FIN30150: Financial Economics 1: Group Project Q3 ETF data

Odhran Murphy 19469442

Import Libraries

In [1]:
```
# Import the standard data analysis libraries
import pandas as pd
import numpy as np

# Plotting library
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
plt.style.use("seaborn-darkgrid")
matplotlib.rcParams['figure.figsize']=[8,4]

# OLS Regression Library
import statsmodels.api as sm
from statsmodels.formula.api import ols
import statsmodels.formula.api as smf

# Import specific libraries to read in data
from pandas_datareader import data as pdr
import yfinance as yf
yf.pdr_override()
import datetime
from datetime import datetime

# OS Module
import os

# Import other modules
from IPython.display import Markdown
```

Define Functions

`returns_describe`

In [2]:
```
def returns_describe(series, title):
    """Adding Skewness and Kurtosis to in built describe function"""

    stats = series.describe()
    stats.loc['skew'] = series.skew().tolist()
    stats.loc['kurt'] = (series.kurtosis() + 3).tolist() # Python calculates excess kurtosis
    display(Markdown(f"**{str(title)} Returns Characteristics**"))
    display(stats)
```

`plot_series_v_normal`

In [3]:
```
def plot_series_v_normal(series, series_label, title):
    """Plot series histogram v normal distribution"""

    # Applying the seaborn distplot function to get histogram & density curve of series
    sns.histplot(series, kde=True, color='darkblue', stat='density', label = series_label)

    # Normal distribution with the same mean & variance
    np.random.seed(0) # Fixed seed to use

    # Random variable with normal distribution
    normal_sample = np.random.normal(np.mean(series), np.std(series), 249)

    # Applying the Gaussian kernel density estimate to get the density curve
    sns.kdeplot(normal_sample, color='red', shade=True, label = "Normal")

    # Calculate series skew and kurtosis
    skewness = series.skew()
    kurtosis = series.kurtosis() + 3 # Python calculates excess kurtosis

    plt.title(title)
    plt.text(-0.12,13, f'Skewness: {skewness:.3f} \nKurtosis: {kurtosis:.3f}')
    plt.xlabel('Returns')
    plt.legend()
    plt.show()
```

Download Data

Dateparsers for Sector ETF and Fama French data

In [4]:
```
# Dateparser to read in Sector ETF data so that the pandas read data function
# recognises the date format in the first column (where dates are expressed as 29JAN1999 for 29 Jan 1999)
dateparserETF = lambda x: datetime.strptime(x, '%d%b%Y')

# Check that date parser works on a given date
print(dateparserETF('29JAN1999'))

# Dateparser to read in Fama French data so that the pandas read data function
# recognises the date format in the first column (where dates are expressed as 202009 for Sep 2020)
dateparserFF = lambda x: datetime.strptime(x, '%Y%m')

# Check that date parser works on a given date
print(dateparserFF('202009'))
```

```
1999-01-29 00:00:00
2020-09-01 00:00:00
```

Set GitHub Repository

In [5]:
```
base_url = 'https://raw.githubusercontent.com/odhran-murphy/FE1_Group_Project/main'
```

Read Sector ETFs data

In [6]:
```
filename = 'SectorETF.csv'
df_etf = pd.read_csv(f'{base_url}/{filename}', parse_dates=['Date'], date_parser=dateparserETF)
df_etf.set_index('Date',inplace=True)
df_etf
```

Out[6]:

Date	N/A/N	NY/AM	S&P 500	XLB	XLC	XLE	XLF	XLI	XLK	XLP	XLU	XLV	XLY	EWPRET	VWPRET
1999-01-29	0.0385	0.0129	0.0428	-0.0371	NaN	-0.0656	0.0173	-0.0108	0.1590	-0.0132	-0.0248	0.0481	0.0514	0.0138	0.0495
1999-02-26	-0.0381	-0.0241	-0.0319	0.0155	NaN	-0.0086	0.0157	0.0090	-0.0992	-0.0105	-0.0276	0.0011	-0.0063	-0.0123	-0.0373
1999-03-31	0.0379	0.0285	0.0390	0.0181	NaN	0.1414	0.0325	0.0210	0.0743	-0.0011	-0.0626	0.0263	0.0487	0.0332	0.0499
1999-04-30	0.0491	0.0533	0.0376	0.2471	NaN	0.1480	0.0702	0.1504	0.0060	-0.0349	0.0966	0.0357	0.0262	0.0828	0.0520
1999-05-28	-0.0207	-0.0186	-0.0232	-0.0910	NaN	-0.0216	-0.0603	-0.0196	0.0034	-0.0104	0.0117	-0.0307	-0.0453	-0.0293	-0.0235
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2019-08-30	-0.0208	-0.0193	-0.0161	-0.0283	-0.0246	-0.0833	-0.0471	-0.0265	-0.0154	0.0217	0.0509	-0.0059	-0.0094	-0.0159	-0.0191
2019-09-30	0.0161	0.0216	0.0188	0.0318	0.0022	0.0397	0.0455	0.0302	0.0158	0.0174	0.0424	-0.0010	0.0127	0.0260	0.0244
2019-10-31	0.0192	0.0104	0.0216	-0.0002	0.0222	-0.0209	0.0250	0.0113	0.0390	-0.0042	-0.0076	0.0513	0.0012	0.0105	0.0163
2019-11-29	0.0361	0.0313	0.0362	0.0318	0.0383	0.0160	0.0505	0.0450	0.0537	0.0137	-0.0187	0.0500	0.0132	0.0284	0.0331
2019-12-31	0.0284	0.0245	0.0298	0.0285	0.0226	0.0600	0.0261	-0.0020	0.0432	0.0241	0.0328	0.0348	0.0276	0.0306	0.0314

252 rows × 15 columns

Read Fama French data

3 factor monthly data

In [7]:

```python
filename = 'FF_data_2022_monthly.csv'
df_ff_3f = pd.read_csv(f'{base_url}/{filename}', parse_dates=['Date'], date_parser=dateparserFF)

#Offset FF data to last business day of month as it should be
df_ff_3f['Date'] = pd.DatetimeIndex(df_ff_3f['Date']) + pd.offsets.BMonthEnd(1)
df_ff_3f.set_index('Date',inplace=True)
df_ff_3f = df_ff_3f.divide(100)

# Converting all FF column names to lower case to avoid duplicates with SP500 ticker names
df_ff_3f = df_ff_3f.rename(columns=str.lower)

df_ff_3f
```

Out[7]:

Date	mkt-rf	smb	hml	rf
1926-07-30	0.0296	-0.0256	-0.0243	0.0022
1926-08-31	0.0264	-0.0117	0.0382	0.0025
1926-09-30	0.0036	-0.0140	0.0013	0.0023
1926-10-29	-0.0324	-0.0009	0.0070	0.0032
1926-11-30	0.0253	-0.0010	-0.0051	0.0031
...	...	...	...	...
2022-05-31	-0.0034	-0.0185	0.0841	0.0003
2022-06-30	-0.0843	0.0209	-0.0597	0.0006
2022-07-29	0.0957	0.0281	-0.0410	0.0008
2022-08-31	-0.0378	0.0139	0.0031	0.0019
2022-09-30	-0.0936	-0.0081	0.0005	0.0019

1155 rows × 4 columns

5 factor monthly data

In [8]:

```python
filename = 'FF_data_2022_monthly_5f.csv'
df_ff_5f = pd.read_csv(f'{base_url}/{filename}', parse_dates=['Date'], date_parser=dateparserFF)

#Offset FF data to last business day of month as it should be
df_ff_5f['Date'] = pd.DatetimeIndex(df_ff_5f['Date']) + pd.offsets.BMonthEnd(1)
df_ff_5f.set_index('Date',inplace=True)
df_ff_5f = df_ff_5f.divide(100)

# Converting all FF column names to lower case to avoid duplicates with SP500 ticker names
df_ff_5f = df_ff_5f.rename(columns=str.lower)
df_ff_5f
```

Out[8]:

