PRÁCTICA FINAL NATALIA PALOMERA Y MARÍA MAYO

Práctica final Machine Learning: aseguradora detección de siniestro (problema de clasificación)

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
import seaborn as sns
import plotly.graph_objects as go
from sklearn.datasets import load_boston
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ParameterGrid
import multiprocessing
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.metrics import precision_score, recall_score, f1_score, confusio
from sklearn.preprocessing import LabelEncoder
from matplotlib.figure import Figure
from sklearn.linear_model import LogisticRegression
```

In [2]:
```
dataframe=pd.read_csv('NCDB_1999_to_2014.csv')
```

```
C:\Users\maria\programacion\lib\site-packages\IPython\core\interactiveshell.p
y:3444: DtypeWarning: Columns (1,2,5,12) have mixed types.Specify dtype option
on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

In [3]:
```
pd. set_option('display.max_columns', None)
pd. set_option('display.max_rows', None)
dataframe.head(10)
```

Out[3]:

	C_YEAR	C_MNTH	C_WDAY	C_HOUR	C_SEV	C_VEHS	C_CONF	C_RCFG	C_WTHR	C_RS
0	1999	1	1	20	2	02	34	UU	1	
1	1999	1	1	20	2	02	34	UU	1	
2	1999	1	1	20	2	02	34	UU	1	
3	1999	1	1	08	2	01	01	UU	5	
4	1999	1	1	08	2	01	01	UU	5	

	C_YEAR	C_MNTH	C_WDAY	C_HOUR	C_SEV	C_VEHS	C_CONF	C_RCFG	C_WTHR	C_RS
5	1999	1	1	17	2	03	QQ	QQ	1	
6	1999	1	1	17	2	03	QQ	QQ	1	
7	1999	1	1	17	2	03	QQ	QQ	1	
8	1999	1	1	17	2	03	QQ	QQ	1	
9	1999	1	1	15	2	01	04	UU	1	

In [4]:

```
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/nume
    return missingsummary
missingsummary(dataframe)
```

Out[4]:

	Total	Missing	Missing_rate (%)
C_YEAR	5860405	0	0.0
C_MNTH	5860405	0	0.0
C_WDAY	5860405	0	0.0
C_HOUR	5860405	0	0.0
C_SEV	5860405	0	0.0
C_VEHS	5860402	3	0.0
C_CONF	5860405	0	0.0
C_RCFG	5860405	0	0.0
C_WTHR	5860405	0	0.0
C_RSUR	5860405	0	0.0
C_RALN	5860405	0	0.0
C_TRAF	5860405	0	0.0
V_ID	5860405	0	0.0
V_TYPE	5860405	0	0.0
V_YEAR	5860405	0	0.0
P_ID	5860405	0	0.0
P_SEX	5860405	0	0.0
P_AGE	5860405	0	0.0
P_PSN	5860405	0	0.0

	Total	Missing	Missing_rate (%)
P_ISEV	5860405	0	0.0
P_SAFE	5860405	0	0.0
P_USER	5860405	0	0.0

In [5]:

```
col_names= dataframe.columns
distrib={}
for i in range(len(col_names)):
    aux = dataframe.groupby(col_names[i])[["C_YEAR"]].count()
    aux ['percentage'] = 100* (aux['C_YEAR']/aux['C_YEAR'].sum())
    distrib[col_names[i]]= aux
    print(distrib[col_names[i]])
```

```
        C_YEAR  percentage
C_YEAR
1999    413509    7.055980
2000    422075    7.202147
2001    409389    6.985678
2002    420008    7.166877
2003    407036    6.945527
2004    389050    6.638620
2005    386470    6.594595
2006    378523    6.458990
2007    368507    6.288081
2008    338268    5.772093
2009    330771    5.644166
2010    334555    5.708735
2011    325153    5.548303
2012    322421    5.501685
2013    317058    5.410172
2014    297612    5.078352
        C_YEAR  percentage
C_MNTH
1       242723    4.141744
2       414025    7.064785
3       417814    7.129439
4       392533    6.698052
5       468235    7.989806
6       520010    8.873278
7       537693    9.175014
8       547045    9.334594
9       512790    8.750078
10      515911    8.803334
11      496954    8.479858
12      275451    4.700204
01      257151    4.387939
02        9262    0.158044
11        2126    0.036277
12      250297    4.270985
UU         385    0.006570
        C_YEAR  percentage
C_WDAY
1       355515    6.066390
2       408828    6.976105
```

```
3       406094    6.929453
4       438388    7.480507
5       483641    8.252689
6       409812    6.992896
7       315770    5.388194
1       420497    7.175221
2       407437    6.952369
3       419466    7.157628
4       431265    7.358962
5       516084    8.806286
6       457376    7.804512
7       388909    6.636214
U         1323    0.022575
        C_YEAR  percentage
C_HOUR
00       88481    1.509810
01       73383    1.252183
02       77500    1.322434
03       64036    1.092689
04       39817    0.679424
05       44952    0.767046
06      106086    1.810216
07      207965    3.548646
08      302834    5.167459
09      227626    3.884134
10      248984    4.248580
11      301916    5.151794
12      368278    6.284173
13      367315    6.267741
14      397342    6.780112
15      492742    8.407985
16      519601    8.866298
17      500203    8.535297
18      375902    6.414267
19      281863    4.809616
20      217277    3.707542
21      204317    3.486397
22      164878    2.813423
23      127698    2.178996
UU       59409    1.013735
        C_YEAR  percentage
C_SEV
1        98633    1.683041
2      5761772   98.316959
         C_YEAR  percentage
C_VEHS
1        661165   11.281905
2       1894124   32.320718
3        387281    6.608437
4         96639    1.649017
5         22860    0.390076
6          7888    0.134598
7          3123    0.053290
8          1570    0.026790
9           931    0.015886
10          761    0.012985
11          466    0.007952
12          428    0.007303
13          246    0.004198
14          261    0.004454
```

15	224	0.003822
16	158	0.002696
17	94	0.001604
18	122	0.002082
19	62	0.001058
20	219	0.003737
21	29	0.000495
22	24	0.000410
24	24	0.000410
25	65	0.001109
26	61	0.001041
27	38	0.000648
28	63	0.001075
31	32	0.000546
33	39	0.000665
35	172	0.002935
36	254	0.004334
37	41	0.000700
38	59	0.001007
39	48	0.000819
43	44	0.000751
44	102	0.001740
46	59	0.001007
56	58	0.000990
57	58	0.000990
58	61	0.001041
72	123	0.002099
77	113	0.001928
01	587278	10.021121
02	1718316	29.320787
03	352189	6.009639
04	86397	1.474250
05	21217	0.362040
06	6501	0.110931
07	2558	0.043649
08	1100	0.018770
09	902	0.015391
10	432	0.007372
11	395	0.006740
12	216	0.003686
13	169	0.002884
14	286	0.004880
15	252	0.004300
16	122	0.002082
17	79	0.001348
18	57	0.000973
19	63	0.001075
20	46	0.000785
21	111	0.001894
22	110	0.001877
23	24	0.000410
24	86	0.001467
25	2	0.000034
26	55	0.000939
27	48	0.000819
29	31	0.000529
30	34	0.000580
32	42	0.000717
34	36	0.000614
35	3	0.000051

38	49	0.000836
40	41	0.000700
41	51	0.000870
51	80	0.001365
54	86	0.001467
71	92	0.001570
77	116	0.001979
UU	541	0.009231

C_CONF	C_YEAR	percentage
01	92430	1.577195
02	186935	3.189797
03	165410	2.822501
04	214350	3.657597
05	13049	0.222664
06	515468	8.795774
21	1771212	30.223372
22	197006	3.361645
23	70088	1.195958
24	52067	0.888454
25	10922	0.186369
31	195284	3.332261
32	47307	0.807231
33	422241	7.204980
34	62181	1.061036
35	868197	14.814625
36	431298	7.359525
41	80961	1.381492
QQ	284980	4.862804
UU	179019	3.054721

C_RCFG	C_YEAR	percentage
01	2079515	35.484152
02	2746752	46.869662
03	289184	4.934540
04	23723	0.404801
05	51196	0.873592
06	6001	0.102399
07	1009	0.017217
08	11780	0.201010
09	1743	0.029742
10	556	0.009487
QQ	144298	2.462253
UU	504648	8.611145

C_WTHR	C_YEAR	percentage
1	4074538	69.526560
2	600226	10.242057
3	600105	10.239992
4	354615	6.051032
5	30330	0.517541
6	82439	1.406712
7	15164	0.258753
Q	15013	0.256177
U	87975	1.501176

C_RSUR	C_YEAR	percentage
1	3846162	65.629628
2	1080964	18.445210
3	259234	4.423483

```
4         72041    1.229284
5        320445    5.467967
6         25355    0.432649
7          6018    0.102689
8          1226    0.020920
9           292    0.004983
Q        170217    2.904526
U         78451    1.338662
         C_YEAR  percentage
C_RALN
1       4162359   71.025108
2        584025    9.965608
3        360870    6.157766
4        224222    3.826050
5         37991    0.648266
6         27626    0.471401
Q         28602    0.488055
U        434710    7.417747
         C_YEAR  percentage
C_TRAF
01      1628309   27.784923
02        19114    0.326155
03       650008   11.091520
04        89688    1.530406
05         4199    0.071650
06        45600    0.778103
07         2419    0.041277
08        24479    0.417702
09          550    0.009385
10         2924    0.049894
11         3421    0.058375
12         3209    0.054757
13         4298    0.073340
15         3852    0.065729
16          817    0.013941
17         3969    0.067726
18      3068048   52.352150
QQ        82018    1.399528
UU       223483    3.813440
       C_YEAR  percentage
V_ID
1     1109276   18.928316
2      764061   13.037683
3      106251    1.813032
4       20933    0.357194
5        4676    0.079790
6        1605    0.027387
7         677    0.011552
8         387    0.006604
9         274    0.004675
10        167    0.002850
11        147    0.002508
12        111    0.001894
13         98    0.001672
14         74    0.001263
15         59    0.001007
16         48    0.000819
17         42    0.000717
18         36    0.000614
19         44    0.000751
```

20	28	0.000478
21	25	0.000427
22	25	0.000427
23	25	0.000427
24	20	0.000341
25	20	0.000341
26	21	0.000358
27	17	0.000290
28	21	0.000358
29	16	0.000273
30	20	0.000341
31	18	0.000307
32	18	0.000307
33	17	0.000290
34	14	0.000239
35	16	0.000273
36	15	0.000256
37	12	0.000205
38	8	0.000137
39	7	0.000119
40	6	0.000102
41	7	0.000119
42	4	0.000068
43	6	0.000102
44	5	0.000085
45	3	0.000051
46	3	0.000051
47	2	0.000034
48	2	0.000034
49	2	0.000034
50	1	0.000017
51	3	0.000051
52	5	0.000085
53	1	0.000017
54	1	0.000017
55	1	0.000017
56	1	0.000017
57	1	0.000017
99	82702	1.411199
01	2000007	34.127454
02	1386353	23.656266
03	192272	3.280865
04	37585	0.641338
05	8114	0.138455
06	2567	0.043802
07	1173	0.020016
08	626	0.010682
09	452	0.007713
10	270	0.004607
11	216	0.003686
12	161	0.002747
13	138	0.002355
14	109	0.001860
15	93	0.001587
16	71	0.001212
17	71	0.001212
18	66	0.001126
19	56	0.000956
20	55	0.000939
21	41	0.000700

22	42	0.000717
23	39	0.000665
24	38	0.000648
25	33	0.000563
26	33	0.000563
27	31	0.000529
28	27	0.000461
29	23	0.000392
30	22	0.000375
31	26	0.000444
32	29	0.000495
33	19	0.000324
34	20	0.000341
35	23	0.000392
36	17	0.000290
37	12	0.000205
38	13	0.000222
39	11	0.000188
40	13	0.000222
41	11	0.000188
42	13	0.000222
43	10	0.000171
44	14	0.000239
45	12	0.000205
46	13	0.000222
47	10	0.000171
48	14	0.000239
49	10	0.000171
50	8	0.000137
51	7	0.000119
52	10	0.000171
53	13	0.000222
54	13	0.000222
55	7	0.000119
56	14	0.000239
57	4	0.000068
58	4	0.000068
59	9	0.000154
60	8	0.000137
61	5	0.000085
62	5	0.000085
63	6	0.000102
64	6	0.000102
65	7	0.000119
66	7	0.000119
67	4	0.000068
68	4	0.000068
69	4	0.000068
70	4	0.000068
71	5	0.000085
72	9	0.000154
73	3	0.000051
74	2	0.000034
75	1	0.000017
76	3	0.000051
77	2	0.000034
83	1	0.000017
85	1	0.000017
86	2	0.000034
99	136635	2.331494

UU 433 0.007389

V_TYPE	C_YEAR	percentage
01	4827390	82.372976
05	66106	1.128011
06	175491	2.994520
07	88057	1.502575
08	70556	1.203944
09	20914	0.356870
10	906	0.015460
11	51260	0.874684
14	122734	2.094292
16	9722	0.165893
17	122907	2.097244
18	3132	0.053443
19	2001	0.034144
20	7635	0.130281
21	825	0.014078
22	4583	0.078203
23	3075	0.052471
NN	227913	3.889032
QQ	25444	0.434168
UU	29754	0.507712

V_YEAR	C_YEAR	percentage
1901	14	0.000239
1903	8	0.000137
1904	2	0.000034
1905	3	0.000051
1906	1	0.000017
1907	2	0.000034
1908	5	0.000085
1909	2	0.000034
1910	1	0.000017
1911	9	0.000154
1912	8	0.000137
1913	4	0.000068
1914	14	0.000239
1915	28	0.000478
1916	7	0.000119
1917	6	0.000102
1918	9	0.000154
1919	17	0.000290
1920	31	0.000529
1921	7	0.000119
1922	16	0.000273
1923	14	0.000239
1924	5	0.000085
1925	12	0.000205
1926	20	0.000341
1927	7	0.000119
1928	22	0.000375
1929	15	0.000256
1930	38	0.000648
1931	24	0.000410
1932	11	0.000188
1933	11	0.000188
1934	10	0.000171
1935	15	0.000256
1936	3	0.000051

1937	18	0.000307
1938	36	0.000614
1939	86	0.001467
1940	24	0.000410
1941	13	0.000222
1942	11	0.000188
1943	4	0.000068
1944	14	0.000239
1945	21	0.000358
1946	27	0.000461
1947	44	0.000751
1948	34	0.000580
1949	38	0.000648
1950	50	0.000853
1951	38	0.000648
1952	52	0.000887
1953	36	0.000614
1954	38	0.000648
1955	72	0.001229
1956	94	0.001604
1957	96	0.001638
1958	76	0.001297
1959	95	0.001621
1960	122	0.002082
1961	74	0.001263
1962	168	0.002867
1963	177	0.003020
1964	258	0.004402
1965	415	0.007081
1966	505	0.008617
1967	678	0.011569
1968	743	0.012678
1969	920	0.015699
1970	973	0.016603
1971	855	0.014589
1972	1409	0.024043
1973	1491	0.025442
1974	1641	0.028001
1975	2210	0.037711
1976	2872	0.049007
1977	4567	0.077930
1978	5990	0.102211
1979	8611	0.146935
1980	11839	0.202017
1981	17290	0.295031
1982	13869	0.236656
1983	19099	0.325899
1984	32451	0.553733
1985	46251	0.789212
1986	69722	1.189713
1987	82980	1.415943
1988	121781	2.078030
1989	139738	2.384443
1990	159421	2.720307
1991	169021	2.884118
1992	192313	3.281565
1993	186598	3.184046
1994	194099	3.312041
1995	225404	3.846219
1996	198814	3.392496

1997	262438	4.478155
1998	295311	5.039089
1999	287640	4.908193
2000	323183	5.514687
2001	278900	4.759057
2002	293534	5.008766
2003	279496	4.769227
2004	225647	3.850365
2005	227478	3.881609
2006	191089	3.260679
2007	184021	3.140073
2008	147655	2.519536
2009	105236	1.795712
2010	91852	1.567332
2011	64215	1.095743
2012	50899	0.868524
2013	35051	0.598099
2014	14366	0.245137
2015	1229	0.020971
NNNN	260256	4.440922
UUUU	324122	5.530710

P_ID	C_YEAR	percentage
01	4170908	71.170986
02	1133519	19.341991
03	341408	5.825672
04	131881	2.250373
05	41503	0.708193
06	11863	0.202426
07	4860	0.082929
08	2260	0.038564
09	1433	0.024452
10	1115	0.019026
11	906	0.015460
12	773	0.013190
13	652	0.011126
14	558	0.009522
15	495	0.008447
16	431	0.007354
17	383	0.006535
18	366	0.006245
19	336	0.005733
20	312	0.005324
21	283	0.004829
22	260	0.004437
23	245	0.004181
24	218	0.003720
25	208	0.003549
26	184	0.003140
27	171	0.002918
28	163	0.002781
29	154	0.002628
30	146	0.002491
31	133	0.002269
32	122	0.002082
33	115	0.001962
34	104	0.001775
35	93	0.001587
36	84	0.001433
37	73	0.001246

38	63	0.001075
39	61	0.001041
40	58	0.000990
41	51	0.000870
42	45	0.000768
43	42	0.000717
44	34	0.000580
45	32	0.000546
46	27	0.000461
47	25	0.000427
48	24	0.000410
49	22	0.000375
50	21	0.000358
51	19	0.000324
52	17	0.000290
53	15	0.000256
54	14	0.000239
55	10	0.000171
56	7	0.000119
57	6	0.000102
58	6	0.000102
59	6	0.000102
60	5	0.000085
61	4	0.000068
62	4	0.000068
63	4	0.000068
64	4	0.000068
65	3	0.000051
66	3	0.000051
67	3	0.000051
68	3	0.000051
69	3	0.000051
70	3	0.000051
71	3	0.000051
72	3	0.000051
73	2	0.000034
74	2	0.000034
75	2	0.000034
76	2	0.000034
77	2	0.000034
78	2	0.000034
79	2	0.000034
80	2	0.000034
81	2	0.000034
82	2	0.000034
83	2	0.000034
84	2	0.000034
85	2	0.000034
86	2	0.000034
87	2	0.000034
88	2	0.000034
89	2	0.000034
90	2	0.000034
91	2	0.000034
92	2	0.000034
93	2	0.000034
94	1	0.000017
95	1	0.000017
99	1	0.000017
NN	10976	0.187291

```
UU         16    0.000273
        C_YEAR  percentage
P_SEX
F      2440421   41.642532
M      3170244   54.095988
N        14786    0.252303
U       234954    4.009177
       C_YEAR  percentage
P_AGE
01      39423    0.672701
02      25371    0.432922
03      25532    0.435670
04      26407    0.450600
05      26642    0.454610
06      26002    0.443689
07      26698    0.455566
08      27120    0.462767
09      27424    0.467954
10      28697    0.489676
11      28195    0.481110
12      30392    0.518599
13      32123    0.548136
14      41043    0.700344
15      52704    0.899324
16      96467    1.646081
17     158163    2.698841
18     171719    2.930156
19     167721    2.861935
20     158747    2.708806
21     147725    2.520730
22     138421    2.361970
23     130566    2.227935
24     122300    2.086886
25     118292    2.018495
26     110289    1.881935
27     105778    1.804961
28     102137    1.742832
29      98413    1.679287
30      99223    1.693108
31      94044    1.604736
32      93307    1.592160
33      92568    1.579550
34      92167    1.572707
35      94472    1.612039
36      92899    1.585198
37      93354    1.592962
38      94091    1.605538
39      94326    1.609547
40      97677    1.666728
41      94266    1.608524
42      95294    1.626065
43      93962    1.603336
44      92742    1.582519
45      92873    1.584754
46      89262    1.523137
47      86978    1.484164
48      85490    1.458773
49      83287    1.421182
50      81968    1.398675
51      77422    1.321103
```

52	75046	1.280560
53	72062	1.229642
54	68252	1.164629
55	64957	1.108405
56	60806	1.037573
57	57115	0.974591
58	54166	0.924271
59	50805	0.866920
60	48971	0.835625
61	43770	0.746877
62	41411	0.706624
63	39182	0.668589
64	36664	0.625622
65	35207	0.600761
66	31622	0.539587
67	29846	0.509282
68	28019	0.478107
69	26383	0.450191
70	26371	0.449986
71	24125	0.411661
72	23618	0.403010
73	22058	0.376390
74	21463	0.366237
75	20598	0.351477
76	19456	0.331991
77	18121	0.309211
78	17096	0.291720
79	16107	0.274844
80	15184	0.259095
81	12607	0.215122
82	11600	0.197939
83	10214	0.174288
84	8496	0.144973
85	7406	0.126374
86	6064	0.103474
87	4796	0.081837
88	3676	0.062726
89	2822	0.048154
90	2192	0.037404
91	1429	0.024384
92	1025	0.017490
93	688	0.011740
94	453	0.007730
95	292	0.004983
96	201	0.003430
97	110	0.001877
98	193	0.003293
99	321	0.005477
NN	18016	0.307419
UU	377140	6.435391

P_PSN	C_YEAR	percentage
11	3926086	66.993425
12	87671	1.495989
13	884534	15.093394
21	215997	3.685701
22	90317	1.541139
23	258122	4.404508
31	4566	0.077913
32	30092	0.513480

```
33         5756    0.098218
96        43850    0.748242
97          191    0.003259
98         6658    0.113610
99       208888    3.564395
NN        14153    0.241502
QQ        25862    0.441301
UU        57662    0.983925
          C_YEAR  percentage
P_ISEV
1       2375208   40.529759
2       3073431   52.444003
3         40354    0.688587
N        290711    4.960596
U         80701    1.377055
          C_YEAR  percentage
P_SAFE
01        198606    3.388947
02       4163903   71.051455
09         96437    1.645569
10           112    0.001911
11            11    0.000188
12         18322    0.312641
13        148730    2.537879
NN        571946    9.759496
QQ         47150    0.804552
UU        615188   10.497363
          C_YEAR  percentage
P_USER
1       3658827   62.433006
2       1561014   26.636623
3        219337    3.742694
4        122907    2.097244
5        122734    2.094292
U        175586    2.996141
```

In [6]:

```
print(distrib['C_MNTH'].index)
dataframe['C_MNTH'] = np.where(dataframe['C_MNTH'] == "01", 1, dataframe['C_MI
dataframe['C_MNTH'] = np.where(dataframe['C_MNTH'] == "02", 2, dataframe['C_MI
dataframe['C_MNTH'] = np.where(dataframe['C_MNTH'] == "11", 1, dataframe['C_MI
dataframe['C_MNTH'] = np.where(dataframe['C_MNTH'] == "12", 1, dataframe['C_MI
print(dataframe.C_MNTH.value_counts())
distrib_cambios=['C_MNTH']
```

```
Index([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, '01', '02', '11', '12', 'UU'], d
type='object', name='C_MNTH')
1     752297
8     547045
7     537693
6     520010
10    515911
9     512790
11    496954
5     468235
2     423287
3     417814
4     392533
12    275451
```

```
UU       385
Name: C_MNTH, dtype: int64
```

In [7]:

```
print(distrib['C_WDAY'].index)
dataframe['C_WDAY'] = np.where(dataframe['C_WDAY'] == '1', 1, dataframe['C_WDA
dataframe['C_WDAY'] = np.where(dataframe['C_WDAY'] == '2', 2, dataframe['C_WDA
dataframe['C_WDAY'] = np.where(dataframe['C_WDAY'] == '3', 3, dataframe['C_WDA
dataframe['C_WDAY'] = np.where(dataframe['C_WDAY'] == '4', 4, dataframe['C_WDA
dataframe['C_WDAY'] = np.where(dataframe['C_WDAY'] == '5', 5, dataframe['C_WDA
dataframe['C_WDAY'] = np.where(dataframe['C_WDAY'] == '6', 6, dataframe['C_WDA
dataframe['C_WDAY'] = np.where(dataframe['C_WDAY'] == '7', 7, dataframe['C_WDA
print(dataframe.C_WDAY.value_counts())
distrib_cambios.append('C_WDAY')

for i in range(len(distrib_cambios)):
    aux = dataframe.groupby(distrib_cambios[i])[["C_YEAR"]].count()
    aux ['percentage'] = 100* (aux['C_YEAR']/aux['C_YEAR'].sum())
    distrib[distrib_cambios[i]]= aux
    print(distrib[distrib_cambios[i]])
```

```
Index([1, 2, 3, 4, 5, 6, 7, '1', '2', '3', '4', '5', '6', '7', 'U'], dtype='ob
ject', name='C_WDAY')
5    999725
4    869653
6    867188
3    825560
2    816265
1    776012
7    704679
U      1323
Name: C_WDAY, dtype: int64
        C_YEAR  percentage
C_MNTH
1       752297   12.836946
2       423287    7.222828
3       417814    7.129439
4       392533    6.698052
5       468235    7.989806
6       520010    8.873278
7       537693    9.175014
8       547045    9.334594
9       512790    8.750078
10      515911    8.803334
11      496954    8.479858
12      275451    4.700204
UU         385    0.006570
        C_YEAR  percentage
C_WDAY
1       776012   13.241610
2       816265   13.928474
3       825560   14.087081
4       869653   14.839469
5       999725   17.058975
6       867188   14.797407
7       704679   12.024408
U         1323    0.022575
```

En esta práctica analizamos los datos de una aseguradora detección de siniestro, nos ha parecido interesante analizar esta práctica ya que en nuestro entorno se producen

muchos accidentes de tráfico y queríamos saber más del tema. Lo primero que realizamos es importar todas las librerías que vamos a utilizar a lo largo de la práctica, e importamos el dataframe con todos los datos a utilizar. Hacemos un head para observar los primeros datos del dataframe y ver que columnas hay y el tipo de información que nos muestran.

- C_YEAR Year
- C_MNTH Month
- C_WDAY Day of week
- C_HOUR Collision hour
- C_SEV Collision severity
- C_VEHS Number of vehicles involved in collision
- C_CONF Collision configuration
- C_RCFG Roadway configuration
- C_WTHR Weather conditiom
- C_RSUR Road surface
- C_RALN Road alignment
- C_TRAF Traffic control
- V_ID Vehicle sequence number
- V_TYPE Vehicle type
- V_YEAR Vehicle model year
- P_ID Person sequence number
- P_SEX 32 Person sex
- P_AGE Person age
- P_PSN Person position
- P_ISEV Medical treatment required
- P_SAFE Safety device used
- P_USER Road user class

Con la función def summary, observamos la cantidad de valores nulos que hay en el dataframe, y vemos que hay 0 lo que nos facilitara mucho la elaboración de la práctica.

Renombramos la columna 'COL_NAMES' por dataframe.columns y hacemos un groupby 'COL_NAMES' relacionándola con C_YEARS siendo capaces de obtener la relación de cada variable con el número de colisiones que han ocurrido a lo largo de los años con los que que vamos a trabajar. De cada variable hemos querido destacar el dato más significativo,

- C_YEAR, en el año 2000 se produjeron más colisiones.
- C_MNTH, en el mes de agosto, se producen mas colisiones.
- C_WDAY el viernes es el día de la semana en el que más accidentes se producen.
- C_HOUR, las 16:00 pm es a la hora que más colisiones se producen.
- C_SEV, la mayoría de las colisiones no provocan muertes.
- C_VEHS, la mayoría de las colisiones implican solamente a 2 vehículos en el accidente.
- C_CONF, la mayoría de las colisiones son de dos vehículos en movimiento, en la misma dirección de viaje, siendo una colisión

trasera.
- C_RCFG, la mayoría de las colisiones se producen en una intersección de al menos dos vías públicas.
- C_WTHR , la mayoría de las colisiones se producen en días soleados y claros.
- C_RSUR, la mayoría de las colisiones se producen en calzadas normales y secas.
- C_RALN, la mayoría de las colisiones se producen en una alineación vial recta y nivelada.
- C_TRAF, la mayoría de las colisiones se producen cuando no hay ningún tipo de control de tráfico.
- V_ID , la mayoría de las colisiones se producen por vehiculos conducidos por una persona con dni de 18 años.
- V_TYPE, el tipo de vehículo que tiene más colisiones es el vehículo de servicio ligero (automóvil de pasajeros, furgoneta de pasajeros, vehículos utilitarios ligeros y
- camionetas pick-up livianas).
- V_YEAR, el modelo de año del vehículo que mas colisiones produce no somos capaces de averigaurlo ya que su parámetro es ‘unkown’.
- P_ID la mayoría de las colisiones se producen por personas con dni de 18 años.
- P_SEX 32 Person sex, la mayoría de las colisiones las producen las personas de genero masculino.
- P_AGE.
- P_PSN la persona que más colisiones sufre es el conductor.
- P_ISEV , en la mayoría de las colisiones el tratamiento médico requerido que se neceisat es producido por una lesión.
- P_SAFE Safety device used Safety device used or child restraint used en la mayoría de las colisiones que se producen se utiliza el dispositivo de seguridad o sistema de retención infantil.
- P_USER: la clase de usuario de la carretera que más colusiones percibe es el conductor de vehículo motorizado.

Nos daba error determinadas variables de C_MNTH y C_WDAY por lo que has hemos convertido todos los valores a numérico ya que algunos eran string y por ello daba error. En el caso de C_WDAY, convertimos todos los días de la semana y en el caso de C_MNTH hemos convertido los meses de enero, febrero, noviembre y diciembre que son las que nos daban problemas.

PREGUNTAS 2 Y 3

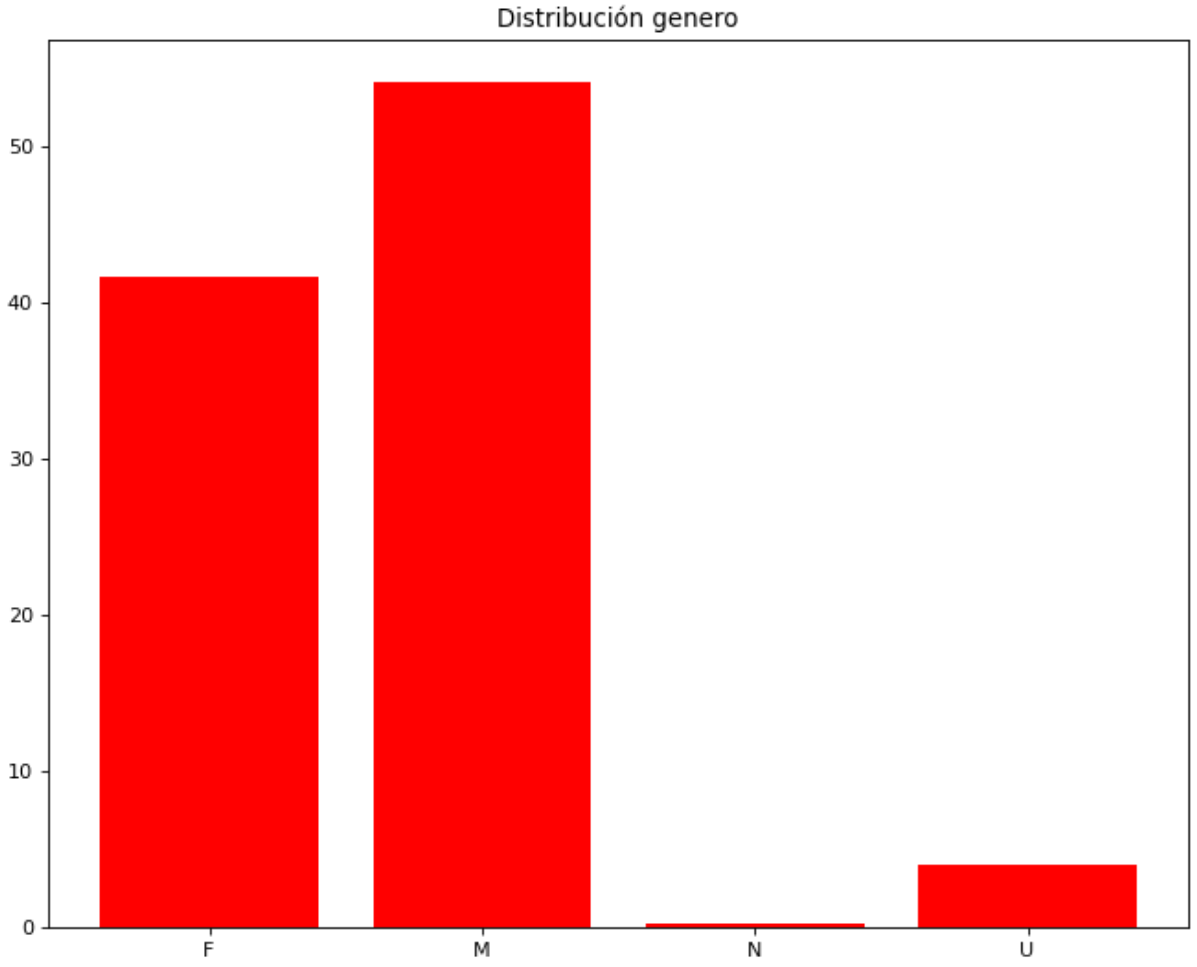
¿Qué tipos de vehículos (modelos, antigüedad, etc.) y conductores son más propensos a tener accidentes (acción correctiva en prima)?

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

Para resolver estas pregunats, hemos creido conveniente utilizar la tecnica clustering de modelos no supervisados, por lo que hacemos un clustering y vemos que clústeres son los que más y menos accidentes tienen. Luego vemos el centro del clúster y vemos cuales son las características que mejor lo describen. Para ello, al ser todo variables categóricas, las transformo en dummy variables para el clustering. Sin embargo, solo selecciono las variables que tienen que ver con los conductores para sacar las conclusiones.

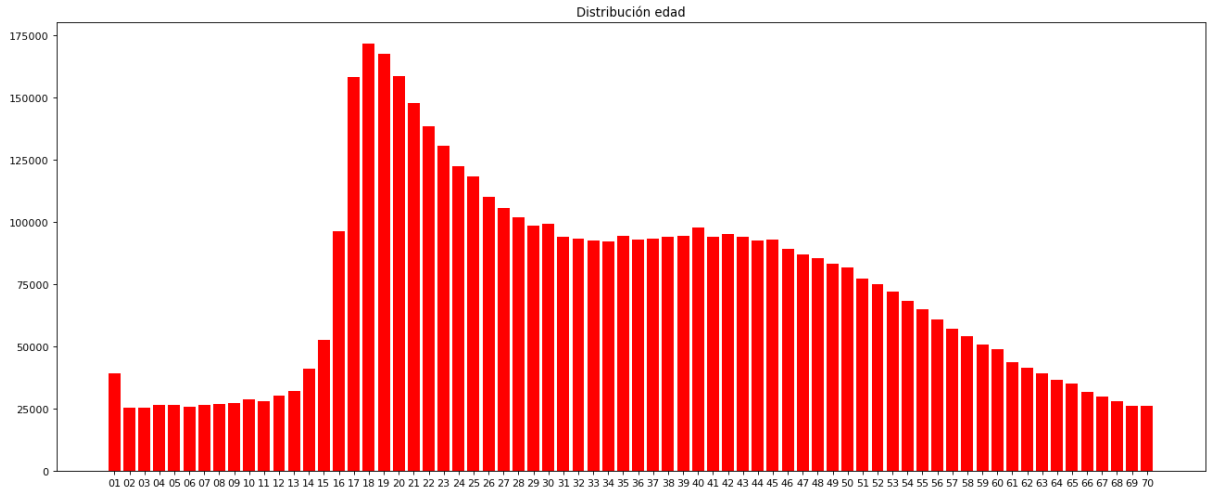
In [8]:
```
from matplotlib.pyplot import figure
figure(figsize=(10,8 ), dpi=80)
plt.bar(distrib['P_SEX'].index,distrib['P_SEX']['percentage'] , color='r')
plt.title("Distribución genero")
```

Out[8]:
```
Text(0.5, 1.0, 'Distribución genero')
```



In [9]:
```
figure(figsize=(20,8 ), dpi=80)
plt.bar(distrib['P_AGE'].index[0:70],distrib['P_AGE']['C_YEAR'][0:70] , color=
plt.title("Distribución edad")
```

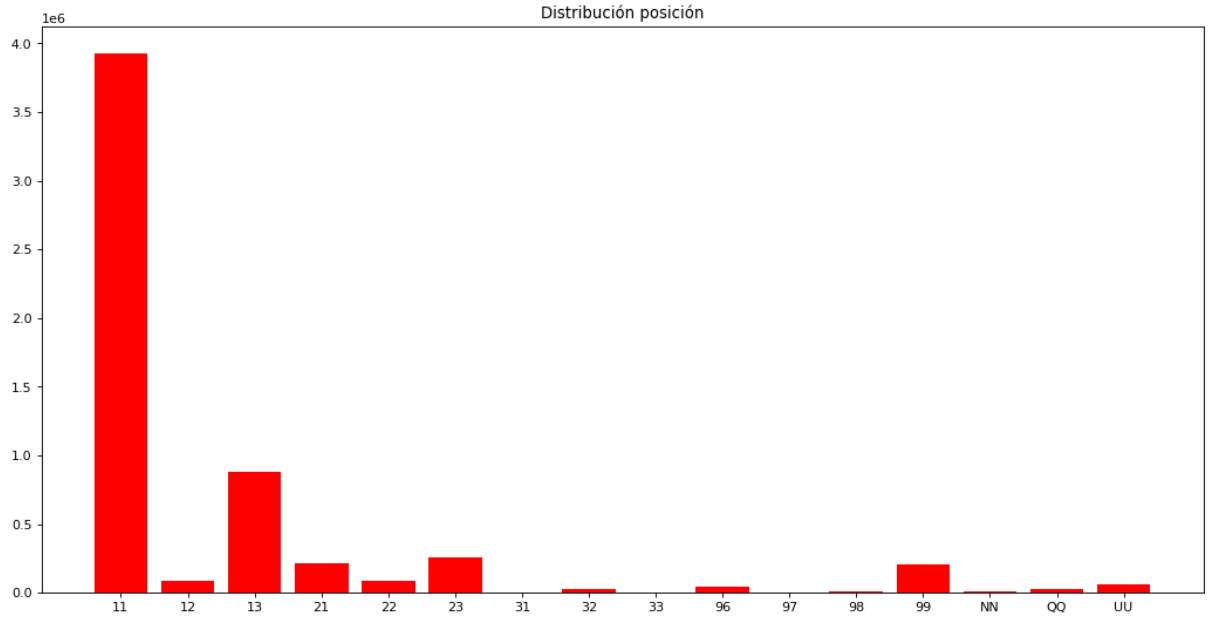
Out[9]:
```
Text(0.5, 1.0, 'Distribución edad')
```



In [10]:

```
figure(figsize=(16,8 ), dpi=80)
labels= ['Conductor', 'Primera fila centro', 'Primera fila derecha', 'Segunda
plt.bar(distrib['P_PSN'].index,distrib['P_PSN']['C_YEAR'] , color='r')
plt.title("Distribución posición")
for i in range(len(labels)):
    print(distrib['P_PSN'].index[i]+ ": "+ labels[i])
```

```
11: Conductor
12: Primera fila centro
13: Primera fila derecha
21: Segunda fila izquierda
22: Segunda fila centro
23: Segunda fila derecha
31: Tercer fila izquierda
32: Tercer fila centro
33: Tercer fila derecha
96: Posición desconocida
97: Sentado encima
98: Pasajero fuera del coche
99: Peatón
NN: NN
QQ: QQ
UU: UU
```



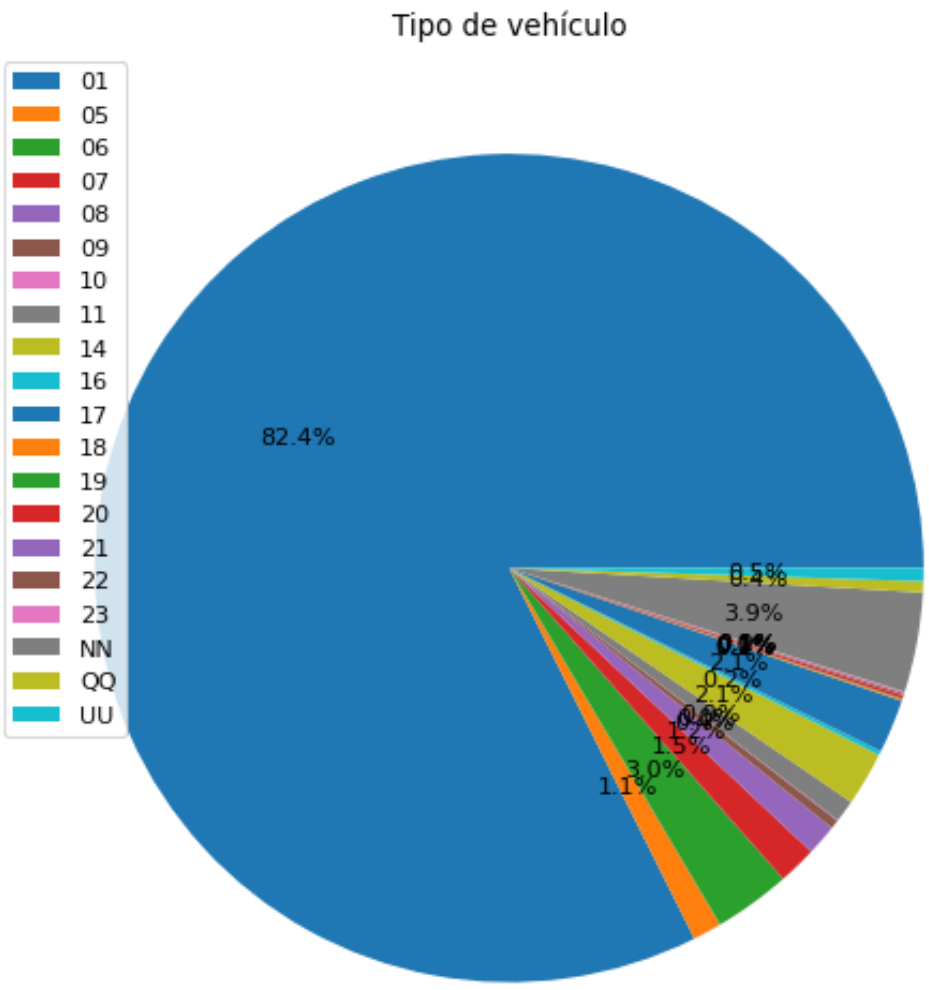
In [11]:

```
from matplotlib.pyplot import figure
```

```
figure(figsize=(10,8 ), dpi=80)

plt.pie(distrib['V_TYPE']['C_YEAR'], autopct='%1.1f%%')
plt.legend( distrib['V_TYPE'].index, loc="best")
plt.title("Tipo de vehículo")
```

Out[11]:

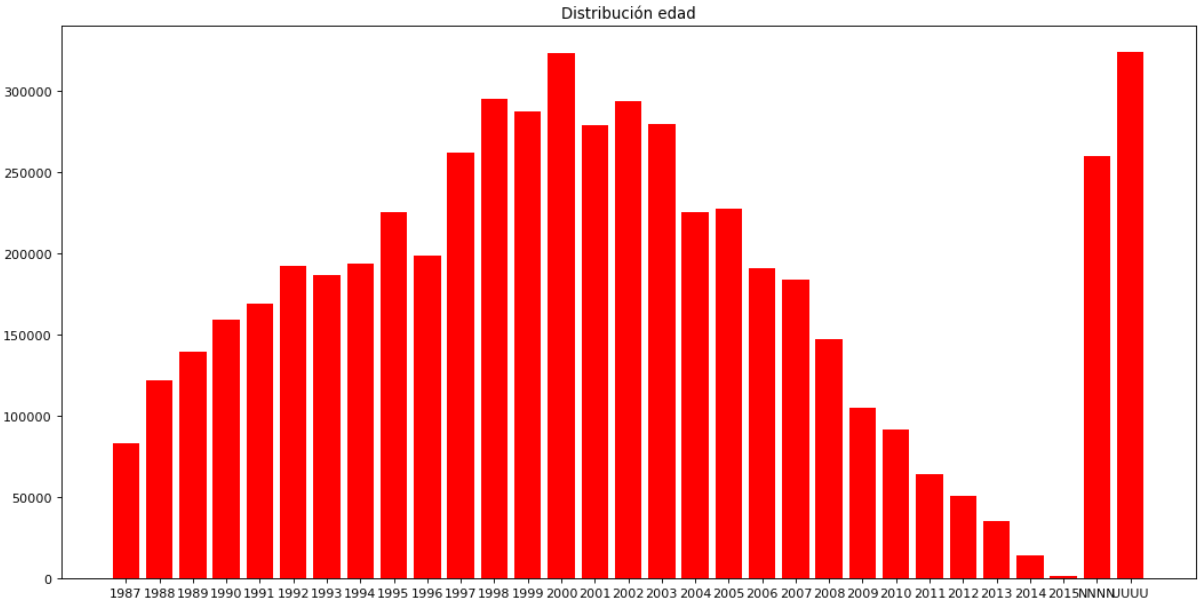
```
Text(0.5, 1.0, 'Tipo de vehículo')
```



In [12]:

```
figure(figsize=(16,8 ), dpi=80)
plt.bar(distrib['V_YEAR'].index[85:],distrib['V_YEAR']['C_YEAR'][85:] , color=
plt.title("Distribución edad")
print(distrib['V_YEAR'].index)
```

```
Index(['1901', '1903', '1904', '1905', '1906', '1907', '1908', '1909', '1910',
       '1911',
       ...
       '2008', '2009', '2010', '2011', '2012', '2013', '2014', '2015', 'NNNN',
       'UUUU'],
      dtype='object', name='V_YEAR', length=116)
```

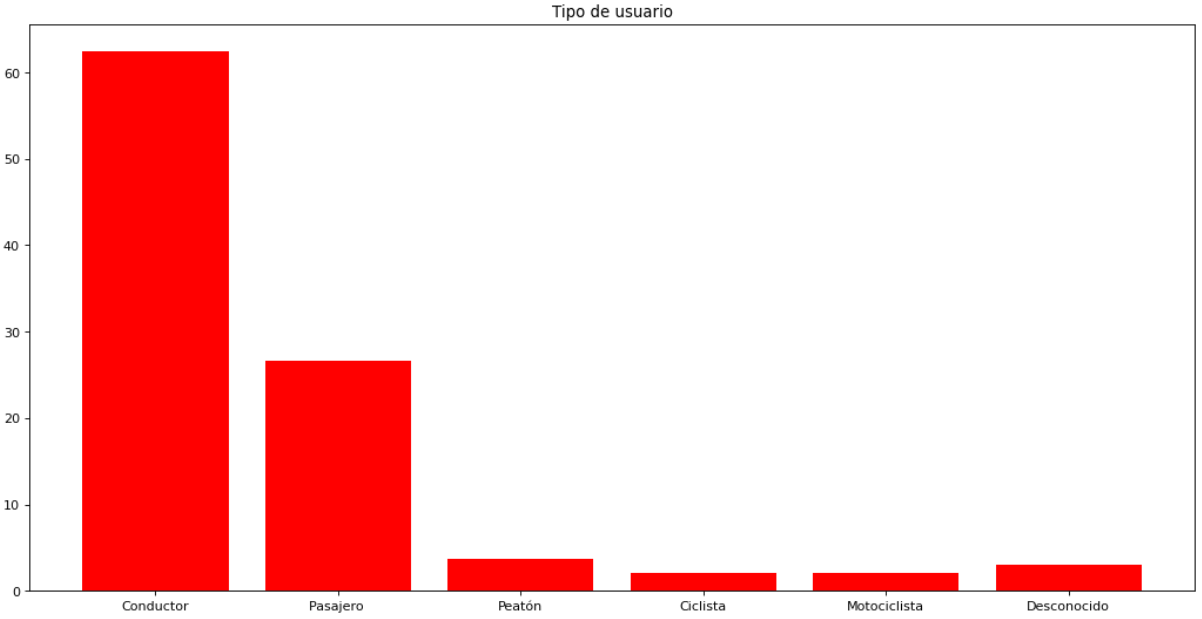


In [13]:

```
figure(figsize=(16,8 ), dpi=80)
label= ['Conductor', 'Pasajero ', 'Peatón', 'Ciclista', 'Motociclista', 'Desc
plt.bar(label,distrib['P_USER']['percentage'] , color='r')
plt.title("Tipo de usuario")
```

Out[13]:

```
Text(0.5, 1.0, 'Tipo de usuario')
```



In [14]:

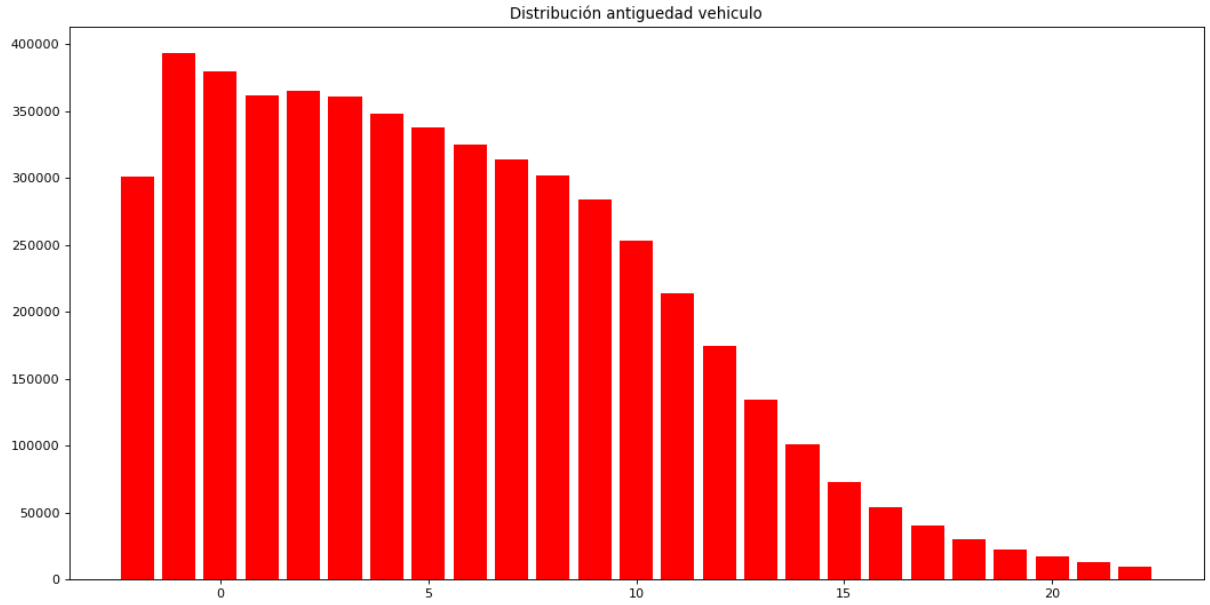
```
def ant(colision, vehiculo):
    if  vehiculo == 'UUUU':
        return None
    elif vehiculo == 'NNNN':
        return None
    else:
        return (int(colision) - int(vehiculo))
dataframe['antiguedad']=dataframe.apply( lambda x: ant(x.C_YEAR, x.V_YEAR), a
figure(figsize=(16,8 ), dpi=80)
```

Out[14]:

```
<Figure size 1280x640 with 0 Axes>

<Figure size 1280x640 with 0 Axes>
```

In [15]:
```
antiguedad = dataframe.groupby('antiguedad')[["C_YEAR"]].count()
antiguedad ['percentage'] = 100* (aux['C_YEAR']/aux['C_YEAR'].sum())
figure(figsize=(16,8 ), dpi=80)
indice= antiguedad.index[0:25].values
valor= antiguedad['C_YEAR'][0:24].values
plt.bar(indice,valor, color='r')
plt.title("Distribución antiguedad vehiculo")
print(antiguedad.index)
```

```
Float64Index([ -2.0,  -1.0,   0.0,   1.0,   2.0,   3.0,   4.0,   5.0,   6.0,
                7.0,
               ...
               93.0,  94.0,  95.0,  96.0,  97.0,  98.0,  99.0, 100.0, 101.0,
              103.0],
             dtype='float64', name='antiguedad', length=105)
```



Primero nos fijamos en la variable P_sex para ver qué porcentaje según la distribución de genero tiene mas accidentes, concluyendo que el genero masculino es el que mas accidentes tiene, cabe destacar que hay bastantes conductores que se dan a la fuga sin ser capaces de identificar su género. Después, nos fijamos en la variable P_age, para ver qué porcentaje según la distribución de edad tiene mas accidentes. Observamos que las personas entre los 16 y 27 años sufren el mayor numero de accidentes, a partir de los 28 años el numero de accidentes va disminuyendo con la edad de los conductores. También nos parece importante saber qué porcentaje según la distribución de la posición de las personas en el vehículo sufrieron el accidente, por ello nos fijamos en la columna 'P_PSN. Observando que la mayoría de los accidentes los sufren los conductores (con un numero mucho mas alto que el resto), seguido de personas en la primera fila derecha, en la segunda fila derecha, peatones, personas en la segunda fila izquierda, segunda fila centro y primera fila centro.

Todas estas conclusiones las hemos sacado creando histogramas, pero para saber el tipo de vehículo que mas accidentes tiene hemos utilizado un pie chart. Fijándonos en los porcentajes del tipo de vehículo que mas accidentes tiene, observamos que el grupo 'light Duty Vehicle (Passenger car, Passenger van, Light utility vehicles and light duty pick up trucks´) es el grupo de vehículos que mas accidentes sufren con 82,4%.

También hemos querido analizar el porcentaje según la distribución del año del modelo de vehículo que mas colisiones ha sufrido, y destaca que los modelos de los vehículos del año 2000 son los que mas colusiones han recibido.

Por último analizamos el porcentaje según la distribución de usuario de los vehículos que mas accidentes tienen, destacando que los conductores son los usuarios que sufren la mayor parte de los accidentes, seguidos del resto de pasajeros del vehículo.

A continuación, exploramos la variable de los años de vida del coche restando el año en el que se creó el vehículo al año de colisión y hacemos un gráfico. Se puede apreciar que como hay menos coches con más de 20 años de antigüedad en circulación, tiene sentido que haya menos accidentes en estos coches y por ello hay una tendencia descendiente de la antigüedad de los coches.

Conclusiones generales

Las conclusiones generales que sacamos sobre las dos primeras preguntas es que la persona más típica a tener accidentes son las personas jóvenes nada más sacarse el carnet de conducir y que, sin ser algo exagerado, los hombres son más proclives que las mujeres a tener accidentes. Por otro lado, los que menos accidentes tienen son los más mayores, a partir de los 50 años la cantidad de accidentes van disminuyendo en ese rango de edad. Por lo que los menos propensos a tener colisiones son mujeres mayores de 50. Por el lado de los vehículos, los más habitual es que sean turismos normales, ya que representan el 80% de los accidentes siento solo de media el 70% de los vehículos en circulación. Esto se puede ver en la web del final del cuadro. Y los coches con más accidentes son los que se fabricaron de 1997, al 2006, aunque también son los que más tiempo llevan en la carretera, lo cual tendría sentido. Los vehículos que menos tendencia tienen a tener un accidente con menos de un 1% son los “Fire engine” y “small school bus.” Estos vehículos, al ser un camión de bomberos y un minibús escolar, tiene sentido que sean los menos probables porque la gente es más prudente al verlos y suelen reducir la velocidad lo cual reduce el riesgo de colisión.
https://tc.canada.ca/sites/default/files/migrated/cmvtcs2014_eng.pdf

PREGUNTA 4

¿Qué es lo que más contribuye a que existan fallecimientos en un accidente?

In [16]:
```
dataframe2, test_df = train_test_split (dataframe, test_size= 0.9, random_sta
```

In [17]:
```
def c_hour(valor):
    if valor=='UU' :
        return valor
    elif int(valor)>=22 or int(valor)<= 6 :
        return 'Noche'
    elif int(valor)>6 and int(valor)<= 14 :
        return 'Dia'
```

```
    elif int(valor)>14 and int(valor)< 22 :
        return 'Tarde'

dataframe2["C_HOUR"]=dataframe2['C_HOUR'].apply( c_hour )
```

```
C:\Users\maria\AppData\Local\Temp/ipykernel_62452/7096890.py:11: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataframe2["C_HOUR"]=dataframe2['C_HOUR'].apply( c_hour )
```

In [18]:

```
def ant(colision, vehiculo):
    if  vehiculo == 'UUUU':
        return None
    elif vehiculo == 'NNNN':
        return None
    else:
        return (int(colision) - int(vehiculo))
dataframe2['antiguedad']=dataframe2.apply( lambda x: ant(x.C_YEAR, x.V_YEAR),
```

```
C:\Users\maria\AppData\Local\Temp/ipykernel_62452/647819580.py:8: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataframe2['antiguedad']=dataframe2.apply( lambda x: ant(x.C_YEAR, x.V_YEA
R), axis=1 )
```

In [19]:

```
eliminar = []
for i in distrib['V_ID'].index:
    if distrib['V_ID'].loc[i, "percentage"]< 0.1:
        eliminar.append(i)
eliminar.remove('UU')
```

In [20]:

```
def V_ID(valor):
    if  valor in eliminar:
        return "Otro"
    else:
        return valor
dataframe2['V_ID']=dataframe2.apply(lambda x: V_ID(x.V_ID), axis=1 )
```

```
C:\Users\maria\AppData\Local\Temp/ipykernel_62452/2883444705.py:6: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataframe2['V_ID']=dataframe2.apply(lambda x: V_ID(x.V_ID), axis=1 )
```

In [21]:

```
eliminar = []
for i in distrib['P_ID'].index:
    if distrib['P_ID'].loc[i, "percentage"]< 0.1:
        eliminar.append(i)
def P_ID(valor):
    if  valor in eliminar:
        return "Otro"
    else:
        return valor
dataframe2['P_ID']=dataframe2.apply(lambda x: P_ID(x.P_ID), axis=1 )
```

```
C:\Users\maria\AppData\Local\Temp/ipykernel_62452/70753948.py:10: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataframe2['P_ID']=dataframe2.apply(lambda x: P_ID(x.P_ID), axis=1 )
```

In [22]:

```
def antiguedad(valor):
    if  valor <=3:
        return "Menos de 3"
    elif  (valor >3) and (valor<=6):
        return "Entre 3 y 6"
    elif  (valor >6) and (valor<=10):
        return "Entre 6 y 10"
    elif  (valor >10) and (valor<=15):
        return "Entre 10 y 15"
    elif  (valor >15) and (valor<=20):
        return "Entre 15 y 20"
    else:
        return 'Mas de 20'
dataframe2['antiguedad']=dataframe2.apply(lambda x: antiguedad(x.antiguedad),
```

```
C:\Users\maria\AppData\Local\Temp/ipykernel_62452/401555874.py:14: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataframe2['antiguedad']=dataframe2.apply(lambda x: antiguedad(x.antigueda
d), axis=1 )
```

In [23]:

```
def P_AGE(valor):
    if valor =='NN':
        return valor
    elif valor == "UU":
        return valor
    else:
        valor = int(valor)
        if valor <=18:
            return "Menos de 18"
        elif  (valor >18) and (valor<=25):
            return "Entre 18 y 25"
        elif  (valor >25) and (valor<=40):
            return "Entre 25 y 40"
```

```
        elif  (valor >40) and (valor<=50):
            return "Entre 40 y 50"
        elif  (valor >50) and (valor<=70):
            return "Entre 50 y 70"
        elif valor>70:
            return 'Mas de 70'
        else:
            return valor
dataframe2['P_AGE']=dataframe2.apply(lambda x: P_AGE(x.P_AGE), axis=1 )
```

```
C:\Users\maria\AppData\Local\Temp/ipykernel_62452/3091324318.py:22: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataframe2['P_AGE']=dataframe2.apply(lambda x: P_AGE(x.P_AGE), axis=1 )
```

In [24]:

```
eliminar = []
for i in distrib['C_VEHS'].index:
    if distrib['C_VEHS'].loc[i, "percentage"]< 0.1:
        eliminar.append(i)
def C_VEHS(valor):
    if  valor in eliminar:
        return "Otro"
    else:
        return valor
dataframe2['C_VEHS']=dataframe2.apply(lambda x: C_VEHS(x.C_VEHS), axis=1 )
```

```
C:\Users\maria\AppData\Local\Temp/ipykernel_62452/2678060654.py:10: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataframe2['C_VEHS']=dataframe2.apply(lambda x: C_VEHS(x.C_VEHS), axis=1 )
```

In [25]:

```
dataframe2= dataframe2.drop("V_YEAR",axis=1)
dataframe2 = dataframe2.drop('C_YEAR', axis=1)
dataframe2 = dataframe2.astype({"C_SEV": str}, errors='raise')
```

Primero, creamos un dataframe nuevo (dataframe2) haciendo test y train para poder realizar un análisis predictivo que nos ayude a saber que variables tienen relevancia a la hora de producir un accidente. Una vez hecho la segmentación, vamos seleccionado las variables que más posibilidades tienen a ser relevantes en la causa de un accidente.

Empezamos la hora del día, dado que hay muchas horas lo dividimos por tres intervalos generales siendo la noche (22:00-6:00), día (6:00-14:00) y tarde (14:00-22:00).

A continuación, exploramos la variable de los años de vida del coche restando el año en el que se creó el vehículo al año de colisión y la añadimos a nuestro dataframe.

En las columnas P_ID, V_ID, C_VEHS, habían categorías con muchos valores insignificantes por lo que los que tengan menos de un 0.1%, los ponemos en un categorías nueva llamada “Otros”, agrupándolos para no saturar el dataframe. Segmentamos los vehículos según su edad calculada anteriormente (la nueva columna antigüedad) en 6 intervalos (menor de 3 años, entre 3 y 6 años, entre 6 y 10 años, entre 10 y 15 años y por último más de 20 años). Hacemos algo similar con la edad de la persona afectada por el accidente (menor de 18, entre 18 y 25, entre 25 y 40, entre 40 y 50, entre 50 y 70 y más de 70).

Por último, eliminamos las columnas de C_YEAR y V_YEAR ya que hemos calculado la antigüedad en otra columna llamada “antigüedad” que es la información que realmente importa, las columnas C_YEAR y V_YEAR ya no nos aportan más información. Además, crearía demasiadas columnas en dummies y saturaría el modelo. Finalmente, cambio el tipo de la columna C_SEV a string para luego usarlo en dummies.

En el ejercicio 5 tras hacer el modelo de logistic regression, creo una lista de coeficientes de mayor o menor según la importancia de las variables. Vemos que las primeras 5 variables con las coeficientes más grandes (y por tanto más relevancia a la hora de que se produzcan fallecimientos) son: P_ISEV_3 (Muerte,) V_TYPE_08 (Tractor con o sin remolque), P_ID_Otro, C_CONF_21 (Dos vehículos en movimiento hacia la misma dirección de viaje produciéndose colisión trasera) y C_CONF_31 (Dos vehículos en movimiento con diferente sentido de marcha produciéndose colisión frontal). Los resultados obtenidos tienen mucho sentido ya que la principal causa de un fallecimiento es la muerte (P_ISEV_3) por ello tiene un coeficiente mucho mas alto que el resto, también es coherente que las variables C_CONF_21 y C_CONF_31 sean de las que mas contribuyan a un fallecimiento ya que en el análisis para las preguntas 2 y3 se veía claramente que la mayoría de accidentes eran provocados por este tipo de colisiones. Los tractores tienen una alta probabilidad de colisión porque no están tan preparados ni acostumbrados a salir a la carretera. Van a una velocidad mucho menor y cuando conducen por carretera con otros coches hay más tendencia a que haya un accidente, por lo que tiene sentido que sea una variable con coeficiente alto.

PREGUNTA 5

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.

In [26]:
```
dataframe2.P_ISEV.value_counts()
```

Out[26]:
```
2    307377
1    237233
N     29122
U      8222
3      4086
Name: P_ISEV, dtype: int64
```

In [27]:
```
def P_ISEV(valor):
    if  valor == "2" :
        return "1"
    elif valor =="3":
        return("1")
    elif valor =='1':
        return 0
    else:
        return valor
dataframe2['P_ISEV']=dataframe2.apply(lambda x: P_ISEV(x.P_ISEV), axis=1 )
```

In [28]:
```
dataframe2 = dataframe2.drop(dataframe2[dataframe2.C_SEV == 'U' ].index)
dataframe2 = dataframe2.drop(dataframe2[dataframe2.C_SEV == 'X' ].index)
dataframe2 = dataframe2.drop(dataframe2[dataframe2.P_ISEV == 'U' ].index)
dataframe2 = dataframe2.drop(dataframe2[dataframe2.P_ISEV == 'X' ].index)
dataframe2 = dataframe2.drop(dataframe2[dataframe2.P_ISEV == 'N' ].index)
```

In [29]:
```
dataframe_dummy=pd.get_dummies(dataframe2, columns=dataframe2.columns, drop_f
train_df, test_df = train_test_split (dataframe_dummy, test_size= 0.9, random_
```

Lo primero que hacemos es coger la variable P_ISEV que indica si se necesita tratamiento médico (si se necesita por la gravedad de la lesión, si no se necesita por la poca gravedad de la lesión, y si se necesita por muerte). Esto lo hemos clasificado con 1 siendo necesidad de tratamiento médico y 0 de que no. A continuación, quitamos las categorías que menos aportan dentro de las columnas de P_ISEV y C_SEV siendo U, X y N porque no revelan mucha información. Creamos un dataframe_dummy aplicando dummies a todas las columnas para poder normalizarlas con el objetivo de luego ser capaces de hacer los modelos. A continuación, hacemos train y test sobre este último dataframe.

Modelo para ver si se ha producido un fallecimiento

In [30]:
```
train_df_fatality=train_df.drop('P_ISEV_1', axis=1)
test_df_fatality=test_df.drop('P_ISEV_1', axis=1)
y_train_fatality=train_df_fatality['C_SEV_2']
x_train_fatality=train_df_fatality.drop('C_SEV_2', axis=1)
y_test_fatality=test_df_fatality['C_SEV_2']
x_test_fatality=test_df_fatality.drop('C_SEV_2', axis=1)
```

Logistic Regression

La Regresion Logística es un método estadístico que trata de modelar la probabilidad de una variable cualitativa binaria en función de una o más variables independientes. En este tipo de algoritmo, se realiza una normalización, y cogeremos el conjunto de datos normalizados de train y test donde no escojamos la variable objetivo.

In [31]:
```
log = LogisticRegression(random_state=100)
log.fit(x_train_fatality,y_train_fatality)
```

```
C:\Users\maria\programacion\lib\site-packages\sklearn\linear_model\_logistic.p
y:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
```

Out[31]:
```
LogisticRegression(random_state=100)
```

In [32]:
```
from matplotlib import pyplot
cols=list(x_train_fatality.columns)
importancia = log.coef_[0]
data_importancia=pd.DataFrame([cols,importancia]).transpose()
```

In [33]:
```
data_importancia.columns = ['Variable', 'Coeficiente']
data_importancia['Coeficiente']=data_importancia['Coeficiente'].apply(abs)
data_importancia.sort_values('Coeficiente', ascending=False)
```

Out[33]:

	Variable	Coeficiente
97	C_TRAF_10	1.986854
39	C_CONF_21	1.873347
141	P_ID_05	1.622580
33	C_VEHS_Otro	1.621021
44	C_CONF_31	1.587329
170	P_SAFE_02	1.566872
115	V_ID_05	1.393084
99	C_TRAF_12	1.323871
122	V_TYPE_08	1.264956
109	V_ID_4	1.259942
144	P_ID_Otro	1.242900
174	P_SAFE_13	1.214521
147	P_SEX_U	1.203595
117	V_ID_Otro	1.176422
175	P_SAFE_NN	1.156622
50	C_CONF_41	1.121812
89	C_TRAF_02	1.072416
177	P_SAFE_UU	1.015961
151	P_AGE_Mas de 70	0.987582
140	P_ID_04	0.979858

	Variable	Coeficiente
137	V_TYPE_UU	0.974642
88	C_RALN_U	0.961468
41	C_CONF_23	0.956960
59	C_RCFG_08	0.909010
104	C_TRAF_18	0.906821
55	C_RCFG_04	0.891691
75	C_RSUR_5	0.890407
130	V_TYPE_19	0.880918
142	P_ID_06	0.826769
24	C_VEHS_4	0.799673
43	C_CONF_25	0.799527
171	P_SAFE_09	0.790433
139	P_ID_03	0.784351
54	C_RCFG_03	0.754753
90	C_TRAF_03	0.744132
93	C_TRAF_06	0.737847
121	V_TYPE_07	0.737220
154	P_AGE_UU	0.736624
57	C_RCFG_06	0.728816
32	C_VEHS_06	0.710776
42	C_CONF_24	0.686289
131	V_TYPE_20	0.677512
176	P_SAFE_QQ	0.677291
68	C_WTHR_6	0.676055
105	C_TRAF_QQ	0.673628
30	C_VEHS_04	0.628373
163	P_PSN_96	0.621708
159	P_PSN_23	0.618501
91	C_TRAF_04	0.583576
73	C_RSUR_3	0.568220
74	C_RSUR_4	0.561209
143	P_ID_NN	0.557911
86	C_RALN_6	0.535155

	Variable	Coeficiente
92	C_TRAF_05	0.532131
129	V_TYPE_18	0.529453
101	C_TRAF_15	0.499881
114	V_ID_04	0.497043
72	C_RSUR_2	0.492961
102	C_TRAF_16	0.483985
106	C_TRAF_UU	0.467824
29	C_VEHS_03	0.464463
128	V_TYPE_17	0.459836
180	P_USER_4	0.459836
127	V_TYPE_16	0.457580
108	V_ID_3	0.447874
19	C_HOUR_Noche	0.447867
153	P_AGE_NN	0.445042
52	C_CONF_UU	0.444347
78	C_RSUR_8	0.434916
53	C_RCFG_02	0.433943
71	C_WTHR_U	0.430514
133	V_TYPE_22	0.427012
118	V_ID_UU	0.425905
134	V_TYPE_23	0.421303
138	P_ID_02	0.418999
8	C_MNTH_10	0.415763
23	C_VEHS_3	0.411231
173	P_SAFE_12	0.402777
135	V_TYPE_NN	0.395284
84	C_RALN_4	0.380002
167	P_PSN_NN	0.370964
146	P_SEX_N	0.368395
40	C_CONF_22	0.360653
37	C_CONF_05	0.354758
51	C_CONF_QQ	0.350925
49	C_CONF_36	0.350114

	Variable	Coeficiente
145	P_SEX_M	0.339721
83	C_RALN_3	0.339661
182	P_USER_U	0.335640
98	C_TRAF_11	0.334250
169	P_PSN_UU	0.333594
10	C_MNTH_12	0.328026
113	V_ID_03	0.320022
82	C_RALN_2	0.312827
13	C_WDAY_3	0.312460
103	C_TRAF_17	0.296966
149	P_AGE_Entre 40 y 50	0.285079
47	C_CONF_34	0.273774
5	C_MNTH_7	0.267304
60	C_RCFG_09	0.259686
7	C_MNTH_9	0.253311
77	C_RSUR_7	0.250787
67	C_WTHR_5	0.238479
112	V_ID_02	0.236014
158	P_PSN_22	0.232408
150	P_AGE_Entre 50 y 70	0.223748
165	P_PSN_98	0.222654
64	C_WTHR_2	0.220351
161	P_PSN_32	0.219799
107	V_ID_2	0.219724
95	C_TRAF_08	0.216150
18	C_WDAY_U	0.215572
0	C_MNTH_2	0.210631
120	V_TYPE_06	0.208556
12	C_WDAY_2	0.208124
152	P_AGE_Menos de 18	0.204441
35	C_CONF_03	0.202411
25	C_VEHS_5	0.200318
110	V_ID_99	0.195155

	Variable	Coeficiente
111	V_ID_01	0.193624
38	C_CONF_06	0.193101
6	C_MNTH_8	0.191803
81	C_RSUR_U	0.182884
69	C_WTHR_7	0.179087
21	C_HOUR_UU	0.178825
4	C_MNTH_6	0.173280
162	P_PSN_33	0.173118
62	C_RCFG_QQ	0.167548
66	C_WTHR_4	0.164006
26	C_VEHS_6	0.163376
46	C_CONF_33	0.162278
186	antiguedad_Mas de 20	0.157286
17	C_WDAY_7	0.155588
65	C_WTHR_3	0.155533
155	P_PSN_12	0.154046
28	C_VEHS_02	0.152870
157	P_PSN_21	0.151596
48	C_CONF_35	0.144844
34	C_CONF_02	0.140960
27	C_VEHS_01	0.139554
166	P_PSN_99	0.137596
56	C_RCFG_05	0.136166
70	C_WTHR_Q	0.128534
63	C_RCFG_UU	0.121343
58	C_RCFG_07	0.116595
11	C_MNTH_UU	0.111354
124	V_TYPE_10	0.109346
116	V_ID_99	0.104727
36	C_CONF_04	0.103071
3	C_MNTH_5	0.097809
119	V_TYPE_05	0.097073
1	C_MNTH_3	0.096950

	Variable	Coeficiente
100	C_TRAF_13	0.096300
126	V_TYPE_14	0.096257
181	P_USER_5	0.096257
79	C_RSUR_9	0.093535
184	antiguedad_Entre 3 y 6	0.090900
179	P_USER_3	0.090428
94	C_TRAF_07	0.085543
183	antiguedad_Entre 15 y 20	0.084250
136	V_TYPE_QQ	0.082921
132	V_TYPE_21	0.081284
168	P_PSN_QQ	0.080255
31	C_VEHS_05	0.079988
16	C_WDAY_6	0.073837
9	C_MNTH_11	0.066578
2	C_MNTH_4	0.064758
20	C_HOUR_Tarde	0.063581
187	antiguedad_Menos de 3	0.063009
14	C_WDAY_4	0.062790
160	P_PSN_31	0.058502
156	P_PSN_13	0.055501
61	C_RCFG_10	0.053756
172	P_SAFE_10	0.046672
178	P_USER_2	0.041749
87	C_RALN_Q	0.041177
80	C_RSUR_Q	0.037133
76	C_RSUR_6	0.032618
22	C_VEHS_2	0.030868
85	C_RALN_5	0.025750
123	V_TYPE_09	0.024278
45	C_CONF_32	0.019865
15	C_WDAY_5	0.016845
185	antiguedad_Entre 6 y 10	0.013580
125	V_TYPE_11	0.013579

	Variable	Coeficiente
148	P_AGE_Entre 25 y 40	0.011087
96	C_TRAF_09	0.009268
164	P_PSN_97	0.000000

In [34]:
```
predictions_log=log.predict(x_test_fatality)
acc_log=accuracy_score(y_test_fatality,predictions_log)
acc_log
```

Out[34]:
```
0.9826254943532856
```

Una vez tenemos toda esta información, empezamos a generar un modelo que prediga si habrá fallecimientos o no Creamos un test_df_fatality y un train_df_fatality sobre los train y test hechos previamente, pero sin la variable objetivo P_ISEV_1. Creo unos sub datatsets x e y sobre estos con y sin la variable objetivo C_SEV_2. Hacemos esta modificación para que el modelo tenga más sentido, ya que si ya sabes si el paciente ha fallecido o no, no hace falta meter la variable de si el paciente ha estado hospitalizado o no, ya que la primera variable ya te dice si ha muerto o no. Y generamos un modelo de regresión logística, al ver que el modelo predice tan bien (accuracy=0,98) no hemos ccreido necesario hacer mas modelos. En este ejercicio desarollamos la parte que contesta al ejercico 4 (coeficientes de las variables mas significativas a la hora de producirse un fallecimiento).

Modelo de si va a necesitarse tratamiento médico

In [35]:
```
train_df_injury=train_df.drop('C_SEV_2', axis=1)
test_df_injury=test_df.drop('C_SEV_2', axis=1)
y_train_injury=train_df_injury['P_ISEV_1']
x_train_injury=train_df_injury.drop('P_ISEV_1', axis=1)
y_test_injury=test_df_injury['P_ISEV_1']
x_test_injury=test_df_injury.drop('P_ISEV_1', axis=1)
```

Empezamos a generar un modelo que prediga si se va a necesitar tratamiento médico o no. Creamos un test_df_injury y un train_df_injury sobre los train y test hechos previamente, pero sin la variable objetivo C_SEV_2. Creo unos sub datatsets x e y sobre estos con y sin la variable objetivo P_ISEV_1. Hacemos esta modificación para que el modelo tenga más sentido, ya que si sabes si ya sabes si hay hospitalizados o no, no hace falta meter la variable si ha muerto un paciente ya que si hay muerte obviamente hay un hospitalizado. En este caso hemos querido aplicar mas los conocimientos de clase y por ello hemos usado distintos tipos de modelos de clasificación, tambein hemos utilizado mas modelos ya que gracias a las metricas hemos podido onservar que no eran modelos que predecian la variable objetivo muy bien. En logistic regression y knn solamente utilizamos test en las métricas ya que no se está iterando el modelo. En el resto de modelos que hemos utilizado (Random Forest Classifier, Gradient Boosting, y el Bagging Classifier) si hay iteraciones y por ello puede haber overfitting, por ellos

utilizamos train y test para comparar entre ellos y para ver que la diferencia no es muy grande, así el modelo no se ajusta demasiado a los datos.
Hemos calculado las métricas de cada modelo, las definiciones de las métricas son las mismas para cada modelo, pero para tenerlo claro hemos querido definirlas:

- Accuracy: X, por lo que el modelo es capaz de predecir correctamente un X % de las observaciones del conjunto de test.
- Predict: asigna cada nueva observación a la clase con mayor probabilidad, sin embargo, no dispone de ningún tipo de información sobre la seguridad con la que el modelo realiza esta asignación.
- Precision: X , con esta métrica medimos la calidad del modelo.
- Recall: X , mide la cantidad que el modelo es capaz de identificar.
- F1 score: X , combina las medidas de precision y recall.
- Confusion matrix
- AUC y ROC: X. Se realiza con la funcion predictions_proba() donde se obtiene la probabilidad con la que el modelo considera que cada observación puede pertenecer a cada una de las clases. El AUC es el area bajo la curva del ROC, nos da una idea de cómo funciona el modelo, con un auc del X este algoritmo tiene un X% de probabilidad de que el modelo distinga entre clase positiva y negativa

Logistic Regression

La Regresion Logística es un método estadístico que trata de modelar la probabilidad de una variable cualitativa binaria en función de una o más variables independientes. En este tipo de algoritmo, se realiza una normalización, y cogeremos el conjunto de datos normalizados de train y test donde no escojamos la variable objetivo.

In [36]:
```
log = LogisticRegression(random_state=100)
log.fit(x_train_injury,y_train_injury)
predictions_log=log.predict(x_test_injury)
```

```
C:\Users\maria\programacion\lib\site-packages\sklearn\linear_model\_logistic.p
y:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
```

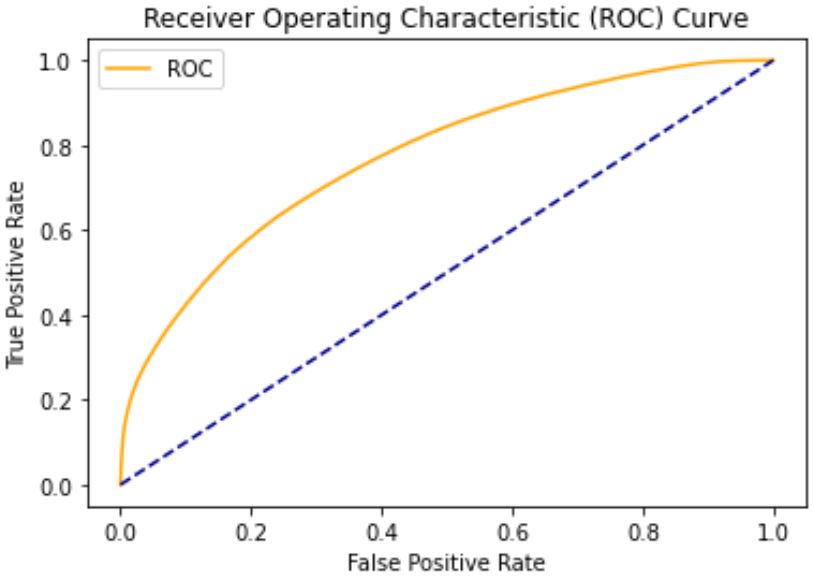
In [37]:
```
acc_log=accuracy_score(y_test_injury,predictions_log)
pre_log=precision_score(y_test_injury,predictions_log)
rec_log=recall_score(y_test_injury,predictions_log)
f1_log=f1_score(y_test_injury,predictions_log)
cm_log=confusion_matrix(y_test_injury,predictions_log)
predictions_proba_log=log.predict_proba(x_test_injury)[:,1]
```

```
auc_log=roc_auc_score(y_test_injury, predictions_proba_log)
fp_log,tp_log, x_log= roc_curve(y_test_injury, predictions_proba_log)

print('Accuracy: '+str(acc_log) )
print('Recall: '+str(rec_log) )
print('Precission: '+str(f1_log) )
print('Precission: '+str(pre_log) )
print('Confussion matrix: ' )
print(cm_log)
print('AUC: ' + str(auc_log))
```

```
Accuracy: 0.699242852253927
Recall: 0.777370440637348
Precission: 0.7458947116289415
Precission: 0.7168686981629714
Confussion matrix:
[[127321  86094]
 [ 62428 217984]]
AUC: 0.7717281241972926
```

In [38]:

```
def plot_roc_curve(fp_log, tp_log):
    plt.plot(fp_log, tp_log, color="orange", label="ROC")
    plt.plot([0, 1], [0, 1], color="darkblue", linestyle="--")
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
plot_roc_curve(fp_log,tp_log)
```



Knn

El K-NN clasifica cada dato nuevo en el grupo que corresponda, según tenga k vecinos más cerca de un grupo o de otro. En este tipo de algoritmo, se realiza una normalización, y cogeremos el conjunto de datos normalizados de train y test donde no escojamos la variable objetivo.

In [40]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=25)
```

```
knn.fit(x_train_injury,y_train_injury)
predictions_knn= knn.predict(x_test_injury)
print(predictions_knn)
```

```
[1 0 1 ... 1 0 0]
```

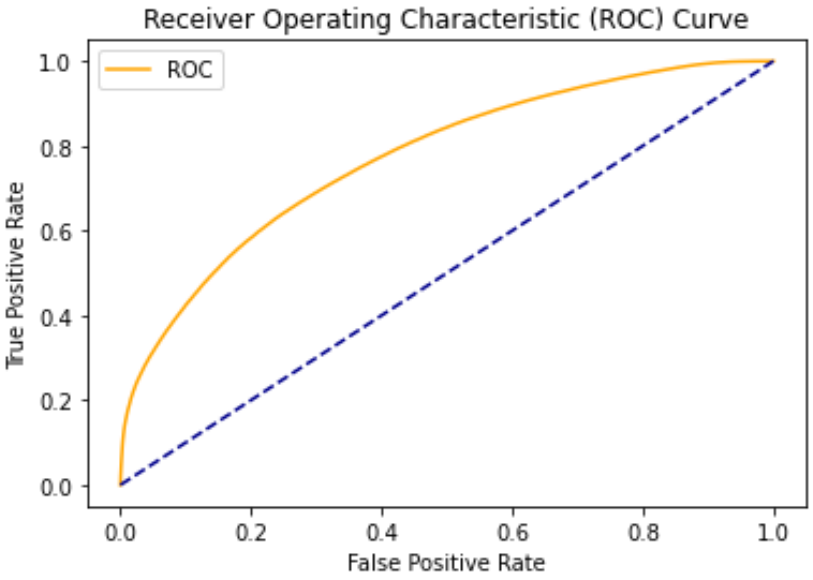
In [41]:

```
acc_knn=accuracy_score(y_test_injury,predictions_knn)
pre_knn=precision_score(y_test_injury,predictions_knn)
rec_knn=recall_score(y_test_injury,predictions_knn)
f1_knn=f1_score(y_test_injury,predictions_knn)
cm_knn=confusion_matrix(y_test_injury,predictions_knn)
predictions_proba_knn=log.predict_proba(x_test_injury)[:,1]
auc_knn=roc_auc_score(y_test_injury, predictions_proba_knn)
fp_knn,tp_knn, x_knn= roc_curve(y_test_injury, predictions_proba_knn)

print('Accuracy: '+str(acc_knn) )
print('Recall: '+str(rec_knn) )
print('Precission: '+str(f1_knn) )
print('Precission: '+str(pre_knn) )
print('Confussion matrix: ' )
print(cm_knn)
print('AUC: ' + str(auc_knn))
```

```
Accuracy: 0.6780552703679629
Recall: 0.7715112049413007
Precission: 0.7312931091306426
Precission: 0.6950603203161395
Confussion matrix:
[[118501  94914]
 [ 64071 216341]]
AUC: 0.7717281241972926
```

In [42]:

```
def plot_roc_curve(fp_knn, tp_knn):
    plt.plot(fp_knn, tp_knn, color="orange", label="ROC")
    plt.plot([0, 1], [0, 1], color="darkblue", linestyle="--")
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
plot_roc_curve(fp_knn,tp_knn)
```



Random Forest

El Random Forest Classifier, se ejeutan varios algoritmos de árbol de deicison en lugar de uno solo.Para clasificar un nuevo objeto basado en atributos, cada arbol de decisión da una clasificación y finalmente la decisión de "votos" es la prediccion del algoritmo.

In [58]:

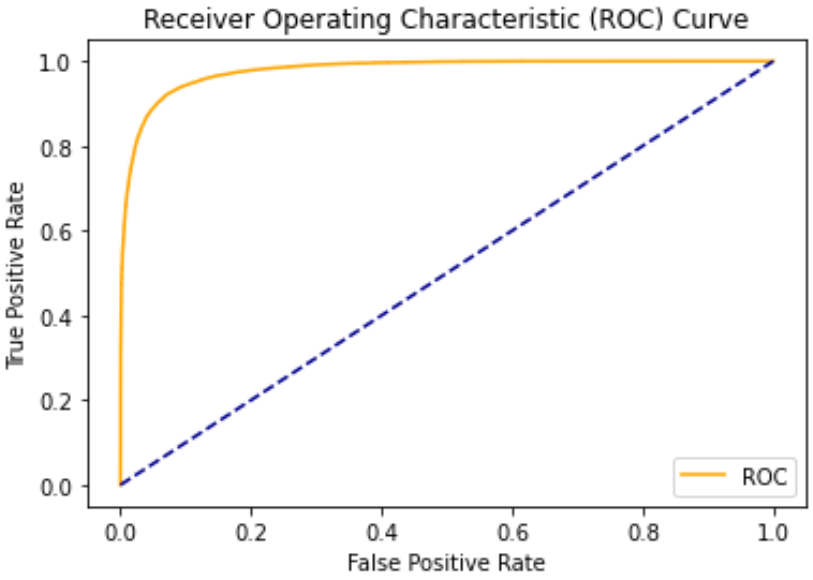
```
from sklearn.ensemble import RandomForestClassifier
n_estimators=[100, 250, 750]

min_samples_split=[10]
for i in min_samples_split:
    print('Mínimo número de elementos por hoja: '+ str(i))
    forest = RandomForestClassifier(n_estimators=750, min_samples_split=i)

    forest.fit(x_train_injury,y_train_injury)
    predictions_forest=forest.predict(x_train_injury)
    acc_forest=accuracy_score(y_train_injury,predictions_forest)
    pre_forest=precision_score(y_train_injury,predictions_forest)
    rec_forest=recall_score(y_train_injury,predictions_forest)
    f1_forest=f1_score(y_train_injury,predictions_forest)
    cm_forest=confusion_matrix(y_train_injury,predictions_forest)
    predictions_proba_forest=forest.predict_proba(x_train_injury)[:,1]
    auc_forest=roc_auc_score(y_train_injury, predictions_proba_forest)
    fp_forest,tp_forest, x_forest= roc_curve(y_train_injury, predictions_proba
    print('Train')
    print('Accuracy : '+str(acc_forest) )
    print('Recall: '+str(rec_forest) )
    print('Precission: '+str(f1_forest) )
    print('Precission: '+str(pre_forest) )
    print('Confussion matrix: ' )
    print(cm_forest)
    print('AUC: ' + str(auc_forest))


    forest.fit(x_train_injury,y_train_injury)
    predictions_forest_test=forest.predict(x_test_injury)
    acc_forest_test=accuracy_score(y_test_injury,predictions_forest_test)
    pre_forest_test=precision_score(y_test_injury,predictions_forest_test)
    rec_forest_test=recall_score(y_test_injury,predictions_forest_test)
    f1_forest_test=f1_score(y_test_injury,predictions_forest_test)
```

```
    cm_forest_test=confusion_matrix(y_test_injury,predictions_forest_test)
    predictions_proba_forest_test=forest.predict_proba(x_test_injury)[:,1]
    auc_forest_test=roc_auc_score(y_test_injury, predictions_proba_forest_tes
    fp_forest_test,tp_forest_test, x_forest_test= roc_curve(y_test_injury, pr

    print('Test')
    print('Accuracy : '+str(acc_forest_test) )
    print('Recall: '+str(rec_forest_test) )
    print('Precission: '+str(f1_forest_test) )
    print('Precission: '+str(pre_forest_test) )
    print('Confussion matrix: ' )
    print(cm_forest_test)
    print('AUC: ' + str(auc_forest_test))
```

```
Mínimo número de elementos por hoja: 10
Train
Accuracy : 0.9215586214438025
Recall: 0.9539789378763969
Precission: 0.9322716686599105
Precission: 0.9115302951041635
Confussion matrix:
[[20943  2875]
 [ 1429 29622]]
AUC: 0.978578239009622
Test
Accuracy : 0.7131849817851191
Recall: 0.7902229576480322
Precission: 0.7578080514626041
Precission: 0.727947674284907
Confussion matrix:
[[130602  82813]
 [ 58824 221588]]
AUC: 0.7888376613677854
```

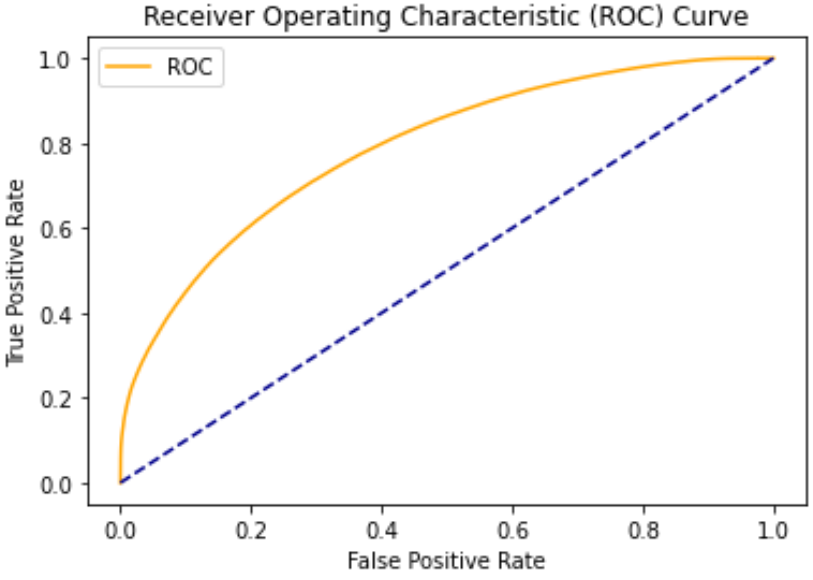
In [59]:

```
def plot_roc_curve(fp_forest,tp_forest):
    plt.plot(fp_forest,tp_forest, color="orange", label="ROC")
    plt.plot([0, 1], [0, 1], color="darkblue", linestyle="--")
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
plot_roc_curve(fp_forest,tp_forest)
```



In [60]:

```
def plot_roc_curve(fp_forest_test,tp_forest_test):
    plt.plot(fp_forest_test,tp_forest_test, color="orange", label="ROC")
    plt.plot([0, 1], [0, 1], color="darkblue", linestyle="--")
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
plot_roc_curve(fp_forest_test,tp_forest_test)
```



Bagging

El Bagging Classifier , está formado por un conjunto de árboles de decisión individuales, se usa committe method, eligiendo la clase más votada entre todos los clasificadores.

In [54]:

```
from  sklearn.ensemble import BaggingClassifier
max_features=[0.3]
for i in max_features:
    print('Numero de variables utilizado: '+ str(i))
    bagging = BaggingClassifier(n_estimators=400,max_features=i )
    bagging.fit(x_train_injury,y_train_injury)
```

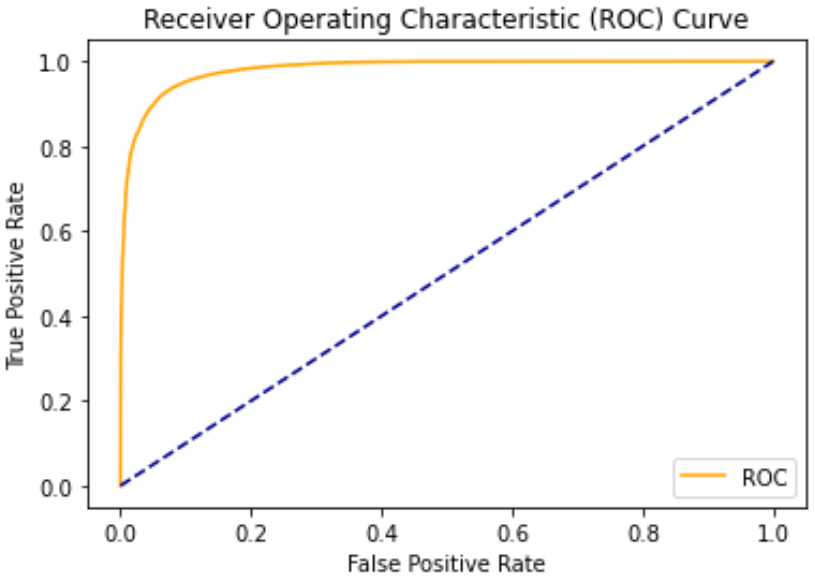
```
    bagging.fit(x_train_injury,y_train_injury)
    predictions_bagging=bagging.predict(x_train_injury)
    acc_bagging=accuracy_score(y_train_injury,predictions_bagging)
    pre_bagging=precision_score(y_train_injury,predictions_bagging)
    rec_bagging=recall_score(y_train_injury,predictions_bagging)
    f1_bagging=f1_score(y_train_injury,predictions_bagging)
    cm_bagging=confusion_matrix(y_train_injury,predictions_bagging)
    predictions_proba_bagging=bagging.predict_proba(x_train_injury)[:,1]
    auc_bagging=roc_auc_score(y_train_injury, predictions_proba_bagging)
    fp_bagging,tp_bagging, x_bagging= roc_curve(y_train_injury, predictions_pr

    print('Train')
    print('Recall: '+str(rec_bagging) )
    print('Precission: '+str(f1_bagging) )
    print('Precission: '+str(pre_bagging) )
    print('Confussion matrix: ' )
    print(cm_bagging)
    print('AUC: ' + str(auc_bagging))

    bagging.fit(x_train_injury,y_train_injury)
    predictions_bagging_test=bagging.predict(x_test_injury)
    acc_bagging_test=accuracy_score(y_test_injury,predictions_bagging_test)
    pre_bagging_test=precision_score(y_test_injury,predictions_bagging_test)
    rec_bagging_test=recall_score(y_test_injury,predictions_bagging_test)
    f1_bagging_test=f1_score(y_test_injury,predictions_bagging_test)
    cm_bagging_test=confusion_matrix(y_test_injury,predictions_bagging_test)
    predictions_proba_bagging_test=bagging.predict_proba(x_test_injury)[:,1]
    auc_bagging_test=roc_auc_score(y_test_injury, predictions_proba_bagging_te
    fp_bagging_test,tp_bagging_test, x_bagging_test= roc_curve(y_test_injury,

    print('Test')
    print('Accuracy : '+str(acc_bagging_test) )
    print('Recall: '+str(rec_bagging_test) )
    print('Precission: '+str(f1_bagging_test) )
    print('Precission: '+str(pre_bagging_test) )
    print('Confussion matrix: ' )
    print(cm_bagging_test)
    print('AUC: ' + str(auc_bagging_test))
```

```
Numero de variables utilizado: 0.3
Train
Recall: 0.9813854626260023
Precission: 0.9239702248298238
Precission: 0.8729017473503294
Confussion matrix:
[[19381  4437]
 [  578 30473]]
AUC: 0.9813080746983425
Test
Accuracy : 0.6982850269426337
Recall: 0.8631656277192131
Precission: 0.7646502253273287
Precission: 0.6863188220048941
Confussion matrix:
[[102790 110625]
 [ 38370 242042]]
AUC: 0.7794030762131104
```

In [55]:

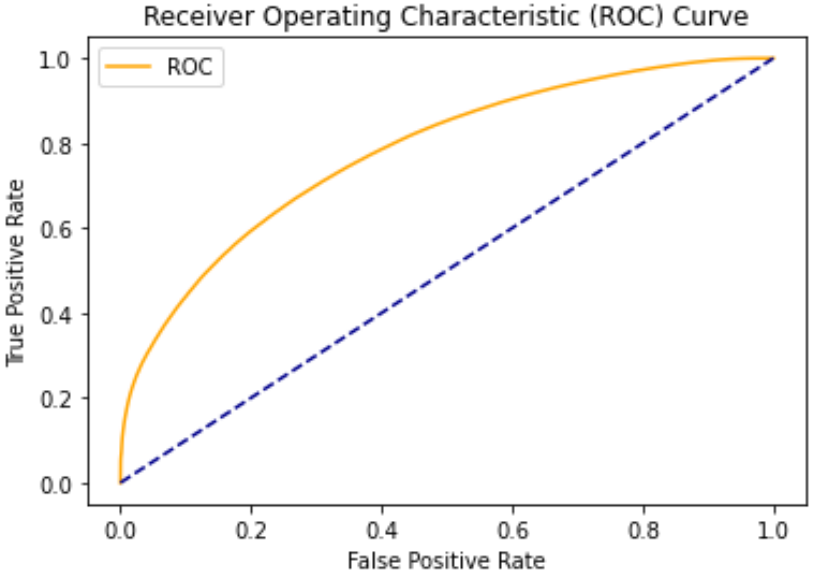
```
def plot_roc_curve(fp_bagging,tp_bagging):
```

```
    plt.plot(fp_bagging,tp_bagging, color="orange", label="ROC")
    plt.plot([0, 1], [0, 1], color="darkblue", linestyle="--")
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
plot_roc_curve(fp_bagging,tp_bagging)
```



In [56]:

```
def plot_roc_curve(fp_bagging_test,tp_bagging_test):
    plt.plot(fp_bagging_test,tp_bagging_test, color="orange", label="ROC")
    plt.plot([0, 1], [0, 1], color="darkblue", linestyle="--")
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
plot_roc_curve(fp_bagging_test,tp_bagging_test)
```



Boosting

El Gradient Boosting, esta formado por un conjunto de árboles de decision individuales,

entrenados de forma secuencial, de forma que cada nuevo árbol trata de mejorar los errores de los árboles anteriores.

In [51]:

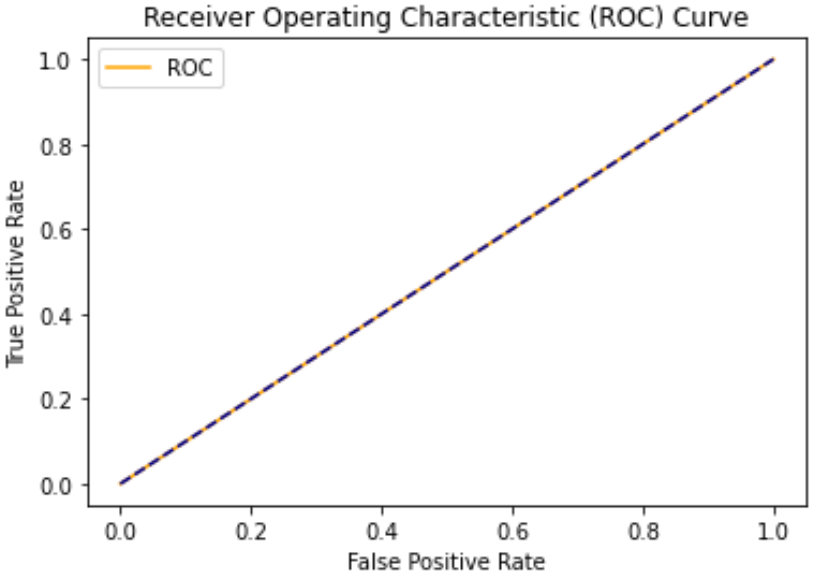
```
from sklearn.ensemble import GradientBoostingClassifier
subsample= [0.3]
for i in subsample:
    print('Numero de variables utilizado: '+ str(i))
    boosting = GradientBoostingClassifier(n_estimators=300, min_samples_leaf=
    boosting.fit(x_train_injury,y_train_injury)
    predictions_boosting=bagging.predict(x_train_injury)
    acc_boosting=accuracy_score(y_train_injury,predictions_boosting)
    pre_boosting=precision_score(y_train_injury,predictions_boosting)
    rec_boosting=recall_score(y_train_injury,predictions_boosting)
    f1_boosting=f1_score(y_train_injury,predictions_boosting)
    cm_boosting=confusion_matrix(y_train_injury,predictions_boosting)
    predictions_proba_boosting=bagging.predict_proba(x_train_injury)[:,1]
    auc_boosting=roc_auc_score(y_train_injury, predictions_proba_boosting)
    fp_boosting,tp_boosting, x_boosting= roc_curve(y_train_injury, predictions

    print('Train')
    print('Accuracy : '+str(acc_boosting) )
    print('Recall: '+str(rec_boosting) )
    print('Precission: '+str(f1_boosting) )
    print('Precission: '+str(pre_boosting) )
    print('Confussion matrix: ' )
    print(cm_boosting)
    print('AUC: ' + str(auc_boosting))

    boosting.fit(x_train_injury,y_train_injury)
    predictions_boosting_test=bagging.predict(x_test_injury)
    acc_boosting_test=accuracy_score(y_test_injury,predictions_boosting_test)
    pre_boosting_test=precision_score(y_test_injury,predictions_boosting_test
    rec_boosting_test=recall_score(y_test_injury,predictions_boosting_test)
    f1_boosting_test=f1_score(y_test_injury,predictions_boosting_test)
    cm_boosting_test=confusion_matrix(y_test_injury,predictions_boosting_test
    predictions_proba_boosting_test=bagging.predict_proba(x_test_injury)[:,1]
    auc_boosting_test=roc_auc_score(y_test_injury, predictions_proba_boosting_
    fp_boosting_test,tp_boosting_test, x_boosting_test= roc_curve(y_test_inju

    print('Test')
    print('Accuracy : '+str(acc_boosting_test) )
    print('Recall: '+str(rec_boosting_test) )
    print('Precission: '+str(f1_boosting_test) )
    print('Precission: '+str(pre_boosting_test) )
    print('Confussion matrix: ' )
    print(cm_boosting_test)
    print('AUC: ' + str(auc_boosting_test))
```

```
Numero de variables utilizado: 0.3
C:\Users\maria\programacion\lib\site-packages\sklearn\metrics\_classification.
py:1248: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
due to no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
Train
Accuracy : 0.43408846525360406
Recall: 0.0
Precission: 0.0
```

```
Precission: 0.0
Confussion matrix:
[[23818     0]
 [31051     0]]
AUC: 0.5
C:\Users\maria\programacion\lib\site-packages\sklearn\metrics\_classification.
py:1248: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
due to no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
Test
Accuracy : 0.4321655154537925
Recall: 0.0
Precission: 0.0
Precission: 0.0
Confussion matrix:
[[213415      0]
 [280412      0]]
AUC: 0.5
```

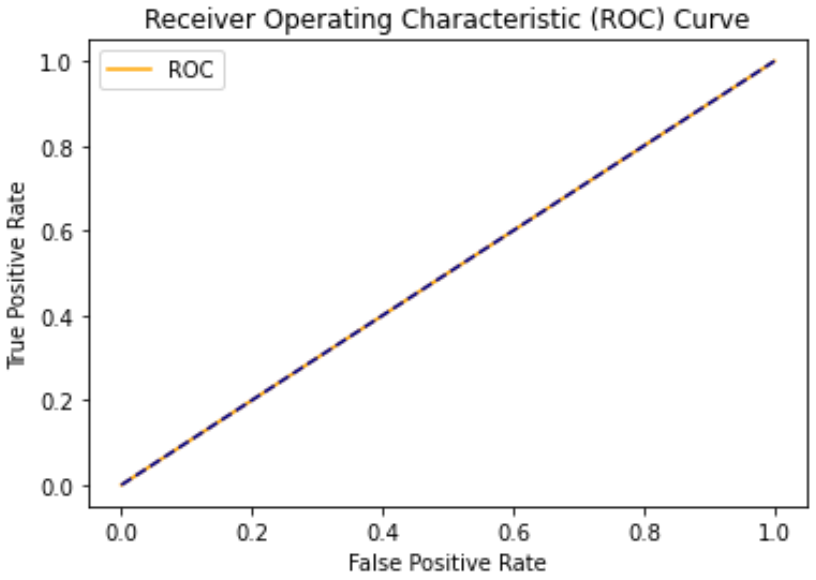
In [52]:

```
def plot_roc_curve(fp_boosting,tp_boosting):
    plt.plot(fp_boosting,tp_boosting, color="orange", label="ROC")
    plt.plot([0, 1], [0, 1], color="darkblue", linestyle="--")
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
plot_roc_curve(fp_boosting,tp_boosting)
```



In [53]:

```
def plot_roc_curve(fp_boosting_test,tp_boosting_test):
    plt.plot(fp_boosting_test,tp_boosting_test, color="orange", label="ROC")
    plt.plot([0, 1], [0, 1], color="darkblue", linestyle="--")
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
plot_roc_curve(fp_boosting_test,tp_boosting_test)
```



Tras analizar todos los modelos se ve claramente que los modelos que mejor predicen de mayor a menor son bagging, random forest, knn, logistic regression y por último boosting

PREGUNTA 6

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).

En un análisis de valor podrías analizar temporalmente los accidentes que puedan ocurrir en un futuro a través de un modelo basado en los meses anteriores. Por ejemplo si quieres predecir los accidentes que habría en los meses de verano (junio, julio, agosto) de 2015, crearías un modelo a partir de las características de los accidentes de los meses junio ,julio, agosto de los años anteriores (2012,2013,2014), No tenemos claro cómo se haría este modelo por lo que no hemos resuelto con código esta pregunta.