Preliminary Analysis

Pilot Testing models with a small dataset for simplicity and speed. I'd rather not run 5 models with large gridsearch parameters on 5000 x 253 dataset.

Some Links I found Helpful

- [https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html#sphx-glr-auto-examples-classification-plot-classifier-comparison-py (https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html#sphx-glr-auto-examples-classification-plot-classifier-comparison-py)](https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html#sphx-glr-auto-examples-classification-plot-classifier-comparison-py)
- [https://www.youtube.com/watch?v=w4frwjt8uCo (https://www.youtube.com/watch?v=w4frwjt8uCo)](https://www.youtube.com/watch?v=w4frwjt8uCo)

Needed Packages

In [1]:
```
import pandas as pd
import yfinance as yf
import pandas_ta as ta #This is a great library that works with pandas to adapted
import matplotlib.pyplot as plt
%matplotlib inline
```

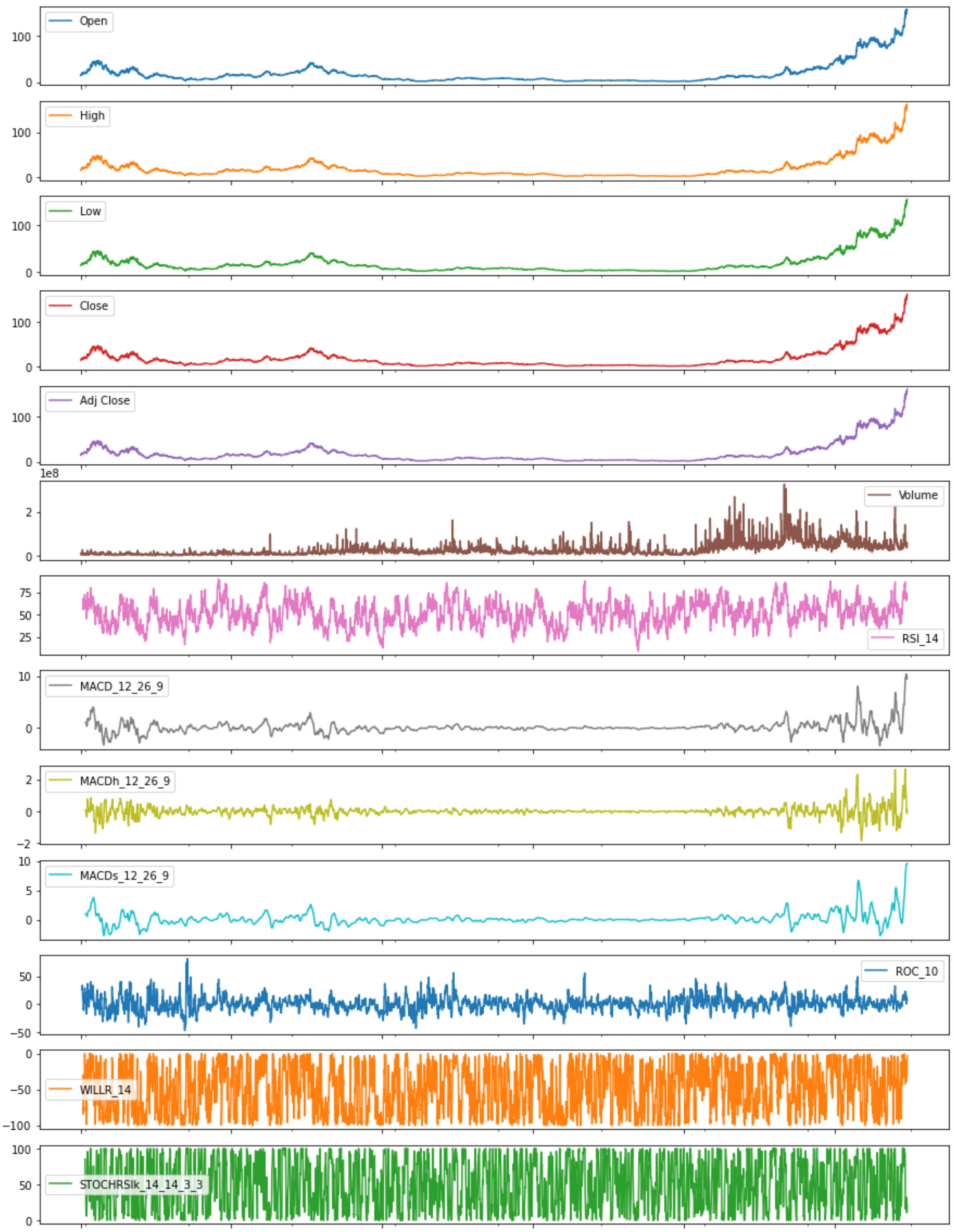
Companies Analysis & Processing

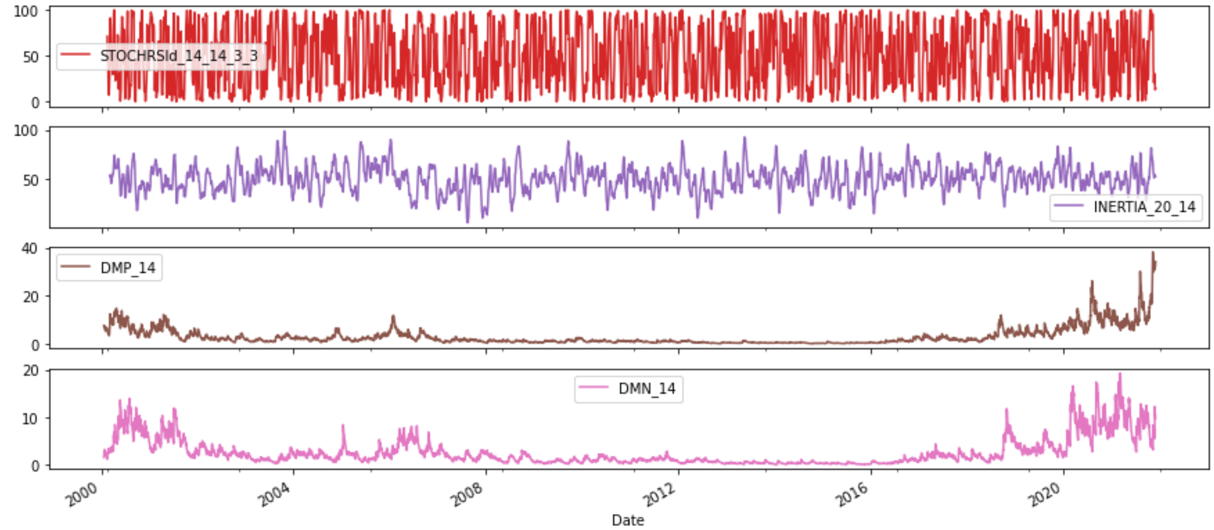
In [2]:
```
target = "^GSPC"
companies = "AMD"
indicies = "^N225 "
commodities = "CL=F"
```

In [3]:
```
AMD = yf.download("AMD", start="2000-01-01", end="2021-11-30", interval="1D")
```

```
[*********************100%***********************]  1 of 1 completed
```

In [4]:
```
AMD.ta.rsi(append=True)
AMD.ta.macd(append=True)
AMD.ta.roc(append=True)
AMD.ta.willr(append=True)
AMD.ta.stochrsi(append=True)
AMD.ta.inertia(append=True)
AMD.ta.dm(append=True)

AMD.plot(subplots=True, figsize=(15,30))
plt.savefig(r"figures\AMD_indicators.png", dpi=600)
```

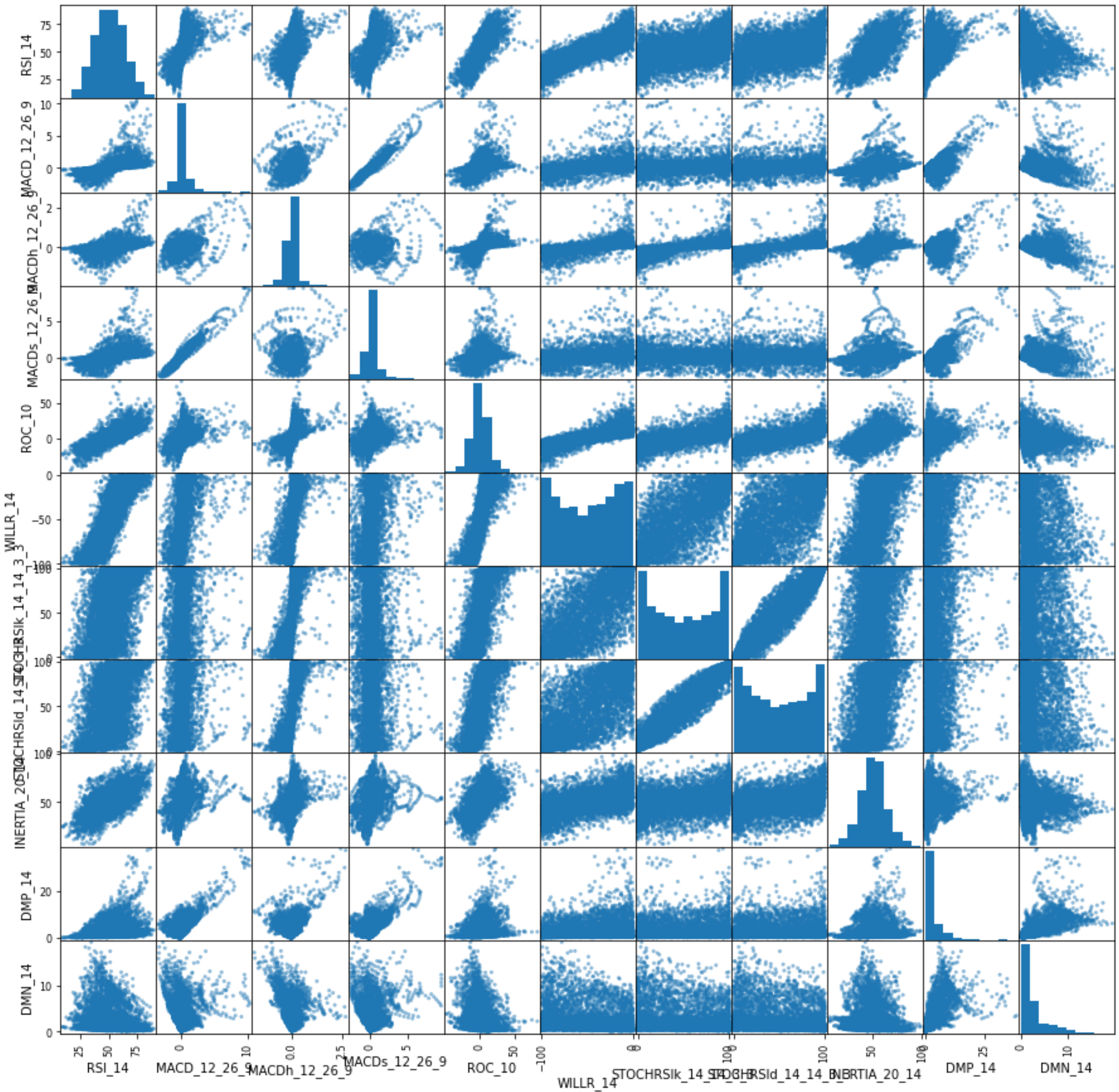




In [5]:
```python
drops = ["Adj Close", "High", "Open", "Low", "Close", "Volume"]
AMD.drop(columns=drops, inplace=True)
AMD.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 5513 entries, 2000-01-03 to 2021-11-29
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   RSI_14                5499 non-null   float64
 1   MACD_12_26_9          5480 non-null   float64
 2   MACDh_12_26_9         5480 non-null   float64
 3   MACDs_12_26_9         5480 non-null   float64
 4   ROC_10                5503 non-null   float64
 5   WILLR_14              5500 non-null   float64
 6   STOCHRSIk_14_14_3_3   5484 non-null   float64
 7   STOCHRSId_14_14_3_3   5482 non-null   float64
 8   INERTIA_20_14         5468 non-null   float64
 9   DMP_14                5500 non-null   float64
 10  DMN_14                5500 non-null   float64
dtypes: float64(11)
memory usage: 516.8 KB
```

In [7]:
```
pd.plotting.scatter_matrix(AMD, figsize=(15,15))
plt.savefig(r"figures\AMD_indicators_scatterMatrix.png", dpi=600)
plt.show()
```

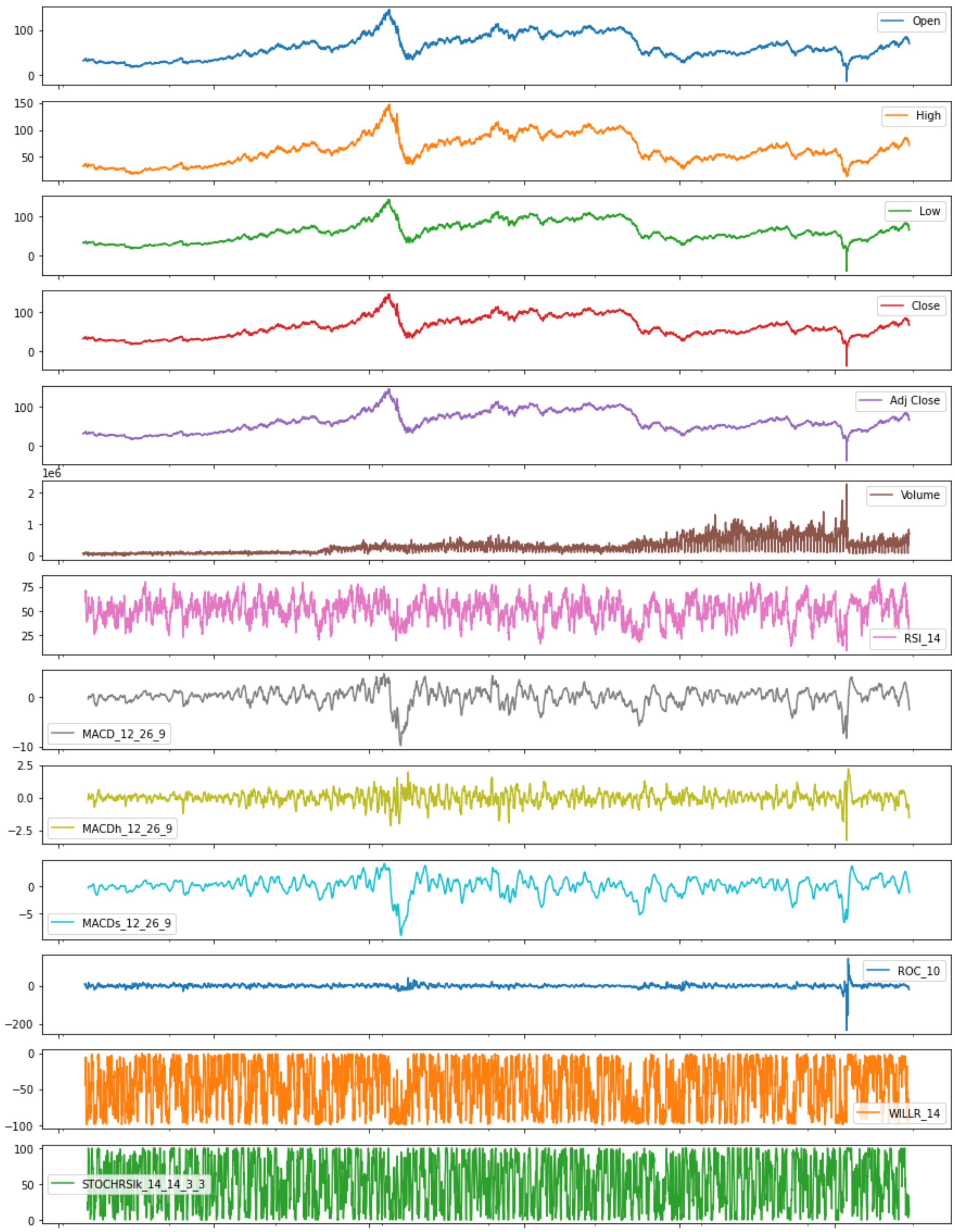


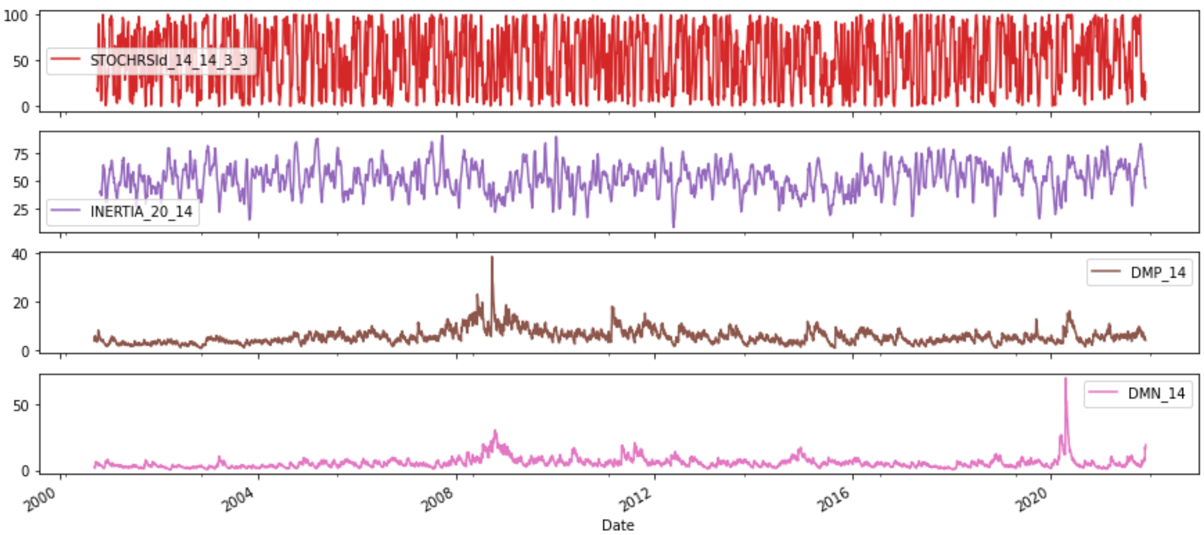
Futures Analysis & Processing

In [8]:
```
CrudeOil = yf.download("CL=F", start="2000-01-01", end="2021-11-30", interval="1D
```

```
[*********************100%***********************]  1 of 1 completed
```

In [9]:
```
CrudeOil.ta.rsi(append=True)
CrudeOil.ta.macd(append=True)
CrudeOil.ta.roc(append=True)
CrudeOil.ta.willr(append=True)
CrudeOil.ta.stochrsi(append=True)
CrudeOil.ta.inertia(append=True)
CrudeOil.ta.dm(append=True)

CrudeOil.plot(subplots=True, figsize=(15,30))
plt.savefig(r"figures\CLF_indicators.png", dpi=600)
```

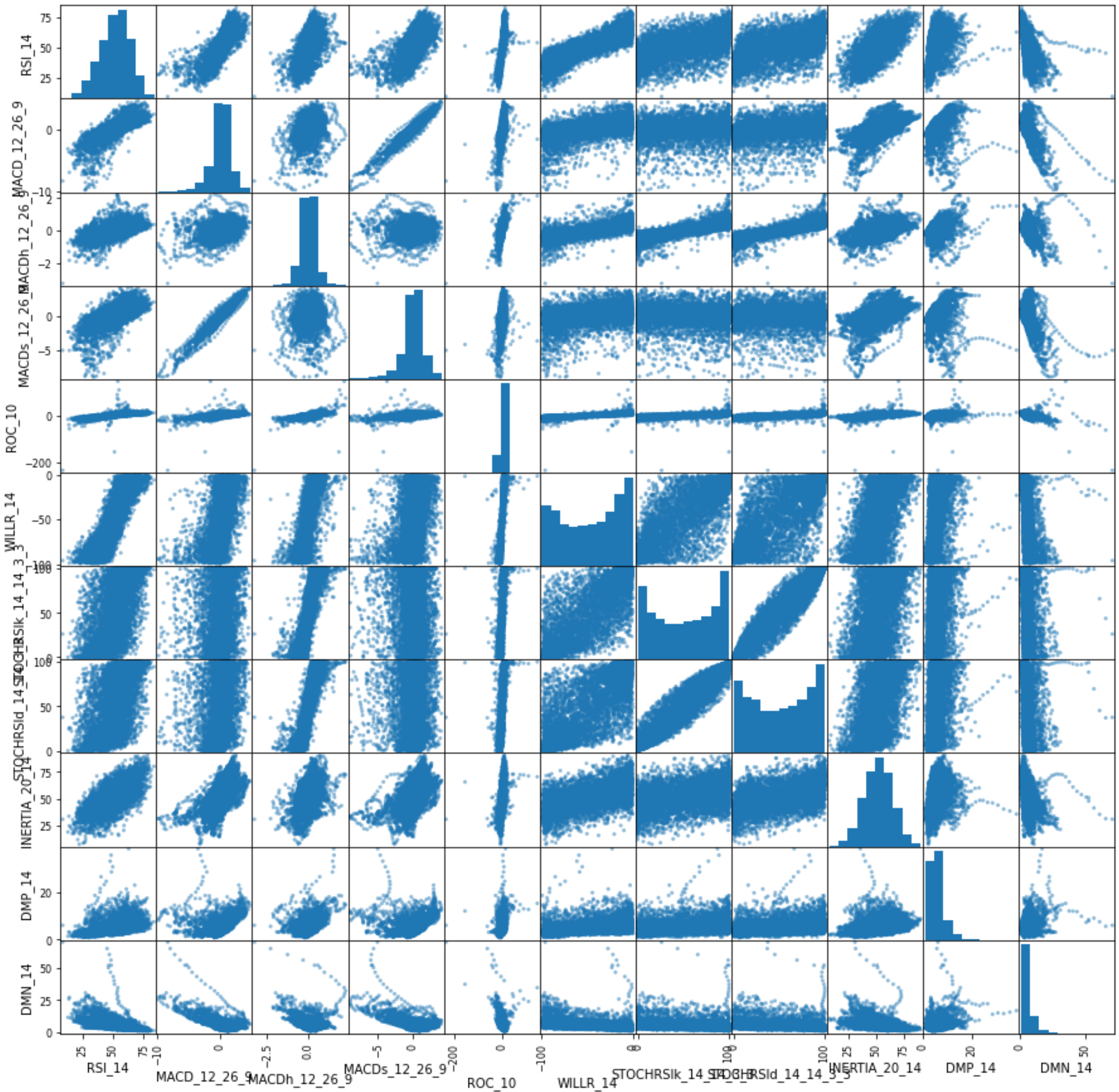




In [10]:
```
drops = ["Adj Close", "High", "Open", "Low", "Close", "Volume"]
CrudeOil.drop(columns=drops, inplace=True)
CrudeOil.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 5341 entries, 2000-08-23 to 2021-11-30
Data columns (total 11 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   RSI_14               5327 non-null   float64
 1   MACD_12_26_9         5308 non-null   float64
 2   MACDh_12_26_9        5308 non-null   float64
 3   MACDs_12_26_9        5308 non-null   float64
 4   ROC_10               5331 non-null   float64
 5   WILLR_14             5328 non-null   float64
 6   STOCHRSIk_14_14_3_3  5312 non-null   float64
 7   STOCHRSId_14_14_3_3  5310 non-null   float64
 8   INERTIA_20_14        5296 non-null   float64
 9   DMP_14               5328 non-null   float64
 10  DMN_14               5328 non-null   float64
dtypes: float64(11)
memory usage: 500.7 KB
```

In [11]:
```
pd.plotting.scatter_matrix(CrudeOil, figsize=(15,15))
plt.savefig(r"figures\CLF_indicators_scatterMatrix.png", dpi=600)
plt.show()
```



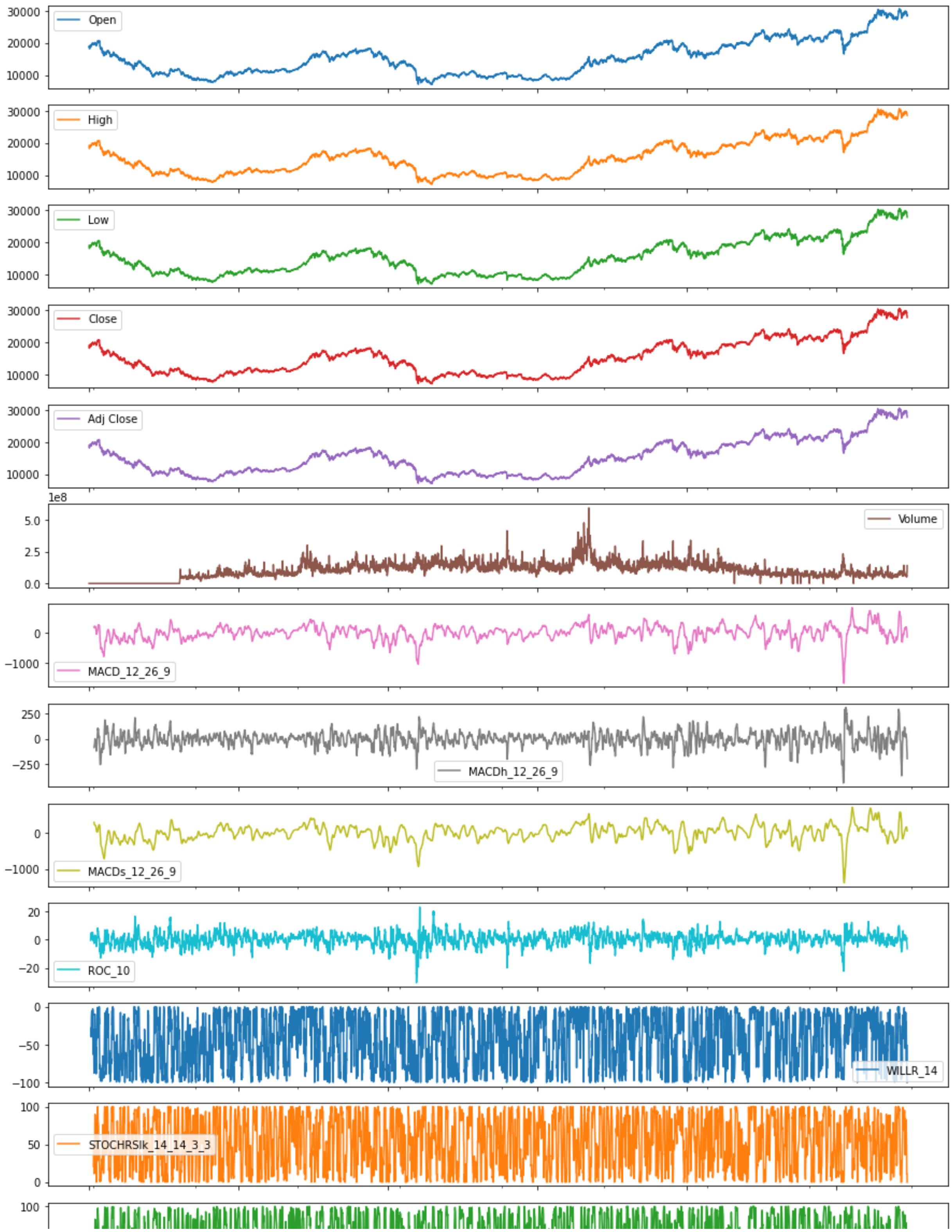
Indexies Analysis & Processing

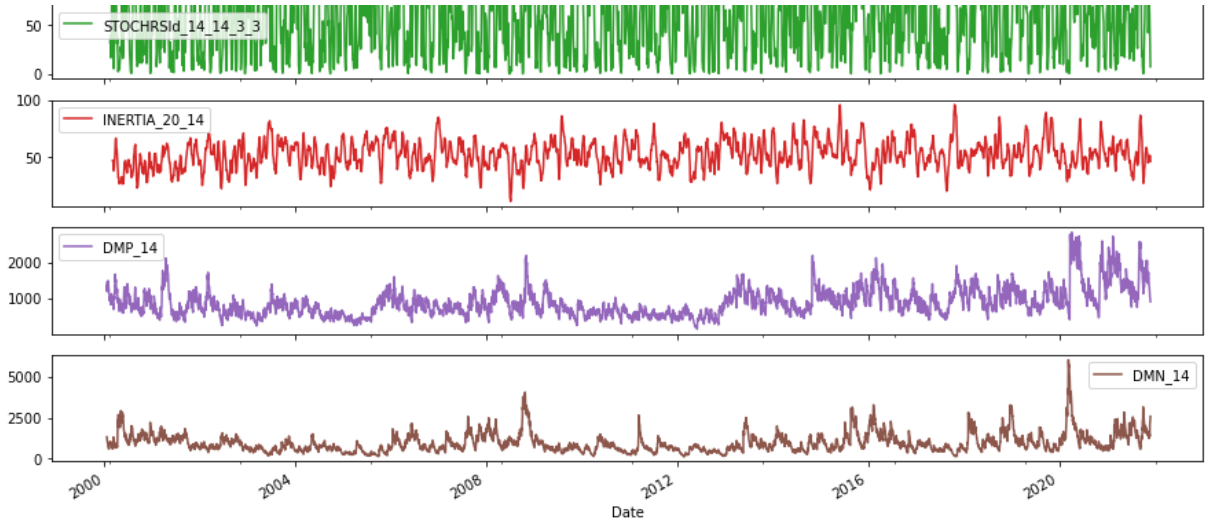
In [12]:
```
Japan = yf.download("^N225", start="2000-01-01", end="2021-11-30", interval="1D")
```

```
[*********************100%***********************]  1 of 1 completed
```

In [13]:

```
Japan.ta.rsi()
Japan.ta.macd(append=True)
Japan.ta.roc(append=True)
Japan.ta.willr(append=True)
Japan.ta.stochrsi(append=True)
Japan.ta.inertia(append=True)
Japan.ta.dm(append=True)

Japan.plot(subplots=True, figsize=(15,30))
plt.savefig(r"figures\JP_indicators.png", dpi=600)
```

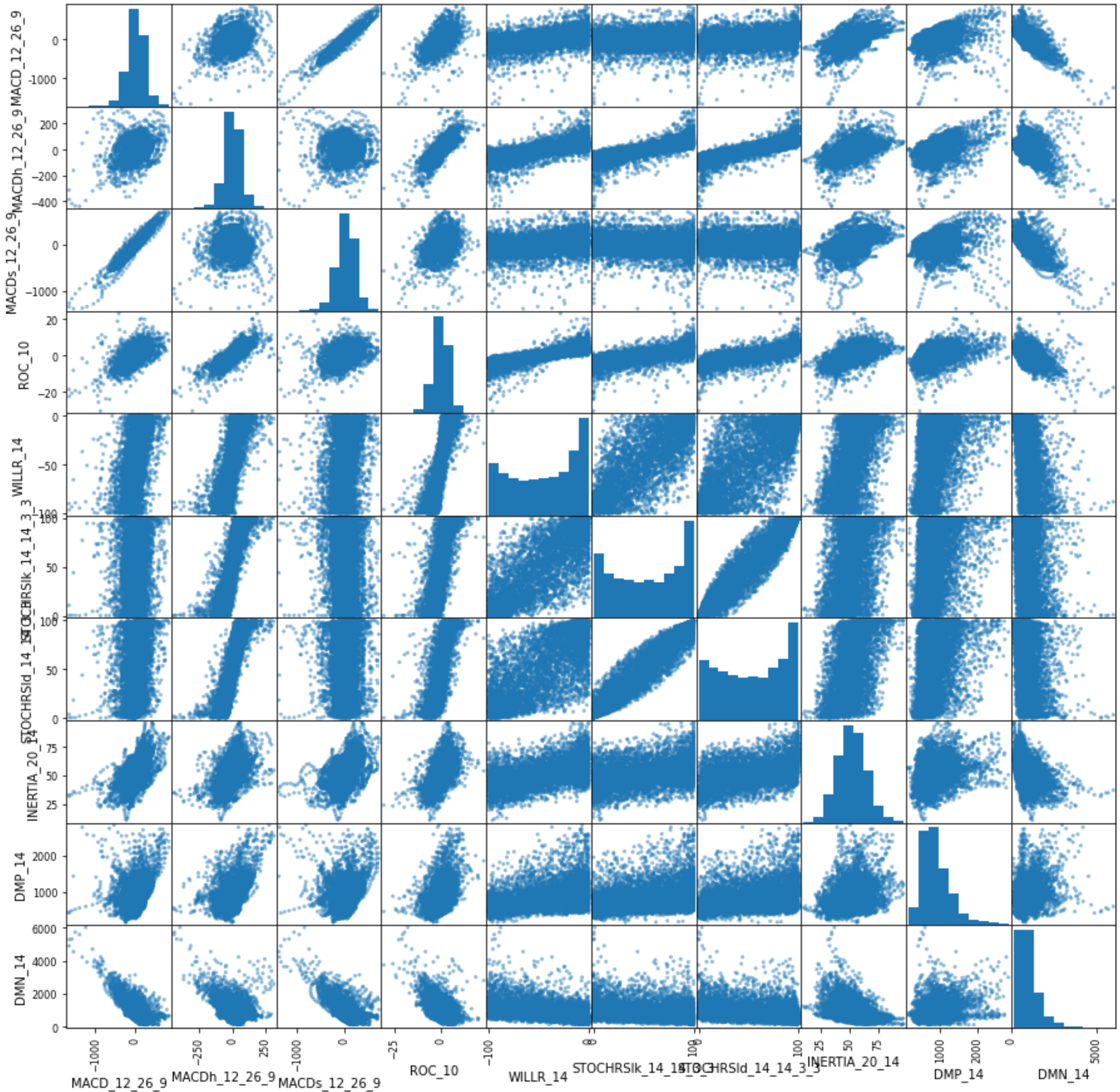




In [14]:
```
drops = ["Adj Close", "High", "Open", "Low", "Close", "Volume"]
Japan.drop(columns=drops, inplace=True)
Japan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 5369 entries, 2000-01-04 to 2021-11-30
Data columns (total 10 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   MACD_12_26_9         5336 non-null   float64
 1   MACDh_12_26_9        5336 non-null   float64
 2   MACDs_12_26_9        5336 non-null   float64
 3   ROC_10               5359 non-null   float64
 4   WILLR_14             5356 non-null   float64
 5   STOCHRSIk_14_14_3_3  5340 non-null   float64
 6   STOCHRSId_14_14_3_3  5338 non-null   float64
 7   INERTIA_20_14        5324 non-null   float64
 8   DMP_14               5356 non-null   float64
 9   DMN_14               5356 non-null   float64
dtypes: float64(10)
memory usage: 461.4 KB
```

In [15]:
```
pd.plotting.scatter_matrix(Japan, figsize=(15,15))
plt.savefig(r"figures\JP_indicators_scatterMatrix.png", dpi=600)
plt.show()
```



Build the Target

In [16]:
```
SPY = yf.download("^GSPC", start="2000-01-01", end="2021-11-30", interval="1D")
```

```
[*********************100%***********************]  1 of 1 completed
```

Below is my equation for building the target dataset. Data Methods: Price is too noisy to attempt to predict. Some say it is quite literally a fools errand. So instead of predicting price directly I have decided to use a smoothed proxy with built in error. The target I have constructed is a 3 class system. I used the S&P500 index (^GSPC) difference in future 30 day expotential moving average (EMA) subtracted by the current EMA-30, divided by the current EMA-30. If the value was within plus or minus 10% change it was labeled as 0, if the value was greater that 10% change if was labeled 1, and if it was negative 10% change it was labeled as -1. This is too insure that the model will perform with in reason.

In [17]:
```
SPY.ta.ema(length = 30, append=True)
SPY["EMA_30_FT"] = SPY.EMA_30.shift(periods=-30)
SPY["Diff"] = (SPY.EMA_30_FT - SPY.EMA_30)
SPY["Diff_ratio"] = SPY.Diff / SPY.EMA_30
SPY["Diff_shift"] = SPY.Diff_ratio.shift(periods=-30)
SPY["Target"] = SPY.Diff_shift.apply(lambda x: -1 if x < -0.01 else (1 if x > 0.0
SPY.head(60)
```

Out[17]:

Date	Open	High	Low	Close	Adj Close	Volume	EMA_30
2000-01-03	1469.250000	1478.000000	1438.359985	1455.219971	1455.219971	931800000	NaN
2000-01-04	1455.219971	1455.219971	1397.430054	1399.420044	1399.420044	1009000000	NaN
2000-01-05	1399.420044	1413.270020	1377.680054	1402.109985	1402.109985	1085500000	NaN
2000-01-06	1402.109985	1411.900024	1392.099976	1403.449951	1403.449951	1092300000	NaN
2000-01-07	1403.449951	1441.469971	1400.729980	1441.469971	1441.469971	1225200000	NaN
2000-01-10	1441.469971	1464.359985	1441.469971	1457.599976	1457.599976	1064800000	NaN
2000-01-11	1457.599976	1458.660034	1434.420044	1438.560059	1438.560059	1014000000	NaN
2000-01-12	1438.560059	1442.599976	1427.079956	1432.250000	1432.250000	974600000	NaN
2000-01-13	1432.250000	1454.199951	1432.250000	1449.680054	1449.680054	1030400000	NaN
2000-01-14	1449.680054	1473.000000	1449.680054	1465.150024	1465.150024	1085900000	NaN
2000-01-18	1465.150024	1465.150024	1451.300049	1455.140015	1455.140015	1056700000	NaN
2000-01-19	1455.140015	1461.390015	1448.680054	1455.900024	1455.900024	1087800000	NaN
2000-01-20	1455.900024	1465.709961	1438.540039	1445.569946	1445.569946	1100700000	NaN
2000-01-21	1445.569946	1453.180054	1439.599976	1441.359985	1441.359985	1209800000	NaN
2000-01-24	1441.359985	1454.089966	1395.420044	1401.530029	1401.530029	1115800000	NaN
2000-01-25	1401.530029	1414.260010	1388.489990	1410.030029	1410.030029	1073700000	NaN
2000-01-26	1410.030029	1412.729980	1400.160034	1404.089966	1404.089966	1117300000	NaN
2000-01-27	1404.089966	1418.859985	1370.989990	1398.560059	1398.560059	1129500000	NaN
2000-01-28	1398.560059	1398.560059	1356.199951	1360.160034	1360.160034	1095800000	NaN

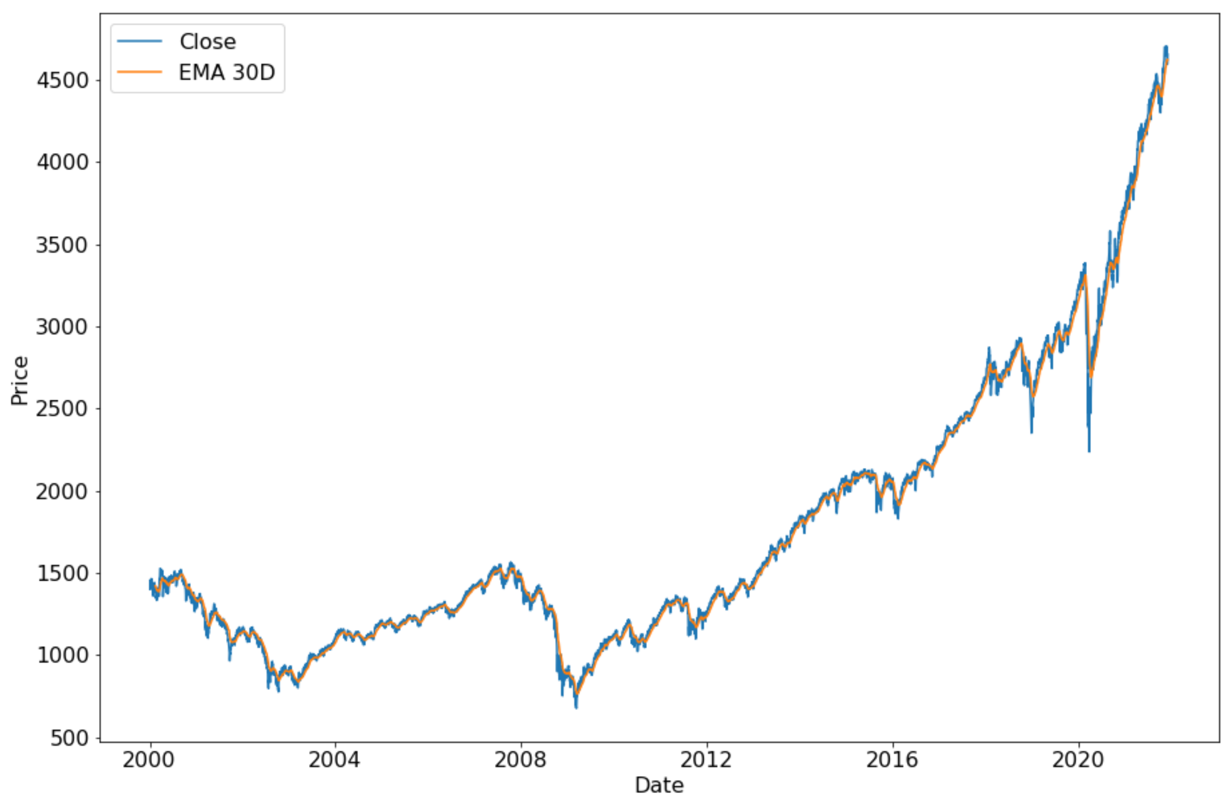
Date	Open	High	Low	Close	Adj Close	Volume	EMA_30
2000-01-31	1360.160034	1394.479980	1350.140015	1394.459961	1394.459961	993800000	NaN
2000-02-01	1394.459961	1412.489990	1384.790039	1409.280029	1409.280029	981000000	NaN
2000-02-02	1409.280029	1420.609985	1403.489990	1409.119995	1409.119995	1038600000	NaN
2000-02-03	1409.119995	1425.780029	1398.520020	1424.969971	1424.969971	1146500000	NaN
2000-02-04	1424.969971	1435.910034	1420.630005	1424.369995	1424.369995	1045100000	NaN
2000-02-07	1424.369995	1427.150024	1413.329956	1424.239990	1424.239990	918100000	NaN
2000-02-08	1424.239990	1441.829956	1424.239990	1441.719971	1441.719971	1047700000	NaN
2000-02-09	1441.719971	1444.550049	1411.650024	1411.709961	1411.709961	1050500000	NaN
2000-02-10	1411.699951	1422.099976	1406.430054	1416.829956	1416.829956	1058800000	NaN
2000-02-11	1416.829956	1416.829956	1378.890015	1387.119995	1387.119995	1025700000	NaN
2000-02-14	1387.119995	1394.930054	1380.530029	1389.939941	1389.939941	927300000	1421.700330
2000-02-15	1389.939941	1407.719971	1376.250000	1402.050049	1402.050049	1092100000	1420.432570
2000-02-16	1402.050049	1404.550049	1385.579956	1387.670044	1387.670044	1018800000	1418.318858
2000-02-17	1387.670044	1399.880005	1380.069946	1388.260010	1388.260010	1034800000	1416.379578
2000-02-18	1388.260010	1388.589966	1345.319946	1346.089966	1346.089966	1042300000	1411.844764
2000-02-22	1346.089966	1358.109985	1331.880005	1352.170044	1352.170044	980000000	1407.994782
2000-02-23	1352.170044	1370.109985	1342.439941	1360.689941	1360.689941	993700000	1404.942857
2000-02-24	1360.689941	1364.800049	1329.880005	1353.430054	1353.430054	1215000000	1401.619450
2000-02-25	1353.430054	1362.140015	1329.150024	1333.359985	1333.359985	1065200000	1397.215614
2000-02-28	1333.359985	1360.819946	1325.069946	1348.050049	1348.050049	1026500000	1394.043642
2000-02-29	1348.050049	1369.630005	1348.050049	1366.420044	1366.420044	1204300000	1392.261474
2000-03-01	1366.420044	1383.459961	1366.420044	1379.189941	1379.189941	1274100000	1391.418150
2000-03-02	1379.189941	1386.560059	1370.349976	1381.760010	1381.760010	1198600000	1390.795044

	Open	High	Low	Close	Adj Close	Volume	EMA_30
Date							
2000-03-03	1381.760010	1410.880005	1381.760010	1409.170044	1409.170044	1150300000	1391.980528
2000-03-06	1409.170044	1409.739990	1384.750000	1391.280029	1391.280029	1029000000	1391.935334
2000-03-07	1391.280029	1399.209961	1349.989990	1355.619995	1355.619995	1314100000	1389.592409
2000-03-08	1355.619995	1373.790039	1346.619995	1366.699951	1366.699951	1203000000	1388.115476
2000-03-09	1366.699951	1401.819946	1357.880005	1401.689941	1401.689941	1123000000	1388.991248
2000-03-10	1401.689941	1413.459961	1392.069946	1395.069946	1395.069946	1138800000	1389.383422
2000-03-13	1395.069946	1398.390015	1364.839966	1383.619995	1383.619995	1016100000	1389.011588
2000-03-14	1383.619995	1395.150024	1359.150024	1359.150024	1359.150024	1094000000	1387.085036
2000-03-15	1359.150024	1397.989990	1356.989990	1392.140015	1392.140015	1302800000	1387.411163
2000-03-16	1392.150024	1458.469971	1392.150024	1458.469971	1458.469971	1482300000	1391.995603
2000-03-17	1458.469971	1477.329956	1453.319946	1464.469971	1464.469971	1295100000	1396.671368
2000-03-20	1464.469971	1470.300049	1448.489990	1456.630005	1456.630005	920800000	1400.539667
2000-03-21	1456.630005	1493.920044	1446.060059	1493.869995	1493.869995	1065900000	1406.560979
2000-03-22	1493.869995	1505.079956	1487.329956	1500.640015	1500.640015	1075000000	1412.630594
2000-03-23	1500.640015	1532.500000	1492.390015	1527.349976	1527.349976	1078300000	1420.031845
2000-03-24	1527.349976	1552.869995	1516.829956	1527.459961	1527.459961	1052200000	1426.962691
2000-03-27	1527.459961	1534.630005	1518.459961	1523.859985	1523.859985	901000000	1433.214129
2000-03-28	1523.859985	1527.359985	1507.089966	1507.729980	1507.729980	959100000	1438.021603

In [18]:
```
Target = SPY.Target
```

In [19]:
```
import yfinance as yf

SP500 = yf.download("^GSPC", start="2000-01-01", end="2021-11-30", interval="1D")
SP500.ta.ema(length=30, append=True)

fig, ax = plt.subplots(figsize=(15,10))

# Set general font size
plt.rcParams['font.size'] = '16'

# Set tick font size
for label in (ax.get_xticklabels() + ax.get_yticklabels()):
    label.set_fontsize(16)

plt.plot(SP500.Close, color="tab:blue", label="Close")
plt.plot(SP500.EMA_30, color="tab:orange", label="EMA 30D")
plt.legend(fontsize=16)
plt.ylabel("Price", fontsize=16)
plt.xlabel("Date", fontsize=16)
fig.savefig(r"figures\SP500_EMA30.png", dpi=600)
```

```
[*********************100%***********************]  1 of 1 completed
```



Building Data Matrix

In [20]:
```
data = pd.concat([AMD, CrudeOil, Japan, Target], axis=1)
data.dropna(axis=0, inplace=True)
data.head(10)
```

Out[20]:

Date	RSI_14	MACD_12_26_9	MACDh_12_26_9	MACDs_12_26_9	ROC_10	WILLR_14	STOCH
2000-10-26	40.485414	-1.986240	0.206340	-2.192581	-9.972299	-57.961783	
2000-10-27	40.978423	-1.885969	0.245290	-2.131258	-6.571429	-56.687898	
2000-10-30	40.437308	-1.800873	0.264308	-2.065181	-0.613497	-58.598726	
2000-10-31	49.528615	-1.524221	0.432768	-1.956989	23.129252	-34.394904	
2000-11-01	53.361708	-1.195371	0.609294	-1.804666	31.379310	-5.426357	
2000-11-02	53.132990	-0.929089	0.700461	-1.629550	8.882521	-18.243243	
2000-11-06	55.139962	-0.485165	0.768360	-1.253525	11.142857	-12.162162	
2000-11-07	48.221425	-0.443183	0.648274	-1.091457	11.764706	-42.990654	
2000-11-08	45.169672	-0.475040	0.493133	-0.968174	13.029316	-56.074766	
2000-11-09	42.873785	-0.549430	0.334995	-0.884425	3.384615	-66.355140	

10 rows × 33 columns

Target practice

In [21]:
```
data.info()
```

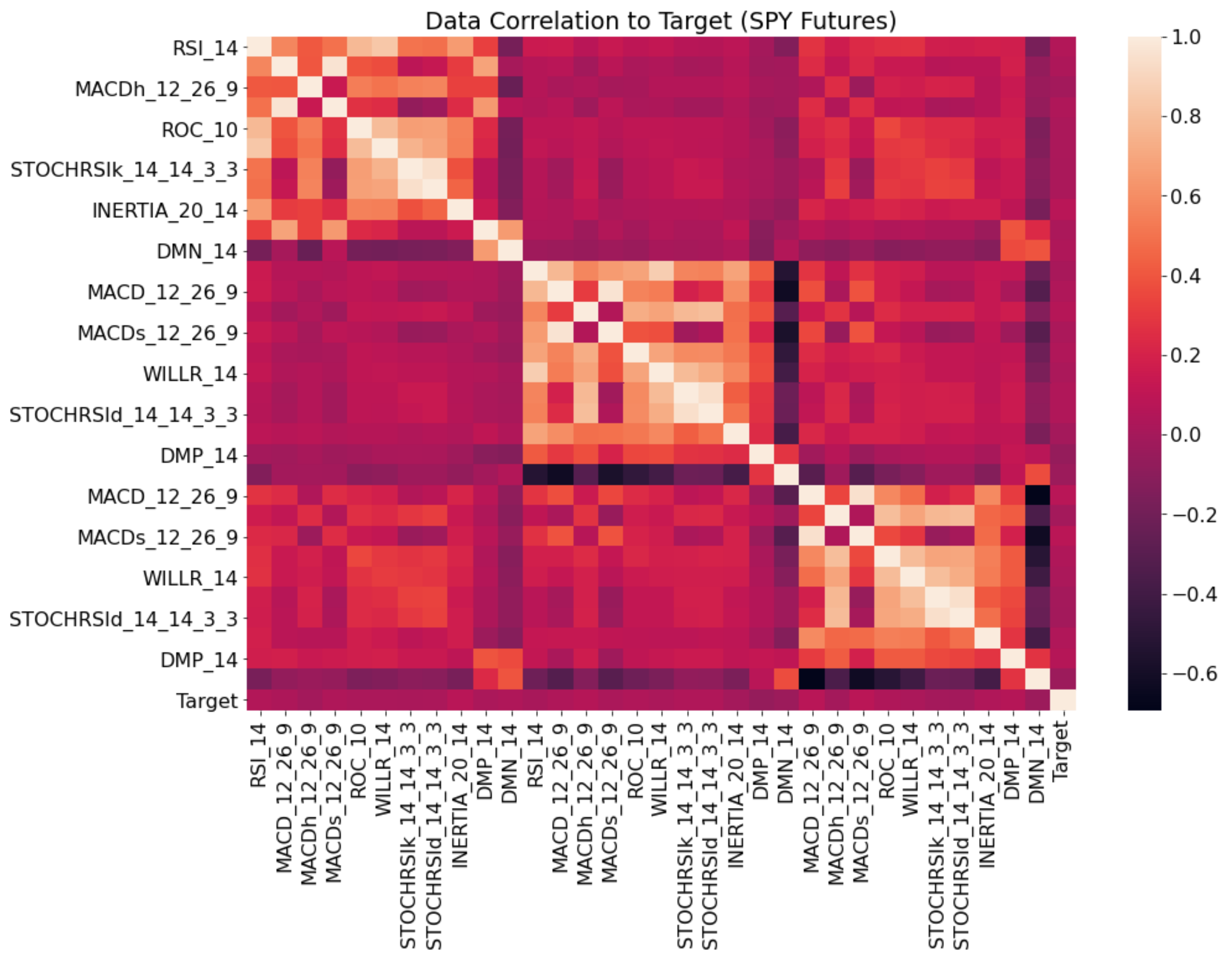
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 4981 entries, 2000-10-26 to 2021-11-29
Data columns (total 33 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   RSI_14               4981 non-null   float64
 1   MACD_12_26_9         4981 non-null   float64
 2   MACDh_12_26_9        4981 non-null   float64
 3   MACDs_12_26_9        4981 non-null   float64
 4   ROC_10               4981 non-null   float64
 5   WILLR_14             4981 non-null   float64
 6   STOCHRSIk_14_14_3_3  4981 non-null   float64
 7   STOCHRSId_14_14_3_3  4981 non-null   float64
 8   INERTIA_20_14        4981 non-null   float64
 9   DMP_14               4981 non-null   float64
 10  DMN_14               4981 non-null   float64
 11  RSI_14               4981 non-null   float64
 12  MACD_12_26_9         4981 non-null   float64
 13  MACDh_12_26_9        4981 non-null   float64
 14  MACDs_12_26_9        4981 non-null   float64
 15  ROC_10               4981 non-null   float64
 16  WILLR_14             4981 non-null   float64
 17  STOCHRSIk_14_14_3_3  4981 non-null   float64
 18  STOCHRSId_14_14_3_3  4981 non-null   float64
 19  INERTIA_20_14        4981 non-null   float64
 20  DMP_14               4981 non-null   float64
 21  DMN_14               4981 non-null   float64
 22  MACD_12_26_9         4981 non-null   float64
 23  MACDh_12_26_9        4981 non-null   float64
 24  MACDs_12_26_9        4981 non-null   float64
 25  ROC_10               4981 non-null   float64
 26  WILLR_14             4981 non-null   float64
 27  STOCHRSIk_14_14_3_3  4981 non-null   float64
 28  STOCHRSId_14_14_3_3  4981 non-null   float64
 29  INERTIA_20_14        4981 non-null   float64
 30  DMP_14               4981 non-null   float64
 31  DMN_14               4981 non-null   float64
 32  Target               4981 non-null   float64
dtypes: float64(33)
memory usage: 1.3 MB
```

In [22]:
```python
Target = data["Target"]
data_matrix = data.drop(columns = "Target"); print(data_matrix.iloc[1])
data_matrix = data_matrix.to_numpy()
```

```
RSI_14                    40.978423
MACD_12_26_9              -1.885969
MACDh_12_26_9              0.245290
MACDs_12_26_9             -2.131258
ROC_10                    -6.571429
WILLR_14                 -56.687898
STOCHRSIk_14_14_3_3       73.248662
STOCHRSId_14_14_3_3       71.327084
INERTIA_20_14             40.858506
DMP_14                     5.262821
DMN_14                     8.288629
RSI_14                    48.221687
MACD_12_26_9               0.056485
MACDh_12_26_9             -0.023714
MACDs_12_26_9              0.080199
ROC_10                    -6.704703
WILLR_14                 -90.146726
STOCHRSIk_14_14_3_3       20.220842
STOCHRSId_14_14_3_3       31.821434
INERTIA_20_14             38.788608
DMP_14                     4.708909
DMN_14                     4.025638
MACD_12_26_9            -344.471385
MACDh_12_26_9            -53.290418
MACDs_12_26_9           -291.180967
ROC_10                    -4.879937
WILLR_14                 -99.637923
STOCHRSIk_14_14_3_3       24.914794
STOCHRSId_14_14_3_3       33.249024
INERTIA_20_14             36.102749
DMP_14                   660.817588
DMN_14                  1588.395090
Name: 2000-10-27 00:00:00, dtype: float64
```

In [23]:

```python
import seaborn as sns

plt.figure(figsize=(15,10))
sns.heatmap(data.corr())
plt.xlabel('')
plt.ylabel('')
plt.title('Data Correlation to Target (SPY Futures)')
plt.show()
```



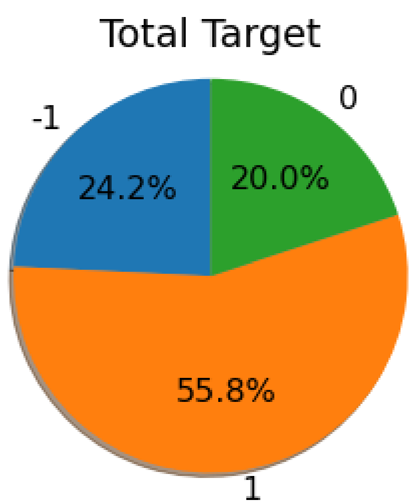
I don't see any multi-colinearity in the Data Correlation plot above. This is a good thing! However, the correlation to the target is pretty low around 20%. We'll see what happens...

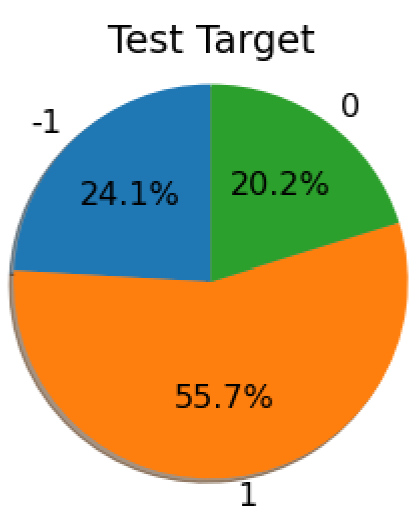
In [24]:

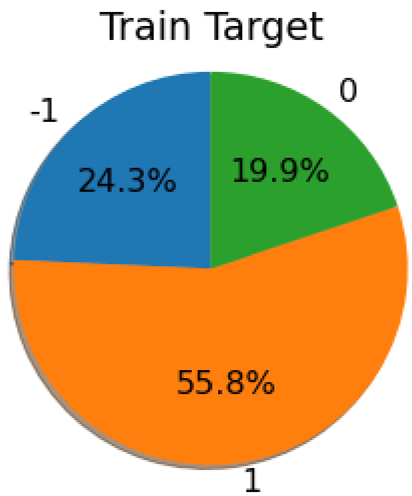
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix

from sklearn.metrics import ConfusionMatrixDisplay

import numpy as np

from sklearn.model_selection import train_test_split

def create_splits(X, y):
    return train_test_split(X, y, test_size=0.30, random_state=2)

X_train, X_test, y_train, y_test = create_splits(data_matrix, Target)

print(f'Training sample: {X_train.shape[0]:,}')
print(f'Test sample: {X_test.shape[0]:,}')
```

```
Training sample: 3,486
Test sample: 1,495
```

In [25]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import PercentFormatter

import matplotlib.pyplot as plt

# Pie chart, where the slices will be ordered and plotted counter-clockwise:
def pie_chart(dataset, **kwargs):
    labels = '-1', '1', '0'
    sizes = [(len(dataset[dataset==-1])/len(dataset)), (len(dataset[dataset==1])/

    fig1, ax1 = plt.subplots()
    ax1.pie(sizes, labels=labels, autopct='%1.1f%%',
            shadow=True, startangle=90)
    ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle
    if "title" in kwargs: ax1.set_title(kwargs["title"])
    plt.show()

pie_chart(Target, title="Total Target")
pie_chart(y_test, title="Test Target")
pie_chart(y_train, title="Train Target")
```



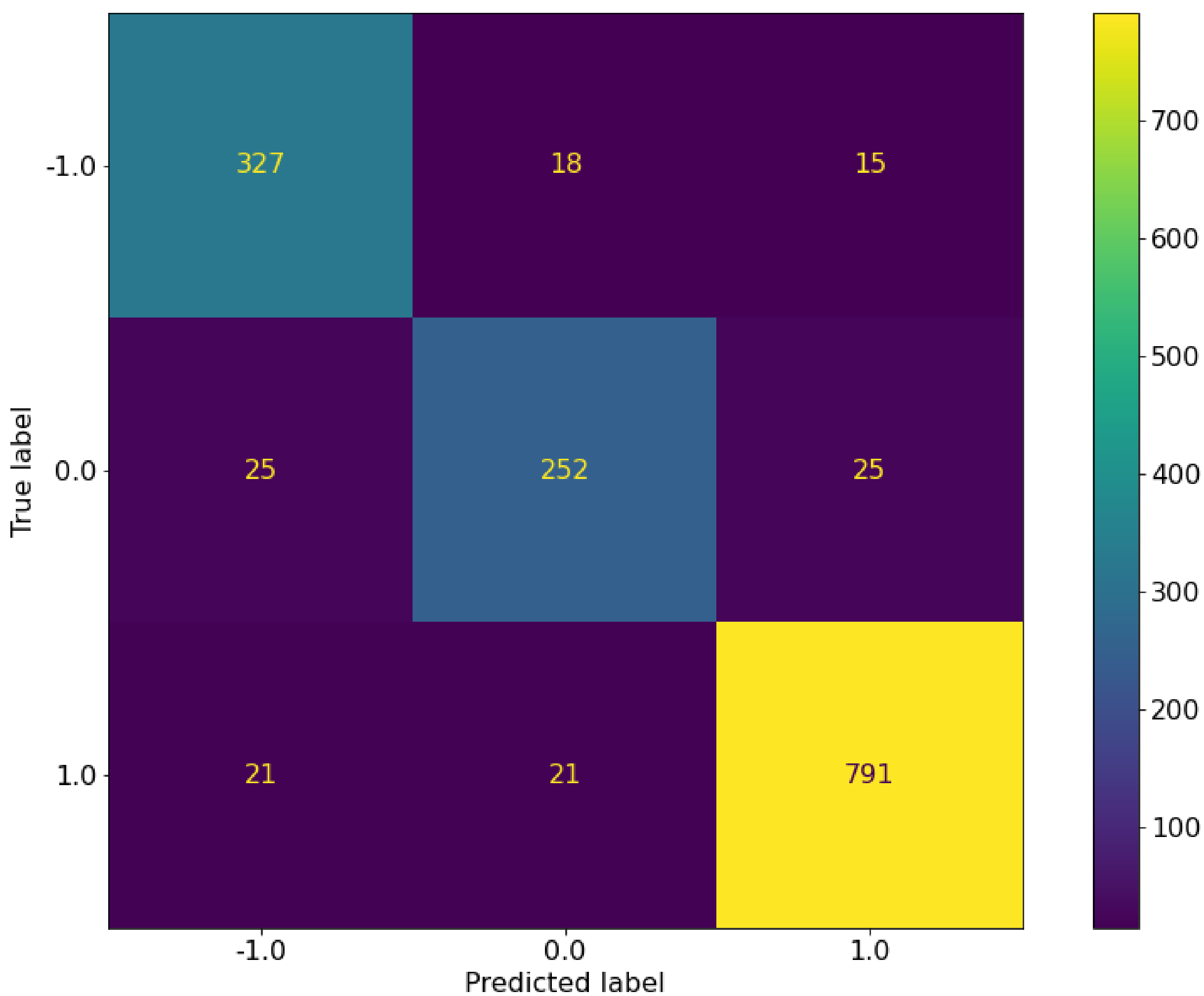




In [26]:
```
svm_m = modeling_pipeline = Pipeline([('scaling', StandardScaler()),
                                      ('pca', PCA()),
                                      ('model', SVC()),
                                      ])

param_grid = [{'model__C': [1, 10, 100, 1000],
               'pca__n_components': [None,1,5,10,15],
               'model__kernel': ['rbf'],
               'model__probability':[True, False]
              }]

scoring = ['f1_macro']

svm_results = GridSearchCV(estimator=svm_m, param_grid=param_grid, scoring=sco
svm_results = svm_results.fit(X_train, y_train)
```

In [27]:
```
# svm_score = svm_results.best_score_
print(f'Support Vector Machine Score: {svm_results.best_score_:.2%}')
print(svm_results.best_estimator_, svm_results.best_params_)
svm = svm_results.best_estimator_

y_pred = svm.predict(X_test)


#Get the confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)

fig, ax = plt.subplots(figsize=(15,10))
# ax.subtitle("Best Estimator Confusion Matrix")
disp = ConfusionMatrixDisplay(confusion_matrix=cf_matrix,
                              display_labels=svm_results.best_estimator_.named_st
disp.plot(ax=ax)
plt.show()
```

```
Support Vector Machine Score: 84.07%
Pipeline(steps=[('scaling', StandardScaler()), ('pca', PCA()),
                ('model', SVC(C=100, probability=True))]) {'model__C': 100, 'mo
del__kernel': 'rbf', 'model__probability': True, 'pca__n_components': None}
```



In [28]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

        -1.0       0.88      0.91      0.89       360
         0.0       0.87      0.83      0.85       302
         1.0       0.95      0.95      0.95       833

    accuracy                           0.92      1495
   macro avg       0.90      0.90      0.90      1495
weighted avg       0.92      0.92      0.92      1495
```

K-Nearest Neighbors (KNN)

In [29]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import ConfusionMatrixDisplay

knn_m = modeling_pipeline = Pipeline([('scaling', StandardScaler()),
                                      ('pca', PCA(n_components=None)),
                                      ('model', KNeighborsClassifier())])



param_grid = [
  {'model__n_neighbors': [1, 4, 8, 10, 15],
   'pca__n_components': [None,1,2,3,4,5,10,15],
   'model__weights': ['uniform','distance']}
 ]

knn_results = GridSearchCV(estimator=knn_m, param_grid=param_grid, scoring='f1_ma
knn_results = knn_results.fit(X_train, y_train)
```

In [33]:
```
knn_score = knn_results.score(X_test, y_test)
print(f'k-Nearest Neighbor Score: {knn_score:.2%}')
print(knn_results.best_estimator_)
```

```
k-Nearest Neighbor Score: 92.15%
Pipeline(steps=[('scaling', StandardScaler()), ('pca', PCA()),
                ('model', KNeighborsClassifier(n_neighbors=1))])
```

Decision Tree (DT)

In [34]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

p2 = Pipeline([('scaling', StandardScaler()),
               ('pca', PCA(n_components=None)),
               ('dt', DecisionTreeClassifier())])

params = {'dt__max_depth': [15, 25, 50, 100],
          'pca__n_components': [None,10,15,20],
          'dt__min_samples_split': [2, 3, 5, 10, 15]}

dt_gscv = GridSearchCV(p2, param_grid=params, cv=5, scoring='accuracy', refit=Tru
dt_gscv = dt_gscv.fit(X_train, y_train)
```

In [35]:
```python
print(f"Best Estimator: {dt_gscv.best_estimator_}")
print(f'Validation score: {dt_gscv.best_score_:.2%}')

dt_pred = dt_gscv.predict(X_test)

print(f'Test score: {dt_gscv.score(X_test, y_test):.2%}')
```

```
Best Estimator: Pipeline(steps=[('scaling', StandardScaler()), ('pca', PCA(n_co
mponents=15)),
                ('dt', DecisionTreeClassifier(max_depth=25))])
Validation score: 66.15%
Test score: 67.36%
```

Random Forest (RF)

In [36]:
```python
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

rf_pipeline = Pipeline([('scaling', StandardScaler()),
                        ('pca', PCA(n_components=None)),
                        ('rf', RandomForestClassifier())
                       ])

param_grid = [{'rf__max_depth': [10, 15, 20],
               'rf__n_estimators': [25, 50, 100],
               'rf__class_weight': [None, 'balanced', 'balanced_subsample'],
               'pca__n_components': [None,10,15,20]
              }]

rf_results = GridSearchCV(estimator=rf_pipeline, param_grid=param_grid, scoring='
rf_results = rf_results.fit(X_train, y_train)
rf_yhat = rf_results.predict(X_test)

rf_results.best_estimator_
```

Out[36]:
```
Pipeline(steps=[('scaling', StandardScaler()), ('pca', PCA(n_components=15)),
                ('rf',
                 RandomForestClassifier(class_weight='balanced',
                                        max_depth=15))])
```

In [37]:
```python
y_testp = rf_results.predict(X_test)
y_testp_rf = rf_results.predict_proba(X_test)


print(f'Validation score: {rf_results.best_score_:.2%}')
print(f'Test score: {rf_results.score(X_test, y_test):.2%}')


from sklearn.metrics import classification_report
print(classification_report(y_test, y_testp))
```

```
Validation score: 72.95%
Test score: 76.87%
              precision    recall  f1-score   support

        -1.0       0.92      0.65      0.76       360
         0.0       0.80      0.58      0.67       302
         1.0       0.79      0.97      0.87       833

    accuracy                           0.82      1495
   macro avg       0.84      0.73      0.77      1495
weighted avg       0.83      0.82      0.81      1495
```

Naive Bayes (NB)

In [40]:
```python
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.model_selection import cross_val_score

nb_model = BernoulliNB()

nb_results = nb_model.fit(X_train, y_train)
nb_cv_score = cross_val_score(nb_results, X_train, y_train, cv=5)
```

In [41]:
```python
print("%0.2f accuracy with a standard deviation of %0.2f" % (nb_cv_score.mean(),
```

```
0.54 accuracy with a standard deviation of 0.01
```

This score might be able to inprove with some tuning of hyperparameters, however I only want an ensemble of 3 and I think I've found the candidates.

Ensemble

Taking 3 best scorers

In [42]:
```
from sklearn.ensemble import VotingClassifier

ems = [('knn', knn_results.best_estimator_),('rf', rf_results.best_estimator_),('
en = VotingClassifier(estimators=ems, weights=None, voting='soft')
en = en.fit(X_train, y_train)
scores = cross_val_score(estimator=en, X=X_train, y=y_train, cv=10, scoring='roc_
print(f'ROC AUC OVO {scores.mean():.2f} (+/- {scores.std():.2f}) [Ensemble]')
```

```
ROC AUC OVO 0.97 (+/- 0.01) [Ensemble]
```

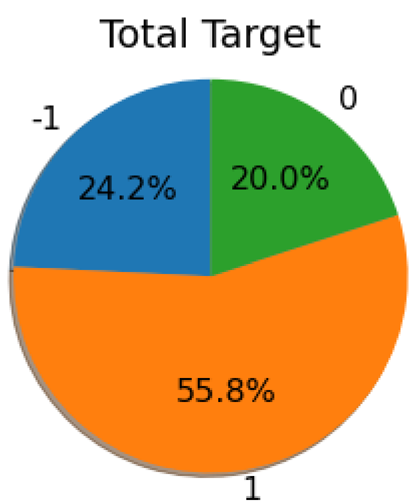
In [43]:
```
y_testp = en.predict(X_test)
y_testp_rf = en.predict_proba(X_test)

print(f'Test score: {en.score(X_test, y_test):.2%}')

from sklearn.metrics import classification_report
print(classification_report(y_test, y_testp))
```

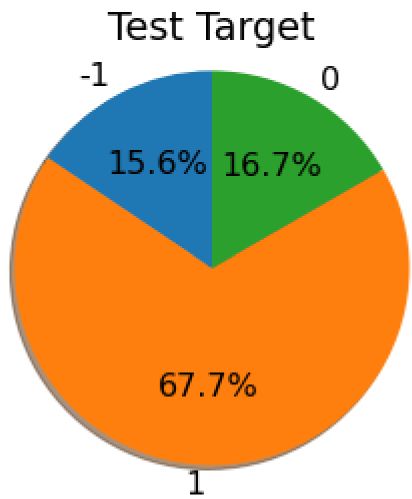
```
Test score: 94.11%
              precision    recall  f1-score   support

        -1.0       0.93      0.94      0.94       360
         0.0       0.89      0.86      0.88       302
         1.0       0.96      0.97      0.97       833

    accuracy                           0.94      1495
   macro avg       0.93      0.92      0.93      1495
weighted avg       0.94      0.94      0.94      1495
```

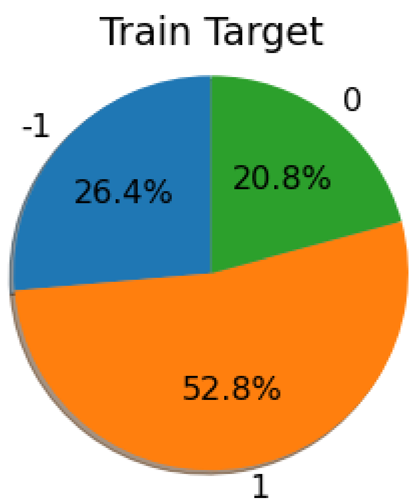
Here is where I decided to use a new data split method. Since I want the model to run forward in time as new stock information comes in I want it to be training to work in that direction. I split the data first into past and present-ish data. Then I use the skleaern training split on the past data and use the present-ish data as a prediction set for after the model has been tested. Note: I strongly believe this method can be more efficient!

In [46]:
```python
def create_timeseries_split(dataframe, test_size):
    stop = int(len(dataframe) * (1-test_size));

    Train =  dataframe.iloc[0:stop]
    Test = dataframe.iloc[stop+1:-1]

    Xtrain = Train.drop(columns="Target")
    ytrain = Train.Target

    Xtest = Test.drop(columns="Target")
    ytest = Test.Target

    return Xtrain, Xtest, ytrain, ytest

X_train, X_test, y_train, y_test = create_timeseries_split(data, 0.20)

print(f'Training sample: {X_train.shape[0]:,}')
print(f'Test sample: {X_test.shape[0]:,}')

pie_chart(Target, title="Total Target")
pie_chart(y_test, title="Test Target")
pie_chart(y_train, title="Train Target")
```

```
Training sample: 3,984
Test sample: 995
```







In [47]:
```
en = en.fit(X_train, y_train)
scores = cross_val_score(estimator=en, X=X_train, y=y_train, cv=10, scoring='roc_
print(f'ROC AUC OVO {scores.mean():.2f} (+/- {scores.std():.2f}) [Ensemble]')
```

```
ROC AUC OVO 0.42 (+/- 0.09) [Ensemble]
```

In [48]:
```
y_testp = en.predict(X_test)
y_testp_rf = en.predict_proba(X_test)

print(f'Test score: {en.score(X_test, y_test):.2%}')

from sklearn.metrics import classification_report
print(classification_report(y_test, y_testp))
```

```
Test score: 28.44%
              precision    recall  f1-score   support

        -1.0       0.06      0.17      0.09       155
         0.0       0.25      0.23      0.24       166
         1.0       0.54      0.32      0.41       674

    accuracy                           0.28       995
   macro avg       0.28      0.24      0.24       995
weighted avg       0.42      0.28      0.33       995
```

In [49]:
```
xX_train, xX_test, yy_train, yy_test = train_test_split(X_train, y_train, test_si

print(f'Training sample: {xX_train.shape[0]:,}')
print(f'Test sample: {xX_test.shape[0]:,}')
```

```
Training sample: 3,187
Test sample: 797
```

In [50]:
```
en_1 = en.fit(xX_train, yy_train)
scores = cross_val_score(estimator=en_1, X=xX_train, y=yy_train, cv=10, scoring='
print(f'ROC AUC OVO {scores.mean():.2f} (+/- {scores.std():.2f}) [Ensemble]')
```

```
ROC AUC OVO 0.99 (+/- 0.01) [Ensemble]
```

In [51]:
```
y_testp = en_1.predict(xX_test)
y_testp_rf = en_1.predict_proba(xX_test)

print(f'Test score: {en_1.score(xX_test, yy_test):.2%}')

from sklearn.metrics import classification_report
print(classification_report(yy_test, y_testp))
```

```
Test score: 95.36%
              precision    recall  f1-score   support

        -1.0       0.94      0.94      0.94       205
         0.0       0.91      0.91      0.91       170
         1.0       0.98      0.98      0.98       422

    accuracy                           0.95       797
   macro avg       0.94      0.94      0.94       797
weighted avg       0.95      0.95      0.95       797
```

In [52]:
```python
y_testp = en_1.predict(X_test)
y_testp_rf = en_1.predict_proba(X_test)

print(f'Test score: {en_1.score(X_test, y_test):.2%}')

from sklearn.metrics import classification_report
print(classification_report(y_test, y_testp))
```

```
Test score: 26.83%
              precision    recall  f1-score   support

        -1.0       0.06      0.16      0.08       155
         0.0       0.22      0.21      0.22       166
         1.0       0.52      0.31      0.39       674

    accuracy                           0.27       995
   macro avg       0.27      0.23      0.23       995
weighted avg       0.40      0.27      0.31       995
```

Hopefully, you have followed the results.

- SVM scored: 84%
- KNN scored: 92%
- DT scored: 67%
- RF scored: 76%
- NB scored: 54%

The models are doing well on the training data however, I think this may be due to the random selection used in the sklearn training split function. The strongest performers are really good as slicing data and drawing boundaries and when the data is random selected from the training set its easier to see patterns of where to draw boundaries. This is a possible explanation for the low scores seen in the ensemble model of the best estimators. There is also issue of possibly insufficient data prodvided. The full use of this process may yield better results. see ProjectNotebook.ipynb