



**JUST DELIVERY**  
FOOD IS ON ITS WAY!

# Just Delivery

Jorge Gómez Galván

**LinkedIn:** [linkedin.com/in/jorgeggalvan](https://linkedin.com/in/jorgeggalvan)  
**E-mail:** [ggalvanjorge@gmail.com](mailto:ggalvanjorge@gmail.com)

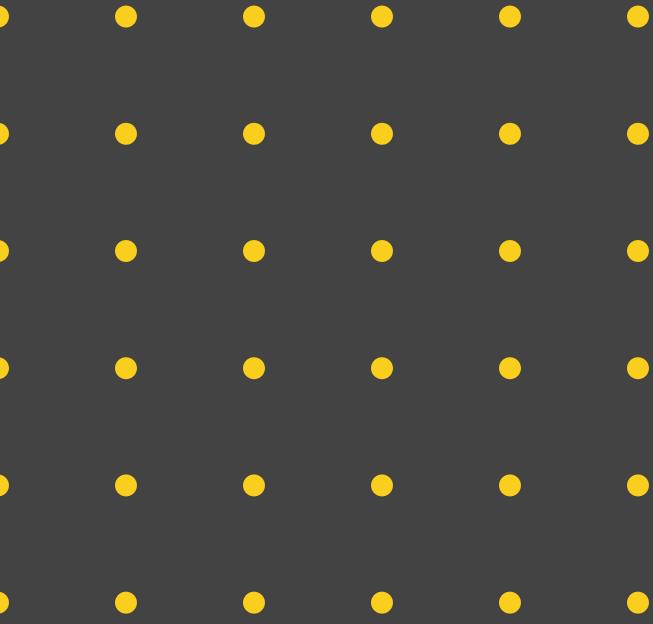




## OBJETIVO

Predecir el número de **pedidos** mensuales de una ciudad para poder ayudar al departamento comercial a elegir por qué ciudades apostar en el próximo mes.

# 01



## Tratamiento previo del dataset



Para poder trabajar con el dataset se han tenido que hacer una serie de modificaciones previas.

Primero, se han renombrado las variables para que sea más cómodo trabajar con él, y después  
se han añadido varias columnas con nuevas variables.

## Creación de nuevas variables para:

**Dimensionar el potencial y alcance por restaurante**

#Pedidos / restaurante  
Población / restaurante



**Dimensionar presencia en una ciudad**

Ratio de restaurantes fidelizados



**Encontrar patrones geográficos**

Ubicación de la ciudad





# Renombre de variables

Nombre de variable original	Nuevo nombre de variable
city	ciudad
population	poblacion
orders	pedidos
density	densidad
avg_income	ingreso_medio
total_restaurants	total_rest
total_restaurants_onboarded	total_rest_fidelizados
self_delivery_restaurants	n_rest_delivery
self_delivery_restaurants_onboarded	n_rest_fidelizados_delivery
non_self_delivery_restaurants	n_rest_no_delivery
non_self_delivery_restaurants_onboarded	n_rest_fidelizados_no_delivery



# Creación de nuevas variables

1

## Número de pedidos por restaurante fidelizado

Para establecer el nivel de hábito que tienen en esa ciudad de pedir delivery

2

## Población por restaurantes

Para conocer si es una ciudad en la que las necesidades de pedidos a domicilio están bien cubiertas o no

3

## Ratio de restaurantes fidelizados

Para conocer la cuota de mercado y grado de penetración entre los restaurantes de la ciudad

4

## Coordenadas y ciudades autónomas

Para conocer si existen patrones regionales o geográficos



# Creación de nuevas variables. Latitud y longitud

- 1 Utilizando las librerías GeocoderTimedout y Nominatim, se ha generado con un bucle una nueva tabla que obtenga la latitud y longitud de las ciudades del dataset original.

## Coordenadas

```
[4]: # Geolocalizador
geolocator = Nominatim(user_agent="myGeocoder", timeout=99999)

ciudades = df['ciudad'].unique() # Nombres de ciudades únicas
coordenadas = {} # Diccionario vacío para almacenar las coordenadas de cada ciudad

# Bucle para obtener la latitud y longitud de las ciudades
for ciudad in ciudades:
    try:
        # Ubicación geográfica de la ciudad utilizando el método 'geocode'
        localizacion = geolocator.geocode(ciudad)
        # Latitud y longitud si se encuentra la ubicación
        coordenadas[ciudad] = (localizacion.latitude, localizacion.longitude)
    except AttributeError:
        # 'NA' en lugar de las coordenadas si no se encuentra la ubicación
        coordenadas[ciudad] = "NA"

# Dataframe con las ciudades y sus coordenadas
df_ciudades = pd.DataFrame(coordenadas.items(), columns=['ciudad', 'coordenadas'])
```

## ¿Por qué se necesitan las coordenadas?

Se quiere comprobar si existe algún patrón de comportamiento entre ciudades más al norte o más al sur.

Se observa que, con esta nueva variable, **el modelo mejora**.

- 2

Se han dividido las coordenadas en latitud y longitud para pasarlas después a formato *float* y realizar un **merge** con el dataset original.

```
[7]: # División de coordenadas en latitud y longitud
df_ciudades[['latitud', 'longitud']] = pd.DataFrame(df_ciudades['coordenadas'].tolist(), index=df_ciudades.index)
# Eliminación de la columna 'coordenadas'
df_ciudades.drop(['coordenadas'], axis=1, inplace=True)

# Conversión a tipo float
df_ciudades['latitud'] = df_ciudades['latitud'].astype(float)
df_ciudades['longitud'] = df_ciudades['longitud'].astype(float)

# Unión de latitud y longitud con el dataframe original
df = df.merge(df_ciudades, how='left', left_on='ciudad', right_on='ciudad')
```

	zados_no_delivery	n_pedidos_por_rest	poblacion_por_rest	poblacion_por_rest_fidelizado	p_rest_fidelizados	latitud	longitud
18	77.64	337.40	2010.02	0.17	40.56	-3.89	
0	10.56	220.14	5806.12	0.04	36.78	-4.10	
0	37.75	341.42	22619.25	0.02	37.67	-1.70	
0	34.07	205.14	6417.86	0.03	36.76	-2.61	
0	49.11	230.35	4679.84	0.05	38.69	0.14	
...	...	...	...	...	...	...	



# Creación de nuevas variables. Comunidad Autónoma

- 3 Utilizando esta función, se ha conseguido obtener la Comunidad Autónoma en base al nombre de la ciudad, distinguiendo entre *state* y *state\_district*. Esto se ha realizado directamente con las coordenadas de las ciudades añadidas.

## Comunidades autónomas

```
[8]: # Función que devuelve el nombre de la comunidad autónoma en función del nombre de la ciudad
def get_comunidad_autonoma(ciudad):

    # Geolocalizador
    geolocator = Nominatim(user_agent="myGeocoder", timeout=99999)

    try:
        # Ubicación geográfica de la ciudad
        location = geolocator.geocode(ciudad+", Spain", exactly_one=True, addressdetails=True, language="es")
        address = location.raw['address']

        # Comprobación de que la ubicación incluya el nombre de la comunidad autónoma en la clave 'state'
        if 'state' in address:
            return address['state']
        # Comprobación de que la ubicación incluya el nombre de la comunidad autónoma en la clave 'state_district'
        elif 'state_district' in address:
            return address['state_district']
        # Devolución de 'NA' si no se encuentra en ninguna de las claves anteriores
        else:
            return 'NA'
        # Devolución de 'NA' si se produce un error en la solicitud de geolocalización
    except (AttributeError, GeocoderTimedOut):
        return 'NA'

    # Aplicación de la función 'get_comunidad_autonoma'
    df['comunidad_autonoma'] = df['ciudad'].apply(lambda x: get_comunidad_autonoma(x))
```

## ¿Por qué necesitamos las Com. Autónomas?

Al igual que con las coordenadas, se quiere comprobar si existen patrones de comportamiento dentro de las mismas Comunidades Autónomas.

