Factor-based Analysis of Mutual Fund Performance

Data Bootcamp

Spring 2021 Final Porject

Ruiyu Wang, Veronica Hu

Introduction

1 Capital Asset Pricing Model

$$R_i - R_f = \alpha_i + \beta(R_M - R_f)$$

- A classical model to compute the expected return of an asset, based on risk free rate and market risk premium.

2 Fama French Three-Factor Model

- Expands on CAPM by adding **size risk** and **value risk** factors to the market risk factors
- Considers the fact that **value and small-cap stocks outperform markets on a regular basis**

Three factors:

- **Excess return on the market**: i.e. the market return less the risk free rate of return
- **Size of firms:** meausred by SMB (small minus big)
  It accounts for publicly traded companies with small market caps that generate higher returns.
  Below is the visualized performance of **Size factor** from Bloomberg, which also suggests that smaller size tends to outperform bigger size:

- **Book-to-market values:** measured by HML (high minus low)
  It accounts for value stocks with high book-to-market ratios that generate higher returns in comparison to

the market.

Below is the visualized performance of **Value factor** from Bloomberg, which also suggests that higher value tends to outperform lower value:

Formula:

$$R_i - R_f = \alpha_i + \beta_1(R_M - R_f) + \beta_2 SMB + \beta_3 HML + \epsilon_i$$

Where:

R_i = total return of a stock or portfolio *i*

R_f = risk free return

$R_i - R_f$ = expected excess return

R_M = total market portfolio return

$R_M - R_f$ = market risk premium

SMB = size premium

HML = value premium

$\beta_{1,2,3}$ = factor coefficients

3 Our project inspiration and plan:

We want to explore and verify the impacts of three factors by running regression over FF3 model using historical data of U.S. mutual fund returns. Furthermore, we want to infer the investment style of the funds based on the factor coefficetients and suggest a potential valid way of classifying differennt categories of mutual funds.

Data Cleaning and Preprocessing

1 Three Factors Dataset (Kenneth French)

Souce: Kenneth R. French's public online data library (https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html).

It contains historical monthly values of the SMB, HML, market return and risk-free return from 1926/7 to 2021/2.

In [1]:

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib as mpl
import seaborn as sb
```

In [2]:

```
ff3 = pd.read_csv('factors.csv', index_col=0)
ff3 = ff3/100 # since the original values are in percentage, devide by 100 to get numeric v
ff3.index= pd.to_datetime(ff3.index, yearfirst=True, format='%Y%m')
```

In [3]:

```
ff3.head()
```

Out[3]:

	Mkt-RF	SMB	HML	RF
1926-07-01	0.0296	-0.0230	-0.0287	0.0022
1926-08-01	0.0264	-0.0140	0.0419	0.0025
1926-09-01	0.0036	-0.0132	0.0001	0.0023
1926-10-01	-0.0324	0.0004	0.0051	0.0032
1926-11-01	0.0253	-0.0020	-0.0035	0.0031

In [4]:

```
factors = ff3.iloc[:, [0, 1, 2]]
```

2 25 Portfolios Dataset (Kenneth French)

Source: same as previous one
It contains historical monthly return of 25 different portfolios from 1926/7 to 2021/2.
The portfolios are sorted and measured by different degrees of ME and BM. E.g. SMALL LoBM means the portfolio contains small size and low book-to-market ratio stocks.

In [5]:

```
ptf = pd.read_csv('portfolio.csv', index_col=0)
ptf = ptf/100
ptf.index= pd.to_datetime(ptf.index, yearfirst=True, format='%Y%m')
```

In [6]:

```
p25 = ptf.sub(ff3['RF'], axis=0)
p25.head()
```

Out[6]:

	SMALL LoBM	ME1 BM2	ME1 BM3	ME1 BM4	SMALL HiBM	ME2 BM1	ME2 BM2	ME2 BM3	M
1926-07-01	0.035582	-0.006319	-0.021634	0.001330	0.018334	0.019704	0.021992	0.002726	-0
1926-08-01	-0.024574	-0.089775	0.021904	0.003586	0.081468	0.019209	-0.014349	0.037584	0
1926-09-01	-0.064413	-0.005289	-0.064282	-0.018668	0.006349	-0.020850	-0.014918	0.008529	-0
1926-10-01	-0.089441	-0.040732	-0.059919	0.053970	-0.028676	-0.021195	-0.035863	-0.053945	-0
1926-11-01	0.031644	0.063376	0.019534	-0.050120	0.002262	0.025951	-0.026790	0.026978	0

5 rows × 25 columns

3 U.S. Mutual Funds

Source: WRDS

In [7]:

```
df_mf = pd.read_csv('mutual fund data.csv')
df_mf.head()
```

```
C:\Users\Jane\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3
146: DtypeWarning: Columns (3,5) have mixed types.Specify dtype option on im
port or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

