

# View Value Estimator

November 18, 2025

```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
from fredapi import Fred
import warnings
warnings.filterwarnings("ignore")
```

```
[2]: df = pd.read_csv("data.csv", sep=";", decimal=",")
df = df.rename(columns={
    "Column1": "Date",
    "Column2": "SPX",
    "Column3": "S5SFTW",
    "Column4": "S5PHRM",
    "Column5": "S5CPGS",
    "Column6": "S5ENRSX",
    "Column7": "S5FDBT",
    "Column8": "S5TECH",
    "Column9": "S5RETL",
    "Column10": "S5BANKX",
    "Column11": "S5HCES",
    "Column12": "S5DIVF",
    "Column13": "S5UTILX",
    "Column14": "S5MEDA",
    "Column15": "S5REAL",
    "Column16": "S5TELSX",
    "Column17": "S5MATRX",
    "Column18": "S5INSU",
    "Column19": "S5FDSR",
    "Column20": "S5HOUS",
    "Column21": "S5SSEQX",
    "Column22": "S5TRAN",
    "Column23": "S5HOTR",
    "Column24": "S5CODU",
    "Column25": "S5AUCO",
    "Column26": "S5COMS",
})
df["Date"] = pd.to_datetime(df["Date"], format="%d/%m/%Y")
```

```
[3]: def GetReturn(df,date,lookback):
    date=pd.to_datetime(date)
    if date not in df["Date"].values:#add breaker if windows not in df
        raise ValueError("Date not in dataframe")
    returns_df =_
    df[["Date","S5SFTW","S5PHRM","S5CPGS","S5ENRSX","S5FDBT","S5TECH","S5RETL","S5BANKX","S5HCE
    copy()

    date_list=returns_df.drop(columns="Date")
    date_index = returns_df.index[returns_df["Date"] == date][0]
    returns_df=returns_df[(returns_df.index<=date_index) & (returns_df.
    index>=date_index-lookback) ]
    returns_df.drop(columns="Date",inplace=True)

    returns_df = np.log(returns_df/ returns_df.shift(1))
    returns_df.dropna(inplace=True)
    #print(returns_df.std().mean()) #verification if std is around 1% daily

    return returns_df

def GetReturnSPX(df,date,lookback):
    date=pd.to_datetime(date)
    if date not in df["Date"].values:#add breaker if windows not in df
        raise ValueError("Date not in dataframe")
    returns_df = df[["Date","SPX"]].copy()

    date_list=returns_df.drop(columns="Date")
    date_index = returns_df.index[returns_df["Date"] == date][0]
    returns_df=returns_df[(returns_df.index<=date_index) & (returns_df.
    index>=date_index-lookback) ]
    returns_df.drop(columns="Date",inplace=True)

    returns_df = np.log(returns_df/ returns_df.shift(1))
    returns_df.dropna(inplace=True)
    #print(returns_df.std().mean()) #verification if std is around 1% daily

    return returns_df

#Returns=GetReturn(df,"2020-05-11",lookback=180)
#ReturnsSPX=GetReturnSPX(df,"2020-05-11",lookback=180)

[4]: def GetSigma(df,date,lookback):
    returns_df=GetReturn(df,date,lookback=lookback)
    #covariance matrix from returns_df
    sigma_windowed=returns_df.cov()
```

```
return sigma_windowed
```

```
#Sigma=GetSigma(df, "2020-05-11", lookback=180)
```

```
[5]: def GetRfDataframe(df):
    fred = Fred(api_key="5c742a53d96bd3085e9199dcdb5af60b")
    riskfree = fred.get_series('DFF')
    # riskfree = fred.get_series('DTB1M0')

    riskfree = riskfree.to_frame(name='FedFunds')
    riskfree.index.name = "Date"
    riskfree = riskfree[riskfree.index >= "2002-01-01"]
    riskfree["FedFunds"] = riskfree["FedFunds"] / 100
    list_days_open = pd.to_datetime(df["Date"], dayfirst=True, errors="coerce")
    list_days_full = pd.to_datetime(riskfree.index, dayfirst=True,
    ↪ errors="coerce")

    list_days_open = [pd.to_datetime(date) for date in list_days_open]
    list_days_full = [pd.to_datetime(date) for date in list_days_full]

    list_days_open_pondered = []
    riskfree_list = []
    count_list = []
    timestamp = 0
    while timestamp < len(list_days_full) - 1:

        if list_days_full[timestamp + 1] in list_days_open:
            list_days_open_pondered.append(list_days_full[timestamp])
            riskfree_list.append(riskfree["FedFunds"].
    ↪ loc[list_days_full[timestamp]])
            count_list.append(1)
            timestamp += 1

        else:
            count = 0
            timestampbis = timestamp
            while (timestamp + 1 < len(list_days_full)) and
    ↪ (list_days_full[timestamp + 1] not in list_days_open):
                timestamp += 1
                count += 1

            list_days_open_pondered.append(list_days_full[timestampbis]) # jour
    ↪ de départ
            riskfree_list.append(riskfree["FedFunds"].
    ↪ loc[list_days_full[timestampbis]])
            count_list.append(count + 1)
```

```

        timestamp += 1

    RfDf=pd.DataFrame({"Date":list_days_open_pondered,"Rf":
↪riskfree_list,"Count":count_list})
    RfDf=RfDf.set_index("Date")
    return RfDf

def GetRiskFree(df,date,lookback,RfDf):
    positionOfStartDate=df.index[df["Date"]==pd.to_datetime(date)][0]-lookback
    #print(positionOfStartDate)
    startDate=pd.to_datetime(df.iloc[positionOfStartDate,0])
    endDate=pd.to_datetime(date)
    RfDf=RfDf[(RfDf.index >= startDate) & (RfDf.index <= endDate)].copy()
    CumulativeRf=[]

    for i in range(len(RfDf)):
        if i==0:
            CumulativeRf.append(pow((1+RfDf["Rf"].iloc[i]),(RfDf["Count"].iloc[i]/
↪360))))
        else:
            CumulativeRf.append(pow((1+RfDf["Rf"].iloc[i]),(RfDf["Count"].iloc[i]/
↪360)))*CumulativeRf[i-1])

    RfDf["CumulativeRf"]=CumulativeRf
    RfDf["CumulativeRf"]= RfDf["CumulativeRf"]-1

    return RfDf["CumulativeRf"].iloc[-1]

RfDf=GetRfDataframe(df)

```

```

[6]: def GetWeight(df,date):
    #for the moment we will use the equal weight
    weight_vector=np.zeros((24,1))
    for i in range(0,24):
        weight_vector[i]=1/24

    return weight_vector
#Weight=GetWeight(df,"2020-05-11")

```

```

[7]: def GetLambda(df,date,timeofcalculation,RfDf):
    returns=GetReturn(df,date,timeofcalculation)
    weight_vector=GetWeight(df=0,date=0)

    mean_return=np.mean(np.dot(returns,weight_vector))
    mean_annual=(1+mean_return)**252-1

```

```

rf_temps=GetRiskFree(df,date,timeofcalculation,RfDf)
rf_annual=(1+rf_temps)**(252/timeofcalculation)-1

Sigma=GetSigma(df,date,timeofcalculation)
Sigma_annual=252*Sigma
var = float((weight_vector.T @ Sigma_annual.values @ weight_vector).item())
print(var)
lambda_value=(mean_annual - rf_annual)/var

excess = mean_annual - rf_annual
sigma2 = var
sigma = np.sqrt(var)
lam = excess / sigma2
sharpe = excess / sigma
print("Excess:", excess, " Var:", sigma2, " Vol:", sigma, " :", lam, "␣
↳Sharpe:", sharpe)

return lambda_value

Lambda=GetLambda(df,"2024-01-11",timeofcalculation=3500,RfDf=RfDf)

