



# TRABAJO PRACTICO INTEGRADOR

MODULO 3 MACHINE LEARNING

## DESCRIPCIÓN BREVE

En este presente documento se podra encontrar con una descripcion del codigo, algunas librerias, la instalacion de las mismas, los problemas que se encontraron al utilizarlos algoritmos hasta poder dejar andando el programa machine learning que se realizo en el presente trabajo y el cual se evalua su correcto funcionamiento y entendimiento en el modulo, esto es una explicacion sobre el programa creado, los desafios encontrados y las soluciopnes dadas

## Autor

Julio Arturo Rodriguez

julio artro rodriguez

## Introduccion

**E**n el presente texto se van a abordar los pasos que se siguieron para la creacion de un programa machine learning cuya utilidad es la deteccion de fraudes bancarios y tiene un acierto del 0.999% utilizando una matriz de confucion accurasy. Pero antes de seguir y explicar quiero agradecer a la Universidad Nacional de Misiones por participar con el programa de alcance federal Argentina programa 4.0, del cual es trabajo final del presente modulo en el que me encuentro y es el trabajo practico integrador este informe, siendo el mismo el modulo 3 del curso de programador machine learning. Dare una aclaracion por adelantado, mientras mas cerca este del 1 esta mas cerca de llegar al 100% de aciertos.

Intentare explicar algunos algoritmos algoritmos dentro de lo posible para una mejor comprension y un abordaje mas completo, forma en la cual el lector entienda de lo que hablo, hare de cuenta que el lector no tiene conocimientos y abarcare desde 0, eso si, en caso de considerarlo necesario por obviedad no explicare todo, solo lo absolutamente necesario teniendo en cuenta que este es el final del modulo 3, por obvias razones puede que saltee algun algoritmo por darlo por ya entendido a estas alturas, lo que si no dare saltos en la explicacion de los algoritmos y tecnicas creadas en este modulo.

Para la creacion del programa se hizo uso de las explicaciones de los profesores y los materiales aportados por los mismos en el aula virtual, tambien se hizo utilización de las clases grabadas y una investigacion cientifica utilizando plataformas como youtube y buscadores como crome y firefox.

## Caso de estudio

El área a investigar son los fraudes bancarios, se nos da un dataset desbalanceado y sin muchos detalles por temas de confidencialidad. El trabajo consistía en crear un programa que distinga y prediga cuáles son fraudes y cuáles no lo eran. Las correctas y las incorrectas nos lo dieron en esta forma. Es importante que las empresas de tarjetas de crédito sean capaces de reconocer las transacciones fraudulentas con tarjeta de crédito para que no se carguen a los clientes artículos que no compraron. El conjunto de datos contiene transacciones realizadas con tarjetas de crédito en septiembre de 2013 por titulares de tarjetas europeos. Este conjunto de datos presenta transacciones que ocurrieron en dos días, donde tenemos 492 fraudes de 284.807 transacciones. El conjunto de datos está muy desequilibrado, la clase positiva (fraudes) representa el 0,172% de todas las transacciones.

Basándonos en lo anterior podemos decir que el programa creado fue un éxito y tuvo una tasa de acierto de casi el 100%.

## Desarrollo del proyecto

En las siguientes páginas se verán los conceptos incorporados y documentados mediante imágenes, explicando la finalidad del

código y su lógica. En pocas palabras explicare como pone su granito de arena cada algoritmo hasta completar el programa

Para el desarrollo utilizamos las siguientes librerías.

1. `#librerías`
2. `import pandas as pd`
3. `import numpy as np`
4. `import tensorflow as tf`
5. `from sklearn.model_selection import train_test_split`
6. `from sklearn.preprocessing import StandardScaler`
7. `from sklearn.metrics import confusion_matrix`
8. `import matplotlib.pyplot as plt`
9. `import seaborn as sns`
10. `from tensorflow import keras`
11. `from keras.layers import Dense`
12. `from keras.models import Sequential`
13. `import numpy as np`
14. `from sklearn.datasets import make_classification`
15. `from collections import Counter`
16. `from imblearn.over_sampling import RandomOverSampler`
17. `import matplotlib.pyplot as plt`

```

#librerias
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow import keras
from keras.layers import Dense
from keras.models import Sequential

#Procesamiento de los datos
#Divide los datos en características (X) y etiquetas (y)
data = pd.read_csv('basedatos.csv')
X = data.drop('class', axis=1)
y = data['class']
df = pd.read_csv(file_path, sep=',')

#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=42)

#Escala las características para normalizar
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

```

## Breve explicacion de las librerias

- pandas= en pocas palabras es un dataframe
- numpy es una librería de analisis de datos
- tensorflow es una librería dedicada al machine learning como dedicada al reconocimiento o vision por computadora
- from sklearn.model\_selection import train\_test\_split este nos permitira dividir el dataset en prueba, test y entrenamiento
- from sklearn.preprocessing import StandardScaler este nos permite estandarizar características eliminando la media y escalando, es decir optimisa los datos
- from sklearn.metrics import confusion\_matrix ahora importo una libreria para utilizar una matris de confucion
- import matplotlib.pyplot as plt ahora estoy importando una libreria de visualizacion de datos
- import seaborn as sns esta es otra librería de visualizacion de datos

- `from tensorflow import keras` esta pertenece a tensorflow, crea redes neuronales
- `from keras.layers import Dense` agrega una capa a la red neuronal
- los demas no los nombre porque son subprogramas de las librerias ya utilizadas.
- Pero hay algo que debo de explicar y es de gran importancia. Un dataset es un archivo o mejor dicho en pocas palabras es una base de datos exclusiva del programa machine learning, ambos guardan archivos.

Luego de instalar e importar las siguientes librerias segui con el siguiente codigo

#Procesamiento de los datos

# Divide los datos en características (X) y etiquetas (y)

```
data = pd.read_csv('basededatos.csv')
```

```
X = data.drop('Class', axis=1)
```

```
y = data['Class']
```

```
df = pd.read_csv(file_path, sep=',')
```

# 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=42)
```

# Escala las características para normalizar

```
scaler = StandardScaler()
```

```
X_train = scaler.fit_transform(X_train)
```



**X\_test = scaler.transform(X\_test)**

Breve explicacion.

En data = **pd.read\_csv('basededatos.csv')** pd es una variable y read\_csv() es una funcion de lectura de datos de pandas.

En esta X = data.drop('Class', axis=1)

y = data['Class'] estamos utilizando las variables x e y para dos tipos de datos, la x son de entrada y la y de salida.

Luego con el siguiente codigo cargamos los datos al dataframe de pandas

**df = pd.read\_csv(file\_path, sep=',')**

en el siguiente codigo asignamos la cantidad de datos a utilizar el prueba, test y entrenamiento

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

Como podemos ver se asigno un 20% a testeo y prueba

**scaler = StandardScaler()**

**X\_train = scaler.fit\_transform(X\_train)**

**X\_test = scaler.transform(X\_test)**

En el codigo anterior escalamos para que ningun dato tenga mas peso que otro.

## Red neuronal

Para la creacion de la red neuronal se utilizo el estandar y utilizado por estudiantes y docentes desde el modulo 2

**model = Sequential()**

**model.add(Dense(units=128, activation='relu', input\_dim=X\_train.shape[1]))**

**model.add(Dense(units=64, activation='relu'))**

**model.add(Dense(units=1, activation='sigmoid'))**



```

Escenario_con_arquitectura_inicial (1) ipynb
Escenario_con_arquitectura_inicial (1) ipynb > import matplotlib.pyplot as plt
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Creación de la red neuronal y entrenamiento

model = Sequential()
model.add(Dense(units=128, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))
# Compilar el modelo
model.compile(loss='binary_crossentropy', optimizer='adam')

# Entrenar el modelo
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)

Epoch 1/10
7121/7121 [=====] - 10s 1ms/step - loss: 0.0059
Epoch 2/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0030
Epoch 3/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0027
Epoch 4/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0025
Epoch 5/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0025
Epoch 6/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0024
Epoch 7/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0021
Epoch 8/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0020
Epoch 9/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0019
Epoch 10/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0018
... <keras.src.callbacks.History at 0x1c9b771d0>

```

despues de eso lo compile para que afuncione utilizando

**odel.compile(loss='binary\_crossentropy', optimizer='adam')**

luego lo entrene para saber si estaba bien

**model.fit(X\_train, y\_train, epochs=10, batch\_size=32, verbose=1)**

luego de eso utilize un mapa de calor

**Calcular la matriz de correlación**

**correlation\_matrix = df.corr()**

# Configurar el tamaño del mapa de calor

**plt.figure(figsize=(25, 13))**

# Crear el mapa de calor de correlación

```
sns.heatmap(correlation_matrix, cmap="RdYlGn")
```

```
# Mostrar el mapa de calor
```

```
plt.show()
```

```
plt.figure(figsize=(25, 13))
```

```
sns.heatmap(correlation_matrix, annot=True, cmap="RdYlGn")
```

```
plt.show()
```

## Matriz de confucion

En la imagen se puede obsevar la cantidad de aciertos positivos, aciertos negativos, en pocas palabras los vv, fv, ff, fv.

En la imagen se puede observar lo siguiente, no verdadero 56.853, no falsos 11, verdaderos falsos 22, verdaderos verdaderos 76.

```

# Ejercicio con arquitectura neural (1)pyth
+ Code + Markdown + Run All + Restart + Clear All Outputs + Variables + Outline
# Realizar predicciones en el conjunto de prueba
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5) # Convierte las probabilidades en etiquetas binarias (0 o 1)

# Calcula la matriz de confusión (usando "y_test" y "y_pred")
# Realizar predicciones en el conjunto de prueba
y_probs = model.predict(X_test)
y_pred = np.round(y_probs)

# Calcular la matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)

# Nombres de clases
class_names = ["No", "Yes"]

# Utilizar la matriz de confusión con pandas
conf_df = pd.DataFrame(conf_matrix, index=class_names, columns=class_names)

# Mostrar la matriz de confusión estilizada
print("\nMatriz de Confusión:")
print(conf_df)

# Calculamos el Accuracy
# Obtener los valores de la matriz de confusión
TN, FP, FN, TP = conf_matrix.ravel()

# Calcular el accuracy
accuracy = (TP + TN) / (TP + FN + FP + TN)

print("Accuracy:", accuracy)

1781/1781 [=====] - 2s 924us/step
1781/1781 [=====] - 2s 924us/step

Matriz de Confusión:
      No  Yes
No  56853  11
Yes   22   76
Accuracy: 0.999420666489185

#Visualizar el desbalanceo del Dataset

```

Despues segui desde donde me quede anteriormente. Volvi a despues de la matriz de calor y puse.

```
import numpy as np
```

```
from sklearn.datasets import make_classification
```

```
from collections import Counter
```

```
from imblearn.over_sampling import RandomOverSampler
```

```
import matplotlib.pyplot as plt
```

```
# Contar las instancias por clase antes del oversampling
```

```
print('Distribución de clases antes del oversampling:', Counter(y))
```

```
plt.bar(Counter(y).keys(), Counter(y).values(), color='b', alpha=0.5,  
label='Antes del oversampling')
```

```
plt.xlabel('Clase')
```

```
plt.ylabel('Número de instancias')
```

```
plt.legend()
```

```
plt.show()
```

con eso vi mejor como se distribuian los datos. Luego aplique el oversampling y vi la nueva distribucion

```
# Aplicar oversampling utilizando RandomOverSampler
```

```
oversampler = RandomOverSampler(random_state=42)
```

```
X_resampled, y_resampled = oversampler.fit_resample(X, y)
```

```
# Contar las instancias por clase después del oversampling
```

```
print('Distribución de clases después del oversampling:',
```

```
Counter(y_resampled))
```

```
# Visualizar la distribución de las clases antes y después del oversampling
plt.bar(Counter(y_resampled).keys(), Counter(y_resampled).values(),
color='r', alpha=0.5, label='Después del oversampling')

plt.xlabel('Clase')

plt.ylabel('Número de instancias')

plt.legend()

plt.show()

despues utilice accurasy

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

y_pred = (y_pred > 0.5) # Convierte las probabilidades en etiquetas
binarias (0 o 1)
```

## Metrica

Despues intente utilizar la matriz de confucion f1 score pero no me dio buenos resultados, cuando me di cuenta que era una clade desbalanciada le agregue smote tomek y los aciertos no pasaban del 50%, decidi cambiar de metrica y probar con accurasy y sin smote tomet me dio un acierto del 40% y con smote tomek me dio 60%, viendo que se podia mejorar utilice un oversampling al hacer copias con los datos dados es posible que haya funcionado mejor porque el smote tomek crea copias sinteticas de los datos cercanos y la diferencia en este caso era enorme, dado el caso era mas rentable replicar parte de los datos.

# P

ara tener una mejor evaluación escogi utilizar la métrica Accuracy , ya que esta es la que mejor encaja en este escenario dado que provee otras y esta era la que mayor porcentaje de aciertos me dio. Siendo el monto anterior

Al aplicar la métrica obtuve que el acierto era del 0.999420666409185 , lo cual nos da a entender que la metrica esta funcionando bien.

```
# Calcula la matriz de confusión (usando "y_test" y "y_pred")
```

```
# Realizar predicciones en el conjunto de prueba
```

```
y_probs = model.predict(X_test)
```

```
y_pred = np.round(y_probs)
```

```
# Calcular la matriz de confusión
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
```

```
# Nombres de clases
```

```
class_names = ['No', 'Yes']
```

```
# Estilizar la matriz de confusión con pandas
```

```
conf_df = pd.DataFrame(conf_matrix, index=class_names,  
columns=class_names)
```

```
# Mostrar la matriz de confusión estilizada
```

```
print("\nMatriz de Confusión:")
```

```
print(conf_df)
```

```
#Calculamos el Accuracy
```

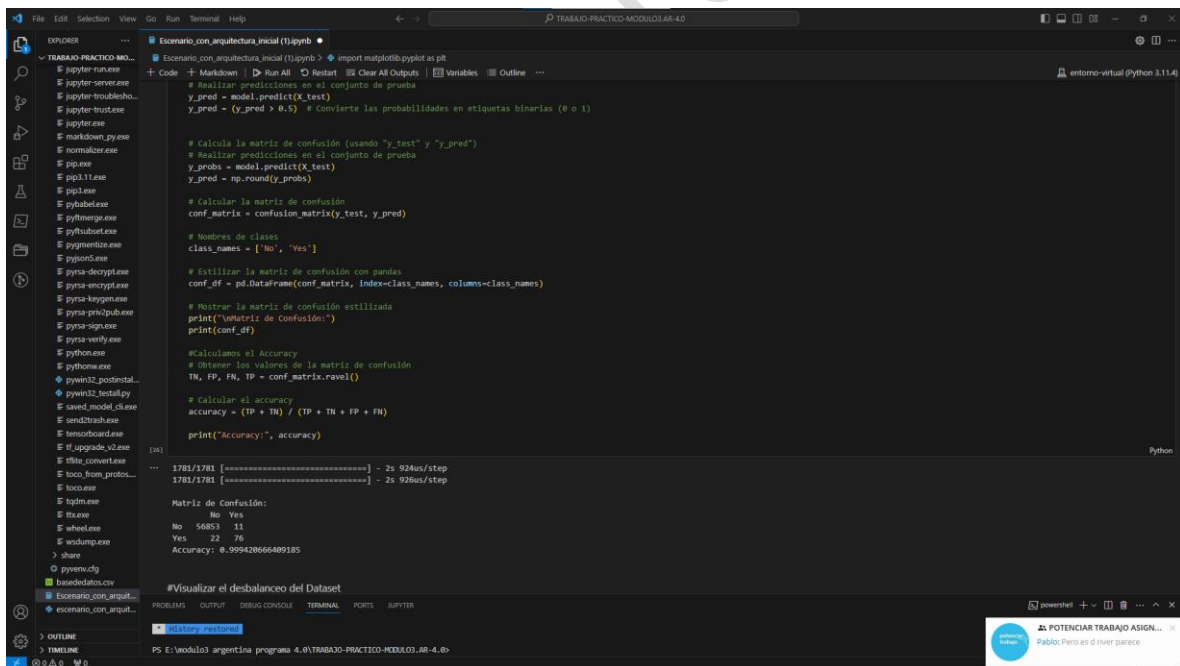
```
# Obtener los valores de la matriz de confusión
```

```
TN, FP, FN, TP = conf_matrix.ravel()
```

```
# Calcular el accuracy
```

```
accuracy = (TP + TN) / (TP + TN + FP + FN)
```

```
print("Accuracy:", accuracy)
```



```

Escenario con arquitectura inicial (11aprb)
+ Code + Markdown + Run All + Restart + Clear All Outputs + Variables + Outline
# Realizar predicciones en el conjunto de prueba
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5) # Convierte las probabilidades en etiquetas binarias (0 o 1)

# Calcular la matriz de confusión (usando "y_test" y "y_pred")
# Realizar predicciones en el conjunto de prueba
y_probs = model.predict(X_test)
y_pred = np.round(y_probs)

# Calcular la matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)

# Nombres de clases
class_names = ['No', 'Yes']

# Utilizar la matriz de confusión con pandas
conf_df = pd.DataFrame(conf_matrix, index=class_names, columns=class_names)

# Mostrar la matriz de confusión estilizada
print("Matriz de Confusión:")
print(conf_df)

# Calculamos el Accuracy
# Obtener los valores de la matriz de confusión
TN, FP, FN, TP = conf_matrix.ravel()

# Calcular el accuracy
accuracy = (TP + TN) / (TP + TN + FP + FN)

print("Accuracy:", accuracy)

1781/1781 [=====] - 2s 924ms/step
1781/1781 [=====] - 2s 926ms/step

Matriz de Confusión:
      No  Yes
No  14853  11
Yes   22   76
Accuracy: 0.999420666409185

#Visualizar el desbalanceo del Dataset

```

Luego utilice nuevamente una matriz de calor

```
plt.figure(figsize=(25, 13))
```

```
sns.heatmap(correlation_matrix, annot=True, cmap="RdYlGn")
```

```
plt.show()
```

lo volvi a entrenar con los nuevos datos

```
# Entrenar el modelo
```

```
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)
```

volvi a utilizar accurasy

```
# Realizar predicciones en el conjunto de prueba
```

```
y_pred = model.predict(X_test)
```

```
y_pred = (y_pred > 0.5) # Convierte las probabilidades en etiquetas binarias (0 o 1)
```

```
# Calcula la matriz de confusión (usando "y_test" y "y_pred")
```

```
# Realizar predicciones en el conjunto de prueba
```

```
y_probs = model.predict(X_test)
```

```
y_pred = np.round(y_probs)
```



```
# Calcular la matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)

# Nombres de clases
class_names = ['No', 'Yes']

# Estilizar la matriz de confusión con pandas
conf_df = pd.DataFrame(conf_matrix, index=class_names,
                        columns=class_names)

# Mostrar la matriz de confusión estilizada
print("\nMatriz de Confusión:")
print(conf_df)

#Calculamos el Accuracy
# Obtener los valores de la matriz de confusión
TN, FP, FN, TP = conf_matrix.ravel()

# Calcular el accuracy
accuracy = (TP + TN) / (TP + TN + FP + FN)

print("Accuracy:", accuracy)

despues
import matplotlib.pyplot as plt

# Calcular la cantidad de muestras por clase
```

```
class_counts = data['Class'].value_counts()
```

```
# Mostrar la cantidad de datos por clase
```

```
for class_name, count in class_counts.items():
```

```
    print(f'Clase: {class_name}, Cantidad de Datos: {count}')
```

```
# Crear el gráfico de barras
```

```
plt.figure(figsize=(8, 6))
```

```
class_counts.plot(kind='bar', color=['blue', 'red'])
```

```
plt.title('Desbalance de Clases')
```

```
plt.xlabel('Clase')
```

```
plt.ylabel('Cantidad de Muestras')
```

```
plt.xticks([0, 1], ['Normal', 'Fraud'])
```

```
plt.show()
```

*nota*

carga de los datos

```
import gdown
```

```
# Define el enlace compartido de Google Drive
```

```
url =
```

```
'https://drive.google.com/uc?id=1hga3zUzjhYqmR_aCpSIhYEktqClCIaV8'
```

```
# Especifica la ubicación donde deseas guardar el archivo
```

```
output = 'E://modulo3 argentina programa 4.0//TRABAJO-PRACTICO-MODULO3.AR-4.0//basededatos.csv'
```

# Descarga el archivo desde el enlace compartido

**gdown.download(url, output, quiet=False)**

por el medio anterior descargamos el dataset

## Elección y Aplicación de Métricas de Evaluación

Para tener una mejor evaluación escogi utilizar la métrica Accuracy , ya que esta es la que mejor encaja en este escenario dado que provee otras y esta era la que mayor porcentaje de aciertos me dio. Siendo el monto anterior

Al aplicar la métrica obtuve que el acierto era del 0.999420666409185 , lo cual nos da a entender que la métrica está funcionando bien.

## Análisis del Desbalance de Clases

Si analizamos los datos podemos ver que existe un desbalance entre las clases por lo que tendremos que balancearlas utilizando oversampling

En este caso decidí utilizar el método oversampling para balancear las clases, y luego si volvemos a entrenar el modelo con nuestro dataset balanceado, al volver a evaluar las métricas anteriores obtenemos que mejoro muy notablemente la cantidad de aciertos que antes no eran ni del 0.0004%

## Mapa de Calor y Matriz de Correlaciones

Para obtener el mapa de calor y la matriz de correlaciones se ejecutó el siguiente código. No me fue de mucha utilidad en este caso. Aun así me hubiera gustado eliminar todas las variables en rojo por la correlación.

```
import matplotlib.pyplot as plt
```

```
# Calcular la matriz de correlación
```

```
correlation_matrix = df.corr()
```

```
# Configurar el tamaño del mapa de calor
```

```
plt.figure(figsize=(25, 13))
```

```
# Crear el mapa de calor de correlación
```

```
sns.heatmap(correlation_matrix, cmap="RdYlGn")
```

```
# Mostrar el mapa de calor
```

```
plt.show()
```

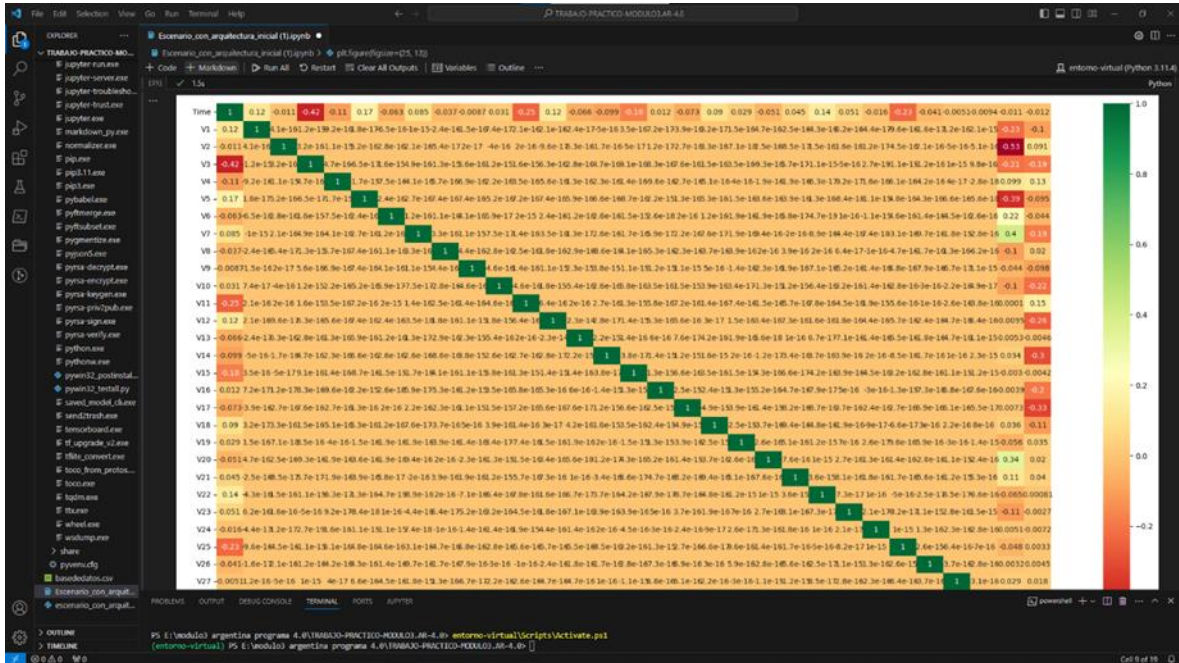
y luego

```
plt.figure(figsize=(25, 13))
```

```
sns.heatmap(correlation_matrix, annot=True, cmap="RdYlGn")
```

```
plt.show()
```

A continuación se muestra el mapa de calor y la matriz de correlaciones obtenida de los datos del escenario:



Se pueden observar al menos 3 variables que de haber eliminado hubiera llegado al 100% de efectividad el programa completo. Las que tienen un inconveniente en rojo y van bajando hasta amarillo y a llegar al verde.

Extra

A sabiendas de que hay 2 tipos de programadores siendo el primero un programador que se sabe la sintaxis y el segundo el que se sabe la lógica, en el dictado de las clases se vio mucho la lógica y se la priorizo no sin dejar de lado la sintaxis que también se vio una guía y se vieron en clases, también se dejó

material extra para complementar y se incentivó la investigación científica. El trabajo realizado en la primera etapa consistió en lógica algorítmica y sintaxis, con algunos ejercicios y cuestionarios mientras que en esta etapa consistió en cuestionarios, material teórico, creación de un programa de detección de fraude bancario que a diferencia del segundo módulo tuvimos más profundidad en el código, no por cantidad, si no porque a la hora de realizar el programa nos dieron todas las herramientas pero el código es nuestra creación al 100%.

granito de arena en este programa.

## Explicación rápida y sencilla del código

Todo lo que está antes de un punto es una variable. Ejemplos

**Pd. title('Desbalance de Clases')**

**Pd. xticks([0, 1], ['Normal', 'Fraud'])**

**Pd. show()**

El lado izquierdo del punto es una variable mientras que su derecha es la función

## Ampliación

Pandas: Se lo utilizo por ser un dataframe que se utiliza para para la manipulación y análisis de datos la cual nos ayuda a cargar, transformar y analizar los datos que utilizamos en este proyecto.

Matplotlib: a matplotlib lo usamos por ser una biblioteca de Python para la visualización mediante su creación de gráficos.

Nbformat: la cual nos permite trabajar con la implementación base de Jupyter Notebook.

Tensorflow: esta librería esta especializada en el aprendizaje automático.

scikit-learn: esta librería de Python esta especializada en el aprendizaje no supervisado y supervisado.

Plotly: se lo utilizo por ser al igual que matplotlib y seaborn una biblioteca especializada en la visualización de datos.

keras\_preprocessing: se lo utilizo para preparar los datos para el entrenamiento del modelo de aprendizaje profundo.

Ipykernel: permite interactuar al equipo con Jupyter notebook en este caso en particular.

Hara evitar que todas esas librerías interrumpieran el correcto funcionamiento de la computadora personal o PC durante la vida diaria y el tiempo que durase el proyecto se dio la opción de hacerlo en Google Colab o en un entorno virtual de Python, siendo el segundo el que utilice para el proyecto. El entorno se creo mediante el siguiente comando en la terminal `PYTHON -M VENV` seguido del nombre del entorno virtual. Al terminar la creación del entorno instale las librerías con los siguientes comandos en la terminal de VisualStudioCode luego de P

iniciar el entorno virtual ingresando el siguiente comando para evitar errores `Set-ExecutionPolicy Unrestricted -Scope Process` y utilizando el script `actívale.ps1` para entornos creados en



Windows, sistema operativo utilizado para el proyecto.

- pip install pandas
- pip install opencv-python
- pip install matplotlib
- pip install nbformat
- pip install scikit-image
- pip install seaborn
- pip install tensorflow
- pip install -U scikit-learn
- pip install plotly
- pip install ipython
- pip install keras\_preprocessing
- pip install ipykernel

1. FROM: se utiliza para la importación de librerías.

2. IMPORT: al igual que from también se usa para la importación de librerías, el orden en que se usa varia, pero se utilizan ambos.

3. READ\_CSV: esto pertenece a la librería de pandas, se utiliza para leer y manipular datos. Gracias a esta función pandas lee un archivo en una copia y lo manipula sin hacer uso del original.

4. PANDAS: es una librería de análisis y manipulación de datos o información

5. OS.PATH.ABSPATH('DATASET'): esta función se utiliza

para obtener la ruta absoluta de la carpeta o directorio dataset donde se guarda la información que se necesita. almacenar datos de forma ordenada con índices.

7. FOR INDEX IN RANGE(LEN: el bucle for es utilizado para que se repita el código un numero finito de veces. En pocas palabras es una secuencia repetitiva.

salidas por pantalla.

10. BRAIN\_DF.INFO(): esto es una función de pandas y nos permite obtener su información, con su me refiero a los datos a manipular.

11. BRAIN\_DF.HEAD(50): head nos permite imprimir un numero determinado de filas, en este caso 50.

13. ['MASK']: con esto accedemos a las columnas.

14. VALUE\_COUNTS(): lo utilizamos para contar los datos en pandas. Específicamente cuantas veces aparece un dato.

15. SHOW(): muestra los datos por pantalla.

16. MARKER\_LINE\_COLOR: se utiliza para dar color a los gráficos en plotly.

17. PLOTLY: se utiliza para visualizar cosas, específicamente gráficos.

18. GO.BAR: se utiliza para crear un grafico de barras en plotly.

19. GO.FIGURE: se utiliza para crear un objeto para los gráficos.

23. MAX: esto nos permite acceder a los máximos valores de un color en una parte de una imagen, su intensidad.

24. MIN: esto hace lo contrario a lo anterior.

25. TITLE: nos permite poner titulo a los gráficos.

26. PLT.SUBPLOTS: subplots crea subtramas, por ejemplo. 1, 1.2, 1.3, etc.

27. CMAP: pertenece a la librería de matplotlib y se utiliza para darle color al gráfico.

28. MATPLOTLIB: es una librería de visualización de datos.

30. TIGHT\_LAYOUT(): ajusta el espacio entre el grafico y subgraficos.

31. RANDOM.RANDINT: esta función se utiliza para usar una variable aleatoria.

32. SHAPE: indica la cantidad de filas y columnas, pertenece a numpy.

33. NUMPY: esta es una librería de Python que se utiliza para una mayor precisión, por ejemplo, esta librería cuenta con la función pi, la cual cuenta con una mayor precisión que si pusiéramos 3.14.

34. DROP: se utiliza para eliminar filas y columnas específicas.

35. COLUMNS: se utiliza con la función anterior y sirve para especificar las filas y columnas a eliminar.

36. APPLY: se utiliza para convertir números en letras.

## TRABAJO INTEGRADOR MODULO II | Julio Arturo Rodriguez

22

37. TRAIN\_TEST\_SPLIT: sirve para dividir los datos en partes.

38. TEST\_SIZE: especifica la cantidad a dividir o su tamaño.

Ahora que ya entendemos algunos conceptos seguiremos explicando sobre el programa.

```
brain_df.shape
```

utilizamos esto para ver las filas y columnas.

Visualizamos que tenemos solo tres columnas ahora

```
brain_df_train.shape
```

Convertir los datos en la columna de máscara a formato de string, para usar el modo categórico en flow\_from\_dataframe

```
brain_df_train['mask'] = brain_df_train['mask'].apply(lambda x: str(x))
```

en pocas palabras entrega los datos al modelo de machine learning.

# Obtenemos la información del dataframe

```
brain_df_train.info()
```

# Dividimos los datos para el entrenamiento y testing dejando un 20% para testeos y 85% para entrenamiento

```
from sklearn.model_selection import train_test_split
```

```
train, test = train_test_split(brain_df_train, test_size = 0.15)
```

Con esto asignamos la cantidad de datos a entrenar y los datos para testear ,con esto obtenemos una vista en general y chequeamos que todo esté bien.

Ahora que hemos visto una explicacion de los datos del programa.

¿Qué es una clase desbalanceada?

Es facil, una clase desbalanceada hace referencia al dataset, lo ideal para que un programa machine learning funcione es que tenga 50 a 50, mitad verdadero y mitad falso para evitar darle mas importancia a uno de ellos, ejemplo un programa que tiene 100 fotos de perros y un millon de fotos de iguana o gartos u otros animales va a tener un inconveniente que se trata de lo anterior, el programa va a priorizar al que mas tenga, provocando un error de aciertos, eso es una clase desbalanceada, y tiene distintas soluciones como puede ser smote – tomek que no fue el caso para este programa, ¿Qué es? O ¿Qué hace? Es un algoritmo que genera un muestras sinteticas de la clase minoritaria, la que menos datos tiene el dataset.

En nuestro caso utilize oversampling que hace mas muestras de lo mismo.

Mientras que un mapa de calor se utiliza para saber si hay algun inconveniente o que tan bien funciona el programa. Es cierto que habia varias, y la que era la mejor opcion resulto no serlo por eso puse otra matriz de confucion, accurasy.

## Imágenes del programa

## TRABAJO PRACTICO INTEGRADOR

```
#librerias
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow import keras
from keras.layers import Dense
from keras.models import Sequential

#Procesamiento de los datos

#Divide los datos en características (X) y etiquetas (y)
data = pd.read_csv('bandeasdatos.csv')
X = data.drop('Class', axis=1)
y = data['Class']

#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=42)

#Escala las características para normalizar
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

#Creación de la red neuronal y entrenamiento

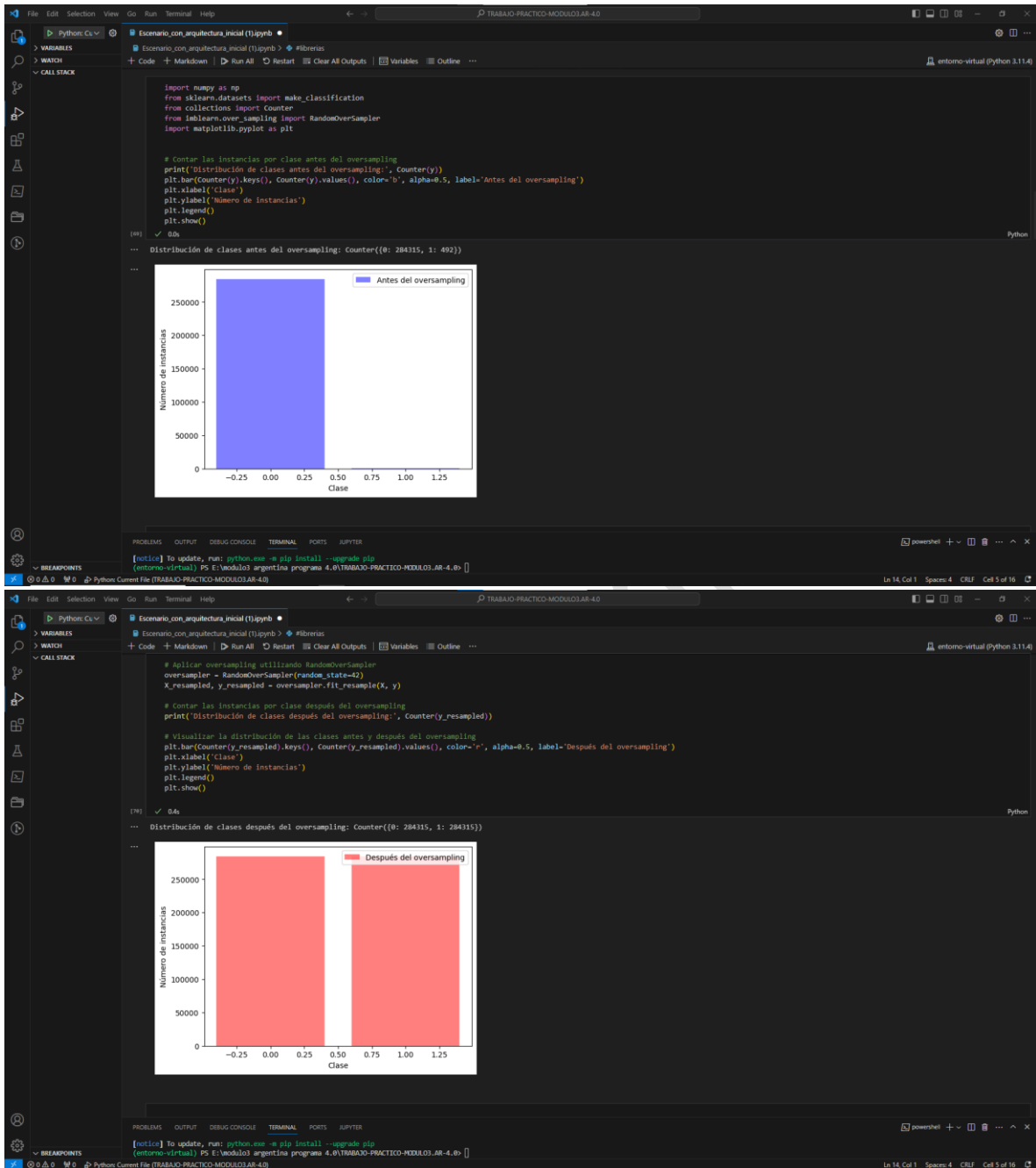
model = Sequential()
#...
model.add(Dense(units=128, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))
#compilar el modelo
model.compile(loss='binary_crossentropy', optimizer='adam')
#Entrenar el modelo
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)

Epoch 1/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0055
Epoch 2/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0032
Epoch 3/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0028
Epoch 4/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0025
Epoch 5/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0023
Epoch 6/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0022
Epoch 7/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0021
Epoch 8/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0021
Epoch 9/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0019
Epoch 10/10
7121/7121 [=====] - 9s 1ms/step - loss: 0.0019

<keras.src.callbacks.history at 0x1aa848500>

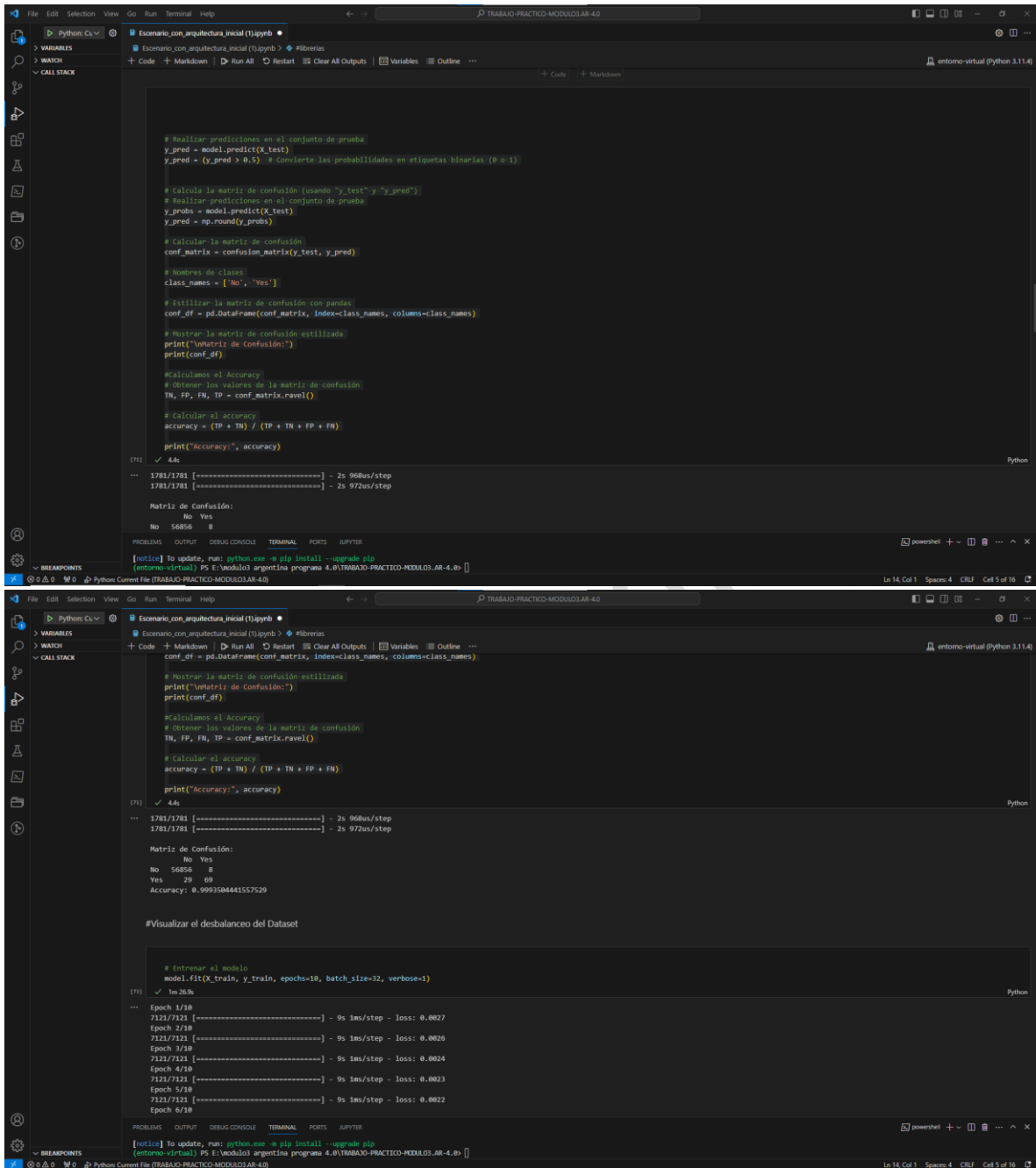
import numpy as np
from sklearn.datasets import make_classification
from collections import Counter
from imblearn.over_sampling import RandomOverSampler
```

# TRABAJO PRACTICO INTEGRADOR





# TRABAJO PRACTICO INTEGRADOR



The image displays two screenshots of a Jupyter Notebook environment within VS Code, showing the implementation of a machine learning model's evaluation and training.

**Top Screenshot:** The code calculates the confusion matrix and accuracy. The output shows the confusion matrix and the calculated accuracy.

```
# Realizar predicciones en el conjunto de prueba
y_pred = model.predict(x_test)
y_pred = (y_pred > 0.5) # convierte las probabilidades en etiquetas binarias (0 o 1)

# Calcula la matriz de confusión (usando "y_test" y "y_pred")
# Realizar predicciones en el conjunto de prueba
y_probs = model.predict(x_test)
y_pred = np.round(y_probs)

# Calcular la matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)

# Nombres de clases
class_names = ['No', 'Yes']

# Estilizar la matriz de confusión con pandas
conf_df = pd.DataFrame(conf_matrix, index=class_names, columns=class_names)

# Mostrar la matriz de confusión estilizada
print("Matriz de Confusión:")
print(conf_df)

# Calculamos el Accuracy
# Obtener los valores de la matriz de confusión
TN, FP, FN, TP = conf_matrix.ravel()

# Calcular el accuracy
accuracy = (TP + TN) / (TP + TN + FP + FN)

print("Accuracy:", accuracy)
```

Output:

```
1781/1781 [=====] - 2s 968us/step
1781/1781 [=====] - 2s 972us/step

Matriz de Confusión:
      No  Yes
No  56856   8
Yes     29   69

Accuracy: 0.9993584441557529
```

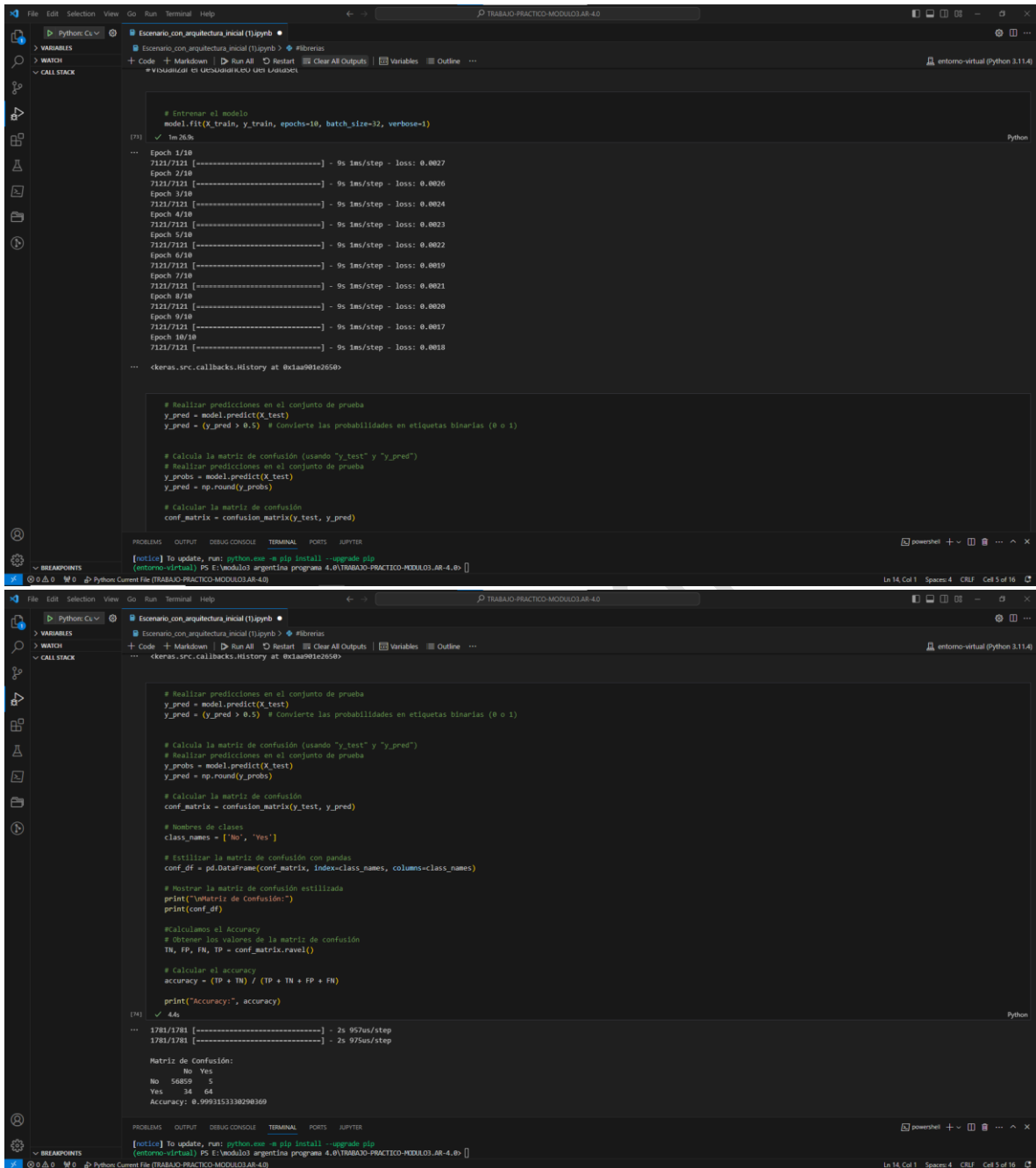
**Bottom Screenshot:** The code trains the model. The output shows the training progress and loss.

```
# Entrenar el modelo
model.fit(x_train, y_train, epochs=10, batch_size=32, verbose=1)
```

Output:

```
Epoch 1/10
7121/7121 [=====] - 0s 1ms/step - loss: 0.0027
Epoch 2/10
7121/7121 [=====] - 0s 1ms/step - loss: 0.0026
Epoch 3/10
7121/7121 [=====] - 0s 1ms/step - loss: 0.0024
Epoch 4/10
7121/7121 [=====] - 0s 1ms/step - loss: 0.0023
Epoch 5/10
7121/7121 [=====] - 0s 1ms/step - loss: 0.0022
Epoch 6/10
```

# TRABAJO PRACTICO INTEGRADOR



```
# Entrenar el modelo
model.fit(x_train, y_train, epochs=10, batch_size=32, verbose=1)

Epoch 1/10
7121/7121 [-----] - 9s 1ms/step - loss: 0.0027
Epoch 2/10
7121/7121 [-----] - 9s 1ms/step - loss: 0.0026
Epoch 3/10
7121/7121 [-----] - 9s 1ms/step - loss: 0.0024
Epoch 4/10
7121/7121 [-----] - 9s 1ms/step - loss: 0.0023
Epoch 5/10
7121/7121 [-----] - 9s 1ms/step - loss: 0.0022
Epoch 6/10
7121/7121 [-----] - 9s 1ms/step - loss: 0.0019
Epoch 7/10
7121/7121 [-----] - 9s 1ms/step - loss: 0.0021
Epoch 8/10
7121/7121 [-----] - 9s 1ms/step - loss: 0.0020
Epoch 9/10
7121/7121 [-----] - 9s 1ms/step - loss: 0.0017
Epoch 10/10
7121/7121 [-----] - 9s 1ms/step - loss: 0.0018

<keras.src.callbacks.History at 0x1a901e2650>

# Realizar predicciones en el conjunto de prueba
y_pred = model.predict(x_test)
y_pred = (y_pred > 0.5) # Convierte las probabilidades en etiquetas binarias (0 o 1)

# Calcula la matriz de confusión (usando "y_test" y "y_pred")
# Realizar predicciones en el conjunto de prueba
y_probs = model.predict(x_test)
y_pred = np.round(y_probs)

# Calcular la matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)

[notice] to update, run: python.exe -m pip install --upgrade pip
(entorno-virtual) PS E:\Modulo3\argentina programa 4.0\TRABAJO-PRACTICO-MODULO3-AR-4.0>

# Realizar predicciones en el conjunto de prueba
y_pred = model.predict(x_test)
y_pred = (y_pred > 0.5) # Convierte las probabilidades en etiquetas binarias (0 o 1)

# Calcula la matriz de confusión (usando "y_test" y "y_pred")
# Realizar predicciones en el conjunto de prueba
y_probs = model.predict(x_test)
y_pred = np.round(y_probs)

# Calcular la matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)

# Nombres de clases
class_names = ["No", "Yes"]

# Estilizar la matriz de confusión con pandas
conf_df = pd.DataFrame(conf_matrix, index=class_names, columns=class_names)

# Mostrar la matriz de confusión estilizada
print("\nMatriz de Confusión:")
print(conf_df)

# Calculamos el Accuracy
# Obtener los valores de la matriz de confusión
TN, FP, FN, TP = conf_matrix.ravel()

# Calcular el accuracy
accuracy = (TP + TN) / (TP + TN + FP + FN)

print("Accuracy:", accuracy)

1781/1781 [-----] - 2s 957ms/step
1781/1781 [-----] - 2s 975ms/step

Matriz de Confusión:
      No  Yes
No  56859   5
Yes   34   64
Accuracy: 0.9993153338298369

[notice] to update, run: python.exe -m pip install --upgrade pip
(entorno-virtual) PS E:\Modulo3\argentina programa 4.0\TRABAJO-PRACTICO-MODULO3-AR-4.0>
```

## TRABAJO PRACTICO INTEGRADOR

Python: Clave

Escenario con arquitectura inicial (1).pyb

Escenario con arquitectura inicial (1).pyb > #librerias

+ Code + Markdown Run All Restart Clear All Outputs Variables Outline

entorno-virtual (Python 3.11.4)

```
# Calcular la accuracy
accuracy = (TP + TN) / (TP + TN + FP + FN)
print("Accuracy:", accuracy)
```

✓ 4s

1781/1781 [=====] - 2s 957ms/step  
1781/1781 [=====] - 2s 975ms/step

Matriz de Confusión:

	No	Yes
No	56859	5
Yes	34	64

Accuracy: 0.999353338298369

```
import matplotlib.pyplot as plt


# Calcular la cantidad de muestras por clase
class_counts = data['class'].value_counts()

# Mostrar la cantidad de datos por clase
for class_name, count in class_counts.items():
    print(f'Clase: {class_name}, Cantidad de Datos: {count}')

# Crear el gráfico de barras
plt.figure(figsize=(8, 6))
class_counts.plot(kind='bar', color=['blue', 'red'])
plt.title('Desbalance de Clases')
plt.xlabel('Clase')
plt.ylabel('Cantidad de Muestras')
plt.xticks([0, 1], ['Normal', 'Fraud'])
plt.show()
```

✓ 0.0s

Clase: 0, Cantidad de Datos: 284315  
Clase: 1, Cantidad de Datos: 492



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

[notice] To update, run: python.exe -m pip install --upgrade pip  
(entorno-virtual) PS E:\Modulo3\argentina programa 4.0\TRABAJO-PRACTICO-MODULO3-AR-4.0>

Ln 6, Col 33 Spaces: 4 CRLF Cell 5 of 16

---

Python: Clave

Escenario con arquitectura inicial (1).pyb

Escenario con arquitectura inicial (1).pyb > #librerias

+ Code + Markdown Run All Restart Clear All Outputs Variables Outline

entorno-virtual (Python 3.11.4)

```
import matplotlib.pyplot as plt

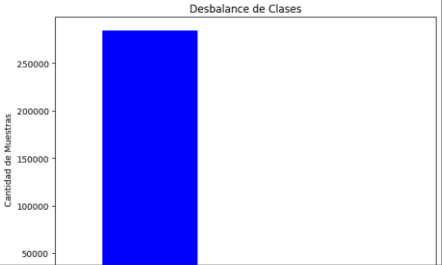
# Calcular la cantidad de muestras por clase
class_counts = data['class'].value_counts()

# Mostrar la cantidad de datos por clase
for class_name, count in class_counts.items():
    print(f'Clase: {class_name}, Cantidad de Datos: {count}')

# Crear el gráfico de barras
plt.figure(figsize=(8, 6))
class_counts.plot(kind='bar', color=['blue', 'red'])
plt.title('Desbalance de Clases')
plt.xlabel('Clase')
plt.ylabel('Cantidad de Muestras')
plt.xticks([0, 1], ['Normal', 'Fraud'])
plt.show()
```

✓ 0.0s

Clase: 0, Cantidad de Datos: 284315  
Clase: 1, Cantidad de Datos: 492

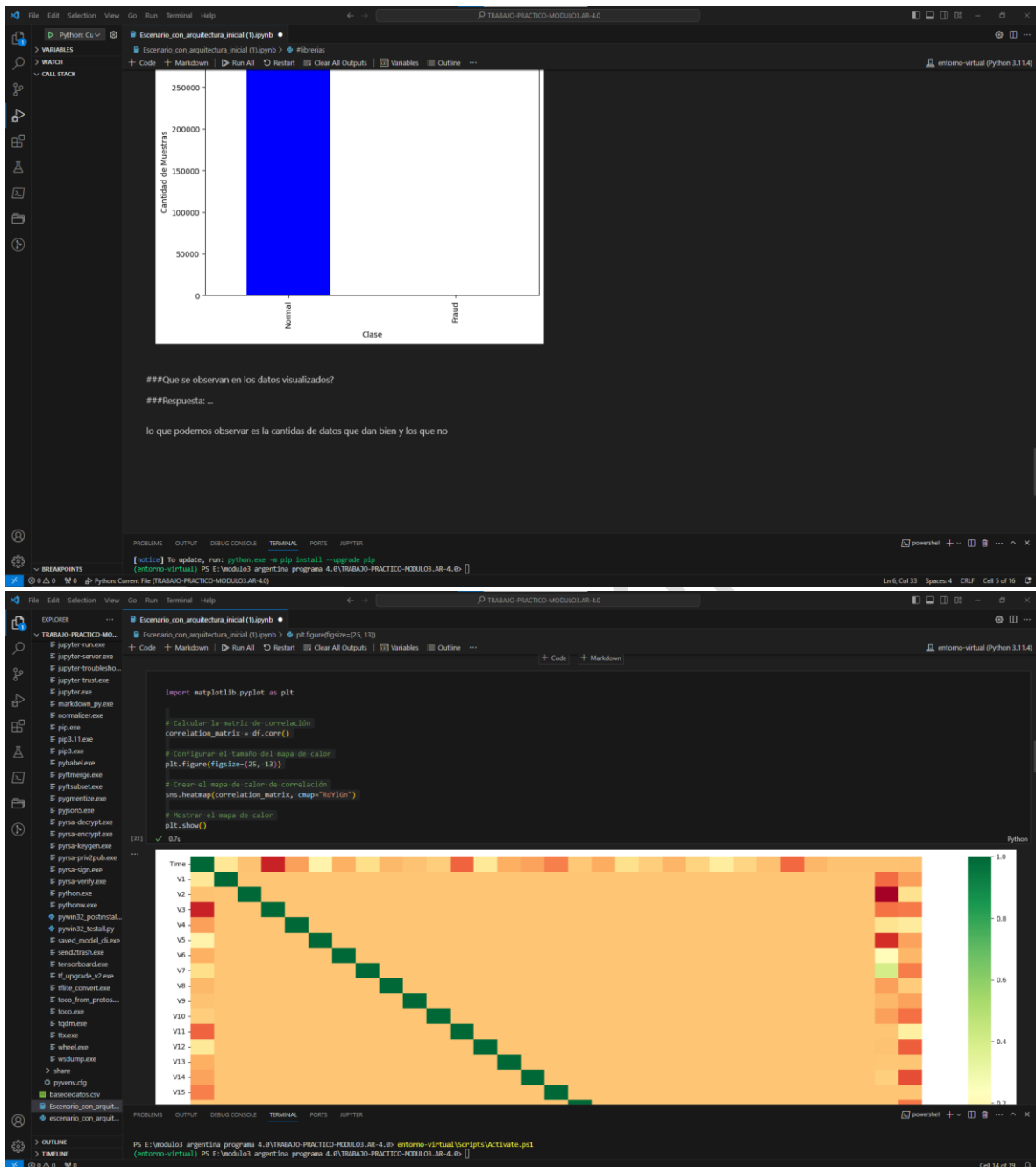


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

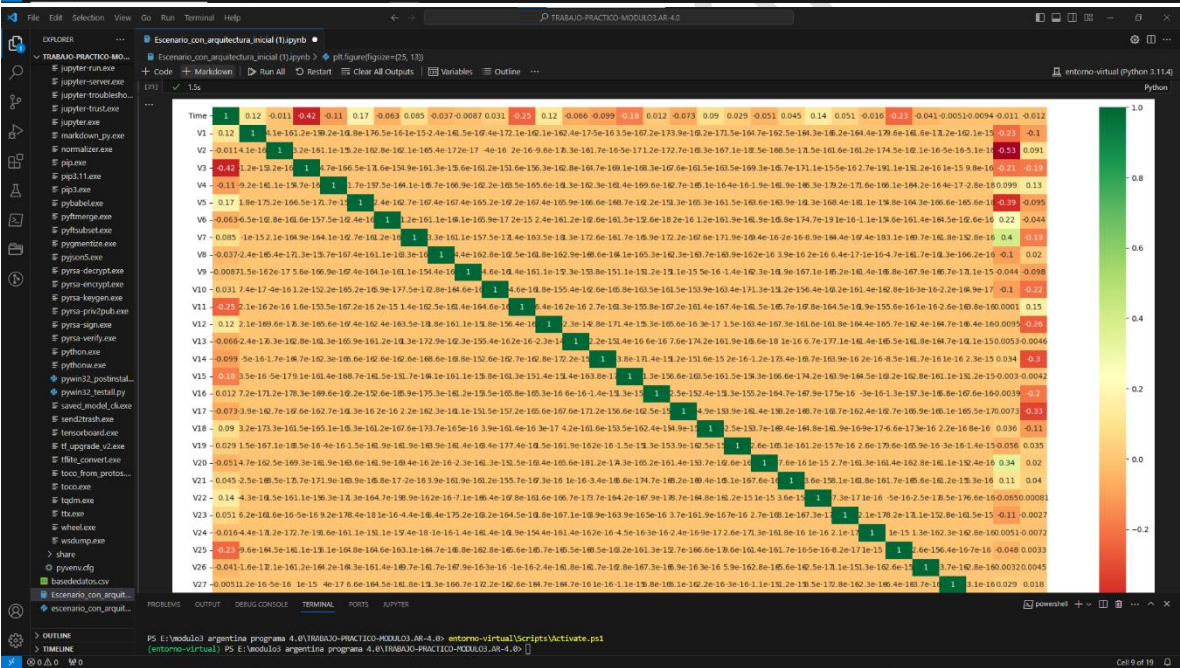
[notice] To update, run: python.exe -m pip install --upgrade pip  
(entorno-virtual) PS E:\Modulo3\argentina programa 4.0\TRABAJO-PRACTICO-MODULO3-AR-4.0>

Ln 6, Col 33 Spaces: 4 CRLF Cell 5 of 16

## TRABAJO PRACTICO INTEGRADOR



TRABAJO PRACTICO INTEGRADOR



Dado que la mayor parte del código es la misma que se utilizó en el segundo módulo pero se vio con mayor profundidad y algunos conceptos nuevos, pondré una explicación de los ya utilizados y explicaré los nuevos adquiridos en este módulo.

## Conclusión

Para concluir se puede decir que el trabajo esta terminado, pero dado que no llego a 1 del todo en la matriz de confucion se puede decir que el mismo aun gosa de posibles mejoras a futuro hasta alcanzarlo.

## Referencias

Youtube [https://www.youtube.com/watch?v=3zBzV\\_h2gkY&list=LL&index=21&t=4792s](https://www.youtube.com/watch?v=3zBzV_h2gkY&list=LL&index=21&t=4792s)

Youtube <https://www.youtube.com/watch?v=B6STReqPpVc&list=LL&index=27&t=2727s>

Youtube <https://www.youtube.com/watch?v=t-bcr6MKPJU&list=LL&index=23>

Youtube <https://www.youtube.com/watch?v=pwrc4vgBdE0&list=LL&index=26&t=2423s>

Youtube [https://www.youtube.com/watch?v=Rgx\\_lkkD\\_jA&list=LL&index=22&t=2822s](https://www.youtube.com/watch?v=Rgx_lkkD_jA&list=LL&index=22&t=2822s)