TRABAJO PRACTICO N°3 – MACHINE LEARNING

ORGANIZACIÓN DE DATOS 75.06

Alumno: Gonzalez, Agustin Nicolas

Padrón: 106086

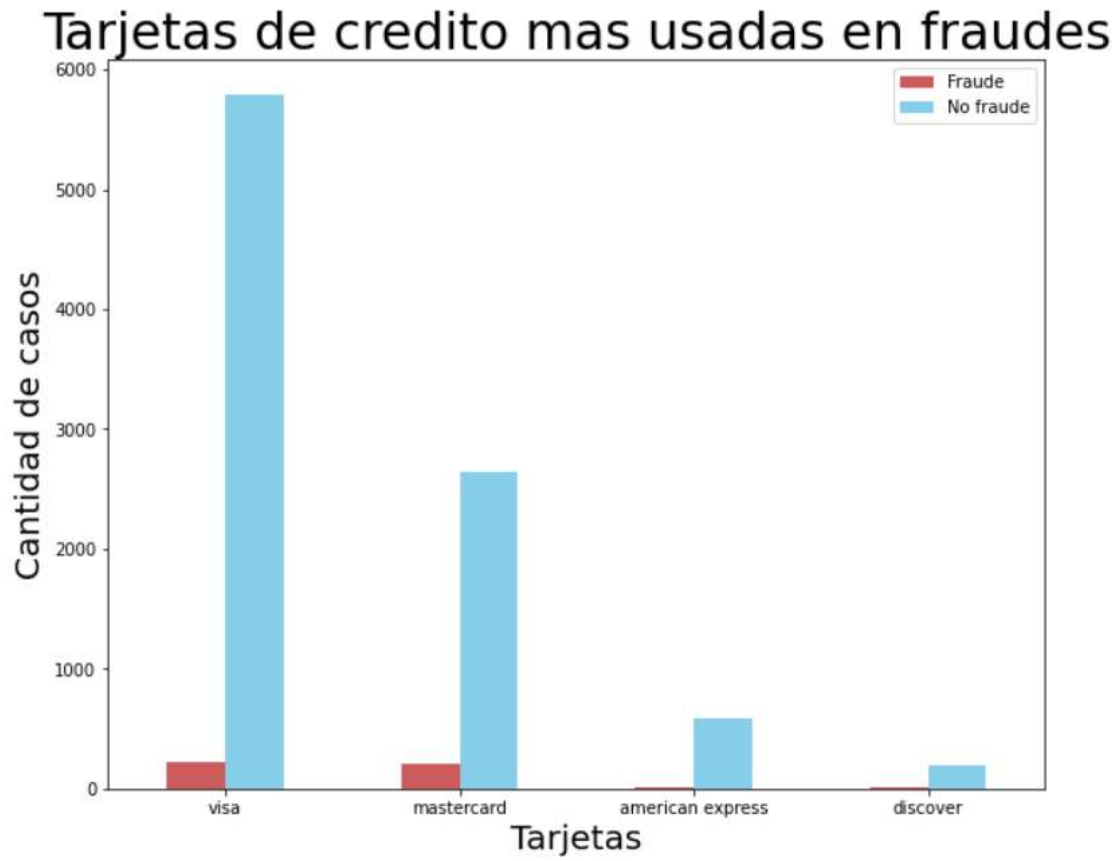
Corrector: Damian.

Link al colab:

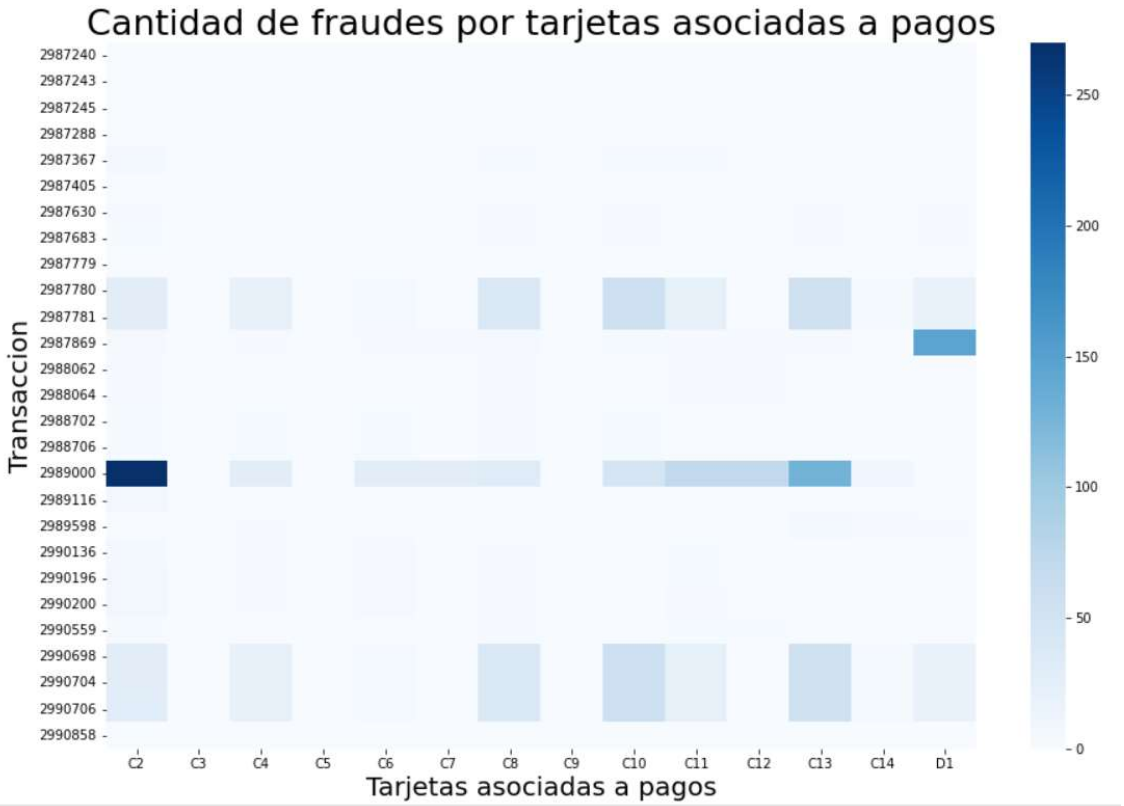
Parte I y II:

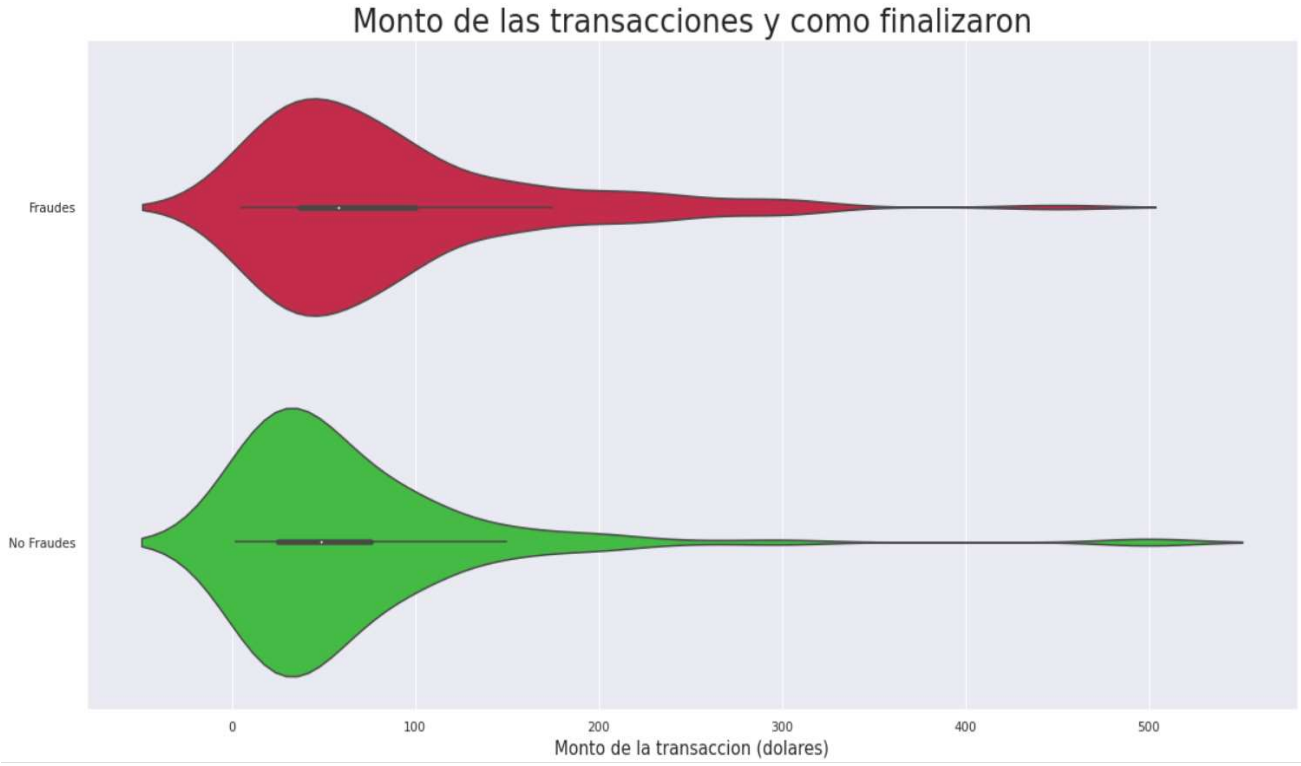
https://colab.research.google.com/drive/1695YCeom45TnhvPfZAKcbguB68gNx3vB?usp=sharing

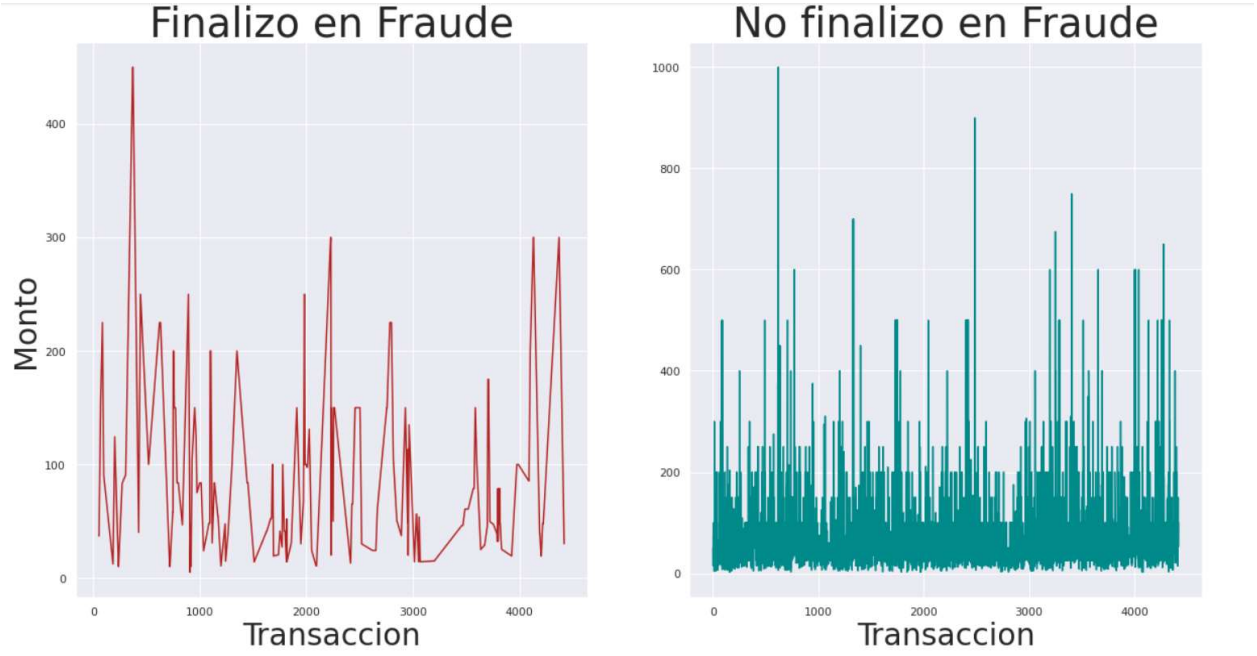
Parte III:

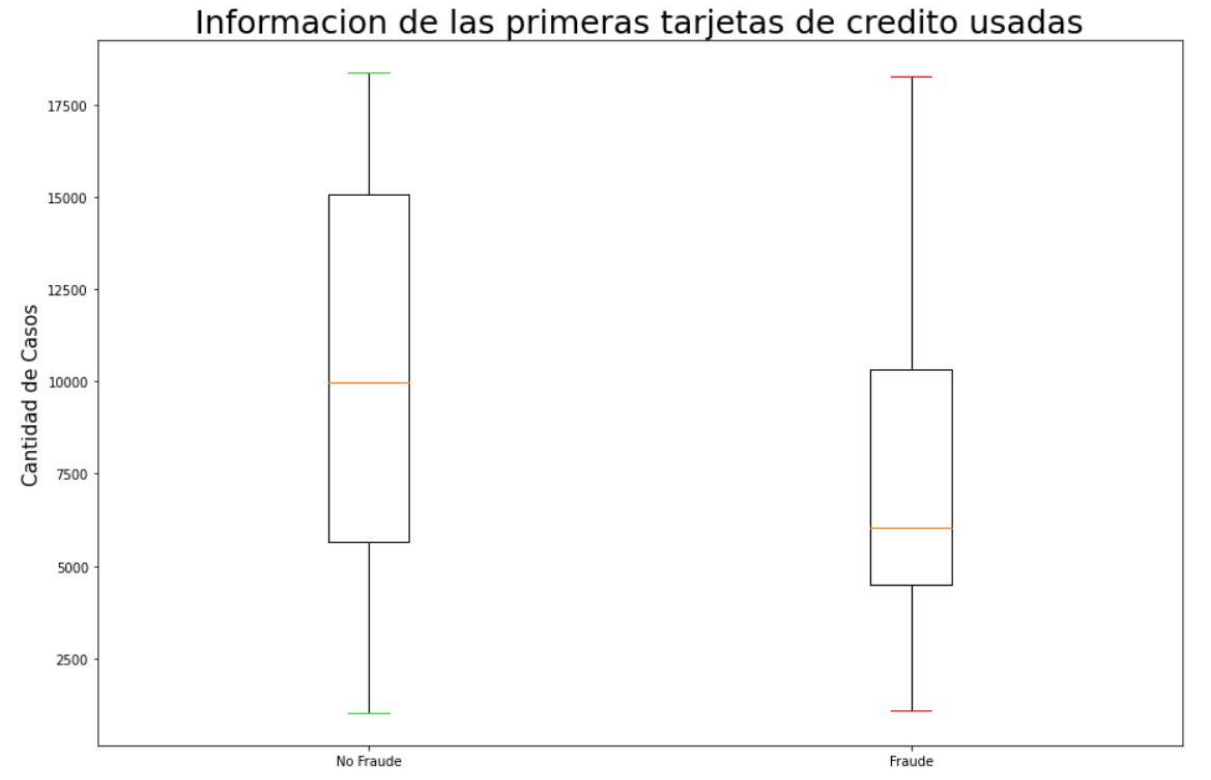

Tarjetas de credito mas usadas en fraudes
Fraude
No fraude
Cantidad de casos
0
1000
2000
3000
4000
5000
6000
visa
mastercard
american express
discover
Tarjetas


Productos mas victimas de fraudes
Fraude
No fraude
Cantidad de casos
0
500
1000
1500
2000
2500
3000
3500
4000
C
H
R
S
Producto


Cantidad de fraudes por tarjetas asociadas a pagos
Transaccion
2987240
2987243
2987245
2987288
2987367
2987405
2987630
2987683
2987779
2987780
2987781
2987869
2988062
2988064
2988702
2988706
2989000
2989116
2989598
2990136
2990196
2990200
2990559
2990698
2990704
2990706
2990858
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C12
C13
C14
D1
Tarjetas asociadas a pagos
250
200
150
100
50
0


Monto de las transacciones y como finalizaron
Fraudes
No Fraudes
0
100
200
300
400
500
Monto de la transaccion (dolares)


Finalizo en Fraude
Monto
Transaccion
No finalizo en Fraude
Transaccion


Informacion de las primeras tarjetas de credito usadas
Cantidad de Casos
No Fraude
Fraude

PARTE 2

```
#PARTE 2
import pandas as pd
from sklearn import svm
from sklearn import datasets
url = '/content/train_identity.csv'
identidad = pd.read_csv(url)
url = '/content/train_transaction.csv'
transaccion = pd.read_csv(url)
joineado = transaccion.merge(identidad, how="outer")
filtrado = joineado.iloc[:,1:]
joineado = None
identidad = None
transaccion = None
```

```
[ ] from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LogisticRegression
    from sklearn import metrics
    claves_mean = {}
    for i in filtrado:
      if i=="isFraud": continue
      try:
        int(filtrado[i][0])
      except:
        mean = filtrado.groupby([i])["isFraud"].mean().to_dict()
        claves_mean[i] = mean
        filtrado[i] = filtrado[i].map(mean)
    filtrado = filtrado.fillna(0)
```

```
[ ] def split_train_y_test(df):
      train = df.loc[df["TransactionDT"]<12000000,:]
      test = df.loc[df["TransactionDT"]>12000000,:]
      x_train = train.iloc[:,1:]
      y_train = train.iloc[:,0]
      x_test = test.iloc[:,1:]
      y_test = test.iloc[:,0]
      return x_train, y_train, x_test, y_test
```

```
#Aca hago mi propio train_test_split,
#separando en una proporcion de 80-20 aproximadamente
#Es mejor forma para obtener el dataset de validacion
#que haciendo un train_test_split, ya que tomando como
#referencia el transactionDT puedo crear una especie
#de relacion entre las transacciones "pasadas" y "futuras"
#y basicamente mas util para mi propio modelo
```

```
[ ] x_train, y_train, x_test, y_test = split_train_y_test(filtrado)
```

```
[ ] import numpy as np #Aca realizo el tuneo de hyperparametros usando un randomized search
    model_val = LogisticRegression()
    param_grid = [{"penalty": ["l1", "l2", "elasticnet", "none"],
                   "C": np.logspace(-4,4,20),
                   "solver": ["lbfgs", "newton-cg", "liblinear", "sag", "saga"],
                   "max_iter": [100, 200]}] #(Aclaracion: Como este era solo un baseline, deje los numeros del max_iter bajos,
                                            #lo mas conveniente para el problema era poner numeros mas altos pero se demoraba
                                            #demasiado y se reiniciaba el colab)
```

```
[ ] from sklearn.model_selection import RandomizedSearchCV
    model = LogisticRegression()
    rand = RandomizedSearchCV(model, param_grid, n_iter=3, n_jobs=1, cv=3, scoring="accuracy", random_state=1)
```

```
[ ] rand.fit(x_train,y_train)
```

```
[ ] rand.best_params_ #Estos son los mejores parametros que encontro
```

```
{'C': 0.012742749857031334,
 'max_iter': 200,
 'penalty': 'none',
 'solver': 'lbfgs'}
```

```
best_model = rand.best_estimator_
#Validacion
```

```
metrics.roc_auc_score(y_train, best_model.predict_proba(x_train)[:,1]) #prediccion con el train
```

```
0.6674520961586159
```

```
metrics.roc_auc_score(y_test, best_model.predict_proba(x_test)[:,1]) #prediccion con el test
```

```
0.6819060619144489
```

```
[ ] #Aca empiezo para hacer la prediccion para la competencia
    x = filtrado.iloc[:,1:]
    y = filtrado.iloc[:,0]
```

```
[ ] best_model.fit(x.values,y.values)
```

```
[ ] url = '/content/drive/MyDrive/Colab Notebooks/test_identity.csv'
    identidad_test = pd.read_csv(url)
    url = '/content/drive/MyDrive/Colab Notebooks/test_transaction.csv'
    transaccion_test = pd.read_csv(url)
    joineado_test = transaccion_test.merge(identidad_test, how="outer")
    filtrado_test = joineado_test.iloc[:,1:]
    identidad_test = None
    transaccion_test = None
```

```
[ ] def renombrar_columna(col):
      nombre = ""
      for letra in col:
        if letra == "-":
          nombre += "_"
        else:
          nombre += letra
      return nombre

    def renombrar_columnas(df):
      for col in df:
        if "id" in col:
          nuevo = renombrar_columna(col)
          df.rename(columns={col:nuevo}, inplace=True)
```

```
[ ] renombrar_columnas(filtrado_test) #Esto es un paso extra para renombrar las columnas de nombre diferente
```

```
for i in filtrado_test:
  try:
    int(filtrado_test[i][0])
  except:
    filtrado_test[i] = filtrado_test[i].map(claves_mean[i])
#Uso el diccionario que cree la primera vez que hice el mean encode
filtrado_test = filtrado_test.fillna(0)
```

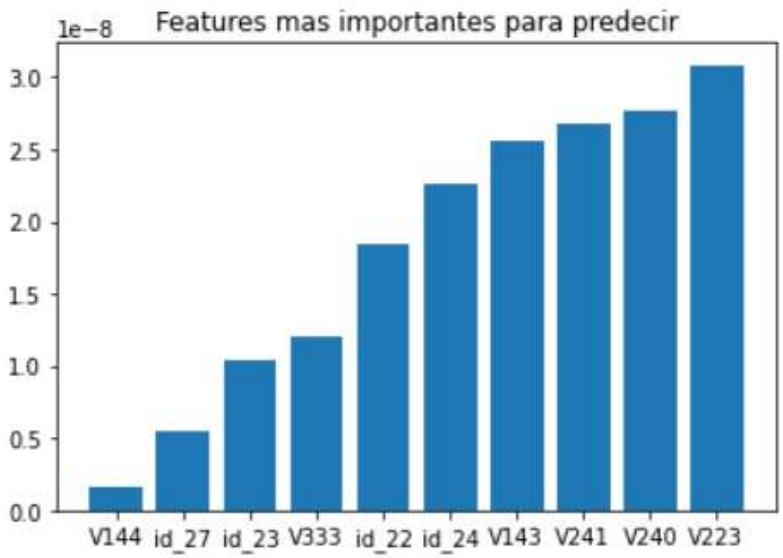
```
[ ] csv_a_guardar = pd.DataFrame({"TransactionID": joineado_test.iloc[:,0], "isFraud": best_model.predict(filtrado_test)})
    csv_a_guardar = csv_a_guardar.set_index("TransactionID")
```

```
[ ] len(csv_a_guardar)
```