```

0.027917132194784967

Excess: 0.08824333617258762 Var: 0.027917132194784967 Vol: 0.1670842068981535  
: 3.1609026155298223 Sharpe: 0.5281369065980994

```

[29]: def GetPMatrix(df,date, lookback,proportion=3):
    #(date)
    #print(proportion)
    #print(lookback)
    ␣
    ↳AssetColumns=["S5SFTW","S5PHRM","S5CPGS","S5ENRSX","S5FDBT","S5TECH","S5RETL","S5BANKX","S5
    bestperformer = []
    performerc = []
    returnBestPerformer=[]
    returnWorstPerformer=[]

    endDateIndex=df.index[df["Date"]==pd.to_datetime(date)][0]
    startDateIndex=df.index[df["Date"]==pd.to_datetime(date)][0]-lookback

    for i in range(2, df.shape[1]): #loop through asset columns
        performerc.append((((float(df.iloc[endDateIndex, i]) / float(df.
    ↳iloc[startDateIndex, i]) - 1) * 100), i - 2,df.columns[i])) #pos of best␣
    ↳stock in a tuple with its return

```

```

performerc.sort(reverse=True)
#print(performerc)
perfMarket= (float(df.iloc[endDateIndex, 1]) / float(df.
↪iloc[startDateIndex, 1]) - 1) * 100
#print(f"Market performance over the period : {perfMarket}%")

for i in range(proportion):
    bestperformer.append(performerc[i][1])
    returnBestPerformer.append(performerc[i][0])

P=np.zeros((proportion,24))
Q=np.zeros((proportion,1))
for lineview in range(proportion):
    for i in range(len(AssetColumns)):
        P[lineview,i]=-1/len(AssetColumns)
        P[lineview,bestperformer[lineview]]=1-1/len(AssetColumns)
        sum=0
        for i in range(len(AssetColumns)):
            sum+=P[lineview,i]
        print("ma somme",sum)
    for i in range(proportion):
        Q[i,0]=((returnBestPerformer[i]-perfMarket)/2)/100

print("P : ",P,"Q : ",Q)

return P, Q
PMatrix,TempoQ=GetPMatrix(df,"2016-05-11",lookback=180,proportion=3)

```

```

ma somme 4.163336342344337e-17
ma somme 2.636779683484747e-16
ma somme 2.636779683484747e-16
P :  [[-0.04166667 -0.04166667 -0.04166667 -0.04166667 -0.04166667 -0.04166667
      -0.04166667 -0.04166667 -0.04166667 -0.04166667  0.95833333 -0.04166667
      -0.04166667 -0.04166667 -0.04166667 -0.04166667 -0.04166667 -0.04166667
      -0.04166667 -0.04166667 -0.04166667 -0.04166667 -0.04166667 -0.04166667]
      [-0.04166667 -0.04166667 -0.04166667 -0.04166667 -0.04166667 -0.04166667
      -0.04166667 -0.04166667 -0.04166667 -0.04166667 -0.04166667  0.95833333
      -0.04166667 -0.04166667 -0.04166667 -0.04166667 -0.04166667 -0.04166667]
      [-0.04166667 -0.04166667 -0.04166667 -0.04166667  0.95833333 -0.04166667
      -0.04166667 -0.04166667 -0.04166667 -0.04166667 -0.04166667 -0.04166667
      -0.04166667 -0.04166667 -0.04166667 -0.04166667 -0.04166667 -0.04166667
      -0.04166667 -0.04166667 -0.04166667 -0.04166667 -0.04166667 -0.04166667] Q :