Date	mkt-rf	smb	hml	rmw	cma	rf
1963-07-31	-0.0039	-0.0041	-0.0097	0.0068	-0.0118	0.0027
1963-08-30	0.0507	-0.0080	0.0180	0.0036	-0.0035	0.0025
1963-09-30	-0.0157	-0.0052	0.0013	-0.0071	0.0029	0.0027
1963-10-31	0.0253	-0.0139	-0.0010	0.0280	-0.0201	0.0029
1963-11-29	-0.0085	-0.0088	0.0175	-0.0051	0.0224	0.0027
...	...	...	...	...	...	...
2022-05-31	-0.0034	-0.0006	0.0841	0.0144	0.0398	0.0003
2022-06-30	-0.0843	0.0130	-0.0597	0.0185	-0.0470	0.0006
2022-07-29	0.0957	0.0188	-0.0410	0.0068	-0.0694	0.0008
2022-08-31	-0.0378	0.0151	0.0031	-0.0480	0.0131	0.0019
2022-09-30	-0.0936	-0.0096	0.0005	-0.0140	-0.0082	0.0019

711 rows × 6 columns

Set start and end dates

In [9]:

```python
start = datetime(1989,12,1)
end = datetime(2022,12,31)
```

Download SP500 data

In [10]:

```python
#Download and save dataset (Only download if data has not already been downloaded)
target = 'SP500.csv'

if not os.path.isfile(target):
    df_sp_500 = pdr.get_data_yahoo('^GSPC', start, end, interval = '1mo')
    df_sp_500.to_csv(target)

df_sp_500 = pd.read_csv(f'{base_url}/{target}') # Always using saved dataset
```

```python
df_sp_500.set_index('Date',inplace=True)
df_sp_500_adj_close = df_sp_500.drop(columns=['Open','High','Low','Close','Volume'])
df_sp_500_adj_close.rename(columns={'Adj Close': 'SP500'}, inplace=True)
df_sp_500_adj_close
```

Out[10]:

Date	SP500
1989-12-01	353.399994
1990-01-01	329.079987
1990-02-01	331.890015
1990-03-01	339.940002
1990-04-01	330.799988
...	...
2022-08-01	3955.000000
2022-09-01	3585.620117
2022-10-01	3871.979980
2022-11-01	3992.929932
2022-11-14	3957.250000

397 rows × 1 columns

Download SP100 data

In [11]:

```python
#Download and save dataset (Only download if data has not already been downloaded)
target = 'SP100.csv'

if not os.path.isfile(target):
    df_sp_100 = pdr.get_data_yahoo('^OEX', start, end, interval = '1mo')
    df_sp_100.to_csv(target)

df_sp_100 = pd.read_csv(f'{base_url}/{target}') # Always using saved dataset
df_sp_100.set_index('Date',inplace=True)
df_sp_100_adj_close = df_sp_100.drop(columns=['Open','High','Low','Close','Volume'])
df_sp_100_adj_close.rename(columns={'Adj Close': 'SP100'}, inplace=True)
df_sp_100_adj_close
```

Out[11]:

Date	SP100
1989-12-01	164.675003
1990-01-01	153.940002
1990-02-01	156.240005
1990-03-01	160.014999
1990-04-01	157.115005
...	...
2022-08-01	1797.949951
2022-09-01	1625.760010
2022-10-01	1740.510010
2022-11-01	1790.209961
2022-11-23	1791.890015

397 rows × 1 columns

Concentate the ETF dataframe, SP100 dataframe, SP500 dataframe and 3 factor FF dataframes

In [12]:

```python
# ETF dataframe edits
df_etf = df_etf.loc[:,'XLB':'XLY'] # Keep necessary columns in Sector ETF data
df_etf = df_etf.resample('BM').last() # Set to month end

# SP100 dataframe edits
df_sp_100_adj_close.index = pd.to_datetime(df_sp_100_adj_close.index) # Set to datetime
df_sp_100_adj_close = df_sp_100_adj_close.resample('BM').last() # Set to month end
df_sp_100_rets = np.log(df_sp_100_adj_close/df_sp_100_adj_close.shift(1)) # Isolate SP100 returns from its dataframe

# SP500 dataframe edits
df_sp_500_adj_close.index = pd.to_datetime(df_sp_500_adj_close.index) # Set to datetime
df_sp_500_adj_close = df_sp_500_adj_close.resample('BM').last() # Set to month end
df_sp_500_rets = np.log(df_sp_500_adj_close/df_sp_500_adj_close.shift(1)) # Isolate SP500 returns from its dataframe

# 3 factor FF dataframe edits
df_ff_3f = df_ff_3f.resample('BM').last()

# Merge dataframes
df = pd.merge(df_etf,df_sp_100_rets, how='inner', left_index=True, right_index=True)
df = pd.merge(df,df_sp_500_rets, how='inner', left_index=True, right_index=True)
df = pd.merge(df,df_ff_3f, how='inner', left_index=True, right_index=True)

display(df)
```

Date	XLB	XLC	XLE	XLF	XLI	XLK	XLP	XLU	XLV	XLY	SP100	SP500	mkt-rf	smb	hml	rf
1999-01-29	-0.0371	NaN	-0.0656	0.0173	-0.0108	0.1590	-0.0132	-0.0248	0.0481	0.0514	0.058532	0.040191	0.0350	0.0076	-0.0460	0.0035
1999-02-26	0.0155	NaN	-0.0086	0.0157	0.0090	-0.0992	-0.0105	-0.0276	0.0011	-0.0063	-0.034988	-0.032815	-0.0408	-0.0609	0.0192	0.0035
1999-03-31	0.0181	NaN	0.1414	0.0325	0.0210	0.0743	-0.0011	-0.0626	0.0263	0.0487	0.044545	0.038061	0.0345	-0.0379	-0.0274	0.0043
1999-04-30	0.2471	NaN	0.1480	0.0702	0.1504	0.0060	-0.0349	0.0966	0.0357	0.0262	0.043963	0.037242	0.0433	0.0391	0.0246	0.0037
1999-05-31	-0.0910	NaN	-0.0216	-0.0603	-0.0196	0.0034	-0.0104	0.0117	-0.0307	-0.0453	-0.025468	-0.025288	-0.0246	0.0334	0.0235	0.0034
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2019-08-30	-0.0283	-0.0246	-0.0833	-0.0471	-0.0265	-0.0154	0.0217	0.0509	-0.0059	-0.0094	-0.018939	-0.018257	-0.0258	-0.0236	-0.0476	0.0016
2019-09-30	0.0318	0.0022	0.0397	0.0455	0.0302	0.0158	0.0174	0.0424	-0.0010	0.0127	0.017541	0.017035	0.0143	-0.0097	0.0674	0.0018
2019-10-31	-0.0002	0.0222	-0.0209	0.0250	0.0113	0.0390	-0.0042	-0.0076	0.0513	0.0012	0.025900	0.020226	0.0206	0.0029	-0.0192	0.0016
2019-11-29	0.0318	0.0383	0.0160	0.0505	0.0450	0.0537	0.0137	-0.0187	0.0500	0.0132	0.035566	0.033480	0.0387	0.0078	-0.0201	0.0012
2019-12-31	0.0285	0.0226	0.0600	0.0261	-0.0020	0.0432	0.0241	0.0328	0.0348	0.0276	0.030656	0.028189	0.0277	0.0073	0.0176	0.0014

252 rows × 16 columns

In [13]:

```python
# Communication Services ETF has missing data so drop this column (index 1 drops column, 0 would drop rows)
df = df.drop('XLC', axis=1)
rets_all = df.loc[:,'XLB':'XLY'].values
```

```
# List of ETFs used
ETF_list = df.loc[:,'XLB':'XLY'].columns.values.tolist()
```

Set in-sample and out-of-sample period

Set the oldest five years of data [1;m] as the initial in-sample period. Data starts at 1999-01-31 so use 2003-12-31 as split date.

