*PRÁCTICA FINAL DE CLASIFICACIÓN*

Teresa de Silva e Inés Jiménez, Máster en Data Science- Modelos Supervisados

Este trabajo está hecho sobre el dataset Canadiense que recopila los datos de accidentes hasta el año 2014. El problema reside en que las empresas aseguradoras necesitan hacer predicciones en base a los datos históricos, ya que tienen que crear provisiones que cubran gastos por accidentes. Para este trabajo vamos a tratarlo como un problema de clasificación. El modelo principal que elaboraremos es en el que dado un accidente podremos predecir sí causará un fallecimiento o no. El orden del proceso de trabajo se basa en:

- Importación y limpieza de datos
- Preprocesamiento de datos
- Análisis estadístico
- Elaboración del modelo de clasificación de accidentes
- Métricas del modelo y resultados

## Parte 1

Importamos las librerías basicas, así como el dataset. Realizamos un análisis preliminar para poder estudiar la distribución del dataset, los valores nulos y outliers.

In [1]:

```
#Basic libraries
import pandas as pd
import numpy as np
import seaborn as sns
import pylab as pl
import scipy.optimize as opt
from itertools import cycle
from matplotlib import pyplot as plt
import plotly.express as px
from sklearn.impute import KNNImputer
from category_encoders.one_hot import OneHotEncoder
import scipy.stats as ss
import warnings


#Sklearn
from scipy import interp
from sklearn import metrics
from sklearn import linear_model
from sklearn import svm, datasets
from sklearn import model_selection
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import label_binarize
from sklearn.metrics import classification_report
from sklearn.multiclass import OneVsRestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree
from sklearn import preprocessing
from sklearn.tree import export_graphviz
from sklearn.tree import export_text
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import GradientBoostingClassifier
from sklearn import metrics
from sklearn import linear_model
from sklearn.multiclass import OneVsRestClassifier
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import _tree
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix, accuracy_score, r
oc_auc_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree


%matplotlib inline


#Gráficos
import matplotlib.pyplot as plt
%matplotlib inline
import plotly.express as px
import seaborn as sns
from matplotlib.colors import ListedColormap
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')
import matplotlib.patches as mpatches

#Función para matriz de confusión

def plot_confusion_matrix(confmat):
    fig, ax = plt.subplots(figsize=(3, 3))
    ax.matshow(confmat, cmap=plt.cm.Blues, alpha=0.5)
    for i in range(confmat.shape[0]):
        for j in range(confmat.shape[1]):
            ax.text(x=j, y=i, s=confmat[i, j], va='center', ha='center')

    plt.xlabel('predicted label')
    plt.ylabel('true label')

    plt.tight_layout()
    plt.show()

cm = plt.cm.RdBu
cm_bright = ListedColormap(['#FF0000', '#0000FF'])

import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
data=pd.read_csv('NCDB_1999_to_2014.csv')
```

Debido al volumen del dataset, hemos estado intentando trabajar con toda la muestra, pero ya llegado al uso de modelo no nos funcionaba jupyter y los tiempos de espera eran demasiado largos. Por ello, seleccionamos una muestra.

In [15]:

```
data=data.sample(n=500000, random_state=5)
```

Comprobamos el tipo de variable que corresponde a cada columna.

In [ ]:

```
data.dtypes
```

*Limpieza de datos*

Para ello, utilizamos la función establecida que busca los Na´s.

In [16]:

```
def missingsummary(dataframe):
    numelements = dataframe.count()
    nummissing = dataframe.isna().sum()
    missingsummary = pd.DataFrame(index=numelements.index,
                                  data={'Total':numelements,
                                        'Missing':nummissing,
                                        'Missing_rate (%)': round(nummissing/numelements * 100, 2)})
    return missingsummary
missingsummary(data)
```

Out[16]:

| | Total | Missing | Missing_rate (%) |
|---|---|---|---|
| C_YEAR | 500000 | 0 | 0.0 |
| C_MNTH | 500000 | 0 | 0.0 |
| C_WDAY | 500000 | 0 | 0.0 |
| C_HOUR | 500000 | 0 | 0.0 |
| C_SEV | 500000 | 0 | 0.0 |
| C_VEHS | 500000 | 0 | 0.0 |
| C_CONF | 500000 | 0 | 0.0 |
| C_RCFG | 500000 | 0 | 0.0 |
| C_WTHR | 500000 | 0 | 0.0 |
| C_RSUR | 500000 | 0 | 0.0 |
| C_RALN | 500000 | 0 | 0.0 |
| C_TRAF | 500000 | 0 | 0.0 |
| V_ID | 500000 | 0 | 0.0 |
| V_TYPE | 500000 | 0 | 0.0 |
| V_YEAR | 500000 | 0 | 0.0 |
| P_ID | 500000 | 0 | 0.0 |
| P_SEX | 500000 | 0 | 0.0 |
| P_AGE | 500000 | 0 | 0.0 |

| | | | |
|---|---|---|---|
| P_PSN | 500000 | 0 | 0.0 |
| P_ISEV | 500000 | 0 | 0.0 |
| P_SAFE | 500000 | 0 | 0.0 |
| P_USER | 500000 | 0 | 0.0 |

In [17]:

```
data['C_VEHS']=data['C_VEHS'].fillna(data['C_VEHS'].mode())
```

Vemos que no hay ningún NA excepto en la columna que indica el número de vehículos en un accidente, los cuáles podemos sustituir rapidamente por la moda. Aunque no haya NA si que tenemos otros datos nulos ( que se pueden ver en el pdf adjunto al dataset en Kaggle), ya que en las colunmnas encontramos valores como "NN" o "UU" que indican desconocimiento o que los datos no han sido dados, respectivamente. Para estudiar la existencia de estos utilizamos la función value_counts() en cada columna.

In [18]:

```
def contar(dataframe):
    return dataframe.value_counts()

df = data.apply(contar)
```

Aunque estuvimos tratando de elaborar una función que nos enseñase de manera más ilustrativa los valores nulos, finalmente utilizamos este "apaño", el cual enseña el recuento de valores para cada valor de letras por columna.

In [19]:

```
df.tail(n=15)
```

Out[19]:

| | C_YEAR | C_MNTH | C_WDAY | C_HOUR | C_SEV | C_VEHS | C_CONF | C_RCFG | C_WTHR | C_RSUR | ... | V_ID | V_T |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 91 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | Nal |
| 92 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | Nal |
| 93 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | Nal |
| 94 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | Nal |
| 95 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | Nal |
| 96 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | Nal |
| 97 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | Nal |
| 98 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | Nal |
| 99 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 11741.0 | Nal |
| F | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | Nal |
| M | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | Nal |
| NN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | Nal |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NNNN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | Nal |
| UU | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | Nal |
| UUUU | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | Nal |

15 rows × 22 columns

Vemos que efectivamente en cada columna encontramos esos valores nulos. Para ello, vamos a reemplazamos los valores nulos por la moda. Los reemplazamos por la moda ya que no queremos que por ejemplo en la columna C_WEEKDAY, un valor sea 2.7, entre martes y miércoles.

In [20]:

```
data['C_MNTH']= data['C_MNTH'].replace(["UU"],8)
data['C_WDAY']= data['C_WDAY'].replace(["U"],5)
data['C_HOUR']= data['C_HOUR'].replace(["UU"],16)
data['C_VEHS']= data['C_VEHS'].replace(["UU"],2)
data['C_RCFG']= data['C_RCFG'].replace(["UU","QQ"],'02')
data['C_WTHR']= data['C_WTHR'].replace(["U","Q"],1)
data['C_RSUR']= data['C_RSUR'].replace(["U","Q"],1)
data['C_RALN']= data['C_RALN'].replace(["U","Q"],1)
data['C_TRAF']= data['C_TRAF'].replace(["UU","QQ"],18)
data['C_CONF']= data['C_CONF'].replace(["UU", "QQ"],21)
data['V_ID']= data['V_ID'].replace(["UU"], "01")
data['V_TYPE']= data['V_TYPE'].replace(["NN", "QQ","UU"],21)
data['P_ID']= data['P_ID'].replace(["NN", "UU"],"01")
data['P_SEX']= data['P_SEX'].replace(["U","N"],"M")
data['P_PSN']= data['P_PSN'].replace(["UU", "QQ","NN"],11)
data['P_ISEV']= data['P_ISEV'].replace(["U", "N"],2)
data['P_SAFE']= data['P_SAFE'].replace(["NN", "QQ","UU"],"02")
data['P_USER']= data['P_USER'].replace(["U"],1)
```

In [ ]:

```
def contar(dataframe):
    return dataframe.value_counts()

df_limpio = data.apply(contar)
```

In [ ]:

```
df_limpio.tail(n=15)
```

Como podemos ver, en la columna Person Age y Vehicle year, la variable más común es un Na. Si sustituimos ese valor por el segundo año o edad que aparece más veces, vamos a distorsionar los datos demasiado para el análisis. Por ello, ya que en la columna Person Age son 18,016 y 377,140 filas y representan un 6.74% del dataset, vamos a eliminarlas. Lo mismo sucede con el año del modelo del coche, donde el valor "UUUU" ocupa 260,256 filas, representando un 4.4%. Es cierto que eliminar datos elimina información, pero preferimos trabajar sin estos datos y sin modificar las distribuciones de las variables donde el valor mas repetido es un valor que no aporta información detallada.

In [21]:

```
data_clean = data.drop(data[data['V_YEAR']=="NNNN"].index)
data_clean_1 = data_clean.drop(data_clean[data_clean['V_YEAR']=="UUUU"].index)
data_clean_2 = data_clean_1.drop(data_clean_1[data_clean_1['P_AGE']=="UU"].index)
data_clean_all = data_clean_2.drop(data_clean_2[data_clean_2['P_AGE']=="NN"].index)
```

In [176]:

```
data_clean_all
```

Out[176]:

| | C_YEAR | C_MNTH | C_WDAY | C_HOUR | C_SEV | C_VEHS | C_CONF | C_RCFG | C_WTHR | C_RSUR | ... | V_ID | V_TYPE | V_YEAR | P_ID | I |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4124328 | 2009 | 8 | 2 | 16 | 2 | 02 | 34 | 05 | 1 | 1 | ... | 2 | 01 | 1987 | 01 | |
| 3824244 | 2008 | 9 | 3 | 16 | 2 | 02 | 35 | 01 | 1 | 1 | ... | 01 | 01 | 2007 | 01 | |
| 336701 | 1999 | 10 | 7 | 09 | 2 | 2 | 23 | 01 | 1 | 1 | ... | 01 | 01 | 1996 | 01 | |
| 788058 | 2000 | 11 | 6 | 15 | 2 | 2 | 21 | 02 | 1 | 1 | ... | 01 | 01 | 1993 | 02 | |
| 4949818 | 2012 | 1 | 7 | 17 | 2 | 2 | 33 | 02 | 1 | 1 | ... | 2 | 01 | 2010 | 01 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 1617196 | 2002 | 11 | 6 | 11 | 2 | 1 | 03 | 02 | 2 | 3 | ... | 1 | 01 | 1993 | 01 | |
| 5546889 | 2013 | 12 | 3 | 08 | 2 | 02 | 21 | 02 | 4 | 1 | ... | 2 | 01 | 2003 | 01 | |
| 2791590 | 2005 | 11 | 3 | 08 | 2 | 3 | 21 | 01 | 1 | 2 | ... | 02 | 01 | 2002 | 01 | |
| 4535196 | 2010 | 10 | 6 | 00 | 2 | 1 | 04 | 02 | 1 | 1 | ... | 1 | 01 | 1995 | 01 | |
| 4937641 | 2012 | 01 | 4 | 10 | 2 | 02 | 36 | 02 | 1 | 2 | ... | 1 | 01 | 2006 | 01 | |

428017 rows × 22 columns

Para continuar con la preparación de datos, vamos a revisar si hay outliers y las distribuciones de cada variable.

Primero convertimos a valor numérico todas las variables, incluido la columna de sexo del conductor, para que esté codificada.

In [22]:

```
from sklearn.preprocessing import OneHotEncoder
outliers_detection = pd.get_dummies(data_clean_all, columns = ['P_SEX'])
print(data_clean_all)
```

```
         C_YEAR C_MNTH C_WDAY C_HOUR  C_SEV C
_VEHS C_CONF C_RCFG C_WTHR  \
4124328    2009      8      2     16      2
02      34     05      1
3824244    2008      9      3     16      2
02      35     01      1
336701     1999     10      7     09      2
2      23     01      1
788058     2000     11      6     15      2
2      21     02      1
4949818    2012      1      7     17      2
```

```
2      33      02       1
...          ...    ...    ...    ...    ...
...    ...    ...    ...
1617196    2002      11       6      11       2
1      03      02       2
5546889    2013      12       3      08       2
02      21      02       4
2791590    2005      11       3      08       2
3      21      01       1
4535196    2010      10       6      00       2
1      04      02       1
4937641    2012      01       4      10       2
02      36      02       1

         C_RSUR  ... V_ID V_TYPE V_YEAR P_ID P
_SEX P_AGE P_PSN P_ISEV P_SAFE  \
4124328       1  ...    2      01   1987   01
M    29    11       2      02
3824244       1  ...   01      01   2007   01
M    84    11       1      02
336701        1  ...   01      01   1996   01
M    71    11       1      02
788058        1  ...   01      01   1993   02
F    13    13       2      02
4949818       1  ...    2      01   2010   01
M    32    11       1      02
...          ...  ...  ...    ...    ...  ...
...   ...    ...    ...    ...
1617196       3  ...    1      01   1993   01
F    70    11       2      02
5546889       1  ...    2      01   2003   01
M    17    11       1      02
2791590       2  ...   02      01   2002   01
M    75    11       1      02
4535196       1  ...    1      01   1995   01
F    19    11       1      02
4937641       2  ...    1      01   2006   01
```

```
F    20    11       1      02

         P_USER
4124328       1
3824244       1
336701        1
788058        2
4949818       1
...         ...
1617196       1
5546889       1
2791590       1
4535196       1
4937641       1

[428017 rows x 22 columns]
```

In [23]:

```
outliers_detection = outliers_detection.apply(pd.to_numeric, errors='coerce')
```

Como podemos ver, todos los datos están detro del rango, y no hay outliers. De todas maneras, vamos a realizar una exploración con PCA para la detección de outliers.

In [9]:

```
outliers_detection.describe()
```

Out[9]:

| | C_YEAR | C_MNTH | C_WDAY | C_HOUR | |
|---|---|---|---|---|---|
| count | 5.022347e+06 | 5.022347e+06 | 5.022347e+06 | 5.022347e+06 | 5.022347e |
| mean | 2.005958e+03 | 6.696799e+00 | 4.023687e+00 | 1.370386e+01 | 1.983385e |
| std | 4.560496e+00 | 3.448714e+00 | 1.935824e+00 | 5.158404e+00 | 1.278252 |
| min | 1.999000e+03 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 | 1.000000e |
| 25% | 2.002000e+03 | 4.000000e+00 | 2.000000e+00 | 1.000000e+01 | 2.000000e |
| 50% | 2.006000e+03 | 7.000000e+00 | 4.000000e+00 | 1.400000e+01 | 2.000000e |
| 75% | 2.010000e+03 | 1.000000e+01 | 6.000000e+00 | 1.700000e+01 | 2.000000e |
| max | 2.014000e+03 | 1.200000e+01 | 7.000000e+00 | 2.300000e+01 | 2.000000e |

8 rows × 23 columns

# PCA

Usaremos PCA para detectar outliers. Vamos a reducir los componentes, para ello haremos un train y test del dataframe de customers.

In [24]:

```
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
```

Hemos intentado hacer PCA para la detección de outliers, pero nos da que existen muchos cuando utilizando la herramienta .describe() podemos ver manualmente que no existen, asi que pensamos que tenemsos algo mal en el PCA.

In [25]:

```
outliers_detection=(outliers_detection-outliers_detection.mean())/outliers_detection.std()
```

In [26]:

```
trainpca,testpca = train_test_split(outliers_detection, test_size = 0.2,random_state=10).copy()
```

In [27]:

```
pca = PCA(n_components=23, random_state=100,whiten=False)
pcaNumModel = pca.fit(trainpca)
principalComponents = pcaNumModel.transform(trainpca)
```

In [28]:

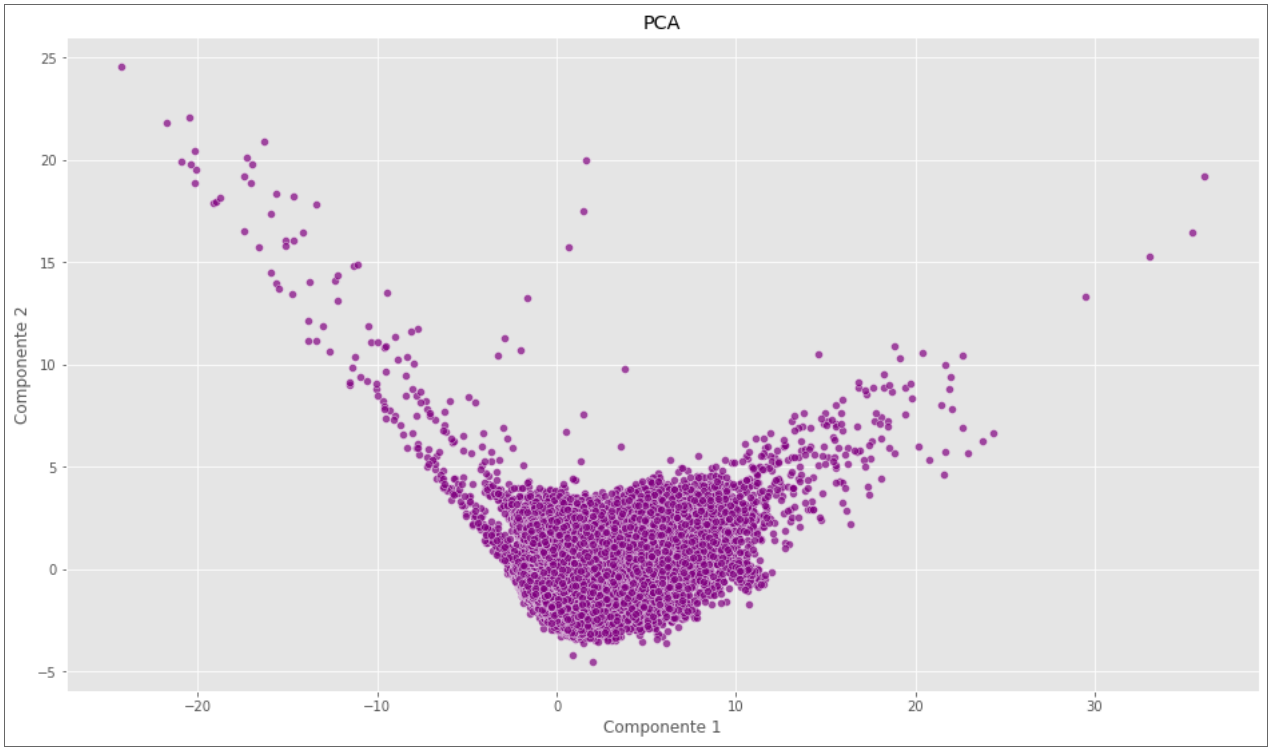
```
x_trans = principalComponents[:, 0]
y_trans = principalComponents[:, 1]
```

In [29]:

```
sns.scatterplot(x_trans, y_trans,
                sizes=(4,4),color='purple',alpha=0.7)
plt.title('PCA')
plt.xlabel('Componente 1')
plt.ylabel('Componente 2')
```

Out[29]:

```
Text(0, 0.5, 'Componente 2')
```



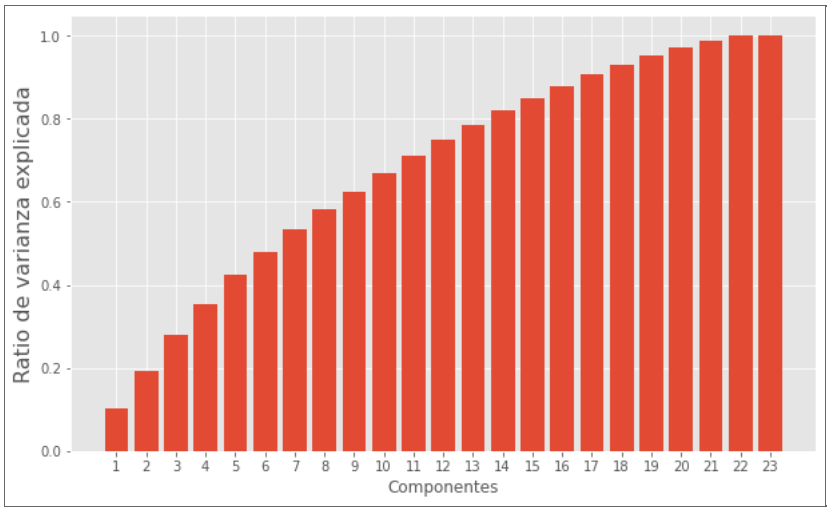

In [30]:

```
print(np.sum(pcaNumModel.explained_variance_ratio_))

ev1=np.cumsum(pcaNumModel.explained_variance_ratio_)

fig, ax = plt.subplots(figsize=(10, 6))

ax.bar(np.arange(1, len(ev1) + 1, 1), ev1)
ax.set_xticks(np.arange(1, len(ev1) + 1, 1))
ax.set_xlabel('Componentes')
ax.set_ylabel('Ratio de varianza explicada', fontsize=16)
```

```
0.9999999999999999
```

Out[30]:

```
Text(0, 0.5, 'Ratio de varianza explicada')
```




In [31]:

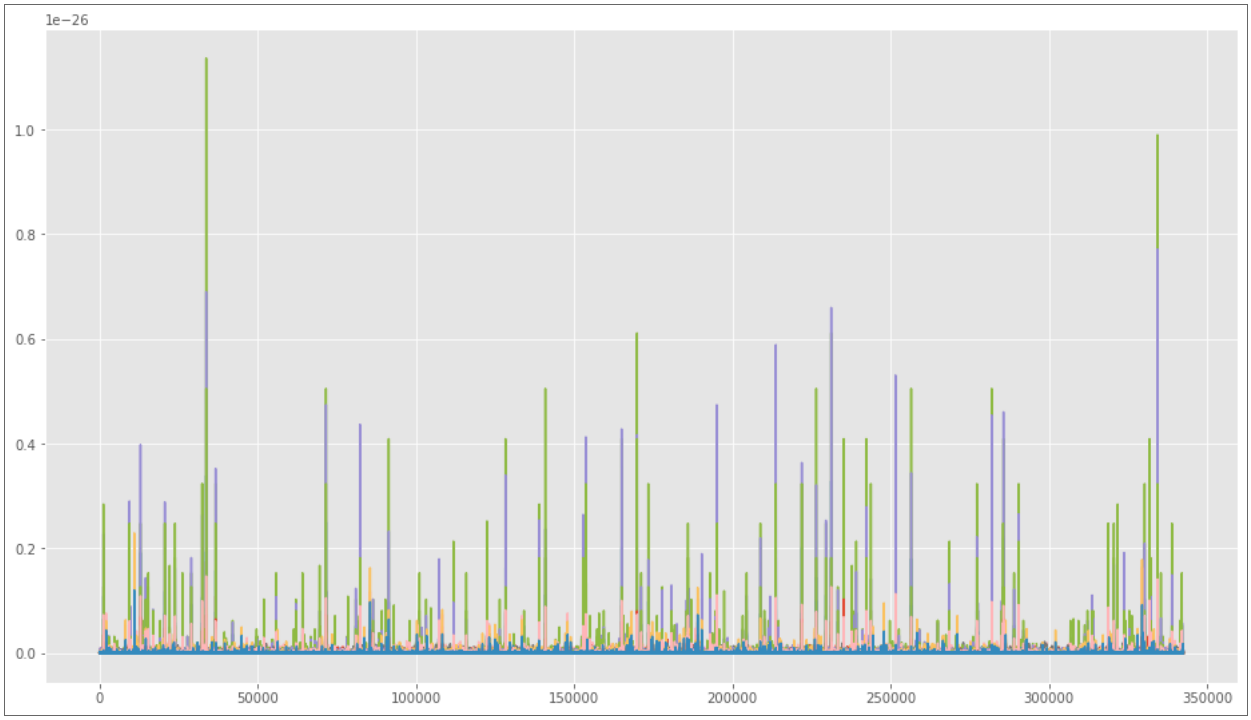
```
trainpca_numpy= trainpca.to_numpy()
train_inverse=pca.inverse_transform(principalComponents)
#MSE
mse_pca= (trainpca_numpy - train_inverse)**2

total=np.sum(mse_pca,1)

total_mean = np.mean(total)

total_std = np.std(total)

outlier= np.where(total > total_mean + 3*total_std)
```

In [32]:

```
plt.plot(mse_pca)
```

Out[32]:

```
[<matplotlib.lines.Line2D at 0x7fb9349812b0>,
 <matplotlib.lines.Line2D at 0x7fb9349812e0>,
 <matplotlib.lines.Line2D at 0x7fb934981460>,
 <matplotlib.lines.Line2D at 0x7fb934981520>,
 <matplotlib.lines.Line2D at 0x7fb9349815e0>,
 <matplotlib.lines.Line2D at 0x7fb9349816a0>,
 <matplotlib.lines.Line2D at 0x7fb934981760>,
 <matplotlib.lines.Line2D at 0x7fb934c996d0>,
 <matplotlib.lines.Line2D at 0x7fb9349818b0>,
 <matplotlib.lines.Line2D at 0x7fb934981970>,
 <matplotlib.lines.Line2D at 0x7fb934981a30>,
 <matplotlib.lines.Line2D at 0x7fb934981af0>,
 <matplotlib.lines.Line2D at 0x7fb934981bb0>,
 <matplotlib.lines.Line2D at 0x7fb934981c70>,
 <matplotlib.lines.Line2D at 0x7fb934981d30>,
 <matplotlib.lines.Line2D at 0x7fb934981df0>,
 <matplotlib.lines.Line2D at 0x7fb934981eb0>,
 <matplotlib.lines.Line2D at 0x7fb934981f70>,
 <matplotlib.lines.Line2D at 0x7fb934987070>,
 <matplotlib.lines.Line2D at 0x7fb934987130>,
 <matplotlib.lines.Line2D at 0x7fb9349871f0>,
 <matplotlib.lines.Line2D at 0x7fb9349872b0>,
 <matplotlib.lines.Line2D at 0x7fb934987370>]
```

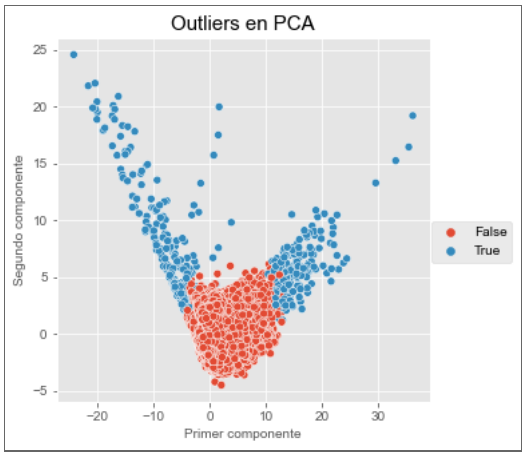



In [33]:

```
mark_outlier = np.where(total > total_mean + 3*total_std, True, False)
plt.subplots(figsize=(5,5))

sns.scatterplot(x=x_trans, y=y_trans, hue= mark_outlier)
plt.xlabel("Primer componente", fontsize = 10)
plt.ylabel("Segundo componente", fontsize = 10)
plt.title("Outliers en PCA", fontsize = 15)
plt.legend(bbox_to_anchor=(1, 0.5), loc=2, borderaxespad=0.1)
sns.set(rc={"figure.figsize":(8, 4)})
plt.figure(num=None, figsize=(10, 10), dpi=80, facecolor='r', edgecolor='k')
```

Out[33]:

```
<Figure size 800x800 with 0 Axes>
```




```
<Figure size 800x800 with 0 Axes>
```

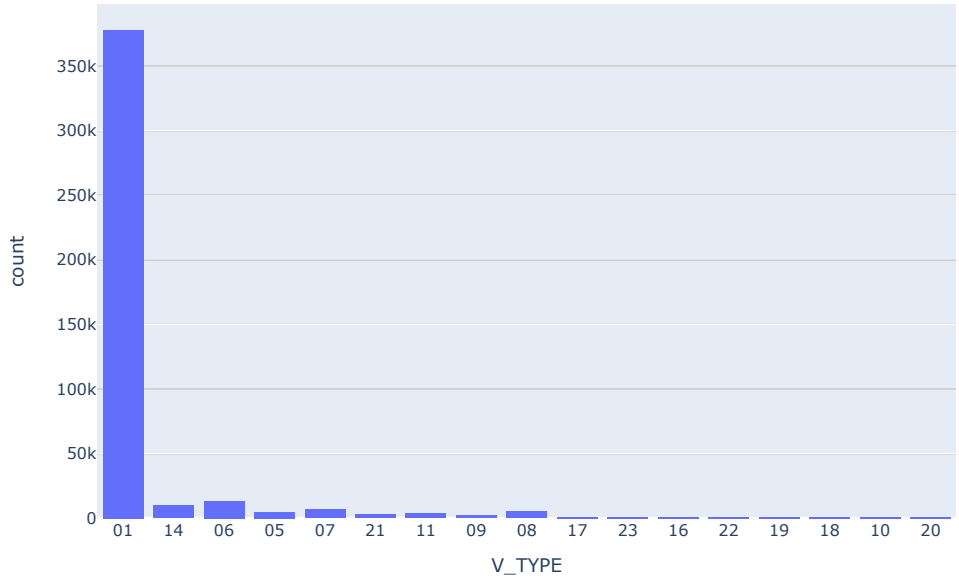
¿Qué importancia tiene la relación entre vehículos y conductores en la prima?

In [34]:

```
vehicles_info=data_clean_all[['V_TYPE', 'V_YEAR']].copy()
```

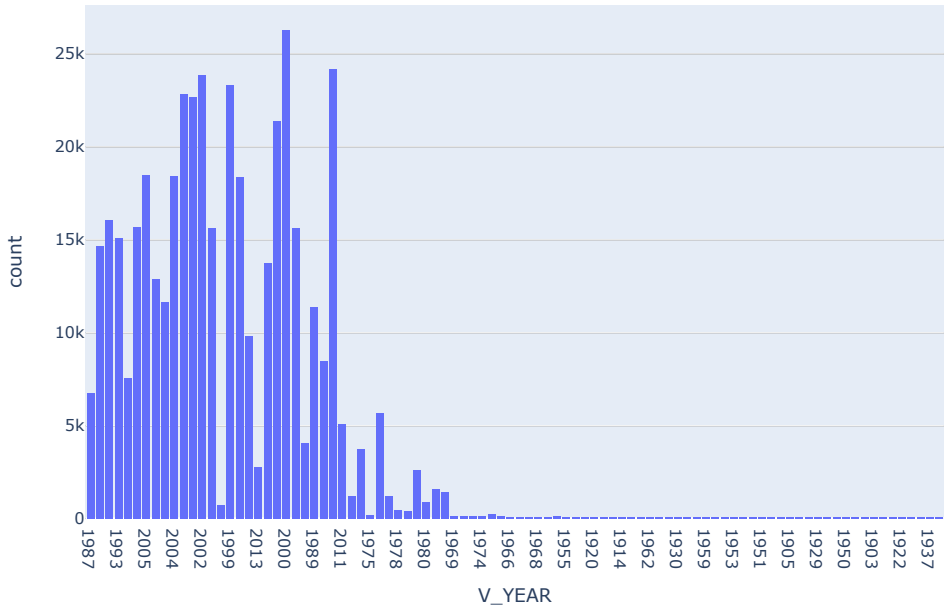
In [35]:

```
fig_v_type=px.histogram(vehicles_info,x="V_TYPE")
fig_v_type
```




In [36]:

```
fig_v_year=px.histogram(vehicles_info,x="V_YEAR")
fig_v_year
```

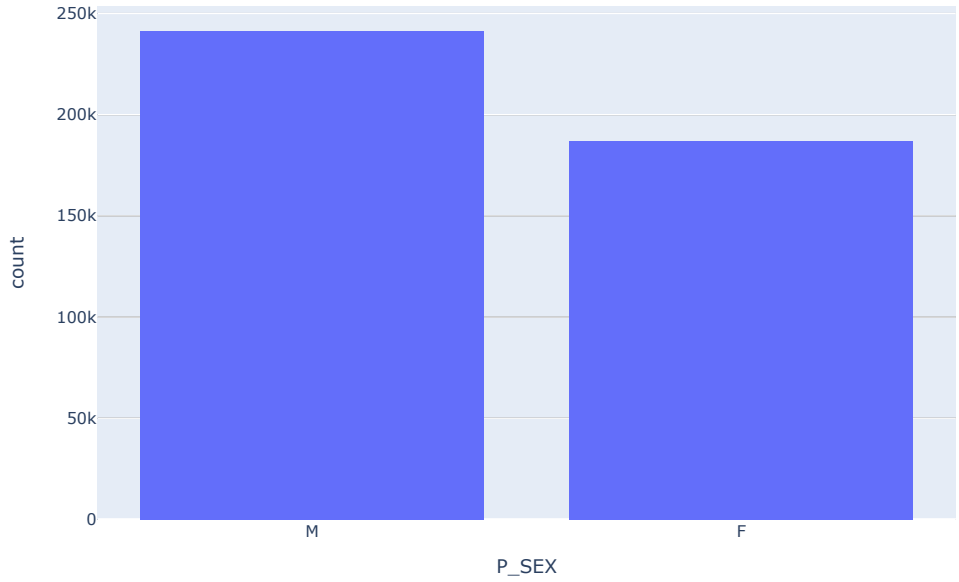



In [37]:

```
person_info=data_clean_all[['P_SEX','P_AGE','P_PSN','P_ISEV','P_SAFE','P_USER']].copy()
```

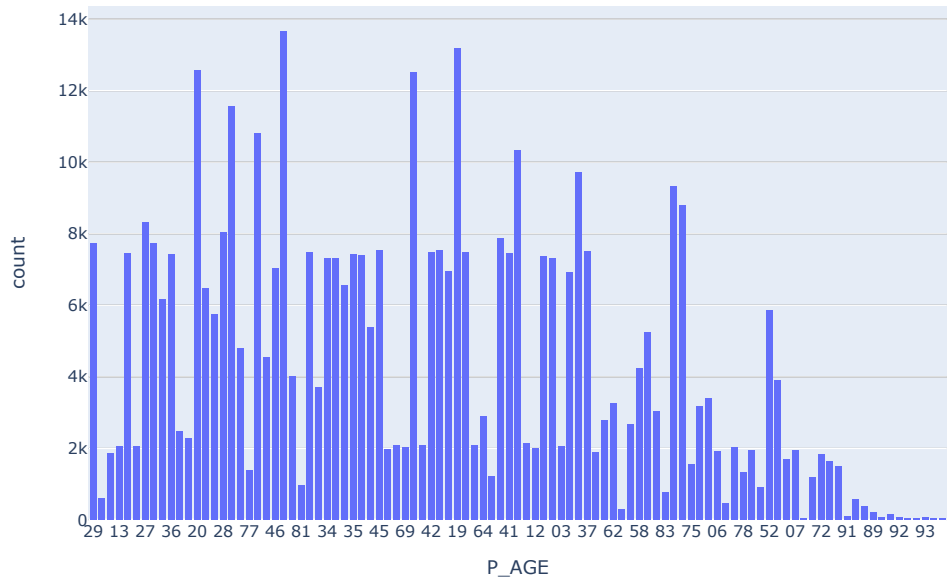
In [38]:

```
fig_v_type=px.histogram(person_info,x="P_SEX")
fig_v_type
```



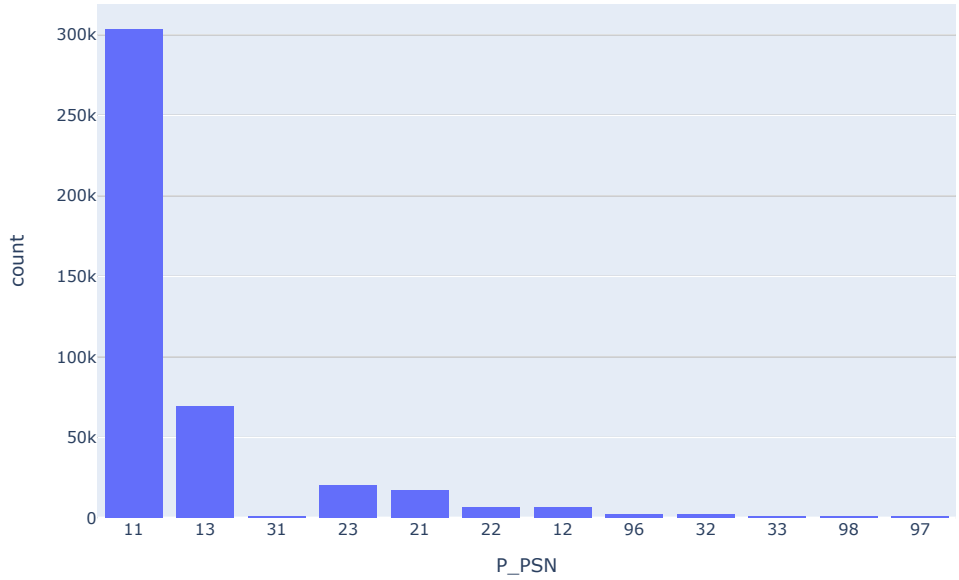

In [39]:

```
fig_v_type=px.histogram(person_info,x="P_AGE")
fig_v_type
```



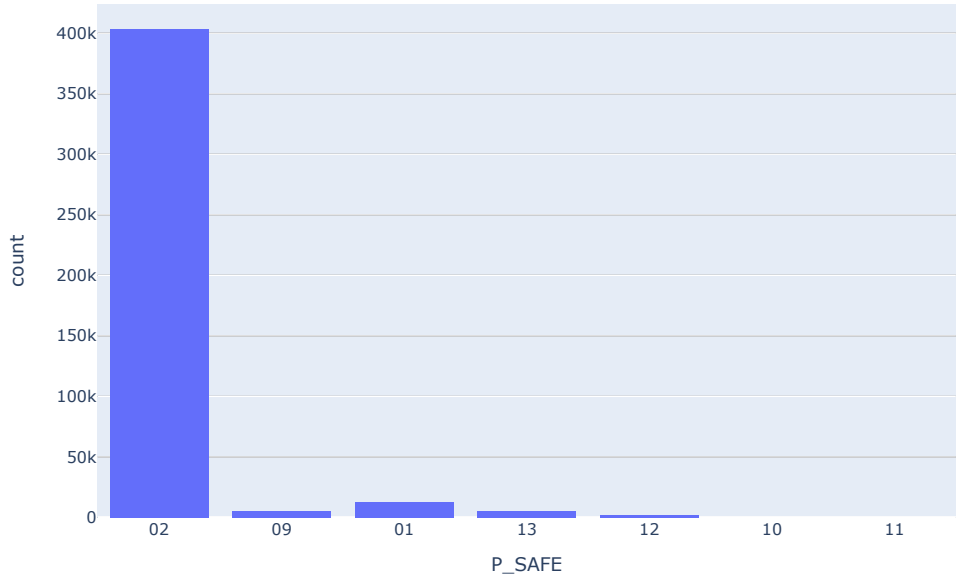

In [40]:

```
fig_v_type=px.histogram(person_info,x="P_PSN")
fig_v_type
```



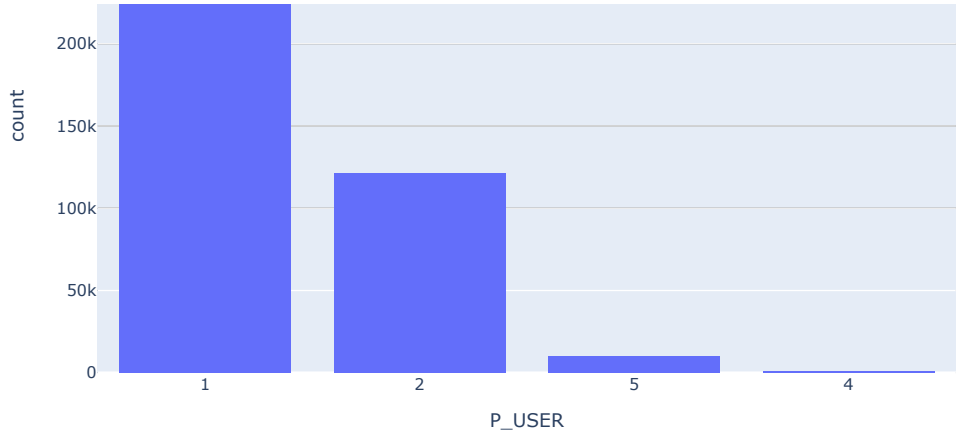

In [41]:

```
fig_v_type=px.histogram(person_info,x="P_SAFE")
fig_v_type
```




In [42]:

```
fig_v_type=px.histogram(person_info,x="P_USER")
fig_v_type
```




## ¿Qué tipos de vehículos (modelos, antigüedad, etc.) y conductores son más propensos a tener accidentes (acción correctiva en prima)?

Dentro de vehículos, el tipo 01 (turismo), y aquellos matriculados en el año 2000 es el año de los modelos. Respecto a los conductores, el sexo más común es el masculino, la edad son 18 años, la posición más común es el conductor, la medida de seguridad lo más común es que si que lleven safety device.

## ¿Qué tipos de vehículos (modelos, antigüedad, etc.) y conductores son menos propensos a tener accidentes (descuento en prima)?

Los vehículos menos propensos son los"fire engines" y el año 1970 para abajo . Respecto a los conductores, se trata del sexo femenino, la edad 97, con medidas de seguridad "otros", y el conductor menos común es el ciclista.

---

## Dado un accidente, ¿se puede generar un modelo que prediga si habrá fallecimientos o no? ¿Si se va a necesitar tratamiento médico o no? Las aseguradoras tienen que inmovilizar capital para pagar estas casuísticas.

### *Preparación de los datos*

Primero, seleccionamos las columnas que vamos a utilizar para el entrenamiento del modelo.

In [43]:

```
data_model=pd.DataFrame(data_clean_all,
                        columns= ['C_YEAR','C_MNTH', 'C_WDAY', 'C_HOUR', 'C_SEV', 'C_VEHS',
                                  'C_WTHR', 'C_RALN', 'V_TYPE','V_YEAR', 'P_AGE', 'P_SAFE', 'P_USER'])
```

In [44]:

```
data_model.dtypes
```

Out[44]:

```
C_YEAR      int64
C_MNTH     object
C_WDAY     object
C_HOUR     object
C_SEV       int64
C_VEHS     object
C_WTHR     object
C_RALN     object
V_TYPE     object
V_YEAR     object
P_AGE      object
P_SAFE     object
P_USER     object
dtype: object
```

In [45]:

```
data_model['C_YEAR'] = data_model['C_YEAR'].astype(int)
data_model['C_VEHS'] = data_model['C_VEHS'].astype(int)
data_model['C_VEHS'] = data_model['C_VEHS'].astype(int)
data_model['P_AGE'] = data_model['P_AGE'].astype(int)
data_model['V_YEAR'] = data_model['V_YEAR'].astype(int)
```

In [46]:

```
data_model['V_AGE']=data_model['C_YEAR']-data_model['V_YEAR']
```

In [47]:

```
data_model
```

Out[47]:

| | C_YEAR | C_MNTH | C_WDAY | C_HOUR | C_SEV | C_VEHS | C_WTHR | C_RALN | V_TYPE | V_YEAR | P_AGE | P_SAFE | P_USER | V_AGE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4124328 | 2009 | 8 | 2 | 16 | 2 | 2 | 1 | 5 | 01 | 1987 | 29 | 02 | 1 | 22 |
| 3824244 | 2008 | 9 | 3 | 16 | 2 | 2 | 1 | 1 | 01 | 2007 | 84 | 02 | 1 | 1 |
| 336701 | 1999 | 10 | 7 | 09 | 2 | 2 | 1 | 1 | 01 | 1996 | 71 | 02 | 1 | 3 |
| 788058 | 2000 | 11 | 6 | 15 | 2 | 2 | 1 | 4 | 01 | 1993 | 13 | 02 | 2 | 7 |
| 4949818 | 2012 | 1 | 7 | 17 | 2 | 2 | 1 | 1 | 01 | 2010 | 32 | 02 | 1 | 2 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1617196 | 2002 | 11 | 6 | 11 | 2 | 1 | 2 | 1 | 01 | 1993 | 70 | 02 | 1 | 9 |
| 5546889 | 2013 | 12 | 3 | 08 | 2 | 2 | 4 | 1 | 01 | 2003 | 17 | 02 | 1 | 10 |
| 2791590 | 2005 | 11 | 3 | 08 | 2 | 3 | 1 | 1 | 01 | 2002 | 75 | 02 | 1 | 3 |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4535196 | 2010 | 10 | 6 | 00 | 2 | 1 | 1 | 2 | 01 | 1995 | 19 | 02 | 1 | 15 |
| 4937641 | 2012 | 01 | 4 | 10 | 2 | 2 | 1 | 1 | 01 | 2006 | 20 | 02 | 1 | 6 |

428017 rows × 14 columns

In [48]:

```
data_model['id']=range(0,428017,1)
```

## Codificamos las variables categóricas

In [50]:

```
coder=OneHotEncoder()
```

In [51]:

```
var_categoricas=(data_model.dtypes[data_model.dtypes=='object']
                 .index.tolist())
```

In [52]:

```
coder.fit(data_model[var_categoricas])
```

Out[52]:

```
OneHotEncoder(cols=['C_MNTH', 'C_WDAY', 'C_HO
UR', 'C_WTHR', 'C_RALN', 'V_TYPE',
                    'P_SAFE', 'P_USER'])
```

In [53]:

```
res= coder.transform(data_model[var_categoricas])
```

In [54]:

```
res
```

Out[54]:

| | C_MNTH_1 | C_MNTH_2 | C_MNTH_3 | C_MNTH_4 | C_MNTH_5 | C_MNTH_6 | C_MNTH_7 | C_MNTH_8 | C_MNTH_9 | C_MNTH_10 | ... | P_SAFE_3 | P_SAF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4124328 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| 3824244 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| 336701 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| 788058 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| 4949818 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 1617196 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| 5546889 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | ... | 0 | |
| 2791590 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| 4535196 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| 4937641 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |

428017 rows × 100 columns

In [55]:

```
res['id']=range(0,428017,1)
```

In [56]:

```
res
```

Out[56]:

| | C_MNTH_1 | C_MNTH_2 | C_MNTH_3 | C_MNTH_4 | C_MNTH_5 | C_MNTH_6 | C_MNTH_7 | C_MNTH_8 | C_MNTH_9 | C_MNTH_10 | ... | P_SAFE_4 | P_SAF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4124328 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| 3824244 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| 336701 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| 788058 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| 4949818 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 1617196 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| 5546889 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | ... | 0 | |
| 2791590 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| 4535196 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| 4937641 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |

428017 rows × 101 columns

In [57]:

```
data_merged=pd.merge(data_model, res, on='id') #original con categoricas
data_merged
```

Out[57]:

| | C_YEAR | C_MNTH | C_WDAY | C_HOUR | C_SEV | C_VEHS | C_WTHR | C_RALN | V_TYPE | V_YEAR | ... | P_SAFE_3 | P_SAFE_4 | P_SAFE_5 | P_SAI |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2009 | 8 | 2 | 16 | 2 | 2 | 1 | 5 | 01 | 1987 | ... | 0 | 0 | 0 | |
| 1 | 2008 | 9 | 3 | 16 | 2 | 2 | 1 | 1 | 01 | 2007 | ... | 0 | 0 | 0 | |
| 2 | 1999 | 10 | 7 | 09 | 2 | 2 | 1 | 1 | 01 | 1996 | ... | 0 | 0 | 0 | |
| 3 | 2000 | 11 | 6 | 15 | 2 | 2 | 1 | 4 | 01 | 1993 | ... | 0 | 0 | 0 | |
| 4 | 2012 | 1 | 7 | 17 | 2 | 2 | 1 | 1 | 01 | 2010 | ... | 0 | 0 | 0 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 428012 | 2002 | 11 | 6 | 11 | 2 | 1 | 2 | 1 | 01 | 1993 | ... | 0 | 0 | 0 | |
| 428013 | 2013 | 12 | 3 | 08 | 2 | 2 | 4 | 1 | 01 | 2003 | ... | 0 | 0 | 0 | |
| 428014 | 2005 | 11 | 3 | 08 | 2 | 3 | 1 | 1 | 01 | 2002 | ... | 0 | 0 | 0 | |
| 428015 | 2010 | 10 | 6 | 00 | 2 | 1 | 1 | 2 | 01 | 1995 | ... | 0 | 0 | 0 | |
| 428016 | 2012 | 01 | 4 | 10 | 2 | 2 | 1 | 1 | 01 | 2006 | ... | 0 | 0 | 0 | |

## 428017 rows × 115 columns

In [58]:

```
data_merged=data_merged.drop(columns=['C_MNTH', 'C_WDAY', 'C_HOUR', 'C_WTHR', 'C_RALN', 'V_TYPE','P_SAFE
', 'P_USER'])
```

In [59]:

```
data_merged=data_merged.drop(columns=['id'])
```

In [60]:

```
data_model=data_merged
```

In [126]:

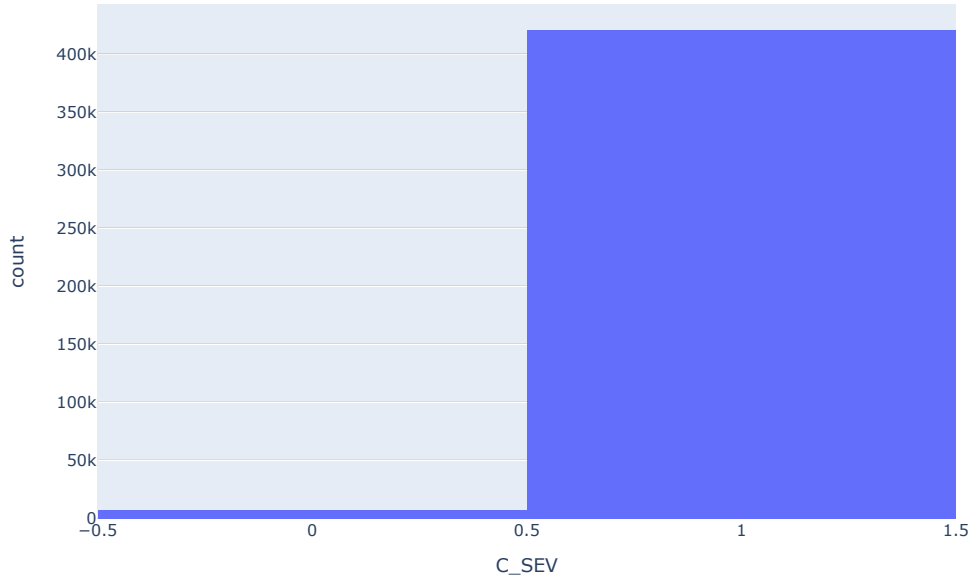
```
data_model=data_model.drop(columns=['C_YEAR'])
```

In [61]:

```
data_model['C_SEV']=np.where(data_model['C_SEV']==1,0,1)
```

In [112]:

```
a=px.histogram(data_model,x="C_SEV")
a
```




---

## Creamos los datasets de train y test, estratificando la variable objetivo para que esté de manera equilibrada.

In [127]:

```
X_pd_loan, X_pd_loan_test, y_pd_loan, y_pd_loan_test = train_test_split(data_model.drop('C_SEV',axis=1),
                                                                        data_model['C_SEV'],
                                                                        stratify=data_model['C_SEV'],
                                                                        test_size=0.2)
data_train = pd.concat([X_pd_loan, y_pd_loan],axis=1)
data_test = pd.concat([X_pd_loan_test, y_pd_loan_test],axis=1)
```

In [128]:

```
print('== Train\n', data_train['C_SEV'].value_counts(normalize=True))
print('== Test\n', data_test['C_SEV'].value_counts(normalize=True))
```

```
== Train
 1    0.982947
0    0.017053
Name: C_SEV, dtype: float64
== Test
 1    0.982945
0    0.017055
Name: C_SEV, dtype: float64
```

In [129]:

```
X_train= data_train.drop('C_SEV',axis=1)
X_test= data_test.drop('C_SEV',axis=1)
y_train= data_train['C_SEV']
y_test= data_test['C_SEV']
```

Para la comprobación de los modelos, vamos a utilizar una matriz que enseña las métricas(accuracy,precisión, recuperación, F1 y soporte del modelo). Para ello, adjuntamos aquí debajo una explicación de las métricas. Por otro lado calculamos la matriz de confusión, la curva ROC y el área bajo la curva.

La precisión es la capacidad de un clasificador de no etiquetar como positiva una instancia que en realidad es negativa. Para cada clase, se define como la relación entre los verdaderos positivos y la suma de un verdadero positivo y un falso positivo.
La recuperación es la capacidad de un clasificador para encontrar todos los casos positivos. Para cada clase, se define como la relación entre los verdaderos positivos y la suma de los verdaderos positivos y los falsos negativos.
La puntuación F1 es una media armónica ponderada de la precisión y la recuperación, de modo que la mejor puntuación es 1,0 y la peor es 0,0. Las puntuaciones F1 son más bajas que las medidas de exactitud, ya que incluyen la precisión y la recuperación en su cálculo. Como regla general, la media ponderada de F1 debe utilizarse para comparar los modelos de clasificación, no la precisión global.
El apoyo es el número de ocurrencias reales de la clase en el conjunto de datos especificado. Un soporte desequilibrado en los datos de entrenamiento puede indicar debilidades estructurales en las puntuaciones reportadas por el clasificador y podría indicar la necesidad de un muestreo estratificado o un rebalanceo. El soporte no cambia entre modelos, sino que diagnostica el proceso de evaluación.
La matriz de confusión representa la precisión del modelo en formato tabular

mediante la representación del recuento de etiquetas correctas/incorrectas:

- TN / True Negative: el caso era negativo y se predijo negativo
- TP / Verdadero positivo: el caso era positivo y se predijo positivo
- FN / Falso negativo: el caso fue positivo pero se predijo negativo
- FP / Falso Positivo: el caso fue negativo pero se predijo positivo

La curva AUC-ROC es la métrica de selección del modelo para el problema de clasificación biclase. ROC es una curva de probabilidad para diferentes clases. La ROC nos indica lo bueno que es el modelo para distinguir las clases dadas, en términos de la probabilidad predicha.
Una curva ROC típica tiene la tasa de falsos positivos (FPR) en el eje X y la tasa de verdaderos positivos (TPR) en el eje Y. El área cubierta por la curva es el área entre la línea naranja (ROC) y el eje. Esta área cubierta es el AUC. Cuanto mayor sea el área cubierta, mejor es el modelo de aprendizaje automático para distinguir las clases dadas. El valor ideal de AUC es 1.

## ÁRBOL DE DECISIÓN

In [130]:

```
# Creación del modelo
# ------------------------------------------------------------------------------
modelo = DecisionTreeRegressor(
            max_depth         = 6,
            random_state      = 123
         )

# Entrenamiento del modelo
# ------------------------------------------------------------------------------
modelo.fit(X_train, y_train)
```

Out[130]:

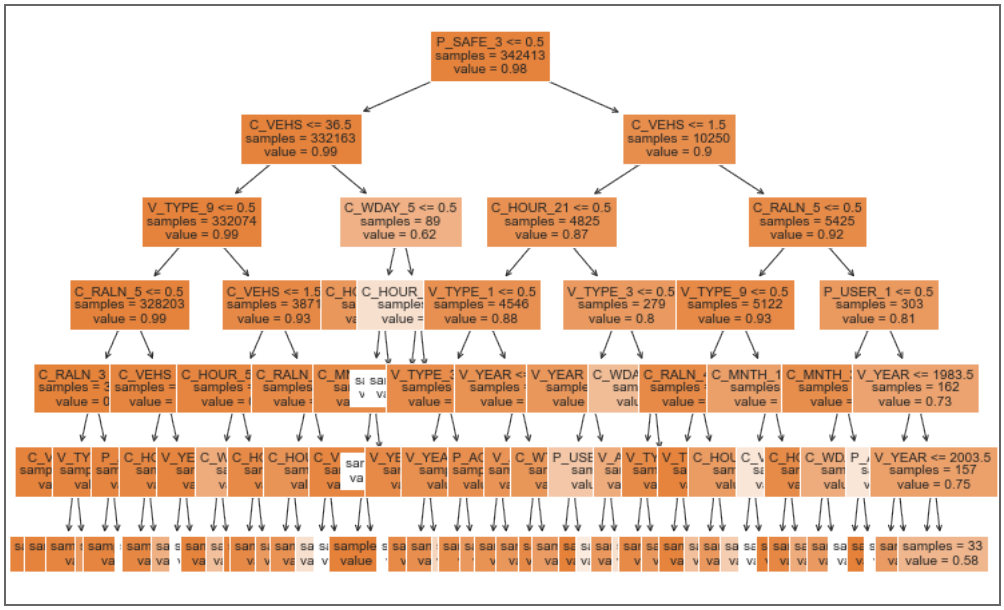
```
DecisionTreeRegressor(max_depth=6, random_state=123)
```

In [131]:

```
# Estructura del árbol creado
# ------------------------------------------------------------------------------
fig, ax = plt.subplots(figsize=(12, 8))

print(f"Profundidad del árbol: {modelo.get_depth()}")
print(f"Número de nodos terminales: {modelo.get_n_leaves()}")

plot= plot_tree (decision_tree = modelo,
            feature_names = data_model.drop(columns = "C_SEV").columns,
            class_names   = 'C_SEV',
            filled        = True,
            impurity      = False,
            fontsize      = 10,
            precision     = 2,
            ax            = ax
        )
```

```
Profundidad del árbol: 6
```

Número de nodos terminales: 53




Vemos que los primeros nodos se separan con el uso de medios de seguridad, seguido de el número de vehículos involucrados en el accidente.

In [132]:

```
texto_modelo = export_text( decision_tree = modelo, feature_names = list(
    data_model.drop(columns = "C_SEV").columns))
print(texto_modelo)
```

```
|--- P_SAFE_3 <= 0.50
|    |--- C_VEHS <= 36.50
|    |    |--- V_TYPE_9 <= 0.50
|    |    |    |--- C_RALN_5 <= 0.50
|    |    |    |    |--- C_RALN_3 <= 0.50
|    |    |    |    |    |--- C_VEHS <= 1.50
|    |    |    |    |    |    |--- value: [0.98]
|    |    |    |    |    |--- C_VEHS >  1.50
|    |    |    |    |    |    |--- value: [0.99]
|    |    |    |    |--- C_RALN_3 >  0.50
|    |    |    |    |    |--- V_TYPE_7 <= 0.50
|    |    |    |    |    |    |--- value: [0.97]
|    |    |    |    |    |--- V_TYPE_7 >  0.50
|    |    |    |    |    |    |--- value: [0.79]
|    |    |    |--- C_RALN_5 >  0.50
|    |    |    |    |--- C_VEHS <= 2.50
|    |    |    |    |    |--- P_AGE <= 96.50
```

```
|   |   |   |   |   |   |--- value: [0.97]
|   |   |   |   |   |--- P_AGE >  96.50
|   |   |   |   |   |   |--- value: [0.00]
|   |   |   |   |--- C_VEHS >  2.50
|   |   |   |   |   |--- C_HOUR_25 <= 0.50
|   |   |   |   |   |   |--- value: [0.95]
|   |   |   |   |   |--- C_HOUR_25 >  0.50
|   |   |   |   |   |   |--- value: [0.62]
|   |   |--- V_TYPE_9 >  0.50
|   |   |   |--- C_VEHS <= 1.50
|   |   |   |   |--- C_HOUR_5 <= 0.50
|   |   |   |   |   |--- V_YEAR <= 1942.50
|   |   |   |   |   |   |--- value: [0.00]
|   |   |   |   |   |--- V_YEAR >  1942.50
|   |   |   |   |   |   |--- value: [0.97]
|   |   |   |   |--- C_HOUR_5 >  0.50
|   |   |   |   |   |--- C_WDAY_6 <= 0.50
|   |   |   |   |   |   |--- value: [0.50]
|   |   |   |   |   |--- C_WDAY_6 >  0.50
|   |   |   |   |   |   |--- value: [1.00]
|   |   |   |--- C_VEHS >  1.50
|   |   |   |   |--- C_RALN_3 <= 0.50
|   |   |   |   |   |--- C_HOUR_18 <= 0.50
|   |   |   |   |   |   |--- value: [0.93]
|   |   |   |   |   |--- C_HOUR_18 >  0.50
|   |   |   |   |   |   |--- value: [0.81]
|   |   |   |   |--- C_RALN_3 >  0.50
|   |   |   |   |   |--- C_HOUR_21 <= 0.50
|   |   |   |   |   |   |--- value: [0.86]
|   |   |   |   |   |--- C_HOUR_21 >  0.50
|   |   |   |   |   |   |--- value: [0.25]
|   |--- C_VEHS >  36.50
|   |   |--- C_WDAY_5 <= 0.50
|   |   |   |--- C_HOUR_3 <= 0.50
|   |   |   |   |--- C_MNTH_2 <= 0.50
|   |   |   |   |   |--- C_VEHS <= 37.50
|   |   |   |   |   |   |--- value: [0.00]
```

```
|    |    |    |    |    |--- C_VEHS >  37.50
|    |    |    |    |    |    |--- value: [1.00]
|    |    |    |    |    |--- C_MNTH_2 >  0.50
|    |    |    |    |    |    |--- value: [0.00]
|    |    |    |    |--- C_HOUR_3 >  0.50
|    |    |    |    |    |--- value: [0.00]
|    |    |    |--- C_WDAY_5 >  0.50
|    |    |    |    |--- C_HOUR_3 <= 0.50
|    |    |    |    |    |--- value: [0.00]
|    |    |    |    |--- C_HOUR_3 >  0.50
|    |    |    |    |    |--- value: [1.00]
|    |--- P_SAFE_3 >  0.50
|    |    |--- C_VEHS <= 1.50
|    |    |    |--- C_HOUR_21 <= 0.50
|    |    |    |    |--- V_TYPE_1 <= 0.50
|    |    |    |    |    |--- V_TYPE_3 <= 0.50
|    |    |    |    |    |    |--- V_YEAR <= 1970.50
|    |    |    |    |    |    |    |--- value: [0.00]
|    |    |    |    |    |    |--- V_YEAR >  1970.50
|    |    |    |    |    |    |    |--- value: [0.93]
|    |    |    |    |    |--- V_TYPE_3 >  0.50
|    |    |    |    |    |    |--- V_YEAR <= 2006.50
|    |    |    |    |    |    |    |--- value: [0.89]
|    |    |    |    |    |    |--- V_YEAR >  2006.50
|    |    |    |    |    |    |    |--- value: [0.33]
|    |    |    |    |--- V_TYPE_1 >  0.50
|    |    |    |    |    |--- V_YEAR <= 1993.50
|    |    |    |    |    |    |--- P_AGE <= 50.50
|    |    |    |    |    |    |    |--- value: [0.90]
|    |    |    |    |    |    |--- P_AGE >  50.50
|    |    |    |    |    |    |    |--- value: [0.82]
|    |    |    |    |    |--- V_YEAR >  1993.50
|    |    |    |    |    |    |--- V_AGE <= 8.50
|    |    |    |    |    |    |    |--- value: [0.87]
|    |    |    |    |    |    |--- V_AGE >  8.50
|    |    |    |    |    |    |    |--- value: [0.82]
|    |    |    |--- C_HOUR_21 >  0.50
```

```
|   |   |   |   |--- V_TYPE_3 <= 0.50
|   |   |   |   |   |--- V_YEAR <= 2007.50
|   |   |   |   |   |   |--- C_WTHR_1 <= 0.50
|   |   |   |   |   |   |   |--- value: [0.93]
|   |   |   |   |   |   |--- C_WTHR_1 >  0.50
|   |   |   |   |   |   |   |--- value: [0.78]
|   |   |   |   |   |--- V_YEAR >  2007.50
|   |   |   |   |   |   |--- P_USER_1 <= 0.50
|   |   |   |   |   |   |   |--- value: [1.00]
|   |   |   |   |   |   |--- P_USER_1 >  0.50
|   |   |   |   |   |   |   |--- value: [0.17]
|   |   |   |   |--- V_TYPE_3 >  0.50
|   |   |   |   |   |--- C_WDAY_10 <= 0.50
|   |   |   |   |   |   |--- V_AGE <= 15.50
|   |   |   |   |   |   |   |--- value: [0.82]
|   |   |   |   |   |   |--- V_AGE >  15.50
|   |   |   |   |   |   |   |--- value: [0.25]
|   |   |   |   |   |--- C_WDAY_10 >  0.50
|   |   |   |   |   |   |--- value: [0.00]
|   |   |--- C_VEHS >  1.50
|   |   |   |--- C_RALN_5 <= 0.50
|   |   |   |   |--- V_TYPE_9 <= 0.50
|   |   |   |   |   |--- C_RALN_4 <= 0.50
|   |   |   |   |   |   |--- V_TYPE_5 <= 0.50
|   |   |   |   |   |   |   |--- value: [0.93]
|   |   |   |   |   |   |--- V_TYPE_5 >  0.50
|   |   |   |   |   |   |   |--- value: [0.81]
|   |   |   |   |   |--- C_RALN_4 >  0.50
|   |   |   |   |   |   |--- V_TYPE_3 <= 0.50
|   |   |   |   |   |   |   |--- value: [0.98]
|   |   |   |   |   |   |--- V_TYPE_3 >  0.50
|   |   |   |   |   |   |   |--- value: [0.50]
|   |   |   |   |--- V_TYPE_9 >  0.50
|   |   |   |   |   |--- C_MNTH_11 <= 0.50
|   |   |   |   |   |   |--- C_HOUR_20 <= 0.50
|   |   |   |   |   |   |   |--- value: [0.82]
|   |   |   |   |   |   |--- C_HOUR_20 >  0.50
```

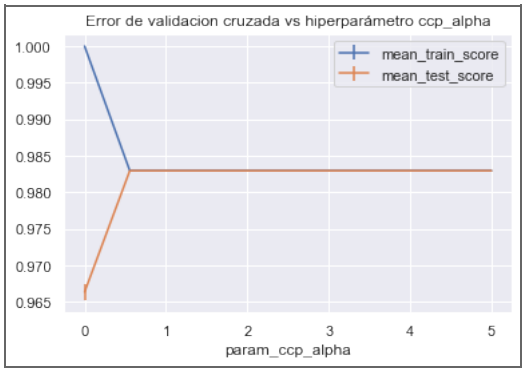
```
|   |   |   |   |   |   |--- value: [0.40]
|   |   |   |   |   |--- C_MNTH_11 >  0.50
|   |   |   |   |   |   |--- C_VEHS <= 3.00
|   |   |   |   |   |   |   |--- value: [0.00]
|   |   |   |   |   |   |--- C_VEHS >  3.00
|   |   |   |   |   |   |   |--- value: [1.00]
|   |   |--- C_RALN_5 >  0.50
|   |   |   |--- P_USER_1 <= 0.50
|   |   |   |   |--- C_MNTH_3 <= 0.50
|   |   |   |   |   |--- C_HOUR_1 <= 0.50
|   |   |   |   |   |   |--- value: [0.95]
|   |   |   |   |   |--- C_HOUR_1 >  0.50
|   |   |   |   |   |   |--- value: [0.67]
|   |   |   |   |--- C_MNTH_3 >  0.50
|   |   |   |   |   |--- C_WDAY_4 <= 0.50
|   |   |   |   |   |   |--- value: [0.80]
|   |   |   |   |   |--- C_WDAY_4 >  0.50
|   |   |   |   |   |   |--- value: [0.00]
|   |   |   |--- P_USER_1 >  0.50
|   |   |   |   |--- V_YEAR <= 1983.50
|   |   |   |   |   |--- P_AGE <= 29.50
|   |   |   |   |   |   |--- value: [1.00]
|   |   |   |   |   |--- P_AGE >  29.50
|   |   |   |   |   |   |--- value: [0.00]
|   |   |   |   |--- V_YEAR >  1983.50
|   |   |   |   |   |--- V_YEAR <= 2003.50
|   |   |   |   |   |   |--- value: [0.79]
|   |   |   |   |   |--- V_YEAR >  2003.50
|   |   |   |   |   |   |--- value: [0.58]
```

Realizamos un podado para identificar la profunidad óptima y otros hiperparámetros, aumentando la capacidad predictiva y reduciendo la varianza.

In [114]:

```
# Post pruning (const complexity pruning) por validación cruzada
# ------------------------------------------------------------------------------
# Valores de ccp_alpha evaluados
param_grid = {'ccp_alpha':np.linspace(0, 5, 10)}

# Búsqueda por validación cruzada
grid = GridSearchCV(
```

```
        # El árbol se crece al máximo posible antes de aplicar el pruning
        estimator = DecisionTreeClassifier(
                            max_depth         = None,
                            min_samples_split = 2,
                            min_samples_leaf  = 1,
                            random_state      = 123
                        ),
        param_grid = param_grid,
        scoring    = 'accuracy',
        cv         = 10,
        refit      = True,
        return_train_score = True
      )

grid.fit(X_train, y_train)

fig, ax = plt.subplots(figsize=(6, 3.84))
scores = pd.DataFrame(grid.cv_results_)
scores.plot(x='param_ccp_alpha', y='mean_train_score', yerr='std_train_score', ax=ax)
scores.plot(x='param_ccp_alpha', y='mean_test_score', yerr='std_test_score', ax=ax)
ax.set_title("Error de validacion cruzada vs hiperparámetro ccp_alpha");
```




In [115]:

```
grid.best_params_
```

Out[115]:

```
{'ccp_alpha': 0.5555555555555556}
```

In [116]:

```
# Estructura del árbol final
# ------------------------------------------------------------------------------
modelo_final = grid.best_estimator_
print(f"Profundidad del árbol: {modelo_final.get_depth()}")
print(f"Número de nodos terminales: {modelo_final.get_n_leaves()}")
```

```
Profundidad del árbol: 0
Número de nodos terminales: 1
```

In [118]:

```
# Error de test del modelo final
#-------------------------------------------------------------------------------
predicciones = modelo_final.predict(X = X_test)

accuracy = accuracy_score(
            y_true    = y_test,
            y_pred    = predicciones,
            normalize = True
           )
print(f"El accuracy de test es: {100 * accuracy} %")
```

```
El accuracy de test es: 98.29447222092426 %
```

Vemos que trás el podado, nuestra accuracy es mayor.

### Grandient BOOSTING

Utilizamos tanto arbol de decisión como el embedding gradient boosting. Esto lo hacemos para poder ver el variable importance que también se pide. A parte, sabemos que usando estos ensambladores, obtenemos mejores resultados.

In [76]:

```
boosting = GradientBoostingClassifier(n_estimators=100)
boosting.fit(X_train, y_train)
```

Out[76]:

```
GradientBoostingClassifier()
```

In [77]:

```
predictions_gb=boosting.predict(X_test)
predictions_gb

print(confusion_matrix(y_test, predictions_gb))
```

```
[[    8  1452]
 [   11 84133]]
```

Podemos ver que en nuestro modelo, funciona mucho mejor para los true positives que para false negatives. Es decir, identifica mejor cuando no va a fallecer alguien que cuando alguien si.

### Metrics

In [90]:

```
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix, accuracy_score, r
oc_auc_score
```

In [113]:

```
print(classification_report(y_test, predictions_gb))
```

```
              precision    recall  f1-score
support

           0       0.42      0.01      0.01
1460
           1       0.98      1.00      0.99
```

```
84144

    accuracy                           0.98
85604
   macro avg       0.70      0.50      0.50
85604
weighted avg       0.97      0.98      0.97
85604
```

Como hemos visto en la matriz de confusión, nuestro modelo clasifica mejor aquellos accidentes en los cuales no hay fallecidos. Creemos que uno de los motivos es que la proporción de no fallecidos es mayor, con lo cual el modelo tiene más datos sobre los que aprender. La precisión media es de 0.7, que indica el numero total de aciertos de positivos y negativos. El recall medio es de 0.5 porque hay pleno de aciertos en no fallecidos pero para accidentes con fallecimientos solo hay un 1%. El f-1 score de media ponderada nos indica un 0.97 de valor, con lo cual sabemos que nace de un acierto muy alto de los accidentes sin fallecidos y pocos aciertos en los fallecimientos.

## Curva ROC

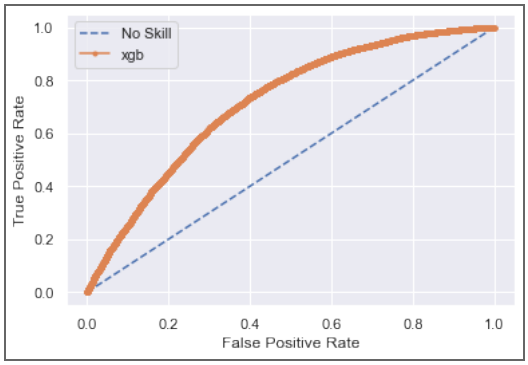
In [120]:

```
pred_test_gb = clf.predict_proba(X_test)
pred_train_gb = clf.predict_proba(X_train)
```

## Para train

In [121]:

```
# keep probabilities for the positive outcome only
yhat_train_gb = pred_train_gb[:, 1]
# calculate roc curves
fpr, tpr, thresholds = roc_curve(y_train, yhat_train_gb)
# plot the roc curve for the model
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')
plt.plot(fpr, tpr, marker='.', label='xgb')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
# show the plot
plt.show()
```




In [122]:

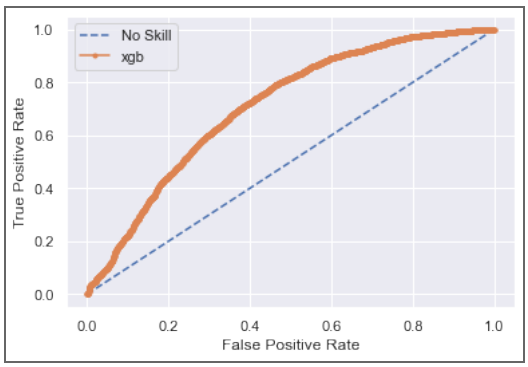
```
# calculate AUC
auc = roc_auc_score(y_train, yhat_train_gb)
print('AUC: %.3f' % auc)
```

```
AUC: 0.718
```

## Para test

In [124]:

```
# keep probabilities for the positive outcome only
yhat_gb = pred_test_gb[:, 1]
# calculate roc curves
fpr, tpr, thresholds = roc_curve(y_test, yhat_gb)
# plot the roc curve for the model
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')
plt.plot(fpr, tpr, marker='.', label='xgb')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
# show the plot
plt.show()
```




In [125]:

```
# calculate AUC
auc = roc_auc_score(y_test, yhat_gb)
print('AUC: %.3f' % auc)
```

```
AUC: 0.711
```

# REGRESIÓN LOGÍSTICA

In [95]:

```
scaler = StandardScaler()
scaler_train = scaler.fit(data_train)
```

In [97]:

```
clf = LogisticRegression(random_state=100)
model_glm = clf.fit(X_train, y_train)
```

In [98]:

```
#Modelo
predictions_rl = model_glm.predict(X_test)
```

In [99]:

```
print(confusion_matrix(y_test, predictions_rl))
```

```
[[    2  1458]
 [    3 84141]]
```

Como podemos ver, hay un numero de true positives muy elevado respecto a los falsos positivos y falsos negativos. Con lo cual el modelo es muy acertado. Sabemos que es verdad que no se puede exigir a un modelo funcionar al 100%, con lo cual esperabamos algunos fallos. En comparación, vemos que la regresión logística, acierta mas los casos de True positives, y en el boosting, true negatives. Sin embargo, ambos son similares.

In [100]:

```
confmat_test_rl  = confusion_matrix(y_test, predictions_rl)
```

In [101]:

```
print(classification_report(y_test, predictions_rl))
```

```
              precision    recall  f1-score   support

           0       0.40      0.00      0.00      1460
           1       0.98      1.00      0.99     84144

    accuracy                           0.98     85604
   macro avg       0.69      0.50      0.50     85604
weighted avg       0.97      0.98      0.97     85604
```

```
85604
```

Respecto a la precisión, tiene una media de 0.69, la cual es más baja que la del árbol. Esto es por lo comentado anteriormente, ya que acierta menos fallecimientos. El recall medio es de 0.5 porque hay pleno de aciertos en no fallecidos pero para accidentes con fallecimientos solo hay un 1%. El f-1 score de media ponderada nos indica un 0.97 de valor, con lo cual sabemos que nace de un acierto muy alto de los accidentes sin fallecidos y pocos aciertos en los fallecimientos.

## Curva ROC

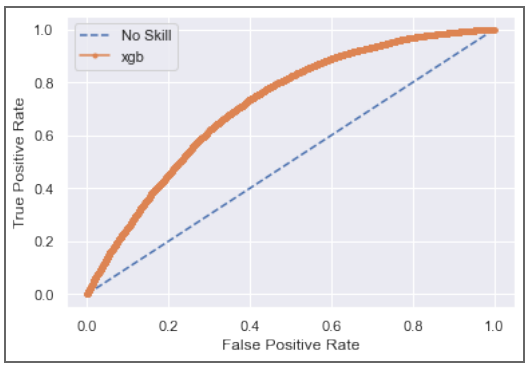
In [103]:

```
pred_test_rl = clf.predict_proba(X_test)
pred_train_rl = clf.predict_proba(X_train)
```

## Para train

In [104]:

```
# keep probabilities for the positive outcome only
yhat_train_rl = pred_train_rl[:, 1]
# calculate roc curves
fpr, tpr, thresholds = roc_curve(y_train, yhat_train_rl)
# plot the roc curve for the model
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')
plt.plot(fpr, tpr, marker='.', label='xgb')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
# show the plot
plt.show()
```




In [105]:

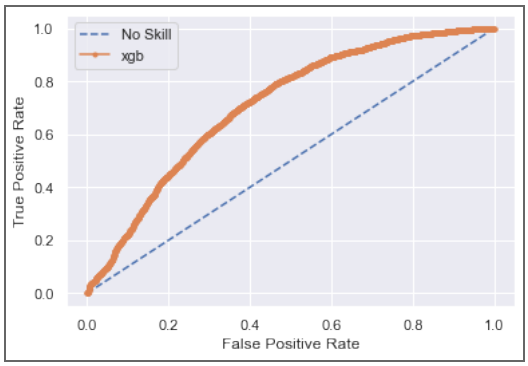
```
# calculate AUC
auc = roc_auc_score(y_train, yhat_train_rl)
print('AUC: %.3f' % auc)
```

```
AUC: 0.718
```

## Y para test

In [106]:

```
# keep probabilities for the positive outcome only
yhat_rl = pred_test_rl[:, 1]
# calculate roc curves
fpr, tpr, thresholds = roc_curve(y_test, yhat_rl)
# plot the roc curve for the model
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')
plt.plot(fpr, tpr, marker='.', label='xgb')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
# show the plot
plt.show()
```




In [107]:

```
# calculate AUC
auc = roc_auc_score(y_test, yhat_rl)
print('AUC: %.3f' % auc)
```

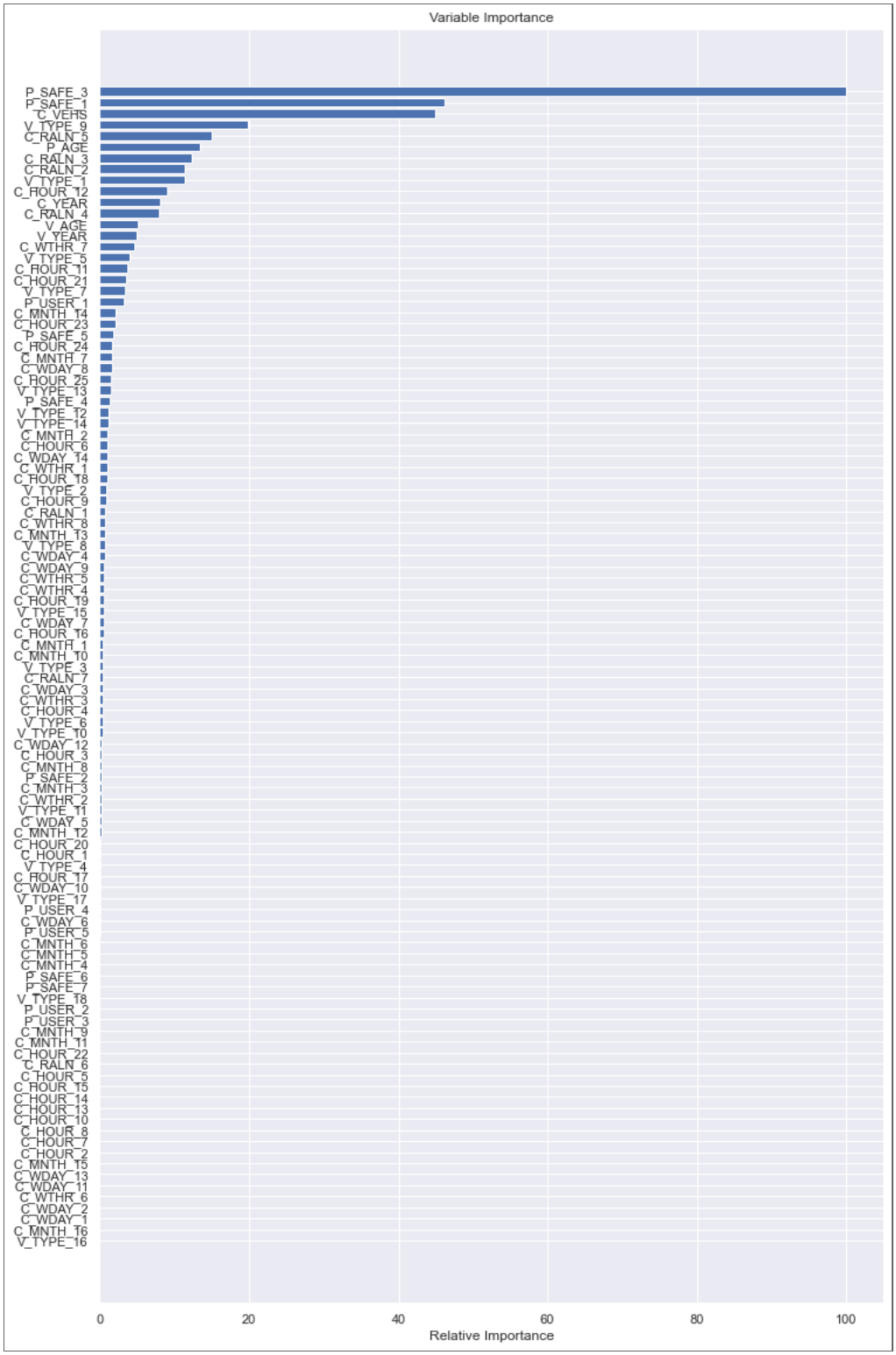
```
AUC: 0.711
```

## ¿QUÉ ES LO QUE MÁS CONTRIBUYE A QUE EXISTAN FALLECIMIENTOS EN UN ACCIDENTE?

## Variable importance

In [89]:

```
feature_importance = boosting.feature_importances_
# make importances relative to max importance
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
# plt.subplot(1, 2, 2)
plt.figure(figsize=(12, 20))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X_train.keys()[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```




Con esta gráfica, podemos ver cuáles son las variables que más afectan a que haya un fallecimiento en un accidente. Las primeras dos hacen referencia a la medida de seguridad tomada por las personas, primero el uso de casco y segundo el no usar cinturón/ silla para niños. EL tercer componente que más influye es el número de vehículos involucrados en el accidente. Después de esto podemos enontrar otras variables de menor importancia como los meses, días u horas. Esto significa que no son variables especialmente determinantes a la hora de predecir la severidad de un accidente.

## Conclusión

En conclusión, sabemos que hemos tenido que trabajar con un dataset muy grande. Esto tiene ventajas ya que nos puede dar mucha información, pero también ciertas desventajas como en nuestro caso, un procesamiento muy lento.

Hemos podido ver las tendencias de accidentes a lo largo de los años gracias a un análisis estadístico de histogramas, en los cuales hemos visto a qué vehículos y personas sería adecuado tanto subir la prima como bajarla.

Trás la limpieza y previo análisis de datos, hemos podido comenzar a trabajar con el modelo. Aunque nos hubiese gustado poder predecir mejor la gravedad de los accidentes, y aunque hemos podido acertar la mayoría de los supuestos donde no hay fallecimientos, el modelo funcionaba y pensamos que sería de utilidad para las aseguradoras. Hemos visto también cuáles son las variables que más afectan en la severidad de un accidente (las cuales no han sido muy sorprendentes, ya que todos conocemos la importancia de las medidas de seguridad obligatorias), y aquellas que no tienen gran significancia.

Nuestros tres algoritmos obtienen resultados similares, sobre todo en la elaboración de la curva ROC. Aunque el AUC podría ser más alto, un valor como 0.718 nos parece aceptable, teniendo en cuenta que podríamos seguir trabajando sobre el modelo para obtener unos resultados brillantes.

LIBERTAD PARA GENERAR ANÁLISIS DE VALOR Y NUEVAS IDEAS. SE DEBE ATACAR MÍNIMO UN MODELO (ESTIMAR SI HABRÁ FALLECIDOS O NO). HECHO ESTO, SE PUEDE PLANTEAR DE FORMA OPCIONAL OTROS ALCANCES (LIBERTAD PARA PLANTEAR OPCIONES).

Analizamos a quién habría que subir más la prima, si a los jóvenes o a los mayores de 80. Analizamos en que rango de edad hay mas fallecimientos.

In [184]:

```
rangos=data_clean_all[['P_AGE','C_SEV']].copy()
```

In [185]:

```
rangos['P_AGE'] = rangos['P_AGE'].astype(int)
#rangos['C_SEV'] = rangos['C_SEV'].astype(int)
```

In [186]:

```
rangos['rangos']=pd.cut(x=rangos['P_AGE'],bins=[0,17,30,65,99], labels=['menor','joven','adulto','ancian
o'])
```

In [187]:

```
rangos=rangos.drop(columns=['P_AGE'])
```

In [188]:

```
rangos['C_SEV']=np.where(rangos['C_SEV']==1,0,1)
```

In [189]:

```
rangos['uno']=1
rangos
```

Out[189]:

| | C_SEV | rangos | uno |
|---|---|---|---|
| 4124328 | 1 | joven | 1 |
| 3824244 | 1 | anciano | 1 |
| 336701 | 1 | anciano | 1 |
| 788058 | 1 | menor | 1 |
| 4949818 | 1 | adulto | 1 |
| ... | ... | ... | ... |
| 1617196 | 1 | anciano | 1 |
| 5546889 | 1 | menor | 1 |
| 2791590 | 1 | anciano | 1 |
| 4535196 | 1 | joven | 1 |
| 4937641 | 1 | joven | 1 |

428017 rows × 3 columns

In [190]:

```
rangos=rangos.groupby(['C_SEV','rangos']).sum().reset_index()
```

In [192]:

```
rangos['porcentaje']=(rangos.C_SEV/rangos.uno)*100
```

In [194]:

```
rangos=rangos.drop(columns=['uno'])
```

In [195]:

```
rangos
```

Out[195]:

| | C_SEV | rangos | porcentaje |
|---|---|---|---|
| 0 | 0 | menor | 0.000000 |
| 1 | 0 | joven | 0.000000 |
| 2 | 0 | adulto | 0.000000 |
| 3 | 0 | anciano | 0.000000 |
| 4 | 1 | menor | 0.001907 |
| 5 | 1 | joven | 0.000771 |

| | | | |
|---|---|---|---|
| 6 | 1 | adulto | 0.000482 |
| 7 | 1 | anciano | 0.003220 |

Podemos ver que en los casos de accidentes de gravedad 1, hay mas fallecimientos en ancianos, seguido de menores. En esas dos clases se podría subir la prima.

PLUS: COMPLEMENTAR CON DATOS ABIERTOS DE CLIMA (AUNQUE CANADÁ ES MUY GRANDE) Y DE OTRA TIPOLOGÍA, ¿HAY ALGÚN TIPO DE RELACIÓN CON TEMPERATURAS MEDIAS, PRECIPITACIÓN MEDIA DEL DÍA/MES, NIEVE...? ¿A MÁS DÍAS FESTIVOS O DE VACACIONES, MÁS ACCIDENTES? ETC.

In [137]:

```
weather_explanation=data_model[['C_WTHR_1','C_WTHR_2','C_WTHR_3','C_WTHR_4','C_WTHR_5','C_WTHR_6','C_WTHR_7',
                                'C_MNTH_1', 'C_MNTH_2','C_MNTH_3','C_MNTH_4','C_MNTH_5','C_MNTH_6',
                                'C_MNTH_7','C_MNTH_8','C_MNTH_9','C_MNTH_10','C_MNTH_11','C_MNTH_12']].copy()
```

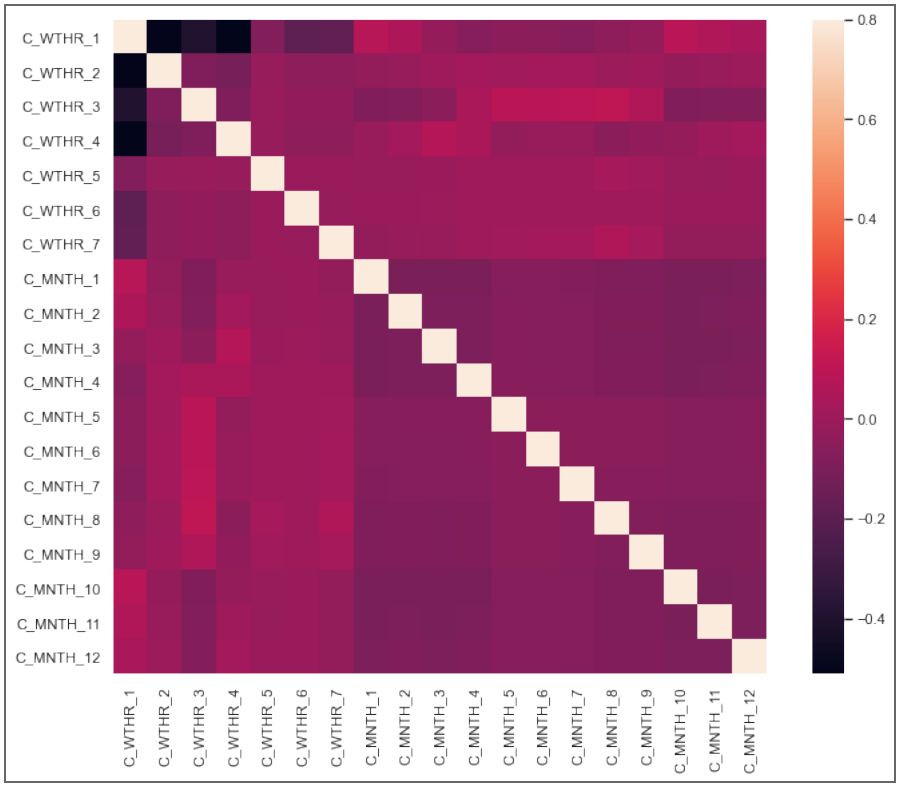
Vemos que las correlaciones no son altas, vemos que aquellas mas altas son cuando se da lluvia.

In [138]:

```
corrmat=weather_explanation.corr()
f, ax=plt.subplots(figsize=(12,9))
sns.heatmap(corrmat,vmax=.8,square=True)
```

Out[138]:

```
<AxesSubplot:>
```


C_WTHR_1
C_WTHR_2
C_WTHR_3
C_WTHR_4
C_WTHR_5
C_WTHR_6
C_WTHR_7
C_MNTH_1
C_MNTH_2
C_MNTH_3
C_MNTH_4
C_MNTH_5
C_MNTH_6
C_MNTH_7
C_MNTH_8
C_MNTH_9
C_MNTH_10
C_MNTH_11
C_MNTH_12
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4