Análisis cuantitativo avanzado

Clase 1: Regresión Lineal

Lic. Lucio José Pantazis

Motivación

Carga de datos

La siguiente base "home_sales_nyc.csv" describe datos sobre ventas inmobiliarias en la ciudad de Nueva York (disponible en https://github.com/DataScienceForPublicPolicy/diys/tree/main/data).

Para incorporarla a la memoria y poder trabajar con esta base, la cargamos directamente de internet usando el link:

```
# Importo la librería necesaria para procesar bases de datos
import pandas as pd

# El link a la base de datos como string
url = 'https://raw.githubusercontent.com/DataScienceForPublicPolicy/diys/main/data/home_sales_nyc.csv'
```

```
# Se usa la librería pandas para leer el .csv
df = pd.read_csv(url)
```

Inspección inicial

Para verificar que la base está bien cargada, visualizamos las primeras 5 filas de la misma usando el método `.head()`:

```
# El método "head" muestra por default las primeras 5 filas de una base
df.head()
```

```
##    time.index  borough neighborhood  ...   sale.date  sale.year  age
## 0          14        2     BATHGATE  ...  2018-08-17       2018  117
## 1         162        2     BATHGATE  ...  2018-03-22       2018  119
## 2         298        2     BATHGATE  ...  2017-11-06       2017  118
## 3         179        2     BATHGATE  ...  2018-03-05       2018  119
## 4         336        2     BATHGATE  ...  2017-09-29       2017  118
##
## [5 rows x 15 columns]
```

Por limitaciones en lo que se puede exponer, no se ven todas las columnas, por lo que se muestran sólo algunas variables y las columnas intermedias se marcan con puntos suspensivos.

Sin embargo, la última línea especifica que la base tiene 15 variables, por más que no se dispongan en el output. Por otro lado, las 5 filas que se mencionan no son las totales de la base de datos, son sólo las que se muestran.

La primer columna no se corresponde con ninguna variable y hace referencia al índice, utilizado para recorrer las distintas filas.

Inspección inicial

Para seguir analizando la base, vamos a apelar al método `.info()`

```
df.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 12668 entries, 0 to 12667
## Data columns (total 15 columns):
##  #   Column                   Non-Null Count  Dtype
## ---  ------                   --------------  -----
##  0   time.index               12668 non-null  int64
##  1   borough                  12668 non-null  int64
##  2   neighborhood             12668 non-null  object
##  3   building.class.category  12668 non-null  object
##  4   zip.code                 12668 non-null  int64
##  5   residential.units        12668 non-null  int64
##  6   commercial.units         12668 non-null  int64
##  7   total.units              12668 non-null  int64
##  8   land.square.feet         12668 non-null  int64
##  9   gross.square.feet        12668 non-null  int64
##  10  year.built               12668 non-null  int64
##  11  sale.price               12668 non-null  float64
##  12  sale.date                12668 non-null  object
##  13  sale.year                12668 non-null  int64
##  14  age                      12668 non-null  int64
## dtypes: float64(1), int64(11), object(3)
## memory usage: 1.4+ MB
```

En este output, se ve que la base tiene 12668 filas y se visualizan 3 tipos de datos:

- object: variables categóricas o cualitativas
- int: variables numéricas de valor entero
- float: variables numéricas

Las variables

- `time.index`: Variable que permite ordenar las ventas cronológicamente. (Variable cuantitativa)
- `borough`: Distrito de cada inmueble. (Variable cualitativa)
- `neighborhood`: Barrio del inmueble. (Variable cualitativa)
- `building.class.category`: Clasificación del inmueble. (Variable cualitativa)
- `zip.code`: Código postal. (Variable cualitativa)
- `residential.units`: Cantidad de unidades residenciales del inmueble. (Variable cuantitativa)
- `commercial.units`: Cantidad de unidades comerciales del inmueble. (Variable cuantitativa)
- `total.units`: Cantidad de unidades totales del inmueble. (Variable cuantitativa)
- `land.square.feet`: Superficie del lote, medido en pies al cuadrado. (Variable cuantitativa)
- `gross.square.feet`: Superficie del inmueble, medido en pies al cuadrado. (Variable cuantitativa)
- `year.built`: Año de construcción del inmueble. (Variable cuantitativa)
- `sale.price`: Precio de venta del inmueble, medido en dólares. (Variable cuantitativa)
- `sale.date`: Fecha de venta del inmueble. (Variable temporal)
- `sale.year`: Año de venta del inmueble. (Variable cuantitativa)
- `age`: Antigüedad del inmueble. (Variable cuantitativa)

Motivación

Haremos foco ahora en cómo se vinculan las siguientes variables:

- `gross.square.feet`: Superficie del inmueble, medido en pies al cuadrado. (Variable cuantitativa)
- `sale.price`: Precio de venta del inmueble, medido en dólares. (Variable cuantitativa)

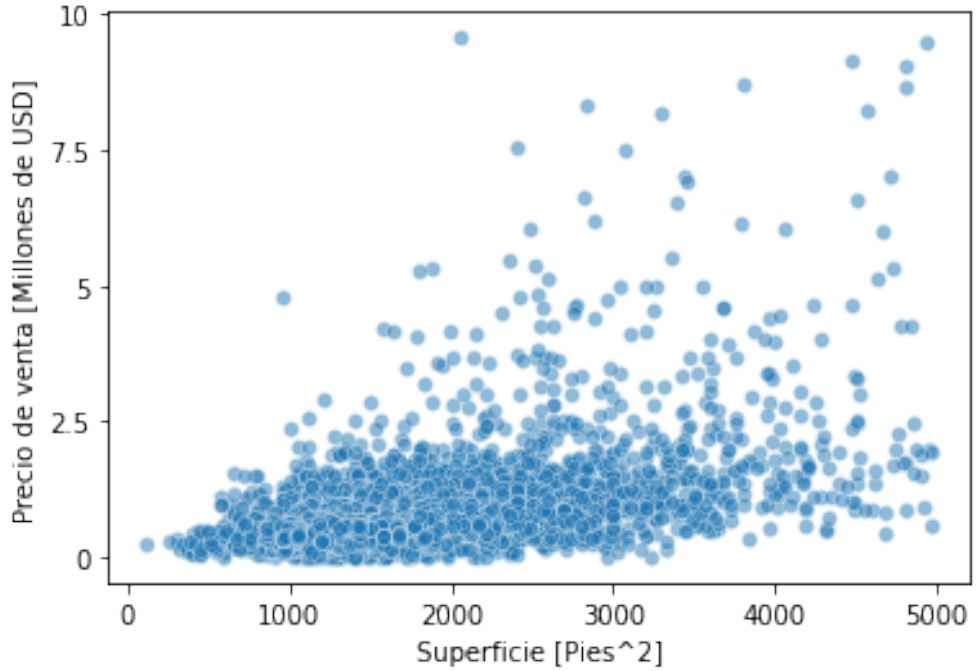
Se puede pensar que estas variables están vinculadas, ya que intuitivamente, a medida que los inmuebles son más grandes, más precio de venta tendrán.

Gráfico exploratorio

Gráfico de dispersión

Para analizar esta dependencia, hacemos un gráfico de dispersión ("scatterplot" en inglés), donde cada inmueble se corresponde con un punto en el gráfico, cuya coordenada horizontal es la superficie, y la coordenada vertical es el precio de venta.

```
# Cargo los paquetes necesarios
import matplotlib.pyplot as plt;import seaborn as sns
# Creo un scatterplot que considere la base de datos df como parámetro
# fijando la coordenada x como la superficie y la coordenada y como el precio de venta
sns.scatterplot(data=df,x="gross.square.feet",y="sale.price",alpha=0.5)
# Al gráfico g se le agregan nombres adecuados para los ejes
plt.xlabel("Superficie [Pies^2]");plt.ylabel("Precio de venta [Millones de USD]")
plt.yticks(ticks=[0,0.25e7,0.5e7,0.75e7,1e7],labels=[0,2.5,5,7.5,10])
# Este comando obliga a Python a mostrar el gráfico
plt.show()
```



(Sí, hay inmuebles de casi 10 millones de dólares. . .)

Gráfico exploratorio

Sin embargo, surgen las siguientes preguntas:

- ¿Cómo es ese crecimiento?
- En base a estos datos, ¿Hay una forma de estimar el precio de venta de un inmueble sabiendo su superficie?

Regresión lineal

Modelos de regresión

En estadística, cuando se busca vincular los valores de una variable de interés a partir de otra/s, se aplica un "modelo de regresión".

En este caso, se busca explicar el precio de venta de un inmueble en función de su superficie.

- La variable de interés principal (en este caso) es el precio del inmueble.
  En los modelos de regresión esta variable se denomina **variable de respuesta** o **variable dependiente**.
  Suele ser notada con la letra Y.
- Para explicar los precios, se utiliza (en este caso) la variable de superficie.
  Generalmente estas variables se denominan **covariables**,
  variables explicativas, **variables predictivas**
  o **variables independientes**. Suelen ser notadas con la letra X

Regresión lineal simple

El modelo más simple de regresión se llama el modelo de regresión *lineal* simple. Según indica su nombre, este modelo asume que las variables

- Y: precio del inmueble
- X: superficie del inmueble

se disponen alrededor de una *recta*:

$$Y \approx a + b \cdot X$$

donde a representa la *ordenada* (en inglés, "intercept")
y b representa la *pendiente* (en inglés, "slope").

En términos de nuestro ejemplo, sería equivalente a plantear que existen dos valores a y b de forma que para cada inmueble (individualizado por el índice i) se cumple:

$$P_i \approx a + b \cdot S_i$$

Comentarios:

- El signo "$\approx$" simboliza una aproximación. Esto se debe a que el precio no tiene porqué obedecer "exactamente" la ecuación.
- Del mismo modo, el modelo no tiene porqué ser válido. Es una estructura en los datos propuesta por el analista, que tiene que contrastarse con los datos para evaluar su validez.

Estimación de a y b

Hasta ahora, el modelo plantea que existen dichas constantes a y b, pero no dice como se calculan.

La estimación busca que para cada inmueble i, la distancia entre el precio y la aproximación sea lo más pequeño posible. Como la distancia debe ser positiva, se utiliza el cuadrado de la diferencia. Es decir, se busca minimizar:

$$(\mathrm{P}_i - a - b \cdot S_i)^2$$

Sin embargo, como el modelo se busca a aplicar a **todos** los inmbuebles, puede suceder que la elección de a y b que minimiza la distancia para un inmueble puede generar una distancia máxima en otro.

Por lo tanto, se buscan los valores de a y b que minimizan la **suma** sobre todos los inmuebles de todas estas diferencias al cuadrado. Los valores **óptimos** se denominan "estimadores de mínimos cuadrados" y suelen notarse con un acento circunflejo: $\widehat{a}$ y $\widehat{b}$.

Estimación en python

Para obtener estos valores en Python, se usa el paquete `sklearn`

```
# Importo el paquete necesario
from sklearn.linear_model import LinearRegression
# covs representa la covariable (superficie)
covs=["gross.square.feet"];X=df[covs]
# resp representa la variable de respuesta (precio)
resp=["sale.price"];y=df[resp]
# Se inicializa la regresión
mod=LinearRegression()
# Se buscan los valores óptimos de a y b para los datos de la base
mod.fit(X,y)
```

```
## LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
import numpy as np
# El valor óptimo de a se obtiene con "intercept_" y el de b con ".coef_".
# Como se devuelven listas, se convierten a variables numéricas con "float"
print([np.round(float(mod.intercept_),2),np.round(float(mod.coef_),2)])
```

```
## [-42584.39, 466.18]
```

Esto significa que según este modelo, para un inmbueble de, por ejemplo, 2000 pies cuadrados, se estima un precio de venta de:

$$-42584.39 + 466.18 \cdot 2000 = 889775.6$$

En código de Python, sería del siguiente modo:

```
print(float(mod.intercept_)+float(mod.coef_)*2000)
```

```
## 889768.1641673434
```

Vemos que el valor no es exactamente el mismo, pero eso se debe a que redondeamos los valores de $\widehat{a}$ y $\widehat{b}$ a dos decimales.

Interpretación de los coeficientes

Según estas estimaciones, se pueden interpretar ambos coeficientes del siguiente modo:

- El valor del intercept (a) nos dice que los precios *estimados* de los inmuebles comienzan desde un valor de -42584.39 USD, correspondiente a un inmueble de "superficie 0".

Comentario: No tiene sentido económico un inmueble de precio negativo, pero tampoco tiene sentido considerar un inmueble de 0 pies cuadrados. Justamente el modelo no es extrapolable a estos valores, ya que recordemos que el modelo es una propuesta que puede ser válida o no en distintos casos.

- El valor de la pendiente (b) nos dice que cada pie cuadrado aporta 466.18 USD al precio *estimado* del inmueble.

Capacidades explicativas y predictivas

La anterior estimación nos permite pensar en dos aspectos del modelo:

- Capacidad explicativa: Por más que hayamos encontrado valores óptimos de a y b para los datos obtenidos, ¿es esa relación lineal entre ambas variables representativa de lo observado? Es decir, ¿el modelo **explica** la relación entre los datos **observados**?
- Capacidad predictiva: Más allá de si los datos quedan bien explicados, en el caso de incluirse un **nuevo** inmueble, ¿es buena la estimación obtenida a través de los coeficientes a y b? Es decir, ¿permite el modelo **predecir** el precio de venta de nuevos inmuebles?

Predicción en Python

Para predecir en Python, una vez obtenidos los coeficientes *a* y *b*, se utiliza el método `predict`. Como parámetro, conviene introducir una base de datos con el mismo nombre de la variable predictiva que se utilizó en la estimación (en este caso, "gross.square.feet"):

```
# Valor NUEVO de superficie utilizado para estimar su precio de venta
X_pred=pd.DataFrame({"gross.square.feet":[2000]})
# Precio de venta estimado para un inmueble de dicha superficie
y_pred=mod.predict(X_pred)
print(float(y_pred))
```

```
## 889768.1641673434
```

Vemos que coincide el valor obtenido anteriormente

```
print(float(mod.intercept_)+float(mod.coef_)*2000)
```

```
## 889768.1641673434
```

Predicción en Python

Del mismo modo, se pueden predecir precios de venta para varios inmuebles:

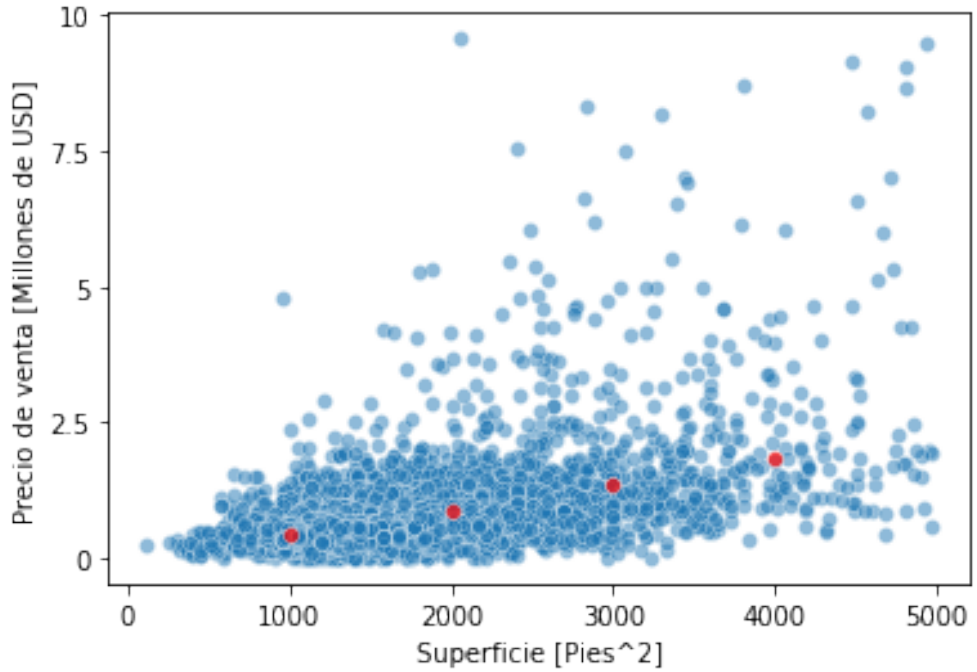
```
import numpy as np
# Valores nuevos de superficie utilizados para estimar su precio de venta
X_pred=pd.DataFrame({"gross.square.feet":[1000,2000,3000,4000]})
# Precios de venta estimados para inmuebles de dichas superficies
y_pred=mod.predict(X_pred)
print([np.round(float(y),2) for y in y_pred])
```

```
## [423591.89, 889768.16, 1355944.44, 1822120.72]
```

Predicción en Python

Por lo tanto, en el gráfico inicial podemos agregar las estimaciones para ver cómo se adapta a los datos originales:

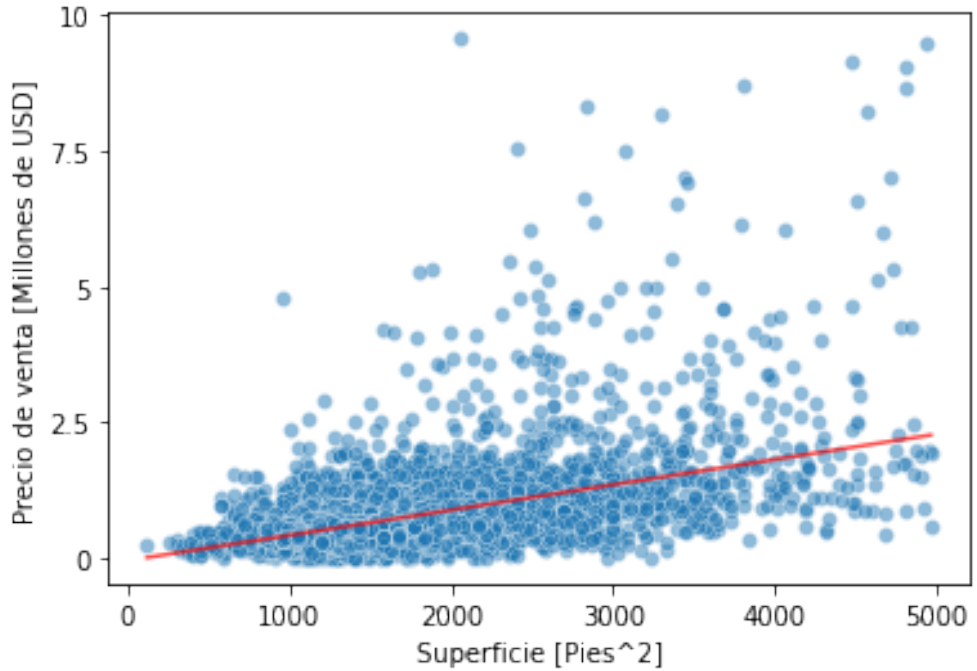
```
# Creo una base de datos con predicciones, manteniendo los nombres de las variables originales
# para superponer en el gráfico consistentemente.
df_pred=X_pred;df_pred["sale.price"]=y_pred
# Agrego los puntos de la base original
sns.scatterplot(data=df,x="gross.square.feet",y="sale.price",alpha=0.5)
# Agrego las predicciones
sns.scatterplot(data=df_pred,x="gross.square.feet",y="sale.price",alpha=0.7,color="red")
# Corrijo los ejes
plt.xlabel("Superficie [Pies^2]")
plt.ylabel("Precio de venta [Millones de USD]")
plt.yticks(ticks=[0,0.25e7,0.5e7,0.75e7,1e7],labels=[0,2.5,5,7.5,10])
plt.show()
```



Predicción en Python

Para evaluar gráficamente el modelo, se puede unir con una línea dos valores predichos, tomando los mínimos y máximos de la coordenada x.

```
# Calculo el mínimo y el máximo
minX=df['gross.square.feet'].min();maxX=df['gross.square.feet'].max()
# Genero una nueva base con las predicciones
df_pred=pd.DataFrame({"gross.square.feet":[minX,maxX]})
df_pred["sale.price"]=mod.predict(df_pred)
# Uno ambas predicciones con una línea
sns.scatterplot(data=df,x="gross.square.feet",y="sale.price",alpha=0.5)
sns.lineplot(data=df_pred,x="gross.square.feet",y="sale.price",alpha=0.7,color="red")
plt.xlabel("Superficie [Pies^2]");plt.ylabel("Precio de venta [Millones de USD]")
plt.yticks(ticks=[0,0.25e7,0.5e7,0.75e7,1e7],labels=[0,2.5,5,7.5,10])
plt.show()
```



Evaluación del modelo

Ahora surge la pregunta: Más allá de lo gráfico, ¿cómo podemos evaluar si este modelo es representativo de los datos?

Un valor estándard para evaluar el rendimiento de un modelo, es la magnitud denominada R^2. Este valor evalúa las similitudes entre la variable dependiente y las predicciones del modelo. Sus valores van entre 0 y 1, y mientras más cercanos esté a 1, mejor es el modelo para representar los datos.

En Python, dicho valor se calcula utilizando el modelo con a y b ya estimados. Como ya se aplicó el método `fit`, se puede calcular el R^2 con el método `score`:

```
mod.score(X,y)
```

```
## 0.2540788486552811
```

Vemos que el valor de R^2 no es tan alto. En el gráfico se ve que para algunos valores altos de la superficie, la recta queda lejos de algunos inmuebles con precios muy altos de venta.

Modelo cuadrático

Modelo cuadrático

Como se dijo previamente, el modelo lineal no tiene porqué representar bien a los datos. Además, observando el crecimiento de los datos, pareciera que puede ser mejor representado por una parábola. Es decir, se puede plantear el siguiente modelo para cada inmueble i:

$$P_i \approx a + b \cdot S_i + c \cdot S_i^2$$

Ahora, deben encontrarse valores óptimos de a, b y c para los datos en cuestión. La idea es la misma que el modelo de la recta. Se utiliza el criterio de minimizar la suma de las distancias al cuadrado:

$$(P_i - a - b \cdot S_i - c \cdot S_i^2)^2$$

Modelo cuadrático en Python

Para calcular estos valores en Python, nos surge un inconveniente: la variable que tiene los valores de la superficie al cuadrado, no está presente en la base de datos. Sin embargo, podemos generarla fácilmente a partir de la variable de superficie. Llamaremos a esta nueva variable "gross.square.feet.sq" y podemos verificar que los valores de esta nueva variable coinciden con la original al cuadrado:

```
df["gross.square.feet.sq"]=df["gross.square.feet"]**2
print(df[["gross.square.feet","gross.square.feet.sq"]].head())
```

```
##    gross.square.feet  gross.square.feet.sq
## 0               1474               2172676
## 1               1470               2160900
## 2               1497               2241009
## 3               1497               2241009
## 4               1340               1795600
```

Estimación de a, b y c

Una vez que tenemos esta variable disponible, para estimar los valores de a, b y c usamos la misma estrategia que usamos para el modelo lineal. Sin embargo, hay algunas salvedades.

Ahora las variables explicativas son 2 (`"gross.square.feet"` y `"gross.square.feet.sq"`) y por lo tanto, el método `coef_` devuelve dos valores (uno para b y el otro para c, mantienen el orden con el que fueron dados como input):

```
# Las covariables son 2 y se guardan en la base X
covs=["gross.square.feet","gross.square.feet.sq"];X=df[covs]
# La variable de respuesta se guarda en la variable y
resp=["sale.price"];y=df[resp]
# Inicializo el modelo y calculo los valores de a, b y c
mod=LinearRegression();mod.fit(X,y)
# Guardo la lista de coeficientes e imprimo sus valores
lis_coef=[[mod.intercept_],mod.coef_[0]]
print([np.round(float(c),2) for it in lis_coef for c in it])
```

```
## [430404.77, -68.91, 0.13]
```

Entonces, para una superficie de 2000 pies cuadrados, se estima un valor de

$$430404.77 - 68.91 \cdot 2000 + 0.13 \cdot 2000^2 = 812584.8$$

Además, se puede ver que aumenta el R^2 de este modelo respecto del modelo anterior

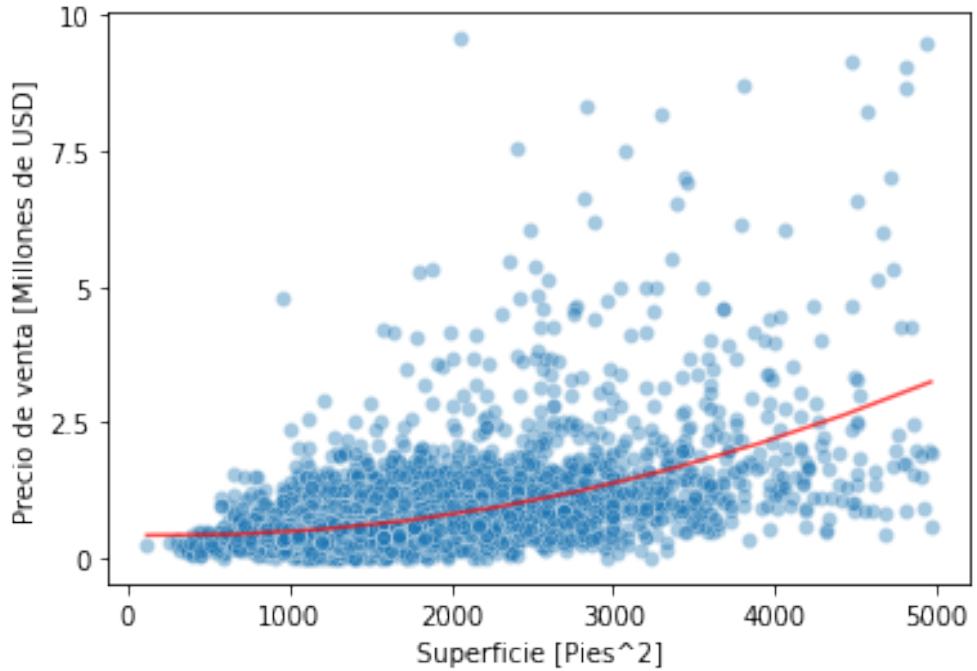
```
# El valor del R^2
print(mod.score(X,y))
```

```
## 0.2807952363411663
```

Gráfico modelo cuadrático

Además de aumentar el R^2, se puede ver gráficamente que las predicciones se ajustan más a los datos, ya que la representación es un poco más versátil que una recta:

```
# Tomamos valores entre el mínimo y el máximo con pasos de 10, agregamos los cuadrados y realizamos las predicciones
df_pred=pd.DataFrame({"gross.square.feet":np.linspace(minX,maxX,10)})
df_pred["gross.square.feet.sq"]=df_pred["gross.square.feet"]**2
df_pred["sale.price"]=mod.predict(df_pred)
# Graficamos los puntos y las predicciones se unen con líneas
sns.scatterplot(data=df,x="gross.square.feet",y="sale.price",alpha=0.4)
sns.lineplot(data=df_pred,x="gross.square.feet",y="sale.price",color="red",alpha=0.7)
plt.xlabel("Superficie [Pies^2]");plt.ylabel("Precio de venta [Millones de USD]")
plt.yticks(ticks=[0,0.25e7,0.5e7,0.75e7,1e7],labels=[0,2.5,5,7.5,10])
plt.show()
```



Regresión lineal múltiple

Regresión lineal múltiple

Notemos que en el código anterior, agregamos una nueva variable y se la pasamos como argumento al comando de regresión lineal, sin ningún tipo de reproche por parte de python.

Esto lleva a pensar lo siguiente: Dado que el mercado inmobiliario es muy complejo, tiene sentido que el precio de un inmueble no dependa únicamente de su superficie. ¿Podemos agregar más variables?

La respuesta es que sí, aunque siempre se asumirá que mantienen una estructura de suma. Por ejemplo, podemos plantear el siguiente modelo para cada inmueble i:

$$P_i \approx a + b \cdot S_i + c \cdot S_i^2 + c \cdot A_i + d \cdot A_i^2 + e \cdot L_i$$

donde A representa la antigüedad del inmueble, y L representa la superficie del lote.

En el caso en el que se utiliza más de una variable predictiva, se dice que se aplica un modelo de regresión lineal **múltiple**.

Estimación con Python

En python, seguimos los mismos pasos para obtener a, b, c, d y e:

```
# Elevamos la variable de antigüedad al cuadrado
df["age.sq"]=df["age"]**2
covs=["gross.square.feet","gross.square.feet.sq","age","land.square.feet","age.sq"]
X=df[covs]
resp=["sale.price"];y=df[resp]
mod=LinearRegression();mod.fit(X,y)
# Mostramos los coeficientes
lis_coef=[[mod.intercept_],mod.coef_[0]]
print([float(c) for it in lis_coef for c in it])
```

```
## [291449.36, -51.77, 0.13, 496.09, -4.82, 15.79]
```

Para realizar predicciones, deberíamos fijar valores de **todas** las variables. Es decir, para un inmueble de 2000 pies cuadrados, de 50 años de antigüedad y 3000 pies cuadrados de lote, se estima el siguiente precio:

$$291449.36 - 51.77 \cdot 2000 + 0.13 \cdot 2000^2 + 496.09 \cdot 50 - 4.82 \cdot 50^2 + 15.79 \cdot 3000 \approx 768033.9$$

Sin embargo, para un inmueble de 2000 pies cuadrados, de 50 años de antigüedad y 4000 pies cuadrados de lote (el único valor distinto), se estima el siguiente precio:

$$291449.36 - 51.77 \cdot 2000 + 0.13 \cdot 2000^2 + 496.09 \cdot 50 - 4.82 \cdot 50^2 + 15.79 \cdot 4000 \approx 783823.9$$

Es decir, con todos los valores iguales salvo uno, cambia el precio estimado.

Vemos además que aumenta aún más el valor de R^2

```
print(mod.score(X,y))
```

```
## 0.29772598628729485
```

Interpretación de los coeficientes

Cuando la regresión es múltiple, hay un detalle adicional a la hora de interpretar los coeficientes, ya que no puede pensarse cómo incrementa la estimación total si son muchas las variables involucradas.

Por ejemplo, en este caso, el coeficiente estimado correspondiente al tamaño del lote ($\hat{e}$) tiene un valor de 15.79. Esto significa que cada pie cuadrado adicional del tamaño del lote, agrega 15.79\$ al precio *estimado* de venta, **siempre y cuando las otras variables se mantengan constantes**.

Esta aclaración es clave ya que si además de aumentar el tamaño del lote, aumentan las otras variables, todas tienen un impacto en el precio estimado.

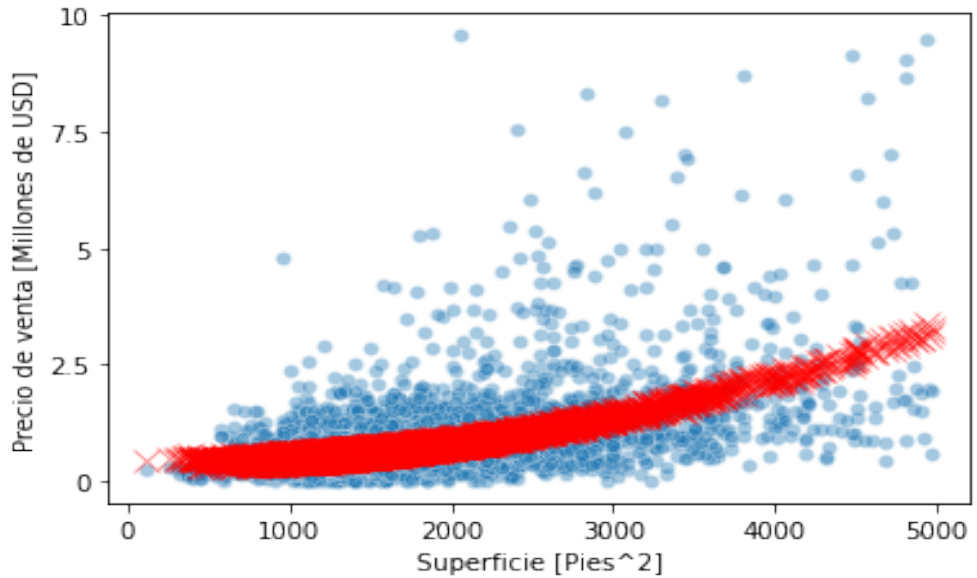
Evaluación gráfica

A diferencia de los casos anteriores, ahora se agrega una complejidad. Como vimos anteriormente, para dos inmuebles de igual superficie, pero distinta antigüedad por ejemplo, los precios de venta estimados pueden diferir.

Por lo tanto, ya no sirve sobreponer una única curva sobre los datos originales ya que las predicciones pueden no estar necesariamente conectadas entre sí. Además, tiene que estar la opción de agregar predicciones para **todas** las combinaciones posibles de las tres variables involucradas.

Por lo tanto, en vez de predecir algunos valores representativos, vamos a predecir **todos** los datos originales. Es decir, las variables que usamos para predecir las aplicamos en todas las instancias (sin repetición de combinaciones):

```
df_pred=X.drop_duplicates().copy()
df_pred["sale.price"]=mod.predict(df_pred)
sns.scatterplot(data=df,x="gross.square.feet",y="sale.price",alpha=0.4)
sns.scatterplot(data=df_pred,x="gross.square.feet",y="sale.price",color="red",alpha=0.5,marker="x",s=100)
plt.xlabel("Superficie [Pies^2]");plt.ylabel("Precio de venta [Millones de USD]")
plt.yticks(ticks=[0,0.25e7,0.5e7,0.75e7,1e7],labels=[0,2.5,5,7.5,10])
plt.show()
```



Comentarios

Notemos que en este caso, las estimaciones no están todas sobre la misma curva. Sobre todo en el medio, los puntos aparecen a distintas alturas para iguales valores de x. Justamente, esto se debe a que entran en juego las otras variables no disponibles en el gráfico (antigüedad y superficie del lote). Distintos valores de estas dos variables proveen distintas estimaciones y por lo tanto, tienen una "coordenada y" diferente.

Por otro lado, vale remarcar que esta variabilidad de las estimaciones también explica el incremento de R^2 con este modelo, ya que al tener más factores en cuenta, las estimaciones se pueden adaptar más a la heterogeneidad de los datos.

Variables categóricas

Variables categóricas

Hay un factor que no fue considerado en el modelo anterior que es de suma importancia en el mercado inmobiliario: la **ubicación** del inmueble.

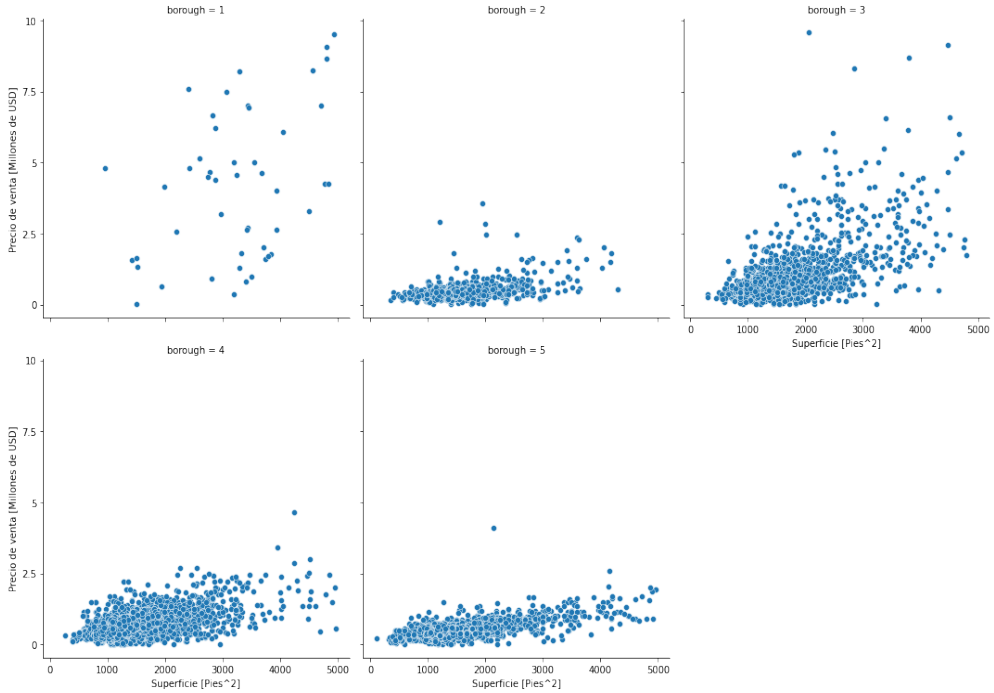
Respecto a la ubicación del inmueble, la base tiene las siguientes variables:

- D_i: el distrito o comuna de cada inmueble (denominada `'borough'` en la base de datos)
- B_i: el barrio del inmueble (denominada `'neighborhood'` en la base de datos)

Gráfico exploratorio

Primero inspeccionemos cómo son los precios de venta respecto de las superficies en cada distrito:

```
g=sns.relplot(x="gross.square.feet", y="sale.price", data=df, kind="scatter",
col="borough",col_wrap=3)
g.set(xlabel="Superficie [Pies^2]",ylabel="Precio de venta [Millones de USD]")
plt.yticks(ticks=[0,0.25e7,0.5e7,0.75e7,1e7],labels=[0,2.5,5,7.5,10])
plt.show()
```



Podemos ver que la estructura heterogénea de los datos originales se vuelve más homogénea cuando se individualiza cada distrito. Por lo que se ve en el gráfico, tendría sentido plantear un modelo que considere una recta distinta por cada distrito.

Regresión lineal con variables categóricas

Como dijimos anteriormente, una regresión lineal que considere (por ejemplo) la variable del distrito y la superficie, sería establecer la siguiente aproximación:

$$P_i \approx a + b \cdot S_i + c \cdot D_i$$

Sin embargo, surgen los siguientes inconvenientes:

- Al ser una variable categórica, ¿Cómo se "suma un distrito" numéricamente?
- Para "sumar" los valores de la variable "distrito", se puede establecer un código numérico para cada distrito (Distrito 1, Distrito 2, etc.). Sin embargo, este orden establecido es completamente arbitrario, podrían intercambiarse los valores (por ejemplo, puede cambiarse el Distrito 1, por el Distrito 2 y viceversa) y alterar todas las sumas que quisimos hacer previamente.

Categorización arbitraria

Respecto a este último inconveniente, la variable `'borough'` ya tiene un código similar, pero presenta la misma arbitrariedad que cualquier código numérico:

```
# Permuto los datos porque aparecen ordenados por barrio, la permutación permite ver distintos barrios y distritos en las
df=df.sample(frac=1,random_state=1).reset_index()
print("Base original:")
```

```
## Base original:
```

```
print(df[["borough","neighborhood"]].head())
# Restando 1, podría llamar al Distrito 4 como 3, al 2 como 1, etc.
# Eso alteraría los valores que se utilizan en la suma.
```

```
##    borough         neighborhood
## 0        5          TOTTENVILLE
## 1        5      MARINERS HARBOR
## 2        4        RICHMOND HILL
## 3        2            RIVERDALE
## 4        2  MORRISANIA/LONGWOOD
```

```
df["borough"]=df["borough"]-1
print("Base con nuevos códigos:")
```

```
## Base con nuevos códigos:
```

```
print(df[["borough","neighborhood"]].head())
# Vuelvo al código original volviendo a sumar 1
```

```
##    borough         neighborhood
## 0        4          TOTTENVILLE
## 1        4      MARINERS HARBOR
## 2        3        RICHMOND HILL
## 3        1            RIVERDALE
## 4        1  MORRISANIA/LONGWOOD
```

```
df["borough"]=df["borough"]+1
```

Por lo tanto, hay que tener cierto cuidado a la hora de incluir variables categóricas en un modelo de regresión.

Variables dummies

Para utilizar un criterio coherente, se crea una columna por cada nivel de la variable categórica que se quiere utilizar. Estas columnas agregadas se llaman "variables dummies" y sólo toman dos valores: 1 y 0. En python, se genera con el método "get_dummies".

```
# Creamos una nueva base df1 con las variables dummies para la variable "borough"
df1=pd.get_dummies(df,columns=["borough"])
# Mostramos los nombres de las columnas de la base original y en la base nueva
print("Columnas base original:")
```

```
## Columnas base original:
```

```
print(df.columns.values)
```

```
## ['index' 'time.index' 'borough' 'neighborhood' 'building.class.category'
##  'zip.code' 'residential.units' 'commercial.units' 'total.units'
##  'land.square.feet' 'gross.square.feet' 'year.built' 'sale.price'
##  'sale.date' 'sale.year' 'age' 'gross.square.feet.sq']
```

```
print("Columnas base nueva:")
```

```
## Columnas base nueva:
```

```
print(df1.columns.values)
```

```
## ['index' 'time.index' 'neighborhood' 'building.class.category' 'zip.code'
##  'residential.units' 'commercial.units' 'total.units' 'land.square.feet'
##  'gross.square.feet' 'year.built' 'sale.price' 'sale.date' 'sale.year'
##  'age' 'gross.square.feet.sq' 'borough_1' 'borough_2' 'borough_3'
##  'borough_4' 'borough_5']
```

Se ve que en la nueva base desaparece la variable `'borough'` y aparecen 5 nuevas variables: `'borough_1'`, `'borough_2'`, `'borough_3'`, `'borough_4'`, `'borough_5'`. Es decir, se eliminó

Variables dummies

Veamos cómo se genera esta transformación y cómo se codifica la información:

```
print("Base original:")
```

```
## Base original:
```

```
print(df[["borough","neighborhood"]].head())
```

```
##    borough         neighborhood
## 0        5          TOTTENVILLE
## 1        5      MARINERS HARBOR
## 2        4        RICHMOND HILL
## 3        2            RIVERDALE
## 4        2  MORRISANIA/LONGWOOD
```

```
covsB=[dV for dV in df1.columns.values if dV.startswith("borough_")]
print("Base con dummies:")
```

```
## Base con dummies:
```

```
print(df1[covsB].head())
```

```
##    borough_1  borough_2  borough_3  borough_4  borough_5
## 0          0          0          0          0          1
## 1          0          0          0          0          1
## 2          0          0          0          1          0
## 3          0          1          0          0          0
## 4          0          1          0          0          0
```

Modelo adaptado a variables categóricas

Según el gráfico exploratorio tenía sentido considerar una recta por cada distrito. Por lo tanto, debería haber un intercept (notado con a) y una pendiente (notado con b) para cada distrito (por ejemplo, a_1 y b_1 representan la ordenada y la pendiente de la recta correspondiente al distrito 1), dando lugar a 10 constantes desconocidas que debemos estimar. Es decir:

$$P_i \approx a_1 + b_1 \cdot S_i \text{ (si el inmueble es del Distrito 1)}$$

$$P_i \approx a_2 + b_2 \cdot S_i \text{ (si el inmueble es del Distrito 2)}$$

$$P_i \approx a_3 + b_3 \cdot S_i \text{ (si el inmueble es del Distrito 3)}$$

$$P_i \approx a_4 + b_4 \cdot S_i \text{ (si el inmueble es del Distrito 4)}$$

$$P_i \approx a_5 + b_5 \cdot S_i \text{ (si el inmueble es del Distrito 5)}$$

Estimación con Python

Para generar estas rectas específicas por distrito, debemos basarnos en las variables dummies, generando otras columnas multiplicando por la variable de superficie.

```
df1=pd.get_dummies(df,columns=["borough"])
# Los nombres de las variables dummies
covsB=[dV for dV in df1.columns.values if dV.startswith("borough_")]
# Los nombres de las nuevas variables de superficie multiplicadas por las va
covsBgsf=[dV+"_gsf" for dV in covsB]
for i in range(5):
  df1[covsBgsf[i]]=df1[covsB[i]]*df1["gross.square.feet"]
```

Estimación con Python

Vemos que estas nuevas columnas guardan la información por distrito:

```
print("Datos originales de superficie:")
```

```
## Datos originales de superficie:
```

```
print(df[["borough","gross.square.feet"]].head())
```

```
##    borough  gross.square.feet
## 0        5               2600
## 1        5                935
## 2        4                975
## 3        2               2000
## 4        2               1152
```

```
covsB=[dV for dV in df1.columns.values if dV.startswith("borough_")]
print("Base con dummies:")
```

```
## Base con dummies:
```

```
print(df1[covsB].head())
```

```
##    borough_1  borough_2  borough_3  ...  borough_3_gsf  borough_4_gsf  borough_5_gsf
## 0          0          0          0  ...              0              0           2600
## 1          0          0          0  ...              0              0            935
## 2          0          0          0  ...              0            975              0
## 3          0          1          0  ...              0              0              0
## 4          0          1          0  ...              0              0              0
##
## [5 rows x 10 columns]
```

Notar que para los inmuebles del distrito 5, se obtienen valores distinto de cero en las columnas correspondientes al distrito 5. Más aún, la variable 'borough_5_gsf' contiene los valores correspondientes a la superficie de cada inmueble de dicho distrito. Lo mismo ocurre con el resto de los distritos.

Estimación de los parámetros

Con estas nuevas variables, podemos generar el modelo deseado:

```
X=df1[covsB].copy()
y=df1["sale.price"].copy()
mod=LinearRegression(fit_intercept=False);mod.fit(X,y)
```

```
## LinearRegression(copy_X=True, fit_intercept=False, n_jobs=None, normalize
```

Calculamos el R^2:

```
print(mod.score(X,y))
```

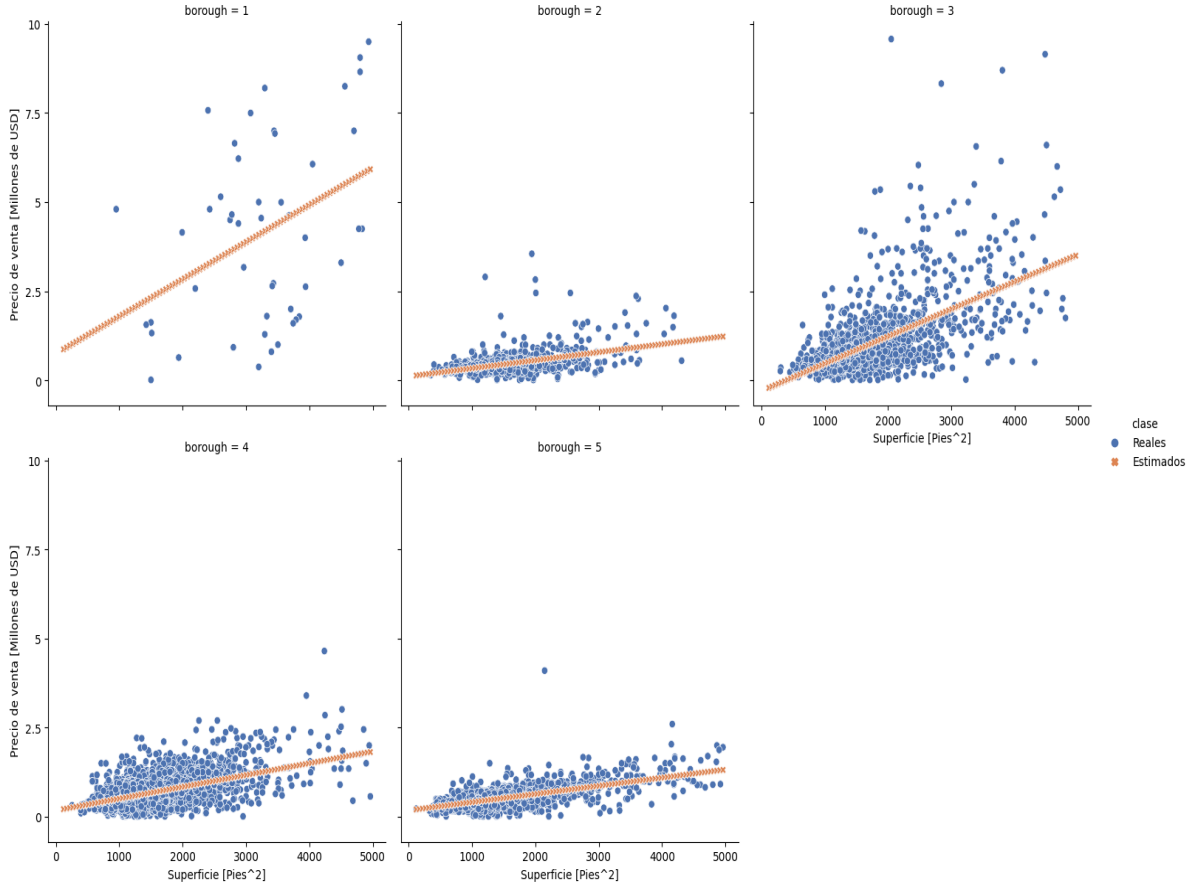
```
## 0.4718757976364857
```

Aunque no llega a valores muy altos, vemos que el R^2 aumenta considerablemente respecto de los modelos anteriores. Esto se da porque las predicciones se adaptan mejor a la heterogeneidad de los datos.

Evaluación gráfica

Se pueden utilizar los coeficientes para adaptar las predicciones a distintas rectas para cada distrito.

```
g=sns.relplot(x="gross.square.feet", y="sale.price", data=df1, kind="scatter",
col="borough",col_wrap=3,hue="clase",style="clase",markers={"Reales":"o","Estimados":"X"},palette="deep")
g.set(xlabel="Superficie [Pies^2]",ylabel="Precio de venta [Millones de USD]")
plt.yticks(ticks=[0,0.25e7,0.5e7,0.75e7,1e7],labels=[0,2.5,5,7.5,10])
plt.show()
```



Overfitting y capacidad predictiva

Como se habló previamente, una vez que se estiman los coeficientes, uno de los objetivos de estos modelos es intentar predecir el precio de venta de nuevos inmuebles a partir de los inmuebles ya observados.

Pero esas predicciones dependen de dos cuestiones:

- Que el nuevo inmueble tenga similitud con los datos utilizados para la estimación.
- La heterogeneidad de los datos utilizados para la estimación. Mientras más casos estén contemplados, mejores serán las predicciones para nuevas observaciones.

Por ejemplo, se ve que los distritos 1 y 5 tienen relaciones muy distintas entre la superficie y su precio de venta. Entonces, si utilizamos los datos del distrito 5 para estimar los valores de a y b, esos valores no serán representativos de los correspondientes al distrito 1.

Overfitting y capacidad predictiva

Veamos esto en el siguiente ejemplo:

```
# Base restringida al distrito 1
df_b1=df.loc[df["borough"]==1].copy()
print(df_b1.head())
```

```
##       index  time.index  borough  ... sale.year  age  gross.square.feet.sq
## 511    3207         231        1  ...      2018  119              22118209
## 662    3191         130        1  ...      2018  117              23348224
## 1009   3195         309        1  ...      2017  107              11778624
## 1014   3211          32        1  ...      2018  103              11847364
## 1145   3216         358        1  ...      2017  118               8294400
##
## [5 rows x 17 columns]
```

```
# Base restringida al distrito 5
df_b5=df.loc[df["borough"]==5].copy()
print(df_b5.head())
```

```
##     index  time.index  borough  ... sale.year age  gross.square.feet.sq
## 0   12157         278        5  ...      2017  23               6760000
## 1   10822         318        5  ...      2017  27                874225
## 5    9522         346        5  ...      2017  37               1157776
## 9    9323         309        5  ...      2017  17               3422500
## 12   9695         183        5  ...      2018  28               3869089
##
## [5 rows x 17 columns]
```

Overfitting y capacidad predictiva

Supongamos que utilizamos los datos del distrito 5 para estimar los coeficientes de la recta.

```
X_train=df_b5[["gross.square.feet"]].copy()
y_train=df_b5["sale.price"].copy()
mod=LinearRegression()
mod.fit(X_train,y_train)
```

```
## LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=
```

```
print([mod.intercept_,float(mod.coef_)])
```

```
## [175724.46669097984, 229.19300465977642]
```

Vemos además que el **mismo modelo** planteado para los datos totales, presenta un mejor R^2 al restringirse sólo a los datos del distrito 5:

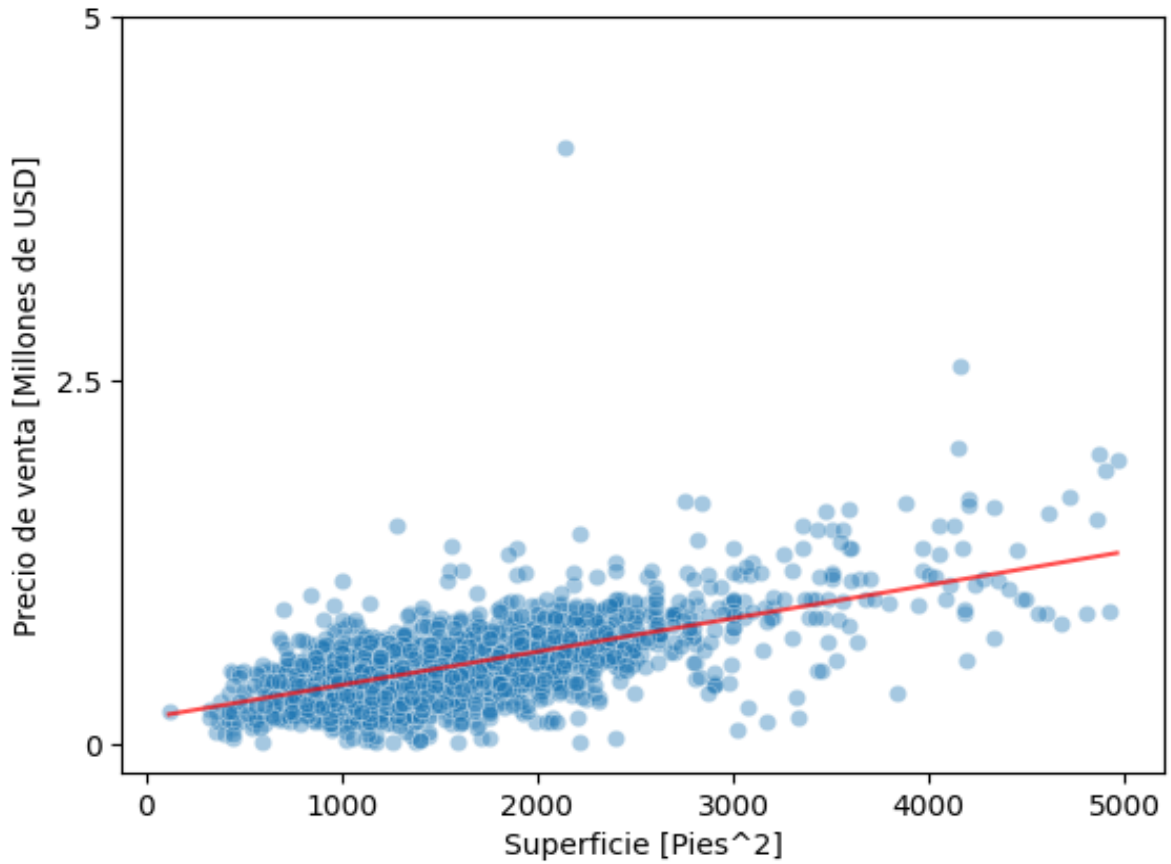
```
print(mod.score(X_train,y_train))
```

```
## 0.3787195552348216
```

Overfitting y capacidad predictiva

Vemos además gráficamente que la recta se ajusta mucho mejor a los datos cuando son específicos al distrito 5, respecto de la base total:

```
minX=float(X_train.min())
maxX=float(X_train.max())
df_pred=pd.DataFrame({"gross.square.feet":[minX,maxX]})
df_pred["sale.price"]=mod.predict(df_pred)
sns.scatterplot(x="gross.square.feet",y="sale.price",data=df_b5,alpha=0.4)
sns.lineplot(x="gross.square.feet",y="sale.price",data=df_pred,color="red",alpha=0.7)
plt.xlabel("Superficie [Pies^2]")
plt.ylabel("Precio de venta [Millones de USD]")
plt.yticks(ticks=[0,0.25e7,0.5e7],labels=[0,2.5,5])
plt.show()
```

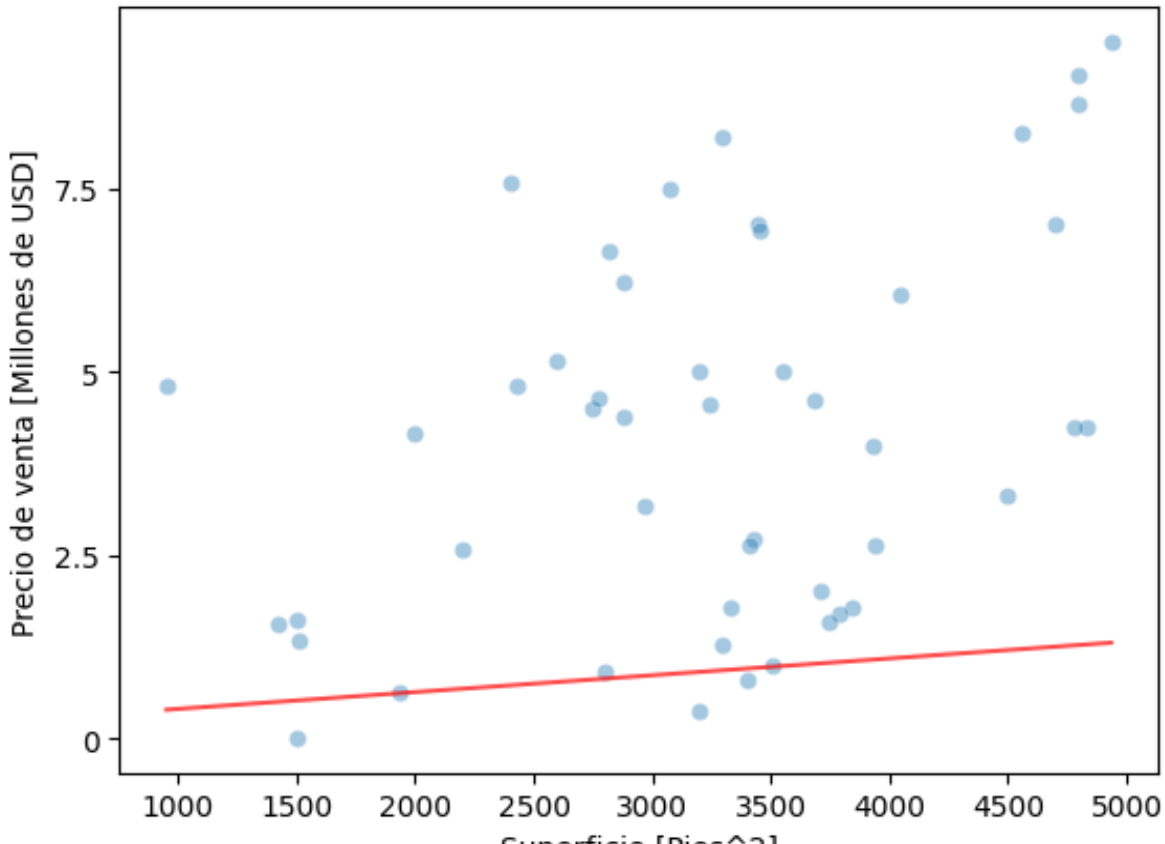


Overfitting y capacidad predictiva

Sin embargo, veamos qué sucede cuando aplicamos estas predicciones a los inmuebles del distrito 1:

```
X_test=df_b1[["gross.square.feet"]].copy()
y_test=df_b1[["sale.price"]].copy()
y_pred=mod.predict(X_test)
df_pred=X_test.copy()
df_pred["sale.price"]=y_pred

sns.scatterplot(x="gross.square.feet",y="sale.price",data=df_b1,alpha=0.4)
sns.lineplot(x="gross.square.feet",y="sale.price",data=df_pred,color="red",alpha=0.7)
plt.xlabel("Superficie [Pies^2]")
plt.ylabel("Precio de venta [Millones de USD]")
plt.yticks(ticks=[0,0.25e7,0.5e7,0.75e7],labels=[0,2.5,5,7.5])
plt.show()
```



Particularidad

Recordemos que el R^2 se utilizaba para medir cuán bien un cierto modelo representa a los datos. Se puede utilizar la misma estrategia para evaluar la capacidad predictiva de un modelo, para ver cómo se ajusta a nuevas observaciones. En este caso, el valor de R^2 da resultado negativo.

```
print(mod.score(X_test,y_test))
```

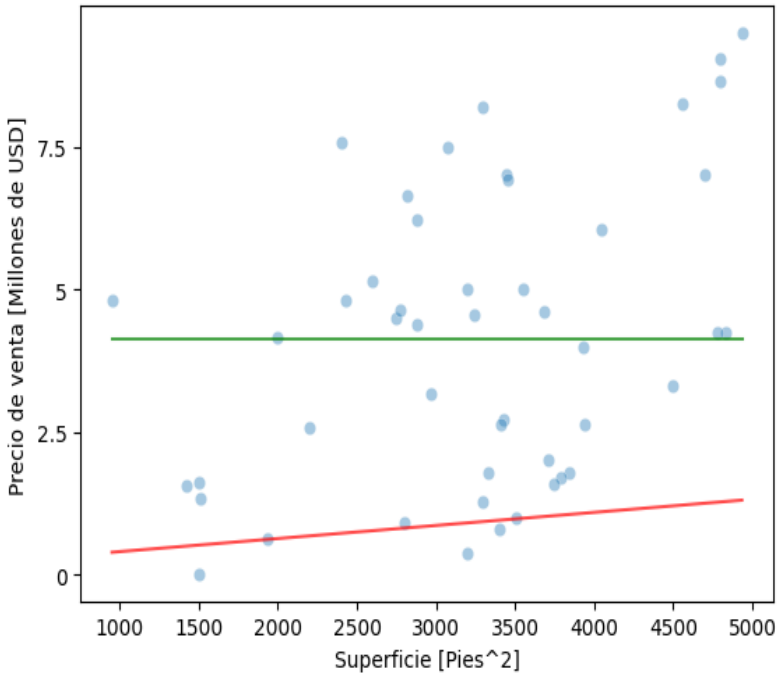
```
## -1.5071647243579966
```

Esto parece contradecir lo dicho anteriormente sobre los valores que puede tomar esta medida (debe estar entre 0 y 1). Sin embargo, a veces se utiliza una corrección para calcular el R^2 que en casos de rectas muy poco representativas, puede dar negativo (más aún cuando se utilizan pocas observaciones, como es este caso).

Particularidad

Este particular caso ocurre cuando la regresión arroja predicciones que son peores que predecir con su propia media (Ver https://stats.stackexchange.com/questions/183265/what-does-negative-r-squared-mean), como vemos en la siguiente figura:

```
df_pred["avg"]=[float(np.mean(y_test))]*int(y_test.shape[0])
sns.scatterplot(x="gross.square.feet",y="sale.price",data=df_b1,alpha=0.4)
sns.lineplot(x="gross.square.feet",y="sale.price",data=df_pred,color="red",alpha=0.7)
sns.lineplot(x="gross.square.feet",y="avg",data=df_pred,color="green",alpha=0.7)
plt.xlabel("Superficie [Pies^2]")
plt.ylabel("Precio de venta [Millones de USD]")
plt.yticks(ticks=[0,0.25e7,0.5e7,0.75e7],labels=[0,2.5,5,7.5])
plt.show()
```



Overfitting

Este es un claro ejemplo del fenómeno llamado "overfitting", en el que busca explicar con tanta precisión un conjunto de datos, que falla al predecir los resultados de datos nuevos. Por lo tanto, todo modelo debe buscar un equilibrio entre ambas capacidades:

- Un modelo con buenas capacidades explicativas permite representar los datos que se utilizaron en la estimación.
- Un modelo con buenas capacidades predictivas permite estimar correctamente las respuestas correspondientes a observaciones nuevas.

En este caso, para evitar el overfitting, la base inicial debe contener datos de todos los distritos.

Más en general, se debe tener bien claro qué características tienen las observaciones utilizadas para estimar, para saber en qué casos se pueden extrapolar dichos resultados.

Descripción gráfica del overfitting