S&P 500 Risk Optimizations Forecast

Contact:

Website LinkedIn Github Portfolio Business Mail

EstebanMqz

EstebanMqz S&P500Risk Optimizations Forecast

Introduction:

Abstract:

Time series modelling is a powerful forecast tool and the stock market tends to be an interesting example because statistical estimators are of special interest.
They are used for general prediction purposes and to make decision-making processes more efficient.
The industries where it can be applied are numerous, but the most common are the following:

- Government
- Banking
- Insurance
- Energy
- Healthcare
- Telecommunications
- Retail
- Education

Since Covid until the present day, data has changed with none to few exceptions, the markets are a good example.
For that reason, from symbols data that integrate the S&P 500 index of the United States, the following processes will be made:

Description:

Detailed Descriptive Statistics techniques from $x_i \in [x_1, x_{500}] \hookrightarrow S\&P500$, in addition to theoretical and
experimental demonstrations with extracted data to establish the usage of $ln(1 + r_t)$ in regards to price returns $\frac{Pt+1}{Pt}$ during the period.

Moreover, estimators and resamplings $x_i \in [x_1, x_{n=25}] \hookrightarrow R_{Sortino_{+25}}$ are modelled in order to have insights from their periodiodicity.

Consequently, optimizations $max_{\vec{w_{j \neq i}}} R_{Sortino_{+25}} \models max|min_{\vec{w_{j \neq i}}} R_k$ are performed and analyzed from a big picture perspective
/ With this in mind, new fluctuations can be integrated in the optimization model that best fits the needs by the time and unfavourable conditions can be anticipated by making w_i adjustments that might escape the initially proposed objective functions and constraints from the whole population.

Finally, with what would have been its past behavior, the following simulations are made:

$$\sum_{j \neq i}^{n} x_i \sim X_i \hookrightarrow max_{\vec{w_{j \neq i}}} R_{Sortino_{+25}}$$

As conclusion, forecasts $X_{(t_1+t_2+..+t_n)} \hookrightarrow max_{\vec{w_{j \neq i}}} R_{Sortino_{+25}}$ are obtained in daily basis.

Table of Contents:

It is divided in the folllowing sections:

0. Virtual Environment:

0.1 Load Dependencies:

In [2]:
```
import functions as fn
import data as dt
import visualizations as vs
```

0.2 Install Libs. & Modules:

Project Creators:

Create `requirements.txt` file:

fn.get_requirements:

Skip to installation if you are not interested in contributing to the project.

In [19]:
```
docstring = """
# -- ---------------------------------------------------------------------------------- -- #
# -- project: S&P500-Risk-Optimized-Portfolios-PostCovid-ML                              -- #
# -- script: requirements.txt: txt file to download Python modules for execution            -- #
# -- author: EstebanMqz                                                                    -- #
# -- license: CC BY 3.0                                                                    -- #
# -- repository: SP500-Risk-Optimized-Portfolios-PostCovid-ML/blob/main/requirements.txt     -- #
# -- ---------------------------------------------------------------------------------- -- #
\n
"""

path = fn.get_requirements(docstring)
```

```
requirements.txt file created in local path: c:\Users\Esteban\Desktop\Projects\Github\Repos_To-do\Languages\Python\Fin_Sim\Projects\SP500-Risk-Optimized-Portfolios-ML\requirements.txt
```

Project Users:

Install packages in created `requirements.txt` file:

fn.library_install:

In [3]:
```
fn.library_install("requirements.txt")
```

```
Requirements installed.

# -- ----------------------------------------------------------------------------
--------- -- #
# -- project: S&P500-Risk-Optimized-Portfolios-PostCovid-ML
-- #
# -- script: requirements.txt: txt file to download Python modules for execution
-- #
# -- author: EstebanMqz
-- #
# -- license: CC BY 3.0
-- #
# -- repository: SP500-Risk-Optimized-Portfolios-PostCovid-ML/blob/main/requiremen
ts.txt    -- #
# -- ----------------------------------------------------------------------------
--------- -- #


numpy >= 1.22.4
pandas >= 1.4.4
matplotlib >= 3.5.3
scipy >= 1.7.3
sklearn >= 1.0.2
logging >= 0.5.1.2
jupyter >= 1.0.0
yahoofinanicals >= 1.14
tabulate >= 0.8.9
IPython >= 8.12.0
fitter >= 1.5.2
```

0.3 Load Libraries & Modules:

In [3]:
```
import numpy as np
import pandas as pd
pd.set_option("display.max_rows", None, "display.max_columns", None
              ,"display.max_colwidth", None, "display.width", None)

import matplotlib
import matplotlib.pyplot as plt
```

```
plt.style.use("dark_background")
%matplotlib inline
import plotly.graph_objects as go
import plotly.express as px
import seaborn as sns

import scipy
import scipy.stats as st
from scipy import optimize
from scipy.optimize import minimize

import sklearn
from sklearn.neighbors import KernelDensity
from sklearn.model_selection import GridSearchCV
from sklearn import metrics

from statsmodels.tsa.stattools import pacf
from statsmodels.tsa.stattools import acf
import statsmodels.api as sm



from yahoofinancials import YahooFinancials
from tabulate import tabulate
import IPython.display as d
import IPython.core.display

import datetime
import time

from io import StringIO
from fitter import Fitter, get_common_distributions, get_distributions
import logging
import ast

import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings("ignore", category=UserWarning)
```

1. Data Extraction and Exploration:

1.1 Data Extraction:

In this section $x_i \in [x_1, x_{500}] \hookrightarrow S\&P500$ quotes are fetched:

Fetching a lot of data from Yahoo Finance by batches is required to avoid host disruptions (other sources could be used).

fn.SP500_tickers:

In [4]:
```
tickers = fn.SP500_tickers(50)
tickers[0][:5], tickers[-1][0:5], sum([len(i) for i in tickers])
```

Out[4]:
```
(['MMM', 'AOS', 'ABT', 'ABBV', 'ACN'], ['ZBH', 'ZION', 'ZTS'], 503)
```

Note: Skip to 1.1.2 if you prefer using .csv creation date.

$6_Y\ x_i \in [x_1, x_{500}] \hookrightarrow S\&P500$ Adj. closes are fetched *(5min)* :

dt.get_historical_price_data:

In [7]:
```
SP_Assets_f = pd.concat([dt.get_historical_price_data(tickers[i][j], 6)
                          for i in range(0, len(tickers)) for j in
range(0, len(tickers[i]))], axis=1)
SP_Assets_f.shape
```

Out[7]:
```
(1509, 503)
```

In [8]:
```
SP_Assets_f.shape
```

Out[8]:
```
(1509, 503)
```

Adj. closes for $S\&P500$:

In [10]:
```
SP_f = dt.get_historical_price_data('^GSPC', 6)
SP_f = SP_f[SP_f.index.isin(SP_Assets_f.index)]
SP_f.shape
```

Out[10]:
```
(1509, 1)
```

Fetched data is saved in *Data* subdirectory:

- `Assets_SP500.csv`

- `SP500_index.csv`

In [ ]:
```
SP_Assets_f.to_csv("Data/Assets_SP500.csv")
SP_f.to_csv("Data/SP500.csv")
SP_f = pd.read_csv("Data/SP500.csv", index_col=0)
SP_Assets_f = pd
```

Fetched x_i data:

In [8]:
```
SP_Assets_f.head(8)
```

Out[8]:

formatted_date	MMM	AOS	ABT	ABBV	ACN	ATVI	ADM	
2017-05-23	160.652100	48.843098	39.439041	50.427128	111.656731	56.066231	36.417500	13
2017-05-24	160.457123	48.987476	39.303604	50.496056	111.473907	56.754932	36.002884	14
2017-05-25	162.122711	49.176952	39.682823	50.794758	112.525116	57.443634	36.062119	14
2017-05-26	163.040833	48.788952	40.369038	50.595627	112.333145	56.531834	35.918282	14
2017-05-30	164.478912	49.068672	40.630878	50.564991	113.137566	56.822842	35.630581	14
2017-05-31	166.128220	49.510818	41.226795	50.564991	113.777443	56.822842	35.182148	14
2017-06-01	166.030731	49.844673	41.624084	51.093464	114.527000	57.773445	35.385220	14
2017-06-02	167.940079	50.756020	41.985241	51.507053	114.938332	57.860744	35.723663	14

In [9]:
```
SP_Assets_f.tail(8)
```

Out[9]:

formatted_date	MMM	AOS	ABT	ABBV	ACN	ATVI	ADM	
2023-05-10	99.389221	69.220001	110.690002	146.419998	268.890015	76.000000	74.198402	3
2023-05-11	99.271019	68.400002	110.050003	146.589996	272.269989	77.040001	74.456863	3
2023-05-12	98.768646	67.239998	110.489998	147.149994	277.190002	77.370003	74.934021	3
2023-05-15	98.985359	68.199997	109.839996	146.589996	277.510010	78.330002	75.610001	3
2023-05-16	96.542496	67.209999	109.389999	143.289993	279.190002	77.779999	73.150002	3
2023-05-17	98.680000	68.279999	108.820000	143.350006	284.630005	77.889999	73.050003	3
2023-05-18	99.639999	69.139999	108.470001	143.440002	287.480011	78.190002	72.790001	3
2023-05-19	99.029999	68.419998	108.930000	145.110001	289.910004	78.589996	73.230003	3

1.1.2 Data Read

To skip data fetching if needed, a data reader is made available:

In [4]:
```
SP_r = pd.read_csv("Data/SP500.csv", index_col=0)
SP_Assets_r = pd.read_csv("Data/Assets_SP500.csv", index_col=0)
```

A Data Quality Report for original Data 6_y is shown:

`dt.DQR`

In [5]:
```
DQR = dt.DQR(SP_Assets_r).sort_values(by='Missing_Values',
ascending=False)
DQR.T
```

Out[5]:

Columns	GEHC	CEG	OGN	CARR	OTIS	CTVA	DOW	FOX	FOXA	MRNA
Data_Type	float64	float64	float64	float64	float64	float64	float64	float64	float64	float64
Unique_Values	100	330	472	764	766	960	1009	961	968	1065
Missing_Values	1402	1173	1001	710	710	504	458	453	452	389
Zero_Values	0	0	0	0	0	0	0	0	0	0
Outliers	0	0	0	0	0	0	0	0	0	15
Unique_Outliers	0	0	0	0	0	0	0	0	0	15

1.2 Data Exploration

Defining Returns:

Accumulated Simple and Log Returns:

- *Multiplicative - Additive*:

R_t are multiplicative because the following is true to calculate their accumulated value by compound interest:

$$R_t = \left[\left(\prod_{t=1}^{n} \frac{P_{t+1}}{P_t} \right) - 1 \right]$$

$ln(r_t)$ are additive because of the exponential law: $\mathrm{e}^{P_t \times P_{t+1}} = \mathrm{e}^{P_t + P_{t+1}}$ which makes the following true:

$$\sum_{t=1}^{n} ln(r_t) = \mathrm{e}^{\sum ln(r_t)} - 1$$

$|ln(r_t) \pm R_t| \to 1 \quad \because \quad n \to \infty.$

Simple and Log Returns Characteristics are the following:

- *Not Symmetric - Symmetric*:
  R_t distribution can have $\pm$ skew which makes the Mo, median and μ not centered in $f(x)$.
  $ln(r_t)$ distribution is symmetrical which makes the Mo, median and μ centered in $f(x)$.

- *Bounded - Unbounded*:
  R_t bounds are inclusive between ± 1
  $ln(r_t)$ bounds are $\pm\infty$.

- *Not Stationary - Stationary*:
  R_t has a trend so they are not stationary, nor *i.i.d* and therefore correlated.

  On the other hand $ln(r_t)$ are stationary, so they are *i.i.d* and therefore not correlated.

- *Not Independent - Independent*:
  R_t are not *i.i.d* because they are non stationary, they do have a trend so they are correlated and ultimately multiplicative.
  On the other hand $ln(r_t)$ are *i.i.d* because they are stationary, they do not have a trend so they aren't correlated, which makes them additive.

Just because Simple Returns are correlated, they shouldn't be used to generate continous random variable simulations as they need to be i.i.d.
Log Returns are i.i.d so they can be used to generate continous random variable simulations, it will be proven later on.

Nevertheless, both Returns will be compared $\forall\ x_i \in [x_1, x_{500}] \hookrightarrow S\&P500$ in Data Exploration.
And $\forall\ x_i \in [x_1, x_{n=25}] \hookrightarrow max|min_{\vec{w_{j\neq i}}} R_k$ Descriptive Statistics.

Continous random variables distributions in Scipy for transformed data:

In [6]:
```
continuous = [d for d in dir(st) if isinstance(getattr(st, d),
getattr(st, "rv_continuous"))]
```

```
discrete = [d for d in dir(st) if isinstance(getattr(st, d),
getattr(st, "rv_discrete"))]
pd.DataFrame(continuous).rename(columns={0:"Continuous"}).T
```

Out[6]:

	0	1	2	3	4	5	6	7	8	9	10	1
Continuous	alpha	anglit	arcsine	argus	beta	betaprime	bradford	burr	burr12	cauchy	chi	chi

Discrete random variables distributions in Scipy not considered:

In [7]:
```
pd.DataFrame(discrete).T.rename(index={0:"Discrete"})
```

Out[7]:

	0	1	2	3	4	5	6	7	8	
Discrete	bernoulli	betabinom	binom	boltzmann	dlaplace	geom	hypergeom	logser	nbinom	nc

1.2.1 $S\&P$ 500 Data:

Start Date modified:

- from $(2017-05-23) \rightarrow (2020-03-02)$

Resulting dates:

- from $(2020-03-02) \rightarrow (2023-05-19)$

All quotes Adj. Closes were fetched for the last 6_Y in the $S\&P$ 500 since its execution date on Data Extraction *(1.1)*.
Since required dates are shorter, they will be modified and general variables for the rest of the project declared:

Define variables:

In [34]:
```
rf, best, r_jump, start, end = .00169, 25, 0.05, "2020-03-02",
SP_Assets_r.tail(1).index[0]
prices_start = SP_Assets_r.loc[start:end]
```

Symbols with missing values are located with a DQR for given dates and shown:

`dt.DQR`

In [35]:
```
DQR_start = dt.DQR(prices_start).sort_values(by='Missing_Values',
ascending=False)
DQR_start.head()
```

Out[35]:

Columns	Data_Type	Unique_Values	Missing_Values	Zero_Values	Outliers	Unique_Outliers
GEHC	float64	100	705	0	0	0
CEG	float64	330	476	0	0	0
OGN	float64	472	304	0	0	0
OTIS	float64	766	13	0	0	0
CARR	float64	764	13	0	0	0

In [36]:
```
index_missing = (DQR_start[DQR_start['Missing_Values'] >=
1].index).values
index_missing, index_missing.shape[0]
```

Out[36]:
```
(array(['GEHC', 'CEG', 'OGN', 'OTIS', 'CARR'], dtype=object), 5)
```

498 columns are left by removing missing values. Prices have a new shape defined by *(actual_cols = original_cols - missing_cols)*:

In [37]:
```
prices_start = prices_start.drop(index_missing, axis=1)
len(prices_start.T), sum([len(i) for i in tickers]),
index_missing.shape[0]
```

Out[37]:
```
(498, 503, 5)
```

`dt.data_describe`

- `fn.VaR`

Prices are statistically described, sorted by Total Change during the period and shown horizontally:

In [43]:
```
prices_stats = dt.data_describe(prices_start, 'prices', .00169, start,
end)
prices_stats = prices_stats.T.sort_values(by = 'Total_Change',
ascending = False).T
prices_stats
```

Out[43]:

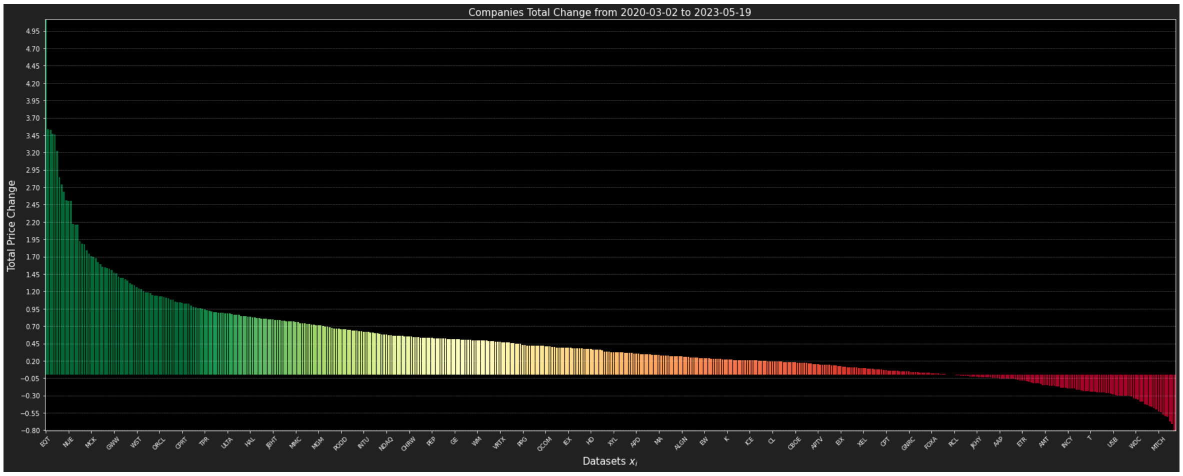
Companies	EQT	NVDA	FSLR	PWR	ON	MRNA	STLD
prices							
min	5.907153	48.951530	30.200001	23.566006	8.450000	21.299999	14.355667
25%	15.630560	130.558857	72.129999	70.172993	31.502501	118.462498	35.405653
50%	20.441086	161.001488	86.075001	106.165596	46.270000	149.470001	60.864223
75%	33.618181	220.079613	114.434999	132.866112	63.880001	180.697498	80.535147
max	49.957397	333.350800	231.690002	174.809998	86.879997	484.470001	135.536499
Mean	24.490194	173.678855	98.593842	100.868067	47.517081	162.176718	61.773341
Yr_Std	11.099876	63.718614	43.411020	39.672720	20.116571	89.729165	28.529020
Total_Change	5.113067	3.537687	3.527920	3.466005	3.456418	3.224900	2.840122
var97.5(-)	47.190206	303.527374	210.131750	166.769331	81.306998	411.051260	121.218307
var2.5(+)	8.470818	65.837145	38.968752	32.075333	13.730000	30.168249	20.844009
Price_skew	0.593010	0.399756	1.103449	-0.208555	-0.073823	1.207771	0.369117
Price_kurtosis	-0.877057	-0.553541	0.623156	-0.987613	-1.117804	1.537299	-0.686932

`vs.cmap_bar`

Total Price Changes $\frac{P_{(2023-05-19)}}{P_{(2020-03-02)}} - 1 \quad \forall \quad x_i \in [x_1, x_{500}] \hookrightarrow S\&P500$ Statistical Descriptions:

In [69]:

```
df_col, df_index = prices_stats.T['Total_Change'],
(prices_stats.T['Total_Change'].index)
x_arange, y_arange = np.arange(0,
prices_stats.T['Total_Change'].index.shape[0], 10),
np.arange(round(prices_stats.T['Total_Change'].min(), 2),
round(prices_stats.T['Total_Change'].max(), 2), .25)
title = (str(prices_stats.T['Total_Change'].index.name) + " Total
Change from " + str(start) + " to " + str(end))
x_label, y_label = "Datasets $x_i$", "Total Price Change"

vs.cmap_bar(df_col, df_index, x_arange, y_arange, title, x_label,
y_label)
```



In [216…

```
Stats_Simple, Simple_ret = data_describe(prices_start, 'Simple', rf,
start, end)[:2]
Stats_Simple = Stats_Simple.T.sort_values(by = 'Yr_Return', ascending
= False).T
Stats_Simple
```

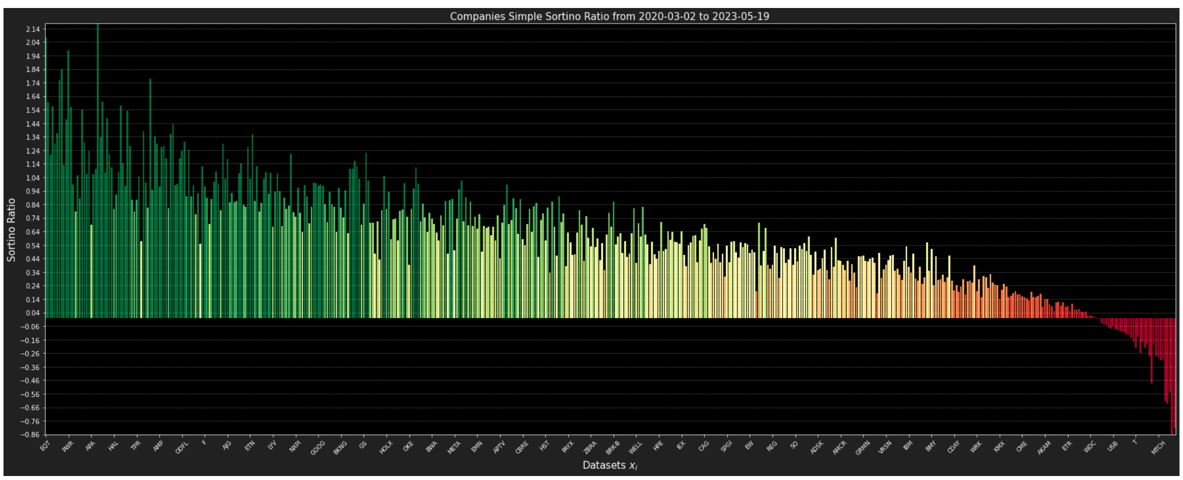
Out[216]:

Companies	EQT	MRNA	ENPH	ON	DVN	TSLA	NVD
Simple							
min	5.907153	21.299999	23.990000	8.450000	4.407207	24.081333	48.9515
25%	15.630560	118.462498	127.905001	31.502501	13.488991	149.851662	130.5588
50%	20.441086	149.470001	170.839996	46.270000	34.782429	218.531662	161.0014
75%	33.618181	180.697498	209.790001	63.880001	54.485501	270.841675	220.0796
max	49.957397	484.470001	336.000000	86.879997	74.357452	409.970001	333.3508
Simple_skew	1.113460	0.432073	-0.024307	0.090499	-0.495730	0.036257	0.0975
Simple_kurtosis	9.333835	2.401019	4.568806	6.803456	9.519589	2.477457	2.2398
Accum_Simple	5.113067	3.224900	2.158949	3.456418	2.742981	2.633711	3.5376
Yr_Return	0.768177	0.759560	0.664292	0.653312	0.650959	0.639953	0.6241
Yr_Std	0.648855	0.794246	0.780860	0.613762	0.685338	0.690983	0.5554
var97.5(-)	0.080761	0.107641	0.099553	0.072557	0.084974	0.092794	0.0714
var2.5(+)	-0.070200	-0.089703	-0.089126	-0.069417	-0.075386	-0.083101	-0.0678
Yr_MaxDrawdown	-0.163584	-0.472458	-0.423201	-0.434585	-0.267167	-0.569167	-0.4212
Sharpe	1.181291	0.954200	0.848553	1.061686	0.947371	0.923703	1.1207
Sortino	2.072235	1.593368	1.206913	1.566658	1.288222	1.364962	1.7621
Calmar	4.695920	1.607676	1.569685	1.503301	2.436527	1.124368	1.4815
Burke	4.685589	1.604099	1.565692	1.499412	2.430202	1.121399	1.4775

Considering Quotes are sorted by Yr_Return, what the following chart shows is basically that even though there are quotes who were highly ranked in terms of Returns, not so much in terms of negative volatility, so people or firms could still have lost.

In [292…

```
df_col, df_index = Stats_Simple.T['Sortino'],
(Stats_Simple.T['Sortino'].index)
x_arange, y_arange = np.arange(0,
Stats_Simple.T['Sortino'].index.shape[0], 10),
np.arange(round(Stats_Simple.T['Sortino'].min(), 2),
round(Stats_Simple.T['Sortino'].max(), 2), .10)
title = (str(Stats_Simple.T['Sortino'].index.name) + " Simple Sortino
Ratio from " + str(start) + " to " + str(end))
x_label, y_label = "Datasets $x_i$", "Sortino Ratio"

vs.cmap_bar(df_col, df_index, x_arange, y_arange, title, x_label,
y_label)
```

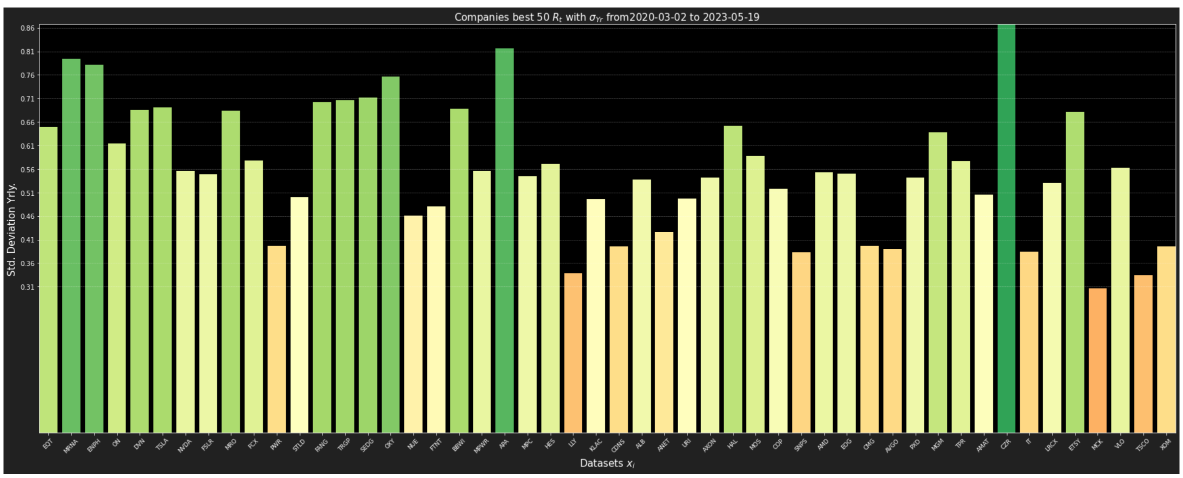


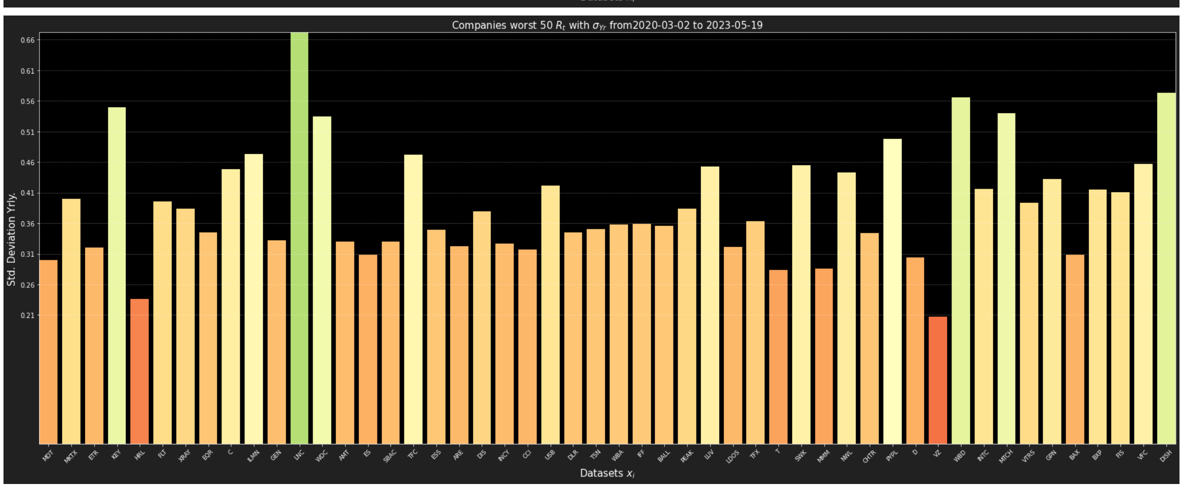
As it was stated, not all risks are bad, so in this case the biggest winners had the most uncertainty which caused by rapid fluctuations which ended in a positive way. Tesla's volatility was one of the highest as well as its sucess in the Top 10.

In [295…

```
df_col, df_index = Stats_Simple.T['Yr_Std'].head(50),
(Stats_Simple.T['Yr_Std'].head(50).index)
x_arange, y_arange = np.arange(0,
Stats_Simple.T['Yr_Std'].head(50).index.shape[0], 1),
np.arange(round(Stats_Simple.T['Yr_Std'].head(50).min(), 2),
round(Stats_Simple.T['Yr_Std'].head(50).max(), 2), .05)
title = (str(Stats_Simple.T['Yr_Std'].head(50).index.name) + " best 50
$R_t$ with $\sigma_{Yr}$ from" + str(start) + " to " + str(end))
```

```
x_label, y_label = "Datasets $x_i$", "Std. Deviation Yrly."

std_head = vs.cmap_bar(df_col, df_index, x_arange, y_arange, title,
x_label, y_label)

df_col, df_index = Stats_Simple.T['Yr_Std'].tail(50),
(Stats_Simple.T['Yr_Std'].tail(50).index)
x_arange, y_arange = np.arange(0,
Stats_Simple.T['Yr_Std'].tail(50).index.shape[0], 1),
np.arange(round(Stats_Simple.T['Yr_Std'].tail(50).min(), 2),
round(Stats_Simple.T['Yr_Std'].tail(50).max(), 2), .05)
title = (str(Stats_Simple.T['Yr_Std'].tail(50).index.name) + " worst
50 $R_t$ with $\sigma_{Yr}$ from" + str(start) + " to " + str(end))
x_label, y_label = "Datasets $x_i$", "Std. Deviation Yrly."

std_tail = vs.cmap_bar(df_col, df_index, x_arange, y_arange, title,
x_label, y_label)
```





In [296…

```
Stats_Log = data_describe(prices_start, 'Log_returns', .00169, start,
end)[0]
```

```
Stats_Log
```

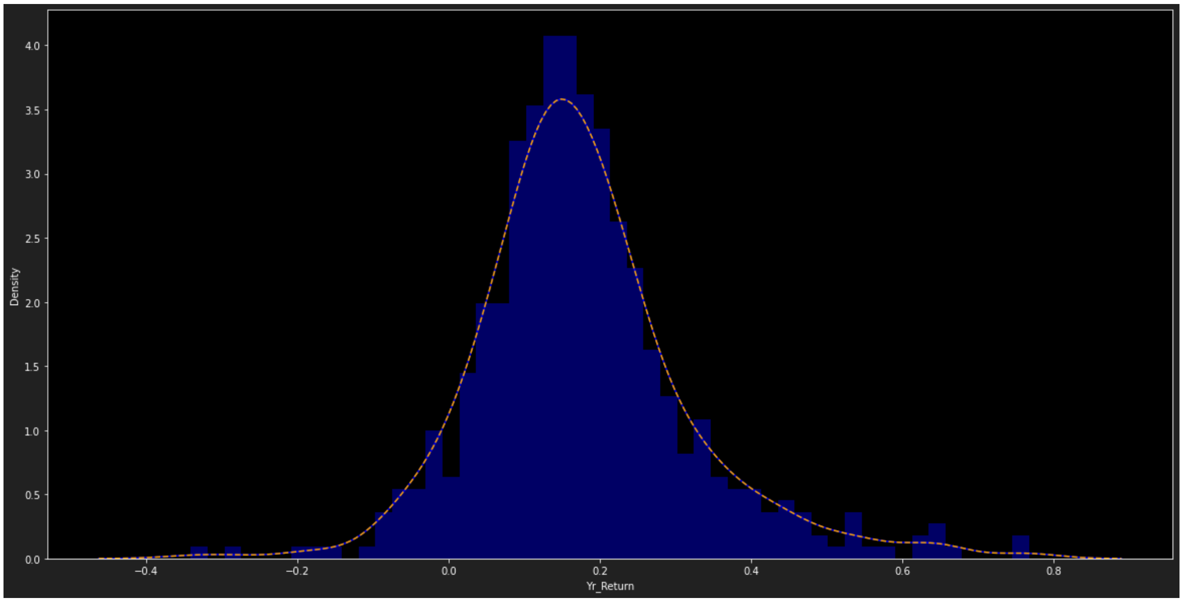
Out[296]:

Companies	MMM	AOS	ABT	ABBV	ACN	ATVI	AI
Log_returns							
min	96.542496	32.723198	59.583965	55.841316	137.243256	51.156666	27.068
25%	123.740786	52.131404	101.151182	94.312912	241.498779	75.115717	47.410
50%	141.973701	59.802975	107.947498	109.430599	275.817825	78.394012	62.561
75%	164.107021	66.729286	115.821287	144.215893	305.763092	82.283163	80.943
max	189.169373	83.477020	137.809631	167.007950	406.562866	102.699326	96.563
Logret_skew	-0.334129	0.323583	-0.098539	-0.989990	0.198536	0.936111	-0.491
Logret_kurtosis	6.685976	3.457841	5.973898	10.603032	4.360104	25.525953	4.325
Accum_Logret	-0.264046	0.772141	0.407254	0.889847	0.601308	0.319729	1.041
Yr_Return	-0.095265	0.177795	0.106157	0.197777	0.146297	0.086204	0.221
Yr_Std	0.286134	0.328314	0.292863	0.260644	0.322502	0.329354	0.310
var97.5(-)	0.033291	0.041606	0.034757	0.030181	0.041690	0.033920	0.036
var2.5(+)	-0.035820	-0.041890	-0.036062	-0.029419	-0.042779	-0.040538	-0.045
Yr_MaxDrawdown	-1.358780	-0.564289	-0.770191	-0.529143	-0.681822	-0.830817	-0.489
Sharpe	-0.338845	0.536391	0.356710	0.752318	0.448391	0.256606	0.708
Sortino	-0.433156	0.823839	0.485497	0.904933	0.628085	0.334881	0.889
Calmar	-0.070111	0.315077	0.137832	0.373768	0.214568	0.103758	0.452
Burke	-0.071355	0.312082	0.135638	0.370574	0.212089	0.101724	0.449

Simple Returns $x_i \in [x_1, x_{500}] \hookrightarrow S\&P500$ are:
Not symmetric, they are skewed, they are not stationary and they are not i.i.d.

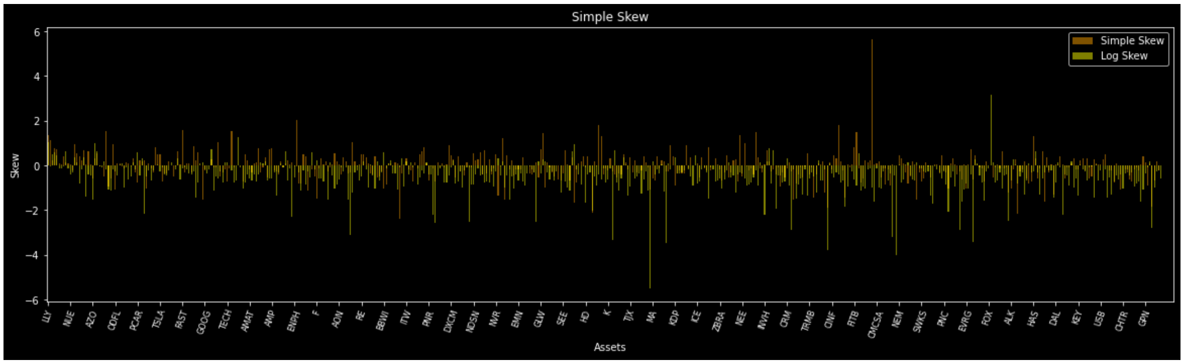
In [289…

```
fig, ax = plt.subplots(figsize=(20, 10))
sns.distplot(.T.Yr_Return, bins=50, color = 'blue', label = '$R_t$')
sns.kdeplot(Stats_Simple.T.Yr_Return, color = 'orange', linestyle="--
")
sns.distplot(Stats_Simple.T.Yr_Return, bins=50, color = 'blue', label
= '$R_t$')
sns.kdeplot(Stats_Simple.T.Yr_Return, color = 'orange', linestyle="--
")

plt.show()
```



In [55]:

```
Simple.T.Simple_skew.plot(kind="bar", figsize=(20, 5), color="orange",
alpha=.5, label="Simple Skew")
Log.T.Logret_skew.plot(kind="bar", figsize=(20, 5), color="yellow",
alpha=.5, label="Log Skew")
plt.xticks(rotation=0, fontsize=2)
#Drop x ticks and make new ticks from 0 to 500 with a step of 10
plt.xticks(np.arange(0, 500, 10))
plt.xticks(fontsize=8)
#x ticks rotation
plt.xticks(rotation=70)

plt.xlabel("Assets")
plt.ylabel("Skew")
plt.title("Simple Skew")
plt.legend()
plt.show()

#Simple.T.Simple_kurtosis.plot(kind="bar", figsize=(20, 5),
color="green", alpha=.5, label="Simple Kurtosis")
```



In [ ]:

```
def Dist_KDE(dataframe1, dataframe2, dist_label1, dist_label2,
x_ticks, y_ticks):
    """
    Function to plot yearly returns distribution and kde with Yearly
Simple & Log returns in index as dataframe with quotes in cols.
    Parameters:
    ----------
    Simple: dataframe
        Dataframe with simple returns.
    Log: dataframe
        Dataframe with log returns.
    color: str
        Color for plot ticks, labels and title text
    Returns:
    -------
    Plot with yearly returns.
    """
    fig, ax = plt.subplots(figsize=(20, 10))
    sns.distplot(dataframe1, bins=50, color = 'red', label =
dist_label1)
    sns.distplot(dataframe2, bins=50, color = 'blue', label =
dist_label2)

    sns.kdeplot(dataframe1, color = 'orange', linestyle="--")
    sns.kdeplot(dataframe2, color = 'teal', linestyle="--")

    plt.title(title,  fontsize=15)
    plt.grid(color='gray', linestyle='--')
    #plt.yticks(y_ticks)
    plt.xticks(x_ticks, rotation=45, fontsize=9)
    plt.xlabel(x_label, fontsize=15), plt.ylabel(y_label, fontsize=15)
```

```
    #plt.margins(x=0, y=0)
    #plt.tight_layout()
    ax.xaxis.label.set_color('red'), ax.yaxis.label.set_color('blue')
    ax.tick_params(axis='x', colors='white'), ax.tick_params(axis='y',
colors='white')
    plt.legend()

    return plt.show()

def Yearly_Returns(Simple, Log, color):
    """
    Function to plot yearly returns distribution and kde with Yearly
Simple & Log returns in index as dataframe with quotes in cols.
    Parameters:
    ----------
    Simple: dataframe
        Dataframe with simple returns.
    Log: dataframe
        Dataframe with log returns.
    color: str
        Color for plot ticks, labels and title text
    Returns:
    -------
    Plot with yearly returns.
    """
    fig, ax = plt.subplots(figsize=(20, 10))
    sns.distplot(Simple.T.Yr_Return, bins=50, color="red",
label="Yearly Simple $R_t$")
    sns.distplot(Log.T.Yr_Return, bins=50, color="blue", label="Yearly
Log $r_t$")

    sns.kdeplot(Simple.T.Yr_Return, color="orange", linestyle="--")
    sns.kdeplot(Log.T.Yr_Return, color="teal", linestyle="--")

    plt.title("$x_i\in [x_1,x_{500}]$ in S&P500 Yearly Returns",
size=20).set_color(color)
    plt.xticks(np.arange(round(min(Simple.T.Yr_Return), 1)*1.5,
round(max(Simple.T.Yr_Return), 1)*1.5, 0.05))
    plt.xticks(rotation=45)
```

```
    plt.xlabel("Yearly Returns")
    plt.ylabel("Frequency")
    ax.xaxis.label.set_color(color), ax.yaxis.label.set_color(color)
    ax.tick_params(axis='x', colors=color), ax.tick_params(axis='y',
colors=color)


    plt.grid(color='gray', linestyle='--')
    plt.legend()

    plt.show()



def Stationarity(x, y, n):
    """
    Function that plots a time-series and its Trend, Seasonality and
Residuals
    returning the Augmented Dickey Fuller p-value for given n periods.

        Parameters
        ----------
        x: DateTime values from economic index (col).
#data_raw['DateTime']
        y: Actual values from economic index (col). #data_raw['Actual']
        n: Periods for decomposition (int).

        Returns
        -------
        lines+marker Series, Trend, Seasonality and Residuals plots in
a didactic graph with plotly.
    """

    decomposition = seasonal_decompose(y, period = n)

    trend = decomposition.trend
    seasonal = decomposition.seasonal
    residual = decomposition.resid

    fig = make_subplots(rows = 4, cols = 1, shared_xaxes = False,
```

```
                        subplot_titles = ('Actual', 'Trend',
'Seasonal', 'Residuals'),
                        vertical_spacing = 0.15, row_width = [0.25,
0.25, 0.25, 0.25])

    fig.add_trace(go.Scatter(x=x, y=y, mode='lines+markers',
name='Actual',
         line=dict(color='black'), marker=dict(symbol=2,
color='black')))

    fig.add_trace(go.Scatter(x=x, y=trend, mode='lines+markers',
name='Trend',
         line=dict(color='black'), marker=dict(symbol=2,
color='blue')), row = 2, col = 1)

    fig.add_trace(go.Scatter(x=x, y=seasonal, mode='lines+markers',
name='Seasonal',
         line=dict(color='black'), marker=dict(symbol=2,
color='green')), row = 3, col = 1)

    fig.add_trace(go.Scatter(x=x, y=residual, mode='lines+markers',
name='Residuals',
         line=dict(color='black'), marker=dict(symbol=2,
color='gray')), row = 4, col = 1)

    fig.show(),fig.show("png")

    return "p-value:", adfuller(y)[1],



def qq(index):
    """
    Function that graphs a QQ-plot intended to model economic index
Actual values.


        Parameters
        ----------
        index: Actual values from economic index (col)

```

```
        Returns
        -------
        QQ-plot for given data.
    """
    sm.qqplot(index, line= 'q', fit  = True)
    pylab.show()
```

2. Descriptive Statistics:

Statistical descriptions are the foundation of the knowledge from data and valuable insights can be communicated.
In this case, statistical descriptions establish foundations that can analyze new data at any given time and evaluate for example
if it is more feasible due to reasons that aren't captured by data to include $\vec{w_{i \neq j}}$ or have $\vec{w}_i$ adjusted and/or discarded from $max|min_{\vec{w}_i} R_{j+}$

For future references, fitted params. estimators $f(\hat{X}_i)$ will be obtained and their relative qualities assesed.

`vs.selection_data`

$x_i \in [x_1, x_{25}] \hookrightarrow R_{Sortino_{+25}}$

In [55]:
```
pd.DataFrame(((Sortino.sort_values(by="sortino",
ascending=False).head(25).T).iloc[7:,
:]).mean(axis=1)).rename(columns={0:"Equiprob. xi mean"})
```

Out[55]:

	Equiprob. xi mean
log_returns	
Accum_Logret	2.120501
Yr_Return	0.338583
Yr_Std	0.400790
var97.5(-)	0.047787
var2.5(+)	-0.047594
sharpe	0.844401
sortino	1.183682
Yr_MaxDrawdown	-0.570987

In [10]:

```
prices, r_log, summary_log = vs.selection_data(SP_Assets_r, "Log", rf,
best, start, end)
prices, r_simple, summary_simple = vs.selection_data(SP_Assets_r,
"Simple", rf, best, start, end)
```

Log Returns r_t Data Selection from which optimizations will be performed are the following:

In [11]:

```
d.Markdown(tabulate(summary_log, headers='keys', tablefmt='pipe'))
```

Out[11]:

	μ_{iyr}	σ_{yr}	R_{Sharpe}	$R_{Sortino}$
LLY	0.389078	0.333508	1.16156	1.83883
PWR	0.465002	0.395741	1.17075	1.62689
MCK	0.309661	0.304455	1.01155	1.49205
EQT	0.56255	0.637027	0.880433	1.44133
FSLR	0.46928	0.542319	0.862205	1.33687
CDNS	0.358509	0.394581	0.904298	1.26689
NVDA	0.46995	0.554642	0.844255	1.25997
SNPS	0.329146	0.381763	0.857747	1.23887
CMG	0.313153	0.391956	0.794637	1.18406
GWW	0.281367	0.32329	0.865097	1.15575
NUE	0.389709	0.462014	0.839842	1.15465
STLD	0.418085	0.500739	0.831562	1.13154
ABC	0.228277	0.293917	0.770924	1.10952
ANET	0.333834	0.424135	0.78311	1.10943
TSCO	0.291765	0.335655	0.864204	1.07846
AAPL	0.271344	0.353806	0.762153	1.06451
IT	0.290663	0.381371	0.757722	1.05999
ORLY	0.287439	0.313096	0.912656	1.03105
GIS	0.19679	0.241113	0.809166	1.02754
ON	0.464334	0.615284	0.75192	1.0257
AZO	0.288956	0.318285	0.902545	1.0184
ORCL	0.235135	0.325226	0.717794	1.01203
UPS	0.219494	0.323335	0.673618	0.985501
FCX	0.390078	0.576151	0.674109	0.971247
LIN	0.210966	0.296352	0.706174	0.970975

vs.BoxHist

In [59]:

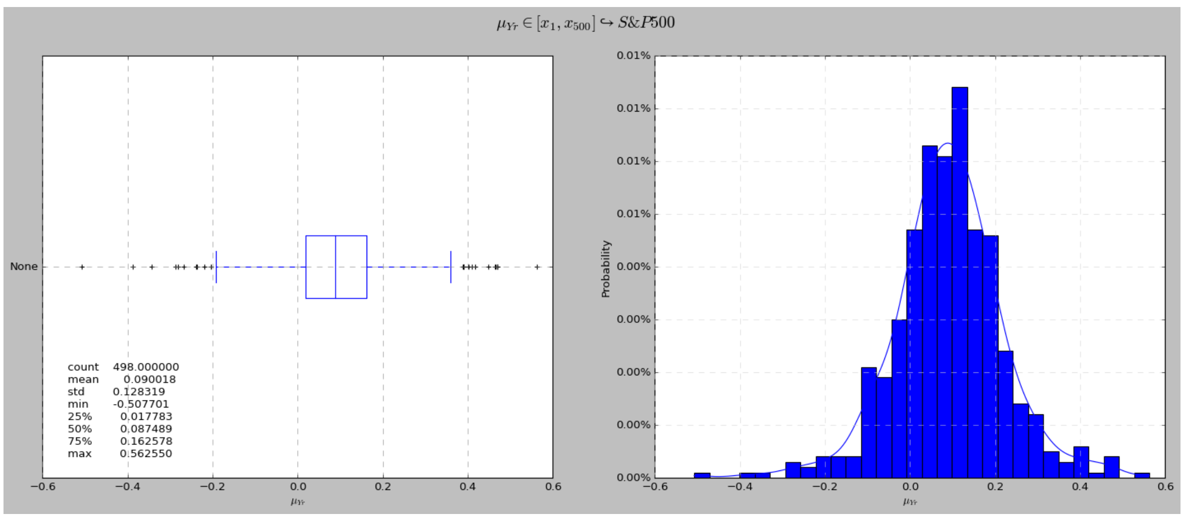
```
def BoxHist(data, output, bins, color, label, title, start, end):
    """Boxplot and Histogram for selected output method for returns
method for data, assuming equiprobable weights.
    Parameters
    ----------
    data : DataFrame
        Data to plot.
```

```
    output: str
        'prices' or 'log_returns' string to return its stats.
    bins : int
        Number of bins for histogram.
    color : str
        Color for plots.
    x1_label : str
        x1_label for boxplot.
    x2_label : str
        x2_label for histogram.
    title : str
        Title for both plots.
    start : str
        Start date for Stats calculations from dt.data_describe.
    end : str
        End date for Stats calculations from dt.data_describe.
    Returns
    -------
    Boxplot and Histogram with Stats visualization



    Returns
    -------
    Boxplot and Histogram of Returns Method with its dt.describe_stats
summary with equiprobable weights.
    """
    plt.style.use("classic")
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(22, 8))
    data.plot.box(ax=ax1, color=color, vert=False)
    Box_Stats = pd.DataFrame(((dt.data_describe(data, output, .00169,
start, end).sort_values(by="sortino",
    ascending=False).head(25).T).iloc[7:,
:]).mean(axis=1)).rename(columns={0:"Equiprob. xi mean"})

    plt.text(0.05, 0.05, data.describe().round(6).to_string(),
transform=ax1.transAxes)

    ax1.set_xlabel(label)
    sns.histplot(data, bins=bins, kde=True, alpha=0.5,
```

```
ax=ax2).legend().remove()
    for patch in ax2.patches:
        patch.set_facecolor(color)
    ax2.set_yticklabels(["{:.2f}%".format(x/10000) for x in
ax2.get_yticks()])
    ax2.set_ylabel("Probability")
    ax2.set_xlabel(label)
    fig.suptitle(str(label) + title, fontsize=18)
    ax1.grid(color="gray", linestyle="--"),
ax2.grid(color="lightgray", linestyle="--")
    #Face color for plots
    ax1.set_facecolor("lightgray"), ax2.set_facecolor("lightgray")

    plt.show()
```

In [31]:

```
BoxHist(r_log.mean()*252, 30, 'blue', "$\mu_{Yr}{{r_{t}}(x_i)$", "$\in
[x_1,x_{500}]$ $\hookrightarrow$ S&P500")
```



In [28]:

```
def BoxHistTest(data, bins, color, label, title):
    """Boxplot and Histogram for given data
    ----------
    data : DataFrame
        Data to plot.
    bins : int
        Number of bins for histogram.
    color : str
        Color for plots.
    x1_label : str
```

```
        x1_label for boxplot.
    x2_label : str
        x2_label for histogram.
    title : str
        Title for both plots.
    Returns
    -------
    Boxplot and Histogram of data
    """
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(22, 8))
    data.plot.box(ax=ax1, color=color, vert=False)
    stats =
pd.DataFrame(dt.data_describe(data).mean(axis=1).round(6)).iloc[3:].rena
{0:label}).dropna().to_string()
    plt.text(0.05, 0.05, stats, transform=ax1.transAxes)
    ax1.set_xlabel(label)
    sns.histplot(data, bins=bins, kde=True, alpha=0.5,
ax=ax2).legend().remove()
    for patch in ax2.patches:
        patch.set_facecolor(color)
    ax2.set_yticklabels(["{:.2f}%".format(x/10000) for x in
ax2.get_yticks()])
    ax2.set_ylabel("Probability")
    ax2.set_xlabel(label)
    fig.suptitle(str(label) + title, fontsize=18, fontweight="bold")
    ax1.grid(color="gray", linestyle="--"),
ax2.grid(color="lightgray", linestyle="--")

    plt.show()
```

In [21]:

```
pd.DataFrame(dt.data_describe(r_simple.sample(100000,
replace=True)).mean(axis=1)).iloc[3:].rename(columns=
{0:"μ(Rt)"}).dropna()
```

Out[21]:

	μ(Rt)
Data_Stats	
min	-0.166529
25%	-0.011588
50%	0.000660
75%	0.013012
max	0.165020
Change	-1.869931
return_y	-248.366857
var97.5	-266.737348
var2.5	235.923951
sharpe	-0.612600
skew	-0.041598
kurtosis	9.839448

In [23]:

```
Xi = pd.DataFrame(r_simple.mean(axis=1)).sample(100000, replace=True)
Xi.sort_index(inplace=True)
Xi = Xi.groupby(Xi.index).mean()
Xi.head()
```

Out[23]:

	0
formatted_date	
2020-03-03	-0.024616
2020-03-04	0.038751
2020-03-05	-0.035667
2020-03-06	-0.020621
2020-03-09	-0.088776

In [24]:

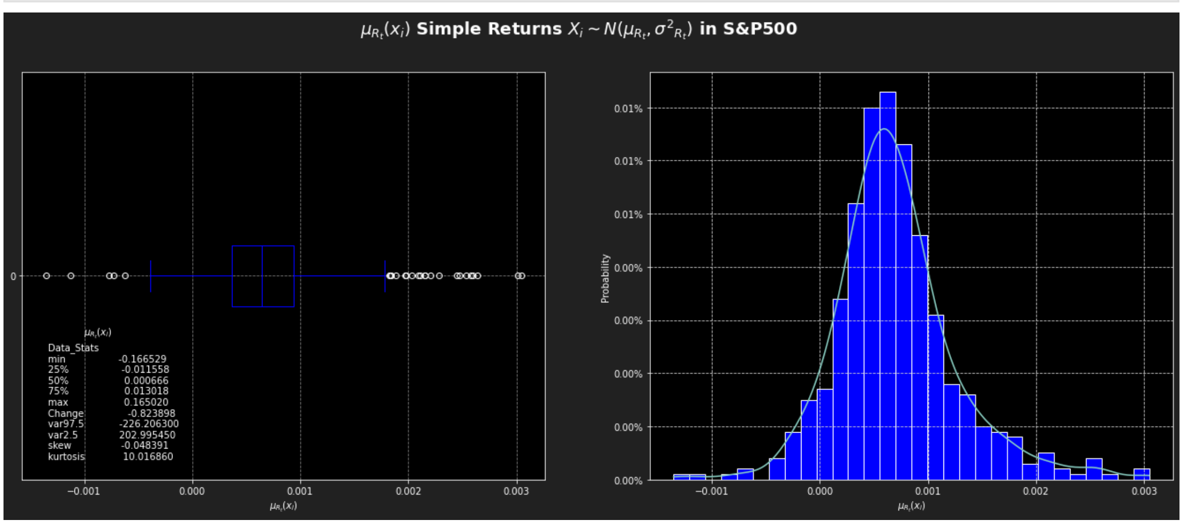
```
dt.data_describe(Xi)
```

Out[24]:

Data_Stats	0
count	811.000000
mean	0.000699
std	274.333863
min	-0.127111
25%	-0.006645
50%	0.001083
75%	0.008164
max	0.115792
Change	-0.887854
return_y	-413.901074
var97.5	-160.905304
var2.5	138.287224
sharpe	-1.508749
skew	-0.502610
kurtosis	11.946669

In [135…

```
BoxHistTest(r_simple.mean().to_frame(), 30, 'blue', "$\mu_{R_{t}}(x_i)$", " Simple Returns $X_i\sim N({\mu}_{R_t}, {\sigma^2}_{R_t})$ in S&P500")
#vs.BoxHist(r_simple.var().to_frame().sample(100000, replace=True).rename(columns={0:"σ(Rt)"}), 20, 'blue', "$\sigma^2_{R_{t}}(X_i)$", " Simple Returns Variance Simulations $X_i\in [X_1,X_{500}]$")
```



Log Returns r_{t_n} :

In [13]:

```
prices, r_log, summary_log = vs.selection_data(SP_Assets_r, "Log", rf,
best, start, execution_date)
r_log.tail()
```

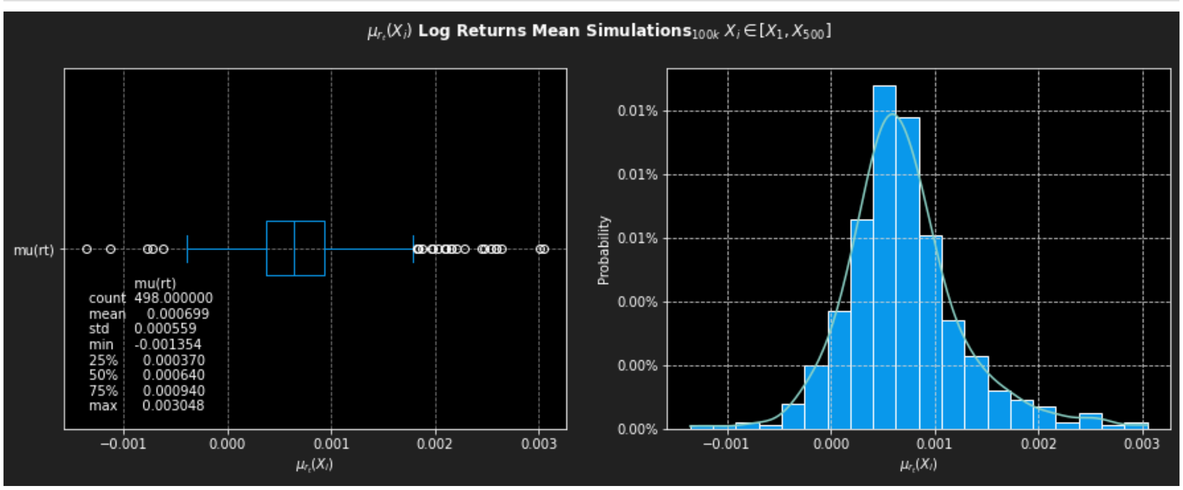
Out[13]:

formatted_date	MMM	AOS	ABT	ABBV	ACN	ATVI	ADM	AD
2023-05-10	0.000992	-0.001444	0.003983	-0.002796	0.020325	0.006468	-0.005078	0.003
2023-05-11	-0.001190	-0.011917	-0.005799	0.001160	0.012492	0.013591	0.003477	-0.007
2023-05-12	-0.005073	-0.017105	0.003990	0.003813	0.017909	0.004274	0.006388	-0.018
2023-05-15	0.002192	0.014176	-0.005900	-0.003813	0.001154	0.012332	0.008981	0.030
2023-05-16	-0.024989	-0.014622	-0.004105	-0.022769	0.006036	-0.007046	-0.033076	-0.001

$\therefore$ Random variables $X_i \sim N(\mu_{r_t}, \sigma^2_{r_t})$:

In [134…

```
vs.BoxHist(r_log.mean().to_frame().rename(columns={0:"mu(rt)"}), 20,
'#0998eb', "$\mu_{r_{t}}(X_i)$", " Log Returns Mean
Simulations$_{100k}$ $X_i\in [X_1,X_{500}]$")
#vs.BoxHist(r_log.var().to_frame().sample(100000,
replace=True).rename(columns={0:"σ(Rt)"}), 20, 'lightblue',
"$\sigma^2_{R_{t}}(X_i)$", " Log Returns Variance Simulations $X_i\in
[X_1,X_{500}]$")
```



The Simple Returns mean differ from Log Returns just enough for the model not to be modelled correctly.

Nevertheless, compounding effects among other factors make Log Returns best suitable for the model.

2.1.3 Cumulative $\bold_{R_t}(X_i)$ & Log $\bold\mu_{r_t}(X_i)$

Simple Returns R_t :

In [163…

```
r_simple_acum = ((1+r_simple).cumprod()-1)
r_simple_acum = r_simple_acum.T[(r_simple_acum.T >= -1) &
(r_simple_acum.T <= 1)].T
r_simple_acum = r_simple_acum.fillna(method='ffill')
r_simple_acum.tail()
```

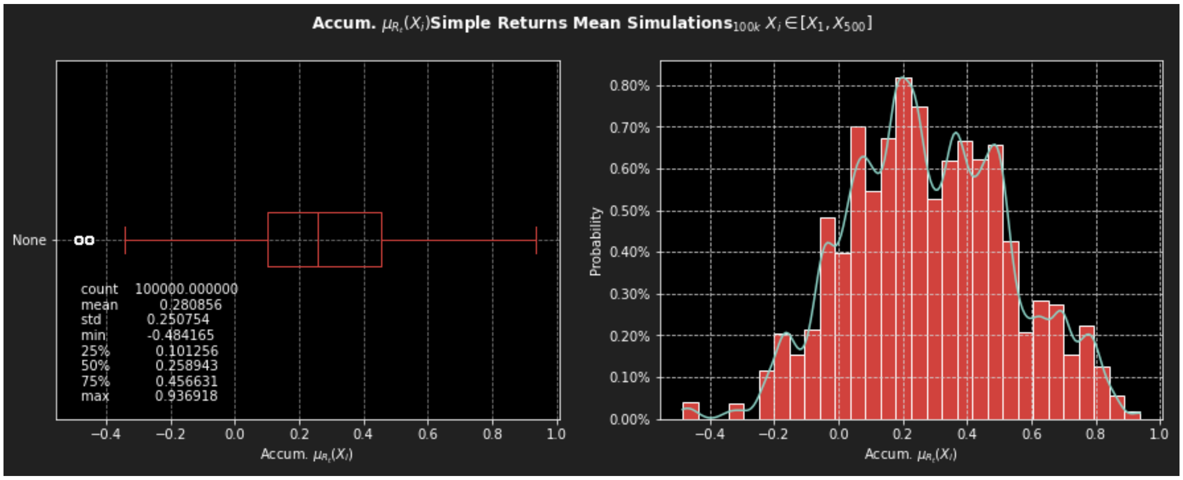
Out[163]:

	MMM	AOS	ABT	ABBV	ACN	ATVI	ADM	ADBE
formatted_date								
2023-05-10	-0.261376	0.792861	0.429992	0.906908	0.485205	0.276236	0.981963	-0.045132
2023-05-11	-0.262254	0.771623	0.421724	0.909122	0.503874	0.293701	0.981963	-0.051904
2023-05-12	-0.265988	0.741578	0.427408	0.916415	0.531050	0.299242	0.981963	-0.068919
2023-05-15	-0.264377	0.766442	0.419011	0.909122	0.532817	0.315363	0.981963	-0.040552
2023-05-16	-0.282532	0.740801	0.413197	0.866144	0.542097	0.306127	0.981963	-0.042106

$\therefore$ Random variables $X_i \sim N(\mu_{R_t}, \sigma^2_{R_t})$:

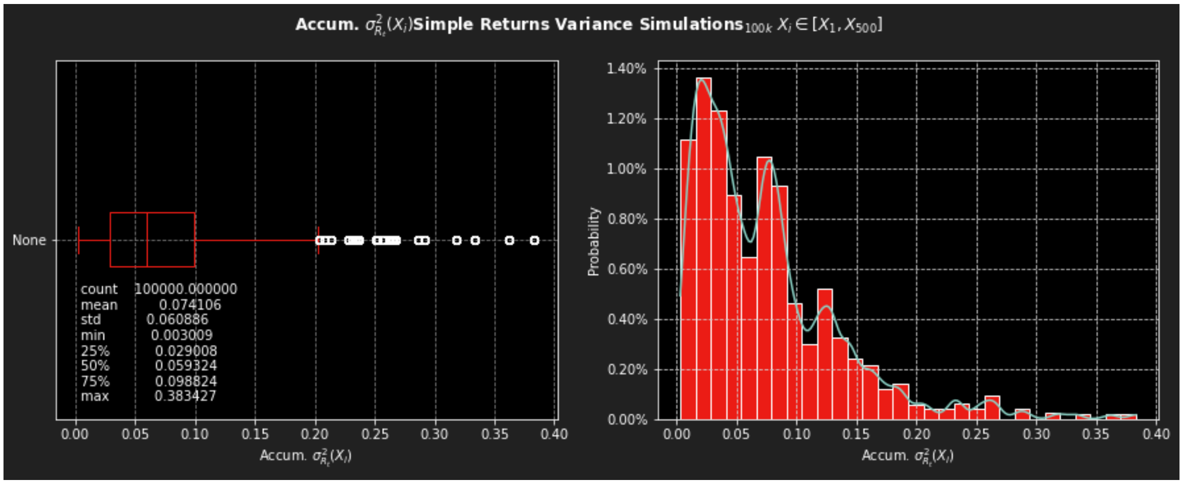
In [165…

```
vs.BoxHist(r_simple_acum.mean().sample(100000,
replace=True).rename(index={0:"Accum_mu(Rt)"}), 30, '#d1423d', "Accum.
$\mu_{R_{t}}(X_i)$", "Simple Returns Mean Simulations$_{100k}$ $X_i\in
[X_1,X_{500}]$")
```



In [190…

```
vs.BoxHist(r_simple_acum.var().sample(100000,
replace=True).rename(index={0:"Accum_s(Rt)"}), 30, '#eb1c15', "Accum.
$\sigma^2_{R_{t}}(X_i)$", "Simple Returns Variance
Simulations$_{100k}$ $X_i\in [X_1,X_{500}]$")
```



Log Returns r_{t_n} :

In [181…

```
r_log_acum = ((1+r_log).cumprod()-1)
r_log_acum.tail()
```

Out[181]:

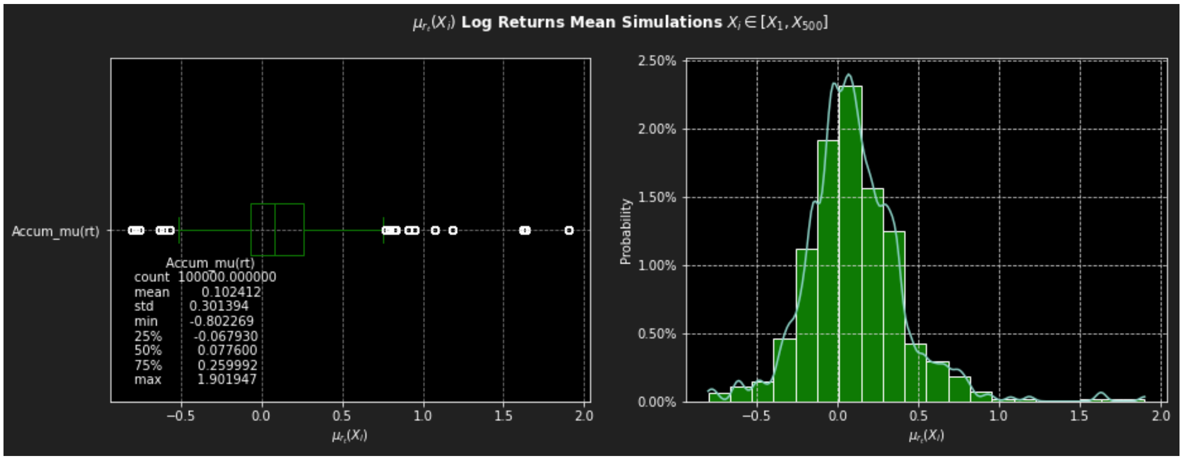
	MMM	AOS	ABT	ABBV	ACN	ATVI	ADM	ADBE
formatted_date								
2023-05-10	-0.352609	0.509598	0.245543	0.707755	0.257566	0.073737	0.770212	-0.279678
2023-05-11	-0.353380	0.491608	0.238320	0.709737	0.273275	0.088331	0.776367	-0.284806
2023-05-12	-0.356660	0.466095	0.243262	0.716256	0.296078	0.092983	0.787715	-0.297757
2023-05-15	-0.355250	0.486878	0.235926	0.709712	0.297574	0.106461	0.803769	-0.276682
2023-05-16	-0.371362	0.465137	0.230852	0.670783	0.305405	0.098664	0.744107	-0.277854

$\therefore$ Random variables $X_i \sim N(\mu_{R_t}, \sigma^2_{R_t})$:

In [182…

```
vs.BoxHist(r_log_acum.mean().to_frame().sample(100000,
replace=True).rename(columns={0:"Accum_mu(rt)"}), 20, '#0e7a04',
"$\mu_{r_{t}}(X_i)$", " Log Returns Mean Simulations $X_i\in
[X_1,X_{500}]$")
#vs.BoxHist(r_log.var().to_frame().sample(100000,
replace=True).rename(columns={0:"σ(Rt)"}), 20, 'green',
```

```
"$\sigma^2_{R_{t}}(X_i)$", " Log Returns Variance Simulations $X_i\in
[X_1,X_{500}]$")
```



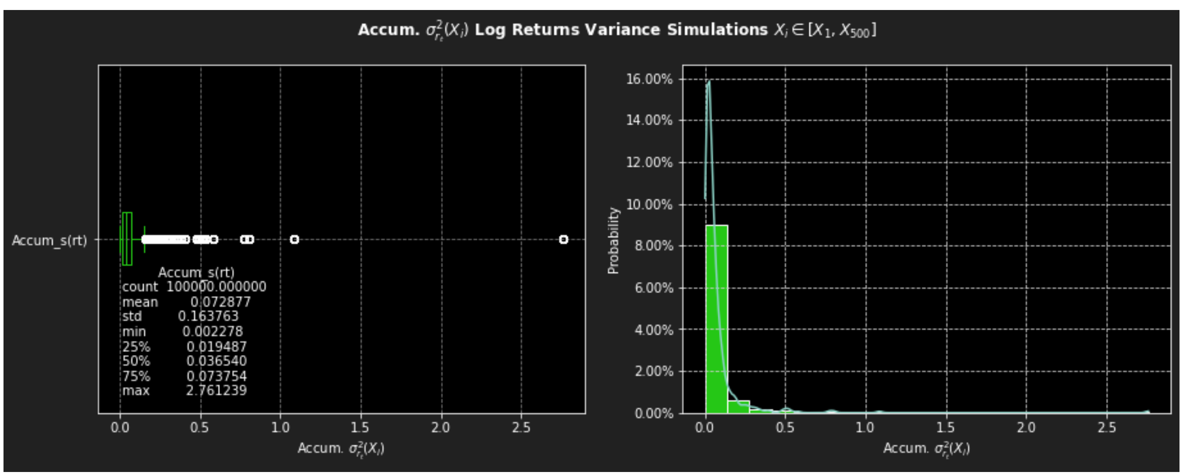
In [187…

```
print("Outliers:", len(r_log_acum.mean(axis=1)
[abs(r_log_acum.mean(axis=1) - np.mean(r_log_acum.mean(axis=1))) < 2 *
np.std(r_log_acum.mean(axis=1))].to_frame())/100000)
```

```
Outliers: 0.00769
```

In [191…

```
vs.BoxHist(r_log_acum.var().to_frame().sample(100000,
replace=True).rename(columns={0:"Accum_s(rt)"}), 20, '#25c716',
"Accum. $\sigma^2_{r_{t}}(X_i)$", " Log Returns Variance Simulations
$X_i\in [X_1,X_{500}]$")
```



Sharpe's Ratio measures the units of risk (σ) per unit of excess returns over a risk-free rate (rf) :

- $R_{Sharpe} = \frac{\mu_i - rf}{\sigma_i(r_t)}$.

Sortino's Ratio measures the units of negative risks $[\sigma_i(r_{t\leq 0})]$ per unit of excess returns over a risk-free rate (rf) :

- $R_{Sortino} = \frac{\mu_i - rf}{\sigma_i(r_{t\leq 0})}$

To avoid risks associated to negative returns, Data Selection $\forall X_i \in [X_1, X_{500}] \rightarrow X_{P_{Rmax_j}}$ is based on $S\&P500$ *Sortino's Ratio Top 25*:

In [ ]:
```
fn.retSLog_Selection(SP_Assets_r, rf, best, start, execution_date)
```

In [ ]:
```
vs.Selection_R_SLog_Plot(SP_Assets_r, rf, best, start,
execution_date, r_jump)
```

2.2 Modelling X_i

In [ ]:
```
def Stats(dataframe, Selection, r, P, percentiles, dist, title,
color):
    """
    Stats is a function that resamples data from a Selection
performed over a dataframe.
    Parameters:
    ----------
    dataframe : dataframe
        Dataframe from which the Selection is made, in order to acess
Selection's original data.
    Selection : list
        Selection to Resample for given period(s) etc. basis whose
period is longer than original data.
    r : str
        Type of return for the model: "Simple" (multiplicative) or
"Log" (additive).
    P : str
        Period of Resample (e.g. "W" for Weekly, "M" for Monthly,
"3T" for Trimestral, "Q" for Quarterly,
        "Y" for Yearly, etc. for Dataframe.resample (see refs.).
    percentiles : list
        List of Returns of Percentiles returned by vs.Stats[0]
dataframe (e.g. [.05, .25, .5, .75, .95]).
    dist : list
        Continuous Distributions to fit on datasets Xi
    title : str
```

```
        Title of the Box-plot
    color : str
        Color of the Box-plot.
    Returns:
    -------
    describe : dataframe
        Stats returns summary statistics (mean, std, min, max, 
percentiles, skewness and kurtosis) in a
        markdown object callable as a dataframe by assigning a 
variable to the function in pos. [2].
    """

    if  r == "Simple" :
        Selection = 
(dataframe[Selection.index].pct_change()).iloc[1:, :].dropna(axis = 
1)
    if  r == "Log" :
        Selection = 
np.log(dataframe[Selection.index]).diff().iloc[1:, :].dropna(axis = 
1)
    if r != "Simple" and r != "Log" :
        print("Aborted: Please select a valid Return type: 'Simple' 
or 'Log'. Stats help command: help(vs.Stats)")

    Selection.index = pd.to_datetime(Selection.index)
    Selection_Mo_r = Selection.resample(P).agg(lambda x: x[-1])
    Selection_Mo_r.plot(kind = "box", figsize = (22, 13), title = 
title, color = color, fontsize = 13)

    for i in range(0, len(Selection_Mo_r.columns)):
        plt.text(x = i + 0.96 , y = Selection_Mo_r.iloc[:, i].mean() 
+ .0075, s = str("$\mu$ = +") + str(round(Selection_Mo_r.iloc[:, 
i].mean(), 4)), fontsize = 6.5, fontweight = "bold", color = 
"lightgreen")
        plt.text(x = i + 0.98 , y = Selection_Mo_r.iloc[:, i].max() + 
.010, s = str("+") + str(round(Selection_Mo_r.iloc[:, i].max(), 3)), 
fontsize = 8.5, color = "green")
        plt.text(x = i + 0.98 , y = Selection_Mo_r.iloc[:, i].min() - 
.015, s = str(round(Selection_Mo_r.iloc[:, i].min(), 3)), fontsize = 
```

```
8.5, color = "red")

    describe = Selection_Mo_r.describe(percentiles)
    describe["mode"] = Selection_Mo_r.mode().iloc[0, :]
    describe["skewness"] = st.skew(Selection_Mo_r)
    describe["kurtosis"] = st.kurtosis(Selection_Mo_r)
    describe.replace("\n", "")

    dist_fit = np.empty(len(Selection_Mo_r.columns), dtype=object)

    for i in range(0, len(Selection.columns)):
        f = Fitter(pd.DataFrame(Selection_Mo_r.iloc[:, i]),
distributions = dist, timeout=5)
        f.fit()
        params, AIC, BIC = [StringIO() for i in range(3)]
        (print(f.get_best(), file=params)),
(print(f.get_best(method="aic"), file=AIC)),
(print(f.get_best(method="bic"), file=BIC))
        params, AIC, BIC = [i.getvalue() for i in [params, AIC, BIC]]
        dist_fit[i] = (params + AIC + BIC).replace("\n", ", ")

    plt.title(title, fontsize = 20)
    plt.axhline(0, color = "red", lw = .5, linestyle = "--")
    plt.axhspan(0, Selection_Mo_r.min().min(), facecolor = "red",
alpha = 0.2)
    plt.axhspan(0, Selection_Mo_r.max().max(), facecolor = "green",
alpha = 0.2)

    plt.xticks(rotation = 45)
    for i, t in enumerate(plt.gca().xaxis.get_ticklabels()):
        if (i % 2) != 0:
            t.set_color("lightgreen")
        else:
            t.set_color("white")

    plt.yticks(np.arange(round(Selection_Mo_r.min().min(), 1),
round(Selection_Mo_r.max().max(), 1), 0.05))
    plt.grid(alpha = 0.5, linestyle = "--", color = "grey")
```

```
    IPython.core.display.clear_output()
    return describe, dist_fit, plt.show()
```

In [ ]:
```
Sortino25[2]
```

In [ ]:
```
Selection.tail()
```

In [ ]:
```
(SP_Assets_r.loc[start:today]
[Sortino25[2].index]).pct_change().iloc[1:, :].dropna(axis =
1).tail()
```

In [ ]:
```
np.log(SP_Assets_r.loc[start:today]
[Sortino25[2].index]).diff().iloc[1:, :].dropna(axis = 1).tail()
```

In [ ]:
```
SP_Assets_r.loc[start:today], Sortino25[2]
```

In [ ]:
```
dist=([d for d in dir(st) if isinstance(getattr(st, d), getattr(st,
"rv_continuous"))])[0:60]

def ret(dataframe, selection, r):
    if  r == "Simple" :
        returns = (dataframe[selection.index]).pct_change().iloc[1:,
:].dropna(axis = 1)
    if  r == "Log" :
        returns = np.log(dataframe[selection.index]).diff().iloc[1:,
:].dropna(axis = 1)
    if r != "Simple" and r != "Log" :
        print("Aborted: Please select a valid Return type: 'Simple'
or 'Log'. selection_data help command: help(vs.selection_data)")

    returns.index = pd.to_datetime(returns.index)
    returns_Mo_r = returns.resample("M").agg(lambda x: x[-1])
    returns_Mo_r.plot(kind = "box", figsize = (22, 13), title =
"test", color = "yellow", fontsize = 13)

    return returns, returns_Mo_r.max()

ret(SP_Assets_r.loc[start:today], Sortino25[2], "Simple")[1]
```

```
#Selection.index = pd.to_datetime(Sortino25[2].index)
# Selection_Mo_r = Selection.resample(P).agg(lambda x: x[-1])
# Selection_Mo_r.plot(kind = "box", figsize = (22, 13), title =
title, color = color, fontsize = 13)

# for i in range(0, len(Selection_Mo_r.columns)):
#     plt.text(x = i + 0.96 , y = Selection_Mo_r.iloc[:, i].mean() +
.0075, s = str("$\mu$ = +") + str(round(Selection_Mo_r.iloc[:,
i].mean(), 4)), fontsize = 6.5, fontweight = "bold", color =
"lightgreen")
#     plt.text(x = i + 0.98 , y = Selection_Mo_r.iloc[:, i].max() +
.010, s = str("+") + str(round(Selection_Mo_r.iloc[:, i].max(), 3)),
fontsize = 8.5, color = "green")
#     plt.text(x = i + 0.98 , y = Selection_Mo_r.iloc[:, i].min() -
.015, s = str(round(Selection_Mo_r.iloc[:, i].min(), 3)), fontsize =
8.5, color = "red")

# describe = Selection_Mo_r.describe(percentiles)
# describe["mode"] = Selection_Mo_r.mode().iloc[0, :]
# describe["skewness"] = st.skew(Selection_Mo_r)
# describe["kurtosis"] = st.kurtosis(Selection_Mo_r)
# describe.replace("\n", "")

# dist_fit = np.empty(len(Selection_Mo_r.columns), dtype=object)

# for i in range(0, len(Selection.columns)):
#     f = Fitter(pd.DataFrame(Selection_Mo_r.iloc[:, i]),
distributions = dist, timeout=5)
#     f.fit()
#     params, AIC, BIC = [StringIO() for i in range(3)]
#     (print(f.get_best(), file=params)),
(print(f.get_best(method="aic"), file=AIC)),
(print(f.get_best(method="bic"), file=BIC))
#     params, AIC, BIC = [i.getvalue() for i in [params, AIC, BIC]]
#     dist_fit[i] = (params + AIC + BIC).replace("\n", ", ")

# plt.title(title, fontsize = 20)
# plt.axhline(0, color = "red", lw = .5, linestyle = "--")
```

```
# plt.axhspan(0, Selection_Mo_r.min().min(), facecolor = "red", alpha
= 0.2)
# plt.axhspan(0, Selection_Mo_r.max().max(), facecolor = "green",
alpha = 0.2)

# plt.xticks(rotation = 45)
# for i, t in enumerate(plt.gca().xaxis.get_ticklabels()):
#     if (i % 2) != 0:
#         t.set_color("lightgreen")
#     else:
#         t.set_color("white")

# plt.yticks(np.arange(round(Selection_Mo_r.min().min(), 1),
round(Selection_Mo_r.max().max(), 1), 0.05))
# plt.grid(alpha = 0.5, linestyle = "--", color = "grey")
# plt.show()
```

$r_{Log}(X_i)$:

In [ ]:
```
Selection = np.log(dataframe[Selection.index]).diff().iloc[1:,
:].dropna(axis = 1)
```

In [ ]:
```
#Stats(dataframe, Selection, r, P, percentiles, dist, title, color):
describe_Wk = Stats(SP_Assets_r.loc[start:today], Sortino25[2],
"Log", "W", [.025, .25, .5, .75, .95], dist,
                    "$S&P$ 500 $r_{Log}(X_i)$ Selection Weekly
Resampling from" + str(start) + "to" + str(today), "lightyellow")
```

In [ ]:
```
describe_Wk[0]
```

In [ ]:
```
describe_Mo = vs.Stats(SP_Assets_r.loc["2020-03-02":today],
Sortino25[2], P[1][0],
                  "$X_i$ Selection Resamplings from $S&P$ 500 on a "
+ str(P[1][1]) + " basis from ", "2020-03-02", today,
                  [.025, .25, .5, .75, .95], dist, color=color[1])

```

In [ ]:
```
describe_Mo[0]
```

In [ ]:
```
describe_Qt = vs.Stats(SP_Assets_r.loc["2020-03-02":today],
Sortino25[2], P[2][0],
                  "$X_i$ Selection Resamplings from $S&P$ 500 on a "
+ str(P[2][1]) + " basis from ", "2020-03-02", today,
                  [.025, .25, .5, .75, .95], dist, color=color[2])
```

In [ ]:
```
describe_Qt[0]
```

Estimators Parameters:

$f(X_i)$ and AIC & BIC:

Distributions and parameters that best estimate $f(X_i)$ are obtained from 104 distribution classes and instances for continuous random variables in `Fitter` module *(see refs.)*.

The AIC *Akaike* & BIC *Bayesian Information Criterion* models are estimators of *relative quality* of predictions in the *Log-Likelihood* for fitted distributions.
Minimum relative values for AIC and BIC are usually preferred and in this case, they are obtained to model X_i resampled data on W, M & Q periods P.
Criterion's goodness of fit is inversely related so they tend to be used together to avoid under/over fitting and they are defined as follows:

- $AIC = 2k - 2ln(\hat{L})$
- $BIC = kln(n) - 2ln(\hat{L})$

where:

k = Params. in model.
n = No° of observations.
$\hat{L}$ = $Likelihood_{f_{max.}}$.

In [ ]:
```
dist_fit=pd.DataFrame([describe_Wk[1], describe_Mo[1],
describe_Qt[1]]).T
dist_fit_format = fn.format_table(dist_fit, Sortino25[2])
dist_fit_format
```

3. Descriptive and Prescriptive Analytics for X_P

3.1 X_P Optimizations Models

Equal weighted datasets are omitted from the analysis for simplicity purposes.

If we have n *unequally* weighted datasets $X_i = 1, 2, .., n$, to model X_P we need μ_P & σ_P.

And their weighted average is concluded:

$$\mu_P = \frac{\sum_{i=1}^{n} w_i \mu_{X_i}}{\sum_{i=1}^{n} w_i}$$

If

$$\sum_{i=1}^{n} w_i = 1$$

then:

$$\mu_P = \sum_{i=1}^{n} w_i \mu_{X_i}$$

For the variance σ_P^2 we need to express $X_{i,j}$ as a matrix from the selection in $S\&P500$ *(A-Z)* quotes where $\sigma_i \sigma_j$ is the product of $X_{i,j}$ units of risk:

$$\sigma_{i,j} = \begin{bmatrix} \sigma_1 & \sigma_{1,2} & \cdots & \sigma_{1,500} \\ \sigma_{2,1} & \sigma_2 & \cdots & \sigma_{2,500} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{500,1} & \cdots & \cdots & \sigma_{500} \end{bmatrix}$$

We also need $X_{i,j}$ correlation coefficients ρ_{ij} = $\frac{Cov(X_i, X_j)}{\sigma_i \sigma_j}$ or units of risk in $X_{i,j}$ that are not shared in their fluctuations directional relationship.

Expressed and substituted as:

$$\sigma_P^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j \sigma_i \sigma_j \rho_{ij}$$

$$\sigma_P^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j Cov(X_i, X_j)$$

A product of matrices $\times$ vectors:

$$\sigma_P^2 = \vec{w}^T \times Cov_{i,j} \times \vec{w}$$

Reduced and expressed as the following in its expanded form:

$$\sigma_P^2 = \begin{bmatrix} w_1 & w_2 & \cdots & w_n \end{bmatrix} \cdot \begin{bmatrix} 1 & \rho_{1,2} & \cdots & \rho_{1,n} \\ \rho_{2,1} & 1 & \cdots & \rho_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n,1} & \cdots & \cdots & 1 \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

Now, the slope can be obtained from X_P and $X_{S\&P500}$ which is expressed as:

$\beta = \frac{Cov(r_P, r_{S\&P500})}{Var(r_{S\&P500})}$

To compute some metrics that include units of sensitivities the following are considered:

- $R_{Treynor} = \frac{Var(r_{S\&P500})(\mu_P - rf)}{Cov(r_P, r_{S\&P500})}$

or the *slope* per unit of P excess returns over the risk-free.

- $R_{Jensen}(r_P, r_{t_{S\&P500}}) = (\mu_P - rf) - \frac{Cov(r_P, r_{t_{S\&P500}})}{Var(r_{t_{S\&P500}})}(\mu_{t_{S\&P500}} - rf)$

or excess returns of P over the risk free minus the *slope* times P excess returns of a benchmark over the risk-free.

Now, the slope can be obtained from X_P and $X_{S\&P500}$ which is expressed as:

$\beta = \frac{Cov(r_P, r_{S\&P500})}{Var(r_{S\&P500})}$

To compute some metrics that include units of sensitivities the following are considered:

- $R_{Treynor} = \frac{Var(r_{S\&P500})(\mu_P - rf)}{Cov(r_P, r_{S\&P500})}$

or the *slope* per unit of P excess returns over the risk-free.

- $R_{Jensen}(r_P, r_{t_{S\&P500}}) = (\mu_P - rf) - \frac{Cov(r_P, r_{t_{S\&P500}})}{Var(r_{t_{S\&P500}})}(\mu_{t_{S\&P500}} - rf)$

or excess returns of P over the risk free minus the *slope* times P excess returns of a benchmark over the risk-free.

Optimizations $\forall w_i$ are made with `Scipy` and validated with `Numpy` from parameters $X_i \rightarrow X_P$ for:

- $R_{Treynor_{Argmax}}$
- $R_{Sharpe_{Argmax}}$
- $R_{Sortino_{Argmax}}$

- $\sigma^2_{P_{Arg_{min}}}$

In [ ]:

```
def Optimizer(Assets, index, rf, title):
    Asset_ret = (Assets.pct_change()).iloc[1:, :].dropna(axis = 1)
    index_ret = index.pct_change().iloc[1:, :].dropna(axis = 1)
    index_ret = index_ret[index_ret.index.isin(Asset_ret.index)]

    mean_ret = Asset_ret.mean() * 252
    cov = Asset_ret.cov() * 252

    N = len(mean_ret)
    w0 = np.ones(N) / N
    bnds = ((0, None), ) * N
    cons = {"type" : "eq", "fun" : lambda weights : weights.sum() - 
1}

    def Max_Sharpe(weights, Asset_ret, rf, cov):
        rp = np.dot(weights.T, Asset_ret)
        sp = np.sqrt(np.dot(weights.T, np.dot(cov, weights)))
        RS = (rp - rf) / sp
        return -(np.divide(np.subtract(rp, rf), sp))

    def Min_Var(weights, cov):
        return np.dot(weights.T, np.dot(cov, weights))

    def Min_Traynor(weights, Asset_ret, rf, cov):
        #(rp - rf) / Beta
        rp = np.dot(weights.T, Asset_ret)
        varp = np.dot(weights.T, np.dot(cov, weights))
        cov
        RT = (rp - rf) / sp
        return -(np.divide(np.subtract(rp, rf), sp))

    #---------------------------------------------------------------
---------------------------------------------------------------------
------------------------------------

    opt_EMV = optimize.minimize(Max_Sharpe, w0, (mean_ret, rf, cov),
'SLSQP', bounds = bnds,
```

```
                                constraints = cons, options={"tol":
1e-10})

    W_EMV = pd.DataFrame(np.round(opt_EMV.x.reshape(1, N), 4),
columns = Asset_ret.columns, index = ["Weights"])
    W_EMV[W_EMV <= 0.0] = np.nan
    W_EMV.dropna(axis = 1, inplace = True)

    RAssets =
Asset_ret[Asset_ret.columns[Asset_ret.columns.isin(W_EMV.columns)]]
    # MuAssets = mean_ret[mean_ret.index.isin(W_EMV.columns)]
    R_EMV = pd.DataFrame((RAssets*W_EMV.values).sum(axis = 1),
columns = ["$r_{Sharpe_{Arg_{max}}}$"])
    index_ret.rename(columns={index_ret.columns[0]: "$r_{mkt}$" },
inplace=True)
    R_EMV.insert(1, index_ret.columns[0], index_ret.values)

    Muopt_EMV = np.dot(opt_EMV.x.T, mean_ret)
    Sopt_EMV = np.sqrt(np.dot(opt_EMV.x.T, np.dot(cov, opt_EMV.x)))
    Beta_EMV = np.divide((np.cov(R_EMV.iloc[0], R_EMV.iloc[1])[0]
[1]), R_EMV.iloc[1].var())
    SR_EMV = (Muopt_EMV - rf) / Sopt_EMV

    #-------------------------------------------------------------------
--------------------------------------------------------------------
------------------------------------

    opt_MinVar = optimize.minimize(Min_Var, np.ones(N) / N, (cov,),
'SLSQP', bounds = bnds,
                                   constraints = cons, options=
{"tol": 1e-10})

    W_MinVar = pd.DataFrame(np.round(opt_MinVar.x.reshape(1, N), 4),
columns = Asset_ret.columns, index = ["Weights"])
    W_MinVar[W_MinVar <= 0.0] = np.nan
    W_MinVar.dropna(axis = 1, inplace = True)

    RAssets_MinVar =
Asset_ret[Asset_ret.columns[Asset_ret.columns.isin(W_MinVar.columns)]]
```

```
    R_MinVar =
pd.DataFrame((RAssets_MinVar*W_MinVar.values).sum(axis = 1), columns
= ["$r_{Var_{Arg_{min}}}$"])
    R_EMV.insert(2, R_MinVar.columns[0], R_MinVar.values)

    Muopt_MinVar = np.dot(opt_MinVar.x.T, mean_ret)
    Sopt_MinVar = np.sqrt(np.dot(opt_MinVar.x.T, np.dot(cov,
opt_MinVar.x)))
    Beta_MinVar = np.divide((np.cov(R_EMV.iloc[2], R_EMV.iloc[1])[0]
[1]), R_EMV.iloc[1].var())
    SR_MinVar = (Muopt_MinVar - rf) / Sopt_MinVar

    #-----------------------------------------------------------------
--------------------------------------------------------------------
-----------------------------------

    #opt_Traynor =

    #-----------------------------------------------------------------
--------------------------------------------------------------------
-----------------------------------

    Mu, Sigma, Beta, SR = [Muopt_EMV, Muopt_MinVar], [Sopt_EMV,
Sopt_MinVar], [Beta_EMV, Beta_MinVar], [SR_EMV, SR_MinVar]
    index = ["$r_{P{Sharpe_{Arg_{max}}}}$", "$r_{Var_{Arg_{min}}}$"]
    Popt = [pd.DataFrame({"$\mu_P$" : Mu[i], "$\sigma_P$" :
Sigma[i], "$\Beta_{P}$": Beta[i], "$r_{Sharpe_{Arg_{max}}}$" :
SR[i]},
                          index = [index[i]]) for i in range(0,
len(Mu))]

    Popt[0].index.name = title
    Popt[1].index.name = title
    R_EMV = R_EMV[[R_EMV.columns[1], R_EMV.columns[2],
R_EMV.columns[0]]]
    #Get the cumulative returns with cumsum for rmkt, rEMV and
rMinVar
    accum = R_EMV.cumsum()

    Argmax = [d.Markdown(tabulate(Popt[i], headers = "keys",
```

```
tablefmt = "pipe")) for i in range(0, len(Popt))]
    R_EMV = d.Markdown(tabulate(R_EMV, headers = "keys", tablefmt =
"pipe"))

    return Argmax, R_EMV, accum
```

In [ ]:
```
bench_md = "$S\&P500_{{20_{03}-23_{05}}}$"
Argmax, R_EMV, accum = vs.Optimizer(SP_Assets_r.loc["2020-03-
02":today], SP_r.loc["2020-03-02":today], 0.0169, bench_md)

Port = display(Argmax[0], Argmax[1])
```

In [ ]:
```
d.Markdown(tabulate(accum.dropna()[0:10], headers = "keys", tablefmt
= "pipe"))
#Non sliced: d.Markdown(tabulate(accum.diff().dropna()[], headers =
"keys", tablefmt = "pipe"))
```

In [ ]:
```
d.display(d.Markdown(tabulate(accum[0:10], headers = "keys",
tablefmt = "pipe")))
```

In [ ]:
```
d.display(d.Markdown(tabulate(accum[0:10], headers = "keys",
tablefmt = "pipe")))
```

In [ ]:
```
vs.Accum_ts(accum)
```

Metrics:

Confusion Matrix:

$$\begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix}$$

Metrics:

- Accuracy: $\frac{TP+TN}{TP+TN+FP+FN}$ or the ability of the classifier to find + and - samples.
- Precision: $\frac{TP}{TP+FP}$ or the ability of the classifier not to label + samples as -.
- Recall: $\frac{TP}{TP+FN}$ or the ability of the classifier to find all + samples.
- F1 Score: $2 * \frac{Precision*Recall}{Precision+Recall}$ or Precision and Recall equilibrated score through the harmonic mean.

- ROC AUC: $\frac{TPR}{FPR}$ or the ability of the classifier to find + samples and not - samples. Where a bigger number denotes a better model.

– Past performance is not a guarantee of future results, the stock market tends to be irrational.

Note:
Do not consider the results and/or its proceedures as an investment advice or recommendation.