U1_AB_CARM

May 19, 2023

- Introducción: La presente libreta tiene como objetivo mostrar la implementación de diferentes algoritmos de Aprendizaje Supervisado y no Supervisado aplicados a la detección de anomalías. Particularmente, en este caso haremos uso del dataset de Kaggle: Credit Card Fraud Detection [1].

1. Carga de la librerias y la información.

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from  functools import reduce

# Data preprocessing
from sklearn.feature_selection import SelectKBest, chi2

# Modeling
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import classification_report
```

1.1 Descripción y observación de los datos.

- La siguiente función tiene como objetivo permitir al usuario tener un resumen descriptivo del dataframe seleccionado, así como la observación de algunas de las entradas del mismo.

[ ]:
```
# Función para leer y hacer una primera revisión de los datos
def read_and_interpret_data(path = '', filename = ''):

    df = pd.read_csv('{0}{1}'.format(path,filename))
    print("~~~~~~Perfil de los datos~~~~~~ : ",df.shape)
    print("~~~~~~Columnas con su datatype~~~~~~ : ")
    print(df.info())
    print("~~~~~~Primeros datos~~~~~~ : ")
    print(df.head())
    print("~~~~~~Últimos datos~~~~~~ : ")
    print(df.tail())
    print("~~~~~~Descripción de los datos~~~~~~ : ")
    print(df.describe())
    print("~~~~~~Datos faltantes~~~~~~ : ")
```

```
    print(df.isna().sum())
    if 'timestamp' in df.columns:
        df['timestamp'] = pd.to_datetime(df['timestamp'],format = "%Y-%m-%d %H:
↪%M:%S")
        print("~~~~~~Años de los datos~~~~~~ :")
        print(df.timestamp.dt.year.unique())
    return df
```

[ ]:
```
df_train_id = read_and_interpret_data('','creditcard.csv')
```

```
~~~~~~Perfil de los datos~~~~~~ :  (284807, 31)
~~~~~~Columnas con su datatype~~~~~~ :
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
```

```
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
None
~~~~~~Primeros datos~~~~~~ :
   Time        V1        V2        V3        V4        V5        V6        V7
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599  \
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  …       V21       V22       V23       V24       V25
0  0.098698  0.363787  … -0.018307  0.277838 -0.110474  0.066928  0.128539  \
1  0.085102 -0.255425  … -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  …  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  … -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  … -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
~~~~~~Últimos datos~~~~~~ :
            Time         V1         V2        V3        V4        V5
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473  \
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546

              V6        V7        V8        V9  …       V21       V22
284802 -2.606837 -4.918215  7.305334  1.914428  …  0.213454  0.111864  \
284803  1.058415  0.024330  0.294869  0.584800  …  0.214205  0.924384
284804  3.031260 -0.296827  0.708417  0.432454  …  0.232045  0.578229
284805  0.623708 -0.686180  0.679145  0.392087  …  0.265245  0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  …  0.261057  0.643078

             V23       V24       V25       V26       V27       V28  Amount
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77  \
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00
```

```
        Class
284802      0
284803      0
284804      0
284805      0
284806      0

[5 rows x 31 columns]
~~~~~~Descripción de los datos~~~~~~ :
                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  1.168375e-15  3.416908e-16 -1.379537e-15  2.074095e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 V5            V6            V7            V8            V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   9.604066e-16  1.487313e-15 -5.556467e-16  1.213481e-16 -2.406331e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       …           V21           V22           V23           V24
count  …  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  \
mean   …  1.654067e-16 -3.568593e-16  2.578648e-16  4.473266e-15
std    …  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    … -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    … -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    … -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    …  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    …  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28         Amount
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000  \
mean   5.340915e-16  1.683437e-15 -3.660091e-16 -1.227390e-16      88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
```

```
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

               Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
~~~~~~Datos faltantes~~~~~~ :
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

El dataframe: “Credit fraud” contiene 31 variables, que constan de las siguientes características:

- Las primeras 30 variables son continuas, exceptuando la última que es la variable objetivo y por ende es categórica.
- La variable time es el momento en el que se realizó la transacción desde la primera observación del dataset.
- La varaible monto, es la cantidad monetaria a la que haciende la transacción.
- Las variables V1-V28, representan a diferentes características que por motivos de seguridad presentan algun tipo de transformación.
- Ninguna de las varaibles presenta algún dato en missing, por lo que se cuenta con la totalidad del dataset para la modelación. Se presentan 284,807 registros.

2 Anális exploratorio de los datos.

- Ante el enorme reto que significa modelar datasets cuya información es cuantiosa, como buena práctica se hace uso de la siguiente función para optimizar el uso de la memoria.

