# Stock Market Forecasting using Time Series Models

This document contains Python code for forecasting stock market prices using time series models. It includes ARIMA, Prophet, and LSTM models, along with visualization and evaluation.

## 1. Data Collection

def fetch\_data(ticker="AAPL", start="2015-01-01", end="2023-12-31"):  
 data = yf.download(ticker, start=start, end=end)  
 data = data[["Close"]]  
 data.dropna(inplace=True)  
 data = data.asfreq('B') # Business days  
 data.fillna(method='ffill', inplace=True)  
 return data

## 2. Visualization

def plot\_data(data):  
 plt.figure(figsize=(14, 5))  
 plt.plot(data, label="Close Price")  
 plt.title("Stock Closing Price")  
 plt.xlabel("Date")  
 plt.ylabel("Price")  
 plt.legend()  
 plt.grid()  
 plt.show()

## 3. ARIMA Model

def arima\_model(data):  
 train = data[:-100]  
 test = data[-100:]  
 model = ARIMA(train, order=(5, 1, 0))  
 model\_fit = model.fit()  
 forecast = model\_fit.forecast(steps=100)  
 rmse = sqrt(mean\_squared\_error(test, forecast))  
 print(f"ARIMA RMSE: {rmse:.2f}")  
 plt.figure(figsize=(14, 5))  
 plt.plot(test.index, test.values, label="Actual")  
 plt.plot(test.index, forecast, label="Forecast")  
 plt.title("ARIMA Forecast")  
 plt.legend()  
 plt.show()

## 4. Prophet Model

def prophet\_model(data):  
 df = data.reset\_index()  
 df.columns = ['ds', 'y']  
 train = df[:-100]  
 test = df[-100:]  
 model = Prophet(daily\_seasonality=True)  
 model.fit(train)  
 future = model.make\_future\_dataframe(periods=100)  
 forecast = model.predict(future)  
 pred = forecast[['ds', 'yhat']].iloc[-100:]  
 rmse = sqrt(mean\_squared\_error(test['y'].values, pred['yhat'].values))  
 print(f"Prophet RMSE: {rmse:.2f}")  
 model.plot(forecast)  
 plt.title("Prophet Forecast")  
 plt.show()

## 5. LSTM Model

def lstm\_model(data):  
 from sklearn.preprocessing import MinMaxScaler  
 from tensorflow.keras.models import Sequential  
 from tensorflow.keras.layers import LSTM, Dense  
 from tensorflow.keras.optimizers import Adam  
  
 scaler = MinMaxScaler()  
 scaled\_data = scaler.fit\_transform(data)  
  
 X, y = [], []  
 for i in range(60, len(scaled\_data)):  
 X.append(scaled\_data[i - 60:i, 0])  
 y.append(scaled\_data[i, 0])  
 X, y = np.array(X), np.array(y)  
  
 X = X.reshape(X.shape[0], X.shape[1], 1)  
  
 model = Sequential()  
 model.add(LSTM(units=50, return\_sequences=True, input\_shape=(X.shape[1], 1)))  
 model.add(LSTM(units=50))  
 model.add(Dense(1))  
  
 model.compile(optimizer=Adam(learning\_rate=0.001), loss='mean\_squared\_error')  
 model.fit(X, y, epochs=5, batch\_size=32, verbose=0)  
  
 test\_data = scaled\_data[-160:]  
 X\_test = []  
 for i in range(60, len(test\_data)):  
 X\_test.append(test\_data[i - 60:i, 0])  
 X\_test = np.array(X\_test)  
 X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)  
  
 predictions = model.predict(X\_test)  
 predictions = scaler.inverse\_transform(predictions)  
 actual = data[-100:].values  
  
 rmse = sqrt(mean\_squared\_error(actual, predictions))  
 print(f"LSTM RMSE: {rmse:.2f}")  
 plt.figure(figsize=(14, 5))  
 plt.plot(actual, label="Actual")  
 plt.plot(predictions, label="LSTM Forecast")  
 plt.legend()  
 plt.title("LSTM Forecast")  
 plt.show()

## 6. Main Pipeline

def main():  
 data = fetch\_data("AAPL")  
 plot\_data(data)  
 print("\n--- ARIMA ---")  
 arima\_model(data)  
 print("\n--- Prophet ---")  
 prophet\_model(data)  
 print("\n--- LSTM ---")  
 lstm\_model(data)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()