REALIZADO POR:

Ruelas Gonzalez Carlos Alexis

Materia:

Extraccion de conocimientos en bases de datos

Actividad

Implementacion en una Web del Modelo Entrenado de regresion lineal simple

Profesor(a):

Florencio López Cruz

Carrera:

Desarrollo y gestión de software

Cuatrimestre y grupo:

9° B.

2 de noviembre del 2025

Implementacion en una Web del Modelo Entrenado de regresion lineal simple

Paso 1: Importación de librerías

```
#Analitica
import numpy as np
import pandas as pd
#Importar libreria
import joblib
#Visualizacion de datos
import matplotlib.pyplot as plt
#Importar la lubreia de division de datos de entrenamiento y prueba
from sklearn.model_selection import train_test_split
#Importando la libreria de regresion lineal
from sklearn.linear_model import LinearRegression
#Importar libreria de metricas del modelo de prediccion
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

Se importan las librerías necesarias de Python para tener todas las herramientas listas para el análisis y el entrenamiento del modelo

Paso 2: Cargamos nuestro dataset

```
data_salary = pd.read_csv('C:/Users/kbaio/Documents/Datasets/SalaryData.csv')
```

```
data_salary
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0
5	2.9	56642.0
6	3.0	60150.0
7	3.2	54445.0
8	3.2	64445.0
9	3.7	57189.0
10	3.9	63218.0
11	4.0	55794.0
12	4.0	56957.0
13	4.1	57081.0
14	4.5	61111.0
15	4.9	67938.0
16	5.1	66029.0
17	5.3	83088.0
18	5.9	81363.0
19	6.0	93940.0
20	6.8	91738.0
21	7.1	98273.0

Se carga el conjunto de datos (por ejemplo, desde un archivo CSV).

Paso 3: Extracción de datos para ‘x’ y ‘y’

```
#Extraccion de datos en las columnas x, y
x = data_salary.iloc[:,:-1].values # El -1 en python es agarrar el ultimo elemento
y = data_salary.iloc[:,1].values   # La posicion 1
```

```
#Años de experiencia
x
```

```
array([[ 1.1],
       [ 1.3],
       [ 1.5],
       [ 2. ],
       [ 2.2],
       [ 2.9],
       [ 3. ],
       [ 3.2],
       [ 3.2],
       [ 3.7],
       [ 3.9],
       [ 4. ],
       [ 4. ],
       [ 4.1],
       [ 4.5],
       [ 4.9],
       [ 5.1],
       [ 5.3],
       [ 5.9],
       [ 6. ],
       [ 6.8],
       [ 7.1],
       [ 7.9],
       [ 8.2],
       [ 8.7],
       [ 9. ],
       [ 9.5],
       [ 9.6],
       [10.3],
       [10.5]])
```

```
#Salario
y
```

```
array([ 39343.,  46205.,  37731.,  43525.,  39891.,  56642.,  60150.,
        54445.,  64445.,  57189.,  63218.,  55794.,  56957.,  57081.,
        61111.,  67938.,  66029.,  83088.,  81363.,  93940.,  91738.,
        98273., 101302., 113812., 109431., 105582., 116969., 112635.,
       122391., 121872.])
```

Se seleccionan las variables independientes (x) y dependientes (y). Con el objetivo de definir qué variable se usará para predecir y cuál será el resultado esperado.

Paso 4: Dividimos datos de entrenamiento y prueba (80 y 20)

```
#Dividir los datos de entrenaimento y prueba 80% entrenamiento y 20% de prueba
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.1)
```

Se dividen los datos en dos grupos:

- Entrenamiento (80%): para que el modelo aprenda.
- Prueba (20%): para evaluar qué tan bien generaliza el modelo.

Paso 5: Construimos y cargamos nuestros modelos

```
#Construir el modelo de regresion lineal
reg = LinearRegression()
```

```
reg.fit(x_train,y_train)
```

LinearRegression
Parameters

```
LinearRegression()
```

LinearRegression
Parameters

Se crea el modelo de regresión lineal y se entrena con los datos de entrenamiento

Paso 6: Generamos nuestros datos de predicción con los datos de prueba

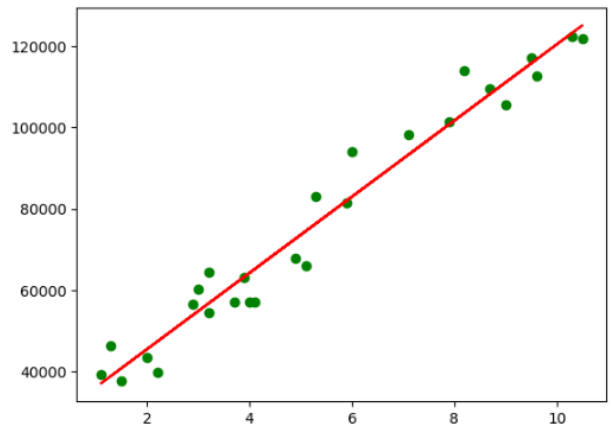
```
y_pred = reg.predict(x_test)
x_pred = reg.predict(x_train)
```

Se usa el modelo entrenado para hacer predicciones sobre el conjunto de prueba

Paso 7: Graficamos con una gráfica de regresión

```
print("Training Set")
plt.scatter(x_train, y_train, color="green")
plt.plot(x_train,x_pred,color="red")
plt.show()
```

```
Training Set
```



Se genera una gráfica que muestra:

- Los puntos reales (datos observados).
- La línea de regresión generada por el modelo.

Paso 8: Generamos métricas de evaluación

```
#Metricas de evaluacion
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)
```

Se calculan métricas como:

- Error cuadrático medio (MSE)
- Error absoluto medio (MAE)
- Coeficiente de determinación (R^2)

Paso 9: Generamos distintos datos de prueba

```
print("===Metrica de evaluacion del modelo===")
print(f"MAE (Error Absoluto Medio): {mae:.2f}")
print(f'MSE (Error Cuadratico Medio): {mse:.2f}')
print(f'RMSE (Raiz del Error Cuadrativo Medio): {mae:.2f}')
print(f"R2 Score: {r2:.4f}")
```

```
===Metrica de evaluacion del modelo===
MAE (Error Absoluto Medio): 5828.01
MSE (Error Cuadratico Medio): 44221975.27
RMSE (Raiz del Error Cuadrativo Medio): 5828.01
R2 Score: 0.8238
```

R= Es una métrica estadística que mide qué tan bien el modelo de regresión logra explicar la variabilidad de los datos observados.

```
#Prueba el modelo
print("Prueba del modelo entrenado")

#Valores de ejemplo para probar
valores_ejemplo = [1, 3, 5, 8, 12]

for valor in valores_ejemplo:
    prediccion = reg.predict([[valor]])[0]
    print(f" - Si X = {valor:2d} -> Salario predicho: ${prediccion:,.2f}")
```

```
Prueba del modelo entrenado
 - Si X =  1 -> Salario predicho: $36,093.93
 - Si X =  3 -> Salario predicho: $54,828.53
 - Si X =  5 -> Salario predicho: $73,563.13
 - Si X =  8 -> Salario predicho: $101,665.04
 - Si X = 12 -> Salario predicho: $139,134.24
```

Se pueden ingresar nuevos valores de x para predecir nuevos resultados

Paso 10: Guardamos nuestro modelo preentrenado

```
#Guardar el modelo entrenado
joblib.dump(reg, ('modelo_salario.pkl'))
print("Modelo guardado como 'modelo_salario.pkl'")

# Funcion para cargar y usar el modelo posteriormente entrenado
def cargarModeloYPredecir(x_val, ruta_modelo='modelo_salario.pkl'):
    modelo = joblib.load(ruta_modelo)
    prediccion = modelo.predict([[x_val]])[0]
    return prediccion
```

```
Modelo guardado como 'modelo_salario.pkl'
```

Finalmente, se guarda el modelo para usarlo después en una aplicación web

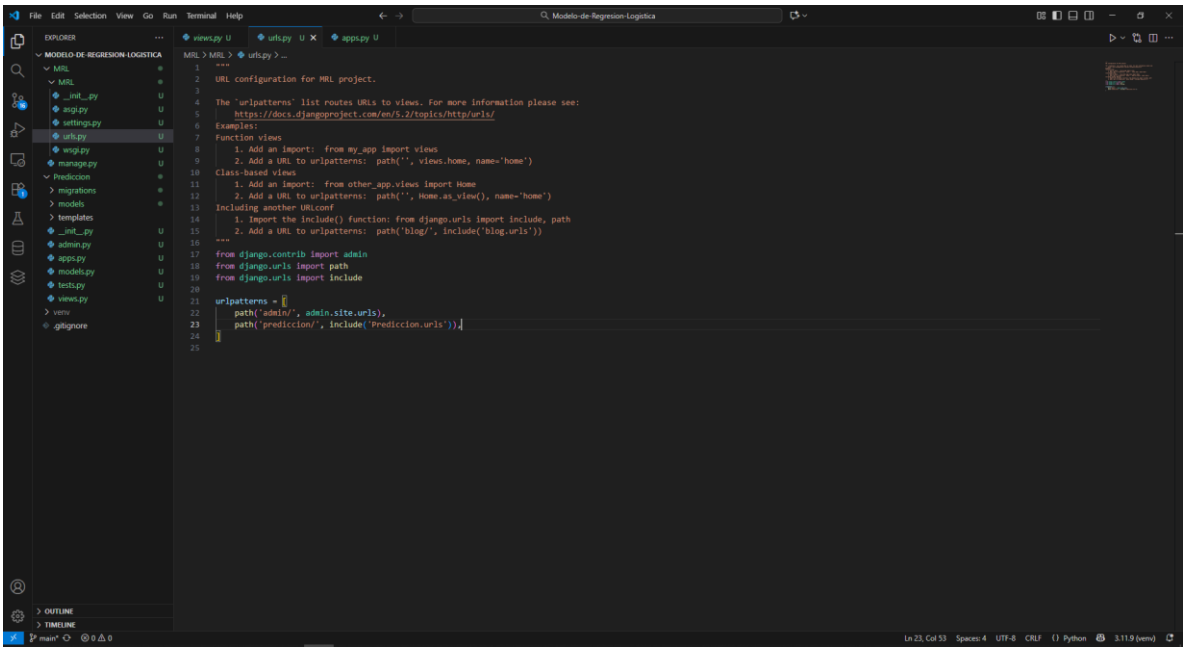
Implementacion web

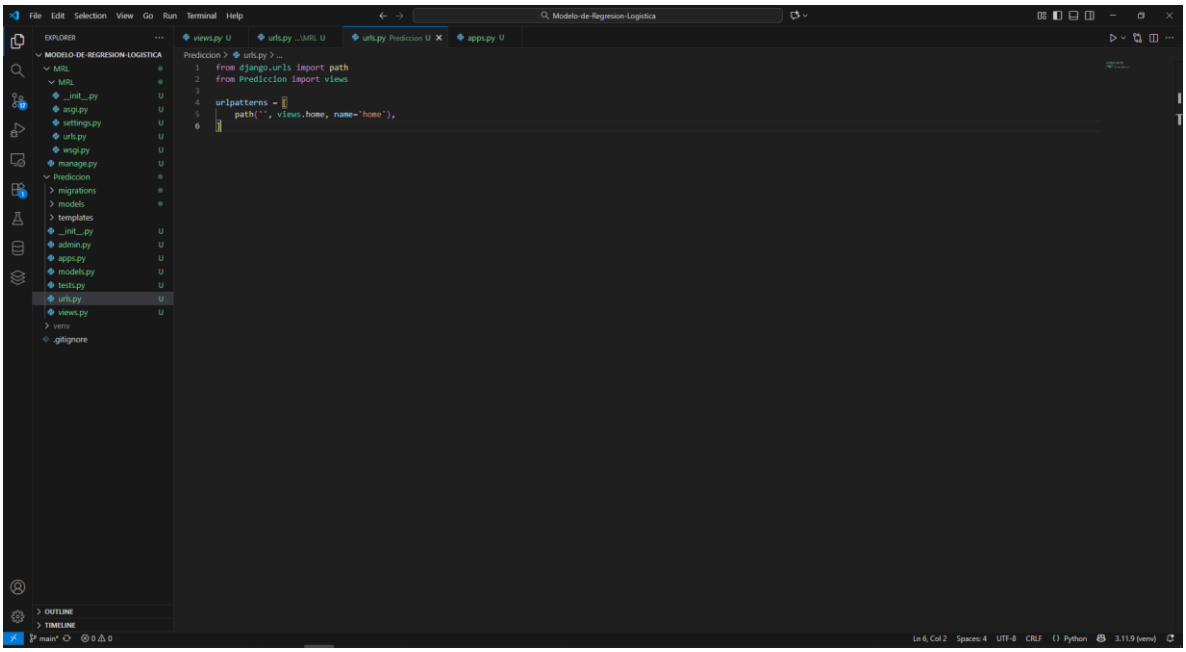
1. Creamos nuestra aplicación web

En este caso se usará el framework de Python django

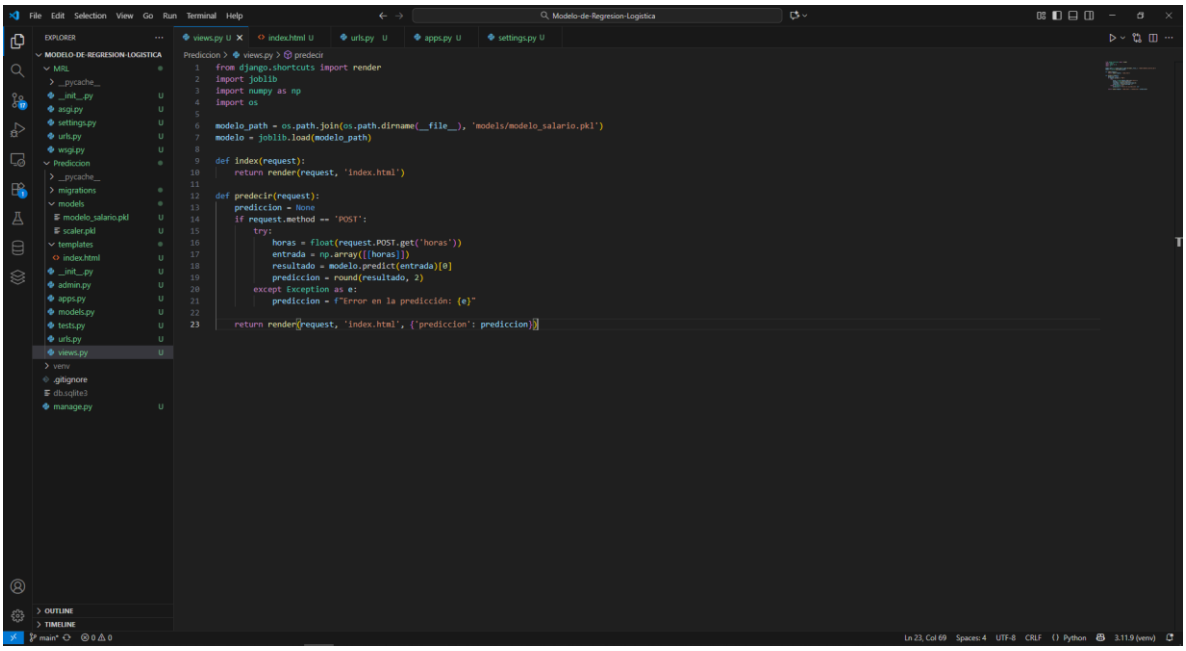
```
baio@DESKTOP-J6ID3GO MINGW64 ~/Documents/Github/Modelo-de-Regresion-Logistica (
ain)
 django-admin startproject MRL
venv)
baio@DESKTOP-J6ID3GO MINGW64 ~/Documents/Github/Modelo-de-Regresion-Logistica (
ain)
 django-admin startapp Predicción
```

2. Ingresamos nuestros urls

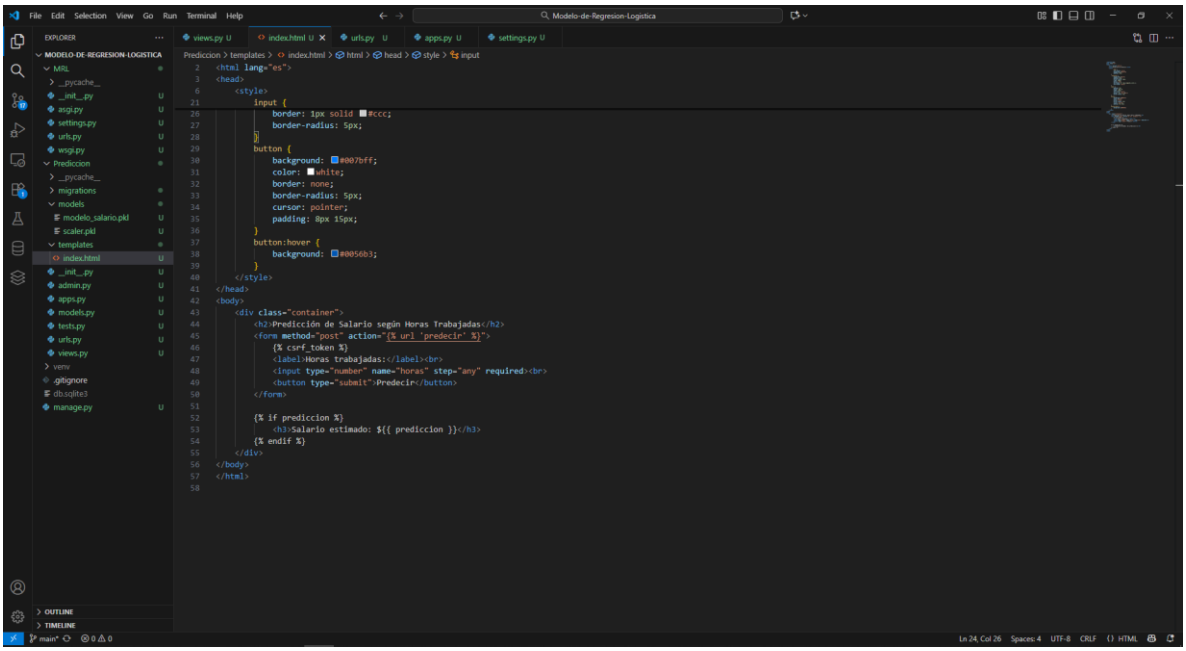




3. Creamos nuestra vista



4. Generamos nuestro html




```
<html lang="es">
<head>
    <style>
        input {
            border: 1px solid #ccc;
            border-radius: 5px;
        }
        button {
            background: #007bff;
            color: white;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            padding: 8px 15px;
        }
        button:hover {
            background: #0056b3;
        }
    </style>
</head>
<body>
    <div class="container">
        <h2>Predicción de Salario según Horas Trabajadas</h2>
        <form method="post" action="{% url 'predecir' %}">
            {% csrf_token %}
            <label>Horas trabajadas:</label><br>
            <input type="number" name="horas" step="any" required><br>
            <button type="submit">Predecir</button>
        </form>

        {% if prediccion %}
            <h3>Salario estimado: ${{ prediccion }}</h3>
        {% endif %}
    </div>
</body>
</html>
```

5. Probamos en nuestro localhost