RegresionLogistica

March 24, 2026

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler

import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

1 Ejercicio 1

1.1 Read Data

[2]:
```
from sklearn.datasets import fetch_openml
import pandas as pd

glass = fetch_openml(name="glass", version=1, as_frame=True)

df = glass.frame

X = df.drop(columns=["Type"])
y = df["Type"]
X.head()
```

[2]:

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe
0	1.51793	12.79	3.50	1.12	73.03	0.64	8.77	0.0	0.00
1	1.51643	12.16	3.52	1.35	72.89	0.57	8.53	0.0	0.00
2	1.51793	13.21	3.48	1.41	72.64	0.59	8.43	0.0	0.00
3	1.51299	14.40	1.74	1.54	74.55	0.00	7.59	0.0	0.00
4	1.53393	12.30	0.00	1.00	70.16	0.12	16.19	0.0	0.24

1.2 Objetivo

Esta base de datos (Glass Identification) tiene 214 observaciones y 9 variables predictoras físico-químicas:

- `RI`: índice de refracción del vidrio.

- `Na`: concentración de sodio.
- `Mg`: concentración de magnesio.
- `Al`: concentración de aluminio.
- `Si`: concentración de silicio.
- `K`: concentración de potasio.
- `Ca`: concentración de calcio.
- `Ba`: concentración de bario.
- `Fe`: concentración de hierro.

La variable de salida `Type` es categórica nominal con 6 clases:

1. `build wind float`
2. `build wind non-float`
3. `vehic wind float`
4. `containers`
5. `headlamps`
6. `tableware`

1.3 Limpieza

[3]:
```
X.info()
# no se encuentran valores nulos, por lo que no es necesario imputar nada.
```

```
<class 'pandas.DataFrame'>
RangeIndex: 214 entries, 0 to 213
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   RI      214 non-null    float64
 1   Na      214 non-null    float64
 2   Mg      214 non-null    float64
 3   Al      214 non-null    float64
 4   Si      214 non-null    float64
 5   K       214 non-null    float64
 6   Ca      214 non-null    float64
 7   Ba      214 non-null    float64
 8   Fe      214 non-null    float64
```

```
dtypes: float64(9)
memory usage: 15.2 KB
```

1.3.1 Selección de Variables

Multicolinealidad

[4]:
```
# VIF
X_vif = sm.add_constant(X, has_constant="add")

vif_data = pd.DataFrame({
    "feature": X_vif.columns,
    "VIF": [variance_inflation_factor(X_vif.values, i) for i in range(X_vif.
 ↪shape[1])]
})

print("Variance Inflation Factor (VIF):")
print(vif_data)
```

```
Variance Inflation Factor (VIF):
  feature           VIF
0   const  3.140751e+06
1      RI  9.504579e+00
2      Na  6.178964e+01
3      Mg  2.078413e+02
4      Al  2.568270e+01
5      Si  5.817673e+01
6       K  4.362078e+01
7      Ca  2.114275e+02
8      Ba  2.613530e+01
9      Fe  1.218039e+00
```

Tenemos variables extremadamente correlacionadas, dejaremos solamente aquellas con menor valor: Fe, RI, Al, reevaluaremos.

[5]:
```
X_small = X[["RI", "Fe", "Al", "Ba", "Na", "Si", "K"]]
X_vif = sm.add_constant(X_small, has_constant="add")

vif_data = pd.DataFrame({
    "feature": X_vif.columns,
    "VIF": [variance_inflation_factor(X_vif.values, i) for i in range(X_vif.
 ↪shape[1])]
})

print("Variance Inflation Factor (VIF):")
print(vif_data)
```

```
Variance Inflation Factor (VIF):
  feature            VIF
0   const  928263.180987
```

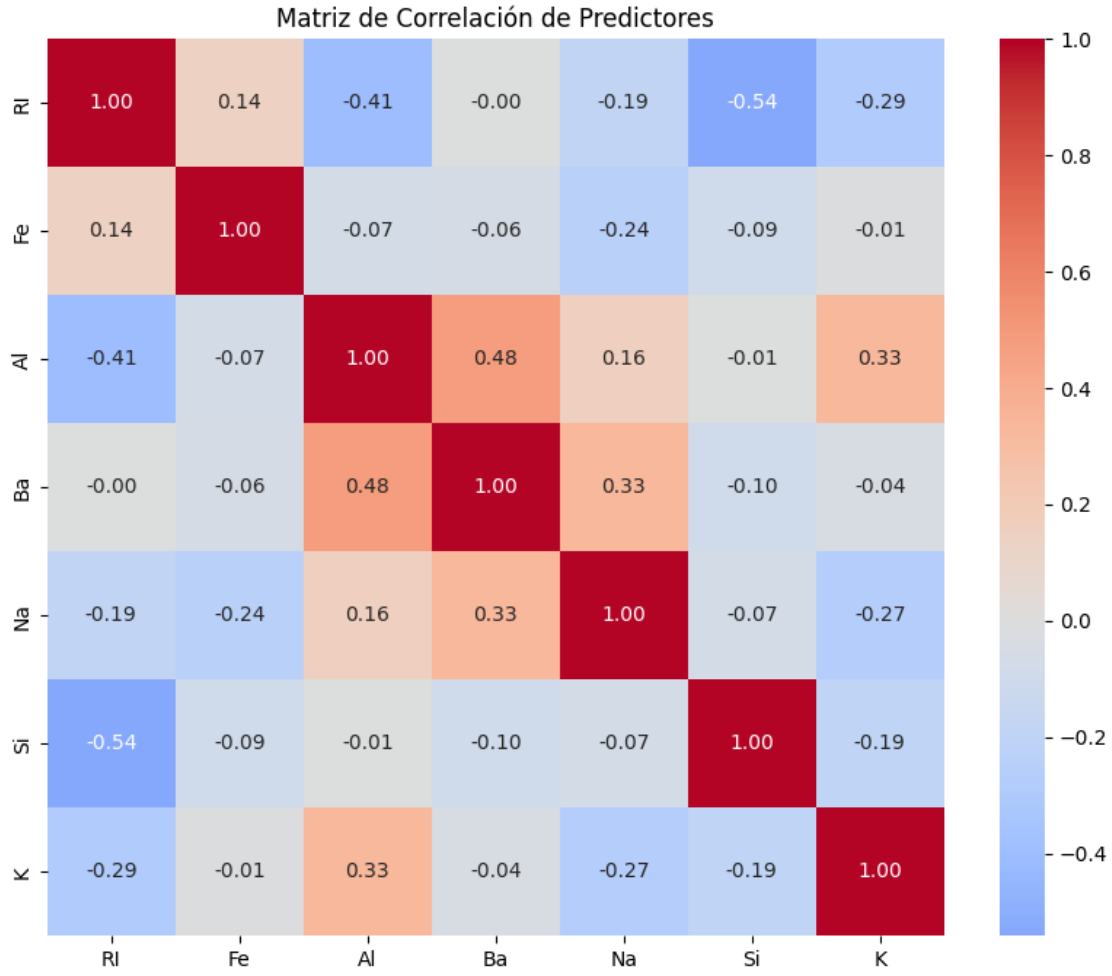
```
1      RI       2.852245
2      Fe       1.089296
3      Al       1.885572
4      Ba       1.581746
5      Na       1.689247
6      Si       2.231868
7       K       1.848954
```

Fuimos agregando las variables con menor valor y revisamos el VIF, esta fue el subconjunto que permaneció con VIF aceptables.

1.3.2 Correlación

Haremos una matriz de correlación, sin variables respuesta pues es categórica nominal.

[6]:
```
# Matriz de correlación
corr_matrix = X_small.corr()

# Visualizar
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0,
            fmt='.2f', square=True)
plt.title('Matriz de Correlación de Predictores')
plt.show()
```



Parece que no tenemos altas correlaciones

[7]:
```
y_codes = y.astype("category").cat.codes

model = sm.MNLogit(y_codes, X_small)
res = model.fit()
print(res.summary())
```

```
Optimization terminated successfully.
         Current function value: nan
         Iterations 6
                          MNLogit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  214
Model:                        MNLogit   Df Residuals:                      179
Method:                           MLE   Df Model:                           30
Date:                Wed, 18 Mar 2026   Pseudo R-squ.:                     nan
```

Time:	22:36:06	Log-Likelihood:	nan
converged:	True	LL-Null:	-322.85
Covariance Type:	nonrobust	LLR p-value:	nan

y=1	coef	std err	z	P>|z|	[0.025	0.975]
RI	nan	nan	nan	nan	nan	nan
Fe	nan	nan	nan	nan	nan	nan
Al	nan	nan	nan	nan	nan	nan
Ba	nan	nan	nan	nan	nan	nan
Na	nan	nan	nan	nan	nan	nan
Si	nan	nan	nan	nan	nan	nan
K	nan	nan	nan	nan	nan	nan

y=2	coef	std err	z	P>|z|	[0.025	0.975]
RI	nan	nan	nan	nan	nan	nan
Fe	nan	nan	nan	nan	nan	nan
Al	nan	nan	nan	nan	nan	nan
Ba	nan	nan	nan	nan	nan	nan
Na	nan	nan	nan	nan	nan	nan
Si	nan	nan	nan	nan	nan	nan
K	nan	nan	nan	nan	nan	nan

y=3	coef	std err	z	P>|z|	[0.025	0.975]
RI	nan	nan	nan	nan	nan	nan
Fe	nan	nan	nan	nan	nan	nan
Al	nan	nan	nan	nan	nan	nan
Ba	nan	nan	nan	nan	nan	nan
Na	nan	nan	nan	nan	nan	nan
Si	nan	nan	nan	nan	nan	nan
K	nan	nan	nan	nan	nan	nan

y=4	coef	std err	z	P>|z|	[0.025	0.975]
RI	nan	nan	nan	nan	nan	nan
Fe	nan	nan	nan	nan	nan	nan
Al	nan	nan	nan	nan	nan	nan
Ba	nan	nan	nan	nan	nan	nan
Na	nan	nan	nan	nan	nan	nan
Si	nan	nan	nan	nan	nan	nan
K	nan	nan	nan	nan	nan	nan

y=5	coef	std err	z	P>|z|	[0.025	0.975]
RI	nan	nan	nan	nan	nan	nan
Fe	nan	nan	nan	nan	nan	nan

```
Al              nan         nan         nan         nan         nan         nan
Ba              nan         nan         nan         nan         nan         nan
Na              nan         nan         nan         nan         nan         nan
Si              nan         nan         nan         nan         nan         nan
K               nan         nan         nan         nan         nan         nan
==============================================================================

/Users/daragama/Documents/Master25/2_ModelosEstadisticos/modelos_estadisticos/li
b/python3.12/site-packages/statsmodels/discrete/discrete_model.py:3027:
RuntimeWarning: overflow encountered in exp
  eXB = np.column_stack((np.ones(len(X)), np.exp(X)))
/Users/daragama/Documents/Master25/2_ModelosEstadisticos/modelos_estadisticos/li
b/python3.12/site-packages/statsmodels/discrete/discrete_model.py:3028:
RuntimeWarning: invalid value encountered in divide
  return eXB/eXB.sum(1)[:,None]
```

Normalmente esto sucede cuando hay algún error numérico, ya revisamos multicolinealidad y no encontramos problema por ese lado. Se intentará primero escalar la variable predictora, si esto no resulta se analizará `X`con más detalle.