In [14]:
```
is_end_date = datetime(2003,12,31)
os_start_date = datetime(2004,1,31)

# In Sample period
df_is = df.loc[:is_end_date,'XLB':'XLY']
dates_is = df_is.index

# rets is set to a numpy array
rets_is = df_is.values
T_in = len(rets_is)
print(f'Number of Months in In-Sample Observations: {T_in}', '\n')
display(Markdown("**In-Sample Dataframe**"))
display(df_is.head())

# Out-of-Sample period
df_os = df.loc[os_start_date:,'XLB':'XLY']
dates_os = df_os.index
T_os = len(dates_os)
display(Markdown("**Out-of-Sample Dataframe**"))
display(df_os.head())
```

```
Number of Months in In-Sample Observations: 60
```

In-Sample Dataframe

Date	XLB	XLE	XLF	XLI	XLK	XLP	XLU	XLV	XLY
1999-01-29	-0.0371	-0.0656	0.0173	-0.0108	0.1590	-0.0132	-0.0248	0.0481	0.0514
1999-02-26	0.0155	-0.0086	0.0157	0.0090	-0.0992	-0.0105	-0.0276	0.0011	-0.0063
1999-03-31	0.0181	0.1414	0.0325	0.0210	0.0743	-0.0011	-0.0626	0.0263	0.0487
1999-04-30	0.2471	0.1480	0.0702	0.1504	0.0060	-0.0349	0.0966	0.0357	0.0262
1999-05-31	-0.0910	-0.0216	-0.0603	-0.0196	0.0034	-0.0104	0.0117	-0.0307	-0.0453

Out-of-Sample Dataframe

Date	XLB	XLE	XLF	XLI	XLK	XLP	XLU	XLV	XLY
2004-02-27	0.0521	0.0525	0.0286	-0.0074	-0.0254	0.0569	0.0189	0.0107	0.0217
2004-03-31	-0.0231	-0.0062	-0.0116	-0.0123	-0.0266	-0.0025	0.0107	-0.0417	-0.0062
2004-04-30	-0.0463	0.0170	-0.0435	-0.0004	-0.0382	0.0131	-0.0428	0.0311	-0.0113
2004-05-31	0.0281	-0.0030	0.0171	0.0246	0.0392	-0.0056	0.0120	-0.0032	0.0064
2004-06-30	0.0500	0.0623	0.0044	0.0617	0.0273	0.0055	0.0172	-0.0037	0.0026

In [15]:
```
# In-sample expected returns vector and standard deviation vector
mu_1d = rets_is.mean(0)
sigma = rets_is.std(0)
N = len(mu_1d) # Number of assets
```

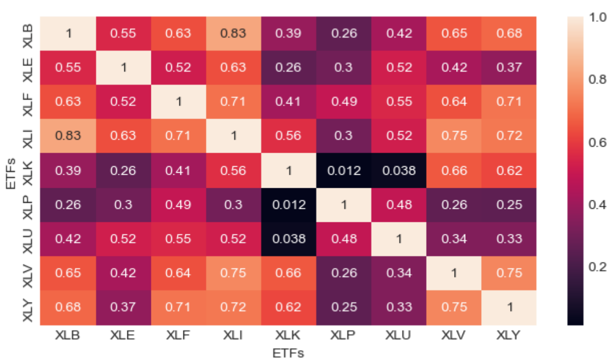
In [16]:
```
returns_describe(df_is, 'ETF: In-Sample')
```

ETF: In-Sample Returns Characteristics

	XLB	XLE	XLF	XLI	XLK	XLP	XLU	XLV	XLY
count	60.000000	60.000000	60.000000	60.000000	60.000000	60.000000	60.000000	60.000000	60.000000
mean	0.007967	0.006093	0.006123	0.004303	-0.001913	-0.001555	0.000175	0.004087	0.005863
std	0.073845	0.061851	0.059939	0.060076	0.107201	0.042919	0.054932	0.047380	0.063012
min	-0.129500	-0.138500	-0.114900	-0.146800	-0.249100	-0.117000	-0.147200	-0.144800	-0.126500
25%	-0.035150	-0.035375	-0.033800	-0.034900	-0.092550	-0.028425	-0.033025	-0.025675	-0.039250
50%	0.007550	-0.008550	-0.003350	0.006250	-0.000350	0.002950	-0.001200	0.003900	0.007200
75%	0.050425	0.030600	0.043375	0.039025	0.084375	0.023150	0.034050	0.035625	0.046975
max	0.247100	0.148000	0.182400	0.150400	0.247700	0.091300	0.131600	0.116000	0.141400
skew	0.491191	0.570017	0.603842	0.154813	-0.027107	-0.522811	-0.118280	-0.308727	-0.027395
kurt	3.749479	3.134369	3.590109	3.226952	2.563831	3.338797	3.529541	3.788637	2.706640

In [17]:
```
# Correlation Matrix
df_iscorr = df_is.corr()

# Plotting of the correlation matrix:
display(Markdown("**ETF Correlation Matrix**"))
sns.heatmap(df_iscorr, annot = True)
plt.xlabel('ETFs')
plt.ylabel('ETFs')
plt.show()
```

ETF Correlation Matrix



In [18]:
```
# Covariance matrix using the numpy array rets (can also estimate covariance of the columns in a dataframe)
```

```python
Sigma = np.cov(rets_is.T) # Must transpose array as default is to estimate covariance along rows rather than down columns
Sigma_df = pd.DataFrame(Sigma, index=ETF_list)
Sigma_df.columns = ETF_list
display(Markdown("**ETF Covariance Matrix**"))
Sigma_df
```

ETF Covariance Matrix

Out[18]:

	XLB	XLE	XLF	XLI	XLK	XLP	XLU	XLV	XLY
XLB	0.005453	0.002493	0.002810	0.003661	0.003049	0.000820	0.001697	0.002268	0.003183
XLE	0.002493	0.003826	0.001918	0.002354	0.001708	0.000794	0.001760	0.001235	0.001451
XLF	0.002810	0.001918	0.003593	0.002541	0.002640	0.001254	0.001797	0.001815	0.002689
XLI	0.003661	0.002354	0.002541	0.003609	0.003632	0.000785	0.001719	0.002147	0.002720
XLK	0.003049	0.001708	0.002640	0.003632	0.011492	0.000054	0.000226	0.003327	0.004192
XLP	0.000820	0.000794	0.001254	0.000785	0.000054	0.001842	0.001139	0.000523	0.000682
XLU	0.001697	0.001760	0.001797	0.001719	0.000226	0.001139	0.003017	0.000874	0.001159
XLV	0.002268	0.001235	0.001815	0.002147	0.003327	0.000523	0.000874	0.002245	0.002240
XLY	0.003183	0.001451	0.002689	0.002720	0.004192	0.000682	0.001159	0.002240	0.003970

In [19]:

```python
# Set up a column unit vector
ell = np.ones((N,1))
```

In [20]:

```python
# Convert 1d array to 2d array (column vector) so that we can use matrix algebra in optimal weights computation
mu = mu_1d[:, np.newaxis]
mu
```

Out[20]:

```
array([[ 0.00796667],
       [ 0.00609333],
       [ 0.00612333],
       [ 0.00430333],
       [-0.00191333],
       [-0.001555  ],
       [ 0.000175  ],
       [ 0.00408667],
       [ 0.00586333]])
```

Global Minimum Variance Portfolio (GMVP)

Weights of GMVP:

$$w_t^{(GMVP)} = (\frac{1}{C})\Sigma^{-1}\ell$$

where

$$C = \ell'\Sigma^{-1}\ell$$

Σ is assumed to be estimated using only information up to time t (i.e. using in sample period data)

Realised scalar out-of-sample return of a portfolio is given by:

$$r_{t+1} = w_t^{'} \times \mathbf{r}_{t+1}$$

In-Sample GMVP

In [21]:

```python
# In-Sample GMVP
Sigma_inv_ell = np.linalg.solve(Sigma, ell)                  # Solving Sigma^{-1}*ell
C = np.dot(ell.T, Sigma_inv_ell)                              # Calculating C = ell'*Sigma^{-1}*ell
w_gmvp_is = (1/C)*Sigma_inv_ell                               # Calculating weights vector: w = (1/C)*Sigma^{-1}*ell
mu_gmvp_is = np.dot(mu.T, w_gmvp_is)                          # In-sample portfolio return
var_gmvp_is = np.dot(w_gmvp_is.T, np.dot(Sigma, w_gmvp_is))  # In-sample portfolio variance
sigma_gmvp_is = np.sqrt(var_gmvp_is)                          # In-sample portfolio std dev
```

Out-of-Sample GMVP

In [22]:

```python
# Out-of-Sample GMVP
w_mvp_os = np.zeros((N,T_os))
r_mvp_os = np.empty(T_os)
for i in range(T_os):
    Sigma_os = np.cov(rets_all[i:T_in+i,:].T)                  # Using [t1,t2] to estimate Sigma ([i,60+i], for i = 0, 1, ...)
    Sigma_inv_ell_os = np.linalg.solve(Sigma_os, ell)          # Storing Sigma^{-1}*ell
    C = np.dot(ell.T, Sigma_inv_ell_os)                         # Calculating C = ell'*Sigma^{-1}*ell
    w = (1/C)*Sigma_inv_ell_os                                  # Calculating weights vector: w = (1/C)*Sigma^{-1}*ell
    w_mvp_os[:,i] = w.T                                         # Storing weights vector
    r_mvp_os[i] = np.dot(rets_all[T_in+i,:], w_mvp_os[:,i])    # Out-of-sample portfolio returns

var_gmvp_os = np.dot(w.T, np.dot(Sigma_os, w))                  # Out-of-sample portfolio variance
sigma_gmvp_os = np.sqrt(var_gmvp_os)                            # Out-of-sample portfolio std dev
mu_mvp_os = r_mvp_os.mean()                                     # Out-of-sample portfolio average returns
```

In [23]:

```python
len(w_mvp_os)
```

Out[23]: 9

In [24]:

```python
len(r_mvp_os)
```

Out[24]: 191

Portfolio 2

Weights of Second Portfolio:

$$w_t^{(2)} = a + BE_t[\tilde{r}_{t+1}]$$

where

$$a = \frac{\Sigma^{-1}\ell}{\ell'\Sigma^{-1}\ell}$$

and

$$B = \frac{1}{\gamma}\left(\Sigma^{-1} - \frac{\Sigma^{-1}\ell\ell'\Sigma^{-1}}{\ell'\Sigma^{-1}\ell}\right)$$

- Σ is assumed to be estimated using only information up to time t (i.e. using in sample period data)
- γ denotes the level of relative risk aversion
- $E_t[\tilde{r}_{t+1}]$ is the vector of CAPM expected excess returns of the stocks estimated using information up to time t (i.e. using in sample period data)

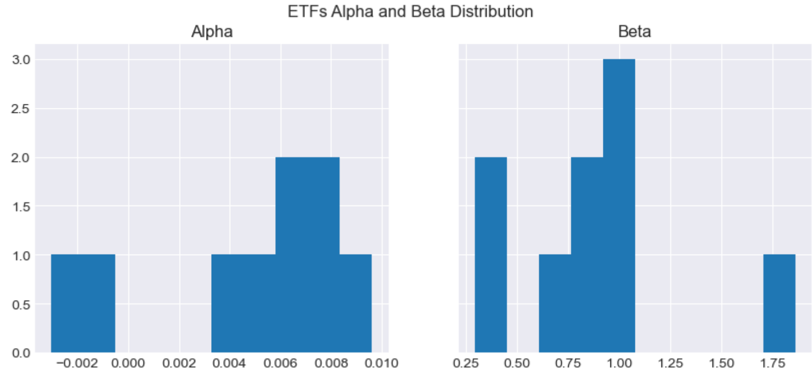
Realised scalar out-of-sample return of a portfolio is given by:

$$r_{t+1} = w_t^{'} \times \mathbf{r}_{t+1}$$

In-Sample Portfolio 2

CAPM to estimate excess returns

In [25]:
```
# Regress excess ETF returns against market returns to estimates alphas and betas

# SP500 excess returns within in sample period
x = df.loc[:is_end_date,'SP500'] - df.loc[:is_end_date,'rf']

# Calculate excess return of each ETF during in sample period
etfs_array = df.loc[:is_end_date,'XLB':'XLY'].values
rf_array = df.loc[:is_end_date,'rf'].values
rf_array = rf_array[:,np.newaxis]
y = etfs_array - rf_array # Array containing dependent variables in each column

# OLS Regression
X = sm.add_constant(x)
ts_res = sm.OLS(y, X, missing='drop').fit()

# Isolate alpha and beta
alpha = ts_res.params.iloc[0,:].values
beta = ts_res.params.iloc[1,:].values

# Plot Histogram of Alphas and Beta
fig, axs = plt.subplots(1, 2, sharey=True, figsize=(10,4))
axs[0].hist(alpha)
axs[0].set_title('Alpha')
axs[1].hist(beta)
axs[1].set_title('Beta')
plt.suptitle('ETFs Alpha and Beta Distribution')
plt.show()

# Format parameter datframe
ts_res.params = ts_res.params.rename(index={'const': 'alpha', 0: 'beta'})
ts_res.params.columns = ETF_list
display(ts_res.params)
```



	XLB	XLE	XLF	XLI	XLK	XLP	XLU	XLV	XLY
alpha	0.009619	0.00620	0.007353	0.005923	0.003600	-0.003024	-0.000729	0.004776	0.007650
beta	0.995633	0.64891	0.900911	0.988249	1.861631	0.295474	0.422349	0.779607	1.025905

In [26]:
```
# Excess return of ETF = alpha + beta*excess returns on market

# SP500 excess returns within in sample period
SP500_exret_is = df.loc[:is_end_date,'SP500'] - df.loc[:is_end_date,'rf']

# Constant beta calculated over in sample period
beta_is = [beta for i in range(T_in)]

# Reorder SP500 excess return list to 2d array
SP500_exret_is = np.array(SP500_exret_is)
SP500_exret_is = SP500_exret_is[:, np.newaxis]

# CAPM calculation - Alpha should be zero
exret_is = beta_is*SP500_exret_is
exret_df_is = pd.DataFrame(data=exret_is, columns=ETF_list, index=dates_is)
display(Markdown("**CAPM expected excess returns of ETFS over In-Sample Period**"))
display(exret_df_is.head())
```

CAPM expected excess returns of ETFS over In-Sample Period

Date	XLB	XLE	XLF	XLI	XLK	XLP	XLU	XLV	XLY
1999-01-29	0.036531	0.023809	0.033055	0.036260	0.068305	0.010841	0.015496	0.028604	0.037641
1999-02-26	-0.036157	-0.023565	-0.032717	-0.035888	-0.067605	-0.010730	-0.015338	-0.028312	-0.037256
1999-03-31	0.033613	0.021908	0.030415	0.033364	0.062850	0.009975	0.014259	0.026320	0.034635
1999-04-30	0.033395	0.021766	0.030218	0.033148	0.062443	0.009911	0.014166	0.026149	0.034411
1999-05-31	-0.028562	-0.018616	-0.025845	-0.028350	-0.053406	-0.008476	-0.012116	-0.022365	-0.029431

Calculate $BE*{t}[\tilde{r}*{t+1}]$

Split $BE_t[\tilde{r}_{t+1}]$ into smaller more managable terms denoting $E_t[\tilde{r}_{t+1}]$ as μ

First term is:

$$d = \frac{1}{\gamma}(\Sigma^{-1}\mu)$$

$$e = \frac{1}{\gamma}\left(\frac{F}{C}\right)\Sigma^{-1}\ell$$

$$F = \ell'\Sigma^{-1}\mu$$

$$C = \ell'\Sigma^{-1}\ell$$

(C as per defintion in GMVP)

Rewriting B to be a function of new variables

$$BE_t[\tilde{r}_{t+1}] = d - e$$

In [27]:
```python
gamma = 3 # Reasonable level of risk aversion found in individual project
```

In [28]:
```python
exret_is_mu_1d = exret_is.mean(0)
exret_is_mu = exret_is_mu_1d[:, np.newaxis]
exret_is_mu
```

Out[28]:
```
array([[-0.00443886],
       [-0.00289305],
       [-0.00401656],
       [-0.00440594],
       [-0.00829976],
       [-0.00131732],
       [-0.00188297],
       [-0.00347574],
       [-0.00457382]])
```

In [29]:
```python
# Solving Sigma^{-1}*mu
Sigma_inv_mu = np.linalg.solve(Sigma, exret_is_mu)

# Calculate d
d = (1/gamma)*(Sigma_inv_mu)

# Checking shape of array
d.shape
```

Out[29]:
```
(9, 1)
```

In [30]:
```python
# Calculating F = ell'*Sigma^{-1}*mu
F = np.dot(ell.T, Sigma_inv_mu)

# Calculate e = (1/gamma)*(F/C)*Sigma^{-1}*ell
e = (1/gamma)*(F/C)*(Sigma_inv_ell)

# Checking shape of array
e.shape
```

Out[30]:
```
(9, 1)
```

In [31]:
```python
#Define Bmu
Bmu = d - e
```

$BE{t}[\tilde{r}{t+1}]$ properties

- As $w_t^{(GMVP)}$ always sums to one, one expects $BE_t[\tilde{r}_{t+1}]$ to sum to zero to ensure that $w_t^{(2)}$ will also sum to one
- $\sum BE_t[\tilde{r}_{t+1}] \overset{n}{\to} 0$ for larger in sample periods

We need a finite sample correction step where we either:

- Shift $BE_t[\tilde{r}_{t+1}]$ so that it sums to zero (Bmu -= Bmu.sum()/Bmu.size())
- Scale $w_t^{(2)}$ so that is sums to one (w_p2 /= w_p2.sum())

Option 1 was chosen, because option 2 modifies GMVP

In [32]:
```python
print(f"{Bmu.sum()=}")
Bmu -= Bmu.sum()/Bmu.size # Correction
print(f"{Bmu.sum()=}")
```

```
Bmu.sum()=-0.24588120593035664
Bmu.sum()=-6.938893903907228e-18
```

Define weights vector

In [33]:
```python
w_p2_is = w_gmvp_is + Bmu # Calculating weights vector: w = a + BE[r_{t}], recognising a as the weights under the GMVP
print(w_p2_is)
print('Sum of weights:', w_p2_is.sum())
```

```
[[-0.063146  ]
 [ 0.18209668]
 [-0.30611104]
 [-0.18525003]
 [-0.15328241]
 [ 0.58888639]
 [ 0.18272211]
 [ 0.65617231]
 [ 0.097912  ]]
Sum of weights: 1.0
```

In [34]:
```python
mu_p2_is = np.dot(mu.T, w_p2_is)                        # In-sample portfolio return
var_p2_is = np.dot(w_p2_is.T, np.dot(Sigma, w_p2_is))      # In-sample portfolio variance
sigma_p2_is = np.sqrt(var_p2_is)                        # In-sample portfolio std dev
```

Out-of-Sample Portfolio 2

In [35]:
```python
# Excess return of ETF = alpha + beta*excess returns on market

# SP500 excess returns within out of sample period
SP500_exret_os = df.loc[os_start_date:,'SP500'] - df.loc[os_start_date:,'rf']

# Constant beta calculated over in sample period applied to out of sample data
beta_os = [beta for i in range(T_os)]

# Reorder SP500 excess return list to 2d array
SP500_exret_os = np.array(SP500_exret_os)
SP500_exret_os = SP500_exret_os[:, np.newaxis]

# CAPM calculation - Alpha should be zero
exret_os = beta_os*SP500_exret_os
exret_df_os = pd.DataFrame(data=exret_os, columns=ETF_list, index=dates_os)
display(Markdown("**CAPM expected excess returns of ETFS over Out-of-Sample Period**"))
display(exret_df_os.head())
```

CAPM expected excess returns of ETFS over Out-of-Sample Period

Date	XLB	XLE	XLF	XLI	XLK	XLP	XLU	XLV	XLY
2004-02-27	0.011485	0.007485	0.010392	0.011399	0.021474	0.003408	0.004872	0.008993	0.011834
2004-03-31	-0.017318	-0.011287	-0.015671	-0.017190	-0.032382	-0.005140	-0.007346	-0.013561	-0.017845
2004-04-30	-0.017656	-0.011507	-0.015976	-0.017525	-0.033013	-0.005240	-0.007490	-0.013825	-0.018193
2004-05-31	0.011361	0.007405	0.010280	0.011277	0.021243	0.003372	0.004819	0.008896	0.011707
2004-06-30	0.016955	0.011050	0.015342	0.016829	0.031702	0.005032	0.007192	0.013276	0.017470

In [36]:
```python
exret_os_mu_1d = exret_os.mean(0)
exret_os_mu = exret_os_mu_1d[:, np.newaxis]
exret_os_mu
```

Out[36]:
```
array([[0.00442096],
       [0.00288139],
       [0.00400036],
       [0.00438817],
       [0.00826629],
       [0.00131201],
       [0.00187537],
       [0.00346173],
       [0.00455538]])
```

In [37]:
```
# Out-of-Sample Port 2
w_p2_os = np.zeros((N,T_os))                              # row=stock, column=time point
r_p2_os = np.empty(T_os)
for i in range(T_os):
    Sigma_os = np.cov(rets_all[i:T_in+i,:].T)            # Using [t1,t2] to estimate Sigma ([i,60+i], for i = 0, 1, ...)
    Sigma_inv_ell_os = np.linalg.solve(Sigma_os, ell)    # Storing Sigma^{-1}*ell
    C = np.dot(ell.T, Sigma_inv_ell_os)                  # Calculating C = ell'*Sigma^{-1}*ell
    w_gmvp_os = (1/C)*Sigma_inv_ell_os                   # Calculating weights vector: w = (1/C)*Sigma^{-1}*ell
    Sigma_inv_mu = np.linalg.solve(Sigma, exret_os_mu)   # Solving Sigma^{-1}*mu
    d = (1/gamma)*(Sigma_inv_mu)                         # Calculate d
    F = np.dot(ell.T, Sigma_inv_mu)                      # Calculating F = ell'*Sigma^{-1}*mu
    e = (1/gamma)*(F/C)*(Sigma_inv_ell)                  # Calculate e = (1/gamma)*(F/C)*Sigma^{-1}*ell
    Bmu = d - e                                          # Define Bmu
    Bmu -= Bmu.sum() / Bmu.size                          # Finite sample correction to ensure Bmu.sum()==0
    w_p2_os[:,i] = (w_gmvp_os + Bmu).T                   # Calculating weights vector: w = a + BE[r_{t}],
                                                         # recognising a as the weights under the GMVP
                                                         # Storing weights vector
    r_p2_os[i] = np.dot(rets_all[T_in+i,:], w_p2_os[:,i])  # Out-of-sample portfolio returns

var_p2_os = np.dot(w.T, np.dot(Sigma_os, w))             # Out-of-sample portfolio variance
sigma_p2_os = np.sqrt(var_p2_os)                         # Out-of-sample portfolio std dev
mu_p2_os = r_p2_os.mean()                                # Out-of-sample portfolio average returns
```

Portfolio Evaluation

In [38]:
```
# Appending MVP and EW portfolios to Out-of-Sample dataframe

df_os['GMVP'] = r_mvp_os
df_os['EW'] = df_os.loc[:,'XLB':'XLY'].mean(axis = 1)
df_os['P2'] = r_p2_os
df_os['rf'] = df.loc[os_start_date:,'rf']
df_os['SP100'] = df.loc[os_start_date:,'SP100']
df_os['SP500'] = df.loc[os_start_date:,'SP500']
ports_os = df_os[['GMVP', 'EW', 'P2', 'SP500', 'SP100']]
```

In [39]:
```
returns_describe(ports_os, 'ETF: Out of Sample Portfolio')
```

ETF: Out of Sample Portfolio Returns Characteristics

	GMVP	EW	P2	SP500	SP100
count	191.000000	191.000000	191.000000	191.000000	191.000000
mean	0.008934	0.008301	0.009074	0.005495	0.004950
std	0.031207	0.038528	0.032691	0.039518	0.038455
min	-0.127474	-0.169122	-0.139569	-0.185636	-0.157717
25%	-0.003843	-0.013072	-0.005327	-0.015415	-0.013428
50%	0.010272	0.012844	0.012636	0.011510	0.011257
75%	0.028543	0.030656	0.029418	0.029306	0.026708
max	0.074518	0.112622	0.084023	0.102307	0.092940
skew	-1.012186	-0.819259	-1.087235	-1.044418	-0.913286
kurt	5.539734	5.717364	5.890867	6.003233	4.948659

In [40]:
```
# Calculcate excess returns
cols = ['GMVP', 'EW', 'P2', 'SP100', 'SP500']
df_os_exret = df_os.apply(lambda x: x[cols] - x['rf'], axis=1)
df_os_exret.columns = [ 'exret' + str(col) for col in cols]
df_os = pd.concat([df_os,df_os_exret], axis = 1)
```

Gains from Diversification

The easiest way to do it is to estimate the expected return and variance of two efficient frontier portfolios and the covariance between these two efficient frontier portfolios. Then use the two fund theorem to sweep out the entire frontier as linear combinations of the two efficient portfolio weights. You can use code from the web also but just reference your source.

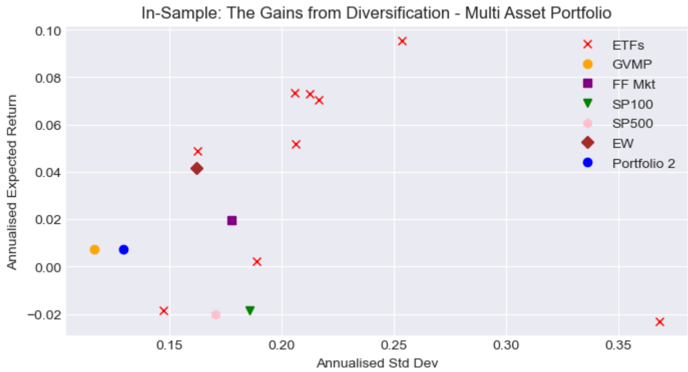
In [41]:
```
# In sample Fama French mean market return and std_dev (for plotting)

# Isolate array of FF market returns over in sample period
ff_mkt_rf = df.loc[:is_end_date,'mkt-rf'].values
rf = df.loc[:is_end_date,'rf'].values
ff_mkt_ret = ff_mkt_rf + rf

# In-sample expected returns vector and standard deviation vector
ff_mu = ff_mkt_ret.mean()
ff_sigma = ff_mkt_ret.std()
```

In [42]:
```
# In sample SP100 mean market return and std_dev (for plotting)

sp100 = df.loc[:is_end_date,'SP100'].values
sp100_mu = sp100.mean()
sp100_sigma = sp100.std()
```

In [43]:
```
# In sample SP500 mean market return and std_dev (for plotting)

sp500 = df.loc[:is_end_date,'SP500'].values
sp500_mu = sp500.mean()
sp500_sigma = sp500.std()
```

In [44]:
```
# In sample Equally Weighted portfolio return and std_dev (for plotting)

ew_port = df_is.loc[:,'XLB':'XLY'].mean(axis = 1)
ew_port_mu = ew_port.mean()
ew_port_sigma = ew_port.std()
```

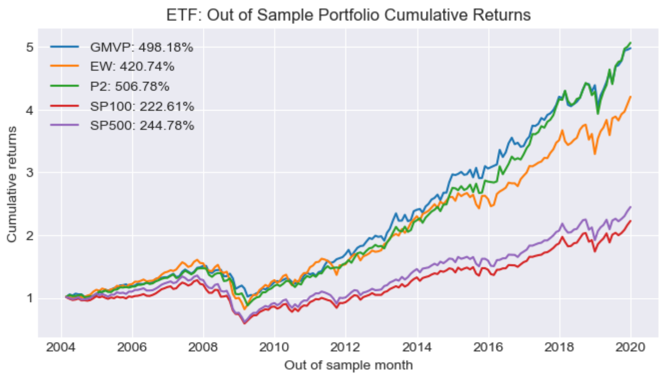
In [45]:
```
# Recreate graph with SP100 constituents assets, SP100, EW, FF, GMVP, portfolio returns (In sample)
plt.plot(sigma*np.sqrt(12), mu_1d*12, 'x', label='ETFs', color='red')
plt.plot(sigma_gmvp_is*np.sqrt(12), mu_gmvp_is*12, 'o', label='GVMP', color='orange')
plt.plot(ff_sigma*np.sqrt(12), ff_mu*12, 's', label='FF Mkt', color='purple')
plt.plot(sp100_sigma*np.sqrt(12), sp100_mu*12, 'v', label='SP100', color='green')
plt.plot(sp500_sigma*np.sqrt(12), sp500_mu*12, 'h', label='SP500', color='pink')
plt.plot(ew_port_sigma*np.sqrt(12), ew_port_mu*12, 'D', label='EW', color='brown')
plt.plot(sigma_p2_is*np.sqrt(12), mu_p2_is*12, 'o', label='Portfolio 2', color='blue')

plt.xlabel('Annualised Std Dev')
plt.ylabel('Annualised Expected Return')
plt.title('In-Sample: The Gains from Diversification - Multi Asset Portfolio')
plt.legend()
plt.show()
```



Cumulative Returns

In [46]:
```
# Cumulative returns

cr_mvp = np.cumprod(1+df_os['GMVP'])
cr_ew = np.cumprod(1+df_os['EW'])
cr_p2 = np.cumprod(1+df_os['P2'])
cr_sp100 = np.cumprod(1+df.loc[os_start_date:,'SP100'])
cr_sp500 = np.cumprod(1+df.loc[os_start_date:,'SP500'])
```

In [47]:
```
# Plot

plt.plot(cr_mvp, label = f'GMVP: {cr_mvp[-1]*100:.2f}%')
plt.plot(cr_ew, label = f'EW: {cr_ew[-1]*100:.2f}%')
plt.plot(cr_p2, label = f'P2: {cr_p2[-1]*100:.2f}%')
plt.plot(cr_sp100, label = f'SP100: {cr_sp100[-1]*100:.2f}%')
plt.plot(cr_sp500, label = f'SP500: {cr_sp500[-1]*100:.2f}%')
plt.xlabel('Out of sample month')
plt.ylabel('Cumulative returns')
plt.title('ETF: Out of Sample Portfolio Cumulative Returns')
plt.legend()
plt.show()
```



Capital Asset Pricing Model

Use the CAPM model to evaluate the out-of-sample performance of the two portfolios. That is, run an ex-post regression with the excess portfolio returns as the dependent variable and the excess S&P 100 (S&P 500) market returns as the independent variable.

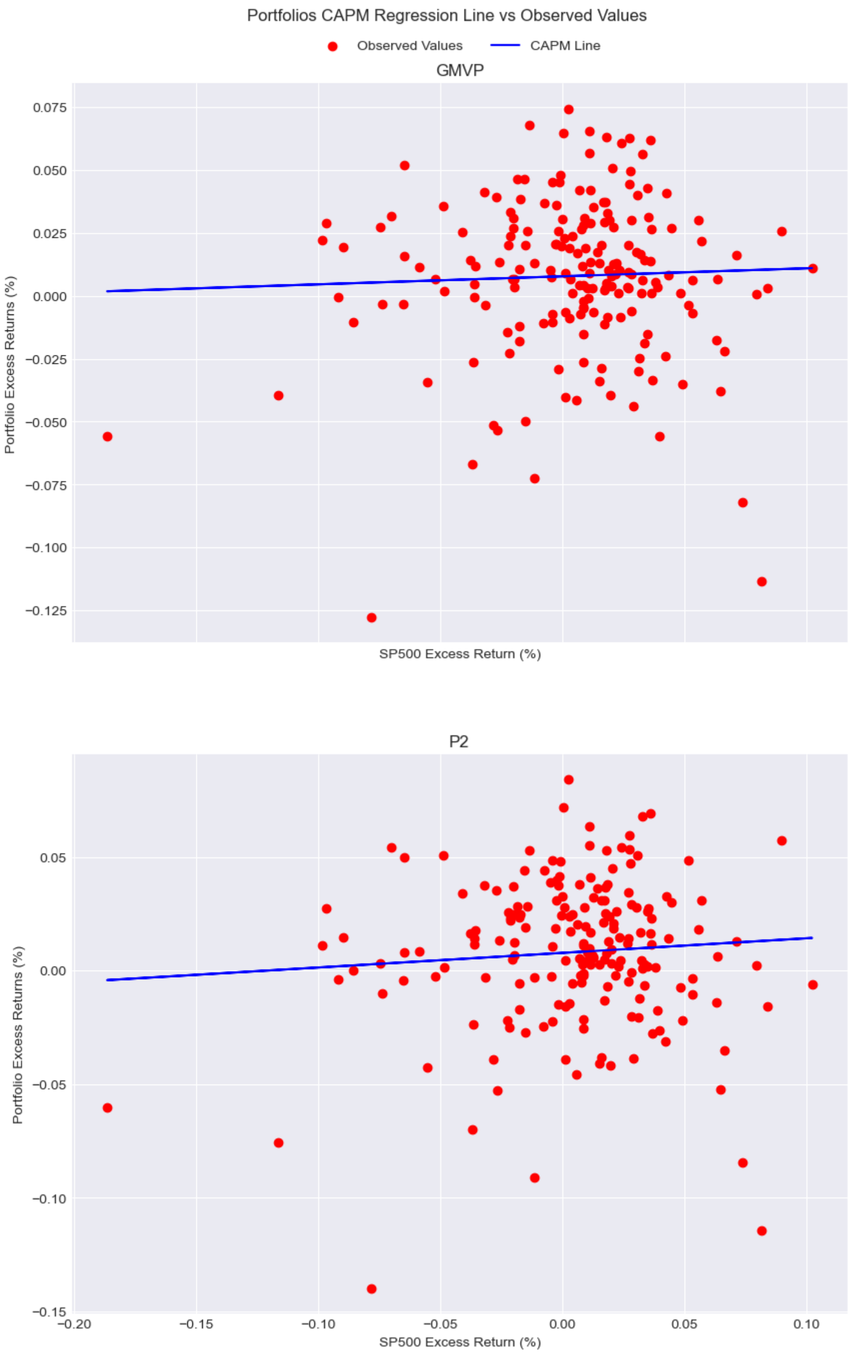
In [48]:
```
df_os_CAPM = df_os[['exretGMVP', 'exretP2', 'exretSP100', 'exretSP500']] # Define new dataframe
portfolios = ['GMVP', 'P2'] # Define list of portfolios
```

In [49]:
```
# SP500 CAPM Regression

sp500_capm_model_result = {} #Setting up empty dictionary

for p in portfolios:
    Y = df_os_CAPM['exret' + p] #dependent variables
    X = df_os_CAPM['exretSP500'] #independent variable
    SP500_capm_model = smf.ols(formula='Y ~ X',data=df_os_CAPM) #estimate each OLS regression
    SP500_capm_name = 'SP500 CAPM Ex-Post Regression: ' + p #name each regression
    sp500_capm_regress_result = SP500_capm_model.fit() #fit each model
    print(SP500_capm_name, '\n', sp500_capm_regress_result.summary(), '\n') #print each regression

    sp500_capm_model_result[p] = sp500_capm_regress_result  #Filling dictionary with each model
```

```
SP500 CAPM Ex-Post Regression: GMVP
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.002
Model:                            OLS   Adj. R-squared:                 -0.004
Method:                 Least Squares   F-statistic:                    0.3084
Date:                Tue, 06 Dec 2022   Prob (F-statistic):              0.579
Time:                        00:21:16   Log-Likelihood:                 391.40
No. Observations:                 191   AIC:                            -778.8
Df Residuals:                     189   BIC:                            -772.3
Df Model:                           1                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.0077      0.002      3.391      0.001       0.003       0.012
X              0.0319      0.057      0.555      0.579      -0.081       0.145
==============================================================================
Omnibus:                       37.560   Durbin-Watson:                   2.184
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               71.725
Skew:                          -0.954   Prob(JB):                     2.66e-16
Kurtosis:                       5.318   Cond. No.                         25.3
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

SP500 CAPM Ex-Post Regression: P2
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.006
Model:                            OLS   Adj. R-squared:                  0.001
Method:                 Least Squares   F-statistic:                     1.148
Date:                Tue, 06 Dec 2022   Prob (F-statistic):              0.285
Time:                        00:21:16   Log-Likelihood:                 382.94
No. Observations:                 191   AIC:                            -761.9
Df Residuals:                     189   BIC:                            -755.4
Df Model:                           1                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.0077      0.002      3.242      0.001       0.003       0.012
X              0.0643      0.060      1.071      0.285      -0.054       0.183
==============================================================================
Omnibus:                       41.424   Durbin-Watson:                   2.164
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               84.820
Skew:                          -1.016   Prob(JB):                     3.82e-19
Kurtosis:                       5.555   Cond. No.                         25.3
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [50]:

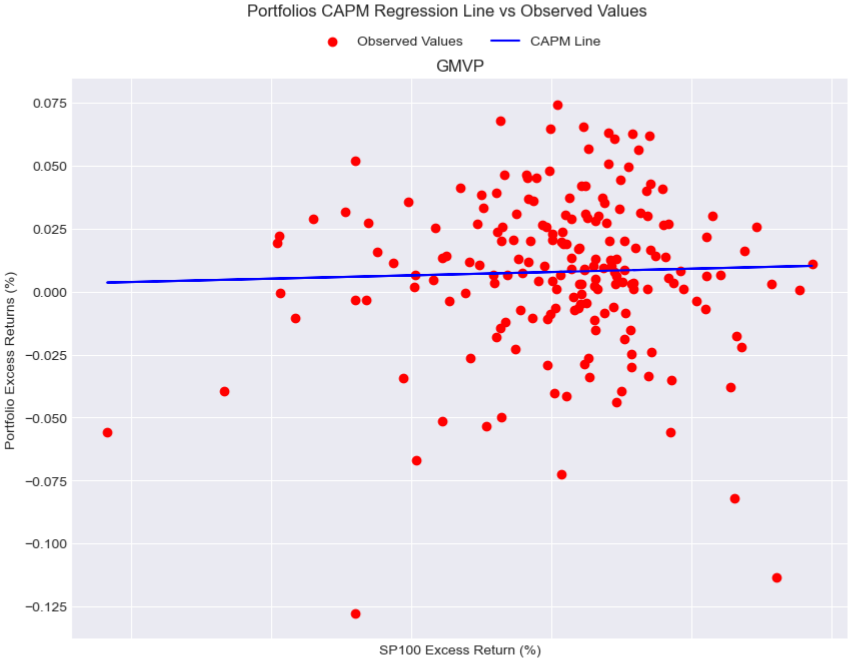
```python
# Plot CAPM
fig, axs = plt.subplots(nrows=len(portfolios), ncols=1, figsize=(10,15), sharex=True)

for k, p in enumerate(portfolios):
    ax = axs[k]
    predict = sp500_capm_model_result[p].predict()
    ax.scatter(df_os_CAPM['exretSP500'], df_os_CAPM['exret' + p],
               label='Observed Values', color = 'red') # plot observed values
    ax.plot(df_os_CAPM['exretSP500'], predict, label = 'CAPM Line', color = 'blue') # draw regression line
    ax.set_title(p)
    ax.set_xlabel("SP500 Excess Return (%)")
    ax.set_ylabel("Portfolio Excess Returns (%)")
plt.suptitle("Portfolios CAPM Regression Line vs Observed Values")
axs[0].legend(loc='upper center', bbox_to_anchor=(0.5, 1.1),ncol=2, fancybox=True, shadow=True) #Adjusting legend position
plt.subplots_adjust(top=0.93) #Adjusting title position
plt.show()
```

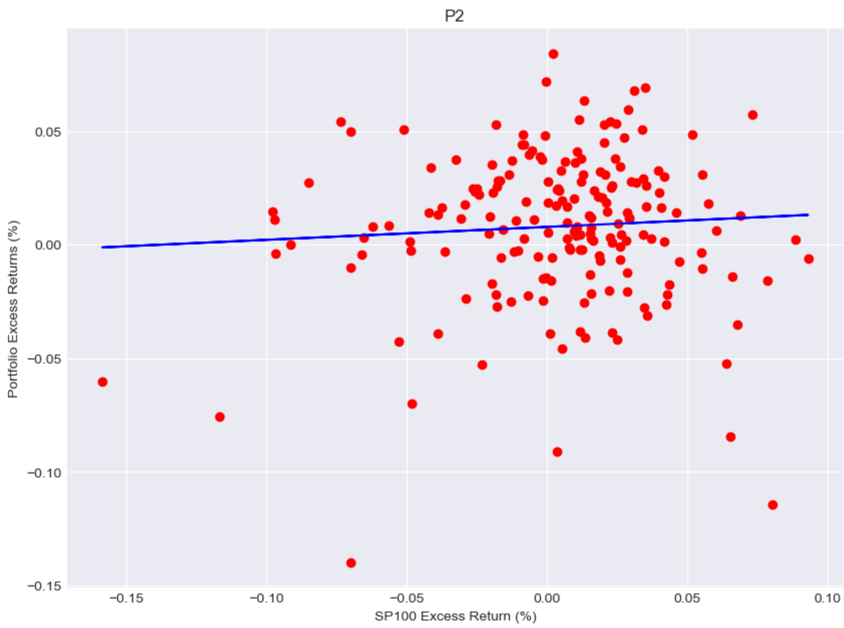


In [51]:
```
# SP100 CAPM Regression

sp100_capm_model_result = {} #Setting up empty dictionary

for p in portfolios:
    Y = df_os_CAPM['exret' + p] #dependent variables
    X = df_os_CAPM['exretSP100'] #independent variable
    SP100_capm_model = smf.ols(formula='Y ~ X',data=df_os_CAPM) #estimate each OLS regression
    SP100_capm_name = 'SP100 CAPM Ex-Post Regression: ' + p #name each regression
    sp100_capm_regress_result = SP100_capm_model.fit() #fit each model
    print(SP100_capm_name, '\n', sp100_capm_regress_result.summary(), '\n') #print each regression

    sp100_capm_model_result[p] = sp100_capm_regress_result  #Filling dictionary with each crypto model
```

```
SP100 CAPM Ex-Post Regression: GMVP
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.001
Model:                            OLS   Adj. R-squared:                 -0.004
Method:                 Least Squares   F-statistic:                    0.2003
Date:                Tue, 06 Dec 2022   Prob (F-statistic):              0.655
Time:                        00:21:16   Log-Likelihood:                 391.34
No. Observations:                 191   AIC:                            -778.7
Df Residuals:                     189   BIC:                            -772.2
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.0078      0.002      3.411      0.001       0.003       0.012
X              0.0264      0.059      0.448      0.655      -0.090       0.143
==============================================================================
Omnibus:                       37.777   Durbin-Watson:                   2.175
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               72.421
Skew:                          -0.958   Prob(JB):                     1.88e-16
Kurtosis:                       5.331   Cond. No.                         26.0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

SP100 CAPM Ex-Post Regression: P2
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.004
Model:                            OLS   Adj. R-squared:                 -0.001
Method:                 Least Squares   F-statistic:                    0.8470
Date:                Tue, 06 Dec 2022   Prob (F-statistic):              0.359
Time:                        00:21:16   Log-Likelihood:                 382.79
No. Observations:                 191   AIC:                            -761.6
Df Residuals:                     189   BIC:                            -755.1
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.0078      0.002      3.270      0.001       0.003       0.013
X              0.0568      0.062      0.920      0.359      -0.065       0.179
==============================================================================
Omnibus:                       41.908   Durbin-Watson:                   2.149
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               86.710
Skew:                          -1.023   Prob(JB):                     1.48e-19
Kurtosis:                       5.590   Cond. No.                         26.0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

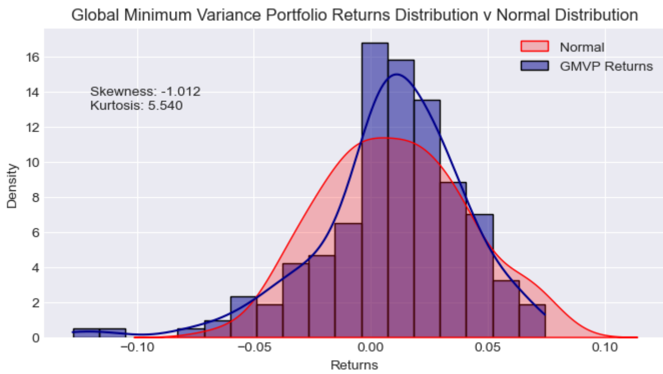
In [52]:
```python
# Plot CAPM
fig, axs = plt.subplots(nrows=len(portfolios), ncols=1, figsize=(10,15), sharex=True)

for k, p in enumerate(portfolios):
    ax = axs[k]
    predict = sp100_capm_model_result[p].predict()
    ax.scatter(df_os_CAPM['exretSP100'], df_os_CAPM['exret' + p],
               label='Observed Values', color = 'red') # plot observed values
    ax.plot(df_os_CAPM['exretSP100'], predict, label = 'CAPM Line', color = 'blue') # draw regression line
    ax.set_title(p)
    ax.set_xlabel("SP100 Excess Return (%)")
    ax.set_ylabel("Portfolio Excess Returns (%)")
plt.suptitle("Portfolios CAPM Regression Line vs Observed Values")
axs[0].legend(loc='upper center', bbox_to_anchor=(0.5, 1.1),ncol=2, fancybox=True, shadow=True) #Adjusting legend position
plt.subplots_adjust(top=0.93) #Adjusting title position
plt.show()
```

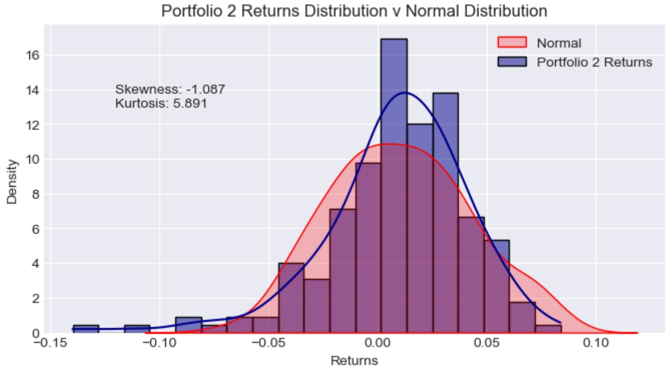


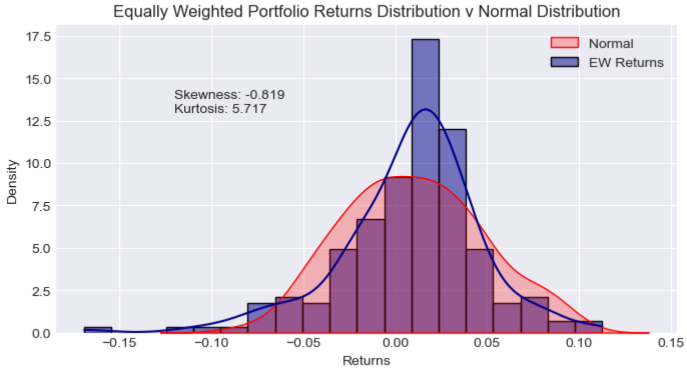


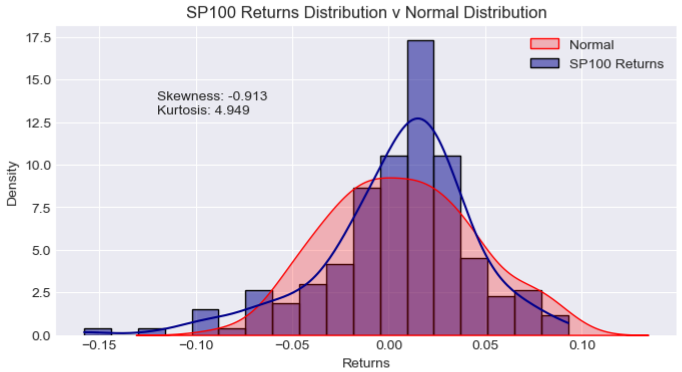
Porfolio Returns Distribution

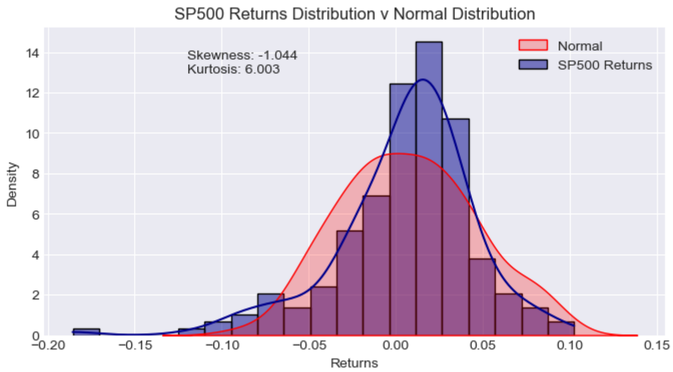
In [53]:
```
plot_series_v_normal(df_os['GMVP'], 'GMVP Returns',
                     'Global Minimum Variance Portfolio Returns Distribution v Normal Distribution')
plot_series_v_normal(df_os['P2'], 'Portfolio 2 Returns',
                     'Portfolio 2 Returns Distribution v Normal Distribution')
plot_series_v_normal(df_os['EW'], 'EW Returns',
                     'Equally Weighted Portfolio Returns Distribution v Normal Distribution')
plot_series_v_normal(df_os['SP100'], 'SP100 Returns',
                     'SP100 Returns Distribution v Normal Distribution')
plot_series_v_normal(df_os['SP500'], 'SP500 Returns',
                     'SP500 Returns Distribution v Normal Distribution')
```




Portfolio 2 Returns Distribution v Normal Distribution
Normal
Portfolio 2 Returns
Skewness: -1.087
Kurtosis: 5.891
Density
Returns


Equally Weighted Portfolio Returns Distribution v Normal Distribution
Normal
EW Returns
Skewness: -0.819
Kurtosis: 5.717
Density
Returns


SP100 Returns Distribution v Normal Distribution
Normal
SP100 Returns
Skewness: -0.913
Kurtosis: 4.949
Density
Returns


SP500 Returns Distribution v Normal Distribution
Normal
SP500 Returns
Skewness: -1.044
Kurtosis: 6.003
Density
Returns