In [1]:
```python
from pycoingecko import CoinGeckoAPI
import pandas as pd
import numpy as np
import seaborn as sns
from statsmodels.tsa.stattools import adfuller,acf
from statsmodels.graphics.tsaplots import plot_acf
from sklearn.decomposition import FactorAnalysis
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from factor_analyzer import FactorAnalyzer
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from factor_analyzer.factor_analyzer import calculate_bartlett_sphe
ricity
from factor_analyzer.factor_analyzer import calculate_kmo
```

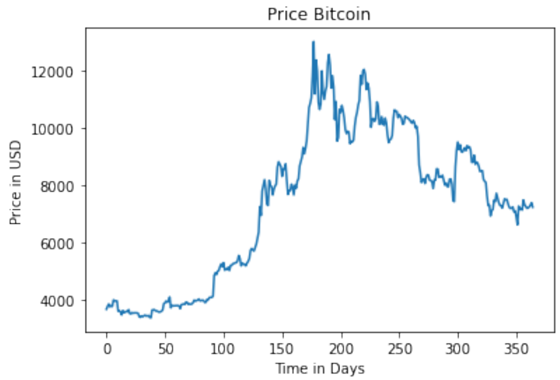
In [2]:
```python
# Calling Coin Gecko API
## Import the data
#cg = CoinGeckoAPI()
#coins_market = cg.get_coins_markets('usd')
#df_coins_market = pd.DataFrame(coins_market)
```

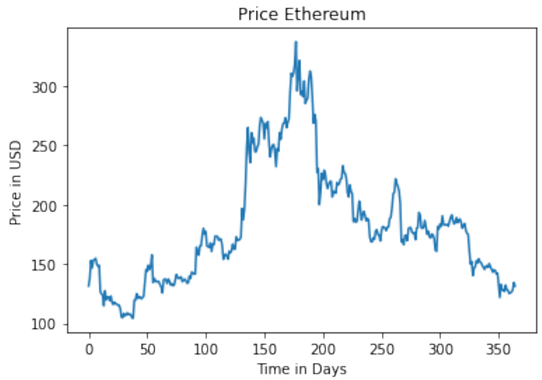
In [3]:
```python
## Get the prices for each crypto for 365 days (01.01.19 - 31.12.19
)
#prices = []
#for i in df_coins_market['id']:
#    a = cg.get_coin_market_chart_range_by_id(i, 'usd', 1546300800,
1577750400)
#    b = a['prices']
#    c = []
#   for j in range(len(b)):
#        c.append(b[j][1])
#    prices.append([i, c])
#del a, b, c, i, j

## Creating the prices, returns and market caps dataframes

#frame = pd.DataFrame(prices)
#coins = frame[1].apply(pd.Series)
#frame = coins.set_index(frame[0])
#frame = frame.dropna(axis=0).transpose()
#frame.to_csv('CC_Prices.csv',index=True,header=True)

# Returns
#returns_data = np.log(frame) - np.log(frame.shift(1))
#returns_data = returns_data[1:]
#returns_data.to_csv('CC_LogReturns.csv', index=True, header=True)

# To ensure working with same figures, we will not generate above d
f each time.
# Then, we saved them into csv to import them just below.

# Importing different df
# Prices
prices_data = pd.read_csv('CC_Prices.csv')
prices_data = prices_data.drop(['Unnamed: 0'], axis = 1)
# Returns
returns_data = pd.read_csv('CC_LogReturns.csv')
returns_data = returns_data.drop(['Unnamed: 0'], axis = 1)
# Market caps
market_cap = pd.read_csv('CC_MarketCaps.csv')
market_cap = market_cap.drop(['Unnamed: 0'], axis = 1)
```

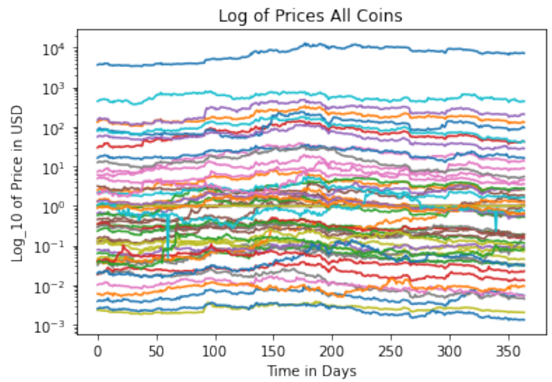
In [4]:
```python
# Exploratory Analysis

# 1) Plotting the prices over the period of various combinations of
coins
# 2) CCs returns and market caps
# 3) Stationarity test
# 4) Autocorrelation test
# 5) Statistical summary of cc
# 6) Scatterplot of correlated cc
# 7) Explorative Clustering with K-Means
```

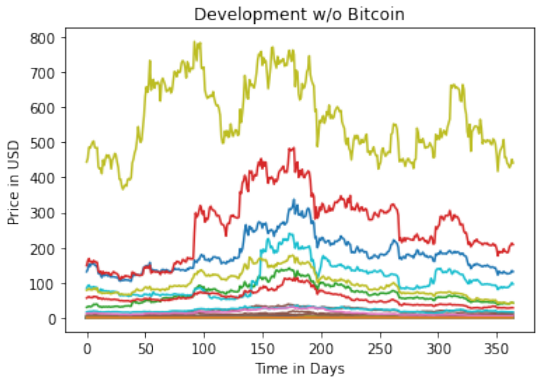
In [5]:

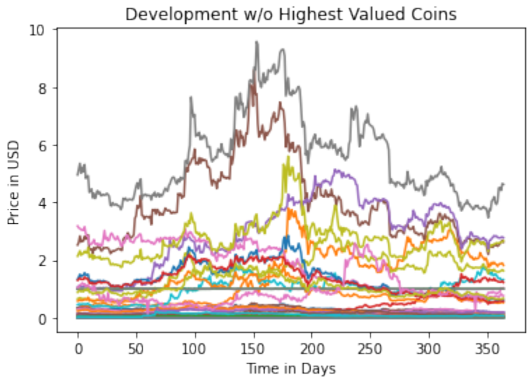
```
# Exploratory Analysis
# 1) Plotting the prices over the period of various combinations of coins

plt.plot(prices_data['bitcoin'])
plt.title('Price Bitcoin')
plt.xlabel('Time in Days')
plt.ylabel('Price in USD')
plt.show()
plt.savefig('Data_Analysis/Bitcoin.jpeg', bbox_inches='tight')
plt.clf()

plt.plot(prices_data['ethereum'])
plt.title('Price Ethereum')
plt.xlabel('Time in Days')
plt.ylabel('Price in USD')
plt.show()
plt.savefig('Data_Analysis/Ether.jpeg', bbox_inches='tight')
plt.clf()

plt.plot(prices_data)
plt.yscale('log')
plt.title('Log of Prices All Coins')
plt.xlabel('Time in Days')
plt.ylabel('Log_10 of Price in USD')
plt.show()
plt.savefig('Data_Analysis/Log_All_coins.jpeg', bbox_inches='tight'
)
plt.clf()

plt.plot(prices_data.drop('bitcoin', axis=1))
plt.title('Development w/o Bitcoin')
plt.xlabel('Time in Days')
plt.ylabel('Price in USD')
plt.show()
plt.savefig('Data_Analysis/w_o_Bitcoin.jpeg',bbox_inches='tight')
plt.clf()

plt.plot(prices_data.drop(['bitcoin', 'ethereum', 'dash', 'neo', 'z
cash', 'maker', 'bitcoin-cash', 'litecoin', 'bitcoin-cash-sv', 'dec
red', 'bitcoin-gold', 'quant-network', 'binancecoin'], axis=1))
plt.title('Development w/o Highest Valued Coins')
plt.xlabel('Time in Days')
plt.ylabel('Price in USD')
plt.show()
plt.savefig('Data_Analysis/w_o_highest.jpeg', bbox_inches='tight')
plt.clf()
```







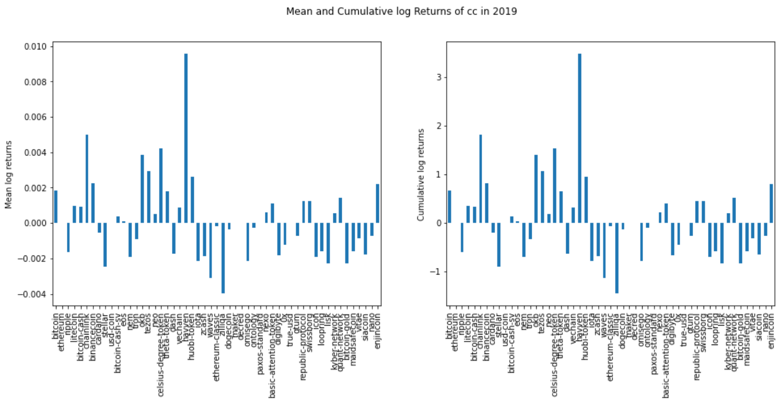


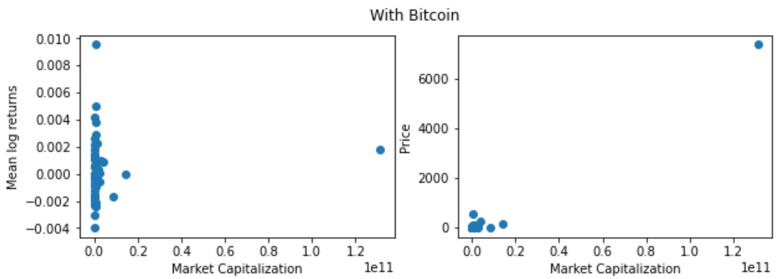


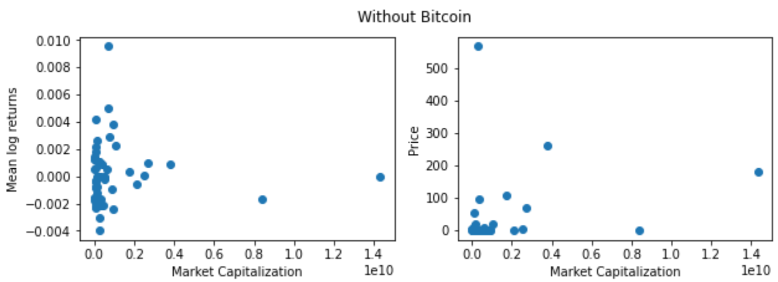

```
<Figure size 432x288 with 0 Axes>
```

In [6]:

```
# Exploratory Analysis
# 2) CCs logreturns and prices/market caps

# 2.A) CCs logreturns

mean_returns = returns_data.mean(axis=0) # mean of 2019 log returns
for each cc
cum_returns = returns_data.sum(axis=0) # cumulative log returns in
2019 for each cc

# Plotting mean and cumulative returns of 2019 for each cc
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10,3))
mean_returns.plot(ax=axes[0], kind='bar')
axes[0].set_ylabel('Mean log returns')
cum_returns.plot(ax=axes[1], kind='bar')
axes[1].set_ylabel('Cumulative log returns')
fig.set_size_inches(16, 6)
fig.suptitle('Mean and Cumulative log Returns of cc in 2019')
fig.savefig('Data_Analysis/returns.jpeg', transparent=True,bbox_inc
hes='tight')

# 2.B) Market Caps consideration with mean returns and prices

mean_price = prices_data.mean()

# 2.B.i) Scatter plots of MarketCap-MeanReturns and MarketCaps-Pric
es (with bitcoin)
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10,3))
axes[0].scatter(market_cap, mean_returns)
axes[0].set_xlabel('Market Capitalization')
axes[0].set_ylabel('Mean log returns')
axes[1].scatter(market_cap, mean_price)
axes[1].set_xlabel('Market Capitalization')
axes[1].set_ylabel('Price')
fig.suptitle('With Bitcoin')
fig.savefig('Data_Analysis/scatter.jpeg', transparent=True,bbox_inc
hes='tight')

# 2.B.ii) Scatter plots of MarketCap-MeanReturns and MarketCaps-Pri
ces (without bitcoin)
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10,3))
axes[0].scatter(market_cap[1:], mean_returns[1:])
axes[0].set_xlabel('Market Capitalization')
axes[0].set_ylabel('Mean log returns')
axes[1].scatter(market_cap[1:], mean_price[1:])
axes[1].set_xlabel('Market Capitalization')
axes[1].set_ylabel('Price')
fig.suptitle('Without Bitcoin')
fig.savefig('Data_Analysis/scatter_without.jpeg', transparent=True,
bbox_inches='tight')
```







