Proyecto Final
Predicción de precio en propiedades de Airbnb

Mario Emilio Jiménez Vizcaíno
A01173359@itesm.mx
Ingeniería en Tecnologías Computacionales
Tecnológico de Monterrey

Franco Daniel Pérez Reyes
A00822080@itesm.mx
Ingeniería en Mecatrónica
Tecnológico de Monterrey

Alberto Dávila Almaguer
A01196987@itesm.mx
Ingeniería en Innovación y Desarrollo
Tecnológico de Monterrey

Carlos Andrés Luna Leyva
A00826148@itesm.mx
Ingeniería en Tecnologías Computacionales
Tecnológico de Monterrey

ABSTRACT

En los últimos años han surgido plataformas en las cuales anfitriones pueden publicitar y contratar el arriendo de sus propiedades con sus huéspedes; anfitriones y huéspedes pueden valorarse mutuamente, como referencia para futuros usuarios. Una de las primeras, y la más exitosa en Estados Unidos es AirBnB, fundada en el año 2008. En este proyecto se utilizarán diversas técnicas de aprendizaje automático con el fin de predecir y optimizar una serie de métricas sobre propiedades, utilizando un dataset de más de 70 mil propiedades listadas entre 2008 y 2017. Ésto con el fin de identificar y estudiar las características más importantes de una propiedad, desde el punto de vista de un anfitrión y de un huésped.

1 INTRODUCCIÓN

El precio de cualquier producto o servicio se ve afectado por variables como oferta, demanda, inflación en el país, etc. Para bienes inmuebles esto es mucho más complejo, ya que entran en consideración factores como la zona donde se encuentra la propiedad, el espacio total que abarca, el número de habitaciones, etc. Cada una de estas variables puede o no afectar a las demás y juntas caracterizan un sistema digno de ser analizado.

El análisis y extracción de conocimiento de sistemas complejos ha sido uno de los objetivos principales del aprendizaje automático, la rama de los algoritmos computacionales cuyo objetivo es emular la inteligencia humana a través de "aprender" del entorno. Las técnicas basadas en el aprendizaje automático se han aplicado con éxito en diversos campos como el reconocimiento de patrones, la visión computacional, diferentes ramas de la ingeniería, las finanzas y la biología[2].

Durante el desarrollo de este proyecto se explorará la capacidad que tienen los algoritmos de regresión para poder predecir variables de interés, que forman parte de sistemas difíciles de modelar: en este estudio, el precio de quedarse una noche en una propiedad.

2 CONCEPTOS PREVIOS

- Conocimientos básicos de estadística
- Conocimientos básicos sobre el aprendizaje automático y los modelos de regresión
- Programación básica en el lenguaje Python
- Conocimientos sobre librerías como `scikit-learn`, `pandas` y `numpy`

3 METODOLOGÍA

3.1 Introducción del dataset

Los datos seleccionados son un subconjunto de las casas o departamentos listados en Airbnb desde el 17 de noviembre de 2008 (la fecha de la primera reseña) hasta el 5 de octubre de 2017 (la fecha de la última reseña) en una de seis ciudades: Boston, Chicago, Washington DC, Los Angeles, New York City y San Francisco.

El dataset fue recuperado del sitio web Kaggle, específicamente de este post llamado "Airbnb price prediction" https://www.kaggle.com/stevezhenghp/airbnb-price-prediction. Este dataset fue originalmente nombrado así ya que se esperaba que fuera utilizado para predecir la columna "log_price" que representa el logaritmo del precio original de la propiedad. Por nuestra parte, intentaremos que nuestro análisis sea más profundo que solamente predecir el precio.

El dataset se compone de alrededor de 74,100 instancias cada una con las siguientes propiedades:

- **id**, un número arbitrario asignado a la casa o departamento
- **log_price**, el logaritmo del precio original de la casa o departamento
- **property_type**, una de 35 cadenas de caracteres que representan el tipo de propiedad
- **room_type**, una de 3 cadenas que representa el tipo de cuarto a rentar
- **amenities**, una lista en formato JSON que representa los servicios con los que cuenta la propiedad
- **accommodates**, el número de personas para los que está diseñada la propiedad
- **bathrooms**, el número de baños en la propiedad
- **bed_type**, el tipo de cama rentada
- **cancellation_policy**, una de 5 cadenas de caracteres que representa qué tipo de póliza se aplica en caso de que el cliente decida cancelar una renta (que su dinero sea devuelto o no)
- **cleaning_fee**, un booleano, verdadero si la renta de la propiedad incluye un cargo por su limpieza
- **city**, una cadena de caracteres que representa la ciudad en la que se encuentra la propiedad
- **description**, texto, la descripción de la propiedad
- **first_review**, la fecha en la que se registró la primera reseña de la propiedad

- **host_has_profile_pic**, un booleano que representa si el anfitrión tiene una foto en su perfil de Airbnb
- **host_identity_verified**, un booleano que representa si el anfitrión ha verificado su identidad con documentos oficiales
- **host_response_rate**, la tasa en la que el anfitrión responde a solicitudes de rentar su propiedad
- **host_since**, la fecha de registro del anfitrión en la plataforma
- **instant_bookable**, un booleano que representa si la propiedad se puede rentar automáticamente (verdadero) o si es necesario que el anfitrión apruebe la renta (falso)
- **last_review**, la fecha en la que se registró la última reseña
- **latitude**, la latitud de las coordenadas de la propiedad
- **longitude**, la longitud de las coordenadas de la propiedad
- **name**, el nombre de la propiedad en Airbnb
- **neighbourhood**, el nombre de la colonia o vecindario en el que se encuentra la propiedad
- **number_of_reviews**, el número de reseñas que le han hecho a la propiedad
- **review_scores_rating**, la calificación promedio de las reseñas, entre 0 y 100
- **thumbnail_url**, la liga de internet a la imagen de la propiedad
- **zipcode**, el código postal de la propiedad
- **bedrooms**, el número de habitaciones en la propiedad
- **beds**, el número de camas en la propiedad

3.2 Análisis estadístico

Para comenzar el análisis del dataset lo primero que se verificó fue que el dataset no tuviera valores faltantes y se obtuvo la información descriptiva de los dos archivos csv, esto con la función `pd.DataFrame.info()`.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 72830 entries, 6901257 to 3534845
Data columns (total 46 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   log_price                    72830 non-null  float64
 1   property_type                72830 non-null  object
 2   room_type                    72830 non-null  object
 3   accommodates                 72830 non-null  int64
 4   bathrooms                    72830 non-null  float64
 5   cleaning_fee                 72830 non-null  bool
 6   city                         72830 non-null  object
 7   host_has_profile_pic         72830 non-null  bool
 8   host_identity_verified       72830 non-null  bool
 9   host_since                   72830 non-null  object
 10  instant_bookable             72830 non-null  bool
 11  latitude                     72830 non-null  float64
 12  longitude                    72830 non-null  float64
 13  number_of_reviews            72830 non-null  int64
 14  review_scores_rating         72830 non-null  float64
 15  zipcode                      72830 non-null  int64
 16  bedrooms                     72830 non-null  int64
 17  beds                         72830 non-null  int64
 18  Wireless_Internet            72830 non-null  bool
 19  Kitchen                      72830 non-null  bool
 20  Heating                      72830 non-null  bool
 21  Essentials                   72830 non-null  bool
 22  Smoke_detector               72830 non-null  bool
 23  Air_conditioning             72830 non-null  bool
 24  TV                           72830 non-null  bool
 25  Shampoo                      72830 non-null  bool
 26  Hangers                      72830 non-null  bool
 27  Carbon_monoxide_detector     72830 non-null  bool
 28  Internet                     72830 non-null  bool
 29  Laptop_friendly_workspace    72830 non-null  bool
 30  Hair_dryer                   72830 non-null  bool
 31  Washer                       72830 non-null  bool
 32  Dryer                        72830 non-null  bool
 33  Iron                         72830 non-null  bool
 34  Family_kid_friendly          72830 non-null  bool
 35  Fire_extinguisher            72830 non-null  bool
 36  First_aid_kit                72830 non-null  bool
 37  Cable_TV                     72830 non-null  bool
 38  Free_parking_on_premises     72830 non-null  bool
 39  24_hour_check_in             72830 non-null  bool
 40  Lock_on_bedroom_door         72830 non-null  bool
 41  Buzzer_wireless_intercom     72830 non-null  bool
 42  Safety_card                  72830 non-null  bool
 43  Self_Check_In                72830 non-null  bool
 44  Elevator                     72830 non-null  bool
 45  Pets_allowed                 72830 non-null  bool
dtypes: bool(32), float64(5), int64(5), object(4)
memory usage: 10.6+ MB
```

Figure 1: Información del dataset

Al confirmar que todo el dataset se encontraba completo se prosiguió con el análisis de cada característica del dataset y su relevancia con el objetivo de proyecto. Como ya se mencionó en secciones pasadas este proyecto tiene como objetivo predecir el precio de reservación en Airbnb de acuerdo con las amenidades que el espacio brinda al cliente. Es por eso que se decidió por eliminar las características relacionadas con el anfitrión del lugar (host_since, host_has_profile_pic, host_identity_verified) las cuales mostraban la fecha de registro del anfitrión, si contaba o no con una foto de perfil y si la identidad del anfitrión ya ha sido verificada. No obstante, es importante mencionar que dichas características serán consideradas para los siguientes análisis y su eliminación se hará al final de estos ya que todavía no se tiene certeza si dichas características tienen una relación relevante con otras características o la característica objetivo (el precio).

Después se pasó a analizar las propiedades intrínsecas de cada característica, esto por medio de estadísticas univariadas, específicamente se seleccionaron las técnicas del umbral de varianza y el coeficiente de correlación de Pearson. La varianza es una medida de dispersión que permite saber qué tanto varían una serie de datos respecto a la media aritmética de dicha serie. El objetivo de utilizar la técnica del umbral de varianza es detectar las características constantes y cuasi-constantes, es decir, las características que tienen un valor para todas las instancias o para la casi todas las instancias en el caso de las características cuasi-constantes. Antes de realizar dicha técnica se graficó la distribución de todas las características, esto con el objetivo de tener una visualización de esto.

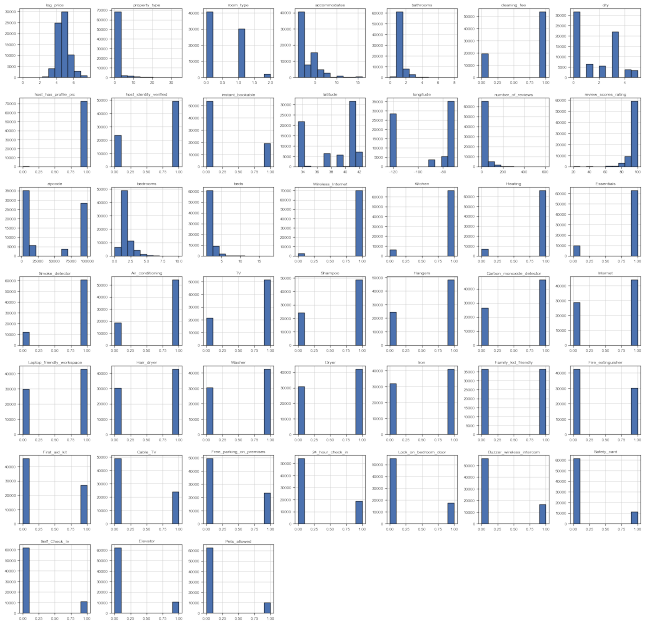


Figure 2: Distribuciones de las características

Como se puede observar en las gráficas, hay características que tienen una distribución cargada a un valor en específico. Para profundizar más en esto se utilizó la función VarianceThreshold() de la librería scikit-learn con un umbral de 0.05 para detectar así las

características que poseen el mismo valor en 95% o más de sus instancias.

Para algunas columnas, el valor de esas características fue principalmente cero, equivaliendo a un valor false. Esto hace que su varianza sea muy pequeña dejando de añadir información al modelo, es por eso que se tomó la decisión de removerlas del dataset.

3.3 Limpieza del dataset

El dataset se encontraba en buen estado, aunque aún así algunas filas tenían valores nulos o inválidos. Para generar un dataset sin irregularidades seguimos los siguientes pasos:

En primer lugar, las columnas con cadenas de caracteres difíciles de analizar (fuera del alcance del proyecto), con demasiados valores irregulares o mal formados, y columnas con valores irrelevantes para el proyecto fueron eliminadas, como:

- Tipo de cama
- Política de cancelación
- Descripción
- Fecha de la primera reseña
- Tasa de respuesta del anfitrión
- Fecha de la última reseña
- Nombre de la propiedad
- Vecindario en el que se ubica la propiedad
- Liga de la imagen de la propiedad

Algunas medidas tomadas con filas cuya información podía perjudicar los resultados fueron:

- 188 filas fueron eliminadas, ya que no contaban con un valor en la columna "host_identity_verified" y en "host_has_profile_pic"
- 135 filas que no tenían el número de camas, o que tenían 0 camas fueron eliminadas
- 958 filas que tenían un código postal inválido o difícil de analizar fueron eliminadas
- Los valores faltantes de la columna "review_scores_rating" (promedio de la calificación de las reseñas) fueron completadas con el valor promedio de la columna
- Los valores faltantes de las columnas de número de baños y de recámaras fueron llenadas con el valor 0

Algunas columnas tenían un tipo de dato incorrecto, por lo que fueron transformadas al tipo de dato que les correspondía, como:

- Las columnas 'host_has_profile_pic', 'host_identity_verified' y 'instant_bookable' fueron transformadas a valores booleanos (verdadero si su valor era la cadena "t")
- Las columnas "beds" y "bedrooms" fueron transformadas a enteros
- Para la columna de código postal, se seleccionaron los primeros 5 caracteres, los cuales fueron utilizados para crear un valor numérico entero
- La columna "host_since" fue convertida al tipo `pd.DateTime`, que representa una fecha en la librería de Python utilizada para manejar los datos: `pandas`

Otra tarea fue separar la columna de "amenities" en partes, ya que originalmente la columna se encontraba codificada como una lista en JSON (separada por comas, con todas las cadenas entre comillas). La mejor opción para representar esta lista fue hacer una matriz de valores booleanos, en donde cada celda representa si la casa o departamento (fila) contaba con el servicio (columna).

La lista de servicios era muy amplia, por lo que decidimos seleccionar solamente los servicios que tuvieran una frecuencia mayor a 10,000 y que no fueran errores de traducción (un par de servicios empiezan con la leyenda "translation_missing"), por lo que finalmente se generó una tabla con 28 columnas que representan 28 servicios o permisos:

- Wireless_Internet
- Kitchen
- Heating
- Essentials
- Smoke_detector
- Air_conditioning
- TV
- Shampoo
- Hangers
- Carbon_monoxide_detector
- Internet
- Laptop_friendly_workspace
- Hair_dryer
- Washer
- Dryer
- Iron
- Family_kid_friendly
- Fire_extinguisher
- First_aid_kit
- Cable_TV
- Free_parking_on_premises
- 24_hour_check_in
- Lock_on_bedroom_door
- Buzzer_wireless_intercom
- Safety_card
- Self_Check_In
- Elevator
- Pets_allowed

3.4 Procesamiento de los datos

Después de todas las modificaciones que se mencionaron en la secciones anteriores se obtuvo el siguiente dataset:

```
df.columns
```

```
Index(['property_type', 'room_type', 'accommodates', 'bathrooms',
       'cleaning_fee', 'city', 'instant_bookable', 'longitude',
       'number_of_reviews', 'review_scores_rating', 'zipcode', 'bedrooms',
       'beds', 'Kitchen', 'Heating', 'Essentials', 'Smoke_detector',
       'Air_conditioning', 'TV', 'Shampoo', 'Hangers',
       'Carbon_monoxide_detector', 'Internet', 'Laptop_friendly_workspace',
       'Hair_dryer', 'Dryer', 'Iron', 'Family_kid_friendly',
       'Fire_extinguisher', 'First_aid_kit', 'Cable_TV',
       'Free_parking_on_premises', '24_hour_check_in', 'Lock_on_bedroom_door',
       'Buzzer_wireless_intercom', 'Safety_card', 'Self_Check_In', 'Elevator',
       'Pets_allowed'],
      dtype='object')
```

```
df.shape
```

```
(72830, 39)
```

```
df.head()
```

id	property_type	room_type	accommodates	bathrooms	cleaning_fee	city	instant_bookable
6901257	Apartment	Entire home/apt	3	1.0	True	NYC	False
6304928	Apartment	Entire home/apt	7	1.0	True	NYC	True
7919400	Apartment	Entire home/apt	5	1.0	True	NYC	True
13418779	House	Entire home/apt	4	1.0	True	SF	False
3808709	Apartment	Entire home/apt	2	1.0	True	DC	True

5 rows × 39 columns

Figure 3: Distribuciones de las características

Como se hizo mención en la sección del análisis estadístico, el dataset contiene en su mayoría características categóricas. Con el fin de facilitar el aprendizaje y reducir el tiempo de ejecución se decidió utilizar un codificador y así hacer que las clases o categorías de cada característica se relacionarán con un valor entero.

Para esto la librería de scikit-learn propone los codificadores OrdinalEncoder() y OneHotEncoder(). Después de analizar cada uno se optó por usar OneHotEncoder(), esto principalmente porque OrdinalEncoder() da una jerarquización a los clases o categorías de cada característica y al tener múltiples características con más de dos clases o categorías esto afectaría los resultados esperados sobretodo en los algoritmo que no están hechos en base a árboles de decisiones ya que como se menciona la librería de scikit-learn estos son los que más se ven afectados.

No obstante, es importante mencionar que una de las desventajas al utilizar el codificador de OneHotEncoder() es que el tiempo de aprendizaje es más tardado a comparación de OrdinalEncoder(). Esto debido a que OneHotEncoder() hace una matriz de ceros y unos para relacionar todas las clases o categorías de las características y de está manera darle un valor numérico a cada clase o categoría sin hacer una jerarquización. Contrario a lo que hace OrdinalEncoder() que asigna un número de 0 al n-1, siendo n el número de clases.

Es importante mencionar que se hizo un fit_transform() para los datos de entrenamiento y transform() para los datos de evaluación, esto por cada iteración.

3.5 Modelación

Posteriormente se declararon las variables "X" y "y", las cuales guardan las características y los precios de las propiedades de Airbnb respectivamente, y se utilizó la función train_test_split() para dividir los distintos dataset, logrando obtener el 80% de los datos para el entrenamiento y el 20% restante para las pruebas.

Se utilizaron los siguientes modelos de regresión con la ayuda de sklearn[3]:

- Random Forest Regressor
- Gradient Boosting Regressor
- Linear Regression
- K-Neighbors Regressor
- Voting Regressor

3.5.1 Regresión por bosques aleatorios. Este modelo tomado del módulo sklearn.ensemble consiste en generar particiones aleatorias del set de entrenamiento a través del bootstraping y utilizarlas para entrenar diferentes árboles de decisión para regresión. El modelo toma el promedio de las predicciones resultantes para generar una predicción más estable, evitando el overfitting característico de los árboles de decisión.

3.5.2 Regresión utilizando Gradient Boosting. Este modelo también pertenece al módulo sklearn.ensemble. Su finalidad es minimizar una función de error que puede ser especificada[1], la opción escogida fue la de error cuadrático medio, que viene por defecto.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})$$

En el caso de esta implementación, el modelo usa árboles de decisión para poder realizar las regresiones. Los parámetros importantes como learning rate o criterio se dejaron como los valores por defecto.

3.5.3 Regresión lineal. El modelo de regresión lineal se basa en la ecuación que define la línea recta para establecer una predicción.

$$y = X\vec{B} + \epsilon$$

El parámetro fit:intercept se cambió al valor "True", mientras que todos los demás parámetros se dejaron con sus valores por defecto.

3.5.4 Regresión utilizando k-Vecinos más cercanos. El modelo de regresión basado en K-Neighbors realiza una predicción obteniendo el promedio de los valores de los vecinos más cercanos.

En este caso el número de vecinos utilizados fue 30. El parámetro n_jobs se definió como -1 para permitir que se utilicen todos los núcleos del cpu en el proceso.

3.5.5 Regresión utilizando un algoritmo de votación. Este modelo permite establecer una predicción obteniendo el promedio de las predicciones de varios modelos, todos aplicados a un mismo set de datos. Al igual que el modelo de K-neighbors el parámetro n_jobs se inicializó como -1. Con la finalidad de obtener una predicción que tomara en cuenta los resultados previos, para este modelo se utilizaron los modelos ya mencionados.

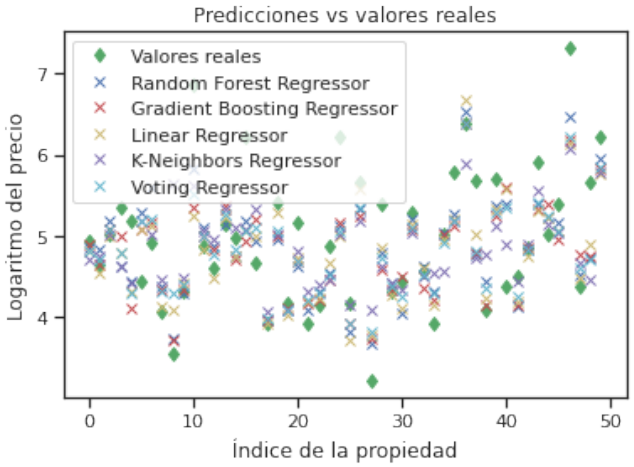
4 RESULTADOS

Todos los modelos se ejecutaron 28 veces, con un for loop y procurando que el parámetro de random_state de train_test_split fuera siempre el número de iteración. Esto para garantizar que cada iteración mostrara un resultado diferente por iteración pero repetible cada que se corra el programa.

Modelo	MSE
RandomForest	0.16103
GradientBoosting	0.17754
LinearRegression	0.16551
KNeighbors	0.22458
VotingRegressor	0.15988

Se inicializaron cinco arrays vacíos antes del ciclo for para que con cada iteración las puntuaciones de cada modelo se fueran guardando en sus respectivos arreglos. Al terminar el ciclo todos los arrays son guardados como csv en la misma carpeta donde el programa se encuentra.

Por último, se grafican los datos reales junto con la predicción de cada modelo para las primeras diez instancias, esto para poder visualizar el comportamiento de los modelos.



Los resultados para el error cuadrático medio obtenidos para los modelos en una sola iteración fueron los siguientes:

4.1 Prueba de Wilcoxon

Se realizó la prueba de Wilcoxon a los 22 resultados de MSE para cada modelo. Los resultados totales se pueden encontrar en el apéndice C, pero en resumen, el ranking de modelos fue en este orden:

(1) Voting Regressor
(2) Random Forest Regressor
(3) Linear Regression
(4) Gradient Boosting
(5) K-Neighbors

5 CONCLUSIONES

No es un resultado fuera de lo común, tomando en cuenta el funcionamiento del Voting Regressor y el Random Forest, los cuales toman en cuenta las predicciones de distintas modelos únicos, con diferentes sesgos.

A pesar de esto, resulta interesante que el modelo de regresión lineal haya resultado estadísticamente mejor que los modelos de gradient boosting y KNeighbors regressor.

REFERENCES

[1] Norman R Draper and Harry Smith. 1998. *Applied regression analysis*. Vol. 326. John Wiley & Sons.
[2] Issam El Naqa and Martin J Murphy. 2015. What is machine learning? In *machine learning in radiation oncology*. Springer, 3–11.
[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

A CÓDIGO PARA LA COMPARACIÓN DE MODELOS DE REGRESIÓN

```
import pathlib
import numpy as np
import pandas as pd
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor, VotingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

df_base = pd.read_csv('airbnb_clean.csv').set_index("id")
df_ammenities = pd.read_csv('airbnb_amenities_clean.csv').set_index("id")
df = pd.concat([df_base, df_ammenities], axis=1)

Y = df['log_price']
df.drop(columns=['log_price', 'Washer', 'latitude', 'host_since',
                 'host_has_profile_pic', 'Wireless_Internet',
                 'host_identity_verified'],
        inplace=True)
X = df

dict_column_dict = {}
column_len_unique_value = []
for column in df.columns:
    column_len_unique_value.append(len(df[column].unique()))
    column_dictionary = {df.columns[i]: column_len_unique_value[i]
                         for i in range(0, len(column_len_unique_value))}
dict_column_dict = (column_dictionary)


categorical_col = []
for key, value in dict_column_dict.items():
    if((value >= 2 and value < 700) and (key != 'bathrooms' and key != 'review_scores_rating')):
        categorical_col.append(key)

for i in range(len(categorical_col)):
    categorical_col[i] = df.columns.get_loc(categorical_col[i])

columnTransformer = ColumnTransformer([('encoder', OneHotEncoder(
    handle_unknown="ignore"), categorical_col)], remainder='passthrough')

rf_mses = []
gb_mses = []
lr_mses = []
kn_mses = []
vr_mses = []


for i in range(30):
    print(f"Calculating scores for iteration {i+1}")
    X_train, X_test, y_train, y_test = train_test_split(
        X, Y, test_size=0.2, random_state=i)
    X_train = columnTransformer.fit_transform(X_train)
    X_test = columnTransformer.transform(X_test)

    random_forest = RandomForestRegressor(n_estimators=100, n_jobs=-1)
    random_forest.fit(X_train, y_train)
    rf_pred = random_forest.predict(X_test)
    rf_mses.append(mean_squared_error(y_test, rf_pred))
```

B CÓDIGO DE PRUEBA DE WILCOXON

```
from pathlib import Path
import numpy as np
import pandas as pd
from scipy.stats import ranksums

mses_dict = {}

for mses in Path('mses_2').iterdir():
    if mses.is_file():
        mses_dict[mses.stem] = np.loadtxt(mses)

if len(mses_dict) == 0:
    print("No MSEs files found")
    print("Run `generate_mses.py` file first")

matrix_greater = []
matrix_less = []

for mses1 in mses_dict.values():
    row_greater = []
    row_less = []

    for mses2 in mses_dict.values():
        row_greater.append(
            ranksums(mses1, mses2, alternative="greater").pvalue)
        row_less.append(ranksums(mses1, mses2, alternative="less").pvalue)

    matrix_greater.append(row_greater)
    matrix_less.append(row_less)

df_greater = pd.DataFrame(
    matrix_greater, index=mses_dict.keys(), columns=mses_dict.keys())
df_less = pd.DataFrame(
    matrix_less, index=mses_dict.keys(), columns=mses_dict.keys())

pd.options.display.float_format = "{:.14f}".format
print("Wilcoxon rank sums test with a greater hypothesis")
print(df_greater)
print()

print("Wilcoxon rank sums test with a less hypothesis")
print(df_less)
```

C SALIDA DE EJECUCIÓN DE PRUEBA DE WILCOXON

C.1 Wilcoxon rank sums test with a greater hypothesis

	GradientBoosting	KNeighbors	LinearRegression	RandomForest	VotingRegressor
GradientBoosting	0.5	0.00000000671996	0.99999999328004	0.99999999328004	0.99999999328004
KNeighbors	0.99999999328004	0.5	0.99999999328004	0.99999999328004	0.99999999328004
LinearRegression	0.00000000671996	0.00000000671996	0.5	0.99819635515675	0.99999850169938
RandomForest	0.00000000671996	0.00000000671996	0.00180364484325	0.5	0.99698639393373
VotingRegressor	0.00000000671996	0.00000000671996	0.00000000001436	0.00301360606627	0.5

C.2 Wilcoxon rank sums test with a less hypothesis

	GradientBoosting	KNeighbors	LinearRegression	RandomForest	VotingRegressor
GradientBoosting	0.5	0.00000000671996	0.99999999328004	0.99999999328004	0.99999999328004
KNeighbors	0.99999999328004	0.5	0.99999999328004	0.99999999328004	0.99999999328004
LinearRegression	0.00000000671996	0.00000000671996	0.5	0.99819635515675	0.99999850169938
RandomForest	0.00000000671996	0.00000000671996	0.00180364484325	0.5	0.99698639393373
VotingRegressor	0.00000000671996	0.00000000671996	0.00000000001436	0.00301360606627	0.5

D MAPA DE CORRELACIÓN ENTRE VARIABLES