

STOCK MARKET ANALYSIS

Of Microsoft



OlhaBabicheva/stock-market-analysis-MSFT



3 Contributors 0 Issues 0 Stars 0 Forks

OlhaBabicheva/stock-market-analysis-MSFT

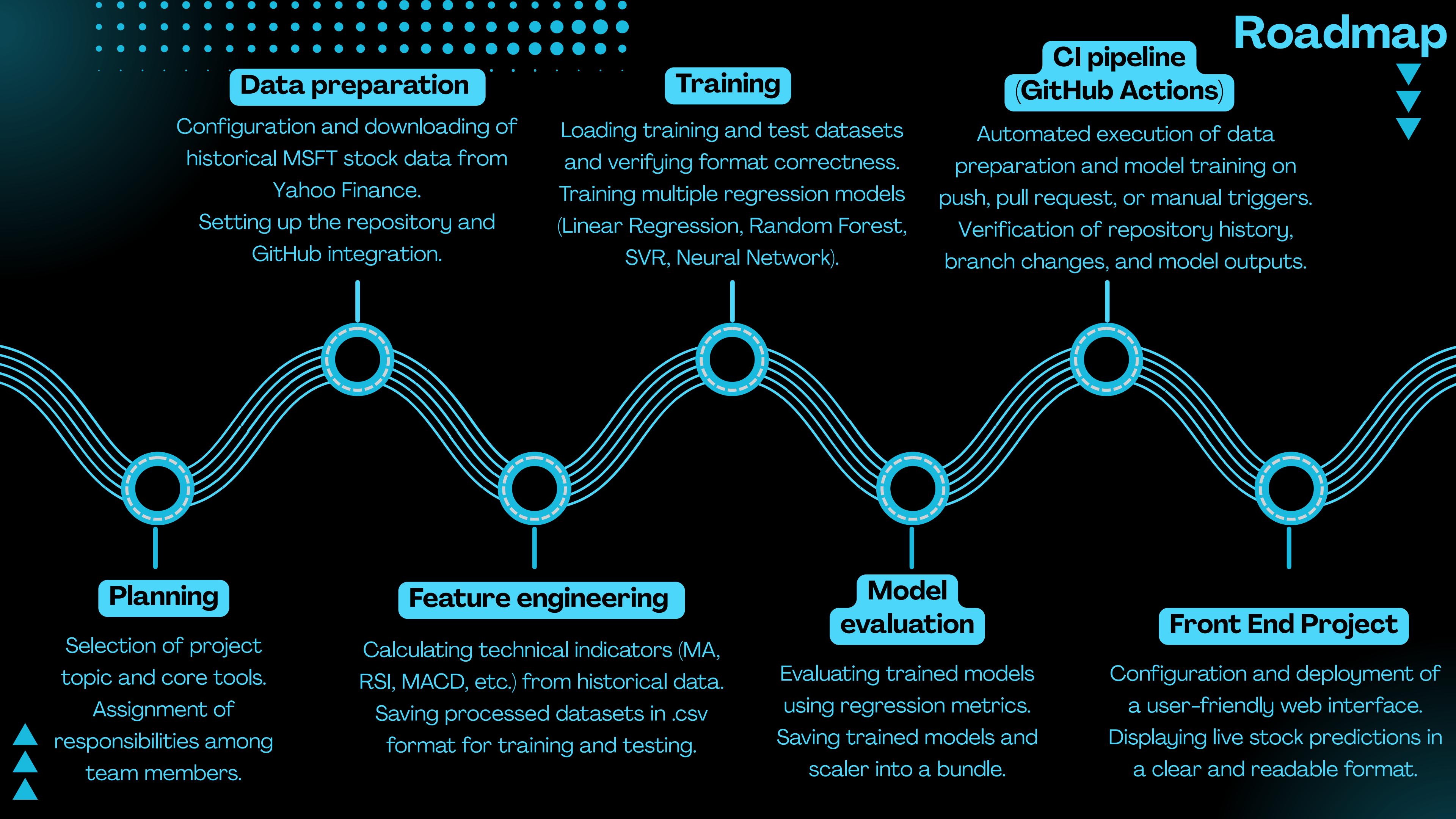
Contribute to OlhaBabicheva/stock-market-analysis-MSFT development by creating an account on GitHub.

[GitHub](#)

