In [ ]:
```
pip install yfinance
```

```
Requirement already satisfied: yfinance in /usr/local/lib/python3.7/dist-packages (0.1.59)
Requirement already satisfied: lxml>=4.5.1 in /usr/local/lib/python3.7/dist-packages (from yfinance) (4.6.3)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.7/dist-packages (from yfinance) (0.0.9)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.7/dist-packages (from yfinance) (1.1.5)
Requirement already satisfied: requests>=2.20 in /usr/local/lib/python3.7/dist-packages (from yfinance) (2.23.0)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.7/dist-packages (from yfinance) (1.19.5)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.24->yfinance) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.24->yfinance) (2018.9)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests>=2.20->yfinance) (2020.12.5)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests>=2.20->yfinance) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests>=2.20->yfinance) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests>=2.20->yfinance) (1.24.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas>=0.24->yfinance) (1.15.0)
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

In [ ]:
```
pip install cvxpy
```

```
Requirement already satisfied: cvxpy in /usr/local/lib/python3.7/dist-packages (1.0.31)
Requirement already satisfied: osqp>=0.4.1 in /usr/local/lib/python3.7/dist-packages (from cvxpy) (0.6.2.post0)
Requirement already satisfied: ecos>=2 in /usr/local/lib/python3.7/dist-packages (from cvxpy) (2.0.7.post1)
Requirement already satisfied: scs>=1.1.3 in /usr/local/lib/python3.7/dist-packages (from cvxpy) (2.1.2)
Requirement already satisfied: multiprocess in /usr/local/lib/python3.7/dist-packages (from cvxpy) (0.70.11.1)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.7/dist-packages (from cvxpy) (1.19.5)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from cvxpy) (1.4.1)
Requirement already satisfied: qdldl in /usr/local/lib/python3.7/dist-packages (from osqp>=0.4.1->cvxpy) (0.1.5.post0
)
Requirement already satisfied: dill>=0.3.3 in /usr/local/lib/python3.7/dist-packages (from multiprocess->cvxpy) (0.3.
3)
```

Packages

In [ ]:
```
import cvxpy
import yfinance
import pandas as pd
import numpy as np
import calendar as cd
import matplotlib.pylab as plt

%matplotlib inline
```

Extraction des données

In [ ]:
```
mapping=pd.read_excel("DataProjets.xlsx",sheet_name="Mapping")[["Sedol","Tickers"]]
data_history = pd.read_csv('data.csv')
data_history
```

Out[ ]:

	Date	AMZN	AES	IBM	AMD	ADBE	APD	BXP	ALL	HON	AA	AMGN	HES	
0	2003-01-02	19.570000	2.498943	51.489540	7.010000	12.759070	25.624266	17.012911	24.491812	15.572003	44.319050	39.027912	14.669938	24
1	2003-01-03	20.520000	2.660913	52.179745	6.940000	13.107950	25.588125	17.082262	24.536983	15.428631	44.789524	38.925026	14.672572	24
2	2003-01-06	20.700001	2.660913	53.419521	7.160000	13.711017	26.491444	17.142359	24.833776	15.952265	45.881027	39.779675	15.012018	25
3	2003-01-07	21.549999	2.599210	54.959698	7.170000	14.216137	26.407146	17.128494	24.407940	15.677981	45.881027	39.755924	14.633093	25
4	2003-01-08	21.020000	2.583784	53.802956	6.690000	13.528016	25.612219	16.948195	24.253092	15.272788	41.119778	38.679737	14.472581	25
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4571	2021-02-23	3194.500000	27.219999	120.709999	84.739998	467.799988	263.329987	101.139999	108.010002	204.136398	24.900000	232.460007	68.260002	136
4572	2021-02-24	3159.530029	28.000000	123.209999	86.940002	476.619995	263.260010	104.690002	108.360001	210.100006	27.200001	229.990005	68.800003	139
4573	2021-02-25	3057.159912	26.350000	122.470001	82.419998	459.160004	260.380005	102.720001	108.830002	204.789993	25.559999	227.520004	66.959999	137
4574	2021-02-26	3092.929932	26.559999	118.930000	84.510002	459.670013	255.619995	99.129997	106.599998	202.350006	24.549999	224.919998	65.529999	135
4575	2021-03-01	3115.225098	26.830000	121.084999	84.336098	464.970001	260.929993	100.709999	108.989998	206.500000	25.660000	226.919998	66.410004	138

4576 rows × 562 columns

In [ ]:
```
#Replace nan with stock prices if needed (no replacement in some cases)
for i in range(len(data_history)-1):
  for column in data_history.columns:
    if str(data_history.at[i,column]) =='nan' and str(data_history.at[i+1,column])!='nan':
      try:
        data_history.at[i,column]=data_history.at[i-1,column]
      except:
        pass
```

In [ ]:
```
MarketCaps=pd.read_csv("profil_lam10_k02_estomega.csv")   #("/content/profil_min_risk_441.csv") ##A modifier en fct du
portefeuille
dic_s_to_t={}
for i in range(len(mapping)):
  l = mapping.iloc[i]
  dic_s_to_t[l['Sedol']]=l['Tickers']

dic_s_to_t['Unnamed: 0']='Date' #No name for this column in the excel file

MarketCaps=MarketCaps.rename(columns=dic_s_to_t)
MarketCaps
```

Out[ ]:

	Date	AMZN	AES	IBM	AMD	ADBE	APD	BXP	ALL	HON	AA	AMGN	HES	AXP	AEP	AFL	AIG	ADI	ALXN	VLO	APA	CMCSA	AAPL	AMA
0	2005-02-28	0.0	0.0	0.0	-0.0	0.0	0.0	0.0	0.0	0.0	-0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	0.
1	2005-03-31	0.0	0.0	0.0	-0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.0	0.0	-0.0	0.33589	0.0	0.0	0.0	0.
2	2005-04-29	0.0	0.0	-0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.0	0.0	-0.0	0.16410	0.0	0.0	0.0	0.
3	2005-05-31	0.0	0.0	-0.0	-0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.0	0.0	0.0	0.00000	0.0	0.0	-0.0	0.
4	2005-06-30	0.0	0.0	-0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00000	0.0	0.0	-0.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
186	2020-08-31	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.00000	0.0	0.0	0.0	0.
187	2020-09-30	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	0.
188	2020-10-30	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.00000	-0.0	0.0	0.0	0.
189	2020-11-30	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	0.
190	2020-12-31	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	0.

191 rows × 442 columns

In [ ]:
```
##On commence en MARS 2005
data_history=data_history.loc[data_history["Date"]>="2005-03-01"].reset_index(drop=True)
```

In [ ]:
```
## On construit le portefeuille AUTOFINANCE correspondant à l'indice précèdent (Attention dernière ligne)
w=MarketCaps.set_index("Date")
origin_index=list(w.columns)

v_data=data_history[data_history["Date"]<="2021-01-31"][['Date']]

data_prices=data_history[data_history["Date"]<="2021-01-31"].set_index("Date")[origin_index]

list_strat=[w.iloc[0]]
v_0=100

v=[v_0]

list_strat=[w.iloc[0]]
list_dates=list(data_prices.index)

i=1
for date in list_dates[1:]:
  if (list_dates[i][5:7])==(list_dates[i-1][5:7]):
    v+=[v[-1]*((list_strat[-1]*(data_prices.loc[date]/data_prices.loc[list_dates[i-1]])).sum())]

  else:
    list_strat.append(w.loc[list_dates[i-1]])
    v+=[v[-1]*((list_strat[-1]*(data_prices.loc[date]/data_prices.loc[list_dates[i-1]])).sum())]
  i+=1

v_data['Price']=v
```

In [ ]:
```
## On transforme le pandas en Date et Valuer du Portefeuille (Ce qu'on veut) + Purificaton des données sur le Yield
v_data.index = v_data["Date"]
own_index_price = v_data["Price"]

yield_own_index=(own_index_price.diff()/own_index_price).iloc[1:]
yield_own_index

x = yield_own_index[yield_own_index.between(yield_own_index.quantile(0.001), yield_own_index.quantile(0.999))] # witho
ut outliers

yield_own_index = x.to_frame()

yield_own_index = pd.DataFrame.rename(yield_own_index,columns={"Price": "Yield"})

own_index_price = own_index_price.to_frame()
own_index_price = pd.DataFrame.rename(own_index_price, columns = {0: "Price"})

## Tableau Yield et Index Price

yield_and_price=yield_own_index.join(own_index_price)
yield_and_price=pd.DataFrame([yield_own_index["Yield"],own_index_price["Price"]])
yield_and_price=yield_and_price.transpose().dropna()
```

In [ ]:
```
##Pour garder les valeurs communes

yield_own_index = pd.DataFrame(yield_and_price["Yield"])
own_index_price = pd.DataFrame(yield_and_price["Price"])
```

In [ ]:
```
##Tableau US Equity
Bench=pd.read_excel("DataProjets.xlsx",sheet_name="Benchmark")
Bench=Bench[["Unnamed: 0","US Equity"]]
Bench=pd.DataFrame.rename(Bench,columns={"Unnamed: 0":"Date","US Equity":"US Equity"})

yield_own_index_US=pd.DataFrame((Bench["US Equity"].diff()/Bench["US Equity"]).iloc[:])

res=(pd.concat([Bench["Date"],yield_own_index_US],axis=1)).dropna()
res.index = pd.to_datetime(res["Date"])

R_US=pd.DataFrame(res["US Equity"])

Rendement_US = R_US


Tableau_index_bench=yield_own_index.join(Rendement_US)
Tableau_index_bench=pd.DataFrame([Tableau_index_bench["Yield"],Tableau_index_bench["US Equity"]])
Tableau_index_bench=Tableau_index_bench.transpose().dropna()

Tableau_index_bench
```

Out[ ]:

	Yield	US Equity
2005-03-02	0.019290	0.000070
2005-03-03	0.016830	0.000328
2005-03-04	0.026284	0.009539
2005-03-07	-0.016524	0.002642
2005-03-08	-0.039533	-0.004723
...	...	...
2021-01-25	0.005326	0.003603
2021-01-26	-0.004643	-0.001485
2021-01-27	0.017793	-0.026353
2021-01-28	-0.003943	0.009766
2021-01-29	-0.018329	-0.019556

3928 rows × 2 columns

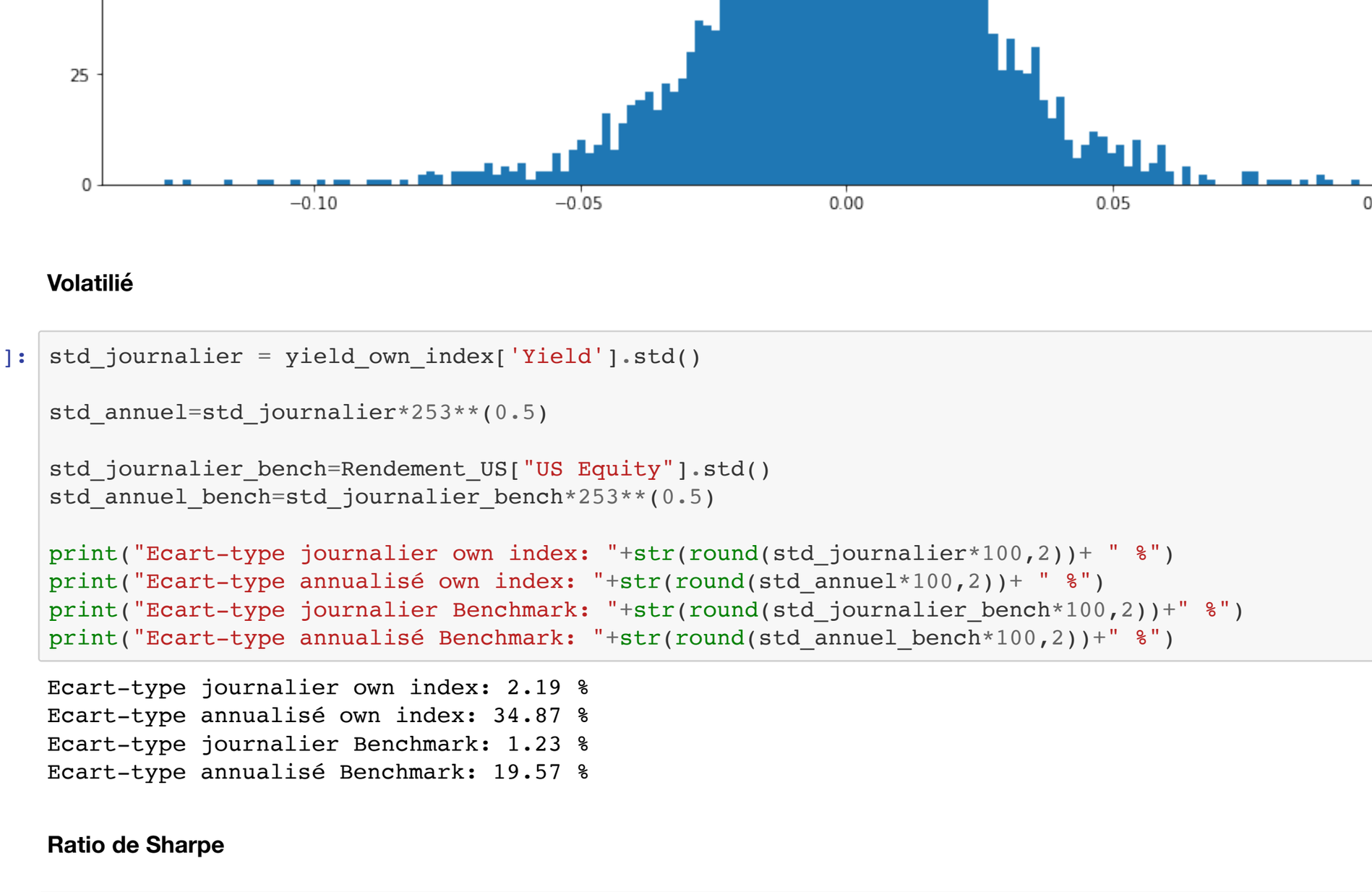
In [ ]:
```
figure = plt.figure(figsize=(25, 8))

ax1 = figure.add_subplot(211)
plt.plot(Tableau_index_bench.index,(Tableau_index_bench["Yield"]))
ax1.legend(["Yield"])
plt.title("Portefeuille")

ax2= figure.add_subplot(212)

plt.plot(Tableau_index_bench.index,Tableau_index_bench["US Equity"])
ax2.legend(["Yield"])
plt.title("Benchmark")
```

Out[ ]: Text(0.5, 1.0, 'Benchmark')

Key Risk Indicators

In [ ]:
```
##Tracer l'histogramme des rendements journaliers

list_rendement=[x[0] for x in yield_own_index.values]
def plot_hists():
  figure = plt.figure(figsize=(15, 20))
  ax1 = figure.add_subplot(211)

  plt.hist(list_rendement, bins=150)

plot_hists()


rendement_journalier=yield_own_index['Yield'].mean()
rendement_annuel=(1+rendement_journalier)**253-1

rendement_journalier_bench=Rendement_US["US Equity"].mean()
rendement_annuel_bench=(1+rendement_journalier_bench)**253-1

print("rendement journalier moyen Portefeuille "+str(round(rendement_journalier*100,3))+'%')
print("rendement journalier moyen Benchmark "+str(round(rendement_journalier_bench*100,3))+'%')

print("rendement annuel moyen Portefeuille "+str(round(rendement_annuel*100,3))+'%')
print("rendement annuel moyen Benchmark "+str(round(rendement_annuel_bench*100,3))+'%')
```

```
rendement journalier moyen Portefeuille 0.063%
rendement journalier moyen Benchmark 0.033%
rendement annuel moyen Portefeuille 17.21%
rendement annuel moyen Benchmark 8.669%
```



Volatilité

In [ ]:
```
std_journalier = yield_own_index['Yield'].std()

std_annuel=std_journalier*253**(0.5)

std_journalier_bench=Rendement_US["US Equity"].std()
std_annuel_bench=std_journalier_bench*253**(0.5)

print("Ecart-type journalier own index: "+str(round(std_journalier*100,2))+ " %")
print("Ecart-type annualisé own index: "+str(round(std_annuel*100,2))+ " %")
print("Ecart-type journalier Benchmark: "+str(round(std_journalier_bench*100,2))+" %")
print("Ecart-type annualisé Benchmark: "+str(round(std_annuel_bench*100,2))+" %")
```

```
Ecart-type journalier own index: 2.19 %
Ecart-type annualisé own index: 34.87 %
Ecart-type journalier Benchmark: 1.23 %
Ecart-type annualisé Benchmark: 19.57 %
```

Ratio de Sharpe

In [ ]:
```
##Rendement Cash Annuel

Bench=pd.read_excel("DataProjets.xlsx",sheet_name="Benchmark")
Bench=Bench[["Unnamed: 0","cash"]]
Bench=pd.DataFrame.rename(Bench,columns={"Unnamed: 0":"Date","cash":"cash"})

own_index_cash=Bench['cash']
own_index_cash=pd.DataFrame(own_index_cash)

annual_yield_own_cash = (own_index_cash.diff(1)/own_index_cash).iloc[1:]

rendement_cash=annual_yield_own_cash['cash'].mean()
rendement_cash_journalier=(1+rendement_cash)**(1/253)-1
```

In [ ]:
```
#Calcul du ratio
Sharpe = (rendement_annuel-rendement_cash)/std_annuel
Sharpe2= (rendement_annuel_bench-rendement_cash)/std_annuel_bench
print("Ratio de Sharpe Portefeuille "+str(round(Sharpe,2)))
print("Ratio de Sharpe Benchmark "+str(round(Sharpe2,2)))
```

```
Ratio de Sharpe Portefeuille 0.52
Ratio de Sharpe Benchmark 0.48
```

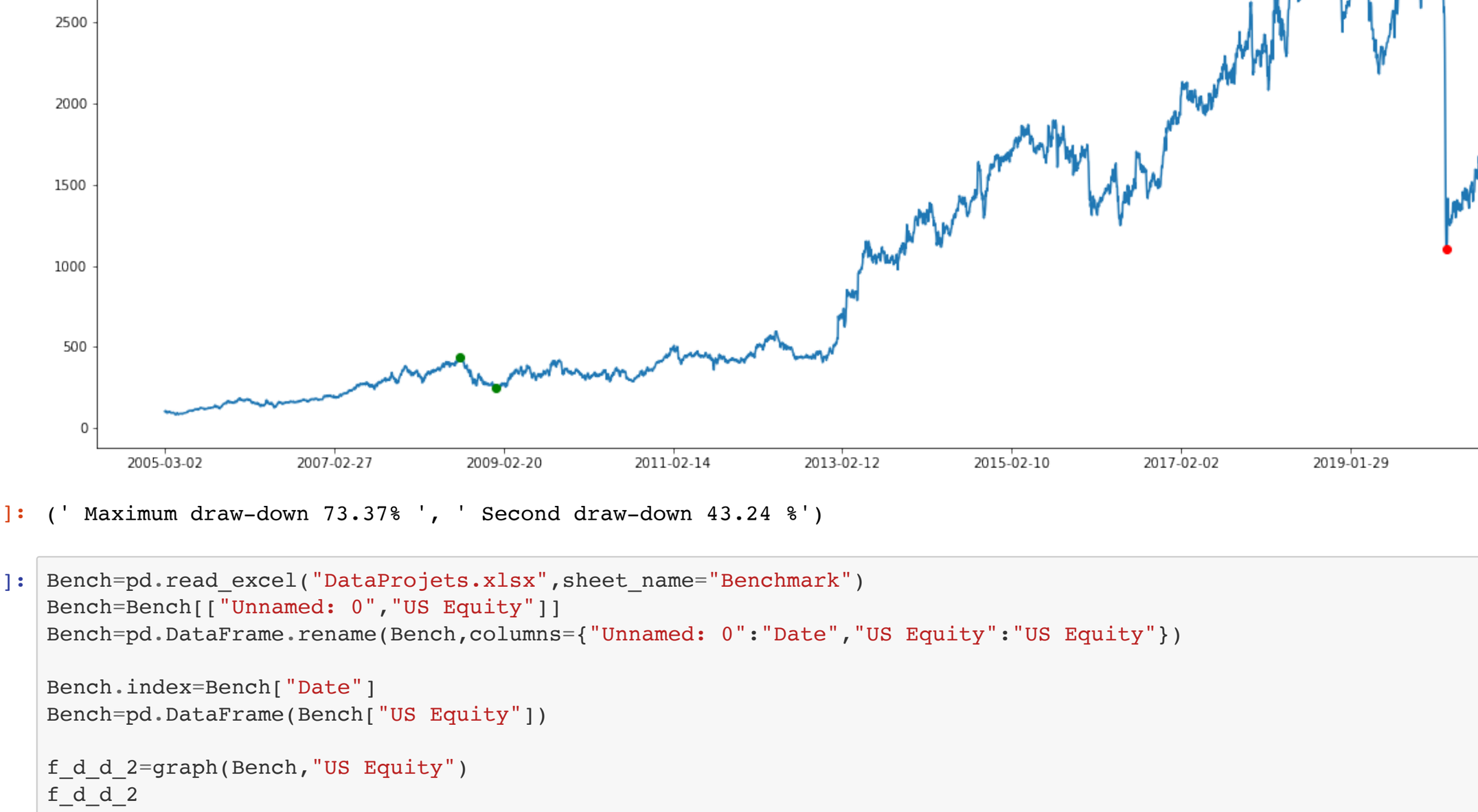
First draw-down (on price)

In [ ]:
```
def max_dd(own_index_price,x):
  n = len(own_index_price)
  dico={}
  liste_use=[]
  for i in range(n):
    price=own_index_price.iloc[i][str(x)]
    price_test=price
    j=i+1
    while price_test<=price and j<n:
      price_test=own_index_price.iloc[j][str(x)]
      j+=1
    mini_arg = i
    min_price = price
    for k in range(i,j):
      if min_price>own_index_price.iloc[k][str(x)]:
        min_price = own_index_price.iloc[k][str(x)]
        mini_arg = k
    if own_index_price.iloc[i][str(x)]>own_index_price.iloc[i-1][str(x)] and i>0 and (mini_arg not in liste_use):
      dico[i]=[(price-min_price)/price,i,mini_arg]
      liste_use.append(mini_arg)
  return(dico)
```

In [ ]:
```
def indices_d_d(own_index_price,x):
  dico=max_dd(own_index_price,x)
  m1,m2=0,0
  k1,k2=0,0
  for i in dico.keys():
    if dico[i][0]>m1:
      try:
        m2=dico[k1][0]
        k2=dico[k1][1]
      except:
        pass
      m1=dico[i][0]
      k1=i
    elif dico[i][0]>m2 and dico[i][0]<m1:
      m2=dico[i][0]
      k2=i
  i1,j1=dico[k1][1],dico[k1][2]
  i2,j2=dico[k2][1],dico[k2][2]
  return((i1,j1),(i2,j2))
```

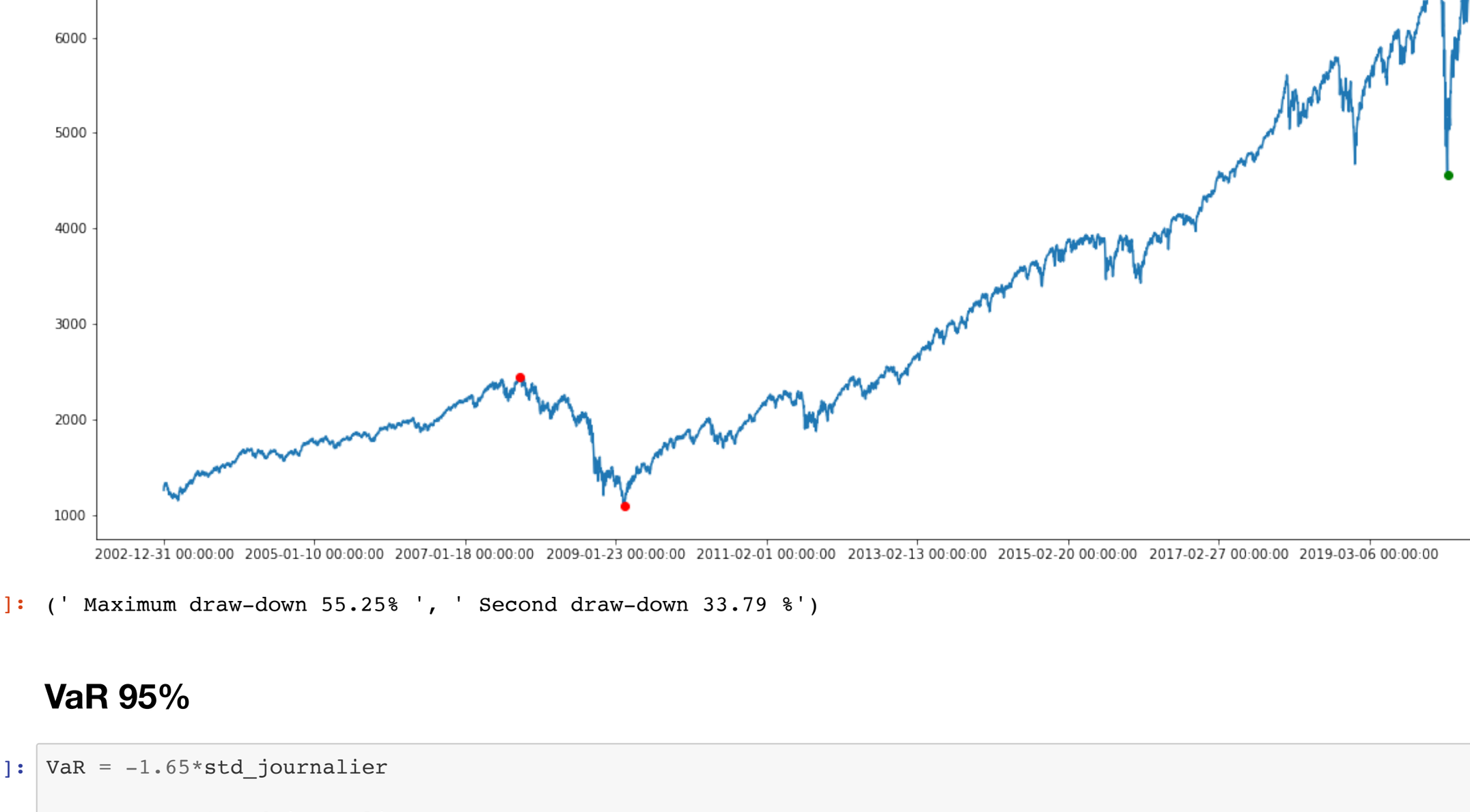
In [ ]:
```
def graph(own_index_price,x):
  ((i1,j1),(i2,j2))=indices_d_d(own_index_price,x)
  figure = plt.figure(figsize=(20, 10))
  xs=list(np.array(own_index_price))
  n=len(own_index_price)
  labels = [own_index_price.index[i] for i in range(0,n)]
  plt.plot(xs)
  plt.plot([i1, j1], [xs[i1], xs[j1]], 'o', color='Red')
  plt.plot([i2, j2], [xs[i2], xs[j2]], 'o', color='Green')
  plt.xticks([i for i in range(0,n) if i%500==0],[labels[i] for i in range(0,n) if i%500==0])
  plt.show()
  return(" Maximum draw-down "+str(round(((xs[i1]-xs[j1])[0]/(xs[i1])[0])*100,2))+ "% "," Second draw-down "+str(round
(((xs[i2]-xs[j2])[0]/(xs[i2])[0])*100,2))+ " %")
```

In [ ]:
```
f_d_d=graph(own_index_price,"Price")
f_d_d
```



Out[ ]: (' Maximum draw-down 73.37% ', ' Second draw-down 43.24 %')

In [ ]:
```
Bench=pd.read_excel("DataProjets.xlsx",sheet_name="Benchmark")
Bench=Bench[["Unnamed: 0","US Equity"]]
Bench=pd.DataFrame.rename(Bench,columns={"Unnamed: 0":"Date","US Equity":"US Equity"})

Bench.index=Bench["Date"]
Bench=pd.DataFrame(Bench["US Equity"])

f_d_d_2=graph(Bench,"US Equity")
f_d_d_2
```



Out[ ]: (' Maximum draw-down 55.25% ', ' Second draw-down 33.79 %')

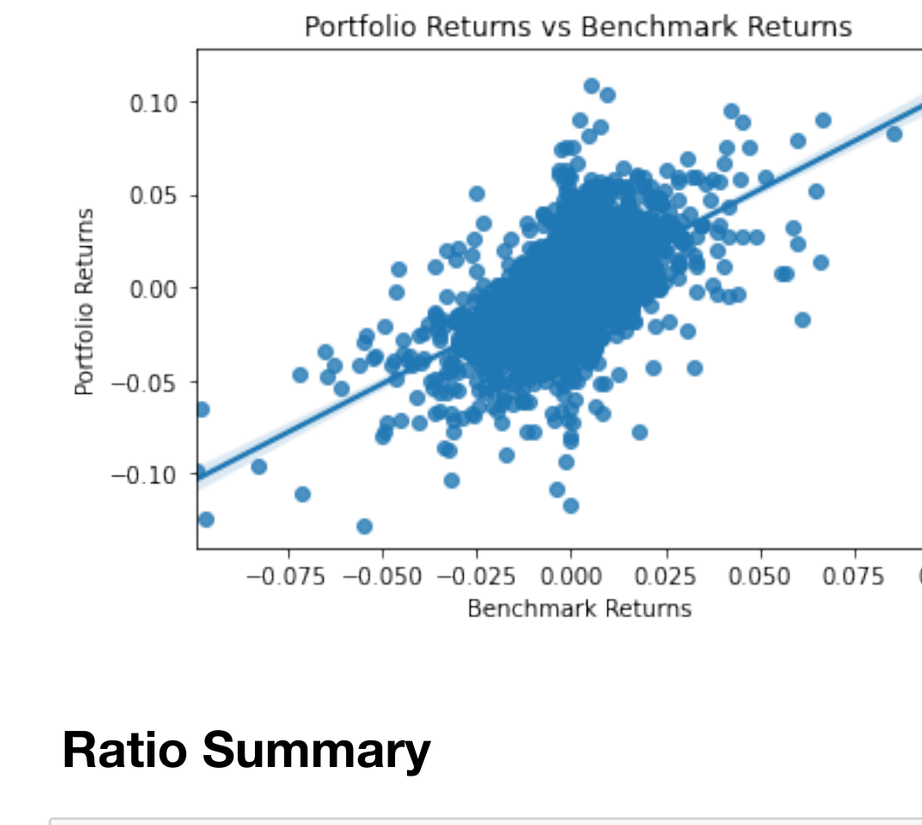
VaR 95%

In [ ]:
```
VaR = -1.65*std_journalier

VaR99 = -2.33*std_journalier

VaR_bench=-1.65*std_journalier_bench

VaR_bench99=-2.33*std_journalier_bench
```

Beta

In [ ]:
```
import seaborn as sns
from scipy import stats

def beta(res_l):
  sns.regplot(res_l["US Equity"],res_l["Yield"])
  plt.xlabel("Benchmark Returns")
  plt.ylabel("Portfolio Returns")
  plt.title("Portfolio Returns vs Benchmark Returns")
  plt.show()
  (beta, alpha) = stats.linregress(res_l["US Equity"],res_l["Yield"])[0:2]
  return(round(beta, 3))

beta_p=beta(Tableau_index_bench)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyw
ord args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



Ratio Summary

In [ ]:
```
Ratio=pd.DataFrame({'Rendement journalier': [str(round(rendement_journalier*100,2))+" %",str(round(rendement_journalie
r_bench*100,2))+" %"],'Rendement annuel':[str(round(rendement_annuel*100,2))+'%',str(round(rendement_annuel_bench*100,
2))+" %"], 'Volatilité journalière':[str(round(std_journalier*100,2))+ " %",str(round(std_journalier_bench*100,2))+ "
%"],'Volatilité annuelle':[str(round(std_annuel*100,2))+ " %",str(round(std_annuel_bench*100,2))+ "%"],"Ratio de Sharp
e":[round(Sharpe,2),round(Sharpe2,2)],"VaR 95%":[str(round(VaR,2)),str(round(VaR_bench,2))],"VaR 99%":[str(round(VaR99
,2)),str(round(VaR_bench99,2))], "First draw-down": [float(f_d_d[0][19:23]),float(f_d_d_2[0][19:23])], "Second draw-do
wn": [float(f_d_d[1][18:23]),float(f_d_d_2[1][18:23])] ,"β":[beta_p,"-"]})
Ratio=Ratio.transpose()
Ratio=Ratio.rename(columns={0:"Own Index",1:"US Equity"})
Ratio
```

Out[ ]:

	Own Index	US Equity
Rendement journalier	0.06 %	0.03 %
Rendement annuel	17.21%	8.67 %
Volatilité journalière	2.19 %	1.23 %
Volatilité annuelle	34.87 %	19.57%
Ratio de Sharpe	0.52	0.48
VaR 95%	-0.04	-0.02
VaR 99%	-0.05	-0.03
First draw-down	73.3	55.2
Second draw-down	43.24	33.79
β	1.044	-

Analyse Sectorielle (À lancer quand c'est utile) ¶

In [ ]:
```
Sector=pd.read_excel("DataProjets.xlsx",sheet_name="Sector")
dic_s_to_t={}
for i in range(len(mapping)):
  l = mapping.iloc[i]
  dic_s_to_t[l['Sedol']]=l['Tickers']

dic_s_to_t['Unnamed: 0']='Date' #No name for this column in the excel file

Sector=Sector.rename(columns=dic_s_to_t)

Sector
```

In [ ]:
```
dic_sector={4: "CONSUMER DISCRETIONARY", 5: "CONSUMER STAPLES", 6: "ENERGY", 7: "FINANCIALS", 8: "HEALTH CARE", 9: "IN
DUSTRIALS", 10: "INFORMATION TECHNOLOGY", 11: "MATERIALS", 14 : "UTILITIES"}
```

In [ ]:
```
def res_par_sect(num_du_secteur):
  index_strat = 0
  res = {}
  for i in range(len(data_history)-1):
    count=0
    coef=0
    for column in MarketCaps.columns:
      if column!="Date":
        if Sector.at[index_strat,column] == num_du_secteur:
          coef+=MarketCaps.at[index_strat,column]
          count+=data_history.at[i,column] * MarketCaps.at[index_strat,column]
    res[data_history.at[i,'Date']]=count/coef
    if data_history.at[i+1,'Date'][5:7]!=data_history.at[i,'Date'][5:7]:
      if index_strat!=216:
        index_strat+=1


  return res
```

In [ ]:
```
def affiche():
  dataf = pd.DataFrame.from_dict(res_par_sect(14), orient='index', columns=[14])

  for num in range(4,12):
    res = res_par_sect(num)
    current=pd.DataFrame.from_dict(res,orient='index',columns=[num])

    dataf = dataf.join(current)
    print(num, " fait!!!!!")

  return dataf
```

Calcul du Beta

In [ ]:
```
benchmark_sheet=pd.read_excel("DataProjets.xlsx",sheet_name="Benchmark")
res=yield_own_index.join(benchmark_sheet)
```

In [ ]:
```
import seaborn as sns
from scipy import stats

def beta(res_1):
  sns.regplot(res_1["US Equity"],res_1["Price"])
  plt.xlabel("Benchmark Returns")
  plt.ylabel("Portfolio Returns")
  plt.title("Portfolio Returns vs Benchmark Returns")
  plt.show()
  (beta, alpha) = stats.linregress(res_1["US Equity"],res_1["Price"])[0:2]
  return("The portfolio beta is", round(beta, 4))

beta_p=beta(Tableau_index_bench)
print(beta_p)
```

Par Secteur

In [ ]:
```
Tableau_Secteurs=pd.read_csv('tableau_secteurs_renormalisé.csv')
Tableau_Secteurs = pd.DataFrame.rename(Tableau_Secteurs,columns={"Unnamed: 0": "Date"})

column_of_dates = Tableau_Secteurs["Date"]

Tableau_Secteurs = pd.DataFrame((Tableau_Secteurs[str(i)].diff()/Tableau_Secteurs[str(i)]).iloc[:] for i in [14,4,5,6,
7,8,9,10,11])
Tableau_Secteurs=Tableau_Secteurs.transpose()

Tableau_Secteurs.index = column_of_dates

Tableau_Secteurs = Tableau_Secteurs.dropna()
Tableau_Secteurs_Cash = Tableau_Secteurs.join(Rendement_US)

Tableau_Secteurs_Cash = Tableau_Secteurs_Cash.dropna()
Tableau_Secteurs_Cash
```

In [ ]:
```
b={}
for i in [4,5,6,7,8,9,10,11]:
  figure = plt.figure(figsize=(20, 10))
  sns.regplot(Tableau_Secteurs_Cash["US Equity"],Tableau_Secteurs_Cash[str(i)],ax=figure.add_subplot(int(str(33)+str(i
-3))))
  plt.xlabel("Benchmark Returns")
  plt.ylabel(dic_sector[i] +" Returns")
  (beta, alpha) = stats.linregress(Tableau_Secteurs_Cash["US Equity"],Tableau_Secteurs_Cash[str(i)])[0:2]
  b[dic_sector[i]]=round(beta,2)
```

In [ ]:
```
b["PORTFOLIO"]=0.98
betas=pd.DataFrame(b,index=["beta"]).transpose()
betas
```

Key Ratio par Secteur

In [ ]:
```
def histogramme_par_secteur():
  count = 0
  for i in [14,4,5,6,7,8,9,10,11]:
    count += 1
    current_prices = pd.DataFrame(Tableau_Secteurs_Cash[str(i)])
    list_rendement=[x[0] for x in current_prices.values]

    figure = plt.figure(figsize=(20, 25))
    ax1 = figure.add_subplot(int("33"+str(count)))
    plt.hist(list_rendement, bins=150)


#histogramme_par_secteur()

dic_means = {}
dic_rendements = {}
dic_volatilites = {}
dic_var_1 = {}
dic_var_2 = {}
dic_sharpe = {}
dic_v = {}

for i in [14,4,5,6,7,8,9,10,11]:
  current_prices = pd.DataFrame(Tableau_Secteurs_Cash[str(i)])
  mean = current_prices.mean()

  volatilite = current_prices.std()*(253**0.5)
  rendement = (1+mean)**253 -1
  sharpe = (rendement-rendement_cash)/volatilite
  std_journalier = current_prices.std()

  dic_means[i] = mean
  dic_rendements[i] = rendement
  dic_volatilites[i] = volatilite
  dic_var_1[i] = -1.65*std_journalier
  dic_var_2[i] = -2.33*std_journalier
  dic_sharpe[i] = sharpe
  dic_v[i] = std_journalier


df_means = pd.DataFrame(np.diag(pd.DataFrame(dic_means.values())))
df_means.index = [dic_sector[i] for i in [14, 4, 5, 6, 7, 8, 9, 10, 11]]
df_means = pd.DataFrame.rename(df_means,columns={0: "Rendements Journaliers (%)"})
df_means = df_means.round(5)*100

df_rendements = pd.DataFrame(np.diag(pd.DataFrame(dic_rendements.values())))
df_rendements.index = [dic_sector[i] for i in [14, 4, 5, 6, 7, 8, 9, 10, 11]]
df_rendements = pd.DataFrame.rename(df_rendements,columns={0: "Rendements Annuels (%)"})
df_rendements = df_rendements.round(3)*100

df_v = pd.DataFrame(np.diag(pd.DataFrame(dic_v.values())))
df_v.index = [dic_sector[i] for i in [14, 4, 5, 6, 7, 8, 9, 10, 11]]
df_v = pd.DataFrame.rename(df_v,columns={0: "Volatilités Journalières (%)"})
df_v = df_v.round(3)*100

df_volatilites = pd.DataFrame(np.diag(pd.DataFrame(dic_volatilites.values())))
df_volatilites.index = [dic_sector[i] for i in [14, 4, 5, 6, 7, 8, 9, 10, 11]]
df_volatilites = pd.DataFrame.rename(df_volatilites,columns={0: "Volatilités Annuelles (%)"})
df_volatilites = df_volatilites.round(3)*100

df_sharpe = pd.DataFrame(np.diag(pd.DataFrame(dic_sharpe.values())))
df_sharpe.index = [dic_sector[i] for i in [14, 4, 5, 6, 7, 8, 9, 10, 11]]
df_sharpe = pd.DataFrame.rename(df_sharpe,columns={0: "Sharpe Ratio"})
df_sharpe = df_sharpe.round(2)

df_var_1 = pd.DataFrame(np.diag(pd.DataFrame(dic_var_1.values())))
df_var_1.index = [dic_sector[i] for i in [14, 4, 5, 6, 7, 8, 9, 10, 11]]
df_var_1 = pd.DataFrame.rename(df_var_1,columns={0: "VaR 95%"})
df_var_1 = df_var_1.round(3)*100

df_var_2 = pd.DataFrame(np.diag(pd.DataFrame(dic_var_2.values())))
df_var_2.index = [dic_sector[i] for i in [14, 4, 5, 6, 7, 8, 9, 10, 11]]
df_var_2 = pd.DataFrame.rename(df_var_2,columns={0: "VaR 99%"})
df_var_2 = df_var_2.round(3)*100

detailed_analysis = df_means.join(df_rendements.join(df_v.join(df_volatilites.join(df_sharpe.join(df_var_1.join(df_var
_2))))))

detailed_analysis
```