

Actividad 7
Regresión Logística

16 de Octubre de 2023
ITESM Puebla

Equipo 1: Isreales

Ángel Rubén Vazquez Rivera	(A01735407)
José Israel Pérez Ontiveros	(A01423294)
Maximiliano Romero Budib	(A01732008)



Índice

1. Introducción	1
2. Desarrollo	1
2.1 Importar Librerías	1
2.2 Agregar datos	1
2.3 Preprocesamiento	2
2.3.1 Obtener información	2
2.3.2 Cuantitativas y cualitativas	2
2.3.3 Valores Nulos	3
2.4 Obtención de Características	6
2.5 Correlación por Regresión Logística	10
2.5.1 Función Variables Independientes	10
2.5.2 Función Ordenar Por métricas Recall	10
2.5.3 Función Modelos de regresión logística	11
2.5.4 Función de categórica a dicotómica	12
2.5.5 Resultados iniciales	12
2.5.6 Generación de modelos de regresión logística	13
2.5.7 Mejores modelos	14
3. Conclusión	14

1. Introducción

En el presente documento se describe de manera eficiente el procedimiento llevado a cabo para aplicar de manera exitosa múltiples modelos de regresión logística a un conjunto de datos específico, siendo así que sea posible definir con exactitud y de manera detallada cuales son los mejores modelos obtenidos al emplear diversas combinaciones.

Al desarrollar el código para dar solución a esta tarea, decidimos optimizar en la mayoría de la posible todo el proceso, siendo así que este cuenta con una estructura muy eficiente y comprensible reduciendo en gran medida el exceso de código, pues ahora el código está completamente modularizado, por lo que todo se basa y sustenta en funciones capaces de aplicar el modelo logístico sin importar los valores de entrada.


Durante el desarrollo se generaron cerca de 70 modelos de regresión logística diferentes, obteniendo cuando menos un 65% de esos modelos de manera efectiva y funcional, es decir, con matrices de confusión balanceadas y matrices lógicas, el resto de modelos simplemente no estaba balanceado o era capaz de predecir más allá de una sola clase de las dos existentes.

Optimizar este tipo de procesos y análisis es bastante conveniente, pues encontrar todas las posibles combinaciones y además todos los posibles modelos funcionales resulta de gran ayuda y además de manera rápida, evitando la repetición de código o procesos innecesarios.

2. Desarrollo

2.1 Importar Librerías

Para poder llevar a cabo el análisis se emplearon todas las siguientes librerías:



```
import pandas as pd
import sys
import warnings
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as special
from scipy.optimize import curve_fit
import seaborn as sns
from sklearn.metrics import r2_score
from funpy modeling.exploratory import freq_tbl
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, precision_recall_fscore_support
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, recall_score, f1_score
```

Imagen 1. Librerías

2.2 Agregar datos

En primera instancia fue necesario importar los datos o información a la que se le aplicarán los análisis pertinentes, para ello realizamos lo siguiente en el código:

```
data = pd.read_csv('./Recursos/TrainingDataComplete.csv', index_col=0)
display(data)
```

✓ 0.1s Python

Imágen 2. Lectura de archivo

2.3 Preprocesamiento

Antes de poder aplicar cualquier tipo de análisis, es necesario limpiar y organizar los datos de la manera más conveniente posible para que la ausencia de datos o el propio formato con el que cuente la información pueda llegar a interferir de algún modo, es por esto que se siguió el siguiente proceso para lograrlo de manera efectiva:

2.3.1 Obtener información

Primero analizamos la composición general de nuestros datos contenidos ya en un dataframe:

```
# Verificamos info del DF
data.info()
```

✓ 0.0s Python

```
<class 'pandas.core.frame.DataFrame'>
Index: 252000 entries, 1 to 252000
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Income                 252000 non-null int64
1   Age                   252000 non-null int64
2   Experience             252000 non-null int64
3   Married/Single         252000 non-null object
4   House_Ownership        252000 non-null object
5   Car_Ownership          252000 non-null object
6   Profession             252000 non-null object
7   CITY                   252000 non-null object
8   STATE                  252000 non-null object
9   CURRENT_JOB_YRS        252000 non-null int64
10  CURRENT_HOUSE_YRS      252000 non-null int64
11  Risk_Flag              252000 non-null int64
dtypes: int64(6), object(6)
memory usage: 25.0+ MB
```

Imágen 3. Info del DataFrame

Dicha información nos muestra un factor muy importante, la presencia de columnas de tipo numérico y de tipo object o string de manera general.

2.3.2 Cuantitativas y cualitativas

Ahora que ya sabemos sobre la presencia de información de tipo numérico (la que no interesa para los modelos de regresión) e información de tipo cualitativa, sin embargo, para poder aplicar un modelo matemático es esencial sólo considerar las columnas de tipo numérico, siendo así que separamos las columnas en esos dos tipos existentes:

```
# Separamos columnas cualitativas de cuantitativas
data['Risk_Flag'] = data['Risk_Flag'].replace({0: 'No', 1: 'Yes'})

dataCuantitativas = data.select_dtypes(include=['int64'])
dataCualitativas = data.select_dtypes(include=['object'])

print(f'Columnas cuantitativas:\n{dataCuantitativas.columns}')
print(f'\nColumnas cualitativas:\n{dataCualitativas.columns}')
```

✓ 0.0s Python

Columnas cuantitativas:
Index(['Income', 'Age', 'Experience', 'CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS'], dtype='object')

Columnas cualitativas:
Index(['Married/Single', 'House_Ownership', 'Car_Ownership', 'Profession',
 'CITY', 'STATE', 'Risk_Flag'],
 dtype='object')

Imagen 4. Separación de columnas

2.3.3 Valores Nulos

Una vez que ya tomamos sólo las columnas de tipo numérico es necesario verificar la composición y pureza de la información contenida en dichas columnas, pues de contar con valores nulos en los registros, los modelos matemáticos podrían ser mal aplicados o inclusive solo fallarían.

```
# Verificamos valores nulos
valores_nulos = data.isnull().sum()
print(valores_nulos)
```

✓ 0.0s Python

Income	0
Age	0
Experience	0
Married/Single	0
House_Ownership	0
Car_Ownership	0
Profession	0
CITY	0
STATE	0
CURRENT_JOB_YRS	0
CURRENT_HOUSE_YRS	0
Risk_Flag	0
dtype: int64	

Imagen 5. Nulos

En este caso en específico ya no contamos con valores nulos en ninguna de las columnas, siendo así que ya están limpios y casi listos para aplicarles un modelo matemático de regresión.

2.3.4 Valores Atípicos

Debemos verificar la existencia de valores atípicos que puedan afectar el rendimiento de nuestra información:



Imágen 6. Valores Atípicos

2.3.5 Detección de valores atípicos

Debemos encontrar estos valores atípicos para poder modificarlos antes de aplicar un modelo de regresión logística en este caso:

```
# Metodo cuantiles
# Encontramos los valores extremos
y = dataCuantitativas

percentile25 = y.quantile(0.25) #Q1
percentile75 = y.quantile(0.75) #Q3

iqr = percentile75 - percentile25

limiteSuperiorIQR = percentile75 + 1.5 * iqr
limiteInferiorIQR = percentile25 - 1.5 * iqr

print(f"Limite superior permitido :")
for col, val in limiteSuperiorIQR.items():
    print(f"{col}: {round(val, 2)}")

print(f"\n\nLimite inferior permitido :")
for col, val in limiteInferiorIQR.items():
    print(f"{col}: {round(val, 2)}")
```

✓ 0.0s Python

Limite superior permitido :
Income: 14939232.5
Age: 110.0
Experience: 30.0
CURRENT_JOB_YRS: 18.0
CURRENT_HOUSE_YRS: 16.0

Limite inferior permitido :
Income: -4958715.5
Age: -10.0
Experience: -10.0

Imagen 7. Valores Atípicos encontrados

Aquí aplicamos un método por cuantiles, llegando así a la obtención de valores atípicos existentes en nuestros datos.

2.3.6 Outliers

Con los valores atípicos encontrados definimos a los outliers:

```
# Encontramos Outliers
outliers = dataCuantitativas[(y>limiteSuperiorIQR) | (y<limiteInferiorIQR)]
print(f"\n\nOutliers:\n")
for col, val in outliers.items():
    print(f"{col}: {round(val, 2)}")
```

✓ 0.0s Python

Outliers:

Income:	Id
NaN	1
NaN	2
NaN	3
NaN	4
NaN	5

Imagen 8. Outliers

En este caso especial, los datos no cuentan con outliers que procesar o sustituir en su defecto por valores nulos, que posteriormente sean sustituidos por valores basados en medidas de tendencia central.

2.4 Obtención de Características

Para poder comprender de mejor manera la información con la que contamos, es necesario realizar representaciones gráficas y más visuales de lo que tiene por ofrecer cada una de nuestras variables o columnas en existencia.

Este proceso ya lo hemos realizado de manera regular en cada una de las actividades vistas en clase, sin embargo, al ser un proceso nefastamente repetitivo e innecesario, es posible optimizar el código de tal forma que en un solo bloque de código y un ciclo, se grafiquen todas las gráficas de todas las columnas:

```
#Arreglo de nombre de las columnas
columnNames = []
for name in data.columns:
    columnNames.append(name)

print(columnNames)

#Mostrar todas las graficas

for name in columnNames:

    if name == "Married/Single":

        table = freq_tbl(data['Married/Single'])
        Filtro = table[table['frequency'] > 1]
        Filtro_setter = Filtro.set_index("Married/Single")

        fig, axes = plt.subplots(1, 1, figsize=(10, 12))
        Filtro_setter["frequency"].plot(kind="pie", figsize=(10,5), shadow=True, autopct="%0.1f %")
        axes.set_title('Married/Single Distribution')

    if name == "House_Ownership":

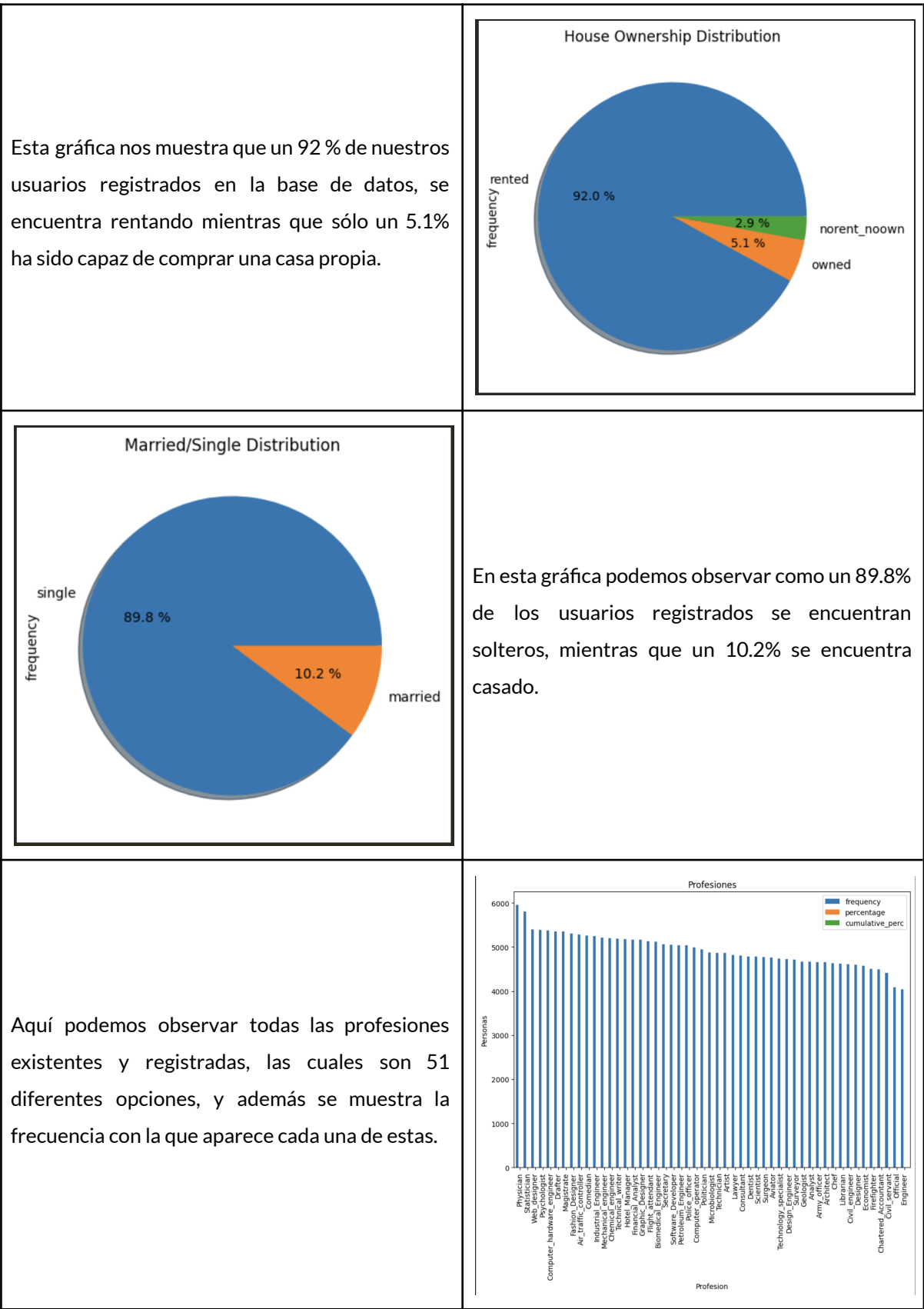
        table = freq_tbl(data['House_Ownership'])
        Filtro1 = table[table['frequency'] > 20]
        Filtro_setter = Filtro1.set_index("House_Ownership")

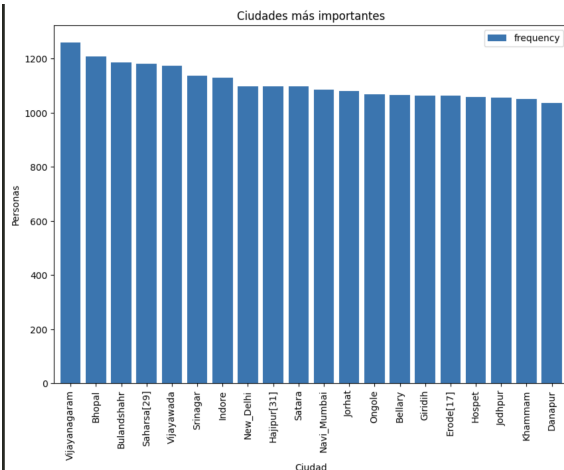
        fig2, axes = plt.subplots(1, 1, figsize=(10, 12))
        Filtro_setter["frequency"].plot(kind="pie", figsize=(10,5), shadow=True, autopct="%0.1f %")
        axes.set_title('House Ownership Distribution')
```

Imagen 9. Características

De este modo obtenemos todas las representaciones gráficas de nuestras columnas presentes en el DataFrame, y podemos observar información interesante que nos ayuda a entender mejor y de manera más real cada una de las características de cada variable.

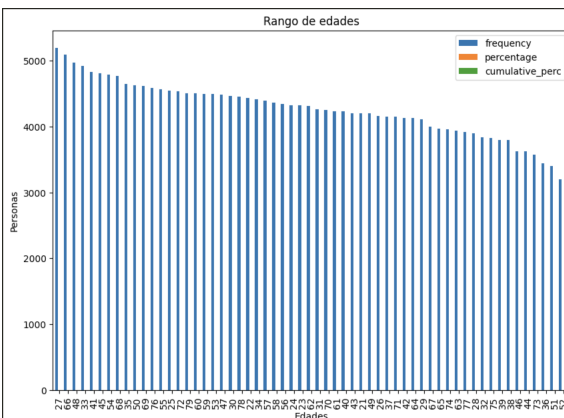
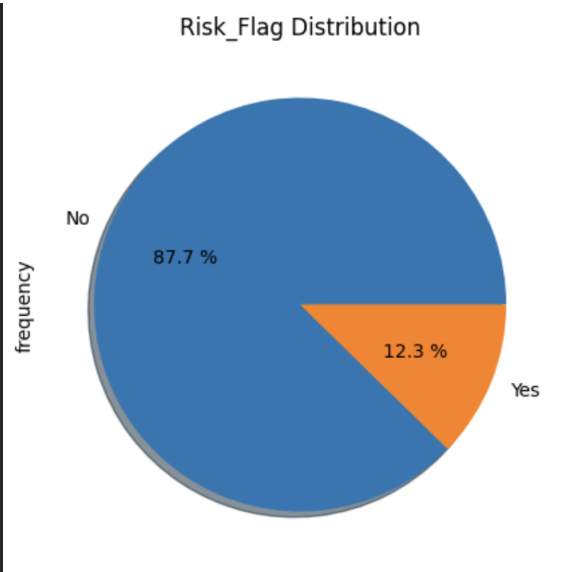
Llegando así a las siguientes gráficas:





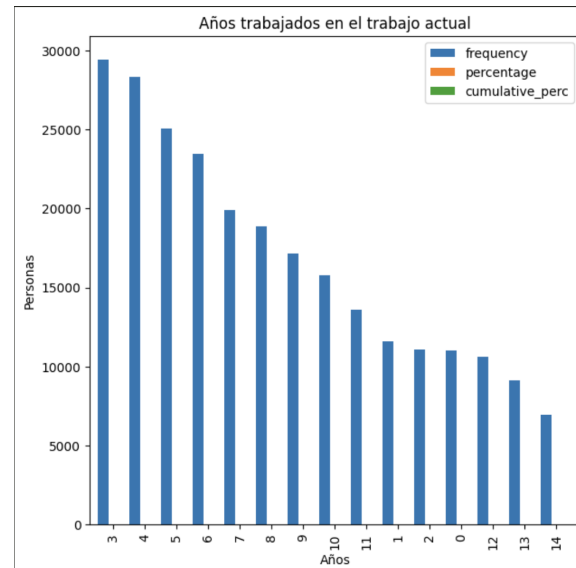
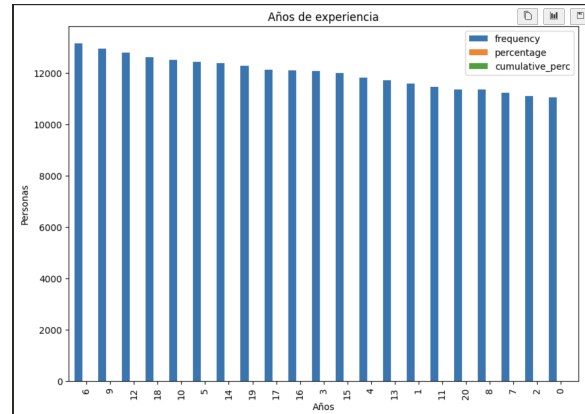
En este caso se muestran las 20 ciudades más frecuentadas en los registros, sin embargo existen más de 350 opciones para ciudad en todo el DataFrame.

En este caso podemos observar que solo un 12.3% de los usuarios registrados se encuentra en riesgo de quebrar financieramente hablando, mientras que un 87.7% de los usuarios se encuentran en buenas condiciones.



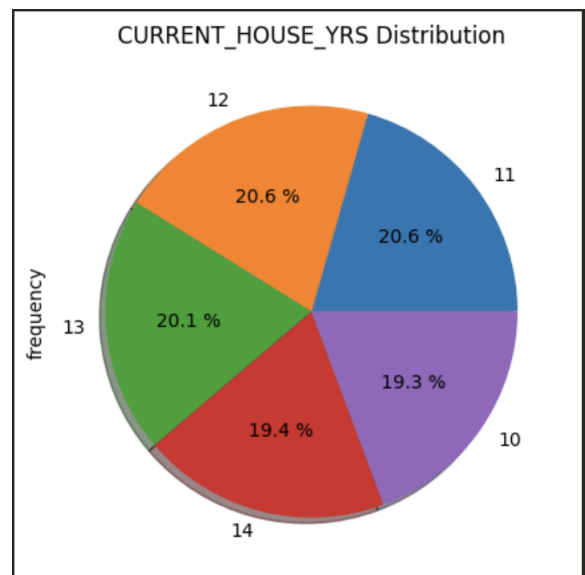
Aquí se muestran todas las edades registradas en el DataFrame, permitiéndonos definir que contamos con personas de entre 21 a 79 años y además se nos muestra la frecuencia de aparición para cada edad, percibiendo que existe una mayor cantidad de personas con 27 años.

En este caso se muestran los años de experiencia con más registros en todo el DataFrame, observando que la mayoría de usuarios cuentan con 6 años de experiencia.



Aquí se observa que la mayoría de los usuarios cuenta con 3 años de experiencia trabajando en general.

En este caso la distribución de los registros en la base de datos es bastante similar, siendo así que no existe algún grupo que destaque sobre el resto de manera notable.



2.5 Correlación por Regresión Logística

Para poder optimizar la aplicación de modelos de regresión logística a cada una de las variables dicotómicas existentes y por crear, es necesario contar con las siguientes funciones:

2.5.1 Función Variables Independientes

Considerando para aplicar el modelo de regresión logística es necesario emplear como variables independientes al conjunto de variables numéricas de diversas maneras o en diferentes combinaciones, se generó una función que recibe un arreglo que contenga los nombres de las columnas por utilizar para determinado modelo de regresión logística, y como valor de retorno, regresa un arreglo de arreglos, donde cada sub-arreglo contiene una posibilidad de combinación del arreglo inicial, es decir, si a la función “generate_combinations” le paso como entrada un arreglo de este tipo:

```
['Income', 'Age', 'Experience', 'CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS']
```

la función regresa un arreglo con 14 sub-arreglos, donde cada uno de estos últimos contienen una de las posibles combinaciones.

```
from itertools import combinations

def generate_combinations(input_array):
    result = []

    # Generate combinations of length 1 to 5
    for length in range(1, 6):
        for comb in combinations(input_array, length):
            result.append(list(comb))

    return result

#Generar el primer arreglo de combinaciones
input_array = ['Income', 'Age', 'Experience', 'CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS']
independientes = generate_combinations(input_array)
print(independientes)
```

✓ 0.0s Python

Imagen 10. Combinación de Variables Independientes

Esta función nos permite contar con todas las posibles combinaciones de las variables independientes a nuestra disposición, logrando así que al generar los diferentes modelos de regresión logística podamos probar la efectividad con cada posible combinación de variables independientes, cubriendo de manera absoluta todos los posibles escenarios.

2.5.2 Función Ordenar Por métricas Recall

Considerando que la clase de nuestro objetivo es la clase de positivo o 1 en este caso, queremos comprender cómo cada modelo representa la sensibilidad que tiene por detectar los escenarios o registros de este tipo.

Para este escenario contamos con el reporte de métricas importantes por cada modelo generado, siendo así que al almacenar esta información en un diccionario que contiene un diccionario por combinación, donde cada una de estas cuenta con dos llaves, una por cada clase de la variable dependiente, siendo así

que con una función es posible acceder a toda esta información almacenada y ordenar los diccionarios correspondientes a cada combinación en un orden descendente en base al valor de Recall de la clase 1 o la clase que estamos buscando en este caso particular, siendo así que recibe un diccionario con todas las combinaciones y métricas correspondientes, para retornar un diccionario ordenado a nuestra conveniencia, y de este modo al final de todo el cálculo, obtener los 3 mejores modelos de regresión logística para cada variable dependiente de tipo dicotómico, logrando así un filtrado increíblemente funcional y eficiente.

La función es la siguiente:

```
#Function to order the dictionaries
def ordenando(metricasFinal):
    metricasFinal = dict(sorted(metricasFinal.items(), key=lambda item: item[1].get(1, {}).get('Recall', 0), reverse=True))
    return metricasFinal
```

✓ 0.0s

Imágen 11. Ordenar por Recall

2.5.3 Función Modelos de regresión logística

Considerando que encontrar los mejores modelos de regresión en base a distintas variables dependientes dicotómicas, se genero una función con la capacidad de generar todos los modelos de regresión posibles por cada combinación existente de las posibles variables independientes, es decir, si contamos con 4 variables independientes, existen 14 maneras distintas de usarlas con la misma variable dependiente para ir verificando que modelo y por qué es mejor en base a la combinación en curso.

La función es la siguiente:

```
def regresionLogistica(data,independientes, dependiente, labels):
    #Metricas por dependiente
    metricasFinal = {}

    #Tomar las variables para nuestro modelo
    for i,combinacion in enumerate(independientes):
        X = data[combinacion]
        y = data[dependiente]

        #Dividir el conjunto de datos en la parte de entrenamiento y prueba
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = None)

        #Escalar los datos
        escalar = StandardScaler()

        #Realizar escalamiento de las variables "X" tanto de entrenamiento como de prueba
        X_train = escalar.fit_transform(X_train)
        X_test = escalar.transform(X_test)

        #Definimos el algoritmo a utilizar
        from sklearn.linear_model import LogisticRegression
        algoritmo = LogisticRegression()

        #Entrenamos el modelo
        algoritmo.fit(X_train, y_train)

        #Realizar prediccion
        y_pred = algoritmo.predict(X_test)
        y_pred

        #Generar mi matriz de confusion
        from sklearn.metrics import confusion_matrix
        matriz = confusion_matrix(y_test, y_pred)
        print(f"({cyan})Matriz de confusion y metricas de combinacion {i}:{reset}\n")
        print(f"({blue})Variables independientes: {reset}{combinacion}")
        print(f"({magenta})Variable dependiente: {reset}{dependiente}")
        display(matriz)

        #Calcular las metricas de cada modelo
        reporte=classification_report(y_test,y_pred,labels=labels, zero_division=1)
        print(reporte)

        precision, recall, f1_score, support = precision_recall_fscore_support(y_test, y_pred, labels=labels, zero_division = 1)

        #Dictionary with all the metric report information per every dependent variable
        metricsPerCombinacion = {}
```

Imágen 12. Función de regresiones logísticas

Esta función recibe el data frame por analizar, el arreglo de todas las posibles combinaciones de variables independientes, la variable dependiente y las clases a revisar, dentro de la función se calcular por cada combinación un modelo de regresión logística, y dentro de la misma función se almacenan los resultados de sus métricas en un diccionario con el fin de almacenar y más adelante ordenar los resultados de todas las combinaciones por cada variable dependiente y de este modo obtener las mejores 3 de cada caso en base al parámetro de recall.

2.5.4 Función de categórica a dicotómica

Para poder llevar a cabo el análisis completo de la situación fue necesario tomar algunas columnas de tipo categórico o numérico y convertirlas en una de tipo dicotómico, esto con el fin de proponerlas como variables dependientes en nuestros casos.

Algunas de ellas resultaron adecuadamente, mientras que otras simplemente no contaban con correlación alguna, es por esto mismo que se generó una función que recibe como entradas el nombre de la columna a convertir, un arreglo de las categorías por agrupar en una nueva y el nombre de esa nueva categoría, de este modo, se convirtieron las columnas para ir probando las regresiones.

Al final como equipo definimos conservar solo cuatro variables dicotómicas como dependientes, siendo así que si cada una cuenta con 14 posibilidades, se cuentan en total con 56 modelos de regresión de los cuales la mayoría sí representa un balanceo evidente en su matriz de confusión.

La función es la siguiente:

```
def normalToDicotomica(nameColumn, categoriesToAgrup, groupName):
    unico = np.unique(data[nameColumn])
    print(f"Las categorías actuales son:{reset} {unico}\n")

    print(f"En base a las categorías necesito que me des todas las categorías que serán agrupadas\n {reset}")

    porAgrupar = []

    for i in range(len(unico) - 1):
        porAgrupar.append(categoriesToAgrup[i])

    nuevaCategoria = groupName

    print(f"Estas son las categorías que vas a agrupar:{reset} {porAgrupar} {reset} en la nueva categoría:{reset} {nuevaCategoria}\n")

    data[nameColumn] = data[nameColumn].replace(porAgrupar, nuevaCategoria)
    resultado = np.unique(data[nameColumn])

    print(f"Ahora las nuevas dos categorías son:{reset} {resultado}")
```

Imagen 13. Función para generar dicotómicas

2.5.5 Resultados iniciales

Para comenzar con la obtención de los mejores modelos, primero se tomaron algunas variables categóricas y numéricas y se transformaron en dicotómicas para poder experimentar más escenarios, pues las variables que ya eran dicotómicas en un principio, no demostraron un buen comportamiento.

```

#House_Ownership to Dicotomic variable
normalToDicotomica("House_Ownership", ["rented", "norent_noown"], "no_owed")

#Income to Dicotomic variable
media = data['Income'].mean()
data['Income'] = np.where(data['Income'] > media, 1, 0)
data

#Experience to Dicotomic variable
data["Experience"] = np.where((data["Experience"]) > 10, 1, 0) #More than 10 years?

#Age to Dicotomic variable
media = data['Age'].mean()
data["Age"] = np.where((data["Age"]) < media, 1, 0) #How is the correlation for the youngs

#CURRENT_JOB_YRS to Dicotomic variable
data["CURRENT_JOB_YRS"] = np.where((data["CURRENT_JOB_YRS"]) >= 7, 1, 0) #How is the correlation for the youngs

data["CURRENT_HOUSE_YRS"] = np.where((data["CURRENT_HOUSE_YRS"]) <= 10, 1, 0)
✓ 0.1s

```

Imagen 14. Dicotomización

En este caso se toman 4 columnas y se dicotomiza, llegando así a 4 variables que son potencialmente útiles como variables dependientes en nuestros modelos, en este caso en base a características de las propias columnas se generaron sólo dos nuevas categorías generales para todos los registros.

2.5.6 Generación de modelos de regresión logística

Ahora que ya contamos con todos los preparativos es posible usar la función para generar regresiones y generar los resultados que sean necesarios:

```

#Call the function with Income as dependent variable
print(f"=====red")
| REGRESION LOGISTICA PARA INCOME |
|------(reset)\n=====

#Array for independent variables
input_array = ['Age', 'Experience', 'CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS']
independientes = generate_combinations(input_array)

metricas0 = regresionLogistica(data, independientes, "Income", [1, 0])

#Call the function with Experience as dependent variable
print(f"=====red")
| REGRESION LOGISTICA PARA EXPERIENCE |
|------(reset)\n=====

#Array for independent variables
input_array1 = ['Age', 'Income', 'CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS']
independientes1 = generate_combinations(input_array1)

metricas1 = regresionLogistica(data, independientes1, "Experience", [1, 0])

#Call the function with Age as dependent variable
print(f"=====red")
| REGRESION LOGISTICA PARA AGE |
|------(reset)\n=====

#Array for independent variables
input_array2 = ['Experience', 'Income', 'CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS']
independientes2 = generate_combinations(input_array2)

metricas2 = regresionLogistica(data, independientes2, "Age", [1, 0])

#Call the function with CURRENT_JOB_YRS as dependent variable
print(f"=====red")
| REGRESION LOGISTICA PARA CURRENT_JOB_YRS |
|------(reset)\n=====

#Array for independent variables
input_array3 = ['Experience', 'Income', 'Age', 'CURRENT_HOUSE_YRS']
independientes3 = generate_combinations(input_array3)

metricas3 = regresionLogistica(data, independientes3, "CURRENT_JOB_YRS", [1, 0])

```

Imagen 15. Regresiones

En este caso generamos 14 modelos de regresión logística por cada una de las variables dependientes o dicotómicas previamente creadas.

Se obtuvieron 56 resultados como el siguiente:

```
| REGRESION LOGISTICA PARA INCOME |
Matriz de confusion y metricas de combinacion 0:
Variables independientes: ['Age']
Variable dependiente: Income
array([[19262, 18475],
       [19123, 18740]])

      precision    recall  f1-score   support

     1       0.50      0.49      0.50      37863
     0       0.50      0.51      0.51      37737

 accuracy          0.50          0.50      75600
 macro avg       0.50      0.50      0.50      75600
 weighted avg    0.50      0.50      0.50      75600

Label: 1
Precision: 0.5035603923149268
Recall: 0.4949422919472836
F1 Score: 0.49921415061669194
Support: 37863

Label: 0
Precision: 0.5018106031001693
Recall: 0.5104274319633251
F1 Score: 0.5060823415044271
Support: 37737
```

Imágen 16. Resultados

De este modo se obtiene la matriz de confusión, y todas las métricas pertinentes para cada clase dentro del caso.

2.5.7 Mejores modelos

Ahora que ya contamos con todos los modelos posibles para nuestra propuesta, generamos una nueva función que toma el diccionario absoluto que contiene toda la información, y hacemos que por cada diccionario perteneciente a una variable dependiente, se ordenen todos los diccionarios contenidos en el anterior que corresponden a cada una de las combinaciones, esto en base al valor de recall para la clase 1.

Logrando de este modo mostrar en pantalla los 3 mejores modelos de regresión logística para cada una de nuestras variables dependientes.

2.6 Resultados finales

Estos son los mejores modelos de regresión encontrados en nuestro análisis:

<pre>Tabla de mejores modelos de regresion logistica para CURRENT_JOB_YRS: Variables independientes: ['Experience', 'CURRENT_HOUSE_YRS'] Variable dependiente: Income Precision: Recall: F1 Score: Support: 0.504 0.812 0.622 37864 Variables independientes: ['CURRENT_HOUSE_YRS'] Variable dependiente: Income Precision: Recall: F1 Score: Support: 0.501 0.809 0.619 37663 Variables independientes: ['CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS'] Variable dependiente: Income Precision: Recall: F1 Score: Support: 0.501 0.809 0.619 37694</pre>	<pre>Tabla de mejores modelos de regresion logistica para Income: Variables independientes: ['Age', 'Income', 'CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS'] Variable dependiente: Experience Precision: Recall: F1 Score: Support: 0.718 0.666 0.691 36378 Variables independientes: ['Income', 'CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS'] Variable dependiente: Experience Precision: Recall: F1 Score: Support: 0.72 0.663 0.69 36338 Variables independientes: ['CURRENT_JOB_YRS'] Variable dependiente: Experience Precision: Recall: F1 Score: Support: 0.714 0.663 0.688 36252</pre>
<pre>Tabla de mejores modelos de regresion logistica para Experience: Variables independientes: ['Experience', 'Income'] Variable dependiente: Age Precision: Recall: F1 Score: Support: 0.493 0.264 0.344 36917 Variables independientes: ['Income', 'CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS'] Variable dependiente: Age Precision: Recall: F1 Score: Support: 0.502 0.186 0.271 37137 Variables independientes: ['Experience', 'Income', 'CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS'] Variable dependiente: Age Precision: Recall: F1 Score: Support: 0.498 0.181 0.265 37257</pre>	<pre>Tabla de mejores modelos de regresion logistica para Age: Variables independientes: ['Experience', 'Age', 'CURRENT_HOUSE_YRS'] Variable dependiente: CURRENT_JOB_YRS Precision: Recall: F1 Score: Support: 0.667 0.718 0.691 33761 Variables independientes: ['Experience', 'Income', 'CURRENT_HOUSE_YRS'] Variable dependiente: CURRENT_JOB_YRS Precision: Recall: F1 Score: Support: 0.664 0.717 0.689 33541 Variables independientes: ['Experience', 'Age'] Variable dependiente: CURRENT_JOB_YRS Precision: Recall: F1 Score: Support: 0.664 0.717 0.689 33565</pre>

De este modo se cuentan con los mejores 12 modelos posibles encontrados en nuestro análisis, los cuales son bastante reales y adecuados en su mayoría.

3. Conclusión

De manera general una vez concluida la actividad, obtuvimos un entendimiento bastante respecto a la aplicación de modelos matemáticos basados en regresión logística, llegando así a un desenlace bastante óptimo.

De igual manera el código desarrollado durante la actividad para generar todos los modelos de manera rápida y eficiente, es una herramienta completamente en otro plano que permite optimizar todo el proceso y a la vez generar resultados fiables con pocas líneas de código y una simplicidad evidente.

Income fue la variable dependiente con más correlación frente a sus posibles combinaciones llegando a rondar el 80% de recall para su clase 1, siendo así un resultado muy bueno.