



**UNIVERSIDAD
DE ANTIOQUIA**

Facultad de Ingeniería

PREDICCIÓN DE PRECIOS DE CASAS

Integrantes:

Andrés Felipe Cerón Muñoz

Esteban Agudelo Casas

Profesor:

Raul Ramos Pollan

Universidad de Antioquia

Facultad de ingeniería

Medellín – Colombia

2022

INTRODUCCIÓN

Dadas las características de una vivienda (zona de clasificación, dimensiones, forma, acceso, etc), se predecirá el precio de venta de esta en el mercado. Se implementará el dataset de kaggle (<https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data>), que tiene 1459 muestras (casas) y columnas (MSSubClass, MSZoning, LotShape, Utilities, LotConfig, Neighborhood, entre otras).

Como métrica de Machine Learning se usará la raíz del error cuadrático medio (RMSE) entre el logaritmo del valor predicho y el logaritmo del precio de venta observado. (Tomar registros significa que los errores al predecir casas caras y casas baratas afectarán el resultado por igual). Si el error de los precios predichos por el algoritmo creado es superior al 20% en comparación con los precios reales de las casas, el modelo no será puesto en producción, ya que tendría poca confiabilidad para predecir y por ende, las ventas de las casas no tendrían un aumento notable.

1. Planteamiento del problema

Para las compañías que desempeñan sus labores comerciales con propiedad raíz, tal como la venta y alquiler de casas, es importante conocer el precio de venta de una propiedad tanto para la empresa y el cliente, ya que se tendrá de entrada se conocerá el presupuesto con el cual se debe contar a la hora de cerrar un trato. Es por todo esto, que se desea desarrollar un modelo que permita predecir el precio final de una propiedad teniendo en cuenta todas las variables que determinan ese precio final.

1.1. Dataset

El dataset proviene de una competencia de Kaggle, donde se proporcionan datos históricos de más de 1400 casas en los Estados Unidos, con múltiples variables tales como: Área, localización, vecindario, dormitorios, utilidades, año de construcción, comodidades, y muchas más). El dataset viene con archivos .csv divididos en train y test, los cuales sirven para entrenar el modelo de predicción y probarlo respectivamente.

1.2. Métrica

Como métrica de Machine Learning se usará la raíz del error cuadrático medio (RECM). La RECM es la raíz cuadrada del promedio de errores cuadrados. El efecto de cada error en la RECM es proporcional al tamaño del error cuadrado; por lo tanto, los errores mayores tienen un efecto desproporcionadamente grande en la RECM. Por lo tanto, la RECM es sensible a los valores atípicos.

Se calcula mediante la siguiente expresión:

$$RECM = \sqrt{\frac{\sum_{t=1}^T (\hat{y} - y_t)^2}{T}}$$

La RECM de los valores predichos \hat{y} para t veces la regresión de la variable dependiente y_t con variables observadas T veces, se calcula para T diferentes predicciones como la raíz cuadrada de la media de los cuadrados de las desviaciones.

1.3. Variable Objetivo

La variable objetivo que se desea predecir, es la última columna de los datos train.csv, la cual corresponde a 'SalePrice'. Se estudiará cuales son las variables principales que se relacionan con el resultado objetivo.

2. Sondeo inicial

Se mostrarán las librerías que se usaron y la carga de los datos correspondientes.

2.1. Librerías

A continuación, se muestran las librerías que se usaron para el procedimiento mencionado anteriormente:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from scipy.stats import norm
from sklearn.model_selection import train_test_split
```

Figura 1. Librerías

2.2. Carga de archivos a utilizar

Se procede a realizar la carga de los datos de entrenamiento train.csv y prueba test.csv para el modelo de predicción, en este caso se muestra en la *Figura 2* las primeras 5 filas de los datos de entrenamiento:

entrenamiento.head()																	
	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12

5 rows × 81 columns

Figura 2. Datos entrenamiento.

Se informan de las dimensiones de los DataFrames:

Dimensión train.csv: (1460, 81)
Dimensión test.csv: (1459, 80)

Figura 3. Dimensiones.

3. Análisis de datos

En este apartado se hace un análisis profundo sobre el comportamiento de la variable objetivo, distribuciones, relaciones con las demás variables independientes. También, se hace la reducción de variables que no influyan considerablemente en el modelo, el filtrado y limpieza general de los datos a trabajar.

3.1. Reduciendo variables:

Ya que la variable que se intenta predecir es "SalePrice", se informa sobre ella usando el método describe (), el cual, entrega datos relevantes, como lo son: Cantidad de datos, promedio, desviación estándar, etc.

Ya que la variable que interesa analizar es el precio de venta "SalePrice", se verifica que no tenga valores vacíos, en caso tal de que los haya, se remueven.

```

entrenamiento["SalePrice"].describe()

count      1460.000000
mean       180921.195890
std        79442.502883
min        34900.000000
25%        129975.000000
50%        163000.000000
75%        214000.000000
max        755000.000000
Name: SalePrice, dtype: float64

```

Figura 4. Descripción SalePrice.

Para ver la forma (distribución) de los datos del precio de venta de las casas, se procede a graficar un histograma:

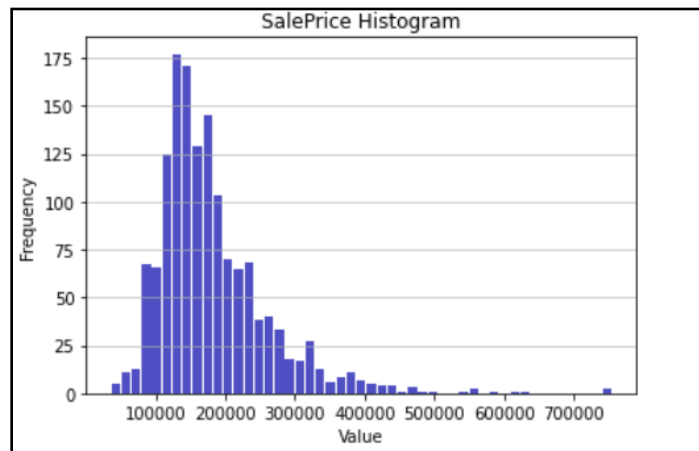


Figura 5. Histograma SalePrice.

Ya que hay muchas variables en este ejercicio, se procede a ver las variables que tengan una mayor correlación con SalePrice, para así, tener a estas en cuenta a la hora de proceder al entrenamiento para predecir el precio. Para ver la correlación de las variables, se crea una matriz de correlación.

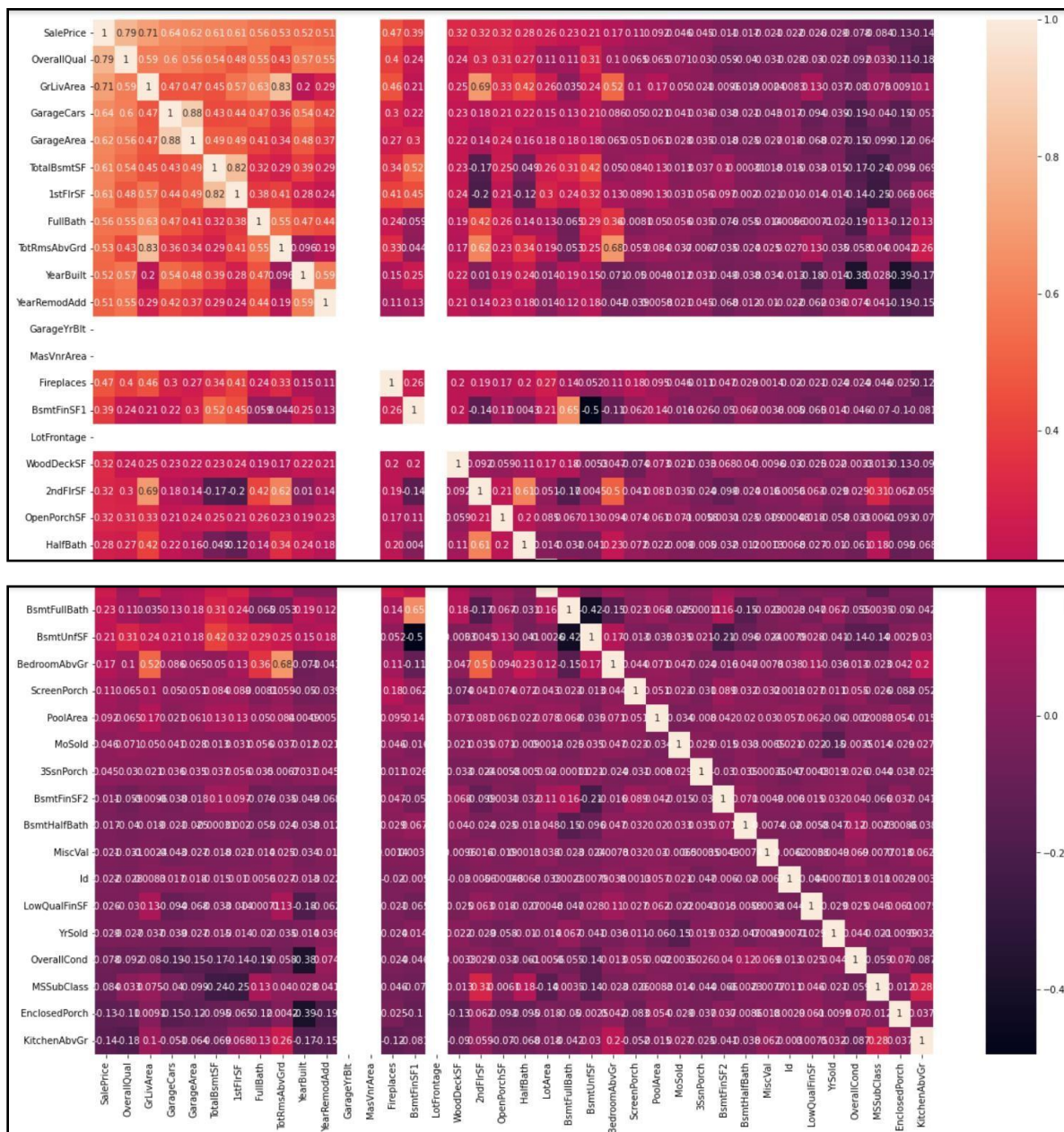


Figura 6. Matriz de correlación.

Se puede observar en la *Figura 6*. los valores de los coeficientes correlación de Pearson correspondientes a la columna "SalePrice". Es una matriz simétrica respecto a la diagonal unitaria.

También, se puede observar de forma mas clara en el siguiente dataframe:

SalePrice	1.000000
OverallQual	0.790982
GrLivArea	0.708624
GarageCars	0.640409
GarageArea	0.623431
TotalBsmstSF	0.613581
1stFlrSF	0.605852
FullBath	0.560664
TotRmsAbvGrd	0.533723
YearBuilt	0.522897
YearRemodAdd	0.507101
GarageYrBlt	0.486362
MasVnrArea	0.477493
Fireplaces	0.466929
BsmstFinSF1	0.386420
LotFrontage	0.351799
WoodDeckSF	0.324413
2ndFlrSF	0.319334
OpenPorchSF	0.315856
HalfBath	0.284108
LotArea	0.263843
BsmstFullBath	0.227122
BsmstUnfSF	0.214479
BedroomAbvGr	0.168213
ScreenPorch	0.111447
PoolArea	0.092404
MoSold	0.046432
3SsnPorch	0.044584
BsmstFinSF2	-0.011378
BsmstHalfBath	-0.016844
MiscVal	-0.021190
Id	-0.021917
LowQualFinSF	-0.025606
YrSold	-0.028923
OverallCond	-0.077856
MSSubClass	-0.084284
EnclosedPorch	-0.128578
KitchenAbvGr	-0.135907

Figura 7. Dataframe de correlación.

A continuación, se crea una nueva variable que almacena el nuevo nivel de correlación positivo mínimo para ser considerado en el modelo. En este caso se deciden variables con un valor de correlación de al menos el 0.3

Se grafica la variable SalePrice junto con las variables que más se correlacionan a ella, tal como se podrá observar, todas estas presentan una correlación positiva, es decir, si una de las variables aumenta, la otra también lo hace:

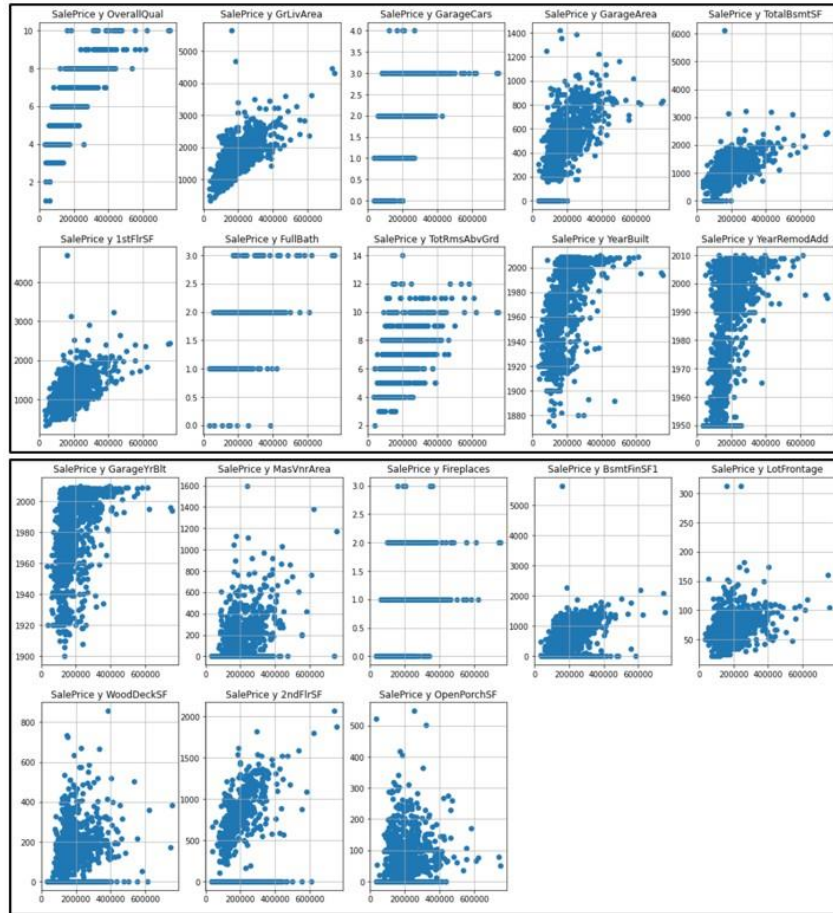


Figura 8. Gráfica correlación.

Se puede observar la variable “SalePrice” en el eje de las x y la otra variable relacionada en el eje y. De izquierda a derecha, disminuye el nivel de correlación de Pearson, siendo la variable independiente “OverallQual” la que más incide en el precio final de la vivienda.

3.2 Eliminación y llenado de datos vacíos

A continuación, se informa acerca de la cantidad de datos vacíos por cada variable independiente y el porcentaje equivalente respecto a la cantidad total de datos respectivamente. Esto para los datos de entrenamiento y prueba.

Se elimina la variable que tenga mayor cantidad de datos nulos en el caso de los datos de entrenamiento y prueba y posteriormente, se llenan las siguientes de mayor cantidad con el promedio asociado a la respectiva variable.

Datos de entrenamiento:

	Cant. Nulos	Porcentaje
LotFrontage	259	0.177397
GarageYrBlt	81	0.055479
MasVnrArea	8	0.005479
FullBath	0	0.000000
OpenPorchSF	0	0.000000
WoodDeckSF	0	0.000000
GarageArea	0	0.000000
GarageCars	0	0.000000
Fireplaces	0	0.000000
TotRmsAbvGrd	0	0.000000
GrLivArea	0	0.000000
OverallQual	0	0.000000
2ndFirSF	0	0.000000
1stFirSF	0	0.000000
TotalBsmntSF	0	0.000000
BsmntFinSF1	0	0.000000
YearRemodAdd	0	0.000000
YearBuilt	0	0.000000
SalePrice	0	0.000000

Figura 9. Datos nulos de entramiento.

Eliminando la variable correspondiente a la primera fila:

	Cant. Nulos	Porcentaje
GarageYrBlt	81	0.055479
MasVnrArea	8	0.005479
OverallQual	0	0.000000
TotRmsAbvGrd	0	0.000000
OpenPorchSF	0	0.000000
WoodDeckSF	0	0.000000
GarageArea	0	0.000000
GarageCars	0	0.000000
Fireplaces	0	0.000000
FullBath	0	0.000000
YearBuilt	0	0.000000
GrLivArea	0	0.000000
2ndFirSF	0	0.000000
1stFirSF	0	0.000000
TotalBsmntSF	0	0.000000
BsmntFinSF1	0	0.000000
YearRemodAdd	0	0.000000
SalePrice	0	0.000000

Figura 10. Datos nulos de entrenamiento.

Datos de prueba:

	Cant. Nulos	Porcentaje
LotFrontage	227	0.155586
GarageYrBlt	78	0.053461
MasVnrArea	15	0.010281
GarageArea	1	0.000685
BsmtFinSF1	1	0.000685
TotalBsmtSF	1	0.000685
GarageCars	1	0.000685
TotRmsAbvGrd	0	0.000000
WoodDeckSF	0	0.000000
Fireplaces	0	0.000000
GrLivArea	0	0.000000
FullBath	0	0.000000
OverallQual	0	0.000000
2ndFlrSF	0	0.000000
1stFlrSF	0	0.000000
YearRemodAdd	0	0.000000
YearBuilt	0	0.000000
OpenPorchSF	0	0.000000

Figura 11. Datos nulos de prueba.

Eliminando la variable correspondiente a la primera fila:

GarageYrBlt	78
MasVnrArea	15
BsmtFinSF1	1
TotalBsmtSF	1
GarageArea	1
GarageCars	1
OverallQual	0
TotRmsAbvGrd	0
WoodDeckSF	0
Fireplaces	0
GrLivArea	0
FullBath	0
YearBuilt	0
2ndFlrSF	0
1stFlrSF	0
YearRemodAdd	0
OpenPorchSF	0

Figura 12. Datos nulos de prueba.

A las 2 primeras filas mostradas la *Figura 10* y *Figura 12*, se completan esa cantidad de datos faltantes teniendo en cuenta el promedio asociado a la respectiva variable.

```
df_entrenamiento['GarageYrBlt'] = df_entrenamiento['GarageYrBlt'].fillna(df_entrenamiento['GarageYrBlt'].mean())
df_entrenamiento['MasVnrArea'] = df_entrenamiento['MasVnrArea'].fillna(df_entrenamiento['MasVnrArea'].mean())

df_prueba['GarageYrBlt'] = df_prueba['GarageYrBlt'].fillna(df_prueba['GarageYrBlt'].mean())
df_prueba['MasVnrArea'] = df_prueba['MasVnrArea'].fillna(df_prueba['MasVnrArea'].mean())
```

Figura 13. Datos finales de entrenamiento y prueba.

3.3. Normalización

Tal como se verá a continuación en la *Figura 14a y 14b*, los datos de la columna SalePrice del DataFrame, no se pueden representar mediante una distribución de probabilidad normal:

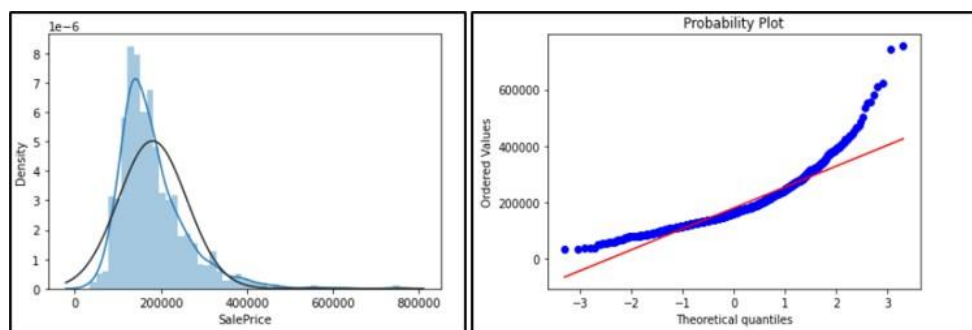


Figura 14a y 14b. Función probabilidad.

En la *Figura 14a y 14b*, los datos de la columna SalePrice del DataFrame (gráfico azul), no sigue la distribución de probabilidad normal (gráfico rojo), por tanto, para tener un mejor entrenamiento de la variable, se debe encontrar una distribución que represente bien el comportamiento de los datos. En la [siguiente página](#) se encuentra que para transformar distribuciones con sesgo positivo (como es nuestro caso), la transformación logarítmica es la más usada, puesto que en la escala logarítmica, la distancia es exactamente la misma entre 1 y 10 que entre 10 y 100 o 100 y 1000, etc. Lo que resulta en que la parte izquierda se expandirá, mientras que la parte derecha se comprimirá, lo que favorecerá a la curva resultante para que se ajuste mejor a una normal. En la *Figura 15a y 15b*, se observa la normalización llevada a cabo.

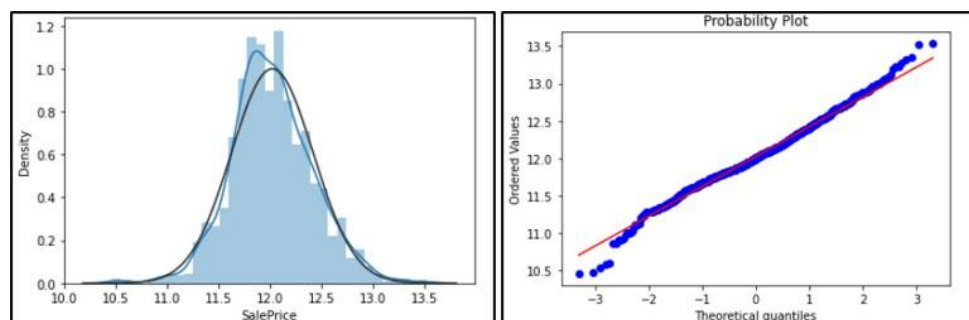


Figura 15a y 15b. Función probabilidad normalizada.

3.4. Variables Dummies

En la [siguiente página](#) se habla acerca de lo que son las variables Dummies o indicadoras, las cuales sirven para identificar las categorías a las cuales pertenecen las observaciones, estas variables pueden tomar valores de 0 o de 1. Por tanto, transformaremos el DataFrame que se tiene en el momento (sin datos faltantes) a variables Dummies.

```
df_entrenamiento = pd.get_dummies(df_entrenamiento)
df_prueba = pd.get_dummies(df_prueba)
```

Figura 16. Dummies.

Se verifica si quedó algún dato faltante en los datos de prueba:

Porcentaje Faltante	
BsmtFinSF1	0.06854
TotalBsmtSF	0.06854
GarageCars	0.06854
GarageArea	0.06854

Figura 17. Datos de prueba.

Esto concuerda según la *Figura 12*, donde se tenía 1 dato vacío por cada variable faltante respectivamente. Estos datos vacíos, al ser solo 1 por cada variable, se procede a llenar con un cero.

```
for col in ( 'GarageArea', 'GarageCars'):  
    df_prueba[col] = df_prueba[col].fillna(0)  
for col in ('BsmtFinSF1','TotalBsmtSF'):  
    df_prueba[col] = df_prueba[col].fillna(0)
```

Figura 18. Llenado de datos de prueba faltantes.

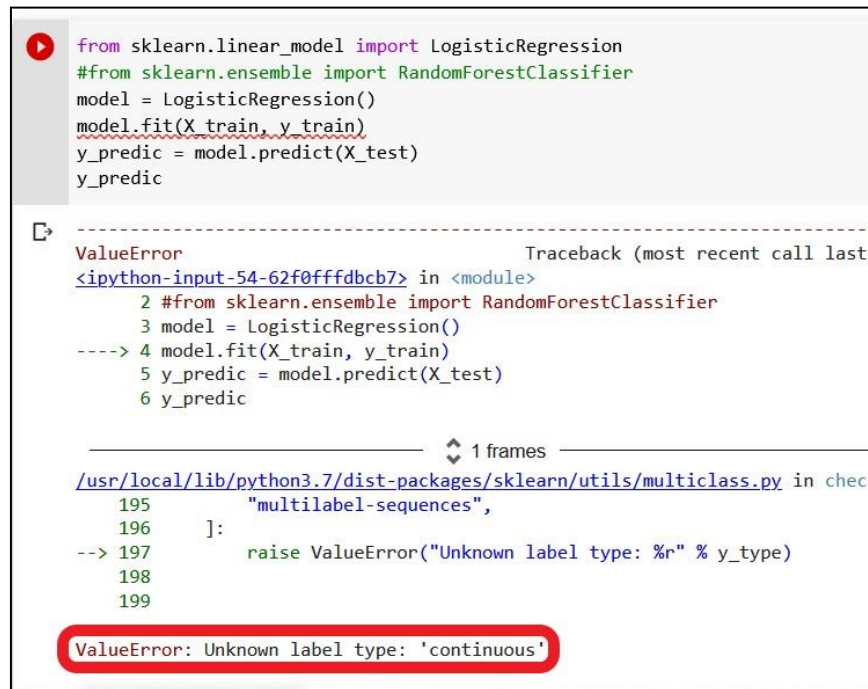
Se procede a seleccionar los datos de entrenamiento que se usarán para entrenar el modelo, y el porcentaje correspondiente a la prueba, como se muestra en la *Figura 19*

```
y_train = df_entrenamiento.SalePrice.values  
df_entrenamiento.drop("SalePrice", axis = 1, inplace = True)  
test_pct = 0.2  
# Con la siguiente línea, de los datos df_entrenamiento_3 y de y_train, serán divididos aleatoriamente de  
# tal forma que el 20% corresponderá a test y el 80% a train  
X_train, X_test, y_train, y_test = train_test_split(df_entrenamiento, y_train, test_size=test_pct)
```

Figura 19. Porcentaje de datos para test y train.

4. Modelos

4.1. Estimador lineal



```
from sklearn.linear_model import LogisticRegression
#from sklearn.ensemble import RandomForestClassifier
model = LogisticRegression()
model.fit(X_train, y_train)
y_predic = model.predict(X_test)
y_predic
```

ValueError Traceback (most recent call last)
<ipython-input-54-62f0fffdcb7> in <module>
 2 #from sklearn.ensemble import RandomForestClassifier
 3 model = LogisticRegression()
----> 4 model.fit(X_train, y_train)
 5 y_predic = model.predict(X_test)
 6 y_predic

----- 1 frames -----
/usr/local/lib/python3.7/dist-packages/sklearn/utils/multiclass.py in check
 195 "multilabel-sequences",
 196]:
--> 197 raise ValueError("Unknown label type: %r" % y_type)
 198
 199

ValueError: Unknown label type: 'continuous'

Figura 20. Error en modelo lineal no continuo.

Tal como se puede apreciar se genera un error ya que, el estimador que se intentó utilizar se usa para predicciones lineales y no continuas. A continuación se investigaron algunos estimadores que se utilizan para realizar predicciones tipo continuas, como lo es para nuestro ejercicio.

4.2 Modelos de decisión y gradientes

Los modelos de decisión, como el random forest, son modelos predictivos formados por reglas binarias (si/no) con las que se consigue repartir las observaciones en función de sus atributos y predecir así el valor de la variable respuesta. Son modelos que tienen grandes ventajas, ya que permite encontrar relaciones lineales y no lineales, y no necesitan que se cumpla una distribución específica en los datos de cada variable. Los métodos estadísticos y de machine learning basados en árboles engloban a un conjunto de técnicas supervisadas no paramétricas que consiguen segmentar el espacio de los predictores en regiones simples, dentro de las cuales es más sencillo manejar las interacciones. Es esta característica la que les proporciona gran parte de su potencial.

A continuación se muestran algunos modelos de predicción que se usaron para la predicción de la variable interés:

```

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor, BaggingRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.linear_model import Ridge, RidgeCV
from sklearn.linear_model import ElasticNet, ElasticNetCV
from sklearn.svm import SVR
from mlxtend.regressor import StackingCVRegressor
import lightgbm as lgb
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import mean_squared_error

import warnings
warnings.filterwarnings('ignore')

```

Figura 21. Otros modelos de predicción.

Se llaman los estimadores que se van a usar:

```

# Instantiate DecisionTreeRegressor
from sklearn.tree import DecisionTreeRegressor
DTR = DecisionTreeRegressor(random_state = 100)

# Instantiate Gradient Boosting Regression
from sklearn.ensemble import GradientBoostingRegressor
params = {'n_estimators': 150, 'max_depth': 5, 'min_samples_split': 2,
          'learning_rate': 0.05, 'loss': 'ls'}

GBR = GradientBoostingRegressor(**params)

# Instantiate svr
from sklearn.svm import SVR
svr = SVR(kernel = 'rbf')

# Instantiate random forest regressor
from sklearn.ensemble import RandomForestRegressor
RFR = RandomForestRegressor(n_estimators = 100, random_state = 0)

# Instantiate XGBRegressor
import xgboost as xgb
XGBR = xgb.XGBRegressor(colsample_bytree=0.4603, gamma=0.0468,
                        learning_rate=0.05, max_depth=3,
                        min_child_weight=1.7817, n_estimators=2200,
                        reg_alpha=0.4640, reg_lambda=0.8571,
                        subsample=0.5213, silent=1,
                        random_state =7, nthread = -1)

# Instantiate LightGBM
import lightgbm as lgb
LGBMR = lgb.LGBMRegressor(objective='regression',num_leaves=5,
                          learning_rate=0.1, n_estimators=500,
                          max_bin = 55, bagging_fraction = 0.8,
                          bagging_freq = 5, feature_fraction = 0.2319,
                          feature_fraction_seed=9, bagging_seed=9,
                          min_data_in_leaf =6, min_sum_hessian_in_leaf = 11)

# Instantiate Lasso
from sklearn.linear_model import Lasso
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
lasso = make_pipeline(RobustScaler(), Lasso(alpha =0.0005, random_state=1))

```

Figura 22. Modelos elegidos.

Se crea una lista que contiene los estimadores anteriormente creados:

```

classifiers = [
    ('DecisionTreeRegressor', DTR),
    ('Gradient Boosting Regression', GBR),
    ('XGBRegressor', XGBR),
    ('Lasso', lasso),
    ('RandomForestRegressor', RFR),
    ('Support Vector Machine Regression', svr),
    ('LightGBM', LGBMR)
]

```

Figura 23. Lista de estimadores.

Para calcular el error de cada estimador, se importan las siguientes librerías, además, se itera con los elementos de la lista para calcular los datos predichos usando cada estimador y calculando su respectivo error, en este caso, la raíz del error cuadrático medio.

```

from sklearn.metrics import accuracy_score
from sklearn import metrics
# Se itera sobre la lista de clasificadores creada
for nombre_modelo, modelo in classifiers:
    # Se fitea el modelo al conjunto de entrenamiento
    modelo.fit(X_train, y_train)

    # Se predice y
    y_predic = modelo.predict(X_test)
    y_predic = y_predic.reshape(-1,1)
    # Se calcula la exactitud con el RMSE
    error = np.sqrt(metrics.mean_squared_error(y_test, y_predic))
    # Se evalúa la exactitud del modelo en conjunto con los datos de prueba
    print('{:s} RMSE : {:.3f}'.format(nombre_modelo, error))

DecisionTreeRegressor RMSE : 0.214
Gradient Boosting Regression RMSE : 0.142
XGBRegressor RMSE : 0.143
Lasso RMSE : 0.145
RandomForestRegressor RMSE : 0.148
Support Vector Machine Regression RMSE : 0.169
LightGBM RMSE : 0.146

```

Figura 24. Errores de estimadores.

Tal como se puede observar, tanto el Gradient Boosting Regression y el LightGBM, presentan el mínimo error (RMSE), por tanto, trabajaremos con ellos. Después de correr varias veces el programa, se verificó que estos dos siempre presentaron el menor error con respecto a los otros, sin embargo, el que presentó un menor error entre estos dos fue principalmente el Gradient Boosting Regression, por tanto, se usará un 80% de los resultados del Gradient Boosting Regression y un 20% de los resultados del LightGBM.

```

GBR.fit(X_train, y_train)

GradientBoostingRegressor(learning_rate=0.05, loss='ls', max_depth=5,
                           n_estimators=150)

LGBMR_predic = np.expml(LGBMR.predict(df_prueba.values))
GBR_predic = np.expml(GBR.predict(df_prueba.values))
Estimador_final_predic = LGBMR_predic*0.2 + GBR_predic*0.8

Estimador_final_predic

array([126302.39800281, 147291.89853217, 182948.97285168, ...,
       165825.56140037, 116886.61789612, 239442.61249864])

```

Figura 25. Estimador usado.

Finalmente, se convierte el resultado obtenido a un dataframe para mejor visualización:

```
df_Estimador_final = pd.DataFrame(Estimador_final_predic, columns=['SalePrice'])
indice = pd.DataFrame(ID_prueba, columns=['Id'])
resultado1 = pd.concat([indice, df_Estimador_final], axis=1)
resultado1.head(20)
```

Figura 26. Código dataframe.

	Id	SalePrice
0	1461	128537.253264
1	1462	151743.983200
2	1463	169541.461615
3	1464	185056.218681
4	1465	191188.049498
5	1466	180545.988483
6	1467	172171.341848
7	1468	164839.509988
8	1469	185428.817223
9	1470	122857.187534
10	1471	188983.919076
11	1472	90230.770401
12	1473	98028.995309
13	1474	156528.353808
14	1475	129929.622427
15	1476	380075.168440
16	1477	248422.284569
17	1478	310700.926557
18	1479	234307.183663
19	1480	493120.929678

Figura 27. Resultado obtenido.

5. Retos y consideraciones de despliegue

Para poder evaluar el desempeño del modelo, se mide el error de los precios predichos por el algoritmo creado, si este error es superior al 20% en comparación con los precios reales de las casas, el modelo no será puesto en producción, ya que tendría poca confiabilidad para predecir y por ende, las ventas de las casas no tendrían un aumento notable. Si el modelo permite tener un ahorro o un mayor control del precio de las viviendas por parte de las compañías de bienes raíces, el modelo podría estar listo para desplegarse. Es necesario estar alimentando el modelo, ya sea con datos almacenados en una nube, que permita estar en constante ajuste al modelo, o en constante entrenamiento del modelo, dado el caso que el error supere el porcentaje anteriormente mencionado.

6. Conclusiones

- Se hace necesario realizar un análisis detallado de los datos, un buen filtrado y correcta eliminación de posibles datos erróneos que puedan afectar el resultado final.
- Es importante conocer la posible distribución del resultado esperado, clasificar correctamente cada variable independiente y tratar siempre de encontrar las variables más influyentes en la predicción.
- Los modelos de decisión ofrecen ventajas respecto a otros, en el sentido de que permiten medir otro tipo de relaciones entre variables, y el parámetro no solamente puede ser numérico, también puede ser categórico.