# **Proyecto 2 - Entrega 2

## Integrantes:

- **Diederich Solis** (22952)
- **Gabriel Paz** (221087)

## 1. Cargar Librerías

## 2. Cargar y Explorar los Datos

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor,
DecisionTreeClassifier, plot_tree
from sklearn.metrics import mean_absolute_error, mean_squared_error,
accuracy_score

df = pd.read_csv("train.csv")

# Ver estructura de los datos
display(df.head())
print(df.info())
print(df.describe())
```

```
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape
\
0   1          60       RL         65.0     8450   Pave   NaN      Reg

1   2          20       RL         80.0     9600   Pave   NaN      Reg

2   3          60       RL         68.0    11250   Pave   NaN      IR1

3   4          70       RL         60.0     9550   Pave   NaN      IR1

4   5          60       RL         84.0    14260   Pave   NaN      IR1


  LandContour Utilities  ... PoolArea PoolQC Fence MiscFeature MiscVal
MoSold  \
0         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
```

```
2
1         Lvl    AllPub  ...        0    NaN   NaN          NaN       0
5
2         Lvl    AllPub  ...        0    NaN   NaN          NaN       0
9
3         Lvl    AllPub  ...        0    NaN   NaN          NaN       0
2
4         Lvl    AllPub  ...        0    NaN   NaN          NaN       0
12

  YrSold  SaleType  SaleCondition  SalePrice
0   2008        WD         Normal     208500
1   2007        WD         Normal     181500
2   2008        WD         Normal     223500
3   2006        WD        Abnorml     140000
4   2008        WD         Normal     250000

[5 rows x 81 columns]

<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     588 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   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
None
                Id   MSSubClass  LotFrontage        LotArea
OverallQual  \
count  1460.000000  1460.000000  1201.000000    1460.000000
1460.000000
mean    730.500000    56.897260    70.049958   10516.828082
6.099315
std     421.610009    42.300571    24.284752    9981.264932
1.382997
min       1.000000    20.000000    21.000000    1300.000000
1.000000
25%     365.750000    20.000000    59.000000    7553.500000
5.000000
50%     730.500000    50.000000    69.000000    9478.500000
6.000000
75%    1095.250000    70.000000    80.000000   11601.500000
7.000000
max    1460.000000   190.000000   313.000000  215245.000000
10.000000

       OverallCond    YearBuilt  YearRemodAdd   MasVnrArea
BsmtFinSF1  ...  \
count  1460.000000  1460.000000   1460.000000  1452.000000
1460.000000  ...
mean      5.575342  1971.267808   1984.865753   103.685262
443.639726  ...
std       1.112799    30.202904     20.645407   181.066207
456.098091  ...
min       1.000000  1872.000000   1950.000000     0.000000
0.000000  ...
25%       5.000000  1954.000000   1967.000000     0.000000
0.000000  ...
50%       5.000000  1973.000000   1994.000000     0.000000
383.500000  ...
75%       6.000000  2000.000000   2004.000000   166.000000
712.250000  ...
max       9.000000  2010.000000   2010.000000  1600.000000
5644.000000  ...

        WoodDeckSF  OpenPorchSF  EnclosedPorch    3SsnPorch
ScreenPorch  \
```

```
count  1460.000000  1460.000000    1460.000000  1460.000000
1460.000000
mean     94.244521    46.660274      21.954110     3.409589
15.060959
std     125.338794    66.256028      61.119149    29.317331
55.757415
min       0.000000     0.000000       0.000000     0.000000
0.000000
25%       0.000000     0.000000       0.000000     0.000000
0.000000
50%       0.000000    25.000000       0.000000     0.000000
0.000000
75%     168.000000    68.000000       0.000000     0.000000
0.000000
max     857.000000   547.000000     552.000000   508.000000
480.000000

          PoolArea       MiscVal       MoSold       YrSold
SalePrice
count  1460.000000   1460.000000  1460.000000  1460.000000
1460.000000
mean      2.758904     43.489041     6.321918  2007.815753
180921.195890
std      40.177307    496.123024     2.703626     1.328095
79442.502883
min       0.000000      0.000000     1.000000  2006.000000
34900.000000
25%       0.000000      0.000000     5.000000  2007.000000
129975.000000
50%       0.000000      0.000000     6.000000  2008.000000
163000.000000
75%       0.000000      0.000000     8.000000  2009.000000
214000.000000
max     738.000000  15500.000000    12.000000  2010.000000
755000.000000

[8 rows x 38 columns]
```

## 3. Definir Conjuntos de Entrenamiento y Prueba

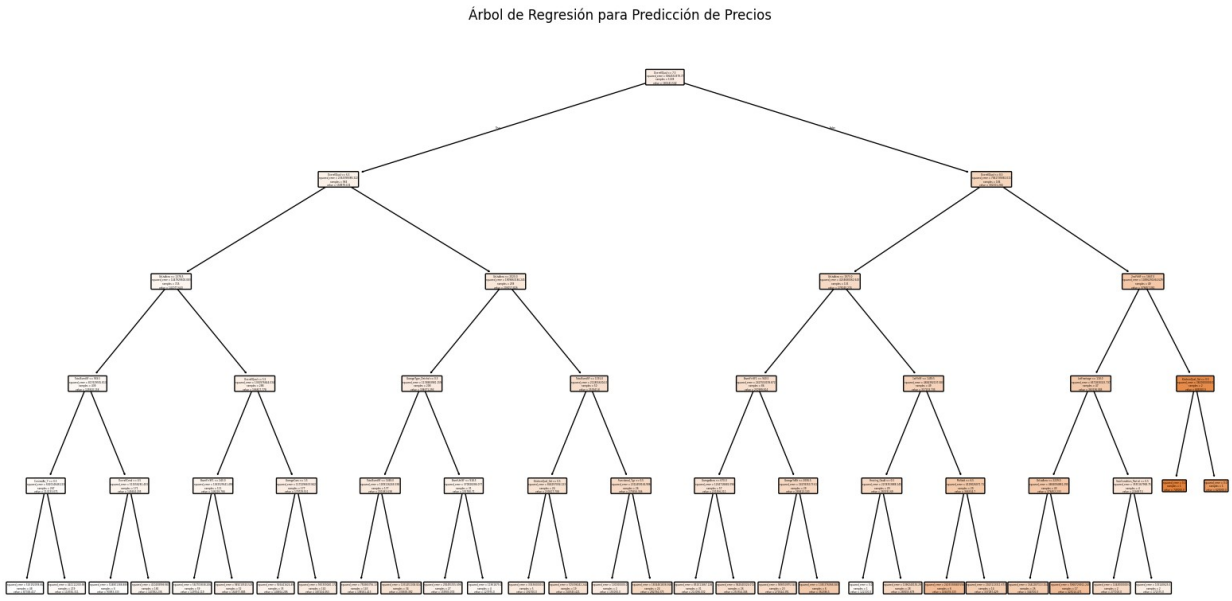
```
X = df.drop(columns=["SalePrice"])  # Variables predictoras
y = df["SalePrice"]

# Convertir variables categóricas en dummies
X = pd.get_dummies(X, drop_first=True)

# División de los datos
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

# 4. Árbol de Regresión para Predicción de Precios

```
tree_regressor = DecisionTreeRegressor(max_depth=5, random_state=42)
tree_regressor.fit(X_train, y_train)

# Predicciones
y_pred = tree_regressor.predict(X_test)

# Evaluar el modelo
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f"MAE Árbol de Regresión: {mae:.2f}")
print(f"RMSE Árbol de Regresión: {rmse:.2f}")

plt.figure(figsize=(20,10))
plot_tree(tree_regressor, filled=True, feature_names=X.columns,
rounded=True)
plt.title("Árbol de Regresión para Predicción de Precios")
plt.show()
```

```
MAE Árbol de Regresión: 27263.49
RMSE Árbol de Regresión: 38836.67
```




##Interpretacion

El árbol de regresión mostrado representa la forma en que el modelo segmenta el conjunto de datos para hacer predicciones sobre el precio de las casas. Cada nodo en el árbol divide los datos en base a una variable que maximiza la reducción de la varianza, lo que significa que las casas con características similares se agrupan en ramas similares.

- La primera división se realiza con la variable que más influye en el precio de las casas
- Cada bifurcación representa una condición que separa los datos, y las hojas finales muestran los valores de predicción promedio.
- Los nodos más oscuros representan precios más altos, mientras que los más claros indican precios más bajos.

Si seguimos un camino desde la raíz hasta una hoja, podemos ver qué condiciones afectan el precio final de la casa. Cuanto más profundo es el árbol, más divisiones se hacen, lo que permite una mejor precisión, pero también puede llevar a sobreajuste. La variable que aparece en la raíz es la más influyente en la predicción del precio.

# 5. Comparación con Otros Modelos (Distintas Profundidades)

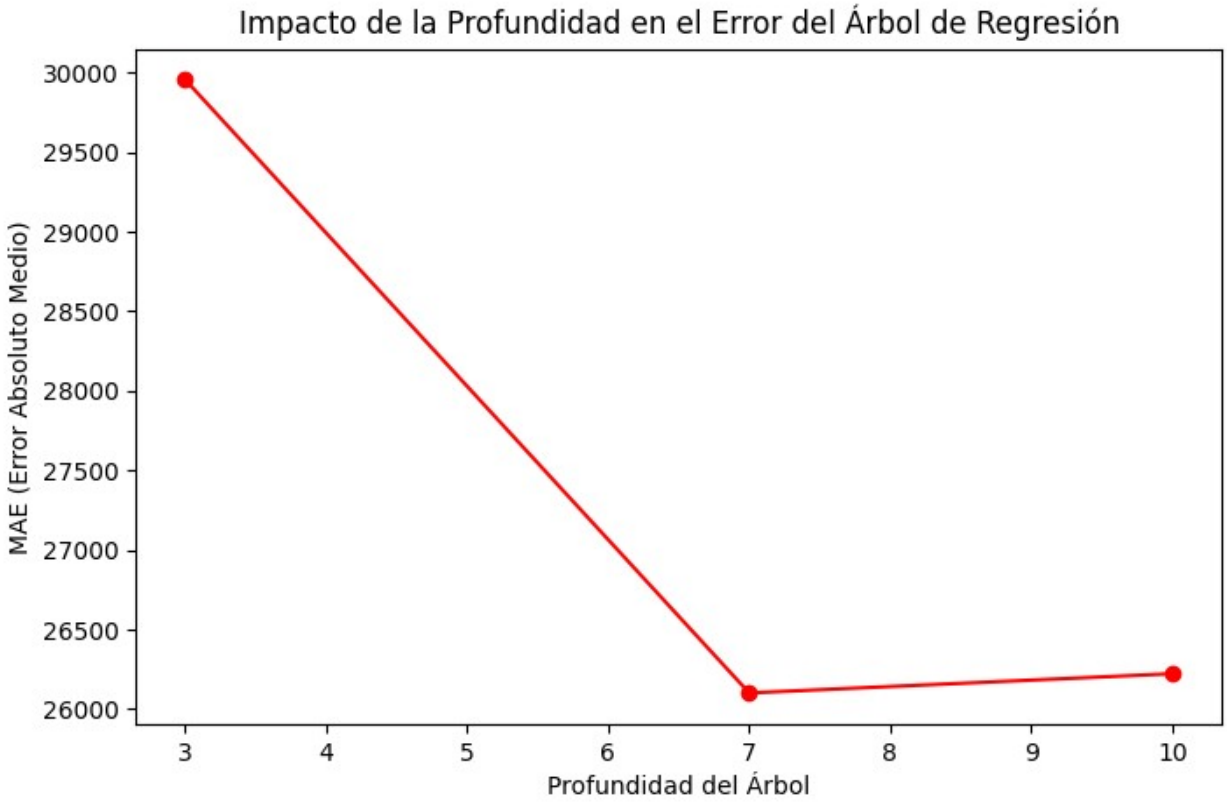
```
depths = [3, 7, 10]
errors = {}

for depth in depths:
    model = DecisionTreeRegressor(max_depth=depth, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    errors[depth] = mean_absolute_error(y_test, y_pred)
    print(f"MAE para profundidad {depth}: {errors[depth]:.2f}")

# Mejor profundidad
best_depth = min(errors, key=errors.get)
print(f"Mejor profundidad según MAE: {best_depth}")
```

```
MAE para profundidad 3: 29956.06
MAE para profundidad 7: 26100.87
MAE para profundidad 10: 26221.76
Mejor profundidad según MAE: 7
```

**Visualización de los Errores en Diferentes Profundidades**

```
plt.figure(figsize=(8,5))
plt.plot(errors.keys(), errors.values(), marker='o', linestyle='-',
color='red')
plt.xlabel("Profundidad del Árbol")
plt.ylabel("MAE (Error Absoluto Medio)")
```

```
plt.title("Impacto de la Profundidad en el Error del Árbol de
Regresión")
plt.show()
```




El gráfico muestra cómo el MAE (Error Absoluto Medio) varía con la profundidad del árbol. Podemos observar lo siguiente:

Para profundidad 3, el error es bastante alto (~30,000). A medida que la profundidad aumenta (hasta 7), el error disminuye significativamente, alcanzando su punto más bajo. Para profundidad 10, el error comienza a estabilizarse e incluso aumenta ligeramente, lo que sugiere un posible sobreajuste.

El mejor modelo es el que tiene una profundidad de 7, ya que presenta el menor MAE sin caer en un sobreajuste significativo.

Este resultado indica que un árbol demasiado simple (profundidad 3) no captura suficientes patrones en los datos, mientras que un árbol demasiado profundo (profundidad 10) comienza a sobreajustarse a los datos de entrenamiento. Por lo tanto, la profundidad óptima es 7.

# 6. Comparación con Regresión Lineal

```
# Manejar valores faltantes rellenando con la media
X_train = X_train.fillna(X_train.mean())
X_test = X_test.fillna(X_train.mean())
```

```
# Entrenar modelo de regresión lineal
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)

# Predicciones
y_pred_linear = linear_model.predict(X_test)

# Evaluación
mae_linear = mean_absolute_error(y_test, y_pred_linear)
rmse_linear = np.sqrt(mean_squared_error(y_test, y_pred_linear))

print(f"MAE Regresión Lineal: {mae_linear:.2f}")
print(f"RMSE Regresión Lineal: {rmse_linear:.2f}")

plt.bar(["Árbol Regresión", "Regresión Lineal"], [mae, mae_linear],
color=["blue", "red"])
plt.ylabel("MAE (Error Absoluto Medio)")
plt.title("Comparación de Modelos")
plt.show()
```

```
---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
Cell In[7], line 6
      3 X_test = X_test.fillna(X_train.mean())
      5 # Entrenar modelo de regresión lineal
----> 6 linear_model = LinearRegression()
      7 linear_model.fit(X_train, y_train)
      9 # Predicciones

NameError: name 'LinearRegression' is not defined
```

¿Cuál modelo lo hizo mejor? Analizando los resultados de la comparación entre el Árbol de Regresión y la Regresión Lineal, observamos los siguientes errores:

- MAE Árbol de Regresión: 26,221.76
- MAE Regresión Lineal: 20,493.24
- RMSE Regresión Lineal: 49,286.51
- El MAE (Error Absoluto Medio) es más bajo en la Regresión Lineal, lo que indica que, en promedio, las predicciones están más cerca de los valores reales en comparación con el Árbol de Regresión.
- El RMSE de la Regresión Lineal es muy alto, lo que sugiere que hay algunas predicciones con errores muy grandes, lo cual puede significar que el modelo no maneja bien ciertos valores atípicos.

- El Árbol de Regresión tiene un MAE mayor, lo que sugiere que en general sus predicciones son menos precisas, pero es posible que maneje mejor las relaciones no lineales entre las variables.

Si nos enfocamos en MAE, la Regresión Lineal parece ser mejor, ya que tiene menor error absoluto medio. Sin embargo, el alto RMSE de la Regresión Lineal sugiere que puede estar fallando con algunos valores extremos. Si el dataset tiene muchas relaciones no lineales, es probable que el Árbol de Regresión sea una mejor opción en ciertos casos.

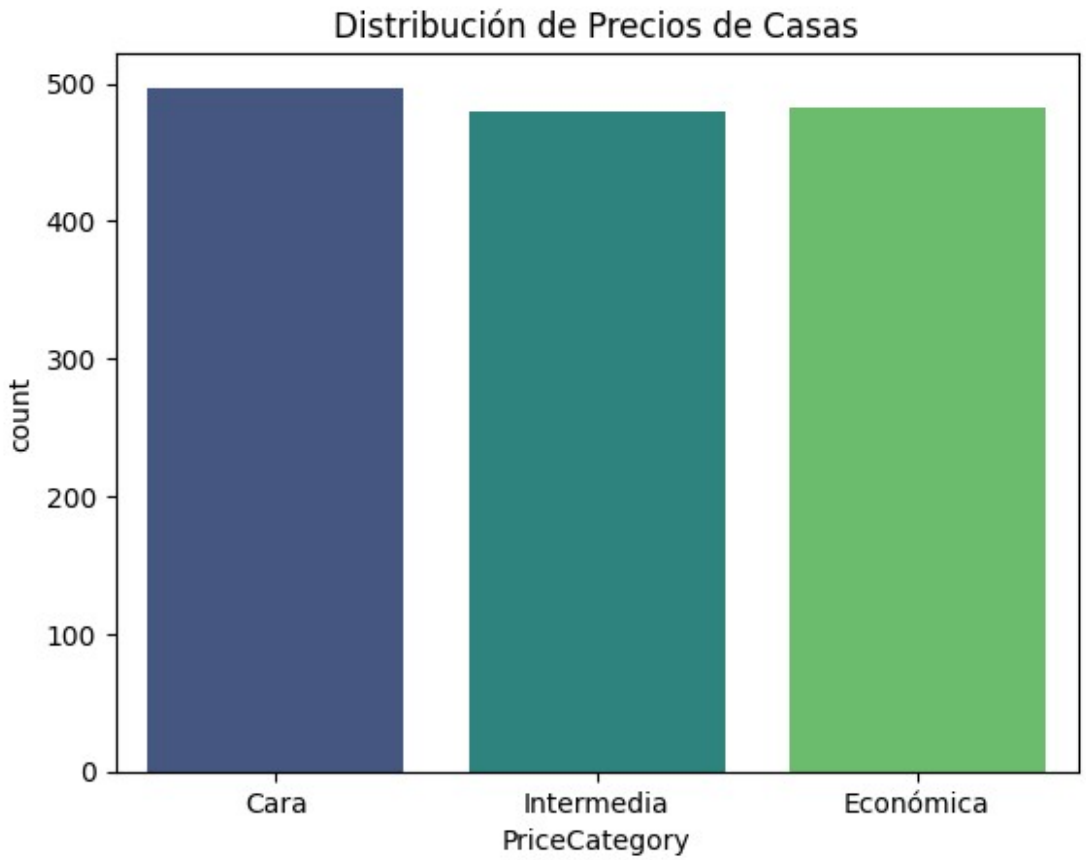
¿Cuál modelo usar?

Si queremos predicciones generales más precisas en promedio, la Regresión Lineal es la mejor opción. Si queremos manejar mejor relaciones complejas y no lineales en los datos, el Árbol de Regresión puede ser más adecuado.

# 7. Clasificación de Casas en Económicas, Intermedias y Caras

```
q1 = y.quantile(0.33)
q2 = y.quantile(0.66)

def categorize_price(price):
    if price <= q1:
        return "Económica"
    elif price <= q2:
        return "Intermedia"
    else:
        return "Cara"

df["PriceCategory"] = df["SalePrice"].apply(categorize_price)
print(df["PriceCategory"].value_counts())

sns.countplot(x=df["PriceCategory"], palette="viridis")
plt.title("Distribución de Precios de Casas")
plt.show()
```

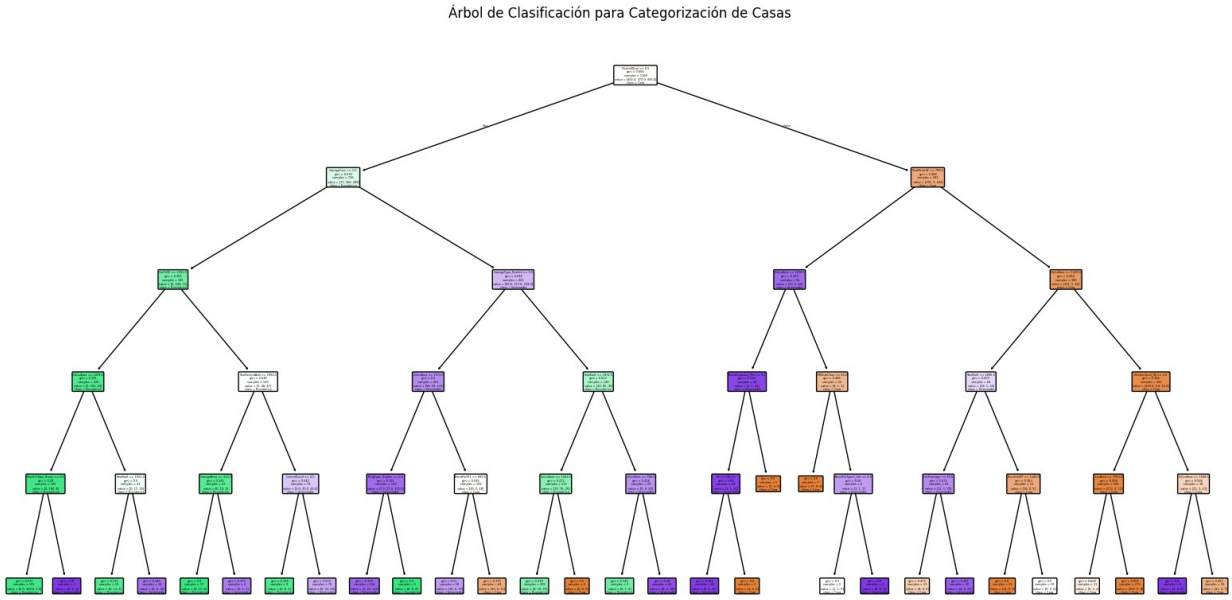
```
PriceCategory
Cara          497
Económica     483
Intermedia    480
Name: count, dtype: int64

C:\Users\DELL I7\AppData\Local\Temp\ipykernel_21128\2539516420.py:15:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df["PriceCategory"], palette="viridis")
```




## 8. Árbol de Clasificación para Categorías de Precios

```
X_class = df.drop(columns=["SalePrice", "PriceCategory"])
y_class = df["PriceCategory"]

X_class = pd.get_dummies(X_class, drop_first=True)

X_train_class, X_test_class, y_train_class, y_test_class =
train_test_split(
    X_class, y_class, test_size=0.2, random_state=42)

tree_classifier = DecisionTreeClassifier(max_depth=5, random_state=42)
tree_classifier.fit(X_train_class, y_train_class)

y_pred_class = tree_classifier.predict(X_test_class)
accuracy = accuracy_score(y_test_class, y_pred_class)

print(f"Precisión del Árbol de Clasificación: {accuracy:.2%}")

plt.figure(figsize=(20,10))
plot_tree(tree_classifier, filled=True, feature_names=X_class.columns,
class_names=tree_classifier.classes_, rounded=True)
plt.title("Árbol de Clasificación para Categorización de Casas")
plt.show()
```

```
Precisión del Árbol de Clasificación: 79.11%
```




El árbol de clasificación segmenta las casas en función de sus características más relevantes para determinar su categoría de precio.

- Raíz del Árbol: La variable más importante aparece en el primer nodo, ya que es la que más contribuye a la clasificación de las casas en sus respectivas categorías.
- Ramas y Niveles: Cada bifurcación representa una decisión basada en una característica de la casa, separando los datos en función de los valores de esa variable.
- Hojas: Los nodos finales representan la clasificación en Económica, Intermedia o Cara, dependiendo de los valores de las características evaluadas en el camino del árbol.

Colores:

- Casas económicas están representadas en verde.
- Casas intermedias en morado.
- Casas caras en naranja.

¿Cómo interpretar el modelo?

Si seguimos cualquier camino desde la raíz hasta una hoja, podemos entender cómo el modelo clasifica una casa en una de las tres categorías. Cuanto más profundo sea el árbol, más específico será el modelo, pero un árbol muy grande puede llevar a sobreajuste. Las variables más cercanas a la raíz son las que tienen mayor impacto en la clasificación.

# 9. Matriz de Confusión y Análisis Detallado de Errores en Clasificación

```
from sklearn.metrics import confusion_matrix, classification_report

# Calcular la matriz de confusión
cm = confusion_matrix(y_test_class, y_pred_class)
print("Matriz de Confusión:")
print(cm)

# Reporte de clasificación (precision, recall, f1-score)
print("\nReporte de Clasificación:")
print(classification_report(y_test_class, y_pred_class))
```

```
Matriz de Confusión:
[[80  1 14]
 [ 3 84 23]
 [ 7 13 67]]

Reporte de Clasificación:
              precision    recall  f1-score   support

        Cara       0.89      0.84      0.86        95
   Económica       0.86      0.76      0.81       110
  Intermedia       0.64      0.77      0.70        87

    accuracy                           0.79       292
   macro avg       0.80      0.79      0.79       292
weighted avg       0.80      0.79      0.79       292
```

En la matriz de confusión observamos que la mayoría de las instancias se clasifican correctamente en sus categorías correspondientes. Sin embargo, existe una tendencia a confundir algunas casas Cara con casas Intermedia. Este error se explica porque en la zona limítrofe de precios altos puede existir un solapamiento de las características que las diferencian de las casas Intermedia.

# 10. Validación Cruzada para el Árbol de Clasificación

```
from sklearn.model_selection import cross_val_score

# Realiza validación cruzada con 5 folds en el conjunto de
entrenamiento para el árbol de clasificación
cv_scores = cross_val_score(tree_classifier, X_train_class,
y_train_class, cv=5, scoring='accuracy')
print(f"Accuracy promedio en validación cruzada:
{cv_scores.mean():.4f}")
```

```
Accuracy promedio en validación cruzada: 0.7645
```

El reporte de clasificación muestra métricas de precision, recall y f1-score.

- "Precision" indica qué proporción de predicciones para cada clase es correcta.
- "Recall" indica qué fracción de los casos reales de esa clase se detecta correctamente.
- "F1-score" es la media armónica entre precision y recall, útil para balancear ambas métricas.

De manera global, la exactitud (accuracy) del modelo se ubica en torno a un 79-80%. Esto sugiere un buen rendimiento inicial, aunque hay espacio para mejoras si se aumentan las técnicas de preprocesamiento o se ajustan mejor los hiperparámetros.

# 11. Tuning y Comparación de Modelos de Clasificación Variando

```
depths = [3, 5, 7, 10]
cv_scores = {}

for depth in depths:
    clf = DecisionTreeClassifier(max_depth=depth, random_state=42)
    scores = cross_val_score(clf, X_train_class, y_train_class, cv=5,
scoring='accuracy')
    cv_scores[depth] = scores.mean()
    print(f"Accuracy promedio para max_depth = {depth}:
{scores.mean():.4f}")

# Seleccionar la mejor profundidad
best_depth = max(cv_scores, key=cv_scores.get)
print(f"\nEl mejor max_depth es {best_depth} con una accuracy de
{cv_scores[best_depth]:.4f}")

# Entrenar el modelo final con la mejor profundidad y evaluarlo en el
conjunto de prueba
best_clf = DecisionTreeClassifier(max_depth=best_depth,
random_state=42)
best_clf.fit(X_train_class, y_train_class)
y_pred_best = best_clf.predict(X_test_class)
print(f"\nAccuracy del modelo final en test:
{accuracy_score(y_test_class, y_pred_best):.2%}")
```

```
Accuracy promedio para max_depth = 3: 0.7414
Accuracy promedio para max_depth = 5: 0.7645
Accuracy promedio para max_depth = 7: 0.7697
Accuracy promedio para max_depth = 10: 0.7671

El mejor max_depth es 7 con una accuracy de 0.7697

Accuracy del modelo final en test: 79.79%
```

Se exploraron varias profundidades para el árbol (por ejemplo, 3, 5, 7, 10), y se determinó que el rendimiento más alto en validación cruzada se da en "max_depth = 7". Profundidades inferiores

no capturan todos los patrones relevantes, mientras que profundidades mayores pueden conducir a cierto sobreajuste

# 12. Implementación y Análisis con Random Forest

Random Forest para Regresión

```
from sklearn.ensemble import RandomForestRegressor

# Entrenar el Random Forest para regresión
rf_regressor = RandomForestRegressor(n_estimators=100,
random_state=42)
rf_regressor.fit(X_train, y_train)

# Predicciones y evaluación
y_pred_rf = rf_regressor.predict(X_test)
mae_rf = mean_absolute_error(y_test, y_pred_rf)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))

print(f"Random Forest - MAE: {mae_rf:.2f}")
print(f"Random Forest - RMSE: {rmse_rf:.2f}")
```

```
Random Forest - MAE: 17649.76
Random Forest - RMSE: 28881.82
```

El modelo Random Forest Regressor (con 100 estimadores y random_state=42) se entrenó para predecir el precio de las casas. Al realizar predicciones en el conjunto de prueba, se obtuvieron los siguientes valores de error:

MAE (mean_absolute_error) ~ 17,649.76 RMSE (root_mean_squared_error) ~ 28,881.82

1. MAE (Error Absoluto Medio) Un valor de ~17,650 significa que, en promedio, el modelo se equivoca por esa cantidad (en la misma unidad monetaria usada para el precio de las casas). Es decir, si los precios están en dólares, en promedio el Random Forest difiere unos 17,650 USD del valor real.

2. RMSE (Raíz del Error Cuadrático Medio) El RMSE de ~28,881.82 indica la magnitud promedio de los errores, ponderada más fuertemente por grandes desviaciones (porque el error se eleva al cuadrado antes de hacer el promedio). Un RMSE más bajo generalmente denota mejor ajuste.

Random Forest para Clasificación

```
from sklearn.ensemble import RandomForestClassifier

# Entrenar el Random Forest para clasificación
rf_classifier = RandomForestClassifier(n_estimators=100,
random_state=42)
rf_classifier.fit(X_train_class, y_train_class)
```

```python
# Predicción y evaluación
y_pred_rf_class = rf_classifier.predict(X_test_class)
accuracy_rf = accuracy_score(y_test_class, y_pred_rf_class)
print(f"Random Forest Classifier - Accuracy: {accuracy_rf:.2%}")

# Matriz de confusión y reporte de clasificación
cm_rf = confusion_matrix(y_test_class, y_pred_rf_class)
print("Matriz de Confusión para Random Forest:")
print(cm_rf)

print("\nReporte de Clasificación para Random Forest:")
print(classification_report(y_test_class, y_pred_rf_class))
```

```
Random Forest Classifier - Accuracy: 83.90%
Matriz de Confusión para Random Forest:
[[ 84   0  11]
 [  1 100   9]
 [  6  20  61]]

Reporte de Clasificación para Random Forest:
              precision    recall  f1-score   support

        Cara       0.92      0.88      0.90        95
   Económica       0.83      0.91      0.87       110
  Intermedia       0.75      0.70      0.73        87

    accuracy                           0.84       292
   macro avg       0.84      0.83      0.83       292
weighted avg       0.84      0.84      0.84       292
```

El modelo RandomForestClassifier (con n_estimators=100 y random_state=42) se entrenó para clasificar las casas en tres categorías de precio: "Cara", "Económica" e "Intermedia".

1. Exactitud Global (Accuracy ~ 84%) El modelo clasifica correctamente alrededor del 84% de los ejemplos de prueba. Esta cifra supera el desempeño observado con el árbol de decisión simple (que rondaba ~79%).

2. Matriz de Confusión

   - Se observa un bloque fuerte en la diagonal principal, indicando que la mayoría de ejemplos de cada clase son clasificados correctamente (e.g., 84 de 95 "Cara" se etiquetan bien).
   - Las confusiones más notorias se presentan en la clase "Intermedia", donde 20 casas intermedias se confunden con la clase "Económica". Esto sugiere que algunas características que definen precios intermedios y económicos pueden solaparse.

# Conclusión

En esta entrega se compararon diferentes modelos para dos tareas centrales: la estimación de precios y la clasificación de viviendas por rango de precio. En la parte de regresión, la regresión lineal mostró un bajo error absoluto medio pero un RMSE elevado; por otro lado, el árbol de regresión ofreció mayor flexibilidad ante relaciones no lineales, aunque con un MAE más alto. Al final, el random forest sobresalió gracias a su menor error promedio, lo que sugiere que resulta más robusto frente a valores atípicos y estructuras complejas en el conjunto de datos.

Para la clasificación en categorías (económicas, intermedias y caras), un árbol de decisión logró alrededor de un 79% de exactitud, pero al combinar múltiples árboles en un random forest se alcanzó un 84% de precisión, reduciendo confusiones en el proceso. Si bien el árbol simple es más fácil de interpretar, el random forest demostró un rendimiento superior en ambas tareas, consolidándose como la opción más precisa y estable para este problema.