Setup

In [499]:
```
import pandas as pd
import numpy as np
import pandas_datareader.data as web
import datetime
import requests
import sys
import time
import random
import traceback
import shelve
import plotly.plotly as py # online
#import plotly.offline as py # offline
import cufflinks as cf
cf.set_config_file(offline=False, world_readable=True, theme='ggplot')
import matplotlib.pyplot as plt
%matplotlib inline
from gurobipy import *
from math import sqrt
```

Get Historical Currency Data

In [500]:
```
# Modified the code from https://github.com/lagerfeuer/cryptocompare/blob/master/cryptocompare/cryptocompare.py

# API
URL_HIST_PRICE_DAILY = 'https://min-api.cryptocompare.com/data/histoday?fsym={}&tsym={}&limit={}'
URL_COIN_LIST = 'https://www.cryptocompare.com/api/data/coinlist/'
CURR = 'USD'

def query_cryptocompare(url,errorCheck=True):
    try:
        response = requests.get(url).json()
    except Exception as e:
        print('Error getting coin information. %s' % str(e))
        return None
    if errorCheck and (response.get('Response') == 'Error'):
        print('[ERROR] %s' % response.get('Message'))
        return None
    return response

def get_coin_list(format=False):
    response = query_cryptocompare(URL_COIN_LIST, False)['Data']
    if format:
        return list(response.keys())
    else:
        return response

def get_historical_price_daily(coin, curr=CURR, days=10):
    return query_cryptocompare(URL_HIST_PRICE_DAILY.format(coin, curr, int(days)))
```

In [467]:
```
#coin_list_full = get_coin_list()
#len(coin_list_full.keys())
```

Out[467]: 2319

In [501]:
```
coin_list = ['ETH','BTC','LTC', 'XRP', 'ETC','XMR','DASH','MAID','REP', 'XLM']

# crypto_data = dict()
# for coin in coin_list:
#     try:
#         crypto_data[coin] = get_historical_price_daily(coin, days=365)
#         time.sleep(1)
#     except Exception:
#         traceback.print_exc()
#         time.sleep(30)

# with shelve.open('shelf') as db:
#     db['crypto_data'] = crypto_data
```

In [469]:
```
# with shelve.open('shelf') as db:
#     db['crypto_data'] = crypto_data
```

In [502]:
```
# Open shelved data
with shelve.open('shelf') as db:
    crypto_data = db['crypto_data']
```

Calculate Daily returns and covariances

In [503]:
```
data = dict()

for key, response in crypto_data.items():
    df = pd.DataFrame(response['Data'])
    df = df.set_index(pd.to_datetime(df['time'],unit='s')).drop(columns='time')['close']
    df.name = 'Close'
    data[key] = pd.DataFrame(df)
```

In [504]:
```
for k, m in data.items():
    m['return'] = (m['Close'] - m['Close'].shift(1)) / m['Close'].shift(1)
```

In [505]:
```
random.seed(1)
n_subset = len(data)

# smaller subset for testing
d2 = {k: data[k] for k in random.sample([key for key, value in data.items()],n_subset)}
returns = pd.concat([x['return'].fillna(0).replace(np.inf, 0) for x in d2.values()], axis=1,
                    keys=d2.keys())

# drop columns if they are all NaN
returns = returns.drop(returns.columns[~returns.notnull().any()], axis=1)

cov_matrix = returns.cov()
means = returns.mean()
returns.columns
```

Out[505]:
```
Index(['LTC', 'BTC', 'ETC', 'ETH', 'XRP', 'XMR', 'MAID', 'XLM', 'REP', 'DASH'], dtype='object')
```

In [506]:
```
returns
```

Out[506]:

	LTC	BTC	ETC	ETH	XRP	XMR	MAID	XLM	REP	DASH
time										
2017-03-11	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2017-03-12	0.018421	0.038916	0.021739	0.086713	0.011200	0.154372	0.000000	0.063817	0.267581	0.047822
2017-03-13	0.126615	0.010677	0.163121	0.220506	0.018987	0.094083	0.000000	0.046406	0.041949	-0.018282
2017-03-14	-0.043578	0.003998	0.006098	0.004569	-0.007764	-0.058410	0.000000	-0.017307	0.102597	0.174839
2017-03-15	0.021583	0.008277	0.072727	0.230931	-0.021909	0.109707	0.000000	0.066593	0.166078	0.094766
2017-03-16	0.016432	-0.064264	0.175141	0.293633	0.008000	0.154244	0.000000	-0.127451	0.008081	-0.067813
2017-03-17	-0.030023	-0.086258	-0.168269	-0.022632	-0.052381	-0.014350	0.000000	0.045535	-0.258517	0.100905
2017-03-18	-0.035714	-0.093617	-0.063584	-0.235612	0.152429	-0.117379	0.000000	0.121606	-0.114865	0.103735
2017-03-19	-0.017284	0.052729	0.197531	0.268235	-0.027616	0.128351	0.000000	-0.142209	0.221374	-0.001379
2017-03-20	0.035176	0.024359	-0.036082	-0.014147	0.031390	0.022385	0.000000	0.175191	0.090000	-0.061884
2017-03-21	-0.007282	0.070434	0.272727	0.003764	-0.011594	-0.036193	0.000000	0.050525	-0.077982	-0.037204
2017-03-22	-0.026895	-0.068287	-0.042017	-0.023904	0.054252	-0.054242	0.000000	-0.104762	0.060945	0.054751
2017-03-23	0.010050	-0.009275	-0.017544	0.037215	0.529903	0.079412	0.000000	0.170213	0.067995	-0.009860
2017-03-24	0.039801	-0.092104	0.075893	0.231250	-0.034545	-0.018619	0.000000	-0.113636	-0.017563	-0.004784
2017-03-25	-0.011962	0.028307	-0.082988	-0.048317	-0.163842	-0.086071	0.000000	0.025641	-0.026816	-0.068177
2017-03-26	-0.007264	0.003250	0.031674	0.000198	0.055180	-0.043038	0.000000	-0.004500	-0.039036	-0.009054
2017-03-27	0.000000	0.078086	-0.078947	-0.031009	0.006403	0.026455	0.000000	-0.060773	0.032258	-0.097100
2017-03-28	0.024390	-0.000689	0.066667	0.024256	0.012725	0.030412	0.000000	0.120321	0.068287	0.031415
2017-03-29	0.028571	-0.002413	0.035714	0.056119	0.062827	0.059530	0.000000	-0.084964	-0.003250	-0.033082
2017-03-30	0.736111	-0.003830	0.228448	-0.021858	0.362562	-0.046270	0.000000	0.199791	0.069565	-0.074445
2017-03-31	-0.053333	0.039695	-0.007018	-0.038528	0.547361	-0.000495	0.000000	-0.007826	0.269309	-0.077119
2017-04-01	0.053521	0.006496	-0.028269	0.013825	0.028037	0.042595	0.000000	0.248904	0.008807	-0.068785
2017-04-02	0.098930	0.010386	-0.021818	-0.040514	1.795455	-0.044656	0.000000	0.545614	-0.053175	-0.169386
2017-04-03	0.042579	0.045708	-0.044610	-0.091040	-0.479512	-0.005470	0.000000	-0.310783	-0.119028	0.115179
2017-04-04	0.046674	-0.005045	0.035019	0.006798	0.183068	0.023500	0.000000	-0.008235	-0.004757	0.099119
2017-04-05	0.381271	-0.010422	0.033835	0.010578	-0.060206	-0.035173	0.000000	0.023580	-0.000956	0.078671
2017-04-06	-0.136400	0.052068	-0.029091	-0.037194	-0.067154	-0.014177	0.000000	-0.081441	-0.011483	-0.107780
2017-04-07	-0.075701	0.001506	0.029963	-0.021282	0.118976	0.017976	0.000000	-0.039562	-0.055179	0.001211
2017-04-08	0.077856	-0.008156	-0.010909	0.048688	-0.037685	0.049950	0.000000	0.126885	0.031762	0.016631
2017-04-09	-0.136023	0.019953	-0.022059	-0.014650	-0.043077	-0.004805	0.000000	-0.096279	-0.003972	-0.032719
...	...	...	...	...	...	...	...	...	...	...
2018-02-10	-0.054851	-0.014665	-0.063399	-0.030904	0.119444	-0.042811	-0.061095	-0.010204	-0.043571	-0.006780
2018-02-11	-0.037764	-0.056560	-0.013704	-0.046441	-0.067282	-0.077905	-0.041606	-0.066237	-0.083333	-0.084682
2018-02-12	0.082785	0.102251	0.242526	0.066602	0.082544	0.084574	0.103483	0.061275	0.080808	0.067445
2018-02-13	-0.014560	-0.041137	0.125042	-0.028072	-0.051346	-0.053107	-0.064044	0.035111	-0.090654	-0.039048
2018-02-14	0.332977	0.110121	0.044277	0.094093	0.145348	0.175358	0.179374	0.125879	0.078520	0.142652
2018-02-15	0.042781	0.057783	-0.027113	0.008521	-0.017699	0.075708	0.054496	-0.016737	0.019630	0.027458
2018-02-16	0.033065	0.015446	0.026979	0.010852	0.000000	-0.007546	-0.040086	0.014526	-0.001495	-0.002658
2018-02-17	0.004466	0.089165	0.001155	0.039178	0.063063	0.092632	0.071648	0.070246	0.031262	0.065216
2018-02-18	-0.066213	-0.061275	-0.010957	-0.062445	-0.093220	-0.079662	-0.142311	-0.091346	-0.050463	-0.072123
2018-02-19	0.036178	0.073441	0.119242	0.028329	0.037383	0.065909	0.031626	0.019094	0.015102	0.060662
2018-02-20	0.033383	0.006631	-0.046887	-0.057747	-0.072072	-0.039569	-0.030009	-0.120316	-0.065725	-0.056986
2018-02-21	-0.083355	-0.068829	-0.057939	-0.051292	-0.076505	0.020666	-0.020254	-0.039004	-0.076396	-0.036109
2018-02-22	-0.082517	-0.060458	-0.090513	-0.042221	-0.063919	-0.108277	-0.077919	-0.073164	-0.057617	-0.048468
2018-02-23	0.070966	0.033261	0.151196	0.062227	0.055930	0.015938	-0.050751	0.056180	0.018527	0.004274
2018-02-24	-0.001210	-0.046168	0.016348	-0.024816	-0.041800	-0.038861	-0.049572	-0.066285	-0.025921	-0.064009
2018-02-25	0.057960	-0.009852	-0.043075	0.008146	0.000333	0.037861	-0.048061	0.019866	0.027311	0.002654
2018-02-26	0.001420	0.074545	0.003989	0.032537	0.031514	0.045420	0.028112	0.015755	0.055215	0.050859
2018-02-27	-0.015872	0.025978	0.011351	0.004564	-0.002151	0.032216	0.003906	0.005076	0.040267	-0.018175
2018-02-28	-0.060469	-0.024571	-0.066498	-0.023039	-0.045602	-0.047781	0.120345	-0.074355	-0.060029	-0.043068
2018-03-01	0.035520	0.057568	0.006011	0.021574	0.034113	0.095290	-0.009427	0.024553	0.027967	0.060390
2018-03-02	0.015622	0.010408	-0.089035	-0.016405	-0.018788	0.095262	0.023291	-0.057396	-0.033633	-0.024523
2018-03-03	-0.010819	0.038236	-0.053788	0.000058	0.000111	0.019225	0.019824	0.091337	0.043228	0.013271

