bem114-pbj-hw3

April 26, 2025

1 Problem 1

[ ]:
```python
# Step 1: Load and clean CRSP data
import pandas as pd

# 1) Read CRSP
crsp = pd.read_csv(
    'crsp_1926_2020.csv',
    parse_dates=['date'],
    dtype={'SHRCD': 'Int64', 'EXCHCD': 'Int64'}
)

# 2) Keep only ordinary/common shares (SHRCD = 10 or 11)
crsp = crsp[crsp['SHRCD'].isin([10, 11])].copy()

# 3) Drop rows missing returns or price/outstanding
crsp = crsp.dropna(subset=['RET', 'PRC', 'SHROUT'])

# 4) Compute market equity (in millions)
crsp['ME'] = (crsp['PRC'].abs() * crsp['SHROUT']) / 1000

# 5) Create a Month index for grouping
crsp['month'] = crsp['date'].dt.to_period('M').dt.to_timestamp()

# Quick check
print(crsp[['date','month','PERMNO','RET','ME']].head())
```

```
        date      month  PERMNO        RET         ME
1 1986-01-31 1986-01-01   10000          C  16.100000
2 1986-02-28 1986-02-01   10000  -0.257143  11.960000
3 1986-03-31 1986-03-01   10000   0.365385  16.330000
4 1986-04-30 1986-04-01   10000  -0.098592  15.172000
5 1986-05-30 1986-05-01   10000  -0.222656  11.793878
```

[ ]:
```python
# Step 2: Load Best-Companies data
import pandas as pd

# 1) Read blocks
```

```python
df = pd.read_excel(
    'bcwlist.xlsx',      # or 'bcwlist.xls'
    sheet_name=0,
    header=2,            # use row 1 (0-based) as column names
    usecols="A:D,F:I"    # grab A-D (first block) and F-I (second block)
)

# 2) Rename the 8 columns to distinguish blocks
df.columns = [
    'Rank1','Company1','PERMNO1','Year1',
    'Rank2','Company2','PERMNO2','Year2'
]

# 3) Slice out each block
df1 = (
    df[['Rank1','Company1','PERMNO1','Year1']]
    .rename(columns={
      'Rank1':'Rank','Company1':'Company',
      'PERMNO1':'PERMNO','Year1':'Year'
    })
)
df2 = (
    df[['Rank2','Company2','PERMNO2','Year2']]
    .rename(columns={
      'Rank2':'Rank','Company2':'Company',
      'PERMNO2':'PERMNO','Year2':'Year'
    })
)
bcw = pd.concat([df1, df2], ignore_index=True)

# 4) Clean and type-cast
bcw = bcw.dropna(subset=['Year','PERMNO'])
bcw['Year']   = bcw['Year'].astype(int)
bcw['PERMNO'] = bcw['PERMNO'].astype(int)

# 5) Build dict
bcw_lists = bcw.groupby('Year')['PERMNO'].apply(list).to_dict()

# Inspect a few years
for y, permnos in sorted(bcw_lists.items())[:3]:
    print(f"{y}: {permnos[:5]} …")
```

```
1984: [22592, 54391, 61241, 60871, 59184] …
1993: [22592, 54391, 61241, 26470, 59184] …
1998: [22592, 54391, 20117, 75510, 26470] …
```

[ ]:

```python
# Step 3 & 4: Define portfolio formation schedule
import pandas as pd

periods = []

for year in sorted(bcw_lists.keys()):
    if year == 1984:
        # Edmans: form 4/1984, hold through 2/1993
        start = pd.Timestamp('1984-04-01')
        end   = pd.Timestamp('1993-02-01')
    elif year == 1993:
        # Edmans: form 3/1993, hold through 1/1998
        start = pd.Timestamp('1993-03-01')
        end   = pd.Timestamp('1998-01-01')
    elif year >= 1998:
        # From 1998 on: assume released Jan 1, form Feb 1, hold til Jan 1 next␣
 ↪year
        start = pd.Timestamp(f'{year}-02-01')
        # If your CRSP ends Dec 2020, the 2020 period will naturally truncate␣
 ↪at 2020-12
        end   = pd.Timestamp(f'{year+1}-01-01')
    else:
        # no list published in other years
        continue

    periods.append({
        'Year':   year,
        'start':  start,
        'end':    end,
        'permnos': bcw_lists[year]
    })

# Convert to DataFrame for inspection
periods_df = pd.DataFrame(periods)
print(periods_df[['Year','start','end']])
```

```
   Year      start        end
0  1984 1984-04-01 1993-02-01
1  1993 1993-03-01 1998-01-01
2  1998 1998-02-01 1999-01-01
3  1999 1999-02-01 2000-01-01
4  2000 2000-02-01 2001-01-01
5  2001 2001-02-01 2002-01-01
6  2002 2002-02-01 2003-01-01
7  2003 2003-02-01 2004-01-01
8  2004 2004-02-01 2005-01-01
9  2005 2005-02-01 2006-01-01
```

```
10  2006 2006-02-01 2007-01-01
11  2007 2007-02-01 2008-01-01
12  2008 2008-02-01 2009-01-01
13  2009 2009-02-01 2010-01-01
14  2010 2010-02-01 2011-01-01
15  2011 2011-02-01 2012-01-01
16  2012 2012-02-01 2013-01-01
17  2013 2013-02-01 2014-01-01
18  2014 2014-02-01 2015-01-01
19  2015 2015-02-01 2016-01-01
20  2016 2016-02-01 2017-01-01
21  2017 2017-02-01 2018-01-01
22  2018 2018-02-01 2019-01-01
23  2019 2019-02-01 2020-01-01
24  2020 2020-02-01 2021-01-01
```

[ ]:
```python
import pandas as pd
from pandas.tseries.offsets import MonthBegin

def make_portfolio_returns(
    crsp: pd.DataFrame,
    permnos: list[int],
    start: pd.Timestamp,
    end: pd.Timestamp,
    equal: bool = True
) -> pd.Series:
    """
    Build monthly returns for permnos between start and end.
    - Coerce RET & ME to float
    - IPOs: add one month after first appearance
    - Delistings: RET=0 then remove next month + rebalance
    """
    # 1) Subset
    mask = (crsp['month'] >= start) & (crsp['month'] <= end)
    sub  = crsp.loc[mask, ['month','PERMNO','RET','ME']].copy()

    # 2) Ensure numeric
    sub['RET'] = pd.to_numeric(sub['RET'], errors='coerce')
    sub['ME']  = pd.to_numeric(sub['ME'],  errors='coerce')

    # 3) Pivot using pivot_table + first to avoid lists
    rets = sub.pivot_table(
        index='month',
        columns='PERMNO',
        values='RET',
        aggfunc='first'
    )
```

```python
    mes = sub.pivot_table(
        index='month',
        columns='PERMNO',
        values='ME',
        aggfunc='first'
    )

    # 4) Record first/last appearance
    first_month = sub.groupby('PERMNO')['month'].min().to_dict()
    last_month  = sub.groupby('PERMNO')['month'].max().to_dict()

    # 5) Build month index only where we have data
    desired = pd.date_range(start=start, end=end, freq='MS')
    months  = [m for m in desired if m in rets.index]

    # 6) Initialize active & weights
    active = {p for p in permnos if first_month.get(p, pd.Timestamp.max) <= start}
    if equal:
        w = {p: 1/len(active) for p in active}
    else:
        me0 = mes.loc[months[0], list(active)]
        w = {p: me0[p]/me0.sum() for p in active}

    port_rets = []
    rebalance = False

    # 7) Loop over available months
    for i, m in enumerate(months):
        # a) full rebalance if flagged or at first month
        if i == 0 or rebalance:
            if equal:
                w = {p: 1/len(active) for p in active}
            else:
                mem = mes.loc[m, list(active)]
                w = {p: mem[p]/mem.sum() for p in active}
            rebalance = False

        # b) Get returns
        r = rets.loc[m, list(active)].fillna(0)

        # c) Portfolio return
        p_ret = sum(w[p] * r[p] for p in active)
        port_rets.append((m, p_ret))

        # d) Drift weights
        w = {p: w[p] * (1 + r[p]) for p in active}
```

```python
        total = sum(w.values())
        w = {p: w[p]/total for p in active}

        # e) Check IPOs/delistings for next month
        next_m = m + MonthBegin(1)
        ipos   = [p for p in permnos if first_month.get(p) == next_m]
        drops  = [p for p in active if last_month.get(p) == m]

        if ipos or drops:
            active = (active | set(ipos)) - set(drops)
            rebalance = True

    # 8) Return as Series
    return pd.Series(
        [ret for (_, ret) in port_rets],
        index=[m for (m, _) in port_rets]
    )
```

