Análisis cuantitativo avanzado

Clase 3: Regresión Logística

Lic. Lucio José Pantazis

Motivación

Cobertura de salud

La siguiente base “acs_health.Rda” describe datos sobre cobertura de salud en una muestra de estadounidenses residentes en la ciudad de Georgia (disponible en https://github.com/DataScienceForPublicPolicy/diys/tree/main/data).

Carga de datos

Para cargar los datos, por el formato ".Rda", es difícil descargarlos directamente desde el link url. Por lo tanto, se puede bajar al disco o cargar el archivo al drive, para que luego el Python pueda interpretarlo.

Además, se requiere un paquete especial para poder leer este tipo de archivos, llamado `pyreadr`.

Una vez instalado el paquete, se puede leer la base de datos guardada en el drive. Para acceder al drive, debo montar el contenido.

```
import pyreadr
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
url="https://github.com/DataScienceForPublicPolicy/diys/tree/main/data"
result=pyreadr.read_r(url)
```

El formato ".Rda" no carga directamente una base de datos, sino que lo guarda en una lista donde está codificada la base con un nombre (denominado "key"). Por lo tanto, primero conviene inspeccionar el nombre para saber cómo se denomina la base:

```
print(result.keys())
```

```
## odict_keys(['health'])
```

Como vemos que la base se denomina "health", la cargamos extrayendo de los resultados la variable con ese nombre.

```
df = result["health"]
```

Carga de la base e inspección

Ya cargada la base, podemos empezar a inspeccionar sus variables y evaluar su estructura.

```
df.head()
```

```
##    id  no.coverage  age  ...                 schl                  esr  p
## 0   1            0   60  ...          HS Degree   Employed Civilian  2
## 1   2            0   44  ...  Undergraduate Degree   Employed Civilian  1
## 2   3            0   40  ...  Undergraduate Degree   Employed Civilian  1
## 3   4            0   76  ...       Graduate Degree  Not in Labor Force  2
## 4   5            0   71  ...          HS Degree  Not in Labor Force  2
##
## [5 rows x 11 columns]
```

Inspección de las variables

Además, veamos qué tipo de datos tienen las variables.

```
df.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 74805 entries, 0 to 74804
## Data columns (total 11 columns):
##  #   Column       Non-Null Count  Dtype
## ---  ------       --------------  -----
##  0   id           74805 non-null  int64
##  1   no.coverage  74805 non-null  int64
##  2   age          74805 non-null  int64
##  3   race         74805 non-null  object
##  4   sex          74805 non-null  object
##  5   wage         74805 non-null  float64
##  6   cit          74805 non-null  object
##  7   mar          74805 non-null  object
##  8   schl         74805 non-null  object
##  9   esr          74805 non-null  object
##  10  puma         74805 non-null  int64
## dtypes: float64(1), int64(4), object(6)
## memory usage: 6.3+ MB
```

- Vemos que la base tiene 74805 registros.
- Entre las variables se encuentran varias ya denominadas como categóricas.

Inspección de las variables

- `id`: Variable identificatoria del individuo (variable cualitativa)
- `no.coverage`: Indica si el individuo tiene cobertura (vale 0) o no tiene cobertura (vale 1) (variable cualitativa)
- `age`: Edad del paciente en años (variable cuantitativa)
- `race`: Categorización del individuo según rasgos étnicos (variable cualitativa)
- `sex`: Sexo biológico del individuo (variable cualitativa)
- `wage`: Ingreso en dólares del individuo (variable cuantitativa)
- `cit`: Variable que indica si el individuo tiene ciudadanía norteamericana (variable cualitativa)
- `mar`: Estado civil del individuo (variable cualitativa)
- `schl`: Nivel de estudios alcanzados (variable cualitativa ordinal)
- `esr`: Estado laboral del individuo (variable cualitativa)

Variable de respuesta

A partir de esta base, podemos tratar de identificar qué factores influyen sobre tener o no seguro médico. En la base, esta variable se identifica con "no.coverage", que está establecida como númerica (`float64`) pero en realidad vemos que es una variable dummy, que sólo toma valores 0 y 1 para identificar si un individuo tiene cobertura médica o no. En este caso, se identifica con un 0 si tiene cobertura y con un 1 si no la tiene.

```
print(df["no.coverage"].value_counts())
```

```
## 0    61114
## 1    13691
## Name: no.coverage, dtype: int64
```

Vemos que si bien la mayoría de los individuos de la muestra tienen seguro médico, considerando las dificultades socioeconómicas que introduce en la vida cotidiana, sigue siendo un porcentaje alto el de las personas sin seguro médico (cerca del 18%):

```
print(df["no.coverage"].value_counts()/df.shape[0]*100)
```

```
## 0    81.697747
## 1    18.302253
## Name: no.coverage, dtype: float64
```

Redefinición de la variable

Para simplificar un poco algunos análisis, podemos definir una variable "cobertura" que en vez de ceros y unos, informe efectivamente si tiene cobertura o no:

```
df["cobertura"]=[0]*df.shape[0]
df.loc[df["no.coverage"]==0,"cobertura"]="Con Cobertura(%)"
df.loc[df["no.coverage"]==1,"cobertura"]="Sin Cobertura(%)"
print(df["cobertura"].value_counts()/df.shape[0]*100)
```

```
## Con Cobertura(%)    81.697747
## Sin Cobertura(%)    18.302253
## Name: cobertura, dtype: float64
```

Análisis exploratorio de datos

Análisis exploratorio de datos

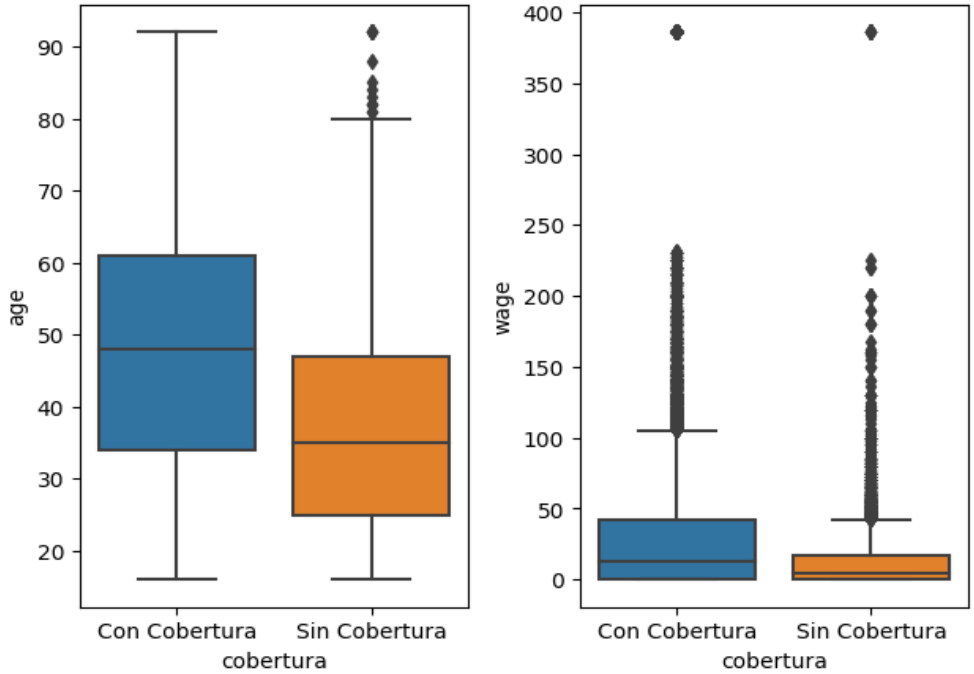
Siempre conviene antes de elegir las variables que consideramos de influencia, ver si hay evidencia que sustente la sospecha. Por lo tanto, vamos a realizar algunos gráficos exploratorios, para lo cual debemos considerar que la variable de respuesta (tener seguro médico o no) es **cualitativa**.

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Variables cuantitativas (boxplots)

Primero, veremos cómo se vinculan con la cobertura médica las siguientes variables *cuantitativas*: `"age"` y `"wage"`. La idea es comparar cómo se distribuyen sus valores entre los que tienen seguro médico y no lo tienen. Por lo tanto, podemos comparar sus boxplots

```
fig,axes=plt.subplots(1,2)
sns.boxplot(data=df,x="cobertura",y="age",ax=axes[0])
sns.boxplot(data=df,x="cobertura",y="wage",ax=axes[1])
plt.show()
```



En ambos casos, se puede ver que si bien los valores mínimos y máximos coinciden entre ambas clases, la distribución es distinta, dando nociones de alguna influencia de ambas variables sobre la variable de cobertura.

Análisis de boxplots

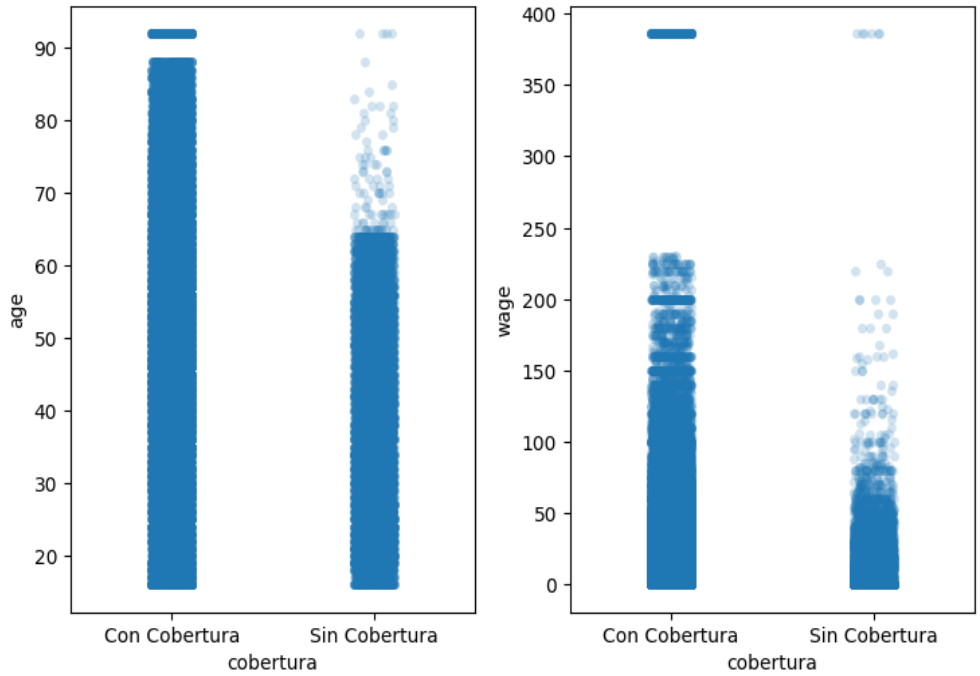
Ambas variables tienden a tener menores valores entre las personas que no tienen cobertura, ya que las medianas y los terceros cuartiles son menores en ambos casos.

- Lo observado tiene cierta lógica para el caso de la variable `"wage"`, ya que las personas que tengan menor salario, es menos probable que puedan acceder a un seguro médico.
- Sin embargo, la intuición nos falla para la variable `"age"`, ya que se podría pensar que a mayor edad, es más costoso conseguir seguro médico. De todas formas, se puede pensar también que el seguro médico se vuelve más necesario a medida que aumenta la edad y es un costo que se afronta a pesar de las dificultades.

Variables cuantitativas (stripplots)

Otra forma de visualizar estas influencias es con un stripplot, que grafica puntos con una cierta dispersión fija que permite ver dónde hay mayor densidad de puntos.

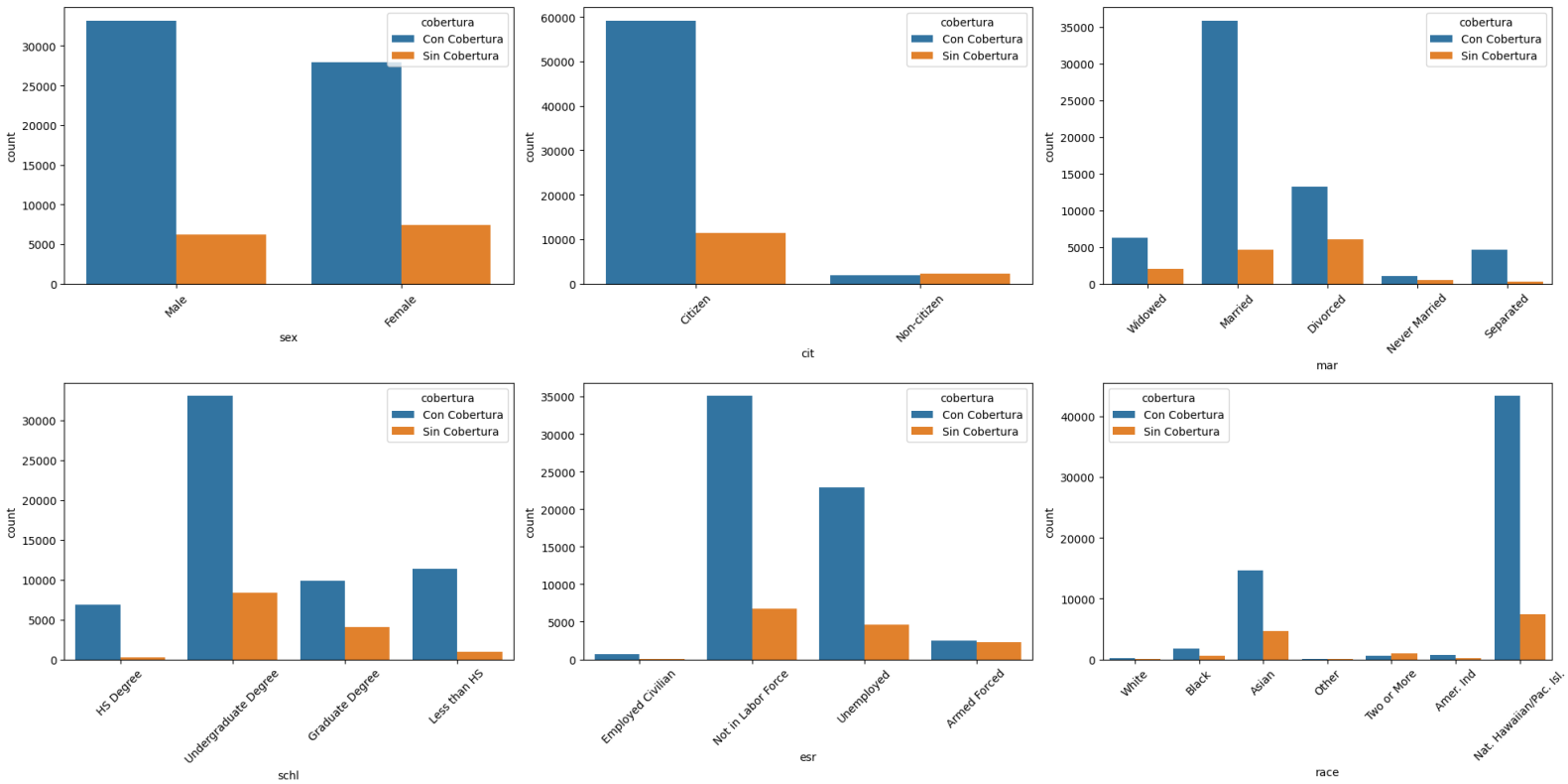
```
fig,axes=plt.subplots(1,2)
# Para enfatizar las mayores densidades, se agrega un valor de opacidad (alpha)
sns.stripplot(data=df,x="cobertura",y="age",ax=axes[0],alpha=0.2)
sns.stripplot(data=df,x="cobertura",y="wage",ax=axes[1],alpha=0.2)
plt.show()
```



Variables cualitativas (countplot)

Ahora vemos cómo se vinculan entre con la cobertura médica las siguientes variables *cualitativas*: `"sex"`, `"cit"`, `"mar"`, `"schl"`, `"esr"` y `"race"`. Podemos comparar los resultados en cada clase utilizando un gráfico de barras o `countplot`:

```
fig,axes=plt.subplots(2,3)
sns.countplot(data=df,x="sex",hue="cobertura",ax=axes[0,0])
sns.countplot(data=df,x="cit",hue="cobertura",ax=axes[0,1])
sns.countplot(data=df,x="mar",hue="cobertura",ax=axes[0,2])
sns.countplot(data=df,x="schl",hue="cobertura",ax=axes[1,0])
sns.countplot(data=df,x="esr",hue="cobertura",ax=axes[1,1])
sns.countplot(data=df,x="race",hue="cobertura",ax=axes[1,2])
plt.show()
```



Análisis countplots

En general podemos ver que estos gráficos no siempre son muy informativos, sobre todo cuando hay muchas clases que tienen cantidades muy dispares de individuos, ya que no todas las barras se aprecian a la misma escala.

Más aún, tampoco se llega a visualizar lo importante, si la relación en cada clase entre ambas barras es la misma para las personas que tienen cobertura y las personas que no la tienen.

Variables cualitativas (Gráfico de mosaico)

Para evitar estos problemas de visualización, se puede hacer un gráfico de mosaico, que permite comparar lo importante, las **proporciones** en ambas clases.

Para realizar varios gráficos de mosaico, importo la función y agregaremos una función auxiliar definida por nosotros para que los gráficos tengan toda la información pertinente.

Los gráficos de mosaico no se pueden disponer en paneles como los gráficos anteriores de barras, por lo tanto, debemos analizar el vínculo con todas las variables por separado.

Gráfico de mosaico (Variable sex)

Ahora procedemos a ver el gráfico de mosaico de la variable "sex" para ver si hay una diferencia por sexo en la cobertura médica:

```
v1="sex";v2="cobertura"
lis=ps_mosaic(df,v1,v2)
props=lis[0];labs=lis[1]
labelizer=lambda k:labs[k]
g=mosaic(df,[v1,v2],labelizer=labelizer,properties=props,label_rotation=[90,0])
plt.show()
```

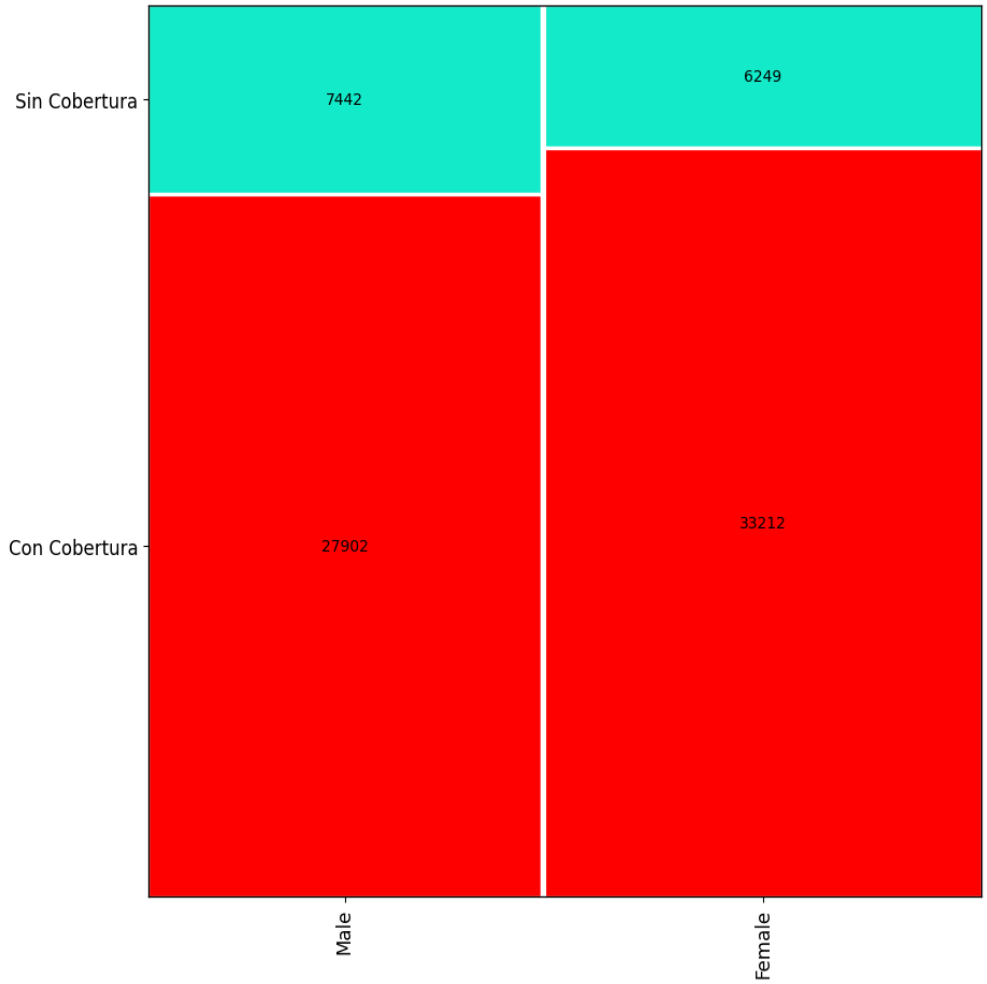


Gráfico de mosaico (Variable cit)

```
v1="cit";v2="cobertura"
lis=ps_mosaic(df,v1,v2)
props=lis[0];labs=lis[1]
labelizer=lambda k:labs[k]
g=mosaic(df,[v1,v2],labelizer=labelizer,properties=props,label_rotation=[90,0])
plt.show()
```

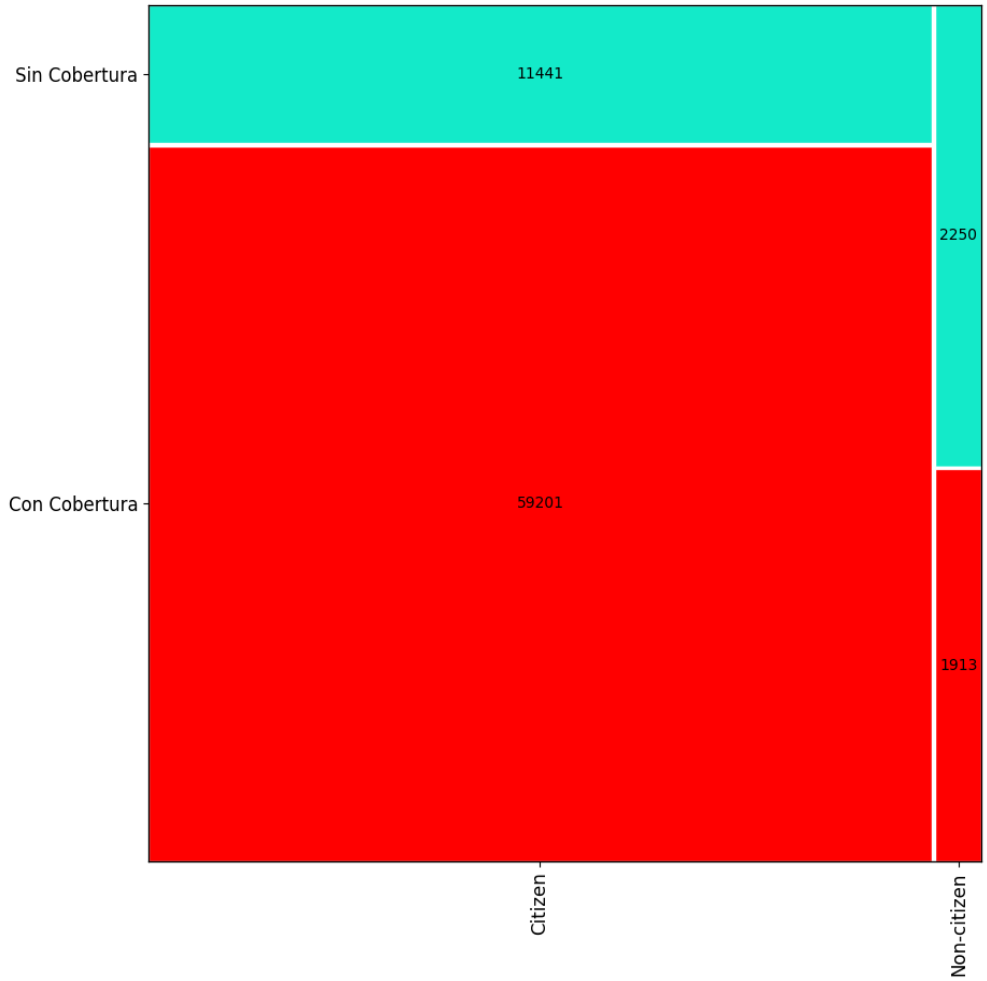


Gráfico de mosaico (Variable mar)

```
v1="mar";v2="cobertura"
lis=ps_mosaic(df,v1,v2)
props=lis[0];labs=lis[1]
labelizer=lambda k:labs[k]
g=mosaic(df,[v1,v2],labelizer=labelizer,properties=props,label_rotation=[90,0])
plt.show()
```

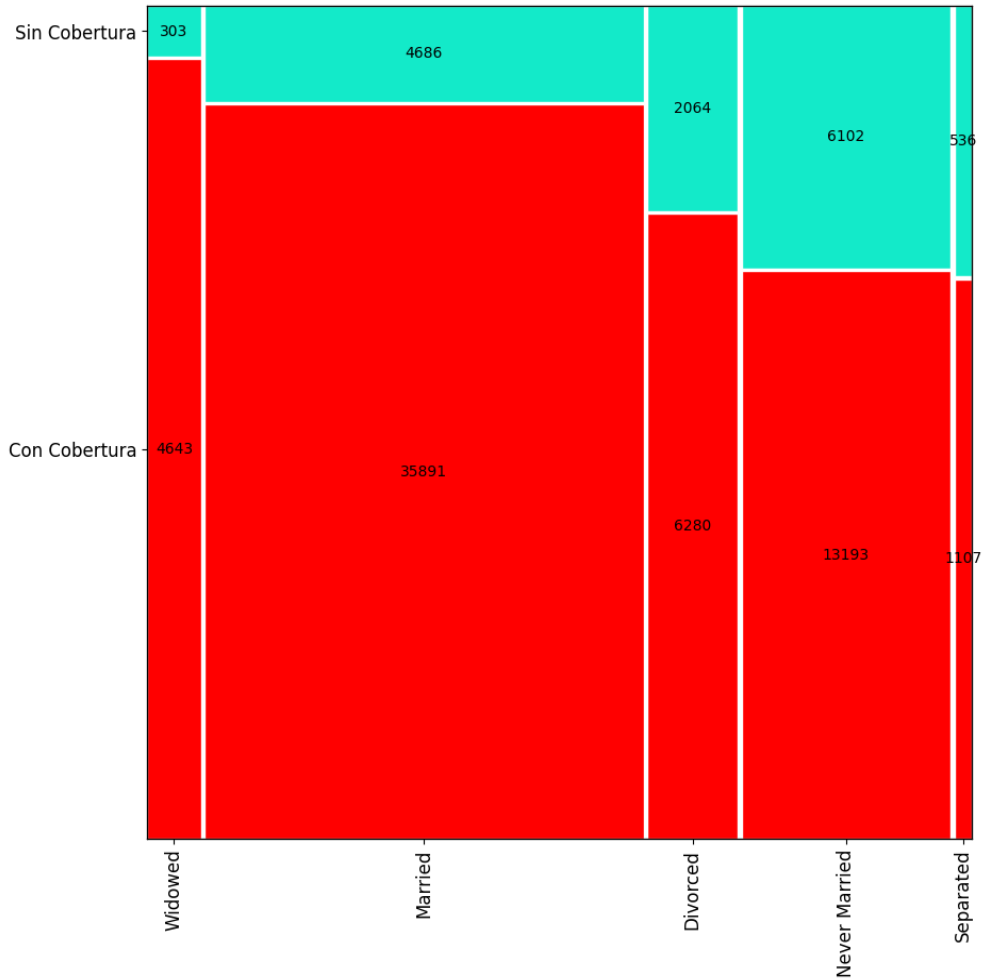


Gráfico de mosaico (Variable `schl`)

```
v1="schl";v2="cobertura"
lis=ps_mosaic(df,v1,v2)
props=lis[0];labs=lis[1]
labelizer=lambda k:labs[k]
g=mosaic(df,[v1,v2],labelizer=labelizer,properties=props,label_rotation=[90,0])
plt.show()
```

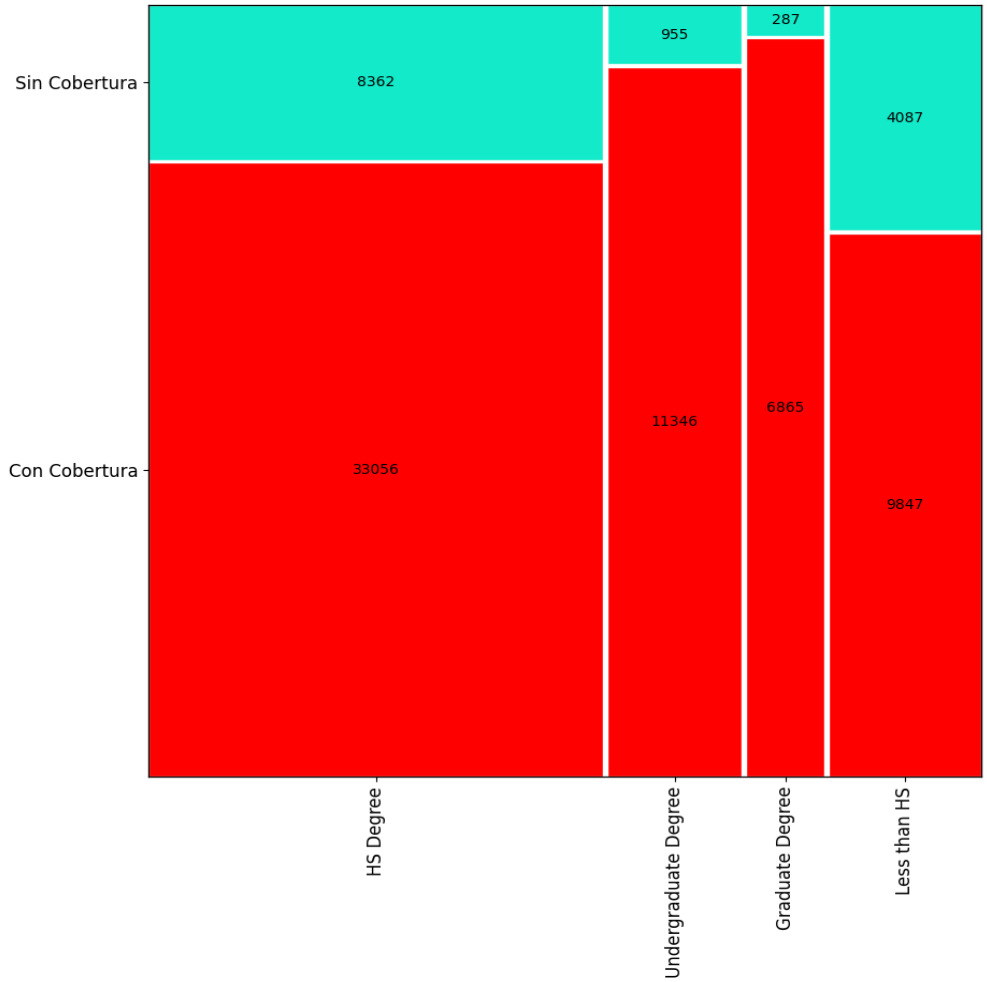


Gráfico de mosaico (Variable esr)

```
v1="esr";v2="cobertura"
lis=ps_mosaic(df,v1,v2)
props=lis[0];labs=lis[1]
labelizer=lambda k:labs[k]
g=mosaic(df,[v1,v2],labelizer=labelizer,properties=props,label_rotation=[90,0])
plt.show()
```

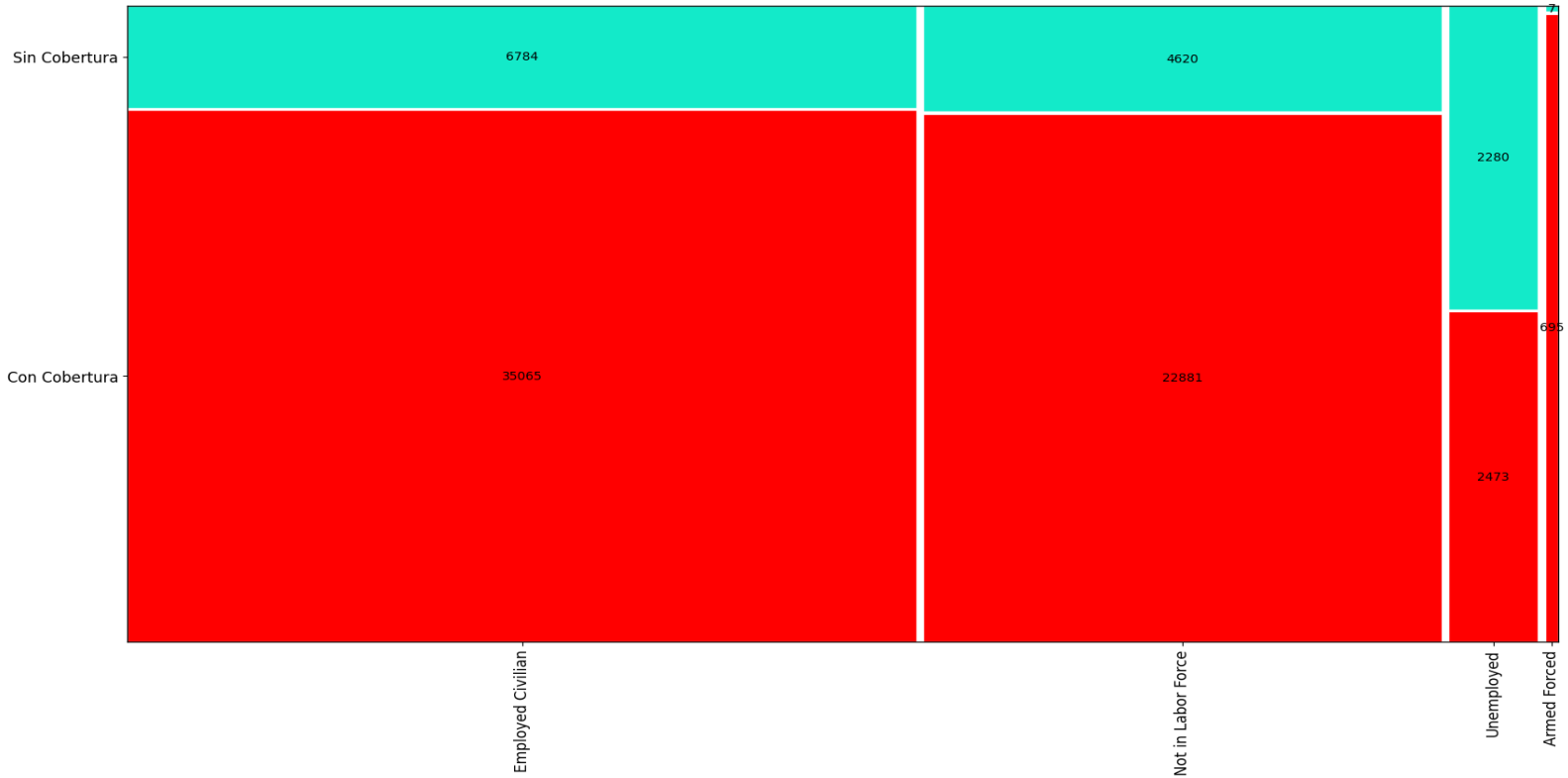
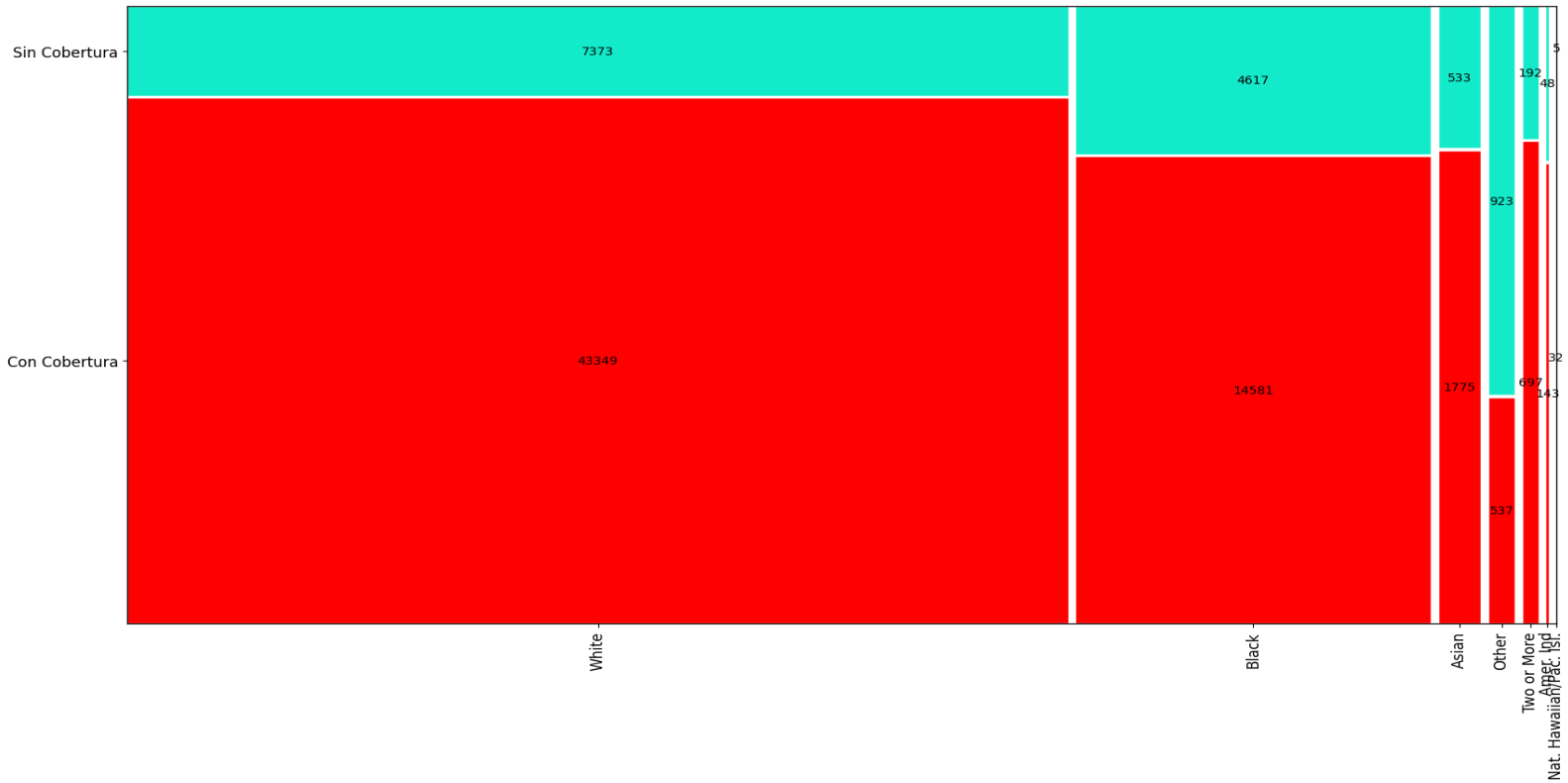


Gráfico de mosaico (Variable race)

```
v1="race";v2="cobertura"
lis=ps_mosaic(df,v1,v2)
props=lis[0];labs=lis[1]
labelizer=lambda k:labs[k]
g=mosaic(df,[v1,v2],labelizer=labelizer,properties=props,label_rotation=[90,0])
plt.show()
```



Análisis Gráficos de Mosaico

Por lo tanto, todas las variables parecen influir sobre la probabilidad de tener cobertura o no, ya que en todos los casos, las proporciones son diferentes por cada clase. De todas formas, esto es un primer indicio, hay que darle sustento desde otro enfoque.

Regresión logística

Regresión logística

Ya vimos en las clases anteriores que los vínculos entre variables se evalúan mediante **modelos de regresión**. Según lo visto, plantear un modelo de regresión lineal para establecer un vínculo entre la variable `"cobertura"` con las variables `"age"`, `"wage"`, `"sex"`, `"cit"`, `"mar"`, `"schl"`, `"esr"` y `"race"` sería plantear el siguiente modelo:

$$\text{cobertura} \approx \beta_0 + \beta_1 \cdot \text{age} + \beta_2 \cdot \text{wage} + \beta_3 \cdot \text{sex} + \beta_4 \cdot \text{cit} + \beta_5 \cdot \text{schl} + \beta_6 \cdot \text{esr} + \beta_7 \cdot \text{race}$$

Regresión logística

Sin embargo, surge un problema ante esta formulación. Más allá de que hay muchas variables predictoras que son categóricas y deben utilizarse las variables "dummy", la variable `"no.coverage"` toma sólo los valores 0 y 1.

Por lo tanto, los valores de las constantes β_0, β_1, β_2, β_3, β_4, β_5, β_6, β_7 no son completamente libres, ya que deberían compensarse para que el resultado siempre sea **exactamente** 0 o 1. Esto no tiene mucho sentido, por lo que se necesita una variable de respuesta que tome valores en un continuo.

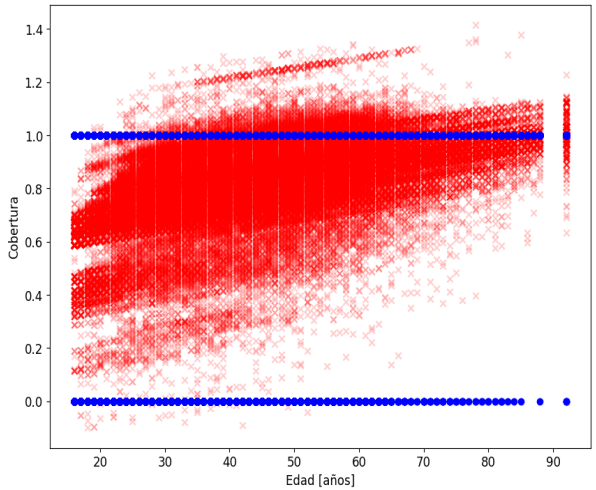
Regresión lineal

Supongamos que queremos realizar una regresión lineal con estas variables (utilizando las variables dummies) y realizamos predicciones para los mismos datos:

```
from sklearn.linear_model import LinearRegression
mod=LinearRegression()
mod.fit(X,y)
y_pred=mod.predict(X)
```

Para poder evaluar cómo son las predicciones, podemos graficar las mismas junto conlas respuestas reales respecto de alguna variable (por ejemplo, la variable de edad):

```
plt.scatter(X["age"],y_pred,c="r",marker="x",alpha=0.2)
plt.scatter(X["age"],y,c="b")
plt.xlabel("Edad [años]");plt.ylabel("Cobertura")
plt.show()
```



Aquí vemos gráficamente que el modelo es erróneo, porque no representa gráficamente a los datos, justamente porque las respuestas deben ser únicamente 1 o 0.

Nociones de probabilidad

Supongamos que vamos a tirar una moneda. Intuitivamente, tenemos la idea de que la **probabilidad** de que el resultado sea cara es del 50%. Pero vamos a detenernos a pensar un minuto en qué representa ese valor, para luego poder interpretar el concepto de probabilidad en otros casos.

Una forma de pensar la probabilidad, es pensar en cuán frecuente es algún resultado de un experimento aleatorio. En este caso,

- el experimento es tirar una moneda
- es aleatorio porque no podemos determinar su resultado antes del experimento
- se quiere determinar con qué frecuencia la moneda sale cara

Nociones de probabilidad

El problema es que para determinar cuán frecuente es algo, necesitamos que el experimento se repita varias veces, y nosotros sólo tiramos la moneda una vez. Pero justamente, podemos interpretar el valor de 50% del siguiente modo: si repitiéramos un gran número de veces el experimento (tirar varias veces una moneda), aproximadamente el 50% de esos resultados serían una cara, ya que la moneda no tiene ninguna preferencia por salir de alguno de los dos lados.

```
print(dfEj.iloc[0:24])
```

```
##     Tirada Resultado  Caras acumuladas  Porcentaje Caras
## 0        1      ceca                 0          0.000000
## 1        2      ceca                 0          0.000000
## 2        3      cara                 1         33.333333
## 3        4      ceca                 1         25.000000
## 4        5      cara                 2         40.000000
## 5        6      ceca                 2         33.333333
## 6        7      ceca                 2         28.571429
## 7        8      cara                 3         37.500000
## 8        9      ceca                 3         33.333333
## 9       10      ceca                 3         30.000000
## 10      11      cara                 4         36.363636
## 11      12      ceca                 4         33.333333
## 12      13      ceca                 4         30.769231
## 13      14      cara                 5         35.714286
## 14      15      cara                 6         40.000000
## 15      16      ceca                 6         37.500000
## 16      17      cara                 7         41.176471
## 17      18      cara                 8         44.444444
## 18      19      cara                 9         47.368421
## 19      20      cara                10         50.000000
## 20      21      cara                11         52.380952
## 21      22      ceca                11         50.000000
## 22      23      cara                12         52.173913
## 23      24      cara                13         54.166667
```

Nociones de probabilidad

El problema es que para determinar cuán frecuente es algo, necesitamos que el experimento se repita varias veces, y nosotros sólo tiramos la moneda una vez. Pero justamente, podemos interpretar el valor de 50% del siguiente modo: si repitiéramos un gran número de veces el experimento (tirar varias veces una moneda), aproximadamente el 50% de esos resultados serían una cara, ya que la moneda no tiene ninguna preferencia por salir de alguno de los dos lados.

```
print(dfEj.iloc[25:N])
```

```
##     Tirada Resultado  Caras acumuladas  Porcentaje Caras
## 25      26      ceca                13         50.000000
## 26      27      cara                14         51.851852
## 27      28      cara                15         53.571429
## 28      29      ceca                15         51.724138
## 29      30      cara                16         53.333333
## 30      31      cara                17         54.838710
## 31      32      ceca                17         53.125000
## 32      33      cara                18         54.545455
## 33      34      cara                19         55.882353
## 34      35      cara                20         57.142857
## 35      36      ceca                20         55.555556
## 36      37      ceca                20         54.054054
## 37      38      cara                21         55.263158
## 38      39      ceca                21         53.846154
## 39      40      ceca                21         52.500000
## 40      41      ceca                21         51.219512
## 41      42      ceca                21         50.000000
## 42      43      ceca                21         48.837209
## 43      44      cara                22         50.000000
## 44      45      ceca                22         48.888889
## 45      46      cara                23         50.000000
## 46      47      cara                24         51.063830
## 47      48      cara                25         52.083333
## 48      49      cara                26         53.061224
```

Probabilidad como respuesta

Por lo tanto, vamos a pensar en la **probabilidad** de que una persona tenga cobertura médica. Es decir, si tuviéramos un gran número de personas **con las mismas características**, ¿qué proporción de ellas tendría cobertura médica?

Por lo tanto, vamos a adaptar el modelo anterior del siguiente modo:

$$P(\text{cobertura} = 1) \approx \beta_0 + \beta_1 \cdot \text{age} + \beta_2 \cdot \text{wage} + \beta_3 \cdot \text{sex} + \beta_4 \cdot \text{cit} + \beta_5 \cdot \text{schl} + \beta_6 \cdot \text{esr} + \beta_7 \cdot \text{race}$$

donde P representa la probabilidad de que la variable `"cobertura"` sea 1. La ventaja de esta propuesta, es que el valor de la probabilidad ya no tiene porqué valer exactamente 0 o 1, sino que puede tomar varios valores. Sin embargo, siguen siendo valores que tienen sentido **entre 0 y 1**. Por lo tanto, los valores de cada constante β siguen estando restringidos.

Odds

Para ampliar el rango de valores posibles, se puede comparar la probabilidad de que una persona tenga cobertura, respecto de la probabilidad de que no la tenga. Para eso se define el siguiente cociente denominado "odds":

$$\text{odds} = \frac{P(\text{cobertura} = 1)}{P(\text{cobertura} = 0)}$$

Considerando que la variable tiene sólo dos valores, en realidad, se puede expresar del siguiente modo:

$$P(\text{cobertura} = 0) = 1 - P(\text{cobertura} = 1)$$

Entonces, podemos reescribir el odds del siguiente modo:

$$\text{odds} = \frac{P(\text{cobertura} = 1)}{1 - P(\text{cobertura} = 1)}$$

- A partir de esta división, si el resultado es mayor que 1, significa que es más probable que una persona tenga cobertura, respecto de que no la tenga. Más aún, si el resultado del cociente es 8, significa que la probabilidad de que una persona tenga cobertura es 8 veces más grande que la probabilidad de que no la tenga.
- Del mismo modo, si el resultado es menor que 1, significa que es menos probable que una persona tenga cobertura, respecto de las personas sin seguro médico.
- El caso en el que el odds vale 1, la probabilidad de cobertura es de $\frac{1}{2} = 0.5$.

Odds como respuesta

Por lo tanto, podemos actualizar el modelo del siguiente modo:

$$\text{odds} = \frac{P(\text{cobertura} = 1)}{1 - P(\text{cobertura} = 1)} \approx \beta_0 + \beta_1 \cdot \text{age} + \beta_2 \cdot \text{wage} + \beta_3 \cdot \text{sex} + \beta_4 \cdot \text{cit} + \beta_5 \cdot \text{schl} + \beta_6 \cdot \text{esr} + \beta_7 \cdot \text{race}$$

Respecto a los valores que puede tomar el odds,

- si el valor de $P(\text{cobertura} = 1)$ es cercano a 0, el denominador es cercano a 1 y el resultado del odds es cercano a 0
- si el valor de $P(\text{cobertura} = 1)$ es cercano a 1, el denominador es cercano a 0 y el resultado del odds puede tomar valores muy altos, abarcando cualquier número positivo.

De todas formas, el resultado del odds siempre es **positivo**. Por lo tanto, sigue habiendo una restricción para las constantes β.

Log Odds como respuesta

Para resolver todo tipo de restricción, se puede tomar el logaritmo natural al cociente odds, ya que al aplicarlo a cualquier número positivo, el logaritmo puede tomar cualquier valor real, ya sea positivo o negativo.

Entonces, el modelo final que se plantea es el siguiente:

$$\log(\text{odds}) = \log\left(\frac{P(\text{cobertura} = 1)}{1 - P(\text{cobertura} = 1)}\right) \approx \beta_0 + \beta_1 \cdot \text{age} + \beta_2 \cdot \text{wage} + \beta_3 \cdot \text{sex} + \beta_4 \cdot \text{cit} + \beta_5 \cdot \text{schl} + \beta_6 \cdot \text{esr} + \beta_7 \cdot \text{race}$$

En este caso, como el logaritmo puede tomar cualquier valor real, las constantes β ya no tienen ningún tipo de restricción. Por lo tanto, buscamos que los coeficientes estimados se atengan a este modelo. La función $\log(\text{odds}(p))$ es a veces llamada “logit”.

Visualización numérica

El vínculo entre las probabilidades p, el odds y el log odds, lo podemos ver dándole valores a p:

```
ps=np.linspace(0.1,0.9,9)
odds=ps/(1-ps)
lOdds=np.log(odds)
df_odds=pd.DataFrame({"p":ps,"odds":odds,"lOdds":lOdds})
print(df_odds)
```

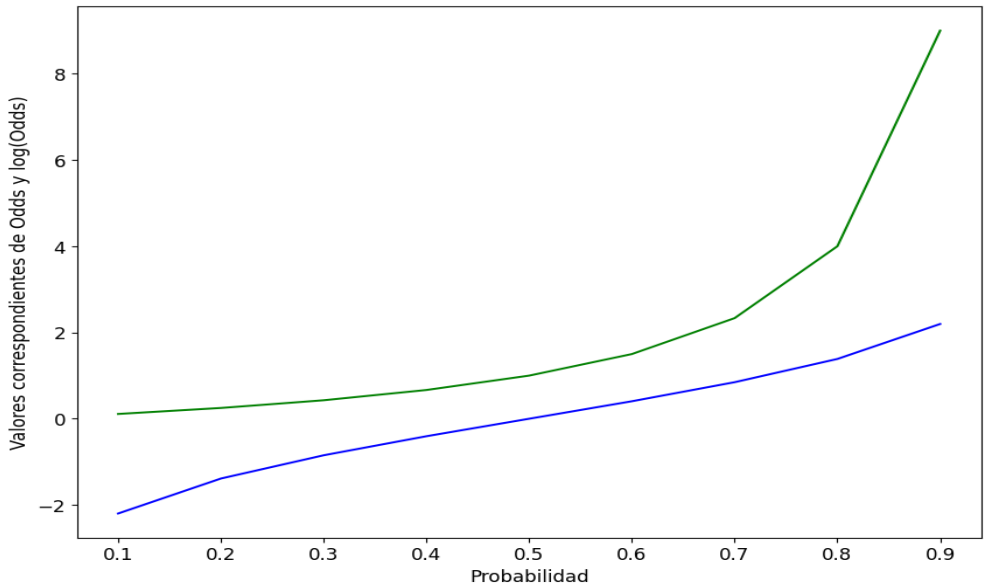
```
##      p      odds     lOdds
## 0  0.1  0.111111 -2.197225
## 1  0.2  0.250000 -1.386294
## 2  0.3  0.428571 -0.847298
## 3  0.4  0.666667 -0.405465
## 4  0.5  1.000000  0.000000
## 5  0.6  1.500000  0.405465
## 6  0.7  2.333333  0.847298
## 7  0.8  4.000000  1.386294
## 8  0.9  9.000000  2.197225
```

Vemos que a partir de probabilidades entre 0 y 1, se obtienen valores positivos para el odds, y de ambos signos para el log Odds.

Explicación gráfica

Gráficamente, graficando el odds con color verde y el log Odds con color azul:

```
plt.plot(ps,odds,c="g")
plt.plot(ps,lOdds,c="b")
plt.show()
```



Estimación

Ahora que ya tenemos el modelo, debemos encontrar los valores óptimos para los coeficientes β_0, β_1, β_2, β_3, β_4, β_5, β_6 y β_7. Estos valores deben buscarse con algún criterio razonable. En este caso, a diferencia de lo observado en regresión lineal, se apela a los "estimadores de máxima verosimilitud".

La idea es la siguiente: Se tiene como dato tanto la cobertura médica o no de cada individuo, y de las variables que influyen sobre dicha condición. A su vez, según el modelo, cada variable tiene una cierta influencia sobre la probabilidad de cobertura (a través de cada β).

¿Cuáles son los valores de β que, a partir de las variables explicativas, aumentan la probabilidad de tener cobertura médica entre los que la tienen y, al mismo tiempo, aumentan la probabilidad de no tener cobertura médica entre los que no la tienen?

Es decir, dados los datos de cobertura y las variables explicativas, ¿cuáles son los valores más verosímiles para cada β?

Estimación en Python

Para estimar los valores de β, en Python se utiliza la función `LogisticRegression` del paquete `sklearn`. Recordemos que ya fueron generadas las variables dummy. La variable de respuesta fue definida como `y`, mientras que las covariables fueron definidas como `X`.

Ahora que tenemos los datos correspondientes para la regresión logística, calculamos los valores de los β con la función `LogisticRegression`:

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(fit_intercept=False)
logreg.fit(X,y)

print(df_coef)
```

```
##         age      wage  ...  esr_Not in Labor Force  esr_Unemployed
##    0.030652  0.025321  ...                 0.18883       -1.099499
##
## [1 rows x 26 columns]
```

Interpretación de los coeficientes

Analizaremos el primer coeficiente, que corresponde a la variable de edad ($\beta_1 = 0.03065163$). La interpretación inicial es similar a la de la regresión múltiple, sólo que debe adaptarse al modelo elegido, cuya variable de respuesta es el log(odds).

Es decir, **manteniendo todas las otras variables constantes**, cada año extra que vive una persona *aumenta* (el signo es positivo) en 0.03065163 el valor estimado del log(odds) de tener cobertura médica.

Para independizarse del logaritmo, se aplica una función exponencial, utilizando el número irracional $e \approx 2.71$. Por lo tanto, **manteniendo todas las otras variables constantes**, cada año extra que vive una persona *aumenta* en $e^{0.03065163} \approx 1.0311$ el valor estimado del odds de tener cobertura médica, que se calcula del siguiente modo: $\frac{P(\text{cobertura} = 1)}{1 - P(\text{cobertura} = 1)}$.

Interpretación de los coeficientes

Sigue sin ser muy interpretable, pero despejando, se puede demostrar que el impacto estimado de cada año vivido sobre la probabilidad de tener seguro médico (manteniendo todas las otras variables constantes) viene dada por la siguiente fórmula

$$\frac{e^{\beta_1}}{1+e^{\beta_1}} \approx \frac{e^{0.03065163}}{1+e^{0.03065163}} \approx 0.5077$$

De todas formas, se puede hacer una interpretación más rudimentaria, en la que se pueden asociar los signos negativos de los coeficientes con una reducción en la probabilidad de tener cobertura médica, mientras que se da lo inverso para coeficientes positivos. A medida que estos coeficientes se alejan del cero, tiene más influencia sobre la probabilidad estimada de tener cobertura médica.

Predicción de probabilidad

En python, una vez estimados los coeficientes β, se puede estimar la probabilidad de tener cobertura sabiendo los valores de las variables predictivas.

Rápidamente, en python se puede hacer del siguiente modo, estimando por ejemplo, la probabilidad de el primer individuo (fila 0) tenga cobertura médica:

```
print(logreg.predict_proba(X)[0,1])
```

```
## 0.9677999736808303
```

Recordemos cómo se interpreta esta probabilidad. El hecho de que dé como resultado 97.66% significa que, según el modelo, si se tuvieran un gran número de personas con **las mismas características** que ese individuo, aproximadamente el 97.66% de ellas tendría cobertura médica.

Verificación del cálculo

Para ver cómo se obtiene este valor, explicaremos a continuación cómo se puede estimar la probabilidad para el primer individuo. Primero vemos los valores de las variables para este individuo y los coeficientes correspondientes

```
print(X.iloc[[0],:])
```

```
##     age  wage  ...  esr_Not in Labor Force  esr_Unemployed
## 0  60.0  66.0  ...                     0.0             0.0
##
## [1 rows x 26 columns]
```

```
print(df_coef)
```

```
##         age      wage  ...  esr_Not in Labor Force  esr_Unemployed
##    0.030652  0.025321  ...                 0.18883       -1.099499
##
## [1 rows x 26 columns]
```

Multiplicando cada valor con su respectivo β se obtiene lo siguiente:

```
dfProd=X.iloc[[0],:]*logreg.coef_

print(dfProd)
```

```
##         age      wage  ...  esr_Not in Labor Force  esr_Unemployed
## 0  1.839098  1.671197  ...                     0.0            -0.0
##
## [1 rows x 26 columns]
```

Vemos que multiplicando los primeros dos β con el valor de la variable correspondiente se obtienen los valores de `dfProd`:

```
print(0.03065163*60)
```

```
## 1.8390978
```

```
print(0.02532117*66)
```

```
## 1.67119722
```

Verificación del cálculo

Entonces, sumando todos estos valores se obtiene la estimación del log(odds):

```
expo=np.sum(dfProd.values)
print(expo)
```

```
3.4030581572488288
```

Recordando cómo se obtiene la probabilidad a partir del log(odds), se termina obteniendo el valor estimado de la probabilidad del siguiente modo:

$$\widehat{p} = \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}}$$

.

Vemos que con esta cuenta da como resultado el mismo que la predicción dada por python.

```
prob=np.exp(expo)/(1+np.exp(expo))
print(prob)
```

```
0.9677999736808304
```

Predicciones de probabilidad

Ahora, en el siguiente vector, vamos a guardar las probabilidades estimadas para todos los individuos. Aquí mostramos las probabilidades estimadas para los primeros 5:

```
y_pred_probs = logreg.predict_proba(X)[:, 1]
print(y_pred_probs[range(5)])
```

```
[0.96779997 0.94911896 0.95148466 0.97996217 0.94817449]
```

Predicción de respuesta

De todas formas, en la práctica, estas probabilidades estimadas pueden servir para hacer un análisis fino de la cobertura médica, pero en definitiva, para nuevas observaciones queremos saber más que nada si para una nueva observación, predecimos si va a tener cobertura medica o no. Es decir, es de interés clasificar a cada individuo según la primer variable binaria que utilizamos como variable de respuesta.

Para clasificar los nuevos individuos según si tienen cobertura o no, el criterio por default para la regresión logística sería clasificarlos como personas con cobertura médica si la probabilidad estimada es mayor que 50% y clasificarla como sin cobertura en caso contrario. Eso en python se hace con el método `predict`:

```
y_pred=logreg.predict(X)
print(y_pred_probs[range(5)])
print(y_pred[range(5)])
```

```
[0.96779997 0.94911896 0.95148466 0.97996217 0.94817449]
```

```
[1. 1. 1. 1. 1.]
```

Notemos que todas las respuestas predichas son 1, es decir, se predice que esas 5 personas tienen cobertura médica. Esto es consistente con que todas las probabilidades estimadas eran mayores que 50%. Más aún, dan cercanos a 95%.

Predicción de respuesta

Podemos ver que los individuos clasificados como 1 tienen probabilidad estimada mayor que 0.5 y los clasificados como 0 tienen probabilidad estimada por debajo de 0.5:

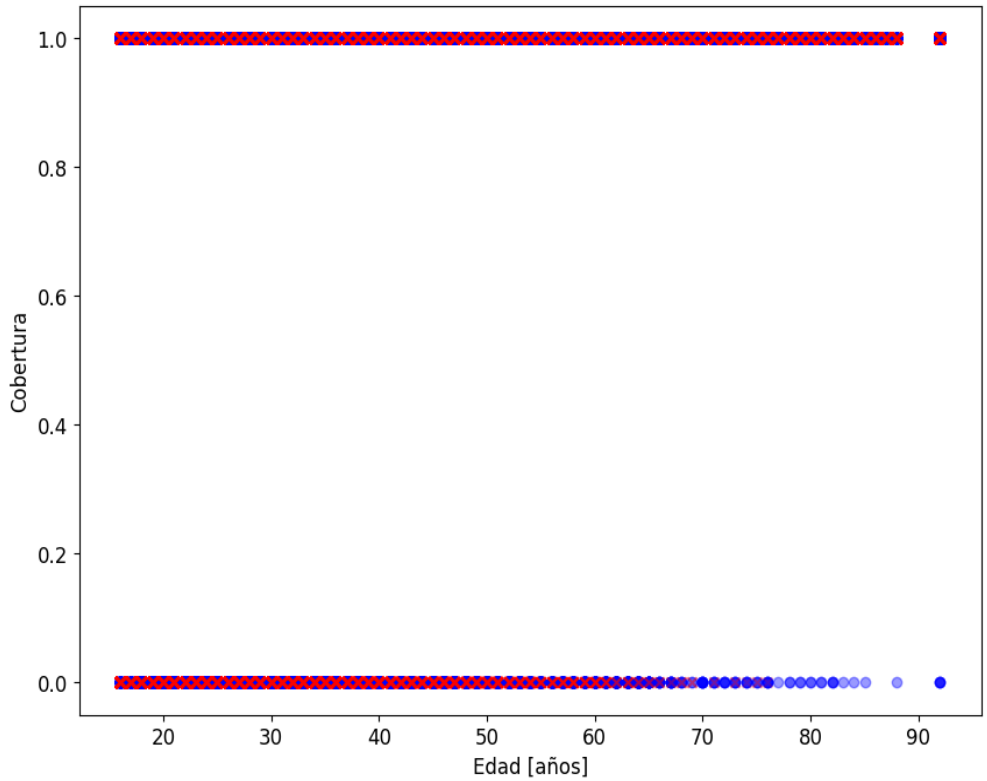
```
Ind1=np.where(y_pred==1)
print(y_pred_probs[Ind1])
print(y_pred[Ind1])
Ind0=np.where(y_pred==0)
print(y_pred_probs[Ind0])
print(y_pred[Ind0])
```

```
Individuos predichos con cobertura

Probabilidad estimada

[0.96779997 0.94911896 0.95148466 ... 0.99887735 0.88699651 0.9736503 ]

Respuesta estimada

[1. 1. 1. ... 1. 1. 1.]

Individuos predichos sin cobertura

Probabilidad estimada

[0.43947055 0.39983114 0.37197409 ... 0.42790857 0.46762629 0.41316799]

Respuesta estimada

[0. 0. 0. ... 0. 0. 0.]
```

Verificación gráfica

Ahora que se pueden predecir las respuestas binarias, repetimos el gráfico en el que se superponen los valores reales con sus predicciones correspondientes para ver que el modelo se adapta mejor a los datos que la regresión lineal:

```
plt.scatter(X["age"],y,alpha=0.4,c="b")
plt.scatter(X["age"],y_pred,alpha=0.2,c="r",marker="x")
plt.xlabel("Edad [años]")
plt.ylabel("Cobertura")
plt.show()
```



Evaluación de las respuestas

Estas predicciones pueden tener errores, y en este caso sabemos las respuestas reales. Para evaluar la calidad de estas predicciones, podemos generar lo que se llama una matriz de confusión.

```
from sklearn.metrics import confusion_matrix
CMat=confusion_matrix(y, y_pred)
dfCM=pd.DataFrame({"Clas s/SM":CMat[:,0],"Clas c/SM":CMat[:,1]},index=["s/SM","c/SM"])
```

```
print(dfCM)
```

```
##       Clas s/SM  Clas c/SM
## s/SM       3173      10518
## c/SM       1902      59212
```

Evaluación de las respuestas

De todas las predicciones, las que son correctas están sobre la diagonal, ya que coincide la clase real con la estimada. Es decir, si consideramos que una persona con seguro médico es una detección "positiva", tendríamos:

- VN= 3173 (verdaderos "negativos")
- FN= 1902 (falsos "negativos")
- FP= 10518 (falsos "positivos")
- VN= 59212 (verdaderos "positivos")

Esto da lugar a una primer métrica de calidad de clasificación denominada "accuracy" y en python se calcula del siguiente modo:

```
print(logreg.score(X,y))
```

```
0.8339683176258271
```

Es decir, bajo estas estimaciones, se predicen correctamente el 83.39% de los datos.

Evaluación de las predicciones

Sin embargo, pueden haber métricas más complicadas, que evalúan si hay tendencias de clasificaciones erróneas en algunas clases particulares.

- Por ejemplo, notemos que entre los clasificados como "negativos", es mayor la proporción de falsos negativos ($\frac{\text{Falsos negativos}}{\text{Clasificados Negativos}} = \frac{1902}{1902+3173} \approx 0.3748$) que la proporción de falsos positivos entre los clasificados como "positivos" ($\frac{\text{Falsos positivos}}{\text{Clasificados Positivos}} = \frac{10518}{10518+59212} \approx 0.1508$).
- Del mismo modo, entre los realmente "negativos", las clasificaciones incorrectas son una proporción elevada ($\frac{\text{Falsos negativos}}{\text{Negativos totales}} = \frac{10518}{10518+3173} \approx 0.7682$), respecto de las clasificaciones incorrectas entre los realmente "positivos" ($\frac{\text{Falsos positivos}}{\text{Positivos totales}} = \frac{1902}{1902+59212} \approx 0.03112$).

Estas magnitudes van mostrando que los errores al predecir las respuestas de los individuos sin cobertura médica son mayores que los errores al predecir los errores de los individuos con cobertura médica. Es decir, la regresión logística propuesta predice mejor a la clase mayoritaria de las personas con seguro médico. El valor del "accuracy" no permitía ver estos sesgos.

Evaluación de las predicciones

Estas cuestiones se pueden ver con la función `classification_report`.

```
from sklearn.metrics import classification_report
CR=classification_report(y, y_pred)

print(CR)
```

```
##              precision recall  f1-score  support
## 0.0               0.63   0.23      0.34    13691
## 1.0               0.85   0.97      0.91    61114
## accuracy                           0.83    74805
## macro avg         0.74    0.6      0.62    74805
## weighted avg      0.81   0.83      0.80    74805
```

Interpretando estos valores,

- la métrica "precision" representa cuantas de las clasificaciones de cada clase son correctas. Es decir:

$$\frac{\text{Verdaderos negativos}}{\text{Clasificados Negativos}} = \frac{3173}{1902+3173} \approx 0.63; \quad \frac{\text{Verdaderos positivos}}{\text{Clasificados Positivos}} = \frac{59212}{10518+59212} \approx 0.85$$

- la métrica "recall" representa la proporción de cada clase que fueron correctamente clasificadas. Es decir:

$$\frac{\text{Verdaderos negativos}}{\text{Negativos totales}} = \frac{3173}{10518+3173} \approx 0.23; \quad \frac{\text{Verdaderos positivos}}{\text{Positivos totales}} = \frac{59212}{1902+59212} \approx 0.97$$

- la métrica "f1-score" combina de forma equitativa tanto las métricas "precision" y "recall", aunque no es exactamente un promedio entre ambas.

Curva ROC

Curva ROC

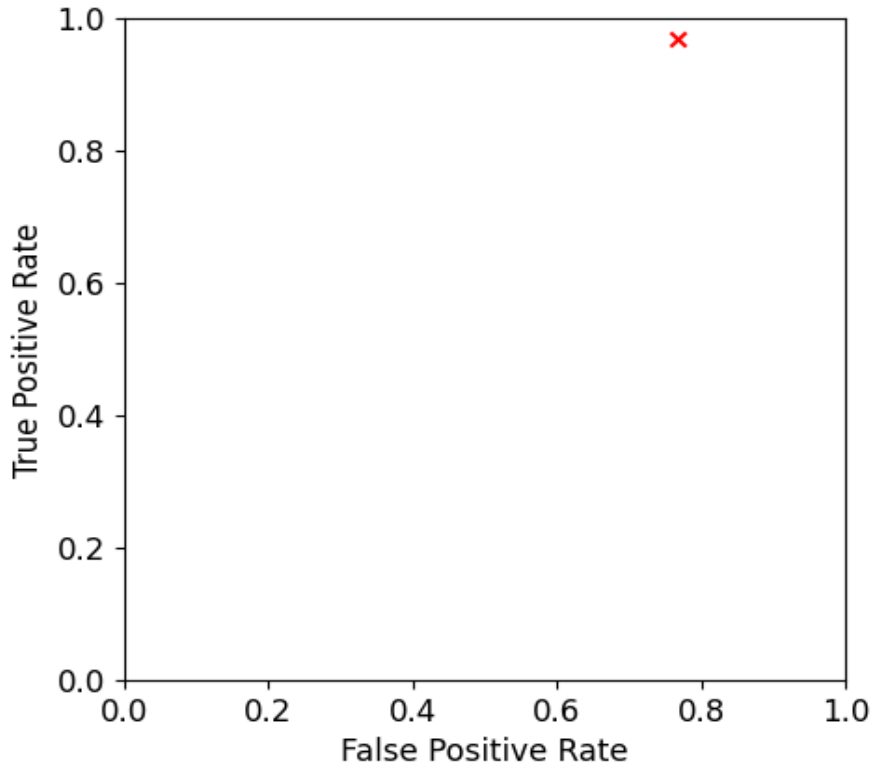
Otra forma de evaluar la calidad de un clasificador **binario** es utilizando lo que se llama la curva ROC (Receiver Operating Characteristic), en el que se evalúan en simultáneo los falsos y los verdaderos positivos. Los primeros se ubican en la coordenada x y los segundos en la coordenada y.

Hemos visto que según la clasificación como "positiva" (con valor 1) si la probabilidad estimada era mayor que 0.5 y "negativa" (con valor 0) si la probabilidad estimada era menor que 0.5, obtuvimos una proporción de falsos positivos entre los realmente negativos dada por $x = \frac{10518}{3173 + 10518} \approx 0.7682$ mientras que entre los positivos, la proporción de verdaderos positivos es $y = \frac{59212}{59212 + 1902} \approx 0.9689$.

Curva ROC

En el plano ROC, se grafica el punto del siguiente modo:

```
x1=10518/(10518+3173);y1=59212/(59212+1902)
plt.scatter(x1,y1,c="r",marker="x")
plt.xlim([0,1]);plt.ylim([0,1])
plt.xlabel('False Positive Rate');plt.ylabel('True Positive Rate')
plt.show()
```



Curva ROC

De todas formas, una vez estimadas las probabilidades, se pueden establecer distintos umbrales (en vez de 0.5) a partir de los cuales, las probabilidades estimadas por debajo de dicho valor, se clasifiquen como 0 y en caso contrario, se clasifiquen como 1. Ahora mostraremos tomando como umbrales 0.1, 0.25, 0.75, 0.9.

```
ps=[0.1,0.25,0.75,0.9];xs=[0]*4;ys=[0]*4;lCM=[]
for i in range(4):
  y_pred_Aux=(y_pred_probs>=ps[i]) * 1
  CMat=confusion_matrix(y, y_pred_Aux)
  xs[i]=CMat[0,1]/np.sum(CMat[0,:])
  ys[i]=CMat[1,1]/np.sum(CMat[1,:])
  lCM[i]=pd.DataFrame({"Clas s/SM":CMat[:,0],"Clas c/SM":CMat[:,1]},index=["s/SM","c/S
```

```
print(sPs[0])
```

```
## p=0.1
```

```
print(lCM[0])
```

```
##       Clas s/SM  Clas c/SM
## s/SM        159      13532
## c/SM         20      61094
```

```
print("Tasa Falsos Positivos:"+str(xs[0]))
```

```
## Tasa Falsos Positivos:0.988386531297933
```

```
print("Tasa Verdaderos Positivos:"+str(ys[0]))
```

```
## Tasa Verdaderos Positivos:0.9996727427430703
```

Curva ROC

De todas formas, una vez estimadas las probabilidades, se pueden establecer distintos umbrales (en vez de 0.5) a partir de los cuales, las probabilidades estimadas por debajo de dicho valor, se clasifiquen como 0 y en caso contrario, se clasifiquen como 1. Ahora mostraremos tomando como umbrales 0.1, 0.25, 0.75, 0.9.

```
ps=[0.1,0.25,0.75,0.9];xs=[0]*4;ys=[0]*4;lCM=[]
for i in range(4):
  y_pred_Aux=(y_pred_probs>=ps[i]) * 1
  CMat=confusion_matrix(y, y_pred_Aux)
  xs[i]=CMat[0,1]/np.sum(CMat[0,:])
  ys[i]=CMat[1,1]/np.sum(CMat[1,:])
  lCM[i]=pd.DataFrame({"Clas s/SM":CMat[:,0],"Clas c/SM":CMat[:,1]},index=["s/SM","c/S
```

```
print(sPs[1])
```

```
## p=0.25
```

```
print(lCM[1])
```

```
##       Clas s/SM  Clas c/SM
## s/SM        887      12804
## c/SM        274      60840
```

```
print("Tasa Falsos Positivos:"+str(xs[1]))
```

```
## Tasa Falsos Positivos:0.9352129135928712
```

```
print("Tasa Verdaderos Positivos:"+str(ys[1]))
```

```
## Tasa Verdaderos Positivos:0.9955165755800635
```

Curva ROC

De todas formas, una vez estimadas las probabilidades, se pueden establecer distintos umbrales (en vez de 0.5) a partir de los cuales, las probabilidades estimadas por debajo de dicho valor, se clasifiquen como 0 y en caso contrario, se clasifiquen como 1. Ahora mostraremos tomando como umbrales 0.1, 0.25, 0.75, 0.9.

```
ps=[0.1,0.25,0.75,0.9];xs=[0]*4;ys=[0]*4;lCM=[]
for i in range(4):
  y_pred_Aux=(y_pred_probs>=ps[i]) * 1
  CMat=confusion_matrix(y, y_pred_Aux)
  xs[i]=CMat[0,1]/np.sum(CMat[0,:])
  ys[i]=CMat[1,1]/np.sum(CMat[1,:])
  lCM[i]=pd.DataFrame({"Clas s/SM":CMat[:,0],"Clas c/SM":CMat[:,1]},index=["s/SM","c/S
```

```
print(sPs[2])
```

```
## p=0.75
```

```
print(lCM[2])
```

```
##       Clas s/SM  Clas c/SM
## s/SM       8626       5065
## c/SM      11880      49234
```

```
print("Tasa Falsos Positivos:"+str(xs[2]))
```

```
## Tasa Falsos Positivos:0.3699510627419473
```

```
print("Tasa Verdaderos Positivos:"+str(ys[2]))
```

```
## Tasa Verdaderos Positivos:0.8056091893837746
```

Curva ROC

De todas formas, una vez estimadas las probabilidades, se pueden establecer distintos umbrales (en vez de 0.5) a partir de los cuales, las probabilidades estimadas por debajo de dicho valor, se clasifiquen como 0 y en caso contrario, se clasifiquen como 1. Ahora mostraremos tomando como umbrales 0.1, 0.25, 0.75, 0.9.

```
ps=[0.1,0.25,0.75,0.9];xs=[0]*4;ys=[0]*4;lCM=[]
for i in range(4):
  y_pred_Aux=(y_pred_probs>=ps[i]) * 1
  CMat=confusion_matrix(y, y_pred_Aux)
  xs[i]=CMat[0,1]/np.sum(CMat[0,:])
  ys[i]=CMat[1,1]/np.sum(CMat[1,:])
  lCM[i]=pd.DataFrame({"Clas s/SM":CMat[:,0],"Clas c/SM":CMat[:,1]},index=["s/SM","c/S
```

```
print(sPs[3])
```

```
## p=0.9
```

```
print(lCM[3])
```

```
##       Clas s/SM  Clas c/SM
## s/SM      12494       1197
## c/SM      27851      33263
```

```
print("Tasa Falsos Positivos:"+str(xs[3]))
```

```
## Tasa Falsos Positivos:0.08742969834197648
```

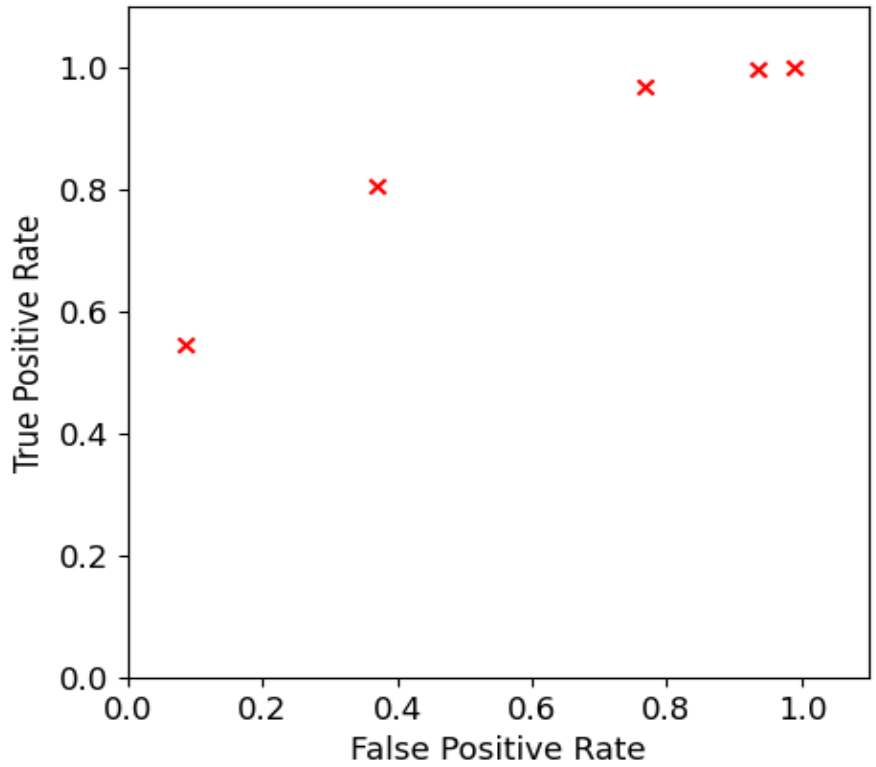
```
print("Tasa Verdaderos Positivos:"+str(ys[3]))
```

```
## Tasa Verdaderos Positivos:0.5442779068625847
```

Curva ROC

Vemos que menores umbrales dan mayor número de detecciones "positivas" y por lo tanto, mayor proporción de verdaderos positivos pero a costa de mayores falsos positivos. Sumemos al gráfico estos puntos:

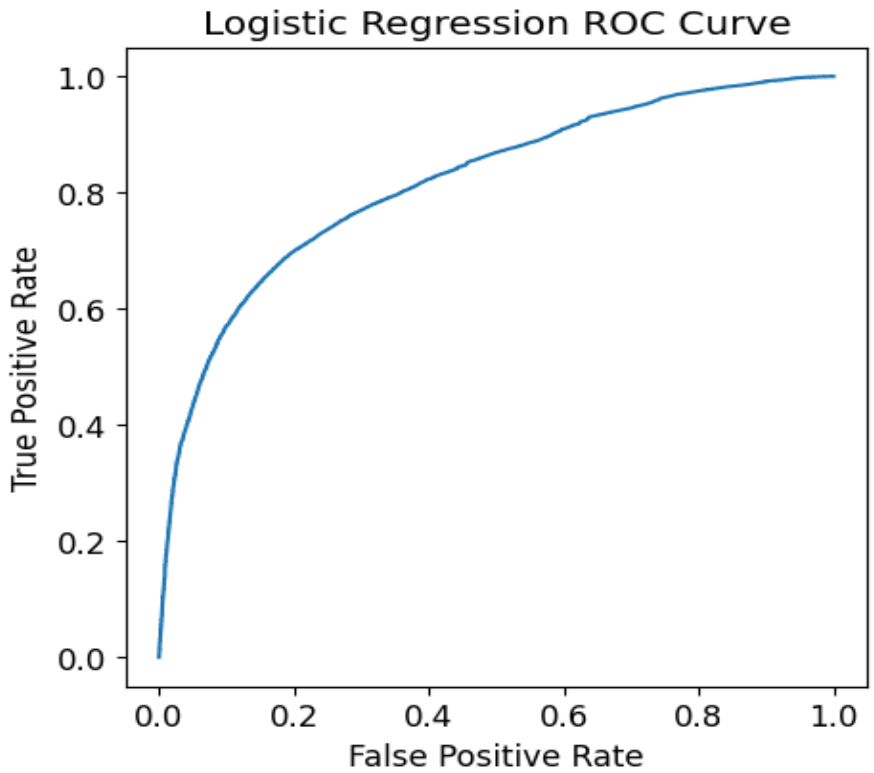
```
plt.scatter(x1,y1,c="r",marker="x")
plt.scatter(xs,ys,c="r",marker="x")
plt.xlim([0,1.1]);plt.ylim([0,1.1])
plt.xlabel('False Positive Rate');plt.ylabel('True Positive Rate')
plt.show()
```



Curva ROC

Dando muchos más valores a los umbrales, recorriendo valores entre 0 y 1, se pueden unir con una línea y obtener la curva que se denomina "Curva ROC":

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y, y_pred_probs)
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate');plt.ylabel('True Positive Rate')
plt.title('Logistic Regression ROC Curve')
plt.show()
```



Área bajo la curva ROC

Obviamente no tenemos detecciones perfectas, pero si pudiéramos tenerlas al menos como objetivo, tendríamos un 0% de falsos positivos y un 100% de verdaderos positivos. Es decir, la detección perfecta está en el punto (0,1) de ese espacio (la esquina superior izquierda). Por lo tanto, mientras más se acerquen la combinaciones de falsos positivos y verdaderos positivos a ese punto, mejor serán las clasificaciones. Por lo tanto, muchas veces se suele calcular el área bajo esa curva, para evaluar cuánto se acerca la curva a ese valor óptimo.

```
from sklearn.metrics import roc_auc_score
print(roc_auc_score(y, y_pred_probs))
```

```
0.8175553580772248
```

Esta área tiene un valor máximo de 1 (área del cuadrado que limita ambas coordenadas), mientras que las áreas debajo de 0.5 (correspondiente al triángulo bajo la diagonal) dan muestras de malas clasificaciones, ya que tendría menor rendimiento que asignar al azar 50% de las observaciones a una clase y al otro 50% a la clase restante.

Comparación con otro modelo

Supongamos que planteamos un peor modelo, en el que el hecho de tener cobertura médica depende **únicamente** de la edad de la persona. Por lo que hemos visto, hay muchos más factores que influyen sobre la cobertura médica, por lo que deberían arrojar peores métricas de evaluación:

```
# Considero en este modelo simple sólo la variable "age"
X0=df[["age"]]
logreg0 = LogisticRegression(fit_intercept=False)
# Calculo los coeficientes
logreg0.fit(X0,y)
# Calculo las predicciones
y_pred0=logreg0.predict(X0)
y_pred_probs0=logreg0.predict_proba(X0)[:,1]
```

Veamos cuánto da la métrica "accuracy":

```
print("Accuracy modelo completo:")
print(logreg.score(X,y))
print("Accuracy modelo simple:")
print(logreg0.score(X0,y))
```

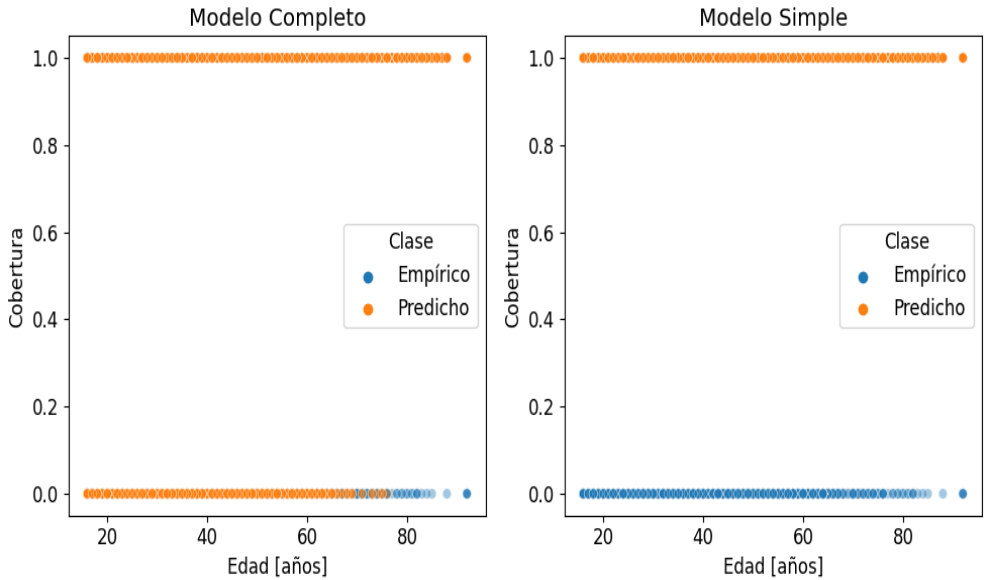
```
Accuracy modelo completo: 0.8339683176258271

Accuracy modelo simple: 0.8169774747677294
```

Vemos que los valores son similares, por lo que no pareciera ser tanto peor el modelo simple respecto del completo.

Comparación gráfica

Veamos cómo se ajustan las predicciones en ambos casos a las respuestas reales:

```
fig,axes=plt.subplots(1,2)
g1=sns.scatterplot(data=dfComp1,x="age",y="Respuesta",alpha=0.4,hue="Clase"
g1.set_xlabel("Edad [años]");g1.set_ylabel("Cobertura");g1.set_title("Modelo
g2=sns.scatterplot(data=dfComp2,x="age",y="Respuesta",alpha=0.4,hue="Clase"
g2.set_xlabel("Edad [años]");g2.set_ylabel("Cobertura");g2.set_title("Modelo
fig.tight_layout()
plt.show()
```



Se ve que en el modelo simple ninguna persona es clasificada como "sin cobertura", por lo que directamente asignó todas las respuestas a la clase mayoritaria con cobertura, sin ningún tipo de distinción.

Comparación de métricas

Esto se ve con más claridad en las matrices de confusión:

```
CMat0=confusion_matrix(y, y_pred0)
dfCM0=pd.DataFrame({"Clas s/SM":CMat0[:,0],"Clas c/SM":CMat0[:,1]},index=["s/SM","c/SM
```

```
print("Modelo Completo:")
```

```
## Modelo Completo:
```

```
print(dfCM)
```

```
##       Clas s/SM  Clas c/SM
## s/SM       3173      10518
## c/SM       1902      59212
```

```
print("Modelo Simple:")
```

```
## Modelo Simple:
```

```
print(dfCM0)
```

```
##       Clas s/SM  Clas c/SM
## s/SM          0      13691
## c/SM          0      61114
```

Comparación de métricas

Más aún, en el `classification_report` se muestran todas las métricas:

```
print("Modelo Completo:")
```

```
## Modelo Completo:
```

```
print(CR)
```

```
##              precision recall  f1-score  support
## 0.0               0.63   0.23      0.34    13691
## 1.0               0.85   0.97      0.91    61114
## accuracy                           0.83    74805
## macro avg         0.74    0.6      0.62    74805
## weighted avg      0.81   0.83      0.80    74805
```

```
print("Modelo Simple:")
```

```
## Modelo Simple:
```

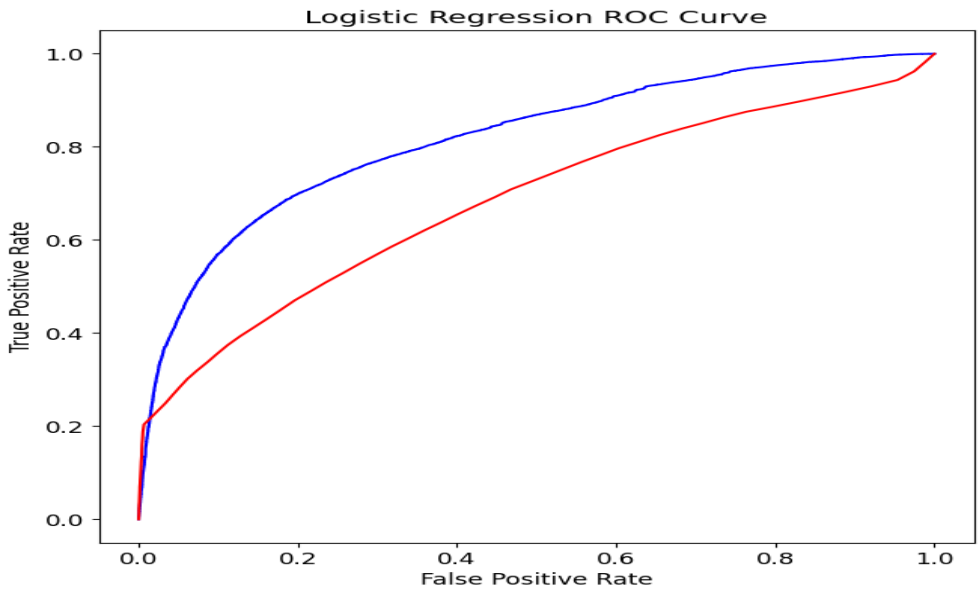
```
print(CR0)
```

```
##              precision recall  f1-score  support
## 0.0               0.63   0.23      0.34    13691
## 1.0               0.85   0.97      0.91    61114
## accuracy                           0.83    74805
## macro avg         0.74    0.6      0.62    74805
## weighted avg      0.81   0.83      0.80    74805
```

Comparación de Curvas ROC

También es notoria la diferencia entre las curvas ROC, donde la curva del modelo simple se agrega en color rojo:

```
fpr0, tpr0, thresholds0 = roc_curve(y, y_pred_probs0)
plt.plot(fpr, tpr,c="b");plt.plot(fpr0, tpr0,c="r")
plt.xlabel('False Positive Rate');plt.ylabel('True Positive Rate')
plt.title('Logistic Regression ROC Curve')
plt.show()
```



Comparación de Áreas bajo la Curva ROC

La diferencia también se expresa en las áreas bajo estas curvas:

```
print("Modelo Completo:")
print(roc_auc_score(y, y_pred_probs))
print("Modelo Simple:")
print(roc_auc_score(y, y_pred_probs0))
```

```
Modelo Completo: 0.8175553580772248

Modelo Simple: 0.6827424900082738
```

Resumen

Hemos visto una forma de modelar el vínculo entre una variable de respuesta **cualitativa binaria** con otras variables explicativas.

- Ventajas:
  - Al plantear un modelo, se puede **medir** la influencia de una variable sobre la variable de respuesta a través del coeficiente β.
  - Además de la respuesta, se puede estimar una probabilidad de que nuevas observaciones cumplan una de ambas condiciones, pudiendo adaptarse a la incertidumbre.
  - Evaluando la influencia de distintas variables, se pueden detectar sesgos y errores que introducen las mismas en las predicciones.
- Desventajas:
  - El modelo hay que armarlo a criterio y no siempre tenemos las variables necesarias para plantear un buen modelo.
  - Las ventajas dependen de que partan de un buen modelo, que como dijimos, no es fácil de conseguir.
  - Es computacionalmente más "caro" que otros modelos más simples, pero que no poseen las ventajas anteriores.

Bibliografía

Jeffrey C. Chen, Edward A. Rubin, Gary J. Cornwall - Data Science for Public Policy (Springer Series in the Data Sciences)-Springer (2021)