Assignment 2

Daniel Fylling

Part 1: Scatterplots and Bivariate Distributions

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as st
```

In [ ]:
```
# Creating data cleaning function to create arrays that can be compared on day-t
def clean(date, x):
    # add data to dataframe
    df = pd.DataFrame({'date': date,'argument': x})

    # convert date column to datetime
    df['date'] = pd.to_datetime(df['date'])

    # date range chosen based on input data. Start when BTC data starts, end whe
    date_range = pd.date_range(start='2019-04-15', end='2022-01-03', freq='D')

    # reindex dataframe with the date range
    array = np.asarray(df.set_index('date').reindex(date_range).reset_index())
    array = np.array(array[:,1], dtype=float)
    return array
```

In [ ]:
```
# sending each input file through data cleaning function.
# Values for skiprows and index for argument column found by inspecting files.

df = pd.read_excel('Basel_Temp_History.xlsx',skiprows=10, header=None)
basel_temp = clean(df[0], df[3])

df = pd.read_csv('Brent_Spot_Price.csv',skiprows=5,header=None)
brent = clean(df[0], df[1])

df = pd.read_csv('BTC_USD.csv',skiprows=1,header=None)
btc = clean(df[0], df[1])

df = pd.read_csv('facebook.csv',skiprows=1,header=None)
face = clean(df[0], df[4])

df = pd.read_csv('google.csv',skiprows=1,header=None)
google = clean(df[0], df[4])

df = pd.read_csv('microsoft.csv',skiprows=1,header=None)
micro = clean(df[0], df[4])

df = pd.read_csv('WTI_Spot_Price.csv',skiprows=5,header=None)
wti = clean(df[0], df[1])
```

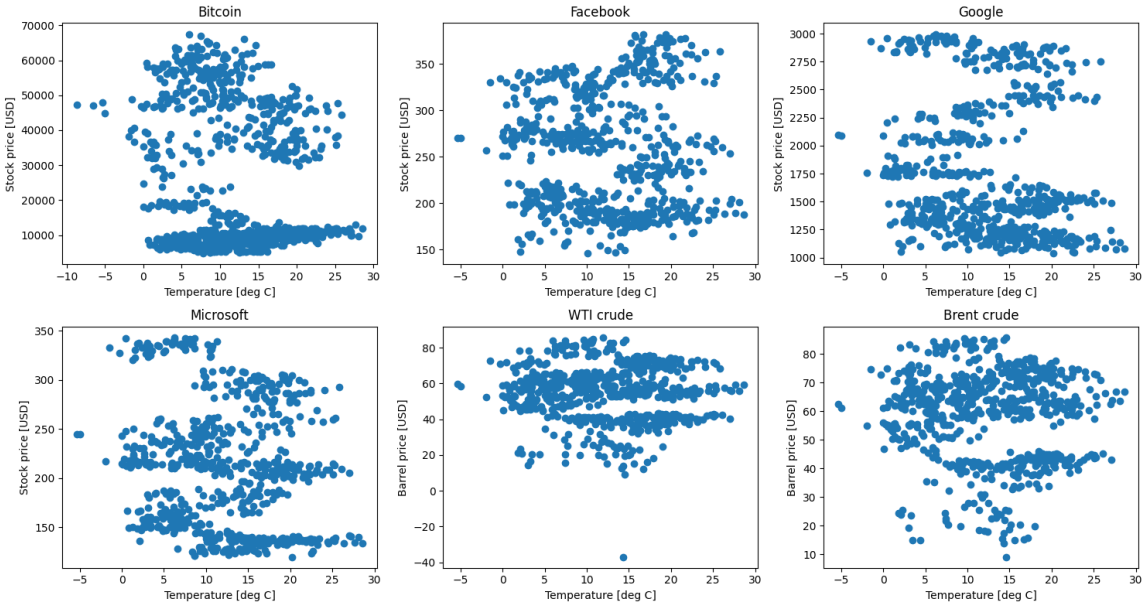
In [ ]:
```
len(wti)
```

Out[ ]: 995

a)

Scatterplot

In [ ]:
```
plt.figure(figsize=(15,8))

plt.subplot(2,3,1)
plt.scatter(basel_temp,btc)
plt.title('Bitcoin')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,2)
plt.scatter(basel_temp,face)
plt.title('Facebook')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,3)
plt.scatter(basel_temp,google)
plt.title('Google')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,4)
plt.scatter(basel_temp,micro)
plt.title('Microsoft')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,5)
plt.scatter(basel_temp,wti)
plt.title('WTI crude')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.subplot(2,3,6)
plt.scatter(basel_temp,brent)
plt.title('Brent crude')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.tight_layout()
```



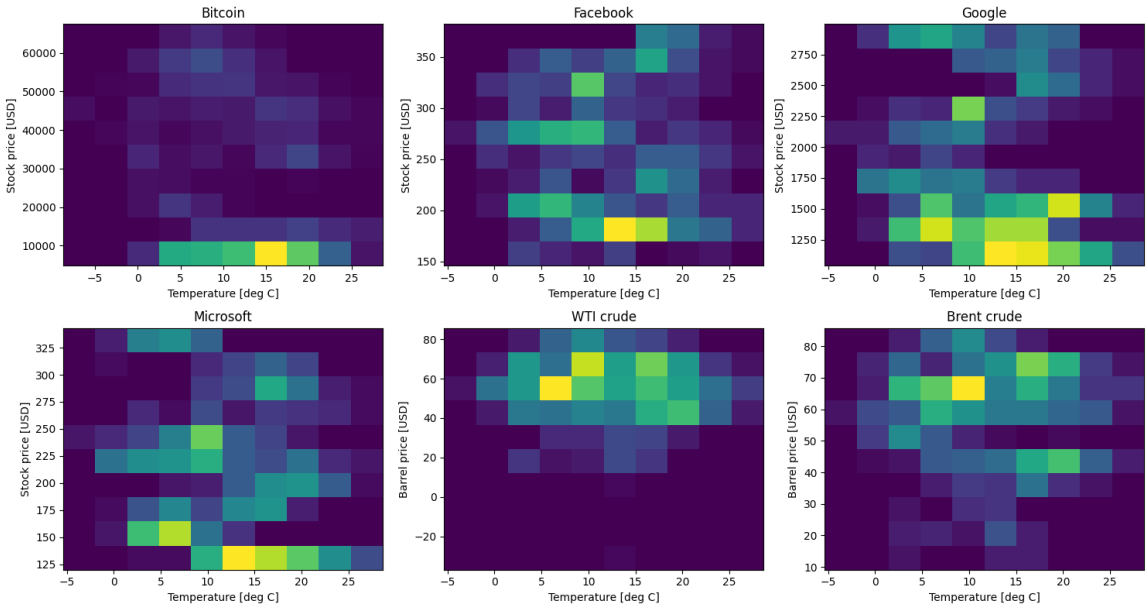
b)

i. 2d histogram

In [ ]:
```
# could not make proper 2d histograms with arrays containing NaN values
# the nan values throw off the color scaling somehow.
def removenan(x,y):
    mask = ~np.logical_or(np.isnan(x), np.isnan(y))
    x = x[mask]
    y = y[mask]
    return x,y
```

In [ ]:
```
plt.figure(figsize=(15,8))

plt.subplot(2,3,1)
x,y = removenan(basel_temp,btc)
plt.hist2d(x,y)
plt.title('Bitcoin')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,2)
x,y = removenan(basel_temp,face)
plt.hist2d(x,y)
plt.title('Facebook')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,3)
x,y = removenan(basel_temp,google)
plt.hist2d(x,y)
plt.title('Google')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,4)
x,y = removenan(basel_temp,micro)
plt.hist2d(x,y)
```

```
plt.title('Microsoft')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,5)
x,y = removenan(basel_temp,wti)
plt.hist2d(x,y)
plt.title('WTI crude')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.subplot(2,3,6)
x,y = removenan(basel_temp,brent)
plt.hist2d(x,y)
plt.title('Brent crude')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.tight_layout()
```

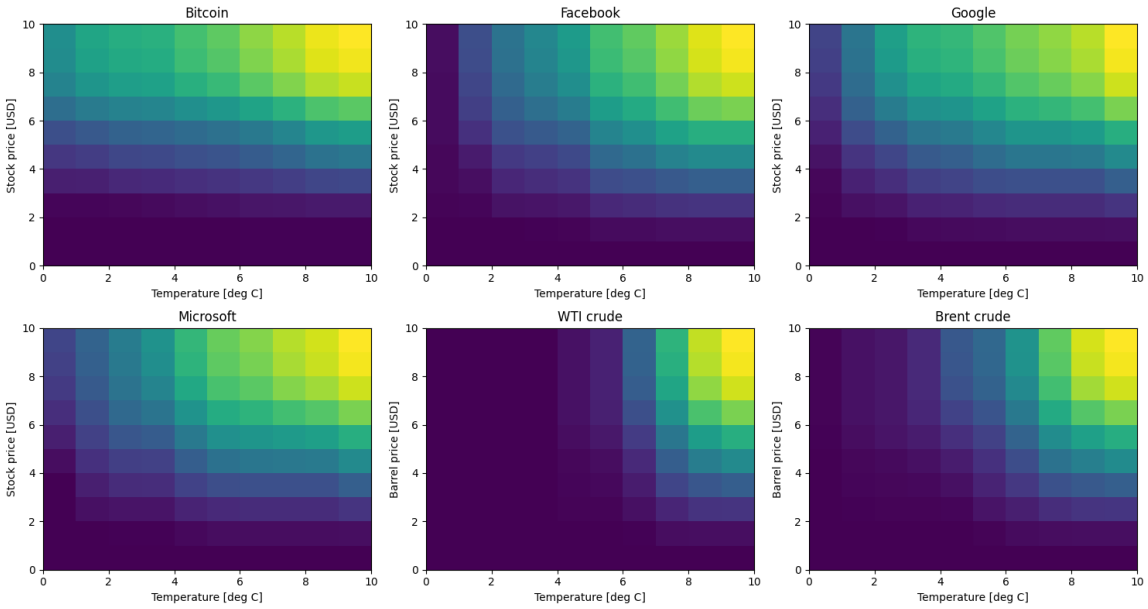


ii. 2d cumulative frequency distribution

In [ ]:
```
def cumfreq(x,y):
    x,y = removenan(x,y)
    freq = np.histogram2d(x, y)[0]
    cum_freq = np.cumsum(np.cumsum(freq,axis=1),axis=0)
    return cum_freq
```

In [ ]:
```
plt.figure(figsize=(15,8))

plt.subplot(2,3,1)
plt.pcolormesh(cumfreq(basel_temp,btc))
plt.title('Bitcoin')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,2)
plt.pcolormesh(cumfreq(basel_temp, face))
plt.title('Facebook')
plt.xlabel('Temperature [deg C]')
```

```
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,3)
plt.pcolormesh(cumfreq(basel_temp, google))
plt.title('Google')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,4)
plt.pcolormesh(cumfreq(basel_temp, micro))
plt.title('Microsoft')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,5)
plt.pcolormesh(cumfreq(basel_temp, wti))
plt.title('WTI crude')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.subplot(2,3,6)
plt.pcolormesh(cumfreq(basel_temp, brent))
plt.title('Brent crude')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.tight_layout()
```



c)

Marginal frequency

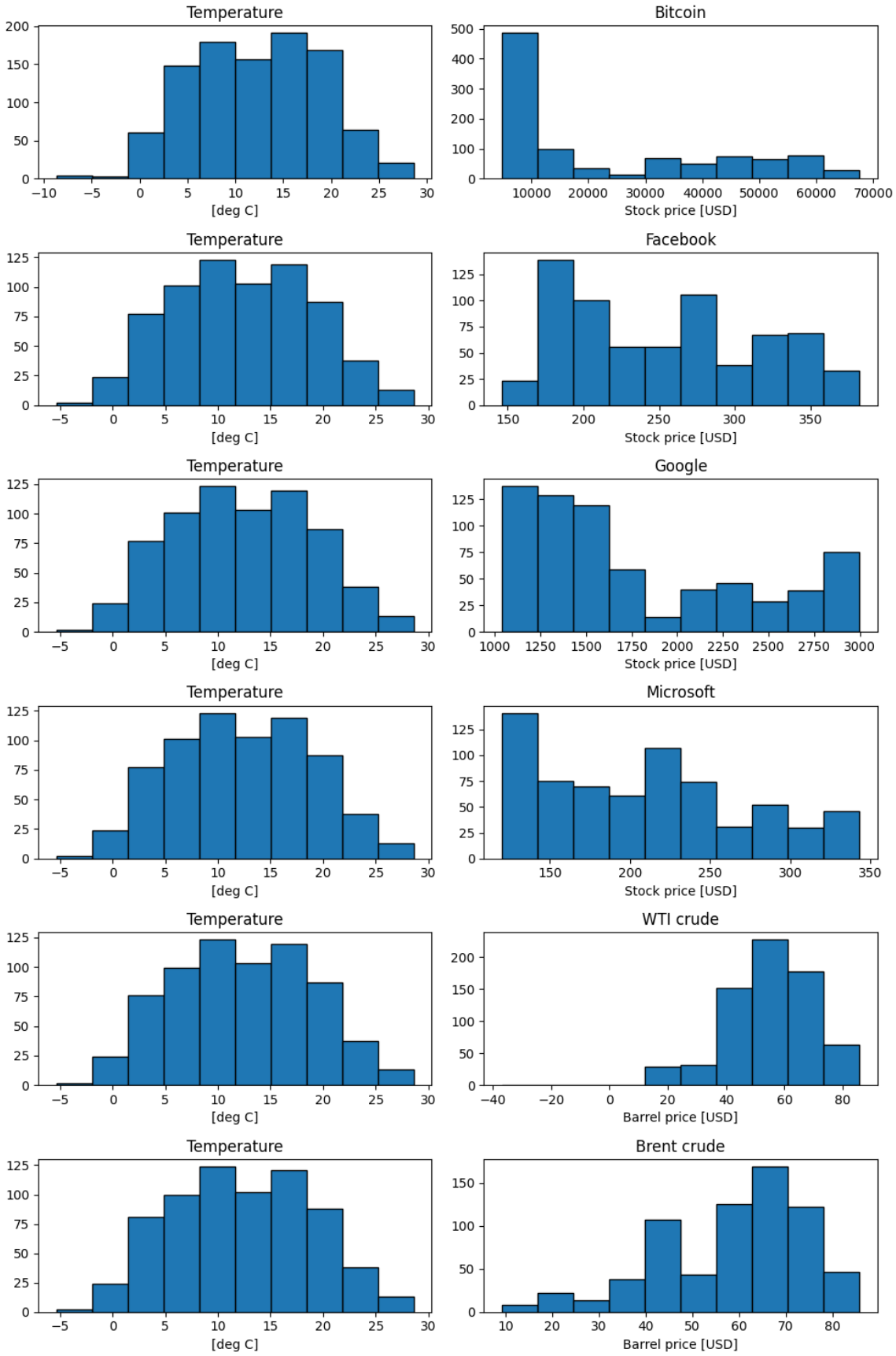
In [ ]:
```
#created function to reformulate original arrays into frequencies and bin mid-po
def marginal(x,y):
    x,y = removenan(x,y)
    freq , bin_x, bin_y = np.histogram2d(x,y)
    freq_x = np.sum(freq, axis=1)
    freq_y = np.sum(freq, axis=0)
    bin_x_mid = (bin_x[1:]+bin_x[0:-1])/2
```

```
    bin_y_mid = (bin_y[1:]+bin_y[0:-1])/2
    return freq_x, freq_y, bin_x_mid, bin_y_mid
```

Choosing to display marginal distribution for temperature for each other variable since I cannot be sure that all datasets have the same missing data points. In the figure below it looks like most data sets have the same missing datapoints, except for Bitcoin. Since other entries have been removed through the removenan function, the distribution left over will be different.

In [ ]:
```
plt.figure(figsize=(10,15))

plt.subplot(6,2,1)
freq_x, freq_y, bin_x_mid, bin_y_mid = marginal(basel_temp, btc)
plt.bar(bin_x_mid, freq_x, width=(bin_x_mid[1]-bin_x_mid[0]), edgecolor='k')
plt.title('Temperature')
plt.xlabel('[deg C]')
plt.subplot(6,2,2)
plt.bar(bin_y_mid, freq_y, width=(bin_y_mid[1]-bin_y_mid[0]), edgecolor='k')
plt.title('Bitcoin')
plt.xlabel('Stock price [USD]')


plt.subplot(6,2,3)
freq_x, freq_y, bin_x_mid, bin_y_mid = marginal(basel_temp, face)
plt.bar(bin_x_mid, freq_x, width=(bin_x_mid[1]-bin_x_mid[0]), edgecolor='k')
plt.title('Temperature')
plt.xlabel('[deg C]')
plt.subplot(6,2,4)
plt.bar(bin_y_mid, freq_y, width=(bin_y_mid[1]-bin_y_mid[0]), edgecolor='k')
plt.title('Facebook')
plt.xlabel('Stock price [USD]')


plt.subplot(6,2,5)
freq_x, freq_y, bin_x_mid, bin_y_mid = marginal(basel_temp, google)
plt.bar(bin_x_mid, freq_x, width=(bin_x_mid[1]-bin_x_mid[0]), edgecolor='k')
plt.title('Temperature')
plt.xlabel('[deg C]')
plt.subplot(6,2,6)
plt.bar(bin_y_mid, freq_y, width=(bin_y_mid[1]-bin_y_mid[0]), edgecolor='k')
plt.title('Google')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,7)
freq_x, freq_y, bin_x_mid, bin_y_mid = marginal(basel_temp, micro)
plt.bar(bin_x_mid, freq_x, width=(bin_x_mid[1]-bin_x_mid[0]), edgecolor='k')
plt.title('Temperature')
plt.xlabel('[deg C]')
plt.subplot(6,2,8)
plt.bar(bin_y_mid, freq_y, width=(bin_y_mid[1]-bin_y_mid[0]), edgecolor='k')
plt.title('Microsoft')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,9)
freq_x, freq_y, bin_x_mid, bin_y_mid = marginal(basel_temp, wti)
plt.bar(bin_x_mid, freq_x, width=(bin_x_mid[1]-bin_x_mid[0]), edgecolor='k')
plt.title('Temperature')
plt.xlabel('[deg C]')
```

```
plt.subplot(6,2,10)
plt.bar(bin_y_mid, freq_y, width=(bin_y_mid[1]-bin_y_mid[0]), edgecolor='k')
plt.title('WTI crude')
plt.xlabel('Barrel price [USD]')

plt.subplot(6,2,11)
freq_x, freq_y, bin_x_mid, bin_y_mid = marginal(basel_temp, brent)
plt.bar(bin_x_mid, freq_x, width=(bin_x_mid[1]-bin_x_mid[0]), edgecolor='k')
plt.title('Temperature')
plt.xlabel('[deg C]')
plt.subplot(6,2,12)
plt.bar(bin_y_mid, freq_y, width=(bin_y_mid[1]-bin_y_mid[0]), edgecolor='k')
plt.title('Brent crude')
plt.xlabel('Barrel price [USD]')

plt.tight_layout()
```



d)

Conditional frequency

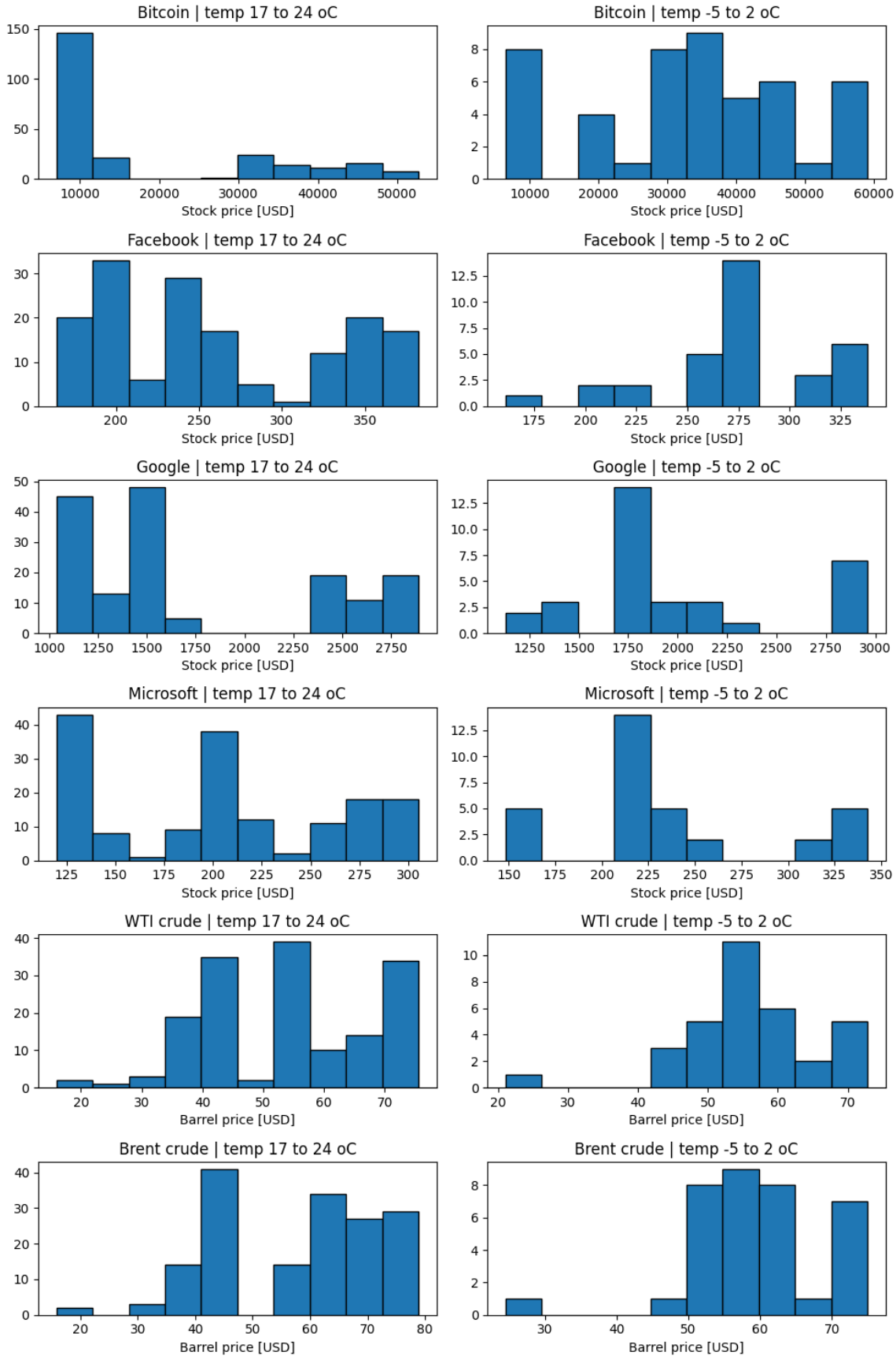
In [ ]:
```
def conditional(x, y, t1, t2):
    x, y = removenan(x,y)
```

```
    mask = np.logical_and(x>t1,  x<t2)
    return y[mask]
```

In [ ]:

```
t1 = 17
t2 = 24
t3 = -5
t4 = 2

plt.figure(figsize=(10,15))

plt.subplot(6,2,1)
plt.hist(conditional(basel_temp, btc, t1, t2), edgecolor='k')
plt.title('Bitcoin | temp 17 to 24 oC')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,2)
plt.hist(conditional(basel_temp, btc, t3, t4), edgecolor='k')
plt.title('Bitcoin | temp -5 to 2 oC')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,3)
plt.hist(conditional(basel_temp, face, t1, t2), edgecolor='k')
plt.title('Facebook | temp 17 to 24 oC')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,4)
plt.hist(conditional(basel_temp, face, t3, t4), edgecolor='k')
plt.title('Facebook | temp -5 to 2 oC')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,5)
plt.hist(conditional(basel_temp, google, t1, t2), edgecolor='k')
plt.title('Google | temp 17 to 24 oC')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,6)
plt.hist(conditional(basel_temp, google, t3, t4), edgecolor='k')
plt.title('Google | temp -5 to 2 oC')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,7)
plt.hist(conditional(basel_temp, micro, t1, t2), edgecolor='k')
plt.title('Microsoft | temp 17 to 24 oC')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,8)
plt.hist(conditional(basel_temp, micro, t3, t4), edgecolor='k')
plt.title('Microsoft | temp -5 to 2 oC')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,9)
plt.hist(conditional(basel_temp, wti, t1, t2), edgecolor='k')
plt.title('WTI crude | temp 17 to 24 oC')
plt.xlabel('Barrel price [USD]')

plt.subplot(6,2,10)
plt.hist(conditional(basel_temp, wti, t3, t4), edgecolor='k')
plt.title('WTI crude | temp -5 to 2 oC')
plt.xlabel('Barrel price [USD]')
```

```
plt.subplot(6,2,11)
plt.hist(conditional(basel_temp, brent, t1, t2), edgecolor='k')
plt.title('Brent crude | temp 17 to 24 oC')
plt.xlabel('Barrel price [USD]')

plt.subplot(6,2,12)
plt.hist(conditional(basel_temp, brent, t3, t4), edgecolor='k')
plt.title('Brent crude | temp -5 to 2 oC')
plt.xlabel('Barrel price [USD]')

plt.tight_layout()
```



2. Covariance and Correlation

a)

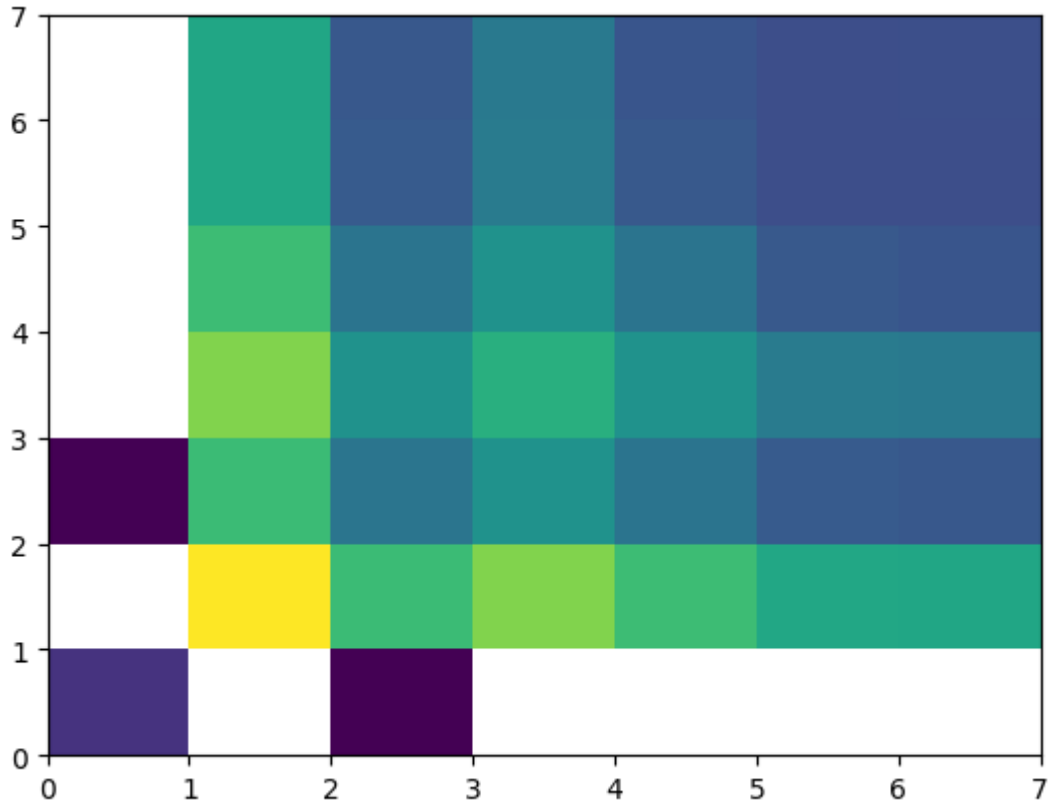
In [ ]:
```
mask = ~(np.isnan(basel_temp) | np.isnan(btc) | np.isnan(face) | np.isnan(google
basel_c = basel_temp[mask]
```

```
btc_c = btc[mask]
face_c = face[mask]
google_c = google[mask]
micro_c = micro[mask]
wti_c = wti[mask]
brent_c = brent[mask]
```

In [ ]:
```
matrix = np.asmatrix(np.c_[basel_c, btc_c, face_c, google_c, micro_c, wti_c, bre
covariance = np.cov(matrix.T)
plt.pcolormesh(np.log(covariance))
print(covariance)
```

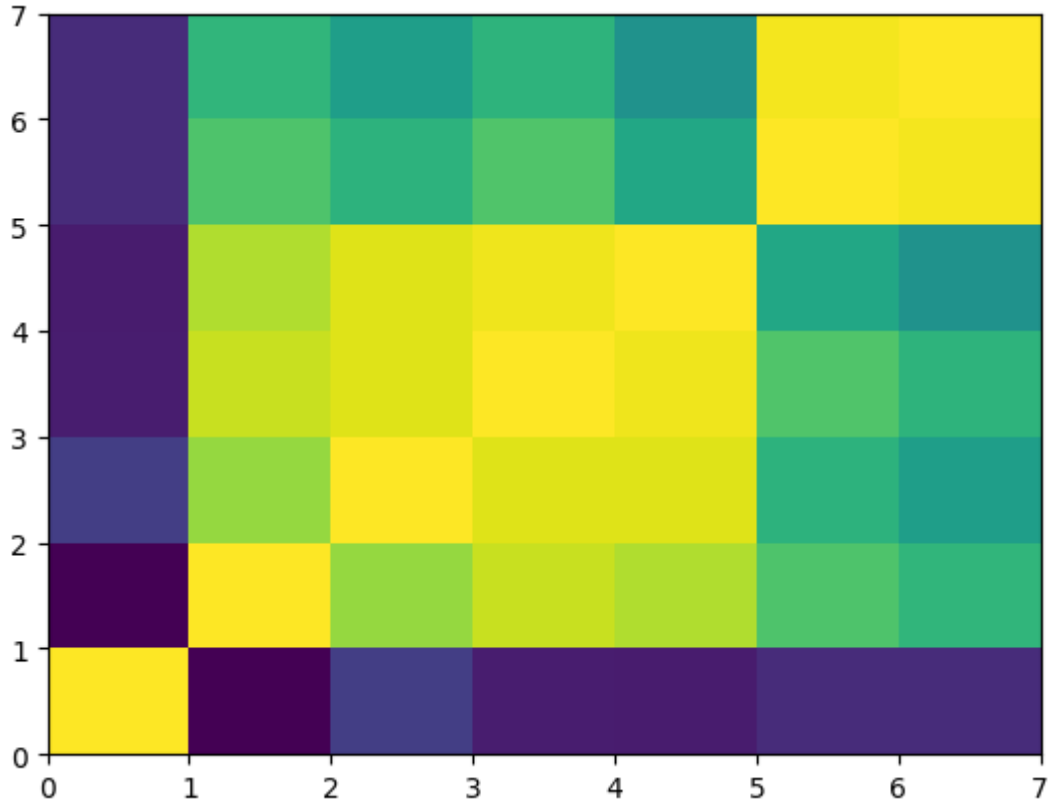
```
[[ 4.31174941e+01 -2.66663236e+04  2.76903962e+00 -4.60996487e+02
  -4.80856796e+01 -6.34338299e+00 -6.34710319e+00]
 [-2.66663236e+04  3.65240465e+08  9.61710183e+05  1.03411610e+07
   1.00388103e+06  1.92859104e+05  1.76011428e+05]
 [ 2.76903962e+00  9.61710183e+05  3.88695980e+03  3.52888739e+04
   3.59734651e+03  5.37665134e+02  4.56962072e+02]
 [-4.60996487e+02  1.03411610e+07  3.52888739e+04  3.61538078e+05
   3.56685778e+04  6.11627650e+03  5.45702193e+03]
 [-4.80856796e+01  1.00388103e+06  3.59734651e+03  3.56685778e+04
   3.76015971e+03  4.79581887e+02  3.91171632e+02]
 [-6.34338299e+00  1.92859104e+05  5.37665134e+02  6.11627650e+03
   4.79581887e+02  2.33521620e+02  2.36551517e+02]
 [-6.34710319e+00  1.76011428e+05  4.56962072e+02  5.45702193e+03
   3.91171632e+02  2.36551517e+02  2.50697219e+02]]
```

```
C:\Users\danfy\AppData\Local\Temp\ipykernel_45852\1468787112.py:3: RuntimeWarni
ng: invalid value encountered in log
  plt.pcolormesh(np.log(covariance))
```



In [ ]:
```
correlation_coeff = np.corrcoef(matrix.T)
plt.pcolormesh(correlation_coeff)
print(correlation_coeff)
```

```
[[ 1.         -0.21249413  0.0067639  -0.11675993 -0.11942242 -0.06321653
  -0.06104837]
 [-0.21249413  1.          0.80714163  0.89991778  0.85662286  0.66036973
   0.58167   ]
 [ 0.0067639   0.80714163  1.          0.94136003  0.94096637  0.56434321
   0.46291406]
 [-0.11675993  0.89991778  0.94136003  1.          0.96739898  0.66565093
   0.57319701]
 [-0.11942242  0.85662286  0.94096637  0.96739898  1.          0.51179505
   0.40289275]
 [-0.06321653  0.66036973  0.56434321  0.66565093  0.51179505  1.
   0.97765907]
 [-0.06104837  0.58167     0.46291406  0.57319701  0.40289275  0.97765907
   1.        ]]
```



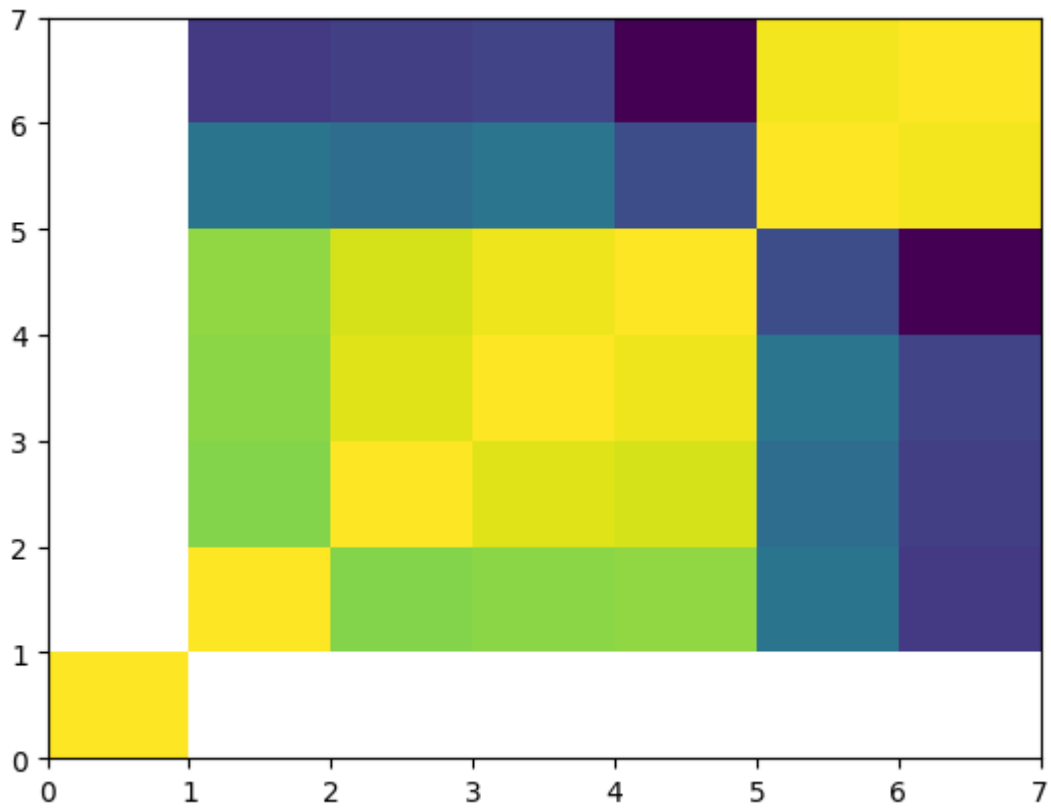
b)

In [ ]:
```
basel_r = ss.rankdata(basel_c)
btc_r = ss.rankdata(btc_c)
face_r = ss.rankdata(face_c)
google_r = ss.rankdata(google_c)
micro_r = ss.rankdata(micro_c)
wti_r = ss.rankdata(wti_c)
brent_r = ss.rankdata(brent_c)
matrix_r = np.asmatrix(np.c_[basel_r, btc_r, face_r, google_r, micro_r, wti_r, b
```

In [ ]:
```
covariance = np.cov(matrix_r.T)
plt.pcolormesh(np.log(covariance))
print(covariance)
```

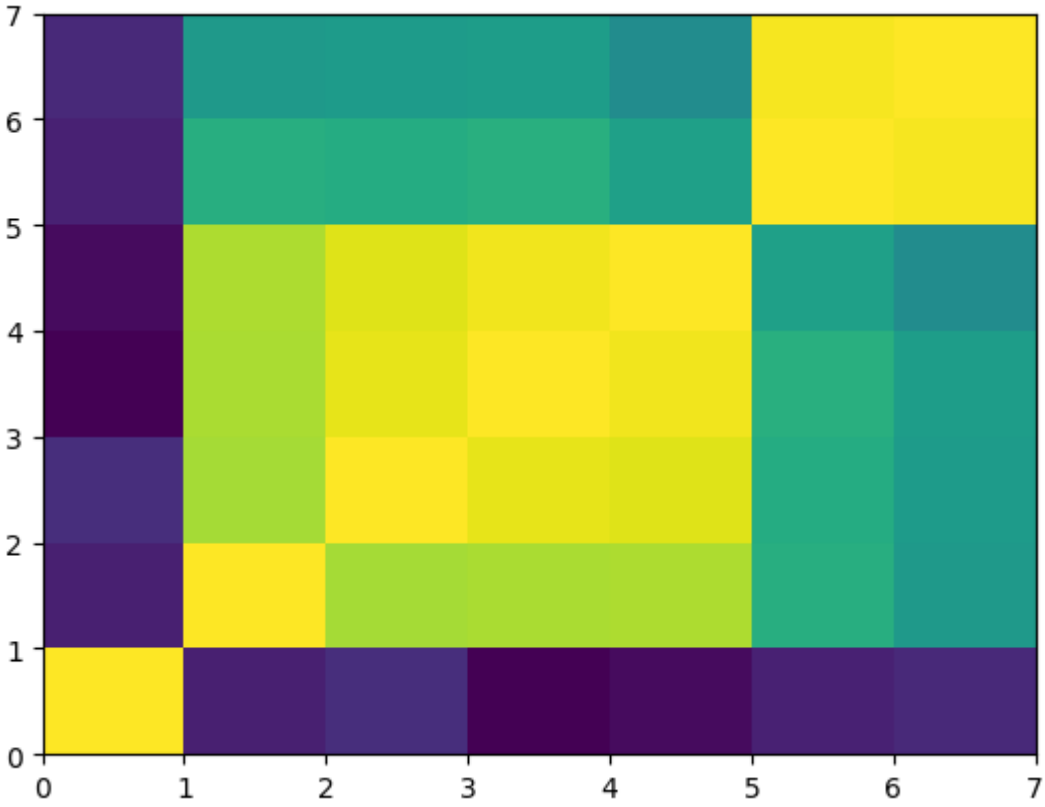
```
[[38024.99925816 -2690.15504451  -788.96513353 -6558.49332344
  -5246.79821958 -2528.04896142 -1463.70845697]
 [-2690.15504451 38025.         32012.11869436 32396.31305638
  32565.67878338 21412.80934718 17467.29080119]
 [ -788.96513353 32012.11869436 38024.99109792 36290.87537092
  35823.09272997 20848.35571217 17779.52596439]
 [-6558.49332344 32396.31305638 36290.87537092 38024.99777448
  37142.97811573 21515.85311573 18075.16172107]
 [-5246.79821958 32565.67878338 35823.09272997 37142.97811573
  38024.9888724  18610.68434718 14978.68954006]
 [-2528.04896142 21412.80934718 20848.35571217 21515.85311573
  18610.68434718 38024.95252226 37396.46216617]
 [-1463.70845697 17467.29080119 17779.52596439 18075.16172107
  14978.68954006 37396.46216617 38024.95771513]]
```

```
C:\Users\danfy\AppData\Local\Temp\ipykernel_45852\3552126990.py:2: RuntimeWarni
ng: invalid value encountered in log
  plt.pcolormesh(np.log(covariance))
```



In [ ]:
```
correlation_coeff = np.corrcoef(matrix_r.T)
plt.pcolormesh(correlation_coeff)
print(correlation_coeff)
```

```
[[ 1.         -0.07074701 -0.02074859 -0.17247846 -0.13798288 -0.06648391
  -0.03849334]
 [-0.07074701  1.          0.84187041  0.85197407  0.85642823  0.56312486
   0.45936359]
 [-0.02074859  0.84187041  1.          0.95439528  0.94209342  0.54828064
   0.46757496]
 [-0.17247846  0.85197407  0.95439528  1.          0.97680433  0.56583477
   0.4753497 ]
 [-0.13798288  0.85642823  0.94209342  0.97680433  1.          0.48943323
   0.39391716]
 [-0.06648391  0.56312486  0.54828064  0.56583477  0.48943323  1.
   0.98347157]
 [-0.03849334  0.45936359  0.46757496  0.4753497   0.39391716  0.98347157
   1.        ]]
```



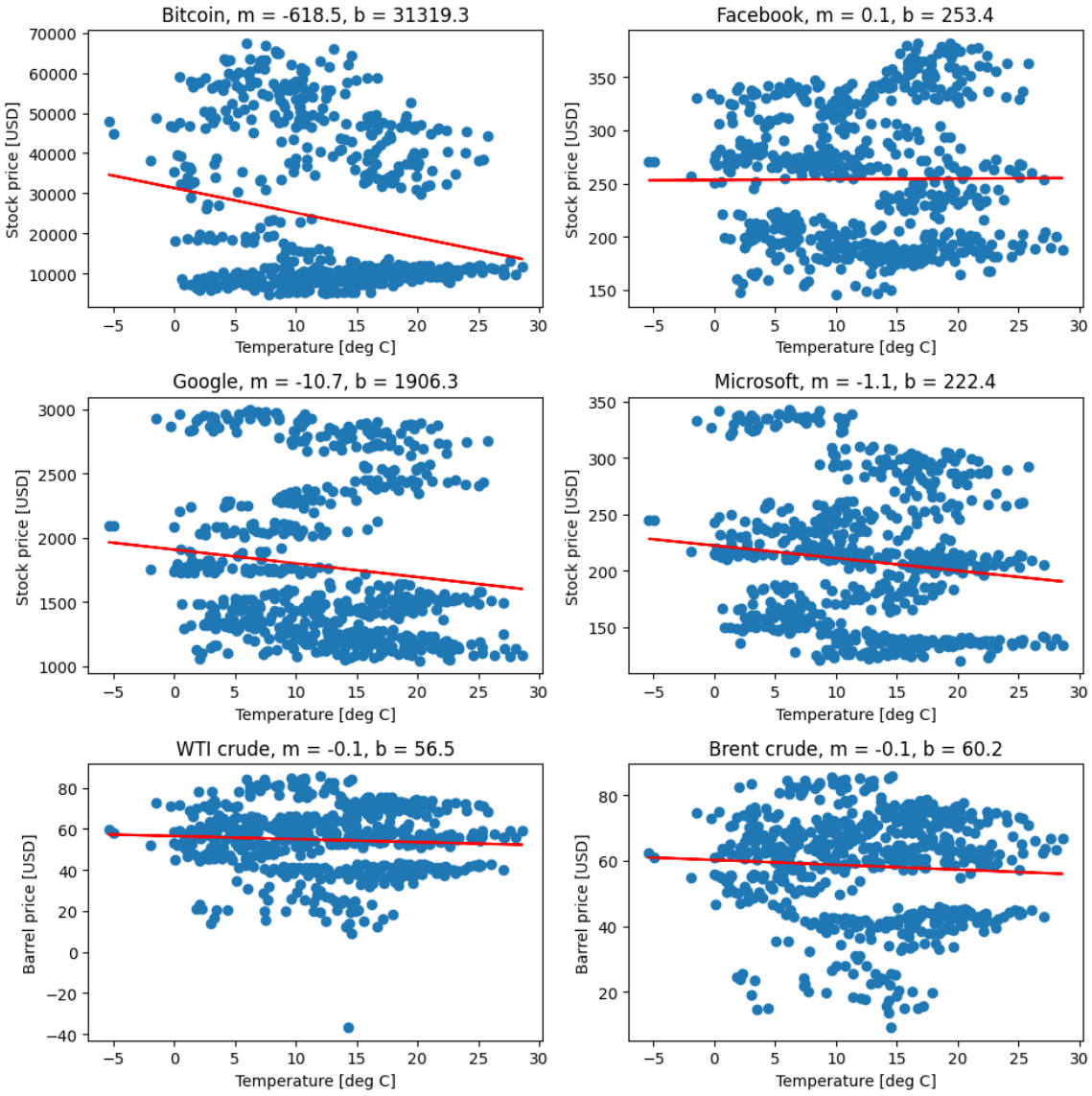
3. Regression

a), b) Calculate regression line

In [ ]:
```
#for simplicities sake I choose to continue working with the cleaned data vector
def regression(x, y):
    ones_vector = np.ones(len(x))
    X = np.c_[ones_vector, x]
    X = np.asmatrix(X)
    Y = np.asmatrix(y).T
    beta = np.linalg.inv(X.T*X)*(X.T*Y)
    return np.asarray(X*beta).flatten(), np.asarray(beta).flatten(), np.asarray(
```

In [ ]:
```
plt.figure(figsize=(10,10))

plt.subplot(3,2,1)
Y_hat, beta, error = regression(basel_c, btc_c)
plt.plot(basel_c, Y_hat, 'r-')
```

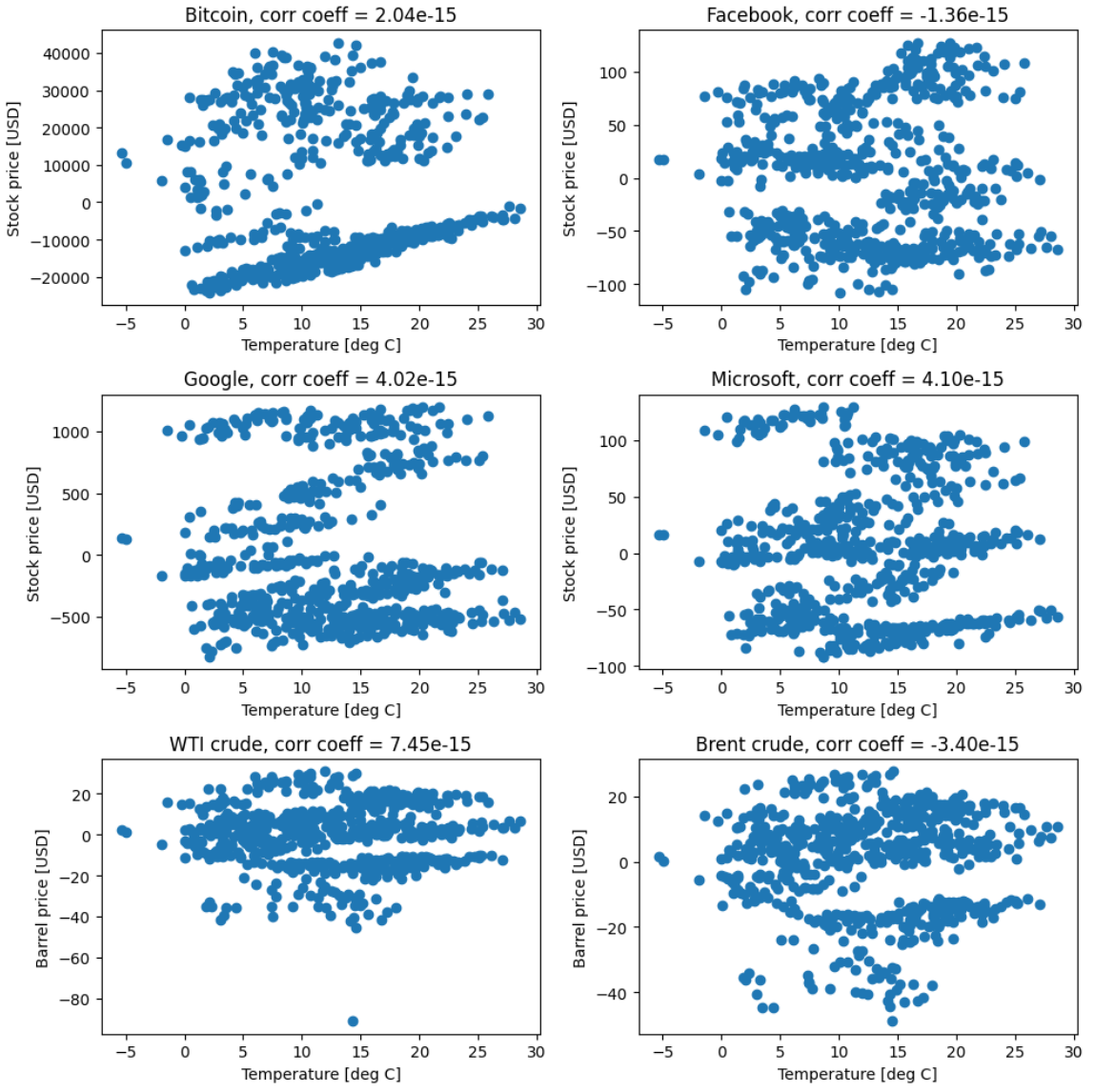
```
plt.scatter(basel_c,btc_c)
plt.title(f'Bitcoin, m = {beta[1]:.1f}, b = {beta[0]:.1f}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(3,2,2)
Y_hat, beta, error = regression(basel_c, face_c)
plt.plot(basel_c, Y_hat, 'r-')
plt.scatter(basel_c,face_c)
plt.title(f'Facebook, m = {beta[1]:.1f}, b = {beta[0]:.1f}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(3,2,3)
Y_hat, beta, error = regression(basel_c, google_c)
plt.plot(basel_c, Y_hat, 'r-')
plt.scatter(basel_c,google_c)
plt.title(f'Google, m = {beta[1]:.1f}, b = {beta[0]:.1f}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(3,2,4)
Y_hat, beta, error = regression(basel_c, micro_c)
plt.plot(basel_c, Y_hat, 'r-')
plt.scatter(basel_c,micro_c)
plt.title(f'Microsoft, m = {beta[1]:.1f}, b = {beta[0]:.1f}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(3,2,5)
Y_hat, beta, error = regression(basel_c, wti_c)
plt.plot(basel_c, Y_hat, 'r-')
plt.scatter(basel_c,wti_c)
plt.title(f'WTI crude, m = {beta[1]:.1f}, b = {beta[0]:.1f}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.subplot(3,2,6)
Y_hat, beta, error = regression(basel_c, brent_c)
plt.plot(basel_c, Y_hat, 'r-')
plt.scatter(basel_c,brent_c)
plt.title(f'Brent crude, m = {beta[1]:.1f}, b = {beta[0]:.1f}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.tight_layout()
```



c) Error / residual

In [ ]:
```
plt.figure(figsize=(10,10))

plt.subplot(3,2,1)
Y_hat, beta, error = regression(basel_c, btc_c)
plt.scatter(basel_c, error)
plt.title(f'Bitcoin, corr coeff = {np.corrcoef(basel_c, error)[0,1]:.2e}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(3,2,2)
Y_hat, beta, error = regression(basel_c, face_c)
plt.scatter(basel_c, error)
plt.title(f'Facebook, corr coeff = {np.corrcoef(basel_c, error)[0,1]:.2e}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(3,2,3)
Y_hat, beta, error = regression(basel_c, google_c)
plt.scatter(basel_c, error)
plt.title(f'Google, corr coeff = {np.corrcoef(basel_c, error)[0,1]:.2e}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')
```

```
plt.subplot(3,2,4)
Y_hat, beta, error = regression(basel_c, micro_c)
plt.scatter(basel_c, error)
plt.title(f'Microsoft, corr coeff = {np.corrcoef(basel_c, error)[0,1]:.2e}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(3,2,5)
Y_hat, beta, error = regression(basel_c, wti_c)
plt.scatter(basel_c, error)
plt.title(f'WTI crude, corr coeff = {np.corrcoef(basel_c, error)[0,1]:.2e}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.subplot(3,2,6)
Y_hat, beta, error = regression(basel_c, brent_c)
plt.scatter(basel_c, error)
plt.title(f'Brent crude, corr coeff = {np.corrcoef(basel_c, error)[0,1]:.2e}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.tight_layout()
```



In [ ]:
```
# Investigating closeness of distributions to normal distributions
```

In [ ]:
```
Y_hat, beta, error = regression(basel_c, btc_c)
print(f'Bitcoin prediction error mean = {np.mean(error):.2e}, var = {np.var(erro

Y_hat, beta, error = regression(basel_c, face_c)
print(f'Facebook prediction error mean = {np.mean(error):.2e}, var = {np.var(err

Y_hat, beta, error = regression(basel_c, google_c)
print(f'Google  prediction error mean = {np.mean(error):.2e}, var = {np.var(erro

Y_hat, beta, error = regression(basel_c, micro_c)
print(f'Microsoft prediction error mean = {np.mean(error):.2e}, var = {np.var(er

Y_hat, beta, error = regression(basel_c, wti_c)
print(f'WTI crude prediction error mean = {np.mean(error):.2e}, var = {np.var(er

Y_hat, beta, error = regression(basel_c, brent_c)
print(f'Brent crude  prediction error mean = {np.mean(error):.2e}, var = {np.var
```

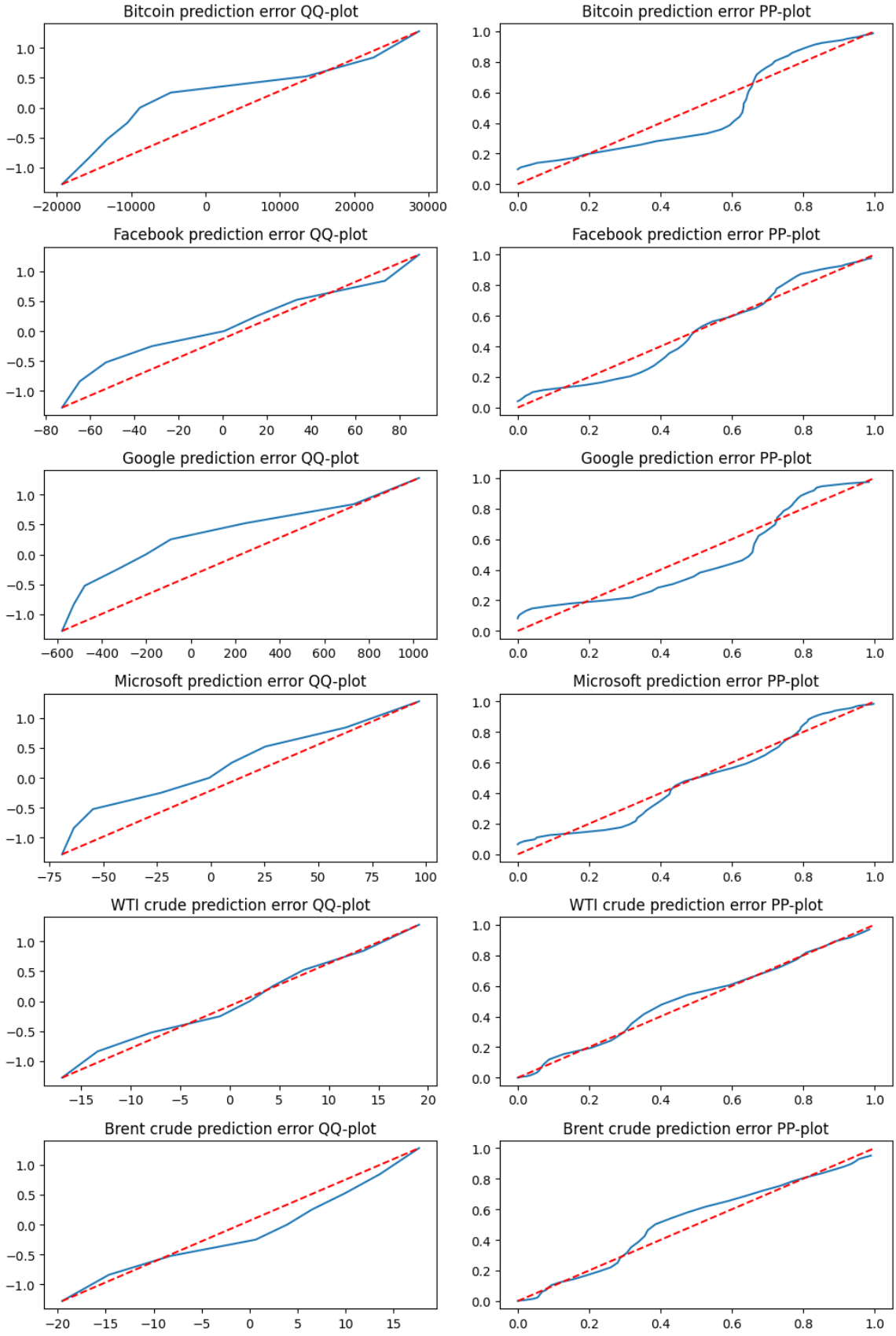
```
Bitcoin prediction error mean = -6.90e-12, var = 3.48e+08
Facebook prediction error mean = 1.51e-13, var = 3.88e+03
Google  prediction error mean = -1.29e-12, var = 3.56e+05
Microsoft prediction error mean = -1.35e-13, var = 3.70e+03
WTI crude prediction error mean = -3.03e-14, var = 2.32e+02
Brent crude  prediction error mean = -3.03e-15, var = 2.49e+02
```

In [ ]:
```
def qq_pp(y):
    range_qq = np.arange(0.1,1,0.1)
    range_pp = np.arange(np.min(y), np.max(y),(np.max(y)- np.min(y))/50)

    Norm_qq = st.norm.ppf(range_qq,loc=0,scale=1)
    Norm_pp = st.norm.cdf(range_pp,loc=0,scale= np.sqrt(np.var(y)))

    n = len(y)
    y_sort = np.sort(y)
    F = (np.arange(n))/(n+1)
    y_qq = np.interp(range_qq , F , y_sort)
    y_pp = np.interp(range_pp , y_sort , F)
    return Norm_qq, Norm_pp, y_qq, y_pp
```

For the below code I thought of a way to compress the iterations into a for loop to avoid copy pasting and manually changing stuff as much. I did not retroactively update code above to this format.

In [ ]:
```
plt.figure(figsize=(10,15))

data = np.c_[btc_c, face_c, google_c, micro_c, wti_c, brent_c]
titles = ('Bitcoin', 'Facebook', 'Google', 'Microsoft', 'WTI crude', 'Brent cruc
labels = ('Stock', 'Stock', 'Stock', 'Stock', 'Barrel', 'Barrel')

for i in range(len(data[0])):
    Y_hat, beta, error = regression(basel_c, data[:,i])
    Norm_qq, Norm_pp, y_qq, y_pp = qq_pp(error)
    plt.subplot(6,2,2*i+1)
    plt.plot(y_qq, Norm_qq)
    plt.plot([np.min(y_qq),np.max(y_qq)], [np.min(Norm_qq),np.max(Norm_qq)],'--r
    plt.title(f'{titles[i]} prediction error QQ-plot')

    plt.subplot(6,2,2*i+2)
```

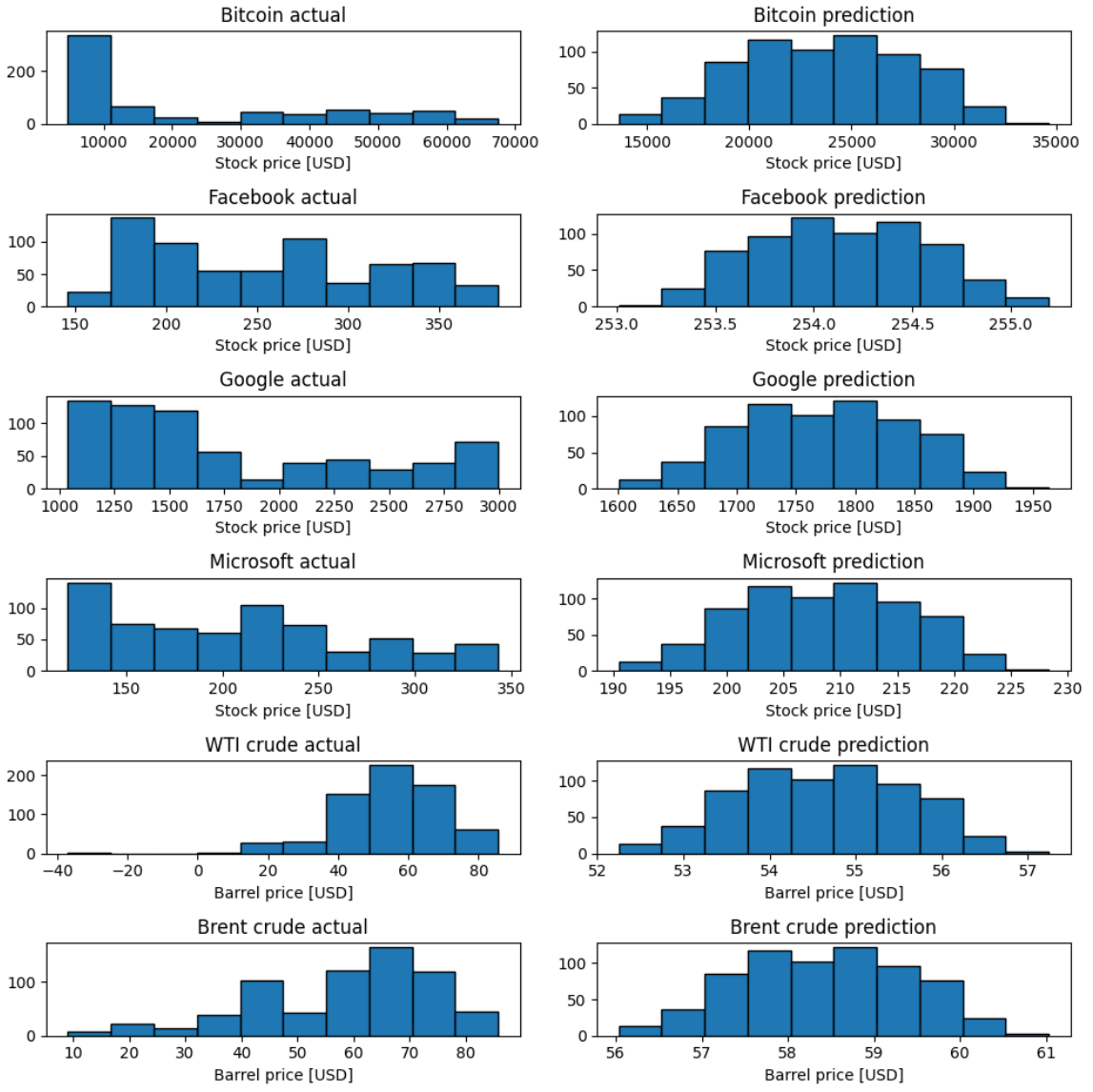
```
    plt.plot(y_pp, Norm_pp)
    plt.plot([0,1], [0,1], 'r--')
    plt.title(f'{titles[i]} prediction error PP-plot')

plt.tight_layout()
```



The predictions are obviously useless when we are trying to make them based on data that has no connection at all.

d) Histograms

In [ ]:
```
plt.figure(figsize=(10,10))

data = np.c_[btc_c, face_c, google_c, micro_c, wti_c, brent_c]
titles = ('Bitcoin', 'Facebook', 'Google', 'Microsoft', 'WTI crude', 'Brent cruc
labels = ('Stock', 'Stock', 'Stock', 'Stock', 'Barrel', 'Barrel')

for i in range(len(data[0])):
    plt.subplot(6,2,2*i+1)
    plt.hist(data[:,i], edgecolor='k')
    plt.title(f'{titles[i]} actual')
    plt.xlabel(f'{labels[i]} price [USD]')

    plt.subplot(6,2,2*i+2)
    Y_hat, beta, error = regression(basel_c, data[:,i])
    plt.hist(Y_hat, edgecolor='k')
    plt.title(f'{titles[i]} prediction')
    plt.xlabel(f'{labels[i]} price [USD]')

plt.tight_layout()
```



Mean and variance of actual and prediction

In [ ]:
```
data = np.c_[btc_c, face_c, google_c, micro_c, wti_c, brent_c]
titles = ('Bitcoin', 'Facebook', 'Google', 'Microsoft', 'WTI crude', 'Brent cruc
labels = ('Stock', 'Stock', 'Stock', 'Stock', 'Barrel', 'Barrel')

for i in range(len(data[0])):
    Y_hat, beta, error = regression(basel_c, data[:,i])
    print(f'{titles[i]} actual---- mean = {np.mean(data[:,i]):.2e}, var = {np.va
    print(f'{titles[i]} prediction mean = {np.mean(Y_hat):.2e}, var = {np.var(Y_
```

```
Bitcoin actual---- mean = 2.37e+04, var = 3.65e+08
Bitcoin prediction mean = 2.37e+04, var = 1.65e+07
Facebook actual---- mean = 2.54e+02, var = 3.88e+03
Facebook prediction mean = 2.54e+02, var = 1.78e-01
Google actual---- mean = 1.77e+03, var = 3.61e+05
Google prediction mean = 1.77e+03, var = 4.92e+03
Microsoft actual---- mean = 2.09e+02, var = 3.75e+03
Microsoft prediction mean = 2.09e+02, var = 5.35e+01
WTI crude actual---- mean = 5.46e+01, var = 2.33e+02
WTI crude prediction mean = 5.46e+01, var = 9.32e-01
Brent crude actual---- mean = 5.84e+01, var = 2.50e+02
Brent crude prediction mean = 5.84e+01, var = 9.33e-01
```

The mean actual vs predictions are all almost identical, as previously calculated for error. This follows by definition of the regression algorithm. The variance of the prediction is related to the absolute value of the slope of the regression curve. A slope of zero would yield a variance of zero as well. A low prediction variance can in such a case indicate that the prediction is indifferent to the predicting variable.

4. Redo everything for slightly different data for no apparent reason

For Q 1-3 it was not specified for how which time interval the analysis should be performed, so I extended it as far as the input data would allow. The limiting factor was the data for Bitcoin stock, where the first entry was 15.04.2019.

Since all datasets except for Bitcoin stock already contain more than 5 years of data, I downloaded a new data sheet for BTC stock containing more than 5 years of data.

Notice that for this section we are importing BTC data from BTC-USD as compared to BTC_USD for the first go-around.

In [ ]:
```
# Creating new cleaning function for new range. It would have been more elegant
def clean2(date, x):
    df = pd.DataFrame({'date': date,'argument': x})
    df['date'] = pd.to_datetime(df['date'])
    date_range = pd.date_range(start='2017-01-03', end='2022-01-03', freq='D')

    # reindex dataframe with the date range
    array = np.asarray(df.set_index('date').reindex(date_range).reset_index())
```

```
    array = np.array(array[:,1], dtype=float)
    return array
```

In [ ]:
```
# sending each input file through data cleaning function.
# Values for skiprows and index for argument column found by inspecting files.

df = pd.read_excel('Basel_Temp_History.xlsx',skiprows=10, header=None)
basel_temp = clean2(df[0], df[3])

df = pd.read_csv('Brent_Spot_Price.csv',skiprows=5,header=None)
brent = clean2(df[0], df[1])

df = pd.read_csv('BTC-USD.csv',skiprows=1,header=None)
btc = clean2(df[0], df[4])

df = pd.read_csv('facebook.csv',skiprows=1,header=None)
face = clean2(df[0], df[4])

df = pd.read_csv('google.csv',skiprows=1,header=None)
google = clean2(df[0], df[4])

df = pd.read_csv('microsoft.csv',skiprows=1,header=None)
micro = clean2(df[0], df[4])

df = pd.read_csv('WTI_Spot_Price.csv',skiprows=5,header=None)
wti = clean2(df[0], df[1])
```

In [ ]:
```
len(wti)
```

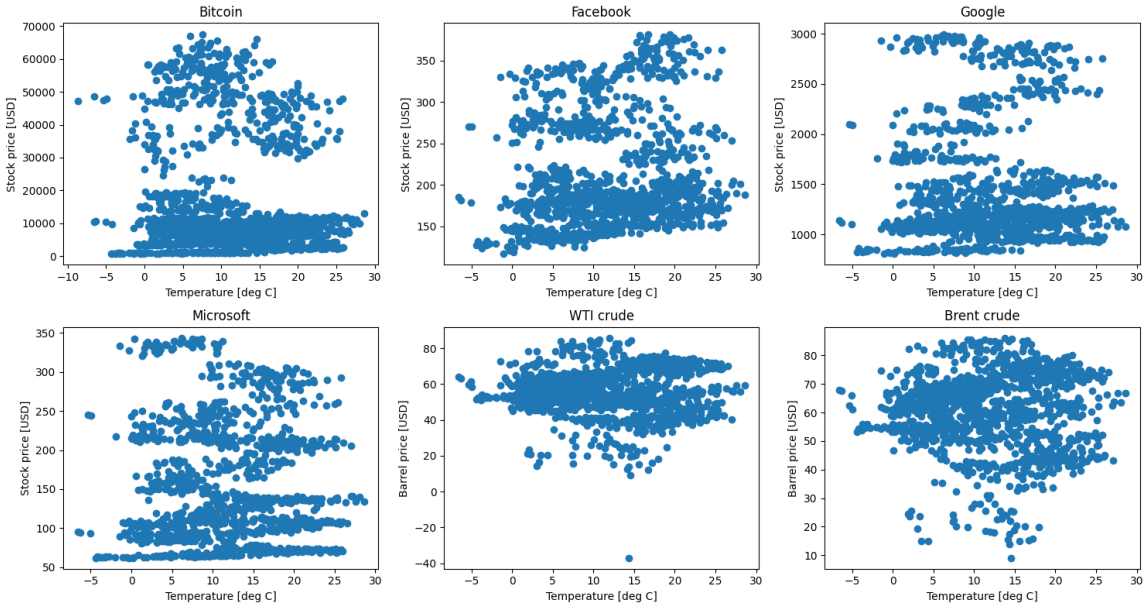
Out[ ]:
```
1827
```

a)

Scatterplot

In [ ]:
```
plt.figure(figsize=(15,8))

plt.subplot(2,3,1)
plt.scatter(basel_temp,btc)
plt.title('Bitcoin')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,2)
plt.scatter(basel_temp,face)
plt.title('Facebook')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,3)
plt.scatter(basel_temp,google)
plt.title('Google')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,4)
plt.scatter(basel_temp,micro)
```

```
plt.title('Microsoft')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,5)
plt.scatter(basel_temp,wti)
plt.title('WTI crude')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.subplot(2,3,6)
plt.scatter(basel_temp,brent)
plt.title('Brent crude')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.tight_layout()
```



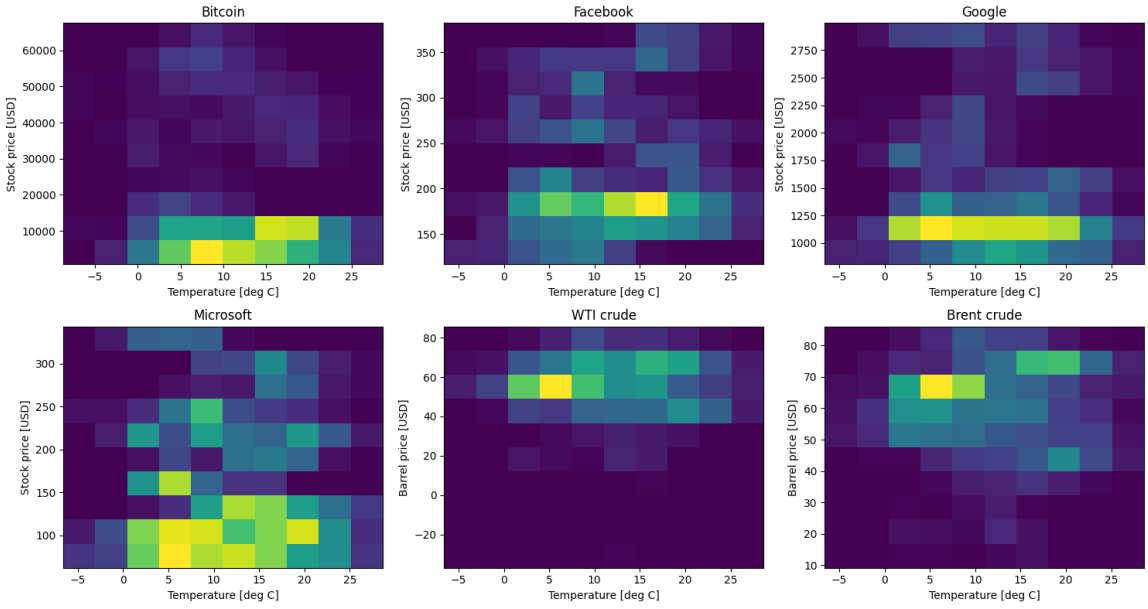
b)

i. 2d histogram

In [ ]:
```
# could not make proper 2d histograms with arrays containing NaN values
# the nan values throw off the color scaling somehow.
def removenan(x,y):
    mask = ~np.logical_or(np.isnan(x), np.isnan(y))
    x = x[mask]
    y = y[mask]
    return x,y
```

In [ ]:
```
plt.figure(figsize=(15,8))

plt.subplot(2,3,1)
x,y = removenan(basel_temp,btc)
plt.hist2d(x,y)
plt.title('Bitcoin')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')
```

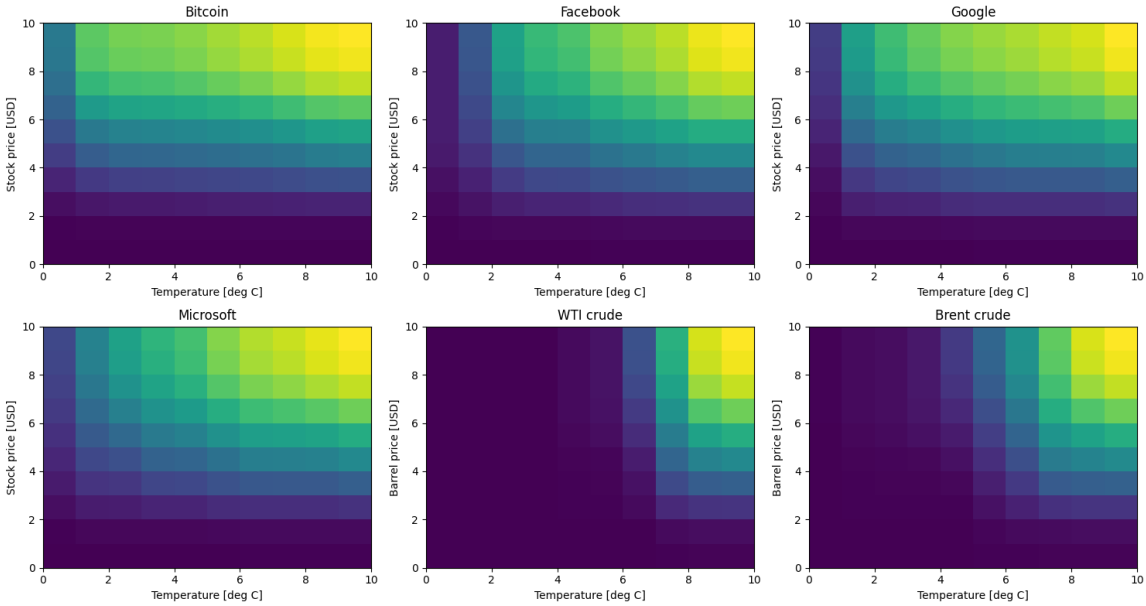
```
plt.subplot(2,3,2)
x,y = removenan(basel_temp,face)
plt.hist2d(x,y)
plt.title('Facebook')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,3)
x,y = removenan(basel_temp,google)
plt.hist2d(x,y)
plt.title('Google')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,4)
x,y = removenan(basel_temp,micro)
plt.hist2d(x,y)
plt.title('Microsoft')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,5)
x,y = removenan(basel_temp,wti)
plt.hist2d(x,y)
plt.title('WTI crude')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.subplot(2,3,6)
x,y = removenan(basel_temp,brent)
plt.hist2d(x,y)
plt.title('Brent crude')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.tight_layout()
```



ii. 2d cumulative frequency distribution

In [ ]:
```
def cumfreq(x,y):
    x,y = removenan(x,y)
    freq = np.histogram2d(x, y)[0]
    cum_freq = np.cumsum(np.cumsum(freq,axis=1),axis=0)
    return cum_freq
```

In [ ]:
```
plt.figure(figsize=(15,8))

plt.subplot(2,3,1)
plt.pcolormesh(cumfreq(basel_temp,btc))
plt.title('Bitcoin')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,2)
plt.pcolormesh(cumfreq(basel_temp, face))
plt.title('Facebook')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,3)
plt.pcolormesh(cumfreq(basel_temp, google))
plt.title('Google')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,4)
plt.pcolormesh(cumfreq(basel_temp, micro))
plt.title('Microsoft')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(2,3,5)
plt.pcolormesh(cumfreq(basel_temp, wti))
plt.title('WTI crude')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.subplot(2,3,6)
plt.pcolormesh(cumfreq(basel_temp, brent))
plt.title('Brent crude')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.tight_layout()
```



c)

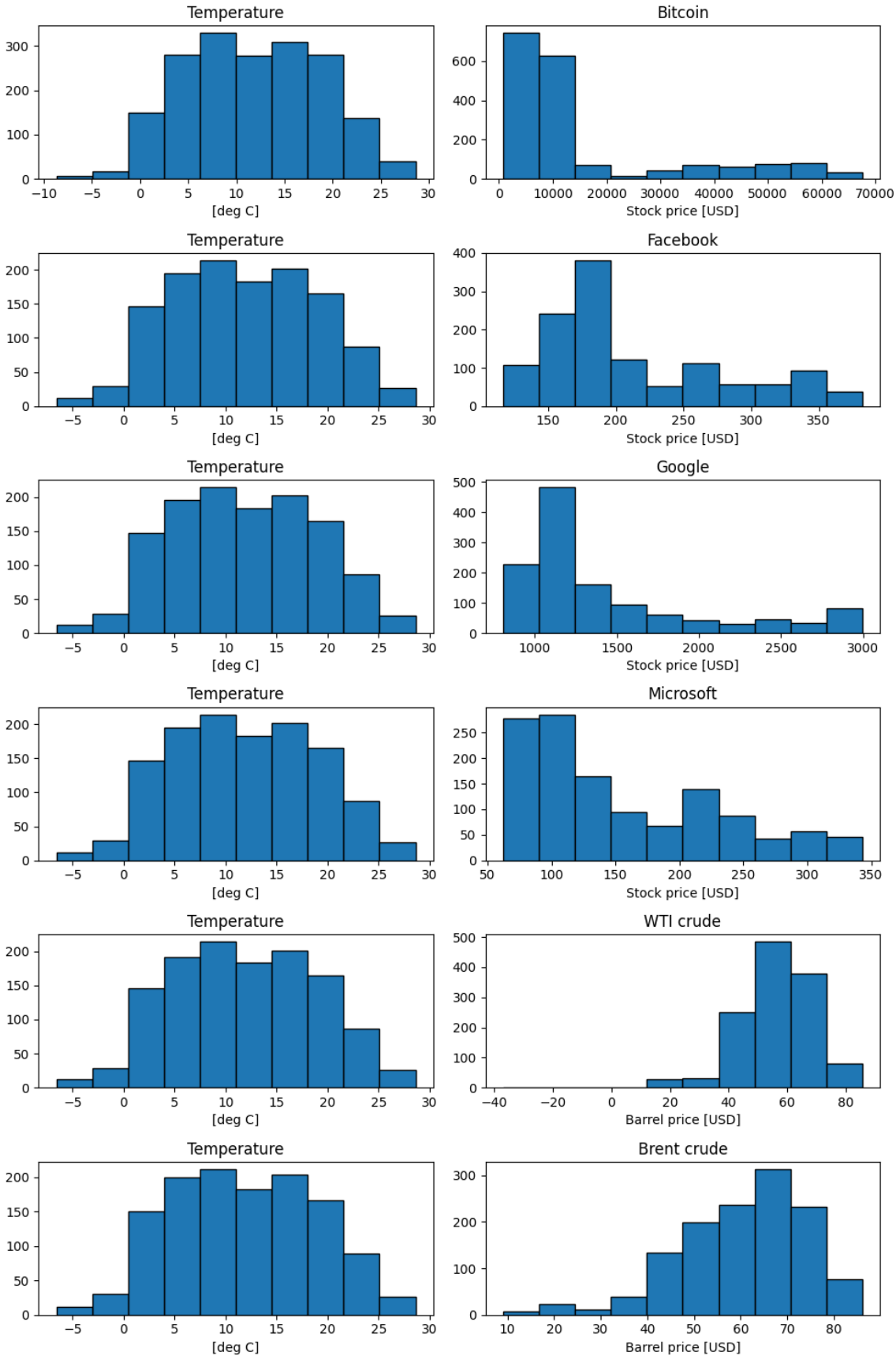
Marginal frequency

In [ ]:
```
#created function to reformulate original arrays into frequencies and bin mid-po
def marginal(x,y):
    x,y = removenan(x,y)
    freq , bin_x, bin_y = np.histogram2d(x,y)
    freq_x = np.sum(freq, axis=1)
    freq_y = np.sum(freq, axis=0)
    bin_x_mid = (bin_x[1:]+bin_x[0:-1])/2
    bin_y_mid = (bin_y[1:]+bin_y[0:-1])/2
    return freq_x, freq_y, bin_x_mid, bin_y_mid
```

Choosing to display marginal distribution for temperature for each other variable since I cannot be sure that all datasets have the same missing data points. In the figure below it looks like most data sets have the same missing datapoints, except for Bitcoin. Since other entries have been removed through the removenan function, the distribution left over will be different.

In [ ]:
```
plt.figure(figsize=(10,15))

plt.subplot(6,2,1)
freq_x, freq_y, bin_x_mid, bin_y_mid = marginal(basel_temp, btc)
plt.bar(bin_x_mid, freq_x, width=(bin_x_mid[1]-bin_x_mid[0]), edgecolor='k')
plt.title('Temperature')
plt.xlabel('[deg C]')
plt.subplot(6,2,2)
plt.bar(bin_y_mid, freq_y, width=(bin_y_mid[1]-bin_y_mid[0]), edgecolor='k')
plt.title('Bitcoin')
plt.xlabel('Stock price [USD]')


plt.subplot(6,2,3)
freq_x, freq_y, bin_x_mid, bin_y_mid = marginal(basel_temp, face)
plt.bar(bin_x_mid, freq_x, width=(bin_x_mid[1]-bin_x_mid[0]), edgecolor='k')
plt.title('Temperature')
```

```
plt.xlabel('[deg C]')
plt.subplot(6,2,4)
plt.bar(bin_y_mid, freq_y, width=(bin_y_mid[1]-bin_y_mid[0]), edgecolor='k')
plt.title('Facebook')
plt.xlabel('Stock price [USD]')


plt.subplot(6,2,5)
freq_x, freq_y, bin_x_mid, bin_y_mid = marginal(basel_temp, google)
plt.bar(bin_x_mid, freq_x, width=(bin_x_mid[1]-bin_x_mid[0]), edgecolor='k')
plt.title('Temperature')
plt.xlabel('[deg C]')
plt.subplot(6,2,6)
plt.bar(bin_y_mid, freq_y, width=(bin_y_mid[1]-bin_y_mid[0]), edgecolor='k')
plt.title('Google')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,7)
freq_x, freq_y, bin_x_mid, bin_y_mid = marginal(basel_temp, micro)
plt.bar(bin_x_mid, freq_x, width=(bin_x_mid[1]-bin_x_mid[0]), edgecolor='k')
plt.title('Temperature')
plt.xlabel('[deg C]')
plt.subplot(6,2,8)
plt.bar(bin_y_mid, freq_y, width=(bin_y_mid[1]-bin_y_mid[0]), edgecolor='k')
plt.title('Microsoft')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,9)
freq_x, freq_y, bin_x_mid, bin_y_mid = marginal(basel_temp, wti)
plt.bar(bin_x_mid, freq_x, width=(bin_x_mid[1]-bin_x_mid[0]), edgecolor='k')
plt.title('Temperature')
plt.xlabel('[deg C]')
plt.subplot(6,2,10)
plt.bar(bin_y_mid, freq_y, width=(bin_y_mid[1]-bin_y_mid[0]), edgecolor='k')
plt.title('WTI crude')
plt.xlabel('Barrel price [USD]')

plt.subplot(6,2,11)
freq_x, freq_y, bin_x_mid, bin_y_mid = marginal(basel_temp, brent)
plt.bar(bin_x_mid, freq_x, width=(bin_x_mid[1]-bin_x_mid[0]), edgecolor='k')
plt.title('Temperature')
plt.xlabel('[deg C]')
plt.subplot(6,2,12)
plt.bar(bin_y_mid, freq_y, width=(bin_y_mid[1]-bin_y_mid[0]), edgecolor='k')
plt.title('Brent crude')
plt.xlabel('Barrel price [USD]')

plt.tight_layout()
```



d)

Conditional frequency

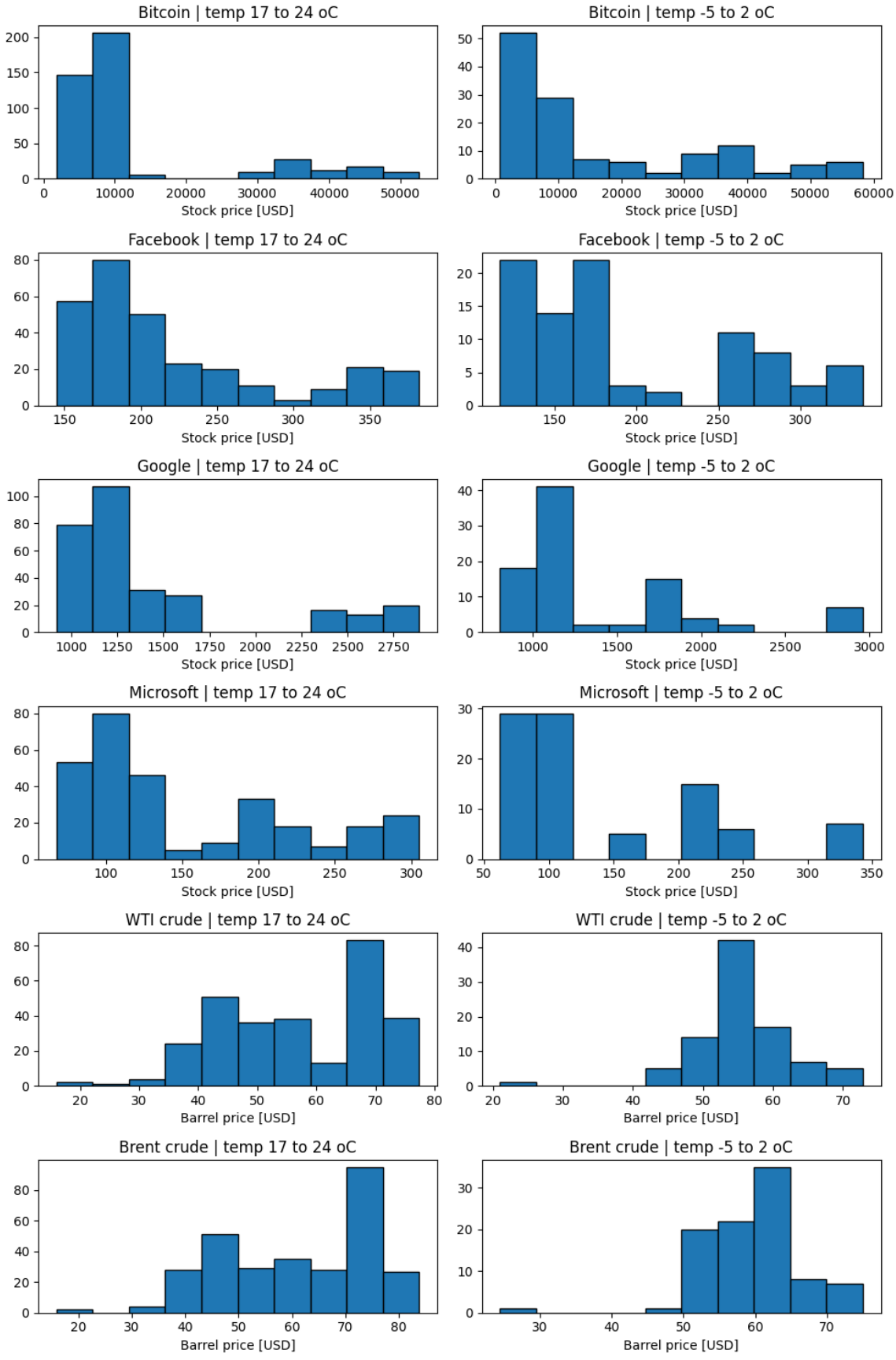
In [ ]:
```
def conditional(x, y, t1, t2):
    x, y = removenan(x,y)
```

```
    mask = np.logical_and(x>t1,  x<t2)
    return y[mask]
```

In [ ]:

```
t1 = 17
t2 = 24
t3 = -5
t4 = 2

plt.figure(figsize=(10,15))

plt.subplot(6,2,1)
plt.hist(conditional(basel_temp, btc, t1, t2), edgecolor='k')
plt.title('Bitcoin | temp 17 to 24 oC')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,2)
plt.hist(conditional(basel_temp, btc, t3, t4), edgecolor='k')
plt.title('Bitcoin | temp -5 to 2 oC')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,3)
plt.hist(conditional(basel_temp, face, t1, t2), edgecolor='k')
plt.title('Facebook | temp 17 to 24 oC')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,4)
plt.hist(conditional(basel_temp, face, t3, t4), edgecolor='k')
plt.title('Facebook | temp -5 to 2 oC')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,5)
plt.hist(conditional(basel_temp, google, t1, t2), edgecolor='k')
plt.title('Google | temp 17 to 24 oC')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,6)
plt.hist(conditional(basel_temp, google, t3, t4), edgecolor='k')
plt.title('Google | temp -5 to 2 oC')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,7)
plt.hist(conditional(basel_temp, micro, t1, t2), edgecolor='k')
plt.title('Microsoft | temp 17 to 24 oC')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,8)
plt.hist(conditional(basel_temp, micro, t3, t4), edgecolor='k')
plt.title('Microsoft | temp -5 to 2 oC')
plt.xlabel('Stock price [USD]')

plt.subplot(6,2,9)
plt.hist(conditional(basel_temp, wti, t1, t2), edgecolor='k')
plt.title('WTI crude | temp 17 to 24 oC')
plt.xlabel('Barrel price [USD]')

plt.subplot(6,2,10)
plt.hist(conditional(basel_temp, wti, t3, t4), edgecolor='k')
plt.title('WTI crude | temp -5 to 2 oC')
plt.xlabel('Barrel price [USD]')
```

```
plt.subplot(6,2,11)
plt.hist(conditional(basel_temp, brent, t1, t2), edgecolor='k')
plt.title('Brent crude | temp 17 to 24 oC')
plt.xlabel('Barrel price [USD]')

plt.subplot(6,2,12)
plt.hist(conditional(basel_temp, brent, t3, t4), edgecolor='k')
plt.title('Brent crude | temp -5 to 2 oC')
plt.xlabel('Barrel price [USD]')

plt.tight_layout()
```



2. Covariance and Correlation

a)

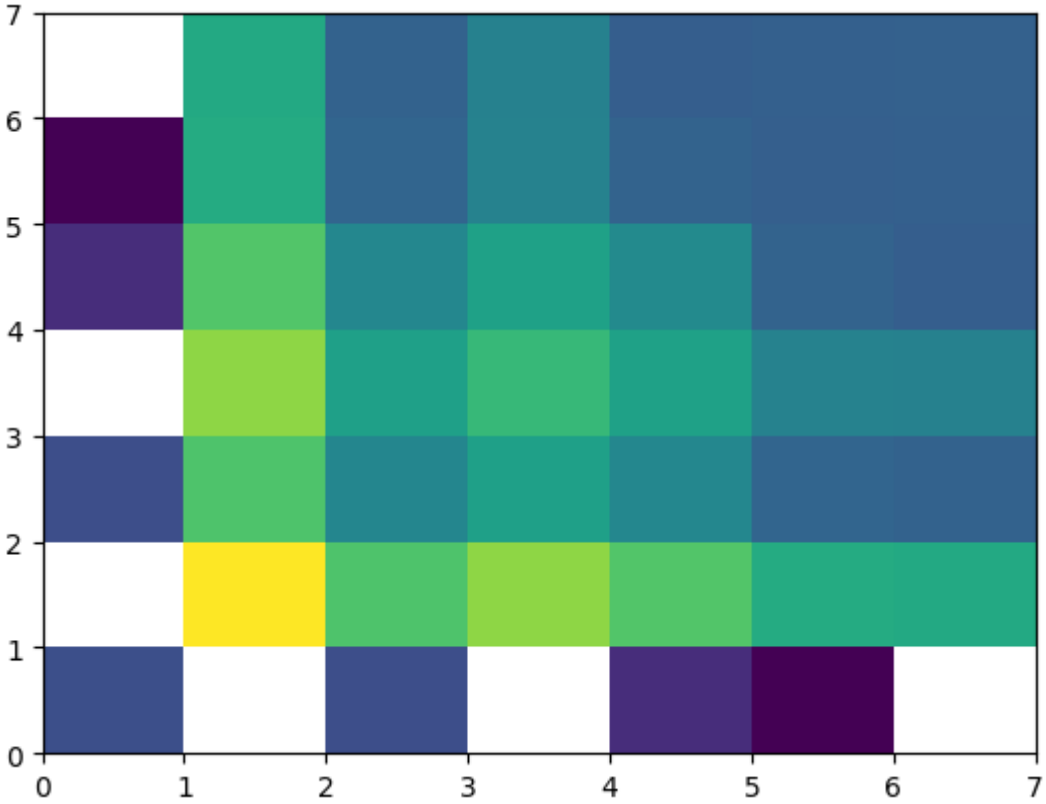
In [ ]:
```
mask = ~(np.isnan(basel_temp) | np.isnan(btc) | np.isnan(face) | np.isnan(google
basel_c = basel_temp[mask]
```

```
btc_c = btc[mask]
face_c = face[mask]
google_c = google[mask]
micro_c = micro[mask]
wti_c = wti[mask]
brent_c = brent[mask]
```

In [ ]:
```
matrix = np.asmatrix(np.c_[basel_c, btc_c, face_c, google_c, micro_c, wti_c, bre
covariance = np.cov(matrix.T)
plt.pcolormesh(np.log(covariance))
print(covariance)
```

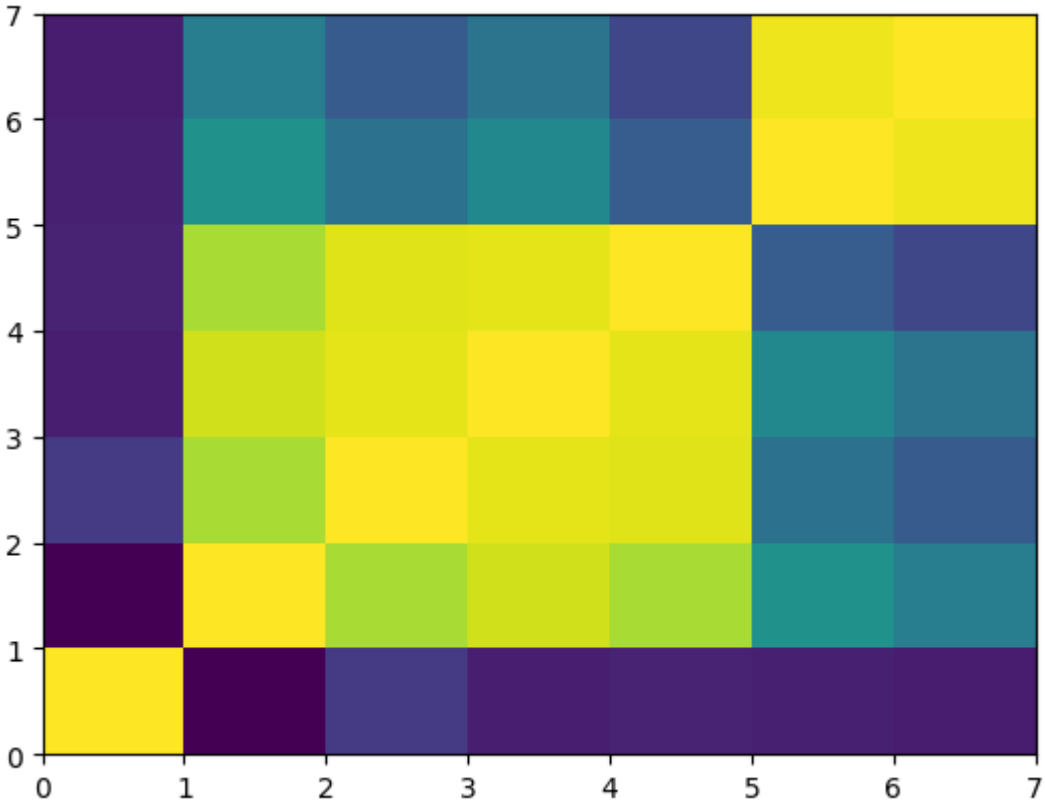
```
[[ 4.85580938e+01 -1.11220462e+04  4.48620010e+01 -1.61593809e+00
   4.74678664e+00  3.37039554e-01 -1.00920729e+00]
 [-1.11220462e+04  2.87192216e+08  9.51718309e+05  9.02872900e+06
   1.09616133e+06  9.86523245e+04  8.48695470e+04]
 [ 4.48620010e+01  9.51718309e+05  4.30239941e+03  3.61993940e+04
   4.68398042e+03  2.64662600e+02  1.94555393e+02]
 [-1.61593809e+00  9.02872900e+06  3.61993940e+04  3.35052803e+05
   4.18250843e+04  3.08886447e+03  2.56565997e+03]
 [ 4.74678664e+00  1.09616133e+06  4.68398042e+03  4.18250843e+04
   5.70889546e+03  2.14597564e+02  1.42554284e+02]
 [ 3.37039554e-01  9.86523245e+04  2.64662600e+02  3.08886447e+03
   2.14597564e+02  1.62961778e+02  1.69981593e+02]
 [-1.00920729e+00  8.48695470e+04  1.94555393e+02  2.56565997e+03
   1.42554284e+02  1.69981593e+02  1.87309093e+02]]
```

```
C:\Users\danfy\AppData\Local\Temp\ipykernel_45852\1468787112.py:3: RuntimeWarni
ng: invalid value encountered in log
  plt.pcolormesh(np.log(covariance))
```



In [ ]:
```
correlation_coeff = np.corrcoef(matrix.T)
plt.pcolormesh(correlation_coeff)
print(correlation_coeff)
```

```
[[ 1.00000000e+00 -9.41819217e-02  9.81504880e-02 -4.00624287e-04
   9.01556299e-03  3.78884798e-03 -1.05820579e-02]
 [-9.41819217e-02  1.00000000e+00  8.56183255e-01  9.20414937e-01
   8.56075561e-01  4.56013935e-01  3.65919979e-01]
 [ 9.81504880e-02  8.56183255e-01  1.00000000e+00  9.53431093e-01
   9.45112840e-01  3.16078048e-01  2.16724735e-01]
 [-4.00624287e-04  9.20414937e-01  9.53431093e-01  1.00000000e+00
   9.56322013e-01  4.18022203e-01  3.23864272e-01]
 [ 9.01556299e-03  8.56075561e-01  9.45112840e-01  9.56322013e-01
   1.00000000e+00  2.22487628e-01  1.37855722e-01]
 [ 3.78884798e-03  4.56013935e-01  3.16078048e-01  4.18022203e-01
   2.22487628e-01  1.00000000e+00  9.72925503e-01]
 [-1.05820579e-02  3.65919979e-01  2.16724735e-01  3.23864272e-01
   1.37855722e-01  9.72925503e-01  1.00000000e+00]]
```



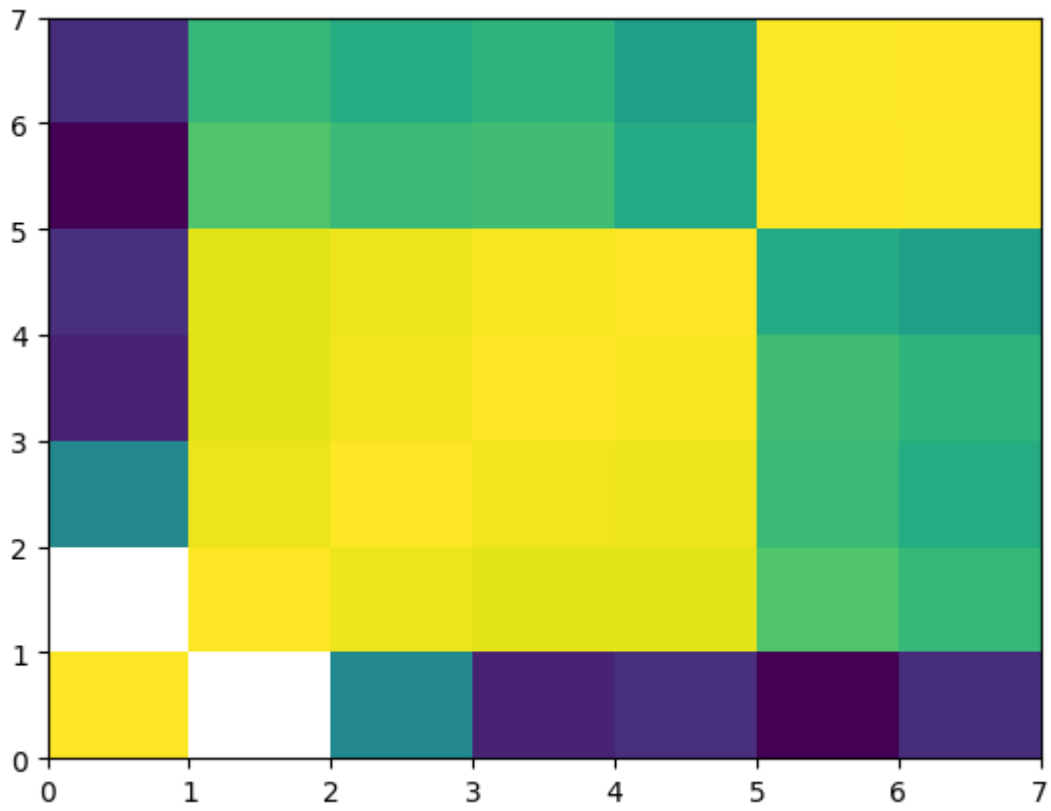
b)

In [ ]:
```
basel_r = ss.rankdata(basel_c)
btc_r = ss.rankdata(btc_c)
face_r = ss.rankdata(face_c)
google_r = ss.rankdata(google_c)
micro_r = ss.rankdata(micro_c)
wti_r = ss.rankdata(wti_c)
brent_r = ss.rankdata(brent_c)
matrix_r = np.asmatrix(np.c_[basel_r, btc_r, face_r, google_r, micro_r, wti_r, b
```

In [ ]:
```
covariance = np.cov(matrix_r.T)
plt.pcolormesh(np.log(covariance))
print(covariance)
```

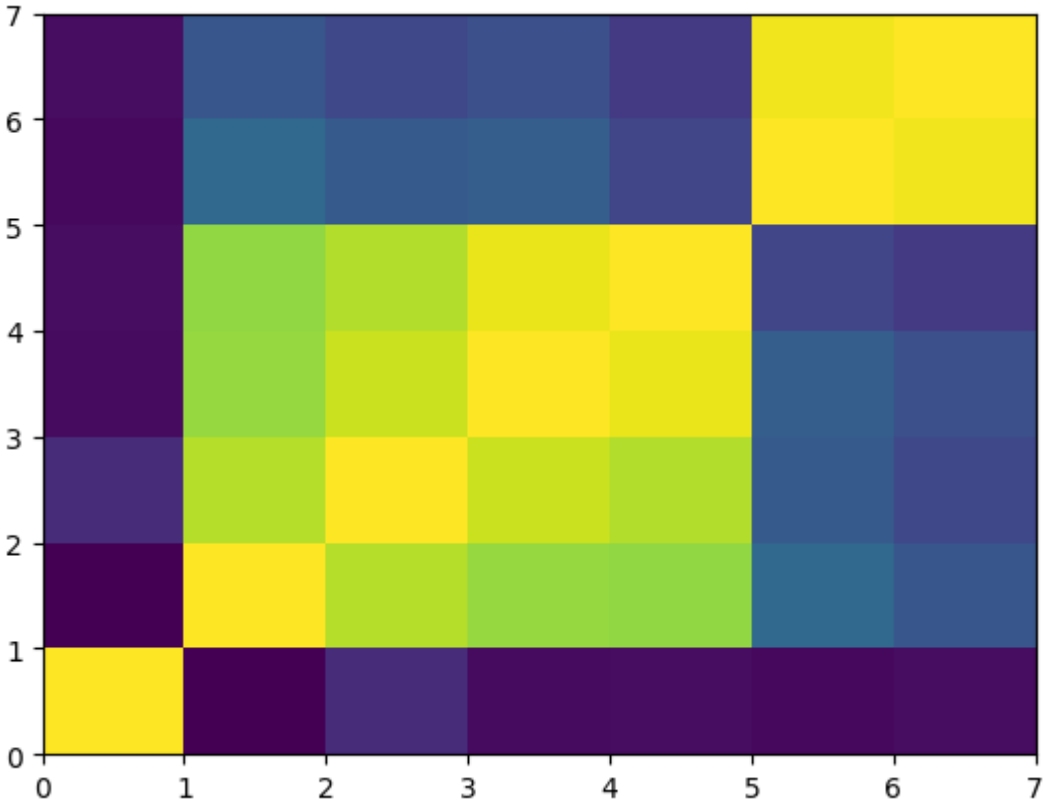
```
[[128236.66545601   -453.30064568  15218.31476998   3436.03551251
    4004.70661824   2345.57808717   3972.99697337]
 [  -453.30064568 128236.66585956 113849.0794996  107727.88942696
  106993.96206618  42368.10794996  33792.90657789]
 [ 15218.31476998 113849.0794996  128236.6440678  117779.68099274
  113558.82485876  35355.33535109  27521.99313963]
 [  3436.03551251 107727.88942696 117779.68099274 128236.6622276
  124104.52239709  37397.59422922  30955.17292171]
 [  4004.70661824 106993.96206618 113558.82485876 124104.52239709
  128236.64285714  26267.21428571  21612.82990315]
 [  2345.57808717  42368.10794996  35355.33535109  37397.59422922
   26267.21428571 128236.56981437 125582.19895077]
 [  3972.99697337  33792.90657789  27521.99313963  30955.17292171
   21612.82990315 125582.19895077 128236.5811138 ]]
```

```
C:\Users\danfy\AppData\Local\Temp\ipykernel_45852\3552126990.py:2: RuntimeWarni
ng: invalid value encountered in log
  plt.pcolormesh(np.log(covariance))
```



In [ ]:
```
correlation_coeff = np.corrcoef(matrix_r.T)
plt.pcolormesh(correlation_coeff)
print(correlation_coeff)
```

```
[[ 1.         -0.00353488  0.11867367  0.02679449  0.03122903  0.01829102
   0.03098177]
 [-0.00353488  1.          0.8878045   0.8400709   0.83434773  0.33039009
   0.26351993]
 [ 0.11867367  0.8878045   1.          0.91845567  0.88554115  0.27570392
   0.21461884]
 [ 0.02679449  0.8400709   0.91845567  1.          0.96777731  0.29162961
   0.24139105]
 [ 0.03122903  0.83434773  0.88554115  0.96777731  1.          0.20483398
   0.16853868]
 [ 0.01829102  0.33039009  0.27570392  0.29162961  0.20483398  1.
   0.97930094]
 [ 0.03098177  0.26351993  0.21461884  0.24139105  0.16853868  0.97930094
   1.        ]]
```



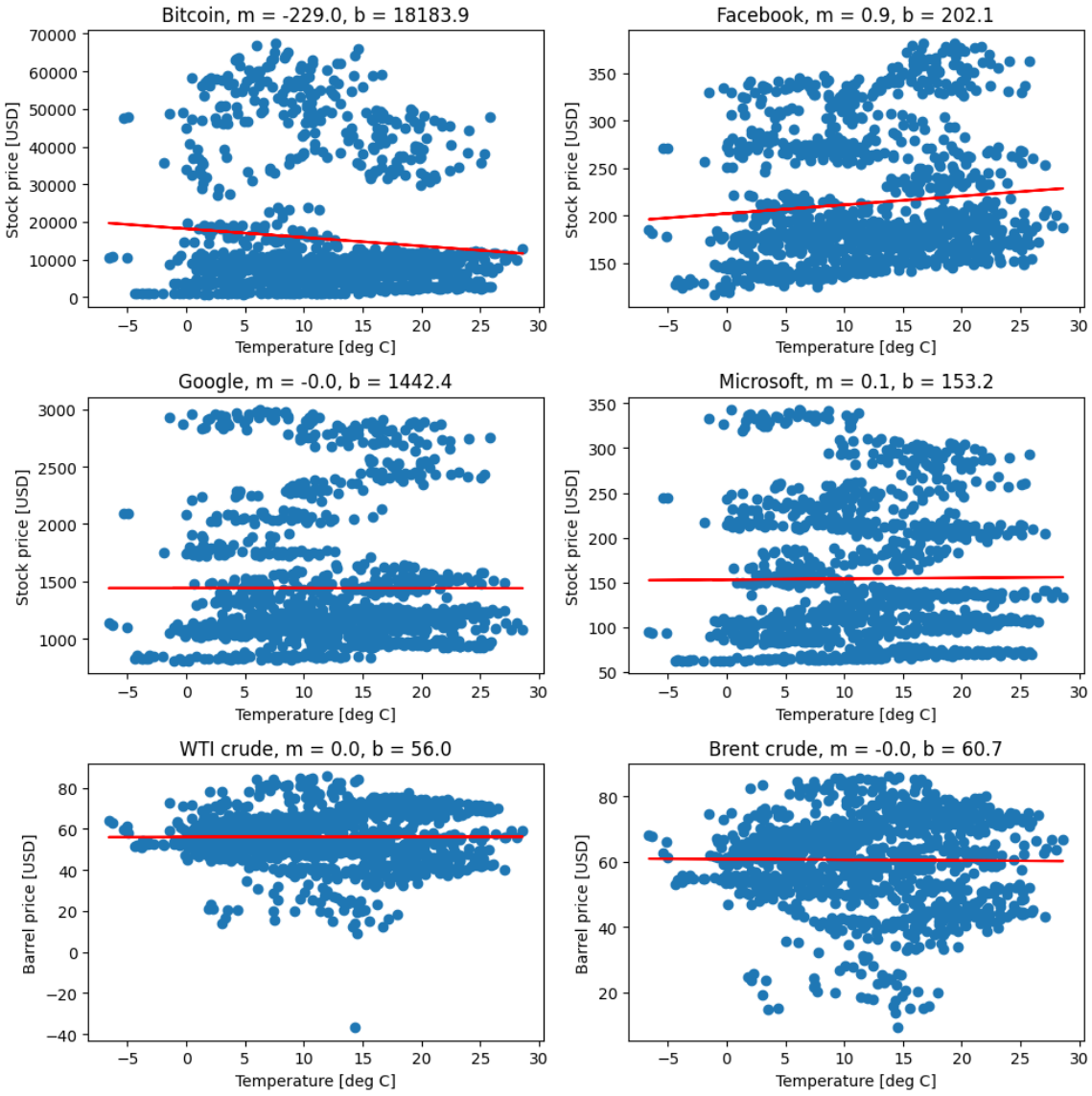
3. Regression

a), b) Calculate regression line

In [ ]:
```
#for simplicities sake I choose to continue working with the cleaned data vector
def regression(x, y):
    ones_vector = np.ones(len(x))
    X = np.c_[ones_vector, x]
    X = np.asmatrix(X)
    Y = np.asmatrix(y).T
    beta = np.linalg.inv(X.T*X)*(X.T*Y)
    return np.asarray(X*beta).flatten(), np.asarray(beta).flatten(), np.asarray(
```

In [ ]:
```
plt.figure(figsize=(10,10))

plt.subplot(3,2,1)
Y_hat, beta, error = regression(basel_c, btc_c)
plt.plot(basel_c, Y_hat, 'r-')
```

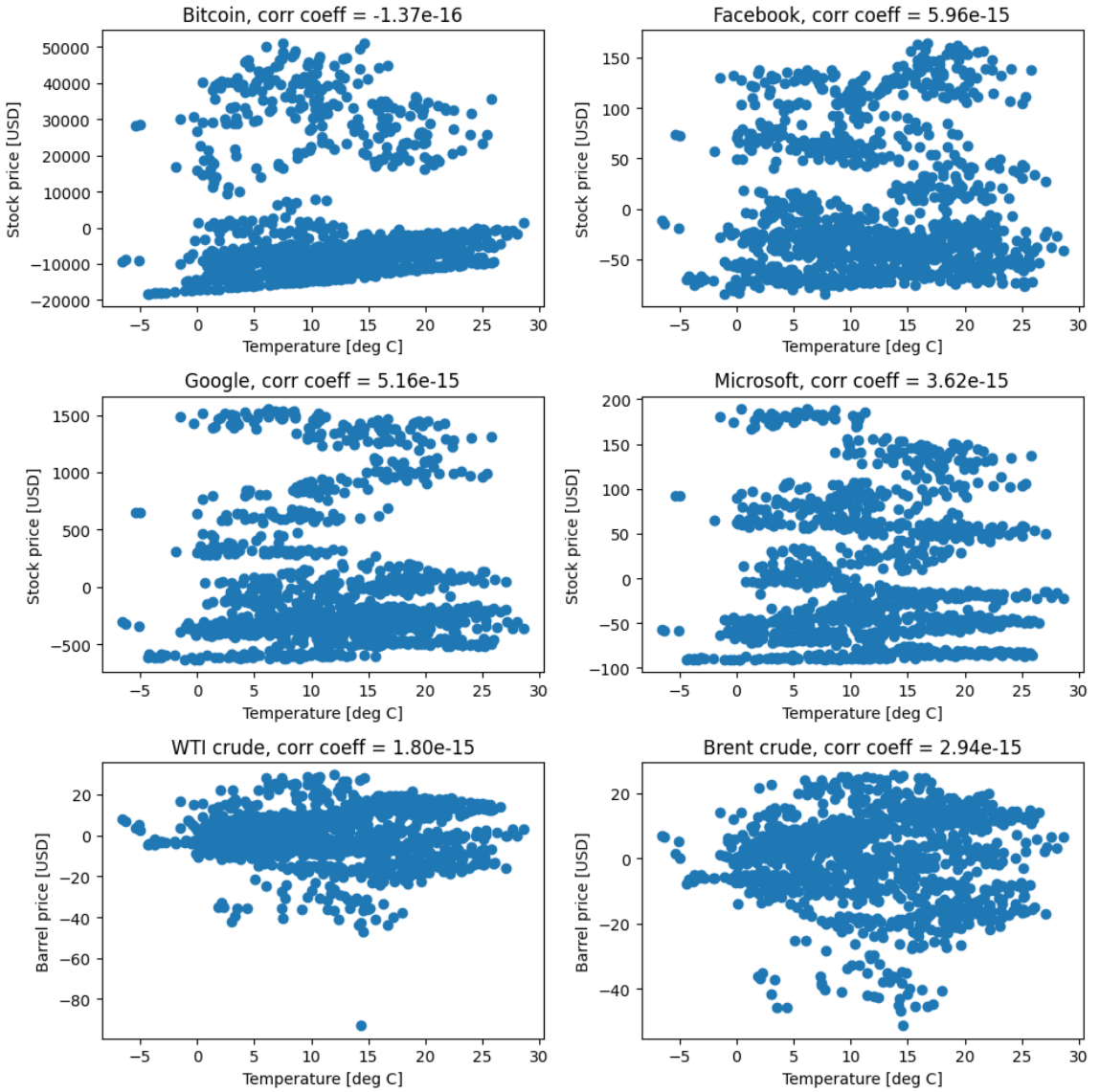
```
plt.scatter(basel_c,btc_c)
plt.title(f'Bitcoin, m = {beta[1]:.1f}, b = {beta[0]:.1f}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(3,2,2)
Y_hat, beta, error = regression(basel_c, face_c)
plt.plot(basel_c, Y_hat, 'r-')
plt.scatter(basel_c,face_c)
plt.title(f'Facebook, m = {beta[1]:.1f}, b = {beta[0]:.1f}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(3,2,3)
Y_hat, beta, error = regression(basel_c, google_c)
plt.plot(basel_c, Y_hat, 'r-')
plt.scatter(basel_c,google_c)
plt.title(f'Google, m = {beta[1]:.1f}, b = {beta[0]:.1f}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(3,2,4)
Y_hat, beta, error = regression(basel_c, micro_c)
plt.plot(basel_c, Y_hat, 'r-')
plt.scatter(basel_c,micro_c)
plt.title(f'Microsoft, m = {beta[1]:.1f}, b = {beta[0]:.1f}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(3,2,5)
Y_hat, beta, error = regression(basel_c, wti_c)
plt.plot(basel_c, Y_hat, 'r-')
plt.scatter(basel_c,wti_c)
plt.title(f'WTI crude, m = {beta[1]:.1f}, b = {beta[0]:.1f}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.subplot(3,2,6)
Y_hat, beta, error = regression(basel_c, brent_c)
plt.plot(basel_c, Y_hat, 'r-')
plt.scatter(basel_c,brent_c)
plt.title(f'Brent crude, m = {beta[1]:.1f}, b = {beta[0]:.1f}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.tight_layout()
```



c) Error / residual

In [ ]:
```
plt.figure(figsize=(10,10))

plt.subplot(3,2,1)
Y_hat, beta, error = regression(basel_c, btc_c)
plt.scatter(basel_c, error)
plt.title(f'Bitcoin, corr coeff = {np.corrcoef(basel_c, error)[0,1]:.2e}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(3,2,2)
Y_hat, beta, error = regression(basel_c, face_c)
plt.scatter(basel_c, error)
plt.title(f'Facebook, corr coeff = {np.corrcoef(basel_c, error)[0,1]:.2e}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(3,2,3)
Y_hat, beta, error = regression(basel_c, google_c)
plt.scatter(basel_c, error)
plt.title(f'Google, corr coeff = {np.corrcoef(basel_c, error)[0,1]:.2e}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')
```

```
plt.subplot(3,2,4)
Y_hat, beta, error = regression(basel_c, micro_c)
plt.scatter(basel_c, error)
plt.title(f'Microsoft, corr coeff = {np.corrcoef(basel_c, error)[0,1]:.2e}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Stock price [USD]')

plt.subplot(3,2,5)
Y_hat, beta, error = regression(basel_c, wti_c)
plt.scatter(basel_c, error)
plt.title(f'WTI crude, corr coeff = {np.corrcoef(basel_c, error)[0,1]:.2e}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.subplot(3,2,6)
Y_hat, beta, error = regression(basel_c, brent_c)
plt.scatter(basel_c, error)
plt.title(f'Brent crude, corr coeff = {np.corrcoef(basel_c, error)[0,1]:.2e}')
plt.xlabel('Temperature [deg C]')
plt.ylabel('Barrel price [USD]')

plt.tight_layout()
```



In [ ]:
```
# Investigating closeness of distributions to normal distributions
```

In [ ]:
```
Y_hat, beta, error = regression(basel_c, btc_c)
print(f'Bitcoin prediction error mean = {np.mean(error):.2e}, var = {np.var(erro

Y_hat, beta, error = regression(basel_c, face_c)
print(f'Facebook prediction error mean = {np.mean(error):.2e}, var = {np.var(err

Y_hat, beta, error = regression(basel_c, google_c)
print(f'Google  prediction error mean = {np.mean(error):.2e}, var = {np.var(erro

Y_hat, beta, error = regression(basel_c, micro_c)
print(f'Microsoft prediction error mean = {np.mean(error):.2e}, var = {np.var(er

Y_hat, beta, error = regression(basel_c, wti_c)
print(f'WTI crude prediction error mean = {np.mean(error):.2e}, var = {np.var(er

Y_hat, beta, error = regression(basel_c, brent_c)
print(f'Brent crude  prediction error mean = {np.mean(error):.2e}, var = {np.var
```

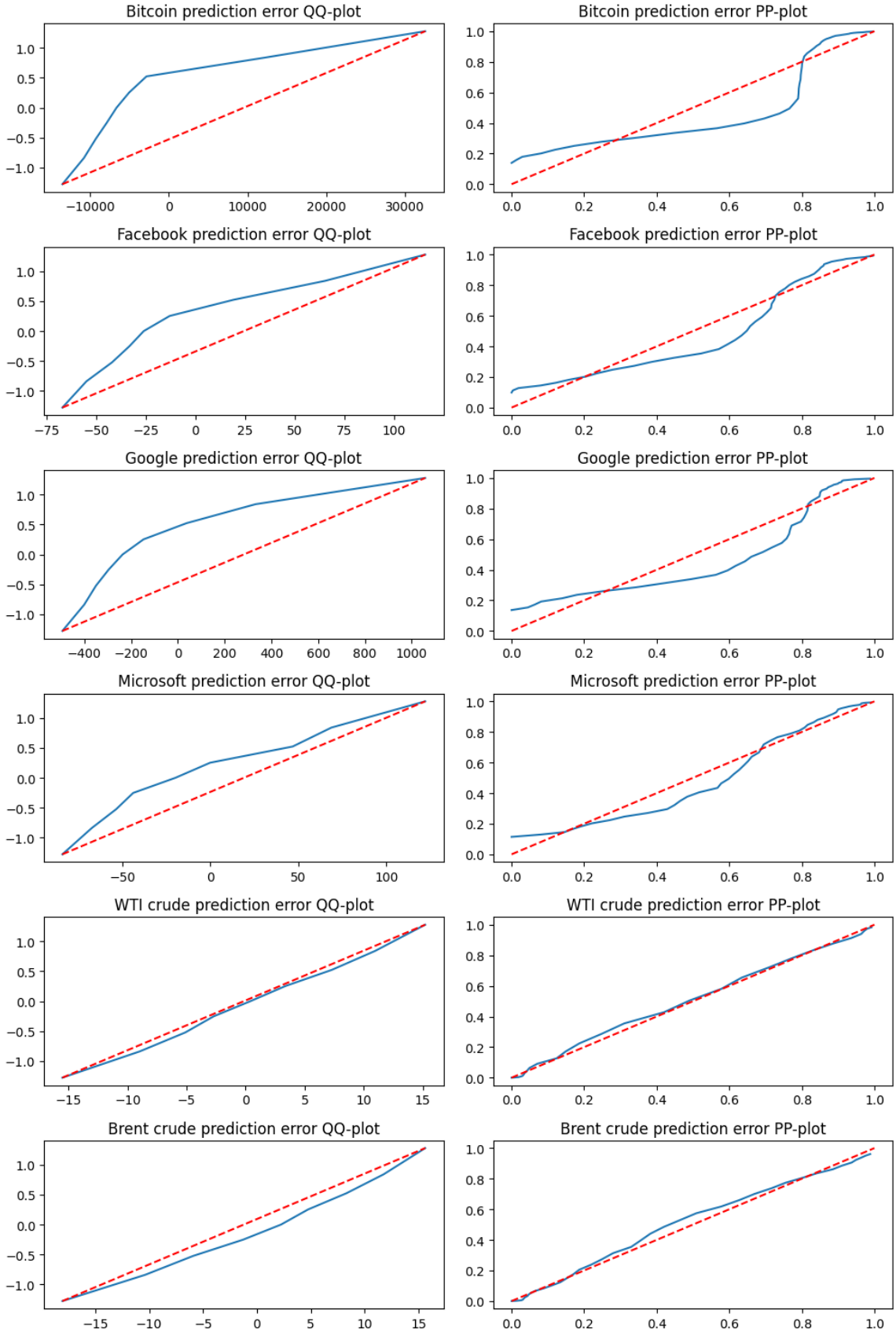
```
Bitcoin prediction error mean = -6.01e-12, var = 2.84e+08
Facebook prediction error mean = -1.91e-13, var = 4.26e+03
Google  prediction error mean = -1.50e-12, var = 3.35e+05
Microsoft prediction error mean = -1.70e-13, var = 5.70e+03
WTI crude prediction error mean = -8.87e-14, var = 1.63e+02
Brent crude  prediction error mean = -1.25e-13, var = 1.87e+02
```

In [ ]:
```
def qq_pp(y):
    range_qq = np.arange(0.1,1,0.1)
    range_pp = np.arange(np.min(y), np.max(y),(np.max(y)- np.min(y))/50)

    Norm_qq = st.norm.ppf(range_qq,loc=0,scale=1)
    Norm_pp = st.norm.cdf(range_pp,loc=0,scale= np.sqrt(np.var(y)))

    n = len(y)
    y_sort = np.sort(y)
    F = (np.arange(n))/(n+1)
    y_qq = np.interp(range_qq , F , y_sort)
    y_pp = np.interp(range_pp , y_sort , F)
    return Norm_qq, Norm_pp, y_qq, y_pp
```

For the below code I thought of a way to compress the iterations into a for loop to avoid copy pasting and manually changing stuff as much. I did not retroactively update code above to this format.

In [ ]:
```
plt.figure(figsize=(10,15))

data = np.c_[btc_c, face_c, google_c, micro_c, wti_c, brent_c]
titles = ('Bitcoin', 'Facebook', 'Google', 'Microsoft', 'WTI crude', 'Brent cruc
labels = ('Stock', 'Stock', 'Stock', 'Stock', 'Barrel', 'Barrel')

for i in range(len(data[0])):
    Y_hat, beta, error = regression(basel_c, data[:,i])
    Norm_qq, Norm_pp, y_qq, y_pp = qq_pp(error)
    plt.subplot(6,2,2*i+1)
    plt.plot(y_qq, Norm_qq)
    plt.plot([np.min(y_qq),np.max(y_qq)], [np.min(Norm_qq),np.max(Norm_qq)],'--r
    plt.title(f'{titles[i]} prediction error QQ-plot')

    plt.subplot(6,2,2*i+2)
```

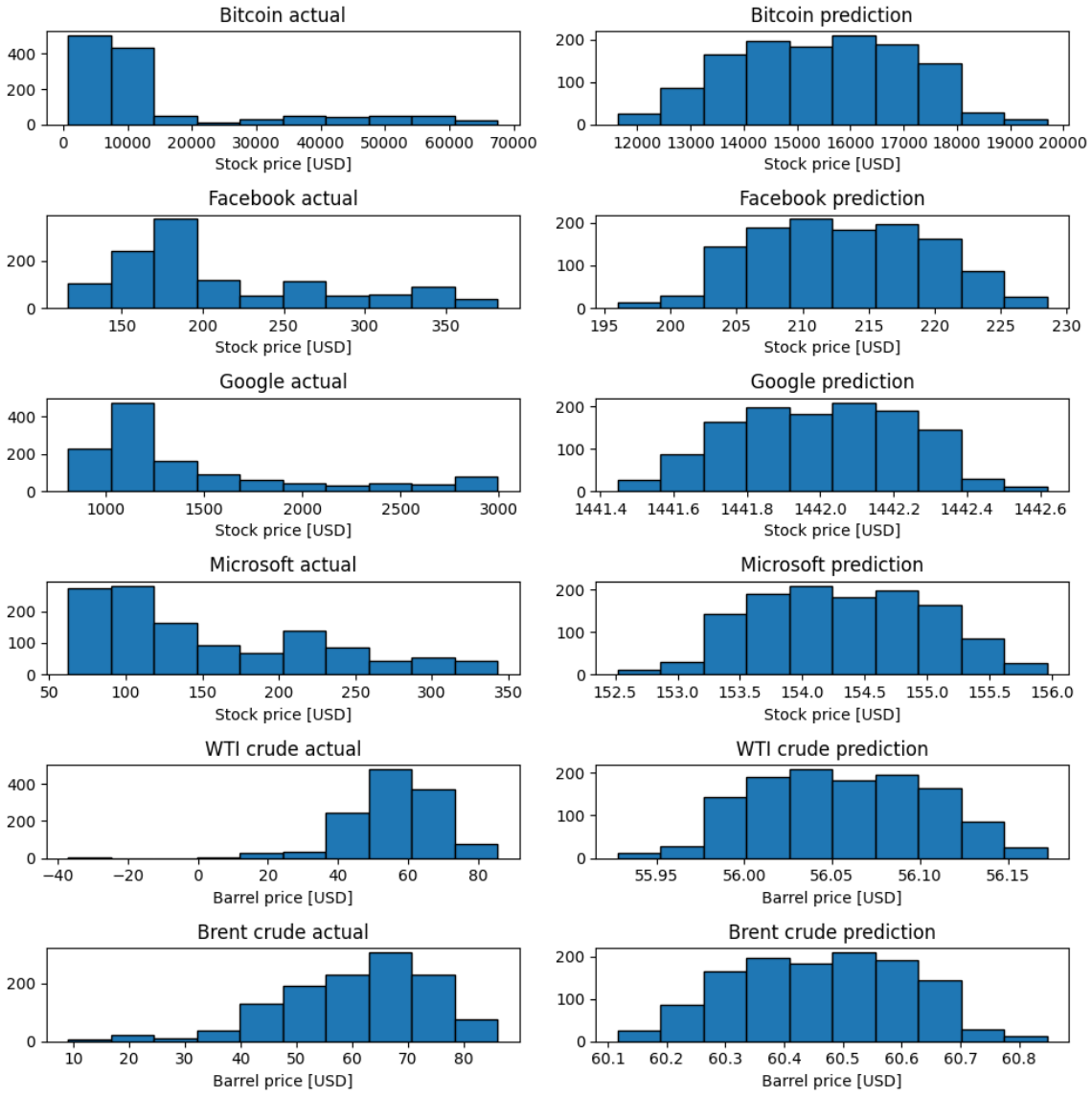
```
    plt.plot(y_pp, Norm_pp)
    plt.plot([0,1], [0,1], 'r--')
    plt.title(f'{titles[i]} prediction error PP-plot')

plt.tight_layout()
```



The predictions are obviously useless when we are trying to make them based on data that has no connection at all.

d) Histograms

In [ ]:
```
plt.figure(figsize=(10,10))

data = np.c_[btc_c, face_c, google_c, micro_c, wti_c, brent_c]
titles = ('Bitcoin', 'Facebook', 'Google', 'Microsoft', 'WTI crude', 'Brent cruc
labels = ('Stock', 'Stock', 'Stock', 'Stock', 'Barrel', 'Barrel')

for i in range(len(data[0])):
    plt.subplot(6,2,2*i+1)
    plt.hist(data[:,i], edgecolor='k')
    plt.title(f'{titles[i]} actual')
    plt.xlabel(f'{labels[i]} price [USD]')

    plt.subplot(6,2,2*i+2)
    Y_hat, beta, error = regression(basel_c, data[:,i])
    plt.hist(Y_hat, edgecolor='k')
    plt.title(f'{titles[i]} prediction')
    plt.xlabel(f'{labels[i]} price [USD]')

plt.tight_layout()
```



Mean and variance of actual and prediction

In [ ]:
```
data = np.c_[btc_c, face_c, google_c, micro_c, wti_c, brent_c]
titles = ('Bitcoin', 'Facebook', 'Google', 'Microsoft', 'WTI crude', 'Brent cruc
labels = ('Stock', 'Stock', 'Stock', 'Stock', 'Barrel', 'Barrel')

for i in range(len(data[0])):
    Y_hat, beta, error = regression(basel_c, data[:,i])
    print(f'{titles[i]} actual---- mean = {np.mean(data[:,i]):.2e}, var = {np.va
    print(f'{titles[i]} prediction mean = {np.mean(Y_hat):.2e}, var = {np.var(Y_
```

```
Bitcoin actual---- mean = 1.55e+04, var = 2.87e+08
Bitcoin prediction mean = 1.55e+04, var = 2.55e+06
Facebook actual---- mean = 2.13e+02, var = 4.30e+03
Facebook prediction mean = 2.13e+02, var = 4.14e+01
Google actual---- mean = 1.44e+03, var = 3.35e+05
Google prediction mean = 1.44e+03, var = 5.37e-02
Microsoft actual---- mean = 1.54e+02, var = 5.70e+03
Microsoft prediction mean = 1.54e+02, var = 4.64e-01
WTI crude actual---- mean = 5.61e+01, var = 1.63e+02
WTI crude prediction mean = 5.61e+01, var = 2.34e-03
Brent crude actual---- mean = 6.05e+01, var = 1.87e+02
Brent crude prediction mean = 6.05e+01, var = 2.10e-02
```

All results are very similar. It would be easier to say something more intelligent if the question was slightly more focused.