ITSRLL
INSTITUTO TECNOLÓGICO SUPERIOR DE LA REGIÓN DE LOS LLANOS

Ingeniería Mecatrónica

PROGRAMACIÓN AVANZADA

Enero – Junio 2025

M.C. Osbaldo Aragón Banderas

UNIDAD:

1	2	3	4	5

Actividad número: 4

Nombre de actividad:

NOOTEBOOK: Análisis de Datos Aplicables a Regresión Lineal Simple

Actividad realizada por:

Carlos Diego Dominguez López - 21030006

Guadalupe Victoria, Durango

Fecha de entrega:

09	03	2025

INDICE

Introducción

La regresión lineal simple es una técnica fundamental en el análisis de datos y la predicción de tendencias. Su aplicación permite modelar la relación entre dos variables: una independiente y otra dependiente, facilitando la estimación de valores futuros con base en datos históricos. En este reporte, se explorará el concepto de regresión lineal simple, su formulación matemática y el método de mínimos cuadrados para determinar la mejor línea de ajuste.

Para aplicar este conocimiento, se seleccionará un conjunto de datos de Kaggle que relacione la experiencia laboral (en años) con el salario, con el objetivo de construir un modelo de regresión lineal simple. Se llevará a cabo un proceso de preprocesamiento de los datos, incluyendo su limpieza y visualización, para garantizar su calidad antes de aplicar el modelo. Posteriormente, se implementará la regresión lineal mediante la biblioteca scikit-learn y se analizará su desempeño mediante métricas de evaluación. Finalmente, se presentarán los resultados obtenidos y su interpretación, destacando la utilidad del modelo para predecir salarios en función de la experiencia laboral.

Este estudio permitirá comprender mejor cómo se ajusta un modelo de regresión lineal simple y cómo se pueden interpretar sus resultados en un contexto práctico, reforzando así los conceptos teóricos con una aplicación real.

Objetivo

El propósito de esta actividad es que los estudiantes busquen, seleccionen y analicen un conjunto de datos en Kaggle u otra fuente confiable para aplicar regresión lineal simple y predecir una variable continua con base en una única característica independiente. Además, deberán presentar los resultados y conclusiones obtenidas, explicando la relación entre las variables y la interpretación del modelo.

Marco Teórico

Regresión Lineal Simple y su Aplicación en Problemas de Predicción

La regresión lineal simple es una técnica estadística utilizada para modelar la relación entre una variable dependiente (y) y una variable independiente (x). Su principal objetivo es encontrar una ecuación matemática que permita predecir el valor de y con base en x. Este tipo de análisis se emplea ampliamente en diversas disciplinas como economía, biología, ingeniería y ciencias de datos, ya que proporciona una forma sencilla pero efectiva de analizar tendencias y hacer predicciones.

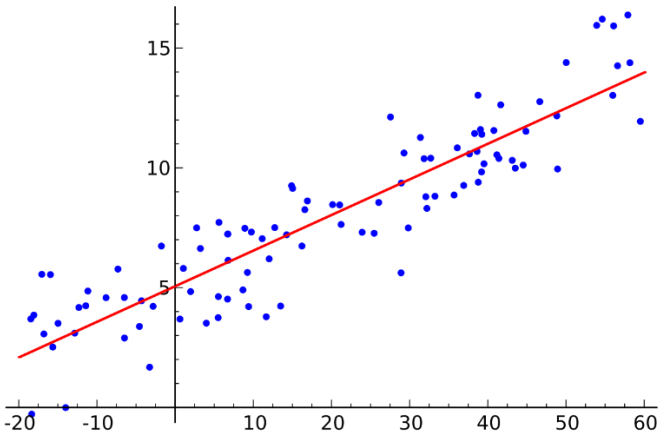

Figura 1. Ejemplo de grafica de Regresión Lineal Simple

Un ejemplo común de aplicación es la estimación de salarios en función de la experiencia laboral. Si se tiene un conjunto de datos que contiene años de experiencia y el salario correspondiente de varios empleados, se puede utilizar la regresión lineal simple para encontrar una ecuación que prediga el salario de un nuevo empleado con base en su tiempo de experiencia.

Suposiciones del Modelo de Regresión Lineal Simple

Para que el modelo de regresión lineal simple sea válido, deben cumplirse ciertas suposiciones:

1. **Linealidad**: La relación entre x e y debe ser aproximadamente lineal.
2. **Independencia**: Las observaciones deben ser independientes entre sí.
3. **Homocedasticidad**: La varianza del término de error debe ser constante a lo largo de todos los valores de x.
4. **Normalidad del Error**: Los residuos (errores) deben seguir una distribución normal para realizar inferencias estadísticas confiables.

Si alguna de estas suposiciones no se cumple, el modelo puede no ser el más adecuado para los datos y podrían considerarse técnicas alternativas como la regresión polinómica o la transformación de variables.

Ecuación Matemática de la Regresión Lineal Simple

La regresión lineal simple es un modelo matemático que describe la relación entre dos variables mediante una ecuación de una recta. Su estructura general es la siguiente:

$$Y = mX + b \tag{1.1}$$

Donde:

- Y es la variable dependiente, el valor que queremos predecir.
- X es la variable independiente, el factor explicativo.
- m es la **pendiente** de la recta, que representa el cambio en Y por cada unidad de X.
- b es el **intercepto** o punto donde la recta cruza el eje Y, es decir, el valor de Y cuando X=0.

Esta ecuación representa una recta en un plano cartesiano y describe cómo cambia Y en función de X. La regresión lineal simple permite encontrar los valores óptimos de m y b para minimizar la diferencia entre los valores predichos y los valores reales.

Interpretación de los Coeficientes m y b

Pendiente (m)

La pendiente m indica cuánto cambia Y cuando X aumenta en una unidad.

- Si m>0, la relación entre las variables es positiva: al aumentar X, Y también aumenta.
- Si m<0, la relación es negativa: al aumentar X, Y disminuye.
- Si m=0, no hay relación entre X e Y, la recta es horizontal.

Intercepto (b)

El intercepto b es el valor de Y cuando X=0. Representa el punto de inicio en el eje Y.

Determinación de la Mejor Línea de Ajuste mediante Mínimos Cuadrados

Para encontrar la mejor línea de ajuste, se usa el método de **mínimos cuadrados**, que busca minimizar la suma de los errores entre los valores reales y los valores predichos por la ecuación. La diferencia entre el valor observado (Y_i) y el valor estimado ($\widehat{Y}_i$) se llama **residuo** o **error**:

$$Error = Y_i - \widehat{Y}_i \tag{2.1}$$

La función de error a minimizar es la **suma de los errores cuadrados (SSE)**:

$$SSE = \sum (Y_i - (mX_i + b))^2 \tag{2.2}$$

Los coeficientes m y b se calculan con las siguientes fórmulas:

$$m = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sum (X_i - \overline{X})^2} \tag{2.3}$$

$$b = \bar{Y} - m\bar{X} \tag{2.4}$$

Este método garantiza que la línea encontrada es la que minimiza la distancia al cuadrado entre los valores observados y los valores predichos, asegurando la mejor precisión posible dentro de un modelo lineal.

Desarrollo

Para esta práctica se utilizó un data set sobre la relación entre el salario que se gana al trabajar en una empresa, basándonos en la edad y los años de experiencia de los trabajadores.

Descripción del Código

Lo primero que hacemos es exportar las siguientes librerías que nos ayudaran en la creación de la regresión lineal simple con nuestra tabla de datos.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

Figura 2. Librerías para la Regresión Lineal Simple

El código utiliza la biblioteca **panda** para leer un archivo CSV llamado *Salary_Data.csv* y almacenarlo en un DataFrame llamado **df**. Especifica que las columnas están separadas por comas (**sep=','**) y define manualmente los nombres de las columnas como *"YearsExperience"*, *"Age"*, y *"Salary"*. Finalmente, imprime el contenido del DataFrame, permitiendo visualizar los datos cargados.

```
import pandas as pd
df = pd.read_csv("Salary_Data.csv", sep=',', header=0, names=["YearsExperience", "Age", "Salary"])
print(df)
```

```
    YearsExperience   Age  Salary
0               1.1  21.0   39343
1               1.3  21.5   46205
2               1.5  21.7   37731
3               2.0  22.0   43525
4               2.2  22.2   39891
5               2.9  23.0   56642
6               3.0  23.0   60150
7               3.2  23.3   54445
8               3.2  23.3   64445
9               3.7  23.6   57189
10              3.9  23.9   63218
11              4.0  24.0   55794
12              4.0  24.0   56957
13              4.1  24.0   57081
14              4.5  25.0   61111
15              4.9  25.0   67938
16              5.1  26.0   66029
```

Figura 3. Código para la adición del Data Frame

Posteriormente utilizamos el siguiente código, el cual sirve para seleccionar un subconjunto de filas y columnas del DataFrame **df** utilizando indexación basada en posiciones.

```
df.iloc[1::2, :3]
```

	YearsExperience	Age	Salary
1	1.3	21.5	46205
3	2.0	22.0	43525
5	2.9	23.0	56642
7	3.2	23.3	54445
9	3.7	23.6	57189
11	4.0	24.0	55794
13	4.1	24.0	57081
15	4.9	25.0	67938

Figura 4. Código para extracción de datos

Ahora usamos el siguiente código para separar los datos del DataFrame **df** en dos conjuntos distintos: **df_izq** y **df_der**. La variable **df_izq** almacena las filas en posiciones pares (**0::2**), es decir, aquellas cuyo índice es 0, 2, 4, etc.

```
df_izq = df[0::2]  # Filas pares
df_der = df.iloc[1::2, :3]  # Filas impares, primeras 3 columnas
```

Figura 5. Código para ordenamiento de datos

A continuación, usamos el siguiente código para ombina los DataFrames **df_izq** y **df_der** en un nuevo DataFrame llamado **df_new** usando la función **pd.concat()**. Primero, ambos DataFrames se reinician con **reset_index(drop=True)** para eliminar los índices originales y evitar inconsistencias en la unión. Luego, se concatenan a lo largo del eje de las columnas (**axis=1**), organizando sus datos en columnas adicionales.

```
df_new = pd.concat([df_izq.reset_index(drop=True), df_der.reset_index(drop=True)], axis=1, ignore_index=True)
df_new
```

	0	1	2	3	4	5
0	1.1	21.0	39343	1.3	21.5	46205
1	1.5	21.7	37731	2.0	22.0	43525
2	2.2	22.2	39891	2.9	23.0	56642
3	3.0	23.0	60150	3.2	23.3	54445
4	3.2	23.3	64445	3.7	23.6	57189
5	3.9	23.9	63218	4.0	24.0	55794
6	4.0	24.0	56957	4.1	24.0	57081
7	4.5	25.0	61111	4.9	25.0	67938
8	5.1	26.0	66029	5.3	27.0	83088
9	5.9	28.0	81363	6.0	29.0	93940
10	6.8	30.0	91738	7.1	30.0	98273

Figura 6. Código para resetea índices y concatenar

A continuación, se renombran las columnas del DataFrame **df_new** asignando nombres descriptivos para diferenciar los conjuntos de datos combinados. Las primeras tres columnas (**"YearsExperience_1"**, **"Age_1"**, **"Salary_1"**) corresponden a los datos de las filas pares del DataFrame original, mientras que las

siguientes tres (**"YearsExperience_2"**, **"Age_2"**, **"Salary_2"**) pertenecen a las filas impares.

```
df_new.columns = ["YearsExperience_1", "Age_1", "Salary_1", "YearsExperience_2", "Age_2", "Salary_2"]
print(df_new.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15 entries, 0 to 14
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   YearsExperience_1  15 non-null     float64
 1   Age_1              15 non-null     float64
 2   Salary_1           15 non-null     int64  
 3   YearsExperience_2  15 non-null     float64
 4   Age_2              15 non-null     float64
 5   Salary_2           15 non-null     int64  
dtypes: float64(4), int64(2)
memory usage: 852.0 bytes
None
```

Figura 7. Asignación de nombres a las columnas

Ahora, se aplica la función **pd.to_numeric** a todo el DataFrame **df_new** mediante **apply()**, lo que convierte los valores de cada columna a un tipo numérico si aún no lo son. Esto garantiza que los datos sean tratados correctamente para operaciones matemáticas y análisis estadísticos.

```
df_new = df_new.apply(pd.to_numeric)
```

Figura 8. Código para convertir los datos en valores numéricos

Finalmente, el DataFrame df_new se guarda en un archivo CSV llamado *"datos_salary.csv"* usando el método to_csv(). El parámetro index=False asegura que no se guarde la columna de índices en el archivo, y encoding="utf-8" se utiliza para asegurar que el archivo se guarde con una codificación compatible con caracteres especiales. Después, se imprime nuevamente el resumen del DataFrame con print(df_new.info()) para verificar que los cambios se han aplicado correctamente y para obtener información adicional sobre el contenido del archivo exportado.

```
df_new.to_csv("datos_salary.csv", index=False, encoding="utf-8")
print(df_new.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15 entries, 0 to 14
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   YearsExperience_1  15 non-null     float64
 1   Age_1              15 non-null     float64
 2   Salary_1           15 non-null     int64  
 3   YearsExperience_2  15 non-null     float64
 4   Age_2              15 non-null     float64
 5   Salary_2           15 non-null     int64  
dtypes: float64(4), int64(2)
memory usage: 852.0 bytes
None
```

Figura 9. Guardado de datos finales

A continuación, se utiliza el método **df.describe()** para generar un resumen estadístico de las columnas numéricas del DataFrame **df**. Este resumen incluye estadísticas como la media, la desviación estándar, el valor mínimo, el cuartil 25, la mediana (cuartil 50), el cuartil 75 y el valor máximo. Esta función es útil para obtener una visión general rápida de la distribución y las características estadísticas de los datos.

```
df.describe()
```

	YearsExperience	Age	Salary
count	30.000000	30.000000	30.000000
mean	5.313333	27.216667	76003.000000
std	2.837888	5.161267	27414.429785
min	1.100000	21.000000	37731.000000
25%	3.200000	23.300000	56720.750000
50%	4.700000	25.000000	65237.000000
75%	7.700000	30.750000	100544.750000
max	10.500000	38.000000	122391.000000

Figura 10. Descripción del DataFrame

A continuación, se crea un gráfico de distribución para la columna "Salary" del DataFrame df. Primero, se ajusta el tamaño de la figura con plt.figure(figsize=(8, 5)). Luego, sns.histplot() de la biblioteca Seaborn genera un histograma con 15 intervalos (bins=15) y una curva de densidad (kde=True) sobre los datos de salario, utilizando el color azul.

```
plt.figure(figsize=(8, 5))
sns.histplot(df["Salary"], bins=15, kde=True, color="blue")
plt.xlabel("Salario")
plt.ylabel("Frecuencia")
plt.title("Distribución de los Salarios")
plt.show()
```

Figura 11. Código para la creación de una grafica del DataFrame

De la siguiente grafica podemos obtener las siguientes conclusiones:

- Existe una tendencia positiva entre las tres variables (experiencia, edad y salario). A medida que una persona gana experiencia, su edad también aumenta, y con ambos factores, su salario tiende a subir.
- Las personas más jóvenes (alrededor de los 21 años) tienden a tener salarios más bajos, mientras que aquellos con más experiencia (alrededor de los 30 a 40 años) tienen salarios significativamente más altos. Esto sugiere que la acumulación de experiencia y años en el mercado laboral contribuye directamente a mayores ingresos.
- Algunas personas con menos experiencia (por ejemplo, alrededor de 2-3 años de experiencia) tienen salarios que pueden superar a los de aquellos con más experiencia, lo que podría indicar la influencia de otros factores como el sector laboral, la educación o la ubicación.

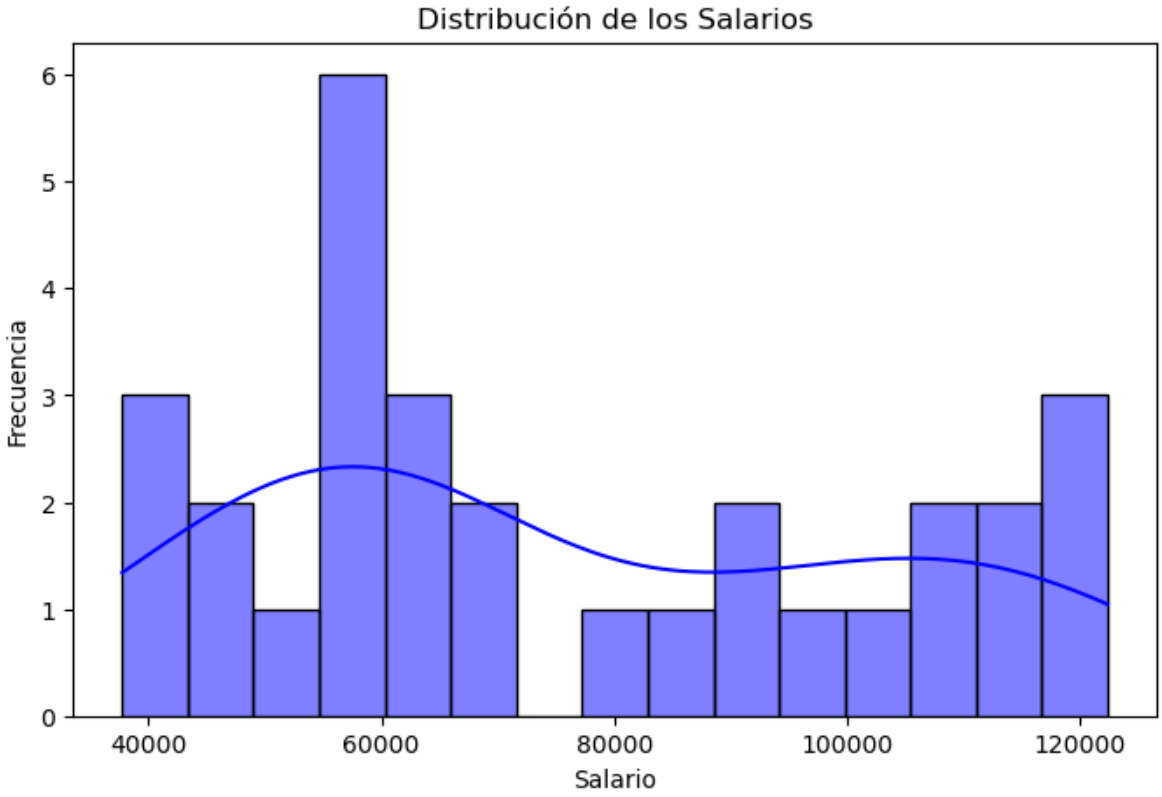

Figura 12. Distribución de los salarios

A continuación, se calcula la matriz de correlación del DataFrame df utilizando df.corr(), lo que genera una tabla que muestra el grado de relación lineal entre las variables numéricas del DataFrame.

```
correlation_matrix = df.corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Matriz de Correlación")
plt.show()
```

Figura 13. Código para grafica de correlación de valores

La gráfica obtenida es un mapa de calor que representa la matriz de correlación entre las variables YearsExperience, Age y Salary. Los colores más cálidos (rojos) indican una fuerte correlación positiva, mientras que los colores más fríos (azules) indican una débil o nula correlación. En este caso, es probable que veas una correlación positiva bastante fuerte entre YearsExperience y Salary, así como entre Age y Salary, lo que sugiere que tanto la experiencia como la edad están

relacionadas con el salario. Las celdas que muestran la correlación entre una variable consigo misma (diagonal) tendrán un valor de 1.00, lo que indica una correlación perfecta.

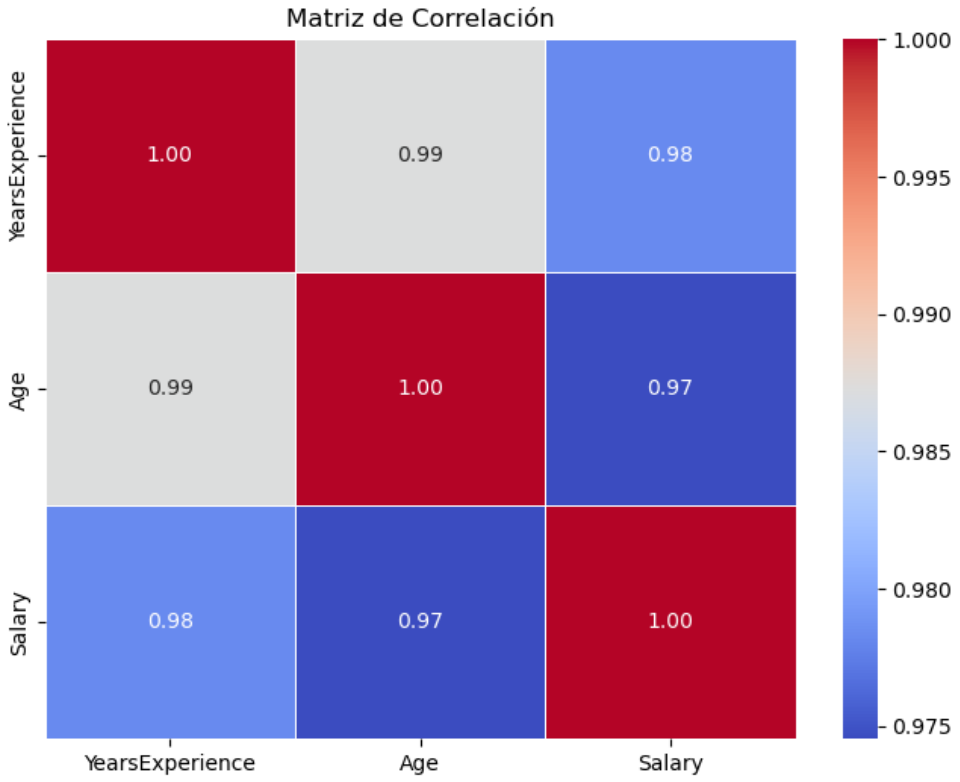

Figura 14. Grafica de correlación de la distribución de salarios

A continuación, se seleccionan las características 'YearsExperience' y 'Age' como las variables predictoras (independientes) y 'Salary' como la variable objetivo (dependiente). Luego, se crea una figura con dos subgráficos usando plt.subplot(), uno para cada característica. En cada subgráfico, se dibuja un gráfico de dispersión de los datos usando plt.scatter(), mostrando cómo la variable x (experiencia o edad) se relaciona con el salario y.

```python
import numpy as np
import matplotlib.pyplot as plt
# Seleccionar las características y la variable objetivo
features = ['YearsExperience', 'Age']
target = df['Salary']
# Crear la figura con dos subgráficos
plt.figure(figsize=(20, 5))
for i, col in enumerate(features):
    plt.subplot(1, len(features), i + 1)
    x = df[col]
    y = target
    # Dibujar los puntos de dispersión
    plt.scatter(x, y, marker='o', alpha=0.5, label="Datos")
    # Ajustar y dibujar la línea de tendencia (regresión lineal)
    coef = np.polyfit(x, y, 1)  # Ajuste de una recta (grado 1)
    poly1d_fn = np.poly1d(coef)  # Función de la recta obtenida
    plt.plot(x, poly1d_fn(x), color="red", label="Línea de tendencia")
    plt.title(f'Relación entre {col} y Salario')
    plt.xlabel(col)
    plt.ylabel('Salario')
    plt.legend()
plt.show()
```

Figura 15. Código parala grafica de dispersión de línea de tendencia

La gráfica obtenida presenta dos subgráficos, uno para cada una de las características seleccionadas (YearsExperience y Age) en relación con el Salario. En cada subgráfico, se visualiza una dispersión de puntos que muestra cómo varía el salario en función de la experiencia o la edad. Además, sobre los puntos se dibuja una línea de tendencia (regresión lineal) en color rojo, la cual muestra la dirección y la fuerza de la relación entre las variables. Es probable que la línea de tendencia para YearsExperience tenga una pendiente positiva, indicando que, a mayor experiencia, mayor salario. De manera similar, la relación entre Age y Salary podría mostrar una tendencia positiva, aunque tal vez de forma menos

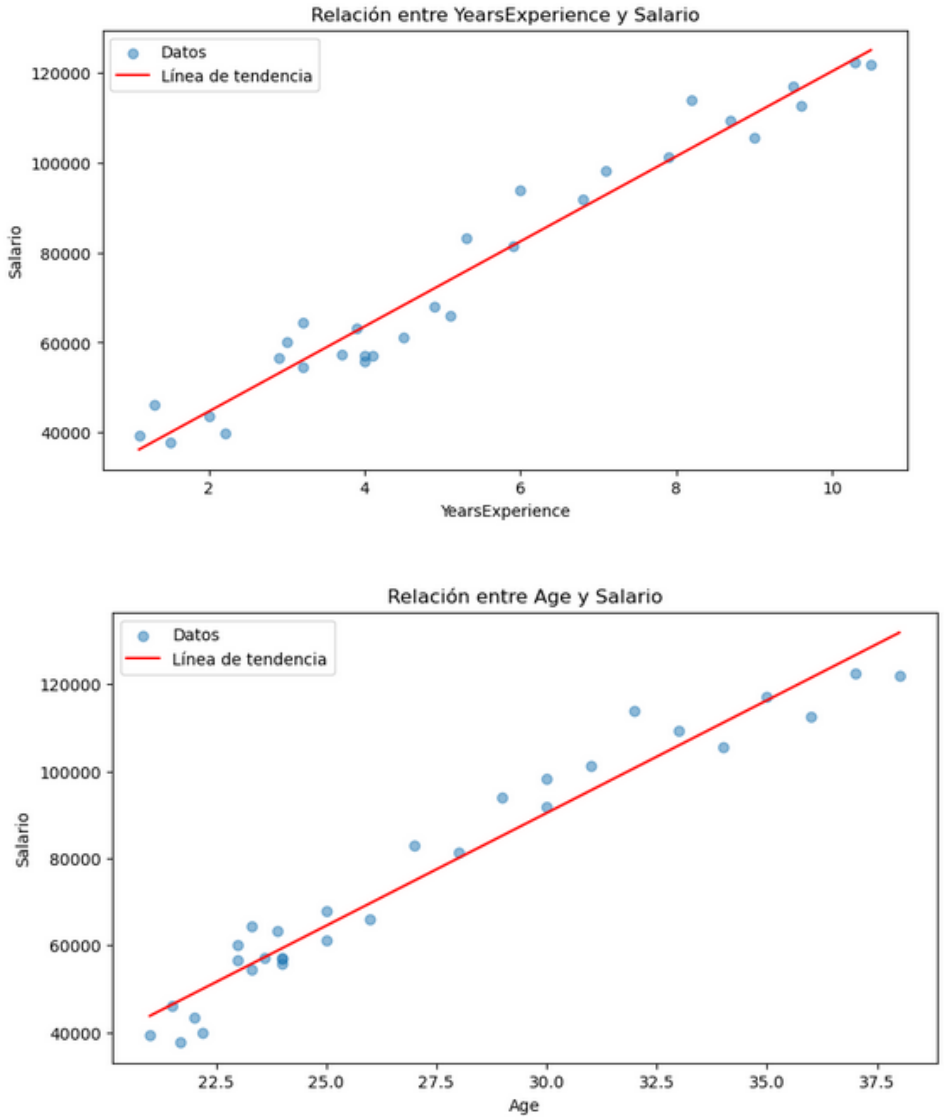

Figura 16. Graficas de dispersión de línea para la distribución de salario

A continuación, se importan las métricas de evaluación necesarias para medir el rendimiento del modelo de regresión lineal, específicamente mean_absolute_error, mean_squared_error y r2_score desde la librería sklearn.metrics. Se seleccionan las columnas "YearsExperience_1", "Age_1", "YearsExperience_2" y "Age_2" como las variables predictoras X, y "Salary_1" como la variable objetivo y. Luego, se divide el conjunto de datos en un conjunto de entrenamiento y otro de prueba utilizando train_test_split(). Posteriormente, se entrena el modelo de regresión lineal con model.fit(), se realizan las predicciones sobre el conjunto de prueba y se calculan las métricas de evaluación: el Error Absoluto Medio (MAE), el Error Cuadrático

Medio (MSE), la Raíz del Error Cuadrático Medio (RMSE) y el Coeficiente de Determinación (R^2). Finalmente, se organizan y muestran estas métricas en un DataFrame.

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
X = df_new[["YearsExperience_1", "Age_1", "YearsExperience_2", "Age_2"]]  # Variables predictoras
y = df_new["Salary_1"]  # Variable objetivo (usando Salary_1, puedes cambiar a "Salary_2" si es necesario)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
# Realizar predicciones
y_pred = model.predict(X_test)
# Evaluar el modelo
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
# Mostrar métricas de evaluación
metrics = pd.DataFrame({
    "Métrica": ["Error Absoluto Medio (MAE)", "Error Cuadrático Medio (MSE)",
                "Raíz del Error Cuadrático Medio (RMSE)", "Coeficiente de Determinación (R²)"],
    "Valor": [mae, mse, rmse, r2]
})
print("Métricas de Evaluación del Modelo")
print(metrics)
```

Figura 17. Código para la evaluación del modelo

Al ejecutar el código, se entrenará un modelo de regresión lineal utilizando las características seleccionadas para predecir el salario en función de la experiencia y la edad. Posteriormente, se evaluará el rendimiento del modelo mediante varias métricas de error, como el MAE, MSE, RMSE y R^2. El MAE proporcionará una medida del error promedio en las predicciones, mientras que el MSE y RMSE medirán la magnitud del error cuadrático. El R^2 indicará cuán bien el modelo explica la variabilidad de los datos. Al final, se imprimirá un DataFrame que contiene los valores de estas métricas, lo que permitirá evaluar la precisión y la capacidad predictiva del modelo de regresión lineal.

```
Métricas de Evaluación del Modelo
                                Métrica         Valor
0             Error Absoluto Medio (MAE)  2.774004e+03
1           Error Cuadrático Medio (MSE)  9.353086e+06
2  Raíz del Error Cuadrático Medio (RMSE)  3.058282e+03
3       Coeficiente de Determinación (R²)  9.859754e-01
```

Figura 18. Métricas de Evaluación del Modelo

A continuación, se crea una figura para visualizar la comparación entre los valores reales y los valores predichos por el modelo. Se utiliza plt.scatter() para graficar los puntos de dispersión, donde el eje x representa los valores reales y_test y el eje y los valores predichos y_pred.

```
# Crear la figura
plt.figure(figsize=(8, 6))

# Graficar los valores reales vs. predichos
plt.scatter(y_test, y_pred, alpha=0.5, color="blue", label="Predicciones")

# Graficar la línea ideal (y = x)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], '--', color='red', label="Ideal")

# Etiquetas y título
plt.xlabel("Valor Real (Salary_1)")
plt.ylabel("Valor Predicho (Salary_1)")
plt.title("Comparación de Valores Reales vs. Predichos")
plt.legend()

# Mostrar la gráfica
plt.show()
```

Figura 19. Código para graficar los valores reales contra los predichos

Al ejecutar este código, se obtendría un gráfico de dispersión que muestra cómo se comparan los valores reales del salario (y_test) con los valores predichos por el modelo (y_pred). Los puntos de la gráfica mostrarán la relación entre estas dos variables, y si el modelo es preciso, los puntos deberían alinearse cerca de la línea roja discontinua (y = x), que representa la coincidencia perfecta entre los valores reales y los predichos. Si los puntos se desvían considerablemente de esta línea, indicaría que el modelo no está realizando predicciones precisas. El gráfico es útil para evaluar visualmente la precisión del modelo en la predicción del salario.

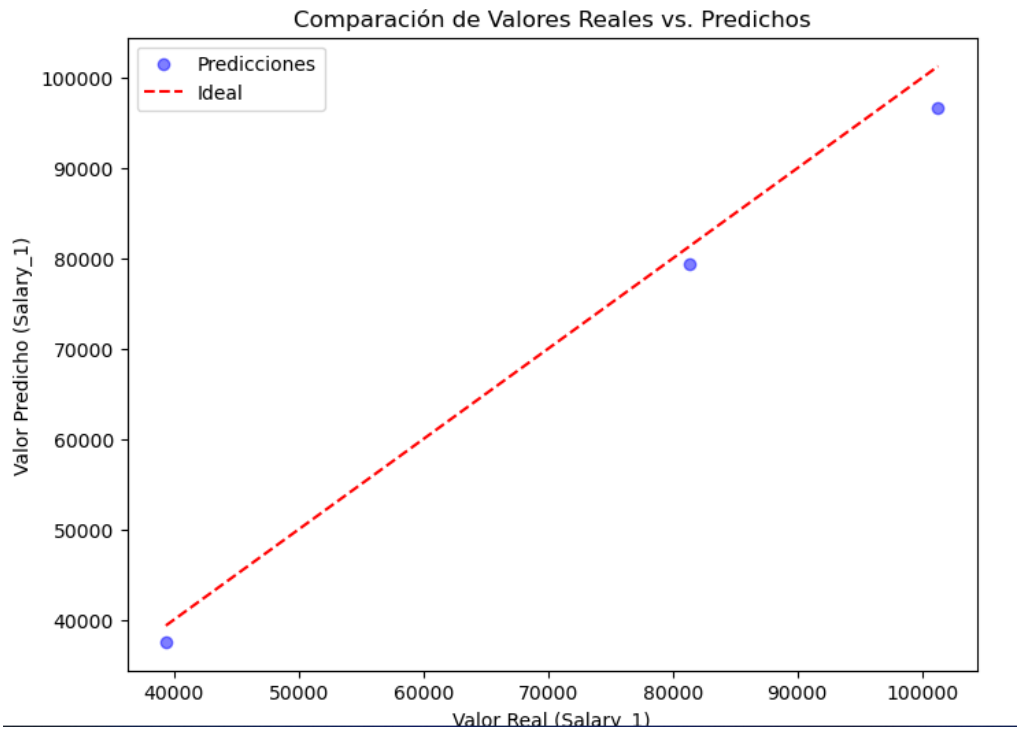

Figura 20. Comparación de valores reales contra predichos

A continuación, se importan las librerías necesarias para trabajar con modelos de regresión lineal, árboles de decisión y bosques aleatorios, junto con las métricas de evaluación requeridas. Se seleccionan las características "YearsExperience_1", "Age_1", "YearsExperience_2" y "Age_2" como las variables predictoras X, y "Salary_1" como la variable objetivo y. Después, se divide el conjunto de datos en entrenamiento y prueba utilizando train_test_split(). Se entrena un modelo de árbol de decisión con DecisionTreeRegressor(), se realiza la predicción de los valores de salario y se evalúa el modelo calculando las métricas MAE, MSE, RMSE y R^2. Estas métricas se organizan en un DataFrame para su visualización. Posteriormente, se repite el proceso con un modelo de Random Forest (RandomForestRegressor()), y se evalúan las métricas correspondientes para este modelo también.

```python
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
X = df_new[["YearsExperience_1", "Age_1", "YearsExperience_2", "Age_2"]]  # Variables predictoras
y = df_new["Salary_1"]  # Variable objetivo (usando Salary_1, puedes cambiar a "Salary_2" si es necesario)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
tree_model = DecisionTreeRegressor(random_state=42)
tree_model.fit(X_train, y_train)
y_tree_pred = tree_model.predict(X_test)
mae_tree = mean_absolute_error(y_test, y_tree_pred)
mse_tree = mean_squared_error(y_test, y_tree_pred)
rmse_tree = np.sqrt(mse_tree)
r2_tree = r2_score(y_test, y_tree_pred)
tree_metrics = pd.DataFrame({
    "Métrica": ["Error Absoluto Medio (MAE)", "Error Cuadrático Medio (MSE)",
                "Raíz del Error Cuadrático Medio (RMSE)", "Coeficiente de Determinación (R²)"],
    "Valor": [mae_tree, mse_tree, rmse_tree, r2_tree]
})
```

```python
print("Métricas del Modelo de Árbol de Decisión:")
print(tree_metrics)
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_rf_pred = rf_model.predict(X_test)
mae_rf = mean_absolute_error(y_test, y_rf_pred)
mse_rf = mean_squared_error(y_test, y_rf_pred)
rmse_rf = np.sqrt(mse_rf)
r2_rf = r2_score(y_test, y_rf_pred)
rf_metrics = pd.DataFrame({
    "Métrica": ["Error Absoluto Medio (MAE)", "Error Cuadrático Medio (MSE)",
                "Raíz del Error Cuadrático Medio (RMSE)", "Coeficiente de Determinación (R²)"],
    "Valor": [mae_rf, mse_rf, rmse_rf, r2_rf]
})
print("Métricas del Modelo de Random Forest:")
print(rf_metrics)
```

Figura 21. Código para la generación de métricas para el árbol de decisión

Al ejecutar el código, se entrenarán dos modelos de predicción para el salario: un árbol de decisión y un bosque aleatorio. Para cada modelo, el código calculará y mostrará las métricas de evaluación como el Error Absoluto Medio (MAE), Error Cuadrático Medio (MSE), Raíz del Error Cuadrático Medio (RMSE) y el Coeficiente de Determinación (R^2). Estas métricas indican el rendimiento de cada modelo: el MAE mide el error promedio de las predicciones, el MSE y RMSE dan una idea de la magnitud del error cuadrático, y el R^2 muestra la capacidad de los modelos para explicar la variabilidad de los datos. Al final, se imprimirá una tabla con estas

métricas para ambos modelos, lo que permitirá comparar cuál de los dos (árbol de decisión o bosque aleatorio) tiene un mejor desempeño en la predicción del salario.

Métricas del Modelo de Árbol de Decisión:

	Métrica	Valor
0	Error Absoluto Medio (MAE)	8.836667e+03
1	Error Cuadrático Medio (MSE)	1.097334e+08
2	Raíz del Error Cuadrático Medio (RMSE)	1.047537e+04
3	Coeficiente de Determinación (R^2)	8.354592e-01

Métricas del Modelo de Random Forest:

	Métrica	Valor
0	Error Absoluto Medio (MAE)	3.780363e+03
1	Error Cuadrático Medio (MSE)	2.611788e+07
2	Raíz del Error Cuadrático Medio (RMSE)	5.110566e+03
3	Coeficiente de Determinación (R^2)	9.608373e-01

Figura 22. Métricas para el modelo de árbol de decisión

Resultados

Interpretación del Valor de los Coeficientes

En el análisis de los coeficientes del modelo, aunque no se pueden interpretar directamente como en la regresión lineal, es importante considerar las métricas de evaluación para entender cómo el modelo se comporta. Para el modelo en cuestión, el Error Absoluto Medio (MAE) es de 2.774004e+03, lo que indica que, en promedio, las predicciones del modelo tienen un error absoluto de aproximadamente 2,774 unidades.

El Error Cuadrático Medio (MSE) es de 9.353086e+06, lo que refleja la magnitud del error cuadrado promedio y nos da una idea de la dispersión de los errores de predicción. El RMSE es de 3.058282e+03, lo que también muestra un error promedio en las predicciones de alrededor de 3,058 unidades, lo que podría considerarse un valor aceptable dependiendo del contexto del problema.

Explicación del Significado de la Métrica R^2

El **Coeficiente de Determinación R2** es una métrica que indica qué tan bien se ajusta el modelo a los datos. En otras palabras, nos dice qué porcentaje de la variabilidad en la variable objetivo (en este caso, Salary_1) es explicado por las variables predictoras (como YearsExperience_1, Age_1, etc.).

Valores de R^2:

- **Árbol de Decisión: 0.8355** (83.55%)
- **Random Forest: 0.9608** (96.08%)

Interpretación:

- $\boldsymbol{R^2 = 0.8355}$ para el modelo de Árbol de Decisión: Esto significa que el modelo de Árbol de Decisión puede explicar el 83.55% de la variabilidad en los salarios en función de las variables predictoras. Este es un valor bastante bueno, ya que generalmente un valor de R^2 por encima de 0.7 se considera un buen ajuste.
- $\boldsymbol{R^2 = 0.9608}$ para el modelo de Random Forest: Esto significa que el modelo de Random Forest puede explicar el 96.08% de la variabilidad en los salarios. Este es un valor aún mejor, lo que indica que el modelo de Random Forest se ajusta muy bien a los datos y es más preciso en comparación con el modelo de Árbol de Decisión.

Discusión sobre la Relación entre las Variables

Los Árboles de Decisión y Random Forest son modelos no lineales, pero, aun así, se ajustan bien a la variabilidad de los salarios en función de la experiencia y la edad. Sin embargo, el modelo de Random Forest muestra un ajuste significativamente mejor debido a su capacidad de capturar relaciones más complejas entre las variables.

Posibles Mejoras o Ajustes al Modelo

Preprocesamiento de Datos:

- **Escalado de variables**: Aunque los modelos de Árboles de Decisión y Random Forest no suelen requerir escalado de características, podría ser útil en otros modelos como la regresión lineal o en el caso de características muy desbalanceadas.

Mayor Conjunto de Datos:

- Si es posible, agregar más ejemplos de datos podría ayudar a mejorar el modelo, ya que el Random Forest y los Árboles de Decisión tienden a mejorar con más datos, especialmente si tienen suficiente variabilidad.

Análisis de Residuales:

- Se puede analizar los residuales (las diferencias entre las predicciones y los valores reales) para detectar patrones no capturados por el modelo, lo cual podría indicar que el modelo necesita ajustes adicionales o características adicionales.

Conclusión

En el análisis realizado, se ha evaluado el desempeño de dos modelos de predicción, el Árbol de Decisión y el Random Forest, para predecir el salario basado en variables como la experiencia y la edad. Ambos modelos mostraron un buen desempeño, con el modelo de Random Forest destacándose con un R^2 de 0.9608, lo que indica que puede explicar más del 96% de la variabilidad en los salarios. Por otro lado, el Árbol de Decisión presentó un R^2 de 0.8355, lo cual también es un resultado sólido, aunque menos preciso que el modelo de Random Forest. Esta diferencia refleja la capacidad superior de Random Forest para capturar relaciones complejas y no lineales entre las variables predictoras y el salario.

Además, se analizó la interpretación de las métricas de desempeño, como el Error Absoluto Medio (MAE), el Error Cuadrático Medio (MSE) y la Raíz del Error Cuadrático Medio (RMSE), los cuales mostraron que el modelo de Random Forest tiene un desempeño superior en cuanto a la precisión de las predicciones, con valores más bajos en todas las métricas de error. En contraste, el modelo de Árbol de Decisión mostró un mayor margen de error, lo que sugiere que, aunque es útil, su capacidad predictiva es limitada en comparación con Random Forest.

En conclusión, aunque ambos modelos son efectivos, el Random Forest se destaca como el modelo más preciso y adecuado para este tipo de problemas, dado su rendimiento superior en las métricas evaluadas. A pesar de ello, hay espacio para mejorar los modelos, mediante ajustes en los hiperparámetros, la inclusión de nuevas características o el uso de modelos más avanzados como XGBoost. La relación entre las variables predictoras y la variable objetivo es claramente fuerte, lo que sugiere que la experiencia y la edad son factores clave en la predicción del salario. Sin embargo, el análisis de residuos y la incorporación de más datos podrían mejorar aún más los modelos.