forex-analysis-eurusd-historical

March 13, 2025

[1]:
```
import os
import pandas as pd
import numpy as np
import yfinance as yf
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from datetime import datetime, timedelta
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, r2_score
warnings.filterwarnings('ignore')
```

[7]:
```
# 1) Descarga datos de los últimos 2 años
end_date = datetime.now()
start_date = end_date - timedelta(days=2*365)
ticker = "EURUSD=X"

# Descarga datos de Yahoo Finance
data = yf.download(ticker, start=start_date, end=end_date)

# Crea el directorio si no existe
os.makedirs("data", exist_ok=True)

# 2) Guardar datos crudos (table Power Bi)
data.to_csv("data/raw_eur_usd_data.csv", sep=",", decimal=".")

# 4) Manejo de missing values
data = data.dropna()

# 5) Asegurar formato fecha
data.reset_index(inplace=True)
data['Date'] = pd.to_datetime(data['Date'])

# 6) Agregar media móvil de 7 días
data['7d_ma'] = data['Close'].rolling(window=7).mean()

# 7) Copia del DataFrame con índice de fecha
```

```
df_copy = data.set_index('Date').copy()

# Guardar datos procesados (en formato parquet)
df_copy.to_parquet("data/processed_eur_usd_data.parquet")
df_copy.to_csv("data/processed_eur_usd_data.csv", sep=",", decimal=".")
```

```
[**********************100%**********************]  1 of 1 completed
```

2) EDA y Transformación Logarítmica

[9]:
```
# cargar datos procesados desde archivo parquet
data = pd.read_parquet("data/processed_eur_usd_data.parquet")
data.head(5)
```

[9]:

Price	Close	High	Low	Open	Volume	7d_ma
Ticker	EURUSD=X	EURUSD=X	EURUSD=X	EURUSD=X	EURUSD=X	
Date						
2023-03-13	1.068365	1.073722	1.065235	1.068365	0	NaN
2023-03-14	1.072501	1.074714	1.067965	1.072501	0	NaN
2023-03-15	1.072766	1.076009	1.052034	1.072766	0	NaN
2023-03-16	1.058335	1.063456	1.055509	1.058335	0	NaN
2023-03-17	1.061413	1.066963	1.061211	1.061413	0	NaN

[11]:
```
# Deshacer el multi-índice de columnas, manteniendo el ultimo nivel
data.columns = data.columns.get_level_values(0)
data.columns.name = None
```

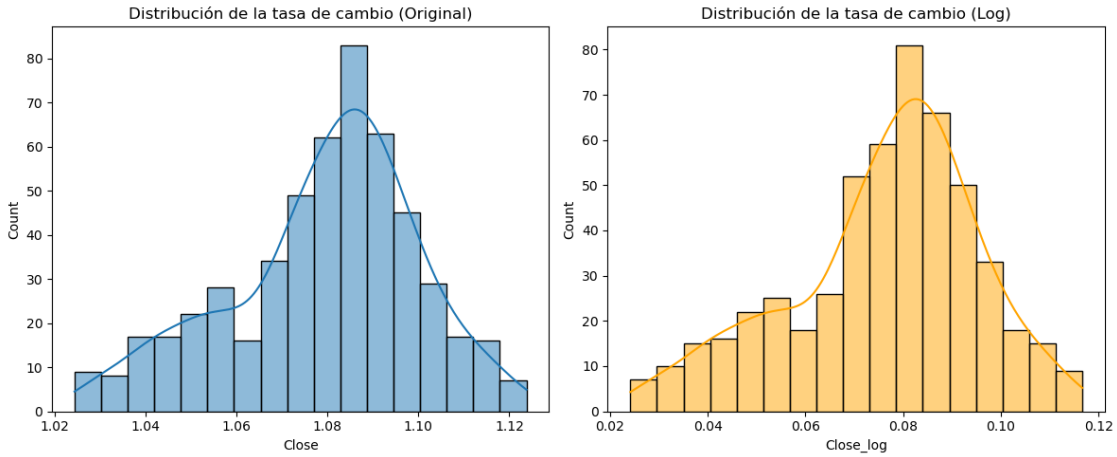
[13]:
```
# Crear columna logarítmica de la tasa de cierre
data['Close_log'] = np.log(data['Close'])

# Guardar los datos con la transformación logarítmica (Para tablero Power Bi)
data.to_csv("data/processed_eur_usd_data_with_log.csv", sep=",", decimal=".")

# a) Crear histogramas de distribución
fig, axes = plt.subplots(1, 2, figsize=(12,5))
sns.histplot(data['Close'], ax=axes[0], kde=True)
axes[0].set_title("Distribución de la tasa de cambio (Original)")

sns.histplot(data['Close_log'], ax=axes[1], kde=True, color='orange')
axes[1].set_title("Distribución de la tasa de cambio (Log)")

plt.tight_layout()
plt.show()
```



[ ]:

[15]:
```
# Resumen de estadísticas originales vs transformadas
stats_original = data['Close'].describe()
stats_log = data['Close_log'].describe()

print("Estadísticas Originales:\n", stats_original)
print("\nEstadísticas Logarítmicas:\n", stats_log)
```

```
Estadísticas Originales:
 count    522.000000
mean       1.079245
std        0.020678
min        1.024443
25%        1.068836
50%        1.082585
75%        1.092804
max        1.123760
Name: Close, dtype: float64

Estadísticas Logarítmicas:
 count    522.000000
mean       0.076077
std        0.019260
min        0.024149
25%        0.066570
50%        0.079352
75%        0.088747
max        0.116680
Name: Close_log, dtype: float64
```

Interpretación Resultados: Estadísticas Originales (Close): count = 523: Se tienen 523 observa-

ciones de la tasa de cambio. mean = 1.082438: En promedio, el tipo de cambio EUR/USD se situó alrededor de 1.0824 durante el periodo analizado. Esto indica que 1 euro, en promedio, equivale a aproximadamente 1.0824 dólares. std (desviación estándar) = 0.016267: La volatilidad (variabilidad) diaria de la tasa fue relativamente baja. Un valor estándar de ~0.016 sobre un promedio de ~1.08 indica que las fluctuaciones suelen ser del orden de poco más de 1 centavo. min = 1.044430: El valor más bajo registrado fue ~1.0444 USD por EUR. 25% = 1.071335: El 25% de los valores estuvieron por debajo de 1.0713, lo que indica que en un cuarto de las observaciones, la tasa fue menor a este nivel. 50% (mediana) = 1.083506: La mediana es muy cercana a la media (1.0835 vs 1.0824). Esto sugiere que la distribución del tipo de cambio está más o menos simétricamente distribuida alrededor del valor central, sin una fuerte asimetría. 75% = 1.092974: El 75% de los valores están por debajo de 1.0930. Esto significa que solo el 25% de las observaciones son mayores a este valor. max = 1.123760: El valor más alto alcanzado fue alrededor de 1.1238 USD por EUR. En general, el rango (min a max) va de ~1.044 a ~1.124. La variación es relativamente estrecha, reflejando una estabilidad considerable en el tipo de cambio EUR/USD durante el periodo. Los cuartiles (25%, 50%, 75%) están muy juntos, indicando una distribución bastante concentrada cerca de la media. Estadísticas Logarítmicas (Close_log): count = 523: Se mantiene el mismo número de observaciones. mean = 0.079103: Este es el promedio de los valores transformados por logaritmo natural. El log transforma multiplicaciones en sumas, y con ello la media logarítmica permite entender mejor las proporciones. Si se aplica la función inversa (exp), exp(0.079103) 1.0823, muy cercano a la media original, lo cual es consistente. std = 0.015029: La desviación estándar en la escala logarítmica es más pequeña y se interpreta como una volatilidad relativa. Una std más pequeña en la escala log indica que la variabilidad porcentual relativa es más limitada. min = 0.043471 y max = 0.116680: En escala logarítmica, el rango es estrecho y al volver a la escala original (exp(min) 1.0444 y exp(max) 1.1237), vemos que se corresponde con los valores mínimos y máximos originales. Cuartiles (25%, 50%, 75%): 25% = 0.068905 → exp(0.068905) 1.0713 50% = 0.080202 → exp(0.080202) 1.0835 75% = 0.088902 → exp(0.088902) 1.0930 Estos cuartiles son la transformación logarítmica directa de los datos originales, manteniendo la misma relación pero en una escala que a menudo es más "suave".

[17]:
```
# Agregar columna de año-mes
data['year_month'] = data.index.to_period('M')

monthly_avg = data.groupby('year_month')['Close'].mean()
print("Promedio mensual:\n", monthly_avg)
```

```
Promedio mensual:
 year_month
2023-03    1.075897
2023-04    1.095870
2023-05    1.087885
2023-06    1.083834
2023-07    1.105559
2023-08    1.091494
2023-09    1.068527
2023-10    1.056466
2023-11    1.080630
2023-12    1.091428
2024-01    1.091784
```

```
2024-02    1.079382
2024-03    1.087403
2024-04    1.073068
2024-05    1.080406
2024-06    1.077042
2024-07    1.084147
2024-08    1.101474
2024-09    1.110223
2024-10    1.090783
2024-11    1.064192
2024-12    1.047404
2025-01    1.035354
2025-02    1.041275
2025-03    1.071537
Freq: M, Name: Close, dtype: float64
```

[19]:
```python
# Eliminar duplicados
data = data[~data.index.duplicated(keep='last')]
```

3) Modelado: Regresión para Predecir Tasas Futuras

[21]:
```python
# Crear features rezagadas (lag features)
df_model = data[['Close_log']].copy()
df_model['Close_log_lag1'] = df_model['Close_log'].shift(1)
df_model['Close_log_lag2'] = df_model['Close_log'].shift(2)
df_model.dropna(inplace=True)

# Split en train y test (por ejemplo, último mes como test)
train_size = int(len(df_model)*0.9)
train = df_model.iloc[:train_size]
test = df_model.iloc[train_size:]

X_train = train[['Close_log_lag1','Close_log_lag2']]
y_train = train['Close_log']
X_test = test[['Close_log_lag1','Close_log_lag2']]
y_test = test['Close_log']

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

# Métricas
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("MAE:", mae)
print("R²:", r2)
```

```
# Predicción a futuro (3 periodos)
# Se toman las últimas filas para crear el pronóstico
last_row = df_model.iloc[-2:]
future_preds = []

# Generar predicciones iterativas:
current_lag1 = df_model['Close_log'].iloc[-1]
current_lag2 = df_model['Close_log'].iloc[-2]

for i in range(3):
    X_future = np.array([[current_lag1, current_lag2]])
    future_log = model.predict(X_future)[0]
    future_preds.append(np.exp(future_log))

    # Actualizar lag para siguiente predicción
    current_lag2 = current_lag1
    current_lag1 = future_log

print("Predicciones a futuro (3 periodos) en valor original:")
print(future_preds)

# Guardar predicciones y métricas
predictions_df = pd.DataFrame({
    'Date': pd.date_range(start=data.index[-1] + pd.Timedelta(days=1),␣
 ↪periods=3),
    'Predicted_Close': future_preds
})

#Almacenar datos para table Power Bi
predictions_df.to_csv("data/future_predictions.csv", index=False, sep=",",␣
 ↪decimal=".")
```

```
MAE: 0.004409395428633363
R²: 0.8447160780320464
Predicciones a futuro (3 periodos) en valor original:
[1.0908425790460883, 1.0905973317121647, 1.0903456808381806]
```

Visualizar las Predicciones Visualizar las predicciones del modelo para el conjunto de prueba: Queremos ver cómo se comparan las predicciones del modelo con los valores reales del Close durante el periodo de prueba. Incluir las predicciones futuras (3 periodos) en la misma gráfica o en una gráfica separada: Mostrar cómo se extienden las predicciones más allá de los datos disponibles.

[23]:
```
y_test_orig = np.exp(y_test)
y_pred_orig = np.exp(y_pred)

test_dates = test.index

future_dates = predictions_df['Date']
```

```
future_preds = predictions_df['Predicted_Close']

plt.figure(figsize=(12,6))
plt.plot(test_dates, y_test_orig, label='Real (Test)', color='blue')
plt.plot(test_dates, y_pred_orig, label='Predicho (Test)', color='red',␣
 ↪linestyle='--')
plt.plot(future_dates, future_preds, label='Predicciones Futuras',␣
 ↪color='green', marker='o')
plt.title('Predicciones Futuras (3 Periodos) + Conjunto de Prueba')
plt.xlabel('Fecha')
plt.ylabel('Precio Close')
plt.legend()
plt.grid(True)
plt.axvline(x=test_dates[-1], color='grey', linestyle='--')
plt.show()
```