0.05626896397942538
Step 92 - Aggregate Update: 0.05412567265090118
Step 93 - Aggregate Update: 0.05211105896715296
Step 94 - Aggregate Update: 0.05008633670193774
Step 95 - Aggregate Update: 0.048074664696120095
Step 96 - Aggregate Update: 0.046070413938348764
Step 97 - Aggregate Update: 0.044067286947754525
Step 98 - Aggregate Update: 0.0420967973951889
Step 99 - Aggregate Update: 0.04014154041253093
Step 100 - Aggregate Update: 0.038208146591554026
Step 101 - Aggregate Update: 0.03630703136619928
Step 102 - Aggregate Update: 0.0344357514312712
Step 103 - Aggregate Update: 0.03258824002011096
Step 104 - Aggregate Update: 0.030774509745175038
Step 105 - Aggregate Update: 0.028986675959217223
Step 106 - Aggregate Update: 0.02721596753867317
Step 107 - Aggregate Update: 0.025487370799616826
Step 108 - Aggregate Update: 0.02378497159327253
Step 109 - Aggregate Update: 0.022114979575128757
Step 110 - Aggregate Update: 0.020475532153138687
Step 111 - Aggregate Update: 0.02024440110746184
Step 112 - Aggregate Update: 0.02306558037610884
```

```
Step 113 - Aggregate Update: 0.02609171064438165
Step 114 - Aggregate Update: 0.02907360283663607
Step 115 - Aggregate Update: 0.03201572317406054
Step 116 - Aggregate Update: 0.03492293434341143
Step 117 - Aggregate Update: 0.0378015325052683
Step 118 - Aggregate Update: 0.04065745432085777
Step 119 - Aggregate Update: 0.043499508580806534
Step 120 - Aggregate Update: 0.046328269688288515
Step 121 - Aggregate Update: 0.04915322187450499
Step 122 - Aggregate Update: 0.05197742521654902
Step 123 - Aggregate Update: 0.05480984298026215
Step 124 - Aggregate Update: 0.05765515118239328
Step 125 - Aggregate Update: 0.06052014463680955
Step 126 - Aggregate Update: 0.06341340358895842
Step 127 - Aggregate Update: 0.06634245651716952
Step 128 - Aggregate Update: 0.06930936407093746
Step 129 - Aggregate Update: 0.07232843207649786
Step 130 - Aggregate Update: 0.07540481825083489
Step 131 - Aggregate Update: 0.07854805619211191
Step 132 - Aggregate Update: 0.08176494302634119
Step 133 - Aggregate Update: 0.08506726405591714
Step 134 - Aggregate Update: 0.08846594382214334
Step 135 - Aggregate Update: 0.0919674407701514
Step 136 - Aggregate Update: 0.09558792133972283
Step 137 - Aggregate Update: 0.09933858041157428
Step 138 - Aggregate Update: 0.1032329014955522
Step 139 - Aggregate Update: 0.10728405066460311
Step 140 - Aggregate Update: 0.11150729568166717
Step 141 - Aggregate Update: 0.1159242288294795
Step 142 - Aggregate Update: 0.120549649405719
Step 143 - Aggregate Update: 0.1254040450735303
Step 144 - Aggregate Update: 0.13051120877061706
Step 145 - Aggregate Update: 0.1358957914134269
Step 146 - Aggregate Update: 0.14158408625355356
Step 147 - Aggregate Update: 0.1476057919512357
Step 148 - Aggregate Update: 0.15399807864725723
Step 149 - Aggregate Update: 0.1607957069199344
Step 150 - Aggregate Update: 0.16804220105986722
Step 151 - Aggregate Update: 0.17578300453588191
Step 152 - Aggregate Update: 0.18407315358203924
Step 153 - Aggregate Update: 0.19296776706769947
Step 154 - Aggregate Update: 0.20253603698638578
Step 155 - Aggregate Update: 0.21285957193614458
Step 156 - Aggregate Update: 0.22401944110761463
Step 157 - Aggregate Update: 0.23611925038560067
Step 158 - Aggregate Update: 0.24927003455274743
Step 159 - Aggregate Update: 0.2636044149573742
Step 160 - Aggregate Update: 0.27927722144216816
Step 161 - Aggregate Update: 0.29646177314672073
Step 162 - Aggregate Update: 0.31536822555802146
Step 163 - Aggregate Update: 0.33623307898033694
Step 164 - Aggregate Update: 0.3593453813316305
Step 165 - Aggregate Update: 0.3850352847753742
Step 166 - Aggregate Update: 0.41370334005842846
Step 167 - Aggregate Update: 0.4458221996537777
Step 168 - Aggregate Update: 0.4819587204079028
```

```
Step 169 - Aggregate Update: 0.5228027135003366
Step 170 - Aggregate Update: 0.5691814180004533
Step 171 - Aggregate Update: 0.62210368653811
Step 172 - Aggregate Update: 0.682802776895425
Step 173 - Aggregate Update: 0.7527837879545141
Step 174 - Aggregate Update: 0.8338997513778419
Step 175 - Aggregate Update: 0.928405819824786
Step 176 - Aggregate Update: 1.0390251759809632
Step 177 - Aggregate Update: 1.168943186302417
Step 178 - Aggregate Update: 1.321604705424205
Step 179 - Aggregate Update: 1.4998834550172493
Step 180 - Aggregate Update: 1.7033498839802461
Step 181 - Aggregate Update: 1.9198419528922521
Step 182 - Aggregate Update: 2.100083918350106
Step 183 - Aggregate Update: 82.4352261746559
Step 184 - Aggregate Update: 2.2427935009910707
Step 185 - Aggregate Update: 2.9567647285574488
Step 186 - Aggregate Update: 7.22153440280712
Step 187 - Aggregate Update: 13.217815051151655
Step 188 - Aggregate Update: 14.255535916344424
Step 189 - Aggregate Update: 9.176827871330097
Step 190 - Aggregate Update: 6.090840794805622
Step 191 - Aggregate Update: 4.457579181423668
Step 192 - Aggregate Update: 3.5084204679814306
Step 193 - Aggregate Update: 2.9012163870230268
Step 194 - Aggregate Update: 2.4835453918436485
Step 195 - Aggregate Update: 2.1801500005481813
Step 196 - Aggregate Update: 1.9498299356329198
Step 197 - Aggregate Update: 1.7682570238199418
Step 198 - Aggregate Update: 1.62034173860107
Step 199 - Aggregate Update: 1.4962816577806421
Step 200 - Aggregate Update: 1.389418414063357
Step 201 - Aggregate Update: 1.2952537303321776
Step 202 - Aggregate Update: 1.2105245761618875
Step 203 - Aggregate Update: 1.132822388379907
Step 204 - Aggregate Update: 1.0605025346539492
Step 205 - Aggregate Update: 0.9923584472977325
Step 206 - Aggregate Update: 0.9275110578958135
Step 207 - Aggregate Update: 0.8654315336656992
Step 208 - Aggregate Update: 0.805800105440099
Step 209 - Aggregate Update: 0.7485423919826246
Step 210 - Aggregate Update: 0.6936361923298193
Step 211 - Aggregate Update: 0.6411977102628867
Step 212 - Aggregate Update: 0.5912994487447492
Step 213 - Aggregate Update: 0.5442121955365877
Step 214 - Aggregate Update: 0.5001048181189702
Step 215 - Aggregate Update: 0.4590540972962174
Step 216 - Aggregate Update: 0.4212483390869153
Step 217 - Aggregate Update: 0.38662894213368304
Step 218 - Aggregate Update: 0.35521935889101997
Step 219 - Aggregate Update: 0.32692813858052716
Step 220 - Aggregate Update: 0.3017148628625108
Step 221 - Aggregate Update: 0.27943447704711843
Step 222 - Aggregate Update: 0.25978829836809325
Step 223 - Aggregate Update: 0.24644055485182292
Step 224 - Aggregate Update: 0.23797640097068395
```

```
Step 225 - Aggregate Update: 0.22963705498119857
Step 226 - Aggregate Update: 0.22145334208534173
Step 227 - Aggregate Update: 0.21343075052917726
Step 228 - Aggregate Update: 0.20564131279101971
Step 229 - Aggregate Update: 0.1980656236328251
Step 230 - Aggregate Update: 0.19067304011055164
Step 231 - Aggregate Update: 0.183475472166446
Step 232 - Aggregate Update: 0.17650583720481272
Step 233 - Aggregate Update: 0.1697450861190628
Step 234 - Aggregate Update: 0.16615333270678434
Step 235 - Aggregate Update: 0.16471988570913254
Step 236 - Aggregate Update: 0.16357912442761346
Step 237 - Aggregate Update: 0.16262675581027963
Step 238 - Aggregate Update: 0.16187749822749709
Step 239 - Aggregate Update: 0.16131975596169923
Step 240 - Aggregate Update: 0.1607428639849502
Step 241 - Aggregate Update: 0.16025661599560692
Step 242 - Aggregate Update: 0.15971477773865672
Step 243 - Aggregate Update: 0.15911311614418366
Step 244 - Aggregate Update: 0.15839476805594188
Step 245 - Aggregate Update: 0.15765464842434085
Step 246 - Aggregate Update: 0.15673359397402464
Step 247 - Aggregate Update: 0.15562239361027252
Step 248 - Aggregate Update: 0.15449174347953942
Step 249 - Aggregate Update: 0.15314264190851645
Step 250 - Aggregate Update: 0.1518304705609239
Step 251 - Aggregate Update: 0.1502013815398442
Step 252 - Aggregate Update: 0.14847427277062764
Step 253 - Aggregate Update: 0.14648732447578539
Step 254 - Aggregate Update: 0.14459224391150372
Step 255 - Aggregate Update: 0.14243621515875304
Step 256 - Aggregate Update: 0.14031576100182974
Step 257 - Aggregate Update: 0.13807088344520935
Step 258 - Aggregate Update: 0.1357878328943798
Step 259 - Aggregate Update: 0.13322133460101782
Step 260 - Aggregate Update: 0.1306131776723305
Step 261 - Aggregate Update: 0.12797832714562674
Step 262 - Aggregate Update: 0.12524099460034677
Step 263 - Aggregate Update: 0.12249993231162648
Step 264 - Aggregate Update: 0.11972707998549481
Step 265 - Aggregate Update: 0.1169214520102031
Step 266 - Aggregate Update: 0.11422565536510376
Step 267 - Aggregate Update: 0.11139539401932552
Step 268 - Aggregate Update: 0.10851331269540765
Step 269 - Aggregate Update: 0.10567629897053621
Step 270 - Aggregate Update: 0.10281665484393443
Step 271 - Aggregate Update: 0.09996372604419435
Step 272 - Aggregate Update: 0.09709020817199132
Step 273 - Aggregate Update: 0.09431264387040983
Step 274 - Aggregate Update: 0.09151515657649156
Step 275 - Aggregate Update: 0.08876872922959933
Step 276 - Aggregate Update: 0.0861505149567563
Step 277 - Aggregate Update: 0.08343947918335459
Step 278 - Aggregate Update: 0.08084831138313575
Step 279 - Aggregate Update: 0.07817923678199179
Step 280 - Aggregate Update: 0.07569631399778132
```

```
Step 281 - Aggregate Update: 0.07322713722507501
Step 282 - Aggregate Update: 0.07084987949201604
Step 283 - Aggregate Update: 0.06840991705085742
Step 284 - Aggregate Update: 0.06619534974129238
Step 285 - Aggregate Update: 0.06394961511847441
Step 286 - Aggregate Update: 0.061680146498531485
Step 287 - Aggregate Update: 0.059458363998146524
Step 288 - Aggregate Update: 0.05728661808144864
Step 289 - Aggregate Update: 0.055237458067111334
Step 290 - Aggregate Update: 0.05341361107089426
Step 291 - Aggregate Update: 0.051416763494543716
Step 292 - Aggregate Update: 0.04947385276935279
Step 293 - Aggregate Update: 0.04750664687563244
Step 294 - Aggregate Update: 0.04578492855860361
Step 295 - Aggregate Update: 0.044145238682631316
Step 296 - Aggregate Update: 0.04249679103442361
Step 297 - Aggregate Update: 0.04085714713188793
Step 298 - Aggregate Update: 0.039269583652327356
Step 299 - Aggregate Update: 0.03776049362586775
Step 300 - Aggregate Update: 0.03632592357831754
Step 301 - Aggregate Update: 0.034951420798606136
Step 302 - Aggregate Update: 0.03358676974352193
Step 303 - Aggregate Update: 0.03226680849606112
Step 304 - Aggregate Update: 0.030924783154532065
Step 305 - Aggregate Update: 0.02969061864479272
Step 306 - Aggregate Update: 0.028484734025482794
Step 307 - Aggregate Update: 0.0273689325062918
Step 308 - Aggregate Update: 0.02636501927470647
Step 309 - Aggregate Update: 0.025205242395230698
Step 310 - Aggregate Update: 0.024150497842057916
Step 311 - Aggregate Update: 0.023211740874032216
Step 312 - Aggregate Update: 0.02234832938503928
Step 313 - Aggregate Update: 0.021426548344052776
Step 314 - Aggregate Update: 0.0204851617346975
Step 315 - Aggregate Update: 0.01966891535776938
Step 316 - Aggregate Update: 0.01881077807824738
Step 317 - Aggregate Update: 0.01804225527030212
Step 318 - Aggregate Update: 0.01731707401272331
Step 319 - Aggregate Update: 0.016647281381125367
Step 320 - Aggregate Update: 0.016043765979517843
Step 321 - Aggregate Update: 0.015349394278359796
Step 322 - Aggregate Update: 0.014555442000556695
Step 323 - Aggregate Update: 0.01389790237010402
Step 324 - Aggregate Update: 0.01335616013837182
Step 325 - Aggregate Update: 0.012832448781210815
Step 326 - Aggregate Update: 0.012364619609016358
Step 327 - Aggregate Update: 0.01187558148646417
Step 328 - Aggregate Update: 0.011336575513013258
Step 329 - Aggregate Update: 0.010854451143998745
Step 330 - Aggregate Update: 0.010403926236193684
Step 331 - Aggregate Update: 0.010001598925782673
Step 332 - Aggregate Update: 0.009621306266836882
Step 333 - Aggregate Update: 0.009020447420965638
Step 334 - Aggregate Update: 0.008734748734866571
Step 335 - Aggregate Update: 0.008363473922983644
Step 336 - Aggregate Update: 0.007933250446782836
```

```
Step 337 - Aggregate Update: 0.007659475530260806
Step 338 - Aggregate Update: 0.007317834239870535
Step 339 - Aggregate Update: 0.006988810120091671
Step 340 - Aggregate Update: 0.006693407658033834
Step 341 - Aggregate Update: 0.00643631696975433
Step 342 - Aggregate Update: 0.006183614471851229
Step 343 - Aggregate Update: 0.005977396323402218
Step 344 - Aggregate Update: 0.0057073202862483186
Step 345 - Aggregate Update: 0.005467793238622676
Step 346 - Aggregate Update: 0.0052435980993266185
Step 347 - Aggregate Update: 0.0050025826584203514
Step 348 - Aggregate Update: 0.004756750623499784
Step 349 - Aggregate Update: 0.004527089521900507
Step 350 - Aggregate Update: 0.00434331164714763
Step 351 - Aggregate Update: 0.004171588450887498
Step 352 - Aggregate Update: 0.004004304940806946
Step 353 - Aggregate Update: 0.0038448970209969957
Step 354 - Aggregate Update: 0.003657136230643232
Step 355 - Aggregate Update: 0.0035270375019251787
Step 356 - Aggregate Update: 0.0033580080327340056
Step 357 - Aggregate Update: 0.003071230549835491
Step 358 - Aggregate Update: 0.0030135909416202367
Step 359 - Aggregate Update: 0.0028076643452550343
Step 360 - Aggregate Update: 0.0026796074697585937
Step 361 - Aggregate Update: 0.00260019349502727
Step 362 - Aggregate Update: 0.0024682877698580796
Step 363 - Aggregate Update: 0.0024272010779355924
Step 364 - Aggregate Update: 0.0023369126127477102
Step 365 - Aggregate Update: 0.0022296930441569884
Step 366 - Aggregate Update: 0.0021948305445533833
Step 367 - Aggregate Update: 0.0021462557784985847
Step 368 - Aggregate Update: 0.0019323876037873333
Step 369 - Aggregate Update: 0.0018518854173379395
Step 370 - Aggregate Update: 0.0018144340997139352
Step 371 - Aggregate Update: 0.0017505478064521185
Step 372 - Aggregate Update: 0.0016893169770497707
Step 373 - Aggregate Update: 0.0016292435359730462
Step 374 - Aggregate Update: 0.0015713261402083845
Step 375 - Aggregate Update: 0.0014207767501233093
Step 376 - Aggregate Update: 0.001383776316529861
Step 377 - Aggregate Update: 0.001372249295670258
Step 378 - Aggregate Update: 0.00125273734661846
Step 379 - Aggregate Update: 0.0012193892994361022
Step 380 - Aggregate Update: 0.0011371732607869944
Step 381 - Aggregate Update: 0.0011208547632293175
Step 382 - Aggregate Update: 0.000996565327838539
Step 383 - Aggregate Update: 0.0010162160130278153
Step 384 - Aggregate Update: 0.0009791388241779941
Step 385 - Aggregate Update: 0.0009496998811755475
Step 386 - Aggregate Update: 0.0008551186749201634
Step 387 - Aggregate Update: 0.0008438497560661062
Step 388 - Aggregate Update: 0.0008696779539008048
Step 389 - Aggregate Update: 0.0007889357093944227
Step 390 - Aggregate Update: 0.0007025476966902033
Step 391 - Aggregate Update: 0.0008003715604161243
Step 392 - Aggregate Update: 0.0006870390656104064
```

```
Step 393 - Aggregate Update: 0.0005913894264040209
Step 394 - Aggregate Update: 0.0006630264882261372
Step 395 - Aggregate Update: 0.0007926828594264634
Step 396 - Aggregate Update: 0.0006812339021990965
Step 397 - Aggregate Update: 0.000474399819268001
Step 398 - Aggregate Update: 0.0004369661622689591
Step 399 - Aggregate Update: 0.00043783791100793223
Step 400 - Aggregate Update: 0.00042561239084193403
Step 401 - Aggregate Update: 0.000404913316145894
Step 402 - Aggregate Update: 0.0003931559947716323
Step 403 - Aggregate Update: 0.0005177158561480155
Step 404 - Aggregate Update: 0.0004143594274097495
Step 405 - Aggregate Update: 0.0003537273247218309
Step 406 - Aggregate Update: 0.00033754985183520603
Step 407 - Aggregate Update: 0.0003209583855436904
Step 408 - Aggregate Update: 0.0003002846456467978
Step 409 - Aggregate Update: 0.0003836573143161104
Step 410 - Aggregate Update: 0.0003550006242960535
Step 411 - Aggregate Update: 0.00029116304209253485
Step 412 - Aggregate Update: 8.520065971850954e-05
Step 413 - Aggregate Update: 0.00019824035776139226
Step 414 - Aggregate Update: 0.00034476217152246136
Step 415 - Aggregate Update: 0.00039169330057120533
Step 416 - Aggregate Update: 0.0003148871512195228
Step 417 - Aggregate Update: 0.00024163500687279793
Step 418 - Aggregate Update: 0.0002591476055755493
Step 419 - Aggregate Update: 0.0002310457939245225
Step 420 - Aggregate Update: 0.00015521152074882139
Step 421 - Aggregate Update: 0.0003330306427500318
Step 422 - Aggregate Update: 0.00020108275120378494
Step 423 - Aggregate Update: 0.00013888257331018394
Step 424 - Aggregate Update: 0.00014473626056599187
Step 425 - Aggregate Update: 0.0001350744982460128
Step 426 - Aggregate Update: 0.00023104751889491126
Step 427 - Aggregate Update: 0.0001082706530297628
Step 428 - Aggregate Update: 0.00013934640999480052
Step 429 - Aggregate Update: 8.377996742581217e-05
Step 430 - Aggregate Update: 9.091219448009724e-05
Step 431 - Aggregate Update: 4.256970571603347e-05
Step 432 - Aggregate Update: 4.3267971520322135e-05
Step 433 - Aggregate Update: 0.00015168483052718784
Step 434 - Aggregate Update: 0.00028980732443528723
Step 435 - Aggregate Update: 0.00021590526074533045
Step 436 - Aggregate Update: 9.634463711449825e-05
Step 437 - Aggregate Update: 0.00011558038085013322
Step 438 - Aggregate Update: 7.357256978934856e-05
Step 439 - Aggregate Update: 2.594840532310627e-05
Step 440 - Aggregate Update: 0.0002553150671360527
Step 441 - Aggregate Update: 5.6155095407461886e-05
Step 442 - Aggregate Update: 9.081661400500707e-05
Step 443 - Aggregate Update: 2.2359298132101912e-05
Step 444 - Aggregate Update: 2.1853612288680324e-05
Step 445 - Aggregate Update: 0.0001433456583725956
Step 446 - Aggregate Update: 3.428384446735322e-05
Step 447 - Aggregate Update: 6.078713776958011e-05
Step 448 - Aggregate Update: 6.230696780562539e-05
```

```
Step 449 - Aggregate Update: 4.853307201813095e-05
Step 450 - Aggregate Update: 5.992255285036663e-05
Step 451 - Aggregate Update: 3.989309503538152e-05
Step 452 - Aggregate Update: 3.970632285188458e-05
Step 453 - Aggregate Update: 2.244427382791514e-05
Step 454 - Aggregate Update: 0.00010093074749306652
Step 455 - Aggregate Update: 0.00018347494128079234
Step 456 - Aggregate Update: 0.0001246067781437432
Step 457 - Aggregate Update: 3.7002809605723996e-05
Step 458 - Aggregate Update: 1.5354768066799807e-05
Step 459 - Aggregate Update: 0.00011989366228704057
Step 460 - Aggregate Update: 9.101110175890881e-05
Step 461 - Aggregate Update: 3.68429291341954e-05
Step 462 - Aggregate Update: 6.835075986089123e-05
Step 463 - Aggregate Update: 1.5670300982151275e-05
Step 464 - Aggregate Update: 1.5114835591134579e-05
Step 465 - Aggregate Update: 1.3307650263527648e-05
Step 466 - Aggregate Update: 1.0626117887824194e-05
Step 467 - Aggregate Update: 2.0744749107848293e-05
Step 468 - Aggregate Update: 1.1813903640245371e-05
Step 469 - Aggregate Update: 1.8895223590220667e-05
Step 470 - Aggregate Update: 1.6894897427732758e-05
Step 471 - Aggregate Update: 6.972957655193568e-05
Step 472 - Aggregate Update: 5.0653135659217696e-05
Step 473 - Aggregate Update: 2.916365681926436e-05
Step 474 - Aggregate Update: 6.171278916156098e-05
Step 475 - Aggregate Update: 2.627701549329231e-05
Step 476 - Aggregate Update: 6.638349038579072e-06
-- Convergence Reached --
CPU times: user 3min 26s, sys: 1min 10s, total: 4min 37s
Wall time: 42.3 s
```

Out[ ]:

Cusip	Date	Excess_returns	Returns_forecast
001084AQ	**2013-01-01**	-0.390	0.360311
	2013-02-01	0.319	4.033868
	2013-03-01	-2.567	1.519521
	2013-04-01	4.725	1.316593
	2013-05-01	2.608	-0.940849

In [ ]:
```
def total_R2(data):
    total_R2 = 1 - ((data['Excess_returns'] - data['Returns_forecast'])**2).

    return total_R2
```

In [ ]:
```
def time_series_R2(data, num_obs_per_asset):
    percentage_obs_per_asset = num_obs_per_asset / num_obs_per_asset.sum()
    asset_R2 = 1 - ((data['Excess_returns'] - data['Returns_forecast'])**2).
    time_series_R_2 = (percentage_obs_per_asset * asset_R2).sum()

    return time_series_R_2
```

In [ ]:
```
def cross_sectional_R2(data):
    time_R2 = 1 - ((data['Excess_returns'] - data['Returns_forecast'])**2).g
    cross_section_R2 = time_R2.mean()

    return cross_section_R2
```

In [ ]:
```
num_obs_per_bond = returns.groupby('Cusip')['Excess_returns'].apply(len)

percentage_obs_per_bond = num_obs_per_bond / num_obs_per_bond.sum()

percentage_obs_per_bond
```

Out[ ]:
```
Cusip
001084AQ    0.000261
00108WAD    0.000005
00108WAF    0.000005
00108WAH    0.000005
00108WAJ    0.000005
              ...
98978VAN    0.000221
98978VAP    0.000221
98978VAQ    0.000072
98978VAS    0.000131
98978VAT    0.000131
Name: Excess_returns, Length: 6492, dtype: float64
```

In [ ]:
```
in_sample_results = pd.DataFrame({'Total R^2': [total_R2(ret_ret_hat_package
                                  'Time-Series R^2': [time_series_R2(ret_ret
                                  'Cross-Sectional R^2': [cross_sectional_R2
                                  index=pd.MultiIndex.from_tuples([(False, 4

in_sample_results
```

Out[ ]:

Intercept	**Num Factors**	**Total R^2**	**Time-Series R^2**	**Cross-Sectional R^2**
False	**4**	0.491586	0.379971	0.364519

In [ ]:
```
%%time

in_sample_results_list = []

for num_factors in range(1, 6):
    for intercept in [True, False]:
        print(f'***Num Factors: {num_factors}, Intercept: {intercept}***')
        regr = ipca.InstrumentedPCA(n_factors=num_factors, intercept=interce
        regr = regr.fit(X=features_lag, y=returns.squeeze(), quiet=True)

        ret_hat_package = pd.DataFrame(regr.predict(features_lag),
                                       index=features_lag.index,
                                       columns=['Returns_forecast'])

        ret_ret_hat_package = pd.concat([returns, ret_hat_package], axis=1)
```

```
        in_sample_results = pd.DataFrame({'Total R^2': [total_R2(ret_ret_hat
                                          'Time-Series R^2': [time_series_R2(r
                                          'Cross-Sectional R^2': [cross_sectio
                                          index=pd.MultiIndex.from_tuples([(ir
        in_sample_results_list.append(in_sample_results)

in_sample_results = pd.concat(in_sample_results_list, axis=0)

in_sample_results
```

```
***Num Factors: 1, Intercept: True***
[=====                                                                       ]
7%
The panel dimensions are:
n_samples: 6492 , L: 49 , T: 160
[============================================================================]
100%
***Num Factors: 1, Intercept: False***
[=====                                                                       ]
7%
The panel dimensions are:
n_samples: 6492 , L: 49 , T: 160
[============================================================================]
100%
***Num Factors: 2, Intercept: True***
[====                                                                        ]
6%
The panel dimensions are:
n_samples: 6492 , L: 49 , T: 160
[============================================================================]
100%
***Num Factors: 2, Intercept: False***
[=====                                                                       ]
7%
The panel dimensions are:
n_samples: 6492 , L: 49 , T: 160
[============================================================================]
100%
***Num Factors: 3, Intercept: True***
[=====                                                                       ]
7%
The panel dimensions are:
n_samples: 6492 , L: 49 , T: 160
[============================================================================]
100%
***Num Factors: 3, Intercept: False***
[=====                                                                       ]
7%
The panel dimensions are:
n_samples: 6492 , L: 49 , T: 160
[============================================================================]
100%
***Num Factors: 4, Intercept: True***
```

```
[=====                                                                          ]
7%
The panel dimensions are:
n_samples: 6492 , L: 49 , T: 160
[===============================================================================]
100%
***Num Factors: 4, Intercept: False***
[=====                                                                          ]
7%
The panel dimensions are:
n_samples: 6492 , L: 49 , T: 160
[===============================================================================]
100%
***Num Factors: 5, Intercept: True***
[=====                                                                          ]
7%
The panel dimensions are:
n_samples: 6492 , L: 49 , T: 160
[===============================================================================]
100%
***Num Factors: 5, Intercept: False***
[=====                                                                          ]
7%
The panel dimensions are:
n_samples: 6492 , L: 49 , T: 160
[===============================================================================]
100%
CPU times: user 1h 35min 23s, sys: 34min 39s, total: 2h 10min 2s
Wall time: 17min 42s
```

Out[ ]:

Intercept	Num Factors	Total R^2	Time-Series R^2	Cross-Sectional R^2
True	**1**	0.491586	0.379971	0.364519
False	**1**	0.482772	0.332029	0.332460
True	**2**	0.534228	0.361602	0.388953
False	**2**	0.525692	0.307145	0.356304
True	**3**	0.562028	0.358100	0.423751
False	**3**	0.554072	0.315450	0.404086
True	**4**	0.575778	0.343096	0.435948
False	**4**	0.568736	0.308445	0.420210
True	**5**	0.582701	0.337493	0.442502
False	**5**	0.578996	0.319730	0.437703

New features

Delta D2D

In [ ]:
```
new_features = features.copy()
```

In [ ]:
```
new_features['Delta_D2D'] = new_features['Distance_to_default'] - new_featur
```

In [ ]:
```
new_features_lag = new_features.groupby('Cusip').shift().dropna()

new_features_lag
```

Out[ ]:

Cusip	Date	Bond_age	Face_value	Coupon	Duration	Spread	Ratin
001084AQ	2013-02-01	0.586301	280000000.0	5.875	6.971084	320.7451	11.
	2013-03-01	0.671233	280000000.0	5.875	6.903765	333.3739	11.
	2013-04-01	0.747945	280000000.0	5.875	6.810539	335.8488	11.
	2013-05-01	0.832877	280000000.0	5.875	6.690296	380.7369	11.
	2013-06-01	0.915068	280000000.0	5.875	6.655261	317.4350	11.
...	...	...	...	...	...	...	.
98978VAT	2023-01-01	2.501370	46700.0	3.000	16.500000	112.0000	9.
	2023-02-01	2.586301	46100.0	3.000	16.277000	113.0000	9.
	2023-03-01	2.671233	47100.0	3.000	16.586000	112.0000	9.
	2023-04-01	2.747945	45500.0	3.000	16.002000	120.0000	9.
	2023-05-01	2.832877	46700.0	3.000	16.359000	115.0000	9.

222101 rows × 50 columns

In [ ]:
```
%%time

new_features_lag = new_features.groupby('Cusip').shift().dropna()

returns_subset = returns.loc[returns.index.intersection(new_features_lag.inc

regr = ipca.InstrumentedPCA(n_factors=4, intercept=False)

regr = regr.fit(X=new_features_lag, y=returns_subset.squeeze(), quiet = True
```

```
ret_hat1_package = pd.DataFrame(regr.predict(new_features_lag),
                               index=new_features_lag.index,
                               columns=['Returns_forecast'])

ret_ret_hat1_package = pd.concat([returns_subset, ret_hat1_package], axis=1)

ret_ret_hat1_package.head()
```

```
The panel dimensions are:
n_samples: 6492 , L: 50 , T: 160
```

```
CPU times: user 59.3 s, sys: 350 ms, total: 59.6 s
Wall time: 58.3 s
```

Out[ ]:

Cusip	Date	Excess_returns	Returns_forecast
001084AQ	**2013-02-01**	0.319	4.057974
	2013-03-01	-2.567	1.523191
	2013-04-01	4.725	1.309578
	2013-05-01	2.608	-0.930350
	2013-06-01	1.255	1.648477

In [ ]:
```
num_obs_per_bond = returns_subset.groupby('Cusip')['Excess_returns'].apply(l

in_sample_results_new = pd.DataFrame({'Total R^2': [total_R2(ret_ret_hat1_pa
                                      'Time-Series R^2': [time_series_R2(ret
                                      'Cross-Sectional R^2': [cross_sectiona
                                     index=pd.MultiIndex.from_tuples([(False

in_sample_results_new
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-45-0b2c8937d4fe> in <cell line: 0>()
----> 1 num_obs_per_bond = returns_subset.groupby('Cusip')['Excess_return
s'].apply(len)
      2 
      3 in_sample_results_new = pd.DataFrame({'Total R^2': [total_R2(ret_ret
_hat1_package)],
      4                                       'Time-Series R^2': [time_serie
s_R2(ret_ret_hat1_package, num_obs_per_bond)],
      5                                       'Cross-Sectional R^2': [cross_
sectional_R2(ret_ret_hat1_package)]},

NameError: name 'returns_subset' is not defined
```

In [ ]:
```
%%time

in_sample_results_new_list = []

for num_factors in range(4, 6):
```

```
    for intercept in [True, False]:
        print(f'***Num Factors: {num_factors}, Intercept: {intercept}***')
        regr = ipca.InstrumentedPCA(n_factors=num_factors, intercept=interce
        regr = regr.fit(X=new_features_lag, y=returns_subset.squeeze(), quie

        ret_hat1_package = pd.DataFrame(regr.predict(new_features_lag),
                                 index=new_features_lag.index,
                                 columns=['Returns_forecast'])

        ret_ret_hat1_package = pd.concat([returns_subset, ret_hat1_package],

        in_sample_results = pd.DataFrame({'Total R^2': [total_R2(ret_ret_hat
                                          'Time-Series R^2': [time_series_R2(r
                                          'Cross-Sectional R^2': [cross_sectic
                                          index=pd.MultiIndex.from_tuples([(ir
        in_sample_results_new_list.append(in_sample_results)

in_sample_results =
```

```
***Num Factors: 4, Intercept: True***
The panel dimensions are:
n_samples: 6492 , L: 50 , T: 160
***Num Factors: 4, Intercept: False***
The panel dimensions are:
n_samples: 6492 , L: 50 , T: 160
***Num Factors: 5, Intercept: True***
The panel dimensions are:
n_samples: 6492 , L: 50 , T: 160
***Num Factors: 5, Intercept: False***
The panel dimensions are:
n_samples: 6492 , L: 50 , T: 160
CPU times: user 5min 47s, sys: 1.49 s, total: 5min 48s
Wall time: 5min 47s
```

In [ ]:

```
in_sample_results_new = pd.concat(in_sample_results_new_list, axis=0)

in_sample_results_new
```

Out[ ]:

Intercept	Num Factors	Total R^2	Time-Series R^2	Cross-Sectional R^2
True	**4**	0.576033	0.341014	0.436059
False	**4**	0.568978	0.307184	0.419934
True	**5**	0.583003	0.326412	0.442684
False	**5**	0.579284	0.316308	0.437517

In [ ]:

```
in_sample_results_combined = pd.concat([in_sample_results, in_sample_results
in_sample_results_combined
```

Out[ ]:

Intercept	Num Factors	Total R^2	Time-Series R^2	Cross-Sectional R^2	Total R^2	Time-Series R^2	Cross Sectiona R^2
True	**4**	0.576033	0.341014	0.436059	0.576033	0.341014	0.43605
False	**4**	0.568978	0.307184	0.419934	0.568978	0.307184	0.41993
True	**5**	0.583003	0.326412	0.442684	0.583003	0.326412	0.44268
False	**5**	0.579284	0.316308	0.437517	0.579284	0.316308	0.43751

In [ ]:

```
for i in new_features.columns:
  print(i)
```

```
Bond_age
Face_value
Coupon
Duration
Spread
Rating
Distance_to_default
Book_leverage
Market_leverage
Operating_leverage
Book_to_price
Earnings_to_price
Marketcap
Debt
Debt_to_ebitda
Spread_to_d2d
Profitability
Prof_change
Mom_6m_equity
Mom_6m
Mom_6m_rating
Mom_6m_spread
Stock_vol
Turnover_vol
VaR
VIX_Beta
Mom_6m_industry
Bond_vol
Bond_skew
Banking
Basic Industry
Telecommunications
Energy
Consumer Non-Cyclical
Leisure
Technology & Electronics
Healthcare
Consumer Goods
Transportation
Consumer Cyclical
Services
Financial Services
Insurance
Automotive
Retail
Capital Goods
Utility
Media
Real Estate
Delta_D2D
```

Spread per duration

Yield premium adjusted for bond term

```
In [ ]: new_features['Spread_per_Duration'] = new_features['Spread'] / new_features[
```

```
In [ ]: %%time

new_features_lag = new_features.groupby('Cusip').shift().dropna()

returns_subset = returns.loc[returns.index.intersection(new_features_lag.inc

regr = ipca.InstrumentedPCA(n_factors=4, intercept=False)

regr = regr.fit(X=new_features_lag, y=returns_subset.squeeze(), quiet = True

ret_hat2_package = pd.DataFrame(regr.predict(new_features_lag),
                               index=new_features_lag.index,
                               columns=['Returns_forecast'])

ret_ret_hat2_package = pd.concat([returns_subset, ret_hat2_package], axis=1)

ret_ret_hat2_package.head()
```

```
The panel dimensions are:
n_samples: 6492 , L: 51 , T: 160
```

```
CPU times: user 55.4 s, sys: 344 ms, total: 55.8 s
Wall time: 58 s
```

Out[ ]:

		Excess_returns	Returns_forecast
Cusip	**Date**		
001084AQ	**2013-02-01**	0.319	4.390151
	2013-03-01	-2.567	1.551130
	2013-04-01	4.725	1.357077
	2013-05-01	2.608	-1.035174
	2013-06-01	1.255	1.698762

```
In [ ]: in_sample_results_new = pd.DataFrame({'Total R^2': [total_R2(ret_ret_hat2_pa
                                      'Time-Series R^2': [time_series_R2(ret
                                      'Cross-Sectional R^2': [cross_sectiona
                                     index=pd.MultiIndex.from_tuples([(False

in_sample_results_new
```

Out[ ]:

		Total R^2	Time-Series R^2	Cross-Sectional R^2
Intercept	**Num Factors**			
False	**4**	0.571197	0.310839	0.420527

Apply ML

We'll try XG Boost and LSTM

XG Boost

In [ ]:
```
!pip install xgboost
```

```
Collecting xgboost
  Downloading xgboost-3.0.1-py3-none-macosx_12_0_arm64.whl.metadata (2.1 kB)
Requirement already satisfied: numpy in /opt/miniconda3/envs/py3k/lib/python
3.12/site-packages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in /opt/miniconda3/envs/py3k/lib/python
3.12/site-packages (from xgboost) (1.13.1)
Downloading xgboost-3.0.1-py3-none-macosx_12_0_arm64.whl (2.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 13.6 MB/s eta 0:00:00
Installing collected packages: xgboost
Successfully installed xgboost-3.0.1
```

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from xgboost import XGBRegressor
from itertools import product
```

In [ ]:
```
# model set up

common_index = features_lag.index.intersection(returns.index)
X = features_lag.loc[common_index].copy()
y = returns.loc[common_index]['Excess_returns'].squeeze().copy()
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,shuffle=False, test
```

In [ ]:
```
xgb_model = XGBRegressor(n_estimators=100,max_depth = 3, learning_rate=0.1,
xgb_model.fit(X_train, y_train)
y_pred = xgb_model.predict(X_test)
```

In [ ]:
```
forecast_df = pd.DataFrame(index=X_test.index)
forecast_df['Returns_forecast'] = y_pred
forecast_df['Excess_returns'] = y_test
```

In [ ]:
```
num_obs_per_bond = forecast_df.groupby('Cusip')['Excess_returns'].apply(len)

r2_total = total_R2(forecast_df)
r2_ts = time_series_R2(forecast_df, num_obs_per_bond)
r2_cs = cross_sectional_R2(forecast_df)

r2_xgb_results = pd.DataFrame({
    'Model': ['XGBoost'],
    'Total R^2': [r2_total],
    'Time-Series R^2': [r2_ts],
    'Cross-Sectional R^2': [r2_cs]
})

r2_xgb_results
```

Out[ ]:

	Model	Total R^2	Time-Series R^2	Cross-Sectional R^2
0	XGBoost	0.264821	0.274359	0.20713

Param	Value	Reason
`n_estimators`	100	Reasonable depth for small-scale ML; balances speed and performance
`max_depth`	3	Restricts model complexity; comparable to linear models like IPCA
`learning_rate`	0.1	Default for stable convergence; allows for gradual model improvements

LSTM

In [ ]:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader, TensorDataset
```

In [ ]:

```
SEQ_LEN = 5

class BondLSTMDataset(Dataset):
    def __init__(self, features_df, returns_df, seq_len=5):
        self.sequences = []
        self.targets = []
        self.cusip_tags = []
        self.dates = []

        for cusip in features_df.index.get_level_values(0).unique():
            feat = features_df.loc[cusip].values
            ret = returns_df.loc[cusip].values
            date_idx = returns_df.loc[cusip].index
            if len(feat) <= seq_len:
                continue
            for i in range(len(feat) - seq_len):
                self.sequences.append(feat[i:i+seq_len])
                self.targets.append(ret[i+seq_len])
                self.cusip_tags.append(cusip)
                self.dates.append(str(date_idx[i+seq_len]))

    def __len__(self):
        return len(self.sequences)

    def __getitem__(self, idx):
        return (
            torch.tensor(self.sequences[idx], dtype=torch.float32),
            torch.tensor(self.targets[idx], dtype=torch.float32),
            self.cusip_tags[idx],
            self.dates[idx]
        )
```

```
# Define the model
class BondLSTM(nn.Module):
    def __init__(self, input_dim, hidden_dim=32, num_layers=1):
        super(BondLSTM, self).__init__()
        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers, batch_first=T
        self.fc = nn.Linear(hidden_dim, 1)

    def forward(self, x):
        out, _ = self.lstm(x)
        out = self.fc(out[:, -1, :])
        return out.squeeze()

# Placeholders
features_lag_ready = features_lag.copy()
returns_ready = returns.copy()

# Use actual shapes
num_features_real = features_lag_ready.shape[1]

# Prepare Dataset and Loader
dataset = BondLSTMDataset(features_lag_ready, returns_ready, seq_len=SEQ_LEN
loader = DataLoader(dataset, batch_size=32, shuffle=False)

# Init model, loss, optimizer
## device = torch.device("cpu")
model = BondLSTM(input_dim=num_features_real).to(device)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

# Train loop
for epoch in range(10):
    for X_batch, y_batch, _, _ in loader:
        X_batch = X_batch.to(device)
        y_batch = y_batch.to(device)
        optimizer.zero_grad()
        preds = model(X_batch)
        loss = criterion(preds, y_batch)
        loss.backward()
        optimizer.step()

# Predict
all_preds = []
all_true = []
all_cusip = []
all_dates = []

model.eval()
with torch.no_grad():
    for X_batch, y_batch, cusip, date in loader:
        X_batch = X_batch.to(device)
        preds = model(X_batch).cpu().numpy()
        all_preds.extend(preds)
        all_true.extend(y_batch.numpy())
        all_cusip.extend(cusip)
        all_dates.extend(date)
```

```
forecast_lstm_df = pd.DataFrame({
    "Cusip": all_cusip,
    "Date": all_dates,
    "Excess_returns": all_true,
    "Returns_forecast": all_preds
})
forecast_lstm_df.set_index(["Cusip", "Date"], inplace=True)
```

In [ ]:
```
num_obs_per_bond_lstm = forecast_lstm_df.groupby('Cusip')['Excess_returns'].

r2_total_lstm = total_R2(forecast_lstm_df)
r2_ts_lstm = time_series_R2(forecast_lstm_df, num_obs_per_bond_lstm)
r2_cs_lstm = cross_sectional_R2(forecast_lstm_df)

r2_lstm_results = pd.DataFrame({
    'Model': ['LSTM'],
    'Total R^2': [r2_total_lstm],
    'Time-Series R^2': [r2_ts_lstm],
    'Cross-Sectional R^2': [r2_cs_lstm]
})
```

In [ ]:
```
r2_lstm_results
```

Out[ ]:

	Model	Total R^2	Time-Series R^2	Cross-Sectional R^2
0	LSTM	[0.020890832]	[0.014727901]	[0.07694814128260459]

hyper tuning XG boost

In [ ]:
```
common_index = features_lag.index.intersection(returns.index)
X = features_lag.loc[common_index].copy()
y = returns.loc[common_index]["Excess_returns"].squeeze().copy()
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=False, tes
```

In [ ]:
```
%%time
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [2,3,4],
    'learning_rate': [0.05, 0.1, 0.2],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'reg_lambda': [1, 5]
    }

results = []
for params in product(*param_grid.values()):
    param_dict = dict(zip(param_grid.keys(), params))
    xgb_model = XGBRegressor(**param_dict, random_state=42)
    xgb_model.fit(X_train, y_train)
    preds = xgb_model.predict(X_test)
```

```
    # Align output for R2 computation
    forecast_df = pd.DataFrame(index=X_test.index)
    forecast_df['Returns_forecast'] = preds
    forecast_df['Excess_returns'] = y_test

    num_obs_per_bond = forecast_df.groupby('Cusip')['Excess_returns'].apply(
    r2_total = total_R2(forecast_df)
    r2_ts = time_series_R2(forecast_df, num_obs_per_bond)
    r2_cs = cross_sectional_R2(forecast_df)

    results.append({
        **param_dict,
        'Total R^2': r2_total,
        'Time-Series R^2': r2_ts,
        'Cross-Sectional R^2': r2_cs
    })

results_df = pd.DataFrame(results)
results_df.sort_values(by='Total R^2', ascending=False, inplace=True)

results_df
```

```
CPU times: user 11min 2s, sys: 3min 17s, total: 14min 20s
Wall time: 2min 9s
```

Out[ ]:

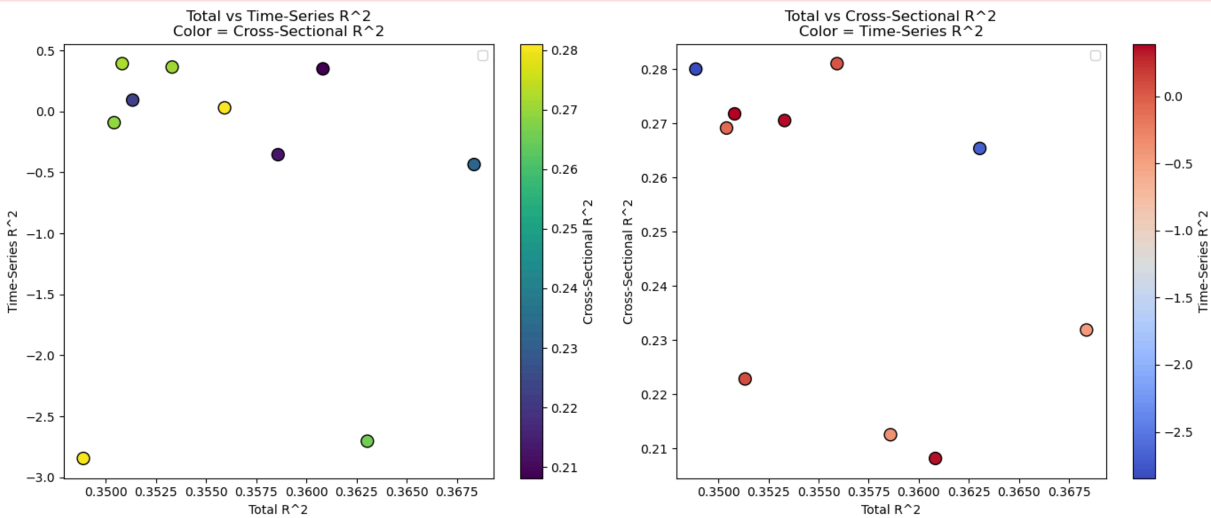
	n_estimators	max_depth	learning_rate	subsample	colsample_bytree	r
213	200	4	0.20	1.0	0.8	
211	200	4	0.20	0.8	1.0	
214	200	4	0.20	1.0	1.0	
210	200	4	0.20	0.8	1.0	
207	200	4	0.10	1.0	1.0	
...	...	...	...	...	...	
2	50	2	0.05	0.8	1.0	
1	50	2	0.05	0.8	0.8	
0	50	2	0.05	0.8	0.8	
3	50	2	0.05	0.8	1.0	
5	50	2	0.05	1.0	0.8	

216 rows × 9 columns

In [ ]:
```
results_df_top5 = results_df.head(10)

plt.figure(figsize=(14, 6))

# Total R2 vs Time-Series R2
plt.subplot(1, 2, 1)
plt.scatter(results_df_top5['Total R^2'], results_df_top5['Time-Series R^2']
```

```
plt.colorbar(label='Cross-Sectional R^2')
plt.xlabel('Total R^2')
plt.ylabel('Time-Series R^2')
plt.title('Total vs Time-Series R^2\nColor = Cross-Sectional R^2')
plt.legend()

# Total R2 vs Cross-Sectional R2
plt.subplot(1, 2, 2)
plt.scatter(results_df_top5['Total R^2'], results_df_top5['Cross-Sectional R
plt.colorbar(label='Time-Series R^2')
plt.xlabel('Total R^2')
plt.ylabel('Cross-Sectional R^2')
plt.title('Total vs Cross-Sectional R^2\nColor = Time-Series R^2')
plt.legend()

plt.tight_layout()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose labe
l start with an underscore are ignored when legend() is called with no argum
ent.
No artists with labels found to put in legend.  Note that artists whose labe
l start with an underscore are ignored when legend() is called with no argum
ent.
```



Out-of-Sample Analysis

XG Boost

In [ ]:
```
returns
```

Out[ ]:

Cusip	Date	Excess_returns
001084AQ	**2013-01-01**	-0.390
	2013-02-01	0.319
	2013-03-01	-2.567
	2013-04-01	4.725
	2013-05-01	2.608
...	**...**	...
98978VAT	**2023-01-01**	0.413
	2023-02-01	-1.077
	2023-03-01	1.050
	2023-04-01	0.904
	2023-05-01	-1.715

222102 rows × 1 columns

In [ ]:

```
def time_series_R2(df):
    grouped = df.groupby('Cusip')
    SSR = sum(((g['Excess_returns'] - g['Returns_forecast']) ** 2).sum() for
    SST = sum(((g['Excess_returns'] - g['Excess_returns'].mean()) ** 2).sum(
    return 1 - SSR / SST
```

In [ ]:

```
# Parameters
# train_window = 120
# lag = 25
# model_params = {
#     'n_estimators': 200,
#     'max_depth': 4,
#     'learning_rate': 0.10,
#     'subsample': 0.8,
#     'colsample_bytree': 0.8,
#     'reg_lambda': 1
# }



dates = features_lag.index.get_level_values('Date').drop_duplicates().sort_v
idx = pd.IndexSlice

# Parameters
min_training_window = 120
training_lag = 25
months = dates[min_training_window:]  # Start from 2015+ range depending on

# Outputs
oos_forecasts = []
```

```python
oos_results = []

for t in months:
    try:
        # Training window: [t - training_window - lag, t - lag]
        train_start = dates[max(0, dates.get_loc(t) - min_training_window)]
        train_end = dates[dates.get_loc(t) - training_lag]

        X_train = features_lag.loc[idx[:, train_start:train_end], :]
        Y_train = returns.loc[idx[:, train_start:train_end], :].squeeze()

        # Test window: t
        X_test = features_lag.loc[idx[:, t], :]
        Y_test = returns.loc[idx[:, t], :].squeeze()

        # Fit model
        model = XGBRegressor(
            n_estimators=100,  #200, reduce for speed
            max_depth=4,
            learning_rate=0.10,
            subsample=0.8,
            colsample_bytree=0.8,
            reg_lambda=1
        )
        model.fit(X_train, Y_train)

        # Forecast
        Y_pred = model.predict(X_test)
        ret_hat_df = pd.DataFrame(Y_pred, index=X_test.index, columns=["Retu

        # Benchmark forecast = mean return per bond
        benchmark = Y_train.groupby("Cusip").mean().to_frame().rename(columr
        ret_actual_df = Y_test.to_frame().rename(columns={"Excess_returns":
        merged = pd.merge(ret_actual_df, benchmark.reset_index(), on="Cusip"
        merged = pd.concat([merged, ret_hat_df], axis=1)

        # R² calculations
        total_r2 = total_R2(merged)
        r2_rel_benchmark = 1 - np.sum((merged['Excess_returns'] - merged['Re
                               np.sum((merged['Excess_returns'] - merged['Re

        merged['Window'] = min_training_window
        merged['Lag'] = training_lag
        merged['Date'] = t

        oos_result = pd.DataFrame({'OOS R^2 Over Benchmark': [r2_rel_benchma
                                  'OOS R^2 Over Zero': [total_r2]},
                                 index=pd.MultiIndex.from_tuples([(training
                                                               names=['Lag

    except Exception as e:
        print(f"Error at {t}: {e}")
        merged = pd.DataFrame()
        oos_result = pd.DataFrame({'OOS R^2 Over Benchmark': [np.nan],
                                  'OOS R^2 Over Zero': [np.nan]},
                                 index=pd.MultiIndex.from_tuples([(training
```

```
    oos_forecasts.append(merged)
    oos_results.append(oos_result)

# Combine and output
boosting_oos_forecasts_df = pd.concat(oos_forecasts)
boosting_oos_results_df = pd.concat(oos_results)
```

In [ ]:
```
boosting_oos_forecasts_df
```

Out[ ]:

		Excess_returns	Returns_benchmark_forecast	Returns_fore
Cusip	**Date**			
00209TAB	**2020-02-01**	0.614	0.888494	0.710
002824BE	**2020-02-01**	-0.451	0.476714	0.398
002824BF	**2020-02-01**	-0.224	0.640286	0.407
002824BG	**2020-02-01**	-1.934	1.113571	0.517
002824BH	**2020-02-01**	-1.944	1.253000	0.534
...	**...**	...	...	
976656CM	**2023-05-01**	NaN	NaN	-0.107
976656CN	**2023-05-01**	NaN	NaN	0.075
98388MAB	**2023-05-01**	NaN	NaN	0.123
98388MAC	**2023-05-01**	NaN	NaN	0.424
98388MAD	**2023-05-01**	NaN	NaN	0.580

73076 rows × 6 columns

In [ ]:
```
boosting_oos_results_df
```

Out[ ]:

Lag	Date	OOS R^2 Over Benchmark	OOS R^2 Over Zero
25	2020-02-01	0.077337	-0.230373
	2020-03-01	0.013492	-0.037359
	2020-04-01	0.115170	0.159813
	2020-05-01	0.234341	0.348466
	2020-06-01	0.410185	0.482070
	2020-07-01	0.372403	0.421732
	2020-08-01	-1.106807	-1.294452
	2020-09-01	-3.298929	-3.797553
	2020-10-01	0.244197	0.399614
	2020-11-01	0.151814	0.273297
	2020-12-01	0.324351	0.393666
	2021-01-01	0.032951	-0.056323
	2021-02-01	0.386399	0.480322
	2021-03-01	-0.608306	-0.668371
	2021-04-01	0.072309	0.239087
	2021-05-01	0.766582	0.430581
	2021-06-01	0.632742	0.321451
	2021-07-01	-7.658160	-30.122082
	2021-08-01	0.839046	-0.410163
	2021-09-01	0.627637	0.014860
	2021-10-01	0.539423	0.353040
	2021-11-01	0.433667	-0.158274
	2021-12-01	0.185572	0.139921
	2022-01-01	0.448198	-0.083631
	2022-02-01	-0.026778	-0.384863
	2022-03-01	0.037026	0.032167
	2022-04-01	-0.283693	-1.933078
	2022-05-01	0.457987	-1.128112
	2022-06-01	0.507724	-0.404975
	2022-07-01	0.843265	0.536115
	2022-08-01	0.840245	-0.089270
	2022-09-01	0.727653	0.227542

Lag	Date	OOS R^2 Over Benchmark	OOS R^2 Over Zero
	2022-10-01	0.772700	0.490517
	2022-11-01	0.236251	-0.186663
	2022-12-01	0.218262	-0.597833
	2023-01-01	0.750035	0.665450
	2023-02-01	-0.011742	-1.988644
	2023-03-01	0.194853	-0.786262
	2023-04-01	0.500903	-0.213524
	2023-05-01	0.349129	-0.671388

In [ ]:
```
boosting_oos_forecasts_df.to_csv('/content/drive/MyDrive/xgboost_forecasts.c
```

In [ ]:
```
dates
```

Out[ ]:
```
DatetimeIndex(['2010-02-01', '2010-03-01', '2010-04-01', '2010-05-01',
               '2010-06-01', '2010-07-01', '2010-08-01', '2010-09-01',
               '2010-10-01', '2010-11-01',
               ...
               '2022-08-01', '2022-09-01', '2022-10-01', '2022-11-01',
               '2022-12-01', '2023-01-01', '2023-02-01', '2023-03-01',
               '2023-04-01', '2023-05-01'],
              dtype='datetime64[ns]', name='Date', length=160, freq=None)
```

In [ ]:
```
# xgb_forecasts = boosting_oos_forecasts_df.copy()
# xgb_forecasts.to_csv(os.path.join(data_path, 'xgboost_forecasts.csv'), inc
```

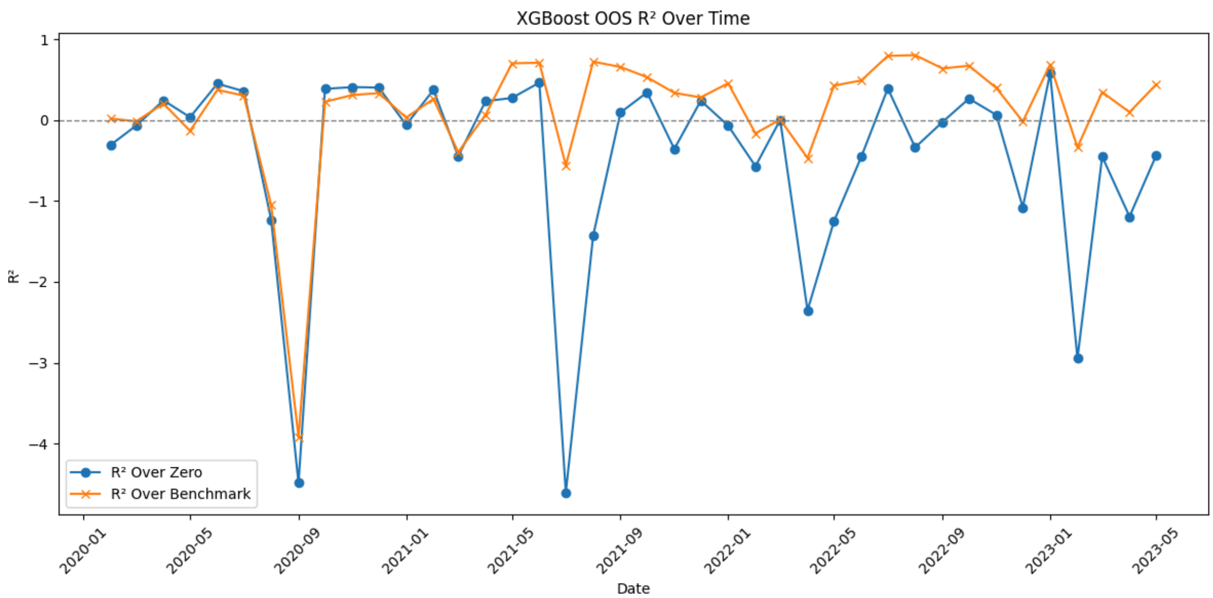
In [ ]:
```
# xgb_results = boosting_oos_results_df.reset_index()
# xgb_results.to_csv(os.path.join(data_path, 'xgboost_results.csv'), index=F
```

Visualization

In [ ]:
```
boosting_oos_forecasts_df = pd.read_csv('/content/drive/MyDrive/DSO 585 - Da
boosting_oos_results_df = pd.read_csv('/content/drive/MyDrive/DSO 585 - Data
```

In [ ]:
```
plot_df = boosting_oos_results_df.reset_index()
plot_df = plot_df.sort_values('Date')

# both R² metrics over time
plt.figure(figsize=(12, 6))
plt.plot(plot_df['Date'], plot_df['OOS R^2 Over Zero'], label='R² Over Zero'
plt.plot(plot_df['Date'], plot_df['OOS R^2 Over Benchmark'], label='R² Over 

plt.axhline(0, color='gray', linestyle='--', linewidth=1)
plt.title('XGBoost OOS R² Over Time')
plt.xlabel('Date')
```

```
plt.ylabel('R²')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



comparing IPCA results

In [ ]:
```
ipca_forecasts = pd.read_csv('/content/drive/MyDrive/DSO 585 - Data Driven C

ipca_results = pd.read_csv('/content/drive/MyDrive/DSO 585 - Data Driven Cor
```

In [ ]:
```
ipca_forecasts
```

Out[ ]:

	Cusip	Date	Excess_returns	Returns_benchmark_forecast	Retur
0	00209TAB	2020-02-01	0.614	0.888494	
1	002824BE	2020-02-01	-0.451	0.476714	
2	002824BF	2020-02-01	-0.224	0.640286	
3	002824BG	2020-02-01	-1.934	1.113571	
4	002824BH	2020-02-01	-1.944	1.253000	
...	...	...	...	...	
730755	976656CM	2023-05-01	NaN	NaN	
730756	976656CN	2023-05-01	NaN	NaN	
730757	98388MAB	2023-05-01	NaN	NaN	
730758	98388MAC	2023-05-01	NaN	NaN	
730759	98388MAD	2023-05-01	NaN	NaN	

730760 rows × 7 columns

In [ ]:

```
ipca_results
```

Out[ ]:

	Intercept	Num Factors	Date	OOS R^2 Over Benchmark	OOS R^2 Over Zero
0	True	1	2020-02-01	0.091470	-0.211527
1	True	1	2020-03-01	0.005530	-0.045731
2	True	1	2020-04-01	0.019585	0.069050
3	True	1	2020-05-01	0.016687	0.163255
4	True	1	2020-06-01	0.098822	0.208656
...	...	...	...	...	...
395	False	5	2023-01-01	0.536672	0.379887
396	False	5	2023-02-01	0.491433	-0.502286
397	False	5	2023-03-01	0.499810	-0.109700
398	False	5	2023-04-01	0.712200	0.300232
399	False	5	2023-05-01	0.512751	-0.251218

400 rows × 5 columns

In [ ]:

```
# Filter the DataFrame
ipca_subset = ipca_results[(ipca_results['Intercept'] == True) & (ipca_resul

# Compute mean R²
r2_zero = ipca_subset['OOS R^2 Over Zero'].mean()
r2_benchmark = ipca_subset['OOS R^2 Over Benchmark'].mean()

print(f"IPCA (Intercept=True, Num_Factors=5):")
print(f"  Avg R² over Zero      : {r2_zero:.4f}")
print(f"  Avg R² over Benchmark : {r2_benchmark:.4f}")
```

```
IPCA (Intercept=True, Num_Factors=5):
  Avg R² over Zero      : -51.2226
  Avg R² over Benchmark : -48.3869
```

In [ ]:

```
# Group by Intercept and Num Factors, then take mean R²
grouped_r2 = ipca_results.groupby(['Intercept', 'Num Factors'])[
    ['OOS R^2 Over Zero', 'OOS R^2 Over Benchmark']
].mean().reset_index()

# Sort by each R² to find best configs
best_zero = grouped_r2.sort_values('OOS R^2 Over Zero', ascending=False).hea
best_benchmark = grouped_r2.sort_values('OOS R^2 Over Benchmark', ascending=
```

```
print("Best by R² Over Zero:")
print(best_zero)

print("\nBest by R² Over Benchmark:")
print(best_benchmark)
```

```
Best by R² Over Zero:
   Intercept  Num Factors  OOS R^2 Over Zero  OOS R^2 Over Benchmark
5       True            1           0.076400                0.393552
7       True            3           0.075776                0.394150
8       True            4           0.072344                0.392517
3      False            4           0.058889                0.387173
4      False            5           0.058801                0.387075

Best by R² Over Benchmark:
   Intercept  Num Factors  OOS R^2 Over Zero  OOS R^2 Over Benchmark
7       True            3           0.075776                0.394150
5       True            1           0.076400                0.393552
8       True            4           0.072344                0.392517
3      False            4           0.058889                0.387173
4      False            5           0.058801                0.387075
```

Either we can use our result (True, 3) or a (False, 5) like the paper. R^2 over benchmark is close

In [ ]:
```
ipca_best = ipca_results[(ipca_results['Intercept'] == True) & (ipca_results
ipca_r2_zero = ipca_best['OOS R^2 Over Zero'].mean()
ipca_r2_benchmark = ipca_best['OOS R^2 Over Benchmark'].mean()
```

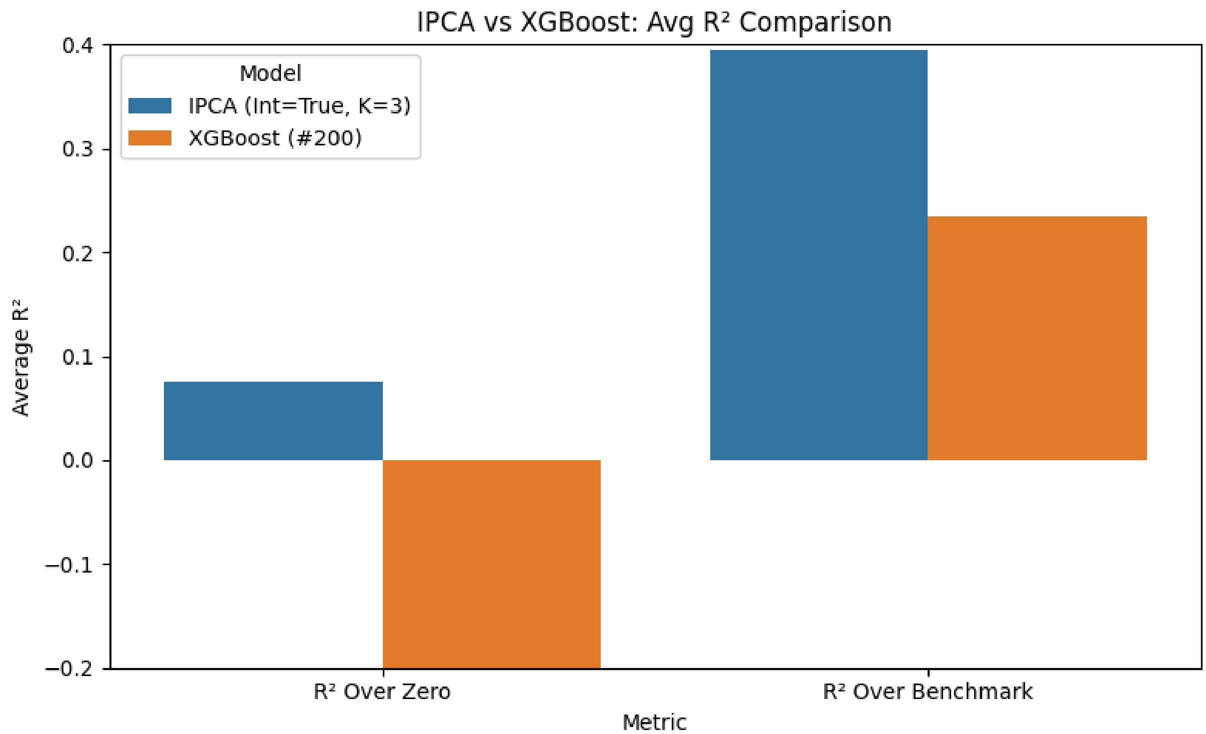
In [ ]:
```
xgb_r2_zero = boosting_oos_results_df['OOS R^2 Over Zero'].mean()
xgb_r2_benchmark = boosting_oos_results_df['OOS R^2 Over Benchmark'].mean()
```

In [ ]:
```
summary = pd.DataFrame({
    'Model': ['IPCA (Int=True, K=3)', 'XGBoost (#200)'],
    'R² Over Zero': [ipca_r2_zero, xgb_r2_zero],
    'R² Over Benchmark': [ipca_r2_benchmark, xgb_r2_benchmark]
})
print(summary)
```

```
                  Model  R² Over Zero  R² Over Benchmark
0  IPCA (Int=True, K=3)      0.075776           0.394150
1        XGBoost (#200)     -0.242071           0.234323
```

