Credit Card Fraud Detection with Random Forest

dataset -> https://www.kaggle.com/mlg-ulb/creditcardfraud

En este proyecto analizamos un conjunto de datos de transacciones con tarjetas de crédito realizadas durante un período de dos días.

Cada transacción tiene 30 características, todas ellas numéricas. Las características V1, V2, . . . , V28 son el resultado de una transformación PCA. Para proteger la confidencialidad, la información básica sobre estas funciones no está disponible. La función Tiempo contiene el tiempo transcurrido desde la primera transacción y la función Monto contiene el monto de la transacción. La variable de respuesta, Clase, es 1 en el caso de fraude y 0 en caso contrario.

El objetivo en este proyecto es construir un modelo que nos ayude a predecir si una transacción con tarjeta de crédito es fraudulenta o no.

Las variables del dataset son:

Tiempo: Número de segundos transcurridos entre una transacción y la primera transacción en el conjunto de datos.

V1-V28: Puede ser el resultado de una reducción de la dimensionalidad de PCA para proteger las identidades de los usuarios y las funciones sensibles (v1-v28).

Amount: Cantidad de transacción.

Class: 1 para transacciones fraudulentas, 0 en caso contrario.

[1]:
```
#import libraries

import numpy as np
import scipy as sp
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[2]:
```
# Load data
data = pd.read_csv('creditcard.csv', header=0)
```

[3]:
```
# Data preparation

data.shape
```

[3]: (284807, 31)

[4]:
```
data.columns
```

[4]:
```
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')
```

[5]:
```
data.head()
```

[5]:
```
   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]
```

[6]:
```
data.sample(5)
```

[6]:
```
            Time        V1        V2        V3        V4        V5        V6  \
55687    47067.0 -1.500544  1.475549  0.503194 -1.632386 -0.413720 -0.495604
129083   78944.0 -0.997035  0.512693  2.201569  3.090123 -0.233869  2.109021
188270  127898.0  1.994690 -0.364561 -0.935029 -0.005480  0.144659  0.483885
193519  130170.0  2.295212 -1.274653 -0.901225 -1.469508 -1.299759 -1.054565
184971  126473.0  1.930527 -0.647935 -0.305714  0.283875 -1.011310 -0.703618

              V7        V8        V9  ...       V21       V22       V23  \
55687   0.003230  0.642250  0.532465  ... -0.216303 -0.368483 -0.051223
129083 -0.878401  1.101786 -0.619495  ...  0.100403  0.422904 -0.071087
188270 -0.521046  0.135746  1.053976  ...  0.165735  0.666933  0.039633
```

```
193519 -0.933855 -0.346781 -1.322026  ... -0.162860  0.081728  0.210588
184971 -0.657569 -0.027765  1.666606  ...  0.187815  0.679975  0.121565

             V24       V25       V26       V27       V28  Amount  Class
55687  -0.350526 -0.069347  0.712115 -0.096524 -0.341537    0.77      0
129083 -0.751963 -0.342607  0.283967  0.105583  0.024435   48.51      0
188270 -0.093336  0.089501 -0.586620  0.050204 -0.044413   15.00      0
193519 -0.024364 -0.141579 -0.161604  0.011824 -0.049779   20.00      0
184971  0.064469 -0.216707  0.122576  0.003712 -0.038358   39.95      0

[5 rows x 31 columns]
```

[7]:
```
data.tail()
```

[7]:
```
            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]
```

[8]:
```
data.describe()
```

[8]:

```
                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  3.918649e-15  5.682686e-16 -8.761736e-15  2.811118e-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  -1.552103e-15  2.040130e-15 -1.698953e-15 -1.893285e-16 -3.147640e-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   ...  1.473120e-16  8.042109e-16  5.282512e-16  4.456271e-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   1.426896e-15  1.701640e-15 -3.662252e-16 -1.217809e-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]
```

[9]:
```
data.info()
```

```
<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
```

Podemos ver que todas las variables son numéricas, por lo que no haremos transformacion de variables falsas. Al arecer no hay valores nulos, pero procedemos a rectificar esto a continuación

[10]:

```
data.isnull().any()
```

[10]:

```
Time      False
V1        False
V2        False
V3        False
V4        False
V5        False
V6        False
V7        False
V8        False
V9        False
V10       False
V11       False
V12       False
V13       False
V14       False
V15       False
V16       False
V17       False
V18       False
V19       False
V20       False
V21       False
V22       False
V23       False
V24       False
V25       False
V26       False
V27       False
V28       False
Amount    False
Class     False
dtype: bool
```

[11]:

```
data.isnull().sum()
```

[11]:

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
```

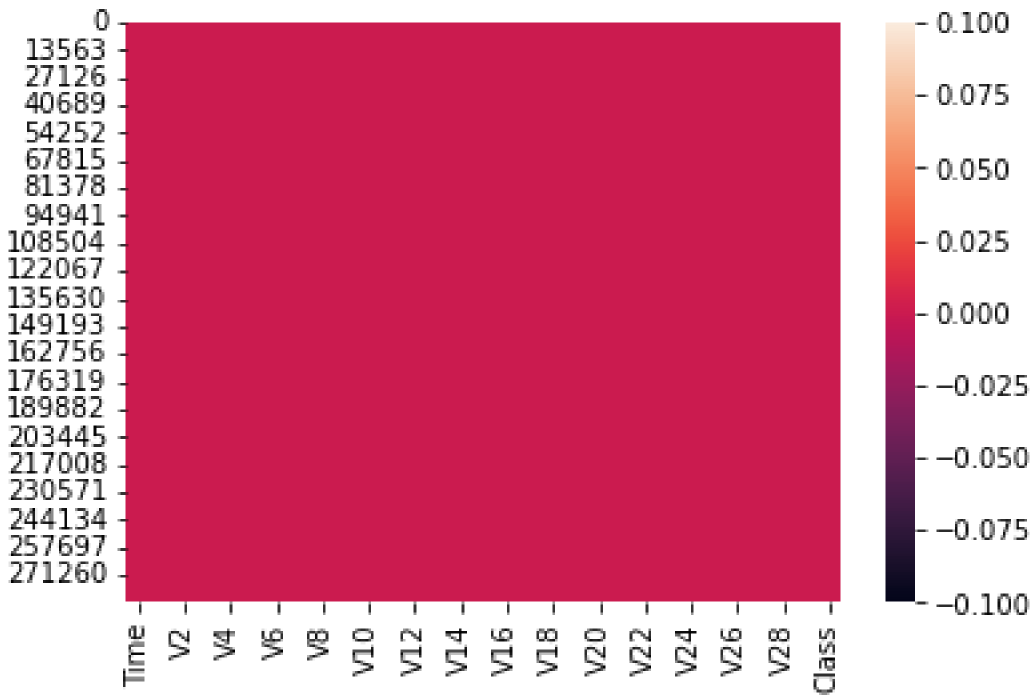
```
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

[12]:
```
data.isnull().any().any()
```

[12]: False

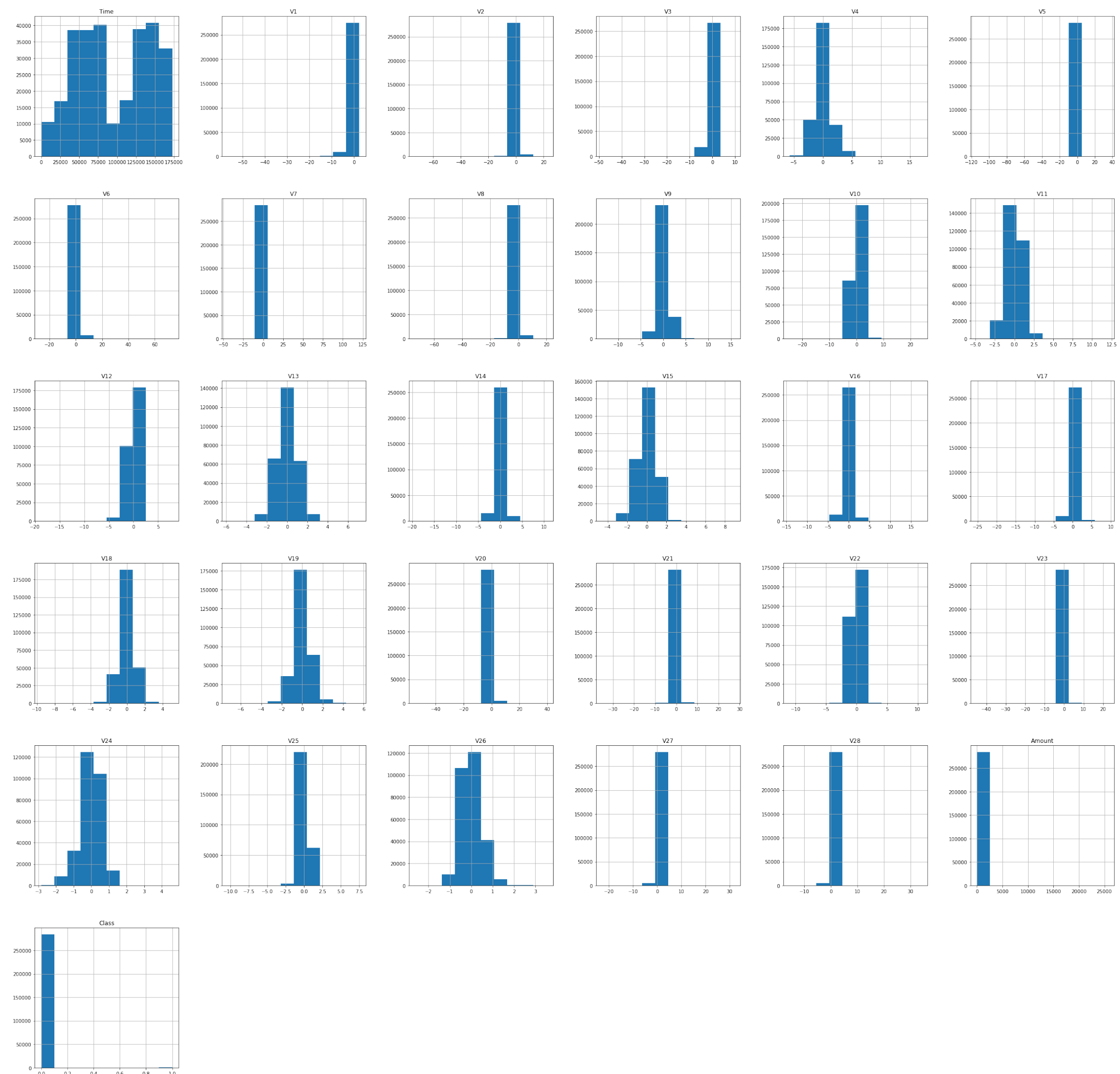
[13]:
```
sns.heatmap(data.isnull())
```

[13]: <AxesSubplot:>



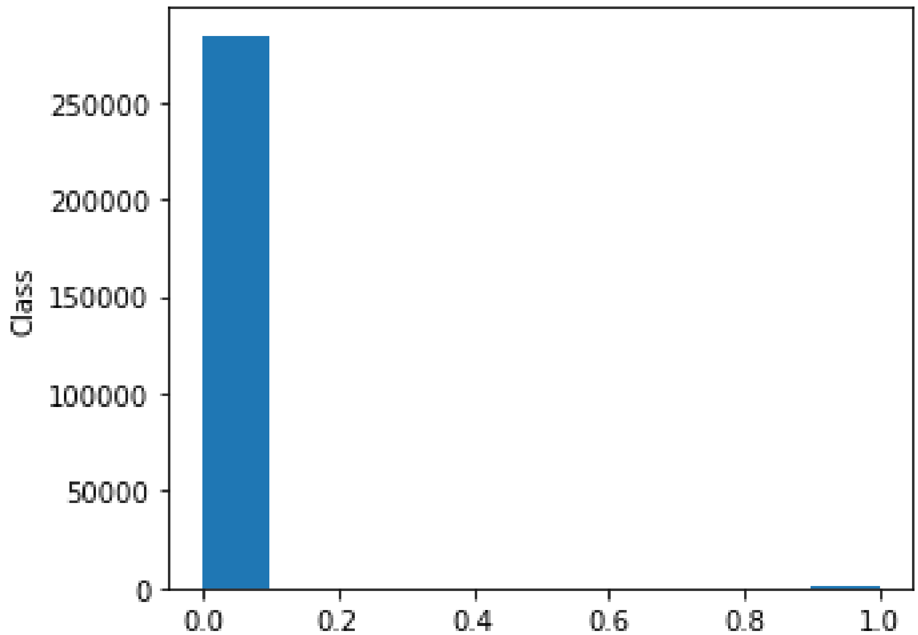
1 Exploratory analisis

[14]:
```
data.hist(figsize=(40,40))
plt.show()
```



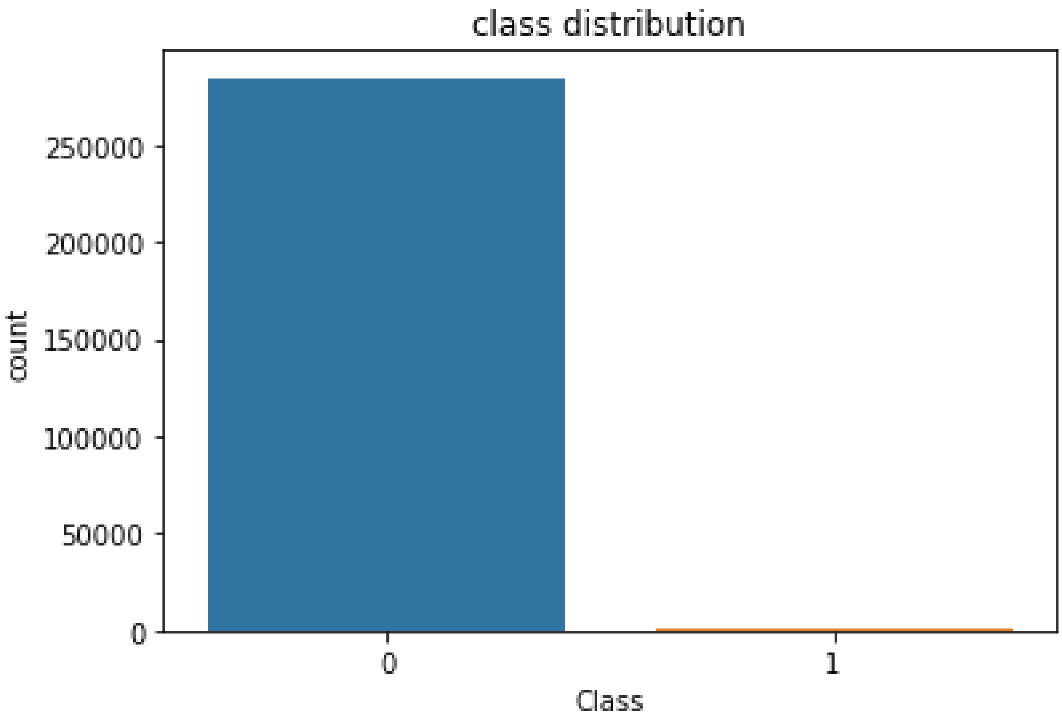
[15]:
```
plt.figure(figsize=(5,4))
plt.hist(data['Class'])
plt.ylabel('Class')
```

[15]:
```
Text(0, 0.5, 'Class')
```



[16]:
```
sns.countplot('Class',data=data)
plt.title("class distribution")
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



[17]:
```
# Podemos ver el porcentaje de fraudes y transacciones reales
data.Class.value_counts()
```

[17]:
```
0    284315
1       492
Name: Class, dtype: int64
```

[18]:
```
data.Class.value_counts(normalize=True)
```

[18]:
```
0    0.998273
1    0.001727
Name: Class, dtype: float64
```

Podemos ver que el porcentaje total de transacciones fraudulentas es de tan solo el 0.1727% lo que epresenta una, cantidad muy pequeña de todas las transacciones.

2 Time

[19]:
```
data['Time'].describe()
```

[19]:
```
count    284807.000000
mean      94813.859575
std       47488.145955
```

```
min           0.000000
25%       54201.500000
50%       84692.000000
75%      139320.500000
max      172792.000000
Name: Time, dtype: float64
```

Como la variable “Time” esta en segundos, nos es mas facil hacer un analisis ya sea por minuto o por hora, en este caso, la variable “Time” le aplicaremos una transformación de segundos a horas para una lectura mas sencilla.

[20]:
```
# Convertimos segundos en horas
data.loc[:,'Time'] = data.Time /3600
```

[21]:
```
data['Time'].describe()
```

[21]:
```
count    284807.000000
mean         26.337183
std          13.191152
min           0.000000
25%          15.055972
50%          23.525556
75%          38.700139
max          47.997778
Name: Time, dtype: float64
```

[22]:
```
data['Time'].max()
```

[22]:
```
47.99777777777778
```

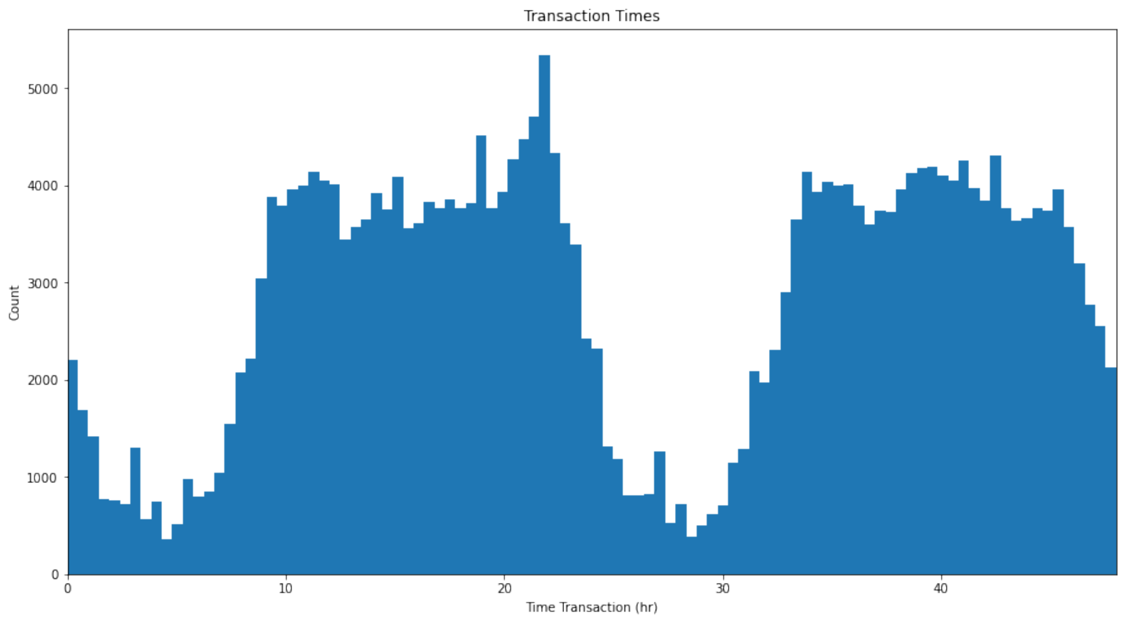
[23]:
```
data['Time'].max() /24
```

[23]:
```
1.9999074074074075
```

Rectificamos que en efecto todas las transacciones se hicieron en un lapso de 2 dias.

[24]:
```
plt.figure(figsize=(15,8))
plt.hist(data['Time'], bins=100)
plt.xlim([0,48])
plt.xlabel('Time Transaction (hr)')
plt.ylabel('Count')
plt.title('Transaction Times')
```

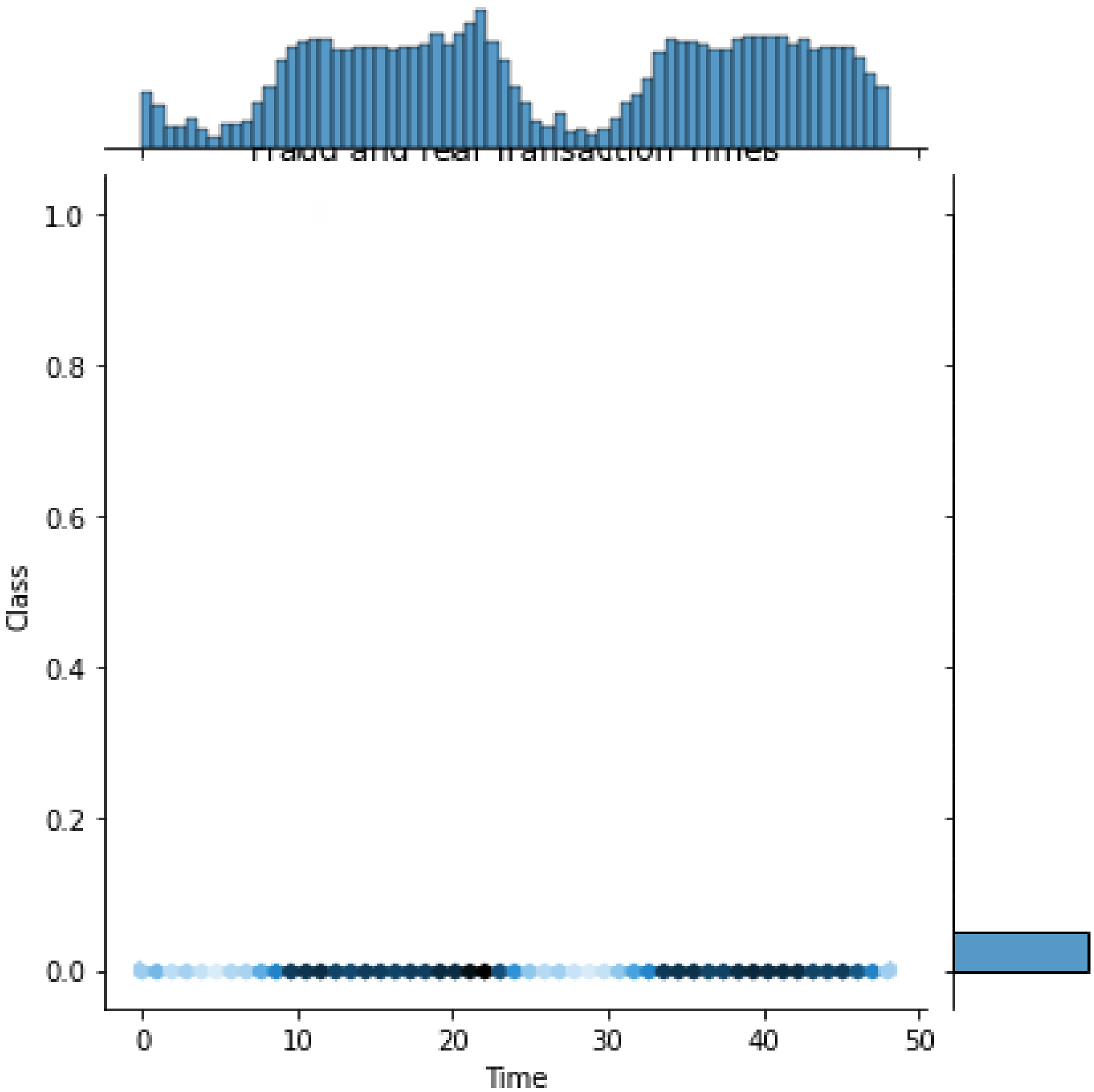
[24]:
```
Text(0.5, 1.0, 'Transaction Times')
```



[25]:
```
plt.figure(figsize=(15,8))
sns.jointplot(x='Time', y = 'Class', data= data, kind='hex')
plt.title('Fraud and real Transaction Times')
```

[25]:
```
Text(0.5, 1.0, 'Fraud and real Transaction Times')
```

```
<Figure size 1080x576 with 0 Axes>
```



Nuevamento podemos notar que la cantidad de transacciones fraudulentas son muy pocas, mas sin en cambio, las transacciones reales son la gran mayoria superioreas al 99.8%.

3 Amount

[26]:
```
data['Amount'].describe()
```

[26]:
```
count    284807.000000
mean         88.349619
std         250.120109
min           0.000000
25%           5.600000
```

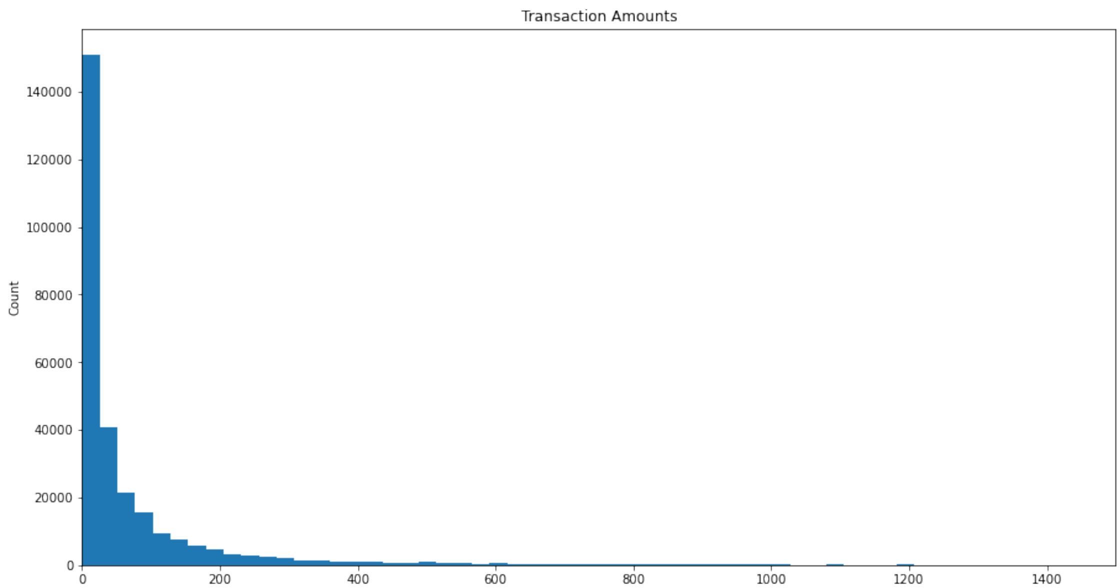
```
50%           22.000000
75%           77.165000
max        25691.160000
Name: Amount, dtype: float64
```

[27]:
```
plt.figure(figsize=(15,8))
plt.hist(data['Amount'], bins=300)
plt.ylabel('Count')
plt.title('Transaction Amounts')
```

[27]:
```
Text(0.5, 1.0, 'Transaction Amounts')
```

[28]:
```
plt.figure(figsize=(15,8))
plt.hist(data['Amount'], bins=1000)
plt.xlim([0,1500])
plt.ylabel('Count')
plt.title('Transaction Amounts')
```

[28]:
```
Text(0.5, 1.0, 'Transaction Amounts')
```

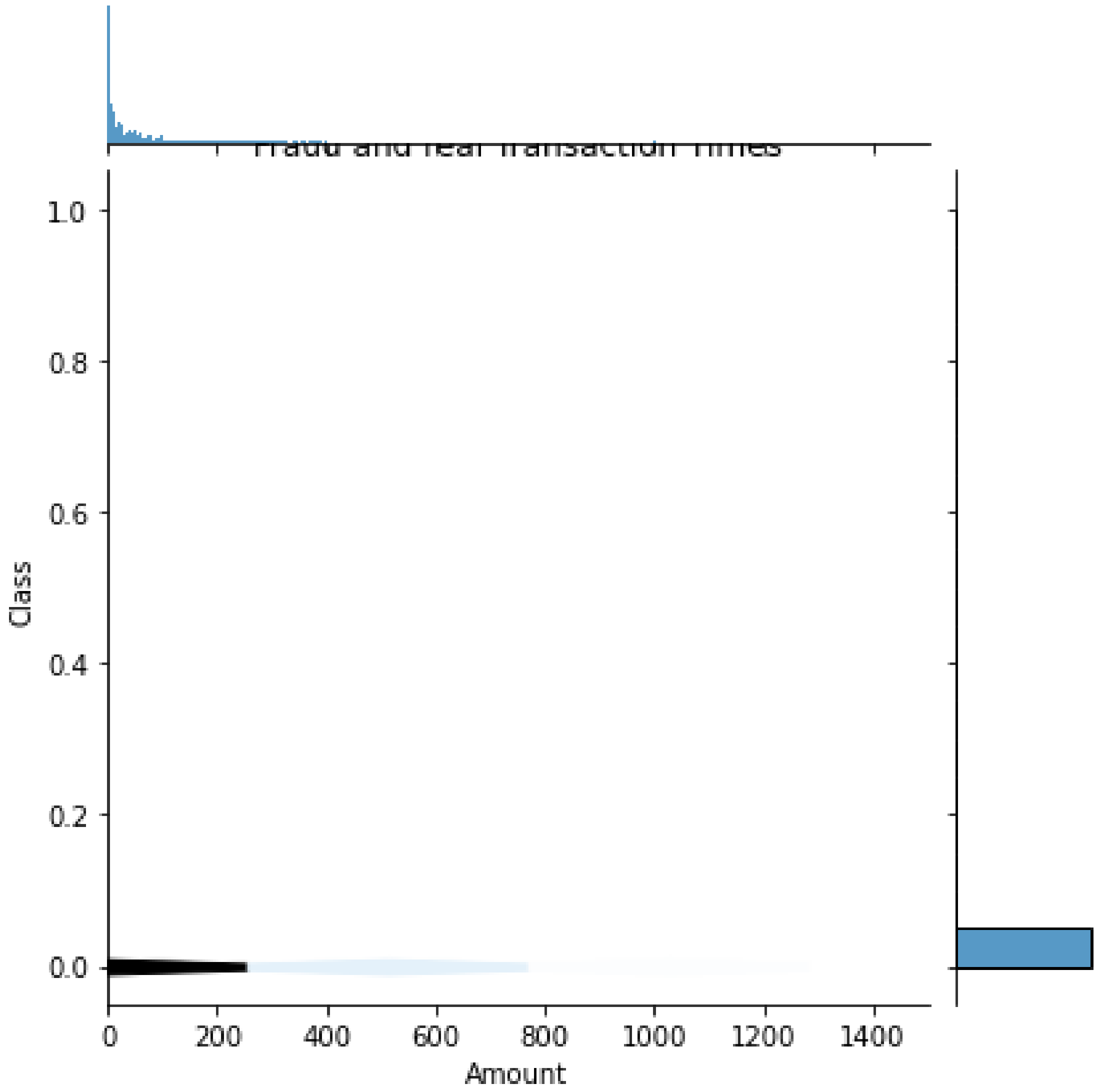


Las graficas anteriores nos muestran que la mayoria de transacciones, tanto fraudulentas como reales, estan por debajo de un monto de 1200.

[29]:
```
plt.figure(figsize=(15,8))
sns.jointplot(x='Amount', y = 'Class', data= data, kind='hex')
plt.xlim([0,1500])
plt.title('Fraud and real Transaction Times')
```

[29]:
```
Text(0.5, 1.0, 'Fraud and real Transaction Times')
```

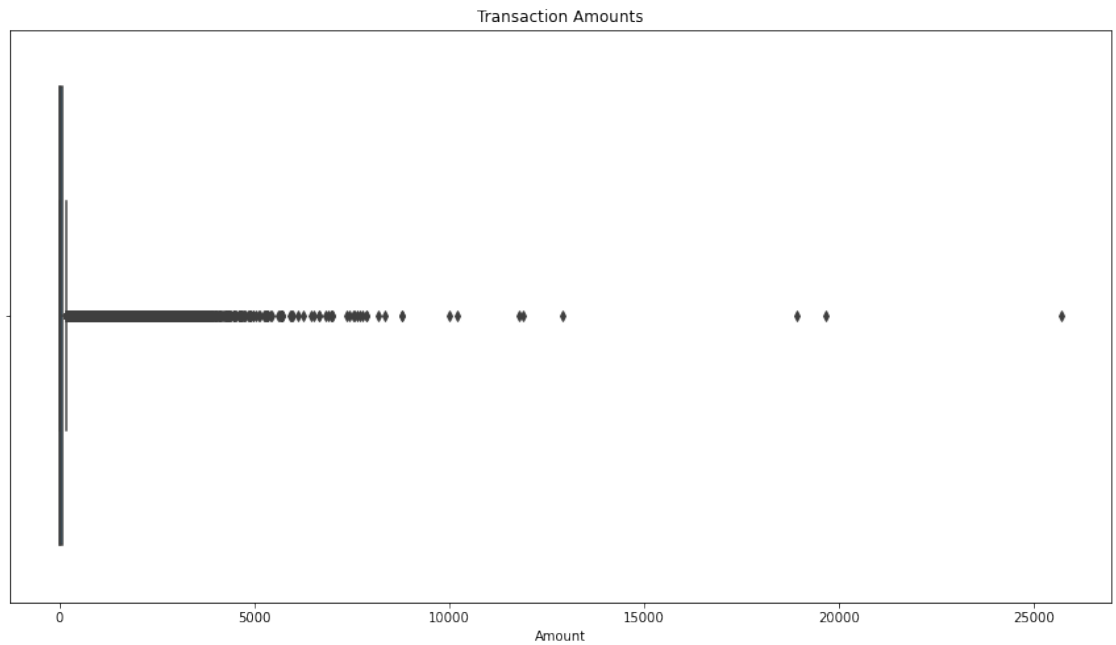
```
<Figure size 1080x576 with 0 Axes>
```



4 Outlier values

```
[30]: plt.figure(figsize=(15,8))
      sns.boxplot(x=data['Amount'])
      plt.title('Transaction Amounts')
```

```
[30]: Text(0.5, 1.0, 'Transaction Amounts')
```



[31]:
```
np.percentile(data.Amount,[99])
```

[31]: array([1017.97])

[32]:
```
np.percentile(data.Amount,[99])[0]
```

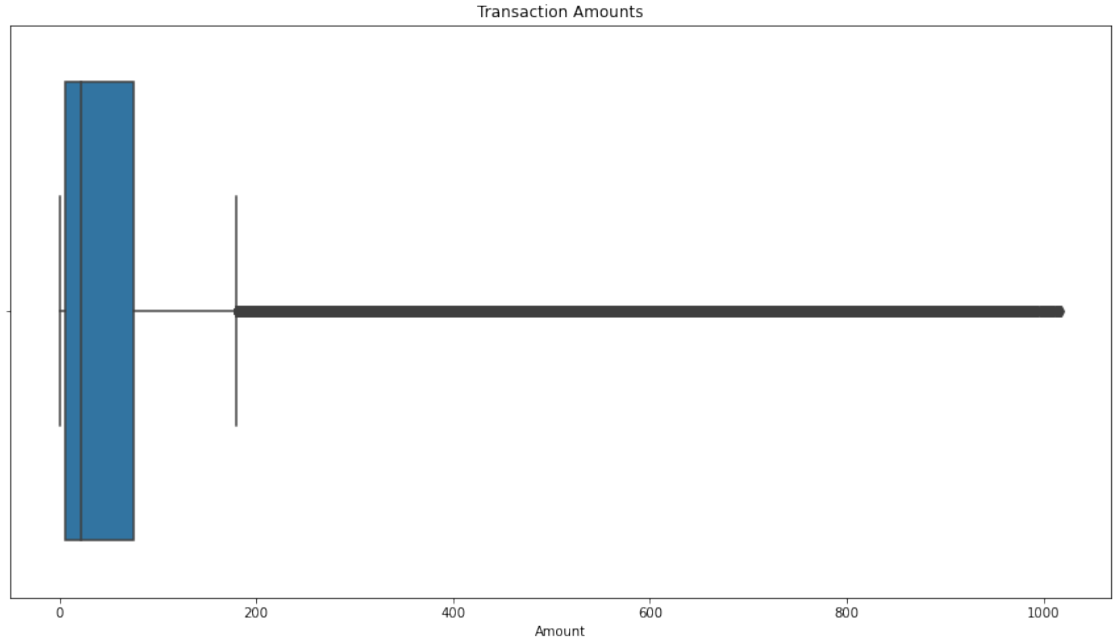
[32]: 1017.9700000000012

[33]:
```
uv = np.percentile(data.Amount,[99])[0]
```

[34]:
```
data = data.drop(data.index[data['Amount'] >= uv])
```

[35]:
```
plt.figure(figsize=(15,8))
sns.boxplot(x=data['Amount'])
plt.title('Transaction Amounts')
```

[35]: Text(0.5, 1.0, 'Transaction Amounts')



[36]:
```
data.shape
```

[36]: (281958, 31)

[37]:
```
data['Amount'].describe()
```

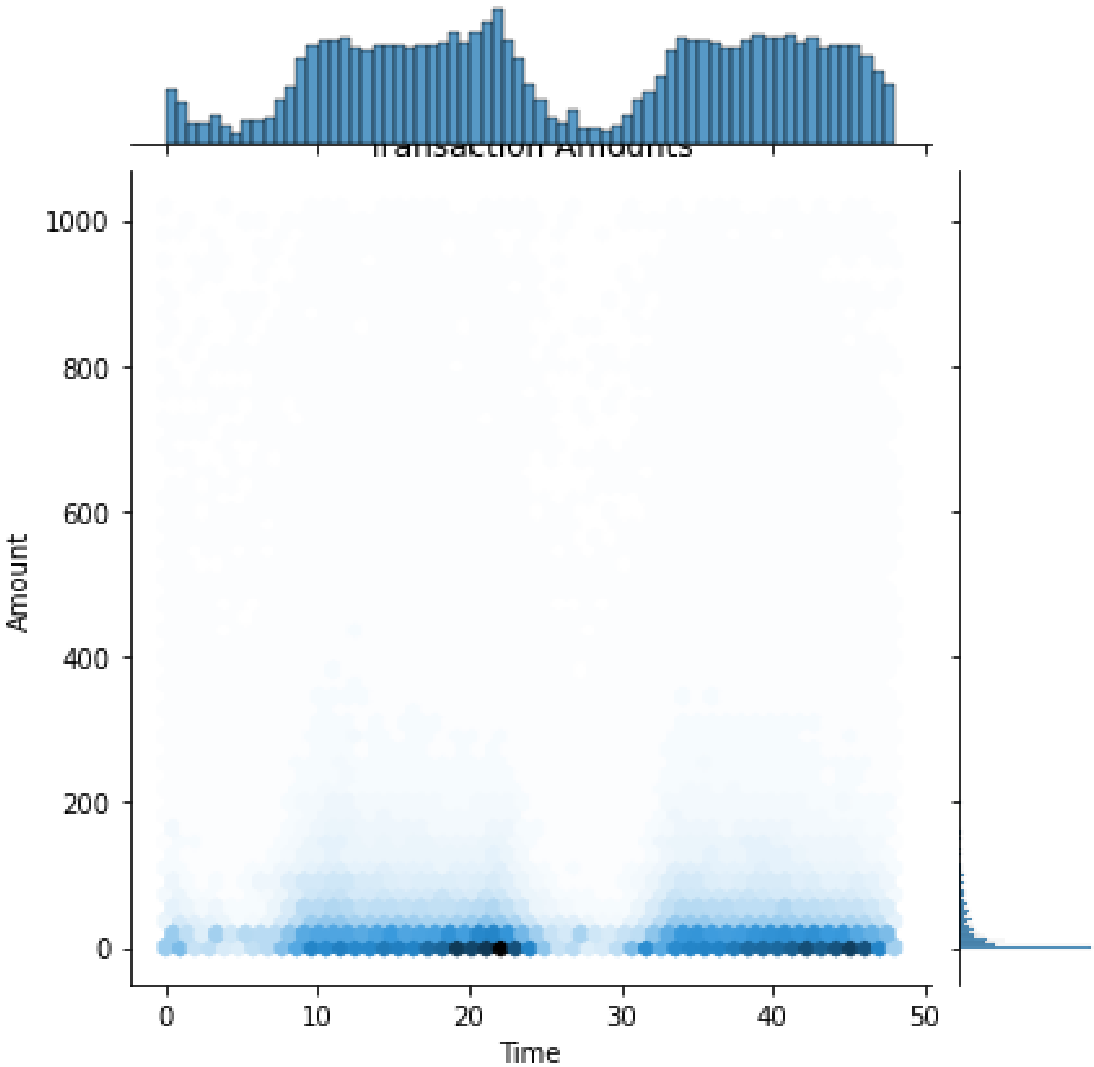
[37]:
```
count    281958.000000
mean         70.716411
std         129.043036
min           0.000000
25%           5.470000
50%          21.190000
75%          74.950000
max        1017.500000
Name: Amount, dtype: float64
```

5 Time vs. Amount

[38]:
```
plt.figure(figsize=(15,8))
sns.jointplot(x='Time', y = 'Amount', data= data, kind='hex',bins=100)
plt.title('Transaction Amounts')
```

[38]: Text(0.5, 1.0, 'Transaction Amounts')

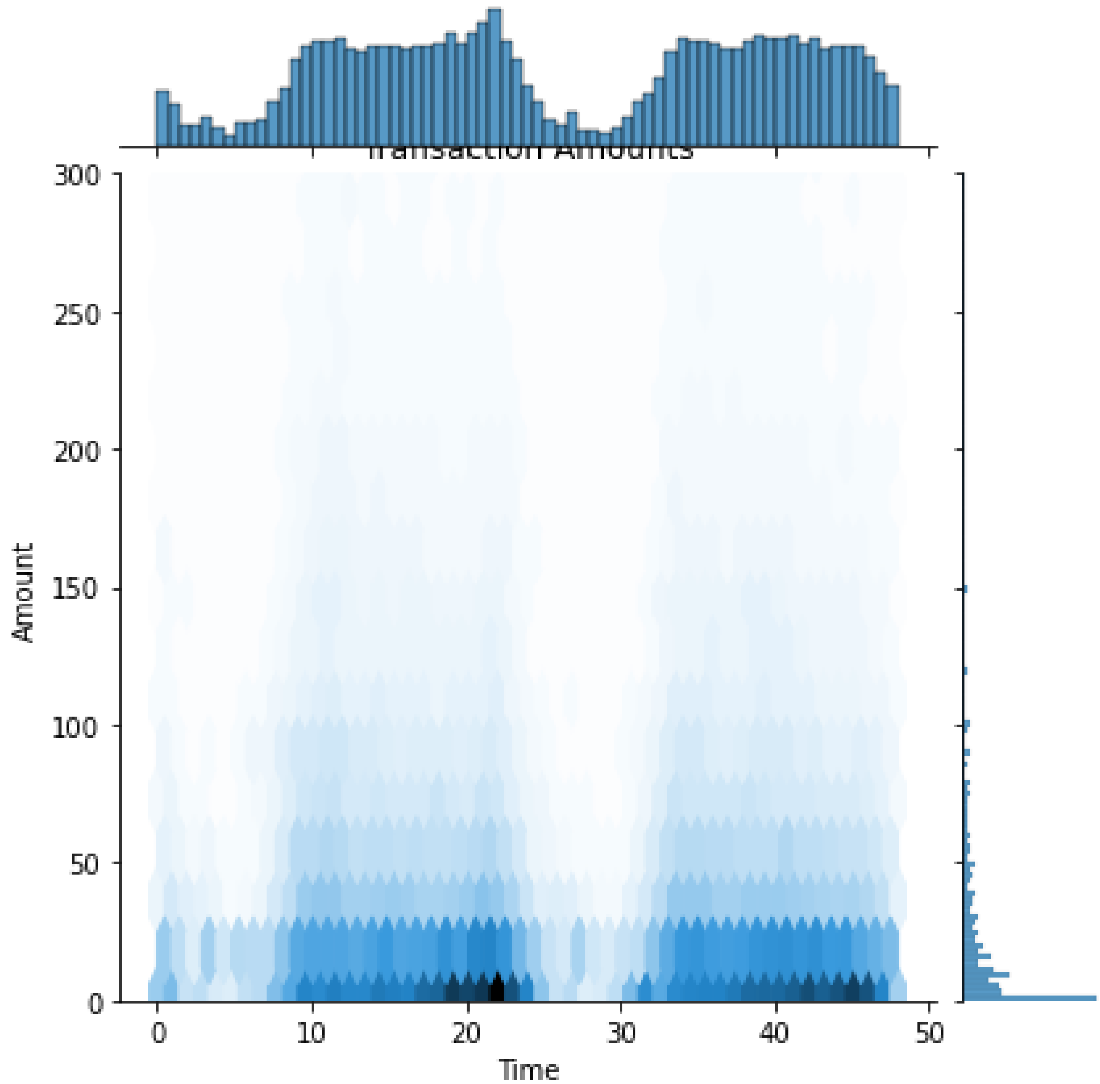
<Figure size 1080x576 with 0 Axes>



[39]:
```
plt.figure(figsize=(15,8))
sns.jointplot(x='Time', y = 'Amount', data= data, kind='hex',bins=100)
plt.ylim([0, 300])
plt.title('Transaction Amounts')
```

[39]: Text(0.5, 1.0, 'Transaction Amounts')

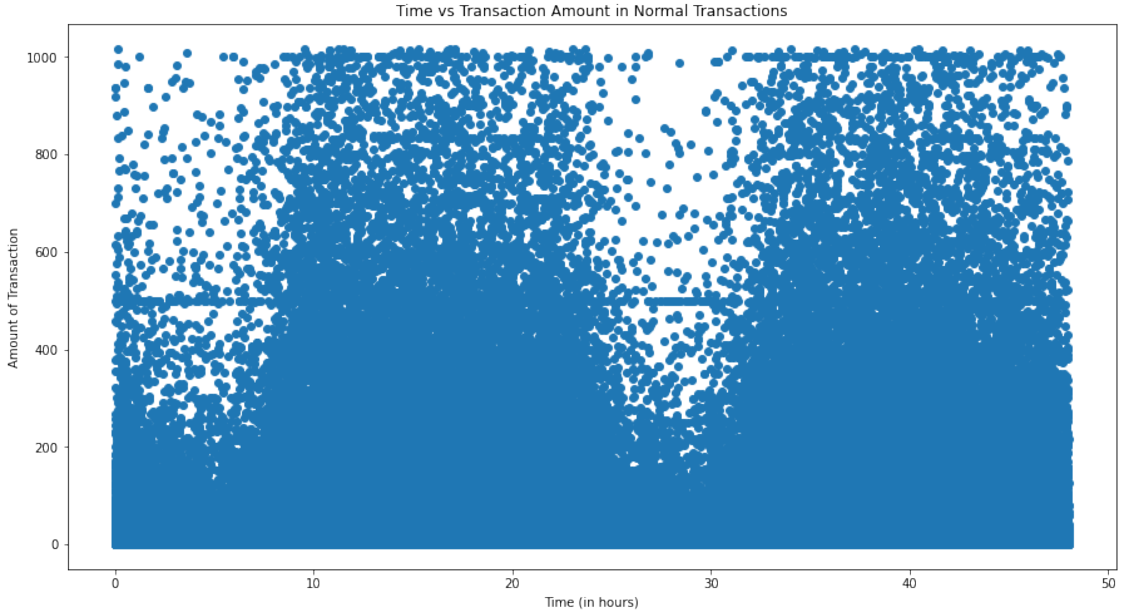
```
<Figure size 1080x576 with 0 Axes>
```



6 Scatter plot de Class vs Amount y Time para transacciones normales

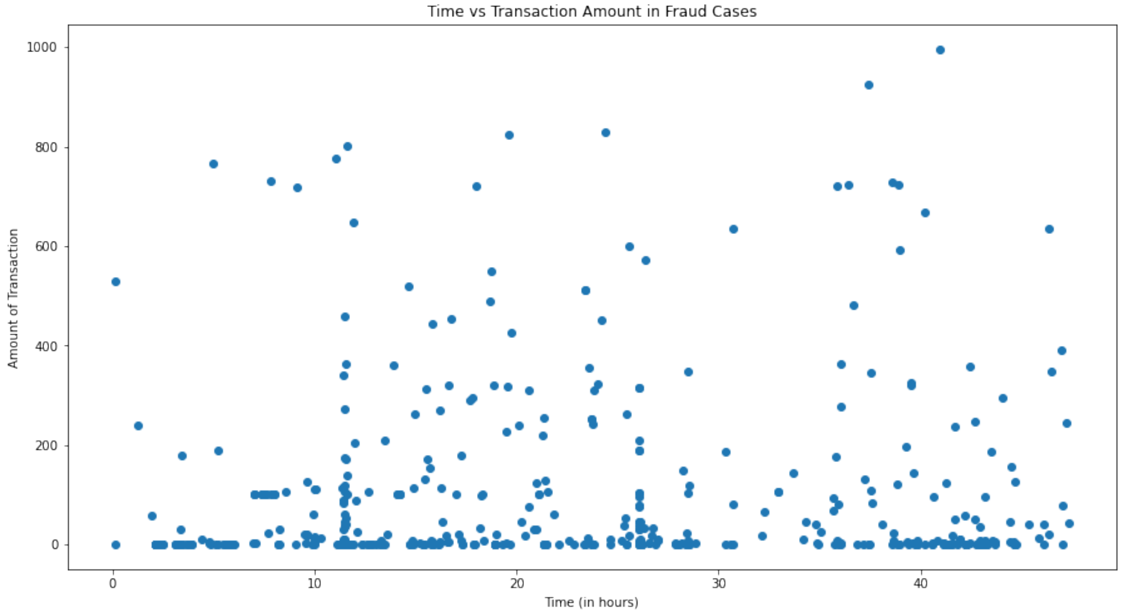
[40]:
```
plt.figure(figsize=(15,8))
fig = plt.scatter(x=data[data['Class'] == 0]['Time'], y=data[data['Class'] ==␣
↪0]['Amount'])
plt.title("Time vs Transaction Amount in Normal Transactions")
plt.xlabel("Time (in hours)")
plt.ylabel("Amount of Transaction")
```

[40]: Text(0, 0.5, 'Amount of Transaction')




```
[41]: plt.figure(figsize=(15,8))
      fig = plt.scatter(x=data[data['Class'] == 1]['Time'], y=data[data['Class'] ==␣
       ↪1]['Amount'])
      plt.title("Time vs Transaction Amount in Fraud Cases")
      plt.xlabel("Time (in hours)")
      plt.ylabel("Amount of Transaction")
```

```
[41]: Text(0, 0.5, 'Amount of Transaction')
```



Podemos notar que hay mas datos anormales en las transacciones fraudulentas comparadas con las transacciones normales.