Sin embargo, con esta variable el modelo **empeora**.

- 4

- Como hay algunas comunidades que no se han logrado obtener, se han añadido manualmente:

```
[9]: # Ciudades que no se han podido obtener su comunidad autónoma
cond_na = df['comunidad_autonoma'] == 'NA'
cond_melilla = df['comunidad_autonoma'] == 'Melilla'
cond_ceuta = df['comunidad_autonoma'] == 'Plazas de Soberanía'

df[cond_na | cond_melilla | cond_ceuta][['ciudad','comunidad_autonoma']]
```

	ciudad	comunidad_autonoma
23	Melilla	Melilla
24	Ceuta	Plazas de Soberanía
204	Almuñécar	NA
393	Gádor	NA
476	Llíria d'Amunt	NA
568	Sada	NA
589	Egurés	NA
642	Sabiñánigo	NA
684	Fernán-Núñez	NA
689	Griñón	NA
808	Peníscola/Peñíscola	NA
980	Llíria de Vall	NA

```
[10]: # Actualización manual de las comunidades autónomas para las ciudades que no se pudieron obtener automáticamente
df.loc[23,'comunidad_autonoma'] = 'Ceuta y Melilla'
df.loc[24,'comunidad_autonoma'] = 'Ceuta y Melilla'
df.loc[204,'comunidad_autonoma'] = 'Andalucía'
df.loc[393,'comunidad_autonoma'] = 'Santa Cruz de Tenerife'
df.loc[476,'comunidad_autonoma'] = 'Cataluña'
df.loc[568,'comunidad_autonoma'] = 'Galicia'
df.loc[589,'comunidad_autonoma'] = 'Navarra'
df.loc[642,'comunidad_autonoma'] = 'Aragón'
df.loc[684,'comunidad_autonoma'] = 'Andalucía'
df.loc[689,'comunidad_autonoma'] = 'Comunidad de Madrid'
df.loc[808,'comunidad_autonoma'] = 'Comunidad Valenciana'
df.loc[980,'comunidad_autonoma'] = 'Cataluña'
```

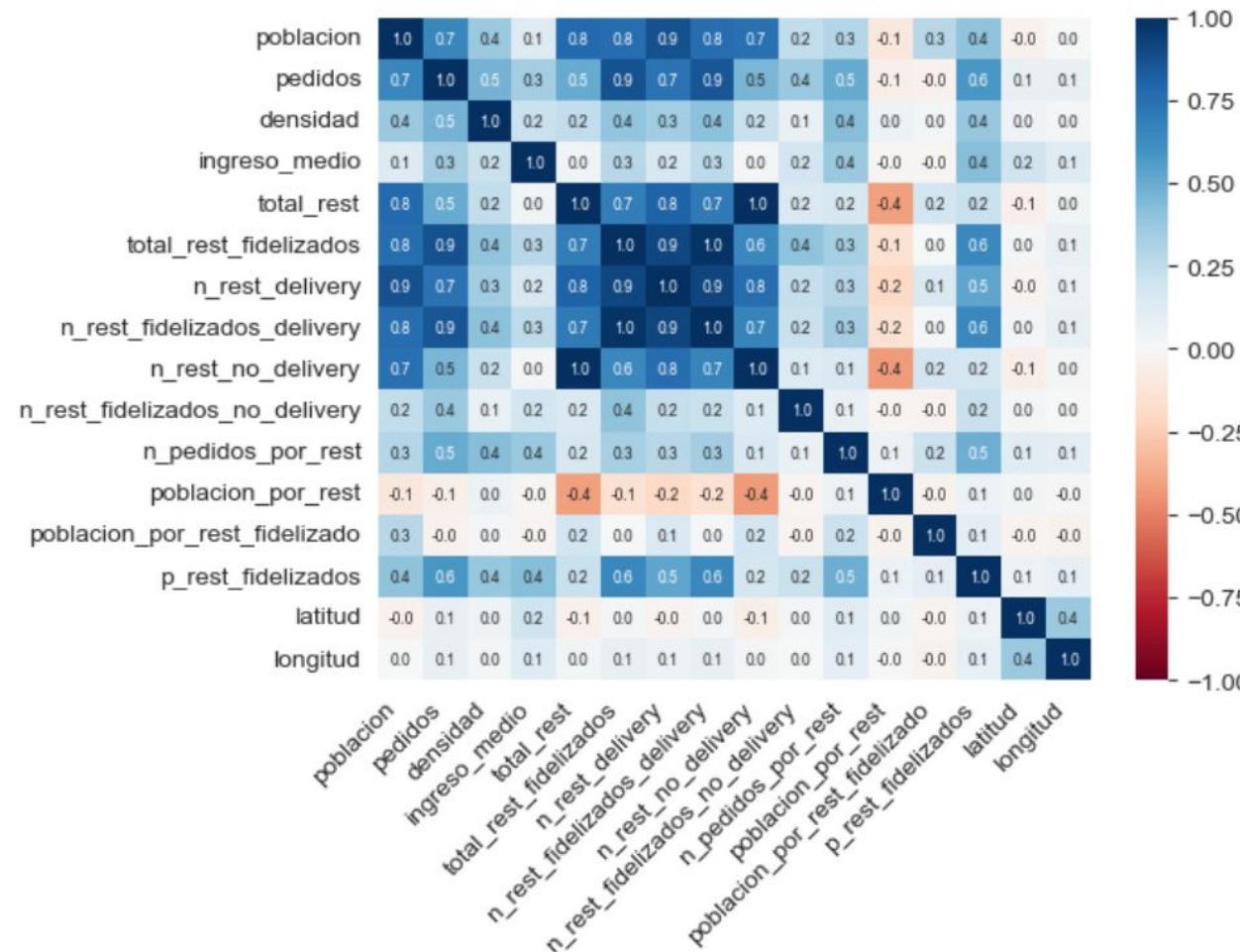


# Análisis exploratorio de datos

Una vez se han incluido todas las nuevas variables en el DataFrame, se ha calculado la matriz de correlación para analizar las relaciones lineales y detectar posibles problemas de multicolinealidad entre las diferentes variables.

```
[13]: # Matriz de correlación
numerical = df.select_dtypes(exclude = 'object') # Selección de variables numéricas
corr_matrix = numerical.corr()

# Mapa de calor con la matriz de correlación
sns.heatmap(corr_matrix, vmin=-1, cmap='RdBu', annot=True, annot_kws={"size": 6}, fmt='.1f')
plt.xticks(rotation=45, ha='right')
plt.show()
```



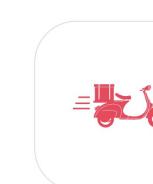
## Insight 1

Correlación positiva muy alta entre la población y el número de restaurantes, tanto totales como en sus diferentes tipos.