[8]:
```python
y_codes = y.astype("category").cat.codes
# print names and its code
print("Codes:", y_codes.unique())

# Escalado de X para mejorar condicionamiento
scaler = StandardScaler()
X_scaled = pd.DataFrame(
    scaler.fit_transform(X),
    columns=X.columns,
    index=X.index,
 )

print("Condicion antes de escalar:", np.linalg.cond(X.to_numpy()))
X_design = sm.add_constant(X_scaled, has_constant="add")
print("Condicion despues de escalar:", np.linalg.cond(X_design.to_numpy()))

model = sm.MNLogit(y_codes, X_scaled)
res = model.fit()
print(res.summary())
```

```
Codes: [0 2 5 1 4 3]
Condicion antes de escalar: 41818.86701920228
Condicion despues de escalar: 39.50791222467228
Optimization terminated successfully.
         Current function value: 0.856528
         Iterations 9
                          MNLogit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  214
```

Model:	MNLogit	Df Residuals:	169
Method:	MLE	Df Model:	40
Date:	Wed, 18 Mar 2026	Pseudo R-squ.:	0.4323
Time:	22:36:06	Log-Likelihood:	-183.30
converged:	True	LL-Null:	-322.85
Covariance Type:	nonrobust	LLR p-value:	1.317e-37

y=1	coef	std err	z	P>|z|	[0.025	0.975]
RI	0.3870	0.823	0.470	0.638	-1.225	2.000
Na	-5.3275	2.076	-2.566	0.010	-9.397	-1.258
Mg	-10.7693	3.717	-2.897	0.004	-18.054	-3.484
Al	-0.9930	1.422	-0.698	0.485	-3.780	1.794
Si	-5.5998	1.945	-2.879	0.004	-9.411	-1.788
K	-3.8042	2.162	-1.759	0.079	-8.043	0.434
Ca	-8.9738	3.693	-2.430	0.015	-16.211	-1.737
Ba	-6.2530	1.777	-3.518	0.000	-9.736	-2.770
Fe	0.2224	0.223	0.996	0.319	-0.215	0.660

y=2	coef	std err	z	P>|z|	[0.025	0.975]
RI	-2.7530	1.018	-2.703	0.007	-4.749	-0.757
Na	-2.0767	2.523	-0.823	0.410	-7.022	2.869
Mg	-6.7540	4.589	-1.472	0.141	-15.748	2.240
Al	-1.4171	1.716	-0.826	0.409	-4.781	1.947
Si	-4.1822	2.385	-1.754	0.080	-8.857	0.492
K	-3.5274	2.556	-1.380	0.168	-8.537	1.482
Ca	-3.4739	4.632	-0.750	0.453	-12.553	5.605
Ba	-1.3844	1.770	-0.782	0.434	-4.854	2.086
Fe	-0.0744	0.320	-0.233	0.816	-0.701	0.552

y=3	coef	std err	z	P>|z|	[0.025	0.975]
RI	0.2842	1.608	0.177	0.860	-2.868	3.436
Na	0.9015	4.085	0.221	0.825	-7.105	8.908
Mg	-5.4053	7.167	-0.754	0.451	-19.453	8.642
Al	4.0681	2.743	1.483	0.138	-1.308	9.444
Si	1.1226	3.971	0.283	0.777	-6.661	8.906
K	0.7894	3.825	0.206	0.836	-6.707	8.286
Ca	-0.8384	7.222	-0.116	0.908	-14.993	13.317
Ba	-0.5900	2.675	-0.221	0.825	-5.832	4.652
Fe	-0.1419	0.447	-0.317	0.751	-1.018	0.735

y=4	coef	std err	z	P>|z|	[0.025	0.975]
RI	4.2210	1.183	3.569	0.000	1.903	6.539
Na	12.1367	5.075	2.391	0.017	2.189	22.084
Mg	11.0228	8.696	1.268	0.205	-6.022	28.067

Al	10.0205	3.256	3.078	0.002	3.640	16.401
Si	12.2266	4.915	2.488	0.013	2.593	21.860
K	7.9118	4.479	1.766	0.077	-0.867	16.691
Ca	12.4449	8.712	1.428	0.153	-4.631	29.521
Ba	6.2031	3.297	1.881	0.060	-0.259	12.666
Fe	-0.0397	0.710	-0.056	0.955	-1.431	1.351

y=5	coef	std err	z	P>|z|	[0.025	0.975]
RI	1.5513	1.705	0.910	0.363	-1.790	4.892
Na	8.8115	4.931	1.787	0.074	-0.854	18.477
Mg	6.0409	8.348	0.724	0.469	-10.321	22.403
Al	6.9294	3.207	2.160	0.031	0.643	13.216
Si	7.5764	4.796	1.580	0.114	-1.824	16.976
K	3.0108	4.327	0.696	0.487	-5.469	11.491
Ca	8.3235	8.461	0.984	0.325	-8.260	24.907
Ba	3.1419	3.180	0.988	0.323	-3.090	9.374
Fe	-0.1077	0.656	-0.164	0.870	-1.394	1.179

1.4 Interpretación del Summary

En regresión logística multinomial se estiman $k-1$ conjuntos de coeficientes, uno por cada clase de respuesta respecto a la clase base ($y = 0$).

Para cada clase, las variables con p-valor menor a 0.05 aportar mayor significancia estadística a los log-odds de pertenecer a esa clase (frente a la clase base). Por ejemplo, para $y = 1$ son `Na` ($p = 0.010$), `Mg` ($p = 0.004$), `Si` ($p = 0.004$), `Ca` ($p = 0.015$) y `Ba` ($p = 0.000$). Sus coeficientes negativos indican que, al aumentar esas variables (en la escala usada en el modelo), disminuyen los log-odds relativos de estar en $y = 1$ frente a $y = 0$.

En la clase $y = 3$, varios coeficientes son cercanos a 0, lo que sugiere menor diferenciación respecto a la clase base. En todas las clases hay algunos coeficientes con p-valores altos; listamos tres ejemplos por clase: - $y = 1$: `RI`, `Al`, `Fe` - $y = 2$: `Fe`, `Ca`, `Ba` - $y = 3$: Hay muchos con p-valor alto, solamente `Al` tiene p-valor < 0.05 - $y = 4$: `Fe`, `Mg`, `K`

El `Fe` aparece en todas las listas, por lo que podría ser una variable poco informativa o parcialmente redundante. Por sería conveniente evaluar el modelo sin `Fe`.

Tenemos como base el log-likelihood de un modelo nulo sin predictores `LL-Null` = -322.85, el log-likelihood de nuestro modelo `Log-Likelihood` = -183.30 con los predictores elegidos si mejora con respecto al nulo. El cuánto mejora el modelo está dado por el pseudo-R^2 que se explicará con mayor detenimiento más adelante.

Por el p-valor de LLR = 1.317e-37 que es extremadamente pequeño tenemos evidencia para rechazar H_0 : todos los coeficientes son cero y decimos que al menos uno de los coeficientes no es cero. Es decir al menos una variable contribuye significativamente al modelo.

En general, el modelo tiene capacidad explicativa pero puede mejorar.

1.5 Búsqueda de Outliers

[9]:
```
# Calculo de residuales
y_pred = res.predict(X_scaled)
n = len(y)
J = len(y_pred.columns)

# Residualess de Pearson
pearson_resid = []
for i in range(n):
    obs = y_codes.iloc[i] if isinstance(y_codes, pd.Series) else y[i]
    y_pred_i = y_pred.iloc[i] if hasattr(y_pred, 'iloc') else y_pred[i]

    # Para la categoría observada
    resid = (1 - y_pred_i[obs]) / np.sqrt(y_pred_i[obs] * (1 - y_pred_i[obs]))
    pearson_resid.append(resid)

residuals = np.array(pearson_resid)


# Varianza
residual_sum_squares = np.sum(residuals**2) # RSS
n = len(y)
p = df.shape[1]
sigma_squared = residual_sum_squares / (n - p - 1)
sigma = np.sqrt(sigma_squared)
standardized_residuals = residuals / sigma
```

1.6 Distancia de Cook y Leverage

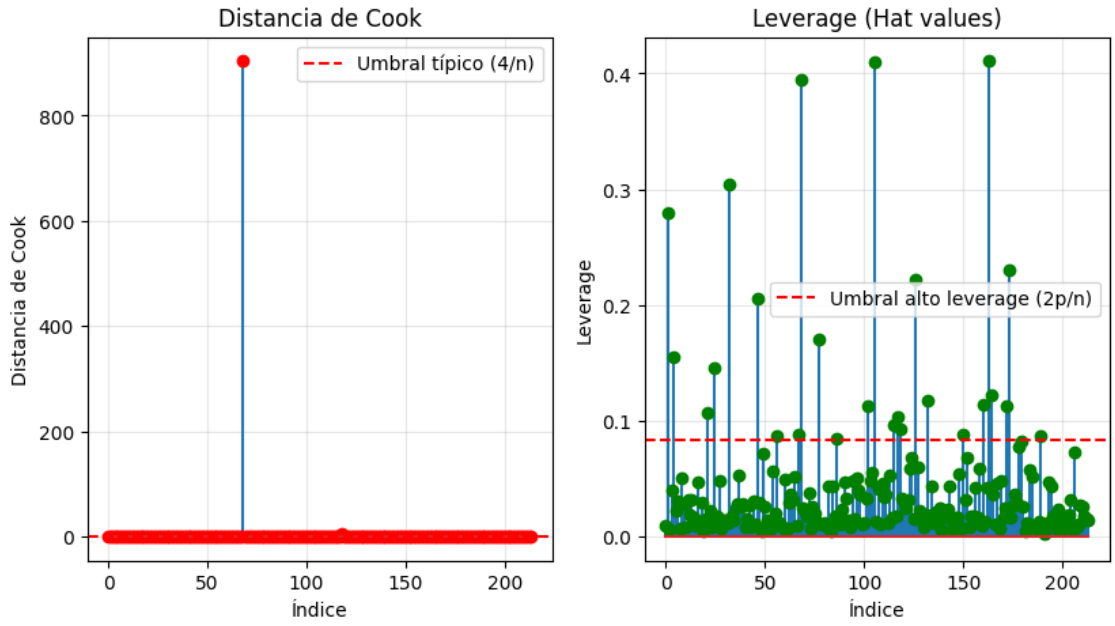
[10]:
```
n, p = X_scaled.shape
J = len(np.unique(y_codes))
X_design = np.array(X_scaled)
hat_matrix_diag = np.diag(X_design @ np.linalg.inv(X_design.T @ X_design) @␣
 ↪X_design.T)
# Leverage alto
thres_leverage = 2*p/len(y)
high_leve = np.where(np.abs(hat_matrix_diag) > thres_leverage)[0]
porcentage = (len(high_leve) / len(y_codes))*100
print(f"Observaciones con leverage alto: ({len(high_leve)} indices -␣
 ↪{porcentage}%) {high_leve}")

std_resid = residuals / np.sqrt(1 - hat_matrix_diag)
cooks_d = (std_resid**2 / (p * (J-1))) * (hat_matrix_diag / (1 -␣
 ↪hat_matrix_diag)**2)
# Cook alto
thres_cooks = 4/len(y_codes)
```

```
high_dist = np.where(np.abs(cooks_d) > thres_cooks)[0]
high_dist = high_dist[np.argsort(cooks_d[high_dist])[::-1]] # ordenamos por␣
 ↪distancia de cook
porcentage_c = (len(high_dist) / len(y_codes))*100
# ordenamos por distancia de cook

print(f"Observaciones con cook alto: ({len(high_dist)} indices -␣
 ↪{porcentage_c}%) {high_dist}")
```

```
Observaciones con leverage alto: (25 indices - 11.682242990654206%) [  1   4  21
24  32  46  56  67  68  77  86 102 105 115 117 118 126 132
 150 160 163 164 172 173 189]
Observaciones con cook alto: (7 indices - 3.2710280373831773%) [ 68 118   1 183
77  39  67]
```

Las observaciones con leverage alto tienen un porcentaje razonable de total de obsrvaciones, pero no es métrica suficiente para considerarlos outliers, las observaciones con distancia de Cook grande se eliminarán. Grafiquemos primero los Leverage y distiancia de Cook y luego los residuales para ayuda visual.

[11]:
```
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.stem(range(len(cooks_d)), cooks_d, markerfmt='ro')
plt.axhline(y=4/len(y_codes), color='r', linestyle='--', label='Umbral típico␣
 ↪(4/n)')
plt.xlabel('Índice')
plt.ylabel("Distancia de Cook")
plt.title('Distancia de Cook')
plt.legend()
plt.grid(True, alpha=0.3)

plt.subplot(1, 3, 2)
plt.stem(range(len(hat_matrix_diag)), hat_matrix_diag, markerfmt='go')
plt.axhline(y=2*X_scaled.shape[1]/len(y), color='r', linestyle='--',
            label='Umbral alto leverage (2p/n)')
plt.xlabel('Índice')
plt.ylabel('Leverage')
plt.title('Leverage (Hat values)')
plt.legend()
plt.grid(True, alpha=0.3)


plt.legend()
plt.grid(True, alpha=0.3)
```

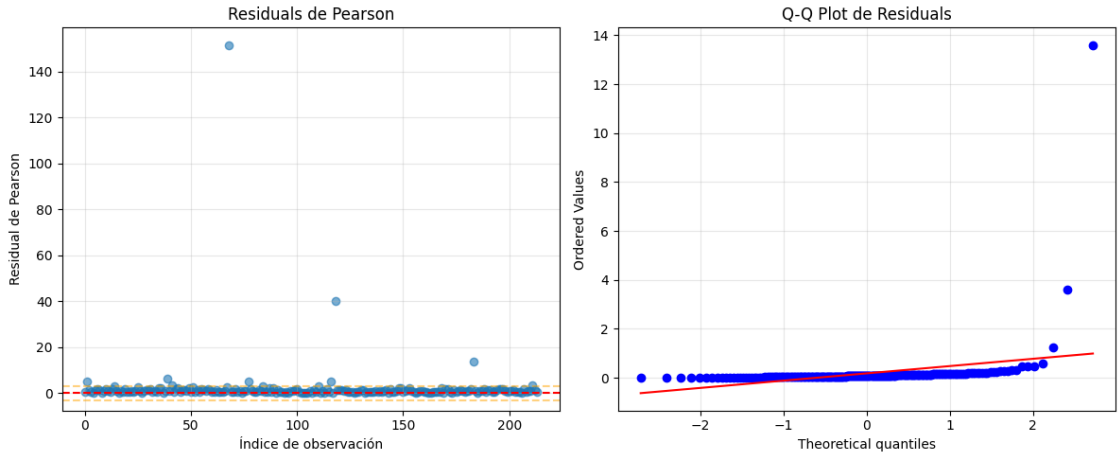


Vemos una observación en particular con distancia de Cook grande podríamos comenzar quitando esa única.

Las 25 observaciones calculadas previamente se pueden ver en la gráfica de Leverage.

[12]:
```
from scipy import stats

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.scatter(range(len(pearson_resid)), pearson_resid, alpha=0.6)
plt.axhline(y=0, color='r', linestyle='--')
plt.axhline(y=3, color='orange', linestyle='--', alpha=0.5)
plt.axhline(y=-3, color='orange', linestyle='--', alpha=0.5)
plt.xlabel('Índice de observación')
plt.ylabel('Residual de Pearson')
plt.title('Residuals de Pearson')
plt.grid(True, alpha=0.3)

ax2 = plt.subplot(1, 2, 2)
stats.probplot(standardized_residuals, dist="norm", plot=ax2)
ax2.set_title('Q-Q Plot de Residuals')
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

```
# Dificil de observar visualmente,
outliers = np.where(np.abs(pearson_resid) > 3)[0]
print(f"Observaciones con residual > 3: ({len(outliers)}) {outliers}")
```



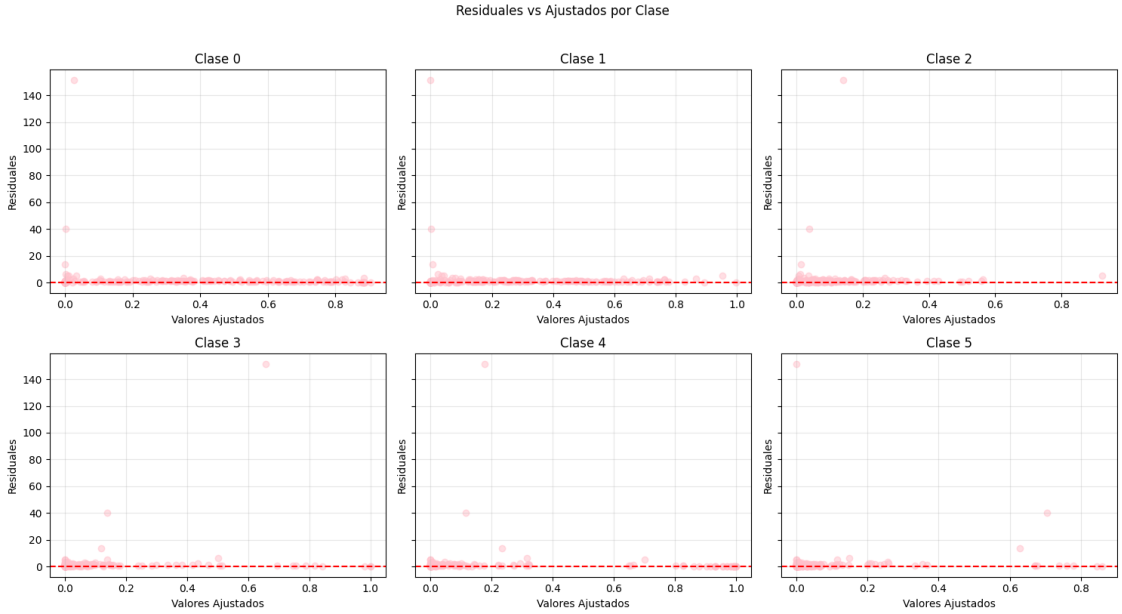

```
Observaciones con residual > 3: (12) [  1  14  39  41  68  77  84 110 116 118
183 211]
```

Se usa tres como threshold basado en la distribución normal $P|z| > 3 \approx 0.0027$ Es decir un residual con una probabilidad muy baja de ocurrir.

Hay 12 valores con alto residual, y vemos que los residuales tienen diferente inclinación que la línea normal con al menos 2 observaciones que visualmente es muy claro que son outliers.

[13]:
```
# Grafica de residuales vs ajustados por clase
from statsmodels.graphics.gofplots import ProbPlot

y_pred_df = y_pred if isinstance(y_pred, pd.DataFrame) else pd.DataFrame(y_pred)

n_classes = y_pred_df.shape[1]
n_cols = 3
n_rows = int(np.ceil(n_classes / n_cols))

fig, axes = plt.subplots(n_rows, n_cols, figsize=(5 * n_cols, 4 * n_rows),␣
 ↪sharey=True)
axes = np.atleast_1d(axes).ravel()
color = "pink"

for j, ax in enumerate(axes):
    if j < n_classes:
        ax.scatter(y_pred_df.iloc[:, j], residuals, alpha=0.5, color=color)
        ax.axhline(y=0, color="r", linestyle="--")
        ax.set_xlabel("Valores Ajustados")
```

```
        ax.set_ylabel("Residuales")
        ax.set_title(f"Clase {y_pred_df.columns[j]}")
        ax.grid(True, alpha=0.3)
    else:
        ax.axis("off")

fig.suptitle("Residuales vs Ajustados por Clase", y=1.02)
plt.tight_layout()
plt.show()
```



A diferencia de la regresión lineal, tenemos un vector de probabilidades por cada observación, por lo que las gráficas anteriores muestran residual vs. $\hat{p}_{ik}$ para cada clase k.

Los residuales de Pearson que usamos aquí son una aproximación diagnóstica (por clase observada), útiles para detectar patrones y posibles observaciones influyentes y conviene interpretarlos junto con otras evidencias (Cook, matriz de confusión, AUC y pruebas de razón de verosimilitud), y no como criterio único.

En todas las categorías podemos identificar al menos 3 observaciones potencialmente atípicas.

Vamos a quitar aquellas observaciones que aparecieron al calcular la distancia de Cook y volver a ajustar el modelo.

[14]:
```
X_no_outliers = X_scaled.drop(index=high_dist)
y_no_outliers = y_codes.drop(index=high_dist)
print(high_dist)
```

```
[ 68 118   1 183  77  39  67]
```

1.7 Segundo Ajuste de Residuales

[15]:
```
print("Codes:", y_codes.unique())
model_sin_outliers = sm.MNLogit(y_no_outliers, X_no_outliers)
res_sin_outliers = model_sin_outliers.fit()
print(res_sin_outliers.summary())
```

```
Codes: [0 2 5 1 4 3]
Optimization terminated successfully.
         Current function value: 0.560671
         Iterations 13
                          MNLogit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  207
Model:                        MNLogit   Df Residuals:                      162
Method:                           MLE   Df Model:                           40
Date:                Wed, 18 Mar 2026   Pseudo R-squ.:                 0.6285
Time:                        22:36:07   Log-Likelihood:                -116.06
converged:                       True   LL-Null:                       -312.41
Covariance Type:            nonrobust   LLR p-value:                 1.791e-59
==============================================================================
       y=1       coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
RI             0.5252      0.847      0.620      0.535      -1.134       2.185
Na            -5.2618      2.224     -2.366      0.018      -9.621      -0.903
Mg           -13.8186      4.101     -3.369      0.001     -21.857      -5.780
Al            -1.5877      1.486     -1.068      0.285      -4.500       1.325
Si            -5.9265      2.072     -2.860      0.004      -9.988      -1.865
K             -4.3333      2.253     -1.923      0.054      -8.750       0.083
Ca           -10.8903      4.010     -2.716      0.007     -18.750      -3.030
Ba           -10.6785      2.371     -4.504      0.000     -15.325      -6.032
Fe             0.3332      0.246      1.353      0.176      -0.150       0.816
------------------------------------------------------------------------------
       y=2       coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
RI            -7.6985      1.765     -4.361      0.000     -11.159      -4.238
Na             4.7687      3.852      1.238      0.216      -2.781      12.318
Mg             5.8174      6.722      0.865      0.387      -7.358      18.993
Al             2.1678      2.395      0.905      0.365      -2.527       6.862
Si             0.6321      3.571      0.177      0.860      -6.368       7.632
K              2.0906      3.643      0.574      0.566      -5.049       9.230
Ca            12.4539      6.998      1.780      0.075      -1.261      26.169
Ba             5.2192      3.391      1.539      0.124      -1.428      11.866
Fe             0.1533      0.428      0.358      0.720      -0.685       0.992
------------------------------------------------------------------------------
       y=3       coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
RI            -1.8034      3.352     -0.538      0.591      -8.373       4.766
```

```
Na            -6.5745      6.337     -1.038      0.299     -18.994       5.845
Mg           -25.7879     12.395     -2.081      0.037     -50.082      -1.494
Al             2.5238      4.590      0.550      0.582      -6.472      11.520
Si            -3.7605      6.063     -0.620      0.535     -15.644       8.123
K             -3.0465      6.008     -0.507      0.612     -14.822       8.729
Ca           -13.4159     12.080     -1.111      0.267     -37.092      10.260
Ba            10.1964      6.093      1.673      0.094      -1.746      22.139
Fe            -2.6224      1.148     -2.285      0.022      -4.872      -0.373
------------------------------------------------------------------------------
       y=4       coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
RI             9.8960      2.940      3.366      0.001       4.133      15.659
Na             6.1120     13.703      0.446      0.656     -20.746      32.970
Mg           -20.0867     29.028     -0.692      0.489     -76.980      36.807
Al             6.9838      9.252      0.755      0.450     -11.150      25.118
Si             6.7633     13.782      0.491      0.624     -20.249      33.776
K              1.4084     12.779      0.110      0.912     -23.637      26.454
Ca           -16.8738     28.967     -0.583      0.560     -73.649      39.901
Ba            10.7752      8.560      1.259      0.208      -6.002      27.552
Fe            -5.3018      1.887     -2.810      0.005      -9.000      -1.603
------------------------------------------------------------------------------
       y=5       coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
RI            -0.4170      5.141     -0.081      0.935     -10.492       9.658
Na            20.6893     12.947      1.598      0.110      -4.687      46.065
Mg             9.5237     21.244      0.448      0.654     -32.114      51.161
Al            12.0724      8.438      1.431      0.153      -4.466      28.610
Si            18.2634     11.814      1.546      0.122      -4.891      41.418
K              7.9867     11.267      0.709      0.478     -14.095      30.069
Ca            18.0294     22.204      0.812      0.417     -25.490      61.549
Ba            14.8104      8.266      1.792      0.073      -1.391      31.011
Fe            -1.5409      1.668     -0.924      0.356      -4.811       1.729
==============================================================================
```

Ahora tratamos de eliminar `Fe` como revisamos anteriormente:

[16]:
```
print("Codes:", y_codes.unique())
X_smaller = X_no_outliers.drop(columns=["Fe"])
model = sm.MNLogit(y_no_outliers, X_smaller)
res = model.fit()
print(res.summary())
```

```
Codes: [0 2 5 1 4 3]
Optimization terminated successfully.
         Current function value: 0.596618
         Iterations 12
                          MNLogit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  207
```

Model:	MNLogit	Df Residuals:	167
Method:	MLE	Df Model:	35
Date:	Wed, 18 Mar 2026	Pseudo R-squ.:	0.6047
Time:	22:36:07	Log-Likelihood:	-123.50
converged:	True	LL-Null:	-312.41
Covariance Type:	nonrobust	LLR p-value:	4.200e-59

y=1	coef	std err	z	P>|z|	[0.025	0.975]
RI	0.3830	0.826	0.464	0.643	-1.236	2.002
Na	-6.4025	2.182	-2.935	0.003	-10.678	-2.127
Mg	-14.8312	4.066	-3.648	0.000	-22.800	-6.862
Al	-2.1918	1.455	-1.506	0.132	-5.044	0.661
Si	-6.8419	2.042	-3.350	0.001	-10.845	-2.839
K	-5.0835	2.249	-2.260	0.024	-9.492	-0.675
Ca	-12.1518	3.981	-3.053	0.002	-19.954	-4.350
Ba	-10.1275	2.266	-4.469	0.000	-14.569	-5.686

y=2	coef	std err	z	P>|z|	[0.025	0.975]
RI	-6.8581	1.594	-4.303	0.000	-9.982	-3.735
Na	3.9975	3.620	1.104	0.270	-3.098	11.093
Mg	4.5002	6.312	0.713	0.476	-7.870	16.871
Al	1.6912	2.281	0.741	0.458	-2.780	6.162
Si	0.3097	3.377	0.092	0.927	-6.310	6.929
K	1.7743	3.446	0.515	0.607	-4.980	8.529
Ca	10.4492	6.541	1.598	0.110	-2.370	23.268
Ba	4.4542	3.044	1.463	0.143	-1.513	10.421

y=3	coef	std err	z	P>|z|	[0.025	0.975]
RI	-2.6358	2.628	-1.003	0.316	-7.786	2.514
Na	4.2810	6.851	0.625	0.532	-9.146	17.708
Mg	-0.9333	11.142	-0.084	0.933	-22.772	20.905
Al	7.6574	4.874	1.571	0.116	-1.895	17.210
Si	4.9338	7.074	0.697	0.486	-8.931	18.799
K	3.6449	6.548	0.557	0.578	-9.189	16.479
Ca	7.7810	11.288	0.689	0.491	-14.343	29.905
Ba	9.8150	5.550	1.768	0.077	-1.064	20.694

y=4	coef	std err	z	P>|z|	[0.025	0.975]
RI	7.6248	2.075	3.675	0.000	3.559	11.691
Na	19.0256	9.892	1.923	0.054	-0.363	38.414
Mg	12.5196	17.850	0.701	0.483	-22.466	47.505
Al	14.0792	7.038	2.001	0.045	0.285	27.873
Si	18.4428	10.183	1.811	0.070	-1.516	38.401
K	11.4643	9.125	1.256	0.209	-6.421	29.349

Ca	13.7390	17.989	0.764	0.445	-21.519	48.997
Ba	14.4166	7.003	2.059	0.040	0.692	28.141

y=5	coef	std err	z	P>|z|	[0.025	0.975]
RI	1.5097	4.055	0.372	0.710	-6.438	9.457
Na	18.6736	8.157	2.289	0.022	2.686	34.662
Mg	13.9013	13.060	1.064	0.287	-11.695	39.498
Al	11.3353	5.922	1.914	0.056	-0.271	22.942
Si	16.5039	7.990	2.066	0.039	0.844	32.164
K	8.2200	6.977	1.178	0.239	-5.455	21.895
Ca	18.8012	13.439	1.399	0.162	-7.539	45.141
Ba	11.6887	5.711	2.047	0.041	0.495	22.883

Al quitar `Fe` la pseudo R^2 disminuyó de 0.6285 a 0.6047. Vamos a comparar los coeficientes por categoría.

Para $y = 1$ el modelo no era muy bueno para está categoría, las probabilidades de no significancia disminuyeron y ahora solo 2/8 de los intervalos de confianza incluyen el cero cuando en el modelo anterior todos lo incluían, es decir que para esta clase se estabilizó el modelo.

Para $y = 2$ no hubo tanta mejora practicamente mantenemos los mismos coeficientes e intervalos para las variables remanentes.

Para $y = 3$ sus coeficientes aumentaron, 2 ya no incluyen al cero en sus intervalos y la probabilidad de no significancia de todas las variables disminuyó. Mucha mejora en esta categoría

Para $y = 4$ y $y = 5$, estas categorías eran las más estables en el modelo anterior y lo siguen siendo, tiene coeficientes muy altos, es decir que el vidrio tiene altas concentraciones de todos los elementos, y al quitar al `Fe` que en ambas era la menos significantes tenemos que todas las variables tienen alta probabilidad de ser significantes.

1.8 Likelihood Ratio Test

Vamos a comparar el modelo con `Fe` (modelo completo) y el modelo sin `Fe` (modelo reducido).

El estadístico de razón de verosimilitud se define como:

$$LR = 2\left(\ell_{completo} - \ell_{reducido}\right)$$

donde ℓ es el log-likelihood maximizado de cada modelo.

Concluiremos que `Fe` no aporta información adicional si ambos log-likelihoods son parecidos y el LR será pequeño.

Si `Fe` sí aporta, el modelo completo tendrá mayor log-likelihood y el LR será grande.

Bajo la hipótesis nula H_0 (el modelo reducido es suficiente), el estadístico LR se distribuye aproximadamente como una chi-cuadrado:

$$LR \sim \chi^2_{df}$$

con grados de libertad

$$df = k_{completo} - k_{reducido}$$

donde k es el número de parámetros estimados en cada modelo.

Luego se calcula el p-valor y se decide:

- Si p-valor < 0.05, se rechaza H_0: el modelo con `Fe` mejora significativamente el ajuste.
- Si p-valor $>= 0.05$, no hay evidencia suficiente para mantener `Fe`.

[17]:
```
# full model vs sin fe
full = res_sin_outliers.llf
sin_fe = res.llf

LR = 2 * (full - sin_fe)
print(f"Estadístico de la prueba de razón de verosimilitud: {LR:.4f}")

n_categorias = len(np.unique(y_codes))  # 6
k_completo = res_sin_outliers.df_model
k_sin_fe = res.df_model

df = k_completo - k_sin_fe
print(f"Grados de libertad: {df}")

p_value = 1 - stats.chi2.cdf(LR, df)
print(f"p-valor: {p_value:.6f}")
```

```
Estadístico de la prueba de razón de verosimilitud: 14.8821
Grados de libertad: 5.0
p-valor: 0.010878
```

Como el p-valor (0.011) es menor que 0.05, rechazamos la hipótesis nula. Por lo tanto, el modelo completo (con Fe) es significativamente mejor que el modelo sin Fe.

1.9 Pseudo-R^2

A diferencia del R^2 en regresión lineal que mide la varianza explicada, el objetivo de las pseudo-R^2 es medir qué tanto mejora un modelo con respecto al modelo nulo (sin variables predictoras). Este modelo nulo tiene un log-likelihood ℓ_0, que representa el caso base.

En modelos logísticos (como la regresión logística multinomial), el R^2 clásico de regresión lineal no aplica, por eso se usan medidas análogas llamadas pseudo-R^2.
La que reporta `statsmodels` en el `summary` de `MNLogit` es la de McFadden:

$$R^2_{\text{McFadden}} = 1 - \frac{\ell_{\text{modelo}}}{\ell_0}$$

donde: - ℓ_{modelo}: log-likelihood del modelo con predictores. - ℓ_0: log-likelihood del modelo nulo.

Si el modelo con predictores apenas mejora al nulo, el cociente $\ell_{\text{modelo}}/\ell_0$ será cercano a 1 y el pseudo-R^2 será pequeño. Si mejora mucho, el cociente disminuye y el pseudo-R^2 aumenta.

En este análisis, valores alrededor de 0.60 indican una mejora fuerte del ajuste respecto al caso base. Además de McFadden, hay otras pseudo-R^2 como:

- **Cox–Snell**

$$R^2_{\text{CS}} = 1 - \left(\frac{L_0}{L_{\text{modelo}}}\right)^{\frac{2}{n}}$$

  donde L_0 y L_{modelo} son likelihoods (no log-likelihoods) y n es el tamaño de muestra. Su límite superior puede ser menor que 1.

- **Nagelkerke** (ajuste de Cox–Snell para escalar a $[0, 1]$)

$$R^2_{\text{N}} = \frac{R^2_{\text{CS}}}{1 - L_0^{\frac{2}{n}}}$$

- **McFadden ajustado**
  penaliza por número de parámetros, por lo que suele ser menor que el de McFadden simple.

Por ello, la interpretación debe hacerse en términos comparativos entre modelos (por ejemplo, con y sin ciertas variables), más que como "porcentaje de varianza explicada" en sentido estricto. Hay otras pseudo-R^2 como

El cálculo de la de McFadden sería:

[18]:
```
# Calculo manual de r^2 macfadden
print(f"Log Verosimilitud del modelo nulo: {res_sin_outliers.llnull}")
print(f"Log Verosimilitud del modelo ajustado: {res_sin_outliers.llf}")

r2_mcfadden = 1 - (res_sin_outliers.llf / res_sin_outliers.llnull)
print(f"R^2 de McFadden: {r2_mcfadden:.4f}")
print(f"R^2 de McFadden (usando prsquared): {res_sin_outliers.prsquared:.4f}")
```

```
Log Verosimilitud del modelo nulo: -312.40778592047354
Log Verosimilitud del modelo ajustado: -116.05890096633483
R^2 de McFadden: 0.6285
R^2 de McFadden (usando prsquared): 0.6285
```

1.10 Predicción

[19]:
```
new_data = {
    "RI": [1.517],
    "Na": [13.64],
    "Mg": [4.49],
    "Al": [1.10],
    "Si": [71.78],
    "K": [0.06],
    "Ca": [8.75],
```

```
    "Ba": [0.00],
    "Fe": [0.00],  # se ignora luego, pero se mantiene para escalar igual que␣
 ↪en entrenamiento
}

new_df = pd.DataFrame(new_data).reindex(columns=X.columns)

# Escalar
new_df_scaled = pd.DataFrame(
    scaler.transform(new_df),
    columns=X.columns,
    index=new_df.index
)

new_df_model = new_df_scaled[["RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba",␣
 ↪"Fe"]]

# Predicción de probabilidades
pred = res_sin_outliers.predict(new_df_model)
display(pred)

# imprimir codigo y nombre
pred_code = pred.idxmax(axis=1).astype(int)
cat_names = y.astype("category").cat.categories
pred_name = pred_code.map(dict(enumerate(cat_names)))

print("\nClase predicha (código):", pred_code.iloc[0])
print("Clase predicha (nombre):", pred_name.iloc[0])
```

	0	1	2	3	4	5
0	0.00847	0.000337	0.991192	2.192112e-15	5.598912e-20	6.927259e-13

```
Clase predicha (código): 2
Clase predicha (nombre): vehic wind float
```

1.11 Matriz de confusión

[20]:
```
# matriz de confusión
from sklearn.metrics import confusion_matrix, classification_report

y_pred = res.predict(X_smaller)
y_pred_classes = y_pred.idxmax(axis=1).astype(int)
cm = confusion_matrix(y_no_outliers, y_pred_classes)
print("Matriz de Confusión:")
display(cm)
```

```
Matriz de Confusión:
```

```
array([[48, 16,  4,  0,  1,  0],
       [18, 48,  3,  2,  1,  0],
       [ 5,  4,  6,  0,  0,  0],
       [ 0,  1,  0, 12,  0,  0],
       [ 0,  0,  0,  0, 29,  0],
       [ 0,  0,  0,  0,  0,  9]])
```

En la matriz las filas son los valores reales y las columnas los valores predecidos, entonces en la diagonal están los elementos en los que sí le atinó y fuera de ella los errores de predicción.

Vemos que en la clase $y = 0$, predijo correctamente 48 veces, pero otras 16 las predijo como clase $y = 1$, otras 4 como clase $y = 2$ y 1 más como clase $y = 4$.

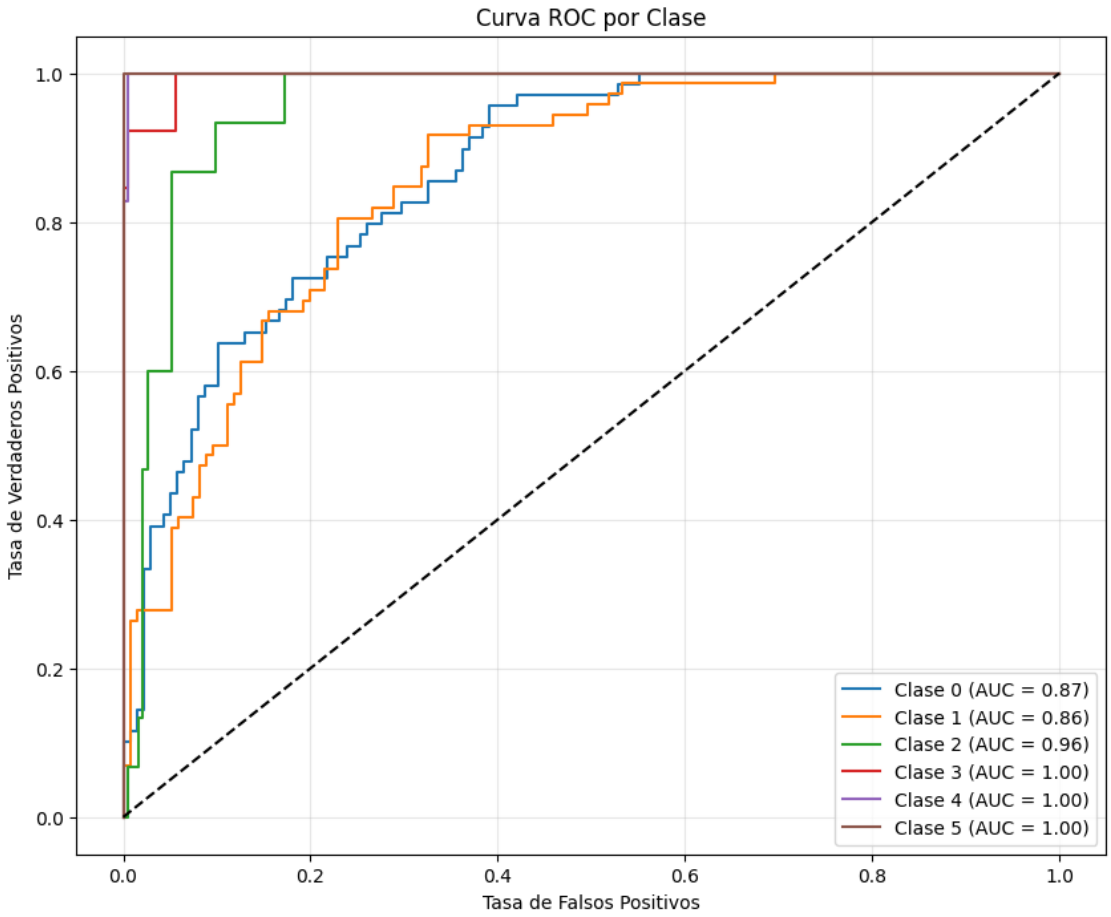
La clase $y = 2$ fue la que más errores tuvo proporcionalmente: de 15 observaciones reales, solo acertó 6, y la mayoría (9) los confundió con clases $y = 0$ y $y = 1$.

En general el modelo se confundió más entre las clases $y = 0$, $y = 1$ y $y = 2$. Las clases $y = 4$ y $y = 5$ las predijo perfectamente (29 y 9 aciertos respectivamente), aunque tuvo falsos positivos: 1 observación predicha como clase $y = 4$ que realmente era $y = 0$, y 1 predicha como clase $y = 5$ que realmente era $y = 0$.

La clase $y = 3$ tuvo 12 aciertos, pero presentó 1 falso positivo (clase $y = 1$ predicha como $y = 3$) y 1 falso negativo (clase $y = 3$ predicha como $y = 1$).

1.12 Curva ROC y AUC

[21]:
```python
# curva ROC y AUC para cada clase
from sklearn.metrics import roc_curve, auc
plt.figure(figsize=(10, 8))
for i in range(len(np.unique(y_codes))):
    fpr, tpr, _ = roc_curve((y_no_outliers == i).astype(int), y_pred.iloc[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'Clase {i} (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('Curva ROC por Clase')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```



Un mal indicador en esta gráfica es aquel que se acerque mucho a la línea diagonal, ya que esto refleja un rendimiento cercano al azar. La curva ROC resume el rendimiento de cada clase.

En este caso, las curvas más cercanas a la diagonal son las de las clases $y = 0$ (AUC = 0.87) y $y = 1$ (AUC = 0.86), que como vimos en la matriz de confusión, son las que presentaron mayores confusiones entre sí. Esto indica que estas dos clases no están muy bien separadas por el modelo.

La clase $y = 2$ presenta un AUC de 0.96, lo que indica una excelente discriminación a pesar de los errores observados en la matriz de confusión (recordemos que el AUC mide el ordenamiento de probabilidades, no la clasificación con umbral fijo).

Las clases $y = 3$, $y = 4$ y $y = 5$ tienen un AUC de 1.00, lo que indica que el modelo puede ordenar perfectamente las probabilidades para estas clases (existe algún umbral que las clasifica sin error), consistente con los aciertos perfectos observados en la matriz para $y = 4$ y $y = 5$.

1.13 Hosmer–Lemeshow Test

El estadístico Hosmer–Lemeshow evalúa el ajuste de un modelo logístico agrupando observaciones con riesgo similar.

H_0: no hay diferencia significativa entre valores observados y predichos (buen ajuste)

H_1: hay diferencia significativa (ajuste deficiente)

El estadístico es:

$$HL = \sum_{g=1}^{G} \frac{(O_g - E_g)^2}{E_g(1 - E_g/n_g)}$$

Donde: - G: número de grupos (típicamente 10 deciles) - O_g: eventos observados en el grupo g - E_g: eventos esperados en el grupo g - n_g: total de observaciones en el grupo g

Bajo H_0, el estadístico HL sigue una distribución chi-cuadrado: $HL \sim \chi^2_{G-2}$ con $G - 2$ grados de libertad.

En la práctica, al agrupar con cuantiles puede haber grupos repetidos y el número efectivo de grupos puede ser menor que G. Por eso, los grados de libertad deben ajustarse al número de grupos efectivos:

$$df = G_{efectivo} - 2$$

Si p-valor < 0.05: se rechaza H_0 (ajuste deficiente).

[22]:
```python
# Hosmer-Lemeshow Test con grados de libertad efectivos
def hosmer_lemeshow(y_true, y_prob, g=10):
    data = pd.DataFrame({"y": y_true, "prob": y_prob})
    data["group"] = pd.qcut(data["prob"], g, duplicates="drop")

    obs = data.groupby("group", observed=True)["y"].sum()
    exp = data.groupby("group", observed=True)["prob"].sum()
    n = data.groupby("group", observed=True)["y"].count()

    # Evita divisiones por cero numéricas en grupos extremos
    denom = exp * (1 - exp / n)
    denom = denom.clip(lower=1e-12)

    hl_stat = np.sum((obs - exp) ** 2 / denom)

    g_eff = int(n.shape[0])
    df_hl = max(g_eff - 2, 1)
    p_value = 1 - stats.chi2.cdf(hl_stat, df_hl)
    return hl_stat, p_value, g_eff, df_hl

# Para cada clase, calcular el test Hosmer-Lemeshow one-vs-rest
for class_idx in range(y_pred.shape[1]):
    y_binary = (y_no_outliers == class_idx).astype(int)
    y_prob_class = y_pred.iloc[:, class_idx]

    hl_stat, hl_p, g_eff, df_hl = hosmer_lemeshow(y_binary, y_prob_class)
```

```
    print(f"\nClase {class_idx} - Hosmer-Lemeshow Test:")
    print(f"  Statistic: {hl_stat:.4f}")
    print(f"  Grupos efectivos: {g_eff}")
    print(f"  Grados de libertad: {df_hl}")
    print(f"  p-value: {hl_p:.6f}")
    if hl_p > 0.05:
        print("  Buen ajuste")
    else:
        print("  Ajuste deficiente")
```

```
Clase 0 - Hosmer-Lemeshow Test:
  Statistic: 5.1138
  Grupos efectivos: 10
  Grados de libertad: 8
  p-value: 0.745347
  Buen ajuste

Clase 1 - Hosmer-Lemeshow Test:
  Statistic: 699.5536
  Grupos efectivos: 10
  Grados de libertad: 8
  p-value: 0.000000
  Ajuste deficiente

Clase 2 - Hosmer-Lemeshow Test:
  Statistic: 2.6214
  Grupos efectivos: 10
  Grados de libertad: 8
  p-value: 0.955831
  Buen ajuste

Clase 3 - Hosmer-Lemeshow Test:
  Statistic: 0.4983
  Grupos efectivos: 10
  Grados de libertad: 8
  p-value: 0.999868
  Buen ajuste

Clase 4 - Hosmer-Lemeshow Test:
  Statistic: 0.4395
  Grupos efectivos: 10
  Grados de libertad: 8
  p-value: 0.999918
  Buen ajuste

Clase 5 - Hosmer-Lemeshow Test:
  Statistic: 0.5616
```

```
Grupos efectivos: 10
Grados de libertad: 8
p-value: 0.999793
Buen ajuste
```

Solamente la clase $y = 1$ tiene un ajuste deficiente. En general este estadístico es un forma de medir que tan bien "calibrado" está el modelo, osea que lo que nos dice el modelo es lo que está realmente pasando. Con la ROC curve podemos ver que tan bien diferencia el modelo a cada clase.

1.14 Modelo alternativo

El modelo que se usará es `sm.Probit`, como solo acepta variables binarias, se hará un ajuste por cada clase, con es (=1) o no es (=0).

[23]:
```
classes = sorted(y_no_outliers.unique())
X_probit = sm.add_constant(X_no_outliers, has_constant="add")

probit_models_res = {}
for cls in classes:
    y_bin = (y_no_outliers == cls).astype(int)
    probit_model = sm.Probit(y_bin, X_probit)
    probit_res = probit_model.fit()
    probit_models_res[cls] = probit_res
    print(probit_res.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.418871
         Iterations 9
                          Probit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  207
Model:                         Probit   Df Residuals:                      197
Method:                           MLE   Df Model:                            9
Date:                Wed, 18 Mar 2026   Pseudo R-squ.:                  0.3419
Time:                        22:36:07   Log-Likelihood:                -86.706
converged:                       True   LL-Null:                       -131.76
Covariance Type:            nonrobust   LLR p-value:                 1.551e-15
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -1.2702      0.292     -4.352      0.000      -1.842      -0.698
RI             0.4767      0.379      1.258      0.208      -0.266       1.219
Na             1.8087      1.113      1.626      0.104      -0.372       3.989
Mg             5.3579      2.074      2.583      0.010       1.293       9.423
Al             0.5914      0.768      0.770      0.441      -0.913       2.096
Si             2.4944      1.059      2.355      0.019       0.418       4.570
K              1.7723      1.026      1.727      0.084      -0.239       3.784
Ca             3.6027      2.015      1.788      0.074      -0.347       7.553
Ba             1.4729      0.786      1.873      0.061      -0.068       3.014
```

```
Fe            -0.0648      0.126     -0.513      0.608      -0.313       0.183
==============================================================================

Possibly complete quasi-separation: A fraction 0.18 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.
Optimization terminated successfully.
         Current function value: 0.498118
         Iterations 10
                          Probit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  207
Model:                         Probit   Df Residuals:                      197
Method:                           MLE   Df Model:                            9
Date:                Wed, 18 Mar 2026   Pseudo R-squ.:                  0.2290
Time:                        22:36:07   Log-Likelihood:                -103.11
converged:                       True   LL-Null:                       -133.74
Covariance Type:            nonrobust   LLR p-value:                 7.657e-10
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.7741      0.259     -2.983      0.003      -1.283      -0.265
RI             0.0071      0.361      0.020      0.984      -0.700       0.714
Na            -3.9514      0.996     -3.966      0.000      -5.904      -1.998
Mg            -5.9272      1.807     -3.280      0.001      -9.469      -2.386
Al            -1.6727      0.681     -2.455      0.014      -3.008      -0.337
Si            -3.6322      0.963     -3.770      0.000      -5.520      -1.744
K             -3.1761      0.831     -3.820      0.000      -4.806      -1.546
Ca            -6.1078      1.798     -3.397      0.001      -9.632      -2.584
Ba            -3.2387      0.948     -3.415      0.001      -5.098      -1.380
Fe             0.0201      0.108      0.186      0.853      -0.192       0.233
==============================================================================
Optimization terminated successfully.
         Current function value: 0.132533
         Iterations 10
                          Probit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  207
Model:                         Probit   Df Residuals:                      197
Method:                           MLE   Df Model:                            9
Date:                Wed, 18 Mar 2026   Pseudo R-squ.:                  0.4902
Time:                        22:36:07   Log-Likelihood:                -27.434
converged:                       True   LL-Null:                       -53.813
Covariance Type:            nonrobust   LLR p-value:                 3.249e-08
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -3.5502      0.663     -5.354      0.000      -4.850      -2.251
```

```
RI            -5.4728      1.135     -4.823      0.000      -7.697      -3.249
Na             4.0765      2.302      1.771      0.077      -0.435       8.588
Mg            10.7944      4.213      2.562      0.010       2.538      19.051
Al             1.9400      1.385      1.401      0.161      -0.774       4.654
Si             2.0573      2.069      0.995      0.320      -1.997       6.112
K              2.3948      2.078      1.153      0.249      -1.677       6.467
Ca            11.8758      4.218      2.815      0.005       3.608      20.143
Ba             3.1430      1.519      2.069      0.039       0.165       6.121
Fe             0.1457      0.236      0.617      0.537      -0.317       0.608
==============================================================================

Possibly complete quasi-separation: A fraction 0.29 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.
Optimization terminated successfully.
         Current function value: 0.098604
         Iterations 9
                          Probit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  207
Model:                         Probit   Df Residuals:                      197
Method:                           MLE   Df Model:                            9
Date:                Wed, 18 Mar 2026   Pseudo R-squ.:                  0.5797
Time:                        22:36:07   Log-Likelihood:                -20.411
converged:                       True   LL-Null:                       -48.564
Covariance Type:            nonrobust   LLR p-value:                 6.861e-09
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -2.7295      0.487     -5.610      0.000      -3.683      -1.776
RI            -0.7088      0.855     -0.829      0.407      -2.385       0.968
Na            -2.9892      2.004     -1.491      0.136      -6.918       0.940
Mg            -4.8955      3.719     -1.316      0.188     -12.184       2.393
Al            -0.7210      1.271     -0.567      0.570      -3.212       1.770
Si            -2.6828      1.866     -1.438      0.151      -6.340       0.974
K             -1.4146      1.674     -0.845      0.398      -4.696       1.867
Ca            -3.4967      3.705     -0.944      0.345     -10.759       3.766
Ba            -1.8324      1.268     -1.445      0.149      -4.319       0.654
Fe            -0.2050      0.258     -0.794      0.427      -0.711       0.301
==============================================================================

Possibly complete quasi-separation: A fraction 0.23 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.
Optimization terminated successfully.
         Current function value: 0.049522
         Iterations 12
                          Probit Regression Results
```

```
==============================================================================
Dep. Variable:                      y   No. Observations:                  207
Model:                         Probit   Df Residuals:                      197
Method:                           MLE   Df Model:                            9
Date:                Wed, 18 Mar 2026   Pseudo R-squ.:                  0.8778
Time:                        22:36:07   Log-Likelihood:                -10.251
converged:                       True   LL-Null:                       -83.864
Covariance Type:            nonrobust   LLR p-value:                 3.311e-27
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -4.4504      1.654     -2.690      0.007      -7.693      -1.208
RI             3.3644      1.225      2.746      0.006       0.963       5.766
Na            16.5209      7.724      2.139      0.032       1.383      31.659
Mg            27.0009     12.997      2.077      0.038       1.527      52.474
Al            10.9051      4.801      2.272      0.023       1.496      20.315
Si            16.5801      7.476      2.218      0.027       1.927      31.233
K             12.9426      6.148      2.105      0.035       0.892      24.993
Ca            24.9167     12.234      2.037      0.042       0.938      48.896
Ba            10.0822      4.657      2.165      0.030       0.955      19.209
Fe            -0.7957      1.409     -0.565      0.572      -3.557       1.965
==============================================================================

Possibly complete quasi-separation: A fraction 0.67 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.000000
         Iterations: 35
                           Probit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  207
Model:                         Probit   Df Residuals:                      197
Method:                           MLE   Df Model:                            9
Date:                Wed, 18 Mar 2026   Pseudo R-squ.:                   1.000
Time:                        22:36:07   Log-Likelihood:            -3.6188e-08
converged:                      False   LL-Null:                       -37.021
Covariance Type:            nonrobust   LLR p-value:                 2.442e-12
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        -44.4878   2.24e+04     -0.002      0.998    -4.4e+04    4.39e+04
RI             6.7679    2.6e+04      0.000      1.000    -5.1e+04     5.1e+04
Na           -27.4310   7.55e+04     -0.000      1.000   -1.48e+05    1.48e+05
Mg           -64.3334   1.32e+05     -0.000      1.000   -2.59e+05    2.59e+05
Al           -14.1159   3.42e+04     -0.000      1.000    -6.7e+04     6.7e+04
Si           -21.6549   5.66e+04     -0.000      1.000   -1.11e+05    1.11e+05
K            -68.0987   8.43e+04     -0.001      0.999   -1.65e+05    1.65e+05
```

```
Ca           -64.0550   1.48e+05     -0.000      1.000    -2.9e+05    2.89e+05
Ba           -65.9464   7.52e+04     -0.001      0.999   -1.47e+05    1.47e+05
Fe            -9.8494   1.43e+04     -0.001      0.999    -2.8e+04    2.79e+04
==============================================================================

Complete Separation: The results show that there iscomplete separation or
perfect prediction.
In this case the Maximum Likelihood Estimator does not exist and the parameters
are not identified.

/Users/daragama/Documents/Master25/2_ModelosEstadisticos/modelos_estadisticos/li
b/python3.12/site-packages/statsmodels/discrete/discrete_model.py:227:
PerfectSeparationWarning: Perfect separation or prediction detected, parameter
may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
/Users/daragama/Documents/Master25/2_ModelosEstadisticos/modelos_estadisticos/li
b/python3.12/site-packages/statsmodels/base/model.py:607: ConvergenceWarning:
Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

1.14.1 Comparación

[24]:
```
logit_llf = res_sin_outliers.llf
print(f"Log-verosimilitud Logit: {logit_llf:.4f}")
for cls in classes:
    probit_models_res[cls].llf
    print(f"Log-verosimilitud Probit clase {cls}: {probit_models_res[cls].llf:.
 ↪4f}")
```

```
Log-verosimilitud Logit: -116.0589
Log-verosimilitud Probit clase 0: -86.7063
Log-verosimilitud Probit clase 1: -103.1104
Log-verosimilitud Probit clase 2: -27.4342
Log-verosimilitud Probit clase 3: -20.4111
Log-verosimilitud Probit clase 4: -10.2511
Log-verosimilitud Probit clase 5: -0.0000
```

La comparación directa entre el modelo MNLogit y los modelos probit binarios no es apropiada, ya que el primero es un modelo multinomial conjunto que estima todas las probabilidades simultáneamente, mientras que los segundos son seis modelos binarios independientes que ignoran la estructura multinomial de los datos. Por lo tanto, no es posible comparar sus log-verosimilitudes de manera válida.

2 Ejercicio 2

2.1 Leer datos

3 Base de Datos Scotland

La base de datos contiene información sobre casos de cáncer de labios en regiones de Escocia durante un período específico. Contiene las siguientes variables

- `cases`: Número de casos observados de cáncer de labios en cada región.
- `expected`: Número de casos esperados bajo la hipótesis nula (ajustado por edad y sexo de la población).
- `AFF`: Índice de deprivación relativa o proporción de trabajadores en ocupaciones agrícolas, forestales o de pesca (según la versión).
- `county.names` Nombres de condado de cada dato.

[26]:
```
path = "scotland.csv"
df_scotland = pd.read_csv(path)
print(df_scotland.shape)
df_scotland.head()
```

```
(56, 4)
```

[26]:

	county.names	cases	expected	AFF
0	skye-lochalsh	9	1.4	0.16
1	banff-buchan	39	8.7	0.16
2	caithness	11	3.0	0.10
3	berwickshire	9	2.5	0.24
4	ross-cromarty	15	4.3	0.10

3.1 Ajuste del Modelo

Esta base de datos está proporcionando un valor esperado para cada observación, si se intentara usar la fórmula `cases ~ AFF + expected` el modelo estaría asignando un coeficiente para este valor esperado, pero eso sería incorrecto pues el valor esperado ya no necesita un coeficiente. El modelo `smf.glm` tiene el parámetro offset que fija el coeficiente en 1. Así el modelo quedaría:

$$log(\mu_i) = \beta_0 + \beta_1 \text{AFF}_i + log(\text{expected}_i)$$

[27]:
```
import statsmodels.formula.api as smf

model = smf.glm(
    formula="cases ~ AFF",
    data=df_scotland,
    family=sm.families.Poisson(),
    offset=np.log(df_scotland["expected"])
)

result = model.fit()
```

```
print(result.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                   cases   No. Observations:                   56
Model:                             GLM   Df Residuals:                       54
Model Family:                  Poisson   Df Model:                            1
Link Function:                     Log   Scale:                          1.0000
Method:                           IRLS   Log-Likelihood:                -223.30
Date:                Wed, 18 Mar 2026   Deviance:                       238.62
Time:                        22:36:07   Pearson chi2:                     266.
No. Iterations:                     5   Pseudo R-squ. (CS):             0.9209
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -0.5423      0.070     -7.800      0.000      -0.679      -0.406
AFF            7.3732      0.596     12.380      0.000       6.206       8.541
==============================================================================
```

3.2 A)

¿Regiones con mayor proporción de trabajadores agrícolas tienen una tasa significativamente mayor de cáncer de labios?, ¿es esto consistente con la exposición solar de las regiones?

Dado que el coeficiente de `AFF` es positivo y su p-valor es muy pequeño ($p < 0.001$), hay evidencia de una asociación positiva y estadísticamente significativa entre `AFF` y la tasa de cáncer de labios. Además, el modelo presenta un pseudo-R^2 (Cox-Snell) alto, lo que sugiere buen ajuste relativo.

En términos prácticos, valores más altos de `AFF` (mayor proporción de actividad agrícola) se asocian con tasas más altas de cáncer de labios.

Como `AFF` está vinculado con ocupaciones al aire libre, esta asociación es consistente con la hipótesis de mayor exposición solar, aunque por sí sola no demuestra causalidad.

3.3 B)

Estimar el riesgo relativo regional

[28]:
```
# valores unicos de county.names
print(df_scotland.shape)
df_scotland["county.names"].unique()
```

```
(56, 4)
```

[28]:
```
<StringArray>
['skye-lochalsh',  'banff-buchan',     'caithness',  'berwickshire',
 'ross-cromarty',        'orkney',         'moray',      'shetland',
      'lochaber',        'gordon', 'western.isles',    'sutherland',
```

```
          'nairn',       'wigtown',       'NE.fife',    'kincardine',
        'badenoch',       'ettrick',     'inverness',      'roxburgh',
           'angus',      'aberdeen',   'argyll-bute',    'clydesdale',
       'kirkcaldy',   'dunfermline',     'nithsdale',  'east.lothian',
  'perth-kinross',  'west.lothian',  'cumnock-doon',     'stewartry',
     'midlothian',      'stirling',  'kyle-carrick',    'inverclyde',
    'cunninghame',     'monklands',     'dumbarton',     'clydebank',
        'renfrew',       'falkirk',   'clackmannan',    'motherwell',
      'edinburgh',    'kilmarnock', 'east.kilbride',      'hamilton',
        'glasgow',        'dundee',   'cumbernauld',      'bearsden',
       'eastwood',  'strathkelvin',     'tweeddale',     'annandale']
Length: 56, dtype: str
```

[29]:
```
df_scotland['predicted'] = result.predict()

df_scotland['riesgo_relativo_observado'] = df_scotland['cases'] /␣
 ↪df_scotland['expected']
df_scotland['riesgo_relativo_modelo'] = df_scotland['cases'] /␣
 ↪df_scotland['predicted']

df_sorted = df_scotland.sort_values('county.names')
df_sorted.head()
```

[29]:

	county.names	cases	expected	AFF	predicted	riesgo_relativo_observado	riesgo_relativo_modelo
14	NE.fife	17	7.8	0.07	7.598728	2.179487	2.237216
21	aberdeen	31	22.7	0.16	42.940345	1.365639	0.721932
20	angus	16	10.5	0.07	10.229057	1.523810	1.564171
55	annandale	0	1.8	0.10	2.187677	0.000000	0.000000
22	argyll-bute	11	8.8	0.10	10.695312	1.250000	1.028488

3.4 C)

Para cada región, estimar la probabilidad de que el número de casos predichos sea mayor al 50% de sus casos observados. Estimar la probabilidad de que el número de casos predichos sea mayor al 50% de sus casos esperados

Para cada región `i`:

Tenemos `cases` (observado)

Tenemos `predicted` (predicho por el modelo, que es el parámetro λ de la Poisson)

Buscamos: $P(X > (0.5)(\text{cases}))$ donde $X \sim \text{Poisson}(\lambda = \text{predicted})$

Y de manera similar para la segunda parte

Buscamos: $P(X > (0.5)(\text{predicted}))$ donde $X \sim \text{Poisson}(\lambda = \text{predicted})$

[30]:
```
def prob_casos_mayor_50pc(lambda_pred, casos):
    umbral = 0.5 * casos
    prob = 1 - stats.poisson.cdf(umbral, lambda_pred)
    return prob

# Aplicar a cada región
df_scotland['prob_pred_mayor_50pc_obs'] = df_scotland.apply(
    lambda row: prob_casos_mayor_50pc
(row['predicted'], row['cases']),
    axis=1
)

df_scotland['prob_pred_mayor_50pc_esp'] = df_scotland.apply(
    lambda row: prob_casos_mayor_50pc
(row['predicted'], row['expected']),
    axis=1
)
```

3.5 D)

Hacer el mapa de riesgo espacial de Escocia.

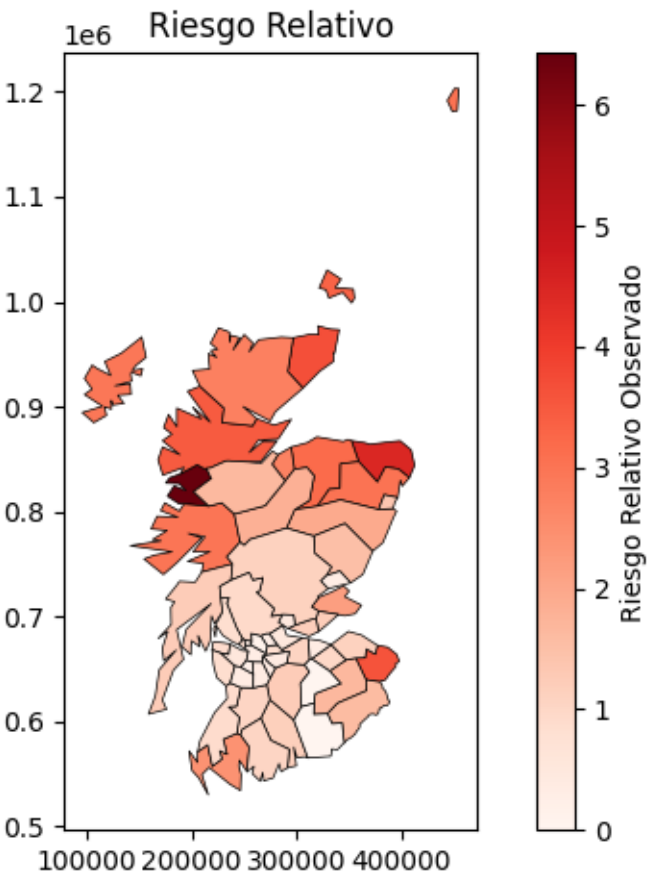
[31]:
```
#!pip install geopandas libpysal
```

[32]:
```
import libpysal
import geopandas as gpd

scotland = libpysal.examples.load_example('Scotlip')
gdf = gpd.read_file(scotland.get_path('scotlip.shp'))
gdf.head()

# Arrreglar nombres para hacer merge
gdf["name_fixed"] = gdf["NAME"].replace(".", "").str.strip().str.lower()
df_scotland["county_fixed"] = (
    df_scotland["county.names"]
    .str.strip()
    .str.lower()
    .str.replace(".", "", regex=False)
)
# orkeny -> okney
df_scotland["county_fixed"] = df_scotland["county_fixed"].replace("orkney",␣
 ↪"okney")
gdf.head()
```

[32]:
```
   CODENO          AREA  PERIMETER  RECORD_ID  DISTRICT           NAME   CODE  \
0    6126  9.740020e+08   184951.0          1         1  Skye-Lochalsh  w6126
1    6016  1.461990e+09   178224.0          2         2   Banff-Buchan  w6016
2    6121  1.753090e+09   179177.0          3         3      Caithness  w6121
3    5601  8.985990e+08   128777.0          4         4   Berwickshire  w5601
4    6125  5.109870e+09   580792.0          5         5  Ross-Cromarty  w6125

   CANCER     POP  CEXP  AFF  \
0       9   28324  1.38   16
1      39  231337  8.66   16
2      11   83190  3.04   10
3       9   51710  2.53   24
4      15  129271  4.26   10

                                              geometry     name_fixed
0  POLYGON ((214091.875 841215.188, 218829 831090…  skye-lochalsh
1  POLYGON ((383866 865862, 398721 867484, 409200…   banff-buchan
2  POLYGON ((311487 968650, 320989 968653, 320254…      caithness
3  POLYGON ((377180 672603, 386871.656 670868.688…   berwickshire
4  POLYGON ((278680.062 882371.812, 294960 887843…  ross-cromarty
```

[33]:
```
print(gdf.shape)
gdf_merged = gdf.merge(df_scotland, how='left', left_on='name_fixed',␣
 ↪right_on='county_fixed')
print(gdf_merged.shape)

gdf_merged.head()


# Mapa de riesgo
ax = gdf_merged.plot(column='riesgo_relativo_observado', cmap='Reds',␣
 ↪legend=True,
        legend_kwds={'label': 'Riesgo Relativo Observado'}, edgecolor='black',␣
 ↪linewidth=0.5)
ax.set_title('Riesgo Relativo')



plt.tight_layout()
plt.show()
```

```
(56, 13)
(56, 23)
```


Riesgo Relativo
1e6
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
100000
200000
300000
400000
6
5
4
3
2
1
0
Riesgo Relativo Observado

[ ]: