In [1]:

```
#Project Milestone 5
#Collecting all the data sets
#Plots
```

Comparing Various Cryptocurrencies & see which is more stable than other

In [2]:

```
#Loading the libraries

from sklearn.metrics import confusion_matrix,accuracy_score,recall_score,precisi
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
from sklearn.svm import SVC
import seaborn as sns
import pandas as pd
import numpy as np
import sklearn
```

```
/Users/dragon/opt/anaconda3/lib/python3.8/site-packages/scipy/__init__.py:138: U
serWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of
SciPy (detected version 1.24.1)
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion} is requ
ired for this version of "
```

In [15]:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
plt.style.use("fivethirtyeight")
%matplotlib inline
import matplotlib
from matplotlib.colors import LinearSegmentedColormap
# For reading stock data from yahoo
from pandas_datareader.data import DataReader

# For time stamps
from datetime import datetime

import warnings
warnings.filterwarnings("ignore")
```

In [13]:

```
pip install pandas_datareader
```

```
Collecting pandas_datareader
  Downloading pandas_datareader-0.10.0-py3-none-any.whl (109 kB)
     |████████████████████████████████| 109 kB 1.0 MB/s eta 0:00:01
```

```
Requirement already satisfied: lxml in /Users/dragon/opt/anaconda3/lib/python3.
8/site-packages (from pandas_datareader) (4.6.3)
Requirement already satisfied: pandas>=0.23 in /Users/dragon/opt/anaconda3/lib/p
ython3.8/site-packages (from pandas_datareader) (1.5.3)
Requirement already satisfied: requests>=2.19.0 in /Users/dragon/opt/anaconda3/l
ib/python3.8/site-packages (from pandas_datareader) (2.25.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /Users/dragon/opt/anaco
nda3/lib/python3.8/site-packages (from pandas>=0.23->pandas_datareader) (2.8.1)
Requirement already satisfied: numpy>=1.20.3 in /Users/dragon/opt/anaconda3/lib/
python3.8/site-packages (from pandas>=0.23->pandas_datareader) (1.24.1)
Requirement already satisfied: pytz>=2020.1 in /Users/dragon/opt/anaconda3/lib/p
ython3.8/site-packages (from pandas>=0.23->pandas_datareader) (2021.1)
Requirement already satisfied: six>=1.5 in /Users/dragon/opt/anaconda3/lib/pytho
n3.8/site-packages (from python-dateutil>=2.8.1->pandas>=0.23->pandas_datareade
r) (1.15.0)
Requirement already satisfied: idna<3,>=2.5 in /Users/dragon/opt/anaconda3/lib/p
ython3.8/site-packages (from requests>=2.19.0->pandas_datareader) (2.10)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /Users/dragon/opt/anacon
da3/lib/python3.8/site-packages (from requests>=2.19.0->pandas_datareader) (1.2
6.4)
Requirement already satisfied: chardet<5,>=3.0.2 in /Users/dragon/opt/anaconda3/
lib/python3.8/site-packages (from requests>=2.19.0->pandas_datareader) (4.0.0)
Requirement already satisfied: certifi>=2017.4.17 in /Users/dragon/opt/anaconda
3/lib/python3.8/site-packages (from requests>=2.19.0->pandas_datareader) (2020.1
2.5)
Installing collected packages: pandas-datareader
Successfully installed pandas-datareader-0.10.0
Note: you may need to restart the kernel to use updated packages.
```

In [6]:

```
# Data Source 1 - Flat file data
btc = pd.read_csv('/Users/dragon/Documents/repo/DSC680/project2/Bitcoin.csv')
btc.head()
```

Out[6]:

	Date	Open	High	Low	Close	Volume	Market Cap
0	27-06-2021	32287.523211	34656.127356	32071.757148	34649.644588	3.551164e+10	6.494617e+11
1	26-06-2021	31594.663571	32637.587193	30184.501794	32186.277671	3.858539e+10	6.032760e+11
2	25-06-2021	34659.104499	35487.248003	31350.883858	31637.780055	4.023090e+10	5.929782e+11
3	24-06-2021	33682.800404	35228.852611	32385.214696	34662.435894	3.312337e+10	6.496440e+11
4	23-06-2021	32515.714303	34753.408503	31772.632355	33723.028978	4.631711e+10	6.320113e+11

In [7]:

```
maxValue=btc[btc['Close']==max(btc.Close)]
print("Highiest value of bitcoin")
maxValue
```

```
Highiest value of bitcoin
```

Out[7]:

	Date	Open	High	Low	Close	Volume	Market Cap
75	13-04-2021	59890.01779	63742.283337	59869.956293	63503.45793	6.998345e+10	1.186364e+12

In [8]:

```
btc.describe()
```

Out[8]:

	Open	High	Low	Close	Volume	Market Cap
count	2983.000000	2983.000000	2983.000000	2983.000000	2.983000e+03	2.983000e+03
mean	6613.843852	6805.260516	6402.037782	6625.160672	1.084006e+10	1.192478e+11
std	11199.197921	11553.544329	10781.352553	11209.848478	1.887826e+10	2.092413e+11
min	68.504997	74.561096	65.526001	68.431000	0.000000e+00	7.784112e+08
25%	429.792999	435.968491	421.852005	429.862000	3.013725e+07	6.294347e+09
50%	2191.560059	2303.899902	2071.989990	2202.419922	9.174120e+08	3.601766e+10
75%	8494.818022	8687.633944	8230.187652	8500.334961	1.575577e+10	1.488823e+11
max	63523.754869	64863.098908	62208.964366	63503.457930	3.509679e+11	1.186364e+12

In [9]:

```
btc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2983 entries, 0 to 2982
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Date        2983 non-null   object 
 1   Open        2983 non-null   float64
 2   High        2983 non-null   float64
 3   Low         2983 non-null   float64
 4   Close       2983 non-null   float64
 5   Volume      2983 non-null   float64
 6   Market Cap  2983 non-null   float64
dtypes: float64(6), object(1)
memory usage: 163.3+ KB
```

In [18]:

```
eth=pd.read_csv('/Users/dragon/Documents/repo/DSC680/project2/Ethereum.csv',pars
eth = eth.iloc[::-1]
eth.tail(5)
```

Out[18]:

Date	Open	High	Low	Close	Volume	Market Cap
2021-06-23	1878.625034	2043.530443	1827.571482	1989.736271	2.840866e+10	2.316253e+11
2021-06-24	1968.957423	2032.339389	1887.432046	1988.456276	2.027285e+10	2.315093e+11
2021-06-25	1989.215789	2017.759464	1794.400424	1813.217232	2.277433e+10	2.111311e+11

Date	Open	High	Low	Close	Volume	Market Cap
2021-06-26	1810.884253	1850.179779	1719.559464	1829.239217	2.063754e+10	2.130212e+11
2021-06-27	1830.996918	1979.958125	1811.245864	1978.894662	1.988547e+10	2.304736e+11

In [19]:

```
maxValue=eth[eth['Close']==max(eth.Close)]
print("Highiest value of Ethereum")
maxValue
```

```
Highiest value of Ethereum
```

Out[19]:

Date	Open	High	Low	Close	Volume	Market Cap
2021-11-05	3948.271909	4178.208815	3783.889474	4168.701049	5.267974e+10	4.828819e+11

In [25]:

```
bit=pd.read_csv('/Users/dragon/Documents/repo/DSC680/project2/Dead Coin/bitconne
bit=bit.iloc[:,1:7]
bit = bit.iloc[::-1]
bit=bit.iloc[:1580]
bit18=bit[:609]
bit.tail(5)
```

Out[25]:

Date	Open	High	Low	Close	Volume	Volume(BCC)
2021-05-16	2.40	2.48	2.14	2.25	0.00	0.00
2021-05-17	2.25	2.30	2.06	2.19	4.26	1.92
2021-05-18	2.19	2.50	2.18	2.32	18.81	8.01
2021-05-19	2.31	2.34	1.68	2.01	0.00	0.00
2021-05-20	2.01	2.28	1.91	2.19	0.00	0.00

Data Visualization

Closing price of various Cryptocurrency in 2018 vs 2020

In [26]:

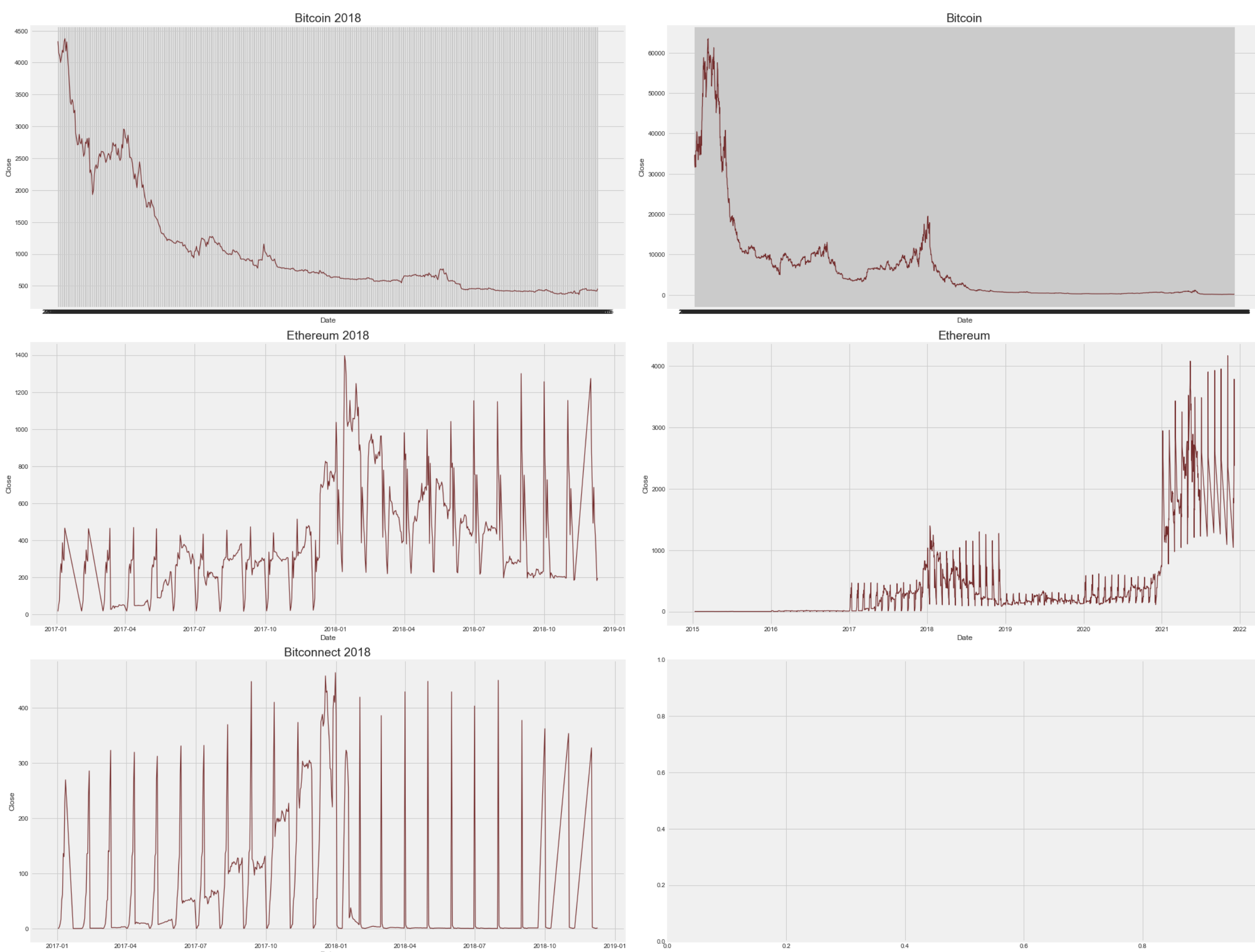
```python
def to2018(df):
    df18=equalize(df,bit)
    return df18.iloc[:len(bit18)]