Olha Babicheva
Anna Saldat
Agata Jabłońska



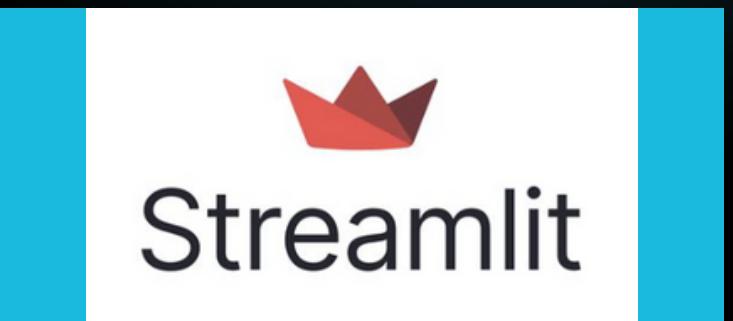
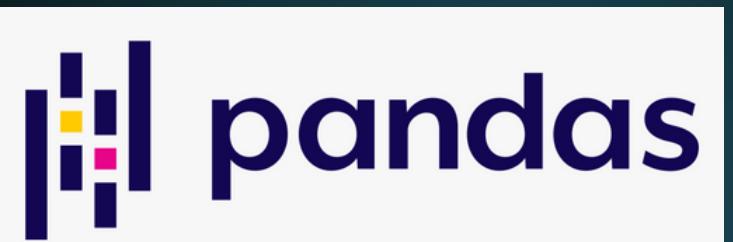
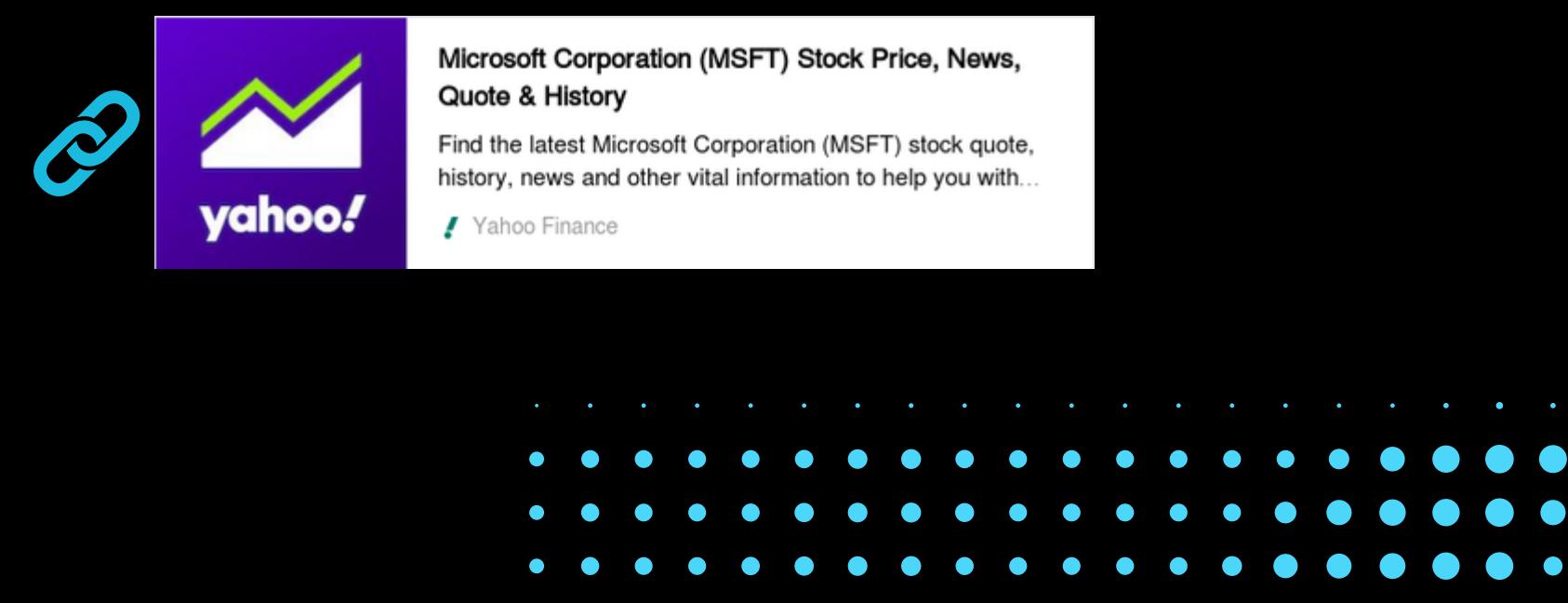
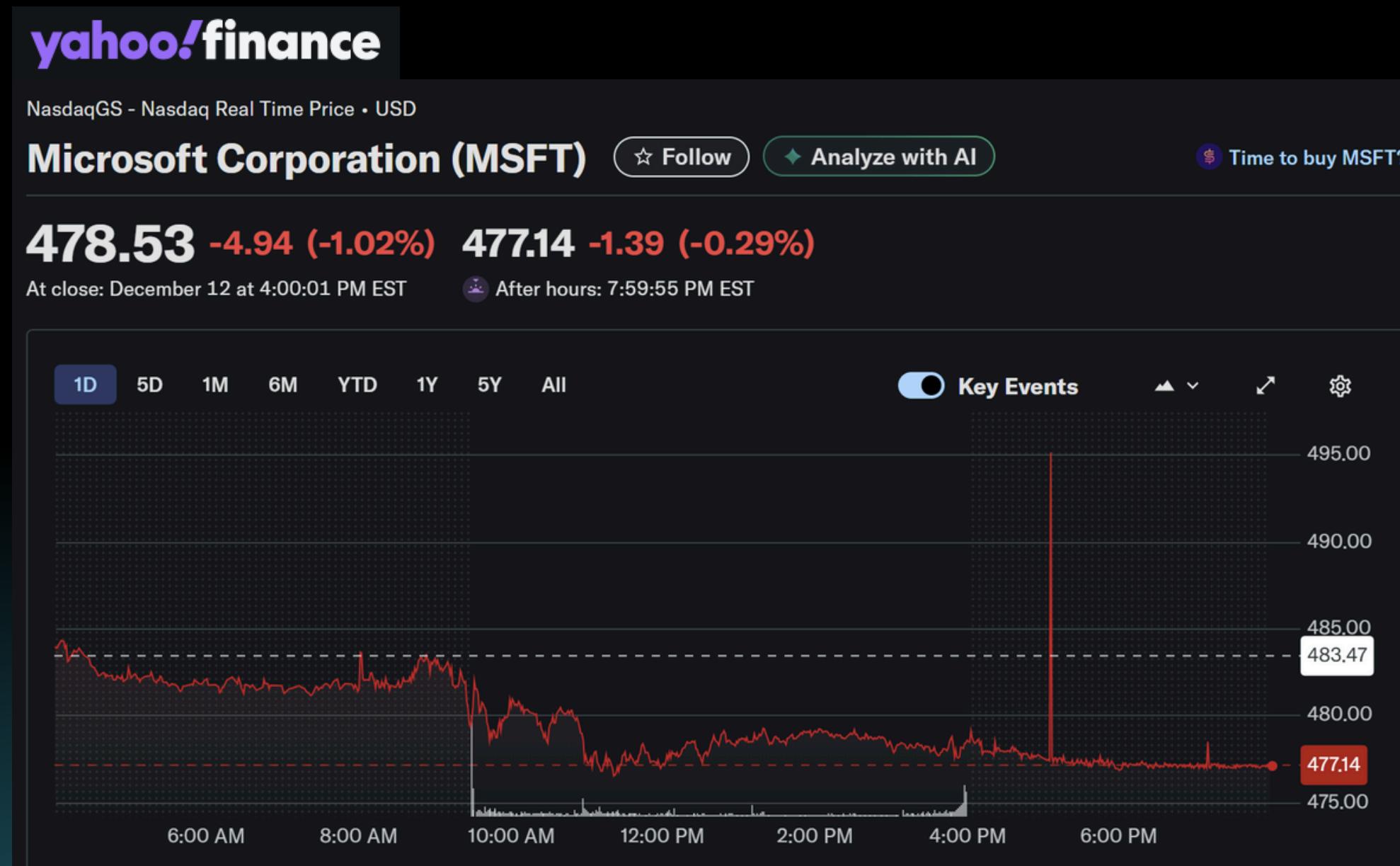
Roadmap



Necessary libraries

- pandas
- scikit-learn
- yfinance
- streamlit (used in next phases)

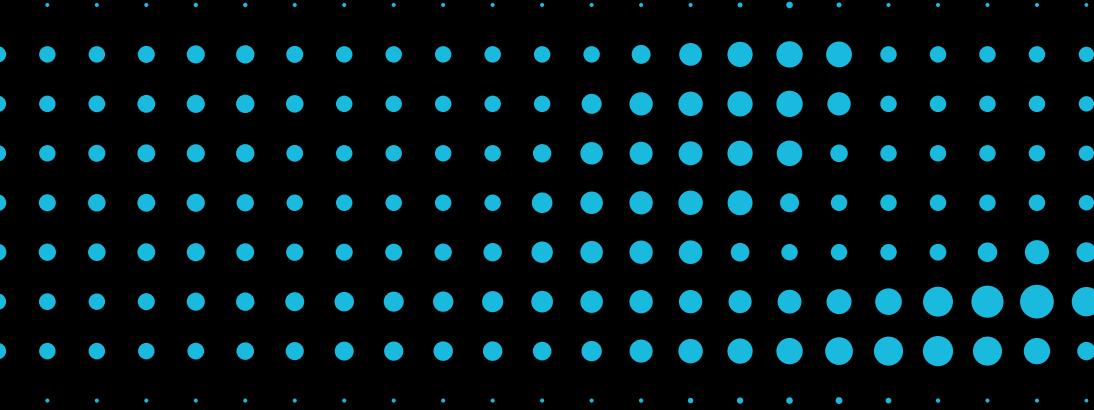
>>> Database **YahooFinance**



Implemented functions



```
1 import yfinance as yf # Yahoo Finance API downloader (fetches historical market data)
2 import numpy as np # Numerical library (used for mathematical operations)
3
4 # ---- CONFIGURATION ----
5 TICKER = 'MSFT'
6 START_DATE = '2019-01-01'
7 TEST_SIZE_RATIO = 0.2
8 TRAIN_FILE = 'train.csv'
9 TEST_FILE = 'test.csv'
```



```
11 def download_and_prepare_data(ticker, start_date, test_ratio):
12     """
13     Downloads historical stock data, performs feature engineering, and splits it
14     chronologically into training and testing sets.
15     """
16     print(f"1. Downloading historical data for {ticker} (Start Date: {start_date})...")
17
18     try:
19         data = yf.download(ticker, start=start_date, end=None, progress=False)
20     except Exception as e:
21         print(f"Error downloading data: {e}")
22         return None
23
24     if data.empty:
25         print("No data retrieved. Exiting data pipeline.")
26         return None
27
28     print(f"Download successful. Total samples: {len(data)}")
29     print("2. Feature Engineering: Calculating Moving Averages and setting target...")
30
31     # Get the Close prices
32     close_prices = data['Close'].copy()
33
34     # Feature 1 & 2: Simple Moving Averages
35     data['MA_10'] = close_prices.rolling(window=10).mean()
36     data['MA_30'] = close_prices.rolling(window=30).mean()
37
38     # Feature 3: Daily Range (Measure of volatility)
39     data['Daily_Range'] = data['High'] - data['Low']
40
```



Implemented functions

```
41 # Feature 4 & 5: Daily Returns
42 #(returns capture day-to-day relative price changes)
43 data['Return_1d'] = close_prices.pct_change()
44 # Log returns are time-additive and commonly assumed to be closer to normality
45 data['Log_Return_1d'] = np.log(close_prices / close_prices.shift(1))
46
47 # Feature 6 & 7: Momentum
48 # (measures medium-term trend persistence)
49 # Positive values indicate upward pressure, negative values downward pressure
50 data['Momentum_5'] = close_prices - close_prices.shift(5)
51 data['Momentum_10'] = close_prices - close_prices.shift(10)
52
53 # Feature 8: RSI (Relative Strength Index)
54 # RSI measures the speed and magnitude of recent price movements
55 # Values >70 often indicate overbought conditions, Values <30 often indicate oversold conditions
56
57 delta = close_prices.diff()
58 # Separate positive and negative price changes
59 gain = delta.clip(lower=0)
60 loss = -delta.clip(upper=0)
61 # Rolling averages of gains and losses (standard 14-day window)
62 avg_gain = gain.rolling(14).mean()
63 avg_loss = loss.rolling(14).mean()
64 # Relative Strength and RSI formula
65 rs = avg_gain / avg_loss
66 data['RSI_14'] = 100 - (100 / (1 + rs))
67
68 # Feature 9 & 10: MACD Indicator -- trend-following momentum indicator
69 # (MACD captures the relationship between short-term and long-term trends)
70
71 # Short-term exponential moving average
72 ema_12 = close_prices.ewm(span=12, adjust=False).mean()
73 # Long-term exponential moving average
74 ema_26 = close_prices.ewm(span=26, adjust=False).mean()
```



Implemented functions

```
75     # MACD line
76     data['MACD'] = ema_12 - ema_26
77     # Signal line (EMA of MACD)
78     data['MACD_Signal'] = data['MACD'].ewm(span=9, adjust=False).mean()
79
80     # Target Variable: Predict the next day's Close price (Next_Close)
81     data['Next_Close'] = close_prices.shift(-1)
82
83     # Define the features we will use for the model
84     FEATURES = [
85         'Close', 'MA_10', 'MA_30',
86         'Volume', 'Daily_Range', 'Return_1d',
87         'Log_Return_1d', 'Momentum_5', 'Momentum_10',
88         'RSI_14', 'MACD', 'MACD_Signal'
89     ]
90
91     # Drop rows that have NaN values due to the initial 30-day rolling
92     # window and the last target row
93     df = data.dropna()
94
95     print(f"Data cleaned. Usable data points: {len(df)}")
96
97     # Calculate the index for the chronological split
98     train_size = int(len(df) * (1 - test_ratio))
99
100    # Split the dataframes
101    df_train = df.iloc[:train_size].copy()
102    df_test = df.iloc[train_size: ].copy()
103
104    return df_train, df_test, FEATURES
```



Implemented functions

```
106 def save_data(df_train, df_test, features):
107     """Saves the train and test DataFrames, including the target and features, to CSV."""
108
109     # Columns to save: Target & Features
110     cols_to_save = ['Next_Close'] + features
111
112     # Save to CSV files
113     df_train[cols_to_save].to_csv(TRAIN_FILE)
114     df_test[cols_to_save].to_csv(TEST_FILE)
115
116     print("3. Data split and saved to CSV files:")
117     print(f"Training set size: {len(df_train)} samples ({TRAIN_FILE})")
118     print(f"Testing set size: {len(df_test)} samples ({TEST_FILE})")
119
120     return features
121
122
123 if __name__ == '__main__':
124     data_output = download_and_prepare_data(TICKER, START_DATE, TEST_SIZE_RATIO)
125
126     if data_output:
127         df_train, df_test, features_list = data_output
128         save_data(df_train, df_test, features_list)
129
```



>>> Implemented functions

```
1 import os # Standard library for interacting with the operating system (checking file paths)
2 import pandas as pd # Data manipulation library
3 import numpy as np # Numerical library (used for mathematical operations)
4 import joblib # Library for saving and loading models
5 from sklearn.linear_model import LinearRegression, Ridge # Ridge adds L2 regularization to Linear Regression
6 from sklearn.ensemble import RandomForestRegressor # A bagging-based ensemble model (often more stable than boosting)
7 from sklearn.svm import SVR # Support Vector Regression (good for non-linear patterns)
8 from sklearn.neural_network import MLPRegressor # Multi-layer Perceptron (Neural Network) regressor
9 from sklearn.preprocessing import StandardScaler # Crucial for models like SVR and Ridge
10 from sklearn.metrics import mean_squared_error, r2_score # Functions to calculate model accuracy
11
12 # ---- CONFIGURATION ---
13 TRAIN_FILE = 'train.csv'
14 TEST_FILE = 'test.csv'
15 MODEL_EXPORT_FILE = 'models_bundle.joblib'
16 FEATURES = [
17     'Close', 'MA_10', 'MA_30',
18     'Volume', 'Daily_Range', 'Return_1d',
19     'Log_Return_1d', 'Momentum_5', 'Momentum_10',
20     'RSI_14', 'MACD', 'MACD_Signal'
21 ]
22 TARGET = 'Next_Close'
```



Implemented functions



```
25 def load_data(train_path, test_path):
26     """Loads the training and testing datasets from CSV files."""
27     if not os.path.exists(train_path) or not os.path.exists(test_path):
28         print(f"Error: One or both data files ({train_path}, {test_path}) not found.")
29         print("Please run 'data_preparation.py' first.")
30         return None, None
31
32     print("1. Loading data from CSV files.")
33     COL_NAMES = ['Date', TARGET] + FEATURES
34     df_train = pd.read_csv(train_path,
35                           index_col=0,
36                           parse_dates=True,
37                           skiprows=3,
38                           header=None,
39                           names=COL_NAMES)
40
41     df_test = pd.read_csv(test_path,
42                           index_col=0,
43                           parse_dates=True,
44                           skiprows=3,
45                           header=None,
46                           names=COL_NAMES)
47
48     print(f"Train samples loaded: {len(df_train)}")
49     print(f"Test samples loaded: {len(df_test)}")
50
51     # Ensure all required features and the target exist
52     if not all(col in df_train.columns for col in FEATURES + [TARGET]):
53         print("Error: Missing required columns in loaded data. Check FEATURES and TARGET lists.")
54         return None, None
55
56     return df_train, df_test
57
```



Implemented functions

```
58 def train_and_evaluate(df_train, df_test, features, target):
59     """Trains multiple models to predict the Next_Close price."""
60
61     # Prepare data for sklearn
62     X_train_raw = df_train[features].values
63     y_train = df_train[target].values
64     X_test_raw = df_test[features].values
65     y_test = df_test[target].values
66
67     # Scaling is mandatory for SVR and Ridge to perform correctly
68     scaler = StandardScaler()
69     X_train = scaler.fit_transform(X_train_raw)
70     X_test = scaler.transform(X_test_raw)
71
72     models = {
73         "Linear Regression": LinearRegression(),
74         "Ridge (L2)": Ridge(alpha=10.0), # Penalizes large coefficients to reduce noise sensitivity
75         "Random Forest": RandomForestRegressor(
76             n_estimators=400, # Number of trees
77             random_state=42
78         ), # Stable bagging
79         "Neural Network (MLP)": MLPRegressor(
80             hidden_layer_sizes=(10, 64, 64), # Three hidden layers with 10, 64 and 64 neurons
81             activation='relu', # Rectified Linear Unit activation function
82             solver='adam', # Optimizer for weight optimization
83             max_iter=1000, # Maximum number of iterations
84             random_state=42 # For reproducible results
85         ),
86         "SVR (RBF Kernel)": SVR(
87             kernel='linear',
88             epsilon=0.1 # Threshold where no penalty is given to errors
89         ), # Non-linear boundary mapping
90     }
91
92     results = []
93     trained_models = {}
```



Implemented functions

```
95     print(f"2. Training models to predict absolute {target}...")
96
97     for name, model in models.items():
98         # Training directly on the absolute price
99         model.fit(X_train, y_train)
100        y_pred = model.predict(X_test)
101        # Evaluation
102        mse = mean_squared_error(y_test, y_pred)
103        rmse = np.sqrt(mse)
104        r2 = r2_score(y_test, y_pred)
105        results.append({
106            "Model": name,
107            "MSE": round(mse, 4),
108            "RMSE (USD)": round(rmse, 4),
109            "R2 Score": round(r2, 4)
110        })
111        trained_models[name] = model
112
113    # Save the bundle: models, scaler, and feature list
114    bundle = {
115        'models': trained_models,
116        'scaler': scaler,
117        'features': features
118    }
119    joblib.dump(bundle, MODEL_EXPORT_FILE)
120    print(f"3. Models and scaler saved to {MODEL_EXPORT_FILE}")
121
122    print("*50)
123    print(pd.DataFrame(results).sort_values(by="RMSE (USD)").to_string(index=False))
124    print("*50)
125
126    comparison_df = pd.DataFrame(results)
127    print("*50)
128    print(f"MODEL COMPARISON SUMMARY (Target: {target})")
129    print("*50)
130    print(comparison_df.sort_values(by="RMSE (USD)").to_string(index=False))
131    print("*50)
132
133    return comparison_df
```



>>> Implemented functions

```
136 if __name__ == '__main__':
137     # Execute the loading and training workflow
138     df_train_data, df_test_data = load_data(TRAIN_FILE, TEST_FILE)
139
140     if df_train_data is not None:
141         train_and_evaluate(df_train_data, df_test_data, FEATURES, TARGET)
142
```



Implemented functions



```
1 from datetime import datetime, timedelta
2 import streamlit as st
3 import numpy as np
4 import yfinance as yf
5 import joblib
6
7 #----APP CONFIGURATION----
8 TICKER = 'MSFT'
9 MODEL_PATH = 'models_bundle.joblib'
10
11 st.set_page_config(page_title="MSFT Price Predictor", layout="wide")
12
13 @st.cache_resource
14 def load_bundle():
15     """Loads the saved models and scaler."""
16     try:
17         return joblib.load(MODEL_PATH)
18     except:
19         return None
20
21     . . .
22     . . .
23     . . .
24     . . .
25     . . .
26     . . .
27     . . .
28     . . .
29     . . .
30     . . .
31     . . .
32     . . .
33     . . .
34     . . .
35     . . .
36     . . .
37     . . .
38     . . .
39     . . .
40     . . .
41     . . .
42     . . .
43     . . .
44     . . .
45     . . .
46     . . .
47     . . .
48     . . .
49     . . .
50     . . .
51     . . .
52     . . .
53     . . .
```

```
21 def get_latest_data(ticker):
22     """Fetches and prepares the most recent data for prediction."""
23     # Fetch enough data to calculate technical indicators (approx 60 days)
24     end_date = datetime.now()
25     start_date = end_date - timedelta(days=60)
26     try:
27         data = yf.download(ticker, start=start_date, end=None, progress=False)
28     except Exception as e:
29         print(f"Error downloading data: {e}")
30         return None
31
32     if data.empty:
33         return None
34
35     # Feature Engineering (Mirrors data_preparation.py)
36     # Get the Close prices
37     close_prices = data['Close'].copy()
38
39     # Feature 1 & 2: Simple Moving Averages
40     data['MA_10'] = close_prices.rolling(window=10).mean()
41     data['MA_30'] = close_prices.rolling(window=30).mean()
42
43     # Feature 3: Daily Range (Measure of volatility)
44     data['Daily_Range'] = data['High'] - data['Low']
45
46     # Feature 4 & 5: Daily Returns
47     data['Return_1d'] = close_prices.pct_change()
48     data['Log_Return_1d'] = np.log(close_prices / close_prices.shift(1))
49
50     # Feature 6 & 7: Momentum
51     data['Momentum_5'] = close_prices - close_prices.shift(5)
52     data['Momentum_10'] = close_prices - close_prices.shift(10)
53     . . .
54     . . .
55     . . .
56     . . .
57     . . .
58     . . .
59     . . .
60     . . .
61     . . .
62     . . .
63     . . .
64     . . .
65     . . .
66     . . .
67     . . .
68     . . .
69     . . .
70     . . .
71     . . .
72     . . .
73     . . .
74     . . .
75     . . .
76     . . .
77     . . .
78     . . .
79     . . .
80     . . .
81     . . .
82     . . .
83     . . .
84     . . .
85     . . .
86     . . .
87     . . .
88     . . .
89     . . .
90     . . .
91     . . .
92     . . .
93     . . .
94     . . .
95     . . .
96     . . .
97     . . .
98     . . .
99     . . .
100    . . .
101    . . .
102    . . .
103    . . .
104    . . .
105    . . .
106    . . .
107    . . .
108    . . .
109    . . .
110    . . .
111    . . .
112    . . .
113    . . .
114    . . .
115    . . .
116    . . .
117    . . .
118    . . .
119    . . .
120    . . .
121    . . .
122    . . .
123    . . .
124    . . .
125    . . .
126    . . .
127    . . .
128    . . .
129    . . .
130    . . .
131    . . .
132    . . .
133    . . .
134    . . .
135    . . .
136    . . .
137    . . .
138    . . .
139    . . .
140    . . .
141    . . .
142    . . .
143    . . .
144    . . .
145    . . .
146    . . .
147    . . .
148    . . .
149    . . .
150    . . .
151    . . .
152    . . .
153    . . .
154    . . .
155    . . .
156    . . .
157    . . .
158    . . .
159    . . .
160    . . .
161    . . .
162    . . .
163    . . .
164    . . .
165    . . .
166    . . .
167    . . .
168    . . .
169    . . .
170    . . .
171    . . .
172    . . .
173    . . .
174    . . .
175    . . .
176    . . .
177    . . .
178    . . .
179    . . .
180    . . .
181    . . .
182    . . .
183    . . .
184    . . .
185    . . .
186    . . .
187    . . .
188    . . .
189    . . .
190    . . .
191    . . .
192    . . .
193    . . .
194    . . .
195    . . .
196    . . .
197    . . .
198    . . .
199    . . .
200    . . .
201    . . .
202    . . .
203    . . .
204    . . .
205    . . .
206    . . .
207    . . .
208    . . .
209    . . .
210    . . .
211    . . .
212    . . .
213    . . .
214    . . .
215    . . .
216    . . .
217    . . .
218    . . .
219    . . .
220    . . .
221    . . .
222    . . .
223    . . .
224    . . .
225    . . .
226    . . .
227    . . .
228    . . .
229    . . .
230    . . .
231    . . .
232    . . .
233    . . .
234    . . .
235    . . .
236    . . .
237    . . .
238    . . .
239    . . .
240    . . .
241    . . .
242    . . .
243    . . .
244    . . .
245    . . .
246    . . .
247    . . .
248    . . .
249    . . .
250    . . .
251    . . .
252    . . .
253    . . .
254    . . .
255    . . .
256    . . .
257    . . .
258    . . .
259    . . .
260    . . .
261    . . .
262    . . .
263    . . .
264    . . .
265    . . .
266    . . .
267    . . .
268    . . .
269    . . .
270    . . .
271    . . .
272    . . .
273    . . .
274    . . .
275    . . .
276    . . .
277    . . .
278    . . .
279    . . .
280    . . .
281    . . .
282    . . .
283    . . .
284    . . .
285    . . .
286    . . .
287    . . .
288    . . .
289    . . .
290    . . .
291    . . .
292    . . .
293    . . .
294    . . .
295    . . .
296    . . .
297    . . .
298    . . .
299    . . .
300    . . .
301    . . .
302    . . .
303    . . .
304    . . .
305    . . .
306    . . .
307    . . .
308    . . .
309    . . .
310    . . .
311    . . .
312    . . .
313    . . .
314    . . .
315    . . .
316    . . .
317    . . .
318    . . .
319    . . .
320    . . .
321    . . .
322    . . .
323    . . .
324    . . .
325    . . .
326    . . .
327    . . .
328    . . .
329    . . .
330    . . .
331    . . .
332    . . .
333    . . .
334    . . .
335    . . .
336    . . .
337    . . .
338    . . .
339    . . .
340    . . .
341    . . .
342    . . .
343    . . .
344    . . .
345    . . .
346    . . .
347    . . .
348    . . .
349    . . .
350    . . .
351    . . .
352    . . .
353    . . .
354    . . .
355    . . .
356    . . .
357    . . .
358    . . .
359    . . .
360    . . .
361    . . .
362    . . .
363    . . .
364    . . .
365    . . .
366    . . .
367    . . .
368    . . .
369    . . .
370    . . .
371    . . .
372    . . .
373    . . .
374    . . .
375    . . .
376    . . .
377    . . .
378    . . .
379    . . .
380    . . .
381    . . .
382    . . .
383    . . .
384    . . .
385    . . .
386    . . .
387    . . .
388    . . .
389    . . .
390    . . .
391    . . .
392    . . .
393    . . .
394    . . .
395    . . .
396    . . .
397    . . .
398    . . .
399    . . .
400    . . .
401    . . .
402    . . .
403    . . .
404    . . .
405    . . .
406    . . .
407    . . .
408    . . .
409    . . .
410    . . .
411    . . .
412    . . .
413    . . .
414    . . .
415    . . .
416    . . .
417    . . .
418    . . .
419    . . .
420    . . .
421    . . .
422    . . .
423    . . .
424    . . .
425    . . .
426    . . .
427    . . .
428    . . .
429    . . .
430    . . .
431    . . .
432    . . .
433    . . .
434    . . .
435    . . .
436    . . .
437    . . .
438    . . .
439    . . .
440    . . .
441    . . .
442    . . .
443    . . .
444    . . .
445    . . .
446    . . .
447    . . .
448    . . .
449    . . .
450    . . .
451    . . .
452    . . .
453    . . .
454    . . .
455    . . .
456    . . .
457    . . .
458    . . .
459    . . .
460    . . .
461    . . .
462    . . .
463    . . .
464    . . .
465    . . .
466    . . .
467    . . .
468    . . .
469    . . .
470    . . .
471    . . .
472    . . .
473    . . .
474    . . .
475    . . .
476    . . .
477    . . .
478    . . .
479    . . .
480    . . .
481    . . .
482    . . .
483    . . .
484    . . .
485    . . .
486    . . .
487    . . .
488    . . .
489    . . .
490    . . .
491    . . .
492    . . .
493    . . .
494    . . .
495    . . .
496    . . .
497    . . .
498    . . .
499    . . .
500    . . .
501    . . .
502    . . .
503    . . .
504    . . .
505    . . .
506    . . .
507    . . .
508    . . .
509    . . .
510    . . .
511    . . .
512    . . .
513    . . .
514    . . .
515    . . .
516    . . .
517    . . .
518    . . .
519    . . .
520    . . .
521    . . .
522    . . .
523    . . .
524    . . .
525    . . .
526    . . .
527    . . .
528    . . .
529    . . .
530    . . .
531    . . .
532    . . .
533    . . .
534    . . .
535    . . .
536    . . .
537    . . .
538    . . .
539    . . .
540    . . .
541    . . .
542    . . .
543    . . .
544    . . .
545    . . .
546    . . .
547    . . .
548    . . .
549    . . .
550    . . .
551    . . .
552    . . .
553    . . .
554    . . .
555    . . .
556    . . .
557    . . .
558    . . .
559    . . .
560    . . .
561    . . .
562    . . .
563    . . .
564    . . .
565    . . .
566    . . .
567    . . .
568    . . .
569    . . .
570    . . .
571    . . .
572    . . .
573    . . .
574    . . .
575    . . .
576    . . .
577    . . .
578    . . .
579    . . .
580    . . .
581    . . .
582    . . .
583    . . .
584    . . .
585    . . .
586    . . .
587    . . .
588    . . .
589    . . .
590    . . .
591    . . .
592    . . .
593    . . .
594    . . .
595    . . .
596    . . .
597    . . .
598    . . .
599    . . .
600    . . .
601    . . .
602    . . .
603    . . .
604    . . .
605    . . .
606    . . .
607    . . .
608    . . .
609    . . .
610    . . .
611    . . .
612    . . .
613    . . .
614    . . .
615    . . .
616    . . .
617    . . .
618    . . .
619    . . .
620    . . .
621    . . .
622    . . .
623    . . .
624    . . .
625    . . .
626    . . .
627    . . .
628    . . .
629    . . .
630    . . .
631    . . .
632    . . .
633    . . .
634    . . .
635    . . .
636    . . .
637    . . .
638    . . .
639    . . .
640    . . .
641    . . .
642    . . .
643    . . .
644    . . .
645    . . .
646    . . .
647    . . .
648    . . .
649    . . .
650    . . .
651    . . .
652    . . .
653    . . .
654    . . .
655    . . .
656    . . .
657    . . .
658    . . .
659    . . .
660    . . .
661    . . .
662    . . .
663    . . .
664    . . .
665    . . .
666    . . .
667    . . .
668    . . .
669    . . .
670    . . .
671    . . .
672    . . .
673    . . .
674    . . .
675    . . .
676    . . .
677    . . .
678    . . .
679    . . .
680    . . .
681    . . .
682    . . .
683    . . .
684    . . .
685    . . .
686    . . .
687    . . .
688    . . .
689    . . .
690    . . .
691    . . .
692    . . .
693    . . .
694    . . .
695    . . .
696    . . .
697    . . .
698    . . .
699    . . .
700    . . .
701    . . .
702    . . .
703    . . .
704    . . .
705    . . .
706    . . .
707    . . .
708    . . .
709    . . .
710    . . .
711    . . .
712    . . .
713    . . .
714    . . .
715    . . .
716    . . .
717    . . .
718    . . .
719    . . .
720    . . .
721    . . .
722    . . .
723    . . .
724    . . .
725    . . .
726    . . .
727    . . .
728    . . .
729    . . .
730    . . .
731    . . .
732    . . .
733    . . .
734    . . .
735    . . .
736    . . .
737    . . .
738    . . .
739    . . .
740    . . .
741    . . .
742    . . .
743    . . .
744    . . .
745    . . .
746    . . .
747    . . .
748    . . .
749    . . .
750    . . .
751    . . .
752    . . .
753    . . .
754    . . .
755    . . .
756    . . .
757    . . .
758    . . .
759    . . .
760    . . .
761    . . .
762    . . .
763    . . .
764    . . .
765    . . .
766    . . .
767    . . .
768    . . .
769    . . .
770    . . .
771    . . .
772    . . .
773    . . .
774    . . .
775    . . .
776    . . .
777    . . .
778    . . .
779    . . .
780    . . .
781    . . .
782    . . .
783    . . .
784    . . .
785    . . .
786    . . .
787    . . .
788    . . .
789    . . .
790    . . .
791    . . .
792    . . .
793    . . .
794    . . .
795    . . .
796    . . .
797    . . .
798    . . .
799    . . .
800    . . .
801    . . .
802    . . .
803    . . .
804    . . .
805    . . .
806    . . .
807    . . .
808    . . .
809    . . .
810    . . .
811    . . .
812    . . .
813    . . .
814    . . .
815    . . .
816    . . .
817    . . .
818    . . .
819    . . .
820    . . .
821    . . .
822    . . .
823    . . .
824    . . .
825    . . .
826    . . .
827    . . .
828    . . .
829    . . .
830    . . .
831    . . .
832    . . .
833    . . .
834    . . .
835    . . .
836    . . .
837    . . .
838    . . .
839    . . .
840    . . .
841    . . .
842    . . .
843    . . .
844    . . .
845    . . .
846    . . .
847    . . .
848    . . .
849    . . .
850    . . .
851    . . .
852    . . .
853    . . .
854    . . .
855    . . .
856    . . .
857    . . .
858    . . .
859    . . .
860    . . .
861    . . .
862    . . .
863    . . .
864    . . .
865    . . .
866    . . .
867    . . .
868    . . .
869    . . .
870    . . .
871    . . .
872    . . .
873    . . .
874    . . .
875    . . .
876    . . .
877    . . .
878    . . .
879    . . .
880    . . .
881    . . .
882    . . .
883    . . .
884    . . .
885    . . .
886    . . .
887    . . .
888    . . .
889    . . .
890    . . .
891    . . .
892    . . .
893    . . .
894    . . .
895    . . .
896    . . .
897    . . .
898    . . .
899    . . .
900    . . .
901    . . .
902    . . .
903    . . .
904    . . .
905    . . .
906    . . .
907    . . .
908    . . .
909    . . .
910    . . .
911    . . .
912    . . .
913    . . .
914    . . .
915    . . .
916    . . .
917    . . .
918    . . .
919    . . .
920    . . .
921    . . .
922    . . .
923    . . .
924    . . .
925    . . .
926    . . .
927    . . .
928    . . .
929    . . .
930    . . .
931    . . .
932    . . .
933    . . .
934    . . .
935    . . .
936    . . .
937    . . .
938    . . .
939    . . .
940    . . .
941    . . .
942    . . .
943    . . .
944    . . .
945    . . .
946    . . .
947    . . .
948    . . .
949    . . .
950    . . .
951    . . .
952    . . .
953    . . .
954    . . .
955    . . .
956    . . .
957    . . .
958    . . .
959    . . .
960    . . .
961    . . .
962    . . .
963    . . .
964    . . .
965    . . .
966    . . .
967    . . .
968    . . .
969    . . .
970    . . .
971    . . .
972    . . .
973    . . .
974    . . .
975    . . .
976    . . .
977    . . .
978    . . .
979    . . .
980    . . .
981    . . .
982    . . .
983    . . .
984    . . .
985    . . .
986    . . .
987    . . .
988    . . .
989    . . .
990    . . .
991    . . .
992    . . .
993    . . .
994    . . .
995    . . .
996    . . .
997    . . .
998    . . .
999    . . .
1000    . . .
1001    . . .
1002    . . .
1003    . . .
1004    . . .
1005    . . .
1006    . . .
1007    . . .
1008    . . .
1009    . . .
1010    . . .
1011    . . .
1012    . . .
1013    . . .
1014    . . .
1015    . . .
1016    . . .
1017    . . .
1018    . . .
1019    . . .
1020    . . .
1021    . . .
1022    . . .
1023    . . .
1024    . . .
1025    . . .
1026    . . .
1027    . . .
1028    . . .
1029    . . .
1030    . . .
1031    . . .
1032    . . .
1033    . . .
1034    . . .
103
```

Implemented functions

```
54     # Feature 8: RSI (Relative Strength Index)
55     delta = close_prices.diff()
56     gain = delta.clip(lower=0)
57     loss = -delta.clip(upper=0)
58     avg_gain = gain.rolling(14).mean()
59     avg_loss = loss.rolling(14).mean()
60     rs = avg_gain / avg_loss
61     data['RSI_14'] = 100 - (100 / (1 + rs))
62
63     # Feature 9 & 10: MACD Indicator
64     ema_12 = close_prices.ewm(span=12, adjust=False).mean()
65     ema_26 = close_prices.ewm(span=26, adjust=False).mean()
66     data['MACD'] = ema_12 - ema_26
67     data['MACD_Signal'] = data['MACD'].ewm(span=9, adjust=False).mean()
68
69     return data.dropna()
70
71 #---- UI LAYOUT ---
72 st.title("📈 Microsoft (MSFT) Stock Price Predictor")
73 st.markdown("This app provides statistical predictions for the **Next Day Closing Price** using pre-trained ML models.")
74
75 bundle = load_bundle()
76
77 # Check if the bundle exists; if not, show an error message
78 if bundle is None:
79     st.error("Model bundle not found. Please run `training.py` first to generate `models_bundle.joblib` .")
80 else:
81     # Fetch the most recent live data for MSFT
82     latest_df = get_latest_data(TICKER)
83
84     if latest_df is not None:
85         # Extract the very last available row of data to use for the prediction
86         current_data = latest_df.iloc[[-1]]
87         # Format the date and price for display
88         last_date = current_data.index[0].strftime('%Y-%m-%d')
89         last_price = float(current_data['Close'].iloc[0])
90
91         # Create three columns for the summary metrics row
92         col1, col2, col3 = st.columns(3)
```



Implemented functions

```
94     with col1:
95         st.subheader("Market Status")
96         # Show the most recent closing price
97         st.metric("Last Known Close", f"${last_price:.2f}")
98         st.caption(f"Data as of: {last_date}")
99
100    with col2:
101        st.subheader("Model Selection")
102        # Allow the user to choose between different trained models (e.g., Random Forest, Linear Regression)
103        selected_model_name = st.selectbox("Choose Model", list(bundle['models'].keys()))
104
105    with col3:
106        st.subheader("Volatility Index")
107        # Show the most recent intraday price fluctuation
108        daily_range = float(current_data['Daily Range'].iloc[0])
109        st.metric("Intraday Range", f"${daily_range:.2f}")
110
111    # Prediction Logic
112    # 1. Get the exact list of features the model was trained on
113    features_to_use = bundle['features']
114    # 2. Extract those values from our current data row
115    X_raw = current_data[features_to_use].values
116    # 3. Apply the saved scaler to normalize the features
117    X_scaled = bundle['scaler'].transform(X_raw)
118
119    # 4. Use the selected model to predict the next closing price
120    model = bundle['models'][selected_model_name]
121    prediction_val = model.predict(X_scaled)
122    prediction = float(prediction_val[0])
123
124    # Add a visual horizontal line
125    st.divider()
126
127    # Display Prediction Statistics
128    st.subheader(f"Statistical Analysis: {selected_model_name}")
129    # Calculate the dollar difference and percentage change from the last known price
130    diff = prediction - last_price
131    percent = (diff / last_price) * 100
132
```





Implemented functions

```
132
133     # Create two columns for results
134     res_col1, res_col2 = st.columns(2)
135     with res_col1:
136         # Show the predicted price with a delta (green/red arrow)
137         st.metric(
138             label="Predicted Next Close",
139             value=f"${prediction:.2f}",
140             delta=f"{diff:+.2f} ({percent:+.2f}%)"
141         )
142
143     with res_col2:
144         # Determine market sentiment based on whether the prediction is up or down
145         sentiment = "Bullish" if diff > 0 else "Bearish"
146         st.info(f"Signal: **{sentiment}** | Expected change of **{percent:.2f}%**")
147
148         # Create an expandable section to let users inspect the raw math behind the prediction
149         with st.expander("View Raw Feature Data (Last 10 Days):"):
150             st.dataframe(latest_df.tail(10))
151
152     else:
153         # Show error if yfinance fails or returns nothing
154         st.error("Failed to fetch recent data from Yahoo Finance.")
155
156 # Add a permanent disclaimer in the sidebar
157 st.sidebar.info("Disclaimer: Statistical models are for informational purposes only. Trading involves significant risk.")
158
```

