

# Predicción de precios de Airbnb en New York



# Introducción

---

Analizamos la base de datos de Airbnb de NY y propusimos un modelo que realizara predicciones del precio de renta por habitación.

La base de datos originalmente contaba con 48,885 observaciones y 16 variables. Después del proceso de limpieza de los datos nos quedamos con 38,811 observaciones y 11 variables.

Adicionalmente, exploramos la técnica de **feature hashing** para manejar variables categóricas que tenían muchas clases.

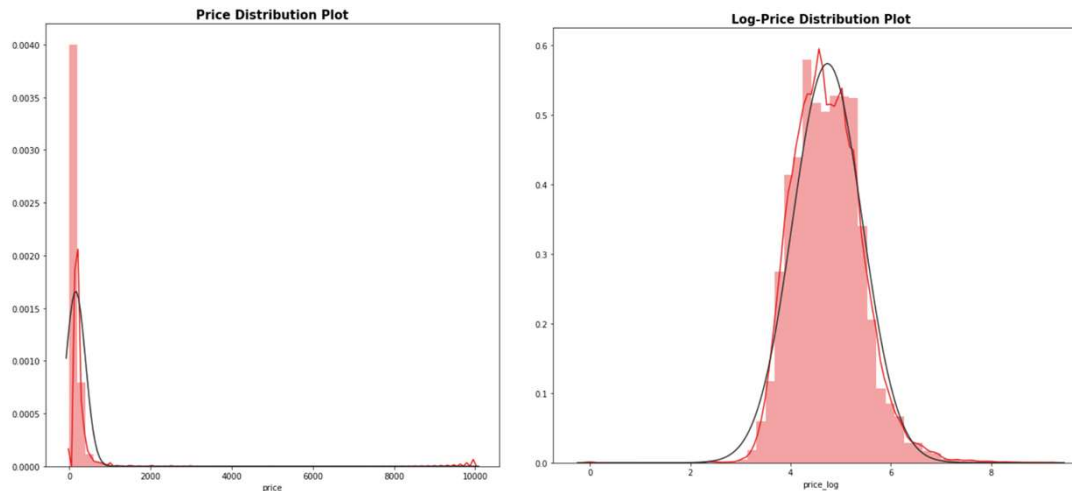


# Descripción de la base de datos

Columna	Tipo de la variable	Descripción de la variable
id	Categórica A-Z	ID de la propiedad
name	Categórica A-Z	Nombre de la propiedad
host_id	Categórica A-Z	Id del Propietario
host_name	Categórica A-Z	Nombre del anfitrión
neighbourhood_group	Categórica A-Z	Delegación
Neighbourhoodarea	Categórica A-Z	Colonia
latitude	Numérica #	Coordenadas de latitud
longitude	Numérica #	Coordenadas de longitud
room_type	Categórica	Tipo de cuarto (1 cuarto, casa entera, otro)
price	Numérica #	precio en dólares
minimum_nights	Numérica #	Cantidad mínima de días necesarios para apartar la habitación
number_of_reviews	Numérica #	Número de reseñas de los huéspedes
last_review	Fecha-categórica 🕒	Fecha en que se realizó la ultima reseña
reviews_per_month	Numérica #	Número de reseñas por mes
calculated_host_listings	Numérica #	Número de propiedades de Airbnb que tiene el anfitrión
availability_365	Numérica #	Número de días al año que la propiedad está disponible para ser alquilada

# Particularidades del problema








- Se usa  **$\log(\text{precio}+1)$**  para modelar el precio porque la distribución del precio está sesgada a la izquierda, y para aproximarla a una normal hacemos la transformación de la variable con  $\log(\text{precio} + 1)$  ya que no existe logaritmo de 0.



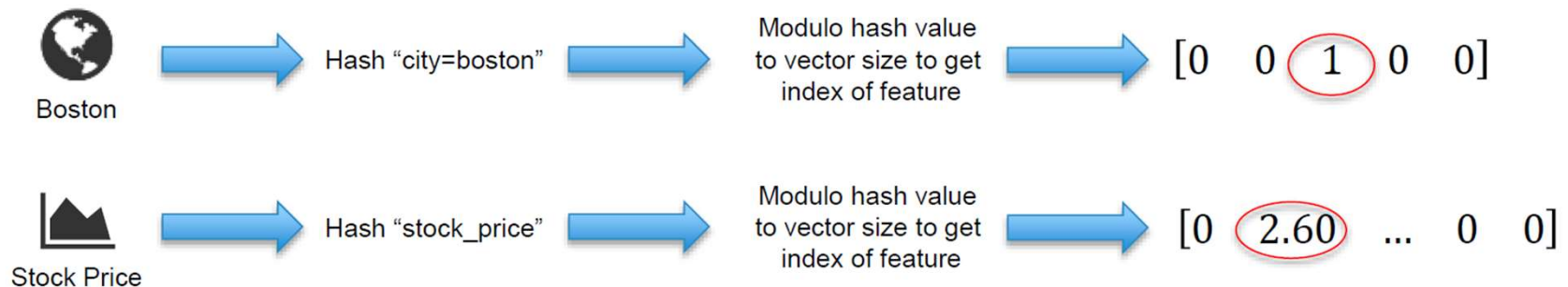
- Se **decidió** usar el **host\_id** como **variable categórica**, lo cual aumenta en gran manera la dimensionalidad del problema. Para mitigar esto usamos la técnica de **feature hashing**.



# Feature Hashing

        
[1, ..., 1, ..., 1, ..., 1, ..., 1, ..., 3.14, ..., 1, ...]

- ¿Qué es? Es una técnica de reducción de dimensionalidad donde se usa una función hash para mapear los valores de los features a índices en un vector de features.
- ¿Cuándo se usa y para qué? Con “high dimensional input data” que es “sparse”.
- **Ventajas:** uso eficiente de memoria, rápido y simple, conserva el *sparsity*, fácil manejo de datos faltantes, ingeniería de características
- **Desventajas:** no hay *inverse mapping*, mala interpretabilidad y características importantes, *hash collisions*, impacto en la precisión de *hash collisions*



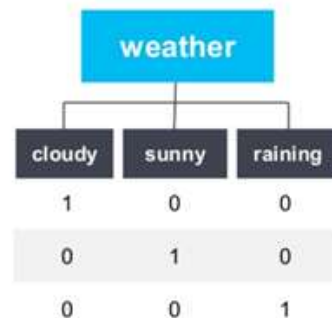
# El truco del Hashing

Se usa una función hash para mapear los valores de los features a índices en un vector de features

## One-hot encoding

day	temperature [F]	weather	bike rentals
3	76	Cloudy	543
4	72	Raining	173
5	78	Sunny	674
6	68	Raining	124

day	temperature [F]	cloudy	sunny	raining	bike rentals
3	76	1	0	0	543
4	72	0	0	1	173
5	78	0	1	0	674
6	68	0	0	1	124



## NLP: Vectorization

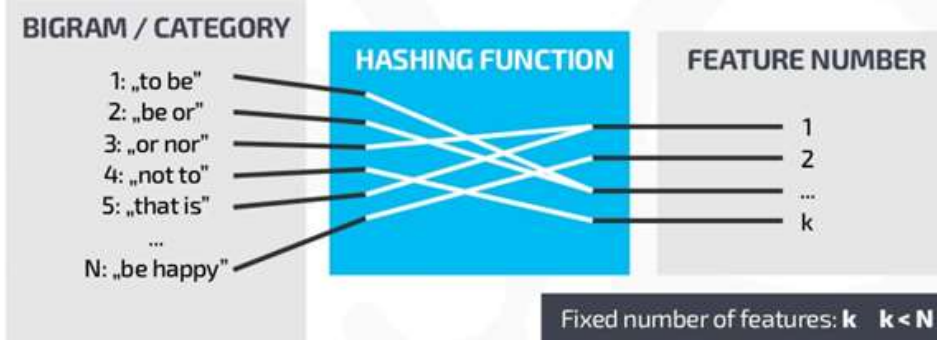
id	message
123	"be happy"
321	"To be or not to be"



id	"be"	"happy"	"be happy"	"to"	"or"	"not"	"to be"	"be or"	"or not"	"not to"
123	1	1	1	0	0	0	0	0	0	0
321	1	0	0	1	1	1	1	1	1	1

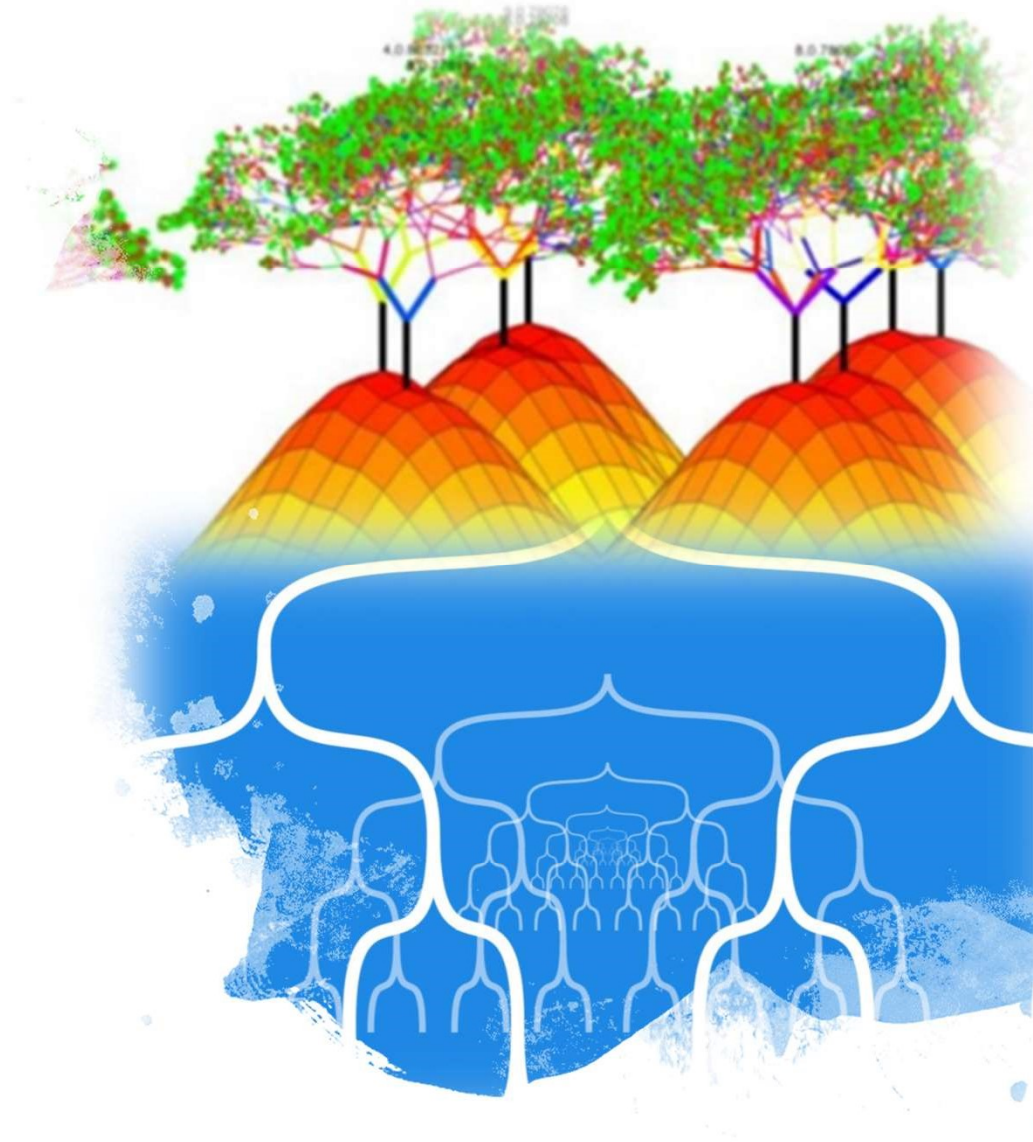
VS

## Hash trick



# Modelos propuestos

- Modelo de regresión lineal
- Modelo de regresión lineal con regularización Ridge
- Bosque Aleatorio
- XGBoost



# Resultados

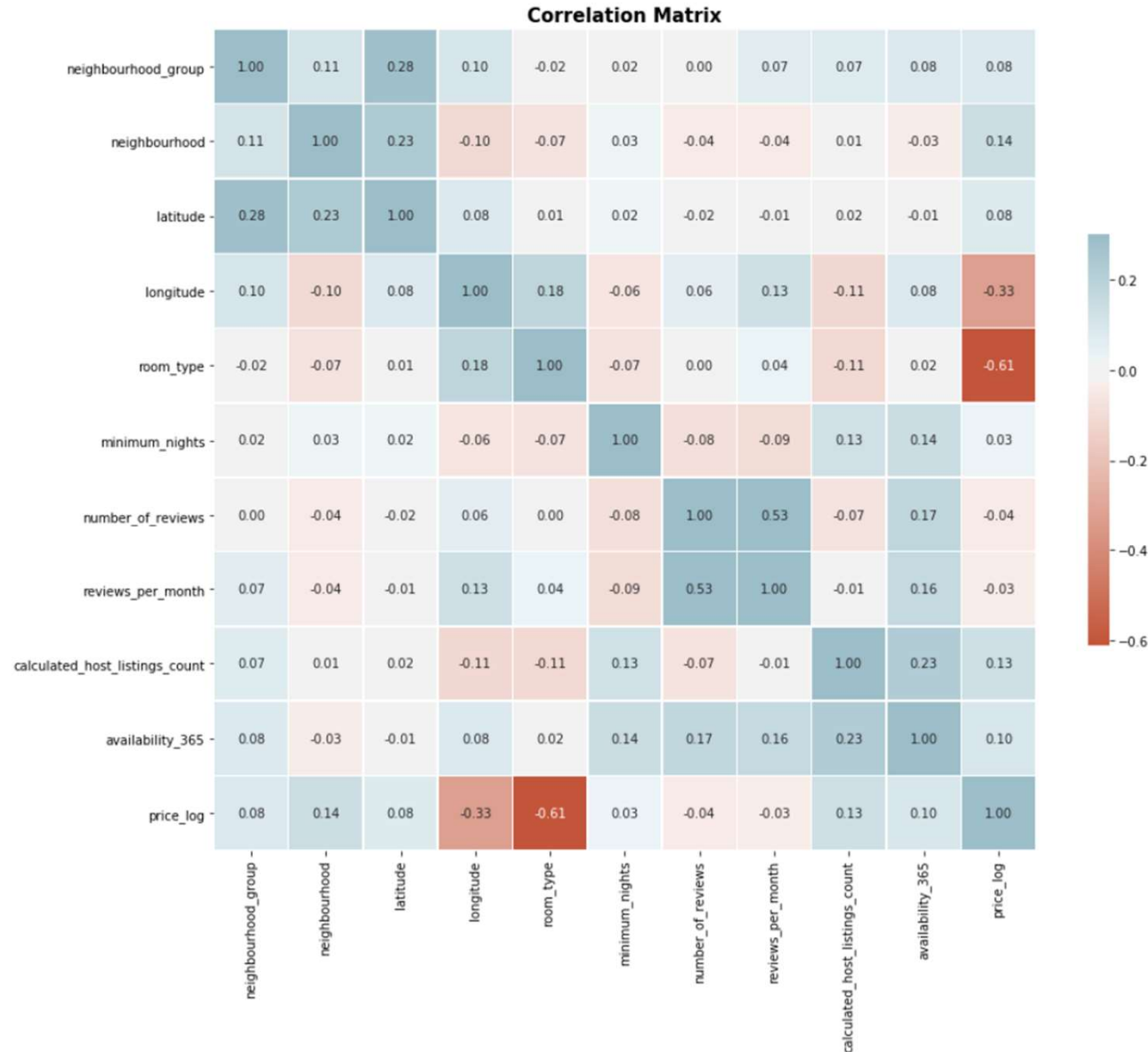
Sin host_id			
Modelo	Conjunto	rmse	r2_score
Regresión	Entrenamiento	0.427846	0.5727581
Regresión	Validación	0.441220	0.5629389
Regresión	Prueba	0.431223	0.5628377
Ridge	Entrenamiento	0.427971	0.5725088
Ridge	Validación	0.440941	0.5634912
Ridge	Prueba	0.431182	0.5629220
Bosque Aleatorio	Entrenamiento	0.240415	0.8650964
Bosque Aleatorio	Validación	0.409476	0.6235672
Bosque Aleatorio	Prueba	<b>0.401028</b>	<b>0.6219172</b>
XGBoost	Entrenamiento	0.389560	0.6457999
XGBoost	Validación	0.418865	0.6061066
XGBoost	Prueba	0.406794	0.6109663

Con host_id			
Modelo	Conjunto	rmse	r2_score
Regresión**	Entrenamiento	0.432760158	0.562886901
Regresión**	Validación	0.449889826	0.545594217
Regresión**	Prueba	0.441625728	0.541491583
Ridge**	Entrenamiento	0.432819305	0.562767409
Ridge**	Validación	0.450041648	0.545287473
Ridge**	Prueba	0.441686858	0.541364639
Bosque Aleatorio	Entrenamiento	0.226554006	0.880203874
Bosque Aleatorio	Validación	0.407512681	0.627167501
Bosque Aleatorio	Prueba	<b>0.399814423</b>	<b>0.624201109</b>



# Conclusiones

- La base de datos puede ser complementada con información adicional que ayudaría a mejorar las predicciones.
- Hay un área de oportunidad para reducir aún mas el error de predicción a través de feature engineering, que por restricciones de tiempo no exploramos a fondo.
- La técnica de feature hashing nos permitió probar modelos con variables categóricas (host\_id) que de otra forma no hubiera sido posible.



# Referencias

---

<https://www.kaggle.com/duygut/airbnb-nyc-price-prediction>

[https://www.slideshare.net/Hadoop\\_Summit/machine-learning-on-hadoop-data-lakes](https://www.slideshare.net/Hadoop_Summit/machine-learning-on-hadoop-data-lakes)

<https://www.slideshare.net/SparkSummit/feature-hashing-for-scalable-machine-learning-spark-summit-east-talk-by-nick-pentreath/8>

¡Gracias!



```
-----Lasso-----  
--Phase-1--  
MAE: 0.375922  
RMSE: 0.520400  
R2 0.530591  
--Phase-2--  
MAE: 0.523562  
RMSE: 0.671290  
R2 0.218595  
-----ElasticNet-----  
--Phase-1 --  
MAE: 0.371707  
RMSE: 0.518862  
R2 0.533362  
--Phase-2--  
MAE: 0.524883  
RMSE: 0.670878  
R2 0.219553
```

```
-----Linear Regression-----  
--Phase-1--  
MAE: 0.377923  
RMSE: 0.522021  
R2 0.527663  
--Phase-2--  
MAE: 0.531963  
RMSE: 0.685894  
R2 0.184227  
-----Ridge -----  
--Phase-1--  
MAE: 0.377915  
RMSE: 0.522038  
R2 0.527631  
--Phase-2--  
MAE: 0.529255  
RMSE: 0.679340  
R2 0.199742
```

# Benchmark mejor resultado en Kaggle