

# Proyecto: Predicción del precio de viviendas usadas en Chile

Este proyecto utiliza una base de datos que contiene valor y características de casas usadas en Chile publicadas en el sitio web <https://chilepropiedades.cl/>.

He accedido a la base de datos de manera gratuita a través de **kaggle**. El dataset junto con el libro de códigos pueden revisarse en el siguiente link:

<https://www.kaggle.com/datasets/gorkigonzalez/casas-usadas-rm-chile-mayo-2020> y está disponible para todo publico.

El objetivo de este proyecto será utilizar esta base primeramente para realizar análisis descriptivo y limpieza de la misma. Posteriormente construiremos algunos modelos que nos permitan predecir los precios de las viviendas usadas. Finalmente se creará una web app que permita predecir el valor de las viviendas en un entorno amigable. Todo se realizará mediante código de python.

De antemano considerar que este proyecto cuenta con ciertas limitaciones sobre todo respecto a la disponibilidad de data. Considerar que la base de datos solo considera el período de mayo 2020 y por lo mismo el dataset puede ser ciertamente limitado al igual que las predicciones.

No obstante lo anterior, el objetivo de este proyecto es más bien poder dar una mirada a las posibilidades existentes respecto al machine learning cuando tenemos una data de este tipo. Teniendo una data actualizada o de mayor envergadura, resulta sencillo actualizar los modelos y procedimientos realizados aquí, incluida la app web para poder contar con predicciones y estimaciones más precisas.

```
In [53]: #Comenzaremos importando las librerías que usaremos
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import matplotlib
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,
mean_absolute_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.model_selection import GridSearchCV
import pickle as pkl
```

```
In [4]: #Ahora leemos el dataset
df = pd.read_excel('casasusadas.xlsx', sheet_name='Hoja1')
df.head()
```

```
Out[4]:
```

	Comuna	Link	Tipo_Vivienda	N_Habitaciones	N_Baños	N_Estacionami
0	Calera de Tango	<a href="https://chilepropiedades.cl/ver-publicacion/ve...">https://chilepropiedades.cl/ver-publicacion/ve...</a>	Casa	5.0	6.0	
1	Calera de Tango	<a href="https://chilepropiedades.cl/ver-publicacion/ve...">https://chilepropiedades.cl/ver-publicacion/ve...</a>	Casa	6.0	6.0	
2	Calera de Tango	<a href="https://chilepropiedades.cl/ver-publicacion/ve...">https://chilepropiedades.cl/ver-publicacion/ve...</a>	Casa	3.0	3.0	
3	Calera de Tango	<a href="https://chilepropiedades.cl/ver-publicacion/ve...">https://chilepropiedades.cl/ver-publicacion/ve...</a>	Casa	8.0	6.0	
4	Calera de Tango	<a href="https://chilepropiedades.cl/ver-publicacion/ve...">https://chilepropiedades.cl/ver-publicacion/ve...</a>	Casa	3.0	2.0	

```
In [9]: #Chequeamos datos perdidos
df.isnull().sum()
```

```
Out[9]: Comuna      0
Link      0
Tipo_Vivienda  0
N_Habitaciones  8
N_Baños    21
N_Estacionamientos  72
Total_Superficie_M2  37
Superficie_Construida_M2  36
Valor_UF    0
Valor_CLP    0
Dirección   37
Quién_Vende  0
Corredor     0
dtype: int64
```

```
In [7]: #Vemos la distribución por comuna
df['Comuna'].value_counts()
```

```
Out[7]: Quilicura          50
Las Condes          50
Pudahuel           49
Ñuñoa              49
Lo Barnechea       49
Vitacura           48
Providencia        48
Peñalolén          48
Maipú              48
Santiago           47
Colina             45
Puente Alto        45
La Reina           45
Lampa              42
San Bernardo       41
Macul              40
San Miguel         38
La Florida         29
Huechuraba         27
La Cisterna        25
Recoleta           25
Independencia      23
Conchalí           21
La Pintana         19
El Bosque          18
Peñaflor           17
Cerrillos          16
Padre Hurtado      15
Renca              15
San Joaquín        14
Calera de Tango    14
Estación Central   13
La Granja          12
Pedro Aguirre Cerda 12
Quinta Normal      11
Lo Prado           9
Lo Espejo          8
San Ramón          5
El Monte           3
San José de Maipo  3
Cerro Navia        3
Name: Comuna, dtype: int64
```

Lo que haremos primeramente será dejar en el dataset solo las variables que vamos a utilizar en nuestro modelo que son las que nos interesan. Quitaremos las variables poco relevantes.

```
In [10]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1139 entries, 0 to 1138
Data columns (total 13 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Comuna                                1139 non-null   object
1   Link                                  1139 non-null   object
2   Tipo_Vivienda                         1139 non-null   object
3   N_Habitaciones                        1131 non-null   float64
4   N_Baños                               1118 non-null   float64
5   N_Estacionamientos                   1067 non-null   object
6   Total_Superficie_M2                  1102 non-null   float64
7   Superficie_Construida_M2             1103 non-null   object
8   Valor_UF                             1139 non-null   float64
9   Valor_CLP                            1139 non-null   int64
10  Dirección                             1102 non-null   object
11  Quién_Vende                           1139 non-null   object
12  Corredor                              1139 non-null   object
dtypes: float64(4), int64(1), object(8)
memory usage: 115.8+ KB
```

```
In [11]: #Eliminamos columnas
df = df.drop(['Link', 'Tipo_Vivienda', 'Total_Superficie_M2',
'Valor_UF', 'Dirección', 'Quién_Vende', 'Corredor',
'N_Estacionamientos'], axis=1)
```

Eliminamos columnas poco relevantes como el link de la venta o el tipo de vivienda (ya que todas corresponden a casas). Eliminamos también la variable Valor\_UF, ya que como variable explicativa utilizaremos Valor\_CLP. Eliminamos también la variable N\_Estacionamientos dado que como observamos más arriba, tenemos bastantes valores perdidos en esa variable.

```
In [12]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1139 entries, 0 to 1138
Data columns (total 5 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Comuna                                1139 non-null   object
1   N_Habitaciones                        1131 non-null   float64
2   N_Baños                               1118 non-null   float64
3   Superficie_Construida_M2             1103 non-null   object
4   Valor_CLP                            1139 non-null   int64
dtypes: float64(2), int64(1), object(2)
memory usage: 44.6+ KB
```

Finalmente quedamos con un dataset que contiene nuestro vector objetivo (Valor\_CLP), y las variables independientes Comuna, N° de habitaciones, N° de baños y Superficie construida, que son los valores que el usuario tendrá que ingresar posteriormente en la web app para poder realizar la predicción.

```
In [13]:
```

```
#Volvemos a chequear valores perdidos  
df.isnull().sum()
```

```
Out[13]: Comuna          0  
         N_Habitaciones  8  
         N_Baños        21  
         Superficie_Construida_M2  36  
         Valor_CLP      0  
         dtype: int64
```

Como se observa, tenemos algunos valores perdidos en las variables "N\_Habitaciones", "N\_Baños", y "Superficie\_Construida\_M2". Lo que haremos será quitar estas filas.

No obstante lo anterior, se debe tener conciencia de que eliminar las filas con valores perdidos no siempre es la solución adecuada. Existen otras soluciones, como la imputación. Por ejemplo, en algunos casos podríamos reemplazar estos valores perdidos por un promedio de las otras observaciones.

```
In [14]: #Eliminamos filas con valores perdidos  
df = df.dropna(subset=['N_Habitaciones', 'N_Baños',  
                      'Superficie_Construida_M2'])
```

```
In [16]: #Chequeamos  
df.isnull().sum()
```

```
Out[16]: Comuna          0  
         N_Habitaciones  0  
         N_Baños        0  
         Superficie_Construida_M2  0  
         Valor_CLP      0  
         dtype: int64
```

Ahora veremos como está distribuida cada variable para ver si debemos realizar algún ajuste extra.

```
In [17]: df['N_Habitaciones'].value_counts()
```

```
Out[17]: 3.0    347
         4.0    341
         5.0    186
         6.0     75
         2.0     64
         7.0     28
         8.0     20
         9.0      6
        10.0      6
         1.0      5
        11.0      3
        16.0      2
        19.0      1
        14.0      1
        Name: N_Habitaciones, dtype: int64
```

```
In [18]: #Dejamos esta variable como int
df['N_Habitaciones'] = df['N_Habitaciones'].astype(int)
```

```
In [20]: df['N_Baños'].value_counts()
```

```
Out[20]: 2.0    350
         3.0    299
         1.0    202
         4.0    139
         5.0     55
         6.0     18
         7.0     14
         8.0      4
        10.0      2
        12.0      1
         9.0      1
        Name: N_Baños, dtype: int64
```

```
In [21]: #Dejamos esta variable como int
df['N_Baños'] = df['N_Baños'].astype(int)
```

```
In [23]: df['Superficie_Construida_M2'].value_counts()
```

```
Out[23]: 140.0    52
        120.0    36
        200.0    31
        100.0    31
        180.0    25
         ..
        426.0     1
        474.0     1
        410.0     1
        421.0     1
        211.0     1
        Name: Superficie_Construida_M2, Length: 286, dtype: int64
```

```
In [24]: #Dejamos esta variable como int
df['Superficie_Construida_M2'] =
df['Superficie_Construida_M2'].astype(int)
```

```
In [25]: df.info()
```

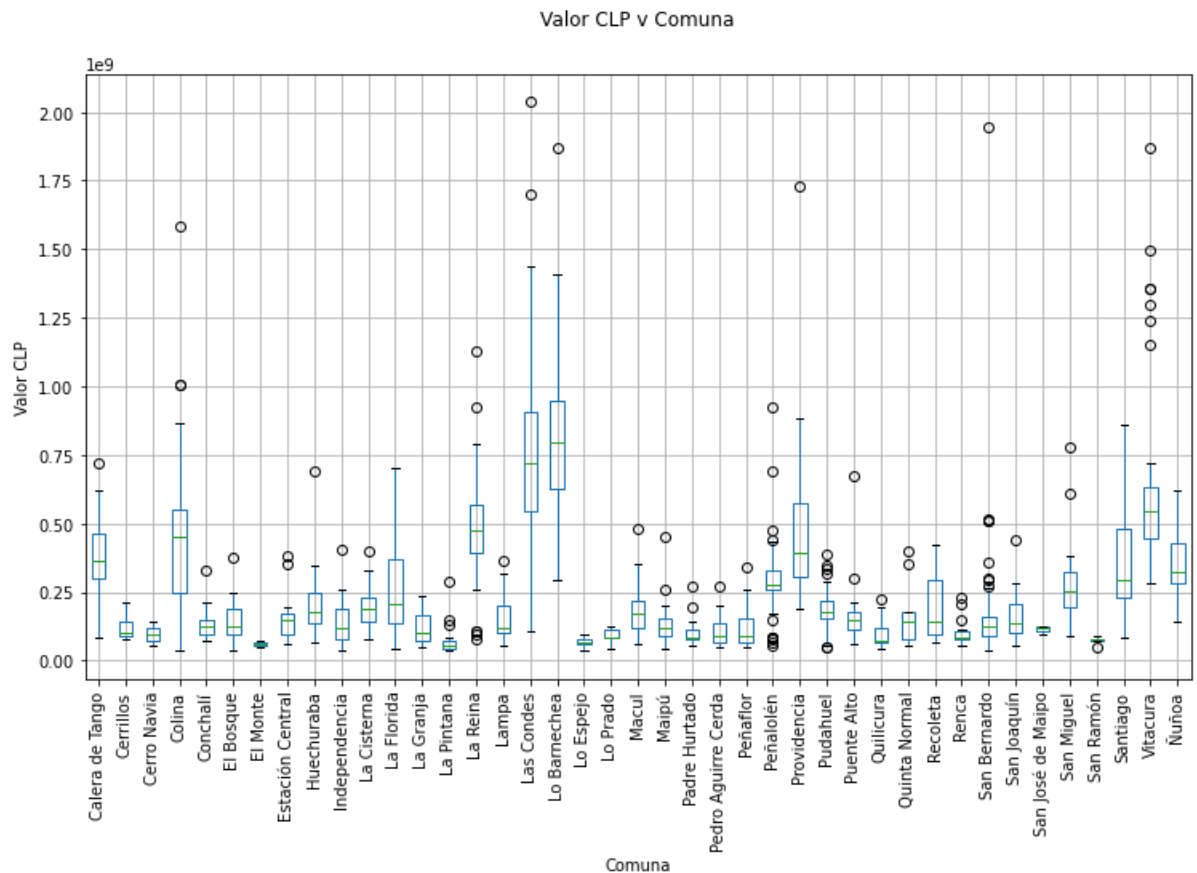
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1085 entries, 0 to 1138
Data columns (total 5 columns):
#   Column                      Non-Null Count  Dtype
---  -
0   Comuna                      1085 non-null   object
1   N_Habitaciones              1085 non-null   int32
2   N_Baños                     1085 non-null   int32
3   Superficie_Construida_M2    1085 non-null   int32
4   Valor_CLP                   1085 non-null   int64
dtypes: int32(3), int64(1), object(1)
memory usage: 38.1+ KB
```

```
In [30]: #Observemos los valores unicos de la variable comuna
df['Comuna'].unique()
```

```
Out[30]: array(['Calera de Tango', 'Cerrillos', 'Cerro Navia', 'Colina',
'Conchalí', 'El Bosque', 'El Monte', 'Estación Central',
'Huechuraba', 'Independencia', 'La Cisterna', 'La Florida',
'La Granja', 'La Pintana', 'La Reina', 'Lampa', 'Las Condes',
'Lo Barnechea', 'Lo Espejo', 'Lo Prado', 'Macul', 'Maipú', 'Ñuñoa',
'Padre Hurtado', 'Pedro Aguirre Cerda', 'Peñaflor', 'Peñalolén',
'Providencia', 'Pudahuel', 'Puente Alto', 'Quilicura',
'Quinta Normal', 'Recoleta', 'Renca', 'San Bernardo',
'San Joaquín', 'San José de Maipo', 'San Miguel', 'San Ramón',
'Santiago', 'Vitacura'], dtype=object)
```

Graficaremos un boxplot del precio de las viviendas por comuna, para tener un vistazo de como están distribuidas estas variables

```
In [29]: fig, ax = plt.subplots(1,1, figsize=(12, 7))
df.boxplot('Valor_CLP', 'Comuna', ax=ax)
plt.suptitle('Valor CLP v Comuna')
plt.title('')
plt.ylabel('Valor CLP')
plt.xticks(rotation=90)
plt.show()
```



Ahora vamos a utilizar LabelEncoder para poder transformar etiquetas de texto (como es la variable "Comuna"), a valores numéricos para poder trabajar con ellos.

```
In [33]: le_comuna = LabelEncoder()
df['Comuna'] = le_comuna.fit_transform(df['Comuna'])
df['Comuna'].unique()
```

```
Out[33]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 40, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32,
        33, 34, 35, 36, 37, 38, 39])
```

Con esto, ya tenemos listo el preprocess de la base.

```
In [35]: #Guardamos nuestra base pre-procesada
df.to_excel('df.xlsx', sheet_name='sheet1', index=False)
```

## Modelos

Lo primero que haremos será separar en nuestro dataset nuestro vector objetivo de las variables independientes.

```
In [36]: X = df.drop("Valor_CLP", axis=1)
y = df['Valor_CLP']
```



```
In [39]: #Haremos primero una regresión lineal
linear_reg = LinearRegression()
linear_reg.fit(X, y.values)
```

```
Out[39]: ▼ LinearRegression
LinearRegression()
```

```
In [40]: y_pred = linear_reg.predict(X)
```

```
In [43]: #Calculamos el error
error = np.sqrt(mean_squared_error(y, y_pred))
error
```

```
Out[43]: 231610457.27333322
```

Como se observa, el error es bastante grande, por lo que haremos pruebas con otros modelos. Intentaremos primeramente con un modelo de árbol de decisión.

```
In [45]: dec_tree_reg = DecisionTreeRegressor(random_state=0)
dec_tree_reg.fit(X, y.values)
```

```
Out[45]: ▼ DecisionTreeRegressor
DecisionTreeRegressor(random_state=0)
```

```
In [46]: y_pred = dec_tree_reg.predict(X)
```

```
In [48]: error = np.sqrt(mean_squared_error(y, y_pred))
print("${:,.02f}".format(error))
```

```
$8,197,189.57
```

Como se observa, el error se ha reducido considerablemente con la incorporación de este modelo. Sin embargo pondremos a prueba también un modelo Random Forest para evaluar cómo se comporta con nuestros datos.

```
In [50]: random_forest_reg = RandomForestRegressor(random_state=0)
random_forest_reg.fit(X, y.values)
```

```
Out[50]: ▼ RandomForestRegressor
RandomForestRegressor(random_state=0)
```

```
In [51]: y_pred = random_forest_reg.predict(X)
```

```
In [52]: error = np.sqrt(mean_squared_error(y, y_pred))
print("${:,.02f}".format(error))
```

\$58,745,953.85

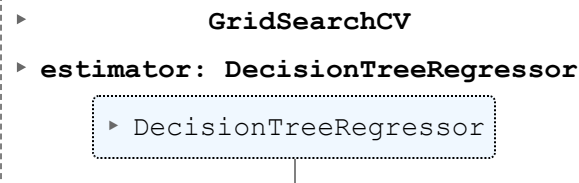
Como se observa, el error no es tan alto como el de nuestro modelo de regresión lineal, pero es más alto que el modelo de árbol de decisión.

Vamos a intentar mejorar los modelos a través de la búsqueda de grilla, así podemos modificar ciertos parámetros para evaluar si conseguimos mejores resultados.

```
In [54]: #Realizamos búsqueda de grilla

max_depth = [None, 2, 4, 6, 8, 10, 12]
#Definimos diccionario
parameters = {"max_depth": max_depth}

#Ocupamos nuestro modelo de árbol de decisión que fue el que tuvo
un mejor rendimiento
regressor = DecisionTreeRegressor(random_state=0)
gs = GridSearchCV(regressor, parameters,
scoring='neg_mean_squared_error')
gs.fit(X, y.values)
```

```
Out[54]: 
```

```
In [55]: regressor = gs.best_estimator_
regressor.fit(X, y.values)
y_pred = regressor.predict(X)
error = np.sqrt(mean_squared_error(y, y_pred))
print("${:,.02f}".format(error))
```

\$141,072,323.52

Como se observa, con la búsqueda de grilla el error sigue siendo alto. Nuestro modelo de árbol de decisión sin búsqueda de grilla sigue siendo el mejor modelo estimado con el error más bajo.

Antes de guardar el modelo de decision tree para utilizarlo posteriormente para la construcción de la web app, haremos una pequeña prueba para asegurarnos por una parte que las etiquetas que guardamos para las comunas se asignan correctamente, y que la carga del modelo también se realiza de manera correcta.

In [56]:

```
X
```

Out[56]:

	Comuna	N_Habitaciones	N_Baños	Superficie_Construida_M2
<b>0</b>	0	5	6	440
<b>1</b>	0	6	6	430
<b>2</b>	0	3	3	140
<b>3</b>	0	8	6	480
<b>4</b>	0	3	2	196
...	...	...	...	...
<b>1134</b>	39	4	3	211
<b>1135</b>	39	3	2	120
<b>1136</b>	39	6	3	242
<b>1137</b>	39	5	3	230
<b>1138</b>	39	4	2	215

1085 rows × 4 columns

In [61]:

```
X = np.array([["Macul", 3, 2, 100]])
X
```

Out[61]:

```
array([[ 'Macul', '3', '2', '100']], dtype='<U11')
```

In [62]:

```
X[:, 0] = le_comuna.transform(X[:, 0])
X = X.astype(float)
X
```

Out[62]:

```
array([[ 20.,   3.,   2., 100.]])
```

El valor numérico para la "Comuna" se está asignando correctamente, ya que "Macul" corresponde al valor "20" asignado por LabelEncoder.

In [63]:

```
#Hacemos la predicción para después compararla con la data cargada
y_pred = dec_tree_reg.predict(X)
y_pred
```

```
C:\Users\Mackarena\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X
does not have valid feature names, but DecisionTreeRegressor was fitted with featu
re names
warnings.warn(
```

```
Out[63]: array([98000000.])
```

```
In [64]: data = {"model": dec_tree_reg, "le_comuna": le_comuna}
with open ('saved_steps.pkl', 'wb') as file:
    pickle.dump(data, file)
```

```
In [65]: with open ('saved_steps.pkl', 'rb') as file:
    data = pickle.load(file)

tree_loaded = data["model"]
le_comuna = data["le_comuna"]
```

```
In [66]: y_pred = tree_loaded.predict(X)
y_pred
```

```
C:\Users\Mackarena\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X
does not have valid feature names, but DecisionTreeRegressor was fitted with featu
re names
warnings.warn(
```

```
Out[66]: array([98000000.])
```

Los resultados dan exactamente igual a los resultados anteriores. Con esto nos aseguramos que la carga del modelo se está haciendo correctamente, y podemos seguir con la construcción de la web app.