TOC

Start

- Copy this notebook. In Google Colab use `File -> Save a Copy in Drive.`
- Use the "Text" blocks to provide explanations wherever you find them necessary.
- Highlight your answers inside these text fields to ensure that we don't miss it while grading your HW.

Setup

- Code to download the data directly from the colab notebook.
- If you find it easier to download the data from the kaggle website (and uploading it to your drive), you can skip this section.

```
## First mount your drive before running analysis code
from google.colab import drive
drive.mount('/content/drive')

## Create a folder for the this HW and change to that dir
%cd drive/MyDrive/cse519
```

```
Mounted at /content/drive
/content/drive/MyDrive/cse519
```

```
## packages
!pip install -q kaggle
!pip install -q pandas
!pip install -q scikit-learn
!pip install -q numpy
!pip install -q Matplotlib
!pip install -q seaborn
!pip install pingouin
```

```
Collecting pingouin
  Downloading pingouin-0.5.3-py3-none-any.whl (198 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 198.6/198.6 kB 4.5 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.19 in /usr/local/lib/python3.10/dist-packages (from pingouin) (1.23.5)
Requirement already satisfied: scipy>=1.7 in /usr/local/lib/python3.10/dist-packages (from pingouin) (1.11.3)
Requirement already satisfied: pandas>=1.0 in /usr/local/lib/python3.10/dist-packages (from pingouin) (1.5.3)
Requirement already satisfied: matplotlib>=3.0.2 in /usr/local/lib/python3.10/dist-packages (from pingouin) (3.7.1)
Requirement already satisfied: seaborn>=0.11 in /usr/local/lib/python3.10/dist-packages (from pingouin) (0.12.2)
Requirement already satisfied: statsmodels>=0.13 in /usr/local/lib/python3.10/dist-packages (from pingouin) (0.14.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from pingouin) (1.2.2)
Collecting pandas-flavor>=0.2.0 (from pingouin)
  Downloading pandas_flavor-0.6.0-py3-none-any.whl (7.2 kB)
Collecting outdated (from pingouin)
  Downloading outdated-0.2.2-py2.py3-none-any.whl (7.5 kB)
Requirement already satisfied: tabulate in /usr/local/lib/python3.10/dist-packages (from pingouin) (0.9.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.2->pingouin) (1.1.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.2->pingouin) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.2->pingouin) (4.43.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.2->pingouin) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.2->pingouin) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.2->pingouin) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.2->pingouin) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.2->pingouin) (2.8.2
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0->pingouin) (2023.3.post1)
Requirement already satisfied: xarray in /usr/local/lib/python3.10/dist-packages (from pandas-flavor>=0.2.0->pingouin) (2023.7.0)
```

```
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.13->pingouin) (0.5.3)
Requirement already satisfied: setuptools>=44 in /usr/local/lib/python3.10/dist-packages (from outdated->pingouin) (67.7.2)
Collecting littleutils (from outdated->pingouin)
  Downloading littleutils-0.2.2.tar.gz (6.6 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from outdated->pingouin) (2.31.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->pingouin) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->pingouin) (3.2.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.2->statsmodels>=0.13->pingouin) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->outdated->pingouin) (
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->outdated->pingouin) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->outdated->pingouin) (2.0.6)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->outdated->pingouin) (2023.7
Building wheels for collected packages: littleutils
  Building wheel for littleutils (setup.py) ... done
  Created wheel for littleutils: filename=littleutils-0.2.2-py3-none-any.whl size=7028 sha256=1a02bdbd0357fb09cd337490aed6cfb9831b91702c
  Stored in directory: /root/.cache/pip/wheels/3d/fe/b0/27a9892da57472e538c7452a721a9cf463cc03cf7379889266
Successfully built littleutils
Installing collected packages: littleutils, outdated, pandas-flavor, pingouin
Successfully installed littleutils-0.2.2 outdated-0.2.2 pandas-flavor-0.6.0 pingouin-0.5.3
```

```
# @title
## Upload the file by clicking on the browse
from google.colab import files
files.upload()

## Create a new API token under "Account" in the kaggle webpage and download the json file


!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!kaggle competitions download -c optiver-trading-at-the-close
!unzip optiver-trading-at-the-close.zip
!ls
```

Q1: Anomalies and Cleaning, 15 pts

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

col_names = [
  "stock_id",
  "date_id",
  "seconds_in_bucket",
  "imbalance_size",
  "imbalance_buy_sell_flag",
  "reference_price",
  "matched_size",
  "far_price",
  "near_price",
  "bid_price",
  "bid_size",
  "ask_price",
  "ask_size",
  "wap",
  "target",
  "time_id",
  "row_id"
]
dtypes = {
  "stock_id": int,
  "date_id":int,
  "seconds_in_bucket":int,
  "imbalance_size":np.float64,
  "imbalance_buy_sell_flag":int,
  "reference_price":np.float64,
  "matched_size":np.float64,
  "far_price":np.float64,
  "near_price":np.float64,
  "bid_price":np.float64,
  "bid_size":np.float64,
  "ask_price":np.float64,
  "ask_size":np.float64,
```

```
  "wap":np.float64,
  "target":np.float64,
  "time_id":int,
  "row_id": "string",
}
csv = pd.read_csv("train.csv")
```

```
csv.head(100000)
```

	stock_id	date_id	seconds_in_bucket	imbalance_size	imbalance_buy_sell_flag	reference_price	matched_size	far_price	near_pri
0	0	0	0	3180602.69	1	0.999812	1.338028e+07	NaN	Na
1	1	0	0	166603.91	-1	0.999896	1.642214e+06	NaN	Na
2	2	0	0	302879.87	-1	0.999561	1.819368e+06	NaN	Na
3	3	0	0	11917682.27	-1	1.000171	1.838975e+07	NaN	Na
4	4	0	0	447549.96	-1	0.999532	1.786061e+07	NaN	Na
...	...	...	...	...	...	...	...	...	
99995	190	9	260	3934.27	-1	0.999586	1.509824e+07	NaN	Na
99996	191	9	260	34768651.60	1	0.999900	5.515022e+08	NaN	Na
99997	192	9	260	1354796.26	-1	0.997366	3.732464e+06	NaN	Na
99998	193	9	260	1178858.50	1	1.000181	6.424064e+07	NaN	Na
99999	194	9	260	3447137.99	-1	1.001393	5.431734e+06	NaN	Na

100000 rows × 17 columns

Clean Data

We first check for

- null values - We find that the above columsn have large number of na values
- duplicates - The dataset doesn't contain any significant duplicate rows. This makse ssense as row_id is a uique key for each entry, which renders the problem of duplicate values as irrelevant. There are still scenarios which might generate duplicate data, but we haven't faced it .
- outliers IN terms of outlers a direct imputation or simply deleting them may lead to loss of critical data operation. Therefore using `df.describe()` and `df.info()` to analyse dataset sand finding the best possible action is the way

```
# fill
df=csv
# df.info()
print(df.isna().sum()) # we get same result as print(df.isnull().sum())
print(df.duplicated().sum())
```

```
stock_id                      0
date_id                       0
seconds_in_bucket             0
imbalance_size              220
imbalance_buy_sell_flag       0
reference_price             220
matched_size                220
far_price                 2894342
near_price                2857180
bid_price                   220
bid_size                      0
ask_price                   220
ask_size                      0
wap                         220
target                       88
time_id                       0
row_id                        0
dtype: int64
0
```

How to treat them?

We shouldn't simply delete rows since only few columns(not entire row)are NaN, especially

- imbalance_size 220
- reference_price 220
- matched_size 220
- far_price 2894342
- near_price 2857180
- bid_price 220
- ask_price 220
- wap 220
- target 88

This hints a corrleation between the way it was calculated for the stocks.

Based on the analysis of the columns ,

- Related values are missing for all related columns (220) I think that we should remove these as most of the columns seems to be empty.
- We still have `far_price` and `near_price` as null as they are much more missing than other values. Initially I assumed we can just leave them as is, as most of the operations are NaN safe, however when I Started building models I realised it is better to remove unnceesssary bulk data especially if it doesnot add to the quality of the model. The mean_squared _error of linear regression improved from `8.89` to `69.xx` with improvement is other parameters as well , on removing these rows and Random _Forest which was earlier unable to build , started showing result in 2 minutes, when i removed these rows.

```
# Column wise treatmnet
mdfd_csv = csv[csv['reference_price'].notna()]

print(mdfd_csv.isna().sum()) # we get same result as print(df.isnull().sum())

# Imputing values :
#For now we impute with -1 , but we
# mdfd_csv.fillna()
```

```
stock_id                       0
date_id                        0
seconds_in_bucket              0
imbalance_size                 0
imbalance_buy_sell_flag        0
reference_price                0
matched_size                   0
far_price                2894122
near_price               2856960
bid_price                      0
bid_size                       0
ask_price                      0
ask_size                       0
wap                            0
target                         0
time_id                        0
row_id                         0
dtype: int64
```

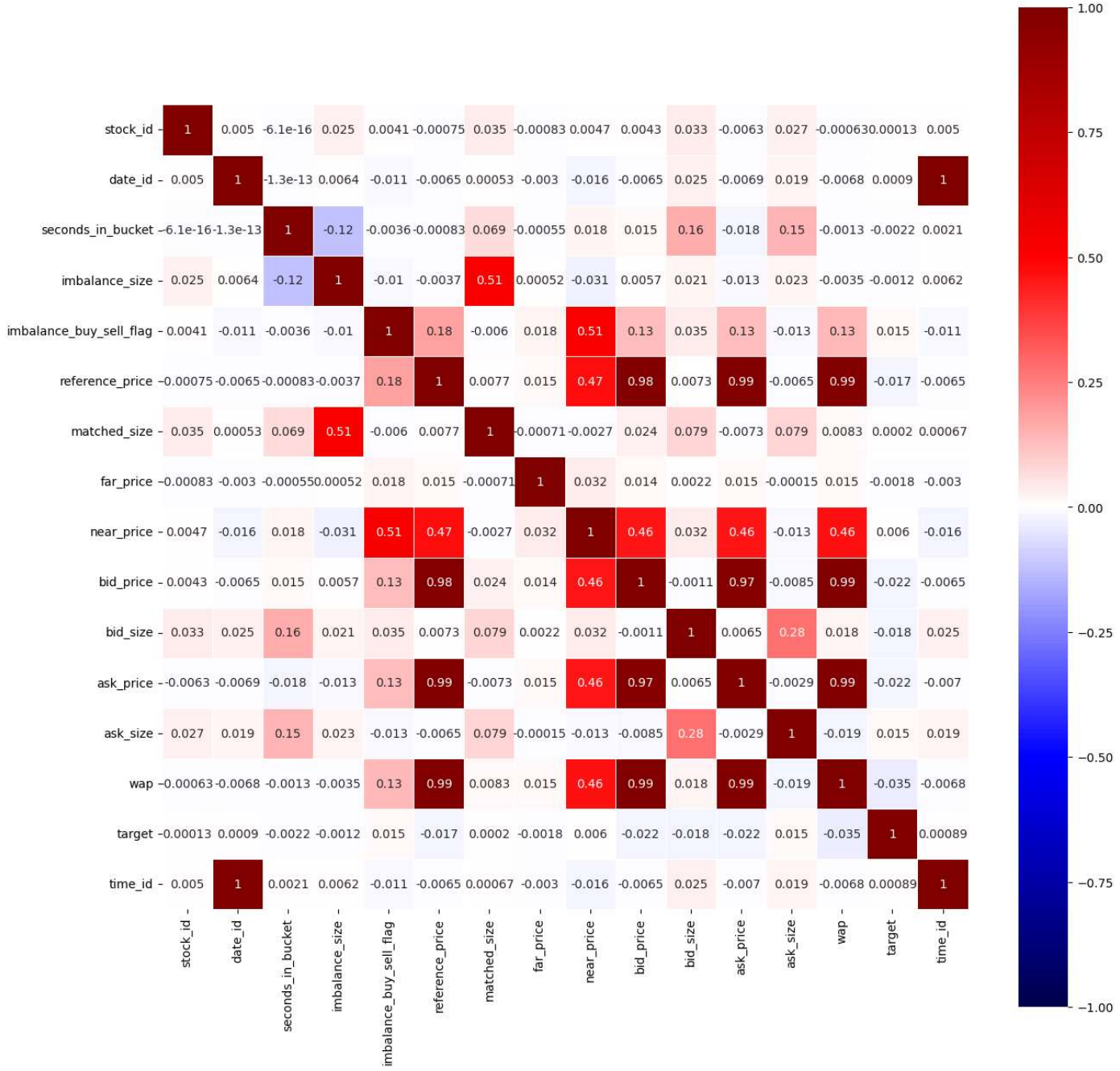
```
## NULL CHECK
# cols = ['far_price','near_price']
cols = ['reference_price']
# df[df[cols].isna().all(1)]
nulldata=mdfd_csv[mdfd_csv[cols].isna().all(1)]
nulldata
# sns.heatmap(nulldata.corr(), cmap='seismic', linewidths=0.5, annot=True);
```

	stock_id	date_id	seconds_in_bucket	imbalance_size	imbalance_buy_sell_flag	reference_price	matched_size	far_price	near_price	bi

Q2: Pairwise Correlation Table and Explanation. 10 pts

```
fig, ax = plt.subplots(figsize=(15,15))
ax=sns.heatmap(mdfd_csv.corr(method='pearson'), vmin=-1, vmax=1,cmap='seismic', linewidths=0.5, annot=True,square=True)
```

```
<ipython-input-7-17a1eaae57c7>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version,
  ax=sns.heatmap(mdfd_csv.corr(method='pearson'), vmin=-1, vmax=1,cmap='seismic', linewidths=0.5, annot=True,square=True)
```



Explanation

1. `time_id` - `date_id` [1] : because both are sequential in nature and continously and steadily increasing.
2. `ask_price` - `bid_price` [.97]: An obvious correlation since both have to match each other for trade to take place so if one increases it automatically affects the other and vice versa.

3. `wap` - `ask_price` / `bid_price` [.99] since the formula directly uses bid price and and ask_price and normalises them in a way to remove the effect of bid_size and ask_size. Thus wap becomes highly correlated with what factor remains. We also confirm this with a very low correlation (close to hovering around zero) between wap and bid_size[.018]/ask_size[-.019] for the data <!-- 1. An interesting find is that this correlation is carried forward to the `target` value. Correlation between -
    - `target` - `bid_size` [-.018]
    - `target` - `ask_size` [+.015]
4. Correlation bwtween `wap` and `target` -->
5. `wap` - `reference_price` [.99] `near_price` [.46] • The `wap` variable, which measures the weighted average price in the non-auction book, is highly correlated with the `reference_price`, and `near_price` variables. This is because these variables are all based on the bid and ask prices and sizes in the market, and reflect the supply and demand of the stocks.
6. The `imbalance_buy_sell_flag` variable, which indicates the direction of auction imbalance, is positively correlated with the `target` [.015] variable. This means that when there is more buy-side imbalance, the stock price tends to increase more in the future, and when there is more sell-side imbalance, the stock price tends to decrease more in the future.
7. imbalance_size -matched_size [.51] - The `imbalance_size` variable, which measures the amount unmatched at the current reference price, is positively correlated with the `matched_size` variable, which measures the amount that can be matched at the current reference price. This is because when there is more imbalance, there is more scope of matching, and vice versa.
8. imbalance_buy_sell_flag - near_price [.51]
9. There isn't a significant correlation between `far_price`, `near_price` [0.032] which can be motivated out of 2 factors
    - lack of enough data as most of these columns are NaN. One reason could be that not all stocsk are traded everyday , so these can be the less popular ones.
    - there is considerable difference in weightage with respect to near_price as compared to far_price. far_price contains only non-aauction data whereas near_price contains both in

```
mdfd_csv.columns

''' `stock_id`, `date_id`, `seconds_in_bucket`, `imbalance_size`,
       `imbalance_buy_sell_flag`, `reference_price`, `matched_size`,
       `far_price`, `near_price`, `bid_price`, `bid_size`, `ask_price`,
       `ask_size`, `wap`, `target`, `time_id`, `row_id`],
      dtype=`object`
      '''
      ''' 'stock_id', 'date_id', 'seconds_in_bucket', 'imbalance_size',
       'imbalance_buy_sell_flag', 'reference_price', 'matched_size',
       'far_price', 'near_price', 'bid_price', 'bid_size', 'ask_price',
       'ask_size', 'wap', 'target', 'time_id', 'row_id']
      '''

     ' `stock_id`, `date_id`, `seconds_in_bucket`, `imbalance_size`,\n       `imbalance_buy_sell_flag`, `reference_price`, `matched_size`,\n      `far_price`, `near_price`, `bid_price`, `bid_size`, `ask_price`,\n       `ask_size`, `wap`, `target`, `time_id`, `row_id`],\n      dtyp
     e=`object`\n      '
```

Q3: Average records stockID vs Day, 25 pts

distance function between entries

- a. autocorrelation, 10 pts
- b. measure the distance, 5 pts
- c. clustering algorithm, 10 pts

Part 3 a Auto correlation

- We divide the stocks first into group for each stock and then move on to further divide the data into data wise groups.
- For simplicity, we calculate distance just based on the previous days's reference_price.
    - Reason: Reference price contains the most wholiistic view of price (big picture) for a day as it contains both auction and non-action order books.
-

Define an

- “average” or “consensus” record for each stock id s on a particular day d measuring how the stock s performs on day d’s close.

> we do this by simply taking mean based on reference_price. this is significant and enough as reference_price in itself contains the most probable price a which trade could have taken place for a day. It includes by definition both non-auction and auction order(combined) order book. `The price at which paired shares are maximized, the imbalance is minimized and the distance from the bid-ask midpoint is minimized, in that order. Can also be thought of as being equal to the near price bounded between the best bid and ask price.`
> __Nasdaq also provides an indication of the fair price called the reference price. The reference price is calculated as follows:
>
> > - If the near price is between the best bid and ask, then the reference price is equal to the near price
> > - If the near price > best ask, then reference price = best ask
> > - If the near price < best bid, then reference price = best bid So the reference price is the near price bounded between the best bid and ask.__

- Then define a distance function between pairs of “stock-day” distance records, measuring how similarly they are.

> We do this using reference_price and its auto correlation based on lag (1-10)

```
#TODO::# Create new dataset with stock and date and datewise aggregated parameters
## day wise average

def set_distance(test_stock):
    df=test_stock.copy()
    df['distance']=np.nan
    df.loc[df.index[0],'distance']= df.loc[df.index[0], 'reference_price']
    for i in range(1, len(df)):
      idx=df.index[i]
      df.loc[idx, 'distance'] = df.loc[idx, 'reference_price']-df.loc[df.index[i-1], 'reference_price']
    return df

## Getting auto correlation of distance for lag 1 to 10
def get_autocorr(dadps):
    lag_range=11
    autocorr=np.zeros(lag_range)
    for i in range (1,lag_range):
      val=dadps.distance.autocorr(lag=i)
      # pd.concat(autocorr,val)
      # autocorr[i]=(val)
      autocorr[i]=val
    return autocorr


# datewise_aggregate_data_per_stock -dadps
def datewise_agg(stock_grp):
  datwise_data_per_stock=stock_grp.groupby('date_id')
  print(datwise_data_per_stock.ngroups)
  dadps=datwise_data_per_stock.agg({'reference_price':'mean','distance':'mean'})

  dadps['date_id']=dadps.index
  # dadps.columns
  return dadps

unique_stock_count=10#(len(set(mdfd_csv.stock_id))) # 10
stkwise_grouped_stocks=mdfd_csv.groupby('stock_id')
stck_grp_list={}
for i in range (1,unique_stock_count):
    stck_grp_list[i]=stkwise_grouped_stocks.get_group(i)


autocorr_df=pd.DataFrame()

for i in stck_grp_list.keys():
  print(i)
  stock_grp_temp=stck_grp_list[i]
  stock_grp=set_distance(stock_grp_temp) ## 3a step 1
  col_name='stock'+str(i)
  datewise_agg_dat=datewise_agg(stock_grp=stock_grp)
  autocorr_df[col_name]=get_autocorr(datewise_agg_dat) ## 3a step 2


autocorr_df
```

```
# print('Column Name : ', columnName)
# print('Column Contents : ', columnData.values)
# x=list(range(1,10))
# fig,ax=plt.subplo
# for (columnName, columnData) in autocorr_df.iteritems():
#     plt.plot(x, columnData, label = columnName)

# plt.legend()
```

```
1
481
2
481
3
481
4
481
5
481
6
481
7
481
8
481
9
481
```

	stock1	stock2	stock3	stock4	stock5	stock6	stock7	stock8	stock9
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	-0.047536	-0.060767	-0.009472	-0.017870	-0.056998	-0.035334	-0.029009	-0.028092	-0.026221
2	-0.045747	0.021878	-0.015568	-0.062949	-0.019530	-0.015275	0.004980	-0.050427	-0.051577
3	0.068300	-0.001010	0.015927	0.028460	0.016163	-0.011317	0.001863	0.071239	0.068333
4	-0.033721	0.011604	0.003411	0.059975	-0.025954	0.013158	0.006779	0.013210	-0.015748
5	-0.025980	-0.012153	-0.000661	-0.089377	-0.012700	0.023071	-0.026392	-0.023013	-0.006230
6	0.031052	0.003349	0.024876	0.027257	-0.014081	-0.044031	-0.034126	0.029576	0.004573
7	0.023577	-0.024594	-0.022048	0.010098	0.052205	0.016329	0.037616	-0.050606	0.029540
8	-0.023550	0.026502	-0.036163	-0.013813	-0.011526	0.055256	-0.018507	0.022780	-0.011097
9	-0.024208	-0.017767	-0.004093	0.015528	0.026921	-0.074791	0.020703	0.024279	-0.030497
10	0.010466	0.032496	0.075114	-0.014868	-0.019498	0.029261	0.003454	-0.063620	0.016563

Problem 3

- a. For each stock, measure the autocorrelation of the average distance between day i and day i+k for -10 <= k <= +10. On average (over all stocks), is there a statistically significant degree of autocorrelation in the market? Present your evidence for or against? Are there particular stocks whose performance is unusually autocorrelated, in a statistically significant way? (10 points)
  Points to consider here ➞ FOr each stock in the list:

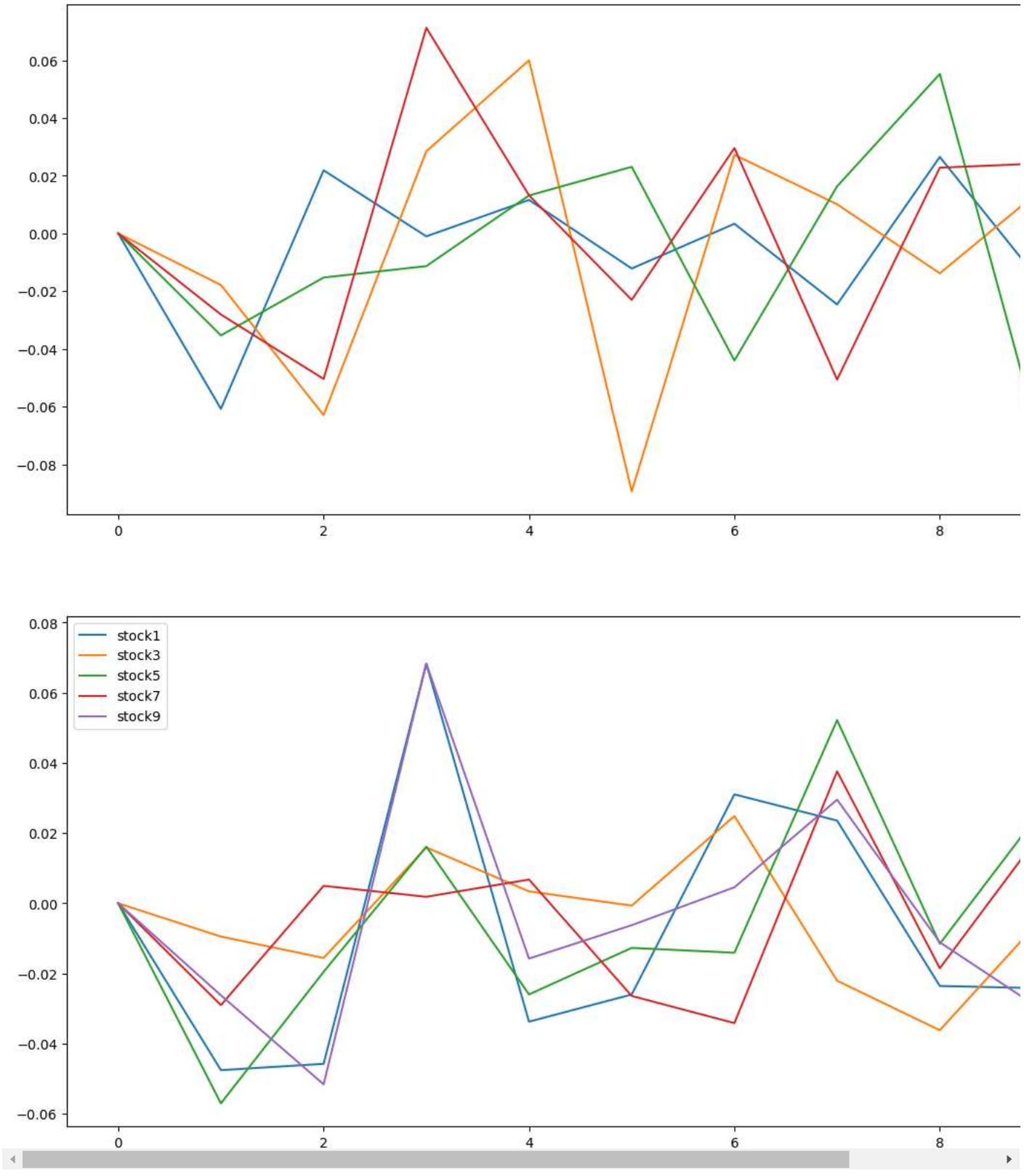
  1. We calculated daywise distance data for each stock, which shows whether the stock has increased or decreased as compared to the previous reference_price.
  2. We then aggregate the data based on mean to get datewise movement of each stock for each day.
  3. We calculated autocorrelation for each stock with lag 0-10 (which translates to -10 to 10) .We maintain this data for each stock.
     Reason: Here we do this for multiple lags to identify multiple patterns in the data set, like repetition in 1 days, 2 days, 7 days (weekly cycles), etc. and use it to plot line chart which gives us fair idea of the relative movement of a stock comapred to itself, across mulitiple timelines.

Here we plot a line chart of multiple stocks in a single line chart to see a pattern of movement. For sake of simplicity and clearity , I have opted to show chart for only 10 stocks,But the applicability is clear.

We see that `Stock 1 nad 9` and `Stock-4 and 6` have some similarity in movement as bothhave closely following autocorrelation lags. This may mean that there might be some relationbewtween them for w.g stocks of same industry, or of indsutry either based on it or sourcing to it.Such factors play a crucvial cyclic effect and can be visualised through autocorrelation lags .

```
x=list(range(0,11))
fig, ax = plt.subplots(2,1,figsize=(10,10))

for (columnName, columnData) in autocorr_df.items():
    idx=int(columnName.replace("stock",""))
    ax[(idx%2)].plot(x, columnData, label = columnName)
    ax[(idx%2)].legend(loc="upper right")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b04b1d6f7c0>
```



Problem 3b

- b. For each pair of stocks a and b, measure the distance between a on day i and b on day i for every day i. Are there pairs of stocks which are unusually similar on a consistent basis? If a for-loop proves too slow, investigate broadcastable numpy/pandas operations. (5 points)

Ans:

We have already seen our line charts from part a . In each line chart we can see stocks which have very similar trajectory of auto correlation. This means they behave similarly in in terms of lags.
This can be due to a multiple of factors , namely, stocks belonging to same sectors which behave similarly due to sudden surge of interest in the sector or lack of interest thereof leading to downlfall .
Points to consider here ➞ FOr each stock in the list:

1. We calculated daywise summarised `reference_price` and `distance` data for each stock, which shows whether the stock has increased or decreased as compared to the previous reference_price.

For sake of compuational ease, I picked 50 stocks. I took less to avoid data assimilationa nd presentation issues, which I was facing when presenting all 200.

1. We then aggregate the data based on mean to get datewise movement of each stock for each day.
2. After that we measure pairwise column Pearson-correlation to get individual correlation between each stock in the subset.

Next we seek to find day wise distance between stocks and a relation based on distance between stock for each day i.

Add On : I was facing a lot of slowness when calculating these parametes for each stock for each day . I employed python multiprocessing to speed it by 50% but it still takes 20+ minutes if we double the size.

Double-click (or enter) to edit

```
cols=[ 'stock_id','reference_price','time_id', 'row_id']
import multiprocessing as mp
import time
# q3b=mdfd_csv[cols]
# q3b.shape
# q3b['distance']=''
pool = mp.Pool(processes=mp.cpu_count())
# autocorr_df


q3b =pd.DataFrame()
unique_stock_count=50#(len(set(mdfd_csv.stock_id))) # 10
stkwise_grouped_stocks=mdfd_csv.groupby('stock_id')
stck_grp_list={}
for i in range (1,unique_stock_count):
    stck_grp_list[i]=stkwise_grouped_stocks.get_group(i)


# datewise_aggregate_data_per_stock_per_day -dadps
def datewise_agg_day(stock_grp,i):
  datwise_data_per_stock=stock_grp.groupby('date_id')
  # print(datwise_data_per_stock.ngroups)

  dadps=datwise_data_per_stock.agg({'reference_price':'mean','distance':'mean'})
  stckNm='stock'+str(i)
  dadps.rename(columns={'reference_price': 'reference_price_'+stckNm, 'distance': 'distanc_'+stckNm}, inplace=True)
  if i==1:
    dadps['date_id']=dadps.index
  # print(dadps.columns)
  # print(dadps)
  return dadps
# start_time = time.time()

def func(arg):
  i,stock_grp_temp=arg
  print(i)
  stock_grp=set_distance(stock_grp_temp) ## 3a step 1
  datewise_agg_dat=datewise_agg_day(stock_grp,i)
  return datewise_agg_dat
```

```
datewise_agg_dat = pool.map( func, [(idx,row) for idx,row in stck_grp_list.items()])
# datewise_agg_dat
for dat in datewise_agg_dat:
  dfs = [q3b,dat]
  q3b = pd.concat([df.stack() for df in dfs], axis=0).unstack()

q3b
```

In th above code, we calculated daywise aggregated data for each day based on mean

```
fil_cols=[x for x in q3b.columns if  'dist' in x]
q3b1=q3b[fil_cols]
q3b1
```

```
fil_cols=[x for x in q3b.columns if  'refer' in x]
q3b2=q3b[fil_cols]
q3b2
```

We get multiple interesting observation on finding pairwise correlation for all columns.

1. All of the columns have a correlation of `> .99`.
2. We get a Confidence INterval 95% fro almost all of them, which means the data is such that almost all have a clearly establsihed correlation.
3. P-value Uncorrrected - `p-unc` - is also 0.0 for almost all whcih means that the dat was strong enough. One of the reason I feel for this highly correlation situation is that reference_price is a perentage and not the actual price. Percentage change tends to smoothen the actual fluctuation (somewhat like log do for power law)

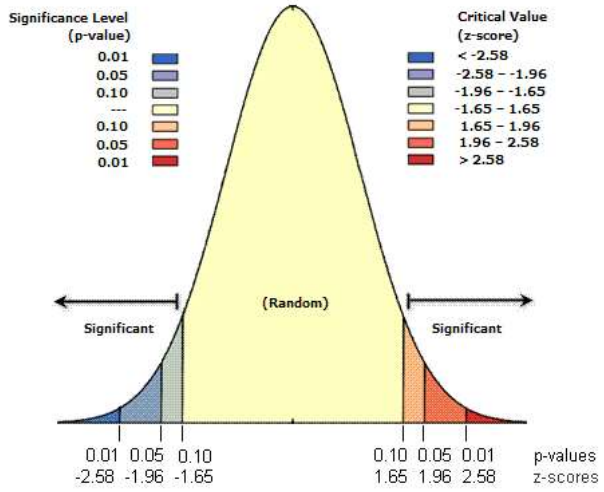
```
import pingouin as pg
pairwise_reln=pg.pairwise_corr(q3b1, method='pearson')
pairwise_reln.sort_values(by=['r']) # Sorting by r values
pairwise_reln
```

	X	Y	method	alternative	n	r	CI95%	p-unc	BF10	power
0	distanc_stock1	distanc_stock10	pearson	two-sided	481	0.994328	[0.99, 1.0]	0.0	inf	1.0
1	distanc_stock1	distanc_stock11	pearson	two-sided	481	0.985677	[0.98, 0.99]	0.0	inf	1.0
2	distanc_stock1	distanc_stock12	pearson	two-sided	481	0.994507	[0.99, 1.0]	0.0	inf	1.0
3	distanc_stock1	distanc_stock13	pearson	two-sided	481	0.993323	[0.99, 0.99]	0.0	inf	1.0
4	distanc_stock1	distanc_stock14	pearson	two-sided	481	0.994686	[0.99, 1.0]	0.0	inf	1.0
...	...	...	...	...	...	...	...	...	...	...
1171	distanc_stock6	distanc_stock8	pearson	two-sided	481	0.996059	[1.0, 1.0]	0.0	inf	1.0

Clear (or rather unclear since values are very close) winner of this are

1. stock1 stock10 pearson two-sided 481 0.994328 [0.99, 1.0]
2. stock1 stock11 pearson two-sided 481 0.985677 [0.98, 0.99]
3. stock1 stock12 pearson two-sided 481 0.994507 [0.99, 1.0]
4. stock1 stock13 pearson two-sided 481 0.993323 [0.99, 0.99]
5. stock1 stock14 pearson two-sided 481 0.994686

Thus we find that these stock have the highest level opf correlation among themselves.



Problem 3c

c. Construct one “average” or “consensus” record for each stock. Now cluster the stocks using a clustering algorithm like k-means. How many big clusters do you find? Create a TSNE-plot of the stocks where you color each stock according to its cluster ID. Do the colors seem visually coherent to you or not?

I used to approaches to cluster the stocks.

Approach 1 Correlation Clustering of Auto correlatrion Matrix

1. In this approach, we use the already calculated stockwise coorelation array. It contaisn an array of autocorrelation with lag 0,10 , (i.e it includes reference_price as well) .

- We use this to compute correlation between the trajectories of autocorrelation, i.e stocks behaving in a similar manner.
- We are using Euclidean-distance to measure autocorrelation between auto correlation values of each stock.
- This gives us a nice correlation matrix which we can visualise in the first matrix.
- But we can do better we see that stock sae not cluster ed based on correlation , if we can cluster this corelation matrix we can get a decent idea of highly correlated stocks on one ned and lessser correlated stock on the other end. We do this by using cluster_correlation function which can be used to rearrange columsn in correlation matrix to give a better picture of relatively highly and lowly correlated stocks

Approach 2 K-means clustering

I used this to simply cluster `refernce` based on centroids we assumed 5 clusters of the stocks (still working with 50 stocks).

- Kmeans usually works well for low dimenisonal data (<=2). this means we have to reduce the datewise dimension of the stock to a single value for each stocks for a span of 481 days.
- We did this by defining a consesnus function based on floowing criteria: We took multipel moving averages , for each stock (5, 10 15, 30 days), we avergaed them up , scaled them and averaged it with the overall mean of the data. val=((moving averages of all 4 period)*scaling factor+ overall average)/2

This gives us a siginificant comparable value for each stock of the same order, as I found that ohterwise most of the data was going in the order < 10^-4. One of the reason being that the data points itself is percentage basis ppoints so is 100 times smaller.

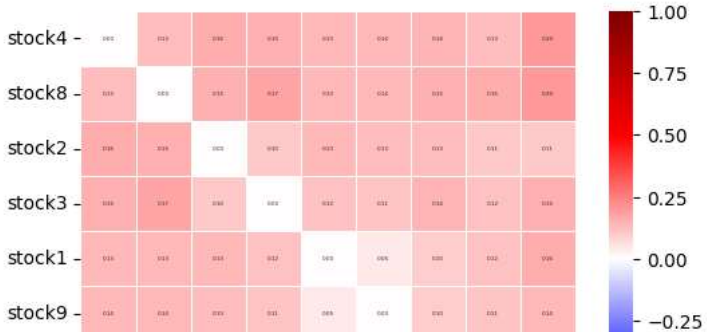
Approach 1 Correlation Clustering of Auto correlatrion Matrix

```
# from scipy.spatial.distance import directed_hausdorff
# ##

autocorr_df
stock_corr_dist=pd.DataFrame(index=autocorr_df.columns ,columns=autocorr_df.columns,dtype=float)
# ax=sns.PairGrid(autocorr_df)
plt.show()
for columnI in autocorr_df.items():
  column_i_nm, column_i=columnI[0], columnI[1]
  for columnJ in autocorr_df.items():
    column_j_nm, column_j=columnJ[0], columnJ[1]
    stock_corr_dist[column_i_nm][column_j_nm] =np.linalg.norm(column_i-column_j)

# stock_corr_dist.info()
# print(type(stock_corr_dist[stock_corr_dist.index[0]][stock_corr_dist.index[0]]))



# print(stock_corr_dist.mean().mean())
# print(stock_corr_dist.max())
# stock_corr_dist

fig, ax = plt.subplots(figsize=(15,15))
corr_hmap = sns.heatmap(stock_corr_dist, vmin=-1, vmax=1,cmap='seismic', linewidths=0.5, fmt='.3f', annot=True,square=True) # time toplot for
plt.show()
# plt.savefig("corr_matrix_incl_anno_double.jpeg", dpi=300)
```


```
stock_corr_dist.shape
```

```
import scipy.cluster.hierarchy as sch

def cluster_corr(corr_array, inplace=False):
    """
    Rearranges the correlation matrix, corr_array, so that groups of highly
    correlated variables are next to eachother

    Parameters
    ----------
    corr_array : pandas.DataFrame or numpy.ndarray
        a NxN correlation matrix

    Returns
    -------
    pandas.DataFrame or numpy.ndarray
        a NxN correlation matrix with the columns and rows rearranged
    """
    pairwise_distances = sch.distance.pdist(corr_array)
    linkage = sch.linkage(pairwise_distances, method='complete')
    cluster_distance_threshold = pairwise_distances.max()/2
    idx_to_cluster_array = sch.fcluster(linkage, cluster_distance_threshold,
                                        criterion='distance')
    idx = np.argsort(idx_to_cluster_array)

    if not inplace:
        corr_array = corr_array.copy()

    if isinstance(corr_array, pd.DataFrame):
        return corr_array.iloc[idx, :].T.iloc[idx, :]
    return corr_array[idx, :][:, idx]

sns.heatmap(cluster_corr(corr_array=stock_corr_dist),vmin=-1, vmax=1,cmap='seismic', linewidths=0.5, fmt='.2f', annot_kws={'size': 3},annot=T
```

```
<Axes: >
```




```
q3b1
```

	distanc_stock1	distanc_stock10	distanc_stock11	distanc_stock12	distanc_stock13	distanc_stock14	distanc_stock15	distanc_stock
0	0.018190	0.018211	0.018198	0.018234	0.018154	0.018224	0.018210	0.0182
1	0.000026	-0.000025	-0.000024	-0.000036	0.000030	-0.000012	-0.000042	0.0001
2	-0.000095	-0.000051	0.000163	-0.000075	-0.000051	-0.000011	0.000048	-0.0001
3	0.000111	0.000081	0.000040	0.000151	0.000081	-0.000002	0.000009	0.0000
4	-0.000034	-0.000013	-0.000263	-0.000087	-0.000029	0.000012	-0.000004	-0.0000
...	...	...	...	...	...	...	...	
476	-0.000043	-0.000039	0.000006	-0.000037	0.000054	-0.000010	0.000037	-0.0000
477	0.000054	0.000014	-0.000042	0.000067	-0.000041	0.000013	0.000035	0.0000
478	0.000043	0.000030	0.000018	-0.000042	0.000083	-0.000005	0.000012	0.0000
479	-0.000030	-0.000059	-0.000005	-0.000016	-0.000031	0.000023	-0.000051	-0.0000
480	-0.000030	0.000044	-0.000129	0.000003	-0.000051	-0.000008	-0.000051	0.0000

481 rows × 49 columns

Approach 2 K-means clustering

```
q3c1=q3b1.copy()
# print(q3c1)
# print(q3c1)

q3c4= pd.DataFrame(index=q3c1.T.index)
scale=100

for (columnName, columnData) in q3c1.T.items():
    mn1=np.mean(q3c1,axis=0)
    q3c4['mean']=mn1

# print(q3c101)
q3c3=pd.DataFrame()
roll_avg=[5,10,15,30]
rm_avg=[]
for (columnName, columnData) in q3c1.items():
    # print('Column Name : ', columnName)
    # print('Column Contents : ', len(columnData.values))
    val=0
    # mn1=np.mean(q3c1,axis=0)
    for rm in roll_avg:
      col_nm='roll_avg'+str(rm)
      q3c3[col_nm]=columnData.rolling(rm).mean()
      val=np.mean(q3c3[col_nm],axis=0)
      val+=val
    rm_avg=np.append(rm_avg,val/4)

print(len(rm_avg))

#     mn1=mn1/5
#     print(mn1)
```

```
#     q3c101['mean']=mn1


q3c3.fillna(0,inplace=True)
# print(q3c3)
# q3c101['mean']=np.mean(q3c3,axis=1)

q3c4['rm_avg']=rm_avg*10000
q3c4['avg']=(q3c4['rm_avg']+q3c4['mean'])/2
q3c4['stock_id']=q3c4.index
q3c4['stock_id']=q3c4['stock_id'].apply(lambda x: float(x.replace("distanc_stock","")))

q3c4.head(10)
```

49

	mean	rm_avg	avg	stock_id
distanc_stock1	0.000038	0.006409	0.003224	1.0
distanc_stock10	0.000038	0.006623	0.003330	10.0
distanc_stock11	0.000038	0.006850	0.003444	11.0
distanc_stock12	0.000038	0.006710	0.003374	12.0
distanc_stock13	0.000038	0.006644	0.003341	13.0
distanc_stock14	0.000038	0.006385	0.003211	14.0
distanc_stock15	0.000038	0.006664	0.003351	15.0
distanc_stock16	0.000038	0.006558	0.003298	16.0
distanc_stock17	0.000038	0.006528	0.003283	17.0
distanc_stock18	0.000038	0.006312	0.003175	18.0

```
# from kneed import KneeLocator
# from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
from matplotlib.lines import Line2D

q3c101=pd.DataFrame()
q3c101=q3c4[['avg','stock_id']]
q3c101=q3c101.fillna(0)
kmeans = KMeans(n_clusters=5, random_state=0)
q3c101['cluster'] = kmeans.fit_predict(q3c101)
df=q3c101

# get centroids
centroids = kmeans.cluster_centers_
cen_x = [i[0] for i in centroids]
cen_y = [i[1] for i in centroids]
## add to df
df['cen_x'] = df.cluster.map({0:cen_x[0], 1:cen_x[1], 2:cen_x[2]})
df['cen_y'] = df.cluster.map({0:cen_y[0], 1:cen_y[1], 2:cen_y[2]})
# define and map colors
colors = ['#DF2020', '#81DF20', '#2095DF','#0000FF','#FFA500']
df['c'] = df.cluster.map({0:colors[0], 1:colors[1], 2:colors[2], 3:colors[3], 4:colors[4]})
df=df.fillna(0)

df = df.sample(frac=1).reset_index(drop=True)
df
```

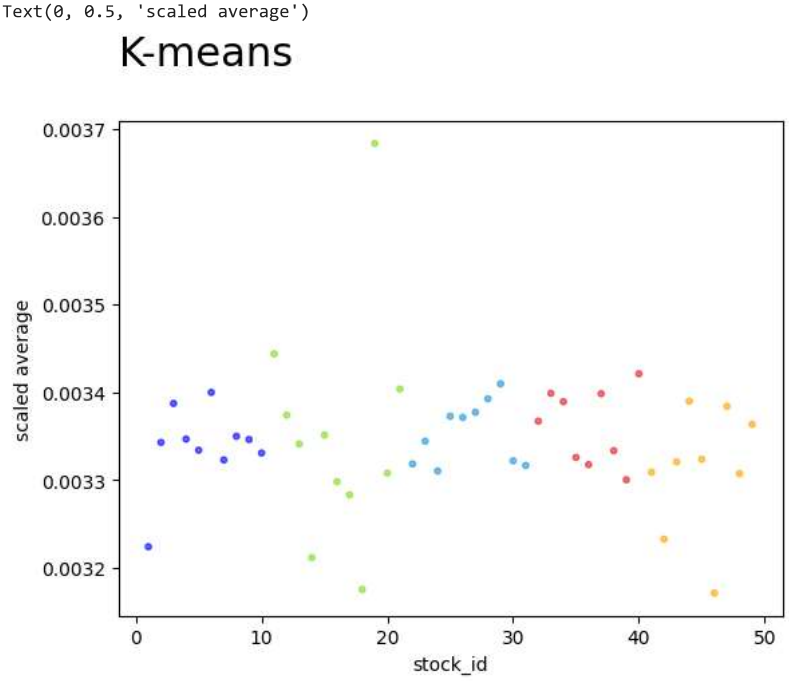
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
```

	avg	stock_id	cluster	cen_x	cen_y	c
0	0.003309	41.0	4	0.000000	0.0	#FFA500
1	0.003323	7.0	3	0.000000	0.0	#0000FF
2	0.003308	20.0	1	0.003352	16.0	#81DF20
3	0.003346	9.0	3	0.000000	0.0	#0000FF
4	0.003350	8.0	3	0.000000	0.0	#0000FF
5	0.003392	28.0	2	0.003353	26.5	#2095DF
6	0.003211	14.0	1	0.003352	16.0	#81DF20
7	0.003307	48.0	4	0.000000	0.0	#FFA500
8	0.003374	12.0	1	0.003352	16.0	#81DF20
9	0.003400	6.0	3	0.000000	0.0	#0000FF
10	0.003283	17.0	1	0.003352	16.0	#81DF20
11	0.003320	43.0	4	0.000000	0.0	#FFA500
12	0.003316	31.0	2	0.003353	26.5	#2095DF
13	0.003444	11.0	1	0.003352	16.0	#81DF20
14	0.003310	24.0	2	0.003353	26.5	#2095DF
15	0.003387	3.0	3	0.000000	0.0	#0000FF
16	0.003322	30.0	2	0.003353	26.5	#2095DF
17	0.003398	37.0	0	0.003361	36.0	#DF2020
18	0.003371	26.0	2	0.003353	26.5	#2095DF
19	0.003325	35.0	0	0.003361	36.0	#DF2020
20	0.003372	25.0	2	0.003353	26.5	#2095DF
21	0.003351	15.0	1	0.003352	16.0	#81DF20
22	0.003298	16.0	1	0.003352	16.0	#81DF20
23	0.003330	10.0	3	0.000000	0.0	#0000FF
24	0.003224	1.0	3	0.000000	0.0	#0000FF
25	0.003363	49.0	4	0.000000	0.0	#FFA500
26	0.003399	33.0	0	0.003361	36.0	#DF2020
27	0.003344	23.0	2	0.003353	26.5	#2095DF
28	0.003403	21.0	1	0.003352	16.0	#81DF20
29	0.003389	34.0	0	0.003361	36.0	#DF2020
30	0.003175	18.0	1	0.003352	16.0	#81DF20
31	0.003334	5.0	3	0.000000	0.0	#0000FF
32	0.003346	4.0	3	0.000000	0.0	#0000FF
33	0.003409	29.0	2	0.003353	26.5	#2095DF
34	0.003333	38.0	0	0.003361	36.0	#DF2020
35	0.003384	47.0	4	0.000000	0.0	#FFA500
36	0.003171	46.0	4	0.000000	0.0	#FFA500
37	0.003300	39.0	0	0.003361	36.0	#DF2020
38	0.003318	22.0	2	0.003353	26.5	#2095DF
39	0.003377	27.0	2	0.003353	26.5	#2095DF
40	0.003367	32.0	0	0.003361	36.0	#DF2020
41	0.003390	44.0	4	0.000000	0.0	#FFA500

```
plt.scatter(df['stock_id'], df['avg'], c=df.c, alpha = 0.6, s=10)

plt.title('K-means\n', loc='left', fontsize=22)
```

```
plt.xlabel('stock_id')
plt.ylabel('scaled average')
```

```
Text(0, 0.5, 'scaled average')
```




```
#### Q4: Closing trajectory of stocks on each day highly correlated, 2
- a. Make three plots, 10 pts
- b. permutation test for statistical confidence, p-value, 15 pts

Is the closing trajectory of stocks on each day highly correlated (“th
up days and
down days in the market”) or is it essentially random (say, “supply an
cause
distinct fluctuations on individual stocks each day”)?


a. Make three plots that convince you of which is the right answer, an
convince
me as well. If there is a formal statistical test to help you do this,
(10 points) <br><br>


Plot Used:
1. Distance plots of 50 stocks for all 481 days
2. Reefrence_proce movement plot  of 50 stocks for all 481 days
3. Cluster correlation matrix for 3 plots based on autocorrelation mat
mentioned in 3c

Through all thiese plot we pobserver 2 thinhgs . There is a definite c
nature in th estocks , some aremore close rto another , the correlatio
helps us identify the c;loseness of th ecorrelation .
Through below analysis we find the most correlated ones.
```

Q4: Closing trajectory of stocks on each day highly correlated, 25 pts

- a. Make three plots, 10 pts
- b. permutation test for statistical confidence, p-value, 15 pts

Is the closing trajectory of stocks on each day highly correlated (“there are up days and down days in the market”) or is it essentially random (say, “supply and demand cause distinct fluctuations on individual stocks each day”)?

a. Make three plots that convince you of which is the right answer, and will convince me as well. If there is a formal statistical test to help you do this, do it. (10 points)

Plot Used:

1. Distance plots of 50 stocks for all 481 days
2. Reefrence_proce movement plot of 50 stocks for all 481 days
3. Cluster correlation matrix for 3 plots based on autocorrelation matrix as mentioned in 3c

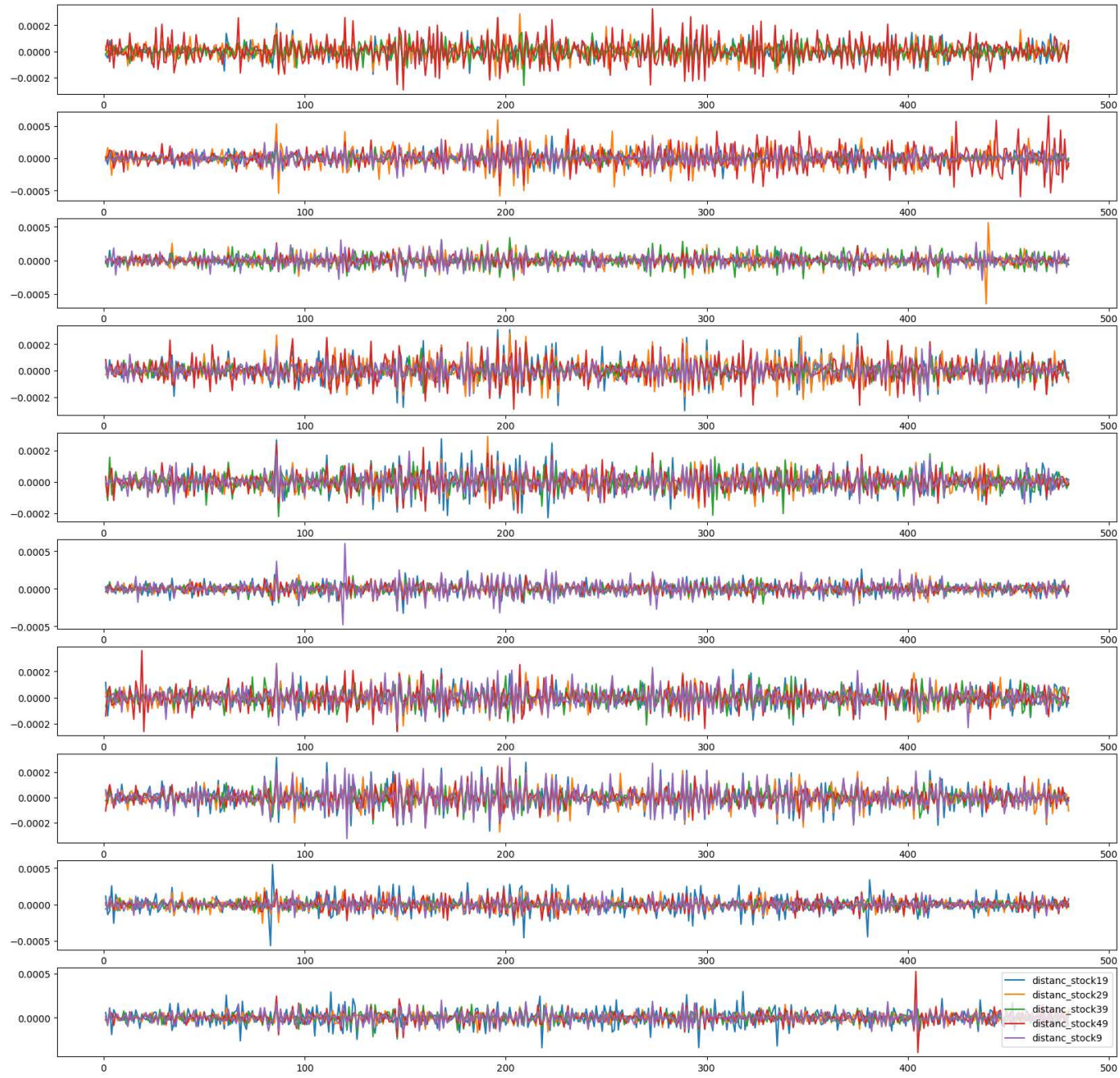
Through all thiese plot we pobserver 2 thinhgs . There is a definite cyclical nature in th estocks , some aremore close rto another , the correlation matrix helps us identify the c;loseness of th ecorrelation . Through below analysis we find the most correlated ones.

Double-click (or enter) to edit

```
q4p1=q3b1
q4p1=q4p1.drop(index=0)

fig, ax = plt.subplots(10,1,figsize=(20,20))

for (columnName, columnData) in q4p1.items():
```

```
    idx=int(columnName.replace("distanc_stock",""))
    ax[idx%10].plot(q4p1.index, columnData, label = columnName)
plt.legend()
```

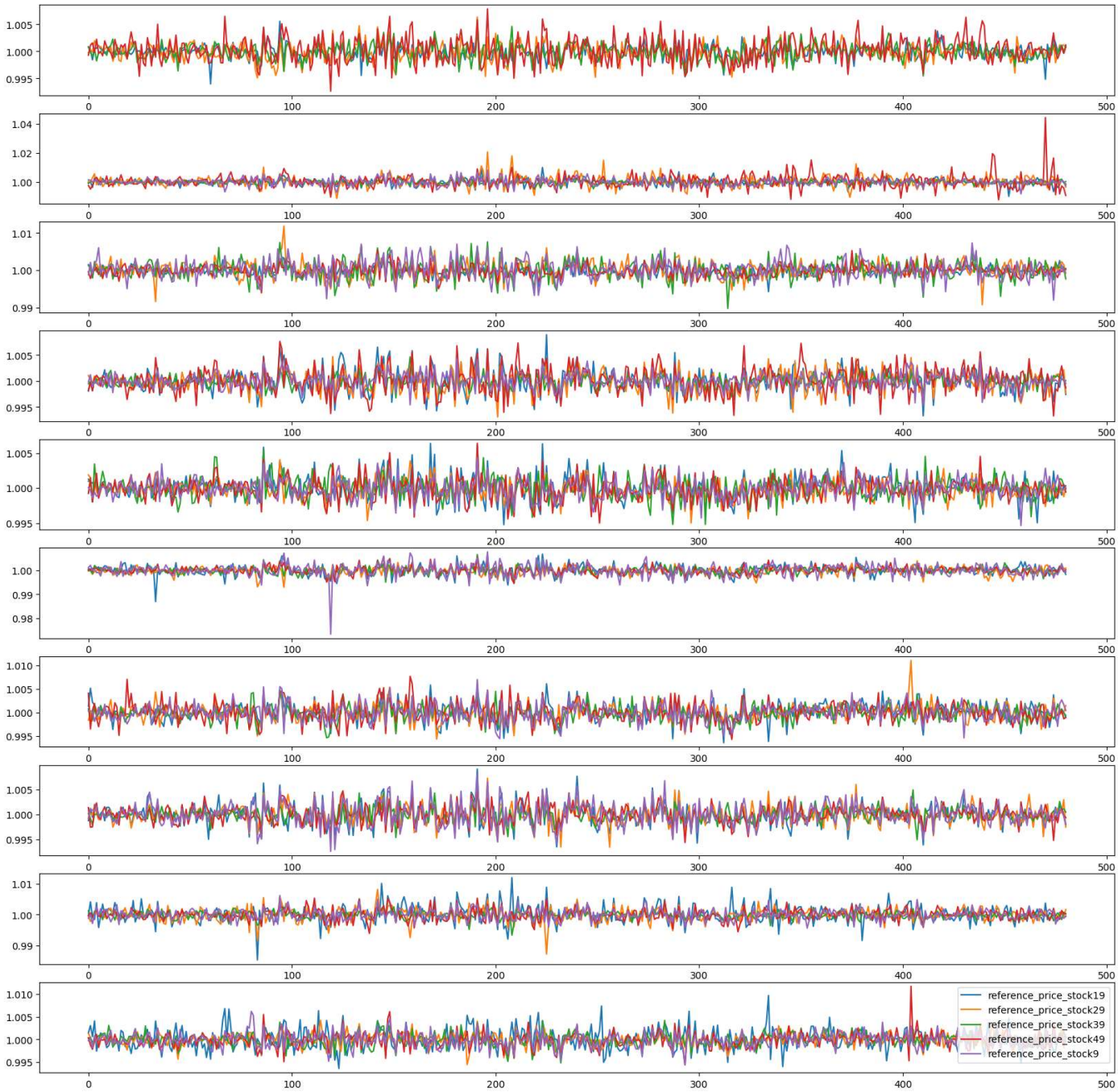
```
<matplotlib.legend.Legend at 0x7b056ead0d60>
```




```
q4p2=q3b2
# q4p2=q4p2.drop(index=0)
fig, ax = plt.subplots(10,1,figsize=(20,20))

for (columnName, columnData) in q4p2.items():
```

```
    idx=int(columnName.replace("reference_price_stock",""))
    ax[idx%10].plot(q4p2.index, columnData, label = columnName)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b056e714370>
```



b. Perform a permutation test to determine the statistical confidence that you believe your answer. In particular, for each stock randomly permute the day index to construct an artificial time series for it. Now measure the consistency of daily performance in this permuted data set. Run enough permutations per variable to establish a p-value of how confident you are in your conclusion (15 points)

```
q4b1=q3b1
q4b1
```

	distanc_stock1	distanc_stock10	distanc_stock11	distanc_stock12	distanc_stock13	distanc_stock14	distanc_stock15	distanc_stock
0	0.000080	-0.000122	-0.000051	-0.000012	-0.000058	-0.000037	0.000042	-0.0000
1	-0.000016	-0.000008	0.000018	-0.000012	0.000057	0.000017	0.000054	-0.0000
2	-0.000145	-0.000038	0.000096	-0.000014	-0.000064	-0.000010	0.000018	-0.0000
3	-0.000098	0.000042	0.000085	0.000002	-0.000072	0.000029	-0.000045	-0.0000
4	-0.000010	-0.000063	0.000083	-0.000004	0.000038	-0.000024	-0.000121	-0.0001
...	...	...	...	...	...	...	...	
476	-0.000044	0.000064	0.000202	0.000071	0.000053	-0.000004	-0.000111	-0.0000
477	0.000136	-0.000015	0.000051	-0.000006	0.000310	0.000106	0.000033	0.0001
478	0.000194	0.000115	-0.000081	0.000056	0.000060	0.000074	0.000027	0.0000
479	0.000056	-0.000004	0.000122	0.000069	0.000017	0.000091	0.000138	0.0000
480	-0.000190	-0.000066	-0.000198	-0.000113	-0.000102	-0.000096	-0.000090	-0.0000

481 rows × 49 columns

```
for  i in range (1,4):
    q4b2=q4b1.sample(frac=(1/i),ignore_index=True)
    q4b2
    pairwise_reln2=pg.pairwise_corr(q4b2, method='pearson')
    pairwise_reln2.sort_values(by=['r']) # Sorting by r values
    print(pairwise_reln2.head(5))
```

```
                X               Y   method alternative    n         r  \
0  distanc_stock1  distanc_stock10  pearson   two-sided  481  0.994328
1  distanc_stock1  distanc_stock11  pearson   two-sided  481  0.985677
2  distanc_stock1  distanc_stock12  pearson   two-sided  481  0.994507
3  distanc_stock1  distanc_stock13  pearson   two-sided  481  0.993323
4  distanc_stock1  distanc_stock14  pearson   two-sided  481  0.994686

          CI95%  p-unc BF10  power
0   [0.99, 1.0]    0.0  inf    1.0
1  [0.98, 0.99]    0.0  inf    1.0
2   [0.99, 1.0]    0.0  inf    1.0
3  [0.99, 0.99]    0.0  inf    1.0
4   [0.99, 1.0]    0.0  inf    1.0
                X               Y   method alternative    n         r  \
0  distanc_stock1  distanc_stock10  pearson   two-sided  240  0.997301
1  distanc_stock1  distanc_stock11  pearson   two-sided  240  0.992259
2  distanc_stock1  distanc_stock12  pearson   two-sided  240  0.997383
3  distanc_stock1  distanc_stock13  pearson   two-sided  240  0.996496
4  distanc_stock1  distanc_stock14  pearson   two-sided  240  0.997520

          CI95%          p-unc BF10  power
0    [1.0, 1.0]  6.002284e-272  nan    1.0
1  [0.99, 0.99]  1.269436e-217  nan    1.0
2    [1.0, 1.0]  1.531493e-273  nan    1.0
3    [1.0, 1.0]  1.781732e-258  nan    1.0
4    [1.0, 1.0]  2.620832e-276  nan    1.0
                X               Y   method alternative    n         r  \
0  distanc_stock1  distanc_stock10  pearson   two-sided  160  0.998008
1  distanc_stock1  distanc_stock11  pearson   two-sided  160  0.995869
2  distanc_stock1  distanc_stock12  pearson   two-sided  160  0.998130
3  distanc_stock1  distanc_stock13  pearson   two-sided  160  0.997812
4  distanc_stock1  distanc_stock14  pearson   two-sided  160  0.998074

          CI95%          p-unc        BF10  power
0   [1.0, 1.0]  1.573875e-191  1.634e+186    1.0
1  [0.99, 1.0]  1.511981e-166  3.531e+161    1.0
2   [1.0, 1.0]  1.049880e-193  2.299e+188    1.0
3   [1.0, 1.0]  2.593453e-188   1.09e+183    1.0
4   [1.0, 1.0]  1.087002e-192  2.288e+187    1.0
```

Even after repeated permutations we see that the correlation stands out automatically .

1. stock1 stock10 pearson two-sided 481 0.994328 [0.99, 1.0]
2. stock1 stock11 pearson two-sided 481 0.985677 [0.98, 0.99]
3. stock1 stock12 pearson two-sided 481 0.994507 [0.99, 1.0]
4. stock1 stock13 pearson two-sided 481 0.993323 [0.99, 0.99]
5. stock1 stock14 pearson two-sided 481 0.994686 [0.99, 1.0]

Here also the winner are more or less the same .This validates our previous result based on correlation of autocorrelation trajectories.

Double-click (or enter) to edit

Q5: Best prediction model, any approaches, 25 pts

Finally, build the best prediction model you can to solve the Kaggle task. Use any data, ideas, and approach that you like, but describe all the approaches you try. For each model you try, report the average absolute error using 5-fold cross-validation. (20 points)

In this we try to implement various models on the Data to predict the `target` value.

We do this by splitting the data set into 80:20 train:test format.

I ran the following algorithms

1. Linear regression
2. Random Forest regression
3. Gradient Boost Method _ implemented but the model was taking too long for significant amoutn of data even after reducing max_depth

---

Standard Approaches to increase performance

1. Add More Data. - can't as we have enough , instead we need to qualitatively increase and quantitatively reduce data here.
2. Treat Missing and Outlier Values. applied in some capacity
3. Feature Engineering. applied in some capacity
4. Feature Selection. applied
5. Multiple Algorithms.
6. Algorithm Tuning.
7. Ensemble Methods. applied
8. Cross Validation.

```
'''
'stock_id', 'date_id', 'seconds_in_bucket', 'imbalance_size', 'imbalance_buy_sell_flag', 'reference_price', 'matched_size',
       'far_price', 'near_price', 'bid_price', 'bid_size', 'ask_price',
       'ask_size', 'wap', 'target', 'time_id', 'row_id'
'''

# fetures_1= reference_price
# f2=wap
# f3 = imbalance_size * imbalance_buy_sell_flag
# f4
basic_cols=['row_id','stock_id', 'seconds_in_bucket','date_id','reference_price','wap','target']
i1data = mdfd_csv[mdfd_csv['far_price'].notna()]
idata=i1data[basic_cols]

idata['imbalance']=mdfd_csv['imbalance_buy_sell_flag']*mdfd_csv['imbalance_size']

idata=idata.fillna(0)
x_cols=['row_id','stock_id','seconds_in_bucket', 'date_id','reference_price','wap','imbalance']
x_data=idata[x_cols]

y_cols=['target']
y_data=idata[y_cols]
```

```
<ipython-input-47-3f88bb19c21c>:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  idata['imbalance']=mdfd_csv['imbalance_buy_sell_flag']*mdfd_csv['imbalance_size']
```

```
from statsmodels.tsa.stattools import adfuller


def adfuller_test(sales):
    result=adfuller(sales)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

adfuller_test(idata['reference_price'])


from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble._forest import RandomForestRegressor

result_cols=['Model','Accuracy','Mean Squared Error']
final_result=pd.DataFrame(columns=result_cols)
final_result.set_index('Model')


def linear_reg_flow(x,y):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
    # print("X_train:",x_train.shape)
    # print("X_test:",x_test.shape)
    # print("Y_train:",y_train.shape)
    # print("Y_test:",y_test.shape)
    linreg=LinearRegression()
    linreg.fit(x_train,y_train)
    y_pred=linreg.predict(x_test)
    # print(y_pred)
    Accuracy=r2_score(y_test,y_pred)*100
    # print(" Accuracy based on r2_score of the model is %.2f" %Accuracy)
    r_squared = linreg.score(x, y)
    #view R-squared value
    # print("Rsquared default of the model is %.2f" %r_squared)
    msqe=mean_squared_error(y_test, y_pred, squared=True)
    # print(" mean_squared_error of the model is %.2f" %msqe)
    result={}
    result['accuracy_r2_score']=Accuracy
    result['r-squared']=r_squared
    result['mean_squared_error']=msqe
    final_result.loc['Linear regression']=['Linear regression',Accuracy,msqe]
    # print(pd.DataFrame.from_dict(result))
    print("Linear Regression:: "+str(result))
    return linreg


def random_forest_flow(x,y):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
    regressor = RandomForestRegressor(n_estimators=5,  random_state=0)
    # fit the regressor with x and y data
    regressor.fit(x_train,y_train.values.ravel())
    # test the output by changing values
    y_pred = regressor.predict(x_test)
    Accuracy=r2_score(y_test,y_pred)*100
    print(" Random Forest Regressor:: Accuracy based on r2_score of the model is %.2f" %Accuracy)
    msqe=mean_squared_error(y_test, y_pred, squared=True)
    print(" mean_squared_error of the model is %.2f" %msqe)
    final_result.loc['Random Forest']=['Random Forest',Accuracy,msqe]
    #view R-squared value
    # print("Rsquared default of the model is %.2f" %r_squared)
    # print(" Out of bag score of the model is %.2f" %regressor.oob_score_)
    # print(regressor.oob_score_)
```

```
    return regressor


def GBM_Regressor_flow(x,y):
    X_train, X_test, y_train, y_test = train_test_split(x,y, test_size=0.20, random_state=100)
    model = GradientBoostingRegressor(n_estimators=1000,criterion='friedman_mse', max_depth=4,min_samples_split=5,
                                      min_samples_leaf=5,max_features=3)

    y_train1=np.ravel(y_train)
    model.fit(X_train,y_train1)
    y_pred = model.predict(X_test)
    # print(y_pred.shape)
    Accuracy=r2_score(y_test,y_pred)*100
    print(" GBM_Regressor_flow:: Accuracy based on r2_score of the model is %.2f" %Accuracy)
    msqe=mean_squared_error(y_test, y_pred, squared=True)
    print(" mean_squared_error of the model is %.2f" %msqe)
    final_result.loc['GBM Regressor']=['GBM Regressor',Accuracy,msqe]


# linear_reg_flow(x_data,y_data)
# # GBM_Regressor_flow(x_data,y_data)
# random_forest_flow(x_data,y_data)
# model=LinearRegression()
model1 = linear_reg_flow(x_data,y_data)
# evaluate model
model2=random_forest_flow(x_data,y_data)
```

```
Linear Regression:: {'accuracy_r2_score': 1.4498826920064767, 'r-squared': 0.014526701051261526, 'mean_squared_error': 69.26479790440112
 Random Forest Regressor:: Accuracy based on r2_score of the model is 12.89
 mean_squared_error of the model is 61.23
```

A clear score improvemt is seen in ncase of random forest however at the cost of performance. as the no row increases , random forest tends to get bigger and bigger due increasing forest sizes. This creates a huge iverheade. I Managed to run this by limiting the n_estimators and reducing the size of the row on which it worked .

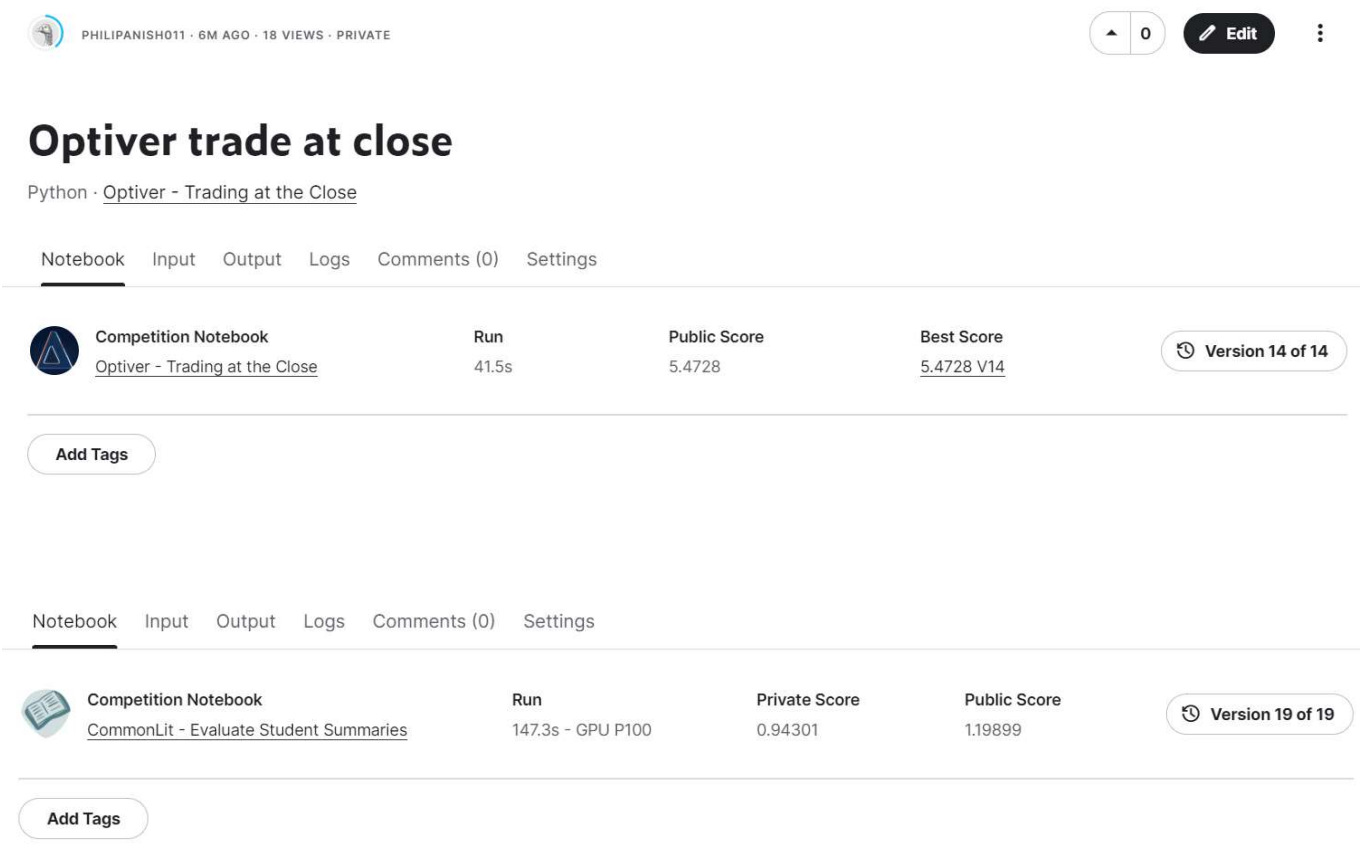
We also ran cross-validation with random forest Cross validation and generally validation model techniques are used not only to avoid overfitting (rarely the case when using linear models) but also when there are different models to compare.

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold


cv = KFold(n_splits=5, random_state=1, shuffle=True)
model=RandomForestRegressor(n_estimators=5,  random_state=0)

scores = cross_val_score(model, x_data, y_data, scoring='accuracy', cv=cv, n_jobs=-1)
scores
```

```
array([nan, nan, nan, nan, nan])
```

Q6: submit model on Kaggle, 0 pts

Public Score: 5.4728 Private Score:
Kaggle profile link: https://www.kaggle.com/code/philipanish011 Screenshot(s): \attached below



References:

- https://www.nyse.com/network/article/nyse-closing-auction
- https://www.nyse.com/article/nyse-closing-auction-insiders-guide
- https://www.investopedia.com/articles/investing/091113/auction-method-how-nyse-stock-prices-are-set.asp
- https://www.investopedia.com/terms/a/at-the-close-order.asp
- https://www.kaggle.com/code/tomforbes/optiver-trading-at-the-close-introduction
- https://neptune.ai/blog/time-series-prediction-vs-machine-learning
- https://towardsdatascience.com/ml-approaches-for-time-series-4d44722e48fe (blog post)
- https://scholar.google.com/scholar?as_ylo=2022&q=predictions+time-series&hl=en&as_sdt=0,33 (advanced research papers)
- https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-to-time-series-analysis/
- [#BEGIN[ChatGPT BingChat])]-This homework will investigate data integration and model building in IPython. It is based on the
- Optiver Trading at the Close Kaggel challenge . Can you identify any patterns in the datasets
- [#BEGIN[ChatGPT BingChat])] -can you help me identify tha way to calculate distance function for these stocks
- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html
- https://www.educative.io/answers/what-is-autocorrelation-in-python
- https://seaborn.pydata.org/generated/seaborn.heatmap.html
- https://sparkbyexamples.com/pandas/pandas-dataframe-isna-function/
- https://www.statology.org/pandas-value_counts-sort/
- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.groupby.html
- https://pandas.pydata.org/docs/user_guide/groupby.html
- https://archive.ph/vstV9
- https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
- https://re-thought.com/how-to-add-new-columns-in-a-dataframe-in-pandas/
- https://www.investopedia.com/terms/a/autocorrelation.asp
- https://stackoverflow.com/questions/37794849/efficient-and-precise-calculation-of-the-euclidean-distance

- https://stackoverflow.com/questions/26716616/convert-a-pandas-dataframe-to-a-dictionary
- https://wil.yegelwel.com/cluster-correlation-matrix/