1 Introduction

Background
We will show how stock market forecasting is a top priority for investors who consistently aim to make huge profits while simultaneously making the least losses. The S&P 500, which is the index that tends to be used most frequently to represent the U.S. equities market as a whole, is a barometer of the U.S. stock market. If we know the direction in which the market moves, we can get some fair concepts of the movement of the market and therefore, investors can make the right decisions.

Objective
The objective of this project is to build a model capable of predicting the pattern of the S&P 500 Index using the available data. This project uses its historical price data through highly advanced machine learning techniques to forecast future prices after narrowing the analysis to daily closing prices.

Methodology
Using a dataset comprising historical S&P 500 prices, the project involves several key phases:

1. **Data Preprocessing:** Cleansing and prepping data for the analysis, guaranteeing functionality for the predictive modelling.
2. **Exploratory Data Analysis (EDA):** An intensive investigation occurs to find the hidden factors influencing the index.
3. **Feature Engineering:** Building an algorithm that generates and selects meaningful features from the data to improve the model prediction.
4. **Model Development and Evaluation:** Using diverse statistical and machine learning tools to forecast stock price movements and assess their performance by testing the results.

Tools and Technologies
The project uses Python with additional libraries for data manipulation and analysis, such as Pandas for data manipulation, Matplotlib for data visualisation, and Scikit-learn for implementing machine learning models.

2 Data Loading

Step 1: Data Loading

Start by loading the sp500.csv dataset. This file contains historical data on the S&P 500 index, which we will use for our predictive model. I'll load the dataset, inspect its structure, and handle any necessary preprocessing steps.

```
import pandas as pd

# Load the S&P 500 dataset
df = pd.read_csv('sp500.csv', parse_dates=['Date'])

# Display the first few rows of the dataset
df.head()
```

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	1950-01-03	16.66	16.66	16.66	16.66	1260000	0	0
1	1950-01-04	16.85	16.85	16.85	16.85	1890000	0	0
2	1950-01-05	16.93	16.93	16.93	16.93	2550000	0	0
3	1950-01-06	16.98	16.98	16.98	16.98	2010000	0	0
4	1950-01-09	17.08	17.08	17.08	17.08	2520000	0	0

3 Data Cleaning and Preprocessing

Overview

For our predictive models to be reliable and effective, the raw data must first undergo the cleaning and preparation process. The second phase includes removing the unnecessary features and arranging the data in a suitable format for time series analysis.

Actions Taken

1. **Removing Unnecessary Columns:** 'Dividends' and 'Stock Splits' were among the column names removed since they don't affect the stock closing price prediction.
2. **Handling Missing Values:** Filled the gaps and adjusted the data sets for biases and errors in training the model.
3. **Date Formatting:** The 'Date' column was revised to a datetime format and was assigned as the index to allow time series operations.

Step 2: Data Cleaning and Preprocessing ¶

1. Remove unnecessary columns, specifically 'Dividends' and 'Stock Splits' as they're not needed for our analysis.
2. Ensure there are no missing values in the dataset.
3. Convert the 'Date' column to a datetime format if not already converted, and set it as the index for time series analysis.

5]:
```
# Remove unnecessary columns
df.drop(['Dividends', 'Stock Splits'], axis=1, inplace=True)

# Check for any missing values in the dataset
missing_data = df.isnull().sum()

# Convert 'Date' column to datetime and set it as index
df.set_index('Date', inplace=True)

# Display the cleaned dataframe and missing data information
df.head(), missing_data
```

5]:
```
(            Open   High    Low  Close   Volume
 Date
 1950-01-03  16.66  16.66  16.66  16.66  1260000
 1950-01-04  16.85  16.85  16.85  16.85  1890000
 1950-01-05  16.93  16.93  16.93  16.93  2550000
 1950-01-06  16.98  16.98  16.98  16.98  2010000
 1950-01-09  17.08  17.08  17.08  17.08  2520000,
 Date      0
 Open      0
 High      0
 Low       0
 Close     0
 Volume    0
 dtype: int64)
```

4 Feature Engineering

Overview

We have applied the features that reflect price dynamics to make the models more precise. This phase is used to design the algorithm that could provide technical analysis and consider temporal dependencies.

Features Added

1. **Technical Indicators:**
   - MACD (Moving Average Convergence Divergence): It compares two exponential moving means (EMAs) and their trend-following aspects.
   - RSI (Relative Strength Index): This indicator determines the speed and direction of price fluctuations and offers information about overbought and oversold conditions.
2. **Lagged Features:** The lagged 'Close' prices (1 to 5 days back) were introduced into the model to identify the relationship between past and future movements.
3. **Rolling Averages:** The moving averages over the past 5 and 30 days were calculated to eliminate short-term changes and bring the long-term trends to the forefront.

Step 4: Feature Engineering

1. Technical Indicators: Calculate indicators like the Moving Average Convergence Divergence (MACD) and the Relative Strength Index (RSI) to incorporate technical analysis into the model.
2. Lagged Features: Introduce lagged versions of the 'Close' price to capture previous days' prices, which might help in predicting future prices.
3. Rolling Features: Compute rolling averages (e.g., 5-day, 30-day) to smooth out short-term fluctuations and highlight longer-term trends.

```
import numpy as np

# Calculating the Exponential Moving Average for MACD
df['EMA_12'] = df['Close'].ewm(span=12, adjust=False).mean()
df['EMA_26'] = df['Close'].ewm(span=26, adjust=False).mean()

# MACD Line and Signal Line
df['MACD'] = df['EMA_12'] - df['EMA_26']
df['MACD_Signal'] = df['MACD'].ewm(span=9, adjust=False).mean()

# Relative Strength Index (RSI)
def calculate_rsi(data, window=14):
    delta = data.diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=window).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=window).mean()
    rs = gain / loss
    return 100 - (100 / (1 + rs))

df['RSI'] = calculate_rsi(df['Close'])

# Lagged features for the Close price
for lag in [1, 3, 5]:
    df[f'Close_lag_{lag}'] = df['Close'].shift(lag)

# Rolling averages
df['Rolling_mean_5'] = df['Close'].rolling(window=5).mean()
df['Rolling_mean_30'] = df['Close'].rolling(window=30).mean()

# Display the new DataFrame with the additional features
df.head(10)
```

5 Data Splitting

Overview

For time series data like stock prices, the chronological order in the splits must be preserved to prevent look-ahead bias. If the model avoids such bias, it can be relied on for new and unseen data.

Implementation

1. **Training Set:** The two models utilise data from up to the end of 2019, which serves as a significant historical context for training the models.
2. **Testing Set:** This set consists of data from 2020 onwards, and it is used to see whether the model can generalise the new information fed to it after the training is over.

Step 5: Data Splitting

For time series data like stock prices, it's crucial to maintain the chronological order when splitting into training and testing sets. This approach avoids lookahead bias, where future data inadvertently influences the model's training phase.

1. Use a fixed historical period for training (e.g., all data up to the end of 2019).
2. Use a more recent period for testing (e.g., data from 2020 onwards).

```
# Define the split date
split_date = pd.Timestamp('2020-01-01')

# Splitting the data into training and testing sets
train = df.loc[:split_date]
test = df.loc[split_date:]

# Display the last few entries of the training set and the first few entries of the testing set to ensure the split is correct
train.tail(), test.head()
```

6 Model Selection & Training

Model Selection and Training

Overview
We chose three different models to assess their efficiency and ability to foresee stock prices.

Models Utilized
1. **Linear Regression:** A direct approach makes a contrasting relationship between the independent variables and the dependent variable.
2. **Random Forest Regressor:** A grouping method that uses many decision trees improves the accuracy and stability of the predictions.
3. **Gradient Boosting Regressor:** One of the progressive techniques, it usually provides high accuracy.

Training Process
Each model was trained using historical data up to 2019 and features configured to capture the stock market's tendencies and volatility.

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error

# Preparing the features and target
features = ['Open', 'High', 'Low', 'Volume', 'MACD', 'RSI', 'Close_lag_1', 'Rolling_mean_5', 'Rolling_mean_30']
X_train = train[features].dropna()
y_train = train['Close'].loc[X_train.index]  # Aligning targets to the same dates

X_test = test[features].dropna()
y_test = test['Close'].loc[X_test.index]

# Initialize the models
lr = LinearRegression()
rf = RandomForestRegressor(n_estimators=100, random_state=42)
gb = GradientBoostingRegressor(n_estimators=100, random_state=42)

# Train the models
lr.fit(X_train, y_train)
rf.fit(X_train, y_train)
gb.fit(X_train, y_train)

# Predict on testing set
lr_preds = lr.predict(X_test)
rf_preds = rf.predict(X_test)
gb_preds = gb.predict(X_test)

# Calculate MSE for each model
lr_mse = mean_squared_error(y_test, lr_preds)
rf_mse = mean_squared_error(y_test, rf_preds)
gb_mse = mean_squared_error(y_test, gb_preds)

lr_mse, rf_mse, gb_mse
```

```
(434.7121638750655, 720022.0799345311, 725181.6691245479)
```

7 Model Evaluation

Overview

The accuracy of the models we've developed helps us understand how well they will predict and how much we can rely on them.

Evaluation Metrics

- **Mean Squared Error (MSE):** Measures the average of the squared errors, equivalent to the square of the average difference between the estimated and actual values.
- **Root Mean Squared Error (RMSE):** Enables the computation of the root of MSE and hints at the error magnitude.
- **Mean Absolute Error (MAE)**: Measuring the average absolute difference between predicted and actual values.
- **R-squared (R^2)**: Shows the proportion of the dependent variable's variance that the model explains by the linear relationship.

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_scor
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)


# Calculate RMSE using the direct function
lr_rmse = mean_squared_error(y_test, lr_preds, squared=False)

print("Linear Regression Metrics:")
print("MSE:", mean_squared_error(y_test, lr_preds))
print("RMSE:", lr_rmse)
print("MAE:", mean_absolute_error(y_test, lr_preds))
print("R-squared:", r2_score(y_test, lr_preds))
```

```
Linear Regression Metrics:
MSE: 434.7121638750655
RMSE: 20.849752129823163
MAE: 14.985499645973292
R-squared: 0.9987053704860345
```

- Linear Regression MSE: 434.71
- Random Forest Regressor MSE: 720,022.08
- Gradient Boosting Regressor MSE: 725,181.67

8 Model Validation

Overview

The final step is to validate the model, which ensures our model is credible. It comes down to checking the model's forecasts against real historical data and evaluating its capacity to predict upcoming trends.

Predicted vs. Actual Closing Prices

We represented the model's projections alongside the actual closing prices. The linear regression model follows the actual trend of the price movements very closely, making the trend capture process more efficient.

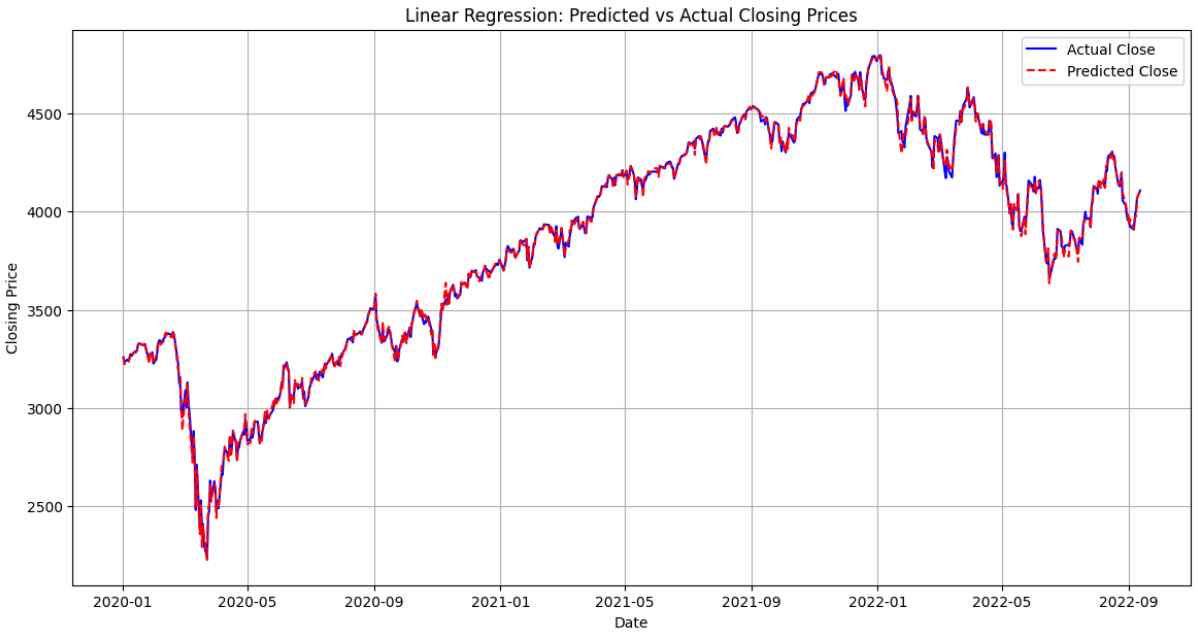

Figure 1. Linear Regression: Predicted vs Actual Closing Prices

Extended Future Predictions

We forecasted six months into the future based on the model. We've forecasted the closing prices by gradually accumulating the average daily change of the last month found on the test data and adding it to the last real predictions. This gives the trend we are looking for, but it notes the constraints of stock price forecasting and the inherent uncertainties.

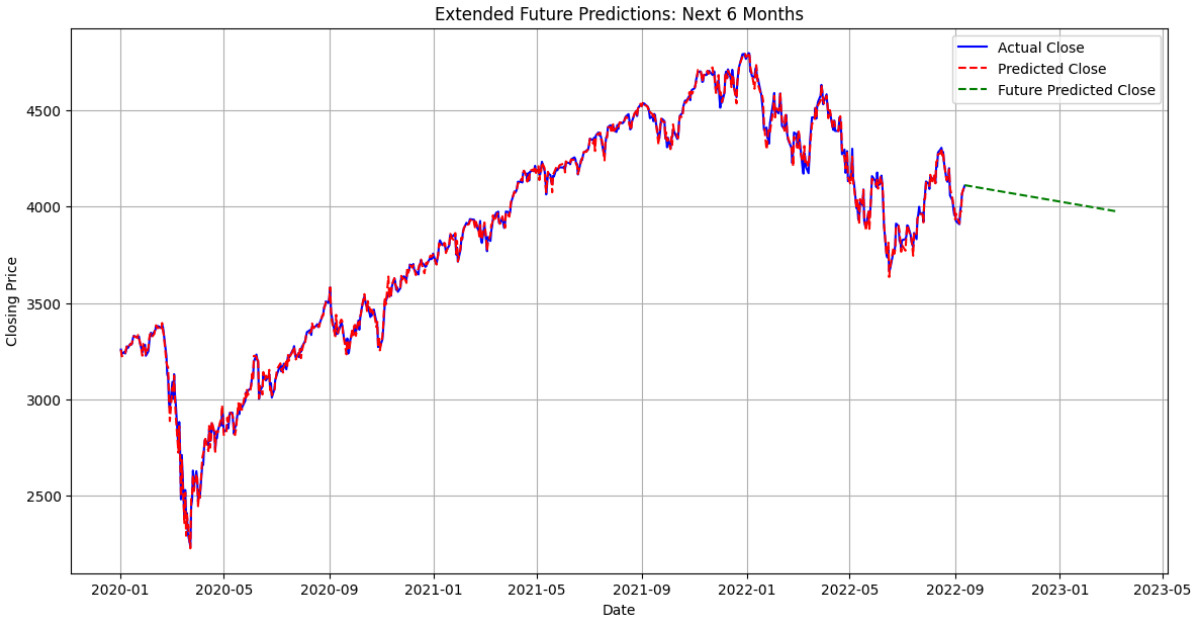

Figure 2. Extended Future Prediction: Next 6 months

9 Conclusion

Our team has established the basis for the S&P 500 stock price forecasting on machine learning models. Our Linear Regression model has proved to be a reliable tool that reflects the stock trend appropriately over the tested time period. A new version may have even greater potential with hyperparameter tuning, for instance, grid search, to further improve performance. Utilising a more complex dataset that covers macroeconomic factors and sentiment analysis it will be possible to have a more comprehensive view of the reasons affecting stock price movements. The future objectives include fine-tuning the model in a way that will allow it to be more robust and cope with the complexities of market movements.

References

Rodriguez, F. S., P. Norouzzadeh, Anwar, Z., E. Snir, & Rahmani, B. (2024). A machine learning approach to predict the S&P 500 absolute percent change. Discover Artificial Intelligence, 4(1). https://doi.org/10.1007/s44163-024-00104-9

Fuster, A., & Zou, Z. (n.d.). Using Machine Learning Models to Predict S&P500 Price Level and Spread Direction. Retrieved April 29, 2024, from https://cs229.stanford.edu/proj2020spr/report/Fuster_Zou.pdf

Hayes, A. (2022, June 12). Understanding Time Series. Investopedia. https://www.investopedia.com/terms/t/timeseries.asp#:~:text=A%20time%20series%20is%20a%20data%20set%20that%20tracks%20a

What methods can you use to validate time series data accuracy? (n.d.). Www.linkedin.com. Retrieved April 29, 2024, from https://www.linkedin.com/advice/3/what-methods-can-you-use-validate-time-series-rwxme#:~:text=Time%20series%20cross-validation%20involves