proyecto utiliza redes neuronales, específicamente un tipo de red neuronal llamada LSTM (Long Short-Term Memory). Las LSTM son un tipo especial de red neuronal recurrente (RNN) diseñadas para trabajar con secuencias y series temporales.

Los LSTM se utilizan predominantemente para aprender, procesar y clasificar datos secuenciales porque estas redes pueden aprender dependencias a largo plazo entre pasos de tiempo de datos.

Este proyecto utiliza principalmente dos conceptos clave de la Inteligencia Artificial:

Este proyecto utiliza principalmente dos conceptos clave de la Inteligencia Artificial:

1. Redes Neuronales Recurrentes (RNN), específicamente Long Short-Term Memory (LSTM): Las LSTM son un tipo especial de RNN diseñadas para manejar problemas de dependencia a largo plazo en series temporales. A diferencia de las RNN tradicionales, las LSTM tienen una estructura interna más compleja (llamada celda de memoria) que les permite aprender, olvidar y actualizar información a lo largo del tiempo. Esto las hace ideales para predecir series temporales como el consumo de energía.
2. Aprendizaje Supervisado: El modelo se entrena utilizando datos históricos etiquetados (consumo de energía pasado), con el objetivo de predecir valores futuros. Este es un ejemplo clásico de aprendizaje supervisado, donde el modelo aprende a mapear entradas (secuencias de consumo pasado) a salidas (consumo futuro).

Descripción paso a paso del código:

1. Importación de bibliotecas: El código comienza importando las bibliotecas necesarias, incluyendo Streamlit para la interfaz de usuario, Pandas y NumPy para manipulación de datos, Scikit-learn para preprocesamiento, TensorFlow/Keras para redes neuronales, y Matplotlib para visualización.
2. Carga y preparación de datos (función load\_and\_prepare\_data):
   * Lee el archivo CSV con los datos históricos.
   * Convierte la columna 'Fecha' a formato datetime y ordena los datos.
   * Agrega características de día de la semana y mes del año.
   * Crea secuencias de datos para el entrenamiento (cada secuencia tiene 7 días de datos).
   * Normaliza los datos usando MinMaxScaler.
3. Creación y entrenamiento del modelo (función create\_and\_train\_model):
   * Define la arquitectura de la red LSTM.
   * Compila el modelo con el optimizador Adam y la función de pérdida MSE.
   * Divide los datos en conjuntos de entrenamiento y prueba.
   * Entrena el modelo con los datos de entrenamiento.
4. Función de predicción (predict\_energy):
   * Toma una secuencia de entrada, la escala, y utiliza el modelo para hacer una predicción.
   * Desescala la predicción para obtener el valor real de consumo de energía.
5. Interfaz de usuario con Streamlit:
   * Crea una interfaz web interactiva.
   * Permite al usuario entrenar el modelo o cargar uno existente.
   * Proporciona un input para seleccionar la fecha de predicción.
   * Muestra los resultados de la predicción y una visualización gráfica.
6. Entrenamiento del modelo (cuando se solicita):
   * Llama a la función de creación y entrenamiento del modelo.
   * Guarda el modelo entrenado en un archivo.
   * Muestra un gráfico de la pérdida durante el entrenamiento.
7. Generación de predicciones:
   * Carga el modelo entrenado.
   * Prepara los datos de entrada para la fecha seleccionada.
   * Realiza la predicción y muestra el resultado.
   * Genera un gráfico que muestra los datos históricos y la predicción.

.

este proyecto utiliza redes neuronales, específicamente un tipo de red neuronal llamada LSTM (Long Short-Term Memory). Las LSTM son un tipo especial de red neuronal recurrente (RNN) diseñadas para trabajar con secuencias y series temporales.

En el código proporcionado, podemos ver el uso de redes neuronales en varias partes:

1. Importación de TensorFlow/Keras:

python

Copy

from tensorflow.keras.models import Sequential, load\_model

from tensorflow.keras.layers import Dense, LSTM, Dropout

from tensorflow.keras.optimizers import Adam

Estas importaciones son de la biblioteca TensorFlow/Keras, que se utiliza para crear y entrenar redes neuronales.

1. Creación del modelo:

python

Copy

def create\_and\_train\_model(X, y):

model = Sequential([

LSTM(50, activation='relu', input\_shape=(X.shape[1], X.shape[2]), return\_sequences=True),

Dropout(0.2),

LSTM(50, activation='relu', return\_sequences=False),

Dropout(0.2),

Dense(25, activation='relu'),

Dense(1)

])

Aquí se define una arquitectura de red neuronal que incluye capas LSTM, capas de Dropout para regularización, y capas Dense (completamente conectadas).

1. Entrenamiento del modelo:

python

Copy

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=32, validation\_data=(X\_test, y\_test), verbose=0)

Esta línea entrena la red neuronal utilizando los datos de entrenamiento.

1. Predicción:

python

Copy

prediction\_scaled = model.predict(input\_scaled)

Aquí se utiliza el modelo entrenado para hacer predicciones.

Las redes LSTM son particularmente adecuadas para este tipo de problema de predicción de series temporales, ya que pueden capturar dependencias a largo plazo en los datos, lo que es crucial para predecir el consumo de energía basado en patrones históricos.