MACHINE LEARNING

House Pricing

Viridiana Espinosa

CONTEXTO

Target

Precio

Problema

Mejorar estratégias de venta y toma de decisiones al estimar los precios de las viviendas en una ciudad específica

Objetivo

Construir modelo de predicción que a partir de las características de una casa pueda estimar el precio de la vivienda diminuyendo un 30% el márgen de error que es la media (desviación actual)

Tipo de problema

Regresión

TIPO	VARIABLES	SELECCIÓN PARA ML
Binaria	* waterfront: Indica si la propiedad tiene vista al agua (0 para no, 1 para sí).	
Categórica Ordinal	* condition: Calificación general de la condición (1 a 5) * floors: Número de pisos * view: Nivel de calidad de la vista de la propiedad (0 a 4).	TODAS
Númerica Continua	* date: Fecha del listado de la propiedad * id: Identificador único de cada propiedad * lat: Coordenada de latitud de la ubicación de la propiedad * long: Coordenada de longitud de la ubicación de la propiedad * price: Precio de la propiedad en moneda * sqft_above: Área habitable sobre el nivel del suelo en pies cuadrados * sqft_basement: Área del sótano en pies cuadrados * sqft_living: Área habitable total en pies cuadrados * sqft_living15: Área promedio habitable de las 15 propiedades más cercanas en pies cuadrados * sqft_lot: Tamaño total del terreno en pies cuadrados * sqft_lot15: Tamaño promedio del terreno de las 15 propiedades más cercanas en pies cuadrados	* lat * sqft_basement * sqft_living * sqft_lot
Númerica Discreta	* bathrooms: Número de baños * bedrooms: Número de habitaciones * grade: Calificación general (1 a 13) * yr_built: Año en que se construyó la propiedad * yr_renovated: Año en que la propiedad fue renovada por última vez (0 si nunca) * zipcode: Código postal de la propiedad	* bathrooms * bedrooms * grade

División - Train & Test

1 División de datos

80% Train
20% Test

2 Tipo de Datos*1

- **Categóricos Oridnales**
- **Númericas Discretos**
- **Númericas Continuas**

3 Revisón de datos

- **Cardinalidad**
- **Valores Nulos (0%)**
- **Tipo de dato: int/float**

1*10 Cat
0.90 Continua

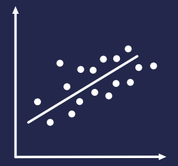
Mini EDA

1 Desviación estándar

Outliers: sqt_living, sqft_lot, grade, sqtf_above, sqft_basement, yr_built, yr_renovated, zipcode, sqft_living15 ysqft_lot15

2 Correlación de Pearson

- Criterio 0.09
- Excluidas - Baja relación con el target:
  - Últimas 6
  - Target, grade, waterfronr, bathrooms, bedrooms, yr_renovated y floors (categóricas)



3 Colinealidad

- Criterio: 0.7
- Excluidas - Variables que tienen alta relación con otras:
  - sqft_above', 'sqft_living15'

4 Logaritmo & Escalación

- Aplicación de logaritmo +1 debido a ceros
- Aplicaciónde escalado - Ajuste entre min y max manteniendo la distribución

5 Ordinal encoding

- Agrupación de valores por categorías más compactas
- Aplicación de un órden: A,B,C... / Bueno.Normal,Malo...
- Aplicación de Ordinal Encoder.fit_transform

Descarte de Waterfrony y yr_renovated debido a que la proporción de datos excedía el 80% en una sola categoría.

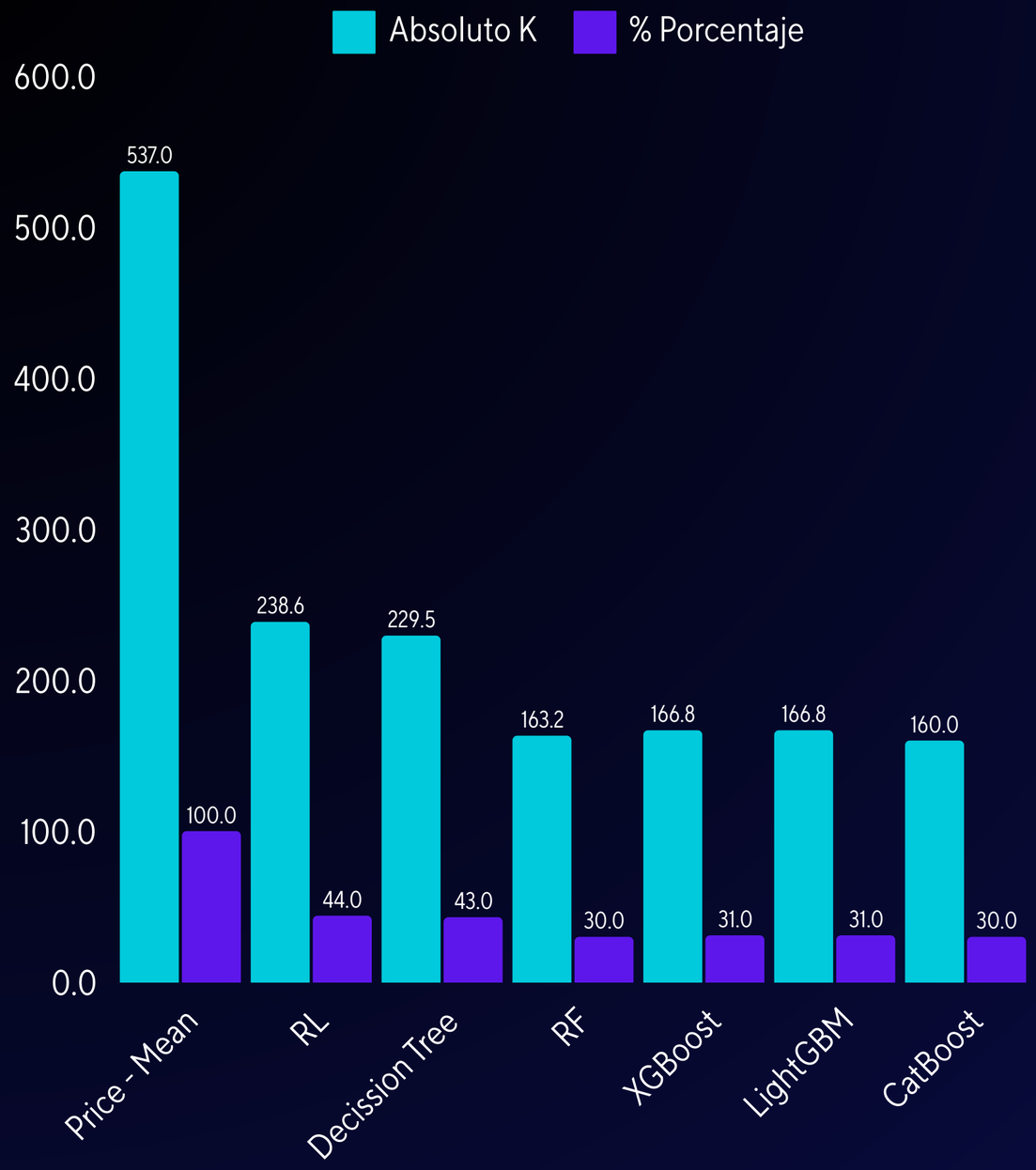

```
price            1.000000
sqft_living      0.701700
grade            0.665093
sqft_above       0.603255
sqft_living15    0.582666
bathrooms        0.526616
view             0.392108
sqft_basement    0.321387
lat              0.310770
bedrooms         0.308297
floors           0.253424
waterfront       0.252946
yr_renovated     0.127737
sqft_lot         0.091039
sqft_lot15       0.079152
zipcode          0.053800
yr_built         0.049099
condition        0.036343
long             0.023485
id               0.020624
Name: price, dtype: float64
```

VARIABLES:

NUMÉRICAS: ['sqft_living', 'view', 'sqft_basement', 'lat', 'sqft_lot']

CATEGÓRICAS: ['ordinal_floors', 'ordinal_bedrooms', 'ordinal_condition', 'ordinal_grade', 'ordinal_bathrooms']

Modelado




```python
model_names = ["Regresion Lineal","DecisionTree","Random Forest","XGBoost","LightGBM","CatBoost"]
lr_rg = LinearRegression()
tree_rg = DecisionTreeRegressor(random_state= 42)
rf_rg = RandomForestRegressor(random_state= 42)
xgb_rg = XGBRegressor(random_state = 42)
lgb_rg = LGBMRegressor(random_state= 42, verbose = -100)
cat_rg = CatBoostRegressor(random_state= 42, verbose = False)

model_set = [lr_rg, tree_rg, rf_rg, xgb_rg, lgb_rg, cat_rg]

```

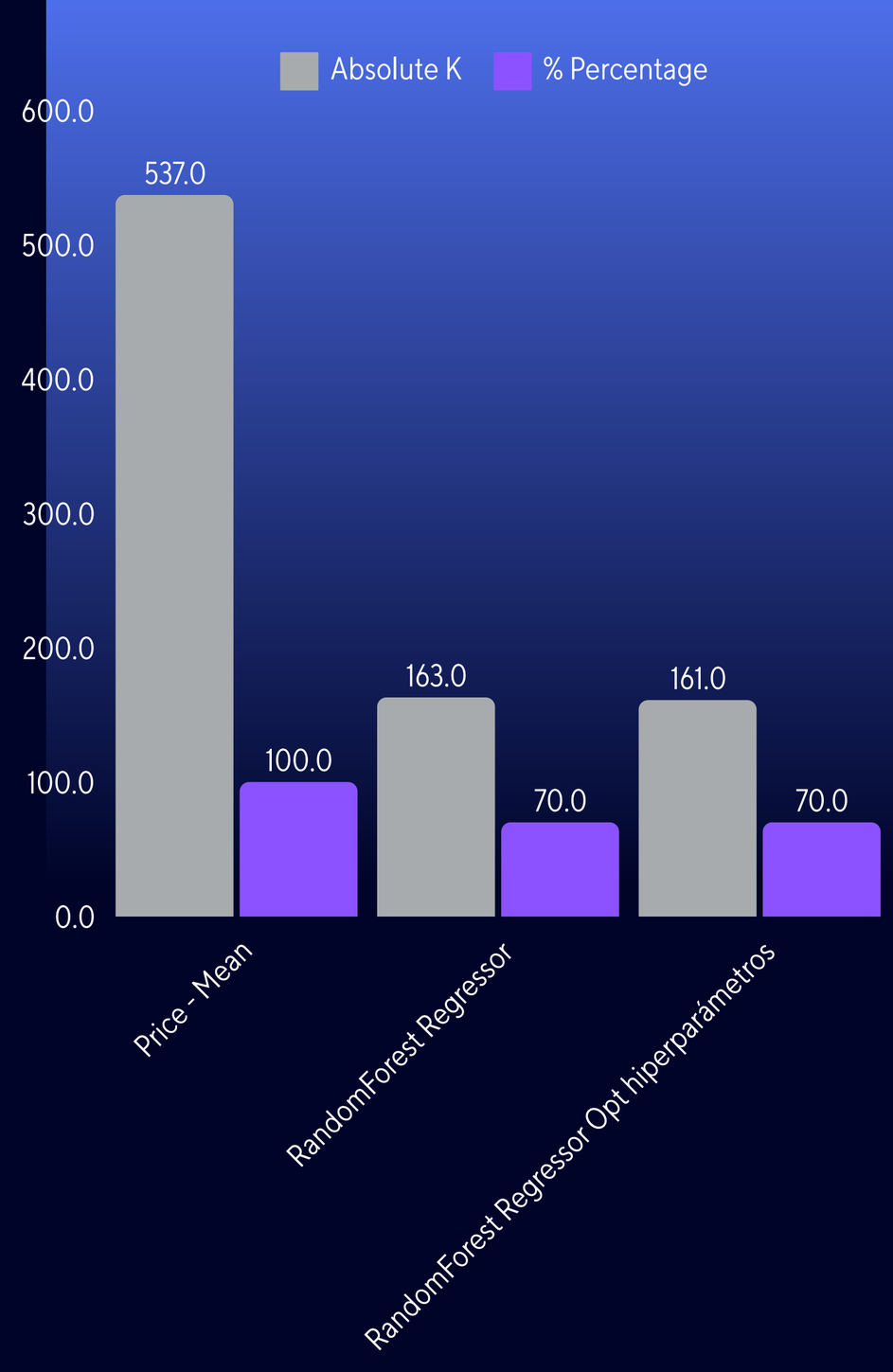
✓ 0.0s

1. Mejores parámteros:
   - RF
   - CatBoost
2. Selección de párametro ganador:
   - Random Forest Regressor
     - Obtención de resultado más eficiente
     - Mejora de desviación en estimación de precios vs mediana de un 30% ($163k/$537k)

Optimización de métricas

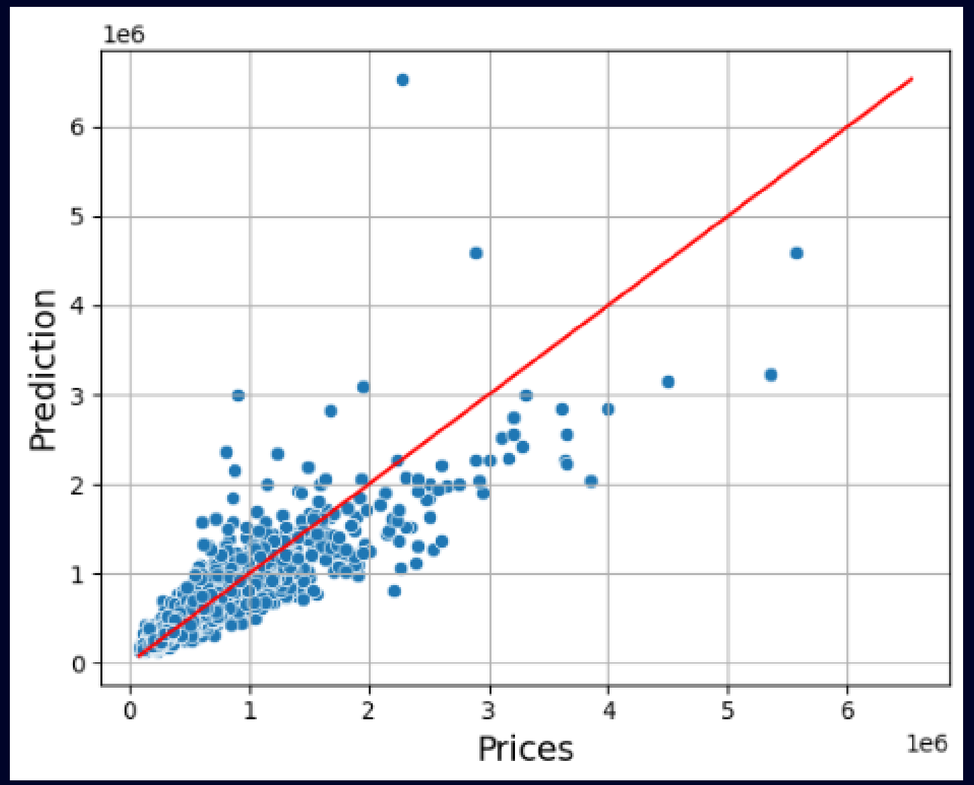
```python
params_grid = {
    "n_estimators": [250,500], #número de árboles, ir probando poquitos, hasta una metrica razonable.- por defecto es 100 - poner hasta 50
                            # El Random Forest no suele empeorar por exceso de
                            # estimadores. A partir de cierto numero no merece la pena
                            # perder el tiempo ya que no mejora mucho más la precisión.
                            # Entre 100 y 200 es una buena cifra
    "max_depth": [100,200], #máximo de hojas (altura y hasta donde llega) -  es None0 (dejar que sea None, probar, 10,4)
                        # No le afecta tanto el overfitting como al decissiontree.
                        # Podemos probar mayores profundidades

    "max_features": [5,7], #suelen ser estándar - SQRT
                                    # Numero de features que utiliza en cada split.
                                    # cuanto más bajo, mejor generalizará y menos overfitting.
    "max_samples": [0.7,None] #suelen ser estándar - None
}

rf_rg = RandomForestRegressor(random_state= 42)
rf_grid = GridSearchCV(rf_rg,
                       param_grid= params_grid,
                       cv = 5,
                       scoring = "neg_mean_squared_error",
                       n_jobs = -1)

rf_grid.fit(X_train_scaled, y_train)
```

Max_depth	100
Max_features	5
n_esitmaros	500
max_samples	Non



Análisis

Métrica	Resultado
Precio Medio Train	537K
MAE	48K
RMSE Train	161K
RMSE Test	184K



Consideraciones a tener en cuenta para un mejor pronóstico

- Algunas variables númericas discretas tuvieron que convertirse a categóricas ordinales como baños, cuartos, etc..
- Se excluyeron aquellas variables que tenían poca relación con el target o mucha relación entre sí (seleccionando solo una
- Se exluyeron aquellas variables que en proporción con su categoría se excedía más del 80/20 - ejemplo waterfront sin (17k) y con (124k)
- A mayor cantidad de condiciones, mayor es el outlier - ejemplo, entre más cuartos menor representación de población
- El mejor modelo fue el CatBoost, sin embargo, el tiempo de optimización es más costoso en tiempo y herramientas, por lo cual se ha seleccionado a RandomForest siendo el segundo mejor en métricas y el mejor en relación coste/eficiencia

Otras métricas optimizadas

```python
### Linear Regresion
from sklearn.linear_model import ElasticNet

param_grid = {
    "alpha": [0.1, 1, 10, 100],
    "l1_ratio": [0.2,0.4,0.6,1]

}
model = ElasticNet()
lr_grid = GridSearchCV( model,
                        cv = 5,
                        param_grid =  param_grid,
                        scoring= "neg_mean_squared_error"
)

lr_grid.fit(X_train_scaled,y_train)
print("LR best_score:", np.sqrt(-lr_grid.best_score_))
metricas_optimizadas["Linear Regresion"] = np.sqrt(-lr_grid.best_score_)
```

✓ 1.0s

```
LR best_score: 238678.22577900416
```

```python
### CatBoost

cat_rg = CatBoostRegressor(verbose = False)

param_grid= {'depth': [3, 6, 12],
            'learning_rate': [0.1, 0.2, 0.3, 0.4],
            #'bagging_fraction': [0.3,0.6,1], No hay hiperparámetro equivalente
            'colsample_bylevel': [0.5,1],
            'iterations': [100, 250, 500, 750],
            "border_count": [125,250]
            }


cat_grid = RandomizedSearchCV(cat_rg,
                              cv = 5,
                              n_iter = 3,
                              param_distributions= param_grid,
                              scoring = "neg_mean_squared_error")


cat_grid.fit(X_train, y_train)
print("CatBoost best_score:", np.sqrt(-cat_grid.best_score_))
metricas_optimizadas["CatBoost"] = np.sqrt(-cat_grid.best_score_)
```

✓ 54.0s

```
CatBoost best_score: 161557.17927476042
```

- Observamos que incluso optimizando los hipérparametros en CatBoost, la desviación/error sigue en un 30%, por lo que esto nos confirma quedarnos con Random Forest Regressor
- No se elige GridSearch como contraste puesto que:
  - El tiempo de obtención de resultados, sigue siendo más rápido en RF. Adicionalmente, la variación sigue estando en un 30%. Puesto que en caso de querer seguir haciendo mejoras a través de los hiperparámetros RF se mantiene como la mejor opción.

Gracias