Máster propio en Data Science & Big Data

Trabajo fin de máster

Clientes potencialmente fraudulentos I (Admiral 3)

Isaac López Jiménez
28-2-2019

1. Introducción y objetivo

Trabajamos en una empresa de seguros en el departamento de Data Science y nuestras funciones son las de limpiar y preparar los datos para su uso en modelos prescriptivos y predictivos para poder crear valor a partir de los datos. Nuestro proyecto actual es el de 'Clientes potencialmente fraudulentos'. Disponemos del histórico de pólizas en las que ha habido indicios de fraude en la contratación de la póliza. El conjunto de datos es bastante amplio por lo que para abordar este problema necesitaremos utilizar técnicas de Data Science (limpiar datos, imputar valores perdidos, modelar...). El objetivo de este proyecto es construir un modelo que ayude a estimar la probabilidad de que una solicitud de precio sea potencialmente fraudulenta.

2. Aproximación elegida

2.1. EDA (Análisis exploratorio de los datos)

En este proyecto trabajaremos con Python y sus librerías. En primer lugar, cargamos los datos desde el archivo csv 'tfm_fraude_I_tramas.csv'. La columna 'ref' serán los índices de nuestro dataframe.

```
In [2]: df = pd.read_csv('tfm_fraude_I_tramas.csv', low_memory=False, index_col = 'ref')
        df.head()
```

Out[2]:

ref	V213	V208	V205	V206	V209	V183	V203	V191	V212	V210	...	X5	X6	X17	X21	X18	X19	X20	X10	X15	y
201390679	0	1	1	1	WEB	014	0.0	7.0	199.0	CO	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0
201390959	0	1	1	1	WEB	ESPA	0.0	4.0	688.0	DI	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0
201397626	11	1	1	1	AGG	ESPA	0.0	11.0	1140.0	SO	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0
201397742	11	1	1	1	WEB	ESPA	99.0	8.0	872.0	CA	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0
201401345	12	1	1	1	AGG	ESPA	0.0	8.0	1143.0	CO	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0

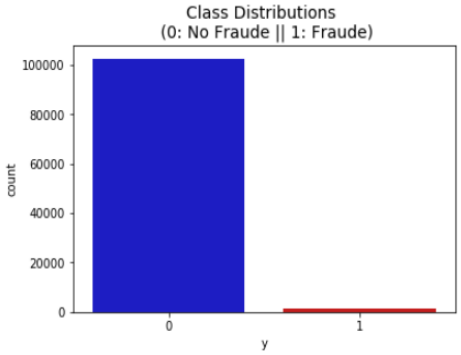
5 rows × 123 columns

Eliminamos las variables **X5, X6, X15, X10 y V162**. Tenemos un dataframe de 103914 registros y 118 columnas. Tenemos por tanto 118 variables contando con la variable respuesta. La variable respuesta **y** es de tipo binaria, con 1 hay indicio de fraude y con 0 no.

Las variables son de tipo:

float64	62
int64	33
object	23

Nos fijamos en la variable respuesta **y**:



Está claramente desbalanceado. No Fraude 98.83 % del dataset. Fraude 1.17 % del dataset. Aplicaremos técnicas de balanceo de los datos.

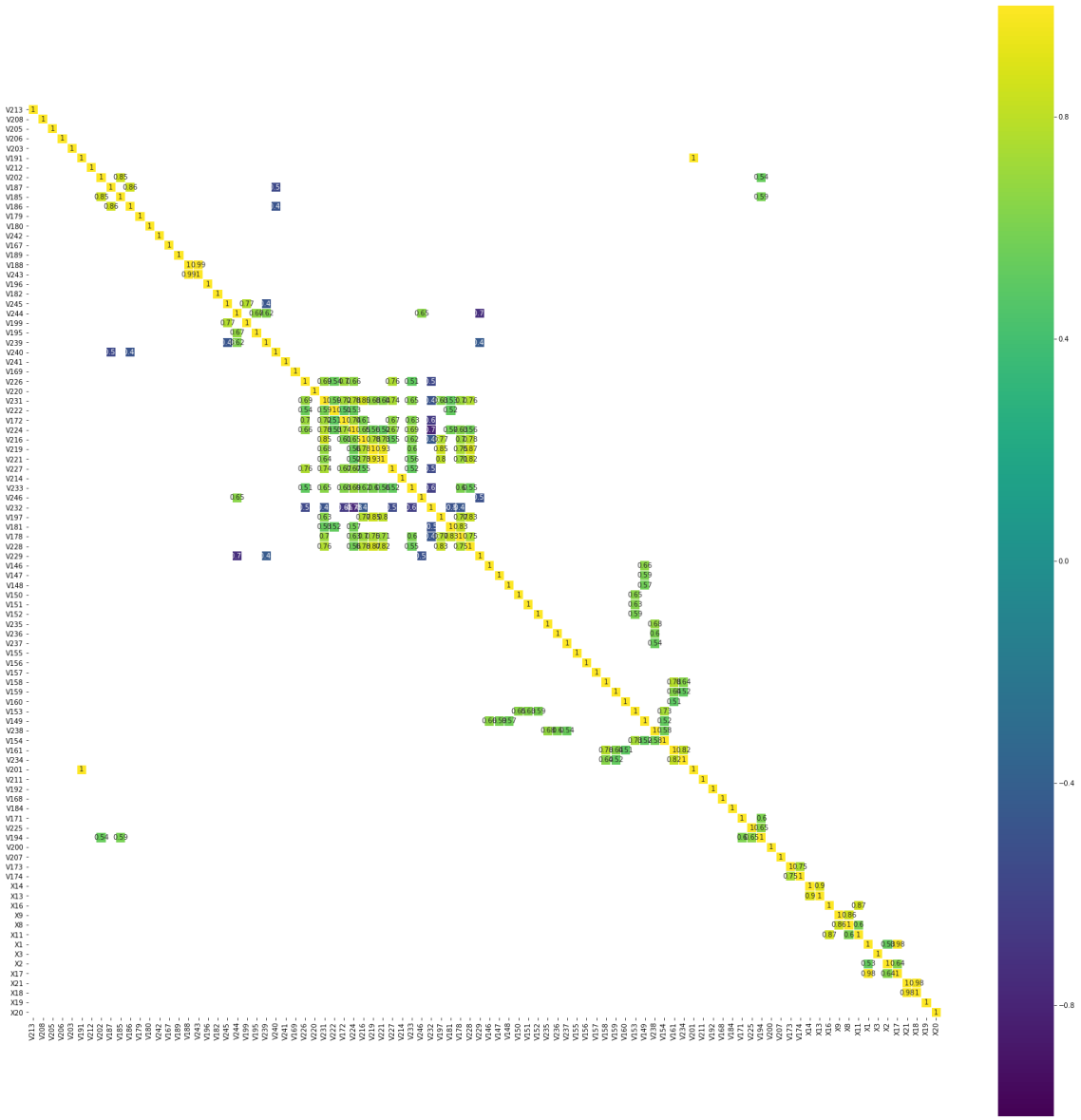
Nos fijaremos ahora en los valores perdidos.

Out[11]:

	Missing Values	% of Total Values
X19	103857	99.9
X20	103796	99.9
X2	79851	76.8
X17	79851	76.8
X1	79851	76.8
X3	79851	76.8
X4	79851	76.8
X12	79848	76.8
X14	79848	76.8
X13	79848	76.8
X16	79848	76.8
X9	79848	76.8
X8	79848	76.8
X11	79848	76.8
X7	79848	76.8
X21	79848	76.8
X18	79848	76.8
V176	72038	69.3
V186	71250	68.6
V187	71246	68.6
V240	71243	68.6
V241	71243	68.6
V211	31847	30.6
V198	24563	23.6
V243	7158	6.9
V200	5491	5.3
V246	221	0.2
V230	193	0.2
...	...	...
V239	4	0.0
V191	4	0.0
V212	4	0.0
V210	4	0.0
V202	4	0.0
V190	4	0.0
V242	4	0.0
V167	4	0.0
V188	4	0.0
V193	4	0.0

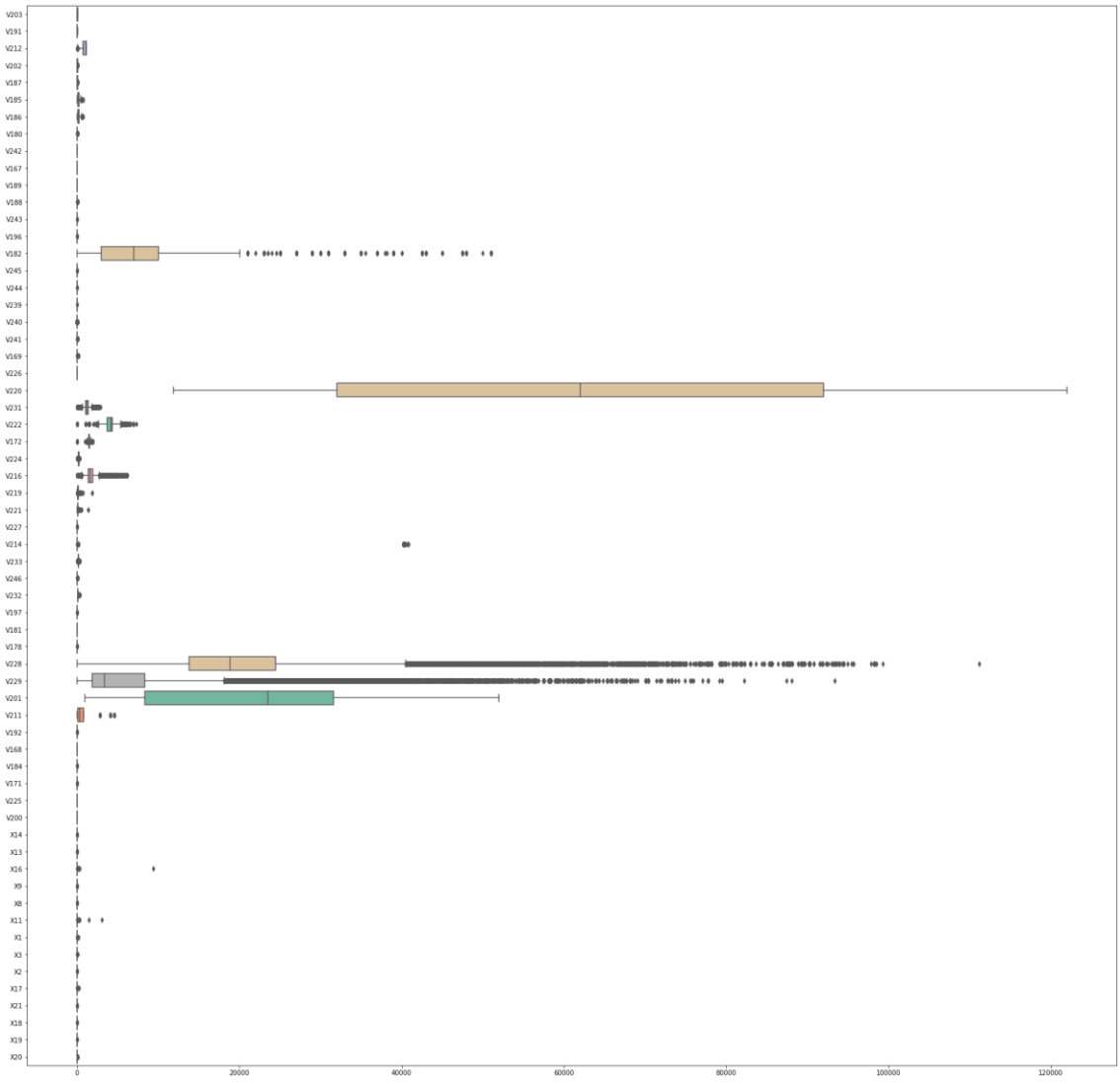
De las 118 variables, hay 77 que tienen valores perdidos (NaN). Podemos observar que algunas variables tienen casi todos sus valores como NaN. Veremos como imputar los valores perdidos.

Matriz de correlaciones:

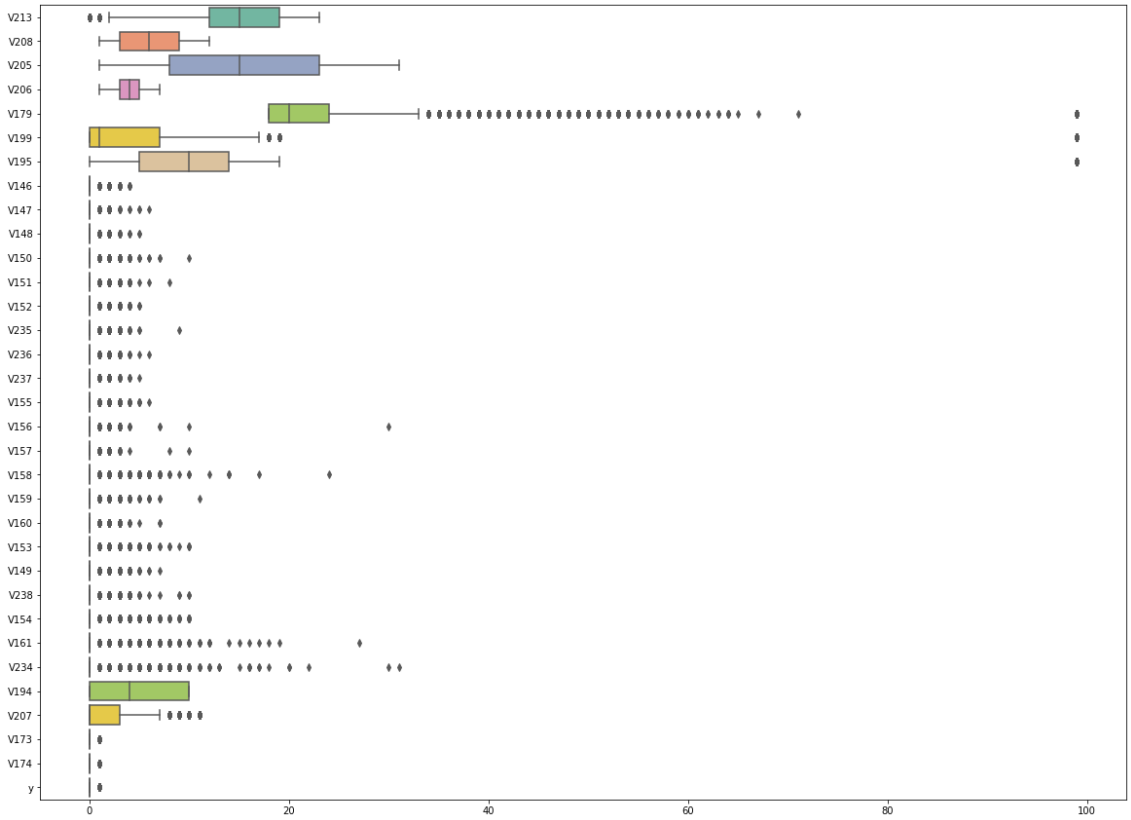


Para estudiar los outliers y el resumen de las variables, separamos el dataframe en tres. Uno con las variables float64, otro con int64 y el ultimo de tipo object.

Gráfica outiers de tipo float64:



Gráfica de outliers de variables de tipo int64:



Existen valores outliers. Tenemos que tener en cuenta que pueden existir datos de tipo numérico que en realidad representen categorías.

2.2. Preprocesado

Una vez realizado todo el análisis previo pasamos a la limpieza de los datos, de los valores atípicos y de los valores perdidos.

Valores perdidos

Si eliminásemos los registros que alguna de sus variables contenga un valor pedido, perderíamos muchísima información ya que hay variables que tienen casi todos sus valores NaN. Por tanto, lo que haremos será sustituir los valores NaN por su moda o su mediana, según sean las variables categóricas o no.

Para las variables de tipo object creamos un diccionario cuyas claves sean los nombres de las variables de tipo object y los valores serán la moda de la respectiva variable. Las variables que tengan la mayoría de sus valores NaN, las cuáles son 'V176', 'X12', 'X4' y 'X7', les cambiamos su valor en el diccionario por ‘No dispone’. Con la función fillna sustituimos los valores perdidos por los del diccionario.

