Zhuo Liuzhou, 30Jul2021

1.1 Review the quality of the data, list any potential errors, and propose corrected values. Please list each quality check error and correction applied.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import datetime
import holidays
import yfinance as yf
pd.options.mode.chained_assignment = None
```

In [2]:

```
sample = pd.read_excel('Sample Dataset.xlsx')
sample.index = range(1,len(sample)+1)
sample.head()
```

Out[2]:

	Date	Signal	Open	High	Low	Close	Adj Close
1	2015-11-19	13.768540	116.440002	116.650002	115.739998	116.059998	108.281601
2	2015-11-20	13.608819	116.480003	117.360001	116.379997	116.809998	108.981323
3	2015-11-23	12.990589	116.709999	117.889999	116.680000	117.389999	109.522453
4	2015-11-24	12.667435	116.879997	118.419998	116.559998	118.250000	110.324837
5	2015-11-25	13.019910	118.300003	119.320000	118.110001	119.169998	111.183159

In [3]:

```
sample.describe()
```

Out[3]:

	Signal	Open	High	Low	Close	Adj Close
count	1038.000000	1038.000000	1038.000000	1038.000000	1038.000000	1038.000000
mean	16.766190	141.847360	142.691801	140.907746	141.840973	136.341060
std	3.095783	18.475574	18.470255	18.404504	18.497010	21.427837
min	0.000000	94.080002	95.400002	93.639999	94.790001	-152.277847
25%	14.691150	132.132496	132.912495	130.542503	131.824993	125.290491
50%	17.298240	146.769997	147.959999	145.634995	146.885002	142.667732
75%	19.030890	155.367496	156.287495	154.422500	155.289993	151.798325
max	35.434147	172.789993	173.389999	171.949997	196.279999	168.842270

one can easily find that **"Adj Close" contain negative value**, and **max of "Close" is larger than max of "High"** both of which are impossible in real market, will address this error in below health checks.

I did below health checks:

(1). No missing signals or prices

(2). Dates are correct:

a. No duplicate dates

b. Dates are ordered

c. No weekends

d. No public holidays

e. No missing workdays

(3). All prices look right:

a. All prices are positive

b. “Low” < (“Open”, “Close”) < “High” for each day

c. “Adj Close” looks right

(4). Signal Outliers

And the errors and correction are listed below, noted that the correction will be done after the checks to consider interaction between errors.

(1) No Null signals or prices.

In [4]:

```
sample.isnull().any(axis = 0)
```

Out[4]:

```
Date          False
Signal        False
Open          False
High          False
Low           False
Close         False
Adj Close     False
dtype: bool
```

(2) Dates:

a. No duplicate dates

In [5]:

```
len(sample['Date'].unique()) == len(sample['Date'])
```

Out[5]:

```
True
```

b. Dates are in ascending order

In [6]:

```
dateDiff = (sample['Date'].diff())[1:]
all([d.days >0 for d in dateDiff])
```

Out[6]:

```
True
```

c. Weekends

In [7]:

```
weekends_index = []
weekends = []
```

```
for i in sample.index:
    if pd.Timestamp.weekday(sample.loc[i,'Date'])>4:
        weekends.append(sample.loc[i,'Date'])
        weekends_index.append(i)
weekends
```

Out[7]:

```
[Timestamp('2018-05-19 00:00:00'),
 Timestamp('2018-05-20 00:00:00'),
 Timestamp('2018-06-23 00:00:00'),
 Timestamp('2018-06-24 00:00:00')]
```

The data has 4 records on weekends

In [8]:

```
sample.loc[weekends_index[0]:weekends_index[1]+2]
```

Out[8]:

	Date	Signal	Open	High	Low	Close	Adj Close
630	2018-05-19	20.448445	162.369995	163.240005	162.360001	162.940002	157.493622
631	2018-05-20	19.483907	163.259995	163.330002	161.630005	161.759995	156.352997
632	2018-05-21	19.031457	162.369995	163.240005	162.360001	162.940002	157.493622
633	2018-05-22	19.823488	163.259995	163.330002	161.630005	161.759995	156.352997

In [9]:

```
sample.loc[weekends_index[2]:weekends_index[3]+2]
```

Out[9]:

	Date	Signal	Open	High	Low	Close	Adj Close
656	2018-06-23	18.995502	167.240005	167.369995	164.139999	165.080002	159.562042
657	2018-06-24	20.274163	165.229996	166.660004	164.850006	166.039993	160.489944
658	2018-06-25	21.123096	167.240005	167.369995	164.139999	165.080002	159.562042
659	2018-06-26	20.198530	165.229996	166.660004	164.850006	166.039993	160.489944

From above can see those prices are the **same as next two workdays, correction is to remove these signals and prices.**

d. Public holidays

In [10]:

```
pub_hol_index = []
pub_hol = []
us_holidays = pd.read_excel('US mkt holidays.xlsx')
us_holidays = us_holidays['Date'].tolist()
```

In [11]:

```
for i in sample.index:
    if sample.loc[i,'Date'] in us_holidays:
        pub_hol.append(sample.loc[i,'Date'])
        pub_hol_index.append(i)
print(pub_hol)
```

```
[Timestamp('2017-07-04 00:00:00')]
```

The data has 1 record on public holidays (US holiday)

In [12]:

```
sample.loc[pub_hol_index[0]-2:pub_hol_index[0]+2]
```

Out[12]:

	Date	Signal	Open	High	Low	Close	Adj Close
406	2017-06-30	16.482053	141.250000	141.669998	140.770004	140.919998	134.474945
407	2017-07-03	16.803540	141.339996	142.500000	141.300003	142.100006	135.600998
408	2017-07-04	15.282748	141.339996	142.600000	141.400003	142.200006	135.700998
409	2017-07-05	15.282748	141.699997	141.850006	140.699997	141.589996	135.114380
410	2017-07-06	15.811562	139.929993	140.470001	138.830002	139.139999	133.349945

The prices of this record are not exactly the same as its neighboring records but are very close to last record (all of High, Low, Close and Adj Close are just **+0.1 from last record**), so guess is autofill down from last record. **Correction is to remove this record.**

e. Missing working days

In [13]:

```
sample['Date'].max()
```

Out[13]:

```
Timestamp('2020-01-06 00:00:00')
```

In [14]:

```
start_date = sample['Date'].min()
mis_workday = []
while start_date < sample['Date'].max():
    if pd.Timestamp.weekday(start_date) <5 and start_date not in us_holidays:
        if start_date not in sample['Date'].tolist():
            mis_workday.append(start_date)
    start_date += datetime.timedelta(days=1)
mis_workday
```

Out[14]:

```
[Timestamp('2018-11-12 00:00:00'),
 Timestamp('2018-11-13 00:00:00'),
 Timestamp('2018-11-14 00:00:00'),
 Timestamp('2018-11-15 00:00:00'),
 Timestamp('2018-11-16 00:00:00')]
```

The dataset has 5 missing working days. Prices for dates between 12Nov2018 – 16Nov2018 (5 working days) is missing, correction is described in below section 1.1 correction

(3) Prices:

a. Negative prices

In [15]:

```
sample[(sample.loc[:,'Open':'Adj Close']<0).any(axis = 1)]
```

Out[15]:

	Date	Signal	Open	High	Low	Close	Adj Close
733	2018-10-10	19.719477	160.820007	160.990005	156.360001	156.559998	-152.277847

In [16]:

```
sample[(sample.loc[:,'Open':'Adj Close']<0).any(axis = 1)].index
```

Out[16]:

```
Int64Index([733], dtype='int64')
```

1 day has negative price: as pointed out before, the “Adj Close” contains one negative value, and it should be computer “typo”

b. “Low” < (“Open”, “Close”) < “High”

In [17]:
```
def price_correct_check(df):
    res = ''
    if df['Low']>df['High']:
        res+='1'
    if df['Close']>df['High']:
        res+='2'
    if df['Close']<df['Low']:
        res+='3'
    if df['Open']>df['High']:
        res+='4'
    if df['Open']<df['Low']:
        res+='5'
    return res
```

In [18]:
```
wrong_price = sample.apply(price_correct_check, axis = 1)
wrong_price = wrong_price[wrong_price.apply(len)>0]
```

In [19]:
```
print("Number of illegal records: {}".format(len(wrong_price)))
```

```
Number of illegal records: 20
```

In [20]:
```
sample.loc[wrong_price.index].head()
```

Out[20]:

	Date	Signal	Open	High	Low	Close	Adj Close
408	2017-07-04	15.282748	141.339996	142.600000	141.400003	142.200006	135.700998
432	2017-08-07	16.298805	140.440002	140.350000	139.710007	140.440002	134.595871
456	2017-09-11	15.838558	140.389999	140.919998	140.229996	139.110001	133.321198
457	2017-09-12	15.518587	141.039993	141.690002	140.820007	139.110001	133.321198
458	2017-09-13	16.158529	141.410004	142.220001	141.320007	139.110001	133.321198

In total there are 20 days with prices not fulfilling “Low” < (“Open”, “Close”) < “High” rule.
Correction is described in below correction section

c. Adj Close

The "Adj Close" if similar to Yahoo Finance, is just "Close" adjusted for dividends and stock split. So the ratio of "Close" to "Adj Close" should be persistent for a period of time.

And because in (3)-a I already detected a negative Adj Close records, so in this section I'll first correct that to positive value then examine.

In [21]:
```
sample['Adj/Close'] = np.abs(sample['Adj Close']/sample['Close'])
sample['Adj/Close']
```

Out[21]:
```
1        0.932980
```

```
2       0.932979
3       0.932979
4       0.932980
5       0.932979
          ...
1034    0.989021
1035    0.989021
1036    0.989021
1037    0.989021
1038    0.989021
Name: Adj/Close, Length: 1038, dtype: float64
```

In [22]:

```
count, division = np.histogram(sample['Adj/Close'])
list(zip(count,division))
```

Out[22]:

```
[(1, 0.7678226093734595),
 (0, 0.8158432673157043),
 (0, 0.8638639252579492),
 (464, 0.911884583200194),
 (571, 0.9599052411424389),
 (0, 1.0079258990846838),
 (0, 1.0559465570269286),
 (0, 1.1039672149691735),
 (1, 1.1519878729114184),
 (1, 1.2000085308536632)]
```

From the histogram count we can see there are three abnormal "Adj Close", and since the number is not large, I examined them one by one

No. 1:

In [23]:

```
sample.loc[sample[sample['Adj/Close']<0.8].index[0]-2:].head()
```

Out[23]:

	Date	Signal	Open	High	Low	Close	Adj Close	Adj/Close
584	2018-03-15	18.100298	158.100006	158.139999	156.369995	156.919998	151.325409	0.964348
585	2018-03-16	19.385186	156.979996	158.270004	156.750000	157.800003	152.174042	0.964348
586	2018-03-19	18.660897	157.169998	157.210007	154.449997	196.279999	150.708221	0.767823
587	2018-03-20	19.177721	156.669998	157.020004	155.770004	156.240005	150.669647	0.964347
588	2018-03-21	19.019439	156.320007	158.259995	156.199997	157.149994	151.547241	0.964348

In [24]:

```
sample[sample['Adj/Close']<0.8].index[0] in wrong_price.index
```

Out[24]:

```
True
```

Noted that this one is due to the illegal "Close" (larger than "High"), so after adjustment (set "Close" = "High") the Adj/CLose becomes:

In [25]:

```
sample.loc[586,'Adj Close']/sample.loc[586,'High']
```

Out[25]:

```
0.9586426708829039
```

which looks normal comparing to its neighborning records.

No. 2:

In [26]:
```
sample[sample[(sample['Adj/Close']<1.2) &\
              (sample['Adj/Close']>1)].index[0]-3:].head()
```

Out[26]:

	Date	Signal	Open	High	Low	Close	Adj Close	Adj/Close
337	2017-03-23	15.705048	134.000000	135.699997	133.639999	134.779999	128.246490	0.951525
338	2017-03-24	15.131940	134.949997	135.470001	133.860001	134.490005	128.339050	0.954265
339	2017-03-27	16.032241	132.759995	135.070007	132.399994	134.740005	158.577637	1.176916
340	2017-03-28	16.215369	134.279999	135.899994	134.139999	135.789993	129.579605	0.954265
341	2017-03-29	15.265496	135.699997	136.490005	135.300003	136.229996	129.999481	0.954265

In [27]:
```
sample[(sample['Adj/Close']<1.2) &\
       (sample['Adj/Close']>1)].index[0] in wrong_price.index
```

Out[27]: `False`

This one's prices look right, so is due to abnormal "Adj Close", and the records before and after it all have Adj Close/Close ratio of 0.954265, so this one should have the same ratio. **Correction is to adjust the "Adj Close" so that "Adj Close"/"Close" for 27Mar2017 = 0.954265**

No. 3:

In [28]:
```
sample[sample[sample['Adj/Close']>1.2].index[0]-3:].head()
```

Out[28]:

	Date	Signal	Open	High	Low	Close	Adj Close	Adj/Close
261	2016-12-01	15.304759	132.250000	132.550003	130.289993	130.970001	124.109283	0.947616
262	2016-12-02	14.990113	130.940002	131.470001	130.520004	130.899994	124.042938	0.947616
263	2016-12-05	16.011011	131.970001	133.330002	131.889999	133.149994	166.175079	1.248029
264	2016-12-06	15.885051	133.520004	134.910004	132.740005	134.589996	127.539658	0.947616
265	2016-12-07	15.406135	134.589996	136.179993	134.179993	135.899994	128.781036	0.947616

In [29]:
```
sample[sample['Adj/Close']>1.2].index[0] in wrong_price.index
```

Out[29]: `False`

This one's issue is same as No. 2, **correction is to adjust the "Adj Close" so that "Adj**

Close"/"Close" for 27Mar2017 = 0.947616

(4) Signals

a. Signals with value "0"

In [30]:
```
sample[sample['Signal']==0]
```

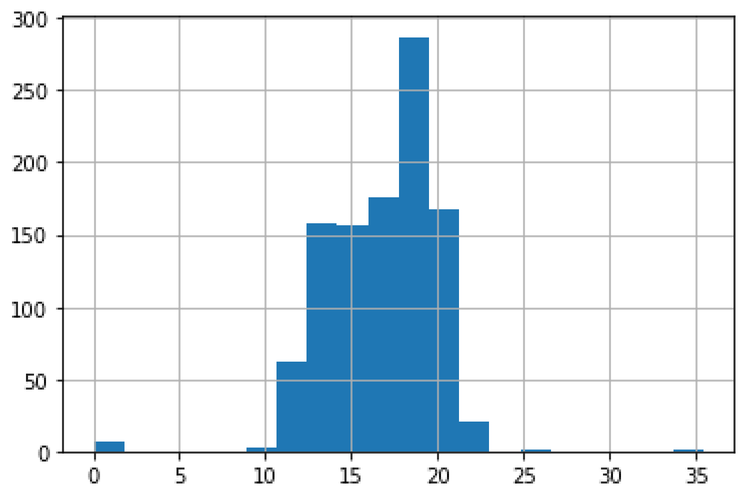
Out[30]:

	Date	Signal	Open	High	Low	Close	Adj Close	Adj/Close
1033	2019-12-27	0.0	167.119995	167.119995	165.429993	165.860001	164.039063	0.989021
1034	2019-12-30	0.0	165.979996	166.210007	164.570007	165.440002	163.623688	0.989021
1035	2019-12-31	0.0	165.080002	166.350006	164.710007	165.669998	163.851135	0.989021
1036	2020-01-02	0.0	166.740005	166.750000	164.229996	165.779999	163.959946	0.989021
1037	2020-01-03	0.0	163.740005	165.410004	163.699997	165.130005	163.317093	0.989021
1038	2020-01-06	0.0	163.850006	165.539993	163.539993	165.350006	163.534668	0.989021

b. Extremely large or small values

In [31]:
```
sample['Signal'].hist(bins=20)
```

Out[31]: <AxesSubplot:>



In [32]:
```
count, division = np.histogram(sample['Signal'])
list(zip(count,division))
```

Out[32]:
```
[(6, 0.0),
 (0, 3.5434147008000005),
 (2, 7.086829401600001),
 (220, 10.630244102400003),
 (332, 14.173658803200002),
```

```
 (455, 17.717073504000002),
 (21, 21.260488204800005),
 (1, 24.803902905600005),
 (0, 28.347317606400004),
 (1, 31.890732307200004)]
```

In [33]:

```
sample.loc[sample[sample['Signal']>35].index[0]-2:].head()
```

Out[33]:

	Date	Signal	Open	High	Low	Close	Adj Close	Adj/Close
499	2017-11-09	17.361475	146.270004	147.389999	145.279999	146.679993	140.916534	0.960707
500	2017-11-10	17.628384	146.710007	147.100006	146.350006	146.570007	140.810852	0.960707
501	2017-11-13	35.434147	145.929993	146.820007	145.500000	146.610001	140.849274	0.960707
502	2017-11-14	17.456319	146.059998	146.490005	145.589996	146.210007	140.465012	0.960707
503	2017-11-15	17.928089	145.350006	146.210007	144.500000	145.630005	139.907806	0.960707

The histogram of signal shows an outlier in the right, with value of 35.43, looking at this record, the prices are not abnormal at that day, so it is highly likely this signal is wrong, will **remove this record** when evaluating the efficacy.

1.1 Corrections

(2) c & d. Weekends and public holidays

In [34]:

```
newdata = sample.drop(index = weekends_index + pub_hol_index)
newdata.index = range(1,len(newdata)+1)
```

(2) e. Missing workdays 12Nov2018 - 16Nov2018

Correction to signals: set them to be zero
Correction to price: interpolation using 9Nov2018 and 19Nov2018 prices

- Close is linearly interpolated from 9Nov to 19Nov, with increments of (148.86-154.08)/6 = -0.87
- Open(T) = Close(T-1)
- High = Open, Low = Close, since 19Nov2018 Close < 9Nov2018 Close
- Adj Close(T) = Close(T) * Adj Close(9Nov2018)/Close(9Nov2018)

In [35]:

```
mis_df = pd.DataFrame(np.zeros((5,8)),columns = sample.columns)
mis_df.loc[:,'Date'] = mis_workday
start_v = sample[sample['Date']=='2018/11/9']['Close'].tolist()[0]
end_v = sample[sample['Date']=='2018/11/19']['Close'].tolist()[0]
inc = (end_v-start_v)/6
mis_df.loc[:,'Close'] = np.cumsum([start_v]+[inc]*5)[1:]
mis_df.loc[:,'Open'] = [start_v]+mis_df.loc[:,'Close'].tolist()[:-1]
mis_df.loc[:,'High'] = mis_df['Open']
mis_df.loc[:,'Low'] = mis_df['Close']
```

```
adj_close = sample[sample['Date']=='2018/11/9']['Adj Close'].tolist()[0]\
    /sample[sample['Date']=='2018/11/9']['Close'].tolist()[0]
mis_df.loc[:,'Adj Close'] = mis_df['Close']*adj_close
mis_df.loc[:,'Adj/Close'] = adj_close

newdata = pd.concat([newdata,mis_df])
newdata = newdata.sort_values(['Date'])
newdata.index = range(1,len(newdata)+1)
```

In [36]:

```
newdata[newdata['Date']>'2018/11/8'].head(n=7)
```

Out[36]:

	Date	Signal	Open	High	Low	Close	Adj Close	Adj/Close
750	2018-11-09	19.074848	156.000000	156.029999	152.949997	154.080002	149.865677	0.972648
751	2018-11-12	0.000000	154.080002	154.080002	153.210002	153.210002	149.019473	0.972648
752	2018-11-13	0.000000	153.210002	153.210002	152.340002	152.340002	148.173268	0.972648
753	2018-11-14	0.000000	152.340002	152.340002	151.470001	151.470001	147.327064	0.972648
754	2018-11-15	0.000000	151.470001	151.470001	150.600001	150.600001	146.480860	0.972648
755	2018-11-16	0.000000	150.600001	150.600001	149.730001	149.730001	145.634655	0.972648
756	2018-11-19	17.769120	151.679993	152.029999	148.369995	148.860001	144.788452	0.972648

(3) a. Negative price, change to positive

In [37]:

```
neg_index = newdata[(newdata.loc[:,'Open':'Adj Close']<0).any(axis = 1)].index[0]
newdata.loc[neg_index,'Adj Close'] = abs(newdata.loc[neg_index,'Adj Close'])
```

In [38]:

```
newdata.loc[neg_index]
```

Out[38]:

```
Date          2018-10-10 00:00:00
Signal                  19.719477
Open                   160.820007
High                   160.990005
Low                    156.360001
Close                  156.559998
Adj Close              152.277847
Adj/Close                0.972648
Name: 728, dtype: object
```

(3) b. Wrong prices, failed “Low” < (“Open”, “Close”) < “High” rule

Correction:

- ”High” < “Low”: set “High” = max(“Open”,”Close”,”Low”), “Low” = min(“Open”,”Close”, “High”);

- “Close” > “High”: set “High” = “Close”;
- “Close” < “Low”: set “Low” = “Close”;
- “Open” > “High”: set “High” = “Open”;
- “Open” < “Low”: set “Low” = “Open”;
- Exception as found in 1 - (3) - c, record on 19Mar2018 with extreme high "Close" should set "Close" = "High" so that "Adj Close"/"Close" is consistent to its neighbering records.

In [39]:

```
wrong_price = newdata.apply(price_correct_check, axis = 1)
wrong_price = wrong_price[wrong_price.apply(len)>0]
for wi in wrong_price.index:
    if newdata.loc[wi,'Date'] == datetime.datetime.strptime('2018-03-19 00:00:00',
                                                            '%Y-%m-%d %H:%M:%S'):
        newdata.loc[wi,'Close'] = newdata.loc[wi,'High']
        newdata.loc[wi,'Adj/Close'] =\
            newdata.loc[wi,'Adj Close']/newdata.loc[wi,'Close']
    else:
        w = wrong_price.loc[wi]
        while len(w) > 0 :
            if w[0] == '1': #Low > High
                temp_max = newdata.loc[wi,'Open':'Close'].max()
                temp_min = newdata.loc[wi,'Open':'Close'].min()
                newdata.loc[wi,'High'] = temp_max
                newdata.loc[wi,'Low'] = temp_min
            elif w[0] == '2': #Close > High
                newdata.loc[wi,'High'] = newdata.loc[wi,'Close']
            elif w[0] == '3': #Close < Low
                newdata.loc[wi,'Low'] = newdata.loc[wi,'Close']
            elif w[0] == '4': #Open > High
                newdata.loc[wi,'High'] = newdata.loc[wi,'Open']
            elif w[0] == '5':
                newdata.loc[wi,'Low'] = newdata.loc[wi,'Open']
            else:
                pass
            w = price_correct_check(newdata.loc[wi])
```

In [40]:

```
(newdata.apply(lambda x: len(price_correct_check(x)), axis = 1)>0).any()
```

Out[40]:

```
False
```

(3) c. abnormal "Adj Close" No 2 and 3

In [41]:

```
newdata[newdata['Date'] == '2017-03-27']
```

Out[41]:

	Date	Signal	Open	High	Low	Close	Adj Close	Adj/Close
339	2017-03-27	16.032241	132.759995	135.070007	132.399994	134.740005	158.577637	1.176916

In [42]:

```
newdata.loc[339,'Adj Close'] = 0.954265 * newdata.loc[339,'Close']
newdata.loc[339,'Adj/Close'] = newdata.loc[339,'Adj Close']/newdata.loc[339,'Close']
newdata[336:].head()
```

Out[42]:

	Date	Signal	Open	High	Low	Close	Adj Close	Adj/Close

	Date	Signal	Open	High	Low	Close	Adj Close	Adj/Close
337	2017-03-23	15.705048	134.000000	135.699997	133.639999	134.779999	128.246490	0.951525
338	2017-03-24	15.131940	134.949997	135.470001	133.860001	134.490005	128.339050	0.954265
339	2017-03-27	16.032241	132.759995	135.070007	132.399994	134.740005	128.577671	0.954265
340	2017-03-28	16.215369	134.279999	135.899994	134.139999	135.789993	129.579605	0.954265
341	2017-03-29	15.265496	135.699997	136.490005	135.300003	136.229996	129.999481	0.954265

In [43]:

```
newdata[newdata['Date'] == '2016-12-05']
```

Out[43]:

	Date	Signal	Open	High	Low	Close	Adj Close	Adj/Close
263	2016-12-05	16.011011	131.970001	133.330002	131.889999	133.149994	166.175079	1.248029

In [44]:

```
newdata.loc[263,'Adj Close'] = 0.947616 * newdata.loc[263,'Close']
newdata.loc[263,'Adj/Close'] = newdata.loc[263,'Adj Close']/newdata.loc[263,'Close']
newdata[260:].head()
```

Out[44]:

	Date	Signal	Open	High	Low	Close	Adj Close	Adj/Close
261	2016-12-01	15.304759	132.250000	132.550003	130.289993	130.970001	124.109283	0.947616
262	2016-12-02	14.990113	130.940002	131.470001	130.520004	130.899994	124.042938	0.947616
263	2016-12-05	16.011011	131.970001	133.330002	131.889999	133.149994	126.175065	0.947616
264	2016-12-06	15.885051	133.520004	134.910004	132.740005	134.589996	127.539658	0.947616
265	2016-12-07	15.406135	134.589996	136.179993	134.179993	135.899994	128.781036	0.947616

Corrected sample data saved as "Sample Dataset_corrected.csv"

In [45]:

```
newdata.to_csv('Sample Dataset_corrected.csv')
```

1.2 Locate the name of the asset

I also tried to locate the asset name of the sample dataset so that I can find the real data to fill the dataset's missing values. Hinted from the description "a well-known broad market ETF", I first try collect the data for the top ETFs from website (https://stockmarketmba.com/listoftop100etfs.php) saved as csv file named "List of Top 100 ETFs.xlsx". Only thing we need is column A - symbols.

In [1]:
```
import yfinance as yf
import numpy as np
import pandas as pd
```

In [2]:
```
ETF_list = pd.read_excel('List of Top 100 ETFs.xlsx', usecols = ['Symbol'])
ETF_list = np.array(ETF_list['Symbol'].tolist())
ETF_list[:10]
```

Out[2]:
```
array(['SPY', 'IVV', 'VTI', 'VOO', 'QQQ', 'VEA', 'IEFA', 'AGG', 'VWO',
       'VTV'], dtype='<U4')
```

Below downloads the historical prices of the 100 ETFs from 1Jan2015 to 31Jan2021, which covers the sample data's dates, saved as csv file named "Prices for 100ETFs 19Nov2015-06Jan2020.csv"

In [3]:
```
ETFdata = pd.DataFrame()
start_date = '2015-11-01'
end_date = '2020-01-31'
```

In [4]:
```
for ticker in ETF_list:
    ticker_holder = yf.Ticker(ticker)
    temp_data = ticker_holder.history(start = start_date,end = end_date,
                                      auto_adjust = False)
    temp_data = temp_data[['Close']]
    temp_data = temp_data.rename({'Close': ticker}, axis='columns')
    ETFdata = pd.concat([ETFdata,temp_data], axis = 1, join = 'outer')
```

In [5]:
```
data = ETFdata['2015-11-19':'2020-01-06']
```

In [6]:
```
data.iloc[:,:10].head()
```

Out[6]:

Date	SPY	IVV	VTI	VOO	QQQ	VEA	IEFA	AGG
2015-11-19	208.550003	209.669998	106.760002	191.220001	113.709999	38.009998	56.209999	108.690002
2015-11-20	209.309998	210.619995	107.220001	191.929993	114.480003	37.930000	56.070000	108.620003
2015-11-23	209.070007	210.220001	107.110001	191.710007	114.150002	37.700001	55.730000	108.650002

Date	SPY	IVV	VTI	VOO	QQQ	VEA	IEFA	AGG
2015-11-24	209.350006	210.520004	107.389999	192.020004	114.050003	37.750000	55.720001	108.750000
2015-11-25	209.320007	210.529999	107.470001	191.960007	114.150002	37.810001	55.840000	108.800003

In [7]:

```
data.to_csv('Prices for 100ETFs 19Nov2015-06Jan2020.csv')
```

Then need to find if there is a match in this 100 ETFs.

In [8]:

```
sample = pd.read_csv('Sample Dataset_corrected.csv')
sample.loc[:,'Date']=pd.to_datetime(sample['Date'])
sample.set_index('Date',inplace = True)
sample_close = sample[['Close']]
sample_close = sample_close.rename({'Close':'Sample'}, axis='columns')
sample_close.head()
```

Out[8]:

Date	Sample
2015-11-19	116.059998
2015-11-20	116.809998
2015-11-23	117.389999
2015-11-24	118.250000
2015-11-25	119.169998

In [9]:

```
new_data = pd.concat([data,sample_close], axis = 1, join = 'outer')
new_data.iloc[:,-5:].head()
```

Out[9]:

Date	IWS	MINT	XLC	IEF	Sample
2015-11-19	70.970001	100.849998	NaN	106.089996	116.059998
2015-11-20	71.129997	100.860001	NaN	105.949997	116.809998
2015-11-23	71.209999	100.839996	NaN	106.089996	117.389999
2015-11-24	71.510002	100.860001	NaN	106.190002	118.250000
2015-11-25	71.500000	100.870003	NaN	106.250000	119.169998

In [10]:

```
diff_frame = new_data.sub(new_data['Sample'].tolist(),axis = 'index')
diff_frame.iloc[:,-5:].head()
```

Out[10]:

Date	IWS	MINT	XLC	IEF	Sample

Date	IWS	MINT	XLC	IEF	Sample
2015-11-19	-45.089997	-15.210000	NaN	-9.970002	0.0
2015-11-20	-45.680001	-15.949997	NaN	-10.860001	0.0
2015-11-23	-46.180000	-16.550003	NaN	-11.300003	0.0
2015-11-24	-46.739998	-17.389999	NaN	-12.059998	0.0
2015-11-25	-47.669998	-18.299995	NaN	-12.919998	0.0

In [11]:

```
diff_series = (np.round(diff_frame - 0, decimals=5)==0).sum(axis = 'index')
diff_series[diff_series>0]
```

Out[11]:

```
IWM       1019
VO           2
IWB          1
Sample    1038
dtype: int64
```

In [12]:

```
#get IWM full data
ticker = 'IWM'
a = yf.Ticker(ticker)
iwm = a.history(start=start_date, end = end_date, auto_adjust = False)
iwm = iwm['2015-11-19':'2020-01-06']
iwm.head()
```

Out[12]:

Date	Open	High	Low	Close	Adj Close	Volume	Dividends	Stock Splits
2015-11-19	116.440002	116.650002	115.739998	116.059998	107.580505	25512500	0.0	0
2015-11-20	116.480003	117.360001	116.379997	116.809998	108.275726	31697700	0.0	0
2015-11-23	116.709999	117.889999	116.680000	117.389999	108.813324	22716400	0.0	0
2015-11-24	116.879997	118.419998	116.559998	118.250000	109.610512	24994500	0.0	0
2015-11-25	118.300003	119.320000	118.110001	119.169998	110.463310	20772600	0.0	0

In [13]:

```
diff_df = iwm.loc[:,'Open':'Close'] - sample.loc[:,'Open':'Close']
diff_df.round(5).sum()
```

Out[13]:

```
Open     -5.15999
High      0.46997
Low      18.41996
Close    32.28998
dtype: float64
```

So **IWM is the underlying asset**. I will use it to evaluate the efficacy of the signals.

In [14]:

```
iwm.to_csv('IWM 19Nov2015-06Jan2020.csv')
```

2. Please analyze the signal's effectiveness or lack thereof in forecasting ETF price, using whatever metrics you think are most relevant.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import matplotlib as mpl
import seaborn as sns
from statsmodels.tsa.stattools import adfuller
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
pd.options.mode.chained_assignment = None
```

In [2]:

```
iwm = pd.read_csv('IWM 19Nov2015-06Jan2020.csv')
iwm = iwm.set_index(['Date'])
signal = pd.read_csv('Sample Dataset_corrected.csv')
signal = signal.set_index(['Date'])
```

In [3]:

```
data = pd.concat([signal['Signal'],iwm.loc[:,'Open':'Adj Close']],axis = 1)
data.head()
```

Out[3]:

	Signal	Open	High	Low	Close	Adj Close
Date						
2015-11-19	13.768540	116.440002	116.650002	115.739998	116.059998	107.580505
2015-11-20	13.608819	116.480003	117.360001	116.379997	116.809998	108.275726
2015-11-23	12.990589	116.709999	117.889999	116.680000	117.389999	108.813324
2015-11-24	12.667435	116.879997	118.419998	116.559998	118.250000	109.610512
2015-11-25	13.019910	118.300003	119.320000	118.110001	119.169998	110.463310

Remove outliers

In [4]:

```
rem_index = list(data[data['Signal']==0].index)
rem_index += list(data[data['Signal']>30].index)
rem_index
```

Out[4]:

```
['2018-11-12',
 '2018-11-13',
 '2018-11-14',
 '2018-11-15',
 '2018-11-16',
 '2019-12-27',
 '2019-12-30',
 '2019-12-31',
 '2020-01-02',
```

```
'2020-01-03',
'2020-01-06',
'2017-11-13']
```

In [5]:

```
data = data.drop(index = rem_index)
```

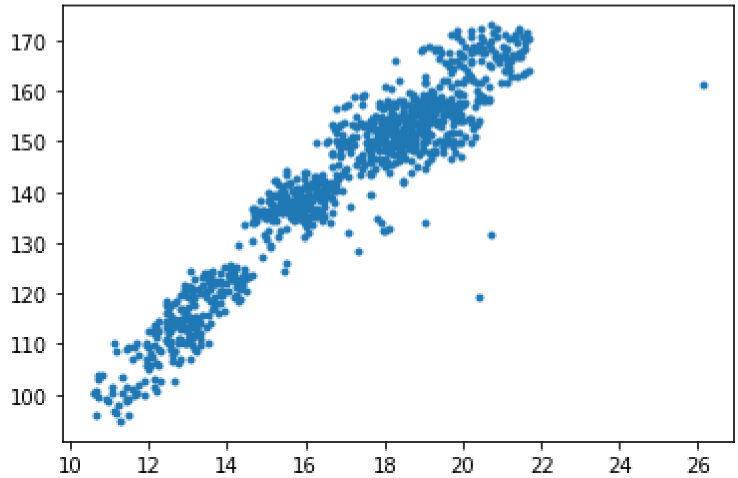
correlation

In [6]:

```
plt.plot(data['Signal'],data['Close'],'.')
```

Out[6]:

```
[&lt;matplotlib.lines.Line2D at 0x1313d7e67c0&gt;]
```



In [7]:

```
np.corrcoef(data['Signal'],data['Close'])
```

Out[7]:

```
array([[1.       , 0.9497295],
       [0.9497295, 1.       ]])
```

np.corrcoef(np.diff(np.log(np.array(data['Signal']))),np.diff(np.log(np.array(data['Close']))))

So signal is highly correlated to the price instead of the return.

For prediction accuracy, assuming same day prediction

OLS Regression, y = Close, x = Signal

Model: y = Intercept + Beta * x

In [8]:

```
x = np.array(data['Signal']).reshape(-1,1)
y = np.array(data['Close'])
```

First split the dataset to a training set and testing set to avoid overfitting

In [9]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.3,
                                                    random_state=1000)
```

In [10]:

```
reg = linear_model.LinearRegression()
```

```
reg.fit(x_train,y_train)
print("Training score: {0:.4f}".format(reg.score(x_train,y_train)))
print("Testing score: {0:.4f}".format(reg.score(x_test,y_test)))
```

```
Training score: 0.9037
Testing score: 0.8965
```

Then using ross validation with 5 folds to get average score for model selection

In [11]:
```
lin_scores = cross_val_score(reg, x_train,y_train,cv=5)
print("CV scores: {}".format(lin_scores))
print("CV mean score: {0:.4f}".format(np.mean(lin_scores)))
```

```
CV scores: [0.93345947 0.86396863 0.93366493 0.86284979 0.90840444]
CV mean score: 0.9005
```

Prediction accuracy for direction

In [12]:
```
y_pred = reg.predict(x)
same_dir = np.where(np.multiply(np.diff(y_pred),np.diff(y))>0,1,0)
print("Percentage of accurate predictions: {0:.4%}".format(\
        sum(same_dir)/(len(y_pred)-1)))
```

```
Percentage of accurate predictions: 47.4146%
```

Mean absolute error and Mean squared error

In [13]:
```
print("MAE: {0:.4f}, MSE: {1:.4f}".format(np.mean(np.abs(y_pred - y))
                                          ,np.mean((y_pred - y)**2)))
```

```
MAE: 4.3113, MSE: 33.1447
```

Complied in function to evaluate more models:

In [14]:
```
def reg_model(x,y):
    res = []
    x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                        test_size=0.3,
                                                        random_state=11)

    reg = linear_model.LinearRegression()
    reg.fit(x_train,y_train)
    #training score
    res.append(reg.score(x_train,y_train))
    #cross validation score
    lin_scores = cross_val_score(reg, x_train,y_train,cv=5)
    res.append(np.mean(lin_scores))
    #test score
    res.append(reg.score(x_test,y_test))
    #prediction accuracy for direction
    y_pred = reg.predict(x)
    same_dir = np.where(np.diff(y_pred)>0,1,0) == np.where(np.diff(y)>0,1,0)
    res.append(sum(same_dir)/(len(y_pred)-1))
    #Mean Abs error
    res.append(np.mean(np.abs(y_pred - y)))
    #Mean squared error
    res.append(np.mean((y_pred - y)**2))

    return res
```

In [15]:
```
res = pd.DataFrame(index = ['Training Score','Cross Val Score',
                            'Test Score', 'Pred Dir Accuracy',
                            'MAE','MSE'])
```

In [16]:
```
res['OLS'] = reg_model(x,y)
res
```

Out[16]:

	OLS
Training Score	0.899328
Cross Val Score	0.897022
Test Score	0.908022
Pred Dir Accuracy	0.475122
MAE	4.313130
MSE	33.134756

polynomial regression

Second order: y = Intercept + Beta1 *x* + *Beta2* x^2

In [17]:
```
x_poly_2 = np.concatenate((x,x**2),axis=1)
res['OLS-Poly2'] = reg_model(x_poly_2,y)
```

Thrid order: y = Intercept + Beta1 *x* + *Beta2* x^2 + Beta3 * x^3

In [18]:
```
x_poly_3 = np.concatenate((x,x**2,x**3),axis=1)
res['OLS-Poly3'] = reg_model(x_poly_3,y)
```

Signal and Open - last day Close for intraday prediction

Model: y = Intercept + Beta1 *x* + *Beta2* [Open at (T) day - Close at (T-1) day]

In [19]:
```
xo = np.concatenate((np.array(data['Signal'])[1:].reshape(-1,1),
                    (np.array(data['Open'][1:])\
                     - np.array(data['Close'][:-1])).reshape(-1,1)),
                   axis = 1)
yo = np.array(data['Close'])[1:]
```

In [20]:
```
res['Sig,Open-Close']=reg_model(xo,yo)
```

In [21]:
```
res
```

Out[21]:

	OLS	OLS-Poly2	OLS-Poly3	Sig,Open-Close
Training Score	0.899328	0.917690	0.917987	0.898230
Cross Val Score	0.897022	0.914870	0.913681	0.894660
Test Score	0.908022	0.923147	0.922986	0.909578
Pred Dir Accuracy	0.475122	0.475122	0.475122	0.473633
MAE	4.313130	3.962962	3.960450	4.323924

	OLS	OLS-Poly2	OLS-Poly3	Sig,Open-Close
MSE	33.134756	27.255578	27.200902	33.185884

Time Series model

For prices data, it worth to try time series model, first step is to stationize the data by take first order difference of it:

In [22]:

```
df = np.diff(data['Close'])
```

Then run unit root test to test for stationarity

In [23]:

```
result = adfuller(df)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
```

```
ADF Statistic: -32.110516
p-value: 0.000000
Critical Values:
        1%: -3.437
        5%: -2.864
        10%: -2.568
```

The statistic is very small compared to the different level of critical values, so can say that **the difference of "Close" is stationary.**

Then is to determine how many lags to included in the model, using acf (auto-correlation function) plot and pacf (partial auto-correlation function) plot:

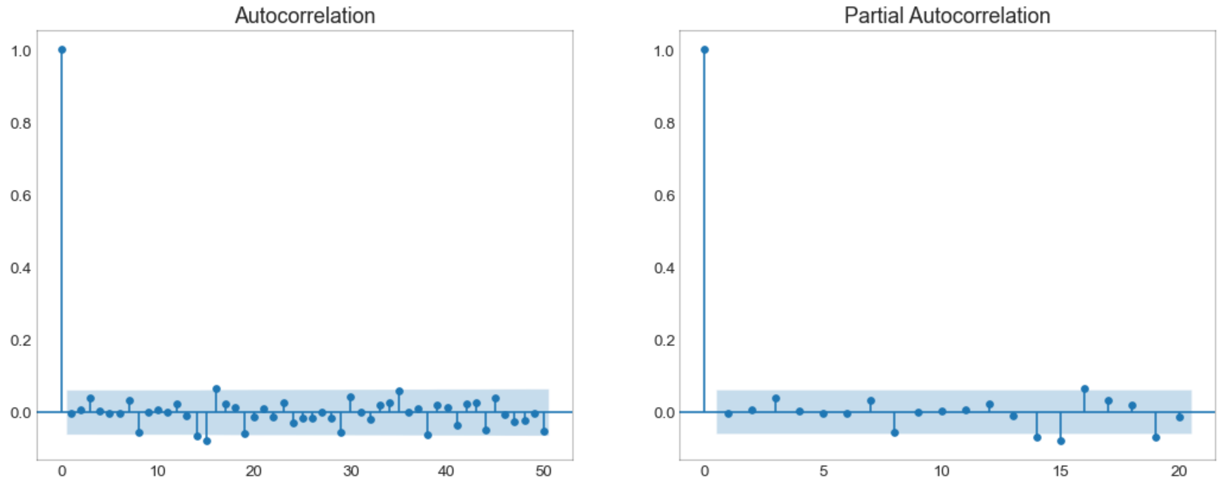
In [24]:

```
large = 22; med = 16; small = 12
params = {'axes.titlesize': large,
          'legend.fontsize': med,
          'figure.figsize': (16, 10),
          'axes.labelsize': med,
          'axes.titlesize': med,
          'xtick.labelsize': med,
          'ytick.labelsize': med,
          'figure.titlesize': large}
plt.rcParams.update(params)
plt.style.use('seaborn-whitegrid')
sns.set_style("white")

fig, (ax1, ax2) = plt.subplots(1, 2,figsize=(16,6), dpi= 80)
plot_acf(df, ax=ax1, lags=50)
plot_pacf(df, ax=ax2, lags=20)


ax1.spines["top"].set_alpha(.3); ax2.spines["top"].set_alpha(.3)
ax1.spines["bottom"].set_alpha(.3); ax2.spines["bottom"].set_alpha(.3)
ax1.spines["right"].set_alpha(.3); ax2.spines["right"].set_alpha(.3)
ax1.spines["left"].set_alpha(.3); ax2.spines["left"].set_alpha(.3)


ax1.tick_params(axis='both', labelsize=12)
ax2.tick_params(axis='both', labelsize=12)
plt.show()
```



From the graphs, it is suggested that no lags are significant for "Close". But I still try 3 different models

1. Close(T) = Intercept + Signal(T) + Signal(T-1) + Close(T-1)

In [25]:
```
y = np.array(data['Close'])[1:]
x1 = np.array(data['Signal'])[1:].reshape(-1,1)
x2 = np.array(data['Signal'])[:-1].reshape(-1,1)
x3 = np.array(data['Close'])[:-1].reshape(-1,1)
x = np.concatenate((x1,x2,x3),axis=1)

res['TS(T-1)']=reg_model(x,y)
```

1. Close(T) = Intercept + Signal(T) + Signal(T-1) + Signal(T-2) + Close(T-1) + Close(T-2)

In [26]:
```
y = np.array(data['Close'])[2:]
x1 = np.array(data['Signal'])[2:].reshape(-1,1)
x2 = np.array(data['Signal'])[1:-1].reshape(-1,1)
x3 = np.array(data['Signal'])[:-2].reshape(-1,1)
x4 = np.array(data['Close'])[1:-1].reshape(-1,1)
x5 = np.array(data['Close'])[:-2].reshape(-1,1)
x = np.concatenate((x1,x2,x3,x4,x5),axis=1)

res['TS(T-2)']=reg_model(x,y)
```

1. Close(T) = Intercept + Signal(T) + Signal(T-1) + [Close(T-1) - Close(T-2)]

In [27]:
```
y = np.array(data['Close'])[2:]
x1 = np.array(data['Signal'])[2:].reshape(-1,1)
x2 = np.array(data['Signal'])[1:-1].reshape(-1,1)
x3 = np.array(data['Close'])[1:-1].reshape(-1,1)-np.array(data['Signal'])[:-2].resha
x = np.concatenate((x1,x2,x3),axis=1)

res['TS(T-2)*']=reg_model(x,y)
```

Comparison of the results

In [28]:
```
res
```

Out[28]:

	OLS	OLS-Poly2	OLS-Poly3	Sig,Open-Close	TS(T-1)	TS(T-2)	TS(T-2)*
Training Score	0.899328	0.917690	0.917987	0.898230	0.994377	0.994621	0.992270
Cross Val Score	0.897022	0.914870	0.913681	0.894660	0.994201	0.994461	0.991959
Test Score	0.908022	0.923147	0.922986	0.909578	0.994117	0.993730	0.991036
Pred Dir Accuracy	0.475122	0.475122	0.475122	0.473633	0.485352	0.478006	0.477028
MAE	4.313130	3.962962	3.960450	4.323924	1.039149	1.036317	1.248715
MSE	33.134756	27.255578	27.200902	33.185884	1.923234	1.896165	2.720528

Including lag 1 price improve the prediction accuracy by a lot, indicated by the increased scores (R^2 socre) and reduced MAE and MSE.

Conclusion

Effectiveness:

1. The Signal can give a pretty good estimation of the price level for some models
2. Only one column of Signal is given which speed up the decision making process so that we don't need to look at many metrics but one single series

Lack:

1. The signal performs poorly in estimating the direction of the prices changes, lower than 50% for the models I chooses (maybe other model can do better), but without more information, it is even worse than tossing a coin.
2. Signals for some days at the end of the datasets are zeros, which is quite an issue as it is the unique product of the company, we can download market data from public source, but are paying for the "Signals".

3. Summary for the Portfolio Manager addressing your observations about the efficacy and believability of the product, and recommendation for next steps.

Believability: From section 1, the dataset failed some of the checks. Some errors are minor, like negative prices and illegal prices, considering the number of errors is not large and this kind of errors don't influence others. But some errors like data on weekends and public holidays, if not copies from other records, then might suggest all following rows are in wrong dates, sabotaging the whole dataset. So, the believability of the dataset is not very sound.

Efficacy: from section 2, the estimation accuracy for the dataset is excellent if is for the level of the price but is poor for the direction of price changes.

Recommendations:

1. Ask vendor to fix the data quality issue.
2. Ask vendor to provide the description of the signals to facilitate model selection.
3. Ask vendor to provide more sample dataset to check the efficacy
4. Comparing the signal to current models to evaluate which one is better