Análisis descriptivo y predictivo de las Elecciones Generales Españolas de 2023

Minería de datos y Modelización Predictiva

Bartolomé Mestre Fons

Miércoles, 18 de Diciembre de 2024

Minería de datos y Modelización Predictiva
2024-2025
UCM - Universidad Complutense de Madrid

Índice

1. Introducción

El objetivo del presente trabajo es la aplicación de los modelos de regresión lineal y logística sobre los datos recaptados por localidad de las Elecciones Generales Españolas de 2023 para la predicción del éxito de los partidos de Derecha. Para ello, se realizará un análisis descriptivo de las variables recaptadas en la base de datos, estipulando su naturaleza así como sus parámetros estadísticos principales. A continuación, se procede a la depuración de los datos (corrección de datos erróneos, análisis de valores nulos y atípicos, etc.), para acto seguido seguir con el desarrollo de los modelos de regresión lineal y logística incluyendo interacciones entre las variables continuas desde diferentes planteamientos: desde la perspectiva de métodos de selección clásica de variables (planteamiento *backward*, *forward* y *stepwise*) así como desde una perspectiva de selección de variables por frecuencia con la metodolgía que ofrezca un mejor rendimiento a priori. De entre los modelos mencionados, se realizará una selección del modelo ganador basándose en parámetros estadísticos que evaluen la calidad del mismo. Finalmente, se analizará la fiabilidad del modelo y se plantearán cuestiones a tener en cuenta para futuras mejoras del mismo.

2. Análisis descriptivo de la base de datos

La base de datos corresponde a los resultados electorales de las Elecciones Generales Españolas de 2023. Registra 36 características de cada entrada de datos de un total de 8117 localidades de España. La variable objetivo para el modelo de regresión lineal es Dcha_Pct, y para el modelo de ajuste logístico es Derecha (entonces las variables objetivo opcionales como AbstentionPtge (el porcentaje de abstención de voto), Izda_Pct (porcentaje de voto de los partidos de izquierda), Otros_Pct (porcentaje de voto de aquellos partidos alternativos), AbstencionAlta (variable dicotómica que indica si la abstención de voto ha sido alta o no) e Izquierda (variable dicotómica que indica si los partidos de izquierda fueron mayoritarios) no se tomarán como variables explicativas de los modelos y no se analizarán). Las variables recaptadas son las siguientes:

- **Name:** Nombre de la localidad. Esta variable identifica el municipio y se comporta como un identificador único de cada localidad (omitir aquellas localidades con mismo nombre pero diferente CodigoProvincia). Para los análisis posteriores se omitirá ya que no aporta información relevante sobre las estadísticas electorales de la misma. Formato esperado: categórico.
- **CodigoProvincia:** Código numérico de la provincia. Es útil para agrupar localidades dentro de las mismas provincias y realizar análisis a nivel provincial. Aquí no se recogen las ciudades autónomas de Ceuta y Melilla. Formato esperado: numérico discreto.
- **CCAA:** Comunidad Autónoma a la que pertenece la localidad. Permite clasificar y analizar los resultados según las regiones administrativas más amplias de España. Formato esperado: categórico.
- **Population:** Población total de la localidad. Variable clave para interpretar el alcance y la representatividad de los resultados electorales. Formato esperado: numérico discreto.
- **TotalCensus:** Total de personas censadas. Refleja el tamaño del electorado potencial y su relación con la participación electoral. Formato esperado: numérico discreto.
- **Dcha_Pct:** Porcentaje de votos obtenidos por partidos de derecha. Indica la proporción de apoyo electoral hacia esta ideología en la localidad. Debe de hallarse entre 0 y 100. Formato esperado: numérico continuo.
- **Derecha:** Variable binaria que indica si los partidos de derecha fueron mayoritarios. Es útil para clasificaciones rápidas del comportamiento electoral. Formato esperado: binario.
- **Age_0-4_Ptge:** Porcentaje de la población entre 0 y 4 años. Refleja la estructura demográfica y puede correlacionarse con patrones de desarrollo en la localidad. Debe de hallarse entre 0 y 100. Formato esperado: numérico continuo.
- **Age_under19_Ptge:** Porcentaje de la población menor de 19 años. Representa la proporción de jóvenes, relevante para el análisis de sostenibilidad demográfica. Debe de hallarse entre 0 y 100 y sumar 100 junto a Age_19_65_pct y Age_over65_pct. Formato esperado: numérico continuo.
- **Age_19_65_pct:** Porcentaje de población entre 19 y 65 años. Indica la población en edad laboral, clave para interpretar dinámicas económicas y sociales. Debe de hallarse entre 0 y 100 y sumar 100 junto a Age_under19_Ptge y Age_over65_pct. Formato esperado: numérico continuo.

- **Age_over65_pct:** Porcentaje de la población mayor de 65 años. Útil para evaluar el envejecimiento demográfico y su impacto en los resultados. Debe de hallarse entre 0 y 100 y sumar 100 junto a Age_19_65_pct y Age_under19_Ptge. Formato esperado: numérico continuo.
- **WomanPopulationPtge:** Porcentaje de mujeres en la población. Permite analizar diferencias de género en la distribución demográfica.Debe de hallarse entre 0 y 100. Formato esperado: numérico continuo.
- **ForeignersPtge:** Porcentaje de población extranjera votante. Puede influir en patrones electorales y económicos. Debe de hallarse entre 0 y 100 y sumar 100 junto a SameComAutonPtge y DifComAutonPtge. Formato esperado: numérico continuo.
- **SameComAutonPtge:** Porcentaje de población que ha realizado el voto en la misma comunidad autónoma. Refleja la movilidad interna y cohesión regional. Debe de hallarse entre 0 y 100 y sumar 100 junto a ForeignersPtge y DifComAutonPtge. Formato esperado: numérico continuo.
- **SameComAutonDiffProvPtge:** Porcentaje de población que ha realizado el voto en la misma comunidad autónoma pero diferente provincia. Indica migración intrarregional. Debe de hallarse entre 0 y 100. Formato esperado: numérico continuo.
- **DifComAutonPtge:** Porcentaje de población que ha realizado el voto en una comunidad autónoma diferente a la de la localidad. Muestra niveles de movilidad interregional. Debe de hallarse entre 0 y 100 y sumar 100 junto a SameComAutonPtge y ForeignersPtge. Formato esperado: numérico continuo.
- **UnemployLess25_Ptge:** Porcentaje de desempleados menores de 25 años. Refleja el desempleo juvenil, un indicador clave de vulnerabilidad económica. Debe de hallarse entre 0 y 100 y sumar 100 junto a Unemploy25_40_Ptge y UnemployMore40_Ptge. Formato esperado: numérico continuo.
- **Unemploy25_40_Ptge:** Porcentaje de desempleados entre 25 y 40 años. Ayuda a evaluar la situación económica de la población en edad productiva temprana. Debe de hallarse entre 0 y 100 y sumar 100 junto a UnemployLess25_Ptge y UnemployMore40_Ptge. Formato esperado: numérico continuo.
- **UnemployMore40_Ptge:** Porcentaje de desempleados mayores de 40 años. Indica desafíos económicos para la población en edad avanzada. Debe de hallarse entre 0 y 100 y sumar 100 junto a UnemployLess25_Ptge y Unemploy25_40_Ptge. Formato esperado: numérico continuo.
- **AgricultureUnemploymentPtge:** Porcentaje de desempleados en el sector agrícola. Relevante en áreas rurales con economías basadas en la agricultura. Formato esperado: numérico continuo.
- **IndustryUnemploymentPtge:** Porcentaje de desempleados en el sector industrial. Indicador del impacto de la desindustrialización. Debe de hallarse entre 0 y 100. Formato esperado: numérico continuo.
- **ConstructionUnemploymentPtge:** Porcentaje de desempleados en el sector de la construcción. Útil para analizar efectos de ciclos económicos en este sector. Debe de hallarse entre 0 y 100. Formato esperado: numérico continuo.
- **ServicesUnemploymentPtge:** Porcentaje de desempleados en el sector servicios. Refleja dinámicas económicas en áreas urbanas y turísticas. Debe de hallarse entre 0 y 100. Formato esperado: numérico continuo.
- **totalEmpresas:** Número total de empresas registradas. Indica el dinamismo económico local. Debe ser positivo. Formato esperado: numérico discreto.
- **Industria:** Número de empresas en el sector industrial. Permite evaluar la base económica de la localidad. Debe ser positivo. Formato esperado: numérico discreto.
- **Construccion:** Número de empresas en el sector de la construcción. Relacionado con la actividad económica y el desarrollo urbano. Debe ser positivo. Formato esperado: numérico discreto.
- **ComercTTEHosteleria:** Número de empresas en comercio, transporte y hostelería. Refleja el peso del sector terciario en la economía local. Debe ser positivo. Formato esperado: numérico discreto.
- **Servicios:** Número de empresas en el sector servicios. Indicador del desarrollo del sector servicios. Debe ser positivo. Formato esperado: numérico discreto.
- **ActividadPpal:** Actividad económica principal de la localidad. Clasifica la economía local en términos generales entre 'Otro', 'ComercTTEHosteleria', 'Servicios', 'Construccionó 'Industria'. Formato esperado: categórico.