Para las variables de tipo float64 el proceso es prácticamente igual. Creamos un diccionario donde las claves serán los nombres de las variables de tipo float64 y los valores serán la mediana de las variables. Las variables 'X11', 'X16', 'V187',

'V186', 'V180', 'V189', 'V240', 'V241', 'X14', 'X13', 'X9', 'X8', 'X1', 'X3', 'X2', 'X17', 'X21', 'X18', 'X19' y 'X20' tienen la mayoría de sus valores NaN, por lo que en el diccionarios modificamos su valor por 9999. Utilizando la misma función, fillna, sustituimos los NaN de las variables float64 por los valores del diccionario.

Las variables que son de tipo int64 no tienen valores perdidos NaN.

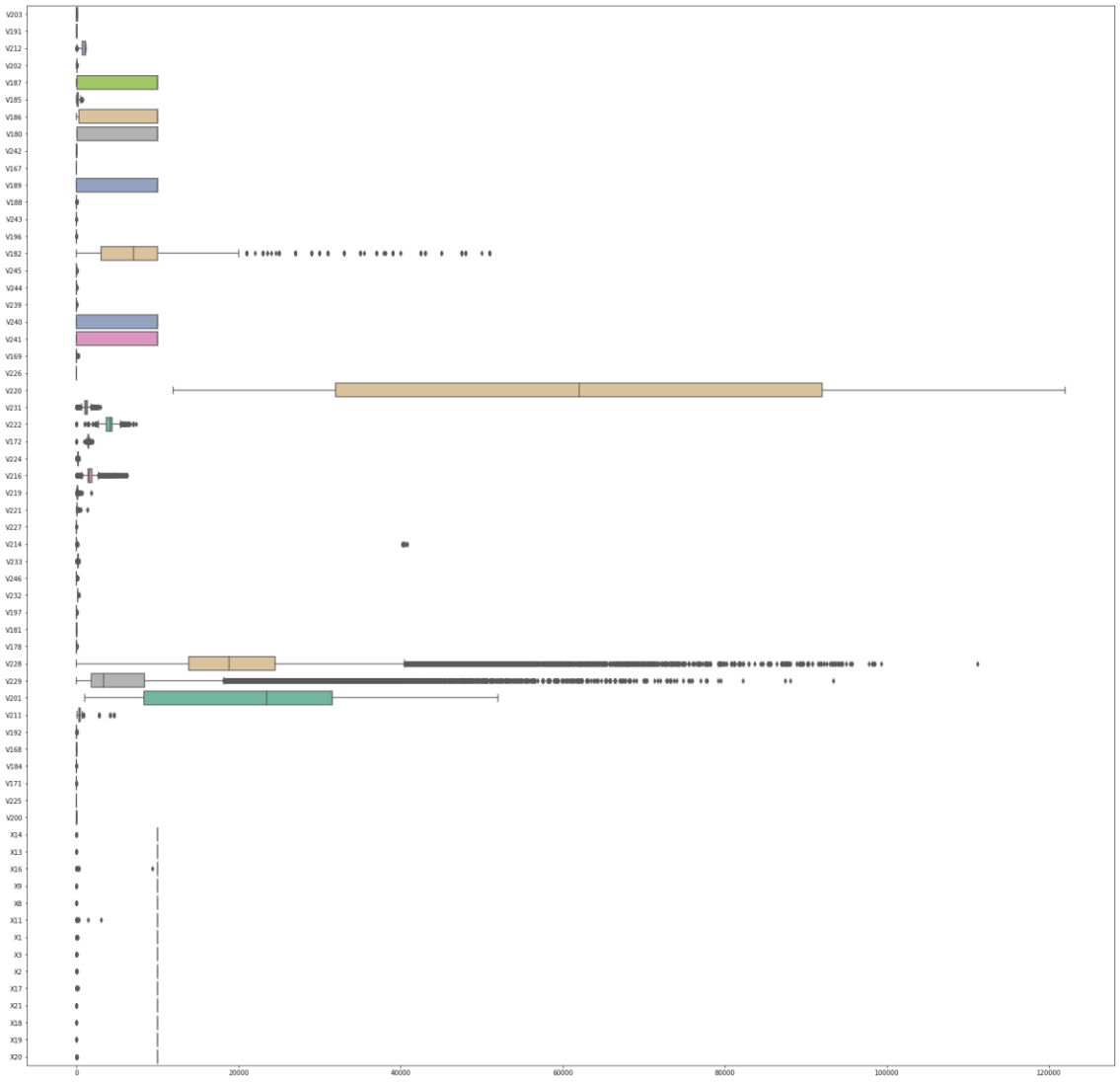
```
In [41]: missing_values = missing_values_table(df)
         missing_values

         Your selected dataframe has 118 columns.
         There are 0 columns that have missing values.

Out[41]:
```

	Missing Values	% of Total Values

Out[37]:

ref	V213	V208	V205	V206	V209	V183	V203	V191	V212	V210	...	X1	X3	X2	X4	X17	X21	X18	X19	X20	y
201390679	0	1	1	1	WEB	014	0.0	7.0	199.0	CO	...	9999.0	9999.0	9999.0	No dispone	9999.0	9999.0	9999.0	9999.0	9999.0	0
201390959	0	1	1	1	WEB	ESPA	0.0	4.0	688.0	DI	...	9999.0	9999.0	9999.0	No dispone	9999.0	9999.0	9999.0	9999.0	9999.0	0
201397626	11	1	1	1	AGG	ESPA	0.0	11.0	1140.0	SO	...	9999.0	9999.0	9999.0	No dispone	9999.0	9999.0	9999.0	9999.0	9999.0	0
201397742	11	1	1	1	WEB	ESPA	99.0	8.0	872.0	CA	...	9999.0	9999.0	9999.0	No dispone	9999.0	9999.0	9999.0	9999.0	9999.0	0
201401345	12	1	1	1	AGG	ESPA	0.0	8.0	1143.0	CO	...	9999.0	9999.0	9999.0	No dispone	9999.0	9999.0	9999.0	9999.0	9999.0	0
201402181	12	1	1	1	WEB	077	0.0	28.0	928.0	DI	...	9999.0	9999.0	9999.0	No dispone	9999.0	9999.0	9999.0	9999.0	9999.0	0
201403985	13	1	1	1	WEB	ESPA	0.0	28.0	355.0	CA	...	9999.0	9999.0	9999.0	No dispone	9999.0	9999.0	9999.0	9999.0	9999.0	0

Outliers

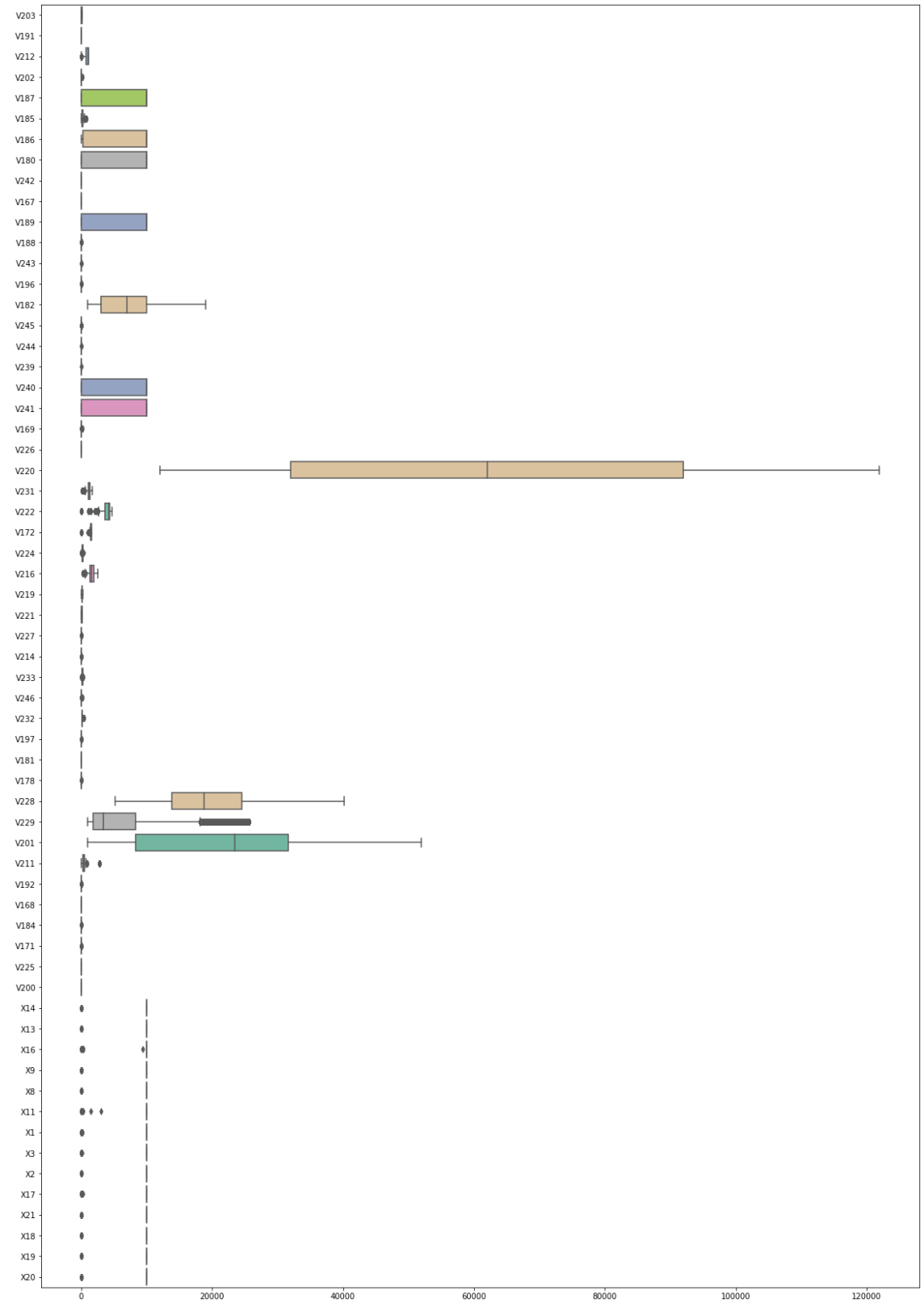


Fijándonos en la gráfica, las variables que habría que corregir los valores atípicos son:

outl = ['V182', 'V231', 'V222', 'V172', 'V216', 'V219', 'V221', 'V214', 'V220', 'V211', 'V228']

Teniendo en cuenta que hay datos de tipo numérico que pueden representar categorías, sólo imputaremos los que son de variables de tipo float64 y las que hemos indicado. Igual que hemos hecho con los valores perdidos, sustituiremos los valores atípicos por el percentil 5 (si son menores que este valor) y por el percentil 95 (si son mayores que este valor).

	0.05	0.95
V182	1000.00	19000.0000
V231	166.00	1705.0000
V222	0.00	4733.0000
V172	0.00	1578.0000
V216	295.65	2495.0000
V219	22.00	190.0000
V221	0.00	138.0000
V214	0.00	15.9000
V220	12003.0	121994.0000
V211	37.00	2814.0000
V228	5199.00	40200.8365


V203
V191
V212
V202
V187
V185
V186
V180
V242
V167
V189
V188
V243
V196
V182
V245
V244
V239
V240
V241
V169
V226
V220
V231
V222
V172
V224
V216
V219
V221
V227
V214
V233
V246
V232
V197
V181
V178
V228
V229
V201
V211
V192
V168
V184
V171
V225
V200
X14
X13
X16
X9
X8
X11
X1
X3
X2
X17
X21
X18
X19
X20
0
20000
40000
60000
80000
100000
120000

2.3. Feature engineering

En este momento recodificamos o agrupamos en intervalos las variables. Las variables de tipo object las recodificaremos para reducir sus categorías.

In [7]:
```
for f in col_obj:
    dist_values = df[f].value_counts().shape[0]
    print('Variable {} has {} distinct values'.format(f, dist_values))
```

```
Variable V209 has 3 distinct values
Variable V183 has 110 distinct values
Variable V210 has 8 distinct values
Variable V175 has 2 distinct values
Variable V176 has 3 distinct values
Variable V198 has 4 distinct values
Variable V190 has 5 distinct values
Variable V193 has 104 distinct values
Variable V204 has 8 distinct values
Variable V218 has 8 distinct values
Variable V230 has 5 distinct values
Variable V217 has 28 distinct values
Variable V223 has 1705 distinct values
Variable V215 has 138 distinct values
Variable V170 has 82 distinct values
Variable V177 has 2 distinct values
Variable V163 has 5 distinct values
Variable V164 has 5 distinct values
Variable V165 has 3 distinct values
Variable V166 has 4 distinct values
Variable X12 has 3 distinct values
Variable X7 has 3 distinct values
Variable X4 has 3 distinct values
```

V223 es la que tiene más categorías. Lo que hemos hecho ha sido agrupar las categorías que tengan menos de 500 observaciones en una categoría común llamada ‘OTRO’.

La variable V183 tiene 110 categorías. Una de ellas, la mayoritaria, es ‘ESPA’, por tanto, agrupamos todas las demás en una categoría que llamaremos ‘OTRA’. Así habremos reducido a solamente dos categorías.

In [17]:
```
for f in col_obj:
    dist_values = df[f].value_counts().shape[0]
    print('Variable {} has {} distinct values'.format(f, dist_values))
```

```
Variable V209 has 3 distinct values
Variable V183 has 2 distinct values
Variable V210 has 6 distinct values
Variable V175 has 2 distinct values
Variable V176 has 3 distinct values
Variable V198 has 4 distinct values
Variable V190 has 5 distinct values
Variable V193 has 42 distinct values
Variable V204 has 7 distinct values
Variable V218 has 5 distinct values
Variable V230 has 5 distinct values
Variable V217 has 16 distinct values
Variable V223 has 42 distinct values
Variable V215 has 33 distinct values
Variable V170 has 26 distinct values
Variable V177 has 2 distinct values
Variable V163 has 5 distinct values
Variable V164 has 5 distinct values
Variable V165 has 3 distinct values
Variable V166 has 4 distinct values
Variable X12 has 3 distinct values
Variable X7 has 3 distinct values
Variable X4 has 3 distinct values
```

Una vez hecho esto realizamos un label encoder para pasar las variables categóricas a datos numéricos.

```
In [26]: from sklearn import preprocessing
         le = preprocessing.LabelEncoder()
         for i in col_obj:
             le.fit(df_obj2[i])
             #list(le.classes_)
             df_obj2[i] = le.transform(df_obj2[i])

In [27]: col_lenc = []
         for i in col_obj:
             col_lenc.append(i+'_encoded')
         #col_lenc
         df_obj2.rename(columns=dict(zip(col_obj, col_lenc)), inplace=True)

In [28]: df_obj2.head()

Out[28]:
```

ref	V209_encoded	V183_encoded	V210_encoded	V175_encoded	V176_encoded	V198_encoded	V190_encoded	V193_encoded	V204_encoded	V218_e
201390679	2	1	1	1	1	0	3	10	0	
201390959	2	0	2	0	1	0	0	19	2	
201397626	0	0	5	0	1	0	0	19	1	
201397742	2	0	0	0	1	0	0	39	3	
201401345	0	0	1	1	2	1	0	32	4	

5 rows × 23 columns

Las variables V212, V180, V241, X11 las hemos agrupado en intervalos.

```
In [20]: bins1 = [0, 5, 10, 20, 30, 40, 55, 9999]
         #names = ['1', '2', '3', '4', '5', '6', '9999']
         df['V241'] = pd.cut(df['V241'], bins1, labels = names)
         df['V241'].value_counts()

Out[20]: 9999    71243
         1       19260
         2        4375
         4        2210
         3        1490
         5        1432
         6          74
         Name: V241, dtype: int64
```

2.4. Feature selection

En primer lugar, hemos tratado el desbalanceo de la variable respuesta. Hemos realizado las dos técnicas: Upsampling y Downsampling. La variable respuesta tiene estos valores.

0 102703

1 1211

Upsampling

La técnica de upsampling consiste en tomar una muestra de registros con reemplazamiento de la clase minoritaria. El tamaño de la muestra será del número de observaciones con la clase mayoritaria. Ahora tendremos más

observaciones que en la original pero el ratio de clases de la variable respuesta es 1:1:

0	102703
1	102703

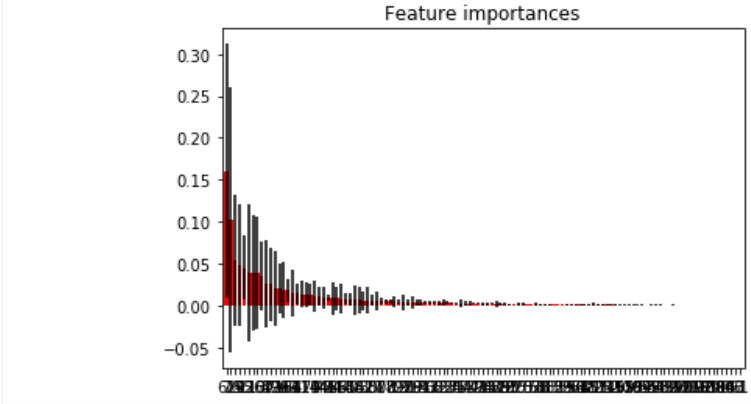
Downsampling

Esta técnica consiste en tomar una muestra sin reemplazamiento de los registros de la clase mayoritaria. El tamaño de la muestra será del número de observaciones de la clase minoritaria. Tenemos entonces menos registros que en el origina,l pero el ratio de las clases de la variable respuesta es de 1:1.

0	1211
1	1211

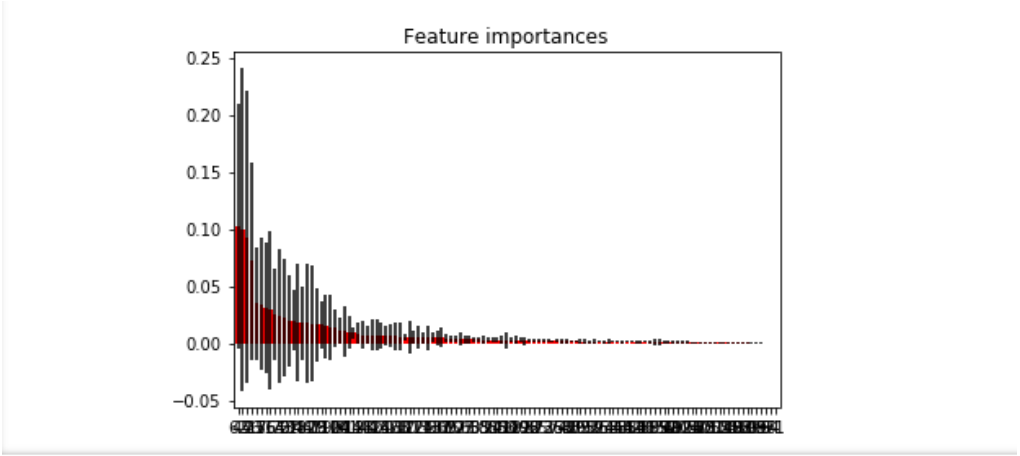
Pasamos ahora a la reducción del número de variables. Hemos utilizado la técnica de selección basada en árboles. Usamos el modelo ExtraTreesClassifier() y lo entrenamos sobre los dataframes que obtenemos de realizar upsampling y downsampling. Obtenemos del modelo entrenado las features_importances_

```
Feature ranking:
1. feature 62 (0.160579)
2. feature 76 (0.102672)
3. feature 15 (0.053861)
4. feature 9 (0.048048)
5. feature 45 (0.044979)
6. feature 2 (0.039514)
7. feature 113 (0.038913)
8. feature 10 (0.038547)
9. feature 64 (0.034886)
10. feature 7 (0.025439)
11. feature 19 (0.025132)
12. feature 71 (0.020450)
13. feature 20 (0.020282)
14. feature 16 (0.018685)
15. feature 69 (0.017537)
16. feature 66 (0.014841)
17. feature 1 (0.013797)
18. feature 41 (0.013045)
```



Y con downsampling:

```
Feature ranking:
1. feature 62 (0.102389)
2. feature 45 (0.099278)
3. feature 2 (0.092476)
4. feature 46 (0.071997)
5. feature 15 (0.034549)
6. feature 7 (0.033985)
7. feature 76 (0.030925)
8. feature 113 (0.028963)
9. feature 64 (0.025758)
10. feature 70 (0.023272)
11. feature 6 (0.022725)
12. feature 27 (0.019793)
13. feature 34 (0.019745)
14. feature 18 (0.018042)
15. feature 112 (0.017671)
16. feature 4 (0.017355)
17. feature 67 (0.016962)
18. feature 19 (0.016185)
```



Seleccionamos para ambos casos aquellas variables cuya importancia sea mayor que 0.01. Nos queda para upsampling una selección de 22 variables y para downsampling, una selección de 24 variables. De todas las que hemos seleccionado ninguna es categórica y no hemos aplicado el OneHotEncoder.

Tenemos ya los datos preparados para aplicarles los modelos.

3. Modelos

En este apartado se presentarán diferentes enfoques de modelos de machine learning.

3.1. ASAP: As Simple As Possible

Creamos un modelo lo más simple posible que permita extraer el máximo valor de los datos. Elegimos el modelo de regresión logística que se utiliza para clasificación, ya que nuestro problema consiste en clasificar con dos posibles estados Si Fraude = 1 / No Fraude = 0. Confiaremos en la implementación del paquete sklearn en Python para ponerlo en práctica. Entrenaremos dos modelos uno para upsampling y otro para downsampling. Creamos nuestro modelo y hacemos que se ajuste (fit) a nuestro conjunto de entradas X y salidas 'y'. Una vez compilado nuestro modelo, le hacemos clasificar todo nuestro conjunto de entradas X utilizando el método "predict(X)" y revisamos algunas de sus salidas y confirmamos cuan bueno fue nuestro modelo utilizando model.score() que nos devuelve la precisión media de las predicciones, en nuestro caso del 93.57% upsampled y del 91.62% para downsampled.

Una buena práctica en Machine Learning es la de subdividir nuestro conjunto de datos de entrada en un set de entrenamiento y otro para validar el modelo (que no se utiliza durante el entrenamiento y por lo tanto la máquina desconoce). Esto evitará problemas en los que nuestro algoritmo pueda fallar por sobreajustar el conocimiento.

Para ello, subdividimos nuestros datos de entrada en forma aleatoria (mezclados) utilizando 80% de registros para entrenamiento y 20% para validar. Volvemos a compilar nuestro modelo de Regresión Logística pero esta vez sólo con 80% de los datos de entrada y calculamos el nuevo scoring. Nos da 93.55% para upsampled y un 91.74% para downsampled.

Y ahora hacemos las predicciones -en realidad clasificación- utilizando nuestro "cross validation set", es decir del subconjunto que habíamos apartado. En este caso vemos que los aciertos fueron del 93.65% para upsampled y un 91.13% para downsampled. Finalmente vemos en pantalla la "matriz de confusión" donde muestra cuantos resultados equivocados tuvo de cada clase (los que no están en la diagonal).

Upsampled

Predicted	0	1	__all__
Actual			
0	19082	1445	20527
1	1162	19393	20555
__all__	20244	20838	41082

Downsampled

Predicted	0	1	__all__
Actual			
0	211	26	237
1	17	231	248
__all__	228	257	485

También podemos ver el reporte de clasificación con nuestro conjunto de validación. La valoración que de aqui nos conviene tener en cuenta es la de F1-score, que tiene en cuenta la precisión y recall. El promedio de F1 es de 94% para upsampled y del 91% para downsampled, lo cual no está nada mal.

3.2. Mejor modelo posible

Probamos diferentes algoritmos de mayor complejidad.

SVM

Una máquina de vectores de soporte (SVM) es un algoritmo de aprendizaje supervisado que se puede emplear para clasificación binaria o regresión.

El procedimiento es el mismo. Hemos utilizado uno modelo por defecto con kernel = linear, por la capacidad de nuestra máquina. Cargamos nuestro modelo y lo entrenamos en los conjuntos train de nuestros datos para upsampled y downsampled. Hacemos las predicciones sobre nuestros conjuntos de validación. Obtenemos una accuracy para upsampled de 92.72% y para downsampled de 89.90%. Finalmente vemos la matriz de confusión:

Downsampled

Predicted	0	1	__all__
Actual			
0	208	29	237
1	20	228	248
__all__	228	257	485

Upsampled

Predicted	0	1	__all__
Actual			
0	18390	2137	20527
1	810	19745	20555
__all__	19200	21882	41082

No hemos podido optimizar los parámetros para buscar el mejor modelo.

Random Forest

Los bosques aleatorios consisten simplemente en la generación de un número de árboles de decisión cuyos nodos adoptan condiciones diferentes para decidir

sobre el mismo conjunto de datos de entrada. El resultado del bosque aleatorio será el promedio del resultado de los n árboles que conformen el bosque.

Al entrenar el modelo con n_estimators = 200 obtenemos para upsampled un accuracy de 99.94% y para downsampled de 96.08%. Finalmente obtenemos las matrices de confusión:

Upsampled

Predicted	0	1	__all__
Actual			
0	20503	24	20527
1	0	20555	20555
__all__	20503	20579	41082

Downsampled

Predicted	0	1	__all__
Actual			
0	228	9	237
1	10	238	248
__all__	238	247	485

3.3. Comparación de modelos

Vemos que con la técnica de upsampling se obtienen mejores resultados que con downsampling, además de que con upsampling tenemos más datos para realizar el entrenamiento del modelo. Por tanto, para realizar el balanceo de los datos nos quedaríamos finalmente la técnica de upsampling.

El modelo que hemos implementado con random forest parece estar sobreajustado, su accuracy es próximo a 1. Hemos intentado optimizar los parámetros para obtener el mejor modelo, pero ha sido imposible con las especificaciones de mi máquina.

```
param_grid = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [4,5,6,7,8],
    'criterion': ['gini', 'entropy']
}
```

Con SVM nos ha ocurrido lo mismo, que hemos intentado optimizar sus parámetros, pero ha sido inviable.

```
param_grid = [
  {'C': [1, 10], 'kernel': ['linear']},
  {'C': [1, 10], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
 ]
```

Sin embargo, cargando un modelo svm por defecto con kernel = linear, hemos obtenido una buena medida de en el accuracy y con un f1 – score de 0.93%.

La regresión logística que hemos implementado con cross-validation, además de su simpleza y rapidez de proceso nos da un buen accuracy mayor que el que obtenemos con svm.

4. Resultados

En vista de los resultados obtenidos y de la imposibilidad de implementar otros modelos (nos hubiera gustado implementar un XGBoost, ya que da mejor solución que otros modelos) y teniendo en cuenta nuestra máquina actual, nos decidiríamos por implementar el modelo de regresión logística con el que hemos obtenido buenos resultados. Para aplicar nuestro modelo a los nuevos datos que nos proporcione la empresa tendremos que realizar las mismas modificaciones que hemos aplicado a nuestro conjunto de datos inicial.

Es importante tener un buen modelo que permita a la empresa detectar posibles clientes fraudulentos por muchos motivos el más evidente es el económico y legal, pero también es indicativo de ser una empresa seria y con valores, da confianza a sus clientes y posibilita la mejora de sus servicios. Por todo esto no escatimaría a la hora de desarrollar un modelo que detecte a los posibles clientes fraudulentos.

Duración del proyecto:

- ETL: 75% (3 semanas)
- Modelado: 25% (1 semana)

5. Apéndice: código

- data_exploration_tfm_IsaacLopez (ETL)
- data_exploration_2_tfm_IsaacLopez (balanceo, selección y carga)