## Insight 2

El número de pedidos está muy relacionado con el número de restaurantes fidelizados, especialmente si tienen delivery.

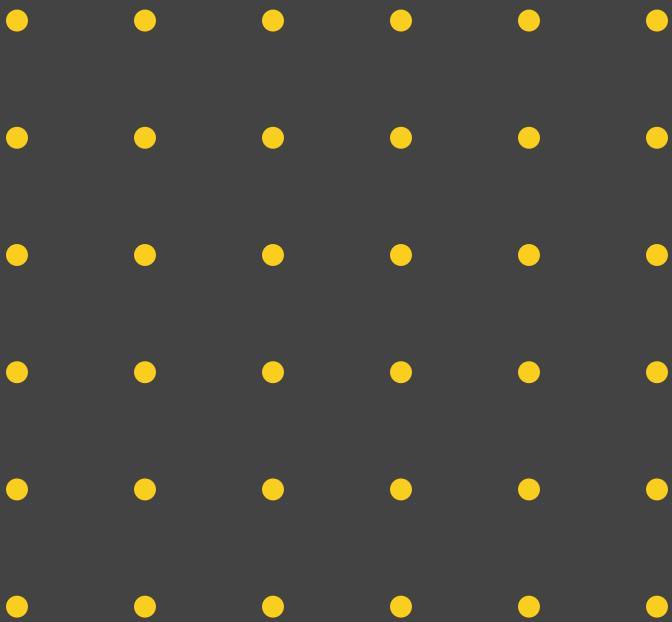


## Insight 3

El número de pedidos se relaciona positivamente con la población total, pero no tanto con la densidad de población. Densidad y población por restaurante son variables poco determinantes.



# 02



## Modelo de regresión



# Elección del modelo de regresión

El objetivo del modelo de regresión en este contexto es **predecir el número de pedidos mensuales** que se pueden esperar en una ciudad en función de variables explicativas.

De esta manera, el modelo permitiría estimar el **potencial de una ciudad** en cuanto a la demanda de comida a domicilio y ayudaría tanto en la toma de decisiones comerciales de Just Delivery, como en la priorización de ciudades y la asignación eficiente de recursos.

Después de entrenar y ajustar varios modelos de regresión utilizando conjuntos de datos de entrenamiento y de prueba con diferentes variables, se ha utilizado **LazyRegressor** para comparar los modelos y comprobar cuál responde mejor.

**GradientBoosting** ha demostrado ser el modelo de regresión con mejores resultados en términos de explicatividad ( $R^2$  ajustado) y errores (RMSE).

Model	Adjusted R-Squared	R-Squared	RMSE	Time Taken
GradientBoostingRegressor	0.88	0.88	162.08	0.29
ExtraTreesRegressor	0.83	0.84	193.25	0.39
BaggingRegressor	0.82	0.83	198.51	0.09
RandomForestRegressor	0.82	0.83	199.09	0.71
XGBRegressor	0.77	0.78	223.58	0.20

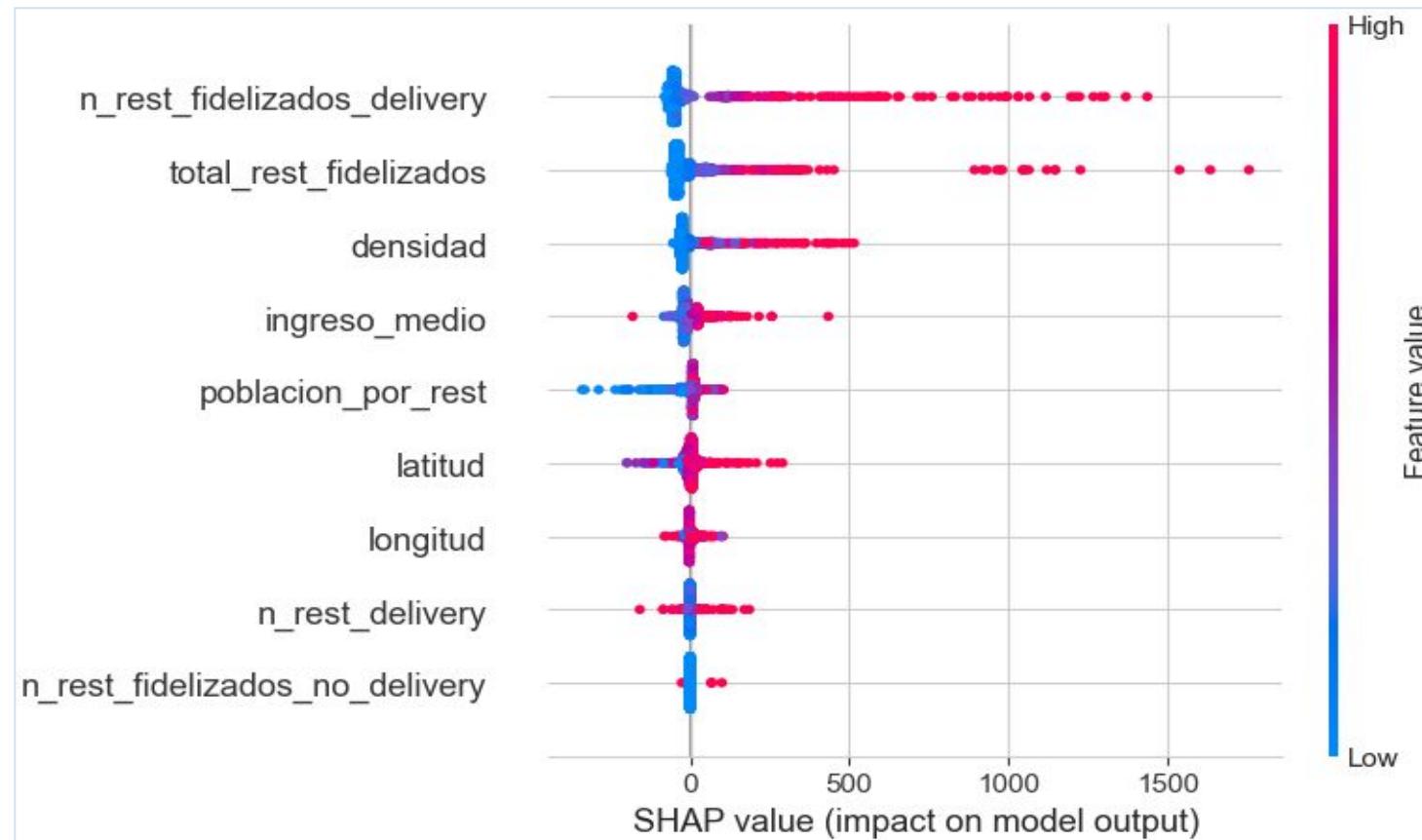
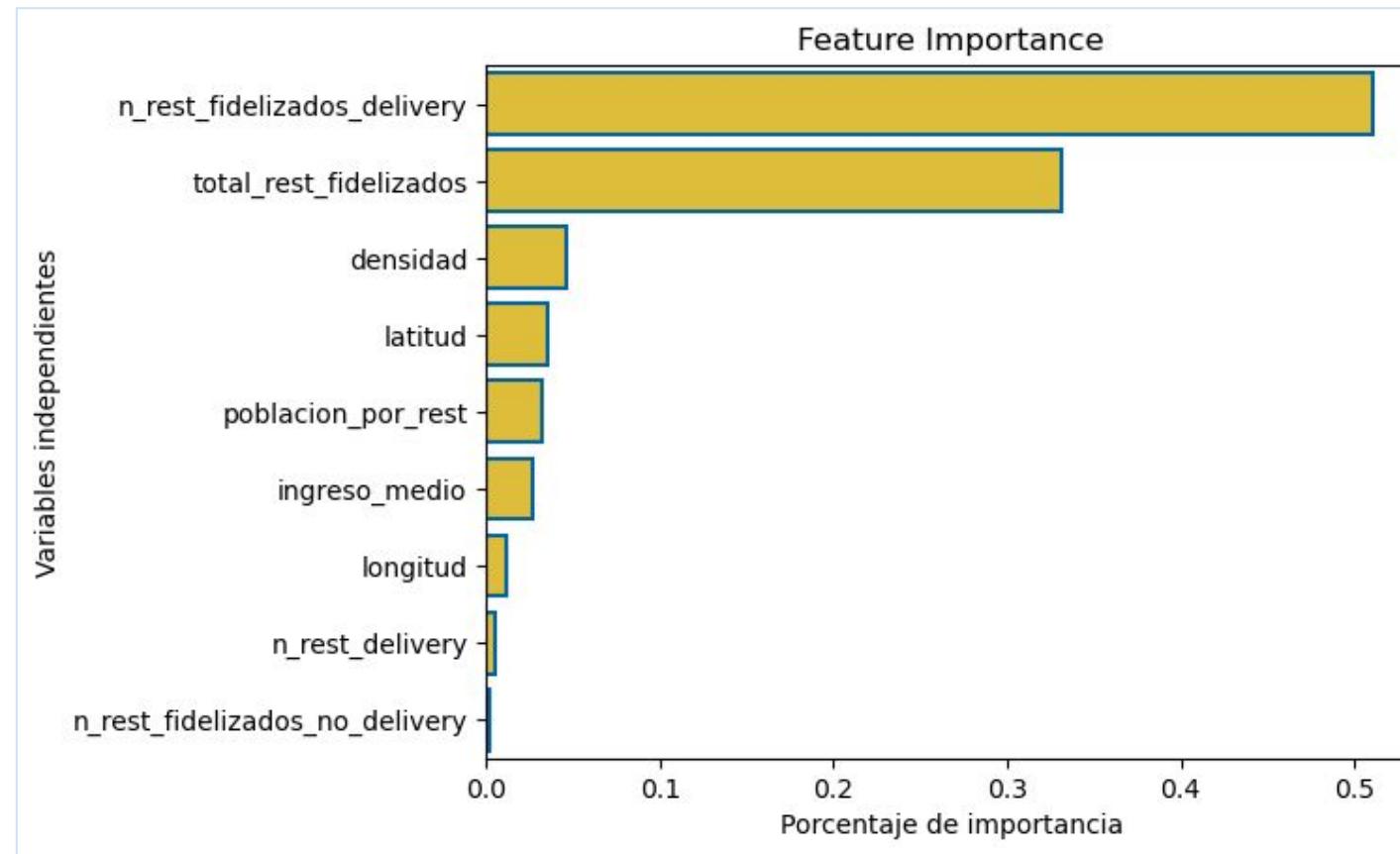




# Selección de variables independientes

Para escoger las variables independientes a introducir en el modelo, se han utilizado las técnicas de **Feature Importance** y **SHAP Values**. Con ellas, se han seleccionado las más influyentes en la variabilidad de pedidos.

Feature Importance muestra **cuánto** contribuye cada variable, mientras que SHAP Values explica **cómo** cada variable contribuye a la predicción:



Cabe destacar que no se ha introducido la variable dependiente o ninguna variable generada a partir de ésta en el modelo porque se estaría tratando de predecir la variable objetivo a partir de sí misma.

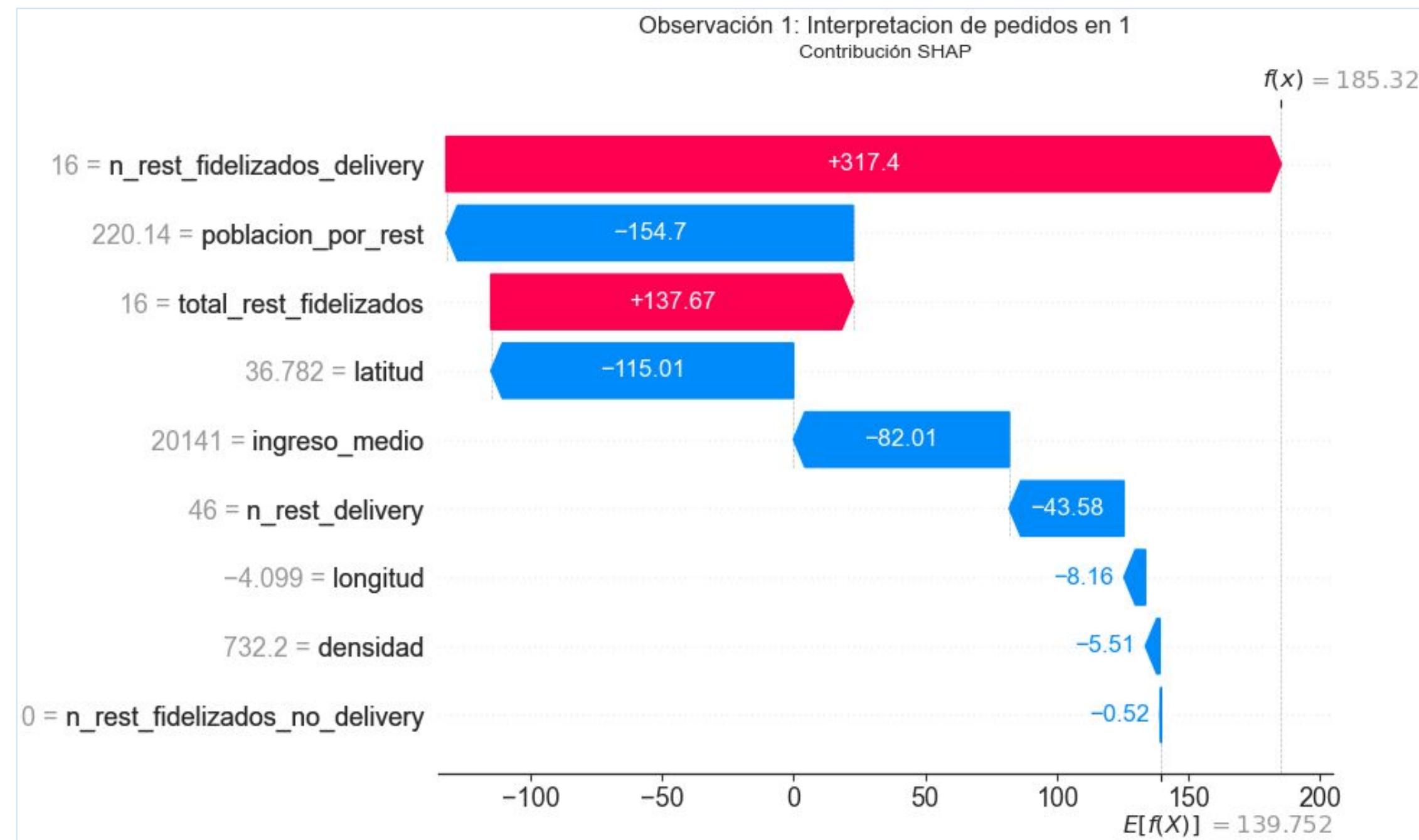


Modelo regresión



# Explicatividad mediante Waterfall SHAP

A continuación, se muestra un **Waterfall SHAP** para una ciudad concreta con el fin de explicar cómo el modelo de regresión realiza las predicciones a partir del valor de las variables independientes introducidas.