```

```
[0.06942108]
[0.06318248]
[0.06245929]]
```

```
[9]: def GetOmega(PMatrix, Sigma, c=0.99):
    #Omega is the uncertainty of the views
    factorC=(1/c-1)
    Omega=factorC*PMatrix@Sigma@np.transpose(PMatrix)

    return Omega
```

```
[10]: def LinkOmegaTau(Omega, P, Sigma):
    #Link omega to tau
    constant=36

    multiple= np.trace(np.transpose(P) @ np.linalg.inv(Omega) @ P) * constant
    numerator= np.trace(np.linalg.inv(Sigma*252))

    result= numerator / multiple
    return result
```

```
[27]: def BlackAndLittermanModel(backtestStartDate, rebalancingFrequency,
    ↳lookbackPeriod, df,RfDf,confidence=0.25,proportion=3,tau=0.01,Lambda=3):
    #implement the full backtest of the black and litterman model

    #-----
    #PARAMETERS
    #-----
    Sigma=GetSigma(df,backtestStartDate,lookback=lookbackPeriod)
    PMatrix,Q= GetPMatrix(df,backtestStartDate,
    ↳lookback=lookbackPeriod,proportion=proportion)
    Omega=GetOmega(PMatrix, Sigma, c=confidence)
    rf=GetRiskFree(df,backtestStartDate,lookbackPeriod,RfDf)
    weights = GetWeight(df, backtestStartDate)
    weights = np.array(weights).reshape(-1, 1)
    uimplied = Lambda * (Sigma @ weights) + rf
    #BL formula
    OmegaLinked=LinkOmegaTau(Omega,PMatrix,Sigma)
    #tau=OmegaLinked
    tau=0.01

    optimizedReturn=(np.linalg.inv(np.linalg.inv(tau*Sigma)+np.
    ↳transpose(PMatrix)@np.linalg.inv(Omega)@PMatrix)) @ (np.linalg.
    ↳inv(tau*Sigma)@uimplied+np.transpose(PMatrix)@np.linalg.inv(Omega)@Q)
```

```

#MarkowitzAllocation
WeightBL=np.linalg.inv(Sigma)@(optimizedReturn-rf)/Lambda
print(np.sum(WeightBL))

if not np.isclose(float(np.sum(WeightBL)), 1.0, atol=1e-6):
    print(WeightBL)
    raise ValueError("Weights do not sum to 1, please investigate.")

return WeightBL

```

```

BlackAndLittermanModel("2018-05-11", rebalancingFrequency=3,
↳lookbackPeriod=180, df=df,RfDf=RfDf)

```

```

is optimizedReturn    uimplied ? False
0      1.0
dtype: float64
sum wrong = 0      1.0
dtype: float64

```

```

-----
AttributeError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_14016\1367375549.py in ?()
    37
    38     return WeightBL
    39
    40
---> 41 BlackAndLittermanModel("2018-05-11", rebalancingFrequency=3,
↳lookbackPeriod=180, df=df,RfDf=RfDf)
    42

~\AppData\Local\Temp\ipykernel_14016\1367375549.py in ?(backtestStartDate,
↳rebalancingFrequency, lookbackPeriod, df, RfDf, confidence, proportion, tau,
↳Lambda)
    27     #MarkowitzAllocation
    28     WeightBL=np.linalg.inv(Sigma)@(optimizedReturn-rf)/Lambda
    29     print(np.sum(WeightBL))
    30     print("sum wrong =", np.sum(WeightBL))
---> 31     print("difference =", np.sum(WeightBL.flatten()) - np.sum(WeightBL)
    32
    33
    34     if not np.isclose(float(np.sum(WeightBL)), 1.0, atol=1e-6):

```

```

D:\visual\Desktop\Research-Project\venv\lib\site-packages\pandas\core\generic.p
↪in ?(self, name)
    6317             and name not in self._accessors
    6318             and self._info_axis.
↪_can_hold_identifiers_and_holds_name(name)
    6319         ):
    6320             return self[name]
-> 6321         return object.__getattr__(self, name)

AttributeError: 'DataFrame' object has no attribute 'flatten'

```

```

[12]: from rich.console import Console
      from rich.panel import Panel
      from tqdm import tqdm

console = Console()

#BACK TESTER
dfbacktest=df.copy()
dfbacktest["Date"] = pd.to_datetime(df["Date"], format="%d/%m/%Y")
dfbacktest["MonthIndex"] = dfbacktest["Date"].dt.to_period("M")

df_length = dfbacktest.shape[1] - 2 # bcs of date and spx
last_rebalance = dfbacktest.loc[0, "Date"] # première date
month_count = 0

# AFFICHAGE STYLE (sans prompts)
hold = 1
hist = 0
proportion = 3
Lambda=3
tau=0.01
confidence=0.00001

console.print(Panel.fit(
    "[bold cyan] PORTFOLIO BACKTESTER[/bold cyan]\n"
    "[dim]Black-Litterman Model[/dim]",
    border_style="cyan"
))

console.print(f"\n[yellow] Configuration :[/yellow]")
console.print(f"    • Hold period: [cyan]{hold}[/cyan] mois")
console.print(f"    • Historique: [cyan]{hist}[/cyan] mois")
console.print(f"    • Proportion: [cyan]{proportion}[/cyan]")
console.print(f"    • Lambda: [cyan]{Lambda:.4f}[/cyan]")
console.print(f"    • Confiance: [cyan]{confidence}[/cyan]")

```

```

console.print(f"    • Taux: [cyan]{tau}[/cyan]\n")

console.print("\n[yellow] Lancement du backtest...[/yellow]\n")

def Backtester(df,hold, hist, proportion,df_toBL,
↳RfDf,confidence2,proportion2,tau2,Lambda2):
    #new dataframe for stock quantity

    StockQty = df.copy()
    StockQty.drop(columns="MonthIndex", inplace=True)
    start=181

    StockQty.loc[:, :] = 0
    #starting data
    MoneyAtStart = 10000000
    month_count=0
    CurrentValue=MoneyAtStart

    #first ligne
    StockQty.loc[start, "Money"] = MoneyAtStart
    StockQty.loc[start, "SPX"] = df.iloc[start, 1]
    StockQty.loc[start, "Date"] = df.iloc[start, 0]

    #start of the algorithm

    for i in tqdm(range(start,df.shape[0]), desc="Backtesting"):
        StockQty.iloc[i,0]=df.iloc[i,0]
        StockQty.iloc[i,1]=df.iloc[i,1]
        fees=0

        if df.loc[i, "Date"].month != df.loc[i-1, "Date"].month:
            month_count += 1

        # Si on atteint la période voulue
        if i>= hist and month_count % hold == 0 and df.loc[i, "Date"].month != df.
↳loc[i - 1, "Date"].month:
            #print(f" Rebalancement déclenché à la date : {df.loc[i, 'Date'].
↳date()})")
            #print(str(df.iloc[i,0]))

        BLWeight=BlackAndLittermanModel(str(df.
↳iloc[i,0]),3,3*22,df_toBL,RfDf,confidence=confidence2,proportion=proportion2,tau=tau2,Lambda2)

```

```

        #print(len(BLWeight))
        for index in range(len(BLWeight)):
            StockQty.iloc[i,index+2]=(BLWeight.iloc[index,0]*CurrentValue)/df.
            ↪iloc[i,index+2] #qty = weight*total value/price

        else :
            for stocks in range(2,StockQty.shape[1]-1):
                StockQty.iloc[i,stocks]=StockQty.iloc[i-1,stocks] #same qty
                #value of pf

            GainOrLoss = 0
            for stocks in range(2, StockQty.shape[1]-1):
                qty = StockQty.iloc[i, stocks]

                if qty != 0.0:
                    price_now = df.iloc[i, stocks]
                    price_prev = df.iloc[i - 1, stocks]
                    GainOrLoss += qty * (price_now - price_prev)

            CurrentValue+=GainOrLoss-fees
            StockQty.iloc[i,-1]=CurrentValue

            StockQty = StockQty.iloc[start:].reset_index(drop=True)
            return StockQty
RfDf=GetRfDataframe(df)
final = Backtester(dfbacktest, hold=hold, hist=hist, proportion=proportion,
            ↪df_toBL=df,RfDf=RfDf,confidence2=confidence,proportion2=proportion,tau2=tau,Lambda2=Lambda)

console.print("\n[green] Backtest terminé avec succès ![/green]\n")

```

## PORTFOLIO BACKTESTER

Black-Litterman Model

### Configuration :

- Hold period: 1 mois
- Historique: 0 mois
- Proportion: 3

- Lambda: 3.0000
- Confiance: 1e-05
- Taux: 0.01

Lancement du backtest...

Backtesting: 100% | 5602/5602 [00:33<00:00, 165.65it/s]

Backtest terminé avec succès !

```
[13]: import plotly.express as px
import pandas as pd

money_norm = (final["Money"]/10000000*100) - 100
spx_norm = (final["SPX"]/final["SPX"].iloc[0]*100) - 100

df_plot = pd.DataFrame({
    "Date": final["Date"],
    "Portfolio": money_norm,
    "SPX": spx_norm
}).melt(id_vars="Date", var_name="Série", value_name="Évolution en %")

fix = px.line(
    df_plot,
    x="Date",
    y="Évolution en %",
    color="Série",
    color_discrete_map={"SPX": "red", "Portfolio": "green"},
    title="Comparaison des évolutions en %"
)

fix.update_layout(hovermode="x unified")
fix.show()
```

```
[14]: AnnualizedDf=final[["Date", "SPX", "Money"]]
AnnualizedDf['Date'] = pd.to_datetime(AnnualizedDf['Date'])
AnnualizedDf['Year'] = AnnualizedDf['Date'].dt.year
```

```

YearList=AnnualizedDf["Year"].unique()
SPXAnnualized=pd.DataFrame(columns=YearList)
StratAnnualized=pd.DataFrame(columns=YearList)

for year in YearList:
    compteurPerYear=0
    for i in AnnualizedDf.index:
        if AnnualizedDf.loc[i,"Year"]==year:
            if compteurPerYear==0:
                SPXAnnualized.loc[compteurPerYear,year]=AnnualizedDf.loc[i,"SPX"]
                StratAnnualized.loc[compteurPerYear,year]=AnnualizedDf.loc[i,"Money"]
            else :
                SPXAnnualized.loc[compteurPerYear,year]=AnnualizedDf.loc[i,"SPX"] /
↳ SPXAnnualized.loc[0,year]*100-100
                StratAnnualized.loc[compteurPerYear,year]=AnnualizedDf.loc[i,"Money"] /
↳ StratAnnualized.loc[0,year]*100-100
                compteurPerYear+=1

for year in YearList:
    SPXAnnualized.loc[0,year]=SPXAnnualized.loc[0,year]/SPXAnnualized.
↳ loc[0,year]*100-100
    StratAnnualized.loc[0,year]=StratAnnualized.loc[0,year]/StratAnnualized.
↳ loc[0,year]*100-100

SPXAvg=[]
StratAvg=[]
for i in SPXAnnualized.index:
    sumSPX=0
    sumStrat=0
    for year in SPXAnnualized.columns:
        sumSPX+=SPXAnnualized.loc[i,year]
        sumStrat+=StratAnnualized.loc[i,year]
    SPXAvg.append(sumSPX/len(YearList))
    StratAvg.append(sumStrat/len(YearList))

SPXAnnualized=SPXAnnualized.drop(columns=[2024,2002]) #too much nan
StratAnnualized=StratAnnualized.drop(columns=[2024,2002])

SPXAvg=[]
StratAVG=[]

```

```

for i in SPXAnnualized.index:
    sumSPX=0
    sumStrat=0
    for year in SPXAnnualized.columns:
        sumSPX+=SPXAnnualized.loc[i,year]
        sumStrat+=StratAnnualized.loc[i,year]
    SPXAvg.append(sumSPX/len(YearList))
    StratAVG.append(sumStrat/len(YearList))

dff = pd.DataFrame({"Index": (range(len(SPXAvg))), "Portfolio": StratAVG, "SPX": ↵
    ↵SPXAvg})

fig = px.line(dff, x="Index", y=["SPX", "Portfolio"], color_discrete_map={"SPX": ↵
    ↵"red", "Portfolio": "green"})
fig.show()

```

[ ]: