Import Libraries

In this cell, we import all the Python libraries we will need. These include tools to handle data (Pandas, NumPy), make charts (Matplotlib, Seaborn), scale values (MinMaxScaler), build models (TensorFlow, Keras), and measure accuracy (mean_squared_error). These libraries make it easy to clean data, build deep learning models, and show results in charts.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, GRU, Dense

%matplotlib inline
pd.set_option("display.max_columns", None)
```

Load and Inspect Dataset:

We load the exchange rate dataset using Pandas and check the shape (rows and columns) of the data. Then we use .head() to view the first few rows. This helps us quickly understand what the data looks like and what kind of values it contains.

Check Column Names and Nulls:

This cell prints all the column names in the dataset and shows how many missing values (NaNs) are in each column. This step helps us find which parts of the data may need to be cleaned before we can use it.

```
df = pd.read_csv("/content/Foreign_Exchange_Rates.csv")

# Show shape and sample
print("Shape:", df.shape)
df.head()
```

```
Shape: (5217, 24)
```

	Unnamed: 0	Time Serie	AUSTRALIA - AUSTRALIAN DOLLAR/US$	EURO AREA - EURO/US$	NEW ZEALAND - NEW ZELAND DOLLAR/US$	UNITED KINGDOM - UNITED KINGDOM POUND/US$	BRAZIL - REAL/US$	CANADA - CANADIAN DOLLAR/US$	CHINA - YUAN/US$	HONG KONG - HONG KONG DOLLAR/US$	INDIA - INDIAN RUPEE/US$	KOREA - WON/US$	MEXICO - MEXICAN PESO/US$
0	0	2000-01-03	1.5172	0.9847	1.9033	0.6146	1.805	1.4465	8.2798	7.7765	43.55	1128	9.4015
1	1	2000-01-04	1.5239	0.97	1.9238	0.6109	1.8405	1.4518	8.2799	7.7775	43.55	1122.5	9.457
2	2	2000-01-05	1.5267	0.9676	1.9339	0.6092	1.856	1.4518	8.2798	7.778	43.55	1135	9.535
3	3	2000-01-06	1.5291	0.9686	1.9436	0.607	1.84	1.4571	8.2797	7.7785	43.55	1146.5	9.567
4	4	2000-01-07	1.5272	0.9714	1.938	0.6104	1.831	1.4505	8.2794	7.7783	43.55	1138	9.52

Rename and Convert Columns:

Here, we rename the date column and make it the index of the dataset. We also convert all exchange rate values to numeric form. If there are any errors during conversion, we turn them into NaN values so we can handle them later.

```
print(df.columns)
df.isnull().sum()
```

```
Index(['Unnamed: 0', 'Time Serie', 'AUSTRALIA - AUSTRALIAN DOLLAR/US$',
       'EURO AREA - EURO/US$', 'NEW ZEALAND - NEW ZELAND DOLLAR/US$',
       'UNITED KINGDOM - UNITED KINGDOM POUND/US$', 'BRAZIL - REAL/US$',
       'CANADA - CANADIAN DOLLAR/US$', 'CHINA - YUAN/US$',
       'HONG KONG - HONG KONG DOLLAR/US$', 'INDIA - INDIAN RUPEE/US$',
       'KOREA - WON/US$', 'MEXICO - MEXICAN PESO/US$',
       'SOUTH AFRICA - RAND/US$', 'SINGAPORE - SINGAPORE DOLLAR/US$',
       'DENMARK - DANISH KRONE/US$', 'JAPAN - YEN/US$',
       'MALAYSIA - RINGGIT/US$', 'NORWAY - NORWEGIAN KRONE/US$',
       'SWEDEN - KRONA/US$', 'SRI LANKA - SRI LANKAN RUPEE/US$',
       'SWITZERLAND - FRANC/US$', 'TAIWAN - NEW TAIWAN DOLLAR/US$',
       'THAILAND - BAHT/US$'],
      dtype='object')
```

	0
Unnamed: 0	0
Time Serie	0
AUSTRALIA - AUSTRALIAN DOLLAR/US$	0
EURO AREA - EURO/US$	0
NEW ZEALAND - NEW ZELAND DOLLAR/US$	0
UNITED KINGDOM - UNITED KINGDOM POUND/US$	0
BRAZIL - REAL/US$	0
CANADA - CANADIAN DOLLAR/US$	0
CHINA - YUAN/US$	0
HONG KONG - HONG KONG DOLLAR/US$	0
INDIA - INDIAN RUPEE/US$	0
KOREA - WON/US$	0
MEXICO - MEXICAN PESO/US$	0
SOUTH AFRICA - RAND/US$	0
SINGAPORE - SINGAPORE DOLLAR/US$	0
DENMARK - DANISH KRONE/US$	0
JAPAN - YEN/US$	0
MALAYSIA - RINGGIT/US$	0
NORWAY - NORWEGIAN KRONE/US$	0
SWEDEN - KRONA/US$	0
SRI LANKA - SRI LANKAN RUPEE/US$	0
SWITZERLAND - FRANC/US$	0
TAIWAN - NEW TAIWAN DOLLAR/US$	0
THAILAND - BAHT/US$	0

dtype: int64

Visualize Missing Data:

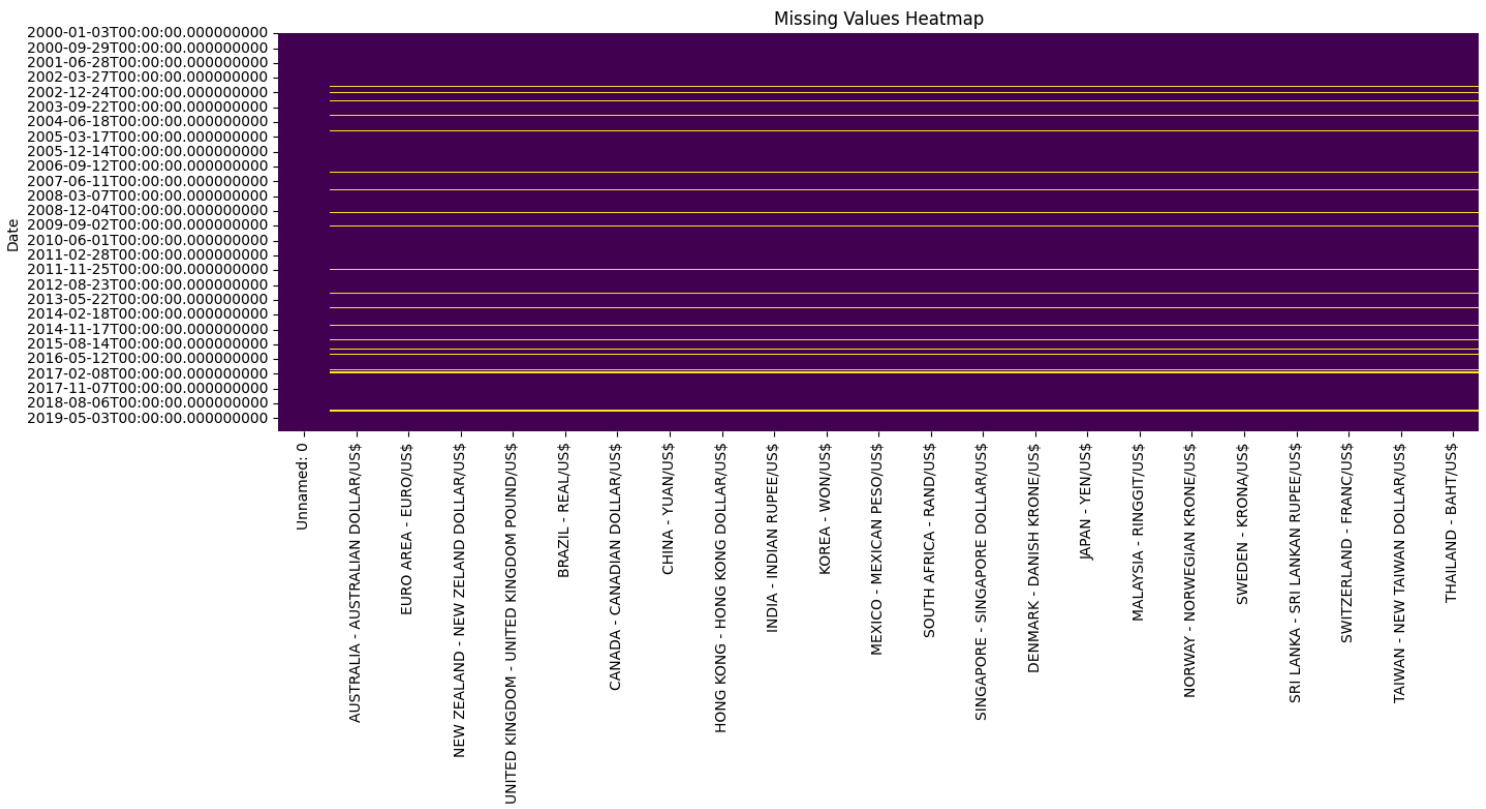
This code creates a heatmap to show missing values in the dataset. Each yellow/bright line represents missing data. It's a visual way to spot which rows or columns have too many missing entries that might affect the model.

```
df.rename(columns={"Time Serie": "Date"}, inplace=True)
df["Date"] = pd.to_datetime(df["Date"])
df.set_index("Date", inplace=True)

# Convert all currency columns to numeric
df = df.apply(pd.to_numeric, errors="coerce")

# Drop rows where all currencies are missing
df.dropna(how="all", inplace=True)
```

Drop Columns with >40% Missing Values:

This cell removes any column that has more than 40% missing data. We do this because too many missing values make the data unreliable. After this, we print the names of the columns that are still in the dataset.

```
plt.figure(figsize=(15, 5))
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title("Missing Values Heatmap")
plt.show()
```



Interpolate Remaining Missing Values

We use interpolation to fill in the remaining missing values. It guesses the missing numbers based on nearby values. This helps keep the dataset complete so we can use it in our model without errors.

```
threshold = 0.4 * len(df)
df = df.dropna(thresh=threshold, axis=1)
print("Remaining columns:", df.columns)
```

```
Remaining columns: Index(['Unnamed: 0', 'AUSTRALIA - AUSTRALIAN DOLLAR/US$',
       'EURO AREA - EURO/US$', 'NEW ZEALAND - NEW ZELAND DOLLAR/US$',
       'UNITED KINGDOM - UNITED KINGDOM POUND/US$', 'BRAZIL - REAL/US$',
       'CANADA - CANADIAN DOLLAR/US$', 'CHINA - YUAN/US$',
       'HONG KONG - HONG KONG DOLLAR/US$', 'INDIA - INDIAN RUPEE/US$',
       'KOREA - WON/US$', 'MEXICO - MEXICAN PESO/US$',
       'SOUTH AFRICA - RAND/US$', 'SINGAPORE - SINGAPORE DOLLAR/US$',
       'DENMARK - DANISH KRONE/US$', 'JAPAN - YEN/US$',
       'MALAYSIA - RINGGIT/US$', 'NORWAY - NORWEGIAN KRONE/US$',
       'SWEDEN - KRONA/US$', 'SRI LANKA - SRI LANKAN RUPEE/US$',
       'SWITZERLAND - FRANC/US$', 'TAIWAN - NEW TAIWAN DOLLAR/US$',
       'THAILAND - BAHT/US$'],
      dtype='object')
```

```
df.interpolate(method='linear', inplace=True)
print("Remaining missing values:", df.isnull().sum().sum())
```

```
Remaining missing values: 0
```

Statistical Summary

This cell shows statistics like mean, max, min, and standard deviation for each currency column. It helps us understand the range and average values of the exchange rates.

```
df.describe()
```

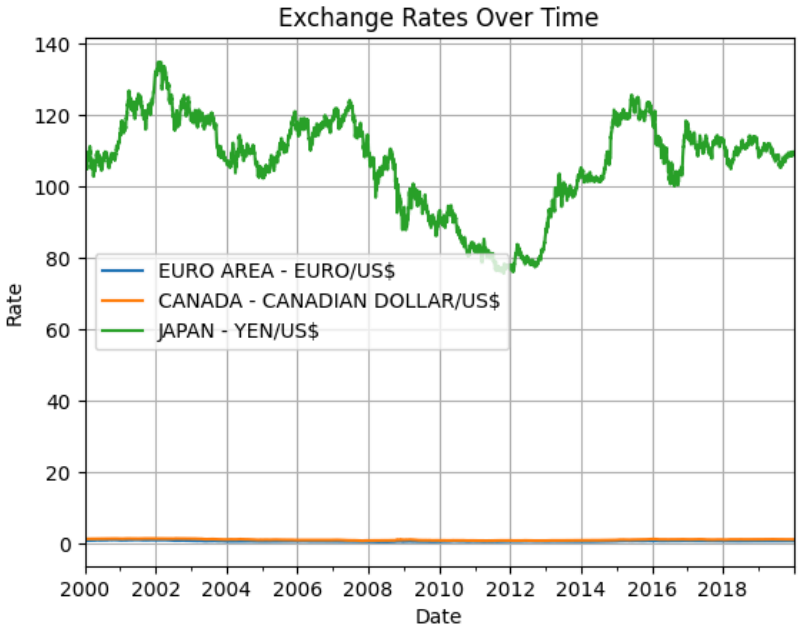
	Unnamed: 0	AUSTRALIA - AUSTRALIAN DOLLAR/US$	EURO AREA - EURO/US$	NEW ZEALAND - NEW ZELAND DOLLAR/US$	UNITED KINGDOM - UNITED KINGDOM POUND/US$	BRAZIL - REAL/US$	CANADA - CANADIAN DOLLAR/US$	CHINA - YUAN/US$	HONG KONG - HONG KONG DOLLAR/US$	INDIA - INDIAN RUPEE/US$	
count	5217.000000	5217.000000	5217.000000	5217.000000	5217.000000	5217.000000	5217.000000	5217.000000	5217.000000	5217.000000	5217
mean	2608.000000	1.332292	0.844075	1.543670	0.640740	2.550725	1.230582	7.199296	7.782607	52.762012	1125
std	1506.162508	0.269807	0.126697	0.337136	0.082644	0.725450	0.182145	0.819867	0.027556	9.695960	103
min	0.000000	0.906900	0.624600	1.134600	0.473800	1.537500	0.916800	6.040200	7.708500	38.480000	903
25%	1304.000000	1.115600	0.751200	1.323800	0.587900	1.946000	1.056000	6.475800	7.756350	45.270000	1067
50%	2608.000000	1.311500	0.815800	1.442600	0.636600	2.331000	1.237100	6.859700	7.780500	48.110000	1127
75%	3912.000000	1.430600	0.900300	1.590700	0.692500	3.132000	1.335700	8.276500	7.799800	62.470000	1180
max	5216.000000	2.071300	1.209200	2.551000	0.828700	4.259400	1.612800	8.280000	7.849900	74.330000	1570

Visualize a Few Exchange Rates:

We plot a line chart of a few exchange rates over time. This gives us a visual idea of how the exchange rates changed over the years. It helps us see trends and patterns in the data.

```
plt.figure(figsize=(15, 6))
df[["EURO AREA - EURO/US$", "CANADA - CANADIAN DOLLAR/US$", "JAPAN - YEN/US$"]].plot()
plt.title("Exchange Rates Over Time")
plt.ylabel("Rate")
plt.grid()
plt.show()
```

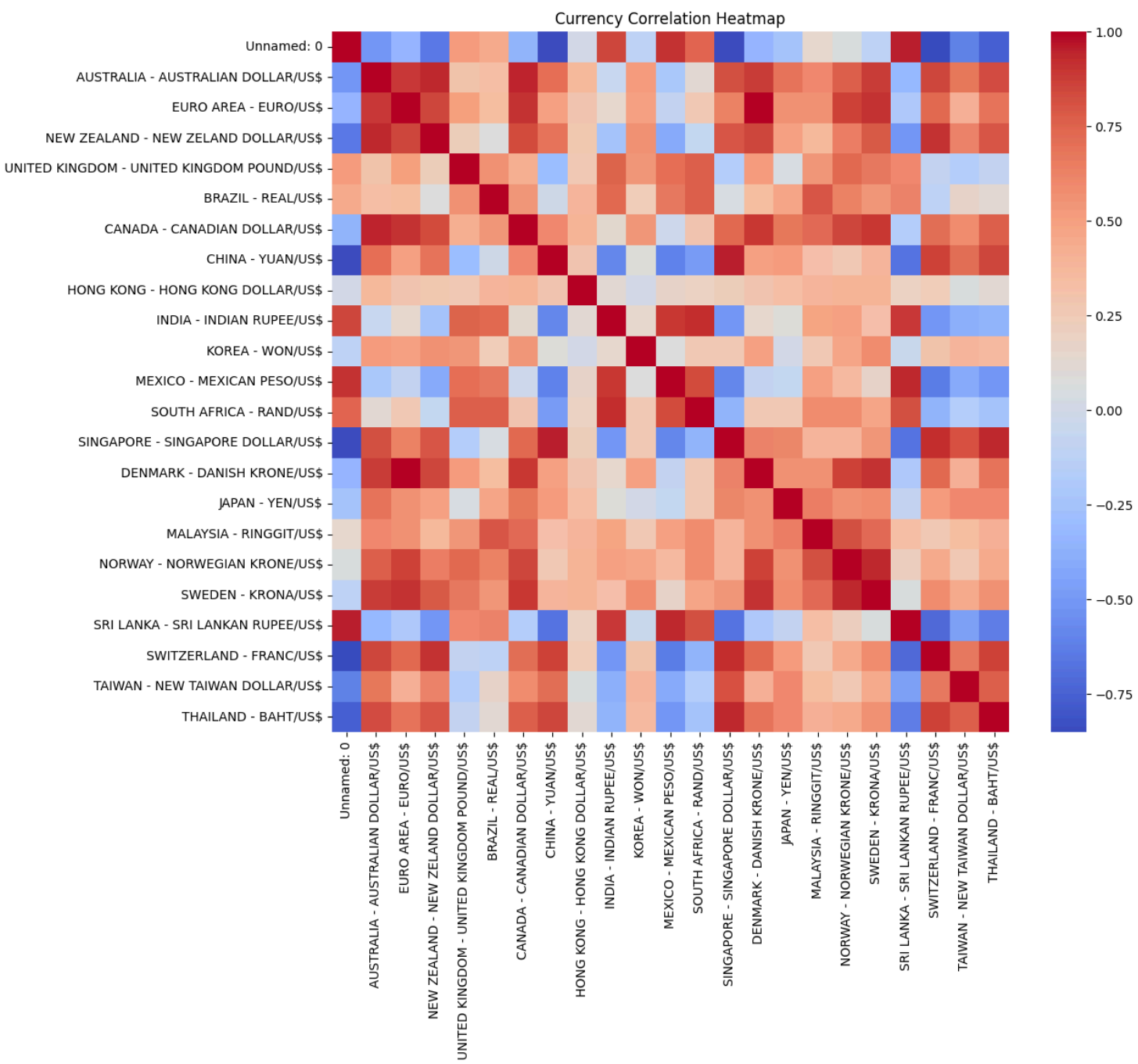
```
<Figure size 1500x600 with 0 Axes>
```



Correlation Heatmap:

This heatmap shows how different currency rates are related to each other. A higher correlation means the rates move together. It helps us know if some currencies behave similarly, which might be useful in prediction.

```
plt.figure(figsize=(12,10))
sns.heatmap(df.corr(), annot=False, cmap='coolwarm')
plt.title("Currency Correlation Heatmap")
plt.show()
```



Select EUR/USD and Normalize

Here, we focus only on the EUR to USD exchange rate. Then we use a scaler to convert the values into a range between 0 and 1. This helps the deep learning models train more easily and faster.

```
data = df[["EURO AREA - EURO/US$"]].copy()
data.columns = ["EUR_USD"]

scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)
```

Create Sequences:

We split the data into sequences of 60 days as input and the next day as the output. This helps the model learn from past values. Each sequence is stored in arrays X (inputs) and y (outputs).

```
X, y = [], []
seq_len = 60
```

```
for i in range(seq_len, len(scaled_data)):
    X.append(scaled_data[i-seq_len:i])
    y.append(scaled_data[i])

X = np.array(X)
y = np.array(y)

# Reshape for LSTM/GRU
X = X.reshape(X.shape[0], X.shape[1], 1)
```

Train-Test Split:

We split the data into training and testing sets. The training set helps the model learn patterns, and the testing set checks if the model can make good predictions on new, unseen data.

```
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]
```

Build and Train LSTM:

We build a model using LSTM (Long Short-Term Memory) and train it on the training data. LSTM is good at learning patterns over long time periods. The model learns to predict the EUR/USD rate based on past data.

```
lstm_model = Sequential()
lstm_model.add(LSTM(50, return_sequences=False, input_shape=(X_train.shape[1], 1)))
lstm_model.add(Dense(1))
lstm_model.compile(optimizer='adam', loss='mse')

lstm_history = lstm_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test), verbose=1)
```

```
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` argumer
  super().__init__(**kwargs)
129/129 ━━━━━━━━━━━━━━━━━━━━ 4s 20ms/step - loss: 0.0167 - val_loss: 1.3347e-04
Epoch 2/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 5s 18ms/step - loss: 3.0757e-04 - val_loss: 1.2946e-04
Epoch 3/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 2.9352e-04 - val_loss: 1.2381e-04
Epoch 4/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 2.8790e-04 - val_loss: 1.1360e-04
Epoch 5/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 2.4022e-04 - val_loss: 1.0577e-04
Epoch 6/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 2.4182e-04 - val_loss: 1.0535e-04
Epoch 7/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 2.2192e-04 - val_loss: 9.7788e-05
Epoch 8/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - loss: 2.0539e-04 - val_loss: 1.2575e-04
Epoch 9/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 2.0115e-04 - val_loss: 9.1885e-05
Epoch 10/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 1.7981e-04 - val_loss: 1.0723e-04
Epoch 11/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 1.9073e-04 - val_loss: 1.0685e-04
Epoch 12/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 1.7821e-04 - val_loss: 7.9112e-05
Epoch 13/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 1.6256e-04 - val_loss: 7.6323e-05
Epoch 14/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - loss: 1.5701e-04 - val_loss: 7.3365e-05
Epoch 15/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - loss: 1.5373e-04 - val_loss: 1.1625e-04
Epoch 16/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 1.6610e-04 - val_loss: 7.0469e-05
Epoch 17/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 1.4722e-04 - val_loss: 6.6156e-05
Epoch 18/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 1.4589e-04 - val_loss: 7.5368e-05
Epoch 19/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 1.4561e-04 - val_loss: 7.6952e-05
Epoch 20/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 1.4130e-04 - val_loss: 6.3825e-05
Epoch 21/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 1.2440e-04 - val_loss: 9.8071e-05
Epoch 22/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 1.3115e-04 - val_loss: 5.7564e-05
Epoch 23/50
```

```
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 1.1993e-04 - val_loss: 5.7627e-05
Epoch 24/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 1.2597e-04 - val_loss: 5.7219e-05
Epoch 25/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 4s 26ms/step - loss: 1.1929e-04 - val_loss: 7.4693e-05
Epoch 26/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 1.1859e-04 - val_loss: 1.7248e-04
Epoch 27/50
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - loss: 1.4035e-04 - val_loss: 5.7078e-05
Epoch 28/50
```

Build and Train GRU:

In this cell, we build a GRU (Gated Recurrent Unit) model. GRU is simpler and faster than LSTM. It also tries to predict the EUR/USD rate, and we train it on the same training data as LSTM.

```
gru_model = Sequential()
gru_model.add(GRU(50, return_sequences=False, input_shape=(X_train.shape[1], 1)))
gru_model.add(Dense(1))
gru_model.compile(optimizer='adam', loss='mse')

gru_history = gru_model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test), verbose=1)
```

```
Epoch 1/10
129/129 ━━━━━━━━━━━━━━━━━━━━ 4s 22ms/step - loss: 0.0681 - val_loss: 1.8560e-04
Epoch 2/10
129/129 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 2.8937e-04 - val_loss: 1.4309e-04
Epoch 3/10
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 2.4301e-04 - val_loss: 9.7552e-05
Epoch 4/10
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - loss: 2.1322e-04 - val_loss: 1.0582e-04
Epoch 5/10
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 1.8158e-04 - val_loss: 1.0135e-04
Epoch 6/10
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 1.6624e-04 - val_loss: 8.5220e-05
Epoch 7/10
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 1.5763e-04 - val_loss: 7.5002e-05
Epoch 8/10
129/129 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 1.5824e-04 - val_loss: 7.5723e-05
Epoch 9/10
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - loss: 1.4596e-04 - val_loss: 8.4453e-05
Epoch 10/10
129/129 ━━━━━━━━━━━━━━━━━━━━ 5s 17ms/step - loss: 1.5640e-04 - val_loss: 6.6902e-05
```

Make Predictions and Inverse Transform:

We use both trained models to predict exchange rates on the test set. Then we convert the predictions back to the original value range using the scaler. This helps us compare the results with the real exchange rates.

```
lstm_pred = lstm_model.predict(X_test)
gru_pred = gru_model.predict(X_test)

lstm_pred = scaler.inverse_transform(lstm_pred)
gru_pred = scaler.inverse_transform(gru_pred)
y_test_inv = scaler.inverse_transform(y_test)
```

```
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
33/33 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step
```

Calculate RMSE:

We calculate the RMSE (Root Mean Squared Error) for both models. RMSE tells us how far off the predictions are from the real values. A lower RMSE means the model made better predictions.

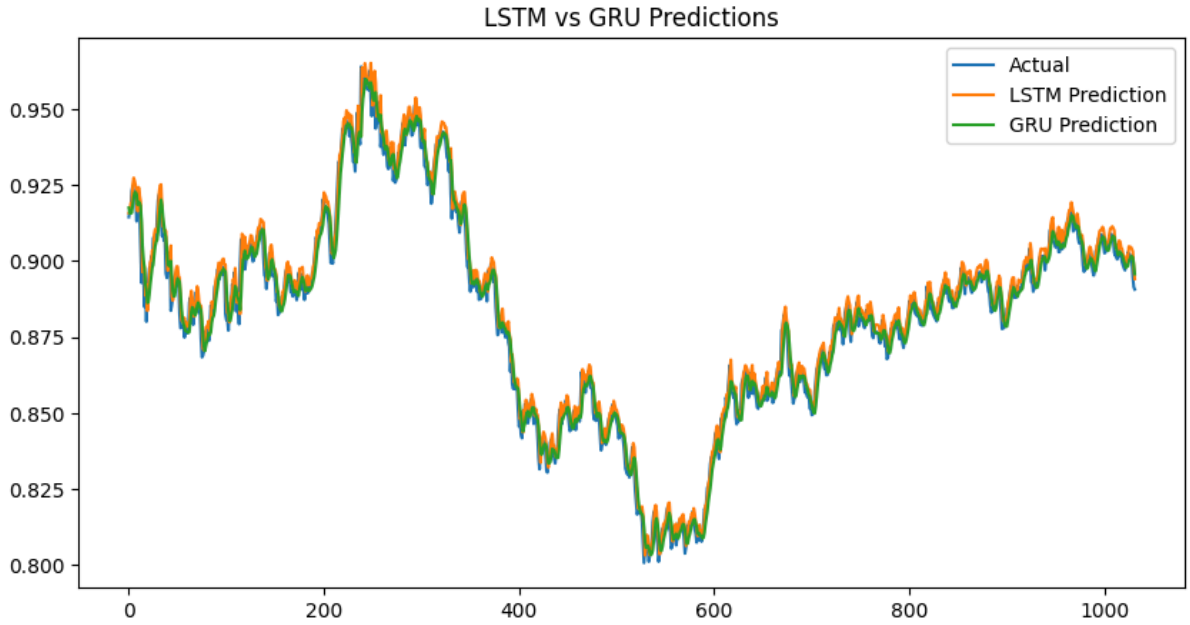
```
lstm_rmse = np.sqrt(mean_squared_error(y_test_inv, lstm_pred))
gru_rmse = np.sqrt(mean_squared_error(y_test_inv, gru_pred))

print(f"LSTM RMSE: {lstm_rmse:.4f}")
print(f"GRU RMSE: {gru_rmse:.4f}")
```

```
LSTM RMSE: 0.0045
GRU RMSE: 0.0048
```

Plot Predictions:

This chart compares actual exchange rates with the predictions made by LSTM and GRU. We can visually see which model is closer to the real values and whether the models followed the trend correctly.

```
plt.figure(figsize=(10,5))
plt.plot(y_test_inv, label='Actual')
plt.plot(lstm_pred, label='LSTM Prediction')
plt.plot(gru_pred, label='GRU Prediction')
plt.legend()
plt.title('LSTM vs GRU Predictions')
plt.show()
```



Final Summary:

Effectiveness of LSTM and GRU

LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) are both strong models used to predict time-based data, but each works better in different situations. LSTM is better when the data has long-term patterns or when past values affect future ones over a long time. It has a special memory system that helps it remember information for longer periods. This makes LSTM good for more complex data, but it also takes more time and power to train. On the other hand, GRU has a simpler design with fewer parts, so it trains faster and works well with smaller datasets or when the patterns in the data are short-term. If you're working with big, detailed data, LSTM might do a better job. But if you need quick results or have less data, GRU is a smart choice.