In [37]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

1.Part 1: Minimum-Tracking-Error Frontier

Let the market return be the target return. Estimate the expected deviation from market return, for the ten industry portfolios:

$$R_i = E\left(\tilde{R}_i - \tilde{R}_m\right)$$

In [38]:
```
df_market_portfolio = pd.read_excel("Market_Portfolio.xlsx", index_col="Dat
df_market_portfolio.head()
```

```
E:\software\anaconda3\Lib\site-packages\openpyxl\worksheet\_read_only.py:
79: UserWarning: Unknown extension is not supported and will be removed
  for idx, row in parser.parse():
```

Out[38]:

	Market
Date	
200401	2.22
200402	1.46
200403	-1.23
200404	-1.75
200405	1.23

In [39]:
```
df_industry_portfolios = pd.read_excel("Industry_Portfolios.xlsx", index_c
df_industry_portfolios.head()
```

```
E:\software\anaconda3\Lib\site-packages\openpyxl\worksheet\_read_only.py:
79: UserWarning: Unknown extension is not supported and will be removed
  for idx, row in parser.parse():
```

Out[39]:

	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other
Date										
200401	0.06	-1.07	-0.62	0.44	4.53	1.41	0.45	3.09	1.92	2.88
200402	4.25	-0.07	1.95	4.69	-2.92	-0.52	6.09	0.89	2.07	2.16
200403	-0.09	-1.15	-0.27	-0.13	-2.55	-2.07	0.29	-3.96	1.13	-0.63
200404	1.42	2.30	-0.17	2.52	-4.91	-0.48	-2.70	3.54	-3.55	-3.76
200405	-1.89	-1.64	1.61	0.39	4.85	-2.95	0.30	-0.42	1.28	1.86

In [40]:
```
df = pd.merge(df_industry_portfolios, df_market_portfolio, right_index=True
df.head()
```

Out[40]:

Date	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other	Market
200401	0.06	-1.07	-0.62	0.44	4.53	1.41	0.45	3.09	1.92	2.88	2.22
200402	4.25	-0.07	1.95	4.69	-2.92	-0.52	6.09	0.89	2.07	2.16	1.46
200403	-0.09	-1.15	-0.27	-0.13	-2.55	-2.07	0.29	-3.96	1.13	-0.63	-1.23
200404	1.42	2.30	-0.17	2.52	-4.91	-0.48	-2.70	3.54	-3.55	-3.76	-1.75
200405	-1.89	-1.64	1.61	0.39	4.85	-2.95	0.30	-0.42	1.28	1.86	1.23

In [41]:
```
lst_industry_names = list(df_industry_portfolios.columns)
df1 = df.copy()

lst_excess_names = []
for name in lst_industry_names:
    col = name + "-Rm"
    lst_excess_names.append(col)
    df1[col] = df1[name] - df1["Market"]
df1.head()
#lst_excess_names
```

Out[41]:

Date	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other	...	NoDur-Rm
200401	0.06	-1.07	-0.62	0.44	4.53	1.41	0.45	3.09	1.92	2.88	...	-2.16
200402	4.25	-0.07	1.95	4.69	-2.92	-0.52	6.09	0.89	2.07	2.16	...	2.79
200403	-0.09	-1.15	-0.27	-0.13	-2.55	-2.07	0.29	-3.96	1.13	-0.63	...	1.14
200404	1.42	2.30	-0.17	2.52	-4.91	-0.48	-2.70	3.54	-3.55	-3.76	...	3.17
200405	-1.89	-1.64	1.61	0.39	4.85	-2.95	0.30	-0.42	1.28	1.86	...	-3.12

5 rows × 21 columns

In [42]:
```
#Ri = E(Ri-Rm)
se_excess_returns = df1[lst_excess_names].apply(np.mean)
se_excess_returns
```

Out[42]:
```
NoDur-Rm    0.154750
Durbl-Rm   -0.014750
Manuf-Rm    0.264750
Enrgy-Rm    0.483083
HiTec-Rm    0.018167
Telcm-Rm    0.133333
Shops-Rm    0.168250
Hlth-Rm     0.035750
Utils-Rm    0.159083
Other-Rm   -0.259000
dtype: float64
```

Also estimate the covariance matrix of return deviations, for the ten industry portfolios:

$$V_{ij} = \mathrm{Cov}\left[\left(\tilde{R}_i - \tilde{R}_m\right), \left(\tilde{R}_j - \tilde{R}_m\right)\right]$$

In [43]:
```
df_covs = df1[lst_excess_names].cov()
df_covs.head()
```

Out[43]:

	NoDur-Rm	Durbl-Rm	Manuf-Rm	Enrgy-Rm	HiTec-Rm	Telcm-Rm	Shops-Rm	Hlth-Rr
NoDur-Rm	5.439696	-6.073035	-1.396192	-1.200533	-1.883151	1.538885	1.140741	3.81513
Durbl-Rm	-6.073035	26.628901	4.908024	-3.481055	1.891577	-1.707625	-0.354335	-8.08294
Manuf-Rm	-1.396192	4.908024	2.950499	1.666133	0.065267	-0.626416	-1.154597	-2.28890
Enrgy-Rm	-1.200533	-3.481055	1.666133	19.274911	-1.516972	-1.040525	-3.710439	-2.48579
HiTec-Rm	-1.883151	1.891577	0.065267	-1.516972	5.098746	-0.773294	-0.245350	-1.93628

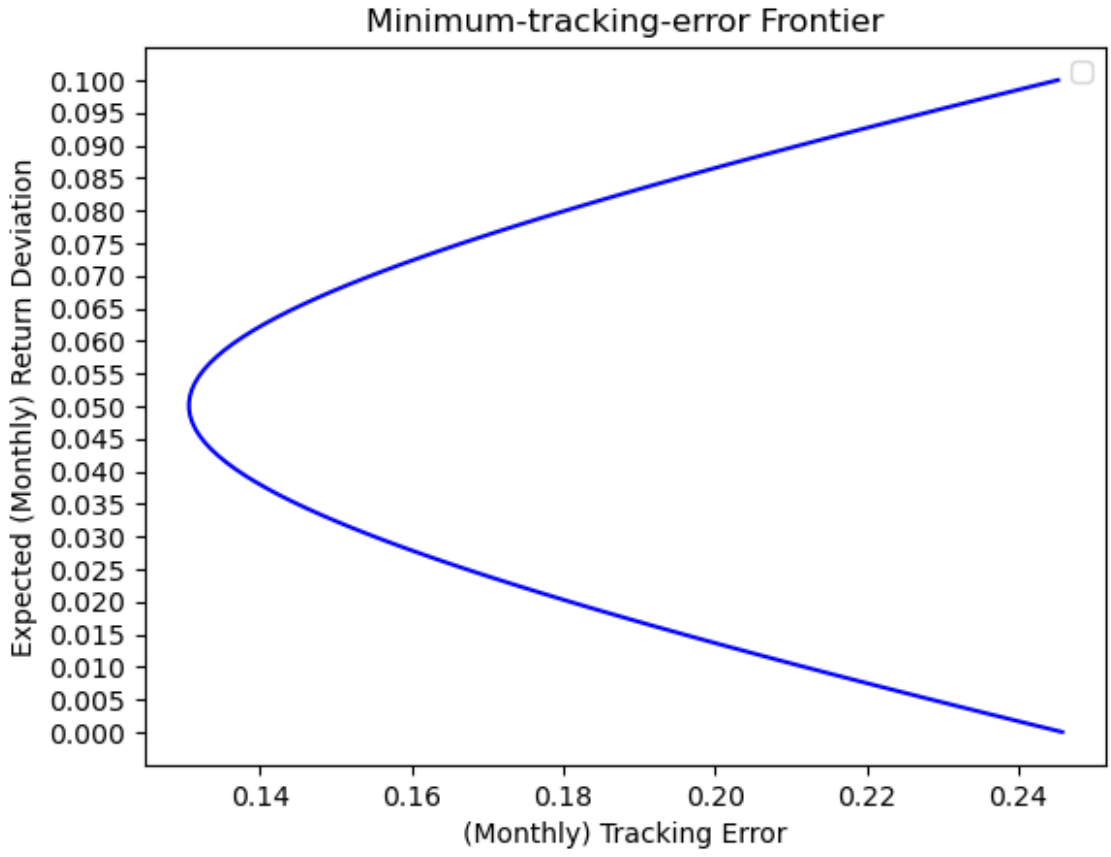
1.1.Plot the minimum-tracking-error frontier generated by the ten industry portfolios.

This graph must have expected (monthly) return deviation on the vertical axis vs (monthly) tracking error on the horizontal axis.

This graph must cover the range from 0% to 0.1% on the vertical axis, in increments of 0.005% (or less).

In [44]:
```
V = df_covs.copy()
V_inv = pd.DataFrame(np.linalg.inv(V), columns=V.columns, index=V.index)

R = se_excess_returns
e = pd.Series([1]*10)
e.index = R.index

alpha = R.dot(V_inv).dot(e)
zeta = R.dot(V_inv).dot(R)
delta = e.dot(V_inv).dot(e)

R_mv = alpha/delta

R_p = np.linspace(0, 0.1, 100)

sigma_p = np.sqrt(1/delta + delta/(zeta*delta-alpha**2)*(R_p-R_mv)**2)

plt.plot(sigma_p, R_p, color='blue', linestyle='-')

plt.yticks(np.arange(0, 0.105, 0.005))
#plt.yticks(np.arange(0, 2.1, 0.1))
plt.xlabel('(Monthly) Tracking Error')
plt.ylabel('Expected (Monthly) Return Deviation')
plt.title('Minimum-tracking-error Frontier')
plt.legend()
plt.show()
```

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

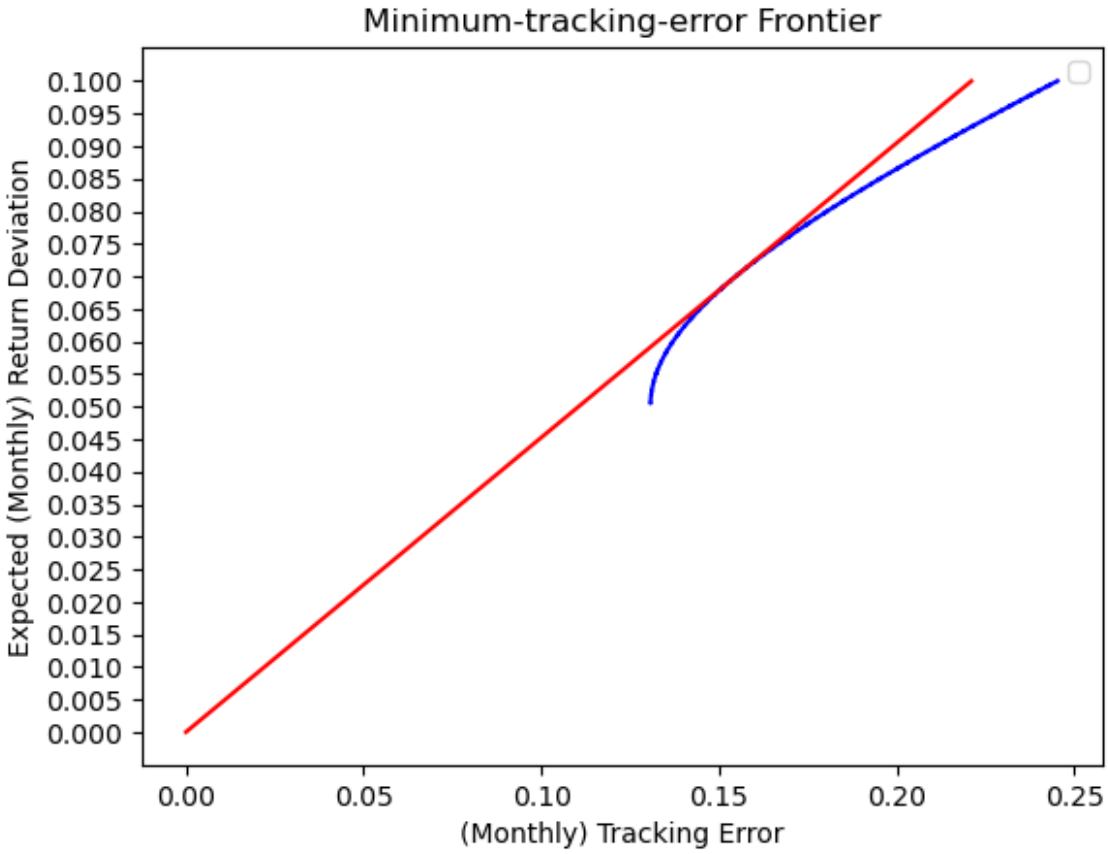


Loading [MathJax]/extensions/Safe.js

1.2.Also plot the line starting from the origin that is tangent to the upper half of the minimum-tracking-error frontier.

In [45]:
```
R_f = 0
R_p = np.linspace(0, 0.1, 100)
R_p_riskless = np.linspace(0, 0.1, 100)

R_p = R_mv + np.sqrt((sigma_p**2-1/delta)*(zeta*delta-alpha**2)/delta)
sigma_p_riskless = (R_p_riskless-R_f)/np.sqrt(zeta - 2*alpha*R_f + delta*(

plt.plot(sigma_p, R_p, color='blue', linestyle='--')
plt.plot(sigma_p_riskless, R_p_riskless, color='red', linestyle='-')

plt.yticks(np.arange(0, 0.105, 0.005))
plt.xlabel('(Monthly) Tracking Error')
plt.ylabel('Expected (Monthly) Return Deviation')
plt.title('Minimum-tracking-error Frontier')
plt.legend()
plt.show()
```

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



1.3.Calculate the information ratio and portfolio weights for the "tangency" portfolio.

In [46]:
```
np.sqrt(zeta - 2*alpha*R_f + delta*R_f**2)
```

Out[46]:
```
0.4524875396199334
```

In [47]:
```
R_tg = (alpha*R_f - zeta)/(delta*R_f - alpha)
R_tg
a = (zeta*V_inv.dot(e)-alpha*V_inv.dot(R))/(zeta*delta-alpha**2)
b = (delta*V_inv.dot(R) - alpha*V_inv.dot(e))/(zeta*delta-alpha**2)
w_star = a + b*R_tg

w_star
```

Out[47]:
```
NoDur-Rm    0.052634
Durbl-Rm    0.000153
Manuf-Rm    0.137627
Enrgy-Rm    0.087032
HiTec-Rm    0.179353
Telcm-Rm    0.071074
Shops-Rm    0.106884
Hlth-Rm     0.102776
Utils-Rm    0.040162
Other-Rm    0.222304
dtype: float64
```

2.Part 2: Minimum-Variance Frontier w/o Short Sales

Use the monthly returns of the ten industry portfolios to generate the minimum-variance frontier without short sales, using Monte Carlo simulation. Portfolio weights will be limited to the range [0, 1].

Randomly draw each element of w, the 10 × 1 vector of portfolio weights, from the (standard) uniform distribution in the range [0, 1].

In [48]:
```
se_w = pd.Series(np.random.rand(10))
se_w
```

Out[48]:
```
0    0.602211
1    0.959675
2    0.293226
3    0.199790
4    0.592547
5    0.012418
6    0.006196
7    0.478541
8    0.179980
9    0.175441
dtype: float64
```

Divide w by the sum of the portfolio weights, to ensure that the portfolio weights sum to one.

In [49]:
```
se_stand_w = se_w/sum(se_w)
se_stand_w.index = lst_industry_names
se_stand_w
```

Out[49]:
```
NoDur    0.172059
Durbl    0.274191
Manuf    0.083778
Enrgy    0.057082
HiTec    0.169298
Telcm    0.003548
Shops    0.001770
Hlth     0.136725
Utils    0.051422
Other    0.050126
dtype: float64
```

Use the normalised w to calculate the mean return and standard deviation of return for the simulated portfolio.

In [50]:
```
se_stand_w.dot(df1[lst_industry_names].cov()).dot(se_stand_w)
```

Out[50]:
```
23.734732055995213
```

In [51]:
```
np.sqrt(se_stand_w.dot(df1[lst_industry_names].cov()).dot(se_stand_w))
```

Out[51]:
```
4.871830462566941
```

In [52]:
```
se_stand_w.dot(df1[lst_industry_names].apply(np.mean))
```

Out[52]:
```
0.8243532583444865
```

Repeat this process until you have (at least) 10^5 data points.

In [66]:
```
n = int(1e5)

df_Ws = np.random.rand(n, len(lst_industry_names))
df_Ws = pd.DataFrame(df_Ws).T
df_Ws.index = lst_industry_names
df_Normal_Ws = df_Ws/df_Ws.sum()
df_Normal_Ws.T.iloc[0:10].T
```

Out[66]:

	0	1	2	3	4	5	6	7	
NoDur	0.083385	0.142234	0.169513	0.258185	0.148120	0.121097	0.035470	0.135268	0.12
Durbl	0.131590	0.016093	0.084574	0.000910	0.061514	0.094277	0.122475	0.110820	0.05
Manuf	0.022918	0.061885	0.003097	0.064197	0.085990	0.079741	0.144775	0.046116	0.13
Enrgy	0.124298	0.108513	0.204166	0.123272	0.077496	0.108577	0.063182	0.074202	0.12
HiTec	0.104436	0.079796	0.060730	0.039966	0.054532	0.123039	0.152408	0.084842	0.06
Telcm	0.114449	0.087939	0.111268	0.123280	0.136800	0.062872	0.148044	0.140624	0.07
Shops	0.167933	0.143762	0.016445	0.100006	0.067921	0.098421	0.020482	0.042873	0.05
Hlth	0.003325	0.132015	0.157669	0.112523	0.135179	0.049455	0.066678	0.167962	0.14
Utils	0.095776	0.081780	0.008021	0.147416	0.142946	0.127892	0.111180	0.023664	0.12
Other	0.151890	0.145984	0.184517	0.030245	0.089503	0.134630	0.135305	0.173628	0.10

In [59]:
```
df_temp = df_Normal_Ws.T.dot(df1[lst_industry_names].cov())
```

In [67]:
```
df_temp.dot(df_Normal_Ws.T.iloc[0:10].T)
```

Out[67]:

	0	1	2	3	4	5	6	
0	18.040379	15.884556	17.137543	14.271411	15.900803	17.328604	18.557215	17.4968
1	18.213210	15.963822	17.393035	14.516489	16.062368	17.495855	18.689038	17.5260
2	17.402297	15.399022	16.510077	13.789712	15.416517	16.697075	17.930758	17.0745
3	20.200501	17.538810	18.983275	15.511943	17.537377	19.267728	20.782170	19.6231
4	18.951595	16.487209	17.927237	14.769235	16.553552	18.116061	19.428035	18.3177
...	...	...	...	...	...	...	...	
99995	17.279939	15.271100	16.555478	13.945987	15.335139	16.665036	17.776718	16.6924
99996	19.502258	16.910097	18.273731	15.156610	17.073645	18.661010	20.139815	18.8918
99997	20.667193	17.890905	19.643245	16.036248	18.001116	19.791917	21.252642	19.9009
99998	18.697119	16.349027	17.698804	14.644291	16.413307	17.922874	19.271364	18.2008
99999	18.826073	16.330910	17.621444	14.551861	16.421383	17.946561	19.311505	18.2568

100000 rows × 10 columns

In [29]:
```
#trying to improve
import pandas as pd
import numpy as np

# 随机生成数据
n = int(1e5)

data = np.random.rand(n, len(lst_industry_names))
data = pd.DataFrame(data).T
data.index = lst_industry_names
df1 = pd.DataFrame(data, columns=lst_industry_names)

# 预分配DataFrame的大小
df_datapoints = pd.DataFrame(index=range(n), columns=["return", "std"])

for i in range(n):
    se_w = pd.Series(data[i])
    se_stand_w = se_w / se_w.sum()

    portfolio_std = np.sqrt(se_stand_w.dot(df1[lst_industry_names].cov()).(
    portfolio_return = se_stand_w.dot(df1[lst_industry_names].mean())

    df_datapoints.loc[i] = [portfolio_return, portfolio_std]
```

```
---------------------------------------------------------------------
--
KeyboardInterrupt                         Traceback (most recent call las
t)
Cell In[29], line 20
     17 se_w = pd.Series(data[i])
     18 se_stand_w = se_w / se_w.sum()
---> 20 portfolio_std = np.sqrt(se_stand_w.dot(df1[lst_industry_names].co
v()).dot(se_stand_w))
     21 portfolio_return = se_stand_w.dot(df1[lst_industry_names].mean())
     23 df_datapoints.loc[i] = [portfolio_return, portfolio_std]

File E:\software\anaconda3\Lib\site-packages\pandas\core\series.py:3121,
in Series.dot(self, other)
   3115         raise Exception(
   3116             f"Dot product shape mismatch, {lvals.shape} vs {rval
s.shape}"
   3117         )
   3119 if isinstance(other, ABCDataFrame):
   3120     return self._constructor(
-> 3121         np.dot(lvals, rvals), index=other.columns, copy=False
   3122     ).__finalize__(self, method="dot")
   3123 elif isinstance(other, Series):
   3124     return np.dot(lvals, rvals)

File <__array_function__ internals>:200, in dot(*args, **kwargs)

KeyboardInterrupt:
```

Loading [MathJax]/extensions/Safe.js

In [21]:
```
df_datapoints = pd.DataFrame(columns=["return", "std"])
for i in range(int(1e5)):
    se_w = pd.Series(np.random.rand(10))
    se_stand_w = se_w/sum(se_w)
    se_stand_w.index = lst_industry_names

    profolio_std = np.sqrt(se_stand_w.dot(df1[lst_industry_names].cov()).d

    profolio_return = se_stand_w.dot(df1[lst_industry_names].apply(np.mean
    df_datapoints.loc[len(df_datapoints)] = [profolio_return, profolio_std
```

In [ ]:
```
df_datapoints
```

2.1.Plot the data points with mean return on the vertical axis vs standard deviation of return on the horizontal axis.

In [ ]:
```
plt.scatter(df_datapoints["std"], df_datapoints["return"])
plt.show()
```

Repeat this entire process by simulating 1/w using the standard uniform distribution ⇒ take the reciprocal of the random draw from the standard uniform distribution as the portfolio weight.

2.2.Plot the new data points (on a separate graph) with mean return on the vertical axis vs standard deviation of return on the horizontal axis.

In [ ]:
```
df_datapoints1 = pd.DataFrame(columns=["return", "std"])
for i in range(int(1e5)):
    se_w = pd.Series(1/np.random.rand(10))
    se_stand_w = se_w/sum(se_w)
    se_stand_w.index = lst_industry_names

    profolio_std = np.sqrt(se_stand_w.dot(df1[lst_industry_names].cov()).d

    profolio_return = se_stand_w.dot(df1[lst_industry_names].apply(np.mean
    df_datapoints1.loc[len(df_datapoints1)] = [profolio_return, profolio_s
```

In [ ]:
```
df_datapoints1
```

In [ ]:
```
plt.scatter(df_datapoints1["std"], df_datapoints1["return"])
plt.show()
```

In [ ]:
```

```