📊 Clasificación de Ingresos con Spark ML - Google Colab

🎯 DataPros - Predicción de Ingresos >50K

Objetivo: Construir un modelo de clasificación binaria con Spark ML utilizando Logistic Regression para predecir si una persona gana más de 50K al año o no.

Dataset: adult_income_sample.csv con 2000 registros simulados (cargado desde Google Drive)

Características:

- `age`: Edad de la persona (años)
- `sex`: Género (Male, Female)
- `workclass`: Tipo de empleo (Private, Self-emp, Gov)
- `fnlwgt`: Peso estadístico asociado al registro
- `education`: Nivel educativo (Bachelors, HS-grad, 11th, Masters, etc.)
- `hours_per_week`: Horas trabajadas por semana
- `label`: Clase objetivo (>50K o <=50K)

INSTRUCCIONES IMPORTANTES:

Antes de ejecutar este notebook:

1. **Sube el archivo `adult_income_sample.csv` a tu Google Drive**
2. **Ejecuta la primera celda para montar Google Drive**
3. **Verifica que el archivo esté en la ruta correcta**
4. **Ejecuta todas las celdas secuencialmente**

1. Configuración del Entorno Google Colab

1. Configuración del Entorno Google Colab

Instalamos PySpark y montamos Google Drive para acceder al dataset.

```
# Instalar PySpark en Google Colab
!pip install pyspark

# Montar Google Drive
from google.colab import drive
drive.mount('/content/drive')

print(" PySpark instalado y Google Drive montado")
```

```
Requirement already satisfied: pyspark in /usr/local/lib/python3.12/dist-packages (3.5.1)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.12/dist-packages (from pyspark) (0.10.9
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", for
 PySpark instalado y Google Drive montado
```

```
# Importar librerías necesarias
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, IntegerType
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml import Pipeline
from pyspark.sql.functions import col
import os

print(" Librerías importadas correctamente")
```

```
 Librerías importadas correctamente
```

```
# Crear sesión de Spark optimizada para Google Colab
spark = SparkSession.builder \
    .appName("IncomeClassification_Colab") \
    .master("local[*]") \
```

```
    .config("spark.sql.adaptive.enabled", "false") \
    .config("spark.sql.adaptive.coalescePartitions.enabled", "false") \
    .config("spark.driver.memory", "2g") \
    .config("spark.executor.memory", "1g") \
    .getOrCreate()

# Configurar nivel de log para reducir salida
spark.sparkContext.setLogLevel("ERROR")

print(" Sesión Spark iniciada en Google Colab")
print(f" Versión de Spark: {spark.version}")
print(f" Aplicación: {spark.sparkContext.appName}")
print(f"  Master: {spark.sparkContext.master}")
```

```
 Sesión Spark iniciada en Google Colab
 Versión de Spark: 3.5.1
 Aplicación: IncomeClassification_Colab
  Master: local[*]
```

📂 2. Carga y Exploración de Datos desde Google Drive

Cargamos el archivo CSV desde Google Drive y exploramos la estructura de los datos.

```
# Verificar que el archivo existe en Google Drive
file_path = "/content/drive/MyDrive/adult_income_sample.csv"

if os.path.exists(file_path):
    print(f" Archivo encontrado: {file_path}")
    print(f" Tamaño del archivo: {os.path.getsize(file_path)} bytes")
else:
    print("ARCHIVO NO ENCONTRADO!")
    print("Archivos disponibles en Google Drive:")
    !ls "/content/drive/MyDrive/" | head -10
    print("\n  Por favor, sube 'adult_income_sample.csv' a tu Google Drive")
    print(" O ajusta la ruta en la variable 'file_path'")
```

```
 Archivo encontrado: /content/drive/MyDrive/adult_income_sample.csv
 Tamaño del archivo: 81264 bytes
```

```
# Cargar datos desde Google Drive
df = spark.read.csv(file_path, header=True, inferSchema=True)

print(f" Datos cargados desde Google Drive: {df.count()} registros, {len(df.columns)} columnas")
print("\n Esquema del DataFrame:")
df.printSchema()
```

```
 Datos cargados desde Google Drive: 2000 registros, 7 columnas

 Esquema del DataFrame:
root
 |-- age: integer (nullable = true)
 |-- sex: string (nullable = true)
 |-- workclass: string (nullable = true)
 |-- fnlwgt: integer (nullable = true)
 |-- education: string (nullable = true)
 |-- hours_per_week: integer (nullable = true)
 |-- label: string (nullable = true)
```

```
# Mostrar primeros 10 registros
print(" Primeros 10 registros:")
df.show(10, truncate=False)
```

```
 Primeros 10 registros:
+---+------+---------+-------+------------+--------------+-----+
|age|sex   |workclass|fnlwgt |education   |hours_per_week|label|
+---+------+---------+-------+------------+--------------+-----+
|56 |Male  |Private  |683155 |Some-college|15            |>50K |
|60 |Male  |Gov      |122268 |HS-grad     |21            |<=50K|
|56 |Male  |Gov      |149337 |Doctorate   |75            |>50K |
|28 |Male  |Private  |924756 |Masters     |24            |>50K |
|20 |Male  |Gov      |341365 |Masters     |53            |<=50K|
|19 |Female|Private  |1382455|9th         |30            |<=50K|
|55 |Male  |Private  |1259617|Doctorate   |64            |>50K |
|38 |Male  |Private  |1274752|Assoc-voc   |76            |>50K |
```

```
|39 |Female|Private  |1001436|Assoc-voc   |49            |<=50K|
|44 |Female|Private  |168730 |9th         |42            |>50K |
+---+------+---------+-------+------------+--------------+-----+
only showing top 10 rows
```

```
# Análisis de la distribución de la variable objetivo
print(" Distribución de la variable objetivo (label):")
df.groupBy("label").count().orderBy("count", ascending=False).show()

print("\n Estadísticas descriptivas de variables numéricas:")
df.describe().show()
```

```
 Distribución de la variable objetivo (label):
+-----+-----+
|label|count|
+-----+-----+
|<=50K| 1026|
| >50K|  974|
+-----+-----+


 Estadísticas descriptivas de variables numéricas:
+-------+------------------+------+---------+-----------------+------------+-----------------+-----+
|summary|               age|   sex|workclass|           fnlwgt|   education|   hours_per_week|label|
+-------+------------------+------+---------+-----------------+------------+-----------------+-----+
|  count|              2000|  2000|     2000|             2000|        2000|             2000| 2000|
|   mean|           46.2575|  NULL|     NULL|      747752.3035|        NULL|          39.1105| NULL|
| stddev|16.487604868723448|  NULL|     NULL|425986.6185501128|        NULL|22.57086643786361| NULL|
|    min|                18|Female|      Gov|            12126|        10th|                1|<=50K|
|    max|                74|  Male| Self-emp|          1499550|Some-college|               79| >50K|
+-------+------------------+------+---------+-----------------+------------+-----------------+-----+
```

🔄 3. Preprocesamiento de Variables Categóricas

Aplicamos StringIndexer y OneHotEncoder para transformar las variables categóricas.

```
# Definir variables categóricas
categorical_cols = ['sex', 'workclass', 'education']

print(" Aplicando StringIndexer a variables categóricas...")

# StringIndexer para variables categóricas
indexers = []
for col_name in categorical_cols:
    indexer = StringIndexer(inputCol=col_name, outputCol=f"{col_name}_index")
    indexers.append(indexer)
    print(f"  ✓ StringIndexer creado para: {col_name}")

# StringIndexer para la variable objetivo (label)
label_indexer = StringIndexer(inputCol="label", outputCol="label_index")
indexers.append(label_indexer)
print(f"  ✓ StringIndexer creado para: label")

print(f"\n Total de StringIndexers: {len(indexers)}")
```

```
 Aplicando StringIndexer a variables categóricas...
  ✓ StringIndexer creado para: sex
  ✓ StringIndexer creado para: workclass
  ✓ StringIndexer creado para: education
  ✓ StringIndexer creado para: label

 Total de StringIndexers: 4
```

```
# OneHotEncoder para variables categóricas (no para la etiqueta)
print(" Aplicando OneHotEncoder a variables categóricas...")

encoders = []
for col_name in categorical_cols:
    encoder = OneHotEncoder(inputCol=f"{col_name}_index", outputCol=f"{col_name}_encoded")
    encoders.append(encoder)
    print(f"  ✓ OneHotEncoder creado para: {col_name}")

print(f"\n Total de OneHotEncoders: {len(encoders)}")
```

```
print(f" \n Total de OneHotEncoders: {len(encoders)} ")
print(" Preprocesamiento de variables categóricas configurado")
```

```
 Aplicando OneHotEncoder a variables categóricas...
  ✓ OneHotEncoder creado para: sex
  ✓ OneHotEncoder creado para: workclass
  ✓ OneHotEncoder creado para: education

 Total de OneHotEncoders: 3
 Preprocesamiento de variables categóricas configurado
```

🔧 4. Ensamblaje de Características

Combinamos todas las características en un vector único usando VectorAssembler.

```
# Definir características para el vector final
numeric_cols = ['age', 'fnlwgt', 'hours_per_week']
categorical_encoded_cols = ['sex_encoded', 'workclass_encoded', 'education_encoded']

# Combinar todas las características
feature_cols = numeric_cols + categorical_encoded_cols

print(" Configurando VectorAssembler...")
print(f" Características numéricas: {numeric_cols}")
print(f" Características categóricas codificadas: {categorical_encoded_cols}")
print(f" Total de características: {len(feature_cols)}")

# Crear VectorAssembler
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")

print("✅ VectorAssembler configurado")
```

```
 Configurando VectorAssembler...
 Características numéricas: ['age', 'fnlwgt', 'hours_per_week']
 Características categóricas codificadas: ['sex_encoded', 'workclass_encoded', 'education_encoded']
 Total de características: 6
✅ VectorAssembler configurado
```

🤖 5. Definición y Entrenamiento del Modelo

Configuramos el modelo de Logistic Regression y creamos el Pipeline completo.

```
# Configurar modelo de Logistic Regression
lr = LogisticRegression(
    featuresCol="features",
    labelCol="label_index",
    predictionCol="prediction",
    probabilityCol="probability",
    maxIter=100,
    regParam=0.01
)

print(" Modelo configurado:")
print(f"   Tipo: Logistic Regression")
print(f"   Máximo iteraciones: {lr.getMaxIter()}")
print(f"   Parámetro regularización: {lr.getRegParam()}")
print(" Modelo de Logistic Regression configurado")
```

```
 Modelo configurado:
   Tipo: Logistic Regression
   Máximo iteraciones: 100
   Parámetro regularización: 0.01
 Modelo de Logistic Regression configurado
```

```
# Crear Pipeline con todas las etapas
stages = indexers + encoders + [assembler, lr]
pipeline = Pipeline(stages=stages)

print(" Pipeline creado con las siguientes etapas:")
for i, stage in enumerate(stages):
    print(f"  {i+1:2d}. {type(stage).__name__}")
```

```
print(f"\n Total de etapas en el Pipeline: {len(stages)}")
print(" Pipeline configurado")
```

```
 Pipeline creado con las siguientes etapas:
   1. StringIndexer
   2. StringIndexer
   3. StringIndexer
   4. StringIndexer
   5. OneHotEncoder
   6. OneHotEncoder
   7. OneHotEncoder
   8. VectorAssembler
   9. LogisticRegression

 Total de etapas en el Pipeline: 9
 Pipeline configurado
```

```
# Entrenar el modelo
print(" Iniciando entrenamiento del modelo...")
model = pipeline.fit(df)
print(" Modelo entrenado exitosamente!")

# Obtener información del modelo entrenado
lr_model = model.stages[-1]  # El último stage es el LogisticRegression
print(f"\n Información del modelo entrenado:")
print(f"   Número de iteraciones realizadas: {lr_model.summary.totalIterations}")
print(f"  Objective History (últimos 5): {lr_model.summary.objectiveHistory[-5:]}")
print(" Entrenamiento completado")
```

```
 Iniciando entrenamiento del modelo...
 Modelo entrenado exitosamente!

 Información del modelo entrenado:
   Número de iteraciones realizadas: 12
  Objective History (últimos 5): [0.6483333925979682, 0.6483333467811659, 0.6483333419198362, 0.648333341653913
 Entrenamiento completado
```

📈 6. Evaluación del Modelo

9. Evaluación del modelo

Evaluamos el modelo entrenado y analizamos las predicciones.

```
# Hacer predicciones sobre el conjunto de entrenamiento
predictions = model.transform(df)

print(" Predicciones generadas")
print(" Mostrando las primeras 20 predicciones con todas las columnas relevantes:")

# Mostrar predicciones
predictions.select("age", "sex", "workclass", "education", "hours_per_week",
                   "label", "prediction", "probability").show(20, truncate=False)
```

```
 Predicciones generadas
 Mostrando las primeras 20 predicciones con todas las columnas relevantes:
+---+------+---------+------------+--------------+-----+----------+----------------------------------------+
|age|sex   |workclass|education   |hours_per_week|label|prediction|probability                             |
+---+------+---------+------------+--------------+-----+----------+----------------------------------------+
|56 |Male  |Private  |Some-college|15            |>50K |0.0       |[0.6759508529977439,0.32404914700225607]|
|60 |Male  |Gov      |HS-grad     |21            |<=50K|0.0       |[0.6529050112508005,0.34709498874919953]|
|56 |Male  |Gov      |Doctorate   |75            |>50K |1.0       |[0.2644233810243991,0.7355766189756009] |
|28 |Male  |Private  |Masters     |24            |>50K |1.0       |[0.26070053389144104,0.739299466108559] |
|20 |Male  |Gov      |Masters     |53            |<=50K|1.0       |[0.2516438156122881,0.7483561843877119] |
|19 |Female|Private  |9th         |30            |<=50K|1.0       |[0.47060749014458114,0.5293925098554189]|
|55 |Male  |Private  |Doctorate   |64            |>50K |1.0       |[0.2176909223071173,0.7823090776928827] |
|38 |Male  |Private  |Assoc-voc   |76            |>50K |1.0       |[0.3933948475870523,0.6066051524129477] |
|39 |Female|Private  |Assoc-voc   |49            |<=50K|1.0       |[0.47633354502406766,0.5236664549759323]|
|44 |Female|Private  |9th         |42            |>50K |0.0       |[0.5051383472308191,0.4948616527691809] |
|45 |Female|Private  |10th        |15            |>50K |0.0       |[0.5854002218696571,0.4145997781303429] |
|64 |Female|Gov      |Masters     |51            |<=50K|1.0       |[0.31418264295530757,0.6858173570446924]|
|61 |Male  |Gov      |Masters     |64            |<=50K|1.0       |[0.28659135610946374,0.7134086438905363]|
|74 |Male  |Private  |HS-grad     |51            |>50K |0.0       |[0.5250622155980815,0.4749377844019185] |
|24 |Female|Gov      |Assoc-voc   |73            |>50K |1.0       |[0.46405264687017184,0.5359473531298282]|
|56 |Female|Private  |Preschool   |4             |>50K |0.0       |[0.6425257251995086,0.35747427480049143]|
|42 |Male  |Gov      |Preschool   |14            |>50K |0.0       |[0.6415476426793341,0.35845235732066594]|
|67 |Male  |Private  |9th         |9             |<=50K|0.0       |[0.6229777715674455,0.3770222284325545] |
|43 |Male  |Gov      |Assoc-voc   |2             |<=50K|0.0       |[0.6899551442441456,0.3100448557558544] |
|37 |Male  |Private  |Bachelors   |60            |>50K |1.0       |[0.22031593281568476,0.7796840671843153]|
+---+------+---------+------------+--------------+-----+----------+----------------------------------------+
```

```
only showing top 20 rows
```

```
# Evaluar el modelo con métricas
evaluator = BinaryClassificationEvaluator(labelCol="label_index", rawPredictionCol="rawPrediction")
auc = evaluator.evaluate(predictions)

print(f" Métricas de evaluación:")
print(f"   Área bajo la curva ROC (AUC): {auc:.4f}")

# Calcular precisión (accuracy)
correct_predictions = predictions.filter(col("label_index") == col("prediction")).count()
total_predictions = predictions.count()
accuracy = correct_predictions / total_predictions

print(f"   Precisión (Accuracy): {accuracy:.4f}")
print(f"   Predicciones correctas: {correct_predictions}")
print(f"   Total de predicciones: {total_predictions}")
```

```
 Métricas de evaluación:
   Área bajo la curva ROC (AUC): 0.6714
   Precisión (Accuracy): 0.6215
   Predicciones correctas: 1243
   Total de predicciones: 2000
```

```
# Matriz de confusión
print(" Matriz de confusión:")
confusion_matrix = predictions.groupBy("label_index", "prediction").count().orderBy("label_index", "prediction
confusion_matrix.show()

# Distribución de predicciones
print("\n Distribución de predicciones:")
predictions.groupBy("prediction").count().orderBy("prediction").show()
```

```
 Matriz de confusión:
+-----------+----------+-----+
|label_index|prediction|count|
```

```
+-----------+----------+-----+
|        0.0|       0.0|  753|
|        0.0|       1.0|  273|
|        1.0|       0.0|  484|
|        1.0|       1.0|  490|
+-----------+----------+-----+


 Distribución de predicciones:
+----------+-----+
|prediction|count|
+----------+-----+
|       0.0| 1237|
|       1.0|  763|
+----------+-----+
```

Reflexión sobre los Resultados

¿Qué observamos sobre los resultados?

1. **Distribución de probabilidades**: El modelo genera probabilidades entre 0 y 1 para cada predicción, siendo más confiable cuando las probabilidades están más cerca de 0 o 1.
2. **Precisión del modelo**: La precisión (accuracy) nos indica qué porcentaje de predicciones fueron correctas.
3. **AUC**: El área bajo la curva ROC nos da una medida de qué tan bien el modelo puede distinguir entre las dos clases.
4. **Características importantes**: Variables como educación, edad y horas trabajadas probablemente tienen mayor peso en las predicciones.

🔮 7. Predicción con Nuevos Datos

Creamos nuevos registros y aplicamos el modelo entrenado para hacer predicciones.

```
# Crear nuevos datos de prueba (0 registros como mínimo pequeñito)
```

```
# Crear nuevos datos de prueba (9 registros como mínimo requerido)
new_data = [
    (35, "Male", "Private", 150000, "Bachelors", 45),      # Perfil promisorio: educación alta, horas normales
    (25, "Female", "Private", 120000, "HS-grad", 35),      # Joven con educación básica
    (45, "Male", "Gov", 200000, "Masters", 50),            # Gobierno, educación alta, experiencia
    (22, "Female", "Self-emp", 80000, "11th", 20),         # Joven, trabajo independiente, educación baja
    (55, "Male", "Private", 300000, "Doctorate", 60),      # Doctorado, muchas horas, experiencia
    (30, "Female", "Private", 100000, "Some-college", 40), # Educación universitaria parcial
    (28, "Male", "Self-emp", 90000, "Bachelors", 55),      # Independiente con título universitario
    (40, "Female", "Gov", 180000, "Masters", 42),          # Gobierno con maestría
    (33, "Male", "Private", 140000, "Assoc-voc", 38)       # Educación técnica/vocacional
]

columns = ["age", "sex", "workclass", "fnlwgt", "education", "hours_per_week"]
new_df = spark.createDataFrame(new_data, columns)

print(" Nuevos datos creados para predicción:")
new_df.show(truncate=False)
```

```
 Nuevos datos creados para predicción:
+---+------+---------+------+------------+--------------+
|age|sex   |workclass|fnlwgt|education   |hours_per_week|
+---+------+---------+------+------------+--------------+
|35 |Male  |Private  |150000|Bachelors   |45            |
|25 |Female|Private  |120000|HS-grad     |35            |
|45 |Male  |Gov      |200000|Masters     |50            |
|22 |Female|Self-emp |80000 |11th        |20            |
|55 |Male  |Private  |300000|Doctorate   |60            |
|30 |Female|Private  |100000|Some-college|40            |
|28 |Male  |Self-emp |90000 |Bachelors   |55            |
|40 |Female|Gov      |180000|Masters     |42            |
|33 |Male  |Private  |140000|Assoc-voc   |38            |
+---+------+---------+------+------------+--------------+
```

```
# Aplicar el modelo entrenado a los nuevos datos
print(" Aplicando modelo entrenado a nuevos datos...")
new_predictions = model.transform(new_df)
```

```
print("\n Resultados de predicciones para nuevos datos:")
new_predictions.select("age", "sex", "workclass", "education", "hours_per_week",
                       "prediction", "probability").show(truncate=False)
```

```
 Aplicando modelo entrenado a nuevos datos...

 Resultados de predicciones para nuevos datos:
+---+------+---------+------------+--------------+----------+----------------------------------------+
|age|sex   |workclass|education   |hours_per_week|prediction|probability                             |
+---+------+---------+------------+--------------+----------+----------------------------------------+
|35 |Male  |Private  |Bachelors   |45            |1.0       |[0.26364581478253624,0.7363541852174638]|
|25 |Female|Private  |HS-grad     |35            |1.0       |[0.48608688736130495,0.5139131126386951]|
