Programm

Group Project1

Group No. 5178

Submitted by: Vikas Gautam

Mail-id: vikasgtm@gmail.com Date: 12/02/2023 Mobile:+91-8197885704

Asset Class: CryptoCurrency

Asset Chosen: Bitcoin, Ethereum, Tether, BNB

In [ ]:
```
# Importing scipy
import scipy
import pandas as pd
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
#!pip install pandoc
```

In [ ]:
```
#!pip install pandas.DataReader
import pandoc
```

In [4]:
```
import pandas_datareader.data as webreader
import matplotlib.pyplot as plt
from matplotlib.ticker import PercentFormatter
import matplotlib.ticker as mtick
from datetime import datetime
import matplotlib.dates as mdates
import numpy as np
```

In [5]:
```
#!pip install yfinance
from scipy.stats import kurtosis
```

As input for the analysis, I have used price data from Yahoo Finance for the period

starting from Jan 1, 2019 to the time of Feb 7,2023.

I have chosen Bitcoin (BTC-USD) and Ethereum's Ether (ETH-USD), Tether USD, and BNB USD

to represent the crypto assets for my portfolio.

Bitcoin USD 3 year historic price link: https://finance.yahoo.com/quote/BTC-USD/history?period1=1675209600&period2=1675987200&interval=1d&filter=history&frequency=1d&includeA

Ethereum USD 3 year historic price link: https://finance.yahoo.com/quote/ETH-USD/history?period1=1675209600&period2=1675987200&interval=1d&filter=history&frequency=1d&includeA https://finance.yahoo.com/quote/ETH-USD/history?period1=1675209600&period2=1675987200&interval=1d&filter=history&frequency=1d&includeA

Tether USD (USDT) 3 year historic price link: https://finance.yahoo.com/quote/USDT-USD/history?period1=1675209600&period2=1675987200&interval=1d&filter=history&frequency=1d&includeA

BNB USD 3 year historic price link: https://finance.yahoo.com/quote/BNB-USD/history?period1=1675209600&period2=1675987200&interval=1d&filter=history&frequency=1d&includeA

1. BITCOIN

In [6]:
```
import yfinance as yfin
BTC = yfin.download('BTC-USD', start='2019-1-1',end='2023-2-7')
```

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

In [7]:
```
BTC.head(5)
```

Out[7]:

Date	Open	High	Low	Close	Adj Close	Volume
2019-01-01	3746.713379	3850.913818	3707.231201	3843.520020	3843.520020	4324200990
2019-01-02	3849.216309	3947.981201	3817.409424	3943.409424	3943.409424	5244856836
2019-01-03	3931.048584	3935.685059	3826.222900	3836.741211	3836.741211	4530215219
2019-01-04	3832.040039	3865.934570	3783.853760	3857.717529	3857.717529	4847965467
2019-01-05	3851.973877	3904.903076	3836.900146	3845.194580	3845.194580	5137609824

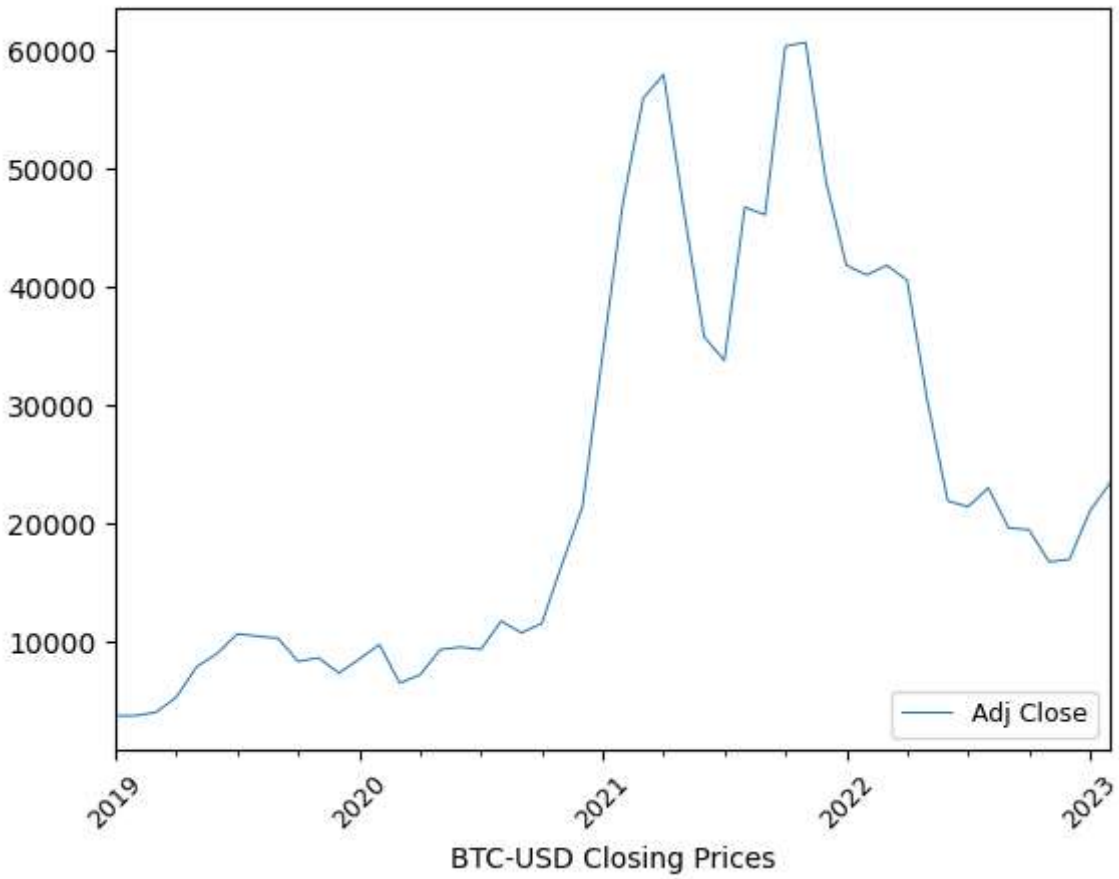
In [8]:
```
BTC.tail(5)
```

Out[8]:

Date	Open	High	Low	Close	Adj Close	Volume
2023-02-02	23720.824219	24167.210938	23468.595703	23471.871094	23471.871094	32066936882
2023-02-03	23469.412109	23678.103516	23279.955078	23449.322266	23449.322266	27083066007
2023-02-04	23446.320312	23556.949219	23291.794922	23331.847656	23331.847656	15639298538
2023-02-05	23332.248047	23423.435547	22841.759766	22955.666016	22955.666016	19564262605
2023-02-06	22954.021484	23119.279297	22692.025391	22760.109375	22760.109375	23825006542

In [9]:
```
#Converting daily return data to monthly data, required statistic: mean, median
data_btc_mean=BTC['Adj Close'].resample('m').mean()
data_btc_med=BTC['Adj Close'].resample('m').median()
```

In [10]:
```
ax11=data_btc_med.plot(linewidth=.7)
#ax11.xaxis.set_major_formatter(mdates.DateFormatter("%b %Y"))
plt.legend(loc='lower right', ncol=3, fontsize=9)
plt.xlabel('BTC-USD Closing Prices')
plt.xticks(rotation=45, fontsize=9)
plt.show( )
```



In [11]:
```
#high volatility of Bitcoin, particularly during 2021-2022, dropping further near 2023
```

In [12]:
```
#I created a dataframe with the daily prices. I use the 'Adj Close' column as
#it is a price adjusted for splits and dividends distribution. To extract daily return
#I calculate the percentage change of the daily prices using the pct_change() method.
datar_btc=pd.DataFrame()
```

```
returns_btc=BTC['Adj Close'].pct_change(1)
datar_btc['ticker%']=returns_btc
datar_btc.head(5)
```

Out[12]:

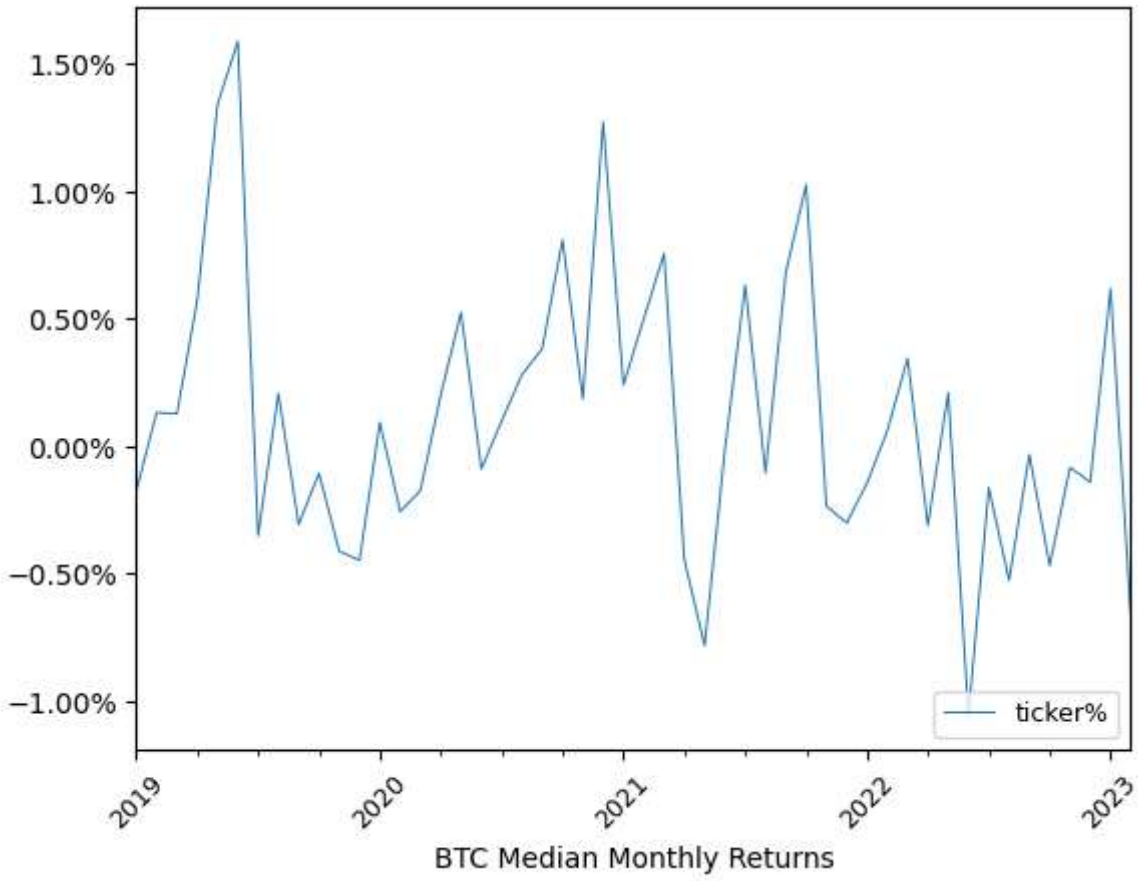
Date	ticker%
2019-01-01	NaN
2019-01-02	0.025989
2019-01-03	-0.027050
2019-01-04	0.005467
2019-01-05	-0.003246

In [13]:
```
#Converting daily return data to monthly data, required statistic: mean, median
datar_meanbtc=datar_btc.resample('m').mean()
datar_medbtc=datar_btc.resample('m').median()
datar_medbtc.head(5)
```

Out[13]:

Date	ticker%
2019-01-31	-0.001776
2019-02-28	0.001302
2019-03-31	0.001252
2019-04-30	0.005685
2019-05-31	0.013388

In [14]:
```
ax11=datar_medbtc.plot(linewidth=.7)
ax11.yaxis.set_major_formatter(mtick.PercentFormatter(1.0))
plt.legend(loc='lower right', ncol=3, fontsize=9)
plt.xlabel('BTC Median Monthly Returns')
plt.xticks(rotation=45, fontsize=9)
plt.show( )
```



In [15]:
```
## Dialy returns provide more data points for data analysis, however, annualised retur
## preferred by investors and give a better basis for the comparison of the assets.
## To annualise daily returns of the assets, I calculate mean returns and
## multiply them by 253 (average number of trading days a year).
## Common measure of return volatility, or risk, is the standard deviation. Standard d
##calculated as a square root of the variance and has an advantage of having the same
## Using standard deviation as a measure of risk, we can calculate risk-adjusted perfo
## for instance Sharpe ratio. Sharpe Ratio is calculated as annualised returns divided
```

In [16]:
```
#Asset return and volatility (risk) are major factors of the portfolio construction.
#Most investors want to maximise their returns while limiting the risk.
#BTC Arithmeti mean of % daily returns
btc_means=datar_btc.mean()
print ("BTC Average Annualised Returns (arithmetic mean) are:", btc_means*253*100)
#daily_btc_cov=[]
#BTC Co-variance
daily_btc_cov=round(datar_btc.cov(),3)
#print (daily_btc_cov.info())
annual_btc_cov=round(daily_btc_cov*253,3)
print("daily BTC covariance:",daily_btc_cov.iloc[0,0])
print("Annual BTC covarince:",annual_btc_cov.iloc[0,0])

#BTC standard Deviation
btc_stddev=datar_btc.std()*np.sqrt(253)
print('BTC Standard Deviation are: ', btc_stddev)

#BTC Variance
print ("BTC Variance:", (btc_stddev*btc_stddev))

#BTCSharpe Ratio
```

```
btc_sr=btc_means*253/btc_stddev
print ('BTC Sharpe Ratio: ',btc_sr)
```

```
BTC Average Annualised Returns (arithmetic mean) are: ticker%    47.791094
dtype: float64
daily BTC covariance: 0.001
Annual BTC covarince: 0.253
BTC Standard Deviation are:  ticker%    0.589818
dtype: float64
BTC Variance: ticker%    0.347885
dtype: float64
BTC Sharpe Ratio:  ticker%    0.810268
dtype: float64
```

In [17]:
```
#Appending lists with code,returns ans annual covarianceL this we will use to find
#respective weights of the assets for the optimal portfolio in terms of min volatilty,
#highest sharpe ratio, high expected retruns
summary_code=[]
summary_ann_returns=[]
summary_ann_cov=[]

summary_code.append("BTC-USD")
summary_ann_returns.append(btc_means*253)
summary_ann_cov.append(annual_btc_cov.iloc[0,0])
```

In [18]:
```
print (summary_code)
print (summary_ann_returns)
print (summary_ann_cov)
```

```
['BTC-USD']
[ticker%    0.477911
dtype: float64]
[0.253]
```

In [ ]:
```

```

2. ETHEREUM

In [19]:
```
#Ethereum
ETH = yfin.download('ETH-USD', start='2019-1-1',end='2023-2-7')
```

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

In [20]:
```
ETH.head(5)
```

Out[20]:

Date	Open	High	Low	Close	Adj Close	Volume
2019-01-01	133.418152	141.397507	132.650711	140.819412	140.819412	2258709868
2019-01-02	141.519516	156.929138	140.650955	155.047684	155.047684	3328240369
2019-01-03	155.196045	155.863052	147.198364	149.135010	149.135010	2676164880
2019-01-04	148.912888	156.878983	147.907104	154.581940	154.581940	3126192535
2019-01-05	154.337418	160.824890	154.337418	155.638596	155.638596	3338211928

In [21]:
```
ETH.tail(5)
```

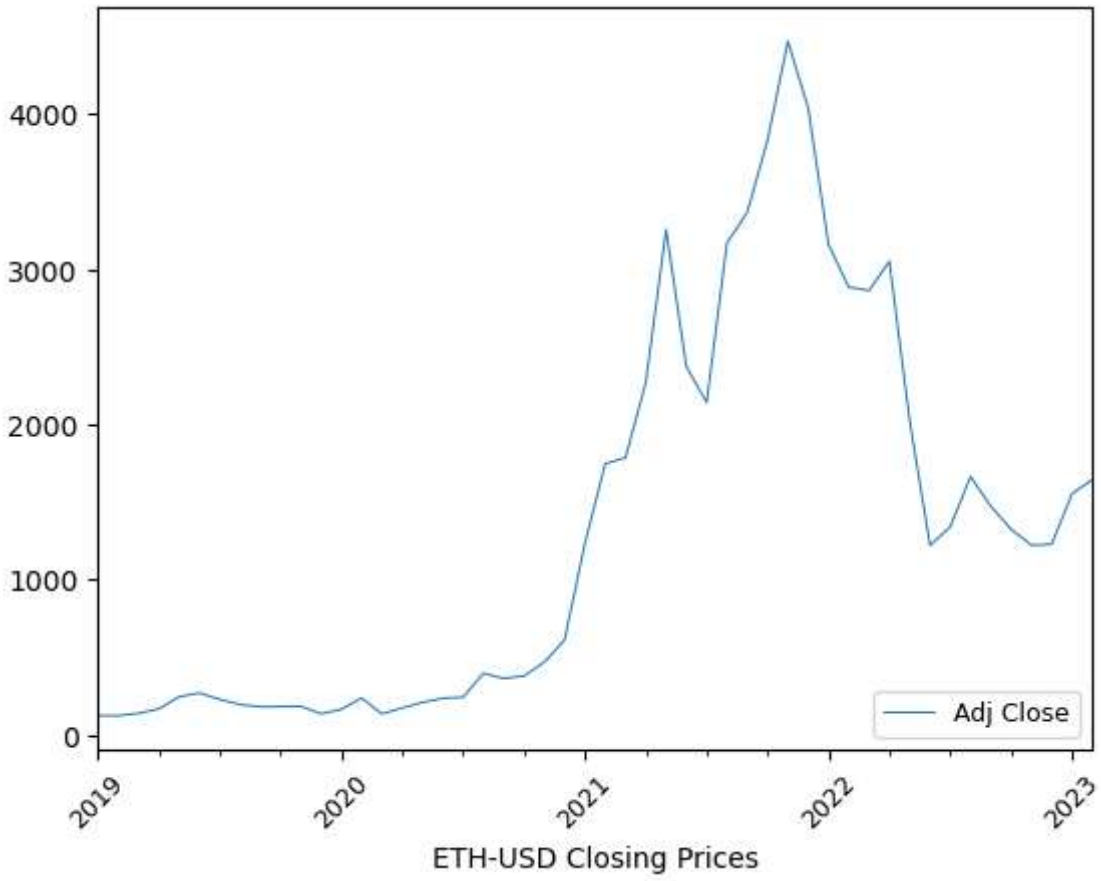
Out[21]:

Date	Open	High	Low	Close	Adj Close	Volume
2023-02-02	1641.365967	1704.458130	1641.322632	1643.241577	1643.241577	10558081069
2023-02-03	1642.904663	1670.696899	1634.223389	1664.745605	1664.745605	8169519805
2023-02-04	1664.472290	1690.099609	1648.189209	1667.059204	1667.059204	5843302512
2023-02-05	1667.166504	1671.770264	1616.391846	1631.645874	1631.645874	6926696531
2023-02-06	1631.645264	1653.715454	1611.319092	1616.247070	1616.247070	6919871886

In [22]:
```
#Converting daily return data to monthly data, required statistic: mean, median
data_eth_mean=ETH['Adj Close'].resample('m').mean()
data_eth_med=ETH['Adj Close'].resample('m').median()
data_eth_med.tail(5)
```

Out[22]:
```
Date
2022-10-31    1322.604248
2022-11-30    1220.122986
2022-12-31    1226.974365
2023-01-31    1552.479492
2023-02-28    1642.517151
Freq: M, Name: Adj Close, dtype: float64
```

In [23]:
```
ax11=data_eth_med.plot(linewidth=.7)
#ax11.xaxis.set_major_formatter(mdates.DateFormatter("%b %Y"))
plt.legend(loc='lower right', ncol=3, fontsize=9)
plt.xlabel('ETH-USD Closing Prices')
plt.xticks(rotation=45, fontsize=9)
plt.show( )
```



In [24]:
```
#high volatility of Ethereum, particularly during 2021-2022, dropping further near 202
```

In [25]:
```
#I created a dataframe with the daily prices. I use the 'Adj Close' column as
#it is a price adjusted for splits and dividends distribution. To extract daily return
#I calculate the percentage change of the daily prices using the pct_change() method.
datar_eth=pd.DataFrame()
returns_eth=ETH['Adj Close'].pct_change(1)
datar_eth['ticker%']=returns_eth
datar_eth.head(5)
```

Out[25]:

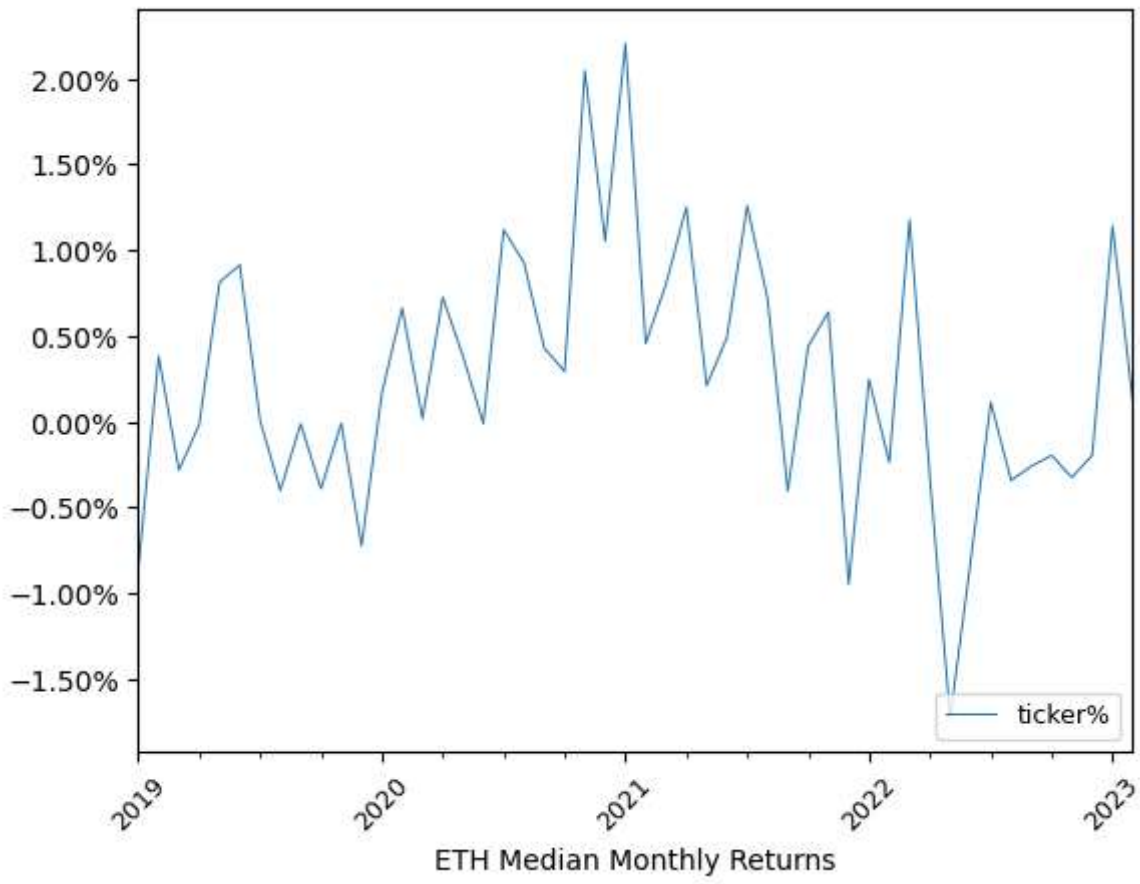
Date	ticker%
2019-01-01	NaN
2019-01-02	0.101039
2019-01-03	-0.038135
2019-01-04	0.036523
2019-01-05	0.006836

In [26]:
```
#Converting daily return data to monthly data, required statistic: mean, median
datar_meaneth=datar_eth.resample('m').mean()
datar_medeth=datar_eth.resample('m').median()
datar_medeth.head(5)
```

Out[26]:

Date	ticker%
2019-01-31	-0.008610
2019-02-28	0.003808
2019-03-31	-0.002840
2019-04-30	-0.000180
2019-05-31	0.008111

In [27]:
```
ax11=datar_medeth.plot(linewidth=.7)
ax11.yaxis.set_major_formatter(mtick.PercentFormatter(1.0))
plt.legend(loc='lower right', ncol=3, fontsize=9)
plt.xlabel('ETH Median Monthly Returns')
plt.xticks(rotation=45, fontsize=9)
plt.show( )
```



In [28]:
```
# Dialy returns provide more data points for data analysis, however, annualised return
# preferred by investors and give a better basis for the comparison of the assets.
# To annualise daily returns of the assets, I calculate mean returns and
# multiply them by 253 (average number of trading days a year).
# Common measure of return volatility, or risk, is the standard deviation.
# Standard deviation is calculated as a square root of the variance and has an
# advantage of having the same measure unit as the mean.
# Using standard deviation as a measure of risk, we can calculate risk-adjusted perfor
# for instance Sharpe ratio. Sharpe Ratio is calculated as annualised returns divided
```

In [29]:
```
#ETH Arithmeti mean of % daily returns
eth_means=datar_eth.mean()
print ("ETH USD Annualised Returns (arithmetic mean) are:", eth_means*253*100)

#ETH Co-variance
daily_eth_cov=round(datar_eth.cov(),3)
annual_eth_cov=round(daily_eth_cov*253,3)
print("daily ETH covariance:",daily_eth_cov.iloc[0,0])
print("Annual ETH covarince:",annual_eth_cov.iloc[0,0])

#ETH standard Deviation
eth_stddev=datar_eth.std()*np.sqrt(253)
print('ETH Standard Deviation are: ', eth_stddev)

#ETH Variance
print ("ETH Variance:", (eth_stddev*eth_stddev))

#ETH Sharpe Ratio
eth_sr=eth_means*253/eth_stddev
print ('ETH Sharpe Ratio: ',btc_sr)
```

```
ETH USD Annualised Returns (arithmetic mean) are: ticker%    70.955402
dtype: float64
daily ETH covariance: 0.002
Annual ETH covarince: 0.506
ETH Standard Deviation are:  ticker%    0.761881
dtype: float64
ETH Variance: ticker%    0.580462
dtype: float64
ETH Sharpe Ratio:  ticker%    0.810268
dtype: float64
```

In [30]:
```
#Appending lists with code,returns ans annual covarianceL this we will use to find
#respective weights of the assets for the optimal portfolio in terms of min volatilty,
#highest sharpe ratio, high expected retruns
summary_code.append("ETH-USD")
summary_ann_returns.append(eth_means*253)

summary_ann_cov=np.append(summary_ann_cov,annual_eth_cov.iloc[0,0])
```

In [ ]:
```
```

In [31]:
```
print (summary_code)
print (summary_ann_returns)
print (summary_ann_cov)
```

```
['BTC-USD', 'ETH-USD']
[ticker%    0.477911
dtype: float64, ticker%    0.709554
dtype: float64]
[0.253 0.506]
```

3. Tether USD (USDT)

In [32]:
```
USDT = yfin.download('USDT-USD', start='2019-1-1',end='2023-2-7')
```

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

In [33]:
```
USDT.tail(5)
```

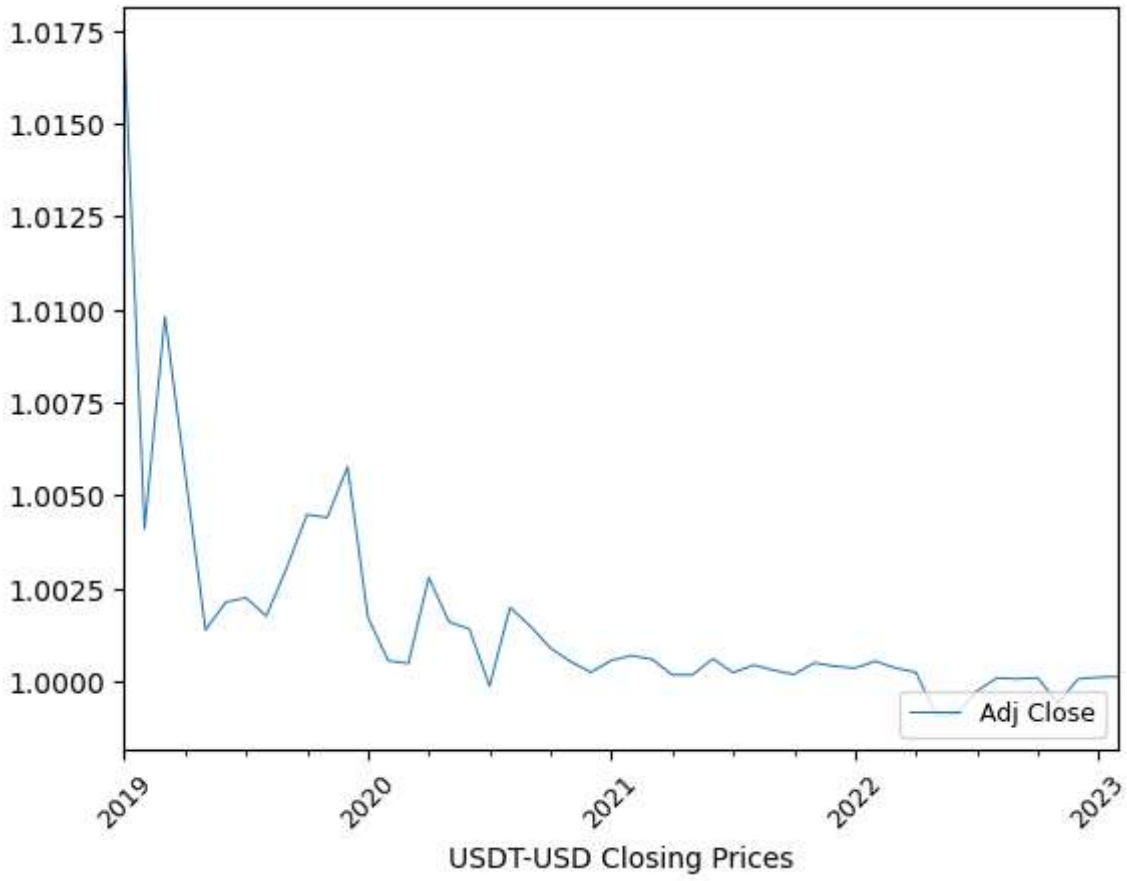
Out[33]:

Date	Open	High	Low	Close	Adj Close	Volume
2023-02-02	1.000088	1.000328	1.000083	1.000140	1.000140	43575657052
2023-02-03	1.000142	1.000276	1.000078	1.000131	1.000131	36792033814
2023-02-04	1.000122	1.000162	1.000054	1.000100	1.000100	24493083193
2023-02-05	1.000093	1.000268	1.000089	1.000198	1.000198	30313569702
2023-02-06	1.000209	1.000274	1.000067	1.000102	1.000102	32088027094

In [34]:
```
#Converting daily return data to monthly data, required statistic: mean, median
data_usdt_mean=USDT['Adj Close'].resample('m').mean()
data_usdt_med=USDT['Adj Close'].resample('m').median()
data_usdt_med.tail(5)
```

Out[34]:
```
Date
2022-10-31    1.000085
2022-11-30    0.999411
2022-12-31    1.000057
2023-01-31    1.000099
2023-02-28    1.000117
Freq: M, Name: Adj Close, dtype: float64
```

In [35]:
```
ax11=data_usdt_med.plot(linewidth=.7)
#ax11.xaxis.set_major_formatter(mdates.DateFormatter("%b %Y"))
plt.legend(loc='lower right', ncol=3, fontsize=9)
plt.xlabel('USDT-USD Closing Prices')
plt.xticks(rotation=45, fontsize=9)
plt.show( )
```



In [36]:
```
#I created a dataframe with the daily prices. I use the 'Adj Close' column as
#it is a price adjusted for splits and dividends distribution. To extract daily return
#I calculate the percentage change of the daily prices using the pct_change() method.
datar_usdt=pd.DataFrame()
returns_usdt=USDT['Adj Close'].pct_change(1)
datar_usdt['ticker%']=returns_usdt
datar_usdt.head(5)
```

Out[36]:

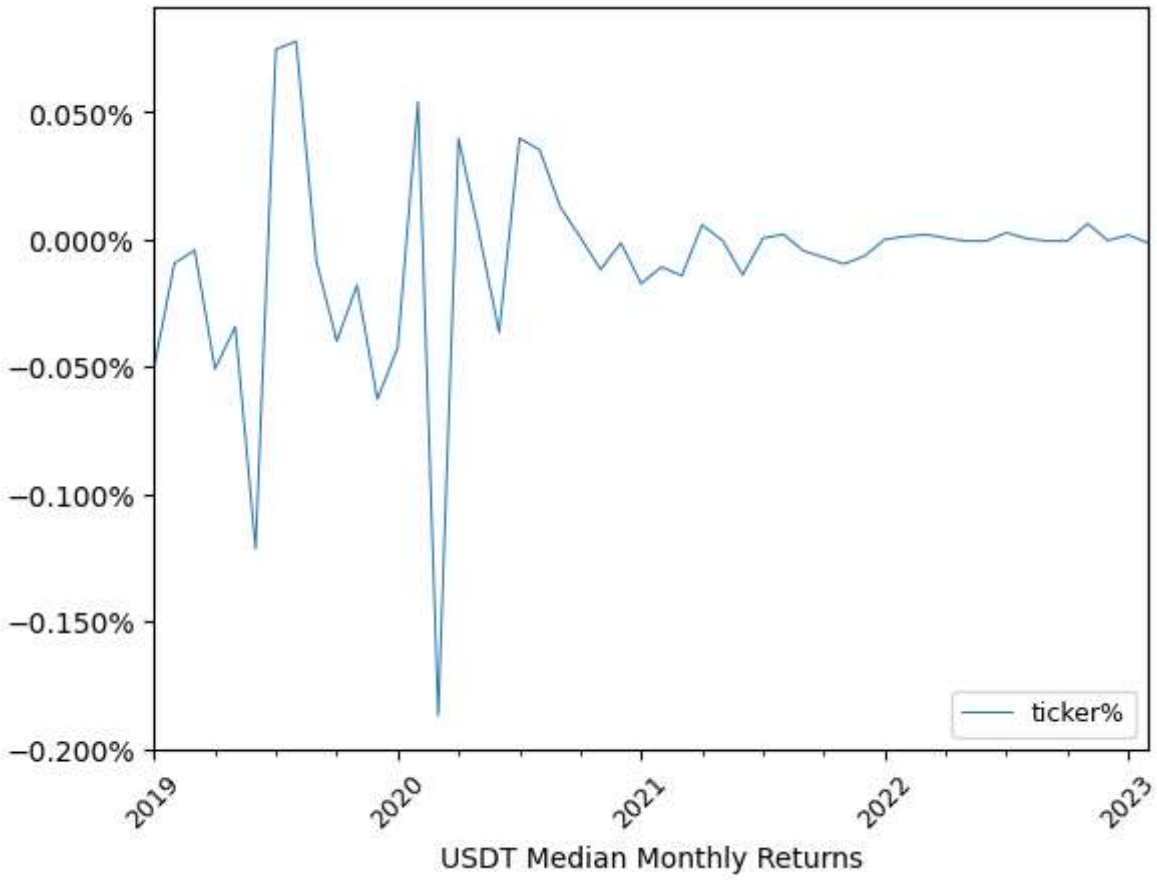
Date	ticker%
2019-01-01	NaN
2019-01-02	0.005575
2019-01-03	-0.004433
2019-01-04	-0.002230
2019-01-05	0.001267

In [37]:
```
#Converting daily return data to monthly data, required statistic: mean, median
datar_meanusdt=datar_usdt.resample('m').mean()
datar_medusdt=datar_usdt.resample('m').median()
datar_medusdt.head(5)
```

Out[37]:

Date	ticker%
2019-01-31	-0.000502
2019-02-28	-0.000094
2019-03-31	-0.000042
2019-04-30	-0.000508
2019-05-31	-0.000343

In [38]:
```
ax11=datar_medusdt.plot(linewidth=.7)
ax11.yaxis.set_major_formatter(mtick.PercentFormatter(1.0))
plt.legend(loc='lower right', ncol=3, fontsize=9)
plt.xlabel('USDT Median Monthly Returns')
plt.xticks(rotation=45, fontsize=9)
plt.show( )
```



In [39]:
```
## Dialy returns provide more data points for data analysis, however, annualised retur
## preferred by investors and give a better basis for the comparison of the assets.
## To annualise daily returns of the assets, I calculate mean returns and
## multiply them by 253 (average number of trading days a year).
## Common measure of return volatility, or risk, is the standard deviation. Standard d
##calculated as a square root of the variance and has an advantage of having the same
## Using standard deviation as a measure of risk, we can calculate risk-adjusted perfo
## for instance Sharpe ratio. Sharpe Ratio is calculated as annualised returns divided
```

In [40]:
```
#USDT Arithmeti mean of % daily returns
usdt_means=datar_usdt.mean()
print ("USDT USD Annualised Returns (arithmetic mean) are:", usdt_means*253*100)

#USDT Co-variance
daily_usdt_cov=round(datar_usdt.cov(),5)
annual_usdt_cov=round(daily_usdt_cov*253,5)
print("daily USDT covariance:",daily_usdt_cov.iloc[0,0])
print("Annual USDT covarince:",annual_usdt_cov.iloc[0,0])

# USDT standard Deviation
usdt_stddev=datar_usdt.std()*np.sqrt(253)
print('USDT Standard Deviation are: ', usdt_stddev)

#USDT Variance
print ("USDT Variance:", (usdt_stddev*usdt_stddev))

#USDT Sharpe Ratio
usdt_sr=usdt_means*253/usdt_stddev
print ('USDT Sharpe Ratio: ',usdt_sr)
```

```
USDT USD Annualised Returns (arithmetic mean) are: ticker%   -0.158103
dtype: float64
daily USDT covariance: 1e-05
Annual USDT covarince: 0.00253
USDT Standard Deviation are:  ticker%    0.054319
dtype: float64
USDT Variance: ticker%    0.002951
dtype: float64
USDT Sharpe Ratio:  ticker%   -0.029106
dtype: float64
```

In [41]:
```
summary_code.append("USDT-USD")
summary_ann_returns.append(usdt_means*253)
```

In [42]:
```
summary_ann_cov=np.append(summary_ann_cov,annual_usdt_cov.iloc[0,0])
```

In [43]:
```
print (summary_code)
print (summary_ann_returns)
print (summary_ann_cov)
```

```
['BTC-USD', 'ETH-USD', 'USDT-USD']
[ticker%    0.477911
dtype: float64, ticker%    0.709554
dtype: float64, ticker%   -0.001581
dtype: float64]
[0.253   0.506   0.00253]
```

4. BNB USD

In [44]:
```
BNB = yfin.download('BNB-USD', start='2019-1-1',end='2023-2-7')
```

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

In [45]:
```
BNB.tail(5)
```

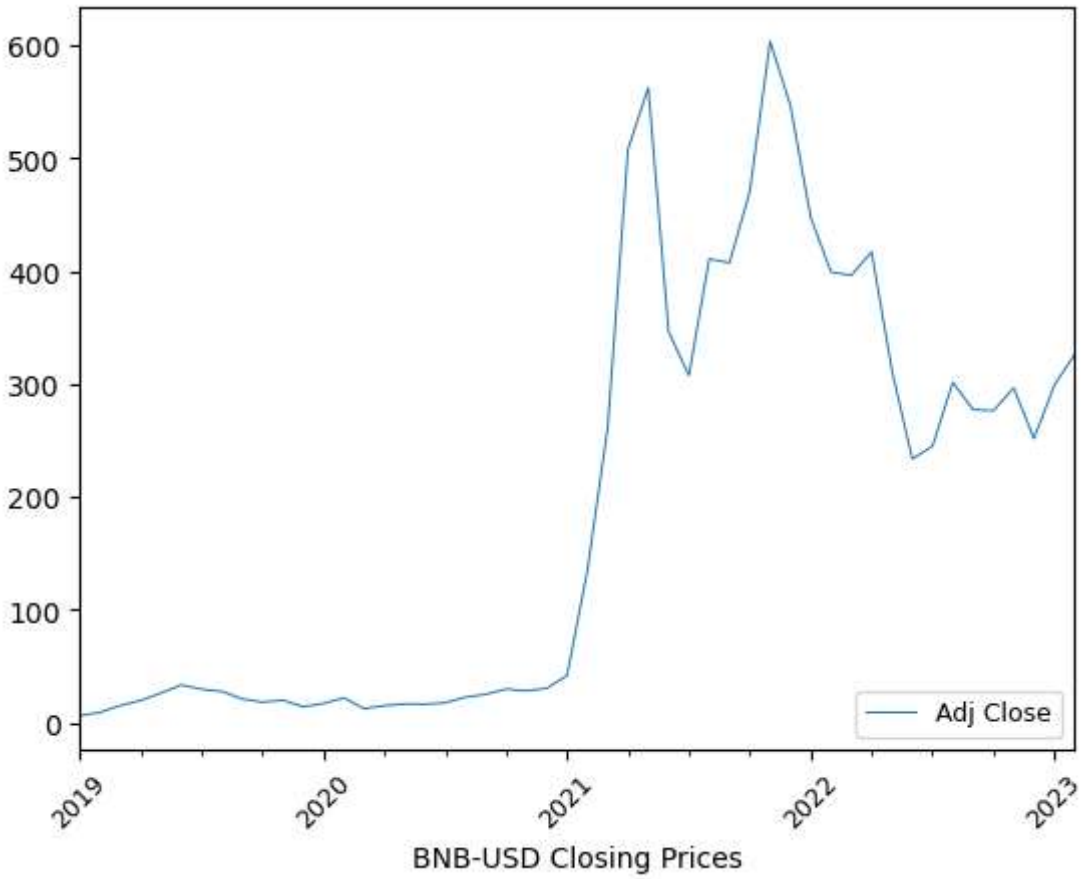
Out[45]:

Date	Open	High	Low	Close	Adj Close	Volume
2023-02-02	317.346191	333.495544	316.711060	323.349792	323.349792	909077028
2023-02-03	323.291351	334.545166	318.903229	332.268677	332.268677	861598551
2023-02-04	332.249695	333.870941	327.741425	330.617096	330.617096	440935963
2023-02-05	330.558258	337.320526	324.879486	327.869354	327.869354	662969265
2023-02-06	327.861847	330.957428	321.946136	324.627167	324.627167	521894604

In [46]:
```
#Converting daily return data to monthly data, required statistic: mean, median
data_bnb_mean=BNB['Adj Close'].resample('m').mean()
data_bnb_med=BNB['Adj Close'].resample('m').median()
data_bnb_med.tail(5)
```

Out[46]:
```
Date
2022-10-31    276.469574
2022-11-30    296.672806
2022-12-31    251.744537
2023-01-31    298.999237
2023-02-28    326.248260
Freq: M, Name: Adj Close, dtype: float64
```

In [47]:
```
ax11=data_bnb_med.plot(linewidth=.7)
#ax11.xaxis.set_major_formatter(mdates.DateFormatter("%b %Y"))
plt.legend(loc='lower right', ncol=3, fontsize=9)
plt.xlabel('BNB-USD Closing Prices')
plt.xticks(rotation=45, fontsize=9)
plt.show( )
```



In [48]:
```
#I created a dataframe with the daily prices. I use the 'Adj Close' column as
#it is a price adjusted for splits and dividends distribution. To extract daily return
#I calculate the percentage change of the daily prices using the pct_change() method.
datar_bnb=pd.DataFrame()
returns_bnb=BNB['Adj Close'].pct_change(1)
datar_bnb['ticker%']=returns_bnb
datar_bnb.head(5)
```

Out[48]:

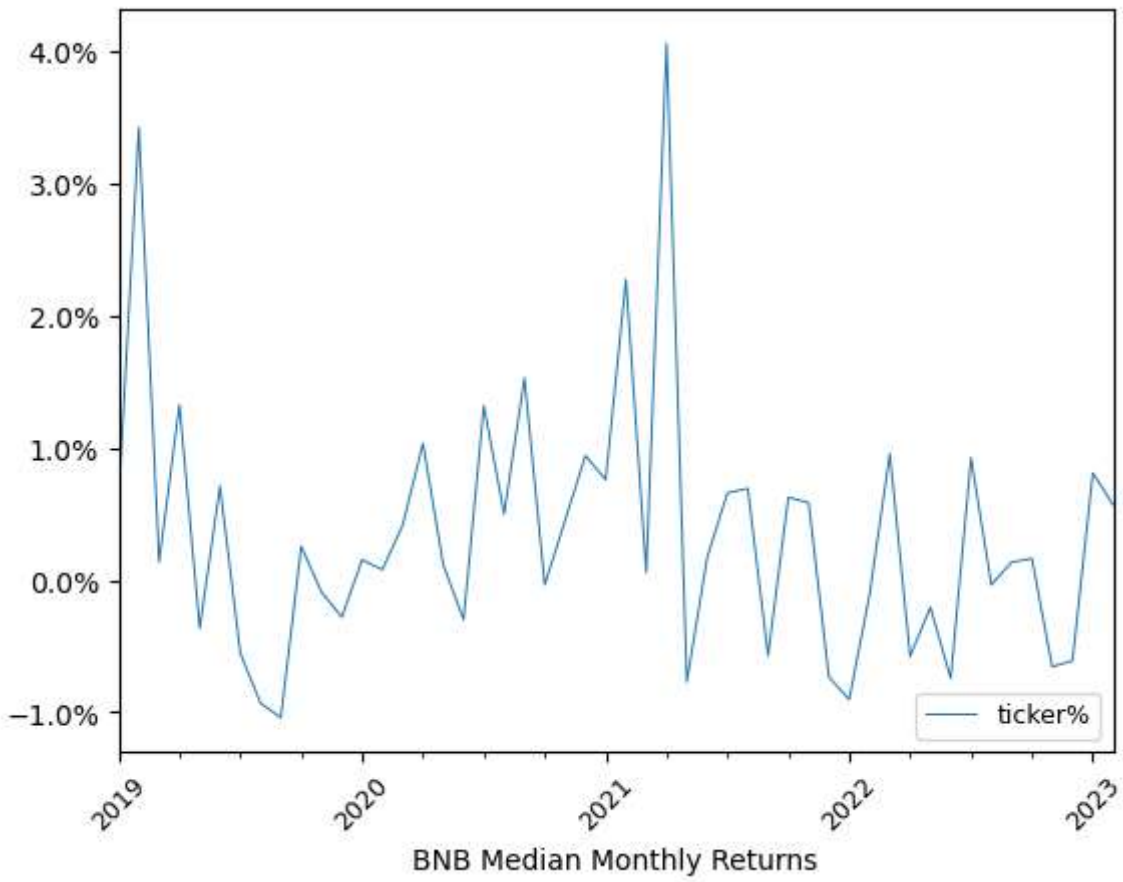
Date	ticker%
2019-01-01	NaN
2019-01-02	0.018656
2019-01-03	-0.046065
2019-01-04	0.027374
2019-01-05	0.000067

In [49]:
```
#Converting daily return data to monthly data, required statistic: mean, median
datar_meanbnb=datar_bnb.resample('m').mean()
datar_medbnb=datar_bnb.resample('m').median()
datar_medbnb.head(5)
```

Out[49]:

Date	ticker%
2019-01-31	0.005950
2019-02-28	0.034277
2019-03-31	0.001378
2019-04-30	0.013270
2019-05-31	-0.003653

In [50]:
```
ax11=datar_medbnb.plot(linewidth=.7)
ax11.yaxis.set_major_formatter(mtick.PercentFormatter(1.0))
plt.legend(loc='lower right', ncol=3, fontsize=9)
plt.xlabel('BNB Median Monthly Returns')
plt.xticks(rotation=45, fontsize=9)
plt.show( )
```



In [51]:
```
## Dialy returns provide more data points for data analysis, however, annualised retur
## preferred by investors and give a better basis for the comparison of the assets.
## To annualise daily returns of the assets, I calculate mean returns and
## multiply them by 253 (average number of trading days a year).
## Common measure of return volatility, or risk, is the standard deviation. Standard d
##calculated as a square root of the variance and has an advantage of having the same
## Using standard deviation as a measure of risk, we can calculate risk-adjusted perfo
## for instance Sharpe ratio. Sharpe Ratio is calculated as annualised returns divided
```

In [52]:
```
#BNB Arithmeti mean of % daily returns
bnb_means=datar_bnb.mean()
print ("BNB USD Annualised Returns (arithmetic mean) are:", bnb_means*253*100)

#BNB Co-variance
daily_bnb_cov=round(datar_bnb.cov(),5)
annual_bnb_cov=round(daily_bnb_cov*253,5)
print("daily BNB covariance:",daily_bnb_cov.iloc[0,0])
print("Annual BNB covarince:",annual_bnb_cov.iloc[0,0])

# BNB standard Deviation
bnb_stddev=datar_bnb.std()*np.sqrt(253)
print('BNB Standard Deviation are: ', bnb_stddev)

#BNB Variance
print ("BNB Variance:", (bnb_stddev*bnb_stddev))

#BNB Sharpe Ratio
bnb_sr=bnb_means*253/bnb_stddev
print ('BNB Sharpe Ratio: ',bnb_sr)
```

```
BNB USD Annualised Returns (arithmetic mean) are: ticker%    102.40598
dtype: float64
daily BNB covariance: 0.00284
Annual BNB covarince: 0.71852
BNB Standard Deviation are:  ticker%    0.848192
dtype: float64
BNB Variance: ticker%    0.71943
dtype: float64
BNB Sharpe Ratio:  ticker%    1.207344
dtype: float64
```

In [53]:
```
summary_code.append("BNB-USD")
#we multiply mean return wuth 253 to get annual returns
summary_ann_returns.append(bnb_means*253)
summary_ann_cov=np.append(summary_ann_cov,annual_bnb_cov.iloc[0,0])
```

In [54]:
```
print (summary_code)
print (summary_ann_returns)
print (summary_ann_cov)
```

```
['BTC-USD', 'ETH-USD', 'USDT-USD', 'BNB-USD']
[ticker%    0.477911
dtype: float64, ticker%    0.709554
dtype: float64, ticker%   -0.001581
dtype: float64, ticker%    1.02406
dtype: float64]
[0.253   0.506   0.00253 0.71852]
```

In [ ]:

Merging the Asset DataFrames

In [64]:
```
#Converting daily return data to yearly data, required statistic: mean, median
#datar_ymeanbtc=datar_btc.resample('Y').mean()
#datar_ymedbtc=datar_btc.resample('Y').median()
```

```
#datar_ymeaneth=datar_eth.resample('Y').mean()
#datar_ymedeth=datar_eth.resample('Y').median()

#datar_ymeanusdt=datar_usdt.resample('Y').mean()
#datar_ymedusdt=datar_usdt.resample('Y').median()

#datar_ymeanbnb=datar_bnb.resample('Y').mean()
#datar_ymedbnb=datar_bnb.resample('Y').median()
```

Merging daily and and mean monthly returns

In [65]:
```
#Concatenating yearly retruns
#ydataset1=pd.concat([datar_ymeanbtc,datar_ymeaneth,datar_ymeanusdt,datar_ymeanbnb],ax
```

In [66]:
```
#setting column names
#ydataset1.columns=['BTC-USD','ETH-USD','USDT-USD','BNB-USD']
#ydataset1.head()
```

In [68]:
```
#Concatenating daily retruns
ddataset2=pd.concat([datar_btc,datar_eth,datar_usdt,datar_bnb],axis=1,join='inner')
ddataset2.columns=['BTC-USD','ETH-USD','USDT-USD','BNB-USD']
ddataset2.head(5)
```

Out[68]:

Date	BTC-USD	ETH-USD	USDT-USD	BNB-USD
2019-01-01	NaN	NaN	NaN	NaN
2019-01-02	0.025989	0.101039	0.005575	0.018656
2019-01-03	-0.027050	-0.038135	-0.004433	-0.046065
2019-01-04	0.005467	0.036523	-0.002230	0.027374
2019-01-05	-0.003246	0.006836	0.001267	0.000067

Correlation Matrix

In [69]:
```
#Correlations are the foundation of portfolio diversification. In short,
#the most diversified portfolio is the portfolio of uncorrelated assets
corrmatrix=ddataset2.corr()
corrmatrix
```

Out[69]:

	BTC-USD	ETH-USD	USDT-USD	BNB-USD
BTC-USD	1.000000	0.820447	-0.035211	0.650885
ETH-USD	0.820447	1.000000	-0.066763	0.680432
USDT-USD	-0.035211	-0.066763	1.000000	-0.062556
BNB-USD	0.650885	0.680432	-0.062556	1.000000

In [ ]:

Variance and Covariance MAtrix

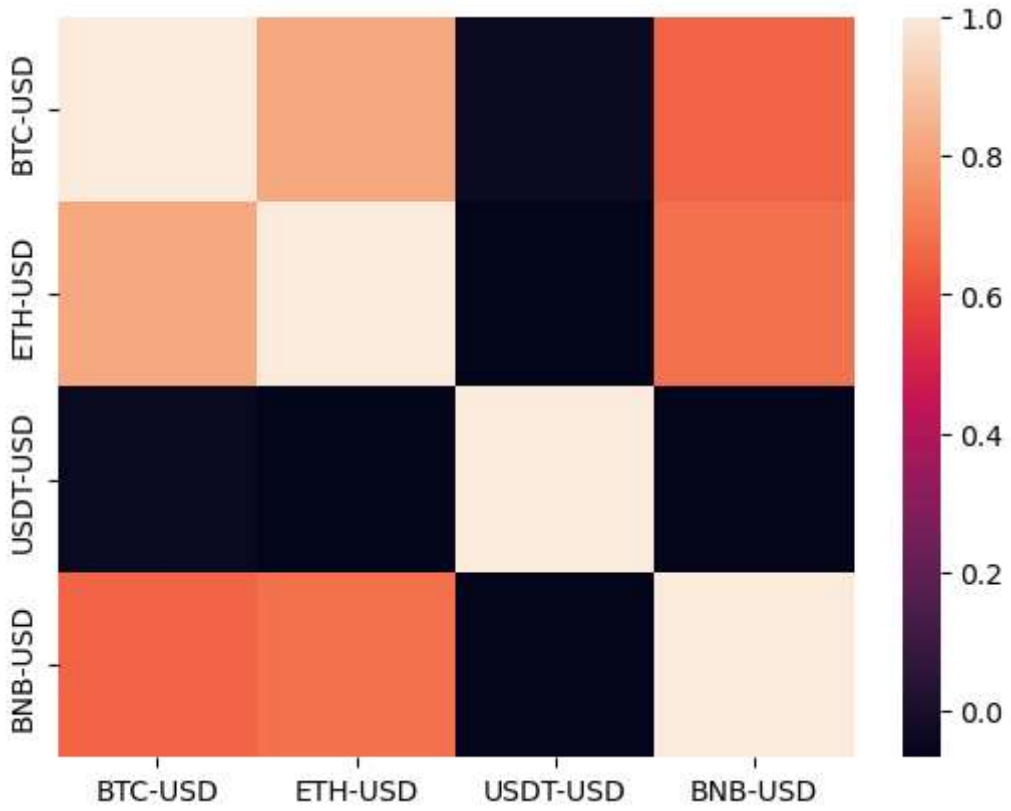
In [70]:
```
#Variance aqnd Covariance MAtrix
#The diagonal elements of the variance_matrix represent the variance of each asset,
#while the off-diagonal terms represent the covariance between the two assets,
#eg: (row=1,column=2) element represents the covariance between BTC and ETH.
#we multiple the covariance matrix with 253 as there are 253 trading days in a year
variance_matrix=ddataset2.cov()*253
variance_matrix
```

Out[70]:

	BTC-USD	ETH-USD	USDT-USD	BNB-USD
BTC-USD	0.347885	0.368685	-0.001128	0.325624
ETH-USD	0.368685	0.580462	-0.002763	0.439710
USDT-USD	-0.001128	-0.002763	0.002951	-0.002882
BNB-USD	0.325624	0.439710	-0.002882	0.719430

In [71]:
```
#plot the heatmap
sns.heatmap(corrmatrix)
```

Out[71]:
```
<AxesSubplot:>
```



Optimal Portfolio

In [ ]:

In [85]:
```
# The task for us is to optimize the weights. So that we can maximize our return or
# minimize our risk or maximise the Sharpe Ratio
# we will compute around 300 probable combinations of weights
no_ofcryptoassets=len(ddataset2.columns)
no_ofcryptoportfolios=500
p_returns, p_volatility, p_sharperatio, p_coinweights =([] for i in range(4))

for b_crypto_portfolio in range(no_ofcryptoportfolios):
    weights = np.random.random(no_ofcryptoassets)
    weights /= np.sum(weights)

    returns = np.dot(weights,summary_ann_returns)*100

    volatility = np.sqrt(np.dot(np.dot(weights.T,variance_matrix),weights))*100
    #print (volatility)
    p_sharperatio.append(returns/volatility)
    p_returns.append(returns[0])
    p_volatility.append(volatility)
    p_coinweights.append(weights)

    # a dictionary for Returns and Risk values of each portfolio
    portfolio = {'volatility': p_volatility,
             'sharpe_ratio': p_sharperatio, 'returns': p_returns}
```

In [86]:
```
for counter,symbol in enumerate(summary_code):
    portfolio[symbol+'-%'] = [Weight[counter] for Weight in p_coinweights]
```

In [87]:
```
df_portfolio1 = pd.DataFrame(portfolio)
```

Portfolio weights combination with Max Returns

In [88]:
```
df_portfolio2 = df_portfolio1.sort_values(by=['returns'],ascending=False)
df_portfolio2.head()
```

Out[88]:

	volatility	sharpe_ratio	returns	BTC-USD-%	ETH-USD-%	USDT-USD-%	BNB-USD-%
38	78.553152	[1.216034846089293]	95.523370	0.009588	0.139868	0.019110	0.831434
151	72.457246	[1.2064496616711675]	87.416019	0.073202	0.251552	0.030036	0.645211
134	72.325744	[1.2011802929094473]	86.876259	0.022514	0.343505	0.034092	0.599888
195	71.764303	[1.2000941998456702]	86.123924	0.209886	0.112560	0.012471	0.665083
291	71.348645	[1.1871360143654988]	84.700546	0.091094	0.357547	0.014481	0.536877

Portfolio1 with Min Volatility (Standard Deviation)

In [90]:
```
#Portfoilio1 with MIN VOLATiLITY
#idxmin gives us the min value for volatiltiy or standard deviation
min_vol_port1=df_portfolio1.iloc[df_portfolio1['volatility'].idxmin()]
min_vol_port1
```

Out[90]:
```
volatility                19.273016
sharpe_ratio    [1.0343340882761498]
returns                   19.934737
BTC-USD-%                  0.035599
ETH-USD-%                   0.18581
USDT-USD-%                 0.728161
BNB-USD-%                   0.05043
Name: 361, dtype: object
```

In [91]:
```
#SHARPE RAIO2: Adjusting Risk free Rate (Rf-0.05)
rf=0.05
df_portfolio1['sharperatio2']=((df_portfolio1['returns']-rf)/df_portfolio1['volatility
print (df_portfolio1.head(5))
```

```
   volatility          sharpe_ratio    returns  BTC-USD-%  ETH-USD-%  \
0   37.133444  [1.1151520095091318]  41.409434   0.101286   0.268022
1   49.144134  [1.1224412450325547]  55.161403   0.226478   0.261830
2   46.129561  [1.1417314238476859]  52.667570   0.353014   0.021317
3   48.469122  [1.1237834009708587]  54.468795   0.319243   0.154418
4   68.094165  [1.0288637093717568]  70.059615   0.231426   0.616639

   USDT-USD-%  BNB-USD-%  sharperatio2
0    0.458595   0.172097      1.113806
1    0.259748   0.251944      1.121424
2    0.290435   0.335235      1.140648
3    0.250041   0.276298      1.122752
4    0.003056   0.148879      1.028129
```

Portfolio combination with Min Volatilty

In [92]:
```
df_portfolio4 = df_portfolio1.sort_values(by=['volatility'],ascending=True)
df_portfolio4.head()
```

Out[92]:

	volatility	sharpe_ratio	returns	BTC-USD-%	ETH-USD-%	USDT-USD-%	BNB-USD-%	sharperatio2
361	19.273016	[1.0343340882761498]	19.934737	0.035599	0.185810	0.728161	0.050430	1.031740
242	19.936251	[0.8533400781083583]	17.012402	0.305942	0.013036	0.665678	0.015345	0.850832
157	26.636588	[0.9839393439359725]	26.208787	0.292779	0.081086	0.562155	0.063980	0.982062
376	28.298559	[1.0543986119493047]	29.837961	0.151341	0.192649	0.567875	0.088134	1.052632
212	28.730817	[1.0098449000588317]	29.013669	0.012849	0.328745	0.607926	0.050480	1.008105

Portfolio2 with Max Sharpe Ratio

In [93]:
```
#Finding the optimal portfolio: highest Sharpe Raio
#risk factor
```

```
optimal_risky_port2=df_portfolio1.iloc[df_portfolio1['sharperatio2'].idxmax()]

print (optimal_risky_port2)
```

```
volatility                78.553152
sharpe_ratio    [1.216034846089293]
returns                   95.52337
BTC-USD-%                 0.009588
ETH-USD-%                 0.139868
USDT-USD-%                 0.01911
BNB-USD-%                 0.831434
sharperatio2              1.215398
Name: 38, dtype: object
```

In [ ]:
```
#This looks like combination for medium return and medium volatitly
```

In [94]:
```
df_portfolio5 = df_portfolio1.sort_values(by=['sharperatio2'],ascending=False)
df_portfolio5.head()
```

Out[94]:

	volatility	sharpe_ratio	returns	BTC-USD-%	ETH-USD-%	USDT-USD-%	BNB-USD-%	sharperatio2
38	78.553152	[1.216034846089293]	95.523370	0.009588	0.139868	0.019110	0.831434	1.215398
263	37.752036	[1.2158807488694126]	45.901974	0.010393	0.092701	0.516954	0.379952	1.214556
240	44.078261	[1.2149175055380166]	53.551451	0.013116	0.132596	0.428688	0.425600	1.213783
93	48.253043	[1.2140319543051816]	58.580736	0.045257	0.093904	0.374402	0.486437	1.212996
396	63.395152	[1.2123898092513927]	76.859636	0.077659	0.104763	0.175599	0.641979	1.211601

PORTFOLIO3

In [95]:
```
#Portfolio with Highest Return
optimal_return_port3=df_portfolio1.iloc[df_portfolio1['returns'].idxmax()]

print (optimal_return_port3)
```

```
volatility                78.553152
sharpe_ratio    [1.216034846089293]
returns                   95.52337
BTC-USD-%                 0.009588
ETH-USD-%                 0.139868
USDT-USD-%                 0.01911
BNB-USD-%                 0.831434
sharperatio2              1.215398
Name: 38, dtype: object
```

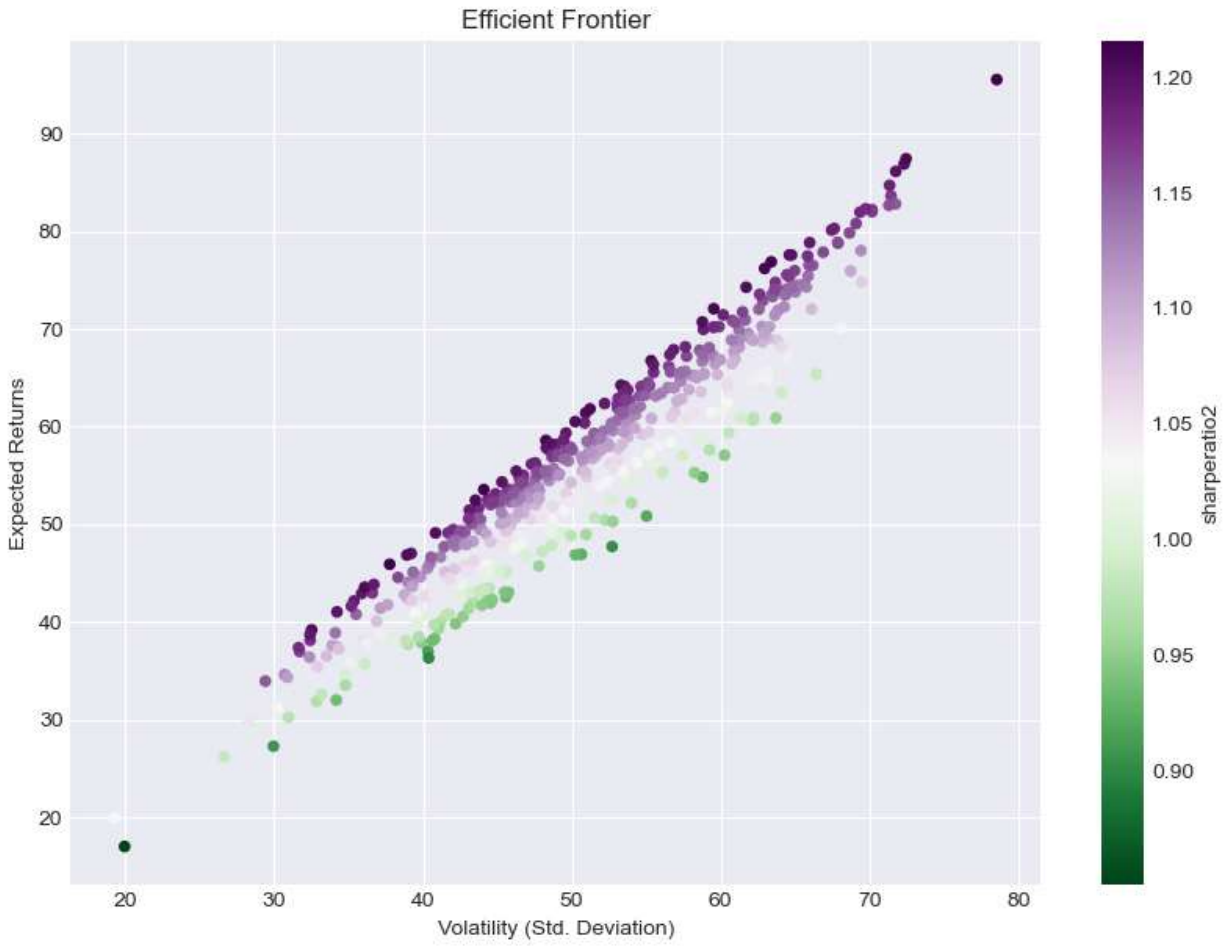
Charts and Graphs

In [96]:
```
#plotting scatter plots
order_cols = ['returns', 'volatility', 'sharpe_ratio','sharperatio2']+[symbol+'-%' for
df3 = df_portfolio1[order_cols]
```

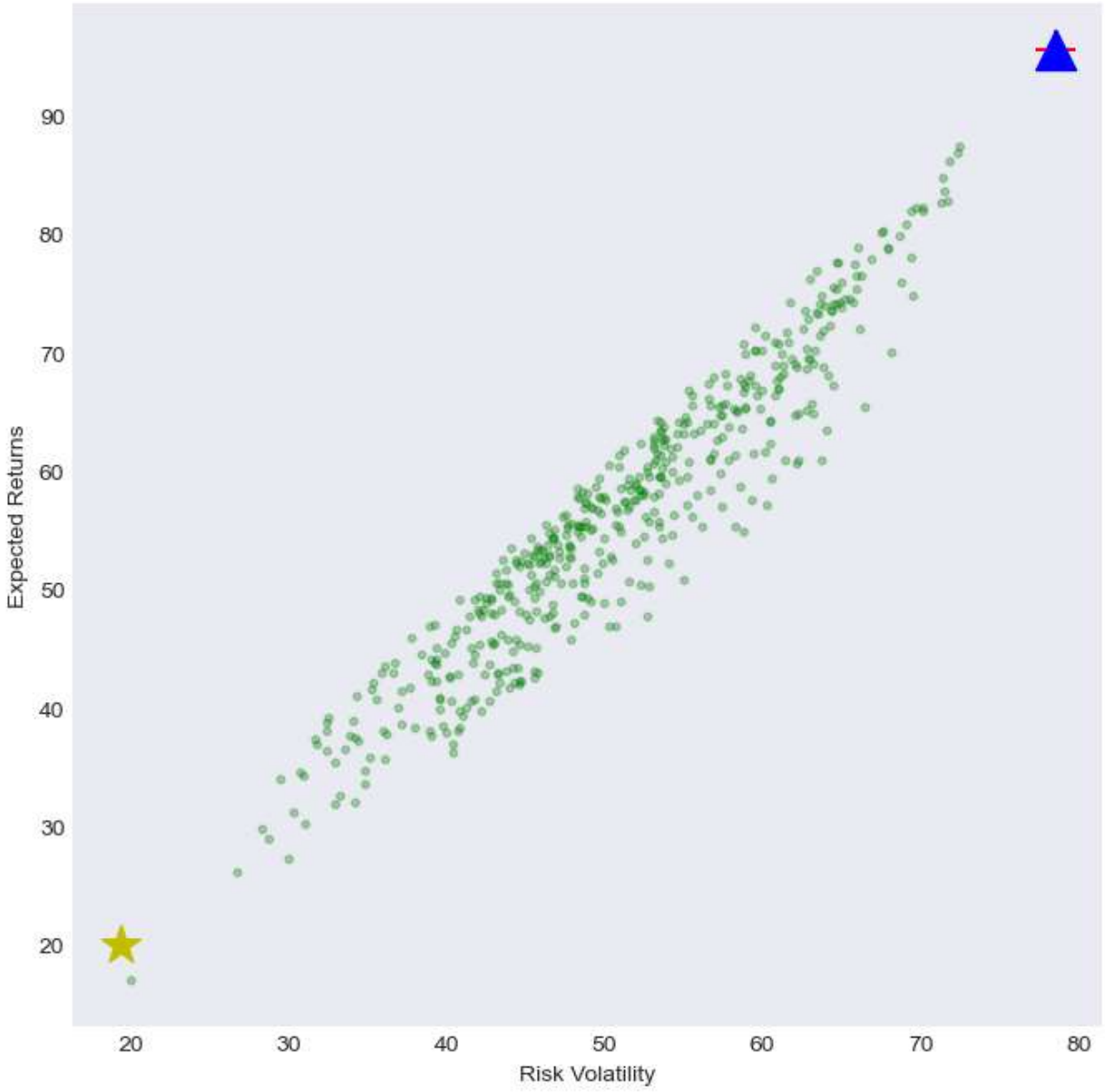
```
sharpe_portfolio = df3.loc[df3['sharperatio2'] == df3['sharperatio2'].max()]
min_variance_port = df3.loc[df3['volatility'] == df3['volatility'].min()]
max_returns_port = df3.loc[df3['returns'] == df3['returns'].max()]

plt.style.use('seaborn-dark')
df3.plot.scatter(x='volatility', y='returns', c='sharperatio2',
                 cmap='PRGn_r', figsize=(10, 7), grid=True)

plt.xlabel('Volatility (Std. Deviation)')
plt.ylabel('Expected Returns')
plt.title('Efficient Frontier')
plt.show()
```



In [97]:
```
#To find out where will the highest Sharpe Ratio point lies on the above curve ?
#plotting optmial portfolio and minimum volatiltiy
plt.subplots(figsize=(8,8))
plt.scatter(x=df3['volatility'],y=df3['returns'],marker='o',s=10,
            alpha=0.3, color='green')
#MiN VoLATiLity- makred by star(*)
plt.scatter(x=min_vol_port1[0],y=min_vol_port1[2],color='y',marker='*',s=300)
#Optimal Risky protfolio based on sharpe ratio, marked by -plus(+)
plt.scatter(optimal_risky_port2[0],optimal_risky_port2[2],color='red',marker='+',s=300
#Optimal Retrun based on retrun: marked by cap(^)
plt.scatter(optimal_return_port3[0],optimal_return_port3[2],color='blue',marker='^',s=
plt.xlabel('Risk Volatility')
plt.ylabel('Expected Returns')
plt.show()
```



In [100…
```
#portfolio with max sharep raio
sharpe_portfolio
```

Out[100]:

	returns	volatility	sharpe_ratio	sharperatio2	BTC-USD-%	ETH-USD-%	USDT-USD-%	BNB-USD-%
38	95.52337	78.553152	[1.216034846089293]	1.215398	0.009588	0.139868	0.01911	0.831434

In [101…
```
min_variance_port
```

Out[101]:

	returns	volatility	sharpe_ratio	sharperatio2	BTC-USD-%	ETH-USD-%	USDT-USD-%	BNB-USD-%
361	19.934737	19.273016	[1.0343340882761498]	1.03174	0.035599	0.18581	0.728161	0.05043

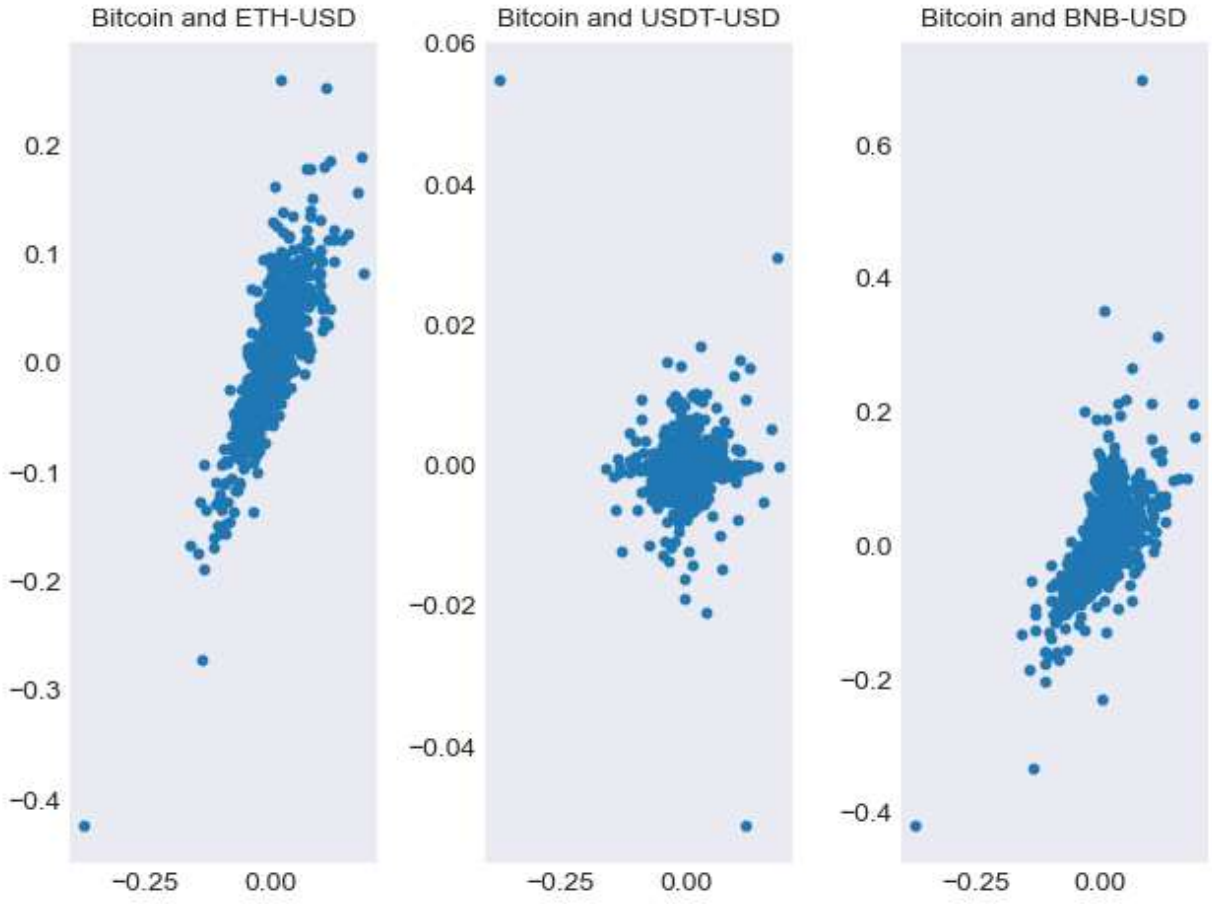
In [102…
```
#The portfolio with max sharpe ratio came out to be same as having max returns
max_returns_port
```

Out[102]:

	returns	volatility	sharpe_ratio	sharperatio2	BTC-USD-%	ETH-USD-%	USDT-USD-%	BNB-USD-%
38	95.52337	78.553152	[1.216034846089293]	1.215398	0.009588	0.139868	0.01911	0.831434

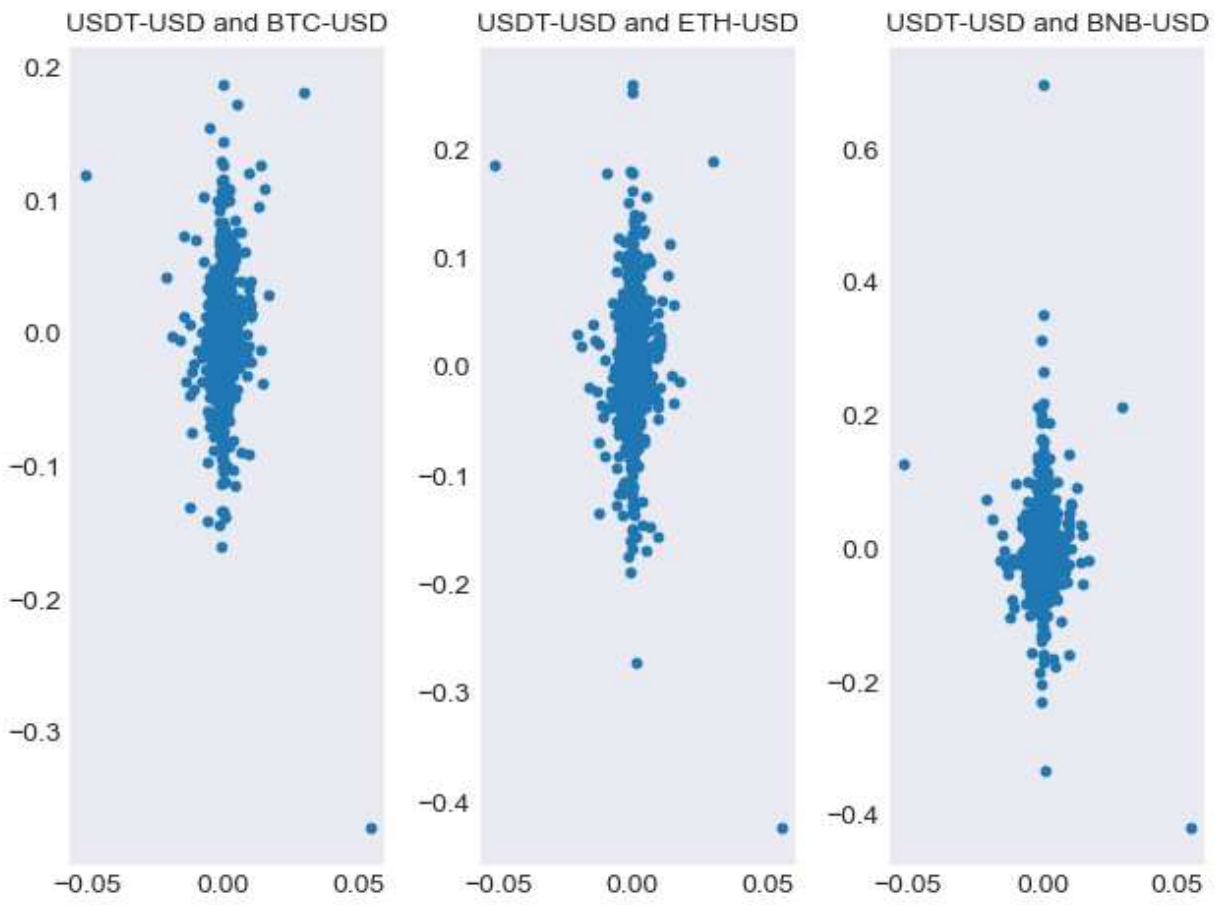
In [103…
```
#Charts and Grpahs
```

In [106…
```
#The scatter plots point at the existence of a linear relationship between Bitcoin and
f4, axp3 = plt.subplots(1,3)
ns=['BTC-USD']
ms=['ETH-USD','USDT-USD','BNB-USD']
i=0
for m in range(3):
    axp3[m].scatter(ddataset2[ns[0]],ddataset2[ms[i]],s=10)
    axp3[m].set_title('Bitcoin and {}'.format(ms[i]),size=10)
    i=i+1
plt.tight_layout()
plt.show()
```



In [107…
```
#The scatter plots point at the existence of a linear relationship between Bitcoin and
f4, axp3 = plt.subplots(1,3)
ns=['USDT-USD']
ms=['BTC-USD','ETH-USD','BNB-USD']
i=0
for m in range(3):
    axp3[m].scatter(ddataset2[ns[0]],ddataset2[ms[i]],s=10)
    axp3[m].set_title('USDT-USD and {}'.format(ms[i]),size=10)
    i=i+1
```

```
plt.tight_layout()
plt.show()
```



Skewness and Kurtosis

In [108…
```
#Skewness based on monthly mean data of percent returns
#sk1=scipy.stats.skew(datar_meanbtc, axis=0,bias=True)
#print (sk1)
```

In [109…
```
#kurtosis based on monthly mean data of percent returns
#ku1=scipy.stats.kurtosis(datar_meanbtc, axis=0, bias=True)
#print (ku1)
```

In [110…
```
#merging mean monthly data of returns
data_monmean=pd.concat([datar_meanbtc,datar_meaneth,datar_meanusdt,datar_meanbnb],axis
data_monmean.columns=['BTC-USD','ETH-USD','USDT-USD','BNB-USD']
data_monmean.head()
```

Out[110]:

Date	BTC-USD	ETH-USD	USDT-USD	BNB-USD
2019-01-31	-0.003197	-0.008022	-0.000385	0.002124
2019-02-28	0.004238	0.009962	0.000129	0.019148
2019-03-31	0.002098	0.001348	-0.000300	0.018145
2019-04-30	0.009478	0.005338	0.000191	0.008992
2019-05-31	0.016343	0.017629	-0.000073	0.014180

In [111…

```
#Skewness based on % monthly mean returns
skw_m1 = data_monmean.skew(axis=0)
print (skw_m1)
```

```
BTC-USD     0.030642
ETH-USD     0.044380
USDT-USD   -0.676967
BNB-USD     2.983495
dtype: float64
```

In [112…

```
#kurtosis based on % daily retruns
kurt_m1 = data_monmean.kurtosis(axis=0)
print (kurt_m1)
```

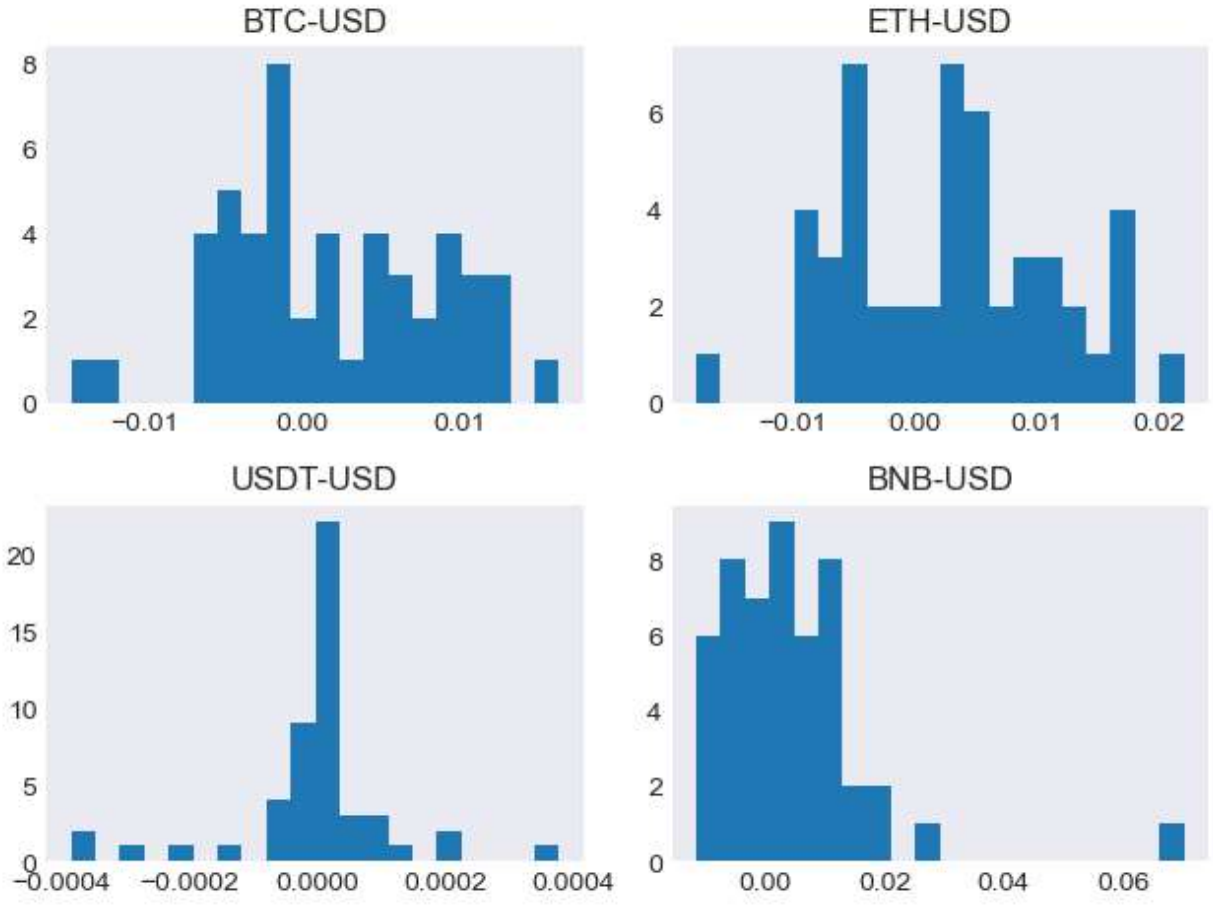
```
BTC-USD     -0.385610
ETH-USD     -0.452063
USDT-USD     4.334549
BNB-USD     14.410001
dtype: float64
```

In [ ]:

```
```

In [113…

```
#plottign distribution of montly mean data
ax3=data_monmean.hist(bins=20,grid=False)
plt.tight_layout()
plt.show()
```



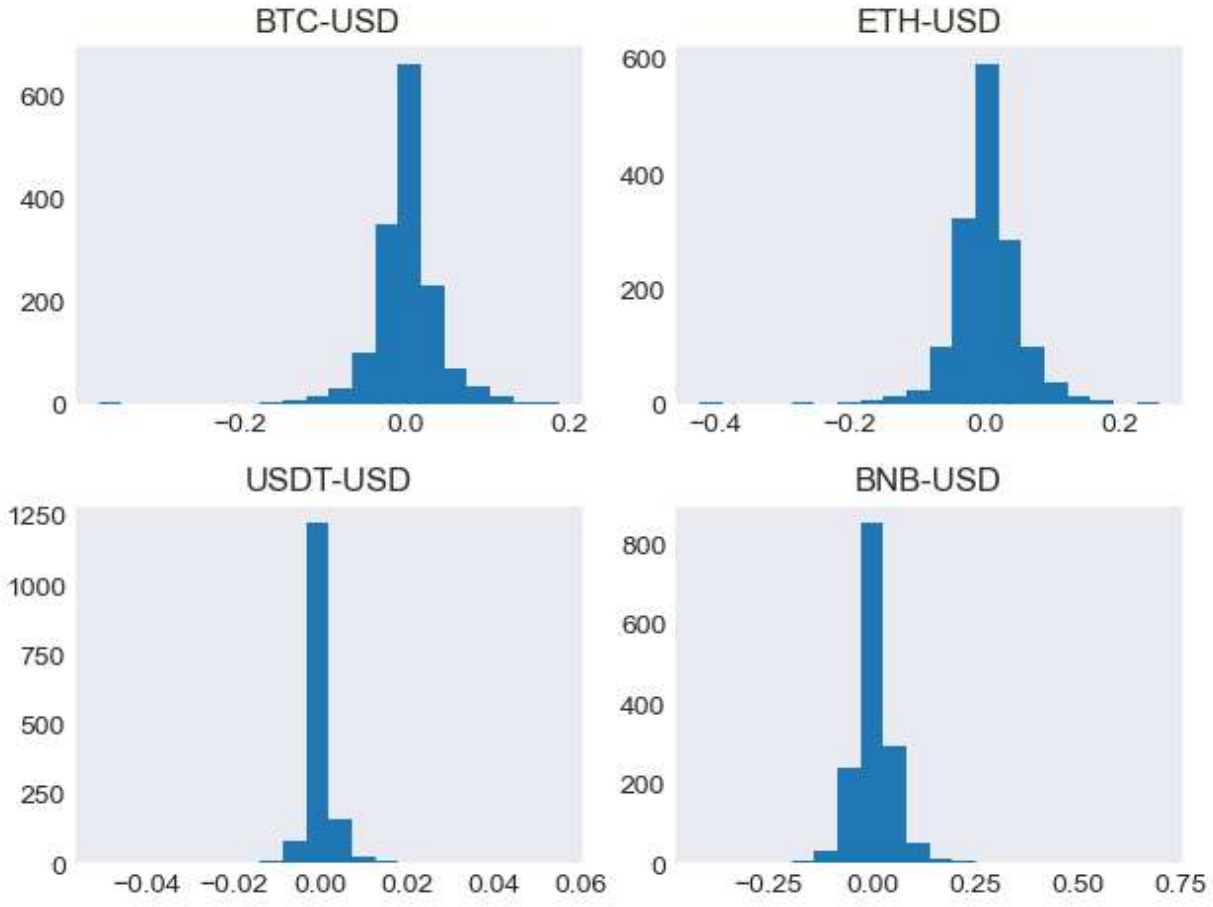
In [114…
```
#Skewness based on % daily returns
skw = ddataset2.skew(axis=0)
print (skw)
```

```
BTC-USD    -0.436793
ETH-USD    -0.396065
USDT-USD    0.803538
BNB-USD     1.485433
dtype: float64
```

In [116…
```
#kurtosis based on % daily retruns
kurt = ddataset2.kurtosis(axis=0)
print (kurt)
```

```
BTC-USD      9.549125
ETH-USD      7.100910
USDT-USD    85.419119
BNB-USD     25.926057
dtype: float64
```

In [117…
```
#plottign distribution of daily return data
ax3=ddataset2.hist(bins=20,grid=False)
plt.tight_layout()
plt.show()
```



Thanks, Vikas EOF.

In [ ]: