

Entrega final de proyecto

POR:

Luisa María Castro Cortez

MATERIA:

Introducción a la inteligencia artificial

PROFESOR:

Raul Ramos Pollan



UNIVERSIDAD DE ANTIOQUIA FACULTAD DE INGENIERÍA
MEDELLÍN 2023

Contenido

1. Planteamiento del problema	3
1.1 Dataset.....	3
1.2 Métricas.....	6
2. Exploración de variables	7
2.1 Análisis de la variable objetivo SalePrice	7
2.2 Descubrimiento de los tipos de datos	9
2.3 Correlación de variables.....	11
2.4 Distribución de las variables numéricas	15
3. Tratamiento de datos	15
3.1 Eliminación de las columnas con muchos datos faltantes.....	15
3.2 Relleno de datos faltantes	16
3.3 Transformación de variables categóricas	16
4. Métodos supervisados	17
4.1 Selección de modelos	17
4.2 Identificación de los mejores hiperparámetros del modelo.....	19
5. Métodos no supervisados	20
5.1 PCA y RandomForest.....	21
5.2 NMF y Árbol de Decisión.....	21
6. Curvas de aprendizaje	22
6.1 Método supervisado	22
6.2 Método no supervisado	22
7. Retos y condiciones de despliegue del modelo	23
8. Conclusiones	24
9. Bibliografía	24

1. Planteamiento del problema

Si nos ponemos en la tarea de preguntar a un comprador de vivienda que nos describa la casa de sus sueños probablemente no empezará por la altura del techo o la proximidad a una estación del metro. Pero existen muchas variables que influyen en las negociaciones sobre el precio de venta de una propiedad. Es por esto por lo que se desea desarrollar un modelo que permita estimar el precio de venta de las viviendas, basado en las variables que pueden presentar las viviendas, al tener mejores estimaciones, se podrá generar un mayor interés por parte de los compradores de vivienda al momento de buscar una nueva propiedad.

1.1 Dataset

El dataset a utilizar proviene de una competencia de kaggle en la cual se proporcionan datos con 79 variables explicativas que describen aspectos de las viviendas residenciales en Ames, Iowa. El dataset este compuesto por un conjunto de archivos .csv y .txt que proporcionan la información requerida.

El archivo que contiene los datos de las viviendas es nombrado data_description y contiene la siguiente información:

- SalePrice - the property's sale price in dollars. This is the target variable that you're trying to predict.
- MSSubClass: The building class
- MSZoning: The general zoning classification
- LotFrontage: Linear feet of street connected to property
- LotArea: Lot size in square feet
- Street: Type of road access
- Alley: Type of alley access
- LotShape: General shape of property
- LandContour: Flatness of the property
- Utilities: Type of utilities available
- LotConfig: Lot configuration
- LandSlope: Slope of property
- Neighborhood: Physical locations within Ames city limits
- Condition1: Proximity to main road or railroad
- Condition2: Proximity to main road or railroad (if a second is present)
- BldgType: Type of dwelling
- HouseStyle: Style of dwelling
- OverallQual: Overall material and finish quality
- OverallCond: Overall condition rating
- YearBuilt: Original construction date
- YearRemodAdd: Remodel date
- RoofStyle: Type of roof
- RoofMatl: Roof material
- Exterior1st: Exterior covering on house
- Exterior2nd: Exterior covering on house (if more than one material)
- MasVnrType: Masonry veneer type
- MasVnrArea: Masonry veneer area in square feet
- ExterQual: Exterior material quality
- ExterCond: Present condition of the material on the exterior
- Foundation: Type of foundation
- BsmtQual: Height of the basement
- BsmtCond: General condition of the basement
- BsmtExposure: Walkout or garden level basement walls
- BsmtFinType1: Quality of basement finished area
- BsmtFinSF1: Type 1 finished square feet
- BsmtFinType2: Quality of second finished area (if present)
- BsmtFinSF2: Type 2 finished square feet
- BsmtUnfSF: Unfinished square feet of basement area
- TotalBsmtSF: Total square feet of basement area

- Heating: Type of heating
- HeatingQC: Heating quality and condition
- CentralAir: Central air conditioning
- Electrical: Electrical system
- 1stFlrSF: First Floor square feet
- 2ndFlrSF: Second floor square feet
- LowQualFinSF: Low quality finished square feet (all floors)
- GrLivArea: Above grade (ground) living area square feet
- BsmtFullBath: Basement full bathrooms
- BsmtHalfBath: Basement half bathrooms
- FullBath: Full bathrooms above grade
- HalfBath: Half baths above grade
- Bedroom: Number of bedrooms above basement level
- Kitchen: Number of kitchens
- KitchenQual: Kitchen quality
- TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)
- Functional: Home functionality rating
- Fireplaces: Number of fireplaces
- FireplaceQu: Fireplace quality
- GarageType: Garage location
- GarageYrBlt: Year garage was built
- GarageFinish: Interior finish of the garage
- GarageCars: Size of garage in car capacity
- GarageArea: Size of garage in square feet
- GarageQual: Garage quality
- GarageCond: Garage condition
- PavedDrive: Paved driveway
- WoodDeckSF: Wood deck area in square feet
- OpenPorchSF: Open porch area in square feet
- EnclosedPorch: Enclosed porch area in square feet
- 3SsnPorch: Three season porch area in square feet
- ScreenPorch: Screen porch area in square feet
- PoolArea: Pool area in square feet
- PoolQC: Pool quality
- Fence: Fence quality
- MiscFeature: Miscellaneous feature not covered in other categories
- MiscVal: \$Value of miscellaneous feature
- MoSold: Month Sold
- YrSold: Year Sold

- SaleType: Type of sale
- SaleCondition: Condition of sale

train.csv - el conjunto de entrenamiento test.csv

- el conjunto de prueba sample_submission.csv

- un envío de referencia de una regresión lineal

sobre el año y el mes de la venta, los metros

cuadrados del lote y el número de dormitorios.

1.2 Métricas

La métrica de evaluación principal para el modelo será error cuadrático medio (RMSE) entre el logaritmo del valor predicho y el logaritmo del precio de venta observado. (Tomar los logaritmos significa que los errores en la predicción de casas caras y casas baratas afectarán por igual al resultado). el cual se calcula mediante la siguiente expresión:

$$\epsilon = \sqrt{\frac{\sum_{i=1}^n (P_i - O_i)^2}{n}}$$

Donde ϵ es el valor del RMSE, n es el número total de observaciones en el dataset, p_i es la predicción de la variable objetivo y O_i es el valor real de la variable objetivo.

Por otra parte, en cuanto a la métrica de negocio, se tiene interés en que las predicciones sean lo suficientemente confiables para saber precio de venta de las viviendas. Con esta información se pueden realizar análisis financieros

para determinar qué tan viables pueden ser comprar ciertas viviendas en particular.

2. Exploración de variables

2.1 Análisis de la variable objetivo SalePrice

Para empezar con la exploración de variables, se analiza el comportamiento de la distribución que tiene la variable objetivo SalePrice, dicho comportamiento se muestra en la Figura 1, donde se puede observar que la variable objetivo tiene una asimetría hacia la derecha. Por lo que se aplica una transformación logarítmica, cuyo resultado se muestra en la Figura 2.

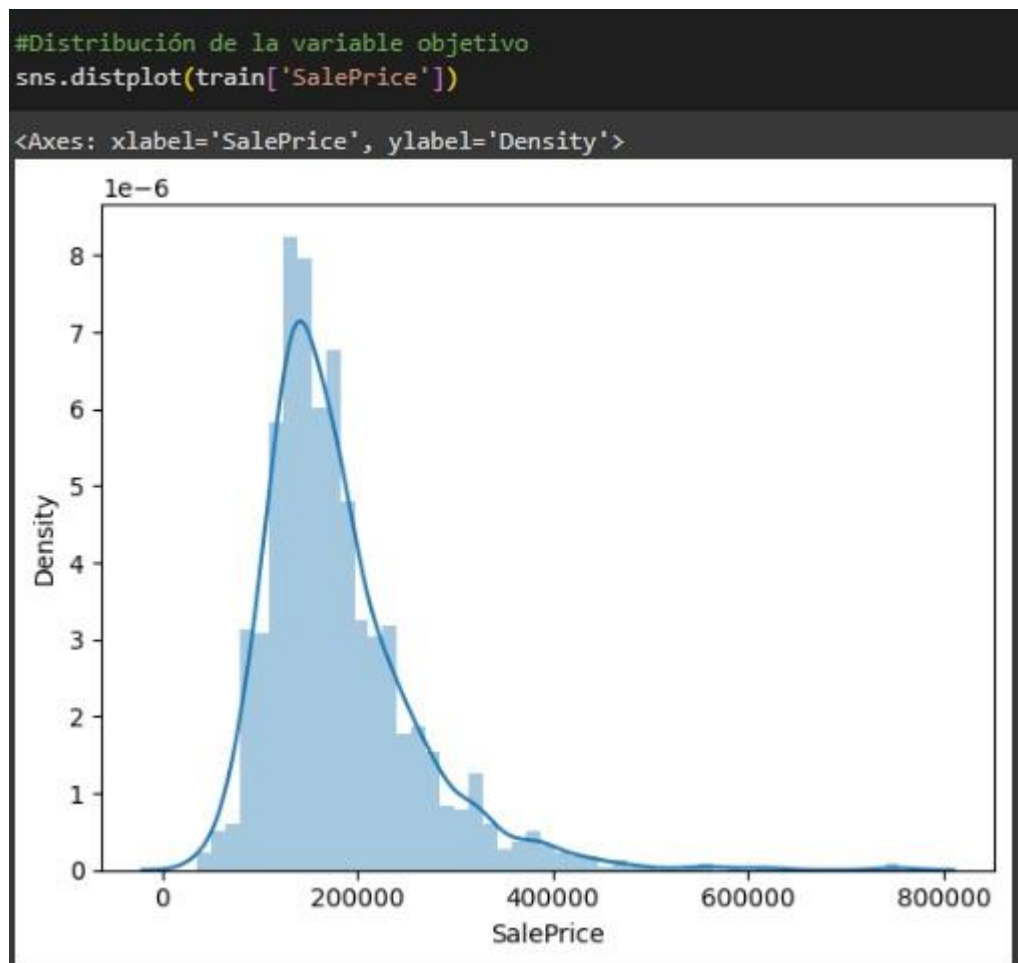


Figura 1. Distribución de la variable objetivo.

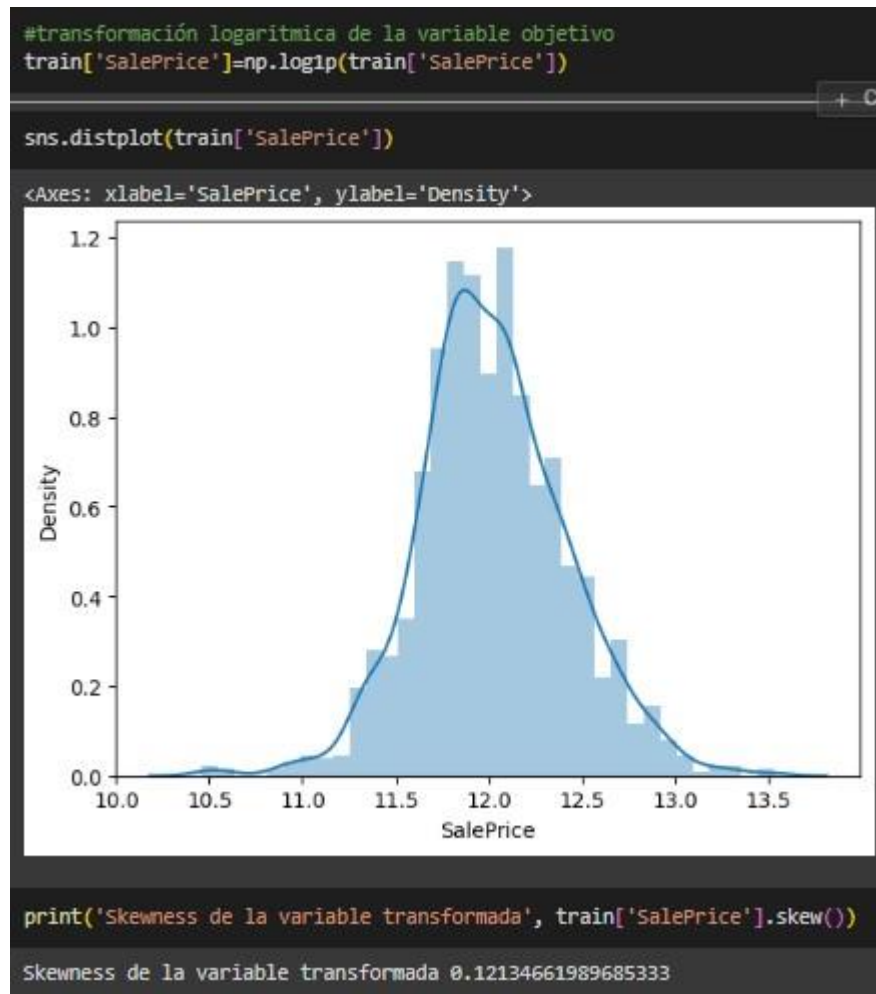


Figura 2. Distribución de la variable objetivo después de la transformación logarítmica.

En la Figura 2, se puede observar que la distribución de la variable objetivo luego de la transformación logarítmica ya tiene un comportamiento más adecuado para realizar un análisis, y es esta variable transformada la que se usará en el entrenamiento y pruebas de los algoritmos. Cabe destacar que la asimetría o skewness de la variable objetivo antes de la transformación era de 1.88, y el valor luego de la transformación es de 0.12, lo cual refleja una gran reducción.

2.2 Descubrimiento de los tipos de datos

```
train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
```

#	COLUMN	NON-NULL	COUNT	DTYPE
0	Id	1460	non-null	int64
1	MSSubClass	1460	non-null	int64
2	MSZoning	1460	non-null	object
3	LotFrontage	1201	non-null	float64
4	LotArea	1460	non-null	int64
5	Street	1460	non-null	object
6	Alley	91	non-null	object
7	LotShape	1460	non-null	object
8	LandContour	1460	non-null	object
9	Utilities	1460	non-null	object
10	LotConfig	1460	non-null	object
11	LandSlope	1460	non-null	object
12	Neighborhood	1460	non-null	object
13	Condition1	1460	non-null	object
14	Condition2	1460	non-null	object
15	BldgType	1460	non-null	object
16	HouseStyle	1460	non-null	object
17	OverallQual	1460	non-null	int64
18	OverallCond	1460	non-null	int64
19	YearBuilt	1460	non-null	int64
20	YearRemodAdd	1460	non-null	int64
21	RoofStyle	1460	non-null	object
22	RoofMatl	1460	non-null	object
23	Exterior1st	1460	non-null	object
24	Exterior2nd	1460	non-null	object
25	MasVnrType	1452	non-null	object
26	MasVnrArea	1452	non-null	float64

27	ExterQual	1460	non-null	object
28	ExterCond	1460	non-null	object
29	Foundation	1460	non-null	object
30	BsmtQual	1423	non-null	object
31	BsmtCond	1423	non-null	object
32	BsmtExposure	1422	non-null	object
33	BsmtFinType1	1423	non-null	object
34	BsmtFinSF1	1460	non-null	int64
35	BsmtFinType2	1422	non-null	object
36	BsmtFinSF2	1460	non-null	int64
37	BsmtUnfSF	1460	non-null	int64
38	TotalBsmtSF	1460	non-null	int64
39	Heating	1460	non-null	object
40	HeatingQC	1460	non-null	object
41	CentralAir	1460	non-null	object
42	Electrical	1459	non-null	object
43	1stFlrSF	1460	non-null	int64
44	2ndFlrSF	1460	non-null	int64
45	LowQualFinSF	1460	non-null	int64
46	GrLivArea	1460	non-null	int64
47	BsmtFullBath	1460	non-null	int64
48	BsmtHalfBath	1460	non-null	int64
49	FullBath	1460	non-null	int64
50	HalfBath	1460	non-null	int64
51	BedroomAbvGr	1460	non-null	int64
52	KitchenAbvGr	1460	non-null	int64
53	KitchenQual	1460	non-null	object
54	TotRmsAbvGrd	1460	non-null	int64
55	Functional	1460	non-null	object
56	Fireplaces	1460	non-null	int64
57	FireplaceQu	770	non-null	object
58	GarageType	1379	non-null	object
59	GarageYrBlt	1379	non-null	float64
60	GarageFinish	1379	non-null	object
61	GarageCars	1460	non-null	int64

62	GarageArea	1460	non-null	int64
63	GarageQual	1379	non-null	object
64	GarageCond	1379	non-null	object
65	PavedDrive	1460	non-null	object
66	WoodDeckSF	1460	non-null	int64
67	OpenPorchSF	1460	non-null	int64
68	EnclosedPorch	1460	non-null	int64
69	3SsnPorch	1460	non-null	int64
70	ScreenPorch	1460	non-null	int64
71	PoolArea	1460	non-null	int64
72	PoolQC	7	non-null	object
73	Fence	281	non-null	object
74	MiscFeature	54	non-null	object
75	MiscVal	1460	non-null	int64
76	MoSold	1460	non-null	int64
77	YrSold	1460	non-null	int64
78	SaleType	1460	non-null	object
79	SaleCondition	1460	non-null	object
80	SalePrice	1460	non-null	float64

Tabla 1 Información de las variables

De las 79 variables que tenemos en cuenta para realizar la predicción del SalePrice el 48% son numéricas y el 52% son categóricas las cuales tendremos que transformar en el proceso de tratamiento de datos. De las 79 variables, 19 variables tienen valores nulos, los cuales tendremos que limpiar antes de aplicar los modelos.

2.3 Correlación de variables

La Tabla 2 muestra los valores de correlación que existen entre las diferentes variables con la variable objetivo. Se puede observar que OverallQual, GrLivArea, GarageCars, GarageArea son las variables que tienen la mayor correlación.

También se puede observar que las variables MiscVal, OverallCond, YrSold, LowQualFinSF, MSSubClass, KitchenAbvGr, EnclosedPorch, tienen una correlación tan baja que puede decirse que no están relacionadas con la variable objetivo.

SALEPRICE	
OVERALLQUAL	0.81718461

GRLIVAREA	0.70092699
GARAGECARS	0.68062487
GARAGEAREA	0.65088768
TOTALBSMTSF	0.61213423
1STFLRSF	0.59698132
FULLBATH	0.59477066
YEARBUILT	0.58657019
YEARREMODADD	0.56560778
GARAGEYRBLT	0.54107278
TOTRMSABVGRD	0.5344224
FIREPLACES	0.48944955
MASVNRAREA	0.43080896
BSMTFINSF1	0.37202325
LOTFRONTAGE	0.35587862
WOODDECKSF	0.33413517
OPENPORCHSF	0.32105325
2NDFLRSF	0.31930014
HALFBATH	0.31398222
LOTAREA	0.25732007
BSMTFULLBATH	0.23622416
BSMTUNFSF	0.22198516
BEDROOMABVGR	0.20904343
SCREENPORCH	0.12120759
POOLAREA	0.0697979
MOSOLD	0.0573295
3SSNPORCH	0.0549002
BSMTFINSF2	0.00483229
BSMTHALFBATH	- 0.00514924
MISCVAL	- 0.02002082
OVERALLCOND	- 0.03686845
YRSOLD	- 0.03726291

LOWQUALFINSF	- 0.03796279
MSSUBCLASS	- 0.07395917
KITCHENABVGR	- 0.14754816
ENCLOSEDPORCH	- 0.14905023

Tabla 2 Correlación de variables con la variable objetivo

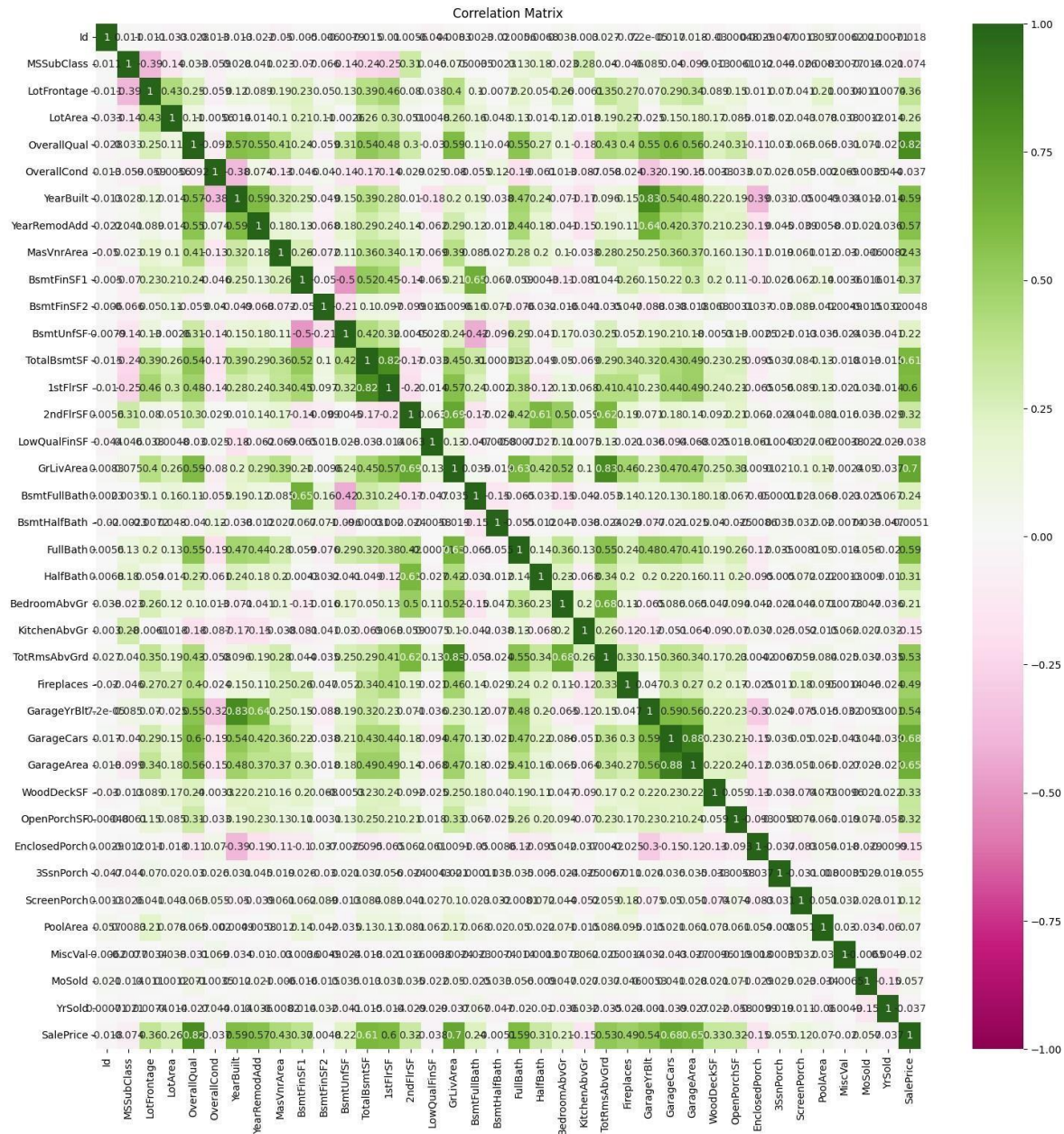


Figura 3 Matriz de correlación

2.4 Distribución de las variables numéricas

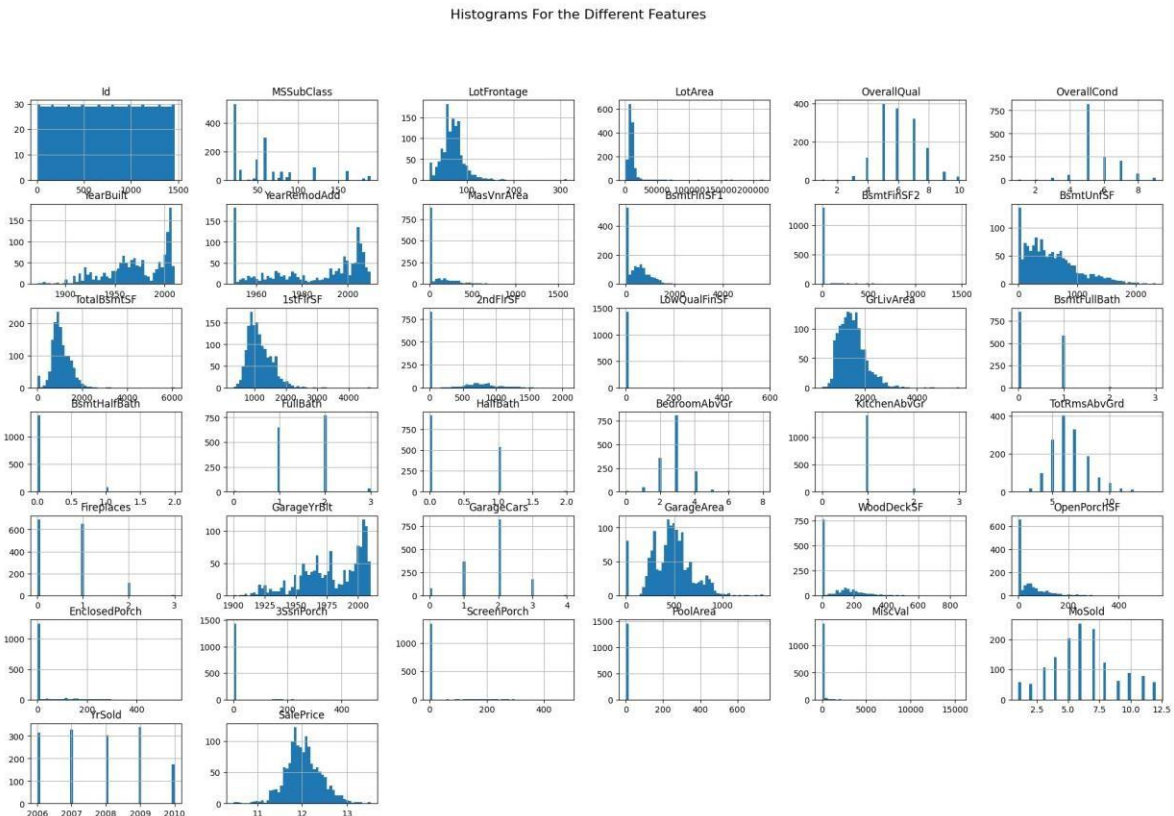


Figura 4 Distribución de variables numéricas

3. Tratamiento de datos

3.1 Eliminación de las columnas con muchos datos faltantes

Como se mostró en la sección anterior, existen variables que tienen datos faltantes. Para este caso se considerará que una variable que tenga más del 50% de datos faltantes será eliminada ya que no aportará la suficiente información al modelo. De esta manera las variables que son eliminadas del dataset son Alley, PoolQC, Fence, MiscFeature.

```
criterio = len(train) * 0.5 #criterio para eliminar la columna (50% de las filas que se tienen)
train.dropna(axis=1, thresh = criterio, inplace = True) #eliminación de las columnas con 50% o más de datos faltantes
print('New Shape of Train Data:',train.shape)

New Shape of Train Data: (1460, 77)

2. Relleno de datos numéricos faltantes

train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 77 columns):
```

Figura 5 Eliminación de variables con datos nulos superiores al 50%

3.2 Relleno de datos faltantes

Para el resto de las variables numéricas que contienen datos faltantes que no son eliminadas rellenan dichos valores nulos con la media de sus datos. La Figura 6 muestra el código usado para rellenar dichos valores.

```
4.2. Relleno de datos numéricos faltantes

train['LotFrontage'].fillna(train.LotFrontage.median(),inplace=True)
train['MasVnrArea'].fillna(train.MasVnrArea.median(),inplace=True)
train['GarageYrBlt'].fillna(train.GarageYrBlt.median(),inplace=True)
```

Figura 6 Relleno de datos numéricos faltantes

3.3 Transformación de variables categóricas

Las variables categóricas no pueden ser usadas en el entrenamiento de un algoritmo, por lo que se debe hacer una transformación para convertirlas a variables numéricas:

```
4.3. Transformación de las variables categoricas

[23] var_categoricas = ['MSZoning','Street','LotShape','LandContour','Utilities','LotConfig
encoder = preprocessing.LabelEncoder()

for i in var_categoricas:

    train[i] = encoder.fit_transform(train[i])

print (train.info())
```



```

63  GarageCond      1460 non-null  int64
64  PavedDrive      1460 non-null  int64
65  WoodDeckSF      1460 non-null  int64
66  OpenPorchSF     1460 non-null  int64
67  EnclosedPorch   1460 non-null  int64
68  3SsnPorch       1460 non-null  int64
69  ScreenPorch     1460 non-null  int64
70  PoolArea        1460 non-null  int64
71  MiscVal         1460 non-null  int64
72  MoSold          1460 non-null  int64
73  YrSold          1460 non-null  int64
74  SaleType        1460 non-null  int64
75  SaleCondition   1460 non-null  int64
76  SalePrice       1460 non-null  int64
dtypes: float64(3), int64(74)

```

Figura 7. Transformación de variables categóricas a variables numéricas

4. Métodos supervisados

4.1 Selección de modelos

4.1.1 Ajuste de la métrica

Como se mencionó en la sección 1.2, la métrica para medir el desempeño de los modelos será el RMSLE (Root Mean Squared Logarithmic Error), sin embargo, como realizamos una transformación logarítmica a la variable objetivo, la métrica que se puede implementar es la RMSE (Root Mean Squared Error), y a este valor sacarle la raíz cuadrada, y de esta forma ya tendríamos el RMSLE. En la Figura 8 se muestra la función mediante la cual se calcula el RMSLE.

```

[25] #Función para calcular el RMSLE de los modelos implementados
def RMSLE(y_actual, y_pred):
    return np.sqrt(mean_squared_error(y_actual, y_pred))

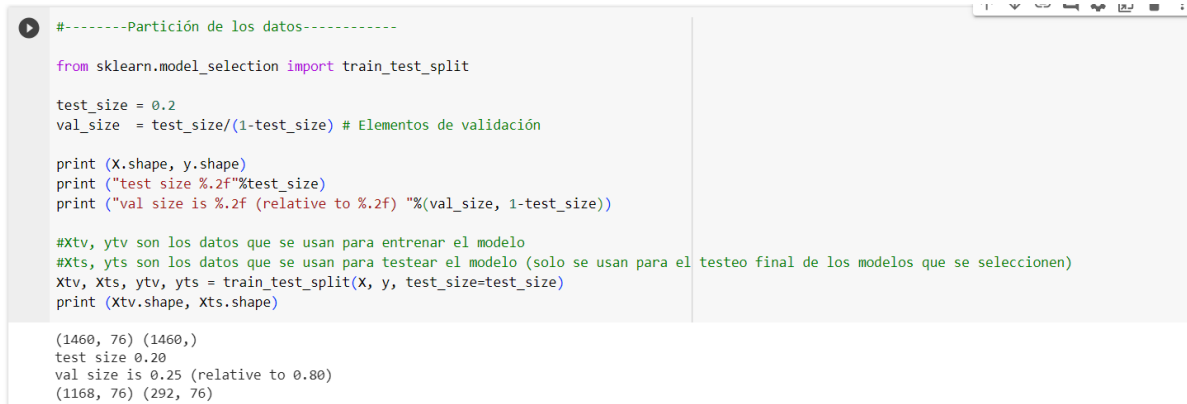
```

Figura 8. Función para calcular el RMSLE

4.1.2 Partición de los datos

Para entrenar los modelos se usó el dataset train. Fue necesario hacer una partición para training y otra para test. Los datos que se guardan para test no fueron utilizados en el entrenamiento. En cuanto a los datos que se usaron para training, se hizo una partición adicional en este conjunto de modo que se tenga una parte para training y la otra parte para hacer la validación. De esta forma la partición de los datos de training se usó en el ajuste del modelo, mientras que los que se guardan para test solo se usaron al final, ya cuando el modelo ha sido entrenado, esto para simular un conjunto de datos completamente nuevos para el algoritmo y

probar así su desempeño. En la Figura 9 se muestra el código implementado para realizar estas particiones. Cabe destacar que para entrenar los algoritmos se decidió eliminar las variables con una correlación menor al 50% ya que no tenían correlación con la variable objetivo. De esta manera se conservan solo aquellas que tienen alta correlación con la variable objetivo.



```
#-----Partición de los datos-----  
  
from sklearn.model_selection import train_test_split  
  
test_size = 0.2  
val_size = test_size/(1-test_size) # Elementos de validación  
  
print (X.shape, y.shape)  
print ("test size %.2f"%test_size)  
print ("val size is %.2f (relative to %.2f) "%(val_size, 1-test_size))  
  
#Xtv, ytv son los datos que se usan para entrenar el modelo  
#Xts, yts son los datos que se usan para testear el modelo (solo se usan para el testeo final de los modelos que se seleccionen)  
Xtv, Xts, ytv, yts = train_test_split(X, y, test_size=test_size)  
print (Xtv.shape, Xts.shape)  
  
(1460, 76) (1460,)  
test size 0.20  
val size is 0.25 (relative to 0.80)  
(1168, 76) (292, 76)
```

Figura 9. Partición de los datos para entrenar los modelos

4.1.3 Selección de modelos

Para desarrollar el análisis se plantearon inicialmente dos modelos: Árbol de decisión y RandomForest. De las dos posibles opciones se realizó una selección inicial para identificar cual daba los mejores resultados. En la Figura 10 se muestra el código implementado para la selección. Cabe resaltar que se implementó una metodología de cross-validation para realizar la selección de los modelos. De este análisis se encontró que el modelo con el mejor desempeño fue el RandomForest con un RMSLE de 0.19 en test y un RMSLE de 0.15 en train.

```
[28] estimator1 = DecisionTreeRegressor(max_depth=5)
     estimator2 = RandomForestRegressor(n_estimators = 2,max_depth = 5)

#Selección de modelos

zscores = []
estimators = [estimator1, estimator2]
for estimator in estimators:
    print("-----")
    z = cross_validate(estimator, Xtv, ytv, return_train_score=True, return_estimator=False,
                      scoring="neg_mean_squared_error", cv=ShuffleSplit(n_splits=10, test_size=val_size))
    report_cv_score(z)
    zscores.append(np.mean(np.sqrt(z['test_score']*(-1))))
best = np.argmin(zscores)
print ("Seleccionado: ", best)
best_estimator = estimators[best]
print ("\n Mejor modelo: ")
print (best_estimator)

-----
RMLSE Test:  0.20453 (± 0.00977043 )
RMLSE Train: 0.15177 (± 0.00326646 )
-----
RMLSE Test:  0.19071 (± 0.01224085 )
RMLSE Train: 0.15205 (± 0.00510834 )
Seleccionado: 1

Mejor modelo:
RandomForestRegressor(max_depth=5, n_estimators=2)
```

Figura 10 Selección del modelo con mejores resultados

4.2 Identificación de los mejores hiperparámetros del modelo

Una vez obtenidos los modelos a trabajar, se hace un estudio para obtener los mejores hiperparametros de las variables. Esto se realiza mediante el módulo GridSearchCV de la librería de `sk.learn.model_selection`. Este módulo permite hacer un análisis variando los hiperparámetros del algoritmo de interés implementando una metodología de cross-validation. Como resultado se obtienen los mejores hiperparámetros entre la selección que se da al módulo arrojando así el mejor estimador. los hiperparametros usados del RandomForest son “n_estimators” y “max_depth”. En la Figura 11 se muestran los valores que se definieron para estos dos parámetros.

```
[30] from sklearn.model_selection import GridSearchCV
```

5.2.2. Random Forest

```
[32] #Esta celda toma un tiempo considerable en completarse
parametros = { 'n_estimators': [5,10,15,20],
               'max_depth':[5,7,9,11]}

forest_reg = GridSearchCV(estimator = estimator2,
                          param_grid = parametros,
                          cv = ShuffleSplit(n_splits= 5, test_size=val_size),
                          scoring = 'neg_mean_squared_error',
                          verbose = 1,
                          return_train_score = True,
                          n_jobs = -1)

forest_reg.fit(Xtv, ytv)
```

Fitting 5 folds for each of 16 candidates, totalling 80 fits

```
GridSearchCV
  estimator: RandomForestRegressor
    RandomForestRegressor
      RandomForestRegressor(max_depth=5, n_estimators=2)
```

```
[33] print("Mejor estimador Random Forest: ",forest_reg.best_estimator_)
      print("Mejores parámetros para el estimador Random Forest: ", forest_reg.best_params_)
```

```
Mejor estimador Random Forest: RandomForestRegressor(max_depth=11, n_estimators=15)
Mejores parámetros para el estimador Random Forest: {'max_depth': 11, 'n_estimators': 15}
```

✓ 0s completed at 10:31 AM

Figure 11. Código para buscar los hiperparametros

Finalmente, del análisis se para el RandomForest los valores para sus hiperparametros, `n_estimators` y `max_depth`, son 15 y 11 respectivamente.

5. Métodos no supervisados

Una vez realizado el análisis con solamente métodos supervisados, se realizó un análisis implementando métodos no supervisados a modo de tratamiento de los datos antes del entrenamiento de los métodos supervisados. En este caso se usaron dos métodos no supervisados: PCA y NMF. De esta manera se podrá reducir la dimensionalidad del dataset y conservar aquellos datos que conserven la mayor cantidad de información y con estos entrenar los algoritmos supervisados.

5.1 PCA y RandomForest

Para implementar el PCA, se realizó un código mediante el cual se exploraron varias opciones de hiperparámetro de este algoritmo. Tomando valores para su hiperparámetros `n_components` de 1,3,5,7 y 9. Se entrenaron varios PCA con estos valores y luego se uso cada uno como el dataset de entrada para el algoritmo RandomForest, y así buscar cual de estos PCA daba el mejor desempeño. Se encontró que el mejor desempeño lo daba aquel modelo con `ncomponents = 7`. La Figura 12 muestra la línea de código implementada para hacer este análisis

```
[54] from sklearn.decomposition import PCA
components = [1,3,5,7,9]
test_size = 0.2
val_size = test_size/(1-test_size)
perf = [] #desempeños de los modelos
Rdm_forest = RandomForestRegressor(n_estimators = 15,max_depth = 11)
for i in components:
    pca = PCA(n_components = i)
    X_t = pca.fit_transform(X)

    #Partición de datos
    #Xtv, ytv son los datos que se usan para entrenar el modelo
    #Xts, yts son los datos que se usan para probar el modelo (solo se usan para el testeo final de los modelos que se seleccionen)
    Xtv, Xts, ytv, yts = train_test_split(X_t, y, test_size=test_size)
    print (Xtv.shape, Xts.shape)

    Rdm_forest.fit(Xtv, ytv)
    perf.append(RMSLE(yts , Rdm_forest.predict(Xts)))
    print("RMSLE del modelo con ", i , 'elementos: ', "{:.5f}".format(RMSLE(yts , Rdm_forest.predict(Xts))))
    print('-----')

print('Mejor RMSLE: ', "{:.5f}".format(np.min(perf)), ' ; obtenido con ', components[np.argmin(perf)], ' componentes para PCA')

(1168, 1) (292, 1)
RMSLE del modelo con 1 elementos: 0.39680
-----
(1168, 3) (292, 3)
RMSLE del modelo con 3 elementos: 0.20235
-----
(1168, 5) (292, 5)
RMSLE del modelo con 5 elementos: 0.21909
-----
(1168, 7) (292, 7)
RMSLE del modelo con 7 elementos: 0.18872
-----
(1168, 9) (292, 9)
RMSLE del modelo con 9 elementos: 0.20480
-----
Mejor RMSLE: 0.18872 ; obtenido con 7 componentes para PCA
```

Figura 12. Exploración del mejor PCA

Una vez obtenido el mejor PCA, se realiza nuevamente el procedimiento descrito anteriormente para obtener los mejores hiperparámetros del RandomForest, usando GridSearchCV. De esta manera se obtiene que para este caso los mejores hiperparámetros son `n_estimators = 15` y `max_depth = 11`.

5.2 NMF y Árbol de Decisión

En cuanto al NMF, el análisis es igual que con PCA. Se buscó el mejor parámetro entre varias opciones dadas. En este caso se obtuvo que el mejor valor para el hiperparámetro de este algoritmo es `n_components = 3`. Usando el dataset transformado con el mejor NMF, se buscó el mejor hiperparámetro para el Árbol de Decisión. Se obtuvo que el mejor hiperparámetro en este caso es `max_depth = 5`. Debido a que el RMSLE de estos modelos superaba el 30% se decidió solo utilizar PCA y RandomFores.

6. Curvas de aprendizaje

6.1 Método supervisado

Para obtener la curva de aprendizaje de los algoritmos se usa el módulo `learning_curves` de la librería de `sk.learn.model_selection`, la cual implementa una metodología de cross-validation para evaluar diferentes desempeños usando diferentes tamaños para el entrenamiento del modelo.

En la Figura 13 se muestra la curva de aprendizaje para el algoritmo RandomForest implementado. En este caso se evidencia una gran brecha entre los datos de training y los datos de test por lo que es necesario reevaluar en caso de que se requiera un error menor en test como disminuir el Bias u overfitting.

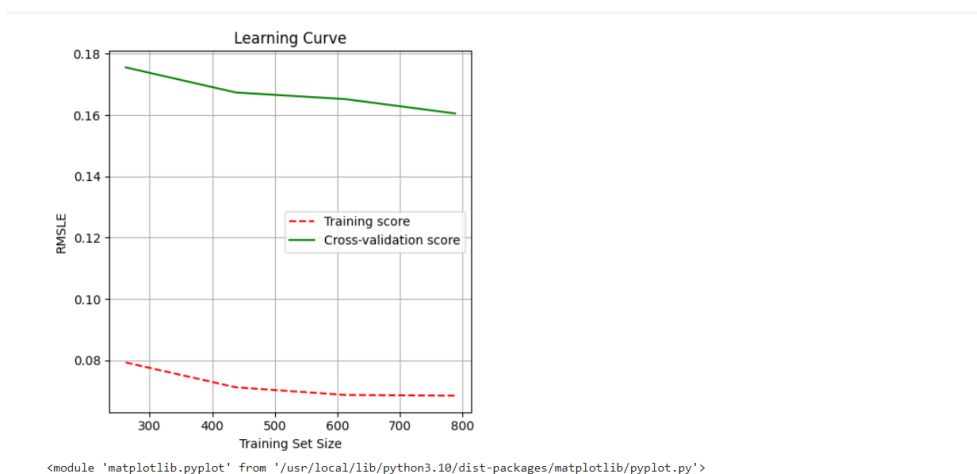
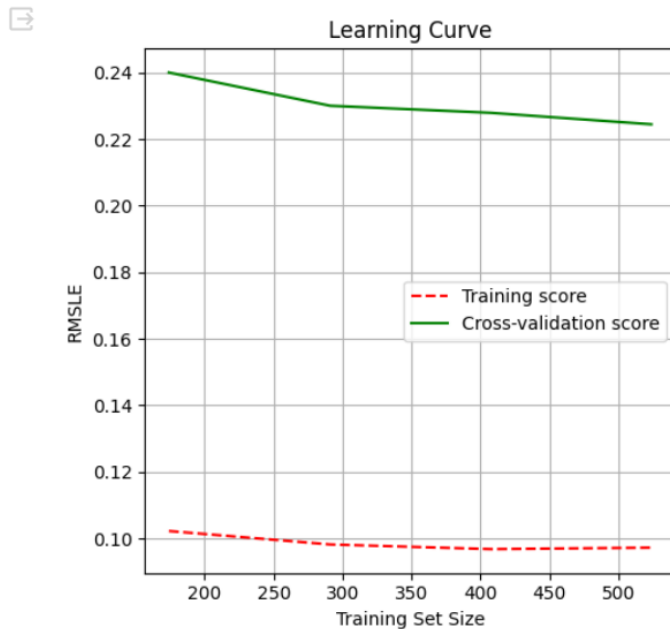


Figura 13 Curva de aprendizaje para el RandomForest

6.2 Método no supervisado

Para obtener la curva de aprendizaje de la combinación de los métodos no supervisados y supervisados, se implementa la misma metodología descrita anteriormente. La Figura 14 muestra la curva de aprendizaje para la combinación de PCA y RandomForest. Se observa que, a pesar de usar más datos para entrenar el modelo, el error en entrenamiento no disminuye por el contrario aumenta en un 1%. El modelo en prueba presenta un comportamiento igual y se ve un incremento de un 4%. Este comportamiento puede deberse a que el modelo está presentando Bias (sesgo). Este error puede tratarse al incrementar la complejidad del modelo o implementando más columnas en el entrenamiento del modelo, por lo que podría explorarse la posibilidad de implementar un PCA con una menor reducción de dimensionalidad.

```
[81] lc_plot(Rdm_forest,Xtv,ytv)
```



<module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>

Figura 14. Curva de aprendizaje PCA+RandomForest.

7. Retos y condiciones de despliegue del modelo

Para evaluar el desempeño mínimo con el cual se consideraría que el modelo genera un beneficio, se debe comparar el precio de venta de las casas generado por las predicciones del modelo con el precio actual del mercado. Si el modelo permite obtener un precio que se encuentre en un rango objetivo mínimo, el cual puede ser establecido por los inversores, gerentes de los equipos de ventas de las compañías de bienes raíces, se puede considerar que el modelo es apto para desplegarse en producción.

Para desplegar el modelo en producción, es necesario contar con una base de datos que almacene la información de las casas especialmente de las variables que tienen mayor correlación con el precio de venta que puedan ser usados los datos para generar las predicciones. Uno de los retos que se tiene para implementar este modelo, es que la información debe ser consistente y se debe actualizar a lo largo del tiempo para que los datos arrojados por el modelo no sean viciados.

8. Conclusiones

Es necesario hacer un análisis detallado de que variables tienen más peso en los modelos entrenados, ya que existen muchas variables que no tiene correlación con la variable objetivo.

Es necesario aumentar la complejidad de algunos modelos ya que se presentan problemas de overfitting.

Para este problema es mejor utilizar modelos supervisados ya que el error es menor al de los modelos no supervisados.

Se puede implementar un modelo de Clustering para analizar los patrones y comportamientos de cada tipo de vivienda.

9. Bibliografía

- Anna Montoya, DataCanary. (2016). House Prices - Advanced Regression Techniques. Kaggle. <https://kaggle.com/competitions/house-pricesadvancedregression-techniques>