[ ]:
```python
# Step 6: Loop over all periods and concatenate
all_eq = []
all_vw = []

for period in periods:              # periods = list of dicts from Step 3/4
    start, end, permnos = period['start'], period['end'], period['permnos']
    eq = make_portfolio_returns(crsp, permnos, start, end, equal=True)
    vw = make_portfolio_returns(crsp, permnos, start, end, equal=False)
    all_eq.append(eq)
    all_vw.append(vw)

# Stitch into two continuous series
eq_series = pd.concat(all_eq).sort_index()
vw_series = pd.concat(all_vw).sort_index()

# Ssanity-check
print("EW head/tail:\n", eq_series.head(), eq_series.tail())
print("VW head/tail:\n", vw_series.head(), vw_series.tail())
```

```
EW head/tail:
 1984-04-01    0.001437
1984-05-01   -0.060751
1984-06-01    0.053143
1984-07-01   -0.042231
1984-08-01    0.127484
dtype: float64 2020-08-01    0.092905
2020-09-01   -0.050784
2020-10-01   -0.029439
2020-11-01    0.152465
```

```
2020-12-01    0.061441
dtype: float64
VW head/tail:
 1984-04-01    0.025485
1984-05-01   -0.047175
1984-06-01    0.015807
1984-07-01   -0.007296
1984-08-01    0.116235
dtype: float64 2020-08-01    0.117021
2020-09-01   -0.039979
2020-10-01   -0.051031
2020-11-01    0.130298
2020-12-01    0.027177
dtype: float64
```

1.1 Problem 2

1.2 Part (a)

[ ]:
```python
import pandas as pd

# Problem 2a: Portfolio statistics

stats = pd.DataFrame({
    'Average Monthly Return': [eq_series.mean(),      vw_series.mean()],
    'Volatility (Std Dev)':    [eq_series.std(),       vw_series.std()],
    'Sharpe Ratio':            [eq_series.mean()/eq_series.std(),
                                vw_series.mean()/vw_series.std()]
}, index=['Equal-weighted','Value-weighted'])

# Display
print(stats.round(4))
```

```
                Average Monthly Return  Volatility (Std Dev)  Sharpe Ratio
Equal-weighted                  0.0120                0.0534        0.2248
Value-weighted                  0.0115                0.0520        0.2216
```

1.3 Part (b)

[ ]:
```python
import pandas as pd
import statsmodels.api as sm

# 1) Load FF5 with skipfooter
ff5 = (
    pd.read_csv(
        'F-F_Research_Data_5_Factors_2x3_CSV.zip',
        compression='zip',
        skiprows=3,
```

```python
        skipfooter=3,
        engine='python'
    )
    .rename(columns={'Unnamed: 0':'Date'})
)

# 2) Keep only valid YYYYMM rows
mask = ff5['Date'].notna() & ff5['Date'].astype(str).str.match(r'^\d{6}$')
ff5 = ff5.loc[mask].copy()

# 3) Parse Date to Timestamp
ff5['Date'] = pd.to_datetime(ff5['Date'], format='%Y%m') \
               .dt.to_period('M').dt.to_timestamp()

# 4) Convert factor values from percent to decimal
for col in ['Mkt-RF','SMB','HML','RMW','CMA','RF']:
    ff5[col] = pd.to_numeric(ff5[col], errors='coerce') / 100

ff5 = ff5.set_index('Date')[['Mkt-RF','SMB','HML','RMW','CMA','RF']]

# 5) Load MOM factor
mom = (
    pd.read_csv(
        'F-F_Momentum_Factor_CSV.zip',
        compression='zip',
        skiprows=13,
        skipfooter=1,
        engine='python'
    )
    .rename(columns={'Unnamed: 0':'Date', 'Mom   ':'Mom'})
)
mask2 = mom['Date'].notna() & mom['Date'].astype(str).str.match(r'^\d{6}$')
mom = mom.loc[mask2].copy()

mom['Date'] = pd.to_datetime(mom['Date'], format='%Y%m') \
               .dt.to_period('M').dt.to_timestamp()
mom['Mom'] = pd.to_numeric(mom['Mom'], errors='coerce') / 100
mom = mom.set_index('Date')['Mom']

# 6) Combine into one DataFrame
factors = ff5.join(mom, how='inner')

# 7) Regression helper
def fit_model(Rp, exog):
    df = pd.concat([Rp.rename('Rp'), factors], axis=1).dropna()
    y  = df['Rp'] - df['RF']
    X  = sm.add_constant(df[exog])
```

```python
    return sm.OLS(y, X).fit()

specs = {
    'CAPM':    ['Mkt-RF'],
    'FF3':     ['Mkt-RF','SMB','HML'],
    'Carhart': ['Mkt-RF','SMB','HML','Mom'],
    'FF5':     ['Mkt-RF','SMB','HML','RMW','CMA']
}

# 8) Run regressions
results = []
for wtype, series in [('EW', eq_series), ('VW', vw_series)]:
    for name, exog in specs.items():
        res = fit_model(series, exog)
        results.append({
            'Portfolio': f"{wtype}-{name}",
            'Alpha (%)': res.params['const'] * 100,  # now in % per month
            't-stat':    res.tvalues['const'],
            'p-value':   res.pvalues['const']
        })

summary_df = pd.DataFrame(results).set_index('Portfolio')
print(summary_df.round(3))
```

```
            Alpha (%)  t-stat  p-value
Portfolio
EW-CAPM         0.081   0.983    0.326
EW-FF3          0.082   1.041    0.299
EW-Carhart      0.148   1.896    0.059
EW-FF5          0.072   0.894    0.372
VW-CAPM         0.087   0.850    0.396
VW-FF3          0.114   1.179    0.239
VW-Carhart      0.147   1.506    0.133
VW-FF5          0.212   2.144    0.033
```

1.4 Part (c)

[ ]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm

# 1) Recompute VW series
all_vw = []
for period in periods:
    vw = make_portfolio_returns(
        crsp,
        period['permnos'],
```

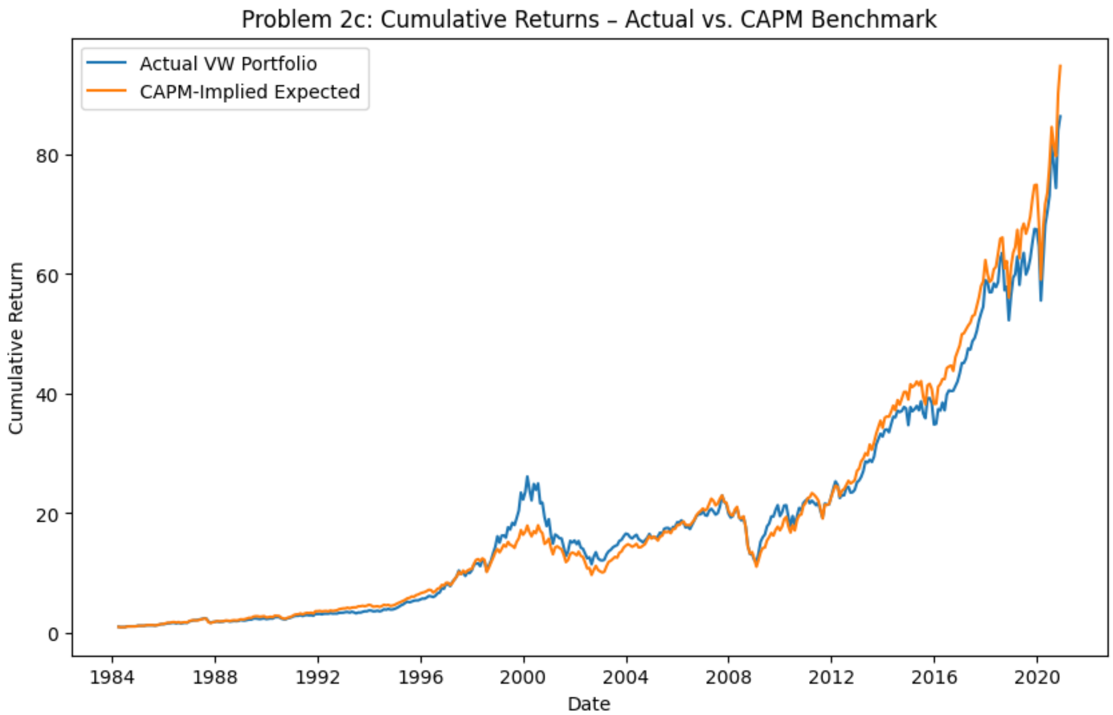
```
        period['start'],
        period['end'],
        equal=False
    )
    all_vw.append(vw)
vw_series = pd.concat(all_vw).sort_index()

# 2) Fit a CAPM on VW excess returns
df_capm = pd.concat([vw_series.rename('Rp'), factors], axis=1).dropna()
y = df_capm['Rp'] - df_capm['RF']
X = sm.add_constant(df_capm['Mkt-RF'])
capm_vw = sm.OLS(y, X).fit()
alpha, beta = capm_vw.params['const'], capm_vw.params['Mkt-RF']

# 3) Compute the CAPM-implied expected return each month
exp_excess = alpha + beta * df_capm['Mkt-RF']
exp_Rp     = exp_excess + df_capm['RF']

# 4) Build cumulative-return series
cum_actual   = (1 + df_capm['Rp']).cumprod()
cum_expected = (1 + exp_Rp).cumprod()

# 5) Plot them together
plt.figure(figsize=(10,6))
plt.plot(cum_actual.index,   cum_actual,   label='Actual VW Portfolio')
plt.plot(cum_expected.index, cum_expected, label='CAPM-Implied Expected')
plt.legend()
plt.title('Problem 2c: Cumulative Returns - Actual vs. CAPM Benchmark')
plt.xlabel('Date')
plt.ylabel('Cumulative Return')
plt.show()
```



1.5 Part (d)

[10]:
```python
import pandas as pd
import statsmodels.api as sm

# 1) Define subsamples around Jan 1, 2010
eq_pre  = eq_series[eq_series.index < '2010-01-01']
eq_post = eq_series[eq_series.index >= '2010-01-01']
vw_pre  = vw_series[vw_series.index < '2010-01-01']
vw_post = vw_series[vw_series.index >= '2010-01-01']

# 2) Regression helper for Carhart
def fit_carhart(r_ts):
    df = pd.concat([r_ts.rename('Rp'), factors], axis=1).dropna()
    y  = df['Rp'] - df['RF']
    X  = sm.add_constant(df[['Mkt-RF','SMB','HML','Mom']])
    return sm.OLS(y, X).fit()

# 3) Fit on each subsample
models = {
    'EW Pre-2010':  fit_carhart(eq_pre),
    'EW Post-2010': fit_carhart(eq_post),
    'VW Pre-2010':  fit_carhart(vw_pre),
```

```
    'VW Post-2010': fit_carhart(vw_post)
}

# 4) Summarize results
rows = []
for name, res in models.items():
    rows.append({
        'Subsample': name,
        'Alpha (%)': res.params['const'] * 100,
        't-stat':    res.tvalues['const'],
        'p-value':   res.pvalues['const']
    })

summary = pd.DataFrame(rows).set_index('Subsample').round(3)
print(summary)
```

```
              Alpha (%)  t-stat  p-value
Subsample
EW Pre-2010       0.215   2.356    0.019
EW Post-2010     -0.084  -0.550    0.583
VW Pre-2010       0.335   2.803    0.005
VW Post-2010     -0.201  -1.195    0.234
```

1.6 Part (e)

[ ]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from io import StringIO

vw_series.index = pd.to_datetime(vw_series.index).to_period('M')

# Extract the VALUE-WEIGHTED block from the CSV
with open('12_Industry_Portfolios.CSV', 'r') as f:
    lines = f.readlines()

start = next(i for i, l in enumerate(lines) if l.startswith(',NoDur'))
stop  = next(i for i, l in enumerate(lines) if 'Average Equal Weighted' in l)

vw_block = ''.join([lines[start]] + lines[start+1:stop])
industries = pd.read_csv(StringIO(vw_block), header=0, index_col=0)

# Clean & parse dates into PeriodIndex
industries.index = (
    pd.to_datetime(industries.index.astype(str), format='%Y%m')
      .to_period('M')
)
```

```python
industries = industries.replace([-99.99, -999], np.nan) / 100

# Merge with Best-Companies VW series
df = pd.concat([vw_series.rename('vw'), industries], axis=1)

print("Index type:", type(df.index))
print("Date range in merged df:", df.index.min(), "to", df.index.max())

# Drop any rows with missing data, then split at Jan 1999 ---
df = df.dropna()
split = pd.Period('1999-01', 'M')
pre  = df[df.index <  split]
post = df[df.index >= split]

print("After dropna(): Pre-1999 shape:", pre.shape, "; Post-1999 shape:", post.
 ↪shape)

# Run the 12-industry regressions, guarding against empty sets
def run_loadings(sub_df, label):
    if sub_df.empty:
        print(f"No observations for {label} - skipping regression.")
        return
    X = sm.add_constant(sub_df[industries.columns])
    y = sub_df['vw']
    res = sm.OLS(y, X).fit()
    print(f'\n=== 12-Industry Loadings ({label}) ===')
    print(res.summary().tables[1])

run_loadings(pre,  'Pre-1999')
run_loadings(post, 'Post-1999')
```

```
Index type: <class 'pandas.core.indexes.period.PeriodIndex'>
Date range in merged df: 1926-07 to 2024-12
After dropna(): Pre-1999 shape: (177, 13) ; Post-1999 shape: (264, 13)

=== 12-Industry Loadings (Pre-1999) ===
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0004      0.001      0.372      0.710      -0.002       0.003
NoDur          0.1841      0.055      3.321      0.001       0.075       0.293
Durbl          0.0154      0.037      0.420      0.675      -0.057       0.088
Manuf          0.1245      0.085      1.468      0.144      -0.043       0.292
Enrgy          0.1321      0.029      4.505      0.000       0.074       0.190
Chems          0.1527      0.058      2.618      0.010       0.038       0.268
BusEq          0.4435      0.030     14.890      0.000       0.385       0.502
Telcm          0.0823      0.036      2.308      0.022       0.012       0.153
```

Utils	-0.0331	0.042	-0.780	0.436	-0.117	0.051
Shops	-0.0098	0.047	-0.208	0.835	-0.103	0.083
Hlth	0.1509	0.039	3.914	0.000	0.075	0.227
Money	0.0597	0.042	1.411	0.160	-0.024	0.143
Other	-0.3373	0.069	-4.920	0.000	-0.473	-0.202

=== 12-Industry Loadings (Post-1999) ===

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0004	0.001	0.348	0.728	-0.002	0.003
NoDur	0.0994	0.058	1.709	0.089	-0.015	0.214
Durbl	0.0386	0.025	1.565	0.119	-0.010	0.087
Manuf	0.0842	0.059	1.426	0.155	-0.032	0.201
Enrgy	0.0284	0.024	1.201	0.231	-0.018	0.075
Chems	0.0368	0.056	0.662	0.509	-0.073	0.146
BusEq	0.5205	0.029	18.111	0.000	0.464	0.577
Telcm	0.0200	0.036	0.559	0.577	-0.050	0.090
Utils	-0.1596	0.035	-4.524	0.000	-0.229	-0.090
Shops	0.0065	0.047	0.138	0.890	-0.086	0.099
Hlth	0.1147	0.038	3.018	0.003	0.040	0.190
Money	0.0980	0.039	2.529	0.012	0.022	0.174
Other	-0.0119	0.071	-0.168	0.867	-0.151	0.128

1.7 Problem 3

1.8 Part (a)

We found the beta of this strategy to be 1.05699. Our portfolio moves slightly more than the market (5% more). GThe strategy invests mostly in large companies that are already part of major market indices. Due to this, the portfolio inherits market wide risk, causing the beta to be near 1. Shorting the overall market against the long position in the list would mostly remove market exposure (beta 0) and isolate the alpha of the strategy, This is very attractuve to institutional investors as it produces alpha while limiting downside risk as they are heavily levered. This is less attractuve to retail investors who are less-levered (if at all), and likely would prefer simple long-only strategies.

1.9 Part (b)

These result show that financial markets do not fully price the value of employee satisfaction since the results of the strategy show statistically significant alpha even after controlling for strandard risk factors (size, value, etc.)/ This suggetss that ocmpanies wiht high employee satisfaction ratings consistently outperform what traditional assset pricing models would predict. Thus, the financial markets do not fully incorporate the value of employee satisfaction into the stock prices.

1.10 Part (c)

We think that the strategy's edge simply got crowded out over time. After Edmans's 2011 study hit the press, investors rushed in to buy the happiest companies, bidding up their prices until any bargain disappeared. New data tools also made tracking employee sentiment almost real time, and firms themselves improved their HR practices, so there's less of a gap to exploit. In short, once everyone knew that happy workplaces could outperform, that information stopped being a secret—and the extra returns evaporated.

1.11 Part (d)

BCW surveys employees and produces information about employee satisfaction. In a sense, this is a somewhat antiquated way of producing information, and it is provided with a considerable time lag. We live in a world where each of us leaves a digital footprint, and the company Bombora purchases and aggregates cookie data from large media companies, and through email data stored in cookies they identify unique company employees. They can track what employees are reading about in real time. How might the employee satisfaction strategy be improved for the modern world?