Minería de datos y modelización predictiva

Hugo Gómez Sabucedo
hugogomezsabucedo@gmail.com

Máster Big Data, Data Science & Inteligencia Artificial

Curso 2024-2025

Universidad Complutense de Madrid

Índice

1. Introducción **3**

2. Importación de datos y análisis descriptivo **3**

2.1. Importación 3

2.2. Análisis descriptivo 4

3. Corrección de errores **6**

4. Análisis de valores atípicos **8**

5. Análisis de valores perdidos **8**

6. Detección de relaciones entre variables **10**

7. Regresión lineal **12**

7.1. Selección clásica 12

7.2. Selección aleatoria 14

7.3. Modelo ganador 16

8. Regresión logística **17**

8.1. Selección clásica 17

8.2. Selección aleatoria 17

8.3. Modelo ganador 17

1. Introducción

En esta práctica se nos pide, a partir de un archivo con datos sobre diferentes resultados electorales, seleccionar unas variables, con el objetivo de construir tanto un modelo de regresión lineal, a partir de una variable objetivo continua; como un modelo de regresión logística, a partir de una variable binaria. Para ello, se deberán asignar correctamente los tipos de los datos y realizar un análisis de los mismos, con el objetivo inicial de depurarlos, para asegurarnos que no tenemos variables incoherentes o que no se ajusten al modelo. A continuación, se corregirán los errores que se hayan detectado, así como los valores atípicos y perdidos. Una vez con los datos limpios, podremos analizar las relaciones entre las distintas variables input y las variables objetivo, para poder proceder con la creación de los dos modelos solicitados, para cada una de las variables.

Cada registro del archivo viene identificado por Name y CodigoProvincia (ya que, observando el conjunto de datos, se han encontrado municipios con el mismo nombre pero en provincias diferentes). Adicionalmente, contienen información sobre la CCAA a la que pertenecen. Por otra parte, tenemos las variables objetivo, que se dividen en las variables continuas (AbstentionPtge, el porcentaje de abstención; IdaPct, el porcentaje de votos a partidos de izquierdas; DchaPct, el porcentaje de votos a partidos de derechas; y OtrosPct, el porcentaje de votos a otros partidos); y las variables binarias o dicotómicas (AbstencionAlta, que vale 1 si el porcentaje de abstención es mayor al 30 % o 0 en otro caso; Izquierda, que toma valor 1 si la suma de votos a los partidos de izquierda es superior a la suma de votos a derchas y otros, y 0 en caso contrario; y Derecha, análoga a la anterior, pero para partidos de derechas). De estas, se escogerá **AbstentionPtge** para realizar la regresión lineal; e **Izquierda** para realizar la regresión logística.

Por otra parte, tenemos un conjunto de 29 variables explicativas, las cuales no entraremos a explicar en detalle, pero que se corresponden con aspectos demográficos o sociológicos de los distintos municipios, como puede ser el porcentaje de población por tramos de edad, el porcentaje de desempleo por edades o sectores, el número de empresas de los municipios por tipo de actividad y la actividad principal del mismo, el porcentaje de población respecto a su CCAA y provincia de nacimiento, o, evidentemente, el censo, población, superficie y densidad del municipio.

De esta forma, nuestro objetivo será construir un modelo de regresión lineal para la variable AbstentionPtge y un modelo de regresión logística para la variable Izquierda, que nos permitan en el primer caso predecir el porcentaje de abstención, y en el segundo caso, la probabilidad de que los partidos de izquierdas sean los más votados.

2. Importación de datos y análisis descriptivo

2.1. Importación

Para importar los datos, una vez establecido el directorio de trabajo a la carpeta correspondiente con `os.chdir`, se emplea la siguiente instrucción:

```
    datos = pd.read_excel("DatosElecciones.xlsx", sheet_name='
    DatosEleccionesEspaña')
```

Lo primero que haremos será definir el ID de los datos, que es el nombre del ayuntamiento y el código de la provincia, mediante la instrucción:

```
datos.set_index(["Name", "CodigoProvincia"], inplace=True, drop=True)
```

Mediante `datos.head(5)` podemos ver las 5 primeras filas, y con `datos.dtypes` podemos ver los tipos de datos de las variables, lo que usaremos para comprobar que cada una de las variables tiene asignado el tipo que le corresponde (es decir, numérica o categórica). Se observa que únicamente se han importado como categóricas (tipo `Object`) las variables Name, CCAA, ActividadPpal y Densidad, mientras que el resto se han importado bien como int o bien como float. Debemos, por tanto, transformar a categóricas las tres variables objetivo binarias: Izquierda, Derecha y AbstencionAlta, mediante la función `astype(str)`. A continuación, crearemos una lista con todas las variables, y las dividiremos en numéricas y categóricas.

```
1   numsACategoricas = ['Izquierda', 'Derecha', 'AbstencionAlta']
2   for v in numsACategoricas:
3       datos[v] = datos[v].astype(str)
4
5   variables = list(datos.columns)
6
7   numericas = datos.select_dtypes(include=['int', 'int32', 'int64','
    float', 'float32', 'float64']).columns
8   categoricas = [v for v in variables if v not in numericas]
```

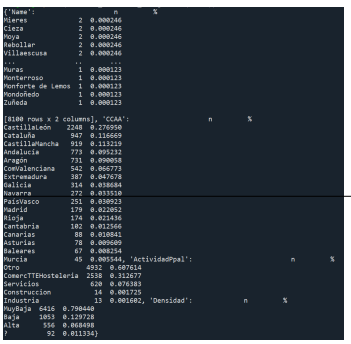
```
    Izquierda          int64
    Derecha            int64
    AbstencionAlta     int64
```

Podemos hacer de nuevo un print de 'categoricas' para comprobar que las variables están asignadas correctamente:

```
    ['Name', 'CCAA', 'AbstencionAlta', 'Izquierda', 'Derecha', '
    ActividadPpal', 'Densidad']
```

Puesto que todas las variables están ya correctamente asignadas, podemos pasar directamente al análisis descriptivo de las mismas.

2.2. Análisis descriptivo



A continuación haremos el análisis descriptivos de las variables explicativas, comenzando en primer lugar por las categóricas.

Como podemos ver, hay algunos municipios que se repiten dos veces, pero esto es debido a que existe el mismo municipio

en dos provincias (por ejemplo, Mieres en Asturias, y Mieres (o Mieras) en Girona). Por otra parte, en la variable CCAA, tenemos múltiples comunidades con una frecuencia inferior al 5 %, que se considera que están poco representadas. Tendremos que reagruparlas en el apartado 3. Una situación similar nos encontramos con ActividadPal, donde las categorías Construcción e Industria tienen una frecuencia de 0.001 ambas. Las reagruparemos con otra categoría. Respecto a Densidad, la única categoría poco representada es *?*, que nos puede indicar que se trataría de valores perdidos.

Analizamos también los valores únicos de cada una de las variables, mediante el siguiente código, y vemos que los valores de las CCAA y la ActividadPpal son los correctos y los esperados, que las variables Izquierda, Derecha y AbstencionAlta solo toman valores 1 y 0, y confirmamos el caso anterior de que la variable Densidad tiene valores perdidos.

```
for cat in categoricas:
    print(datos[cat].unique())
```

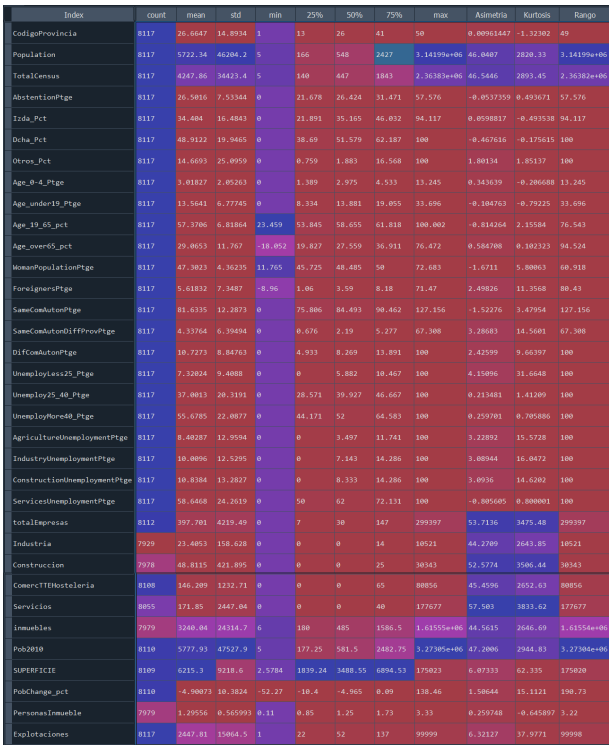
```
['Abadía' 'Abertura' 'Acebo' ... 'Zarzosa de Río Pisuerga' 'Zazuar'
'Zuñeda']
['Extremadura' 'Andalucía' 'ComValenciana' 'CastillaMancha' 'Galicia'
'Cataluña' 'PaísVasco' 'Aragón' 'CastillaLeón' 'Rioja' 'Madrid' 'Murcia'
'Navarra' 'Asturias' 'Canarias' 'Cantabria' 'Baleares']
['0' '1']
['1' '0']
['0' '1']
['Otro' 'ComercTTEHosteleria' 'Servicios' 'Construccion' 'Industria']
['MuyBaja' '?' 'Baja' 'Alta']
```

Analizamos ahora las variables numéricas. En primer lugar, analizamos los valores distintos con `cuentaDistintos`, donde vemos que todas las variables tienen un número suficiente de datos. La que menos valores distintos tiene es PersonasInmueble con 282 valores, pero es un valor esperable. Empleamos la función `describe` para hallar valores como la media o los cuartiles, ampliándola con la asimetría, la curtosis y el rango.

```
descriptivos_num = datos.describe().T
for num in numericas:
    descriptivos_num.loc[num, "Asimetria"] = datos[num].skew()
    descriptivos_num.loc[num, "Kurtosis"] = datos[num].kurtosis()
    descriptivos_num.loc[num, "Rango"] = np.ptp(datos[num].dropna().
    values)
```

Podemos ver que las variables Poblacion, TotalCensus, inmuebles, Pob2010 y totalEmpresas presentan un rango muy elevado, lo cual tiene sentido debido a ciudades como Madrid o Barcelona tienen una población muy elevada, lo cual impacta en las otras cuatro variables mencionadas. Podríamos pensar en aplicar alguna transformación, por ejemplo logarítmica, para reducir el impacto de estas anomalías, pero he considerado que estos aportan un valor, ya que estos municipios con población elevada deberían de alguna forma poder tener más peso. Por otra parte, se observan algunas variables que, siendo porcentajes, bien tienen valores mínimos negativos, como Age_over65_pct y ForeignersPtge, bien tienen valores máximos mayores que

100, comoAge_19_65_pct o SameComAutonPtge. Este caso de porcentajes negativos o mayores que 100 se encuentra también para PobChange_pct, pero aquí sí que tiene sentido, ya que pueden indicar o un decrecimiento de la población o un aumento muy grande de la misma. Por último, se ve que el valor máximo de Explotaciones es de 99999, lo cual nos podría indicar que esta variable tiene algún valor perdido.



Index	count	mean	std	min	25%	50%	75%	max	Asimetría	Kurtosis	Rango
CodigoProvincia	8117	26.6647	14.8934	1	13	26	41	50	0.00961447	-1.32302	49
Population	8117	5722.34	46204.2	5	166	548	2427	3.14199e+06	46.0407	2820.33	3.14199e+06
TotalCensus	8117	4247.86	34423.4	5	140	447	1843	2.36383e+06	46.5446	2893.45	2.36382e+06
AbstentionPtge	8117	26.5016	7.53344	0	21.678	26.424	31.471	57.576	-0.0537359	0.493671	57.576
Izda_Pct	8117	34.404	16.4843	0	21.891	35.165	46.032	94.117	0.0598817	-0.493538	94.117
Dcha_Pct	8117	48.9122	19.9465	0	38.69	51.579	62.187	100	-0.467616	-0.175615	100
Otros_Pct	8117	14.6693	25.0959	0	0.759	1.883	16.568	100	1.80134	1.85137	100
Age_0-4_Ptge	8117	3.01827	2.05263	0	1.389	2.975	4.533	13.245	0.343639	-0.206688	13.245
Age_under19_Ptge	8117	13.5641	6.77745	0	8.334	13.881	19.055	33.696	-0.104763	-0.79225	33.696
Age_19_65_pct	8117	57.3706	6.81864	23.459	53.845	58.655	61.818	100.002	-0.814264	2.15584	76.543
Age_over65_pct	8117	29.0653	11.767	-18.052	19.827	27.559	36.911	76.472	0.584708	0.102323	94.524
WomanPopulationPtge	8117	47.3023	4.36235	11.765	45.725	48.485	50	72.683	-1.6711	5.80063	60.918
ForeignersPtge	8117	5.61832	7.3487	-8.96	1.06	3.59	8.18	71.47	2.49826	11.3568	80.43
SameComAutonPtge	8117	81.6335	12.2873	0	75.806	84.493	90.462	127.156	-1.52276	3.47954	127.156
SameComAutonDiffProvPtge	8117	4.33764	6.39494	0	0.676	2.19	5.277	67.308	3.28683	14.5601	67.308
DifComAutonPtge	8117	10.7273	8.84763	0	4.933	8.269	13.891	100	2.42599	9.66397	100
UnemployLess25_Ptge	8117	7.32024	9.4088	0	0	5.882	10.467	100	4.15096	31.6648	100
Unemploy25_40_Ptge	8117	37.0013	20.3191	0	28.571	39.927	46.667	100	0.213481	1.41209	100
UnemployMore40_Ptge	8117	55.6785	22.0877	0	44.171	52	64.583	100	0.259701	0.705886	100
AgricultureUnemploymentPtge	8117	8.40287	12.9594	0	0	3.497	11.741	100	3.22892	15.5728	100
IndustryUnemploymentPtge	8117	10.0096	12.5295	0	0	7.143	14.286	100	3.08944	16.0472	100
ConstructionUnemploymentPtge	8117	10.8384	13.2827	0	0	8.333	14.286	100	3.0936	14.6202	100
ServicesUnemploymentPtge	8117	58.6468	24.2619	0	50	62	72.131	100	-0.805605	0.800001	100
totalEmpresas	8112	397.701	4219.49	0	7	30	147	299397	53.7136	3475.48	299397
Industria	7929	23.4053	158.628	0	0	0	14	10521	44.2709	2643.85	10521
Construccion	7978	48.8115	421.895	0	0	0	25	30343	52.5774	3506.44	30343
ComercTTEHosteleria	8108	146.209	1232.71	0	0	0	65	80056	45.4596	2652.63	80056
Servicios	8055	171.85	2447.04	0	0	0	40	177677	57.503	3833.62	177677
inmuebles	7979	3240.04	24314.7	6	180	485	1586.5	1.61555e+06	44.5615	2646.69	1.61554e+06
Pob2010	8110	5777.93	47527.9	5	177.25	581.5	2482.75	3.27305e+06	47.2006	2944.83	3.27304e+06
SUPERFICIE	8109	6215.3	9218.6	2.5784	1839.24	3488.55	6894.53	175023	6.07333	62.335	175020
PobChange_pct	8110	-4.90073	10.3824	-52.27	-10.4	-4.965	0.09	138.46	1.50644	15.1121	190.73
PersonasInmueble	7979	1.29556	0.565993	0.11	0.85	1.25	1.73	3.33	0.259748	-0.645897	3.22
Explotaciones	8117	2447.81	15064.5	1	22	52	137	99999	6.32127	37.9771	99998

Figura 1: Análisis descriptivo de variables numéricas

Por otra parte, analizamos también los valores faltantes, y nos encontramos que las siguientes variables presentan algún valor faltante, pero en ningún caso un número tan elevado como para que nos podamos plantear descartarlas.

```
totalEmpresas          5
Industria            188
Construccion         139
ComercTTEHosteleria    9
Servicios             62
inmuebles            138
Pob2010                7
SUPERFICIE             8
PobChange_pct          7
PersonasInmueble     138
```

Una vez hemos analizado todos los datos y detectado todos los errores posibles (seguramente haya más, pero evidentemente es imposible detectar todos los errores que presentan los datos), podremos pasar a corregirlos.

3. Corrección de errores

En relación a las variables categóricas, como vimos en el apartado anterior, únicamente debemos realizar reagrupaciones. Por una parte, reagruparemos las categorías Construccion e Industria de la variable *ActividadPpal*, en la categoría Otro, debido a las bajas frecuencias observadas para estas categorías. Esto se hace mediante el siguiente código:

```
datos['ActividadPpal'] = datos['ActividadPpal'].replace(\{'Construccion':
 'Otro', 'Industria': 'Otro'\})
```

Respecto a la variable *CCAA*, no podemos agrupar aleatoriamente, sino que debemos hacerlo teniendo en cuenta aquellas que presenten características similares. En este caso ya que estudiaremos la variable AbstentionPtge, hemos decidido tener este parámetro en cuenta para realizar la agrupación, obteniendo los resultados que se ven en la figura 2.

```
promedio_abstencion = datos.groupby('CCAA')['AbstentionPtge'].mean().
reset_index()
```

Las agrupaciones que realizaremos serán entonces: Canarias, Asturias, Baleares y País Vasco; Galicia y Navarra; Murcia, Cantabria y Extremadura; Madrid y Aragón; y Rioja y la Comunidad Valenciana. Si ahora volvemos a analizar la frecuencia de los valores para la variable, vemos en la figura 3 que la nueva ĆCAAcon menos frecuencia es la agrupación de Canarias, Asturias, Baleares y País Vasco, con un 5.96 %.

CCAA	AbstentionPtge
Rioja	19.049741
ComValenciana	21.888273
CastillaMancha	22.698995
CastillaLeón	23.822925
Aragón	25.033557
Madrid	25.072363
Extremadura	26.541096
Cantabria	26.880235
Murcia	27.383689
Andalucía	28.702028
Navarra	30.024482
Galicia	30.870268
PaísVasco	31.964825
Baleares	33.574701
Asturias	33.762987
Cataluña	34.286721
Canarias	34.843398

Figura 2: CCAA agrupadas por abstención

	n	%
CastillaLeón	2248	0.276950
Cataluña	947	0.116669
CastillaMancha	919	0.113219
Madrid-Aragón	910	0.112110
Andalucía	773	0.095232
Rioja-ComValenciana	716	0.088210
Galicia-Navarra	586	0.072194
Mur-Can-Ext	534	0.065788
Can-Ast-Bal-PV	484	0.059628

Figura 3: CCAA tras reagrupación

Lo siguiente que haremos será sustituir los valores que encontramos como perdidos (el 99999 que encontramos en Explotaciones), así como algún posible NaN que pueda haber en los datos y que no hayamos detectado.

```
for v in categoricas:
    datos[v] = datos[v].replace('nan', np.nan)

datos['Explotaciones'] = datos['Explotaciones'].replace(99999,np.nan)
```

Por último, corregiremos los errores que encontramos con los porcentajes, indicando que se pasen a valores perdidos aquellos donde el porcentaje no esté entre 0 y 100.

```
datos['Age_over65_pct'] = [x if 0<=x<=100 else np.nan for x in datos['
Age_over65_pct']]
datos['ForeignersPtge'] = [x if 0<=x<=100 else np.nan for x in datos['
ForeignersPtge']]
datos['Age_19_65_pct'] = [x if 0<=x<=100 else np.nan for x in datos['
Age_19_65_pct']]
datos['SameComAutonPtge'] = [x if 0<=x<=100 else np.nan for x in datos['
SameComAutonPtge']]
```

Una vez hecho esto, definimos las variables objetivo `varObjCont = datos[‘AbstentionPtge‘]` y `varObjBin = datos[‘Izquierda‘]`, y eliminamos todas las variables objetivo de los datos, mediante la funcion `.drop` . A partir de esto, renombramos

los datos como datos_input, y creamos las variables numericas_input y categoricas_input con la división en categóricas y numéricas de las variables.

4. Análisis de valores atípicos

Para tratar los valores atípicos, en primer lugar, mediremos la proporción de valores missing que hay por cada variable, y tratándolos con la función atipicosAmissing, empleando para ello el código:

```
resultados = {x: atipicosAmissing(datos_input[x])[1] / len(datos_input)
for x in numericas_input}
for x in numericas_input:
    datos_input[x] = atipicosAmissing(datos_input[x])[0]
```

Así, nos encontramos con los siguientes porcentajes de valores atípicos que, como podemos observar, se encuentran preesentes en la mayoría de variables. Cabe destacar el elevado número de valores atípicos en, por ejemplo, Population o TotalCensus que, como comentamos en el 2.2, se debe a ciudades con mucha población, y que en este caso serán tratados como atípicos y, por tanto, convertidos a missing. Otra opción podría haber sido hacer una transformación logarítmica de los datos, pero se observó que, aun realizándola, se seguían detectando numerosos valores atípicos, por lo que lo más adecuado parece dejar que sean tratados directamente como atípicos.

```
{'Population': 0.05297523715658495, 'TotalCensus': 0.050880867315510656,
'Age_0-4_Ptge': 0.0, 'Age_under19_Ptge': 0.0, 'Age_19_65_pct':
0.00012319822594554638, 'Age_over65_pct': 0.0, 'WomanPopulationPtge':
0.0, 'ForeignersPtge': 0.006036713071331773, 'SameComAutonPtge':
0.00012319822594554638, 'SameComAutonDiffProvPtge':
0.0024639645189109276, 'DifComAutonPtge': 0.00012319822594554638, '
UnemployLess25_Ptge': 0.0, 'Unemploy25_40_Ptge': 0.0, '
UnemployMore40_Ptge': 0.0, 'AgricultureUnemploymentPtge':
0.0013551804854010103, 'IndustryUnemploymentPtge': 0.0, '
ConstructionUnemploymentPtge': 0.0, 'ServicesUnemploymentPtge': 0.0, '
totalEmpresas': 0.10484169027965998, 'Industria': 0.09239866945915978, '
Construccion': 0.0903042996180855, 'ComercTTEHosteleria':
0.09868177898238266, 'Servicios': 0.11851669335961562, 'inmuebles':
0.09042749784403105, 'Pob2010': 0.0974497967229272, 'SUPERFICIE':
0.027103609708020206, 'PobChange_pct': 0.0012319822594554638, '
PersonasInmueble': 0.0, 'Explotaciones': 0.050880867315510656}
```

5. Análisis de valores perdidos

En el caso de los valores perdidos, lo primero que haremos será obtener la matriz de correlación de valores ausentes, mediante la función proporcionada `patron_perdidos`, a la que se realizó una modificación para eliminar los valores numéricos de la misma.

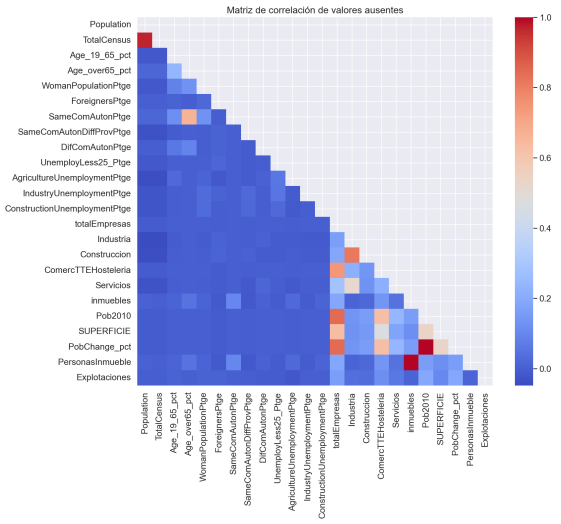


Figura 4: Matriz de valores perdidos

CCAA	0
Population	804
TotalCensus	782
Age_0-4_Ptge	0
Age_under19_Ptge	0
Age_19_65_pct	24
Age_over65_pct	3
WomanPopulationPtge	21
ForeignersPtge	653
SameComAutonPtge	3
SameComAutonDiffProvPtge	165
DifComAutonPtge	40
UnemployLess25_Ptge	26
Unemploy25_40_Ptge	0
UnemployMore40_Ptge	0
AgricultureUnemploymentPtge	162
IndustryUnemploymentPtge	48
ConstructionUnemploymentPtge	53
ServicesUnemploymentPtge	0
totalEmpresas	5
Industria	188
Construccion	139
ComercTTEHosteleria	9
Servicios	62
ActividadPpal	0
inmuebles	138
Pob2010	7
SUPERFICIE	8
Densidad	0
PobChange_pct	7
PersonasInmueble	138
Explotaciones	189

Figura 5: Valores perdidos por variable

Como podemos ver, se observa una gran correlación entre TotalCensus y Población, Inmuebles y PersonasInmueble y Pob2010 y PobChange_pct. Esto puede tener sentido, ya que son variables que están relacionadas entre si. También se observa un gran número de valores perdidos en ForeignersPtge, concretamente 653, aunque esto podría tener explicación en algunos municipios que no tengan ninguna persona extranjera censada. Se observa también un número elevado de valores perdidos en las variables AgricultureUnemploymentPtge, SameComAutonDiffProvPtge, Industria, Construccion o Explotaciones, todos los cuales tendremos que tratar posteriormente. Sin embargo, si analizamos la proporción de valores missing por variable (prop_missingsVars), vemos que las que más valores perdidos tienen son, naturalmente, Population y TotalCensus, con un 9.905 % y un 9.634 % respectivamente, seguidas de ForeignersPtge con un 8.044 % y luego las otras variables comentadas, con apenas un 2 %. Ninguna supera, por tanto, el 50 % de valores missing con los que se eliminaría la variable.

```
datos_input[variables_input].isna().sum()
prop_missingsVars = datos_input.isna().sum()/len(datos_input)

#Media de valores perdidos para cada una de las filas
datos_input['prop_missings'] = datos_input.isna().mean(axis = 1)
datos_input['prop_missings'].describe()
len(datos_input['prop_missings'].unique())

# Transformamos a categorica
datos_input["prop_missings"] = datos_input["prop_missings"].astype(str)
variables_input.append('prop_missings')
categoricas_input.append('prop_missings')
```

Por otra parte, se crea la columna *prop_missings*, que representa el porcentaje de valores perdidos por cada una de las filas. Si hacemos un describe de esta variable, vemos que la que más valores perdidos tiene presenta un 33.33 % de valores perdidos, mientras que la mediana

es apenas un 3.03 %. Ya que ninguno de los registros no tiene tampoco más de un 50 % de datos faltantes, vemos que no es necesario eliminar ninguno. Lo que si realizaremos es un análisis de los valores únicos de esta nueva columna creada, ya que se crea como continua, pero podría tener sentido que fuese categórica. Y, efectivamente, vemos que tiene solamente 9 valores únicos, por lo que, como se indica, se transforma a categórica esta variable.

Una vez ya hemos analizado los valores missing, tenemos que ver como vamos a tratarlos. Para ello, se nos proporcionan dos funciones en el script de clase, `ImputacionCuant` e `ImputacionCuali`. Por ello, solamente tendremos que aplicarlas a las variables, por una parte a las variables numéricas, y por otra parte a las variables categóricas.

```
for x in numericas_input:
    datos_input[x] = ImputacionCuant(datos_input[x], 'aleatorio')

for x in categoricas_input:
    datos_input[x] = ImputacionCuali(datos_input[x], 'aleatorio')
```

Lo último que nos queda por hacer es revisar, con `datos_input.isna().sum()`, que no quede ningún valor faltante. Aunque no se incluya aquí la salida de este comando, se comprueba que hay 0 valores faltantes en todas las variables, por lo que podemos considerar que ya tenemos nuestros datos limpios. El último paso será volcarlos a un archivo pickle, de forma que podamos guardar una copia de estos datos para recuperarlos en cualquier momento que queramos, sin tener que volver a ejecutar el código anterior nuevamente. Para ello, empleamos el siguiente código:

```
datosEleccionesDep = pd.concat([varObjBin, varObjCont, datos_input], axis
 = 1)
with open('datosEleccionesDep.pickle', 'wb') as archivo:
    pickle.dump(datosEleccionesDep, archivo)
```

6. Detección de relaciones entre variables

Antes de comenzar a realizar los modelos de regresión, debemos analizar las relaciones entre las variables, para ver si hay casos de variables altamente correlacionadas que debamos eliminar, y hacer una preselección de las variables que serán las más significativas. Esto se haría también para analizar si la relación entre las variables no es lineal y poder transformarlas, pero en este caso se nos pide en el enunciado del ejercicio que esto no se haga. Para esto, primeramente, se analizará el valor del estadístico V de Cramer, que nos permite capturar las relaciones entre las variables, tanto lineales como no lineales. Es un estadístico acotado entre 0 y 1, donde un valor próximo a 1 indica una asociación perfecta entre los datos, por lo que nos interesará quedarnos con aquellas variables con una V de Cramer más elevada.

Como se ve, la variable que más relación tiene con ambas variables objetivo es la CCAA, mientras que hay un número considerable de ellas que tiene un valor de la V de Cramer inferior a 0.1, en caso de la binaria, o de 0.15, en caso de la continua. De momento, no realizaremos nada con las variables, pero estos dos gráficos nos servirán para, al momento de seleccionar las variables para ambas regresiones, ver cuáles debemos descartar o no.

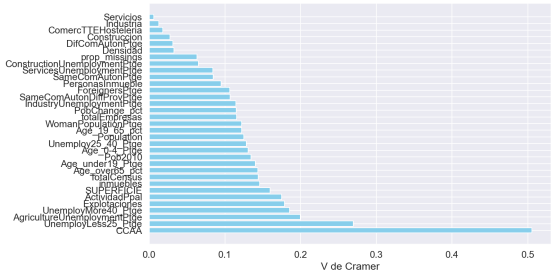


Figura 6: V de Cramer para la variable binaria

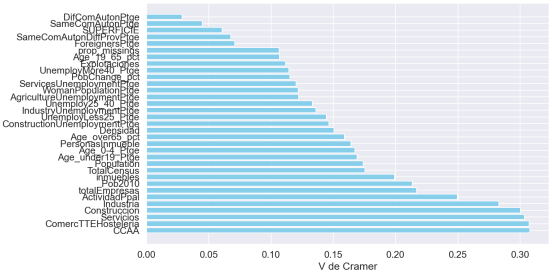
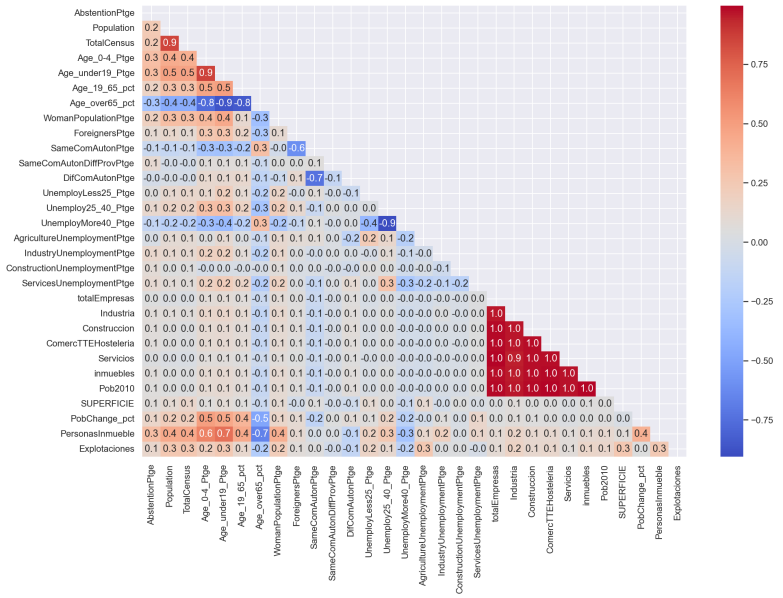


Figura 7: V de Cramer para la variable continua

Lo que sí haremos en este punto es dibujar la matriz de correlación entre las variables continuas, para ver si hay alguna variable que debamos eliminar. Lo primero que observamos es que tenemos numerosas variables con un valor de 1 en esta matriz, lo cual nos indica una correlación perfecta entre las mismas. Son, concretamente, las variables TotalEmpresas, Industria, Construccion, ComercTTEHosteleria, Servicios, Inmuebles y Pob2010. Esto se debe a que dichas variables se refieren al número de empresas de x tipo que hay en el municipio, por lo que es de esperar que estén relacionadas no sólo con TotalEmpresas, sino también entre sí. Además, como es de esperar, encontramos una alta relación entre Population y totalCensus, así como entre Age_under19_Ptge y Age_0-4_Pctge, y Age_over65_pct, que presenta una elevada correlación negativa con el resto de variables referidas a los porcentajes de edad. Todas estas variables no serán tenidas en cuenta a la hora de pasar las variables a los modelos y, además, serán eliminadas. Concretamente, eliminaremos Population, TotalEmpresas, Age_under19_Ptge y Age_over65_pct.



Una vez hemos realizado esto y eliminado estas variables para evitar que creen problemas de multicolinealidad, crearemos dos subconjuntos a partir de los datos:

```
todo_cont = pd.concat([datos_input, varObjCont], axis = 1)
todo_bin = pd.concat([datos_input, varObjBin], axis = 1)
```

En uno de ellos, añadimos a los datos la variable objetivo continua, y al otro le añadimos la variable binaria. Estos serán los datos que empleemos para crear los modelos de regresión.

7. Regresión lineal

Antes de comenzar a realizar la selección de variables, debemos dividir los datos en un conjunto de entrenamiento y un conjunto de test, usando el 20 % de los datos como test (y el 80 % como entrenamiento).

```
x_train, x_test, y_train, y_test = train_test_split(todo_cont, varObjCont
, test_size = 0.2, random_state = 29112002)
```

7.1. Selección clásica

Debido a la capacidad computacional, a la correlación de variables estudiada en 6 y a los valores de la V de Cramer, es necesario descartar ciertas variables para la hora de crear el modelo. Como se explicó, se descartarán aquellas variables con correlación entre ellas, variables que puedan obtenerse de otras (por ejemplo, TotalCensus se obtiene de Population), y las que tienen un valor de la V de Cramer más pequeño. De esta forma, las variables seleccionadas son las siguientes:

```
1 var_cont = ['TotalCensus', 'WomanPopulationPtge', 'UnemployLess25_Ptge',
  'Unemploy25_40_Ptge', 'UnemployMore40_Ptge', 'AgricultureUnemploymentPtge
  ', 'IndustryUnemploymentPtge', 'ConstructionUnemploymentPtge', '
  ServicesUnemploymentPtge', 'PobChange_pct', 'PersonasInmueble', '
  Explotaciones']
2 var_categ = ['CCAA', 'ActividadPpal', 'Densidad']
```

A continuación, se crean las interacciones entre las variables. Como se nos indica que únicamente debemos considerar las interacciones entre variables continuas, la lista de interacciones será igual a la de variables continuas, pero podrían añadirse también las categóricas, lo cual probablemente daría también más robustez al modelo resultante. A partir de esta lista, se crea una lista con las interacciones únicas, dos a dos, de las variables. Esta será la que empleemos para la creación del modelo.

```
1 interacciones = var_cont #+ var_categ
2 interacciones_unicas = list(itertools.combinations(interacciones, 2))
```

Una vez creadas las interacciones, emplearemos las funciones de lm para crear los modelos para la selección clásica de variables. En este caso, se crearán 6 modelos: los resultantes de aplicar los métodos stepwise, backward y forward, empleando los criterios AIC y BIC. A modo de breve resumen, el método forward se caracteriza por crear un modelo añadiendo variables una a una, hasta que no haya ninguna variable que mejore el modelo; el backward comienza con todas las variables y va eliminando las menos significativas, hasta que solo queden aquellas significativas; y el stepwise es una combinación de los anteriores, iniciando con un modelo vacío y agregando variables más significativas, pero revisando tras cada adición si alguna variable ha dejado de serlo. Este es el que proporciona un mejor equilibrio entre un modelo con muchas variables y uno que sea simple y eficiente. Respecto a los criterios AIC/BIC, el primero favorece modelos más complejos, mientras que el BIC penaliza más los modelos con más parámetros.

```
modeloStepAIC = lm_stepwise(y_train, x_train, var_cont, var_categ,
interacciones_unicas, 'AIC')
modeloBackAIC = lm_backward(y_train, x_train, var_cont, var_categ,
interacciones_unicas, 'AIC')
modeloForwAIC = lm_forward(y_train, x_train, var_cont, var_categ,
interacciones_unicas, 'AIC')
modeloStepBIC = lm_stepwise(y_train, x_train, var_cont, var_categ,
interacciones_unicas, 'BIC')
modeloBackBIC = lm_backward(y_train, x_train, var_cont, var_categ,
interacciones_unicas, 'BIC')
modeloForwBIC = lm_forward(y_train, x_train, var_cont, var_categ,
interacciones_unicas, 'BIC')
```

Una vez hemos obtenido los 6 modelos, debemos elegir el ganador de entre ellos. Para ello, tenemos la tabla 1, que la hemos obtenido con el código que se muestra a continuación (variando para cada modelo). En general, podemos observar unos valores de R^2 bastante bajos, lo que significa que el modelo no logra tener una capacidad predictiva demasiado elevada. Esto podría deberse a varios motivos, como que algunas de las variables que hemos seleccionado no tengan especial relación con la variable objetivo, que se hayan excluido variables que sí son relevantes, o que la relación entre las variables no sea lineal, o incluso que se hayan cometido algunos errores en el proceso de limpieza de datos que lleven a la presencia de ruido en los mismos.

```
Rsq(modelo['Modelo'], y_train, modelo['X']) # R2 de train
x_test = crear_data_modelo(x_test,modelo['Variables']['cont'], modelo['
Variables']['categ'], modelo['Variables']['inter'])
Rsq(modelo['Modelo'], y_test, x_test) # R2 de test
len(modelo['Modelo'].params) # Número de parámetros
```

Con los datos de los que disponemos, debemos elegir el modelo que equilibre mejor el ajuste (R^2 train), la generalización (R^2 test) y un menor número de parámetros, ya que un número elevado de los mismos podría dar lugar a un sobreajuste del modelo. Analizando únicamente el R^2, podríamos pensar que el mejor modelo es el *Backward AIC*, ya que tiene un R^2 de train más elevado. Sin embargo, vemos también que es el que presenta una mayor diferencia entre el R^2 de train y el R^2 de test, así como el mayor número de parámetros con diferencia. Por otra parte, los modelos de *Stepwise* y *Forward* con BIC son los que presentan un menor R^2 para los datos de test, con lo que, pese a ser los modelos más parsimoniosos, son los que peor capacidad

Método	Métrica	R^2 test	R^2 train	Nº parámetros
Backward	AIC	0.392081	0.336811	56
Backward	BIC	0.383429	0.335341	35
Stepwise	AIC	0.382744	0.333387	39
Stepwise	BIC	0.369415	0.324226	18
Forward	AIC	0.382683	0.332820	40
Forward	BIC	0.369415	0.324227	18

Cuadro 1: Comparación de modelos de regresión lineal

explicativa tienen, por lo que se descartan. El resto de modelos (*Backward BIC*, *Stepwise AIC* y *Forward AIC*) presentan tanto un R^2 de test como de train muy similares, con lo cual balancean bien el ajuste con la generalización. Por tanto, para discriminar entre ellos, se elije el que menor número de parámetros tiene, que en este caso es el **Backward BIC**, ya que mantendrá la mejor de las capacidades predictivas de todos los modelos analizados, siendo a la vez fácilmente interpretable por el bajo número de variables y disminuyendo el riesgo de un sobreajuste.

7.2. Selección aleatoria

Por otra parte, se nos pide realizar una selección aleatoria de las variables. Esto implica generar varias submuestras de forma aleatoria, para realizar sobre ellas una selección de variables clásica y generar un resumen de los modelos que se han generado, de forma que se elegirán los modelos que hayan sido generados más veces, ya que son más estables y buenos candidatos a que se les aplique la validación cruzada.

Para la selección aleatoria, el código empleado es el que se muestra a continuación, en el que se realizan 30 iteraciones para generar los modelos. Se ha elegido el método stepwise, porque es el más completo, ya que combina las ventajas del método forward con las del backward, permitiendo "replantearse" si una variable debe entrar al modelo o no con cada iteración

```
variables_seleccionadas = {'Formula': [],'Variables': []}

for x in range(20):
    print('---------------------------- iter: ' + str(x))

    # Dividir los datos de entrenamiento en conjuntos de entrenamiento y prueba.
    x_train2, x_test2, y_train2, y_test2 = train_test_split(x_train, y_train, test_size = 0.3, random_state = 29112002 + x)

    # Realizar la selección stepwise utilizando el criterio BIC en la submuestra.
    modelo = lm_stepwise(y_train2.astype(int), x_train2, var_cont, var_categ, interacciones_unicas, 'BIC')

    # Almacenar las variables seleccionadas y la fórmula correspondiente.
    variables_seleccionadas['Variables'].append(modelo['Variables'])
```

```
    variables_seleccionadas['Formula'].append(sorted(modelo['Modelo'].
    model.exog_names))

variables_seleccionadas['Formula'] = list(map(lambda x: '+'.join(x),
variables_seleccionadas['Formula']))
```

Una vez generados los modelos, podemos calcular la frecuencia de cada fórmula, para con ello ver cuales son los dos modelos más frecuentes y sus correspondientes variables.

```
frecuencias = Counter(variables_seleccionadas['Formula'])
frec_ordenada = pd.DataFrame(list(frecuencias.items()), columns = ['
Formula', 'Frecuencia'])
frec_ordenada = frec_ordenada.sort_values('Frecuencia', ascending = False
).reset_index()

# Identificar las dos modelos más frecuentes y las variables
correspondientes.
var_1 = variables_seleccionadas['Variables'][variables_seleccionadas['
Formula'].index(frec_ordenada['Formula'][0])]
var_2 = variables_seleccionadas['Variables'][variables_seleccionadas['
Formula'].index(frec_ordenada['Formula'][1])]
```

De esta forma, podemos ver que las salidas de los modelos que más se repiten son las siguientes para el modelo 1 y el modelo 2:

```
{'cont': [],
 'categ': ['CCAA', 'ActividadPpal'],
 'inter': [('Unemploy25_40_Ptge', 'UnemployMore40_Ptge'),  ('
 ConstructionUnemploymentPtge', 'Explotaciones'),  ('
 ConstructionUnemploymentPtge', 'ServicesUnemploymentPtge'), ('
 UnemployLess25_Ptge', 'ServicesUnemploymentPtge'),  ('
 ConstructionUnemploymentPtge', 'PersonasInmueble'),  ('
 UnemployMore40_Ptge', 'IndustryUnemploymentPtge')]
}
```

```
{'cont': [],
'categ': ['CCAA', 'ActividadPpal'],
'inter': [('Unemploy25_40_Ptge', 'UnemployMore40_Ptge'), ('
ConstructionUnemploymentPtge', 'Explotaciones'), ('UnemployLess25_Ptge',
'ServicesUnemploymentPtge'), ('ConstructionUnemploymentPtge', '
ServicesUnemploymentPtge'), ('Unemploy25_40_Ptge', '
ConstructionUnemploymentPtge'), ('UnemployMore40_Ptge', 'PersonasInmueble
'), ('UnemployMore40_Ptge', 'IndustryUnemploymentPtge'), ('
UnemployLess25_Ptge', 'PobChange_pct')]
}
```

7.3. Modelo ganador

Una vez hemos determinado el modelo ganador de la selección clásica, y obtenido los dos modelos que más se repiten en la selección aleatoria, realizaremos la validación cruzada para determinar el modelo ganador de entre los tres. Para ello, se realizan 50 ejecuciones de un bucle en el que aplicamos la función proporcionada `validacion_cruzada_lm` a los tres modelos, y mostramos los resultados en un boxplot para determinar el ganador.

```
results = pd.DataFrame({'Rsquared': [], 'Resample': [], 'Modelo': []})

# Realizamos la validación cruzada
for rep in range(50):
    modelo1 = validacion_cruzada_lm(5, x_train, y_train, ganador['Variables']['cont']
                , ganador['Variables']['categ'], ganador['Variables']['inter'])
    modelo2 = validacion_cruzada_lm(5, x_train, y_train, var_1['cont'], var_1['categ'], var_1['inter'])
    modelo3 = validacion_cruzada_lm(5, x_train, y_train, var_2['cont'], var_2['categ'], var_2['inter'])

    results_rep = pd.DataFrame({
        'Rsquared': modelo1 + modelo2 + modelo3
        , 'Resample': ['Rep' + str((rep + 1))]*5*3
        , 'Modelo': [1]*5 + [2]*5 + [3]*5
    })
    results = pd.concat([results, results_rep], axis = 0)

# Dibujamos el boxplot
plt.figure(figsize=(10, 6))
plt.grid(True)
grupo_metrica = results.groupby('Modelo')['Rsquared']
boxplot_data = [grupo_metrica.get_group(grupo).tolist() for grupo in grupo_metrica.groups]
plt.boxplot(boxplot_data, labels=grupo_metrica.groups.keys())
plt.xlabel('Modelo')
plt.ylabel('Rsquared')
plt.show()
```

Como se puede ver en el boxplot, el claro ganador es el modelo 1, es decir, el que obtuvimos de la selección clásica, ya que, aunque sí es cierto que presenta un variabilidad mayor, si observamos el número de parámetros vemos que es de 35 (`len(ganador['Modelo'].params)`), mientras que para los otros dos (`len(frec_ordenada['Formula'][0].split('+'))` y `len(frec_ordenada['Formula'][1].split('+'))`) es de 490 y 565, por lo que esta ligera mejora en la variabilidad no justifica en absoluto el aumento desmedido del número de parámetros.

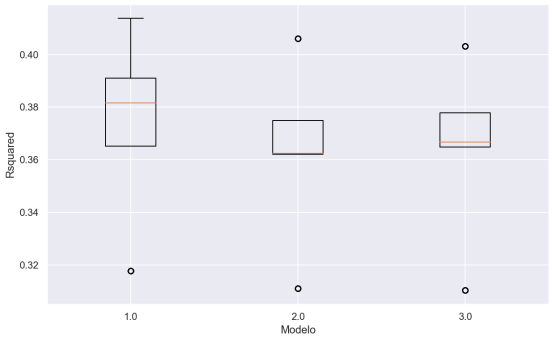


Figura 8: Validación cruzada para regresión lineal

POR QUE GANADOR? CALIDAD? INTERPRETAR COEFICIENTES DE DOS VARIABLES (BINARIA Y CONTINUA)

8. Regresión logística

8.1. Selección clásica

Método	**Métrica**	R^2 **test**	R^2 **train**	**Nº parámetros**
Backward	AIC			
Backward	BIC			
Stepwise	AIC			
Stepwise	BIC			
Forward	AIC			
Forward	BIC			

Cuadro 2: Comparación de modelos de regresión lineal

8.2. Selección aleatoria

8.3. Modelo ganador