Universidad Politecnica de Yucatán

U2 - Implementing a Predictor from scratch

Subject : Machine Learning

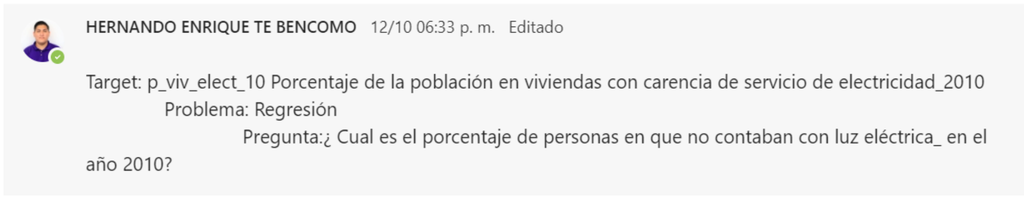
Grade and group: 9°B

Teacher: Victor Alejandro Ortiz

Student´s name: Hernando Enrique Te Bencomo

Date: 23/10/2023

Target selected:

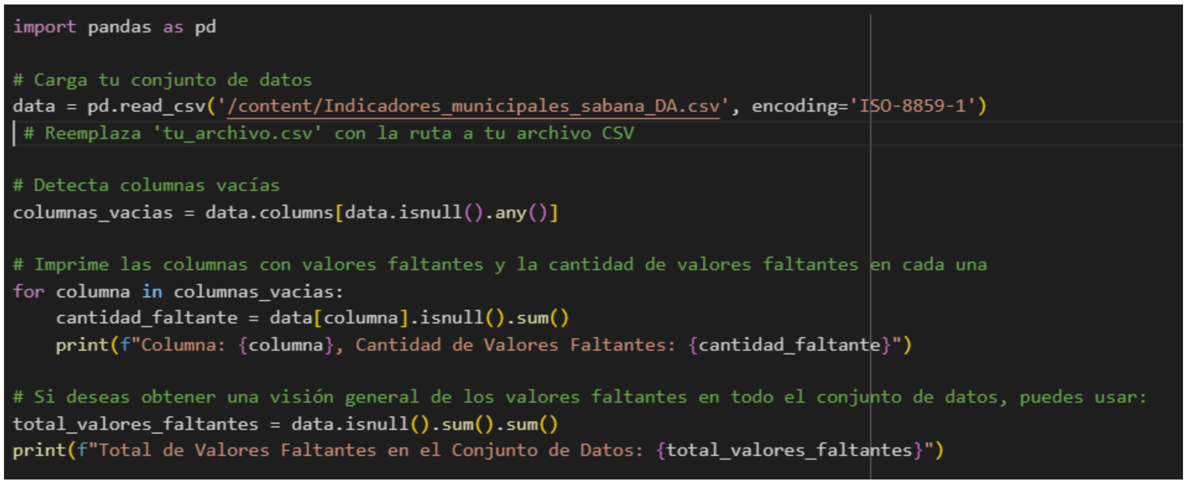

HERNANDO ENRIQUE TE BENCOMO 12/10 06:33 p. m. Editado

Target: p_viv_elect_10 Porcentaje de la población en viviendas con carencia de servicio de electricidad_2010
Problema: Regresión
Pregunta:¿ Cual es el porcentaje de personas en que no contaban con luz eléctrica_ en el año 2010?

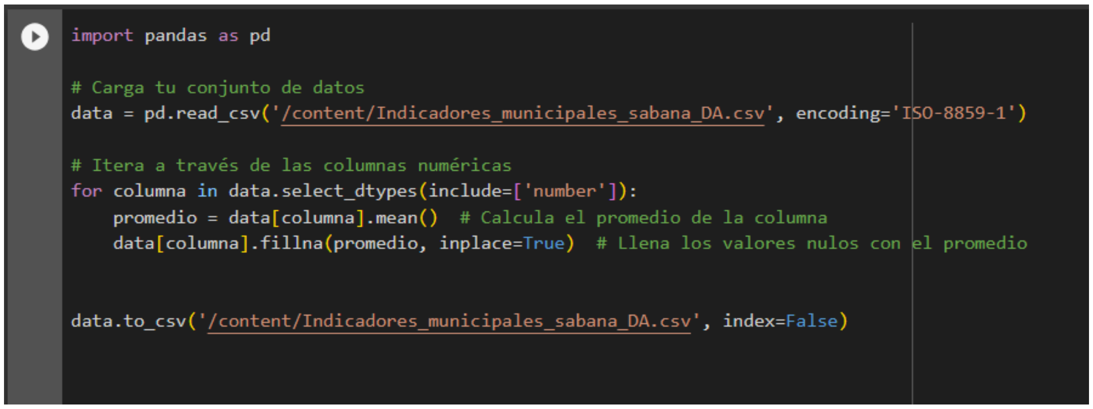
To read a dataset, the pandas pandas library is imported.

The dataset is read using the following part of the code, which loads files from Google Colab. Subsequently, the file path is copied, and the ISO-8859-1 encoding of a single byte is added, which means that each character is represented with a single byte (8 bits). This allows it to represent 256 different characters, including letters, numbers, punctuation marks, and special characters.

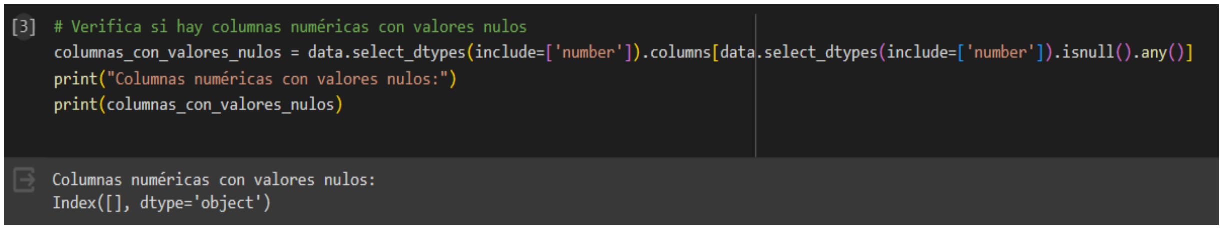
The next step was to search for columns with empty data in their rows, as this affects their manipulation concerning training and testing if empty data is retained.


```
import pandas as pd

# Carga tu conjunto de datos
data = pd.read_csv('/content/Indicadores_municipales_sabana_DA.csv', encoding='ISO-8859-1')
# Reemplaza 'tu_archivo.csv' con la ruta a tu archivo CSV

# Detecta columnas vacías
columnas_vacias = data.columns[data.isnull().any()]

# Imprime las columnas con valores faltantes y la cantidad de valores faltantes en cada una
for columna in columnas_vacias:
    cantidad_faltante = data[columna].isnull().sum()
    print(f"Columna: {columna}, Cantidad de Valores Faltantes: {cantidad_faltante}")

# Si deseas obtener una visión general de los valores faltantes en todo el conjunto de datos, puedes usar:
total_valores_faltantes = data.isnull().sum().sum()
print(f"Total de Valores Faltantes en el Conjunto de Datos: {total_valores_faltantes}")
```

Once the columns with empty rows have been determined, the average is used as a filling method, which is taken with respect to the rest of the data in the column to be filled in and the missing rows are filled in.

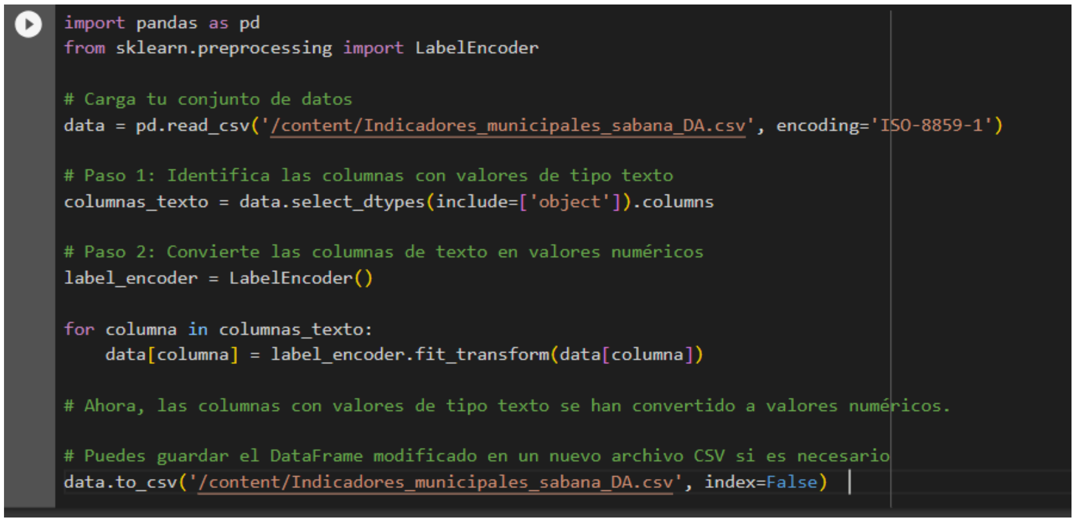

```
import pandas as pd

# Carga tu conjunto de datos
data = pd.read_csv('/content/Indicadores_municipales_sabana_DA.csv', encoding='ISO-8859-1')

# Itera a través de las columnas numéricas
for columna in data.select_dtypes(include=['number']):
    promedio = data[columna].mean()  # Calcula el promedio de la columna
    data[columna].fillna(promedio, inplace=True)  # Llena los valores nulos con el promedio


data.to_csv('/content/Indicadores_municipales_sabana_DA.csv', index=False)
```

These lines of code are used to check if the previous method worked to fill the columns with empty rows, which should be numeric

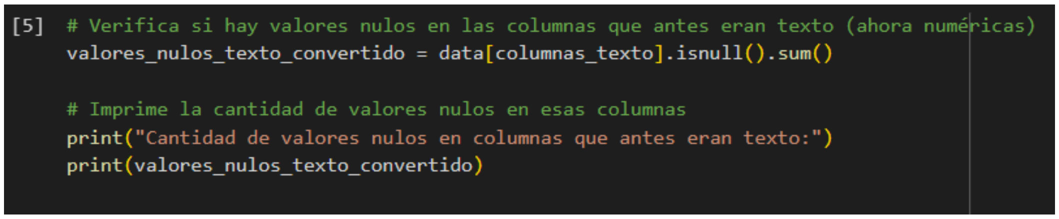


```
[3] # Verifica si hay columnas numéricas con valores nulos
    columnas_con_valores_nulos = data.select_dtypes(include=['number']).columns[data.select_dtypes(include=['number']).isnull().any()]
    print("Columnas numéricas con valores nulos:")
    print(columnas_con_valores_nulos)

    Columnas numéricas con valores nulos:
    Index([], dtype='object')
```

The next thing was that the columns that are not numerical but text type were identified and changed to numerical type. From these new columns of numeric type identify if there are null values.

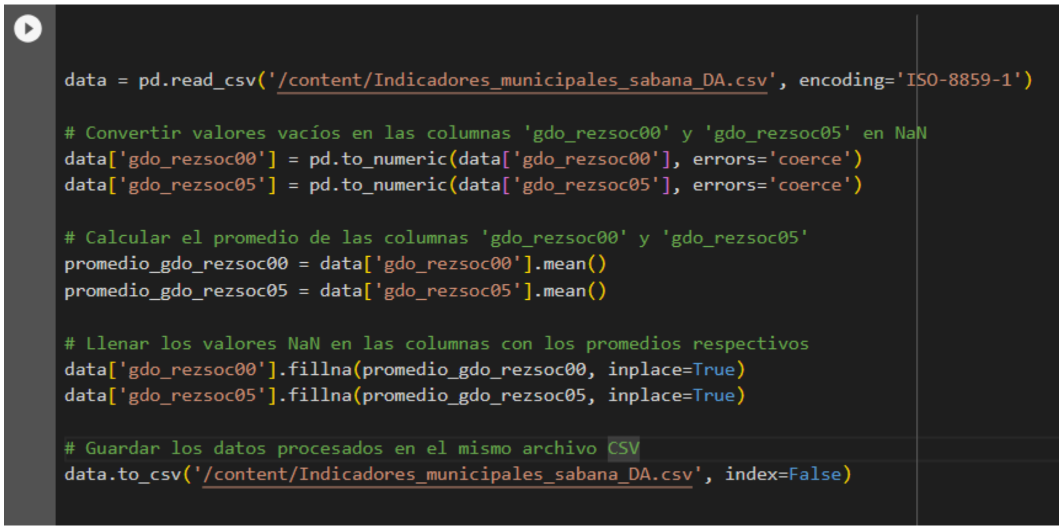


```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Carga tu conjunto de datos
data = pd.read_csv('/content/Indicadores_municipales_sabana_DA.csv', encoding='ISO-8859-1')

# Paso 1: Identifica las columnas con valores de tipo texto
columnas_texto = data.select_dtypes(include=['object']).columns

# Paso 2: Convierte las columnas de texto en valores numéricos
label_encoder = LabelEncoder()

for columna in columnas_texto:
    data[columna] = label_encoder.fit_transform(data[columna])

# Ahora, las columnas con valores de tipo texto se han convertido a valores numéricos.

# Puedes guardar el DataFrame modificado en un nuevo archivo CSV si es necesario
data.to_csv('/content/Indicadores_municipales_sabana_DA.csv', index=False)
```



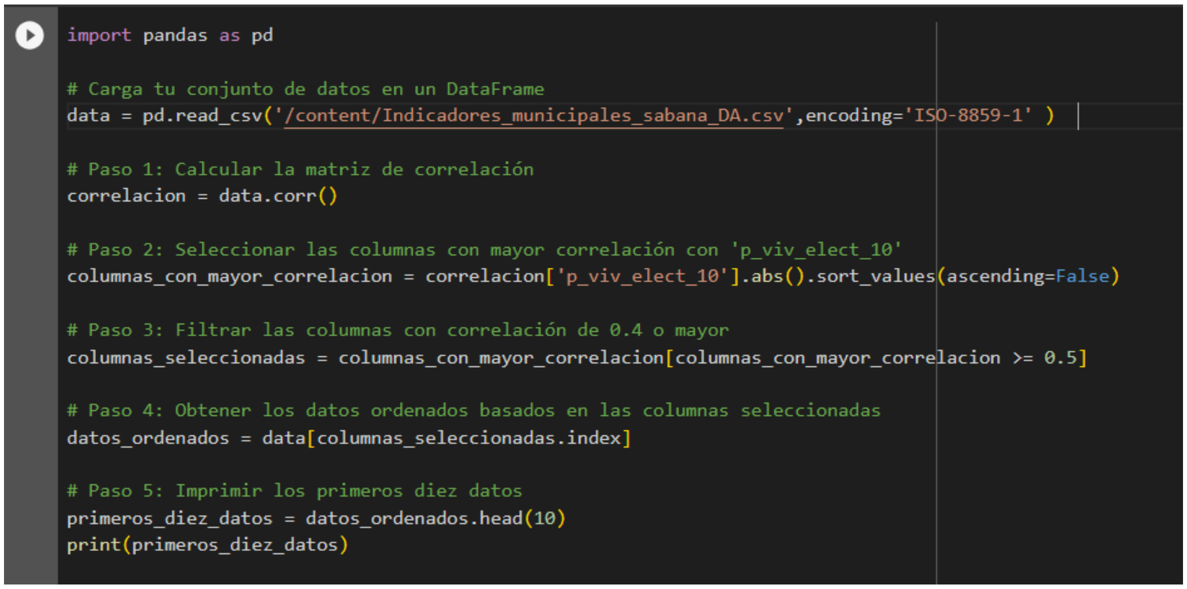
```
[5] # Verifica si hay valores nulos en las columnas que antes eran texto (ahora numéricas)
    valores_nulos_texto_convertido = data[columnas_texto].isnull().sum()

    # Imprime la cantidad de valores nulos en esas columnas
    print("Cantidad de valores nulos en columnas que antes eran texto:")
    print(valores_nulos_texto_convertido)
```

The change of the columns with string values to numerical values was made because it was determined that there were columns with null rows, being the features 'gdo_rezsoc00' and 'gdo_rezsoc05' columns with empty rows. that is why the same technique of averaging was applied to fill the empty rows.



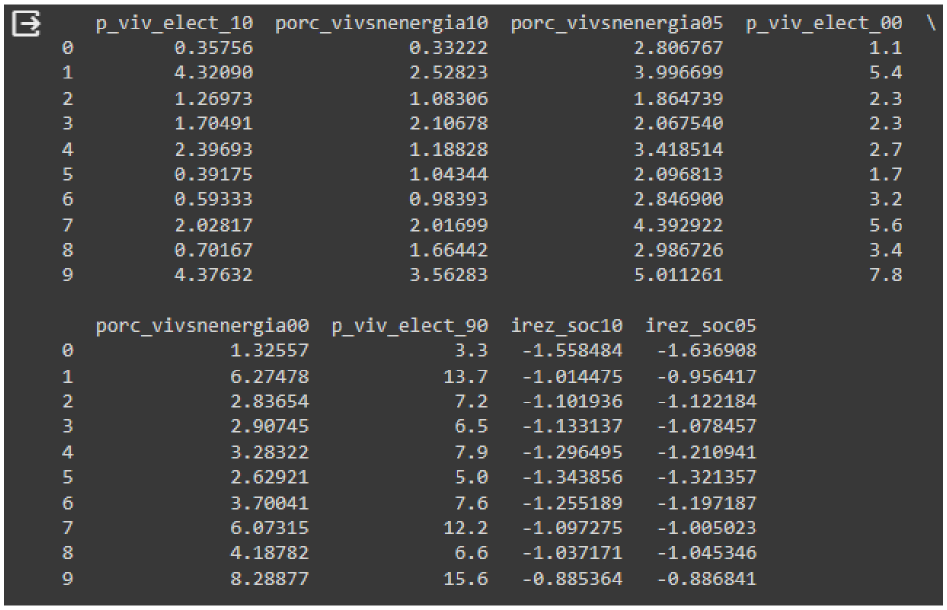
```python
data = pd.read_csv('/content/Indicadores_municipales_sabana_DA.csv', encoding='ISO-8859-1')

# Convertir valores vacíos en las columnas 'gdo_rezsoc00' y 'gdo_rezsoc05' en NaN
data['gdo_rezsoc00'] = pd.to_numeric(data['gdo_rezsoc00'], errors='coerce')
data['gdo_rezsoc05'] = pd.to_numeric(data['gdo_rezsoc05'], errors='coerce')

# Calcular el promedio de las columnas 'gdo_rezsoc00' y 'gdo_rezsoc05'
promedio_gdo_rezsoc00 = data['gdo_rezsoc00'].mean()
promedio_gdo_rezsoc05 = data['gdo_rezsoc05'].mean()

# Llenar los valores NaN en las columnas con los promedios respectivos
data['gdo_rezsoc00'].fillna(promedio_gdo_rezsoc00, inplace=True)
data['gdo_rezsoc05'].fillna(promedio_gdo_rezsoc05, inplace=True)

# Guardar los datos procesados en el mismo archivo CSV
data.to_csv('/content/Indicadores_municipales_sabana_DA.csv', index=False)
```

With the dataset already modified, we proceeded to extract the correlation of the target 'p_viv_elect_10', which is the percentage of people without electricity in their homes in 2010, then the 10 best features that have a close relationship with the target where the Pearson correlation which is a measure of the linear relationship between two variables. Pearson correlation measures the strength and direction of the relationship between the target variable and the predictor variables. Its value varies between -1 and 1. A value of 1 indicates a perfect positive correlation, -1 indicates a perfect negative correlation, and 0 indicates no linear correlation.



```python
import pandas as pd

# Carga tu conjunto de datos en un DataFrame
data = pd.read_csv('/content/Indicadores_municipales_sabana_DA.csv',encoding='ISO-8859-1' )

# Paso 1: Calcular la matriz de correlación
correlacion = data.corr()

# Paso 2: Seleccionar las columnas con mayor correlación con 'p_viv_elect_10'
columnas_con_mayor_correlacion = correlacion['p_viv_elect_10'].abs().sort_values(ascending=False)

# Paso 3: Filtrar las columnas con correlación de 0.4 o mayor
columnas_seleccionadas = columnas_con_mayor_correlacion[columnas_con_mayor_correlacion >= 0.5]

# Paso 4: Obtener los datos ordenados basados en las columnas seleccionadas
datos_ordenados = data[columnas_seleccionadas.index]

# Paso 5: Imprimir los primeros diez datos
primeros_diez_datos = datos_ordenados.head(10)
print(primeros_diez_datos)
```

These were the columns that had the greatest correlation with our target. From here begins the division of our dataset into our X and our Y.



```
   p_viv_elect_10  porc_vivsnenergia10  porc_vivsnenergia05  p_viv_elect_00  \
0         0.35756              0.33222             2.806767             1.1
1         4.32090              2.52823             3.996699             5.4
2         1.26973              1.08306             1.864739             2.3
3         1.70491              2.10678             2.067540             2.3
4         2.39693              1.18828             3.418514             2.7
5         0.39175              1.04344             2.096813             1.7
6         0.59333              0.98393             2.846900             3.2
7         2.02817              2.01699             4.392922             5.6
8         0.70167              1.66442             2.986726             3.4
9         4.37632              3.56283             5.011261             7.8

   porc_vivsnenergia00  p_viv_elect_90  irez_soc10  irez_soc05
0              1.32557             3.3   -1.558484   -1.636908
1              6.27478            13.7   -1.014475   -0.956417
2              2.83654             7.2   -1.101936   -1.122184
3              2.90745             6.5   -1.133137   -1.078457
4              3.28322             7.9   -1.296495   -1.210941
5              2.62921             5.0   -1.343856   -1.321357
6              3.70041             7.6   -1.255189   -1.197187
7              6.07315            12.2   -1.097275   -1.005023
8              4.18782             6.6   -1.037171   -1.045346
9              8.28877            15.6   -0.885364   -0.886841
```

Importing train_test_split from sklearn.model_selection in Python is used to split a data set into two subsets: one for training and one for testing.

The function returns four sets of data (NumPy arrays or similar Python data structures):

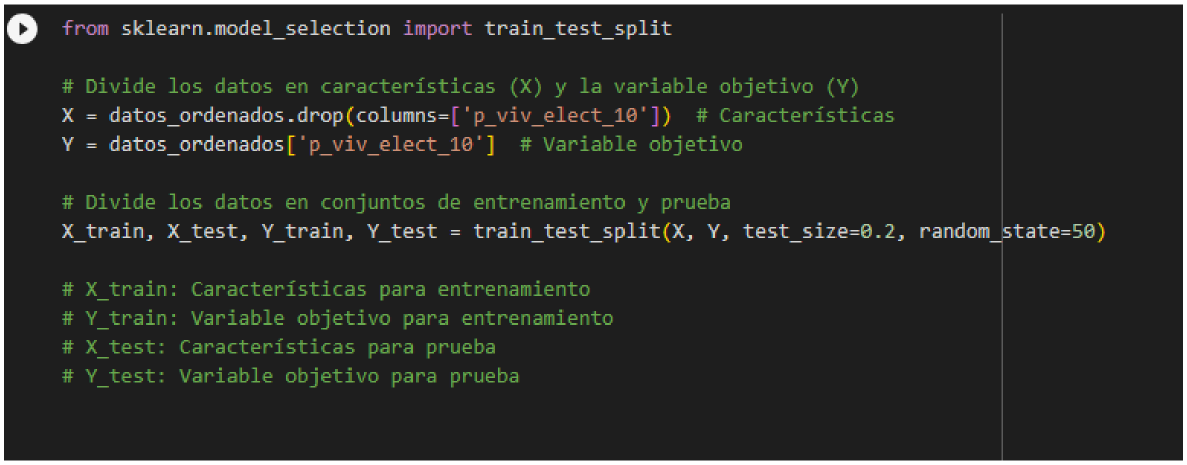
X_train: Training feature set.

X_test: Test feature set.

y_train: Set of training labels (only if you are working on a supervised problem).

y_test: Set of test labels (only if you are working on a supervised problem).

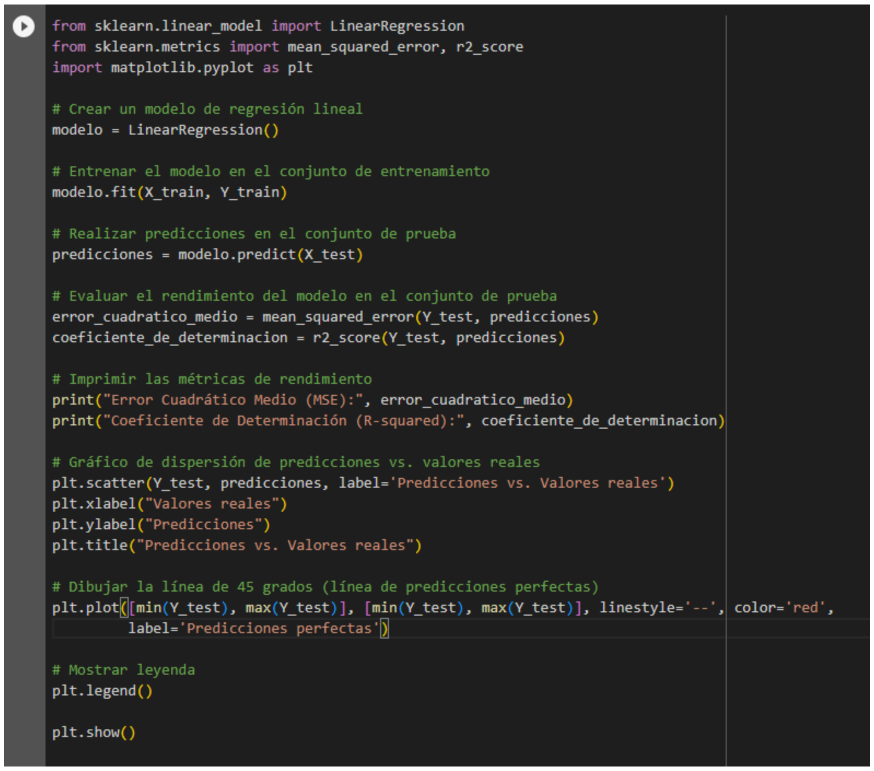
Where the train grabbed 80% of the data and the test 20%.



```python
from sklearn.model_selection import train_test_split

# Divide los datos en características (X) y la variable objetivo (Y)
X = datos_ordenados.drop(columns=['p_viv_elect_10'])  # Características
Y = datos_ordenados['p_viv_elect_10']  # Variable objetivo

# Divide los datos en conjuntos de entrenamiento y prueba
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=50)

# X_train: Características para entrenamiento
# Y_train: Variable objetivo para entrenamiento
# X_test: Características para prueba
# Y_test: Variable objetivo para prueba
```

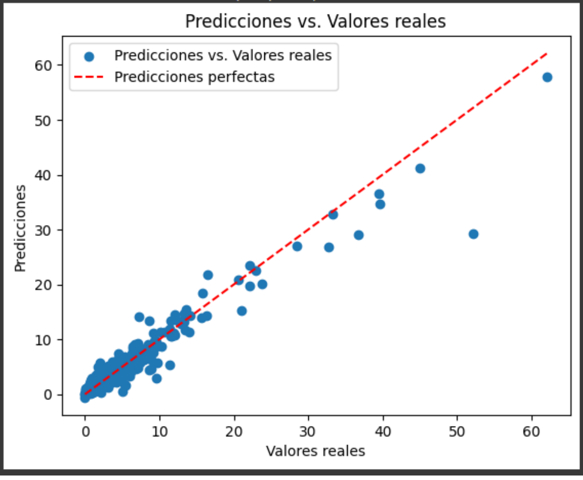
Once the dataset is finished training, the linear regression model is developed that will determine the percentage of people in any of the 32 states who did not have electricity in their homes in 2010.

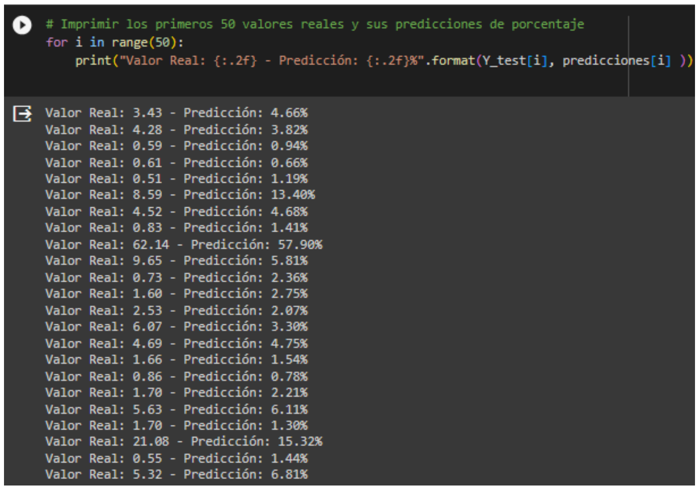
The model was trained, predictions were made on the test set, from there to evaluate the performance of the model using the mean square error which measures the quality of the model predictions by calculating the average of the squared errors between the model predictions and the actual values in a data set.
Likewise, to evaluate the coefficient of determination, which measures the goodness of fit of the model, the proportion of variability in the data can be explained by the predictions of the model.



```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Crear un modelo de regresión lineal
modelo = LinearRegression()

# Entrenar el modelo en el conjunto de entrenamiento
modelo.fit(X_train, Y_train)

# Realizar predicciones en el conjunto de prueba
predicciones = modelo.predict(X_test)

# Evaluar el rendimiento del modelo en el conjunto de prueba
error_cuadratico_medio = mean_squared_error(Y_test, predicciones)
coeficiente_de_determinacion = r2_score(Y_test, predicciones)

# Imprimir las métricas de rendimiento
print("Error Cuadrático Medio (MSE):", error_cuadratico_medio)
print("Coeficiente de Determinación (R-squared):", coeficiente_de_determinacion)

# Gráfico de dispersión de predicciones vs. valores reales
plt.scatter(Y_test, predicciones, label='Predicciones vs. Valores reales')
plt.xlabel("Valores reales")
plt.ylabel("Predicciones")
plt.title("Predicciones vs. Valores reales")

# Dibujar la línea de 45 grados (línea de predicciones perfectas)
plt.plot([min(Y_test), max(Y_test)], [min(Y_test), max(Y_test)], linestyle='--', color='red',
         label='Predicciones perfectas')

# Mostrar leyenda
plt.legend()

plt.show()
```

```
Error Cuadrático Medio (MSE): 2.944372319500498
Coeficiente de Determinación (R-squared): 0.9281920409668123
```

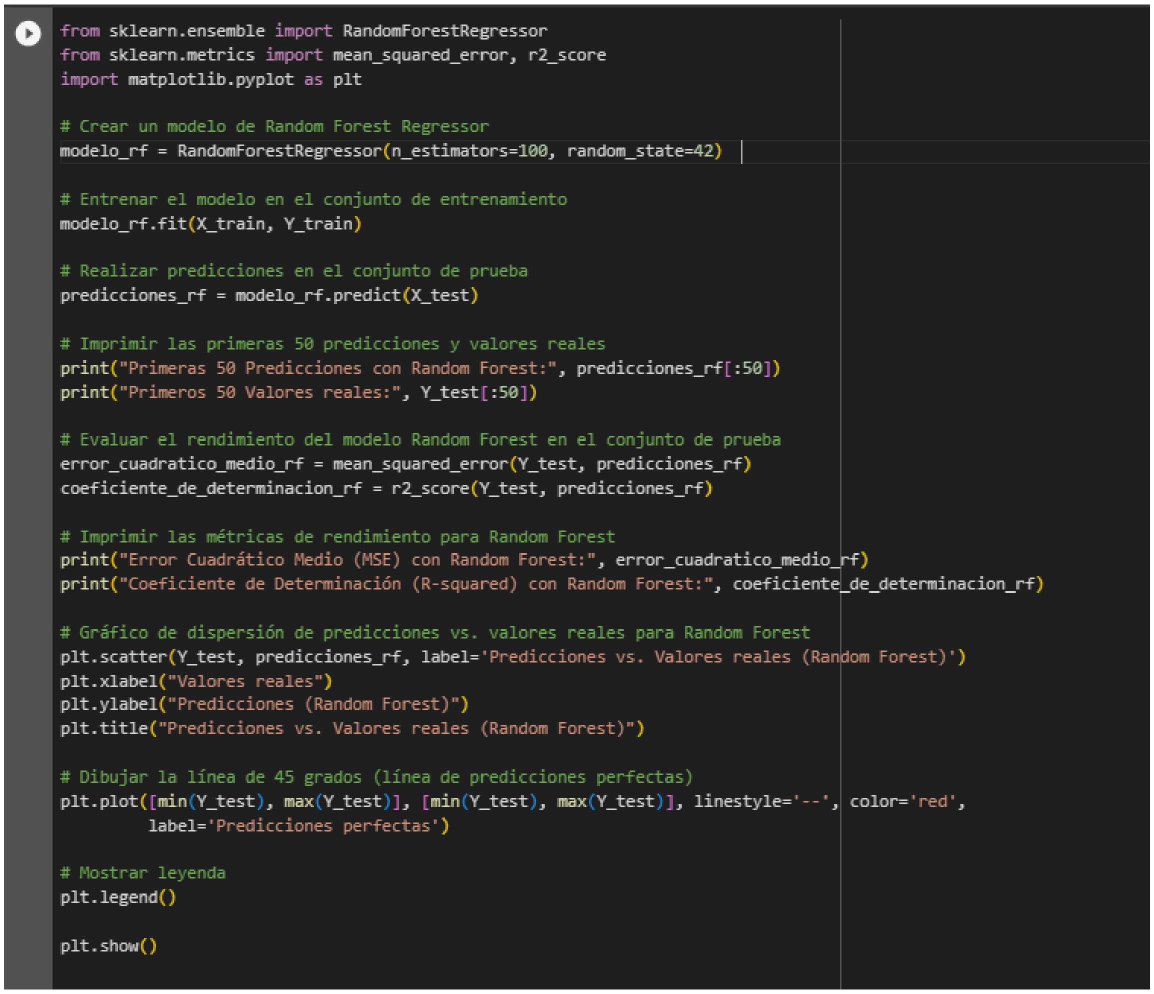




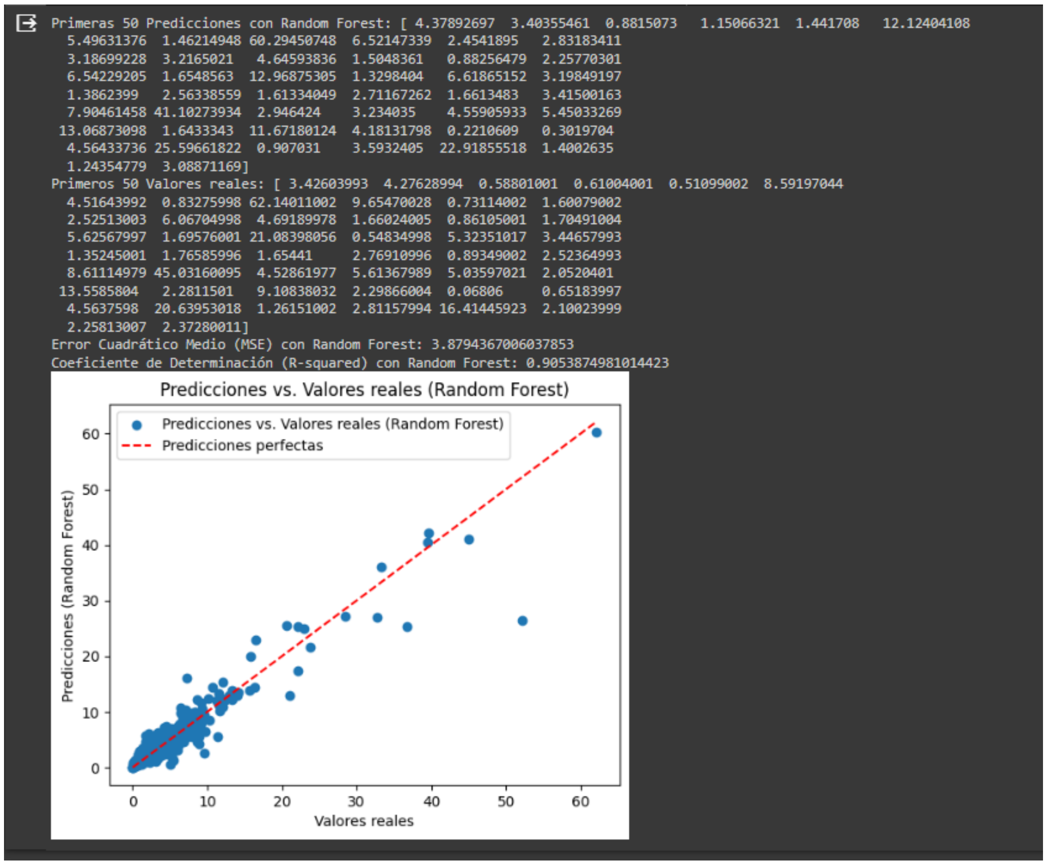
```python
# Imprimir los primeros 50 valores reales y sus predicciones de porcentaje
for i in range(50):
    print("Valor Real: {:.2f} - Predicción: {:.2f}%".format(Y_test[i], predicciones[i] ))
```

```
Valor Real: 3.43 - Predicción: 4.66%
Valor Real: 4.28 - Predicción: 3.82%
Valor Real: 0.59 - Predicción: 0.94%
Valor Real: 0.61 - Predicción: 0.66%
Valor Real: 0.51 - Predicción: 1.19%
Valor Real: 8.59 - Predicción: 13.40%
Valor Real: 4.52 - Predicción: 4.68%
Valor Real: 0.83 - Predicción: 1.41%
Valor Real: 62.14 - Predicción: 57.90%
Valor Real: 9.65 - Predicción: 5.81%
Valor Real: 0.73 - Predicción: 2.36%
Valor Real: 1.60 - Predicción: 2.75%
Valor Real: 2.53 - Predicción: 2.07%
Valor Real: 6.07 - Predicción: 3.30%
Valor Real: 4.69 - Predicción: 4.75%
Valor Real: 1.66 - Predicción: 1.54%
Valor Real: 0.86 - Predicción: 0.78%
Valor Real: 1.70 - Predicción: 2.21%
Valor Real: 5.63 - Predicción: 6.11%
Valor Real: 1.70 - Predicción: 1.30%
Valor Real: 21.08 - Predicción: 15.32%
Valor Real: 0.55 - Predicción: 1.44%
Valor Real: 5.32 - Predicción: 6.81%
```

This code shows how to use a RandomForestRegressor regression model from scikit-learn, where the model was trained, to make predictions and evaluate its performance.
The necessary libraries were imported. RandomForestRegressor is the class that was used to create a regression model based on Random Forest. mean_squared_error and r2_score are functions for calculating performance metrics. matplotlib.pyplot is used to create visualizations.
The RandomForestRegressor model was instantiated with 100 trees in the forest (n_estimators=100) and the random seed (random_state=42) was set to ensure reproducibility.
The RandomForestRegressor model is instantiated with 100 trees in the forest (n_estimators=100) and the random seed (random_state=42) is set to ensure reproducibility.
The trained model was used to make predictions on a test set. X_test is the features of the test set, and predictions_rf will contain the predictions made by the Random Forest model.
The first 50 predictions made by the model and the first 50 actual values from the test set are printed for quick viewing.
Two performance metrics are calculated: the Mean Squared Error (MSE) and the Coefficient of Determination (R-squared). These metrics are used to evaluate how well the model fits the test data. Y_test are the actual test set labels.
A scatter plot is created that shows how the model predictions compare to the actual values. Actual values are plotted on the x-axis, and predictions are plotted on the y-axis. This graph allows you to visualize the performance of the model.



```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Crear un modelo de Random Forest Regressor
modelo_rf = RandomForestRegressor(n_estimators=100, random_state=42)

# Entrenar el modelo en el conjunto de entrenamiento
modelo_rf.fit(X_train, Y_train)

# Realizar predicciones en el conjunto de prueba
predicciones_rf = modelo_rf.predict(X_test)

# Imprimir las primeras 50 predicciones y valores reales
print("Primeras 50 Predicciones con Random Forest:", predicciones_rf[:50])
print("Primeros 50 Valores reales:", Y_test[:50])

# Evaluar el rendimiento del modelo Random Forest en el conjunto de prueba
error_cuadratico_medio_rf = mean_squared_error(Y_test, predicciones_rf)
coeficiente_de_determinacion_rf = r2_score(Y_test, predicciones_rf)

# Imprimir las métricas de rendimiento para Random Forest
print("Error Cuadrático Medio (MSE) con Random Forest:", error_cuadratico_medio_rf)
print("Coeficiente de Determinación (R-squared) con Random Forest:", coeficiente_de_determinacion_rf)

# Gráfico de dispersión de predicciones vs. valores reales para Random Forest
plt.scatter(Y_test, predicciones_rf, label='Predicciones vs. Valores reales (Random Forest)')
plt.xlabel("Valores reales")
plt.ylabel("Predicciones (Random Forest)")
plt.title("Predicciones vs. Valores reales (Random Forest)")

# Dibujar la línea de 45 grados (línea de predicciones perfectas)
plt.plot([min(Y_test), max(Y_test)], [min(Y_test), max(Y_test)], linestyle='--', color='red',
         label='Predicciones perfectas')

# Mostrar leyenda
plt.legend()

plt.show()
```



```
Primeras 50 Predicciones con Random Forest: [ 4.37892697  3.40355461  0.8815073   1.15066321  1.441708   12.12404108
  5.49631376  1.46214948 60.29450748  6.52147339  2.4541895   2.83183411
  3.18699228  3.2165021   4.64593836  1.5048361   0.88256479  2.25770301
  6.54229205  1.6548563  12.96875305  1.3298404   6.61865152  3.19849197
  1.3862399   2.56338559  1.61334049  2.71167262  1.6613483   3.41500163
  7.90461458 41.10273934  2.946424    3.234035    4.55905933  5.45033269
 13.06873098  1.6433343  11.67180124  4.18131798  0.2210609   0.3019704
  4.56433736 25.59661822  0.907031    3.5932405  22.91855518  1.4002635
  1.24354779  3.08871169]
Primeros 50 Valores reales: [ 3.42603993  4.27628994  0.58801001  0.61004001  0.51099002  8.59197044
  4.51643992  0.83275998 62.14011002  9.65470028  0.73114002  1.60079002
  2.52513003  6.06704998  4.69189978  1.66024005  0.86105001  1.70491004
  5.62567997  1.69576001 21.08398056  0.54834998  5.32351017  3.44657993
  1.35245001  1.76585996  1.65441     2.76910996  0.89349002  2.52364993
  8.61114979 45.03160095  4.52861977  5.61367989  5.03597021  2.0520401
 13.5585804   2.2811501   9.10838032  2.29866004  0.06806     0.65183997
  4.5637598  20.63953018  1.26151002  2.81157994 16.41445923  2.10023999
  2.25813007  2.37280011]
Error Cuadrático Medio (MSE) con Random Forest: 3.8794367006037853
Coeficiente de Determinación (R-squared) con Random Forest: 0.9053874981014423
```

Predicciones vs. Valores reales (Random Forest)
● Predicciones vs. Valores reales (Random Forest)
Predicciones perfectas
Predicciones (Random Forest)
Valores reales
0
10
20
30
40
50
60

Random Forest is easy to implement and tune. It does not require as delicate hyperparameter tuning as other algorithms and tends to work reasonably well "out of the box".

Random Forest is a solid choice in regression problems when looking for an accurate model that is resistant to overfitting. It is used in a wide range of regression applications such as sales forecasting, price prediction, risk assessment and many others.

A link to a github project where you publish your code.

[MC-deliver/TP1/U2_Implementing_a_Predictor_from_scratch_Hernanedo_Te_9°B.ipynb at main · H3RT3BE/MC-deliver (github.com)]

a. A reflexion over what were your main challenges during this project, how do you see this could be applied to the field of Robotics.

   One of the biggest difficulties was having a dataset that did not have columns with empty boxes since this, if it had been handled with missing data, would have affected the training and the prediction of what is being worked on, so methods that allow the correct resolution of empty data.

   In the field of computational robotics where this would be worked on would be in sensors of any device, if any data is empty, the prediction that I make after training it and when testing it could be incorrect, which is why one of the important parts is to verify that The dataset is complete as well as whether the values are of type, whether numerical, string or other, but in most cases it will be numerical from there to work calmly and then apply all the process that it entails.

   Another important point is that having such a large data you should not think about going column by column verifying which ones will be useful for my target to manage, but instead look for methods such as in this case where a correlation method was used so that it is easier to select. What features are in line with my target.