prepocess_spy

January 28, 2025

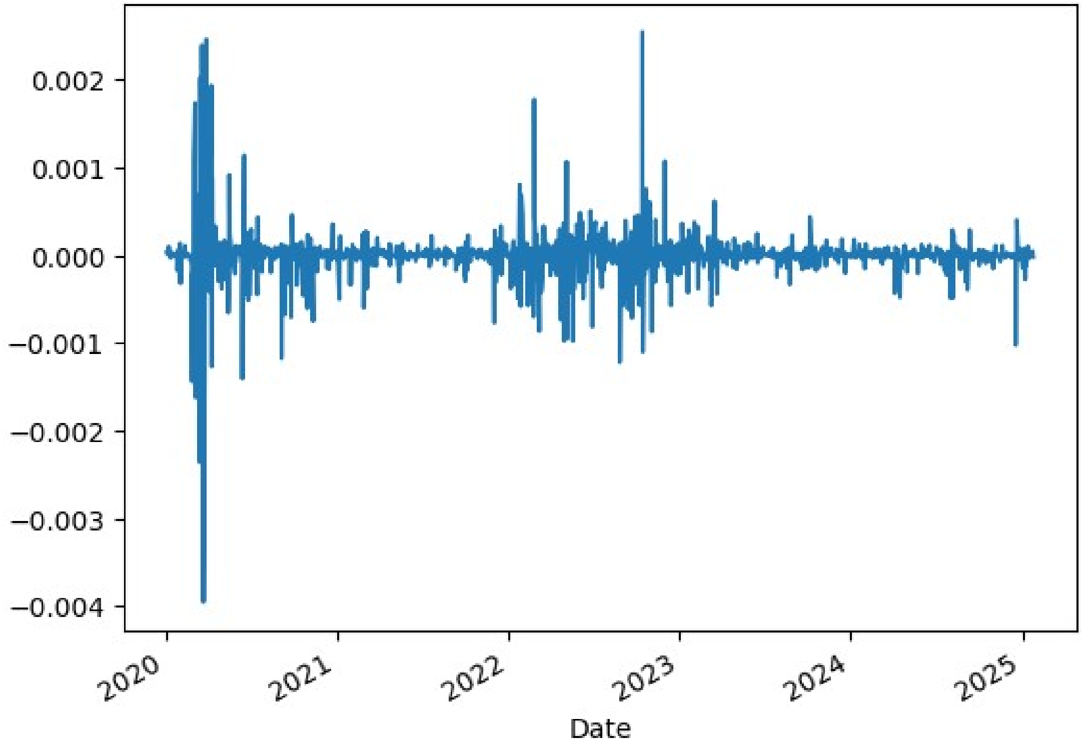
[121]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import yfinance as yf
```

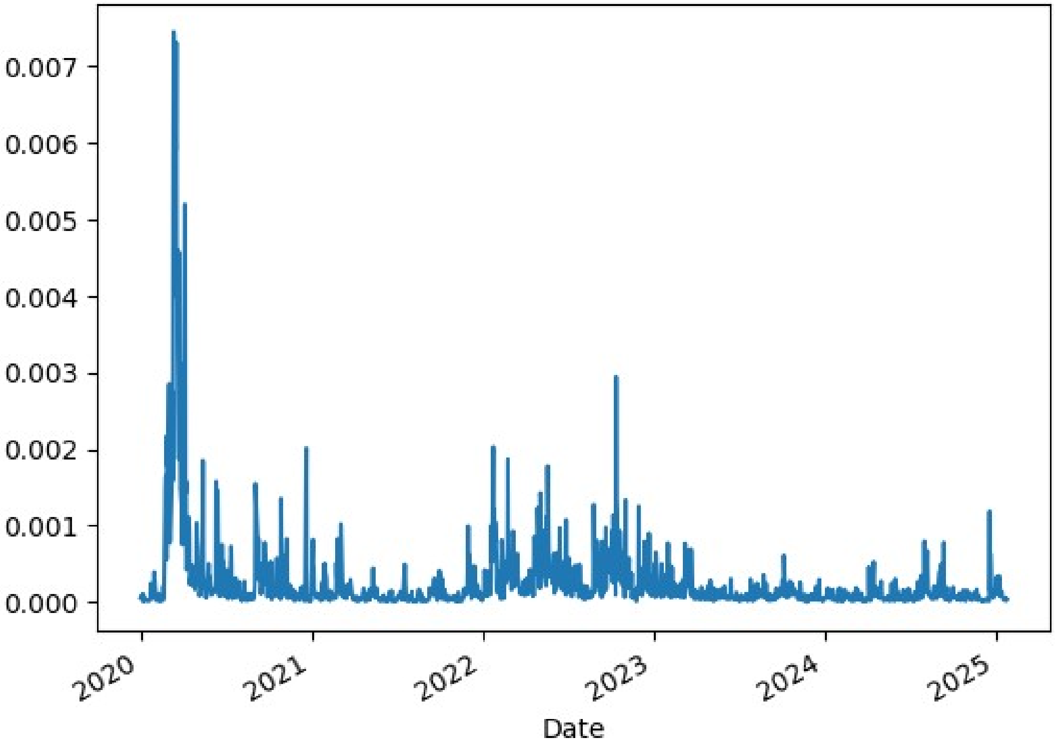
[122]:
```
df = pd.read_csv('preprocess_spy.csv', header=0, index_col=0)
df.head()
```

[122]:
```
         date  dividends  week_day month  day  quarter diff_close_close\
0  2020-01-02        0.0  Thursday     1    2        1         2.790588
1  2020-01-03        0.0    Friday     1    3        1        -2.280609
2  2020-01-06        0.0    Monday     1    6        1         1.140289
3  2020-01-07        0.0   Tuesday     1    7        1        -0.843567
4  2020-01-08        0.0 Wednesday     1    8        1         1.594543

   pct_change pct_open_closepct_close_open…   ph  spgi  trow      tsla  \
0    0.009352       0.005220       0.004111 …  0.0   0.0   0.0  0.317294
1   -0.007572      -0.011420       0.003892 …  0.0   0.0   0.0  0.068317
2    0.003815      -0.005955       0.009829 …  0.0   0.0   0.0  0.056199
3   -0.002811      -0.001915      -0.000898 …  0.0   0.0   0.0  0.210498
4    0.005329       0.000650       0.004676 …  0.0   0.0   0.0  0.302827

   tsm  txn       unh   v       wmt       xom
0  0.0  0.0  0.000000 0.0  0.436396 0.000000
1  0.0  0.0  0.000000 0.0  0.241386 0.000000
2  0.0  0.0  0.000000 0.0 -0.4109540.265997
3  0.0  0.0  0.157431 0.0  0.059462 0.198767
4  0.0  0.0  0.134564 0.0 -0.071081 -0.075503

[5 rows x 53 columns]
```

[123]:
```
df = pd.read_csv('preprocess_spy.csv', header=0, index_col=0)
df['dividends'] = df['dividends'].apply(lambda x: True if x > 0 else False).
 ↪astype(int)
df.drop(columns=['day', 'quarter', 'seasonal_diff','diff_close_close'],␣
 ↪inplace=True)
df.drop(columns=[ 'high_open_ratio', 'high_close_ratio', 'low_open_ratio',
        'low_close_ratio'], inplace=True)
```

[124]:
```
 # from 2020 to 2025
spy = yf.Ticker("SPY")
hist = spy.history(start="2020-01-01", end="2025-01-25")
hist.head()



hist['product_open'] = (hist['High'] - hist['Low']) / (hist['Open'])
hist['product_close'] = (hist['High'] - hist['Low']) / (hist['Close'])

hist['product_diff'] = hist['product_open'] - hist['product_close']
hist['product'] = hist['product_open'] * hist['product_close']

hist['product_diff'].plot()
plt.show();

hist['product'].plot()
plt.show();
```





```
[ ]:
```

```
[125]: hist.head()
```

```
[125]:                            Open        High         Low       Close  \
Date
2020-01-02 00:00:00-05:00 299.961853 301.213476 299.025448 301.194916
2020-01-03 00:00:00-05:00 297.755279 300.054558 297.699654 298.914185
2020-01-06 00:00:00-05:00 297.134101 300.138003 297.013570 300.054565
2020-01-07 00:00:00-05:00 299.479722 299.961845 298.756566 299.210876
2020-01-08 00:00:00-05:00 299.405545 302.038574 299.155231 300.805511

                            Volume  Dividends  Stock Splits  Capital Gains  \
Date
2020-01-02 00:00:00-05:00 59151200        0.0           0.0            0.0
2020-01-03 00:00:00-05:00 77709700        0.0           0.0            0.0
2020-01-06 00:00:00-05:00 55653900        0.0           0.0            0.0
2020-01-07 00:00:00-05:00 40496400        0.0           0.0            0.0
2020-01-08 00:00:00-05:00 68296000        0.0           0.0            0.0

                          product_open  product_close  product_diff  product
Date
```

```
2020-01-02 00:00:00-05:00  0.007294     0.007264     0.000030 0.000053
2020-01-03 00:00:00-05:00  0.007909     0.007878     0.000031 0.000062
2020-01-06 00:00:00-05:00  0.010515     0.010413     0.000102 0.000109
2020-01-07 00:00:00-05:00  0.004025     0.004028    -0.000004 0.000016
2020-01-08 00:00:00-05:00  0.009630     0.009585     0.000045 0.000092
```

[126]:
```
hist['trend'] = hist.apply(lambda row: 1 if row['Close'] >= row['Open'] else 0,
↪axis=1)
```

[127]:
```
# to datetime
hist['date'] = pd.to_datetime(hist.index)
hist.reset_index(inplace=True, drop=True)

hist['date'] = hist['date'].dt.date
df['date'] = pd.to_datetime(df['date'])
df['date'] = df['date'].dt.date


hist = hist[['date', 'product', 'product_diff', 'trend']]
df.drop(columns=['trend'], inplace=True)
# merge the two dataframes on the date
df = pd.merge(df, hist, on='date', how='left')

df.head(1)
```

[127]:
```
         date  dividends week_day month  pct_change pct_open_close\
0  2020-01-02          0  Thursday     1    0.009352        0.00522

   pct_close_opensecond_diff_seasonalcontinuous_increased\
0        0.004111           3.078827                    2

   continuous_decreased…      tsla tsm  txn  unh   v       wmt  xom  \
0                    0  …  0.317294 0.0  0.0  0.0  0.0  0.436396 0.0

    product product_diff trend
0  0.000053      0.00003     1

[1 rows x 47 columns]
```

[128]:
```
# one hot encoding for weekdays

df = pd.get_dummies(df, columns=['week_day'], dtype=int)
```

[129]:
```
df.columns
```

[129]:
```
Index(['date', 'dividends', 'month', 'pct_change', 'pct_open_close',
       'pct_close_open', 'second_diff_seasonal', 'continuous_increased',
       'continuous_decreased', 'aapl', 'acn', 'adi', 'ame', 'amp', 'amzn',
       'aph', 'avgo', 'brk-b', 'cost', 'dov', 'googl', 'hd', 'intu', 'itw',
       'jpm', 'lly', 'ma', 'mco', 'meta', 'msft', 'nvda', 'orcl', 'payx', 'ph',
       'spgi', 'trow', 'tsla', 'tsm', 'txn', 'unh', 'v', 'wmt', 'xom',
       'product', 'product_diff', 'trend', 'week_day_Friday',
       'week_day_Monday', 'week_day_Thursday', 'week_day_Tuesday',
       'week_day_Wednesday'],
      dtype='object')
```

```
[130]: # convert the month integer to datetime and then to string (ex. Jan) and one␣
        ↪hot encode it
       df['month'] = df['month'].apply(lambda x: pd.to_datetime(f'2020-{x}-01').
        ↪strftime('%b'))

       df = pd.get_dummies(df, columns=['month'], dtype=int)

       df.columns
```

```
[130]: Index(['date', 'dividends', 'pct_change', 'pct_open_close', 'pct_close_open',
              'second_diff_seasonal', 'continuous_increased', 'continuous_decreased',
              'aapl', 'acn', 'adi', 'ame', 'amp', 'amzn', 'aph', 'avgo', 'brk-b',
              'cost', 'dov', 'googl', 'hd', 'intu', 'itw', 'jpm', 'lly', 'ma', 'mco',
              'meta', 'msft', 'nvda', 'orcl', 'payx', 'ph', 'spgi', 'trow', 'tsla',
              'tsm', 'txn', 'unh', 'v', 'wmt', 'xom', 'product', 'product_diff',
              'trend', 'week_day_Friday', 'week_day_Monday', 'week_day_Thursday',
              'week_day_Tuesday', 'week_day_Wednesday', 'month_Apr', 'month_Aug',
              'month_Dec', 'month_Feb', 'month_Jan', 'month_Jul', 'month_Jun',
              'month_Mar', 'month_May', 'month_Nov', 'month_Oct', 'month_Sep'],
             dtype='object')
```

```
[131]: df.columns
```

```
[131]: Index(['date', 'dividends', 'pct_change', 'pct_open_close', 'pct_close_open',
              'second_diff_seasonal', 'continuous_increased', 'continuous_decreased',
              'aapl', 'acn', 'adi', 'ame', 'amp', 'amzn', 'aph', 'avgo', 'brk-b',
              'cost', 'dov', 'googl', 'hd', 'intu', 'itw', 'jpm', 'lly', 'ma', 'mco',
              'meta', 'msft', 'nvda', 'orcl', 'payx', 'ph', 'spgi', 'trow', 'tsla',
              'tsm', 'txn', 'unh', 'v', 'wmt', 'xom', 'product', 'product_diff',
              'trend', 'week_day_Friday', 'week_day_Monday', 'week_day_Thursday',
              'week_day_Tuesday', 'week_day_Wednesday', 'month_Apr', 'month_Aug',
              'month_Dec', 'month_Feb', 'month_Jan', 'month_Jul', 'month_Jun',
              'month_Mar', 'month_May', 'month_Nov', 'month_Oct', 'month_Sep'],
             dtype='object')
```

```
[ ]:
```

```
[132]: # move trend to the end

       df[['pct_change',␣
        ↪'pct_close_open','second_diff_seasonal','continuous_increased','continuous_decre
        ↪'product', 'product_diff', 'aapl', 'acn', 'adi', 'ame', 'amp', 'amzn',␣
        ↪'aph', 'avgo', 'brk-b',
              'cost', 'dov', 'googl', 'hd', 'intu', 'itw', 'jpm', 'lly', 'ma', 'mco',
              'meta', 'msft', 'nvda', 'orcl', 'payx', 'ph', 'spgi', 'trow', 'tsla',
              'tsm', 'txn', 'unh', 'v', 'wmt', 'xom']] = df[['pct_change',␣
        ↪'pct_close_open','second_diff_seasonal','continuous_increased','continuous_decre
        ↪'product', 'product_diff', 'aapl', 'acn', 'adi', 'ame', 'amp', 'amzn',␣
        ↪'aph', 'avgo', 'brk-b',
              'cost', 'dov', 'googl', 'hd', 'intu', 'itw', 'jpm', 'lly', 'ma', 'mco',
              'meta', 'msft', 'nvda', 'orcl', 'payx', 'ph', 'spgi', 'trow', 'tsla',
              'tsm', 'txn', 'unh', 'v', 'wmt', 'xom']].shift(1)
```

```
[133]: df.isna().sum()
```

```
[133]: date              0
       dividends         0
       pct_change        1
       pct_open_close    0
       pct_close_open    1
                        ..
       month_Mar         0
       month_May         0
       month_Nov         0
       month_Oct         0
       month_Sep         0
       Length: 62, dtype: int64
```

```
[134]: comp_dict = {}
       for i, comp in enumerate(df.columns.tolist()):
           comp_dict[comp] = i
```

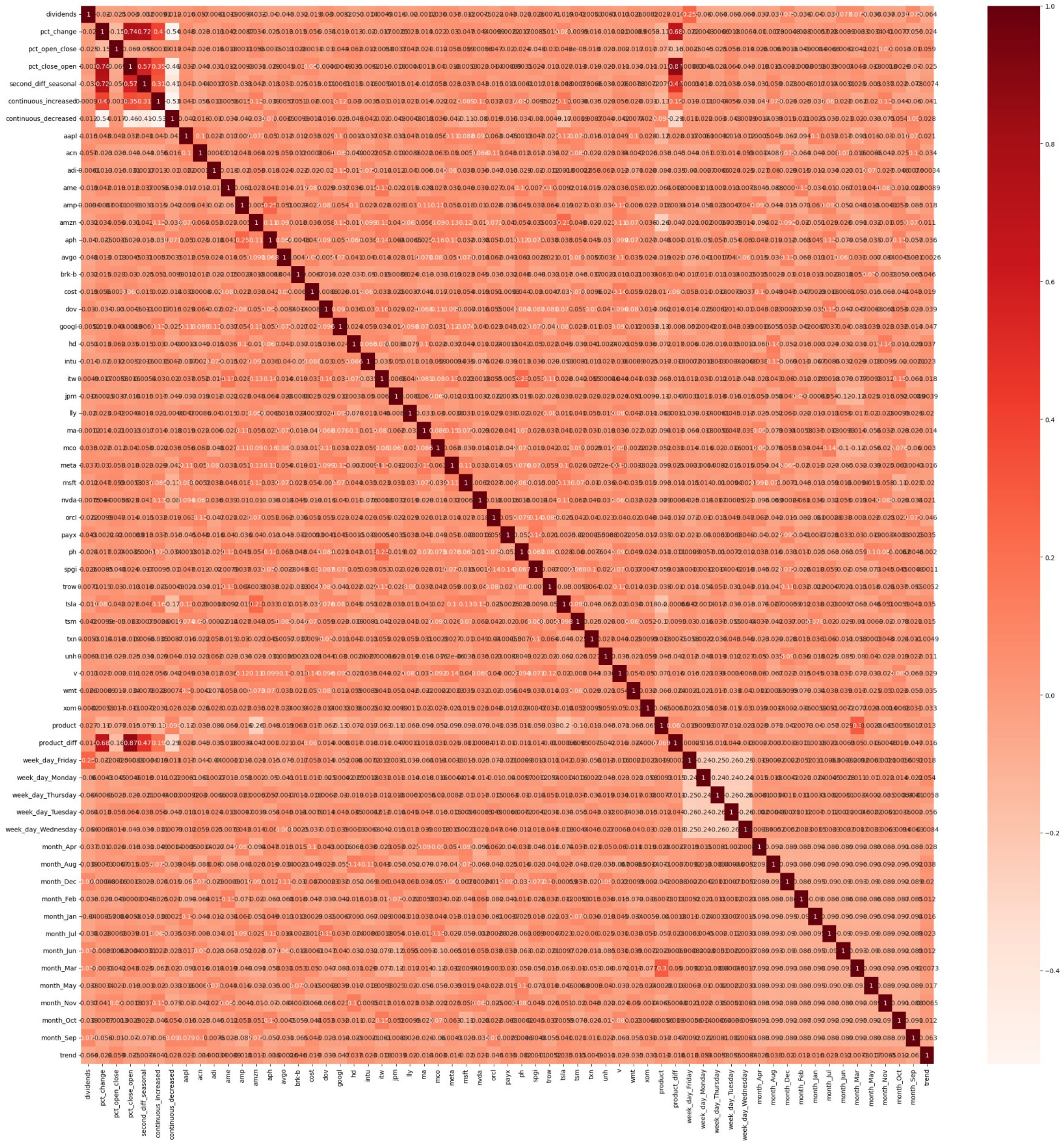
```
[508]:
```

```
[135]: compay_columns = df.columns[comp_dict['aapl']:comp_dict['xom']].tolist()

       for comp in compay_columns:
           df[comp] = (df[comp].shift(-1, fill_value=0) + df[comp])/2
```

```
[136]: # move trend to last column

       trend = df.pop('trend')
       df['trend'] = trend
```

[137]:

```
cor = df.loc[:, 'dividends':].corr()

# create a heat map

plt.figure(figsize=(30, 30))
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
plt.show()
```



[138]:

```
df.drop(columns='date').corr()['trend'].sort_values(ascending=False)
```

```
[138]: trend              1.000000
       pct_open_close     0.059147
       week_day_Monday    0.053844
       brk-b              0.046010
       month_Aug          0.037525
                            …
       orcl              -0.045927
       googl             -0.047263
       week_day_Tuesday  -0.055558
       month_Sep         -0.063031
       dividends         -0.064086
       Name: trend, Length: 61, dtype: float64
```

```
[139]: df.describe()
```

```
[139]:          dividends   pct_change  pct_open_closepct_close_open\
       count  1273.000000 1272.000000    1273.000000     1272.000000
       mean      0.015711    0.000649       0.000409        0.000235
       std       0.124404    0.013184       0.008799        0.009328
       min       0.000000   -0.109424      -0.104485       -0.056612
       25%       0.000000   -0.005094      -0.002664       -0.004316
       50%       0.000000    0.000963       0.000769        0.000747
       75%       0.000000    0.007429       0.003962        0.005265
       max       1.000000    0.090603       0.060376        0.047994

              second_diff_seasonalcontinuous_increasedcontinuous_decreased\
       count         1272.000000         1272.000000         1272.000000
       mean             0.099614            1.188679            0.790881
       std              6.288492            1.566947            1.133974
       min            -28.235977            0.000000            0.000000
       25%             -3.506115            0.000000            0.000000
       50%              0.184875            1.000000            0.000000
       75%              3.712875            2.000000            1.000000
       max             28.215988           10.000000            7.000000

                    aapl         acn          adi  …    month_Feb   month_Jan \
       count  1267.000000 1267.000000 1267.000000 …  1273.000000 1273.000000
       mean      0.124624    0.055725    0.022470 …     0.075412    0.091123
       std       0.122300    0.097780    0.072933 …     0.264159    0.287898
       min      -0.285980   -0.416715   -0.237138 …     0.000000    0.000000
       25%       0.052644    0.000000    0.000000 …     0.000000    0.000000
       50%       0.137695    0.000000    0.000000 …     0.000000    0.000000
       75%       0.210602    0.107110    0.000000 …     0.000000    0.000000
       max       0.444473    0.458561    0.446510 …     1.000000    1.000000
```

```
        month_Jul   month_Jun   month_Mar   month_May   month_Nov \
count 1273.000000 1273.000000 1273.000000 1273.000000 1273.000000
mean     0.082482    0.082482    0.087196    0.082482    0.080911
std      0.275206    0.275206    0.282232    0.275206    0.272806
min      0.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000    0.000000    0.000000    0.000000    0.000000
50%      0.000000    0.000000    0.000000    0.000000    0.000000
75%      0.000000    0.000000    0.000000    0.000000    0.000000
max      1.000000    1.000000    1.000000    1.000000    1.000000

        month_Oct   month_Sep       trend
count 1273.000000 1273.000000 1273.000000
mean     0.085625    0.080911    0.552239
std      0.279919    0.272806    0.497459
min      0.000000    0.000000    0.000000
25%      0.000000    0.000000    0.000000
50%      0.000000    0.000000    1.000000
75%      0.000000    0.000000    1.000000
max      1.000000    1.000000    1.000000

[8 rows x 61 columns]
```

[140]:

```
# standardize the data
from sklearn.preprocessing import StandardScaler

# columns to standardize

stnd_list = ['pct_change', 'pct_close_open', 'second_diff_seasonal',
      'continuous_increased', 'continuous_decreased', 'product',
      'product_diff', 'aapl', 'acn', 'adi', 'ame', 'amp', 'amzn', 'aph',
      'avgo', 'brk-b', 'cost', 'dov', 'googl', 'hd', 'intu', 'itw', 'jpm',
      'lly', 'ma', 'mco', 'meta', 'msft', 'nvda', 'orcl', 'payx', 'ph',
      'spgi', 'trow', 'tsla', 'tsm', 'txn', 'unh', 'v', 'wmt', 'xom']

scaler = StandardScaler()

df[stnd_list] = scaler.fit_transform(df[stnd_list])

df.dropna(inplace=True)

df.head()
```

[140]:

```
        date  dividends pct_change pct_open_closepct_close_open\
1  2020-01-03         0   0.660344      -0.011420      0.415661
2  2020-01-06         0  -0.623821      -0.005955      0.392218
3  2020-01-07         0   0.240200      -0.001915      1.028901
4  2020-01-08         0  -0.262594       0.000650     -0.121477
```

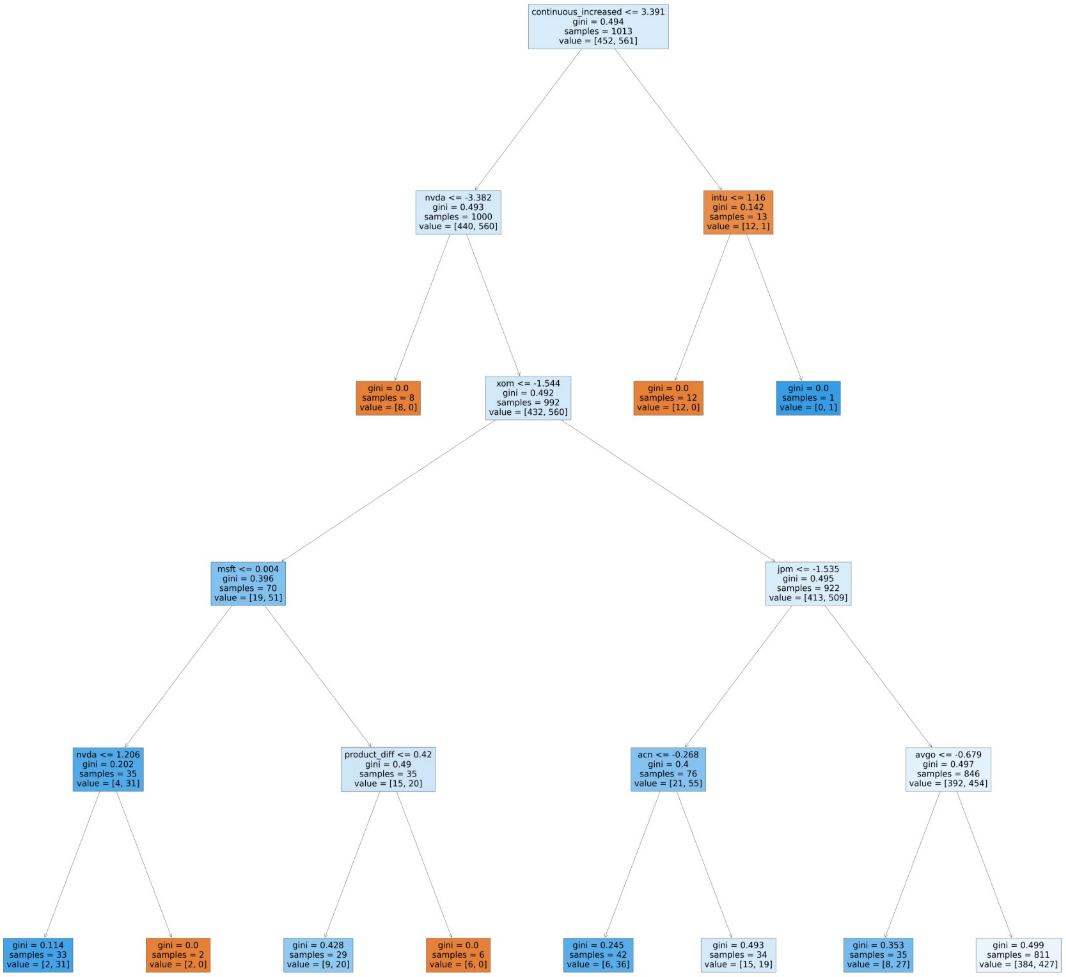
```
5  2020-01-09        0    0.355112      0.005271      0.476265

   second_diff_seasonal  continuous_increased  continuous_decreased      aapl  \
1              0.473943              0.517975             -0.697716  0.647447
2             -0.453158             -0.758894              0.184486 -0.343627
3             -0.036448             -0.120459             -0.697716 -0.270196
4             -0.493873             -0.758894              0.184486  1.206030
5              0.333805             -0.120459             -0.697716  1.481839

        acn  ...  month_Feb  month_Jan  month_Jul  month_Jun  month_Mar  \
1 -0.570130  ...          0          1          0          0          0
2 -0.570130  ...          0          1          0          0          0
3  1.207838  ...          0          1          0          0          0
4  1.207838  ...          0          1          0          0          0
5 -0.570130  ...          0          1          0          0          0

   month_May  month_Nov  month_Oct  month_Sep  trend
1          0          0          0          0      1
2          0          0          0          0      1
3          0          0          0          0      0
4          0          0          0          0      1
5          0          0          0          0      1

[5 rows x 62 columns]
```

[176]:

```python
#df.drop(columns='date', inplace=True)

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score


X = df.drop(columns='trend')
y = df['trend']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

clf = DecisionTreeClassifier(max_depth=5)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

print(accuracy_score(y_test, y_pred))

plt.figure(figsize=(100, 100))
```

```
plot_tree(clf, filled=True, feature_names=X.columns)
plt.show();
```

```
0.5354330708661418
```



```
[173]: feature_importances = clf.feature_importances_
feature_names = X_train.columns#Eğer pandas DataFrame kullanıyorsanız
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance':␣
 ↪feature_importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

print(importance_df[:20])
```

```
                    Feature  Importance
```

```
4   second_diff_seasonal 0.061119
7                   aapl   0.053523
12                  amzn   0.044716
2         pct_open_close  0.044683
27                  msft   0.044027
39                   wmt   0.043474
42          product_diff  0.037222
22                   jpm   0.034629
28                  nvda   0.032686
3         pct_close_open  0.031751
26                  meta   0.028453
33                  trow   0.027167
40                   xom   0.026803
35                   tsm   0.026367
36                   txn   0.024482
15                 brk-b   0.024224
29                  orcl   0.023725
34                  tsla   0.023253
23                   lly   0.022895
8                    acn   0.022194
```

```
[146]: # create a confusion matrix

       from sklearn.metrics import confusion_matrix

       confusion_matrix(y_test, y_pred)
```

```
[146]: array([[48, 69],
              [64, 73]])
```

```
[147]: # create a classification report

       from sklearn.metrics import classification_report

       print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.43      0.41      0.42       117
           1       0.51      0.53      0.52       137

    accuracy                           0.48       254
   macro avg       0.47      0.47      0.47       254
weighted avg       0.47      0.48      0.48       254
```

```
[148]: # try xgboost
       from xgboost import XGBClassifier
```

```
xgb = XGBClassifier()

xgb.fit(X_train, y_train)

y_pred = xgb.predict(X_test)

print(accuracy_score(y_test, y_pred))
```

```
0.4921259842519685
```

[149]:

```
# create a confusion matrix

confusion_matrix(y_test, y_pred)
```

[149]:

```
array([[50, 67],
       [62, 75]])
```

[170]:

```
# random forest
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=10)

rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)

print(accuracy_score(y_test, y_pred))

# create a confusion matrix

confusion_matrix(y_test, y_pred)
```

```
0.5039370078740157
```

[170]:

```
array([[58, 59],
       [67, 70]])
```

[182]:

```
# neural network tensorflow

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(128, activation='relu', input_shape=[X_train.shape[1]]),
    layers.Dense(64, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

model.fit(X_train, y_train, epochs=1000)

y_pred = model.predict(X_test)

y_pred = [1 if x > 0.5 else 0 for x in y_pred]

print(accuracy_score(y_test, y_pred))

# create a confusion matrix

confusion_matrix(y_test, y_pred)
```

```
Epoch 1/1000
32/32 [==============================] - 1s 980us/step - loss: 0.6968 -
accuracy: 0.5281
Epoch 2/1000
32/32 [==============================] - 0s 980us/step - loss: 0.6603 -
accuracy: 0.5953
Epoch 3/1000
32/32 [==============================] - 0s 1ms/step - loss: 0.6290 - accuracy:
0.6841
Epoch 4/1000
32/32 [==============================] - 0s 850us/step - loss: 0.5884 -
accuracy: 0.7394
Epoch 5/1000
32/32 [==============================] - 0s 859us/step - loss: 0.5375 -
accuracy: 0.7572
Epoch 6/1000
32/32 [==============================] - 0s 851us/step - loss: 0.4740 -
accuracy: 0.8164
Epoch 7/1000
32/32 [==============================] - 0s 790us/step - loss: 0.4041 -
accuracy: 0.8490
Epoch 8/1000
32/32 [==============================] - 0s 847us/step - loss: 0.3226 -
accuracy: 0.8894
Epoch 9/1000
32/32 [==============================] - 0s 924us/step - loss: 0.2371 -
accuracy: 0.9348
Epoch 10/1000
```