

# CAR INSURANCE CLAIMS PREDICTION

Comisión: 42390

Alumna: Valdivia Micaela Jorgelina

Profesor: Ruiz Jorge

Tutor: Ocaña Anderson

---



---

## ÍNDICE

- **Introducción : Contexto comercial, empresarial y Análítico**
- **Data Wrangling: - Comprensión básica de nuestra data, limpiar, transformar y reestructurar para un mejor análisis**
- **Exploratory Data Analysis: “Buscamos identificar tendencias, patrones y relaciones, mediante las variables y gráficos”**
- **Machine Learning: “Construcción de Modelos de predicción, utilizando varios algoritmos”**
- **Machine Learning: “Random Forest, profundizando y desafiando nuestros modelos y hiperparámetros”**
- **Conclusiones Finales**

---

# Introducción

Somos una compañía de seguros patrimoniales con el objetivo de dar un óptimo servicio, y podamos proteger el futuro de nuestros clientes con precios que se ajusten a cada perfil del asegurado.

## Contexto Empresarial:

En nuestro sector de modelling decidimos challengear el modelo de predicción de siniestralidad utilizado para el pricing de seguros de autos.

Actualmente utilizamos un modelo de regresión logística, y consideramos ciertas variables para la predicción de que ocurra o no un siniestro.

Pero nuestra hipótesis es comparar con nuevos modelos, con el fin de poder otorgar al asegurado un precio adecuado según las distintas variables y evitar la selección adversa de riesgos con un buen modelo de scoring.

De este modo, posicionar a la empresa como gran competidor en el mercado asegurador.

## Problema comercial:

En simples palabras, el scoring permite que el asegurado pague una prima de acuerdo con su perfil de riesgo, es decir, características del auto, el uso (en términos de kilometraje), la experiencia del conductor, etc.

Entonces la tarea no es más que crear dicho modelo para predecir la siniestralidad, considerándola como un egreso de la compañía, ya que si el siniestro ocurre la compañía debe resarcir la suma asegurada pactada en la póliza.

A raíz de esto, surgen preguntas a responder por parte de nuestro sector:

¿Qué hace que el conductor tenga más siniestralidad y genere más egreso a la compañía?  
¿Los factores de scoring son los necesarios para determinar un buen precio al asegurado y lo suficiente en caso de que haya usuarios con riesgo alto?

---

## **Contexto analítico:**

Los datos ya se han recopilado y están en uso para nuestro equipo:

La fuente de la información es "Car\_Insurance\_Claim.csv", el cual contiene el historial de los asegurados en la compañía desde el año 2012 - 2020, donde en la columna "outcome" se podrá observar de estos usuarios quienes tuvieron siniestros y quienes no.

---

# Data Wrangling: "Conozcamos más de nuestro Dataset y empecemos a limpiar, transformar y organizar nuestros datos"

## CONOCIENDO Y EXPLORANDO NUESTRO DATASET

Evaluaremos los datos actuales para comprender qué piezas de información podrían faltar y analizar cómo son los campos / variables de nuestra base.

Debemos analizar cada una de ellas individualmente y todas como un todo, para averiguar cómo exactamente debemos complementarlas. En cada etapa, debemos tener en cuenta nuestro objetivo: predecir que un siniestro ocurra o no.

Eso significa que deberíamos pensar en la siguiente pregunta: "¿Qué información sería útil para predecir la siniestralidad de un seguro?".

### Cuadro resumen campos del dataset:

CAMPOS	DESCRIPCIÓN	TYPE
ID	POLICY ID	INT64
AGE	RANGO EDAD DE ASEGURADO	OBJECT
GENDER	GÉNERO DE ASEGURADO	OBJECT
DRIVING_EXPERIENCE	RANGO DE AÑOS DE EXPERIENCIA DE ASEGURADO	OBJECT
EDUCATION	NIVEL DE EDUCACIÓN DE ASEGURADO	OBJECT
INCOME	NIVEL SOCIOECONÓMICO DE ASEGURADO	OBJECT
CREDIT_SCORE	SCORE DE CREDIT DE CADA ASEGURADO	FLOAT64
VEHICLE_OWNERSHIP	SI ES DUEÑO O NO	INT64
VEHICLE_YEAR_RANGE	RANGO DE AÑO DE CREACIÓN DE AUTO (after or before 2015)	OBJECT
VEHICLE_MONTH	MES DEL AUTO CREACIÓN	INT64
VEHICLE_YEAR	AÑO DEL AUTO CREACIÓN	INT64
VEHICLE_YEAR_MONTH	AÑO MES AUTO CREACIÓN	INT64
MARRIED	FLAG CASADO/A O NO	INT64
CHILDREN	FLAG CON O SIN HIJOS	INT64
POSTAL_CODE	CÓDIGO POSTAL	INT64
ANNUAL_MILEAGE	KILOMETRAJE USADO POR EL AUTO EN EL AÑO	FLOAT64
VEHICLE_TYPE	MODELO DEL AUTO	OBJECT
SPEEDING_VIOLATIONS	CANTIDAD DE INFRACCIONES DE VELOCIDAD	INT64
DUIS	CANTIDAD DE INFRACCIONES POR ESTUPEFACIENTES	INT64
PAST_ACCIDENTS	ACCIDENTES PASADOS	INT64
TARGET	TUVO UN SINIESTRO O NO	INT64
OUTCOME		

---

## LIMPIANDO/TRANSFORMANDO NUESTRO DATASET

### LIMPIANDO:

#### Primero: Distinguimos nuestros valores nulos

Data columns (total 22 columns):			
#	Column	Non-Null Count	Dtype
0	ID	10000 non-null	int64
1	AGE	10000 non-null	object
2	GENDER	10000 non-null	object
3	RACE	10000 non-null	object
4	DRIVING_EXPERIENCE	10000 non-null	object
5	EDUCATION	10000 non-null	object
6	INCOME	10000 non-null	object
7	CREDIT_SCORE	9018 non-null	float64
8	VEHICLE OWNERSHIP	10000 non-null	int64
9	VEHICLE_YEAR_RANGE	10000 non-null	object
10	VEHICLE_YEAR	10000 non-null	int64
11	VEHICLE_YEAR_MONTH	10000 non-null	int64
12	VEHICLE_MONTH	10000 non-null	int64
13	MARRIED	10000 non-null	int64
14	CHILDREN	10000 non-null	int64
15	POSTAL_CODE	10000 non-null	int64
16	ANNUAL_MILEAGE	9043 non-null	float64
17	VEHICLE_TYPE	10000 non-null	object
18	SPEEDING_VIOLATIONS	10000 non-null	int64
19	DUIS	10000 non-null	int64
20	PAST_ACCIDENTS	10000 non-null	int64
21	OUTCOME	10000 non-null	int64

---

## **Segundo, ¿Cómo lidiar con la info faltante?**

### **Opción a: Eliminar datos**

- a. eliminar toda la fila
- b. eliminar toda columna

### **Opción b: Reemplazar data**

- a. reemplazar data con la media
- b. reemplazarla con la frecuencia
- c. reemplazarla en base a otras funciones

Dado que ambas columnas (Credit\_score y Annual\_mileage) no tienen nulos en todas las filas, no tiene sentido eliminar la columna (opción b), además de que son pocos datos nulos y las columnas son importantes para nuestro análisis. Entonces tenemos la opción de eliminar filas o reemplazar data.

Considerando que:

El promedio de credit\_score: 0.52 y el promedio de annual\_mileage es 11.697 Km

Vemos que son 982 casos para credit score, lo cuales dentro de los datos totales que tenemos representa un 10% de nuestra base.

Si optamos por ponerle la media diríamos que esos asegurados tienen un buen score crediticio y los daríamos por usuarios tal vez buenos y podríamos estar justamente describiéndolos mal si bien no afectaría a nuestra media total en el ponderado, optamos por eliminarlos y no generar esa falla de información, además los consideramos pocos casos.

---

En el caso de annual\_mileage al eliminar credit score, nos quedarán menos usuarios de esos 957 (cuando vimos los true y false más arriba se ven coinciden muchos) en este caso optamos por colocar la media, ya que es distinto que en este caso el atributo del cuanto maneja es importante pero no sabemos aún si esos usuarios son "buenos" o no, y al ser aún menos casos, optamos por la media

### Tercero: decisión final:

Se observa que hay nulls en ANNUAL MILEAGE y CREDIT SCORE: ambos los consideramos de importancia en primera instancia de nuestro proyecto

Credit score: Aunque sean 1.000 casos de asegurados sin esa data, ponerles 0 en score indicaría que es muy riesgoso y capaz esos asegurados no los son y al momento de ofrecerles analizarlos no sería buena esa decisión.

ANNUAL MILEAGE: Lo mismo con el kilometraje anual, si es cero es que no uso el auto y no sería un dato real.

Final decisión: Para los kilómetros si vamos a considerar la media (con lo cual prevalecerán 9.043 usuarios) pero eliminaremos los faltantes del score porque ahí si consideramos que no es un valor que podamos reemplazar con la media.

### Resultados finales:

	ID	CREDIT_SCORE	VEHICLE_OWNERSHIP	VEHICLE_YEAR	VEHICLE_YEAR_MONTH	VEHICLE_MONTH	MARRIED	CHILDREN	POSTAL_CODE	ANNUAL_MILEAGE	SPEEDING_VIOLATIONS	DUIS	PAST_ACCIDENTS	OUTCOME
count	9018.0	9018.0	9018.0	9018.0	9018.0	9018.0	9018.0	9018.0	9018.0	9018.0	9018.0	9018.0	9018.0	9018.0
mean	501452.4	0.5	0.7	2014.9	201494.2	6.5	0.5	0.7	19796.9	11693.5	1.5	0.2	1.1	0.3
std	290500.8	0.1	0.5	2.4	242.9	3.5	0.5	0.5	18803.1	2682.9	2.2	0.6	1.7	0.5
min	101.0	0.1	0.0	2012.0	201201.0	1.0	0.0	0.0	10238.0	2000.0	0.0	0.0	0.0	0.0
25%	249058.2	0.4	0.0	2013.0	201305.0	3.0	0.0	0.0	10238.0	10000.0	0.0	0.0	0.0	0.0
50%	503338.0	0.5	1.0	2014.0	201410.0	7.0	1.0	1.0	10238.0	11693.5	0.0	0.0	0.0	0.0
75%	756898.5	0.6	1.0	2016.0	201612.0	10.0	1.0	1.0	32765.0	13000.0	2.0	0.0	2.0	1.0
max	999976.0	1.0	1.0	2020.0	202012.0	12.0	1.0	1.0	92101.0	22000.0	22.0	6.0	15.0	1.0

---

## TRANSFORMANDO

Generamos campos descriptivos para un mejor entendimiento de la base en caso de necesitar referencia de qué significa el 0 y 1 de los siguientes campos:

**VEHICLE\_OWNERSHIP\_desc:** Non\_Owner (si el campo VEHICLE\_OWNERSHIP es 0) -Owner (si el campo VEHICLE\_OWNERSHIP es 1)

**MARRIED\_desc:** Non\_Married (si el campo MARRIED es 0) -Married (si el campo MARRIED es 1)

**CHILDREN\_desc:** WO\_CHILDREN (si el campo CHILDREN es 0) -With\_CHILDREN (si el campo CHILDREN es 1)

**OUTCOME\_desc:** WO\_Claim (si el campo OUTCOME es 0) -Claim (si el campo OUTCOME es 1)

**CREDIT\_SCORE\_range:** si el CREDIT\_SCORE < 0.4, entonces "RIESGO\_ALTO" , si el CREDIT\_SCORE >= 0.4 y <0.7, entonces "RIESGO\_MEDIO" , si el CREDIT\_SCORE >= 0.7, entonces "RIESGO\_BAJO" .

---

## **Exploratory Data Analysis:** “Buscamos identificar tendencias, patrones y relaciones, mediante las variables y gráficos”

Siendo el objetivo de poder predecir que ocurra o no un siniestro, y de esta manera ofrecerle un óptimo precio a los asegurados a la hora de ofrecerles una póliza, nos vamos a encontrar con un problema de clasificación. Pero al pensar en nuestro modelo, ¿podemos considerar que entre más factores agreguemos mejor predictor será ?

Entonces, considerando lo mencionado, nos preguntamos:

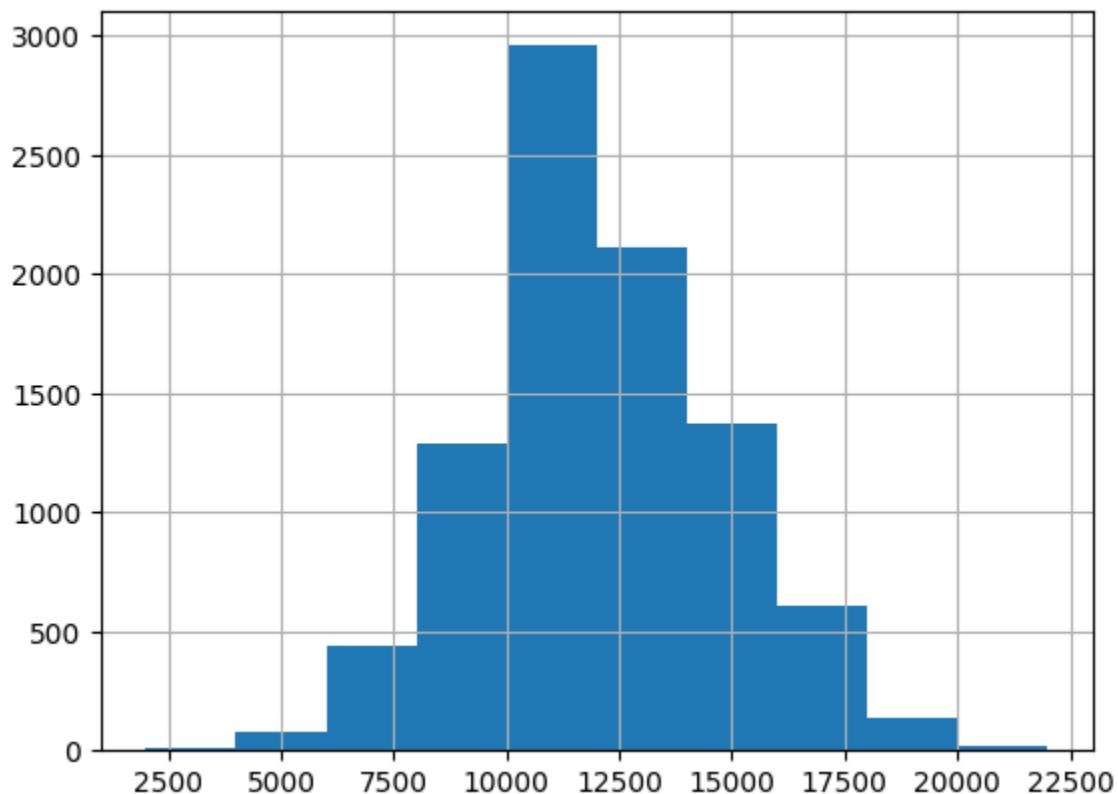
- 1) ¿A mayor cantidad de años de experiencia implica mayor exposición y mayor cantidad de siniestros o menor cantidad de accidentes por mejor desempeño del conductor? Y, que tenga mayor kilometraje de uso ¿implica también mayor cantidad de accidentes? De este modo la compañía podrá alcanzar una selección de los mejores factores de scoreo en su pricing, y no haya selección adversa de riesgos.
- 2) Siendo una de nuestras variables de interés, el kilometraje, ¿cómo será dicha variable, si analizamos considerando sub segmentos / agrupaciones, según sexo y según ingreso? Por ejemplo, un hombre de nivel socioeconómico pobre ¿maneja mucho menos en promedio que la mediana, por su bajo nivel de ingreso ?
- 3) Siendo otra de nuestras variables de interés, el credit score ¿cómo será dicha variable, si analizamos considerando sub segmentos / agrupaciones, según ingreso?

- 
- 4) ¿Cuánto más multas tiene un asegurado, mayor es la probabilidad que cometa un accidente? O, ¿ Cuánto menos educación o más antiguo sea el auto mayor cantidad de accidentes de tráfico?

### ¿Que un conductor tenga más experiencia implica menos siniestros?

Que un usuario tenga más kilometraje de uso en su automóvil, puede significar dos situaciones para una compañía de seguros, que su asegurado es experimentado y tiene menos propensión a sufrir un siniestro, o tiene mayor exposición constante, por ejemplo salir a la calle diariamente para ir a su trabajo.

