Predicción: Nikkei 225 (^N225). Daniel Sainz y Alfonso Méndez.

Introducción:

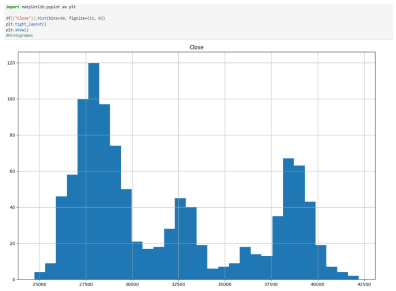
El índice Nikkei 225 es uno de los principales referentes del mercado bursátil japonés y del mercado financiero global. Compuesto por 225 empresas líderes de Japón, incluye compañías de sectores como tecnología, manufactura, finanzas y servicios. Es calculado por el periódico económico Nihon Keizai Shimbun desde 1950 y su metodología consiste en un promedio ponderado por precio de las acciones componentes.
El comportamiento del Nikkei 225 refleja las tendencias económicas y financieras de Japón, y a menudo es utilizado como un indicador de la salud económica de Asia en general. Su evolución está influenciada por factores como las tasas de interés, las políticas gubernamentales, las innovaciones tecnológicas y las dinámicas del comercio internacional.
En este proyecto, se aborda la tarea de **predecir** el **precio** de **cierre** del **Nikkei 225** utilizando una red neuronal de tipo Long Short-Term Memory (LSTM). Las redes LSTM son especialmente adecuadas para el análisis de series temporales, ya que tienen la capacidad de capturar dependencias a largo plazo en los datos. El uso de este tipo de técnicas de aprendizaje profundo ofrece una alternativa robusta para modelar patrones ocultos y comportamientos complejos en las series financieras. [1] [2] [3]

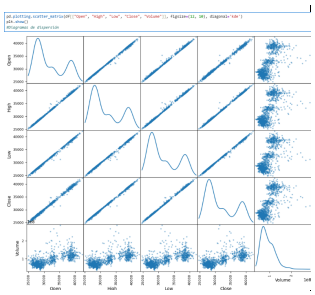
Materiales:

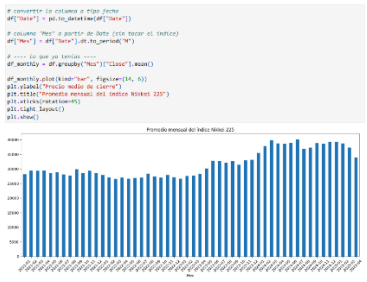
Datos: Se utilizaron datos históricos diarios del índice Nikkei 225 desde el 4 de enero del 2021 hasta el 22 de abril del 2025, descargados desde Yahoo Finance [4], incluyendo precios de apertura, máximo, mínimo, cierre y volumen de operaciones.

Preprocesamiento: Se verificó la ausencia de duplicados y valores nulos. Los datos también fueron normalizados utilizando MinMaxScaler, escalándolos al rango [0, 1]. [5]

Visualización de datos:







La visualización de datos fue una parte importante del proyecto, ya que permitió entender mejor la distribución y el comportamiento de las variables antes de aplicar el modelo.

A través de histogramas, diagramas de dispersión y gráficos de barras, se pudieron identificar patrones, relaciones entre variables y posibles irregularidades en los datos.

Estas gráficas ayudaron a tener una idea más clara de la estructura del conjunto de datos y facilitaron la preparación adecuada para el entrenamiento del modelo.

Software y herramientas utilizadas: Python 3.13.1, Pandas, Scikit-learn, Matplotlib, Plotly Express, TensorFlow/Keras. [6]

Métodos:

En este proyecto se utilizó una red neuronal de tipo Long Short-Term Memory (LSTM), especialmente diseñada para trabajar con datos secuenciales como series temporales.

Las redes LSTM se crearon como solución a los problemas de vanishing y exploding gradients que dificultaban el aprendizaje de redes recurrentes clásicas (RNN). Gracias a sus compuertas internas —de olvido, entrada y salida—, las LSTM son capaces de conservar o descartar información de manera controlada, permitiendo así recordar dependencias a largo plazo.

La arquitectura de las LSTM mantiene un estado interno (cell state) que se actualiza en cada paso temporal. A través del forget gate, se decide qué información eliminar; con el input gate, se incorpora nueva información; y con el output gate, se determina qué parte del estado debe pasar a la salida.

Además, el uso de funciones de activación como sigmoide y tanh ayuda a mantener los valores del estado en rangos controlados, evitando explosiones numéricas.

Dentro de la arquitectura del modelo, usamos la función de activación ReLU (Rectified Linear Unit) en la capa oculta. ReLU es una función que convierte todos los valores negativos en cero y deja pasar los positivos tal cual están.

Su principal ventaja es que permite evitar algunos problemas típicos de funciones antiguas como el sigmoide o la tanh.

En este proyecto, la función ReLU se usó en la capa de 32 neuronas después de las capas LSTM, ayudando al modelo a aprender relaciones no lineales entre las características extraídas de la secuencia. [7]

Para el entrenamiento del modelo, se utilizó el algoritmo Nadam, una variante de Adam que incorpora momentum, junto con la función de pérdida Mean Squared Error (MSE). El MSE fue escogido porque mide el promedio de los errores al cuadrado, penalizando más los errores grandes, lo que ayuda a lograr predicciones más estables. [8]

El preprocesamiento de los datos consistió en normalizar los valores mediante MinMaxScaler y en estructurar las observaciones en ventanas deslizantes de 60 pasos para adaptarlas a la entrada de la red LSTM.

Esta metodología permitió construir un modelo capaz de aprender la dinámica del índice Nikkei 225 y realizar predicciones razonablemente precisas sobre su evolución. [9] [10]

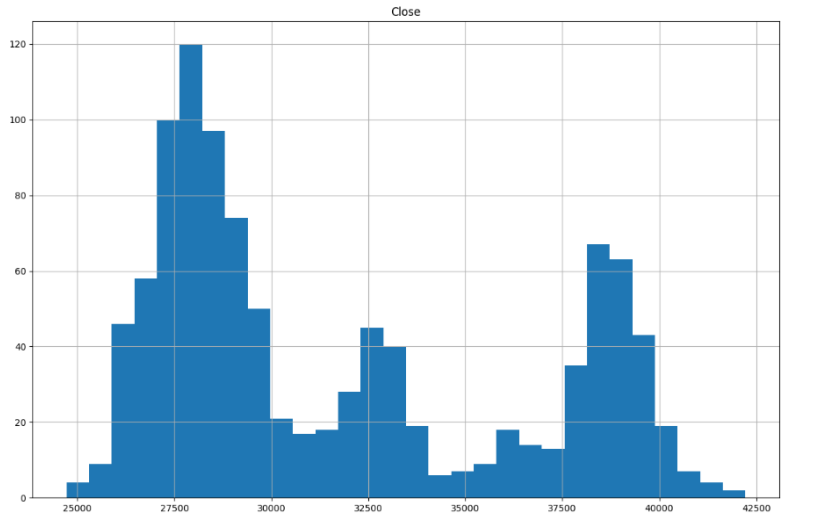
Experimentación:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_percentage_error

# Cargar CSV tomando la fila 2 como encabezado real
df = pd.read_csv("nikkei225.csv", header=2)

# Renombrar la columna 'Price' a 'Date'
df.rename(columns={"Price": "Date"}, inplace=True)



# Renombrar columnas manualmente
df.columns = ["Date","Close", "High", "Low", "Open", "Volume"]
print(df.head())
print("\nColumnas actuales:", df.columns)
```

Primero se importaron las librerías necesarias para trabajar con los datos, graficarlos y preparar el modelo. Después se cargó el archivo nikkei225.csv, indicando que el encabezado real estaba en la segunda fila.

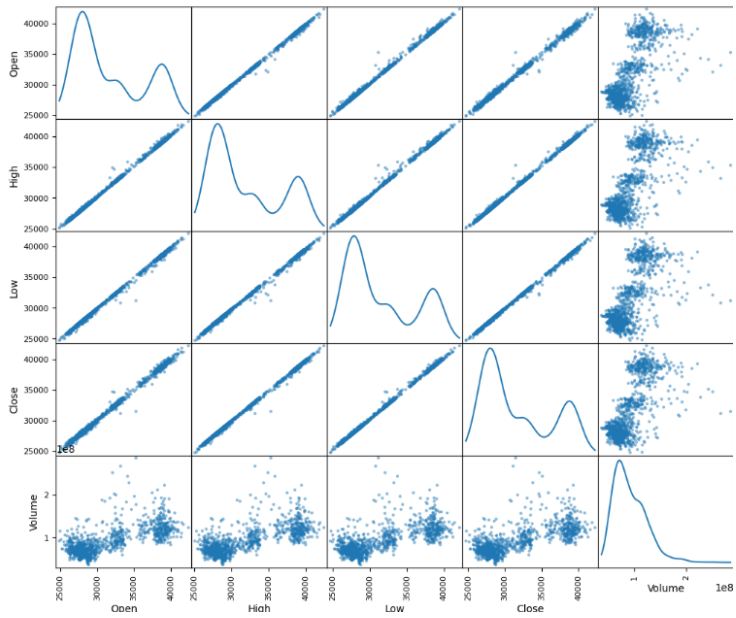
Una vez cargado, se renombró la columna "Price" como "Date", para que fuera más claro el manejo de la fecha. También se ajustaron los nombres de las demás columnas principales, quedando como "Date", "Close", "High", "Low", "Open" y "Volume", dejando el dataset listo para trabajar.

```python
import matplotlib.pyplot as plt

df[["Close"]].hist(bins=30, figsize=(12, 8))
plt.tight_layout()
plt.show()
#Histogramas
```



Después de preparar los datos, se creó un histograma del precio de cierre (Close).
Este gráfico ayuda a ver cómo están distribuidos los precios a lo largo del tiempo. Se usaron 30 bins para observar mejor si había acumulaciones de precios en ciertos rangos o si había valores que se salían mucho del resto (outliers).
El histograma mostró varias zonas donde los precios se concentraban más.

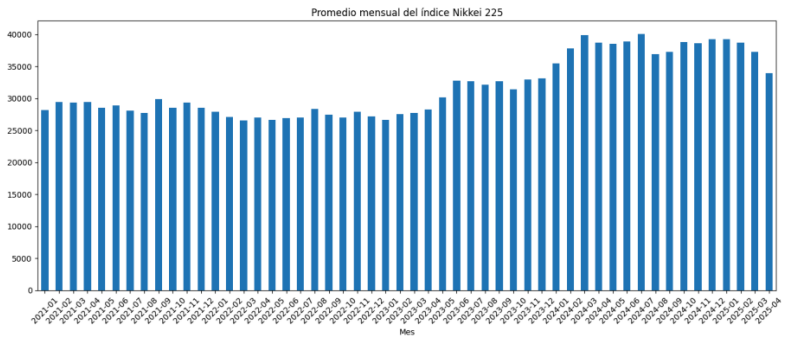
```
pd.plotting.scatter_matrix(df[["Open", "High", "Low", "Close", "Volume"]], figsize=(12, 10), diagonal='kde')
plt.show()
#Diagramas de dispersión
```



Luego se creó una matriz de diagramas de dispersión para las variables Open, High, Low, Close y Volume.

Esto permitió observar cómo se relacionaban entre sí y detectar patrones de dependencia o agrupaciones entre las variables. Por ejemplo, se pudo observar que Open, High, Low y Close están muy relacionadas entre sí, mientras que el Volume presenta un comportamiento más disperso respecto a los precios.

```
# convertir la columna a tipo fecha
df["Date"] = pd.to_datetime(df["Date"])

# columna "Mes" a partir de Date (sin tocar el índice)
df["Mes"] = df["Date"].dt.to_period("M")

# ---- lo que ya tenías ----
df_monthly = df.groupby("Mes")["Close"].mean()

df_monthly.plot(kind="bar", figsize=(14, 6))
plt.ylabel("Precio medio de cierre")
plt.title("Promedio mensual del índice Nikkei 225")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Después se creó una nueva columna llamada “Mes” para agrupar los datos de cierre (Close) por mes.
Luego se calculó la media de cierre de cada mes y se representó en un gráfico de barras.
Este gráfico permitió ver cómo ha ido cambiando el precio promedio del índice Nikkei 225 mes a mes, identificando tendencias o variaciones a lo largo del tiempo.

Vamos a dividir los datos en 2 conjuntos, uno para entrenar la red neuronal LSTM y otro para posteriormente, una vez entrenada la red neuronal, poder probar con datos que no haya visto antes si funciona bien. En nuestro caso hemos decidido dejar como parte de los datos de "test" todos los que pertenecen al año 2025. Así también podremos comparar con cómo ha ido Nikkei 225 en la vida real.

```
test_size = df[df.Date.dt.year==2023].shape[0]
train_close = df.Close[:-test_size].values.reshape(-1, 1)
test_size
```

```
246
```

Hemos decidido coger todos los datos hasta el 2023 para entrenar y los datos del 2024 y lo que resta de 2025 para test. Así tenemos suficientes datos para que el modelo no quede overfitteado y tengamos bastantes datos para probar. Para ver bien la diferencia vamos a graficarlo tambien. Estan los datos de training a negro, y los de test en azul.

Vamos a escalar los datos. Esto es porque el Nikkei 225 cotiza entre 25 000 – 40 000¥.

```
scaler = MinMaxScaler()
scaler.fit(train_close)
```

```
▾ MinMaxScaler
MinMaxScaler()
```

```
train_scaled = scaler.transform(train_close)
test_scaled  = scaler.transform(df.Close[-test_size:].values.reshape(-1,1))
```

MinMaxScaler reescala cada dato del vector a un rango pequeño (por defecto entre 0 y 1).

Fit() aprende los valores mínimo y máximo solo con los datos de entrenamiento (train_close) Para que el modelo no pueda ver los datos de test y se haga overfitting.

Transform() aplica la misma fórmula a todo el entrenamiento para generar train_scaled.

Luego concatenamos los precios de todos los años con reshape para formar una matriz 2D. Esto garantiza que training y test compartan la misma escala.

```
window_size = 60
train_data = df.Close[:-test_size]
train_data = scaler.transform(train_data.values.reshape(-1,1))
```

Aqui vamos a usar ventanas deslizantes.

Hemos elegido que cada entrada a la LSTM solo tenga los 60 dias de cierre anteriores (aproximadamente unos 2 meses de mercado abierto). Necesitamos una secuencia de longitud fija, y con 60 dias tenemos para aprender bien tendencias a corto y medio plazo sin hacer la matriz demasiado grande y selecciono solo los datos de entrenamiento.

Ahora voy a separar los datos de entrenamiento y los de prueba.

Los de entrenamiento:

```
X_train = []
y_train = []

for i in range(window_size, len(train_data)):
    X_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])
```

Cada iteracion representa una ventana deslizante. Recorre la serie escalada “train_data” desde el indice 60 hasta el final. Asi convierte la serie temporal en un problema de aprendizaje supervisado. El LSTM recibe secuencias de 60 epocas y aprende a predecir la siguiente.

Los de prueba:

Datos de prueba *Prueba*

```
3]: test_data = df.Close[-test_size-60:]
    test_data = scaler.transform(test_data.values.reshape(-1,1))
```

```
4]: X_test = []
    y_test = []

    for i in range(window_size, len(test_data)):
        X_test.append(test_data[i-60:i, 0])
        y_test.append(test_data[i, 0])
```

Df.close[-test_size-60:] lo que hace es extraer los ultimos registros de test_size + los 60 anteriores. Asi la primera ventana de prueba tendra tambien 60 observaciones completas.

Se normaliza con el mismo scaler aprendido en training (asi tenemos los mismos limites minimos y maximos)

```
X_train = np.array(X_train)
X_test  = np.array(X_test)
y_train = np.array(y_train)
y_test  = np.array(y_test)
```

```
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test  = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
y_train = np.reshape(y_train, (-1,1))
y_test  = np.reshape(y_test, (-1,1))
```

```
print('X_train Shape: ', X_train.shape)
print('y_train Shape: ', y_train.shape)
print('X_test Shape:  ', X_test.shape)
print('y_test Shape:  ', y_test.shape)
```

Después se convirtieron lo siguientes conjuntos: X_train, X_test, y_train y y_test en arrays de Numpy para prepararlos para el modelo.

Luego se reestructuraron (reshape) para que tuvieran el formato que necesita una red LSTM: una dimensión temporal y una característica por paso de tiempo.

Finalmente se verificaron las dimensiones (shape) de cada conjunto para asegurarse de que estuvieran correctas antes del entrenamiento.

El modelo LSTM

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Dropout, Dense

def define_model(window_size):
    input1 = Input(shape=(window_size,1))
    x = LSTM(units = 64, return_sequences=True)(input1)
    x = Dropout(0.2)(x)
    x = LSTM(units = 64, return_sequences=True)(x)
    x = Dropout(0.2)(x)
    x = LSTM(units = 64)(x)
    x = Dropout(0.2)(x)
    x = Dense(32, activation='relu')(x)      # capa oculta densa
    dnn_output = Dense(1, activation='linear')(x)   # salida de regresión


    model = Model(inputs=input1, outputs=[dnn_output])
    model.compile(loss='mean_squared_error', optimizer='Nadam')
    model.summary()

    return model
```

Se definió la arquitectura del modelo LSTM.
El modelo tiene tres capas LSTM con 64 unidades cada una, intercaladas con capas de Dropout al 20% para evitar sobreajuste.
Después se añadió una capa densa de 32 neuronas con activación ReLU, y una capa de salida densa de 1 neurona con activación lineal para predecir el valor final.
El modelo se compiló usando la función de pérdida **MSE** y el optimizador **Nadam**.

```
model = define_model(window_size)
history = model.fit(X_train, y_train, epochs=150, batch_size=32, validation_split=0.1, verbose=1)
```

Después se entrenó el modelo usando los datos de entrenamiento.
Se entrenó durante 150 épocas, con un tamaño de batch de 32, y se reservó el 10% de los datos de entrenamiento para validación.
Durante el proceso, se registraron los valores de pérdida para ir evaluando la evolución del aprendizaje.

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 60, 1)	0
lstm_3 (LSTM)	(None, 60, 64)	16,896
dropout_3 (Dropout)	(None, 60, 64)	0
lstm_4 (LSTM)	(None, 60, 64)	33,024
dropout_4 (Dropout)	(None, 60, 64)	0
lstm_5 (LSTM)	(None, 64)	33,024
dropout_5 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2,080
dense_3 (Dense)	(None, 1)	33

Total params: 85,057 (332.25 KB)

Trainable params: 85,057 (332.25 KB)

Non-trainable params: 0 (0.00 B)

El modelo cuenta con 85,057 parámetros entrenables, lo que le da suficiente capacidad para aprender patrones complejos en los datos, pero sin ser excesivamente pesado, evitando problemas de sobreajuste si se entrena correctamente.

Evaluación

Evaluar el modelo

```
from sklearn.metrics import mean_squared_error
result = model.evaluate(X_test, y_test)
y_pred = model.predict(X_test)

MAPE = mean_absolute_percentage_error(y_test, y_pred)
Accuracy = 1 - MAPE

print("Test Loss:", result)
print("Test MAPE:", MAPE)
print("Test Accuracy:", Accuracy)
mse = mean_squared_error(y_test, y_pred)
print(f"MSE (¥): {mse:,.2f}")
```

A model evaluate le damos todas las ventanas de prueba (X_test) y comparamos sus salidas con los valores reales (y_test).

Model.predict calcula las predicciones una a una para poder usarlas despues en otros indicadores.

MAPE (Mean Absolute Percentage Error): mide el error porcentual promedio entre los valores reales y los valores predichos. Cuanto más bajo sea el MAPE, mejor es el modelo.

Accuracy: se calculó como 1 - MAPE, para tener una estimación aproximada de la precisión del modelo en la predicción de precios.

MSE (Mean Squared Error) Media del cuadrado de los errores. Penaliza mucho los fallos grandes. [11]

```
# ==== BLOQUE NUEVO ====
# 2-D siempre para inverse_transform
y_true = scaler.inverse_transform(y_test)                # (N,1)
y_pred = scaler.inverse_transform(y_pred.reshape(-1,1))  # (N,1)

# nº real de muestras de test (N = 226)
n_test   = len(y_true)
dates_ts = df['Date'].iloc[-n_test:]                     # fechas alineadas
```

```
plt.figure(figsize=(15,6), dpi=150)

# tramo de entrenamiento
plt.plot(
    df['Date'].iloc[:-n_test],
    scaler.inverse_transform(train_data).ravel(),
    color='black', lw=2, label='Training Data'
)

# valores reales de test
plt.plot(
    dates_ts, y_true.ravel(),
    color='blue', lw=2, label='Actual Test Data'
)

# predicciones
plt.plot(
    dates_ts, y_pred.ravel(),
    color='red', lw=2, label='Predicted Test Data'
)

plt.title('Nikkei 225 – Reales vs. Predicción LSTM')
plt.xlabel('Date'); plt.ylabel('Price')
plt.legend(loc='upper left'); plt.grid(True)
plt.show()
```

Para poder interpretar mejor las predicciones, se revirtieron las transformaciones de escala aplicadas a y_test y y_pred usando inverse_transform. Así se recuperaron los valores reales en su escala original de precios.
Luego se creó una gráfica donde se compararon tres tramos: el final de los datos de entrenamiento (Training Data), los valores reales del conjunto de prueba (Actual Test Data) y las predicciones generadas por el modelo (Predicted Test Data).

<u>Resultados:</u>

```
Test Loss: 0.006730309221893549
Test MAPE: 0.0902713326892698
Test Accuracy: 0.9097286673107302
MSE (¥): 0.01
```

Test Loss (0.0067) -> Es el error cuadratico medio normalizado, es decir, calculado sobre los datos ya escalados.

MAPE (0.0903 = 9%) -> Esto quiere decir, que, de media, la prediccion se desvia un 9% del valor real. Si el Nikkei está en 35 000 ¥, el error medio sería ≈ 3 150 ¥ arriba o abajo.

Accuracy (0.909 = 91%) -> Acertamos el 91% del precio y falló en el 9%

MSE (0.01) -> Antes de des-escalar, por eso es tan pequeño.

En la gráfica, los valores reales se representaron en azul y las predicciones en rojo, lo que permitió visualizar fácilmente qué tan bien el modelo consiguió seguir la tendencia real del índice Nikkei 225.

Los resultados obtenidos tras el entrenamiento y evaluación del modelo muestran que el LSTM ha sido capaz de seguir la evolución del índice Nikkei 225 de forma bastante precisa.

El modelo consiguió un Test Loss bajo (0.0067), un MAPE de aproximadamente 9%, y una precisión cercana al 91%, lo cual refleja un muy buen ajuste a los datos.

El MSE también fue bajo (0.01), indicando que las diferencias cuadráticas entre los valores predichos y los valores reales fueron pequeñas.

Al observar la gráfica de comparación, se puede ver que el modelo siguió bien la tendencia general del mercado, aunque en periodos de alta volatilidad existieron algunas desviaciones.

Este comportamiento es normal en modelos financieros, donde los cambios bruscos y repentinos son difíciles de anticipar con exactitud.

En general, los resultados son positivos y demuestran que el enfoque de usar redes LSTM para este tipo de predicciones es adecuado, aunque siempre se pueden explorar mejoras como ampliar el conjunto de datos, ajustar los hiperparámetros o probar otras arquitecturas más complejas.

Conclusiones:

En este proyecto hemos conseguido desarrollar un modelo de predicción para el precio de cierre diario del índice Nikkei 225 utilizando una red neuronal de tipo LSTM. [12]

A lo largo del trabajo hemos pasado por diferentes fases: carga de datos, preprocesado de datos, visualizar las distitnas variables y el dataset en general, normalizar los datos,

seperar en datos de test y de entrenamiento, análisis exploratorio y modelado de la serie temporal.

El uso de técnicas de preprocesamiento como la normalización y la creación de ventanas de datos nos ha permitido preparar adecuadamente la información para el entrenamiento del modelo.

La elección de una arquitectura LSTM ha sido buena, ya que este tipo de redes son capaces de captar patrones y dependencias a largo plazo, algo fundamental en series temporales financieras.

Aunque existen factores externos impredecibles que siempre afectan los mercados, el modelo ha demostrado tener un buen desempeño general.

En conclusión, el proyecto cumplió su objetivo de construir una herramienta capaz de anticipar de manera razonable la evolución del Nikkei 225.

Referencias:

Referencias

[1] W. contributors, «Wikipedia,» Wikipedia Foundation, 2024. [En línea]. Available: https://en.wikipedia.org/wiki/Nikkei_225.

[2] W. contributors, «Wikipedia,» 2024. [En línea]. Available: https://en.wikipedia.org/wiki/Long_short-term_memory.

[3] M. Qiu, « Application of artificial neural network for the prediction of stock market returns: The case of the Japanese stock market,» *Chaos, Solitons & Fractals,* p. 7, 2016.

[4] Y. Finance, «Yahoo Finance,» 2025. [En línea]. Available: https://finance.yahoo.com/quote/%5EN225/.

[5] A. S. M. J. A. F. A. F. Khairul Nizam Abd Halim, «Data Pre-Processing Algorithm for Neural Network Binary Classification Model in Bank Tele-Marketing,» *International Journal of Innovative Technology and Exploring Engineering (IJITEE),* 2020.

[6] R. B. Z. De Bharath Ramsundar, TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning, 2018.

[7] V. V. R. Deepanshu Gupta, «Solar Power Prediction Based on Recurrent Neural Networks Using LSTM and Dense Layer With ReLU Activation Function,» *2023 3rd International Conference on Intelligent Technologies (CONIT),* 2023.

[8] M. H. E. A. R. A. E. Mohamed M. Zahra, “Performance Functions Alternatives of Mse for Neural Networks Learning,” *International Journal of Engineering Research & Technology (IJERT),* p. 4, 2014.

[9] C. B. G. Maldonado, «CUNEF,» [En línea]. Available: https://cunef.instructure.com/courses/9439/pages/3-redes-neuronales-lstm?module_item_id=247531.

[10] Y. B. A. C. Ian Goodfellow, Deep Learning, MIT Press, 2016.

[11] N. aparece, «IBM Cloud Learn Hub,» 2023. [En línea]. Available: https://www.ibm.com/cloud/learn/machine-learning.

[12] A. M. Mhamed Hamiche, «Stock Market Prediction Using LSTM Recurrent Neural Network,» *Procedia Computer Science,* 2020.

---

i