[ ]:
```
# Definición de la función para la reducción de uso de memoria
def reduce_mem_usage(df):
    ### itera a través de todas las columnas de un "dataframe" y modifica el␣
 ↪tipo de datos
    ### para reducir el uso de la memoria.        ###
    # Se imprime el tamaño inicial del dataframe.
    start_mem = df.memory_usage().sum() / 1024**2
    print('El uso de memoria del "dataframe" es {:.2f} MB'.format(start_mem))

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            # Si el tipo de dato es númerico se reduce el espacio al mínimo␣
 ↪tamaño posible que permite almacenar
            # el dato más grande o pequeño de la columna
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).
 ↪max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.
 ↪int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.
 ↪int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.
 ↪int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
```

```
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.
↪float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.
↪float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
            #Si el tipo de dato es caracter, se transforma el tipo de dato a␣
↪categoríco o categoría.
        else:
            df[col] = df[col].astype('category')
    # Se imprime el resultado final.
    end_mem = df.memory_usage().sum() / 1024**2
    print('El uso de memoria después de la optimización es: {:.2f} MB'.
↪format(end_mem))
    print('Disminuido en {:.1f}%'.format(100 * (start_mem - end_mem) /␣
↪start_mem))
    return df
```

[ ]:
```
#Se procede a guardar cada uno de los datasets.
credit_fraud1 = pd.read_csv('creditcard.csv')
#Se aplica la función en los diversos dataframes definidos.
credit_fraud2 = reduce_mem_usage(credit_fraud1)
```

```
El uso de memoria del "dataframe" es 67.36 MB
El uso de memoria después de la optimización es: 17.11 MB
Disminuido en 74.6%
```

2.1 Análisis de la información.

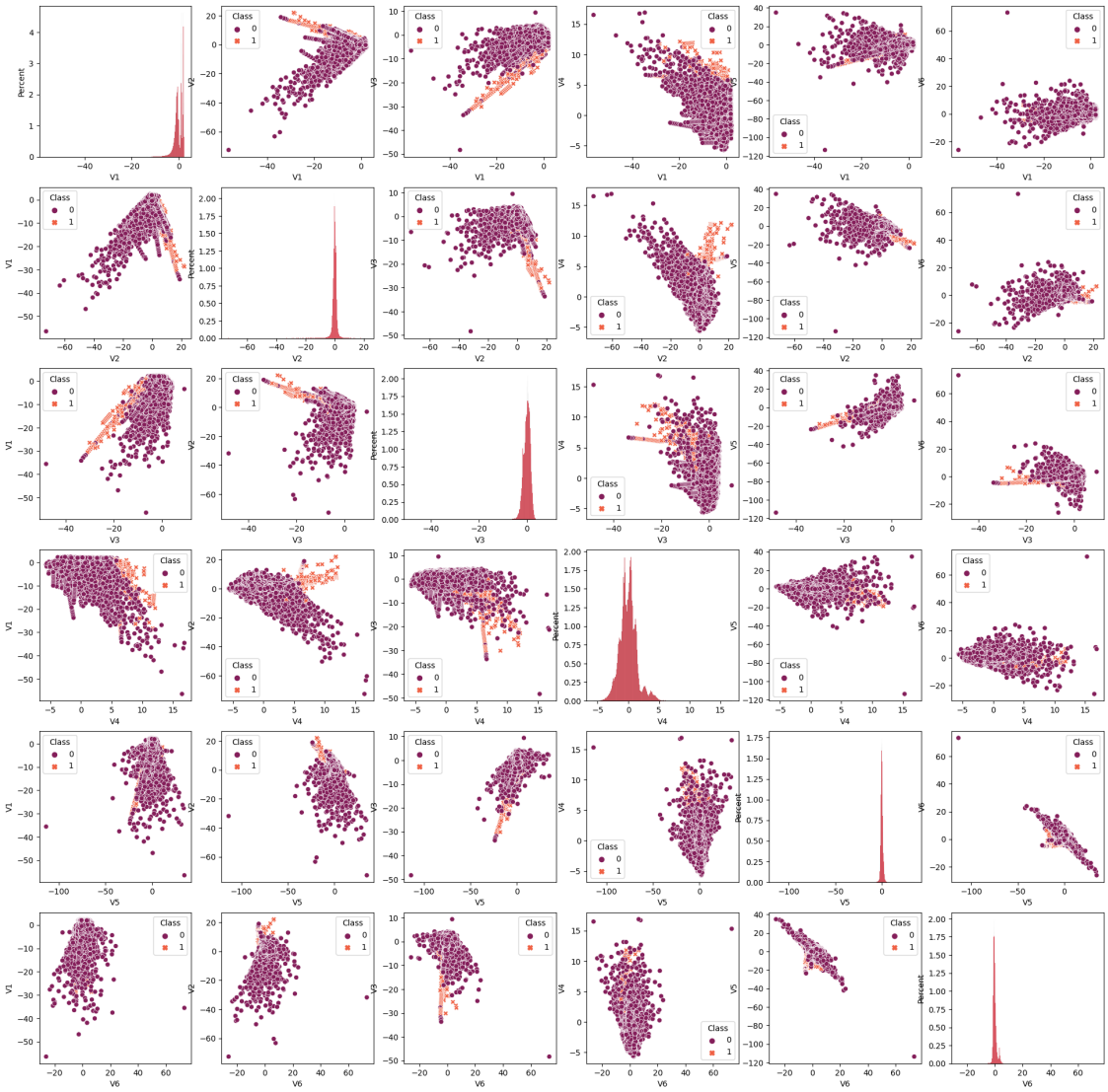
- A continuación se procede al análisis visual de la información, haciendo uso de diversos gráficos (histogramas, scatter plots o box plots). Con el objetivo de identificar patrones o características de población que nos ayuden a identificar aquellas varaibles que pudieran ser de mayor utilidad para nuestro objetivo planteado.