In [7]:

```
# Exploratory Analysis
# 3) Stationarity Test

# initialization
cc_names = returns_data.columns # columns names of returns db (≠ cr
yptos)
nb_cc = len(cc_names) # number of cc

# testing stationarity, from https://www.hackdeploy.com/augmented-d
ickey-fuller-test-in-python/

class StationarityTests: # implementing a class of functions to tes
t stationarity
    def __init__(self, significance=.05):
        self.SignificanceLevel = significance
        self.pValue = None
        self.isStationary = None

    def ADF_Stationarity_Test(self, timeseries, printResults = True
):

        #Dickey-Fuller test:
        adfTest = adfuller(timeseries, autolag='AIC')

        self.pValue = adfTest[1]

        if (self.pValue<self.SignificanceLevel):
            self.isStationary = True
        else:
            self.isStationary = False

        if printResults:
            dfResults = pd.Series(adfTest[0:4], index=['ADF Test St
atistic','P-Value','# Lags Used','# Observations Used'])

            #Add Critical Values
            for key,value in adfTest[4].items():
                dfResults['Critical Value (%s)'%key] = value

            print('Augmented Dickey-Fuller Test Results:')
            print(dfResults)

statio_cc = [] # creating a matrix which will give if cc are statio
nary or not
for i in cc_names:
    sTest = StationarityTests()
    sTest.ADF_Stationarity_Test(returns_data[i],printResults = True
)
    print("Is the time series stationary? {0}".format(sTest.isStati
onary))
    statio_cc.append([i,sTest.isStationary])

statio_cc = pd.DataFrame(statio_cc) # converting it to a panda df
statio_cc.index = cc_names; statio_cc.columns = ["cc","Stationary"]
statio_cc = statio_cc['Stationary']

# exporting it to a csv
statio_cc.to_csv('Data_Analysis/statio_cc.csv',index = True,header
= True)

del i,sTest # deleting useless variables
```

```
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -19.786298
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
```

```
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -20.576621
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -1.510723e+01
P-Value                 7.787822e-28
# Lags Used             1.000000e+00
# Observations Used     3.620000e+02
Critical Value (1%)    -3.448544e+00
Critical Value (5%)    -2.869557e+00
Critical Value (10%)   -2.571041e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -18.924694
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -19.712866
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -20.545566
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -19.175157
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
```

```
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -20.227707
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -1.411358e+01
P-Value                 2.495172e-26
# Lags Used             1.000000e+00
# Observations Used     3.620000e+02
Critical Value (1%)    -3.448544e+00
Critical Value (5%)    -2.869557e+00
Critical Value (10%)   -2.571041e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -1.349253e+01
P-Value                 3.086296e-25
# Lags Used             3.000000e+00
# Observations Used     3.600000e+02
Critical Value (1%)    -3.448646e+00
Critical Value (5%)    -2.869602e+00
Critical Value (10%)   -2.571065e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -5.752633e+00
P-Value                 5.917885e-07
# Lags Used             8.000000e+00
# Observations Used     3.550000e+02
Critical Value (1%)    -3.448906e+00
Critical Value (5%)    -2.869716e+00
Critical Value (10%)   -2.571126e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -20.972097
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -6.853173e+00
```

```
P-Value                 1.674041e-09
# Lags Used             4.000000e+00
# Observations Used     3.590000e+02
Critical Value (1%)    -3.448697e+00
Critical Value (5%)    -2.869625e+00
Critical Value (10%)   -2.571077e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -18.957967
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -1.829869e+01
P-Value                 2.289063e-30
# Lags Used             0.000000e+00
# Observations Used     3.630000e+02
Critical Value (1%)    -3.448494e+00
Critical Value (5%)    -2.869535e+00
Critical Value (10%)   -2.571029e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -8.310567e+00
P-Value                 3.798940e-13
# Lags Used             5.000000e+00
# Observations Used     3.580000e+02
Critical Value (1%)    -3.448749e+00
Critical Value (5%)    -2.869647e+00
Critical Value (10%)   -2.571089e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -19.004528
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -1.667601e+01
P-Value                 1.530571e-29
# Lags Used             1.000000e+00
# Observations Used     3.620000e+02
Critical Value (1%)    -3.448544e+00
Critical Value (5%)    -2.869557e+00
Critical Value (10%)   -2.571041e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -9.293486e+00
P-Value                 1.162775e-15
# Lags Used             3.000000e+00
# Observations Used     3.600000e+02
Critical Value (1%)    -3.448646e+00
Critical Value (5%)    -2.869602e+00
Critical Value (10%)   -2.571065e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -1.820070e+01
P-Value                 2.406276e-30
# Lags Used             0.000000e+00
# Observations Used     3.630000e+02
Critical Value (1%)    -3.448494e+00
Critical Value (5%)    -2.869535e+00
Critical Value (10%)   -2.571029e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -20.673967
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -1.575759e+01
P-Value                 1.202422e-28
# Lags Used             1.000000e+00
# Observations Used     3.620000e+02
Critical Value (1%)    -3.448544e+00
Critical Value (5%)    -2.869557e+00
Critical Value (10%)   -2.571041e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -1.862336e+01
P-Value                 2.060393e-30
# Lags Used             0.000000e+00
# Observations Used     3.630000e+02
Critical Value (1%)    -3.448494e+00
Critical Value (5%)    -2.869535e+00
Critical Value (10%)   -2.571029e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -20.825826
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
```

```
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -21.129376
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -21.190128
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -20.580488
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -20.057417
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -7.968188e+00
P-Value                 2.827586e-12
# Lags Used             8.000000e+00
# Observations Used     3.550000e+02
Critical Value (1%)    -3.448906e+00
Critical Value (5%)    -2.869716e+00
Critical Value (10%)   -2.571126e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -20.263258
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -20.951633
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -19.296688
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -20.111680
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -1.403604e+01
P-Value                 3.368036e-26
# Lags Used             3.000000e+00
# Observations Used     3.600000e+02
Critical Value (1%)    -3.448646e+00
Critical Value (5%)    -2.869602e+00
Critical Value (10%)   -2.571065e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -1.301100e+01
P-Value                 2.568073e-24
# Lags Used             2.000000e+00
# Observations Used     3.610000e+02
Critical Value (1%)    -3.448595e+00
Critical Value (5%)    -2.869580e+00
Critical Value (10%)   -2.571053e+00
dtype: float64
Is the time series stationary? True
```

```
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -21.177564
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -1.302811e+01
P-Value                 2.376157e-24
# Lags Used             2.000000e+00
# Observations Used     3.610000e+02
Critical Value (1%)    -3.448595e+00
Critical Value (5%)    -2.869580e+00
Critical Value (10%)   -2.571053e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -20.451431
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -7.985504e+00
P-Value                 2.555473e-12
# Lags Used             8.000000e+00
# Observations Used     3.550000e+02
Critical Value (1%)    -3.448906e+00
Critical Value (5%)    -2.869716e+00
Critical Value (10%)   -2.571126e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -1.503771e+01
P-Value                 9.694212e-28
# Lags Used             1.000000e+00
# Observations Used     3.620000e+02
Critical Value (1%)    -3.448544e+00
Critical Value (5%)    -2.869557e+00
Critical Value (10%)   -2.571041e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -8.871350e+00
P-Value                 1.395223e-14
# Lags Used             5.000000e+00
# Observations Used     3.580000e+02
Critical Value (1%)    -3.448749e+00
Critical Value (5%)    -2.869647e+00
Critical Value (10%)   -2.571089e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -21.238390
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -21.837875
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -20.711441
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -7.902592e+00
P-Value                 4.146848e-12
# Lags Used             4.000000e+00
# Observations Used     3.590000e+02
Critical Value (1%)    -3.448697e+00
Critical Value (5%)    -2.869625e+00
Critical Value (10%)   -2.571077e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -1.042857e+01
P-Value                 1.633222e-18
# Lags Used             4.000000e+00
# Observations Used     3.590000e+02
Critical Value (1%)    -3.448697e+00
Critical Value (5%)    -2.869625e+00
Critical Value (10%)   -2.571077e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -20.653101
P-Value                   0.000000
# Lags Used               0.000000
```

```
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -19.794544
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -7.405633e+00
P-Value                 7.357744e-11
# Lags Used             6.000000e+00
# Observations Used     3.570000e+02
Critical Value (1%)    -3.448801e+00
Critical Value (5%)    -2.869670e+00
Critical Value (10%)   -2.571101e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -1.398303e+01
P-Value                 4.144550e-26
# Lags Used             3.000000e+00
# Observations Used     3.600000e+02
Critical Value (1%)    -3.448646e+00
Critical Value (5%)    -2.869602e+00
Critical Value (10%)   -2.571065e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic     -8.227295e+00
P-Value                 6.196857e-13
# Lags Used             6.000000e+00
# Observations Used     3.570000e+02
Critical Value (1%)    -3.448801e+00
Critical Value (5%)    -2.869670e+00
Critical Value (10%)   -2.571101e+00
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic      -22.333999
P-Value                   0.000000
# Lags Used               0.000000
# Observations Used     363.000000
Critical Value (1%)      -3.448494
Critical Value (5%)      -2.869535
Critical Value (10%)     -2.571029
dtype: float64
Is the time series stationary? True
Augmented Dickey-Fuller Test Results:
ADF Test Statistic       -4.063051
P-Value                   0.001114
# Lags Used              10.000000
# Observations Used     353.000000
Critical Value (1%)      -3.449011
Critical Value (5%)      -2.869763
Critical Value (10%)     -2.571151
dtype: float64
Is the time series stationary? True
```

In [8]:

```
# printing last df of interest: all cryptos seem to be stationary = good news
print("True means that the considered cc is stationary while False means the opposite…")
print("Houra. All cryptos seem to be stationary. Then, we can keep working with these time series.")

print(statio_cc)
```

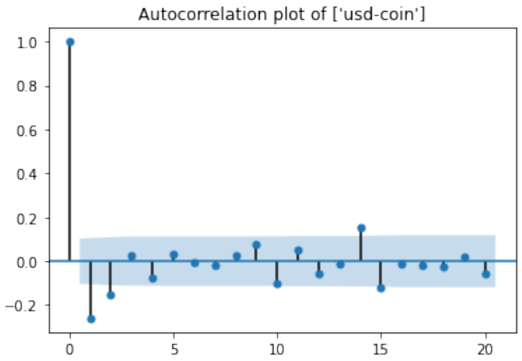
```
True means that the considered cc is stationary while False means
the opposite…
Houra. All cryptos seem to be stationary. Then, we can keep workin
g with these time series.
bitcoin                  True
ethereum                 True
ripple                   True
litecoin                 True
bitcoin-cash             True
chainlink                True
binancecoin              True
cardano                  True
stellar                  True
usd-coin                 True
bitcoin-cash-sv          True
eos                      True
nem                      True
tron                     True
okb                      True
tezos                    True
neo                      True
celsius-degree-token     True
theta-token              True
dash                     True
vechain                  True
havven                   True
huobi-token              True
iota                     True
zcash                    True
waves                    True
ethereum-classic         True
zilliqa                  True
dogecoin                 True
maker                    True
decred                   True
omisego                  True
ontology                 True
paxos-standard           True
```

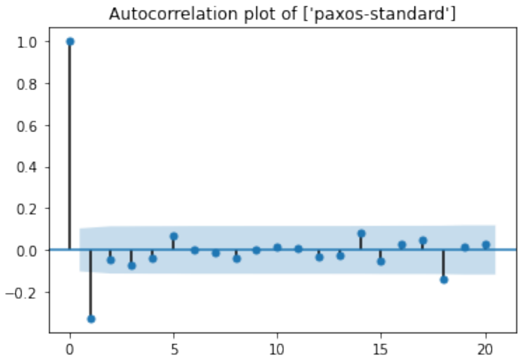
```
nexo                        True
basic-attention-token       True
digibyte                    True
0x                          True
true-usd                    True
qtum                        True
republic-protocol           True
swissborg                   True
icon                        True
loopring                    True
lisk                        True
kyber-network               True
quant-network               True
bitcoin-gold                True
maidsafecoin                True
vitae                       True
siacoin                     True
nano                        True
enjincoin                   True
Name: Stationary, dtype: bool
```

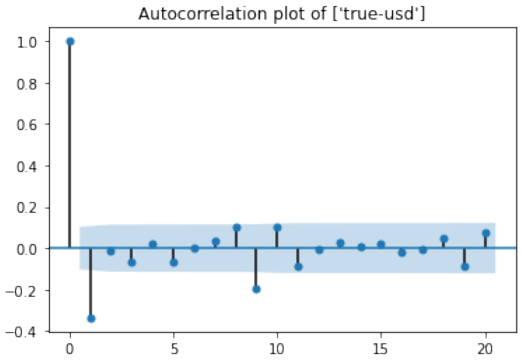
In [9]:

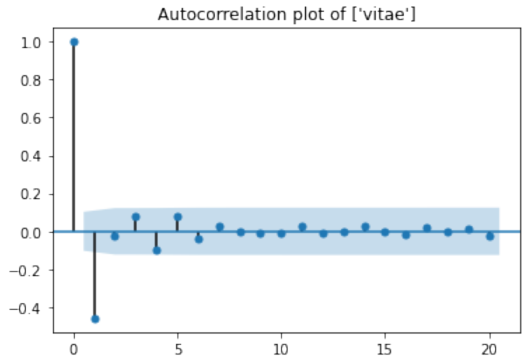
```
# Exploratory Analysis
# 4) Autocorrelation Test

# In this section, the autocorrelation of each cc is computed with 20 lags
# Autocorrelation plots will be shown for autocorrelated cc

#plot_acf(returns_data['bitcoin'], lags = 20, alpha = 0.05)
#ac_bitcoin = acf(returns_data['bitcoin'],nlags=20)

#for i in range(len(ac_bitcoin)):
 #   if (ac_bitcoin[i]<0.035):
  #      print(ac_bitcoin[i])

lags_names = [] # useful for below
for i in range(0,21):
    lags_names.append('lag' + str(i))

autocorr_cc = pd.DataFrame(np.zeros((21,nb_cc))) # creating autocorrelation matrix
autocorr_cc.columns = cc_names # adding cryptos names
autocorr_cc.index = lags_names # adding lags names
autocorrelated_cc = [] # creating a list to add the autocorrelated cryptos

for i in range(nb_cc): # making a loop to determine which cryptos are autocorrelated and add them to the list
    ccname = cc_names[i]
    autocorr_cc[ccname] = acf(returns_data[ccname],nlags = 20, fft=False)
    for j in range(len(autocorr_cc)):
        lagnb = 'lag'+ str(j)
        ac = autocorr_cc[ccname][lagnb]
        if ((ac >= 0.25 or ac <= -0.25) and ac != 1.):
            autocorrelated_cc.append([ccname])
del ccname

print(str(len(autocorrelated_cc))+" cryptocurrencies show autocorrelation with at least one of their last 20 lags.")

for i in range(len(autocorrelated_cc)): # autocorrelation plots for cryptos in the list
    ccname = autocorrelated_cc[i]
    plot_acf(returns_data[ccname], lags = 20, alpha = 0.05,title = 'Autocorrelation plot of '+ str(ccname))
    plt.savefig('Data_Analysis/acplt'+  str(ccname) + '.jpeg',transparent=True)
```

```
4 cryptocurrencies show autocorrelation with at least one of their last 20 lags.
```









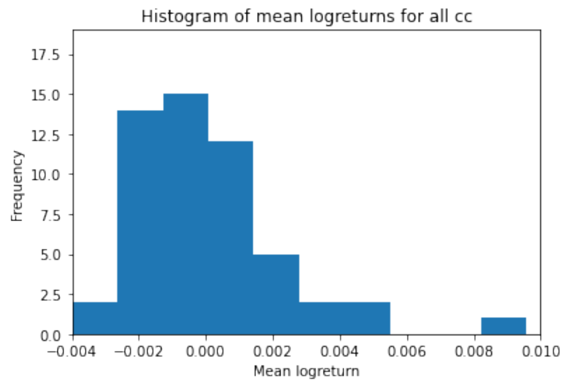
In [10]:

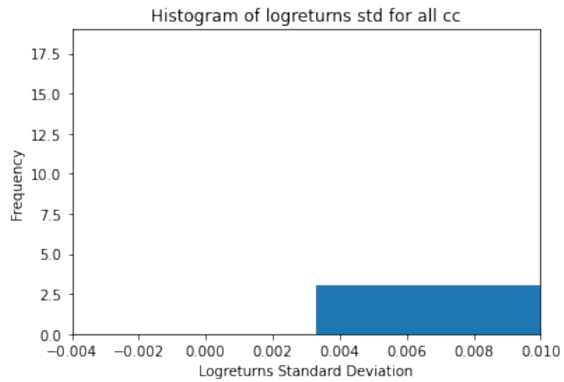
```
# Exploratory Analysis
# 5) Statistical summary of cc

# statistical summary of cryptos
summary_cc = returns_data.describe();
print(summary_cc)
summary_cc = summary_cc.transpose();

# plotting the histogram of logreturns mean for all cc
fig = plt.figure()
plt.hist(summary_cc['mean'])
plt.xlabel('Mean logreturn')
plt.ylabel('Frequency')
plt.title('Histogram of mean logreturns for all cc')
plt.xlim(-0.004, 0.010)
plt.ylim(0, 19)
plt.savefig('Data_Analysis/LogReturnsMean_Histogram.jpeg',transpare
nt=True)
plt.show()

# plotting the histogram of logreturns std for all cc
fig = plt.figure()
plt.hist(summary_cc['std'])
plt.xlabel('Logreturns Standard Deviation')
plt.ylabel('Frequency')
plt.title('Histogram of logreturns std for all cc')
plt.xlim(-0.004, 0.010)
plt.ylim(0, 19)
plt.savefig('Data_Analysis/LogReturnsStd_Histogram.jpeg',transparen
t=True)
plt.show()
```

```
            bitcoin      ethereum      ripple     litecoin   bitcoin-c
ash  \
count  364.000000  3.640000e+02  364.000000  364.000000    364.000
000
mean     0.001850  7.129810e-08   -0.001626    0.000961      0.000
923
std      0.035509  4.246187e-02    0.036729    0.048855      0.053
251
```

```
min     -0.150643 -1.753563e-01   -0.130690   -0.159533     -0.280
861
25%     -0.013290 -1.824251e-02   -0.016664   -0.025892     -0.020
805
50%      0.001023 -7.617596e-04   -0.001335   -0.000842     -0.002
254
75%      0.017332  1.888169e-02    0.011610    0.024168      0.023
530
max      0.159276  1.477720e-01    0.226891    0.261960      0.364
697

        chainlink  binancecoin     cardano     stellar    usd-coin
...  \
count  364.000000   364.000000  364.000000  364.000000  364.000000
...
mean     0.004992     0.002261   -0.000532   -0.002448    0.000010
...
std      0.065036     0.044104    0.046071    0.041849    0.003273
...
min     -0.208088    -0.154145   -0.180399   -0.130523   -0.019772
...
25%     -0.032302    -0.023766   -0.025893   -0.023299   -0.001420
...
50%     -0.001832    -0.000275    0.001846   -0.003477    0.000166
...
75%      0.032873     0.026335    0.022196    0.017211    0.001593
...
max      0.476072     0.176847    0.191987    0.258028    0.015020
...

         loopring        lisk  kyber-network  quant-network  bitco
in-gold  \
count  364.000000  364.000000     364.000000     364.000000     364
.000000
mean    -0.001591   -0.002260       0.000545       0.001428      -0
.002284
std      0.057431    0.041671       0.061801       0.077367       0
.041670
min     -0.192991   -0.182994      -0.201941      -0.231459      -0
.205101
25%     -0.030804   -0.024331      -0.030000      -0.042156      -0
.023119
50%      0.000114   -0.002035      -0.003271      -0.007291      -0
.000368
75%      0.024985    0.019425       0.027482       0.035867       0
.020749
max      0.409660    0.158836       0.358089       0.414337       0
.160100

       maidsafecoin       vitae     siacoin        nano   enjincoi
n
count    364.000000  364.000000  364.000000  364.000000  364.00000
0
mean      -0.001609   -0.000865   -0.001795   -0.000732    0.00218
6
std        0.056445    0.233123    0.043664    0.048957    0.08068
2
```

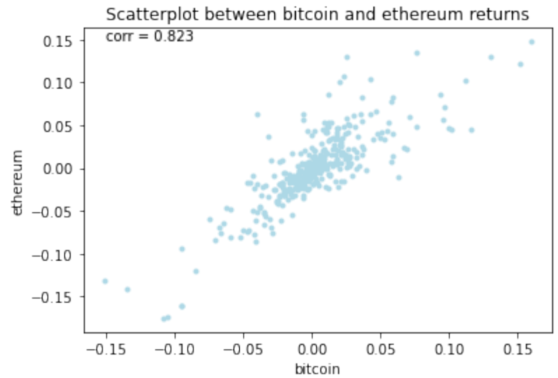
```
min       -0.443581   -2.517366   -0.192094   -0.144826   -0.20958
3
25%       -0.022910   -0.055540   -0.025023   -0.027692   -0.03368
9
50%       -0.001002   -0.003625   -0.000540   -0.002657   -0.00053
2
75%        0.023601    0.052783    0.022080    0.023005    0.02967
7
max        0.229984    2.522906    0.159691    0.211709    0.75121
9

[8 rows x 53 columns]
```



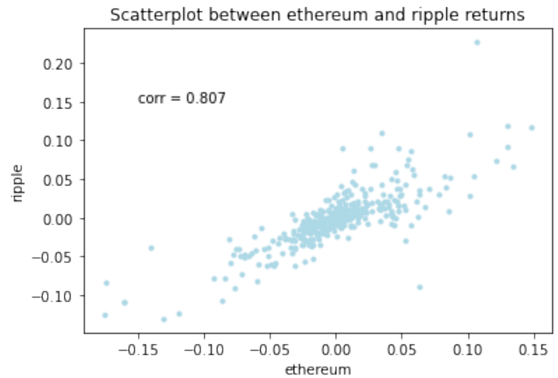


```
In [11]: # Exploratory Analysis
         # 6) Scatterplot of correlated cc

         # correlation matrix of all cryptos logreturns
         corr_matrix = pd.DataFrame(np.zeros((nb_cc,nb_cc)))
         corr_matrix.index = corr_matrix.columns = cc_names
         corr_matrix = returns_data.corr()
         corr_matrix = round(corr_matrix,3)

         # exporting it to csv and excel (useful for reporting)
         corr_matrix.to_csv('Data_Analysis/cc_corrmatrix.csv', index=True, h
         eader=True)
         corr_matrix.to_excel('Data_Analysis/cc_corrmatrix.xlsx',index=True,
         header=True)

         # scatter plot between highly (negatively or positively) correlated
         cryptos

         highcorr_matrix = pd.DataFrame(np.zeros((nb_cc,nb_cc)))
         highcorr_matrix.index = highcorr_matrix.columns = cc_names
         highcorr_cc = []
         print("16 pairs of cryptocurrencies are highly correlated in their
         log returns")
         for i in range(nb_cc): # just a for loop to determine what are the
         correlated cryptos
             abc = cc_names[i]
             for j in range(nb_cc):
                 if (i<j):
                     dcb = cc_names[j]
                     corr = corr_matrix[abc][dcb]
                     if ((corr >= 0.8 or corr <= -0.8) and corr != 1 ):
                         highcorr_matrix[abc][dcb] = corr_matrix[abc][dcb]
                         print(abc + ' is strongly correlated to ' + dcb)
                         corr_str = str(round(corr,3))
                         fig = plt.figure()
                         ax = fig.add_subplot(111)
                         ax.scatter(returns_data[abc],returns_data[dcb], col
         or='lightblue', marker='.')
                         ax.set(title='Scatterplot between '+abc+' and '+dcb
         +' returns',ylabel=dcb,xlabel=abc)
                         ax.annotate('corr = '+corr_str,xy=(-0.15,0.15))
                         plt.savefig('Data_Analysis/sc_'+ abc + '_' + dcb +'
         .jpeg',transparent=True)
                         plt.show()
                         highcorr_cc.append([abc,dcb,corr])

         highcorr_cc = pd.DataFrame(highcorr_cc)
         highcorr_cc.columns = ["Crypto 1","Crypto 2","Pair correlation"]
         highcorr_cc.to_csv('Data_Analysis/highcorr_cc.csv', index=True, hea
         der=True)
         #sns.regplot(x=abc,y=dcb,ci = None,data=ret_cc)
```

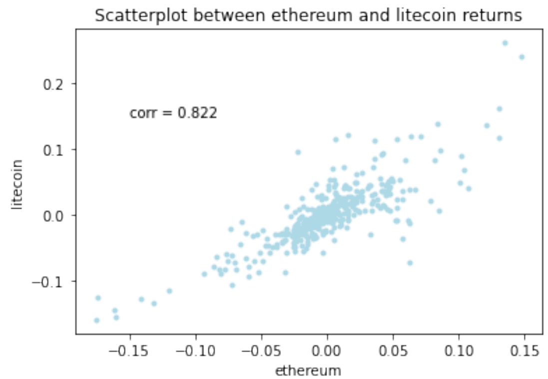
```
16 pairs of cryptocurrencies are highly correlated in their log re
turns
bitcoin is strongly correlated to ethereum
```



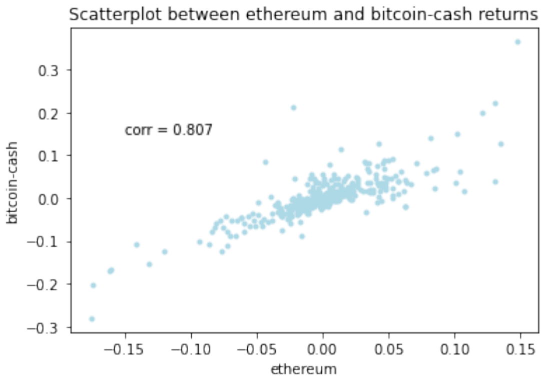

```
ethereum is strongly correlated to ripple
```



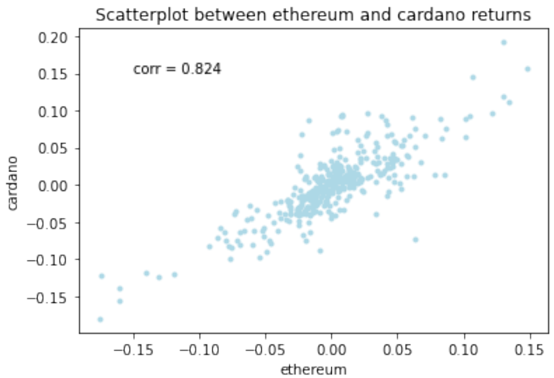

```
ethereum is strongly correlated to litecoin
```



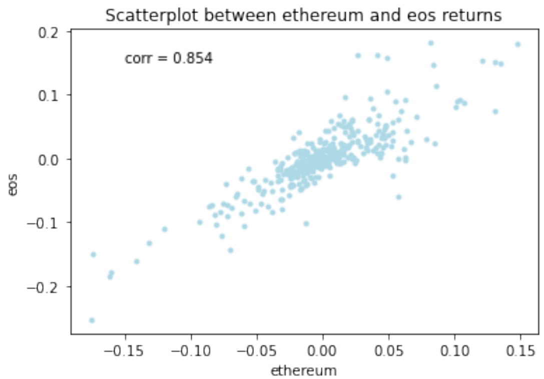

```
ethereum is strongly correlated to bitcoin-cash
```



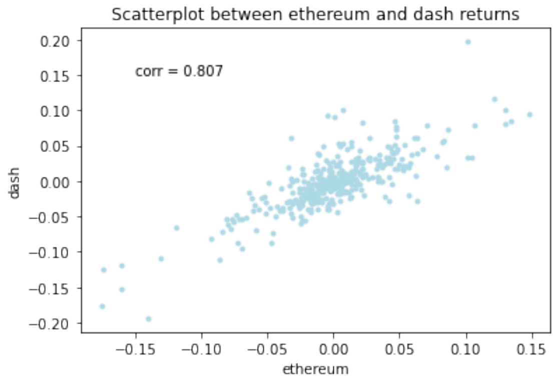
ethereum is strongly correlated to cardano



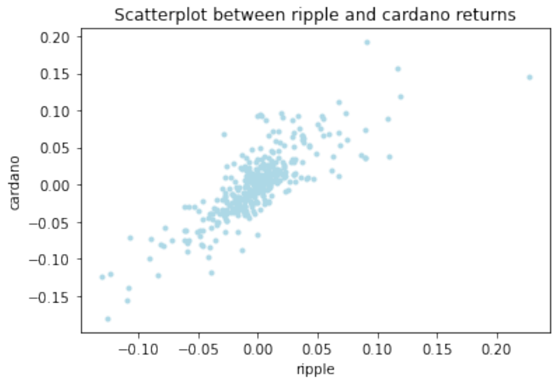
ethereum is strongly correlated to eos



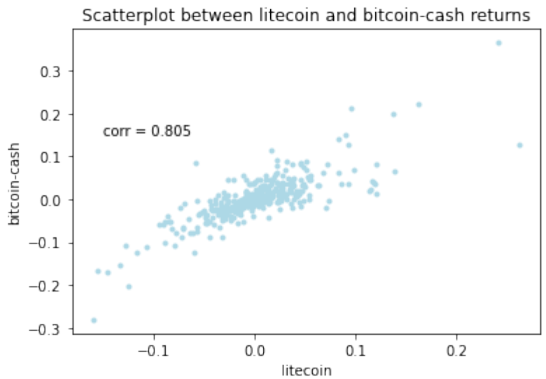
ethereum is strongly correlated to dash



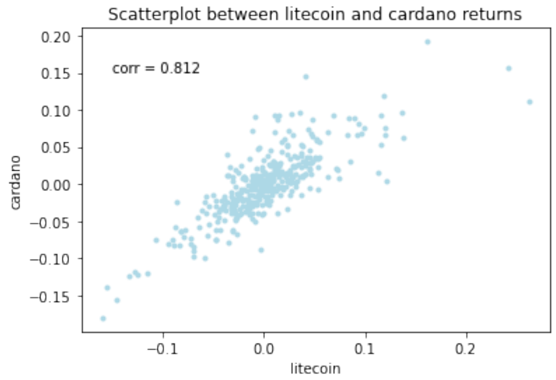
ripple is strongly correlated to cardano



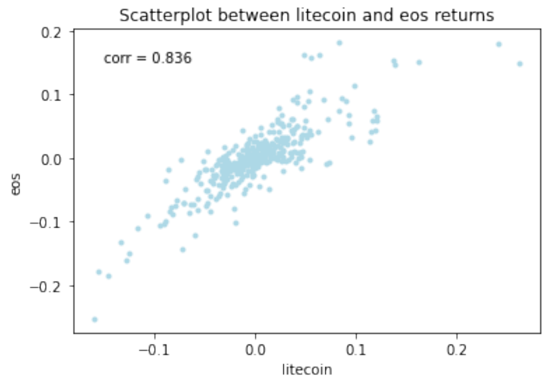
litecoin is strongly correlated to bitcoin-cash



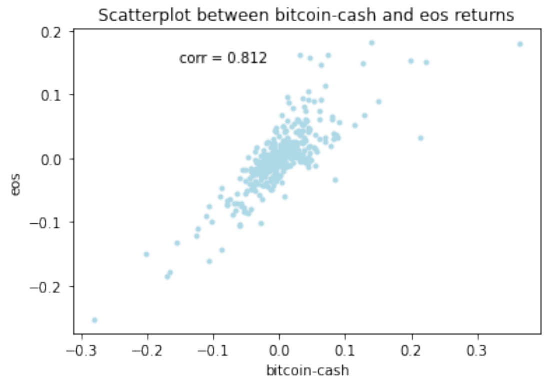
litecoin is strongly correlated to cardano



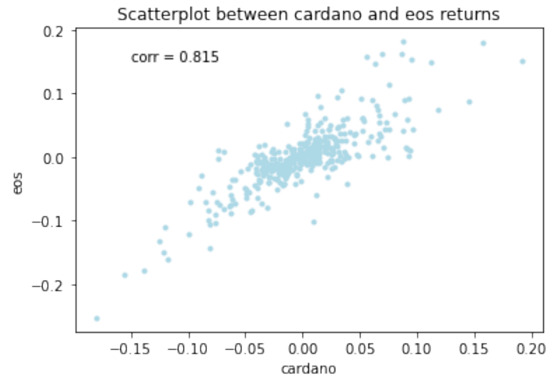
litecoin is strongly correlated to eos



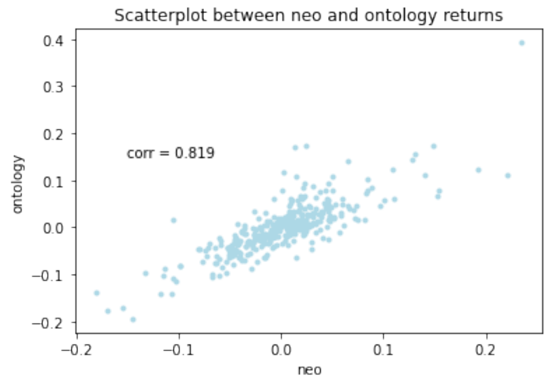
bitcoin-cash is strongly correlated to eos



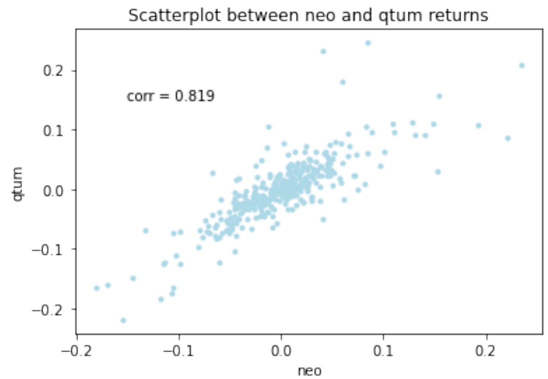
cardano is strongly correlated to eos



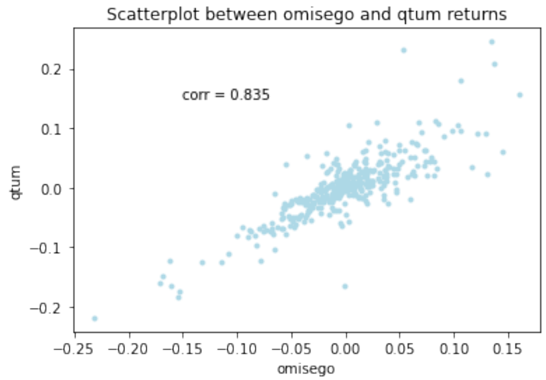
neo is strongly correlated to ontology



neo is strongly correlated to qtum



omisego is strongly correlated to qtum

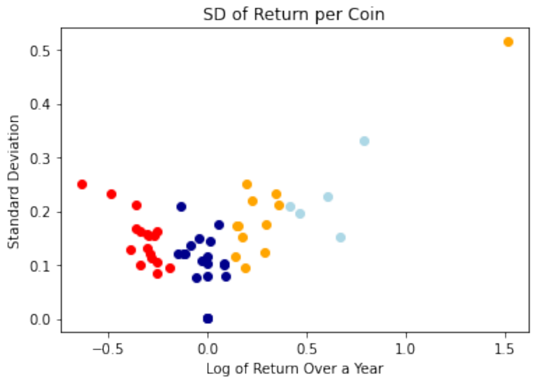


In [12]:
```
highcorr_cc
```

Out[12]:

	Crypto 1	Crypto 2	Pair correlation
0	bitcoin	ethereum	0.823
1	ethereum	ripple	0.807
2	ethereum	litecoin	0.822
3	ethereum	bitcoin-cash	0.807
4	ethereum	cardano	0.824
5	ethereum	eos	0.854
6	ethereum	dash	0.807
7	ripple	cardano	0.814
8	litecoin	bitcoin-cash	0.805
9	litecoin	cardano	0.812
10	litecoin	eos	0.836
11	bitcoin-cash	eos	0.812
12	cardano	eos	0.815
13	neo	ontology	0.819
14	neo	qtum	0.819
15	omisego	qtum	0.835

In [13]:
```
#Preparing the data
log_frame = np.log10(prices_data)
log_return = list()
volatility = list()
for i in range (0, prices_data.shape[1]):
    log_return.append(i)
    volatility.append(i)
    log_return[i]= log_frame.iloc[364][i] - log_frame.iloc[0][i]
    aux = np.std(log_frame.iloc[:,[i]]) #change here to get other d
iagram
    volatility[i] = aux[0]

##### Creating the Clusters
db_vol_ret =pd.DataFrame(data=log_return,columns=['return'])
db_vol_ret['volatillity'] = volatility
n_kmeans = 5
km = KMeans(n_clusters=n_kmeans)
y_predicted = km.fit_predict(db_vol_ret)
cols=prices_data.columns
db_vol_ret.index=cols
db_vol_ret['cluster'] = y_predicted

##plotting the result
df = db_vol_ret
df1= df[df.cluster == 0]
df2= df[df.cluster == 1]
df3= df[df.cluster == 2]
df4= df[df.cluster == 3]
df5= df[df.cluster == 4]

plt.scatter(df1['return'],df1['volatillity'],color='lightblue')
plt.scatter(df2['return'],df2['volatillity'],color='red')
plt.scatter(df3['return'],df3['volatillity'],color='orange')
plt.scatter(df4['return'],df4['volatillity'],color='darkblue')
plt.scatter(df5['return'],df5['volatillity'],color='orange')
plt.xlabel('Log of Return Over a Year')
plt.ylabel('Standard Deviation')
plt.title('SD of Return per Coin')
plt.savefig('Data_Analysis/K-Means_Explorative.jpeg',transparent=Tr
ue)
```



In [14]:
```
# Clustering

# 1) Rolling windows computation for each variable
# 2) PCA: Principal Components Analysis followed by a k-means clust
ering algorithm on its components
    # 2.a) Application on a single window
    # 2.b) Application on all rolling windows
# 3) LFM: Linear Factor Model followed by a k-means clustering algo
rithm on its factors
    # 3.a) Application on a single window
    # 3.b) Application on all rolling windows
```

In [15]:
```
# Clustering

# 1) Rolling windows computation for each variable

window_size = 25 # determining the size of the rolling windows we a
re going to compute

# Mean
window_mean = returns_data.rolling(window=window_size).mean()
window_mean = window_mean.dropna(axis=0)

# Variance
window_var = returns_data.rolling(window=window_size).var()
window_var = window_var.dropna(axis=0)

# Skewness (IMPORTANT: need to take rolling window of 3 to calculat
e it)
window_skew = returns_data.rolling(window=window_size).skew()
window_skew = window_skew.dropna(axis=0)

# Kurtosis (IMPORTANT: need to take rolling window of 4 to calculat
e it)
window_kurt = returns_data.rolling(window=window_size).kurt()
window_kurt = window_kurt.dropna(axis=0)

# Quantile 0.05
window_005_quantile = returns_data.rolling(window=window_size).quan
tile(0.05)
window_005_quantile = window_005_quantile.dropna(axis=0)

# Quantile 0.10
window_010_quantile = returns_data.rolling(window=window_size).quan
tile(0.10)
window_010_quantile = window_010_quantile.dropna(axis=0)

# Quantile 0.15
window_015_quantile = returns_data.rolling(window=window_size).quan
tile(0.15)
window_015_quantile = window_015_quantile.dropna(axis=0)

# Quantile 0.85
window_085_quantile = returns_data.rolling(window=window_size).quan
tile(0.85)
window_085_quantile = window_085_quantile.dropna(axis=0)

# Quantile 0.90
window_090_quantile = returns_data.rolling(window=window_size).quan
tile(0.90)
window_090_quantile = window_090_quantile.dropna(axis=0)

# Quantile 0.95
window_095_quantile = returns_data.rolling(window=window_size).quan
tile(0.95)
window_095_quantile = window_095_quantile.dropna(axis=0)
```

In [16]:

```python
# Clustering

# 2) PCA: Principal Components Analysis followed by a k-means clust
ering algorithm on its components
    # 2.a) Application on a single window

# Initialization: Df creation for the first window
frames = [window_mean.iloc[0], window_var.iloc[0],window_skew.iloc[
0],
         window_kurt.iloc[0], window_005_quantile.iloc[0], window_
010_quantile.iloc[0],
         window_015_quantile.iloc[0], window_085_quantile.iloc[0],
window_090_quantile.iloc[0],
         window_095_quantile.iloc[0]]
frames = pd.concat(frames, axis=1)

scaler = MinMaxScaler()
scaler.fit(frames)
frames_norm = pd.DataFrame(scaler.transform(frames),index=frames.in
dex)
frames_norm.columns = ("Mean","Variance","Skewness","Kurtosis","Q.0
5","Q.10","Q.15","Q.85","Q.90","Q.95")
frames_norm # final dataframe of the different variables (moments +
quantiles) for each cc
```

Out[16]:

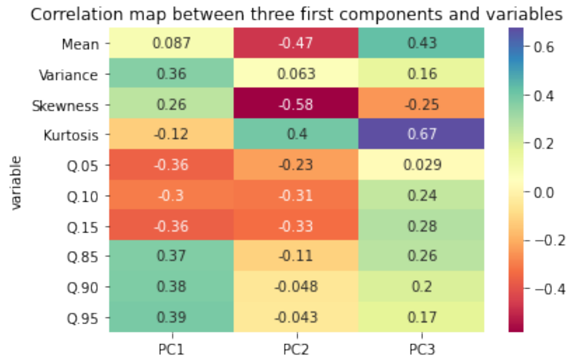
	Mean	Variance	Skewness	Kurtosis	Q.05	Q.10	Q.15	Q
bitcoin	0.452785	0.036397	0.331646	0.533070	0.866977	0.857108	0.795706	0.126
ethereum	0.373463	0.126541	0.490807	0.381679	0.717934	0.703016	0.617143	0.228
ripple	0.394568	0.045151	0.243746	0.543274	0.828487	0.798767	0.754050	0.152
litecoin	0.555107	0.127172	0.459584	0.269824	0.743675	0.729847	0.569958	0.324
bitcoin-cash	0.350650	0.117031	0.310603	0.456292	0.745819	0.704867	0.573326	0.252
chainlink	0.857243	0.285613	0.766090	0.086631	0.688798	0.631513	0.457865	0.571
binancecoin	0.557537	0.078890	0.476226	0.248050	0.750718	0.795175	0.787277	0.247
cardano	0.517659	0.135110	0.411017	0.281094	0.687142	0.687088	0.667592	0.344
stellar	0.395852	0.063071	0.277525	0.586199	0.842281	0.811095	0.695320	0.152
usd-coin	0.484375	0.000076	0.519645	0.000000	0.994371	0.990759	0.987988	0.006
bitcoin-cash-sv	0.379381	0.098485	0.524917	0.378820	0.720022	0.702363	0.715194	0.141
eos	0.450602	0.140224	0.250159	0.512511	0.714111	0.656757	0.630455	0.273
nem	0.365815	0.056671	0.000000	1.000000	0.864951	0.818855	0.723256	0.085
tron	0.770416	0.169139	0.783793	0.254763	0.672338	0.677591	0.708745	0.477
okb	0.440298	0.063585	0.401350	0.249096	0.810168	0.783890	0.679960	0.232
tezos	0.393754	0.062071	0.415455	0.227131	0.726126	0.716814	0.705144	0.167
neo	0.495926	0.129943	0.506314	0.230482	0.718625	0.699737	0.566782	0.322
celsius-								
degree-token	0.570627	0.396070	0.686168	0.032444	0.530491	0.349559	0.135969	0.781
theta-token	0.622234	0.097213	0.400937	0.223484	0.750082	0.765116	0.709763	0.344
dash	0.414075	0.080756	0.132420	0.653051	0.787047	0.748551	0.687504	0.203
vechain	0.554601	0.142732	0.407833	0.333851	0.790151	0.774625	0.663902	0.414
havven	0.665166	1.000000	0.739511	0.291345	0.000000	0.463509	0.362429	1.000
huobi-token	0.468196	0.019779	0.281235	0.390303	0.825736	0.929264	0.914674	0.108
iota	0.314220	0.109169	0.283378	0.492814	0.711080	0.725876	0.606744	0.187
zcash	0.415513	0.052098	0.489467	0.190111	0.788733	0.760545	0.708519	0.160
waves	0.373286	0.097996	0.620151	0.195245	0.660518	0.610567	0.651526	0.173
ethereum-classic	0.368525	0.088621	0.299894	0.191444	0.647977	0.620735	0.534614	0.239
zilliqa	0.582904	0.164631	0.427396	0.216618	0.636559	0.752141	0.596247	0.403
dogecoin	0.381843	0.013755	0.338410	0.410453	0.918119	0.886395	0.830606	0.061
maker	0.518546	0.081290	0.464392	0.304803	0.784596	0.805196	0.685902	0.202
decred	0.511717	0.092068	0.192923	0.435792	0.719274	0.697971	0.717630	0.235
omisego	0.418226	0.118135	0.244667	0.445809	0.695275	0.712865	0.609747	0.243
ontology	0.523413	0.164460	0.294581	0.482852	0.724299	0.711107	0.685501	0.383
paxos-standard	0.479212	0.000756	0.661242	0.201102	0.983101	0.974438	0.961012	0.008
nexo	0.261813	0.107989	0.677696	0.104169	0.695444	0.661039	0.498435	0.271
basic-attention-token	0.472470	0.076096	0.227172	0.364998	0.707767	0.791544	0.696162	0.205
digibyte	0.415516	0.087528	0.375215	0.495065	0.806204	0.759969	0.679434	0.142
0x	0.466866	0.086281	0.389634	0.288530	0.818995	0.751255	0.613172	0.247
true-usd	0.481884	0.000000	0.560671	0.336334	1.000000	1.000000	1.000000	0.000
qtum	0.450330	0.105988	0.291503	0.350532	0.700734	0.676025	0.573238	0.248
republic-protocol	0.429244	0.147351	0.223370	0.515597	0.707584	0.682968	0.593466	0.324
swissborg	0.438780	0.153354	0.707793	0.288496	0.684259	0.578432	0.322368	0.177
icon	0.449165	0.115217	0.252734	0.499565	0.806919	0.745133	0.640403	0.317
loopring	1.000000	0.571763	1.000000	0.625049	0.521136	0.662301	0.515390	0.639
lisk	0.431322	0.060039	0.304863	0.353564	0.796203	0.776220	0.645435	0.169
kyber-network	0.393132	0.130373	0.331460	0.294902	0.675333	0.606811	0.482169	0.313
quant-network	0.307977	0.345017	0.743336	0.086841	0.464331	0.358161	0.072315	0.474
bitcoin-gold	0.364782	0.041147	0.475113	0.222223	0.824513	0.784611	0.710499	0.156
maidsafecoin	0.400714	0.152884	0.335370	0.229302	0.502005	0.685862	0.547440	0.336

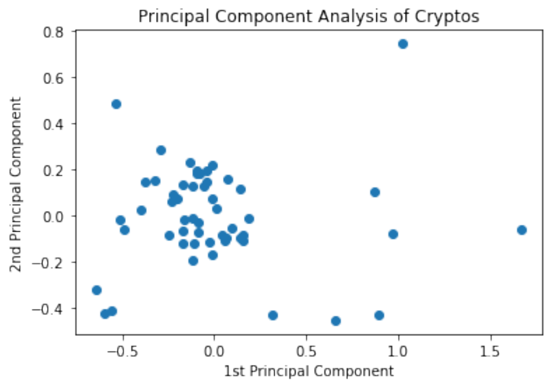
vitae	0.000000	0.642389	0.459100	0.254114	0.096983	0.000000	0.000000	0.29
siacoin	0.456635	0.077417	0.483797	0.133857	0.732783	0.721068	0.615132	0.26
nano	0.598563	0.103888	0.312633	0.374919	0.756824	0.791224	0.752652	0.32
enjincoin	0.422799	0.115993	0.234280	0.437748	0.657322	0.716010	0.605453	0.26

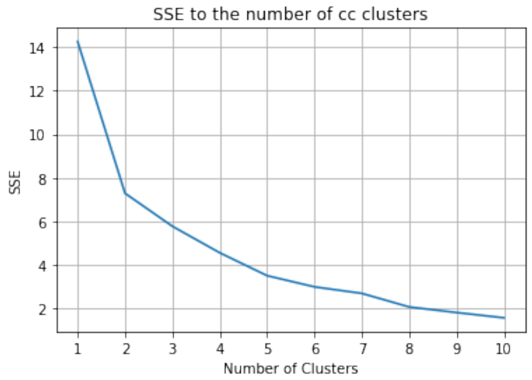
In [17]:

```python
# Clustering

# 2) PCA: Principal Components Analysis followed by a k-means clustering algorithm on its components
    # 2.a) Application on a single window

# PCA is firstly made
pca = PCA(n_components=3, random_state=123) # calling PCA function
pca.fit(frames_norm)
print(pca.explained_variance_)
principal_components = pd.DataFrame(pca.fit_transform(frames_norm),
                                    index=frames_norm.index)

# from these variances, the number of components is determined
n_components = 3 #as the third component still explains more than 10% of the variance

loadings = pca.components_ # inspired by https://reneshbedre.github.io/blog/pca_3d.html
num_pc = pca.n_features_
pc_list = ["PC"+str(i) for i in list(range(1, num_pc+1))]
loadings_df = pd.DataFrame.from_dict(dict(zip(pc_list, loadings)))
loadings_df['variable'] = frames_norm.columns.values
loadings_df = loadings_df.set_index('variable')
loadings_df
ax = sns.heatmap(loadings_df, annot=True, cmap='Spectral')
plt.title('Correlation map between three first components and variables')
plt.savefig('Clusters/pca_corrmap.jpeg',transparent=True)
plt.show()

# projecting log returns on two first components
plt.scatter(principal_components.iloc[:,0], principal_components.iloc[:,1])
plt.xlabel('1st Principal Component',fontsize=10)
plt.ylabel('2nd Principal Component',fontsize=10)
plt.title("Principal Component Analysis of Cryptos",fontsize=12)
plt.savefig('Clusters/pca.jpeg',transparent=True)
plt.show()

# Then, the k-means algorithm may be applied on this time-step

# To check the number of clusters needed
sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, init = "random", n_init=10, max_iter = 300,
                    random_state = 123)
    kmeans.fit(principal_components)
    sse.append(kmeans.inertia_)

plt.plot(range(1, 11), sse)
plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title("SSE to the number of cc clusters",fontsize=12)
plt.grid(True)
plt.savefig('Clusters/pca_nb_kmeans.jpeg',transparent=True)
plt.show()

print("By looking through the different windows, taking 4 clusters would be on average appropriate ")
n_kmeans = 4
kmeans = KMeans(n_clusters=n_kmeans,random_state=123) # defining the function kmeans with the
```

```
[0.18880196 0.04837439 0.03711979]
```



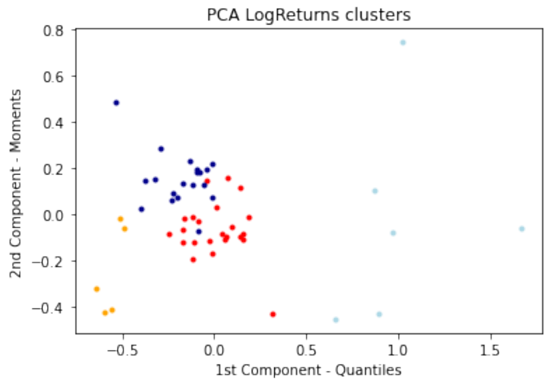




By looking through the different windows, taking 4 clusters would be on average appropriate

In [18]:
```
# clustering the first rolling window
print("Let's try to divide the first window into four clusters:")
label = kmeans.fit_predict(principal_components)
#print(label)

filtered_label0 = principal_components[label == 0]
filtered_label1 = principal_components[label == 1]
filtered_label2 = principal_components[label == 2]
filtered_label3 = principal_components[label == 3]

#Plotting the results
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(filtered_label0[0],filtered_label0[1], color='red', mark
er='.')
ax.scatter(filtered_label1[0],filtered_label1[1], color='lightblue'
, marker='.')
ax.scatter(filtered_label2[0],filtered_label2[1], color='orange', m
arker='.')
ax.scatter(filtered_label3[0],filtered_label3[1], color='darkblue',
marker='.')
ax.set(title='PCA LogReturns clusters',ylabel='2nd Component - Mome
nts',xlabel='1st Component - Quantiles')
plt.savefig('Clusters/PCA_clusters.jpeg',transparent=True)
plt.show()
```

Let's try to divide the first window into four clusters:



In [19]:
```
del filtered_label0,filtered_label1,filtered_label2,filtered_label3
```

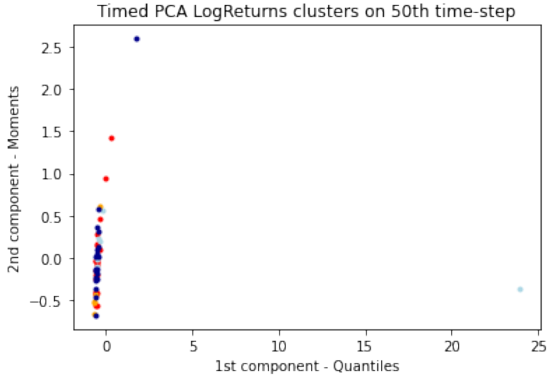
In [20]:
```python
# Clustering

# 2) PCA: Principal Components Analysis followed by a k-means clust
ering algorithm on its components
    # 2.b) Application on all rolling windows

kmeans = KMeans(n_clusters=n_kmeans,random_state=123) # defining th
e function kmeans with the
# optimal number of clusters found in last part

# Loop on the windows
labels = [] # creating a labels df for later
nt = len(window_mean);nb_cc; # time indices
timed_PCA = np.zeros((nt,nb_cc,n_components)) # 3D matrix for the P
CA components at each time-step (days*nb_cc*nb_pc)
timed_PCA_label = np.zeros((nt,nb_cc)) # matrix for labels= in whic
h cluster is a cc
#df = frames_norm

for i in range(nt):
    win_frames = []
    win_frames = [window_mean.iloc[i], window_var.iloc[i],
                  window_skew.iloc[i], window_kurt.iloc[i],
                  window_005_quantile.iloc[i], window_010_quantile.
iloc[i],
                  window_015_quantile.iloc[i], window_085_quantile.
iloc[i], window_090_quantile.iloc[i],
                  window_095_quantile.iloc[i]] # computation of the
rolling windows through time
    win_frames = pd.concat(win_frames, axis=1)
    win_frames_norm = pd.DataFrame(scaler.transform(win_frames),
                                   index=win_frames.index)
    # all rolling windows are created
    pca.fit(win_frames_norm) # the pca is initialized
    timed_PCA[i] = pca.fit_transform(win_frames_norm) # pca is done
    timed_PCA_label[i] = kmeans.fit_predict(timed_PCA[i]) # cluster
s labels are assigned to each cc

timed_PCA_label = pd.DataFrame(timed_PCA_label)
timed_PCA_label.columns = cc_names

# instead of analyzing each time-step, let's analyze the distributi
on of clusters assignation for each cc
index = pd.DataFrame([frames.index])
table = []
for i in index.transpose()[0]:
    y = timed_PCA_label[i].value_counts(normalize=True)
    table.append(y)
pca_km_table = pd.DataFrame(table)
del y,table;

print(pca_km_table) # printing table of interest
```

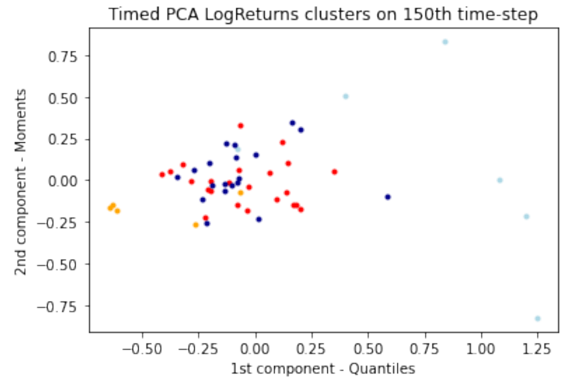
	0.0	1.0	2.0	3.0
bitcoin	0.458824	0.158824	0.170588	0.211765
ethereum	0.461765	0.173529	0.185294	0.179412
ripple	0.447059	0.167647	0.191176	0.194118
litecoin	0.458824	0.170588	0.191176	0.179412
bitcoin-cash	0.552941	0.167647	0.141176	0.138235
chainlink	0.408824	0.167647	0.235294	0.188235
binancecoin	0.514706	0.152941	0.185294	0.147059
cardano	0.494118	0.147059	0.208824	0.150000
stellar	0.517647	0.164706	0.161765	0.155882
usd-coin	0.370588	0.123529	0.208824	0.297059
bitcoin-cash-sv	0.458824	0.191176	0.205882	0.144118
eos	0.417647	0.191176	0.211765	0.179412
nem	0.461765	0.214706	0.152941	0.170588
tron	0.488235	0.161765	0.205882	0.144118
okb	0.391176	0.214706	0.214706	0.179412
tezos	0.394118	0.191176	0.294118	0.120588
neo	0.488235	0.164706	0.208824	0.138235
celsius-degree-token	0.238235	0.232353	0.317647	0.211765
theta-token	0.420588	0.188235	0.232353	0.158824
dash	0.500000	0.173529	0.179412	0.147059
vechain	0.405882	0.194118	0.214706	0.185294
havven	0.155882	0.300000	0.291176	0.252941
huobi-token	0.414706	0.167647	0.208824	0.208824
iota	0.535294	0.144118	0.185294	0.135294
zcash	0.505882	0.167647	0.176471	0.150000
waves	0.485294	0.150000	0.208824	0.155882
ethereum-classic	0.467647	0.211765	0.173529	0.147059
zilliqa	0.441176	0.173529	0.252941	0.132353
dogecoin	0.488235	0.173529	0.176471	0.161765
maker	0.467647	0.214706	0.167647	0.150000
decred	0.502941	0.188235	0.158824	0.150000
omisego	0.514706	0.161765	0.200000	0.123529
ontology	0.461765	0.179412	0.200000	0.158824
paxos-standard	0.373529	0.123529	0.205882	0.297059
nexo	0.352941	0.214706	0.238235	0.194118
basic-attention-token	0.408824	0.179412	0.244118	0.167647
digibyte	0.479412	0.170588	0.202941	0.147059
0x	0.461765	0.197059	0.200000	0.141176
true-usd	0.361765	0.129412	0.211765	0.297059
qtum	0.494118	0.197059	0.161765	0.147059
republic-protocol	0.297059	0.244118	0.264706	0.194118
swissborg	0.329412	0.247059	0.244118	0.179412
icon	0.494118	0.200000	0.164706	0.141176
loopring	0.391176	0.223529	0.214706	0.170588
lisk	0.476471	0.191176	0.182353	0.150000
kyber-network	0.276471	0.258824	0.258824	0.205882
quant-network	0.194118	0.273529	0.282353	0.250000
bitcoin-gold	0.602941	0.132353	0.164706	0.100000
maidsafecoin	0.461765	0.220588	0.197059	0.120588
vitae	0.108824	0.438235	0.241176	0.211765
siacoin	0.500000	0.173529	0.185294	0.141176
nano	0.452941	0.191176	0.208824	0.147059
enjincoin	0.294118	0.264706	0.279412	0.161765

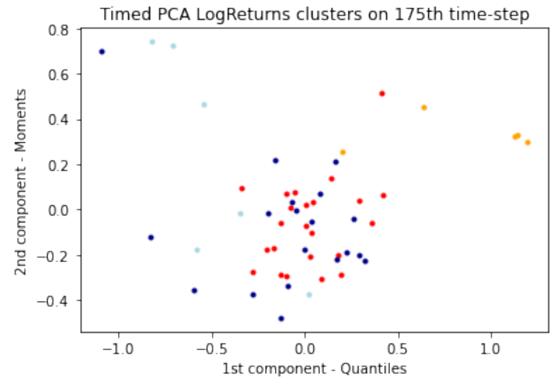
In [21]:

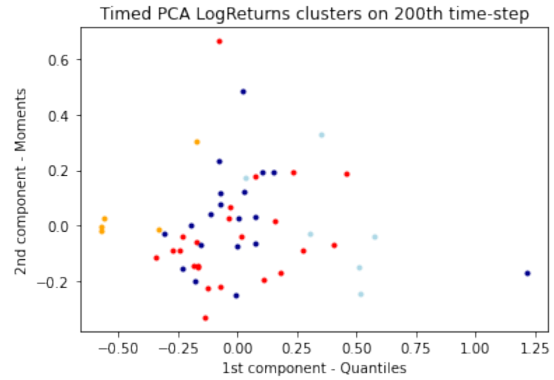
```
# Plotting the results for some rolling windows
print("Let's see how cc log returns are clustered for some rolling
windows")
rw_choice = [50,150,175,200,300]
rw_choice

for r in rw_choice:
    selected_ts = r
    filtered_data = timed_PCA[selected_ts]
    filtered_label0 = filtered_data[label == 0]
    filtered_label1 = filtered_data[label == 1]
    filtered_label2 = filtered_data[label == 2]
    filtered_label3 = filtered_data[label == 3]

        #Plotting the results
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.scatter(filtered_label0[:,0],filtered_label0[:,1], color='re
d', marker='.')
    ax.scatter(filtered_label1[:,0],filtered_label1[:,1], color='li
ghtblue', marker='.')
    ax.scatter(filtered_label2[:,0],filtered_label2[:,1], color='or
ange', marker='.')
    ax.scatter(filtered_label3[:,0],filtered_label3[:,1], color='da
rkblue', marker='.')
    ax.set(title='Timed PCA LogReturns clusters on '+str(selected_t
s) + 'th time-step',ylabel='2nd component - Moments',xlabel='1st co
mponent - Quantiles')
    plt.savefig('Clusters/timedpca_clusters'+str(selected_ts)+'rw.j
peg',transparent=True)
    plt.show()
```

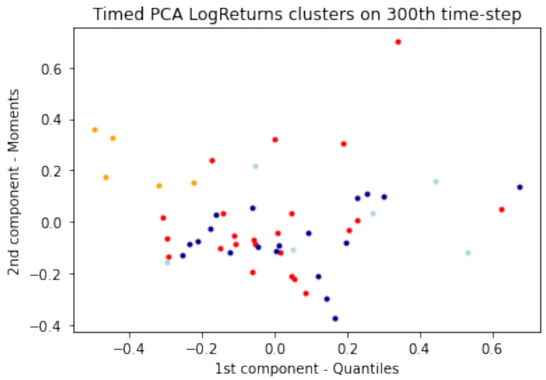
```
Let's see how cc log returns are clustered for some rolling window
s
```











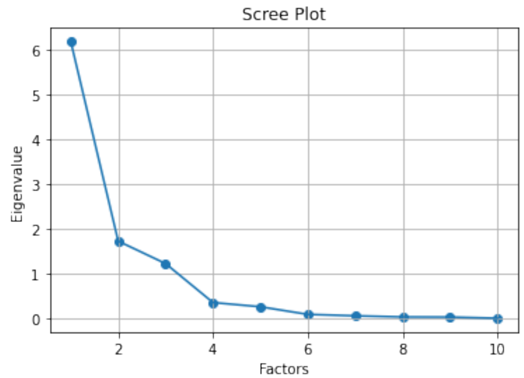
In [22]:
```
del r, rw_choice,filtered_label0,filtered_label1,filtered_label2,la
bel
```

In [23]:
```
# Distribution of cc log returns in clusters
pca_km_table = pca_km_table.round(2)

print("Here is the distribution of cc log returns in each cluster:"
)
print(pca_km_table)

pca_km_table1 = pd.DataFrame(pca_km_table[:17])
pca_km_table2 = pd.DataFrame(pca_km_table[17:34])
pca_km_table3 = pd.DataFrame(pca_km_table[34:])


pca_km_table1.to_csv('Clusters/pca_km_table1.csv', index=True, head
er=True)
pca_km_table2.to_csv('Clusters/pca_km_table2.csv', index=True, head
er=True)
pca_km_table3.to_csv('Clusters/pca_km_table3.csv', index=True, head
er=True)
```

Here is the distribution of cc log returns in each cluster:

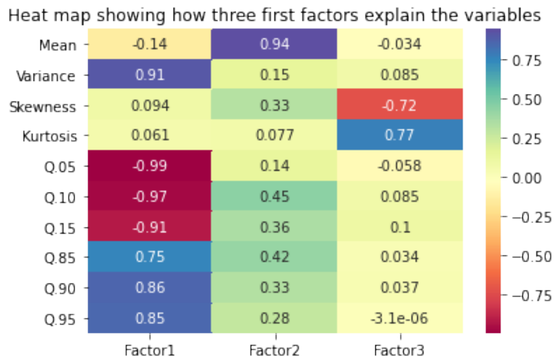
	0.0	1.0	2.0	3.0
bitcoin	0.46	0.16	0.17	0.21
ethereum	0.46	0.17	0.19	0.18
ripple	0.45	0.17	0.19	0.19
litecoin	0.46	0.17	0.19	0.18
bitcoin-cash	0.55	0.17	0.14	0.14
chainlink	0.41	0.17	0.24	0.19
binancecoin	0.51	0.15	0.19	0.15
cardano	0.49	0.15	0.21	0.15
stellar	0.52	0.16	0.16	0.16
usd-coin	0.37	0.12	0.21	0.30
bitcoin-cash-sv	0.46	0.19	0.21	0.14
eos	0.42	0.19	0.21	0.18
nem	0.46	0.21	0.15	0.17
tron	0.49	0.16	0.21	0.14
okb	0.39	0.21	0.21	0.18
tezos	0.39	0.19	0.29	0.12
neo	0.49	0.16	0.21	0.14
celsius-degree-token	0.24	0.23	0.32	0.21
theta-token	0.42	0.19	0.23	0.16
dash	0.50	0.17	0.18	0.15
vechain	0.41	0.19	0.21	0.19
havven	0.16	0.30	0.29	0.25
huobi-token	0.41	0.17	0.21	0.21
iota	0.54	0.14	0.19	0.14
zcash	0.51	0.17	0.18	0.15
waves	0.49	0.15	0.21	0.16
ethereum-classic	0.47	0.21	0.17	0.15
zilliqa	0.44	0.17	0.25	0.13
dogecoin	0.49	0.17	0.18	0.16
maker	0.47	0.21	0.17	0.15
decred	0.50	0.19	0.16	0.15
omisego	0.51	0.16	0.20	0.12
ontology	0.46	0.18	0.20	0.16
paxos-standard	0.37	0.12	0.21	0.30
nexo	0.35	0.21	0.24	0.19
basic-attention-token	0.41	0.18	0.24	0.17
digibyte	0.48	0.17	0.20	0.15
0x	0.46	0.20	0.20	0.14
true-usd	0.36	0.13	0.21	0.30
qtum	0.49	0.20	0.16	0.15
republic-protocol	0.30	0.24	0.26	0.19
swissborg	0.33	0.25	0.24	0.18
icon	0.49	0.20	0.16	0.14
loopring	0.39	0.22	0.21	0.17
lisk	0.48	0.19	0.18	0.15
kyber-network	0.28	0.26	0.26	0.21
quant-network	0.19	0.27	0.28	0.25
bitcoin-gold	0.60	0.13	0.16	0.10
maidsafecoin	0.46	0.22	0.20	0.12
vitae	0.11	0.44	0.24	0.21
siacoin	0.50	0.17	0.19	0.14
nano	0.45	0.19	0.21	0.15
enjincoin	0.29	0.26	0.28	0.16

In [24]:
```
# Clustering

# 3) LFM: Linear Factor Model followed by a k-means clustering algo
rithm on its factors
    # 3.a) Application on a single window

# Initialization: Firstly, test if it makes sense to make a LFM

# It checks whether variables are correlated or not. If significant
, ok. If not, a factor
# analysis should not be done.
chi_square_value,p_value = calculate_bartlett_sphericity(frames_nor
m)
chi_square_value, p_value
# It seems to be statistically significant so we can continue.

# It computes a score of the suitability of the data to a Factor An
alysis
kmo_all,kmo_model=calculate_kmo(frames_norm)
kmo_model
# 0.715 seems ok

# Creating factor analysis object and perform factor analysis
fa = FactorAnalyzer()
fa_test = fa.analyze(frames_norm, 4)

# Checking eigen values: it will show us the number of significant
factors

# Computing Eigenvalues
ev, v = fa.get_eigenvalues()
ev

# Creating a scree plot to display the different eigen values
plt.scatter(range(1,frames_norm.shape[1]+1),ev)
plt.plot(range(1,frames_norm.shape[1]+1),ev)
plt.title('Scree Plot')
plt.xlabel('Factors')
plt.ylabel('Eigenvalue')
plt.grid()
plt.savefig('Clusters/lfm_scree.jpeg',transparent=True)
plt.show()

n_factors = 3
fa = FactorAnalyzer()
fa_test = fa.analyze(frames_norm, n_factors)
print("The number of variables should be reduced to three since it'
s the number of eigen values above 1 (which signifies that a factor
is able to explain more than a single variable).")
```




```
The number of variables should be reduced to three since it's the
number of eigen values above 1 (which signifies that a factor is a
ble to explain more than a single variable).
```

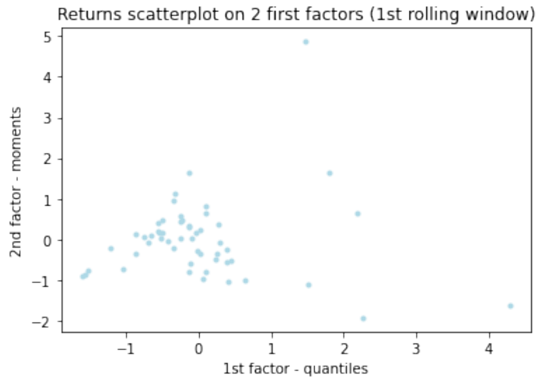
In [25]:
```
loadings_lfm = pd.DataFrame(fa.loadings)
ax = sns.heatmap(loadings_lfm, annot=True, cmap='Spectral')
plt.title('Heat map showing how three first factors explain the var
iables')
plt.savefig('Clusters/lfm_heatmap.jpeg',transparent=True)
plt.show()
```

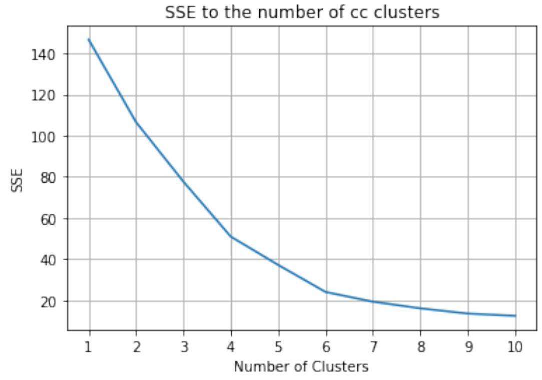


In [26]:
```
del sse,k,kmeans,n_kmeans
```

In [27]:
```
# Linear Factor model estimation

# Application on the first rolling window

frames_norm_np = np.array(frames_norm)

fanalysis = FactorAnalysis(n_components=n_factors)
df_3d = fanalysis.fit_transform(frames_norm_np); df_3d = pd.DataFra
me(df_3d)
```

```
df_3d.columns = ["Factor1","Factor2","Factor3"]
df_3d.index = cc_names

fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(df_3d['Factor1'],df_3d['Factor2'], color='lightblue', ma
rker='.')
ax.set(title='Returns scatterplot on 2 first factors (1st rolling w
indow)',ylabel='2nd factor - moments',xlabel='1st factor - quantile
s')
plt.savefig('Clusters/lfm_sc.jpeg',transparent=True)
plt.show()

mu = np.mean(frames_norm.transpose())
Y = returns_data.iloc[0];Y
X = np.zeros((len(frames_norm),3))
X[:,0] = df_3d['Factor1'];X[:,1] = df_3d['Factor2']; X[:,2] = mu
mlr = LinearRegression().fit(X,Y)

regressor = LinearRegression()
regressor.fit(X,Y)
regressor.coef_

# Then, the k-means algorithm may be applied on this time-step

# To check the number of clusters needed. The criteria is the sse (
sum of squared errors).
# Searching for the smallest one, a tradeoff needs to be found betw
een a small sse but a small number of kmeans.
sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, init = "random", n_init=10, max_i
ter = 300,
                    random_state = 123)
    kmeans.fit(df_3d)
    sse.append(kmeans.inertia_)

plt.plot(range(1, 11), sse)
plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title("SSE to the number of cc clusters",fontsize=12)
plt.grid(True)
plt.savefig('Clusters/lfm_nb_kmeans.jpeg',transparent=True)
plt.show()

print("Again, taking 4 clusters seems to be a good tradeoff between
precision and differentiation")
n_kmeans = 4
```

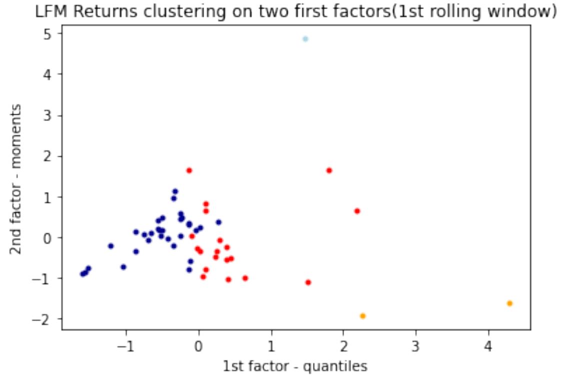





```
Again, taking 4 clusters seems to be a good tradeoff between preci
sion and differentiation
```

In [28]:
```python
# Now we know the number of clusters we want, let's try it for the first rolling window.

# initialize kmeans class object
kmeans = KMeans(n_clusters = n_kmeans,random_state=123) # defining the function kmeans
#predict the labels of clusters.
label = kmeans.fit_predict(df_3d)
print(label)

#Getting unique labels

u_labels = np.unique(label)

#filter rows of original data: it separates the data into the different clusters
filtered_label0 = df_3d[label == 0]
filtered_label1 = df_3d[label == 1]
filtered_label2 = df_3d[label == 2]
filtered_label3 = df_3d[label == 3]

#Plotting the results
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(filtered_label0['Factor1'],filtered_label0['Factor2'], color='red', marker='.')
ax.scatter(filtered_label1['Factor1'],filtered_label1['Factor2'], color='lightblue', marker='.')
ax.scatter(filtered_label2['Factor1'],filtered_label2['Factor2'], color='orange', marker='.')
ax.scatter(filtered_label3['Factor1'],filtered_label3['Factor2'], color='darkblue', marker='.')
ax.set(title='LFM Returns clustering on two first factors(1st rolling window)',ylabel='2nd factor - moments',xlabel='1st factor - quantiles')
plt.savefig('Clusters/lfm_clusters.jpeg',transparent = True)
plt.show()
```

```
[3 3 3 0 3 0 3 0 3 3 3 3 3 0 3 3 0 0 0 3 0 2 3 3 3 3 3 0 3 3 3 3 0
 3 0 3 3
 0 3 3 0 0 0 2 3 0 0 3 3 1 0 0 3]
```

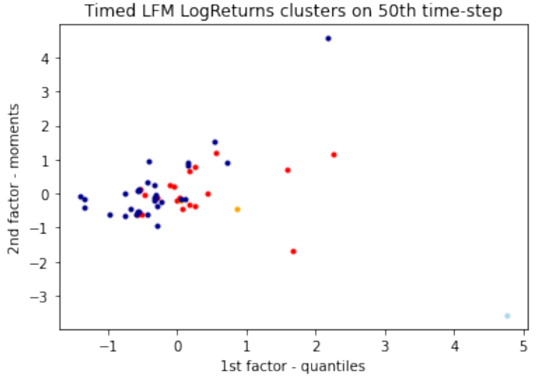


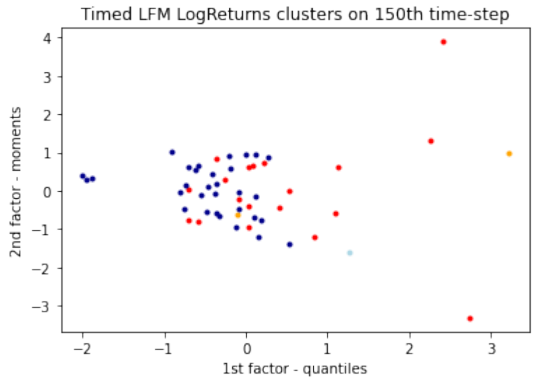
In [29]:
```python
# Clustering

# 3) LFM: Linear Factor Model followed by a k-means clustering algorithm on its factors
    # 3.b) Application on all rolling windows

# Timed linear factor model
nt = len(window_mean)
nb_cc = len(frames_norm)
labels = []
timed_LFM = np.zeros((nt,nb_cc,n_factors)) # 3D matrix for the projection of cc returns on two factors
timed_label = np.zeros((nt,nb_cc)) # output matrix with the clusters labels for each cc


for i in range(nt):
    win_frames = []
    win_frames = [window_mean.iloc[i],window_var.iloc[i], window_skew.iloc[i],
                  window_kurt.iloc[i], window_005_quantile.iloc[i], window_010_quantile.iloc[i],
                  window_015_quantile.iloc[i], window_085_quantile.iloc[i], window_090_quantile.iloc[i],
                  window_095_quantile.iloc[i]]
    win_frames = pd.concat(win_frames, axis=1)
    win_frames_norm = pd.DataFrame(scaler.transform(win_frames),
                                   index=win_frames.index)
    timed_LFM[i] = fanalysis.fit_transform(win_frames_norm)
    timed_label[i] = kmeans.fit_predict(timed_LFM[i])

timed_label = pd.DataFrame(timed_label)
timed_label.columns = cc_names
```

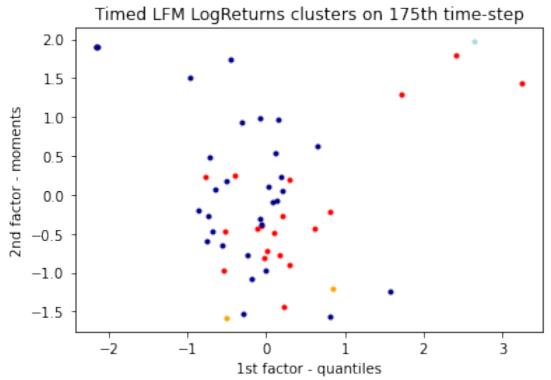
In [30]:

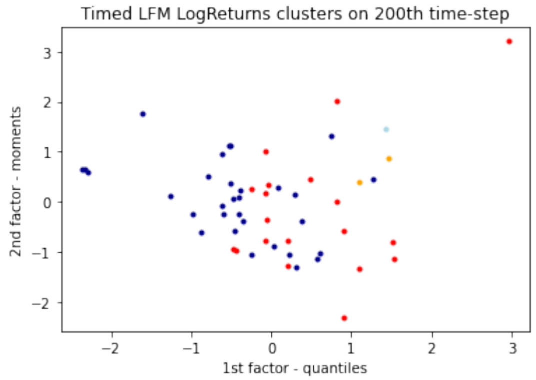
```
# Plotting the results for some rolling windows
print("Let's see how cc log returns are clustered for some rolling
windows")
rw_choice = [50,150,175,200,300]
rw_choice

for r in rw_choice:
    selected_ts = r
    filtered_data = timed_LFM[selected_ts]
    filtered_label0 = filtered_data[label == 0]
    filtered_label1 = filtered_data[label == 1]
    filtered_label2 = filtered_data[label == 2]
    filtered_label3 = filtered_data[label == 3]

        #Plotting the results
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.scatter(filtered_label0[:,0],filtered_label0[:,1], color='re
d', marker='.')
    ax.scatter(filtered_label1[:,0],filtered_label1[:,1], color='li
ghtblue', marker='.')
    ax.scatter(filtered_label2[:,0],filtered_label2[:,1], color='or
ange', marker='.')
    ax.scatter(filtered_label3[:,0],filtered_label3[:,1], color='da
rkblue', marker='.')
    ax.set(title='Timed LFM LogReturns clusters on '+str(selected_t
s) + 'th time-step',ylabel='2nd factor - moments',xlabel='1st facto
r - quantiles')
    plt.savefig('Clusters/timedlfm_clusters'+str(selected_ts)+'rw.j
peg',transparent=True)
    plt.show()
```

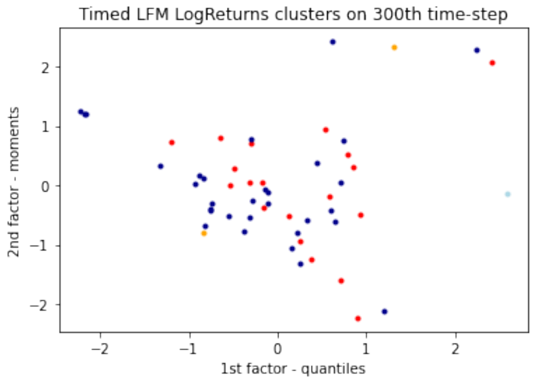
```
Let's see how cc log returns are clustered for some rolling window
s
```











In [31]:

```
# Distribution of cc log returns in clusters

index = pd.DataFrame([frames.index])
lfm_distribution = []
for i in index.transpose()[0]:
    y = timed_label[i].value_counts(normalize=True)
    lfm_distribution.append(y)
lfm_distribution = pd.DataFrame(lfm_distribution);lfm_distribution
= lfm_distribution.round(2)
lfm_distribution.columns = ["Cluster1","Cluster2","Cluster3","Clust
er4"]

print("Here is the distribution of cc log returns in each cluster:"
)
print(lfm_distribution)

lfm_distribution1 = pd.DataFrame(lfm_distribution[:17])
lfm_distribution2 = pd.DataFrame(lfm_distribution[17:34])
lfm_distribution3 = pd.DataFrame(lfm_distribution[34:])


lfm_distribution1.to_csv('Clusters/LFM_distribution1.csv', index=Tr
ue, header=True)
lfm_distribution2.to_csv('Clusters/LFM_distribution2.csv', index=Tr
ue, header=True)
lfm_distribution3.to_csv('Clusters/LFM_distribution3.csv', index=Tr
ue, header=True)
```

Here is the distribution of cc log returns in each cluster:

	Cluster1	Cluster2	Cluster3	Cluster4
bitcoin	0.45	0.23	0.15	0.17
ethereum	0.44	0.25	0.14	0.17
ripple	0.42	0.24	0.15	0.19
litecoin	0.46	0.22	0.17	0.15
bitcoin-cash	0.49	0.21	0.14	0.16
chainlink	0.35	0.24	0.20	0.21
binancecoin	0.47	0.21	0.14	0.18
cardano	0.44	0.30	0.12	0.13
stellar	0.43	0.23	0.15	0.19
usd-coin	0.40	0.19	0.15	0.27
bitcoin-cash-sv	0.38	0.21	0.23	0.17
eos	0.41	0.26	0.15	0.18
nem	0.41	0.24	0.16	0.19
tron	0.40	0.26	0.17	0.17
okb	0.35	0.26	0.20	0.19
tezos	0.44	0.22	0.18	0.16
neo	0.45	0.26	0.17	0.13
celsius-degree-token	0.25	0.27	0.28	0.20
theta-token	0.36	0.28	0.16	0.20
dash	0.47	0.23	0.12	0.18
vechain	0.34	0.22	0.24	0.21
havven	0.23	0.21	0.32	0.24
huobi-token	0.34	0.26	0.20	0.21
iota	0.46	0.21	0.14	0.19
zcash	0.48	0.23	0.14	0.15
waves	0.40	0.24	0.17	0.19
ethereum-classic	0.45	0.26	0.15	0.14
zilliqa	0.39	0.25	0.17	0.19
dogecoin	0.39	0.24	0.16	0.21
maker	0.40	0.22	0.18	0.21
decred	0.41	0.22	0.16	0.21
omisego	0.45	0.24	0.16	0.15
ontology	0.39	0.25	0.17	0.19
paxos-standard	0.40	0.18	0.15	0.27
nexo	0.30	0.26	0.20	0.24
basic-attention-token	0.33	0.20	0.26	0.21
digibyte	0.39	0.22	0.20	0.19
0x	0.30	0.29	0.21	0.21
true-usd	0.40	0.18	0.15	0.27
qtum	0.39	0.29	0.17	0.14
republic-protocol	0.29	0.26	0.21	0.24
swissborg	0.29	0.31	0.19	0.20
icon	0.39	0.28	0.16	0.17
loopring	0.35	0.31	0.17	0.17
lisk	0.36	0.25	0.19	0.19
kyber-network	0.26	0.31	0.19	0.23
quant-network	0.18	0.33	0.25	0.24
bitcoin-gold	0.52	0.22	0.11	0.15
maidsafecoin	0.35	0.27	0.19	0.19
vitae	0.12	0.29	0.27	0.31
siacoin	0.44	0.20	0.19	0.17
nano	0.41	0.26	0.18	0.15
enjincoin	0.26	0.33	0.20	0.20

In [ ]:

```

```