Netflix Appetency

Más allá del mes de prueba

Proyecto Final

Melissa Sofía Miranda Peñafiel

Índice general

1

Introducción

Resumen Ejecutivo

"There is a revolution happening, and within two years I think that Wi-Fi and Netflix will be built into all the televisions…" - Reed Hastings (2009), fundador de Netflix.

Cada vez existen más plataformas de streaming en las que podemos ver todo tipo de contenido multimedia con el beneficio de ver un capítulo detrás de otro y sin ver anuncios. Las posibilidades del streaming son inmensas y están teniendo esta fama porque el usuario optimiza su tiempo como él quiere, sin interrupciones, disfrutando de todo el contenido y no teniendo que conformarse con los programas de TV convencionales plagados de publicidad. Es por esto que muchas plataformas ofrecen un periodo de prueba esperando atraer nuevos expectadores y así, completar la suscripción a la plataforma.

Netflix es una de las plataformas de streaming más grandes hoy en día. El mes de prueba que otorga a los usuarios para acceder a su contenido original cautiva a los espectadores y puede llegar a ser suficiente para que paguen por la suscripción, con la expectativa de buen entretenimiento a un precio accesible.

Sin embargo, el reto para Netflix es predecir si los usuarios se suscribirán o no terminando el mes de prueba.

Por este motivo, el proyecto está enfocado en tratar de predecir si un usuario se suscribirá al final del mes de prueba de acuerdo con sus interacciones en la plataforma y, en caso de que no se suscriba, encontrar las posibles razones y tomar acciones para completar la mayor cantidad de suscripciones al final del mes.

Para lograrlo, se entrenaran modelos de aprendizaje supervizado, para tratar de predecir si se suscribirá o no, y no supervizado, para obtener información extra de los datos.

2

Sobre los datos

La base está conformada por 100,000 datos de 509 variables. Se encuentra segmentada en train y test con 70,000 y 30,000 datos respectivamente.

Los datos fueron extraídos de Kaggle [1] y fueron proporcionados por Netflix, provocando que, por confidencialidad, no se publicara con exactitud la descripción de las variables *"feature"* (que conforman 507 columnas de las 509 de la base). Las dos columnas restantes son *'id'* y *'target'*.

- **id:** muestra el número de identificación del usuario;
- **target:** muestra un 1 si el usuario se suscribió a Netflix y 0 si no.

Para los fines de este proyecto se trabajó únicamente con los datos de la base train para poder evaluar la capacidad de los modelos, ya que la base test no contaba con valores para la variable respuesta (*'target'*) y no era posible evaluar el poder predictivo de los modelos.

3

Limpieza de la base

La base *train* contaba con variables categóricas y numéricas. Debido a la cantidad y naturaleza censurada de los datos, no fue posible realizar un análisis exploratorio detallado de la base, sin embargo pudimos observar que, al menos, el 70.18 % de los usuarios no contratan la suscripción al finalizar el mes de prueba.

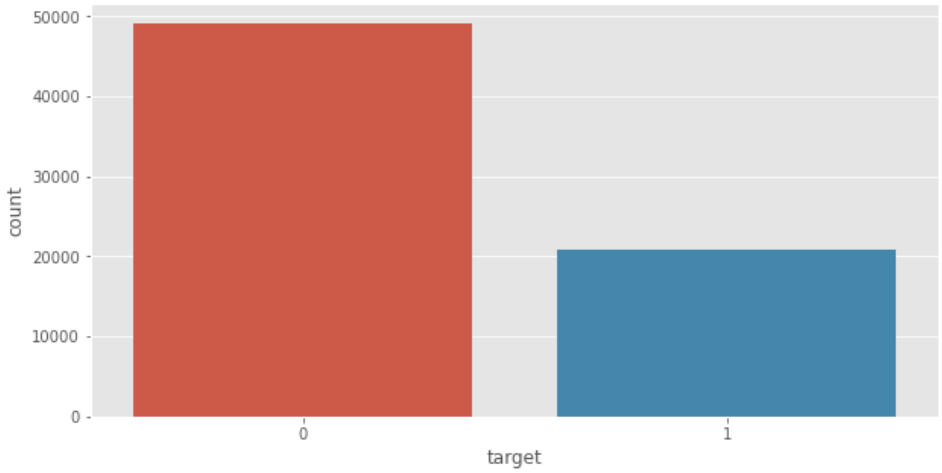


Figura 3.1: Cantidad de usuarios no suscritos (categoría 0) y suscritos (categoría 1)

Como no había forma de saber el significado de las variables *feature* era imposible determinar su importancia para el análisis a primera vista, por lo que se optó por eliminar aquellas columnas que tuvieran más del 10 % de datos faltantes.

Para el tratamiento de los valores faltantes se observó lo siguiente:

- Número de variables que contenían al menos un valor faltante: **73**;
- Número de variables que no contenían ningún valor faltante: **435**.

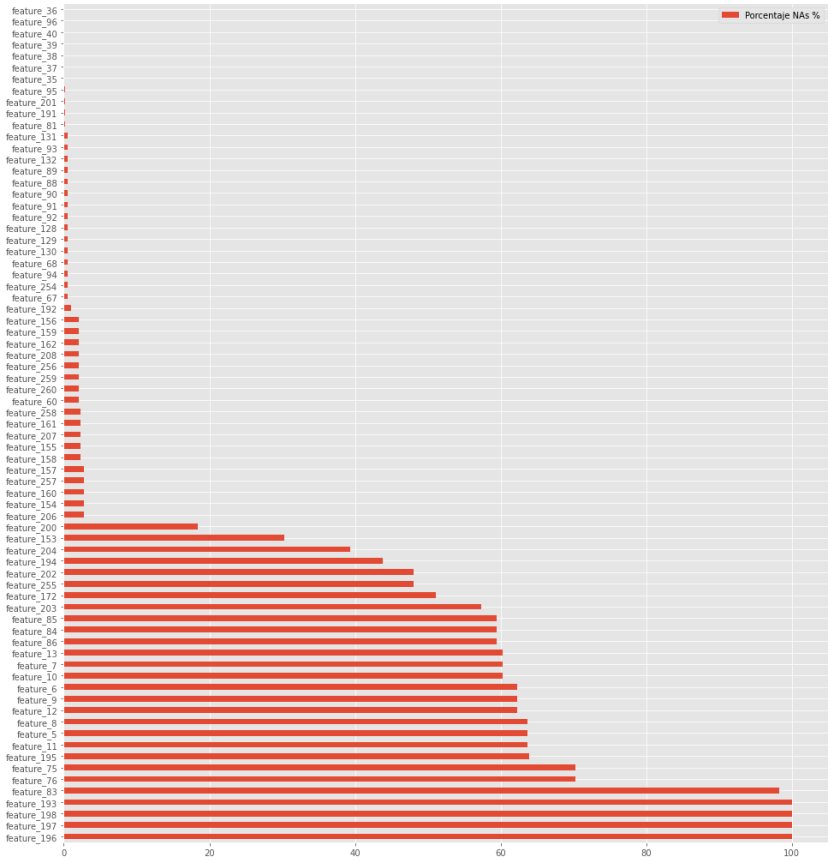


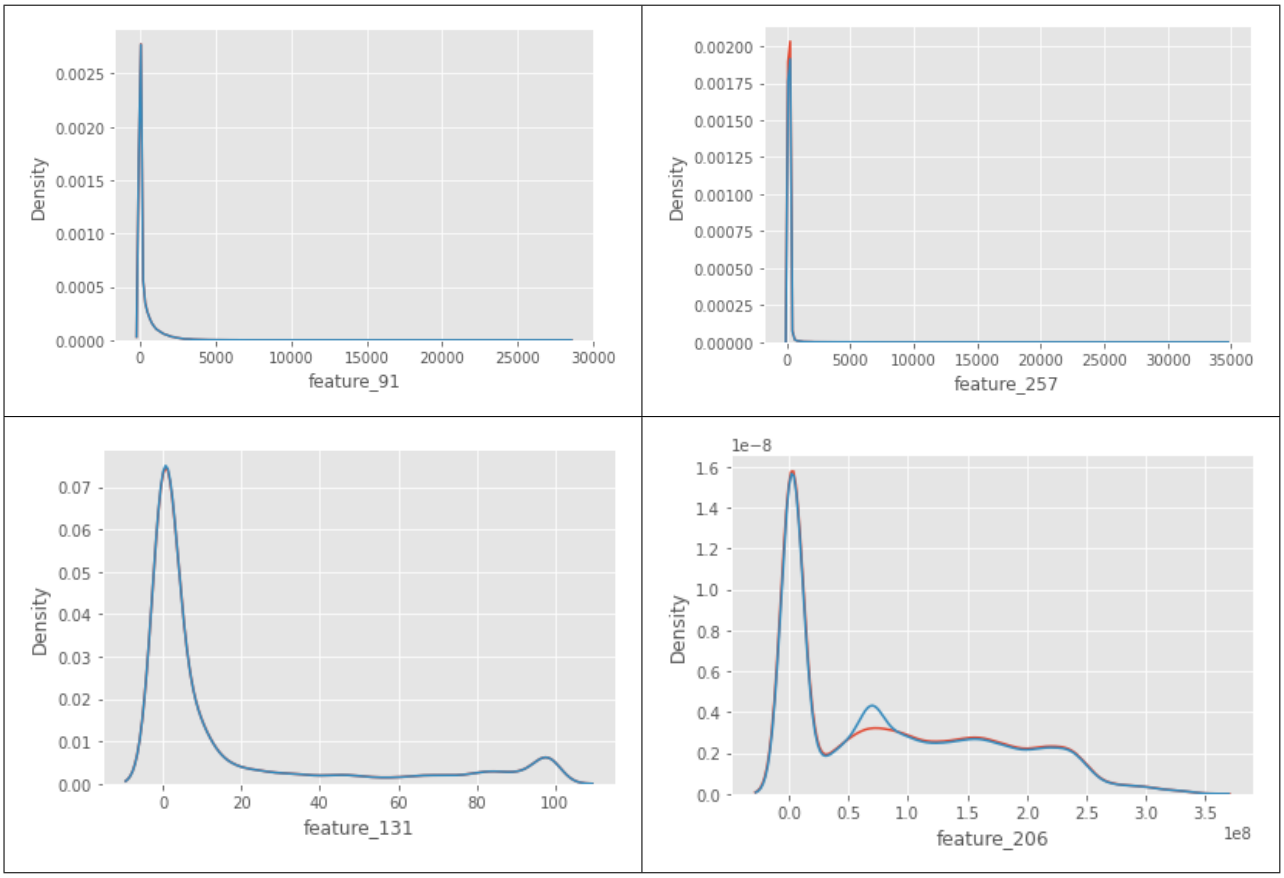
Figura 3.2: Porcentaje de valores faltantes en cada variable que contaba con ellos.

A su vez, las variables **categóricas** que contaban con sólo una categoría o que contaban con más de 100 categorías distintas fueron eliminadas ya que se consideró que no aportarían información extra a los modelos.

En el caso de las variables **numéricas** se eliminaron las que eran constantes (i.e. sólo contaban con un valor en todas sus entradas).

También se eliminaron 5 variables que correspondían a fechas.

Y para no tener datos faltantes se imputaron los datos por el método de la mediana. Adicional, se hizo una prueba de Kolmogorov-Smirnov para ver que la imputación conservaba la distribución original de los datos.



Cuadro 3.1: Distribución de algunas variables después de la imputación

Después de este tratamiento, la base contaba con 403 variables, 341 variables continuas y 62 variables categóricas.

4

Aprendizaje Supervisado

4.1. Selección de variables

Se observó la correlación entre las variables *feature* y la variable *target* para determinar qué variables podrían tener un mayor peso dentro de la modelación.

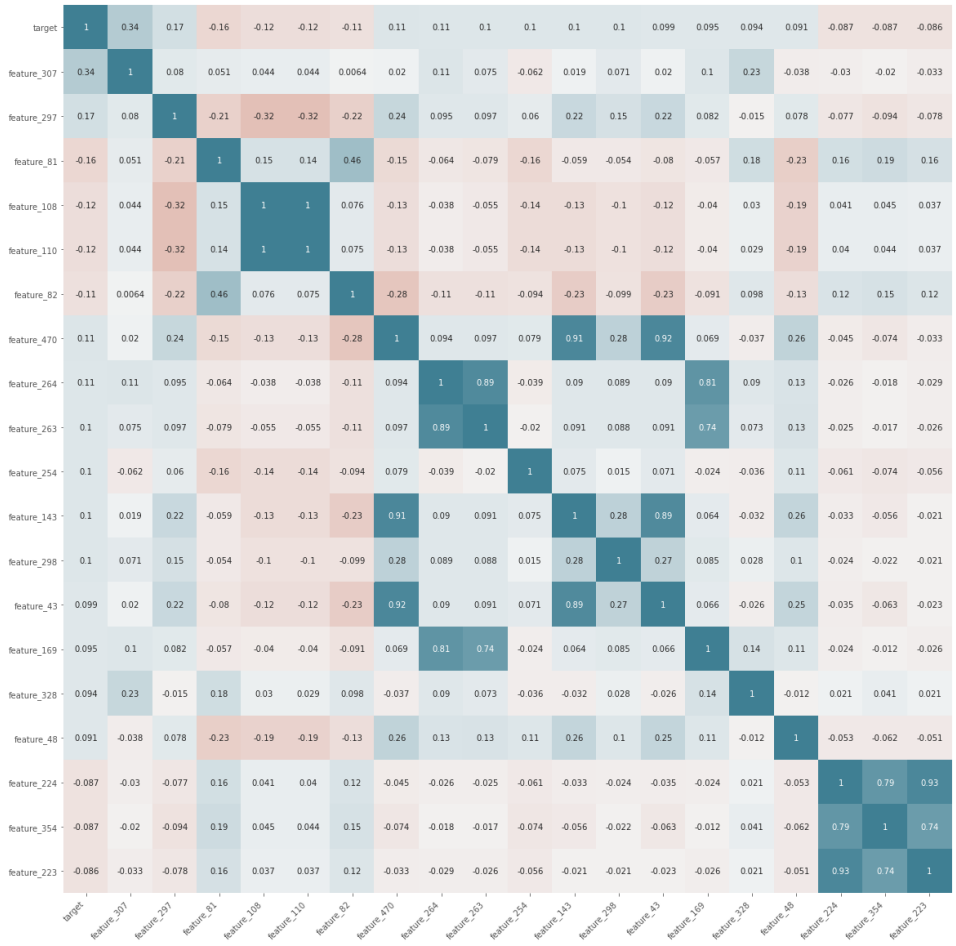


Figura 4.1: Mapa de calor de correlaciones de las primeras 20 variables con mayor correlación respecto a la variable respuesta *target*.

Se utilizó la función **SelectKBest** con las variables numéricas para contrastar las variables observadas anteriormente con las que obtuviera esta función. De esta manera se escogieron las primeras 15 ”mejores”variables para proceder con los modelos.

La base de datos fue particionada en una proporción de 80/20 para tener una parte de **train** y una de **validate** respectivamente; posteriormente la parte **train** volvió a particionarse en un 80/20 para tener los conjuntos tradiconales de *train* y *test.*

4.2. Regresión logística

Una regresión logística es un modelo lineal que por medio de una función logit nos permite hacer predicciones de clasificación.

Se ajustó el modelo obteniendo lo siguiente:

Conjunto	ROC	ACC
train	0.750	0.751
test	0.752	0.746
validate	0.758	0.757

Cuadro 4.1: Métricas ROC y Accuracy para los conjuntos *train* , *test* y *validate*.

	No suscrito	Suscrito
No suscrito	28,607	2,812
Suscrito	8,331	5,049

Cuadro 4.2: Matriz de confusión para el conjunto *train*.

	No suscrito	Suscrito
No suscrito	7,086	683
Suscrito	2,158	1,273

Cuadro 4.3: Matriz de confusión para el conjunto *test*.

	No suscrito	Suscrito
No suscrito	9,011	928
Suscrito	2,476	1,585

Cuadro 4.4: Matriz de confusión para el conjunto *validate*.

Dadas las métricas anteriores se podría concluir que el modelo parece ajustarse bien y clasifica de mejor manera a los usuarios que **no** se suscriben al servicio, teniendo una cifra de éxito mayor al 80 % (F1 Score). Por otro lado, se tiene una cifra menor al 50 % para clasificar a usuarios que **sí** suscriben.

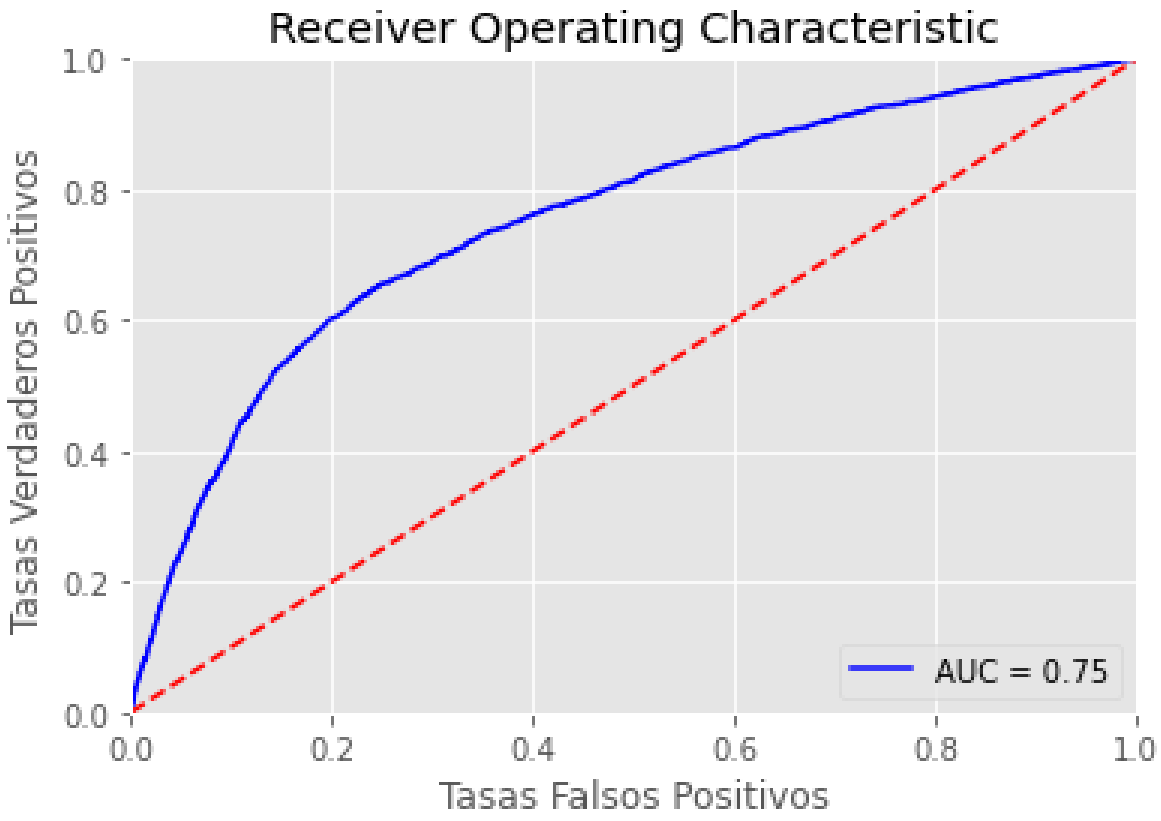


Figura 4.2: Curva ROC

Un AUC de 0.75 nos indica que existe un 75 % de probabilidad de que el diagnóstico realizado a un usuario sea correcto.

4.3. Árbol de decisión

Un árbol de decisión es un mapa de los posibles resultados de una serie de decisiones relacionadas.

Un árbol de decisión es un modelo que predice el valor de una variable objetivo por medio de reglas de decisión inferidas por las características de los datos.