Out[7]:

	ticker	caldt	crsp_fundno	mtna	mret	mnav
0	AABIX	19970829	1	46.629	-0.010477	15.0700000
1	AABIX	19970930	1	57.900	0.014313	15.2000000
2	AABIX	19971128	1	75.033	0.006949	15.2600000
3	AABIX	19970530	1	26.157	0.013523	14.8700000
4	AABIX	19970228	1	3.926	R	14.9800000

Dataset description:

- ticker

- caldt: date of row
- crsp_fundno: a unique fund number
- mtna: total net asset value
- mret: return per share
- mnav: net asset value per share

In [8]:

```
# transform 'caldt' into date values
df_mf['year'] = pd.to_datetime(df_mf['caldt'].astype('str')).dt.year
df_mf['month'] = pd.to_datetime(df_mf['caldt'].astype('str')).dt.month
df_mf['date'] = pd.to_datetime(df_mf['year'].astype('str')+'-'+df_mf['month'].astype('str')
```

In [9]:

```
# Since some of the returns are uncollectable (filled with 'R'), we drop the corresponding
df_mf['mret'] = pd.to_numeric(df_mf['mret'], errors='coerce')
df_mf.dropna(axis = 0, subset = ['mret'], inplace=True)
```

Regression Analysis

1 Regression on 25 Portfolios

In [10]:

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib as mpl
import seaborn as sb
```

In [11]:

```
start = '1970-1'
end = '2020-12'
X = factors[start:end]
y = p25[start:end]
```

In [12]:

```
X = sm.add_constant(X)
```

In [13]:

```
%%time
reg = sm.OLS(y, X).fit()
params = reg.params
params = params.transpose()
```

```
Wall time: 22 ms
```

In [14]:

```
# Label the 25 porfolios according to their degress of ME (size) and BM (value)
params['ME'] = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5]
params['BM'] = [1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
params
```

10	-0.000642	1.113020	0.749742	-0.397463	3	1
11	0.000900	1.027625	0.592411	0.148790	3	2
12	-0.000040	0.983709	0.424503	0.419216	3	3
13	0.000447	0.987596	0.423913	0.613741	3	4
14	0.000546	1.086580	0.563674	0.824808	3	5
15	0.001054	1.077703	0.409935	-0.393261	4	1
16	-0.000278	1.077309	0.222021	0.203096	4	2
17	-0.000277	1.037715	0.167077	0.432522	4	3
18	0.000119	1.028841	0.195000	0.580909	4	4
19	-0.000875	1.162543	0.264285	0.816818	4	5
20	0.001332	0.975050	-0.244951	-0.354433	5	1
21	0.000467	0.980890	-0.191877	0.075393	5	2
22	0.000335	0.951572	-0.242035	0.279352	5	3

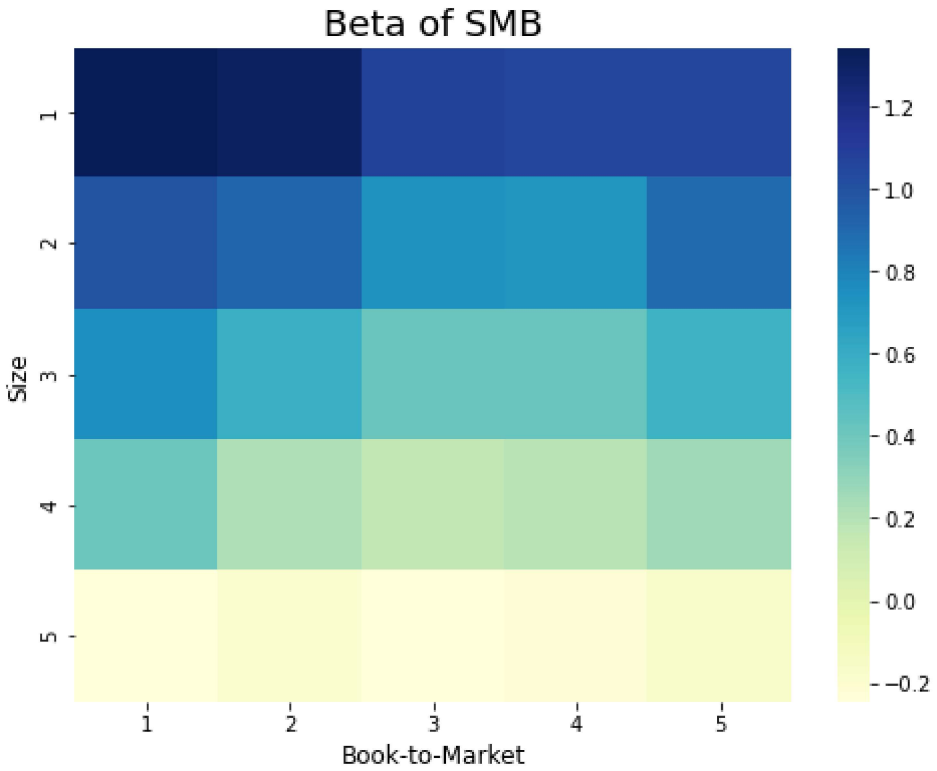
Visulize the Performance of Size Factor (SMB)

In [15]:

