Práctica 2: Minería de Datos. Analizando un archivo CSV mediante el uso de Python.

Práctica realizada por Lorena Almoguera

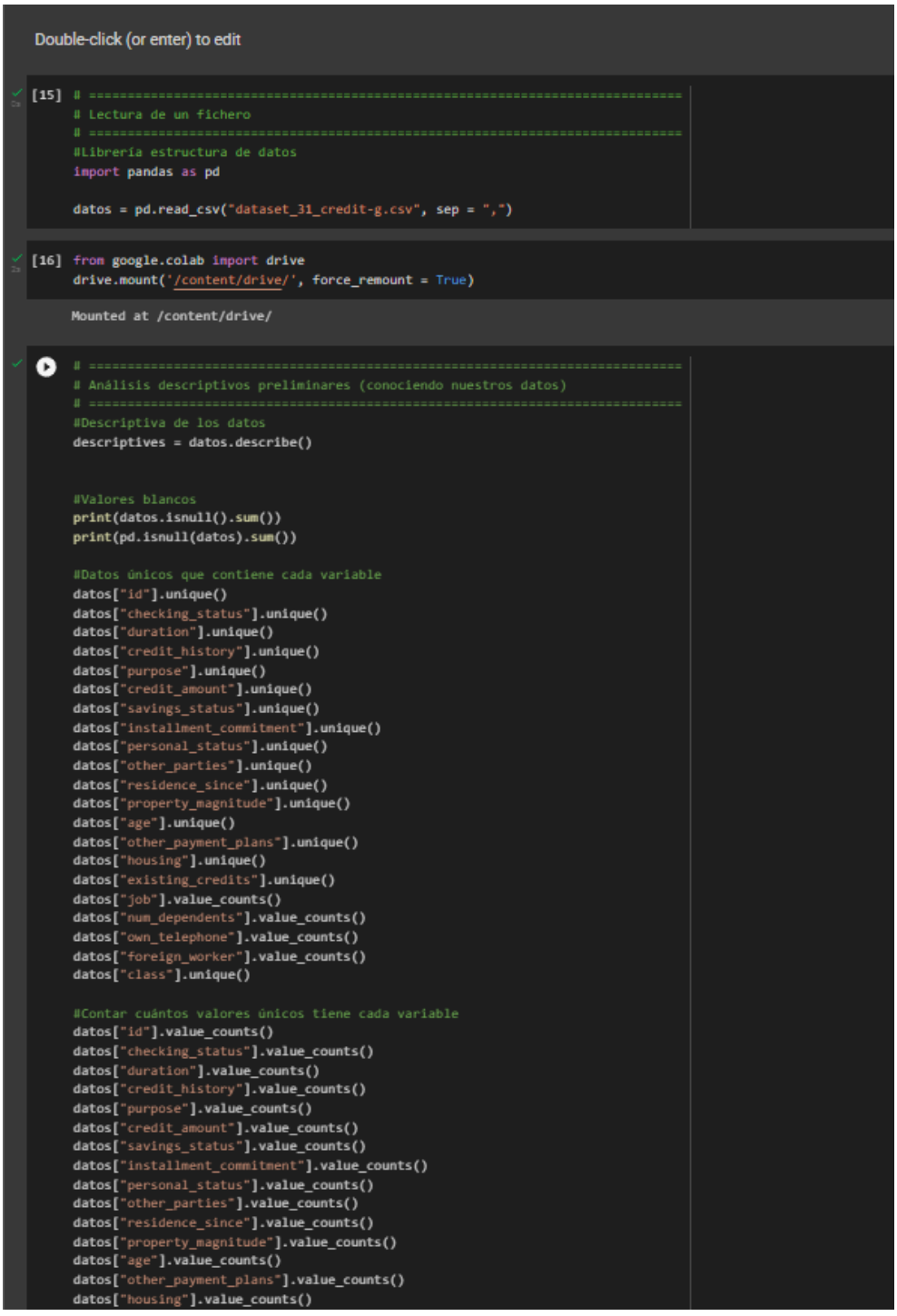
Índice

Introducción

En esta práctica vamos a realizar las mismas tareas que en la práctica anterior. Esta vez, las realizaremos en el entorno de trabajo Google Colaboratory y haremos uso del lenguaje de programación PYTHON. Emplearemos el mismo data set que en la práctica anterior. Recordamos que nuestro dataset estudia el perfil del solicitante de un crédito. Se han recopilado los datos de 1000 individuos en el dataset.

Análisis descriptivos - preliminares

En primer lugar, tras conectar nuestro drive a Google colaboratory, procederemos a cambiar el nombre de las variables, a las que nos indica nuestro documento csv.



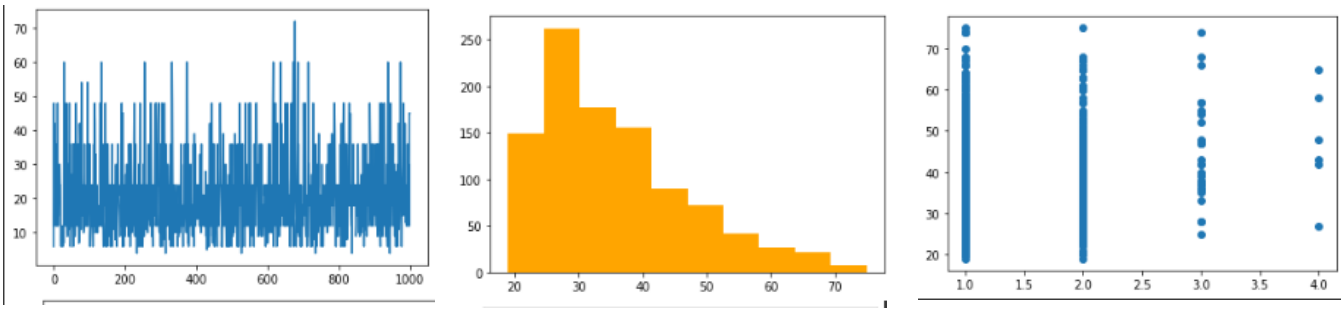
```
Double-click (or enter) to edit

[15] # ======================================================================
     # Lectura de un fichero
     # ======================================================================
     #Librería estructura de datos
     import pandas as pd

     datos = pd.read_csv("dataset_31_credit-g.csv", sep = ",")

[16] from google.colab import drive
     drive.mount('/content/drive/', force_remount = True)

     Mounted at /content/drive/

     # ======================================================================
     # Análisis descriptivos preliminares (conociendo nuestros datos)
     # ======================================================================
     #Descriptiva de los datos
     descriptives = datos.describe()


     #Valores blancos
     print(datos.isnull().sum())
     print(pd.isnull(datos).sum())

     #Datos únicos que contiene cada variable
     datos["id"].unique()
     datos["checking_status"].unique()
     datos["duration"].unique()
     datos["credit_history"].unique()
     datos["purpose"].unique()
     datos["credit_amount"].unique()
     datos["savings_status"].unique()
     datos["installment_commitment"].unique()
     datos["personal_status"].unique()
     datos["other_parties"].unique()
     datos["residence_since"].unique()
     datos["property_magnitude"].unique()
     datos["age"].unique()
     datos["other_payment_plans"].unique()
     datos["housing"].unique()
     datos["existing_credits"].unique()
     datos["job"].value_counts()
     datos["num_dependents"].value_counts()
     datos["own_telephone"].value_counts()
     datos["foreign_worker"].value_counts()
     datos["class"].unique()

     #Contar cuántos valores únicos tiene cada variable
     datos["id"].value_counts()
     datos["checking_status"].value_counts()
     datos["duration"].value_counts()
     datos["credit_history"].value_counts()
     datos["purpose"].value_counts()
     datos["credit_amount"].value_counts()
     datos["savings_status"].value_counts()
     datos["installment_commitment"].value_counts()
     datos["personal_status"].value_counts()
     datos["other_parties"].value_counts()
     datos["residence_since"].value_counts()
     datos["property_magnitude"].value_counts()
     datos["age"].value_counts()
     datos["other_payment_plans"].value_counts()
     datos["housing"].value_counts()
```

Gráficos Descriptivos. Correlaciones.

A continuación, vamos a generar unos gráficos. En esto caso vamos a seleccionar variables de tipo numérico: “duration”, “age” y “existing_credits”.

```
# ========================================================================
# Gráficos descriptivos. Correlaciones
# ========================================================================
import matplotlib.pyplot as plt
# Gráfico de líneas
plt.plot(datos["duration"])
plt.show()
# Histogramas
plt.hist(datos["age"], facecolor = 'orange')
plt.show()
# Gráfico de dispersión
plt.scatter(datos["existing_credits"],datos["age"])
plt.show()
```

Ejecutamos el programa y nos generará lo siguiente:



plt.plot(datos[“duration”]) corresponde con la gráfica de la izquierda. Esta **gráfica de líneas** nos representa la duración de los créditos (eje y) frente a al número de solicitantes (eje x).

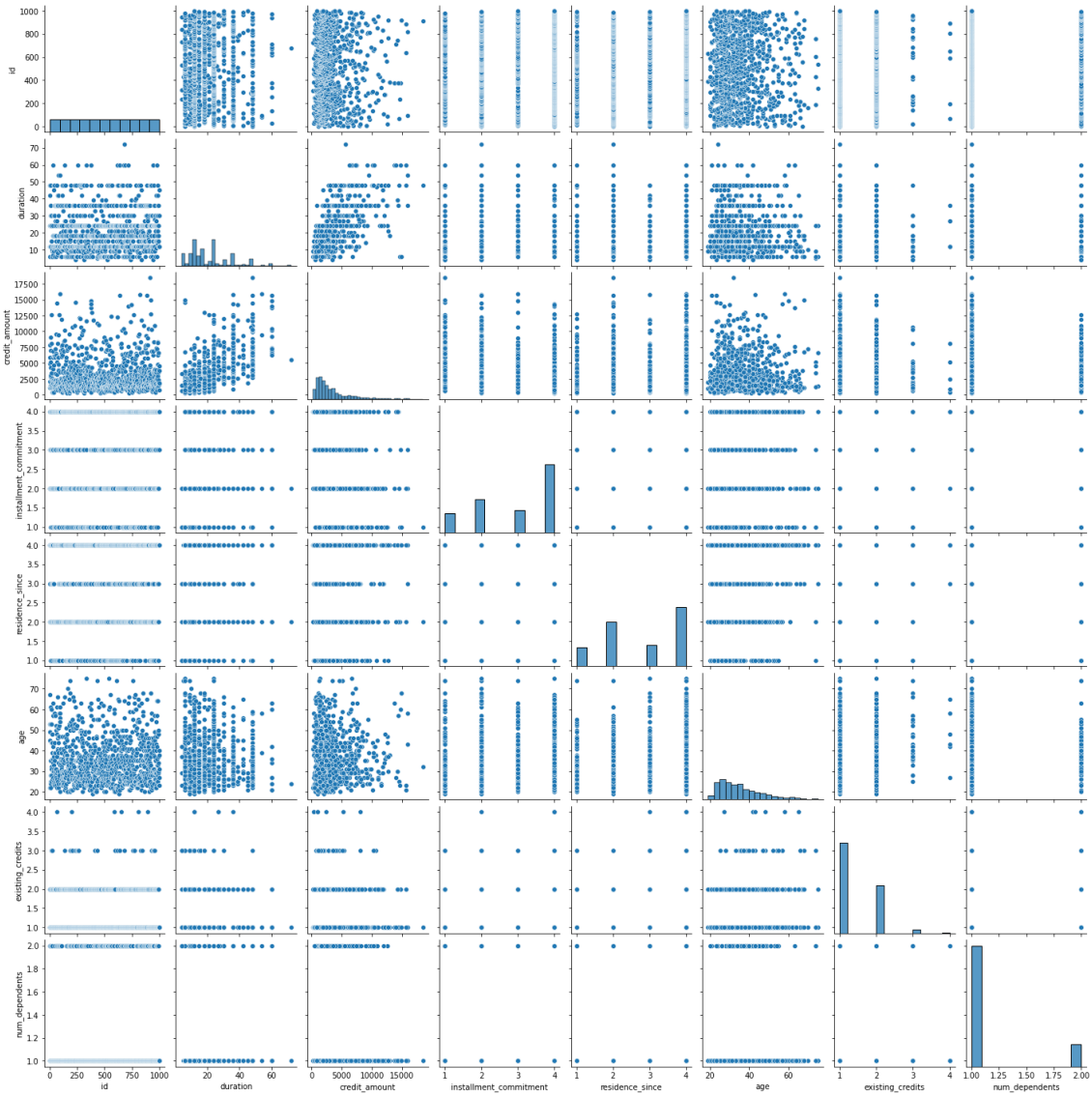
plt.hist(datos[“age”], facecolor = ‘orange’) corresponde con la gráfica del centro que es un ***histograma***. Esta gráfica nos representa (eje x) la edad de las personas que han solicitado el crédito frente al número de solicitantes (eje y). Por ejemplo: La mayoría de las personas que han solicitado un crédito tienen menos de 30 años.

plt.scatter(datos[“existing_credits”],datos[“age”]) corresponde con la gráfica de la derecha, que es una gráfica de tipo ***scatter***, es decir dispersión. Está gráfica nos representa la cantidad de créditos existentes (eje x), frente a la edad de los solicitantes (eje y).

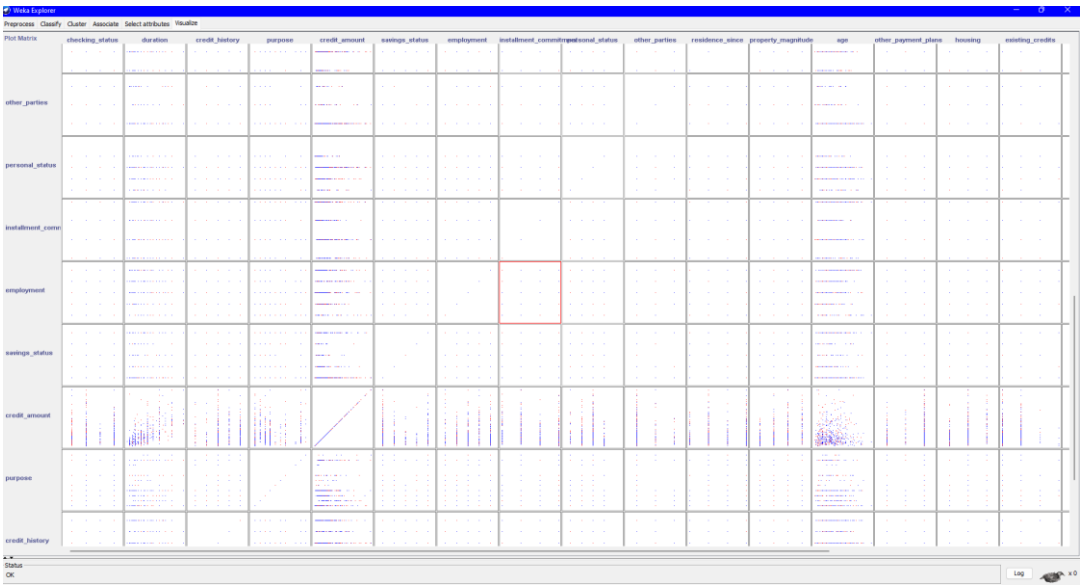
Continuamos implementando el siguiente código:

```
import seaborn as sns
sns.pairplot(datos.select_dtypes(exclude=[object]))
```

Colaboratory, nos generará lo siguiente:



Esta gráfica es parecida a la que nos generaba el programa WEKA.



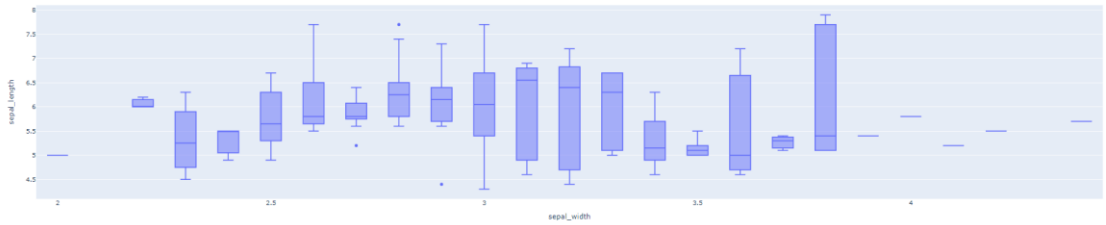
Colaboratory, nos cruza nuestras variables, generándonos diferentes gráficas, y nos ayuda a obtener una vista más general de las relaciones que tienen estás entre sí.

A continuación, implementamos el siguiente código:

```
import plotly.express as px

df = px.data.iris()

fig = px.box(df, x="sepal_width", y="sepal_length")

fig.show()
```

A través de este apartado generamos una gráfica relacionada con el Iris Dataset. Este tipo de gráficas estudian 4 características, el ancho y el largo de los sépalos y pétalos de las flores de Iris. En nuestro caso vamos a estudiar solo los sépalos.

Colaboratory, nos generará lo siguiente:



Esta gráfica implementa el Iris Dataset, y nos ayuda a encontrar patrones, relaciones y plantear hipótesis y preguntas sobre nuestros datos. Sin embargo, en este caso, no estamos empleando los datos de nuestro dataset, sino que estamos empleando los datos que se nos han proporcionado originalmente en el código. Por tanto, esta parte en concreta no nos proporciona información nueva sobre nuestro fichero csv.

Continuamos haciendo uso del siguiente código. Este nos generará un archivo tipo .html.

```
import plotly
from plotly.offline import plot
import plotly.graph_objs as go

plotly.offline.plot({
"data": [
    go.Scatter(    x=[1, 2, 3, 4],
    y=[10, 11, 12, 13], mode='markers',
    marker=dict(
        size=[40, 60, 80, 100]))],
"layout": go.Layout(showlegend=False,
    height=600,
    width=600,
)
})
```

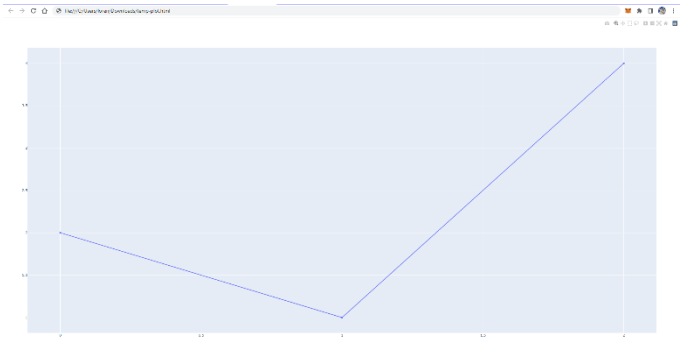
```
'temp-plot.html'
```

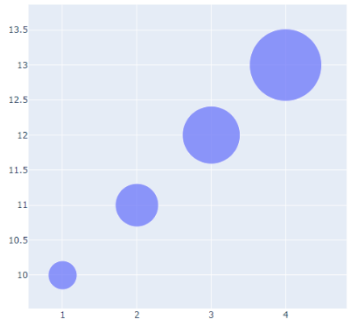
```
[44] fig = go.Figure(data=[{'type': 'scatter', 'y': [2, 1, 4]}])

plot(fig)
```

```
'temp-plot.html'
```

Este fragmento de código hace uso de go.Scatter(), una función que realiza el scattering, es decir dispersión.

Abrimos los dos archivos temp-plot.html para observar la gráfica que nos ha gen erado.





Preprocesamiento

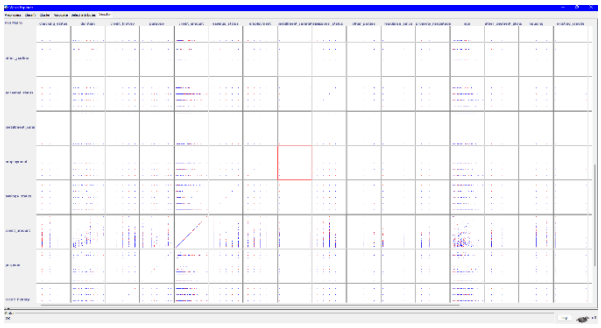
A continuación, aplicamos el siguiente código, donde remplazaremos las variables originales por variables de tipo numérico. En nuestro caso utilizaremos la opción 1 y haremos uso de la variable “age”.

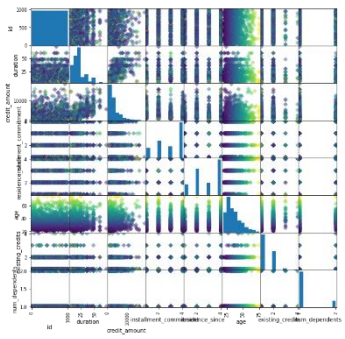
```
# ======================================================================
# Preprocesamiento
# ======================================================================
#Borrar atributos (variables) que no sean numéricos
#1º opción. Seleccionar solo las variables numéricas y añadir variable objetivo
num_datos = datos.select_dtypes(exclude=[object])
num_datos.loc[:, "age"] = datos["age"]

#2º opción. Borrar columnas
#num_datos = datos.drop(["id", "credit_history", "purpose", "credit_amount", "savings_status", "personal_status", "other_parties", "residence_since", "property_magnitude"],
#                       axis = 1) #filas 0


#Reemplazar los valores
#num_datos["checking_status"][83] = 10

#Borrar registros nulos (o vacios)
num_datos = num_datos.dropna()

#Convertir var. obj. en numérico
#num_datos["age"].unique()

#num_datos.loc[num_datos.loc[:, "age"] == "+", "age"] = 1 #Selección de la Class == '+'
#num_datos.loc[num_datos.loc[:, "age"] == "-", "age"] = 0 #Selección de la Class == '-'


num_datos["age"].unique()

pd.plotting.scatter_matrix(num_datos,
                           c = num_datos['age'], # color
                           figsize = [10, 10], # tamaño de la ventana
                           s=35, #tamaño del marcador
                           marker = 'D') # tipo de marcar los puntos
```

Una vez más, hacemos uso de dispersión al utilizar pd.plotting.scatter_matrix(), donde realizamos la dispersión dentro la matriz. Nos enfocamos en la edad del usuario.

Por tanto, Google Colaboratory nos generará lo siguiente:

Colaboratory, nos cruza nuestras variables, generándonos diferentes gráficas, y nos ayuda a obtener una vista más general de las relaciones que tienen estás entre sí.





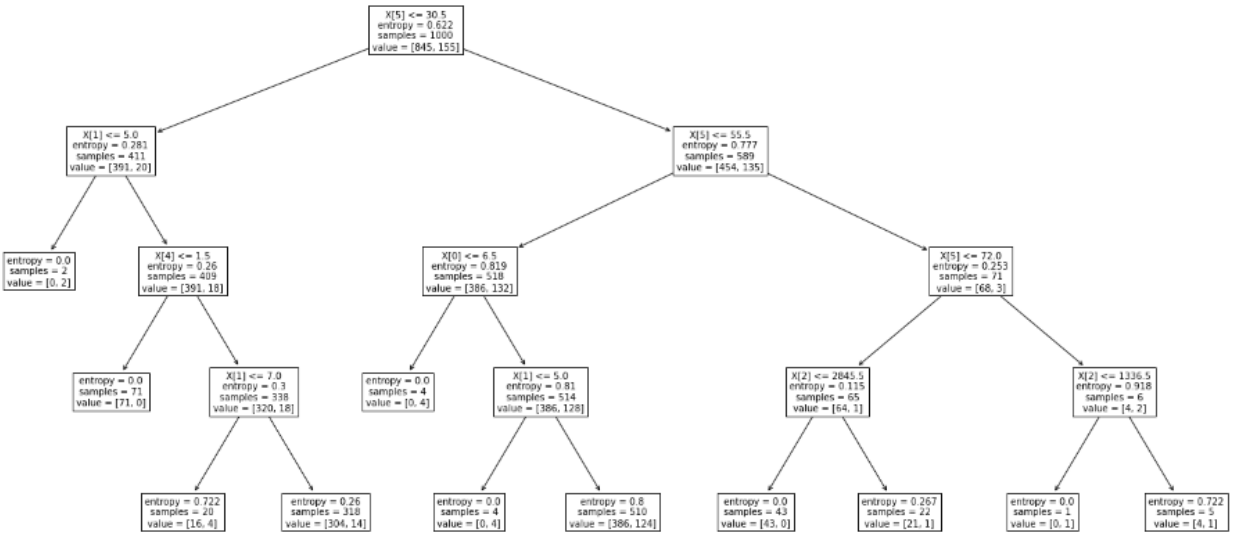
Como podemos observar, las gráficas generadas (derecha) se parecen una vez más a lo previamente generado en la práctica anterior en WEKA (izquierda).

Métodos supervisados. Decision Trees. Sklearn. CLASIFICACIÓN.

Empleamos el siguiente código, habiendo utilizado variables numéricas en los apartados anteriores. Esto es importante, ya que si no hacemos esto las variables X e Y pasaran a ser *Strings*. Por tanto, es muy importante que hayamos empleado variables de tipo ENTERO para la realización de este apartado.

```
# ========================================================================
# Métodos supervisados. Decission Trees - Sklearn. CLASIFICACION
# ========================================================================
# Importando el árbol de decisión
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt

#Selección de las variables
X = num_datos.values[:, :-1] #Variables explicativas
y = num_datos.values[:, -1] #variable objetivo
y = y.astype("int")

# Creando el modelo
model = DecisionTreeClassifier(criterion='entropy', max_depth=4)

# Ajustando el modelo
model = model.fit(X, y)

#Dibujar el árbol entrenado
fig, ax = plt.subplots(figsize=(25, 12)) #Tamaño del gráfico
tree.plot_tree(model, fontsize = 10)


num_datos.columns
```

Tras emplear este código Google Colaboratory nos genera lo siguiente:



El árbol de decisión esta construido por los siguientes elementos: nodo, vectores, flechas y etiquetas.

Para entender el árbol de decisión de mejor manera tenemos que entender los términos representados en cada uno de estos nodos, mientras que X[i] hace referencia al índice de la variable dentro del array de variables. Dependiendo de si es menor o mayor al dato indicado, tirará o no tirará hacia el nodo siguiente.

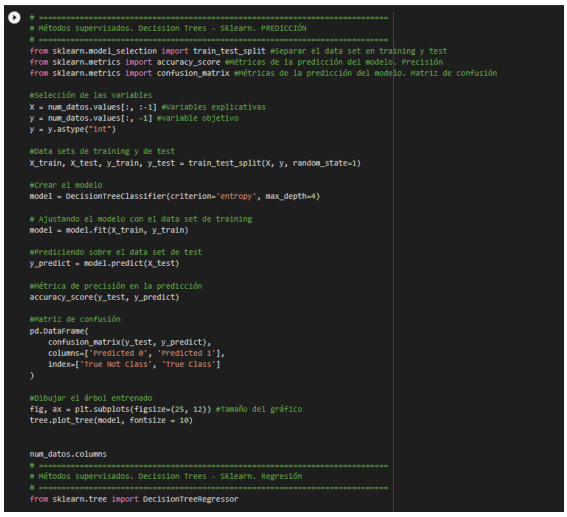
Value se refiere a los valores. Samples se refiere a la cantidad de muestras en este apartado. Entropy se refiere a la medida de la aleatoriedad en la información que se procesa.

Este árbol de decisión es generado a partir del algoritmo de clasificación, y representa unas reglas, que sirven para representar y categorizar una serie de condiciones que ocurren de forma sucesiva.

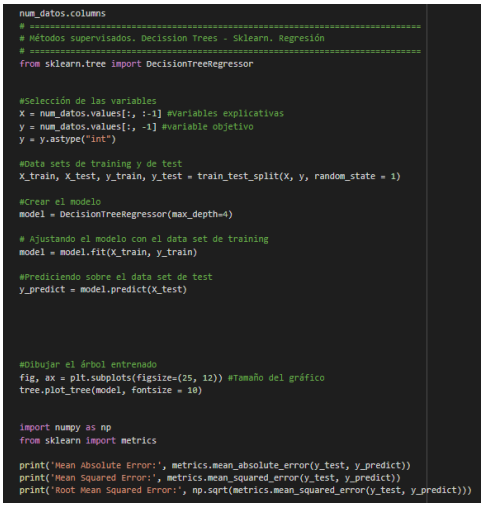
Ejemplo de interpretación del árbol: En el nodo padre, la variable en el índice 5 si es menor o igual que 30,5 tendrá una entropía de 0,622. 1000 personas coinciden con estos datos. De esas 1000 personas, 845 van a la izquierda frente a 155 que van a la derecha

Métodos supervisados. Decision Trees. Sklearn. Predicción y Regresión.

A continuación, empleamos el siguiente código:



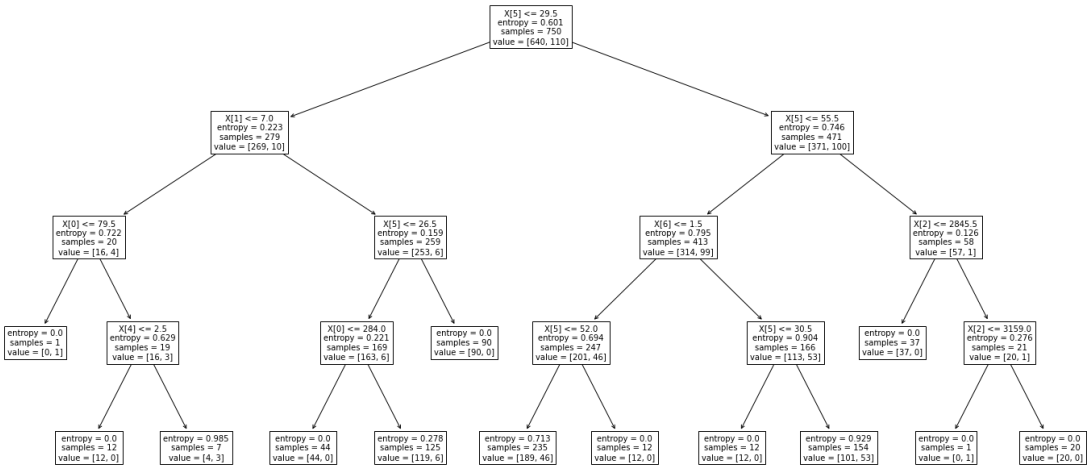
```
# ========================================================================
# Métodos supervisados. Decission Trees - Sklearn. PREDICCIÓN
# ========================================================================
from sklearn.model_selection import train_test_split #Separar el data set en training y test
from sklearn.metrics import accuracy_score #Métricas de la predicción del modelo. Precisión
from sklearn.metrics import confusion_matrix #Métricas de la predicción del modelo. Matriz de confusión

#Selección de las variables
X = num_datos.values[:, :-1] #Variables explicativas
y = num_datos.values[:, -1] #variable objetivo
y = y.astype("int")

#Data sets de training y de test
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

#Crear el modelo
model = DecisionTreeClassifier(criterion='entropy', max_depth=4)

# Ajustando el modelo con el data set de training
model = model.fit(X_train, y_train)

#Prediciendo sobre el data set de test
y_predict = model.predict(X_test)

#Métrica de precisión en la predicción
accuracy_score(y_test, y_predict)

#Matriz de confusión
pd.DataFrame(
    confusion_matrix(y_test, y_predict),
    columns=['Predicted 0', 'Predicted 1'],
    index=['True Not Class', 'True Class']
)

#Dibujar el árbol entrenado
fig, ax = plt.subplots(figsize=(25, 12)) #Tamaño del gráfico
tree.plot_tree(model, fontsize = 10)


num_datos.columns
# ========================================================================
# Métodos supervisados. Decission Trees - Sklearn. Regresión
# ========================================================================
from sklearn.tree import DecisionTreeRegressor
```



```
num_datos.columns
# ========================================================================
# Métodos supervisados. Decission Trees - Sklearn. Regresión
# ========================================================================
from sklearn.tree import DecisionTreeRegressor


#Selección de las variables
X = num_datos.values[:, :-1] #Variables explicativas
y = num_datos.values[:, -1] #variable objetivo
y = y.astype("int")

#Data sets de training y de test
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 1)

#Crear el modelo
model = DecisionTreeRegressor(max_depth=4)

# Ajustando el modelo con el data set de training
model = model.fit(X_train, y_train)

#Prediciendo sobre el data set de test
y_predict = model.predict(X_test)




#Dibujar el árbol entrenado
fig, ax = plt.subplots(figsize=(25, 12)) #Tamaño del gráfico
tree.plot_tree(model, fontsize = 10)


import numpy as np
from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_predict))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_predict))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_predict)))
```

Como podemos ver, el código nos genera un árbol de decisión a partir de un árbol de clasificación y posteriormente emplea un algoritmo de regresión.

Google Colaboratory nos generará el siguiente árbol de decisión:



El árbol de regresión tiene como objetivo predecir la variable respuesta y calcula la relación estimada entre una variable dependiente y una o varias variables explicativas. Gracias al uso de este algoritmo, el usuario podrá modelar la relación entre las variables elegidas.

Ejemplo de interpretación de nodo árbol: En el nodo padre la variable en el índice 5 si es menor o igual que 29,5 tendrá una entropía de 0,601. 750 personas coinciden con estos datos. De estas 750 personas, 640 van a la izquierda y 110 van a la derecha.

Adicionalmente, el código nos proporciona el Mean Absolute Error, Mean Squared Error y el Root Mean Squared Error.

```
Mean Absolute Error: 0.2504406642633162
Mean Squared Error: 0.14222157733374274
Root Mean Squared Error: 0.37712276162244934
```

Métodos NO Supervisados. Clústering – Kmeans, Sklearn.

A continuación, empleamos el siguiente código que nos proporcionará una gráfica de clustering a través del uso del algoritmo KMEANS. Recordamos que esto también lo hicimos en la práctica anterior mediante el uso de la aplicación WEKA.

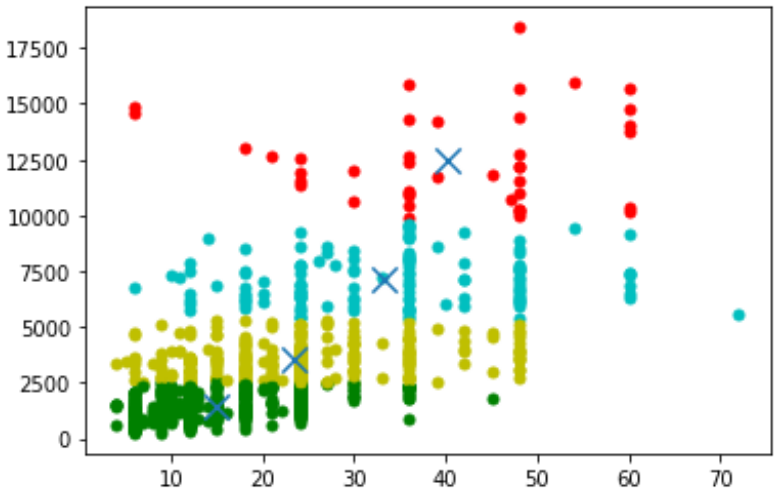
```
# ========================================================================
# Métodos NO supervisados. Clústering - KMeans, Sklearn
# ========================================================================

from sklearn.cluster import KMeans

#Seleccionar los atributos "credit_amount" y "installment_commitment"
K_datos = num_datos[["duration", "credit_amount", "installment_commitment", "residence_since"]]

#Crear modelo
kmeans = KMeans(n_clusters = 4)

#Ajustar modelo
kmeans.fit(K_datos)

#Dibujar el clústering
centroids = kmeans.cluster_centers_
labels = kmeans.labels_

#Dibujar los puntos con los colores de cada clúster
colors = ["g.","r.","c.","y."]
for i in range(len(K_datos)):
    plt.plot(K_datos.iloc[i,0], K_datos.iloc[i,1], colors[labels[i]], markersize = 10)

#Dibujar los centroides de cada clúster
plt.scatter(centroids[:, 0],centroids[:, 1], marker = "x", s=150, linewidths = 5, zorder = 10)

plt.show()



K_datos["cluster"] = labels


summary0 = K_datos[K_datos.cluster == 0].describe()
summary1 = K_datos[K_datos.cluster == 1].describe()
summary2 = K_datos[K_datos.cluster == 2].describe()
summary3 = K_datos[K_datos.cluster == 3].describe()
```

En k_datos empleamos datos de tipo numérico. En nuestro caso emplearemos las variables “duration”, “credit_amount”, “installment_comittment” y “residence_since”.

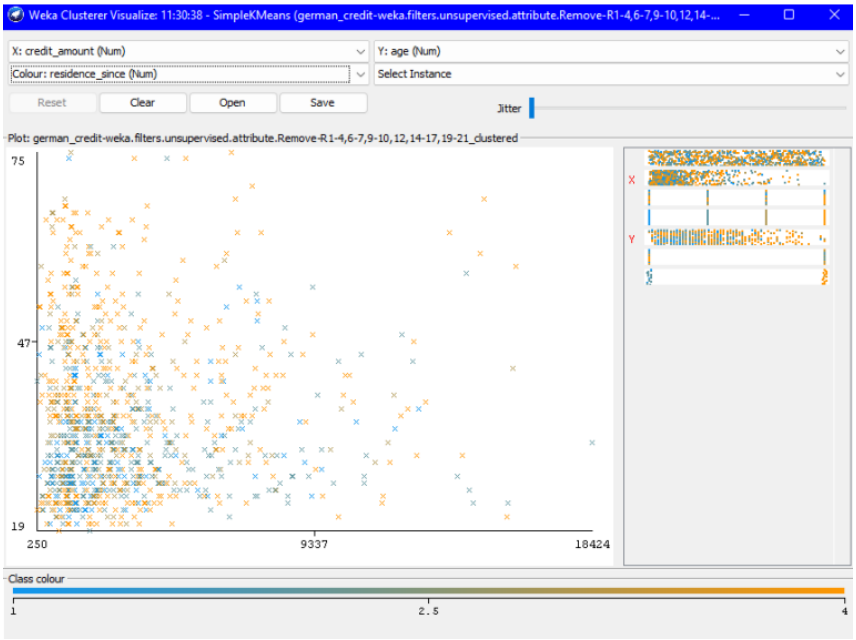
Recordamos la definición del algoritmo K-means que nos aportó WEKA.

Vamos a agrupar datos haciendo uso del algoritmo de agrupamiento K-medias. El programa permite hacer uso de la distancia euclidiana (por defecto) o la distancia de manhattan. En caso de que se utilice la segunda, los centroides se calcularán como la mediana de los componentes en lugar de la media.

Google Colaboratory nos ha generado la siguiente gráfica de dispersión:



Vamos a compararlo con la gráfica nuestra de WEKA.



A pesar de ser una gráfica diferente (teniendo en cuenta que estamos empleando en WEKA la variable age y class) podemos observar que efectivamente, WEKA también nos generó una gráfica de dispersión (scatter) en la práctica anterior.

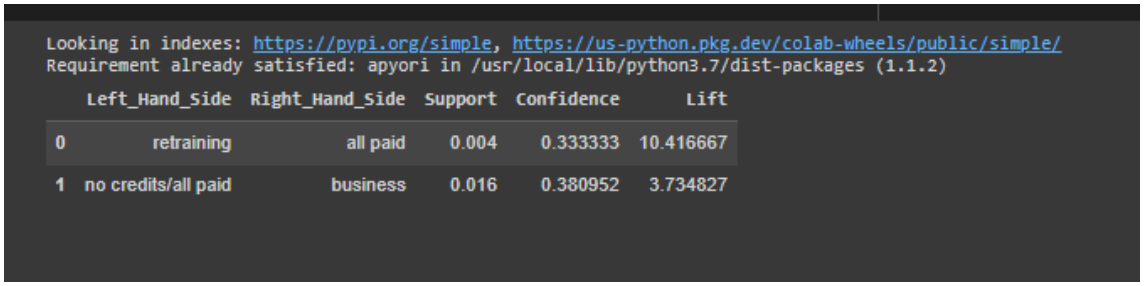
Reglas de Asociación

A continuación, empleamos el siguiente código:

```
###### Reglas de asociación

!pip install apyori

import numpy as np
import pandas as pd

datos = datos[["checking_status", "credit_history", "purpose", "class"]]

# Intializing the list
transacts = []
# populating a list of transactions
for i in range(0, 500):
  transacts.append([str(datos.values[i,j]) for j in range(0, 3)]) # la ultima columna 3

from apyori import apriori
rule = apriori(transactions = transacts, min_support = 0.003, min_confidence = 0.2, min_lift = 3, min_length = 2, max_length = 2)



output = list(rule) # returns a non-tabular output
# putting output into a pandas dataframe
def inspect(output):
    lhs         = [tuple(result[2][0][0])[0] for result in output]
    rhs         = [tuple(result[2][0][1])[0] for result in output]
    support     = [result[1] for result in output]
    confidence = [result[2][0][2] for result in output]
    lift        = [result[2][0][3] for result in output]
    return list(zip(lhs, rhs, support, confidence, lift))
output_DataFrame = pd.DataFrame(inspect(output), columns = ['Left_Hand_Side', 'Right_Hand_Side', 'Support', 'Confidence', 'Lift'])


output_DataFrame
```

Donde empleamos en los datos, variables de tipo nominal. En nuestro caso emplearemos las variables “checking_status”, “credit_history”, “purpose” y “class”.

Google colaboratory nos ha generado la siguiente tabla a raíz de esta elección de variables:



```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: apyori in /usr/local/lib/python3.7/dist-packages (1.1.2)
```

	Left_Hand_Side	Right_Hand_Side	Support	Confidence	Lift
0	retraining	all paid	0.004	0.333333	10.416667
1	no credits/all paid	business	0.016	0.380952	3.734827

Observando la table generada podemos llegar a la conclusión de que el código nos está mostrando la relación entre la variable *purpose*, y la variable *checking_status*. Nos muestra que el crédito ha sido pagado al completo en el caso de *retraining*, con un porcentaje de confianza del 33%. Mientras que nos muestra que no tienes créditos o que han sido pagados al completo en el caso de *business*, con un porcentaje de confianza del 38%.

La tabla solo nos muestra resultados si hay coincidencias con las variables representadas.

Para generar esta tabla hemos empleado el algoritmo a priori. Este algoritmo también lo hemos utilizado en la práctica anterior. Vamos a recordar que es lo que hace dicho algoritmo:

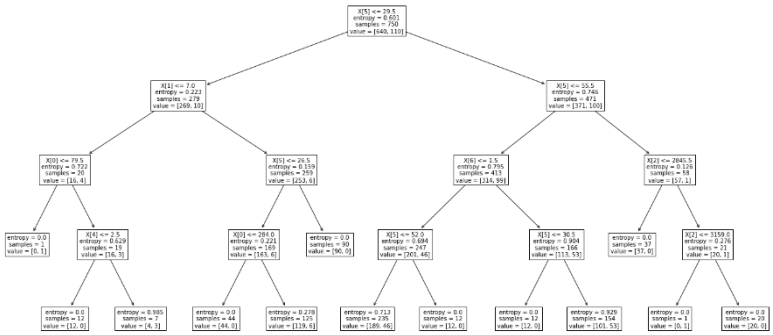
El algoritmo A priori reduce de manera iterativa el soporte mínimo hasta que encuentra el número requerido de reglas con el mínimo de confianza requerida. El algoritmo tiene como opción picar las reglas de clases de asociación.

Similitud con WEKA.

A continuación, vamos a observar las gráficas que generamos en la práctica anterior, para ver que podemos realizar las mismas acciones en la aplicación WEKA que en Google Colaboratory.

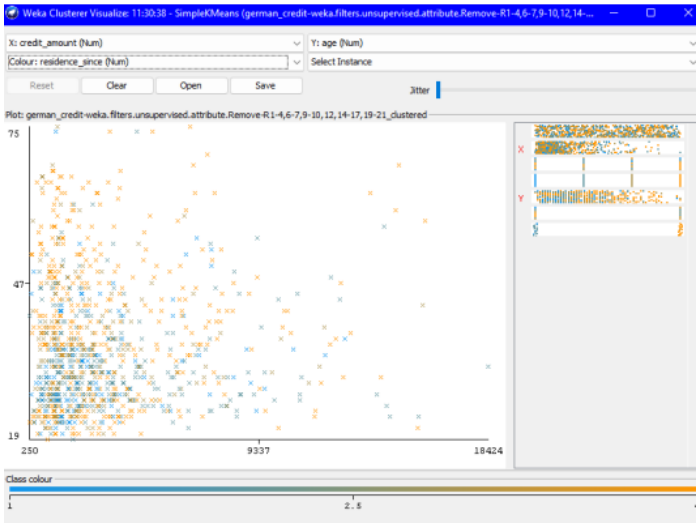
Arboles de decisión:

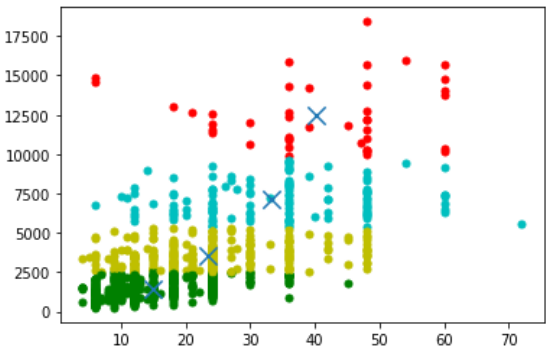
WEKA también nos permite generar arboles de decisión. Sin embargo, en Google Colaboratory, estos son visualmente más atractivos y nos permite visualizar la información de manera más sencilla.



SimpleKmeans:

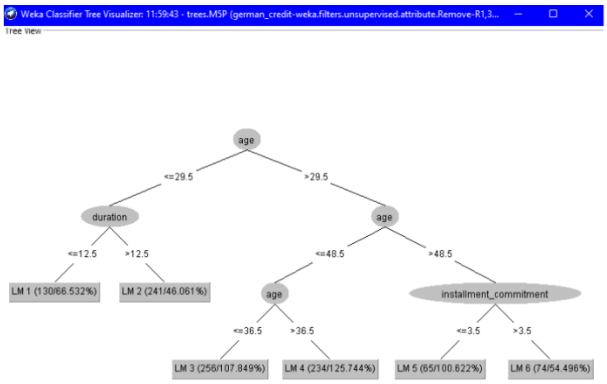
Weka también nos permite visualizar gráficas de dispersión tras hacer uso del algoritmo SimpleKmeans. Sin embargo, la gráfica generada por Google Colaboratory es visualmente más atractiva. Tenemos en cuenta que estamos observando datos distintos en las dos gráficas.

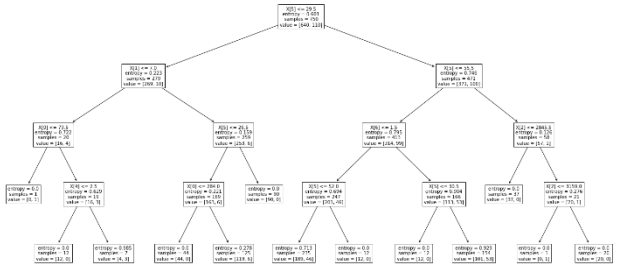




Arboles de Regresión: WEKA también nos permite generar arboles de regresión como podemos visualizar en el siguiente gráfico.

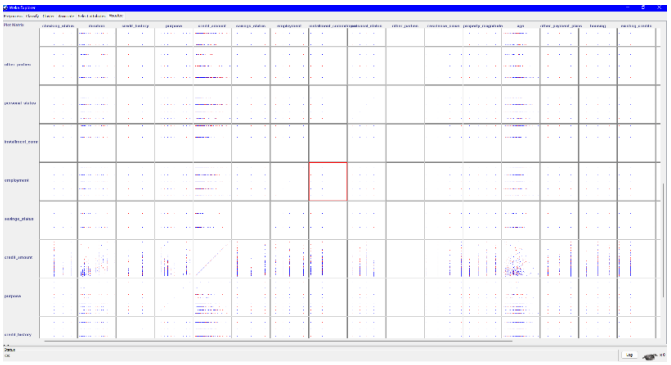
A continuación, vamos a mostrar el árbol generado por Google Colaboratory.

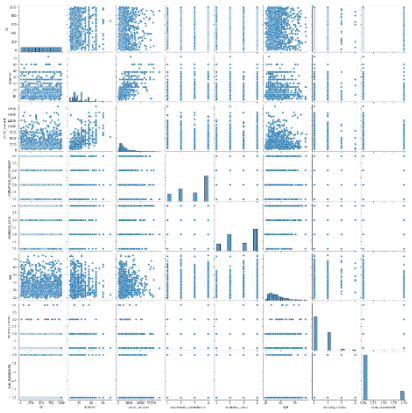




Efectivamente, podemos generar arboles de regresión en los dos entornos de trabajo.

Visualización general:





WEKA también nos brinda con la oportunidad de poder visualizar todas las gráficas generadas por la relación que tienen las variables entre sí. Google Colaboratory, también nos permite hacer esto como podemos ver aquí:

Conclusión

Google Colaboratory y WEKA nos permiten estudiar los datos de un archivo .csv de manera eficiente. Podemos realizar las mismas tareas y hacer uso de diferentes tipos de algoritmos para obtener resultados distintos, en los dos entornos de trabajo. Sin embargo, Google Colaboratory es visualmente muchísimo más atractivo y nos permite customizar visualmente hablando los gráficos.