7 V1-V28

[42]:
```
pca_vars = ['V%i' % k for k in range(1,29)]
```

[43]:
```
v1_v28 = data[pca_vars].describe()
```

[44]:
```
data[pca_vars].describe()
```

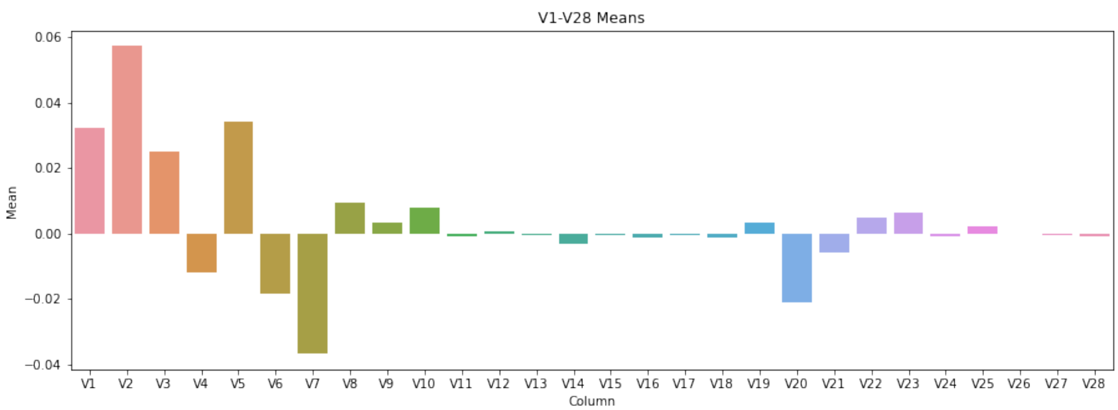
[44]:
```
                  V1             V2             V3             V4  \
count  281958.000000  281958.000000  281958.000000  281958.000000
mean        0.032412       0.057244       0.025099      -0.011762
std         1.889011       1.462565       1.480447       1.402561
min       -46.855047     -47.429676     -33.680984      -5.683171
25%        -0.900389      -0.575563      -0.861092      -0.853928
50%         0.038225       0.076168       0.194044      -0.027356
75%         1.320950       0.811145       1.036410       0.730534
max         2.454930      22.057729       9.382558      13.129143

                  V5             V6             V7             V8  \
count  281958.000000  281958.000000  281958.000000  281958.000000
mean        0.034301      -0.018477      -0.036813       0.009585
std         1.262505       1.283427       1.085699       1.180338
min       -23.669726     -23.496714     -43.557242     -73.216718
25%        -0.672125      -0.772081      -0.559532      -0.203697
50%        -0.044095      -0.280934       0.031891       0.025318
75%         0.619279       0.381228       0.554993       0.330946
max        34.099309      16.614054      15.661716      20.007208

                  V9            V10  ...            V19            V20  \
count  281958.000000  281958.000000  ...  281958.000000  281958.000000
mean        0.003263       0.007892  ...       0.003492      -0.020938
std         1.095067       1.082691  ...       0.810549       0.620143
min       -13.434066     -24.588262  ...      -4.932733     -23.420173
25%        -0.637854      -0.528076  ...      -0.451211      -0.212305
50%        -0.048919      -0.089475  ...       0.006492      -0.064456
75%         0.597817       0.458492  ...       0.461014       0.127079
max        15.594995      23.745136  ...       5.591971      16.756448

                 V21            V22            V23            V24  \
count  281958.000000  281958.000000  281958.000000  281958.000000
mean       -0.005957       0.004976       0.006338      -0.000726
```

```
std         0.717382       0.718989       0.504557       0.604855
min       -34.830382      -8.887017     -36.666000      -2.836627
25%        -0.229121      -0.538139      -0.158656      -0.355164
50%        -0.031745       0.009951      -0.010088       0.040474
75%         0.181180       0.531329       0.147358       0.438540
max        27.202839      10.503090      22.083545       4.022866

                 V25            V26            V27            V28
count  281958.000000  281958.000000  281958.000000  281958.000000
mean        0.002199      -0.000112      -0.000388      -0.000911
std         0.514396       0.480772       0.383250       0.297726
min        -7.495741      -2.068561     -22.565679     -11.710896
25%        -0.315238      -0.326429      -0.069292      -0.053002
50%         0.018177      -0.051616       0.002066       0.010831
75%         0.351447       0.239905       0.091115       0.075941
max         7.519589       3.517346       9.879903      22.620072

[8 rows x 28 columns]
```

[45]:
```
# graficamos las media para un analisi mas senillo
vs_mean =  data[pca_vars].mean()
```

[46]:
```
plt.figure(figsize=(15,5))
sns.barplot(x =pca_vars, y =vs_mean)
plt.xlabel('Column')
plt.ylabel('Mean')
plt.title('V1-V28 Means')
```

[46]: Text(0.5, 1.0, 'V1-V28 Means')



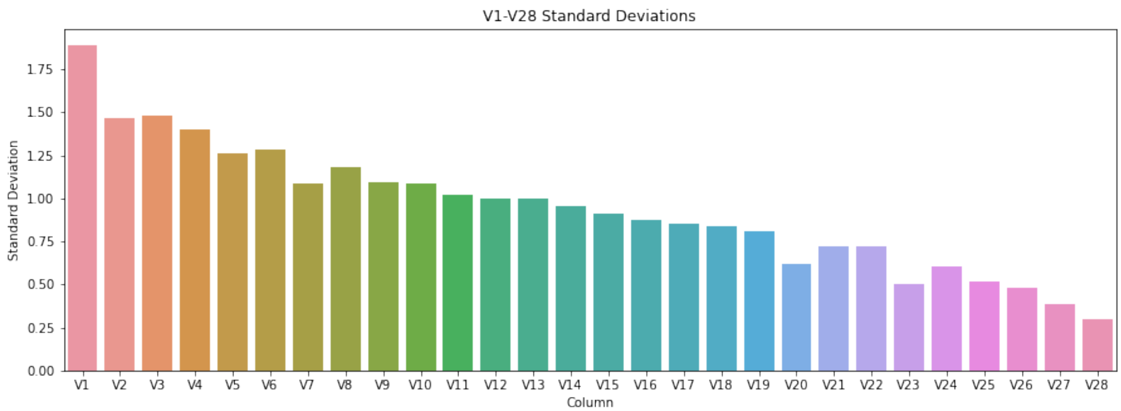
[47]:
```
data[pca_vars].mean().mean()
```

[47]: 0.0029847600034513524

La media de todos los V1-V28 son aproximadamente 0, Ahora graficamos la desviacion estandar.

[48]:
```
plt.figure(figsize=(15,5))
sns.barplot(x=pca_vars, y=data[pca_vars].std())
plt.xlabel('Column')
plt.ylabel('Standard Deviation')
plt.title('V1-V28 Standard Deviations')
```

[48]: Text(0.5, 1.0, 'V1-V28 Standard Deviations')



[49]:
```
# buscamos el valo máximo
data[pca_vars].std().max()
```

[49]: 1.8890107451038372

[50]:
```
# tambien el valor minimo
data[pca_vars].std().min()
```

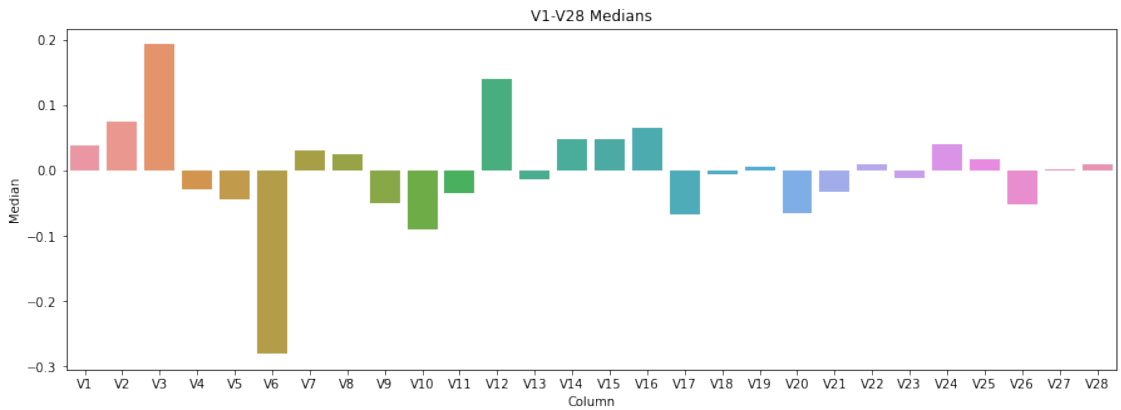
[50]: 0.297725889070566

Las variables de PCA tienen una variación unitaria aproximada, pero es tan pequeña como ~ 0.3 y tan alta como ~ 1.9. Grafiquemos las medianas:

Grafiquemos las medianas:

[51]:
```
plt.figure(figsize=(15,5))
sns.barplot(x=pca_vars, y=data[pca_vars].median())
plt.xlabel('Column')
plt.ylabel('Median')
plt.title('V1-V28 Medians')
```

[51]: Text(0.5, 1.0, 'V1-V28 Medians')



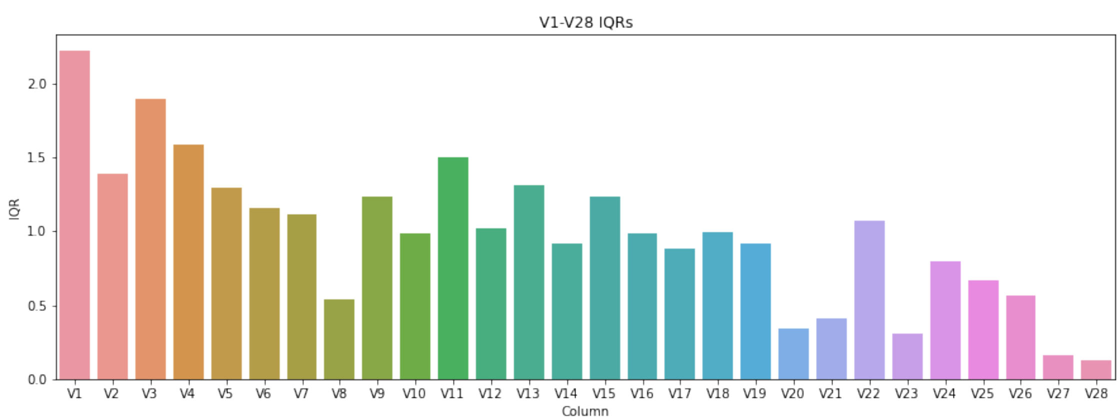
[52]:
```
data[pca_vars].median().mean()
```

[52]: -0.000368439804139721

En promedio las medianas tambien son de aproximadamente cero, a continuacion vamos a ver el IQR.

[53]:
```
plt.figure(figsize=(15,5))
sns.barplot(x=pca_vars,
            y=data[pca_vars].quantile(0.75) - data[pca_vars].quantile(0.25))
plt.xlabel('Column')
plt.ylabel('IQR')
plt.title('V1-V28 IQRs')
```

[53]: Text(0.5, 1.0, 'V1-V28 IQRs')



8 Correlaciones

Vamos a ver las correlaciones existentes entre las difeentes variables que tenemos en el dataset.

[54]:
```
data.corr()
```

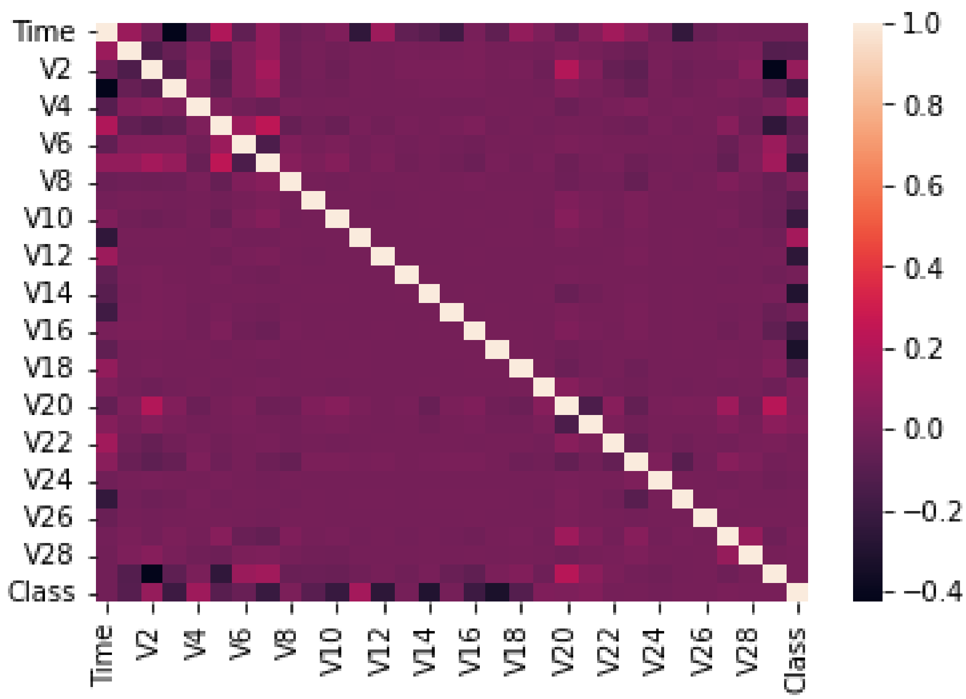
[54]:
```
            Time        V1        V2        V3        V4        V5        V6  \
Time    1.000000  0.121719 -0.013433 -0.428389 -0.106329  0.188803 -0.065923
V1      0.121719  1.000000 -0.137679 -0.053279  0.036045 -0.067225  0.036311
V2     -0.013433 -0.137679  1.000000 -0.100534  0.067348 -0.103004  0.045302
V3     -0.428389 -0.053279 -0.100534  1.000000  0.023411 -0.076402  0.043433
V4     -0.106329  0.036045  0.067348  0.023411  1.000000  0.026918 -0.013803
V5      0.188803 -0.067225 -0.103004 -0.076402  0.026918  1.000000  0.121793
V6     -0.065923  0.036311  0.045302  0.043433 -0.013803  0.121793  1.000000
V7      0.094762  0.095904  0.153701  0.103362 -0.038154  0.237086 -0.143865
V8     -0.037081 -0.025370 -0.031441 -0.025227  0.007741 -0.051169  0.031465
V9     -0.007785  0.002366 -0.003521 -0.005879 -0.000847 -0.012620  0.006214
V10     0.029072 -0.014768 -0.030542 -0.014604  0.005182 -0.038691  0.022305
V11    -0.248186  0.006206  0.006211  0.003841 -0.002061  0.007815 -0.006259
V12     0.126110 -0.000339  0.001067 -0.004323 -0.000044 -0.013836  0.008263
V13    -0.066603  0.009178  0.012366  0.005680 -0.003935  0.008121 -0.005238
V14    -0.100501  0.002635  0.017673  0.007405 -0.003182 -0.001882  0.003539
V15    -0.185172  0.011903  0.014498  0.007677 -0.005183  0.009432 -0.005944
V16     0.011652  0.019149  0.018499  0.011890 -0.007565  0.026582 -0.017893
V17    -0.072683 -0.003522  0.002157 -0.002354  0.001814 -0.005888  0.004735
V18     0.091990  0.000804  0.009030 -0.000512 -0.003064  0.002953  0.000869
V19     0.028798 -0.005669 -0.024231 -0.006913  0.005236 -0.006708  0.002147
V20    -0.057632  0.027530  0.197171  0.036976 -0.037163  0.006210  0.020485
V21     0.047460  0.002581  0.043071  0.007465 -0.006721 -0.003306  0.008007
V22     0.144904 -0.018593 -0.050173 -0.013854  0.011812 -0.011466  0.002896
V23     0.063175 -0.039700 -0.079628 -0.052067  0.027212 -0.023718  0.002433
V24    -0.015817  0.005138  0.006834  0.005255 -0.001948  0.007915 -0.002328
V25    -0.236276 -0.007436 -0.024426 -0.012884  0.008081 -0.009002  0.002844
V26    -0.041111 -0.001562 -0.005666  0.000637  0.001099  0.002472 -0.002147
V27    -0.006060  0.015215 -0.009870  0.017880  0.000047  0.062330 -0.042616
V28    -0.010592  0.025226  0.051634  0.007316 -0.015756 -0.028076  0.018487
Amount -0.015647 -0.112535 -0.428632 -0.081178  0.017183 -0.245463  0.121001
Class  -0.012379 -0.105888  0.103811 -0.198453  0.134873 -0.104286 -0.045428

              V7        V8        V9  ...       V21       V22       V23  \
Time    0.094762 -0.037081 -0.007785  ...  0.047460  0.144904  0.063175
V1      0.095904 -0.025370  0.002366  ...  0.002581 -0.018593 -0.039700
V2      0.153701 -0.031441 -0.003521  ...  0.043071 -0.050173 -0.079628
V3      0.103362 -0.025227 -0.005879  ...  0.007465 -0.013854 -0.052067
V4     -0.038154  0.007741 -0.000847  ... -0.006721  0.011812  0.027212
V5      0.237086 -0.051169 -0.012620  ... -0.003306 -0.011466 -0.023718
V6     -0.143865  0.031465  0.006214  ...  0.008007  0.002896  0.002433
V7      1.000000  0.054966  0.018041  ... -0.001647  0.009057 -0.029269
```

```
V8      0.054966  1.000000 -0.001479  ... -0.004628 -0.005725 -0.051319
V9      0.018041 -0.001479  1.000000  ...  0.007485  0.000495  0.023271
V10     0.048150 -0.007334 -0.002875  ...  0.012249 -0.004726  0.019210
V11    -0.009031  0.002639 -0.001409  ...  0.003879 -0.000148  0.010665
V12     0.018266 -0.001947 -0.004636  ... -0.001157  0.003339  0.015933
V13    -0.011821  0.001856 -0.000606  ...  0.002792  0.001609  0.004323
V14    -0.001717  0.000829  0.004216  ... -0.012455  0.004938  0.008027
V15    -0.013026  0.004671 -0.003362  ...  0.004534  0.001716  0.016136
V16    -0.032539  0.009784 -0.003294  ...  0.008946 -0.001535  0.014905
V17     0.009870 -0.002081  0.001747  ... -0.005766  0.002205  0.002129
V18    -0.005747 -0.001711  0.003258  ... -0.008843  0.003932 -0.024019
V19     0.009298 -0.001196 -0.003085  ...  0.013913 -0.009233 -0.013136
V20    -0.030519 -0.022588  0.034385  ... -0.142024  0.057220 -0.065441
V21    -0.001647 -0.004628  0.007485  ...  1.000000  0.017831 -0.014973
V22     0.009057 -0.005725  0.000495  ...  0.017831  1.000000 -0.057115
V23    -0.029269 -0.051319  0.023271  ... -0.014973 -0.057115  1.000000
V24    -0.006719  0.005585  0.000019  ...  0.003250  0.002044  0.027741
V25    -0.002235 -0.008103  0.003898  ...  0.005325 -0.015881 -0.100292
V26    -0.004629  0.001146  0.001785  ...  0.004501 -0.003554 -0.006436
V27    -0.064257  0.031007  0.000560  ...  0.032490 -0.007443  0.058902
V28     0.024464 -0.004060 -0.015194  ...  0.001580  0.009126  0.025334
Amount  0.144730 -0.038871 -0.035250  ...  0.068163  0.013205 -0.009405
Class  -0.214164  0.020532 -0.098838  ...  0.041921  0.000878  0.001860

             V24       V25       V26       V27       V28    Amount     Class
Time   -0.015817 -0.236276 -0.041111 -0.006060 -0.010592 -0.015647 -0.012379
V1      0.005138 -0.007436 -0.001562  0.015215  0.025226 -0.112535 -0.105888
V2      0.006834 -0.024426 -0.005666 -0.009870  0.051634 -0.428632  0.103811
V3      0.005255 -0.012884  0.000637  0.017880  0.007316 -0.081178 -0.198453
V4     -0.001948  0.008081  0.001099  0.000047 -0.015756  0.017183  0.134873
V5      0.007915 -0.009002  0.002472  0.062330 -0.028076 -0.245463 -0.104286
V6     -0.002328  0.002844 -0.002147 -0.042616  0.018487  0.121001 -0.045428
V7     -0.006719 -0.002235 -0.004629 -0.064257  0.024464  0.144730 -0.214164
V8      0.005585 -0.008103  0.001146  0.031007 -0.004060 -0.038871  0.020532
V9      0.000019  0.003898  0.001785  0.000560 -0.015194 -0.035250 -0.098838
V10    -0.000551 -0.000213 -0.001428  0.008284 -0.020450 -0.043390 -0.218642
V11     0.000738  0.001633 -0.001070 -0.005177 -0.003804 -0.020063  0.154896
V12     0.000362  0.003945  0.000838  0.004085 -0.004081  0.006579 -0.261096
V13    -0.001044  0.000655  0.000345 -0.004518 -0.001624 -0.018196 -0.004375
V14     0.000316  0.004226  0.000030  0.006303  0.005308  0.010685 -0.303472
V15    -0.001265  0.002466 -0.000578 -0.007242 -0.004456 -0.040485 -0.004385
V16    -0.002695  0.001389  0.001951 -0.018033  0.001409 -0.071817 -0.197696
V17     0.000801  0.001344  0.000622  0.006262  0.000740  0.016514 -0.326349
V18    -0.000423 -0.002758 -0.000409  0.008850  0.001335  0.035764 -0.111875
V19    -0.001116 -0.007379 -0.001114 -0.008156 -0.000449 -0.023044  0.034537
V20     0.010634  0.021964  0.015971  0.132456 -0.016928  0.214737  0.025631
V21     0.003250  0.005325  0.004501  0.032490  0.001580  0.068163  0.041921
```

```
V22      0.002044 -0.015881 -0.003554 -0.007443  0.009126  0.013205  0.000878
V23      0.027741 -0.100292 -0.006436  0.058902  0.025334 -0.009405  0.001860
V24      1.000000  0.002943 -0.000449 -0.009654  0.003706 -0.017712 -0.007439
V25      0.002943  1.000000 -0.003919  0.007259 -0.002766 -0.000056  0.004286
V26     -0.000449 -0.003919  1.000000 -0.004088  0.002290  0.005272  0.004696
V27     -0.009654  0.007259 -0.004088  1.000000  0.104656 -0.025165  0.016994
V28      0.003706 -0.002766  0.002290  0.104656  1.000000  0.004772  0.011205
Amount  -0.017712 -0.000056  0.005272 -0.025165  0.004772  1.000000  0.008464
Class   -0.007439  0.004286  0.004696  0.016994  0.011205  0.008464  1.000000

[31 rows x 31 columns]
```

[55]:
```python
sns.heatmap(data.corr())
```

[55]: <AxesSubplot:>



9 Modeling

[56]:
```python
#Train-Test split
```

[57]:
```python
from sklearn.model_selection import train_test_split
```

[58]:
```python
X= data.drop('Class',axis=1)
```

[59]:
```
X.head()
```

[59]:
```
       Time        V1        V2        V3        V4        V5        V6  \
0  0.000000 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388
1  0.000000  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361
2  0.000278 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499
3  0.000278 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203
4  0.000556 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921

         V7        V8        V9  ...       V20       V21       V22       V23  \
0  0.239599  0.098698  0.363787  ...  0.251412 -0.018307  0.277838 -0.110474
1 -0.078803  0.085102 -0.255425  ... -0.069083 -0.225775 -0.638672  0.101288
2  0.791461  0.247676 -1.514654  ...  0.524980  0.247998  0.771679  0.909412
3  0.237609  0.377436 -1.387024  ... -0.208038 -0.108300  0.005274 -0.190321
4  0.592941 -0.270533  0.817739  ...  0.408542 -0.009431  0.798278 -0.137458

        V24       V25       V26       V27       V28  Amount
0  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62
1 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69
2 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50
4  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99

[5 rows x 30 columns]
```

[60]:
```
y = data['Class']
```

[61]:
```
y.head()
```

[61]:
```
0    0
1    0
2    0
3    0
4    0
Name: Class, dtype: int64
```

[62]:
```
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size=0.25,␣
 ↪random_state=0)
```

Uaremos el 75% de los datos para entrenamiento y el 25% para pruebas.

[63]:
```
X_train.shape
```

[63]:
```
(211468, 30)
```

[64]:
```
X_test.shape
```

[64]:
```
(70490, 30)
```

[65]:
```
y_train.shape
```

[65]: (211468,)

[66]:
```
y_test.shape
```

[66]: (70490,)

[67]:
```
import statsmodels.api as sn
from sklearn.linear_model import LinearRegression
```

[68]:
```
X_cons = sn.add_constant(X)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

[69]:
```
X_cons.head()
```

[69]:
```
   const      Time        V1        V2        V3        V4        V5  \
0    1.0  0.000000 -1.359807 -0.072781  2.536347  1.378155 -0.338321
1    1.0  0.000000  1.191857  0.266151  0.166480  0.448154  0.060018
2    1.0  0.000278 -1.358354 -1.340163  1.773209  0.379780 -0.503198
3    1.0  0.000278 -0.966272 -0.185226  1.792993 -0.863291 -0.010309
4    1.0  0.000556 -1.158233  0.877737  1.548718  0.403034 -0.407193

         V6        V7        V8  ...       V20       V21       V22       V23  \
0  0.462388  0.239599  0.098698  ...  0.251412 -0.018307  0.277838 -0.110474
1 -0.082361 -0.078803  0.085102  ... -0.069083 -0.225775 -0.638672  0.101288
2  1.800499  0.791461  0.247676  ...  0.524980  0.247998  0.771679  0.909412
3  1.247203  0.237609  0.377436  ... -0.208038 -0.108300  0.005274 -0.190321
4  0.095921  0.592941 -0.270533  ...  0.408542 -0.009431  0.798278 -0.137458

        V24       V25       V26       V27       V28  Amount
0  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62
1 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69
2 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50
4  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99

[5 rows x 31 columns]
```

[70]:
```
lm = sn.OLS(y, X_cons).fit()
```

[71]:
```
lm.summary()
```

[71]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Class   R-squared:                       0.523
Model:                            OLS   Adj. R-squared:                  0.523
Method:                 Least Squares   F-statistic:                 1.030e+04
Date:                Tue, 19 Oct 2021   Prob (F-statistic):               0.00
Time:                        15:28:10   Log-Likelihood:             6.0243e+05
No. Observations:              281958   AIC:                        -1.205e+06
Df Residuals:                  281927   BIC:                        -1.204e+06
Df Model:                          30
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0011      0.000      6.123      0.000       0.001       0.001
Time        -1.51e-05     5.6e-06    -2.699      0.007   -2.61e-05   -4.14e-06
V1            -0.0018   3.96e-05    -46.621      0.000      -0.002      -0.002
V2             0.0031   8.18e-05     38.221      0.000       0.003       0.003
V3            -0.0050     5.2e-05    -95.682      0.000      -0.005      -0.005
V4             0.0037   4.19e-05     88.839      0.000       0.004       0.004
V5            -0.0021   7.55e-05    -28.089      0.000      -0.002      -0.002
V6            -0.0018   5.54e-05    -31.765      0.000      -0.002      -0.002
V7            -0.0070   8.67e-05    -81.128      0.000      -0.007      -0.007
V8             0.0009   4.91e-05     17.794      0.000       0.001       0.001
V9            -0.0036      5e-05    -71.816      0.000      -0.004      -0.003
V10           -0.0080   5.38e-05   -148.475      0.000      -0.008      -0.008
V11            0.0062   5.58e-05    111.803      0.000       0.006       0.006
V12           -0.0107   5.47e-05   -196.458      0.000      -0.011      -0.011
V13           -0.0002   5.43e-05     -3.647      0.000      -0.000   -9.16e-05
V14           -0.0132   5.73e-05   -229.827      0.000      -0.013      -0.013
V15           -0.0002   6.08e-05     -3.337      0.001      -0.000   -8.38e-05
V16           -0.0092   6.21e-05   -148.474      0.000      -0.009      -0.009
V17           -0.0159   6.37e-05   -250.188      0.000      -0.016      -0.016
V18           -0.0056   6.58e-05    -85.139      0.000      -0.006      -0.005
V19            0.0019   6.81e-05     28.317      0.000       0.002       0.002
V20           -0.0001      0.000     -0.771      0.441      -0.000       0.000
V21            0.0019   8.25e-05     23.291      0.000       0.002       0.002
V22            0.0003   7.87e-05      3.506      0.000       0.000       0.000
V23            0.0002      0.000      1.890      0.059   -7.78e-06       0.000
V24           -0.0005   8.91e-05     -5.646      0.000      -0.001      -0.000
V25            0.0004      0.000      3.240      0.001       0.000       0.001
V26            0.0004      0.000      3.631      0.000       0.000       0.001
V27            0.0017      0.000     11.856      0.000       0.001       0.002
V28            0.0012      0.000      6.634      0.000       0.001       0.002
Amount      1.219e-05   9.83e-07     12.399      0.000    1.03e-05    1.41e-05
```

```
==============================================================================
Omnibus:                   590571.822   Durbin-Watson:                   1.966
Prob(Omnibus):                  0.000   Jarque-Bera (JB):       8650260111.411
Skew:                          17.570   Prob(JB):                         0.00
Kurtosis:                     860.361   Cond. No.                         610.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

[72]:
```
lm_a =LinearRegression()
```

[73]:
```
lm_a.fit(X_train, y_train)
```

[73]:
```
LinearRegression()
```

[74]:
```
y_test_a = lm_a.predict(X_test)
```

[75]:
```
y_train_a = lm_a.predict(X_train)
```

[76]:
```
from sklearn.metrics import r2_score
```

[77]:
```
r2_score(y_test, y_test_a)
```

[77]:
```
0.5427028695528663
```

[78]:
```
r2_score(y_train, y_train_a)
```

[78]:
```
0.5153711917156873
```

[79]:
```
# Trainning classification tree
from sklearn import tree
```

[80]:
```
clftree = tree.DecisionTreeClassifier(max_depth = 3)
```

[81]:
```
clftree.fit(X_train, y_train)
```

[81]:
```
DecisionTreeClassifier(max_depth=3)
```

[82]:
```
y_train_pred = clftree.predict(X_train)
```

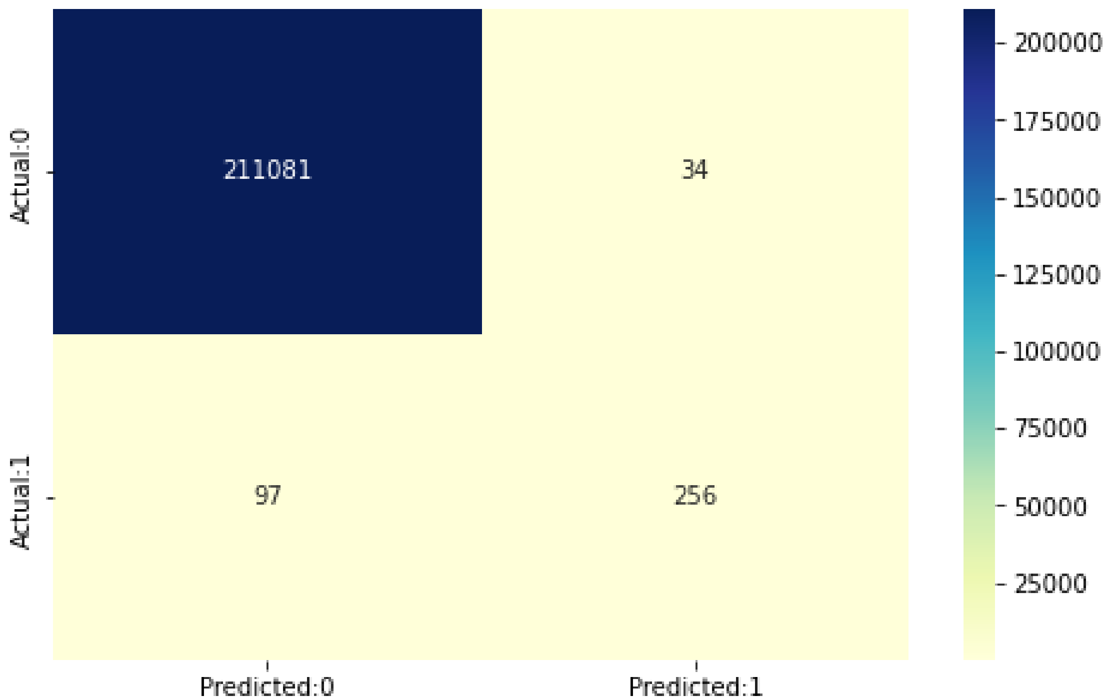
[83]:
```
y_test_pred = clftree.predict(X_test)
```

[84]:
```
# model performance
from sklearn.metrics import accuracy_score, confusion_matrix
```

[85]:
```
cmy_train = confusion_matrix(y_train, y_train_pred)
```

[86]:
```
conf_matrix=pd.DataFrame(data=cmy_train,columns=['Predicted:0','Predicted:
↪1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

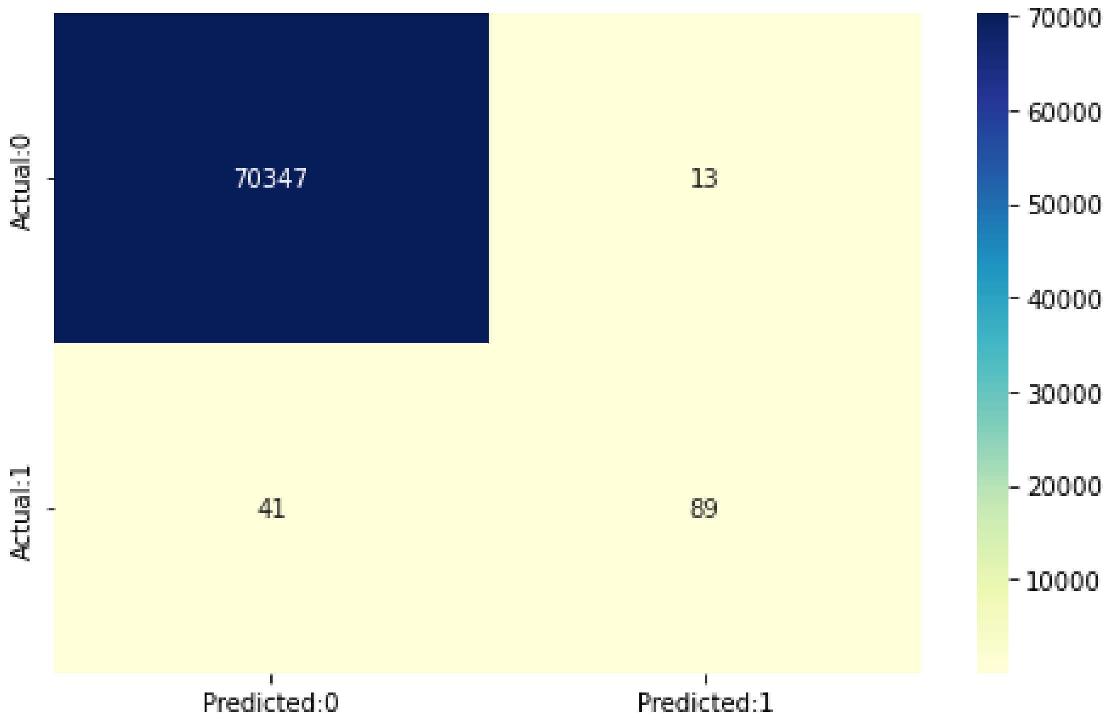
[86]: <AxesSubplot:>



[87]:
```
cmy_test =confusion_matrix(y_test, y_test_pred)
```

[88]:
```
conf_matrix=pd.DataFrame(data=cmy_test,columns=['Predicted:0','Predicted:
↪1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

[88]: <AxesSubplot:>



[89]:
```
accuracy_score(y_test, y_test_pred)
```

[89]: 0.9992339338913321

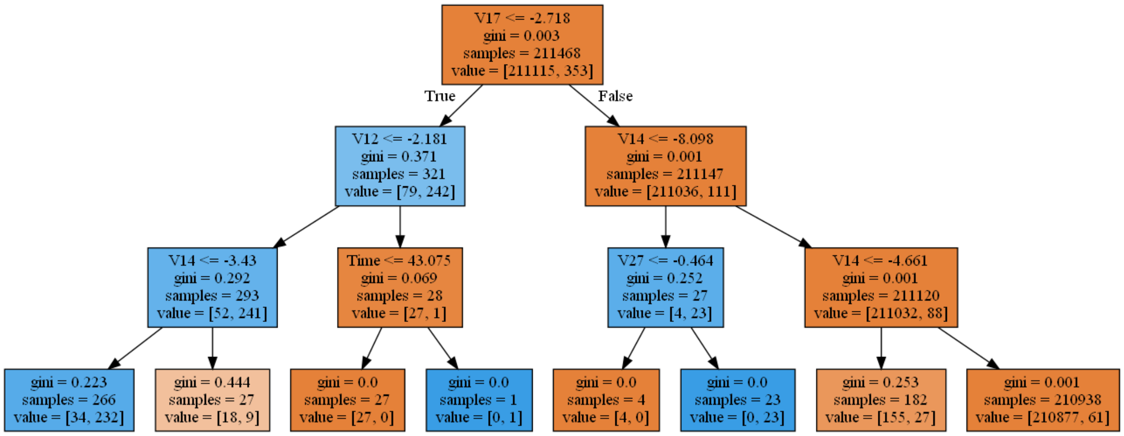
[90]:
```
# plotting decission tree
dot_data = tree.export_graphviz(clftree, out_file=None, feature_names=X_train.
 ↪columns, filled=True)
```

[91]:
```
from IPython.display import Image
import pydotplus
```

[92]:
```
graph =pydotplus.graph_from_dot_data(dot_data)
```

[93]:
```
Image(graph.create_png())
```

[93]:



[94]:
```
#controlling tree growth
```

[95]:
```
clftree2 = tree.DecisionTreeClassifier(min_samples_leaf = 20, max_depth=4)
```

[96]:
```
clftree2.fit(X_train , y_train)
```

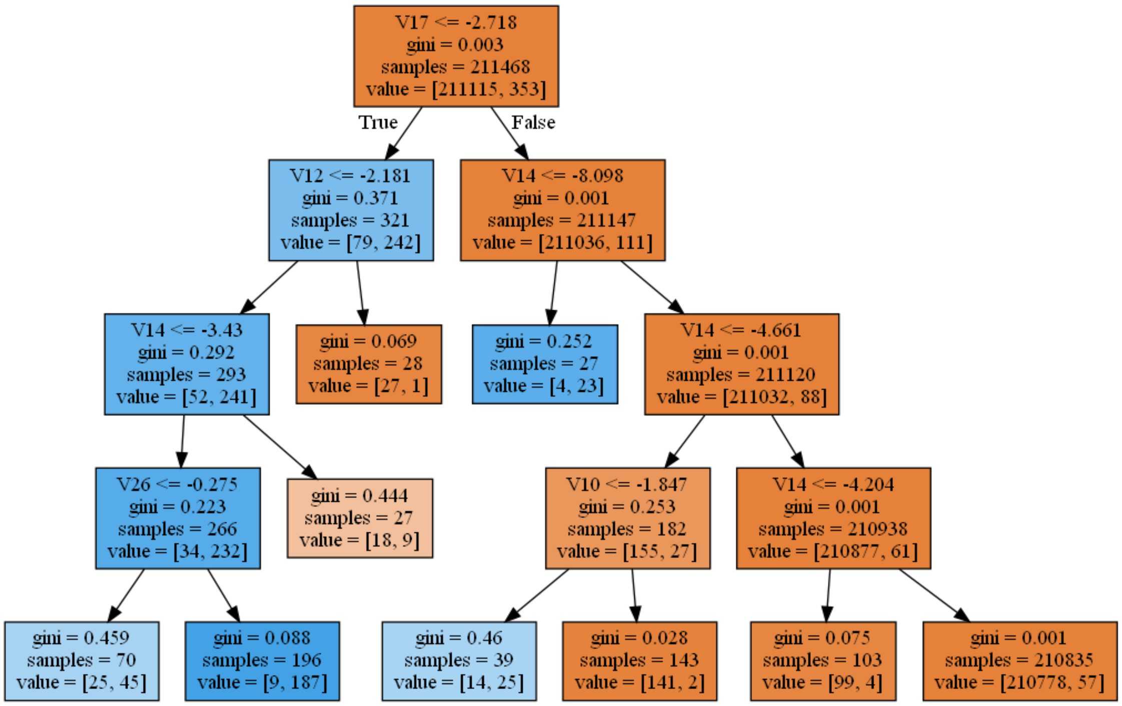
[96]:
```
DecisionTreeClassifier(max_depth=4, min_samples_leaf=20)
```

[97]:
```
dot_data = tree.export_graphviz(clftree2, out_file=None, feature_names=X_train.
 ↪columns, filled=True)
```

[98]:
```
graph2 =pydotplus.graph_from_dot_data(dot_data)
```

[99]:
```
Image(graph2.create_png())
```

[99]:



[100]:
```
accuracy_score(y_test, clftree2.predict(X_test))
```

[100]: 0.9993332387572705

[101]:
```
# Random forest
from sklearn.ensemble import RandomForestClassifier
```

[102]:
```
rf_clf = RandomForestClassifier(n_estimators = 50, n_jobs = -1, random_state = 1)
```

[103]:
```
rf_clf.fit(X_train, y_train)
```

[103]: RandomForestClassifier(n_estimators=50, n_jobs=-1, random_state=1)

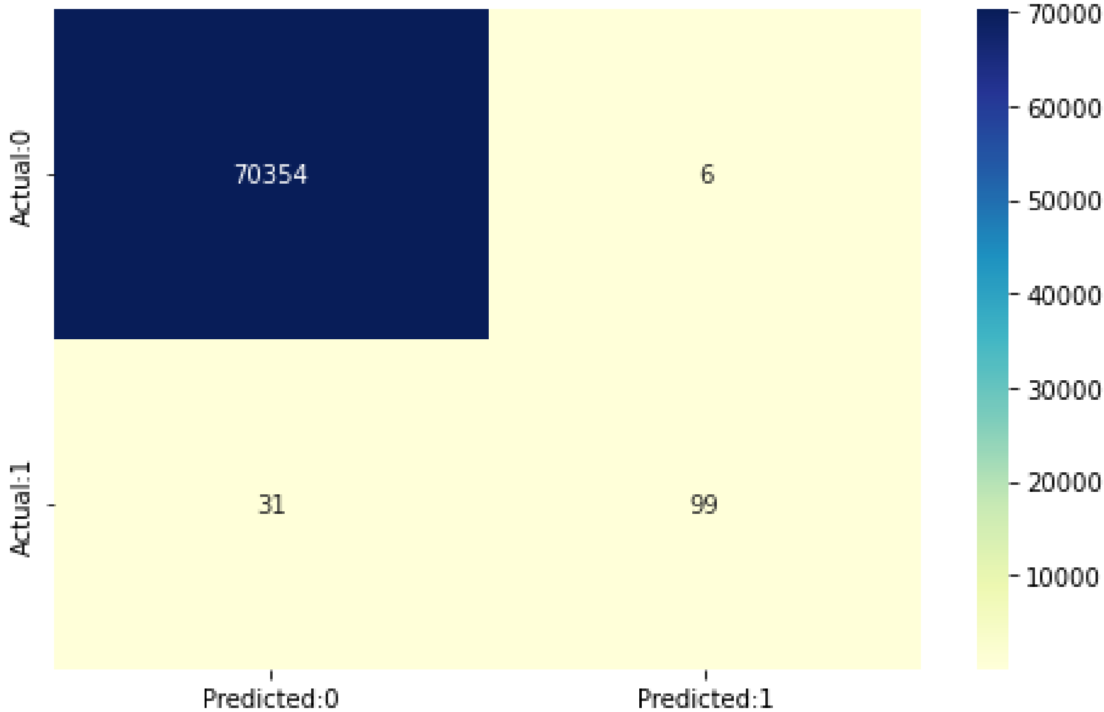
[104]:
```
rf_clf.predict(X_train)
```

[104]: array([0, 0, 0, ..., 0, 0, 0], dtype=int64)

[105]:
```
cm_rf =confusion_matrix(y_test, rf_clf.predict(X_test))
```

[106]:
```
conf_matrix=pd.DataFrame(data=cm_rf,columns=['Predicted:0','Predicted:
↪1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

[106]: <AxesSubplot:>



[107]:
```
accuracy_score(y_test, rf_clf.predict(X_test))
```

[107]: 0.9994751028514683

[108]:
```
# Grid search
from sklearn.model_selection import GridSearchCV
```

[109]:
```
rf_clf = RandomForestClassifier(n_estimators = 20, random_state = 1)
```

[110]:
```
params_grid = {'max_features': [3,4,5],
               'min_samples_split': [2,3,5]}
```

[111]:
```
grid_search = GridSearchCV(rf_clf, params_grid,
                           n_jobs=-1, cv=5, scoring='accuracy')
```

[112]:
```
grid_search.fit(X_train, y_train)
```

[112]:
```
GridSearchCV(cv=5,
             estimator=RandomForestClassifier(n_estimators=20, random_state=1),
             n_jobs=-1,
             param_grid={'max_features': [3, 4, 5],
                         'min_samples_split': [2, 3, 5]},
             scoring='accuracy')
```

[113]:
```
grid_search.best_params_
```

[113]: {'max_features': 4, 'min_samples_split': 5}

[114]:
```
cvrf_clf = grid_search.best_estimator_
```

[115]:
```
cvrf_clf
```

[115]:
```
RandomForestClassifier(max_features=4, min_samples_split=5, n_estimators=20,
                       random_state=1)
```

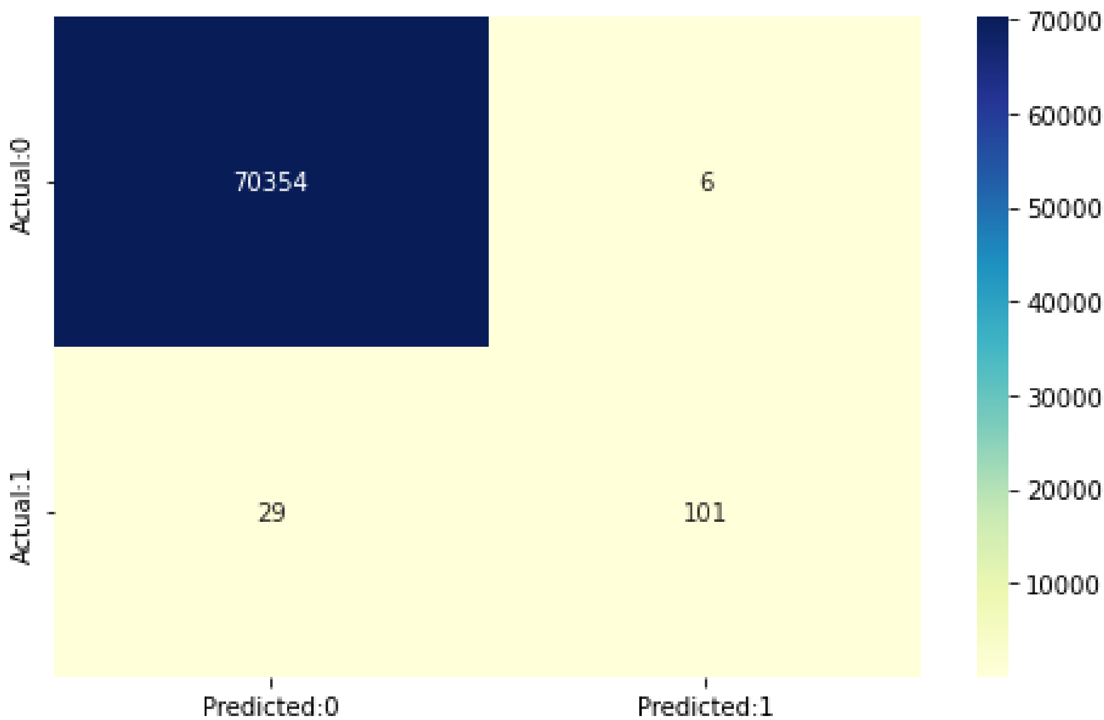
[116]:
```
accuracy_score(y_test, cvrf_clf.predict(X_test))
```

[116]: 0.9995034756703078

[117]:
```
cm_brf = confusion_matrix(y_test, cvrf_clf.predict(X_test))
```

[118]:
```
conf_matrix=pd.DataFrame(data=cm_brf ,columns=['Predicted:0','Predicted:
↪1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

[118]: <AxesSubplot:>



Podemos ver que el accuracy_score de la regresion logistica es mucho menor a la de random fores con con los mejores parametros, siendo la primer puntuación de 54.27028695528663% comparado con el 99.95034756703078%, notamos que hay una diferencia de 45.680060611744146 puntos porcentuales entre cada modelo, esto nos indica que el mejor modelo par atratar este problema fue el de random forest, controlando el crecimiento de los arboles y dando varios valores de max_features y min_samples_split, notamos que a artir de que aplicamos el modelo de arboles de decisión, el accuracy_score, aumenta lentamente aproximadamente en un 0.01% por cada vez que mejoramos el modelo.

[ ]:

```

```