|45 |Male  |Gov      |Masters     |50            |1.0       |[0.29650705107610337,0.7034929489238966]|
|22 |Female|Self-emp |11th        |20            |0.0       |[0.6300916264372447,0.3699083735627553] |
|55 |Male  |Private  |Doctorate   |60            |1.0       |[0.23783568418952328,0.7621643158104767]|
|30 |Female|Private  |Some-college|40            |0.0       |[0.58014197331719,0.41985802668281]     |
|28 |Male  |Self-emp |Bachelors   |55            |1.0       |[0.31019607742397,0.68980392257603]     |
|40 |Female|Gov      |Masters     |42            |1.0       |[0.3103617065984341,0.6896382934015659] |
|33 |Male  |Private  |Assoc-voc   |38            |0.0       |[0.5086291423398143,0.49137085766018573]|
+---+------+---------+------------+--------------+----------+----------------------------------------+
```

```
# Interpretar las predicciones de manera más legible
print(" Interpretación de las predicciones:")
print("=" * 80)

results = new_predictions.select("age", "sex", "workclass", "education", "hours_per_week", "prediction", "prob

for i, row in enumerate(results, 1):
    prediction_label = ">50K" if row.prediction == 1.0 else "<=50K"
    prob_high = row.probability[1]  # Probabilidad de clase >50K
    prob_low = row.probability[0]   # Probabilidad de clase <=50K

    print(f" Persona {i}:")
    print(f"    Perfil: {row.age} años, {row.sex}, {row.workclass}, {row.education}, {row.hours_per_week}h/sem
    print(f"    Predicción: {prediction_label}")
    print(f"    Probabilidad >50K: {prob_high:.3f} | Probabilidad <=50K: {prob_low:.3f}")
    print("-" * 80)
```

```
 Interpretación de las predicciones:
================================================================================
 Persona 1:
    Perfil: 35 años, Male, Private, Bachelors, 45h/semana
    Predicción: >50K
    Probabilidad >50K: 0.736 | Probabilidad <=50K: 0.264
--------------------------------------------------------------------------------
 Persona 2:
    Perfil: 25 años, Female, Private, HS-grad, 35h/semana
    Predicción: >50K
    Probabilidad >50K: 0.514 | Probabilidad <=50K: 0.486
--------------------------------------------------------------------------------
 Persona 3:
    Perfil: 45 años, Male, Gov, Masters, 50h/semana
    Predicción: >50K
    Probabilidad >50K: 0.703 | Probabilidad <=50K: 0.297
--------------------------------------------------------------------------------
 Persona 4:
    Perfil: 22 años, Female, Self-emp, 11th, 20h/semana
    Predicción: <=50K
    Probabilidad >50K: 0.370 | Probabilidad <=50K: 0.630
--------------------------------------------------------------------------------
 Persona 5:
    Perfil: 55 años, Male, Private, Doctorate, 60h/semana
    Predicción: >50K
    Probabilidad >50K: 0.762 | Probabilidad <=50K: 0.238
--------------------------------------------------------------------------------
 Persona 6:
    Perfil: 30 años, Female, Private, Some-college, 40h/semana
    Predicción: <=50K
    Probabilidad >50K: 0.420 | Probabilidad <=50K: 0.580
--------------------------------------------------------------------------------
 Persona 7:
    Perfil: 28 años, Male, Self-emp, Bachelors, 55h/semana
    Predicción: >50K
    Probabilidad >50K: 0.690 | Probabilidad <=50K: 0.310
--------------------------------------------------------------------------------
 Persona 8:
    Perfil: 40 años, Female, Gov, Masters, 42h/semana
    Predicción: >50K
    Probabilidad >50K: 0.690 | Probabilidad <=50K: 0.310
```

```
--------------------------------------------------------------------------------
 Persona 9:
    Perfil: 33 años, Male, Private, Assoc-voc, 38h/semana
    Predicción: <=50K
    Probabilidad >50K: 0.491 | Probabilidad <=50K: 0.509
--------------------------------------------------------------------------------
```

🎯 Resumen Final del Proyecto

¡Clasificación de Ingresos completada exitosamente para DataPros!

```
# Resumen final del proyecto
print("🎯 RESUMEN DEL PROYECTO - DATAPROS")
print("=" * 60)
print("✅ Datos cargados desde Google Drive: 2000 registros")
print("✅ Variables categóricas procesadas con StringIndexer y OneHotEncoder")
print("✅ Características ensambladas con VectorAssembler")
print("✅ Modelo de Logistic Regression entrenado con Pipeline")
print("✅ Modelo evaluado con métricas de rendimiento")
print("✅ Predicciones realizadas sobre 9 nuevos registros")
print(f"✅ Precisión del modelo: {accuracy:.1%}")
print(f"✅ AUC del modelo: {auc:.3f}")
print("\n🔬 DataPros ahora puede predecir ingresos basado en características demográficas y laborales!")
print("\n🌐 Ejecutado exitosamente en Google Colab con datos desde Google Drive")

# Cerrar sesión Spark
spark.stop()
print("\n🛑 Sesión Spark cerrada correctamente")
```

```
🎯 RESUMEN DEL PROYECTO - DATAPROS
============================================================
✅ Datos cargados desde Google Drive: 2000 registros
✅ Variables categóricas procesadas con StringIndexer y OneHotEncoder
✅ Características ensambladas con VectorAssembler
✅ Modelo de Logistic Regression entrenado con Pipeline
✅ Modelo evaluado con métricas de rendimiento
✅ Predicciones realizadas sobre 9 nuevos registros
```

```
✅ Precisión del modelo: 62.2%
✅ AUC del modelo: 0.671

🔬 DataPros ahora puede predecir ingresos basado en características demográficas y laborales!

🌐 Ejecutado exitosamente en Google Colab con datos desde Google Drive

🛑 Sesión Spark cerrada correctamente
```