Predicciones de generación de energía neta con el data set **EIA Form 923 – Power Plant Operations Report**, el cual contiene información mensual sobre la operación de plantas eléctricas en Estados Unidos. Este Dataset posee múltiples variables con candidatas lo suficientemente válidas para ser la variable a inferir pero en este caso se tomó la "Producción neta de energía eléctrica" (en megavatios-hora).

Como preprocesamiento debido a la redundancia del propio data set se decidió filtrar las columnas que en lo personal considere las mas relevantes como Id planta, Nombre planta, Estado, Combustible consumido (millones de BTU), Energia neta generada (Megavatios Hora) pero se termino usando para X: El consumo de combustible, año y el estado (se descartaron variables como el id y el nombre de planta porque una no tenía relación matemática con la variable a inferir y generaba baja precisión en el modelo mientras que la otra era una variable nominal) para Y: se utilizó La producción de energía neta porque me parecía un dato relevante.

One-hot encoding es una técnica que basicamete nos brinda columnas binarias por cada categoría de una columna en este caso la columna “State” y con drop\_first=True deducimos el valor de una columna tiendo el resto como 0 (evita analizar de más).

Realizamos la división de los datos en test\_size=0.2 (dándonos 20% para test y el resto para entrenamiento y como la función train\_test\_split() saca filas aleatorias para el entrenamiento necesita una semilla random\_state=42 para evitar que por aleatoriedad vuelva a sacar la misma.

El escalador básicamente usa esta formula Xnorm = X – U (media) / O (desviación) y en el caso de regresión lineal solo es necesario escalar X mientras en MLP es necesario escalar Y también ya que es más sensible a la escala de salida.

Np.c es una función para concatenar arrays, (X\_train\_norm.shape[0], 1) esa generando una columna de 1 en la posición 0 y une a X\_train\_norm.

La explicación de porque el costo es demasiado alto en el entrenamiento es porque la variable Y es demasiado grande y se agranda mucho mas al elevarla al cuadrado (y inferida - valor real)^2

En la ultima grafica se puede ver los puntos azules como la representación de los datos de prueba y los inferidos mientras que la línea roja es la regresión lienal (theta) en un grafico entre prediccion y valor real

MLP  
En este caso de la transformación se explico anteriormente que se debe estandarizar los valores de y por la sensibilidad de los datos para esto antes de pasarle el vector a StandardScaler se debe transformar el vector a un array de 1 columna con reshape().

Y por eficiencia, compatibilidad y simplicidad de nuestros datos los convertimos a tensores con datos de tipo float32.

Al momento de modelar se tuvo las consideraciones necesarias basado en los datos de entrada del Dataset por eso se evito colocar mas capas y neuronas necesarias ya que el modelo no lograba converger y sufría de overfiting.

Respecto a la grafica el modelo converge rápido y sin problemas y la pequeña oscilación que se nota es por el optimizador Stocastic Gradient Descent (SGD) mas especifico su momento que funciona como la inercia en física dando un 90 de fuerza para converger en ese sentido y el otro 10 para exploración (el revote).

Al momento de evaluar el modelo lo cambiamos a modo evaluacio para poder desactivar las gradientes, una vez desactivada las gradientes le pasamos X de prueba con formato .numpy ya que sin eso no podría invertir la normalización.