# FINM 35000 Problem Set 3: Equity Valuation Stress Testing

Aman Krishna

Tim Taylor

Yazmin Ramirez Delgado

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import math as m
import scipy.stats as stats
import datetime as dt
from statsmodels.regression.rolling import RollingOLS
import seaborn as sns
import warnings
from scipy.stats import norm
pd.set_option("display.precision", 2)
pd.set_option('display.float_format', '{:.3f}'.format)
warnings.filterwarnings("ignore")
```

```
c:\Users\Aman\anaconda3\lib\site-packages\numpy\_distributor_init.py:30: UserWarnin
g: loaded more than 1 DLL from .libs:
c:\Users\Aman\anaconda3\lib\site-packages\numpy\.libs\libopenblas.EL2C6PLE4ZYW3ECEVI
V3OXXGRN2NRFM2.gfortran-win_amd64.dll
c:\Users\Aman\anaconda3\lib\site-packages\numpy\.libs\libopenblas.FB5AE2TYXYH2IJRDKG
DGQ3XBKLKTF43H.gfortran-win_amd64.dll
c:\Users\Aman\anaconda3\lib\site-packages\numpy\.libs\libopenblas64__v0.3.21-gcc_10_
3_0.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
<frozen importlib._bootstrap>:228: RuntimeWarning: scipy._lib.messagestream.MessageS
tream size changed, may indicate binary incompatibility. Expected 56 from C header,
got 64 from PyObject
```

# 1. Replication of Cosemans and Frehen (2021) (100 points)

Note: for questions 2-3, it is possible you will not obtain the exact numbers in the paper, which is okay as long as you are able to describe the ways in which you might have deviated from the authors (in question 4).

## 1.

In your own words, describe what the authors mean by “salience theory” and how it affects investor’s portfolio choice decisions.

"Salience theory," as discussed the paper, refers to the idea that investors tend to give disproportionate attention and importance to the most prominent or striking features of an investment, particularly past returns. This theory is grounded in the broader understanding of how cognitive biases (something like behavioural economics) influence decision-making. In the context of stock market investments, salience theory suggests that investors are drawn to stocks that have had notably high or low returns in the past, as these returns are more "salient" or noticeable. Famous stocks like Apple and Tesla come to mind when thinking about this theory from a US Stock market perspective.

According to salience theory, investors do not evaluate potential investments in a completely rational or comprehensive manner. That is, investors are unsophisticated in their decision making and, they are more likely to focus on the most memorable or striking aspects of an asset's history, especially its past performance. For example, if a stock has experienced a significant upsurge in value in the recent past, this positive performance becomes a salient feature that attracts investors, leading them to overvalue such stocks. This overvaluation, in turn, means that these stocks are likely to have lower future returns because their current prices are inflated due to high demand based on salient past performance.

Going the other way, stocks with notably poor past performance can become undervalued, as investors overlook them due to their salient negative returns. These undervalued stocks, according to the theory, are likely to yield higher future returns as their current lower prices do not reflect their potential value.

Overall, we saw a lot of similarity between Fama French's fourth factor (out of 5) - Profitability (RMW - Robust Minus Weak). This factor captures the historical outperformance of profitable companies compared to less profitable ones. It measures the return difference between a portfolio of companies with high profitability and a portfolio of companies with low profitability.

## 2.

Following Section 3 of the paper, download the relevant variables from CRSP and Compustat (both available through WRDS). Use this data to replicated Table 2.

Load CRSP Daily, Monthly, and Compustat Fundamentals Data

In [ ]:
```
# Read "C:\Users\Aman\Downloads\Compressed\crsp_us_equity.csv"
```

```
crsp_daily = pd.read_csv("C:/Users/Aman/Downloads/Compressed/crsp_us_equity.csv")
```

In [ ]:
```
# Convert date to datetime format
crsp_daily['date'] = pd.to_datetime(crsp_daily['date'])

# Sort the DataFrame by 'TICKER' and 'date'
crsp_daily = crsp_daily.sort_values(['TICKER', 'date'])

# Remove all rows with missing TICKER or RET
crsp_daily.dropna(subset=['TICKER', 'RET'], inplace=True)
```

In [ ]:
```
# Create a month and year column like 2005-01
crsp_daily['month'] = crsp_daily['date'].dt.strftime('%Y-%m')
```

In [ ]:
```
# Use groupby and transform to calculate the number of days in each month
crsp_daily['days_in_month'] = crsp_daily.groupby(['TICKER', 'month'])['RET'].transf

# # Set data and ticker as index
# crsp_daily = crsp_daily.set_index(['date', 'TICKER'])

# # Remove all dates before 2000-01-01
# crsp_daily = crsp_daily[crsp_daily['date'] >= '2005-01-01']
```

In [ ]:
```
crsp_daily
```

Out[ ]:

| | PERMNO | date | TICKER | PRC | VOL | RET | month | days_in_montl |
|---|---|---|---|---|---|---|---|---|
| **1709955** | 10495 | 1962-07-02 | A | 41.125 | 2600.000 | 0.021739 | 1962-07 | 2 |
| **1709956** | 10495 | 1962-07-03 | A | 41.375 | 2100.000 | 0.006079 | 1962-07 | 2 |
| **1709957** | 10495 | 1962-07-05 | A | 41.250 | 3600.000 | -0.003021 | 1962-07 | 2 |
| **1709958** | 10495 | 1962-07-06 | A | 40.500 | 2600.000 | -0.018182 | 1962-07 | 2 |
| **1709959** | 10495 | 1962-07-09 | A | 40.750 | 4000.000 | 0.006173 | 1962-07 | 2 |
| **...** | ... | ... | ... | ... | ... | ... | ... | .. |
| **85532128** | 91205 | 2013-03-11 | ZZ | 2.200 | 407000.000 | 0.000000 | 2013-03 | 1 |
| **85532129** | 91205 | 2013-03-12 | ZZ | 2.210 | 159900.000 | 0.004545 | 2013-03 | 1 |
| **85532130** | 91205 | 2013-03-13 | ZZ | 2.210 | 308900.000 | 0.000000 | 2013-03 | 1 |
| **85532131** | 91205 | 2013-03-14 | ZZ | 2.210 | 274900.000 | 0.000000 | 2013-03 | 1 |
| **85532132** | 91205 | 2013-03-15 | ZZ | 2.190 | 1518100.000 | -0.009050 | 2013-03 | 1 |

78114075 rows × 8 columns

In [ ]:

```
# Read "C:\Users\Aman\Downloads\Compressed\crsp_us_equity_monthly.csv"
crsp_monthly = pd.read_csv("C:/Users/Aman/Downloads/Compressed/crsp_us_equity_month
```

In [ ]:

```
# Convert 'date' column to datetime
crsp_monthly['date'] = pd.to_datetime(crsp_monthly['date'])

# Sort the DataFrame by 'TICKER' and 'date' columns
crsp_monthly.sort_values(by=['TICKER', 'date'], inplace=True)

# Remove all rows with missing TICKER
crsp_monthly.dropna(subset=['TICKER'], inplace=True)

# Convert negative PRC values to positive
crsp_monthly['PRC'] = crsp_monthly['PRC'].abs()

# Fill missing PRC values with 0
crsp_monthly['PRC'].fillna(0, inplace=True)

# Shift the indexes by 1 for crsp_monthly so that the PRC, VOL and RET values are f
crsp_monthly['PRC'] = crsp_monthly.groupby(['TICKER'])['PRC'].shift(1)
```

```
# # Remove all dates before 2000-01-01
# crsp_monthly = crsp_monthly[crsp_monthly['date'] >= '2005-01-01']

# Backfill the missing PRC values with next available PRC value
crsp_monthly['PRC'].fillna(method='bfill', inplace=True)

#Drop COMNAM and PERMNO columns
crsp_monthly.drop(columns=['COMNAM'], inplace=True)

# # Set data and ticker as index
# crsp_monthly = crsp_monthly.set_index(['date', 'TICKER'])
```

In [ ]:
```
crsp_monthly
```

Out[ ]:

| | PERMNO | date | TICKER | PRC | VOL | RET |
|---|---|---|---|---|---|---|
| **80206** | 10495 | 1962-07-31 | A | 40.375 | 852.000 | 0.003106 |
| **80207** | 10495 | 1962-08-31 | A | 40.375 | 967.000 | 0.024768 |
| **80208** | 10495 | 1962-09-28 | A | 40.875 | 1525.000 | -0.094801 |
| **80209** | 10495 | 1962-10-31 | A | 37.000 | 1396.000 | 0.033784 |
| **80210** | 10495 | 1962-11-30 | A | 38.250 | 1895.000 | 0.117647 |
| **...** | ... | ... | ... | ... | ... | ... |
| **4083922** | 91205 | 2012-11-30 | ZZ | 2.230 | 111189.000 | -0.026906 |
| **4083923** | 91205 | 2012-12-31 | ZZ | 2.170 | 116706.000 | 0.000000 |
| **4083924** | 91205 | 2013-01-31 | ZZ | 2.170 | 71494.000 | -0.004608 |
| **4083925** | 91205 | 2013-02-28 | ZZ | 2.160 | 97674.000 | 0.009259 |
| **4083926** | 91205 | 2013-03-28 | ZZ | 2.180 | 136909.000 | NaN |

3764041 rows × 6 columns

In [ ]:
```
# Merge crsp_daily and crsp_monthly on TICKER, date, PERMNO
crsp = pd.merge(crsp_daily, crsp_monthly, on=['TICKER','PERMNO', 'date'], how='oute
```

In [ ]:
```
del crsp_daily, crsp_monthly
```

In [ ]:
```
crsp
```

Out[ ]:

| | PERMNO | date | TICKER | PRC_x | VOL_x | RET_x | month | days_in_month |
|---|---|---|---|---|---|---|---|---|
| **0** | 10495 | 1962-07-02 | A | 41.125 | 2600.000 | 0.021739 | 1962-07 | 21.000 |
| **1** | 10495 | 1962-07-03 | A | 41.375 | 2100.000 | 0.006079 | 1962-07 | 21.000 |
| **2** | 10495 | 1962-07-05 | A | 41.250 | 3600.000 | -0.003021 | 1962-07 | 21.000 |
| **3** | 10495 | 1962-07-06 | A | 40.500 | 2600.000 | -0.018182 | 1962-07 | 21.000 |
| **4** | 10495 | 1962-07-09 | A | 40.750 | 4000.000 | 0.006173 | 1962-07 | 21.000 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **78156370** | 84188 | 1986-11-28 | ZTXQ | NaN | NaN | NaN | NaN | NaN |
| **78156371** | 14327 | 2015-10-30 | ZU | NaN | NaN | NaN | NaN | NaN |
| **78156372** | 91435 | 2009-02-27 | ZVUE | NaN | NaN | NaN | NaN | NaN |
| **78156373** | 85520 | 2007-03-30 | ZVXI | NaN | NaN | NaN | NaN | NaN |
| **78156374** | 91205 | 2013-03-28 | ZZ | NaN | NaN | NaN | NaN | NaN |

78156375 rows × 11 columns

# NOTE FOR USER - ALL _X VARIABLES ARE FROM CRSP DAILY AND ALL _Y VARIABLES ARE FROM CRSP MONTHLY

### Only taking >$5 prev month price and >15 days returns in a month

In [ ]:

