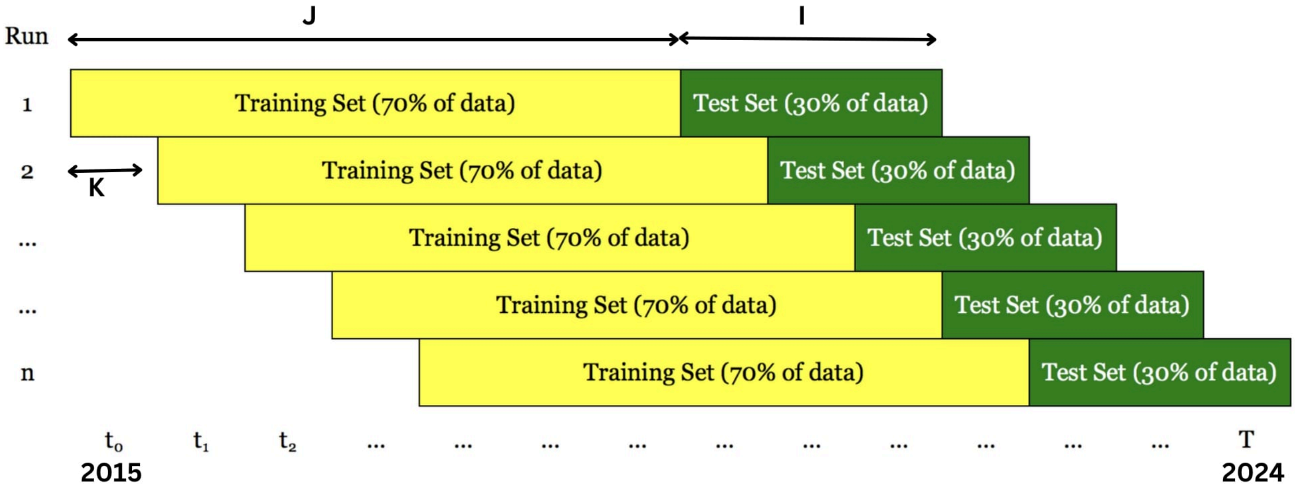
```
from GeneralFunction import download_data, calculate_metrics,split_data,weight_dic,transaction_costs,objective, optimize_risk
import pandas as pd
import numpy as np
import plotly.express as px
import warnings
warnings.filterwarnings("ignore")
```



Backtester function

Our aim is to use backtesting to demonstrate the stability of our portfolio. This is done by comparing it to other portfolios. By stability we mean that the distribution over a period of the various indicators: perf, risk and efficiency are concentrated, which in a sense is an assurance of returns, unlike a strategy that would be very volatile. However, it is essential that the distribution of our portfolio's var is better than that of lazy portfolios.

Therefore we apply a rolling window walk forward Backtest i,j,k are the test length, train length and step length respectively. The values i and k have been chosen to be equal to avoid overlap during rolling and the value 63 to match the recalibration periods of the portfolio weights to the quarterly results of the companies.

the value chosen for j is a period of 6 months in business day.

For each run we calibrate the weights on the training sample then we calculate the metrics on the test sample. So we end up with a metric for each run what gives us a set of metrics. The BT will be performed on two different periods : Covid period and period excluding COVID.

Equal risk portfolio backtester

```
def rolling_rp_optimization(i, j, k, start_date, end_date, tickers):
    try:
        # Vérification des paramètres d'entrée
        if j < i:
            raise ValueError("Le deuxième paramètre (j) doit être supérieur au premier (i).")
    except ValueError as e:
        print(f"Erreur d'entrée : {e}")
        return

    # Étape 1 : Récupérer les données
    try:
        returns = download_data(tickers, start_date, end_date)
        spy_returns = download_data(['SPY'], start_date, end_date)
        if returns is None or spy_returns is None:
            return
    except Exception as e:
        print(f"Erreur lors de la récupération des données : {e}")
        return

    # Étape 2 : Initialisation des variables
    try:
        train_data, test_data, spy_test_data = split_data(returns, spy_returns, j)
        metrics_list = []
        start_idx = 0
        weights_mat = []
        ptf_returns = []
    except Exception as e:
        print(f"Erreur lors de l'initialisation des variables : {e}")
```

```
        return

    # Étape 3 : Optimisation RP pour chaque tranche
    try:

          while start_idx + i <= len(test_data) and start_idx + i <= len(spy_test_data):

              # Découpage des données de test
              slice_data = test_data.iloc[start_idx:start_idx + i]
              spy_slice_data = spy_test_data.iloc[start_idx:start_idx + i].to_numpy().flatten()

              # Optimisation RP sur les données d'entraînement actuelles
              weights = optimize_risk_par(train_data)
              if weights is None:
                  return
              weights_mat.append(weight_dic(tickers,weights))

              # Calcul des rendements du portefeuille pour la tranche
              portfolio_returns = slice_data @ weights

              ptf_returns.extend(portfolio_returns)


              # Calcul des métriques
              metrics = calculate_metrics(portfolio_returns, spy_slice_data)
              metrics_list.append(metrics)
              # Mise à jour des données d'entraînement
              train_data = pd.concat([train_data, slice_data.iloc[:k]]).iloc[-j:]
              start_idx += k


          ptf_returns = pd.Series(ptf_returns)
          ptf_returns=ptf_returns.add(1).cumprod().sub(1)*100
          metrics_list = pd.DataFrame(metrics_list)
          weights_mat = pd.DataFrame(weights_mat)


          return metrics_list,weights_mat,ptf_returns
    except Exception as e:
          print(f"Erreur lors de l'optimisation ou du calcul des métriques : {e}")
          return
```

min-var portfolio backtester

```
def rolling_min_var_optimization(i, j, k, start_date, end_date, tickers):
    try:
        # Vérification des paramètres d'entrée
        if j < i:
            raise ValueError("Le deuxième paramètre (j) doit être supérieur au premier (i).")
    except ValueError as ve:
        print(f"Erreur de validation : {ve}")
        return None, None

    # Étape 1 : Récupération des données
    try:
        returns = download_data(tickers, start_date, end_date)
        spy_returns = download_data(['SPY'], start_date, end_date)

        if returns is None or spy_returns is None:
            return None, None
    except Exception as e:
        print(f"Erreur lors de la récupération des données : {e}")
        return None, None

    # Étape 2 : Initialisation
    try:
        train_data, test_data, spy_test_data = split_data(returns, spy_returns, j)
        metrics_list = []
        weights_list = []
        start_idx = 0
        ptf_returns = []
    except Exception as e:
        print(f"Erreur lors de l'initialisation : {e}")
        return None, None

    # Étape 3 : Optimisation et calcul des métriques pour chaque tranche
    try:

        while start_idx + i <= len(test_data) and start_idx + i <= len(spy_test_data):
```

```python
        # Découpage des données
        slice_data = test_data.iloc[start_idx:start_idx + i]
        spy_slice_data = spy_test_data.iloc[start_idx:start_idx + i].to_numpy().flatten()

        # Optimisation Min-Var
        weights = optimize_min_var(train_data)
        if weights is None:
            print(f"Tranche {len(metrics_list) + 1} ignorée : optimisation échouée.")
            break

        # Conversion des poids en format numpy array
        weights_array = np.array(list(weights.values()))*0.7
        weights_title = list(weights.keys())
        weights_array = [0.1 if x < 0.1 else x for x in weights_array]
        new_weights_dict = dict(zip(weights_title, weights_array))
        weights_list.append(new_weights_dict)
        # Calcul des rendements du portefeuille
        portfolio_returns = slice_data @ weights_array
        ptf_returns.extend(portfolio_returns)

        # Calcul des métriques
        metrics = calculate_metrics(portfolio_returns, spy_slice_data)
        metrics_list.append(metrics)
        if metrics is None:
            print(f"Tranche {len(metrics_list) + 1} ignorée : erreur dans les métriques.")
            break



        # Mise à jour des données d'entraînement
        train_data = pd.concat([train_data, slice_data.iloc[:k]]).iloc[-j:]
        start_idx += k
    ptf_returns = pd.Series(ptf_returns)
    ptf_returns = ptf_returns.add(1).cumprod().sub(1)*100

    # Étape 4 : Retour des résultats

    metrics_df = pd.DataFrame(metrics_list)
    weights_list = pd.DataFrame(weights_list)
    return metrics_df, weights_list,ptf_returns

except Exception as e:
    print(f"Erreur inattendue : {e}")
    return None, None
```

Equal weights portfolio backtester

```python
def rolling_equal_weight_BT(i, j, k, start_date, end_date, tickers):
    try:
        # Vérification des paramètres d'entrée
        if j < i:
            raise ValueError("Le deuxième paramètre (j) doit être supérieur au premier (i).")
    except ValueError as ve:
        print(f"Erreur de validation : {ve}")
        return None, None

    # Étape 1 : Récupération des données
    try:
        returns = download_data(tickers, start_date, end_date)
        spy_returns = download_data(['SPY'], start_date, end_date)

        if returns is None or spy_returns is None:
            return None, None
    except Exception as e:
        print(f"Erreur lors de la récupération des données : {e}")
        return None, None

    # Étape 2 : Initialisation
    try:
        train_data, test_data, spy_test_data = split_data(returns, spy_returns, j)
        metrics_list = []
        portfolio_returns_list = []
        start_idx = 0
    except Exception as e:
        print(f"Erreur lors de l'initialisation : {e}")
        return None, None

    # Étape 3 : Boucle sur les tranches de test
```

```
    try:

        while start_idx + i <= len(test_data) and start_idx + i <= len(spy_test_data):

            # Découpage des données
            slice_data = test_data.iloc[start_idx:start_idx + i]
            spy_slice_data = spy_test_data.iloc[start_idx:start_idx + i].to_numpy().flatten()

            # Portefeuille également pondéré (Equal Weight)
            equal_weights = np.ones(slice_data.shape[1]) / slice_data.shape[1]
            portfolio_returns = slice_data.dot(equal_weights)
            portfolio_returns_list.extend(portfolio_returns)


            # Calcul des métriques
            metrics = calculate_metrics(portfolio_returns, spy_slice_data)
            metrics_list.append(metrics)



            # Mise à jour des données d'entraînement
            train_data = pd.concat([train_data, slice_data.iloc[:k]]).iloc[-j:]
            start_idx += k

        portfolio_returns = pd.Series(portfolio_returns_list)
        ptf_returns = portfolio_returns.add(1).cumprod().sub(1)*100
        # Étape 4 : Retour des résultats
        metrics_df = pd.DataFrame(metrics_list)
        return metrics_df,ptf_returns

    except Exception as e:
        print(f"Erreur inattendue : {e}")
        return None, None
```

Benchmark backtester

```
def rolling_backtest_SPY(i, j, k, start_date, end_date):
    try:
        # Vérification des paramètres d'entrée
        if j < i:
            raise ValueError("Le deuxième paramètre (j) doit être supérieur au premier (i).")
    except ValueError as ve:
        print(f"Erreur de validation : {ve}")
        return None, None

    # Étape 1 : Récupérer les données de SPY
    try:
        spy_returns = download_data(['SPY'], start_date, end_date)
        if spy_returns is None:
            return None, None
    except Exception as e:
        print(f"Erreur lors de la récupération des données : {e}")
        return None, None

    # Étape 2 : Initialisation
    try:
        train_data, test_data, _ = split_data(spy_returns, spy_returns, j)
        metrics_list = []
        portfolio_returns_list = []
        start_idx = 0
    except Exception as e:
        print(f"Erreur lors de l'initialisation : {e}")
        return None, None

    # Étape 3 : Backtest pour chaque tranche
    try:
        while start_idx + i <= len(test_data):
            # Découpage des données de test
            slice_data = test_data.iloc[start_idx:start_idx + i]

            # Calcul des rendements du portefeuille (SPY utilisé directement)
            portfolio_returns = slice_data
            portfolio_returns_list.append(portfolio_returns)

            # Calcul des métriques
            metrics = calculate_metrics(portfolio_returns, portfolio_returns)
            metrics_list.append(metrics)
```

```
            # Mise à jour des données d'entraînement
            train_data = pd.concat([train_data, slice_data.iloc[:k]]).iloc[-j:]
            start_idx += k
    except Exception as e:
        print(f"Erreur lors du backtest ou du calcul des métriques : {e}")
        return None, None


    portfolio_returns = pd.concat(portfolio_returns_list)

    # Étape 4 : Retourner les résultats
    try:
        metrics_df = pd.DataFrame(metrics_list)
        return metrics_df,portfolio_returns
    except Exception as e:
        print(f"Erreur lors de la conversion en DataFrame : {e}")
        return None, None
```

Pre-COVID period

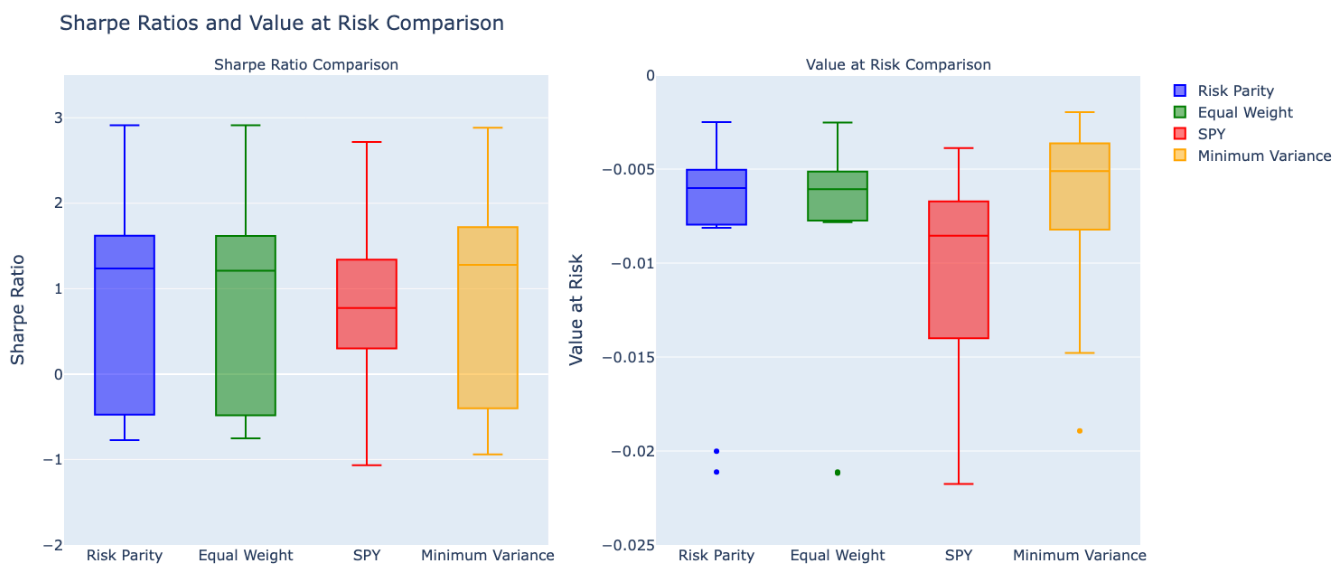
```
params = {
        "i": 63 ,
        "j": 189,
        "k": 63,
        "start_date": "2015-01-01",
        "end_date": "2019-06-01",
        "tickers": ['SPHQ','SPYD','SPLV','IVE','SPMO']
        }

rp_metrics, rp_port,rp_return = rolling_rp_optimization(**params)
ew_metrics,ew_port = rolling_equal_weight_BT(**params)
spy_metrics,spy_port = rolling_backtest_SPY(params["i"], params["j"], params["k"], params["start_date"], params["end_date"])
mv_metrics,mv_port,mv_return = rolling_min_var_optimization(**params)
```

```
from plotly.subplots import make_subplots
import plotly.graph_objects as go
```

```
def plot_ratios(column_name, k, l):
    # Combine ratios into a single DataFrame
    ratios = pd.concat(
        [
            rp_metrics[column_name].rename("Risk Parity"),
            ew_metrics[column_name].rename("Equal Weight"),
            spy_metrics[column_name].rename("SPY"),
            mv_metrics[column_name].rename("Minimum Variance")
        ],
        axis=1
    )

    # Create and display the box plot using Plotly
    fig = go.Figure()

    # Add box plots for each column in ratios
    for column in ratios.columns:
        fig.add_trace(go.Box(y=ratios[column], name=column))

    # Update layout for better visualization
    fig.update_layout(
        title=f"{column_name} ratios Comparison",
        yaxis_title=column_name,
        xaxis_title="Portfolio Types",
        boxmode="group",  # Better layout for comparison
        height=1000,       # Increase figure height
        width=1000,        # Increase figure width
        font=dict(size=10), # Font size for better readability
        yaxis=dict(
            title=column_name,
            range=[k, l]  # Set custom y-axis range
        )
    )

    # Return the figure object
    return fig
```

```
# Create subplots layout
```

```
fig = make_subplots(
    rows=1, cols=2,
    subplot_titles=["Sharpe Ratio Comparison", "Value at Risk Comparison"]
)

# Plot Sharpe Ratio and VaR figures
sharpe_fig = plot_ratios('Sharpe Ratio', -2, 3.5)
var_fig = plot_ratios('VaR (at 95% confidence)', -0.04, 0.005)

# Add traces with consistent colors
add_colored_traces(fig, sharpe_fig, row=1, col=1)
add_colored_traces(fig, var_fig, row=1, col=2)

# Update layout for the subplots with separate y-axis ranges
fig.update_layout(
    title="Sharpe Ratios and Value at Risk Comparison",
    height=700,
    width=1500,
    font=dict(size=16),
    yaxis1=dict(
        title="Sharpe Ratio",
        range=[-2, 3.5]
    ),
    yaxis2=dict(
        title="Value at Risk",
        range=[-0.025, 0]
    )
)

# Show the plot
fig.show()
```



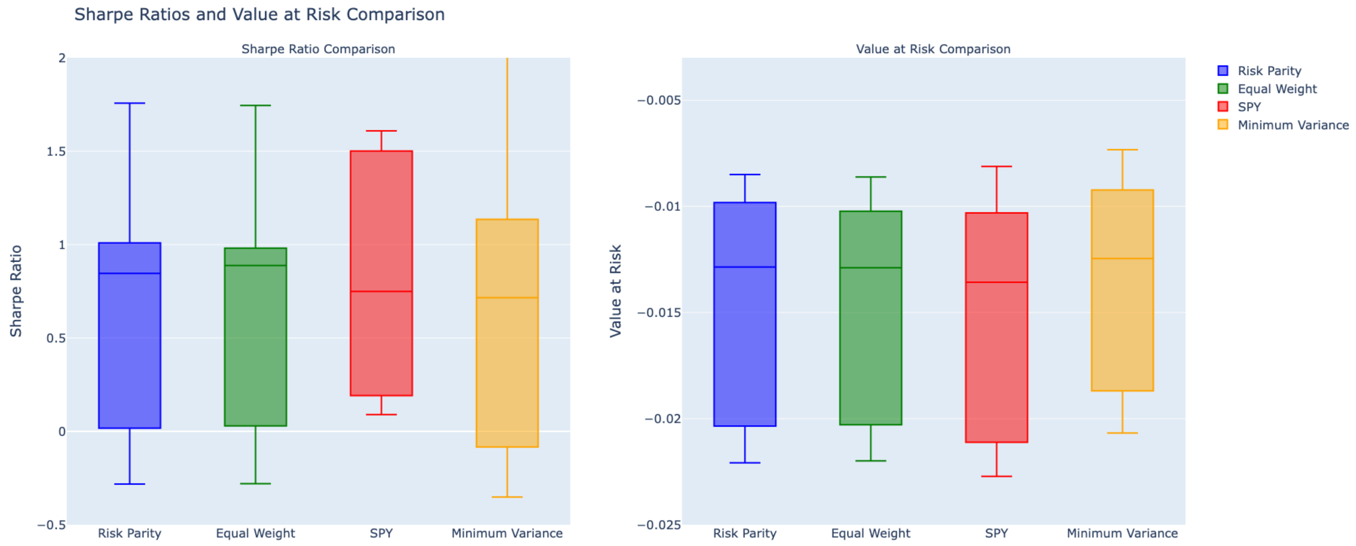
Pre-covid period conclusion:

The ERC portfolio has a distribution of its sharpe ratio similar to that of the equal weight portfolio, but compared with the benchmark it has a wider distribution, albeit in the positive. The same is true of the min-var. In terms of Value at Risk, the benchmark is making much bigger losses than the other 3 portfolios. The ERC is tighter than the min-var portfolio.

Covid period Backtest

```
### Stress Test (Covid period)
params = {
        "i": 63 ,
        "j": 189,
        "k": 63,
        "start_date": "2019-06-01", # premier cas Européen en France
        "end_date": "2021-12-01", # 2 mois après découverte du vaccin
```

```
        "tickers": ['SPHQ','SPYD','SPLV','IVE','SPMO']
        }

rp_metrics, rp_port, rp_return = rolling_rp_optimization(**params)
ew_metrics,ew_port = rolling_equal_weight_BT(**params)
spy_metrics,spy_port = rolling_backtest_SPY(params["i"], params["j"], params["k"], params["start_date"], params["end_date"])
mv_metrics,mv_port, mv_return = rolling_min_var_optimization(**params)

from plotly.subplots import make_subplots

# Define a custom color palette for consistent colors
color_palette = {
    "Risk Parity": "blue",
    "Equal Weight": "green",
    "SPY": "red",
    "Minimum Variance": "orange"
}

# Function to apply consistent colors when adding traces
def add_colored_traces(fig, input_fig, row, col):
    for trace in input_fig.data:
        trace.update(marker=dict(color=color_palette[trace.name]))  # Apply custom color
        fig.add_trace(trace, row=row, col=col)

# Create subplots layout without shared y-axes
fig = make_subplots(
    rows=1, cols=2,
    subplot_titles=["Sharpe Ratio Comparison", "Value at Risk Comparison"]
)

# Generate plots for Sharpe Ratio and Value at Risk (VaR)
sharpe_fig = plot_ratios('Sharpe Ratio', -1000, 1340)
var_fig = plot_ratios('VaR (at 95% confidence)', -0.03, 0.001)

# Add traces with consistent colors
add_colored_traces(fig, sharpe_fig, row=1, col=1)
add_colored_traces(fig, var_fig, row=1, col=2)

# Update layout with separate y-axis ranges and improved visuals
fig.update_layout(
    title="Sharpe Ratios and Value at Risk Comparison",
    height=800,
    width=1800,
    font=dict(size=16),
    yaxis1=dict(
        title="Sharpe Ratio",
        range=[-0.5, 2]
    ),
    yaxis2=dict(
        title="Value at Risk",
        range=[-0.025, -0.003]
    )
)

# Show the plot
fig.show()
```



COVID test conculsion :

The benchmark has a better sharpe ratio distribution (tighter and above the others in the positives) but the probable losses on this portfolio are greater, as shown by the Var distribution. The Risk parity portfolio remains once again the best performance-risk alternative. (This outperformance of the sharpe ratio of the benchmark compared to other portfolios during the covid period is due to the outperformance of the Tec sector.)

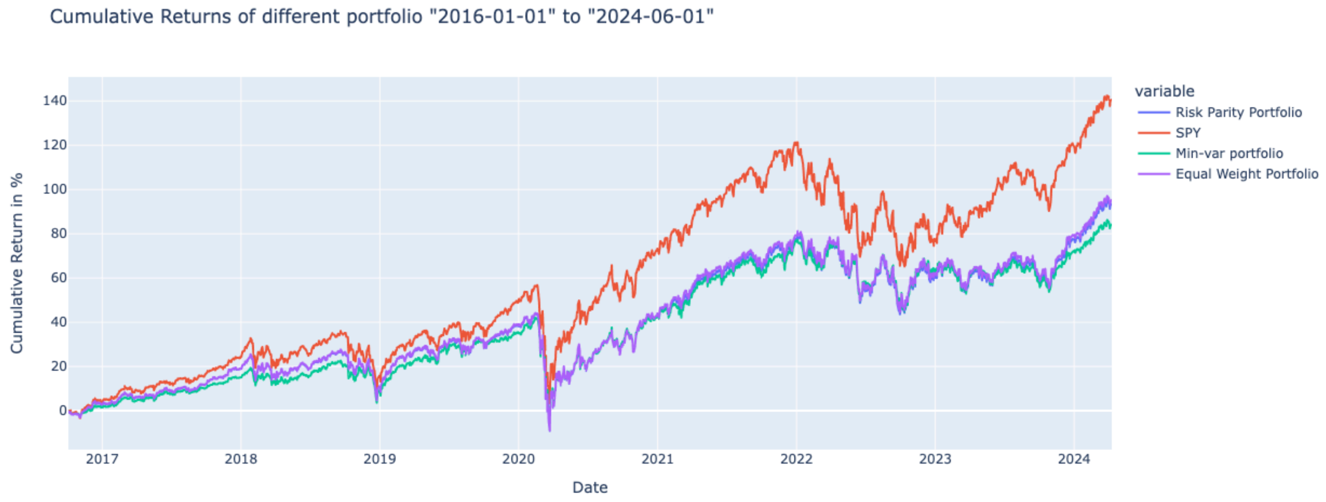
General BT conclusion:

Compared to other portfolios, the ERC has either a tighter distribution of its metrics (an assurance that it can be projected into the future) or a wider distribution but in the positive (which tells us that even if the value seems less certain, it will be positive).

Cumulative return

```
#L'objectif ici était de faire une agglomération annuelle des valeurs

params = {
        "i": 63,
        "j": 189,
        "k": 63,
        "start_date": "2016-01-01",
        "end_date": "2024-06-01",
        "tickers": ['SPHQ','SPYD','SPLV','IVE','SPMO']
        }

rp_metrics, rp_port, rp_return= rolling_rp_optimization(**params)
ew_metrics,ew_port= rolling_equal_weight_BT(**params)
spy_metrics, spy_port = rolling_backtest_SPY(params["i"], params["j"], params["k"], params["start_date"], params["end_date"])
mv_metrics,mw_port, mv_return = rolling_min_var_optimization(**params)
spy_port = spy_port.add(1).cumprod().sub(1)*100


rp_return = pd.DataFrame(rp_return)
rp_return.columns = ["Risk Parity Portfolio"]
rp_return.set_index(spy_port.index, inplace=True)

mv_return = pd.DataFrame(mv_return)
mv_return.columns = ["Min-var portfolio"]
mv_return.set_index(spy_port.index, inplace=True)

ew_port = pd.DataFrame(ew_port)
ew_port.columns = ["Equal Weight Portfolio"] # Changed column name to avoid duplication
ew_port.set_index(spy_port.index, inplace=True)

# Convert spy_port to Series before concatenation
spy_port = spy_port.squeeze() # or spy_port = spy_port.iloc[:, 0] if it has multiple columns
spy_port.name = "SPY" # Assign a name for the Series

df = pd.concat([rp_return,spy_port,mv_return,ew_port], axis = 1)

#Visualize cumulative returns of each stock in the portfolio
```

```
fig = px.line(df,
              x=df.index,
              y=df.columns,
              title='Cumulative Returns of different portfolio "2016-01-01" to "2024-06-01" ')

fig.update_xaxes(title_text='Date')
fig.update_yaxes(title_text='Cumulative Return in %')


fig.show()
```



The benchmark has a higher cumulative return than the other portfolios. However, it is exposed to greater risk, as demonstrated by its higher daily Value at Risk.

Cumulative statistics over the whole backtest period For 5 period of 350 business days (This involves aggregating data to obtain statistics covering a wide area.)

```
params = {
        "i": 350 ,
        "j": 350,
        "k": 350,
        "start_date": "2016-01-01",
        "end_date": "2024-08-01",
        "tickers": ['SPHQ','SPYD','SPLV','IVE','SPMO']
        }

rp_metrics, rp_port,rp_return= rolling_rp_optimization(**params)
ew_metrics,ew_port= rolling_equal_weight_BT(**params)
spy_metrics, spy_port = rolling_backtest_SPY(params["i"], params["j"], params["k"], params["start_date"], params["end_date"]
mv_metrics,mw_port,mv_return= rolling_min_var_optimization(**params)
a = rp_metrics[['Sharpe Ratio','Treynor Ratio','VaR (at 95% confidence)']].describe()
a
```

```
[*********************100%***********************]  5 of 5 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  5 of 5 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  5 of 5 completed
[*********************100%***********************]  1 of 1 completed
```

	Sharpe Ratio	Treynor Ratio	VaR (at 95% confidence)
count	5.000000	5.000000	5.000000
mean	1.556978	0.137478	-0.016125
std	1.203178	0.118543	0.005129
min	-0.174634	-0.019680	-0.022841
25%	0.845845	0.073791	-0.018153
50%	2.011940	0.141883	-0.017425
75%	2.323134	0.200911	-0.012488
max	2.778604	0.290486	-0.009716