```
fig, ax = plt.subplots(figsize=(8, 6))
sb.heatmap(params.pivot(index='ME', columns='BM', values='SMB'), cmap='YlGnBu')
ax.set_title('Beta of SMB', size=18)
ax.set_ylabel('Size', size=12)
ax.set_xlabel('Book-to-Market', size=12)
```

Out[15]:

```
Text(0.5, 33.0, 'Book-to-Market')
```



Conclusion:

- The smaller the Size of a portfolio, the higher the beta of SMB

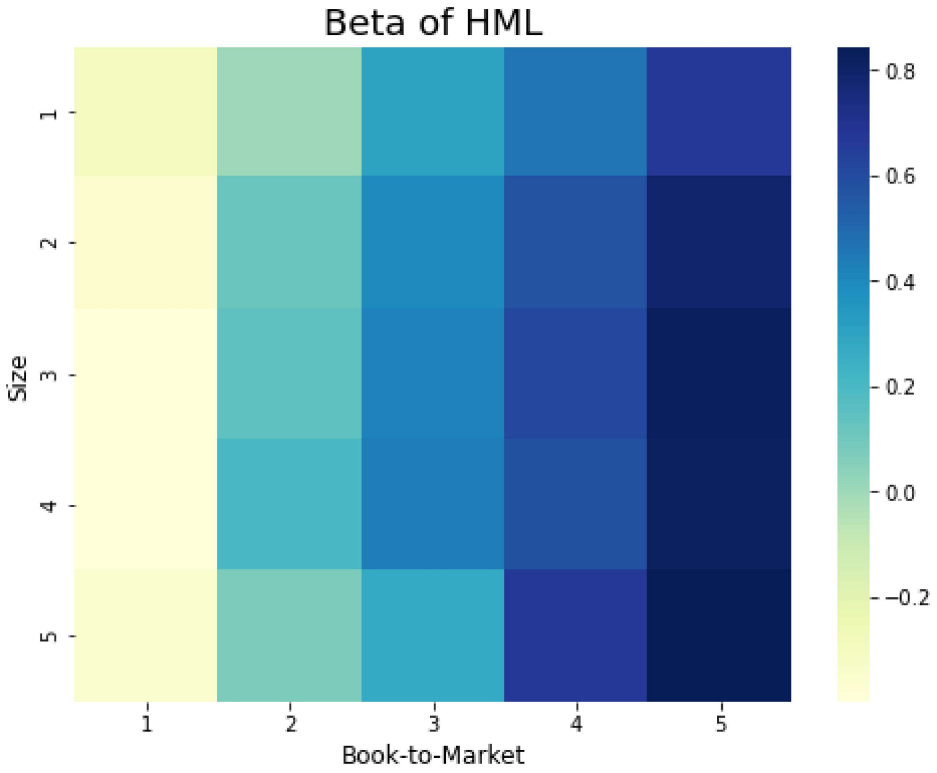
Visulize the Performance of Value Factor (HML)

In [16]:

```
fig, ax = plt.subplots(figsize=(8,6))
sb.heatmap(params.pivot(index='ME', columns='BM', values='HML'), cmap='YlGnBu')
ax.set_title('Beta of HML', size=18)
ax.set_ylabel('Size', size=12)
ax.set_xlabel('Book-to-Market', size=12)
```

Out[16]:

```
Text(0.5, 33.0, 'Book-to-Market')
```



Conclusion:

- The higher the Book-to-Market ratio of a portfolio, the higher the beta of HML

2 Regression on U.S. Mutual Funds

In [17]:

```
def regress_beta(n):
    mf = df_mf.loc[df_mf['crsp_fundno']==n, :].sort_values('date').set_index('date')
    mf['return'] = pd.to_numeric(mf['mret'], errors='coerce')
    df = mf[['return']].join(ff3).dropna()
    df['R-RF'] = df['return'].sub(df['RF'], axis=0)
    df = sm.add_constant(df)
    X = df.iloc[:, [0, 2, 3, 4]]
    y = df.iloc[:, [6]]
    reg = sm.OLS(y, X).fit()
    score = reg.rsquared
    beta = pd.DataFrame(reg.params)
    beta.columns = [n]
    return beta, score
```

In [18]:

```
counts = df_mf.groupby('crsp_fundno').agg({'date':'count'})
counts.describe()
```

Out[18]:

	date
count	68686.000000
mean	117.196940
std	98.108308
min	1.000000
25%	43.000000
50%	91.000000
75%	160.000000
max	711.000000

- 25% mutual funds have records less than 43 rows
- For regression results to be significant, we set the threshold number of rows to be 60 (5 years) for each mutual funds

In [19]:

```
fund = counts.loc[counts['date']>60, :].index.tolist()
```

In [20]:

```
print('Number of mutual funds:', len(fund))
```

```
Number of mutual funds: 44581
```

- Due to large size of dataset, we fit regression models with subsets instead of running the entire dataset in a single loop

In [71]:

```
%%time
subset1 = list(np.array_split(fund, 8)[0])
beta = regress_beta(subset1[0])[0]
scores = [regress_beta(subset1[0])[1]]
for i in range(1, len(subset1)):
    beta_t, score = regress_beta(subset1[i])
    beta = beta.join(beta_t)
    scores.append(score)
    print(subset1[i], score)
beta_subset1 = beta.transpose()
beta_subset1['scores'] = scores
```

...

In [72]:

```
subset2 = list(np.array_split(fund, 8)[1])
beta = regress_beta(subset2[0])[0]
scores = [regress_beta(subset2[0])[1]]
for i in range(1, len(subset2)):
    beta_t, score = regress_beta(subset2[i])
    beta = beta.join(beta_t)
    scores.append(score)
    print(subset2[i], score)
beta_subset2 = beta.transpose()
beta_subset2['scores'] = scores
```

...

In [75]:

```
subset3 = list(np.array_split(fund, 8)[2])
beta = regress_beta(subset3[0])[0]
scores = [regress_beta(subset3[0])[1]]
for i in range(1, len(subset3)):
    beta_t, score = regress_beta(subset3[i])
    beta = beta.join(beta_t)
    scores.append(score)
    print(subset3[i], score)
beta_subset3 = beta.transpose()
beta_subset3['scores'] = scores
```

```
14980 0.9571790570485275
14981 0.9389609115267159
14982 0.9186855609415473
14983 0.9082137040396866
14985 0.4056150806187422
14986 0.7459278638876237
14987 0.06506464639644782
14988 0.9677916545295829
14989 0.9632060460892059
14990 0.933709152790967
14991 0.06020889419600062
14992 0.8671882582331016
14993 0.9314023527763763
14994 0.01475051192136867
14995 0.9051586534990157
14996 0.4896952032204299
14997 0.8672604101389564
14998 0.9344985441652841
14999 0.5902945145418219
```

In [76]:

```
subset4 = list(np.array_split(fund, 8)[3])
beta = regress_beta(subset4[0])[0]
scores = [regress_beta(subset4[0])[1]]
for i in range(1, len(subset4)):
    beta_t, score = regress_beta(subset4[i])
    beta = beta.join(beta_t)
    scores.append(score)
    print(subset4[i], score)
beta_subset4 = beta.transpose()
beta_subset4['scores'] = scores
```

...

In [77]:

```
subset5 = list(np.array_split(fund, 8)[4])
beta = regress_beta(subset5[0])[0]
scores = [regress_beta(subset5[0])[1]]
for i in range(1, len(subset5)):
    beta_t, score = regress_beta(subset5[i])
    beta = beta.join(beta_t)
    scores.append(score)
    print(subset5[i], score)
beta_subset5 = beta.transpose()
beta_subset5['scores'] = scores
```

...

In [78]:

```
subset6 = list(np.array_split(fund, 8)[4])
beta = regress_beta(subset6[0])[0]
scores = [regress_beta(subset6[0])[1]]
for i in range(1, len(subset6)):
    beta_t, score = regress_beta(subset6[i])
    beta = beta.join(beta_t)
    scores.append(score)
    print(subset6[i], score)
beta_subset6 = beta.transpose()
beta_subset6['scores'] = scores
```

...

In [79]:

```
subset7 = list(np.array_split(fund, 8)[4])
beta = regress_beta(subset7[0])[0]
scores = [regress_beta(subset7[0])[1]]
for i in range(1, len(subset7)):
    beta_t, score = regress_beta(subset7[i])
    beta = beta.join(beta_t)
    scores.append(score)
    print(subset7[i], score)
beta_subset7 = beta.transpose()
beta_subset7['scores'] = scores
```

...

In [80]:

```
subset8 = list(np.array_split(fund, 8)[7])
beta = regress_beta(subset8[0])[0]
scores = [regress_beta(subset8[0])[1]]
for i in range(1, len(subset8)):
    beta_t, score = regress_beta(subset8[i])
    beta = beta.join(beta_t)
    scores.append(score)
    print(subset8[i], score)
beta_subset8 = beta.transpose()
beta_subset8['scores'] = scores
```

...

In [88]:

```
beta_all = pd.DataFrame()
```

In [89]:

```
beta_all = beta_all.append(beta_subset1)
beta_all = beta_all.append(beta_subset2)
beta_all = beta_all.append(beta_subset3)
beta_all = beta_all.append(beta_subset4)
beta_all = beta_all.append(beta_subset5)
beta_all = beta_all.append(beta_subset6)
beta_all = beta_all.append(beta_subset7)
beta_all = beta_all.append(beta_subset8)
beta_all
```

Out[89]:

	const	Mkt-RF	SMB	HML	scores
3	-0.002804	0.058050	-0.041801	-0.075091	0.028372
8	-0.000401	0.596206	-0.056313	-0.122313	0.755791
9	0.000272	0.581427	-0.046686	-0.153696	0.768596
10	-0.003252	0.563248	-0.023553	-0.020911	0.822673
11	0.000811	0.304492	-0.141316	0.212634	0.438649
...	...	...	...	...	...
96335	-0.001005	0.957397	0.035685	0.400172	0.956040
96524	-0.001761	0.912500	0.492713	0.298988	0.897553
96534	-0.002754	0.549313	0.010626	0.082657	0.791524
96645	-0.000443	0.000755	0.000636	-0.002563	0.024291
98235	0.006316	0.665935	0.317223	-0.141083	0.769365

44583 rows × 5 columns

In [21]:

```
# beta_all.to_csv('beta_all.csv')
# beta_all = pd.read_csv('beta_all.csv')
# beta_all['scores'] = pd.to_numeric(beta_all['scores'])
```

3 Analysis on regression results

In [22]:

```
beta_all.describe()
```

Out[22]:

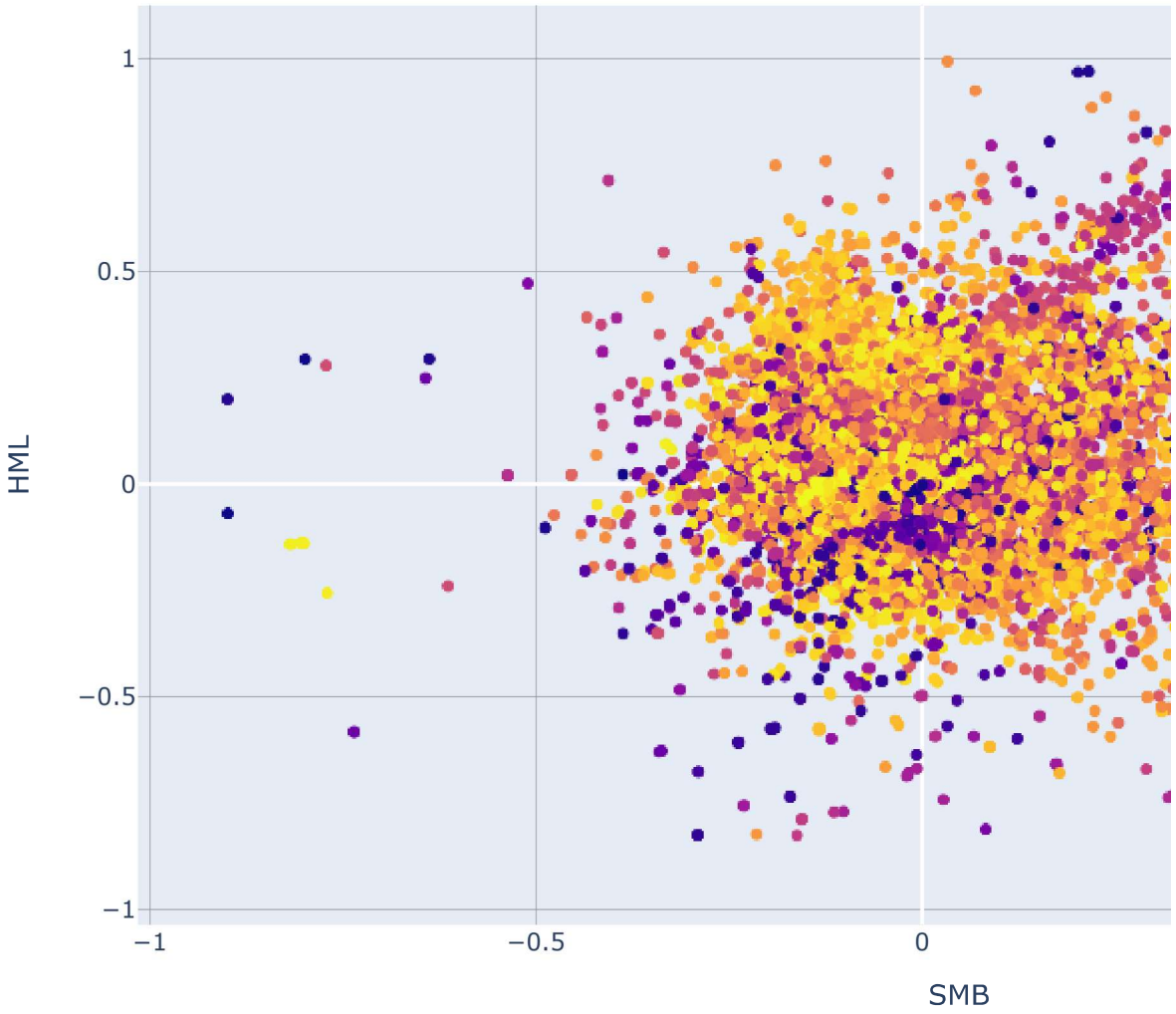
	Unnamed: 0	const	Mkt-RF	SMB	HML	scores
count	44583.000000	44583.000000	44583.000000	44583.000000	44583.000000	44583.000000
mean	28100.445641	-0.000614	0.625509	0.069636	0.016348	0.588891
std	17838.724707	0.058158	0.474831	0.440654	0.325295	0.360973
min	3.000000	-0.053708	-13.651503	-75.760371	-34.819522	0.000019
25%	14810.500000	-0.002021	0.121513	-0.052992	-0.057140	0.169085
50%	28718.000000	-0.000614	0.779892	0.000214	0.003598	0.759981
75%	36060.500000	0.000531	0.993183	0.105551	0.099231	0.903817
max	98235.000000	12.159674	4.748766	4.455806	4.090151	0.999179

In [23]:

```
# get rid of outliers
beta_norm = beta_all.loc[(beta_all['Mkt-RF']<=1)&(beta_all['Mkt-RF']>=-1)&(beta_all['SMB']<
```

In [24]:

```
import plotly.express as px
fig = px.scatter(beta_norm, x='SMB', y='HML', color='scores', width=1000, height=600)
fig.show()
```



- We can see that the scatter plot cannot provide very useful information about the mutual funds given their betas of SMB and HML
- We want to further look at the some subsets of the mutual funds and categorize them based on their betas and scores

a. We categorize mutual funds into 2 genres: Fixed Income Fund or Stock Fund based on their R-squared, since FF3 Model is more compatible to predict equity asset value

In [25]:

```
beta_norm['category']=['' for i in range(beta_norm.shape[0])]
```

```
<ipython-input-25-ae96296dabac>:1: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
```

In [26]:

```
beta_norm.loc[beta_norm['scores'] < 0.4, ['category']] = "Fixed Income"
beta_norm.loc[beta_norm['scores'] > 0.4, ['category']] = "Stock"
beta_norm['category'].value_counts()
```

```
C:\Users\Jane\anaconda3\lib\site-packages\pandas\core\indexing.py:1765: Sett
ingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)

C:\Users\Jane\anaconda3\lib\site-packages\pandas\core\indexing.py:1765: Sett
ingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
```

Out[26]:

```
Stock          19896
Fixed Income    13901
Name: category, dtype: int64
```

In [27]:

```
stockfund_beta = beta_norm.loc[beta_norm['category'] == "Stock"]
```

b. Among Stock Funds, further categorize them into 4 types based on the betas of SML and HML

- For those with HML betas < 0, we label them as "Growth Stock"; otherwise, they are "Value Stock"
- For those with SML betas < 0, we label them as "Large Cap Stock"; otherwise, they are "Small Cap Stock"

In [28]:

```
stockfund_beta.loc[(stockfund_beta['SMB'] > 0) & (stockfund_beta['HML'] > 0), ['category']]
stockfund_beta.loc[(stockfund_beta['SMB'] > 0) & (stockfund_beta['HML'] < 0), ['category']]
stockfund_beta.loc[(stockfund_beta['SMB'] < 0) & (stockfund_beta['HML'] > 0), ['category']]
stockfund_beta.loc[(stockfund_beta['SMB'] < 0) & (stockfund_beta['HML'] < 0), ['category']]
stockfund_beta['category'].value_counts()
```

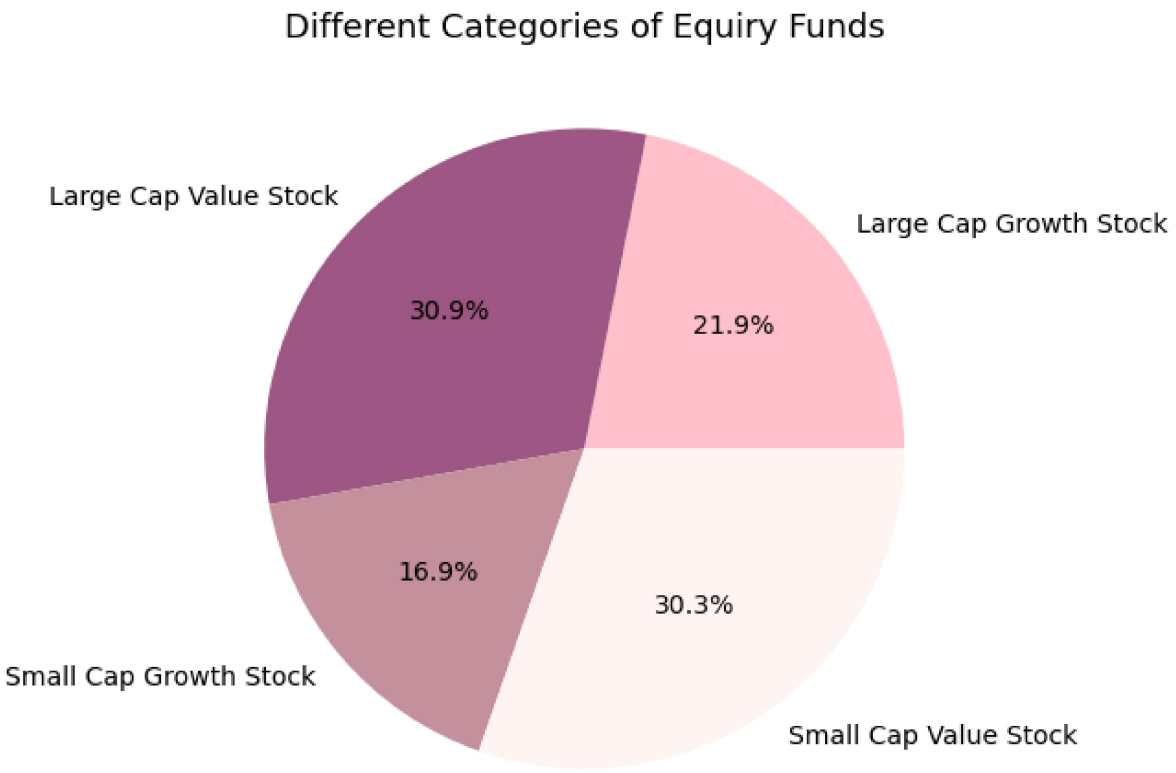
```
C:\Users\Jane\anaconda3\lib\site-packages\pandas\core\indexing.py:1765: S
ettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)

C:\Users\Jane\anaconda3\lib\site-packages\pandas\core\indexing.py:1765: S
ettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

In [29]:

```
fig, ax = plt.subplots()
plt.style.use('fivethirtyeight')
colors = ['pink','#9d5783','#c3909b', '#fff4f2']
stockfund_beta.groupby('category')['SMB'].count().plot.pie(colors = colors, autopct = '%1.1
ax.set_title('Different Categories of Equiry Funds', size=18)
ax.set_ylabel('')
# labels = ["Small Cap Value Stock", "Small Cap Growth Stock", "Large Cap Value Stock", "La
#plt.pie(stockfund_beta['category'].value_counts(), labels = labels, colors = colors, autop
```

Out[29]:

```
Text(0, 0.5, '')
```



c.

In [30]:

```
mf_list = df_mf.loc[df_mf["crsp_fundno"].isin(fund)]
mf_list['mtna'] = pd.to_numeric(mf_list['mtna'], errors='coerce')
```

```
<ipython-input-30-9d0a4689a878>:2: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
```

In [31]:

```
NAV= mf_list.groupby('crsp_fundno').agg({'mtna':'mean'})
Return = mf_list.groupby('crsp_fundno').agg({'mret':'mean'})
stockfund_beta['NAV'] = NAV
stockfund_beta['Return'] = Return*12
```

```
<ipython-input-31-71a92e8e783d>:3: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)

<ipython-input-31-71a92e8e783d>:4: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
```

In [38]:

```
#fig = px.scatter(stockfund_beta, x='Return', y='NAV', color='category')
#size = average asset value
#fig.show()
```

Unsupervised Learning: KMeans

- In the previous section, we categorize mutual funds by manually setting thresholds of parameters
- We want to validate the heuristic method we have used, and explore if there exists more effective classifying method using Machine Learning

a. Select the optimal k (i.e. number of clusters/categories)

In [33]:

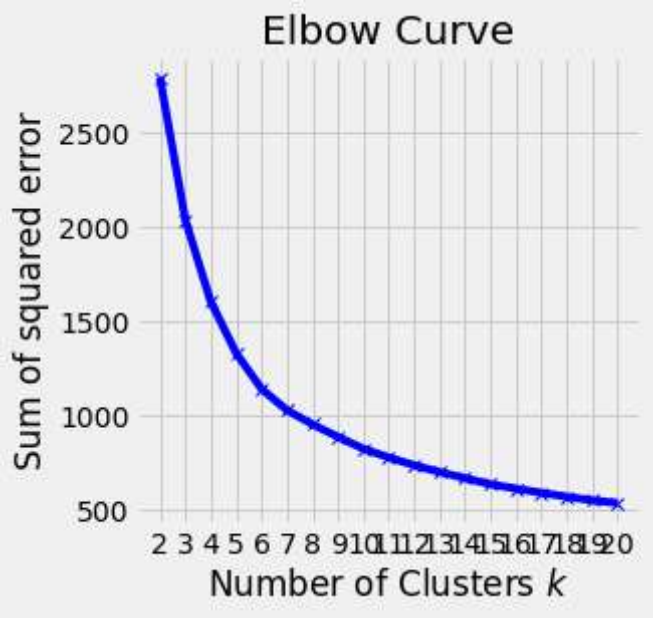
```
from sklearn.cluster import KMeans
```

In [34]:

```
n_clusters = range(2, 21)
def kmeans(k):
    model = KMeans(k)
    X = beta_norm.iloc[:, [2,3,4]]  ## the features used: beta of SMB, beta of HML, scores
    model.fit(X)
    y_kmeans = model.predict(X)
    return model.inertia_
```

In [35]:

```
scores  = [kmeans(k) for k in n_clusters]
plt.figure(figsize=(4,4))
plt.plot(n_clusters,scores, 'bx-')
plt.xticks(range(2,21))
plt.xlabel('Number of Clusters $k$');
plt.ylabel('Sum of squared error');
plt.title('Elbow Curve');
```



- From the above curve, we could see that roughly at **k = 6**, the sum of squared error achieves an elbow point
- Hence we pick k = 6 as an optimal parameter of the model

b. Fit the model and compare results

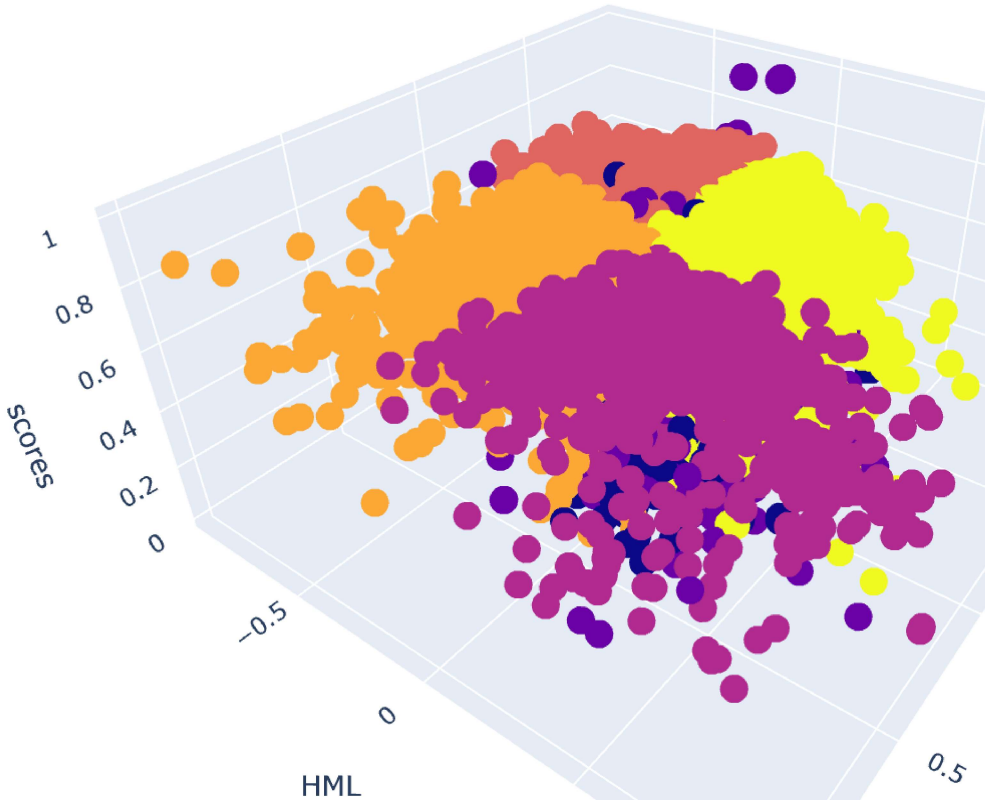
In [36]:

```
model = KMeans(6)
X = beta_norm.iloc[:, [2,3,4]]
model.fit(X)
y_kmeans = model.predict(X)
beta_norm['cat_pred'] = y_kmeans
```

```
<ipython-input-36-871f4284954a>:5: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
```

In [37]:

```
fig = px.scatter_3d(beta_norm, x='SMB', y='HML', z='scores', color='cat_pred', width=900, h
fig.show()
```



Conclusion:

- From the plot, we can see that the KMeans model categorize the mutual funds into 6 clustors in a similar pattern as we did previously
  - For funds with scores lower than (0.24, 0.3), most are labeled as category 1
  - For funds with scores between 0.25 and 0.4, majority are labeled as category 3, with some others labeled as 4 or 5 (positive SMB)
  - For funds with scores above roughly 0.4, they are generally labeled into 4 categories based on signs of SML and HML
- We could conclude that our heuristic method of categorizing mutual funds based on scores (compatibility with FF3 model) and betas (size and value factors) are sound and reasonable in general.