QQQ Stock Price Prediction

Rachel Kim 300394050

Ki

01

Data Set & Loading

Data Set

QQQ is an ETF that tracks the **NASDAQ-100**, investing in major tech stocks like **Apple** and **Tesla**.

With over 50% in tech, it has delivered a long-term average return of **15%** per year, but it is highly volatile and sensitive to **interest rate changes** and **macroeconomic conditions**.

Data Loading

Download market data from **Yahoo! Finance API**

```
## Stock dataset install
import yfinance as yf

# download using yhfinance
df= yf.download("QQQ", start="2020-01-01", end="2024-12-31", auto_adjust=False)
```

Data Peroid : **2020.01.01 ~ 2024.12.31** (5 years)

Price	Date	Adj Close	Close	High	Low	Open	Volume
Ticker		QQQ	QQQ	QQQ	QQQ	QQQ	QQQ
0	2020-01-02	209.325867	216.160004	216.160004	213.979996	214.399994	30969400
1	2020-01-03	207.408478	214.179993	215.470001	213.279999	213.300003	27518900
2	2020-01-06	208.744888	215.559998	215.589996	212.240005	212.500000	21655300
3	2020-01-07	208.715805	215.529999	216.139999	214.850006	215.639999	22139300
4	2020-01-08	210.284561	217.149994	218.139999	215.160004	215.500000	26397300

Null Values: 0

NA Values: False

02

EDA

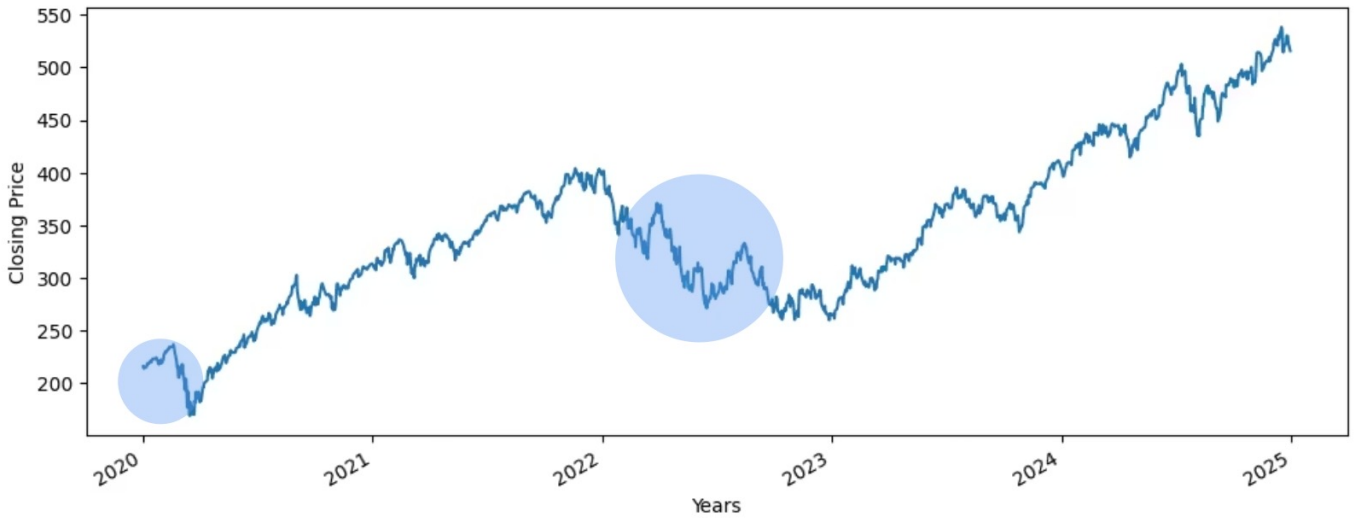
(Exploratory Data Analysis)

Stock Price Trends (2020~2024)

For understand the data, we visualized the stock price trends from 2020 to 2024.

This will help quickly **identify trends, volatility, outliers, and specific patterns**, providing valuable insights for model building.

```
import matplotlib.pyplot as plt

# Visualize 5 years of stock price data
fig, ax = plt.subplots(figsize=(10, 4))

ax.plot(df['date'], df['close'])
ax.set_xlabel('Years')
ax.set_ylabel('Closing Price')
fig.autofmt_xdate()

plt.tight_layout()
plt.show()
```



Overall, there is an upward trend,
but there are **sharp declines in the early part of 2020 and between 2022 and 2023.**

Key Events Impacting the Stock Market

2020.2

COVID-19 pandemic broke out, increasing global uncertainty and causing a sharp decline in stock prices.

2022.02

Russia vs. Ukraine war began.

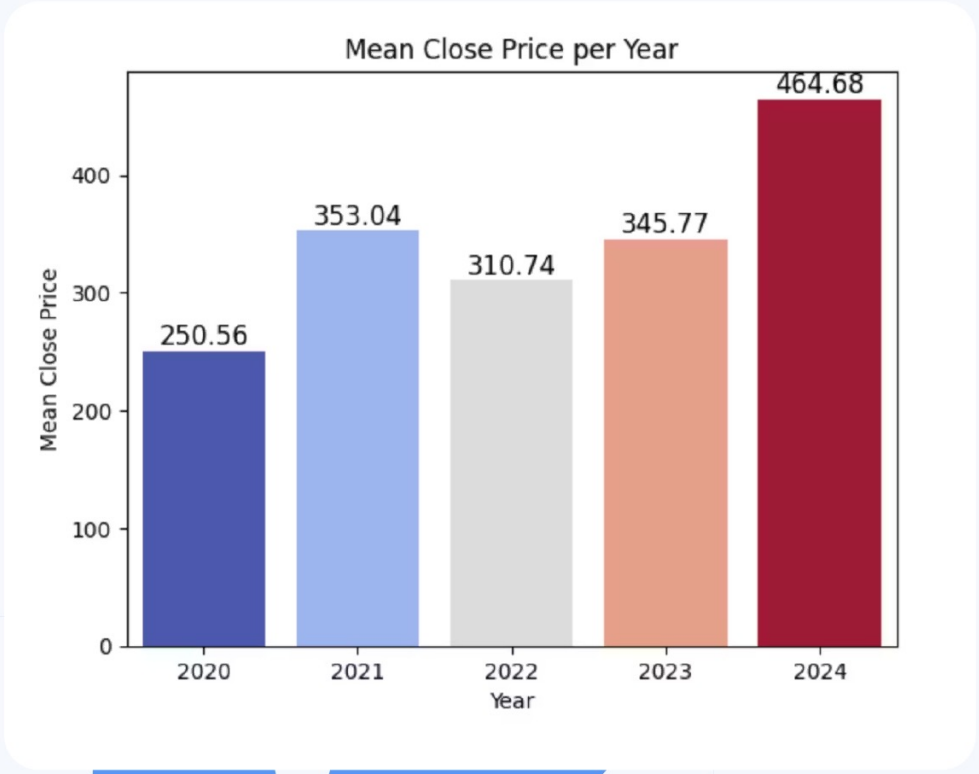
2022.06~09

Fed's interest rate hikes (each 0.75%) were implemented.

Yearly Trend

<Yearly Mean Close Price>

	year	close
0	2020	250.558380
1	2021	353.040874
2	2022	310.738645
3	2023	345.773800
4	2024	464.681512



📊 Annual Trend (5-Year Long-Term Performance)

Annual Average Return from 2020 to 2024: +16.7% (250 → 464)

- **2021:** +40.9% (250 → 353) → Tech sector boom
- **2022:** -12.0% (353 → 310) → Impact of interest rate hikes
- **2024:** +34.4% (345 → 464) → AI rally driving growth

03

Feature Engineering

(Technical Indicators)

Added Columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1257 entries, 0 to 1256
Data columns (total 23 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   date             1257 non-null   datetime64[ns]
 1   adj close        1257 non-null   float64
 2   close            1257 non-null   float64
 3   high             1257 non-null   float64
 4   low              1257 non-null   float64
 5   open             1257 non-null   float64
 6   volume           1257 non-null   int64
 7   month            1257 non-null   int32
 8   year             1257 non-null   int32
 9   MA_5             1253 non-null   float64
 10  MA_20            1238 non-null   float64
 11  MA_50            1208 non-null   float64
 12  RSI              1244 non-null   float64
 13  EMA_12           1257 non-null   float64
 14  EMA_26           1257 non-null   float64
 15  MACD             1257 non-null   float64
 16  Bollinger_Upper  1238 non-null   float64
 17  Bollinger_Lower  1238 non-null   float64
 18  Volume_MA_20     1238 non-null   float64
 19  Open_Close_Diff  1257 non-null   float64
 20  High_Low_Diff    1257 non-null   float64
 21  High_Open_Diff   1257 non-null   float64
 22  Low_Close_Diff   1257 non-null   float64
dtypes: datetime64[ns](1), float64(19), int32(2), int64(1)
```

04

Modeling (1)

Models

01 **Baseline Model - Linear Regression**

02 **XGBoost**

03 **LSTM**

MAE (Mean Average Error)

MAE is the average of the absolute difference between the predicted value and the actual value. The lower this value, the better the model.

```python
# Calculate 5-day Moving Average (MA_5)
df['MA_5'] = df['close'].rolling(window=5).mean()

# Use the 5-day Moving Average as the baseline model's prediction
# Since we want to predict the next day's closing price, shift the MA_5 by 1 day
df['predicted_close_ma'] = df['MA_5'].shift(1)

# Remove the NaN values caused by the shift
df = df.dropna(subset=['predicted_close_ma'])

# Define the features and target for the MA model
X_baseline = df[['MA_5']]  # Using the MA_5 as the feature
y_baseline = df['close']   # Actual close prices are the target

# Train-test split
X_train_baseline, X_test_baseline, y_train_baseline, y_test_baseline
= train_test_split(X_baseline, y_baseline, test_size=0.2, random_state=42)

# Train a model (Linear Regression)
baseline_model = LinearRegression()
baseline_model.fit(X_train_baseline, y_train_baseline)

# Prediction
y_pred_baseline = baseline_model.predict(X_test_baseline)

# Evaluate the baseline model
mae_baseline = mean_absolute_error(y_test_baseline, y_pred_baseline)
print(f"Baseline MAE (MA_5): {mae_baseline}")
```

Baseline Model

1. Calculating a 5-day moving average
2. Using the moving average as a predicted value
3. Data preprocessing (removing NaN values)
4. Learning a linear regression model
5. Model evaluation (with MAE)

Baseline MAE

4.0336

```python
# Fill missing values
df.ffill(inplace=True)

# Define the features and target variable
# Use the technical indicators (excluding date and adj close)
features = ['MA_5', 'MA_20', 'MA_50', 'RSI', 'EMA_12', 'EMA_26', 'MACD', 'Bollinger_Upper', 'Bollinger_Lower',
            'Volume_MA_20', 'Open_Close_Diff', 'High_Low_Diff', 'High_Open_Diff', 'Low_Close_Diff']
target = 'close'

X = df[features]
y = df[target]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train XGBoost model (with tuned hyperparameters)
model = xgb.XGBRegressor(
    n_estimators=1000, #EX
    learning_rate=0.01, #EX
    max_depth=6, #EX
    subsample=0.8,#EX
    colsample_bytree=0.8, #EX
    random_state=42
)

model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
print(f"XGBoost MAE: {mae}")
mae_xgboost = mae
```

XGBoost

1. Missing value handling
2. Define feature variables and target variables
3. Separate training data and test data (80%, 20%)
4. Train XGBoost regression model (Tune hyperparameters)
5. Perform prediction (Test data)
6. Model evaluation (with MAE)

XGBoost MAE

2.8195

```python
def create_dataset(data, window_size=60):
    X, y = [], []
    for i in range(window_size, len(data)):
        X.append(data[i-window_size:i, 0])  # X contains the data for the window size
        y.append(data[i, 0])  # y is the next day's closing price
    return np.array(X), np.array(y)

window_size = 60  # Use a window of 60 days to predict the next day
X, y = create_dataset(scaled_data, window_size)

# Reshape data (LSTM input requires 3D array)
X = np.reshape(X, (X.shape[0], X.shape[1], 1))  # (samples, timesteps, features)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build LSTM model
model = Sequential()

# LSTM layer
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))  # Dropout to prevent overfitting

# LSTM layer
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))

# Fully connected layer
model.add(Dense(units=1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))
```

LSTM

1. Normalize data (using MinMaxScaler)
2. Generate time series data (60-day window)
3. Transform data (convert to 3D array to fit LSTM input format)
4. Separate training/test data
5. Train LSTM model
6. Perform prediction
7. Model evaluation (with MAE)

LSTM MAE

0.0232

Results of Modeling (1)

1

LSTM (Tech)

0.0232

MAE

2

XGBoost (Tech)

2.8195

MAE

3

Baseline (MA_5)

4.0336

MAE

The **LSTM model** appears to have the **best performance**, as it recorded the lowest MAE.
However, there is a **possibility of overfitting.**
Therefore, we proceed by adding additional indicators for a more detailed analysis.

05

Modeling (2)

Technical + Macroeconomic indicators

Goal: To provide the model with a more **holistic view** of the financial market, improving prediction accuracy and generalizability we added three important factors from macroeconomic to our dataset, including:

1.Interest Rate

2.Gold Price

3.Consumer Price Index (CPI).

```
[ ] print(df[['date', 'close', 'interest_rate', 'gold_price', 'cpi']].head())
    print(df[['interest_rate', 'gold_price', 'cpi']].isnull().sum())
```

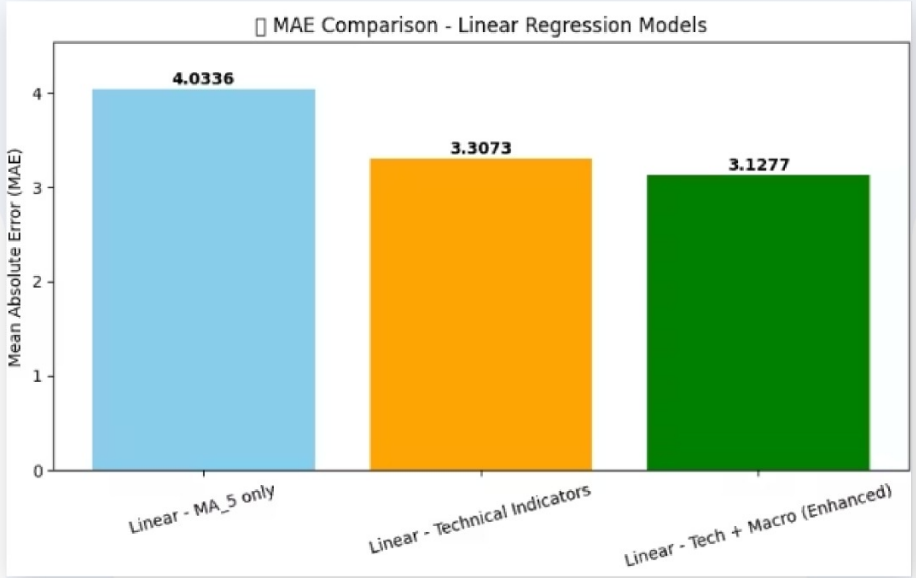
```
        date       close  interest_rate   gold_price      cpi
0 2020-01-09  218.990005           1.55  1551.699951  259.127
1 2020-01-10  218.429993           1.55  1557.500000  259.127
2 2020-01-13  220.949997           1.55  1548.400024  259.127
3 2020-01-14  220.080002           1.55  1542.400024  259.127
4 2020-01-15  220.169998           1.55  1552.099976  259.127
interest_rate    0
gold_price       0
cpi              0
dtype: int64
```

Baseline Linear Regression Comparison

```
mae_results = pd.DataFrame({
    'Model': ['MA_5 only', 'Technica
    'MAE': [mae_ma5, mae_baseline_te
})

print("MAE Comparison Table (Linear
display(mae_results)
```

MAE Comparison Table (Linear Regress

	Model	MAE
0	MA_5 only	4.033623
1	Technical Indicators	3.307302



7.2. Enhanced Linear Regression (with external features)

```
features_enhanced = [
    'MA_5', 'RSI', 'MACD', 'Volume_MA_20', 'Open_Close_Diff', 'High_Lo
    'interest_rate', 'gold_price', 'cpi'
]
```

```
# Drop NA rows for clean data
df_enhanced = df.dropna()

# Features and target
X_enhanced = df_enhanced[features_enhanced]
y_enhanced = df_enhanced['close']

# Split train/test
X_train_e, X_test_e, y_train_e, y_test_e = train_test_split(X_enhanced

# Train Linear Regression
lr_model_e = LinearRegression()
lr_model_e.fit(X_train_e, y_train_e)

# Predict and evaluate
y_pred_e = lr_model_e.predict(X_test_e)
mae_enhanced_base = mean_absolute_error(y_test_e, y_pred_e)

print(f"Enhanced Linear Regression MAE (Tech + Macro): {mae_enhanced_b
```

```
Enhanced Linear Regression MAE (Tech + Macro): 3.1277
```

Error margin decreased by adding more Tech indicators

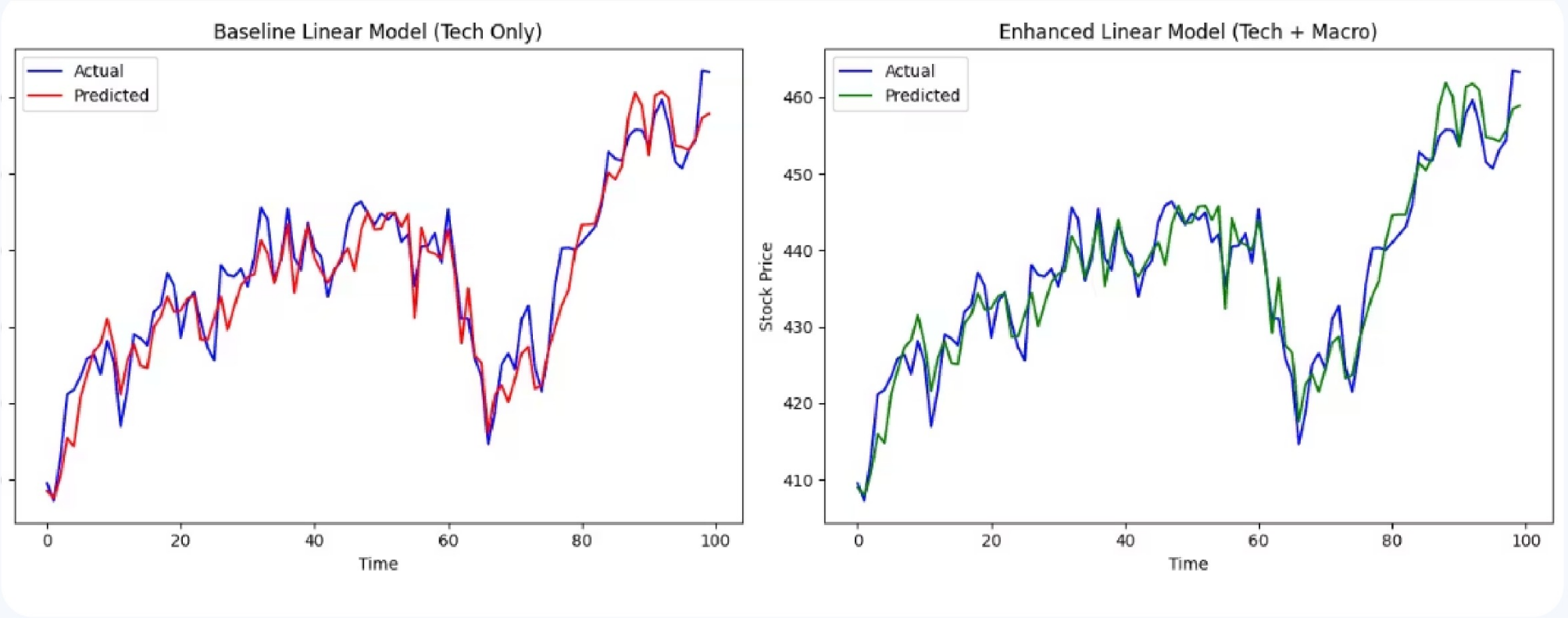
3.30 after applying tech indicators

Comparison Bar graphs

The lowest MAE with tech + macro indicators

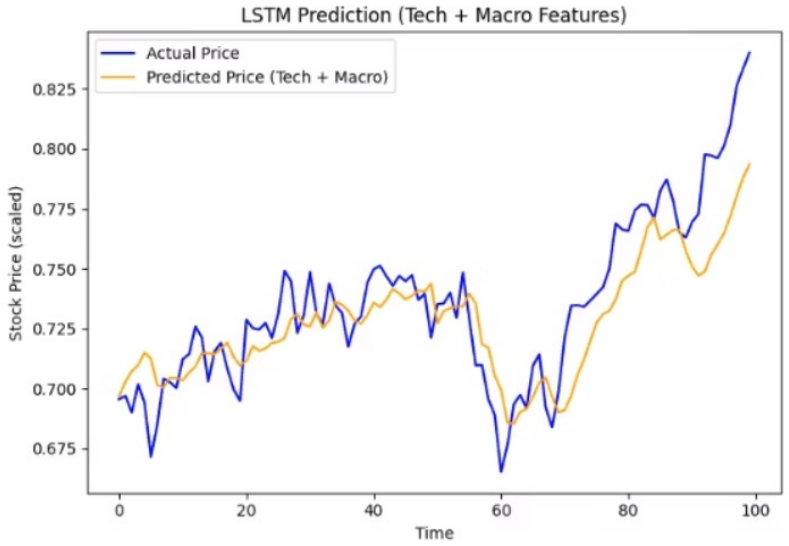
External MAE

Tech + Macro: 3.12



The enhanced model (green line) follows the actual stock prices (blue line) more closely than the baseline model (red line), indicating better accuracy. It clearly shows that adding macroeconomic data improves the model's predictive power.

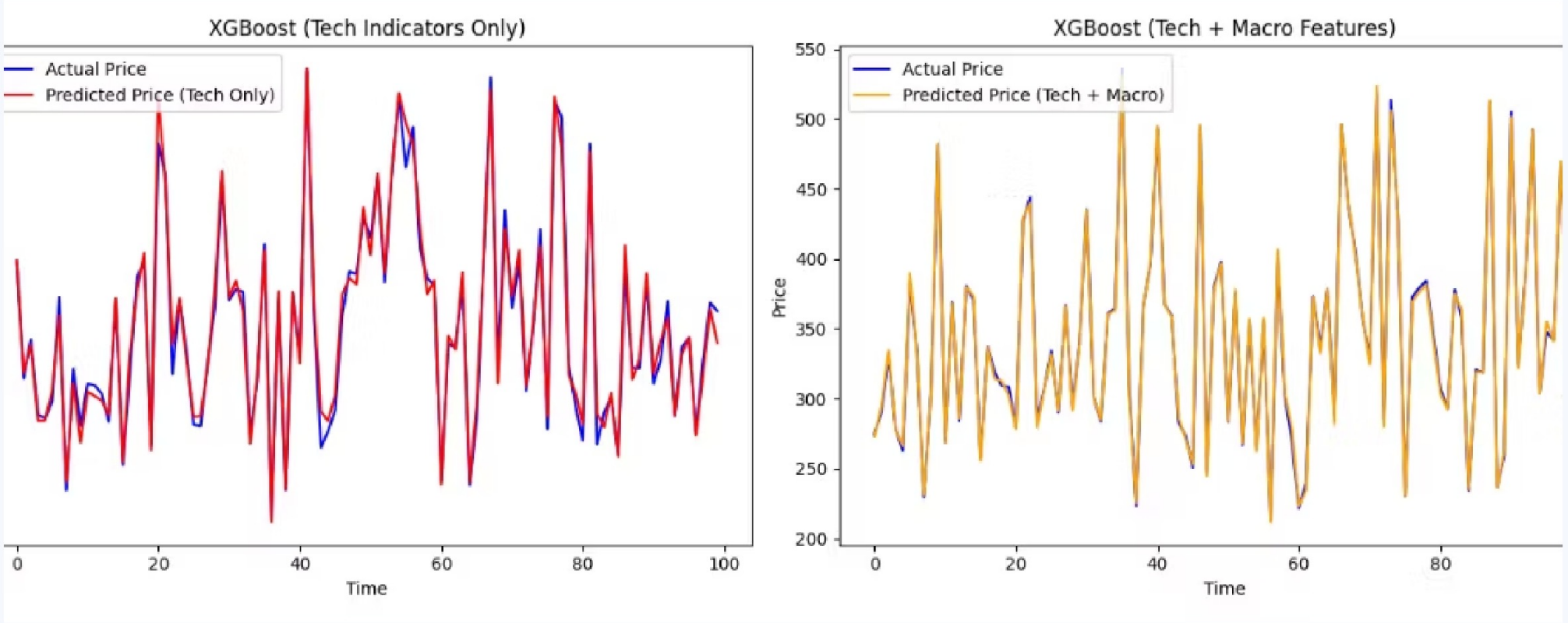
LSTM with Macro Features



The enhanced model shows improved alignment with actual prices, **especially during sharp** market movements, indicating that macro features help the LSTM model capture broader market dynamics more effectively.

Both models achieved identical MAE of 16.48
This suggests that either the macro features were not informative for short-term stock prediction, or the LSTM model couldn't effectively learn from them in its current configuration.

XGBoost + Macroeconomic



It shows XGBoost predictions using technical indicators only (left) versus technical plus macroeconomic features (right). The enhanced model with macro features closely tracks actual prices, **especially during sharp fluctuations**. This indicates that combining macroeconomic data with technical indicators improves XGBoost's prediction accuracy.

06

Accuracy

Accuracy Scores of Models

8.1. XGBoost Accuracy Score

```
[ ]  # Set a 1% error margin
     tolerance = 0.01

     # Accuracy for Tech-Only XGBoost Model
     accuracy_xgb_tech = np.mean(np.abs((y_test - y_pred) / y_test) < tolerance)
     print(f"📊 XGBoost Accuracy (Tech Only, ±1%): {accuracy_xgb_tech:.2%}")

     # Accuracy for Tech + Macro XGBoost Model
     accuracy_xgb_macro = np.mean(np.abs((y_test_m - y_pred_macro) / y_test_m) < tolerance)
     print(f"📈 XGBoost Accuracy (Tech + Macro, ±1%): {accuracy_xgb_macro:.2%}")
```

```
📊 XGBoost Accuracy (Tech Only, ±1%): 1.63%
📈 XGBoost Accuracy (Tech + Macro, ±1%): 74.79%
```

8.2. LSTM Accuracy Score

```
# 10% error margin
tolerance = 0.01

# Accuracy for LSTM (Tech Only)
accuracy_lstm_tech = np.mean(np.abs((y_test_lstm - y_pred_lstm) / y_test_lstm) < tolerance)
print(f"📊 LSTM Accuracy (Tech Only, ±1%): {accuracy_lstm_tech:.2%}")

# Accuracy for LSTM (Tech + Macro)
accuracy_lstm_macro = np.mean(np.abs((y_test_lstm_macro - y_pred_lstm_macro) / y_test_lstm_macro) < toleranc
print(f"📈 LSTM Accuracy (Tech + Macro, ±1%): {accuracy_lstm_macro:.2%}")
```

```
📊 LSTM Accuracy (Tech Only, ±1%): 6.26%
📈 LSTM Accuracy (Tech + Macro, ±1%): 6.26%
```

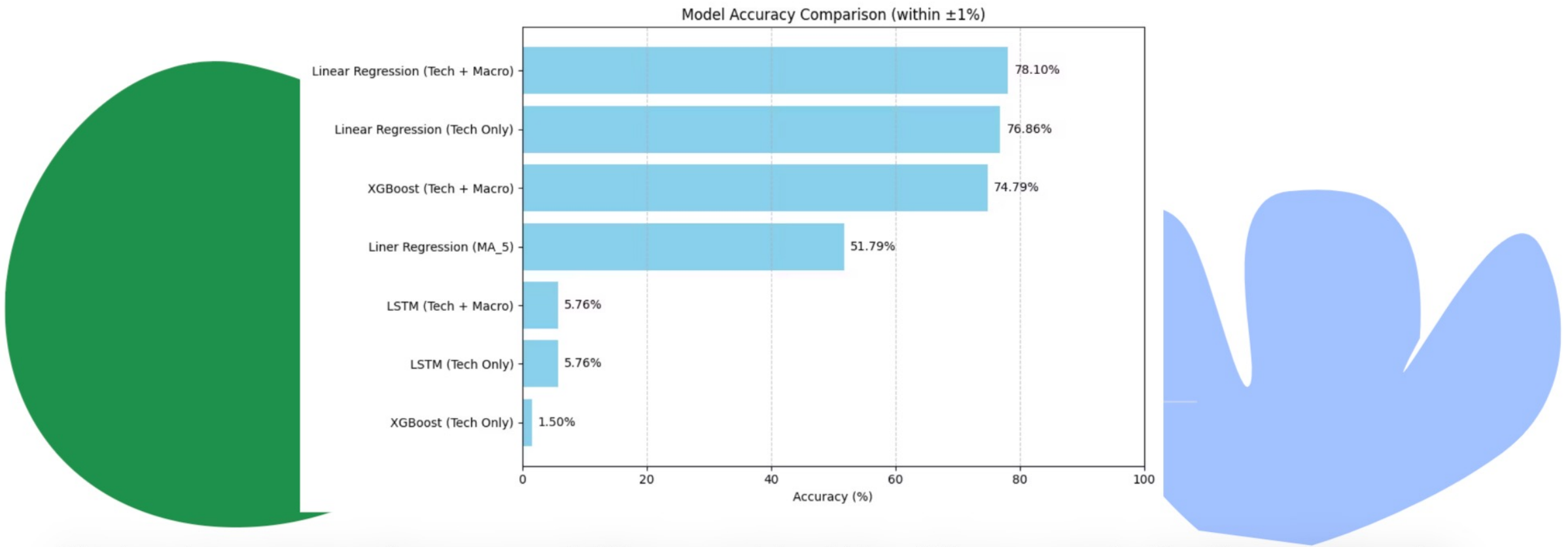
8.3. Baseline Accruracy Score

```
[ ]  # Set tolerance
     tolerance = 0.01  # 1% margin

     # Accuracy calculation (within ±1% of true value)
     accuracy_ma5 = np.mean(np.abs((y_test_baseline - y_pred_baseline) / y_test_baseline) < tolerance)

     print(f"📉 Baseline Linear Regression Accuracy (MA_5 only, ±1%): {accuracy_ma5:.2%}")


     # Accuracy calculation (within ±10% of true value)
     accuracy_tech_lr = np.mean(np.abs((y_test_b - y_pred_b) / y_test_b) < tolerance)

     print(f"📊 Linear Regression Accuracy (Technical Indicators, ±1%): {accuracy_tech_lr:.2%}")

     # Accuracy calculation (within ±10% of true value)
     accuracy_tech_macro = np.mean(np.abs((y_test_e - y_pred_e) / y_test_e) < tolerance)

     print(f"📊 Linear Regression Accuracy (Technical+Macro Indicators, ±1%): {accuracy_tech_macro:.2%}")
```

```
📉 Baseline Linear Regression Accuracy (MA_5 only, ±1%): 51.79%
📊 Linear Regression Accuracy (Technical Indicators, ±1%): 76.86%
📊 Linear Regression Accuracy (Technical+Macro Indicators, ±1%): 78.10%
```

The percentage of model predictions that fall within a **1% error margin** of the actual stock price; this margin is referred to as **1% tolerance.** A higher accuracy score means more predictions were very close to the real values.

Model Accuracy Comparison



This bar chart compares the accuracy of different models within a ±1% error margin. The **Linear Regression (Tech + Macro)** model achieved the highest accuracy at **78.10%**, followed by **Linear Regression (Tech Only)** and **XGBoost (Tech + Macro)**. In contrast, both **LSTM models** showed the **same low accuracy (5.76%)**, suggesting that LSTM struggled to learn effectively from the data; possibly due to insufficient training time, model complexity, or the short sequence length used for time series input.

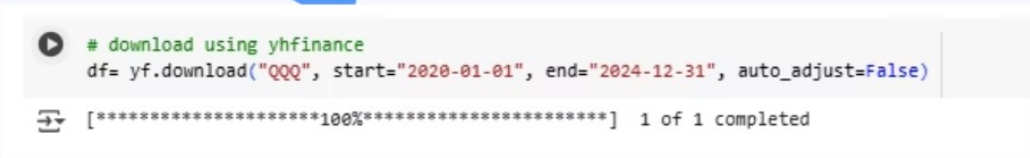
Key Takeaways

QQQ analytics project demonstrated that combining technical indicators with macroeconomic features (external features) significantly improves the accuracy of traditional machine learning models like Linear Regression and XGBoost for stock price prediction.

Among all approaches, **Linear Regression with both technical and macro data delivered the highest accuracy**, proving that simplicity, when paired with meaningful features, can outperform more complex models like LSTM. This highlights the importance of thoughtful feature engineering and model selection over relying solely on deep learning architectures.

References

QQQ Historical Stock Data (2020–2024) was retrieved using the yfinance Python library.
Source: **Yahoo Finance**, via yfinance package.
Ticker: **"QQQ"** — Invesco QQQ Trust
Accessed using:



"📊Stock Market Analysis 📈 + Prediction Using LSTM." *Kaggle.com*, www.kaggle.com/code/faressayah/stock-market-analysis-prediction-using-lstm.

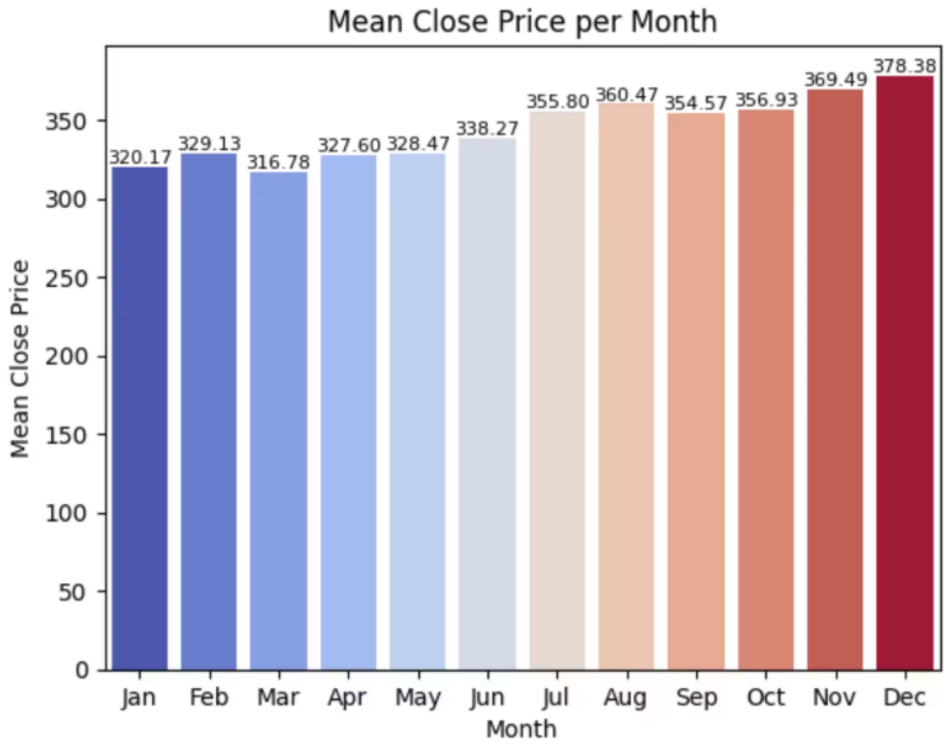
"LSTM Time Series + Stock Price Prediction = FAIL." *Kaggle.com*, www.kaggle.com/code/carlmcbrideellis/lstm-time-series-stock-price-prediction-fail.

zeyadsayedadbullah. "Stock Price Prediction Using XGBOOST PROPHET ARIMA." *Kaggle.com*, Kaggle, 15 Mar. 2024, www.kaggle.com/code/zeyadsayedadbullah/stock-price-prediction-using-xgboost-prophet-arima. Accessed 9 Apr. 2025.

Appendix 1. Monthly Trends in Stock Seasonality

<Monthly Mean Close Price>

	month	close
0	1	320.169998
1	2	329.128750
2	3	316.779909
3	4	327.599127
4	5	328.470763
5	6	338.271619
6	7	355.802095
7	8	360.470450
8	9	354.570971
9	10	356.928442
10	11	369.487185
11	12	378.381808



📈 Monthly Seasonality

- **Bullish in the Second Half** (July to December)
  From June (338) to December (378), a +11.8% increase.
  Especially a sharp rise from November (369) to December (378)
  → aligns with the "Year-end Rally" phenomenon.

- **Volatility in the First Half** (January to March)
  From January (320) to March (316), a -1.3% decrease
  → no sign of the "January Effect"

- **Summer Stagnation** (July to September)
  From July (355) to September (354), a slight decline of -0.3%.

Appendix 2. Technical Indicators

Indicator	Description
Moving Averages (MA)	Smooths stock price movements to identify short-term and long-term trends (ex. 5-day, 20-day, 50-day, 200-day)
Relative Strength Index (RSI)	Measures whether a stock is overbought or oversold, reflecting price momentum.
MACD (Moving Average Convergence Divergence)	Identifies trend changes and momentum using two moving averages.
Bollinger Bands	Measures price volatility and deviation from the average price.
Volume Volatility	Analyzes the relationship between price and trading volume to predict market movements.
Open-Close Difference	Indicates daily price volatility and potential trend strength.
High-Low Difference	Represents intraday volatility, with larger differences signaling higher fluctuations.
High-Open Difference	Measures how much the high price (high) has increased compared to the opening price (open) during the day.
Low-Close Difference	Measures how much the closing price (close) has recovered compared to the low price (low) during the day.

Thank you!