Se optó por empezar con la malla que el modelo ajusta por default, sin embargo se obtuvieron resultados sobreajustados. Se procedió a hacer una búsqueda por los mejores parámetros para el modelo ajustando 3,000 modelos distintos.

```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'entropy',
 'max_depth': 5,
 'max_features': 'log2',
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 7,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

Figura 4.3: Parámetros del mejor modelo de árbol de decisión.

El modelo utilizado toma como criterio la entropía (criterio que toma por automático), una profundidad máxima de 5 divisiones y que al menos tenga una muestra de 7 registros por hoja.

Conjunto	ROC	ACC
train	0.747	0.754
test	0.747	0.750
validate	0.752	0.755

Cuadro 4.5: Métricas ROC y ACC para los conjuntos *train* y *test*.

	No suscrito	Suscrito
No suscrito	28,081	3,338
Suscrito	7,685	5,695

Cuadro 4.6: Matriz de confusión para el conjunto *train*.

	No suscrito	Suscrito
No suscrito	6,953	816
Suscrito	1,986	1,445

Cuadro 4.7: Matriz de confusión para el conjunto *test*.

	No suscrito	Suscrito
No suscrito	8,826	1,113
Suscrito	2,311	1,750

Cuadro 4.8: Matriz de confusión para el conjunto *validate*.

Los resultados parecen ser buenos, mas no superan a los del modelo de regresión logística.

4.4. Balanceo de clases

Retomando la observación que se hizo anteriormente, las clases de la variable respuesta no están balanceadas, por esto se buscó balancearlas para tratar de obtener mejores resultados.

De acuerdo a las particiones de **train**,**test** y **validate** se observó lo siguiente:

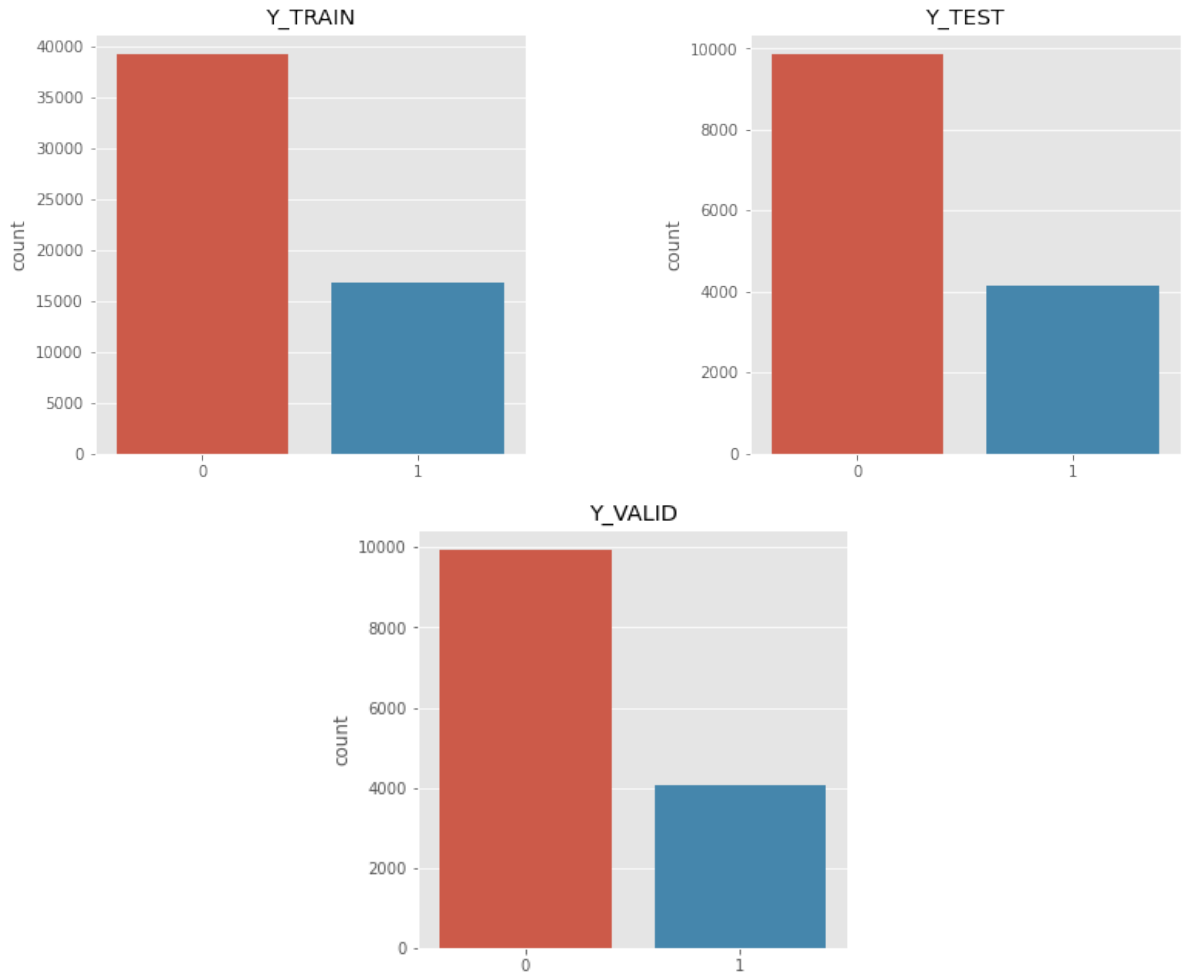


Figura 4.4: Cantidad de usuarios no suscritos (0) y suscritos (1) para el conjunto *train*, *test* y *validate*.

Oversampling

Esta técnica consiste en hacer sobremuestreo (agregar más copias) de la clase más pequeña; en este caso, la clase 1 que corresponde a usuarios suscritos. Se realizó únicamente para el conjunto **train**.

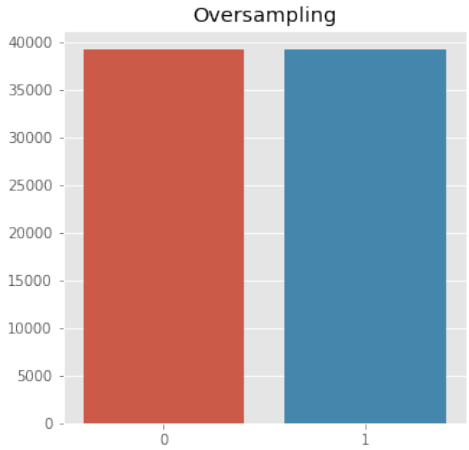


Figura 4.5: Cantidad de usuarios no suscritos y suscritos para el conjunto *train* después de usar la técnica de **oversampling**.

Se obtuvieron **31,419** observaciones para ambas clases.
Usando esta muestra para los modelos creados se obtuvo lo siguiente:

Modelo	ROC	F1-Score (clase 1)
Logístico	0.7587	0.5811
Árbol	0.7366	0.5708
Referencia logístico	0.7584	0.482
Referencia árbol	0.721	0.486

Cuadro 4.9: Métricas ROC y F-1 score para ambos modelos usando los datos muestreados con **oversampling**.

	No suscrito	**Suscrito**
No suscrito	7,513	2,426
Suscrito	1,404	2,657

Cuadro 4.10: Matriz de confusión para el conjunto *validate* del modelo logístico.

	No suscrito	**Suscrito**
No suscrito	7,763	2,176
Suscrito	1,570	2,491

Cuadro 4.11: Matriz de confusión para el conjunto *validate* del árbol de decisión.

Como podemos ver, en efecto mejoró el poder predictivo de los modelos con este muestreo.

Undersampling

Esta técnica de submuestreo consiste en eliminar algunas observaciones de la clase con mayor conteo.

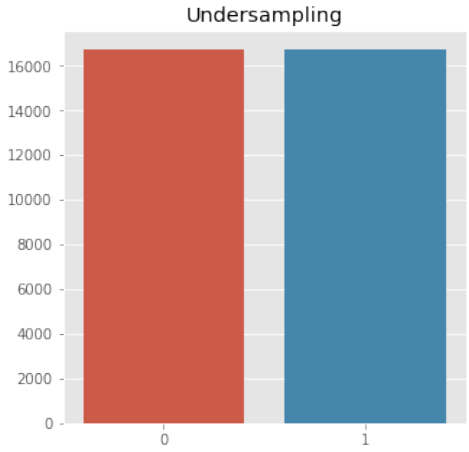


Figura 4.6: Cantidad de usuarios no suscritos y suscritos para el conjunto *train* después de usar la técnica de **undersampling**.

Se obtuvieron **13,380** observaciones para ambas clases.
Usando esta muestra para los modelos creados se obtuvo lo siguiente:

Modelo	ROC	F1-Score (clase 1)
Logístico	0.7586	0.5804
Árbol	0.6739	0.4902
Referencia logístico	0.7584	0.482
Referencia árbol	0.721	0.486

Cuadro 4.12: Métricas ROC y F-1 score para ambos modelos usando los datos muestreados con **undersampling**.

	No suscrito	**Suscrito**
No suscrito	7,513	2,426
Suscrito	1,409	2,652

Cuadro 4.13: Matriz de confusión para el conjunto *validate* del modelo logístico.

	No suscrito	**Suscrito**
No suscrito	6,812	3,127
Suscrito	1,727	2,334

Cuadro 4.14: Matriz de confusión para el conjunto *validate* del árbol de decisión.

Podemos observar que esta técnica no fue tan efectiva como el *oversampling* para mejorar la capacidad predictiva de los modelos.

SMOTE

Por sus siglas en inglés *"Synthetic Minority Over-sampling Technique"*, esta técnica utiliza el algoritmo de vecinos más cercanos para crear nuevos ejemplos de entrenamiento con la clase que tiene menos elementos.

Al igual que en la técnica de oversampling, se obtuvieron **31,419** observaciones por clase.

Usando esta muestra para los modelos creados se obtuvo lo siguiente:

Modelo	ROC	F1-Score (clase 1)
Logístico	0.7472	0.5756
Árbol	0.6695	0.4796
Referencia logístico	0.7584	0.482
Referencia árbol	0.721	0.486

Cuadro 4.15: Métricas ROC y F-1 score para ambos modelos usando los datos muestreados con **SMOTE**.

	No suscrito	**Suscrito**
No suscrito	7,403	2,536
Suscrito	1,395	2,666

Cuadro 4.16: Matriz de confusión para el conjunto *validate* del modelo logístico.

	No suscrito	**Suscrito**
No suscrito	7,863	2,076
Suscrito	2,125	1,936

Cuadro 4.17: Matriz de confusión para el conjunto *validate* del árbol de decisión.

Podemos observar que esta fue la técnica menos efectiva en el balanceo de clases.

4.5. Conclusión

La **Modelación Supervisada** utiliza los valores de uno o varios campos de entrada para predecir el valor de uno o varios resultados de salida. Las técnicas de modelado incluyen aprendizaje automático, inducción de reglas, identificación de subgrupos y métodos estadísticos; por lo que no es sencillo encontrar un único *buen modelo*, pues este dependerá de los datos que quieran predecirse y de las métricas utilizadas en él.

En nuestro caso, al principio, la respuesta parecía un poco obvia debido a que la variable respuesta estaba conformada de ceros y unos y lo primero que se viene a la mente es una regresión logística y, aunque el mejor modelo fue la regresión logística con clases balanceadas, pudimos comprobar que no era el único modelo capaz de predecir si la persona se suscribiría o no al final del mes de prueba y tampoco lo obtuvimos con uno o dos intentos.

Con respecto a la modelación, podemos concluir que, si bien no hay manera de asegurar un único mejor modelo, es posible entrenar mejores modelos ajustando los parámetros correctos como ocurrió con la RL y el *oversampling* sencillo. Y con respecto al rendimiento de los modelos podemos decir que, en general, tienen buena capacidad predictiva.

Conclusión sobre los datos

Como se vio desde la figura 3.1 la base tiene clases desbalanceadas en su variable objetivo, por lo que no es raro que los modelos tengan mayor eficiencia a la hora de predecir la clase de las personas no suscritas, sin embargo, es posible realizar otros análisis con los modelos obtenidos.

Por ejemplo: en el caso del árbol, si la variable *'feature_45'* fuera que el cliente ve mucho contenido para niños, Netflix podría enfocarse en tener más contenido de ese tipo en su catálogo y así atraer más clientes captando mayor número de suscripciones.

	importance	feature
0	0.583337	feature_45
1	0.277502	feature_15
2	0.056728	feature_43
3	0.026202	feature_38
4	0.013520	feature_40
5	0.008272	feature_47
6	0.008057	feature_46
7	0.007707	feature_39
8	0.005600	feature_25
9	0.004821	feature_44
10	0.003185	feature_26
11	0.001979	feature_37
12	0.001630	feature_35
13	0.001192	feature_36
14	0.000268	feature_14

Figura 4.7: Variables que influyen más en la pureza del árbol de decisión.

5

Aprendizaje No Supervisado

5.1. Selección de variables

Se observó la correlación entre las variables *feature* y la variable *target* para determinar qué variables podrían tener un mayor peso dentro de la modelación.

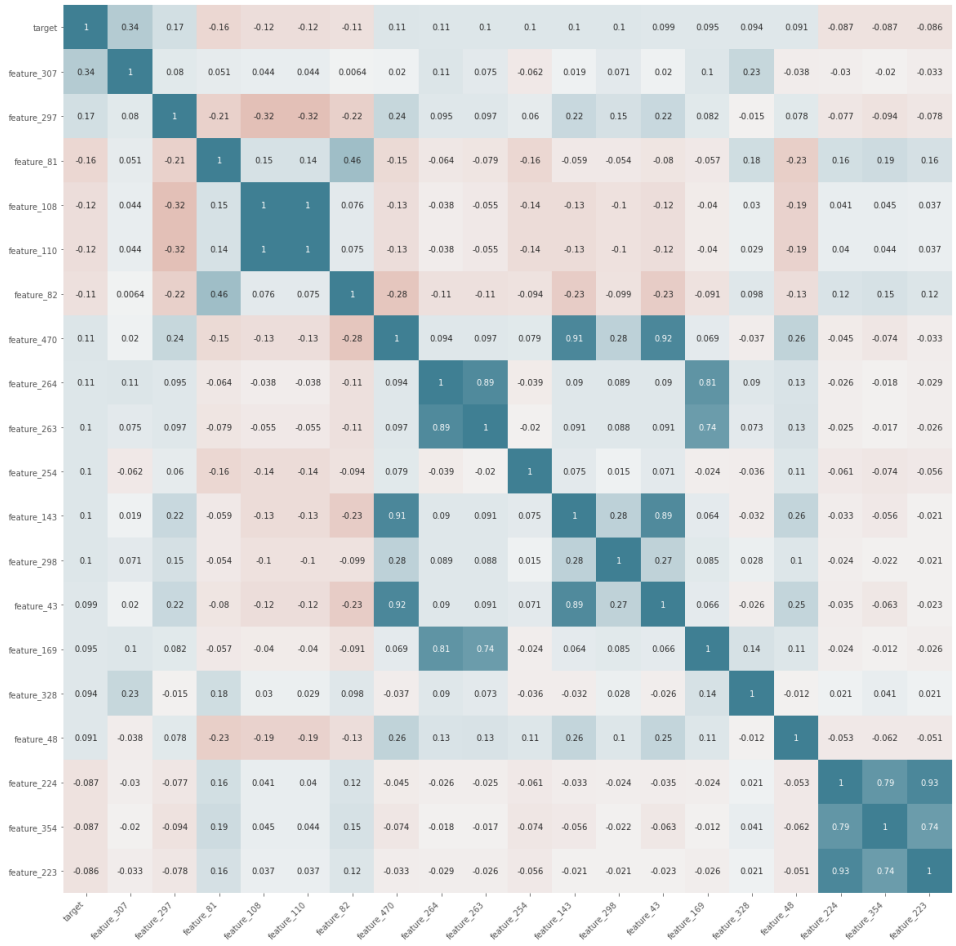


Figura 5.1: Mapa de calor de correlaciones de las primeras 20 variables con mayor correlación respecto a la variable respuesta *target*.

Se utilizó la función **SelectKBest** con las variables numéricas para contrastar las variables observadas anteriormente con las que obtuviera esta función. De esta manera se escogieron las primeras 15 ”mejores”variables; adicional, se decidió agregar la variable más importante del árbol de decisión de los modelos supervizados y las siguientes cinco varibales numéricas más correlacionadas con la variable *’target’* para proceder con los modelos.

5.1.1. Análisis de Negocio

Para esta parte del proyecto se decidió enfocar el análisis al grupo de personas que no completaron la suscripción al final del mes *(target = 0)*, esto con el objetivo de tratar de identificar algunas de las características que influyen en su decisión, sin embargo, fue necesario **especular** acerca del significado de las variables *’feature’* para poder identificar algunas de sus características.

Después de realizar un análisis de las variables continuas, se decidió darles el siguiente significado:

Variable	Significado
feature 81	Número de episodios en la serie vista
feature 82	Tiempo en horas que se vió Netflix al mes
feature 169	Tiempo que se jugó un juego de Netflix al mes
feature 26	Tiempo que se vio algo en la categoría de ‘Drama’
feature 66	Tiempo que se vio contenido exclusivo de Netflix
feature 128	% de uso de las funciones de Netflix (streaming, games, fast laughs, downloads)
feature 132	Tiempo que se vio algo del top 10 de la región
feature 156	Tiempo que se vio la categoria de ‘Ciencia-Ficción’
feature 207	Tiempo que se vio contenido para niños

Cuadro 5.1: Significado tentativo de las variables *’feature’*

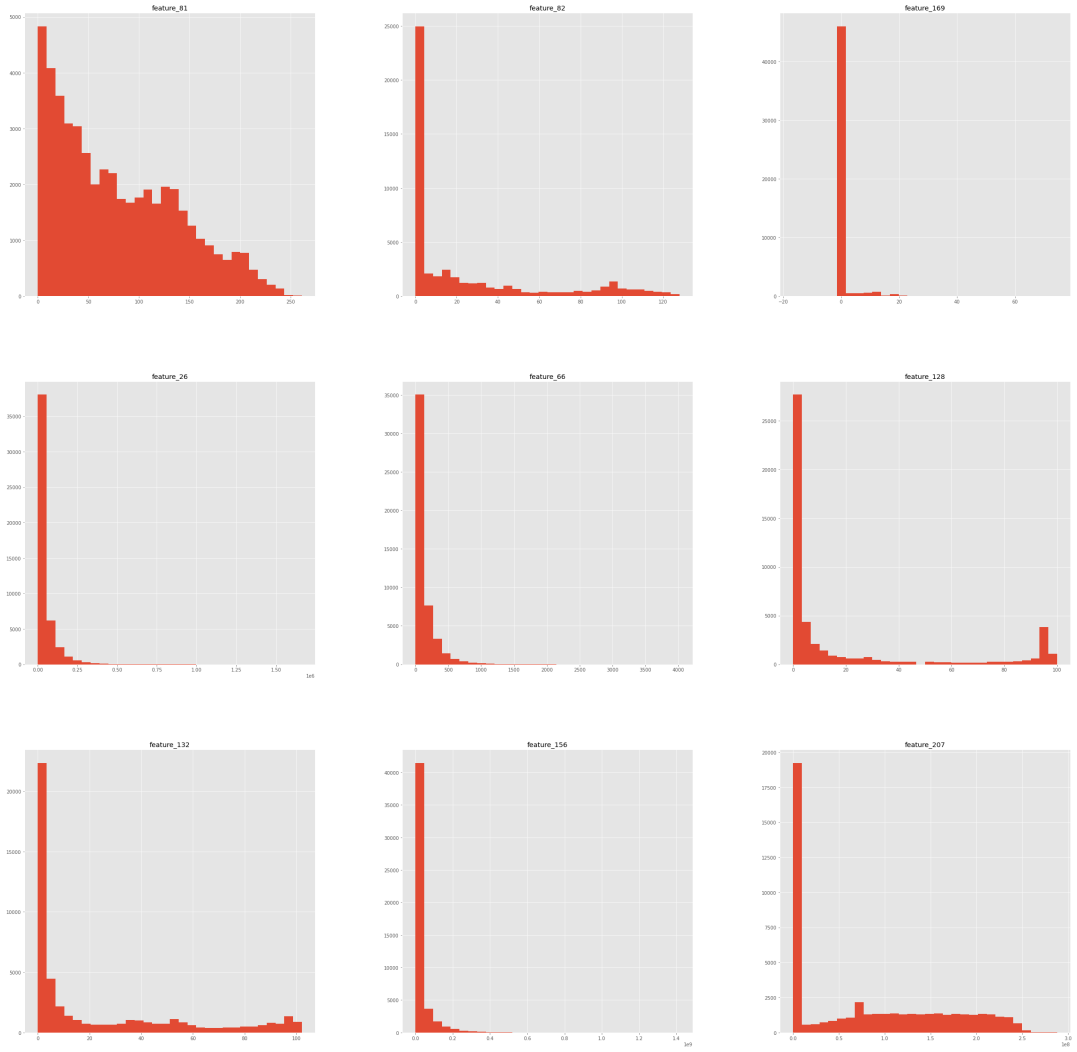


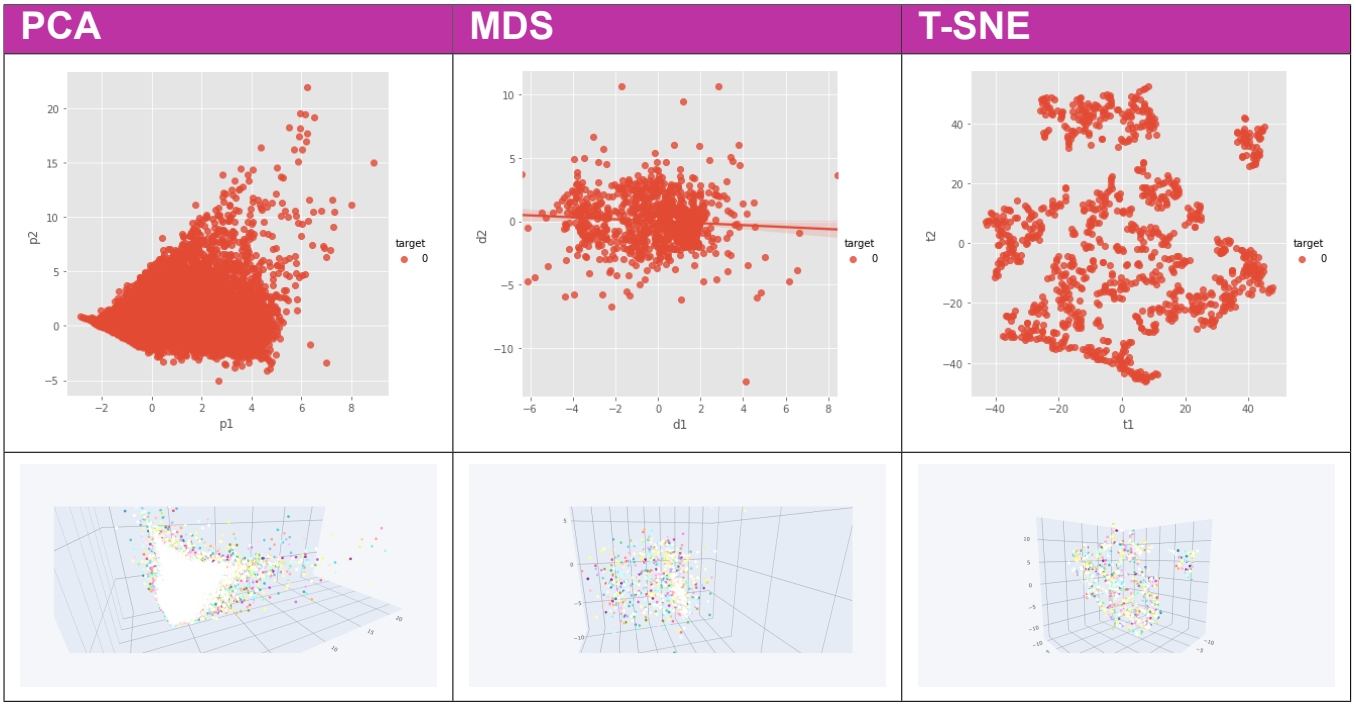
Figura 5.2: Variables continuas para la modelación

5.2. Modelado

Primeras Visualizaciones

Como buscabamos poder visualizar grupos en los datos, se aplicaron tres métodos de reducción de dimensiones:

- **PCA:** Principal Component Analysis
- **MDS:** Multidimensional Scaling
- **T-SNE:** T-Stochastic Neighbor Embedding



Cuadro 5.2: Visualizaciones con 2 componentes en 2D y 3D

5.2.1. Cluster Jerarquico

Este tipo de cluster se divide en *aglomerativo* y *divisivo*, procede sucesivamente fusionando grupos más pequeños en grupos más grandes (aglomerativo) o dividiendo grupos más grandes en grupos más pequeños (divisivo). El resultado del algoritmo es un árbol de conglomerados que muestra cómo se relacionan los clusters y se llaman dendogramas.

Primero se realizó el dendograma por el método de *WARD*, que es un método de linkage que fija un centroide por grupo, entonces al agregar una nueva observación queremos que la distancia entre ese punto y el centroide sea mínima y se agrupa al cluster donde quede la menor distancia a los centroides.

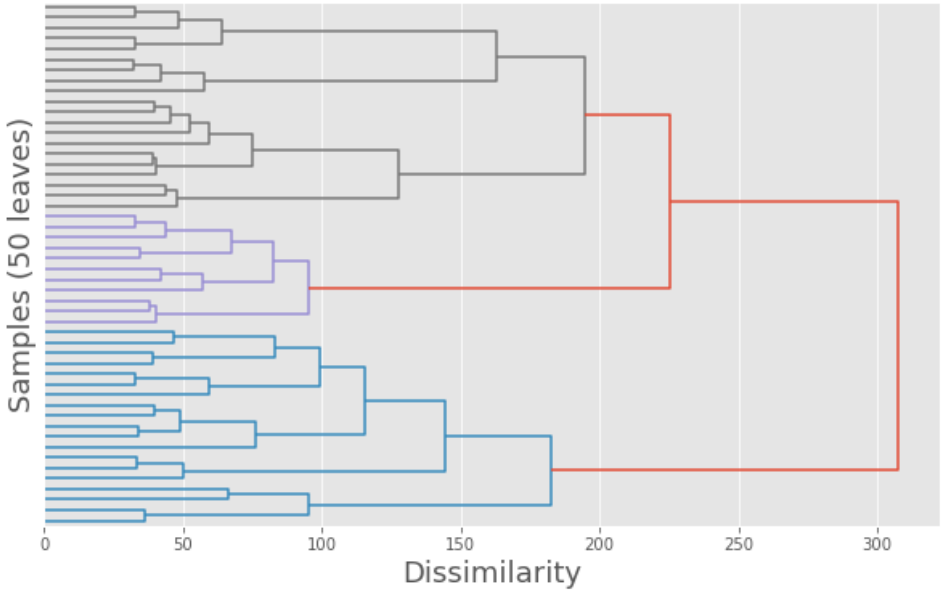
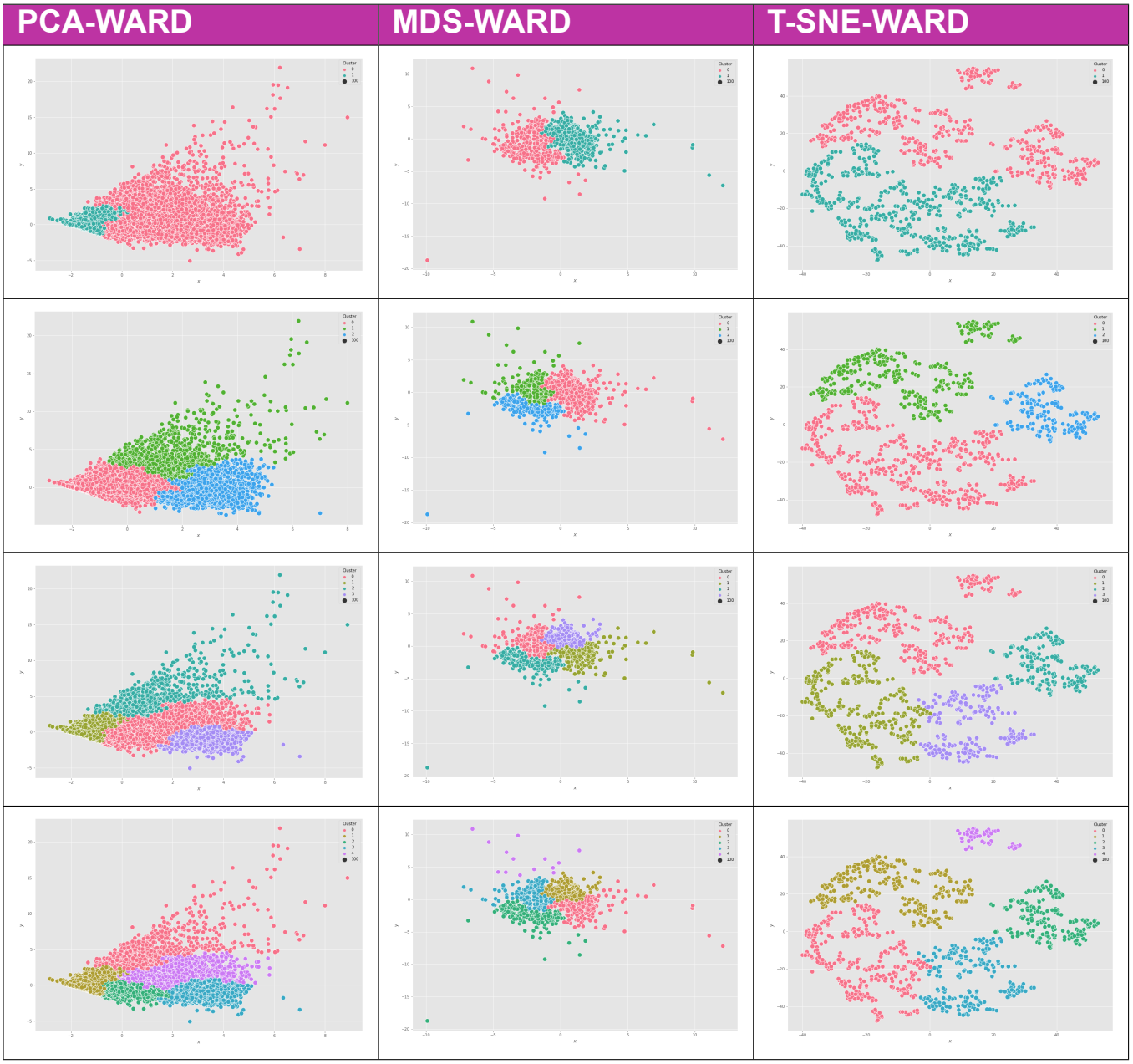


Figura 5.3: Dendograma por método de WARD

A un nivel de disimilitud de 200 pordemos observar que habría tres grupos, sin embargo, esto es solo una aproximación por lo que se optó por probar la agrupación por el método de WARD con 2,3,4 y 5 grupos para los tres métodos de reducción de dimensiones.



Cuadro 5.3: Visualizaciones de cluster jerarquico por método de WARD

5.2.2. Cluster de Optimización

Este tipo de clusterización busca optimizar la varianza por medio del algoritmo de *K-means*, parte de una configuración inicial y va moviendo las combinaciones, por lo que es necesario partir de un número inicial de cluster. Para esto, podemos obtener el número óptimo de cluster por el ’método del codo’ (WCSS), la silueta o DB-Score, cabe mencionar que no hay un ’mejor método’, solo son guias.

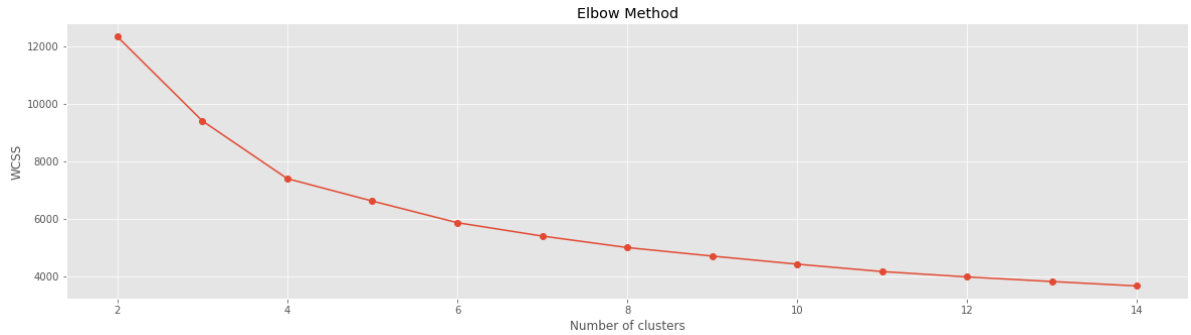


Figura 5.4: Método del codo

En el método del codo, el número óptimo de cluster se identifica donde la gráfica deja de decrecer, en este caso podríamos decir que sugiere 5 o 6 cluster.

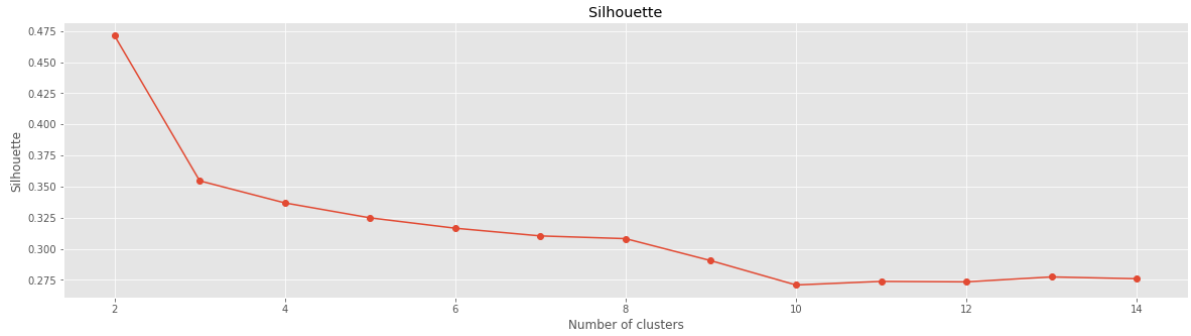


Figura 5.5: Método de la Silueta

El método de la silueta determina el número óptimo de cluster donde la gráfica se acerque más al 1, por lo que podemos observar sugiere 2 o 3 cluster.

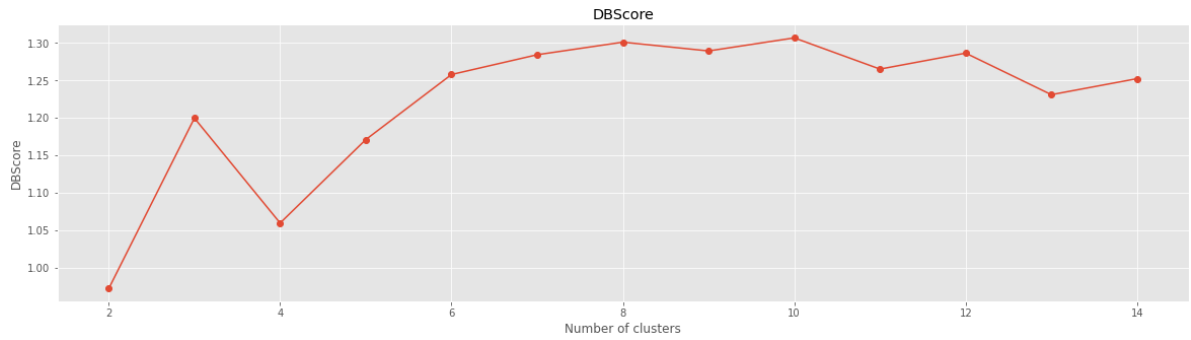
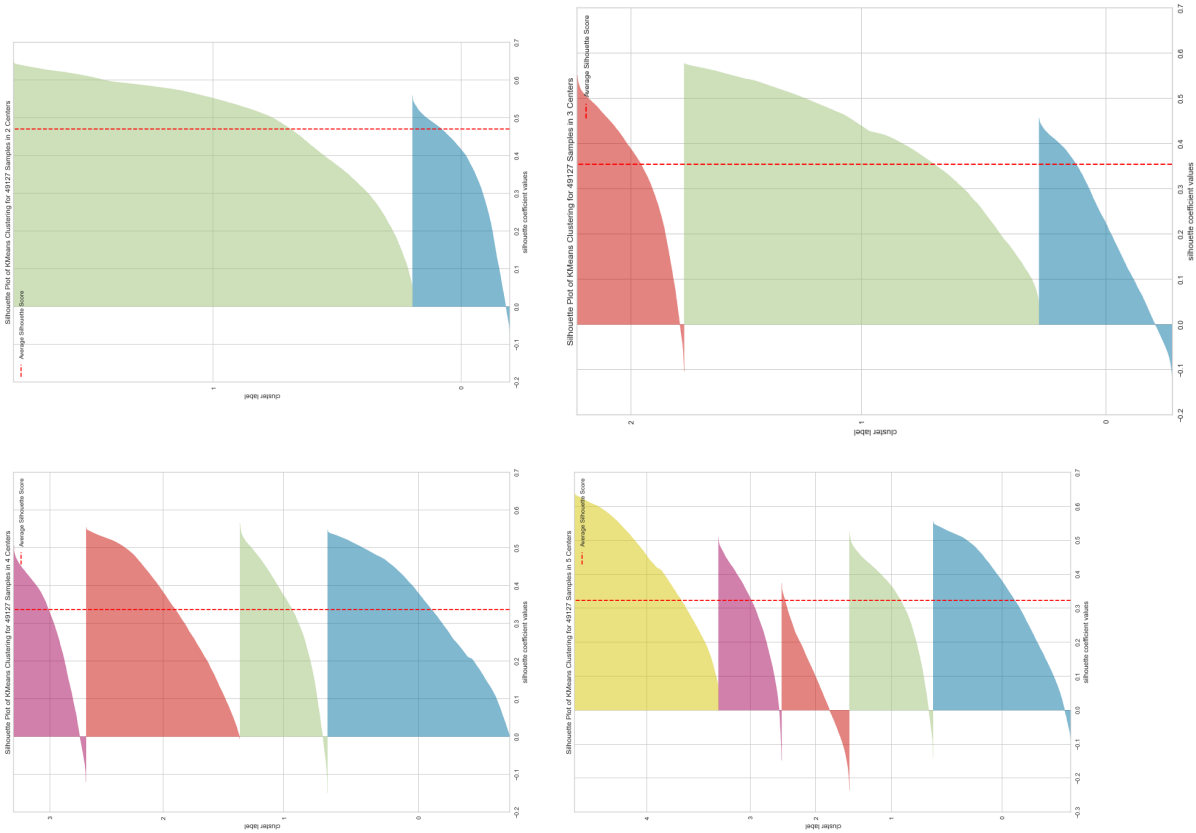


Figura 5.6: DB-Score

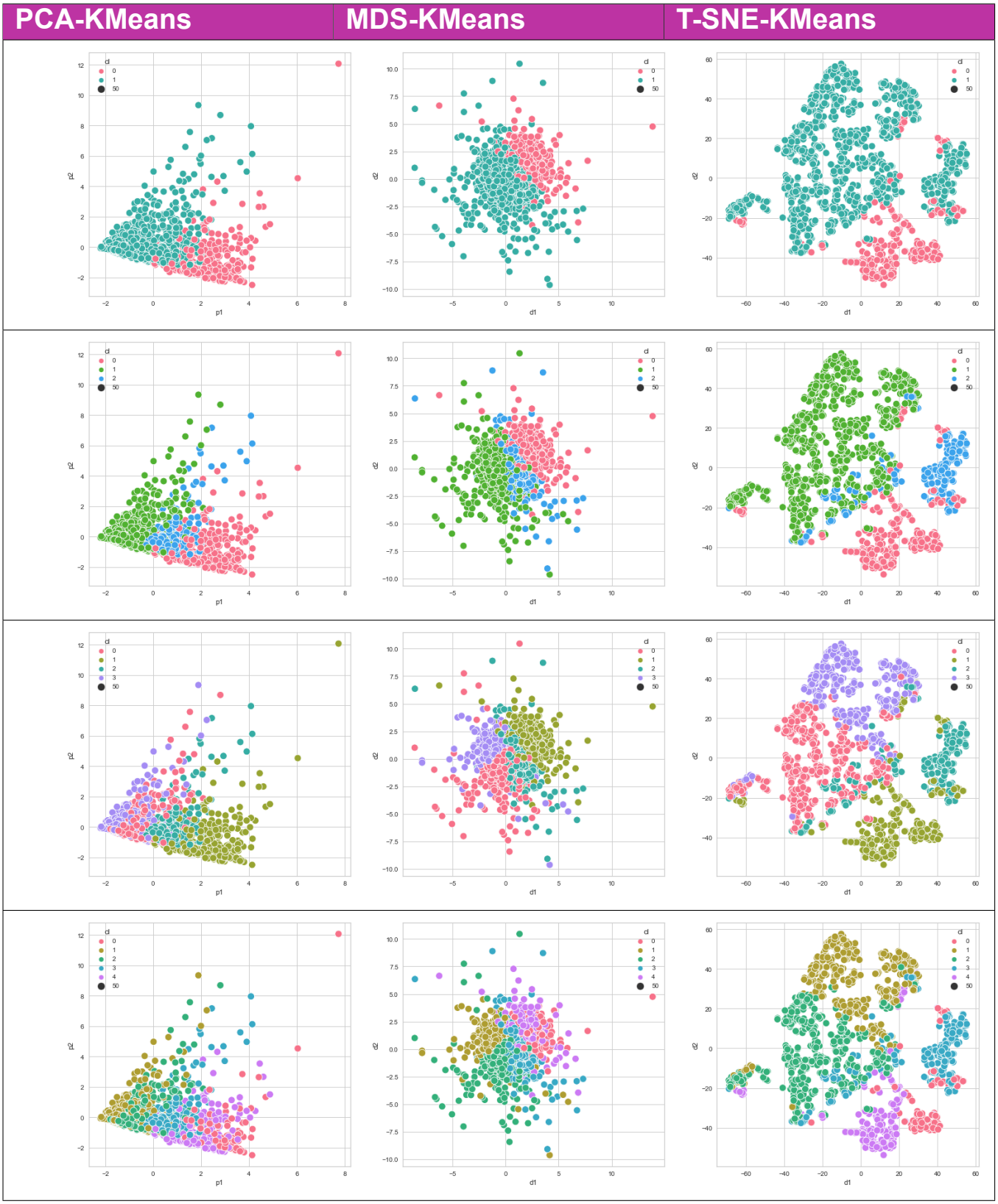
El Davies-Bouldin Index, busca medir la cohesión y la separación entre grupos, y determina el número de cluster donde la gráfica alcance el valor más pequeño, por lo que se sugiere 2 o 4 cluster.

Se decidió graficar las siluetas para 2,3,4 y 5 cluster para ver cuál nos convenía y se observó que con 4 cluster se obtenian grupos balanceados.



Cuadro 5.4: Visualizaciones de las siluetas de los grupos

Posteriormente se aplicó el algoritmo con el método de K-Means para 2,3,4 y 5 grupos obteniendo las siguientes visualizaciones.

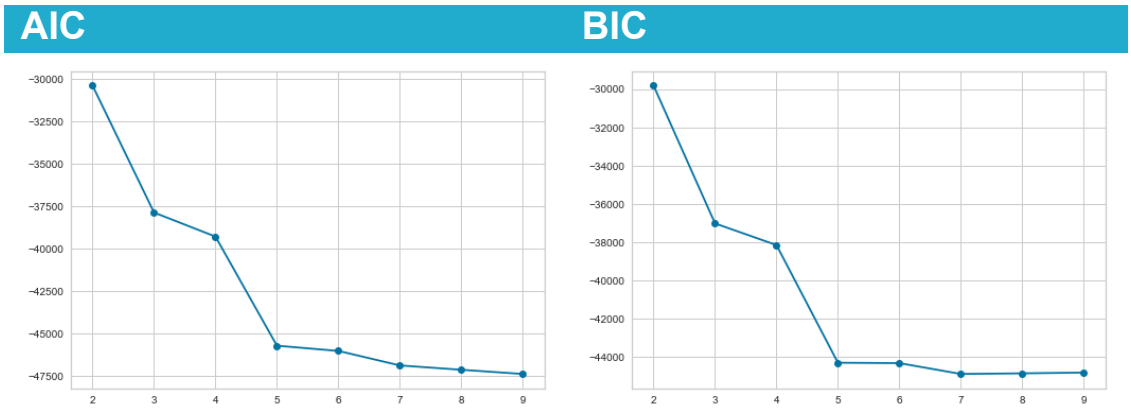


Cuadro 5.5: Visualizaciones de cluster de optimización por método de K-Means

5.2.3. Cluster de Densidad

El Cluster de Densidad se basa en la detección de en qué áreas existen concentraciones de puntos y dónde están separados por áreas vacías o con escasos puntos. Los puntos que no forman parte de un clúster se etiquetan como ruido; Y uno de los métodos para agrupar por densidad es el de *Gaussian Mixture*, este modelo asume que todos los puntos de datos son generados a través de una mezcla de un número finito de distribuciones gaussianas con parámetros desconocidos.

Para este tipo de cluster también es necesario partir de un número inicial de cluster, y para determinarlo tenemos el AIC, BIC y el método de la silueta.



Cuadro 5.6: Gráficas de AIC y BIC

Con estos métodos, el número de cluster se determina donde la gráfica alcance el valor más pequeño, por lo que se sugiere 5 cluster.

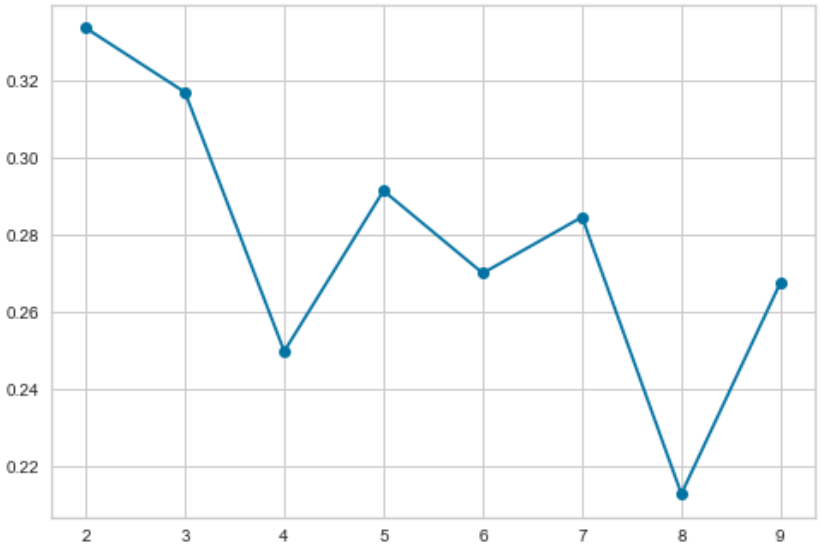
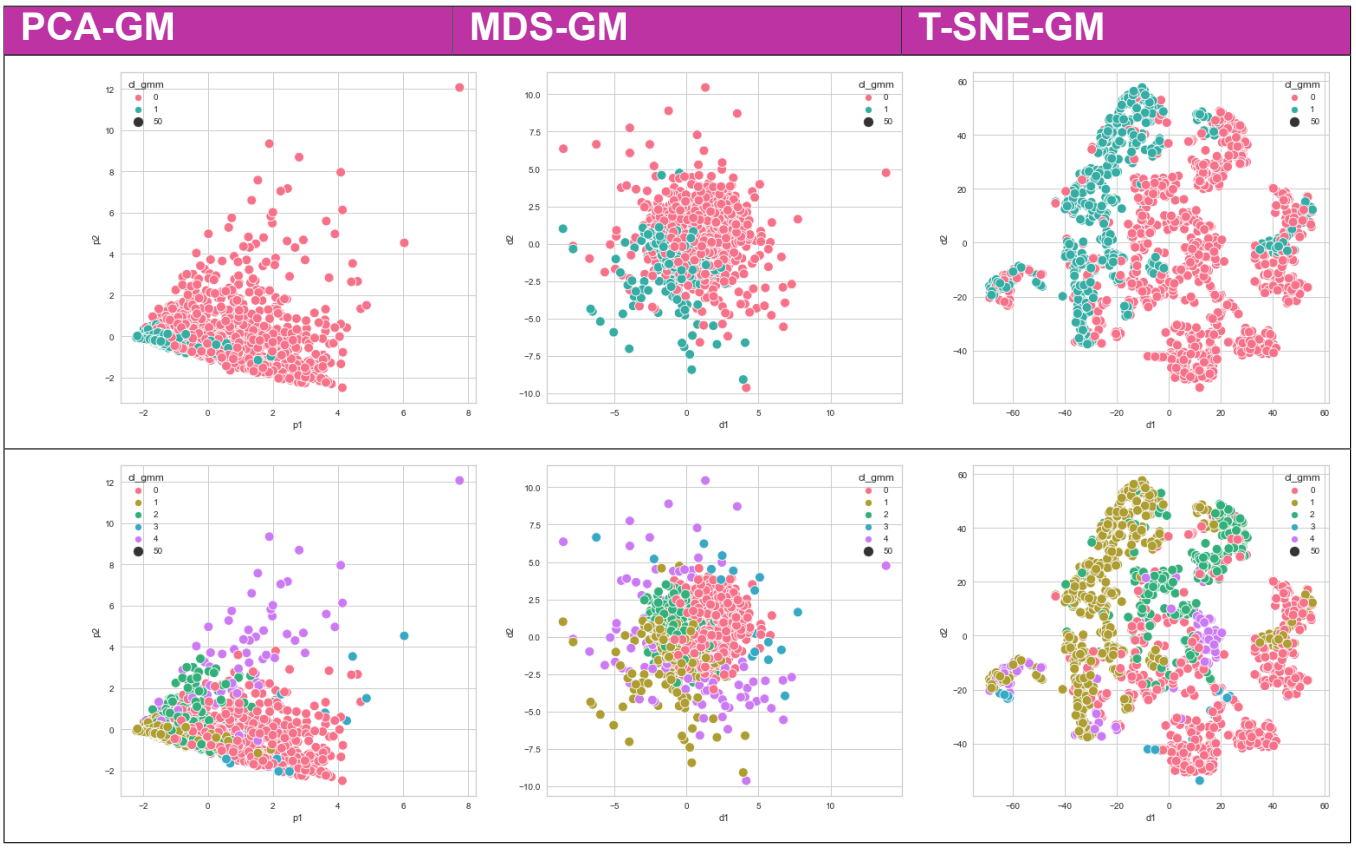


Figura 5.7: Método de la Silueta

El método de la silueta determina el número óptimo de cluster donde la gráfica se acerque más al 1, por lo que podemos observar sugiere 2 o 3 cluster.

Posteriormente se aplicó el algoritmo con el método de *Gaussian Mixture* para 2 y 5 grupos obteniendo las siguientes visualizaciones.



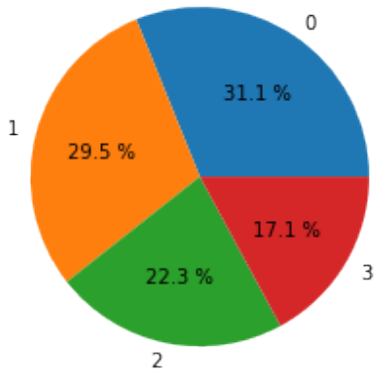
Cuadro 5.7: Visualizaciones de cluster de optimización por método de Gaussian Mixture

5.2.4. Perfilamiento

Una vez que escogimos un método de cluster, es importante poder describir los grupos obtenidos para identificar qué los caracteriza. En nuestro caso, después de evaluar los clusters obtenidos por métodos jerárquicos, de densidad y optimización, escogimos el cluster de cuatro grupos para T-SNE por método de WARD pues fue el que nos permitía diferenciar los grupos con más facilidad.

TSNE-WARD

Proporción de grupos



A continuación se presenta el perfilamiento de los cuatro grupos.

Cluster	# Episodios serie	Tiempo que se vio netflix	Tiempo que se jugó un juego	Categoria Drama	Contenido exclusivo	% Uso de funciones	Top 10	Categoria Sci-Fi	Categoria Niños
0	0.261012	0.311463	0.166503	-0.177534	-0.165435	-0.401723	-0.279055	0.236947	-0.520186
1	-0.850586	-0.471783	-0.152680	-0.490249	-0.555312	-0.535951	-0.665048	-0.384464	0.707558
2	0.844319	0.233789	-0.091959	-0.092965	-0.085545	2.074007	1.797918	0.311469	-0.310579
3	-0.318199	-0.329137	-0.107434	0.937453	0.974095	-0.269554	-0.167390	-0.329397	0.569504

Grupo de Ocio [0]	Grupo Contenido para Niños [1]
■ Vio series no tan largas ■ Ve netflix regularmente ■ Fue el grupo que más jugó juegos ■ Casi no vió la categoría de Drama ■ Casi no vio contenido de netflix ■ Usó poco los portales de netflix ■ Casi no vio contenido del top 10 de la región ■ Vio la categoría de ciencia ficción ■ Casi no vio contenido de niños ■ **Este grupo ve contenido de Sci-Fi y le gustan los juegos**	■ Vio las series más cortas ■ Casi no ve la plataforma ■ Uso menos los juegos ■ Casi no ve contenido de drama ■ Vio poco contenido exclusivo de netflix ■ Fue el que menos ocupó los portales ■ Vio menos el top de la región ■ Casi no vio Sci-Fi ■ Vio más contenido para niños ■ **Personas que usan la plataforma para sus hijos, casi no lo ven pero son los que más ven cosas de niños.**
Grupo Serie del Momento[2]	**Grupo Familiar [3]**
■ Ve las series más largas ■ Ve netflix regularmente ■ Jugó muy poco ■ No vió categoría de Drama ■ No vió contenido exclusivo ■ Uso todas las funciones ■ Vio mucho contenido del top ■ Vio mucha Sci-Fi ■ Casi no vio contenido de niños ■ **Por sus características durante el mes se podría decir que entró a ver contenido específico pues vio series largas del top de la región**	■ Vio series cortas ■ Casi no ve la plataforma ■ Jugó poco ■ Vio más la categoría de drama ■ Vio más contenido exclusivo de netflix ■ Uso poco los portales ■ Ve poco del top de la región ■ Casi no ve Sci-Fi ■ Ve contenido para niños ■ **Por ser un grupo donde se ven categorías variadas se infiere que es un perfil familiar.**

Cuadro 5.8: Perfilamiento de grupos

5.3. Conclusión

La **Modelación No Supervizada** utiliza algoritmos que basan su proceso de entrenamiento en un juego de datos sin etiquetas o clases previamente definidas. Es decir, a priori no se conoce ningún valor objetivo o de clase, ya sea categórico o numérico. El aprendizaje no supervisado está dedicado a las tareas de agrupamiento, también llamadas clustering o segmentación, donde su objetivo es encontrar grupos similares en el conjunto de datos.

Por lo anterior, es difícil encontrar un *'buen modelo'* pues, además de que son más pesados computacionalmente, son a prueba y error ya que no sabemos qué estamos buscando exactamente; por lo que podemos concluir que la modelación no supervizada resulta útil para encontrar información implícita en los datos que puede resultar en información valiosa.

Conclusión sobre los datos

Como nos centramos en el grupo de personas que no se suscribió podemos fijarnos en sus características para ofrecerles mejor contenido;

- Para el grupo de ocio, que es el que más juega, se podría poner una mayor variedad de juegos para que completara la suscripción.
- Para el grupo de contenido para niños, hacer recomendaciones para público adulto con el objetivo de también llamar la atención de los padres.
- Para el grupo de serie del momento, seguir sobre la línea de crear contenido original de Netflix, en especial en el área de Sci-Fi.
- Para el grupo familiar se podrían crear recomendaciones de películas familiares o recomendar más los juegos para niños.

6

Conclusión

Finalmente, regresando a nuestra problemática inicial donde tratamos de predecir si un usuario se suscribirá o no y qué es lo que afecta en su desición; podemos decir que, en un inicio podría preferirse un método de aprendizaje supervisado pues contamos con una variable objetivo y el mismo modelo da la significancia de las variables, sin embargo, el aprendizaje no supervisado también permite ver características importantes en los datos que pueden enfocarse a otras áreas del negocio como el marketing pues habría facilidad de identificar lo que le gusta al público e implementar estrategías para vender más.

Por lo anterior, se recomienda hacer un análisis conjunto de ambos tipos de modelación; primero, antes de cumplir el mes de prueba, predecir con el modelo de regresión si se suscribirá o no y, en caso de que no fuera a suscribirse, perfilarlo en alguno de los grupos encontrados con TSNE-WARD y, con base en el resultado, programar algoritmos personalizados de recomendación de programas para llamar la atención de los espectadores. Adicional, dentro de las recomendaciones de programas se sugiere incluir principalmente contenido exclusivo de Netflix, ya que en caso de gustar al usuario aumenta las probabilidades de suscripción.

7

Anexo

7.1. Código

A continuación se presenta el código del proyecto; desarrollado en python con la herramienta Jupyter Notebook.

Referencias

[1] Kaggle. *Netflix Appetency*. 2022. url: https://www.kaggle.com/competitions/netflix-appetency/data (visitado 10-05-2022).

========= Netflix Appetency ==========

---------- Más allá del mes de prueba ----------

Melissa Sofía Miranda Peñafiel

melissamiranda@ciencias.unam.mx (mailto:melissamiranda@ciencias.unam.mx)

En este Notebook desarrollaremos la limpieza de la base de datos

In [2]:
```
import os
import sys
import glob
import math
import random

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pylab as plt
from itertools import cycle

from random import choice, choices
pd.set_option("display.max_columns", 600)

plt.style.use("ggplot")
color_pal = plt.rcParams["axes.prop_cycle"].by_key()["color"]
color_cycle = cycle(plt.rcParams["axes.prop_cycle"].by_key()["color"])

os.getcwd()
```

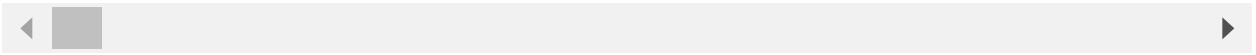
Out[2]:
```
'C:\\Users\\mime1001\\Documents\\Diplomado\\Proyecto\\Entrega 3'
```

In [2]:
```
# Importamos la base
df_train = pd.read_csv (os.getcwd() + '/Data/train.csv')
df_train
#df_train.info()
```

Out[2]:

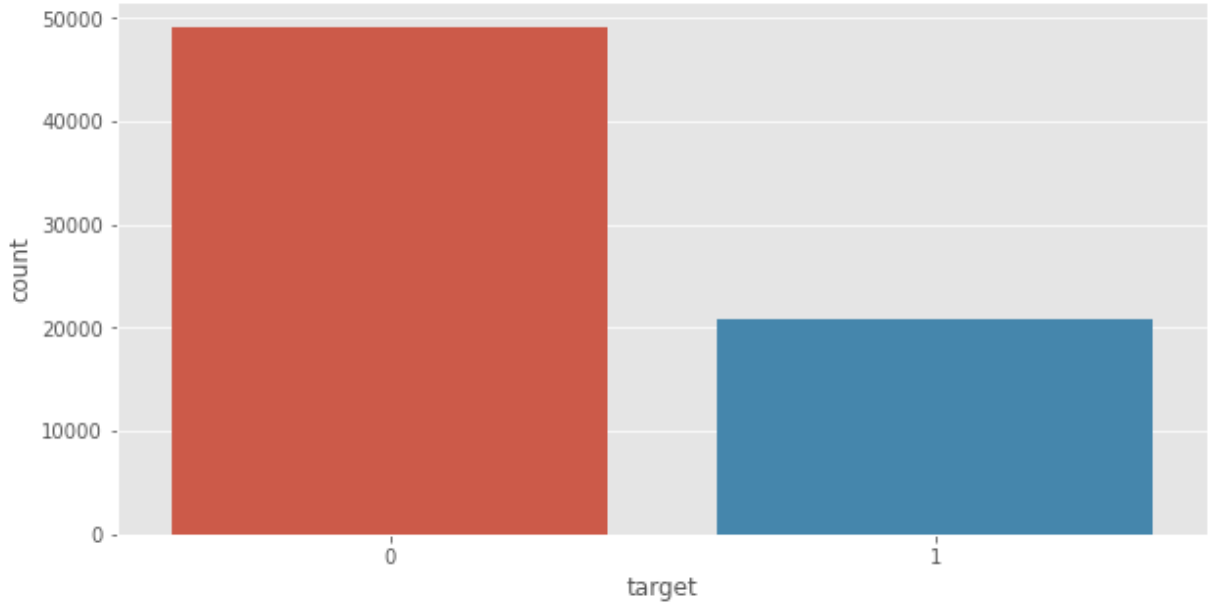
	id	target	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	fe
0	0	0	C0	C0	C1	C5	C11	37.560	54.756667	54
1	1	0	C0	C0	C3	C5	C1	NaN	NaN	
2	2	0	C0	C0	C3	C5	C2	NaN	NaN	
3	3	0	C0	C0	C1	C5	C1	NaN	NaN	
4	4	1	C0	C0	C3	C3	C11	37.480	37.480000	37
...	...	...	...	...	...	...	...	...	...	
69995	99994	0	C0	C0	C3	C1	C11	NaN	NaN	
69996	99996	0	C0	C0	C5	C5	C2	39.398	35.022000	45
69997	99997	0	C0	C0	C3	C2	C11	NaN	NaN	
69998	99998	0	C0	C0	C1	C5	C1	30.060	35.765000	35
69999	99999	1	C0	C0	C2	C5	C2	NaN	NaN	

70000 rows × 509 columns



In [3]:
```
# Vemos la variable de interés sobre la que se hará la predicción
target=df_train[['target']]
target

# Vemos cómo están distribuidas las clases de la variable target
fig, ax = plt.subplots(figsize=(10,5))
sns.countplot(df_train.target)
plt.show()
```

```
C:\Users\mime1001\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12, th
e only valid positional argument will be `data`, and passing other arguments wi
thout an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [4]:
```
display(df_train['target'].value_counts())
display(df_train['target'].value_counts(1))
```

```
0    49127
1    20873
Name: target, dtype: int64

0    0.701814
1    0.298186
Name: target, dtype: float64
```

In [5]:
```
# Quitamos la variable id
train=df_train.drop(columns=['id'])
train
print(train.shape)
# (70000, 508)
train.dtypes.value_counts()
# int64      273
# float64    143
# object      92
# dtype: int64

# Hay una variable duplicada, la quitamos
train = train.drop_duplicates().reset_index(drop=True)

# Veamos los datos faltantes
NanColumnsTrain=train.columns[train.isna().any()].tolist()
NanColumnsTrain
len(NanColumnsTrain) # ¿Cuántas columnas tienen NAs?

train[train.columns[train.isnull().any()]].isnull().sum()
train[train.columns[train.isnull().any()]].isnull().sum() * 100 / train.shape[0]
```

```
(70000, 508)
```

Out[5]:
```
feature_5      63.679481
feature_6      62.300890
feature_7      60.296576
feature_8      63.679481
feature_9      62.300890
                  ...
feature_256     2.072887
feature_257     2.765754
feature_258     2.285747
feature_259     2.072887
feature_260     2.072887
Length: 73, dtype: float64
```

In [6]:
```
###Vamos a ver los missing values

def missing_zero_values_table(df):
        zero_val = (df == 0.00).astype(int).sum(axis=0)
        mis_val = df.isnull().sum()
        mis_val_percent = 100 * df.isnull().sum() / len(df)
        mz_table = pd.concat([zero_val, mis_val, mis_val_percent], axis=1)
        mz_table = mz_table.rename(
        columns = {0 : 'Zero Values', 1 : 'Missing Values', 2 : '% of Total Value
        mz_table['Total Zero Missing Values'] = mz_table['Zero Values'] + mz_tabl
        mz_table['% Total Zero Missing Values'] = 100 * mz_table['Total Zero Miss
        mz_table['Data Type'] = df.dtypes
        mz_table = mz_table[
            mz_table.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)
        print ("Your selected dataframe has " + str(df.shape[1]) + " columns and
            "There are " + str(mz_table.shape[0]) +
              " columns that have missing values.")
        return mz_table



missing_zero_values_table(train)
```

```
Your selected dataframe has 508 columns and 69999 Rows.
There are 73 columns that have missing values.
```

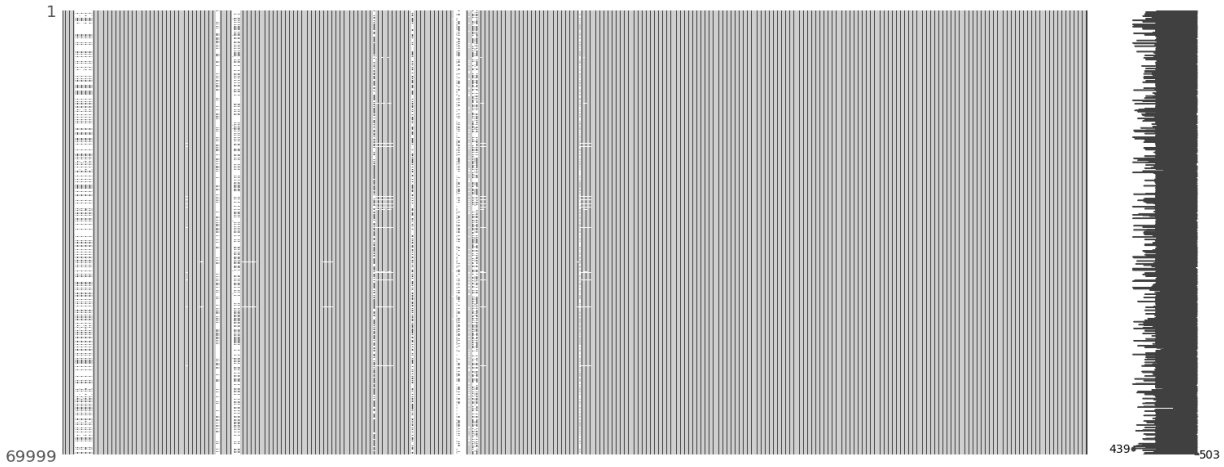
Out[6]:

	Zero Values	Missing Values	% of Total Values	Total Zero Missing Values	% Total Zero Missing Values	Data Type
feature_193	0	69999	100.0	69999	100.0	float64
feature_196	0	69999	100.0	69999	100.0	float64
feature_197	0	69999	100.0	69999	100.0	float64
feature_198	0	69999	100.0	69999	100.0	float64
feature_83	0	68823	98.3	68823	98.3	float64
...	...	...	...	...	...	...
feature_39	39088	85	0.1	39173	56.0	float64
feature_38	1370	85	0.1	1455	2.1	float64
feature_37	1712	85	0.1	1797	2.6	float64
feature_36	29213	85	0.1	29298	41.9	float64
feature_35	39080	85	0.1	39165	56.0	float64

73 rows × 6 columns

In [7]:
```
import missingno as msno
msno.matrix(train)
```

Out[7]: <AxesSubplot:>



In [8]:
```
# Vemos el número de NAs que tiene cada variable en orden descendiente
display(df_train.isna().sum(axis=0).sort_values(ascending=False).to_frame().T)

plt.figure(figsize=(25, 9))
```

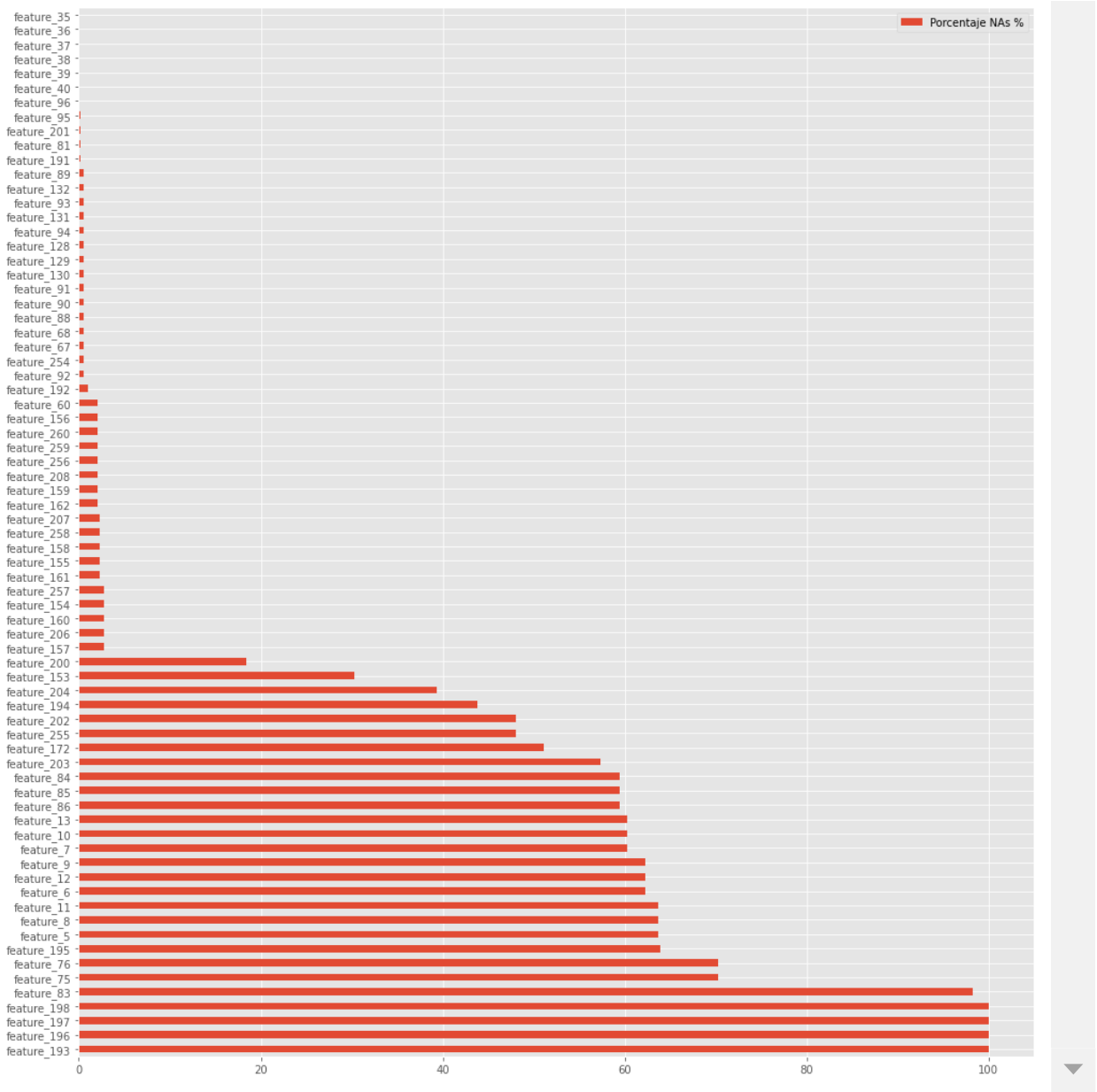
	feature_193	feature_196	feature_197	feature_198	feature_83	feature_76	feature_75	feature_19
0	70000	70000	70000	70000	68824	49208	49208	4472

Out[8]: <Figure size 1800x648 with 0 Axes>

<Figure size 1800x648 with 0 Axes>

In [9]:
```
print("No. de columnas con valores faltantes")
print(len(train.columns[train.isna().any()]))

print("No. de columnas sin datos faltantes")
print(len(train.columns[train.notna().all()]))

print("No. total de columnas en el dataframe")
print(len(train.columns))

# Función para graficar porcentaje de NAs por variable
import matplotlib.pyplot as plt
def plot_nas(df: pd.DataFrame):
    if df.isnull().sum().sum() != 0:
        na_df = (df.isnull().sum() / len(df)) * 100
        na_df = na_df.drop(na_df[na_df == 0].index).sort_values(ascending=False)
        missing_data = pd.DataFrame({'Porcentaje NAs %' :na_df})
        plot_width, plot_height = (16,18)
        plt.rcParams['figure.figsize'] = (plot_width,plot_height)
        missing_data.plot(kind = "barh")

        plt.show()
    else:
        print('No NAs found')

plot_nas(train)
train.shape
```

```
No. de columnas con valores faltantes
73
No. de columnas sin datos faltantes
435
No. total de columnas en el dataframe
508
```



Out[9]: (69999, 508)

In [10]:
```
null_values_columns=[]
percent_null_values=[]
for column in train.columns:
    percent_null_value=((train[column].isna().sum())/(len(train))) * 100
    if(percent_null_value > 0):
        null_values_columns.append(column)
        percent_null_values.append(percent_null_value)
df_null=pd.DataFrame(null_values_columns,columns=['column'])
df_null['percent_null_values']=percent_null_values
df_null.sort_values('percent_null_values',inplace=True)

# Vemos cuáles variables tienen más del 10% de NAs
large_null_columns=df_null[df_null['percent_null_values']>10]['column'].values

large_null_columns

# Quitamos estas variables
train.drop(large_null_columns,axis=1,inplace=True)
print(train.shape)

# Vemos que se hayan borrado
display(train.isna().sum(axis=0).sort_values(ascending=False).to_frame().T)
```

```
(69999, 480)
```

	feature_160	feature_157	feature_257	feature_154	feature_206	feature_161	feature_258	feature_
0	1936	1936	1936	1936	1936	1600	1600	

In [11]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69999 entries, 0 to 69998
Columns: 480 entries, target to feature_506
dtypes: float64(121), int64(273), object(86)
memory usage: 256.3+ MB
```

In [12]:
```
# De las variables object muestra cuántas categorías únicas tienen
display(train.select_dtypes('object').nunique().sort_values().to_frame().T)
```

	feature_55	feature_187	feature_188	feature_190	feature_249	feature_248	feature_0	feature_42
0	1	1	1	1	1	1	2	2

In [13]:
```
train.columns.tolist()
```

```
 'feature_280',
 'feature_281',
 'feature_282',
 'feature_283',
 'feature_284',
 'feature_285',
 'feature_286',
 'feature_287',
 'feature_288',
 'feature_289',
 'feature_290',
 'feature_291',
 'feature_292',
 'feature_293',
 'feature_294',
 'feature_295',
 'feature_296',
 'feature_297',
 'feature_298',
 'feature_299'
```

In [14]:
```
# Quitar variables que son fechas
# Originalmente van de la 191-204 pero varias se fueron en los NA
date_columns = ['feature_191','feature_192','feature_199','feature_201']

train[date_columns].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69999 entries, 0 to 69998
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   feature_191  69840 non-null  object
 1   feature_192  69331 non-null  object
 2   feature_199  69999 non-null  object
 3   feature_201  69875 non-null  object
dtypes: object(4)
memory usage: 2.1+ MB
```

In [15]:
```
# Quitar variables que son fechas
date_columns = ['feature_191','feature_192','feature_199','feature_201']

train[date_columns].head()

train = train.drop(columns=date_columns)

train.columns.tolist()
```

```
 'feature_488',
 'feature_489',
 'feature_490',
 'feature_491',
 'feature_492',
 'feature_493',
 'feature_494',
 'feature_495',
 'feature_496',
 'feature_497',
 'feature_498',
 'feature_499',
 'feature_500',
 'feature_501',
 'feature_502',
 'feature_503',
 'feature_504',
 'feature_505',
 'feature_506']
```

In [16]:
```
#Quitar variables que tienen sólo una categoría

single_feature_columns = ['feature_55', 'feature_187', 'feature_188','feature_190

train = train.drop(columns=single_feature_columns)
train.columns.tolist()
```

```
 'feature_225',
 'feature_226',
 'feature_227',
 'feature_228',
 'feature_229',
 'feature_230',
 'feature_231',
 'feature_232',
 'feature_233',
 'feature_234',
 'feature_235',
 'feature_236',
 'feature_237',
 'feature_238',
 'feature_239',
 'feature_240',
 'feature_241',
 'feature_242',
 'feature_243',
```

In [17]:
```
# Quitar variables con más de 100 valores distintos
more_cat_feature_columns = ['feature_17', 'feature_18', 'feature_19', 'feature_20
                            'feature_54', 'feature_133', 'feature_185', 'feature_
                            'feature_280', 'feature_287', 'feature_288']

#'feature_199','feature_192','feature_201','feature_191'---ya las habiamos elimin

train = train.drop(columns=more_cat_feature_columns)
train.columns.tolist()
```

Out[17]:
```
['target',
 'feature_0',
 'feature_1',
 'feature_2',
 'feature_3',
 'feature_4',
 'feature_14',
 'feature_15',
 'feature_16',
 'feature_23',
 'feature_24',
 'feature_25',
 'feature_26',
 'feature_27',
 'feature_28',
 'feature_29',
 'feature_30',
 'feature_31',
 'feature_32',
```

In [18]:
```
train.to_csv('Data/train_Cat.csv', index = False)
```

In [19]:
```
# ========================================================================
# ====================================== C H E C K P O I N T ==============
# ========================================================================
```

In [20]:
```
train = pd.read_csv (os.getcwd() + '/Data/train_Cat.csv')
train.head()
```

Out[20]:

	target	feature_0	feature_1	feature_2	feature_3	feature_4	feature_14	feature_15	feature_16
0	0	C0	C0	C1	C5	C11	0.0	0.000	C0
1	0	C0	C0	C3	C5	C1	0.0	0.000	C2
2	0	C0	C0	C3	C5	C2	0.0	0.000	C1
3	0	C0	C0	C1	C5	C1	0.0	0.000	C0
4	1	C0	C0	C3	C3	C11	0.0	0.258	C2

In [21]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69999 entries, 0 to 69998
Columns: 456 entries, target to feature_506
dtypes: float64(121), int64(273), object(62)
memory usage: 243.5+ MB
```

In [22]:
```
# Limpiemos las variables numéricas
### selecionamos todas las var num
target_column = 'target'
drop_columns = ['id', target_column]

num_columns = [col for col in train.select_dtypes(['int64', 'float64']).columns i
train[num_columns].head()
```

Out[22]:

	feature_14	feature_15	feature_25	feature_26	feature_35	feature_36	feature_37	feature_38
0	0.0	0.000	0	0	-17.527805	-77.832935	0.003995	15989.268352
1	0.0	0.000	0	0	0.000000	0.000000	-18.899991	5.169404
2	0.0	0.000	0	0	0.000000	0.000000	7.167159	-33.030314
3	0.0	0.000	0	0	0.000000	0.000000	-4.065911	126.551718
4	0.0	0.258	4716	26992	0.000000	0.000000	32.781988	-29.673155

In [23]:
```
# Número de valores únicos por variable
train[num_columns].nunique().sort_values().to_frame().T
```

Out[23]:

	feature_409	feature_478	feature_427	feature_476	feature_432	feature_474	feature_395	feature_
0	1	1	1	1	1	1	1	

In [24]:
```
single_feature_columns = [
       'feature_461', 'feature_474', 'feature_472', 'feature_409',
       'feature_469', 'feature_467', 'feature_411', 'feature_465',
       'feature_312', 'feature_463', 'feature_313', 'feature_375',
       'feature_427', 'feature_412', 'feature_413', 'feature_455',
       'feature_453', 'feature_419', 'feature_451', 'feature_100',
       'feature_449', 'feature_421', 'feature_445', 'feature_433',
       'feature_432', 'feature_459', 'feature_476', 'feature_405',
       'feature_425', 'feature_498', 'feature_227', 'feature_228',
       'feature_149', 'feature_495', 'feature_385', 'feature_493',
       'feature_478', 'feature_491', 'feature_389', 'feature_489',
       'feature_252', 'feature_487', 'feature_502', 'feature_505',
       'feature_484', 'feature_395', 'feature_482', 'feature_377',
       'feature_358', 'feature_480', 'feature_397', 'feature_357',
       'feature_500']


#train[single_feature_columns].info()
```

In [25]:
```
# Quitamos las variables que tienen sólo un valor único ie son constantes
single_feature_columns = [
       'feature_461', 'feature_474', 'feature_472', 'feature_409',
       'feature_469', 'feature_467', 'feature_411', 'feature_465',
       'feature_312', 'feature_463', 'feature_313', 'feature_375',
       'feature_427', 'feature_412', 'feature_413', 'feature_455',
       'feature_453', 'feature_419', 'feature_451', 'feature_100',
       'feature_449', 'feature_421', 'feature_445', 'feature_433',
       'feature_432', 'feature_459', 'feature_476', 'feature_405',
       'feature_425', 'feature_498', 'feature_227', 'feature_228',
       'feature_149', 'feature_495', 'feature_385', 'feature_493',
       'feature_478', 'feature_491', 'feature_389', 'feature_489',
       'feature_252', 'feature_487', 'feature_502', 'feature_505',
       'feature_484', 'feature_395', 'feature_482', 'feature_377',
       'feature_358', 'feature_480', 'feature_397', 'feature_357',
       'feature_500']

train = train.drop(columns=single_feature_columns)
```

In [26]:
```
################ Vamos a ver de nuevo los nulos

print(missing_zero_values_table(train).index)
missing_zero_values_table(train)
```

```
Your selected dataframe has 403 columns and 69999 Rows.
There are 42 columns that have missing values.
Index(['feature_206', 'feature_257', 'feature_157', 'feature_160',
       'feature_154', 'feature_207', 'feature_161', 'feature_258',
       'feature_158', 'feature_155', 'feature_256', 'feature_60',
       'feature_208', 'feature_162', 'feature_159', 'feature_259',
       'feature_156', 'feature_260', 'feature_254', 'feature_129',
       'feature_130', 'feature_90', 'feature_94', 'feature_67', 'feature_68',
       'feature_88', 'feature_91', 'feature_92', 'feature_128', 'feature_93',
       'feature_131', 'feature_89', 'feature_132', 'feature_81', 'feature_95',
       'feature_39', 'feature_37', 'feature_38', 'feature_36', 'feature_40',
       'feature_96', 'feature_35'],
      dtype='object')
Your selected dataframe has 403 columns and 69999 Rows.
There are 42 columns that have missing values.
```

Out[26]:

	Zero Values	Missing Values	% of Total Values	Total Zero Missing Values	% Total Zero Missing Values	Data Type
feature_206	3	1936	2.8	1939	2.8	float64
feature_257	0	1936	2.8	1936	2.8	float64
feature_157	40514	1936	2.8	42450	60.6	float64
feature_160	40408	1936	2.8	42344	60.5	float64
feature_154	40444	1936	2.8	42380	60.5	float64
feature_207	0	1600	2.3	1600	2.3	float64
feature_161	35529	1600	2.3	37129	53.0	float64
feature_258	0	1600	2.3	1600	2.3	float64
feature_158	35703	1600	2.3	37303	53.3	float64
feature_155	35587	1600	2.3	37187	53.1	float64
feature_256	0	1451	2.1	1451	2.1	float64
feature_60	8531	1451	2.1	9982	14.3	float64
feature_208	0	1451	2.1	1451	2.1	float64
feature_162	27847	1451	2.1	29298	41.9	float64
feature_159	28049	1451	2.1	29500	42.1	float64
feature_259	149	1451	2.1	1600	2.3	float64
feature_156	27919	1451	2.1	29370	42.0	float64
feature_260	0	1451	2.1	1451	2.1	float64
feature_254	67612	411	0.6	68023	97.2	float64
feature_129	41202	411	0.6	41613	59.4	float64
feature_130	64374	411	0.6	64785	92.6	float64

	Zero Values	Missing Values	% of Total Values	Total Zero Missing Values	% Total Zero Missing Values	Data Type
feature_90	51763	411	0.6	52174	74.5	float64
feature_94	40443	411	0.6	40854	58.4	float64
feature_67	40443	411	0.6	40854	58.4	float64
feature_68	0	411	0.6	411	0.6	float64
feature_88	46825	411	0.6	47236	67.5	float64
feature_91	12050	411	0.6	12461	17.8	float64
feature_92	30684	411	0.6	31095	44.4	float64
feature_128	40443	411	0.6	40854	58.4	float64
feature_93	1239	382	0.5	1621	2.3	float64
feature_131	28218	382	0.5	28600	40.9	float64
feature_89	41817	382	0.5	42199	60.3	float64
feature_132	14633	382	0.5	15015	21.5	float64
feature_81	914	159	0.2	1073	1.5	float64
feature_95	28502	98	0.1	28600	40.9	float64
feature_39	39088	85	0.1	39173	56.0	float64
feature_37	1712	85	0.1	1797	2.6	float64
feature_38	1370	85	0.1	1455	2.1	float64
feature_36	29213	85	0.1	29298	41.9	float64
feature_40	29233	85	0.1	29318	41.9	float64
feature_96	14930	85	0.1	15015	21.5	float64
feature_35	39080	85	0.1	39165	56.0	float64

In [27]:
```
######## visualizar las variables
missing_var=['feature_206', 'feature_257', 'feature_157', 'feature_160',
       'feature_154', 'feature_207', 'feature_161', 'feature_258',
       'feature_158', 'feature_155', 'feature_256', 'feature_60',
       'feature_208', 'feature_162', 'feature_159', 'feature_259',
       'feature_156', 'feature_260', 'feature_254', 'feature_129',
       'feature_130', 'feature_90', 'feature_94', 'feature_67', 'feature_68',
       'feature_88', 'feature_91', 'feature_92', 'feature_128', 'feature_93',
       'feature_131', 'feature_89', 'feature_132', 'feature_81', 'feature_95',
       'feature_39', 'feature_37', 'feature_38', 'feature_36', 'feature_40',
       'feature_96', 'feature_35']


plt.rcParams["figure.figsize"] = [25, 25]
train[missing_var].hist(bins=30)
```

Out[27]:
```
array([[<AxesSubplot:title={'center':'feature_206'}>,
        <AxesSubplot:title={'center':'feature_257'}>,
        <AxesSubplot:title={'center':'feature_157'}>,
        <AxesSubplot:title={'center':'feature_160'}>,
        <AxesSubplot:title={'center':'feature_154'}>,
        <AxesSubplot:title={'center':'feature_207'}>],
       [<AxesSubplot:title={'center':'feature_161'}>,
        <AxesSubplot:title={'center':'feature_258'}>,
        <AxesSubplot:title={'center':'feature_158'}>,
        <AxesSubplot:title={'center':'feature_155'}>,
        <AxesSubplot:title={'center':'feature_256'}>,
        <AxesSubplot:title={'center':'feature_60'}>],
       [<AxesSubplot:title={'center':'feature_208'}>,
        <AxesSubplot:title={'center':'feature_162'}>,
        <AxesSubplot:title={'center':'feature_159'}>,
        <AxesSubplot:title={'center':'feature_259'}>,
        <AxesSubplot:title={'center':'feature_156'}>,
        <AxesSubplot:title={'center':'feature_260'}>],
       [<AxesSubplot:title={'center':'feature_254'}>,
        <AxesSubplot:title={'center':'feature_129'}>,
        <AxesSubplot:title={'center':'feature_130'}>,
        <AxesSubplot:title={'center':'feature_90'}>,
        <AxesSubplot:title={'center':'feature_94'}>,
        <AxesSubplot:title={'center':'feature_67'}>],
       [<AxesSubplot:title={'center':'feature_68'}>,
        <AxesSubplot:title={'center':'feature_88'}>,
        <AxesSubplot:title={'center':'feature_91'}>,
        <AxesSubplot:title={'center':'feature_92'}>,
        <AxesSubplot:title={'center':'feature_128'}>,
        <AxesSubplot:title={'center':'feature_93'}>],
       [<AxesSubplot:title={'center':'feature_131'}>,
        <AxesSubplot:title={'center':'feature_89'}>,
        <AxesSubplot:title={'center':'feature_132'}>,
        <AxesSubplot:title={'center':'feature_81'}>,
        <AxesSubplot:title={'center':'feature_95'}>,
        <AxesSubplot:title={'center':'feature_39'}>],
       [<AxesSubplot:title={'center':'feature_37'}>,
        <AxesSubplot:title={'center':'feature_38'}>,
        <AxesSubplot:title={'center':'feature_36'}>,
        <AxesSubplot:title={'center':'feature_40'}>,
```

```
<AxesSubplot:title={'center':'feature_96'}>,
<AxesSubplot:title={'center':'feature_35'}>]], dtype=object)
```

In [28]:
```
# Hacemos un primer overview de nuetsras varibles categoricas
for col in missing_var:
    display(train[col].value_counts().reset_index())
```

33250	8.203488e+05	1

33251 rows × 2 columns

	index	feature_259
0	83.0	1813
1	88.0	1785
2	93.0	1638
3	78.0	1548
4	98.0	1349
...	...	...
1862	2923.0	1
1863	34983.0	1
1864	951.0	1

In [29]:
```
train.to_csv('Data/train_pre_imp.csv', index = False)
```

In [30]:
```
# ================================================================================
# ======================================= C H E C K P O I N T=====================
# ================================================================================
```

In [3]:
```
train = pd.read_csv (os.getcwd() + '/Data/train_pre_imp.csv')
train
#train[missing_var].info()
```

Out[3]:

	target	feature_0	feature_1	feature_2	feature_3	feature_4	feature_14	feature_15	feature_
0	0	C0	C0	C1	C5	C11	0.000	0.000000	
1	0	C0	C0	C3	C5	C1	0.000	0.000000	
2	0	C0	C0	C3	C5	C2	0.000	0.000000	
3	0	C0	C0	C1	C5	C1	0.000	0.000000	
4	1	C0	C0	C3	C3	C11	0.000	0.258000	
...	...	...	...	...	...	...	...	...	
69994	0	C0	C0	C3	C1	C11	0.000	0.000000	
69995	0	C0	C0	C5	C5	C2	6.928	13.721667	
69996	0	C0	C0	C3	C2	C11	0.000	0.000000	
69997	0	C0	C0	C1	C5	C1	0.000	0.000000	
69998	1	C0	C0	C2	C5	C2	0.000	0.000000	

69999 rows × 403 columns

In [4]:
```
target_column = 'target'
drop_columns = ['id', target_column]

missing_var=['feature_206', 'feature_257', 'feature_157', 'feature_160',
       'feature_154', 'feature_207', 'feature_161', 'feature_258',
       'feature_158', 'feature_155', 'feature_256', 'feature_60',
       'feature_208', 'feature_162', 'feature_159', 'feature_259',
       'feature_156', 'feature_260', 'feature_254', 'feature_129',
       'feature_130', 'feature_90', 'feature_94', 'feature_67', 'feature_68',
       'feature_88', 'feature_91', 'feature_92', 'feature_128', 'feature_93',
       'feature_131', 'feature_89', 'feature_132', 'feature_81', 'feature_95',
       'feature_39', 'feature_37', 'feature_38', 'feature_36', 'feature_40',
       'feature_96', 'feature_35']

###Vamos a ver los missing values

def missing_zero_values_table(df):
        zero_val = (df == 0.00).astype(int).sum(axis=0)
        mis_val = df.isnull().sum()
        mis_val_percent = 100 * df.isnull().sum() / len(df)
        mz_table = pd.concat([zero_val, mis_val, mis_val_percent], axis=1)
        mz_table = mz_table.rename(
        columns = {0 : 'Zero Values', 1 : 'Missing Values', 2 : '% of Total Value
        mz_table['Total Zero Missing Values'] = mz_table['Zero Values'] + mz_tabl
        mz_table['% Total Zero Missing Values'] = 100 * mz_table['Total Zero Miss
        mz_table['Data Type'] = df.dtypes
        mz_table = mz_table[
            mz_table.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)
        print ("Your selected dataframe has " + str(df.shape[1]) + " columns and
            "There are " + str(mz_table.shape[0]) +
              " columns that have missing values.")
        return mz_table



missing_zero_values_table(train)
```

```
Your selected dataframe has 403 columns and 69999 Rows.
There are 42 columns that have missing values.
```

Out[4]:

	Zero Values	Missing Values	% of Total Values	Total Zero Missing Values	% Total Zero Missing Values	Data Type
feature_206	3	1936	2.8	1939	2.8	float64
feature_257	0	1936	2.8	1936	2.8	float64
feature_157	40514	1936	2.8	42450	60.6	float64
feature_160	40408	1936	2.8	42344	60.5	float64
feature_154	40444	1936	2.8	42380	60.5	float64
feature_207	0	1600	2.3	1600	2.3	float64
feature_161	35529	1600	2.3	37129	53.0	float64
feature_258	0	1600	2.3	1600	2.3	float64

	Zero Values	Missing Values	% of Total Values	Total Zero Missing Values	% Total Zero Missing Values	Data Type
feature_158	35703	1600	2.3	37303	53.3	float64
feature_155	35587	1600	2.3	37187	53.1	float64
feature_256	0	1451	2.1	1451	2.1	float64
feature_60	8531	1451	2.1	9982	14.3	float64
feature_208	0	1451	2.1	1451	2.1	float64
feature_162	27847	1451	2.1	29298	41.9	float64
feature_159	28049	1451	2.1	29500	42.1	float64
feature_259	149	1451	2.1	1600	2.3	float64
feature_156	27919	1451	2.1	29370	42.0	float64
feature_260	0	1451	2.1	1451	2.1	float64
feature_254	67612	411	0.6	68023	97.2	float64
feature_129	41202	411	0.6	41613	59.4	float64
feature_130	64374	411	0.6	64785	92.6	float64
feature_90	51763	411	0.6	52174	74.5	float64
feature_94	40443	411	0.6	40854	58.4	float64
feature_67	40443	411	0.6	40854	58.4	float64
feature_68	0	411	0.6	411	0.6	float64
feature_88	46825	411	0.6	47236	67.5	float64
feature_91	12050	411	0.6	12461	17.8	float64
feature_92	30684	411	0.6	31095	44.4	float64
feature_128	40443	411	0.6	40854	58.4	float64
feature_93	1239	382	0.5	1621	2.3	float64
feature_131	28218	382	0.5	28600	40.9	float64
feature_89	41817	382	0.5	42199	60.3	float64
feature_132	14633	382	0.5	15015	21.5	float64
feature_81	914	159	0.2	1073	1.5	float64
feature_95	28502	98	0.1	28600	40.9	float64
feature_39	39088	85	0.1	39173	56.0	float64
feature_37	1712	85	0.1	1797	2.6	float64
feature_38	1370	85	0.1	1455	2.1	float64
feature_36	29213	85	0.1	29298	41.9	float64
feature_40	29233	85	0.1	29318	41.9	float64
feature_96	14930	85	0.1	15015	21.5	float64
feature_35	39080	85	0.1	39165	56.0	float64

In [5]:
```
# ========================================================================================
# ======================================= imputamos con la MEDIANA ===============
# ========================================================================================
from sklearn.impute import SimpleImputer

train2=train.copy()

for col in missing_var:
    train[col]=SimpleImputer(missing_values=np.nan, strategy='median').fit_transf


#### Vemos que ya no haya missing vals
missing_zero_values_table(train)
```

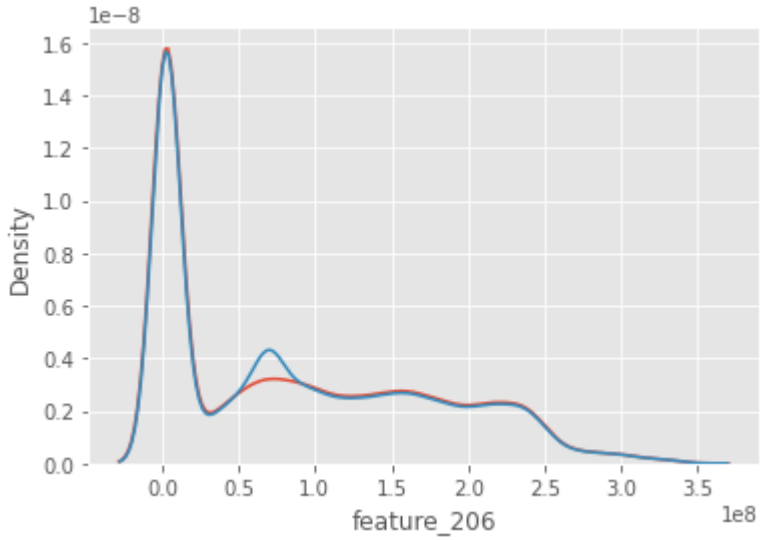
```
Your selected dataframe has 403 columns and 69999 Rows.
There are 0 columns that have missing values.
```

Out[5]:

	Zero Values	Missing Values	% of Total Values	Total Zero Missing Values	% Total Zero Missing Values	Data Type

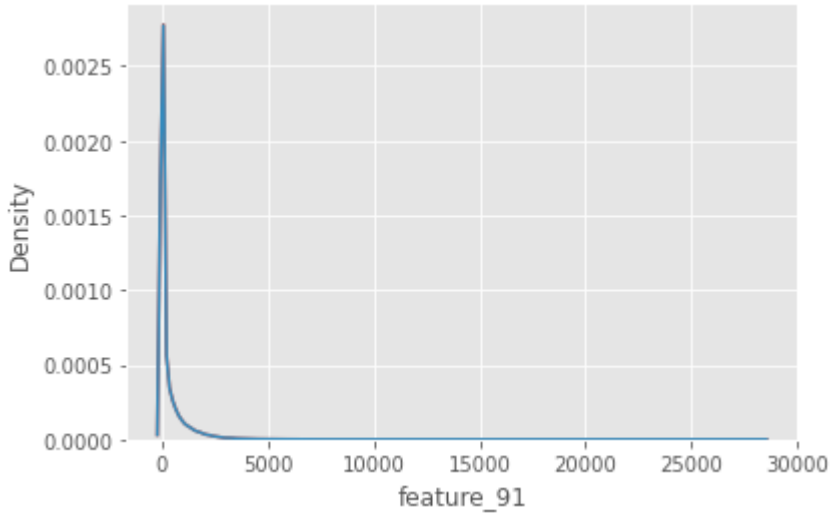
In [6]:
```
missing_var=['feature_206', 'feature_257', 'feature_157', 'feature_160',
       'feature_154', 'feature_207', 'feature_161', 'feature_258',
       'feature_158', 'feature_155', 'feature_256', 'feature_60',
       'feature_208', 'feature_162', 'feature_159', 'feature_259',
       'feature_156', 'feature_260', 'feature_254', 'feature_129',
       'feature_130', 'feature_90', 'feature_94', 'feature_67', 'feature_68',
       'feature_88', 'feature_91', 'feature_92', 'feature_128', 'feature_93',
       'feature_131', 'feature_89', 'feature_132', 'feature_81', 'feature_95',
       'feature_39', 'feature_37', 'feature_38', 'feature_36', 'feature_40',
       'feature_96', 'feature_35']


#Vamos a ver que no se lastimó la distribución
sns.kdeplot(train2['feature_206'])
sns.kdeplot(train['feature_206'])


#for col in missing_var:
 #   display(sns.kdeplot(train2[col]))
 #   display(sns.kdeplot(train[col]))
```

Out[6]: <AxesSubplot:xlabel='feature_206', ylabel='Density'>



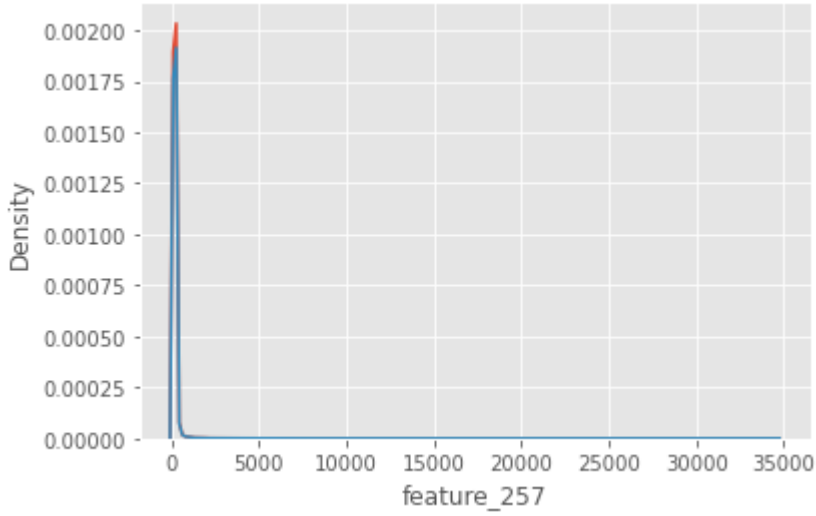
In [7]:
```
sns.kdeplot(train2['feature_91'])
sns.kdeplot(train['feature_91'])
```

Out[7]: <AxesSubplot:xlabel='feature_91', ylabel='Density'>



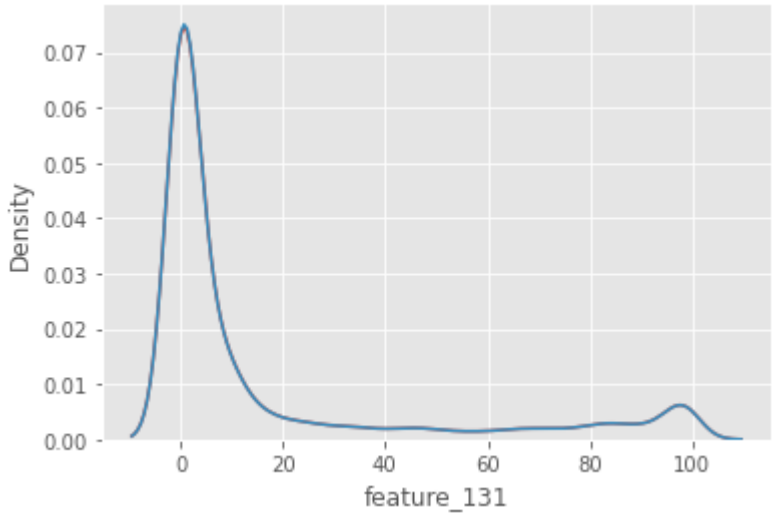
In [8]:
```
sns.kdeplot(train2['feature_257'])
sns.kdeplot(train['feature_257'])
```

Out[8]: <AxesSubplot:xlabel='feature_257', ylabel='Density'>



In [9]:
```
sns.kdeplot(train2['feature_131'])
sns.kdeplot(train['feature_131'])
```

Out[9]: <AxesSubplot:xlabel='feature_131', ylabel='Density'>



In [34]:
```
plt.rcParams["figure.figsize"] = [25, 25]
train[missing_var].hist(bins=30)
```

Out[34]:
```
array([[<AxesSubplot:title={'center':'feature_206'}>,
        <AxesSubplot:title={'center':'feature_257'}>,
        <AxesSubplot:title={'center':'feature_157'}>,
        <AxesSubplot:title={'center':'feature_160'}>,
        <AxesSubplot:title={'center':'feature_154'}>,
        <AxesSubplot:title={'center':'feature_207'}>],
       [<AxesSubplot:title={'center':'feature_161'}>,
        <AxesSubplot:title={'center':'feature_258'}>,
        <AxesSubplot:title={'center':'feature_158'}>,
        <AxesSubplot:title={'center':'feature_155'}>,
        <AxesSubplot:title={'center':'feature_256'}>,
        <AxesSubplot:title={'center':'feature_60'}>],
       [<AxesSubplot:title={'center':'feature_208'}>,
        <AxesSubplot:title={'center':'feature_162'}>,
        <AxesSubplot:title={'center':'feature_159'}>,
        <AxesSubplot:title={'center':'feature_259'}>,
        <AxesSubplot:title={'center':'feature_156'}>,
        <AxesSubplot:title={'center':'feature_260'}>],
       [<AxesSubplot:title={'center':'feature_254'}>,
        <AxesSubplot:title={'center':'feature_129'}>,
        <AxesSubplot:title={'center':'feature_130'}>,
        <AxesSubplot:title={'center':'feature_90'}>,
        <AxesSubplot:title={'center':'feature_94'}>,
        <AxesSubplot:title={'center':'feature_67'}>],
       [<AxesSubplot:title={'center':'feature_68'}>,
        <AxesSubplot:title={'center':'feature_88'}>,
        <AxesSubplot:title={'center':'feature_91'}>,
        <AxesSubplot:title={'center':'feature_92'}>,
        <AxesSubplot:title={'center':'feature_128'}>,
        <AxesSubplot:title={'center':'feature_93'}>],
       [<AxesSubplot:title={'center':'feature_131'}>,
        <AxesSubplot:title={'center':'feature_89'}>,
        <AxesSubplot:title={'center':'feature_132'}>,
        <AxesSubplot:title={'center':'feature_81'}>,
        <AxesSubplot:title={'center':'feature_95'}>,
        <AxesSubplot:title={'center':'feature_39'}>],
       [<AxesSubplot:title={'center':'feature_37'}>,
        <AxesSubplot:title={'center':'feature_38'}>,
        <AxesSubplot:title={'center':'feature_36'}>,
        <AxesSubplot:title={'center':'feature_40'}>,
        <AxesSubplot:title={'center':'feature_96'}>,
        <AxesSubplot:title={'center':'feature_35'}>]], dtype=object)
```

In [35]:
```
train.columns
```

Out[35]:
```
Index(['target', 'feature_0', 'feature_1', 'feature_2', 'feature_3',
       'feature_4', 'feature_14', 'feature_15', 'feature_16', 'feature_23',
       ...
       'feature_490', 'feature_492', 'feature_494', 'feature_496',
       'feature_497', 'feature_499', 'feature_501', 'feature_503',
       'feature_504', 'feature_506'],
      dtype='object', length=403)
```

In [36]:
```
# Hacemos un primer overview de nuetsras varibles categoricas
for col in train.columns:
    display(train[col].value_counts().reset_index())
```

	index	feature_490
0	0	67780
1	-99	2219

	index	feature_492
0	0	67780
1	99	2219

	index	feature_494
0	0	67780
1	99	2219

In [37]:
```
train.to_csv('Data/train_data_imp_1.csv', index = False)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
#================================================ FIN ============================
```

In [ ]:

======= Modelación Supervizada ========

Melissa Sofía Miranda Peñafiel

melissamiranda@ciencias.unam.mx (mailto:melissamiranda@ciencias.unam.mx)

En este notebook vamos a desarrollar modelos supervizados; una regresión logística y un árbol de de desición

>>>>>> Arquitectura de Datos <<<<<<<

---------- Modelación Supervizada ----------

In [1]:
```
#Cargamos librerias
import os
import sys
import glob
import math
import random

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pylab as plt
from itertools import cycle

from random import choice, choices
pd.set_option("display.max_columns", 600)

plt.style.use("ggplot")
color_pal = plt.rcParams["axes.prop_cycle"].by_key()["color"]
color_cycle = cycle(plt.rcParams["axes.prop_cycle"].by_key()["color"])

os.getcwd()
```

Out[1]:
```
'C:\\Users\\mime1001\\Documents\\Diplomado\\Proyecto\\Entrega 3'
```

In [2]:
```
# Importamos la base ya limpia
df_train = pd.read_csv (os.getcwd() + '/Data/train_data_imp_1.csv')
train=df_train
display(df_train.info())
df_train
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69999 entries, 0 to 69998
Columns: 403 entries, target to feature_506
dtypes: float64(120), int64(221), object(62)
memory usage: 215.2+ MB
```

None

Out[2]:

	target	feature_0	feature_1	feature_2	feature_3	feature_4	feature_14	feature_15	feature_
0	0	C0	C0	C1	C5	C11	0.000	0.000000	
1	0	C0	C0	C3	C5	C1	0.000	0.000000	
2	0	C0	C0	C3	C5	C2	0.000	0.000000	
3	0	C0	C0	C1	C5	C1	0.000	0.000000	
4	1	C0	C0	C3	C3	C11	0.000	0.258000	
...	...	...	...	...	...	...	...	...	
69994	0	C0	C0	C3	C1	C11	0.000	0.000000	
69995	0	C0	C0	C5	C5	C2	6.928	13.721667	
69996	0	C0	C0	C3	C2	C11	0.000	0.000000	
69997	0	C0	C0	C1	C5	C1	0.000	0.000000	
69998	1	C0	C0	C2	C5	C2	0.000	0.000000	

69999 rows × 403 columns

In [8]:
```
#Vemos valores ausentes
def missing_zero_values_table(df):
        zero_val = (df == 0.00).astype(int).sum(axis=0)
        mis_val = df.isnull().sum()
        mis_val_percent = 100 * df.isnull().sum() / len(df)
        mz_table = pd.concat([zero_val, mis_val, mis_val_percent], axis=1)
        mz_table = mz_table.rename(
        columns = {0 : 'Zero Values', 1 : 'Missing Values', 2 : '% of Total Value
        mz_table['Total Zero Missing Values'] = mz_table['Zero Values'] + mz_tabl
        mz_table['% Total Zero Missing Values'] = 100 * mz_table['Total Zero Miss
        mz_table['Data Type'] = df.dtypes
        mz_table = mz_table[
            mz_table.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)
        print ("Your selected dataframe has " + str(df.shape[1]) + " columns and
            "There are " + str(mz_table.shape[0]) +
              " columns that have missing values.")
        return mz_table



missing_zero_values_table(train)
```

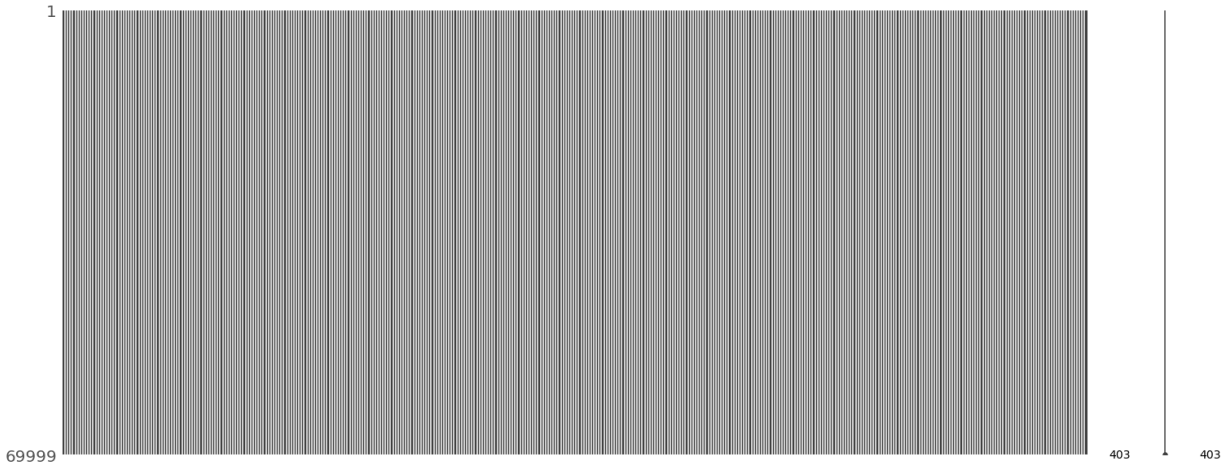
```
Your selected dataframe has 403 columns and 69999 Rows.
There are 0 columns that have missing values.
```

Out[8]:

	Zero Values	Missing Values	% of Total Values	Total Zero Missing Values	% Total Zero Missing Values	Data Type

In [9]:
```
import missingno as msno
msno.matrix(train)
```

Out[9]: <AxesSubplot:>



In [10]:
```
display(df_train.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69999 entries, 0 to 69998
Columns: 403 entries, target to feature_506
dtypes: float64(120), int64(221), object(62)
memory usage: 215.2+ MB

None
```

In [6]:
```
#################### CORRELACIÓN ENTRE LAS VARIABLES NUMÉRICAS Y TARGET

target_column = 'target'
drop_columns = ['id', target_column]


num_columns = [col for col in df_train.select_dtypes(['int64', 'float64']).columr

corr_df = df_train[['target'] + num_columns].corr(method='pearson')
corr_df
```

Out[6]:

	target	feature_14	feature_15	feature_25	feature_26	feature_35	feature_36	featu
target	1.000000	0.015662	0.017865	-0.046023	-0.049170	0.006516	0.005990	-0.0
feature_14	0.015662	1.000000	0.737889	0.083695	0.087566	-0.000460	-0.000614	-0.0
feature_15	0.017865	0.737889	1.000000	0.090409	0.106344	-0.000633	-0.000776	-0.0
feature_25	-0.046023	0.083695	0.090409	1.000000	0.908288	-0.001029	-0.001952	-0.0
feature_26	-0.049170	0.087566	0.106344	0.908288	1.000000	-0.001339	-0.001986	-0.0
...	...	...	...	...	...	...	...	
feature_499	-0.071182	-0.008594	-0.007689	0.017465	0.051215	-0.000403	0.001195	0.0
feature_501	0.071182	0.008594	0.007689	-0.017465	-0.051215	0.000403	-0.001195	-0.0
feature_503	0.071182	0.008594	0.007689	-0.017465	-0.051215	0.000403	-0.001195	-0.0
feature_504	-0.071182	-0.008594	-0.007689	0.017465	0.051215	-0.000403	0.001195	0.0
feature_506	0.071182	0.008594	0.007689	-0.017465	-0.051215	0.000403	-0.001195	-0.0

341 rows × 341 columns

In [12]:
```
a=corr_df['target'].abs().sort_values(ascending=False).to_frame()
aa= a.index.tolist()
aa
a.T
```

Out[12]:

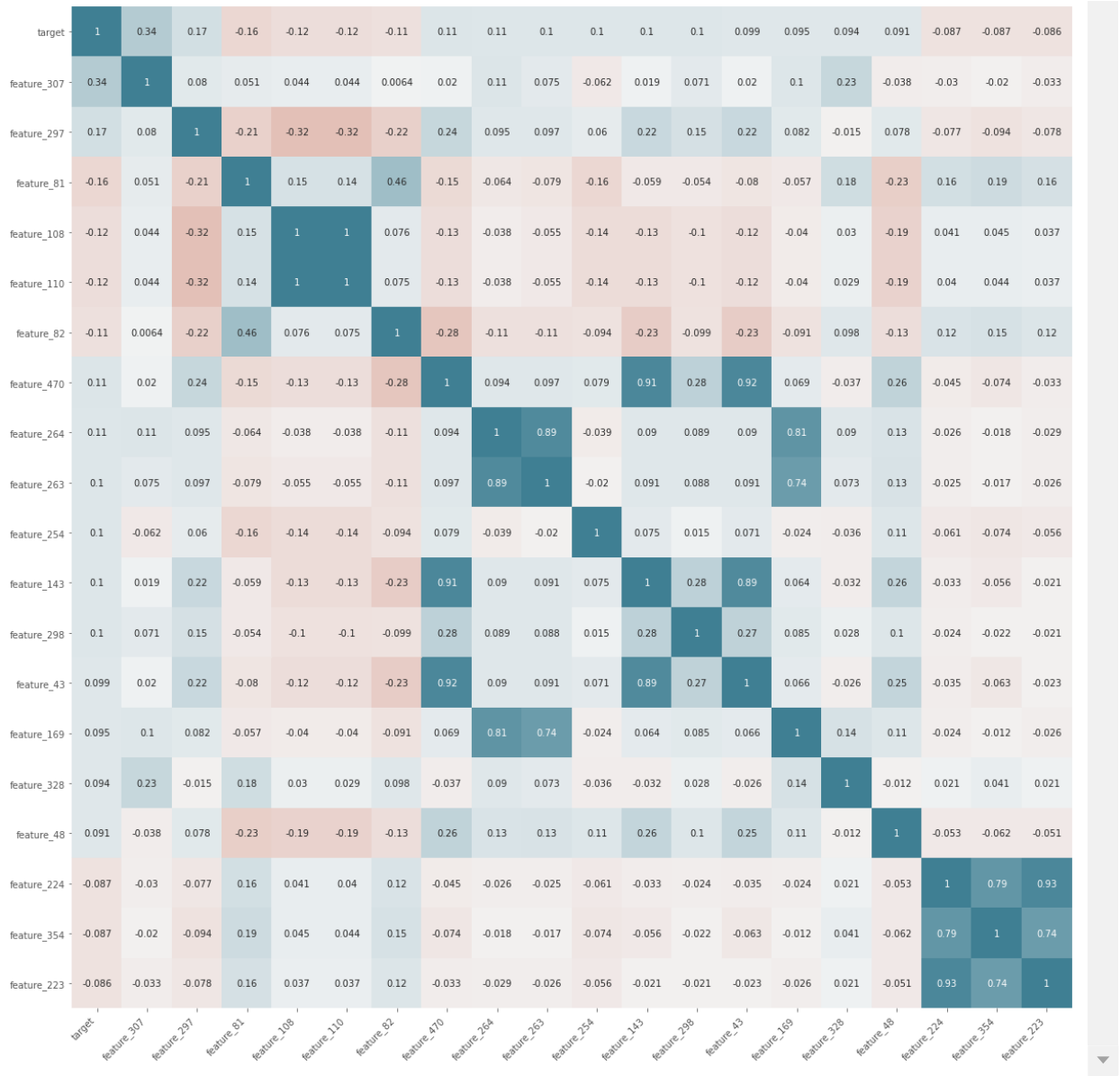
	target	feature_307	feature_297	feature_81	feature_108	feature_110	feature_82	feature_47
target	1.0	0.335277	0.166039	0.156985	0.122208	0.121303	0.113788	0.10976

In [13]:
```
# #sacamos las primeras 20
# a=corr_df['target'].abs().sort_values(ascending=False).to_frame()
# aa= a.index.tolist()

top_20=['target','feature_307','feature_297', 'feature_81', 'feature_108', 'featu
 'feature_264','feature_263', 'feature_254', 'feature_143', 'feature_298', 'featu
 'feature_48', 'feature_224', 'feature_354', 'feature_223']

data_corr= df_train[top_20].corr(method='pearson')

data_corr.head(2)
```

Out[13]:

	target	feature_307	feature_297	feature_81	feature_108	feature_110	feature_82	fe
target	1.000000	0.335277	0.166039	-0.156985	-0.122208	-0.121303	-0.113788	
feature_307	0.335277	1.000000	0.080474	0.050983	0.043558	0.043582	0.006428	

In [14]:
```
# Obtenemos la matriz de correlación con mapa de calor
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(20, 20))

sns.heatmap(
    data_corr,
    annot     = True,
    cbar      = False,
    annot_kws = {"size": 10},
    vmin      = -1,
    vmax      = 1,
    center    = 0,
    cmap      = sns.diverging_palette(20, 220, n=200),
    square    = True,
    ax        = ax
)

ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation = 45,
    horizontalalignment = 'right',
)

ax.tick_params(labelsize = 10)
```



========== MODELACIÓN ==========

In [3]:
```
################################################ MODELOS ######################
######No podemos usar test porque no hay manera de contrastar

import matplotlib.pyplot as plt
import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
from sklearn.feature_selection import SelectKBest, f_classif, f_regression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler
import cufflinks as cf
cf.go_offline()
```

In [4]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69999 entries, 0 to 69998
Columns: 403 entries, target to feature_506
dtypes: float64(120), int64(221), object(62)
memory usage: 215.2+ MB
```

In [7]:
```
######################################## Regresion logistica ###################

random.seed(10)
num_columns = [col for col in df_train.select_dtypes(['int64', 'float64']).columr

x = df_train[num_columns]
y = df_train['target'].values


# Particionamos nuestros datos para entrenar el modelo
x_train_c, x_valid, y_train_c, y_valid = train_test_split(x, y, test_size = 0.2,

# Particionamos el train para tener train y test
x_train, x_test, y_train, y_test = train_test_split(x_train_c, y_train_c, test_si


tables=[x_train, x_test, y_train, y_test,x_train_c, x_valid, y_train_c, y_valid]


for tab in tables:
    print(str('tab=') + str(len(tab)))

display(len(x_train_c))
display(len(x_train))
```

```
tab=44799
tab=11200
tab=44799
tab=11200
tab=55999
tab=14000
tab=55999
tab=14000

55999

44799
```

In [10]:
```
# Aunque ya vimos las variables mas correlacionadas veremos que tanto contrasta
# con un K-best
kb = SelectKBest(k = 15, score_func=f_classif)
kb
```

Out[10]:
```
SelectKBest(k=15)
```

In [11]:
```
#Obtenemos las mejores 15 variables
kb.fit(x_train, y_train)

ls_best = [x for x, y in zip(x.columns, kb.get_support()) if y]
ls_best


# top_20=['target','feature_307','feature_297', 'feature_81', 'feature_108', 'fea
# 'feature_264','feature_263', 'feature_254', 'feature_143', 'feature_298', 'feat
# 'feature_48', 'feature_224', 'feature_354', 'feature_223']
```

Out[11]:
```
['feature_43',
 'feature_81',
 'feature_82',
 'feature_108',
 'feature_110',
 'feature_143',
 'feature_169',
 'feature_254',
 'feature_263',
 'feature_264',
 'feature_297',
 'feature_298',
 'feature_307',
 'feature_328',
 'feature_470']
```

In [40]:
```
#extraemos los datos en las mejores variables
#tables=[x_train, x_test, y_train, y_test,x_train_c, x_valid, y_train_c, y_valid]
X_train = x_train[ls_best]
X_test = x_test[ls_best]
X_valid=x_valid[ls_best]

display(X_train.info())
display(X_test.info())
display(X_valid.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 44799 entries, 46619 to 44160
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   feature_43   44799 non-null  int64  
 1   feature_81   44799 non-null  float64
 2   feature_82   44799 non-null  int64  
 3   feature_108  44799 non-null  int64  
 4   feature_110  44799 non-null  int64  
 5   feature_143  44799 non-null  int64  
 6   feature_169  44799 non-null  float64
 7   feature_254  44799 non-null  float64
 8   feature_263  44799 non-null  int64  
 9   feature_264  44799 non-null  int64  
 10  feature_297  44799 non-null  int64  
 11  feature_298  44799 non-null  int64  
 12  feature_307  44799 non-null  int64  
 13  feature_328  44799 non-null  int64  
 14  feature_470  44799 non-null  int64  
dtypes: float64(3), int64(12)
memory usage: 5.5 MB

None

<class 'pandas.core.frame.DataFrame'>
Int64Index: 11200 entries, 56716 to 55994
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   feature_43   11200 non-null  int64  
 1   feature_81   11200 non-null  float64
 2   feature_82   11200 non-null  int64  
 3   feature_108  11200 non-null  int64  
 4   feature_110  11200 non-null  int64  
 5   feature_143  11200 non-null  int64  
 6   feature_169  11200 non-null  float64
 7   feature_254  11200 non-null  float64
 8   feature_263  11200 non-null  int64  
 9   feature_264  11200 non-null  int64  
 10  feature_297  11200 non-null  int64  
 11  feature_298  11200 non-null  int64  
 12  feature_307  11200 non-null  int64  
 13  feature_328  11200 non-null  int64  
 14  feature_470  11200 non-null  int64  
dtypes: float64(3), int64(12)
memory usage: 1.4 MB
```

```
None

<class 'pandas.core.frame.DataFrame'>
Int64Index: 14000 entries, 38105 to 3140
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   feature_43   14000 non-null  int64  
 1   feature_81   14000 non-null  float64
 2   feature_82   14000 non-null  int64  
 3   feature_108  14000 non-null  int64  
 4   feature_110  14000 non-null  int64  
 5   feature_143  14000 non-null  int64  
 6   feature_169  14000 non-null  float64
 7   feature_254  14000 non-null  float64
 8   feature_263  14000 non-null  int64  
 9   feature_264  14000 non-null  int64  
 10  feature_297  14000 non-null  int64  
 11  feature_298  14000 non-null  int64  
 12  feature_307  14000 non-null  int64  
 13  feature_328  14000 non-null  int64  
 14  feature_470  14000 non-null  int64  
dtypes: float64(3), int64(12)
memory usage: 1.7 MB

None
```

In [41]:
```
#Traemos la regresión lineal
from sklearn.linear_model import LogisticRegression
logistic=LogisticRegression(solver='liblinear',penalty='l1')
logistic.fit(X_train,y_train)
```

Out[41]:
```
LogisticRegression(penalty='l1', solver='liblinear')
```

In [42]:
```
#predecimos en test
logistic.predict(X_test)
```

Out[42]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [43]:
```
#sacamos las probas de cada entrada
logistic.predict_proba(X_test)
```

Out[43]:
```
array([[0.73975617, 0.26024383],
       [0.87063371, 0.12936629],
       [0.89538196, 0.10461804],
       ...,
       [0.86561043, 0.13438957],
       [0.67122894, 0.32877106],
       [0.80718682, 0.19281318]])
```

In [44]:
```
#sacamos la segunda entrada de las probas
logistic.predict_proba(X_test)[:,1]
```

Out[44]:
```
array([0.26024383, 0.12936629, 0.10461804, ..., 0.13438957, 0.32877106,
       0.19281318])
```

In [14]:
```
from sklearn.metrics import roc_auc_score,accuracy_score,confusion_matrix
def metricas2(model,Xv,yv): #Mide efectividad de un Modelo Predictivo
    print( " Roc Validate: %.3f" %roc_auc_score(y_score=model.predict_proba(Xv)[:
    print( " Acc Validate: %.3f" %accuracy_score(y_pred=model.predict(Xv),y_true=
    print( " Matrix Conf Validate: ", "\n",confusion_matrix(y_pred=model.predict(
```

In [47]:
```
# Obtenemos las métricas de clasificación para cada conjunto de datos
metricas2(logistic,X_train,y_train)
```

```
 Roc Validate: 0.750
 Acc Validate: 0.751
 Matrix Conf Validate:
 [[28607  2812]
 [ 8331  5049]]
```

In [48]:
```
metricas2(logistic,X_test,y_test)
```

```
 Roc Validate: 0.752
 Acc Validate: 0.746
 Matrix Conf Validate:
 [[7086  683]
 [2158 1273]]
```

In [49]:
```
metricas2(logistic,X_valid,y_valid)
```

```
 Roc Validate: 0.758
 Acc Validate: 0.757
 Matrix Conf Validate:
 [[9011  928]
 [2476 1585]]
```

In [50]:
```
#pd.to_pickle(logistic,'Regresion_Log_75.pkl')
```

In [56]:
```
#sacamos el F1 score
#qué tan bueno es prediciendo la clase 1
from sklearn.metrics import f1_score
round(f1_score(y_pred=logistic.predict(X_valid),y_true=y_valid),2)
```

Out[56]:
```
0.48
```

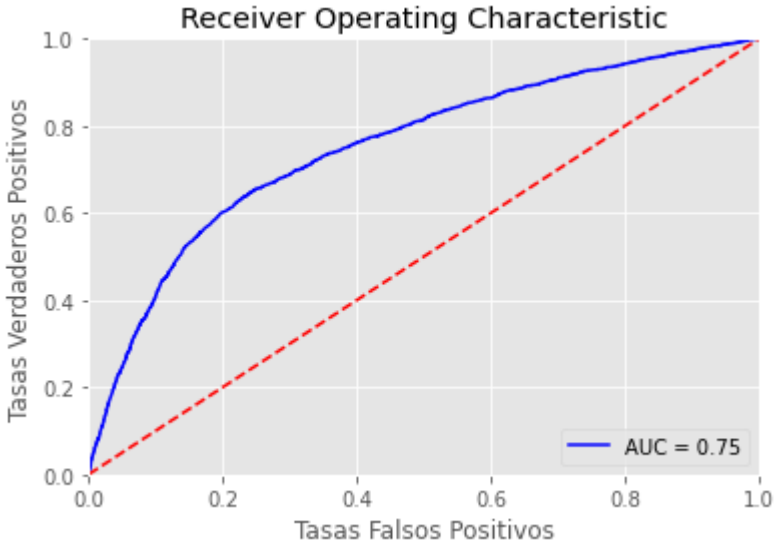
In [57]:
```
#F1 score para la clase cero
round(f1_score(y_pred=logistic.predict(X_valid),y_true=y_valid,pos_label=0),2)
```

Out[57]:
```
0.84
```

In [59]:
```
# Obtenemos las métricas de clasificación para las clases
from sklearn.metrics import classification_report
print(classification_report(y_valid,logistic.predict(X_valid)))
```

```
              precision    recall  f1-score   support

           0       0.78      0.91      0.84      9939
           1       0.63      0.39      0.48      4061

    accuracy                           0.76     14000
   macro avg       0.71      0.65      0.66     14000
weighted avg       0.74      0.76      0.74     14000
```

In [60]:
```
# Graficamos curva ROC
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
fpr, tpr, thresh = roc_curve(y_test,logistic.predict_proba(X_test)[:,1])
plt.title('Receiver Operating Characteristic')
auc = roc_auc_score(y_test,logistic.predict_proba(X_test)[:,1])
plt.plot(fpr, tpr, 'b', label = f'AUC = {auc.round(2)}')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('Tasas Verdaderos Positivos')
plt.xlabel('Tasas Falsos Positivos')
plt.show()
```



In [12]:
```
######################################## Árbol de decisión ####################
random.seed(10)
#num_columns = [col for col in df_train.select_dtypes(['int64', 'float64']).colum

#x = df_train[num_columns]
#y = df_train['target'].values


# Particionamos nuestros datos para entrenar el modelo
#x_train_c, x_valid, y_train_c, y_valid = train_test_split(x, y, test_size = 0.2,

# Particionamos el train para tener train y test
#x_train, x_test, y_train, y_test = train_test_split(x_train_c, y_train_c, test_s

X_train = x_train[ls_best]
X_test = x_test[ls_best]
X_valid=x_valid[ls_best]
# Primer intento sin paramgrid

from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
```

Out[12]:
```
DecisionTreeClassifier

DecisionTreeClassifier()
```

In [15]:
```
metricas2(classifier,X_train,y_train)
```

```
Roc Validate: 0.967
Acc Validate: 0.899
Matrix Conf Validate:
[[30506   913]
[ 3600  9780]]
```

In [16]:
```
metricas2(classifier,X_test,y_test)
```

```
Roc Validate: 0.628
Acc Validate: 0.684
Matrix Conf Validate:
[[6332 1437]
[2100 1331]]
```

In [17]:
```
metricas2(classifier,X_valid,y_valid)
```

```
Roc Validate: 0.626
Acc Validate: 0.690
Matrix Conf Validate:
[[8062 1877]
[2461 1600]]
```

In [18]:
```
best_tree=pd.read_pickle('Arbol_75.pkl')
```

In [83]:
```
# Hacemos un paramgrid random para encontrar una buena malla
'''param_grid = {"max_depth": range(1, 6),
              "max_features": ["auto", "sqrt", "log2"],
              "min_samples_leaf": [x/100 for x in range(5, 70, 1)] + [x for x in r
               "criterion": ["gini", "entropy"],
               "splitter": ["best", "random"],
               "class_weight": ["balanced", None]}

from sklearn.model_selection import RandomizedSearchCV
Random=RandomizedSearchCV(DecisionTreeClassifier(),param_grid,scoring='roc_auc',r
```

In [84]:
```
'''Random.fit(X_train,y_train)'''
```

```
Fitting 3 folds for each of 1000 candidates, totalling 3000 fits
```

Out[84]:

▸ **RandomizedSearchCV**

▸ **estimator: DecisionTreeClassifier**

▸ DecisionTreeClassifier

In [19]:
```
#best_tree=Random.best_estimator_
best_tree.get_params()

# Esta es la mejor estimación; toma como criterio la entropía
# profundidad 4, acota la variable por raiz cuadrada, nos dice que
# al menos hay que tener una muestra por hoja [min samples leaf], y si
# hay menos de 2 ya no se vuelve a particionar (el de abajo)
```

Out[19]:
```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'entropy',
 'max_depth': 5,
 'max_features': 'log2',
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 7,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

In [20]:
```
metricas2(best_tree,X_train,y_train)
```

```
 Roc Validate: 0.747
 Acc Validate: 0.754
 Matrix Conf Validate:
 [[28081  3338]
 [ 7685  5695]]
```

In [21]:
```
metricas2(best_tree,X_test,y_test)
```

```
Roc Validate: 0.747
Acc Validate: 0.750
Matrix Conf Validate:
[[6953  816]
[1986 1445]]
```

In [22]:
```
metricas2(best_tree,X_valid,y_valid)
```

```
Roc Validate: 0.752
Acc Validate: 0.755
Matrix Conf Validate:
[[8826 1113]
[2311 1750]]
```

In [91]:
```
#pd.to_pickle(best_tree,'Arbol_62.pkl') #semilla 1
#pd.to_pickle(best_tree,'Arbol_75.pkl') #semilla 1
```

In [23]:
```
# Qué tanto influye la var para tener más pureza
best_tree.feature_importances_
importances=best_tree.feature_importances_

feature_importance= sorted(zip(importances, list(x.columns)), reverse=True)

df_importances = pd.DataFrame(feature_importance, columns=['importance', 'feature
importance= list(df_importances['importance'])
feature= list(df_importances['feature'])

df_importances
```

Out[23]:

	importance	feature
0	0.583337	feature_45
1	0.277502	feature_15
2	0.056728	feature_43
3	0.026202	feature_38
4	0.013520	feature_40
5	0.008272	feature_47
6	0.008057	feature_46
7	0.007707	feature_39
8	0.005600	feature_25
9	0.004821	feature_44
10	0.003185	feature_26
11	0.001979	feature_37
12	0.001630	feature_35
13	0.001192	feature_36
14	0.000268	feature_14

In [26]:
```
#dibujamos el árbol
plt.figure(figsize=(110,110))
plot_tree(best_tree,feature_names=X_train.columns)
```

```
 Text(0.6875, 0.4166666666666667, 'feature_82 <= 41.5\nentropy = 0.988\nsampl
es = 2777\nvalue = [1206, 1571]'),
 Text(0.65625, 0.25, 'feature_82 <= 28.5\nentropy = 0.977\nsamples = 1191\nva
lue = [489, 702]'),
 Text(0.640625, 0.08333333333333333, 'entropy = 0.993\nsamples = 708\nvalue =
[319, 389]'),
 Text(0.671875, 0.08333333333333333, 'entropy = 0.936\nsamples = 483\nvalue =
[170, 313]'),
 Text(0.71875, 0.25, 'feature_298 <= 0.5\nentropy = 0.993\nsamples = 1586\nva
lue = [717, 869]'),
 Text(0.703125, 0.08333333333333333, 'entropy = 0.995\nsamples = 1467\nvalue
= [674, 793]'),
 Text(0.734375, 0.08333333333333333, 'entropy = 0.944\nsamples = 119\nvalue =
[43, 76]'),
 Text(0.875, 0.5833333333333334, 'feature_143 <= 0.5\nentropy = 0.822\nsample
s = 1511\nvalue = [388, 1123]'),
 Text(0.8125, 0.4166666666666667, 'feature_169 <= 11.758\nentropy = 0.873\nsa
mples = 716\nvalue = [210, 506]'),
 Text(0.78125, 0.25, 'feature_81 <= 44.5\nentropy = 0.824\nsamples = 472\nval
ue = [122, 350]'),
```

In [ ]:
```
##########################################################################################
########################################## BALANCEO DE CLASES ##########################
##########################################################################################
```

In [31]:
```
######################################### OVERSAMPLING Y UNDERSAMPLING ##########
target_column = 'target'
drop_columns = ['id', target_column]
num_columns = [col for col in df_train.select_dtypes(['int64', 'float64']).columr

x = df_train[num_columns].fillna(0)
y = df_train['target'].values


# Particionamos nuestros datos para entrenar el modelo
#x_train_c, x_valid, y_train_c, y_valid = train_test_split(x, y, test_size = 0.2,

# Particionamos el train para tener train y test
#x_train, x_test, y_train, y_test = train_test_split(x_train_c, y_train_c, test_s

X_train = x_train[ls_best]
X_test = x_test[ls_best]
X_valid=x_valid[ls_best]

# Vemos cómo están distribuidas las clases en el conjunto test
display(pd.DataFrame(y_test).value_counts())
#display(pd.DataFrame(y_test).value_counts(1))
fig, ax = plt.subplots(figsize=(5,5))
sns.countplot(y_test)
plt.title('Y_TEST')
plt.show()
```

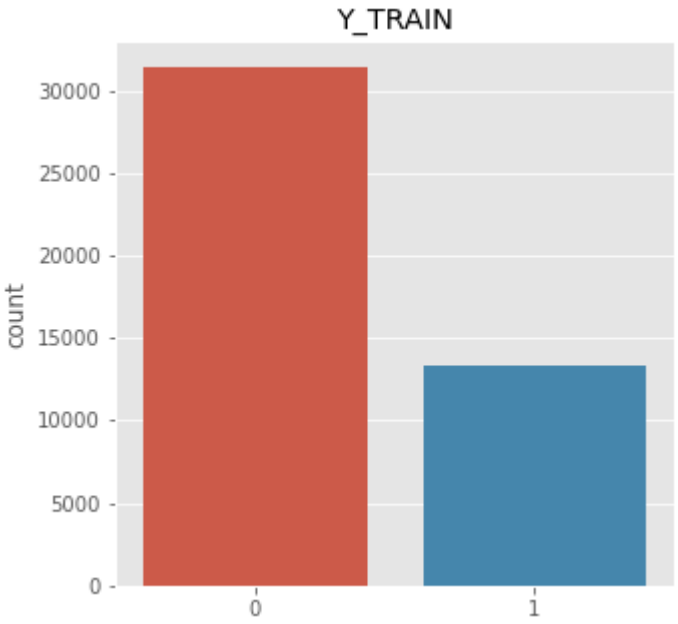
```
0    7769
1    3431
dtype: int64

C:\Users\mime1001\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning:

Pass the following variable as a keyword arg: x. From version 0.12, the only va
lid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
```


Y_TEST
8000
7000
6000
5000
4000
3000
2000
1000
0
count
0
1

In [32]:
```
# Vemos cómo están distribuidas las clases en el conjunto train
display(pd.DataFrame(y_train).value_counts())

fig, ax = plt.subplots(figsize=(5,5))
sns.countplot(y_train)
plt.title('Y_TRAIN')
plt.show()
```

```
0    31419
1    13380
dtype: int64
```

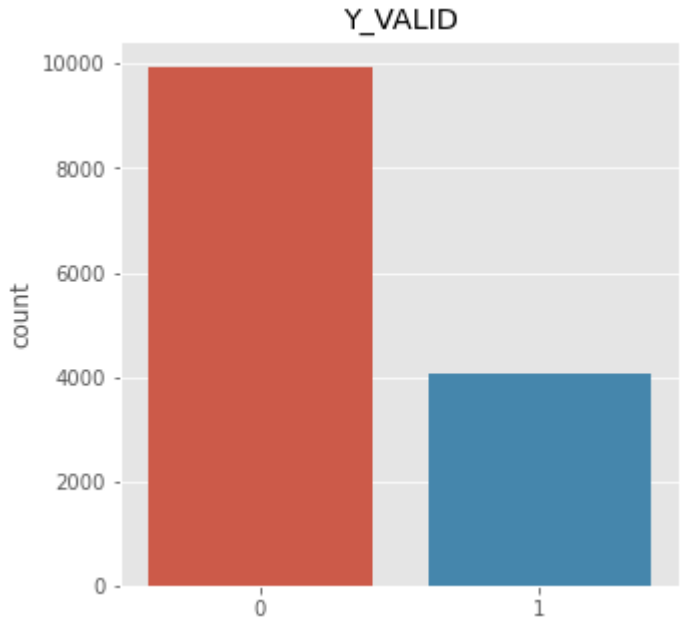
```
C:\Users\mime1001\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning:

Pass the following variable as a keyword arg: x. From version 0.12, the only va
lid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
```



In [33]:
```
# Vemos cómo están distribuidas las clases en el conjunto valid
display(pd.DataFrame(y_valid).value_counts())

fig, ax = plt.subplots(figsize=(5,5))
sns.countplot(y_valid)
plt.title('Y_VALID')
plt.show()
```

```
0    9939
1    4061
dtype: int64
```

```
C:\Users\mime1001\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning:

Pass the following variable as a keyword arg: x. From version 0.12, the only va
lid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
```

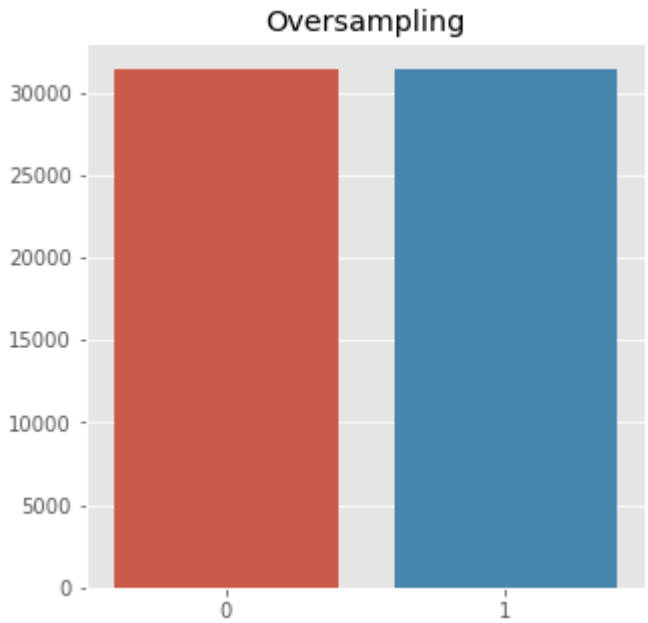


In [34]:
```
########### Hacemos el oversampling
#pip install imblearn
from imblearn.over_sampling import RandomOverSampler
ran=RandomOverSampler()


# Sólo lo estamos haciendo para la parte train
X_ran_over, y_ran_over= ran.fit_resample(X_train,y_train)


x=pd.DataFrame(y_ran_over).value_counts().values
display(pd.DataFrame(y_ran_over).value_counts())
fig, ax = plt.subplots(figsize=(5,5))
sns.barplot([0,1],x)
plt.title('Oversampling')
```

```
0    31419
1    31419
dtype: int64
```

```
C:\Users\mime1001\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning:

Pass the following variables as keyword args: x, y. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without a
n explicit keyword will result in an error or misinterpretation.
```

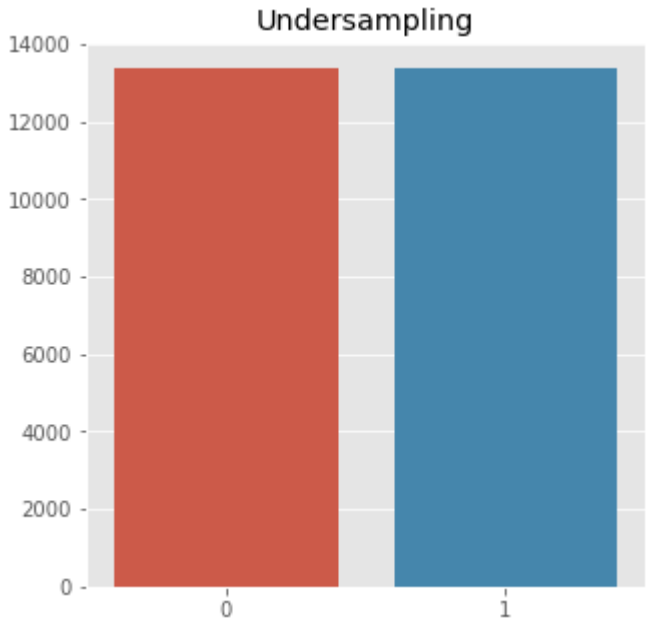
Out[34]:
```
Text(0.5, 1.0, 'Oversampling')
```



In [35]:
```
########### Hacemos undersampling
from imblearn.under_sampling import RandomUnderSampler
ron=RandomUnderSampler()
X_ran_under, y_ran_under= ron.fit_resample(X_train,y_train)
x=pd.DataFrame(y_ran_under).value_counts().values
display(pd.DataFrame(y_ran_under).value_counts())
fig, ax = plt.subplots(figsize=(5,5))
sns.barplot([0,1],x)
plt.title('Undersampling')
```

```
0    13380
1    13380
dtype: int64

C:\Users\mime1001\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning:

Pass the following variables as keyword args: x, y. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without a
n explicit keyword will result in an error or misinterpretation.
```

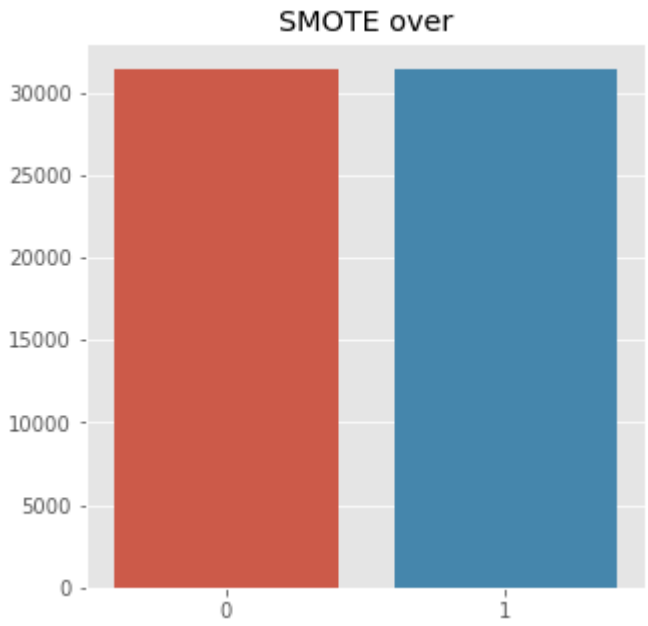
Out[35]: Text(0.5, 1.0, 'Undersampling')



In [36]:
```
############## Hacemos SMOTE
from imblearn.over_sampling import SMOTE
smote = SMOTE(sampling_strategy='minority')
X_sm, y_sm = smote.fit_resample(X_train, y_train)
x=pd.DataFrame(y_sm).value_counts().values
display(pd.DataFrame(y_sm).value_counts())
fig, ax = plt.subplots(figsize=(5,5))
sns.barplot([0,1],x)
plt.title('SMOTE over')
```

```
0    31419
1    31419
dtype: int64

C:\Users\mime1001\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning:

Pass the following variables as keyword args: x, y. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without a
n explicit keyword will result in an error or misinterpretation.
```

Out[36]:
```
Text(0.5, 1.0, 'SMOTE over')
```



In [48]:
```
# Nos traemos nuetsros modelos
best_tree=pd.read_pickle('Arbol_75.pkl')
reglin=pd.read_pickle('Regresion_Log_75.pkl')

from sklearn.metrics import f1_score
def evaluate_sampling(model,X_ran,y_ran,X_test,y_test):
    model.fit(X_ran,y_ran)
    print('ROC:')
    print(round(roc_auc_score(y_test,model.predict_proba(X_test)[:,1]),4))
    print('F1-score clase 1:')
    print(round(f1_score(y_test,model.predict(X_test)),4))
    print(confusion_matrix(y_test,model.predict(X_test)))
```

In [49]:
```
############################## Regresion lineal ###################################
print('Normal:')
display(evaluate_sampling(reglin,X_train,y_train, X_valid,y_valid))

print('Oversampling:')
display(evaluate_sampling(reglin,X_ran_over,y_ran_over, X_valid,y_valid))

print('Undersampling:')
display(evaluate_sampling(reglin,X_ran_under,y_ran_under, X_valid,y_valid))

print('Smote over:')
display(evaluate_sampling(reglin,X_sm,y_sm, X_valid,y_valid))
```

```
Normal:
ROC:
0.7584
F1-score clase 1:
0.4824
[[9011  928]
 [2475 1586]]

None

Oversampling:
ROC:
0.7587
F1-score clase 1:
0.5811
[[7513 2426]
 [1404 2657]]

None

Undersampling:
ROC:
0.7586
F1-score clase 1:
0.5804
[[7513 2426]
 [1409 2652]]

None

Smote over:
ROC:
0.7472
F1-score clase 1:
0.5756
[[7403 2536]
 [1395 2666]]

None
```

In [50]:
```
############################## Árbol ######################################
print('Normal:')
display(evaluate_sampling(best_tree,X_train,y_train, X_valid,y_valid))

print('Oversampling:')
display(evaluate_sampling(best_tree,X_ran_over,y_ran_over, X_valid,y_valid))

print('Undersampling:')
display(evaluate_sampling(best_tree,X_ran_under,y_ran_under,X_valid,y_valid))

print('Smote over:')
display(evaluate_sampling(best_tree,X_sm,y_sm, X_valid,y_valid))
```

```
Normal:
ROC:
0.7214
F1-score clase 1:
0.4859
[[8821 1118]
 [2399 1662]]

None

Oversampling:
ROC:
0.7366
F1-score clase 1:
0.5708
[[7763 2176]
 [1570 2491]]

None

Undersampling:
ROC:
0.6739
F1-score clase 1:
0.4902
[[6812 3127]
 [1727 2334]]

None

Smote over:
ROC:
0.6695
F1-score clase 1:
0.4796
[[7863 2076]
 [2125 1936]]

None
```

In [ ]:

In [ ]:
```
#================================================ FIN ========================
```

In [ ]:
```

```

======= Modelación No Supervizada ======

Melissa Sofía Miranda Peñafiel

melissamiranda@ciencias.unam.mx (mailto:melissamiranda@ciencias.unam.mx)

En este notebook vamos a desarrollar modelos no supervizados; diferentes métodos de clusterización

>>>>>>> Arquitectura de Datos <<<<<<<<<

---------- Modelación No Supervizada ----------

In [38]:
```
# ========================================================================
# ===================================== C H E C K P O I N T=====================
# ========================================================================
```

In [39]:
```
train = pd.read_csv (os.getcwd() + '/Data/train_data_imp_1.csv')
train
```

Out[39]:

	target	feature_0	feature_1	feature_2	feature_3	feature_4	feature_14	feature_15	feature_
0	0	C0	C0	C1	C5	C11	0.000	0.000000	
1	0	C0	C0	C3	C5	C1	0.000	0.000000	
2	0	C0	C0	C3	C5	C2	0.000	0.000000	
3	0	C0	C0	C1	C5	C1	0.000	0.000000	
4	1	C0	C0	C3	C3	C11	0.000	0.258000	
...	...	...	...	...	...	...	...	...	...
69994	0	C0	C0	C3	C1	C11	0.000	0.000000	
69995	0	C0	C0	C5	C5	C2	6.928	13.721667	
69996	0	C0	C0	C3	C2	C11	0.000	0.000000	
69997	0	C0	C0	C1	C5	C1	0.000	0.000000	
69998	1	C0	C0	C2	C5	C2	0.000	0.000000	

69999 rows × 403 columns

In [40]:
```
def missing_zero_values_table(df):
        zero_val = (df == 0.00).astype(int).sum(axis=0)
        mis_val = df.isnull().sum()
        mis_val_percent = 100 * df.isnull().sum() / len(df)
        mz_table = pd.concat([zero_val, mis_val, mis_val_percent], axis=1)
        mz_table = mz_table.rename(
        columns = {0 : 'Zero Values', 1 : 'Missing Values', 2 : '% of Total Value
        mz_table['Total Zero Missing Values'] = mz_table['Zero Values'] + mz_tabl
        mz_table['% Total Zero Missing Values'] = 100 * mz_table['Total Zero Miss
        mz_table['Data Type'] = df.dtypes
        mz_table = mz_table[
            mz_table.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)
        print ("Your selected dataframe has " + str(df.shape[1]) + " columns and
            "There are " + str(mz_table.shape[0]) +
              " columns that have missing values.")
        return mz_table



missing_zero_values_table(train)
```

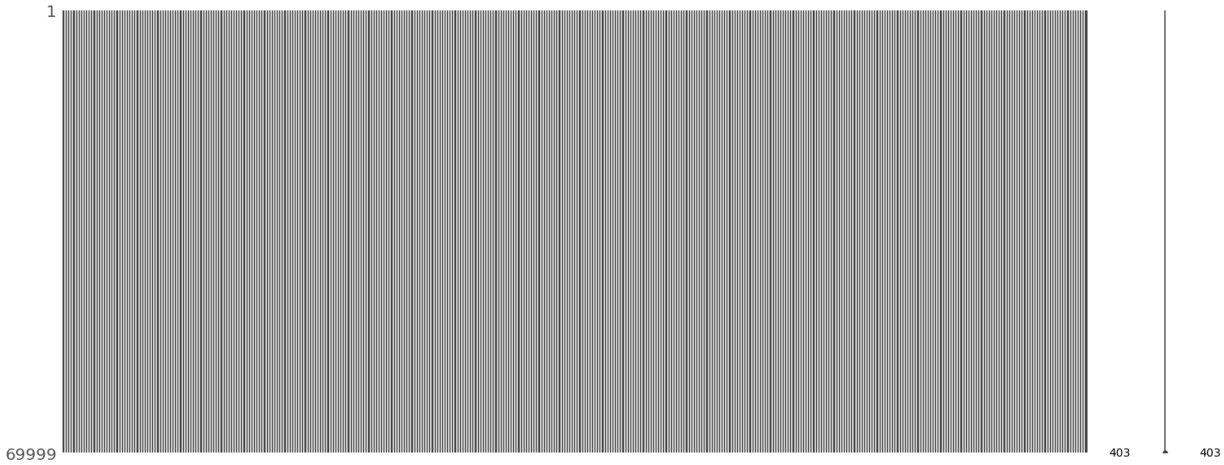
```
Your selected dataframe has 403 columns and 69999 Rows.
There are 0 columns that have missing values.
```

Out[40]:

	Zero Values	Missing Values	% of Total Values	Total Zero Missing Values	% Total Zero Missing Values	Data Type

In [41]:
```
msno.matrix(train)
```

Out[41]: <AxesSubplot:>



In [42]:
```
#* VARS CAT
# 0,1,2,3,4,16,23,24,27,28,29,30,31,32,33,34,41,42,58,97,134,135,152,163-165,186,
# 250,251,270-278,281-286,289-293,336-338,371

#* VARS CAT COD
#43,44,45,46,47,48,49,50,51,52,53,62,64,67,68,69,70,71,72,73,77,78,79,80,87,98,99
#106,107,108,109,110,111,112,113,114,115,116,117,118,119,120-127,136-148,150,151,
#294-311,314-335,339,341,355,359,363,367,369,372-374,376,378,380,382,384,386,388,
#400,402,404,406,408,410,414,416,418,420,422,424,426,428,430,434,436,438,440,442,
#456,458,460,462,464,466,468,470,471=473=475=477,479,481,483,485,486,488,490,492,
#503,504,506
```

In [43]:
```
# ================================================================================
# ======================================= M O D E L A D O ========================
# ================================================================================

#trabajamos con DF=train
train
```

Out[43]:

feature_494	feature_496	feature_497	feature_499	feature_501	feature_503	feature_504	feature_506
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
99	99	-99	-99	99	99	-99	99
0	0	0	0	0	0	0	0

In [44]:
```
##################### CORRELACIÓN ENTRE LAS VARIABLES NUMÉRICAS Y TARGET ######
target_column = 'target'
drop_columns = ['id', target_column]

num_columns = [col for col in train.select_dtypes(['int64', 'float64']).columns i

corr_df = train[['target'] + num_columns].corr(method='pearson')
corr_df
```

Out[44]:

	target	feature_14	feature_15	feature_25	feature_26	feature_35	feature_36	featu
target	1.000000	0.015662	0.017865	-0.046023	-0.049170	0.006516	0.005990	-0.0
feature_14	0.015662	1.000000	0.737889	0.083695	0.087566	-0.000460	-0.000614	-0.0
feature_15	0.017865	0.737889	1.000000	0.090409	0.106344	-0.000633	-0.000776	-0.0
feature_25	-0.046023	0.083695	0.090409	1.000000	0.908288	-0.001029	-0.001952	-0.0
feature_26	-0.049170	0.087566	0.106344	0.908288	1.000000	-0.001339	-0.001986	-0.0
...	...	...	...	...	...	...	...	
feature_499	-0.071182	-0.008594	-0.007689	0.017465	0.051215	-0.000403	0.001195	0.0
feature_501	0.071182	0.008594	0.007689	-0.017465	-0.051215	0.000403	-0.001195	-0.0
feature_503	0.071182	0.008594	0.007689	-0.017465	-0.051215	0.000403	-0.001195	-0.0
feature_504	-0.071182	-0.008594	-0.007689	0.017465	0.051215	-0.000403	0.001195	0.0
feature_506	0.071182	0.008594	0.007689	-0.017465	-0.051215	0.000403	-0.001195	-0.0

341 rows × 341 columns

In [45]:
```
a=corr_df['target'].abs().sort_values(ascending=False).to_frame()
aa= a.index.tolist()
aa
a.T
```

Out[45]:

	target	feature_307	feature_297	feature_81	feature_108	feature_110	feature_82	feature_47
target	1.0	0.335277	0.166039	0.156985	0.122208	0.121303	0.113788	0.10976

In [46]:
```
#sacamos las primeras 30
#a=corr_df['target'].abs().sort_values(ascending=False).to_frame()
#aa= a.index.tolist()
#aa

# top 31
top_31=['target','feature_307','feature_297','feature_81','feature_108','feature_
   'feature_264','feature_254','feature_263','feature_143','feature_298','feature
   'feature_48','feature_224','feature_354','feature_223','feature_353','feature_
    'feature_302','feature_402','feature_440','feature_369','feature_26','feature

#['feature_7', 'feature_9', 'feature_5', 'feature_11', 'feature_12', 'feature_6',

data_corr= train[top_31].corr(method='pearson')

##data_corr
```

In [47]:
```
# Obtenemos la matriz de correlación con mapa de calor
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(20, 20))

sns.heatmap(
    data_corr,
    annot     = True,
    cbar      = False,
    annot_kws = {"size": 10},
    vmin      = -1,
    vmax      = 1,
    center    = 0,
    cmap      = sns.diverging_palette(20, 220, n=200),
    square    = True,
    ax        = ax
)

ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation = 45,
    horizontalalignment = 'right',
)

ax.tick_params(labelsize = 10)
```

In [48]:
```
#train30=train[top_31].drop(columns=['target'])
#train31=train[top_31]
```

Vamos a seleccionar variables con un K-best

In [51]:
```
import matplotlib.pyplot as plt
import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
from sklearn.feature_selection import SelectKBest, f_classif, f_regression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler
import cufflinks as cf
cf.go_offline()



# Aunque ya vimos las variables mas correlacionadas veremos que tanto contrasta
# con un K-best
kb = SelectKBest(k = 15, score_func=f_classif)
```

In [52]:
```
x=train[num_columns]
x
```

Out[52]:

	feature_14	feature_15	feature_25	feature_26	feature_35	feature_36	feature_37	featu
0	0.000	0.000000	0	0	-17.527805	-77.832935	0.003995	15989.26
1	0.000	0.000000	0	0	0.000000	0.000000	-18.899991	5.16
2	0.000	0.000000	0	0	0.000000	0.000000	7.167159	-33.03
3	0.000	0.000000	0	0	0.000000	0.000000	-4.065911	126.55
4	0.000	0.258000	4716	26992	0.000000	0.000000	32.781988	-29.67
...	...	...	...	...	...	...	...	
69994	0.000	0.000000	1647	12835	17.820060	-3.898450	0.350586	12.06
69995	6.928	13.721667	5491	61194	0.000000	-100.000000	-38.232474	265.82
69996	0.000	0.000000	19173	57103	0.000000	0.000000	-59.850864	-34.25
69997	0.000	0.000000	0	0	-64.459028	100.000000	158.045433	100.00
69998	0.000	0.000000	0	0	0.000000	0.000000	0.000000	0.00

69999 rows × 340 columns

In [53]:
```
y=train['target']
y
```

Out[53]:
```
0        0
1        0
2        0
3        0
4        1
        ..
69994    0
69995    0
69996    0
69997    0
69998    1
Name: target, Length: 69999, dtype: int64
```

In [54]:
```
kb.fit(x, y)
```

Out[54]:
```
SelectKBest

SelectKBest(k=15)
```

In [55]:
```
ls_best = [x for x, y in zip(x.columns, kb.get_support()) if y]
ls_best
```

Out[55]:
```
['feature_43',
 'feature_81',
 'feature_82',
 'feature_108',
 'feature_110',
 'feature_143',
 'feature_169',
 'feature_254',
 'feature_263',
 'feature_264',
 'feature_297',
 'feature_298',
 'feature_307',
 'feature_328',
 'feature_470']
```

In [ ]:
```
top_31=['target','feature_307','feature_297','feature_81','feature_108','feature_
   'feature_264','feature_254','feature_263','feature_143','feature_298','feature
   'feature_48','feature_224','feature_354','feature_223','feature_353','feature_
    'feature_302','feature_402','feature_440','feature_369','feature_26','feature
```

Nos quedamos con las 15 variables del K-BEST que coinciden con el top 15 de correlación + variable más importante para el árbol de decición del modulo pasado

In [59]:
```
top_16=['target','feature_307','feature_297','feature_81','feature_108','feature_
        'feature_82','feature_470','feature_264','feature_254','feature_263',
        'feature_143','feature_298','feature_43','feature_169','feature_328',
        'feature_26']
```

In [62]:
```
train_16=train[top_16]
display(train_16.info())
train_16
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69999 entries, 0 to 69998
Data columns (total 17 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   target       69999 non-null  int64  
 1   feature_307  69999 non-null  int64  
 2   feature_297  69999 non-null  int64  
 3   feature_81   69999 non-null  float64
 4   feature_108  69999 non-null  int64  
 5   feature_110  69999 non-null  int64  
 6   feature_82   69999 non-null  int64  
 7   feature_470  69999 non-null  int64  
 8   feature_264  69999 non-null  int64  
 9   feature_254  69999 non-null  float64
 10  feature_263  69999 non-null  int64  
 11  feature_143  69999 non-null  int64  
 12  feature_298  69999 non-null  int64  
 13  feature_43   69999 non-null  int64  
 14  feature_169  69999 non-null  float64
 15  feature_328  69999 non-null  int64  
 16  feature_26   69999 non-null  int64  
dtypes: float64(3), int64(14)
memory usage: 9.1 MB

None
```

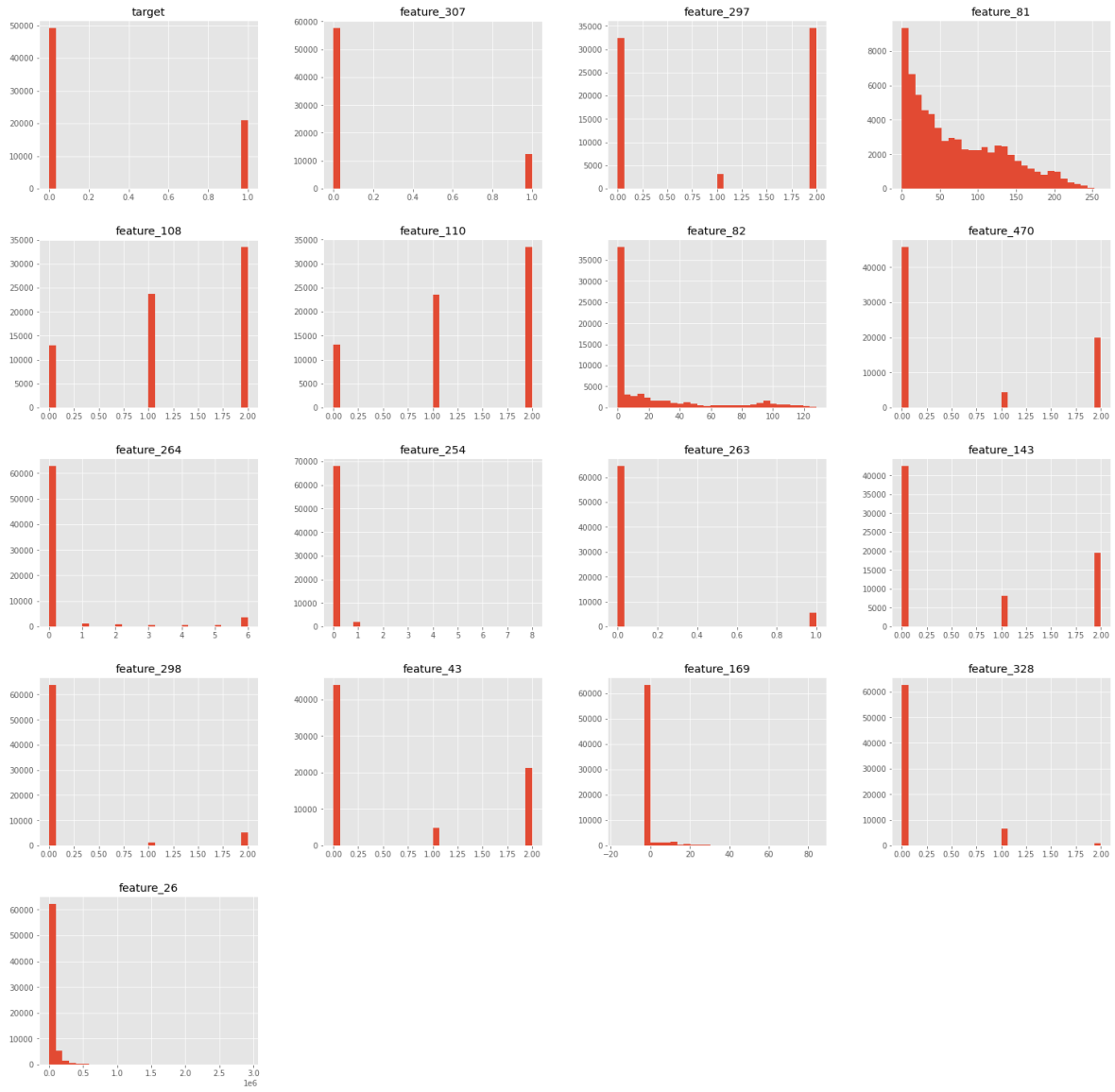
Out[62]:

	target	feature_307	feature_297	feature_81	feature_108	feature_110	feature_82	featur
0	0	0	2	4.0	0	0	0	
1	0	1	2	34.0	2	2	0	
2	0	0	2	36.0	2	2	0	
3	0	1	2	48.0	1	1	0	
4	1	1	2	102.0	1	1	11	
...	...	...	...	...	...	...	...	
69994	0	0	2	88.0	1	1	48	
69995	0	0	0	16.0	2	2	15	
69996	0	1	0	149.0	2	2	16	
69997	0	0	0	2.0	0	0	0	
69998	1	0	2	0.0	0	0	0	

69999 rows × 17 columns

In [61]:
```
plt.rcParams["figure.figsize"] = [25, 25]
train_16[top_16].hist(bins=30)
```

Out[61]:
```
array([[<AxesSubplot:title={'center':'target'}>,
        <AxesSubplot:title={'center':'feature_307'}>,
        <AxesSubplot:title={'center':'feature_297'}>,
        <AxesSubplot:title={'center':'feature_81'}>],
       [<AxesSubplot:title={'center':'feature_108'}>,
        <AxesSubplot:title={'center':'feature_110'}>,
        <AxesSubplot:title={'center':'feature_82'}>,
        <AxesSubplot:title={'center':'feature_470'}>],
       [<AxesSubplot:title={'center':'feature_264'}>,
        <AxesSubplot:title={'center':'feature_254'}>,
        <AxesSubplot:title={'center':'feature_263'}>,
        <AxesSubplot:title={'center':'feature_143'}>],
       [<AxesSubplot:title={'center':'feature_298'}>,
        <AxesSubplot:title={'center':'feature_43'}>,
        <AxesSubplot:title={'center':'feature_169'}>,
        <AxesSubplot:title={'center':'feature_328'}>],
       [<AxesSubplot:title={'center':'feature_26'}>, <AxesSubplot:>,
        <AxesSubplot:>, <AxesSubplot:>]], dtype=object)
```



In [63]:
```
for col in train_16.columns:
    display(train_16[col].value_counts().reset_index())
```

2	1	3064

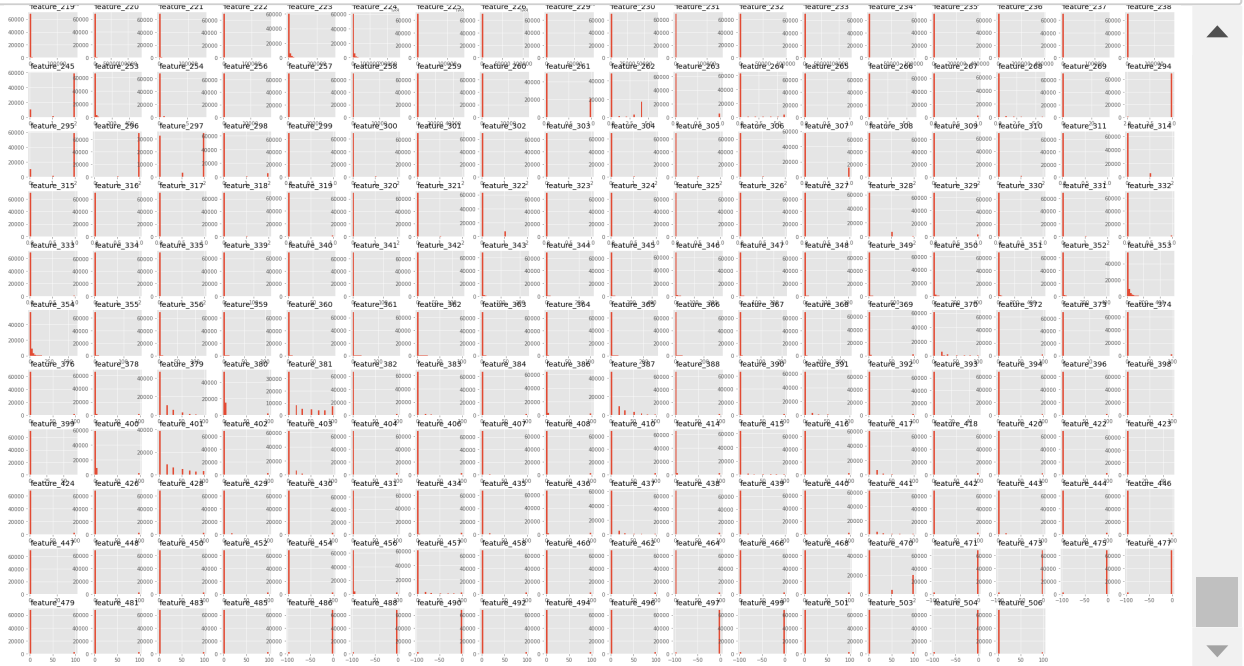
	index	feature_81
0	1.0	1338
1	5.0	1269
2	2.0	1145
3	4.0	1139
4	3.0	1104
...	...	...
254	252.0	2
255	259.0	2
256	255.0	2
257	247.0	1

In [64]:
```
# 81,82,169,26
#66,128,132,156,207
train_num=train[num_columns]
train_num
```

Out[64]:

	feature_14	feature_15	feature_25	feature_26	feature_35	feature_36	feature_37	f
0	0.000	0.000000	0	0	-17.527805	-77.832935	0.003995	159
1	0.000	0.000000	0	0	0.000000	0.000000	-18.899991	
2	0.000	0.000000	0	0	0.000000	0.000000	7.167159	-3
3	0.000	0.000000	0	0	0.000000	0.000000	-4.065911	1
4	0.000	0.258000	4716	26992	0.000000	0.000000	32.781988	-3
...	...	...	...	...	...	...	...	
69994	0.000	0.000000	1647	12835	17.820060	-3.898450	0.350586	
69995	6.928	13.721667	5491	61194	0.000000	-100.000000	-38.232474	2
69996	0.000	0.000000	19173	57103	0.000000	0.000000	-59.850864	-3
69997	0.000	0.000000	0	0	-64.459028	100.000000	158.045433	1

In [67]:
```
plt.rcParams["figure.figsize"] = [40, 40]
train[num_columns].hist(bins=30)
```



Nos quedamos con las 16 vars mencionadas + 5 variables continuas

In [70]:
```
top_21=['target','feature_307','feature_297','feature_81','feature_108','feature_
        'feature_82','feature_470','feature_264','feature_254','feature_263',
        'feature_143','feature_298','feature_43','feature_169','feature_328',
        'feature_26','feature_66','feature_128','feature_132','feature_156','feat

#66,128,132,156,207

train21=train[top_21]
train21
```

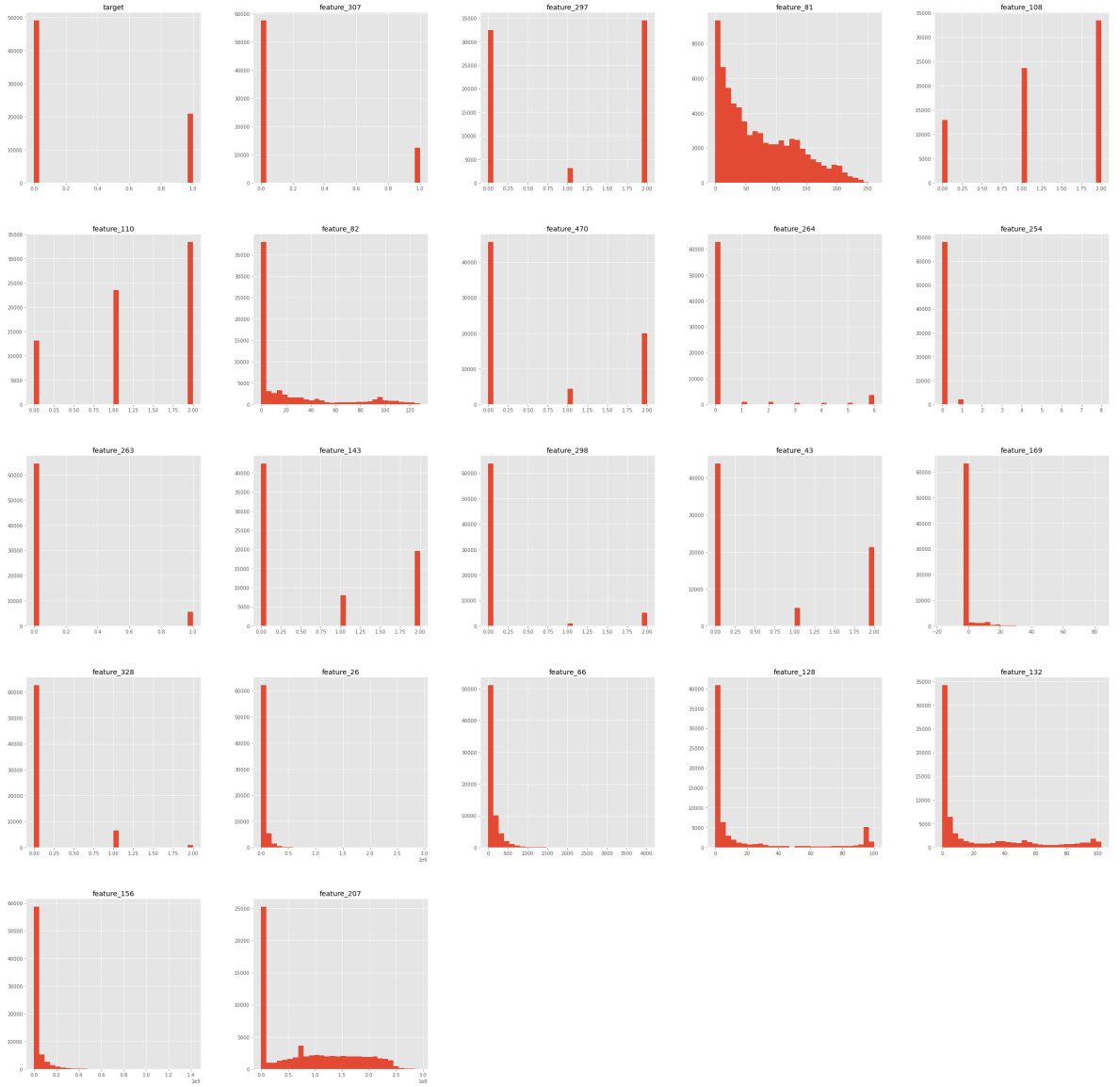
Out[70]:

	target	feature_307	feature_297	feature_81	feature_108	feature_110	feature_82	feature_47
0	0	0	2	4.0	0	0	0	
1	0	1	2	34.0	2	2	0	
2	0	0	2	36.0	2	2	0	
3	0	1	2	48.0	1	1	0	
4	1	1	2	102.0	1	1	11	
...	...	...	...	...	...	...	...	.
69994	0	0	2	88.0	1	1	48	
69995	0	0	0	16.0	2	2	15	
69996	0	1	0	149.0	2	2	16	
69997	0	0	0	2.0	0	0	0	
69998	1	0	2	0.0	0	0	0	

69999 rows × 22 columns

In [71]:
```
plt.rcParams["figure.figsize"] = [40, 40]
train[top_21].hist(bins=30)
```

Out[71]:
```
array([[<AxesSubplot:title={'center':'target'}>,
        <AxesSubplot:title={'center':'feature_307'}>,
        <AxesSubplot:title={'center':'feature_297'}>,
        <AxesSubplot:title={'center':'feature_81'}>,
        <AxesSubplot:title={'center':'feature_108'}>],
       [<AxesSubplot:title={'center':'feature_110'}>,
        <AxesSubplot:title={'center':'feature_82'}>,
        <AxesSubplot:title={'center':'feature_470'}>,
        <AxesSubplot:title={'center':'feature_264'}>,
        <AxesSubplot:title={'center':'feature_254'}>],
       [<AxesSubplot:title={'center':'feature_263'}>,
        <AxesSubplot:title={'center':'feature_143'}>,
        <AxesSubplot:title={'center':'feature_298'}>,
        <AxesSubplot:title={'center':'feature_43'}>,
        <AxesSubplot:title={'center':'feature_169'}>],
       [<AxesSubplot:title={'center':'feature_328'}>,
        <AxesSubplot:title={'center':'feature_26'}>,
        <AxesSubplot:title={'center':'feature_66'}>,
        <AxesSubplot:title={'center':'feature_128'}>,
        <AxesSubplot:title={'center':'feature_132'}>],
       [<AxesSubplot:title={'center':'feature_156'}>,
        <AxesSubplot:title={'center':'feature_207'}>, <AxesSubplot:>,
        <AxesSubplot:>, <AxesSubplot:>]], dtype=object)
```



In [72]:
```
train21.to_csv('Data/train_top21.csv', index = False)
```

In [ ]:
```
# ================================================================================
# ======================================= C H E C K P O I N T=====================
# ================================================================================
```

In [74]:
```
top_21=['target','feature_307','feature_297','feature_81','feature_108','feature_
        'feature_82','feature_470','feature_264','feature_254','feature_263',
        'feature_143','feature_298','feature_43','feature_169','feature_328',
        'feature_26','feature_66','feature_128','feature_132','feature_156','feat

train21 = pd.read_csv (os.getcwd() + '/Data/train_top21.csv')
train21
```

Out[74]:

	target	feature_307	feature_297	feature_81	feature_108	feature_110	feature_82	feature_47
0	0	0	2	4.0	0	0	0	
1	0	1	2	34.0	2	2	0	
2	0	0	2	36.0	2	2	0	
3	0	1	2	48.0	1	1	0	
4	1	1	2	102.0	1	1	11	
...	...	...	...	...	...	...	...	.
69994	0	0	2	88.0	1	1	48	
69995	0	0	0	16.0	2	2	15	
69996	0	1	0	149.0	2	2	16	
69997	0	0	0	2.0	0	0	0	
69998	1	0	2	0.0	0	0	0	

69999 rows × 22 columns

In [75]:
```
train0=train21[train21['target']==0]
train0
```

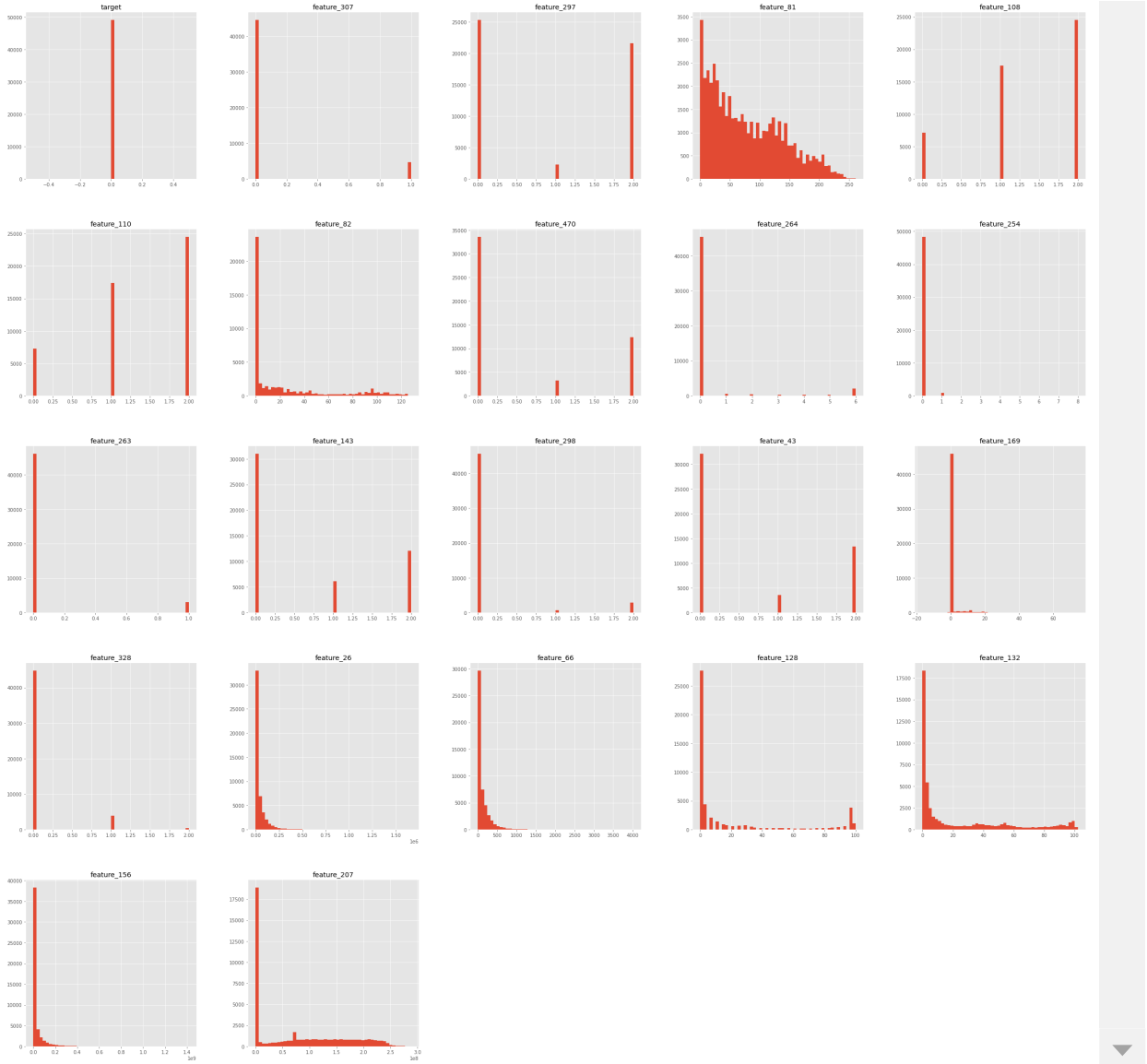
Out[75]:

	target	feature_307	feature_297	feature_81	feature_108	feature_110	feature_82	feature_47
0	0	0	2	4.0	0	0	0	
1	0	1	2	34.0	2	2	0	
2	0	0	2	36.0	2	2	0	
3	0	1	2	48.0	1	1	0	
6	0	0	0	169.0	1	1	123	
...	...	...	...	...	...	...	...	.
69993	0	0	0	144.0	2	2	97	
69994	0	0	2	88.0	1	1	48	
69995	0	0	0	16.0	2	2	15	
69996	0	1	0	149.0	2	2	16	
69997	0	0	0	2.0	0	0	0	

49127 rows × 22 columns

In [77]:
```
plt.rcParams["figure.figsize"] = [40, 40]
train0[top_21].hist(bins=50)
```

Out[77]:
```
array([[<AxesSubplot:title={'center':'target'}>,
        <AxesSubplot:title={'center':'feature_307'}>,
        <AxesSubplot:title={'center':'feature_297'}>,
        <AxesSubplot:title={'center':'feature_81'}>,
        <AxesSubplot:title={'center':'feature_108'}>],
       [<AxesSubplot:title={'center':'feature_110'}>,
        <AxesSubplot:title={'center':'feature_82'}>,
        <AxesSubplot:title={'center':'feature_470'}>,
        <AxesSubplot:title={'center':'feature_264'}>,
        <AxesSubplot:title={'center':'feature_254'}>],
       [<AxesSubplot:title={'center':'feature_263'}>,
        <AxesSubplot:title={'center':'feature_143'}>,
        <AxesSubplot:title={'center':'feature_298'}>,
        <AxesSubplot:title={'center':'feature_43'}>,
        <AxesSubplot:title={'center':'feature_169'}>],
       [<AxesSubplot:title={'center':'feature_328'}>,
        <AxesSubplot:title={'center':'feature_26'}>,
        <AxesSubplot:title={'center':'feature_66'}>,
        <AxesSubplot:title={'center':'feature_128'}>,
        <AxesSubplot:title={'center':'feature_132'}>],
       [<AxesSubplot:title={'center':'feature_156'}>,
        <AxesSubplot:title={'center':'feature_207'}>, <AxesSubplot:>,
        <AxesSubplot:>, <AxesSubplot:>]], dtype=object)
```


target
feature_307
feature_297
feature_81
feature_108
feature_110
feature_82
feature_470
feature_264
feature_254
feature_263
feature_143
feature_298
feature_43
feature_169
feature_328
feature_26
feature_66
feature_128
feature_132
feature_156
feature_207

In [80]:
```
for col in train0.columns:
    display(train0[col].value_counts().sort_index().reset_index())
```

	index	target
0	0	49127

	index	feature_307
0	0	44497
1	1	4630

	index	feature_297
0	0	25282
1	1	2288
2	2	21557

In [79]:
```
train0.to_csv('Data/train_data_imp.csv', index = False)
```

In [49]:
```
#pd.to_pickle(best_tree,'Arbol_70.pkl')
```

Vamos a trabajar con los usuarios no suscritos (target=0)

In [1]:
```
import os
import sys
import glob
import math
import random

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pylab as plt
from itertools import cycle

from random import choice, choices
pd.set_option("display.max_columns", 600)

plt.style.use("ggplot")
color_pal = plt.rcParams["axes.prop_cycle"].by_key()["color"]
color_cycle = cycle(plt.rcParams["axes.prop_cycle"].by_key()["color"])

os.getcwd()
```

Out[1]:
```
'C:\\Users\\mime1001\\Documents\\Diplomado\\Proyecto\\Entrega 3'
```

In [2]:
```
# Importamos la base ya limpia
df_train = pd.read_csv (os.getcwd() + '/Data/train_data_imp.csv')
train=df_train
display(df_train.info())
df_train
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49127 entries, 0 to 49126
Data columns (total 22 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   target       49127 non-null  int64  
 1   feature_307  49127 non-null  int64  
 2   feature_297  49127 non-null  int64  
 3   feature_81   49127 non-null  float64
 4   feature_108  49127 non-null  int64  
 5   feature_110  49127 non-null  int64  
 6   feature_82   49127 non-null  int64  
 7   feature_470  49127 non-null  int64  
 8   feature_264  49127 non-null  int64  
 9   feature_254  49127 non-null  float64
 10  feature_263  49127 non-null  int64  
 11  feature_143  49127 non-null  int64  
 12  feature_298  49127 non-null  int64  
 13  feature_43   49127 non-null  int64  
 14  feature_169  49127 non-null  float64
 15  feature_328  49127 non-null  int64  
 16  feature_26   49127 non-null  int64  
 17  feature_66   49127 non-null  int64  
 18  feature_128  49127 non-null  float64
 19  feature_132  49127 non-null  float64
 20  feature_156  49127 non-null  float64
 21  feature_207  49127 non-null  float64
dtypes: float64(7), int64(15)
memory usage: 8.2 MB

None
```

Out[2]:

	target	feature_307	feature_297	feature_81	feature_108	feature_110	feature_82	feature_47
0	0	0	2	4.0	0	0	0	
1	0	1	2	34.0	2	2	0	
2	0	0	2	36.0	2	2	0	
3	0	1	2	48.0	1	1	0	
4	0	0	0	169.0	1	1	123	
...	...	...	...	...	...	...	...	.
49122	0	0	0	144.0	2	2	97	
49123	0	0	2	88.0	1	1	48	
49124	0	0	0	16.0	2	2	15	
49125	0	1	0	149.0	2	2	16	
49126	0	0	0	2.0	0	0	0	

49127 rows × 22 columns

In [3]:
```
###Vamos a ver los missing values

def missing_zero_values_table(df):
        zero_val = (df == 0.00).astype(int).sum(axis=0)
        mis_val = df.isnull().sum()
        mis_val_percent = 100 * df.isnull().sum() / len(df)
        mz_table = pd.concat([zero_val, mis_val, mis_val_percent], axis=1)
        mz_table = mz_table.rename(
        columns = {0 : 'Zero Values', 1 : 'Missing Values', 2 : '% of Total Value
        mz_table['Total Zero Missing Values'] = mz_table['Zero Values'] + mz_tabl
        mz_table['% Total Zero Missing Values'] = 100 * mz_table['Total Zero Miss
        mz_table['Data Type'] = df.dtypes
        mz_table = mz_table[
            mz_table.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)
        print ("Your selected dataframe has " + str(df.shape[1]) + " columns and
            "There are " + str(mz_table.shape[0]) +
              " columns that have missing values.")
        return mz_table



missing_zero_values_table(train)
```

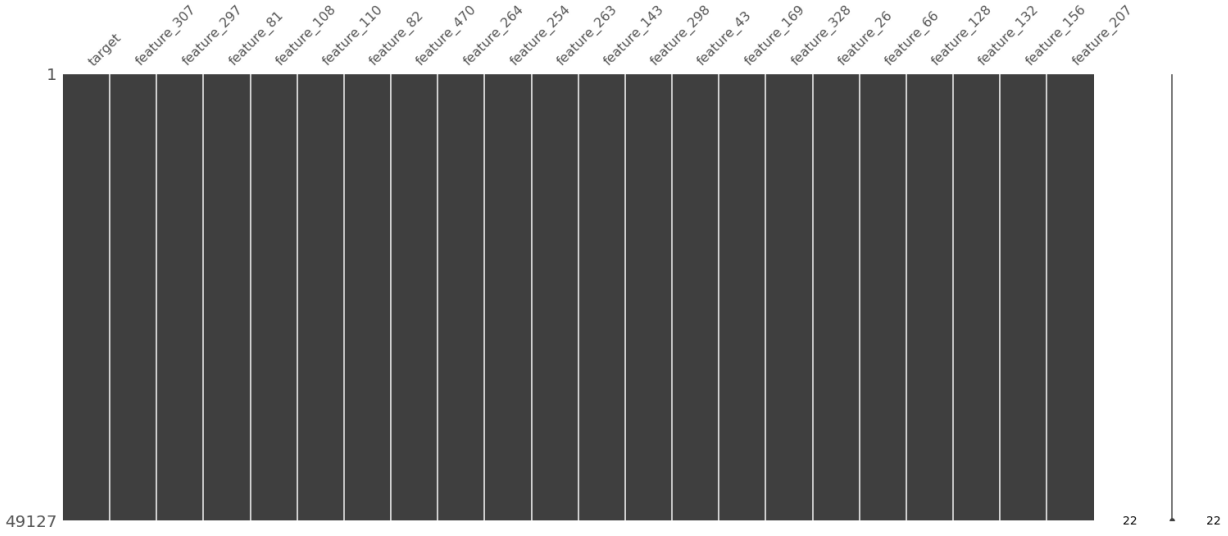
```
Your selected dataframe has 22 columns and 49127 Rows.
There are 0 columns that have missing values.
```

Out[3]:

	Zero Values	Missing Values	% of Total Values	Total Zero Missing Values	% Total Zero Missing Values	Data Type

In [4]:
```
import missingno as msno
msno.matrix(train)
```

Out[4]: <AxesSubplot:>



In [231]:
```
# Vamos a ver las variables
for col in df_train.columns:
    display(df_train[col].value_counts().sort_index().reset_index())
```

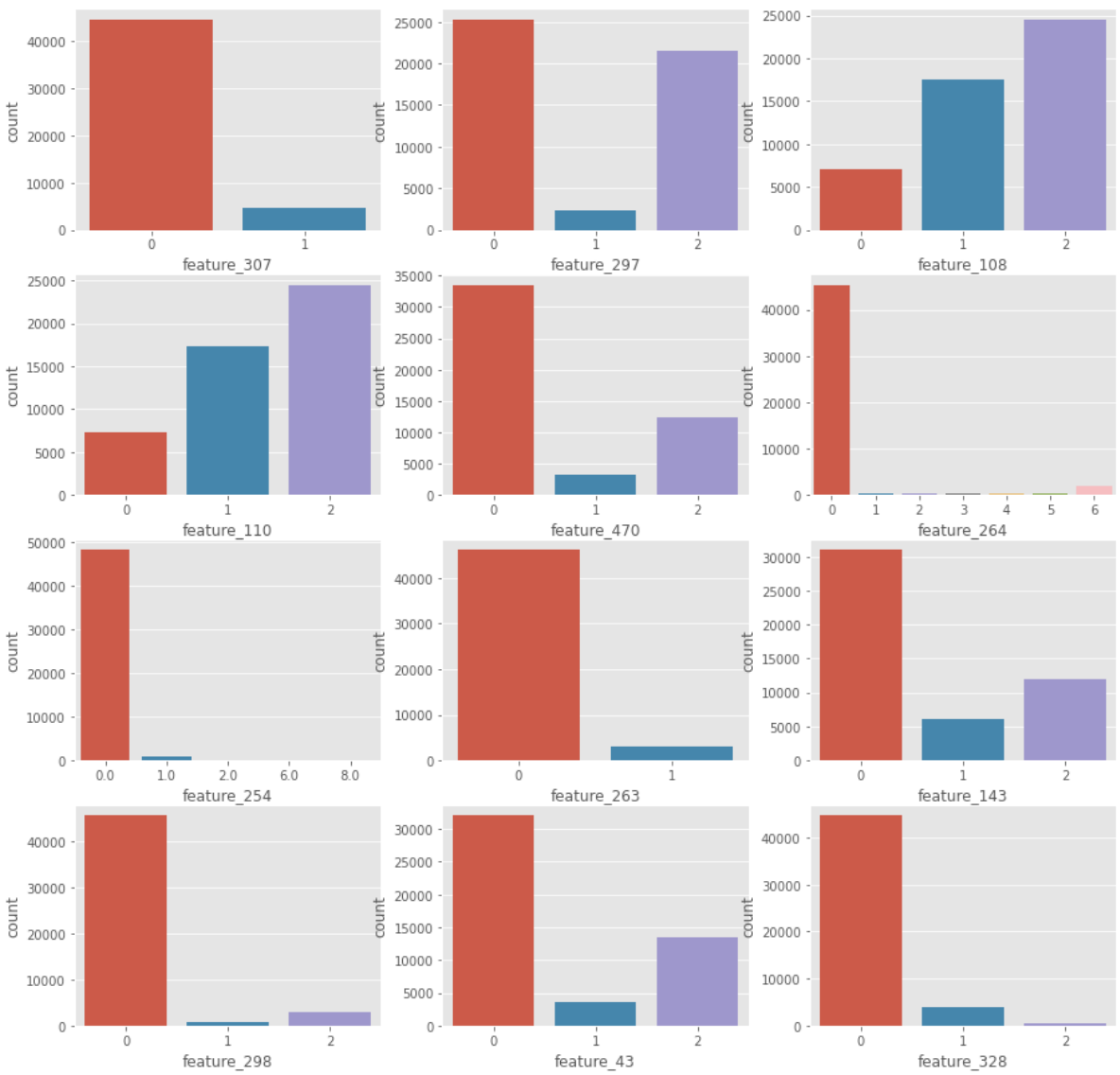
5	5	239
6	6	2077

	index	feature_254
0	0.0	48315
1	1.0	803
2	2.0	7
3	6.0	1
4	8.0	1

In [6]:
```
#cat=307,297,108,110,470,264,254,263,143,298,43,328

#num=81,82,169,26,66,128,132,156,207



num_columns=['feature_81','feature_82','feature_169','feature_26','feature_66',
             'feature_128','feature_132','feature_156','feature_207']

cat_columns=['feature_307','feature_297','feature_108','feature_110','feature_470
             'feature_254','feature_263','feature_143','feature_298','feature_43'
```

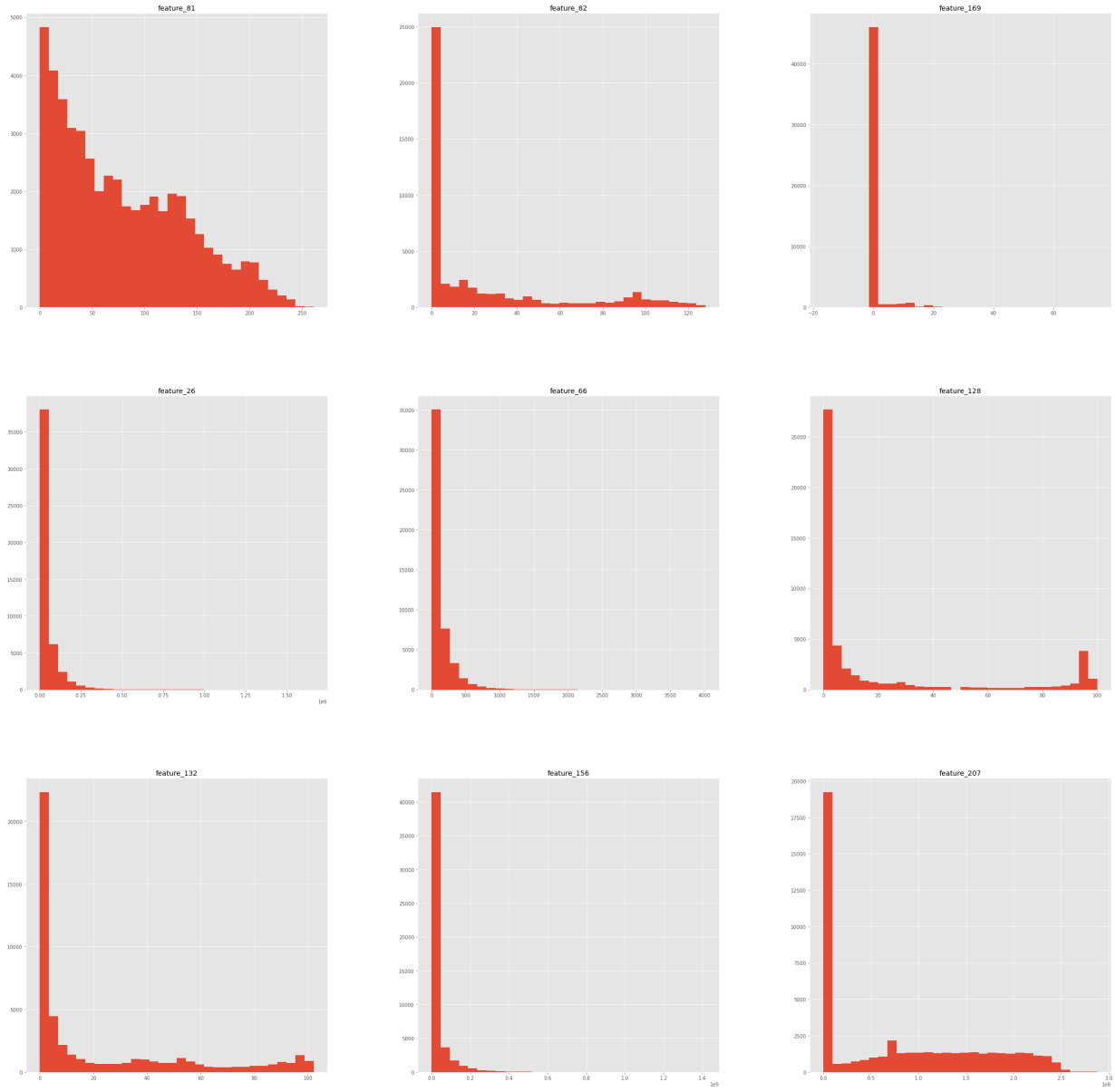
In [7]:
```
# Graficamos todas nuestras variables categóricas para ver si algúna se encuentra
# respecto a sus categorías y nos pueda causar problemas en los futuros modelos.
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

plt.rcParams["figure.figsize"] = [15, 15]
fig, ax =plt.subplots(4,3)

sns.countplot(df_train[cat_columns[0]], ax = ax[0,0])
sns.countplot(df_train[cat_columns[1]], ax = ax[0,1])
sns.countplot(df_train[cat_columns[2]], ax = ax[0,2])
sns.countplot(df_train[cat_columns[3]], ax = ax[1,0])
sns.countplot(df_train[cat_columns[4]], ax = ax[1,1])
sns.countplot(df_train[cat_columns[5]], ax = ax[1,2])
sns.countplot(df_train[cat_columns[6]], ax = ax[2,0])
sns.countplot(df_train[cat_columns[7]], ax = ax[2,1])
sns.countplot(df_train[cat_columns[8]], ax = ax[2,2])
sns.countplot(df_train[cat_columns[9]], ax = ax[3,0])
sns.countplot(df_train[cat_columns[10]], ax = ax[3,1])
sns.countplot(df_train[cat_columns[11]], ax = ax[3,2])
```

Out[7]:
```
<AxesSubplot:xlabel='feature_328', ylabel='count'>
```


count
feature_307
feature_297
feature_108
feature_110
feature_470
feature_264
feature_254
feature_263
feature_143
feature_298
feature_43
feature_328

In [9]:
```
plt.rcParams["figure.figsize"] = [40, 40]
df_train[num_columns].hist(bins=30)
```

Out[9]:
```
array([[<AxesSubplot:title={'center':'feature_81'}>,
        <AxesSubplot:title={'center':'feature_82'}>,
        <AxesSubplot:title={'center':'feature_169'}>],
       [<AxesSubplot:title={'center':'feature_26'}>,
        <AxesSubplot:title={'center':'feature_66'}>,
        <AxesSubplot:title={'center':'feature_128'}>],
       [<AxesSubplot:title={'center':'feature_132'}>,
        <AxesSubplot:title={'center':'feature_156'}>,
        <AxesSubplot:title={'center':'feature_207'}>]], dtype=object)
```



In [232]:
```
#81----5.0--261.0----#episodios en la serie vista
#82----0--128----tiempo en horas que se vio netflix al mes
#169---0.00--74.52-----tiempo en minutos que se jugo un juego en netflix
#26----0--1,662,716-----tiempo que se vio  categoría d edramas
#66----0--4,021--tiempo que se vio contenido exclusivo de netflix
#128---0.00--100---% d euso d elas funciones de netflix (streaming, games, fast l
#132---0.000--102.14--tiempo que se vio algo del top 10 de la region
#156---000--muy grande--Tiempo que se vio la categoría de ciancia ficción
#207---7--muy grande---tiempo que se vio contenido para niños


num_columns=['feature_81','feature_82','feature_169','feature_26','feature_66',
             'feature_128','feature_132','feature_156','feature_207']
```

In [ ]:
```

```

In [ ]:
```
#128

#a=df_train[num_columns[5]].mean()
#display(a)

#df_train[num_columns[5]]=df_train[num_columns[5]].replace(0.00,a)
#df_train[num_columns[5]].value_counts()
```

In [ ]:
```
#plt.rcParams["figure.figsize"] = [5, 5]
#df_train[num_columns[5]].hist(bins=50)
```

========== MODELACIÓN ===========

In [ ]:
```
# ================================================================================
# ======================================= M O D E L A D O ========================
# ================================================================================
```

In [10]:
```
#* VARS CAT
# 0,1,2,3,4,16,23,24,27,28,29,30,31,32,33,34,41,42,58,97,134,135,152,163-165,186,
# 250,251,270-278,281-286,289-293,336-338,371

#* VARS CAT COD
#43,44,45,46,47,48,49,50,51,52,53,62,64,67,68,69,70,71,72,73,77,78,79,80,87,98,99
#106,107,108,109,110,111,112,113,114,115,116,117,118,119,120-127,136-148,150,151,
#294-311,314-335,339,341,355,359,363,367,369,372-374,376,378,380,382,384,386,388,
#400,402,404,406,408,410,414,416,418,420,422,424,426,428,430,434,436,438,440,442,
#456,458,460,462,464,466,468,470,471=473=475=477,479,481,483,485,486,488,490,492,
#503,504,506


### Vamos a sacar las variables numericas
target_column = 'target'
drop_columns = [target_column]

#num_columns = [col for col in train.select_dtypes(['int64', 'float64']).columns


train_num=train[num_columns]
train_num
```

Out[10]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	featu
0	4.0	0	0.0	0	0	0.00	0.805000	1.0244
1	34.0	0	0.0	0	0	3.57	1.150000	0.0000
2	36.0	0	0.0	0	0	0.00	0.000000	0.0000
3	48.0	0	0.0	0	0	0.00	2.723333	0.0000
4	169.0	123	0.0	109	5	3.57	11.041667	7.7846
...	...	...	...	...	...	...	...	
49122	144.0	97	0.0	34394	83	85.71	96.371667	0.0000
49123	88.0	48	0.0	12835	69	96.43	98.391667	9.1345
49124	16.0	15	0.0	61194	229	0.00	0.538333	9.4514
49125	149.0	16	0.0	57103	143	96.43	98.311667	0.0000
49126	2.0	0	0.0	0	0	0.00	0.000000	2.1450

49127 rows × 9 columns

Visualización PCA

In [22]:
```
#Quitamos la variable objetivo de los datos en el paso anterior

# Escalamos los datos
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
data_1 = sc.fit_transform(train_num)
data_1
```

Out[22]:
```
array([[-1.23525473, -0.68298154, -0.21034499, ..., -0.71696474,
         1.29830933, -0.98976747],
       [-0.73648054, -0.68298154, -0.21034499, ..., -0.70594588,
        -0.42508228,  1.15353209],
       [-0.70322893, -0.68298154, -0.21034499, ..., -0.7426754 ,
        -0.42508228,  0.30156247],
       ...,
       [-1.03574506, -0.26087066, -0.21034499, ..., -0.72548173,
        -0.42349236,  1.58182466],
       [ 1.17548718, -0.23272993, -0.21034499, ...,  2.39727316,
        -0.42508228,  0.53246916],
       [-1.26850634, -0.68298154, -0.21034499, ..., -0.7426754 ,
        -0.06424484, -1.00103984]])
```

In [23]:
```
#Nos traemos PCA con 2 comp
from sklearn.decomposition import PCA
pca=PCA(n_components=2)

pca.fit(data_1)
```

Out[23]:
```
PCA(n_components=2)
```

In [24]:
```
pca.explained_variance_ratio_
```

Out[24]:
```
array([0.27878553, 0.18963769])
```

In [25]:
```
pca.explained_variance_ratio_.cumsum()
```

Out[25]:
```
array([0.27878553, 0.46842321])
```

In [26]:
```
data_pca = pd.DataFrame(pca.transform(data_1),columns=['p1','p2'])
data_pca
```

Out[26]:

	p1	p2
0	-1.346520	-0.700573
1	-1.699756	-0.128386
2	-1.592280	-0.242254
3	-1.694227	-0.086172
4	1.109887	-1.151116
...	...	...
49122	3.380914	-1.278264
49123	2.872593	-1.678960
49124	-1.335177	1.264774
49125	2.759872	-0.558766
49126	-1.549766	-0.444905

49127 rows × 2 columns

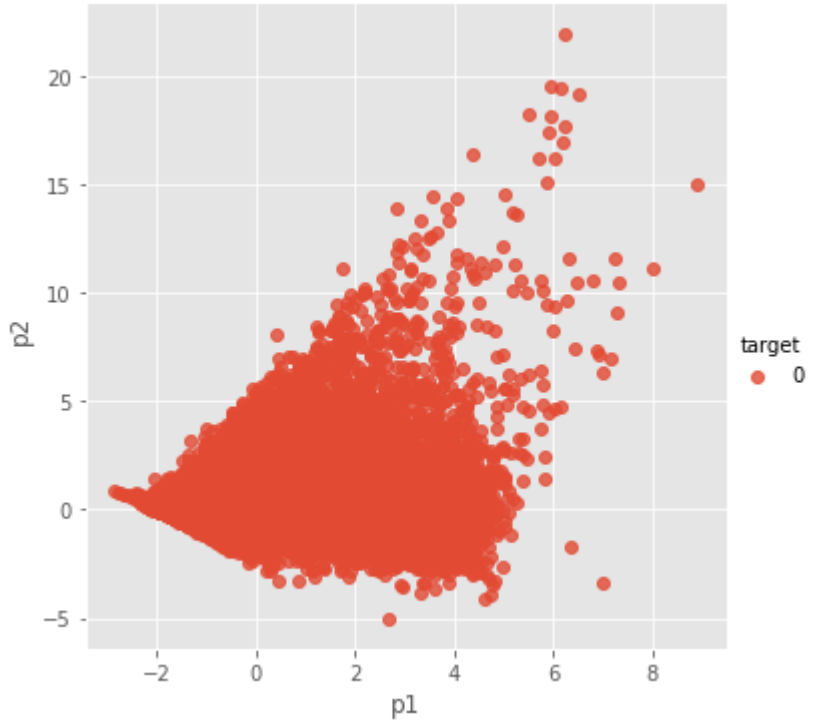
In [27]:
```
data_pca.corr()
```

Out[27]:

	p1	p2
p1	1.000000e+00	-1.045328e-16
p2	-1.045328e-16	1.000000e+00

In [28]:
```
pca_ = data_pca.copy()
pca_['target']=train['target']

import seaborn as sns
sns.lmplot(data=pca_,x='p1',y='p2',fit_reg=False,hue='target')
```

Out[28]:
```
<seaborn.axisgrid.FacetGrid at 0x1c8a9f809d0>
```



MDS

In [30]:
```
# data_1 = datos num escalados con standard scaler
from sklearn.manifold import MDS, TSNE
mds = MDS(n_components=2,n_jobs=-1)
df1 = data_1.copy()
df1=pd.DataFrame(data_1, columns= train_num.columns)
df1['target'] = train['target']

df1
```

Out[30]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	feature
0	-1.235255	-0.682982	-0.210345	-0.524176	-0.609718	-0.566082	-0.716965	1.29
1	-0.736481	-0.682982	-0.210345	-0.524176	-0.609718	-0.458378	-0.705946	-0.42
2	-0.703229	-0.682982	-0.210345	-0.524176	-0.609718	-0.566082	-0.742675	-0.42
3	-0.503719	-0.682982	-0.210345	-0.524176	-0.609718	-0.566082	-0.655696	-0.42
4	1.508003	2.778328	-0.210345	-0.522751	-0.583386	-0.458378	-0.390019	-0.29
...	...	...	...	...	...	...	...	
49122	1.092358	2.046669	-0.210345	-0.074519	-0.172604	2.019712	2.335312	-0.42
49123	0.161313	0.667773	-0.210345	-0.356375	-0.246334	2.343125	2.399828	1.11
49124	-1.035745	-0.260871	-0.210345	0.275856	0.596296	-0.566082	-0.725482	-0.42
49125	1.175487	-0.232730	-0.210345	0.222371	0.143382	2.343125	2.397273	-0.42
49126	-1.268506	-0.682982	-0.210345	-0.524176	-0.609718	-0.566082	-0.742675	-0.06

49127 rows × 10 columns

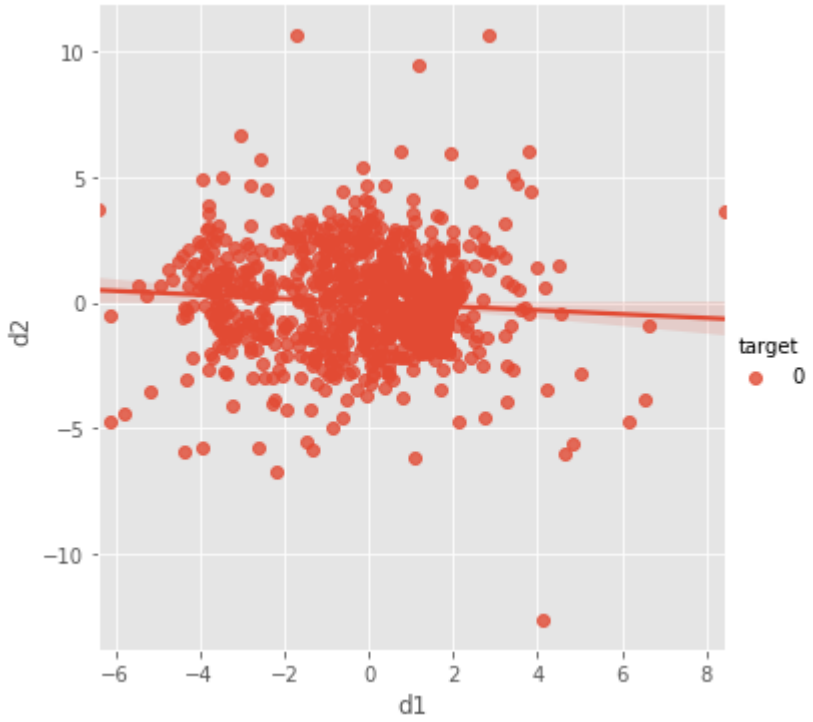
In [31]:
```
sample=df1.sample(frac=.02)
sample['target'].value_counts()
```

Out[31]:
```
0    983
Name: target, dtype: int64
```

In [32]:
```
mds = mds.fit_transform(sample.drop(columns = ['target']))
Xm = pd.DataFrame(mds,columns=['d1','d2'])

Xm['target'] = sample['target'].reset_index().drop(columns = ['index'])
```

In [33]:
```
sns.lmplot(data=Xm,x='d1',y='d2',fit_reg=True,hue='target')
```

Out[33]:
```
<seaborn.axisgrid.FacetGrid at 0x1c8b1d14c40>
```



TSNE

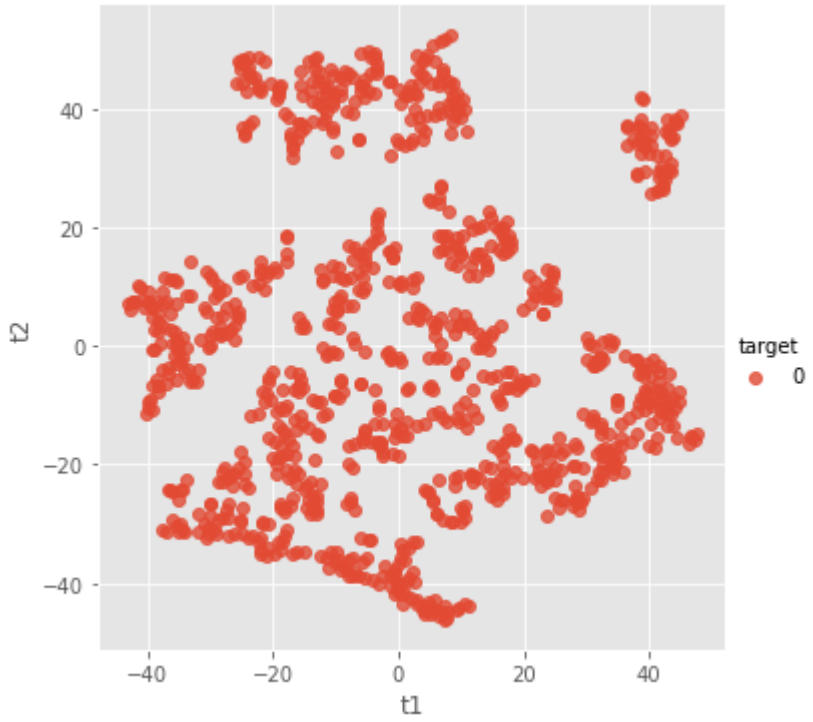
In [34]:
```
tsne = TSNE(n_components=2, perplexity=20)

Xt   = pd.DataFrame(tsne.fit_transform(sample.drop(columns = ['target'])),columns=
```

In [35]:
```
Xt['target'] = sample['target'].reset_index().drop(columns = ['index'])

Xt['target'].value_counts()
```

Out[35]:
```
0    983
Name: target, dtype: int64
```

In [36]:
```
sns.lmplot(data=Xt,x='t1',y='t2',fit_reg=False,hue='target')
```

Out[36]:
```
<seaborn.axisgrid.FacetGrid at 0x1c8b1daed30>
```



========== C H E C K P O I N T ==========

In [37]:
```
import numpy as np
import pandas as pd

from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.decomposition import PCA
from sklearn.manifold import MDS,TSNE
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

from ipywidgets import widgets

import matplotlib.pyplot as plt
import seaborn as sns
import cufflinks as cf
import os
import random

random.seed(10)

cf.go_offline()
os.getcwd()
```

Out[37]:
```
'C:\\Users\\mime1001\\Documents\\Diplomado\\Proyecto\\Entrega 3'
```

In [38]:
```
train = pd.read_csv (os.getcwd() + '/Data/train_data_imp.csv')
train
```

Out[38]:

	target	feature_307	feature_297	feature_81	feature_108	feature_110	feature_82	feature_47
0	0	0	2	4.0	0	0	0	
1	0	1	2	34.0	2	2	0	
2	0	0	2	36.0	2	2	0	
3	0	1	2	48.0	1	1	0	
4	0	0	0	169.0	1	1	123	
...	...	...	...	...	...	...	...	.
49122	0	0	0	144.0	2	2	97	
49123	0	0	2	88.0	1	1	48	
49124	0	0	0	16.0	2	2	15	
49125	0	1	0	149.0	2	2	16	
49126	0	0	0	2.0	0	0	0	

49127 rows × 22 columns

In [39]:
```
### Vamos a sacar las variables numericas
target_column = 'target'
drop_columns = [target_column]

num_columns=['feature_81','feature_82','feature_169','feature_26','feature_66',
             'feature_128','feature_132','feature_156','feature_207']

cat_columns=['feature_307','feature_297','feature_108','feature_110','feature_470
             'feature_254','feature_263','feature_143','feature_298','feature_43

train_num=train[num_columns]
train_num
```

Out[39]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	featu
0	4.0	0	0.0	0	0	0.00	0.805000	1.0244
1	34.0	0	0.0	0	0	3.57	1.150000	0.0000
2	36.0	0	0.0	0	0	0.00	0.000000	0.0000
3	48.0	0	0.0	0	0	0.00	2.723333	0.0000
4	169.0	123	0.0	109	5	3.57	11.041667	7.7846
...	...	...	...	...	...	...	...	
49122	144.0	97	0.0	34394	83	85.71	96.371667	0.0000
49123	88.0	48	0.0	12835	69	96.43	98.391667	9.1345
49124	16.0	15	0.0	61194	229	0.00	0.538333	9.4514
49125	149.0	16	0.0	57103	143	96.43	98.311667	0.0000
49126	2.0	0	0.0	0	0	0.00	0.000000	2.1450

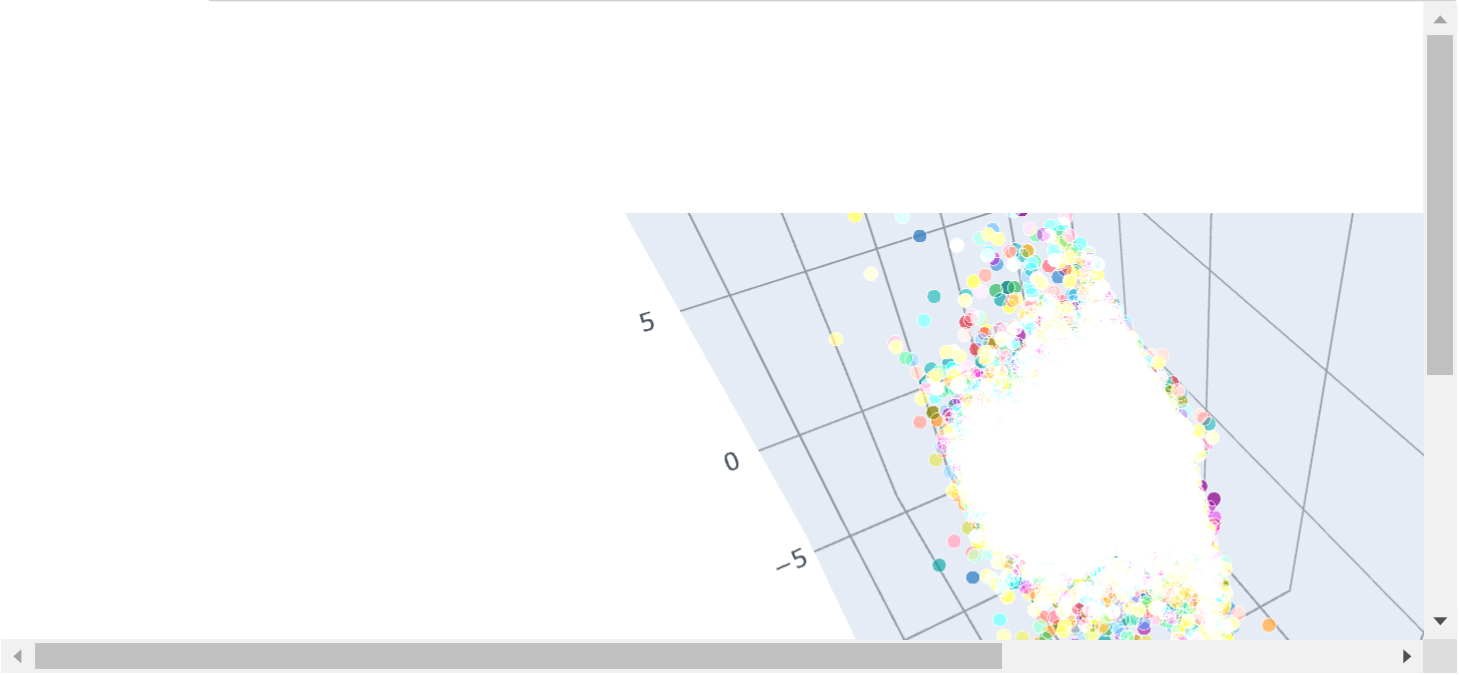
49127 rows × 9 columns

In [40]:
```
train_num.shape,train_num.dropna().shape
```

Out[40]:
```
((49127, 9), (49127, 9))
```

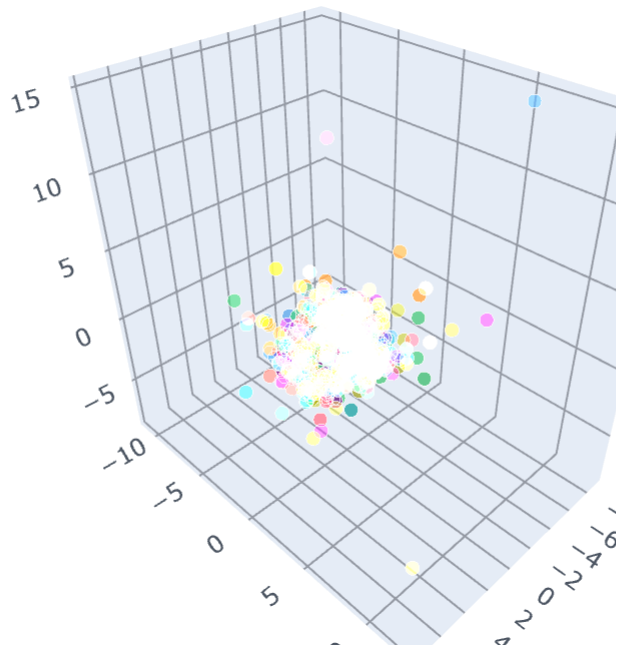
PCA-3D

In [44]:
```
######################################## TRAIN_NUM ################### PCA
X = train_num.copy()

pi = make_pipeline(StandardScaler(),PCA(n_components=3))
Xp = pd.DataFrame(pi.fit_transform(X),columns=['d1','d2','d3'])
Xp.iplot(kind='scatter3d',x='d1',y='d2',z='d3',mode='markers',size=8)
```



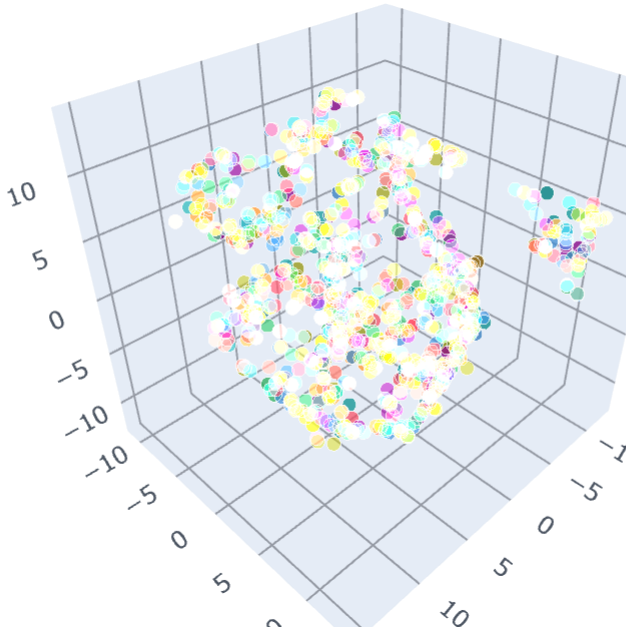
MDS_3D

In [42]:
```
######################################## TRAIN_NUM ##################### MDS
random.seed(10)
sample1=train_num.sample(frac=.02)
#sample1['target'].value_counts()


X = sample1.copy()

pi = make_pipeline(StandardScaler(),MDS(n_components=3,n_jobs=-1))
Xp = pd.DataFrame(pi.fit_transform(X),columns=['d1','d2','d3'])
Xp.iplot(kind='scatter3d',x='d1',y='d2',z='d3',mode='markers',size=8)
```



TSNE_3D

In [43]:
```
######################################## TRAIN 30 ##################### TSNE
random.seed(10)
sample1=train_num.sample(frac=.02)
#sample1['target'].value_counts()


X = sample1.copy()

pi = make_pipeline(StandardScaler(),TSNE(n_components=3, perplexity=20))
Xp = pd.DataFrame(pi.fit_transform(X),columns=['d1','d2','d3'])
Xp.iplot(kind='scatter3d',x='d1',y='d2',z='d3',mode='markers',size=8)
```



>>>>>>>>>>>>>>>>>> Cluster Jerarquico <<<<<<<<<<<<<<<<<<<<<<<

WARD

In [1]:
```
import numpy as np
import pandas as pd

from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.decomposition import PCA
from sklearn.manifold import MDS,TSNE
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

from ipywidgets import widgets

import matplotlib.pyplot as plt
import seaborn as sns
import cufflinks as cf
import os
import random

cf.go_offline()
os.getcwd()
```

Out[1]:
```
'C:\\Users\\mime1001\\Documents\\Diplomado\\Proyecto\\Entrega 3'
```

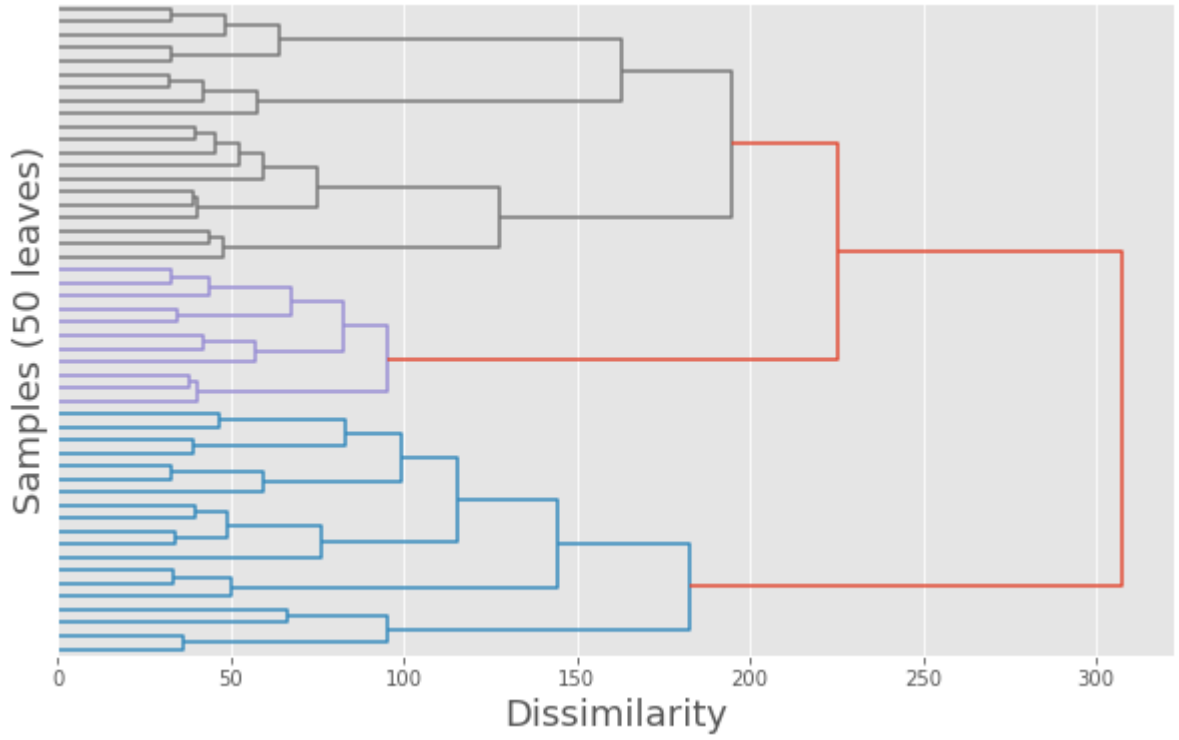
In [2]:
```
train = pd.read_csv (os.getcwd() + '/Data/train_data_imp.csv')


### Vamos a sacar las variables numericas
target_column = 'target'
drop_columns = [target_column]

#num_columns = [col for col in train.select_dtypes(['int64', 'float64']).columns
num_columns=['feature_81','feature_82','feature_169','feature_26','feature_66',
             'feature_128','feature_132','feature_156','feature_207']

cat_columns=['feature_307','feature_297','feature_108','feature_110','feature_476
             'feature_254','feature_263','feature_143','feature_298','feature_43'


train_num=train[num_columns]
train_num
```

Out[2]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	featu
0	4.0	0	0.0	0	0	0.00	0.805000	1.0244
1	34.0	0	0.0	0	0	3.57	1.150000	0.0000
2	36.0	0	0.0	0	0	0.00	0.000000	0.0000
3	48.0	0	0.0	0	0	0.00	2.723333	0.0000
4	169.0	123	0.0	109	5	3.57	11.041667	7.7846
...	...	...	...	...	...	...	...	
49122	144.0	97	0.0	34394	83	85.71	96.371667	0.0000
49123	88.0	48	0.0	12835	69	96.43	98.391667	9.1345
49124	16.0	15	0.0	61194	229	0.00	0.538333	9.4514
49125	149.0	16	0.0	57103	143	96.43	98.311667	0.0000
49126	2.0	0	0.0	0	0	0.00	0.000000	2.1450

49127 rows × 9 columns

In [47]:
```
from sklearn.cluster import AgglomerativeClustering
from scipy.spatial.distance import pdist
from scipy.cluster.hierarchy import linkage, dendrogram


train_num1=train_num.sample(frac=.6)


from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
data_1 = sc.fit_transform(train_num1)


dm = pdist(data_1, metric='euclidean')
Z = linkage(dm, method='ward')

fig, ax = plt.subplots(figsize=(10, 6))

d = dendrogram(Z, orientation='right', truncate_mode='lastp', p=50, no_labels=Tru

ax.set_xlabel('Dissimilarity', fontsize=18)
ax.set_ylabel('Samples (50 leaves)', fontsize=18)

plt.show()
```



========== C H E C K P O I N T ==========

In [1]:
```
import numpy as np
import pandas as pd

from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.decomposition import PCA
from sklearn.manifold import MDS,TSNE
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

from ipywidgets import widgets
from sklearn.cluster import AgglomerativeClustering
from scipy.spatial.distance import pdist
from scipy.cluster.hierarchy import linkage, dendrogram

import matplotlib.pyplot as plt
import seaborn as sns
import cufflinks as cf
import os
import random

cf.go_offline()
os.getcwd()
```

Out[1]:
```
'C:\\Users\\mime1001\\Documents\\Diplomado\\Proyecto\\Entrega 3'
```

In [2]:
```
train = pd.read_csv (os.getcwd() + '/Data/train_data_imp.csv')


### Vamos a sacar las variables numericas
target_column = 'target'
drop_columns = [target_column]

#num_columns = [col for col in train.select_dtypes(['int64', 'float64']).columns

num_columns=['feature_81','feature_82','feature_169','feature_26','feature_66',
             'feature_128','feature_132','feature_156','feature_207']

cat_columns=['feature_307','feature_297','feature_108','feature_110','feature_476
             'feature_254','feature_263','feature_143','feature_298','feature_43

train_num=train[num_columns]
train_num
```

Out[2]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	fe
0	4.0	0	0.0	0	0	0.00	0.805000	1.02
1	34.0	0	0.0	0	0	3.57	1.150000	0.00
2	36.0	0	0.0	0	0	0.00	0.000000	0.00
3	48.0	0	0.0	0	0	0.00	2.723333	0.00
4	169.0	123	0.0	109	5	3.57	11.041667	7.78
...	...	...	...	...	...	...	...	
49122	144.0	97	0.0	34394	83	85.71	96.371667	0.00
49123	88.0	48	0.0	12835	69	96.43	98.391667	9.13
49124	16.0	15	0.0	61194	229	0.00	0.538333	9.45
49125	149.0	16	0.0	57103	143	96.43	98.311667	0.00
49126	2.0	0	0.0	0	0	0.00	0.000000	2.14

PCA- AGLOMERATIVE CLUSTER - WARD

In [3]:
```
#Quitamos la variable objetivo de los datos en el paso anterior

# Escalamos los datos
from sklearn.preprocessing import StandardScaler
X = train_num.copy()
pi = make_pipeline(StandardScaler(),PCA(n_components=2))
data_pca=pd.DataFrame(pi.fit_transform(X),columns=['p1','p2'])
pca_ = data_pca.copy()
```

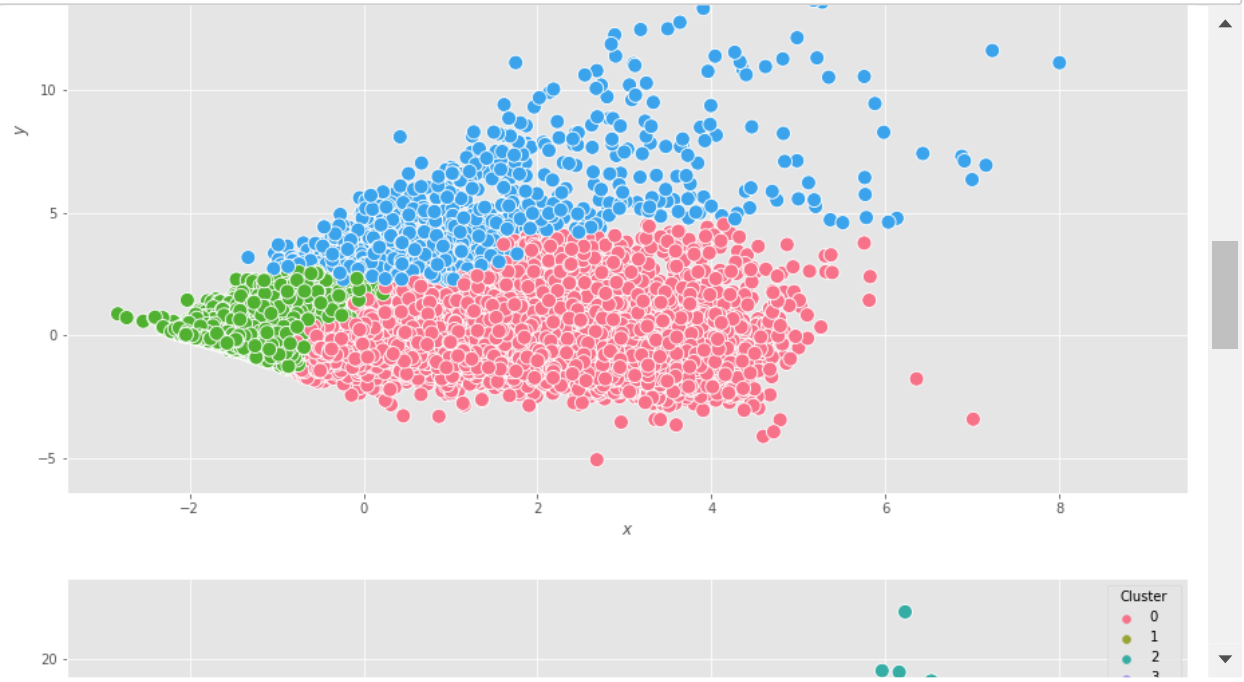
In [4]:
```
pca_
```

Out[4]:

	p1	p2
0	-1.346520	-0.700573
1	-1.699756	-0.128386
2	-1.592280	-0.242254
3	-1.694227	-0.086172
4	1.109887	-1.151116
...	...	...
49122	3.380914	-1.278264
49123	2.872593	-1.678960
49124	-1.335177	1.264774
49125	2.759872	-0.558766
49126	-1.549766	-0.444905

49127 rows × 2 columns

In [63]:
```
## samplemaos por que son muuchos datos
random.seed(10)
pca_=pca_.sample(frac=.6)


for n in (2,3,4,5):
    ag = AgglomerativeClustering(n_clusters=n, affinity='euclidean', linkage='war
    Y_pred = ag.fit_predict(pca_)

    df_pred = pd.Series(Y_pred, name='Cluster', index=pca_.index)
    pdff = pd.concat([pca_, df_pred], axis=1)

    fig, ax = plt.subplots(figsize=(15, 10))
    sns.scatterplot(x='p1',
                    y='p2',
                    hue='Cluster',
                    size=100,

                    sizes=(120, 120),
                    palette=sns.color_palette("husl", n),
                    data=pdff,
                    ax=ax)

    ax.set_xlabel(r'$x$')
    ax.set_ylabel(r'$y$')

    plt.show()
```



MDS - AGLOMERATIVE CLUSTER - WARD

In [5]:
```
from sklearn.manifold import MDS, TSNE
random.seed(10)
sample1=train_num.sample(frac=.02)

X = sample1.copy()
pi = make_pipeline(StandardScaler(),MDS(n_components=2,n_jobs=-1))
data_mds=pd.DataFrame(pi.fit_transform(X),columns=['d1','d2'])
data_mds
```

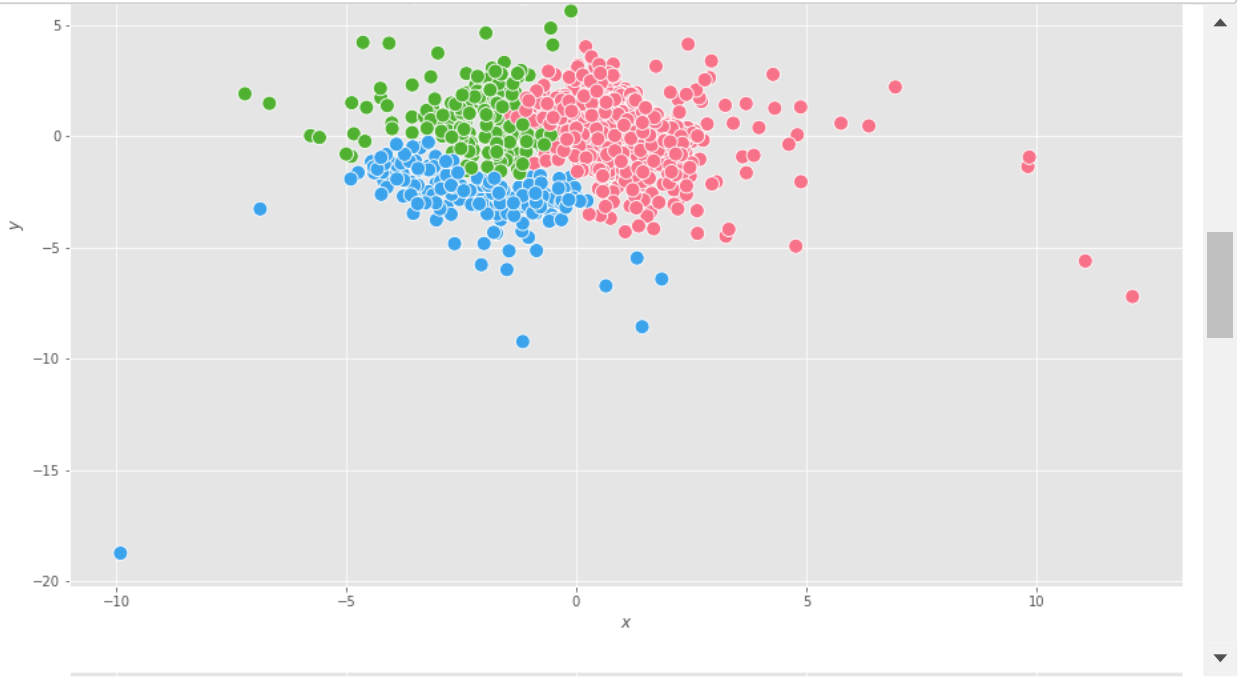
Out[5]:

	d1	d2
0	4.451728	1.534800
1	-0.275993	2.067398
2	1.103050	-3.296250
3	1.358428	-2.152067
4	0.768267	-0.077750
...	...	...
978	0.051133	-1.861216
979	1.270789	4.702645
980	0.309347	-1.231162
981	-1.031336	0.779469
982	-0.337771	0.551430

983 rows × 2 columns

In [6]:
```
mdsdf=data_mds.copy()
```

In [66]:
```
mdsdf=data_mds.copy()

for n in (2,3,4,5):
    ag = AgglomerativeClustering(n_clusters=n, affinity='euclidean', linkage='war
    Y_pred = ag.fit_predict(mdsdf)

    df_pred = pd.Series(Y_pred, name='Cluster', index=data_mds.index)
    pdff = pd.concat([mdsdf, df_pred], axis=1)

    fig, ax = plt.subplots(figsize=(15, 10))
    sns.scatterplot(x='d1',
                    y='d2',
                    hue='Cluster',
                    size=100,

                    sizes=(120, 120),
                    palette=sns.color_palette("husl", n),
                    data=pdff,
                    ax=ax)

    ax.set_xlabel(r'$x$')
    ax.set_ylabel(r'$y$')

    plt.show()
```



In [7]:
```
mdsdf
```

Out[7]:

	d1	d2
0	-1.051764	-3.280572
1	-1.282348	-0.706699
2	0.423039	1.991443
3	0.787312	1.756758
4	-1.942828	2.629944
...	...	...
978	-2.921619	-1.436566
979	1.315340	0.801647
980	-0.987430	0.001338
981	-1.980148	0.735598
982	-1.506865	-0.201481

983 rows × 2 columns

In [7]:
```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
mds_perf=ss.fit_transform(sample1.copy())
mds_perf

mds_to_profile=pd.DataFrame(mds_perf,columns=train_num.columns)
mds_to_profile
```

Out[7]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	feature_1
0	-0.060874	1.355941	-0.201783	0.124659	-0.345043	2.544449	2.122705	2.5589
1	-0.009274	-0.216584	-0.201783	0.099085	0.277757	1.862707	0.586903	-0.4213
2	-0.387673	-0.682518	-0.201783	-0.579810	-0.647545	-0.409659	-0.660441	2.7402
3	0.988325	-0.041859	1.687554	-0.579810	-0.647545	-0.523229	-0.679974	-0.0663
4	-0.404873	-0.682518	-0.201783	0.186958	0.004912	-0.409659	0.949884	0.7278
...	...	...	...	...	...	...	...	
978	-0.611273	-0.682518	-0.201783	-0.564264	-0.594163	-0.523229	-0.608242	0.4770
979	1.435524	0.424074	-0.201783	2.217879	1.671643	2.544449	2.365094	-0.2565
980	0.076726	-0.274826	-0.201783	-0.579810	-0.647545	-0.523229	-0.627010	-0.3509
981	-0.714472	-0.682518	-0.201783	-0.203524	0.182855	0.385653	0.818896	-0.4213
982	0.695925	0.598799	-0.201783	-0.393238	-0.374700	-0.182518	-0.461332	-0.3410

983 rows × 9 columns

In [8]:
```
ag = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='ward')
ag
```

Out[8]:
```
AgglomerativeClustering(n_clusters=4)
```

In [9]:
```
Y_pred = ag.fit_predict(mdsdf)
Y_pred
df_pred = pd.Series(Y_pred, name='Cluster', index=data_mds.index)
df_pred
pdff = pd.concat([mdsdf, df_pred], axis=1)
pdff
```

Out[9]:

	d1	d2	Cluster
0	4.451728	1.534800	1
1	-0.275993	2.067398	3
2	1.103050	-3.296250	0
3	1.358428	-2.152067	0
4	0.768267	-0.077750	0
...	...	...	...
978	0.051133	-1.861216	0
979	1.270789	4.702645	1
980	0.309347	-1.231162	0
981	-1.031336	0.779469	3
982	-0.337771	0.551430	0

983 rows × 3 columns

In [10]:
```
mds_to_profile['cl']= pdff['Cluster']
mds_to_profile
```

Out[10]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	feature_1
0	-0.060874	1.355941	-0.201783	0.124659	-0.345043	2.544449	2.122705	2.5589
1	-0.009274	-0.216584	-0.201783	0.099085	0.277757	1.862707	0.586903	-0.4213
2	-0.387673	-0.682518	-0.201783	-0.579810	-0.647545	-0.409659	-0.660441	2.7402
3	0.988325	-0.041859	1.687554	-0.579810	-0.647545	-0.523229	-0.679974	-0.0663
4	-0.404873	-0.682518	-0.201783	0.186958	0.004912	-0.409659	0.949884	0.7278
...	...	...	...	...	...	...	...	
978	-0.611273	-0.682518	-0.201783	-0.564264	-0.594163	-0.523229	-0.608242	0.4770
979	1.435524	0.424074	-0.201783	2.217879	1.671643	2.544449	2.365094	-0.2565
980	0.076726	-0.274826	-0.201783	-0.579810	-0.647545	-0.523229	-0.627010	-0.3509
981	-0.714472	-0.682518	-0.201783	-0.203524	0.182855	0.385653	0.818896	-0.4213
982	0.695925	0.598799	-0.201783	-0.393238	-0.374700	-0.182518	-0.461332	-0.3410

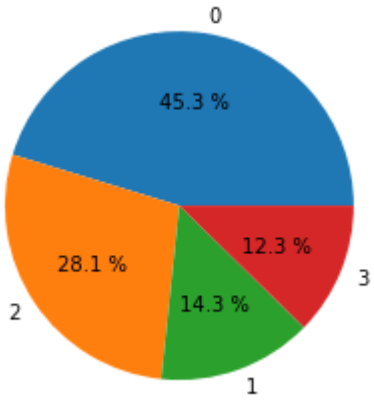
983 rows × 10 columns

In [11]:
```
z=mds_to_profile.groupby('cl').mean().style.background_gradient(cmap='Blues')
z
```

Out[11]:

cl	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	feature_15
0	0.126664	0.275394	-0.118422	-0.219449	-0.204728	-0.310050	-0.230169	0.30924
1	0.880733	0.515774	0.010680	0.358288	0.340858	2.131060	1.947823	-0.03758
2	-0.827269	-0.590091	0.161021	-0.334876	-0.372894	-0.483721	-0.612147	-0.38552
3	0.394855	-0.267847	0.055787	1.153404	1.206295	-0.239672	-0.026985	-0.21411

In [37]:
```
circ=pd.DataFrame(mds_to_profile['cl'].value_counts())
circ

nombres=['0','2','1','3']

plt.pie(circ['cl'], labels= nombres,autopct="%0.1f %%")
plt.show()
```



In [36]:
```
display(circ.index)
display(circ)
```

```
Int64Index([0, 2, 1, 3], dtype='int64')
```

	cl
0	445
2	276
1	141
3	121

In [21]:
```
z1=z=mds_to_profile.groupby('cl').mean()
z1
```

Out[21]:

cl	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	feature_
0	0.865844	0.405304	-0.042999	0.391283	0.419205	1.316429	1.304370	-0.174
1	-0.191703	-0.202460	-0.089783	-0.252303	-0.271697	-0.468278	-0.415030	0.235
2	-0.651666	-0.535657	0.118789	-0.168061	-0.184087	-0.499823	-0.594135	-0.378
3	0.683386	1.305372	-0.131058	0.205981	0.236796	-0.257525	-0.019681	1.183

In [22]:
```
type(z1)
```

Out[22]:
```
pandas.core.frame.DataFrame
```

In [27]:
```
a=pd.DataFrame(z1.to_records())
a
```

Out[27]:

	cl	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	feature_
0	0	0.865844	0.405304	-0.042999	0.391283	0.419205	1.316429	1.304370	-0.174
1	1	-0.191703	-0.202460	-0.089783	-0.252303	-0.271697	-0.468278	-0.415030	0.235
2	2	-0.651666	-0.535657	0.118789	-0.168061	-0.184087	-0.499823	-0.594135	-0.378
3	3	0.683386	1.305372	-0.131058	0.205981	0.236796	-0.257525	-0.019681	1.183

In [26]:
```
mds_to_profile.groupby('cl').mean()
```

Out[26]:

cl	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	feature_15
0	0.865844	0.405304	-0.042999	0.391283	0.419205	1.316429	1.304370	-0.17464
1	-0.191703	-0.202460	-0.089783	-0.252303	-0.271697	-0.468278	-0.415030	0.23583
2	-0.651666	-0.535657	0.118789	-0.168061	-0.184087	-0.499823	-0.594135	-0.37878
3	0.683386	1.305372	-0.131058	0.205981	0.236796	-0.257525	-0.019681	1.18375

In [ ]:
```

```

In [14]:
```
X=sample1.copy()
sc = StandardScaler()
Xs = pd.DataFrame(sc.fit_transform(X),columns=num_columns)
Xs['cl_mds'] = mds_to_profile['cl']
Xs
```

Out[14]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	feature_1
0	-0.477182	2.617094	-0.208743	0.921380	1.709349	-0.077041	-0.258941	-0.3212
1	-0.280651	1.013156	-0.208743	-0.479538	-0.453719	-0.384567	-0.667135	-0.3175
2	-1.214173	-0.675199	-0.208743	0.592095	1.095712	-0.589393	-0.552567	-0.4326
3	-1.066775	-0.506364	-0.208743	0.829524	0.967871	0.025372	-0.424257	-0.4326
4	-1.034020	-0.675199	2.243201	-0.511500	-0.596901	-0.384567	-0.600793	-0.1677
...	...	...	...	...	...	...	...	
978	0.325319	-0.675199	-0.208743	0.470889	0.083212	-0.179454	-0.426676	2.6183
979	1.062310	-0.675199	-0.208743	-0.511500	-0.596901	-0.589393	0.859001	-0.4326
980	-0.166008	-0.506364	-0.208743	-0.409830	-0.397469	-0.179454	-0.329761	-0.3709
981	-1.115907	-0.675199	-0.208743	-0.511500	-0.596901	-0.589393	-0.754220	-0.1164
982	-0.395294	0.450371	-0.208743	-0.511500	-0.596901	-0.589393	-0.754220	-0.2135

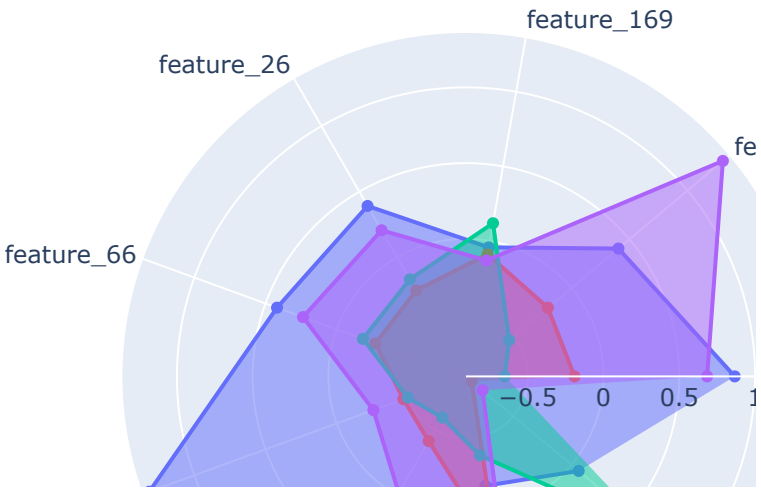
983 rows × 10 columns

In [15]:
```
aux = Xs[num_columns+['cl_mds']].groupby('cl_mds').mean()
aux
```

Out[15]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	featu
cl_mds								
0	0.865844	0.405304	-0.042999	0.391283	0.419205	1.316429	1.304370	-0.1
1	-0.191703	-0.202460	-0.089783	-0.252303	-0.271697	-0.468278	-0.415030	0.2
2	-0.651666	-0.535657	0.118789	-0.168061	-0.184087	-0.499823	-0.594135	-0.3
3	0.683386	1.305372	-0.131058	0.205981	0.236796	-0.257525	-0.019681	1.1

In [16]:
```
import plotly.graph_objects as go
fig = go.Figure()

for i,row in aux.iterrows():
    fig.add_trace(go.Scatterpolar(r=row.values,
                                  theta=num_columns,
                                  fill='toself',
                                  name=f'cluster {i}'))
fig.show()
```



TSNE - AGLOMERATIVE CLUSTER - WARD

In [3]:
```
from sklearn.manifold import MDS, TSNE
random.seed(10)
sample1=train_num.sample(frac=.02)

X = sample1.copy()
pi = make_pipeline(StandardScaler(),TSNE(n_components=2,perplexity=20))
data_tsne=pd.DataFrame(pi.fit_transform(X),columns=['d1','d2'])
data_tsne
```

```
C:\Users\mime1001\Anaconda3\lib\site-packages\sklearn\manifold\_t_sne.py:795: F
utureWarning:

The default initialization in TSNE will change from 'random' to 'pca' in 1.2.

C:\Users\mime1001\Anaconda3\lib\site-packages\sklearn\manifold\_t_sne.py:805: F
utureWarning:

The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
```

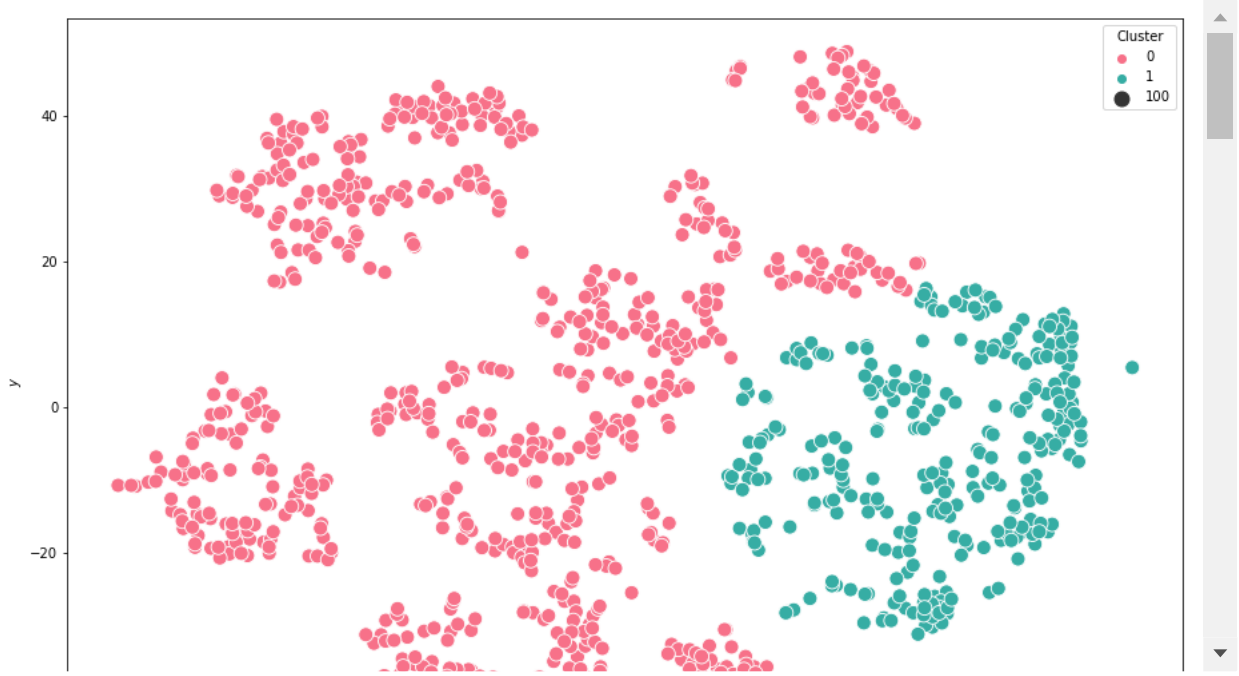
Out[3]:

	d1	d2
0	23.371346	-11.206347
1	3.131590	29.563349
2	-24.760332	-23.977510
3	30.208942	-19.775650
4	24.197075	30.553276
...	...	...
978	-12.927750	-37.991676
979	-56.489048	-5.359496
980	33.364605	23.816820
981	1.108104	29.250780
982	34.658699	-15.640128

983 rows × 2 columns

In [4]:
```
t_sne=data_tsne.copy()
```

In [100]:
```
t_sne=data_tsne.copy()


for n in (2,3,4,5):
    ag = AgglomerativeClustering(n_clusters=n, affinity='euclidean', linkage='war
    Y_pred = ag.fit_predict(t_sne)

    df_pred = pd.Series(Y_pred, name='Cluster', index=data_tsne.index)
    pdff = pd.concat([t_sne, df_pred], axis=1)

    fig, ax = plt.subplots(figsize=(15, 10))
    sns.scatterplot(x='d1',
                    y='d2',
                    hue='Cluster',
                    size=100,

                    sizes=(120, 120),
                    palette=sns.color_palette("husl", n),
                    data=pdff,
                    ax=ax)

    ax.set_xlabel(r'$x$')
    ax.set_ylabel(r'$y$')

    plt.show()
```



In [5]:
```
t_sne
```

Out[5]:

	d1	d2
0	23.371346	-11.206347
1	3.131590	29.563349
2	-24.760332	-23.977510
3	30.208942	-19.775650
4	24.197075	30.553276
...	...	...
978	-12.927750	-37.991676
979	-56.489048	-5.359496
980	33.364605	23.816820
981	1.108104	29.250780
982	34.658699	-15.640128

In [6]:
```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
tsne_perf=ss.fit_transform(sample1.copy())
tsne_perf

tsne_to_profile=pd.DataFrame(tsne_perf,columns=train_num.columns)
tsne_to_profile
```

Out[6]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	feature_1
0	-0.232607	2.055346	-0.192592	0.047947	0.111493	-0.576591	-0.655945	-0.1132
1	-0.248994	-0.608108	-0.192592	-0.024352	-0.014235	0.289693	-0.105250	-0.3925
2	-1.232234	-0.688819	-0.192592	-0.550863	-0.596309	-0.468344	-0.637749	-0.3510
3	0.455661	1.947731	-0.192592	-0.250823	-0.214469	-0.576591	-0.673768	-0.3526
4	1.766648	-0.688819	1.856968	2.080845	0.674941	2.239058	2.297049	-0.2873
...	...	...	...	...	...	...	...	
978	-0.642290	-0.688819	-0.192592	-0.227964	0.158059	-0.576591	-0.674889	0.8243
979	-0.937262	0.172095	2.734995	-0.550863	-0.596309	-0.576591	-0.743726	-0.1823
980	1.602775	-0.688819	-0.192592	-0.317500	-0.372793	2.347306	2.055106	-0.3925
981	0.340950	-0.688819	-0.192592	-0.372176	-0.377449	-0.576591	0.100246	-0.3925
982	1.422514	2.243671	-0.192592	0.487399	0.083553	-0.576591	-0.690951	-0.2633

983 rows × 9 columns

In [103]:
```
ag = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='ward')
ag
```

Out[103]:
```
AgglomerativeClustering(n_clusters=4)
```

In [104]:
```
Y_pred = ag.fit_predict(t_sne)
Y_pred
df_pred = pd.Series(Y_pred, name='Cluster', index=data_tsne.index)
df_pred
pdff = pd.concat([t_sne, df_pred], axis=1)
pdff
```

Out[104]:

	d1	d2	Cluster
0	4.584978	-32.637280	0
1	-16.661797	-30.869528	0
2	32.038467	-13.487659	1
3	-34.674686	-18.587141	0
4	30.224991	-13.440310	1
...	...	...	...
978	7.885480	14.491110	0
979	43.349033	-0.919605	1
980	-0.813230	-22.179525	0
981	-27.383667	30.465626	3
982	26.986536	2.473361	1

983 rows × 3 columns

In [105]:
```
tsne_to_profile['cl']= pdff['Cluster']
tsne_to_profile
```

Out[105]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	feature_1
0	-0.634190	-0.612764	-0.217309	-0.551649	-0.599971	-0.547214	-0.606106	-0.2845
1	0.790837	-0.671063	-0.217309	-0.516718	-0.455111	-0.547214	-0.318503	1.7096
2	-0.732467	-0.583615	-0.217309	-0.597571	-0.652121	-0.547214	-0.728900	-0.1099
3	-0.093662	2.506242	-0.217309	-0.469195	-0.449317	-0.547214	-0.675354	-0.3801
4	-0.552291	-0.671063	-0.217309	-0.597571	-0.652121	-0.547214	-0.710871	-0.3101
...	...	...	...	...	...	...	...	
978	1.085671	-0.671063	-0.217309	0.689837	2.349380	0.097941	0.309870	-0.4080
979	-1.109199	-0.671063	-0.217309	-0.535359	-0.559410	-0.547214	-0.693437	-0.4080
980	-0.077282	-0.671063	-0.217309	-0.292099	-0.275485	-0.547214	-0.624135	-0.4080
981	1.593439	0.290873	-0.217309	-0.406325	-0.559410	2.355833	1.545392	-0.0904
982	-0.552291	-0.671063	-0.217309	0.167427	-0.263896	-0.547214	-0.728900	-0.4080

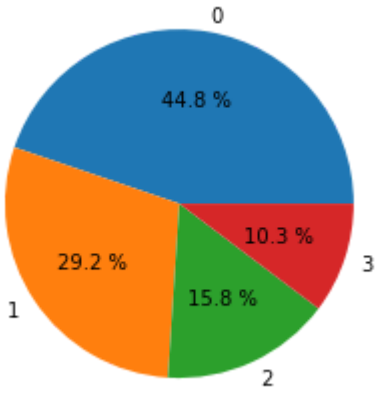
983 rows × 10 columns

In [106]:
```
tsne_to_profile.groupby('cl').mean().style.background_gradient(cmap='Blues')
```

Out[106]:

cl	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	feature_15
0	0.187435	0.305779	-0.171944	0.247583	0.270032	-0.393767	-0.260819	0.24534
1	-0.852204	-0.515565	-0.209493	-0.373352	-0.402013	-0.505639	-0.599636	-0.35849
2	0.556009	-0.382742	1.567827	-0.094189	-0.126264	-0.142699	0.083843	-0.24929
3	0.683577	0.336008	-0.145617	0.049862	0.060105	2.147021	1.796051	0.12978

In [107]:
```
circ=pd.DataFrame(tsne_to_profile['cl'].value_counts())
circ

nombres=['0','1','2','3']

plt.pie(circ['cl'], labels= nombres,autopct="%0.1f %%")
plt.show()


display(circ.index)
display(circ)
```




```
Int64Index([0, 1, 3, 2], dtype='int64')
```

	cl
0	440
1	287
3	155
2	101

In [108]:
```
A=tsne_to_profile.groupby('cl').mean()
A=pd.DataFrame(A.to_records())
A
```

Out[108]:

	cl	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	feature_
0	0	0.187435	0.305779	-0.171944	0.247583	0.270032	-0.393767	-0.260819	0.245
1	1	-0.852204	-0.515565	-0.209493	-0.373352	-0.402013	-0.505639	-0.599636	-0.358
2	2	0.556009	-0.382742	1.567827	-0.094189	-0.126264	-0.142699	0.083843	-0.249
3	3	0.683577	0.336008	-0.145617	0.049862	0.060105	2.147021	1.796051	0.129

In [109]:
```
X=sample1.copy()
sc = StandardScaler()
Xs = pd.DataFrame(sc.fit_transform(X),columns=num_columns)
Xs['cl_tsne'] = tsne_to_profile['cl']
Xs
```

Out[109]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	feature_1
0	-0.634190	-0.612764	-0.217309	-0.551649	-0.599971	-0.547214	-0.606106	-0.2845
1	0.790837	-0.671063	-0.217309	-0.516718	-0.455111	-0.547214	-0.318503	1.7096
2	-0.732467	-0.583615	-0.217309	-0.597571	-0.652121	-0.547214	-0.728900	-0.1099
3	-0.093662	2.506242	-0.217309	-0.469195	-0.449317	-0.547214	-0.675354	-0.3801
4	-0.552291	-0.671063	-0.217309	-0.597571	-0.652121	-0.547214	-0.710871	-0.3101
...	...	...	...	...	...	...	...	
978	1.085671	-0.671063	-0.217309	0.689837	2.349380	0.097941	0.309870	-0.4080
979	-1.109199	-0.671063	-0.217309	-0.535359	-0.559410	-0.547214	-0.693437	-0.4080
980	-0.077282	-0.671063	-0.217309	-0.292099	-0.275485	-0.547214	-0.624135	-0.4080
981	1.593439	0.290873	-0.217309	-0.406325	-0.559410	2.355833	1.545392	-0.0904
982	-0.552291	-0.671063	-0.217309	0.167427	-0.263896	-0.547214	-0.728900	-0.4080

983 rows × 10 columns

In [110]:
```
aux = Xs[num_columns+['cl_tsne']].groupby('cl_tsne').mean()
aux
```

Out[110]:

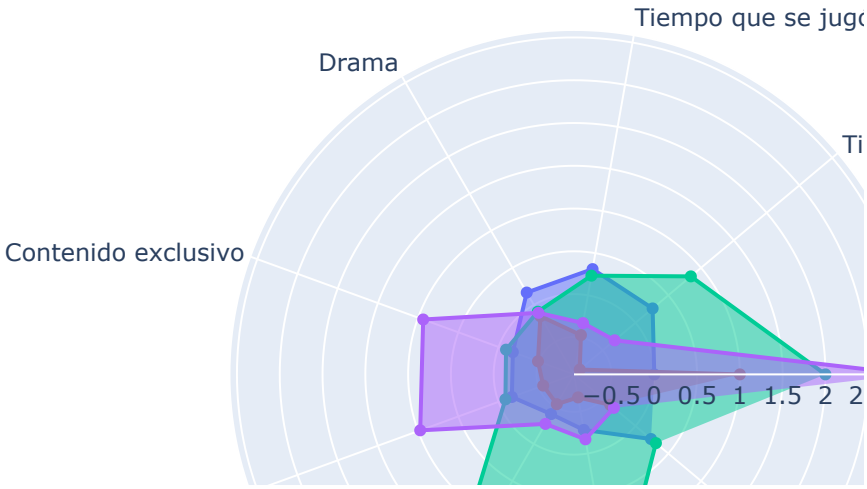
cl_tsne	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	featu
0	0.187435	0.305779	-0.171944	0.247583	0.270032	-0.393767	-0.260819	0.2
1	-0.852204	-0.515565	-0.209493	-0.373352	-0.402013	-0.505639	-0.599636	-0.3
2	0.556009	-0.382742	1.567827	-0.094189	-0.126264	-0.142699	0.083843	-0.2
3	0.683577	0.336008	-0.145617	0.049862	0.060105	2.147021	1.796051	0.1

In [9]:
```
################3 presentación
aux1=pd.DataFrame()
aux1['Cluster']=['0','1','2','3']
aux1['# Episodios serie']=[0.261012,-0.850586,0.844319,-0.318199]
aux1['Tiempo que se vio netflix']=[0.311463,-0.471783,0.233789,-0.329137]
aux1['Tiempo que se jugó un juego']=[0.166503,-0.15268,-0.091959,-0.107434]
aux1['Categoria Drama']=[-0.177534,-0.490249,-0.092965,0.937453]
aux1['Contenido exclusivo']=[-0.165435,-0.555312,-0.085545,0.974095]
aux1[' % Uso de funciones']=[-0.401723,-0.535951,2.074007,-0.269554]
aux1['Top 10']=[-0.279055,-0.665048,1.797918,-0.16739]
aux1['Categoria Sci-Fi']=[0.236947,-0.384464,0.311469,-0.329397]
aux1['Categoria Niños']=[-0.520186,0.707558,-0.310579,0.569504]


aux1.style.background_gradient(cmap='Blues')
```

Out[9]:

	Cluster	# Episodios serie	Tiempo que se vio netflix	Tiempo que se jugó un juego	Categoria Drama	Contenido exclusivo	% Uso de funciones	Top 10	Categoria Sci-Fi
0	0	0.261012	0.311463	0.166503	-0.177534	-0.165435	-0.401723	-0.279055	0.236947
1	1	-0.850586	-0.471783	-0.152680	-0.490249	-0.555312	-0.535951	-0.665048	-0.384464
2	2	0.844319	0.233789	-0.091959	-0.092965	-0.085545	2.074007	1.797918	0.311469
3	3	-0.318199	-0.329137	-0.107434	0.937453	0.974095	-0.269554	-0.167390	-0.329397

In [10]:
```
import plotly.graph_objects as go
fig = go.Figure()

num_columns_2=['# episodios serie','Tiempo que se vio netflix','Tiempo que se jug
               'Drama','Contenido exclusivo',' % uso de funciones','Top 10','Sci-

for i,row in aux1.iterrows():
    fig.add_trace(go.Scatterpolar(r=row.values,
                                  theta=num_columns_2,
                                  fill='toself',
                                  name=f'cluster {i}'))
fig.show()
```



>>>>>>>>>>>>>>>>>> Cluster Optimización <<<<<<<<<<<<<<<<<<<<<<

In [70]:
```
import numpy as np
import pandas as pd

from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.decomposition import PCA
from sklearn.manifold import MDS,TSNE
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

from ipywidgets import widgets
from sklearn.cluster import AgglomerativeClustering
from scipy.spatial.distance import pdist
from scipy.cluster.hierarchy import linkage, dendrogram

import matplotlib.pyplot as plt
import seaborn as sns
import cufflinks as cf
import os
import random

cf.go_offline()
os.getcwd()
```

Out[70]:
```
'C:\\Users\\mime1001\\Documents\\Diplomado\\Proyecto\\Entrega 3'
```

In [71]:
```
train = pd.read_csv (os.getcwd() + '/Data/train_data_imp.csv')


### Vamos a sacar las variables numericas
target_column = 'target'
drop_columns = [target_column]

#num_columns = [col for col in train.select_dtypes(['int64', 'float64']).columns
num_columns=['feature_81','feature_82','feature_169','feature_26','feature_66',
             'feature_128','feature_132','feature_156','feature_207']

cat_columns=['feature_307','feature_297','feature_108','feature_110','feature_476
             'feature_254','feature_263','feature_143','feature_298','feature_43'

train_num=train[num_columns]
train_num
```

Out[71]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	featu
0	4.0	0	0.0	0	0	0.00	0.805000	1.0244
1	34.0	0	0.0	0	0	3.57	1.150000	0.0000
2	36.0	0	0.0	0	0	0.00	0.000000	0.0000
3	48.0	0	0.0	0	0	0.00	2.723333	0.0000
4	169.0	123	0.0	109	5	3.57	11.041667	7.7846
...	...	...	...	...	...	...	...	
49122	144.0	97	0.0	34394	83	85.71	96.371667	0.0000
49123	88.0	48	0.0	12835	69	96.43	98.391667	9.1345
49124	16.0	15	0.0	61194	229	0.00	0.538333	9.4514
49125	149.0	16	0.0	57103	143	96.43	98.311667	0.0000
49126	2.0	0	0.0	0	0	0.00	0.000000	2.1450

49127 rows × 9 columns

Selección del numero de cluster

In [72]:
```
from sklearn.preprocessing import MinMaxScaler
sc_mm = MinMaxScaler()
Xs = pd.DataFrame(index = train_num.index, data = sc_mm.fit_transform(train_num),
Xs.head()
```

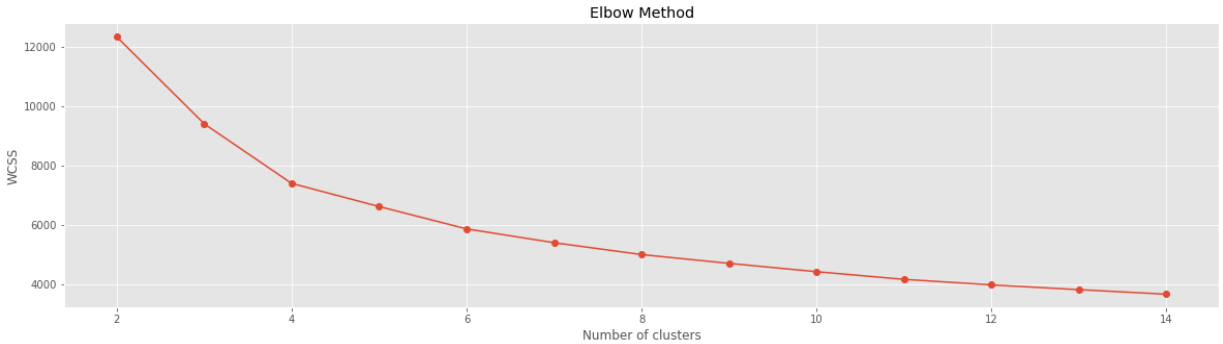
Out[72]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	feature_156
0	0.015326	0.000000	0.181908	0.000000	0.000000	0.0000	0.007881	0.072309
1	0.130268	0.000000	0.181908	0.000000	0.000000	0.0357	0.011258	0.000000
2	0.137931	0.000000	0.181908	0.000000	0.000000	0.0000	0.000000	0.000000
3	0.183908	0.000000	0.181908	0.000000	0.000000	0.0000	0.026661	0.000000
4	0.647510	0.960938	0.181908	0.000066	0.001243	0.0357	0.108094	0.005495

Metodo del codo

Ver desde donde deja de bajar

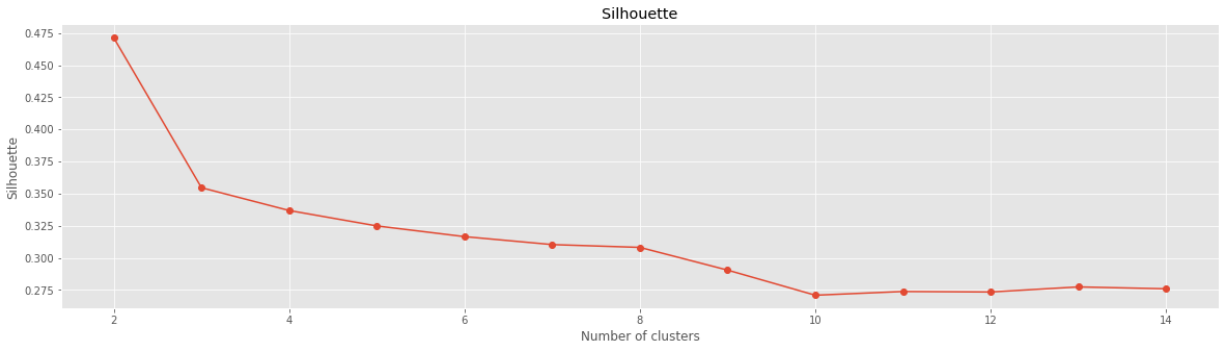
In [73]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import davies_bouldin_score, silhouette_score, silhouette_sa

wcss = []

for i in range(2, 15):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, rand
    kmeans.fit(Xs)
    wcss.append(kmeans.inertia_)
plt.subplots(figsize=(20,5))
plt.plot(range(2,15 ), wcss,marker= 'o')
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
sns.despine()
plt.show()
```



Silueta

Que se acercque a 1

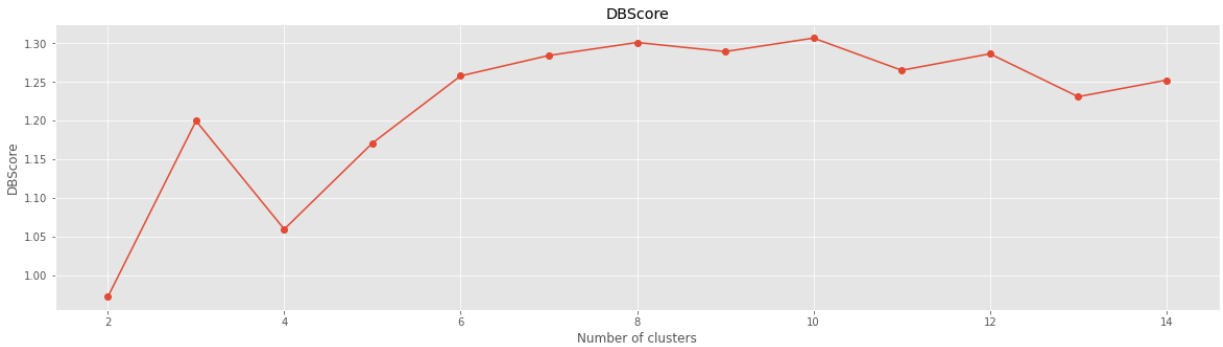
In [74]:
```
lst_silhouette = []
for k in range(2,15):
    cl_sil = KMeans(n_clusters=k)
    cl_sil.fit(Xs)
    lst_silhouette.append(silhouette_score(Xs,cl_sil.predict(Xs)))
plt.subplots(figsize=(20,5))
plt.plot(range(2, 15), lst_silhouette,marker= 'o')
plt.title('Silhouette')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette')
sns.despine()
plt.show()
```



Davies Bouldin

Buscamos el valor más pequeño

In [75]:
```
lst_dbscore=[]
for k in range(2,15):
    cl_db = KMeans(n_clusters=k)
    cl_db.fit(Xs)
    lst_dbscore.append(davies_bouldin_score(Xs,cl_db.predict(Xs)))
plt.subplots(figsize=(20,5))
plt.plot(range(2, 15), lst_dbscore,marker= 'o')
plt.title('DBScore')
plt.xlabel('Number of clusters')
plt.ylabel('DBScore')
sns.despine()
plt.show()
```



- Del codo se ven como 5 o 6

- De la silueta se ven como 2 y 3
- De DB se ven 2 y 4

In [ ]:
```
#pip install yellowbrick
```

In [76]:
```
from yellowbrick.cluster import SilhouetteVisualizer

for j,i in enumerate([2,3,4,5,6]):
    fig, ax = plt.subplots(figsize=(10,15))
    km = KMeans(n_clusters=i, init='k-means++', n_init=10, max_iter=100, random_s


    visualizer = SilhouetteVisualizer(km,ax=ax, colors='yellowbrick',)

    visualizer.fit(Xs)
    visualizer.show()
```

========== C H E C K P O I N T ==========

In [77]:
```
import numpy as np
import pandas as pd

from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.decomposition import PCA
from sklearn.manifold import MDS,TSNE
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

from ipywidgets import widgets
from sklearn.cluster import AgglomerativeClustering
from scipy.spatial.distance import pdist
from scipy.cluster.hierarchy import linkage, dendrogram

import matplotlib.pyplot as plt
import seaborn as sns
import cufflinks as cf
import os
import random

cf.go_offline()
os.getcwd()
```

Out[77]:
```
'C:\\Users\\mime1001\\Documents\\Diplomado\\Proyecto\\Entrega 3'
```

In [78]:
```
train = pd.read_csv (os.getcwd() + '/Data/train_data_imp.csv')


### Vamos a sacar las variables numericas
target_column = 'target'
drop_columns = [target_column]

#num_columns = [col for col in train.select_dtypes(['int64', 'float64']).columns
num_columns=['feature_81','feature_82','feature_169','feature_26','feature_66',
             'feature_128','feature_132','feature_156','feature_207']

cat_columns=['feature_307','feature_297','feature_108','feature_110','feature_476
             'feature_254','feature_263','feature_143','feature_298','feature_43'

train_num=train[num_columns]
train_num
```

Out[78]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	featu
0	4.0	0	0.0	0	0	0.00	0.805000	1.0244
1	34.0	0	0.0	0	0	3.57	1.150000	0.0000
2	36.0	0	0.0	0	0	0.00	0.000000	0.0000
3	48.0	0	0.0	0	0	0.00	2.723333	0.0000
4	169.0	123	0.0	109	5	3.57	11.041667	7.7846
...	...	...	...	...	...	...	...	
49122	144.0	97	0.0	34394	83	85.71	96.371667	0.0000
49123	88.0	48	0.0	12835	69	96.43	98.391667	9.1345
49124	16.0	15	0.0	61194	229	0.00	0.538333	9.4514
49125	149.0	16	0.0	57103	143	96.43	98.311667	0.0000
49126	2.0	0	0.0	0	0	0.00	0.000000	2.1450

49127 rows × 9 columns

In [79]:
```
# Vamos a samplear para PCA, MDS y TSNE los mismos datos
random.seed(7)
a=random.sample(range(1,len(train_num),1),1400)
a

train_sam=train_num.iloc[a]
train_sam
```

Out[79]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	featu
21223	191.0	0	0.000000	11311	37	100.00	100.000000	2.5973
9887	33.0	4	0.000000	16409	47	0.00	1.650000	0.0000
25876	28.0	0	0.000000	0	0	0.00	0.000000	9.0000
42660	99.0	0	0.000000	21396	54	28.57	57.403333	0.0000
3165	139.0	105	0.000000	34481	177	92.86	92.590000	0.0000
...	...	...	...	...	...	...	...	
32147	74.0	0	0.000000	31752	88	0.00	2.166667	8.0601
12933	25.0	0	0.000000	0	0	0.00	0.000000	9.0462
19767	128.0	115	0.000000	14060	33	0.00	3.260000	5.4053
8301	74.0	5	0.000000	135780	1070	7.14	3.393333	2.8535
2851	199.0	0	5.288333	5850	14	96.43	98.810000	4.3333

1400 rows × 9 columns

In [80]:
```
######################################################## PCA
from sklearn.preprocessing import StandardScaler
X = train_sam.copy()
pi = make_pipeline(StandardScaler(),PCA(n_components=2))
data_pca=pd.DataFrame(pi.fit_transform(X),columns=['p1','p2'])
pca_ = data_pca.copy()
pca_
```

Out[80]:

	p1	p2
0	2.604581	-1.909930
1	-1.585352	0.121805
2	-1.601598	-0.200426
3	0.043542	-0.597660
4	3.395292	-1.107896
...	...	...
1395	-0.685068	0.109018
1396	-1.451138	-0.236081
1397	1.535770	-1.148138
1398	1.089939	4.366929
1399	2.506437	-2.037116

1400 rows × 2 columns

In [81]:
```
######################################################## MDS
from sklearn.manifold import MDS, TSNE
X = train_sam.copy()
pi = make_pipeline(StandardScaler(),MDS(n_components=2,n_jobs=-1))
data_mds=pd.DataFrame(pi.fit_transform(X),columns=['d1','d2'])
data_mds
```

Out[81]:

	d1	d2
0	2.389285	3.305593
1	-1.814755	0.283268
2	-1.436081	-0.638565
3	-0.728588	2.220252
4	3.553996	1.866137
...	...	...
1395	-0.345049	-1.771325
1396	-1.209308	-1.342362
1397	3.935973	-9.078602
1398	2.699113	-4.631248
1399	2.392990	3.827244

1400 rows × 2 columns

In [82]:
```
######################################################## TSNE
from sklearn.manifold import MDS, TSNE
X = train_sam.copy()
pi = make_pipeline(StandardScaler(),TSNE(n_components=2,perplexity=20))
data_tsne=pd.DataFrame(pi.fit_transform(X),columns=['d1','d2'])
data_tsne
```

Out[82]:

	d1	d2
0	13.544980	-48.216633
1	-14.255484	40.330456
2	-30.243027	12.633109
3	21.134142	29.441469
4	35.994881	-37.384102
...	...	...
1395	-29.006811	-20.503294
1396	-33.999680	-6.844424
1397	-31.254869	-37.635826
1398	13.723424	4.906366
1399	11.911777	-53.751362

1400 rows × 2 columns

In [83]:
```
from sklearn.preprocessing import MinMaxScaler
sc_mm = MinMaxScaler()
Xs = pd.DataFrame(index = train_num.index, data = sc_mm.fit_transform(train_num),
Xs

Xs=Xs.iloc[a]
Xs
```

Out[83]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	featu
21223	0.731801	0.000000	0.181908	0.006803	0.009202	1.0000	0.978968	1.8332
9887	0.126437	0.031250	0.181908	0.009869	0.011689	0.0000	0.016153	0.0000
25876	0.107280	0.000000	0.181908	0.000000	0.000000	0.0000	0.000000	6.3523
42660	0.379310	0.000000	0.181908	0.012868	0.013429	0.2857	0.561961	0.0000
3165	0.532567	0.820312	0.181908	0.020738	0.044019	0.9286	0.906427	0.0000
...	...	...	...	...	...	...	...	
32147	0.283525	0.000000	0.181908	0.019096	0.021885	0.0000	0.021211	5.6889
12933	0.095785	0.000000	0.181908	0.000000	0.000000	0.0000	0.000000	6.3849
19767	0.490421	0.898438	0.181908	0.008456	0.008207	0.0000	0.031914	3.8151
8301	0.283525	0.039062	0.181908	0.081662	0.266103	0.0714	0.033220	2.0140
2851	0.762452	0.000000	0.239964	0.003518	0.003482	0.9643	0.967319	3.0585

1400 rows × 9 columns

In [84]:
```
from sklearn.cluster import KMeans
km = KMeans(n_clusters = 2,max_iter=100)
km.fit(Xs)
```

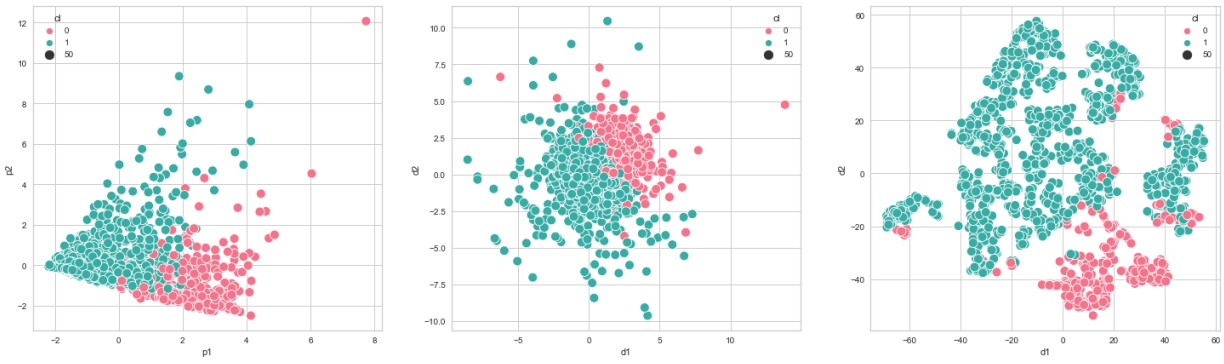
Out[84]:
```
KMeans
KMeans(max_iter=100, n_clusters=2)
```

In [85]:
```
mdsdf=data_mds.copy()
t_sne=data_tsne.copy()
```

In [86]:
```
Xs["cl"] = pca_['cl'] = mdsdf['cl'] = t_sne['cl'] = km.fit_predict(Xs)
```

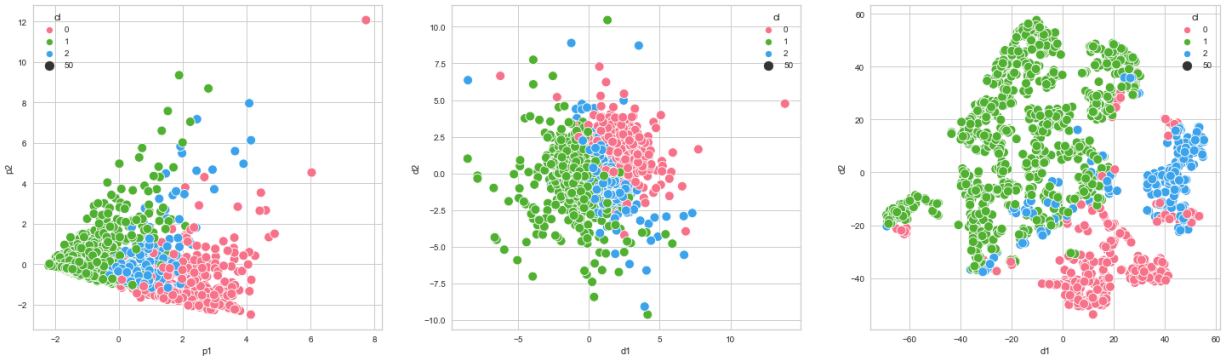
In [87]:
```
fig, ax = plt.subplots(1, 3,figsize=(25, 7))

sns.scatterplot(x='p1', y='p2',hue='cl', size=50, sizes=(120, 120), palette=sns.c
sns.scatterplot(x='d1', y='d2',hue='cl', size=50, sizes=(120, 120), palette=sns.c
sns.scatterplot(x='d1', y='d2',hue='cl', size=50, sizes=(120, 120), palette=sns.c
```

Out[87]: <AxesSubplot:xlabel='d1', ylabel='d2'>



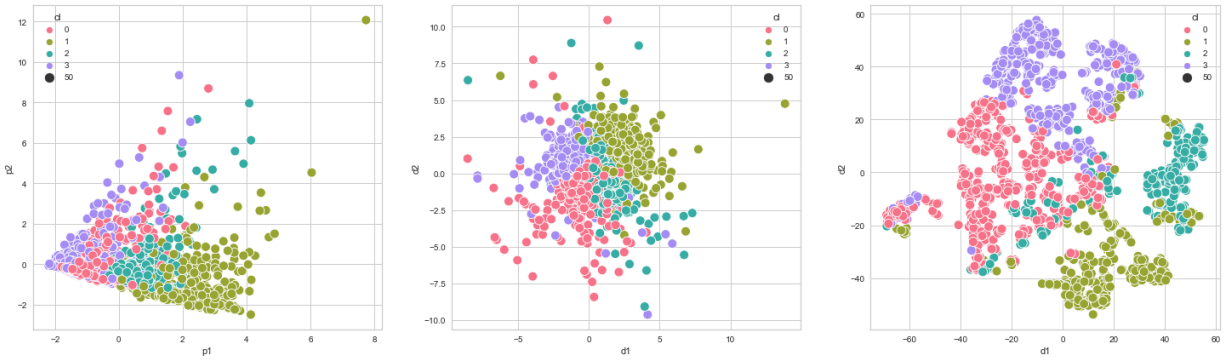
In [88]:
```
from sklearn.cluster import KMeans
km = KMeans(n_clusters = 3,max_iter=100)
km.fit(Xs)


Xs["cl"] = pca_['cl'] = mdsdf['cl'] = t_sne['cl'] = km.fit_predict(Xs)


fig, ax = plt.subplots(1, 3,figsize=(25, 7))

sns.scatterplot(x='p1', y='p2',hue='cl', size=50, sizes=(120, 120), palette=sns.c
sns.scatterplot(x='d1', y='d2',hue='cl', size=50, sizes=(120, 120), palette=sns.c
sns.scatterplot(x='d1', y='d2',hue='cl', size=50, sizes=(120, 120), palette=sns.c
```

Out[88]: <AxesSubplot:xlabel='d1', ylabel='d2'>



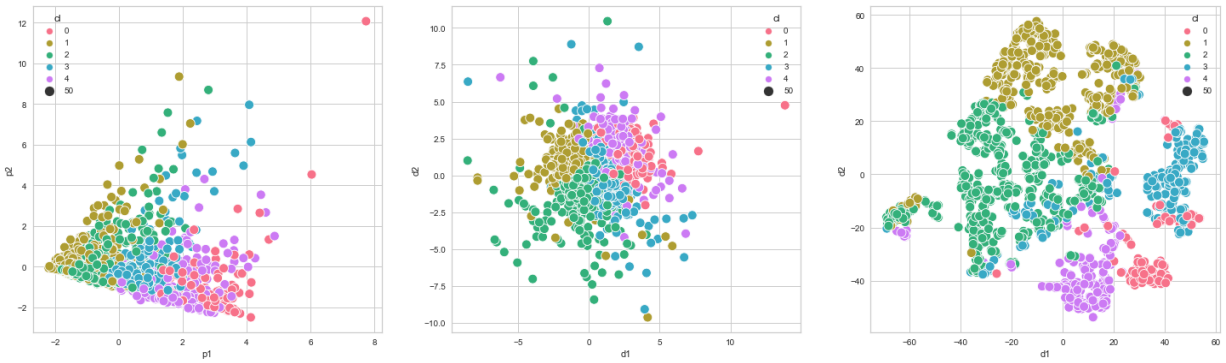
In [89]:
```
from sklearn.cluster import KMeans
km = KMeans(n_clusters = 4,max_iter=100)
km.fit(Xs)


Xs["cl"] = pca_['cl'] = mdsdf['cl'] = t_sne['cl'] = km.fit_predict(Xs)


fig, ax = plt.subplots(1, 3,figsize=(25, 7))

sns.scatterplot(x='p1', y='p2',hue='cl', size=50, sizes=(120, 120), palette=sns.c
sns.scatterplot(x='d1', y='d2',hue='cl', size=50, sizes=(120, 120), palette=sns.c
sns.scatterplot(x='d1', y='d2',hue='cl', size=50, sizes=(120, 120), palette=sns.c
```

Out[89]: <AxesSubplot:xlabel='d1', ylabel='d2'>



In [90]:
```
from sklearn.cluster import KMeans
km = KMeans(n_clusters = 5,max_iter=100)
km.fit(Xs)


Xs["cl"] = pca_['cl'] = mdsdf['cl'] = t_sne['cl'] = km.fit_predict(Xs)


fig, ax = plt.subplots(1, 3,figsize=(25, 7))

sns.scatterplot(x='p1', y='p2',hue='cl', size=50, sizes=(120, 120), palette=sns.c
sns.scatterplot(x='d1', y='d2',hue='cl', size=50, sizes=(120, 120), palette=sns.c
sns.scatterplot(x='d1', y='d2',hue='cl', size=50, sizes=(120, 120), palette=sns.c
```

Out[90]: <AxesSubplot:xlabel='d1', ylabel='d2'>



>>>>>>>>>>>>>>>>>>>> Cluster Densidad <<<<<<<<<<<<<<<<<<<<

In [91]:
```
import numpy as np
import pandas as pd

from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.decomposition import PCA
from sklearn.manifold import MDS,TSNE
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

from ipywidgets import widgets

import matplotlib.pyplot as plt
import seaborn as sns
import cufflinks as cf
import os
import random
import warnings
warnings.filterwarnings("ignore")

cf.go_offline()
os.getcwd()
```

Out[91]:
```
'C:\\Users\\mime1001\\Documents\\Diplomado\\Proyecto\\Entrega 3'
```

In [92]:
```
train = pd.read_csv (os.getcwd() + '/Data/train_data_imp.csv')


### Vamos a sacar las variables numericas
target_column = 'target'
drop_columns = [target_column]

#num_columns = [col for col in train.select_dtypes(['int64', 'float64']).columns
num_columns=['feature_81','feature_82','feature_169','feature_26','feature_66',
             'feature_128','feature_132','feature_156','feature_207']

cat_columns=['feature_307','feature_297','feature_108','feature_110','feature_476
             'feature_254','feature_263','feature_143','feature_298','feature_43'


train_num=train[num_columns]
train_num
```

Out[92]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	featu
0	4.0	0	0.0	0	0	0.00	0.805000	1.0244
1	34.0	0	0.0	0	0	3.57	1.150000	0.0000
2	36.0	0	0.0	0	0	0.00	0.000000	0.0000
3	48.0	0	0.0	0	0	0.00	2.723333	0.0000
4	169.0	123	0.0	109	5	3.57	11.041667	7.7846
...	...	...	...	...	...	...	...	
49122	144.0	97	0.0	34394	83	85.71	96.371667	0.0000
49123	88.0	48	0.0	12835	69	96.43	98.391667	9.1345
49124	16.0	15	0.0	61194	229	0.00	0.538333	9.4514
49125	149.0	16	0.0	57103	143	96.43	98.311667	0.0000
49126	2.0	0	0.0	0	0	0.00	0.000000	2.1450

49127 rows × 9 columns

In [93]:
```
# Vamos a samplear para PCA, MDS y TSNE los mismos datos
random.seed(7)
a=random.sample(range(1,len(train_num),1),1400)
a

train_sam=train_num.iloc[a]
train_sam
```

Out[93]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	featu
21223	191.0	0	0.000000	11311	37	100.00	100.000000	2.5973
9887	33.0	4	0.000000	16409	47	0.00	1.650000	0.0000
25876	28.0	0	0.000000	0	0	0.00	0.000000	9.0000
42660	99.0	0	0.000000	21396	54	28.57	57.403333	0.0000
3165	139.0	105	0.000000	34481	177	92.86	92.590000	0.0000
...	...	...	...	...	...	...	...	
32147	74.0	0	0.000000	31752	88	0.00	2.166667	8.0601
12933	25.0	0	0.000000	0	0	0.00	0.000000	9.0462
19767	128.0	115	0.000000	14060	33	0.00	3.260000	5.4053
8301	74.0	5	0.000000	135780	1070	7.14	3.393333	2.8535
2851	199.0	0	5.288333	5850	14	96.43	98.810000	4.3333

1400 rows × 9 columns

In [94]:
```
from sklearn.preprocessing import MinMaxScaler
sc_mm = MinMaxScaler()
Xs = pd.DataFrame(index = train_num.index, data = sc_mm.fit_transform(train_num),
Xs

Xs=Xs.iloc[a]
Xs
```

Out[94]:

	feature_81	feature_82	feature_169	feature_26	feature_66	feature_128	feature_132	featu
21223	0.731801	0.000000	0.181908	0.006803	0.009202	1.0000	0.978968	1.8332
9887	0.126437	0.031250	0.181908	0.009869	0.011689	0.0000	0.016153	0.0000
25876	0.107280	0.000000	0.181908	0.000000	0.000000	0.0000	0.000000	6.3523
42660	0.379310	0.000000	0.181908	0.012868	0.013429	0.2857	0.561961	0.0000
3165	0.532567	0.820312	0.181908	0.020738	0.044019	0.9286	0.906427	0.0000
...	...	...	...	...	...	...	...	
32147	0.283525	0.000000	0.181908	0.019096	0.021885	0.0000	0.021211	5.6889
12933	0.095785	0.000000	0.181908	0.000000	0.000000	0.0000	0.000000	6.3849
19767	0.490421	0.898438	0.181908	0.008456	0.008207	0.0000	0.031914	3.8151
8301	0.283525	0.039062	0.181908	0.081662	0.266103	0.0714	0.033220	2.0140
2851	0.762452	0.000000	0.239964	0.003518	0.003482	0.9643	0.967319	3.0585

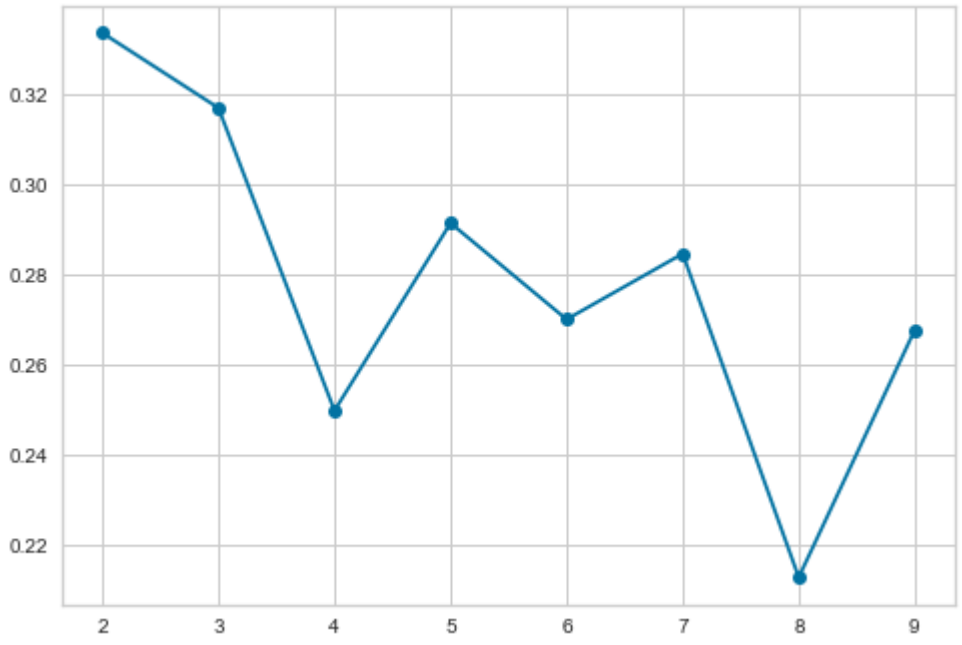
1400 rows × 9 columns

Gaussian Mixture

In [95]:
```
from sklearn.mixture import GaussianMixture
l = []
for k in range(2,10):
    print(k)

    km = GaussianMixture(n_components=k,covariance_type='spherical')

    km.fit(Xs)
    l.append((k,silhouette_score(Xs,km.predict(Xs))))
silueta = pd.DataFrame(l,columns=['k','silueta'])

plt.plot(silueta['k'],silueta['silueta'],marker='o')
```

```
2
3
4
5
6
7
8
9
```

Out[95]:
```
[<matplotlib.lines.Line2D at 0x1c8bcf11100>]
```



In [96]:
```
km.bic(Xs)
```

Out[96]:
```
-17716.92330721584
```

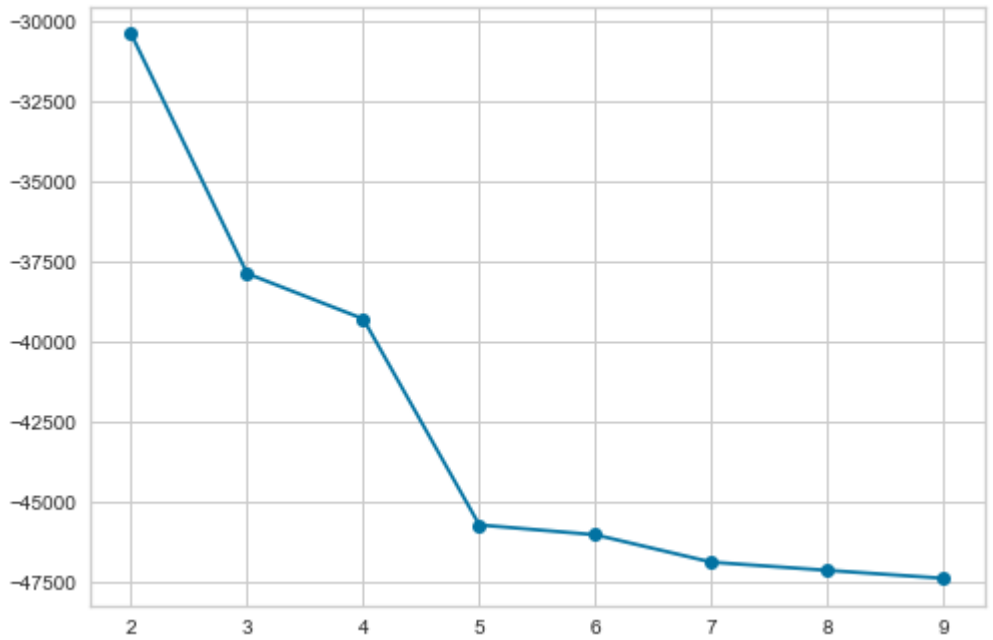
In [97]:
```
km.aic(Xs)
```

Out[97]:
```
-18230.857603744967
```

In [98]:
```
l_bic = []
l_aic=[]
for k in range(2,10):
    km = GaussianMixture(n_components=k)
    km.fit(Xs)
    l_bic.append((k,km.bic(Xs)))
    l_aic.append((k,km.aic(Xs)))
aic = pd.DataFrame(l_aic,columns=['k','aic'])
bic = pd.DataFrame(l_bic,columns=['k','bic'])
```

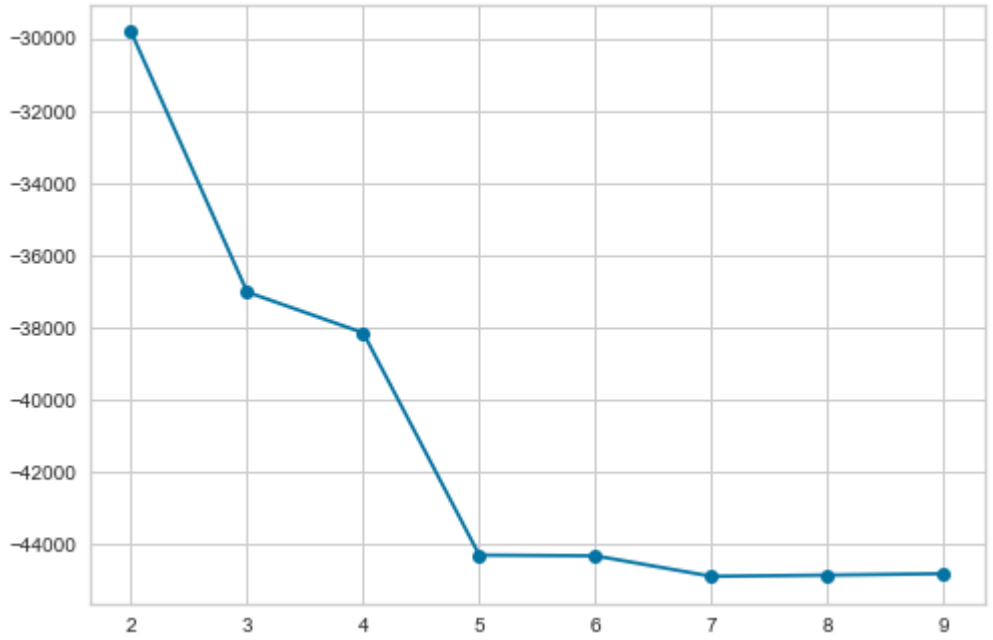
In [99]:
```
plt.plot(aic['k'],aic['aic'],marker='o')
```

Out[99]:
```
[<matplotlib.lines.Line2D at 0x1c8b9c8afd0>]
```



In [100]:
```
plt.plot(bic['k'],bic['bic'],marker='o')
```

Out[100]:
```
[<matplotlib.lines.Line2D at 0x1c8b82fdc70>]
```



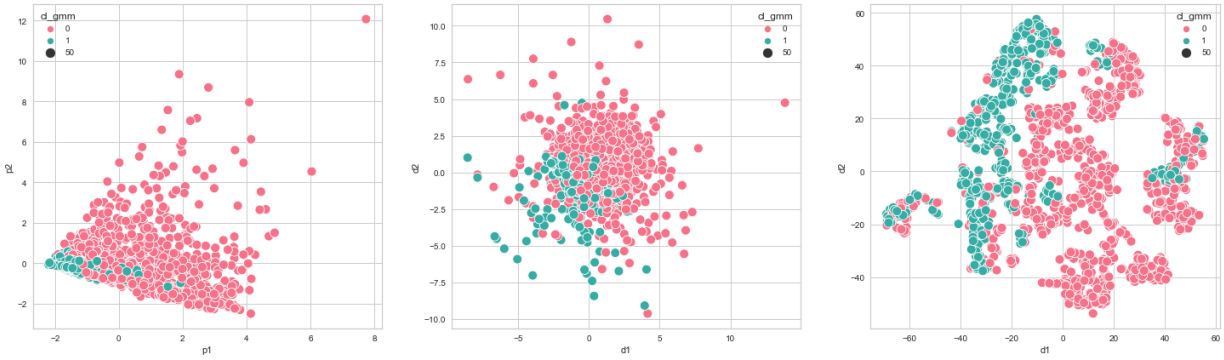
In [101]:
```
gmm=GaussianMixture(n_components=2)
gmm.fit(Xs)
```

Out[101]:
```
GaussianMixture(n_components=2)
```

In [102]:
```
Xs["cl_gmm"] = pca_['cl_gmm'] = mdsdf['cl_gmm'] = t_sne['cl_gmm'] = gmm.fit_predi
```

In [103]:
```
fig, ax = plt.subplots(1, 3,figsize=(25, 7))

sns.scatterplot(x='p1', y='p2',hue='cl_gmm', size=50, sizes=(120, 120), palette=s
sns.scatterplot(x='d1', y='d2',hue='cl_gmm', size=50, sizes=(120, 120), palette=s
sns.scatterplot(x='d1', y='d2',hue='cl_gmm', size=50, sizes=(120, 120), palette=s
```

Out[103]:
```
<AxesSubplot:xlabel='d1', ylabel='d2'>
```



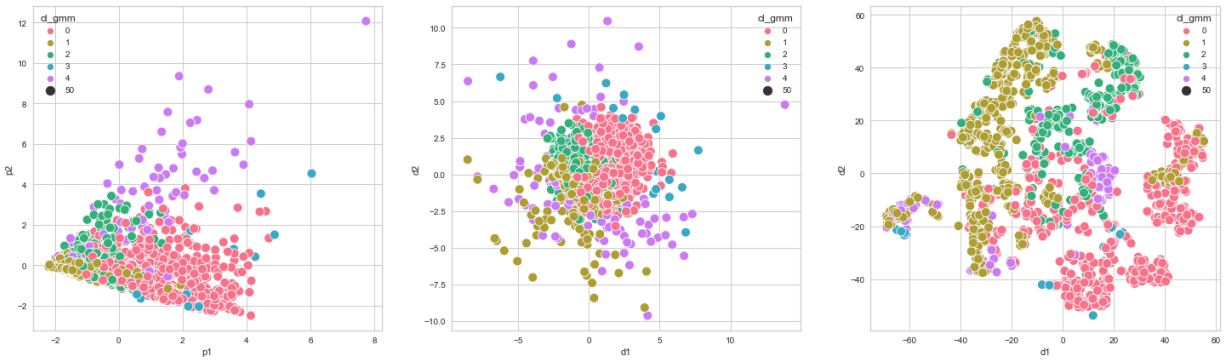
In [104]:
```
gmm=GaussianMixture(n_components=5)
gmm.fit(Xs)
```

Out[104]:
```
GaussianMixture
GaussianMixture(n_components=5)
```

In [105]:
```
Xs["cl_gmm"] = pca_['cl_gmm'] = mdsdf['cl_gmm'] = t_sne['cl_gmm'] = gmm.fit_predi
```

In [106]:
```
fig, ax = plt.subplots(1, 3,figsize=(25, 7))

sns.scatterplot(x='p1', y='p2',hue='cl_gmm', size=50, sizes=(120, 120), palette=s
sns.scatterplot(x='d1', y='d2',hue='cl_gmm', size=50, sizes=(120, 120), palette=s
sns.scatterplot(x='d1', y='d2',hue='cl_gmm', size=50, sizes=(120, 120), palette=s
```

Out[106]:
```
<AxesSubplot:xlabel='d1', ylabel='d2'>
```



Type *Markdown* and LaTeX: α^2

In [ ]:
```
```

In [ ]:
```
#================================================ FIN ============================
```

In [ ]:
```

```