- **inmuebles:** Número total de inmuebles en la localidad. Relacionado con la densidad de población y el desarrollo urbano. Debe ser positivo. Formato esperado: numérico discreto.
- **Pob2010:** Población de la localidad en 2010. Permite calcular cambios demográficos a lo largo del tiempo. Debe de ser positivo. Formato esperado: numérico discreto.
- **SUPERFICIE:** Superficie de la localidad en kilómetros cuadrados. Útil para evaluar la densidad de población. Debe ser positivo. Formato esperado: numérico continuo.
- **Densidad:** Descripción de la densidad de población. Indica la distribución espacial de la población como 'Muy baja', 'Baja', 'Altaó '?'para valaroes nulos. Formato esperado: categórico.
- **PobChange_pct:** Cambio porcentual de población entre 2010 y 2023. Refleja tendencias demográficas. Puede ser tanto positivo como negativo. Formato esperado: numérico continuo.
- **PersonasInmueble:** Número promedio de personas por inmueble. Ayuda a evaluar condiciones de habitabilidad. Debe ser positivo. Formato esperado: numérico continuo.
- **Explotaciones:** Número de explotaciones agrícolas registradas. Indica la actividad económica en áreas rurales. Debe ser positivo. Formato esperado: numérico discreto.

Primero, se importan aquellos módulos y librerías necesarios para depurar los datos:

```
import pandas as pd
import numpy as np
from scipy.stats import skew
import matplotlib.pyplot as plt
```

En Python, se importar el archivo que contiene los datos mediante el siguiente código:

```
df = pd.read_excel('DatosEleccionesEspaña.xlsx')
```

Se obtiene el formato de cada variable con

```
df.select_dtypes()
```

y al notificar que algunas variables no tienen el formato deseado, se modifican con los siguientes comandos:

```
df[[ 'totalEmpresas', 'Industria', 'Construccion', 'ComercTTEHosteleria',
    'Servicios', 'inmuebles', 'Pob2010']] = df[[
    'totalEmpresas', 'Industria', 'Construccion', 'ComercTTEHosteleria',
    'Servicios', 'inmuebles', 'Pob2010']].astype(pd.Int64Dtype())
```

Aquellas variables no deseadas que se han mencionado anteriormente se eliminan de nuestro dataset tal que

```
df = df.drop(columns = ['AbstentionPtge', 'Izda_Pct', 'Otros_Pct', 'AbstencionAlta',
                                          'Izquierda'])
```

Ahora, si bien la siguiente línea no es necesaria, permite asegurarse de que no existen datos duplicados (dos filas de datos correspondientes a la misma localidad, por ejemplo):

```
df = df.drop_duplicates()
```

Ahora, con el fin de visualizar los principales parámetros estadísticos que describen la distribución de los datos, se calculan los siguientes parámetros con el siguiente código:

```
df_numeric_description = pd.DataFrame(
    columns = [
        'Variable', 'Valores diferentes', 'Media', 'Desviación Estándar',
        'Moda', 'Mínimo', 'Máximo',
        'Coeficiente de Pearson', 'Coeficiente de Curtosis', 'Asimetría',
        'Valores nulos', 'Valores erróneos'
    ]
)
```

```
df_numeric_description['Variable'] = list(df_numeric.columns)
df_numeric_description['Valores diferentes'] = [
df_numeric[col].nunique() for col in list(df_numeric.columns)
]
df_numeric_description['Media'] = [np.mean(df_numeric[col]) for col in list(df_numeric.columns)]
df_numeric_description['Moda'] = [df_numeric[col].mode()[0] for col in list(df_numeric.columns)]
df_numeric_description['Máximo'] = [np.max(df_numeric[col]) for col in list(df_numeric.columns)]
df_numeric_description['Mínimo'] = [np.min(df_numeric[col]) for col in list(df_numeric.columns)]
df_numeric_description['Desviación Estándar'] = [
np.std(df_numeric[col]) for col in list(df_numeric.columns)
]
df_numeric_description['Coeficiente de Pearson'] =
df_numeric_description['Desviación Estándar']
/ df_numeric_description['Media']
df_numeric_description['Asimetría'] = [
df_numeric[col].skew() for col in list(df_numeric.columns)
]
df_numeric_description['Coeficiente de Curtosis'] = [
df_numeric[col].kurt() for col in list(df_numeric.columns)
]
df_numeric_description['Valores erróneos'] = [
errors_list[column] if column in list(errors_list.keys()) else 0
for column in list(df_numeric.columns)
]
df_numeric_description['Valores nulos'] = [
np.sum(pd.isna(df_numeric[col])) for col in list(df_numeric.columns)
]
```

Aquí, df_numeric corresponde a aquellas columnas del DataFrame que sean numéricas, halladas con

```
df.select_dtypes(include=['number'])
```

y df_numeric_description será un DataFame que contenga los parámetros estadísticos de cada variable más relevantes. Por otra parte, errors_list es una lista que contiene el número de datos erróneos por columna siguiendo las indicaciones anteriores en cuanto a la naturaleza de cada variable.
Se han depurado previamente los datos para mostrar los parámetros estadísticos anteriores (donde en la sección 3 se argumentan las técnicas adoptadas a la hora de validar y corregir las entradas numéricas. No se han modificado en este paso aún los valores atípicos). Se adjunta a continuación el resultado:

(Var.ID).Variable	**Valores diferentes**	**Media**	**Desviación Estándar**	**Moda**	**Mínimo**
1.Population	3595	5722.345	46201.330	111	5
2.TotalCensus	3308	4247.864	34421.319	82	5
3.Dcha_Pct	6680	48.912	19.945	50	0
4.Age_0-4_Ptge	3759	3.018	2.052	0	0
5.Age_under19_Ptge	5889	13.564	6.777	0	0
6.Age_19_65_pct	6212	57.365	6.802	60	23.459
7.Age_over65_pct	7076	29.069	11.763	37	7.039
8.WomanPopulationPtge	4523	47.302	4.362	50	11.765
9.ForeignersPtge	7129	7.650	7.329	0	0
10.SameComAutonPtge	6150	81.623	12.271	100	0
11.SameComAutonDiffProvPtge	4207	4.338	6.395	0	0
12.DifComAutonPtge	5573	10.727	8.847	0	0
13.UnemployLess25_Ptge	2340	7.320	9.408	0	0
14.Unemploy25_40_Ptge	2679	37.001	20.318	0	0
15.UnemployMore40_Ptge	3172	55.678	22.086	100	0
16.AgricultureUnemploymentPtge	2523	8.403	12.959	0	0
17.IndustryUnemploymentPtge	2536	10.010	12.529	0	0
18.ConstructionUnemploymentPtge	2503	10.838	13.282	0	0
19.ServicesUnemploymentPtge	2903	58.647	24.260	0	0

20.totalEmpresas	1224	397.701	4219.233	0	0
21.Industria	307	23.405	158.618	0	0
22.Construccion	454	48.811	421.869	0	0
23.ComercTTEHosteleria	800	146.209	1232.637	0	0
24.Servicios	755	171.850	2446.890	0	0
25.inmuebles	3085	3240.038	24313.158	89	6
26.Pob2010	3622	5777.930	47524.953	101	5
27.SUPERFICIE	8108	6215.296	9218.036	2453	2.578
28.PobChange_pct	3047	-4.901	10.382	0	-52.270
29.PersonasInmueble	281	1.296	0.566	0.920	0.110
30.Explotaciones	757	122.229	225.652	15	1

Var.ID	**Máximo**	**Coeficiente de Pearson**	**Coeficiente de Curtosis**	**Asimetría**	**Valores nulos**	**Valores erróneos**
1	3141991	8.074	2820.327	46.041	0	0
2	2363829	8.103	2893.453	46.545	0	0
3	100	0.408	-0.176	-0.468	0	0
4	13.245	0.680	-0.207	0.344	0	0
5	33.696	0.500	-0.792	-0.105	0	0
6	100	0.119	2.012	-0.848	0	1
7	76.472	0.405	0.101	0.586	0	3
8	72.683	0.092	5.801	-1.671	0	0
9	71.369	0.958	10.665	2.485	0	653
10	100	0.150	3.480	-1.534	0	0
11	67.308	1.474	14.560	3.287	0	0
12	100	0.825	9.664	2.426	0	0
13	100	1.285	31.665	4.151	0	0
14	100	0.549	1.412	0.213	0	0
15	100	0.397	0.706	0.260	0	0
16	100	1.542	15.573	3.229	0	0
17	100	1.252	16.047	3.089	0	0
18	100	1.225	14.620	3.094	0	0
19	100	0.414	0.800	-0.806	0	0
20	299397	10.609	3475.480	53.714	5	0
21	10521	6.777	2643.855	44.271	188	0
22	30343	8.643	3506.444	52.577	139	0
23	80856	8.431	2652.635	45.460	9	0
24	177677	14.239	3833.616	57.503	62	0
25	1615548	7.504	2646.689	44.561	138	0
26	3273049	8.225	2944.830	47.201	7	0
27	175022.910	1.483	62.335	6.073	8	0
28	138.460	-2.118	15.112	1.506	7	0
29	3.330	0.437	-0.646	0.260	138	0
30	4759	1.846	65.272	6.151	189	189

Cuadro 2: Parámetros estadísticos relevantes de las variables.

Cabe mencionar en primer lugar la alta dispersión (medida por la desviación estándar) de algunas variables (produciendo un elevado valor del Coeficiente de Pearson), especialmente en las variables Population y TotalCensus, así como aquellas otras variables que dan cuenta del número de empresas según el sector (totalEmpresas, Industria, Construccion, etc.). Este hecho se debe a que se recogen datos de todas las localidades del territorio español, desde poblaciones de tan solo 5 habitantes hasta la propia capital con 3141991 habitantes. Como solución, en la sección 3.3 se realiza un tratamiento exhaustivo de los valores atípicos. Por otra parte, el Coeficiente de Curtosis, indicador de cuán plana o picuda es la distribución de datos, es extremadamente alto para algunas variables como Population o TotalCensus (2820.327 y 2893.453, respectivamente). Lo mismo sucede con totalEmpresas, Industria, Construccion, ComercTTEHosteleria y el resto que dan cuenta del número de empresas de cada sector (variando desde 3833.616 a 2643.855). Por otra parte, estas mismas variables poseen también valores positivos y altos del coeficiente de Asimetría, hecho que indican un fuerte sesgo a la derecha (2820.327 y 2893.453 para las variables de Population y TotalCensus, respectivamente). Si bien estos resultados indican que su distribución no responde a una distribución normal y carecen de significado estadístico, son un indicativo de

que es aconsejable realizar una transformación sobre ellas con el fin de reducir su dispersión (sección 3.3). Para otras variables, su comportamiento se ajusta más a una distribución normal.
Por otra parte, en la penúltima columna se especifican los valores perdidos. Se destacan especialmente las variables Industria y Explotaciones (con 188 y 189 valores perdidos, respectivamente, donde en el segundo caso se hallaron los valores nulos escritos como '99999'). Si bien los valores nulos son numerosos, no hacen peligrar la fiabilidad de la variable en conjunto (como máximo representan una pérdida del 2.3 % de los datos, una pérdida asumible estableciendo un máximo asumible del 10 %). Finalmente, en la última columna se muestran los errores en las variables, esto es, entradas que se hallan en disonancia con la tipología esperada (véase 3.1).

Todo el análisis anterior se ha realizado para aquellas variables numéricas del dataset. Se procede de forma análoga para las variables categóricas y binarias construyendo un DataFame que recoja sus principales caracaterísticas estadísticas:

```
df_cat_bin = df[[column for column in list(df.columns)
if column not in list(df_numeric.columns)]]
df_bin_cat_description = pd.DataFrame(
    columns = [
        'Variable', 'Valor más frecuente', 'Valores diferentes',
        'Valores nulos', 'Valores erróneos'
    ]
)

df_bin_cat_description['Valores diferentes'] = [
df_cat_bin[col].nunique() for col in list(df_cat_bin.columns)
]
df_bin_cat_description['Variable'] = list(df_cat_bin.columns)
df_bin_cat_description['Valor más frecuente'] = [
df_cat_bin[col].mode()[0] for col in list(df_cat_bin.columns)
]
df_bin_cat_description['Valores nulos'] = [
np.sum(pd.isna(df_cat_bin[col])) for col in list(df_cat_bin.columns)
]

# Valores NaN detectados como '?' en la variable Densidad.
df_bin_cat_description.loc[
df_bin_cat_description['Variable'] == 'Densidad', 'Valores nulos'
]
=
np.sum([1 for item in df_cat_bin['Densidad'] if item == '?'])

# Ningún error detectado
df_bin_cat_description['Valores erróneos'] = np.zeros(df_cat_bin.shape[1])
df_bin_cat_description['Valores erróneos'] =
 df_bin_cat_description['Valores erróneos'].astype('int')
```

El resultado es el siguiente:

Variable	**Valor más frecuente**	**Valores diferentes**	**Valores nulos**	**Valores erróneos**
Name	Arroyomolinos	8100	0	0
CCAA	CastillaLeón	17	0	0
Derecha	1	2	0	0
ActividadPpal	Otro	5	0	0
Densidad	MuyBaja	4	92	0

Cuadro 3: Tabla con datos de variables, frecuencias y valores.

Cabe destacar que a pesar de que la variable Name es un identificador único de cada localidad, la cantidad de valores diferentes no coincide con el número total de datos por columna (8117). Esto se debe a la presencia de localidades con el mismo nombre pero situadas en diferentes provincias. Por otra parte, se menciona que la

actividad sectorial más común entre las diferentes localidades es Otros, mientras que la Densidad poblacional más común es MuyBaja (gran parte de los regitros provienen de poblaciones de pocos habitantes). A la vez, el valor más repetido para la variable Derecha es 1, es decir, el número de localidades donde han ganado partidos políticos de derecha excede los de izquierda. Por último, se destaca la presencia de 92 valores nulos en la variable Densidad (corresponde a aquellas entradas marcadas como '?'en vez de np.nan).

3. Depuración de datos

A continuación se procede a filtrar y corregir los datos, desde el tratamiento de errores hasta valores perdidos para finalmente ser imputados por valores alternativos. A la vez, en la sección 3.3 se realizarán una serie de transformaciones sobre aquellas variables con altas dispersiones con el fin de normalizar su comportamiento.

3.1. Tratamiento de los errores detectados

Se han detectado los siguientes errores en la base de datos, indicando en qué columna se han hallado, su tipología, su número y como se ha gestionado.

1. **Age_19_65_pct**
   - **Número**: 1
   - **Tipología**: el valor porcentual debe de ser inferior a 100.
   - **Solución**: se sustituye dicho dato por np.nan para su posterior tratamiento en 3.2:
     ```
     df_numeric.loc[:, 'Age_19_65_pct'] = [
     np.nan if x > 100 else x for x in df_numeric.loc[:,'Age_19_65_pct']
     ]
     ```
2. **Age_over65_pct**
   - **Número**: 3
   - **Tipología**: el valor porcentual debe de ser superior a 0.
   - **Solución**: dado que la suma de todos los porcentajes debe dar 100, se computa el nuevo valor como la diferencia entre 100 y las demás 2 variables:

     ```
     df_numeric.loc[:, 'Age_over65_pct'] =
      [100 - df_numeric.loc[i, 'Age_19_65_pct'] - df_numeric.loc[i, 'Age_under19_Ptge']
     if x<0  else x for i, x in enumerate(df_numeric.loc[:, 'Age_over65_pct'])]
     ```

     En el paso anterior se había imputado como np.nan un dato que superaba 100, cuyo índice coincide con uno de los 3 de los datos incorrectos aquí, por lo que se devuelve un valor nulo también. Esto se corrige en 3.2.

3. **ForeignersPtge**
   - **Número**: 653 [1]
   - **Tipología**: el valor porcentual debe de ser superior a 0.
   - **Solución**: dado que la suma de todos los porcentajes debe dar 100 y además el valor debe ser positivo, se computa el nuevo valor como la diferencia entre 100 y las demás 2 variable, siempre que aquel porcentaje que pueda dar problemas sea inferior a 100:

     ```
     df_numeric.loc[:, 'ForeignersPtge'] =
      [100 - df_numeric.loc[i, 'SameComAutonPtge'] - df_numeric.loc[i, 'DifComAutonPtge']
     if df_numeric.loc[i, 'SameComAutonPtge'] <= 100 else x
     for i, x in enumerate(df_numeric.loc[:, 'ForeignersPtge'])]
     ```

     Aquí cabe discutir la fiabilidad de la variable ForeignersPtge: se impone que la suma de las 3 variables porcentuales de el 100 % del voto. Como se lee en el pie de página, 7570 entradas no cumplen dicha condición, por lo que se estima que aquella variable que debe ajustarse a dicha condición sea aquella con más errores (aquella menos fiable), escogiendo ForeignersPtge para tal propósito. Como se verá a continuación, la variable SameComAutonPtge posee a la vez errores, pero son solo 3 y no corresponde a aquellas entradas donde ForeignersPtge toma un valor negativo.

[1] Se han detectado 653 valores negativos, pero si la suma de las 3 variables porcentuales respectivas debe dar 100, entones se identifican sin previa modificación 7570 valores incorrectos, si se considera esta variable como la incorrecta y las demás como las certeras.

4. **SameComAutonPtge**
   - **Número**: 3
   - **Tipología**: el valor porcentual debe de ser inferior a 100.
   - **Solución**: se sustituye por el correspondiente para que la suma de las 3 variables porcentuales de el 100 %:

```
df_numeric.loc[:, 'SameComAutonPtge'] =
 [100 - df_numeric.loc[i, 'ForeignersPtge'] - df_numeric.loc[i, 'DifComAutonPtge']
if x > 100 else x for i, x in enumerate(df_numeric.loc[:, 'SameComAutonPtge'])]
```

5. **Densidad**
   - **Número**: 92
   - **Tipología**: los valores nulos se han insertado como '?'.
   - **Solución**: se sustituyen por np.nan para su posterior tratamiento en la sección 3.2:

```
df_cat_bin.loc[:, 'Densidad'] = df_cat_bin.loc[:, 'Densidad'].replace('?', np.nan)
```

Como añadido, se observa la baja representación de las clases Construcción e Industria en la variable categórica ActividadPpal (0.1725 % y 0.1602 %, respectivamente). Así, se juntan las dos clases en una misma categoría con el siguiente código:

```
df_cat_bin['ActividadPpal'] = [
'Construccion/Industria' if x in ['Industria', 'Construccion']
else x for x in df_cat_bin['ActividadPpal']
]
```

3.2. Tratamiento de los valores perdidos

A continuación, se procede a imputar aquellos valores nulos por valores alternativos siguiendo un conjunto de estrategias:

- En la variable Densidad se escoge imputar aquellos valores nulos por su correspondiente moda, es decir, 'Muy bajo'. Se realiza como sigue:

```
df_cat_bin.loc[:, 'Densidad'].fillna('Muy baja', inplace = True)
```

  Se ha optado por esta estrategia al desear preservar el máximo número de datos, considerando que aquellas localidades cuya Densidad es desconocida será con mayor probabilidad 'Muy baja'.

- En la variable Age_19_65_pct se tenía un valor nulo impuesto por nosotros mismos al superar el valor máximo de 100. Se opta por sustituirlo por la media, al observar que su distribución (a partir de los coeficientes de Curtosis y Asimetría) no se aleja demasiado de una distribución Gaussiana. Se procede como sigue:

```
df_numeric.loc[:, 'Age_19_65_pct'].fillna(
df_numeric.loc[:, 'Age_19_65_pct'].mean(), inplace = True
)
```

- Una vez corregido el único valor nulo de Age_19_65_pct, ya es posible modificar también el valor nulo de Age_over65_pct teniendo en cuenta la condición de que la suma de los porcentajes debe dar 100:

```
nan_index = df_numeric[df_numeric.loc[:, 'Age_over65_pct'].isna()].index
df_numeric.loc[:, 'Age_over65_pct'].fillna(
100 - df_numeric.loc[nan_index, 'Age_19_65_pct'] - df_numeric.loc[nan_index,

 inplace = True
)
```

Ahora, se plantea la imputación de los valores nulos en aquellas columnas que no poseen ningún error en su escritura. Ante esta problemática, se propone imputar los valores nulos por los dados por los cuartiles dado su coeficiente de Asimetría, tal que si el coeficiente de Asimetría es positivo (sesgo hacia la derecha, donde la media es superior a la mediana) se imputa el valor perdido por uno correspondiente a un cuartil bajo para acertar allí donde se concentra la mayoría de datos, y a la inversa para valores de la Asimetría negativos. Se implementa en Python mediante una función como sigue:

```
def fillna_skew(skewness, data):
    """
    Selecciona un cuartil por el que sustituir aquellos datos perdidos.

    Args:
        skewnesss (float): asimetría de la muestra de datos.
        data (pd.Series): conjunto de datos con valores perdidos.

    Returns:
        data: conjunto de datos corregidos.

    """
    if skewness > 1:
        quantile_value = 0.10
    elif 0 < skewness <= 1:
        quantile_value = 0.25
    elif -1 <= skewness < 0:
        quantile_value = 0.75
    elif skewness < -1:
        quantile_value = 0.90
    else:
        quantile_value = 0.50

    data = data.fillna(data.quantile(quantile_value))
    return data
```

Se aplica al conjunto de valores nulos restantes de la siguiente manera:

```
for i, column in enumerate(list(df_numeric.columns)):
    df_numeric[column] =
 fillna_skew(df_numeric_description.loc[i, 'Asimetría'], df_numeric.loc[:, column])
```

3.3. Tratamiento de los valores atípicos

Existen motivos suficientes para considerar transformar algunas variables dada su dispersión y presencia de abundantes valores atípicos. Para ello, se utiliza la función logaritmo, que reduce notablemente la dispersión de los datos. Las variables seleccionadas son Population, TotalCensus, TotalEmpresas, Industria, Construccion, ComercioTHHosteleria, Servicios, inmuebles, Pob2010, SUPERFICIE y Explotaciones. La aplicación se realiza mediante el siguiente código:

```
df_numeric['log(Population)'] = np.log1p(df_numeric['Population'])
df_numeric['log(TotalCensus)'] = np.log1p(df_numeric['TotalCensus'])
df_numeric['log(totalEmpresas)'] = np.log1p(df_numeric['totalEmpresas'])
df_numeric['log(Industria)'] = np.log1p(df_numeric['Industria'])
df_numeric['log(Construccion)'] = np.log1p(df_numeric['Construccion'])
df_numeric['log(ComercTTEHosteleria)'] = np.log1p(df_numeric['ComercTTEHosteleria'])
df_numeric['log(Servicios)'] = np.log1p(df_numeric['Servicios'])
df_numeric['log(inmuebles)'] = np.log1p(df_numeric['inmuebles'])
df_numeric['log(Pob2010)'] = np.log1p(df_numeric['Pob2010'])
df_numeric['log(SUPERFICIE)'] = np.log1p(df_numeric['SUPERFICIE'])
df_numeric['log(Explotaciones)'] = np.log1p(df_numeric['Explotaciones'])
```

Finalmente se eliminan las variables anteriores sin transformar:

```
df_numeric = df_numeric.drop(
columns = ['Population', 'TotalCensus', 'totalEmpresas', 'Industria', 'Construccion',
       'ComercTTEHosteleria', 'Servicios', 'inmuebles', 'Pob2010', 'SUPERFICIE',
       'Explotaciones']
)
```

Se hace uso de la fución np.log1p(...) ya que np.log(...) es problemática para argumentos igual a 0.
Por otra parte, no se han identificado errores atípicos como outliers imputados por error o de otras tipologías.

A continuación, dado que se tenían variables categóricas, es necesario convertirlas a binarias. Se pretende binarizar las variables CCAA, AtividadPpal y Densidad. Incluyendo la variable CCAA codificada se permite introducir las diferencias autonómicas en los modelos posteriores. Se implementa con el siguiente código:

```
df_cat_bin = pd.get_dummies(df_cat_bin, columns = ['CCAA', 'ActividadPpal','Densidad'])
df_cat_bin.replace({False: 0, True: 1}, inplace=True)
```

Se eliminan aquellas columnas de las que no se hará uso (un identificador único no aportará información alguna):

```
df_cat_bin = df_cat_bin.drop(columns = ['Name', 'CodigoProvincia'])
```

Y se guardan los datos depurados en un nuevo archivo Excel tras concatenar los DataFames de las variables numéricas con el de las variables categóricas y binarias:

```
df_cleaned = pd.concat([df_cat_bin, df_numeric], axis = 1)
df_cleaned.to_excel('DatosEleccionesEspaña_cleaned.xlsx')
```

4. Modelización

Primero, se importan las librerías necesarias en esta sección:

```
import time
import pandas as pd
import numpy as np
import seaborn as sns
import statsmodels.api as sm
import scipy.stats as stats
from sklearn.model_selection import train_test_split
from collections import Counter
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict, StratifiedKFold
from sklearn.metrics import log_loss
```

Con el fin de introducir las interacciones al modelo, se analiza mediante una matriz de correlación como se correlacionan las diferentes variables numéricas entre ellas:

```
# Se importan los datos anteriormente guardados.
df = pd.read_excel('DatosEleccionesEspaña_cleaned.xlsx')
corr_matrix = df.select_dtypes(include = ['number', 'float']).corr(method = 'pearson')
mask = np.triu(np.ones_like(corr_matrix, dtype = bool))
```

El resultado graficado (haciendo uso de sns.heatmap, utilizando en el argumento mask la máscara creada arriba) es el siguiente:

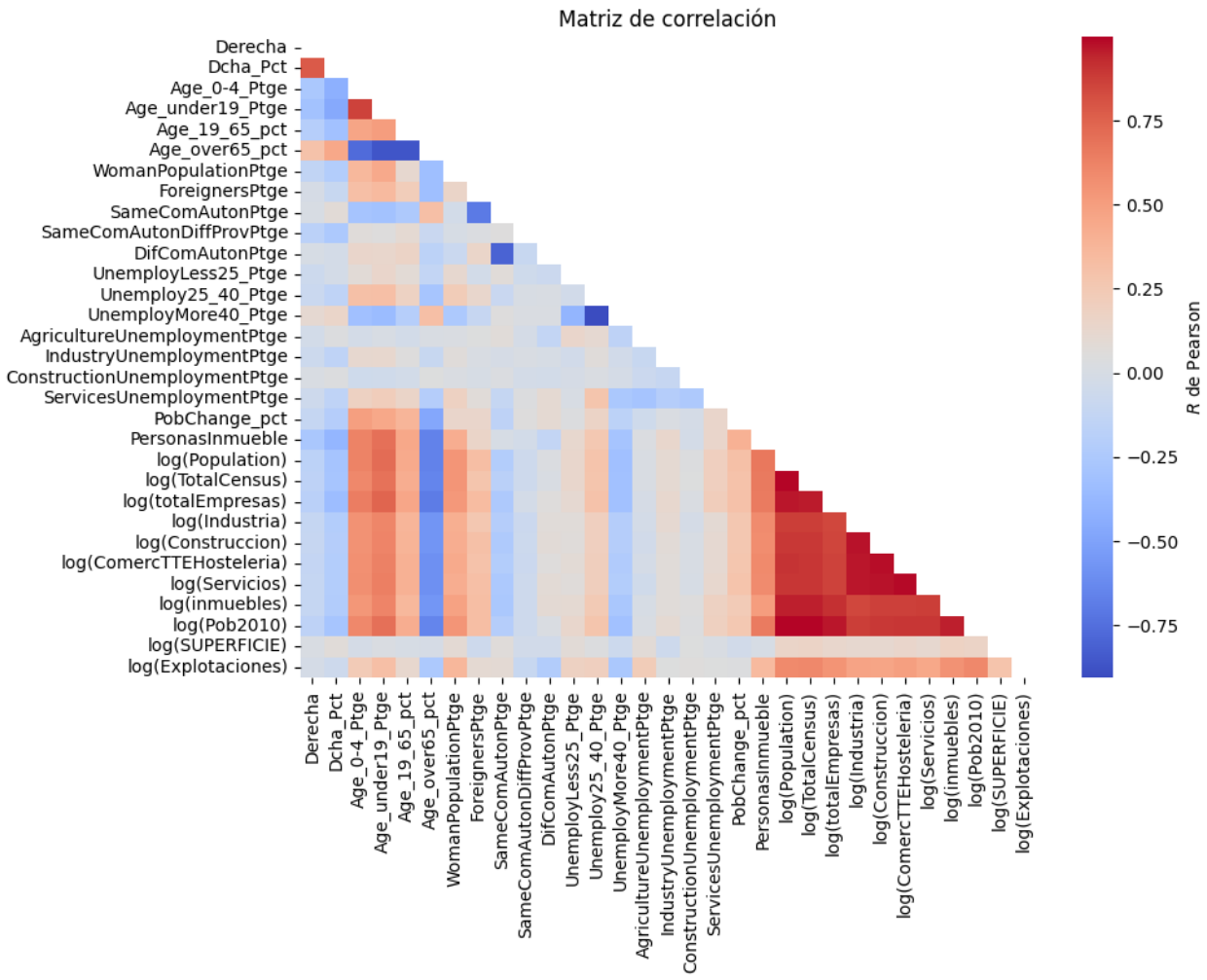

Figura 1: Matriz de correlación entre las variables explicativas y objetivos. Se elimina el triángulo superior para evitar redundancia.

Se identifican fuertes correlaciones tanto positivas como negativas entre diferentes variables: se halla una notable correlación positiva entre la variable Derecha y Dcha_Pct (0.782), así como entre aquellas variables que dan cuenta del porcentaje de voto por edad del votante (todas deben sumar 100, de ahí una fuerte correlación positiva o negativa). De la misma manera, se identifican correlaciones positivas entre log(Population), log(TotalCensus) y las variables que explican el número de empresas de cada sector. Efectivamente, se espera que halla un mayor número de empresas allí donde la población es mayor. Si bien cada correlación relevante encapsula un fenómeno social (por ejemplo, que exista una correlación de 0.739 entre el logaritmo del número total de empresas y el porcentaje de votantes menores a 19 años indica mayor actividad empresarial allí donde haya una mayor población joven), se limita el análisis a las interacciones que muestren un coeficiente de correlación en valor absoluto superior o igual a 0.5. También, hay que tener en cuenta los fenómenos de colinearidad, que conducen a la inestabilidad del modelo dados valores de correlación extremadamente altos entre dos variables (igual o superiores a 0.9 en valor absoluto). Así, de las dos variables que cumplen dicho requisito se decide eliminar la que presente una menor relevancia para el modelo (menor correlación con el resto de variables). Las variables eliminadas son: UnemployMore40_Ptge, log(TotalCensus) y log(Pob2010). Se han conervado las variables log(Industria), log(Construccion), log(ComercTTEHosteleria) y log(Servicios) ya que a pesar de que pueden presentar efectos de colinearidad, permiten describir los modelos principales de generación de riqueza de cada población. Finalmente, se ejecuta el siguiente código:

```
corr_matrix_float = df.select_dtypes(include = ['float']).corr()
df_inter = df.copy()
remove_features = []
columns_to_add = {}

for feature_1 in list(df_inter.select_dtypes(include = ['float']).columns):
```

```
    remove_features.append(feature_1)
    features_selected = [f for f in list(df_interr.columns) if f not in remove_features]

    for feature_2 in features_selected:
        if np.abs(corr_matrix_float.loc[feature_1, feature_2]) >= 0.5:
            columns_to_add[f'{feature_1}:{feature_2}'] = df_inter[feature_1] * df_inter[feature_2]

df_to_add = pd.DataFrame(columns_to_add)
df_inter = pd.concat([df_inter, df_to_add], axis = 1)
```

El anterior código permite calcular las interacciones Variable1:Variable2 omitiendo escribir la interacción Variable2:Variable1 al considerarla la misma. También se omiten así las interacciones de la variable consigo misma.

4.1. Regresión lineal

Se toma como variable objetivo la variable Dcha_Pct. Se separan los datos entre las variables explicativas y objetivo:

```
target_var = df_inter['Dcha_Pct']
input_var = df_inter.drop(columns = ['Dcha_Pct', 'Derecha'])
```

A continuación, se dividen los datos entre una parte dedicada al entrenamiento del modelo (80 %), y otra orientada al testeo (20 %):

```
x_train, x_test, y_train, y_test =
 train_test_split(input_var, target_var, test_size = 0.2, random_state = 12345)
```

Se plantean a continuación las siguientes estrategias de selección de variables.

4.1.1. Método *backward*

Consiste en partir de un modelo de regresión lineal teniendo en cuenta todas las variables implicadas menos una, calcular un parámetro estadístico que de cuenta de la precisión del ajuste y comparar el resultado con lo obtenido si se tuviesen en cuenta todas las variables. Si la precisión obtenida es superior, se elimina definitivamente la variable del modelo. Sino, se prueba con otra variable, reinstaurando la variable anterior y eliminando otra nueva temporalmente hasta su comprobación. Este algoritmo se itera sucesivamente hasta que ya no se pueda aumentar más la precisión del modelo. Los parámetros que indicarán cuando eliminar o mantener la variable pueden ser dos:

- BIC (Criterio de información Bayesiano): expresado como $BIC = kln(n) - 2ln(L)$ donde k es el número de parámetros del modelo, n el número de observaciones y L su verosimilitud (medida de qué tan bien los datos observados son explicados por el modelo), busca encontrar el modelo más probable dado los datos (máxima evidencia a posteriori). Prefiere modelos más simples cuando los datos son limitados.

- AIC (Criterio de información de Akaike): expresado como $AIC = 2k - 2ln(L)$, busca encontrar el modelo que minimice la pérdida de información. Prioriza un buen ajuste del modelo, incluso si esto significa incluir más parámetros.

Ambas medidas no son absolutas, y se utilizan para comparar el rendimiento de los diferentes modelos. Con esto, se implementa el método *backward* con la siguiente función:

```
def backward_feature_selection(X_train, y_train, method = 'AIC'):
"""
    Realiza la selección de características hacia atrás (backward feature selection)
    para un modelo de regresión lineal utilizando los criterios AIC o BIC.

    Args:
```

```
        X_train (pd.DataFrame): Conjunto de características de entrenamiento.
        y_train (pd.Series): Variable objetivo para el conjunto de entrenamiento.
        X_test (pd.DataFrame): Conjunto de características de prueba (no utilizado
directamente en el algoritmo).
        y_test (pd.Series): Variable objetivo para el conjunto de prueba (no utilizada
directamente en el algoritmo).
        method (str): Métrica para evaluar la calidad del modelo. Opciones: 'AIC' o 'BIC'.
                      Por defecto, utiliza 'AIC'.

    Returns:
        tuple:
            - list: Lista de las mejores características seleccionadas después del proceso.
            - float: Valor del mejor puntaje (AIC o BIC) del modelo seleccionado.

    Raises:
        Exception: Si el argumento `method` no es válido (debe ser 'AIC' o 'BIC').
    """

    remaining_features = list(X_train.columns)
    best_features = remaining_features.copy()
    best_score = float('inf')

    while len(remaining_features) > 0:
        scores = []
        feature_to_remove = None
        for feature in remaining_features:

            temp_features = [f for f in remaining_features if f != feature]
            model = sm.OLS( y_train, X_train[temp_features]).fit()

            if method == 'AIC':
                score = model.aic

            elif method == 'BIC':
                score = model.bic

            else:
                raise Exception('Métrica no válida.')

            scores.append((score, feature))

        scores.sort(reverse=False)
        current_score, feature_to_remove = scores[0]
        if current_score < best_score:
            print(f'Variable eliminada: {feature_to_remove}, {method}: {current_score}')
            best_score = current_score
            best_features.remove(feature_to_remove)
            remaining_features.remove(feature_to_remove)

        else:
            break

    return(best_features, best_score)
```

En la sección 5 en una misma tabla se adjuntarán todos los resultados obtenidos. Dado que se parte del máximo número de parámetros desde un principio, se espera que este método sea más lento que los siguientes.

4.1.2. Método *forward*

El procedimiento es similar al del método anterior pero a la inversa: se añaden sucesivamente variables al modelo si estas permiten obtener una mejor precisión. El proceso termina una vez dicha precisión ya no es mejorable. Los coeficientes usados para la medición de la precisión serán de nuevo el BIC y el AIC. La siguiente función implementa dicho método:

```
def forward_feature_selection(X_train, y_train, method = 'AIC'):
"""
    Realiza la selección de características hacia adelante (forward feature selection).

    Args:
        X_train (pd.DataFrame): Conjunto de características de entrenamiento.
        y_train (pd.Series): Variable objetivo para el conjunto de entrenamiento.
        X_test (pd.DataFrame): Conjunto de características de prueba (no utilizado
directamente en el algoritmo).
        y_test (pd.Series): Variable objetivo para el conjunto de prueba (no utilizada
directamente en el algoritmo).
        method (str): Métrica para evaluar la calidad del modelo. Opciones: 'AIC' o 'BIC'.
                      Por defecto, utiliza 'AIC'.

    Returns:
        tuple:
            - list: Lista de las mejores características seleccionadas después del proceso.
            - float: Valor del mejor puntaje (AIC o BIC) del modelo seleccionado.

    Raises:
        Exception: Si el argumento ‘method‘ no es válido (debe ser 'AIC' o 'BIC').

    """
    remaining_features = list(X_train.columns)
    best_features = []
    best_score = float('inf')

    while len(remaining_features) > 0:
        scores = []
        feature_to_add = None

        for feature in remaining_features:
            # Create a temporary dataset without the current feature

            temp_features = best_features + [feature]
            model = sm.OLS( y_train, X_train[temp_features]).fit()

            if method == 'AIC':
                score = model.aic

            elif method == 'BIC':
                score = model.bic

            else:
                raise Exception('Métrica no válida.')

            scores.append((score, feature))

        scores.sort(reverse=False)
        current_score, feature_to_add = scores[0]

        if current_score < best_score:
            print(f'Variable añadida {feature_to_add}, {method}: {current_score}')
            best_score = current_score
```

```
            best_features.append(feature_to_add)
            remaining_features.remove(feature_to_add)

        else:
            break

    return(best_features, best_score)
```

Dado que el modelo parte de 1 parámetro y se añaden sucesivamente, se espera que sea el modelo con menor carga computacional y más rápido.

