



MACHINE LEARNING

House Pricing

Viridiana Espinosa

CONTEXTO

Crisis financiera: El mercado inmobiliario de EE. UU. en alerta



Escrito por [Sergio Sánchez](#) — 5 meses atrás En [Sector Financiero](#)



Target



Precio



Problema

Mejorar estrategias 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 disminuyendo un 30% el margen de error que es la media (desviación actual)



Tipo de problema

Regresión

División - Train & Test

| TIPO | VARIABLES | SELECCIÓN PARA ML |
|-----------------------|--|---|
| Binaria | * waterfront: Indicates if property has waterfront view (0 for no, 1 for yes). | |
| Categorica Ordinal | * condition: Overall condition rating (1 to 5) floors: Number of floors * view: Quality level of property view (0 to 4) | TODAS |
| Númerica Continua | * date: Date of property listing * id: Unique identifier for each property * lat: Latitude coordinate of property location * long: Longitude coordinate of property location * price: Property price in currency * sqft_above: Living area above ground level in square feet * sqft_basement: Basement area in square feet * sqft_living: Living area size in square feet * sqft_living15: Living area size of 15 nearest properties in square feet * sqft_lot: Total lot size in square feet * sqft_lot15: Lot size of 15 nearest properties in square feet | * lat * sqft_basement * sqft_living * sqft_lot |
| Númerica Discreta | * bathrooms: Number of bathrooms * bedrooms: Number of bedrooms * grade: Overall grade rating (1 to 13) * yr_built: Year property was built * yr_renovated: Year property was last renovated (0 if never) * zipcode: Property location zip code | * bathrooms * bedrooms * grade |

1 División de datos

80% Train
20% Test

2 Tipo de Datos^{*1}

- **Categoricos**
Ordinales
- **Númericas Discretos**
- **Númericas**
Continuas

3 Revisión de datos

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

¹*10 Cat
0.90 Continua

1

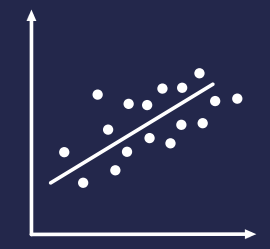
Desviación estándar

Outliers: sqft_living, sqft_lot, grade, sqft_above, sqft_basement, yr_built, yr_renovated, zipcode, sqft_living15, sqft_lot15

2

Correlación de Pearson

- Criterio 0.09
- Excluidas - Baja relación con el target:
 - Últimas 6
 - Target, grade, waterfront, 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
- Aplicación de escalado - Ajuste entre min y max manteniendo la distribución



5

Ordinal encoding

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

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

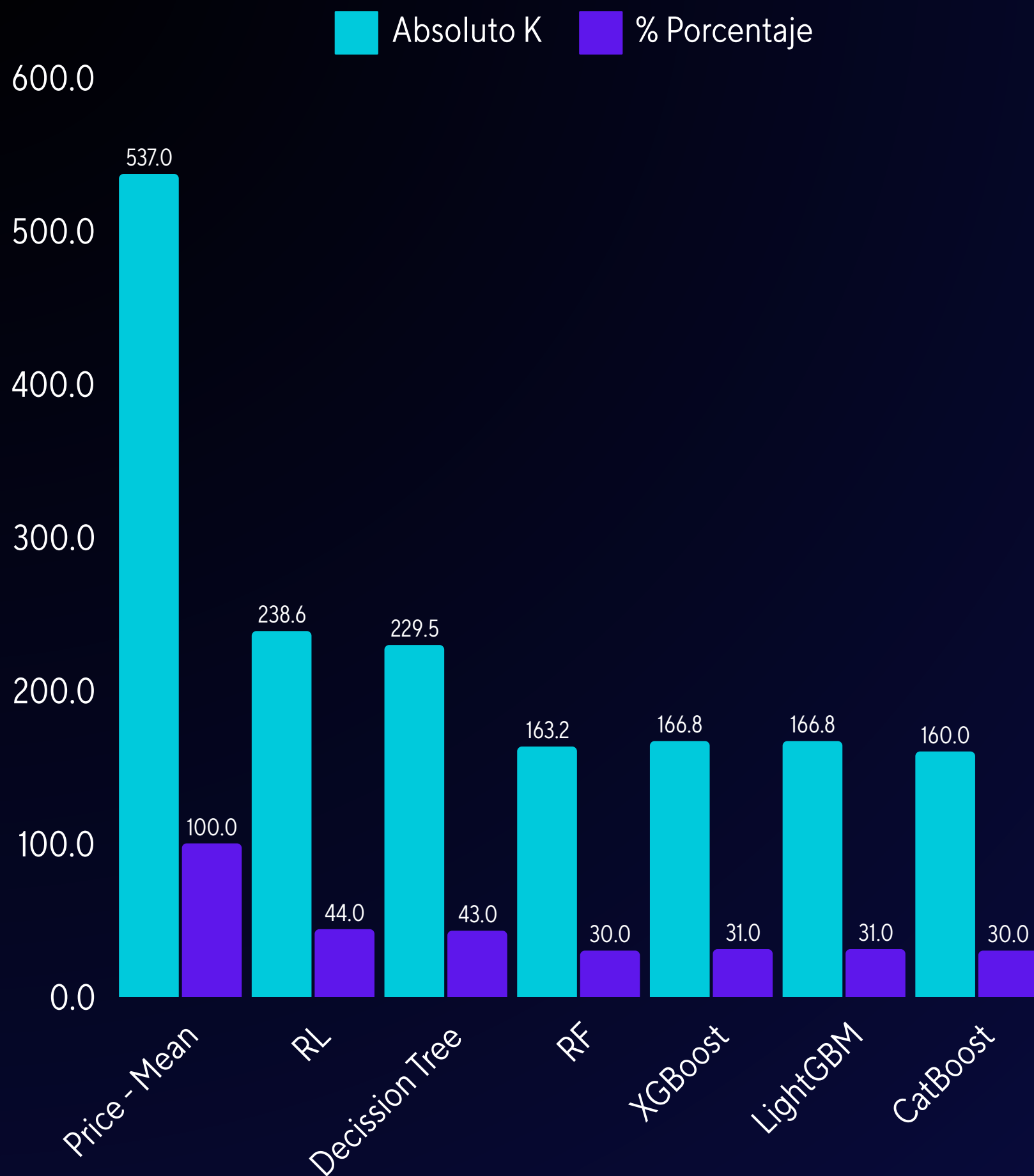


| | |
|-----------------------------|----------|
| price | 1.000000 |
| sqft_living | 0.701700 |
| grade | 0.665093 |
| sqft_above | 0.603255 |
| sqft_living15 | 0.582666 |
| bathrooms | 0.526616 |
| view | 0.392108 |
| sqft_basement | 0.321387 |
| lat | 0.310770 |
| bedrooms | 0.308297 |
| floors | 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

```
1 model_names = ["Regresion Lineal", "DecisionTree", "Random Forest", "XGBoost", "LightGBM", "CatBoost"]
2 lr_rg = LinearRegression()
3 tree_rg = DecisionTreeRegressor(random_state= 42)
4 rf_rg = RandomForestRegressor(random_state= 42)
5 xgb_rg = XGBRegressor(random_state = 42)
6 lgb_rg = LGBMRegressor(random_state= 42, verbose = -100)
7 cat_rg = CatBoostRegressor(random_state= 42, verbose = False)
8
9 model_set = [lr_rg, tree_rg, rf_rg, xgb_rg, lgb_rg, cat_rg]
10
✓ 0.0s
```

1 Mejores parámetros:

- RF
- CatBoost

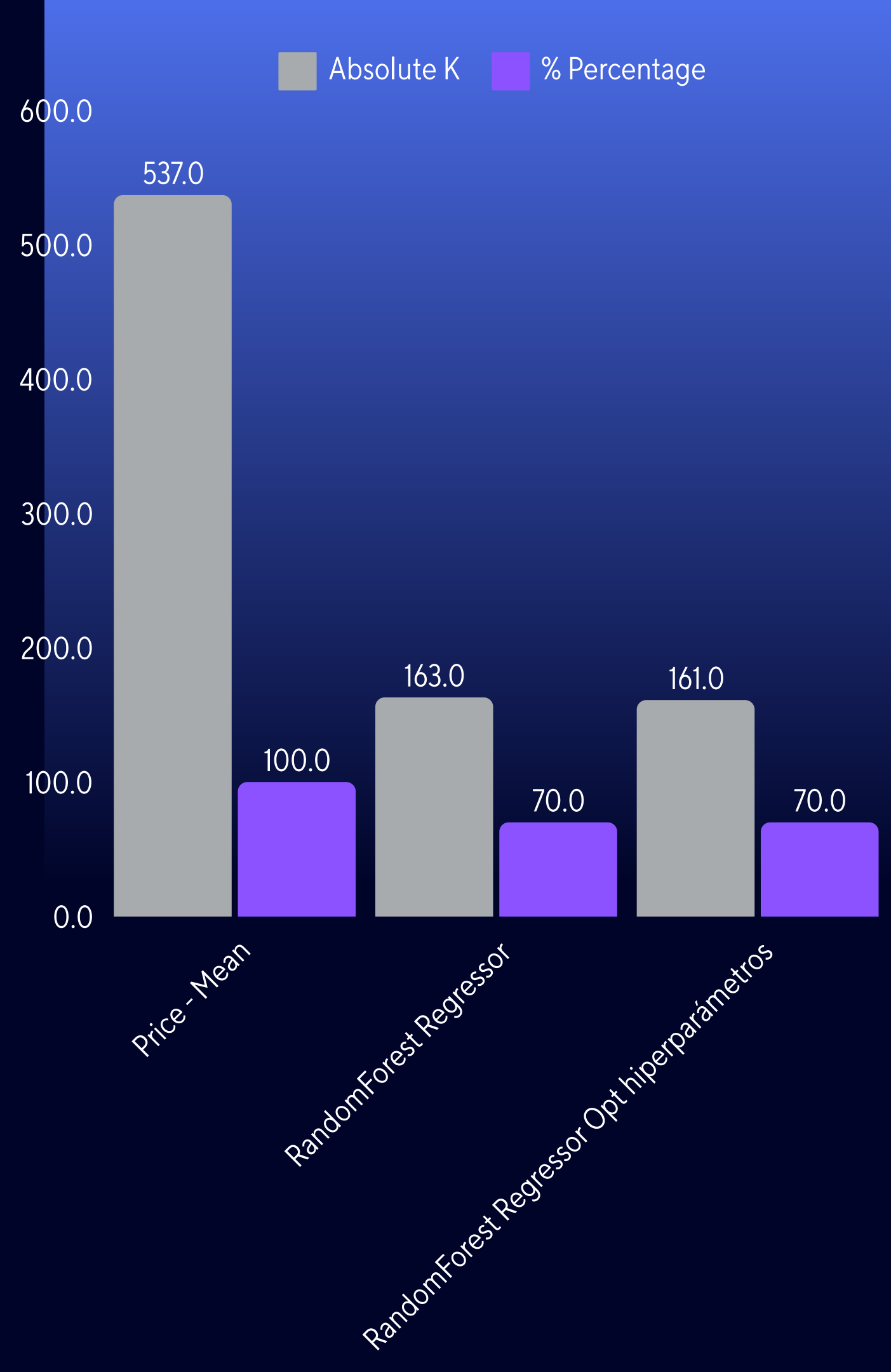
2 Selección de parámetro ganador:

- 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

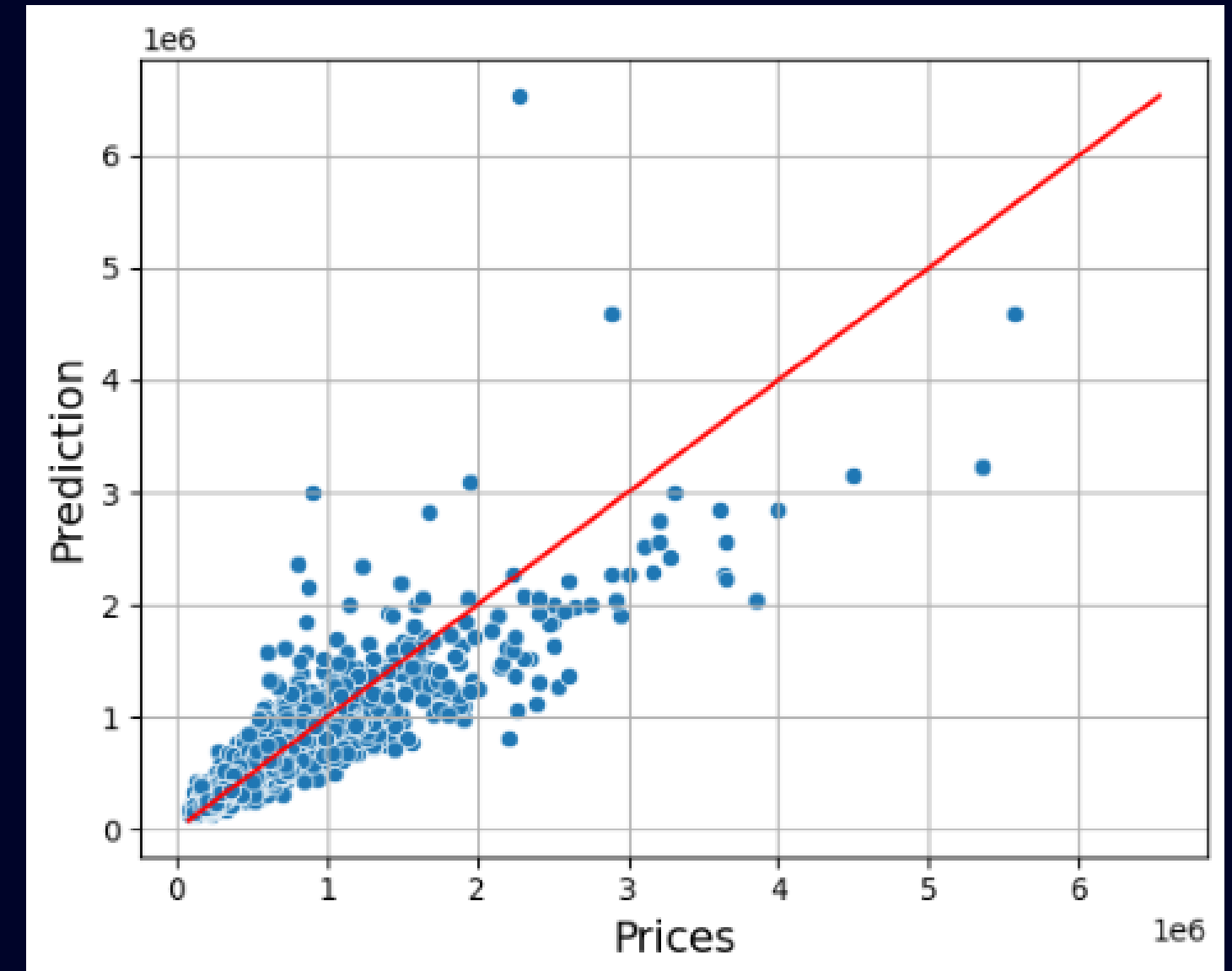
```
1 params_grid = {
2     "n_estimators": [250,500], #número de árboles, ir probando poquitos, hasta una metrica razonable.- por defecto es 100 - poner hasta 50
3     # El Random Forest no suele empeorar por exceso de
4     # estimadores. A partir de cierto numero no merece la pena
5     # perder el tiempo ya que no mejora mucho más la precisión.
6     # Entre 100 y 200 es una buena cifra
7     "max_depth": [100,200], #máximo de hojas (altura y hasta donde llega) - es None0 (dejar que sea None, probar, 10,4)
8     # No le afecta tanto el overfitting como al decissiontree.
9     # Podemos probar mayores profundidades
10
11     "max_features": [5,7], #suelen ser estándar - SQRT
12     # Numero de features que utiliza en cada split.
13     # cuanto más bajo, mejor generalizará y menos overfitting.
14     "max_samples": [0.7,None] #suelen ser estándar - None
15 }
16
17 rf_rg = RandomForestRegressor(random_state= 42)
18 rf_grid = GridSearchCV(rf_rg,
19     param_grid= params_grid,
20     cv = 5,
21     scoring = "neg_mean_squared_error",
22     n_jobs = -1)
23
24 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 numéricas 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 excluyeron 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

```
1  ### Linear Regression
2  from sklearn.linear_model import ElasticNet
3
4  param_grid = {
5      "alpha": [0.1, 1, 10, 100],
6      "l1_ratio": [0.2, 0.4, 0.6, 1]
7  }
8
9  model = ElasticNet()
10 lr_grid = GridSearchCV( model,
11                          cv = 5,
12                          param_grid = param_grid,
13                          scoring= "neg_mean_squared_error"
14 )
15
16 lr_grid.fit(X_train_scaled,y_train)
17 print("LR best_score:", np.sqrt(-lr_grid.best_score_))
18 metricas_optimizadas["Linear Regression"] = np.sqrt(-lr_grid.best_score_)
```

✓ 1.0s

LR best_score: 238678.22577900416

```
1  ### CatBoost
2
3  cat_rg = CatBoostRegressor(verbose = False)
4
5  param_grid= {'depth': [3, 6, 12],
6              'learning_rate': [0.1, 0.2, 0.3, 0.4],
7              #'bagging_fraction': [0.3,0.6,1], No hay hiperparámetro equivalente
8              'colsample_bylevel': [0.5,1],
9              'iterations': [100, 250, 500, 750],
10             "border_count": [125,250]
11             }
12
13
14  cat_grid = RandomizedSearchCV(cat_rg,
15                               cv = 5,
16                               n_iter = 3,
17                               param_distributions= param_grid,
18                               scoring = "neg_mean_squared_error")
19
20
21  cat_grid.fit(X_train, y_train)
22  print("CatBoost best_score:", np.sqrt(-cat_grid.best_score_))
23  metricas_optimizadas["CatBoost"] = np.sqrt(-cat_grid.best_score_)
```

61] ✓ 54.0s

• CatBoost best_score: 161557.17927476042

- Observamos que incluso optimizando los hiperparámetros 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