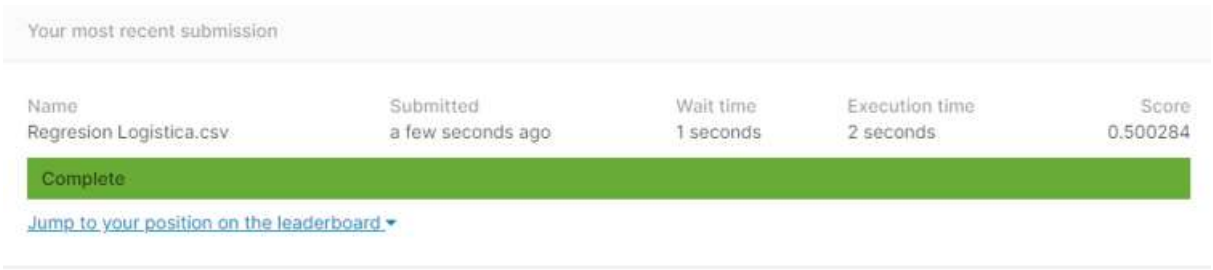
```
506691
```

```
[ ] from google.colab import files
    csv_a_guardar.to_csv("Regresion Logistica")
    files.download("Regresion Logistica")
```

```
[ ]  #Features mas importantes a predecir
     from matplotlib import pyplot
     importancia = best_model.coef_
     df = pd.DataFrame({"feature": x.columns, "coef": importancia[0]})
     nuevo = df[df["coef"]>0]
     nuevo = nuevo.sort_values("coef").head(10)
     pyplot.bar(nuevo["feature"], nuevo["coef"])
     pyplot.title("Features mas importantes para predecir")
```



Score Competencia:



PARTE 3

```
#PARTE 3
import pandas as pd
from sklearn import svm
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import nltk
from sklearn.feature_extraction.text import CountVectorizer
!pip install nltk
nltk.download('stopwords')

url = '/content/drive/MyDrive/Colab Notebooks/train_identity.csv'
identidad = pd.read_csv(url)
url = '/content/drive/MyDrive/Colab Notebooks/train_transaction.csv'
transaccion = pd.read_csv(url)
joineado = transaccion.merge(identidad, how="outer")
filtrado = joineado.iloc[:,1:]
joineado = None
identidad = None
transaccion = None
```

```
[ ] #Utilizacion de countvectorizer para vectorizar la columna id_31 que contiene el navegador
    explorador = filtrado["id_31"].fillna(" ")
    vectorizer = CountVectorizer(lowercase=True, stop_words=nltk.corpus.stopwords.words('spanish'), max_features=30000)
    vectorizer.fit(explorador)
    bag_of_words = vectorizer.transform(explorador)
    nuevo_feature = bag_of_words.toarray()
    filtrado["id_31"] = nuevo_feature
    nuevo_feature = None
    bag_of_words = None
```

```
[ ] #One hot encoding para variables categoricas con poca cardinalidad
    dummies = pd.get_dummies(filtrado["ProductCD"], prefix="ProductCD")
    filtrado = pd.concat([filtrado, dummies], axis=1)
    dummies = pd.get_dummies(filtrado["DeviceType"], prefix="DeviceType")
    filtrado = pd.concat([filtrado, dummies], axis=1)
    dummies = pd.get_dummies(filtrado["P_emaildomain"], prefix="P_emaildomain")
    filtrado = pd.concat([filtrado, dummies], axis=1)
    dummies = pd.get_dummies(filtrado["card4"], prefix="card4")
    filtrado = pd.concat([filtrado, dummies], axis=1)
    dummies = pd.get_dummies(filtrado["card6"], prefix="card6")
    filtrado = pd.concat([filtrado, dummies], axis=1)
    dummies = pd.get_dummies(filtrado["id_15"], prefix="id_15")
    filtrado = pd.concat([filtrado, dummies], axis=1)
    dummies = pd.get_dummies(filtrado["id_16"], prefix="id_6")
    filtrado = pd.concat([filtrado, dummies], axis=1)
    filtrado = filtrado.drop(["ProductCD", "card4", "card6", "id_15", "id_16", "DeviceInfo", "P_emaildomain"], axis=1)
```

```
#Mean encoding para el resto de variables categoricas que poseen una alta cardinalidad
claves_mean = {}
for i in filtrado:
  if i=="isFraud": continue
  try:
    int(filtrado[i][0])
  except:
    mean = filtrado.groupby([i])["isFraud"].mean().to_dict()
    claves_mean[i] = mean
    filtrado[i] = filtrado[i].map(mean)
filtrado = filtrado.fillna(0)
```

```
[ ]  def split_train_y_test(df):                                    #Aca hago m
       train = df.loc[df["TransactionDT"]<12000000,:]
       test = df.loc[df["TransactionDT"]>12000000,:]
       x_train = train.iloc[:,1:]
       y_train = train.iloc[:,0]
       x_test = test.iloc[:,1:]
       y_test = test.iloc[:,0]
       return x_train, y_train, x_test, y_test
```

```
[ ]  x_train, y_train, x_test, y_test = split_train_y_test(filtrado)
```

```
]  #gridesearch para arbol de decision
   import pandas as pd
   import numpy as np
   from sklearn import svm
   from sklearn import datasets
   from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV, cross_val_score
   from sklearn import metrics, preprocessing, tree
   from sklearn.metrics import f1_score, make_scorer
```

```
 ]  from sklearn.tree import DecisionTreeClassifier as dt
    model = dt()
```

```
[ ]  #Validacion
     parameters = {"max_depth": [1,2,3,4,5], "min_samples_leaf": [1,2,3,4,5], "min_samples_split": [2,3,4,5], "criterion": ["gini", "entropy"]}
     search_obj = GridSearchCV(model, parameters, cv=5, scoring="f1_macro")
     fit_obj = search_obj.fit(x_train, y_train)
     best_model = fit_obj.best_estimator_
```

```
[ ] best_model.get_params()
```

```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'entropy',
 'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

```
[ ] metrics.roc_auc_score(y_train, best_model.predict_proba(x_train)[:,1]) #prediccion con el train (logicamente con el train se obtiene un mejor score)
```

```
0.8849867599969774
```

```
[ ] metrics.roc_auc_score(y_test, best_model.predict_proba(x_test)[:,1]) #prediccion con el test
```

```
0.8690681275440714
```

```
x_train = None
y_train = None    #Estos None estan simplemente para liberar memoria
x_test = None
y_test = None
x = filtrado.iloc[:,1:]
y = filtrado.iloc[:,0]
filtrado = None
best_model.fit(x.values,y.values)
x = None
y = None
```

```
[ ] #Aca creo el csv para la competencia
    url = '/content/drive/MyDrive/Colab Notebooks/test_identity.csv'
    identidad_test = pd.read_csv(url)
    url = '/content/drive/MyDrive/Colab Notebooks/test_transaction.csv'
    transaccion_test = pd.read_csv(url)
    joineado_test = transaccion_test.merge(identidad_test, how="outer")
    filtrado_test = joineado_test.iloc[:,1:]
    identidad_test = None
    transaccion_test = None
```

```
[ ] ids = joineado_test.iloc[:,0]
    joineado_test = None
```

```
[ ] explorador = filtrado_test["id-31"].fillna(" ")
    vectorizer = CountVectorizer(lowercase=True, stop_words=nltk.corpus.stopwords.words('spanish'), max_features=30000)
    vectorizer.fit(explorador)
    bag_of_words = vectorizer.transform(explorador)
    nuevo_feature = bag_of_words.toarray()
    filtrado_test["id-31"] = nuevo_feature
```

```
[ ]  def renombrar_columna(col):
       nombre = ""
       for letra in col:
         if letra == "-":
           nombre += "_"
         else:
           nombre += letra
       return nombre

     def renombrar_columnas(df):
       for col in df:
         if "id" in col:
           nuevo = renombrar_columna(col)
           df.rename(columns={col:nuevo}, inplace=True)

     renombrar_columnas(filtrado_test)
```

```
[ ]  dummies = pd.get_dummies(filtrado_test["ProductCD"], prefix="ProductCD")
     filtrado_test = pd.concat([filtrado_test, dummies], axis=1)
     dummies = pd.get_dummies(filtrado_test["DeviceType"], prefix="DeviceType")
     filtrado_test = pd.concat([filtrado_test, dummies], axis=1)
     dummies = pd.get_dummies(filtrado_test["P_emaildomain"], prefix="P_emaildomain")
     filtrado_test = pd.concat([filtrado_test, dummies], axis=1)
     dummies = pd.get_dummies(filtrado_test["card4"], prefix="card4")
     filtrado_test = pd.concat([filtrado_test, dummies], axis=1)
     dummies = pd.get_dummies(filtrado_test["card6"], prefix="card6")
     filtrado_test = pd.concat([filtrado_test, dummies], axis=1)
     dummies = pd.get_dummies(filtrado_test["id_15"], prefix="id_15")
     filtrado_test = pd.concat([filtrado_test, dummies], axis=1)
     dummies = pd.get_dummies(filtrado_test["id_16"], prefix="id_6")
     filtrado_test = pd.concat([filtrado_test, dummies], axis=1)
     filtrado_test = filtrado_test.drop(["ProductCD", "card4", "card6", "id_15", "id_16", "DeviceInfo", "P_emaildomain"], axis=1)
```

```
[ ]  for i in filtrado_test:
       try:
         int(filtrado_test[i][0])
       except:
         filtrado_test[i] = filtrado_test[i].map(claves_mean[i])

     filtrado_test = filtrado_test.fillna(0)
```

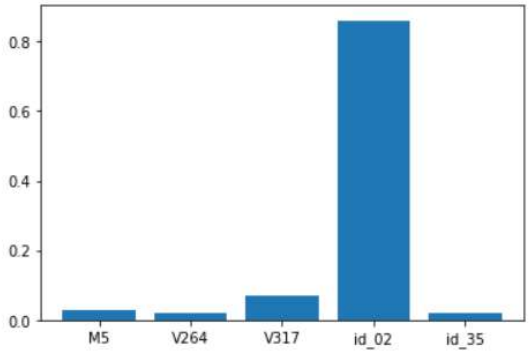
```
     prediccion = best_model.predict(filtrado_test)
```

```
[ ]  csv_a_guardar = pd.DataFrame({"TransactionID": ids, "isFraud": prediccion})
     csv_a_guardar = csv_a_guardar.set_index("TransactionID")
```

```
[ ]  from google.colab import files
     csv_a_guardar.to_csv("Arbol")
     files.download("Arbol")
```

```
[ ]  #Aca reviso la importancia de los feature en el arbol
     import matplotlib.pyplot as plt
     df = pd.DataFrame({"Feature": filtrado_test.columns, "Importancia": best_model.feature_importances_})
     df = df.loc[df["Importancia"]!=0,:]
     plt.bar(df["Feature"], df["Importancia"])
```

```
<BarContainer object of 5 artists>
```




```
#tuneo para xgboost
import pandas as pd
import numpy as np
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
import xgboost
```

```
[ ]  params = {'max_depth': [3, 5, 6, 10],
               'learning_rate': [0.01, 0.1, 0.2, 0.3],
               'submuestra': np.arange (0.5, 1.0, 0.1),
               'n_estimators': [100, 500, 1000]}

     xgbr = xgboost.XGBRegressor(semilla = 20)

     clf = RandomizedSearchCV(estimator = xgbr,
                              param_distributions = params,
                              scoring = 'neg_mean_squared_error',
                              n_iter = 3,
                              verbose = 1, n_jobs=1)

     fit_obj = clf.fit(x_train, y_train)
```

```
[ ]  params = { 'max_depth': [3,6,10],
                'learning_rate': [0.01, 0.05, 0.1],
                'n_estimators': [100, 500],
                'colsample_bytree': [0.3, 0.7]}
     xgbr = xgboost.XGBRegressor()
     clf = GridSearchCV(estimator=xgbr,
                        param_grid=params,
                        scoring='neg_mean_squared_error',
                        verbose=100)

     fit_obj = clf.fit(x_train, y_train)
```

Aca quiero aclarar que probe estos dos algoritmos para buscar hyperparametros para este caso, probe también modificando sus valores pero ninguno llego a funcionar, quedaban horas corriendo hasta que se cerraba el colab, el que mas aguantaba era el randomized pero tampoco terminaba, duro hasta 8 horas

Cuando probe poniendo a mano todos los hyperparametros lo máximo que alcance como score de validación fue 0.70, por lo que decidí intrascendete exponerlo.

Score Competencia: