

## **LABORATORIO N°5**

### **CARS PRICE PREDICTION**

Andrés Felipe Esquivel Ruiz; Código 12967.

Nathaly Daniela Mejía Meléndez; Código 84588.

Universidad ECCI

Seminario Big Data

Elías Buitrago Bolívar

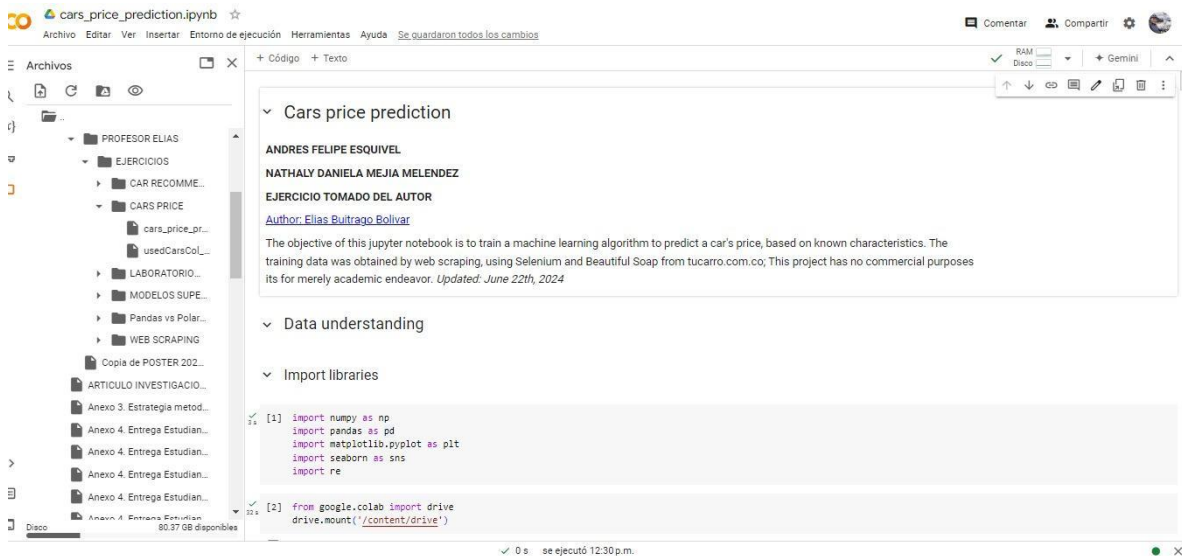
Julio de 2024

## Descripción de la Actividad:

Como objetivo de la actividad esta el poder comprender como entrenar un algoritmo de machine learning mediante el cual vamos a poder predecir el precio de un carro de acuerdo a los datos obtenidos previamente en la actividad de WebScrapping, la marca de vehículos seleccionada para la extracción de datos es Renault en su Línea Logan.

## Desarrollo de la Actividad:

Como Primer paso realizamos la importación de librerías utilizadas en el cargue y procesamiento de los datos, así como realizar el enlace a Google Drive para el cargue de los datos:



```
cars_price_prediction.ipynb
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se guardaron todos los cambios

+ Código + Texto

Cars price prediction
ANDRES FELIPE ESQUIVEL
NATHALY DANIELA MEJIA MELENDEZ
EJERCICIO TOMADO DEL AUTOR
Author: Elias Buitrago Bolivar

The objective of this jupyter notebook is to train a machine learning algorithm to predict a car's price, based on known characteristics. The training data was obtained by web scrapping, using Selenium and BeautifulSoup from tucarro.com.co; This project has no commercial purposes its for merely academic endeavor. Updated: June 22th, 2024

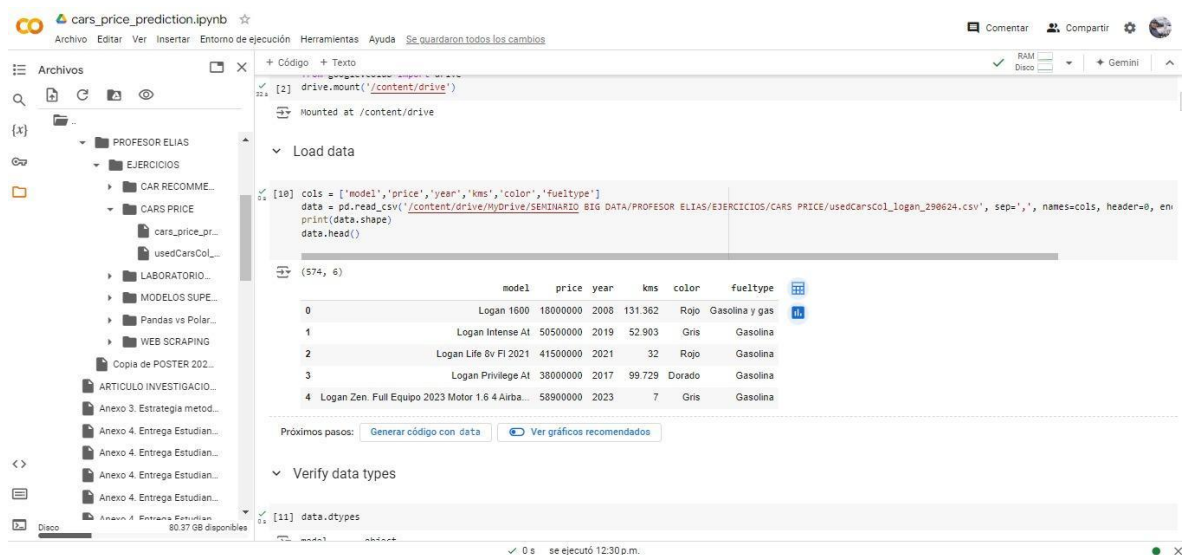
Data understanding

Import libraries

[1] import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re

[2] from google.colab import drive
drive.mount('/content/drive')
```

Posteriormente realizamos el cargue de la Data mediante la ruta seleccionada desde Google drive  
/content/drive/MyDrive/SEMINARIO BIG DATA/PROFESOR ELIAS/EJERCICIOS/CARS PRICE/usedCarsCol\_logan\_290624.csv



```
cars_price_prediction.ipynb
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se guardaron todos los cambios

+ Código + Texto

[2] drive.mount('/content/drive')
Mounted at /content/drive

Load data

[10] cols = ['model', 'price', 'year', 'kms', 'color', 'fueltype']
data = pd.read_csv('/content/drive/MyDrive/SEMINARIO BIG DATA/PROFESOR ELIAS/EJERCICIOS/CARS PRICE/usedCarsCol_logan_290624.csv', sep=',', names=cols, header=0, en
print(data.shape)
data.head()

(574, 6)

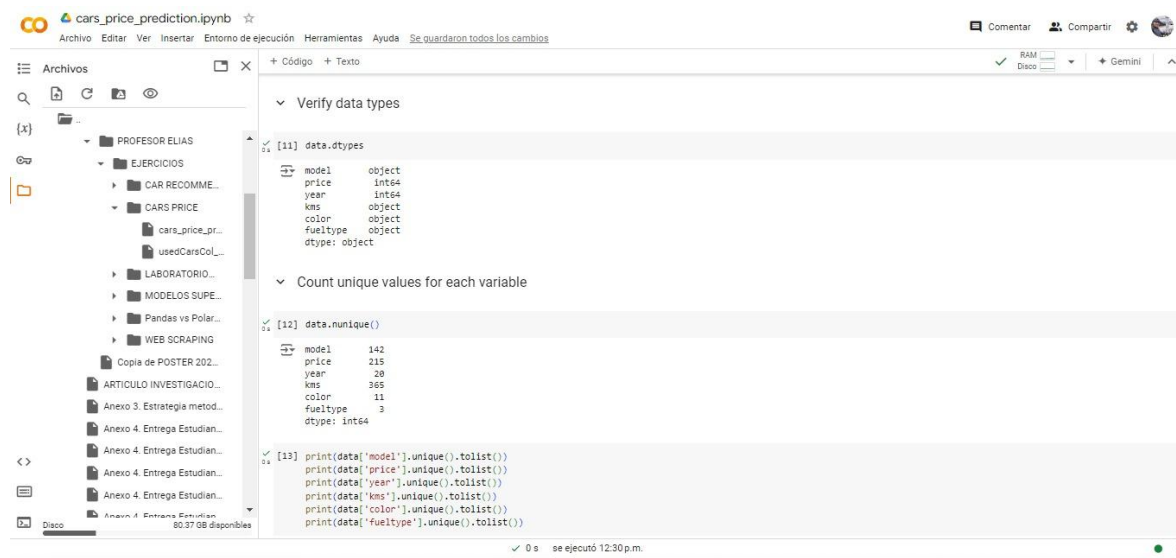
Proximos pasos: Generar código con data Ver gráficos recomendados

Verify data types

[11] data.dtypes
```

	model	price	year	kms	color	fueltype
0	Logan 1600	18000000	2008	131.362	Rojo	Gasolina y gas
1	Logan Intense At	50500000	2019	52.903	Gris	Gasolina
2	Logan Life 8v FI 2021	41500000	2021	32	Rojo	Gasolina
3	Logan Privilege At	38000000	2017	99.729	Dorado	Gasolina
4	Logan Zen Full Equipo 2023 Motor 1.6 4 Airba...	58900000	2023	7	Gris	Gasolina

## Realizamos Verificación de la Data



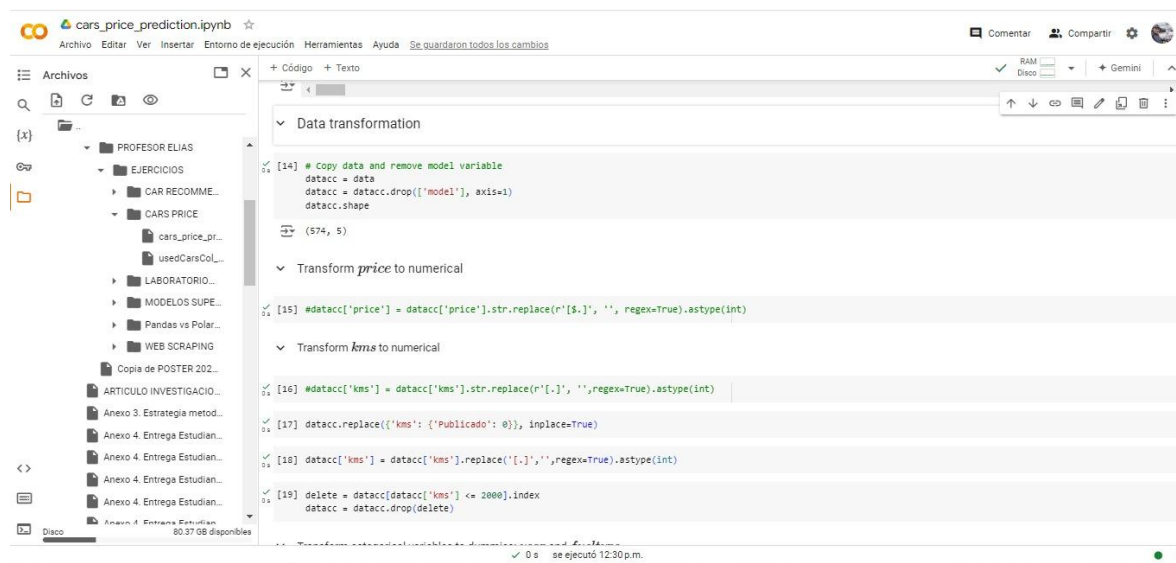
The screenshot shows a Jupyter Notebook titled 'cars\_price\_prediction.ipynb'. The left sidebar displays a file explorer with a directory structure including 'PROFESOR ELIAS', 'EJERCICIOS', 'CAR RECOMME...', 'CARS PRICE', and various 'Anexo' files. The main area contains two code cells. The first cell, titled 'Verify data types', shows the output of `data.dtypes`, which lists the data types for each variable: `model` (object), `price` (int64), `year` (int64), `kms` (object), `color` (object), `fueltype` (object), and `dtype` (object). The second cell, titled 'Count unique values for each variable', shows the output of `data.nunique()`, which lists the number of unique values for each variable: `model` (142), `price` (215), `year` (20), `kms` (365), `color` (11), `fueltype` (3), and `dtype` (int64). Below this, there is a print statement that lists the unique values for each variable.

```
[11] data.dtypes
model    object
price    int64
year     int64
kms      object
color    object
fueltype object
dtype: object

[12] data.nunique()
model     142
price     215
year       20
kms       365
color      11
fueltype   3
dtype: int64

[13] print(data['model'].unique().tolist())
print(data['price'].unique().tolist())
print(data['year'].unique().tolist())
print(data['kms'].unique().tolist())
print(data['color'].unique().tolist())
print(data['fueltype'].unique().tolist())
```

Posteriormente realizamos la transformación de los datos a formato numérico realizando la simplificación eliminando separadores como puntos y comas con el fin de poder dejar un dato numérico concreto.



The screenshot shows a Jupyter Notebook titled 'cars\_price\_prediction.ipynb'. The left sidebar displays a file explorer with a directory structure including 'PROFESOR ELIAS', 'EJERCICIOS', 'CAR RECOMME...', 'CARS PRICE', and various 'Anexo' files. The main area contains three code cells. The first cell, titled 'Data transformation', shows the output of `dataacc = data.drop('model', axis=1)` and `dataacc.shape`, which is (574, 5). The second cell, titled 'Transform price to numerical', shows the output of `dataacc['price'] = dataacc['price'].str.replace(r'[.], ', '', regex=True).astype(int)`. The third cell, titled 'Transform kms to numerical', shows the output of `dataacc['kms'] = dataacc['kms'].str.replace(r'[.], ', '', regex=True).astype(int)`, `dataacc.replace({'kms': {'Publicado': 0}}, inplace=True)`, `dataacc['kms'] = dataacc['kms'].replace(['.'], '', regex=True).astype(int)`, and `delete = dataacc[dataacc['kms'] <= 2000].index` and `dataacc = dataacc.drop(delete)`.

```
[14] # copy data and remove model variable
dataacc = data.drop('model', axis=1)
dataacc.shape
(574, 5)

[15] #dataacc['price'] = dataacc['price'].str.replace(r'[.], ', '', regex=True).astype(int)

[16] #dataacc['kms'] = dataacc['kms'].str.replace(r'[.], ', '', regex=True).astype(int)

[17] dataacc.replace({'kms': {'Publicado': 0}}, inplace=True)

[18] dataacc['kms'] = dataacc['kms'].replace(['.'], '', regex=True).astype(int)

[19] delete = dataacc[dataacc['kms'] <= 2000].index
dataacc = dataacc.drop(delete)
```

Luego realizamos la transformación de variables categóricas en variables ficticias año y tipo de combustible

cars\_price\_prediction.ipynb

Transform categorical variables to dummies: *year* and *fueltype*

```
[20] # object to categorical
datacc['color'] = datacc['color'].astype('category')
datacc['fueltype'] = datacc['fueltype'].astype('category')
datacc.dtypes
```

```
[21] # Convert to dummies
# Convert category to codes
datacc['color'] = pd.Categorical(datacc['color']).codes
datacc['fueltype'] = pd.Categorical(datacc['fueltype']).codes
datacc.head()
```

	price	year	kms	color	fueltype
0	18000000	2008	131362	7	1
1	50500000	2019	52903	4	0
3	38000000	2017	99729	3	0
8	35000000	2016	78066	7	0
10	55900000	2023	2705	1	0

Próximos pasos: Generar código con datacc Ver gráficos recomendados

0 s se ejecutó 12:30 p.m.

cars\_price\_prediction.ipynb

Data Exploration

```
[22] # Descriptive statistics
datacc.describe()[['price', 'year', 'kms']]
```

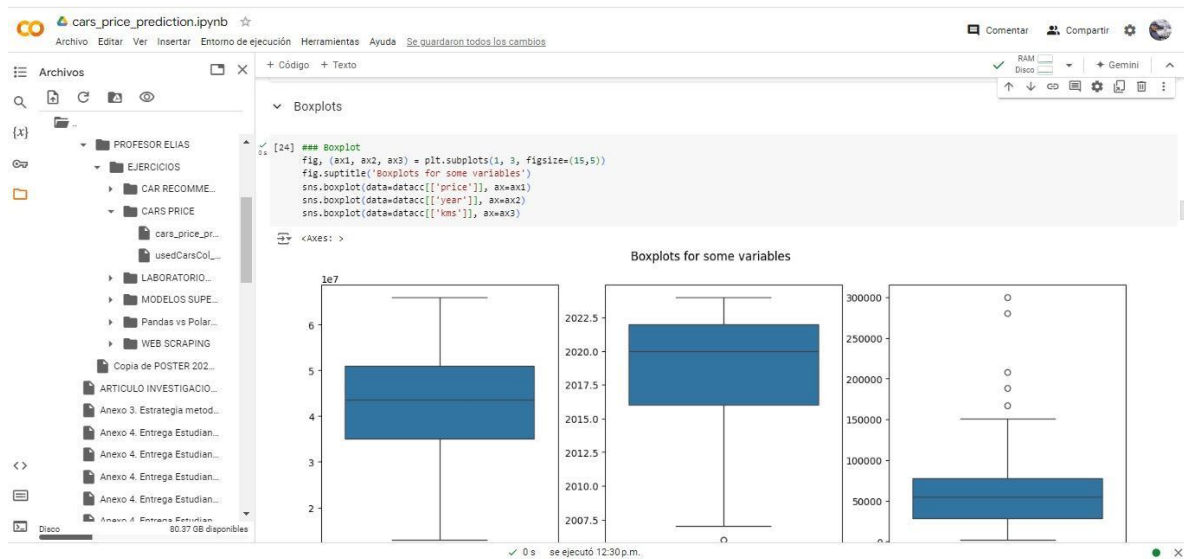
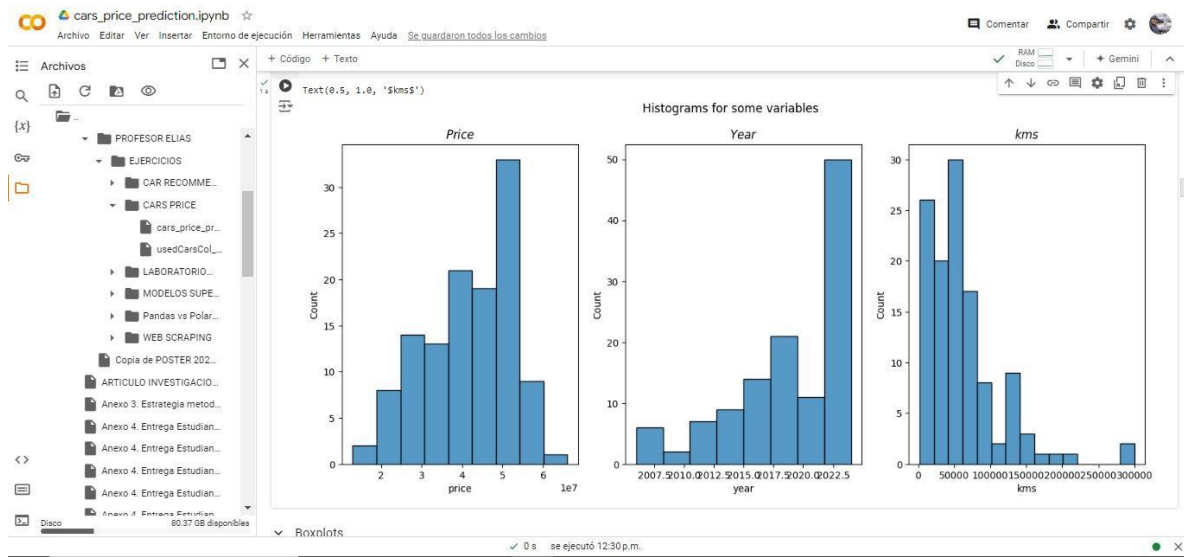
	price	year	kms
count	1.200000e+02	120.000000	120.000000
mean	4.165471e+07	2018.541667	61730.266667
std	1.109279e+07	4.505638	52249.461488
min	1.300000e+07	2006.000000	2264.000000
25%	3.500000e+07	2016.000000	28059.000000
50%	4.350000e+07	2020.000000	54939.000000
75%	5.100000e+07	2022.000000	77947.500000
max	6.599000e+07	2024.000000	300072.000000

Histograms

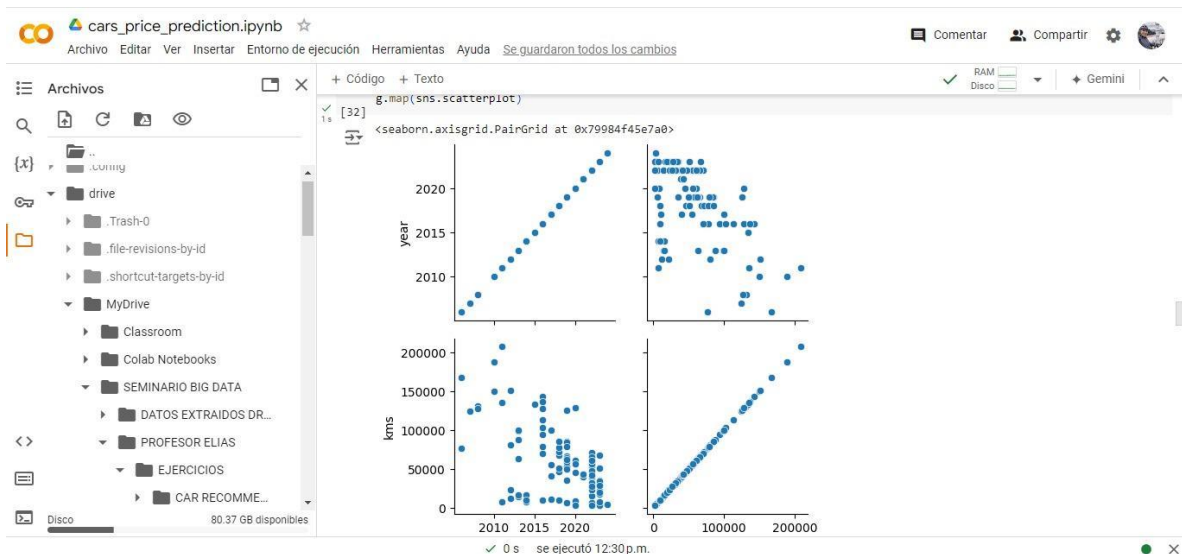
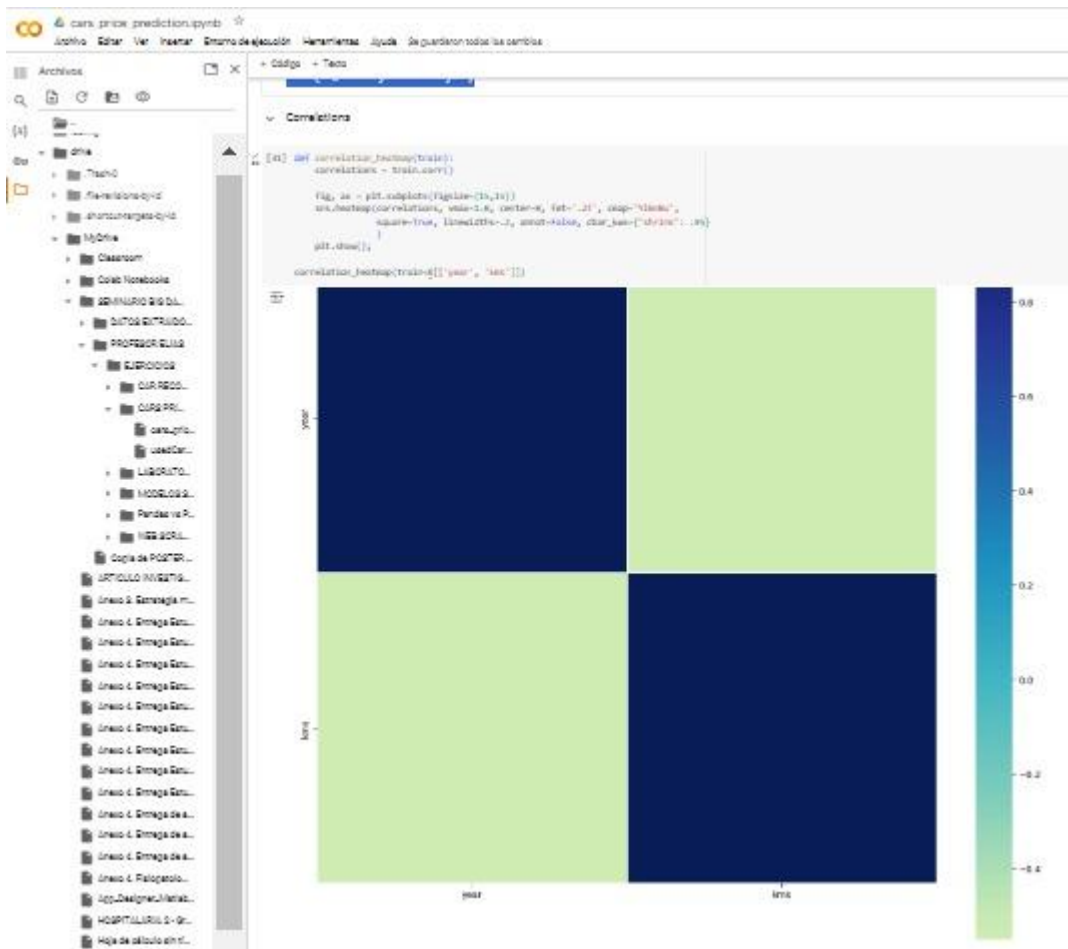
```
[23] # Histograms
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15,6))
fig.suptitle('Histograms for some variables')
sns.histplot(datacc['price'], ax=ax1)
ax1.set_title('$prices')
sns.histplot(datacc['year'], ax=ax2)
ax2.set_title('$years')
```

0 s se ejecutó 12:30 p.m.

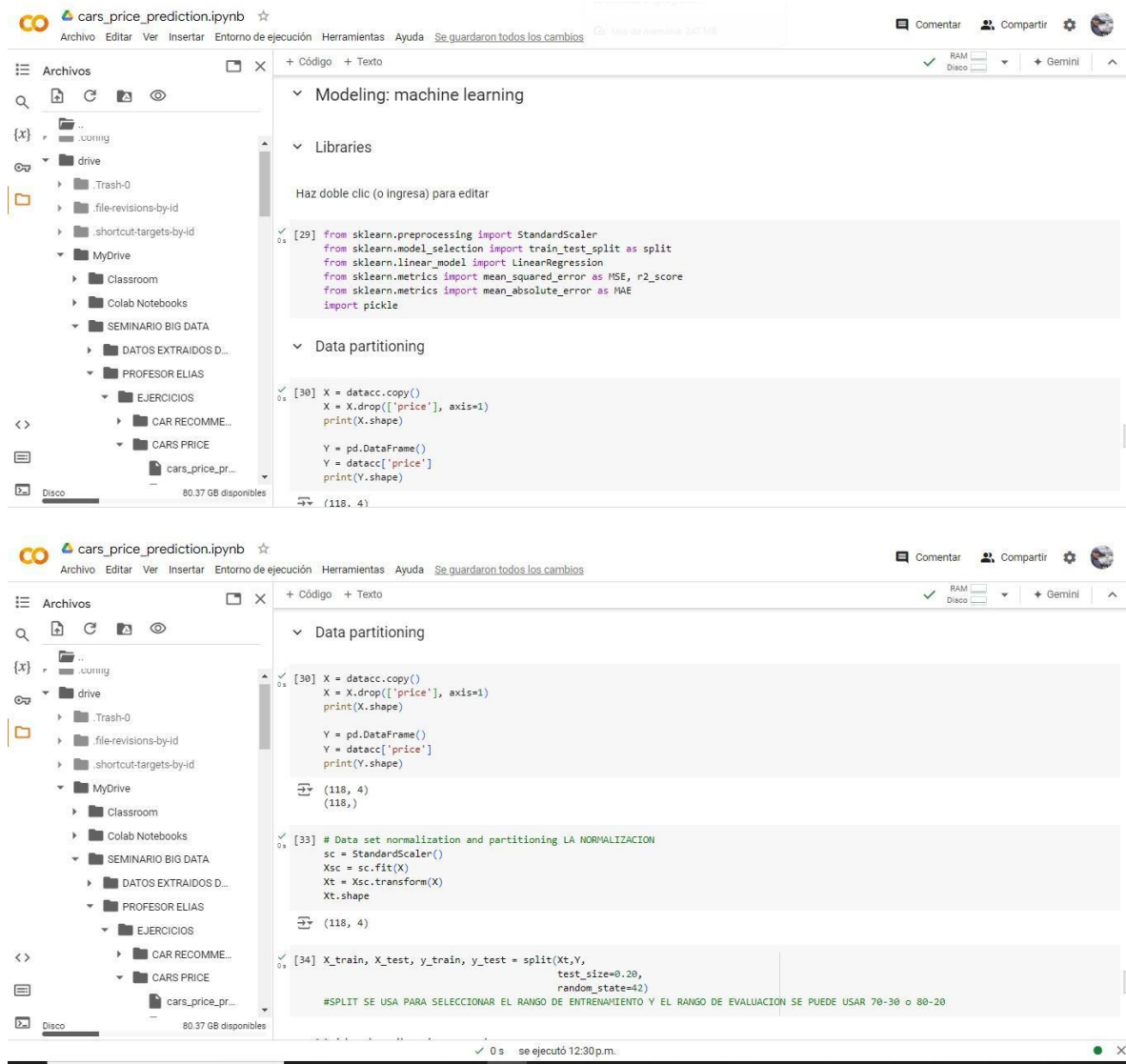
A continuación, realizamos la visualización de los datos organizados mediante los histogramas y diagramas de bloques:



También se visualizan los diagramas de correlación



Luego de tener los datos organizados y poder visualizarlos mediante las herramientas de graficas mencionadas anteriormente procedemos a llamar las librerías Necesarias para entrenar el modelo y realizamos la partición de los datos con el fin de Normalizar los datos.



The image displays two screenshots of a Jupyter Notebook interface, showing the process of preparing data for a machine learning model.

**Top Screenshot:** The notebook is titled "cars\_price\_prediction.ipynb". The left sidebar shows a file explorer with a tree view of the file system. The main area is divided into two sections: "Modeling: machine learning" and "Libraries". The "Libraries" section contains the following code:

```
[29] from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split as split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error as MSE, r2_score
from sklearn.metrics import mean_absolute_error as MAE
import pickle
```

The "Data partitioning" section contains the following code:

```
[30] X = dataacc.copy()
X = X.drop(['price'], axis=1)
print(X.shape)

Y = pd.DataFrame()
Y = dataacc['price']
print(Y.shape)
```

The output of the code shows the shape of the data: (118, 4) for X and (118,) for Y.

**Bottom Screenshot:** The notebook is titled "cars\_price\_prediction.ipynb". The left sidebar shows a file explorer with a tree view of the file system. The main area is divided into two sections: "Data partitioning" and "Data set normalization and partitioning LA NORMALIZACION". The "Data partitioning" section contains the following code:

```
[30] X = dataacc.copy()
X = X.drop(['price'], axis=1)
print(X.shape)

Y = pd.DataFrame()
Y = dataacc['price']
print(Y.shape)
```

The output of the code shows the shape of the data: (118, 4) for X and (118,) for Y.

The "Data set normalization and partitioning LA NORMALIZACION" section contains the following code:

```
[33] # Data set normalization and partitioning LA NORMALIZACION
sc = StandardScaler()
Xsc = sc.fit(X)
Xt = Xsc.transform(X)
Xt.shape
```

The output of the code shows the shape of the data: (118, 4).

The "Data partitioning" section contains the following code:

```
[34] X_train, X_test, y_train, y_test = split(Xt, Y,
                                             test_size=0.20,
                                             random_state=42)

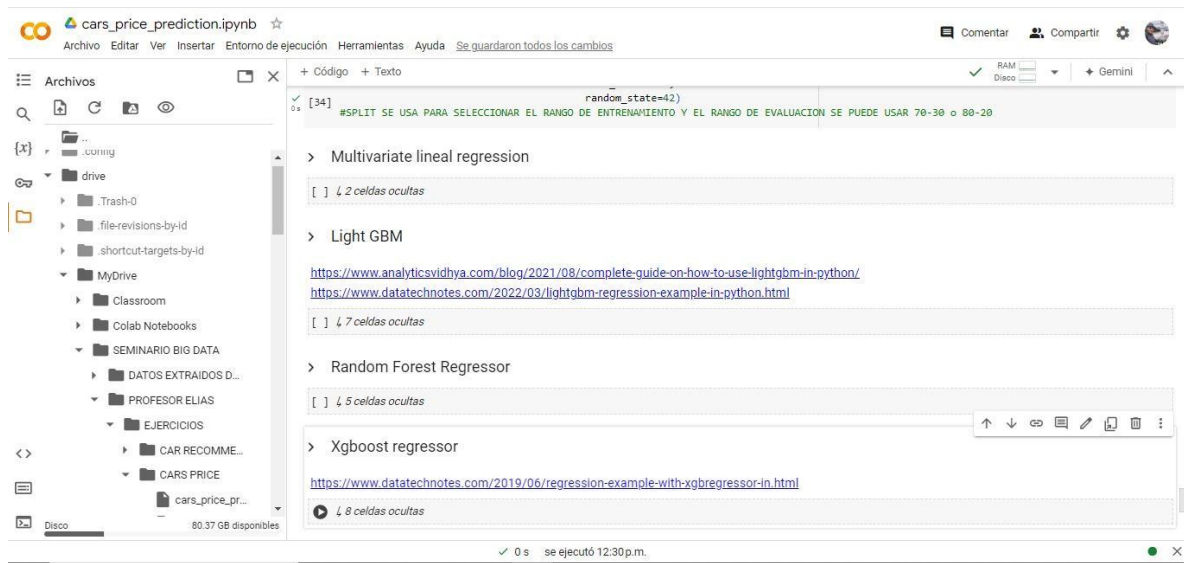
#SPLIT SE USA PARA SELECCIONAR EL RANGO DE ENTRENAMIENTO Y EL RANGO DE EVALUACION SE PUEDE USAR 70-30 o 80-20
```

The output of the code shows the shape of the data: (118, 4).

Ahora vamos a Entrenar 4 modelos Diferentes para la predicción los cuales son:

1. Modelo de Regresión
2. Modelo Ligth GBM
3. Random Forest
4. Xgboost regressor

Este Entrenamiento lo realizamos con una Learning Rate de 0.1 Inicialmente Obteniendo los siguientes resultados:



## 1. Modelo de Regresión



## 2. Modelo Ligth GBM





### 3. Random Forest

```
Random Forest Regressor

[44] from sklearn.ensemble import RandomForestRegressor

[45] model3 = RandomForestRegressor()
model3.fit(X_train, y_train)
y_pred3 = model3.predict(X_test)

[46] # accuracy check
rmse = MSE(y_test, y_pred3, squared=False)
mae = MAE(y_test, y_pred3)
r2 = r2_score(y_test, y_pred3)
print("RMSE: %.2f" % rmse)
print("MAE: %.2f" % mae)
print("R2: %.2f" % r2)

RMSE: 6539990.09
MAE: 4560037.50
R2: 0.80

Save the model

[47] with open('model3.pkl', 'wb') as f:
    pickle.dump(model3, f)
```

### 4. Xgboost regressor

```
[54] # accuracy check
rmse = MSE(y_test, y_pred4, squared=False)
mae = MAE(y_test, y_pred4)
r2 = r2_score(y_test, y_pred4)
print("RMSE: %.2f" % rmse)
print("MAE: %.2f" % mae)
print("R2: %.2f" % r2)

RMSE: 6364375.46
MAE: 4624120.50
R2: 0.81
```

Luego de Evaluar los modelos obtuvimos la siguiente Información:

MODELO	RMSE	R2	MEJOR RESULTADO
Modelo de Regresion	\$ 6.137.147,00	0.82	SI
Modelo Ligth GBM	\$ 7.894.339,00	0.71	NO
Modelo Random Forest	\$ 6.539.990,00	0.80	NO
Modelo Xgboost regressor	\$ 6.364.375,00	0.81	NO

Luego de lo Anterior decidimos variar la Learning Rate a 0.05 en 2 de los modelos los cuales Fueron **Modelo Ligth GBM y Xgboost regressor** Con el fin de evaluar los resultados después del cambio, obteniendo los siguientes resultados:

## 1. Modelo Ligth GBM (Con tasa de aprendizaje 0.05)

```
[LightGBM] [warning] No further splits with positive gain, best gain: -inf
```

```
[40] # accuracy check
rmse = MSE(y_test, y_pred2, squared=False)
mae = MAE(y_test, y_pred2)
r2 = r2_score(y_test, y_pred2)
print("RMSE: %.2f" % rmse)
print("MAE: %.2f" % mae)
print("R2: %.2f" % r2)
```

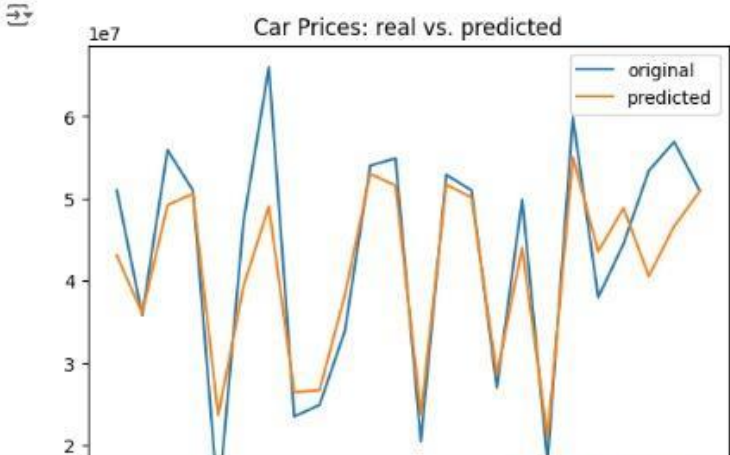
```
RMSE: 7983772.69
MAE: 6500527.29
R2: 0.70
```

## 4. Xgboost regressor (Con tasa de aprendizaje 0.05)

```
[61] # accuracy check
rmse = MSE(y_test, y_pred4, squared=False)
mae = MAE(y_test, y_pred4)
r2 = r2_score(y_test, y_pred4)
print("RMSE: %.2f" % rmse)
print("MAE: %.2f" % mae)
print("R2: %.2f" % r2)
```

```
RMSE: 6497684.12
MAE: 4908124.83
R2: 0.80
```

```
x_ax = range(len(y_test))
plt.plot(x_ax, y_test, label="original")
plt.plot(x_ax, y_pred4, label="predicted")
plt.title("Car Prices: real vs. predicted")
plt.legend()
plt.show()
```



0 s se ejecutó 12:43 p.m.

Obteniendo los siguientes resultados conforme a la anterior Iteración:

MODELO	CON TASA DE (0.1)		CON TASA DE (0.05)	
	RMSE	R2	RMSE	R2
Modelo Ligth GBM	\$ 7.894.339,00	0.71	\$ 7.983.772,00	0.70
Modelo Xgboost regressor	\$ 6.364.375,00	0.81	\$ 6.497.684,00	0.80

De acuerdo a lo anterior encontramos que al variar la Tasa de Aprendizaje y disminuirla a 0.05 la predicción se vuelve mas ineficaz por lo tanto debemos realizar varias Iteraciones aumentando la tasa de Aprendizaje para Obtener mejores resultados teniendo cuidado de no incurrir en un Sobreajuste.

### **CONCLUSION:**

Como conclusión podemos decir que es importante poder realizar iteraciones en los modelos para poder identificar cual de ellos es más eficaz y de mejor ajuste al proyecto, adicionalmente es importante recalcar que los modelos que entrenamos tengan un proceso correctamente ejecutado para que la precisión del modelo mejore y sea más Confiable.