#### Primero analizaremos la variable kilometraje

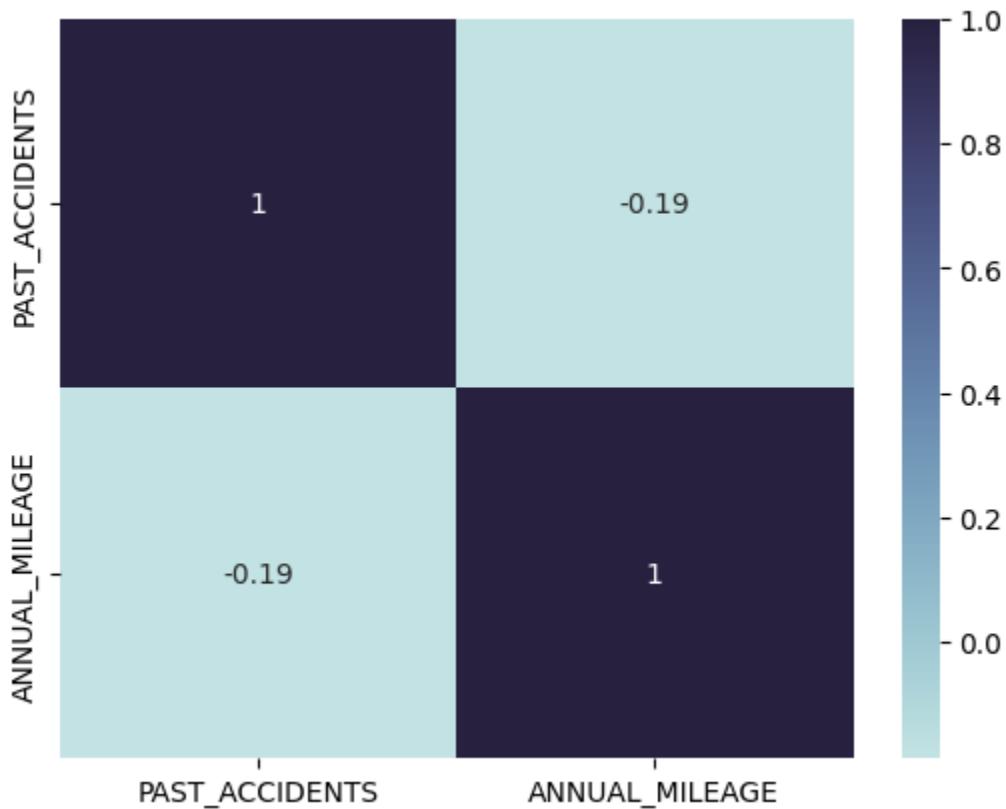


El histograma muestra que el centro de los datos se encuentra cerca de 11 mil km (mediana & media ), y que la extensión de los mismos va de unos 2 mil km a 22.5 mil km. Vemos entonces una simetría de los datos. Probablemente estos tengan una distribución

---

normal. No vemos valores inusuales (alguna barra fuera de average con una frecuencia alta).

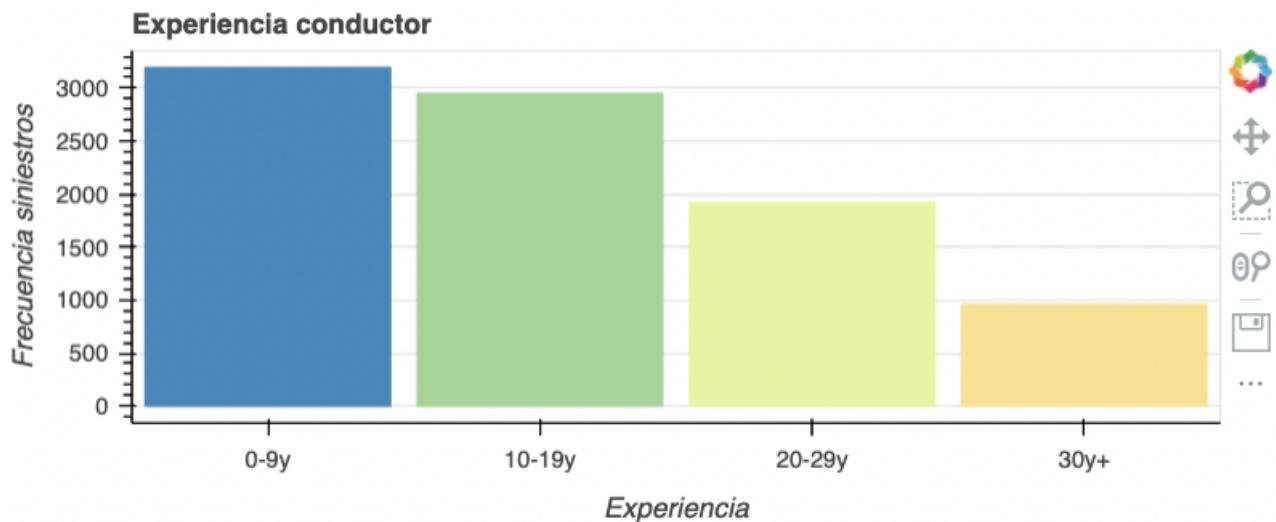
**Segundo veremos como es la correlación entre el kilometraje y los accidentes de tráfico:**



**INSIGHT:** Podríamos considerar que el kilometraje podría tener una correlación positiva, cercana al 1, es decir, una correlación fuerte, con la cantidad de accidentes. Pero vemos todo lo contrario, una correlación débil negativa, quiere decir que, a mayor cantidad de uso que los asegurados le den a su auto, definiendo uso como el kilometraje, no vemos una mayor cantidad de accidentes. Aunque esta correlación es débil (cercana al cero), va de la mano con lo visto anteriormente, mayor kilometraje no implica mayor exposición sino mayor experiencia del conductor, y menor cantidad de accidentes. Por lo cual, en este estudio, el scoring de PAYD (Pay as you drive) no será tan representativo para esta muestra de asegurados analizados.

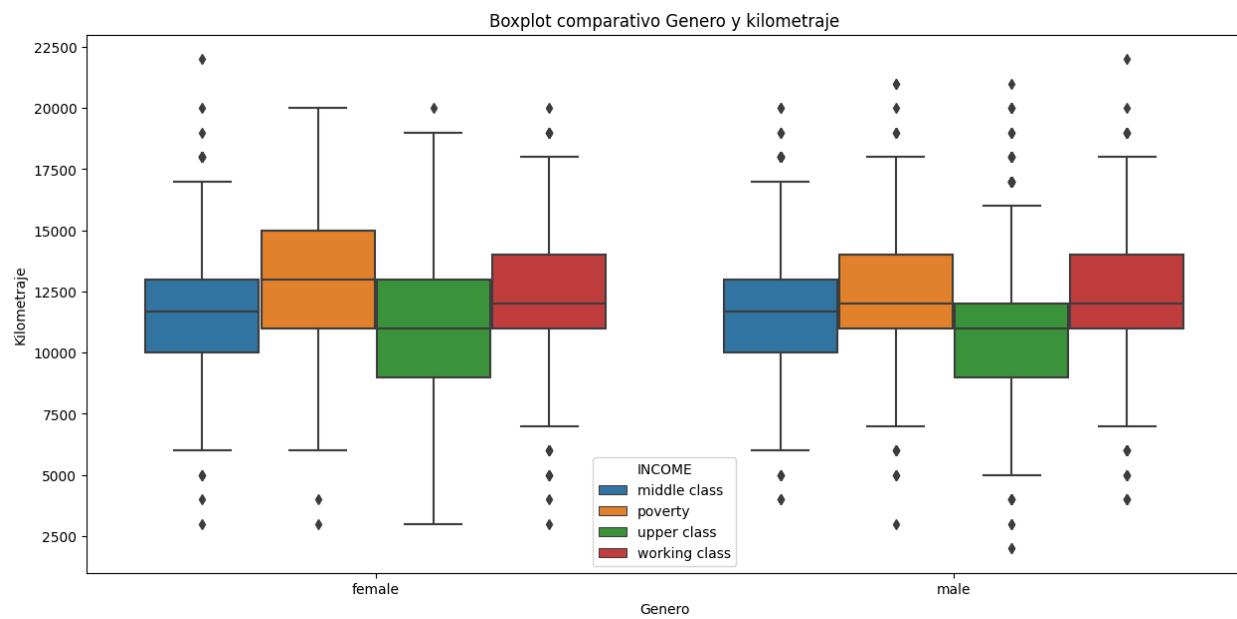
---

Tercero analizamos la experiencia del conductor (medidas en años):



**INSIGHT:** El gráfico Barchart refleja un dato de gran importancia, que consideramos en nuestra hipótesis, la experiencia que tiene un conductor a la hora de manejar su vehículo muestra menor siniestralidad, es decir, vemos como aquellos asegurados que tienen una experiencia “+30 years” tiene menor cantidad de siniestros que aquellos usuarios con experiencia “20-29 years”, éste menor que “10-19 years”, y este último menor que “0-9 years”. De este modo, estas barras escalonadas, nos indica que a mayor experiencia menor siniestros.

**¿Según el ingreso y/o sexo un asegurado usa más o menos su automóvil?**



1. Outliers: Lo primero que nos indica el Boxplot, es los valores atípicos del kilometraje, tanto por encima y por debajo de las líneas del box. En el ingreso “middle class”, tanto del género femenino y masculino, vemos varios de estos valores, y “upper class” del sexo masculino. En el nivel “poverty” de las mujeres, vemos muy pocos valores atípicos por debajo del box nomás.

2. Simetría: Habíamos visto, según el histograma que el kilometraje, era simétrico, pero cuando consideramos otras variables como el ingreso y sexo, los datos están sesgados, la mediana está más cerca de la parte superior o inferior de la caja, para el caso de “working class” de las mujeres, y para el caso de los hombres todos menos “middle class”.

---

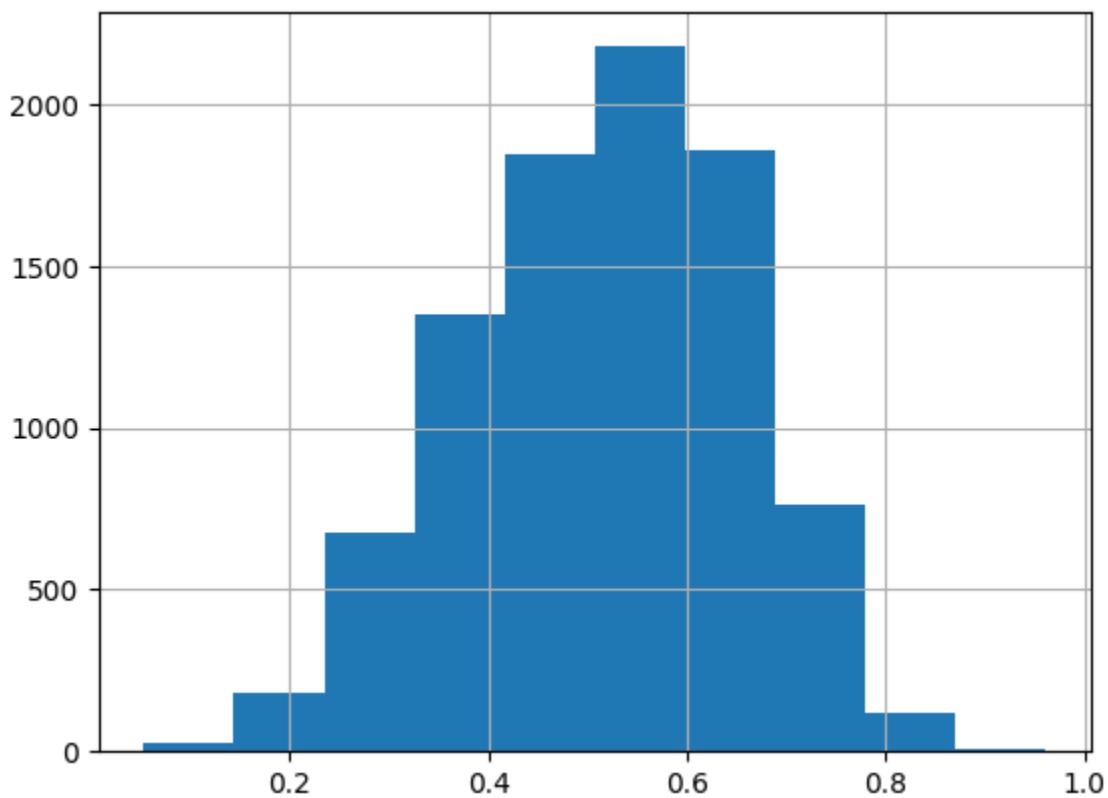
3. Bigotes: el bigote inferior representa el mínimo valor posible que no puede ser, ejemplo 6 K para el ingreso “poverty” de mujeres, entonces será sustituido por el valor mínimo inferior incluido en nuestra muestra. El bigote superior es el máximo de los datos, es decir 20 K para el segmento mencionado.

**INSIGHT FINAL** vemos que casi todos los grupos tienen un kilometraje muy cerca del promedio pero algunos grupos como, hombres de upper class y poverty, y ambos sexos en working class, tienen una asimetría, es decir, su media está alejada de su mediana. Entonces, en base a nuestra pregunta de interés, un hombre de ingreso bajo (poverty) en promedio usa menos que la mediana.

Y por último, hay una cantidad considerable de outliers en segmentos de upper class hombres y middle class mujeres, que vamos a considerar analizar en mayor profundidad.

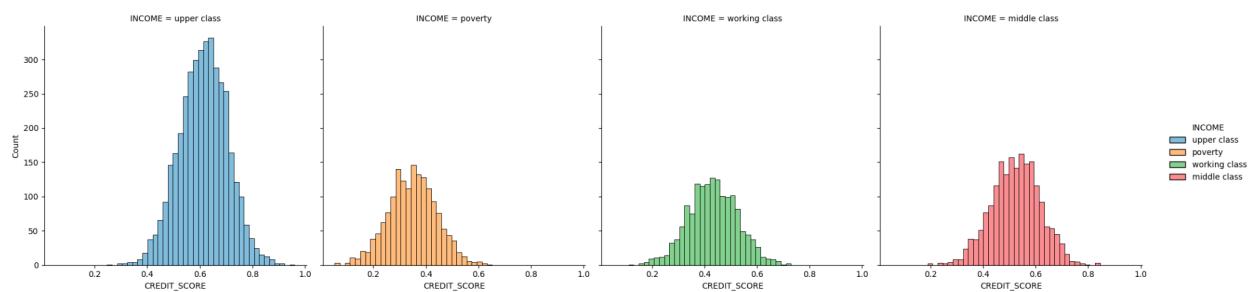
**Que un usuario tenga un mayor ingreso ¿implica que tenga un mejor score crediticio?**

**Primero analizaremos la variable Credit Score:**



**INSIGHT:** El histograma muestra que el centro de los datos se encuentra cerca del 50% (mediana & media), y que la extensión de los mismos va de unos 0 a 1 (al ser un score). Vemos entonces una simetría de los datos. Como ya mencionamos en kilometraje, probablemente estos tengan una distribución normal. Tampoco vemos valores inusuales, pero según el boxplot ya analizado si existen outliers.

### Segundo, variable Credit Score respecto nivel socioeconómico:

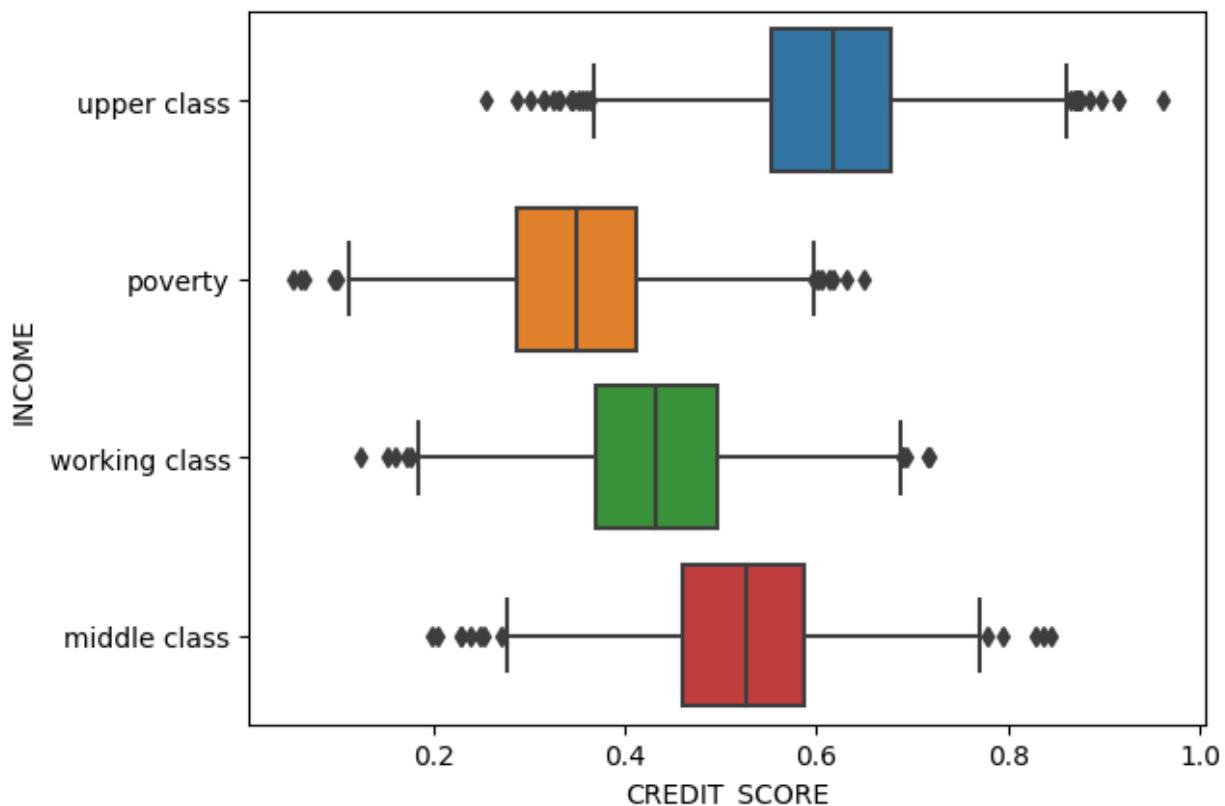


---

**INSIGHT:** Todos parecen tener una distribución Normal con media en 0.5, además observamos que la clase “poverty” se aleja un poco de esta media teniendo un menor score y como la clase “upper class” tiende una tendencia a un score mayor.

En cuanto a cantidades de asegurados, primero está la clase “upper class”, segundo, “poverty”, tercero “working class” y último “middle class”.

**Tercero trabajaremos con el gráfico Box Plot para obtener conclusiones de la variable Credit Score según nivel socioeconómico:**



**INSIGHT:** Podemos ver de manera más clara como la clase “upper” tiene un mayor score.

Vemos para todos los casos que la distribución es simétrica (la línea de división está en el medio)

---

Respecto de los valores atípicos, notamos que hay varios outliers, sobre todo en el nivel "upper".

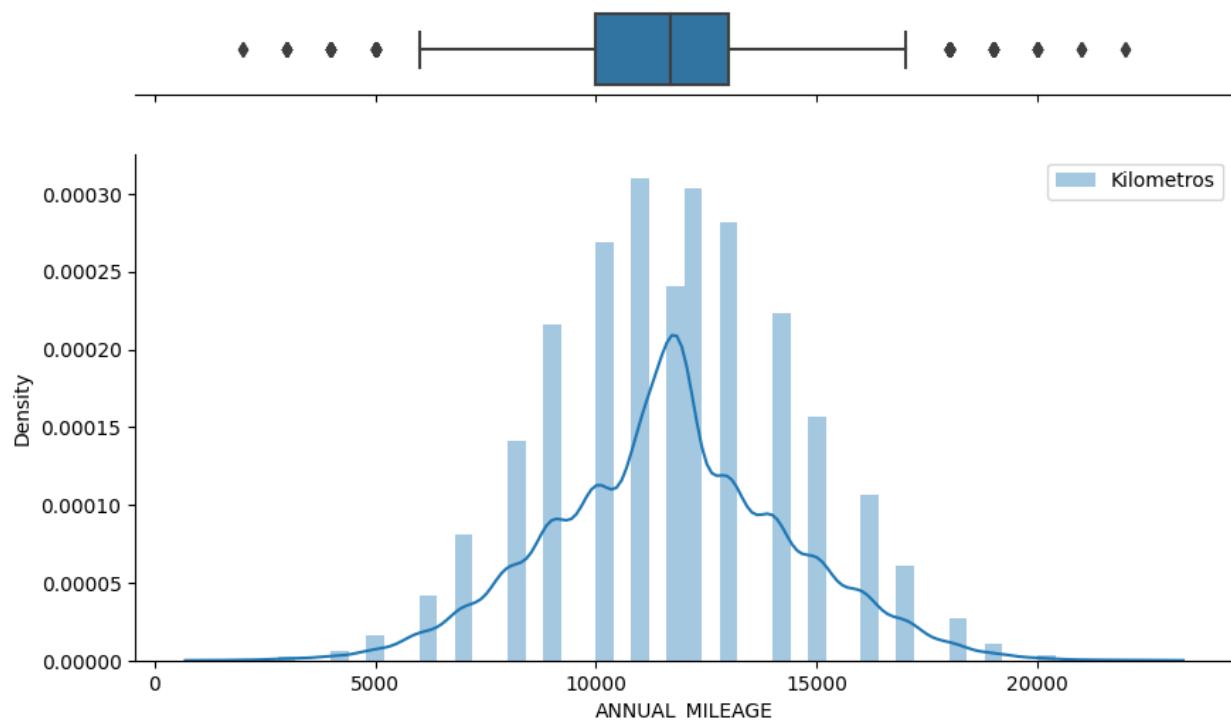
Por lo cual, el desafío es identificar esos outliers y ver cómo tratarlos, y por otro lado, considerar que la distribución de credit score es simétrica.

## Outliers

A raíz de la anterior conclusión vamos a adentrarnos en el análisis de Outliers, si bien es parte de Data Wrangling, también es gracias nuestros análisis de EDA que vemos necesario darle un análisis más exhaustivo:

Mediante el método rango intercuartílico veremos los outliers de las variables kilometraje y credit score:

### Kilometraje:



---

Rangos de cuartiles:

El valor máximo de kilómetros anuales es 17.500 y la cantidad outliers superiores de kilómetros anuales son 154. El valor mínimo de kilómetros anuales es 5.500 y la cantidad outliers inferiores de kilómetros anuales: 91

Summary:

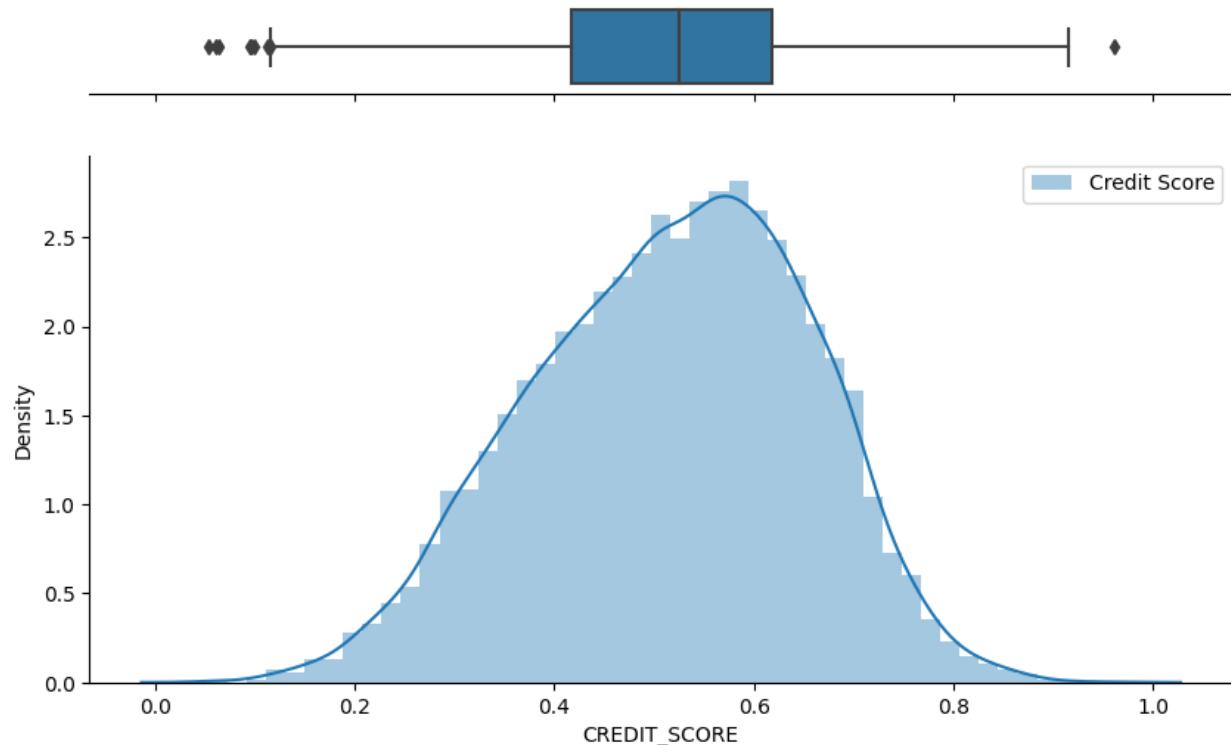
$Q < Q1 - 1.5 * IQR$

Valor mínimo = 6K → **hay 91 outliers**

$Q > Q3 + 1.5 * IQR$

Valor máximo = 18K → **hay 18 outliers**

**Credit Score:**



---

Rangos de cuartiles:

El valor máximo de credit score es 0.92 y la cantidad outliers superiores de credit score es 1. El valor mínimo de credit score es 0.12 y la cantidad outliers inferiores de credit score: 8

Summary:

$Q < Q1 - 1.5 * IQR$

Valor mínimo = 12% → **hay 8 outliers**

$Q > Q3 + 1.5 * IQR$

Valor máximo = 98% → **hay 1 outlier**

### **INSIGHT FINAL OUTLIERS:**

Dado la no significativa cantidad de outliers, tanto para la variable de credit score y kilometraje, decidimos no considerar ni eliminarlos ni imputarlos en primera instancia, ya que veremos como performa el modelo de regresión logística sin este tratamiento, y posteriormente ver como predice si en todo caso, lo eliminamos/tratamos.

---

## **Machine Learning:** "Construcción de Modelos de predicción, utilizando varios algoritmos"

Como nos encontramos con que nuestra variable target es una variable dicotómica, que ocurra o no un siniestro (Target= variable OUTCOME), tenemos un problema de clasificación, donde, en primera instancia, vamos a dividir nuestro proceso en 3 modelos:

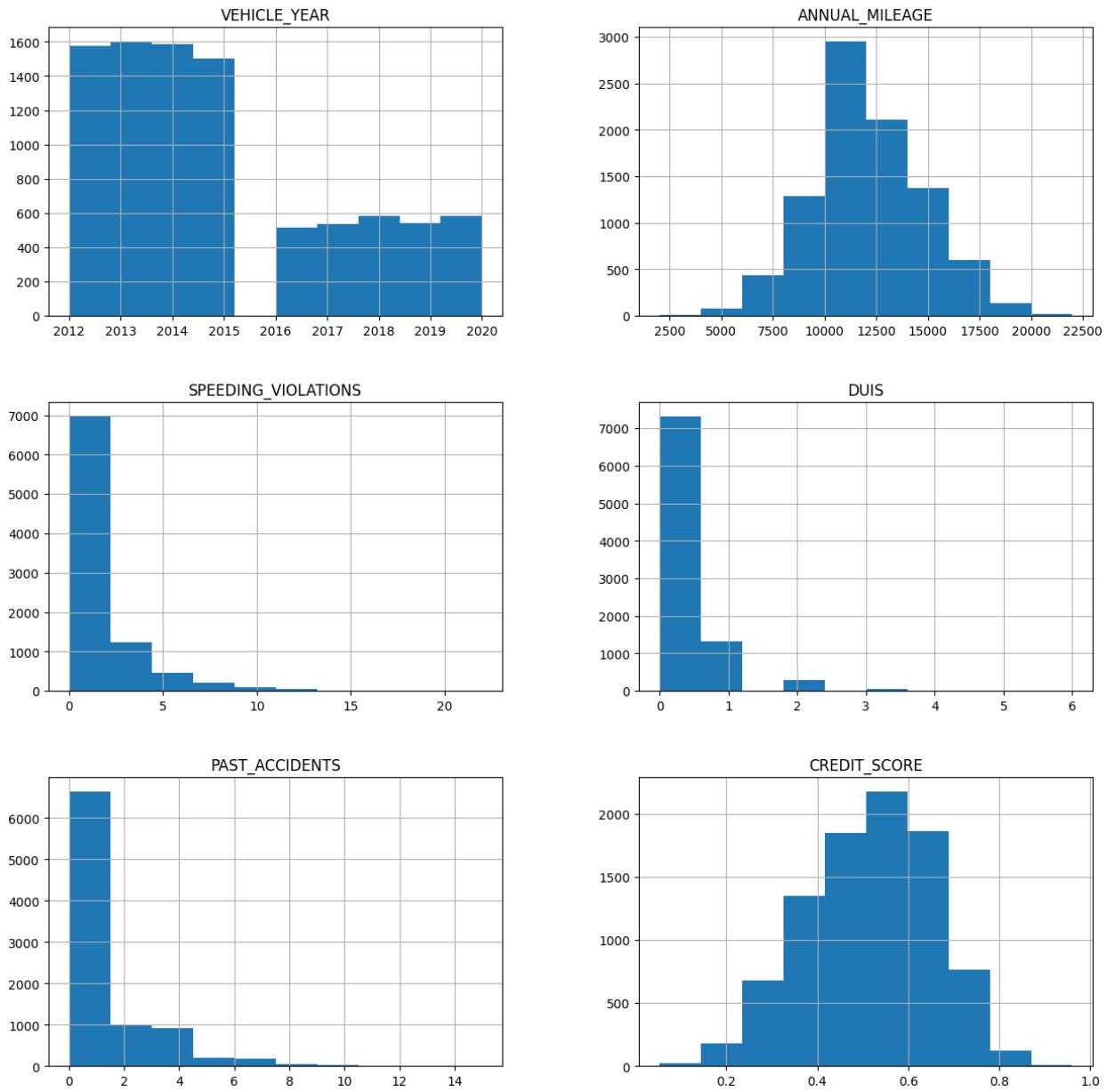
1. **Modelo de regresión logística "Primeros pasos":** Lo llamamos así porque planteamos un modelo "sencillo" sin PCA, que es el modelo actual con el cuál trabajamos en el sector de modelling.
2. **Modelo de regresión logística con PCA:** luego de realizar un análisis de PCA nos encontraremos con otro escenario, pudiendo haber trabajado con la dimensionalidad y haremos el mismo modelo de regresión logística
3. **Árbol de decisión con PCA:** luego de realizar un análisis de PCA, pudiendo haber trabajado con la dimensionalidad usaremos para modelar "árboles de decisión"

### **Modelo de regresión logística "Primeros pasos":**

El primer escenario propuesto por nuestro sector es ver variables principales del auto y de accidentes/multas, como variables independientes y como hipótesis que son determinantes para que se produzca un siniestro o no.

TODO ESTE PREVIO A UN ANÁLISIS DE PCA

#### **Variables:**



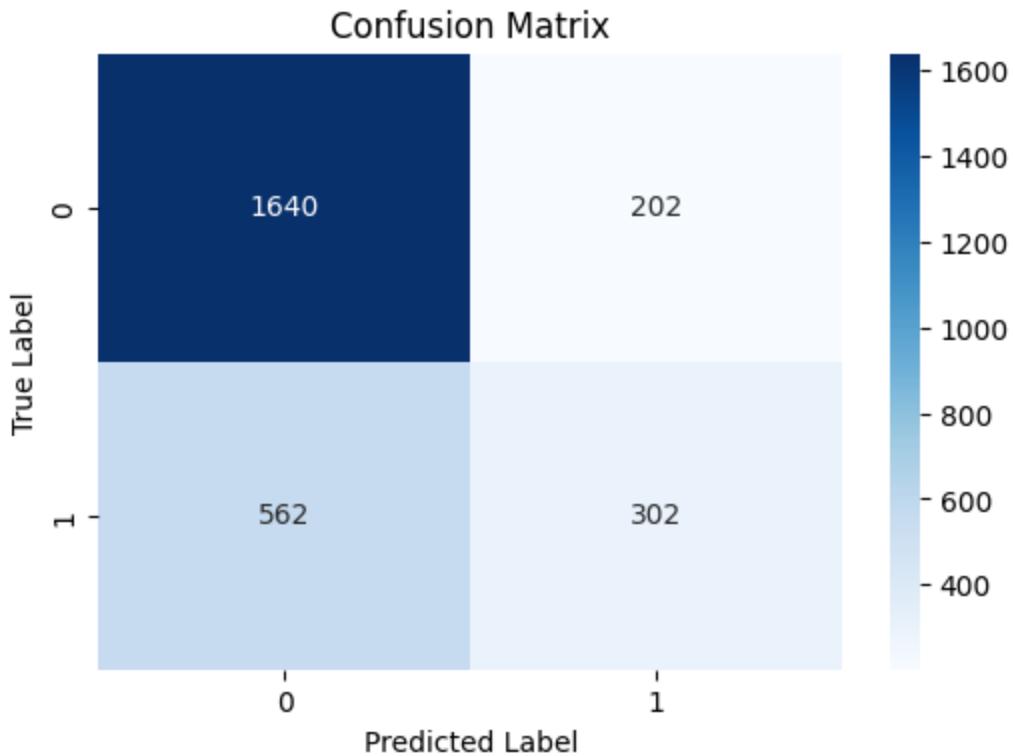
Cada variable en gráfico para ver su distribución, vemos que credit score y annual mileage podrían corresponder con una distribución normal

## MODELADO: Regresión logística - Aprendizaje supervisado

```
[ ] from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)  
  
▶ from sklearn import linear_model  
from sklearn import model_selection  
from sklearn.metrics import classification_report  
from sklearn.metrics import confusion_matrix  
from sklearn.metrics import accuracy_score  
  
model = linear_model.LogisticRegression()  
model.fit(X_train, y_train)  
  
☞ LogisticRegression  
LogisticRegression()
```

Dividimos la base en test y entrenamiento, y procedemos a construir nuestro **modelo de regresión logística**.

#### Matriz de confusión:



A partir de esta matriz podemos obtener nuestras métricas de nuestro problema de clasificación para determinar que “tan buen” predictor es el modelo:

---

## **Accuracy**

% de aciertos sobre el set de entrenamiento: 0.73

% de aciertos sobre el set de evaluación: 0.72

**HIGHLIGHTS:** Un accuracy alto en el conjunto de entrenamiento puede indicar que el modelo está aprendiendo los patrones en los datos de entrenamiento

Una diferencia pequeña entre el accuracy en el conjunto de entrenamiento y el conjunto de prueba podría indicar un buen equilibrio entre ajuste y generalización. Lo cual no es el caso, ambos rondan en el 70% aprox.

## **Precisión**

% de precisión positiva sobre el set de evaluación: 0.60

% de precisión negativa sobre el set de evaluacion: 0.74

**HIGHLIGHTS:** Para la clase 1, que haya siniestros, es cercano a 50%, sugiere que el modelo podría estar adivinando o teniendo un rendimiento similar al azar en la identificación de casos positivos

Para la clase 0, que no haya siniestros, es cercano a 100%, indica que el modelo está haciendo un buen trabajo en identificar correctamente los casos positivos en comparación con los casos falsos positivos.

## **Recall score**

% de recall sobre el set positivo: 0.35

---

% de recall sobre el set negativo: 0.89

**HIGHLIGHTS:**Para la clase 1, que haya siniestros, es cercano a 50%, puede sugerir que el modelo está teniendo dificultades para capturar la mayoría de los casos positivos y podría estar perdiendo una cantidad significativa de casos reales.

Para la clase 0, que no haya siniestros, es cercano a 100%, indica que el modelo está siendo efectivo en identificar la mayoría de los casos negativos en comparación con los casos que realmente son negativos.

### F1 score

% de F1 sobre el set de entrenamiento positivo: 0.45

% de F1 sobre el set de entrenamiento negativo: 0.82

**HIGHLIGHTS:**Si el F1-Score en el conjunto de entrenamiento es significativamente más alto que en el conjunto de prueba, esto puede ser una indicación de overfitting. El modelo está memorizando los datos de entrenamiento y no generaliza bien a nuevos datos.

En nuestro caso no difieren mucho, lo cual podría indicar que no hay overfitting

### FINAL SUMMARY:

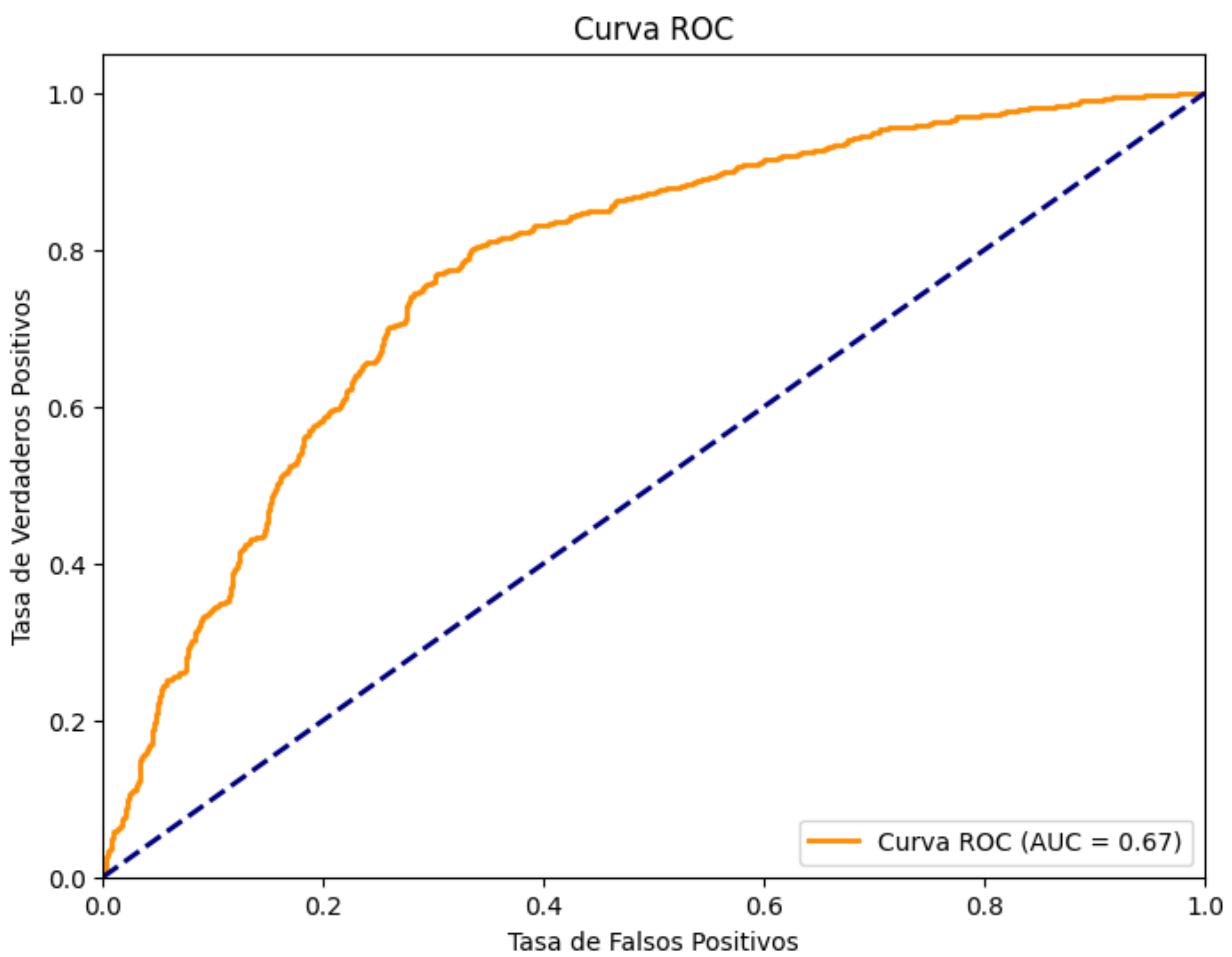
---

	precision	recall	f1-score	support
0	0.74	0.89	0.81	1842
1	0.60	0.35	0.44	864
accuracy			0.72	2706
macro avg	0.67	0.62	0.63	2706
weighted avg	0.70	0.72	0.69	2706

**INSIGHT:** Todas las métricas tienen valores muy altos, para el caso de clase 0, sin siniestros. Pero para el caso de con siniestros son muy bajos los %, lo que lleva a creer que el modelo puede no estar clasificando muy bien las observaciones.

## ROC Curve gráfico y métrica

Es una herramienta gráfica utilizada para evaluar el rendimiento de un modelo de clasificación en diferentes umbrales de decisión. La curva ROC muestra la relación entre la tasa de verdaderos positivos (TPR) y la tasa de falsos positivos (FPR) a medida que se ajusta el umbral de decisión del modelo.



El área bajo la curva ROC (AUC-ROC) es otra métrica importante asociada con la curva ROC. Representa el área debajo de la curva y proporciona una medida de cuán bien el modelo puede distinguir entre las clases. Un valor de AUC-ROC cercano a 1 indica un mejor rendimiento del modelo en la clasificación, que es lo que vamos a calcular:

% de área debajo de curva roc sobre el set de test : 0.67

**INSIGHT:** con un 67% nos indica un buen rendimiento (mayor al 50%) del modelo pero no sabemos si con otro modelo en la clasificación podemos lograr una mejor distinción entre clases.

---

## Modelo de regresión logística con PCA:

### Primero, Análisis de Componentes Principales (PCA)

Ya que tenemos variables cualitativas (dicotómicas) en un análisis PCA, primero vamos a convertir las variables numéricas utilizando técnicas adecuadas, como la codificación de variables dummy. No agrupamos si no reducimos dimensionalidad:

**OBJETIVO:** Encontrar nuevas dimensiones (llamadas componentes principales) que son combinaciones lineales de las dimensiones originales. Estas nuevas dimensiones están ordenadas de tal manera que la primera componente principal captura la máxima varianza en los datos, la segunda componente principal captura la siguiente máxima varianza, y así sucesivamente. Entonces buscamos simplificar datos complejos y mejorar la comprensión y la eficiencia en el análisis de datos.

### Paso 1: convertimos variables categóricas en dummy

```

data_encoded = pd.get_dummies(df_PCA, columns=columns)
data_encoded.dtypes

CREDIT_SCORE                      float64
ANNUAL_MILEAGE                    float64
SPEEDING_VIOLATIONS                int64
DUIS                               int64
PAST_ACCIDENTS                     int64
AGE_26-39                          uint8
AGE_40-64                          uint8
AGE_65+                            uint8
GENDER_male                         uint8
RACE_minority                       uint8
DRIVING_EXPERIENCE_10-19y          uint8
DRIVING_EXPERIENCE_20-29y          uint8
DRIVING_EXPERIENCE_30y+             uint8
EDUCATION_none                      uint8
EDUCATION_university                 uint8
INCOME_poverty                      uint8
INCOME_upper class                  uint8
INCOME_working class                 uint8
VEHICLE_OWNERSHIP_1                 uint8
VEHICLE_YEAR_RANGE_before 2015       uint8
MARRIED_1                           uint8
CHILDREN_1                          uint8
VEHICLE_TYPE_sports car            uint8
OUTCOME_1                           uint8
dtype: object

```

## Paso 2: Separamos las variables numéricas y nos quedamos solo con las numéricas

```

numeric_features = data_encoded.drop(['OUTCOME_1'], axis=1)
data_encoded

   CREDIT_SCORE ANNUAL_MILEAGE SPEEDING_VIOLATIONS DUIS PAST_ACCIDENTS AGE_26-39 AGE_40-64 AGE_65+ GENDER_male RACE_minority ... EDUCATION_university INCOME_poverty INCOME_upper class INCOME_working class
0      0.629027    12000.00000        0     0      0     0     0     1      0      0   ...           0      0      1      0
1      0.357757    16000.00000        0     0      0     0     0     0      1      0   ...           0      1      0      0
2      0.493146    11000.00000        0     0      0     0     0     0      0      0   ...           0      0      0      1
3      0.206013    11000.00000        0     0      0     0     0     0      1      0   ...           1      0      0      0
4      0.388366    12000.00000        2     0      1     1     0     0      0      1   ...           0      0      0      1
...     ...
9013   0.582787    16000.00000        0     0      1     1     0     0      0      0   ...           1      0      1      0
9014   0.522231    11693.45932        1     0      0      1     0     0      0      0   ...           0      0      0      0
9015   0.470940    14000.00000        0     0      0      1     0     0      1      0   ...           0      0      0      0
9016   0.364185    13000.00000        2     0      1     1     0     0      0      0   ...           0      1      0      0
9017   0.435225    13000.00000        0     0      0      1     0     0      0      0   ...           0      0      0      1
9018 rows x 24 columns

```

## Paso 3: Estandarizamos variables

---

## Estandarizar las variables numéricas

```
[ ] scaler = StandardScaler()  
numeric_scaled = scaler.fit_transform(numeric_features)
```

## Paso 4: Aplicamos PCA

```
[ ] # Aplicar PCA  
pca = PCA(n_components=5) # Elegir el número de componentes principales  
principal_components = pca.fit_transform(numeric_scaled)  
pca.fit(df_bis)
```

```
▼       PCA  
PCA(n_components=5)
```

```
► pca.components_  
[ ] array([[-8.51041403e-06,  9.99999961e-01, -2.53296619e-04,  
-2.24109842e-05, -1.15174457e-04],  
[ 1.09370950e-02,  2.78705100e-04,  8.79870767e-01,  
9.25768209e-02,  4.65979901e-01],  
[ 4.20803755e-03, -1.67535106e-05, -4.68322139e-01,  
3.28685717e-03,  8.83541659e-01],  
[ 1.19884833e-02, -3.27770689e-06, -8.03553679e-02,  
9.95615719e-01, -4.63533237e-02],  
[-9.99859464e-01, -5.57276687e-06,  6.69011636e-03,  
1.29640972e-02,  8.25989865e-03]])
```

## Paso 5: vemos como es la correlación con los features tanto en tabla como gráfico

Tabla:

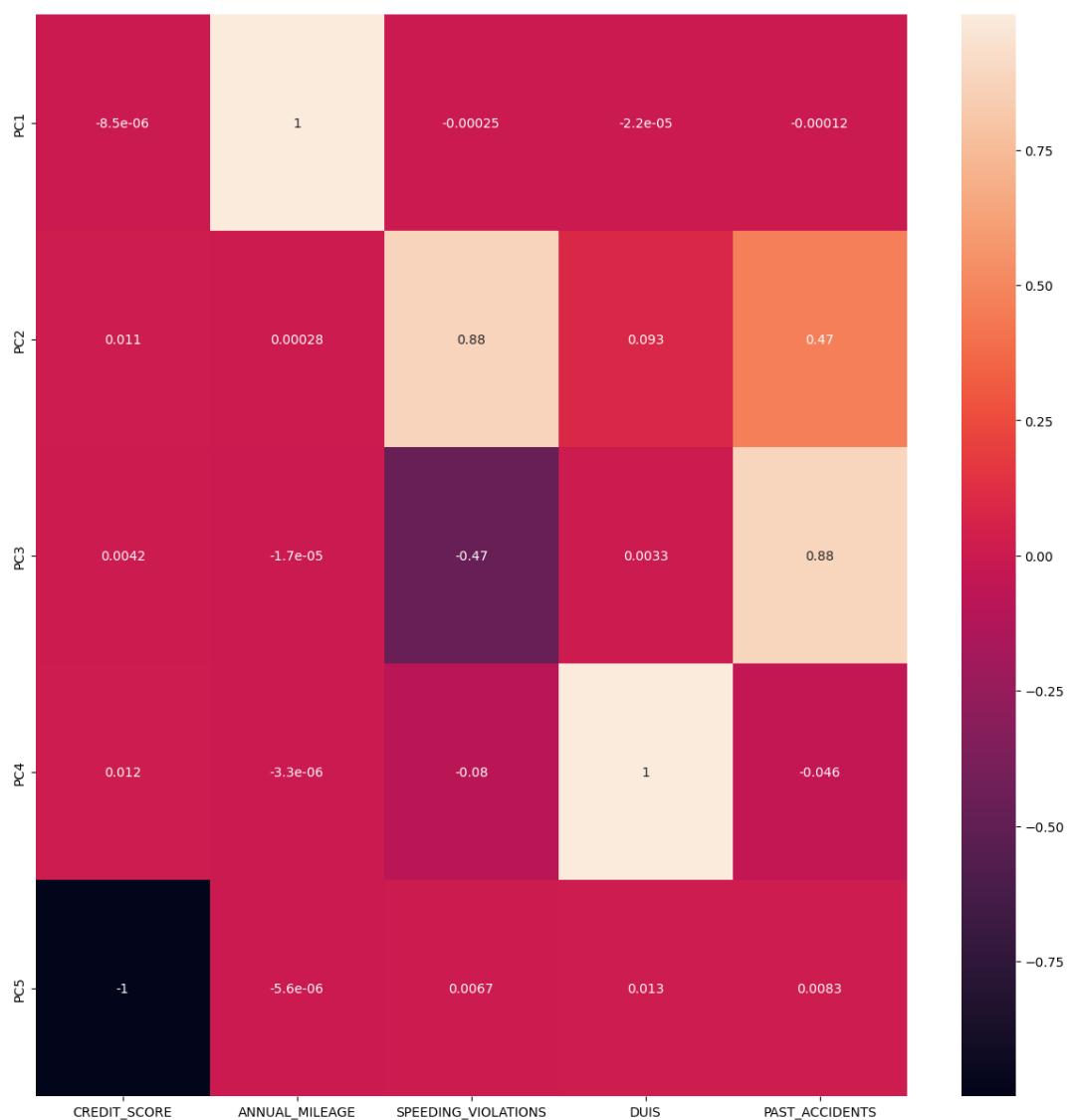
```

lista_componentes = ['PC1', 'PC2', 'PC3', 'PC4', 'PC5']
componentes = pd.DataFrame(data=pca.components_, columns = df_bis.columns, index=lista_componentes)
componentes

```

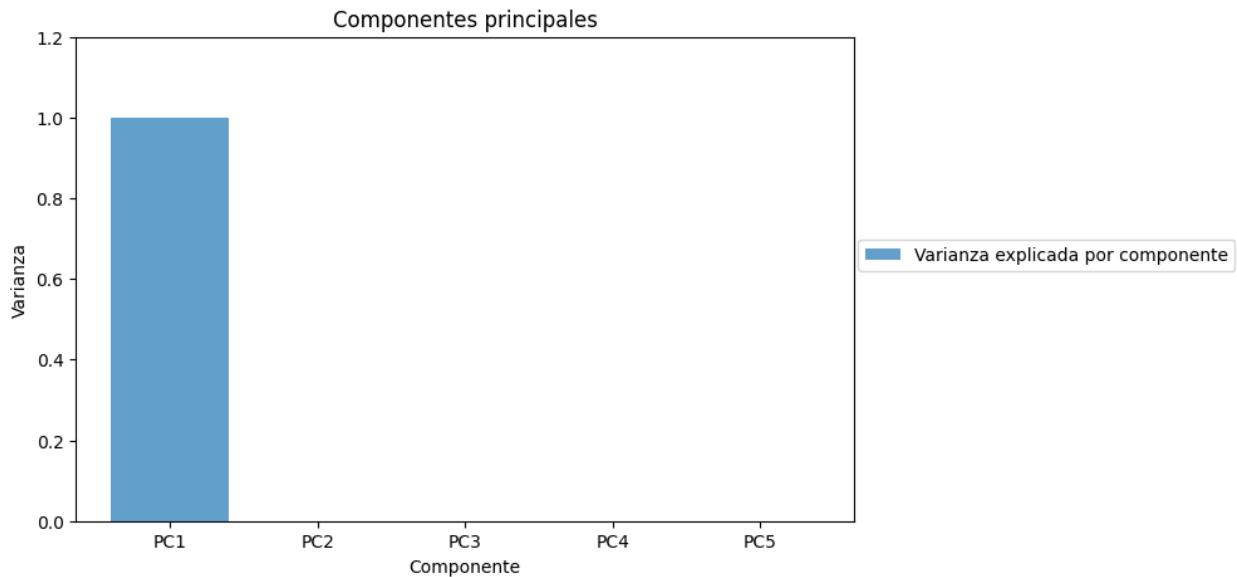
	CREDIT_SCORE	ANNUAL_MILEAGE	SPEEDING_VIOLATIONS	DUIS	PAST_ACCIDENTS	
PC1	-0.000009	1.000000	-0.000253	-0.000022	-0.000115	
PC2	0.010937	0.000279	0.879871	0.092577	0.465980	
PC3	0.004208	-0.000017	-0.468322	0.003287	0.883542	
PC4	0.011988	-0.000003	-0.080355	0.995616	-0.046353	
PC5	-0.999859	-0.000006	0.006690	0.012964	0.008260	

Gráfico:



---

Vemos que credit score tiene correlación negativa con PC5, Annual mileage con PC1, SPEEDING violations con PC2, DUIS con PC4 y Past accidents con PC3, lo cual es bastante variable como se relacionan las 5 variables con los nuevos componentes.



Entonces PC1 explica en gran parte la varianza, por lo cual decidimos reducir la dimensionalidad en esta sola variable.

---

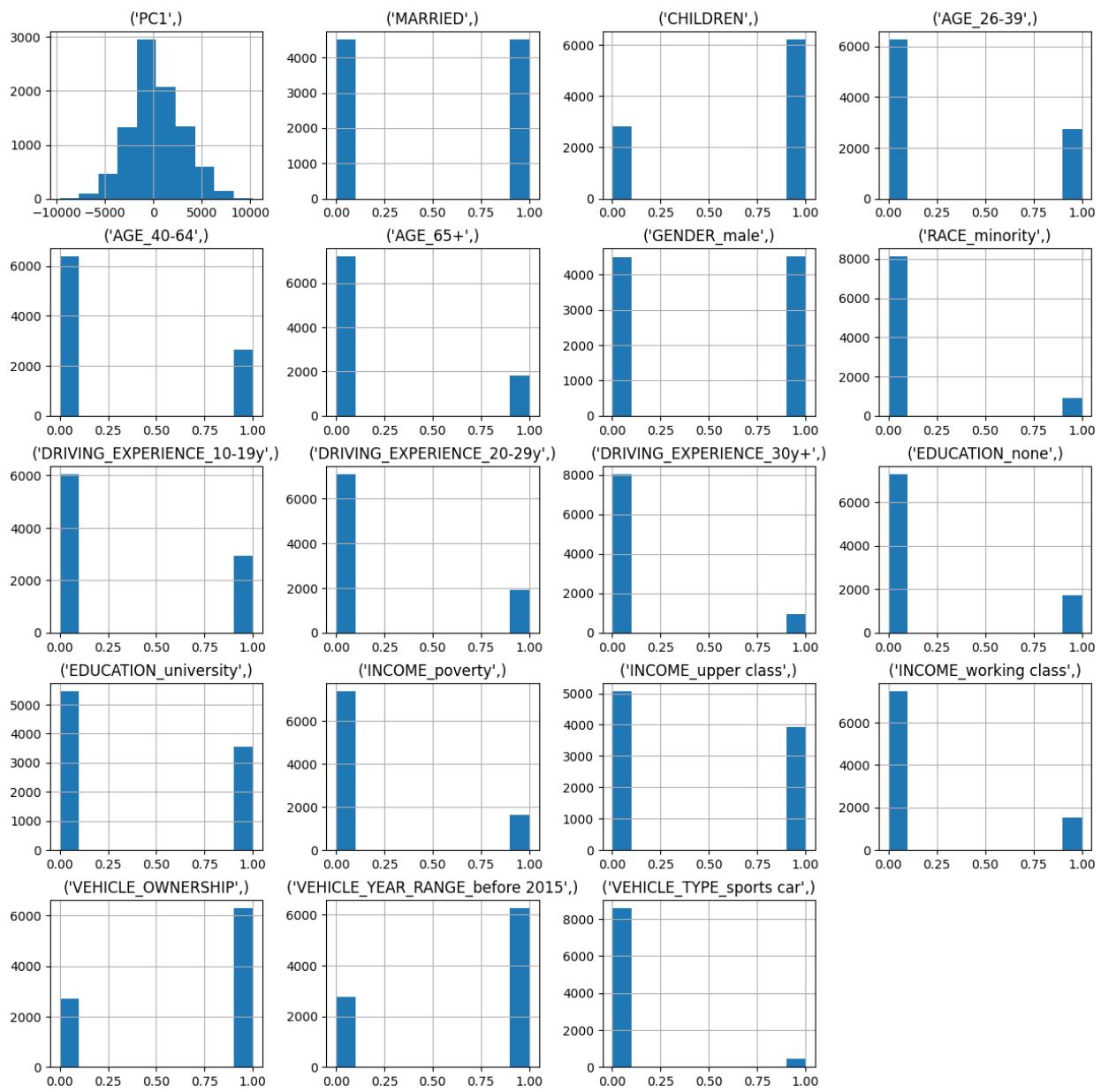
## Paso 6: Concatenamos nuestras variables dummies con PC1

```
# Concatenar los componentes principales con las variables categóricas
final_df = proyecciones
final_df
```

	PC1	MARRIED	CHILDREN	AGE_26-39	AGE_40-64	AGE_65+	GENDER_male	RACE_minority	DRIVING_EXPERIENCE_10-19y	DRIVING_EXPERIENCE_20-29y	DRIVING_EXPERIENCE_30y+	EDUCATION_none	EDUCATION_university	INCOME_poverty
0	306.541169	0	1	0	0	1	0	0	0	0	0	0	0	0
1	4306.541015	0	0	0	0	0	1	0	0	0	0	1	0	1
2	-693.458791	0	0	0	0	0	0	0	0	0	0	0	0	0
3	-693.458788	0	1	0	0	0	1	0	0	0	0	0	1	0
4	306.540549	0	0	1	0	0	1	0	1	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9013	4306.540898	0	0	1	0	0	0	0	1	0	0	0	1	0
9014	0.000249	0	1	1	0	0	0	0	1	0	0	1	0	0
9015	2306.541092	0	1	1	0	0	1	0	0	0	0	0	0	0
9016	1306.540511	0	1	1	0	0	0	0	1	0	0	0	0	1
9017	1306.541132	1	1	1	0	0	0	0	0	0	0	1	0	0

## Comienzo del modelado (Modelo de regresión logística con PCA)

Nuestras variables:



Nuevamente realizamos el split de los datos entre train y test, y corremos nuestro modelo de regresión logística;

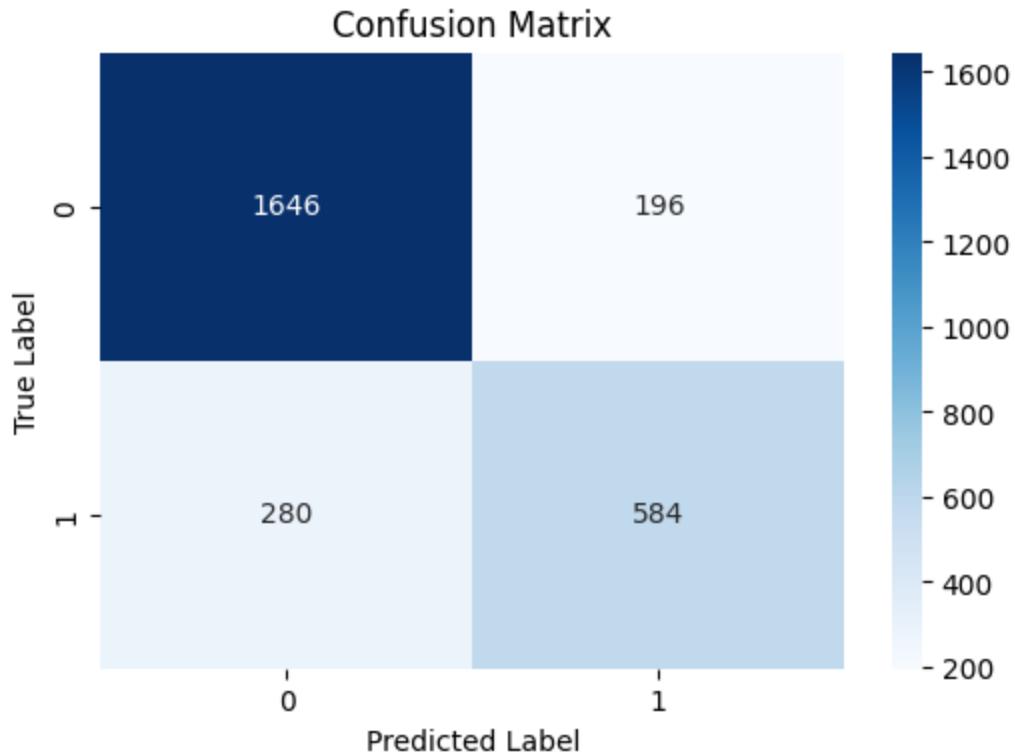
```
[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size=0.3, random_state=42)

▶ model2 = linear_model.LogisticRegression()
    model2.fit(X_test, y_test)

[ ] #Predicciones
    predictions_2_test = model2.predict(X_test)

[ ] #Predicciones
    predictions_2_train = model2.predict(X_train)
```

Calculamos nuestra confusión de matriz;



**INSIGHT:** Verdadero Positivo (VP): Se refiere a los casos en los que el modelo predijo correctamente la clase positiva 1 = 196

Falso Positivo (FP): Representa los casos en los que el modelo predijo

---

incorrectamente la clase positiva cuando en realidad era negativa = 9.1 e+02

Falso Negativo (FN): Indica los casos en los que el modelo predijo incorrectamente la clase negativa cuando en realidad era positiva. 5.5 e +03

Verdadero Negativo (VN): Significa que el modelo predijo correctamente la clase negativa = 6.4 e +02

Pero para una mejor interpretación vamos a considerar el accuracy para interpretar resultados

### Métricas a partir de nuestra matriz de confusión

#### Accuracy

% de aciertos sobre el set de entrenamiento: 0.83

% de aciertos sobre el set de evaluación: 0.82

**HIGHLIGHTS:** Un accuracy alto en el conjunto de entrenamiento puede indicar que el modelo está aprendiendo los patrones en los datos de entrenamiento

Una diferencia pequeña entre el accuracy en el conjunto de entrenamiento y el conjunto de prueba podría indicar un buen equilibrio entre ajuste y generalización. Lo cual no es el caso, ambos rondan en el 80% aprox.

#### Precisión

Ya vemos de por sí, que nuestro análisis con regresión logística en base a nuestra intuición quedarnos solo con las numéricas tiene un menor accuracy (68%) que el nuevo modelo de regresión con PCA (82%)

% de precisión positiva sobre el set de evaluación: 0.75

% de precisión negativa sobre el set de evaluación: 0.85

---

**HIGHLIGHTS:** Para la clase 1, que haya siniestros, es cercano a 100%, indica que el modelo está haciendo un buen trabajo en identificar correctamente los casos positivos en comparación con los casos falsos positivos.

Para la clase 0, que no haya siniestros, es cercano a 100%, indica que el modelo está haciendo un buen trabajo en identificar correctamente los casos negativos en comparación con los casos falsos negativos.

### **Recall score**

% de recall sobre el set positivo: 0.68

% de recall sobre el set negativo: 0.90

**HIGHLIGHTS:** Para la clase 1, que haya siniestros, es cercano a 50%, puede sugerir que el modelo está teniendo dificultades para capturar la mayoría de los casos positivos y podría estar perdiendo una cantidad significativa de casos reales.

Para la clase 0, que no haya siniestros, es cercano a 100%, indica que el modelo está siendo efectivo en identificar la mayoría de los casos negativos en comparación con los casos que realmente son negativos.

### **F1 score**

% de F1 sobre el set de entrenamiento positivo: 0.71

% de F1 sobre el set de entrenamiento negativo: 0.87

---

**HIGHLIGHTS:** Si el F1-Score en el conjunto de entrenamiento es significativamente más alto que en el conjunto de prueba, esto puede ser una indicación de overfitting. El modelo está memorizando los datos de entrenamiento y no generaliza bien a nuevos datos.

en nuestro caso no difieren mucho, lo cual podría indicar que no hay overfitting

Vemos que a comparación de nuestra regresión previa (44%) es mucho mayor y más cerca del 1 (71%)

#### FINAL SUMMARY:

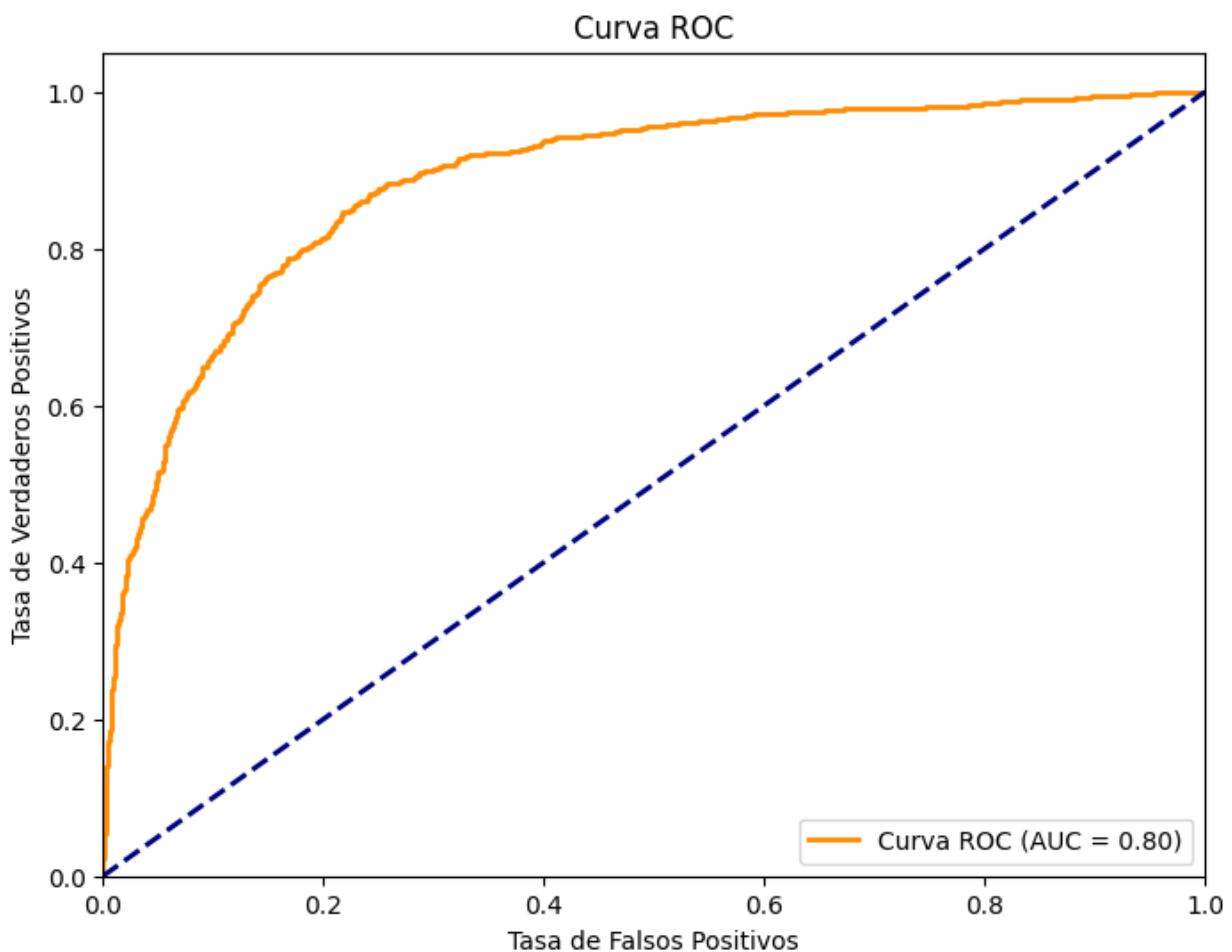
→	precision	recall	f1-score	support
0	0.85	0.89	0.87	1842
1	0.75	0.68	0.71	864
accuracy			0.82	2706
macro avg	0.80	0.78	0.79	2706
weighted avg	0.82	0.82	0.82	2706

**INSIGHT:** Todas las métricas tienen valores muy altos, a diferencia del modelo previo, en algunos clusters más que en otros, lo que lleva a creer que el modelo es muy bueno clasificando las observaciones.

---

## ROC Curve gráfico y métrica

Curve ROC: Como ya mencionamos el área debajo de la curva ROC proporciona una medida de cuán bien el modelo puede distinguir entre las clases. Un valor de AUC-ROC cercano a 1 indica un mejor rendimiento del modelo en la clasificación.



**INSIGHT:** Vemos que con un 80% logramos una mejor distinción entre clases que la regresión logística previa con un área debajo de la curva = 67%

---

## Modelo de Árbol de Decisión con PCA:

Concepto: El objetivo del árbol de decisión es dividir los datos de manera que las clases o los valores predichos sean lo más homogéneos posible en cada hoja del árbol. Este enfoque facilita la interpretación y la toma de decisiones basada en reglas simples.

Comenzamos utilizando la reducción de la dimensionalidad en PC1 (Proceso PCA), y a partir de ese dataset modelamos con el Árbol de decisión, previamente dividiendo nuevamente nuestros datos en train test:

```
[ ] from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size=0.3, random_state=42)

❶ from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn import tree
# Crear la instancia del modelo
clf = DecisionTreeClassifier(max_depth=6)

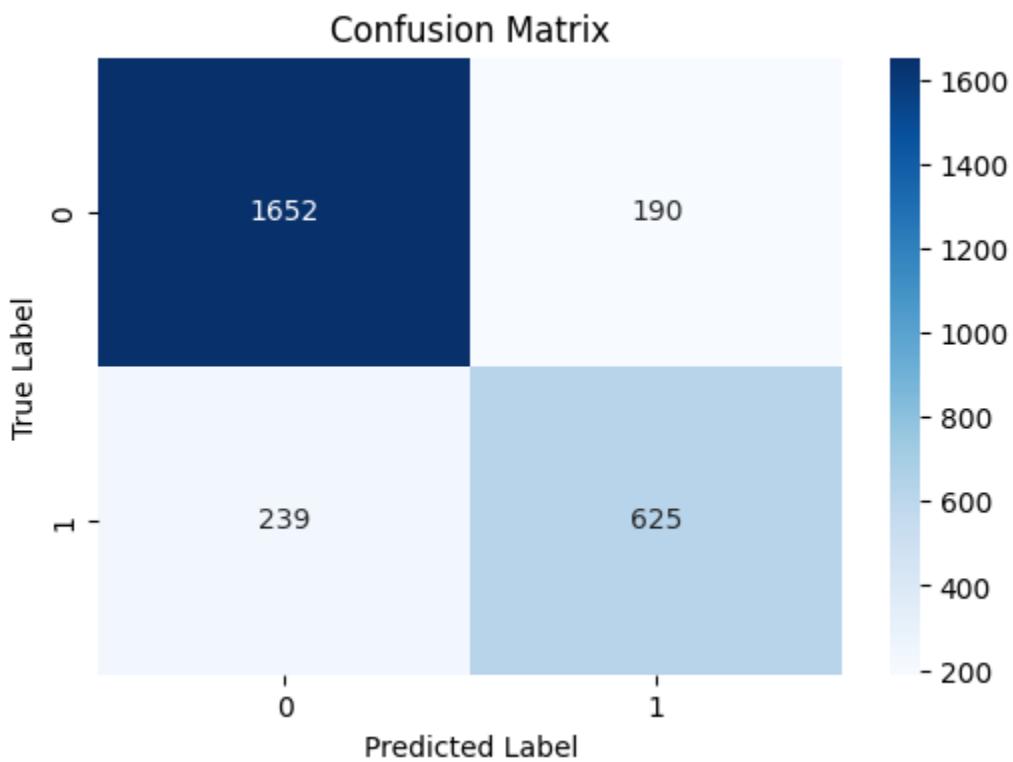
#X = ya las definimos desde PCA (PC1 ) y tambien consideramos las dicotomicas
#y = outcome

clf = clf.fit(X_test, y_test)

[ ] predictions_3_test = clf.predict(X_test)

[ ] predictions_3_train = clf.predict(X_train)
```

Construimos nuestra matriz de confusión:



## Métricas a partir de nuestra matriz de confusión

### Accuracy

% de aciertos sobre el set de entrenamiento: 0.82

% de aciertos sobre el set de evaluación: 0.84

**HIGHLIGHTS:** Un accuracy alto en el conjunto de entrenamiento puede indicar que el modelo está aprendiendo los patrones en los datos de entrenamiento

Una diferencia pequeña entre el accuracy en el conjunto de entrenamiento y el conjunto de prueba podría indicar un buen equilibrio entre ajuste y generalización. Lo cual no es el caso, ambos rondan en el 80% aprox.

### Precisión

% de precisión positiva sobre el set de evaluación: 0.77

% de precisión negativa sobre el set de evaluación: 0.87

---

**HIGHLIGHTS:** Para la clase 1, que haya siniestros, es cercano a 100%, indica que el modelo está haciendo un buen trabajo en identificar correctamente los casos positivos en comparación con los casos falsos positivos.

Para la clase 0, que no haya siniestros, es cercano a 100%, indica que el modelo está haciendo un buen trabajo en identificar correctamente los casos negativos en comparación con los casos falsos negativos.

### **Recall score**

% de recall sobre el set positivo: 0.72

% de recall sobre el set negativo: 0.90

**HIGHLIGHTS:** Para la clase 1, que haya siniestros, es cercano a 50%, puede sugerir que el modelo está teniendo dificultades para capturar la mayoría de los casos positivos y podría estar perdiendo una cantidad significativa de casos reales.

Para la clase 0, que no haya siniestros, es cercano a 100%, indica que el modelo está siendo efectivo en identificar la mayoría de los casos negativos en comparación con los casos que realmente son negativos.

### **F1 score**

% de F1 sobre el set de entrenamiento positivo: 0.74

% de F1 sobre el set de entrenamiento negativo: 0.88

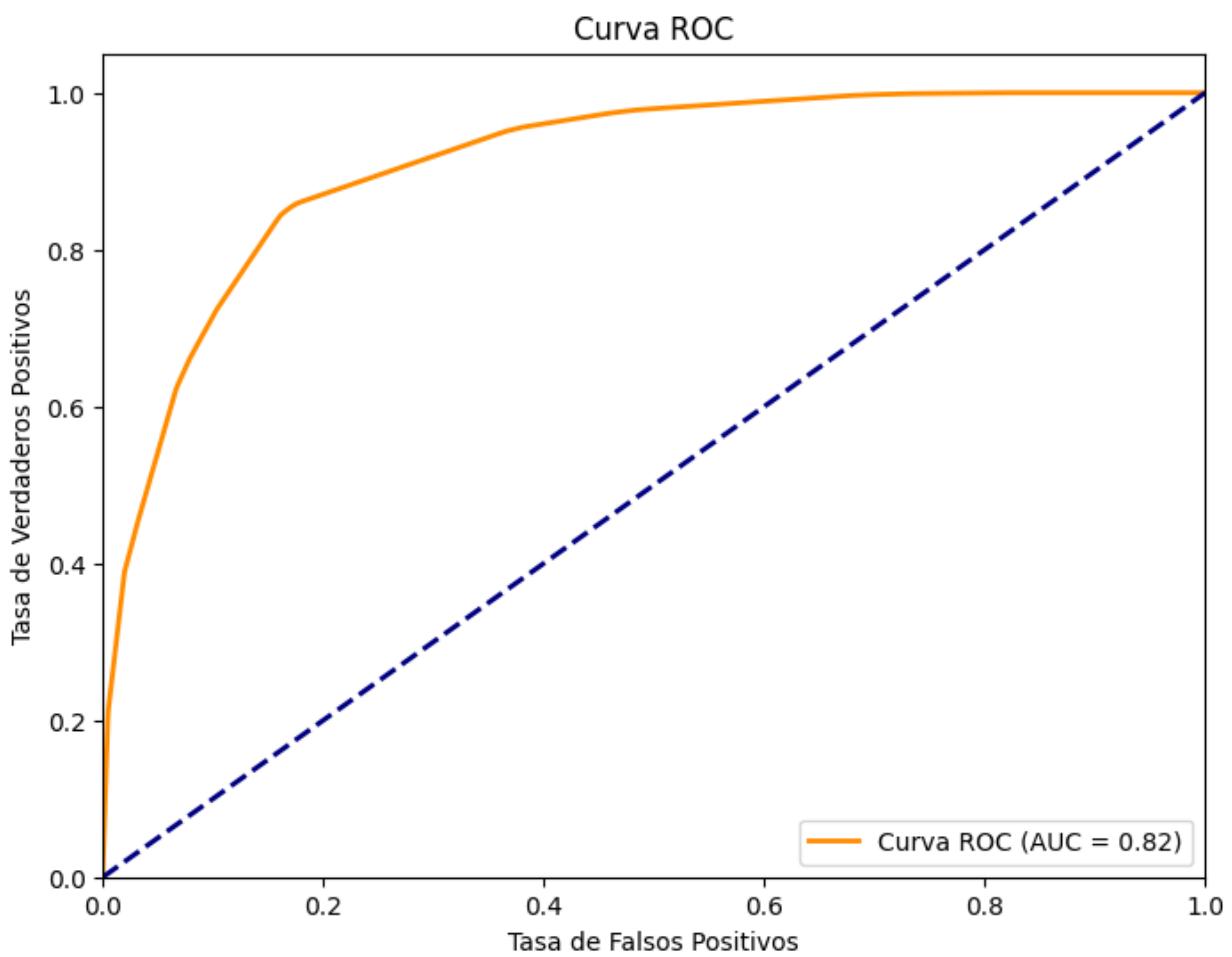
---

**HIGHLIGHTS:** Si el F1-Score en el conjunto de entrenamiento es significativamente más alto que en el conjunto de prueba, esto puede ser una indicación de overfitting. El modelo está memorizando los datos de entrenamiento y no generaliza bien a nuevos datos.

En nuestro caso no difieren mucho, lo cual podría indicar que no hay overfitting

### **ROC Curve gráfico y métrica**

Curve ROC: Como ya mencionamos el área debajo de la curva ROC proporciona una medida de cuán bien el modelo puede distinguir entre las clases. Un valor de AUC-ROC cercano a 1 indica un mejor rendimiento del modelo en la clasificación.



**INSIGHT:** Vemos que con un 82% logramos una mejor distinción entre clases que la regresión logística previa con un área debajo de la curva = 80%

### Summary final de Evaluación de los modelos

Modelo	Accuracy	Precisión	Recall	F1 score	Área Curva ROC
Regresión Logística	72.8%	59.9%	35.0%	44.6%	68.3%
Regresión Logística (CON PCA)	82.8%	74.9%	67.6%	71.1%	80.4%
Árbol de Decisión (CON PCA)	83.5%	76.7%	72.3%	74.0%	82.0%

### INSIGHTS FINALES

---

## Accuracy

Todos los modelos seleccionados tienen un accuracy cercano al 1, lo cual indica que el modelo está haciendo un buen trabajo en predecir las clases correctamente. Pero el modelo de árboles de decisión es con el mayor accuracy, es decir que el rendimiento de un modelo de clasificación según accuracy es el mejor.

Sin embargo, es importante recordar que la precisión no siempre es la métrica adecuada para evaluar el rendimiento del modelo, especialmente cuando hay un desequilibrio entre las clases o cuando los costos de los errores varían. En esos casos, otras métricas como la exhaustividad (recall), el F1-Score y la matriz de confusión pueden proporcionar una imagen más completa del rendimiento del modelo en diferentes situaciones.

## F1 score

Los modelos con PCA: Indican, con un F1 score cercano a 1, un buen equilibrio entre precisión y exhaustividad. El modelo está haciendo un buen trabajo en predecir tanto las clases positivas como las negativas.

Solo el modelo de regresión logística sin PCA, tiene un F1 score con valor cercano al 50%, puede sugerir que el modelo está desequilibrado hacia la precisión o la exhaustividad. Es decir, el modelo podría estar sacrificando la capacidad de detectar casos positivos para mejorar la precisión, o viceversa.

Siendo que el modelo de árbol de decisión tiene el mayor F1 score lo cual es el modelo con mejor equilibrio de precisión y exhaustividad

## ROC Curve

Los modelos tienen un área cercana a 1, lo cual indica que el modelo tiene un excelente rendimiento. Cuanto más cerca esté del valor 1.0, mejor es la capacidad del modelo para

---

distinguir entre las clases y para predecir correctamente en diferentes umbrales de decisión. Solamente el modelo sin PCA tiene un valor de área debajo de la curva cercano al 50%, lo cual indica que el modelo tiene un rendimiento similar al azar. En este caso, el modelo no tiene una capacidad significativamente mejor que una elección aleatoria, lo que puede indicar problemas con el modelo o el conjunto de datos.

Nuevamente el modelo de árbol de decisión tiene mejor rendimiento del modelo en términos de su capacidad para distinguir entre las clases

El modelo de árbol de decisión sería entonces el modelo que tiene el mejor rendimiento, según Accuracy, F1 score y ROC curve.

### **Overfitting or Underfitting**

Accuracy:

Primero vemos que ninguna medida de accuracy tiene valor 1.0 o 100%: Significa que todas las predicciones realizadas por el modelo son correctas, lo cual es raro y podría indicar un posible problema, como overfitting.

Detectar el underfitting mediante el F1-Score implica observar cómo esta métrica se comporta en los conjuntos de entrenamiento y prueba. El F1-Score es una métrica que combina tanto la precisión como la exhaustividad (recall), por lo que puede proporcionar información sobre cómo el modelo está equilibrando la capacidad de detectar casos positivos con la precisión de sus predicciones. Aquí tienes algunas señales que podrían sugerir underfitting en función del F1-Score:

---

Bajo F1-Score en Conjunto de Entrenamiento y Prueba: Si tanto el F1-Score en el conjunto de entrenamiento como en el conjunto de prueba son bajos, esto podría indicar que el modelo no está capturando correctamente las relaciones entre las características y las etiquetas de clase. El modelo no está logrando un equilibrio entre precisión y exhaustividad.

En todos los modelos podemos concluir que puede no haber ni overfitting ni underfitting

---

## **Machine Learning:** “Random Forest, profundizando y desafiando nuestros modelos y hiperparámetros”

En esta sección en primera instancia vamos a modelar con un nuevo modelo (**Random Forest**) y vamos a buscar nuestros mejores hiperparametros.

Luego analizaremos el desbalance de nuestra base y cómo podemos, mediante distintos métodos, realizar un fix para lograr una **base balanceada**.

Por otra parte, consideraremos en lugar de usar PCA usar **MCA** para variables categóricas.

Y por último, vamos a evaluar el rendimiento del modelo de Random Forest mediante **cross validation**.

### **Análisis de Hiperparámetros & Random Forest**

Encontrar los mejores hiperparámetros puede mejorar significativamente el rendimiento del modelo.

En nuestro caso utilizaremos el enfoque de **Búsqueda Exhaustiva (Grid Search)**, el cual define un conjunto de valores para cada hiperparámetro que deseas ajustar.

Realiza una búsqueda exhaustiva, entrenando y evaluando el modelo para todas las combinaciones posibles de hiperparámetros.

### **HIPERPARAMETROS RANDOM FOREST CLASSIFIER**

**max\_depth:** Limita la profundidad máxima de los árboles. Puede ayudar a evitar el sobreajuste. El valor predeterminado es None, lo que significa que los árboles crecerán hasta que todos los nodos contengan menos de min\_samples\_split ejemplos.

**min\_samples\_leaf:** El número mínimo de ejemplos requeridos en una hoja. Controla el tamaño mínimo de las hojas del árbol. El valor predeterminado es 1.

---

**n\_estimators:** Este parámetro controla el número de árboles en el bosque aleatorio. Un valor más alto generalmente conduce a un mejor rendimiento, pero también aumenta la complejidad y el tiempo de entrenamiento. El valor predeterminado es 100.

### Modelo Random Forest utilizando enfoque de Grid Search:

```
RF = RandomForestClassifier()
hiperparametros={'max_depth':[15,20], 'min_samples_leaf':[40,30], 'max_leaf_nodes':[40,30], 'n_estimators':[100,90]}
grilla=GridSearchCV(RF, hiperparametros, cv=5, scoring='accuracy', verbose=3)
grilla.fit(X_train,y_train)

#max_depth: define la profundidad máxima permitida
#min_samples_leaf: establece el número mínimo de muestras requeridas en un nodo hoja del árbol.
#max_leaf_nodes: limita el número máximo de nodos hoja que puede tener un árbol
#n_estimators: indica la cantidad de árboles

Fitting 5 folds for each of 16 candidates, totalling 80 fits
```

### Nuestros mejores hiperparámetros:

```
grilla.best_params_

{'max_depth': 15,
 'max_leaf_nodes': 30,
 'min_samples_leaf': 30,
 'n_estimators': 100}
```

### Corremos nuestro modelo con esos hiperparámetros:

```
[ ] # Fit best model
modelo_RF=grilla.best_estimator_
modelo_RF.fit(X_train, y_train)

RandomForestClassifier(max_depth=15, max_leaf_nodes=30, min_samples_leaf=30)

▶ #hago las predicciones con x_test, x_train
y_pred_test = modelo_RF.predict(X_test)
y_pred_train = modelo_RF.predict(X_train)
```

---

## Evaluación del modelo:

### Accuracy

% de aciertos sobre el set de entrenamiento: 0.82

% de aciertos sobre el set de evaluación: 0.82

**HIGHLIGHTS:** Un accuracy alto en el conjunto de entrenamiento puede indicar que el modelo está aprendiendo los patrones en los datos de entrenamiento

Una diferencia pequeña entre el accuracy en el conjunto de entrenamiento y el conjunto de prueba podría indicar un buen equilibrio entre ajuste y generalización. Lo cual no es el caso, ambos rondan en el 82% aprox.

### Precisión

% de precisión positiva sobre el set de evaluación: 0.76

% de precisión negativa sobre el set de evaluación: 0.84

**HIGHLIGHTS:** Para la clase 1, que haya siniestros, es cercano a 100%, indica que el modelo está haciendo un buen trabajo en identificar correctamente los casos positivos en comparación con los casos falsos positivos.

Para la clase 0, que no haya siniestros, es cercano a 100%, indica que el modelo está haciendo un buen trabajo en identificar correctamente los casos negativos en comparación con los casos falsos negativos.

---

## Recall score

% de recall sobre el set positivo: 0.63

% de recall sobre el set negativo: 0.90

**HIGHLIGHTS:** Para la clase 1, que haya siniestros, es cercano a 50%, puede sugerir que el modelo está teniendo dificultades para capturar la mayoría de los casos positivos y podría estar perdiendo una cantidad significativa de casos reales.

Para la clase 0, que no haya siniestros, es cercano a 100%, indica que el modelo está siendo efectivo en identificar la mayoría de los casos negativos en comparación con los casos que realmente son negativos.

## F1 score

% de F1 sobre el set de entrenamiento positivo: 0.69

% de F1 sobre el set de entrenamiento negativo: 0.87

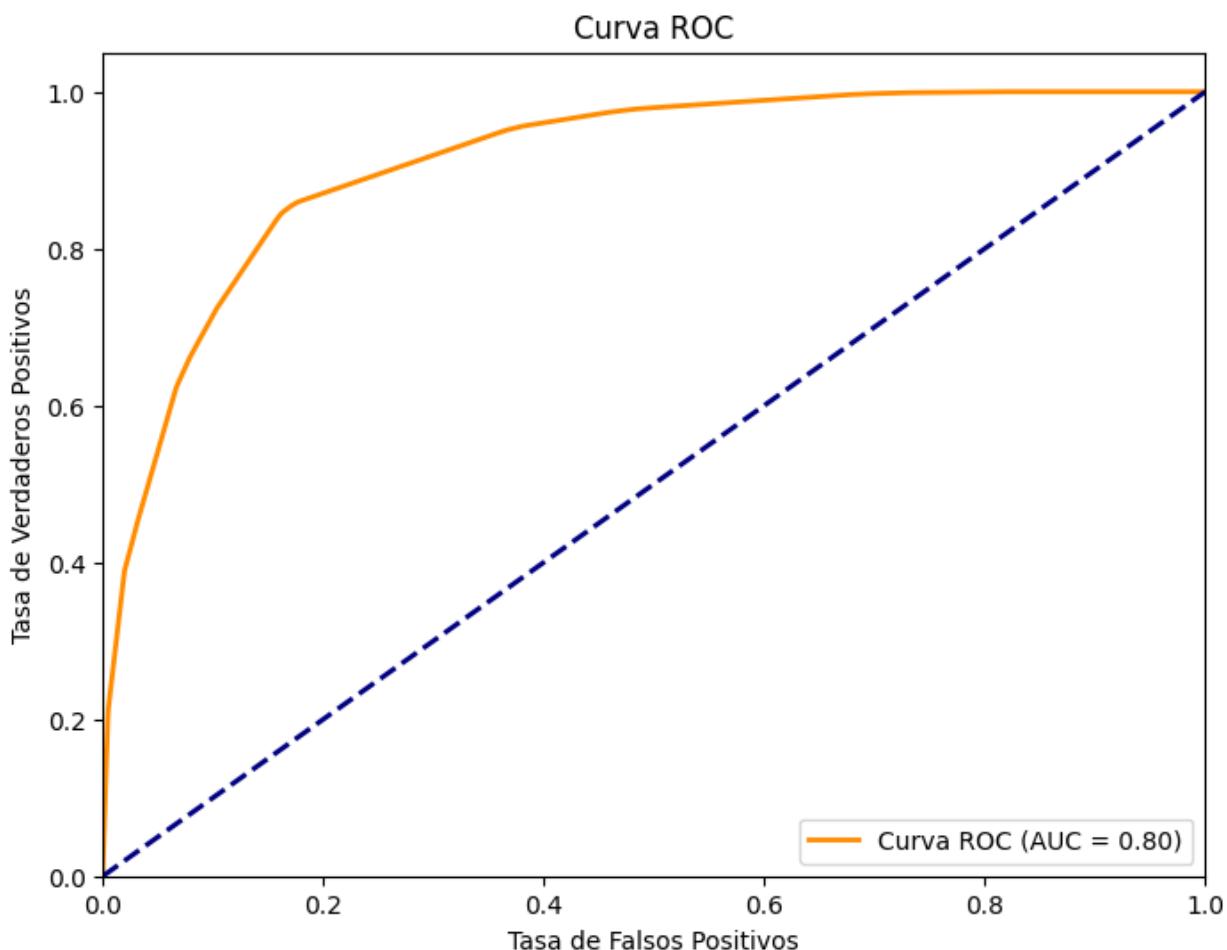
**HIGHLIGHTS:** Si el F1-Score en el conjunto de entrenamiento es significativamente más alto que en el conjunto de prueba, esto puede ser una indicación de overfitting. El modelo está memorizando los datos de entrenamiento y no generaliza bien a nuevos datos.

En nuestro caso no difieren mucho (69,3% vs 69,3%) lo cual podría indicar que no hay overfitting

---

## ROC Curve gráfico y métrica

Curve ROC: Como ya mencionamos el área debajo de la curva ROC proporciona una medida de cuán bien el modelo puede distinguir entre las clases. Un valor de AUC-ROC cercano a 1 indica un mejor rendimiento del modelo en la clasificación.



**INSIGHT:** Vemos que con un 80% logramos una mejor distinción entre clases que los modelos previos.

## Summary comparativo modelos

Modelo	Accuracy	Precisión	Recall	F1 score	Área Curva ROC
Regresión Logística	72.8%	59.9%	35.0%	44.6%	68.3%
Regresión Logística (CON PCA)	82.8%	74.9%	67.6%	71.1%	80.4%
Árbol de Decisión (CON PCA)	83.5%	76.7%	72.3%	74.0%	82.0%
Random Forest con Grid Search	82.4%	75.7%	63.2%	69.3%	80.2%

## INSIGHT FINAL:

Sumamos random forest e intentamos buscarlos mejores hiperparámetros, pero con los mejores, si bien logramos un % de accuracy cercano al 100%, tenemos un modelo con una menor accuracy, precisión, F1 score y ROC curve que el mejor modelo visto de árbol de decisión (sin la búsqueda de los best hiperparametros), pero sí podemos observar que la diferencia entre F1 score test y train árbol de decisión era de 71% vs 74%, con Random Forest es mucho menor de 69% vs 69%, lo cuál reducimos la posibilidad que haya overfitting.

## SMOTE, RandomOverSampler & SMOTETomek: Balanceo de base y variables sintéticas.

Primero vamos a chequear con nuestra última construcción del modelo (Random Forest con mejora de hiperparámetros) si hay un desbalance de la base o no:

```
[ ] values_counts_originales = np.asarray(np.unique(y1, return_counts=True))
values_counts_originales

array([[ 0,  1],
       [6189, 2829]])
```

Vemos como la base se encuentra desbalanceada, hay 6.2 K de asegurados sin siniestros, es decir, el 68.6% de la base.

---

Vamos a volver a realizar el split de nuestra base pero esta vez vamos a incorporar stratify=y1, que asegura que las proporciones de las clases en y\_train y y\_test sean similares a la proporción original de y.

```
X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size=0.1, random_state=42, stratify=y)

values_counts_y_train = np.asarray(np.unique(y_train, return_counts=True))
values_counts_y_train

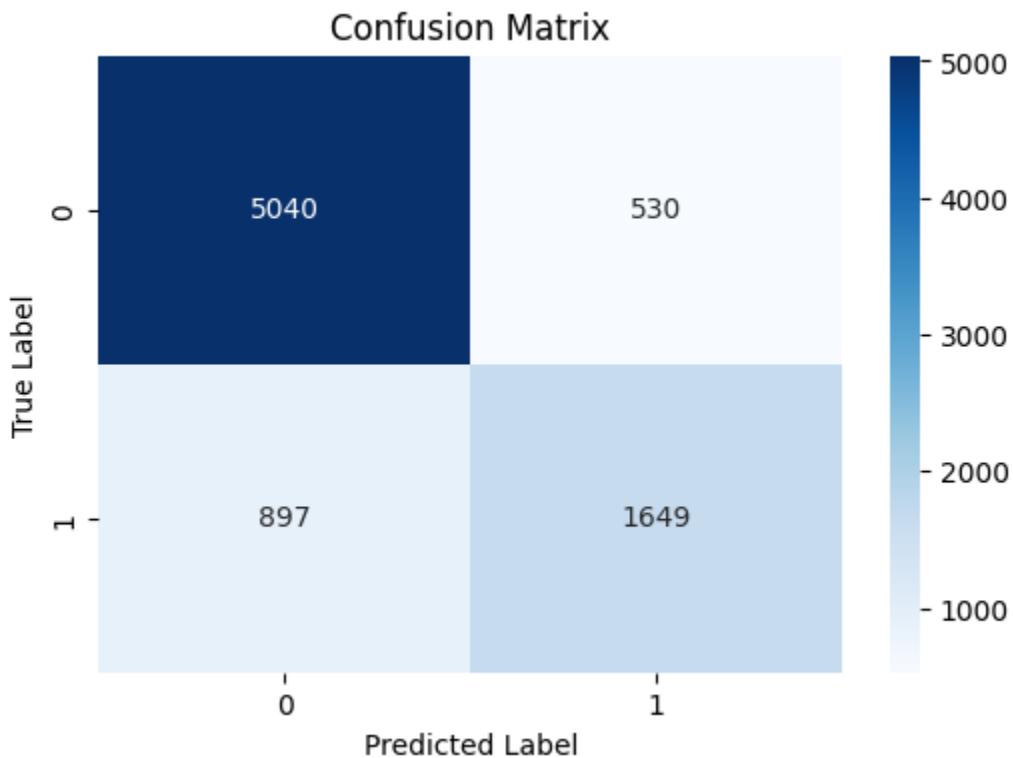
array([[ 0,   1],
       [5570, 2546]])
```

Vamos a volver a utilizar nuestro modelo de random forest, donde reducimos la posibilidad que haya overfitting con este nuevo "split":

```
RF = RandomForestClassifier()
hiperparametros={'max_depth': [15,20], 'min_samples_leaf': [40,30], 'max_leaf_nodes': [40,30], 'n_estimators': [100,90]}
grilla=GridSearchCV(RF, hiperparametros, cv=5, scoring='accuracy', verbose=3)
grilla.fit(X_train,y_train)

#max_depth: define la profundidad máxima permitida
#min_samples_leaf: establece el número mínimo de muestras requeridas en un nodo hoja del árbol.
#max_leaf_nodes: limita el número máximo de nodos hoja que puede tener un árbol
#n_estimators: indica la cantidad de árboles
```

Volvemos a ver la performance de nuestro modelo con la matriz de confusión:



Y este es nuestro resumen sin balanceo

	precision	recall	f1-score	support
0	0.87	0.91	0.89	619
1	0.77	0.70	0.74	283
accuracy			0.84	902
macro avg	0.82	0.80	0.81	902
weighted avg	0.84	0.84	0.84	902

## SMOTE

Ahora si, vamos a "balancear" nuestra base con la técnica "SMOTE": crea nuevas muestras interpolando entre instancias de la clase minoritaria existente. Esto se hace tomando una instancia de la clase minoritaria (OUTCOME = 1) eligiendo vecinos similares y creando instancias sintéticas a lo largo de las líneas que conectan la instancia original con sus vecinos.

```
# Aplicar SMOTE en el conjunto de entrenamiento
from imblearn.over_sampling import SMOTE
smote = SMOTE(k_neighbors=5, random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

#Comprobemos que las clases estan balanceadas
values_counts_SMOTE = np.asarray(np.unique(y_train_smote, return_counts=True))
values_counts_SMOTE

array([[ 0,  1],
       [5570, 5570]])
```

```
# Entrenar un Random Forest en los datos con SMOTE
RF_SMOTE = RandomForestClassifier()
hiperparametros={'max_depth':[15,20], 'min_samples_leaf':[40,30], 'max_leaf_nodes':[40,30], 'n_estimators':[100,90]}
grilla=GridSearchCV(RF_SMOTE, hiperparametros, cv=5, scoring='accuracy', verbose=3)
grilla.fit(X_train_smote, y_train_smote)
```

Nuevas métricas:

	precision	recall	f1-score	support
0	0.90	0.84	0.87	619
1	0.69	0.81	0.75	283
accuracy			0.83	902
macro avg	0.80	0.82	0.81	902
weighted avg	0.84	0.83	0.83	902

## INSIGHT

Vemos como ciertas métricas mejoran con nuestra base balanceada sobre todo en nuestra clase minoritaria (1), excepto la precisión, que pasa de 77% a 70%, pero si ponemos foco en el recall: vemos mejoró muchísimo, de 68% a 81%, está calculando pocos falsos negativos, y buscamos optimizar esa métrica porque no es lo mismo hacer una reserva pequeña en nuestra compañía, pensando que ciertos usuarios no van a tener siniestros, cuando si van a siniestrarse.

## RandomOverSampler

---

Vamos a "balancear" nuestra base con la técnica dos "RandomOverSampler": crea copias aleatorias de ejemplos de la clase minoritaria (con siniestros, OUTCOME =1 )hasta que las proporciones de las clases sean más equitativas.

```
[ ] from imblearn.over_sampling import RandomOverSampler  
  
[ ] # Aplicar Random Oversampling  
ros = RandomOverSampler(random_state=42)  
X_train_resampled, y_train_resampled = ros.fit_resample(X_train, y_train)  
  
[ ] #Comprobemos que las clases estan balabceadas  
values_counts_ros = np.asarray(np.unique(y_train_resampled, return_counts=True))  
values_counts_ros  
  
array([[ 0,  1],  
       [5570, 5570]])  
  
[ ] # Entrenar un Random Forest en los datos con RANDOMOVERSAMPLER  
RF_ros = RandomForestClassifier()  
hiperparametros={'max_depth':[15,20], 'min_samples_leaf':[40,30], 'max_leaf_nodes':[40,30], 'n_estimators':[100,90]}  
grilla=GridSearchCV(RF_ros, hiperparametros, cv=5, scoring='accuracy', verbose=3)  
grilla.fit(X_train_resampled, y_train_resampled)
```

## Métricas:

	precision	recall	f1-score	support
0	0.92	0.83	0.87	619
1	0.70	0.83	0.76	283
accuracy			0.83	902
macro avg	0.81	0.83	0.82	902
weighted avg	0.85	0.83	0.84	902

## INSIGHT:

Vemos una leve mejora de las métricas entre SMOTE vs RandomOver Sampler, sobre todo en la minoría de los caso, el recall pasa de 81% a 83%

## SMOTE Tomek

Vamos a balancear nuestra base con la técnica tres "SMOTETomek": combina SMOTE con el método Tomek, donde las instancias sintéticas generadas por SMOTE se combinan con la

---

eliminación de los ejemplos redundantes proporcionados por el método Tomek. Esto ayuda a mejorar la calidad de las instancias generadas por SMOTE y puede resultar en un conjunto de datos más equilibrado y mejor separado.

```
| from imblearn.combine import SMOTETomek  
  
| # Aplicar SMOTETomek  
smotetomek = SMOTETomek(random_state=42)  
X_train_SMOTE_T, y_train_SMOTE_T = smotetomek.fit_resample(X_train, y_train)  
  
| #Comprobemos que las clases estan balabceadas  
values_counts_SMOTE_T = np.asarray(np.unique(y_train_SMOTE_T, return_counts=True))  
values_counts_SMOTE_T  
  
| array([[ 0,  1],  
|        [5352, 5352]])  
  
| # Entrenar un Random Forest en los datos con SMOTETOMEK  
RF_SMOTE_T = RandomForestClassifier()  
hiperparametros={'max_depth':[15,20],'min_samples_leaf':[40,30], 'max_leaf_nodes':[40,30], 'n_estimators':[100,90]}  
grilla=GridSearchCV(RF_SMOTE_T, hiperparametros, cv=5, scoring='accuracy', verbose=3)  
grilla.fit(X_train_SMOTE_T, y_train_SMOTE_T)
```

## Métricas:

	precision	recall	f1-score	support
0	0.90	0.84	0.87	619
1	0.70	0.79	0.74	283
accuracy			0.83	902
macro avg	0.80	0.82	0.80	902
weighted avg	0.83	0.83	0.83	902

## INSIGHT FINAL:

Comparamos todas nuestras métricas de cada técnica y en la última aplicada, SMOTE TOMEK, vemos cómo empeoran las métricas respecto de Random Oversampling, sobre todo en la clase de minoría, por lo cual, vamos a considera la técnica de "Random Oversampling", como la mejor técnica de balanceo para nuestra base en particular.

## SUMMARY FINAL COMPARATIVO:

Modelo	Accuracy	Precisión	Recall	F1 score	Area Curva ROC
Regresión Logística	72.8%	59.9%	35.0%	44.6%	68.3%
Regresión Logística (CON PCA)	82.8%	74.9%	67.6%	71.1%	80.4%
Árbol de Decisión (CON PCA)	83.5%	76.7%	72.3%	74.0%	82.0%
Random Forest con Grid Search	82.4%	75.7%	63.2%	69.3%	80.2%
Random Forest con Grid Search + RandomOverSampler	83.0%	70.0%	83.0%	76.0%	

## MCA & Random Forest

La MCA (Análisis de Correspondencias Múltiples) es una técnica estadística utilizada principalmente para analizar la asociación entre dos o más conjuntos de variables categóricas.

En lugar de aplicar PCA, analizar las variables continuas, vamos a aplicar MCA, para analizar variables categóricas.

```
#dataset con variables categoricas con OHE
from prince import MCA
mca = MCA(n_components=7)
mca.fit(data_encoded)
mca_results = mca.transform(data_encoded)
mca_results
```

component	eigenvalue	% of variance	% of variance (cumulative)
0	0.202	20.24%	20.24%
1	0.113	11.30%	31.55%
2	0.099	9.92%	41.47%
3	0.066	6.62%	48.09%
4	0.062	6.23%	54.32%
5	0.054	5.45%	59.77%
6	0.053	5.27%	65.04%

Con 7 variables logramos explicar el 65%

	0	1	2	3	4	5	6
<b>AGE_26-39_0</b>	2%	3%	6%	0%	0%	0%	0%
<b>AGE_26-39_1</b>	4%	6%	13%	1%	1%	0%	0%
<b>AGE_40-64_0</b>	1%	9%	1%	0%	0%	0%	0%
<b>AGE_40-64_1</b>	2%	21%	1%	0%	0%	0%	0%
<b>AGE_65+_0</b>	2%	2%	2%	0%	0%	0%	0%
<b>AGE_65+_1</b>	7%	10%	10%	1%	1%	0%	0%
<b>GENDER_male_0</b>	0%	0%	0%	7%	0%	19%	5%
<b>GENDER_male_1</b>	0%	0%	0%	7%	0%	19%	5%
<b>RACE_minority_0</b>	0%	0%	0%	1%	0%	1%	1%
<b>RACE_minority_1</b>	0%	0%	0%	9%	0%	10%	6%

Generamos nuestras variables en base al MCA calculado:

```
X_MCA = final_df.drop(['OUTCOME'], axis=1)
y_MCA= final_df['OUTCOME']
```

Volvemos a construir nuestro modelo de Random Forest con Grid Search y balanceo de RANDOM OVERSAMPLING:

```
MODELO RANDOM FOREST CON BALANCEO

[ ] X_train, X_test, y_train, y_test = train_test_split(X_MCA, y_MCA, test_size=0.1, random_state=42, stratify=y)

RANDOM OVERSAMPLING

[ ] # Aplicar Random Oversampling
ros = RandomOverSampler(random_state=42)
X_train_resampled_MCA, y_train_resampled_MCA = ros.fit_resample(X_train, y_train)

[ ] #Comprobemos que las clases estan balabceadas
values_counts_ros = np.asarray(np.unique(y_train_resampled_MCA, return_counts=True))
values_counts_ros

array([[ 0,  1],
       [5570, 5570]])

[ ] # Entrenar un Random Forest en los datos con RANDOMOVERSAMPLER
RF_ros_MCA = RandomForestClassifier()
hiperparametros={'max_depth':[15,20], 'min_samples_leaf':[40,30], 'max_leaf_nodes':[40,30], 'n_estimators':[100,90]}
grilla=GridSearchCV(RF_ros_MCA, hiperparametros, cv=5, scoring='accuracy', verbose=3)
grilla.fit(X_train_resampled_MCA, y_train_resampled_MCA)
```

## Métricas finales con MCA:

	precision	recall	f1-score	support
0	0.92	0.83	0.87	619
1	0.69	0.83	0.76	283
accuracy			0.83	902
macro avg	0.80	0.83	0.81	902
weighted avg	0.85	0.83	0.83	902

---

## INSIGHT

En nuestro último modelo de Random forest con Random Oversampling con PCA, tenemos un igual recall (0.83) respecto de MCA (0.83) y un menor f1 score en PCA (0.75) vs MCA (0.76) cuando ocurre un siniestro (OUTCOME =1), entonces concluimos que nuestro modelo mejora con MCA

## SUMMARY FINAL COMPARATIVO

Modelo	Accuracy	Precisión	Recall	F1 score	Área Curva ROC
Regresión Logística	72.8%	59.9%	35.0%	44.6%	68.3%
Regresión Logística (CON PCA)	82.8%	74.9%	67.6%	71.1%	80.4%
Árbol de Decisión (CON PCA)	83.5%	76.7%	72.3%	74.0%	82.0%
Random Forest con Grid Search	82.4%	75.7%	63.2%	69.3%	80.2%
Random Forest con Grid Search + RandomOverSampler	83.0%	70.0%	83.0%	76.0%	
Random Forest con Grid Search + RandomOverSampler + MCA	83.0%	69.0%	83.0%	76.0%	

## Cross Validation: K fold

Vamos a entrenar nuestro modelo de RandomForest con el objetivo de evaluar el rendimiento de un modelo y reducir el riesgo de sobreajuste.

### Stratified K - fold (con 5 divisiones)

Usaremos Stratified K-Fold, el cual es una variante de la validación cruzada K-Fold que mantiene la proporción de las clases en cada fold. Se asegura de que cada fold tenga una distribución de clases similar a la del conjunto de datos completo.

```

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold, KFold
from sklearn.metrics import accuracy_score

# Definir los parámetros que deseas probar en el GridSearch
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [5, 10]
}

# Crear el clasificador RandomForest
clf = RandomForestClassifier()

# Crear el objeto StratifiedKFold con 5 divisiones (k=5)
S_kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Crear el objeto GridSearchCV
grid_search = GridSearchCV(clf, param_grid, cv=S_kfold, scoring='accuracy', return_train_score=True, verbose=3)

# Ajustar el GridSearch en los datos de entrenamiento
grid_search.fit(X_train, y_train)

# Obtener los mejores parámetros y el mejor estimador del GridSearch
best_params = grid_search.best_params_

print("Mejores hiperparámetros:", best_params)

```

```

# Realizar predicciones en todo el conjunto de datos utilizando el mejor estimador
best_estimator = grid_search.best_estimator_
y_pred = best_estimator.predict(X_train)

# Calcular la precisión
accuracy = accuracy_score(y_train, y_pred)
accuracy

```

0.8645884672252341

## INSIGHT:

Luego de las iteraciones, lo que nos está diciendo en conclusión la técnica de Stratified K-fold que los mejores hiperparámetros son {'max\_depth': 10, 'n\_estimators': 100} mi profundidad es 10 y cantidad de árboles son 100. Con estos mejores estimadores hacemos la predicción el modelo de RandomForest, es decir re-entrenamos el modelo con dichos hiperparámetros.

Procedemos a calcular el accuracy= 86%

	mean_train_score	mean_test_score	params
2	0.872	0.831	{'max_depth': 10, 'n_estimators': 100}
3	0.872	0.830	{'max_depth': 10, 'n_estimators': 200}
1	0.811	0.807	{'max_depth': 5, 'n_estimators': 200}
0	0.809	0.805	{'max_depth': 5, 'n_estimators': 100}

De los 4 splits y calcula la media de cada uno, se puede ver en cada uno las mejores combinaciones de cada uno. De este modo, se puede ver que la técnica de cross validation busca siempre que la varianza sea bastante baja y el bias también. Siendo el objetivo principal no tengamos overfitting, es con una profundidad de 10 y 100 árboles

### K - fold (con 5 divisiones)

Usaremos K-Fold, el cual consiste en dividir el conjunto de datos en K partes (folds) y realizar K iteraciones del proceso de entrenamiento y evaluación. En cada iteración, un fold se utiliza como conjunto de prueba, mientras que los K-1 folds restantes se utilizan para entrenar el modelo. Este proceso se repite K veces, cada vez utilizando un fold diferente como conjunto de prueba

```
# Crear el objeto KFold con 5 divisiones (k=5)
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# Crear el objeto GridSearchCV con KFold
grid_search = GridSearchCV(clf, param_grid, refit=True, scoring='accuracy', return_train_score=True)

# Ajustar el GridSearch en los datos de entrenamiento
grid_search.fit(X_train, y_train)

# Obtener los mejores parámetros y el mejor estimador del GridSearch
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

# Imprimir los mejores parámetros encontrados
print("Mejores parámetros:", best_params)

Mejores parámetros: {'max_depth': 10, 'n_estimators': 200}
```

K fold: a diferencia de stratified K fold la distribución de las clases, no nos aseguramos que las clases están distribuidas proporcionalmente. Vemos en esta técnica la conclusión es distinta: los mejores hiperparámetros son {'max\_depth': 10, 'n\_estimators': 200} mi profundidad es 10 y cantidad de árboles son 200 (antes 100).

---

	<code>mean_train_score</code>	<code>mean_test_score</code>	<code>params</code>
3	0.871	0.828	{'max_depth': 10, 'n_estimators': 200}
2	0.871	0.826	{'max_depth': 10, 'n_estimators': 100}
0	0.809	0.804	{'max_depth': 5, 'n_estimators': 100}
1	0.811	0.804	{'max_depth': 5, 'n_estimators': 200}

## INSIGHT FINAL

### K fold vs Stratified K fold

Primero si comparamos K fold vs Stratified K fold, respecto del accuracy da mayor K fold que S\_Kfold, 86.6% vs 86.4%, nosotros estamos prediciendo que ocurra o no un siniestro, lo cual la precisión es importante para ver qué precio cobrar al asegurado, pero en cuanto a cantidad podemos decir que ese 0.2% podría no llegar a ser tan representativo, pero optamos por K fold, suponemos que la base no está tan desequilibrada sino daría mucho peor k fold que stratified k fold.

---

## Conclusiones Finales

Si comparamos el 86.6% de accuracy del re entrenamiento con K fold vs el último modelo que buscamos optimizar hiperparametros y con MCA, donde la accuracy es 83%. Hay mejoras en la performance del modelo (tanto con Stratified K fold y K fold), ya que con estas técnicas de cross validation, se obtiene una evaluación más robusta del rendimiento del modelo al considerar múltiples particiones del conjunto de datos. Esto puede ayudar a tener una mejor estimación del rendimiento del modelo en datos no vistos y reducir el riesgo de sobreajuste, que es lo que principalmente estamos buscando.

**Por lo cual, nuestra decisión final como modelo a utilizar, en nuestro sector de Data Science, es un Modelo Random Forest considerando la búsqueda de los mejores hiperparametros y con la técnica de Cross Validation (K fold). Ya que no solo obtenemos mejoras en la performance del modelo a nivel métricas sino también un equilibrio entre bias y varianza (ni un high variance y low bias o low variance y high bias), buscar el trade-off sesgo-varianza.**