



Análisis del precio de las Viviendas en Colombia (2019-2020)

Realizado por: Mario E. Otero A.

LinkedIn: <https://www.linkedin.com/in/marioeoteroa/>

Github: <https://github.com/marioeoteroa/BootcampDSCodigofacilito>

Base de Datos tomada de: <https://www.kaggle.com/julianusugaortiz/colombia-housing-properties-price>

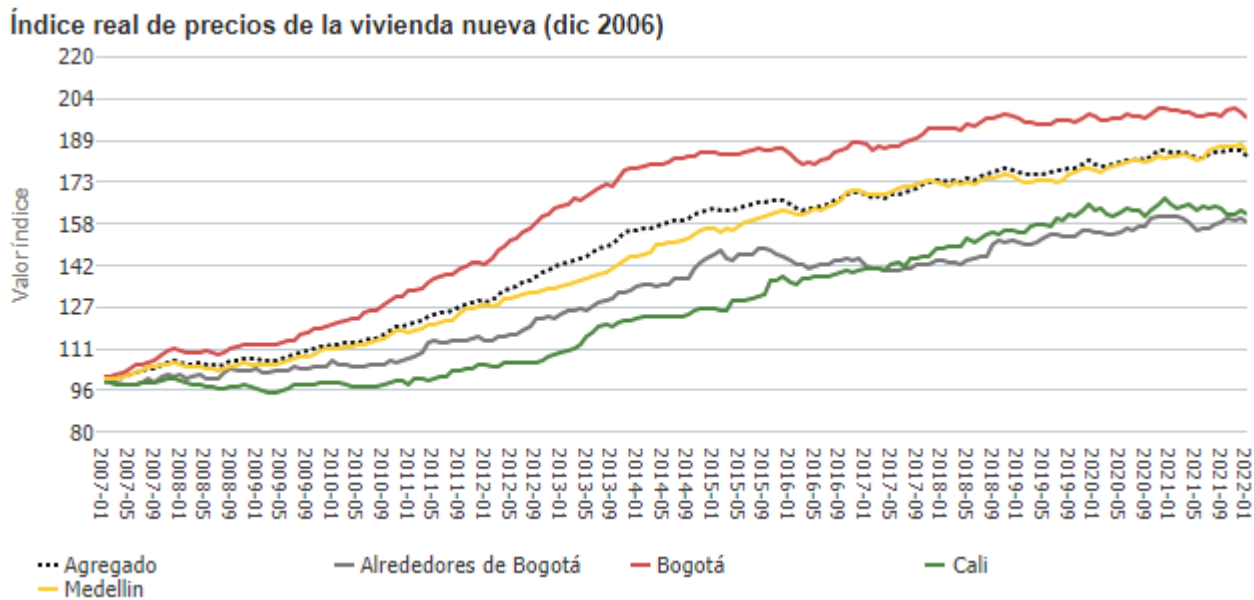


Foto: Getty Images/iStockphoto

El precio de la vivienda en Colombia ha ido aumentando significativamente a lo largo de las 2 ultimas decadas, esto se debe al incremento de inmuebles del segmento de Vivienda de Interés Social (VIS), al aumento de los costos de los materiales, entre otros, además se viene presentando un incremento de entrega de viviendas que las personas han comprado como inversión y destinan estos inmuebles en arriendo. según el más reciente informe de la plataforma inmobiliaria CienCuadras. El informe, a su vez,

determinó que la mayoría de las viviendas buscadas en Colombia tenían, mínimo, 2 habitaciones, 2 baños y, en general, se encontraban en estratos 2 y 3.

Tomado de: <https://www.portafolio.co/economia/finanzas/cuanto-vale-comprar-vivienda-en-colombia-561210> <https://www.semana.com/mejor-colombia/articulo/vivienda-una-prioridad-de-inversion-para-los-colombianos/202130/>



Tomada de: <https://www.banrep.gov.co/es/estadisticas/indice-precios-vivienda-nueva-ipvnbr>

Objetivos

Objetivo General

Estimar el precio de las viviendas en Colombia basados en técnicas de aprendizaje de máquina usando diferentes características a partir de un dataset con información de viviendas de los años 2019 y 2020.

Objetivos específicos

- Evaluar el estado de la base de datos encontrada en Kaggle.
- Realizar una limpieza y selección de las características que se usarán en los algoritmos.
- Entrenar diferentes algoritmos de aprendizaje de máquina supervisado utilizando las características extraídas del dataset para la estimación de los precios de las viviendas.
- Identificar las características y el algoritmo que proporcionan el mejor desempeño en la estimación de los precios de las viviendas a partir de métodos estadísticos.

Análisis Exploratorio de Datos

Instalar la librería de geopandas aquí

```
#!/pip install geopandas==0.10.2
```

```
# Importar librerías aquí
```

```
import numpy as np
import pandas as pd
import geopandas as gpd
import seaborn as sns
import matplotlib.pyplot as plt
from plotly.subplots import make_subplots
import plotly.graph_objects as go
import folium
from folium import Choropleth, Circle, Marker
from folium.plugins import HeatMap, MarkerCluster

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.ensemble import RandomForestRegressor

from pylab import *
import numpy as np
import matplotlib.pyplot as plt
from pandas import DataFrame
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn import linear_model
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import cross_validate
from statistics import mean
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.svm import SVR

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn import linear_model

from sklearn.model_selection import cross_validate
from statistics import mean
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeRegressor
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
```

```
# Importar el dataset (.csv)
```

```
dfOriginal = pd.read_csv('/work/co_properties.csv')
dfOriginal.head(5)
```

	id object	ad_type object	start_date obje...	end_date object	created_on obje...	lat
2	4et4/CQ6/jiiA31 QcGbBSQ==	Propiedad	2020-04-07	2020-05-22	2020-04-07	
1	Eb0qfrqoJKUuVFz kBymDgA==	Propiedad	2020-04-07	2020-05-15	2020-04-07	
4	Pg12IF9sRDSCcWZ U6L2yig==	Propiedad	2020-04-07	2020-07-20	2020-04-07	
3	DnzyL0D2CU/exv0 dQhVS/A==	Propiedad	2020-04-07	2020-07-02	2020-04-07	
0	Z5GURF86+s3KVdb vKdx4dQ==	Propiedad	2020-04-07	2020-05-22	2020-04-07	

```
#Observar el tamaño del dataset
```

```
dfOriginal.shape
```

```
#Observar las columnas que conforman el dataset
```

```
dfOriginal.columns
```

```
df = dfOriginal
#Extraer las propiedades que son de tipo Apartamento y Casa
property_types = (df["property_type"] == "Apartamento") | (df["property_type"] == "Casa")
property_types
df = df[property_types]

#Extraer las propiedades que solamente se encuentran en venta
operation_type = (df["operation_type"] == "Venta")
operation_type
df = df[operation_type]

#df.head(5)
dfbackup = df
```

```
#Eliminar las columnas que no son de interes para el proyecto
columns = df.columns
columns = columns[(columns != "id") & (columns != "start_date") & (columns != "end_date") & (columns != "operation_type")]
df = df[columns]
#df.head(5)
```

```
#Observar el nuevo tamaño del dataset
```

```
df.shape
```

```
#Renombrar las columnas de Departamento y Ciudad
df = df.rename(columns={'12': 'Department', '13': 'City'})
df.columns
```

```
#Contar la cantidad de datos con información nula
df.isnull().sum()
```

```
#Remover todas las filas que presenten datos nulos  
df.dropna(inplace = True)  
df.reset_index(drop=True,inplace=True)
```

```
df.isnull().sum()
```

```
#Observar el tamaño del dataset sin datos nulos  
df_filtrado = df  
df_filtrado.shape
```

```
#Filtrar nuevamente el dataset para eliminar datos incorrectos  
  
df_filtrado = df_filtrado[df_filtrado["price"] > 10000000]  
  
df_filtrado = df_filtrado[df_filtrado["price"] < 1000000001]  
  
df_filtrado = df_filtrado[df_filtrado["surface_total"] < 2000]
```



```
df_filtrado = df_filtrado[df_filtrado["surface_covered"] < 1000]

df_filtrado = df_filtrado[df_filtrado["bedrooms"] > -1]

df_filtrado = df_filtrado[df_filtrado["bathrooms"] > -1]

df_filtrado = df_filtrado[df_filtrado["surface_total"] > 0]

df_filtrado = df_filtrado[df_filtrado["surface_covered"] > 0]
```

#Añadir una columna con el calculo de una nueva característica

```
df_filtrado['P/sqmtr'] = df_filtrado['price'] // df_filtrado['surface_total']
```

#Observar que se alla agregado la nueva columna
df_filtrado.shape

#Guardar el .csv que corresponde al dataset filtrado con el que se va a trabajar
df_filtrado.to_csv('df_filtrado.csv')

#Leer el data set y observar qué columnas lo componen
df_filtrado = pd.read_csv('/work/df_filtrado.csv')
df_filtrado.head(5)

	Unnamed: 0 int64	lat float64	lon float64	Department obje...	City object	bedr
0	0	11.006	-74.808	Atlántico	Barranquilla	
1	2	4.689	-74.05	Cundinamarca	Bogotá D.C	
2	3	3.446	-76.551	Valle del Cauca	Cali	
3	4	3.383	-76.537	Valle del Cauca	Cali	
4	5	3.437	-76.548	Valle del Cauca	Cali	

#Eliminar una columna indeseada

```
columns = df_filtrado.columns
columns = columns[(columns != "Unnamed: 0")]
df_filtrado = df_filtrado[columns]
```

df_filtrado.shape

df_filtrado.head(10)

	lat float64 3.255 - 11.006	lon float64 -76.551 - -74.0...	Department obje... Valle del ... 60% Cundinamar... 30% Atlántico 10%	City object Cali 40% Bogotá D.C 30% 3 others 30%	bedrooms float64 2.0 - 5.0	bath
6	4.753	-74.069	Cundinamarca	Bogotá D.C	5	
1	4.689	-74.05	Cundinamarca	Bogotá D.C	2	
4	3.437	-76.548	Valle del Cauca	Cali	3	
7	3.438	-76.536	Valle del Cauca	Cali	4	
5	4.723	-74.039	Cundinamarca	Bogotá D.C	3	
2	3.446	-76.551	Valle del Cauca	Cali	3	
3	3.383	-76.537	Valle del Cauca	Cali	3	
9	3.255	-76.54	Valle del Cauca	Jamundí	3	
0	11.006	-74.808	Atlántico	Barranquilla	2	
8	3.576	-76.489	Valle del Cauca	Yumbo	2	

Visualización efectiva de Datos

```
#Mostrar el tamaño del dataset filtrado con el que se va a trabajar el proyecto
fig = make_subplots(rows=1, cols=2)
fig.add_trace(go.Indicator(mode = "number", value = df_filtrado.shape[0], number={'font':{'cc
fig.add_trace(go.Indicator(mode = "number", value = df_filtrado.shape[1], number={'font':{'cc
fig.show()
```


¿Donde están ubicadas las viviendas?

```
#Generar el mapa original
m_5 = folium.Map(location=[5.170035, -74.914305], tiles='cartodbpositron', zoom_start=6)

# Añadir el mapa de calor con las coordenadas al mapa original
HeatMap(data=df_filtrado[['lat', 'lon']], radius=10).add_to(m_5)

# Mostrar el mapa
m_5
```

```
df_filtrado.Department
```

```
#Se organizan los departamentos por la cantidad de viviendas en venta
Departamentos = df_filtrado['Department'].value_counts().reset_index()
Departamentos.columns = ['Department', 'counts']
Departamentos
```

	Department	counts
0	Cundinamarca	4256
1	Valle del Cauca	3870
2	Antioquia	2495
3	Atlántico	1925
4	Risaralda	385
5	Norte de Santander	221
6	Santander	214
7	Quindío	189
8	Bolívar	59
9	Tolima	53

```
#Graficar viviendas en venta por departamentos
```

```
x = Departamentos['Department'] #Aplicamos slicing como en una lista de Python.
y = Departamentos['counts']
```

```
plt.bar(x, y)
plt.style.use("fivethirtyeight")
```

```
plt.title('Viviendas por Departamentos')
plt.xlabel('Departamentos')
plt.ylabel('Cantidad de Viviendas')
plt.xticks(rotation='vertical') #Método que se usa para rotar el texto de los puntos en X para
plt.show()
```

```
x = Departamentos['Department'][0:10]#Aplicamos slicing como en una lista de Python.  
y = Departamentos['counts'][0:10]  
  
plt.bar(x, y)  
plt.title('Viviendas por Departamentos')  
plt.xlabel('Departamentos')  
plt.ylabel('Cantidad de Viviendas')  
plt.xticks(rotation='vertical') #Método que se usa para rotar el texto de los puntos en X par  
plt.show()
```

¿Qué tipos de vivienda se ofrecen?

df_filtrado.property_type

```
#Se halla la cantidad de cada uno de los tipos de vivienda
TipoVivienda = df_filtrado['property_type'].value_counts().reset_index()
TipoVivienda.columns = ['property_type', 'counts']
TipoVivienda
```

	property_type ...	counts int64	
0	Apartamento	9328	
1	Casa	4475	

```
#Graficar el porcentaje de tipos de vivienda que hay
plt.pie(TipoVivienda['counts'],labels = TipoVivienda['property_type'], autopct='%1.0f%%', pct
```

Precios de la vivienda en Colombia

```
#Se organizan los precios de las viviendas
Precios = df_filtrado['price']
Precios = df_filtrado.sort_values('price', ascending=0)
#Precios.head(100)
```

```
#Calculo del Rango de Precios
RangoPrecios = max(Precios['price']) - min(Precios['price'])
RangoPrecios
```

```
#Precio máximo
max(Precios['price'])
```

```
#Precio minimo
min(Precios['price'])
```


```
#Graficar histograma con la frecuencia de los precios de las viviendas
plt.hist(Precios['price'], bins=100)

plt.grid(True)
plt.xlabel('Precio')
plt.ylabel('Frecuencia')
plt.title('Histograma de los Precios de las Viviendas')
plt.xlim(28000000, 100000000)
plt.ylim(0,500)
plt.show()
```

```
#Organización de la media del precio del metro cuadrado por departamento
sp = df_filtrado[['Department', 'P/sqmtr']].groupby(['Department']).agg(['mean'])
sp.columns=['Media del Precio del Metro Cuadrado']

sp = sp.nlargest(30, ['Media del Precio del Metro Cuadrado'])

sp
```

	<div>Media del Pre... 2096691.2328042...</div> <div></div>	
Cundinamarca	4822493.44924812	
Bolívar	4319057.796610169	
Magdalena	4131217.5384615385	
Antioquia	3565034.454509018	
Boyacá	3330058.566666667	
Atlántico	3037159.315844156	
Caldas	2909760	
Valle del...	2810801.757364341	
Santander	2719460.26635514	
Córdoba	2700000	

```
sp.plot.bar()
```


Relaciones vs Precio de la Vivienda

#Graficar relación existente entre la cantidad de cuartos y el precio de la vivienda

```
sns.regplot(x = df_filtrado['bedrooms'], y = df_filtrado['price'])
```

```
#Graficar relación existente entre la cantidad de baños y el precio de la vivienda  
sns.regplot(x = df_filtrado['bathrooms'], y = df_filtrado['price'])
```

```
#Graficar relación existente entre la superficie total y el precio de la vivienda  
sns.regplot(x = df_filtrado['surface_total'], y = df_filtrado['price'])
```

#Graficar la relación existente entre el precio y la superficie total de la vivienda

```
x = df_filtrado['price']
y = df_filtrado['surface_total']

fig, ax = plt.subplots(figsize=(15, 10))

cmap = plt.cm.rainbow
norm = plt.Normalize(vmin=0, vmax=600)

color=cmap(norm(df_filtrado["surface_covered"].values))

plt.scatter(x, y, color = color)
plt.title('Precio vs Superficie Total')
plt.xlabel('Precio')
plt.ylabel('Superficie Total')
#ax.set_xlim(0, 15000000000)

sm = plt.cm.ScalarMappable(cmap=cmap, norm=norm)
sm.set_array([])
fig.colorbar(sm)

plt.show()
```





/shared-libs/python3.7/py-core/lib/python3.7/site-packages/ipykernel_launcher.py:21: MatplotlibDeprecationWarning:

Auto-removal of grids by pcolor() and pcolormesh() is deprecated since 3.5 and will be removed two minor releases

```
#Graficar un mapa de calor con la correlación existente entre las diferentes variables  
sns.heatmap(df_filtrado.corr(), annot = True)
```

Modelado de Datos

df_filtrado

	<div>lat float64</div> <div>2.44 - 11.269</div> <div></div>	<div>lon float64</div> <div>-77.078 - -72.46</div> <div></div>	<div>Department object</div> <div>Cundinamarca 30.8%</div> <div>Valle del Cauca 28%</div> <div>15 others 41.1%</div>	<div>City object</div> <div>Bogotá D.C 25.4%</div> <div>Cali 24.7%</div> <div>114 others 49.8%</div>	<div>bedrooms float64</div> <div>1.0 - 35.0</div> <div></div>	<div>bathrooms float64</div> <div>1.0 - 6.0</div> <div></div>
0	11.006	-74.808	Atlántico	Barranquilla	2	
1	4.689	-74.05	Cundinamarca	Bogotá D.C	2	
2	3.446	-76.551	Valle del Cauca	Cali	3	
3	3.383	-76.537	Valle del Cauca	Cali	3	
4	3.437	-76.548	Valle del Cauca	Cali	3	
5	4.723	-74.039	Cundinamarca	Bogotá D.C	3	
6	4.753	-74.069	Cundinamarca	Bogotá D.C	5	
7	3.438	-76.536	Valle del Cauca	Cali	4	
8	3.576	-76.489	Valle del Cauca	Yumbo	2	

9	3.255	-76.54	Valle del Cauca	Jamundí	3
---	-------	--------	-----------------	---------	---

#Convertir en variable numérica la variable categorica de tipo de vivienda

```
dfnew = pd.get_dummies(df_filtrado, columns=['property_type'],drop_first=True, dtype=float)
dfnew
```

```
dfnew = dfnew.rename(columns={'property_type_Casa': 'property_type',})
dfnew
```

	lat float64 2.44 - 11.269	lon float64 -77.078 - -72.46	Department object Cundinamarca 30.8% Valle del Cauca 28% 15 others 41.1%	City object Bogotá D.C. 25.4% Cali 24.7% 114 others 49.8%	bedrooms float64 1.0 - 35.0	bathrooms float64 1.0
0	11.006	-74.808	Atlántico	Barranquilla	2	
1	4.689	-74.05	Cundinamarca	Bogotá D.C	2	
2	3.446	-76.551	Valle del Cauca	Cali	3	
3	3.383	-76.537	Valle del Cauca	Cali	3	
4	3.437	-76.548	Valle del Cauca	Cali	3	
5	4.723	-74.039	Cundinamarca	Bogotá D.C	3	
6	4.753	-74.069	Cundinamarca	Bogotá D.C	5	
7	3.438	-76.536	Valle del Cauca	Cali	4	
8	3.576	-76.489	Valle del Cauca	Yumbo	2	
9	3.255	-76.54	Valle del Cauca	Jamundí	3	

```
dfnew.reset_index(drop=True, inplace=True)
dfnew
```

	lat float64 2.44 - 11.269	lon float64 -77.078 - -72.46	Department object Cundinamarca 30.8% Valle del Cauca 28% 15 others 41.1%	City object Bogotá D.C. 25.4% Cali 24.7% 114 others 49.8%	bedrooms float64 1.0 - 35.0	bathrooms float64 1.0
0	11.006	-74.808	Atlántico	Barranquilla	2	
1	4.689	-74.05	Cundinamarca	Bogotá D.C	2	

2	3.446	-76.551	Valle del Cauca	Cali	3
3	3.383	-76.537	Valle del Cauca	Cali	3
4	3.437	-76.548	Valle del Cauca	Cali	3
5	4.723	-74.039	Cundinamarca	Bogotá D.C	3
6	4.753	-74.069	Cundinamarca	Bogotá D.C	5
7	3.438	-76.536	Valle del Cauca	Cali	4
8	3.576	-76.489	Valle del Cauca	Yumbo	2
9	3.255	-76.54	Valle del Cauca	Jamundí	3

dfnew.columns

```
#Seleccionar las características usadas para realizar los modelos
train_features = dfnew[['bedrooms', 'bathrooms', 'surface_total', 'surface_covered', 'P/sqmtr', 'price']]
train_features.head(5)
```

	bedrooms float64	bathrooms float...	surface_total ...	surface_cover...	P/sqmtr float64	prop
0	2	2	95	95	2947368	
1	2	3	125	125	6480000	
2	3	2	91	91	5439560	
3	3	2	100	100	3500000	
4	3	3	155	139	4064516	

```
#Seleccionar el objetivo a estimar de los modelos
targets = dfnew[['price']]
targets.head(5)
```

price float64

0	280000000	
1	810000000	
2	495000000	
3	350000000	
4	630000000	

```
X = train_features
```

```
#Selecciono target
```

```
y = targets
```

Linear regression

```
#Con un ciclo for busco el modelo de regresión lineal que mejor desempeño presenta
```

```
ScorePrecision = []
```

```
for j in range(0,1):
```

```
    for i in range(0,5):
```

```
        #Selecciono predictores
```

```
X = train_features.iloc[:,i]
```

```
        #Selecciono target
```

```
y = targets.iloc[:,j]
```

```
from sklearn.model_selection import train_test_split
```

```
#separo datos de entrenamiento y prueba
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3)
```

```
X_train = X_train.values.reshape(-1,1)
```

```
X_test = X_test.values.reshape(-1,1)
```

```
## se escalan los datos
```

```
# from sklearn.preprocessing import MinMaxScaler
```

```
# escalar = MinMaxScaler(feature_range=(-1,1))
```

```
# X_train = escalar.fit_transform(X_train)
```

```
# X_test = escalar.fit_transform(X_test)
```

```
#MODELO DE REGRESION LINEAL SIMPLE
```

```
from sklearn import linear_model
```

```
LR_simple = linear_model.LinearRegression()
```

```

#Entreno el modelo
LR_simple.fit(X_train,y_train)

#Realizo una prediccion
y_pred = LR_simple.predict(X_test)

#Verifico precisión del modelo
precision = LR_simple.score(X_train,y_train)
ScorePrecision.append(precision)
# print('MODELO DE REGRESION LINEAL SIMPLE #' + str(i))
# print('Precision del modelo (R^2):', precision)

mejor = max(ScorePrecision)
posmejor = ScorePrecision.index(mejor)

print('-----')
print('Modelo con Mayor Precision (R^2) es:', posmejor)
print('La Precision (R^2) es:', mejor)
print('-----')

```

```

-----
Modelo con Mayor Precision (R^2) es: 1
La Precision (R^2) es: 0.3445607863599959
-----

```

ScorePrecision

Para la regresión lineal simple la característica que mejor estima el precio de la vivienda es la cantidad de baños con un coeficiente R^2 de 0.335

Multiple linear regression

```

#Con un ciclo for busco el modelo de regresión lineal multiple que mejor desempeño presenta
import time
start_time = time.time()

print('-----INICIO PROCESAMIENTO-----')

ScorePrecision = []
precision = []

```

```
precisionnew = []
RMSETrain = []
RMSETest = []
R2Train = []
R2Test = []
MSETrain = []
MSETest = []
MAETrain = []
MAETest = []
TablaVarios = []

variables = 6
for q in range(2,variables):
    for j in range(0,1):
        # Loading the data.

        #Selecciono predictores
        X = train_features.iloc[:,0:q].to_numpy()

        #Selecciono target
        y = targets.iloc[:,j].to_numpy()

        #separo datos de entrenamiento y prueba
        X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, random_

        ## se escalan los datos
        escalar = MinMaxScaler(feature_range=(-1,1))
        X_train = escalar.fit_transform(X_train)
        X_test = escalar.fit_transform(X_test)

        #MODELO DE REGRESION LINEAL SIMPLE
        LR_multiple = linear_model.LinearRegression(n_jobs=-1)

        #Entreno el modelo
        LR_multiple.fit(X_train,y_train)

        #Realizo una prediccion
        y_pred = LR_multiple.predict(X_test)

        y_train_pred = LR_multiple.predict(X_train)
        y_test_pred = LR_multiple.predict(X_test)

        #Verifico precisión del modelo
        precision = LR_multiple.score(X_train,y_train)
        ScorePrecision.append(precision)

        #RMSE root_mean_squared_error
        RMSETrainF = mean_squared_error(y_train, y_train_pred, squared=False)
        RMSETestF = mean_squared_error(y_test, y_test_pred, squared=False)

        RMSETrain.append(RMSETrainF)
        RMSETest.append(RMSETestF)
```

```
RMSETrainFinal = pd.DataFrame(RMSETrain)
RMSETestFinal = pd.DataFrame(RMSETest)

#r2 score
R2TrainF = r2_score(y_train, y_train_pred)
R2TestF = r2_score(y_test, y_test_pred)

R2Train.append(R2TrainF)
R2Test.append(R2TestF)

R2TrainFinal = pd.DataFrame(R2Train)
R2TestFinal = pd.DataFrame(R2Test)

#MSE mean_squared_error
MSETrainF = mean_squared_error(y_train, y_train_pred, squared=True)
MSETestF = mean_squared_error(y_test, y_test_pred, squared=True)

MSETrain.append(MSETrainF)
MSETest.append(MSETestF)

MSETrainFinal = pd.DataFrame(MSETrain)
MSETestFinal = pd.DataFrame(MSETest)

#MAE mean_absolute_error
MAETrainF = mean_absolute_error(y_train, y_train_pred)
MAETestF = mean_absolute_error(y_test, y_test_pred)

MAETrain.append(MAETrainF)
MAETest.append(MAETestF)

MAETrainFinal = pd.DataFrame(MAETrain)
MAETestFinal = pd.DataFrame(MAETest)

RMSE1= pd.concat([RMSETrainFinal,RMSETestFinal], axis=1)
R21 = pd.concat([R2TrainFinal,R2TestFinal], axis=1)
MSE1= pd.concat([MSETrainFinal,MSETestFinal], axis=1)
MAE1= pd.concat([MAETrainFinal,MAETestFinal], axis=1)

tabla1 = pd.concat([R21,RMSE1,MSE1,MAE1], axis=1)

print('-----')
print('PROCESAMIENTO FINALIZADO EXITOSAMENTE!!!')
print('-----')

TIEMPO = (time.time() - start_time)/60

# print("--- %s Segundos ---" % (time.time() - start_time))
print("--- %s Minutos ---" % (TIEMPO))

mejor = max(R2Train)
posmejor = R2Train.index(mejor)+1
mejor2 = max(R2Test)
```

```
posmejor2 = R2Test.index(mejor2)+1
tablaMLR.columns = ['R2Train' 'R2Test' 'RMSETrain' 'RMSETest' 'MSETrain' 'MSETest' 'MAETrain'

-----INICIO PROCESAMIENTO-----
-----
PROCESAMIENTO FINALIZADO EXITOSAMENTE!!!
-----

--- 0.003441111246744792 Minutos ---
```

```
tablaMLR.head()
```

	R2Train float64	R2Test float64	RMSETrain float...	RMSETest float64	MSETrain float64	MSETest
0	0.3810491496065753	0.30311628386307954	176493504.19090974	188998181.49909422	31149957021586680	357
1	0.439771025165252	0.3582847189024235	167912654.44587028	181362986.12917128	28194659523058240	328
2	0.48975316403122815	0.27297136158768	160247330.7147741	193042630.3146333	25679207001210188	372
3	0.7012969100565352	0.6295871712097473	122608478.04554479	137790881.01299748	15032838888644838	189

Para la regresión lineal multiple las características que mejor estiman el precio de la vivienda son la cantidad de cuartos, cantidad de baños, la superficie total de la vivienda y la superficie cubierta de la vivienda, con un coeficiente R^2 de 0.701

Decision Tree

```
#Se definen los hiperparametros a optimizar
```

```
max_depth = 10
```

```
minsamplesplit = 20
```

```
minsampleleaf = 20
```

```
param_grid = {"model__max_depth": list(range(1, max_depth + 1)),
              "model__min_samples_split": list(range(1, minsamplesplit + 1)),
              "model__min_samples_leaf": list(range(1, minsampleleaf + 1))
              }
```

```
#Se realiza una optimización de hiperparametros con 10 validaciones cruzadas
```

```
import time
```

```
start_time = time.time()
```

```
print('-----INICIO PROCESAMIENTO-----')
```

```
# Train, test split.
```

```
data_split = train_test_split(train_features, targets, test_size=0.3, random_state=4444)
X_train, X_test, targets_train, targets_test = data_split
```

```

# Find the best model per target.
for target in range(targets.shape[1]):
    # Getting the target.
    y_train = targets_train.iloc[:, target]
    y_test = targets_test.iloc[:, target]

    # Model definition.
    pipe = Pipeline([("minmax", MinMaxScaler((-1, 1))),
                     ("model", DecisionTreeRegressor(random_state=4444))])
    # By default GridSearchCV uses a 5-kfold validation strategy.
    search = GridSearchCV(pipe, param_grid, cv=10, n_jobs=-1)
    search.fit(X_train.values, y_train.values)

    # Getting the test score.
    y_hat_test = search.predict(X_test.values)
    test_score = r2_score(y_test.values, y_hat_test)

    # Printing stats.
    print(f"Columna: {targets.columns[target]}")
    print(f"Best CV score: {search.best_score_:0.3f}")
    print(f"Test score: {test_score:0.3f}")
    print(f"Best Parameters:\n {search.best_params_}")
    print("")

print('-----')
print('PROCESAMIENTO FINALIZADO EXITOSAMENTE!!!')
print('-----')

TIEMPO = (time.time() - start_time)/60

# print("--- %s Segundos ---" % (time.time() - start_time))
print("--- %s Minutos ---" % (TIEMPO))

```

```
{'model__max_depth': 10, 'model__min_samples_leaf': 1, 'model__min_samples_split': 5}
```

```
-----
PROCESAMIENTO FINALIZADO EXITOSAMENTE!!!
-----
```

```
--- 2.168781594435374 Minutos ---
```

```
/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/model_selection/_validation.py:372: FitFailed
```

2000 fits failed out of a total of 40000.

The score on these train-test partitions for these parameters will be set to nan.

If these failures are not expected, you can try to debug them by setting `error_score='raise'`.

Below are more details about the failures:

```
-----
2000 fits failed with the following error:
```

```
Traceback (most recent call last):
```

```
File "/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/model_selection/_validation.py", line
https://deepnote.com/@marioeoteroa/Proyecto-Bootcamp-CodigoFacilito-ajFRH5JbTmWHnMNIbRPJ9Q
```

```

estimator.fit(X_train, y_train, **fit_params)
File "/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/pipeline.py", line 394, in fit
    self._final_estimator.fit(Xt, y, **fit_params_last_step)
File "/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/tree/_classes.py", line 1320, in fit
    X_idx_sorted=X_idx_sorted,
File "/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/tree/_classes.py", line 254, in fit
    % self.min_samples_split
ValueError: min_samples_split must be an integer greater than 1 or a float in (0.0, 1.0]; got the integer 1

/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/model_selection/_search.py:972: UserWarning:

```

```

-----INICIO PROCESAMIENTO----- Columna: price Best CV score: 0.987 Test
score: 0.990 Best Parameters: {'model__max_depth': 10, 'model__min_samples_leaf': 1,
'model__min_samples_split': 5} -----
PROCESAMIENTO FINALIZADO EXITOSAMENTE!!! -----
---- --- 2.127672537167867 Minutos ---

```

```

#Se definen los hiperparametros optimos encontrados
maxdepth1 = 10
minsampleleaf1 = 1
minsamplesplit1 = 5

```

```

TiempoEntrenamientoTRAIN=[]
TiempoEvaluacionTRAIN=[]
RMSETrain=[]
R2Train=[]
MSETrain=[]
MAETrain=[]

```

```

TiempoEntrenamientoTEST=[]
TiempoEvaluacionTEST=[]
RMSETest=[]
R2Test=[]
MSETest=[]
MAETest=[]

```

```

#Se generan los modelos con el algoritmo elegido y calculan las metricas de evaluación

```

```

import time
start_time = time.time()

print('-----INICIO PROCESAMIENTO-----')

X = train_features

#Selecciono target
y = targets

```



```

#separo datos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, random_state=4444)

## se escalan los datos
# escalar = MinMaxScaler(feature_range=(-1,1))
# X_train = escalar.fit_transform(X_train)
# X_test = escalar.fit_transform(X_test)
#MODELO DE REGRESION LINEAL SIMPLE
LR_simple = Pipeline([("minmax", MinMaxScaler((-1, 1))),
                      ("model", DecisionTreeRegressor(max_depth=maxdepth1,
                                                         min_samples_leaf=minsampleleaf1,
                                                         min_samples_split=minsamplesplit1,
                                                         random_state=4444))])

# LR_simple.fit(X_train, y_train)
# LR_simple = RandomForestRegressor(max_depth=maxdepth1,n_estimators=nestimators,min_samples_
scoring = ['r2', 'neg_root_mean_squared_error', 'neg_mean_squared_error', 'neg_mean_absolute_err

###Con datos de Train
scores1 = cross_validate(LR_simple, X_train, y_train, scoring=scoring, n_jobs=-1, cv=10)
TiempoPromedioEntrenamientoTRAIN = mean(scores1['fit_time'])
TiempoPromedioEvaluacionTRAIN = mean(scores1['score_time'])
RMSEPromedioTRAIN = mean(scores1['test_neg_root_mean_squared_error'])
R2PromedioTRAIN = mean(scores1['test_r2'])
MSEPromedioTRAIN = mean(scores1['test_neg_mean_squared_error'])
MAEPromedioTRAIN = mean(scores1['test_neg_mean_absolute_error'])
TiempoEntrenamientoTRAIN.append(TiempoPromedioEntrenamientoTRAIN)
TiempoEvaluacionTRAIN.append(TiempoPromedioEvaluacionTRAIN)
RMSETrain.append(RMSEPromedioTRAIN)
R2Train.append(R2PromedioTRAIN)
MSETrain.append(MSEPromedioTRAIN)
MAETrain.append(MAEPromedioTRAIN)

RMSETrainFinal=pd.DataFrame(RMSETrain)
R2TrainFinal=pd.DataFrame(R2Train)
MSETrainFinal=pd.DataFrame(MSETrain)
MAETrainFinal=pd.DataFrame(MAETrain)

###Con datos de Test
scores2 = cross_validate(LR_simple, X_test, y_test, scoring=scoring, n_jobs=-1, cv=10)

# # Getting the test score.
# y_hat_test = LR_simple.predict(X_test)
# test_score = r2_score(y_test, y_hat_test)

TiempoPromedioEntrenamientoTEST = mean(scores2['fit_time'])
TiempoPromedioEvaluacionTEST = mean(scores2['score_time'])
RMSEPromedioTEST = mean(scores2['test_neg_root_mean_squared_error'])
R2PromedioTEST = mean(scores2['test_r2'])
MSEPromedioTEST = mean(scores2['test_neg_mean_squared_error'])
MAEPromedioTEST = mean(scores2['test_neg_mean_absolute_error'])
TiempoEntrenamientoTEST.append(TiempoPromedioEntrenamientoTEST)
TiempoEvaluacionTEST.append(TiempoPromedioEvaluacionTEST)
RMSETest.append(RMSEPromedioTEST)

```

```

R2Test.append(R2PromedioTEST)
MSETest.append(MSEPromedioTEST)
MAETest.append(MAEPromedioTEST)

RMSETestFinal=pd.DataFrame(RMSETest)
R2TestFinal=pd.DataFrame(R2Test)
MSETestFinal=pd.DataFrame(MSETest)
MAETestFinal=pd.DataFrame(MAETest)

RMSE1 = pd.concat([RMSETrainFinal,RMSETestFinal], axis=1)
R21 = pd.concat([R2TrainFinal,R2TestFinal], axis=1)
MSE1= pd.concat([MSETrainFinal,MSETestFinal], axis=1)
MAE1= pd.concat([MAETrainFinal,MAETestFinal], axis=1)
####Creacion Tablas
tabla1 = pd.concat([R21,RMSE1,MSE1,MAE1], axis=1)
tabla1.columns = ['R2Train', 'R2Test', 'RMSETrain', 'RMSETest', 'MSETrain', 'MSETest', 'MAETrain', 'MAETest']

```

-----INICIO PROCESAMIENTO-----

tabla1.head()

	R2Train float64	R2Test float64	RMSETrain float...	RMSETest float64	MSETrain float64	MSETest float64
0	0.9868935078857479	0.9795300357687989	-25536440.519596	-31744515.006299447	-658123311065205.9	-1065205.9

Para los arboles de decisión los hiperparametros que mejor estiman el precio de la vivienda son:
 maxdepth = 10, minsampleleaf = 1, minsamplesplit = 5, con un coeficiente R^2 de 0.979

Random Forest

#Se definen los hiperparametros a optimizar

```

param_gride = {"model__max_depth": [5,10,15,20], "model__min_samples_split": [5,10,15,20], "model__min_samples_leaf": [1,2,3,4,5,6,7,8,9,10]}
#param_gride = {"model__max_depth": [5,10,20], "model__min_samples_split": [5,10,20], "model__min_samples_leaf": [1,2,3,4,5,6,7,8,9,10]}
#param_gride = {"model__max_depth": [5,20], "model__min_samples_split": [5,10], "model__min_samples_leaf": [1,2,3,4,5,6,7,8,9,10]}
#param_gride = {"model__max_depth": [5], "model__min_samples_split": [5], "model__min_samples_leaf": [1,2,3,4,5,6,7,8,9,10]}

# param_gride = {"model__max_depth": [10,20,30,40,50,60,70,80,90,100],
#                 "model__min_samples_split": [2,4,6,8,10,12,14,16,18,20],
#                 "model__min_samples_leaf": [1,2,3,4,5,6,7,8,9,10],
#                 "model__n_estimators": [100,200,300,400,500,600,700,800,900,1000]}

```

#Se realiza una optimización de hiperparametros con 10 validaciones cruzadas

```

import time
start_time = time.time()
print('-----INICIO PROCESAMIENTO-----')

# Train, test split.
data_split = train_test_split(train_features, targets, test_size=0.3, random_state=4444)
X_train, X_test, targets_train, targets_test = data_split

# Find the best model per target.
for target in range(targets.shape[1]):
    # Getting the target.
    y_train = targets_train.iloc[:, target]
    y_test = targets_test.iloc[:, target]

    # Model definition.
    pipe = Pipeline([("minmax", MinMaxScaler((-1, 1))),
                     ("model", RandomForestRegressor(random_state=4444, n_jobs=-1))])
    # By default GridSearchCV uses a 5-kfold validation strategy.
    search = GridSearchCV(estimator=pipe, param_grid = param_gride, cv=10, n_jobs=-1)
    search.fit(X_train.values, y_train.values)

    # Getting the test score.
    y_hat_test = search.predict(X_test.values)
    test_score = r2_score(y_test.values, y_hat_test)

    # Printing stats.
    print(f"Columna: {targets.columns[target]}")
    print(f"Best CV score: {search.best_score_:0.3f}")
    print(f"Test score: {test_score:0.3f}")
    print(f"Best Parameters:\n {search.best_params_}")
    print("")

print('-----')
print('PROCESAMIENTO FINALIZADO EXITOSAMENTE!!!')
print('-----')

TIEMPO = (time.time() - start_time)/60

# print("--- %s Segundos ---" % (time.time() - start_time))
print("--- %s Minutos ---" % (TIEMPO))

```

```

-----INICIO PROCESAMIENTO-----
Columna: price
Best CV score: 0.995
Test score: 0.996
Best Parameters:
{'model__max_depth': 20, 'model__min_samples_leaf': 5, 'model__min_samples_split': 5, 'model__n_estimators': 100}
-----
PROCESAMIENTO FINALIZADO EXITOSAMENTE!!!

```

```
-----
--- 3.8578356504440308 Minutos ---
```

```
-----INICIO PROCESAMIENTO----- Columna: price Best CV score: 0.995 Test
score: 0.996 Best Parameters: {'model__max_depth': 20, 'model__min_samples_leaf': 5,
'model__min_samples_split': 5, 'model__n_estimators': 100} -----
----- PROCESAMIENTO FINALIZADO EXITOSAMENTE!!! -----
----- --- 4.091889691352844 Minutos ---
```

```
#Se definen los hiperparametros optimos encontrados
```

```
maxdepth1 = 20
minsampleleaf1 = 5
minsamplesplit1 = 5
n_estimators = 100
```

```
TiempoEntrenamientoTRAIN=[]
TiempoEvaluacionTRAIN=[]
RMSETrain=[]
R2Train=[]
MSETrain=[]
MAETrain=[]
```

```
TiempoEntrenamientoTEST=[]
TiempoEvaluacionTEST=[]
RMSETest=[]
R2Test=[]
MSETest=[]
MAETest=[]
```

```
X = train_features
```

```
#Selecciono target
```

```
y = targets
```

```
#separo datos de entrenamiento y prueba
```

```
X_train1, X_test1, y_train1, y_test1 = train_test_split(X,y, test_size = 0.3, random_state=44
```

```
## se escalan los datos
```

```
# escalar = MinMaxScaler(feature_range=(-1,1))
```

```
# X_train = escalar.fit_transform(X_train)
```

```
# X_test = escalar.fit_transform(X_test)
```

```
#MODELO DE RANDOM FOREST REGRESSOR
```

```
LR_simple1 = Pipeline([("minmax", MinMaxScaler((-1, 1))),
                        ("model", RandomForestRegressor(max_depth=maxdepth1,
                                                         n_estimators=n_estimators,
                                                         min_samples_leaf=minsampleleaf1,
                                                         min_samples_split=minsamplesplit1,
                                                         random_state=4444,n_jobs=-1))])
```

```

#Se generan los modelos con el algoritmo elegido y calculan las metricas de evaluación
LR_simple1.fit(X_train1, y_train1.values.ravel())
#LR_simple = RandomForestRegressor(max_depth=maxdepth1,n_estimators=nestimators,min_samples_i
scoring = ['r2', 'neg_root_mean_squared_error', 'neg_mean_squared_error', 'neg_mean_absolute_err

###Con datos de Train
scores1 = cross_validate(LR_simple1, X_train1, y_train1, scoring=scoring, n_jobs=-1, cv=10)
TiempoPromedioEntrenamientoTRAIN = mean(scores1['fit_time'])
TiempoPromedioEvaluacionTRAIN = mean(scores1['score_time'])
RMSEPromedioTRAIN = mean(scores1['test_neg_root_mean_squared_error'])
R2PromedioTRAIN = mean(scores1['test_r2'])
MSEPromedioTRAIN = mean(scores1['test_neg_mean_squared_error'])
MAEPromedioTRAIN = mean(scores1['test_neg_mean_absolute_error'])

RMSETrain.append(RMSEPromedioTRAIN)
R2Train.append(R2PromedioTRAIN)
MSETrain.append(MSEPromedioTRAIN)
MAETrain.append(MAEPromedioTRAIN)

RMSETrainFinal=pd.DataFrame(RMSETrain)
R2TrainFinal=pd.DataFrame(R2Train)
MSETrainFinal=pd.DataFrame(MSETrain)
MAETrainFinal=pd.DataFrame(MAETrain)

###Con datos de Test
scores2 = cross_validate(LR_simple1, X_test1, y_test1, scoring=scoring, n_jobs=-1, cv=10)

# # Getting the test score.
# y_hat_test = LR_simple.predict(X_test)
# test_score = r2_score(y_test, y_hat_test)

TiempoPromedioEntrenamientoTEST = mean(scores2['fit_time'])
TiempoPromedioEvaluacionTEST = mean(scores2['score_time'])
RMSEPromedioTEST = mean(scores2['test_neg_root_mean_squared_error'])
R2PromedioTEST = mean(scores2['test_r2'])
MSEPromedioTEST = mean(scores2['test_neg_mean_squared_error'])
MAEPromedioTEST = mean(scores2['test_neg_mean_absolute_error'])
TiempoEntrenamientoTEST.append(TiempoPromedioEntrenamientoTEST)
TiempoEvaluacionTEST.append(TiempoPromedioEvaluacionTEST)
RMSETest.append(RMSEPromedioTEST)
R2Test.append(R2PromedioTEST)
MSETest.append(MSEPromedioTEST)
MAETest.append(MAEPromedioTEST)

RMSETestFinal=pd.DataFrame(RMSETest)
R2TestFinal=pd.DataFrame(R2Test)
MSETestFinal=pd.DataFrame(MSETest)
MAETestFinal=pd.DataFrame(MAETest)

RMSE1 = pd.concat([RMSETrainFinal,RMSETestFinal], axis=1)
R21 = pd.concat([R2TrainFinal,R2TestFinal], axis=1)

```

```
MSE1= pd.concat([MSETrainFinal,MSETestFinal], axis=1)
MAE1= pd.concat([MAETrainFinal,MAETestFinal], axis=1)
####Creacion Tablas
tabla2 = pd.concat([R21,RMSE1,MSE1,MAE1], axis=1)
tabla2.columns = ['R2Train','R2Test','RMSETrain','RMSETest','MSETrain','MSETest','MAETrain','MAETest']
```

```
tabla2.head(1)
```

	R2Train float64	R2Test float64	RMSETrain float...	RMSETest float64	MSETrain float64	MSETest float64
0	0.9952417541641426	0.9878998894676966	-15191634.560280189	-24409685.264608756	-240601962269160.75	-6101962269160.75

Para los bosques aleatorios los hiperparametros que mejor estiman el precio de la vivienda son: maxdepth = 20, minsampleleaf = 5, minsamplesplit = 5 y nestimators = 100, con un coeficiente R^2 de 0.987

```
test_X = X_test1
test_preds = LR_simple1.predict(test_X)
test_preds = pd.DataFrame(test_preds)
test_preds
```

	0 float64 61784470.615827 ...	
0	414585450.9999802	
1	821168971.7332991	
2	701727734.7809889	
3	569409610.7752174	
4	155385782.656071	
5	105878158.5851541	
6	732522383.568177	
7	637839271.2010212	
8	467461508.47879267	
9	334001885.3934848	

targets



0	280000000
1	810000000
2	495000000
3	350000000
4	630000000
5	500000000
6	850000000
7	600000000
8	170000000
9	330000000

Support Vector Machine

#Se definen los hiperparametros a optimizar

```
#param_gride = {"model__kernel": ['poly', 'rbf'],
#               "model__gamma": ['scale', 'auto'],
#               "model__degree": [1, 2, 3, 4, 5],
#               "model__epsilon": [0.0001, 0.00025, 0.0005, 0.001, 0.01, 0.05, 0.1, 0.5, 1],
#               "model__C": [1, 5, 10, 25, 50, 100]}
#
```

#Se realiza una optimización de hiperparametros con 10 validaciones cruzadas

```
import time
start_time = time.time()

print('-----INICIO PROCESAMIENTO-----')

X = train_features

#Selecciono target
```



```

y = targets

# Train, test split.
data_split = train_test_split(train_features, targets, test_size=0.3, random_state=4444)
X_train, X_test, targets_train, targets_test = data_split

# Find the best model per target.
#for target in range(targets.shape[1]):
#    # Getting the target.
#    y_train = targets_train.iloc[:, target]
#    y_test = targets_test.iloc[:, target]

# Model definition.
pipe = Pipeline([("minmax", MinMaxScaler((-1, 1))),
                  ("model", SVR())])
# By default GridSearchCV uses a 5-kfold validation strategy.
search = GridSearchCV(estimator=pipe, param_grid = param_gride, cv=10, n_jobs=-1)
search.fit(X_train.values, y_train.values)

# Getting the test score.
y_hat_test = search.predict(X_test.values)
test_score = r2_score(y_test.values, y_hat_test)

# Printing stats.
print(f"Columna: {targets.columns[target]}")
print(f"Best CV score: {search.best_score_:0.3f}")
print(f"Test score: {test_score:0.3f}")
print(f"Best Parameters:\n {search.best_params_}")
print("")

print('-----')
print('PROCESAMIENTO FINALIZADO EXITOSAMENTE!!!')
print('-----')

TIEMPO = (time.time() - start_time)/60

# print("--- %s Segundos ---" % (time.time() - start_time))
print("--- %s Minutos ---" % (TIEMPO))

```

```

-----INICIO PROCESAMIENTO-----
Columna: price
Best CV score: 0.108
Test score: 0.127
Best Parameters:
{'model__C': 100, 'model__degree': 5, 'model__epsilon': 0.0001, 'model__gamma': 'scale', 'model__kernel':

-----
PROCESAMIENTO FINALIZADO EXITOSAMENTE!!!
-----
--- 167.60582628647487 Minutos ---

```

-----INICIO PROCESAMIENTO----- Columna: price Best CV score: 0.108 Test
 score: 0.127 Best Parameters: {'model__C': 100, 'model__degree': 5, 'model__epsilon': 0.0001,
 'model__gamma': 'scale', 'model__kernel': 'poly'} -----
 - PROCESAMIENTO FINALIZADO EXITOSAMENTE!!! -----
 ----- 167.60582628647487 Minutos ---

```
#Se definen los hiperparametros optimos encontrados
C1 = 100
degree1 = 5
epsilon1 = 0.0001
gamma1 = 'scale'
kernel1 = 'poly'

#Se generan los modelos con el algoritmo elegido y calculan las metricas de evaluación

X = train_features

#Selecciono target
y = targets

#separo datos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, random_state=4444)

## se escalan los datos
escalar = MinMaxScaler(feature_range=(-1,1))
X_train = escalar.fit_transform(X_train)
X_test = escalar.fit_transform(X_test)
#MODELO DE REGRESION LINEAL SIMPLE
LR_simple = SVR(kernel=kernel1,degree=degree1,
                 gamma=gamma1,epsilon=epsilon1, C=C1)

# LR_simple.fit(X_train, y_train)
scoring = ['r2', 'neg_root_mean_squared_error', 'neg_mean_squared_error', 'neg_mean_absolute_err

TiempoEntrenamientoTRAIN=[]
TiempoEvaluacionTRAIN=[]
RMSETrain=[]
R2Train=[]
MSETrain=[]
MAETrain=[]

TiempoEntrenamientoTEST=[]
TiempoEvaluacionTEST=[]
RMSETest=[]
R2Test=[]
MSETest=[]
MAETest=[]
```

```
###Con datos de Train
```

```

scores1 = cross_validate(LR_simple, X_train, y_train, scoring=scoring, n_jobs=-1, cv=10)
TiempoPromedioEntrenamientoTRAIN = mean(scores1['fit_time'])
TiempoPromedioEvaluacionTRAIN = mean(scores1['score_time'])
RMSEPromedioTRAIN = mean(scores1['test_neg_root_mean_squared_error'])
R2PromedioTRAIN = mean(scores1['test_r2'])
MSEPromedioTRAIN = mean(scores1['test_neg_mean_squared_error'])
MAEPromedioTRAIN = mean(scores1['test_neg_mean_absolute_error'])
TiempoEntrenamientoTRAIN.append(TiempoPromedioEntrenamientoTRAIN)
TiempoEvaluacionTRAIN.append(TiempoPromedioEvaluacionTRAIN)
RMSETrain.append(RMSEPromedioTRAIN)
R2Train.append(R2PromedioTRAIN)
MSETrain.append(MSEPromedioTRAIN)
MAETrain.append(MAEPromedioTRAIN)

RMSETrainFinal=pd.DataFrame(RMSETrain)
R2TrainFinal=pd.DataFrame(R2Train)
MSETrainFinal=pd.DataFrame(MSETrain)
MAETrainFinal=pd.DataFrame(MAETrain)

###Con datos de Test
scores2 = cross_validate(LR_simple, X_test, y_test, scoring=scoring, n_jobs=-1, cv=10)

# # Getting the test score.
# y_hat_test = LR_simple.predict(X_test)
# test_score = r2_score(y_test, y_hat_test)

TiempoPromedioEntrenamientoTEST = mean(scores2['fit_time'])
TiempoPromedioEvaluacionTEST = mean(scores2['score_time'])
RMSEPromedioTEST = mean(scores2['test_neg_root_mean_squared_error'])
R2PromedioTEST = mean(scores2['test_r2'])
MSEPromedioTEST = mean(scores2['test_neg_mean_squared_error'])
MAEPromedioTEST = mean(scores2['test_neg_mean_absolute_error'])
TiempoEntrenamientoTEST.append(TiempoPromedioEntrenamientoTEST)
TiempoEvaluacionTEST.append(TiempoPromedioEvaluacionTEST)
RMSETest.append(RMSEPromedioTEST)
R2Test.append(R2PromedioTEST)
MSETest.append(MSEPromedioTEST)
MAETest.append(MAEPromedioTEST)

RMSETestFinal=pd.DataFrame(RMSETest)
R2TestFinal=pd.DataFrame(R2Test)
MSETestFinal=pd.DataFrame(MSETest)
MAETestFinal=pd.DataFrame(MAETest)

RMSE1 = pd.concat([RMSETrainFinal,RMSETestFinal], axis=1)
R21 = pd.concat([R2TrainFinal,R2TestFinal], axis=1)
MSE1= pd.concat([MSETrainFinal,MSETestFinal], axis=1)
MAE1= pd.concat([MAETrainFinal,MAETestFinal], axis=1)
####Creacion Tablas
tabla3 = pd.concat([R21,RMSE1,MSE1,MAE1], axis=1)
tabla3.columns = ['R2Train', 'R2Test', 'RMSETrain', 'RMSETest', 'MSETrain', 'MSETest', 'MAETrain', '

```

```
tabla3.head(1)
```

	R2Train float64	R2Test float64	RMSETrain float...	RMSETest float64	MSETrain float64	MSET
0	0.1088928439816 486	0.0237101144518 4824	-211618707.7658 3248	-223340976.4032 2065	-44831100936028 890	-49

Para las máquinas de soporte vectorial los hiperparametros que mejor estiman el precio de la vivienda son: C1 = 100, degree = 5, epsilon = 0.0001, gamma = 'scale', kernel = 'poly', con un coeficiente R² de 0.024

```
tablaFINAL = pd.concat([tabla1,tabla2,tabla3], axis=0)
tablaFINAL.head(3)
```

	R2Train float64	R2Test float64	RMSETrain float...	RMSETest float64	MSETrain float64	MSET
0	0.9868935078857 479	0.9795300357687 989	-25536440.51959 6	-31744515.00629 9447	-65812331106520 5.9	-10
0	0.9952417541641 426	0.9878998894676 966	-15191634.56028 0189	-24409685.26460 8756	-24060196226916 0.75	-61
0	0.1088928439816 486	0.0237101144518 4824	-211618707.7658 3248	-223340976.4032 2065	-44831100936028 890	-49

Conclusiones

A continuación, se presentan las conclusiones del trabajo realizado, haciendo énfasis especial en el cumplimiento de los objetivos. El objetivo general de este trabajo era estimar el precio de las viviendas en Colombia basados en técnicas de aprendizaje de máquina usando diferentes características a partir de un dataset con información de viviendas de los años 2019 y 2020. Esto se logró utilizando una base de datos de viviendas en Colombia. De esta base de datos se extrajeron características con las que se implementaron diversos algoritmos de aprendizaje de máquina para estimar los precios de las viviendas en Colombia. Además, la implementación de varios algoritmos de aprendizaje de máquina supervisado permitió obtener diferentes desempeños y encontrar que a mayor complejidad del algoritmo se lograban mejores resultados. Siguiendo esta lógica se abre la ruta para desarrollar proyectos futuros con algoritmos más robustos, como lo son las redes neuronales o implementar aprendizaje profundo, también el extraer diferentes características y el uso de diferentes técnicas de selección de estas, esto abre una posibilidad a obtener mejores desempeños en la estimación de los precios de las viviendas en Colombia. Esto puede llevarse a cabo con las diferentes máquinas disponibles en Deepnote, las cuales tienen las características para desarrollar actividades que tengan un costo computacional alto, disminuyendo así los tiempos de procesamiento requeridos para el proyecto. Los hallazgos más relevantes son los siguientes: - La mayoría de las viviendas se encuentran ubicadas en 3 ciudades principales del país como lo son Bogotá, Medellín y Cali. - El tipo de vivienda que predomina por más del doble de viviendas ofrecidas actualmente son los apartamentos. - El rango de precios de vivienda más recurrente va de los 200 millones a los 300 millones de pesos. - La influencia de Bogotá y Cartagena es evidente en el precio medio del metro cuadrado a nivel nacional. - La técnica de Machine Learning con mejor desempeño es Random Forest. Las oportunidades a futuro son las siguientes: - Realizar una mejor selección de características para generar los modelos. - Realizar una optimización de hiperparametros más exhaustiva. - Emplear técnicas de Machine Learning

más robustas para buscar mejores desempeños. - Evaluar el desempeño con diferentes métricas a las utilizadas en este proyecto.