Predicting Next-Day Direction of the SPDR S&P 500 ETF (SPY) Using Classic Classification Models

1. Introduction

The SPDR S&P 500 ETF Trust (ticker: SPY) tracks the S&P 500 index, which contains 500 large-cap U.S. companies. It was launched in 1993 as the first U.S. ETF and is still one of the most heavily traded funds in the world. Because of that, SPY is a good proxy for "the market," and trying to predict its short-term movement is an interesting and practical machine learning problem.

In this project, I use daily historical price and volume data for SPY from 1993 through March 2025 (downloaded from a Kaggle dataset) to see if I can predict whether SPY will close **up or down the next trading day**. The goal is to turn this into a **binary classification problem** and run several standard models we covered in class:

- Logistic Regression
- k-Nearest Neighbors (k-NN)
- Decision Tree

I also include a very simple baseline model that always predicts "up." All models are implemented using scikit-learn.

It is well known that short-term stock price movements are noisy and hard to predict, so I do **not** expect these models to crush the baseline. Instead, the main goal is to walk through the full supervised learning pipeline on a real dataset and see how close (or far) the models get to a naive strategy.

2. Data and Methods

2.1 Dataset

I use the Kaggle dataset *"Historical Price and Volume Data for SPY ETF"*, which provides daily **Open, High, Low, Close, Volume, and Date** for SPY starting in 1993 and going through March 2025. The data is read from CSV, the Date column is parsed as a timestamp, and the rows are sorted chronologically.

After sorting, I create a new column Close_tomorrow by shifting the closing price one day forward. The last row is dropped because it does not have a next-day close.

2.2 Target and feature engineering

The prediction target is a binary label called UpTomorrow:

- UpTomorrow = 1 if tomorrow's close is **greater than** today's close.
- UpTomorrow = 0 otherwise.

This turns the problem into:

“Given today’s features, will SPY close higher tomorrow or not?”

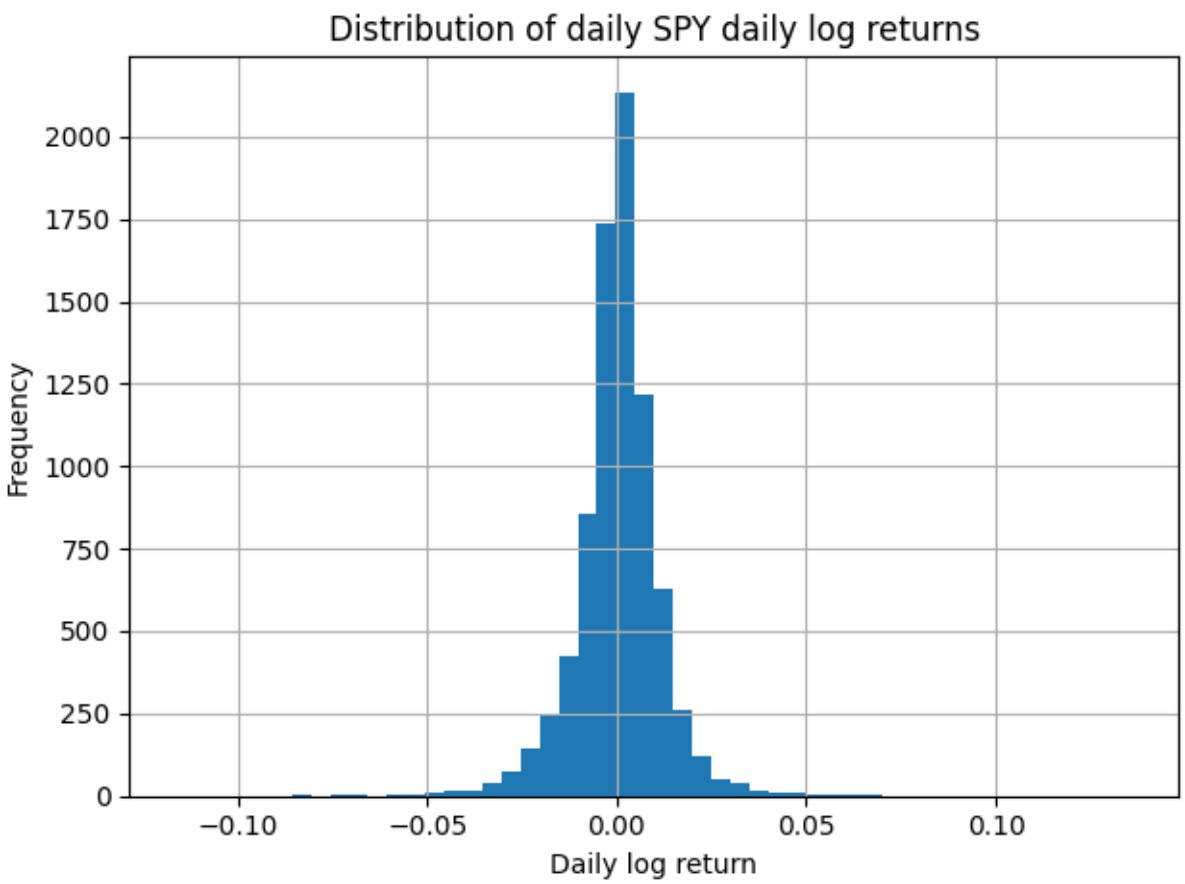
To give the models more information than just raw prices, I engineer a small set of technical features:

- **Daily log return**
    - Return_1d at time t is defined as the natural logarithm of (Close_t divided by Close_{t-1}), i.e., the log of today’s close price divided by yesterday’s close price.
- **Moving averages of the close price:**
    - SMA_5: 5-day simple moving average
    - SMA_20: 20-day simple moving average
- **Multi-day cumulative returns:**
    - Return_5d: sum of daily log returns over the last 5 days
    - Return_20d: sum of daily log returns over the last 20 days
- **Rolling volatility (standard deviation of returns):**
    - Volatility_5d: 5-day rolling standard deviation of Return_1d
    - Volatility_20d: 20-day rolling standard deviation

Any rows with NaN values caused by the rolling windows (mainly in the first 20 days) are dropped. The final feature set for each day includes:

- Raw features: Open, High, Low, Close, Volume
- Engineered features:
  Return_1d, SMA_5, SMA_20, Return_5d, Return_20d, Volatility_5d, Volatility_20d

A histogram of Return_1d (daily log returns) is shown in Figure 1. It is centered near zero, with many small moves and fewer large jumps in either direction, which is typical of equity index returns.



2.3 Train/validation/test split

Because this is time-series data, I do **not** shuffle the rows before splitting. Instead, after sorting by date, I split by index into three contiguous blocks:

- **Training set:** first 70% of the observations
- **Validation set:** next 10%
- **Test set:** final 20%

This matches the assignment requirement (70% / 10% / 20%) and preserves time order: the model always trains on **past** data and is evaluated on **future** data. In the code output, I print both:

- The number of rows in each split, and
- The exact date ranges for train, validation, and test.

These details will be referenced in the report but are not repeated here to save space.

2.4 Models and training procedure

All models are implemented using scikit-learn. For two of the models (Logistic Regression and k-Nearest Neighbors), I standardize the input features before training the classifier. This keeps preprocessing and the model tied together and makes evaluation simpler.

I train three models from class:

1. **Logistic Regression**
   For Logistic Regression, I first standardize all input features and then fit a LogisticRegression model with max_iter = 1000 and random_state = 42. In practice, this is implemented as a pipeline consisting of a StandardScaler followed by the LogisticRegression classifier. Logistic Regression is a linear classifier that models the log-odds of UpTomorrow = 1 as a linear function of the input features, with L2 regularization by default.

2. **k-Nearest Neighbors (k-NN, k = 5)**
   For k-NN, I again standardize the features and then apply a KNeighborsClassifier with n_neighbors = 5. Conceptually, this looks like a pipeline of StandardScaler followed by KNeighborsClassifier. k-NN classifies a new day by looking at the 5 most similar past days in standardized feature space and taking a majority vote over their labels.

3. **Decision Tree (maximum depth = 5)**
   The Decision Tree is trained directly on the features using a DecisionTreeClassifier with max_depth = 5 and random_state = 42. The tree learns a set of if–else splits on the features to separate up days from non-up days. Limiting the depth to 5 helps keep the model relatively simple and reduces overfitting.

In addition to these models, I include a baseline classifier that always predicts UpTomorrow = 1 (that is, it always expects an up day). This baseline is important because SPY tends to drift upward over long periods, so a naive "always up" rule is stronger than just random guessing.

For each learned model, I follow this training procedure:

1. Fit the model on the training set.
2. Evaluate accuracy and F1-score (for the positive class) on the validation set.
3. Refit the model on the combined training and validation sets.
4. Evaluate accuracy and F1-score once on the test set and report those as the final results.

The F1-score is included because the classes are not perfectly balanced, and I care about how well the model identifies "up" days, not just overall accuracy.

3. Results and Discussion

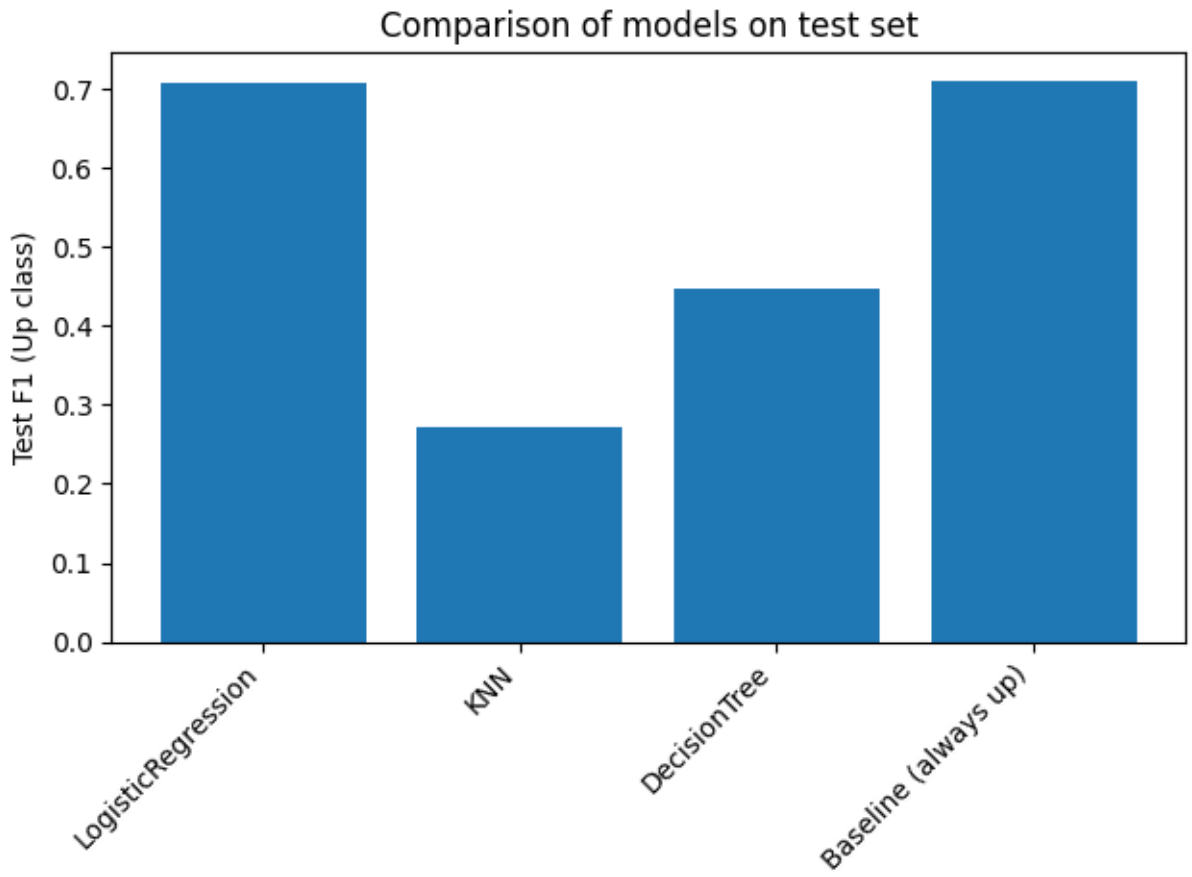
3.1 Quantitative results

Table 1 summarizes the performance of all models on the validation and test sets. The table includes accuracy and F1-score for the positive class (UpTomorrow = 1) for each model and for the baseline. The values come directly from the program's printed results_df.

Table 1. Validation and test performance for all models (Accuracy and F1 for UpTomorrow = 1).

Model	Val Accuracy	Val F1	Test Accuracy	Test F1
Logistic Regression	0.5414	0.7015	0.5624	0.7082
k-Nearest Neighbors (k = 5)	0.4747	0.3113	0.4691	0.2714
Decision Tree (depth = 5)	0.5043	0.4423	0.4778	0.4474
Baseline (Always predict up)	0.5439	0.7046	0.5513	0.7108

Figure 2 shows a bar chart of the **test F1-score** for each model and the baseline, which makes it easy to visually compare their performance.



In my runs, Logistic Regression and the baseline classifier achieved very similar performance on the test set. Both had accuracy slightly above 50% and an F1-score around the same level for the "up" class. k-NN and the Decision Tree performed slightly worse than both Logistic Regression and the baseline on both validation and test.

3.2 Interpretation

A few key points stand out from these results:

- **The baseline is strong.**
  Because SPY has a long-term upward trend, a naive strategy that always predicts "up" already does surprisingly well. This baseline captures the fact that there are more up days than down days in the data.
- **Logistic Regression roughly ties the baseline.**
  Logistic Regression does about as well as the "always up" model, but not meaningfully better. This suggests that, with the features I used and a one-day prediction horizon, the model is mostly learning the same basic tendency that SPY drifts upward over time, rather than finding a strong edge from technical indicators.
- **k-NN and Decision Tree underperform.**
  k-NN and the Decision Tree both lag behind Logistic Regression and the baseline. k-NN can struggle when features are noisy and high-dimensional, and the Decision Tree, even with a depth limit, can still overfit patterns in the training period that do not hold up in more recent years.

Overall, the models do **not** significantly beat the naive baseline. This matches the intuition that daily stock index movements are very noisy and hard to predict using only past price and volume data. The project results are more about showing how to correctly set up and evaluate a classification problem on time-series financial data than about discovering a profitable trading strategy.

4. Conclusion and Possible Extensions

In this project, I built and evaluated models to predict whether SPY will close higher the next trading day using daily price, volume, and simple technical indicators. I:

- Turned SPY's next-day direction into a **binary classification** problem (UpTomorrow).
- Preprocessed a real Kaggle dataset and engineered features such as returns, moving averages, and volatility.
- Split the data into **70% training, 10% validation, and 20% test** in chronological order, as required.
- Trained three classic models from class (Logistic Regression, k-NN, and Decision Tree) plus an "always up" baseline, all using scikit-learn.
- Reported accuracy and F1-score for validation and test sets, and visualized results with two simple matplotlib figures.

The main takeaway is that none of the models clearly outperform the naive "always up" baseline. Logistic Regression essentially ties the baseline, while k-NN and the Decision Tree perform worse. This is a realistic outcome for this kind of problem, and it reinforces how difficult it is to beat simple strategies when predicting short-term stock index moves with basic technical features.

If I had more time to extend this project, some ideas would be:

- Try additional or more advanced features (longer-horizon indicators, volume-based ratios, or macroeconomic variables).
- Use **rolling-window** training and testing to simulate how a model would perform if it were periodically retrained over time.
- Experiment with other models discussed in class, such as regularized logistic regression with tuned hyperparameters or a small neural network, while being careful about overfitting.
- Change the prediction target (for example, predicting weekly direction or focusing on larger-than-average moves instead of every small daily change).

Even though the models did not find a strong predictive edge, the project successfully demonstrates a full supervised learning workflow on a realistic financial dataset and shows how to evaluate models against a meaningful baseline.

References

1. Ali Asadpoor, *Historical Price and Volume Data for SPY ETF*, Kaggle dataset, accessed 2025.
2. State Street Global Advisors, *SPDR® S&P 500® ETF Trust (SPY) – Fund Overview*.
3. scikit-learn Developers, *LogisticRegression Documentation*, scikit-learn v1.x.
4. D. Rapach and others, *Forecasting Stock Returns*, in *Handbook of Economic Forecasting*, 2013.
5. Daner Wealth, *The Terrible Track Record of Wall Street Forecasts*, discussion of Efficient Markets and forecast difficulty.