Universidad Andrés Bello®

Proyecto Ciencia de Datos: Desarrollo de un modelo predictivo para estimar el precio de las casas en la Región Metropolitana basado en sus características.

Angelo Cancino, Macarena Castro, Diego Pavez,
Alonso Riveros, Felipe Ruiz

Profesor John Ríos Griego

1. Descripción del Proyecto

Descripción del Proyecto

El proyecto consiste en analizar los precios de las casas en la Región Metropolitana (RM) de Chile utilizando un conjunto de datos seleccionado. El objetivo principal es desarrollar un modelo predictivo que permita estimar el precio de una casa basado en sus características específicas. Para ello, se aplicarán técnicas de la limpieza de datos, análisis exploratorio (análisis EDA), selección de características y modelado predictivo utilizando técnicas de redes neuronales, tanto en Python como en RStudio.

Justificación de la Relevancia

Este proyecto es altamente relevante en el campo de la Ciencia de Datos debido a la importancia del mercado inmobiliario tanto para los compradores y vendedores como para los inversores y economistas. Poder predecir los precios de las casas con precisión ayuda a tomar decisiones informadas, optimiza inversiones y aporta valor a la planificación urbana. Esto puede tener más potencial en el área de los vendedores, ya que ellos podrán saber un aproximado confiable de cuál sería el precio estimado de su vivienda en base a los resultados obtenidos. A su vez, ayuda al vendedor a ver qué datos aumentan o disminuyen el precio de su vivienda, como puede ser los espacios de estacionamiento que tiene, la cantidad de habitaciones o la comuna en la que reside.

2. Argumentación de Factibilidad

El proyecto es factible de realizarse dentro del marco de la asignatura por varias razones:

- **Recursos disponibles**: Contamos con un conjunto de datos relevante y completo sobre los precios de casas en la RM, lo que proporciona una base sólida para el análisis y modelado.
- **Tiempo y alcance**: El análisis y modelado de precios de viviendas es un proyecto manejable dentro de un período académico, con etapas bien definidas que se pueden abordar de manera progresiva.
- **Competencias de los estudiantes**: Los estudiantes han adquirido conocimientos en técnicas de análisis de datos, limpieza de datos, y modelado predictivo durante la asignatura, lo que les permite abordar este proyecto con las competencias necesarias.

3. Objetivos

- **Objetivo General**: Desarrollar un modelo predictivo para estimar el precio de las casas en la Región Metropolitana basado en sus características.
- **Objetivos Específicos**:
    1. Realizar una limpieza y preparación de los datos proporcionados.
    2. Llevar a cabo un análisis exploratorio de datos (EDA) para identificar patrones y relaciones entre variables.
    3. Seleccionar las características más relevantes para el modelo predictivo.
    4. Implementar un modelo de red neuronal en R Studio para predecir los precios de las casas según las variables.
    5. Implementar un modelo de regresión lineal ordinaria en Python para predecir los precios de las casas según las variables.
    6. Validar y comparar ambos modelos utilizando la métrica Error Cuadrático Medio (mse), Raíz del Error Cuadrático Medio (rmse) y el Error Absoluto Medio (mae).

4. Análisis Inicial del Conjunto de Datos

El conjunto de datos proporcionado obtenido del Portal Inmobiliario de Chile, contiene información sobre los precios de casas en la Región Metropolitana, con un total de 7,779 entradas y 12 columnas. Este registro data del 18 de Julio de 2023 y solo cuenta con casas en venta, excluyendo departamentos o terrenos. A continuación, se presenta un análisis inicial de las características del conjunto de datos:

Descripción de las Columnas

- **Price_CLP**: Precio en pesos chilenos (CLP).
- **Price_UF**: Precio en Unidades de Fomento (UF).
- **Price_USD**: Precio en dólares estadounidenses (USD).
- **Comuna**: Comuna donde se encuentra la propiedad.
- **Ubicación**: Dirección específica de la propiedad.
- **Dorms**: Número de dormitorios.
- **Baths**: Número de baños.
- **Built Area**: Área construida en metros cuadrados.
- **Total Area**: Área total del terreno en metros cuadrados.
- **Parking**: Número de espacios de estacionamiento.
- **id**: Identificador único de la propiedad.
- **Realtor**: Nombre de la inmobiliaria o agente inmobiliario.

Observaciones

- Las columnas **Baths**, **Built Area**, **Total Area**, **Parking** y **Realtor** tienen valores nulos que deberán ser tratados durante la fase de preparación de los datos.
- Los precios están disponibles en tres unidades diferentes (**CLP**, **UF**, **USD**), lo que permite flexibilidad en el análisis dependiendo de la unidad que se considere más relevante.
- Una limpieza de datos en este caso es de vital importancia, ya que como indica la misma fuente de donde fue sacada esta base de datos, al ser un Web Scrape es muy posible que hayan avisos cuya información fue ingresada incorrectamente. ej: Una casa con 100 km2 de terreno.

5. Método de Trabajo (CRISP-DM)

Utilizaremos la metodología CRISP-DM (Cross-Industry Standard Process for Data Mining) para estructurar el trabajo, que consta de las siguientes fases:

1. **Comprensión del negocio**: Entender el objetivo del análisis de precios de casas y sus implicaciones.
2. **Comprensión de los datos**: Explorar y familiarizarse con el conjunto de datos proporcionado.
3. **Preparación de los datos**: Limpiar y preparar los datos para el análisis.
4. **Modelado**: Seleccionar y aplicar técnicas de modelado predictivo adecuadas.
5. **Evaluación**: Evaluar la calidad y precisión de los modelos construidos.
6. **Despliegue**: Preparar el modelo final para su uso práctico.

6. Generación de Evidencias

A lo largo de cada etapa de la metodología CRISP-DM, se generarán y documentará las siguientes evidencias:

- **Comprensión de los datos**: Informes de análisis exploratorio de datos (EDA), incluyendo visualizaciones y estadísticas descriptivas.

  En este caso, se almacena la base de datos en la variable “dataset” y se realizan 3 acciones para verificar la data:

  1. Se llama a la variable para ver si la base de datos está bien alojada y en orden

```
[5] dataset
```

	Price_CLP	Price_UF	Price_USD	Comuna	Ubicacion	Dorms	Baths	Built_Area	Total_Area	Parking	id	Realtor
0	409285000	11500	509695	QuintaNormal	Hoevel4548y4558	7	4.0	384.0	732.0	3.0	11700213	NaN
1	105000000	2950	130760	PedroAguirreCerda	Rucalhue	2	1.0	112.0	145.0	1.0	10894299	Legales y Propiedades SpA
2	128124000	3600	159557	EstaciónCentral	AvenidaLasParcelas	3	1.0	59.0	243.0	2.0	10257635	Propiedadesrs
3	75000000	2107	93400	Colina	PasajeGonzaloRojas	3	1.0	103.0	73.0	1.0	9232092	Patricio Gajardo propiedades
4	53000000	1489	66002	Colina	HernánDíazArrieta2820	2	1.0	57.0	67.0	1.0	7085397	Patricio Gajardo propiedades
...	...	...	...	...	...	...	...	...	...	...	...	...
7774	2491300000	70000	3102491	LasCondes	CalleSanJosédeLaSierra	5	5.0	600.0	1800.0	5.0	4708915	Propiedades Viña Limitada ®
7775	242012000	6800	301385	Peñalolén	PasajeMarNegro	4	2.0	124.0	200.0	1.0	6641660	NaN
7776	3736950000	105000	4653736	LasCondes	CaminoLasFlores/CaminoPiedraRoja	5	7.0	460.0	4925.0	8.0	6032811	Tsi Property
7777	569440000	16000	709141	LaPintana	LosCipreses/LosDuraznos	4	2.0	311.0	2011.0	1.0	5314376	Tsi Property
7778	355828820	9998	443124	Talagante	LucasPacheco/Balmaceda	5	3.0	225.0	366.0	NaN	6186867	Tsi Property

7779 rows × 12 columns

  2. Se realiza un .describe() para obtener información relevante de las estadísticas descriptivas de los datos

```
[ ] dataset.describe()
```

	Price_CLP	Price_UF	Price_USD	Dorms	Baths	Built_Area	Total_Area	Parking	id
count	7.779000e+03	7779.000000	7.779000e+03	7779.000000	7714.000000	7533.000000	7571.000000	5489.000000	7.779000e+03
mean	3.642481e+08	10234.571153	4.536091e+05	3.994087	2.653746	229.923669	807.919826	2.980506	9.910828e+06
std	3.868810e+08	10870.491584	4.817945e+05	1.622821	1.465103	1676.898812	9050.893115	17.749384	2.046317e+06
min	2.085000e+03	0.000000	3.000000e+00	1.000000	1.000000	1.000000	1.000000	1.000000	1.213620e+06
25%	1.200000e+08	3372.000000	1.494400e+05	3.000000	2.000000	85.000000	129.500000	1.000000	8.563078e+06
50%	2.050000e+08	5760.000000	2.552930e+05	4.000000	2.000000	128.000000	210.000000	2.000000	1.054807e+07
75%	4.911420e+08	13800.000000	6.116340e+05	5.000000	3.000000	200.000000	443.000000	3.000000	1.152463e+07
max	5.516450e+09	155000.000000	6.869801e+06	27.000000	29.000000	120000.000000	678000.000000	1269.000000	1.234149e+07

3. Finalmente se realiza un .isna() con el objetivo de obtener que variables tienen valores null que podrían afectar a los modelos futuros

```
[ ]  dataset.isna().sum()

Price_CLP        0
Price_UF         0
Price_USD        0
Comuna           0
Ubicacion       42
Dorms            0
Baths           65
Built_Area     246
Total_Area     208
Parking       2290
id               0
Realtor        595
dtype: int64
```

- **Preparación de los datos**: Descripción de los pasos de limpieza y transformación de datos.
    1. Se realiza un .dropna() para eliminar los valores null detectados. Luego se realiza un .drop duplicates para eliminar posibles filas que estén duplicados en la base de datos y para finalizar se reinicia el índice de las variables para que queden en un buen orden ascendente. Se visualizan los datos para ver los resultados.

```
[6]  dataset.isna().sum()

Price_CLP        0
Price_UF         0
Price_USD        0
Comuna           0
Ubicacion       42
Dorms            0
Baths           65
Built_Area     246
Total_Area     208
Parking       2290
id               0
Realtor        595
dtype: int64
```

```
[ ]  dataset.dropna(axis=0, inplace=True)
     dataset.isna().sum()
     dataset = dataset.drop_duplicates()
```

```
[ ]  dataset.reset_index(drop=True, inplace=True)
     dataset.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4809 entries, 0 to 4808
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Price_CLP   4809 non-null   int64
 1   Price_UF    4809 non-null   int64
 2   Price_USD   4809 non-null   int64
 3   Comuna      4809 non-null   object
 4   Ubicacion   4809 non-null   object
 5   Dorms       4809 non-null   int64
 6   Baths       4809 non-null   float64
 7   Built_Area  4809 non-null   float64
 8   Total_Area  4809 non-null   float64
 9   Parking     4809 non-null   float64
 10  id          4809 non-null   int64
 11  Realtor     4809 non-null   object
dtypes: float64(4), int64(5), object(3)
memory usage: 451.0+ KB
```

2. Se le hace un .drop() a variables que no servirán en el modelo, siendo en este caso Price_CLP, Price_USD, Ubicacion, id y Realtor. Los dos primeros se sacan porque se utilizará el valor en UF para evaluar el modelo y los valores ubicación, id y realtor no aportan a los cálculos. Luego de esto se le asignan valores numéricos a cada comuna para mejorar los resultados finales.

```
[9]  dataset.drop(['Price_CLP', 'Price_USD','Ubicacion','id', 'Realtor'], axis=1, inplace=True)
```

```
<ipython-input-9-e90037d4b74f>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataset.drop(['Price_CLP', 'Price_USD','Ubicacion','id', 'Realtor'], axis=1, inplace=True)
```

```
[10] categorias_unicas = dataset['Comuna'].unique()
     mapeo_Comunas = {categoria: i for i, categoria in enumerate(categorias_unicas)}
     dataset['Comuna'] = dataset['Comuna'].map(mapeo_Comunas)
```

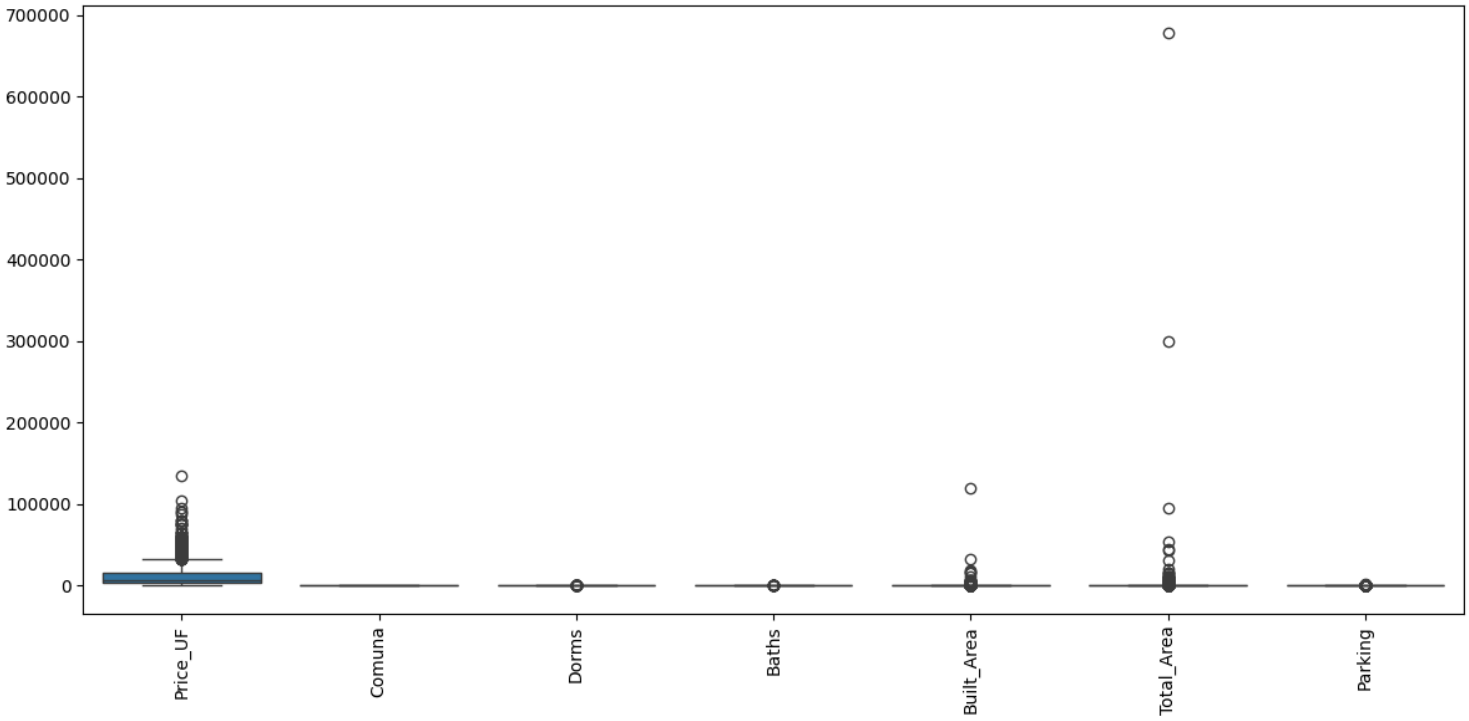
```
<ipython-input-10-906596f94c20>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataset['Comuna'] = dataset['Comuna'].map(mapeo_Comunas)
```

```
categorias_unicas
```

```
array(['PedroAguirreCerda', 'EstaciónCentral', 'Colina', 'LaFlorida',
       'Maipú', 'SanBernardo', 'QuintaNormal', 'Lampa', 'LaPintana',
       'PuenteAlto', 'Santiago', 'Huechuraba', 'SanMiguel', 'Ñuñoa',
       'Pudahuel', 'Buin', 'Quilicura', 'Tiltil', 'LoBarnechea',
       'Providencia', 'Conchalí', 'ElMonte', 'Macul', 'Peñalolén',
       'CaleradeTango', 'LasCondes', 'PadreHurtado', 'LaCisterna',
       'Melipilla', 'LaReina', 'Recoleta', 'Peñaflor', 'LoEspejo',
       'Vitacura', 'Talagante', 'ElBosque', 'Cerrillos', 'Renca',
       'SanJoaquín', 'IsladeMaipo', 'Independencia', 'Pirque', 'SanRamón',
       'Paine', 'LaGranja', 'LoPrado', 'Curacaví', 'CerroNavia',
       'SanJosédeMaipo'], dtype=object)
```

3. Se realiza un boxplot en busca de valores atípicos en las variables. Como se puede ver abajo, la variables total_area tiene ciertos valores que tienen valores altos con mucha diferencia de los demás, pudiendo complicar el rendimiento de los modelos. Para esto se realiza una limpieza de valores atípicos utilizando cuartiles para filtrar de forma más eficiente los datos de esta variable. Una vez eliminados estos datos atípicos, se realiza otro boxplot para verificar que los cambios tuvieron éxito.



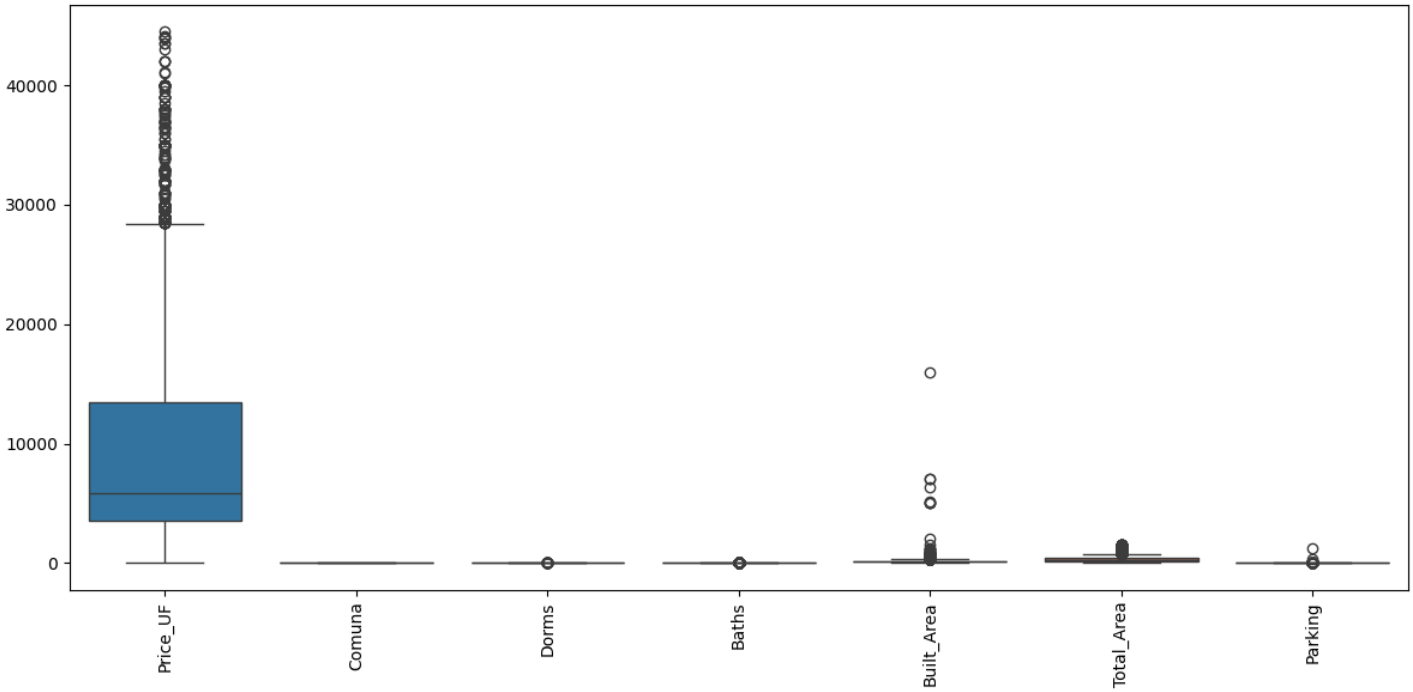

```
datasetcopy = dataset.copy()
```
✓ 0.0s

```
# Función para filtrar outliers basados en cuartiles
def filter_outliers_by_quantile(df, column_name):
    Q1 = df[column_name].quantile(0.25)
    Q3 = df[column_name].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 3 * IQR
    upper_bound = Q3 + 3 * IQR
    return df[(df[column_name] >= lower_bound) & (df[column_name] <= upper_bound)]

# Lista de columnas en las que quieres filtrar outliers
columns_to_filter = ['Total_Area','Price_UF']

# Aplica la función filter_outliers_by_quantile a cada columna en columns_to_filter

for column in columns_to_filter:
    datasetcopy = filter_outliers_by_quantile(datasetcopy, column)
```
✓ 0.0s



4. Se separan las variables Independientes y Dependientes, quedando en “X” todas las variables menos Price_UF, y en “Y” se guarda Price_UF, la cual es la variable a predecir. Para luego separar los datos en conjuntos de entrenamiento y testeo, con un “train_size = 0.8” que significa que se utilizará un 80% de los datos para entrenamiento y el 20% restante para prueba. Finalmente queda que del total de 4410 datos, 3528 quedaron para entrenamiento y 882 para testeo.

```
def scale_data(data):
    max = datasetcopy.max()
    min = datasetcopy.min()
    data_scaled = (data - min) / (max - min)

    return data_scaled
data_scaled = scale_data(datasetcopy)
```

✓ 0.0s

```
X = data_scaled.drop(['Price_UF'], axis=1)
Y = data_scaled['Price_UF']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.8, random_state=1)
print("Datos totales =", X.shape)
print("Train (80%) =", X_train.shape)
print("Testeo (20%) =", X_test.shape)
```

✓ 0.0s

```
Datos totales = (4410, 6)
Train (80%) = (3528, 6)
Testeo (20%) = (882, 6)
```

- **Modelado y Evaluación**: Resultados de los modelos aplicados, incluyendo hiperparámetros y selección de características. Métricas de desempeño de los modelos, como el MAE (Mean Absolute Error), RMSE (Root Mean Squared Error), y R^2 (coeficiente de determinación).

  Para el modelo se aplicó una regresión lineal ordinaria (OLS), donde los coeficientes de las variables representan el cambio en la variable dependiente (Price_UF) por cada unidad de cambio en la variable correspondiente sin afectar al resto de variables. Por ejemplo, una unidad de cambio en “Baths” cambia en promedio en 0.8763 unidades la variable “Price_UF”, mientras que las demás variables permanecen constantes. Las características utilizadas en este modelo son “Comuna”, ”Dorms”, “Baths”, “Built_Area”, “Total_Area” y “Parking”.

```
# Ajustar el modelo de regresión lineal
model = sm.OLS(Y, X).fit()
# Imprimir los resultados del modelo
print(model.summary())
```

  Finalmente para evaluar el modelo se tiene que el valor de R^2 0.871, lo que indica un 87.1% de variabilidad en “Price_UF” a partir de las variables independientes.

```
                            OLS Regression Results
==============================================================================================
Dep. Variable:               Price_UF   R-squared (uncentered):                   0.871
Model:                            OLS   Adj. R-squared (uncentered):              0.870
Method:                 Least Squares   F-statistic:                              4936.
Date:                Sun, 30 Jun 2024   Prob (F-statistic):                        0.00
Time:                        21:13:48   Log-Likelihood:                          3823.0
No. Observations:                4410   AIC:                                     -7634.
Df Residuals:                    4404   BIC:                                     -7596.
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Comuna         0.0672      0.006     11.273      0.000       0.055       0.079
Dorms         -0.0719      0.025     -2.870      0.004      -0.121      -0.023
Baths          0.8763      0.031     27.936      0.000       0.815       0.938
Built_Area     0.4116      0.073      5.674      0.000       0.269       0.554
Total_Area     0.6313      0.011     58.458      0.000       0.610       0.652
Parking        0.2047      0.099      2.064      0.039       0.010       0.399
==============================================================================
Omnibus:                      757.523   Durbin-Watson:                   1.820
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             7512.456
Skew:                           0.517   Prob(JB):                         0.00
Kurtosis:                       9.310   Cond. No.                         30.9
...
```

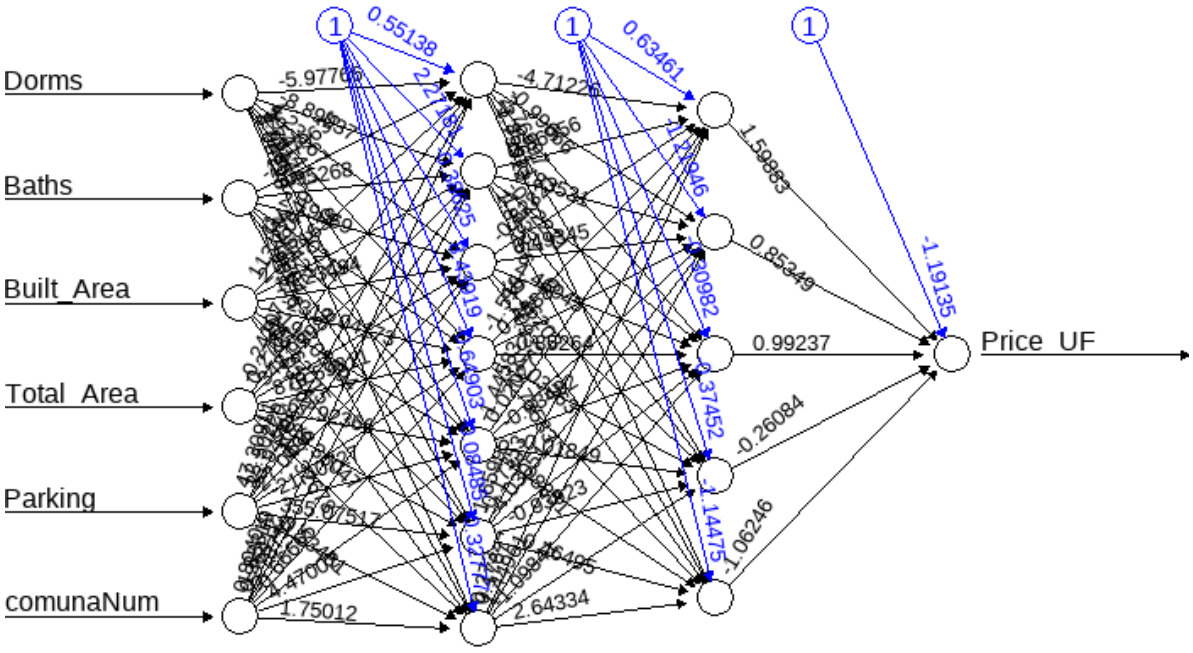
```python
mae = mean_absolute_error(Y_test, ypred)
mse = mean_squared_error(Y_test, ypred)
rmse = np.sqrt(mse)
r2 = r2_score(Y_test, ypred)

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

✓ 0.0s

```
Mean Absolute Error (MAE): 0.06677128284795408
Mean Squared Error (MSE): 0.010130877310357368
Root Mean Squared Error (RMSE): 0.10065225934055017
```

Haciendo la comparativa con el modelo realizado en R podemos ver lo siguiente:

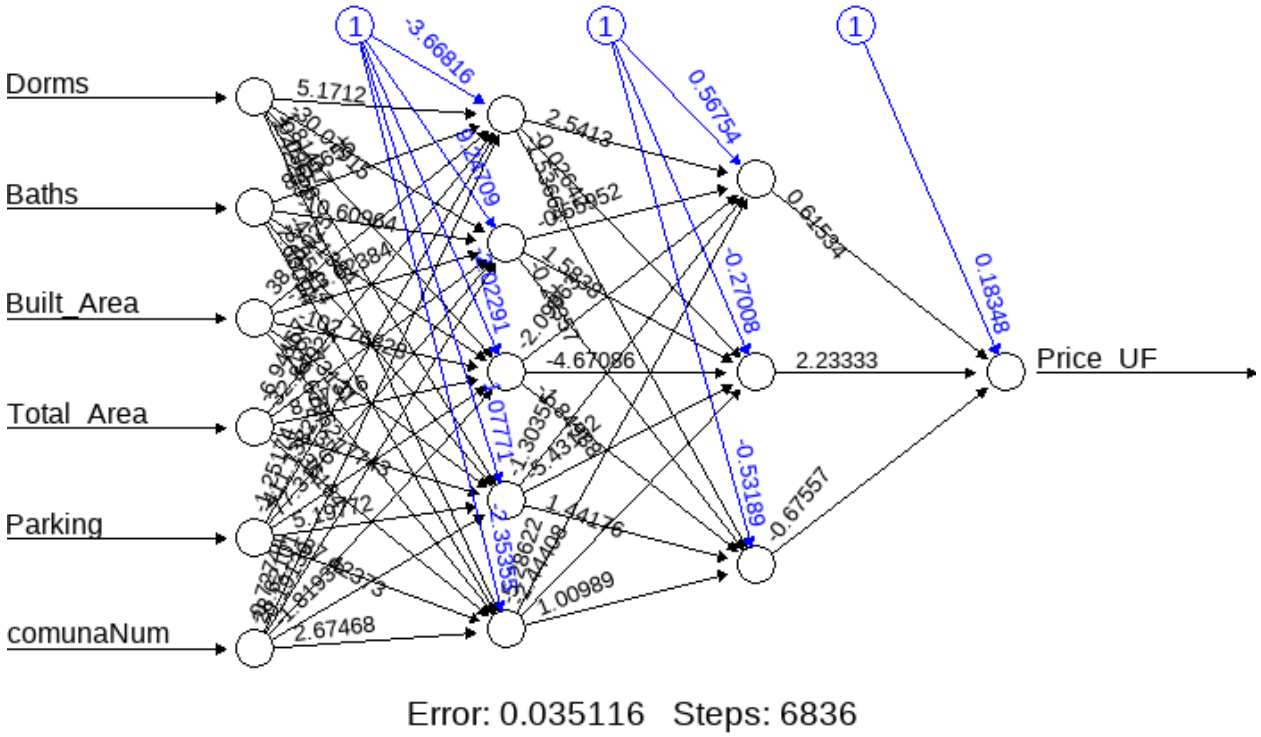


Luego se tiene que las predicciones realizadas en esta red neuronal, la cual consiste de 7 neuronas y 3 capas, están bastante alejadas de lo que debería ser la uf real.

Description: df [20 × 2]

	net_result <dbl>	uf_real <dbl>
3786	0.33645010	0.011141353
503	0.15187679	0.095232398
3430	0.13187341	0.070556918
3696	0.05985649	0.085511755
2035	0.21808730	0.009458934
3033	0.09394095	0.153556260
3790	0.05133925	0.011485314
3052	0.19153551	0.008830831
2967	0.15618553	0.005256625
470	0.02540873	0.069061434

Dado que el error cuadrático medio del primer modelo entregó un valor bastante alejado del valor óptimo (0), se optó por modificar los parámetros de la red neuronal y realizar un segundo modelo que consistía de 5 neuronas y 3 capas. En el segundo modelo realizado con la data de test se puede ver que el error disminuye de 3.3 a 0.35.



Con la disminución del error se puede ver que la predicción se acerca más a la uf real.

	net_result <dbl>	uf_real <dbl>
3	0.073754400	0.091396483
5	0.002199667	0.004232219
8	0.020932503	0.046554406
22	0.033492003	0.025004486
35	0.391244857	0.394329126
39	0.037730191	0.032795956
40	0.015284205	0.002504935
41	0.099303278	0.104579171
43	1.000565598	1.000000000
44	0.397955444	0.401806544

En la siguiente imagen vamos a poder visualizar las métricas que ocupamos para poder determinar si el modelo es acertado o erróneo.

```
"MSE: 0.000365787192429356"
"RMSE: 0.0191255638460506"
"MAE: 0.0134793409852426"
"R²: 0.969150998674256"
```