

## ▼ Time Series Forecasting for Currency Exchange using Various Models

In this notebook, we will forecast the official exchange rate of Argentina using various models. We will use ARIMA, Prophet, XGBoost, LSTM, and GRU models to make predictions and compare their performance based on Mean Squared Error (MSE). The best model will be selected based on MSE and the predictions will be saved into an Excel file for further analysis.

```
!pip install Prophet
```

```
Requirement already satisfied: Prophet in /usr/local/lib/python3.11/dist-packages (1.1.7)
Requirement already satisfied: cmdstanpy>=1.0.4 in /usr/local/lib/python3.11/dist-packages (from Prophet) (1.2.5)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.11/dist-packages (from Prophet) (2.0.2)
Requirement already satisfied: matplotlib>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from Prophet) (3.10.0)
Requirement already satisfied: pandas>=1.0.4 in /usr/local/lib/python3.11/dist-packages (from Prophet) (2.2.2)
Requirement already satisfied: holidays<1,>=0.25 in /usr/local/lib/python3.11/dist-packages (from Prophet) (0.74)
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.11/dist-packages (from Prophet) (4.67.1)
Requirement already satisfied: importlib_resources in /usr/local/lib/python3.11/dist-packages (from Prophet) (6.5.2)
Requirement already satisfied: stadio<2.0.0,>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from cmdstanpy>=1.0.4->Prophet) (0.5)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.11/dist-packages (from holidays<1,>=0.25->Prophet) (2.9.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->Prophet) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->Prophet) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->Prophet) (4.58)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->Prophet) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->Prophet) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->Prophet) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.0.0->Prophet) (3.2.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.4->Prophet) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.4->Prophet) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil->holidays<1,>=0.25->Prophet)
```

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.arima.model import ARIMA
from prophet import Prophet
import xgboost as xgb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, GRU, Dense
from tensorflow.keras.callbacks import EarlyStopping

# Load the data
file_path = 'external_data.xlsx'
df = pd.read_excel(file_path, sheet_name='Data')

# Filter data for Argentina's exchange rate
df_argentina = df[df['Country Name'] == 'Argentina']
exchange_rate = df_argentina[df_argentina['Series Name'] == 'Official exchange rate (LCU per US$, period average)']
exchange_rate = exchange_rate.reset_index(drop=True)
exchange_rate = exchange_rate.interpolate(method='linear', axis=0)

# Select the relevant time series for exchange rate
exchange_rate_data = exchange_rate.iloc[0, 4:].values
years = exchange_rate.columns[4:]

# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(exchange_rate_data.reshape(-1, 1))

# Prepare data for training and testing
def create_dataset(data, time_step=1):
    X, y = [], []
    for i in range(len(data) - time_step - 1):
        X.append(data[i:(i + time_step), 0])
        y.append(data[i + time_step, 0])
    return np.array(X), np.array(y)

time_step = 10
X, y = create_dataset(scaled_data, time_step)
X = X.reshape(X.shape[0], X.shape[1], 1)
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

→ <ipython-input-4-84f83dbe5824>:9: FutureWarning: DataFrame.interpolate with object dtype is deprecated and will raise in a future version
exchange_rate = exchange_rate.interpolate(method='linear', axis=0)

years

→ Index(['2000 [YR2000]', '2001 [YR2001]', '2002 [YR2002]', '2003 [YR2003]',
       '2004 [YR2004]', '2005 [YR2005]', '2006 [YR2006]', '2007 [YR2007]',
       '2008 [YR2008]', '2009 [YR2009]', '2010 [YR2010]', '2011 [YR2011]',
       '2012 [YR2012]', '2013 [YR2013]', '2014 [YR2014]', '2015 [YR2015]',
       '2016 [YR2016]', '2017 [YR2017]', '2018 [YR2018]', '2019 [YR2019]',
       '2020 [YR2020]', '2021 [YR2021]', '2022 [YR2022]', '2023 [YR2023]',
       '2024 [YR2024]'],
      dtype='object')

# ARIMA Predictions
arima_model = ARIMA(scaled_data, order=(5, 1, 0))
arima_model_fit = arima_model.fit()
arima_pred = arima_model_fit.forecast(steps=len(y_test))
arima_pred = scaler.inverse_transform(arima_pred.reshape(-1, 1))

→ /usr/local/lib/python3.11/dist-packages/statsmodels/tsa/statespace/sarimax.py:966: UserWarning: Non-stationary starting autoregressive parameters
  warn('Non-stationary starting autoregressive parameters')
/usr/local/lib/python3.11/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to converge. See warning("Maximum Likelihood optimization failed to converge", ConvergenceWarning)
  warnings.warn("Maximum Likelihood optimization failed to converge", ConvergenceWarning)

# Clean the 'Year' column for Prophet compatibility
years_cleaned = [str(year).split(" ")[0] for year in years]

# Prepare the data for Prophet
prophet_df = pd.DataFrame({
    'ds': pd.to_datetime(years_cleaned), # Use cleaned year column
    'y': exchange_rate_data
})

# Fit the Prophet model
prophet_model = Prophet()
prophet_model.fit(prophet_df)

# Forecast for the next len(y_test) years
future = prophet_model.make_future_dataframe(periods=len(y_test), freq='Y')
prophet_forecast = prophet_model.predict(future)

# Get predictions
prophet_pred = prophet_forecast['yhat'][len(y_test):].values
prophet_pred = scaler.inverse_transform(prophet_pred.reshape(-1, 1))

→ INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
INFO:prophet:n_changepoints greater than number of observations. Using 19.
DEBUG:cmdstanpy:input tempfile: /tmp/tmp8kqxf4yr/yf90invp.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmp8kqxf4yr/5uurexrt.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=35942']
23:29:13 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
23:29:13 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
/usr/local/lib/python3.11/dist-packages/prophet/forecaster.py:1854: FutureWarning: 'Y' is deprecated and will be removed in a future version
dates = pd.date_range()

# XGBoost Predictions
xgboost_model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=1000, learning_rate=0.05)
xgboost_model.fit(X_train.reshape(X_train.shape[0], -1), y_train)
xgboost_pred = xgboost_model.predict(X_test.reshape(X_test.shape[0], -1))
xgboost_pred = scaler.inverse_transform(xgboost_pred.reshape(-1, 1))
```

```
# LSTM Predictions
lstm_model = Sequential()
lstm_model.add(LSTM(units=50, return_sequences=True, input_shape=(time_step, 1)))
lstm_model.add(LSTM(units=50, return_sequences=False))
lstm_model.add(Dense(units=1))
lstm_model.compile(optimizer='adam', loss='mean_squared_error')
lstm_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test), callbacks=[EarlyStopping(monitor='val_loss')])
lstm_pred = lstm_model.predict(X_test)
lstm_pred = scaler.inverse_transform(lstm_pred)
```

Epoch 1/50  
 /usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to `LSTM`.  
 super().\_\_init\_\_(\*\*kwargs)  
 1/1 2s 2s/step - loss: 9.0942e-04 - val\_loss: 0.0393  
 Epoch 2/50  
 1/1 0s 119ms/step - loss: 5.3447e-04 - val\_loss: 0.0347  
 Epoch 3/50  
 1/1 0s 64ms/step - loss: 5.4120e-04 - val\_loss: 0.0327  
 Epoch 4/50  
 1/1 0s 66ms/step - loss: 6.4427e-04 - val\_loss: 0.0328  
 Epoch 5/50  
 1/1 0s 70ms/step - loss: 6.1791e-04 - val\_loss: 0.0340  
 Epoch 6/50  
 1/1 0s 69ms/step - loss: 5.3768e-04 - val\_loss: 0.0355  
 Epoch 7/50  
 1/1 0s 74ms/step - loss: 4.8828e-04 - val\_loss: 0.0369  
 Epoch 8/50  
 1/1 0s 68ms/step - loss: 4.8956e-04 - val\_loss: 0.0379  
 1/1 0s 192ms/step

```
# GRU Predictions
gru_model = Sequential()
gru_model.add(GRU(units=50, return_sequences=True, input_shape=(time_step, 1)))
gru_model.add(GRU(units=50, return_sequences=False))
gru_model.add(Dense(units=1))
gru_model.compile(optimizer='adam', loss='mean_squared_error')
gru_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test), callbacks=[EarlyStopping(monitor='val_loss')])
gru_pred = gru_model.predict(X_test)
gru_pred = scaler.inverse_transform(gru_pred)
```

Epoch 1/50  
 /usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to `GRU`.  
 super().\_\_init\_\_(\*\*kwargs)  
 1/1 5s 5s/step - loss: 8.9236e-04 - val\_loss: 0.0329  
 Epoch 2/50  
 1/1 0s 190ms/step - loss: 5.6304e-04 - val\_loss: 0.0298  
 Epoch 3/50  
 1/1 0s 201ms/step - loss: 7.5609e-04 - val\_loss: 0.0314  
 Epoch 4/50  
 1/1 0s 207ms/step - loss: 5.6082e-04 - val\_loss: 0.0342  
 Epoch 5/50  
 1/1 0s 215ms/step - loss: 4.5324e-04 - val\_loss: 0.0364  
 Epoch 6/50  
 1/1 0s 248ms/step - loss: 5.1792e-04 - val\_loss: 0.0368  
 Epoch 7/50  
 1/1 0s 121ms/step - loss: 5.7002e-04 - val\_loss: 0.0355  
 1/1 0s 240ms/step

```
# Calculate MSE for each model
from sklearn.metrics import mean_squared_error

mse_arima = mean_squared_error(y_test, arima_pred)
mse_prophet = mean_squared_error(y_test, prophet_pred)
mse_xgboost = mean_squared_error(y_test, xgboost_pred)
mse_lstm = mean_squared_error(y_test, lstm_pred)
mse_gru = mean_squared_error(y_test, gru_pred)

# Store the MSE results in a dictionary
mse_results = {
    "ARIMA": mse_arima,
    "Prophet": mse_prophet,
    "XGBoost": mse_xgboost,
    "LSTM": mse_lstm,
    "GRU": mse_gru
}

# Find the model with the lowest MSE
best_model_name = min(mse_results, key=mse_results.get)
```

```
mse_results
```

```
→ { 'ARIMA': 41924262.48570872,
    'Prophet': 134644337208.53316,
    'XGBoost': 4786.512993004125,
    'LSTM': 1097.600446904106,
    'GRU': 1703.2043835880652}
```

```
best_model_name
```

```
# prompt: transforme years em int o formato dele esta assim 2003 [YR2003]
```

```
# Convert years to integer format
```

```
years_int = [int(str(year).split(" ")[0]) for year in years]
```

```
# Function to predict next 3 years using each model
```

```
def predict_next_3_years(model_name, model, scaled_data, scaler, time_step=10, prophet_df=None):
    if model_name == 'ARIMA':
```

```
        # For ARIMA, we refit on the whole scaled data and forecast
        arima_model_full = ARIMA(scaled_data, order=(5, 1, 0))
        arima_model_fit_full = arima_model_full.fit()
        forecast = arima_model_fit_full.forecast(steps=3)
        forecast = scaler.inverse_transform(forecast.reshape(-1, 1)).flatten()
```

```
    elif model_name == 'Prophet':
```

```
        # Prophet uses its own forecasting method
        future = model.make_future_dataframe(periods=3, freq='Y')
        forecast = model.predict(future)
        forecast = forecast['yhat'].tail(3).values
```

```
    elif model_name == 'XGBoost':
```

```
        # XGBoost requires creating future sequences
        last_sequence = scaled_data[-time_step:].reshape(1, -1)
        forecast = []
        current_input = last_sequence.copy()
        for _ in range(3):
            next_pred_scaled = model.predict(current_input)
            forecast.append(next_pred_scaled[0])
            # Update the input sequence by removing the oldest and adding the newest prediction
            current_input = np.roll(current_input, -1, axis=1)
            current_input[0, -1] = next_pred_scaled[0]
        forecast = scaler.inverse_transform(np.array(forecast).reshape(-1, 1)).flatten()
```

```
    elif model_name in ['LSTM', 'GRU']:
```

```
        # LSTM and GRU require creating future sequences
        last_sequence = scaled_data[-time_step:].reshape(1, time_step, 1)
        forecast = []
        current_input = last_sequence.copy()
        for _ in range(3):
            next_pred_scaled = model.predict(current_input)
            forecast.append(next_pred_scaled[0, 0])
            # Update the input sequence by removing the oldest and adding the newest prediction
            current_input = np.roll(current_input, -1, axis=1)
            current_input[0, -1, 0] = next_pred_scaled[0, 0]
        forecast = scaler.inverse_transform(np.array(forecast).reshape(-1, 1)).flatten()
```

```
    return forecast
```

```
# Get forecasts for the next 3 years from each model
```

```
arima_future_pred = predict_next_3_years('ARIMA', arima_model_fit, scaled_data, scaler)
```

```
prophet_future_pred = predict_next_3_years('Prophet', prophet_model, scaled_data, scaler, prophet_df=prophet_df)
```

```
xgboost_future_pred = predict_next_3_years('XGBoost', xgboost_model, scaled_data, scaler, time_step=time_step)
```

```
lstm_future_pred = predict_next_3_years('LSTM', lstm_model, scaled_data, scaler, time_step=time_step)
```

```
gru_future_pred = predict_next_3_years('GRU', gru_model, scaled_data, scaler, time_step=time_step)
```

```
# Combine forecasts with weights based on inverse MSE
```

```
# Calculate weights as inverse of MSE
```

```
total_inverse_mse = sum(1/mse for mse in mse_results.values())
```

```
weights = {model_name: (1/mse) / total_inverse_mse for model_name, mse in mse_results.items()}
```

```
# Weighted average forecast for the next 3 years
```

```
weighted_forecast = (weights['ARIMA'] * arima_future_pred +
                     weights['Prophet'] * prophet_future_pred +
                     weights['XGBoost'] * xgboost_future_pred +
                     weights['LSTM'] * lstm_future_pred +
                     weights['GRU'] * gru_future_pred)
```

```

weights['GRU'] * gru_future_pred)

# Create a DataFrame for the results
forecast_years = [int(years_int[-1]) + i + 1 for i in range(3)]
forecast_df = pd.DataFrame({
    'Year': forecast_years,
    'ARIMA Forecast': arima_future_pred,
    'Prophet Forecast': prophet_future_pred,
    'XGBoost Forecast': xgboost_future_pred,
    'LSTM Forecast': lstm_future_pred,
    'GRU Forecast': gru_future_pred,
    'Weighted Average Forecast': weighted_forecast,
    'Comments': [f"Weighted average forecast based on inverse MSE weights: {weights}" for _ in range(3)]
})

# Prepare historical data to append
historical_df = pd.DataFrame({
    'Year': years_int,
    'Exchange Rate': exchange_rate_data,
    'ARIMA Forecast': np.nan,
    'Prophet Forecast': np.nan,
    'XGBoost Forecast': np.nan,
    'LSTM Forecast': np.nan,
    'GRU Forecast': np.nan,
    'Weighted Average Forecast': np.nan,
    'Comments': 'Historical data'
})

# Concatenate historical data and forecast data
final_df = pd.concat([historical_df, forecast_df], ignore_index=True)

# Rename 'Exchange Rate' column to be consistent or keep it separate
final_df.rename(columns={'Exchange Rate': 'Historical Exchange Rate'}, inplace=True)

# Save the results to an Excel file
final_df.to_excel('exchange_rate_forecast_with_history.xlsx', index=False)

```

→ /usr/local/lib/python3.11/dist-packages/statsmodels/tsa/statespace/sarimax.py:966: UserWarning: Non-stationary starting autoregressive parameters  
warn('Non-stationary starting autoregressive parameters'  
1/1 ━━━━━━ 0s 37ms/step  
/usr/local/lib/python3.11/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to  
warnings.warn("Maximum Likelihood optimization failed to "  
/usr/local/lib/python3.11/dist-packages/prophet/forecaster.py:1854: FutureWarning: 'Y' is deprecated and will be removed in a future  
dates = pd.date\_range(  
1/1 ━━━━━━ 0s 36ms/step  
1/1 ━━━━━━ 0s 41ms/step  
1/1 ━━━━━━ 0s 39ms/step  
1/1 ━━━━━━ 0s 28ms/step  
1/1 ━━━━━━ 0s 30ms/step  
<ipython-input-56-6a93e6757dae>:88: FutureWarning: The behavior of DataFrame concatenation with empty or all-NA entries is deprecated  
final\_df = pd.concat([historical\_df, forecast\_df], ignore\_index=True)

**The forecasts do not perform well because they are not capable of capturing sudden, non-linear changes in the exchange rate. This is a common limitation of many forecasting models,**

- ✓ **which tend to work better with stable or predictable trends. To improve forecasting accuracy, models need to be adjusted to account for these types of shocks, or additional external factors should be incorporated into the forecasting process.**

final\_df

	Year	Historical Exchange Rate	ARIMA Forecast	Prophet Forecast	XGBoost Forecast	LSTM Forecast	GRU Forecast	Weighted Average Forecast	Comments
0	2000	0.9995	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
1	2001	0.9995	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
2	2002	3.063257	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
3	2003	2.900629	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
4	2004	2.923301	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
5	2005	2.903658	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
6	2006	3.054313	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
7	2007	3.095649	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
8	2008	3.144165	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
9	2009	3.710107	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
10	2010	3.896295	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
11	2011	4.11014	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
12	2012	4.536934	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
13	2013	5.459353	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
14	2014	8.075276	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
15	2015	9.233186	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
16	2016	14.758175	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
17	2017	16.562707	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
18	2018	28.094992	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
19	2019	48.147892	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
20	2020	70.539167	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
21	2021	94.990742	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
22	2022	130.61655	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
23	2023	296.258042	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
24	2024	914.694883	NaN	NaN	NaN	NaN	NaN	NaN	Historical data
25	2025	NaN	2438.790586	335.720148	69.373856	45.924629	80.824188	60.830533	Weighted average forecast based on inverse MSE...

Start coding or [generate](#) with AI.