

## ✓ Programming Assignment 3

UP feature and ensemble models exploration continues from the jump-start code below.

## MSDS 451 Feature Engineering

### Financial Machine Learning: Adding Features to the Mix

Thomas W. Miller, August 25, 2025

## ✓ Overview

We again use machine learning classifiers, including tree-based ensemble boosting and bagging methods, to predict the direction of oil futures (up or down) using a number of lagged price features. In particular, we look at daily closing spot prices for [West Texas Intermediate \(ticker WTI\)](#) with lags of one to seven days, as well as features based on opening, closing, high, and low price points, and daily trading volume.

Tree-based ensemble models for predicting the direction of daily returns set the stage for testing the predictive utility of additional features. The domain of potential features or leading indicators is wide, including those associated with other price series, economic indicators, international events, securities filings, analyst and news reports, and media measures. Here we explore a set of nine features defined from a range of exchange-traded funds (ETFs).

The model-building process demonstrated here can be employed for any asset or portfolio of assets.

The model building process involves these steps:

- Define price-based features for the target asset.
- Define binary features (Up or not Up) for other assets, economic measures, market measures, worldwide events, and media measures.
- Fit an ensemble model to the full set of features within a time series cross-validation design using a tree-structured ensemble with hyperparameters set to their default values. Note the test set classification performance in predicting movement (Up or not Up) in the target asset.
- Utilized randomized search across key hyperparameters to determine "best" settings.
- Use a hold-out test set to evaluate model performance with "best" hyperparameter settings.
- Rank the importance of features from the model evaluation, and select the best features for subsequent model development.
- Repeat the model-building process, adding new features to the mix.

## ✓ Import Libraries

We draw on Python packages for data manipulation and modeling. Most important are Polars, a high-performance alternative to Pandas for data manipulation, and Scikit-Learn for machine learning study design and modeling algorithms.

```
!pip install catboost
```

```
🔄 Collecting catboost
  Downloading catboost-1.2.8-cp312-cp312-manylinux2014_x86_64.whl.metadata (1.2 kB)
Requirement already satisfied: graphviz in /usr/local/lib/python3.12/dist-packages (from catboost) (0.21)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (from catboost) (3.10.0)
Requirement already satisfied: numpy<3.0,>=1.16.0 in /usr/local/lib/python3.12/dist-packages (from catboost) (2.0.2)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.12/dist-packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-packages (from catboost) (1.16.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.12/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.12/dist-packages (from catboost) (1.17.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas>=0.24->catboost) (2.9.0.pc
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas>=0.24->catboost) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas>=0.24->catboost) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib->catboost) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib->catboost) (4.59.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib->catboost) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib->catboost) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib->catboost) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib->catboost) (3.2.3)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.12/dist-packages (from plotly->catboost) (8.5.0)
```

Downloading catboost-1.2.8-cp312-cp312-manylinux2014\_x86\_64.whl (99.2 MB)  
99.2/99.2 MB 7.9 MB/s eta 0:00:00

Installing collected packages: catboost  
Successfully installed catboost-1.2.8

```
import os
# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter(action='ignore', category=FutureWarning)

# datetime functions needed for filtering across DataFrames with differing time frame
from datetime import datetime, timezone

# Import Python Packages for data manipulation, data pipelines, and databases
import numpy as np
import pyarrow # foundation for polars
import polars as pl # DataFrame work superior to Pandas

# Plotting
import matplotlib.pyplot as plt
# Display static plots directly in the notebook output
%matplotlib inline
# create stylized visualizations, including heat maps
import seaborn as sns

# Preprocessing
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import (RandomizedSearchCV,
                                     TimeSeriesSplit)
from sklearn.model_selection import cross_validate

# utilized in all possible subsets classification work
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss

# needed for randomized search
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform

# metrics in xgboost tuning and final model evaluation
from sklearn.metrics import (accuracy_score,
                             classification_report,
                             roc_curve,
                             roc_auc_score,
                             RocCurveDisplay,
                             ConfusionMatrixDisplay,
                             confusion_matrix,
                             precision_score,
                             recall_score,
                             f1_score
                             )

from sklearn.ensemble import AdaBoostClassifier

from catboost import CatBoostClassifier

# XGBoost Package... more complete than SciKit-Learn boosting methods
import xgboost as xgb
from xgboost import XGBRegressor, XGBClassifier, plot_importance

from sklearn.ensemble import RandomForestClassifier

import warnings
# Suppress warnings for cleaner output
warnings.filterwarnings('ignore')
```

## ✓ Retrieve Data

In previous work, we retrieved price data for WTI and nine ETFs from Yahoo Finance. The code for WTI is shown in the next commented-out cell.

```

'''
Previous work to retrieve data from Yahoo Finance had this structure for each of the ETFs in the study

symbol = 'WTI'
start_date = '2000-01-01'
end_date = '2025-08-19'

ticker = yf.Ticker(symbol)
historical_data = ticker.history(start = start_date, end = end_date, period = '1mo')
print(historical_data)

print("type of historical_data", type(historical_data))

historical_data.to_csv("wti_daily_data.csv")
'''

'''
Previous work to retrieve data from Yahoo Finance had this structure for each of the ETFs in the study\n\nsymbol = \'WTI\'\nstart_da
te = \'2000-01-01\'\nend_date = \'2025-08-19\'\n\nticker = yf.Ticker(symbol)\nhistorical_data = ticker.history(start = start_date, end
= end_date, period = \'1mo\')\nprint(historical_data)\n\nprint("type of historical_data", type(historical_data))\n\nhistorical_data.to_

```

## ✓ Polars DataFrame Development

The following code cell demonstrates Polars use with the time series DataFrame for our selected market/ticker, WTI.

```

wti = pl.read_csv("/content/wti_daily_data.csv", try_parse_dates=True)

# check the original schema
print(wti.schema)

# drop useless columns Dividends and StockSplits
wti = wti.drop(['Dividends', 'StockSplits'])

# create lag price features
wti = wti.with_columns((pl.col('Close')).shift().alias('CloseLag1'))
wti = wti.with_columns((pl.col('CloseLag1')).shift().alias('CloseLag2'))
wti = wti.with_columns((pl.col('CloseLag2')).shift().alias('CloseLag3'))

# create high-minus-low (HML) for day and its lags
wti = wti.with_columns((pl.col('High') - pl.col('Low')).alias('HML'))
wti = wti.with_columns((pl.col('HML')).shift().alias('HMLLag1'))
wti = wti.with_columns((pl.col('HMLLag1')).shift().alias('HMLLag2'))
wti = wti.with_columns((pl.col('HMLLag2')).shift().alias('HMLLag3'))

# create a net change for the day as the open minus closing price OMC
# also create the corresponding lag metrics
wti = wti.with_columns((pl.col('Open') - pl.col('Close')).alias('OMC'))
wti = wti.with_columns((pl.col('OMC')).shift().alias('OMCLag1'))
wti = wti.with_columns((pl.col('OMCLag1')).shift().alias('OMCLag2'))
wti = wti.with_columns((pl.col('OMCLag2')).shift().alias('OMCLag3'))

# create volume lag metrics
wti = wti.with_columns((pl.col('Volume')).shift().alias('VolumeLag1'))
wti = wti.with_columns((pl.col('VolumeLag1')).shift().alias('VolumeLag2'))
wti = wti.with_columns((pl.col('VolumeLag2')).shift().alias('VolumeLag3'))

# compute 10-day exponential moving averages of closing prices
# compute around CloseLag1 to avoid any "leakage" in explanatory variable set
# note also the 10-day buffer between train and test in time-series cross-validation
wti = wti.with_columns((pl.col('CloseLag1').ewm_mean(half_life=1, ignore_nulls=True)).alias('CloseEMA2'))
wti = wti.with_columns((pl.col('CloseLag1').ewm_mean(half_life=2, ignore_nulls=True)).alias('CloseEMA4'))
wti = wti.with_columns((pl.col('CloseLag1').ewm_mean(half_life=4, ignore_nulls=True)).alias('CloseEMA8'))

# log daily returns
wti = wti.with_columns(np.log(pl.col('Close')/pl.col('CloseLag1')).alias('LogReturn'))

# set volume features to Float64 for subsequent use in Numpy arrays
wti = wti.with_columns(
    pl.col('Volume').cast(pl.Float64).round(0),
    pl.col('VolumeLag1').cast(pl.Float64).round(0),
    pl.col('VolumeLag2').cast(pl.Float64).round(0),
    pl.col('VolumeLag3').cast(pl.Float64).round(0),
)

# round other features to three decimal places for reporting and subsequent analytics
wti = wti.with_columns(
    pl.col('Close').cast(pl.Float64).round(3),
    pl.col('Open').cast(pl.Float64).round(3),
    pl.col('High').cast(pl.Float64).round(3),
    pl.col('Low').cast(pl.Float64).round(3),
    pl.col('OMC').cast(pl.Float64).round(3),
    pl.col('OMCLag1').cast(pl.Float64).round(3),
    pl.col('OMCLag2').cast(pl.Float64).round(3),
    pl.col('OMCLag3').cast(pl.Float64).round(3),
    pl.col('HML').cast(pl.Float64).round(3),
    pl.col('HMLLag1').cast(pl.Float64).round(3),
    pl.col('HMLLag2').cast(pl.Float64).round(3),
    pl.col('HMLLag3').cast(pl.Float64).round(3),
    pl.col('VolumeLag1').cast(pl.Float64).round(0),
    pl.col('VolumeLag2').cast(pl.Float64).round(0),
    pl.col('VolumeLag3').cast(pl.Float64).round(0),
)

```

```

pl.col('Open').round(3),
pl.col('High').round(3),
pl.col('Low').round(3),
pl.col('Close').round(3),
pl.col('CloseLag1').round(3),
pl.col('CloseLag2').round(3),
pl.col('CloseLag3').round(3),
pl.col('HML').round(3),
pl.col('HMLLag1').round(3),
pl.col('HMLLag2').round(3),
pl.col('HMLLag3').round(3),
pl.col('OMC').round(3),
pl.col('OMCLag1').round(3),
pl.col('OMCLag2').round(3),
pl.col('OMCLag3').round(3),
pl.col('CloseEMA2').round(3),
pl.col('CloseEMA4').round(3),
pl.col('CloseEMA8').round(3))

# no correction for class imbalance in this analysis
# define binary target/response 1 = market price up since previous day, 0 = even or down
wti = wti.with_columns(pl.when(pl.col('LogReturn')>0).then(pl.lit(1)).otherwise(pl.lit(0)).alias('Target'))

# save to external comma-delimited text file for checking calculations in Excel
wti.write_csv("wti-with-computed-features.csv")

🔗 Schema({'Date': Datetime(time_unit='us', time_zone='UTC'), 'Open': Float64, 'High': Float64, 'Low': Float64, 'Close': Float64, 'Volume':

```

## ▼ Descriptive Statistics for Price Features

```

# Drop the rows with null values such as the initial lag rows
wti = wti.drop_nulls()

# Descriptive statistics
wtiStatistics = wti.drop('Date').describe()

print(wtiStatistics.columns)

wtiStatisticsToPrint = wtiStatistics.transpose(include_header=True).drop(['column_1', 'column_5', 'column_7'])

print(wtiStatisticsToPrint.schema)

with pl.Config(
    tbl_rows = 60,
    tbl_width_chars = 200,
    tbl_cols = -1,
    float_precision = 3,
    tbl_hide_dataframe_shape = True,
    tbl_hide_column_data_types = True):
    print(wtiStatisticsToPrint)

🔗 ['statistic', 'Open', 'High', 'Low', 'Close', 'Volume', 'CloseLag1', 'CloseLag2', 'CloseLag3', 'HML', 'HMLLag1', 'HMLLag2', 'HMLLag3', '
Schema({'column': String, 'column_0': String, 'column_2': String, 'column_3': String, 'column_4': String, 'column_6': String, 'column_8':

```

| column     | column_0 | column_2             | column_3            | column_4 | column_6  | column_8   |
|------------|----------|----------------------|---------------------|----------|-----------|------------|
| statistic  | count    | mean                 | std                 | min      | 50%       | max        |
| Open       | 5169.0   | 9.888208357515962    | 8.24379751138907    | 1.105    | 6.597     | 44.596     |
| High       | 5169.0   | 10.104958405881215   | 8.390030911332957   | 1.145    | 6.784     | 44.626     |
| Low        | 5169.0   | 9.646586767266395    | 8.068196720476314   | 1.028    | 6.4       | 43.235     |
| Close      | 5169.0   | 9.871213774424453    | 8.228563569925155   | 1.047    | 6.592     | 44.172     |
| Volume     | 5169.0   | 1748263.726059199    | 1829939.6046243927  | 34700.0  | 1177400.0 | 40429700.0 |
| CloseLag1  | 5169.0   | 9.873441671503192    | 8.22790431775934    | 1.047    | 6.592     | 44.172     |
| CloseLag2  | 5169.0   | 9.87565660669375     | 8.22725538628012    | 1.047    | 6.592     | 44.172     |
| CloseLag3  | 5169.0   | 9.877919326755658    | 8.226617508094625   | 1.047    | 6.592     | 44.172     |
| HML        | 5169.0   | 0.4583662217063263   | 0.43874669689562873 | 0.029    | 0.327     | 4.129      |
| HMLLag1    | 5169.0   | 0.4583936931708261   | 0.43872807289866955 | 0.029    | 0.327     | 4.129      |
| HMLLag2    | 5169.0   | 0.45853685432385377  | 0.43871551870478587 | 0.029    | 0.327     | 4.129      |
| HMLLag3    | 5169.0   | 0.4587094215515574   | 0.43873022336223927 | 0.029    | 0.327     | 4.129      |
| OMC        | 5169.0   | 0.016985683884697237 | 0.37995566715807194 | -3.223   | 0.01      | 3.705      |
| OMCLag1    | 5169.0   | 0.016974656606693762 | 0.3799555359557333  | -3.223   | 0.01      | 3.705      |
| OMCLag2    | 5169.0   | 0.01702050686786613  | 0.3799653728485099  | -3.223   | 0.01      | 3.705      |
| OMCLag3    | 5169.0   | 0.01715302766492551  | 0.3800928215711145  | -3.223   | 0.01      | 3.705      |
| VolumeLag1 | 5169.0   | 1748148.5974076223   | 1829989.5770774593  | 34700.0  | 1176900.0 | 40429700.0 |
| VolumeLag2 | 5169.0   | 1748359.3538402012   | 1829930.1201336666  | 34700.0  | 1177400.0 | 40429700.0 |


|            |        |                         |                     |                     |           |                    |
|------------|--------|-------------------------|---------------------|---------------------|-----------|--------------------|
| VolumeLag3 | 5169.0 | 1750131.4954536662      | 1833301.7931244925  | 34700.0             | 1178000.0 | 40429700.0         |
| CloseEMA2  | 5169.0 | 9.875673437802282       | 8.219729461746063   | 1.111               | 6.614     | 43.432             |
| CloseEMA4  | 5169.0 | 9.878829560843492       | 8.210186049750966   | 1.136               | 6.624     | 43.018             |
| CloseEMA8  | 5169.0 | 9.885304314180692       | 8.190856957522218   | 1.157               | 6.649     | 42.696             |
| LogReturn  | 5169.0 | -0.00039874914065335176 | 0.04178042201340917 | -0.2644790687687591 | 0.0       | 0.3488180995786815 |
| Target     | 5169.0 | 0.4778487134842329      | 0.4995574042820073  | 0.0                 | 0.0       | 1.0                |

## ▼ Base Feature List Defined on WTI Pricing Alone

Features or explanatory variables, also known as an independent variables, are used to predict the values of target variables. The initial list of features includes the price-based features defined above, everything except the continuous response **LogReturn** if we wanted to employ regression and the binary response **Target** for classification, which is the focus of this project. This complete feature list is used in evaluating all methods.

We retain Date and Target at this point... Date is needed to select across WTI and the many ETFs. Target is needed as the response in training. These will be dropped from the training features set later.

```
# Select Features for the Model, exclude current day price variables ... no "leakage"
# note for moving averages, we have excluded the current day, and provide a 10-day gap
# so these may be included in the set
wti = wti.drop(['LogReturn', 'Open', 'High', 'Low', 'Close', 'Volume', 'HML', 'OMC'])
wti.head()
```

 shape: (5, 17)

|  | Date                    | CloseLag1 | CloseLag2 | CloseLag3 | HMLLag1 | HMLLag2 | HMLLag3 | OMCLag1 | OMCLag2 | OMCLag3 | VolumeLag1 | VolumeLag2 | VolumeLag3 |
|--|-------------------------|-----------|-----------|-----------|---------|---------|---------|---------|---------|---------|------------|------------|------------|
|  | datetime[μs, UTC]       | f64       | f64       | f64       | f64     | f64     | f64     | f64     | f64     | f64     | f64        | f64        | f64        |
|  | 2005-02-02 05:00:00 UTC | 13.196    | 13.189    | 13.406    | 0.232   | 0.79    | 0.942   | -0.007  | 0.217   | 0.725   | 656500.0   | 1.7768e6   | 9.8135e6   |
|  | 2005-02-03 05:00:00 UTC | 13.196    | 13.196    | 13.189    | 0.152   | 0.232   | 0.79    | 0.029   | -0.007  | 0.217   | 265300.0   | 656500.0   | 1.7768e6   |
|  | 2005-02-04 05:00:00 UTC | 13.225    | 13.196    | 13.196    | 0.087   | 0.152   | 0.232   | 0.0     | 0.029   | -0.007  | 303200.0   | 265300.0   | 656500.0   |

## ▼ Read Data for Nine Exchange Traded Funds

The following code cells read data from nine exchange-traded funds (ETFs) selected to cover a wide range of asset classes. All of these funds have about a billion or more assets under management (AUM). We note the range of dates covered in each ETF DataFrame. Later we will ensure that the same date ranges are covered by WTI and the nine ETFs. Beginning dates vary, but all time series end on August 19, 2025.

For each selected ETF, we compute a variable named **Up** that is 1 if the price of the ETF went up the previous day, zero if not.

```
# work with SPY data
spy = pl.read_csv("/content/spy_daily_data.csv", try_parse_dates=True)

# drop useless columns Dividends, StockSplits, and CapitalGains
spy = spy.drop(['Dividends', 'StockSplits', 'CapitalGains'])

spy = spy.with_columns((pl.col('Close')).shift().alias('CloseLag1'))
spy = spy.with_columns((pl.col('CloseLag1')).shift().alias('CloseLag2'))
spy = spy.with_columns(pl.when(pl.col('CloseLag1') > pl.col('CloseLag2')).then(pl.lit(1)).otherwise(pl.lit(0)).alias('SPYUp'))
# check the schema
print(spy.schema)

# Drop initial lag row
spy = spy.drop_nulls()

spyStatistics = spy.describe()
print(spyStatistics.columns)
spyStatisticsToPrint = spyStatistics.transpose(include_header=True).drop(['column_1', 'column_5', 'column_6', 'column_7'])
print(wtiStatisticsToPrint.schema)
with pl.Config(
    tbl_rows = 60,
    tbl_width_chars = 200,
    tbl_cols = -1,
    float_precision = 3,
    tbl_hide_dataframe_shape = True,
    tbl_hide_column_data_types = True):
```

```
print(spyStatisticsToPrint)
```

```
# print a few records at the beginning of the DataFrame
```

```
print(spy.head())
```

```
Schema({'Date': Datetime(time_unit='us', time_zone='UTC'), 'Open': Float64, 'High': Float64, 'Low': Float64, 'Close': Float64, 'Volume':  
['statistic', 'Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'CloseLag1', 'CloseLag2', 'SPYUp']  
Schema({'column': String, 'column_0': String, 'column_2': String, 'column_3': String, 'column_4': String, 'column_6': String, 'column_8':
```

| column    | column_0 | column_2                          | column_3           | column_4                  | column_8                  |
|-----------|----------|-----------------------------------|--------------------|---------------------------|---------------------------|
| statistic | count    | mean                              | std                | min                       | max                       |
| Date      | 6444     | 2012-10-26 16:36:45.027933+00:... | null               | 2000-01-05 05:00:00+00:00 | 2025-08-19 04:00:00+00:00 |
| Open      | 6444.0   | 190.82591842184576                | 144.02936415534128 | 50.113057480959675        | 645.989990234375          |
| High      | 6444.0   | 191.92668889779023                | 144.7563930674051  | 51.62493276253082         | 646.1900024414062         |
| Low       | 6444.0   | 189.6269088728895                 | 143.23284984431902 | 49.48618592302649         | 642.6799926757812         |
| Close     | 6444.0   | 190.84933958343657                | 144.06736661670922 | 50.23106002807617         | 644.9500122070312         |
| Volume    | 6444.0   | 105232386.51458721                | 90179712.63917024  | 1436600.0                 | 871026300.0               |
| CloseLag1 | 6444.0   | 190.76379166087477                | 143.96436701907055 | 50.23106002807617         | 644.9500122070312         |
| CloseLag2 | 6444.0   | 190.67826132712196                | 143.85915506172498 | 50.23106002807617         | 644.9500122070312         |
| SPYUp     | 6444.0   | 0.5446927374301676                | 0.4980371984141425 | 0.0                       | 1.0                       |

```
shape: (5, 9)
```

| Date                       | Open      | High      | Low       | ... | Volume   | CloseLag1 | CloseLag2 | SPYUp |
|----------------------------|-----------|-----------|-----------|-----|----------|-----------|-----------|-------|
| ---                        | ---       | ---       | ---       | --- | ---      | ---       | ---       | ---   |
| datetime[μs,<br>UTC]       | f64       | f64       | f64       | ... | i64      | f64       | f64       | i32   |
| 2000-01-05<br>05:00:00 UTC | 88.657974 | 89.6677   | 86.955297 | ... | 12177900 | 88.539185 | 92.142517 | 0     |
| 2000-01-06<br>05:00:00 UTC | 88.459986 | 89.6479   | 87.272072 | ... | 6227200  | 88.697571 | 88.539185 | 1     |
| 2000-01-07<br>05:00:00 UTC | 88.895594 | 92.340546 | 88.737205 | ... | 8066500  | 87.272072 | 88.697571 | 0     |
| 2000-01-10<br>05:00:00 UTC | 92.657303 | 93.073073 | 91.885159 | ... | 5741700  | 92.340546 | 87.272072 | 1     |
| 2000-01-11<br>05:00:00 UTC | 92.380116 | 92.558303 | 90.915022 | ... | 7503700  | 92.657303 | 92.340546 | 1     |

```
# work with GLD data
```

```
gld = pl.read_csv("/content/gld_daily_data.csv", try_parse_dates=True)
```

```
# drop useless columns Dividends, StockSplits, and CapitalGains
```

```
gld = gld.drop(['Dividends', 'StockSplits', 'CapitalGains'])
```

```
gld = gld.with_columns((pl.col('Close')).shift().alias('CloseLag1'))
```

```
gld = gld.with_columns((pl.col('CloseLag1')).shift().alias('CloseLag2'))
```

```
gld = gld.with_columns(pl.when(pl.col('CloseLag1')>pl.col('CloseLag2')).then(pl.lit(1)).otherwise(pl.lit(0)).alias('GLDUp'))
```

```
# check the schema
```

```
print(gld.schema)
```

```
# Drop initial lag row
```

```
gld = gld.drop_nulls()
```

```
gldStatistics = gld.describe()
```

```
print(gldStatistics.columns)
```

```
gldStatisticsToPrint = gldStatistics.transpose(include_header=True).drop(['column_1', 'column_5', 'column_6', 'column_7'])
```

```
print(wtiStatisticsToPrint.schema)
```

```
with pl.Config(
```

```
    tbl_rows = 60,
```

```
    tbl_width_chars = 200,
```

```
    tbl_cols = -1,
```

```
    float_precision = 3,
```

```
    tbl_hide_dataframe_shape = True,
```

```
    tbl_hide_column_data_types = True):
```

```
    print(gldStatisticsToPrint)
```

```
# print a few records at the beginning of the DataFrame
```

```
print(gld.head())
```

```
Schema({'Date': Datetime(time_unit='us', time_zone='UTC'), 'Open': Float64, 'High': Float64, 'Low': Float64, 'Close': Float64, 'Volume':  
['statistic', 'Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'CloseLag1', 'CloseLag2', 'GLDUp']  
Schema({'column': String, 'column_0': String, 'column_2': String, 'column_3': String, 'column_4': String, 'column_6': String, 'column_8':
```

| column    | column_0 | column_2 | column_3 | column_4 | column_8 |
|-----------|----------|----------|----------|----------|----------|
| statistic | count    | mean     | std      | min      | max      |

|           |        |                                   |                     |                           |                           |
|-----------|--------|-----------------------------------|---------------------|---------------------------|---------------------------|
| Date      | 5218   | 2015-04-04 10:38:49.858183+00:... | null                | 2004-11-22 05:00:00+00:00 | 2025-08-19 04:00:00+00:00 |
| Open      | 5218.0 | 132.1412112242237                 | 51.52670299042503   | 41.029998779296875        | 317.489990234375          |
| High      | 5218.0 | 132.77317369738134                | 51.70272920655233   | 41.36000061035156         | 317.6300048828125         |
| Low       | 5218.0 | 131.45804141663464                | 51.31802554569786   | 41.02000045776367         | 315.0400085449219         |
| Close     | 5218.0 | 132.14241081638417                | 51.533912524483476  | 41.2599983215332          | 316.2900085449219         |
| Volume    | 5218.0 | 9420078.114220008                 | 6644238.186068209   | 319300.0                  | 93804200.0                |
| CloseLag1 | 5218.0 | 132.09248939242204                | 51.49232280202278   | 41.2599983215332          | 316.2900085449219         |
| CloseLag2 | 5218.0 | 132.0421693442927                 | 51.44969185599339   | 41.2599983215332          | 316.2900085449219         |
| GLDUp     | 5218.0 | 0.5298965120735915                | 0.49915323048311416 | 0.0                       | 1.0                       |

shape: (5, 9)

| Date                        | Open      | High      | Low       | ... | Volume   | CloseLag1 | CloseLag2 | GLDUp |
|-----------------------------|-----------|-----------|-----------|-----|----------|-----------|-----------|-------|
| ---<br>datetime[μs,<br>UTC] | f64       | f64       | f64       |     | i64      | f64       | f64       | i32   |
| 2004-11-22 05:00:00 UTC     | 44.75     | 44.970001 | 44.740002 | ... | 11996000 | 44.779999 | 44.380001 | 1     |
| 2004-11-23 05:00:00 UTC     | 44.880001 | 44.919998 | 44.720001 | ... | 3169200  | 44.950001 | 44.779999 | 1     |
| 2004-11-24 05:00:00 UTC     | 44.93     | 45.049999 | 44.790001 | ... | 6105100  | 44.75     | 44.950001 | 0     |
| 2004-11-26 05:00:00 UTC     | 45.25     | 45.599998 | 45.060001 | ... | 3097700  | 45.049999 | 44.75     | 1     |
| 2004-11-29 05:00:00 UTC     | 45.099998 | 45.5      | 45.080002 | ... | 3759000  | 45.290001 | 45.049999 | 1     |

```
# work with VGT data
vgt = pl.read_csv("/content/vgt_daily_data.csv", try_parse_dates=True)

# drop useless columns Dividends, StockSplits, and CapitalGains
vgt = vgt.drop(['Dividends', 'StockSplits', 'CapitalGains'])

vgt = vgt.with_columns((pl.col('Close')).shift().alias('CloseLag1'))
vgt = vgt.with_columns((pl.col('CloseLag1')).shift().alias('CloseLag2'))
vgt = vgt.with_columns(pl.when(pl.col('CloseLag1')>pl.col('CloseLag2')).then(pl.lit(1)).otherwise(pl.lit(0)).alias('VGTUp'))
# check the schema
print(vgt.schema)

# Drop initial lag row
vgt = vgt.drop_nulls()

vgtStatistics = vgt.describe()
print(vgtStatistics.columns)
vgtStatisticsToPrint = vgtStatistics.transpose(include_header=True).drop(['column_1', 'column_5', 'column_6', 'column_7'])
print(vgtStatisticsToPrint.schema)
with pl.Config(
    tbl_rows = 60,
    tbl_width_chars = 200,
    tbl_cols = -1,
    float_precision = 3,
    tbl_hide_dataframe_shape = True,
    tbl_hide_column_data_types = True):
    print(vgtStatisticsToPrint)

# print a few records at the beginning of the DataFrame
print(vgt.head())
```

➡ Schema({'Date': Datetime(time\_unit='us', time\_zone='UTC'), 'Open': Float64, 'High': Float64, 'Low': Float64, 'Close': Float64, 'Volume': ['statistic', 'Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'CloseLag1', 'CloseLag2', 'VGTUp']  
Schema({'column': String, 'column\_0': String, 'column\_2': String, 'column\_3': String, 'column\_4': String, 'column\_6': String, 'column\_8'

| column    | column_0 | column_2                          | column_3           | column_4                  | column_8                  |
|-----------|----------|-----------------------------------|--------------------|---------------------------|---------------------------|
| statistic | count    | mean                              | std                | min                       | max                       |
| Date      | 5421     | 2014-11-08 04:40:16.602103+00:... | null               | 2004-02-03 05:00:00+00:00 | 2025-08-19 04:00:00+00:00 |
| Open      | 5421.0   | 167.88651112658872                | 165.87005650920398 | 25.400250326119036        | 710.0499877929688         |
| High      | 5421.0   | 169.15025730560342                | 167.19942905975995 | 26.273459632207647        | 710.8800048828125         |
| Low       | 5421.0   | 166.46979142040922                | 164.35842233160983 | 24.68993342530967         | 704.0                     |
| Close     | 5421.0   | 167.9006570834688                 | 165.88368753316217 | 25.04936981201172         | 707.2899780273438         |
| Volume    | 5421.0   | 360279.91145545105                | 400808.0053902826  | 0.0                       | 6564500.0                 |
| CloseLag1 | 5421.0   | 167.78119283618796                | 165.74161113685028 | 25.04936981201172         | 707.2899780273438         |
| CloseLag2 | 5421.0   | 167.65927450781356                | 165.59152803560164 | 25.04936981201172         | 707.2899780273438         |
| VGTUp     | 5421.0   | 0.5543257701531082                | 0.4970857990731627 | 0.0                       | 1.0                       |

shape: (5, 9)

| Date | Open | High | Low | ... | Volume | CloseLag1 | CloseLag2 | VGTUp |
|------|------|------|-----|-----|--------|-----------|-----------|-------|
|------|------|------|-----|-----|--------|-----------|-----------|-------|

| ---<br>datetime[μs,<br>UTC] | ---<br>f64 | ---<br>f64 | ---<br>f64 | ... | ---<br>i64 | ---<br>f64 | ---<br>f64 | ---<br>i32 |
|-----------------------------|------------|------------|------------|-----|------------|------------|------------|------------|
| 2004-02-03<br>05:00:00 UTC  | 40.925945  | 40.942719  | 40.774989  | ... | 231100     | 41.194313  | 41.160759  | 1          |
| 2004-02-04<br>05:00:00 UTC  | 39.83571   | 39.83571   | 39.709911  | ... | 51000      | 40.942719  | 41.194313  | 0          |
| 2004-02-05<br>05:00:00 UTC  | 40.003442  | 40.003442  | 39.709913  | ... | 2600       | 39.709911  | 40.942719  | 0          |
| 2004-02-06<br>05:00:00 UTC  | 40.17116   | 40.716278  | 40.17116   | ... | 1000       | 39.91119   | 39.709911  | 1          |
| 2004-02-09<br>05:00:00 UTC  | 40.791757  | 40.917555  | 40.607254  | ... | 3200       | 40.716278  | 39.91119   | 1          |

```
# work with VB data
vb = pl.read_csv("/content/vb_daily_data.csv", try_parse_dates=True)

# drop useless columns Dividends, StockSplits, and CapitalGains
vb = vb.drop(['Dividends', 'StockSplits', 'CapitalGains'])

vb = vb.with_columns((pl.col('Close')).shift().alias('CloseLag1'))
vb = vb.with_columns((pl.col('CloseLag1')).shift().alias('CloseLag2'))
vb = vb.with_columns(pl.when(pl.col('CloseLag1')>pl.col('CloseLag2')).then(pl.lit(1)).otherwise(pl.lit(0)).alias('VBUp'))
# check the schema
print(vb.schema)

# Drop initial lag row
vb = vb.drop_nulls()

vbStatistics = vb.describe()
print(vbStatistics.columns)
vbStatisticsToPrint = vbStatistics.transpose(include_header=True).drop(['column_1', 'column_5', 'column_6', 'column_7'])
print(vbStatisticsToPrint.schema)
with pl.Config(
    tbl_rows = 60,
    tbl_width_chars = 200,
    tbl_cols = -1,
    float_precision = 3,
    tbl_hide_dataframe_shape = True,
    tbl_hide_column_data_types = True):
    print(vbStatisticsToPrint)

# print a few records at the beginning of the DataFrame
print(vb.head())

🔗 Schema({'Date': Datetime(time_unit='us', time_zone='UTC'), 'Open': Float64, 'High': Float64, 'Low': Float64, 'Close': Float64, 'Volume':
['statistic', 'Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'CloseLag1', 'CloseLag2', 'VBUp']
Schema({'column': String, 'column_0': String, 'column_2': String, 'column_3': String, 'column_4': String, 'column_6': String, 'column_8'
```

| column    | column_0 | column_2                          | column_3            | column_4                  | column_8                  |
|-----------|----------|-----------------------------------|---------------------|---------------------------|---------------------------|
| statistic | count    | mean                              | std                 | min                       | max                       |
| Date      | 5421     | 2014-11-08 04:40:16.602103+00:... | null                | 2004-02-03 05:00:00+00:00 | 2025-08-19 04:00:00+00:00 |
| Open      | 5421.0   | 106.9052813338388                 | 62.1452158206943    | 23.633474085495507        | 258.1815273395355         |
| High      | 5421.0   | 107.60790542830034                | 62.54629697613005   | 24.28031945289275         | 260.4463040442688         |
| Low       | 5421.0   | 106.04389329572601                | 61.673058648440175  | 19.9520221629505          | 257.5090323220884         |
| Close     | 5421.0   | 106.86636685982025                | 62.11471701676378   | 23.546703338623047        | 258.69580078125           |
| Volume    | 5421.0   | 491339.45766463754                | 522990.5325407978   | 200.0                     | 9288100.0                 |
| CloseLag1 | 5421.0   | 106.82761647115882                | 62.09329247825284   | 23.546703338623047        | 258.69580078125           |
| CloseLag2 | 5421.0   | 106.78877441576543                | 62.07172201176196   | 23.546703338623047        | 258.69580078125           |
| VBUp      | 5421.0   | 0.5366168603578676                | 0.49870340245454003 | 0.0                       | 1.0                       |

shape: (5, 9)

| Date                        | Open       | High       | Low        | ... | Volume     | CloseLag1  | CloseLag2  | VBUp       |
|-----------------------------|------------|------------|------------|-----|------------|------------|------------|------------|
| ---<br>datetime[μs,<br>UTC] | ---<br>f64 | ---<br>f64 | ---<br>f64 | --- | ---<br>i64 | ---<br>f64 | ---<br>f64 | ---<br>i32 |
| 2004-02-03<br>05:00:00 UTC  | 36.107417  | 36.18088   | 36.048645  | ... | 1200       | 36.14415   | 35.997215  | 1          |
| 2004-02-04<br>05:00:00 UTC  | 35.872324  | 35.872324  | 35.299305  | ... | 1600       | 36.048645  | 36.14415   | 0          |
| 2004-02-05<br>05:00:00 UTC  | 35.549099  | 35.622565  | 35.336054  | ... | 4000       | 35.299305  | 36.048645  | 0          |
| 2004-02-06<br>05:00:00 UTC  | 35.615208  | 36.254341  | 35.615208  | ... | 2000       | 35.409519  | 35.299305  | 1          |
| 2004-02-09                  | 36.511441  | 36.636328  | 36.504093  | ... | 2900       | 36.254341  | 35.409519  | 1          |



[illegible]

```
# work with IVE data
ive = pl.read_csv("/content/ive_daily_data.csv", try_parse_dates=True)

# drop useless columns Dividends, StockSplits, and CapitalGains
ive = ive.drop(['Dividends', 'StockSplits', 'CapitalGains'])

ive = ive.with_columns((pl.col('Close')).shift().alias('CloseLag1'))
ive = ive.with_columns((pl.col('CloseLag1')).shift().alias('CloseLag2'))
ive = ive.with_columns(pl.when(pl.col('CloseLag1') > pl.col('CloseLag2')).then(pl.lit(1)).otherwise(pl.lit(0)).alias('IVEUp'))
# check the schema
print(ive.schema)

# Drop initial lag row
ive = ive.drop_nulls()

iveStatistics = ive.describe()
print(iveStatistics.columns)
iveStatisticsToPrint = iveStatistics.transpose(include_header=True).drop(['column_1', 'column_5', 'column_6', 'column_7'])
print(wtiStatisticsToPrint.schema)
with pl.Config(
    tbl_rows = 60,
    tbl_width_chars = 200,
    tbl_cols = -1,
    float_precision = 3,
    tbl_hide_dataframe_shape = True,
    tbl_hide_column_data_types = True):
    print(iveStatisticsToPrint)

# print a few records at the beginning of the DataFrame
print(ive.head())
```

```
Schema({
  'Date': Datetime(time_unit='us', time_zone='UTC'),
  'Open': Float64,
  'High': Float64,
  'Low': Float64,
  'Close': Float64,
  'Volume':
    ['statistic', 'Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'CloseLag1', 'CloseLag2', 'IVEUp']
  Schema({
    'column_0': String,
    'column_2': String,
    'column_3': String,
    'column_4': String,
    'column_6': String,
    'column_8':
```

| column    | column_0 | column_2                          | column_3            | column_4                  | column_8                  |
|-----------|----------|-----------------------------------|---------------------|---------------------------|---------------------------|
| statistic | count    | mean                              | std                 | min                       | max                       |
| Date      | 6343     | 2013-01-08 00:46:19.883336+00:... | null                | 2000-05-31 04:00:00+00:00 | 2025-08-19 04:00:00+00:00 |
| Open      | 6343.0   | 74.6655833423332                  | 46.11985052593699   | 21.26051340242963         | 203.21139722740784        |
| High      | 6343.0   | 75.06144188464853                 | 46.32247720139887   | 22.014917670226524        | 203.8427634546091         |
| Low       | 6343.0   | 74.22128225486621                 | 45.922028399013676  | 20.797465831147417        | 202.95489456330537        |
| Close     | 6343.0   | 74.66454975014508                 | 46.129873687318245  | 21.3702449798584          | 203.3889617919922         |
| Volume    | 6343.0   | 677624.5940406747                 | 792860.168455924    | 400.0                     | 20403600.0                |
| CloseLag1 | 6343.0   | 74.63817534400859                 | 46.10495666736668   | 21.3702449798584          | 203.3889617919922         |
| CloseLag2 | 6343.0   | 74.61178671650268                 | 46.080269741101944  | 21.3702449798584          | 203.3889617919922         |
| IVEUp     | 6343.0   | 0.5344474223553524                | 0.49885128818986785 | 0.0                       | 1.0                       |

```
shape: (5, 9)
```

| Date<br>---<br>datetime[μs,<br>UTC] | Open<br>---<br>f64 | High<br>---<br>f64 | Low<br>---<br>f64 | ... | Volume<br>---<br>i64 | CloseLag1<br>---<br>f64 | CloseLag2<br>---<br>f64 | IVEUp<br>---<br>i32 |
|-------------------------------------|--------------------|--------------------|-------------------|-----|----------------------|-------------------------|-------------------------|---------------------|
| 2000-05-31<br>04:00:00 UTC          | 34.549207          | 34.899005          | 34.549207         | ... | 12200                | 34.567142               | 34.046928               | 1                   |
| 2000-06-01<br>04:00:00 UTC          | 34.890046          | 35.257782          | 34.836231         | ... | 25400                | 34.827251               | 34.567142               | 1                   |
| 2000-06-02<br>04:00:00 UTC          | 35.858717          | 35.93047           | 35.751087         | ... | 10000                | 35.257782               | 34.827251               | 1                   |
| 2000-06-05<br>04:00:00 UTC          | 35.786959          | 35.786959          | 35.544792         | ... | 15500                | 35.751087               | 35.257782               | 1                   |
| 2000-06-06<br>04:00:00 UTC          | 35.508916          | 35.544792          | 35.508916         | ... | 4100                 | 35.589638               | 35.751087               | 0                   |

```
# work with XLI data
xli = pl.read_csv("/content/xli_daily_data.csv", try_parse_dates=True)

# drop useless columns Dividends, StockSplits, and CapitalGains
xli = xli.drop(['Dividends', 'StockSplits', 'CapitalGains'])

xli = xli.with_columns((pl.col('Close')).shift().alias('CloseLag1'))
xli = xli.with_columns((pl.col('CloseLag1')).shift().alias('CloseLag2'))
```

```
xli = xli.with_columns(pl.when(pl.col('CloseLag1')>pl.col('CloseLag2')).then(pl.lit(1)).otherwise(pl.lit(0)).alias('XLIUp'))
# check the schema
print(xli.schema)

# Drop initial lag row
xli = xli.drop_nulls()
```

```
xliStatistics = xli.describe()
print(xliStatistics.columns)
xliStatisticsToPrint = xliStatistics.transpose(include_header=True).drop(['column_1', 'column_5', 'column_6', 'column_7'])
print(xliStatisticsToPrint.schema)
with pl.Config(
    tbl_rows = 60,
    tbl_width_chars = 200,
    tbl_cols = -1,
    float_precision = 3,
    tbl_hide_dataframe_shape = True,
    tbl_hide_column_data_types = True):
    print(xliStatisticsToPrint)
```

```
# print a few records at the beginning of the DataFrame
print(xli.head())
```

```
Schema({'Date': Datetime(time_unit='us', time_zone='UTC'), 'Open': Float64, 'High': Float64, 'Low': Float64, 'Close': Float64, 'Volume':
['statistic', 'Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'CloseLag1', 'CloseLag2', 'XLIUp']
Schema({'column': String, 'column_0': String, 'column_2': String, 'column_3': String, 'column_4': String, 'column_6': String, 'column_8'
```

| column    | column_0 | column_2                          | column_3           | column_4                  | column_8                  |
|-----------|----------|-----------------------------------|--------------------|---------------------------|---------------------------|
| statistic | count    | mean                              | std                | min                       | max                       |
| Date      | 6444     | 2012-10-26 16:36:45.027933+00:... | null               | 2000-01-05 05:00:00+00:00 | 2025-08-19 04:00:00+00:00 |
| Open      | 6444.0   | 47.13428678572114                 | 33.75933048208735  | 11.236274653018539        | 155.07000732421875        |
| High      | 6444.0   | 47.445254711260986                | 33.95744459372276  | 11.497583816634966        | 155.14999389648438        |
| Low       | 6444.0   | 46.80200222742193                 | 33.55937913538951  | 10.989484954974614        | 154.07000732421875        |
| Close     | 6444.0   | 47.13715054336207                 | 33.76749713408024  | 11.149171829223633        | 154.99000549316406        |
| Volume    | 6444.0   | 8557567.116697703                 | 7392044.013283453  | 400.0                     | 79118200.0                |
| CloseLag1 | 6444.0   | 47.1164178205972                  | 33.74450283125619  | 11.149171829223633        | 154.99000549316406        |
| CloseLag2 | 6444.0   | 47.09581460485097                 | 33.72157067152891  | 11.149171829223633        | 154.99000549316406        |
| XLIUp     | 6444.0   | 0.5332091868404718                | 0.4989346455594378 | 0.0                       | 1.0                       |

shape: (5, 9)

| Date                    | Open      | High      | Low       | ... | Volume | CloseLag1 | CloseLag2 | XLIUp |
|-------------------------|-----------|-----------|-----------|-----|--------|-----------|-----------|-------|
| ---                     | ---       | ---       | ---       | --- | ---    | ---       | ---       | ---   |
| datetime[μs, UTC]       | f64       | f64       | f64       | ... | i64    | f64       | f64       | i32   |
| 2000-01-05 05:00:00 UTC | 17.728652 | 17.807798 | 17.550574 | ... | 129200 | 17.758335 | 18.262877 | 0     |
| 2000-01-06 05:00:00 UTC | 17.679196 | 17.9661   | 17.609944 | ... | 54000  | 17.679186 | 17.758335 | 0     |
| 2000-01-07 05:00:00 UTC | 18.124382 | 18.599257 | 18.124382 | ... | 32900  | 17.916634 | 17.679186 | 1     |
| 2000-01-10 05:00:00 UTC | 18.698189 | 18.757548 | 18.539897 | ... | 122500 | 18.599257 | 17.916634 | 1     |
| 2000-01-11 05:00:00 UTC | 18.658613 | 18.658613 | 18.361816 | ... | 109800 | 18.599257 | 18.599257 | 0     |

```
# work with XLU data
xlu = pl.read_csv("/content/xlu_daily_data.csv", try_parse_dates=True)

# drop useless columns Dividends, StockSplits, and CapitalGains
xlu = xlu.drop(['Dividends', 'StockSplits', 'CapitalGains'])

xlu = xlu.with_columns((pl.col('Close')).shift().alias('CloseLag1'))
xlu = xlu.with_columns((pl.col('CloseLag1')).shift().alias('CloseLag2'))
xlu = xlu.with_columns(pl.when(pl.col('CloseLag1')>pl.col('CloseLag2')).then(pl.lit(1)).otherwise(pl.lit(0)).alias('XLUUp'))
# check the schema
print(xlu.schema)

# Drop initial lag row
xlu = xlu.drop_nulls()

xluStatistics = xlu.describe()
print(xluStatistics.columns)
xluStatisticsToPrint = xluStatistics.transpose(include_header=True).drop(['column_1', 'column_5', 'column_6', 'column_7'])
print(xluStatisticsToPrint.schema)
with pl.Config(
```

```
tbl_rows = 60,
tbl_width_chars = 200,
tbl_cols = -1,
float_precision = 3,
tbl_hide_dataframe_shape = True,
tbl_hide_column_data_types = True):
print(xluStatisticsToPrint)
```

```
# print a few records at the beginning of the DataFrame
print(xlu.head())
```

```
Schema({'Date': Datetime(time_unit='us', time_zone='UTC'), 'Open': Float64, 'High': Float64, 'Low': Float64, 'Close': Float64, 'Volume':
['statistic', 'Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'CloseLag1', 'CloseLag2', 'XLUUp']
Schema({'column': String, 'column_0': String, 'column_2': String, 'column_3': String, 'column_4': String, 'column_6': String, 'column_8'
```

| column    | column_0 | column_2                          | column_3            | column_4                  | column_8                  |
|-----------|----------|-----------------------------------|---------------------|---------------------------|---------------------------|
| statistic | count    | mean                              | std                 | min                       | max                       |
| Date      | 6444     | 2012-10-26 16:36:45.027933+00:... | null                | 2000-01-05 05:00:00+00:00 | 2025-08-19 04:00:00+00:00 |
| Open      | 6444.0   | 31.894989147350113                | 19.721009673760992  | 6.889745684259615         | 87.3499984741211          |
| High      | 6444.0   | 32.123881544096875                | 19.861535759212497  | 7.41416278528114          | 87.66999816894531         |
| Low       | 6444.0   | 31.650661453103062                | 19.573122239135174  | 6.736037375918337         | 86.12999725341797         |
| Close     | 6444.0   | 31.89608001967974                 | 19.7271434580805    | 6.885226726531982         | 87.31999969482422         |
| Volume    | 6444.0   | 8837735.055865921                 | 7372839.575100792   | 3200.0                    | 90263100.0                |
| CloseLag1 | 6444.0   | 31.88441676186302                 | 19.717318739292757  | 6.885226726531982         | 87.31999969482422         |
| CloseLag2 | 6444.0   | 31.8729350237577                  | 19.707778243189146  | 6.885226726531982         | 87.31999969482422         |
| XLUUp     | 6444.0   | 0.5335195530726257                | 0.49891388733682634 | 0.0                       | 1.0                       |

```
shape: (5, 9)
```

| Date                    | Open      | High      | Low       | ... | Volume | CloseLag1 | CloseLag2 | XLUUp |
|-------------------------|-----------|-----------|-----------|-----|--------|-----------|-----------|-------|
| ---                     | ---       | ---       | ---       | --- | ---    | ---       | ---       | ---   |
| datetime[μs, UTC]       | f64       | f64       | f64       | ... | i64    | f64       | f64       | i32   |
| 2000-01-05 05:00:00 UTC | 11.165537 | 11.248862 | 11.018116 | ... | 273400 | 10.921968 | 11.26168  | 0     |
| 2000-01-06 05:00:00 UTC | 11.178358 | 11.178358 | 11.050166 | ... | 34500  | 11.197585 | 10.921968 | 1     |
| 2000-01-07 05:00:00 UTC | 11.197586 | 11.274501 | 11.146309 | ... | 46200  | 11.178358 | 11.197585 | 0     |
| 2000-01-10 05:00:00 UTC | 11.268088 | 11.345003 | 11.248859 | ... | 15500  | 11.274501 | 11.178358 | 1     |
| 2000-01-11 05:00:00 UTC | 11.261679 | 11.261679 | 11.107849 | ... | 25700  | 11.312955 | 11.274501 | 1     |

```
# work with SLV data
slv = pl.read_csv("/content/slv_daily_data.csv", try_parse_dates=True)
```

```
# drop useless columns Dividends, StockSplits, and CapitalGains
slv = slv.drop(['Dividends', 'StockSplits', 'CapitalGains'])
```

```
slv = slv.with_columns((pl.col('Close')).shift().alias('CloseLag1'))
slv = slv.with_columns((pl.col('CloseLag1')).shift().alias('CloseLag2'))
slv = slv.with_columns(pl.when(pl.col('CloseLag1')>pl.col('CloseLag2')).then(pl.lit(1)).otherwise(pl.lit(0)).alias('SLVUp'))
# check the schema
print(slv.schema)
```

```
# Drop initial lag row
slv = slv.drop_nulls()
```

```
slvStatistics = slv.describe()
print(slvStatistics.columns)
slvStatisticsToPrint = slvStatistics.transpose(include_header=True).drop(['column_1', 'column_5', 'column_6', 'column_7'])
print(wtiStatisticsToPrint.schema)
with pl.Config(
    tbl_rows = 60,
    tbl_width_chars = 200,
    tbl_cols = -1,
    float_precision = 3,
    tbl_hide_dataframe_shape = True,
    tbl_hide_column_data_types = True):
    print(slvStatisticsToPrint)
```

```
# print a few records at the beginning of the DataFrame
print(slv.head())
```

```

Schema({'Date': Datetime(time_unit='us', time_zone='UTC'), 'Open': Float64, 'High': Float64, 'Low': Float64, 'Close': Float64, 'Volume':
['statistic', 'Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'CloseLag1', 'CloseLag2', 'SLVUp']
Schema({'column': String, 'column_0': String, 'column_2': String, 'column_3': String, 'column_4': String, 'column_6': String, 'column_8'

```

| column    | column_0 | column_2                          | column_3           | column_4                  | column_8                  |
|-----------|----------|-----------------------------------|--------------------|---------------------------|---------------------------|
| statistic | count    | mean                              | std                | min                       | max                       |
| Date      | 4856     | 2015-12-23 02:12:58.418451+00:... | null               | 2006-05-02 04:00:00+00:00 | 2025-08-19 04:00:00+00:00 |
| Open      | 4856.0   | 19.84649361829585                 | 6.487981729697927  | 8.710000038146973         | 47.619998931884766        |
| High      | 4856.0   | 20.0214888811897                  | 6.55865103290389   | 9.050000190734863         | 48.349998474121094        |
| Low       | 4856.0   | 19.649800242859214                | 6.395723154542012  | 8.449999809265137         | 46.54999923706055         |
| Close     | 4856.0   | 19.840868817129873                | 6.487317455039558  | 8.850000381469727         | 47.2599983215332          |
| Volume    | 4856.0   | 15420806.878088962                | 16683460.219457056 | 1039000.0                 | 295400000.0               |
| CloseLag1 | 4856.0   | 19.83675020118129                 | 6.484757133225693  | 8.850000381469727         | 47.2599983215332          |
| CloseLag2 | 4856.0   | 19.832479608314237                | 6.481893562087745  | 8.850000381469727         | 47.2599983215332          |
| SLVUp     | 4856.0   | 0.5168863261943987                | 0.4997662319131529 | 0.0                       | 1.0                       |

shape: (5, 9)

| Date                    | Open   | High   | Low    | ... | Volume   | CloseLag1 | CloseLag2 | SLVUp |
|-------------------------|--------|--------|--------|-----|----------|-----------|-----------|-------|
| ---                     | ---    | ---    | ---    | --- | ---      | ---       | ---       | ---   |
| datetime[μs, UTC]       | f64    | f64    | f64    | ... | i64      | f64       | f64       | i32   |
| 2006-05-02 04:00:00 UTC | 14.245 | 14.4   | 14.1   | ... | 12511000 | 13.87     | 13.812    | 1     |
| 2006-05-03 04:00:00 UTC | 14.45  | 14.464 | 13.413 | ... | 15141000 | 14.365    | 13.87     | 1     |
| 2006-05-04 04:00:00 UTC | 13.95  | 14.287 | 13.68  | ... | 11075000 | 13.925    | 14.365    | 0     |
| 2006-05-05 04:00:00 UTC | 14.0   | 14.03  | 13.75  | ... | 6586000  | 14.0      | 13.925    | 1     |
| 2006-05-08 04:00:00 UTC | 13.8   | 14.0   | 13.506 | ... | 9453000  | 13.995    | 14.0      | 0     |

```

# work with USO data
uso = pl.read_csv("/content/uso_daily_data.csv", try_parse_dates=True)

# drop useless columns Dividends, StockSplits, and CapitalGains
uso = uso.drop(['Dividends', 'StockSplits', 'CapitalGains'])

uso = uso.with_columns((pl.col('Close')).shift().alias('CloseLag1'))
uso = uso.with_columns((pl.col('CloseLag1')).shift().alias('CloseLag2'))
uso = uso.with_columns(pl.when(pl.col('CloseLag1')>pl.col('CloseLag2')).then(pl.lit(1)).otherwise(pl.lit(0)).alias('USOUp'))
# check the schema
print(uso.schema)

# Drop initial lag row
uso = uso.drop_nulls()

usoStatistics = uso.describe()
print(usoStatistics.columns)
usoStatisticsToPrint = usoStatistics.transpose(include_header=True).drop(['column_1', 'column_5', 'column_6', 'column_7'])
print(wtiStatisticsToPrint.schema)
with pl.Config(
    tbl_rows = 60,
    tbl_width_chars = 200,
    tbl_cols = -1,
    float_precision = 3,
    tbl_hide_dataframe_shape = True,
    tbl_hide_column_data_types = True):
    print(usoStatisticsToPrint)

# print a few records at the beginning of the DataFrame
print(uso.head())

```

```

Schema({'Date': Datetime(time_unit='us', time_zone='UTC'), 'Open': Float64, 'High': Float64, 'Low': Float64, 'Close': Float64, 'Volume':
['statistic', 'Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'CloseLag1', 'CloseLag2', 'USOUp']
Schema({'column': String, 'column_0': String, 'column_2': String, 'column_3': String, 'column_4': String, 'column_6': String, 'column_8'

```

| column    | column_0 | column_2                          | column_3           | column_4                  | column_8                  |
|-----------|----------|-----------------------------------|--------------------|---------------------------|---------------------------|
| statistic | count    | mean                              | std                | min                       | max                       |
| Date      | 4869     | 2015-12-13 15:52:11.238447+00:... | null               | 2006-04-12 04:00:00+00:00 | 2025-08-19 04:00:00+00:00 |
| Open      | 4869.0   | 207.56218537914114                | 172.12513325570848 | 17.280000686645508        | 952.6400146484375         |
| High      | 4869.0   | 209.87204556632614                | 174.16241714493188 | 18.0                      | 953.3599853515625         |
| Low       | 4869.0   | 205.02635254098982                | 169.7785603051359  | 16.8799991607666          | 932.0                     |
| Close     | 4869.0   | 207.54419582231404                | 172.08918039384076 | 17.040000915527344        | 939.8400268554688         |
| Volume    | 4869.0   | 3012812.4988704044                | 4036484.7365261046 | 14888.0                   | 124913013.0               |

|           |        |                    |                     |                    |                   |
|-----------|--------|--------------------|---------------------|--------------------|-------------------|
| CloseLag1 | 4869.0 | 207.64140263532167 | 172.14643555816238  | 17.040000915527344 | 939.8400268554688 |
| CloseLag2 | 4869.0 | 207.73813501876296 | 172.2031777305166   | 17.040000915527344 | 939.8400268554688 |
| USOUUp    | 4869.0 | 0.5103717395769152 | 0.49994375754988574 | 0.0                | 1.0               |

shape: (5, 9)

| Date                     | Open       | High       | Low        | ... | Volume | CloseLag1  | CloseLag2  | USOUp |
|--------------------------|------------|------------|------------|-----|--------|------------|------------|-------|
| ---<br>datetime[μs, UTC] | f64        | f64        | f64        |     | i64    | f64        | f64        | i32   |
| 2006-04-12 04:00:00 UTC  | 545.76001  | 550.47998  | 542.47998  | ... | 156038 | 545.599976 | 544.159973 | 1     |
| 2006-04-13 04:00:00 UTC  | 540.0      | 551.919983 | 539.200012 | ... | 70088  | 542.719971 | 545.599976 | 0     |
| 2006-04-17 04:00:00 UTC  | 553.599976 | 559.200012 | 549.440002 | ... | 114713 | 550.559998 | 542.719971 | 1     |
| 2006-04-18 04:00:00 UTC  | 560.799988 | 568.400024 | 556.559998 | ... | 115338 | 558.320007 | 550.559998 | 1     |
| 2006-04-19 04:00:00 UTC  | 564.640015 | 577.280029 | 563.919983 | ... | 98725  | 566.0      | 558.320007 | 1     |

```
# Find earliest date that can be used across all DataFrames
minDateAll = max(min(wti['Date']),min(spy['Date']),min(gld['Date']),
                 min(vgt['Date']),min(vb['Date']),min(ive['Date']),
                 min(xli['Date']),min(xlu['Date']),min(slv['Date']),min(uso['Date']))
```

```
print("minimum Date on which to select, minDateAll:", minDateAll)
```

```
➡ minimum Date on which to select, minDateAll: 2006-05-02 04:00:00+00:00
```

```
# find latest date across all DataFrames
```

```
maxDateAll = min(max(wti['Date']),max(spy['Date']),max(gld['Date']),
                 max(vgt['Date']),max(vb['Date']),max(ive['Date']),
                 max(xli['Date']),max(xlu['Date']),max(slv['Date']),max(uso['Date']))
```

```
print("maximum Date on which to select, axDateAll:", maxDateAll)
```

```
➡ maximum Date on which to select, axDateAll: 2025-08-19 04:00:00+00:00
```

```
# select training data from minDateAll to 2024-12-31, test data all dates in 2025
beginTrain = minDateAll
endTrain = datetime(2024, 12, 31, tzinfo=timezone.utc)
beginTest = datetime(2025, 1, 1, tzinfo=timezone.utc)
endTest = maxDateAll
wtiTrain = wti.filter((pl.col('Date')>=beginTrain) & (pl.col('Date')<=endTrain))
wtiTest = wti.filter((pl.col('Date')>=beginTest) & (pl.col('Date')<=endTest))
```

```
print("Beginning of wti training set")
print(wtiTrain.head())
print("End of wti training set")
print(wtiTrain.tail())
print("Beginning of wti test set")
print(wtiTest.head())
print("End of wti test set")
print(wtiTest.tail())
```

```
➡ Beginning of wti training set
shape: (5, 17)
```

| Date                     | CloseLag1 | CloseLag2 | CloseLag3 | ... | CloseEMA2 | CloseEMA4 | CloseEMA8 | Target |
|--------------------------|-----------|-----------|-----------|-----|-----------|-----------|-----------|--------|
| ---<br>datetime[μs, UTC] | f64       | f64       | f64       |     | f64       | f64       | f64       | i32    |
| 2006-05-02 04:00:00 UTC  | 32.19     | 31.062    | 30.669    | ... | 31.649    | 31.684    | 31.697    | 1      |
| 2006-05-03               | 33.471    | 32.19     | 31.062    | ... | 32.56     | 32.208    | 31.979    | 0      |

|                               |        |        |        |     |        |        |        |   |
|-------------------------------|--------|--------|--------|-----|--------|--------|--------|---|
| 04:00:00<br>UTC               |        |        |        |     |        |        |        |   |
| 2006-05-04<br>04:00:00<br>UTC | 33.158 | 33.471 | 32.19  | ... | 32.859 | 32.486 | 32.167 | 1 |
| 2006-05-05<br>04:00:00<br>UTC | 35.013 | 33.158 | 33.471 | ... | 33.936 | 33.226 | 32.62  | 0 |
| 2006-05-08<br>04:00:00<br>UTC | 33.645 | 35.013 | 33.158 | ... | 33.791 | 33.349 | 32.783 | 0 |

| Date<br>---<br>datetime[μs<br>, UTC] | CloseLag1<br>---<br>f64 | CloseLag2<br>---<br>f64 | CloseLag3<br>---<br>f64 | ... | CloseEMA2<br>---<br>f64 | CloseEMA4<br>---<br>f64 | CloseEMA8<br>---<br>f64 | Target<br>---<br>i32 |
|--------------------------------------|-------------------------|-------------------------|-------------------------|-----|-------------------------|-------------------------|-------------------------|----------------------|
| 2024-12-23<br>05:00:00<br>UTC        | 1.413                   | 1.344                   | 1.442                   | ... | 1.42                    | 1.483                   | 1.585                   | 1                    |
| 2024-12-24<br>05:00:00<br>UTC        | 1.481                   | 1.413                   | 1.344                   | ... | 1.451                   | 1.482                   | 1.569                   | 1                    |
| 2024-12-26<br>05:00:00<br>UTC        | 1.501                   | 1.481                   | 1.413                   | ... | 1.476                   | 1.488                   | 1.558                   | 1                    |
| 2024-12-27<br>05:00:00<br>UTC        | 1.54                    | 1.501                   | 1.481                   | ... | 1.508                   | 1.503                   | 1.555                   | 1                    |
| 2024-12-30<br>05:00:00<br>UTC        | 1.579                   | 1.54                    | 1.501                   | ... | 1.544                   | 1.525                   | 1.559                   | 1                    |

| Date<br>---<br>datetime[μs<br>, UTC] | CloseLag1<br>---<br>f64 | CloseLag2<br>---<br>f64 | CloseLag3<br>---<br>f64 | ... | CloseEMA2<br>---<br>f64 | CloseEMA4<br>---<br>f64 | CloseEMA8<br>---<br>f64 | Target<br>---<br>i32 |
|--------------------------------------|-------------------------|-------------------------|-------------------------|-----|-------------------------|-------------------------|-------------------------|----------------------|
| 2025-01-02<br>05:00:00               | 1.628                   | 1.638                   | 1.579                   | ... | 1.61                    | 1.579                   | 1.581                   | 1                    |

Using the dates defined by looking across all ETFs and the WTI time series.

```
print(wtiTest['Target'].value_counts())
yTest = np.array(wtiTest['Target'])
```

| Target<br>--- | count<br>--- |
|---------------|--------------|
| i32           | u32          |
| 1             | 2227         |
| 0             | 2471         |

| Target<br>--- | count<br>--- |
|---------------|--------------|
| i32           | u32          |
| 1             | 72           |
| 0             | 85           |

The initial feature set will include price features from WTI. Having obtained the Target from both the training and test data for WTI, we remove it from the WTI feature set in both the training and test sets.

```
# Having extracted the Target from the WTI training and test sets, we can delete it from the set of features
```

```
XTrain = wtiTrain.drop(['Target'])
```

```
XTrain.head()
```

```
XTest = wtiTest.drop(['Target'])
```

```
XTest.head()
```

↗ shape: (5, 16)

| Date                    | CloseLag1 | CloseLag2 | CloseLag3 | HMLLag1 | HMLLag2 | HMLLag3 | OMCLag1 | OMCLag2 | OMCLag3 | VolumeLag1 | VolumeLag2 | VolumeLag3 |
|-------------------------|-----------|-----------|-----------|---------|---------|---------|---------|---------|---------|------------|------------|------------|
| datetime[μs, UTC]       | f64       | f64       | f64       | f64     | f64     | f64     | f64     | f64     | f64     | f64        | f64        | f64        |
| 2025-01-02 05:00:00 UTC | 1.628     | 1.638     | 1.579     | 0.059   | 0.108   | 0.098   | 0.0     | -0.029  | -0.01   | 2.2437e6   | 3.6429e6   | 2.5184e6   |
| 2025-01-03 05:00:00 UTC | 1.746     | 1.628     | 1.638     | 0.128   | 0.059   | 0.108   | -0.078  | 0.0     | -0.029  | 3.4821e6   | 2.2437e6   | 3.6429e6   |
| 2025-01-06 05:00:00 UTC | 1.727     | 1.746     | 1.628     | 0.088   | 0.128   | 0.059   | 0.039   | -0.078  | 0.0     | 1.4409e6   | 3.4821e6   | 2.2437e6   |

```
# filter by date for all the ETF DataFrames, defining training and test sets
```

```
spyTrain = spy.filter((pl.col('Date')>=beginTrain) & (pl.col('Date')<=endTrain))
```

```
spyTest = spy.filter((pl.col('Date')>=beginTest) & (pl.col('Date')<=endTest))
```

```
gldTrain = gld.filter((pl.col('Date')>=beginTrain) & (pl.col('Date')<=endTrain))
```

```
gldTest = gld.filter((pl.col('Date')>=beginTest) & (pl.col('Date')<=endTest))
```

```
vgtTrain = vgt.filter((pl.col('Date')>=beginTrain) & (pl.col('Date')<=endTrain))
```

```
vgtTest = vgt.filter((pl.col('Date')>=beginTest) & (pl.col('Date')<=endTest))
```

```
vbTrain = vb.filter((pl.col('Date')>=beginTrain) & (pl.col('Date')<=endTrain))
```

```
vbTest = vb.filter((pl.col('Date')>=beginTest) & (pl.col('Date')<=endTest))
```

```
iveTrain = ive.filter((pl.col('Date')>=beginTrain) & (pl.col('Date')<=endTrain))
```

```
iveTest = ive.filter((pl.col('Date')>=beginTest) & (pl.col('Date')<=endTest))
```

```
xliTrain = xli.filter((pl.col('Date')>=beginTrain) & (pl.col('Date')<=endTrain))
```

```
xliTest = xli.filter((pl.col('Date')>=beginTest) & (pl.col('Date')<=endTest))
```

```
xluTrain = xlu.filter((pl.col('Date')>=beginTrain) & (pl.col('Date')<=endTrain))
```

```
xluTest = xlu.filter((pl.col('Date')>=beginTest) & (pl.col('Date')<=endTest))
```

```
slvTrain = slv.filter((pl.col('Date')>=beginTrain) & (pl.col('Date')<=endTrain))
```

```
slvTest = slv.filter((pl.col('Date')>=beginTest) & (pl.col('Date')<=endTest))
```

```
usoTrain = uso.filter((pl.col('Date')>=beginTrain) & (pl.col('Date')<=endTrain))
```

```
usoTest = uso.filter((pl.col('Date')>=beginTest) & (pl.col('Date')<=endTest))
```

```
# verify consistent shapes of DataFrames
```

```
print("Train DataFrame shapes:")
```

```
print("wtiTrain.shape: ", wtiTrain.shape)
```

```
print("spyTrain.shape: ", spyTrain.shape)
```

```
print("gldTrain.shape: ", gldTrain.shape)
```

```
print("vgtTrain.shape: ", vgtTrain.shape)
```

```
print("vbTrain.shape: ", vbTrain.shape)
```

```
print("iveTrain.shape: ", iveTrain.shape)
```

```
print("xliTrain.shape: ", xliTrain.shape)
```

```
print("xluTrain.shape: ", xluTrain.shape)
```

```
print("slvTrain.shape: ", slvTrain.shape)
```

```
print("usoTrain.shape: ", usoTrain.shape)
```

```
# verify consistent shapes of DataFrames
```

```
print("Test DataFrame shapes:")
```

```
print("wtiTest.shape: ", wtiTest.shape)
```

```
print("spyTest.shape: ", spyTest.shape)
```

```
print("gldTest.shape: ", gldTest.shape)
```

```
print("vgtTest.shape: ", vgtTest.shape)
```

```
print("vbTest.shape: ", vbTest.shape)
```

```
print("iveTest.shape: ", iveTest.shape)
```

```
print("xliTest.shape: ", xliTest.shape)
```

```

print("xluTest.shape: ", xluTest.shape)
print("slvTest.shape: ", slvTest.shape)
print("usoTest.shape: ", usoTest.shape)

```

```

↗ Train DataFrame shapes:
wtiTrain.shape: (4698, 17)
spyTrain.shape: (4698, 9)
gldTrain.shape: (4698, 9)
vgtTrain.shape: (4698, 9)
vbTrain.shape: (4698, 9)
iveTrain.shape: (4698, 9)
xliTrain.shape: (4698, 9)
xluTrain.shape: (4698, 9)
slvTrain.shape: (4698, 9)
usoTrain.shape: (4698, 9)
Test DataFrame shapes:
wtiTest.shape: (157, 17)
spyTest.shape: (157, 9)
gldTest.shape: (157, 9)
vgtTest.shape: (157, 9)
vbTest.shape: (157, 9)
iveTest.shape: (157, 9)
xliTest.shape: (157, 9)
xluTest.shape: (157, 9)
slvTest.shape: (157, 9)
usoTest.shape: (157, 9)

```

```

# Add Up indicator variables to the wti DataFrames
wtiTrainETF = wtiTrain
wtiTrainETF = wtiTrainETF.with_columns(spyTrain['SPYUp'])
wtiTrainETF = wtiTrainETF.with_columns(gldTrain['GLDUp'])
wtiTrainETF = wtiTrainETF.with_columns(vgtTrain['VGTUp'])
wtiTrainETF = wtiTrainETF.with_columns(vbTrain['VBUp'])
wtiTrainETF = wtiTrainETF.with_columns(iveTrain['IVEUp'])
wtiTrainETF = wtiTrainETF.with_columns(xliTrain['XLIUp'])
wtiTrainETF = wtiTrainETF.with_columns(xluTrain['XLUUp'])
wtiTrainETF = wtiTrainETF.with_columns(slvTrain['SLVUp'])
wtiTrainETF = wtiTrainETF.with_columns(usoTrain['USOUp'])
print(wtiTrainETF.schema)
print(wtiTrainETF.head())

```

```

wtiTestETF = wtiTest
wtiTestETF = wtiTestETF.with_columns(spyTest['SPYUp'])
wtiTestETF = wtiTestETF.with_columns(gldTest['GLDUp'])
wtiTestETF = wtiTestETF.with_columns(vgtTest['VGTUp'])
wtiTestETF = wtiTestETF.with_columns(vbTest['VBUp'])
wtiTestETF = wtiTestETF.with_columns(iveTest['IVEUp'])
wtiTestETF = wtiTestETF.with_columns(xliTest['XLIUp'])
wtiTestETF = wtiTestETF.with_columns(xluTest['XLUUp'])
wtiTestETF = wtiTestETF.with_columns(slvTest['SLVUp'])
wtiTestETF = wtiTestETF.with_columns(usoTest['USOUp'])
print(wtiTestETF.schema)
print(wtiTestETF.head())

```

```

↗ Schema({'Date': Datetime(time_unit='us', time_zone='UTC'), 'CloseLag1': Float64, 'CloseLag2': Float64, 'CloseLag3': Float64, 'HMLLag1':
shape: (5, 26)

```

| Date                    | CloseLag1 | CloseLag2 | CloseLag3 | ... | XLIUp | XLUUp | SLVUp | USOUp |
|-------------------------|-----------|-----------|-----------|-----|-------|-------|-------|-------|
| ---                     | ---       | ---       | ---       | ... | ---   | ---   | ---   | ---   |
| datetime[μs, UTC]       | f64       | f64       | f64       | ... | i32   | i32   | i32   | i32   |
| 2006-05-02 04:00:00 UTC | 32.19     | 31.062    | 30.669    | ... | 1     | 0     | 1     | 1     |
| 2006-05-03 04:00:00 UTC | 33.471    | 32.19     | 31.062    | ... | 1     | 1     | 1     | 1     |
| 2006-05-04 04:00:00 UTC | 33.158    | 33.471    | 32.19     | ... | 1     | 0     | 0     | 0     |
| 2006-05-05 04:00:00 UTC | 35.013    | 33.158    | 33.471    | ... | 1     | 1     | 1     | 0     |
| 2006-05-08 04:00:00 UTC | 33.645    | 35.013    | 33.158    | ... | 1     | 1     | 0     | 0     |

```

Schema({'Date': Datetime(time_unit='us', time_zone='UTC'), 'CloseLag1': Float64, 'CloseLag2': Float64, 'CloseLag3': Float64, 'HMLLag1':
shape: (5, 26)

```

| Date                    | CloseLag1 | CloseLag2 | CloseLag3 | ... | XLIUp | XLUUp | SLVUp | USOUp |
|-------------------------|-----------|-----------|-----------|-----|-------|-------|-------|-------|
| ---                     | ---       | ---       | ---       | ... | ---   | ---   | ---   | ---   |
| datetime[μs, UTC]       | f64       | f64       | f64       | ... | i32   | i32   | i32   | i32   |
| 2025-01-02 05:00:00 UTC | 1.628     | 1.638     | 1.579     | ... | 0     | 0     | 0     | 1     |
| 2025-01-03 05:00:00 UTC | 1.746     | 1.628     | 1.638     | ... | 0     | 1     | 1     | 1     |
| 2025-01-06 05:00:00 UTC | 1.727     | 1.746     | 1.628     | ... | 1     | 1     | 1     | 1     |



|                         |       |       |       |     |   |   |   |   |
|-------------------------|-------|-------|-------|-----|---|---|---|---|
| 2025-01-07 05:00:00 UTC | 1.619 | 1.727 | 1.746 | ... | 0 | 0 | 1 | 0 |
| 2025-01-08 05:00:00 UTC | 1.599 | 1.619 | 1.727 | ... | 0 | 0 | 1 | 1 |

```
# we can now drop Date and Target from wti training and test sets
# as we now have the full set o features for training and testing
wtiTrainETF = wtiTrainETF.drop(['Date','Target'])
wtiTestETF = wtiTestETF.drop(['Date','Target'])
```

```
trainStatistics = wtiTrainETF.describe()
print(wtiTrainETF.columns)
trainStatisticsToPrint = trainStatistics.transpose(include_header=True).drop(['column_1', 'column_5', 'column_6', 'column_7'])
print(trainStatisticsToPrint.schema)
with pl.Config(
    tbl_rows = 60,
    tbl_width_chars = 200,
    tbl_cols = -1,
    float_precision = 3,
    tbl_hide_dataframe_shape = True,
    tbl_hide_column_data_types = True):
    print(trainStatisticsToPrint)
```

```
# print a few records at the beginning of the DataFrame
print(wtiTrainETF.head())
```

```
➡ ['CloseLag1', 'CloseLag2', 'CloseLag3', 'HMLLag1', 'HMLLag2', 'HMLLag3', 'OMCLag1', 'OMCLag2', 'OMCLag3', 'VolumeLag1', 'VolumeLag2', 'V
Schema{'column': String, 'column_0': String, 'column_2': String, 'column_3': String, 'column_4': String, 'column_8': String})
```

| column     | column_0 | column_2            | column_3            | column_4 | column_8   |
|------------|----------|---------------------|---------------------|----------|------------|
| statistic  | count    | mean                | std                 | min      | max        |
| CloseLag1  | 4698.0   | 9.434744146445293   | 7.903321463290654   | 1.047    | 44.172     |
| CloseLag2  | 4698.0   | 9.441019795657727   | 7.9087858952326995  | 1.047    | 44.172     |
| CloseLag3  | 4698.0   | 9.447220093656876   | 7.91400686144687    | 1.047    | 44.172     |
| HMLLag1    | 4698.0   | 0.45111323967645806 | 0.43053528722285817 | 0.029    | 4.129      |
| HMLLag2    | 4698.0   | 0.4514563644103873  | 0.4308960144294226  | 0.029    | 4.129      |
| HMLLag3    | 4698.0   | 0.45179480630055346 | 0.4312426208526598  | 0.029    | 4.129      |
| OMCLag1    | 4698.0   | 0.01769391230310771 | 0.37248492041266446 | -3.223   | 3.705      |
| OMCLag2    | 4698.0   | 0.01758918688037466 | 0.3725618641070302  | -3.223   | 3.705      |
| OMCLag3    | 4698.0   | 0.01758407833120477 | 0.37256294191238093 | -3.223   | 3.705      |
| VolumeLag1 | 4698.0   | 1846607.4499787143  | 1858696.1408019532  | 52700.0  | 40429700.0 |
| VolumeLag2 | 4698.0   | 1846130.779054917   | 1858810.9684406945  | 52700.0  | 40429700.0 |
| VolumeLag3 | 4698.0   | 1845878.9272030653  | 1858934.5724908514  | 52700.0  | 40429700.0 |
| CloseEMA2  | 4698.0   | 9.441036824180502   | 7.901227922219755   | 1.156    | 43.432     |
| CloseEMA4  | 4698.0   | 9.450132183908048   | 7.900732474686015   | 1.285    | 43.018     |
| CloseEMA8  | 4698.0   | 9.46854576415496    | 7.899496747575      | 1.382    | 42.696     |
| SPYUp      | 4698.0   | 0.5506598552575565  | 0.49747990726518226 | 0.0      | 1.0        |
| GLDUp      | 4698.0   | 0.5259684972328651  | 0.4993783325710364  | 0.0      | 1.0        |
| VTGUp      | 4698.0   | 0.5576841209025117  | 0.49671426317032225 | 0.0      | 1.0        |
| VBUp       | 4698.0   | 0.5357598978288634  | 0.49877267659502655 | 0.0      | 1.0        |
| IVEUp      | 4698.0   | 0.5340570455512984  | 0.4988918682142226  | 0.0      | 1.0        |
| XLIUp      | 4698.0   | 0.5398041719880801  | 0.49846616338488664 | 0.0      | 1.0        |
| XLUUp      | 4698.0   | 0.5363984674329502  | 0.4987264730981392  | 0.0      | 1.0        |
| SLVUp      | 4698.0   | 0.5163899531715623  | 0.4997844912621346  | 0.0      | 1.0        |
| USOUp      | 4698.0   | 0.511068539804172   | 0.4999306820015263  | 0.0      | 1.0        |

shape: (5, 24)

| CloseLag1 | CloseLag2 | CloseLag3 | HMLLag1 | ... | XLIUp | XLUUp | SLVUp | USOUp |
|-----------|-----------|-----------|---------|-----|-------|-------|-------|-------|
| ---       | ---       | ---       | ---     |     | ---   | ---   | ---   | ---   |
| f64       | f64       | f64       | f64     |     | i32   | i32   | i32   | i32   |
| 32.19     | 31.062    | 30.669    | 1.128   | ... | 1     | 0     | 1     | 1     |
| 33.471    | 32.19     | 31.062    | 1.681   | ... | 1     | 1     | 1     | 1     |
| 33.158    | 33.471    | 32.19     | 1.375   | ... | 1     | 0     | 0     | 0     |
| 35.013    | 33.158    | 33.471    | 1.688   | ... | 1     | 1     | 1     | 0     |
| 33.645    | 35.013    | 33.158    | 1.899   | ... | 1     | 1     | 0     | 0     |

```
testStatistics = wtiTestETF.describe()
print(wtiTestETF.columns)
testStatisticsToPrint = testStatistics.transpose(include_header=True).drop(['column_1', 'column_5', 'column_6', 'column_7'])
print(testStatisticsToPrint.schema)
with pl.Config(
    tbl_rows = 60,
    tbl_width_chars = 200,
    tbl_cols = -1,
```

```
float_precision = 3,
tbl_hide_dataframe_shape = True,
tbl_hide_column_data_types = True):
print(testStatisticsToPrint)
```

```
# print a few records at the beginning of the DataFrame
print(wtiTestETF.head())
```

```
Schema({'column': String, 'column_0': String, 'column_2': String, 'column_3': String, 'column_4': String, 'column_8': String})
```

| column     | column_0 | column_2             | column_3             | column_4 | column_8  |
|------------|----------|----------------------|----------------------|----------|-----------|
| statistic  | count    | mean                 | std                  | min      | max       |
| CloseLag1  | 157.0    | 1.583496815286624    | 0.23880529630030126  | 1.096    | 2.346     |
| CloseLag2  | 157.0    | 1.5828471337579617   | 0.23851536438187623  | 1.096    | 2.346     |
| CloseLag3  | 157.0    | 1.5820127388535032   | 0.23829673344386137  | 1.096    | 2.346     |
| HMLLag1    | 157.0    | 0.08693630573248408  | 0.05223058853366095  | 0.03     | 0.398     |
| HMLLag2    | 157.0    | 0.08730573248407644  | 0.052172748365944654 | 0.03     | 0.398     |
| HMLLag3    | 157.0    | 0.08761146496815288  | 0.05209331490177502  | 0.03     | 0.398     |
| OMCLag1    | 157.0    | 0.004267515923566881 | 0.06086306426616007  | -0.119   | 0.288     |
| OMCLag2    | 157.0    | 0.004210191082802551 | 0.06089029983030787  | -0.119   | 0.288     |
| OMCLag3    | 157.0    | 0.003891719745222931 | 0.060832639225176156 | -0.119   | 0.288     |
| VolumeLag1 | 157.0    | 1774694.2675159236   | 1436394.6061296004   | 469000.0 | 9514600.0 |
| VolumeLag2 | 157.0    | 1793519.1082802548   | 1441411.6644551635   | 469000.0 | 9514600.0 |
| VolumeLag3 | 157.0    | 1805398.7261146498   | 1439638.7979587486   | 469000.0 | 9514600.0 |
| CloseEMA2  | 157.0    | 1.582611464968153    | 0.2311157066659726   | 1.111    | 2.235     |
| CloseEMA4  | 157.0    | 1.5808598726114649   | 0.2211442694956234   | 1.136    | 2.122     |
| CloseEMA8  | 157.0    | 1.5780191082802548   | 0.20381714628488098  | 1.157    | 1.988     |
| SPYUp      | 157.0    | 0.5668789808917197   | 0.4970926415014129   | 0.0      | 1.0       |
| GLDUp      | 157.0    | 0.5796178343949044   | 0.49519988886944     | 0.0      | 1.0       |
| VGTOp      | 157.0    | 0.5605095541401274   | 0.4979133332299239   | 0.0      | 1.0       |
| VBUUp      | 157.0    | 0.4968152866242038   | 0.5015898291312316   | 0.0      | 1.0       |
| IVEUp      | 157.0    | 0.5414012738853503   | 0.49987749601678194  | 0.0      | 1.0       |
| XLIUp      | 157.0    | 0.535031847133758    | 0.5003673319951818   | 0.0      | 1.0       |
| XLUUp      | 157.0    | 0.5668789808917197   | 0.4970926415014129   | 0.0      | 1.0       |
| SLVUp      | 157.0    | 0.535031847133758    | 0.5003673319951818   | 0.0      | 1.0       |
| USOUUp     | 157.0    | 0.4840764331210191   | 0.5013455681821809   | 0.0      | 1.0       |

```
shape: (5, 24)
```

| CloseLag1 | CloseLag2 | CloseLag3 | HMLLag1 | ... | XLIUp | XLUUp | SLVUp | USOUUp |
|-----------|-----------|-----------|---------|-----|-------|-------|-------|--------|
| ---       | ---       | ---       | ---     | ... | ---   | ---   | ---   | ---    |
| f64       | f64       | f64       | f64     | ... | i32   | i32   | i32   | i32    |
| 1.628     | 1.638     | 1.579     | 0.059   | ... | 0     | 0     | 0     | 1      |
| 1.746     | 1.628     | 1.638     | 0.128   | ... | 0     | 1     | 1     | 1      |
| 1.727     | 1.746     | 1.628     | 0.088   | ... | 1     | 1     | 1     | 1      |
| 1.619     | 1.727     | 1.746     | 0.167   | ... | 0     | 0     | 1     | 0      |
| 1.599     | 1.619     | 1.727     | 0.098   | ... | 0     | 0     | 1     | 1      |

```
# Standardize features in the training data
featureNames = wtiTrainETF.columns
print("Feature names correspond to Numpy array columns:",featureNames)
scaler = StandardScaler()
XTrain = scaler.fit_transform(np.array(wtiTrainETF))
```

```
Feature names correspond to Numpy array columns: ['CloseLag1', 'CloseLag2', 'CloseLag3', 'HMLLag1', 'HMLLag2', 'HMLLag3', 'OMCLag1', 'OM
```

```
# Standardize features for hold-out test set
featureNames = wtiTestETF.columns
print("Feature names correspond to Numpy array columns:",featureNames)
scaler = StandardScaler()
XTest = scaler.fit_transform(np.array(wtiTestETF))
```

```
Feature names correspond to Numpy array columns: ['CloseLag1', 'CloseLag2', 'CloseLag3', 'HMLLag1', 'HMLLag2', 'HMLLag3', 'OMCLag1', 'OM
```

```
# Defining training features and target for tree-structured ensemble modeling (XGBoost)
X = XTrain # the full training data set with ETS Up indicators
y = yTrain # the cloned values just computed
```

```
# Splitting the X (XTrain) and y (yTrain) data
# into cross-validation train and test sets
# within the Scikit-Learn framework using TimeSeriesSplit
# with a gap, for the number of samples to exclude from
```

```

# the end of each train set and before the next test set.
tscv = TimeSeriesSplit(gap=10, n_splits=5)

all_splits = list(tscv.split(X, y))
train_0, test_0 = all_splits[0]
train_1, test_1 = all_splits[1]
train_2, test_2 = all_splits[2]
train_3, test_3 = all_splits[3]
train_4, test_4 = all_splits[4]

# examine the objects created for cross-validation splits
print("type(all_splits):", type(all_splits), " outer list length", len(all_splits))
print()
print("train_0 has",len(train_0),"with indices from ",min(train_0),"to",max(train_0))
print("test_0 has",len(test_0),"with indices from ",min(test_0),"to",max(test_0))
print()
print("train_1 has",len(train_1),"with indices from ",min(train_1),"to",max(train_1))
print("test_1 has",len(test_1),"with indices from ",min(test_1),"to",max(test_1))
print()
print("train_2 has",len(train_2),"with indices from ",min(train_2),"to",max(train_2))
print("test_2 has",len(test_2),"with indices from ",min(test_2),"to",max(test_2))
print()
print("train_3 has",len(train_3),"with indices from ",min(train_3),"to",max(train_3))
print("test_3 has",len(test_3),"with indices from ",min(test_3),"to",max(test_3))
print()
print("train_4 has",len(train_4),"with indices from ",min(train_4),"to",max(train_4))
print("test_4 has",len(test_4),"with indices from ",min(test_4),"to",max(test_4))

# to see all indices we can uncomment these statements
# print("elements of all_splits list of lists,\n shows index numbers for each the five lists")
# print(all_splits)

```

```

↗ type(all_splits): <class 'list'>  outer list length 5

```

```

train_0 has 773 with indices from  0 to 772
test_0 has 783 with indices from  783 to 1565

train_1 has 1556 with indices from  0 to 1555
test_1 has 783 with indices from  1566 to 2348

train_2 has 2339 with indices from  0 to 2338
test_2 has 783 with indices from  2349 to 3131

train_3 has 3122 with indices from  0 to 3121
test_3 has 783 with indices from  3132 to 3914

train_4 has 3905 with indices from  0 to 3904
test_4 has 783 with indices from  3915 to 4697

```

```

model = XGBClassifier(objective='binary:logistic', n_estimators=1000, random_state=2025)

```

```

# Evaluate a Classification Model Within the Time Series Cross-Validation Design
# Prior to executing a full-blown search for the "best" classification model,
# we test the cross-validation design on a binary classification model,
# revising code provided in online documentation for Scikit-Learn:
# Time-related feature engineering. In particular,
# we define appropriate metrics for assessing classification performance.
def evaluate(model, X, y, cv, model_prop=None, model_step=None):
    cv_results = cross_validate(
        model,
        X,
        y,
        cv=cv,
        scoring=["accuracy"],
        return_estimator=model_prop is not None,
    )
    if model_prop is not None:
        if model_step is not None:
            values = [
                getattr(m[model_step], model_prop) for m in cv_results["estimator"]
            ]
        else:
            values = [getattr(m, model_prop) for m in cv_results["estimator"]]
        print(f"Mean model.{model_prop} = {np.mean(values)}")
    accuracy = -cv_results["test_accuracy"]

# print used in earlier testing

```

```

# print(
#     f"Mean Accuracy:      {-accuracy.mean():.3f} +/- {-accuracy.std():.3f}\n"
# )
return (-accuracy.mean(), accuracy.std())

evaluate(model, X, y, cv=tscv, model_prop="n_estimators")

➡ Mean model.n_estimators = 1000.0
   (np.float64(0.4947637292464878), np.float64(0.02598091670999012))

# print results from evaluate for the model with default hyperparameter settings
accuracyMean, accuracyStd = evaluate(model, X, y, cv=tscv, model_prop="n_estimators")
print(
    f"Mean Accuracy:      {accuracyMean:.3f} +/- {-accuracyStd:.3f}\n"
)

➡ Mean model.n_estimators = 1000.0
   Mean Accuracy:      0.495 +/- 0.026

# Randomized search to find the best set of hyperparameters

param_dist = {
    'max_depth': randint(3, 10),
    'min_child_weight': randint(1, 10),
    'subsample': uniform(0.5, 1),
    'learning_rate': uniform(0.01, 0.1),
    'n_estimators': randint(100, 1000),
}
xgb_model = xgb.XGBClassifier(objective='binary:logistic', use_label_encoder=False, eval_metric='logloss', random_state=2025)

random_search = RandomizedSearchCV(
    estimator=xgb_model,
    param_distributions=param_dist,
    n_iter=100, # Number of parameter settings that are sampled.
    scoring='accuracy',
    cv = TimeSeriesSplit(gap=10, n_splits=5),
    random_state=2025,
    n_jobs=-1 # Use all available cores
)

random_search.fit(X, y)


print("Best parameters:", random_search.best_params_)
print("Best score:", random_search.best_score_)

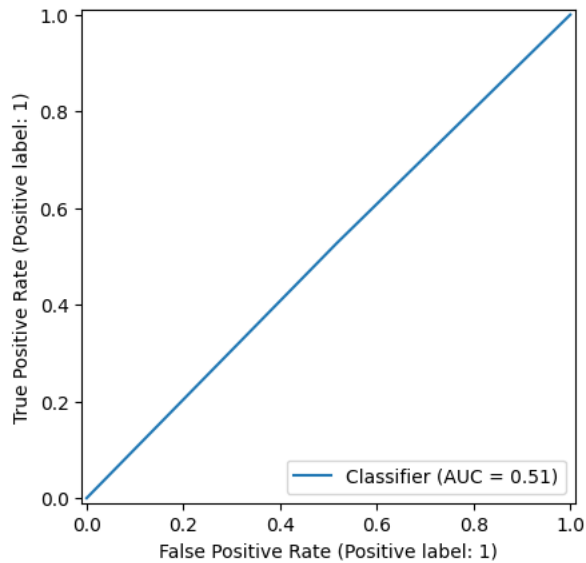
➡ Best parameters: {'learning_rate': np.float64(0.05637364430976246), 'max_depth': 6, 'min_child_weight': 1, 'n_estimators': 691, 'subsample': 0.5970606684977592}
   Best score: 0.5106002554278416

# Evaluate Model Classification Performance in the Test Set with "Best" hyperparameter settings
# final model evaluation
finalModel = XGBClassifier(objective='binary:logistic', eval_metric='logloss', random_state=2025,
                           max_depth = 6, min_child_weight = 1, subsample = 0.5970606684977592, learning_rate = 0.05637364430976246, n_estimators = 691)

finalModel.fit(X, y) # fit to the training data
ypred = finalModel.predict(XTest) # predictions on the hold-out test data
RocCurveDisplay.from_predictions(yTest, ypred)


```

 <sklearn.metrics.\_plot.roc\_curve.RocCurveDisplay at 0x7ebc004bf740>

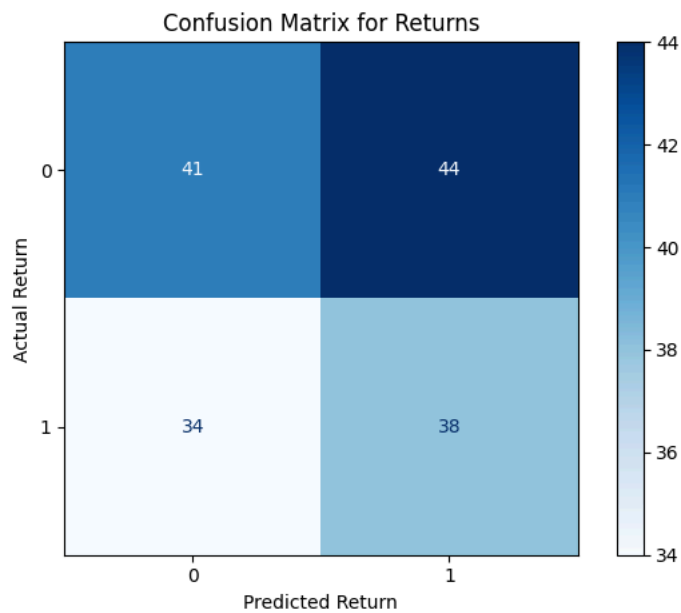


```
print("Confusion Matrix")
print(confusion_matrix(yTest, ypred))
disp = ConfusionMatrixDisplay.from_predictions(yTest, ypred,
                                              display_labels=["0", "1"],
                                              cmap = plt.cm.Blues)


plt.title("Confusion Matrix for Returns")
plt.xlabel("Predicted Return")
plt.ylabel("Actual Return")
plt.tight_layout()
plt.show()
```

 Confusion Matrix

```
[[41 44]
 [34 38]]
```



```
print(classification_report(yTest, ypred, labels = ["0", "1"]))
```



|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.55      | 0.48   | 0.51     | 85      |
| 1            | 0.46      | 0.53   | 0.49     | 72      |
| micro avg    | 0.50      | 0.50   | 0.50     | 157     |
| macro avg    | 0.51      | 0.51   | 0.50     | 157     |
| weighted avg | 0.51      | 0.50   | 0.50     | 157     |

```
# Assess and print feature importances
model.fit(X, y)
print(model.feature_importances_)

[0.0485498  0.05014864 0.04153407 0.04655541 0.0437216  0.04683292
 0.04472881 0.04442059 0.04525414 0.04386874 0.04124043 0.04550721
 0.04161915 0.04029098 0.04140518 0.03696298 0.03331797 0.04327295
 0.03996009 0.0403973  0.03649111 0.03022377 0.03888723 0.03480893]
```

## Ranking Features by Importance

This completes the exploration of additional features being added to the mix. Now we identify which of those features are most important in classifying the next day's return direction (up or even/down).

```
featureNames = wtiTrainETF.columns
print("Feature names correspond to Numpy array columns:",featureNames)

# Get feature importances
importances = np.round(model.feature_importances_, decimals = 3)

# Create Polars DataFrame
importanceDF = pl.DataFrame({"feature": featureNames, "importance": importances})

with pl.Config(
    tbl_rows = 60):
    print(importanceDF.sort("importance", descending=True))
```

```
Feature names correspond to Numpy array columns: ['CloseLag1', 'CloseLag2', 'CloseLag3', 'HMLLag1', 'HMLLag2', 'HMLLag3', 'OMCLag1', 'OM
shape: (24, 2)
```

| feature    | importance |
|------------|------------|
| ---        | ---        |
| str        | f32        |
| CloseLag2  | 0.05       |
| CloseLag1  | 0.049      |
| HMLLag1    | 0.047      |
| HMLLag3    | 0.047      |
| VolumeLag3 | 0.046      |
| OMCLag1    | 0.045      |
| OMCLag3    | 0.045      |
| HMLLag2    | 0.044      |
| OMCLag2    | 0.044      |
| VolumeLag1 | 0.044      |
| VGUp       | 0.043      |
| CloseLag3  | 0.042      |
| CloseEMA2  | 0.042      |
| VolumeLag2 | 0.041      |
| CloseEMA8  | 0.041      |
| CloseEMA4  | 0.04       |
| VBUp       | 0.04       |
| IVEUp      | 0.04       |
| SLVUp      | 0.039      |
| SPYUp      | 0.037      |
| XLIUp      | 0.036      |
| USOUp      | 0.035      |
| GLDUp      | 0.033      |
| XLUUp      | 0.03       |

## Repeat the Modeling Process

For subsequent model development, we can retain the top features and then run additional tests with new features using the procedure demonstrated in this analysis

## References

- [yfinance GitHub](#)
- [yfinance Documentation](#)
- [Polars Online User Guide](#)
- [Build Polars Database](#)

- [YouTube. Polars and Time Series: What It Can Do, and How to Overcome Any Limitation](#)
- [Awesome Quant: Python for Quantitative Finance](#)
- [Cross-validation](#)
- [TimeSeriesSplit](#)
- [RandomizedSearchCV](#)
- [Hyperparameter Tuning](#)
- [Metrics and Scoring](#)
- [Introduction to Boosted Trees](#)
- [XGBoost documentation](#)
- [XGBoost in Python documentation](#)
- [Auto-Sklearn for AutoML in an Scikit-Learn Environment.](#)

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
# list of the ETF tickers
etfs = ['spy', 'gld', 'ive', 'slv', 'uso', 'vb', 'vgt', 'xli', 'xlu']
etf_dfs = {}

# Loop through each ETF to reate the 'Up' feature
for etf in etfs:
    file_path = f"/content/{etf}_daily_data.csv"
    try:
        df = pl.read_csv(file_path, try_parse_dates=True)
        # Drop irrelevant columns
        df = df.drop(['Dividends', 'StockSplits'])
        # Create lagged closing price for previous day
        df = df.with_columns(pl.col('Close').shift(1).alias('CloseLag1'))
        # Create the 'Up' feature based on yesterday's price
        df = df.with_columns(
            (pl.col('Close').shift(-1) > pl.col('Close')).alias(f'{etf}Up')).cast(pl.Int32)
        )
        etf_dfs[etf] = df
    except pl.ColumnNotFoundError as e:
        print(f"Column missing in {etf}: {e}. Skipping this ETF.")
    except Exception as e:
        print(f"Error processing {etf}: {e}. Skipping this ETF.")

if 'spy' in etf_dfs:
    print(etf_dfs['spy'].head())
```

shape: (5, 9)

| Date<br>---<br>datetime[μs<br>, UTC] | Open<br>---<br>f64 | High<br>---<br>f64 | Low<br>---<br>f64 | ... | Volume<br>---<br>i64 | CapitalGain<br>s<br>---<br>f64 | CloseLag1<br>---<br>f64 | spyUp<br>---<br>i32 |
|--------------------------------------|--------------------|--------------------|-------------------|-----|----------------------|--------------------------------|-------------------------|---------------------|
| 2000-01-03<br>05:00:00<br>UTC        | 93.924388          | 93.924388          | 91.152589         | ... | 8164300              | 0.0                            | null                    | 0                   |
| 2000-01-04<br>05:00:00<br>UTC        | 90.934811          | 91.271387          | 88.46989          | ... | 8089800              | 0.0                            | 92.142517               | 1                   |
| 2000-01-05<br>05:00:00<br>UTC        | 88.657974          | 89.6677            | 86.955297         | ... | 12177900             | 0.0                            | 88.539185               | 0                   |
| 2000-01-06<br>05:00:00<br>UTC        | 88.459986          | 89.6479            | 87.272072         | ... | 6227200              | 0.0                            | 88.697571               | 1                   |
| 2000-01-07<br>05:00:00<br>UTC        | 88.895594          | 92.340546          | 88.737205         | ... | 8066500              | 0.0                            | 87.272072               | 1                   |

```
# read prepared wti
wti = pl.read_csv("/content/wti-with-computed-features.csv", try_parse_dates=True)

# merge WTI with each ETF df
for etf, df in etf_dfs.items():
    wti = wti.join(df.select(['Date', f'{etf}Up']), on='Date', how='left')
```

```

# drop NAs
wti = wti.drop_nulls()

# separate features and target
etf_features = [f'{etf}Up' for etf in etfs]
base_features = wti.columns[1:13]
X_columns = base_features + etf_features

X = wti.select(X_columns).to_numpy()
y = wti['Target'].to_numpy()
dates = wti['Date'].to_numpy()

# TimeSeriesSplit with a gap to prevent look-ahead bias
tscv = TimeSeriesSplit(n_splits=5, gap=10)

# initialize CatBoost and XGBoost
models = {
    'CatBoost': CatBoostClassifier(verbose=0, random_state=42),
    'XGBoost': XGBClassifier(random_state=42, eval_metric='logloss', use_label_encoder=False)
}

# Train and evaluate each model using cross-validation
for name, model in models.items():
    print(f"Training and evaluating {name}...")
    cv_results = cross_validate(model, X, y, cv=tscv, scoring=['accuracy', 'roc_auc'], return_train_score=False)
    print(f"{name} Cross-Validation Results:")
    print(f"    Accuracy: {cv_results['test_accuracy'].mean():.4f} +/- {cv_results['test_accuracy'].std():.4f}")
    print(f"    AUC: {cv_results['test_roc_auc'].mean():.4f} +/- {cv_results['test_roc_auc'].std():.4f}")

↔ Training and evaluating CatBoost...
CatBoost Cross-Validation Results:
Accuracy: 0.6494 +/- 0.0917
AUC: 0.7146 +/- 0.1055
Training and evaluating XGBoost...
XGBoost Cross-Validation Results:
Accuracy: 0.7046 +/- 0.0631
AUC: 0.7749 +/- 0.0824

# Re-fit CatBoost model on the full dataset to get feature importance
catboost_model = CatBoostClassifier(verbose=0, random_state=42)
catboost_model.fit(X, y)

# Get feature importance and feature names
feature_importance = catboost_model.get_feature_importance()
sorted_idx = np.argsort(feature_importance)
sorted_features = np.array(X_columns)[sorted_idx]

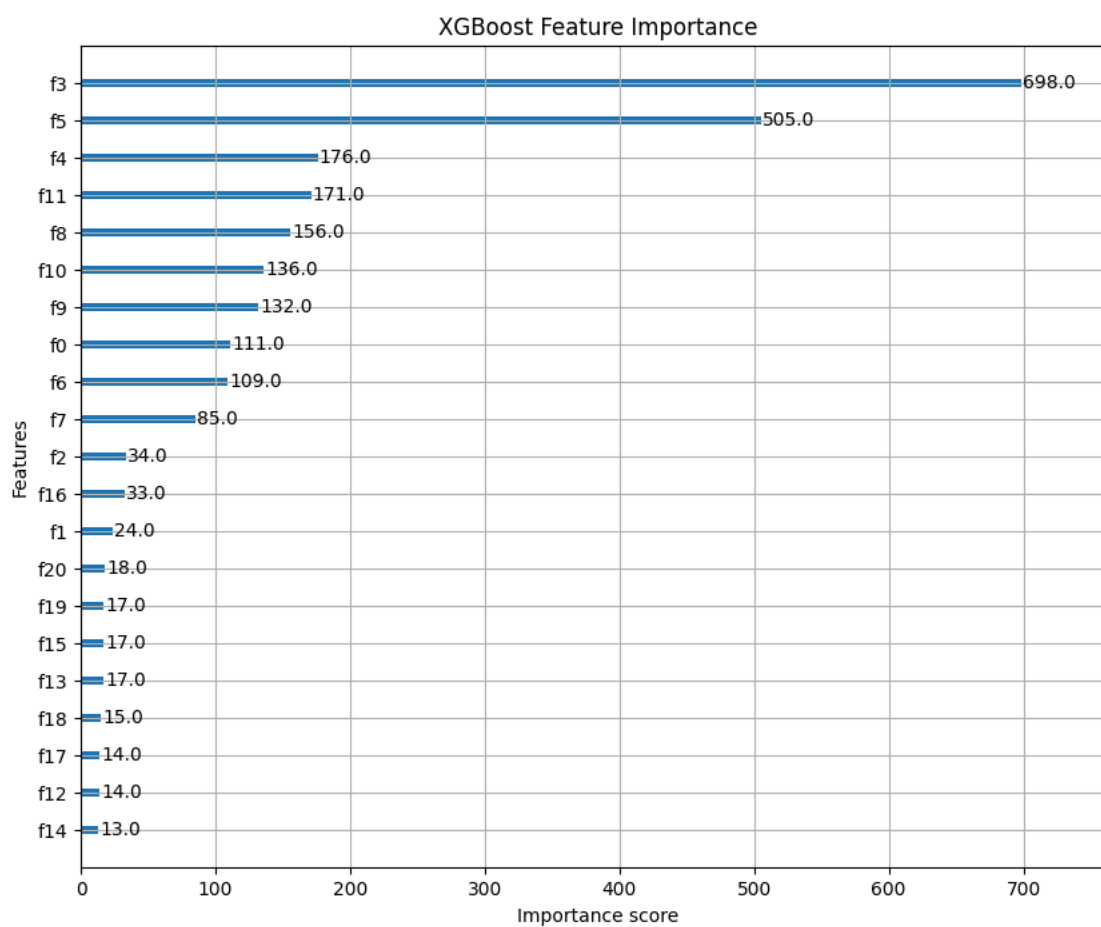
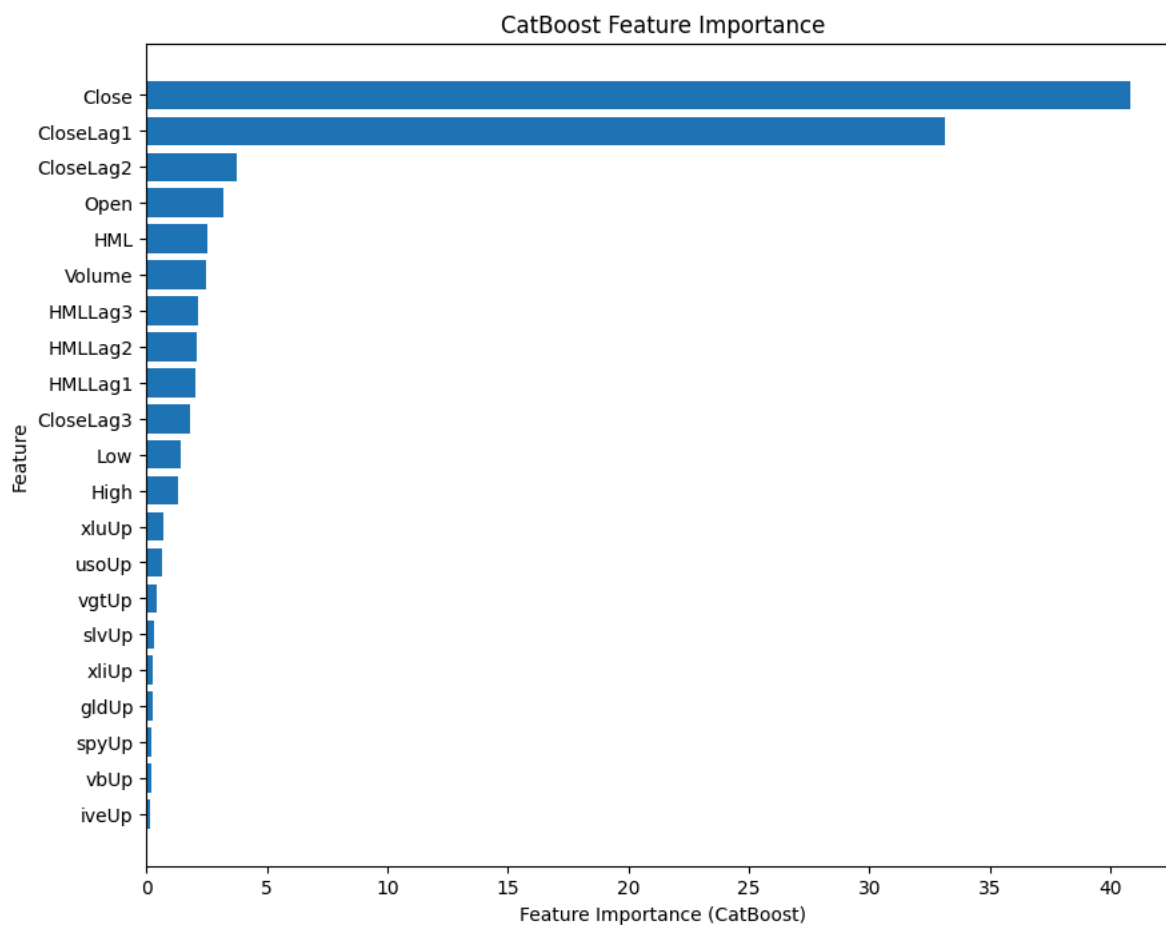
# Plot feature importance for CatBoost
plt.figure(figsize=(10, 8))
plt.barh(sorted_features, feature_importance[sorted_idx])
plt.xlabel("Feature Importance (CatBoost)")
plt.ylabel("Feature")
plt.title("CatBoost Feature Importance")
plt.show()

# Re-fit XGBoost model on the full dataset to get feature importance
xgboost_model = XGBClassifier(random_state=42, eval_metric='logloss', use_label_encoder=False)
xgboost_model.fit(X, y)

# Plot feature importance for XGBoost
plt.figure(figsize=(10, 8))
plot_importance(xgboost_model, ax=plt.gca())
plt.title("XGBoost Feature Importance")
plt.show()

```





apply momentum signal and EMA

```

import polars as pl
import matplotlib.pyplot as plt

def compute_ema(df: pl.DataFrame, column: str, window: int) -> pl.Series:
    """
    Computes the EMA for a given column.
    """
    return df.get_column(column).ewm_mean(com=window)

def create_momentum_signal(df: pl.DataFrame, col_40: str, col_80: str) -> pl.Series:
    """
    Creates a binary momentum signal: 1 if 40-day EMA > 80-day EMA, 0 otherwise.
    """
    return (df[col_40] > df[col_80]).cast(pl.Int32)

def backtest_strategy(data: pl.DataFrame, signal_columns: list, initial_capital: float = 100000.0, risk_level: str = 'aggressive') -> pl.Data:
    """
    Performs a backtest of a multi-level momentum trading strategy based on risk tolerance.

    Args:
        data (pl.DataFrame): The DataFrame with all price and signal data.
        signal_columns (list): A list of columns to use as signals for the strategy.
        initial_capital (float): The starting capital for the simulation.
        risk_level (str): 'aggressive', 'moderate', or 'conservative'.

    Returns:
        pl.DataFrame: A DataFrame with the daily portfolio value.
    """
    cash = initial_capital
    position = 0
    portfolio_history = []

    # Aggressive: Buy if WTI signal is 1
    # Moderate: Buy if WTI signal is 1 AND at least 2 ETF signals are 1
    # Conservative: Buy if WTI signal is 1 AND at least 3 ETF signals are 1

    for i in range(len(data)):
        current_date = data['Date'][i]
        current_price = data['Close'][i]

        # Determine the number of active buy signals for the current day
        active_signals = sum([data[col][i] for col in signal_columns])
        wti_signal = data[signal_columns[0]][i]

        # Check buy condition based on risk level
        buy_condition = False
        if risk_level == 'aggressive' and wti_signal == 1:
            buy_condition = True
        elif risk_level == 'moderate' and wti_signal == 1 and active_signals >= 3:
            buy_condition = True
        elif risk_level == 'conservative' and wti_signal == 1 and active_signals >= 4:
            buy_condition = True

        # Check sell condition
        sell_condition = not buy_condition and position > 0

        # Trading logic
        if buy_condition and position == 0:
            if cash > 0:
                shares_to_buy = int(cash / current_price)
                if shares_to_buy > 0:
                    cash -= shares_to_buy * current_price
                    position += shares_to_buy
        elif sell_condition:
            cash += position * current_price
            position = 0

        # Record portfolio value at the end of the day
        current_value = cash + (position * current_price)
        portfolio_history.append({'date': current_date, 'value': current_value})

    return pl.DataFrame(portfolio_history)

# rUN
if __name__ == "__main__":

```

```

# Load WTI data and create the new EMA features
wti_df = pl.read_csv("/content/wti-with-computed-features.csv", try_parse_dates=True)
wti_df = wti_df.sort("Date")

etfs = ['spy', 'gld', 'ive', 'slv', 'uso', 'vb', 'vgt', 'xli', 'xlu']

# Compute 40-day and 80-day EMAs for WTI
wti_df = wti_df.with_columns(
    compute_ema(wti_df, 'Close', 40).alias('40d_ema'),
    compute_ema(wti_df, 'Close', 80).alias('80d_ema')
)
# Create WTI momentum signal
wti_df = wti_df.with_columns(
    create_momentum_signal(wti_df, '40d_ema', '80d_ema').alias('WTI_signal')
)

# Load ETF data, create momentum signals, and merge
main_df = wti_df
signal_columns = ['WTI_signal']

for ticker in etfs:
    try:
        etf_df = pl.read_csv(f"/content/{ticker}_daily_data.csv", try_parse_dates=True)
        etf_df = etf_df.sort("Date")
        etf_df = etf_df.with_columns(
            compute_ema(etf_df, 'Close', 40).alias(f'{ticker}_40d_ema'),
            compute_ema(etf_df, 'Close', 80).alias(f'{ticker}_80d_ema')
        )
        etf_df = etf_df.with_columns(
            create_momentum_signal(etf_df, f'{ticker}_40d_ema', f'{ticker}_80d_ema').alias(f'{ticker}_signal')
        )
        main_df = main_df.join(etf_df.select(['Date', f'{ticker}_signal']), on='Date', how='left')
        signal_columns.append(f'{ticker}_signal')

    except Exception as e:
        print(f"Could not process {ticker}: {e}")
        continue

# Drop NAs
main_df = main_df.drop_nulls()

# Perform backtests for different risk levels
print("Beginning Backtests...")

# Aggressive Strategy
portfolio_aggressive = backtest_strategy(main_df, signal_columns, risk_level='aggressive')
final_capital_aggressive = portfolio_aggressive.get_column('value')[-1]
print("\nAggressive Strategy Results")
print(f"Final Capital: ${final_capital_aggressive:.2f}")

# Moderate Strategy
portfolio_moderate = backtest_strategy(main_df, signal_columns, risk_level='moderate')
final_capital_moderate = portfolio_moderate.get_column('value')[-1]
print("\nModerate Strategy Results")
print(f"Final Capital: ${final_capital_moderate:.2f}")

# Conservative Strategy
portfolio_conservative = backtest_strategy(main_df, signal_columns, risk_level='conservative')
final_capital_conservative = portfolio_conservative.get_column('value')[-1]
print("\nConservative Strategy Results")
print(f"Final Capital: ${final_capital_conservative:.2f}")

# Plot the results
fig, ax = plt.subplots(figsize=(15, 8))

# Calculate and plot Buy & Hold performance
initial_capital = 100000.0
buy_and_hold_values = (main_df['Close'] / main_df['Close'][0]) * initial_capital
ax.plot(main_df['Date'], buy_and_hold_values, label='Buy & Hold WTI', color='gray', linestyle='--')

ax.plot(portfolio_aggressive.get_column('date'), portfolio_aggressive.get_column('value'), label='Aggressive Strategy', color='red')
ax.plot(portfolio_moderate.get_column('date'), portfolio_moderate.get_column('value'), label='Moderate Strategy', color='orange')
ax.plot(portfolio_conservative.get_column('date'), portfolio_conservative.get_column('value'), label='Conservative Strategy', color='green')

ax.set_title('Algorithmic Trading Strategy Performance vs. Buy & Hold')
ax.set_xlabel('Date')
ax.set_ylabel('Portfolio Value ($)')
ax.grid(True)

```

```

ax.legend()
plt.tight_layout()
plt.show()

```

Beginning Backtests...

Aggressive Strategy Results  
Final Capital: \$44265.67

Moderate Strategy Results  
Final Capital: \$45881.17

Conservative Strategy Results  
Final Capital: \$43900.19



```

import polars as pl
import matplotlib.pyplot as plt

def compute_ema(df: pl.DataFrame, column: str, window: int) -> pl.Series:
    """
    Computes the Exponential Moving Average (EMA) for a given column.
    """
    return df.get_column(column).ewm_mean(com=window)

def create_momentum_signal(df: pl.DataFrame, col_40: str, col_80: str) -> pl.Series:
    """
    Creates a binary momentum signal: 1 if 40-day EMA > 80-day EMA, 0 otherwise.
    """
    return (df[col_40] > df[col_80]).cast(pl.Int32)

def backtest_strategy(data: pl.DataFrame, signal_column: str, initial_capital: float = 100000.0) -> tuple:
    """
    Performs a simple backtest of the momentum trading strategy.
    """
    cash = initial_capital

```

```

cash = initial_cash
position = 0
trade_log = []

# Iterate through the data day by day
for i in range(1, len(data)):
    current_date = data['Date'][i]
    current_price = data['Close'][i]
    previous_signal = data[signal_column][i - 1]
    current_signal = data[signal_column][i]

    # Buy signal: previous signal was 0, current is 1, and no current position
    if previous_signal == 0 and current_signal == 1 and position == 0:
        if cash > 0:
            shares_to_buy = int(cash / current_price)
            if shares_to_buy > 0:
                cost = shares_to_buy * current_price
                cash -= cost
                position += shares_to_buy
                trade_log.append({
                    'date': current_date,
                    'type': 'BUY',
                    'shares': shares_to_buy,
                    'price': current_price,
                    'total_value': cost
                })

    # Sell signal: previous signal was 1, current is 0, and there is a position
    elif previous_signal == 1 and current_signal == 0 and position > 0:
        revenue = position * current_price
        cash += revenue
        trade_log.append({
            'date': current_date,
            'type': 'SELL',
            'shares': position,
            'price': current_price,
            'total_value': revenue
        })
        position = 0

# Close any open position at the end of the simulation
if position > 0:
    final_value = position * data['Close'][len(data) - 1]
    cash += final_value
    trade_log.append({
        'date': data['Date'][len(data) - 1],
        'type': 'FINAL SELL',
        'shares': position,
        'price': data['Close'][len(data) - 1],
        'total_value': final_value
    })

final_capital = cash + (position * data['Close'][len(data) - 1])
trade_df = pl.DataFrame(trade_log)

return trade_df, final_capital

# Run
if __name__ == "__main__":

    # Load WTI data and create the new EMA features
    wti_df = pl.read_csv("/content/wti-with-computed-features.csv", try_parse_dates=True)
    wti_df = wti_df.sort("Date")

    # Compute 40-day and 80-day EMAs for WTI
    wti_df = wti_df.with_columns(
        compute_ema(wti_df, 'Close', 40).alias('40d_ema'),
        compute_ema(wti_df, 'Close', 80).alias('80d_ema')
    )

    # Create the WTI momentum signal
    wti_df = wti_df.with_columns(
        create_momentum_signal(wti_df, '40d_ema', '80d_ema').alias('WTIMomentumSignal')
    )

    # Perform a backtest on the WTI momentum strategy
    trade_log, final_capital = backtest_strategy(wti_df.drop_nulls(), 'WTIMomentumSignal')

```

```

print("WTI Momentum Strategy Backtest Results:")
print(trade_log)
print(f"\nInitial Capital: $100,000.00")
print(f"Final Capital: ${final_capital:.2f}")

# Plot results
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(15, 12), sharex=True)

# Plot the WTI closing price and EMAs
ax1.plot(wti_df['Date'], wti_df['Close'], label='WTI Close Price')
ax1.plot(wti_df['Date'], wti_df['40d_ema'], label='40-Day EMA', linestyle='--')
ax1.plot(wti_df['Date'], wti_df['80d_ema'], label='80-Day EMA', linestyle='--')
ax1.set_title('WTI Price and Exponential Moving Averages')
ax1.set_ylabel('Price ($)')
ax1.grid(True)
ax1.legend()

# Plot the momentum signal
ax2.plot(wti_df['Date'], wti_df['WTIMomentumSignal'], label='WTI Momentum Signal', color='purple')
ax2.set_title('WTI Momentum Buy Signal (40d EMA > 80d EMA)')
ax2.set_xlabel('Date')
ax2.set_ylabel('Signal (1=Buy)')
ax2.grid(True)
ax2.legend()
plt.tight_layout()
plt.show()

```