[ ]:
```
credit_fraud.columns
```

[ ]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

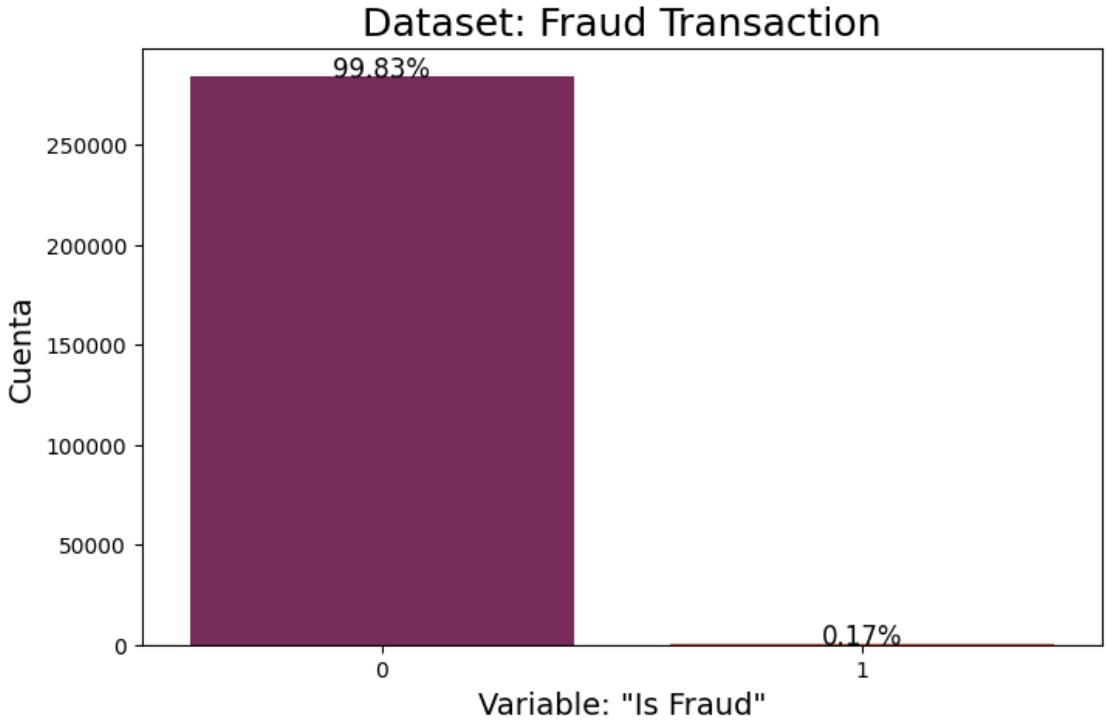
[ ]:
```
credit_fraud = pd.read_csv('creditcard.csv')

list_v = list(credit_fraud.columns[1:-3])
list_o = list(credit_fraud.columns[[0, -2, -1]])
```

```
fig, ax = plt.subplots(6,6, figsize=(24,24))
for i in range(6):
    for j in range(6):
        if i == j:
            sns.histplot(data=credit_fraud[[list_v[i]]], x=list_v[i],
                         stat= 'percent', color=sns.color_palette("rocket")[3],␣
 ↪ax=ax[i,j])
        else:
            sns.scatterplot(data=credit_fraud[[list_v[i], list_v[j],␣
 ↪list_o[-1]]], x=list_v[i],
                            y= list_v[j], hue= list_o[-1], style= list_o[-1],␣
 ↪palette= 'rocket',
                            ax=ax[i,j] )
plt.show()
```



- Se observa de estas primeras varaibles, que aquellas operaciones marcadas como fraudulentas presentan cierta concentración en algunas combinaciones de variables. Por lo que, en futuros pasos crearemos variables con combinaciones lineales de dichas variables con el objetivo de encontrar alguna que pueda brindar un mayor predictivo a la que se obtinene con las variables por si solas.
- Un segundo punto a destacar es el hecho que se observa una concentración de los histrogramas. Por lo que es claro, que existen outliers en la población. Los cuales habrá que evaluarlos para saber si mantenerlos o no, en función a si las observaciones corresponden a aquellas operaciones fraudulentas o no.

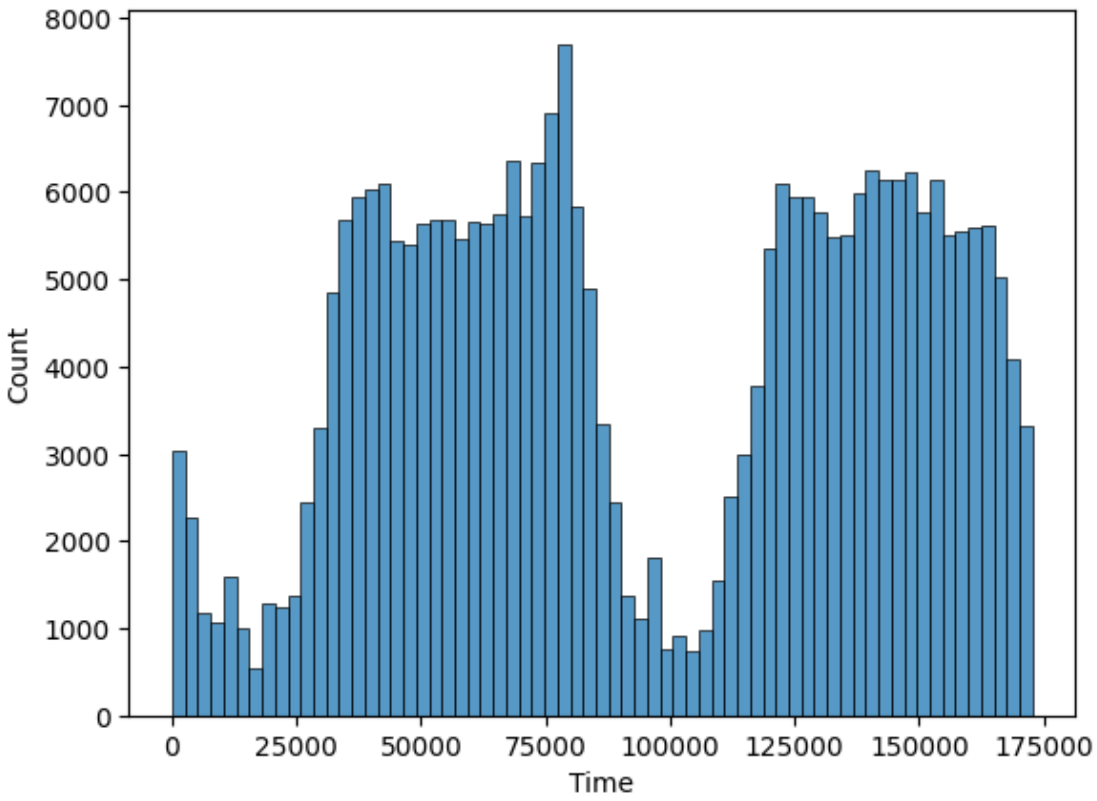
[ ]:
```
fig, ax  = plt.subplots(figsize=(8, 5))
sns.countplot(data = credit_fraud[[list_o[-1]]], x= list_o[-1],␣
 ↪palette="rocket",
               ax=ax)
ax.set_title('Dataset: Fraud Transaction', fontsize=18)
ax.set_xlabel('Variable: "Is Fraud"', fontsize=14)
ax.set_ylabel('Cuenta', fontsize=14)
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x()+p.get_width()/2.,
            height,
            f'{height/credit_fraud.shape[0] * 100:.2f}%',
            ha='center', fontsize=12)
```



- Como era de esperarse dada la naturaleza del problema. Se observa que la variable objetivo presenta una distribución totalmente desbalanceada donde solo el 0.17% de las operaciones marcadas como fraudulentas.

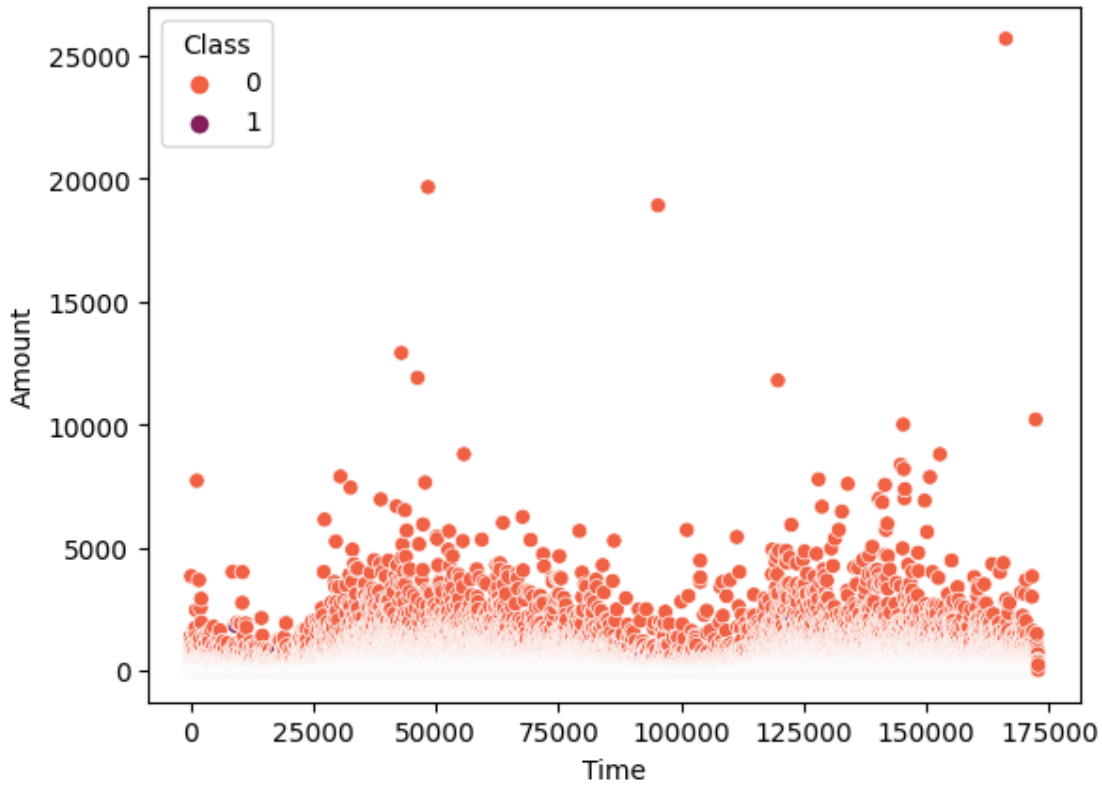
[ ]:
```
sns.histplot(data=credit_fraud, x= list_o[0])
```

[ ]:
```
<Axes: xlabel='Time', ylabel='Count'>
```



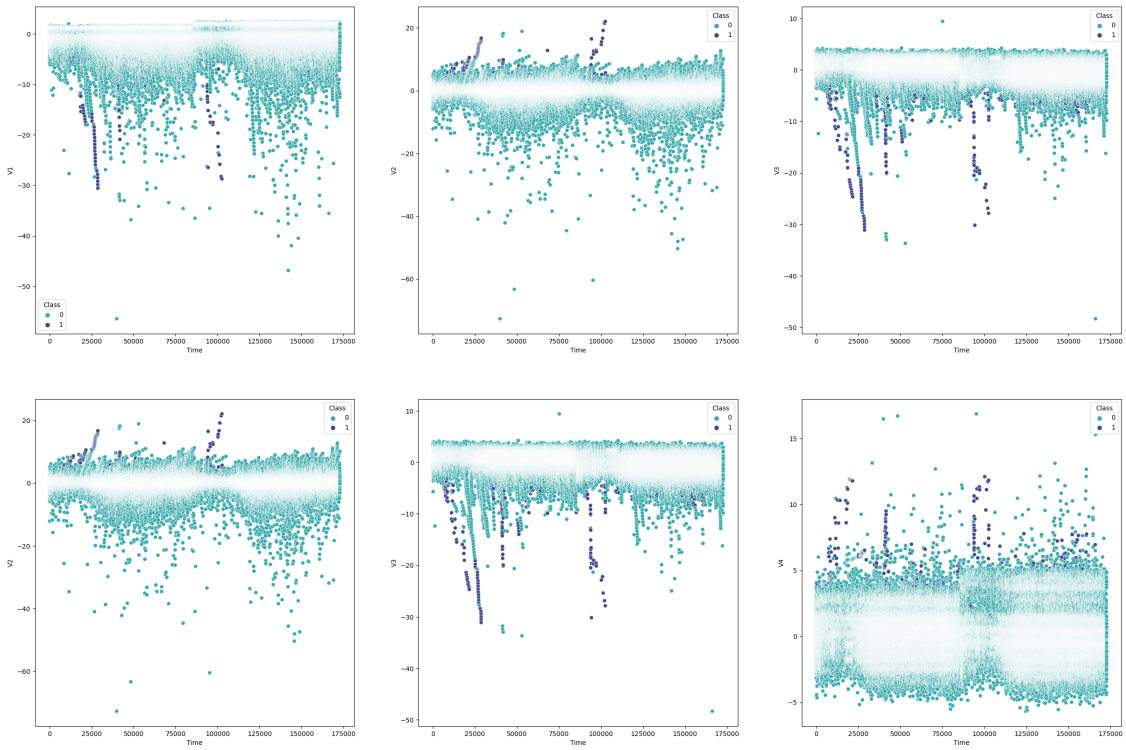
[ ]:
```
sns.scatterplot(data=credit_fraud, x=list_o[0],
                                y= list_o[1], hue= list_o[-1], palette='rocket_r')
```

[ ]:
```
<Axes: xlabel='Time', ylabel='Amount'>
```



- La variable tiempo presenta uan distribución bimodal, y si gráficamos el monto de las transacciones vs este no se nota una concentración de la variable objetivo en ellas.

[ ]:
```
fig, ax = plt.subplots(2,3, figsize=(30, 20))
for i in range(2):
    for j in range(3):
        sns.scatterplot(data=credit_fraud[[list_o[0], list_v[i+j],␣
↪list_o[-1]]], x=list_o[0],
                            y= list_v[i+j], hue= list_o[-1], palette=␣
↪'mako_r', ax=ax[i,j] )
    #ax[i].set_title("Distribución de "+(card))
```

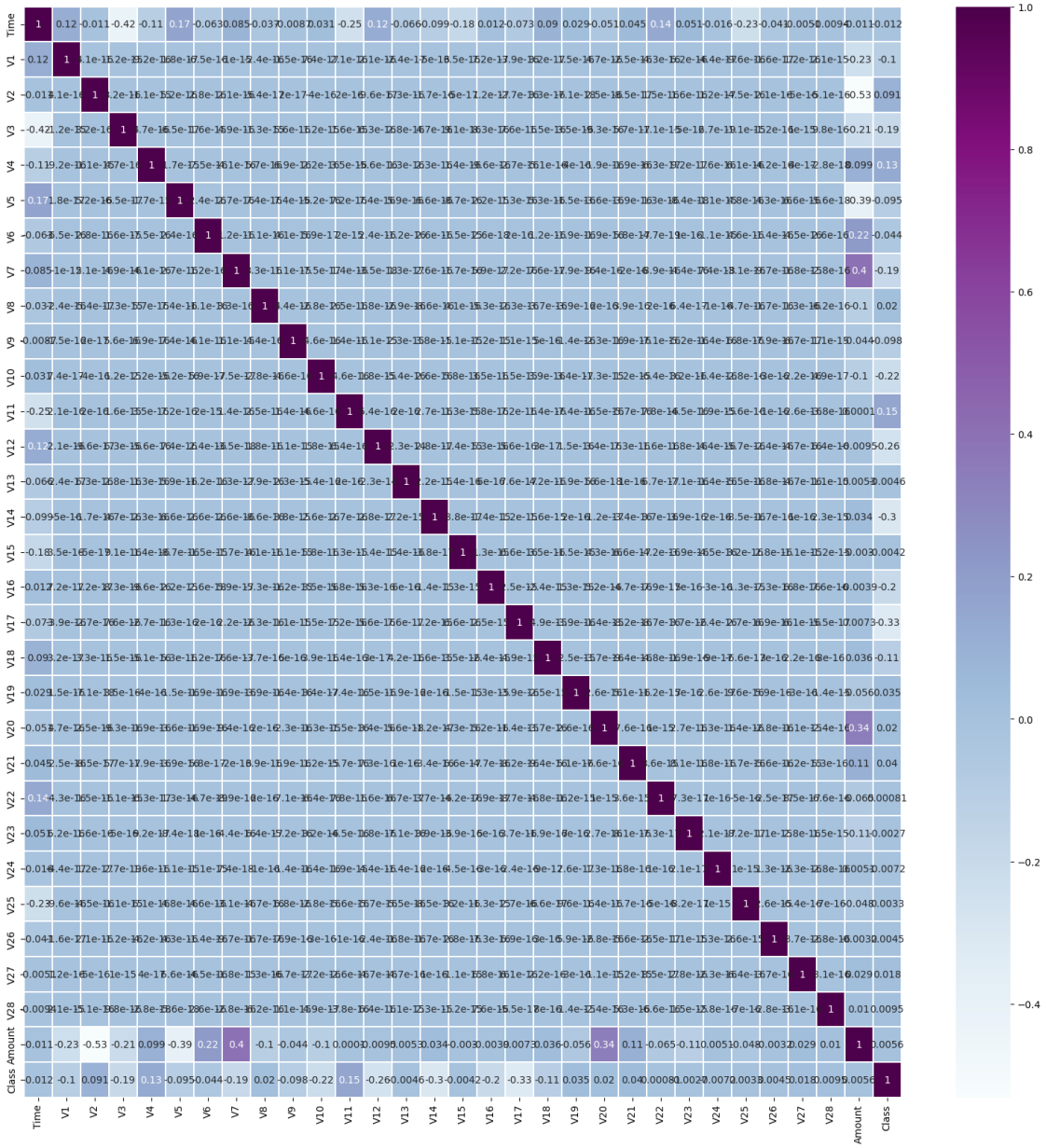


Asi mismo comparamos el tiempo de la transaccionalidad vs las primeras caracterísitcas encriptadas.

Finalemente, hacemos un análisis de correlación para cononcer, si existe algun par de varaibles cuya correlación sea sospechosamente fuerte.

[ ]:
```
corr = credit_fraud.corr()
fig, ax = plt.subplots(figsize=(20, 20))
sns.heatmap(corr, linewidths=.2, annot=True, ax=ax, cmap="BuPu")
```

[ ]: <Axes: >



3) Ingenieria de variables

- Derivado del Análisis exploratorio, se procede a realizar la siguiente ingenieria de variables:

1) Nos des-haremos de los outliers.
2) Estandarizaremos la linformación.

[ ]:
```
def aux_outliers(a,b,c):
    a=set(a)
    b=set(b)
    c=set(c)
```

```
    a_=a.intersection(b)

    b_=b.intersection(c)

    c_=a.intersection(c)

    outliers_index=list(set(list(a_)+list(b_)+list(c_)))
    return outliers_index


def OUTLIERS(df,cols):
    results=pd.DataFrame()
    data_iqr=df.copy()
    data_per=df.copy()
    total=[]
    total_per=[]
    total_z=[]
    indices_=[]

    for col in cols:
        #IQR
        Q1=df[col].quantile(0.25)
        Q3=df[col].quantile(0.75)
        IQR=Q3-Q1
        INF=Q1-1.5*(IQR)
        SUP=Q3+1.5*(IQR)


        n_outliers=df[(df[col] < INF) | (df[col] > SUP)].shape[0]
        total.append(n_outliers)
        indices_iqr=list(df[(df[col] < INF) | (df[col] > SUP)].index)
        #data_iqr=data_iqr[~(data_iqr[col] < INF) | (data_iqr[col] > SUP)].
 ↪reset_index(drop=True)

        #Percentiles
        INF_pe=np.percentile(df[col].dropna(),1)

        SUP_pe=np.percentile(df[col].dropna(),99)
        n_outliers_per=df[(df[col] < INF_pe) | (df[col] > SUP_pe)].shape[0]
        total_per.append(n_outliers_per)
        indices_per=list(df[(df[col] < INF_pe) | (df[col] > SUP_pe)].index)
        #data_per=data_per[~(data_per[col] < INF_pe) | (data_per[col] >␣
 ↪SUP_pe)].reset_index(drop=True)

        #MEAN CHANGE
```

```
        #Obtenemos todos los percentiles además del máximo
        perc_100 = [x / 100 for x in range(100)]
        dist = df[col].describe(perc_100).iloc[4:]
        #Obtenemos el cambio entre percentiles
        change_dist = df[col].describe(perc_100).iloc[4:].diff()
        #Obtenemos el cambio promedio entre percentiles
        mean_change = df[col].describe(
            perc_100).iloc[4:].diff().mean()
        #Si el cambio entre el percentil 99 y el maximo es mayor a el cambio␣
↪promedio entonces:
        if change_dist["max"] > mean_change:
            #La banda superior será el máximo menos el cambio promedio
            ub = dist["max"] - mean_change
            #si la banda superior es más pequeña que el percentil 99 ,␣
↪modificamos la banda para que tome el percentil 99
            if ub < dist["99%"]:
                ub = dist["99%"]
        else:
        #Si el cambio entre el percentil 99 y el maximo es menor o igual a el␣
↪cambio promedio entonces se toma el percentil 99
            ub = dist["max"]

        if change_dist["1%"] > mean_change:
            lb = dist["0%"] + mean_change
            if lb > dist["1%"]:
                lb = dist["1%"]
        else:
            lb = dist["0%"]
        n_total_z=df[(df[col] < lb) | (df[col] > ub)].shape[0]
        total_z.append(n_total_z)
        indices_z=list(df[(df[col] < lb) | (df[col] > ub)].index)

        indices_.append(aux_outliers(indices_iqr,indices_per,indices_z))

    results["features"]=cols
    results["n_outliers_IQR"]=total
    results["n_outliers_Percentil"]=total_per
    results["n_outliers_Mean_Change"]=total_z
    results["n_outliers_IQR_%"]=round((results["n_outliers_IQR"]/df.
↪shape[0])*100,2)
    results["n_outliers_Percentil_%"]=round((results["n_outliers_Percentil"]/df.
↪shape[0])*100,2)
    ␣
↪results["n_outliers_Mean_Change_%"]=round((results["n_outliers_Mean_Change"]/
↪df.shape[0])*100,2)
    results["indices"]=indices_
```

```
    results["total_outliers"]=results["indices"].map(lambda x:len(x))
    results["%_outliers"]=results["indices"].map(lambda x:round(((len(x)/df.
 ↪shape[0])*100),2))
    results=results[['features', 'n_outliers_IQR', 'n_outliers_Percentil',
       'n_outliers_Mean_Change', 'n_outliers_IQR_%', 'n_outliers_Percentil_%',
       'n_outliers_Mean_Change_%',  'total_outliers', '%_outliers','indices']]
    return results
```

Con la primera variable que buscaremos eliminar outliers es con la variable de "Amount".

[ ]:
```
OUTLIERS(df=credit_fraud, cols= [(list_o[1])])
```

[ ]:
```
  features  n_outliers_IQR  n_outliers_Percentil  n_outliers_Mean_Change
0   Amount           31904                  5618                       1  \

   n_outliers_IQR_%  n_outliers_Percentil_%  n_outliers_Mean_Change_%
0              11.2                   1.97                       0.0  \

   total_outliers  %_outliers
0            2849         1.0  \

                                                 indices
0  [163840, 163841, 221184, 212995, 212992, 49161…
```

[ ]:
```
outliers=OUTLIERS(df=credit_fraud, cols= [(list_o[1])])
indices=list(outliers["indices"].values)
indices=list(set(reduce(lambda x,y: x+y, indices)))

lenght=len(indices)
#Imprimimos el total de observaciones que se dejarían fuera y porcentaje que␣
 ↪representan del data set original.
print(lenght, lenght/credit_fraud.shape[0])
```

```
2849 0.01000326536917983
```

- Observamos el impacto en nuestra variable objetivo.

[ ]:
```
credit_fraud['Class'].value_counts()
```

[ ]:
```
Class
0    284315
1       492
Name: count, dtype: int64
```

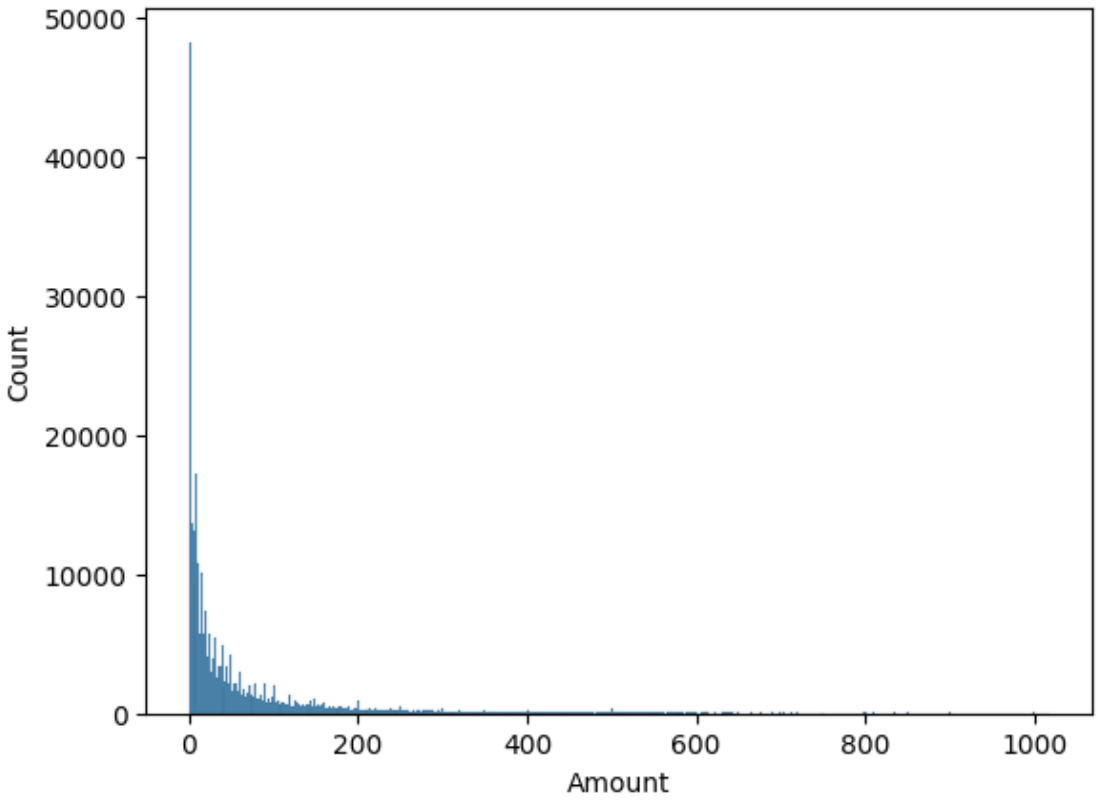
[ ]:
```
data=credit_fraud[~credit_fraud.index.isin(indices)] #Se eliminan de db los␣
 ↪registros outliars
```

[ ]:
```
data['Class'].value_counts()
```

[ ]:
```
Class
0    281475
1       483
Name: count, dtype: int64
```

[ ]:
```
sns.histplot(data, x='Amount')
```

[ ]:
```
<Axes: xlabel='Amount', ylabel='Count'>
```



3.2) Procedemos al escalamiento de la información.

[ ]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

[ ]:
```
a1 = list(credit_fraud.columns[:-1])
data[a1] = scaler.fit_transform(data[a1])
```

```
/tmp/ipykernel_1278/329753021.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
```

```
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data[a1] = scaler.fit_transform(data[a1])
```

[ ]:
```
data.head()
```

[ ]:
```
        Time        V1        V2        V3        V4        V5        V6  \
0   0.000000  0.922638  0.681518  0.841021  0.375357  0.403874  0.597323
1   0.000000  0.974385  0.686395  0.785989  0.325921  0.410769  0.583742
2   0.000006  0.922667  0.663279  0.823300  0.322286  0.401020  0.630684
3   0.000006  0.930618  0.679899  0.823759  0.256209  0.409552  0.616890
4   0.000012  0.926726  0.695197  0.818086  0.323522  0.402682  0.588187

         V7        V8        V9  …       V21       V22       V23       V24  \
0   0.739575  0.786444  0.475312  …  0.561184  0.472656  0.622227  0.423290
1   0.734198  0.786298  0.453981  …  0.557840  0.425389  0.625831  0.363989
2   0.748894  0.788042  0.410603  …  0.565477  0.498125  0.639586  0.313047
3   0.739541  0.789434  0.414999  …  0.559734  0.458599  0.620867  0.242154
4   0.745541  0.782484  0.490950  …  0.561327  0.499497  0.621767  0.434127

         V25       V26       V27       V28    Amount  Class
0   0.507766  0.336462  0.699610  0.340504  0.147047      0
1   0.510339  0.392856  0.695216  0.341546  0.002644      0
2   0.477385  0.345417  0.693787  0.339377  0.372147      0
3   0.542320  0.330588  0.697426  0.342908  0.121376      0
4   0.485486  0.460239  0.702256  0.347385  0.068786      0

[5 rows x 31 columns]
```

4) Modelado.

- Se aplicará el clásico modelo de regresión logística como primera aproximación a nuestro problema.

[ ]:
```
target = "Class"
X = data[[x for x in data.columns if x!=target]]
y = data[target]

X.shape
```

[ ]:
```
(281958, 30)
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

[ ]:
```
logistic = LogisticRegression()
logistic.fit(X_train, y_train)
```

[ ]:
```
LogisticRegression()
```

```
[ ]: def classification_metrics(X, y, estimator):
        ls_scores_roc = cross_val_score(estimator=estimator, X=X, y=y,␣
     ↪scoring="roc_auc", n_jobs=-1, cv=4)
        print(f"ROC media: {np.mean(ls_scores_roc):,.2f}, desviación estándar: {np.
     ↪std(ls_scores_roc)}")
```

```
[ ]: print(classification_report(logistic.predict(X_train), y_train))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    197154
           1       0.54      0.87      0.67       216

    accuracy                           1.00    197370
   macro avg       0.77      0.93      0.83    197370
weighted avg       1.00      1.00      1.00    197370
```

```
[ ]: print(classification_report(logistic.predict(X_test), y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     84509
           1       0.51      0.89      0.65        79

    accuracy                           1.00     84588
   macro avg       0.76      0.94      0.83     84588
weighted avg       1.00      1.00      1.00     84588
```

5) Conclusisones:

- Como primera aproximación, es evidente que la precisión se tan baja dado las características de la información. A partir de esto se trabajara en crear varaibles que permitan una mejor segmentación, las cuales puedan ser utilizadas en modelos más compljeos.

Referencias:

1) Credit Card Fraud Detection. (2018, 23 marzo). Kaggle. https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud