In [1]:

```
import pandas as pd
import numpy as np
from gurobipy import *
import math
import os
import matplotlib.pyplot as plt
```

In [2]:

```
table = pd.read_csv("C:/Users/NGDRS-1/Downloads/Merged.csv", encoding = "ISO-8859-1", engin
table['Date'] = table.Date.apply(lambda x: pd.to_datetime(x).strftime('%d/%m/%Y'))
#table = table.iloc[::-1]
table = table.set_index('Date')
l = list(a for a in range(99))
data = table.iloc[:,l]
table = data.loc['22/11/2018':'20/11/2020']
```

In [3]:

```
returns_daily = table.pct_change()
for column in returns_daily:
    returns_daily[column] = returns_daily[column].mask(returns_daily[column]<-0.475, np.nan
avg = returns_daily.mean() * 250/4
cov_daily = returns_daily.cov()
cov = cov_daily * 250/4
#std_daily = returns_daily.std()
#std = std_daily * math.sqrt(125/2)
```

In [4]:

```
cov
```

Out[4]:

	Bajaj Finserv Limited (BAJAJFINSV.BO)	Reliance Industries Limited (RELIANCE.BO)	Bajaj Finance Limited (BAJFINANCE.BO)	State Bank of India (SBIN.BO)	IndusIn l (INDUSINDE
Bajaj Finserv Limited (BAJAJFINSV.BO)	0.049448	0.017487	0.047122	0.027873	0.
Reliance Industries Limited (RELIANCE.BO)	0.017487	0.035971	0.017823	0.016096	0.
Bajaj Finance Limited (BAJFINANCE.BO)	0.047122	0.017823	0.060485	0.029915	0.
State Bank of India (SBIN.BO)	0.027873	0.016096	0.029915	0.043528	0.
IndusInd Bank Limited (INDUSINDBK.BO)	0.040979	0.017055	0.043965	0.036115	0.
...	...	...	...	...	
Decred INR (DCR-INR)	-0.004020	0.002368	-0.007066	0.004837	-0.
district0x INR (DNT-INR	0.000331	0.004224	0.005549	0.011940	0.
Golem INR (GNT-INR)	0.002368	0.002347	0.004004	0.004226	0.
AdEx INR (ADX-INR)	0.007291	0.017398	0.003726	0.004757	0.
Zcoin INR (XZC-INR)	0.003318	0.004940	0.001421	0.004279	0.

99 rows × 99 columns

In [5]:

```
model = Model('min_risk')
```

```
Using license file C:\Users\NGDRS-1\gurobi.lic
Academic license - for non-commercial use only - expires 2021-01-24
```

In [6]:

```
tickers = table.columns
variables = pd.Series(model.addVars(tickers),index=tickers)
```

In [7]:

```
port_risk = cov.dot(variables).dot(variables)
```

In [8]:

```
model.setObjective(port_risk,GRB.MINIMIZE)
```

In [9]:

```
model.addConstr(variables.sum() == 1,'weights')
model.update()
```

In [10]:

```
model.setParam('OutputFlag',0)
model.update()
```

In [11]:

```
model.optimize()
```

In [12]:

```
n = 0
weights = {}
for v in variables:
    weights.update({tickers[n]:v.x})
    n = n + 1
weights = pd.DataFrame([weights])
weights = weights.transpose()
weights.columns = ['Weights']

print('\nMin Risk, Optimal Weights Per Stock')
print(weights['Weights'])
```

```
Min Risk, Optimal Weights Per Stock
Bajaj Finserv Limited (BAJAJFINSV.BO)        7.353054e-09
Reliance Industries Limited (RELIANCE.BO)    9.099981e-03
Bajaj Finance Limited (BAJFINANCE.BO)        4.829999e-09
State Bank of India (SBIN.BO)                1.356169e-02
IndusInd Bank Limited (INDUSINDBK.BO)        2.126749e-09
                                                 ...
Decred INR (DCR-INR)                         1.719720e-09
district0x INR (DNT-INR                      1.824351e-09
Golem INR (GNT-INR)                          8.306592e-09
AdEx INR (ADX-INR)                           1.912537e-09
Zcoin INR (XZC-INR)                          2.082247e-09
Name: Weights, Length: 99, dtype: float64
```

In [13]:

```
print('\nMinimized Portfolio Variance : '+str(port_risk.getValue()))
```

```
Minimized Portfolio Variance : 0.001657278621745999
```

In [14]:

```
min_vol = math.sqrt(port_risk.getValue())
print('Volatility : '+str(min_vol))
```

```
Volatility : 0.04070968707501937
```

In [15]:

```
port_return = avg.dot(variables)
Rmin = port_return.getValue()
print('Expected Return (Rmin) : '+str(Rmin))
```

```
Expected Return (Rmin) : 0.015437707935745977
```

In [16]:

```
Rmax = avg.max()
```

In [17]:

```
target = model.addConstr(port_return == Rmin, 'target')
```

In [18]:

```
eff = {}
iterations = 50
diff = (Rmax-Rmin)/(iterations-1)
Rrange = np.arange(Rmin,Rmax+diff,diff)
for r in Rrange:
    target.rhs = r
    model.optimize()
    temp = math.sqrt(port_risk.getValue())
    eff.update({temp:r})
```

In [19]:

```
frontier = pd.DataFrame([eff]).transpose()
frontier.columns = ['Returns']
frontier['Risk'] = frontier.index
frontier = frontier.reset_index(drop=True)
```

In [20]:

```
print('\nEfficient Frontier')
print(frontier)
```

```
Efficient Frontier
     Returns      Risk
0   0.015438  0.040710
1   0.031156  0.041618
2   0.046874  0.044251
3   0.062592  0.048452
4   0.078310  0.053930
5   0.094029  0.060408
6   0.109747  0.067627
7   0.125465  0.075402
8   0.141183  0.083588
9   0.156901  0.092077
10  0.172619  0.100796
11  0.188338  0.109697
12  0.204056  0.118740
13  0.219774  0.127895
14  0.235492  0.137139
15  0.251210  0.146455
16  0.266928  0.155905
17  0.282647  0.165723
18  0.298365  0.175876
19  0.314083  0.186316
20  0.329801  0.197029
21  0.345519  0.208001
22  0.361237  0.219193
23  0.376956  0.230575
24  0.392674  0.242119
25  0.408392  0.253802
26  0.424110  0.265609
27  0.439828  0.277540
28  0.455546  0.289746
29  0.471265  0.302274
30  0.486983  0.315086
31  0.502701  0.328147
32  0.518419  0.341428
33  0.534137  0.354906
34  0.549855  0.368557
35  0.565574  0.382381
36  0.581292  0.396625
37  0.597010  0.411338
38  0.612728  0.426492
39  0.628446  0.442054
40  0.644164  0.458023
41  0.659883  0.474364
42  0.675601  0.491038
43  0.691319  0.508017
44  0.707037  0.525371
45  0.722755  0.543156
46  0.738473  0.562992
47  0.754192  0.588848
48  0.769910  0.620295
49  0.785628  0.656531
```

In [21]:

```
frontier['Sharpe'] = frontier['Returns']/frontier['Risk']
idx = frontier['Sharpe'].max()
sharpeMax = frontier.loc[frontier['Sharpe'] == idx]
sharpeMax = sharpeMax.reset_index(drop=True)
```

In [22]:

```
target.rhs = sharpeMax['Returns'][0]
model.optimize()
n = 0
sharpe_weights = {}
for v in variables:
    sharpe_weights.update({tickers[n]:v.x})
    n = n + 1
sharpe_weights = pd.DataFrame([sharpe_weights])
sharpe_weights = sharpe_weights.transpose()
sharpe_weights.columns = ['Weights']
```

In [23]:

```
print('\nMaximum Sharpe Ratio')
print(sharpeMax)
print(sharpe_weights)
```

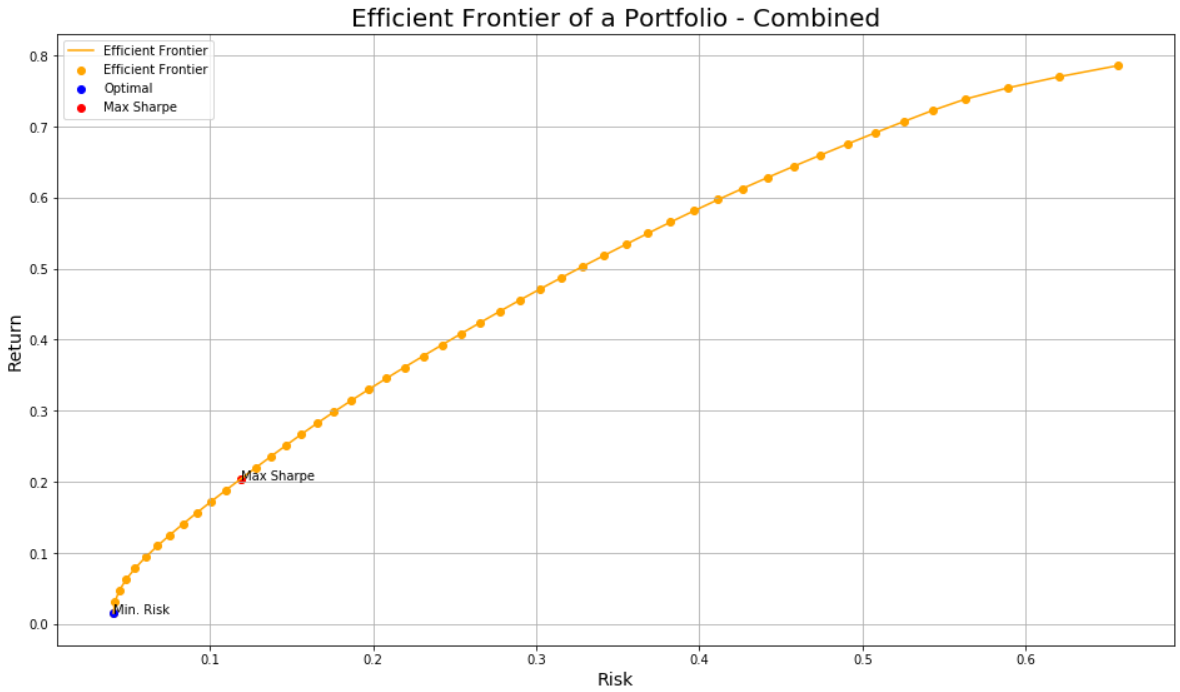
```
Maximum Sharpe Ratio
    Returns     Risk    Sharpe
0  0.204056  0.11874  1.718508
                                                 Weights
Bajaj Finserv Limited (BAJAJFINSV.BO)       1.285341e-10
Reliance Industries Limited (RELIANCE.BO)   8.671169e-09
Bajaj Finance Limited (BAJFINANCE.BO)       6.761822e-02
State Bank of India (SBIN.BO)               4.025741e-11
IndusInd Bank Limited (INDUSINDBK.BO)       2.642307e-11
...                                                  ...
Decred INR (DCR-INR)                        1.596528e-11
district0x INR (DNT-INR                     4.817486e-09
Golem INR (GNT-INR)                         4.412741e-11
AdEx INR (ADX-INR)                          1.309661e-02
Zcoin INR (XZC-INR)                         1.161460e-11

[99 rows x 1 columns]
```

In [25]:

```
fig, ax = plt.subplots(nrows=1,ncols=1)
fig.set_size_inches(16,9)
ax.set_title('Efficient Frontier of a Portfolio - Combined',fontsize=20)
ax.set_xlabel('Risk',fontsize=14)
ax.set_ylabel('Return',fontsize=14)

ax.scatter(x=frontier['Risk'],y=frontier['Returns'],color='orange',label='Efficient Frontie
ax.plot()#x=frontier['Risk'],y=frontier['Returns'],color='orange')
temp = pd.DataFrame([eff]).transpose()
temp.columns = ['Efficient Frontier']
temp.plot(color='orange',label='Efficient Frontier',ax=ax)

#ax.scatter(x=std,y=avg,color='green',label='Stocks')
i = 0
#for stock in tickers:
#    ax.annotate(stock,(std[i],avg[i]))
#    i = i + 1

ax.scatter(x=min_vol,y=Rmin,color='blue',label='Optimal')
ax.annotate('Min. Risk',(min_vol,Rmin))

ax.scatter(x=sharpeMax['Risk'],y=sharpeMax['Returns'],color='red',label='Max Sharpe')
ax.annotate('Max Sharpe',(sharpeMax['Risk'],sharpeMax['Returns']))

ax.grid()
ax.legend(loc='upper left')
```

Out[25]:

```
<matplotlib.legend.Legend at 0x28e34e5da88>
```



In [26]:

```
# Calculate mean returns for each stock
avg_rets = returns_daily.mean()

# Calculate mean returns for portfolio overall,
# using dot product to
# normalize individual means against investment weights
 # https://en.wikipedia.org/wiki/Dot_product#:~:targetText=In%20mathematics%2C%20the%20dot%
port_mean = avg_rets.dot(sharpe_weights)

# Calculate portfolio standard deviation
port_stdev = np.sqrt(sharpe_weights.T.dot(cov).dot(sharpe_weights))
```

In [27]:

```
initial_investment = 10000
# Calculate mean of investment
mean_investment = (1+port_mean) * initial_investment

# Calculate standard deviation of investmnet
stdev_investment = initial_investment * port_stdev
```

In [28]:

```
# Select our confidence interval (I'll choose 95% here)
conf_level1 = 0.05

# Using SciPy ppf method to generate values for the
# inverse cumulative distribution function to a normal distribution
# Plugging in the mean, standard deviation of our portfolio
# as calculated above
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html
from scipy.stats import norm
cutoff1 = norm.ppf(conf_level1, mean_investment, stdev_investment)
cutoff1
```

Out[28]:

```
array([[8079.54878586]])
```

In [29]:

```
#Finally, we can calculate the VaR at our confidence interval
var_1d1 = initial_investment - cutoff1
var_1d1
#output
```

Out[29]:

```
array([[1920.45121414]])
```