In [121…

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pandas_datareader as web
import datetime as dt

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier   # to run RF
from sklearn.metrics import precision_score   # accuracy metric
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

Loading and Cleaning Data

In [174…

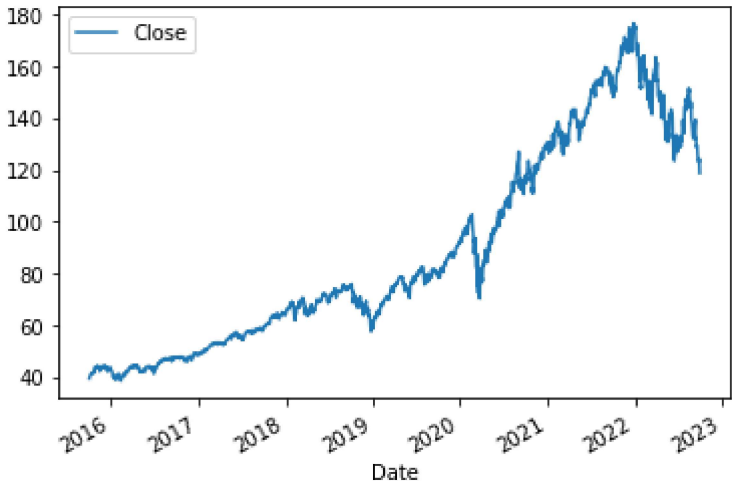
```
# Load Data
company = "XLK"
start = dt.datetime(2015,10,1)
end = dt.datetime(2022,10,1)

data = web.DataReader(company, 'yahoo', start, end)
```

In [175…

```
data.plot.line(y="Close", use_index = True)
```

Out[175…

```
<matplotlib.axes._subplots.AxesSubplot at 0x16d815c1790>
```



Classification over Prediction, since investors would typically care more about getting accurate directional changes instead of the exact price. Could be accurate on price but not whether it will go up or down.

In [176…

```
# Create a column to predict tomorrows price by shifting the Adj Close by 1
data["Tomorrow"] = data["Adj Close"].shift(-1)
data["Target"] = (data["Tomorrow"] > data["Adj Close"]).astype(int)
data
```

Out[176…

Date	High	Low	Open	Close	Volume	Adj Close	Tomorrow	Target

Date	High	Low	Open	Close	Volume	Adj Close	Tomorrow	Target
2015-10-01	39.639999	38.990002	39.500000	39.520000	13518000.0	36.007347	36.517570	1
2015-10-02	40.080002	38.919998	39.060001	40.080002	14917000.0	36.517570	37.237362	1
2015-10-05	40.970001	40.250000	40.330002	40.869999	8568900.0	37.237362	37.282913	1
2015-10-06	41.009998	40.720001	40.730000	40.919998	9857900.0	37.282913	37.456028	1
2015-10-07	41.279999	40.619999	41.139999	41.110001	15218900.0	37.456028	37.629143	1
...	...	...	...	...	...	...	...	...
2022-09-26	125.430000	122.529999	123.360001	122.879997	8596800.0	122.879997	123.040001	1
2022-09-27	125.680000	121.900002	124.510002	123.040001	9357400.0	123.040001	124.339996	1
2022-09-28	124.940002	121.279999	122.220001	124.339996	9466700.0	124.339996	121.099998	0
2022-09-29	122.839996	119.669998	122.559998	121.099998	15548300.0	121.099998	118.779999	0
2022-09-30	122.400002	118.559998	120.610001	118.779999	9755100.0	118.779999	NaN	0

1763 rows × 8 columns

Training Random Forest Model

Random forest- work by training a bunch of individual decision trees with randomized parameters and averaging the result. Resistant to overfitting, run quickly, and can pick up nonlinear tendencies.

In [177…

```
# n_estimator is # of decision trees
# higher typically imporves accuracy up to a limit
# min_sample_split helps against overfitting, but may reduce accuracy
# random_state means that if we run the same model twice, the random numbers generated
model = RandomForestClassifier(n_estimators = 100, min_samples_split = 100, random_stat

train = data.iloc[:-100]
test = data.iloc[-100:]

predictors = ["Adj Close", "Volume", "Open", "High", "Low"]
model.fit(train[predictors], train["Target"])
```

Out[177…

```
RandomForestClassifier(min_samples_split=100, random_state=1)
```

In [178…

```
preds = model.predict(test[predictors])
```

In [179…

```
# Convert to series
preds = pd.DataFrame(preds, index=test.index)
preds
```

Out[179…

Date	0
2022-05-10	1
2022-05-11	1
2022-05-12	1
2022-05-13	0
2022-05-16	0
…	…
2022-09-26	1
2022-09-27	1
2022-09-28	1
2022-09-29	1
2022-09-30	1

100 rows × 1 columns

In [180…

```
precision_score(test["Target"], preds)
```

Out[180…

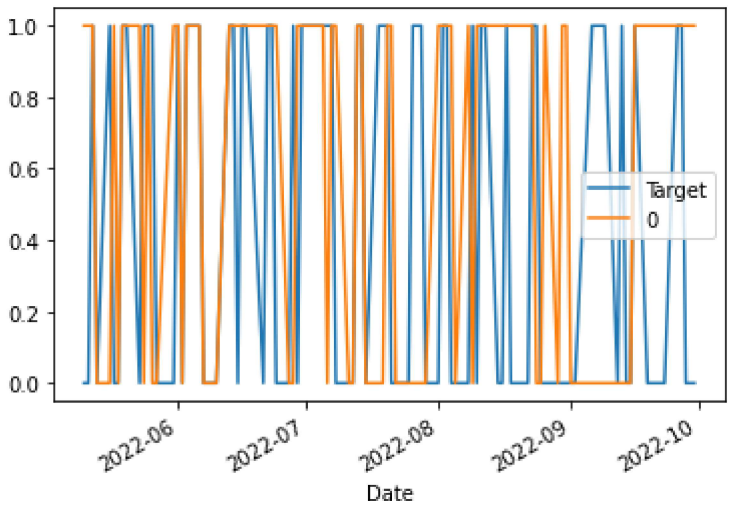
```
0.48333333333333334
```

In [181…

```
combined = pd.concat([test["Target"], preds], axis=1)
combined.plot()
```

Out[181…

```
<matplotlib.axes._subplots.AxesSubplot at 0x16d81e57cd0>
```



Orange shows predictions, blue shows actual. This model seems to have predicted an inconsistant and seemingly random outputs with no clear bias in predicting up or down.

Backtesting and Validation

In [186…]
```
def predict(train, test, predictors, model):
    model.fit(train[predictors], train["Target"])
    preds = model.predict(test[predictors])
    preds = pd.Series(preds, index=test.index, name = "Predictions")
    combined = pd.concat([test["Target"], preds], axis =1)

    return (combined)
```

In [187…]
```
# take ~6 years of data to predict the last year, with around 1 year steps
def backtest(data, model, predictors, start = 1000, step = 250):
    all_predictions = []                                    # create a list

    for i in range(start, data.shape[0], step):             # create train and tes
        train = data.iloc[0:i].copy()
        test = data.iloc[i:(i+step)].copy()
        predictions = predict(train, test, predictors, model)
        all_predictions.append(predictions)                 # append all predictio
    return pd.concat(all_predictions)                       # concatenate all resu
```

In [188…]
```
predictions = backtest(data, model, predictors)
predictions
```

Out[188…]

Date	Target	Predictions
2019-09-23	0	1
2019-09-24	1	0
2019-09-25	1	1
2019-09-26	0	1
2019-09-27	1	0
…	…	…
2022-09-26	1	1
2022-09-27	1	1
2022-09-28	0	1
2022-09-29	0	1
2022-09-30	0	1

763 rows × 2 columns

In [189…]
```
predictions["Predictions"].value_counts()
```

Out[189…
```
1    397
0    366
Name: Predictions, dtype: int64
```

In [190…
```
precision_score(predictions["Target"], predictions["Predictions"])
```

Out[190…
```
0.5088161209068011
```

In [191…
```
predictions["Target"].value_counts() / predictions.shape[0]
```

Out[191…
```
1    0.533421
0    0.466579
Name: Target, dtype: float64
```

If we just buy and sell at the end of the day, this would outperform our initial precision

Improving The Model

In [192…
```
horizons = [2, 5, 60, 250]

new_predictors = []

for horizon in horizons:
    rolling_averages = data.rolling(horizon).mean()

    ratio_column = f"Close_Ratio_{horizon}"
    data[ratio_column] = data["Adj Close"] / rolling_averages["Adj Close"]

    trend_column = f"Trend_{horizon}"
    data[trend_column] = data.shift(1).rolling(horizon).sum()["Target"]

    new_predictors += [ratio_column, trend_column]
```

In [193…
```
# we drop the NAs, so now we have 6 years of data
data = data.dropna()
```

In [194…
```
model = RandomForestClassifier(n_estimators = 200, min_samples_split = 50, random_state
```

In [195…
```
# Here we add more conditions for our predictor function.
# We set a custom threshold. Our model must have more confidence (0.6),
# reducing trading days, but improves accuracy

def predict(train, test, predictors, model):
    model.fit(train[predictors], train["Target"])
    preds = model.predict_proba(test[predictors])[:,1]    #proba returns probability of
    preds[preds >= 0.6] =1
    preds[preds < 0.6] = 0
    preds = pd.Series(preds, index=test.index, name = "Predictions")
    combined = pd.concat([test["Target"], preds], axis =1)
    return combined
```

In [196…

```
predictions = backtest(data, model, new_predictors)
```

In [197…
```
predictions["Predictions"].value_counts()
```

Out[197…
```
0.0    330
1.0    182
Name: Predictions, dtype: int64
```

In [198…
```
precision_score(predictions["Target"], predictions["Predictions"])
```

Out[198…
```
0.5164835164835165
```

From this, we can see that the accuracy improved by around 0.008 or 0.8%

Comparison: Logistic Regression as Baseline

In [199…
```
# Machine Learning
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

In [200…
```
# Copy for new data set to not disturb original
# 6/7 split to train first 6 years and test more recent year.
data2 = data.copy()
split = int((6/7)*len(data2))

X = pd.DataFrame(data2[["Close_Ratio_2", "Close_Ratio_5", "Close_Ratio_60", "Close_Rati
y = data2["Target"]


X_train, X_test, y_train, y_test = X[:split], X[split:], y[:split], y[split:]

model = LogisticRegression()
model = model.fit (X_train,y_train)
```

In [201…
```
pd.DataFrame(zip(X.columns, np.transpose(model.coef_)))
```

Out[201…

	0	1
0	Close_Ratio_2	[-0.46678304148165795]
1	Close_Ratio_5	[-0.5918309723647427]
2	Close_Ratio_60	[0.24611605546633902]
3	Close_Ratio_250	[-0.9744898946325947]

In [202…
```
probability = pd.DataFrame(model.predict_proba(X_test))
probability
```

Out[202…

	0	1
0	0.445409	0.554591

	0	1
1	0.439514	0.560486
2	0.438851	0.561149
3	0.441425	0.558575
4	0.430831	0.569169
...	...	...
211	0.363074	0.636926
212	0.363031	0.636969
213	0.365153	0.634847
214	0.369135	0.630865
215	0.361004	0.638996

216 rows × 2 columns

In [203…

```
predicted = model.predict(X_test)
predicted
```

Out[203…

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [204…

```
print(metrics.confusion_matrix(y_test, predicted))
```

```
[[  0 118]
 [  0  98]]
```

In [205…

```
print(model.score(X_test,y_test))
```

```
0.4537037037037037
```

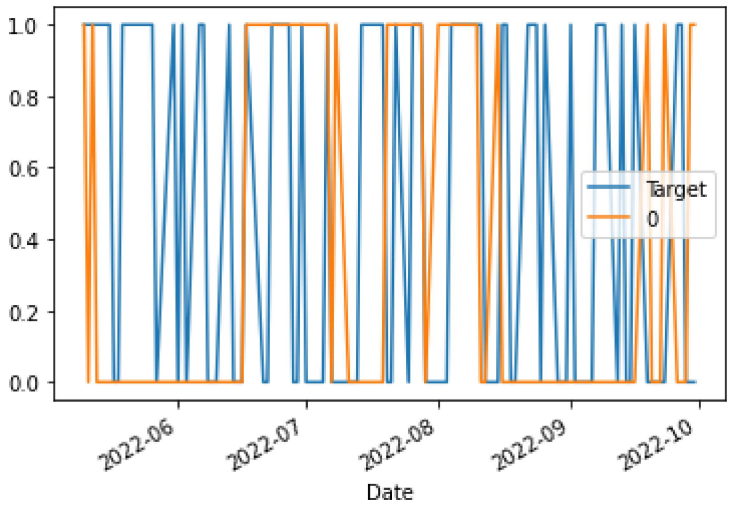
Comparison with Energy

In [206…

```
company = "XLE"
start = dt.datetime(2015,10,1)
end = dt.datetime(2022,10,1)

data3 = web.DataReader(company, 'yahoo', start, end)
data3["Tomorrow"] = data3["Adj Close"].shift(-1)
data3["Target"] = (data3["Tomorrow"] > data3["Adj Close"]).astype(int)
```

In [207…
```
model = RandomForestClassifier(n_estimators = 100, min_samples_split = 100, random_stat

train = data3.iloc[:-100]
test = data3.iloc[-100:]

predictors = ["Adj Close", "Volume", "Open", "High", "Low"]
model.fit(train[predictors], train["Target"])

preds = model.predict(test[predictors])
preds = pd.DataFrame(preds, index=test.index)
precision_score(test["Target"], preds)

combined = pd.concat([test["Target"], preds], axis=1)
combined.plot()
```

Out[207…
```
<matplotlib.axes._subplots.AxesSubplot at 0x16d81567a00>
```



In [219…
```
train = data3.iloc[:-100]
test = data3.iloc[-100:]

predictors = ["Adj Close", "Volume", "Open", "High", "Low"]
model.fit(train[predictors], train["Target"])

preds = model.predict(test[predictors])
preds = pd.DataFrame(preds, index=test.index)
precision_score(test["Target"], preds)
```

Out[219…
```
0.51
```

In [208…
```
predictions = backtest(data3, model, predictors)
predictions["Predictions"].value_counts()
```

Out[208…
```
0.0    716
1.0     47
Name: Predictions, dtype: int64
```

In [209…
```
precision_score(predictions["Target"], predictions["Predictions"])
```

Out[209…
```
0.46808510638297873
```

In [210…
```
predictions["Target"].value_counts() / predictions.shape[0]
```

Out[210…
```
1    0.507208
0    0.492792
Name: Target, dtype: float64
```

If we just buy and sell at the end of the day, this would outperform our initial precision

In [211…
```
horizons = [2, 5, 60, 250]

new_predictors = []

for horizon in horizons:
    rolling_averages = data3.rolling(horizon).mean()

    ratio_column = f"Close_Ratio_{horizon}"
    data3[ratio_column] = data3["Adj Close"] / rolling_averages["Adj Close"]

    trend_column = f"Trend_{horizon}"
    data3[trend_column] = data3.shift(1).rolling(horizon).sum()["Target"]

    new_predictors += [ratio_column, trend_column]
```

In [212…
```
data3 = data3.dropna()
model = RandomForestClassifier(n_estimators = 200, min_samples_split = 50, random_state
```

In [213…
```
predictions = backtest(data3, model, new_predictors)
predictions["Predictions"].value_counts()
```

Out[213…
```
0.0    490
1.0     22
Name: Predictions, dtype: int64
```

In [214…
```
precision_score(predictions["Target"], predictions["Predictions"])
```

Out[214…
```
0.5909090909090909
```

We can see there is a great increase in accuracy up to 60%.

In [215…
```
data4 = data3.copy()
split = int((6/7)*len(data4))

X = pd.DataFrame(data4[["Close_Ratio_2", "Close_Ratio_5", "Close_Ratio_60", "Close_Rati
y = data4["Target"]


X_train, X_test, y_train, y_test = X[:split], X[split:], y[:split], y[split:]

model = LogisticRegression()
model = model.fit (X_train,y_train)

pd.DataFrame(zip(X.columns, np.transpose(model.coef_)))
```

Out[215...

	0	1
0	Close_Ratio_2	[-0.2849988247842387]
1	Close_Ratio_5	[-0.4790947637830209]
2	Close_Ratio_60	[0.09166326552560429]
3	Close_Ratio_250	[-0.027590708060639346]

In [216...

```
probability = pd.DataFrame(model.predict_proba(X_test))

predicted = model.predict(X_test)
probability
```

Out[216...

	0	1
0	0.491197	0.508803
1	0.496034	0.503966
2	0.500015	0.499985
3	0.500082	0.499918
4	0.494581	0.505419
...	...	...
211	0.488239	0.511761
212	0.490124	0.509876
213	0.494861	0.505139
214	0.501643	0.498357
215	0.501180	0.498820

216 rows × 2 columns

In [217...

```
print(metrics.confusion_matrix(y_test, predicted))
print(model.score(X_test,y_test))
```

```
[[27 72]
 [33 84]]
0.5138888888888888
```

Next Steps

If time permits, I could potentially add more parameters, including data on economic cycles and analyze inflation. On a more technical route, I could instead try to implement tick level data, since it may be more appropriate for an algorithm that is meant to help a frequent trading strategy.

In addition, I could try to implement other forms of machine learning models, including neural networks and SVM since they are also great for classification