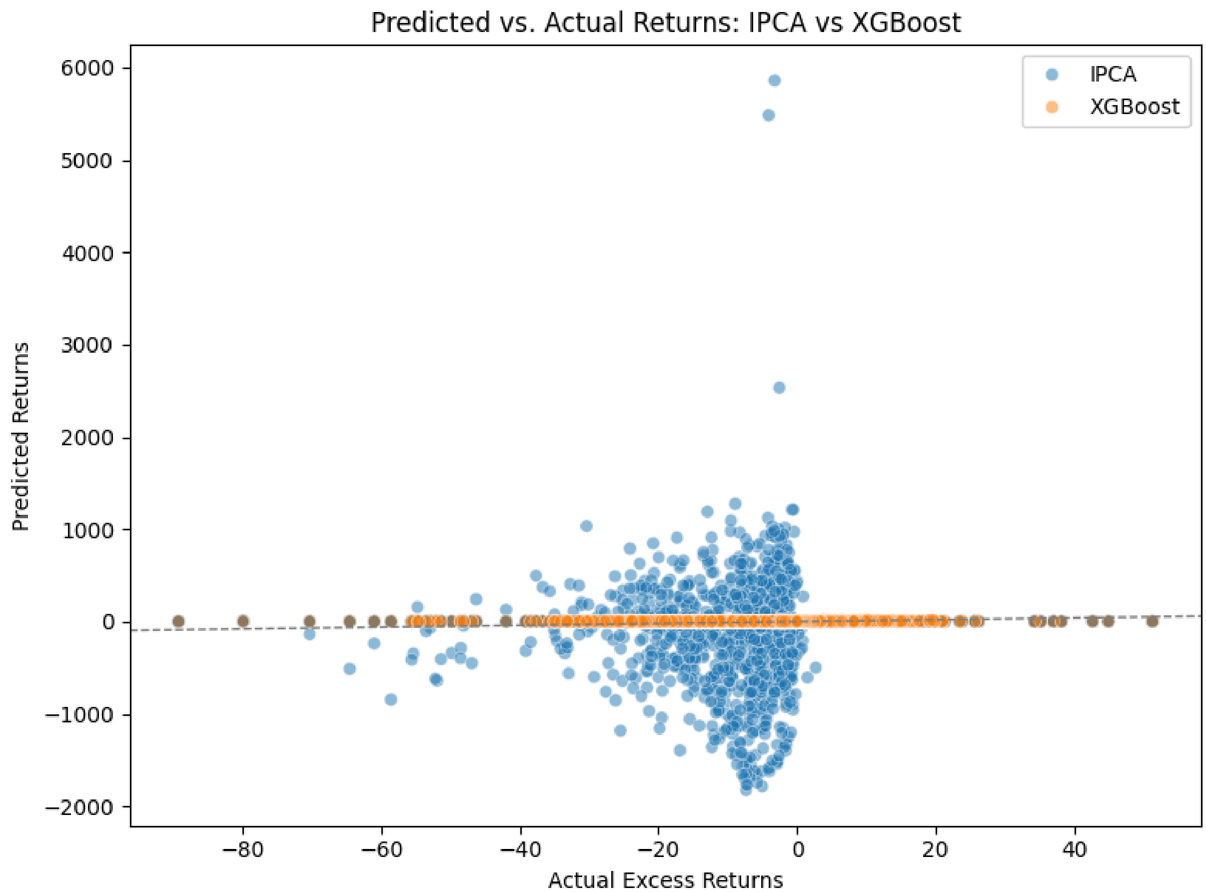
In [ ]:
```
summary_melted = summary.melt(id_vars='Model', var_name='Metric', value_name

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))
sns.barplot(data=summary_melted, x='Metric', y='R²', hue='Model')
plt.title("IPCA vs XGBoost: Avg R² Comparison")
plt.ylabel("Average R²")
plt.ylim(-0.2, 0.4)
```

```python
plt.tight_layout()
plt.show()
```



In [ ]:
```python
ipca_forecasts['Model'] = 'IPCA'
boosting_oos_forecasts_df['Model'] = 'XGBoost'

ipca_cols = ['Cusip', 'Date', 'Excess_returns', 'Returns_forecast', 'Model']
xgb_df = boosting_oos_forecasts_df.drop(columns='Date', errors='ignore').res
xgb_df = xgb_df[['Cusip', 'Date', 'Excess_returns', 'Returns_forecast']]
xgb_df['Model'] = 'XGBoost'

ipca_df = ipca_forecasts[ipca_cols].copy()


combined = pd.concat([ipca_df, xgb_df], axis=0, ignore_index=True)
```

In [ ]:
```python
plt.figure(figsize=(8, 6))
sns.scatterplot(data=combined, x='Excess_returns', y='Returns_forecast', hue
plt.axline((0, 0), slope=1, color='gray', linestyle='--', linewidth=1)
plt.xlabel('Actual Excess Returns')
plt.ylabel('Predicted Returns')
plt.title('Predicted vs. Actual Returns: IPCA vs XGBoost')
plt.legend()
plt.tight_layout()
plt.show()
```



check how many months of predictions i can analyze

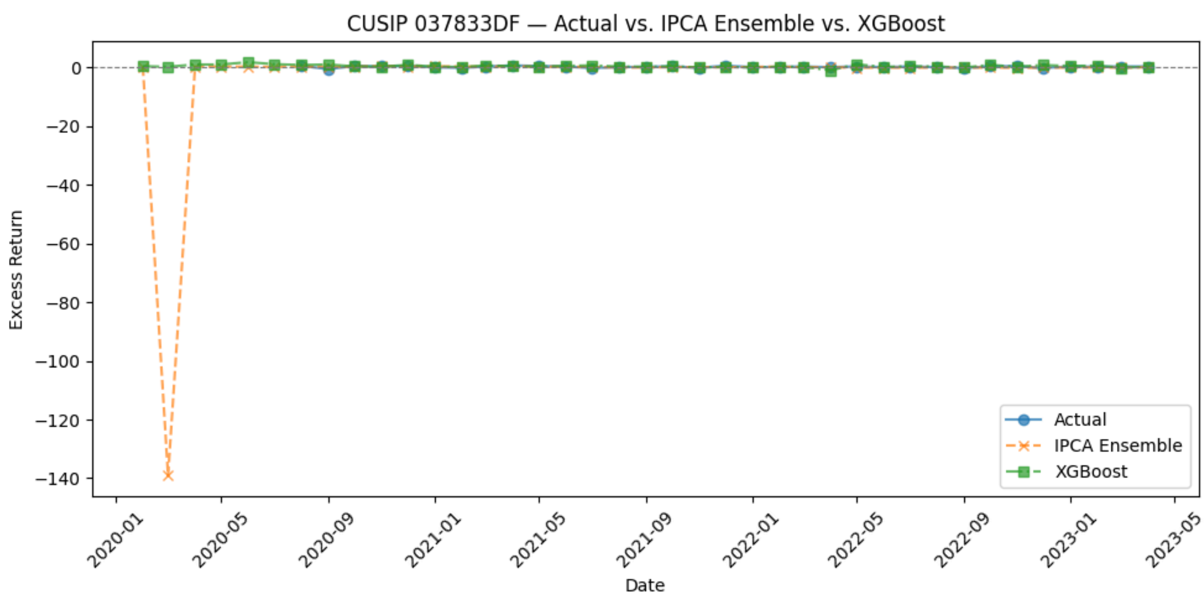
In [ ]:
```
# 1) Make sure your Date column is datetime
combined['Date'] = pd.to_datetime(combined['Date'], errors='coerce')

# 2) Drop any NaNs
combined_clean = combined.dropna(subset=['Excess_returns','Returns_forecast'

# 3) Count how many forecasts each (Model, Cusip) has
counts = combined_clean.groupby(['Model','Cusip'])['Date'].nunique().reset_i

# 4) See the top 5 bonds by number of forecasts for each model
top_ipca = counts[counts['Model']=='IPCA'].sort_values('n_forecasts', ascend
top_xgb  = counts[counts['Model']=='XGBoost'].sort_values('n_forecasts', asc

print("Top IPCA bonds by number of OOS months:\n", top_ipca)
print("\nTop XGBoost bonds by number of OOS months:\n", top_xgb)
```

```
Top IPCA bonds by number of OOS months:
     Model     Cusip  n_forecasts
722  IPCA  264399ED           40
721  IPCA  264399DK           40
729  IPCA  26441CAS           40
730  IPCA  26441CAT           40
32   IPCA  00507VAN           40

Top XGBoost bonds by number of OOS months:
         Model     Cusip  n_forecasts
2996  XGBoost  264399ED           40
2995  XGBoost  264399DK           40
3003  XGBoost  26441CAS           40
3004  XGBoost  26441CAT           40
2306  XGBoost  00507VAN           40
```

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt

# 1) Prepare IPCA ensemble
ipca = ipca_forecasts.copy()
ipca['Date'] = pd.to_datetime(ipca['Date'], errors='coerce')
ipca = ipca[ipca['Cusip']=='037833DF']
ipca_ensemble = (
    ipca
      .groupby('Date')['Returns_forecast']
      .mean()
      .reset_index(name='IPCA_ensemble')
)

# 2) Prepare XGBoost forecasts
xgb = boosting_oos_forecasts_df.copy()

# drop any existing Date column so reset_index can re-insert it safely
xgb = xgb.drop(columns='Date', errors='ignore').reset_index()

# now ensure Date is datetime
xgb['Date'] = pd.to_datetime(xgb['Date'], errors='coerce')

# filter to your bond and select only what you need
xgb = (
    xgb[xgb['Cusip']=='037833DF']
      .loc[:, ['Date','Returns_forecast']]
      .rename(columns={'Returns_forecast':'XGB_pred'})
)

# 3) Actual returns
actual = (
    ipca[['Date','Excess_returns']]
      .drop_duplicates()
      .rename(columns={'Excess_returns':'Actual'})
)

# 4) Merge all three, inner-join to keep only dates present in every series
merged = (
    actual
```

```
      .merge(ipca_ensemble, on='Date', how='inner')
      .merge(xgb,         on='Date', how='inner')
      .sort_values('Date')
)

# 5) Plot
plt.figure(figsize=(10,5))
plt.plot(merged['Date'], merged['Actual'],        'o-',  label='Actual',
plt.plot(merged['Date'], merged['IPCA_ensemble'], 'x--', label='IPCA Ensembl
plt.plot(merged['Date'], merged['XGB_pred'],      's-.', label='XGBoost',

plt.title("CUSIP 037833DF — Actual vs. IPCA Ensemble vs. XGBoost")
plt.xlabel("Date")
plt.ylabel("Excess Return")
plt.axhline(0, color='gray', linestyle='--', linewidth=0.8)
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

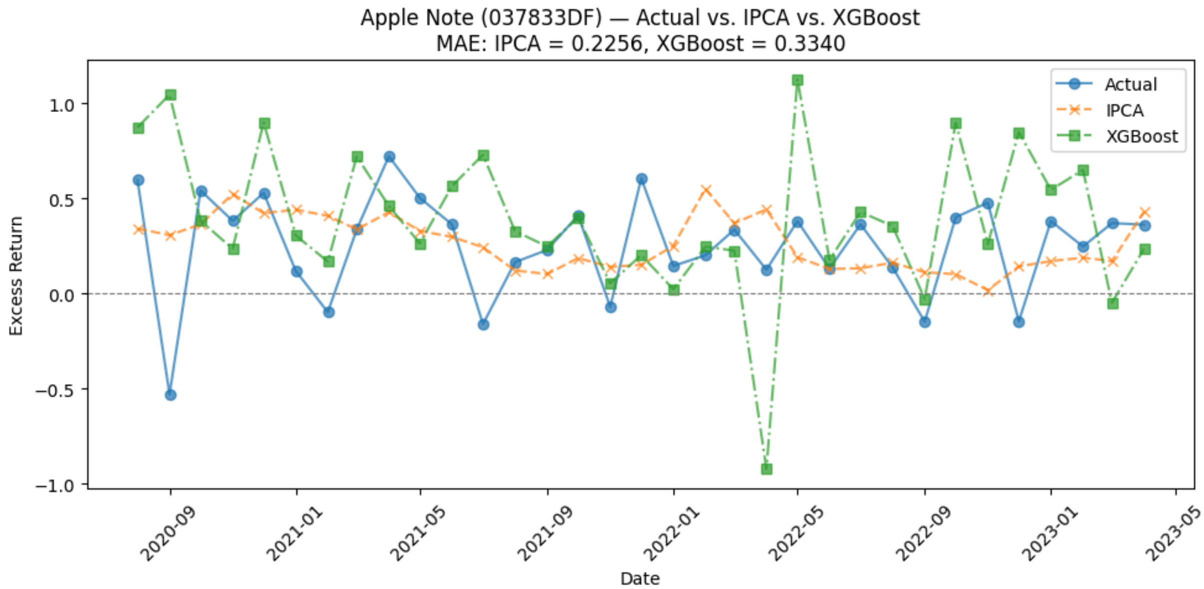


We'll use a apple Corp note for showcase

In [ ]:
```
# 1) Prepare IPCA for Apple note (037833DF)
df = ipca_forecasts.copy()
df = df.reset_index()
df['Date'] = pd.to_datetime(df['Date'], errors='coerce')
df = df[df['Cusip'] == '037833DF'].dropna(subset=['Excess_returns','Returns_

# 2) Find best IPCA config by MAE
mae_df = (
    df
    .groupby(['Intercept','Num_Factors'])
    .apply(lambda g: mean_absolute_error(g['Excess_returns'], g['Returns_for
    .reset_index(name='MAE')
)
best = mae_df.sort_values('MAE').iloc[0]
best_intercept = best['Intercept']
```

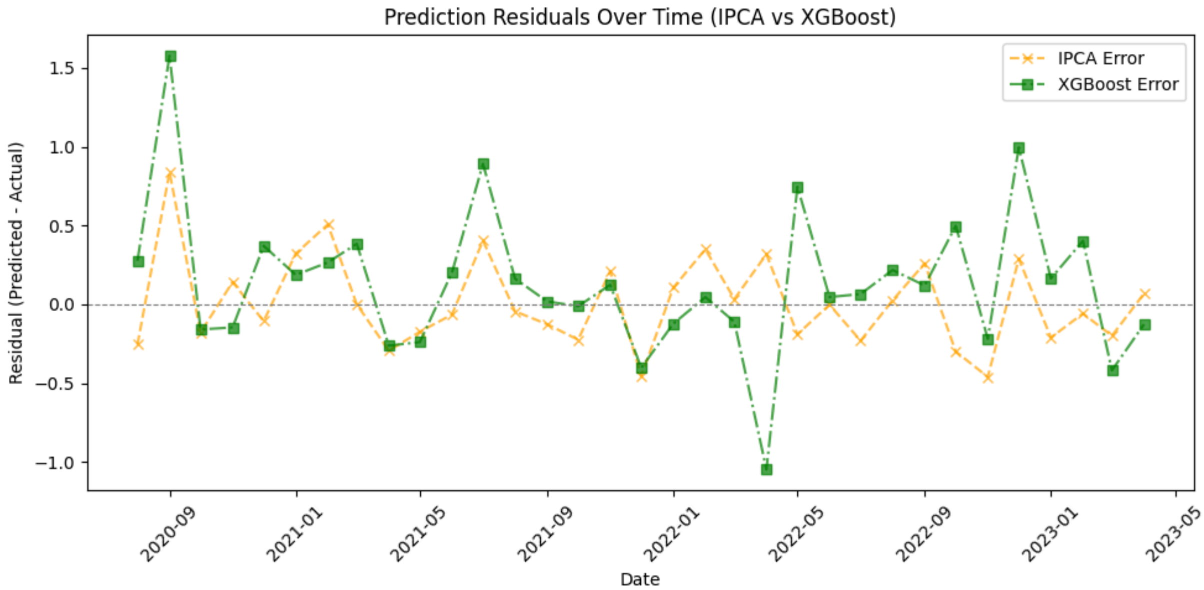
```python
best_k         = best['Num_Factors']
best_mae       = best['MAE']

# 3) Extract best IPCA series
ipca_best = df[
    (df['Intercept'] == best_intercept) &
    (df['Num_Factors'] == best_k)
].sort_values('Date')[['Date', 'Excess_returns', 'Returns_forecast']]
ipca_best = ipca_best.rename(columns={'Returns_forecast': 'IPCA_pred', 'Exce

# 4) Prepare XGBoost for the same bond
xgb = boosting_oos_forecasts_df.copy()
xgb = xgb.drop(columns=['Date'], errors='ignore').reset_index()
xgb['Date'] = pd.to_datetime(xgb['Date'], errors='coerce')
xgb = xgb[xgb['Cusip'] == '037833DF']
xgb = xgb[['Date', 'Returns_forecast']].dropna().rename(columns={'Returns_fo

# 5) Merge for plotting
merged = ipca_best.merge(xgb, on='Date', how='inner').sort_values('Date')

# 6) Compute XGBoost MAE (on same dates)
xgb_mae = mean_absolute_error(merged['Actual'], merged['XGB_pred'])

# 7) Plot
plt.figure(figsize=(10,5))
plt.plot(merged['Date'], merged['Actual'],     'o-',  label='Actual',
plt.plot(merged['Date'], merged['IPCA_pred'],  'x--', label=f'IPCA', alpha=0
plt.plot(merged['Date'], merged['XGB_pred'],   's-.', label='XGBoost',

plt.title(f"Apple Note (037833DF) — Actual vs. IPCA vs. XGBoost\n"
          f"MAE: IPCA = {best_mae:.4f}, XGBoost = {xgb_mae:.4f}")
plt.xlabel("Date")
plt.ylabel("Excess Return")
plt.axhline(0, color='gray', linestyle='--', linewidth=0.8)
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

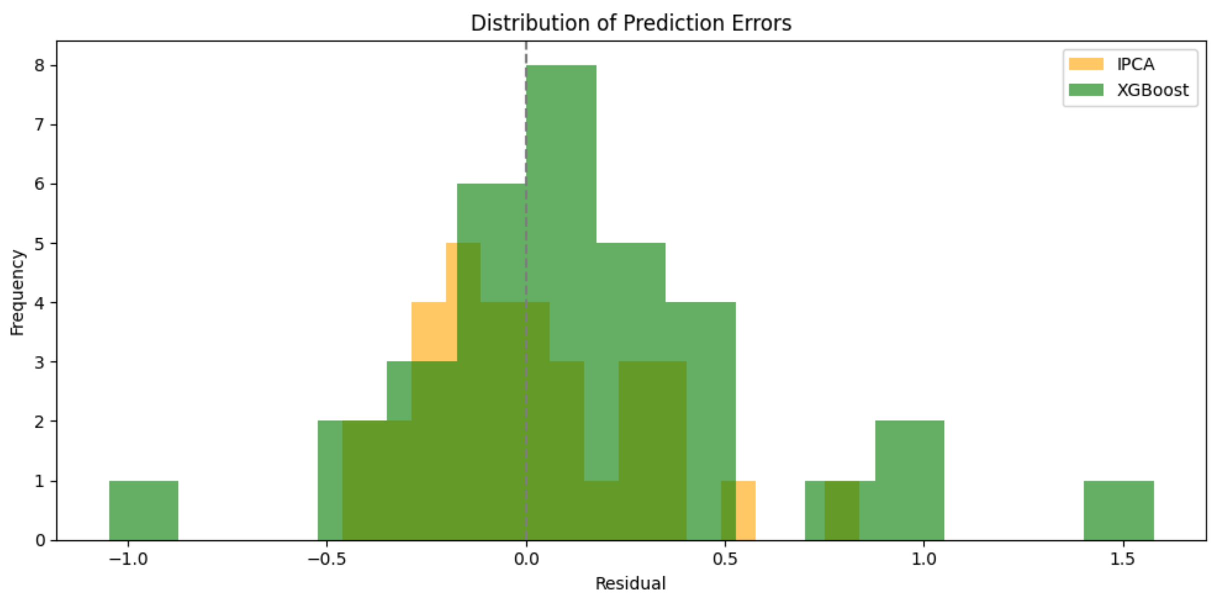
```
<ipython-input-95-14d93d800d02>:11: DeprecationWarning: DataFrameGroupBy.app
ly operated on the grouping columns. This behavior is deprecated, and in a f
uture version of pandas the grouping columns will be excluded from the opera
tion. Either pass `include_groups=False` to exclude the groupings or explici
tly select the grouping columns after groupby to silence this warning.
  .apply(lambda g: mean_absolute_error(g['Excess_returns'], g['Returns_forec
ast']))
```



In [ ]:

```
# Assume 'merged' already contains: ['Date', 'Actual', 'IPCA_pred', 'XGB_pre

# Compute residuals
merged['IPCA_error'] = merged['IPCA_pred'] - merged['Actual']
merged['XGB_error'] = merged['XGB_pred'] - merged['Actual']

# Consistent color assignment
colors = {'IPCA': 'orange', 'XGBoost': 'green'}

# Plot residuals over time
plt.figure(figsize=(10, 5))
plt.plot(merged['Date'], merged['IPCA_error'], 'x--', label='IPCA Error', co
plt.plot(merged['Date'], merged['XGB_error'], 's-.', label='XGBoost Error',
plt.axhline(0, color='gray', linestyle='--', linewidth=0.8)
plt.title('Prediction Residuals Over Time (IPCA vs XGBoost)')
plt.xlabel('Date')
plt.ylabel('Residual (Predicted - Actual)')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Plot histogram of residuals
plt.figure(figsize=(10, 5))
plt.hist(merged['IPCA_error'], bins=15, alpha=0.6, label='IPCA', color=color
plt.hist(merged['XGB_error'], bins=15, alpha=0.6, label='XGBoost', color=col
plt.axvline(0, color='gray', linestyle='--')
plt.title('Distribution of Prediction Errors')
plt.xlabel('Residual')
plt.ylabel('Frequency')
plt.legend()
plt.tight_layout()
plt.show()
```

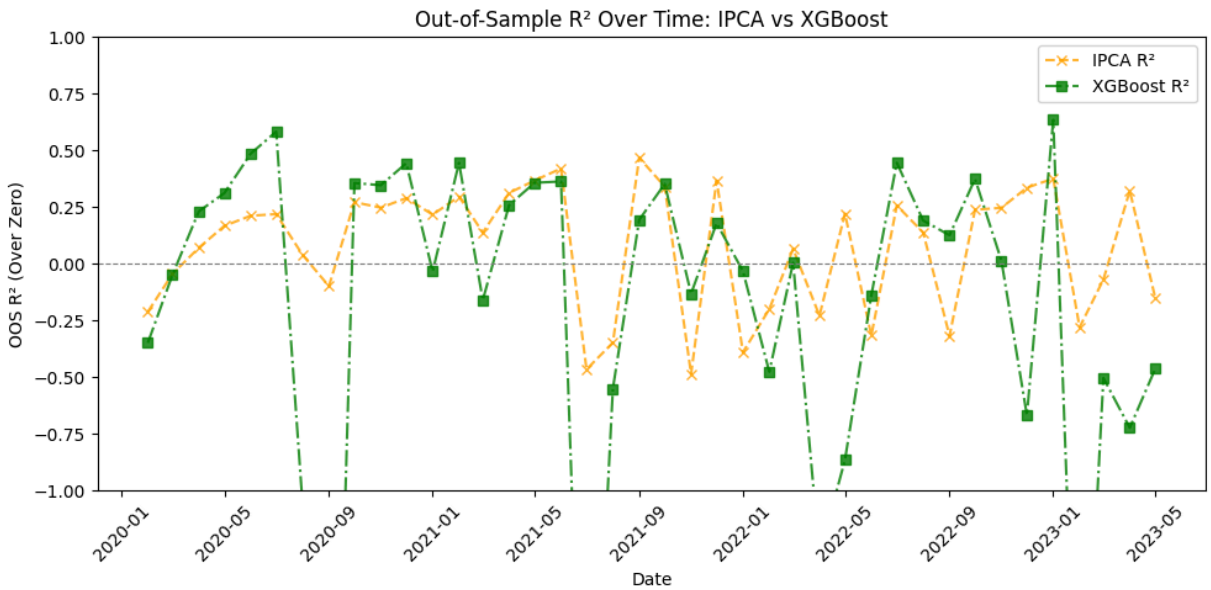




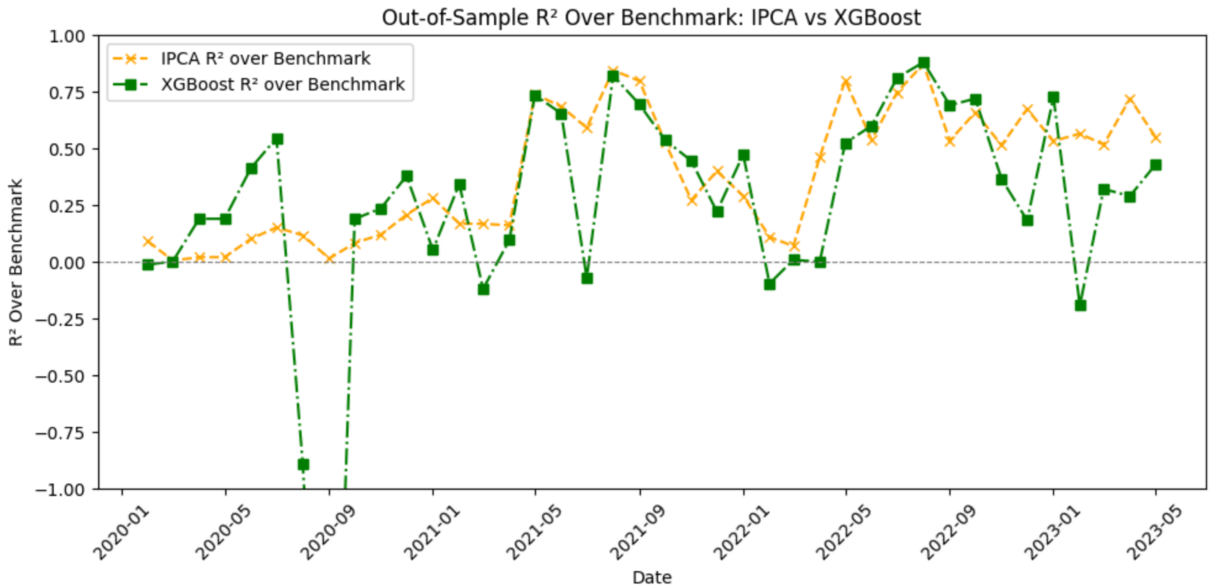
In [ ]:

```
# Prepare IPCA results
ipca_plot = ipca_results.copy()
ipca_plot['Date'] = pd.to_datetime(ipca_plot['Date'])
ipca_plot = ipca_plot[ipca_plot['Intercept'] == True]
ipca_plot = ipca_plot[ipca_plot['Num Factors'] == 3]  # or your best config

# Prepare XGBoost results
xgb_plot = boosting_oos_results_df.reset_index()
xgb_plot['Date'] = pd.to_datetime(xgb_plot['Date'])

# Merge on Date
compare_df = ipca_plot[['Date', 'OOS R^2 Over Zero']].rename(columns={'OOS R
compare_df = compare_df.merge(xgb_plot[['Date', 'OOS R^2 Over Zero']].rename

# Plot
plt.figure(figsize=(10,5))
plt.plot(compare_df['Date'], compare_df['IPCA_R2'], 'x--', label='IPCA R²',
plt.plot(compare_df['Date'], compare_df['XGB_R2'],  's-.', label='XGBoost R²
plt.axhline(0, color='gray', linestyle='--', linewidth=0.8)
```

```
plt.title('Out-of-Sample R² Over Time: IPCA vs XGBoost')
plt.xlabel('Date')
plt.ylabel('OOS R² (Over Zero)')
plt.legend()
plt.ylim(-1, 1)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [ ]:
```
# 1. Prep IPCA results: pick best config (e.g., Intercept=True, K=3)
ipca_plot = ipca_results.copy()
ipca_plot['Date'] = pd.to_datetime(ipca_plot['Date'])
ipca_plot = ipca_plot[(ipca_plot['Intercept'] == True) & (ipca_plot['Num Fac
ipca_plot = ipca_plot[['Date', 'OOS R^2 Over Benchmark']].rename(columns={'O

# 2. Prep XGBoost results
xgb_plot = boosting_oos_results_df.reset_index()
xgb_plot['Date'] = pd.to_datetime(xgb_plot['Date'])
xgb_plot = xgb_plot[['Date', 'OOS R^2 Over Benchmark']].rename(columns={'OOS

# 3. Merge on Date
compare_df = ipca_plot.merge(xgb_plot, on='Date', how='inner')

# 4. Plot R² Over Benchmark
plt.figure(figsize=(10,5))
plt.plot(compare_df['Date'], compare_df['IPCA_R2_Bench'], 'x--', label='IPCA
plt.plot(compare_df['Date'], compare_df['XGB_R2_Bench'],  's-.', label='XGBo
plt.axhline(0, color='gray', linestyle='--', linewidth=0.8)
plt.title('Out-of-Sample R² Over Benchmark: IPCA vs XGBoost')
plt.xlabel('Date')
plt.ylabel('R² Over Benchmark')
plt.ylim(-1, 1)
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Try to find what impact R-square

Fed rates

In [ ]:
```
fed_rates = pd.read_excel('/content/drive/MyDrive/DSO 585 - Data Driven Cons

fed_rates
```

```
/usr/local/lib/python3.11/dist-packages/openpyxl/styles/stylesheet.py:237: U
serWarning: Workbook contains no default style, apply openpyxl's default
  warn("Workbook contains no default style, apply openpyxl's default")
```

Out[ ]:

	Effective Date	Rate Type	Rate (%)	1st Percentile (%)	25th Percentile (%)	75th Percentile (%)	99th Percentile (%)	(%
0	12/31/2024	EFFR	4.33	4.31	4.33	4.35	4.45	
1	12/31/2024	OBFR	4.33	4.20	4.33	4.34	4.40	
2	12/30/2024	EFFR	4.33	4.31	4.33	4.33	4.35	
3	12/30/2024	OBFR	4.33	4.27	4.32	4.34	4.38	
4	12/27/2024	EFFR	4.33	4.31	4.33	4.34	4.35	
...	...	...	...	...	...	...	...	
3009	01/04/2019	OBFR	2.40	2.29	2.38	2.40	2.60	
3010	01/03/2019	EFFR	2.40	2.34	2.40	2.41	2.60	
3011	01/03/2019	OBFR	2.40	2.29	2.38	2.40	2.60	
3012	01/02/2019	EFFR	2.40	2.36	2.40	2.41	2.60	
3013	01/02/2019	OBFR	2.40	2.29	2.38	2.40	2.60	

3014 rows × 19 columns

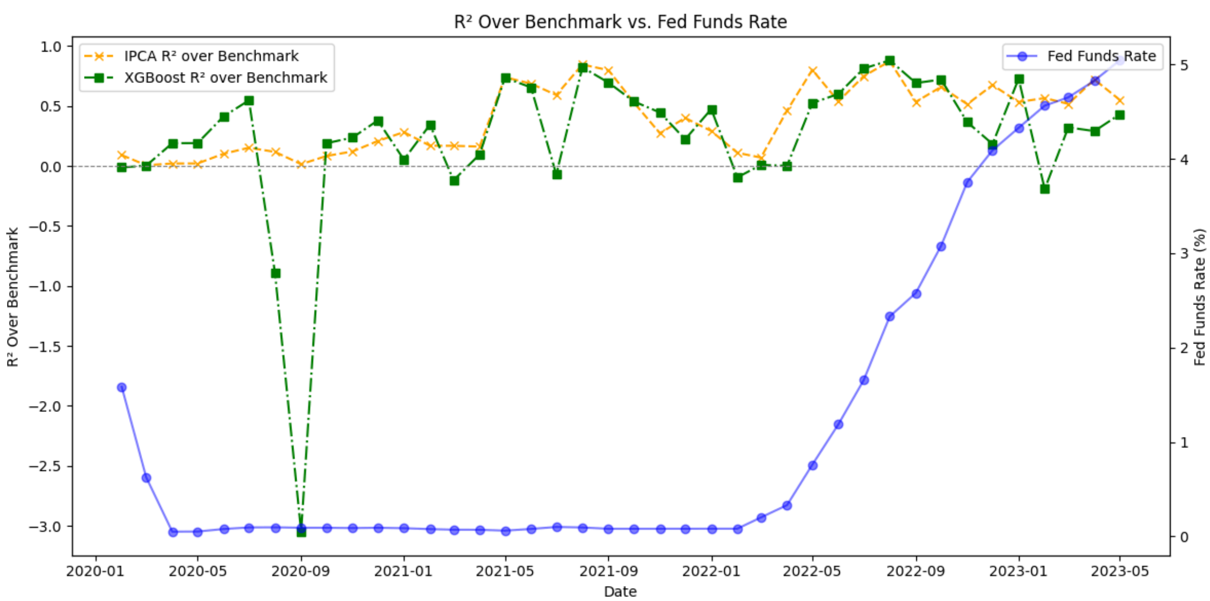
In [ ]:

```
# 1. Clean and filter Fed rates
fed_rates['Effective Date'] = pd.to_datetime(fed_rates['Effective Date'])
effr = fed_rates[fed_rates['Rate Type'] == 'EFFR'].copy()
effr = effr[['Effective Date', 'Rate (%)']].rename(columns={'Effective Date'

# 2. Convert daily rates to monthly average
effr_monthly = effr.copy()
effr_monthly['Month'] = effr_monthly['Date'].dt.to_period('M')
effr_monthly = (
    effr_monthly.groupby('Month')['FedFundsRate']
    .mean()
    .reset_index()
)
effr_monthly['Date'] = effr_monthly['Month'].dt.to_timestamp()
effr_monthly = effr_monthly.drop(columns='Month')
```

In [ ]:

```
ipca_plot = ipca_results.copy()
ipca_plot['Date'] = pd.to_datetime(ipca_plot['Date'])
ipca_plot = ipca_plot[(ipca_plot['Intercept'] == True) & (ipca_plot['Num Fac
ipca_plot = ipca_plot[['Date', 'OOS R^2 Over Benchmark']].rename(columns={'O

xgb_plot = boosting_oos_results_df.reset_index()
xgb_plot['Date'] = pd.to_datetime(xgb_plot['Date'])
xgb_plot = xgb_plot[['Date', 'OOS R^2 Over Benchmark']].rename(columns={'OOS


compare_df = ipca_plot.merge(xgb_plot, on='Date', how='inner')
compare_df = compare_df.merge(effr_monthly, on='Date', how='left')
```

```
fig, ax1 = plt.subplots(figsize=(12, 6))

# R² on left axis
ax1.plot(compare_df['Date'], compare_df['IPCA_R2_Bench'], 'x--', label='IPCA
ax1.plot(compare_df['Date'], compare_df['XGB_R2_Bench'],  's-.', label='XGBo
ax1.set_ylabel('R² Over Benchmark')
ax1.axhline(0, color='gray', linestyle='--', linewidth=0.8)
ax1.legend(loc='upper left')

# Fed rate on right axis
ax2 = ax1.twinx()
ax2.plot(compare_df['Date'], compare_df['FedFundsRate'], 'o-', label='Fed Fu
ax2.set_ylabel('Fed Funds Rate (%)')
ax2.legend(loc='upper right')

# X axis
plt.title('R² Over Benchmark vs. Fed Funds Rate')
ax1.set_xlabel('Date')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



No clear relationship

VIX

In [ ]:
```
vix = pd.read_csv('/content/drive/MyDrive/DSO 585 - Data Driven Consulting/C

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

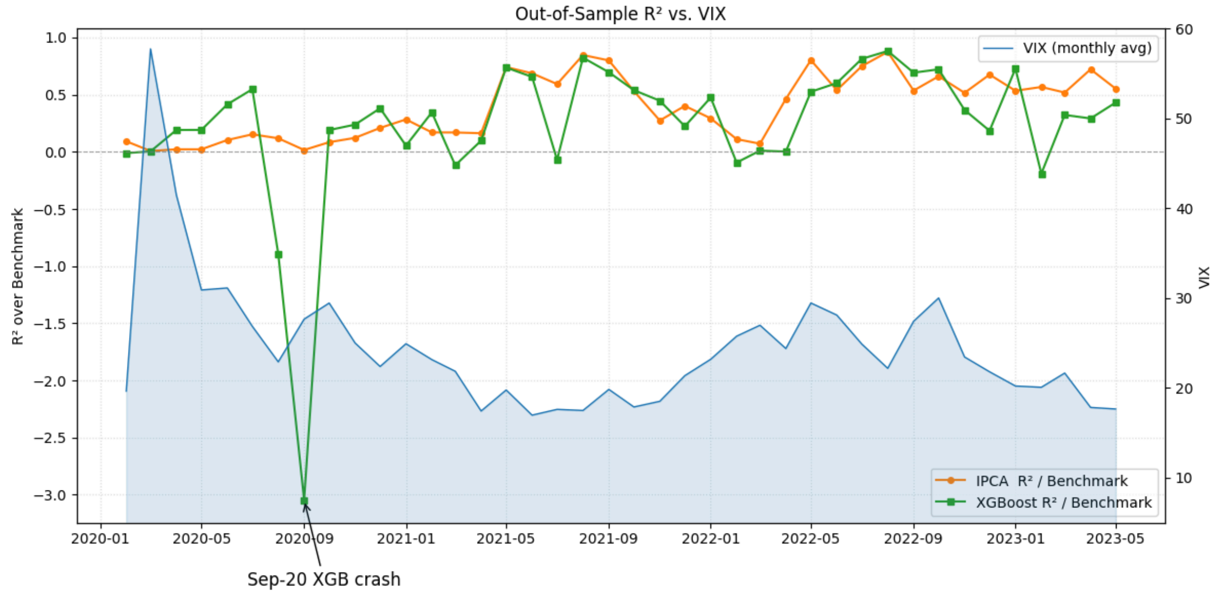
In [ ]:
```
vix_clean = vix.iloc[2:].copy().reset_index(drop=True)

vix_clean = vix_clean.rename(columns={'Price': 'Date'})

vix_clean['Date'] = pd.to_datetime(vix_clean['Date'], errors='coerce')
vix_clean['VIX_Close'] = pd.to_numeric(vix_clean['Close'], errors='coerce')

vix_clean = vix_clean[['Date', 'VIX_Close']].dropna()
```

In [ ]:
```
vix_clean['Month'] = vix_clean['Date'].dt.to_period('M')
vix_monthly = (
    vix_clean.groupby('Month')['VIX_Close']
             .mean()
             .reset_index()
)
vix_monthly['Date'] = vix_monthly['Month'].dt.to_timestamp()
vix_monthly = vix_monthly[['Date', 'VIX_Close']]
```

In [ ]:
```
compare_df = compare_df.merge(vix_monthly, on='Date', how='left')
```

In [ ]:
```
# --- R² lines (left axis) -------------------------------------------------
fig, ax1 = plt.subplots(figsize=(12, 6))

ax1.plot(compare_df['Date'],
```

```
        compare_df['IPCA_R2_Bench'],
        'o-', color='tab:orange', markersize=4, linewidth=1.5,
        label='IPCA  R² / Benchmark')

ax1.plot(compare_df['Date'],
         compare_df['XGB_R2_Bench'],
         's-', color='tab:green', markersize=4, linewidth=1.5,
         label='XGBoost R² / Benchmark')

ax1.set_ylabel('R² over Benchmark')
ax1.axhline(0, color='gray', linestyle='--', linewidth=0.8, alpha=0.7)
ax1.grid(which='major', linestyle=':', alpha=0.4)
ax1.legend(loc='lower right')

# --- annotate Sep-2020 XGB dip (if it exists) -----------------------------
sep20 = compare_df.loc[compare_df['Date'] == pd.Timestamp('2020-09-01')]
if not sep20.empty:
    y_val = sep20['XGB_R2_Bench'].values[0]
    ax1.annotate('Sep-20 XGB crash',
                 xy=(pd.Timestamp('2020-09-01'), y_val),
                 xytext=(-40, -60), textcoords='offset points',
                 arrowprops=dict(arrowstyle='->', color='black'),
                 fontsize=12)

# --- VIX line / area (right axis) ---------------------------------------
ax2 = ax1.twinx()
ax2.fill_between(compare_df['Date'],
                 compare_df['VIX_Close'],
                 color='tab:blue', alpha=0.15)
ax2.plot(compare_df['Date'],
         compare_df['VIX_Close'],
         color='tab:blue', linewidth=1,
         label='VIX (monthly avg)')
ax2.set_ylabel('VIX')
ax2.set_ylim(5, 60)
ax2.legend(loc='upper right')

plt.title('Out-of-Sample R² vs. VIX')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



This notebook was converted with convert.ploomber.io