In [1]:
```
import numpy as np
import pandas as pd
from pandas_datareader import data as pdr
import matplotlib.pyplot as plt
import scipy.stats as sst
from scipy import stats
import statsmodels.api as sm
```

In [2]:
```
tickers=['AMZN',"TSLA",'BTC-USD','ETH-USD','QQQ','VGT']
stocks=pd.DataFrame()
for t in tickers:
    stocks[t]=pdr.DataReader(t,data_source='yahoo',start='1990-1-1')['Adj Close']
```

In [3]:
```
stock_returns=stocks/stocks.shift(1)-1
```

In [4]:
```
stock_returns=stock_returns.dropna()
stock_returns
```

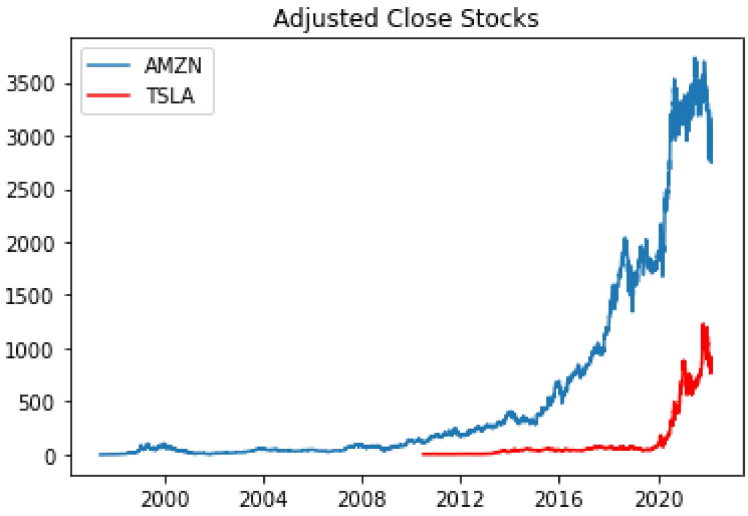
Out[4]:

Date	AMZN	TSLA	BTC-USD	ETH-USD	QQQ	VGT
2017-11-10	-0.003348	0.000000	-0.073554	-0.067411	-0.000065	0.000364
2017-11-13	0.003395	0.040958	-0.008862	0.058355	0.001236	-0.000182
2017-11-14	0.006793	-0.021243	0.011626	0.066037	-0.003639	-0.001760
2017-11-15	-0.008928	0.008422	0.102444	-0.012659	-0.004696	-0.007600
2017-11-16	0.009408	0.003855	0.076023	-0.007298	0.012780	0.013784
...	...	...	...	...	...	...
2022-03-01	-0.015765	-0.006962	0.026889	0.018253	-0.015311	-0.019237
2022-03-02	0.006024	0.017955	-0.009706	-0.007525	0.016779	0.020826
2022-03-03	-0.027320	-0.046142	-0.033520	-0.039202	-0.014285	-0.015554
2022-03-04	-0.015264	-0.001191	-0.078069	-0.076668	-0.014492	-0.019945
2022-03-07	-0.056220	-0.040213	-0.024979	-0.045075	-0.036881	-0.037497

1086 rows × 6 columns

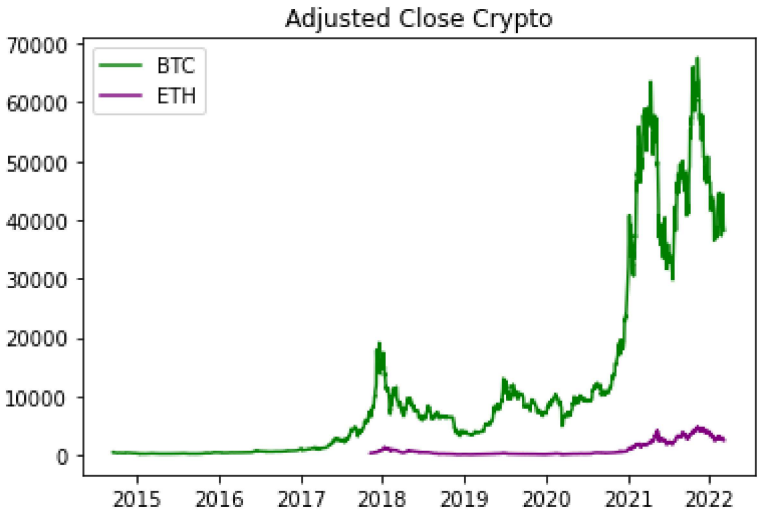
In [5]:
```
plt.plot(stocks['AMZN'],label='AMZN')
plt.plot(stocks['TSLA'],label='TSLA',color='r')
plt.legend()
plt.title('Adjusted Close Stocks')
```

Out[5]:
```
Text(0.5, 1.0, 'Adjusted Close Stocks')
```



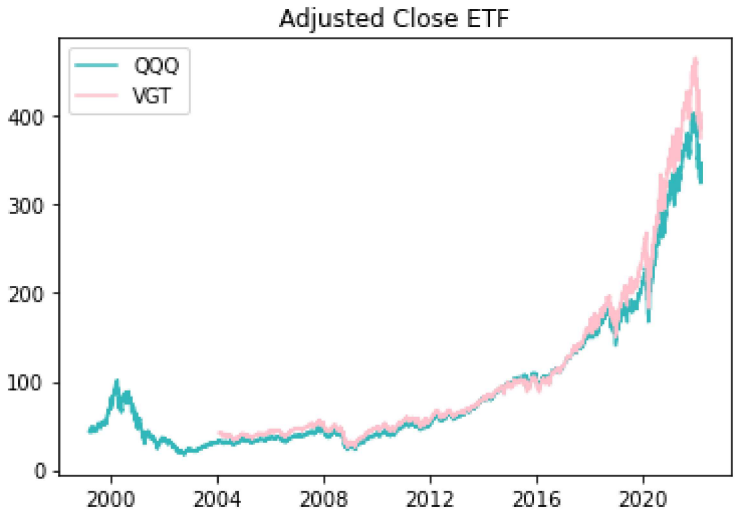
In [6]:
```
plt.plot(stocks['BTC-USD'],label='BTC',color='g')
plt.plot(stocks['ETH-USD'],label='ETH',color='purple')
plt.legend()
plt.title('Adjusted Close Crypto')
```

Out[6]:
```
Text(0.5, 1.0, 'Adjusted Close Crypto')
```



In [7]:
```
plt.plot(stocks['QQQ'],label='QQQ',color='#32b8ba')
plt.plot(stocks['VGT'],label='VGT',color='pink')
plt.legend()
plt.title('Adjusted Close ETF')
```

Out[7]:
```
Text(0.5, 1.0, 'Adjusted Close ETF')
```



Daily Average Returns

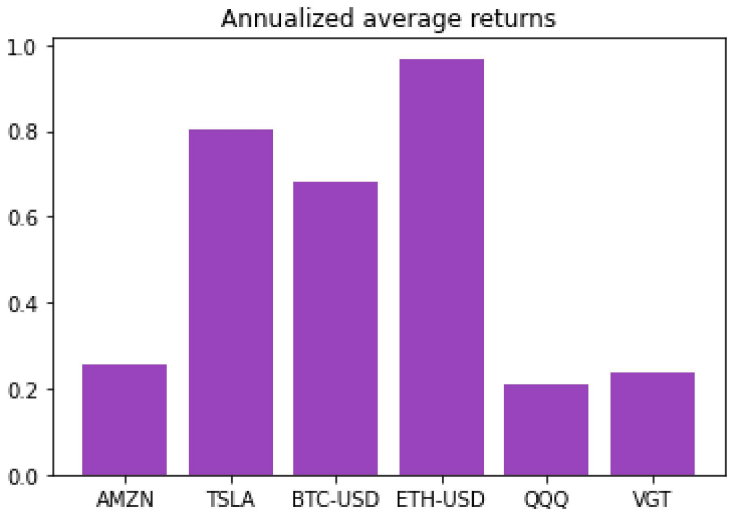
In [8]:
```
AMZN_mean=stock_returns['AMZN'].mean()
TSLA_mean=stock_returns['TSLA'].mean()
BTC_mean=stock_returns['BTC-USD'].mean()
ETH_mean=stock_returns['ETH-USD'].mean()
QQQ_mean=stock_returns['QQQ'].mean()
VGT_mean=stock_returns['VGT'].mean()
```

Yearly Avg Returns

In [9]:
```
yearly_returns=stock_returns.mean()*250
yearly_returns
```

Out[9]:
```
AMZN       0.255417
TSLA       0.802216
BTC-USD    0.682861
ETH-USD    0.967362
QQQ        0.208824
VGT        0.234994
dtype: float64
```

In [10]:
```
plt.bar(tickers,yearly_returns,color='#9944bb')
plt.title('Annualized average returns')
plt.show()
```



Risk Levels

In [11]:
```
AMZN_std=stock_returns['AMZN'].std()
TSLA_std=stock_returns['TSLA'].std()
BTC_std=stock_returns['BTC-USD'].std()
ETH_std=stock_returns['ETH-USD'].std()
QQQ_std=stock_returns['QQQ'].std()
VGT_std=stock_returns['VGT'].std()
```

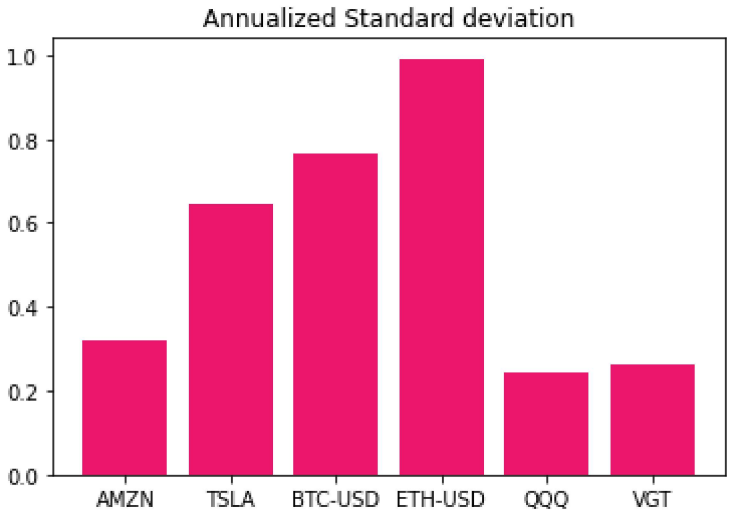
In [12]:
```
ETH_std
```

Out[12]:
```
0.06270419109103884
```

In [13]:
```
stocks_std_yearly=stock_returns.std()*250**0.5
stocks_std_yearly
```

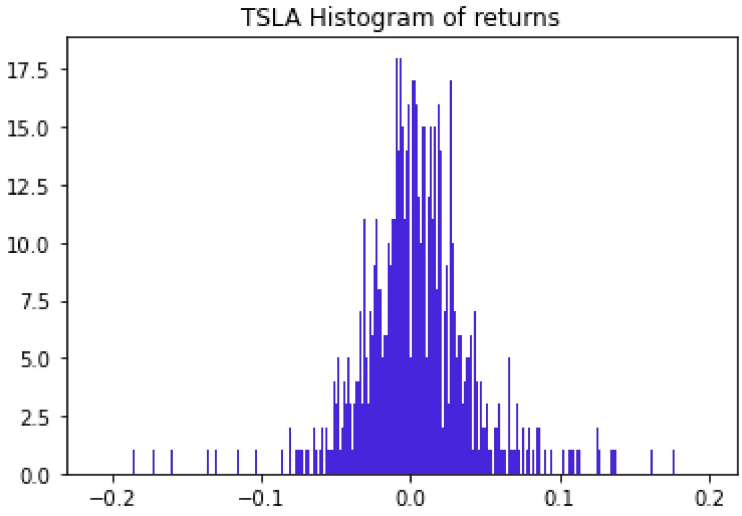
Out[13]:
```
AMZN       0.318300
TSLA       0.644139
BTC-USD    0.766715
ETH-USD    0.991440
QQQ        0.243320
VGT        0.263732
dtype: float64
```

In [14]:
```
plt.bar(tickers,stocks_std_yearly,color='#e9166b')
plt.title('Annualized Standard deviation')
plt.show()
```

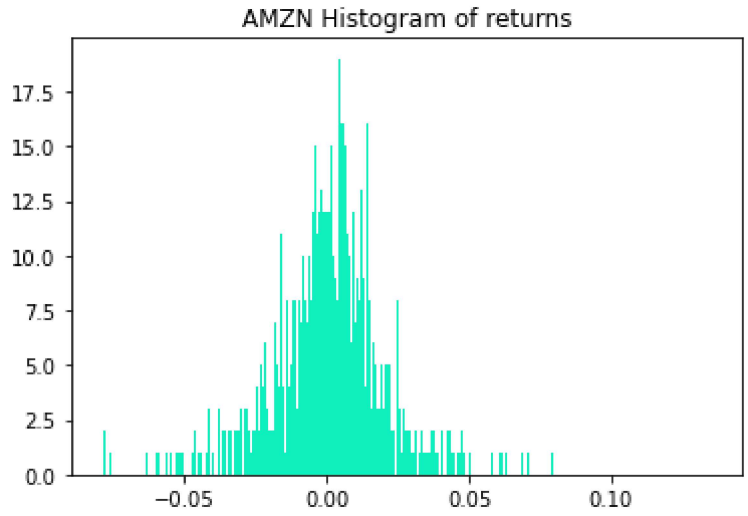


Histogram comparison

In [15]:
```
H1=stock_returns['TSLA']
plt.hist(H1,bins=500,color='#4625da')
plt.title('TSLA Histogram of returns')
plt.show()
```

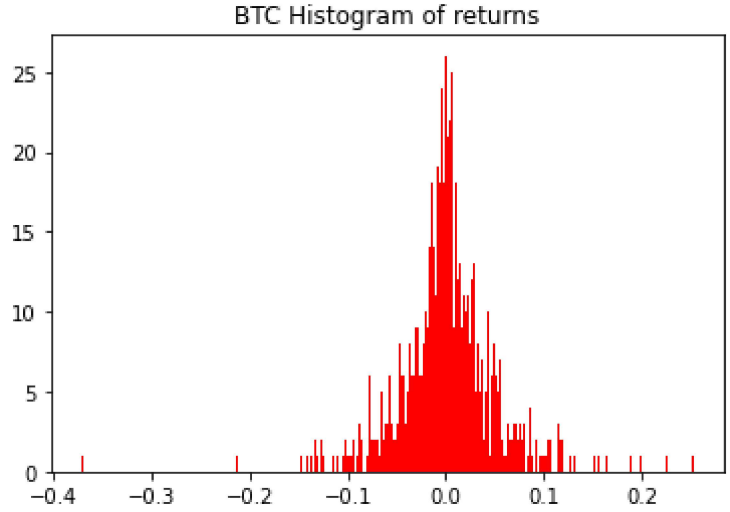


In [16]:
```
H4=stock_returns['AMZN']
plt.hist(H4,bins=500,color='#0ef1be')
plt.title('AMZN Histogram of returns')
plt.show()
```



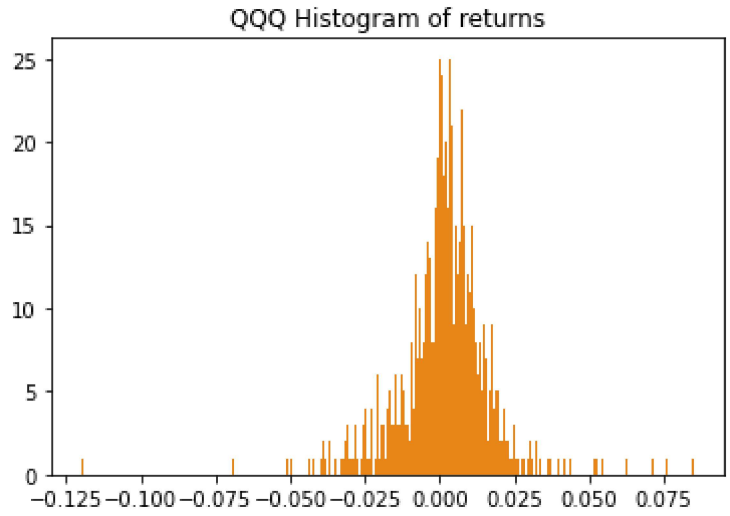
In [17]:

```
H2=stock_returns['BTC-USD']
plt.hist(H2,bins=500,color='red')
plt.title('BTC Histogram of returns')
plt.show()
```



In [18]:

```
H3=stock_returns['QQQ']
plt.hist(H3,bins=500,color='#e78518')
plt.title('QQQ Histogram of returns')
plt.show()
```



Sharpe Ratio

In [19]:
```
SR_AMZN=((AMZN_mean*250)-0.03)/(AMZN_std*250**0.5)
SR_AMZN
```

Out[19]:
```
0.708192024270355
```

In [20]:
```
SR_TSLA=((TSLA_mean*250)-0.03)/(TSLA_std*250**0.5)
SR_TSLA
```

Out[20]:
```
1.1988342866344264
```

In [21]:
```
SR_BTC=((BTC_mean*250)-0.03)/(BTC_std*250**0.5)
SR_BTC
```

Out[21]:
```
0.8515044731600592
```

In [22]:
```
SR_ETH=((ETH_mean*250)-0.03)/(ETH_std*250**0.5)
SR_ETH
```

Out[22]:
```
0.9454550019935622
```

In [23]:
```
SR_QQQ=((QQQ_mean*250)-0.03)/(QQQ_std*250**0.5)
SR_QQQ
```

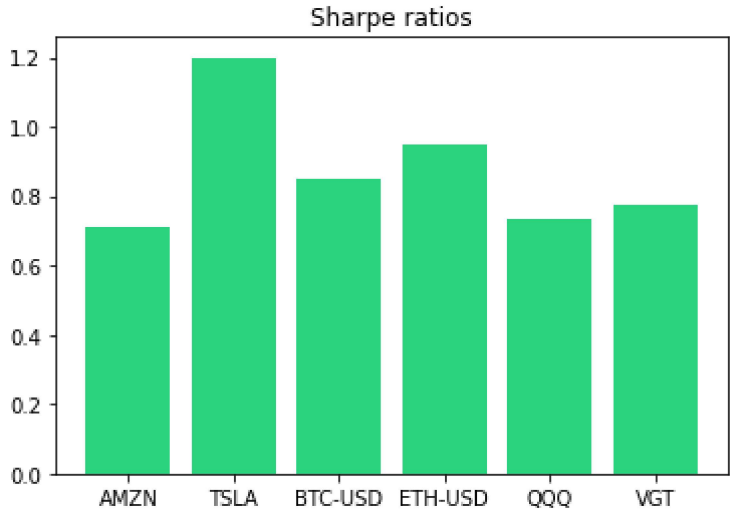
Out[23]:
```
0.7349333172873931
```

In [24]:
```
SR_VGT=((VGT_mean*250)-0.03)/(VGT_std*250**0.5)
SR_VGT
```

Out[24]:
```
0.7772792567377342
```

In [25]:
```
sharpe_ratios=[SR_AMZN,SR_TSLA,SR_BTC,SR_ETH,SR_QQQ,SR_VGT]
```

In [26]:
```
plt.bar(tickers,sharpe_ratios,color='#2cd37e')
plt.title('Sharpe ratios')
plt.show()
```



Forming Portfolios

Stock Portfolio (Neutral Risk Investor)

In [27]:
```
returns1=stock_returns[['AMZN','TSLA']]
```

In [28]:
```
weights=np.array([0.5,0.5])
port_ret1=np.dot(returns1,weights)
port_ret1=port_ret1[~np.isnan(port_ret1)]
port_ret1_mean=port_ret1.mean()
port_ret1_mean_annual=port_ret1_mean*250
port_ret1_mean_annual
```

Out[28]:
```
0.5288167083314586
```

In [29]:
```
port_var1=port_ret1.var()*250
port_std1=port_var1**0.5
port_std1
```

Out[29]:
```
0.4109642827455384
```

In [30]:
```
port1_SR=(port_ret1_mean_annual-0.03)/(port_std1)
port1_SR
```

Out[30]:
```
1.213771437748805
```

Stock Portfolio(Risk Averse Investor)

In [31]:
```
weights2=np.array([0.8,0.20])
port_ret2=np.dot(returns1,weights2)
port_ret2=port_ret2[~np.isnan(port_ret2)]
port_ret2_mean=port_ret2.mean()
port_ret2_mean_annual=port_ret2_mean*250
port_ret2_mean_annual
```

Out[31]:
```
0.3647770148816826
```

In [32]:
```
port_var2=port_ret2.var()*250
port_std2=port_var2**0.5
port_std2
```

Out[32]:
```
0.32700595175742064
```

In [33]:
```
port2_SR=(port_ret2_mean_annual-0.03)/(port_std2)
port2_SR
```

Out[33]:
```
1.023764286498451
```

Stock Portfolio (Risk Inclined Investor)

In [34]:
```
weights3=np.array([0.2,0.80])
port_ret3=np.dot(returns1,weights3)
port_ret3=port_ret3[~np.isnan(port_ret3)]
port_ret3_mean=port_ret3.mean()
port_ret3_mean_annual=port_ret3_mean*250
port_ret3_mean_annual
```

Out[34]:
```
0.6928564017812344
```

In [35]:
```
port_var3=port_ret3.var()*250
port_std3=port_var3**0.5
port_std3
```

Out[35]:
```
0.5430654398177384
```

In [36]:
```
port3_SR=(port_ret3_mean_annual-0.03)/(port_std3)
port3_SR
```

Out[36]:
```
1.2205829227573382
```

Crypto Portfolio (Risk Neutral Investor)

In [37]:
```
returns2=stock_returns[['BTC-USD','ETH-USD']]
```

In [38]:
```
weights4=np.array([0.5,0.5])
port_ret4=np.dot(returns2,weights4)
```

```
port_ret4=port_ret4[~np.isnan(port_ret4)]
port_ret4_mean=port_ret4.mean()
port_ret4_mean_annual=port_ret4_mean*250
port_ret4_mean_annual
```

Out[38]: 0.8251115511494322

In [39]:
```
port_var4=port_ret4.var()*250
port_std4=port_var4**0.5
port_std4
```

Out[39]: 0.8184564502637282

In [40]:
```
port4_SR=(port_ret4_mean_annual-0.03)/(port_std4)
port4_SR
```

Out[40]: 0.9714769196250166

Crypto Portfolio (Risk Averse Investor)

In [41]:
```
weights5=np.array([0.8,0.20])
port_ret5=np.dot(returns2,weights5)
port_ret5=port_ret5[~np.isnan(port_ret5)]
port_ret5_mean=port_ret5.mean()
port_ret5_mean_annual=port_ret5_mean*250
port_ret5_mean_annual
```

Out[41]: 0.7397611597324719

In [42]:
```
port_var5=port_ret5.var()*250
port_std5=port_var5**0.5
port_std5
```

Out[42]: 0.7699222947120113

In [43]:
```
port5_SR=(port_ret5_mean_annual-0.03)/(port_std5)
port5_SR
```

Out[43]: 0.9218607703754796

Crypto Portfolio (Risk Incline Investor)

In [44]:
```
weights6=np.array([0.2,0.80])
port_ret6=np.dot(returns2,weights6)
port_ret6=port_ret6[~np.isnan(port_ret6)]
port_ret6_mean=port_ret6.mean()
port_ret6_mean_annual=port_ret6_mean*250
port_ret6_mean_annual
```

Out[44]: 0.9104619425663928

In [45]:
```
port_var6=port_ret6.var()*250
port_std6=port_var6**0.5
port_std6
```

Out[45]:
```
0.9108367775096501
```

In [46]:
```
port6_SR=(port_ret6_mean_annual-0.03)/(port_std6)
port6_SR
```

Out[46]:
```
0.9666517254317439
```

ETF Portfolio (Risk Neutral)

In [47]:
```
returns3=stock_returns[['QQQ','VGT']]
```

In [48]:
```
weights7=np.array([0.5,0.5])
port_ret7=np.dot(returns3,weights7)
port_ret7=port_ret7[~np.isnan(port_ret7)]
port_ret7_mean=port_ret7.mean()
port_ret7_mean_annual=port_ret7_mean*250
port_ret7_mean_annual
```

Out[48]:
```
0.22190865781179245
```

In [49]:
```
port_var7=port_ret7.var()*250
port_std7=port_var7**0.5
port_std7
```

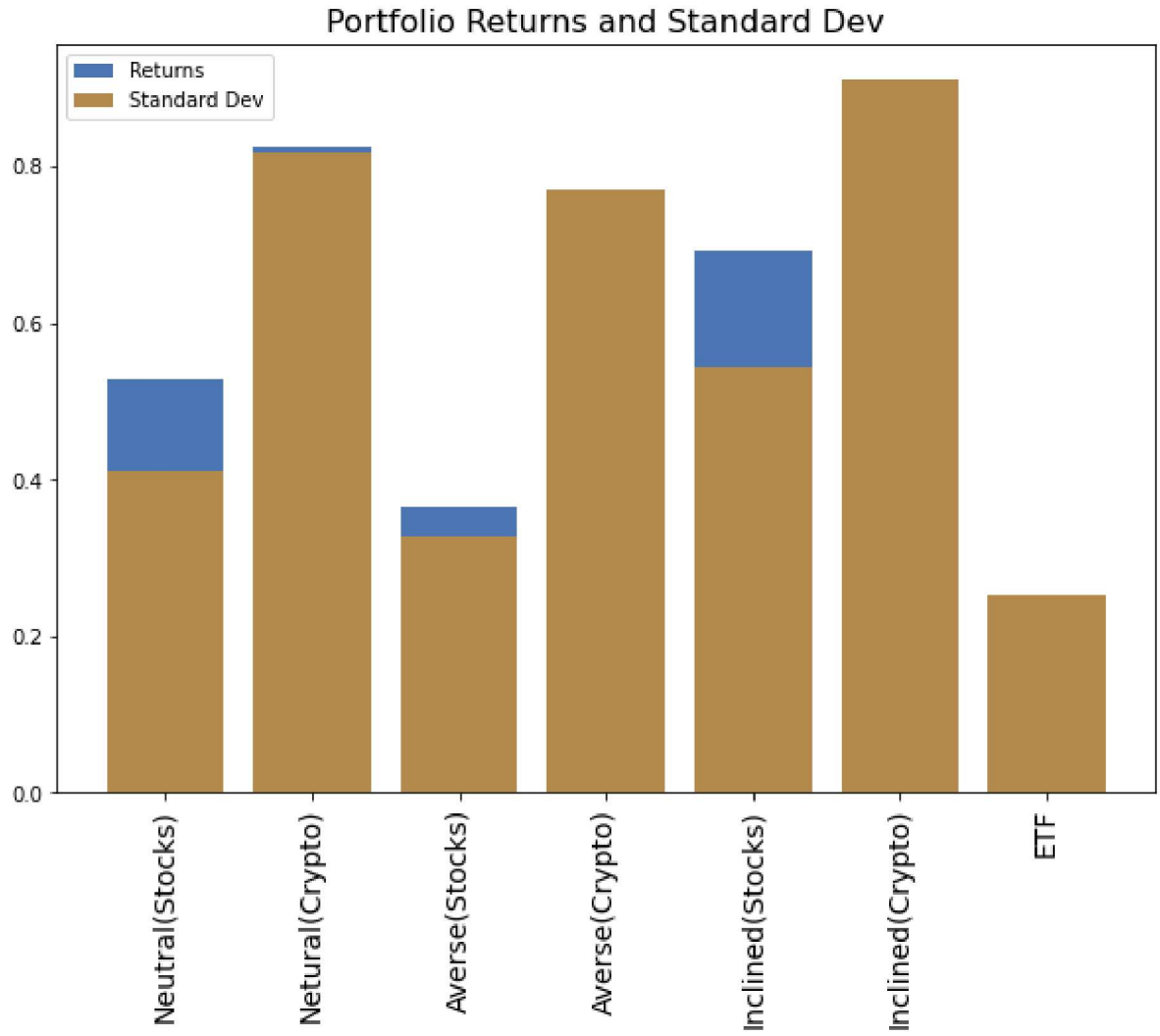
Out[49]:
```
0.2520947113881759
```

In [50]:
```
port7_SR=(port_ret7_mean_annual-0.03)/(port_std7)
port7_SR
```

Out[50]:
```
0.7612561832615804
```

Visualizing Risk and Return for Portfolios

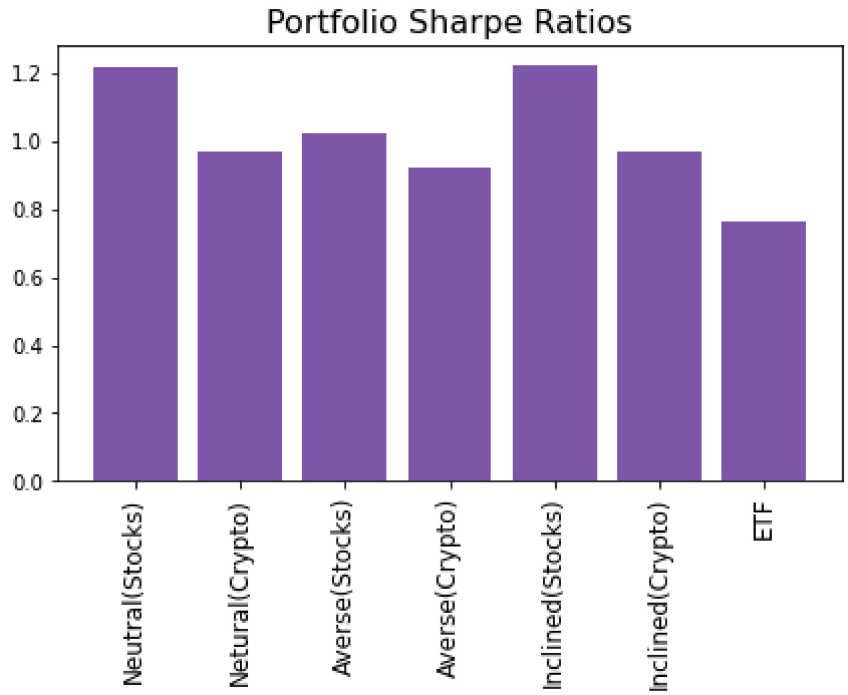
In [51]:
```
labels=['Neutral(Stocks)','Netural(Crypto)','Averse(Stocks)','Averse(Crypto)','Inclined
legend=['Returns','Standard Dev']
portfolio_returns=[port_ret1_mean_annual,port_ret4_mean_annual,port_ret2_mean_annual,po
portfolio_std=[port_std1,port_std4,port_std2,port_std5,port_std3,port_std6,port_std7]
plt.figure(figsize=(10, 7))
plt.title('Portfolio Returns and Standard Dev',fontsize=16)
plt.xticks(rotation='vertical',fontsize=14)
plt.bar(labels,portfolio_returns,color='#4c76b3')
plt.bar(labels,portfolio_std,color='#B3894C')
plt.legend(legend)
plt.show()
```



Visualizing Portfolio Sharpe Ratios

In [52]:
```
portfolio_SR=[port1_SR,port4_SR,port2_SR,port5_SR,port3_SR,port6_SR,port7_SR]
plt.figure(figsize=(7, 4))
plt.title('Portfolio Sharpe Ratios',fontsize=16)
plt.xticks(rotation='vertical',fontsize=12)
plt.bar(labels,portfolio_SR,color='#7d56a9')
```

Out[52]:
```
<BarContainer object of 7 artists>
```



Effect of Diversification

In [53]:
```
returns4=stock_returns[['AMZN','ETH-USD','QQQ']]
```

In [54]:
```
weights8=np.array([0.33,0.34,0.33])
port_ret8=np.dot(returns4,weights8)
port_ret8=port_ret8[~np.isnan(port_ret8)]
port_ret8_mean=port_ret8.mean()
port_ret8_mean_annual=port_ret8_mean*250
port_ret8_mean_annual
```

Out[54]:
```
0.48210264195483493
```

In [55]:
```
port_var8=port_ret8.var()*250
port_std8=port_var8**0.5
port_std8
```

Out[55]:
```
0.410207732698418
```

In [56]:
```
port8_SR=(port_ret8_mean_annual-0.03)/(port_std8)
port8_SR
```

Out[56]:
```
1.1021309593088968
```

In [57]:
```
weights9=np.array([0.25,0.50,0.25])
port_ret9=np.dot(returns4,weights9)
port_ret9=port_ret9[~np.isnan(port_ret9)]
port_ret9_mean=port_ret9.mean()
```

```
port_ret9_mean_annual=port_ret9_mean*250
port_ret9_mean_annual
```

Out[57]:
```
0.5997413235442163
```

In [58]:
```
port_var9=port_ret9.var()*250
port_std9=port_var9**0.5
port_std9
```

Out[58]:
```
0.538439415679639
```

In [59]:
```
port9_SR=(port_ret9_mean_annual-0.03)/(port_std9)
port9_SR
```

Out[59]:
```
1.0581345030713747
```

In [60]:
```
# Simple Calculator I used to test out different weights
x=float(input('Enter value of weight on Amazon: '))
y=float(input('Enter value of weight on ETH: '))
z=float(input('Enter value of weight on QQQ: '))
weights10=np.array([x,y,z])
port_ret10=np.dot(returns4,weights10)
port_ret10=port_ret10[~np.isnan(port_ret10)]
port_ret10_mean=port_ret10.mean()
port_ret10_mean_annual=port_ret10_mean*250
port_var10=port_ret10.var()*250
port_std10=port_var10**0.5
port10_SR=(port_ret10_mean_annual-0.03)/(port_std10)
print(f'portfolio return={port_ret10_mean_annual}')
print(f'portfolio Standard Dev={port_std10}')
print(f'portfolio sharpe ratio={port10_SR}')
```

```
Enter value of weight on Amazon: 0
Enter value of weight on ETH: 0
Enter value of weight on QQQ: 0
portfolio return=0.0
portfolio Standard Dev=0.0
portfolio sharpe ratio=-inf
C:\Users\jovan\AppData\Local\Temp/ipykernel_2980/835865667.py:12: RuntimeWarning: divide by zero encountered in double_scalars
  port10_SR=(port_ret10_mean_annual-0.03)/(port_std10)
```

In [61]:
```
returns5=stock_returns[['TSLA','BTC-USD','VGT']]
```

In [62]:
```
weights11=np.array([0.50,0.25,0.25])
port_ret11=np.dot(returns5,weights11)
port_ret11=port_ret11[~np.isnan(port_ret11)]
port_ret11_mean=port_ret11.mean()
port_ret11_mean_annual=port_ret11_mean*250
port_ret11_mean_annual
```

Out[62]:
```
0.6305717353334881
```

In [63]:
```
port_var11=port_ret11.var()*250
port_std11=port_var11**0.5
port_std11
```

Out[63]:
```
0.43256245256436376
```

In [64]:
```
port11_SR=(port_ret11_mean_annual-0.03)/(port_std11) #Diversification increased Sharpe
port11_SR
```

Out[64]:
```
1.3884046841632078
```

In [65]:
```
a=float(input('Enter value of weight on Tesla: '))
b=float(input('Enter value of weight on BTC: '))
c=float(input('Enter value of weight on VGT: '))
weights12=np.array([a,b,c])
port_ret12=np.dot(returns5,weights12)
port_ret12=port_ret12[~np.isnan(port_ret12)]
port_ret12_mean=port_ret12.mean()
port_ret12_mean_annual=port_ret12_mean*250
port_var12=port_ret12.var()*250
port_std12=port_var12**0.5
port12_SR=(port_ret12_mean_annual-0.03)/(port_std12)
print(f'portfolio return={port_ret12_mean_annual}')
print(f'portfolio Standard Dev={port_std12}')
print(f'portfolio sharpe ratio={port12_SR}')
```

```
Enter value of weight on Tesla: 0
Enter value of weight on BTC: 0
Enter value of weight on VGT: 0
portfolio return=0.0
portfolio Standard Dev=0.0
portfolio sharpe ratio=-inf
C:\Users\jovan\AppData\Local\Temp/ipykernel_2980/3047388515.py:11: RuntimeWarning: divid
e by zero encountered in double_scalars
  port12_SR=(port_ret12_mean_annual-0.03)/(port_std12)
```

In [66]:
```
# Even for very risk averse individuals, diversification increases sharpe ratio a decen
```

In [ ]:
```

```

In [67]:
```
# Here i can test out alot of different combinations
x1=float(input('Enter value of weight on Amazon: '))
x2=float(input('Enter value of weight on Tesla: '))
x3=float(input('Enter value of weight on BTC: '))
x4=float(input('Enter value of weight on ETH: '))
x5=float(input('Enter value of weight on QQQ: '))
x6=float(input('Enter value of weight on VGT: '))
weights13=np.array([x1,x2,x3,x4,x5,x6])
port_ret13=np.dot(stock_returns,weights13)
port_ret13=port_ret13[~np.isnan(port_ret13)]
port_ret13_mean=port_ret13.mean()
port_ret13_mean_annual=port_ret13_mean*250
port_var13=port_ret13.var()*250
```

```
port_std13=port_var13**0.5
port13_SR=(port_ret13_mean_annual-0.03)/(port_std13)
print(f'portfolio return={port_ret13_mean_annual}')
print(f'portfolio Standard Dev={port_std13}')
print(f'portfolio sharpe ratio={port13_SR}')
```

```
Enter value of weight on Amazon: 0
Enter value of weight on Tesla: 0
Enter value of weight on BTC: 0
Enter value of weight on ETH: 0
Enter value of weight on QQQ: 0
Enter value of weight on VGT: 0
portfolio return=0.0
portfolio Standard Dev=0.0
portfolio sharpe ratio=-inf
```

```
C:\Users\jovan\AppData\Local\Temp/ipykernel_2980/2864828409.py:15: RuntimeWarning: divid
e by zero encountered in double_scalars
  port13_SR=(port_ret13_mean_annual-0.03)/(port_std13)
```

In [68]:

```
weights14=np.array([0.10,0.50,0.10,0.10,0.10,0.10])
port_ret14=np.dot(stock_returns,weights14)
port_ret14=port_ret14[~np.isnan(port_ret14)]
port_ret14_mean=port_ret14.mean()
port_ret14_mean_annual=port_ret14_mean*250
port_var14=port_ret14.var()*250
port_std14=port_var14**0.5
port14_SR=(port_ret14_mean_annual-0.03)/(port_std14)
print(f'portfolio return={port_ret14_mean_annual}')
print(f'portfolio Standard Dev={port_std14}')
print(f'portfolio sharpe ratio={port14_SR}')
```

```
portfolio return=0.636053862424304
portfolio Standard Dev=0.4270010834440392
portfolio sharpe ratio=1.4193262872686143
```

In [69]:

```
weights15=np.array([0.10,0.10,0.10,0.50,0.10,0.10])
port_ret15=np.dot(stock_returns,weights15)
port_ret15=port_ret15[~np.isnan(port_ret15)]
port_ret15_mean=port_ret15.mean()
port_ret15_mean_annual=port_ret15_mean*250
port_var15=port_ret15.var()*250
port_std15=port_var15**0.5
port15_SR=(port_ret15_mean_annual-0.03)/(port_std15)
print(f'portfolio return={port_ret15_mean_annual}')
print(f'portfolio Standard Dev={port_std15}')
print(f'portfolio sharpe ratio={port15_SR}')
```

```
portfolio return=0.7021122648629499
portfolio Standard Dev=0.5929777026298619
portfolio sharpe ratio=1.1334528463416504
```

Hypothesis Testing

F-test and 2 sample T-test for Tesla

In [70]:

```
sst.ttest_ind(a=stock_returns['TSLA'],b=port_ret14,equal_var=False) #Fail to reject
```

Out[70]:
```
Ttest_indResult(statistic=0.44806399131456015, pvalue=0.6541585133227834)
```

In [71]:
```
F_stat=max(stock_returns['TSLA'].var()/port_ret14.var(),port_ret14.var()/stock_returns[
F_stat
```

Out[71]:
```
2.2756294688604433
```

In [72]:
```
crit_value=sst.f.ppf(0.95,dfn=stock_returns['TSLA'].count()-1,dfd=6426-1)
crit_value
```

Out[72]:
```
1.078073591885743
```

In [73]:
```
p_value=1-sst.f.cdf(F_stat,dfn=stock_returns['TSLA'].count()-1,dfd=6426-1) #Reject Null
p_value
```

Out[73]:
```
1.1102230246251565e-16
```

In [74]:
```
sst.ttest_ind(a=port_ret3,b=port_ret14,equal_var=False)
```

Out[74]:
```
Ttest_indResult(statistic=0.17129301308571881, pvalue=0.8640102334680422)
```

In [75]:
```
F_stat4=max(port_ret3.var()/port_ret14.var(),port_ret14.var()/port_ret3.var()) #with ju
F_stat4
```

Out[75]:
```
1.6175077760824315
```

In [76]:
```
crit_value4=sst.f.ppf(0.95,dfn=2142-1,dfd=6426-1)
crit_value4
```

Out[76]:
```
1.0592589822435832
```

In [77]:
```
F_stat5=max(port_ret11.var()/port_ret14.var(),port_ret14.var()/port_ret11.var())
F_stat5
```

Out[77]:
```
1.0262181323182957
```

In [78]:
```
crit_value5=sst.f.ppf(0.95,dfn=3213-1,dfd=6426-1)
crit_value5
```

Out[78]:
```
1.051302117637701
```

In [79]:
```
p_value5=1-sst.f.cdf(F_stat5,dfn=3213-1,dfd=6426-1)
p_value5
```

Out[79]:
```
0.19727016975089096
```

F Test and 2 sample t-test for ETH

In [80]:
```
sst.ttest_ind(a=stock_returns['ETH-USD'],b=port_ret15,equal_var=False) # Fail to reject
```

Out[80]:
```
Ttest_indResult(statistic=0.4784931756269527, pvalue=0.6323582046213813)
```

In [81]:
```
F_stat6=max(stock_returns['ETH-USD'].var()/port_ret15.var(),port_ret15.var()/stock_retu
F_stat6
```

Out[81]:
```
2.7954802069330795
```

In [82]:
```
crit_value6=sst.f.ppf(0.95,dfn=stock_returns['ETH-USD'].count()-1,dfd=6426-1)
crit_value6
```

Out[82]:
```
1.078073591885743
```

In [83]:
```
F_stat7=max(port_ret6.var()/port_ret15.var(),port_ret15.var()/port_ret6.var())
F_stat7
```

Out[83]:
```
2.3594152921687144
```

In [84]:
```
crit_value7=sst.f.ppf(0.95,dfn=2142-1,dfd=6426-1)
crit_value7
```

Out[84]:
```
1.0592589822435832
```

In [85]:
```
p_value7=1-sst.f.cdf(F_stat7,dfn=3213-1,dfd=6426-1)
p_value7
```

Out[85]:
```
1.1102230246251565e-16
```

In [86]:
```
F_stat8=max(port_ret9.var()/port_ret15.var(),port_ret15.var()/port_ret9.var())
F_stat8
```

Out[86]:
```
1.2128386763498795
```

In [87]:
```
crit_value8=sst.f.ppf(0.95,dfn=3213-1,dfd=6426-1)
crit_value8
```

Out[87]:
```
1.051302117637701
```

In [88]:
```
p_value8=1-sst.f.cdf(F_stat8,dfn=3213-1,dfd=6426-1)
p_value8
```

Out[88]:
```
8.79163408740169e-11
```

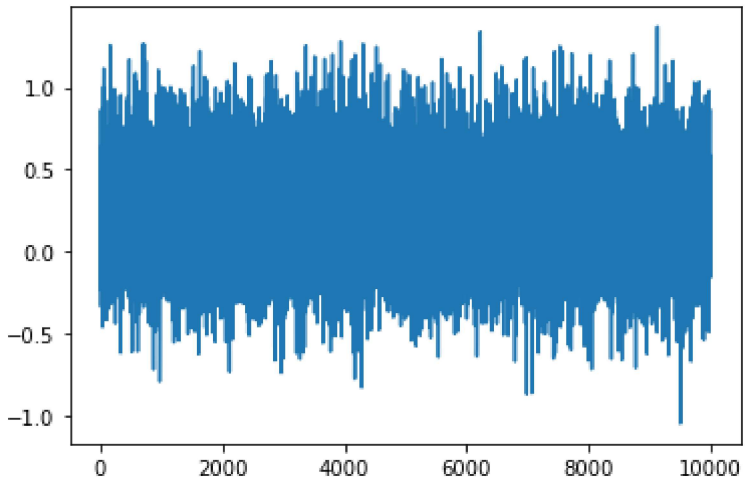
Final Project

In [89]:
```
amz_m=AMZN_mean*250
amz_stdev=AMZN_std*250**0.5
iteration=10000
```

In [90]:
```
amz=np.random.normal(amz_m,amz_stdev,iteration)
amz
```

Out[90]:
```
array([0.22594958, 0.13726951, 0.32902293, ..., 0.34164924, 0.07339173,
       0.20112771])
```

In [91]:
```
plt.plot(amz)
plt.show()
```



In [92]:
```
hist=np.histogram(amz,bins=100)
hist_dist=sst.rv_histogram(hist)
1-hist_dist.cdf(0.5)
```

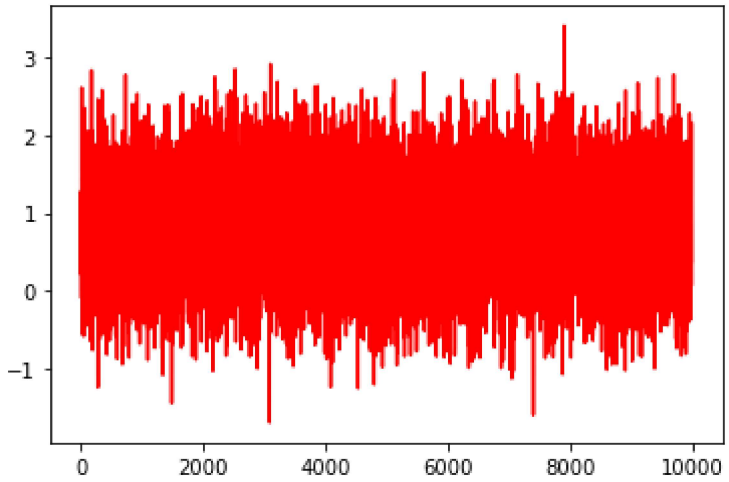
Out[92]:
```
0.2266665128995845
```

In [93]:
```
tsla_m=TSLA_mean*250
tsla_stdev=TSLA_std*250**0.5
iteration=10000
```

In [94]:
```
tsla=np.random.normal(tsla_m,tsla_stdev,iteration)
tsla
```

Out[94]:
```
array([1.03815369, 0.6340503 , 0.97964431, ..., 0.50107381, 1.38272136,
       0.08452843])
```

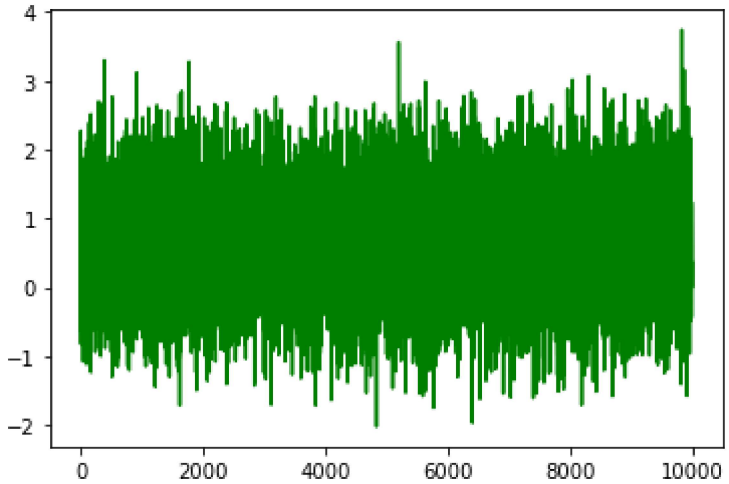
In [95]:
```
plt.plot(tsla,color='r')
plt.show()
```



In [96]:
```
hist2=np.histogram(tsla,bins=100)
hist2_dist=sst.rv_histogram(hist2)
1-hist2_dist.cdf(0.5)
```

Out[96]:
```
0.6814943113803358
```

In [97]:
```
BTC_m=BTC_mean*250
BTC_stdev=BTC_std*250**0.5
iteration=10000
BTC=np.random.normal(BTC_m,BTC_stdev,iteration)
plt.plot(BTC,color='g')
plt.show()
```

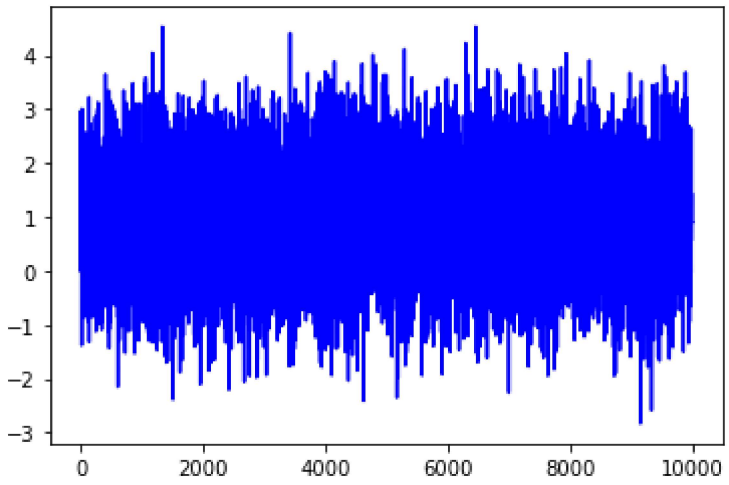


In [98]:
```
hist3=np.histogram(BTC,bins=100)
hist3_dist=sst.rv_histogram(hist3)
1-hist3_dist.cdf(0.5)
```

Out[98]:
```
0.588752410649653
```

In [99]:
```
ETH_m=ETH_mean*250
ETH_stdev=ETH_std*250**0.5
iteration=10000
ETH=np.random.normal(ETH_m,ETH_stdev,iteration)
```

```
plt.plot(ETH,color='b')
plt.show()
```



In [100…

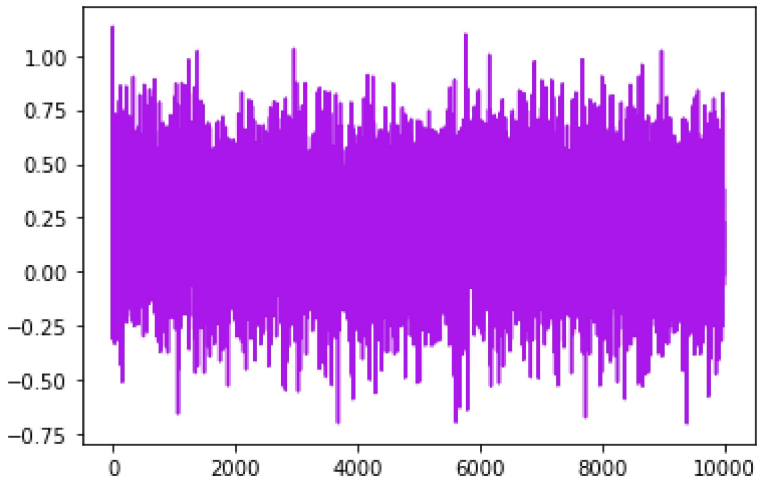
```
hist4=np.histogram(ETH,bins=100)
hist4_dist=sst.rv_histogram(hist4)
1-hist4_dist.cdf(0.5)
```

Out[100…

```
0.6794705379570987
```

In [101…

```
QQQ_m=QQQ_mean*250
QQQ_stdev=QQQ_std*250**0.5
iteration=10000
QQQ=np.random.normal(QQQ_m,QQQ_stdev,iteration)
plt.plot(QQQ,color='#a817e8')
plt.show()
```



In [102…

```
hist5=np.histogram(QQQ,bins=100)
hist5_dist=sst.rv_histogram(hist5)
1-hist5_dist.cdf(0.5)
```

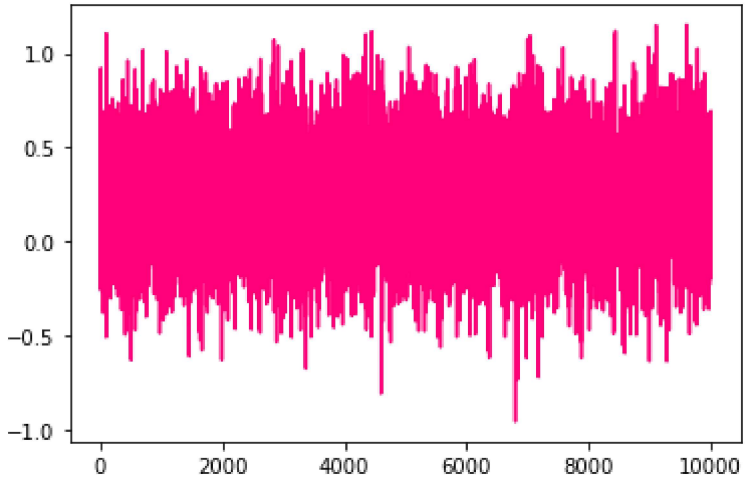
Out[102…

```
0.11563183871683924
```

In [103…

```
VGT_m=VGT_mean*250
```

```
VGT_stdev=VGT_std*250**0.5
iteration=10000
VGT=np.random.normal(VGT_m,VGT_stdev,iteration)
plt.plot(VGT,color='#ff0079')
plt.show()
```



In [104…

```
hist6=np.histogram(VGT,bins=100)
hist6_dist=sst.rv_histogram(hist6)
1-hist6_dist.cdf(0.5)
```

Out[104… 0.1564860187846313

In [138…

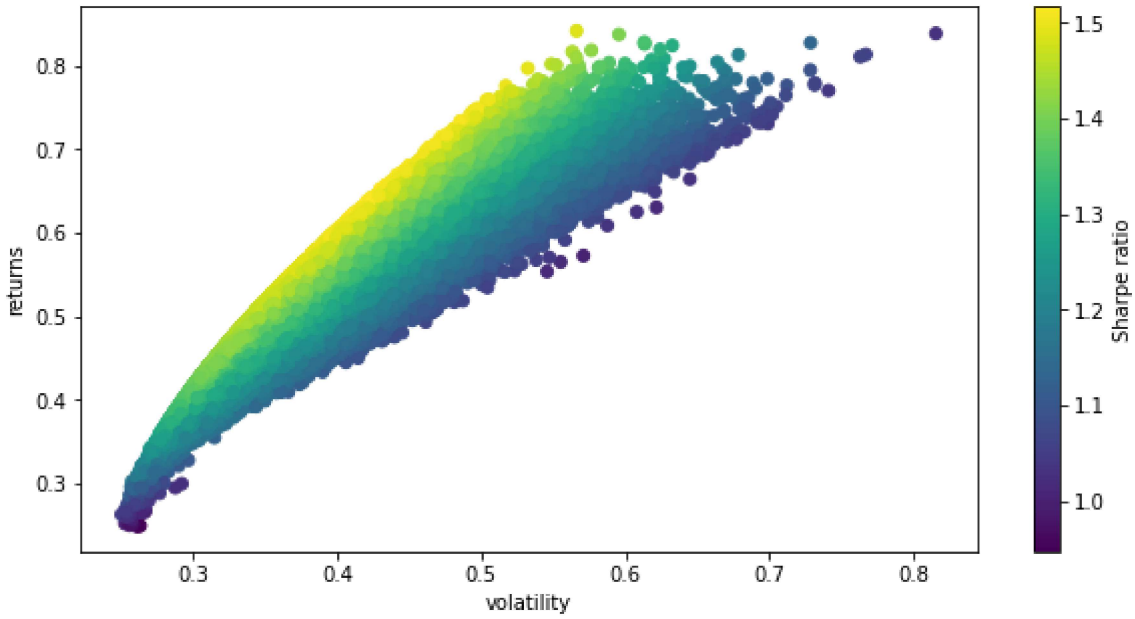
```
portfolios= 100000
RF = 0 #assuming rf is 0 to make it easier, wont change the final outcome regardless mu

portfolio_returns = []
portfolio_risk = []
sharpe_ratio_port = []
portfolio_weights = []

for portfolio in range (portfolios):
    weights = np.random.random_sample((len(tickers)))
    weights = weights / np.sum(weights) #This is to make sure the sum of weights is 1
    annualize_return = np.sum((stock_returns.mean() * weights) * 250)
    portfolio_returns.append(annualize_return) #add returns to list of port returns
    #variance
    matrix_covariance_portfolio = (stock_returns.cov())*250
    portfolio_variance = np.dot(weights.T,np.dot(matrix_covariance_portfolio, weights))
    portfolio_standard_deviation= np.sqrt(portfolio_variance)
    portfolio_risk.append(portfolio_standard_deviation)
    #sharpe_ratio
    sharpe_ratio = ((annualize_return)/portfolio_standard_deviation)
    sharpe_ratio_port.append(sharpe_ratio)


    portfolio_weights.append(weights)
```

In [ ]:

In [139…
```
portfolio_risk = np.array(portfolio_risk)
portfolio_returns = np.array(portfolio_returns)
sharpe_ratio_port = np.array(sharpe_ratio_port)
```

In [140…
```
plt.figure(figsize=(10, 5))
plt.scatter(portfolio_risk, portfolio_returns, c=portfolio_returns / portfolio_risk)
plt.xlabel('volatility')
plt.ylabel('returns')
plt.colorbar(label='Sharpe ratio')
```

Out[140… <matplotlib.colorbar.Colorbar at 0x1d7b6fb5f70>



In [141…
```
porfolio_metrics = [portfolio_returns,portfolio_risk,sharpe_ratio_port, portfolio_weigh
#from Python list we create a Pandas DataFrame
portfolio_dfs = pd.DataFrame(porfolio_metrics)
portfolio_dfs = portfolio_dfs.T
#Rename the columns:
portfolio_dfs.columns = ['Port Returns','Port Risk','Sharpe Ratio','Portfolio Weights']

#convert from object to float the first three columns.
for col in ['Port Returns', 'Port Risk', 'Sharpe Ratio']:
    portfolio_dfs[col] = portfolio_dfs[col].astype(float)
portfolio_dfs
```

Out[141…

	Port Returns	Port Risk	Sharpe Ratio	Portfolio Weights
0	0.390566	0.313491	1.245862	[0.28538011457386275, 0.2717486965652323, 0.00...
1	0.450497	0.373753	1.205335	[0.22713011989424262, 0.013942770242693171, 0....
2	0.516000	0.350601	1.471758	[0.24455025712932205, 0.2658272397101366, 0.08...
3	0.553690	0.424637	1.303913	[0.16545875090293874, 0.17934592157777177, 0.4...
4	0.433372	0.314336	1.378691	[0.1370887608256247, 0.15557690038651095, 0.19...
...	...	...	...	...

	Port Returns	Port Risk	Sharpe Ratio	Portfolio Weights
99995	0.559598	0.380887	1.469195	[0.01364551914856885, 0.2780891319302786, 0.09...
99996	0.655074	0.439383	1.490895	[0.27394592941647067, 0.3623312372307938, 0.05...
99997	0.620564	0.447950	1.385343	[0.10807136113014384, 0.22885763878922785, 0.0...
99998	0.563748	0.406083	1.388258	[0.011586989985433635, 0.1954077463893398, 0.2...
99999	0.607186	0.415797	1.460292	[0.2808028196389986, 0.2710408104562849, 0.144...

100000 rows × 4 columns

In [142...
```
porfolio_metrics = [portfolio_returns,portfolio_risk,sharpe_ratio_port, portfolio_weigh
#from Python list we create a Pandas DataFrame
portfolio_dfs = pd.DataFrame(porfolio_metrics)
portfolio_dfs = portfolio_dfs.T
#Rename the columns:
portfolio_dfs.columns = ['Port Returns','Port Risk','Sharpe Ratio','Portfolio Weights']

#convert from object to float the first three columns.
for col in ['Port Returns', 'Port Risk', 'Sharpe Ratio']:
    portfolio_dfs[col] = portfolio_dfs[col].astype(float)
portfolio_dfs
```

Out[142...

	Port Returns	Port Risk	Sharpe Ratio	Portfolio Weights
0	0.390566	0.313491	1.245862	[0.28538011457386275, 0.2717486965652323, 0.00...
1	0.450497	0.373753	1.205335	[0.22713011989424262, 0.013942770242693171, 0....
2	0.516000	0.350601	1.471758	[0.24455025712932205, 0.2658272397101366, 0.08...
3	0.553690	0.424637	1.303913	[0.16545875090293874, 0.17934592157777177, 0.4...
4	0.433372	0.314336	1.378691	[0.1370887608256247, 0.15557690038651095, 0.19...
...	...	...	...	...
99995	0.559598	0.380887	1.469195	[0.01364551914856885, 0.2780891319302786, 0.09...
99996	0.655074	0.439383	1.490895	[0.27394592941647067, 0.3623312372307938, 0.05...
99997	0.620564	0.447950	1.385343	[0.10807136113014384, 0.22885763878922785, 0.0...
99998	0.563748	0.406083	1.388258	[0.011586989985433635, 0.1954077463893398, 0.2...
99999	0.607186	0.415797	1.460292	[0.2808028196389986, 0.2710408104562849, 0.144...

100000 rows × 4 columns

In [143...
```
portfolio_dfs['weights']=[np.around(num, 3) for num in portfolio_dfs['Portfolio Weights
portfolio_dfs #List comprehension to round weights
```

Out[143...

	Port Returns	Port Risk	Sharpe Ratio	Portfolio Weights	weights

	Port Returns	Port Risk	Sharpe Ratio	Portfolio Weights	weights
0	0.390566	0.313491	1.245862	[0.28538011457386275, 0.2717486965652323, 0.00...	[0.285, 0.272, 0.001, 0.009, 0.428, 0.005]
1	0.450497	0.373753	1.205335	[0.22713011989424262, 0.013942770242693171, 0....	[0.227, 0.014, 0.263, 0.126, 0.273, 0.097]
2	0.516000	0.350601	1.471758	[0.24455025712932205, 0.2658272397101366, 0.08...	[0.245, 0.266, 0.085, 0.128, 0.25, 0.026]
3	0.553690	0.424637	1.303913	[0.16545875090293874, 0.17934592157777177, 0.4...	[0.165, 0.179, 0.418, 0.039, 0.096, 0.102]
4	0.433372	0.314336	1.378691	[0.1370887608256247, 0.15557690038651095, 0.19...	[0.137, 0.156, 0.197, 0.037, 0.298, 0.176]
...	...	...	...	...	...
99995	0.559598	0.380887	1.469195	[0.01364551914856885, 0.2780891319302786, 0.09...	[0.014, 0.278, 0.098, 0.171, 0.096, 0.343]
99996	0.655074	0.439383	1.490895	[0.27394592941647067, 0.3623312372307938, 0.05...	[0.274, 0.362, 0.054, 0.254, 0.051, 0.004]
99997	0.620564	0.447950	1.385343	[0.10807136113014384, 0.22885763878922785, 0.0...	[0.108, 0.229, 0.028, 0.336, 0.184, 0.116]
99998	0.563748	0.406083	1.388258	[0.011586989985433635, 0.1954077463893398, 0.2...	[0.012, 0.195, 0.219, 0.167, 0.096, 0.312]
99999	0.607186	0.415797	1.460292	[0.2808028196389986, 0.2710408104562849, 0.144...	[0.281, 0.271, 0.145, 0.205, 0.075, 0.024]

100000 rows × 5 columns

In [144...

```
Highest_sharpe_port = portfolio_dfs.iloc[portfolio_dfs['Sharpe Ratio'].idxmax()]
#portfolio with the minimum risk
min_risk = portfolio_dfs.iloc[portfolio_dfs['Port Risk'].idxmin()]
print('Highest sharpe ratio portfolio')
print(Highest_sharpe_port)
print('Lowest Risk Portfolio')
print(min_risk)
```

```
Highest sharpe ratio portfolio
Port Returns                                                0.652585
Port Risk                                                   0.430032
Sharpe Ratio                                                1.517524
Portfolio Weights    [0.24485779772326968, 0.44617036979764574, 0.1...
weights                        [0.245, 0.446, 0.105, 0.154, 0.013, 0.036]
Name: 60877, dtype: object
Lowest Risk Portfolio
Port Returns                                                0.262865
Port Risk                                                   0.250378
Sharpe Ratio                                                1.049872
Portfolio Weights    [0.1716907220175638, 0.007133977426485462, 0.0...
weights                        [0.172, 0.007, 0.069, 0.006, 0.561, 0.185]
Name: 91256, dtype: object
```

In [145...

```
max_risk = portfolio_dfs.iloc[portfolio_dfs['Port Risk'].idxmax()]
print(max_risk)
```

```
Port Returns                                              0.838662
Port Risk                                                 0.815507
Sharpe Ratio                                              1.028393
Portfolio Weights    [0.047721616495915595, 0.01116946813886075, 0....
weights                        [0.048, 0.011, 0.101, 0.754, 0.041, 0.045]
Name: 45779, dtype: object
```

In [146...

```
max_return = portfolio_dfs.iloc[portfolio_dfs['Port Returns'].idxmax()]
print(max_return)
```

```
Port Returns                                              0.841785
Port Risk                                                 0.566158
Sharpe Ratio                                              1.486837
Portfolio Weights    [0.0032262469815577785, 0.6059981447059009, 0....
weights                        [0.003, 0.606, 0.053, 0.327, 0.006, 0.005]
Name: 2979, dtype: object
```

In [147...

```
Lowest_sharpe_port = portfolio_dfs.iloc[portfolio_dfs['Sharpe Ratio'].idxmin()]
print(Lowest_sharpe_port)
```

```
Port Returns                                              0.249587
Port Risk                                                 0.263425
Sharpe Ratio                                              0.947466
Portfolio Weights    [0.4336771750993164, 0.02149875953048767, 2.58...
weights                        [0.434, 0.021, 0.0, 0.006, 0.421, 0.117]
Name: 29013, dtype: object
```

In [148...

```
min_return = portfolio_dfs.iloc[portfolio_dfs['Port Returns'].idxmin()]
print(min_return)
```

```
Port Returns                                              0.248465
Port Risk                                                 0.261515
Sharpe Ratio                                              0.950101
Portfolio Weights    [0.41535168602185357, 0.020271800830861757, 0....
weights                        [0.415, 0.02, 0.01, 0.001, 0.428, 0.126]
Name: 41176, dtype: object
```

In [116...

```
tickers2=['AMZN',"TSLA",'BTC-USD','ETH-USD','QQQ','VGT','^GSPC','^IRX']
stocks2=pd.DataFrame()
for t in tickers2:
    stocks2[t]=pdr.DataReader(t,data_source='yahoo',start='1990-1-1')['Adj Close']
```

In [117...

```
stock_returns2=(stocks2/stocks2.shift(1))-1
stock_returns2['^IRX']=stocks2['^IRX']/(100*250)
stock_returns2=stock_returns2.dropna(axis=0,how='any')
stock_returns2
```

Out[117...

Date	AMZN	TSLA	BTC-USD	ETH-USD	QQQ	VGT	^GSPC	^IRX
2017-11-10	-0.003348	0.000000	-0.073554	-0.067411	-0.000065	0.000364	-0.000898	0.000048

Date	AMZN	TSLA	BTC-USD	ETH-USD	QQQ	VGT	^GSPC	^IRX
2017-11-13	0.003395	0.040958	-0.008862	0.058355	0.001236	-0.000182	0.000984	0.000047
2017-11-14	0.006793	-0.021243	0.011626	0.066037	-0.003639	-0.001760	-0.002310	0.000050
2017-11-15	-0.008928	0.008422	0.102444	-0.012659	-0.004697	-0.007600	-0.005526	0.000049
2017-11-16	0.009408	0.003855	0.076023	-0.007298	0.012780	0.013784	0.008196	0.000049
...	...	...	...	...	...	...	...	...
2022-03-01	-0.015765	-0.006962	0.026889	0.018253	-0.015311	-0.019237	-0.015474	0.000013
2022-03-02	0.006024	0.017955	-0.009706	-0.007525	0.016779	0.020826	0.018643	0.000013
2022-03-03	-0.027320	-0.046142	-0.033520	-0.039202	-0.014285	-0.015554	-0.005255	0.000013
2022-03-04	-0.015264	-0.001191	-0.078069	-0.076668	-0.014492	-0.019945	-0.007934	0.000012
2022-03-07	-0.056220	-0.040213	-0.023972	-0.043688	-0.036881	-0.037497	-0.029518	0.000012

1086 rows × 8 columns

In [118...
```
X1=sm.add_constant(stock_returns2['^GSPC']-stock_returns2['^IRX'])
X1
```

```
C:\Users\jovan\anaconda3\lib\site-packages\statsmodels\tsa\tsatools.py:142: FutureWarnin
g: In a future version of pandas all arguments of concat except for the argument 'objs'
will be keyword-only
  x = pd.concat(x[::order], 1)
```

Out[118...

Date	const	0
2017-11-10	1.0	-0.000946
2017-11-13	1.0	0.000936
2017-11-14	1.0	-0.002359
2017-11-15	1.0	-0.005575
2017-11-16	1.0	0.008147
...	...	...
2022-03-01	1.0	-0.015486
2022-03-02	1.0	0.018630
2022-03-03	1.0	-0.005268
2022-03-04	1.0	-0.007946
2022-03-07	1.0	-0.029530

1086 rows × 2 columns

In [119...

```
reg1=sm.OLS(stock_returns2['AMZN']-stock_returns2['^IRX'],X1).fit()
reg1.summary()
```

Out[119...

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.392
Model:	OLS	**Adj. R-squared:**	0.391
Method:	Least Squares	**F-statistic:**	697.6
Date:	Mon, 07 Mar 2022	**Prob (F-statistic):**	4.26e-119
Time:	15:04:10	**Log-Likelihood:**	2970.7
No. Observations:	1086	**AIC:**	-5937.
Df Residuals:	1084	**BIC:**	-5927.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0005	0.000	1.067	0.286	-0.000	0.001
0	0.9545	0.036	26.413	0.000	0.884	1.025

Omnibus:	262.980	**Durbin-Watson:**	1.991
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2837.300
Skew:	0.789	**Prob(JB):**	0.00
Kurtosis:	10.760	**Cond. No.**	75.8

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [120...

```
reg2=sm.OLS(stock_returns2['TSLA']-stock_returns2['^IRX'],X1).fit()
reg2.summary()
```

Out[120...

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.194
Model:	OLS	**Adj. R-squared:**	0.193
Method:	Least Squares	**F-statistic:**	261.1
Date:	Mon, 07 Mar 2022	**Prob (F-statistic):**	8.88e-53
Time:	15:04:10	**Log-Likelihood:**	2052.5
No. Observations:	1086	**AIC:**	-4101.
Df Residuals:	1084	**BIC:**	-4091.
Df Model:	1		

Covariance Type:		nonrobust				
	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	0.0025	0.001	2.247	0.025	0.000	0.005
0	1.3600	0.084	16.157	0.000	1.195	1.525

Omnibus:	155.645	**Durbin-Watson:**	2.034
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1107.435
Skew:	0.431	**Prob(JB):**	3.34e-241
Kurtosis:	7.872	**Cond. No.**	75.8

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [121…
```
reg3=sm.OLS(stock_returns2['BTC-USD']-stock_returns2['^IRX'],X1).fit()
reg3.summary()
```

Out[121…

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.045
Model:	OLS	**Adj. R-squared:**	0.044
Method:	Least Squares	**F-statistic:**	50.92
Date:	Mon, 07 Mar 2022	**Prob (F-statistic):**	1.76e-12
Time:	15:04:10	**Log-Likelihood:**	1771.1
No. Observations:	1086	**AIC:**	-3538.
Df Residuals:	1084	**BIC:**	-3528.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	0.0023	0.001	1.602	0.109	-0.001	0.005
0	0.7783	0.109	7.136	0.000	0.564	0.992

Omnibus:	122.790	**Durbin-Watson:**	2.011
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	873.988
Skew:	0.221	**Prob(JB):**	1.64e-190
Kurtosis:	7.373	**Cond. No.**	75.8

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [122...
```
reg4=sm.OLS(stock_returns2['ETH-USD']-stock_returns2['^IRX'],X1).fit()
reg4.summary()
```

Out[122...

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.057
Model:	OLS	**Adj. R-squared:**	0.056
Method:	Least Squares	**F-statistic:**	65.53
Date:	Mon, 07 Mar 2022	**Prob (F-statistic):**	1.53e-15
Time:	15:04:10	**Log-Likelihood:**	1498.9
No. Observations:	1086	**AIC:**	-2994.
Df Residuals:	1084	**BIC:**	-2984.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0033	0.002	1.767	0.077	-0.000	0.007
0	1.1344	0.140	8.095	0.000	0.859	1.409

Omnibus:	156.119	**Durbin-Watson:**	2.050
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1041.188
Skew:	0.458	**Prob(JB):**	8.11e-227
Kurtosis:	7.709	**Cond. No.**	75.8

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [123...
```
reg5=sm.OLS(stock_returns2['QQQ']-stock_returns2['^IRX'],X1).fit()
reg5.summary()
```

Out[123...

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.854
Model:	OLS	**Adj. R-squared:**	0.853
Method:	Least Squares	**F-statistic:**	6319.
Date:	Mon, 07 Mar 2022	**Prob (F-statistic):**	0.00
Time:	15:04:10	**Log-Likelihood:**	4035.8
No. Observations:	1086	**AIC:**	-8068.

Df Residuals:	1084	**BIC:**	-8058.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0003	0.000	1.465	0.143	-8.89e-05	0.001
0	1.0774	0.014	79.493	0.000	1.051	1.104

Omnibus:	96.199	**Durbin-Watson:**	1.981
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	467.273
Skew:	-0.236	**Prob(JB):**	3.41e-102
Kurtosis:	6.179	**Cond. No.**	75.8

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [124...
```
reg6=sm.OLS(stock_returns2['VGT']-stock_returns2['^IRX'],X1).fit()
reg6.summary()
```

Out[124...

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.870
Model:	OLS	**Adj. R-squared:**	0.870
Method:	Least Squares	**F-statistic:**	7232.
Date:	Mon, 07 Mar 2022	**Prob (F-statistic):**	0.00
Time:	15:04:10	**Log-Likelihood:**	4011.5
No. Observations:	1086	**AIC:**	-8019.
Df Residuals:	1084	**BIC:**	-8009.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0003	0.000	1.732	0.084	-4.21e-05	0.001
0	1.1787	0.014	85.042	0.000	1.151	1.206

Omnibus:	54.472	**Durbin-Watson:**	1.989
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	166.428
Skew:	-0.147	**Prob(JB):**	7.25e-37
Kurtosis:	4.895	**Cond. No.**	75.8

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [125...
```
sec_beta=pd.DataFrame(np.nan,index=tickers,columns=['beta','intercept'])
sec_beta
```

Out[125...

	beta	intercept
AMZN	NaN	NaN
TSLA	NaN	NaN
BTC-USD	NaN	NaN
ETH-USD	NaN	NaN
QQQ	NaN	NaN
VGT	NaN	NaN

In [126...
```
for t in tickers2:
    slope,intercept,r_value,p_value,std_err=stats.linregress(stock_returns2['^GSPC'],st
    sec_beta.loc[t,'beta']=slope
    sec_beta.loc[t,'intercept']=intercept
sec_beta
```

Out[126...

	beta	intercept
AMZN	0.954578	0.000511
TSLA	1.359741	0.002481
BTC-USD	0.777988	0.002316
ETH-USD	1.133730	0.003264
QQQ	1.077394	0.000259
VGT	1.178730	0.000309
^GSPC	1.000000	0.000000
^IRX	-0.000064	0.000042

In [127...
```
reg7=sm.OLS(port_ret1-stock_returns2['^IRX'],X1).fit() #portfolio of Amzn and Tesla
reg7.summary()
```

Out[127...

OLS Regression Results

Dep. Variable:	^IRX	**R-squared:**	0.345
Model:	OLS	**Adj. R-squared:**	0.344
Method:	Least Squares	**F-statistic:**	570.8
Date:	Mon, 07 Mar 2022	**Prob (F-statistic):**	1.11e-101

Time:	15:04:11	**Log-Likelihood:**	2652.5
No. Observations:	1086	**AIC:**	-5301.
Df Residuals:	1084	**BIC:**	-5291.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0015	0.001	2.350	0.019	0.000	0.003
0	1.1573	0.048	23.891	0.000	1.062	1.252

Omnibus:	146.407	**Durbin-Watson:**	2.022
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	830.101
Skew:	0.469	**Prob(JB):**	5.57e-181
Kurtosis:	7.179	**Cond. No.**	75.8

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [128…
```
reg8=sm.OLS(port_ret11-stock_returns2['^IRX'],X1).fit() #portfolio of TSLA and BTC and
reg8.summary()
```

Out[128…

OLS Regression Results

Dep. Variable:	^IRX	**R-squared:**	0.318
Model:	OLS	**Adj. R-squared:**	0.317
Method:	Least Squares	**F-statistic:**	505.0
Date:	Mon, 07 Mar 2022	**Prob (F-statistic):**	4.02e-92
Time:	15:04:11	**Log-Likelihood:**	2574.9
No. Observations:	1086	**AIC:**	-5146.
Df Residuals:	1084	**BIC:**	-5136.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0019	0.001	2.772	0.006	0.001	0.003
0	1.1693	0.052	22.473	0.000	1.067	1.271

Omnibus:	106.319	**Durbin-Watson:**	2.022
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	395.791

Skew:	0.412	**Prob(JB):**	1.13e-86
Kurtosis:	5.840	**Cond. No.**	75.8

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [129…
```
reg10=sm.OLS(port_ret9-stock_returns2['^IRX'],X1).fit() #portfolio of ETH and Amzn and
reg10.summary()
```

Out[129…

OLS Regression Results

Dep. Variable:	^IRX	**R-squared:**	0.173
Model:	OLS	**Adj. R-squared:**	0.173
Method:	Least Squares	**F-statistic:**	227.5
Date:	Mon, 07 Mar 2022	**Prob (F-statistic):**	8.22e-47
Time:	15:04:11	**Log-Likelihood:**	2232.9
No. Observations:	1086	**AIC:**	-4462.
Df Residuals:	1084	**BIC:**	-4452.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	0.0018	0.001	1.941	0.052	-1.96e-05	0.004
0	1.0753	0.071	15.084	0.000	0.935	1.215

Omnibus:	146.109	**Durbin-Watson:**	2.031
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	859.369
Skew:	0.454	**Prob(JB):**	2.46e-187
Kurtosis:	7.262	**Cond. No.**	75.8

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [130…
```
reg9=sm.OLS(port_ret15-stock_returns2['^IRX'],X1).fit() #Portfolio of 50% eth and 10% o
reg9.summary() #const is now even more significant
```

Out[130…

OLS Regression Results

Dep. Variable:	^IRX	**R-squared:**	0.150
Model:	OLS	**Adj. R-squared:**	0.150

Method:	Least Squares	**F-statistic:**	191.7
Date:	Mon, 07 Mar 2022	**Prob (F-statistic):**	2.88e-40
Time:	15:04:11	**Log-Likelihood:**	2113.1
No. Observations:	1086	**AIC:**	-4222.
Df Residuals:	1084	**BIC:**	-4212.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0022	0.001	2.116	0.035	0.000	0.004
0	1.1022	0.080	13.847	0.000	0.946	1.258

Omnibus:	115.895	**Durbin-Watson:**	2.046
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	706.713
Skew:	0.259	**Prob(JB):**	3.46e-154
Kurtosis:	6.918	**Cond. No.**	75.8

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.