House Prices - Advanced Regression Techniques

Javier Rabal

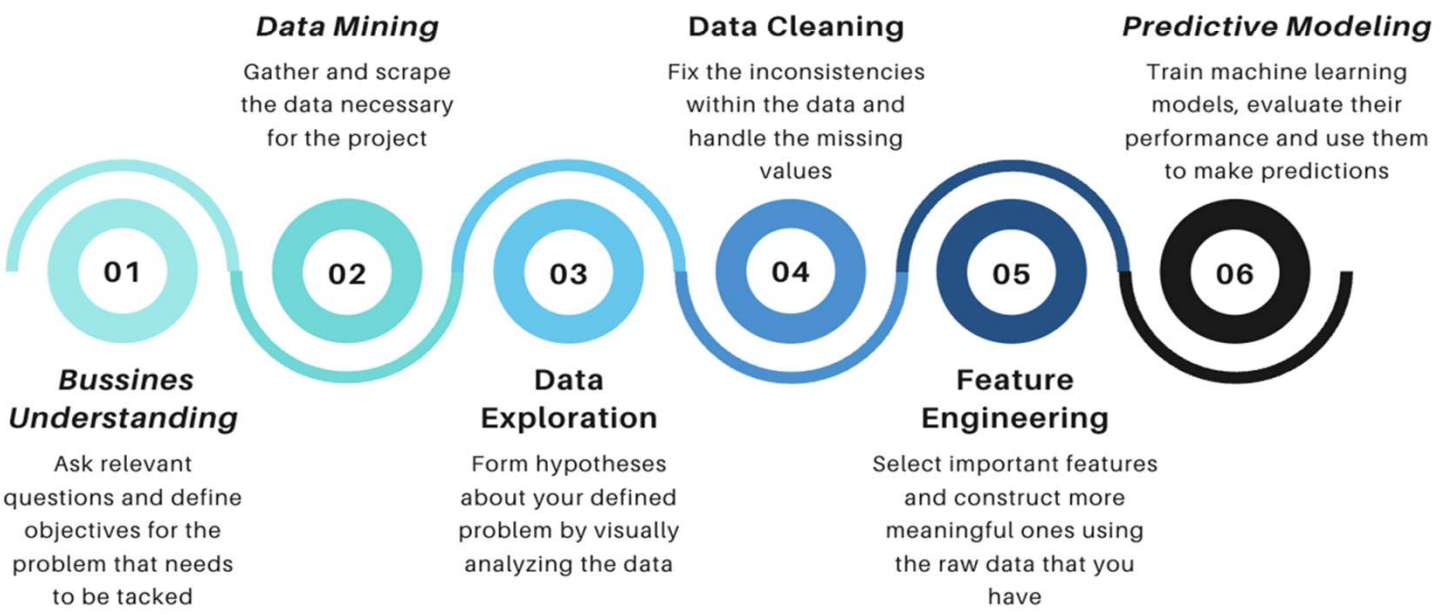
Natalia Sisamón

CESTE

kaggle

INDICE

Data Science Lifecycle



01

BUSSINESS UNDERSTANDING

OBJETIVO	Predecir el precio de cada casa
KPI	RMSE entre el logaritmo del valor real y el valor predicho

02

DATA MINING

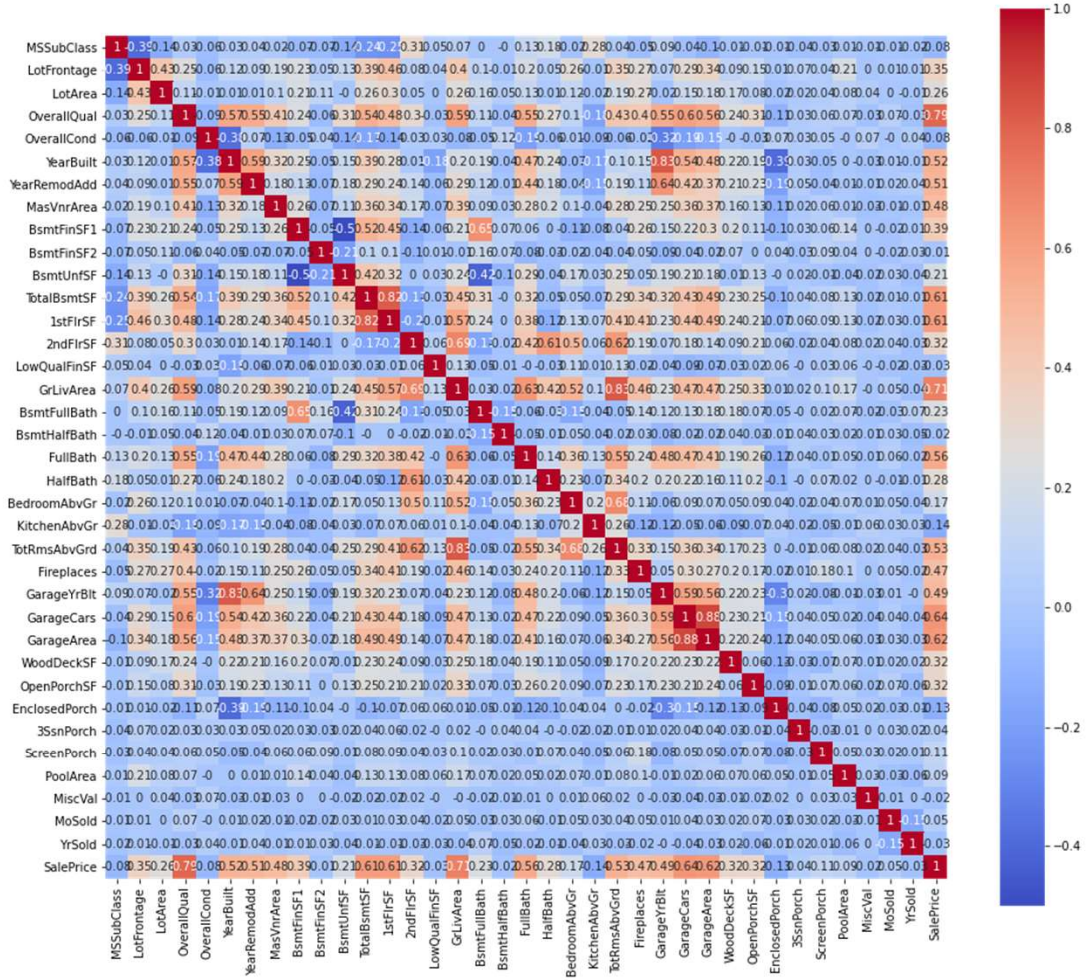
En este caso los datos se descargan directamente de la página de kaggle:

https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data

03

DATA EXPLORATION

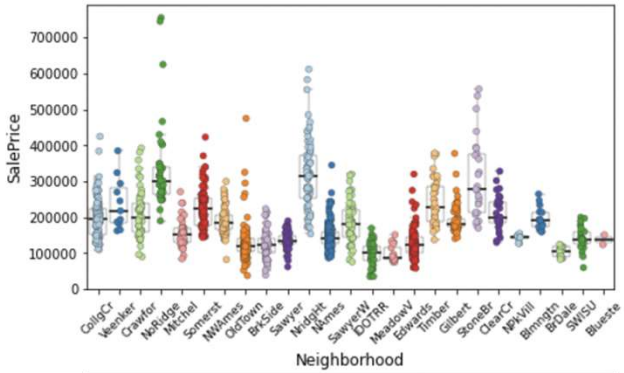
CORRELACIÓN DE VARIABLES

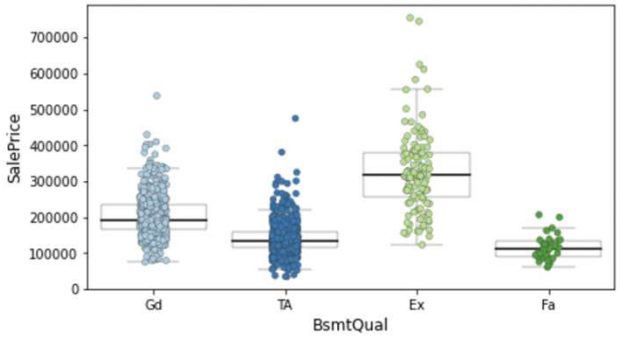


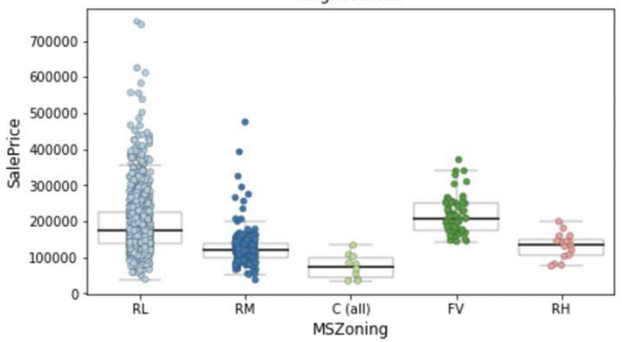
03

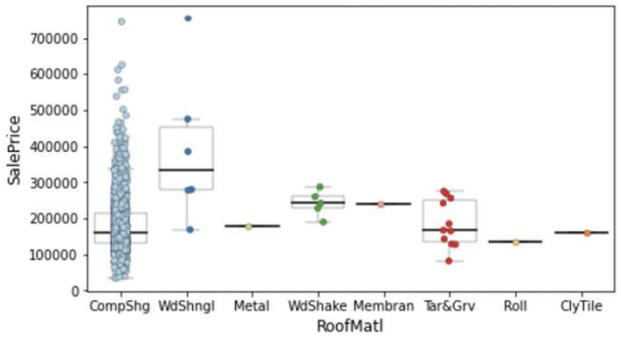
DATA EXPLORATION

VARIABLES CATEGÓRICAS





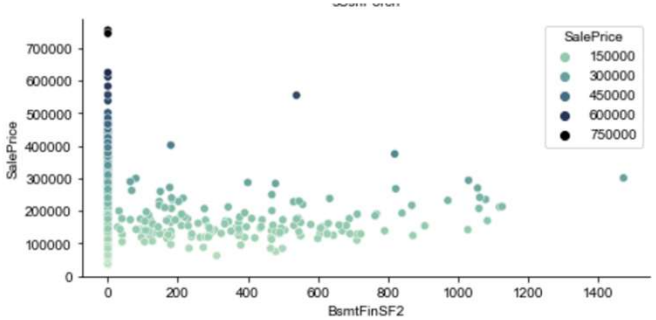


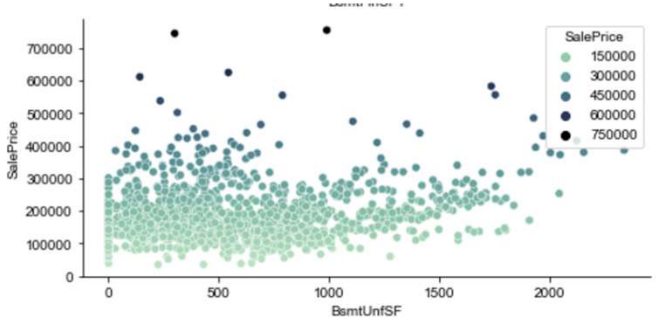


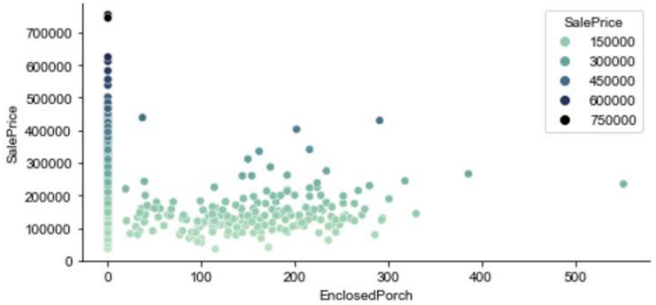
03

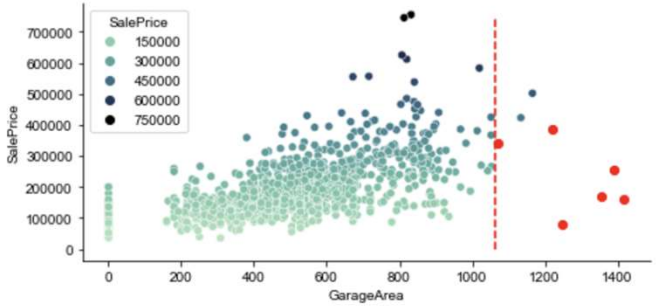
DATA EXPLORATION

VARIABLES NUMÉRICAS





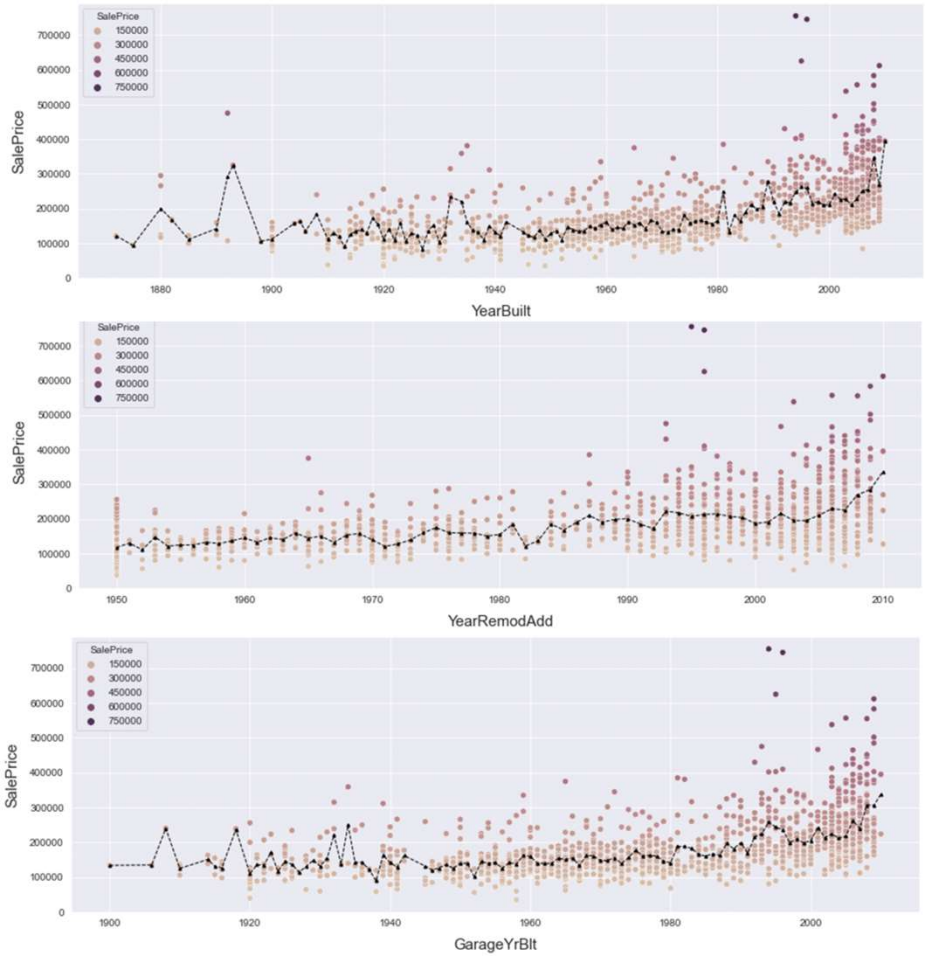




03

DATA EXPLORATION

VARIABLES TEMPORALES



03

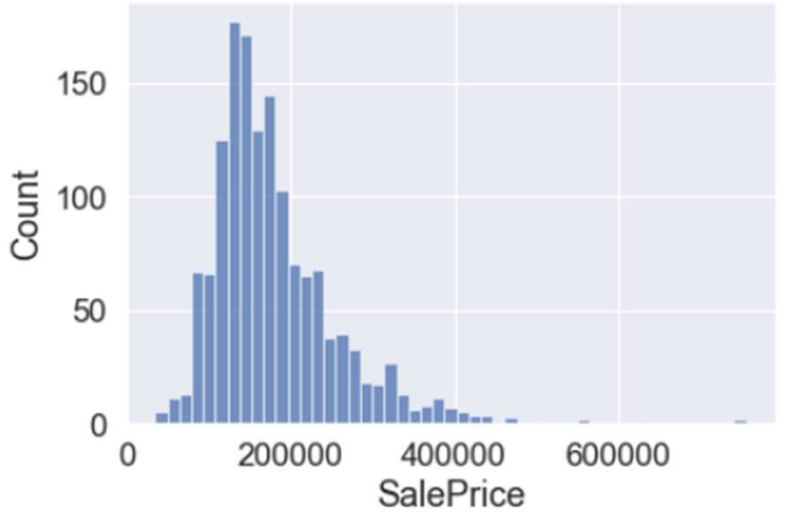
DATA EXPLORATION

VARIABLE TARGET

```
df_train['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
```

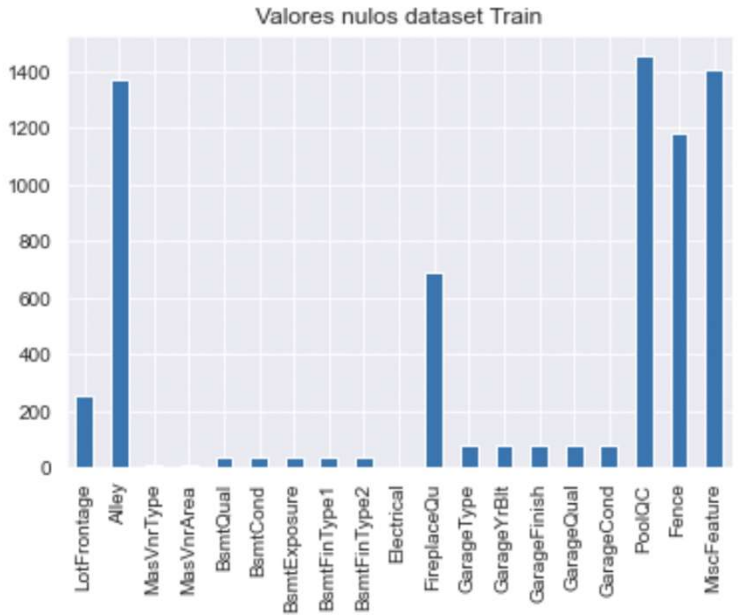
```
sns.histplot(df_train['SalePrice']);
```

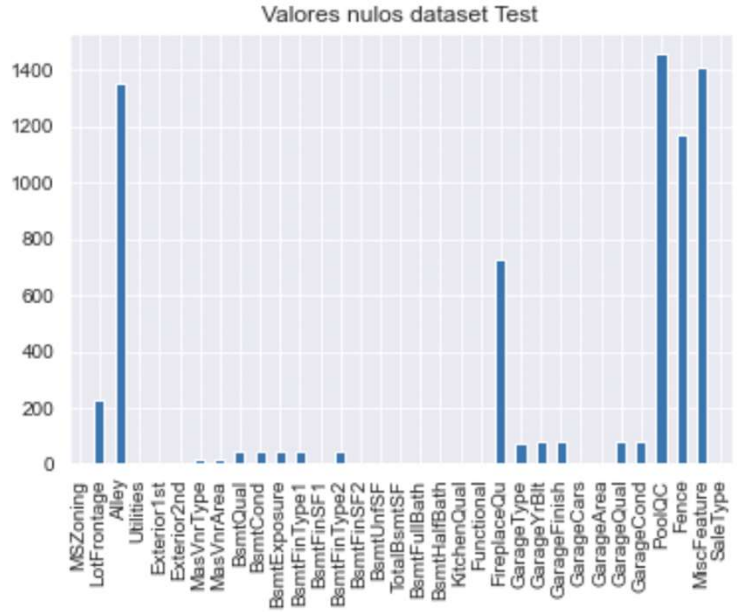


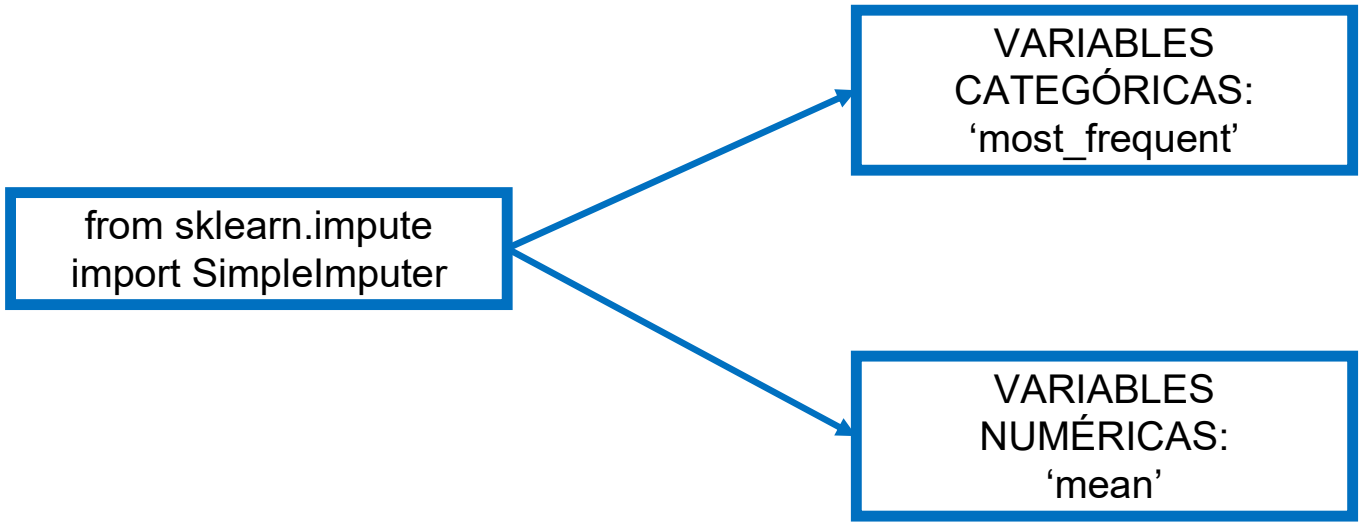
04

DATA CLEANING

VALORES NULOS | **PRUEBAS**






from sklearn.impute
import SimpleImputer
VARIABLES
CATEGÓRICAS:
‘most_frequent’
VARIABLES
NUMÉRICAS:
‘mean’

```
# se cambian los valores ausentes por el valor "No_Fireplace"
df["FireplaceQu"] = df["FireplaceQu"].fillna("No_Fireplace")
# se cambian los valores ausentes por el valor "No_Garage"
df["GarageCond"] = df["GarageCond"].fillna("No_Garage")
# se cambian los valores ausentes por el valor "No_Garage"
df["GarageType"] = df["GarageType"].fillna("No_Garage")
# se cambian los valores ausentes por el valor "No_Garage"
df["GarageFinish"] = df["GarageFinish"].fillna("No_Garage")
# se cambian los valores ausentes por el valor "No_Garage"
df["GarageQual"] = df["GarageQual"].fillna("No_Garage")
# se cambian los valores ausentes por el valor "No_Basement"
df["BsmtExposure"] = df["BsmtExposure"].fillna("No_Basement")
# se cambian los valores ausentes por el valor "No_Basement"
df["BsmtFinType2"] = df["BsmtFinType2"].fillna("No_Basement")
# se cambian los valores ausentes por el valor "No_Basement"
df["BsmtCond"] = df["BsmtCond"].fillna("No_Basement")
# se cambian los valores ausentes por el valor "No_Basement"
df["BsmtQual"] = df["BsmtQual"].fillna("No_Basement")
# se cambian los valores ausentes por el valor "No_Basement"
df["BsmtFinType1"] = df["BsmtFinType1"].fillna("No_Basement")
```

04

DATA CLEANING

VALORES NULOS

MEJOR RESULTADO

```
# Se cambian los 8 valores nulos por el valor 'None' que es el más veces se repite
df["MasVnrType"] = df["MasVnrType"].fillna("None")
# Se cambia el valor nulo por el valor 'SBrkr' que es el más veces se repite
df["Electrical"] = df["Electrical"].fillna("SBrkr")
# Se sustituyen los valores nulos por la media de toda la columna
df["LotFrontage"] = df["LotFrontage"].transform(lambda x: x.fillna(x.mean()))
# Se sustituyen los valores nulos por el año de construcción de la casa 'YearBuilt'
df["GarageYrBlt"] = df["GarageYrBlt"].transform(lambda x: x.fillna(df_train["YearBuilt"]))
# Se sustituyen los 8 valores nulos por 0 debido a que en estos casos se ha
#sustituido el valor de la columna 'MasVnrType' por el valor 'None'
df["MasVnrArea"] = df["MasVnrArea"].fillna(0)
```

04

DATA CLEANING

VALORES NULOS

MEJOR RESULTADO

TEST

```
# Se sustituyen los 2 valores nulos por 0 debido a que es el más frecuente
df_test["BsmtFullBath"] = df_test["BsmtFullBath"].fillna(0)
# Se sustituyen los 2 valores nulos por la media de toda la columna
df_test["BsmtHalfBath"] = df_test["BsmtHalfBath"].transform(lambda x: x.fillna(x.mean()))
# Se sustituye el valor nulo por la media de toda la columna
df_test["TotalBsmtSF"] = df_test["TotalBsmtSF"].transform(lambda x: x.fillna(x.mean()))
# Se sustituye el valor nulo por la media de toda la columna
df_test["BsmtUnfSF"] = df_test["BsmtUnfSF"].transform(lambda x: x.fillna(x.mean()))
# Se sustituye el valor nulo por la media de toda la columna
df_test["BsmtFinSF2"] = df_test["BsmtFinSF2"].transform(lambda x: x.fillna(x.mean()))
# Se sustituye el valor nulos por 2 debido a que es el más frecuente
df_test["GarageCars"] = df_test["GarageCars"].fillna(2)
# Se sustituye el valor nulo por la media de toda la columna
df_test["BsmtFinSF1"] = df_test["BsmtFinSF1"].transform(lambda x: x.fillna(x.mean()))
# Se sustituye el valor nulo por la media de toda la columna
df_test["GarageArea"] = df_test["GarageArea"].transform(lambda x: x.fillna(x.mean()))
# Se sustituye los valores nulos por 'RL' debido a que es el más frecuente
df_test["MSZoning"] = df_test["MSZoning"].fillna('RL')
# Se sustituye los valores nulos por 'AllPub' debido a que es el más frecuente
df_test["Utilities"] = df_test["Utilities"].fillna('AllPub')
# Se sustituye los valores nulos por 'Typ' debido a que es el más frecuente
df_test["Functional"] = df_test["Functional"].fillna('Typ')
# Se sustituye los valores nulos por 'VinylSd' debido a que es el más frecuente
df_test["Exterior2nd"] = df_test["Exterior2nd"].fillna('VinylSd')
# Se sustituye los valores nulos por 'TA' debido a que es el más frecuente
df_test["KitchenQual"] = df_test["KitchenQual"].fillna('TA')
# Se sustituye los valores nulos por 'WD' debido a que es el más frecuente
df_test["SaleType"] = df_test["SaleType"].fillna('WD')
# Se sustituye los valores nulos por 'VinylSd' debido a que es el más frecuente
df_test["Exterior1st"] = df_test["Exterior1st"].fillna('VinylSd')
```

04

DATA CLEANING

OUTLIERS → PRUEBAS

```
from sklearn.ensemble
import IsolationForest
```

```
from sklearn.neighbors
import LocalOutlierFactor
```

OUTLIERS → MEJOR RESULTADO

No se realiza limpieza de outliers

CON POCAS CATEGORÍAS

from category_encoders import OneHotEncoder

CON MUCHAS CATEGORÍAS

from category_encoders import TargetEncoder

04

DATA CLEANING

VARIABLES CATEGORICAS

MEJOR RESULTADO

VARIABLES CON ORDEN NUMÉRICO

Se eliminan las variables con una correlación menor a 0.2 con la variable objetivo

```
rating = {'None': 0, 'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5}
ordinal_encoding = {
        'LotShape': {'None': 0, 'Reg': 1, 'IR1': 2, 'IR2': 3, 'IR3': 4},
        'Utilities': {'None': 0, 'ElO': 1, 'NoSeWa': 2, 'NoSeWr': 3, 'AllPub': 4},
        'LandSlope': {'None': 0, 'Gtl': 1, 'Mod': 2, 'Sev': 3},
        'ExterQual': rating,
        'ExterCond': rating,
        'BsmtQual': {'No_Basement': 0, 'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5},
        'BsmtCond': {'No_Basement': 0, 'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5},
        'BsmtExposure': {'No_Basement': 0, 'No': 1, 'Mn': 2, 'Av': 3, 'Gd': 4},
        'BsmtFinType1': {'No_Basement': 0, 'Unf': 1, 'LwQ': 2, 'Rec': 3, 'BLQ': 4, 'ALQ': 5, 'GLQ': 6},
        'BsmtFinType2': {'No_Basement': 0, 'Unf': 1, 'LwQ': 2, 'Rec': 3, 'BLQ': 4, 'ALQ': 5, 'GLQ': 6},
        'HeatingQC': rating,
        'CentralAir': {'None': 0, 'N': 1, 'Y': 2},
        'Electrical': {'None': 0, 'Mix': 1, 'FuseP': 2, 'FuseF': 3, 'FuseA': 4, 'SBrkr': 5},
        'KitchenQual': rating,
        'Functional': {'None': 0, 'Sal': 1, 'Sev': 2, 'Maj2': 3, 'Maj1': 4, 'Mod': 5,
                       'Min2': 6, 'Min1': 7, 'Typ': 8},
        'FireplaceQu': {'No_Fireplace': 0, 'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5},
        'GarageFinish': {'No_Garage': 0, 'Unf': 1, 'RFn': 2, 'Fin': 3},
        'GarageQual': {'No_Garage': 0, 'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5},
        'GarageCond': {'No_Garage': 0, 'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5},
        'PavedDrive': {'None': 0, 'N': 1, 'P': 2, 'Y': 3}
    }
```

```
[('ExterQual', 0.6826392416562593),
 ('KitchenQual', 0.6595997207286625),
 ('BsmtQual', 0.5852071991725201),
 ('GarageFinish', 0.5492467563332124),
 ('FireplaceQu', 0.5204376059504018),
 ('HeatingQC', 0.4276487073988035),
 ('BsmtExposure', 0.37469622100088673),
 ('BsmtFinType1', 0.30490787307063333),
 ('GarageQual', 0.2738390740062235),
 ('GarageCond', 0.2631907844703991),
 ('CentralAir', 0.2513281638401551),
 ('PavedDrive', 0.23135695225722716),
 ('BsmtCond', 0.21260715648557876),
 ('PoolQC', 0.11548430473054794),
 ('Functional', 0.10761889324399436),
 ('Street', 0.04103553550004947),
 ('ExterCond', 0.018899118482413036),
 ('BsmtFinType2', -0.005323160673474943),
 ('LandSlope', -0.05115224817946656),
 ('Fence', -0.146941526435884),
 ('LotShape', -0.2677593139178232)]
```

04

DATA CLEANING

VARIABLES CATEGORICAS

MEJOR RESULTADO

VARIABLES SIN ORDEN NUMÉRICO

```
['MSSubClass',
 'MSZoning',
 'Alley',
 'LandContour',
 'Utilities',
 'LotConfig',
 'Neighborhood',
 'Condition1',
 'Condition2',
 'BldgType',
 'HouseStyle',
 'RoofStyle',
 'RoofMatl',
 'Exterior1st',
 'Exterior2nd',
 'MasVnrType',
 'Foundation',
 'Heating',
 'Electrical',
 'GarageType',
 'MiscFeature',
 'MoSold',
 'SaleType',
 'SaleCondition']
```

from category_encoders import OneHotEncoder

Una vez realizado OneHotEncoder, se eliminan las columnas que tienen menos de 42 apariciones

```
# Se cuentan las columnas con pocas apariciones
cont = 0
list_poca_apa = []
y_col = df_train['SalePrice']
df_train = df_train.drop(['SalePrice'], axis=1)
for x in df_train.columns:
    if (df_train[x].sum()<42) & (df_test[x].sum()<42):
        cont = cont + 1
        list_poca_apa.append(x)
```

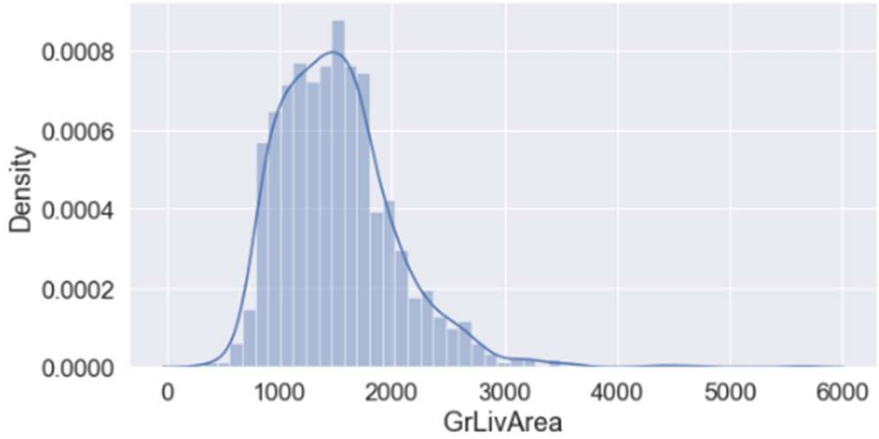
04

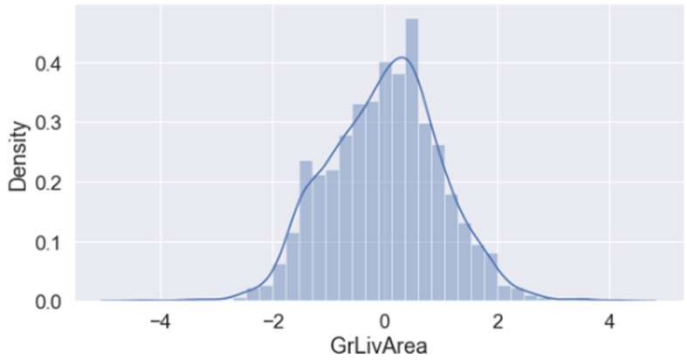
DATA CLEANING

SKEWNESS

PRUEBAS

VARIABLES NUMÉRICAS:
BOX COX





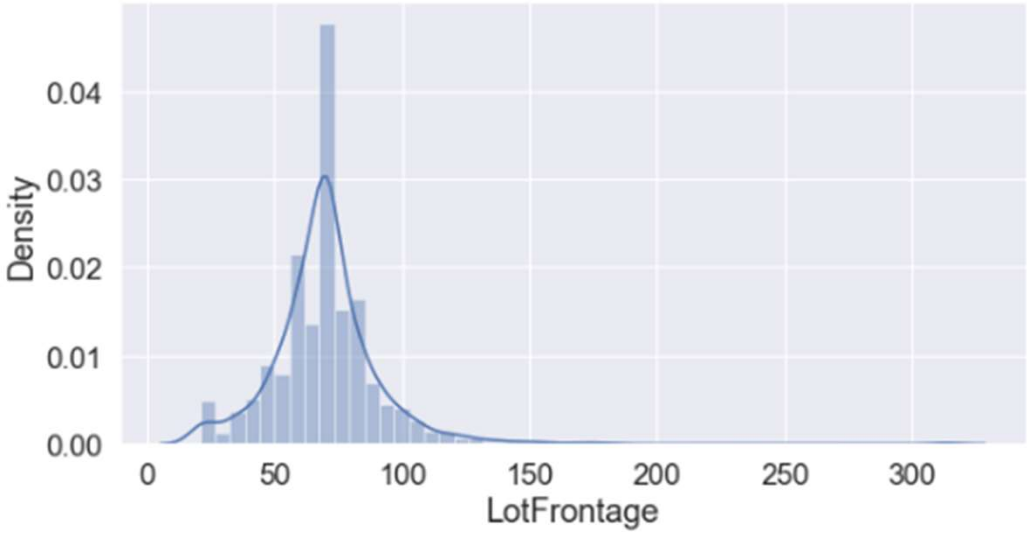
04

DATA CLEANING

SKEWNESS

MEJOR RESULTADO

VARIABLES NUMÉRICAS:
LOG TRANSFORMATION




```
df_train['LotFrontage'] = np.log1p(df_train['LotFrontage'])
df_test['LotFrontage'] = np.log1p(df_test['LotFrontage'])
```

04

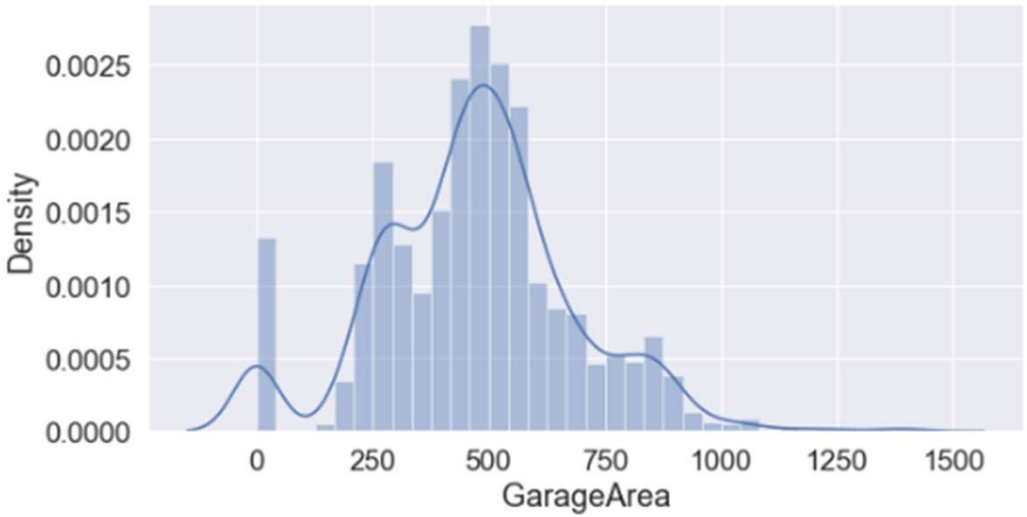
DATA CLEANING

SKEWNESS

MEJOR RESULTADO

VARIABLES NUMÉRICAS:
SEGMENTAR VARIABLES

```
def Gar_category(cat):
    if cat <= 250:
        return 1
    elif cat <= 500 and cat > 250:
        return 2
    elif cat <= 1000 and cat > 500:
        return 3
    return 4
```



04

DATA CLEANING

MEJOR RESULTADO

VARIABLES TEMPORALES:
CREACIÓN DE VARIABLES

```
df_train['YearBuilt_age'] = df_train['YearBuilt'].apply(lambda x: 0 if x==0 else (2022 - x))
df_test['YearBuilt_age'] = df_test['YearBuilt'].apply(lambda x: 0 if x==0 else (2022 - x))
df_train['YearRemodAdd_age'] = df_train['YearRemodAdd'].apply(lambda x: 0 if x==0 else (2022 - x))
df_test['YearRemodAdd_age'] = df_test['YearRemodAdd'].apply(lambda x: 0 if x==0 else (2022 - x))
df_train['YrSold_age'] = df_train['YrSold'].apply(lambda x: 0 if x==0 else (2022 - x))
df_test['YrSold_age'] = df_test['YrSold'].apply(lambda x: 0 if x==0 else (2022 - x))
df_train['GarageYrBlt_age'] = df_train['GarageYrBlt'].apply(lambda x: 0 if x==0 else (2022 - x))
df_test['GarageYrBlt_age'] = df_test['GarageYrBlt'].apply(lambda x: 0 if x==0 else (2022 - x))

df_train['renovated'] = df_train['YearRemodAdd'] + df_train['YearBuilt']
df_test['renovated'] = df_test['YearRemodAdd'] + df_test['YearBuilt']

df_train = df_train.drop(['YearBuilt', 'YearRemodAdd', 'YrSold', 'GarageYrBlt'], axis=1)
df_test = df_test.drop(['YearBuilt', 'YearRemodAdd', 'YrSold', 'GarageYrBlt'], axis=1)
```

05

FEATURE ENGINEERING

PRUEBAS

CREACIÓN DE VARIABLES

```
df['haspool'] = df['PoolArea'].apply(lambda x: 1 if x > 0 else 0)
df['has2ndfloor'] = df['2ndFlrSF'].apply(lambda x: 1 if x > 0 else 0)
df['hasgarage'] = df['GarageArea'].apply(lambda x: 1 if x > 0 else 0)
df['hasbsmt'] = df['TotalBsmtSF'].apply(lambda x: 1 if x > 0 else 0)
df['hasfireplace'] = df['Fireplaces'].apply(lambda x: 1 if x > 0 else 0)
```

```
df['TotalBsmtBath'] = df['BsmtFullBath']+(1/2)*df['BsmtHalfBath']
df['TotalBath'] = df['FullBath']+(1/2)*df['HalfBath']
df['TotalSf'] = df['1stFlrSF']+df['2ndFlrSF']+df['TotalBsmtSF']
df['TotalSqrFootage'] = df['BsmtFinSF1']+df['BsmtFinSF2']+df['1stFlrSF']+df['2ndFlrSF']
```

05

FEATURE ENGINEERING

MEJOR RESULTADO

CREACIÓN DE VARIABLES

```
df_train["SqFtPerRoom"] = df_train["GrLivArea"] / (df_train["TotRmsAbvGrd"] + df_train["FullBath"] +
                                                   df_train["HalfBath"] + df_train["KitchenAbvGr"])

df_train['TotalBath'] = df_train['FullBath'] + df_train['BsmtFullBath'] + 0.5 * df_train['BsmtHalfBath'] +
                        0.5 * df_train['HalfBath']

df_train['HighQualSF'] = df_train['1stFlrSF'] + df_train['2ndFlrSF']
```

05

FEATURE ENGINEERING

PRUEBAS

LASSO PARA VER LA IMPORTANCIA DE VARIABLES

```
lasso=Lasso(alpha=0.001)
lasso.fit(X_train[variables_numericas],y_train)
FI_lasso = pd.DataFrame({"Feature Importance":lasso.coef_}, index=X_train[variables_numericas].columns)
FI_lasso['feature imp abs']=abs(FI_lasso["Feature Importance"])
FI_lasso.sort_values("feature imp abs",ascending=False)
```

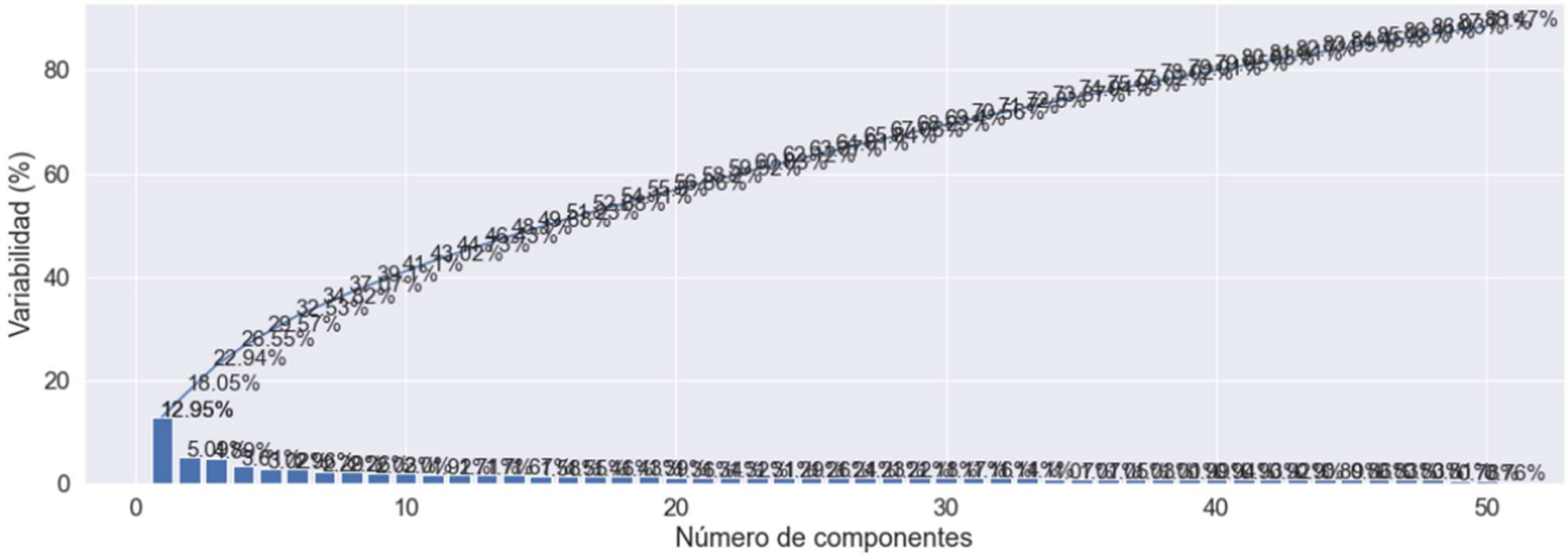
```
variables_to_drop_lasso=['MSSubClass', 'LotArea', 'MasVnrArea', 'BsmtHalfBath', 'BedroomAbvGr',
       'KitchenAbvGr', 'GarageYrBlt', 'OpenPorchSF', 'MiscVal', 'MoSold',
       'YrSold']
```

05

FEATURE ENGINEERING

PRUEBAS

PCA todas las variables numéricas

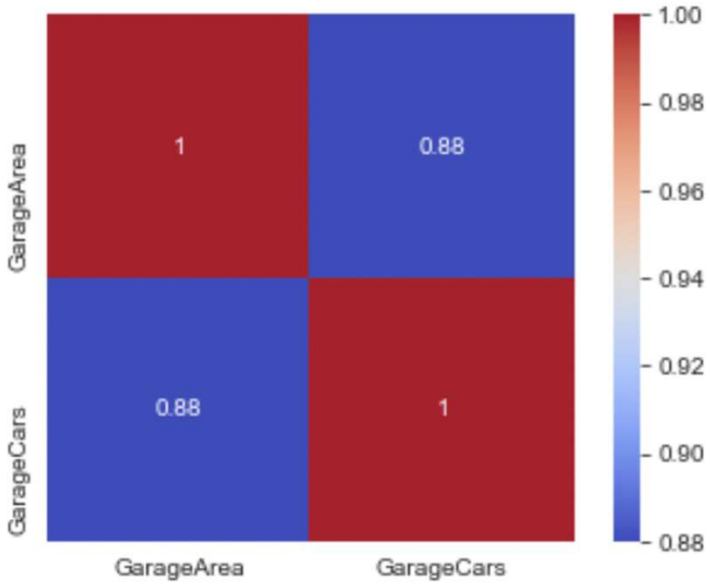


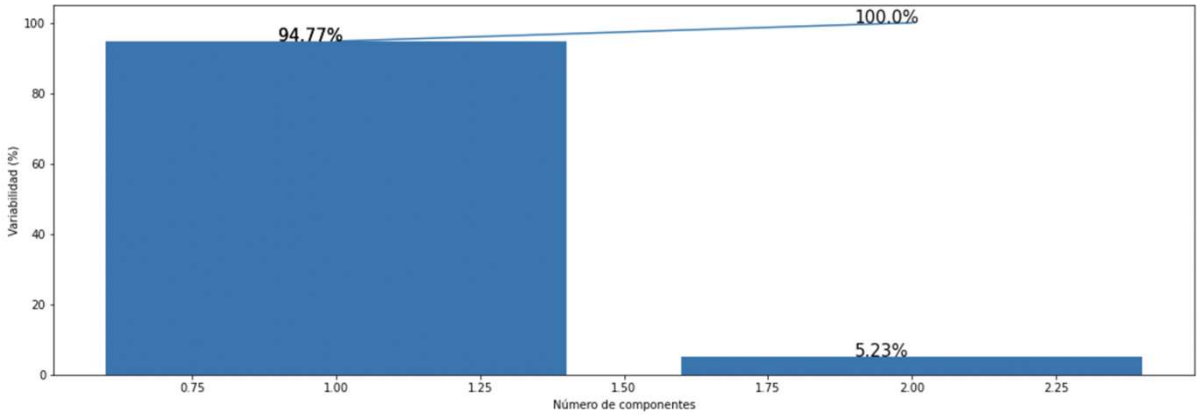
05

FEATURE ENGINEERING

PRUEBAS

PCA Garage Cars & Garage Area



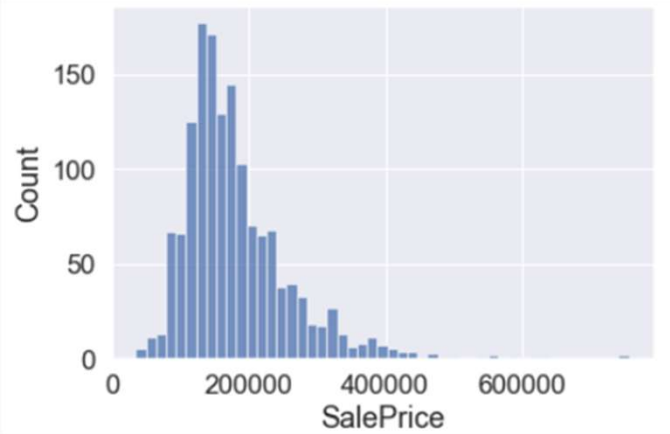


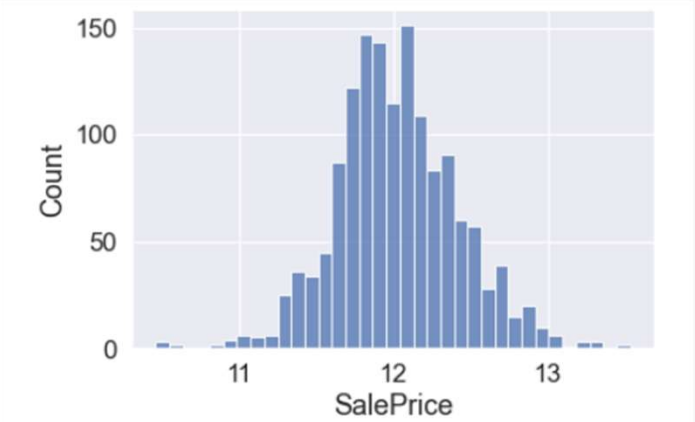
05

FEATURE ENGINEERING

MEJOR RESULTADO

LOG TRANSFORMATION A LA VARIABLE TARGET





06

PREDICTIVE MODELING

PRUEBAS

LASSO

RIDGE

SVR

LGBM REGRESSOR

ELASTIC NET

XGB REGRESSOR

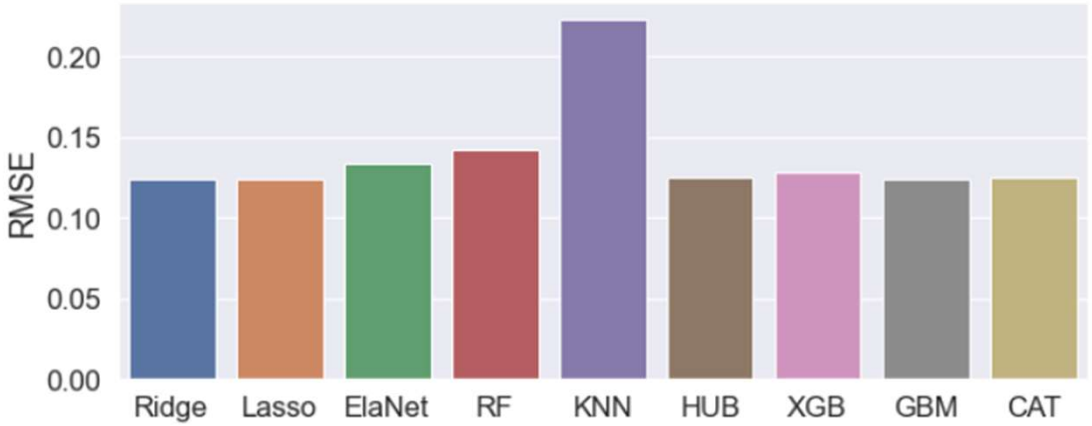
AVERAGING MODELS:

- LGMR
- Lasso
- Ridge
- XGBR

RANDOM FOREST

HISTGRADIENT BOOST REGRESSOR

GRADIENT BOOST REGRESSOR



MEJOR RESULTADO

```
estimators = [ ('ridge', Ridge()), ('lasso', Lasso())]
reg = StackingRegressor(estimators=estimators,
            final_estimator=Ridge())
grid_stack1 = reg.fit(X_train, y_train)
r2_score_stack1 = grid_stack1.score(X_test, y_test)
```

from sklearn.ensemble import StackingRegressor

```
estimators = [('cat', CatBoostRegressor()), ('xgb', XGBRegressor())]
reg = StackingRegressor(estimators=estimators,
            final_estimator=CatBoostRegressor())
grid_stack4 = reg.fit(X_train, y_train)
r2_score_stack4 = grid_stack4.score(X_test, y_test)
```

MEJOR RESULTADO

BLENDING MODELS:

```
y_xgb = grid_xgb.predict(X_test)
y_gbm = grid_gbm.predict(X_test)
y_cat = grid_cat.predict(X_test)
y_stack1 = grid_stack1.predict(X_test)
y_stack4 = grid_stack4.predict(X_test)
y_mean_models = (0.35 * y_stack1 + 0.15 * y_xgb + 0.15 * y_gbm + 0.1 * y_cat + 0.25 * y_stack4) / 1.0
```

CLASIFICACIÓN

PUESTO	SCORE	JUGADORES TOTALES	PORCENTAJE
105	0.11574	4186	2%