TRAVEL STOCK PORTFOLIO OPTIMAZATION

Stock data

Use yfinance library to get the *actual stock data* from the 20 companies of travel&tourism industry in US stock market.

Introduction

After a COVID-19-induced two-year break, many experts believe the travel & tourism industry start recovering recently. Investors also begin to consider adding stocks of travel sectors into their portfolios and looking for a travel economy's bounce back. But facing the uncertainty of its future, how can investors optimize their asset combination - minimize risk to achieve a certain return?   By using the Markowitz portfolio model to achieve a travel stock portfolio optimization, our project is willing to answer the question and perform analysis to it.

Data Summary

We use yfinance library to get the latest 3 months of *actual stock data* from the 20 companies of travel industry. To get bigger picture of the whole industry, we choose multiple market leaders that cover different aspects of the travel industry.

20 Stocks:

1. **Booking Holdings (BKNG)** : Online travel portal
2. **Airbnb (ABNB)**: Online travel portal
3. **Expedia Group (EXPE)**: Online travel portal
4. **Trip.com Group Limited (TCOM)**: Online travel portal
5. **Tripadvisor Inc (TRIP)**: Online travel portal
6. **Southwest Airlines (NYSE:LUV)**: Airline company
7. **Boeing (NYSE:BA)**: Aircraft producer
8. **Delta Air Lines (DAL)**: Airline company

9. **Hyatt Hotels Corporation (H)**: Hotel company
10. **Marriott International (MAR)**:Hotel company
11. **Royal Caribbean Group (RCL)**: Cruise line company
12. **Norwegian Cruise Line Holdings Ltd (NCLH)**: Cruise line company
13. **Carnival Group （CCL）**： Cruise line company
14. **Avis Budget Group (CAR)**:Car rental company
15. **Hertz Global Holdings (HTZ)**: Car rental company
16. **MakeMyTrip Limited (MMYT)**: Online travel portal
17. **Travel + Leisure Co. (TNL)**: Club & resort company
18. **The Walt Disney Company(DIS)**: Theme park and hotel company
19. **American Airlines Group (AAL)**: Airline company
20. **Sabre Corporation (SABR)**: Travel technology

Model Formulation

Parameters

We use the Greek values that are traditional in finance:

- δ: n-element vector measuring the average returns for each stock
- σ: n x n matrix measuring the covariance among stocks

There is one additional parameter when solving the model parametrically:

- r: target return

Decision Variables

- $x \geq 0$: n-element vector where each element represents the fraction of the porfolio to invest in each stock

Objective Function

Minimize the total risk, a convex quadratic function:

$$\min x^t \cdot \sigma \cdot x$$

Constraints

Allocate the entire portfolio: the total investments should be 1.0 (100%), where e is a unit vector (all 1's):

$$e \cdot x = 1$$

Return: When we solve the model parametrically for different return values r, we add a constraint on the target return:

$$\delta \cdot x = r$$

Python Implementation

```
!pip install gurobipy
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: gurobipy in /usr/local/lib/python3.7/dist-packages (9.5.2)
```

```
!pip install yfinance
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: yfinance in /usr/local/lib/python3.7/dist-packages (0.1.75)
Requirement already satisfied: requests>=2.26 in /usr/local/lib/python3.7/dist-packages (from yfinance) (2.28.1)
Requirement already satisfied: appdirs>=1.4.4 in /usr/local/lib/python3.7/dist-packages (from yfinance) (1.4.4)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.7/dist-packages (from yfinance) (0.0.11)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.7/dist-packages (from yfinance) (1.3.5)
Requirement already satisfied: lxml>=4.5.1 in /usr/local/lib/python3.7/dist-packages (from yfinance) (4.9.1)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.7/dist-packages (from yfinance) (1.21.6)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.24.0->yfinance) (2022.4)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.24.0->yfinance) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas>=0.24.0->yfinance) (1
Requirement already satisfied: charset-normalizer<3,>=2 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (2.1.1)
```

```
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (2022.9.24)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (1.24.3)
```

```
import yfinance as yf

stocks = ['BKNG', 'LUV', 'ABNB', 'MAR', 'DIS', 'RCL', 'AAL', 'EXPE', 'BA', 'NCLH', 'CCL', 'TRIP', 'HTZ', 'DAL', 'CAR',
          'MMYT', 'TNL', 'H', 'TCOM', 'SABR']

data = yf.download(stocks, period='3mo')
```

```
[*********************100%***********************]  20 of 20 completed
```

```
print(type(data))
print(data)
```

```
<class 'pandas.core.frame.DataFrame'>
                          Adj Close                                       \
                                AAL        ABNB          BA         BKNG
Date
2022-07-06 00:00:00-04:00    13.890   92.879997  136.309998  1750.859985
2022-07-07 00:00:00-04:00    14.100   97.500000  139.970001  1785.920044
2022-07-08 00:00:00-04:00    13.970   97.349998  139.070007  1769.479980
2022-07-11 00:00:00-04:00    13.430   95.099998  136.990005  1724.550049
2022-07-12 00:00:00-04:00    14.770   96.550003  147.149994  1738.250000
...                             ...         ...         ...          ...
2022-09-30 00:00:00-04:00    12.040  105.040001  121.080002  1643.209961
2022-10-03 00:00:00-04:00    11.920  105.000000  126.050003  1678.930054
2022-10-04 00:00:00-04:00    12.950  110.809998  133.509995  1759.040039
2022-10-05 00:00:00-04:00    12.870  111.760002  132.110001  1726.709961
2022-10-06 00:00:00-04:00    12.735  111.599998  132.309998  1709.880005


                                                                  \
                                  CAR   CCL        DAL        DIS
Date
2022-07-06 00:00:00-04:00  148.929993  8.74  29.530001  96.080002
```

```
2022-07-07 00:00:00-04:00  156.979996  9.33  29.910000   97.430000
2022-07-08 00:00:00-04:00  156.669998  9.01  29.719999   95.860001
2022-07-11 00:00:00-04:00  149.309998  8.49  29.290001   93.639999
2022-07-12 00:00:00-04:00  152.639999  9.13  31.090000   93.599998
...                               ...   ...        ...         ...
2022-09-30 00:00:00-04:00  148.460007  7.03  28.059999   94.330002
2022-10-03 00:00:00-04:00  157.869995  6.85  28.260000   97.129997
2022-10-04 00:00:00-04:00  167.899994  7.76  30.750000  101.440002
2022-10-05 00:00:00-04:00  173.809998  7.43  30.520000  100.800003
2022-10-06 00:00:00-04:00  177.919998  7.02  30.592501  100.040001

                                                        ...   Volume          \
                                 EXPE          H        ...      HTZ      LUV
Date                                                    ...
2022-07-06 00:00:00-04:00   93.870003  73.169998  ...  2806500  5000400
2022-07-07 00:00:00-04:00   96.699997  74.769997  ...  4980700  4127200
2022-07-08 00:00:00-04:00   94.870003  74.370003  ...  3320600  3820000
2022-07-11 00:00:00-04:00   91.160004  74.160004  ...  4901700  5360700
2022-07-12 00:00:00-04:00   92.029999  74.260002  ...  3488300  8130800
...                               ...        ...  ...      ...      ...
2022-09-30 00:00:00-04:00   93.690002  80.959999  ...  3097400  6891600
2022-10-03 00:00:00-04:00   94.400002  82.410004  ...  2739600  5734600
2022-10-04 00:00:00-04:00  102.099998  86.070000  ...  3333100  7459700
2022-10-05 00:00:00-04:00   98.989998  85.750000  ...  8803400  4884300
2022-10-06 00:00:00-04:00   95.819901  83.699997  ...  2924741  3629494

                                                                          \
                               MAR    MMYT      NCLH       RCL     SABR
Date
2022-07-06 00:00:00-04:00  1880400  151200  32183500   7569000  7124200
2022-07-07 00:00:00-04:00  2017100   91800  18191300   7575300  5009400
2022-07-08 00:00:00-04:00  1631400  176900  16784100   5903100  3747300
2022-07-11 00:00:00-04:00  2161700  306200  14974800   5911200  5322500
2022-07-12 00:00:00-04:00  1691700  184200  17505200   7652800  6341600
...                            ...     ...       ...       ...      ...
2022-09-30 00:00:00-04:00  2100800  411800  52971100  20285000  5688400
2022-10-03 00:00:00-04:00  2107000  210200  33704600  10575700  5241000
2022-10-04 00:00:00-04:00  2128600  248600  42042800  14696000  6259200
```

Compute Greeks

Using the downloaded stock data, find the delta (return), sigma (covariance) and standard deviation values for stock prices:

```
import numpy as np

closes = np.transpose(np.array(data.Close)) # matrix of daily closing prices
absdiff = np.diff(closes)                    # change in closing price each day
reldiff = np.divide(absdiff, closes[:,:-1]) # relative change in daily closing price
delta = np.mean(reldiff, axis=1)             # mean price change
sigma = np.cov(reldiff)                      # covariance (standard deviations)
std = np.std(reldiff, axis=1)                # standard deviation
```

Minimize risk by solving QP model

```
import gurobipy as gp
from gurobipy import GRB
from math import sqrt

# Create an empty model
m = gp.Model('portfolio')

# Add matrix variable for the stocks
x = m.addMVar(len(stocks))

# Objective is to minimize risk (squared).  This is modeled using the
# covariance matrix, which measures the historical correlation between stocks
portfolio_risk = x @ sigma @ x
m.setObjective(portfolio_risk, GRB.MINIMIZE)

# Fix budget with a constraint
m.addConstr(x.sum() == 1, 'budget')

# Verify model formulation
m.write('portfolio_selection_optimization.lp')
```

```
# Optimize model to find the minimum risk portfolio
m.optimize()
```

```
Restricted license - for non-production use only - expires 2023-10-25
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (linux64)
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads
Optimize a model with 1 rows, 20 columns and 20 nonzeros
Model fingerprint: 0x41ac30cc
Model has 210 quadratic objective terms
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [0e+00, 0e+00]
  QObjective range [5e-04, 1e-02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve time: 0.02s
Presolved: 1 rows, 20 columns, 20 nonzeros
Presolved model has 210 quadratic objective terms
Ordering time: 0.00s

Barrier statistics:
 Free vars  : 19
 AA' NZ     : 1.900e+02
 Factor NZ  : 2.100e+02
 Factor Ops : 2.870e+03 (less than 1 second per iteration)
 Threads    : 1

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   2.28950169e+05 -2.28950169e+05  1.60e+04 6.60e-05  1.00e+06     0s
   1   7.12641867e+04 -7.12933776e+04  8.28e+02 3.42e-06  5.84e+04     0s
   2   3.49102801e+01 -7.99693148e+01  1.61e+01 6.65e-08  1.17e+03     0s
   3   1.04825545e-03 -4.52159310e+01  1.61e-05 6.65e-14  2.26e+00     0s
   4   1.04789053e-03 -4.55531964e-02  5.03e-10 2.06e-18  2.33e-03     0s
   5   1.00001429e-03 -3.80512677e-04  7.98e-12 1.73e-18  6.90e-05     0s
   6   5.63082484e-04 -1.11843599e-03  2.08e-17 1.39e-17  8.41e-05     0s
   7   4.56113353e-04  2.36259631e-04  6.94e-18 8.67e-18  1.10e-05     0s
   8   3.90550334e-04  3.14158798e-04  3.47e-17 1.02e-17  3.82e-06     0s
   9   3.68959897e-04  3.63930339e-04  6.94e-18 9.97e-18  2.51e-07     0s
```

```
   10   3.66496788e-04  3.66320959e-04  2.29e-16 4.37e-18  8.79e-09     0s
   11   3.66379067e-04  3.66372454e-04  1.41e-16 8.08e-18  3.31e-10     0s

Barrier solved model in 11 iterations and 0.06 seconds (0.00 work units)
Optimal objective 3.66379067e-04
```

Display minimum risk portfolio using Pandas

```
import pandas as pd
minrisk_volatility = sqrt(m.ObjVal)
minrisk_return = delta @ x.X
pd.DataFrame(data=np.append(x.X, [minrisk_volatility, minrisk_return]),
                          index=stocks + ['Volatility', 'Expected Return'],
                          columns=['Minimum Risk Portfolio'])
```

	Minimum Risk Portfolio
BKNG	7.013269e-08
LUV	8.510661e-08
ABNB	1.118918e-03
MAR	1.346897e-07
DIS	2.509288e-08
RCL	1.669345e-08
AAL	2.421677e-07
EXPE	2.765420e-01
BA	3.664324e-08
NCLH	9.532448e-05
CCL	2.063286e-02

Compute the efficient frontier

Solve the QP parametrically to find the lowest risk portfolio for different expected returns.

```
# Create an expression representing the expected return for the portfolio
portfolio_return = delta @ x
target = m.addConstr(portfolio_return == minrisk_return, 'target')

# Solve for efficient frontier by varying target return
frontier = np.empty((2,0))
for r in np.linspace(delta.min(), delta.max(), 25):
    target[0].rhs = r
    m.optimize()
    frontier = np.append(frontier, [[sqrt(m.ObjVal)],[r]], axis=1)
```

```
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (linux64)
```

```
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads
Optimize a model with 2 rows, 20 columns and 40 nonzeros
Model fingerprint: 0x58feabcc
Model has 210 quadratic objective terms
Coefficient statistics:
  Matrix range     [6e-05, 1e+00]
  Objective range  [0e+00, 0e+00]
  QObjective range [5e-04, 1e-02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e-03, 1e+00]
Presolve time: 0.02s
Presolved: 2 rows, 20 columns, 40 nonzeros
Presolved model has 210 quadratic objective terms
Ordering time: 0.00s

Barrier statistics:
 Free vars  : 19
 AA' NZ     : 2.100e+02
 Factor NZ  : 2.310e+02
 Factor Ops : 3.311e+03 (less than 1 second per iteration)
 Threads    : 1

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   2.28352980e+05 -2.28352980e+05  1.55e+04 9.66e-03  1.00e+06     0s
   1   3.94707900e+04 -3.94941896e+04  1.23e+03 7.69e-04  8.30e+04     0s
   2   1.02003901e+02 -1.36121182e+02  1.65e+01 1.03e-05  1.13e+03     0s
   3   3.48074367e-03 -3.46688377e+01  2.20e-02 1.37e-08  3.23e+00     0s
   4   2.32170232e-03 -1.58889067e+00  1.73e-03 1.08e-09  2.02e-01     0s
   5   2.04082179e-03 -3.62470073e-01  1.74e-04 1.09e-10  4.25e-02     0s
   6   2.04157261e-03  1.30122021e-03  1.80e-07 1.12e-13  6.21e-05     0s
   7   2.04157459e-03  2.04083423e-03  1.80e-10 1.12e-16  6.21e-08     0s
   8   2.04157459e-03  2.04157385e-03  1.80e-13 1.82e-12  6.21e-11     0s

Barrier solved model in 8 iterations and 0.07 seconds (0.00 work units)
Optimal objective 2.04157459e-03

Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (linux64)
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads
Optimize a model with 2 rows, 20 columns and 40 nonzeros
Model fingerprint: 0x4130d330
```

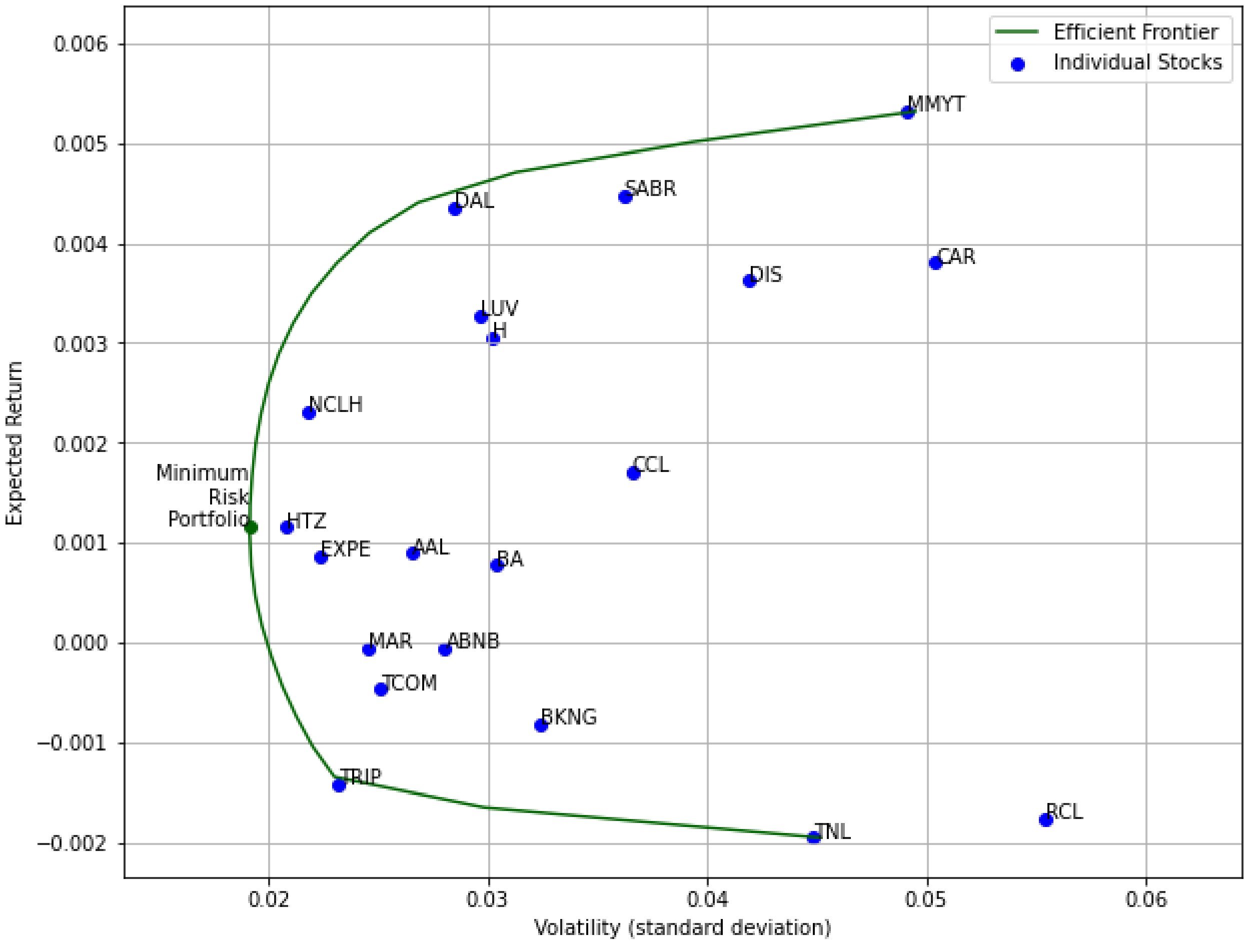
```
Model has 210 quadratic objective terms
Coefficient statistics:
  Matrix range     [6e-05, 1e+00]
  Objective range  [0e+00, 0e+00]
  QObjective range [5e-04, 1e-02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e-03, 1e+00]
Presolve time: 0.02s
Presolved: 2 rows, 20 columns, 40 nonzeros
Presolved model has 210 quadratic objective terms
Ordering time: 0.00s

Barrier statistics:
 Free vars  : 19
 AA' NZ     : 2.100e+02
 Factor NZ  : 2.310e+02
```

Plot results

Use the matplot library to plot the optimized solutions, along with the individual stocks:

```
import matplotlib.pyplot as plt
#plt.figure(figsize=(10,10))

fig, ax = plt.subplots(figsize=(10,8))

# Plot volatility versus expected return for individual stocks
ax.scatter(x=std, y=delta,
           color='Blue', label='Individual Stocks')
for i, stock in enumerate(stocks):
    ax.annotate(stock, (std[i], delta[i]))

# Plot volatility versus expected return for minimum risk portfolio
ax.scatter(x=minrisk_volatility, y=minrisk_return, color='DarkGreen')
ax.annotate('Minimum\nRisk\nPortfolio', (minrisk_volatility, minrisk_return),
            horizontalalignment='right')
```

```
# Plot efficient frontier
ax.plot(frontier[0], frontier[1], label='Efficient Frontier', color='DarkGreen')

# Format and display the final plot
ax.axis([frontier[0].min()*0.7, frontier[0].max()*1.3, delta.min()*1.2, delta.max()*1.2])
ax.set_xlabel('Volatility (standard deviation)')
ax.set_ylabel('Expected Return')
ax.legend()
ax.grid()
plt.show()
```



```
import pandas as pd
from statistics import mean
minrisk_volatility = sqrt(m.ObjVal)
average_return = mean(delta)
pd.DataFrame(data=np.append(x.X, [minrisk_volatility, average_return]),
                          index=stocks + ['Volatility', 'Expected Return'],
                          columns=['Average Return Portfolio'])
```

	Average Return Portfolio
BKNG	9.264283e-14
LUV	3.866901e-14
ABNB	1.090486e-13
MAR	1.091973e-13
DIS	9.022266e-14
[illegible]	[illegible]
DAL	1.580898e-13
CAR	2.340063e-13
MMYT	1.000000e+00
TNL	8.747396e-14
H	3.757630e-14
TCOM	9.883157e-14
SABR	1.936499e-13
Volatility	4.952134e-02
Expected Return	1.452806e-03

Summary

As we showed above, when we looking for a portfolio at a minimum risk, our expected return is around 0.0016, and our minimum variance is around 0.019. There are 3 stocks(TNL, MMYT, TRIP) in the curved line. Most of the stocks are in a safe region. 'EXPE', 'TRIP', 'HTZ', and 'H' are the ones recommended to buy the most. MMT is the aggressive one with high risk and high return; RCL is the one with the highest risk but low return, which is not recommended. When we set our target return to average return, the minimum varience increase to 0.049 and the expected return is 0.0015 and most of the money should buy 'MMT'. Depending on the instability of the whole stock market in the last 3 months, I think the travel stocks perform ok. We are looking for the travel sectors to come back be stronger.