Apple Stocks Prediction

In this project we will Build a model that predict the apple stock price for a period of time based on historical data and follow the trend of it to make the best predictions

Setup & Install

For this part we will import the the needed librares that we will use in this project

Then we have to make sure that we have TesorFlow for the LSTM (Long-Short-Term-Memory)

Model

```
import sys, subprocess, importlib, warnings, os, math, random
warnings.filterwarnings("ignore")

def ensure(pkg, import_name=None, extras=None):
    """Install `pkg` if missing, then import and return module."""
    name = import_name or pkg
    try:
        return importlib.import_module(name)
    except ImportError:
        to_install = pkg if not extras else f"{pkg}[{extras}]"
        print(f"[INFO] Installing {to_install} ...")
        subprocess.check_call([sys.executable, "-m", "pip", "install", "-q", to_install])
        return importlib.import_module(name)

np = ensure("numpy")
pd = ensure("pandas")
plt = ensure("matplotlib.pyplot", import_name="matplotlib.pyplot")
skmetrics = ensure("sklearn.metrics", import_name="sklearn.metrics")
MinMaxScaler = ensure("sklearn.preprocessing", import_name="sklearn.preprocessing").MinMaxScaler
train_test_split = ensure("sklearn.model_selection", import_name="sklearn.model_selection").train_test_split
yfinance = ensure("yfinance")
pmdarima = ensure("pmdarima")
auto_arima = pmdarima.arima.auto_arima

# TensorFlow (CPU) for LSTM
try:
    tf = importlib.import_module("tensorflow")
except ImportError:
    print("[INFO] Installing tensorflow (CPU) ...")
    subprocess.check_call([sys.executable, "-m", "pip", "install", "-q", "tensorflow"])
    tf = importlib.import_module("tensorflow")
from tensorflow import keras
from tensorflow.keras import layers

# Reproducibility
SEED = 42
random.seed(SEED); np.random.seed(SEED); tf.random.set_seed(SEED)
```

PART 1: Download Data

In this part we will download the data that contains the price and some features that will help us

The data source is Yahoo Finance which is a website that provide the prices of stocks for every company that have a stocks in the all world

I added a RuntimeError Rais to make sure that my code will not crash

We must set the data for 5 days a week cause the marked only works in the business days

We must drop the null data because it will effect the quality of the model

And we will only take the [Close,Volume] columns for our study will explain why later

This study takes 10 years of price change

```
import datetime as dt
END = dt.date.today()
START = dt.date(2015, 1, 1)

print("[INFO] Downloading AAPL data from Yahoo Finance ...")
df = yfinance.download("AAPL", start=str(START), end=str(END), progress=False)
if df is None or df.empty:
    raise RuntimeError("Failed to download AAPL data from Yahoo. Check your internet
connection.")
# Use Adjusted Close for modeling
df = df[['Close','Volume']].copy()
#df.rename(columns={'Adj Close':'AdjClose'}, inplace=True)
df.dropna(inplace=True)

# Convert to business day frequency and forward fill
df = df.asfreq('B') # Bussnuis Day
#df['AdjClose'] = df['AdjClose'].ffill()
df['Close'] = df['Close'].ffill()
df['Volume'] = df['Volume'].ffill()
```

PART 2: Feature Engineering

Ok lets go for the features and see what we can make with them and what we can extract

-df['Return1'] = df['Close'].pct_change() # Calculates the **daily return** (percentage change) of the closing price.

-df['RollMean5'] = df['Close'].rolling(5).mean()#Computes a **5-day rolling average** of the closing price.

-df['RollStd5'] = df['Close'].rolling(5).std()#Calculates the **5-day rolling standard deviation** of the closing price.

By the end of all above you will git a data looks like this that have a new columns we will need it for another use

Price	Close	Volume	Return1	RollMean5	RollStd5
Ticker	AAPL	AAPL			
Date					
2015-01-02	24.261044	212818400.0	-0.028172	24.031592	0.527647
2015-01-05	23.577572	257142000.0	-0.028172	24.031592	0.527647
2015-01-06	23.579788	263188400.0	0.000094	24.031592	0.527647
2015-01-07	23.910433	160423600.0	0.014022	24.031592	0.527647
2015-01-08	24.829124	237458000.0	0.038422	24.031592	0.527647

PART 3: Train/Test Split

This part will split the data into training and testing

The training: have [Return1,RollMean5,RollStd5]

The testing: have [Close] column

```
# Last ~120 business days as test (about ~6 months)
TEST_SIZE = 120
if len(df) <= TEST_SIZE + 200:
    TEST_SIZE = max(30, len(df)//5)  # fallback

target = df['Close'].copy()
exog = df[['Return1','RollMean5','RollStd5']].copy()

y_train, y_test = target.iloc[:-TEST_SIZE], target.iloc[-TEST_SIZE:]
X_train, X_test = exog.iloc[:-TEST_SIZE], exog.iloc[-TEST_SIZE:]

print(f"[INFO] Train length: {len(y_train)}, Test length:
{len(y_test)}")
```

PART 4) Metrics Helpers

We have 2 functions in this part of the code the first one is (mape) which is the **mean absolute percentage error** (**MAPE**), also known as **mean absolute percentage deviation** (**MAPD**), which is a measure of prediction accuracy of a forecasting method in statistics. It usually expresses accuracy as a ratio

And the second part is (evaluate) which have a 3 measures

- rmse
- mae
- mp

we have empty list for the results and make a new datafream called predictions_df for the test

ARIMA \SARIMA \SARIMAX Part (5,6,7)

Ok lets go for the big family of time series

1. ARIMA (AutoRegressive Integrated Moving Average)

ARIMA is a well-known statistical model that predicts future values in a time series based on past observations. It combines three components:

- **AutoRegressive (AR)**: Refers to using past values to predict future values.
- **Integrated (I)**: Helps make the time series stationary by differencing, which removes trends or seasonality.
- **Moving Average (MA)**: Refers to using past forecast errors to adjust future predictions.

2. SARIMA (Seasonal ARIMA)

SARIMA extends ARIMA by handling seasonality. In real-world data, seasonality (e.g., quarterly sales peaks) is often present, and SARIMA can capture these recurring patterns. SARIMA adds seasonal terms for AR, I, and MA.

3.SARIMAX (Seasonal ARIMA with Exogenous Variables)

SARIMAX builds on SARIMA by allowing **exogenous variables (X)**, meaning it includes external factors that can impact the time series. This is especially useful when factors like the rolling we added

Let's dive into each one of them

ARIMA

I had use auto_arima function which is a powerful tool for automatically identifying the best ARIMA (AutoRegressive Integrated Moving Average) model for a given time series dataset. It is part of the **pmdarima** library in Python and is widely used for time series forecasting. The function selects the optimal model by testing various combinations of parameters and evaluating them using criteria like AIC (Akaike Information Criterion), AICc, or BIC.

Key Features

- Automatically determines the values for **p**, **d**, and **q** (ARIMA parameters).
- Supports **seasonal ARIMA** (SARIMA) with parameters **P**, **D**, **Q**, and **m**.
- Includes differencing tests to determine the order of integration (**d**).
- Allows customization of search space and constraints.
- Handles exogenous variables (ARIMAX models).

SARIMA

The same function but includes a seasonal factor to the function

SARIMAX

Here where the plot twist is we will use the features we generated earlier to add a little bit of spice and make the training more realistic and more aware about what happens in another days

```
sarimax_model = auto_arima(
    y=y_train,
    X=X_train,
    seasonal=True, m=5,
    stepwise=True,
    error_action='ignore',
    suppress_warnings=True,
    trace=False,
    max_p=3, max_q=3,
max_P=2, max_Q=2   )
```

PART 8: LSTM

The LSTM is one of the most powerful Deep Learning models that not just only learns from the data but also remembers what happen in the past and effect it in the future.

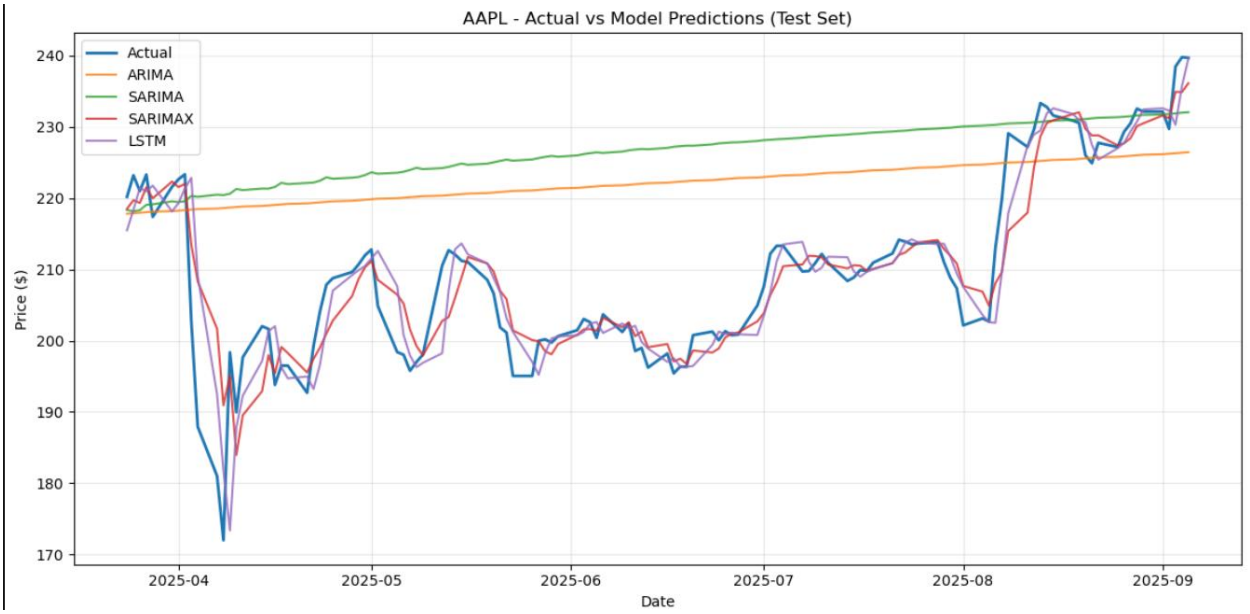
You can build an infinite number of layers but it's not always good to add a lot of layers sometimes one layer can give you what 10 layers can, so you have to try your luck or make one.

You have to chose the number of epochs you want and always remember to add an early stop function so if the things go away you have a backup plan

```
print("[INFO] Training LSTM ...")
try:
    # Scale target for LSTM
    scaler = MinMaxScaler()
    scaled_all = scaler.fit_transform(target.values.reshape(-1,1)).astype(np.float32)
    scaled_train = scaled_all[:-TEST_SIZE]

    # Ensure we have enough data for the window
    WINDOW = min(60, len(scaled_train) // 4)  # Adaptive window size
    scaled_test = scaled_all[-(TEST_SIZE + WINDOW):]  # include history for windowing

    def create_sequences(arr, window):
        Xs, ys = [], []
        for i in range(window, len(arr)):
            Xs.append(arr[i-window:i, 0])
            ys.append(arr[i, 0])
        return np.array(Xs), np.array(ys)

    X_train_seq, y_train_seq = create_sequences(scaled_train, WINDOW)
    X_test_seq,  y_test_seq  = create_sequences(scaled_test, WINDOW)

    if len(X_train_seq) == 0 or len(X_test_seq) == 0:
        raise ValueError("Insufficient data for LSTM windowing")

    # Reshape for LSTM: (samples, timesteps, features)
    X_train_seq = X_train_seq.reshape((X_train_seq.shape[0], X_train_seq.shape[1], 1))
    X_test_seq  = X_test_seq.reshape((X_test_seq.shape[0], X_test_seq.shape[1], 1))

    # Build a simple but solid LSTM model
    model = keras.Sequential([
        layers.Input(shape=(WINDOW, 1)),
        layers.LSTM(64, return_sequences=True),
        layers.Dropout(0.1),
        layers.LSTM(32),
        layers.Dense(16, activation='relu'),
        layers.Dense(1)
    ])

    model.compile(optimizer='adam', loss='mse')
    early = keras.callbacks.EarlyStopping(monitor='val_loss', patience=8,
restore_best_weights=True)
    history = model.fit(
        X_train_seq, y_train_seq,
        validation_split=0.15,
        epochs=80,
        batch_size=32,
        callbacks=[early],
        verbose=0
    )

    # Predict & invert scale
    lstm_pred_scaled = model.predict(X_test_seq, verbose=0).reshape(-1,1)
    lstm_pred = scaler.inverse_transform(lstm_pred_scaled).ravel()

    # Align LSTM predictions with test set
    # Take the last len(y_test) predictions to align with test period
    predictions_df['LSTM'] = lstm_pred[-len(y_test):]
    results.append(evaluate(y_test, predictions_df['LSTM'], "LSTM"))
    print("[INFO] LSTM model trained successfully")

except Exception as e:
    print(f"[WARNING] LSTM failed: {e}")
    predictions_df['LSTM'] = np.nan
```

PART 9: Results & Plots

Model \ Metrics	RMSE	MAE	MAPE%
SARIMAX	**4.994**	**3.336**	**6.4430**
LSTM	**5.2008**	**3.2007**	**6.7071**
SARIMA	**17.0114**	**14.8226**	**7.7441**
ARIMA	**20.3195**	**17.5953**	**9.1818**

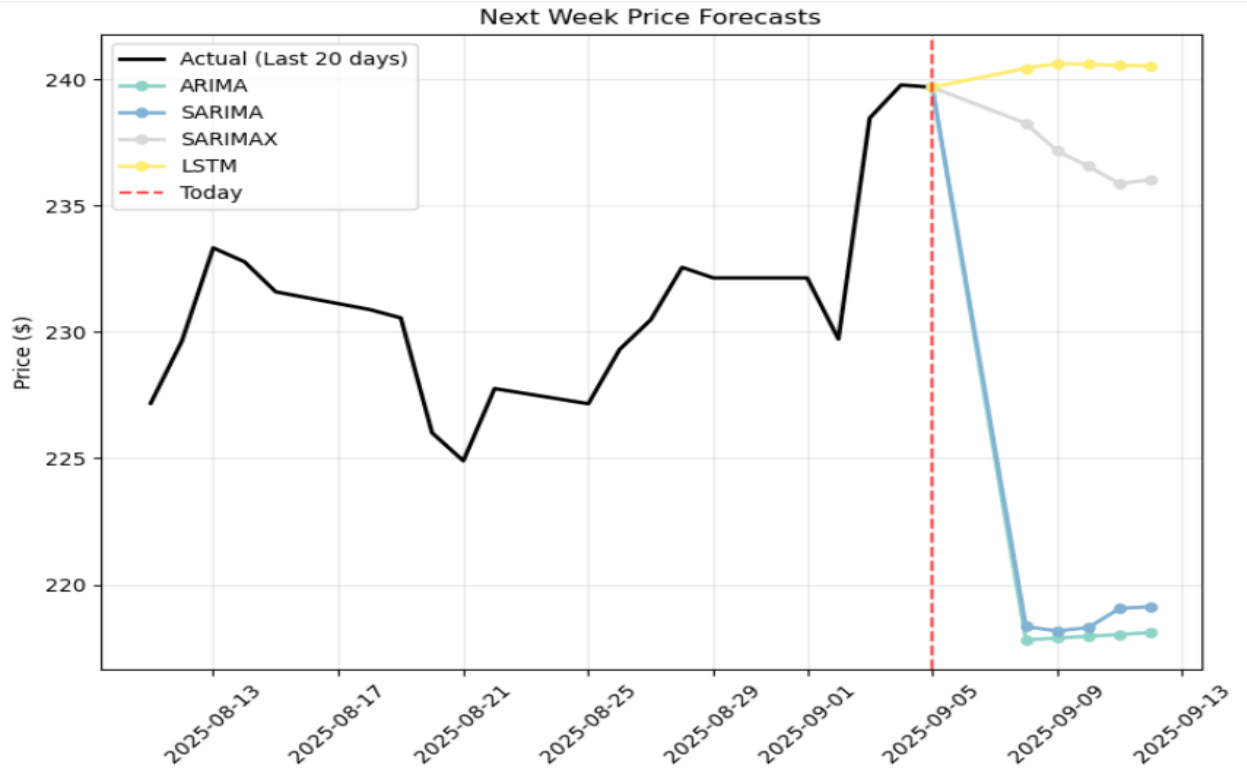


We can see that the [SARIMAX,LSTM] where the best models and give the best results for testing

but look here, the ARIMAX give us the best metrics output but that's don't necessarily be the same in forecasting

PART 10: Forecasts

We will forecast the stock price for one week ahead and see what our model will predict



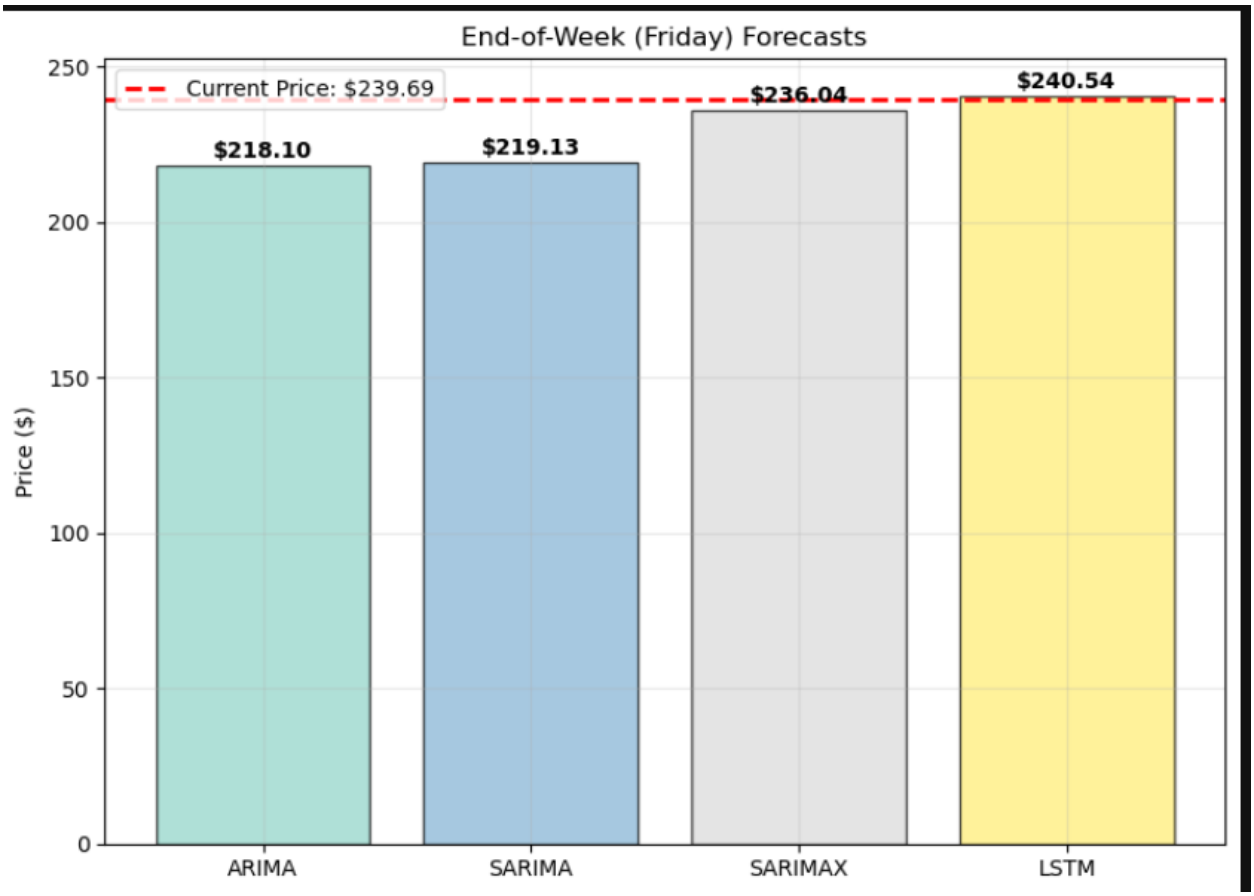
according to ARIMA and SARIMA the apple stock will go down and this not right and not realistic so we will not use them

but in the other hand the LSTM and SARIMAX gives us a realistic output and could happen in real life

Model Forecast Comparisons

- ARIMA (light blue):
    - Predicts a sharp decline in prices immediately after September 5.
    - Prices drop from around $240 to nearly $218, then remain flat with slight stabilization.
    - This suggests ARIMA overfits short-term fluctuations and fails to capture the strong upward trend.
- SARIMA (teal green):

    - Produces a similar downward trajectory to ARIMA, though slightly less extreme.
    - Also fails to maintain the rising momentum from recent data, suggesting it emphasizes seasonality but misinterprets the trend.
- SARIMAX (gray):
    - Forecasts a moderate decline after September 5, settling between $236–$237.
    - Unlike ARIMA/SARIMA, SARIMAX incorporates exogenous factors, which help it retain some stability, though still trending downward.
- LSTM (yellow):
    - Shows a continued upward movement and stabilization above $240.
    - This aligns more closely with the recent strong upward trend in the actual prices.
    - LSTM's ability to capture nonlinear patterns and recent momentum makes its forecast appear more realistic compared to ARIMA-family models.



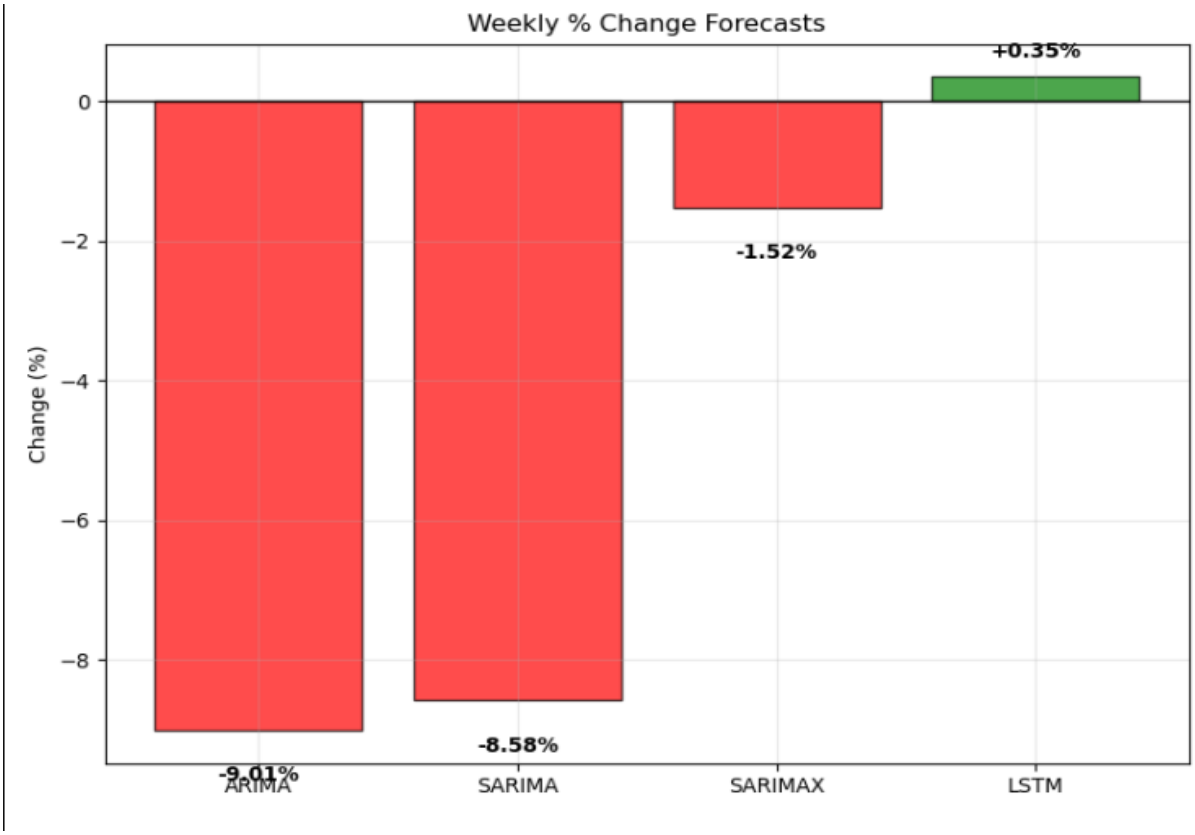
End-of-Week (Friday) Forecasts

Current price (Sep 5, 2025): $239.69 (red dashed line).

- **Forecasted Friday prices:**
  - **ARIMA: $218.10 (−9.0% vs current)**
  - **SARIMA: $219.13 (−8.6%)**
  - **SARIMAX: $236.04 (−1.5%)**
  - **LSTM: $240.54 (+0.35%)**

Insight:

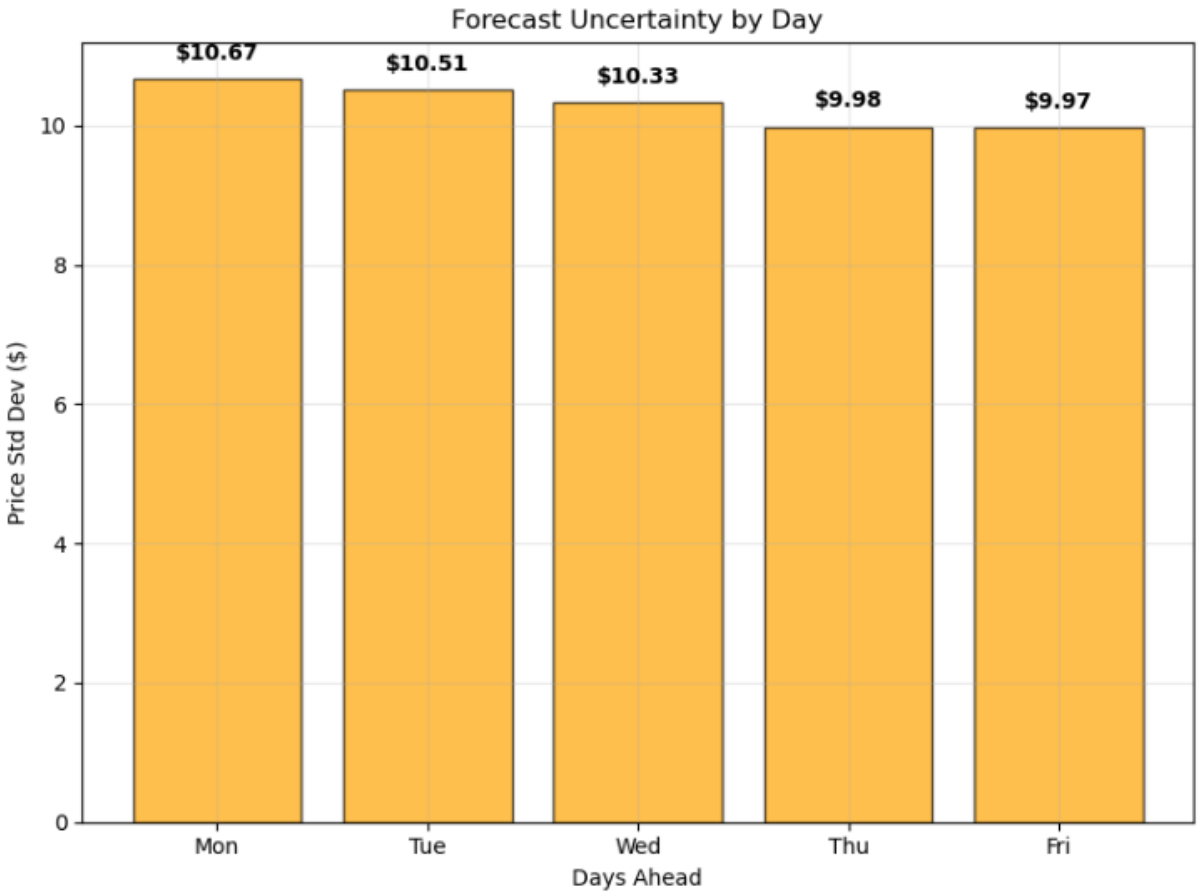
- **ARIMA & SARIMA expect a big correction (~−9%).**
- **SARIMAX is more conservative (−1.5%).**
- **LSTM is the only model forecasting growth, aligning with current bullish momentum.**



Weekly % Change Forecasts

- **ARIMA:** −9.01%
- **SARIMA:** −8.58%
- **SARIMAX:** −1.52%
- **LSTM:** +0.35%

Insight: Traditional statistical models strongly expect declines.
LSTM, in contrast, suggests **stability with slight growth**, which appears more realistic given recent price surges.



Forecast Uncertainty by Day

- Daily forecast **standard deviation (uncertainty):**
    - Monday: $10.67
    - Tuesday: $10.51
    - Wednesday: $10.33
    - Thursday: $9.98
    - Friday: $9.97

Insight: Uncertainty is **highest earlier in the week**, then narrows slightly towards Friday. This reflects typical forecasting behavior: as the horizon extends, models converge toward stable ranges.

Key Takeaways

1. **ARIMA & SARIMA** forecast sharp corrections that appear unrealistic given recent momentum.
2. **SARIMAX** moderates the decline but still biases downward.
3. **LSTM** is the only model projecting **continued stability/uptrend**, consistent with observed bullish behavior.
4. **Uncertainty remains high** ($10 average standard deviation), so forecasts should be interpreted cautiously.
5. For decision-making:
    - If you want **trend-following**, LSTM is the best choice.
    - If you want **risk-aware conservative planning**, SARIMAX provides a safer (though bearish) middle ground.