

✓ premise: study underlying model dynamics

challenge assumptions on self-factor causality in prices

install

Double-click (or enter) to edit

install critical libraries to the underlying os

```
!pip3 install altair
!pip3 install altair-viewer
!pip3 install -U altair_viewer
!pip3 install statsmodels
!pip3 install imblearn

Requirement already satisfied: altair in /usr/local/lib/python3.10/dist-packages (4.2.2)
Requirement already satisfied: entrypoints in /usr/local/lib/python3.10/dist-packages (from altair) (0.4)
Requirement already satisfied: Jinja2 in /usr/local/lib/python3.10/dist-packages (from altair) (3.1.2)
Requirement already satisfied: jsonschema>=3.0 in /usr/local/lib/python3.10/dist-packages (from altair) (4.19.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from altair) (1.23.5)
Requirement already satisfied: pandas>=0.18 in /usr/local/lib/python3.10/dist-packages (from altair) (1.5.3)
Requirement already satisfied: toolz in /usr/local/lib/python3.10/dist-packages (from altair) (0.12.0)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair) (2023.12.1)
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair) (0.35.0)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair) (0.11.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.18->altair) (2.8.1)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.18->altair) (2023.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2->altair) (2.1.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas>=0.18->altair) (1.16.0)
Collecting altair-viewer
  Downloading altair_viewer-0.4.0-py3-none-any.whl (844 kB)
    844.5/844.5 kB 5.0 MB/s eta 0:00:00
Requirement already satisfied: altair in /usr/local/lib/python3.10/dist-packages (from altair-viewer) (4.2.2)
Collecting altair-data-server>=0.4.0 (from altair-viewer)
  Downloading altair_data_server-0.4.1-py3-none-any.whl (12 kB)
Requirement already satisfied: portpicker in /usr/local/lib/python3.10/dist-packages (from altair-data-server>=0.4.0->altair-viewer) (1.5.2)
Requirement already satisfied: tornado in /usr/local/lib/python3.10/dist-packages (from altair-data-server>=0.4.0->altair-viewer) (6.2)
Requirement already satisfied: entrypoints in /usr/local/lib/python3.10/dist-packages (from altair->altair-viewer) (0.4)
Requirement already satisfied: Jinja2 in /usr/local/lib/python3.10/dist-packages (from altair->altair-viewer) (3.1.2)
Requirement already satisfied: jsonschema>=3.0 in /usr/local/lib/python3.10/dist-packages (from altair->altair-viewer) (4.19.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from altair->altair-viewer) (1.23.5)
Requirement already satisfied: pandas>=0.18 in /usr/local/lib/python3.10/dist-packages (from altair->altair-viewer) (1.5.3)
Requirement already satisfied: toolz in /usr/local/lib/python3.10/dist-packages (from altair->altair-viewer) (0.12.0)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair->altair-viewer) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair->altair-viewer) (2023.12.1)
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair->altair-viewer) (0.35.0)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair->altair-viewer) (0.11.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.18->altair->altair-viewer) (2.8.1)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.18->altair->altair-viewer) (2023.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2->altair->altair-viewer) (2.1.3)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from portpicker->altair-data-server>=0.4.0->altair-viewer) (5.9.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas>=0.18->altair->altair-viewer) (1.16.0)
Installing collected packages: altair-data-server, altair-viewer
Successfully installed altair-data-server-0.4.1 altair-viewer-0.4.0
Requirement already satisfied: altair_viewer in /usr/local/lib/python3.10/dist-packages (0.4.0)
Requirement already satisfied: altair in /usr/local/lib/python3.10/dist-packages (from altair_viewer) (4.2.2)
Requirement already satisfied: altair-data-server>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from altair_viewer) (0.4.1)
Requirement already satisfied: portpicker in /usr/local/lib/python3.10/dist-packages (from altair-data-server>=0.4.0->altair_viewer) (1.5.2)
Requirement already satisfied: tornado in /usr/local/lib/python3.10/dist-packages (from altair-data-server>=0.4.0->altair_viewer) (6.2)
Requirement already satisfied: entrypoints in /usr/local/lib/python3.10/dist-packages (from altair->altair_viewer) (0.4)
Requirement already satisfied: Jinja2 in /usr/local/lib/python3.10/dist-packages (from altair->altair_viewer) (3.1.2)
Requirement already satisfied: jsonschema>=3.0 in /usr/local/lib/python3.10/dist-packages (from altair->altair_viewer) (4.19.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from altair->altair_viewer) (1.23.5)
Requirement already satisfied: pandas>=0.18 in /usr/local/lib/python3.10/dist-packages (from altair->altair_viewer) (1.5.3)
Requirement already satisfied: toolz in /usr/local/lib/python3.10/dist-packages (from altair->altair_viewer) (0.12.0)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair->altair_viewer) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair->altair_viewer) (2023.12.1)
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair->altair_viewer) (0.35.0)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair->altair_viewer) (0.11.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.18->altair->altair_viewer) (2.8.1)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.18->altair->altair_viewer) (2023.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2->altair->altair_viewer) (2.1.3)
```

Here pip or the pip3 installations for altair, altair-viewer, -U altair viewer, statsmodels, and imblearn were successfully installed.

```
import pandas as pd
import altair as alt
import matplotlib.pyplot as plt #graphics --viz
from imblearn.over_sampling import ADASYN #synthetic minority oversampling
from sklearn.neighbors import KNeighborsClassifier #ML
from sklearn.preprocessing import StandardScaler #---
from statsmodels.tsa.api import VAR #granger causality
from statsmodels.tsa.vector_ar.var_model import VARResults, VARResultsWrapper
from sklearn.model_selection import train_test_split #TTS ,ML
from sklearn.metrics import accuracy_score #error analysis
from sklearn.metrics import multilabel_confusion_matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import classification_report
from scipy import stats
```

retrieve the data...

grab data from gh

Here we can see the different installs which will be used to retrieve data which will provide us with a data set.

```
#load up binary binned pipeline
url_m = 'https://raw.githubusercontent.com/stefanbund/py3100/main/binary_binned_pipeline.csv'
mdf = pd.read_csv(url_m) #make a pandas dataframe
mdf #matrix dataframe
```

	Unnamed: 0	group	time	s_MP	change	type	p_MP	precursor_buy_cap_pct_change	precursor
0	0	2	1.660222e+12	30.00	-5.333889e-04	precursor	29.99		-0.012510
1	1	4	1.660222e+12	29.83	-6.637375e-05	precursor	29.88		0.002322
2	2	6	1.660222e+12	29.92	-6.345915e-04	precursor	29.91		0.005934
3	3	8	1.660222e+12	29.90	-5.020193e-04	precursor	29.91		-0.002813
4	4	10	1.660223e+12	29.91	-1.469841e-03	precursor	29.90		-0.000211
...
6411	6411	12824	1.699042e+12	12.09	6.835784e-08	precursor	12.09		0.007670
6412	6412	12826	1.699043e+12	12.10	8.291806e-05	precursor	12.10		0.128049
6413	6413	12828	1.699044e+12	12.10	4.140439e-04	precursor	12.10		-0.005610
6414	6414	12830	1.699045e+12	12.18	-3.610930e-03	precursor	12.13		-0.003349
6415	6415	12832	1.699046e+12	12.14	-1.646768e-04	precursor	12.15		-0.003861

6416 rows x 39 columns

Here we have a data set from a result of data mining and inside of the data set we can see what is called a precursor, the statistics related to the precursor, and the surge, along with how much the price has surged. The precursor is the activity before the surge and the label at the end represents whether it was a good trade or not with one meaning it was a good trade.

variables associated

```
mdf.columns
```

```
Index(['Unnamed: 0', 'group', 'time', 's_MP', 'change', 'type', 'p_MP',
      'precursor_buy_cap_pct_change', 'precursor_ask_cap_pct_change',
      'precursor_bid_vol_pct_change', 'precursor_ask_vol_pct_change',
      'length', 'sum_change', 'max_surge_mp', 'min_surge_mp',
      'max_precursor_mp', 'min_precursor_mp', 'area', 'surge_targets_met_pct',
      'group.1', 'time.1', 's_MP.1', 'change.1', 'type.1', 'p_MP.1',
      'precursor_buy_cap_pct_change.1', 'precursor_ask_cap_pct_change.1',
      'precursor_bid_vol_pct_change.1', 'precursor_ask_vol_pct_change.1',
      'length.1', 'sum_change.1', 'max_surge_mp.1', 'min_surge_mp.1',
      'max_precursor_mp.1', 'min_precursor_mp.1', 'area.1', 'surge_area',
      'surge_targets_met_pct.1', 'label'],
      dtype='object')
```

Here the columns have to do with the data variables associated and the mdf or the matrix data frame.

✓ reliability of label

correct classification

what is the average "1" trade's value?

```
mdf[mdf['label']==1]['surge_targets_met_pct'].mean() #.74 or above

1.1650823181204584
```

Here when we have a label as one it means how much did the price surge or how many of the precursor price points were profitable, and the 1.1650823181204584 above means that 100% of the points were profitable. The label is only one when the search targets reach or are 0.74 and above.

```
ones = mdf[mdf['label']==1] #good trades
zeroes = mdf[mdf['label']==0] #baddies
```

Here we have two different data frames which are ones and zeros and ones are good trades, meanwhile zeros are bad trades.

study the underlying dichotomies in your data

```
ones.shape[0] # finger monkeys

119
```

The finger monkeys above represent the ones which are the good trades.

```
zeroes.shape[0] #evil monkeys

6297
```

The evil monkeys above represent the zeros or baddies, or the bad trades.

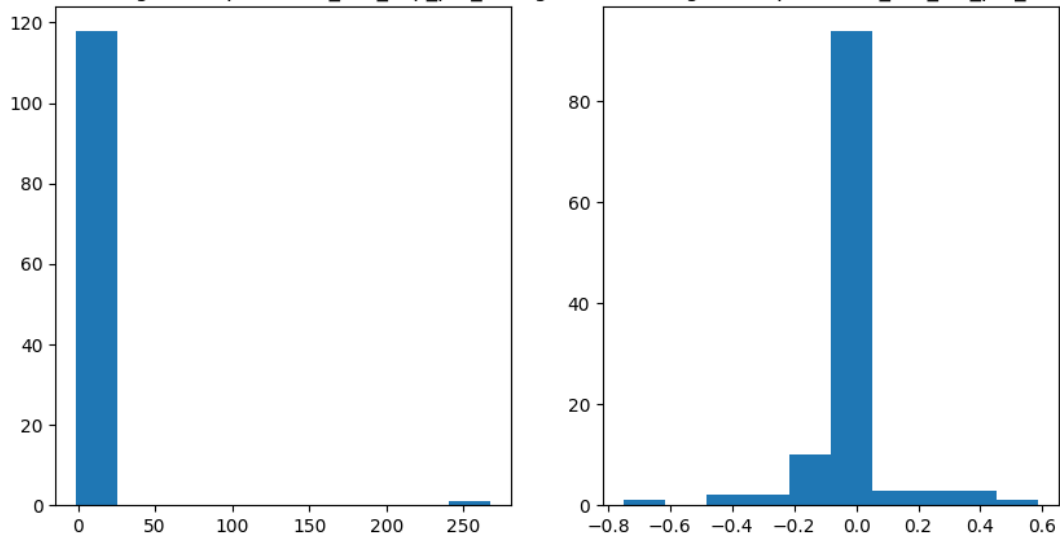
dimensions = features in the data we wish to study, before we classify

```
dimensions = ['precursor_ask_cap_pct_change', 'precursor_ask_vol_pct_change']
```

Here the ask is our cell orders and the cap is the capitalization which is the amount of money represented by the value of all the cell orders.

```
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].hist(ones['precursor_ask_cap_pct_change'], bins=10)
axs[0].set_title('ONES: Histogram of precursor_ask_cap_pct_change')
axs[1].hist(ones['precursor_ask_vol_pct_change'], bins=10)
axs[1].set_title('ONES: Histogram of precursor_ask_vol_pct_change')
plt.show()
```

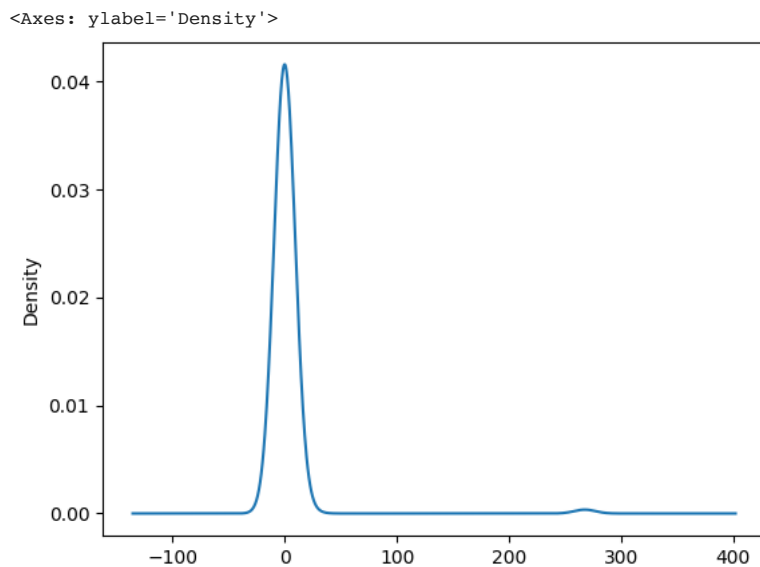
ONES: Histogram of precursor_ask_cap_pct_changeONES: Histogram of precursor_ask_vol_pct_change



On the left we can see the distribution of the ones or the good trades on the precursor ask capitalization percent change; and on the right we can the distribution of the ones on the precursor ask volume percent change.

✓ cap vs vol probability density, ONES

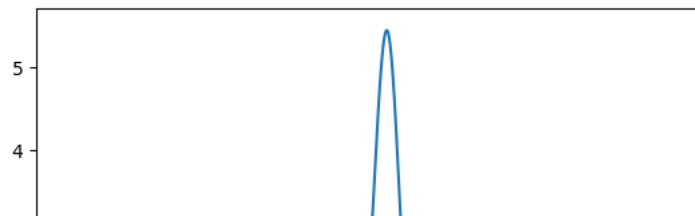
```
ones['precursor_ask_cap_pct_change'].plot.density() #precursor_ask_vol_pct_change
```



Here we can see the distribution of the density of ones or the good trades for the precursor ask capitalization percent change.

```
ones['precursor_ask_vol_pct_change'].plot.density() #precursor_ask_vol_pct_change
```

<Axes: ylabel='Density'>



Here we can see the distribution of the density of ones or the good trades for the precursor ask volume percent change.

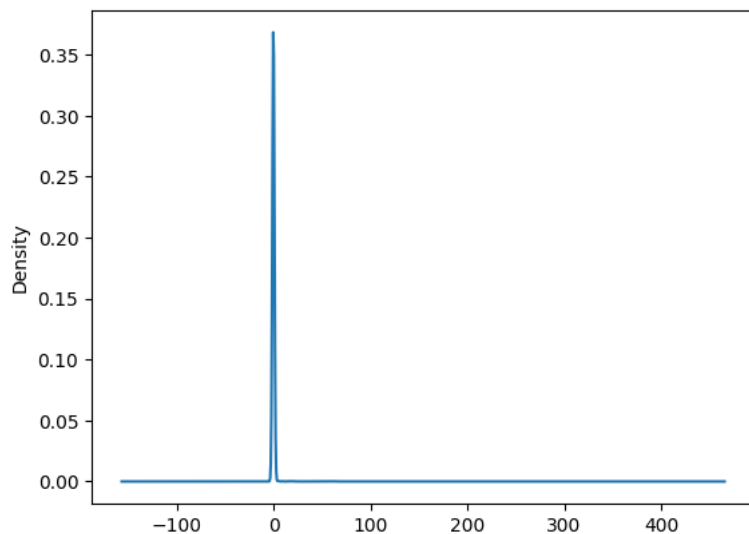
0 | | |

✓ cap vs vol probability density, ZEROES

| | |

```
zeroes['precursor_ask_cap_pct_change'].plot.density() #precursor_ask_vol_pct_change
```

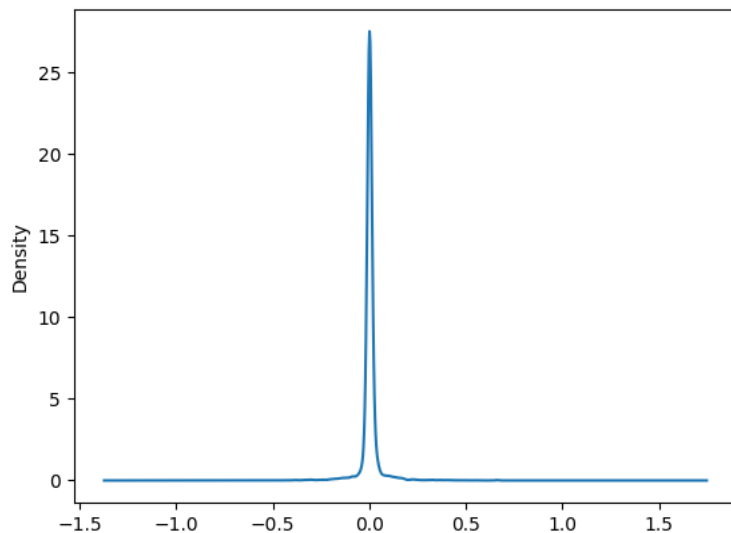
<Axes: ylabel='Density'>



Here we can see the distribution of the density of zeros or the bad trades for the precursor ask capitalization percent change.

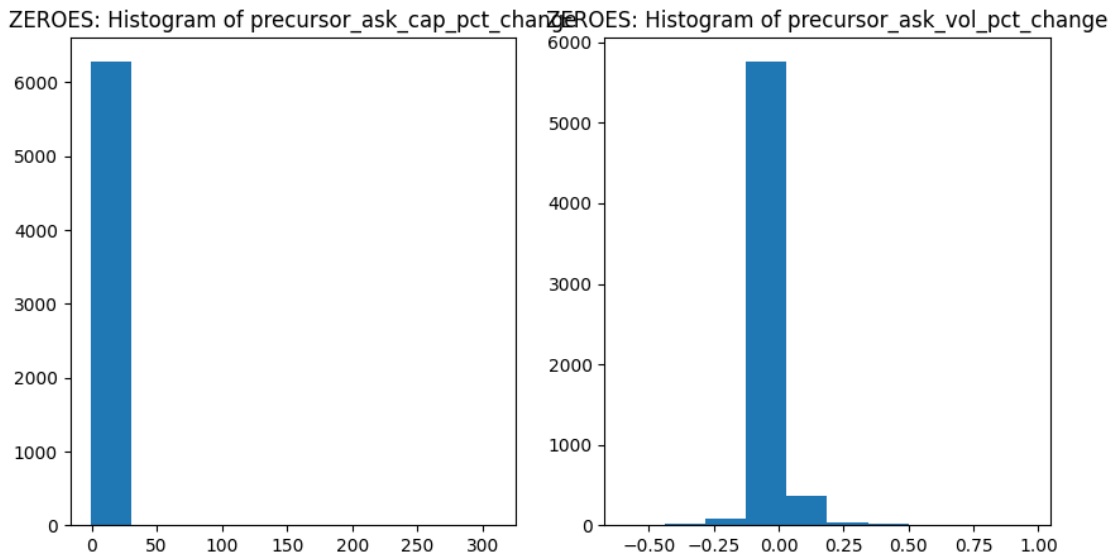
```
zeroes['precursor_ask_vol_pct_change'].plot.density() #precursor_ask_vol_pct_change
```

<Axes: ylabel='Density'>



Here we can see the distribution of the density of zeros or the bad trades for the precursor ask volume percent change.

```
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].hist(zeroes['precursor_ask_cap_pct_change'], bins=10)
axs[0].set_title('ZEROES: Histogram of precursor_ask_cap_pct_change')
axs[1].hist(zeroes['precursor_ask_vol_pct_change'], bins=10)
axs[1].set_title('ZEROES: Histogram of precursor_ask_vol_pct_change')
plt.show()
```



On the left we can see the distribution of the zeros or the bad trades on the precursor ask capitalization percent change; and on the right we can see the distribution of the zeros on the precursor ask volume percent change.

✓ Modeling and Prediction Using KNeighbors

```
m2_pipeline = mdf #pd.read_csv("0 Data Processing/binary_binned_pipeline.csv") #use mdf instead

corr_list = [
    'precursor_buy_cap_pct_change', 'precursor_ask_cap_pct_change',
    'precursor_bid_vol_pct_change', 'precursor_ask_vol_pct_change', 'length', 'sum_change', 'surge_targets_met_pct', 'time', 'label']

m2_pipeline = m2_pipeline[corr_list]
keepable = ['precursor_buy_cap_pct_change',
            'precursor_ask_cap_pct_change',
            'precursor_bid_vol_pct_change',
            'precursor_ask_vol_pct_change',
            'sum_change', 'length', 'time']

y = m2_pipeline['label'].values # y is always a vector, a list of labels
X = m2_pipeline[keepable].values #x matrix is a list of values/dimensions

X_resampled, y_resampled = ADASYN(random_state=42).fit_resample(X, y) #create synthetic classes

X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

scaler = StandardScaler() #standardize all numerics
X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.fit_transform(X_test)

knn = KNeighborsClassifier(algorithm='auto', n_jobs=1, n_neighbors=3)
knn.fit(X_train_scaled, y_train)
```

```
▼ KNeighborsClassifier
KNeighborsClassifier(n_jobs=1, n_neighbors=3)
```

Here we used y which is always a vector, or a list of labels and x matrix which is a list of values and dimensions to scale and resampled to use in a algorithm K-Neighbors Classifier.

```
corr_list = [
'precursor_buy_cap_pct_change', 'precursor_ask_cap_pct_change',
'precursor_bid_vol_pct_change', 'precursor_ask_vol_pct_change', 'length', 'sum_change', 'surge_targets_met_pct', 'time', 'label']

m2_pipeline = m2_pipeline[corr_list]
m2_pipeline.corr()
```

	precursor_buy_cap_pct_change	precursor_ask_cap_pct_change	precursor_bid
precursor_buy_cap_pct_change	1.000000	0.195900	
precursor_ask_cap_pct_change	0.195900	1.000000	
precursor_bid_vol_pct_change	0.547428	0.190969	
precursor_ask_vol_pct_change	0.177817	0.217833	
length	-0.074944	0.055215	
sum_change	0.136782	-0.131603	
surge_targets_met_pct	-0.001754	0.067987	
time	-0.068998	-0.044788	
label	-0.025531	0.041780	

Here we have a data set of the buy and ask for the capitalization percent change and the bid and ask for the volume percent change along with the lenght, sum change, surge targets, time, and labels.

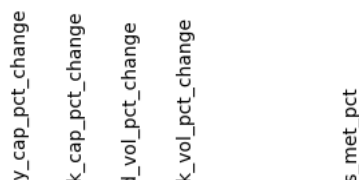
```
import pandas as pd
import matplotlib.pyplot as plt

# create a sample dataframe
df = m2_pipeline

# calculate the correlation matrix
corr_matrix = df.corr()

# plot the correlation matrix
plt.matshow(corr_matrix)
plt.xticks(range(len(corr_matrix.columns)), corr_matrix.columns, rotation=90)
plt.yticks(range(len(corr_matrix.columns)), corr_matrix.columns)

plt.show()
```



Here we can see the relative correlation between the points or the different type of classifiers. Here the correlation matrix indicates which variables they correlate with or have a similar linear behavior. As we can see above the precursor buy capitalization percent change and the precursor bid volume percent change are highly correlated.

```
#predict
y_pred_knn = knn.predict(X_test_scaled)
```

```
#plot decision boundary
# Assuming your KNN model is stored in the variable 'knn'
# plot_decision_boundary(knn, X_test_scaled, y_test)
# plt.show()
```

Here we predict the test variables.

```
# Compute the k-neighborhood graph for your data
graph = knn.kneighbors_graph(X)
graph
```

```
<6416x12589 sparse matrix of type '<class 'numpy.float64'>'
  with 19248 stored elements in Compressed Sparse Row format>
```

Here we are computing the K Neighborhood graph for our data to visualize it.

```
#display confusion matrix

y_pred_knn = knn.predict(X_test_scaled)

accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(accuracy_knn)

labels_ = m2_pipeline['label'].unique()
print(labels_)
ConfusionMatrixDisplay.from_predictions(y_test, y_pred_knn, labels=labels_)

print(classification_report(y_test, y_pred_knn ,zero_division=1))
```



```
0.971405877680699
[0 1]
```

	precision	recall	f1-score	support
0	0.99	0.95	0.97	1267
1	0.95	0.99	0.97	1251
accuracy			0.97	2518
macro avg	0.97	0.97	0.97	2518

Above we have a confusion matrix which is a type of error analysis. Here we tried to predict the test values against the K Neighbors and we scored the degree that the K Neighbors could recognize what something was. Precision and recall are two types of errors which are the degrees of the predictive model falsely recognizing something such as it be shown as a negative when it is actually a positive or it comes up as a positive when it is a negative. The purple shaded boxes represent the errors or mistakes which were made when trying to predict the zero otherwise known as bad trades.



▼ Causality Studies



```
ones_r = mdf[mdf['label']==1][keepable] #good trades
ones_r
```

	precursor_buy_cap_pct_change	precursor_ask_cap_pct_change	precursor_bid_vol_pct_change	precu
47	0.008169	-0.000036	0.003968	
108	-0.002202	-0.000064	-0.003449	
144	-0.204558	0.000178	-0.088451	
159	0.006487	-0.000134	0.003800	
189	0.001016	-0.000022	0.002570	
...	
6283	-0.012920	-0.003338	-0.002208	
6292	-0.008260	0.011126	-0.000741	
6312	0.215995	0.003481	0.027989	
6315	0.059232	0.009487	0.009968	
6395	0.005644	0.002875	0.000943	

119 rows x 7 columns

Here we have a casual inference and here we are looking for different ways that certain patterns are true under different circumstances.

```
def test_granger(df, p):
    """
    Fits a VAR(p) model on the input df and performs pairwise Granger Causality tests
    """
    # Fit VAR model on first-order differences
    model = VAR(df.diff().dropna())
    results = model.fit(p)
    # Initialize p-value matrix
    p_matrix = pd.DataFrame(index=df.columns, columns=df.columns)
    # Perform pairwise Granger Causality tests
    for caused in df.columns:
        for causing in df.columns:
            if caused != causing:
                test_result = results.test_causality(caused, causing)
                p_value = test_result.pvalue
                p_matrix.loc[caused, causing] = p_value
    # Ensure all columns have float dtype
    p_matrix = p_matrix.astype(float)
    return p_matrix

p=7
ones = mdf[mdf['label']==1] #good trades
p_matrix0 = test_granger(ones_r, p)
caul_mtrx = p_matrix0.rename(index={item: f"{item} caused by" for item in p_matrix0.index})
caul_mtrx.where(caul_mtrx.isna(), caul_mtrx <= 0.01)
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: Va
self._init_dates(dates, freq)

precursor_buy_cap_pct_change precursor_ask_cap_pct_c

precursor_buy_cap_pct_change
caused by NaN

precursor_ask_cap_pct_change
caused by False


precursor_bid_vol_pct_change
caused by False

precursor_ask_vol_pct_change
caused by False

sum_change caused by False
```

Here we took the ones data and ran the granger network. The table is similar to a correlation table as we are trying to see if there is any likelihood of a variable proceeding the other.

```
zeroes_r = mdf[mdf['label']==0][keepable] #bad trades
p_matrix1 = test_granger(zeroes_r, p)
caul_mtrx = p_matrix1.rename(index={item: f"{item} caused by" for item in p_matrix1.index})
caul_mtrx.where(caul_mtrx.isna(), caul_mtrx <= 0.01)
```



```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: Va
self._init_dates(dates, freq)

precursor_buy_cap_pct_change precursor_ask_cap_pct_c

precursor_buy_cap_pct_change
caused by NaN

precursor_ask_cap_pct_change
caused by False

precursor_bid_vol_pct_change
caused by False

precursor_ask_vol_pct_change
caused by False

sum_change caused by False
```

Here we took the zeros data and ran the granger network. Just like the table before we are also trying to see if there is any likelihood of a variable proceeding the other but this time for the zeros data.