Modelo regresión



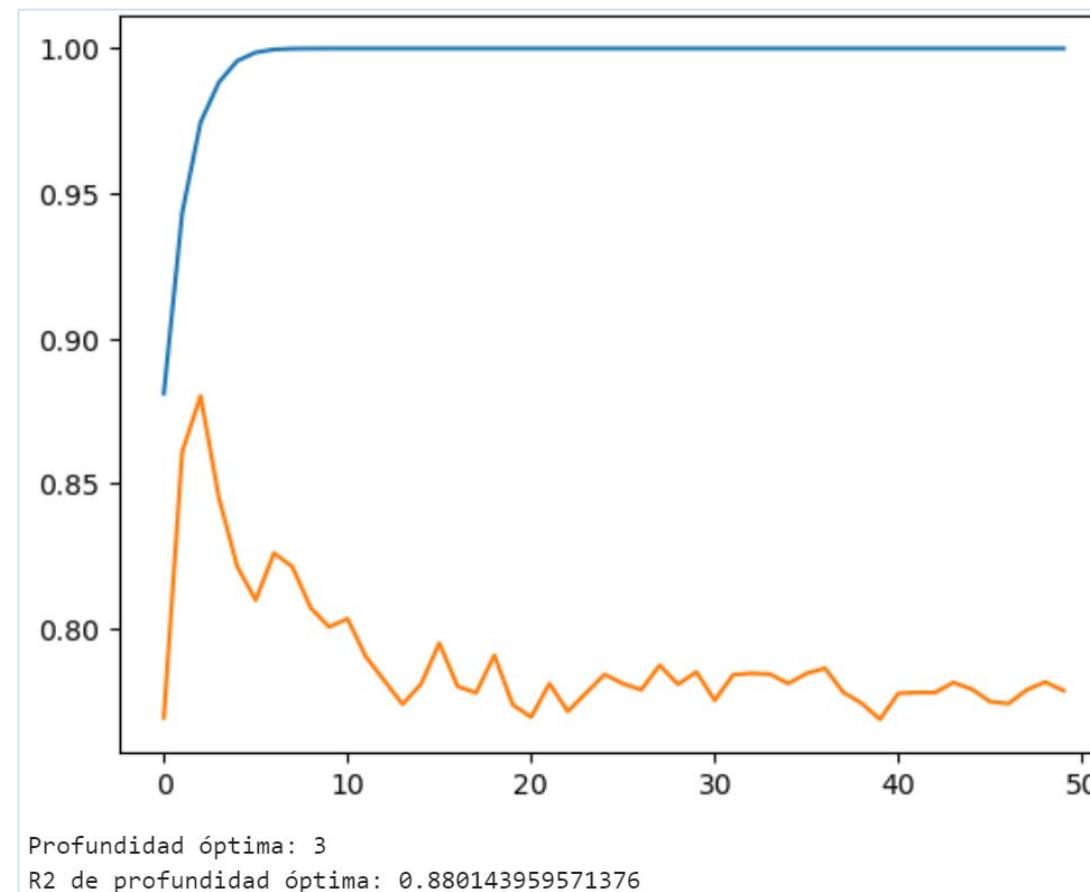


# Ajuste de profundidad

Una vez seleccionadas las **variables independientes**, es importante evitar que el modelo **sobreajuste**, es decir, que aprenda de memoria los datos de entrenamiento en lugar de generalizar y hacer predicciones precisas con nuevos datos.

Para ello, se ha calculado el número óptimo de niveles, que es el punto en el que comienza a decaer.

En este caso, la profundidad óptima es 3, con un **R<sup>2</sup> del 88%**.



```
# Bucle para encontrar la profundidad adecuada del modelo para que este no sobreajuste
for i in range(1, 51):

    # Modelo Gradient Boosting Regressor
    gbr = GradientBoostingRegressor(max_depth = i)
    gbr_modelo = gbr.fit(x_train, y_train)

    # Predicción con Gradient Boosting Regressor
    y_pred = gbr.predict(x_test)

    # Puntuaciones de R2 en entrenamiento y prueba
    score_train.append(gbr.score(x_train, y_train))
    score_test.append(gbr.score(x_test, y_test))

    # R2 en conjunto de prueba
    r2 = r2_score(y_test, y_pred)

    # Actualización de la profundidad óptima y el R2 máximo
    if r2 > r2_maximo:
        max_depth_optimo = i
        r2_maximo = r2
```



# Resultados del modelo de regresión

Para evaluar el rendimiento del modelo se han calculado las siguientes métricas:

Métrica	Valores
R <sup>2</sup> ajustado	88%
Media de pedidos	187,6
MAE	74
MAPE	22,2

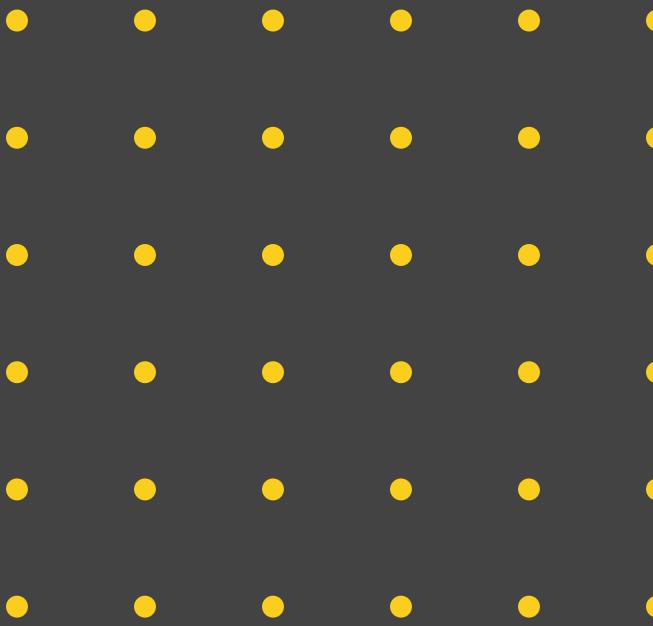
A pesar de que se ha logrado obtener un R<sup>2</sup> ajustado alto, el MAPE alcanzado es tan elevado que tiene muy poca fiabilidad.

Por esta razón es por la que se ha decidido iterar el modelo inicial y realizar un modelo de clasificación



Modelo regresión

# 03



## Modelo de clasificación



# Elección del modelo de clasificación

Al tener un MAPE alto en la regresión realizada, se ha decidido convertir la variable objetivo a **categórica**. Para ello, se han utilizado los cuartiles para dividir el número de pedidos en cuatro categorías en función de su **potencialidad**:



Para seleccionar el modelo, se ha obtenido que el **Decision Tree** es el que mejores resultados en cuanto a precisión, pero, para **reducir el sobreajuste** al que tiende este modelo, se ha optado por **Random Forest**.

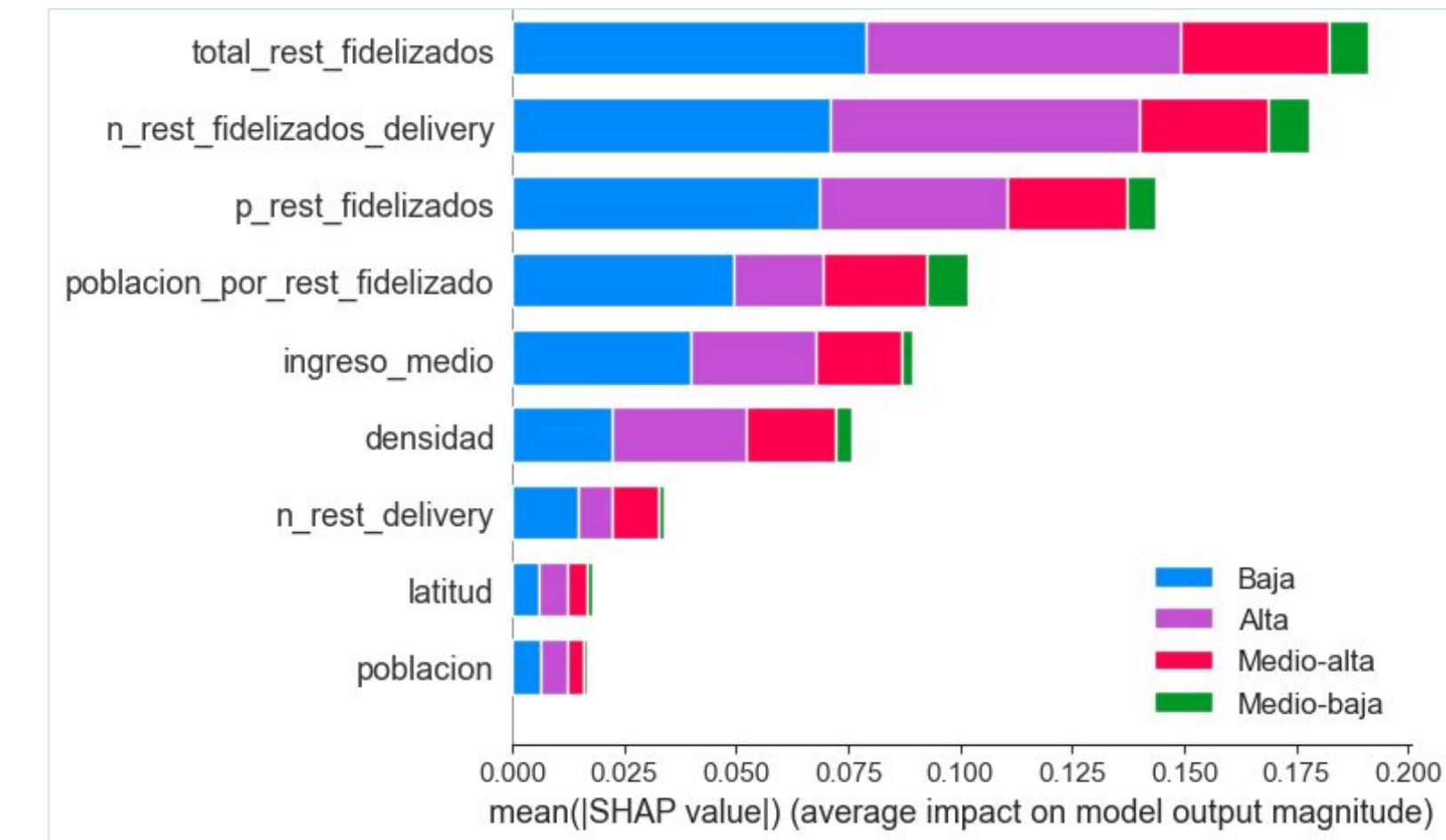
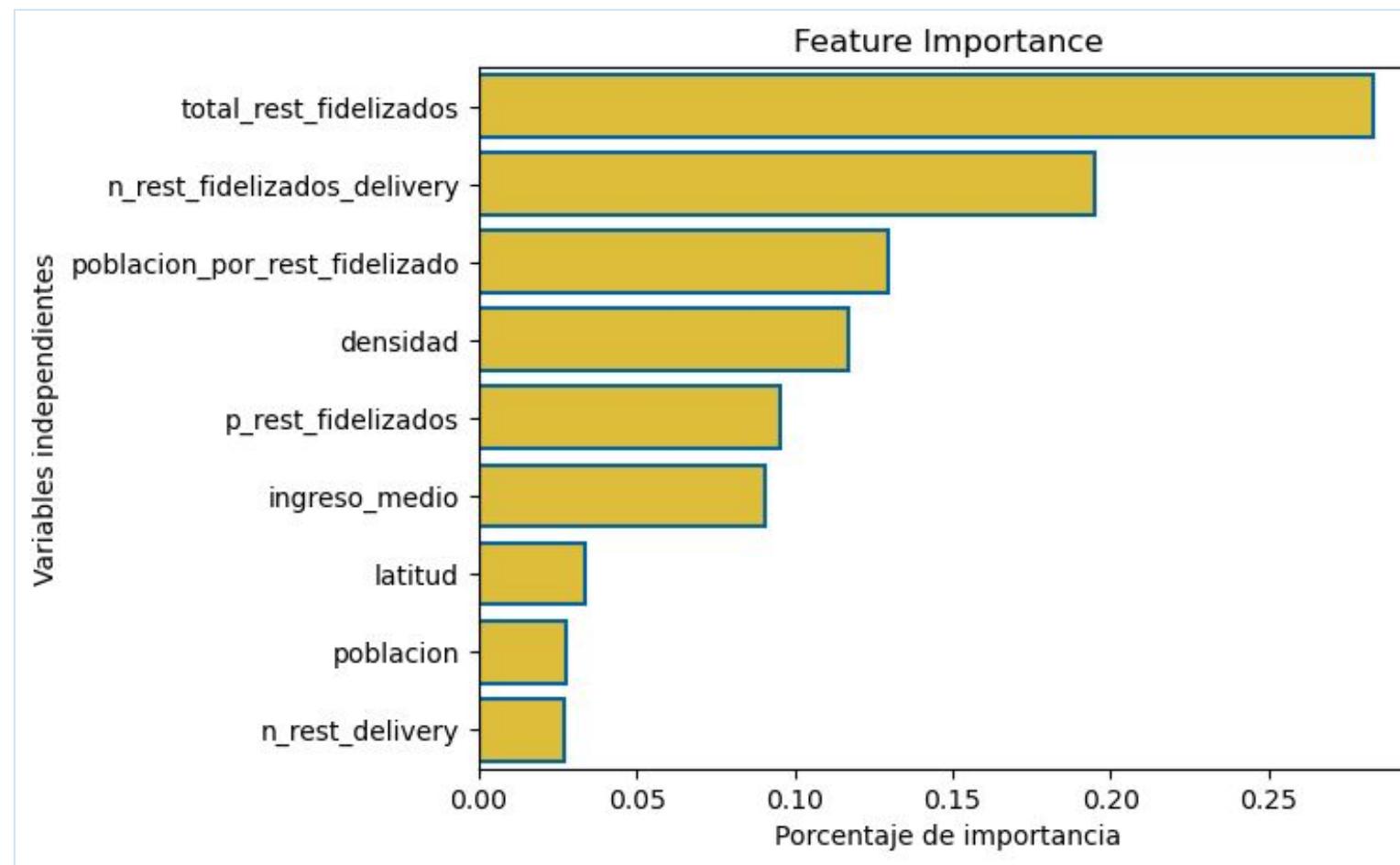
Respecto a los resultados, el enfoque ha sido obtener la máxima precisión posible en la **categoría 'Alta'**, ya que ésta es la única relevante para el caso de uso actual. Mejorar la precisión en las otras categorías no aportaría valor al análisis y podría disminuir la precisión en la categoría 'Alta', que es la más importante de cara a una toma de decisiones efectiva.



# Selección de variables independientes

Al igual que en el modelo de regresión, para determinar las variables que más influyen en la variabilidad de los pedidos, se han utilizado las técnicas de **Feature Importance** y **SHAP Values**.

Los resultados han sido los siguientes:





# Ajuste de profundidad

Elegidas las **variables independientes**, se ha procedido a encontrar la profundidad óptima, sin sobreajustar el modelo a los datos de entrenamiento y siempre priorizando mantener la máxima precisión en la categoría 'Alta'.

Se ha encontrado que la profundidad de 3 es la que obtiene la mejor precisión para la categoría 'Alta'.

```
# Bucle para encontrar la profundidad adecuada para que el modelo no sobreajuste
for i in range(1, 51):

    # Modelo Random Forest Classifier
    rfc = RandomForestClassifier(max_depth = i)
    rfc_modelo = rfc.fit(x_train, y_train)

    # Predicción con Random Forest Classifier
    y_pred = rfc.predict(x_test)

    # Precisión de la categoría 'Alta'
    precision_alta = precision_score(y_test, y_pred, labels=['Alta'], average='macro')

    # Actualización de la profundidad óptima y el R2 máximo
    if precision_alta > precision_max_alta:
        max_depth_optimo = i
        precision_max_alta = precision_alta
```



# Resultados del modelo de clasificación

Para evaluar el rendimiento del modelo se ha calculado la tabla de confusión que resume los verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos en cada categoría.

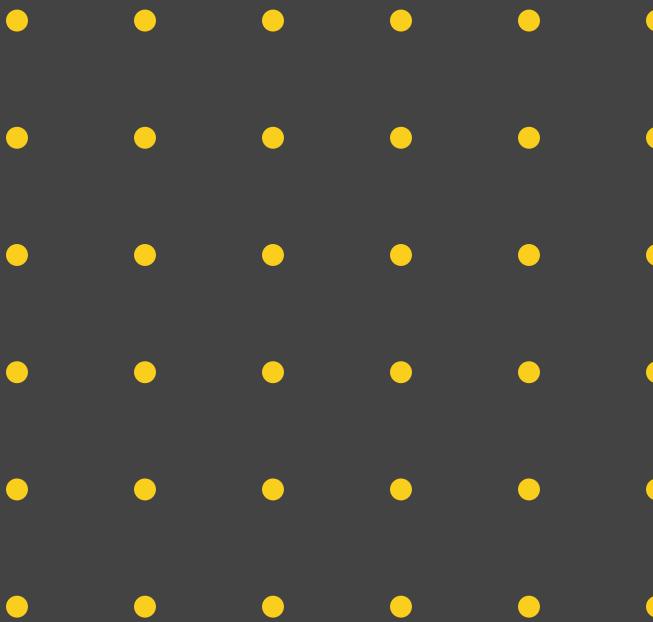
Categorías	Precisión	Recall	F1-score	Support
Alta	81%	81%	81%	48
Medio-alta	55%	52%	53%	46
Medio-baja	0%	0%	0%	16
Baja	72%	87%	78%	90

Dentro de esta tabla, realmente la métrica que más sentido tiene analizar a nivel negocio es la precisión de las ciudades con **potencial ‘Alto’**, ya que es en esas ciudades en las que se busca centrar la atención y el foco del equipo comercial.



Modelo clasificación

# 04



## Modelo de clasificación alternativo



# Elección de variable dependiente alternativa y de modelo

A pesar de haber obtenido un buen resultado en la precisión de la categoría ‘Alta’, se ha querido buscar una variable más significativa en la potencialidad de la ciudad. Como no se cuenta con ningún dato referente a los ingresos, se ha decidido optar por el número de pedidos mensuales que se realizan por restaurante fidelizado en cada ciudad.



Se han probado con varios modelos de clasificación y el que mejor responde a la predicción de las ciudades con potencialidad alta es el **Gradient Boosting**, por lo que es el que se ha utilizado.

Como en los modelos anteriores, también se ha buscado la profundidad óptima para obtener el mejor desempeño del modelo para nuestro objetivo.

Modelo clasificación  
alternativo

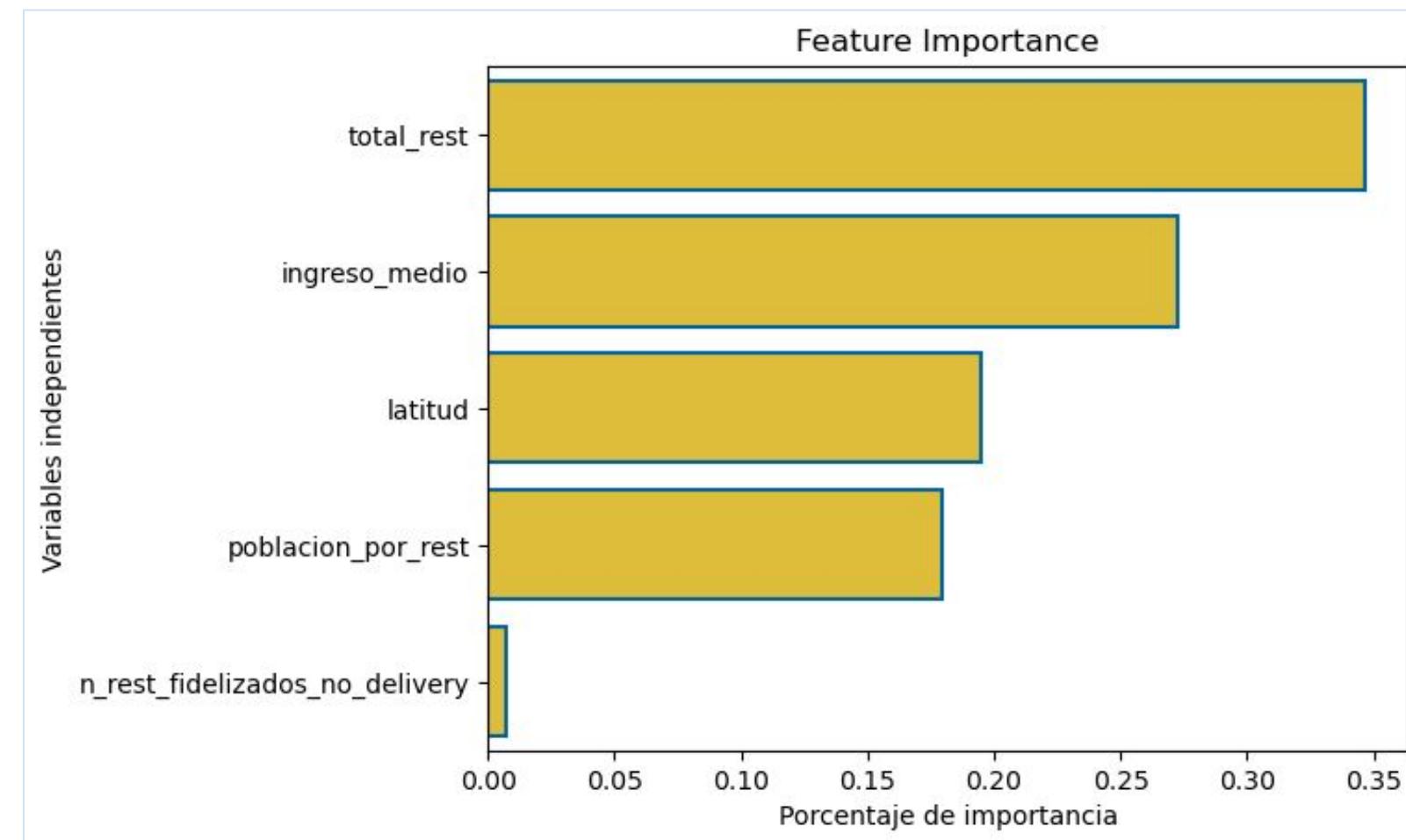


# Selección de variables independientes

Con el fin de identificar las variables que más afectan a la variabilidad del número de pedidos por restaurante fidelizado, se ha aplicado la técnica de **Feature Importance**.

Para este modelo en particular, no se ha podido utilizar SHAP Values debido a que esta técnica no es compatible con modelos de clasificación no binarios que utilicen Gradient Boosting como algoritmo.

Estas fueron las variables elegidas junto con su importancia en el modelo:



Modelo clasificación alternativo





# Resultados del modelo alternativo

Nuevamente, se ha elaborado una tabla de confusión que resume los resultados correctos e incorrectos del modelo en cada categoría, mostrando el número de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.

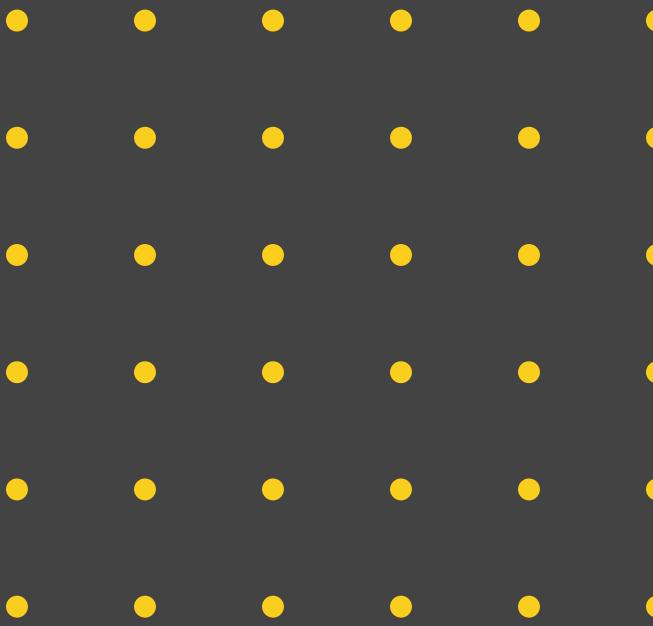
Categorías	Precisión	Recall	F1-score	Support
Alta	74%	52%	61%	48
Medio-alta	29%	26%	27%	39
Medio-baja	-	-	-	-
Baja	74%	87%	80%	113

Al estar enfocados en ciudades con potencial 'Alto', lo más importante vuelve a ser la precisión de esa categoría. Por lo tanto, esa es la métrica que debemos analizar cuidadosamente para tomar decisiones informadas en el negocio del delivery.



Modelo clasificación  
alternativo

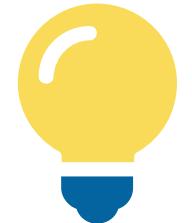
# 05



## Resumen del proyecto



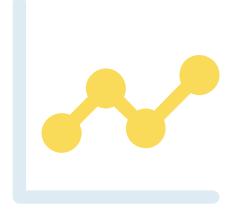
# Resumen del proyecto



1

## Elección del modelo y cálculo de potencialidad

Probando diferentes modelos para comprobar cuál se adapta mejor a nuestro objetivo



2

## Propuestas de mejoras para el modelo elegido

Una vez elegido el modelo, establecer pautas a futuro de cómo pueden mejorar los resultados obtenidos



3

## Definición de las acciones de negocio

Trasladar todo el trabajo previo a insights accionables que sean útiles para el departamento comercial



# 01. Elección de modelo y cálculo de potencialidad

A pesar de que el primer modelo de clasificación es el que mejor resultados tiene en cuanto a precisión en las ciudades con potencialidad alta, se ha considerado que la variable de **pedidos por restaurante fidelizado** es más significativa para la toma de decisiones estratégica y de negocio.

Por lo tanto, este último modelo es el seleccionado y el que se utilizará para ayudar al equipo comercial a estimar el potencial de una ciudad, a modo de poder planificar cuáles son las ciudades a priorizar en cada mes.

**Los pasos que han seguido para calcular potencial son los siguientes:**

- 1** Escalar entre 0 y 1 las variables independientes usadas en el modelo.
- 2** Multiplicar el valor de las variables escaladas por el porcentaje de importancia determinado por Feature Importance.
- 3** Ordenar de mayor a menor las ciudades en base al resultado calculado.

## Top 10 ciudades con mayor potencial

1. Rozas de Madrid, Las
2. Fuengirola
3. Benidorm
4. Majadahonda
5. Pozuelo de Alarcón
6. San Sebastián de los Reyes
7. Arona
8. Tres Cantos
9. Sant Cugat del Vallès
10. Adeje





## 02. Propuestas de mejora para el modelo elegido

De cara a mejorar el modelo, existen diversas iniciativas que se podrían considerar:

- a)** Incluir variables adicionales con valor de negocio que pudieran tener un impacto en el modelo. Por ejemplo, el ticket medio por ciudad, las reviews de los clientes, el tipo de restaurante, etc.
- b)** Clasificar restaurantes por tiempo de atención.
- c)** Utilizar técnicas de clustering para tratar zonas más grandes que la ciudad.



# 03. Acciones de negocio

## Identificar las ciudades con mayor potencial

Los departamentos de negocio pueden identificar las ciudades con mayor potencial para delivery. De esta manera, se puede enfocar la expansión de la empresa en aquellas ciudades donde es más probable que se obtengan mejores resultados.

## Comparar el potencial de crecimiento de las ciudades para elegir la mejor opción de expansión

Se podrían seleccionar aquellas ciudades que tengan un mayor potencial de crecimiento basado, por ejemplo, en el grado de penetración, la cuota de mercado o el peso de los restaurantes con delivery.

## Optimizar expansión

Se puede utilizar el modelo para fidelizar el número adecuado de restaurantes para expandirse en cada una de las ciudades seleccionadas.

## Personalizar la estrategia de marketing para cada ciudad en función de su potencial de delivery

Por ejemplo, si se conocen los tipos de restaurantes de cada ciudad, si hay alguna que tenga un alto potencial para la comida italiana, las campañas se podrían enfocar en promocionar estos restaurantes.





**JUST DELIVERY**

FOOD IS ON ITS WAY!