	LTC	BTC	ETC	ETH	XRP	XMR	MAID	XLM	REP	DASH
time										
2018-03-04	0.014599	0.003407	0.023570	0.010729	0.113090	0.053671	-0.011279	0.042278	-0.005737	0.013211
2018-03-05	-0.011999	-0.005536	-0.101253	-0.017819	-0.057300	0.004421	-0.019417	-0.036424	-0.011541	-0.010885
2018-03-06	-0.067457	-0.061646	-0.065185	-0.039709	-0.043916	-0.070372	-0.082426	-0.044387	-0.064216	-0.052340
2018-03-07	-0.053617	-0.075161	-0.118904	-0.079148	-0.053035	-0.021234	-0.044510	-0.025472	-0.098429	-0.116825
2018-03-08	-0.054934	-0.061624	0.016011	-0.069628	-0.056473	-0.180947	-0.150762	-0.049508	-0.008457	-0.035366
2018-03-09	0.060061	-0.006865	-0.008555	0.040196	0.022724	0.033517	0.058511	-0.004853	-0.048850	0.010711
2018-03-10	-0.048718	-0.049227	-0.060854	-0.061382	-0.059009	-0.108264	-0.081658	-0.058843	-0.101902	-0.028924
2018-03-11	0.061534	0.087451	0.034333	0.056060	0.056000	0.103322	0.119699	0.034888	0.090469	0.105418

