INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

Campus Puebla


Materia: Gestión de proyectos de plataformas tecnológicas (Gpo 201)

# Actividad 3 (Regresión Logística)

Gabriel Chávez Moscoso | A00837679

Profesor: Alfredo García Suarez

Heroica Puebla de Zaragoza, Puebla

Octubre 2025

# Tabla de coeficientes

| y depend | x indep | Precisión 1 | Precisión 0 | Exactitud | Sensibilidad 1 | Sensibilidad 0 | puntaje |
|---|---|---|---|---|---|---|---|
| **instant_bookable** | host_response_rate, review_scores_value, room_type | 0.70892018779342 72 | 0.63763297872340 43 | 0.62958648806057 07 | 0.21695402298850 575 | 0.93927522037218 41 | 0.33223322332233 224 |
| **host_is_superhost** | host_response_rate, review_scores_rating, review_scores_cleanliness, review_scores_communication, number_of_reviews | 0.61290322580645 16 | 0.73381294964028 78 | 0.68142108328479 91 | 0.63776223776223 77 | 0.71257485029940 12 | 0.62508567511994 52 |
| **room_type** | price, accommodates, bathrooms, bedrooms | 0.91357273280584 3 | 0.93243243243243 25 | 0.91438555620267 91 | 0.99667994687915 01 | 0.32701421800947 866 | 0.95331851381390 92 |
| **host_response_time** | host_acceptance_rate, host_is_superhost, review_scores_communication | 0.92587412587412 59 | 0.26846307385229 54 | 0.54222481071636 57 | 0.47455197132616 49 | 0.83540372670807 45 | 0.62748815165876 78 |
| **availability_365** | price, minimum_nights, review_scores_rating, reviews_per_month | 0.76621923937360 18 | 0.32320777642770 354 | 0.55387303436225 98 | 0.55152979066022 54 | 0.56 | 0.64138576779026 21 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **property_type** | price, bedrooms, bathrooms, review_scores_rating, review_scores_location | 0.13692480359147025 | 0.9370460048426115 | 0.52184041933605513 | 0.70114942528735564 | 0.50162022034996676 | 0.22910798122065673 |
| **host_response_rate** | review_scores_communication, review_scores_accuracy, number_of_reviews, host_is_superhost | 0.99221357063403778 | 0.06845965770171149 | 0.55212580081537557 | 0.53929866989117729 | 0.88888888888888888 | 0.69878574226400031 |
| **host_acceptance_rate** | review_scores_communication, host_response_rate, number_of_reviews, review_scores_rating | 0.65680056377730 8 | 0.38255033557046 98 | 0.60920209668025 63 | 0.83512544802867 38 | 0.18968386023294 51 | 0.73530571992110 45 |
| **price** | room_type_encoded, accommodates, bathrooms, bedrooms, review_scores_rating | 0.86099865047233 47 | 0.67008196721311 47 | 0.75247524752475 25 | 0.66458333333333 33 | 0.86393659180977 54 | 0.75014697236919 46 |
| **review_scores_rating** | room_type, accommo | 0.98141263940520 45 | 0.04065934065934 066 | 0.48281887012230 634 | 0.47567567567567 57 | 0.71153846153846 16 | 0.64077669902912 63 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | dates, bathrooms, bedrooms, price | | | | | | |

# Reporte comparativo

---

**1) Y: instant_bookable**

No hubo proceso de dicotomización, ya era variable binaria del dataset

X: host_response_rate, review_scores_value, room_type
Por qué esas X: miden agilidad operativa del anfitrión (response), confianza/valor percibido (reviews) y perfil de riesgo del producto (room_type), factores plausibles para permitir reservas instantáneas.
Desempeño: Modelo conservador ya que acierta cuando predice "sí", pero recupera pocos positivos.

```
Matriz de Confusión:
[[959  62]
 [545 151]]
```

Con instant_bookable, la matriz muestra un modelo conservador: identifica muy bien los no instantáneos, pero omiten muchos "1" (sensibilidad ≈0.22), aunque cuando predice "1" suele acertar (precisión ≈0.71); la exactitud global ronda 0.63. En términos de negocio en Airbnb, reduce el riesgo de activar Instant Book indebidamente, pero pierde oportunidades; si se busca captar más listados aptos, conviene bajar el umbral o re-equilibrar clases.

2) Y: host_is_superhost

No hubo proceso de dicotomización, ya era variable binaria del dataset

X: host_response_rate, review_scores_rating, review_scores_cleanliness, review_scores_communication, number_of_reviews
Por qué esas X: reflejan criterios de calidad/experiencia relacionados con el estatus Superhost (alta respuesta, buenas calificaciones, volumen de reseñas).
Desempeño: Balanceado ya que capta razonablemente a los Superhost sin exceso de falsos positivos.

```
Matriz de Confusión:
[[714 288]
 [259 456]]
```

Para host_is_superhost, la matriz  indica un modelo balanceado: sensibilidad (clase 1) ≈0.64, con precisión de la clase 1 ≈0.61 y exactitud ≈0.68. En términos prácticos para Airbnb, reconoce razonablemente a los Superhost sin generar demasiados falsos positivos; si la prioridad fuera captar aún más Superhost, podría ajustarse el umbral para ganar recall a costa de algo de precisión.

**3) Y: room_type**

X: price, accommodates, bathrooms, bedrooms
Por qué esas X: las características y capacidad del anuncio y su precio son fuertes señales estructurales del tipo de cuarto.
Desempeño: Muy alto ya que separa con claridad el tipo, aunque la sensibilidad 0 baja (0.33) sugiere sesgo hacia la clase positiva.

```
Matriz de Confusión:
[[  69  142]
 [   5 1501]]
```

Para room_type, el modelo es excelente para detectar *Entire home/apt*, pero confunde con frecuencia *Private/Shared* como *Entire*; si se requiere menor tasa de falsos positivos, conviene subir el umbral o re-equilibrar clases para ganar especificidad.

## Convertir room_type en variable dicotómica

Vamos a convertir room_type en dos categorías:

- "Entire home/apt" = 1
- Otros tipos (Private room, Shared room) = 0

```
# Primero veamos los valores únicos actuales
print("Valores únicos en room_type:", np.unique(df['room_type']))

# Convertimos room_type a variable dicotómica
df['room_type_binary'] = (df['room_type'] == 'Entire home/apt').astype(int)

# Verificamos la conversión
print("\nValores únicos en room_type_binary:", np.unique(df['room_type_binary']))
print("\nDistribución de room_type_binary:")
print(df['room_type_binary'].value_counts())
```

```
Valores únicos en room_type: ['Entire home/apt' 'Hotel room' 'Private room' 'Shared room']

Valores únicos en room_type_binary: [0 1]

Distribución de room_type_binary:
room_type_binary
1    4992
0     731
Name: count, dtype: int64
```

**4) Y: host_response_time**

X: host_acceptance_rate, host_is_superhost, review_scores_communication
Por qué esas X: aceptación, estatus y comunicación se asocian con prontitud en responder.
Desempeño: Preciso pero conservador ya que identifica positivos con mucha precisión, aunque se le escapan muchos (recall moderado).

```
Matriz de Confusión:
[[269  53]
 [733 662]]
```

Para host_response_time, la matriz refleja un modelo muy preciso cuando predice respuesta rápida. En Airbnb, discrimina bien a los anfitriones lentos y evita falsos positivos de "rápido", pero no recupera a la mitad de los que sí responden en una hora; si se busca mayor cobertura de rápidos, conviene bajar el umbral o re‑equilibrar clases.

## Convertir host_response_time en variable dicotómica

Vamos a convertir host_response_time en dos categorías:

- Respuesta rápida (within an hour) = 1
- Respuesta más lenta (más de una hora) = 0

```
# Primero veamos los valores únicos actuales
print("Valores únicos en host_response_time:", np.unique(df['host_response_time']))

# Convertimos host_response_time a variable dicotómica
df['response_time_binary'] = (df['host_response_time'] == 'within an hour').astype(int)

# Verificamos la conversión
print("\nValores únicos en response_time_binary:", np.unique(df['response_time_binary']))
print("\nDistribución de response_time_binary:")
print(df['response_time_binary'].value_counts())
```

```
Valores únicos en host_response_time: ['a few days or more' 'within a day' 'within a few hours' 'within an hour']

Valores únicos en response_time_binary: [0 1]

Distribución de response_time_binary:
response_time_binary
1    4651
0    1072
Name: count, dtype: int64
```

**5) Y: availability_365**

X: price, minimum_nights, review_scores_rating, reviews_per_month
Por qué esas X: la estrategia de precios y mínimos, la demanda (reviews/mes) y la calidad influyen en la disponibilidad anual.
Desempeño: Mejora el equilibrio frente a otros, aunque el poder explicativo global es medio.

```
Matriz de Confusión:
[[266 209]
 [557 685]]
```

Para availability_365, la matriz indica que cuando marca “alta disponibilidad” suele acertar, pero recupera sólo un poco más de la mitad de los casos positivos y deja muchos fuera. En términos de negocio, el modelo sirve para confirmar listados muy disponibles, pero no para detectarlos exhaustivamente; convendría ajustar umbral, si se busca mayor cobertura de positivos.

## Convertir availability_365 en variable dicotómica

Vamos a convertir availability_365 en dos categorías:

- Alta disponibilidad (>180 días) = 1
- Baja disponibilidad (≤180 días) = 0

```
# Primero veamos los valores únicos actuales y la distribución
print("Estadísticas de availability_365:")
print(df['availability_365'].describe())

# Convertimos availability_365 a variable dicotómica
df['availability_binary'] = (df['availability_365'] > 180).astype(int)

# Verificamos la conversión
print("\nValores únicos en availability_binary:", np.unique(df['availability_binary']))
print("\nDistribución de availability_binary:")
print(df['availability_binary'].value_counts())
```

```
Estadísticas de availability_365:
count    5723.000000
mean      253.241482
std       118.140893
min         0.000000
25%       170.000000
50%       299.000000
75%       355.000000
max       365.000000
Name: availability_365, dtype: float64

Valores únicos en availability_binary: [0 1]

Distribución de availability_binary:
availability_binary
1    4138
0    1585
Name: count, dtype: int64
```

**6) Y: property_type**

X: price, bedrooms, bathrooms, review_scores_rating, review_scores_location
Por qué esas X: el tamaño/amenidades, el precio y la ubicación percibida suelen diferenciar el tipo de propiedad.
Desempeño: Débil ya que hay alta recuperación de positivos con mucha imprecisión (muchos falsos positivos).

```
Matriz de Confusión:
[[774 769]
 [ 52 122]]
```

Para property_type, la matriz recupera muchos positivos pero con demasiados falsos positivos, por lo que confunde con frecuencia propiedades que no son apartment/condo como si lo fueran. Para Airbnb, este modelo no es confiable para clasificar el tipo de propiedad; conviene elevar el umbral, añadir rasgos estructurales o replantear la dicotomización.

## Convertir property_type en variable dicotómica

Vamos a convertir property_type en dos categorías:

- "Apartment/Condo" (Apartamentos y Condominios) = 1
- Otros tipos de propiedades = 0

```
# Primero veamos los valores únicos actuales
print("Valores únicos en property_type:")
print(df['property_type'].value_counts())

# Convertimos property_type a variable dicotómica
df['property_type_binary'] = df['property_type'].str.contains('Apartment|Condo|Condominium|Loft', case=False).astype(int)

# Verificamos la conversión
print("\nValores únicos en property_type_binary:", np.unique(df['property_type_binary']))
print("\nDistribución de property_type_binary:")
print(df['property_type_binary'].value_counts(dropna=False))
```

```
Valores únicos en property_type:
property_type
Entire rental unit                  2273
Entire home                         1655
Private room in home                 453
Entire condo                         359
Entire townhouse                     235
Entire guesthouse                    162
Room in hotel                        160
Private room in rental unit           59
Entire loft                           56
Entire serviced apartment             46
Entire guest suite                    38
Private room in townhouse             35
Shared room in home                   27
Entire bungalow                       23
Tiny home                             23
Private room in condo                 21
Room in boutique hotel                19
Entire villa                          15
Private room in serviced apartment    10
Camper/RV                              9
```

**7) Y: host_response_rate**

X: review_scores_communication, review_scores_accuracy, number_of_reviews, host_is_superhost
Por qué esas X: la comunicación/precisión en el anuncio y la experiencia/estatus están ligadas a responder consistentemente.
Desempeño: Muy preciso para positivos, con recall moderado.

```
Matriz de Confusión:
[[ 56   7]
 [762 892]]
```

Para host_response_rate, en términos prácticos, casi no comete falsos positivos, pero omite ~46% de los hosts realmente de alta respuesta. Si se busca mayor cobertura de positivos, conviene bajar el umbral o re-equilibrar clases y en Airbnb, esto significa que cuando clasifica a un host como de alta respuesta casi siempre acierta, pero omite cerca de la mitad de los que realmente lo son.

## Convertir host_response_rate en variable dicotómica

Vamos a convertir host_response_rate en dos categorías:

- Alta tasa de respuesta (≥90%) = 1
- Baja tasa de respuesta (<90%) = 0

```python
# Primero veamos los valores únicos actuales y su distribución
print("Distribución de host_response_rate original:")
print(df['host_response_rate'].value_counts().sort_index())

# Convertimos host_response_rate a numérico
# Primero reemplazamos los valores nulos con '0%'
df['host_response_rate_clean'] = df['host_response_rate'].fillna('0%')
# Convertimos a float (asumiendo que los valores ya están en decimal)
df['host_response_rate_clean'] = df['host_response_rate_clean'].apply(lambda x: float(str(x)))

# Analizamos la distribución de los valores numéricos
print("\nEstadísticas de host_response_rate_clean:")
print(df['host_response_rate_clean'].describe())

# Convertimos a binario (1 para ≥98%, 0 para <98%)
df['response_rate_binary'] = (df['host_response_rate_clean'] >= 0.98).astype(int)

# Verificamos la conversión
print("\nValores únicos en response_rate_binary:", np.unique(df['response_rate_binary']))
print("\nDistribución de response_rate_binary:")
print(df['response_rate_binary'].value_counts())
```

```
Distribución de host_response_rate original:
host_response_rate
0.95      86
0.96     139
0.97      62
0.98     176
0.99     273
1.00    4987
Name: count, dtype: int64

Estadísticas de host_response_rate_clean:
count    5723.000000
mean        0.996860
```

**8) Y: host_acceptance_rate**

X: review_scores_communication, host_response_rate, number_of_reviews, review_scores_rating
Por qué esas X: la responsividad, la experiencia y la calidad percibida suelen aumentar la probabilidad de aceptar solicitudes.
Desempeño: Fuerte en recall ya que detecta bien tasas altas de aceptación, con precisión aceptable.

```
Matriz de Confusión:
[[114 487]
 [184 932]]
```

Para host_acceptance_rate, en términos prácticos, el modelo captura a la mayoría de los hosts con alta aceptación, pero sobremarca casos que no lo son; si se prioriza precisión, conviene elevar el umbral o penalizar positivos erróneos, y si se prioriza cobertura, mantener el corte en 85% es coherente.

## Convertir host_acceptance_rate en variable dicotómica

Vamos a convertir host_acceptance_rate en dos categorías:

- Alta tasa de aceptación (≥85%) = 1
- Baja tasa de aceptación (<85%) = 0

```python
# Primero veamos los valores únicos actuales y su distribución
print("Distribución de host_acceptance_rate original:")
print(df['host_acceptance_rate'].value_counts().sort_index())

# Convertimos host_acceptance_rate a numérico
# Primero reemplazamos los valores nulos con '0%'
df['host_acceptance_rate_clean'] = df['host_acceptance_rate'].fillna('0%')
# Convertimos a float
df['host_acceptance_rate_clean'] = df['host_acceptance_rate_clean'].apply(lambda x: float(str(x).rstrip('%')) / 100)

# Analizamos la distribución de los valores numéricos
print("\nEstadísticas de host_acceptance_rate_clean:")
print(df['host_acceptance_rate_clean'].describe())

# Mostramos los percentiles para ayudar a elegir un mejor umbral
print("\nPercentiles de host_acceptance_rate_clean:")
percentiles = [25, 50, 75]
for p in percentiles:
    print(f"Percentil {p}: {df['host_acceptance_rate_clean'].quantile(p/100):.2f}")

# Convertimos a binario usando el percentil 50 como umbral
umbral = df['host_acceptance_rate_clean'].median()
print(f"\nUsando umbral de {umbral:.2f} (mediana)")
df['acceptance_rate_binary'] = (df['host_acceptance_rate_clean'] >= umbral).astype(int)

# Verificamos la conversión
print("\nValores únicos en acceptance_rate_binary:", np.unique(df['acceptance_rate_binary']))
print("\nDistribución de acceptance_rate_binary:")
print(df['acceptance_rate_binary'].value_counts())
```

```
Distribución de host_acceptance_rate original:
host_acceptance_rate
0.70      16
0.71       6
0.72       3
```

**9) Y: price**

X: room_type_encoded, accommodates, bathrooms, bedrooms, review_scores_rating

Por qué esas X: el tipo, la capacidad y las amenidades determinan el nivel de precio; la reputación puede ajustar el valor.

Desempeño: Sólido y equilibrado ya que tiene buen poder para distinguir rangos altos.

```
Matriz de Confusión:
[[654 103]
 [322 638]]
```

Para price la matriz indica en palabras simples que el modelo identifica bien los listados caros con pocos falsos positivos, aunque deja fuera ~1/3 de los realmente caros. Para Airbnb, es útil para segmentar premium sin sobre-marcar, y si se busca capturar más “precios altos” conviene bajar el umbral o enriquecer variables.

## Convertir price en variable dicotómica

Vamos a convertir price en dos categorías usando la mediana como punto de corte:

- Precio alto (≥ mediana) = 1
- Precio bajo (< mediana) = 0

```python
# Primero veamos la distribución actual y estadísticas de price
print("Estadísticas de price:")
print(df['price'].describe())

# Mostramos los percentiles para entender mejor la distribución
print("\nPercentiles de price:")
percentiles = [25, 50, 75]
for p in percentiles:
    print(f"Percentil {p}: ${df['price'].quantile(p/100):.2f}")

# Usamos la mediana como punto de corte
umbral = df['price'].median()
print(f"\nUsando umbral de ${umbral:.2f} (mediana)")

# Convertimos a binario
df['price_binary'] = (df['price'] >= umbral).astype(int)

# Verificamos la conversión
print("\nValores únicos en price_binary:", np.unique(df['price_binary']))
print("\nDistribución de price_binary:")
print(df['price_binary'].value_counts())
```

```
Estadísticas de price:
count    5723.000000
mean      123.189411
std        66.039277
min        11.000000
25%        80.000000
50%       105.000000
75%       147.000000
max       382.000000
Name: price, dtype: float64

Percentiles de price:
Percentil 25: $80.00
```

**10) Y: review_scores_rating**

X: room_type, accommodates, bathrooms, bedrooms, price
Por qué esas X: el confort/espacio y el precio condicionan la satisfacción agregada reflejada en la calificación global.
Desempeño: Muy preciso cuando predice alto rating, con recall moderado y exactitud global baja.

```
Matriz de Confusión:
[[ 37  15]
 [873 792]]
```

Para review_scores_rating, el modelo sirve para confirmar listados con calificación alta (casi sin falsos positivos), pero omite muchos que realmente la tienen; si se busca mayor cobertura de "≥4.5", conviene bajar el umbral de decisión o re-equilibrar clases.

# Convertir review_scores_rating en variable dicotómica

Vamos a convertir la variable review_scores_rating en una variable dicotómica usando un umbral de 4.5:

- 1 = Calificación alta (≥ 4.5)
- 0 = Calificación baja (< 4.5)

```python
# Primero veamos la distribución actual y estadísticas
print("Estadísticas de review_scores_rating:")
print(df['review_scores_rating'].describe())

# Mostramos los percentiles para entender mejor la distribución
print("\nPercentiles de review_scores_rating:")
percentiles = [25, 50, 75, 90]
for p in percentiles:
    print(f"Percentil {p}: {df['review_scores_rating'].quantile(p/100):.2f}")

# Convertimos a binario usando 4.5 como umbral (calificaciones de 4.5 o más son consideradas altas)
umbral = 4.5
print(f"\nUsando umbral de {umbral:.1f}")

# Convertimos a binario
df['rating_binary'] = (df['review_scores_rating'] >= umbral).astype(int)

# Verificamos la conversión
print("\nValores únicos en rating_binary:", np.unique(df['rating_binary']))
print("\nDistribución de rating_binary:")
print(df['rating_binary'].value_counts())
```

```
Estadísticas de review_scores_rating:
count    5723.000000
mean        4.816663
std         0.156250
min         4.300000
25%         4.700000
50%         4.820000
75%         4.970000
max         5.000000
Name: review_scores_rating, dtype: float64

Percentiles de review_scores_rating:
Percentil 25: 4.70
```

---

# Comparativo

**Ranking por desempeño global (puntaje/F1):**

1. **room_type 0.95** — separación casi perfecta del *Entire home/apt*.
2. **price 0.75** — sólido y equilibrado para distinguir precios altos.
3. **host_acceptance_rate 0.74** — gran *recall* (detecta la mayoría de tasas altas).
4. **host_response_rate 0.70** — altísima precisión cuando predice tasa alta.
   5–6) **availability_365 0.64**, **review_scores_rating 0.64** — útiles para "confirmar" positivos, cobertura media.
5. **host_response_time 0.63** — preciso pero conservador.
6. **host_is_superhost 0.63** — balanceado, sin exceso de falsos positivos.
7. **instant_bookable 0.33** — conservador; omite muchos casos positivos.
8. **property_type 0.23** — débil; muchos falsos positivos.
   Actividad 3 (Regresión Logístic…

**Lecturas clave por perfil de error:**

- **Conservadores (alta precisión, bajo *recall*):** *instant_bookable*, *host_response_time*, *review_scores_rating*, *host_response_rate*.
- **De alta cobertura (alto *recall*):** *host_acceptance_rate* y, por construcción del set, *room_type*.
- **Con mejor exactitud global:** *room_type* (**0.91**) y *price* (**0.75**).
- **A mejorar:** *property_type* (F1 **0.23**) requiere más rasgos estructurales (m², amenities, año/edificio) o redefinir su dicotomía.

# Conclusión

En conjunto, los resultados muestran que los atributos estructurales del anuncio explican mejor los desenlaces que dependen de la configuración del producto, con room_type destacando por su separación casi perfecta y price mostrando un desempeño sólido y equilibrado para distinguir niveles altos, mientras que los desenlaces ligados al comportamiento del anfitrión dependen más del umbral de decisión y del balance de clases, con host_acceptance_rate fuerte en recall pero con baja especificidad, host_response_rate muy preciso cuando marca tasas altas, host_response_time e

instant_bookable conservadores y review_scores_rating útil para confirmar calificaciones altas pero con cobertura moderada; property_type, en cambio, es débil y requiere señales adicionales

Para Airbnb se recomienda adoptar umbrales adaptativos por objetivo y mercado, bajando el umbral y reponderando clases en instant_bookable y host_response_time cuando la prioridad sea captar más casos positivos en listados con señales fuertes como Entire home o apt, rating igual o superior a 4.5 y host_response_rate igual o superior a 90, elevando el umbral en host_acceptance_rate cuando la prioridad sea reducir falsos positivos, enriqueciendo la señal donde el modelo es débil con metros cuadrados, amenities y características del edificio para property_type y con ubicación fina y estacionalidad para price, calibrando probabilidades antes de decidir y validando todo mediante pruebas A/B por ciudad y temporada con métricas como PR-AUC, conversión, cancelaciones y satisfacción del huésped para escalar después los umbrales segmentados que demuestren mejora