4.1.3. Método *stepwise*

En el modelo *stepwise* se juntan las mejores características de los 2 modelos anteriores: la rapidez del modelo *forward* y la precisión del *backward*. Aquí, el modelo posee muchas similitudes con el modelo *forward* salvo que tras la adición de una nueva variable se permite eliminar aquella variable de las guardadas que pueda mejorar la precisión, esto es, un paso *backward* integrado tras cada paso *forward*. Se implementa con la siguiente función:

```
def stepwise_feature_selection(X_train, y_train, method = 'AIC'):
   """
    Realiza la selección de características paso a paso (stepwise feature selection).

    Args:
        X_train (pd.DataFrame): Conjunto de características de entrenamiento.
        y_train (pd.Series): Variable objetivo para el conjunto de entrenamiento.
        X_test (pd.DataFrame): Conjunto de características de prueba (no utilizado
directamente en el algoritmo).
        y_test (pd.Series): Variable objetivo para el conjunto de prueba (no utilizada
directamente en el algoritmo).
        method (str): Métrica para evaluar la calidad del modelo. Opciones: 'AIC' o 'BIC'.
                      Por defecto, utiliza 'AIC'.

    Returns:
        tuple:
            - list: Lista de las mejores características seleccionadas después del proceso.
            - float: Valor del mejor puntaje (AIC o BIC) del modelo seleccionado.

    Raises:
        Exception: Si el argumento ‘method‘ no es válido (debe ser 'AIC' o 'BIC').

    """
    selected_features = []
    remaining_features = list(X_train.columns)
    best_score = float('inf')
    improved = True

    while improved:
        improved = False

        # Paso Forward.
        forward_scores = []
        for feature in remaining_features:
            temp_features = selected_features + [feature]
            model = sm.OLS(y_train, X_train[temp_features]).fit()

            if method == 'BIC':
                score = model.bic

            elif method == 'AIC':
```

```
                score = model.aic

            else:
                raise Exception('Métrica no válida.')

            forward_scores.append((score, feature))

        if forward_scores:
            forward_scores.sort(reverse=False)
            best_forward_score, best_forward_feature = forward_scores[0]
            if best_forward_score < best_score:
                best_score = best_forward_score
                selected_features.append(best_forward_feature)
                remaining_features.remove(best_forward_feature)
                improved = True
                print(f"Añadiendo {best_forward_feature}, Nuevo {method}: {best_forward_score}")

        if len(selected_features) > 2: # Si hay más de 2 variables añadidas.
            # Paso Backward.
            backward_scores = []
            for feature in selected_features:
                temp_features = [f for f in selected_features if f != feature]
                model = sm.OLS(y_train, X_train[temp_features]).fit()

                if method == 'BIC':
                    score = model.bic

                elif method == 'AIC':
                    score = model.aic

                else:
                    raise Exception('Métrica no válida.')

                backward_scores.append((score, feature))

            if backward_scores:
                backward_scores.sort(reverse=False)
                best_backward_score, worst_feature = backward_scores[0]
                if best_backward_score < best_score:
                    best_score = best_backward_score
                    selected_features.remove(worst_feature)
                    remaining_features.append(worst_feature)
                    improved = True
                    print(f"Eliminando {worst_feature}, Nuevo {method}: {best_backward_score}")

    return(selected_features, best_score)
```

4.1.4. Método de selección de variables por frecuencia

En el método de selección de variables por frecuencia se crearán subdivisiones de los datos de entrenamiento (n iteraciones) para obtener el conjunto de parámetros que optimiza la precisión usando uno de los 3 modelos anteriores. Se almacenan los resultados para a continuación devolver aquel conjunto de parámetros que se repite con mayor frecuencia. De esta menera, este método se nutre de los demás modelos y solo implementa la ejecución de uno de esos modelos escogido sobre nuevas divisiones de entrenamiento. Se implementa con la siguiente función:

```
def random_features_selector(x_train, y_train, n_iter, method = 'stepwise', metric = 'AIC'):
    """
    Realiza una selección de características repetitiva para un modelo de regresión.
```

```
    Args:
        x_train (pd.DataFrame): Conjunto de características de entrenamiento.
        y_train (pd.Series): Variable objetivo para el conjunto de entrenamiento.
        n_iter (int): Número de iteraciones para generar divisiones del
        conjunto de datos.
        method (str): Método de selección de características a utilizar. Opciones:
                      'forward', 'backward', 'stepwise'. Por defecto: 'stepwise'.
        metric (str): Métrica utilizada para evaluar la calidad del modelo en cada iteración.
                      Opciones: 'AIC' o 'BIC'. Por defecto: 'AIC'.

    Returns:
        tuple:
            - list: Lista de las características más seleccionadas en las iteraciones.
            - int: Número de veces que la combinación más común de características
              fue seleccionada.

    Raises:
        Exception: Si el nombre del método especificado no es válido.

    """
    variables = []
    for x in range(n_iter):
        x_train2, x_test2, y_train2, y_test2 =
 train_test_split(
x_train, y_train, test_size = 0.2,
 random_state = np.random.randint(0,10000)
)
        if method == 'forward':
            model = forward_feature_selection(
x_train2, y_train2, x_test2, y_test2, method = metric
)
        elif method == 'backward':
            model = backward_feature_selection(
x_train2, y_train2, x_test2, y_test2, method = metric
)
        elif method == 'stepwise':
            model = stepwise_feature_selection(
x_train2, y_train2, x_test2, y_test2, method = metric
)
        else:
            raise Exception('Método no reconocido.')
        variables.append(tuple(model[0]))
    counter = Counter(variables)
    most_common_formula, count = counter.most_common(1)[0]

    return(most_common_formula, count)
```

4.2. Regresión logística

Se toma como variable objetivo la variable binaria Derecha. Se separan los datos entre las variables explicativas y objetivo:

```
target_var_cat = df_inter['Derecha']
input_var_cat = df_inter.drop(columns = ['Derecha', 'Dcha_Pct'])
```

A continuación, se separan los datos entre aquella división dedicada al entrenamiento (80 %), y otra orientada al testeo (20 %):

```
x_train_cat, x_test_cat, y_train_cat, y_test_cat =
 train_test_split(input_var_cat, target_var_cat, test_size = 0.2, random_state = 12345)
```

En cuanto al cálculo de los parámetros BIC y AIC, para ajustes logísticos no existe una fución integrada en Python por lo que se define manualmente:

```
def calculate_bic(logit_result):
    """
    Calcula el Criterio de Información Bayesiano (BIC).

    Args:
        logit_result: statsmodels LogitResults
            El objeto del modelo Logit ajustado

    Returns:
        float
            El valor del BIC para el modelo ajustado.

    """
    n_obs = logit_result.nobs
    k_params = logit_result.df_model + 1
    log_likelihood = logit_result.llf

    bic = -2 * log_likelihood + k_params * np.log(n_obs)

    return bic

def calculate_aic(logit_result):
    """
    Calcula el Criterio de Información de Akaike (AIC).

    Args:
        logit_result: statsmodels LogitResults
            El objeto del modelo Logit ajustado.

    Returns:
        float
            El valor del AIC para el modelo ajustado.
    """
    k_params = logit_result.df_model + 1  # df_model excludes the intercept
    log_likelihood = logit_result.llf

    aic = -2 * log_likelihood + 2 * k_params

    return aic
```

Dado que los códigos de la Regresión Lineal y Logística comparten muchas similitudes, solo se indicarán las principales diferencias entre ambas versiones para cada modelo.

4.2.1. Método *backward*

Solo se requiere intercambiar:

```
model = sm.OLS( y_train, X_train[temp_features]).fit()
```

por

```
model = sm.Logit( y_train, X_train[temp_features]).fit(method='bfgs')
```

y

```
score = model.aic -> calculate_aic(model)
score = model.bic -> calculate_bic(model)
```

4.2.2. Método *forward*

Misma modificación que en 4.2.1, aplicada al código del método *forward*.

4.2.3. Método *stepwise*

Misma modificación que en 4.2.1, aplicada al código del método *stepwise*.

4.2.4. Método de selección de variables por frecuencia

Ninguna modificación a realizar.

5. Resultados

5.1. Regresión lineal

Se adjuntan a continuación los resultados obtenidos para los 4 métodos de selección de variables y en función del parámetro de precisión escogido (BIC o AIC):

Método	**Métrica**	R^2 Train	R^2 Test	**Nº Parámetros**	**Tiempo de cómputo (segundos, train)**
Backward	AIC = 48867.97	0.7299	0.7656	70	1750.01
Backward	BIC = 49198.07	0.7292	0.7524	45	1530.47
Forward	AIC = 48883.48	0.7278	0.7507	58	303.17
Forward	BIC = 49163.11	0.7223	0.7462	30	109.06
Stepwise	AIC = 48868.19	0.7276	0.7504	46	381.39
Stepwise	BIC = 49126.07	0.7231	0.7464	28	117.23
Sel. Var. Frec. (Forward)	AIC = 48865.12	0.7216	0.7446	58	4372.22 (23 iter.)
Sel. Var. Frec. (Stepwise)	BIC = 49129.23	0.7273	0.7452	26	3567.59 (50 iter.)
Base sin interacciones		0.7156	0.7373	51	-
Base con interacciones		0.7333	0.7538	119	-

Cuadro 4: Comparativa entre los métodos de selección de variables para la regresión lineal. Los tiempos de cómputo se han calculado con la muestra de entrenamiento. Quedan vacías las celdas de Tiempo de cómputo para los modelos Base sin interacciones y Base con interacciones ya que no han ejecutado ninguna estrategia de selección de variables.

Se ha escogido realizar el método de selección de variables por frecuencia con la estrategia *forward* (la que ofrece un menor tiempo de cómputo con AIC) para un total de 23 iteraciones para el caso AIC, y estrategia *stepwise* con 50 iteraciones para el caso BIC. Este número de iteraciones ha sido escogido con el fin de que el tiempo total de cómputo no vaya más allá de las 2 horas, si bien son insuficientes para realizar un diagnóstico certero del rendimiento ya que la frecuencia del conjunto de variables seleccionadas máxima es de tan solo 2 en ambos casos. Dadas las limitaciones computacionales, se recomienda de cara a futuros trabajos utilizar un mayor número de iteraciones.

De los resultados obtenidos, se destaca que todas las modificaciones realizadas sobre los datos han permitido dar un valor de R^2 relativamente alto que permite asegurar una apreciable linealidad en los datos. Los modelos proporcionados ofrecen todos mejores resultados que aquellos en los que no se han considerado interacciones (Base sin interacciones), y tan solo uno (*Bcakward* AIC) ofrece mejores resultados que el modelo con interacciones (Base con interacciones). Ahora bien, se observa que el modelo que ofrece un mejor balance entre el tiempo de cómputo requerido y precisión es el método *forward* AIC con un R^2 de 0.7507 para el testeo, seguido del modelo *stepwise* AIC con $R^2 = 0{,}7504$.El modelo *backward* AIC ofrece el mejor R^2 de testeo a costa del mayor tiempo computacional.En cuanto al resto de modelos, el más rápido es el *forward* BIC con 109.06 segundos, seguido del modelo *stepwise* BIC (117.23 segundos). Así, los métodos *stepwise* AIC/*forward* AIC son los que mejor juntan precisión con un tiempo de cómputo reducido. Dicho empate se resuelve mediante Validación Cruzada. Se escogen 20 validaciones:

```
promedio_r2 = np.mean(ross_val_score(
LinearRegression(),
 input_var, target_var,
 cv = 20, scoring = 'r2'
          )
```

)

Los resultados son los siguientes:

Método	**Métrica**	R^2 **promedio**
Backward	AIC	0.7282±0.0300
Backward	BIC	0.7255±0.0300
Forward	AIC	0.7269±0.0300
Forward	BIC	0.7233±0.0300
Stepwise	AIC	0.7276±0.0300
Stepwise	BIC	0.7244±0.0300
Sel. Var. (Forward)	AIC	0.7253±0.0300
Sel. Var. (Stepwise)	BIC	0.7238± 0.0300
Base sin interacciones		0.7150±0.0300
Base con interacciones		0.7245±0.0300

Cuadro 5: Validación cruzada para los modelos de regresión lineal.

En el siguiente gráfico se puede visualizar como varía cada R^2 en la validación cruzada para cada modelo:

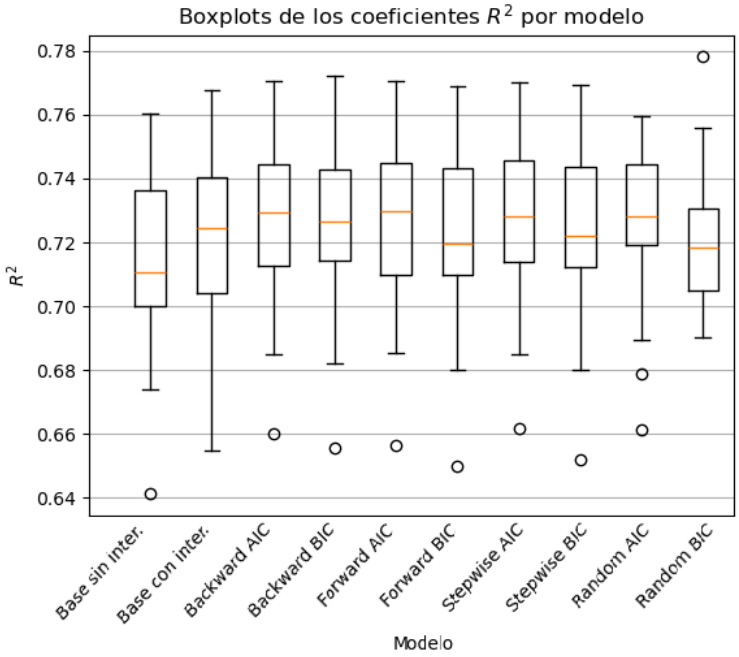

Figura 2: Boxplots de R^2 de los modelos lineales.

Con esto, se declara el modelo ***stepwise* AIC** como el que mejores resultados ofrece (valorando la precisión con el tiempo de cómputo. Si se tiene en cuenta el número de variables eliminadas, entonces el ganador sería el modelo *stepwise* BIC). Se muestran a continuación un resumen del modelo ganador con model.summary, siendo model el modelo ganador previamente entrenado son sm.OLS:

Variable dependiente	R^2	**Modelo**	R^2 **ajustado**	**Método**	**Estadístico F**
Dcha_Pct	0.7276	OLS	0.7276	Mínimos Cuadrados	4429
Prob (F)	**Log-Verosimilitud**	**No. Obs.**	**AIC**	**Df residuales**	**BIC**
0.00	-30434.0	8117	48860	8053	49163
Df modelo	**Tipo de covarianza**	**Omnibus**	**Durbin -Watson**	**Prob (Omnibus)**	**JB**
46	no robusta	289.704	1.887	0.000	755.015
Asimetría	**Prob (JB)**	**Curtosis**			
-0.145	0	4.460			

Cuadro 6: Resultados estadísticos del modelo ganador (*Stepwise* AIC).

Se analizan finalmente algunos parámetros de interés del ajuste realizado por el modelo ganador. Dichos parámetros se consultan con el comando model.summary(), siendo model el modelo entrenado:

	Coef.	**Std. Err.**	**z**	**P>|z|**	**[0.025**	**0.975]**
CCAA_PaísVasco	-40.4980	0.7665	-52.8364	0	-42.0005	-38.9955
CCAA_Cantabria	8.3881	1.0874	7.7141	0	6.2566	10.5196
PersonasInmueble	11.9087	1.8454	6.4532	0	8.2912	15.5261
ActividadPpal_Construccion/Industria	3.9528	2.0556	1.9229	0.0545	-0.0767	7.9823

Cuadro 7: Coeficientes del modelo ganador de la regresión lineal.

Para la variable continua ActividadPpal_Construccion/Industria el parámetro del ajuste toma el valor de 3.9528 y para la variable binaria CCAA_PaísVasco el parámetro toma el valor de -40.4980. De esta manera, uno puede hallar que aquellas comunidades autónomas cuya actividad principal sea Industria/Construcción apoya el Porcentaje de Voto de Derechas al ser el parámetro positivio. No sucede lo mismo para la variable binaria destacada, que al ser negativa penaliza la variable objetivo. Este fenómeno indica que difícilmente la ideología política imperante en el País Vasco es de izquierdas u otra que no sea derecha.

5.2. Regresión logística

Se adjuntan a continuación los resultados obtenidos para los 4 métodos de selección de variables en función del parámetro de precisión escogido (BIC o AIC). Si bien sm.Logit() no dispone de ninguna herramienta con la que obtener el valor del pseudo-R^2, se obtiene indirectamente a partir de la conversión del *summary* a un DataFrame con el código float(model.summary2().tables[0].iloc[0,3]). En cuanto a los métodos de selección de variables por frecuencia, se ha escogido utilizar el método *stepwise* ya que es el que ofrece el menor tiempo de cómputo. En la siguiente página se adjunta los resultados obtenidos.

Método	**Métrica**	$Pseudo-R^2$ Train	$Pseudo-R^2$ Test	**Nº Parámetros**	**Tiempo de cómputo (segundos, train)**
Backward	AIC = 4935.12	0.438	0.456	69	3697.67
Backward	BIC = 4939.20	0.450	0.474	23	2842.69
Forward	AIC = 4768.56	0.453	0.476	30	594.22
Forward	BIC = 4947.97	0.447	0.473	22	336.60
Stepwise	AIC = 4832.77	0.440	0.463	15	288.24
Stepwise	BIC = 4942.91	0.438	0.461	14	270.86
Sel. Var. (Stepwise)	AIC = 4858.40	0.439	0.451	14	6406.10 (25 iter.)
Sel. Var. (Stepwise)	BIC = 4939.24	0.435	0.458	15	4839.17 (25 iter.)
Base sin interacciones		0.430	0.458	51	-
Base con interacciones		0.352	0.390	119	-

Cuadro 8: Comparación de métodos de selección de características de la regresión logística.

Se ha escogido realizar el método de selección de variables por frecuencia para un total de 25 iteraciones para los casos donde se usen los parámetros AIC y BIC. En ambos casos, la frecuencia de la fórmula más repetida es de 2. Tal y como se había adelantado antes, es necesario aumentar el número de iteraciones a costa de un mayor tiempo de cómputo para ofrecer un resultado más certero.
De los resultados obtenidos, se destaca que ya de por si el pseudo-R^2 ofrecido por el modelo sin añadir interacciones es alto para el testeo (0.458). Se observa como la mayoría de los modelos ofrecen valores de pseudo-R^2 superiores al ofrecido por el modelo sin interacciones tanto en el training como en el testeo. De estos, se destacan los modelos *forward* AIC y BIC (con valores del pseudo-R^2 de 0.476 y 0.473 para el testeo, respectivamente) y *backward* BIC (0.474 para el testeo). Por otra parte, el número de parámetros de cada modelo es ahora reducido, partiendo de 52 (base sin interacciones) y 119 (con interacciones) parámetros a tan solo 14 en algunos casos (*Stepwise* BIC y *Sel. Var.* (Stepwise) AIC), ofreciendo un pseudo-R^2 razonablemente alto. Este fenómeno es debido a la naturaleza de la variable objetivo (continua o binaria). Por ello, los modelos han conseguido con éxito realizar una selección de variables adecuada, reduciendo la complejidad del modelo. Como era de esperar, los modelos *stepwise* y *forward* son los que ofrecen un menor tiempo de cómputo, al revés que para el método *backward*.
Se procede a continuación a realizar un análisis por Validación Cruzada para elegir un ganador. Si bien no existe una función que permita realizar la validación cruzada dado un ajuste logístico, se implementa como sigue:

```
def crossval_log(X, y, n_cv):
   """
    Calcula el valor promedio del pseudo-R² a partir de la
    validación cruzada en un modelo de regresión logística.

   Args:
        X (pd.DataFrame): Conjunto de características (entradas)
         para ajustar el modelo de regresión logística.
        y (pd.Series): Variable objetivo (salida) para el modelo
         de regresión logística.
       n_cv (int): El número de valdaciones cruzadas a realizar.

    Returns:
        float: El valor promedio del pseudo-R² calculado a partir de la validación cruzada.
        list: Una lista de los valores pseudo-R² hallados.
    """
    logreg = LogisticRegression(solver = 'liblinear')
    cv = StratifiedKFold(n_splits=n_cv, shuffle=True, random_state=42)
    pseudo_r2_values = []

    for train_idx, val_idx in cv.split(X, y):
        X_train, X_val = X.iloc[train_idx], X.iloc[val_idx]
        y_train, y_val = y.iloc[train_idx], y.iloc[val_idx]

        logreg.fit(X_train, y_train)
        y_val_probs = logreg.predict_proba(X_val)[:, 1]

        log_likelihood_model = -log_loss(y_val, y_val_probs, normalize=False)

        X_val_intercept = sm.add_constant(np.ones_like(y_val))  # Null model features
        null_model = sm.Logit(y_val, X_val_intercept).fit(disp=0)
        log_likelihood_null = null_model.llf

        pseudo_r2 = 1 - (log_likelihood_model / log_likelihood_null)
        pseudo_r2_values.append(pseudo_r2)

    mean_pseudo_r2 = np.mean(pseudo_r2_values)
    return(mean_pseudo_r2, pseudo_r2_values)
```

Se obtuvieron los siguientes resultados:

Se representan las distribuciones de los diferentes valores del pseudo-R^2 obtenidos según el modelo:

Método	**Métrica**	R^2 **promedio**
Backward	AIC	0.452±0.030
Backward	BIC	0.453±0.030
Forward	AIC	0.454±0.030
Forward	BIC	0.448±0.030
Stepwise	AIC	0.442±0.030
Stepwise	BIC	0.439±0.030
Sel. Var. (Stepwise)	AIC	0.438±0.030
Sel. Var. (Stepwise)	BIC	0.436±0.030
Base sin interacciones		0.444±0.040
Base con interacciones		0.451±0.040

Cuadro 9: Validación cruzada para los modelos de regresión logística.

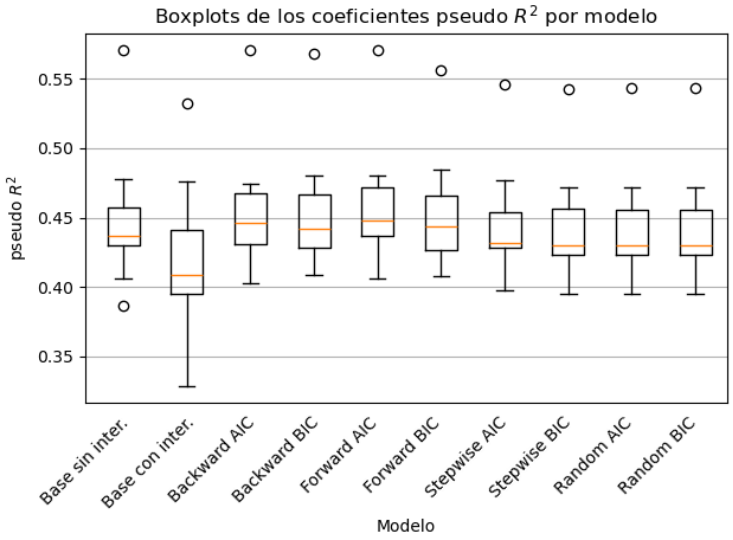

Figura 3: Boxplots del pseudo-R^2 de los modelos logísticos.

Por ello, se escoge como método ganador el modelo ***Forward*** **AIC** ya que ofrece la mejor relación entre el tiempo de cómputo, el número de variables eliminadas y la precisión ofrecida. Se evaluan a continuación parámetros estadísticios adicionales del modelo ganador con la siguiente función:

```
def params_df(model, x_data, y_data, cut_point):
    """
    Calcula parámetros estadísticos para evaluar el desempeño de un modelo de
    regresión logística.

    Args:
        model: El modelo de regresión logística que implementa el método ‘predict_proba‘.
        x_data (DataFrame): Datos de entrada utilizados para generar predicciones
       (array-like o DataFrame).
        y_data (DataFrame): Etiquetas reales de los datos de entrada (array-like o Series).
        cut_point (float): Umbral para convertir las probabilidades en predicciones binarias.

    Returns:
        pd.DataFrame: Un DataFrame que contiene las métricas calculadas:
            - 'Punto de corte': El valor del umbral utilizado para clasificar.
            - 'Precisión': Proporción de predicciones correctas sobre el total.
            - 'Sensitividad': Proporción de verdaderos positivos sobre la suma de
               positivos reales.
```

```
        - 'Especificidad': Proporción de verdaderos negativos sobre la suma de
          negativos reales.
        - 'Proporción positivos': Proporción de verdaderos positivos sobre todos los
          predichos como positivos.
        - 'Proporción negativos': Proporción de verdaderos negativos sobre todos los
          predichos como negativos.
    """

    probs = model.predict_proba(x_data)[:, 1]

    preds = (probs > cut_point).astype(int)

    cm = confusion_matrix(y_data, preds)
    tn, fp, fn, tp = cm.ravel()

    stats_df = pd.DataFrame({
        'Punto de corte': [cut_point],
        'Precisión': [(tp + tn) / (tn + fp + fn + tp)],
        'Sensitividad': [tp / (tp + fn)],
        'Especificidad': [tn / (tn + fp)],
        'Proporción true positivos': [tp / (tp + fp)],
        'Proporción true negativos': [tn / (tn + fn)]
    })

    return stats_df
```

Estos son los resultados:

Modelo	**Punto de corte**	**Precisión**	**Sensitividad**	**Especificidad**	**Proporción T positivos**	**Proporción T negativos**
Forward AIC	0.3	0.836207	0.964859	0.632166	0.806208	0.918981
Base sin inter.	0.3	0.778941	0.961847	0.488854	0.749023	0.889855

Cuadro 10: Evaluación estadística del modelo ganador para regresión logística.

Se ha tomado para este ejercicio un punto de corte de 0.3, tal y como se indica en el enunciado del mismo. Como se observa, el modelo ganador rinde por encima del modelo sin interacciones.
Se resume el modelo ganador en la siguiente tabla (datos obtenidos mediante model.summary2(), donde model es el modelo ganador entrenado con sm.Logit):

Modelo	**Método**	**Variable dependiente**	**Pseudo R-cuadrado**	**AIC**	**No. Observaciones**
Logit	MLE	Derecha	0.454	4767.12	8117
BIC	**Df Modelo**	**Log-Verosimilitud**	**Df Residuos**	**LL-Null**	**Valor p de LLR**
4938.91	30	-2890.6	8087	-5303	0
No. Iteraciones					
10					

Cuadro 11: Resultados del modelo ganador de la regresión logística.

Dado que ahora el número de parámetros del modelo es reducido, se adjuntan a continuación algunos parámetros de interés:

	Coef.	**Std. Err.**	**z**	**P>|z|**	**[0.025**	**0.975]**
CCAA_Cataluña	-5.852809	0.497286	-11.769501	0	-6.827472	-4.878146
CCAA_PaísVasco	-2.634700	0.337405	-7.808707	0	-3.296003	-1.973398
CCAA_Galicia	2.541634	0.307751	8.258729	0	1.938453	3.144816
Age_over65_pct	0.008529	0.005380	1.585217	0.11292	-0.002016	0.019074

Cuadro 12: Coeficientes del modelo ganador de la regresión logística.

Finalmente, uno puede extraer conclusiones del comportamiento del voto a partir de un análisis de los coeficientes de ajustes. En particular, para la variable binaria CCAA_Galicia el coeficiente toma el valor más alto de entre ellos, 2.541634. Este hecho indica que la Comunidad Autónoma de Galicia tiene más probabilidades de ser de Derecha que de Izquierda. En cambio, el coefiicente más bajo es -5.852809 para la variable CCAA_Cataluña, hecho que indica que las localidades situadas en Cataluña son más partidarias de partidos de izquierda u otros. También, se observa que la presencia de una población con edad superior a 65 años apenas influye en la decisión de voto.

6. Conclusiones

Tras el análisis exhaustivo de los resultados obtenidos, se llega a las siguientes conclusiones:

1. Los resultados electorales de los partidos de derecha pueden ser explicados mediante regresiones lineales y logísticas.
2. La aplicación de transformaciones sobre los datos originales permiten mejorar la linealidad del conjunto.
3. Existen fuertes correlaciones entre las diferentes variables tratadas, visualizado en la matriz de correlación de la sección 4.
4. La adición de parámetros de interacción en el modelo de regresión lineal ofrece mejores resultados que el modelo sin.
5. La implementación de la estrategia *stepwise* AIC para regresiones lineales es el método que ofrece mejores resultados en cuanto al balance entre precisión, simpleza y tiempo de cómputo.
6. La adición de parámetros de interacción en el modelo de regresión logística no ofrece mejoras en la precisión.
7. La implementación de la estrategia *forward* AIC para regresiones logísticas es el método que ofrece mejores resultados, ofreciendo una gran rapidez de cálculo y una precisión aceptable.
8. Los métodos *backward* son los que requieren de un mayor tiempo de cómputo, mientras que las estrategias *forward* y *stepwise* exhiben tiempos menores.
9. Es necesario el uso de mayores recursos computacionales para poder ejecutar el método de selección de variables por frecuencia para un mayor número de iteraciones, y así mejorar su validez.

7. Bibliografía

Las fuentes consultadas a lo largo de la redacción del presente documento y los códigos en Python se adjuntan a continuación:

- Espinola, Rosa. (2024). Apuntes de la Asignatura Minería de Datos y Modelización Predictiva, Universidad Complutense de Madrid.