Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import pandas_datareader.data as web
import datetime
import matplotlib.pyplot as plt
%matplotlib inline
```

Getting Data

In [2]:
```
start = datetime.date(1982,4,20)
end = datetime.date(2021,12,31)

data = web.DataReader(name='^GSPC',data_source='yahoo',start=start,end=end)
data.drop(columns=['Volume','Adj Close'], inplace=True)
```

In [3]:
```
#creating raw Excel file with data

#data.to_excel(excel_writer='S&P500_Daily.xlsx',sheet_name='Data')
```

In [4]:
```
#check for NaN values
if data.shape == data.dropna().shape:
    print('No empty cells \n')
else:
    num_drop = data.shape[0]-data.dropna().shape[0]
    print('{} rows dropped due to empty cells \n'.format(num_drop))

print('Database data types:\n{}'.format(data.dtypes))
data.head()
```

```
No empty cells

Database data types:
High     float64
Low      float64
Open     float64
Close    float64
dtype: object
```

Out[4]:

Date	High	Low	Open	Close
1982-04-20	117.139999	114.830002	115.800003	115.440002
1982-04-21	115.870003	115.300003	115.480003	115.720001
1982-04-22	117.250000	115.720001	115.720001	117.190002
1982-04-23	118.639999	117.190002	118.019997	118.639999
1982-04-26	119.330002	118.250000	118.940002	119.260002

Returns Calculations

In [5]:
```
data['Daily_Change'] = ((data['Close']-data['Open'])/data['Open'])*100
data['True_Range'] = ((data['High']-data['Low'])/data['Low'])*100
data.head()
```

Out[5]:

Date	High	Low	Open	Close	Daily_Change	True_Range
1982-04-20	117.139999	114.830002	115.800003	115.440002	-0.310881	2.011667
1982-04-21	115.870003	115.300003	115.480003	115.720001	0.207826	0.494362
1982-04-22	117.250000	115.720001	115.720001	117.190002	1.270309	1.322156
1982-04-23	118.639999	117.190002	118.019997	118.639999	0.525337	1.237304
1982-04-26	119.330002	118.250000	118.940002	119.260002	0.269043	0.913321

Statistical calculations

In [6]:

```
descriptive_statistics = data[['Daily_Change','True_Range']].describe().round(2)
descriptive_statistics
```

Out[6]:

	Daily_Change	True_Range
count	10012.00	10012.00
mean	0.04	1.24
std	1.07	0.96
min	-20.47	0.00
25%	-0.42	0.67
50%	0.06	1.00
75%	0.54	1.50
max	10.79	25.74

In [7]:

```
calculations1 = {
    'average_returns':np.average(data.Daily_Change),
    'average_positive_return':data.Daily_Change[data.Daily_Change > 0].agg(func='mean'),
    'average_negative_return':data.Daily_Change[data.Daily_Change < 0].agg(func='mean'),
    'positive_change':(data.Daily_Change[data.Daily_Change > 0].agg(func='count')/len(data
    'negative_change':(data.Daily_Change[data.Daily_Change < 0].agg(func='count')/len(data
    'no_change':(data.Daily_Change[data.Daily_Change == 0].agg(func='count')/len(data.Dail

calculations2 = {
    'adjusted_poitive_return':(calculations1['average_positive_return']*calculations1['pos
    'adjusted_negative_return':(calculations1['average_negative_return']*calculations1['ne
    '1_std_dev_upper':descriptive_statistics.Daily_Change.loc['mean']+(1*descriptive_stati
    '1_std_dev_lower':descriptive_statistics.Daily_Change.loc['mean']-(1*descriptive_stati
    '2_std_dev_upper':descriptive_statistics.Daily_Change.loc['mean']+(2*descriptive_stati
    '2_std_dev_lower':descriptive_statistics.Daily_Change.loc['mean']-(2*descriptive_stati
    '3_std_dev_upper':descriptive_statistics.Daily_Change.loc['mean']+(3*descriptive_stati
    '3_std_dev_lower':descriptive_statistics.Daily_Change.loc['mean']-(3*descriptive_stati
    '1_std_dev_normal_dist':68.20,
    '2_std_dev_normal_dist':95.40,
    '3_std_dev_normal_dist':99.80}

calculations3 = {
    '1_std_dev_actual_dist':(len(data[data.Daily_Change.ge(calculations2['1_std_dev_lower'
            data.Daily_Change.le(calculations2['1_std_dev_upper'])])/descriptive_statistic
    '2_std_dev_actual_dist':(len(data[data.Daily_Change.ge(calculations2['2_std_dev_lower'
            data.Daily_Change.le(calculations2['2_std_dev_upper'])])/descriptive_statistic
    '3_std_dev_actual_dist':(len(data[data.Daily_Change.ge(calculations2['3_std_dev_lower'
            data.Daily_Change.le(calculations2['3_std_dev_upper'])])/descriptive_statistic
```

```
# concatinating all calculations into a dataframe

calc_df = pd.concat([
    pd.DataFrame(calculations1,index=[0]).transpose(),
    pd.DataFrame(calculations2,index=[0]).transpose(),
    pd.DataFrame(calculations3,index=[0]).transpose()])
calc_df.rename(columns={0:'Values (in %)'},inplace=True)
calc_df['Values (in %)'] = calc_df['Values (in %)'].round(3)
```

In [8]:
```
calc_df
```

Out[8]:

	Values (in %)
average_returns	0.038
average_positive_return	0.701
average_negative_return	-0.729
positive_change	53.546
negative_change	46.314
no_change	0.140
adjusted_poitive_return	0.375
adjusted_negative_return	-0.338
1_std_dev_upper	1.110
1_std_dev_lower	-1.030
2_std_dev_upper	2.180
2_std_dev_lower	-2.100
3_std_dev_upper	3.250
3_std_dev_lower	-3.170
1_std_dev_normal_dist	68.200
2_std_dev_normal_dist	95.400
3_std_dev_normal_dist	99.800
1_std_dev_actual_dist	79.405
2_std_dev_actual_dist	95.076
3_std_dev_actual_dist	98.532

Grouping data into bins and calculating frequencies

In [9]:
```
data['Daily Change Bins'] = pd.cut(data.Daily_Change, bins=np.linspace(-4,4,33))
data['True Range Bins'] = pd.cut(data.True_Range, bins=np.linspace(0,4,33))

#daily change frequencies
freq1 = pd.DataFrame(data['Daily Change Bins'].value_counts().sort_index())
freq1.reset_index(inplace=True)
freq1.rename(columns={'index':'Bins_DC','Daily Change Bins':'Frequency_DC'},inplace=True)

freq1_sum = np.sum(freq1.Frequency_DC)
freq1['Probabilities_DC'] = (freq1.Frequency_DC/freq1_sum)*100
freq1['Cumulative_DC'] = freq1.Probabilities_DC.cumsum()
```

```
#true range frequencies
freq2 = pd.DataFrame(data['True Range Bins'].value_counts().sort_index())
freq2.reset_index(inplace=True)
freq2.rename(columns={'index':'Bins_TR','True Range Bins':'Frequency_TR'},inplace=True)

freq2_sum = np.sum(freq2.Frequency_TR)
freq2['Probabilities_TR'] = (freq2.Frequency_TR/freq2_sum)*100
freq2['Cumulative_TR'] = freq2.Probabilities_TR.cumsum()

#concatinating 2 frequencies
freq = pd.concat([freq1,freq2],axis=1)
```

In [10]:
```
freq
```

Out[10]:

	Bins_DC	Frequency_DC	Probabilities_DC	Cumulative_DC	Bins_TR	Frequency_TR	Probabilities_TR	Cumulative_T
0	(-4.0, -3.75]	11	0.110709	0.110709	(0.0, 0.125]	0	0.000000	0.00000
1	(-3.75, -3.5]	9	0.090580	0.201288	(0.125, 0.25]	40	0.406339	0.40633
2	(-3.5, -3.25]	9	0.090580	0.291868	(0.25, 0.375]	347	3.524990	3.93132
3	(-3.25, -3.0]	23	0.231481	0.523349	(0.375, 0.5]	795	8.075985	12.00731
4	(-3.0, -2.75]	23	0.231481	0.754831	(0.5, 0.625]	973	9.884193	21.89150
5	(-2.75, -2.5]	43	0.432770	1.187601	(0.625, 0.75]	1004	10.199106	32.09061
6	(-2.5, -2.25]	58	0.583736	1.771337	(0.75, 0.875]	974	9.894352	41.98496
7	(-2.25, -2.0]	74	0.744767	2.516103	(0.875, 1.0]	901	9.152783	51.13774
8	(-2.0, -1.75]	119	1.197665	3.713768	(1.0, 1.125]	786	7.984559	59.12230
9	(-1.75, -1.5]	182	1.831723	5.545491	(1.125, 1.25]	665	6.755384	65.87769
10	(-1.5, -1.25]	213	2.143720	7.689211	(1.25, 1.375]	566	5.749695	71.62738
11	(-1.25, -1.0]	305	3.069646	10.758857	(1.375, 1.5]	454	4.611946	76.23933
12	(-1.0, -0.75]	479	4.820853	15.579710	(1.5, 1.625]	387	3.931329	80.17066
13	(-0.75, -0.5]	672	6.763285	22.342995	(1.625, 1.75]	320	3.250711	83.42137
14	(-0.5, -0.25]	969	9.752415	32.095411	(1.75, 1.875]	274	2.783421	86.20479
15	(-0.25, 0.0]	1424	14.331723	46.427134	(1.875, 2.0]	204	2.072328	88.27712
16	(0.0, 0.25]	1545	15.549517	61.976651	(2.0, 2.125]	206	2.092645	90.36976

	Bins_DC	Frequency_DC	Probabilities_DC	Cumulative_DC	Bins_TR	Frequency_TR	Probabilities_TR	Cumulative_T
17	(0.25, 0.5]	1181	11.886071	73.862721	(2.125, 2.25]	160	1.625356	91.99512
18	(0.5, 0.75]	843	8.484300	82.347021	(2.25, 2.375]	142	1.442503	93.43762
19	(0.75, 1.0]	583	5.867552	88.214573	(2.375, 2.5]	100	1.015847	94.45347
20	(1.0, 1.25]	404	4.066023	92.280596	(2.5, 2.625]	98	0.995530	95.44900
21	(1.25, 1.5]	242	2.435588	94.716184	(2.625, 2.75]	72	0.731410	96.18041
22	(1.5, 1.75]	175	1.761272	96.477456	(2.75, 2.875]	69	0.700935	96.88134
23	(1.75, 2.0]	109	1.097021	97.574477	(2.875, 3.0]	62	0.629825	97.51117
24	(2.0, 2.25]	68	0.684380	98.258857	(3.0, 3.125]	35	0.355547	97.86672
25	(2.25, 2.5]	64	0.644122	98.902979	(3.125, 3.25]	48	0.487607	98.35432
26	(2.5, 2.75]	34	0.342190	99.245169	(3.25, 3.375]	35	0.355547	98.70987
27	(2.75, 3.0]	23	0.231481	99.476651	(3.375, 3.5]	33	0.335230	99.04510
28	(3.0, 3.25]	12	0.120773	99.597424	(3.5, 3.625]	25	0.253962	99.29906
29	(3.25, 3.5]	19	0.191224	99.788647	(3.625, 3.75]	28	0.284437	99.58350
30	(3.5, 3.75]	11	0.110709	99.899356	(3.75, 3.875]	21	0.213328	99.79683
31	(3.75, 4.0]	10	0.100644	100.000000	(3.875, 4.0]	20	0.203169	100.00000

Plotting Returns

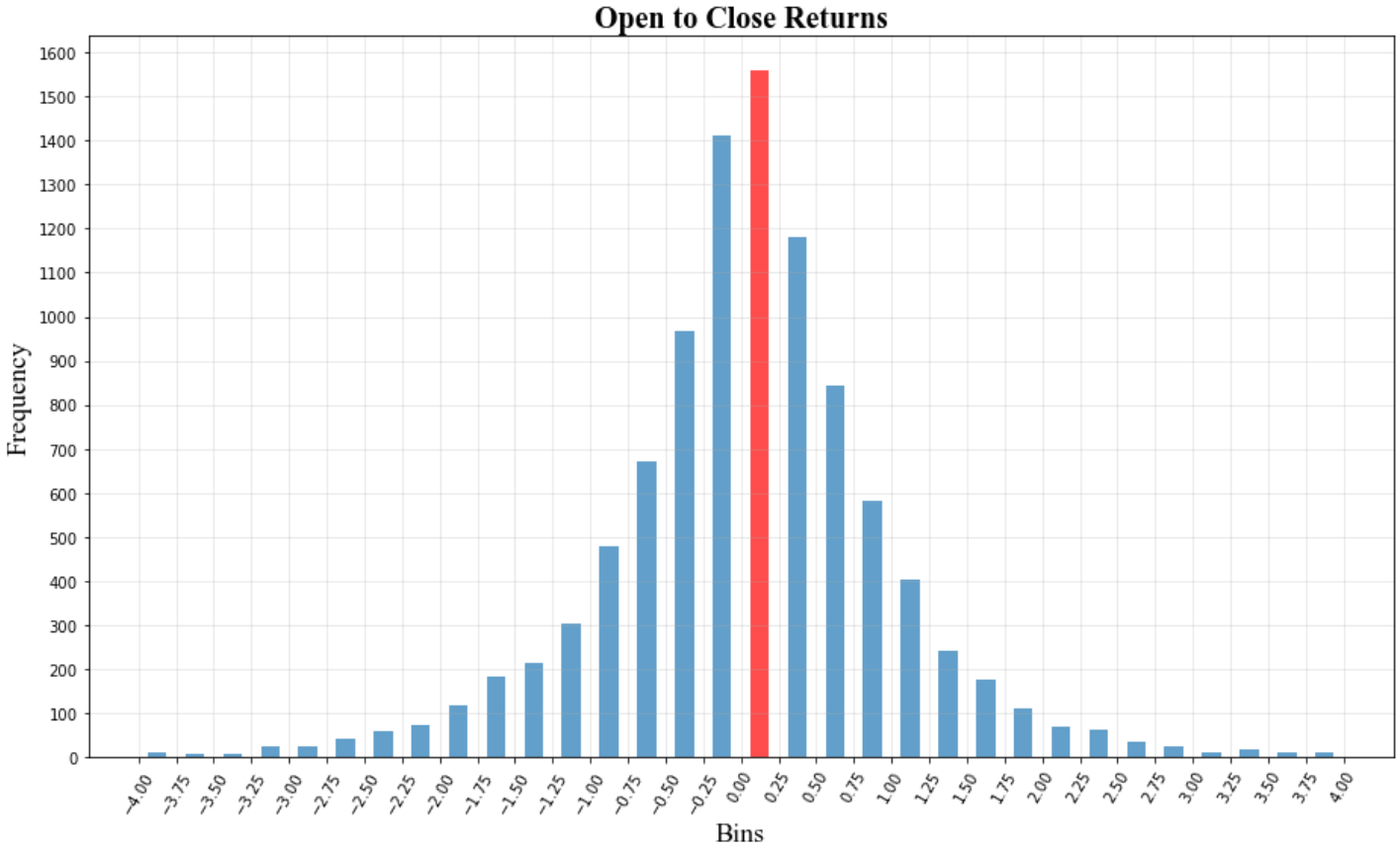
In [11]:

```
#returns plot function

def returns_dc():
    fig = plt.figure(figsize=(13,8))
    n, bins, patches = plt.hist(data.Daily_Change,bins=np.linspace(-4,4,33),rwidth=0.5,alp
    patches[16].set_fc('red')

    plt.xticks(np.linspace(-4,4,33))
    plt.yticks(range(0,1700,100))
    x = plt.gca().xaxis
    for item in x.get_ticklabels():
        item.set_rotation(60)

    plt.title('Open to Close Returns',fontdict={'size':20,'weight':'bold','family':'Times
    plt.xlabel('Bins',fontdict={'size':18,'family':'Times New Roman'})
    plt.ylabel('Frequency',fontdict={'size':18,'family':'Times New Roman'})
    plt.grid(alpha=0.3)
```

```
    plt.tight_layout()

def returns_tr():
    fig = plt.figure(figsize=(13,8))
    n, bins, patches = plt.hist(data.True_Range,bins=np.linspace(0,4,33),rwidth=0.5,alpha=
    patches[5].set_fc('red')

    plt.xticks(np.linspace(0,4,33))
    plt.yticks(range(0,1100,100))
    x = plt.gca().xaxis
    for item in x.get_ticklabels():
        item.set_rotation(60)

    plt.title('High to Low Returns',fontdict={'size':20,'weight':'bold','family':'Times Ne
    plt.xlabel('Bins',fontdict={'size':18,'family':'Times New Roman'})
    plt.ylabel('Frequency',fontdict={'size':18,'family':'Times New Roman'})
    plt.grid(alpha=0.3)

    plt.tight_layout()
```

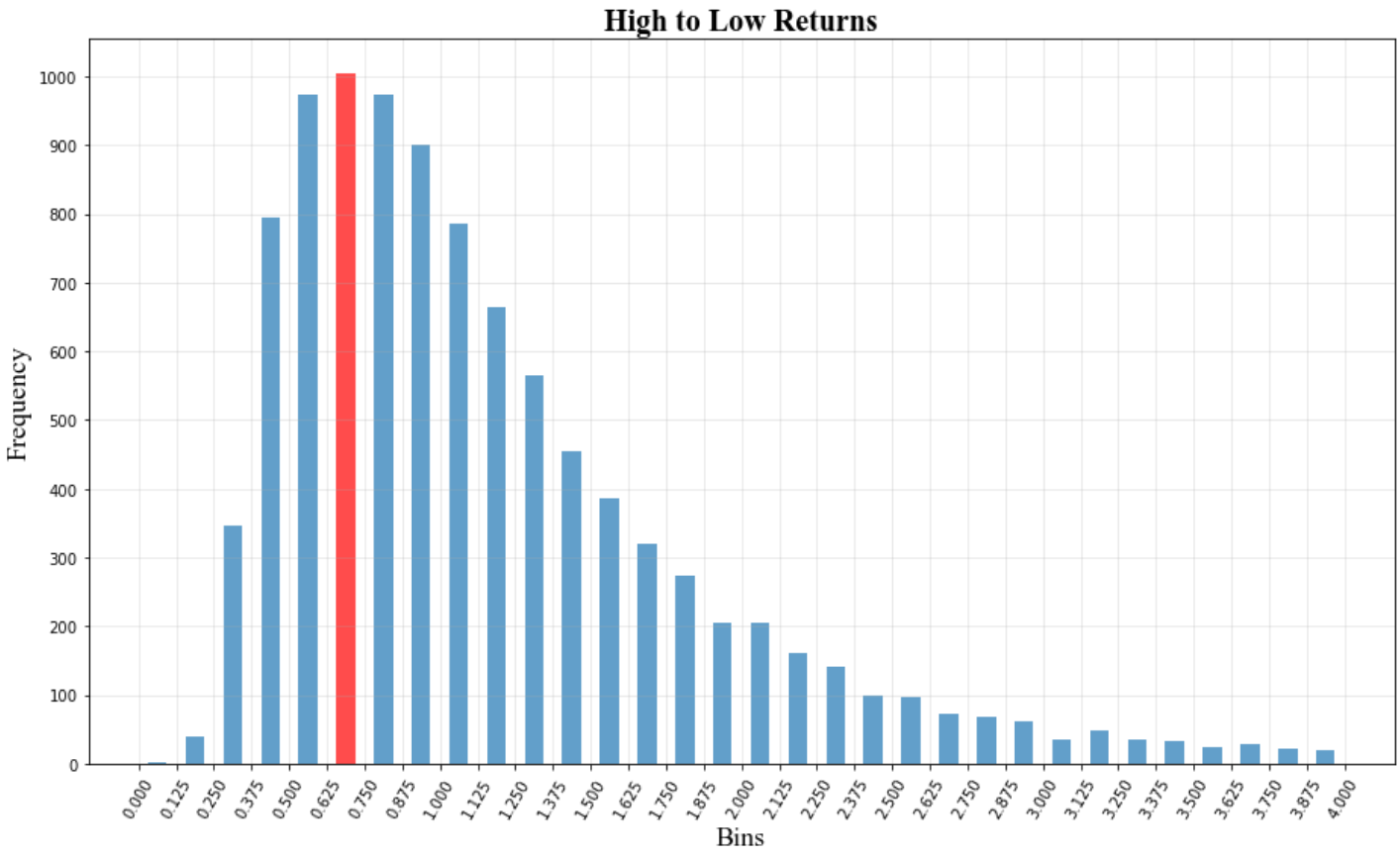
In [12]:

```
returns_dc()
```



In [13]:

```
returns_tr()
```


High to Low Returns
Frequency
Bins
0
100
200
300
400
500
600
700
800
900
1000
0.000
0.125
0.250
0.375
0.500
0.625
0.750
0.875
1.000
1.125
1.250
1.375
1.500
1.625
1.750
1.875
2.000
2.125
2.250
2.375
2.500
2.625
2.750
2.875
3.000
3.125
3.250
3.375
3.500
3.625
3.750
3.875
4.000