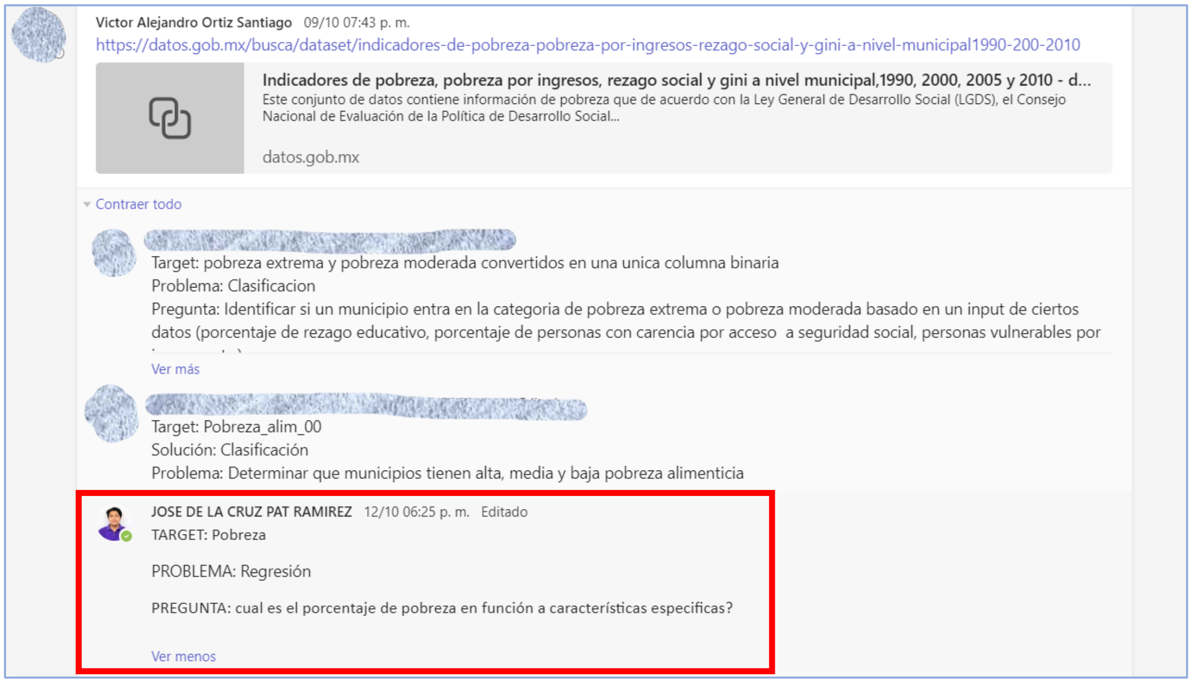
U2 - Implementing a Predictor from scratch

Machine learning

Jose De La Cruz Pat Ramirez
UNIVERSIDAD POLITÉCNICA DE YUCATÁN
23/10/2023

Target, Problem



Victor Alejandro Ortiz Santiago 09/10 07:43 p. m.
https://datos.gob.mx/busca/dataset/indicadores-de-pobreza-pobreza-por-ingresos-rezago-social-y-gini-a-nivel-municipal1990-200-2010

Indicadores de pobreza, pobreza por ingresos, rezago social y gini a nivel municipal,1990, 2000, 2005 y 2010 - d...
Este conjunto de datos contiene información de pobreza que de acuerdo con la Ley General de Desarrollo Social (LGDS), el Consejo Nacional de Evaluación de la Política de Desarrollo Social...

datos.gob.mx

Contraer todo

[redacted]
Target: pobreza extrema y pobreza moderada convertidos en una unica columna binaria
Problema: Clasificacion
Pregunta: Identificar si un municipio entra en la categoria de pobreza extrema o pobreza moderada basado en un input de ciertos datos (porcentaje de rezago educativo, porcentaje de personas con carencia por acceso a seguridad social, personas vulnerables por

Ver más

[redacted]
Target: Pobreza_alim_00
Solución: Clasificación
Problema: Determinar que municipios tienen alta, media y baja pobreza alimenticia

JOSE DE LA CRUZ PAT RAMIREZ 12/10 06:25 p. m. Editado
TARGET: Pobreza

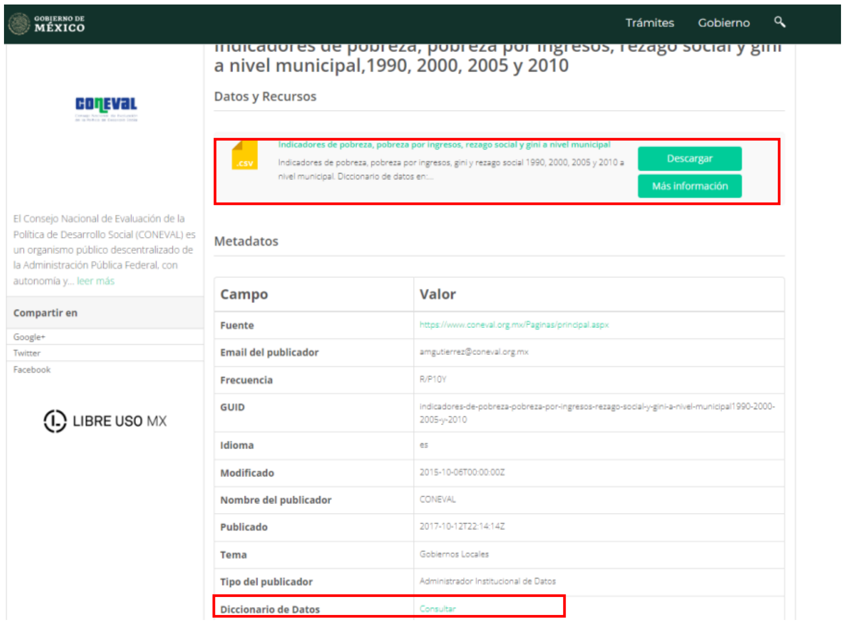
PROBLEMA: Regresión

PREGUNTA: cual es el porcentaje de pobreza en función a características especificas?

Ver menos

Dataset manipulation

1.- First thing to do is search, download and decompress the dataset, in this case the dataset is poverty indices in Mexico, it was obtained from the following link:
https://datos.gob.mx/busca/dataset/indicadores-de-pobreza-pobreza-por-ingresos-rezago-social-y-gini-a-nivel-municipal1990-200-2010
In the same way, the data dictionary obtained is downloaded in the same place.



Indicadores de pobreza, pobreza por ingresos, rezago social y gini a nivel municipal,1990, 2000, 2005 y 2010

Datos y Recursos

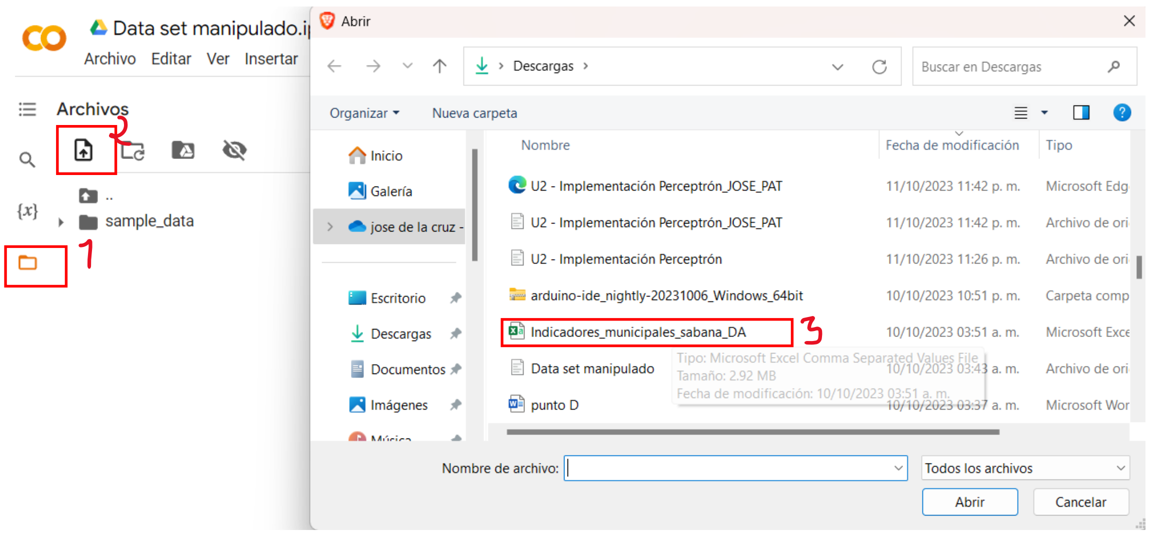
Indicadores de pobreza, pobreza por ingresos, rezago social y gini a nivel municipal
Indicadores de pobreza, pobreza por ingresos, gini y rezago social 1990, 2000, 2005 y 2010 a nivel municipal. Diccionario de datos en:...

El Consejo Nacional de Evaluación de la Política de Desarrollo Social (CONEVAL) es un organismo público descentralizado de la Administración Pública Federal, con autonomía y... leer más

Metadatos

Campo	Valor
Fuente	https://www.coneval.org.mx/Paginas/principal.aspx
Email del publicador	amgutierrez@coneval.org.mx
Frecuencia	R/P10Y
GUID	indicadores-de-pobreza-pobreza-por-ingresos-rezago-social-y-gini-a-nivel-municipal1990-2000-2005-y-2010
Idioma	es
Modificado	2015-10-06T00:00:00Z
Nombre del publicador	CONEVAL
Publicado	2017-10-12T22:14:14Z
Tema	Gobiernos Locales
Tipo del publicador	Administrador Institucional de Datos
Diccionario de Datos	Consultar

2.- As a second step, load the dataset to the work environment (Google Collab)



Nombre	Fecha de modificación	Tipo
U2 - Implementación Perceptrón_JOSE_PAT	11/10/2023 11:42 p. m.	Microsoft Edg
U2 - Implementación Perceptrón_JOSE_PAT	11/10/2023 11:42 p. m.	Archivo de ori
U2 - Implementación Perceptrón	11/10/2023 11:26 p. m.	Archivo de ori
arduino-ide_nightly-20231006_Windows_64bit	10/10/2023 10:51 p. m.	Carpeta comp
Indicadores_municipales_sabana_DA	10/10/2023 03:51 a. m.	Microsoft Exce
Data set manipulado	10/10/2023 03:43 a. m.	Archivo de ori
punto D	10/10/2023 03:37 a. m.	Microsoft Wor

Tipo: Microsoft Excel Comma Separated Values File
Tamaño: 2.92 MB
Fecha de modificación: 10/10/2023 03:51 a. m.

3.- Once the dataset has been uploaded, it must be imported for reading through a code that contains "Pandas", a Python library to load a CSV file into a Data Frame.

```
import pandas as pd

df = pd.read_csv('/content/Indicadores_municipales_sabana_DA.csv',
encoding='ISO-8859-1')
```
+
What is being done is defining a data frame that will contain the information from the CSV file uploaded and the file read with the parameter 'ISO-8859-1', which is a type of character encoding, and is used to ensure that the characters specials are interpreted correctly when reading the CSV file.

4.- The dataset is explored a little to see its dimensions. In this case, it is observed that the data set has 139 columns, that is, we have 138 features and our target.
This is done with the function:
```
print(df.head())
```
which prints the column headers and the first 5 entries, and also shows the dimension of the dataset.

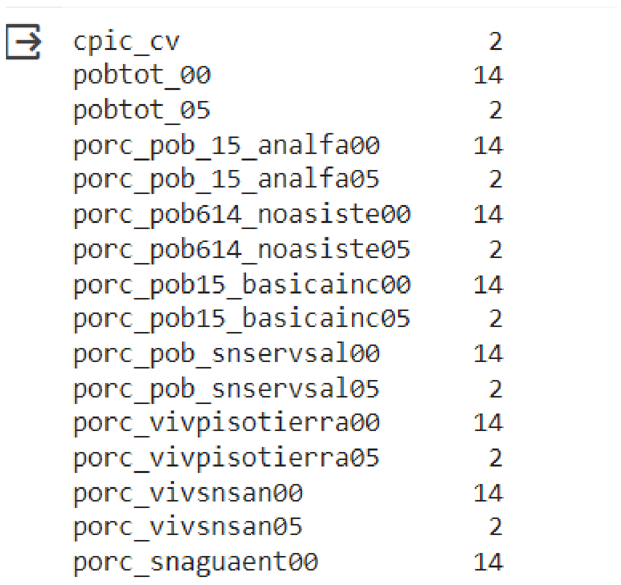
```
   pobreza_patrim_00  pobreza_patrim_10  gini_90  gini_00   gini_10
0               33.7          41.900398    0.473    0.425  0.422628
1               48.9          59.175800    0.379    0.533  0.343879
2               57.9          56.504902    0.414    0.465  0.386781
3               40.1          51.164501    0.392    0.541  0.344984
4               42.2          45.703899    0.391    0.469  0.458083

[5 rows x 139 columns]
```

5.- What follows is to explore the dataset, find the null or nan values within the data set since failure to do so may negatively affect the training model. This part of this code shows the columns of the DataFrame 'df' that have null values and the number of null values in each.

```
nan_values = df.isna().sum()
print(nan_values[nan_values > 0])
```

creates a Boolean Data Frame of the same size as 'df', where each value is True if the corresponding value in 'df' is NaN and False if it is not. Then, .sum() is used to sum the number of True values (i.e. null values) in each column, resulting in a Pandas Series containing the number of null values in each column.
later save the values in a variable called "nan_values" and then print those that are greater than 0.



```
cpic_cv                   2
pobtot_00                14
pobtot_05                 2
porc_pob_15_analfa00     14
porc_pob_15_analfa05      2
porc_pob614_noasiste00   14
porc_pob614_noasiste05    2
porc_pob15_basicainc00   14
porc_pob15_basicainc05    2
porc_pob_snservsal00     14
porc_pob_snservsal05      2
porc_vivpisotierra00     14
porc_vivpisotierra05      2
porc_vivsnsan00          14
porc_vivsnsan05           2
porc_snaguaent00         14
```

6.- The next thing is to normalize the dataset; the first thing is to look for the non-numeric values within the dataset and turn them into numerical values.

```
non_numeric_columns = df.select_dtypes(exclude=['number']).columns
df = pd.get_dummies(df, columns=non_numeric_columns)
```

This code transforms non-numeric columns into a DataFrame using one-hot encoding, which is useful for preparing the data for use in our algorithms. Each unique category in the non-numeric columns becomes a new binary column in the resulting DataFrame.

7.- What follows is to fill the null values (NaN) found in the DataFrame with the average value of each corresponding column.

```
df.fillna(df.mean(), inplace=True)
```

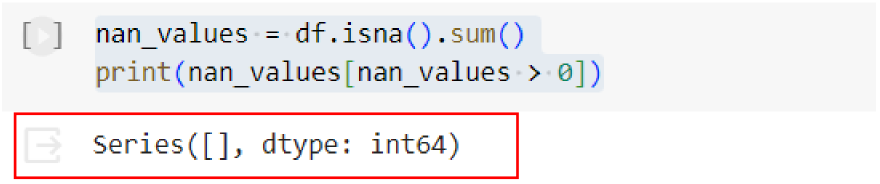
df.fillna(df.mean()): Calculates the mean value of each column in the DataFrame 'df' using the df.mean() function. Then, fill the null values in each column with the corresponding mean value from that column.

inplace=True: Set to True, meaning the operation is performed directly on the DataFrame 'df' instead without assigning the result to a new variable. This modifies 'df' instead.

8.- What follows is just a step to verify our dataset again, so we check again if there are null values in our dataset, for this we use the same function as in step 5

```
nan_values = df.isna().sum()
print(nan_values[nan_values > 0])
```

and we get the following, since our data set no longer contains null data



```
nan_values = df.isna().sum()
print(nan_values[nan_values > 0])
```

```
Series([], dtype: int64)
```

9.- Now we must work on the dataset to keep the important features to work on the model. This is done by calculating the correction of each of the features with respect to the target. First, the N features most correlated with the target are selected. target variable 'poverty' into a DataFrame 'df' and creates a new DataFrame called 'data_selected' containing only those features.

Correlation is a statistical measure that quantifies the relationship between two variables. In the context of data analysis and feature selection, correlation is used to determine the relationship between each feature (column) in a data set and a specific target variable. A high positive correlation value indicates that as a feature increases, the target variable also tends to increase, and a high negative correlation value indicates that as a feature increases, the target variable tends to decrease.

$$= \frac{\sum (X-\bar{X})(Y-\bar{Y})}{\sqrt{\sum (X-\bar{X})^2 \sum (Y-\bar{Y})^2}}$$

Example:

Suppose we have a data set that records the number of hours of study and the scores of a group of students on an exam. We want to understand the relationship between the number of hours students' study and their test scores.

Variable

X: Study hours.
Variable

Y: Grade on the exam.

Student	Study Hours (X)	Exam Score (Y)
1	2	85
2	3	88
3	1	72

We calculate the means of x and y
We calculate the sum of the products of the differences between each value and its mean.
We calculate the sum of the squares of the differences between each value and its mean.
We apply the Pearson correlation formula.

We obtain -0.17 of correlation.

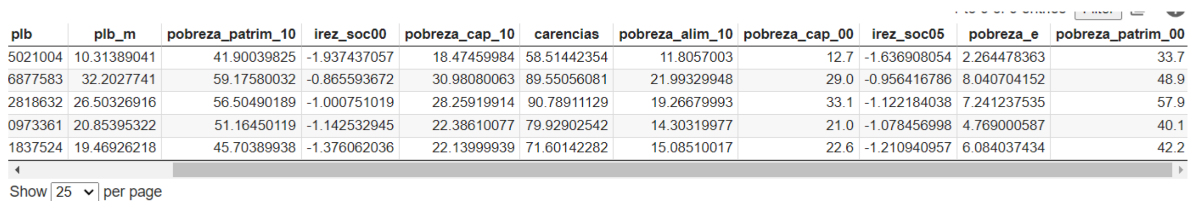
```
target = 'pobreza'
correlation_with_target = df.corr()[target]
correlation_with_target =
correlation_with_target.sort_values(ascending=False)
N = 12
selected_features = correlation_with_target.head(N).index.tolist()

data_selected = df[selected_features]
```

This part allows selecting the characteristics most correlated with a target variable ('poverty' in this case) in a DataFrame 'df'. These features are considered the most important in terms of their relationship to the target variable and are stored in a new DataFrame called 'data_selected'.

This is how our features were obtained and therefore our new data set with our characteristics of interest and our target.

```
print(data_selected.head())
```



plb	plb_m	pobreza_patrim_10	irez_soc00	pobreza_cap_10	carencias	pobreza_alim_10	pobreza_cap_00	irez_soc05	pobreza_e	pobreza_patrim_00
5021004	10.31389041	41.90039825	-1.937437057	18.47459984	58.51442354	11.8057003	12.7	-1.636908054	2.264478363	33.7
6877583	32.2027741	59.17580032	-0.865593672	30.98080063	89.55056081	21.99329948	29.0	-0.956416786	8.040704152	48.9
2818632	26.50326916	56.50490189	-1.000751019	28.25919914	90.78911129	19.26679993	33.1	-1.122184038	7.241237535	57.9
0973361	20.85395322	51.16450119	-1.142532945	22.38610077	79.92902542	14.30319977	21.0	-1.078456998	4.769000587	40.1
1837524	19.46926218	45.70389938	-1.376062036	22.13999939	71.60142282	15.08510017	22.6	-1.210940957	6.084037434	42.2

Show 25 per page

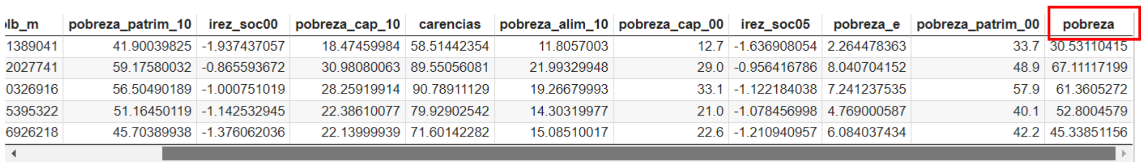
This means that our features were selected by the correlation calculation method, selecting the 12 with the highest correlation.

10.- Now it is important to organize our dataset by putting our target at the end of the dataset. This is done through this code.

```
data_selected = data_selected[[col for col in data_selected.columns if
col != 'pobreza'] + ['pobreza']]

X = data_selected.iloc[:, :-1]
y = data_selected.iloc[:, -1]
```

The code creates a new version of data_selected where it rearranges the columns. First, select all columns in data_selected other than 'poverty' using a list comprehension. Then, add the 'poverty' column at the end. This is done to ensure that the target variable 'poverty' is in the last column of the DataFrame.



ılb_m	pobreza_patrim_10	irez_soc00	pobreza_cap_10	carencias	pobreza_alim_10	pobreza_cap_00	irez_soc05	pobreza_e	pobreza_patrim_00	pobreza
1389041	41.90039825	-1.937437057	18.47459984	58.51442354	11.8057003	12.7	-1.636908054	2.264478363	33.7	30.53110415
2027741	59.17580032	-0.865593672	30.98080063	89.55056081	21.99329948	29.0	-0.956416786	8.040704152	48.9	67.11117199
0326916	56.50490189	-1.000751019	28.25919914	90.78911129	19.26679993	33.1	-1.122184038	7.241237535	57.9	61.3605272
5395322	51.16450119	-1.142532945	22.38610077	79.92902542	14.30319977	21.0	-1.078456998	4.769000587	40.1	52.8004579
6926218	45.70389938	-1.376062036	22.13999939	71.60142282	15.08510017	22.6	-1.210940957	6.084037434	42.2	45.33851156

11.- Now what remains is to divide the already worked dataset into small testing and training datasets, this was done through the SKALEARN library.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_selected, y,
test_size=0.2, random_state=42)
```

Build the model from scratch

1.- A simple linear regression model is implemented, a common choice in statistics and machine learning for regression problems. Linear regression is valued for its simplicity and ability to interpret, since it models the relationship between the independent and dependent variables as a straight line, facilitating the interpretation of the coefficients. Furthermore, it is computationally efficient, being suitable for moderate to large data sets. It is especially applied when a linear relationship between the variables is assumed, being widely known and easy to use in various fields. It is also used as a starting point, providing meaningful and clear results, and if necessary, more complex models can be explored when the relationship is non-linear.

```
import numpy as np
X_train_with_intercept = np.c_[X_train, np.ones(X_train.shape[0])]
X_test_with_intercept = np.c_[X_test, np.ones(X_test.shape[0])]
```

X_train_with_intercept = np.c_[X_train, np.ones(X_train.shape[0])] and the independent term) to the training (X_train) and test (X_test) data matrices. This is necessary to include the independent term in the linear regression model.

These new matrices have an additional column composed solely of ones, known as the "independent term" or "intercept" in the context of a linear regression model.

```
coefficients = np.linalg.lstsq(X_train_with_intercept, y_train,
rcond=None)[0]

m, b = coefficients[0], coefficients[1]
```

coefficients = np.linalg.lstsq(X_train_with_intercept, y_train, rcond=None)[0]: Use the least squares method to find the coefficients of the linear regression model. coefficients contain the values of the slope (m) and the intercept (b) of the regression line.

```
y_pred = X_test_with_intercept.dot(coefficients)
mse = np.mean((y_pred - y_test) ** 2)
print(f'Mean Squared Error: {mse}')
```

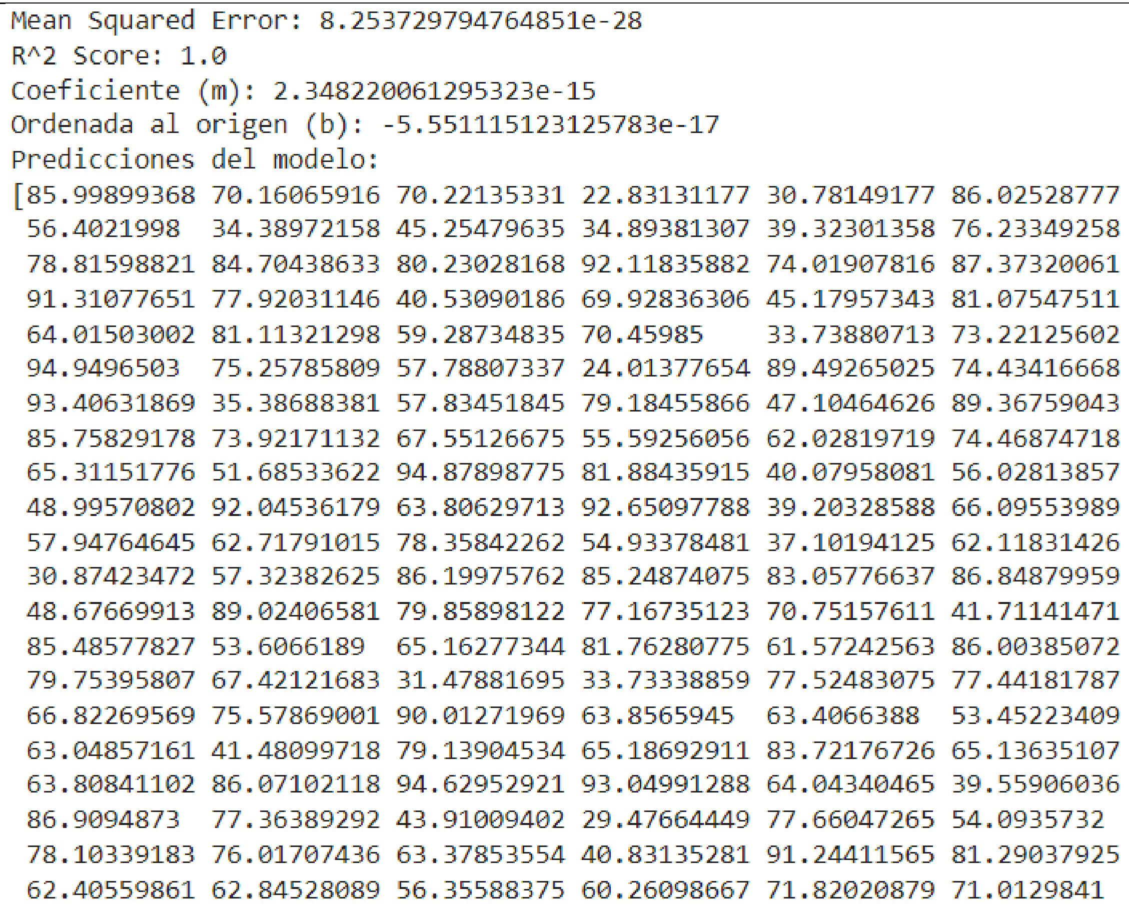
y_pred = X_test_with_intercept.dot(coefficients): Utiliza los coeficientes del modelo para realizar predicciones en el conjunto de datos de prueba X_test_with_intercept. Las predicciones se almacenan en y_pred. mse = np.mean((y_pred - y_test) ** 2): Calcula el Error Cuadrático Medio (MSE), que mide la discrepancia promedio entre las predicciones (y_pred) y los valores reales (y_test). Un MSE más bajo indica un mejor ajuste del modelo.

```
ssr = np.sum((y_pred - y_test) ** 2)
sst = np.sum((y_test - np.mean(y_test)) ** 2)
r2 = 1 - (ssr / sst)
print(f'R^2 Score: {r2}')
```

ssr = np.sum((y_pred - y_test) ** 2): Calculates the sum of the squares of the residuals (errors) between the predictions and the actual values. This measures variability not explained by the model.
sst = np.sum((y_test - np.mean(y_test)) ** 2): Calculates the total sum of squares, which measures the total variability in the actual values.
r2 = 1 - (ssr / sst): Calculates the coefficient of determination (R^2), which provides a measure of how much variability in the target variable is explained by the regression model. An R^2 close to 1 indicates a good model fit.

```
print(f'Coeficiente (m): {m}')
print(f'Ordenada al origen (b): {b}')

print("Predicciones del modelo:")
print(y_pred)
```



```
Mean Squared Error: 8.253729794764851e-28
R^2 Score: 1.0
Coeficiente (m): 2.348220061295323e-15
Ordenada al origen (b): -5.551115123125783e-17
Predicciones del modelo:
[85.99899368 70.16065916 70.22135331 22.83131177 30.78149177 86.02528777
 56.4021998  34.38972158 45.25479635 34.89381307 39.32301358 76.23349258
 78.81598821 84.70438633 80.23028168 92.11835882 74.01907816 87.37320061
 91.31077651 77.92031146 40.53090186 69.92836306 45.17957343 81.07547511
 64.01503002 81.11321298 59.28734835 70.45985    33.73880713 73.22125602
 94.9496503  75.25785809 57.78807337 24.01377654 89.49265025 74.43416668
 93.40631869 35.38688381 57.83451845 79.18455866 47.10464626 89.36759043
 85.75829178 73.92171132 67.55126675 55.59256056 62.02819719 74.46874718
 65.31151776 51.68533622 94.87898775 81.88435915 40.07958081 56.02813857
 48.99570802 92.04536179 63.80629713 92.65097788 39.20328588 66.09553989
 57.94764645 62.71791015 78.35842262 54.93378481 37.10194125 62.11831426
 30.87423472 57.32382625 86.19975762 85.24874075 83.05776637 86.84879959
 48.67669913 89.02406581 79.85898122 77.16735123 70.75157611 41.71141471
 85.48577827 53.6066189  65.16277344 81.76280775 61.57242563 86.00385072
 79.75395807 67.42121683 31.47881695 33.73338859 77.52483075 77.44181787
 66.82269569 75.57869001 90.01271969 63.8565945  63.4066388  53.45223409
 63.04857161 41.48099718 79.13904534 65.18692911 83.72176726 65.13635107
 63.80841102 86.07102118 94.62952921 93.04991288 64.04340465 39.55906036
 86.9094873  77.36389292 43.91009402 29.47664449 77.66047265 54.0935732
 78.10339183 76.01707436 63.37853554 40.83135281 91.24411565 81.29037925
 62.40559861 62.84528089 56.35588375 60.26098667 71.82020879 71.0129841
```

Analyzing the data it can be seen that the model is performing well, comparing it with the original dataset and the target it is observed that the predictions are within what was expected, with almost perfect values or not so far from the real values.

construction of the model with functions

```
pip install tensorflow
```

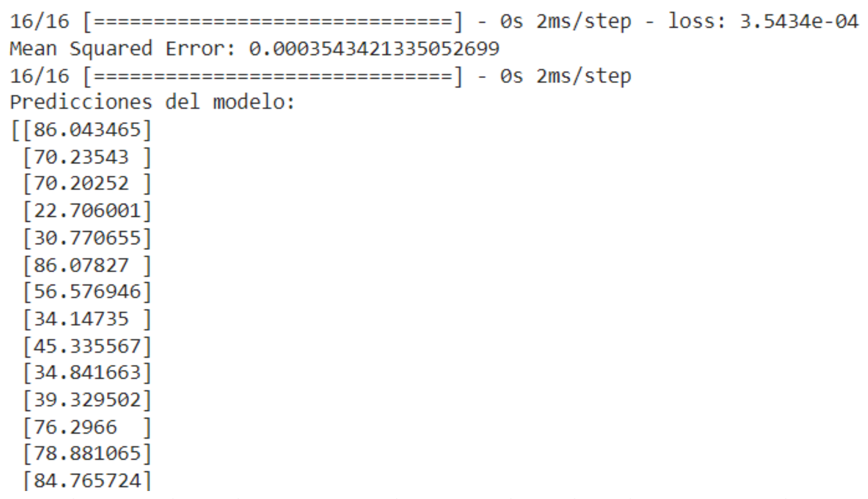
```
import numpy as np
import tensorflow as tf

X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)

model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(units=1,
input_shape=(X_train.shape[1],)))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, epochs=1000, verbose=0)
mse = model.evaluate(X_test, y_test)
print("Mean Squared Error:", mse)
y_predictions = model.predict(X_test)

print("Predicciones del modelo:")
print(y_pred)
```

we have used TensorFlow, a machine learning library, to build a linear regression model. This model aims to predict numerical values from data. First, we prepare our training and test data, and then create a simple neural network model. The network consists of a single layer (a dense layer) with a unit (neuron) and input dimensions that match the number of features in our data. We then configure the model to minimize the loss, which in this case is the mean squared error (MSE), using an optimizer called 'adam', a common and effective choice. We then train the model for 1000 "epochs" to adjust its weights and make accurate predictions. Finally, we evaluate the performance of the model on our test data and calculate the MSE to measure how well the predictions fit the actual values. The model predictions are stored in 'y_predictions'. This regression model is a starting point in machine learning and can be useful for predicting numerical values from data, which is essential in various applications, from predicting the price of a house to understanding trends in business data. The choice of this model is due to its simplicity and accessibility, its general applicability in a variety of situations, and the TensorFlow environment that offers powerful tools. Furthermore, it is a good starting point that allows us to compare its performance with more complex models if necessary.



```
16/16 [==============================] - 0s 2ms/step - loss: 3.5434e-04
Mean Squared Error: 0.00035434213350526999
16/16 [==============================] - 0s 2ms/step
Predicciones del modelo:
[[86.043465]
 [70.23543 ]
 [70.20252 ]
 [22.706001]
 [30.770655]
 [86.07827 ]
 [56.576946]
 [34.14735 ]
 [45.335567]
 [34.841663]
 [39.329502]
 [76.2966  ]
 [78.881065]
 [84.765724]
```

The neural network implemented with TensorFlow also obtained very low MSE of around 3.54e-04, indicating a good fit of the model to the data. As in the first model, the R^2 is very close to 1.0, which suggests a high predictive capacity. The predictions of this model are virtually identical to those of the first model, further supporting the quality of the fit.

In summary, both models achieved an exceptionally good fit to the data, with MSE and R^2 very close to ideal values.

conclusion

"For me, the most challenging aspect of this project was managing the data set to prepare it for model implementation. I initially faced a bit of misinformation about various data manipulation tasks such as data normalization, handling missing values and feature selection methods. This experience highlighted the importance of data preprocessing in the machine learning workflow. I realized that understanding how to prepare and clean data is a crucial step in building accurate and reliable models. While presented some initial difficulties, it also served as a valuable learning opportunity, helping me gain confidence in these essential data preparation techniques." In the application areas of robotics it can be done for the prediction of trajectories
GitHub link
https://github.com/soypatty/Regresion-model-.git