def equalize(df,dfs):
    low=len(dfs)
    high=len(df)
    dff=high-low

    return df.iloc[dff:]
```

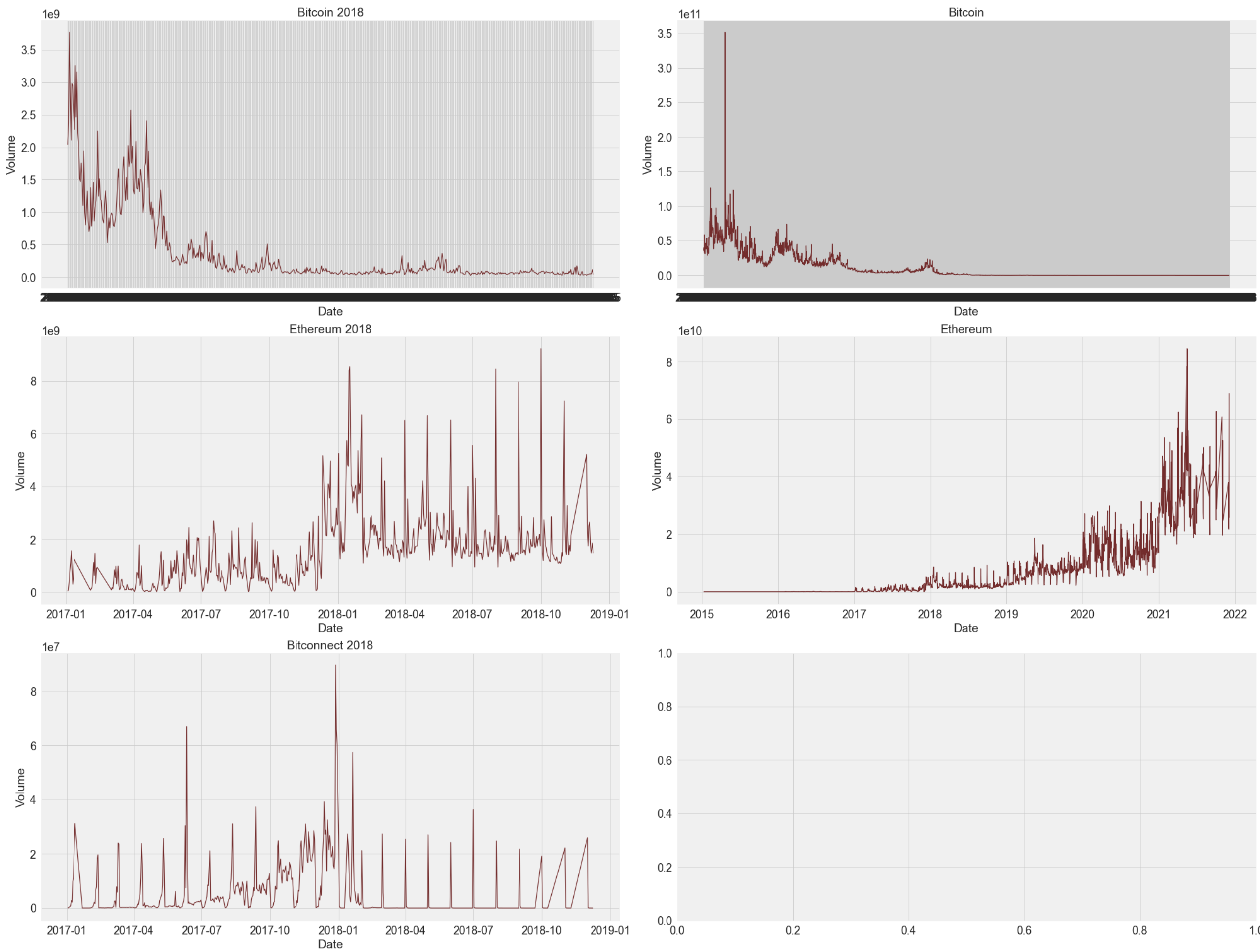
In [28]:

```python
btc18=to2018(btc)
eth18=to2018(eth)
crypto=["Bitcoin 2018","Bitcoin","Ethereum 2018","Ethereum","Bitconnect 2018"]
cryptoDf=[btc18,btc,eth18,eth,bit18]
num_plots = 6
total_cols = 2
total_rows = 3
fig, axs = plt.subplots(nrows=total_rows, ncols=total_cols,
                        figsize=(14*total_cols, 7*total_rows), constrained_layou
for i, var in enumerate(crypto):
    row = i//total_cols
    pos = i % total_cols
    sns.set_context('paper', font_scale = 2)
    plot =  sns.lineplot(data=cryptoDf[i], x="Date", y="Close",color='#732C2C',p
    axs[row][pos].set_title(crypto[i])
```



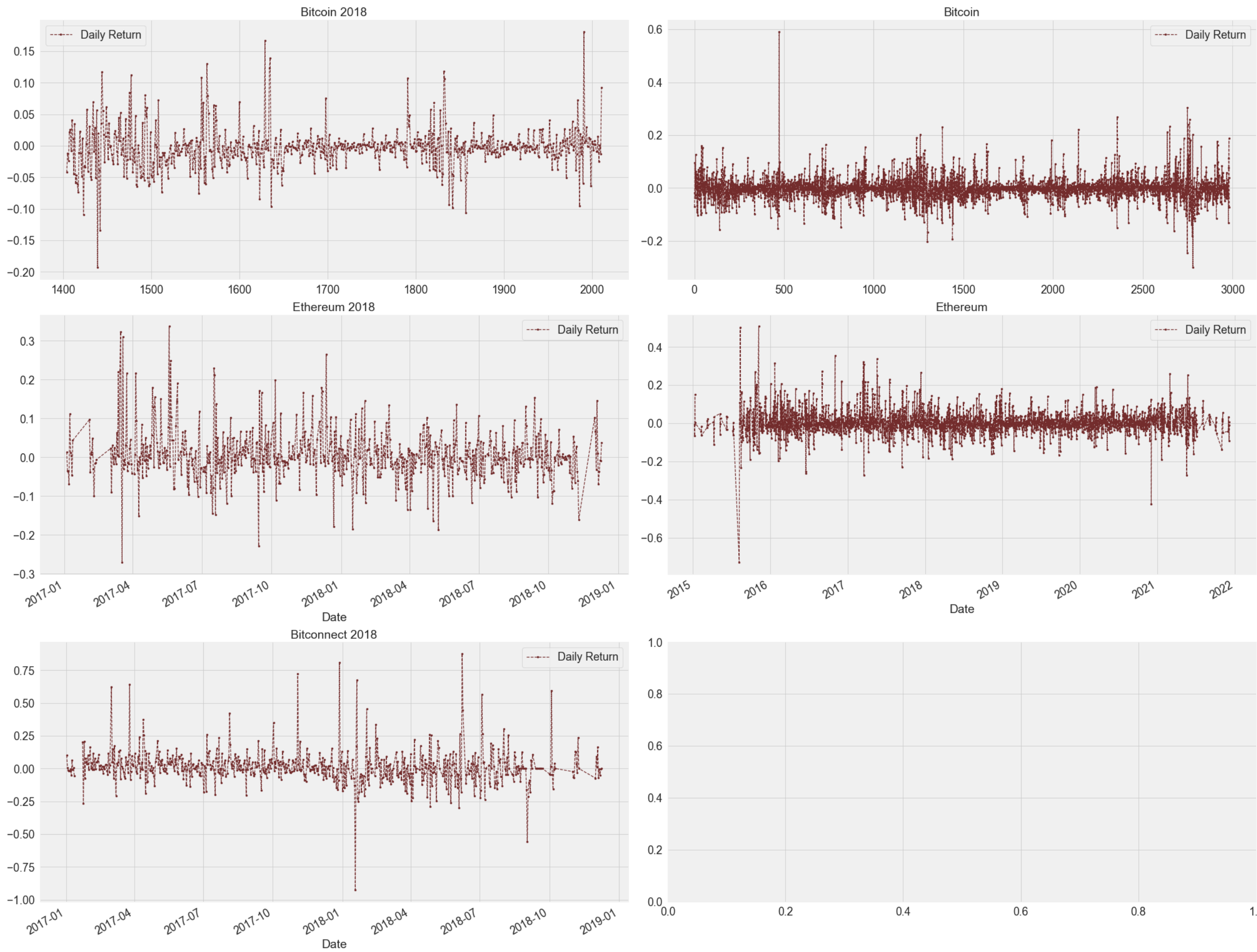
In [29]:

```
fig, axs = plt.subplots(nrows=total_rows, ncols=total_cols,
                        figsize=(14*total_cols, 7*total_rows), constrained_layou
for i, var in enumerate(crypto):
    row = i//total_cols
    pos = i % total_cols
    sns.set_context('paper', font_scale = 2)
    plot =  sns.lineplot(data=cryptoDf[i], x="Date", y="Volume",color='#732C2C',
    axs[row][pos].set_title(crypto[i])
```



In [30]:

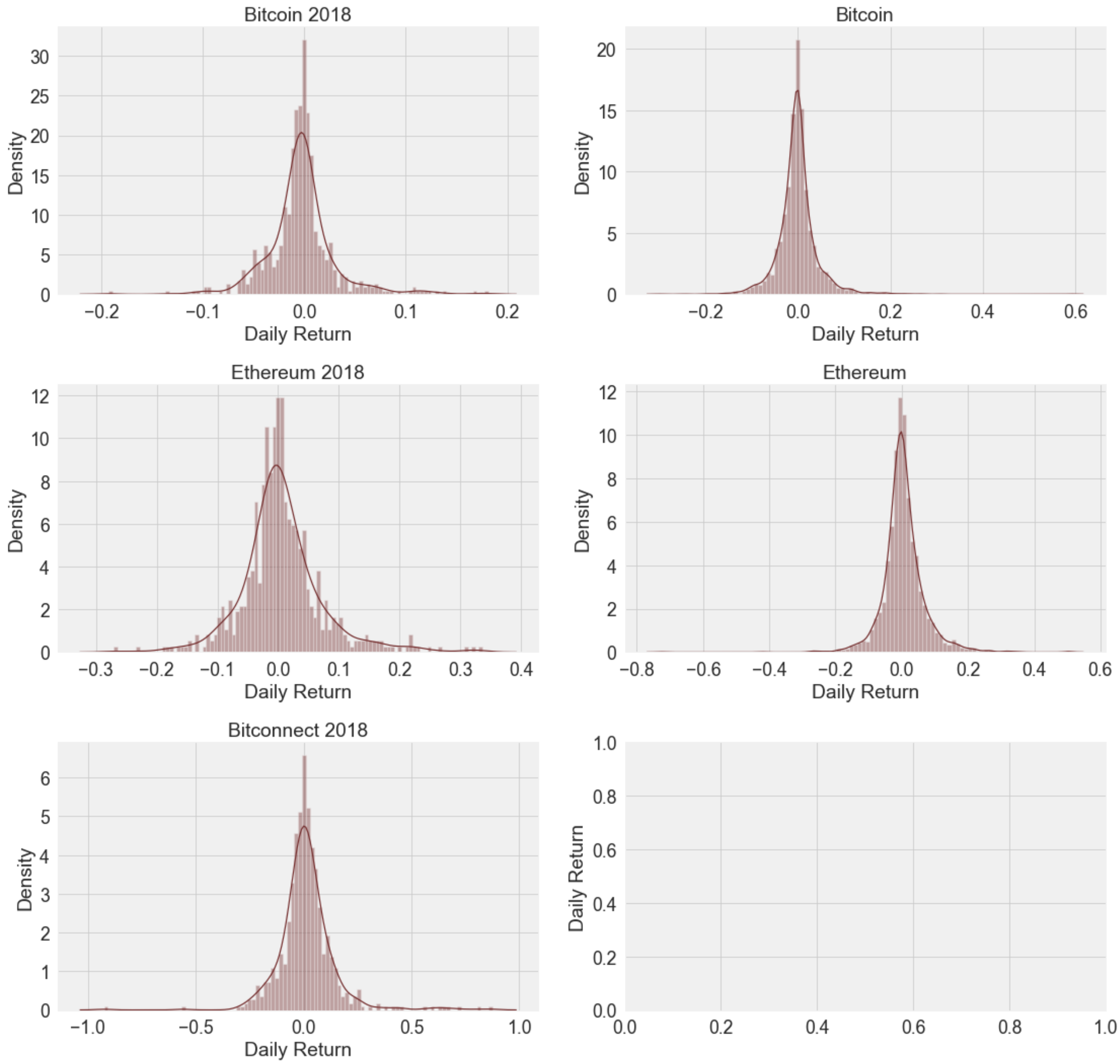
```
for df in cryptoDf:
    df['Daily Return'] = df['Close'].pct_change()
fig, axs = plt.subplots(nrows=total_rows, ncols=total_cols,
                        figsize=(14*total_cols, 7*total_rows), constrained_layou
for i, var in enumerate(crypto):
    row = i//total_cols
    pos = i % total_cols
    cryptoDf[i]['Daily Return'].plot(ax=axs[row][pos], legend=True,color='#732C2
    axs[row][pos].set_title(crypto[i])
```



In [31]:

```
fig, axs = plt.subplots(nrows=total_rows, ncols=total_cols,
                        figsize=(8*total_cols, 5*total_rows))
for i, var in enumerate(crypto):
    row = i//total_cols
    pos = i % total_cols
    plot =sns.distplot(cryptoDf[i]['Daily Return'], bins=100, color='#732C2C',ax
    axs[row][pos].set_title(crypto[i])
    plt.ylabel('Daily Return')

plt.tight_layout()
```



Correlation

For Correlation we will need a dataset of Closing price of all the cryptocurrency in our list. So, we will make two dataset one for 2018 and one for 2021.

In [32]:

```
closeDf18=pd.DataFrame()
closeDf18['btc']=btc18['Close']
closeDf18['eth']=eth18['Close']
closeDf18['bit']=bit18['Close']
returns18 = closeDf18.pct_change()
returns18.head()
```

Out[32]:

	btc	eth	bit
1403	NaN	NaN	NaN
1404	-0.042255	NaN	NaN
1405	-0.012285	NaN	NaN

	btc	eth	bit
1406	-0.024090	NaN	NaN
1407	0.021471	NaN	NaN

In [35]:

```
btc=equalize(btc,eth)
closeDf=pd.DataFrame()
closeDf['btc']=btc['Close']
closeDf['eth']=eth['Close']
returns = closeDf.pct_change()
returns.head()
```

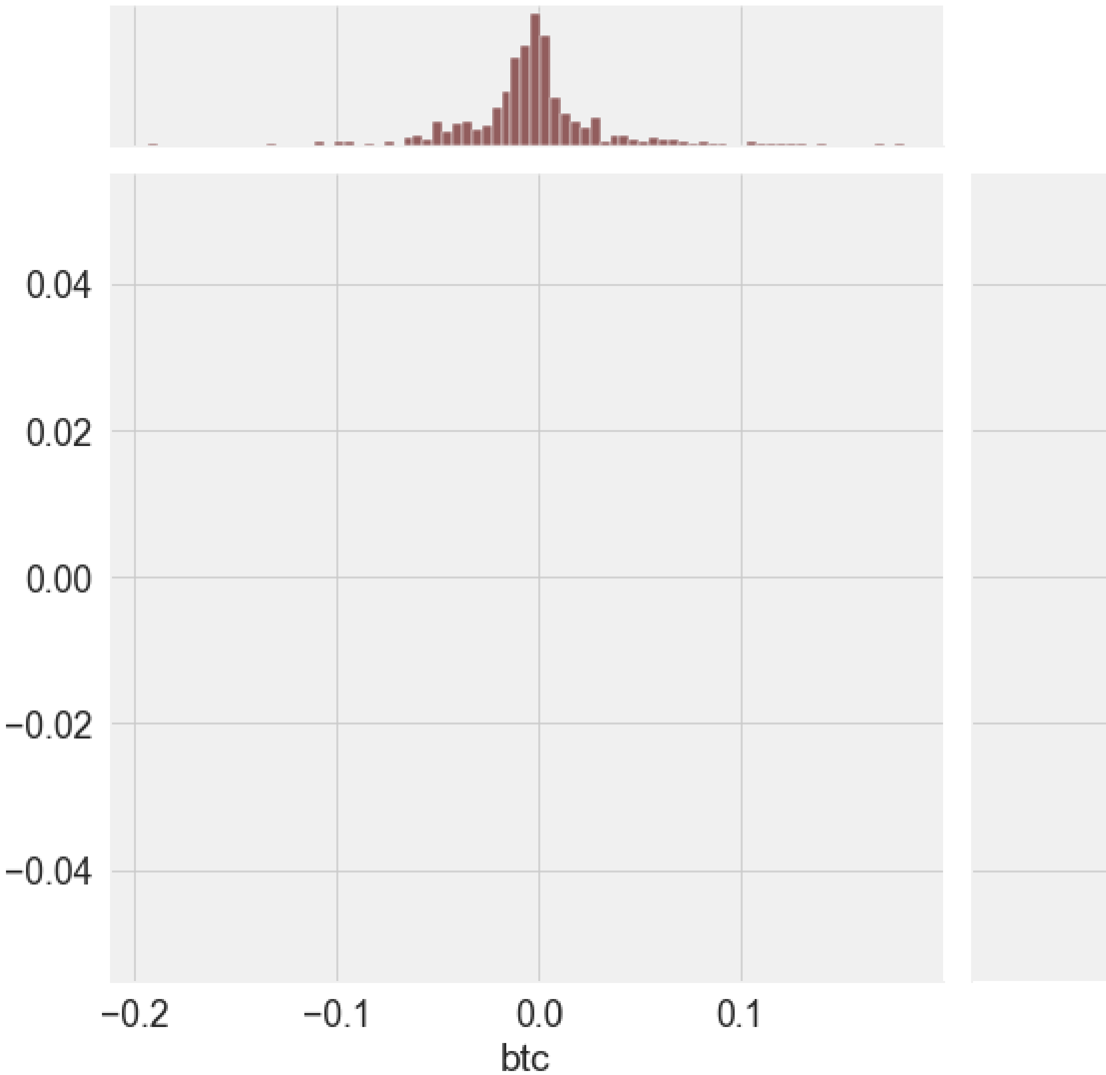
Out[35]:

	btc	eth
831	NaN	NaN
832	-0.009502	NaN
833	-0.001805	NaN
834	0.005837	NaN
835	-0.021689	NaN

Now we can compare the daily percentage return of two cryptocurrency to check how correlated.

In [36]:

```
sns.jointplot(data=returns18, x='btc', y="bit", kind='scatter',color='#732C2C',h
plt.show()
```

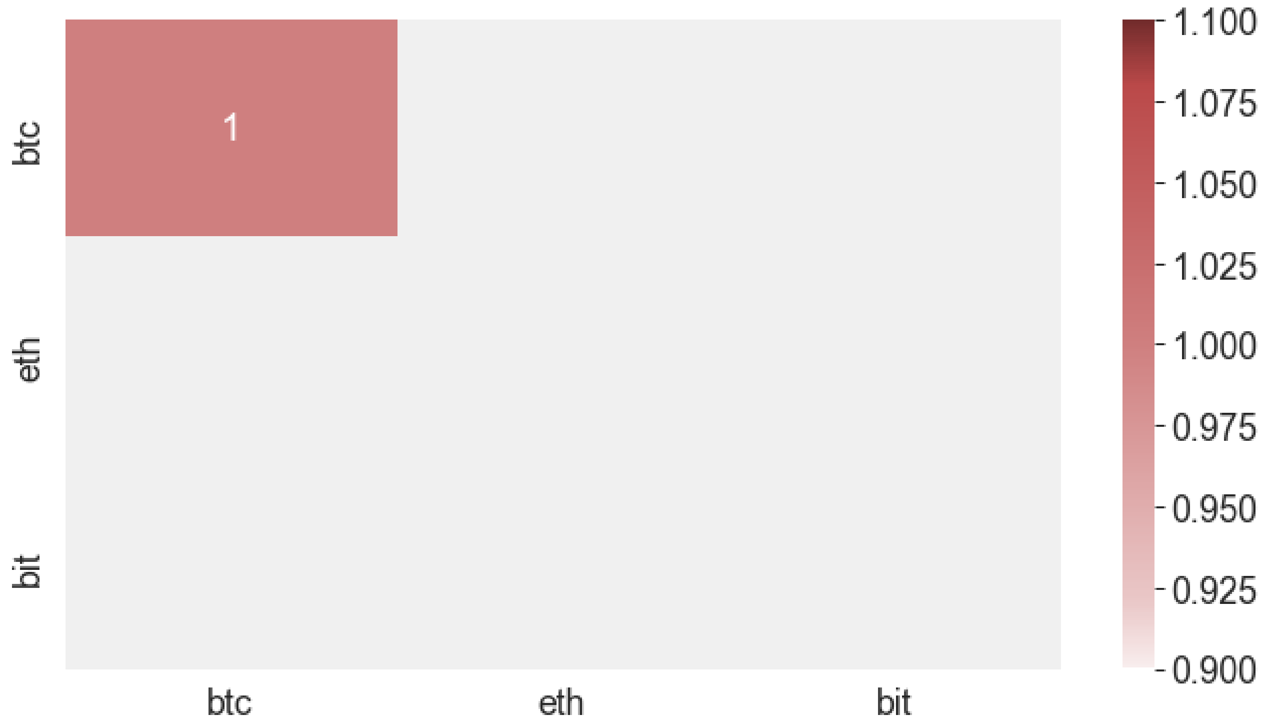


In [38]:

```
def NonLinCdict(steps, hexcol_array):
    cdict = {'red': (), 'green': (), 'blue': ()}
    for s, hexcol in zip(steps, hexcol_array):
        rgb =matplotlib.colors.hex2color(hexcol)
        cdict['red'] = cdict['red'] + ((s, rgb[0], rgb[0]),)
        cdict['green'] = cdict['green'] + ((s, rgb[1], rgb[1]),)
        cdict['blue'] = cdict['blue'] + ((s, rgb[2], rgb[2]),)
    return cdict

hc = ['#F8EDED', '#EAC8C8', '#CF7F7F', '#BA4949', '#732C2C']
th = [0, 0.1, 0.5, 0.9, 1]

cdict = NonLinCdict(th, hc)
cm = LinearSegmentedColormap('test', cdict)
plt.figure(figsize=(10,6))
sns.heatmap(returns18.corr(), annot=True, cmap=cm)
plt.show()
```



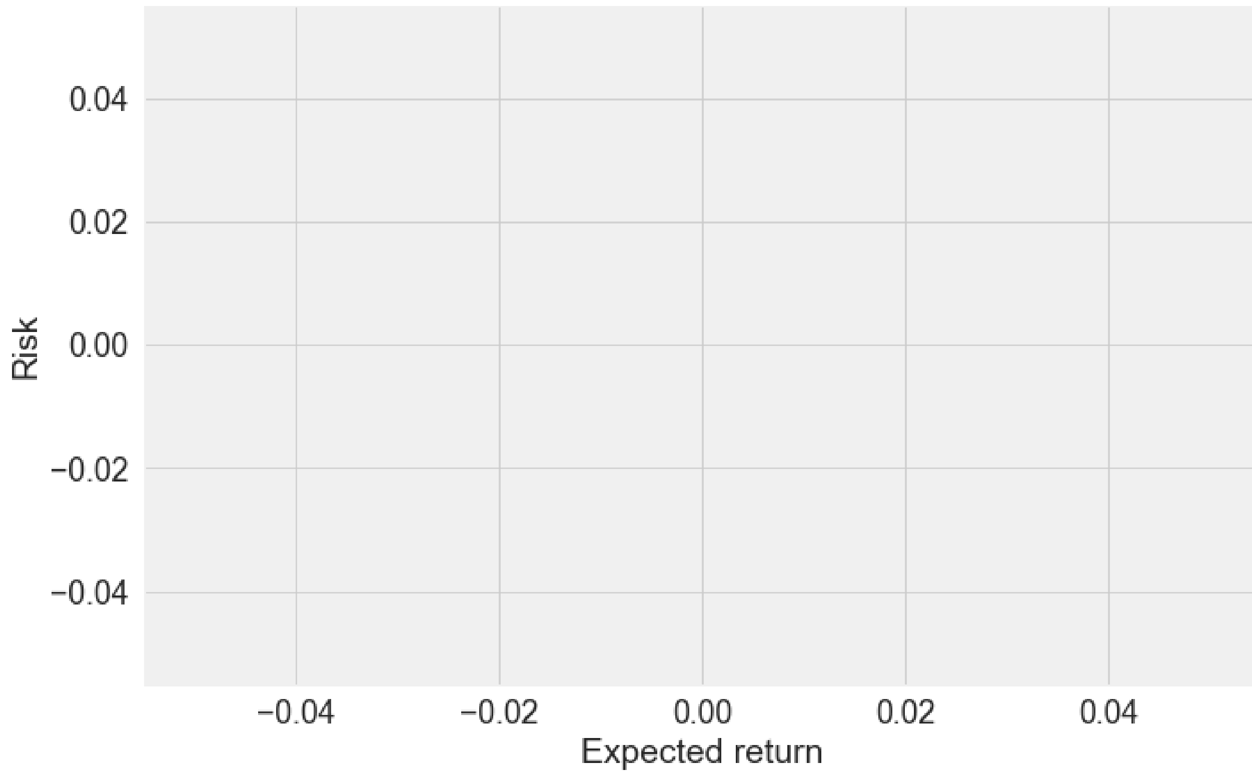
In [39]:

```
rets = returns18.dropna()

area = np.pi * 20

plt.figure(figsize=(10, 7))
plt.scatter(rets.mean(), rets.std(), s=area)
plt.xlabel('Expected return')
plt.ylabel('Risk')

for label, x, y in zip(rets.columns, rets.mean(), rets.std()):
    plt.annotate(label, xy=(x, y), xytext=(50, 50), textcoords='offset points',
                 arrowprops=dict(arrowstyle='-', color='#732C2C', connectionstyl
```



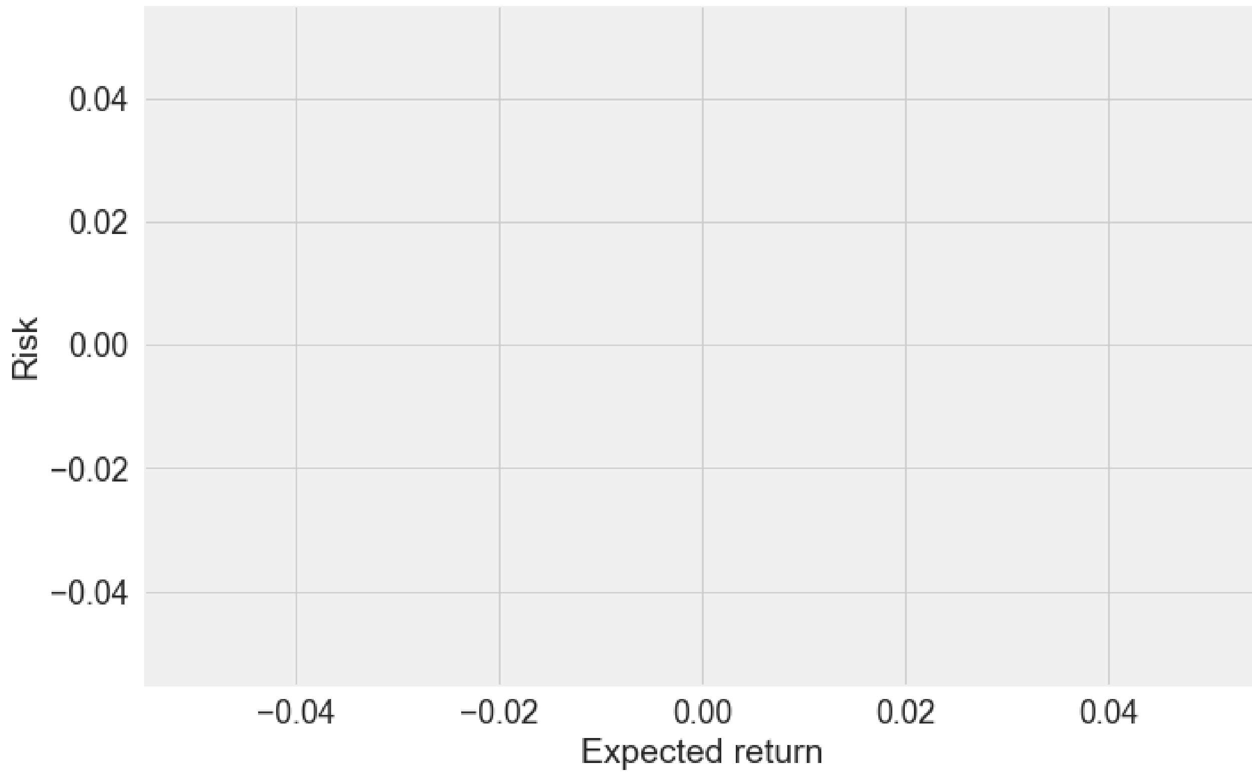
In [40]:

```
rets = returns.dropna()

area = np.pi * 20

plt.figure(figsize=(10, 7))
plt.scatter(rets.mean(), rets.std(), s=area)
plt.xlabel('Expected return')
plt.ylabel('Risk')

for label, x, y in zip(rets.columns, rets.mean(), rets.std()):
    plt.annotate(label, xy=(x, y), xytext=(50, 50), textcoords='offset points',
                 arrowprops=dict(arrowstyle='-', color='#732C2C', connectionstyl
```



Splitting Data

In [51]:

```
df = btc
df
```

Out[51]:

	Date	Open	High	Low	Close	Volume	Market Cap
831	19-03-2019	4032.692007	4082.216011	4023.812545	4071.190200	9.344920e+09	7.164770e+10
832	18-03-2019	4029.968458	4071.556731	4009.117226	4032.507385	9.646954e+09	7.095817e+10
833	17-03-2019	4047.719571	4054.122014	4006.411148	4025.228980	8.221625e+09	7.082194e+10
834	16-03-2019	3963.900120	4077.036282	3961.657527	4048.725904	9.856167e+09	7.122797e+10
835	15-03-2019	3926.663231	3968.542866	3914.015357	3960.911187	9.394211e+09	6.967500e+10
...	...	...	...	...	...	...	...
2978	02-05-2013	116.379997	125.599998	92.281898	105.209999	0.000000e+00	1.168517e+09
2979	01-05-	139.000000	139.889999	107.720001	116.989998	0.000000e+00	1.298955e+09

	Date	Open	High	Low	Close	Volume	Market Cap
	2013						
2980	30-04-2013	144.000000	146.929993	134.050003	139.000000	0.000000e+00	1.542813e+09
2981	29-04-2013	134.444000	147.488007	134.000000	144.539993	0.000000e+00	1.603769e+09
2982	28-04-2013	135.300003	135.979996	132.100006	134.210007	0.000000e+00	1.488567e+09

2152 rows × 8 columns

In [61]:

```
columns = ['Open','High','Low','Close','Volume','Market Cap','Daily Return']
df = df.loc[:, columns]
df.head(10)
```

Out[61]:

	Open	High	Low	Close	Volume	Market Cap	D Ret
831	4032.692007	4082.216011	4023.812545	4071.190200	9.344920e+09	7.164770e+10	-0.003
832	4029.968458	4071.556731	4009.117226	4032.507385	9.646954e+09	7.095817e+10	-0.009
833	4047.719571	4054.122014	4006.411148	4025.228980	8.221625e+09	7.082194e+10	-0.001
834	3963.900120	4077.036282	3961.657527	4048.725904	9.856167e+09	7.122797e+10	0.005
835	3926.663231	3968.542866	3914.015357	3960.911187	9.394211e+09	6.967500e+10	-0.021
836	3905.576999	3946.504287	3901.296877	3924.369118	1.048079e+10	6.902470e+10	-0.009
837	3913.047443	3926.597729	3891.904192	3906.717169	9.469185e+09	6.870670e+10	-0.004
838	3903.758294	3926.889083	3863.559100	3909.156209	9.809887e+09	6.874300e+10	0.000
839	3953.740174	3966.384737	3889.239133	3905.227320	1.012590e+10	6.866693e+10	-0.001
840	3966.174233	3966.174233	3924.381059	3951.599828	9.713268e+09	6.947530e+10	0.011

In [63]:

```
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.3)
y_train.value_counts()
```

Out[63]:

```
Daily Return
-0.300543    1
 0.006073    1
 0.006251    1
 0.006221    1
 0.006220    1
            ..
-0.010655    1
-0.010838    1
-0.010879    1
-0.010885    1
 0.268148    1
Length: 1506, dtype: int64
```

Scaling & predicting

In [75]:

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
x_train=scaler.fit_transform(x_train)
x_test=scaler.transform(x_test)
x_train
from sklearn import preprocessing
from sklearn import utils

#convert y values to categorical values
lab = preprocessing.LabelEncoder()
y_train = lab.fit_transform(y_train)
y_test = lab.fit_transform(y_test)
```

In [65]:

```
models={
    'LR':LogisticRegression(),
    'KNN':KNeighborsClassifier(),
    'DT':DecisionTreeClassifier(),
    'SVC':SVC(),
    'NB':GaussianNB(),
    'XGC':XGBClassifier(),
    'RF':RandomForestClassifier()

}
```

In [72]:

```
models={

    'KNN':KNeighborsClassifier(),
    'DT':DecisionTreeClassifier(),


}
```

In [ ]:

```
from sklearn import preprocessing
from sklearn import utils

#convert y values to categorical values
lab = preprocessing.LabelEncoder()
y_transformed = lab.fit_transform(y)
```

In [80]:

```
for name,model in  models.items():
    print(f'using {name}: ')
    model.fit(x_train,y_train)
    y_pred=model.predict(x_test)
    print(f'Training Accuracy :{accuracy_score(y_train,model.predict(x_train))}'
    print(f'Testing Accuracy :{accuracy_score(y_test,y_pred)}')
    print(f'Confusion matrix:\n {confusion_matrix(y_test,y_pred)}')
    print('-'*33)
```

```
using KNN:
Training Accuracy :0.2151394422310757
```

```
Testing Accuracy :0.0
Confusion matrix:
 [[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
---------------------------------
using DT:
Training Accuracy :1.0
Testing Accuracy :0.0
Confusion matrix:
 [[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
---------------------------------
```

In [81]:

```
models={
    'RF':RandomForestClassifier()

}
```

In [87]:

```
for name,model in  models.items():
    print(f'using {name}: ')
    model.fit(x_train,y_train)
    y_pred=model.predict(x_test)
    print(f'Training Accuracy :{accuracy_score(y_train,model.predict(x_train))}'
    print(f'Testing Accuracy :{accuracy_score(y_test,y_pred)}')
    print(f'Confusion matrix:\n {confusion_matrix(y_test,y_pred)}')
    print('-'*33)
```

```
using RF:
Training Accuracy :1.0
Testing Accuracy :0.0
Confusion matrix:
 [[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
---------------------------------
```

In [90]:

```
models={

    'SVC':SVC()

}
```

In [91]:

```
for name,model in  models.items():
    print(f'using {name}: ')
    model.fit(x_train,y_train)
```

```
    y_pred=model.predict(x_test)
    print(f'Training Accuracy :{accuracy_score(y_train,model.predict(x_train))}'
    print(f'Testing Accuracy :{accuracy_score(y_test,y_pred)}')
    print(f'Confusion matrix:\n {confusion_matrix(y_test,y_pred)}')
    print('-'*33)
```

```
using SVC:
Training Accuracy :1.0
Testing Accuracy :0.0
Confusion matrix:
 [[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
---------------------------------
```

Exploratory data analysis also suggested Market Cap was important features in deciding. Decision trees doesn't require to much data preparation or handling of outliers like logistic regression. They are easy to understand. Decision tress can easily overfit , so we have to be careful using decision tree. I think some challenges around the Hypothesis Testing and choice of the test statistic. I chose the difference between the means of the two groups as my main test statistics, however I still think could I use some other comparisons too as test statistics such as standard deviation or chi squared based tests. I feel still as a novice and learning as I read and practice with different datasets.

In [ ]:

```

```