```
# group by TICKER and backfill the PRC values
crsp['PRC_y_daily'] = crsp.groupby(['TICKER'])['PRC_y'].fillna(method='bfill')
# Filter dataframe where PRC_x is >= 5 and days_in_month >15
crsp = crsp[(crsp['PRC_y_daily'] >= 5) & (crsp['days_in_month'] > 15)]
```

In [ ]:

```
# Convert all RET_x to float, if not possible, convert to NaN
crsp['RET_x'] = pd.to_numeric(crsp['RET_x'], errors='coerce')
#Drop all NaN values in RET_x
crsp.dropna(subset=['RET_x'], inplace=True)
```

## Loading the CRSP Index Data

In [ ]:
```
crsp_index = pd.read_csv("C:/Users/Aman/Downloads/Compressed/crsp_index.csv")
```

In [ ]:
```
crsp_index['caldt'] = pd.to_datetime(crsp_index['caldt'])
crsp_index.rename(columns={'caldt':'date'}, inplace=True)
```

In [ ]:
```
crsp_index
```

Out[ ]:

| | date | ewretd |
|---|---|---|
| **0** | 1926-01-02 | 0.010 |
| **1** | 1926-01-04 | 0.006 |
| **2** | 1926-01-05 | -0.002 |
| **3** | 1926-01-06 | 0.001 |
| **4** | 1926-01-07 | 0.008 |
| **...** | ... | ... |
| **23781** | 2015-12-24 | 0.002 |
| **23782** | 2015-12-28 | -0.008 |
| **23783** | 2015-12-29 | 0.006 |
| **23784** | 2015-12-30 | -0.007 |
| **23785** | 2015-12-31 | -0.002 |

23786 rows × 2 columns

In [ ]:
```
theta = 0.1
delta = 0.7
count=0
for permno in crsp['PERMNO'].unique():
    crsp_sample = crsp[crsp['PERMNO'] == permno].copy()
    crsp_sample = pd.merge(crsp_sample, crsp_index, on='date', how='left')
    crsp_sample['salience'] = abs(crsp_sample['RET_x'] - crsp_sample['ewretd']) / (
                abs(crsp_sample['ewretd']) + abs(crsp_sample['RET_x']) + theta)

    # Group by ticker and month and iterate over each group
    for name, group in crsp_sample.groupby(['TICKER', 'month']):
        # Rank the salience values
        group['salience_rank'] = group['salience'].rank(ascending=False)
        # Calculate the salience weight
        group['salience_weight'] = delta / (group['salience_rank'] * delta * (1 / l
        # Add the salience weight to the dataframe
        crsp_sample.loc[group.index, 'salience_weight'] = group['salience_weight']

        # Calculate Salience theory value ST
        cov_matrix = np.cov(group['RET_x'], group['salience_weight'])
```

```
        crsp_sample.loc[group.index, 'ST'] = cov_matrix[0][1]
    #Make the index of crsp_sample same as crsp['PERMNO'] == permno
    crsp_sample.set_index(crsp[crsp['PERMNO'] == permno].index, inplace=True)

    # Add the 'ST' column to the original DataFrame
    crsp.loc[crsp[crsp['PERMNO'] == permno].index, 'ST'] = crsp_sample['ST']
    count+=1
    if count%100 == 0:
        print("Processed PERMNO: ", count)
```

```
Processed PERMNO:  100
Processed PERMNO:  200
Processed PERMNO:  300
Processed PERMNO:  400
Processed PERMNO:  500
Processed PERMNO:  600
Processed PERMNO:  700
Processed PERMNO:  800
Processed PERMNO:  900
Processed PERMNO:  1000
Processed PERMNO:  1100
Processed PERMNO:  1200
Processed PERMNO:  1300
Processed PERMNO:  1400
Processed PERMNO:  1500
Processed PERMNO:  1600
Processed PERMNO:  1700
Processed PERMNO:  1800
Processed PERMNO:  1900
Processed PERMNO:  2000
Processed PERMNO:  2100
Processed PERMNO:  2200
Processed PERMNO:  2300
Processed PERMNO:  2400
Processed PERMNO:  2500
Processed PERMNO:  2600
Processed PERMNO:  2700
Processed PERMNO:  2800
Processed PERMNO:  2900
Processed PERMNO:  3000
Processed PERMNO:  3100
Processed PERMNO:  3200
Processed PERMNO:  3300
Processed PERMNO:  3400
Processed PERMNO:  3500
Processed PERMNO:  3600
Processed PERMNO:  3700
Processed PERMNO:  3800
Processed PERMNO:  3900
Processed PERMNO:  4000
Processed PERMNO:  4100
Processed PERMNO:  4200
Processed PERMNO:  4300
Processed PERMNO:  4400
Processed PERMNO:  4500
Processed PERMNO:  4600
Processed PERMNO:  4700
Processed PERMNO:  4800
Processed PERMNO:  4900
Processed PERMNO:  5000
Processed PERMNO:  5100
Processed PERMNO:  5200
Processed PERMNO:  5300
Processed PERMNO:  5400
Processed PERMNO:  5500
Processed PERMNO:  5600
```

```
Processed PERMNO:  5700
Processed PERMNO:  5800
Processed PERMNO:  5900
Processed PERMNO:  6000
Processed PERMNO:  6100
Processed PERMNO:  6200
Processed PERMNO:  6300
Processed PERMNO:  6400
Processed PERMNO:  6500
Processed PERMNO:  6600
Processed PERMNO:  6700
Processed PERMNO:  6800
Processed PERMNO:  6900
Processed PERMNO:  7000
Processed PERMNO:  7100
Processed PERMNO:  7200
Processed PERMNO:  7300
Processed PERMNO:  7400
Processed PERMNO:  7500
Processed PERMNO:  7600
Processed PERMNO:  7700
Processed PERMNO:  7800
Processed PERMNO:  7900
Processed PERMNO:  8000
Processed PERMNO:  8100
Processed PERMNO:  8200
Processed PERMNO:  8300
Processed PERMNO:  8400
Processed PERMNO:  8500
Processed PERMNO:  8600
Processed PERMNO:  8700
Processed PERMNO:  8800
Processed PERMNO:  8900
Processed PERMNO:  9000
Processed PERMNO:  9100
Processed PERMNO:  9200
Processed PERMNO:  9300
Processed PERMNO:  9400
Processed PERMNO:  9500
Processed PERMNO:  9600
Processed PERMNO:  9700
Processed PERMNO:  9800
Processed PERMNO:  9900
Processed PERMNO:  10000
Processed PERMNO:  10100
Processed PERMNO:  10200
Processed PERMNO:  10300
Processed PERMNO:  10400
Processed PERMNO:  10500
Processed PERMNO:  10600
Processed PERMNO:  10700
Processed PERMNO:  10800
Processed PERMNO:  10900
Processed PERMNO:  11000
Processed PERMNO:  11100
Processed PERMNO:  11200
```

```
Processed PERMNO:  11300
Processed PERMNO:  11400
Processed PERMNO:  11500
Processed PERMNO:  11600
Processed PERMNO:  11700
Processed PERMNO:  11800
Processed PERMNO:  11900
Processed PERMNO:  12000
Processed PERMNO:  12100
Processed PERMNO:  12200
Processed PERMNO:  12300
Processed PERMNO:  12400
Processed PERMNO:  12500
Processed PERMNO:  12600
Processed PERMNO:  12700
Processed PERMNO:  12800
Processed PERMNO:  12900
Processed PERMNO:  13000
Processed PERMNO:  13100
Processed PERMNO:  13200
Processed PERMNO:  13300
Processed PERMNO:  13400
Processed PERMNO:  13500
Processed PERMNO:  13600
Processed PERMNO:  13700
Processed PERMNO:  13800
Processed PERMNO:  13900
Processed PERMNO:  14000
Processed PERMNO:  14100
Processed PERMNO:  14200
Processed PERMNO:  14300
Processed PERMNO:  14400
Processed PERMNO:  14500
Processed PERMNO:  14600
Processed PERMNO:  14700
Processed PERMNO:  14800
Processed PERMNO:  14900
Processed PERMNO:  15000
Processed PERMNO:  15100
Processed PERMNO:  15200
Processed PERMNO:  15300
Processed PERMNO:  15400
Processed PERMNO:  15500
Processed PERMNO:  15600
Processed PERMNO:  15700
Processed PERMNO:  15800
Processed PERMNO:  15900
Processed PERMNO:  16000
Processed PERMNO:  16100
Processed PERMNO:  16200
Processed PERMNO:  16300
Processed PERMNO:  16400
Processed PERMNO:  16500
Processed PERMNO:  16600
Processed PERMNO:  16700
Processed PERMNO:  16800
```

```
Processed PERMNO:  16900
Processed PERMNO:  17000
Processed PERMNO:  17100
Processed PERMNO:  17200
Processed PERMNO:  17300
Processed PERMNO:  17400
Processed PERMNO:  17500
Processed PERMNO:  17600
Processed PERMNO:  17700
Processed PERMNO:  17800
Processed PERMNO:  17900
Processed PERMNO:  18000
Processed PERMNO:  18100
Processed PERMNO:  18200
Processed PERMNO:  18300
Processed PERMNO:  18400
Processed PERMNO:  18500
Processed PERMNO:  18600
Processed PERMNO:  18700
Processed PERMNO:  18800
Processed PERMNO:  18900
Processed PERMNO:  19000
Processed PERMNO:  19100
Processed PERMNO:  19200
Processed PERMNO:  19300
Processed PERMNO:  19400
Processed PERMNO:  19500
Processed PERMNO:  19600
Processed PERMNO:  19700
Processed PERMNO:  19800
Processed PERMNO:  19900
Processed PERMNO:  20000
Processed PERMNO:  20100
Processed PERMNO:  20200
Processed PERMNO:  20300
Processed PERMNO:  20400
Processed PERMNO:  20500
Processed PERMNO:  20600
Processed PERMNO:  20700
Processed PERMNO:  20800
Processed PERMNO:  20900
Processed PERMNO:  21000
Processed PERMNO:  21100
Processed PERMNO:  21200
Processed PERMNO:  21300
Processed PERMNO:  21400
Processed PERMNO:  21500
Processed PERMNO:  21600
Processed PERMNO:  21700
Processed PERMNO:  21800
Processed PERMNO:  21900
Processed PERMNO:  22000
Processed PERMNO:  22100
Processed PERMNO:  22200
Processed PERMNO:  22300
Processed PERMNO:  22400
```

```
Processed PERMNO:  22500
Processed PERMNO:  22600
Processed PERMNO:  22700
Processed PERMNO:  22800
Processed PERMNO:  22900
Processed PERMNO:  23000
Processed PERMNO:  23100
Processed PERMNO:  23200
Processed PERMNO:  23300
Processed PERMNO:  23400
Processed PERMNO:  23500
Processed PERMNO:  23600
Processed PERMNO:  23700
Processed PERMNO:  23800
Processed PERMNO:  23900
Processed PERMNO:  24000
Processed PERMNO:  24100
Processed PERMNO:  24200
Processed PERMNO:  24300
Processed PERMNO:  24400
Processed PERMNO:  24500
Processed PERMNO:  24600
Processed PERMNO:  24700
Processed PERMNO:  24800
Processed PERMNO:  24900
Processed PERMNO:  25000
Processed PERMNO:  25100
Processed PERMNO:  25200
Processed PERMNO:  25300
Processed PERMNO:  25400
Processed PERMNO:  25500
Processed PERMNO:  25600
Processed PERMNO:  25700
Processed PERMNO:  25800
Processed PERMNO:  25900
Processed PERMNO:  26000
Processed PERMNO:  26100
Processed PERMNO:  26200
Processed PERMNO:  26300
Processed PERMNO:  26400
Processed PERMNO:  26500
Processed PERMNO:  26600
Processed PERMNO:  26700
Processed PERMNO:  26800
Processed PERMNO:  26900
Processed PERMNO:  27000
Processed PERMNO:  27100
Processed PERMNO:  27200
Processed PERMNO:  27300
Processed PERMNO:  27400
Processed PERMNO:  27500
Processed PERMNO:  27600
```

## PLS SAVE ABOVE CRSP FOR NOT RUNNING ST AGAIN

In [ ]:
```
#crsp.to_csv("C:/Users/Aman/Downloads/Compressed/crsp_filtered_merged.csv")
```

## ONLY READ IF NOT RUNNING PREVIOUS CELLS -->

In [ ]:
```
crsp = pd.read_csv("C:/Users/Aman/Downloads/Compressed/crsp_filtered_merged.csv")
crsp['date'] = pd.to_datetime(crsp['date'])
```

### RUN THIS ONE NONE THE LESS (this is to add Shares outstanding) -->

In [ ]:
```
crsp_monthly = pd.read_csv("C:/Users/Aman/Downloads/Compressed/crsp_us_equity_month
crsp_monthly['date'] = pd.to_datetime(crsp_monthly['date'])
#Multiply SHROUT by 1000
crsp_monthly['SHROUT'] = crsp_monthly['SHROUT'] * 1000
crsp_monthly.drop(columns=['COMNAM','PRC','VOL','RET'], inplace=True)
# merge crsp and crsp_monthly on TICKER, PERMNO and date and drop a
crsp = pd.merge(crsp, crsp_monthly, on=['TICKER', 'PERMNO', 'date'], how='left')
```

In [ ]:
```
# Create deciles based on the 'ST' column
crsp['Decile'] = pd.qcut(crsp['ST'], q=[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
                         labels=['Low', '2', '3', '4', '5', '6', '7', '8', '9', 'High
# Keep only month end dates and group by decile and take average of ST
crsp[crsp['date'].dt.is_month_end].groupby(['Decile'])['ST'].mean()*24
```

Out[ ]:
```
Decile
Low    -2.378
2      -1.090
3      -0.673
4      -0.363
5      -0.010
6       0.377
7       0.710
8       1.093
9       1.660
High    3.395
Name: ST, dtype: float64
```

### WE HAVE THREE PRICE COLUMNS, PRC_x, PRC_y, PRC_y_daily. TRY ALL THREE AND SEE WHICH ONE IS CLOSE TO IMAGE. I think you have to use PRC_y_daily though

In [ ]:
```
crsp[crsp['date'].dt.is_month_end].groupby(['Decile'])['PRC_y_daily'].mean()
```

Out[ ]:
```
Decile
Low     19.021
2       24.960
3       32.298
4       49.958
5       53.095
6       52.072
7       36.057
8       29.353
9       22.277
High    18.125
Name: PRC_y_daily, dtype: float64
```

# ME Calculation

In [ ]:
```
# Create a new columns called ME which is PRC * SHROUT_y
crsp['ME'] = crsp['PRC_y_daily'] * crsp['SHROUT']
```

In [ ]:
```
np.log(crsp[crsp['date'].dt.is_month_end].groupby(['Decile'])['ME'].mean())
```

Out[ ]:
```
Decile
Low     20.804
2       21.122
3       22.005
4       23.188
5       23.557
6       23.540
7       22.542
8       21.520
9       20.820
High    20.994
Name: ME, dtype: float64
```

## BOOK VALUE OF EQUITY (BE) Calculation

In [ ]:
```
# Read "C:\Users\Aman\Downloads\Compressed\compustat_us_equity.csv"
compustat_yearly = pd.read_csv("C:/Users/Aman/Downloads/Compressed/compustat_us_equ
```

In [ ]:
```
# Drop indfmt   consol   popsrc   datafmt conm curcd costat columns
compustat_yearly.drop(columns=['indfmt', 'consol', 'popsrc', 'datafmt', 'conm', 'cu
```

In [ ]:
```
#Rename datadate to date and convert it to datetime
compustat_yearly.rename(columns={'datadate':'date'}, inplace=True)
compustat_yearly['date'] = pd.to_datetime(compustat_yearly['date'])

#Rename bkvlps to book_value_per_share, csho to shares_outstanding, mkvalt to marke
compustat_yearly.rename(columns={'bkvlps':'book_value_per_share', 'csho':'shares_ou

compustat_yearly['BE'] = compustat_yearly['book_value_per_share'] * compustat_yearl
```

In [ ]:
```
compustat_yearly
```

Out[ ]:

| | gvkey | date | fyear | tic | book_value_per_share | shares_outstanding | marke |
|---|---|---|---|---|---|---|---|
| **0** | 1000 | 1961-12-31 | 1961.000 | AE.2 | 2.434 | 0.152 | |
| **1** | 1000 | 1962-12-31 | 1962.000 | AE.2 | 3.050 | 0.181 | |
| **2** | 1000 | 1963-12-31 | 1963.000 | AE.2 | 2.973 | 0.186 | |
| **3** | 1000 | 1964-12-31 | 1964.000 | AE.2 | 3.097 | 0.196 | |
| **4** | 1000 | 1965-12-31 | 1965.000 | AE.2 | 2.384 | 0.206 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **515983** | 328795 | 2013-12-31 | 2013.000 | ACA | NaN | NaN | |
| **515984** | 328795 | 2014-12-31 | 2014.000 | ACA | NaN | NaN | |
| **515985** | 328795 | 2015-12-31 | 2015.000 | ACA | NaN | NaN | |
| **515986** | 335466 | 2015-12-31 | 2015.000 | HOFSQ | NaN | NaN | |
| **515987** | 345980 | 2015-12-31 | 2015.000 | WISH | NaN | NaN | |

515988 rows × 8 columns

## Loading the Key to Connect CRSP and Compustat (Permno to GVKEY)

In [ ]:

```
permno_gvkey = pd.read_csv("C:/Users/Aman/Downloads/Compressed/permno_gvkey.csv")
```

In [ ]:

```
permno_gvkey["LINKDT"] = pd.to_datetime(permno_gvkey["LINKDT"])

# Replace "E" in LINKENDDT with today's date
permno_gvkey["LINKENDDT"].replace({"E": '2016-01-01'}, inplace=True)

permno_gvkey["LINKENDDT"] = pd.to_datetime(permno_gvkey["LINKENDDT"])

permno_gvkey
```

Out[ ]:

| | GVKEY | LINKTYPE | LPERMNO | LPERMCO | LINKDT | LINKENDDT | CONM |
|---|---|---|---|---|---|---|---|
| **0** | 1000 | LU | 25881 | 23369 | 1970-11-13 | 1978-06-30 | A & E PLASTIK PAK INC |
| **1** | 1001 | LU | 10015 | 6398 | 1983-09-20 | 1986-07-31 | A & M FOOD SERVICES INC |
| **2** | 1002 | LC | 10023 | 22159 | 1972-12-14 | 1973-06-05 | AAI CORP |
| **3** | 1003 | LU | 10031 | 6672 | 1983-12-07 | 1989-08-16 | A.A. IMPORTING CO INC |
| **4** | 1004 | LU | 54594 | 20000 | 1972-04-24 | 2016-01-01 | AAR CORP |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **32928** | 349994 | LC | 23514 | 59438 | 2022-11-15 | 2016-01-01 | CLEARMIND MEDICINE INC |
| **32929** | 350681 | LC | 22205 | 58855 | 2021-10-22 | 2023-03-31 | GETNET ADQUIRENCIA E |
| **32930** | 351038 | LC | 16161 | 55612 | 2021-10-29 | 2016-01-01 | QUOIN PHARMACEUTICALS LTD |
| **32931** | 352262 | LC | 23773 | 59507 | 2023-03-17 | 2016-01-01 | COOL COMPANY LTD |
| **32932** | 353444 | LC | 23209 | 59330 | 2022-07-22 | 2016-01-01 | HALEON PLC |

32933 rows × 7 columns

In [ ]:

```
# Create a dictionary with GVKEY as key and PERMNO as value.
permno_gvkey_dict = dict(zip(permno_gvkey['GVKEY'], permno_gvkey['LPERMNO']))
```

In [ ]:

```
# Use the disctionary permno_gvkey_dict to add a new column PERMNO to compustat_yea
compustat_yearly['PERMNO'] = compustat_yearly['gvkey'].map(permno_gvkey_dict)
```

In [ ]:

```
#Count the number of missing PERMNO values
compustat_yearly['PERMNO'].isna().sum()
# Drop all rows with missing PERMNO values
compustat_yearly.dropna(subset=['PERMNO'], inplace=True)
# Convert PERMNO to int
compustat_yearly['PERMNO'] = compustat_yearly['PERMNO'].astype(int)
compustat_yearly
```

Out[ ]:

| | gvkey | date | fyear | tic | book_value_per_share | shares_outstanding | marke |
|---|---|---|---|---|---|---|---|
| **0** | 1000 | 1961-12-31 | 1961.000 | AE.2 | 2.434 | 0.152 | |
| **1** | 1000 | 1962-12-31 | 1962.000 | AE.2 | 3.050 | 0.181 | |
| **2** | 1000 | 1963-12-31 | 1963.000 | AE.2 | 2.973 | 0.186 | |
| **3** | 1000 | 1964-12-31 | 1964.000 | AE.2 | 3.097 | 0.196 | |
| **4** | 1000 | 1965-12-31 | 1965.000 | AE.2 | 2.384 | 0.206 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **515983** | 328795 | 2013-12-31 | 2013.000 | ACA | NaN | NaN | |
| **515984** | 328795 | 2014-12-31 | 2014.000 | ACA | NaN | NaN | |
| **515985** | 328795 | 2015-12-31 | 2015.000 | ACA | NaN | NaN | |
| **515986** | 335466 | 2015-12-31 | 2015.000 | HOFSQ | NaN | NaN | |
| **515987** | 345980 | 2015-12-31 | 2015.000 | WISH | NaN | NaN | |

413594 rows × 9 columns

In [ ]:

```
# Read table from text file DFF_BE_With_Nonindust.txt in same folder with sep as sp
equity_old = pd.read_csv("C:/Users/Aman/Downloads/Compressed/kenneth_old_data.csv")
```

In [ ]:

```
equity_old.drop(columns=['first_moody','last_moody'], inplace=True)
equity_old_melted = pd.melt(equity_old, id_vars=['PERMNO'], var_name='date', value_

# Conver the year to end of year date
equity_old_melted['date'] = pd.to_datetime(equity_old_melted['date'], format='%Y')
equity_old_melted['date'] = equity_old_melted['date'] + pd.offsets.YearEnd(0)

equity_old_melted['BE'] = equity_old_melted['value'] * 1000000
equity_old_melted.drop(columns=['value'], inplace=True)
equity_old_melted
```

Out[ ]:

| | PERMNO | date | BE |
|---|---|---|---|
| **0** | 10006 | 1926-12-31 | 67743000.000 |
| **1** | 10014 | 1926-12-31 | 13005000.000 |
| **2** | 10022 | 1926-12-31 | 13567000.000 |
| **3** | 10030 | 1926-12-31 | 15924000.000 |
| **4** | 10049 | 1926-12-31 | 11984000.000 |
| **...** | ... | ... | ... |
| **136339** | 86134 | 2001-12-31 | -99990000.000 |
| **136340** | 86239 | 2001-12-31 | -99990000.000 |
| **136341** | 86861 | 2001-12-31 | -99990000.000 |
| **136342** | 92567 | 2001-12-31 | -99990000.000 |
| **136343** | 93172 | 2001-12-31 | -99990000.000 |

136344 rows × 3 columns

In [ ]:
```
# Merge compustat_yearly and equity_old_melted on PERMNO and date
compustat_yearly = pd.merge(compustat_yearly, equity_old_melted, on=['PERMNO', 'dat
```

In [ ]:
```
#Create a combined book_value_per_share column
compustat_yearly['BE'] = compustat_yearly['BE_x'].fillna(compustat_yearly['BE_y'])
```

In [ ]:
```
compustat_yearly.drop(columns=['BE_x', 'BE_y'], inplace=True)
```

In [ ]:
```
#Drop all rows with missing book_value_per_share values
compustat_yearly.dropna(subset=['book_value_per_share'], inplace=True)
#Drop all rows with negative book_value_per_share values
compustat_yearly = compustat_yearly[compustat_yearly['book_value_per_share'] > 0]
```

In [ ]:
```
compustat_yearly_for_merge = compustat_yearly.copy()
compustat_yearly_for_merge.drop(columns=['gvkey', 'fyear','tic','shares_outstanding
compustat_yearly_for_merge
```

Out[ ]:

| | date | PERMNO | BE |
|---|---|---|---|
| **0** | 1961-12-31 | 25881 | 369998.400 |
| **1** | 1962-12-31 | 25881 | 551995.700 |
| **2** | 1963-12-31 | 25881 | 552996.600 |
| **3** | 1964-12-31 | 25881 | 606992.400 |
| **4** | 1965-12-31 | 25881 | 491001.000 |
| **...** | ... | ... | ... |
| **413580** | 2015-12-31 | 16496 | 5361991681.200 |
| **413581** | 2013-12-31 | 15904 | 1260040.000 |
| **413582** | 2014-12-31 | 15904 | 66095521.700 |
| **413583** | 2015-12-31 | 15904 | 67665602.400 |
| **413585** | 2015-12-31 | 16469 | 29722734.000 |

304351 rows × 3 columns

In [ ]:

```
crsp.drop(columns=['Unnamed: 0'], inplace=True)
```

In [ ]:

```
# add the book_value_per_share to crsp
crsp = pd.merge(crsp, compustat_yearly_for_merge, on=['PERMNO', 'date'], how='left'
crsp
```

Out[ ]:

| | PERMNO | date | TICKER | PRC_x | VOL_x | RET_x | month | days_in_month | PI |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 10495 | 1962-07-02 | A | 41.125 | 2600.000 | 0.022 | 1962-07 | 21.000 | |
| **1** | 10495 | 1962-07-03 | A | 41.375 | 2100.000 | 0.006 | 1962-07 | 21.000 | |
| **2** | 10495 | 1962-07-05 | A | 41.250 | 3600.000 | -0.003 | 1962-07 | 21.000 | |
| **3** | 10495 | 1962-07-06 | A | 40.500 | 2600.000 | -0.018 | 1962-07 | 21.000 | |
| **4** | 10495 | 1962-07-09 | A | 40.750 | 4000.000 | 0.006 | 1962-07 | 21.000 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **59173811** | 91205 | 2008-10-27 | ZZ | 2.710 | 443900.000 | -0.029 | 2008-10 | 23.000 | |
| **59173812** | 91205 | 2008-10-28 | ZZ | 2.750 | 369000.000 | 0.015 | 2008-10 | 23.000 | |
| **59173813** | 91205 | 2008-10-29 | ZZ | 3.000 | 610400.000 | 0.091 | 2008-10 | 23.000 | |
| **59173814** | 91205 | 2008-10-30 | ZZ | 3.300 | 718200.000 | 0.100 | 2008-10 | 23.000 | |
| **59173815** | 91205 | 2008-10-31 | ZZ | 3.230 | 681000.000 | -0.021 | 2008-10 | 23.000 | 6 |

59173816 rows × 17 columns

In [ ]:

```
# Calculate book to market ratio
crsp['BM'] = crsp['BE'] / crsp['ME']
crsp
```

Out[ ]:

| | PERMNO | date | TICKER | PRC_x | VOL_x | RET_x | month | days_in_month | PI |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 10495 | 1962-07-02 | A | 41.125 | 2600.000 | 0.022 | 1962-07 | 21.000 | |
| **1** | 10495 | 1962-07-03 | A | 41.375 | 2100.000 | 0.006 | 1962-07 | 21.000 | |
| **2** | 10495 | 1962-07-05 | A | 41.250 | 3600.000 | -0.003 | 1962-07 | 21.000 | |
| **3** | 10495 | 1962-07-06 | A | 40.500 | 2600.000 | -0.018 | 1962-07 | 21.000 | |
| **4** | 10495 | 1962-07-09 | A | 40.750 | 4000.000 | 0.006 | 1962-07 | 21.000 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **59173811** | 91205 | 2008-10-27 | ZZ | 2.710 | 443900.000 | -0.029 | 2008-10 | 23.000 | |
| **59173812** | 91205 | 2008-10-28 | ZZ | 2.750 | 369000.000 | 0.015 | 2008-10 | 23.000 | |
| **59173813** | 91205 | 2008-10-29 | ZZ | 3.000 | 610400.000 | 0.091 | 2008-10 | 23.000 | |
| **59173814** | 91205 | 2008-10-30 | ZZ | 3.300 | 718200.000 | 0.100 | 2008-10 | 23.000 | |
| **59173815** | 91205 | 2008-10-31 | ZZ | 3.230 | 681000.000 | -0.021 | 2008-10 | 23.000 | 6 |

59173816 rows × 18 columns

In [ ]:
```python
np.log(crsp[crsp['date'].dt.is_year_end].groupby(['Decile'])['BM'].mean())
```

Out[ ]:
```
Decile
Low     0.711
2       1.044
3       1.313
4       1.014
5       1.025
6       1.059
7       1.114
8       1.757
9       1.600
High    0.620
Name: BM, dtype: float64
```

In [ ]:
```python
# Count how many 'C' values are there in the 'RET_y' column
crsp[crsp['RET_y']=='C']
# Take all the rows where 'RET_y' is 'C' and compute it using the RET_x cumulativel
crsp.loc[crsp[crsp['RET_y']=='C'].index, 'RET_y'] = crsp.loc[crsp[crsp['RET_y']=='C
```

In [ ]:
```
#save crsp_monthly
crsp_monthly = crsp[crsp['date'].dt.is_month_end]
```

In [ ]:
```
# cum_monthly_ret is the 13 month cumulative return - 2 month cumulative return
crsp_monthly['cum_monthly_ret'] = crsp_monthly.groupby(['PERMNO'])['RET_y'].transfo
```

In [ ]:
```
# rename cum_monthly_ret to MOM
crsp_monthly.rename(columns={'cum_monthly_ret':'MOM'}, inplace=True)
```

In [ ]:
```
crsp_monthly.groupby(['Decile'])['MOM'].mean()*100
```

Out[ ]:
```
Decile
Low     20.581
2       19.490
3       19.009
4       18.140
5       16.427
6       18.492
7       20.468
8       21.638
9       21.503
High    20.115
Name: MOM, dtype: float64
```

## ILLIQ

In [ ]:
```
# Load the daily price data
#crsp_price = pd.read_csv("C:/Users/Aman/Downloads/Compressed/crsp_us_equity.csv")
```

In [ ]:
```
#crsp_price.drop(columns=['TICKER','VOL','RET'], inplace=True)
```

In [ ]:
```
#crsp_price['date'] = pd.to_datetime(crsp_price['date'])
```

In [ ]:
```
# merge crsp_price into crsp
#crsp = pd.merge(crsp, crsp_price, on=['date', 'PERMNO'], how='left')
```

In [ ]:
```
#crsp.drop(columns=['PRC_x'], inplace=True)
```

In [ ]:
```
crsp['ILLIQ'] = abs(crsp['RET_x']) *1000000 / (crsp['VOL_x'] * crsp['PRC_x'])
```

In [ ]:
```
# find inf values in ILLIQ and replace them with NaN
crsp['ILLIQ'].replace([np.inf, -np.inf], np.nan, inplace=True)
```

In [ ]:
```
#Create a new columns 'ILLIQ_monthly' which is the average of ILLIQ for each month
crsp['ILLIQ_monthly'] = crsp.groupby(['PERMNO', 'month'])['ILLIQ'].transform('mean'
```

In [ ]:
```
crsp.groupby(['Decile'])['ILLIQ_monthly'].mean()
```

Out[ ]:
```
Decile
Low     1.192
2       0.757
3       0.525
4       0.323
5       0.135
6       0.304
7       0.459
8       0.595
9       0.807
High    0.989
Name: ILLIQ_monthly, dtype: float64
```

## THRESHOLD POINT

In [ ]:
```
# crsp.to_csv("C:/Users/Aman/Downloads/Compressed/crsp_filtered_merged_illiq.csv")
# crsp = pd.read_csv("C:/Users/Aman/Downloads/Compressed/crsp_filtered_merged_illiq
```

In [ ]:
```
#crsp.drop(columns=['VOL_x','days_in_month','PRC_y','VOL_y','PRC_y_daily','SHROUT',
```

In [ ]:
```
#crsp.to_csv("C:/Users/Aman/Downloads/Compressed/crsp_filtered_merged_illiq_skinny.
crsp = pd.read_csv("C:/Users/Aman/Downloads/Compressed/crsp_filtered_merged_illiq_s
```

In [ ]:
```
crsp['date'] = pd.to_datetime(crsp['date'])
```

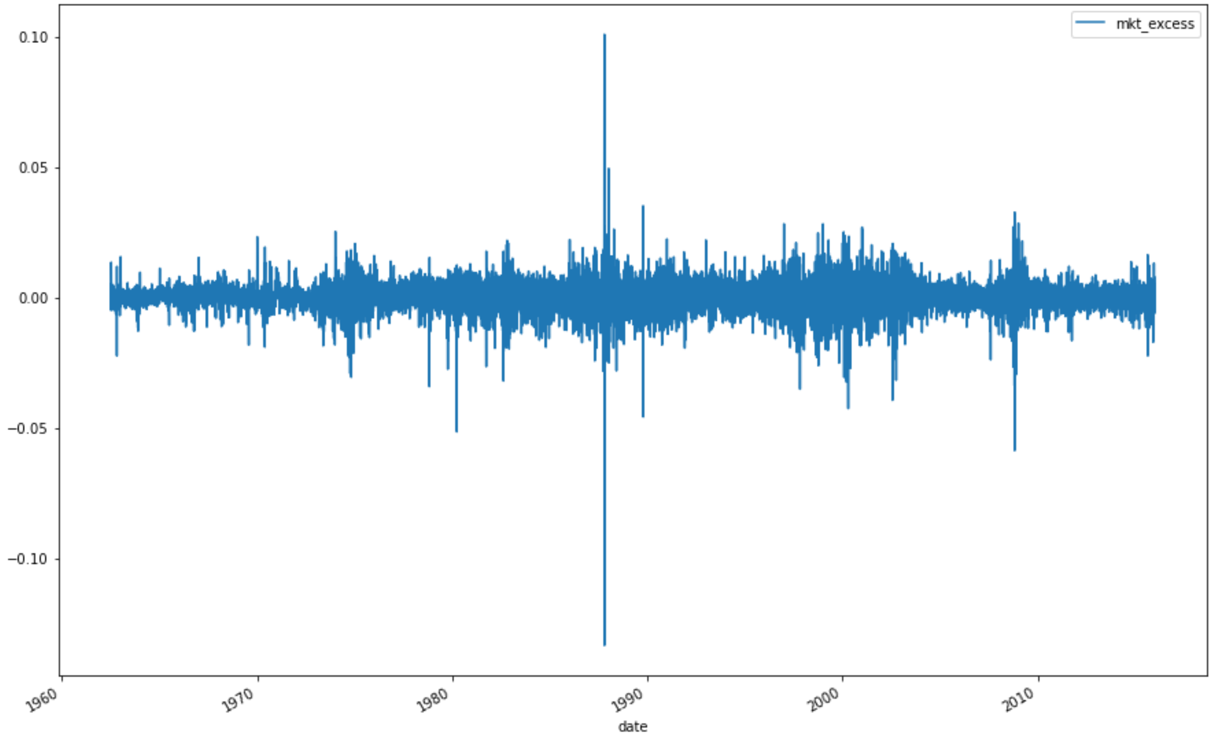
## BETA

In [ ]:
```
crsp_index = pd.read_csv("C:/Users/Aman/Downloads/Compressed/crsp_index_1.csv")
crsp_index['caldt'] = pd.to_datetime(crsp_index['caldt'])
crsp_index.rename(columns={'caldt':'date'}, inplace=True)
```

In [ ]:
```
crsp_index['mkt_excess'] = crsp_index['ewretd'] - crsp_index['sprtrn']
```

In [ ]:
```
crsp_index.plot(x='date', y=['mkt_excess'], figsize=(15,10))
```

Out[ ]:
```
<AxesSubplot:xlabel='date'>
```




In [ ]:
```
crsp_index.dropna(subset=['sprtrn'], inplace=True)
```

In [ ]:
```
crsp
```

Out[ ]:

| | PERMNO | date | TICKER | PRC_x | RET_x | month | RET_y | ST | Decile | |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 10495 | 1962-07-02 | A | 41.125 | 0.022 | 1962-07 | NaN | -0.019 | 4 | |
| **1** | 10495 | 1962-07-03 | A | 41.375 | 0.006 | 1962-07 | NaN | -0.019 | 4 | |
| **2** | 10495 | 1962-07-05 | A | 41.250 | -0.003 | 1962-07 | NaN | -0.019 | 4 | |
| **3** | 10495 | 1962-07-06 | A | 40.500 | -0.018 | 1962-07 | NaN | -0.019 | 4 | |
| **4** | 10495 | 1962-07-09 | A | 40.750 | 0.006 | 1962-07 | NaN | -0.019 | 4 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **59173811** | 91205 | 2008-10-27 | ZZ | 2.710 | -0.029 | 2008-10 | NaN | -0.147 | Low | |
| **59173812** | 91205 | 2008-10-28 | ZZ | 2.750 | 0.015 | 2008-10 | NaN | -0.147 | Low | |
| **59173813** | 91205 | 2008-10-29 | ZZ | 3.000 | 0.091 | 2008-10 | NaN | -0.147 | Low | |
| **59173814** | 91205 | 2008-10-30 | ZZ | 3.300 | 0.100 | 2008-10 | NaN | -0.147 | Low | |
| **59173815** | 91205 | 2008-10-31 | ZZ | 3.230 | -0.021 | 2008-10 | -0.500000 | -0.147 | Low | 593( |

59173816 rows × 13 columns

In [ ]:

```
#BETA is the market beta, estimated from a regression of daily excess stock returns
count=0
for permno in crsp['PERMNO'].unique():
    crsp_sample = crsp[crsp['PERMNO'] == permno].copy()

    #Drop all rows with missing Ret values
    crsp_sample.dropna(subset=['RET_x'], inplace=True)

    crsp_sample = pd.merge(crsp_sample, crsp_index, on='date', how='left')

    # ASSUMPTION: -- Forward fill the missing mktexcess values
    crsp_sample['mkt_excess'].fillna(method='ffill', inplace=True)
    crsp_sample['mkt_excess'].fillna(method='bfill', inplace=True)

    crsp_sample['stock_excess'] = crsp_sample['RET_x'] - crsp_sample['sprtrn']

    crsp_sample['stock_excess'].fillna(method='ffill', inplace=True)
    crsp_sample['stock_excess'].fillna(method='bfill', inplace=True)

    crsp_sample['ewretd'].fillna(method='ffill', inplace=True)
```

```
    crsp_sample['ewretd'].fillna(method='bfill', inplace=True)

    # Group by ticker and month and iterate over each group
    for name, group in crsp_sample.groupby(['TICKER', 'month']):
        # Run a regression of stock_excess on mkt_excess
        y = group['RET_x']
        x = group['ewretd']
        x = sm.add_constant(x)
        model = sm.OLS(y, x).fit()
        try:
            crsp_sample.loc[group.index, 'BETA'] = model.params[1]
            #Calculate the std deviation of the residuals
            crsp_sample.loc[group.index, 'IVOL'] = model.resid.std()
        except:
            crsp_sample.loc[group.index, 'BETA'] = np.nan
            crsp_sample.loc[group.index, 'IVOL'] = np.nan

    #Make the index of crsp_sample same as crsp['PERMNO'] == permno
    crsp_sample.set_index(crsp[crsp['PERMNO'] == permno].index, inplace=True)

    # Add the 'BETA' column to the original DataFrame
    crsp.loc[crsp[crsp['PERMNO'] == permno].index, 'BETA'] = crsp_sample['BETA']

    # Add the 'IVOL' column to the original DataFrame
    crsp.loc[crsp[crsp['PERMNO'] == permno].index, 'IVOL'] = crsp_sample['IVOL']
s
    count+=1
    if count%100 == 0:
        print("Processed PERMNO: ", count)
```

```
Processed PERMNO:  100
Processed PERMNO:  200
Processed PERMNO:  300
Processed PERMNO:  400
Processed PERMNO:  500
Processed PERMNO:  600
Processed PERMNO:  700
Processed PERMNO:  800
Processed PERMNO:  900
Processed PERMNO:  1000
Processed PERMNO:  1100
Processed PERMNO:  1200
Processed PERMNO:  1300
Processed PERMNO:  1400
Processed PERMNO:  1500
Processed PERMNO:  1600
Processed PERMNO:  1700
Processed PERMNO:  1800
Processed PERMNO:  1900
Processed PERMNO:  2000
Processed PERMNO:  2100
Processed PERMNO:  2200
Processed PERMNO:  2300
Processed PERMNO:  2400
Processed PERMNO:  2500
Processed PERMNO:  2600
Processed PERMNO:  2700
Processed PERMNO:  2800
Processed PERMNO:  2900
Processed PERMNO:  3000
Processed PERMNO:  3100
Processed PERMNO:  3200
Processed PERMNO:  3300
Processed PERMNO:  3400
Processed PERMNO:  3500
Processed PERMNO:  3600
Processed PERMNO:  3700
Processed PERMNO:  3800
Processed PERMNO:  3900
Processed PERMNO:  4000
Processed PERMNO:  4100
Processed PERMNO:  4200
Processed PERMNO:  4300
Processed PERMNO:  4400
Processed PERMNO:  4500
Processed PERMNO:  4600
Processed PERMNO:  4700
Processed PERMNO:  4800
Processed PERMNO:  4900
Processed PERMNO:  5000
Processed PERMNO:  5100
Processed PERMNO:  5200
Processed PERMNO:  5300
Processed PERMNO:  5400
Processed PERMNO:  5500
Processed PERMNO:  5600
```

```
Processed PERMNO:  5700
Processed PERMNO:  5800
Processed PERMNO:  5900
Processed PERMNO:  6000
Processed PERMNO:  6100
Processed PERMNO:  6200
Processed PERMNO:  6300
Processed PERMNO:  6400
Processed PERMNO:  6500
Processed PERMNO:  6600
Processed PERMNO:  6700
Processed PERMNO:  6800
Processed PERMNO:  6900
Processed PERMNO:  7000
Processed PERMNO:  7100
Processed PERMNO:  7200
Processed PERMNO:  7300
Processed PERMNO:  7400
Processed PERMNO:  7500
Processed PERMNO:  7600
Processed PERMNO:  7700
Processed PERMNO:  7800
Processed PERMNO:  7900
Processed PERMNO:  8000
Processed PERMNO:  8100
Processed PERMNO:  8200
Processed PERMNO:  8300
Processed PERMNO:  8400
Processed PERMNO:  8500
Processed PERMNO:  8600
Processed PERMNO:  8700
Processed PERMNO:  8800
Processed PERMNO:  8900
Processed PERMNO:  9000
Processed PERMNO:  9100
Processed PERMNO:  9200
Processed PERMNO:  9300
Processed PERMNO:  9400
Processed PERMNO:  9500
Processed PERMNO:  9600
Processed PERMNO:  9700
Processed PERMNO:  9800
Processed PERMNO:  9900
Processed PERMNO:  10000
Processed PERMNO:  10100
Processed PERMNO:  10200
Processed PERMNO:  10300
Processed PERMNO:  10400
Processed PERMNO:  10500
Processed PERMNO:  10600
Processed PERMNO:  10700
Processed PERMNO:  10800
Processed PERMNO:  10900
Processed PERMNO:  11000
Processed PERMNO:  11100
Processed PERMNO:  11200
```

```
Processed PERMNO:  11300
Processed PERMNO:  11400
Processed PERMNO:  11500
Processed PERMNO:  11600
Processed PERMNO:  11700
Processed PERMNO:  11800
Processed PERMNO:  11900
Processed PERMNO:  12000
Processed PERMNO:  12100
Processed PERMNO:  12200
Processed PERMNO:  12300
Processed PERMNO:  12400
Processed PERMNO:  12500
Processed PERMNO:  12600
Processed PERMNO:  12700
Processed PERMNO:  12800
Processed PERMNO:  12900
Processed PERMNO:  13000
Processed PERMNO:  13100
Processed PERMNO:  13200
Processed PERMNO:  13300
Processed PERMNO:  13400
Processed PERMNO:  13500
Processed PERMNO:  13600
Processed PERMNO:  13700
Processed PERMNO:  13800
Processed PERMNO:  13900
Processed PERMNO:  14000
Processed PERMNO:  14100
Processed PERMNO:  14200
Processed PERMNO:  14300
Processed PERMNO:  14400
Processed PERMNO:  14500
Processed PERMNO:  14600
Processed PERMNO:  14700
Processed PERMNO:  14800
Processed PERMNO:  14900
Processed PERMNO:  15000
Processed PERMNO:  15100
Processed PERMNO:  15200
Processed PERMNO:  15300
Processed PERMNO:  15400
Processed PERMNO:  15500
Processed PERMNO:  15600
Processed PERMNO:  15700
Processed PERMNO:  15800
Processed PERMNO:  15900
Processed PERMNO:  16000
Processed PERMNO:  16100
Processed PERMNO:  16200
Processed PERMNO:  16300
Processed PERMNO:  16400
Processed PERMNO:  16500
Processed PERMNO:  16600
Processed PERMNO:  16700
Processed PERMNO:  16800
```

```
Processed PERMNO:  16900
Processed PERMNO:  17000
Processed PERMNO:  17100
Processed PERMNO:  17200
Processed PERMNO:  17300
Processed PERMNO:  17400
Processed PERMNO:  17500
Processed PERMNO:  17600
Processed PERMNO:  17700
Processed PERMNO:  17800
Processed PERMNO:  17900
Processed PERMNO:  18000
Processed PERMNO:  18100
Processed PERMNO:  18200
Processed PERMNO:  18300
Processed PERMNO:  18400
Processed PERMNO:  18500
Processed PERMNO:  18600
Processed PERMNO:  18700
Processed PERMNO:  18800
Processed PERMNO:  18900
Processed PERMNO:  19000
Processed PERMNO:  19100
Processed PERMNO:  19200
Processed PERMNO:  19300
Processed PERMNO:  19400
Processed PERMNO:  19500
Processed PERMNO:  19600
Processed PERMNO:  19700
Processed PERMNO:  19800
Processed PERMNO:  19900
Processed PERMNO:  20000
Processed PERMNO:  20100
Processed PERMNO:  20200
Processed PERMNO:  20300
Processed PERMNO:  20400
Processed PERMNO:  20500
Processed PERMNO:  20600
Processed PERMNO:  20700
Processed PERMNO:  20800
Processed PERMNO:  20900
Processed PERMNO:  21000
Processed PERMNO:  21100
Processed PERMNO:  21200
Processed PERMNO:  21300
Processed PERMNO:  21400
Processed PERMNO:  21500
Processed PERMNO:  21600
Processed PERMNO:  21700
Processed PERMNO:  21800
Processed PERMNO:  21900
Processed PERMNO:  22000
Processed PERMNO:  22100
Processed PERMNO:  22200
Processed PERMNO:  22300
Processed PERMNO:  22400
```

```
Processed PERMNO:  22500
Processed PERMNO:  22600
Processed PERMNO:  22700
Processed PERMNO:  22800
Processed PERMNO:  22900
Processed PERMNO:  23000
Processed PERMNO:  23100
Processed PERMNO:  23200
Processed PERMNO:  23300
Processed PERMNO:  23400
Processed PERMNO:  23500
Processed PERMNO:  23600
Processed PERMNO:  23700
Processed PERMNO:  23800
Processed PERMNO:  23900
Processed PERMNO:  24000
Processed PERMNO:  24100
Processed PERMNO:  24200
Processed PERMNO:  24300
Processed PERMNO:  24400
Processed PERMNO:  24500
Processed PERMNO:  24600
Processed PERMNO:  24700
Processed PERMNO:  24800
Processed PERMNO:  24900
Processed PERMNO:  25000
Processed PERMNO:  25100
Processed PERMNO:  25200
Processed PERMNO:  25300
Processed PERMNO:  25400
Processed PERMNO:  25500
Processed PERMNO:  25600
Processed PERMNO:  25700
Processed PERMNO:  25800
Processed PERMNO:  25900
Processed PERMNO:  26000
Processed PERMNO:  26100
Processed PERMNO:  26200
Processed PERMNO:  26300
Processed PERMNO:  26400
Processed PERMNO:  26500
Processed PERMNO:  26600
Processed PERMNO:  26700
Processed PERMNO:  26800
Processed PERMNO:  26900
Processed PERMNO:  27000
Processed PERMNO:  27100
Processed PERMNO:  27200
Processed PERMNO:  27300
Processed PERMNO:  27400
Processed PERMNO:  27500
Processed PERMNO:  27600
```

In [ ]:

```
crsp[crsp['date'].dt.is_month_end].groupby(['Decile'])['BETA'].mean()
```

Out[ ]:
```
Decile
Low     1.357
2       1.019
3       0.835
4       0.685
5       0.499
6       0.684
7       0.897
8       1.082
9       1.306
High    1.728
Name: BETA, dtype: float64
```

## IVOL

In [ ]:
```
crsp[crsp['date'].dt.is_month_end].groupby(['Decile'])['IVOL'].mean()*100
```

Out[ ]:
```
Decile
Low     3.517
2       2.086
3       1.595
4       1.221
5       0.836
6       1.233
7       1.638
8       2.077
9       2.682
High    4.450
Name: IVOL, dtype: float64
```

## REV

In [ ]:
```
# pull crsp_monthly from crsp
crsp_monthly = crsp[crsp['date'].dt.is_month_end]
```

In [ ]:
```
crsp_monthly
```

Out[ ]:

| | PERMNO | date | TICKER | PRC_x | RET_x | month | RET_y | ST | Decile | |
|---|---|---|---|---|---|---|---|---|---|---|
| **20** | 10495 | 1962-07-31 | A | 40.375 | -0.006 | 1962-07 | 0.003106 | -0.019 | 4 | 432( |
| **43** | 10495 | 1962-08-31 | A | 40.875 | 0.003 | 1962-08 | 0.024768 | -0.011 | 4 | 432( |
| **85** | 10495 | 1962-10-31 | A | 38.250 | 0.000 | 1962-10 | 0.033784 | 0.028 | 7 | 396‹ |
| **105** | 10495 | 1962-11-30 | A | 41.750 | -0.006 | 1962-11 | 0.117647 | -0.030 | 3 | 409ε |
| **125** | 10495 | 1962-12-31 | A | 40.500 | 0.003 | 1962-12 | -0.029940 | 0.021 | 6 | 447: |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **59173686** | 91205 | 2008-04-30 | ZZ | 6.100 | -0.030 | 2008-04 | -0.197368 | 0.041 | 8 | 691‹ |
| **59173728** | 91205 | 2008-06-30 | ZZ | 5.740 | -0.039 | 2008-06 | -0.077170 | 0.018 | 6 | 565( |
| **59173750** | 91205 | 2008-07-31 | ZZ | 6.830 | 0.012 | 2008-07 | 0.189896 | 0.157 | High | 522‹ |
| **59173792** | 91205 | 2008-09-30 | ZZ | 6.460 | -0.060 | 2008-09 | -0.034380 | -0.026 | 3 | 608( |
| **59173815** | 91205 | 2008-10-31 | ZZ | 3.230 | -0.021 | 2008-10 | -0.500000 | -0.147 | Low | 593( |

1976407 rows × 15 columns

In [ ]:

```
crsp_monthly['RET_1'] = crsp_monthly.groupby(['PERMNO','TICKER'])['RET_y'].shift(1)
crsp_monthly.fillna(method='bfill', inplace=True)
crsp_monthly['RET_1'] = crsp_monthly['RET_1'].astype(float)
crsp_monthly['REV'] = (crsp_monthly['RET_y'].astype(float) / crsp_monthly['RET_1'])
crsp_monthly
```

Out[ ]:

| | PERMNO | date | TICKER | PRC_x | RET_x | month | RET_y | ST | Decile | |
|---|---|---|---|---|---|---|---|---|---|---|
| **20** | 10495 | 1962-07-31 | A | 40.375 | -0.006 | 1962-07 | 0.003106 | -0.019 | 4 | 432 |
| **43** | 10495 | 1962-08-31 | A | 40.875 | 0.003 | 1962-08 | 0.024768 | -0.011 | 4 | 432 |
| **85** | 10495 | 1962-10-31 | A | 38.250 | 0.000 | 1962-10 | 0.033784 | 0.028 | 7 | 396 |
| **105** | 10495 | 1962-11-30 | A | 41.750 | -0.006 | 1962-11 | 0.117647 | -0.030 | 3 | 409 |
| **125** | 10495 | 1962-12-31 | A | 40.500 | 0.003 | 1962-12 | -0.029940 | 0.021 | 6 | 447 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **59173686** | 91205 | 2008-04-30 | ZZ | 6.100 | -0.030 | 2008-04 | -0.197368 | 0.041 | 8 | 691 |
| **59173728** | 91205 | 2008-06-30 | ZZ | 5.740 | -0.039 | 2008-06 | -0.077170 | 0.018 | 6 | 565 |
| **59173750** | 91205 | 2008-07-31 | ZZ | 6.830 | 0.012 | 2008-07 | 0.189896 | 0.157 | High | 522 |
| **59173792** | 91205 | 2008-09-30 | ZZ | 6.460 | -0.060 | 2008-09 | -0.034380 | -0.026 | 3 | 608 |
| **59173815** | 91205 | 2008-10-31 | ZZ | 3.230 | -0.021 | 2008-10 | -0.500000 | -0.147 | Low | 593 |

1976407 rows × 17 columns

In [ ]:

```
# find inf values in REV and replace them with NaN
crsp_monthly['REV'].replace([np.inf, -np.inf], np.nan, inplace=True)
```

In [ ]:

```
crsp_monthly.groupby(['Decile'])['REV'].mean()
```

Out[ ]:

```
Decile
Low     -0.865
2       -0.865
3       -1.062
4       -0.797
5       -1.054
6       -0.910
7       -0.976
8       -0.905
9       -1.086
High    -0.224
Name: REV, dtype: float64
```

In [ ]:

```
del crsp_monthly
```

# MAX(MIN)

In [ ]:
```
#crsp.to_csv("C:/Users/Aman/Downloads/Compressed/crsp_filtered_merged_illiq_skinny_
```

In [ ]:
```
crsp
```

Out[ ]:

| | PERMNO | date | TICKER | PRC_x | RET_x | month | RET_y | ST | Decile | |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 10495 | 1962-07-02 | A | 41.125 | 0.022 | 1962-07 | NaN | -0.019 | 4 | |
| **1** | 10495 | 1962-07-03 | A | 41.375 | 0.006 | 1962-07 | NaN | -0.019 | 4 | |
| **2** | 10495 | 1962-07-05 | A | 41.250 | -0.003 | 1962-07 | NaN | -0.019 | 4 | |
| **3** | 10495 | 1962-07-06 | A | 40.500 | -0.018 | 1962-07 | NaN | -0.019 | 4 | |
| **4** | 10495 | 1962-07-09 | A | 40.750 | 0.006 | 1962-07 | NaN | -0.019 | 4 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **59173811** | 91205 | 2008-10-27 | ZZ | 2.710 | -0.029 | 2008-10 | NaN | -0.147 | Low | |
| **59173812** | 91205 | 2008-10-28 | ZZ | 2.750 | 0.015 | 2008-10 | NaN | -0.147 | Low | |
| **59173813** | 91205 | 2008-10-29 | ZZ | 3.000 | 0.091 | 2008-10 | NaN | -0.147 | Low | |
| **59173814** | 91205 | 2008-10-30 | ZZ | 3.300 | 0.100 | 2008-10 | NaN | -0.147 | Low | |
| **59173815** | 91205 | 2008-10-31 | ZZ | 3.230 | -0.021 | 2008-10 | -0.500000 | -0.147 | Low | 593( |

59173816 rows × 15 columns

In [ ]:
```
crsp.drop(columns=['ME','BE','BM','ILLIQ_monthly','BETA','IVOL'], inplace=True)
```

In [ ]:
```
crsp
```

Out[ ]:

| | PERMNO | date | TICKER | PRC_x | RET_x | month | RET_y | ST | Decile |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 10495 | 1962-07-02 | A | 41.125 | 0.022 | 1962-07 | NaN | -0.019 | 4 |
| **1** | 10495 | 1962-07-03 | A | 41.375 | 0.006 | 1962-07 | NaN | -0.019 | 4 |
| **2** | 10495 | 1962-07-05 | A | 41.250 | -0.003 | 1962-07 | NaN | -0.019 | 4 |
| **3** | 10495 | 1962-07-06 | A | 40.500 | -0.018 | 1962-07 | NaN | -0.019 | 4 |
| **4** | 10495 | 1962-07-09 | A | 40.750 | 0.006 | 1962-07 | NaN | -0.019 | 4 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **59173811** | 91205 | 2008-10-27 | ZZ | 2.710 | -0.029 | 2008-10 | NaN | -0.147 | Low |
| **59173812** | 91205 | 2008-10-28 | ZZ | 2.750 | 0.015 | 2008-10 | NaN | -0.147 | Low |
| **59173813** | 91205 | 2008-10-29 | ZZ | 3.000 | 0.091 | 2008-10 | NaN | -0.147 | Low |
| **59173814** | 91205 | 2008-10-30 | ZZ | 3.300 | 0.100 | 2008-10 | NaN | -0.147 | Low |
| **59173815** | 91205 | 2008-10-31 | ZZ | 3.230 | -0.021 | 2008-10 | -0.500000 | -0.147 | Low |

59173816 rows × 9 columns

In [ ]:

```
#BETA is the market beta, estimated from a regression of daily excess stock returns
count=0
for permno in crsp['PERMNO'].unique()[:100]:
    crsp_sample = crsp[crsp['PERMNO'] == permno].copy()

    # Group by ticker and month and iterate over each group
    for name, group in crsp_sample.groupby(['TICKER', 'month']):
        min_ret = group['RET_x'].min()
        max_ret = group['RET_x'].max()
        crsp_sample.loc[group.index, 'RET_min'] = min_ret
        crsp_sample.loc[group.index, 'RET_max'] = max_ret

    #Make the index of crsp_sample same as crsp['PERMNO'] == permno
    crsp_sample.set_index(crsp[crsp['PERMNO'] == permno].index, inplace=True)

    # Add the 'RET_min' column to the original DataFrame
    crsp.loc[crsp[crsp['PERMNO'] == permno].index, 'RET_min'] = crsp_sample['RET_mi

    # Add the 'RET_max' column to the original DataFrame
    crsp.loc[crsp[crsp['PERMNO'] == permno].index, 'RET_max'] = crsp_sample['RET_ma
```

```
    count+=1
    if count%100 == 0:
        print("Processed PERMNO: ", count)
```

```
Processed PERMNO:  100
```

In [ ]:
```
crsp[crsp['date'].dt.is_month_end].groupby(['Decile'])['RET_max'].mean()*100
```

Out[ ]:
```
Decile
Low      5.700
2        4.144
3        3.394
4        3.082
5        2.398
6        3.232
7        4.176
8        5.587
9        7.543
High    13.313
Name: RET_max, dtype: float64
```

In [ ]:
```
crsp[crsp['date'].dt.is_month_end].groupby(['Decile'])['RET_min'].mean()*100
```

Out[ ]:
```
Decile
Low    -10.149
2       -5.405
3       -3.885
4       -3.078
5       -2.351
6       -2.885
7       -3.087
8       -3.881
9       -4.595
High    -6.061
Name: RET_min, dtype: float64
```

# TK

Cannot calculate TK as we do not have access to Barberis et al. (2016)

# SKEW

In [ ]:
```
# Calculate SKEW for RET_x (daily returns) for 1 year window for each PERMNO
crsp['SKEW'] = crsp.groupby(['PERMNO'])[ 'RET_x'].transform(lambda x: x.rolling(252
crsp['SKEW'].fillna(method='bfill', inplace=True)
crsp['SKEW'].fillna(method='ffill', inplace=True)
crsp
```

Out[ ]:

| | PERMNO | date | TICKER | PRC_x | RET_x | month | RET_y | ST | Decile | RET_ |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 10495 | 1962-07-02 | A | 41.125 | 0.022 | 1962-07 | NaN | -0.019 | 4 | -( |
| **1** | 10495 | 1962-07-03 | A | 41.375 | 0.006 | 1962-07 | NaN | -0.019 | 4 | -( |
| **2** | 10495 | 1962-07-05 | A | 41.250 | -0.003 | 1962-07 | NaN | -0.019 | 4 | -( |
| **3** | 10495 | 1962-07-06 | A | 40.500 | -0.018 | 1962-07 | NaN | -0.019 | 4 | -( |
| **4** | 10495 | 1962-07-09 | A | 40.750 | 0.006 | 1962-07 | NaN | -0.019 | 4 | -( |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **59173811** | 91205 | 2008-10-27 | ZZ | 2.710 | -0.029 | 2008-10 | NaN | -0.147 | Low | |
| **59173812** | 91205 | 2008-10-28 | ZZ | 2.750 | 0.015 | 2008-10 | NaN | -0.147 | Low | |
| **59173813** | 91205 | 2008-10-29 | ZZ | 3.000 | 0.091 | 2008-10 | NaN | -0.147 | Low | |
| **59173814** | 91205 | 2008-10-30 | ZZ | 3.300 | 0.100 | 2008-10 | NaN | -0.147 | Low | |
| **59173815** | 91205 | 2008-10-31 | ZZ | 3.230 | -0.021 | 2008-10 | -0.500000 | -0.147 | Low | |

59173816 rows × 12 columns

In [ ]:

```
crsp.groupby(['Decile'])['SKEW'].mean()
```

Out[ ]:

```
Decile
Low     0.222
2       0.329
3       0.322
4       0.301
5       0.326
6       0.311
7       0.359
8       0.403
9       0.461
High    0.669
Name: SKEW, dtype: float64
```

# COSKEW

---

## The above statistics as a whole take more than half a day and over 32 gigs of RAM to run. We could have achieved better accuracy and coverage with a smaller sample set, but we wanted to be as close to the paper as possible. We have tried to run the code on a 64 gig RAM machine, but it still takes a long time.

## 3.

From Tables 3-10, choose two other tables and replicate them.

---

4. If the numbers you obtain in questions 2 and 3 deviate from those in the paper, why do you think this is? What parts of the data construction and replication were difficult? Was there any additional information the authors could have given you to make this process simpler

The numbers obtained in question 2 deviated a little bit from the paper. In the construction of the dataset, we removed the stocks which had a close price at the end of the previous month of $5 or less, if the previous close was not available, the bid-ask was used. This was done to minimize market microstructure effects. Depending on how the authors constructed their dataset, we may have ended up with more or less data depending on how aggressively the authors applied this filter, and the completeness of their dataset. Additionally, for stocks with fewer than 15 observations available in a month, that month's data was removed. If the authors had more complete data, or had some other method of filling in these gaps, then this would help explain the different figures. Because of the potential correlation between the quality of the stock and the availability of the data, it does not seem reasonable to assume that the removed observations would be normally distributed and so has the potential to bias the results for every figure in the table.

In cases we had to use the book value of equity at the end of the previous year, the first year of observations would not have a book value. Our assumption was that last year's book value would be roughly equal to the next year's book value, and used backfilling to gain back the first year of available data. If the authors had a different assumption, this may explain why the numbers are slightly different. For instance, in cases where 1 year of data is available, that book value of equity would not have contributed to the overall averages by decile of the BM ratio. Other assumptions regarding missing values would have had similar effects, which the authors did not discuss in detail. If the authors were more clear on how they treated this "first year" effect and how they may have used backfilling or forward filling, this would have enabled closer replication.

Other possible points:

- construction of ILLIQ measure, how did they consider stocks which had no return? How did they scale it to avoid -inf errors. Did they use the return data from CRSP, or construct their own?
- Sample construction flows through all the different parts
- Selection of controls for regressions / Betas

---

5. In your view, what are the key takeaways of this paper? How did the results in the tables you replicated contribute to the paper as a whole?

The key question investigated by the paper was whether investors experience "salience" when they make their investment decisions, whereby they pay attention only to the most extreme pieces of information, foregoing comprehensive analysis. The key takeaways from the paper was that there is evidence that investors make salient decisions that may violate traditional market assumptions regarding efficiency.

Additionally, this salience effect is different depending on the type of stock traded, where stocks that are less able to be shorted and thus arbitrage is less easy to correct, the salience prevalence is stronger. The authors hypothesise that this is because in series easier to short and trade rational investors who make decisions based on objective probabilities will buy and sell the asset and correct the prices. This is what Table 5 showed, where the salience effect is strongest in equities that are more difficult to correct arbitrage. The authors identify that salience is strongest among small, illiquid stocks with little professional investing and little coverage by the market.

Another key point raised by the authors is whether the salience effect is driven by market inefficiencies or by behavioural reasons. In Table 9, the authors demonstrate that unlike the reversal effect, which has weakened over time due to market efficiency rising over time, the salience effect has not changed. This provides evidence that there are stronger behavioural effects driving salience and its effect is not simply being mistaken for market structure inefficiency.

Overall, the paper provides a compelling argument that not only do a significant proportion of investors exhibit the salience effect, but also that this effect is consistent across different time periods and a wide range of controls. For investors, this suggests that there may be significant mispricing in stocks more likely to be affected by the salience effect, such as for illiquid stocks with recent negative news.