report_src_file

October 21, 2022

1 DATA1030 Final Project - Henry Pasts

1.1 Can you create a profitable and feasible stock market trading strategy from a Machine Learning Pipeline?

The problem I want to solve is whether we can accurately predict the next day's price of the S&P 500 using current and historical data.

The end goal of the project is to build a stock trading simulation on the test data that will buy or sell (depending on current positioning) a stock market index at the next open.[1] It will use the data available up to the current point of evaluation to make predictions regarding the direction and magnitude of the next day's price.

1. [^](#cite_ref-1) This Project uses the S%P 500, which consists of the 500 largest U.S. Based Companies.

1.1.1 Target Variable:

I have 2 candidates for the target variable and plan to test them separately: 1. The Price of the next day (continuous) 2. The percent change of the next day's price (continuous)

Thus, this is a regression problem, as the values of the next day's price and magniture (percentage) are both continues features.

1.1.2 Importance and Interesting Aspects of the Project

The CMT Association[1] studies price trends and anylsis using a tool called "Technical Analysis"[2]. Mathematicians and other people interested in stocks have formulated indicators that take in a certain period (data from x days ago) of the last day's price, open, low, high, volume, etc, in order to track and hopefully predict future price trends and momentums. The CMT has been criticized for putting too much emphasis on the predictive ability of indicators, which many skeptics have questioned. I think indicators are valuable, but I also agree that more emphasis needs to be put on prediction and testing predicitve power, thus I plan to incorporate machine learning algoritms to find patterns and increase prediction confidence of the technical indicators and to better bound probabilities.

1.[^](#cite_ref-1) CMT Website

2.[^](#cite_ref-2) Description of Technical Analysis

1.1.3 The Data

Description of Features:

Original Features:

- Date: The date for the data (Datetime64 Object)
- Open: The Open Price (\$) for the date (float64)
- High: The High Price (\$) for the date (float64)
- Low: The Low Price (\$) for the date (float64)
- Close: The Close Price (\$) for the date (float64)
- Volume: The total dollar amount ($) traded for the date (int64)

Description of Added Features: I added standard tools from technical analysis that pull information from price. This work serves as 'feature engineering' to some extent but more research will be done on engineering features if model performance is not satisfactory. All feature 'windows', i.e. the lookback period were optimized by using the training data and finding the window values that best correlation to future price. Other features, such as the previous returns, are more relevant to percent change and may be dropped if better results come from predicting price and not the percent change of the next day's price.

There is a separate Jupyter Notebook titled "indicator_optimization" that runs this optimization.

Features:

Momentum Indicators:

- AwesomeIndicator: Measures market momentum by subtracting a moving average from the central bar points (high + low) / 2. Full Description (float64)
- KAMA: A moving average that accounts for volatility. Full Description (float64)
- PercentagePriceOscillator: Measures the difference between two moving averages. Full Description (float64)
- ROC: Calculates the rate of change of price Full Description (float64)
- RSI: Compares the magnitude of prior gains and losses to measure how fast price is changing Full Description (float64)
- StochRSI: Modified RSI that is sensitized. Full Description (float64)
- StochasticOscillator: Puts the last close in perspective to the high and lows over a period Full Description (float64)
- TSI: Measures trend direction and whether there has been increased buying pressure over the last x days. Full Description (float64)

Volume Indicators:

- AccDistIndicator: Measures price accumulation in accordance with volume. Full Description (float64)
- ChaikinMoneyFlow: Measures money flow into a security. Full Description (float64)
- EaseOfMovement: Compares price change with volume to assess trend Full Description (float64)
- ForceIndex: Measures how strong buying power is Full Description (float64)

- MFI: Uses price and volume to measure buying and selling pressures. Full Description (float64)
- NegativeVolumeIndex: Measures the amount of negative volume. Full Description (float64)
- OnBalanceVolume: It relates price and volume in the market and is based on total volume. Full Description (float64)
- VolumePriceTrend: Runs on cummulative volume and adds or subtracts volume depending on the return associated with the volume. Full Description (float64)
- VolumeWeightedAveragePrice: Weights Volume with Price Full Description (float64)

Volatility Indicators:

- Average True Range: Provides a degree of price volatility with ranges. Full Description (float64)
- Ulcer Index: Measures Price volatility as a measure of price depreciation from a high. Full Description (float64)

Trend Indicators:

- ADX: Measures the average direction of previous prices. Full Description (float64)
- Aroon: Identifies when trends are going to reverse by looking at days since a high price. Full Description (float64)
- CCI: Measures today's price change vs its average price change over a period Full Description (float64)
- DPO: Used to identify cycles in price Full Description (float64)
- EMA: Measures an average of price using weights on more important data. Full Description (float64)
- SMA: A Simple Moving Average of Price - Sum(all x prices) / x. Full Description (float64)
- WMA: Assigns more weight to more recent data. Full Description (float64)
- Prev Close Return 1 Day: The Close Return from 1 Day ago. (close - close_1_day_ago) / close_1_day_ago (float64)
- Prev Close Return 2 Days: The Close Return from 2 Days ago. (float64)
- Prev Close Return 3 Days: The Close Return from 3 Days ago. (float64)
- Month: Ordinal Feature Reflecting the Month, a number 1 through 12 (int64)

Target Variables:

- Percent Next Day: The Return of the Next Day (next_price - current_price) / current_price (float64)
- Price Next Day: The Price of the Next Day (float64)

The data was collected using Yahoo Finance's API yfinance[3] (pip install yfinance).

It was collected using the following code:

```
import pandas as pd
import numpy as np
import yfinance as yf
```

spy_ohlc_df = yf.download('SPY', start='1993-02-01', end='2022-09-30')

1.1.4 Other Relevant Research:

I see other research as a good way to see what I can expect my model to be able to accomplish.

Firuz Kamalov, Linda Smail, Ikhlaas Gurrib, 2021, used Deep Learning to predict the direction of the next day's price of the S&P 500 with a non-trivial accuracy of 56%. [a]

Alex Fuster and Zhichao Zou at Stanford University used ARIMA, Logistic Regression, Gaussian Discriminant Analysis (GDA), Support Vector Machines, and Neural Networks to predict the spread of a stock pair using a co-integration with S&P500 data and were able to predict the direction of spreads between individual stocks in the S&P 500 with an accuracy and precision of 57-58%, as well as a recall of 90%+. [b]

Other research not cited here seems to point to a 54-57% ability to predict the next day's price with mediocre precision and high recall. Getting the direction correct is crucial to avoid getting into a market that will be negative the next day. If the model gets the direction right, then the returns of the next day are a second layer of analysis. To start, I will derive the direction from the predicted magnitude of the next day or if the price is predicted higher or lower than the current day.

3.[^](#cite_ref-3) Yahoo!, Y!Finance, and Yahoo! finance are registered trademarks of Yahoo, Inc. yfinance is not affiliated, endorsed, or vetted by Yahoo, Inc. It's an open-source tool that uses Yahoo's publicly available APIs, and is intended for research and educational purposes. You should refer to Yahoo!'s terms of use (here, here, and here) for details on your rights to use the actual data downloaded. Remember - the Yahoo! finance API is intended for personal use only.

[1]:
```
# all imports
import pandas as pd
import numpy as np
import yfinance as yf

import matplotlib.pyplot as plt

from ta.momentum import AwesomeOscillatorIndicator, KAMAIndicator, \
    PercentagePriceOscillator, ROCIndicator, RSIIndicator, StochRSIIndicator, \
    StochasticOscillator, TSIIndicator

from ta.volume import AccDistIndexIndicator, ChaikinMoneyFlowIndicator, \
    EaseOfMovementIndicator, ForceIndexIndicator, MFIIndicator, \
     NegativeVolumeIndexIndicator, OnBalanceVolumeIndicator, \
        VolumePriceTrendIndicator, VolumeWeightedAveragePrice

from ta.volatility import AverageTrueRange, UlcerIndex

from ta.trend import ADXIndicator, AroonIndicator, CCIIndicator, DPOIndicator, \
    EMAIndicator, SMAIndicator, WMAIndicator

import matplotlib.dates as mdates
```

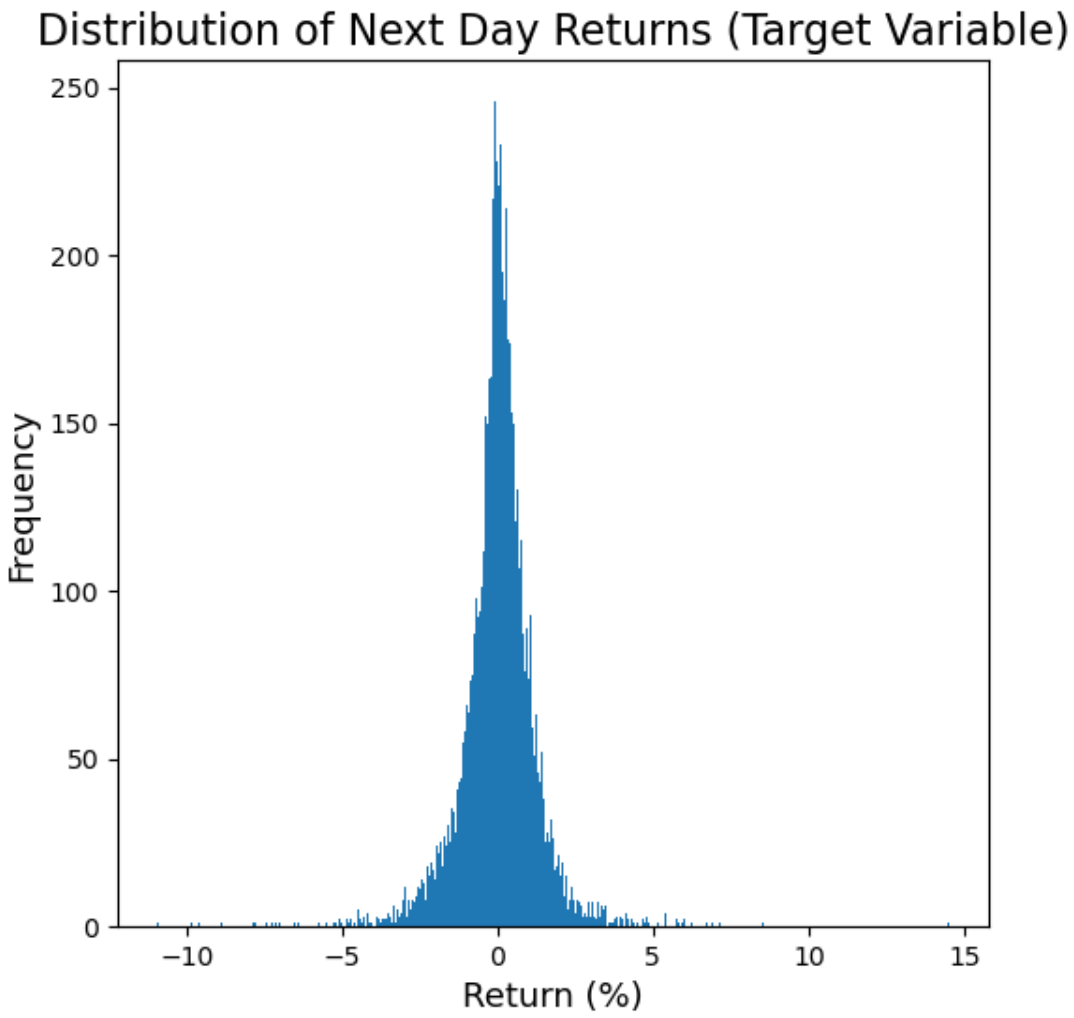
```
import matplotlib as mpl
import matplotlib.gridspec as gridspec

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler, MinMaxScaler, OrdinalEncoder
```

2 Exploratory Data Analysis

2.1 Visualize Target Variable

[5]:
```
spy_ohlc_df = pd.read_excel("../data/S&P500 Data.xlsx")

# print columns and rows:
print("\n")
print("There are " + str(spy_ohlc_df.shape[0]) + " Rows and " + str(spy_ohlc_df.
 ↪shape[1])
      + " Cols for SPY")
print("\n")
```

```
There are 7269 Rows and 39 Cols for SPY
```

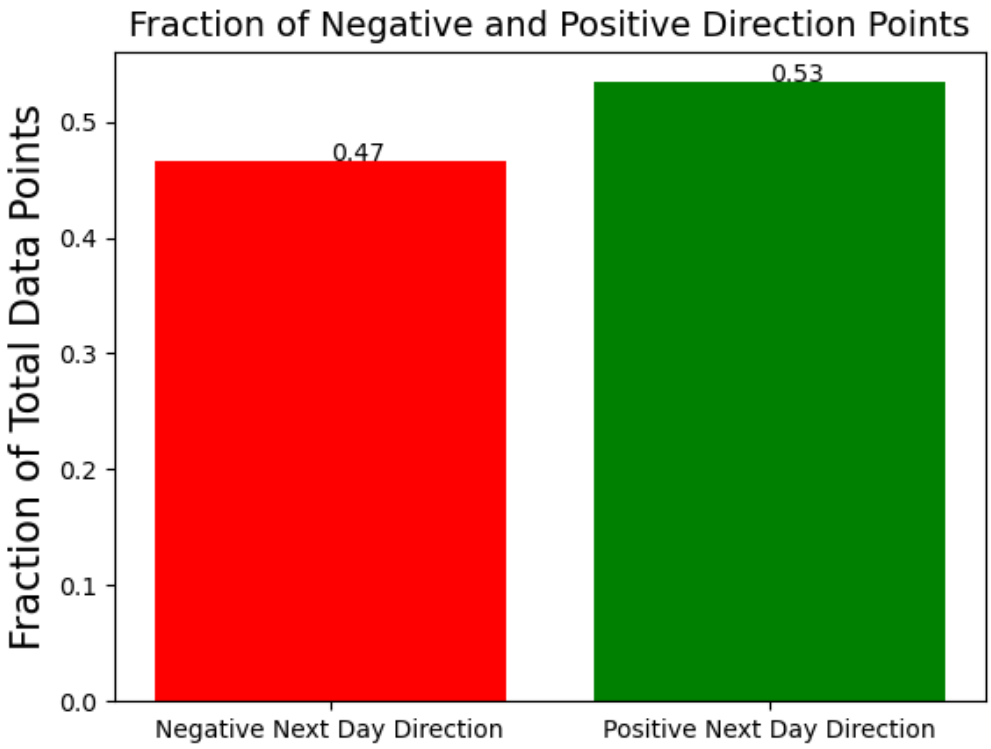


The next day returns of the price seem to be normally distributed with long tails in both directions. This is important information to know when predicting the next day's percentage change, as we know the general distribution the returns are coming from and know that it is centered around 0 with a 1.2% standard deviation.

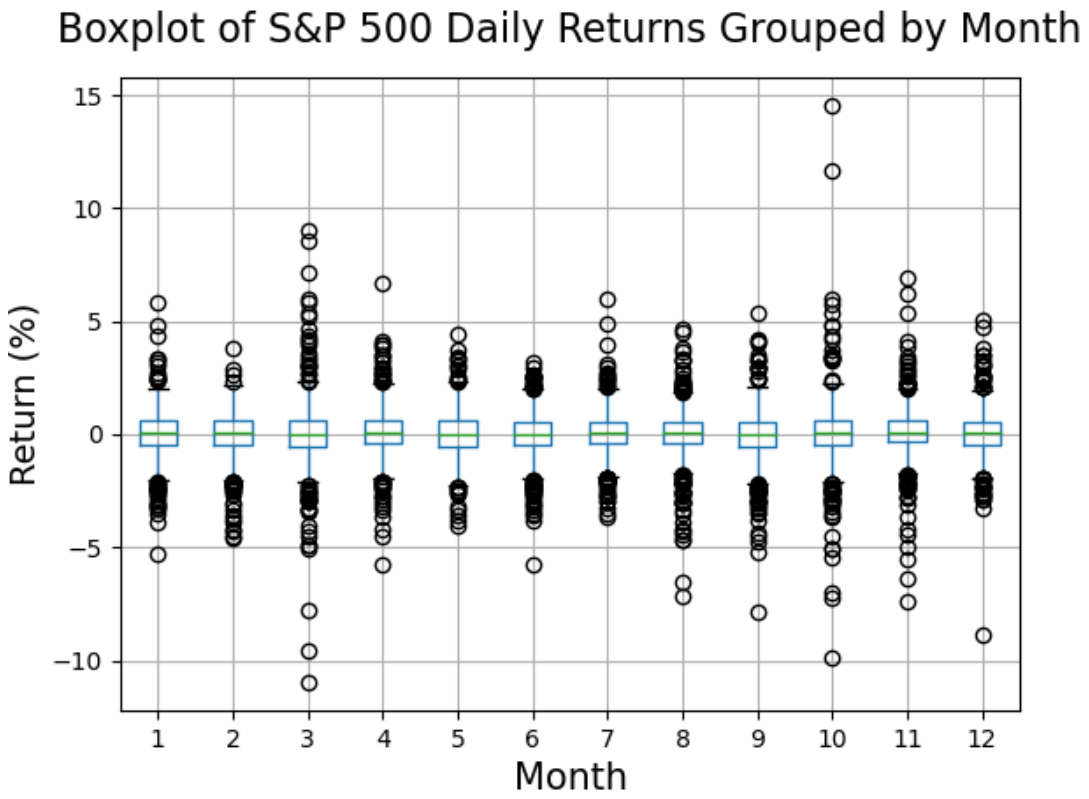
[16]:
```
spy_target_var_df = spy_ohlc_df[["Dates", "Close", "Percent Next Day", "Price␣
↪Next Day",
                                 "Prev Close Return 1 Day"]].copy()

print("Summary Stats For Next Day Returns:")
print("\n")
spy_target_var_df["Future Gain"] = spy_target_var_df["Percent Next Day"] * 100
print(spy_target_var_df["Future Gain"].describe())
```

```
Summary Stats For Next Day Returns:
```

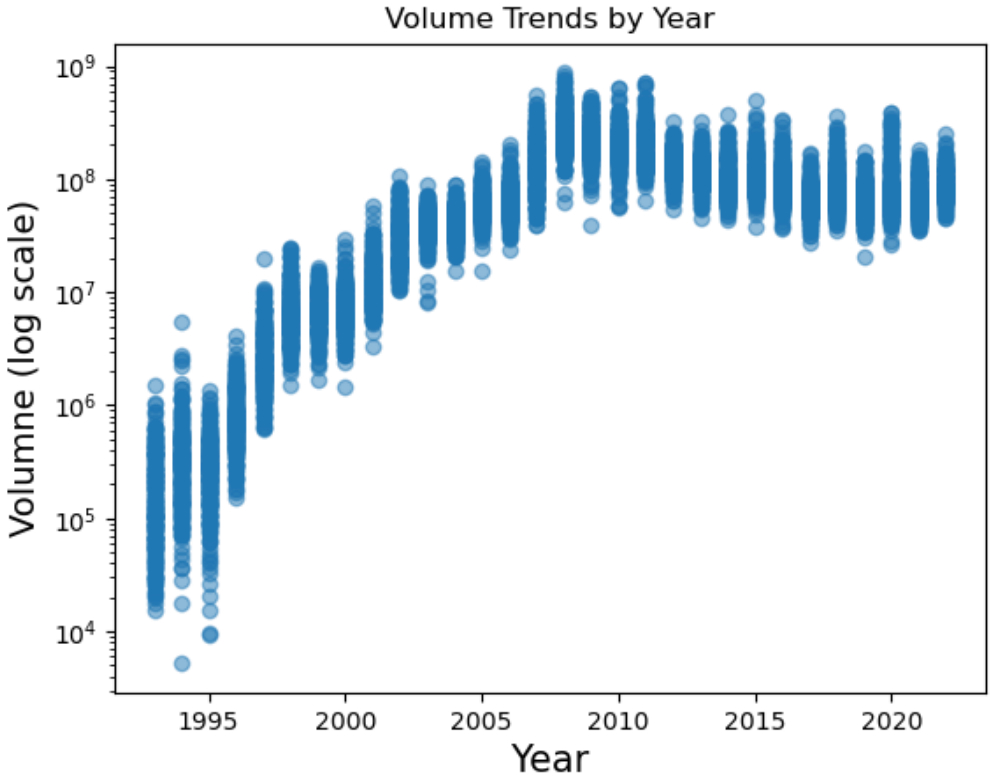
```
count    7269.000000
mean        0.035781
std         1.202915
min       -10.942374
25%        -0.469539
50%         0.063151
75%         0.598878
max        14.519772
Name: Future Gain, dtype: float64
```

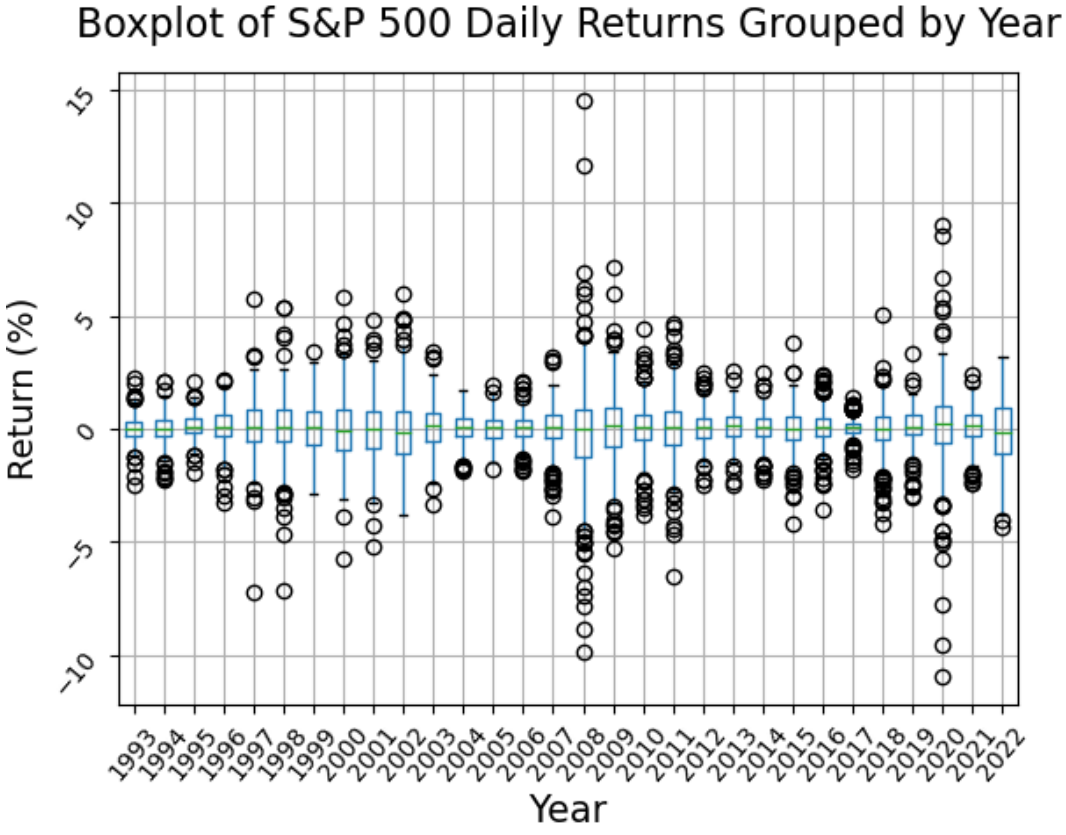


Roughly 50% of the target variable points are positive and 50% are negative, with a slight positive bias. This gives us more information on how hard it may be to predict the magnitude of the next day's price and whether that value is positive or negative and what we can expect a model outperformance of simply choosing positive may represent.



Returns by month vary and some months express heighted variance, thus we add the Month variable to the feature matrix to account for this and ordinally encode it. March and October look to have more tail-heavy events, which might help the model better gauge season effects


Volume Trends by Year
Volumne (log scale)
10^9
10^8
10^7
10^6
10^5
10^4
1995
2000
2005
2010
2015
2020
Year


Boxplot of S&P 500 Daily Returns Grouped by Year
Return (%)
15
10
5
0
−5
−10
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
Year

Initially, I thought that as volume increase, so would volatility, but volatility looks to be more correlated with the state of the economy, such as major economic events such as the Financial Crisis of 2008 or the Covid-19 Pandemic.

3 Data Preprocessing

My data is not iid, as previous prices are directly related and correlated to previous prices.

Since I am working with time-series data, I split my data with a 60-20-20 train-validation-test split with the earliest data in train.

I believe the splitting the data in the tradition time-series manner will be useful, given the forward predictions we are trying to make. It makes sense to see if we can create a model that works well on the price period from 1993 to 2010 and can generalize well to the validation and test data from 2010-2022. Additionally, since we have a major stock market event, namely the 2008 financial crisis in the training data and another major event in the test data, namely the 2020 Covid-19 pandemic, we have volatile events in both periods.

Splitting the data is important for deployment, as the model will take in today's prices and historical prices to predict tomorrow's price. Since tomorrow is unknown, treating time linearly by training on a period in the past and seeing if it generalizes well to test data that is close to the future is a good plan for the implementation aspects of the model.

[17]:
```
print("Does the DF contain any nan values?", spy_ohlc_df.isnull().values.any())
```

```
Does the DF contain any nan values? False
```

[18]:
```
# Splitting:
def basic_split(X,y,train_size,val_size,test_size,random_state):

    # test the inputs
    if train_size + val_size + test_size != 1 or type(random_state) != int:
        if train_size + val_size + test_size != 1 and type(random_state) == int:
            raise ValueError("Train Size + Val Size + Test Size must equal 1.")
        elif train_size + val_size + test_size == 1 and type(random_state) !=␣
↪int:
            raise ValueError("Random State must be an integer.")
        else:  # both tests fail:
            raise ValueError("Train Size + Val Size + Test Size must equal 1␣
↪and Random State must be an integer.")

    # perform basic split
    X_train, X_other, y_train, y_other = train_test_split(X, y,␣
↪train_size=train_size,
                                                          ␣
↪test_size=(val_size+test_size), random_state=random_state)
    X_val, X_test, y_val, y_test = train_test_split(X_other, y_other,␣
↪train_size=(val_size/(val_size + test_size)),
```

```
                                                    test_size=(test_size/
 ↪(val_size + test_size)),
                                                    random_state=random_state)

    return X_train, y_train, X_val, y_val, X_test, y_test
```

[20]:
```
# Month Ordinal Encoding:
enc = OrdinalEncoder()
spy_ohlc_df["Month"] = enc.fit_transform(spy_ohlc_df["Month"].to_numpy().
 ↪reshape(-1, 1))

# note: the above line is done here because graphs require Date to be DateTime

# 60-20-20 (train-val-test) for Price:
y_price = spy_ohlc_df["Price Next Day"].to_numpy()
X_price = spy_ohlc_df.drop(labels=["Percent Next Day", "Price Next Day"],␣
 ↪axis=1)
X_train_price, y_train_price, X_val_price, y_val_price, X_test_price,␣
 ↪y_test_price = \
    basic_split(X_price, y_price, 0.6, 0.2, 0.2, 7)
print("X train price Len:", len(X_train_price))
print("Y train price Len:", len(y_train_price))

print("X val price Len:", len(X_val_price))
print("Y val price Len:", len(y_val_price))

print("X test price Len:", len(X_test_price))
print("Y test price Len:", len(y_test_price))

print("\n")

# 60-20-20 (train-val-test) for Percent Change:
y_pct = spy_ohlc_df["Percent Next Day"].to_numpy()
X_pct = spy_ohlc_df.drop(labels=["Percent Next Day", "Price Next Day"], axis=1)
X_train_pct, y_train_pct, X_val_pct, y_val_pct, X_test_pct, y_test_pct = \
    basic_split(X_pct, y_pct, 0.6, 0.2, 0.2, 7)
print("X train pct Len:", len(X_train_pct))
print("Y train pct Len:", len(y_train_pct))

print("X val pct Len:", len(X_val_pct))
print("Y val pct Len:", len(y_val_pct))

print("X test pct Len:", len(X_test_pct))
print("Y test pct Len:", len(y_test_pct))
```

```
X train price Len: 4361
Y train price Len: 4361
```

```
X val price Len: 1454
Y val price Len: 1454
X test price Len: 1454
Y test price Len: 1454


X train pct Len: 4361
Y train pct Len: 4361
X val pct Len: 1454
Y val pct Len: 1454
X test pct Len: 1454
Y test pct Len: 1454
```

Now, I will apply a StandardScaler to all continuous columns as well as a separate DataFrame that uses MinMaxScaler and I will compare both in my modeling process. It makes sense to normalize variables related to percent change with a StandardScaler due to the mean and standard deviation normalization. In order to creata a more stable approximation of the next day's price, a MinMaxScaler may be appropriate, so testing both depending on the target variable (next day's price or next day's percent change) is needed.

I apply an OrdinalEncoder to the Month variable to convert it to a value (1 to 12).

For now, I drop the Date variable, which is of type 'DateTime', but will experiment with including it instead of the month variable as a timestamp number.

[21]:
```
# Apply Scalers:

# StandardScaler on data with target variable of 'Price':
standard_scaler_X = StandardScaler()
standard_scaler_X.fit(X_train_price)

X_train_price = standard_scaler_X.transform(X_train_price)
X_val_price = standard_scaler_X.transform(X_val_price)
X_test_price = standard_scaler_X.transform(X_test_price)

# reshape y:
y_train_price = y_train_price.reshape(-1, 1)
y_val_price = y_val_price.reshape(-1, 1)
y_test_price = y_test_price.reshape(-1, 1)

standard_scaler_y = StandardScaler()
standard_scaler_y.fit(y_train_price)

y_train_price = standard_scaler_y.transform(y_train_price)
y_val_price = standard_scaler_y.transform(y_val_price)
y_test_price = standard_scaler_y.transform(y_test_price)


# MinMaxScaler on data with target variable of 'Percent Change':
```

```
min_max_scaler_X = MinMaxScaler()
min_max_scaler_X.fit(X_train_pct)

X_train_pct = min_max_scaler_X.transform(X_train_pct)
X_val_pct = min_max_scaler_X.transform(X_val_pct)
X_test_pct = min_max_scaler_X.transform(X_test_pct)

# reshape y:
y_train_pct = y_train_pct.reshape(-1, 1)
y_val_pct = y_val_pct.reshape(-1, 1)
y_test_pct = y_test_pct.reshape(-1, 1)

min_max_scaler_y = MinMaxScaler()
min_max_scaler_y .fit(y_train_pct)

y_train_pct = min_max_scaler_y.transform(y_train_pct)
y_val_pct = min_max_scaler_y.transform(y_val_pct)
y_test_pct = min_max_scaler_y.transform(y_test_pct)

print("Success")
```

```
Success
```

[25]:
```
# print columns and rows after Preprocessing:
print("\n")
print("There are " + str(spy_ohlc_df.shape[1]) + " Preprocessed Features in the␣
 ↪Data")
print("\n")
```

```


There are 39 Preprocessed Features in the Data
```

4 Github Repo

https://github.com/henrypasts/data1030_project.git

5 References

a.[^](#cite_ref-4) Kamaloc, Firuz, Smail, Linda, Gurrib, Ikhlaas (2021). FORECASTING WITH DEEP LEARNING: S&P 500 INDEX

b.[^](#cite_ref-5) Fuster, Alex, Zou, Zhichao (2018). Using Machine Learning Models to Predict S&P500 Price Level and Spread Direction