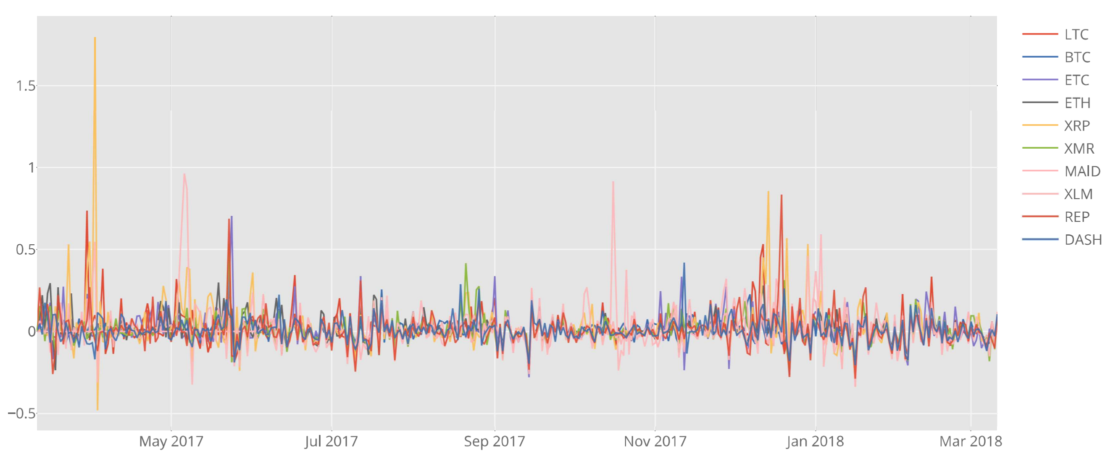
366 rows × 10 columns

In [507]:
```
means
```

Out[507]:
```
LTC      0.014739
BTC      0.007218
ETC      0.011628
ETH      0.012355
XRP      0.021774
XMR      0.011504
MAID     0.002374
XLM      0.023045
REP      0.009638
DASH     0.008104
dtype: float64
```

In [477]:
```
# plotly
returns.iplot(filename='pred460_crypto_returns',online=True)
```

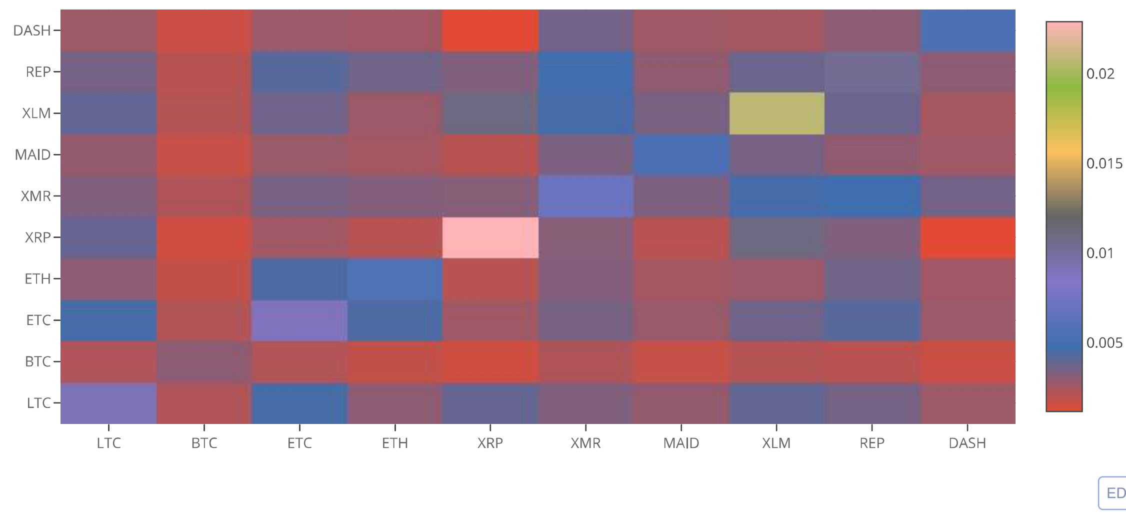
Out[477]:



EDIT CHART

In [480]:
```
# # plotly
cov_matrix.iplot(kind='heatmap', filename='pred460_crypto_heatmap', online=True)
```

Out[480]:

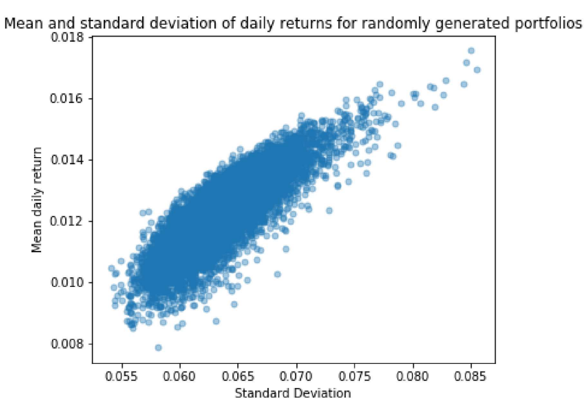


In [508]:
```
def random_portfolio(returns, means):
    '''
    Returns the mean and standard deviation of returns for a random portfolio
    '''
    k = np.random.rand(len(means))
    p = np.asmatrix(means)
    w = np.asmatrix(k / sum(k))
    C = np.asmatrix(cov_matrix)

    mu = w * p.T
    sigma = np.sqrt(w * C * w.T)

    # This recursion reduces outliers to keep plots pretty
    if sigma > 2:
        return random_portfolio(returns)
    return mu, sigma
```

In [509]:
```
n_portfolios = 10000
mns, stds = np.column_stack([
    random_portfolio(returns, means)
    for _ in range(n_portfolios)
])
```

In [510]:
```
fig = plt.figure()
fig.set_size_inches(6, 5)
plt.plot(stds, mns, 'o', markersize=5, alpha=0.4)
plt.xlabel('Standard Deviation')
plt.ylabel('Mean daily return')
plt.title('Mean and standard deviation of daily returns for randomly generated portfolios')
#py.iplot_mpl(fig, filename='pred460_crypto_mean_std', strip_style=True) # online
#py.iplot_mpl(fig, strip_style=True) # offline
```

Out[510]: Text(0.5,1,'Mean and standard deviation of daily returns for randomly generated portfolios')



Optimization

In [511]:
```
# based on example from http://www.gurobi.com/documentation/7.5/examples/portfolio_py.html#subsubsection:portfolio.py
stock_volatility = returns.std()
stock_return = returns.mean()
```

In [512]:
```
# Create an empty model
m = Model('portfolio')
```

In [513]:
```
# Add a variable for each stock
stocks = list(stock_volatility.index)
vars = pd.Series(m.addVars(stocks, name=stocks), index=stocks)
```

In [514]:
```
# Objective is to minimize risk (squared).  This is modeled using the
# covariance matrix, which measures the historical correlation between stocks.
sigma = cov_matrix
portfolio_risk = sigma.dot(vars).dot(vars)
```

In [515]:
```
m.setObjective(portfolio_risk, GRB.MINIMIZE)
```

In [516]:
```
# Fix budget with a constraint
m.addConstr(vars.sum() == 1, 'budget')

# Optimize model to find the minimum risk portfolio
m.setParam('OutputFlag', 0)
m.optimize()
```

In [517]:
```
# Create an expression representing the expected return for the portfolio
portfolio_return = stock_return.dot(vars)

# Display minimum risk portfolio
print('Minimum Risk Portfolio:\n')
for v in vars:
    if v.x > 0:
        print('\t%s\t: %g' % (v.varname, v.x))
minrisk_volatility = sqrt(portfolio_risk.getValue())
print('\nVolatility      = %g' % minrisk_volatility)
minrisk_return = portfolio_return.getValue()
print('Expected Return = %g' % minrisk_return)
```

```
Minimum Risk Portfolio:

        LTC     : 8.14799e-07
        BTC     : 0.572576
        ETC     : 6.82929e-08
        ETH     : 0.109035
        XRP     : 0.0388385
        XMR     : 9.9466e-09
        MAID    : 0.117862
        XLM     : 1.6999e-07
        REP     : 1.66961e-06
        DASH    : 0.161686

Volatility      = 0.0496372
Expected Return = 0.00791584
```

In [518]:
```
min_risk_vars = pd.Series(data=[v.x for v in vars if v.x > 0], index=[v.varname for v in vars if v.x > 0]).sort_values(ascending=False)
min_risk_vars.iplot(kind='bar',colorscale='ggplot', title='Minimum Risk Portfolio', online=False, filename='pred460_crypto_min_risk_portfolio'
)
```

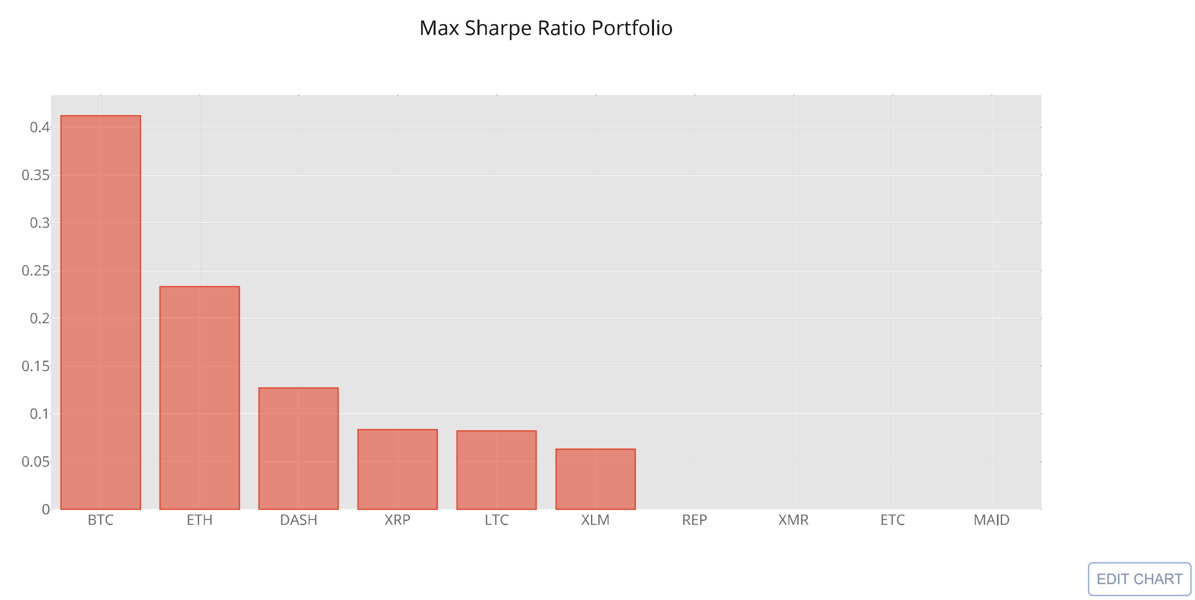
In [519]:
```
# Add (redundant) target return constraint
target = m.addConstr(portfolio_return == minrisk_return, 'target')
```

In [520]:
```
def sharpe_ratio(mean, sd, riskfree=0.0):
    return ((mean - riskfree) / sd)
```

In [521]:
```
# Solve for efficient frontier by varying target return
max_sharpe_return = -np.Inf
max_sharpe_volitility = -np.Inf
max_sharpe_a = -np.Inf
max_sharpe_vars = pd.Series()
frontier = pd.Series()
for r in np.linspace(stock_return.min(), stock_return.max(), 100):
    target.rhs = r
    m.optimize()
    frontier.loc[sqrt(portfolio_risk.getValue())] = r
    a = sharpe_ratio(portfolio_return.getValue(), portfolio_risk.getValue())
    if a > max_sharpe_a:
        max_sharpe_a = a
        max_sharpe_return = portfolio_return.getValue()
        max_sharpe_volitility = sqrt(portfolio_risk.getValue())
        max_sharpe_vars = pd.Series(data=[v.x for v in vars if v.x > 0], index=[v.varname for v in vars if v.x > 0])
```

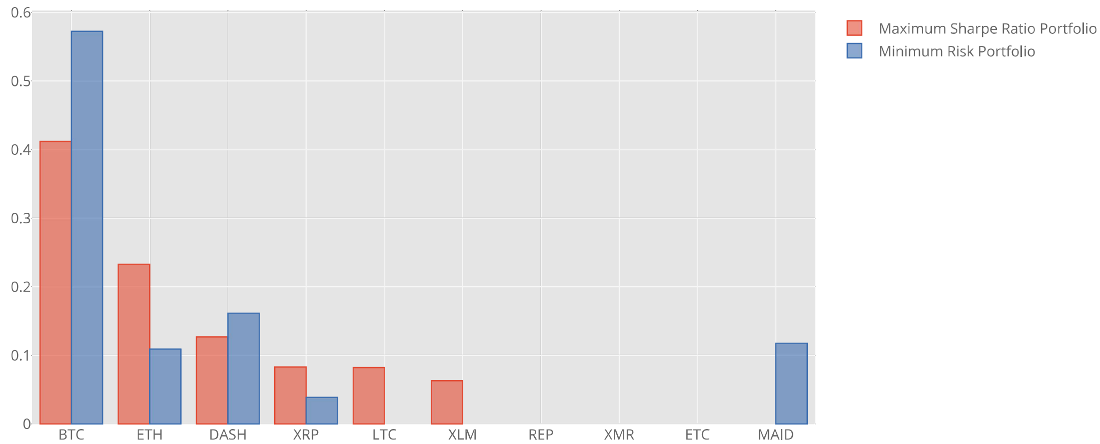
In [522]:
```
max_sharpe_vars.sort_values(ascending=False).iplot(kind='bar',colorscale='ggplot', title='Max Sharpe Ratio Portfolio', online=True, filename=
'pred460_crypto_max_sharpe_portfolio')
```

Out[522]:



In [496]:
```
b = pd.concat({'Maximum Sharpe Ratio Portfolio':max_sharpe_vars,'Minimum Risk Portfolio':min_risk_vars},
              axis=1).sort_values('Maximum Sharpe Ratio Portfolio',ascending=False)
b.iplot(kind='bar', filename='pred460_crypto_portfolio_options_bar', online=True)
```

Out[496]:



EDIT CHART

In [523]:
```
# Simulated sharpe ratio
s = pd.DataFrame(data={'sd': [float(x) for x in stds], 'mean': [float(x) for x in mns]}, index=range(len(mns)))
s['sharpe_ratio'] = s.apply(lambda row: sharpe_ratio(row['mean'], row['sd']), axis=1)
max_sharpe = s.loc[s['sharpe_ratio'].idxmax()]
s.head()
```

Out[523]:

	mean	sd	sharpe_ratio
0	0.013880	0.068976	0.201228
1	0.009981	0.056833	0.175612
2	0.013351	0.063016	0.211861
3	0.013069	0.068882	0.189731
4	0.012673	0.063774	0.198721

In [528]:
```
# Plot volatility versus expected return for individual stocks
ax = plt.gca()
fig = plt.gcf()
fig.set_size_inches(11,8)

# Plot volatility versus expected return for random portolios
plt.plot(stds, mns, 'o', markersize=4, alpha=0.4, color='orange', label='Simulated portfolios')

ax.scatter(x=stock_volatility, y=stock_return,
           color='Blue', label='Individual Currencies')
for i, stock in enumerate(stocks):
    ax.annotate(stock, (stock_volatility[i], stock_return[i]))

# Plot max sharpe ratio from optimization
ax.scatter(x=max_sharpe_volitility, y=max_sharpe_return, color='red')
ax.annotate('Maximum\nOptimized\nSharpe Ratio', (max_sharpe_volitility, max_sharpe_return),
            horizontalalignment='right')

# Plot max sharpe ratio from simulations
ax.scatter(x=max_sharpe['sd'], y=max_sharpe['mean'], color='red')
ax.annotate('Maximum\nSimulated\nSharpe Ratio', (max_sharpe['sd'], max_sharpe['mean']),
            horizontalalignment='left')

# 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
frontier.plot(color='DarkGreen', label='Efficient Frontier', ax=ax)


# Format and display the final plot
#ax.axis([0.005, 0.045, -0.001, 0.004])
ax.set_xlabel('Volatility (standard deviation)')
ax.set_ylabel('Daily Expected Return')
ax.legend()
ax.grid()
plt.show()
```