Especialización en Ciencia de Datos

TRABAJO FINAL INTEGRADOR

DETECCIÓN DE OPERACIONES DURANTE LA PERFORACIÓN DE UN POZO

Alumno: Cristian Lissauer

Tutor: Dra. Leticia Gómez

Ciudad de Buenos Aires, Enero 2021

Índice de Contenidos

1. Introducción

Las compañías petroleras vuelcan la información obtenida durante las perforaciones en sistemas de software especializados para registrar dicha información.

En la actualidad una compañía petrolera tiene la necesidad de estandarizar en esos sistemas la carga de las operaciones de perforación y que las mismas dejen de ser a criterio y de forma subjetiva dependiendo de cada Company Man (Supervisor de Pozo), quién carga manualmente el detalle de operaciones.

Lo que se busca en este estudio es encontrar las operaciones definidas en la compañía para la planificación y ejecución de la perforación del pozo basándose en los datos propios e históricos arrojados por los equipos de perforación.

Los datos obtenidos durante la perforación ya se encuentran sensorizados en cada uno de los equipos de perforación los cuales serán necesarios para el desarrollo de la solución. Esta incluirá la recolección de esos datos, como también la revisión y corrección de los mismos. Se analizarán distintas variantes de algoritmos de machine learning para poder ver con exactitud cuál es el mejor algoritmo que se adecua a nuestro dominio del problema y proceder a la selección del algoritmo óptimo para el modelo de datos como también su desarrollo para poder clasificar las operaciones de perforación en tiempo real y con gran exactitud sin intervención humana.

2. Estado del Arte

Las compañías petroleras necesitan diariamente registrar sus actividades realizadas durante la perforación de cada uno de los pozos que les pertenece su explotación.

Para obtener esa información se suelen utilizar sensores o formas de medición que permiten registrar datos obtenidos durante la operación.

Una vez obtenidos los datos, hay una persona encargada de cargar esos datos en el software de carga de la compañía determinando cuales son las operaciones que se están llevando a cabo en relación a esos datos.

El tiempo y la complejidad que conlleva el trabajo en sí de la perforación más la carga de todos esos datos en el sistema con sus operaciones provoca que en la actualidad no se registren correctamente en tiempo y forma las actividades u operaciones que se realizan durante la perforación de un pozo de Oil & Gas dado que se analizan en forma manual.

Durante las operaciones de Perforación, el Company Man (Supervisor de Pozo) que como dijimos carga manualmente el detalle de operaciones, en muchos casos, la carga la hace al finalizar el día, declarando las tareas que se realizaron y la duración aproximada de cada una de ellas.

Contando con información sensorizada en los equipos de perforación, se quiere utilizar esos datos para agilizar la carga de información de las operaciones, con mayor precisión, y que esto contribuya a mejorar la planificación de perforación de futuros pozos además de poder comprobar el estado de cada uno de los pozos en tiempo real lo cual es imposible en estos momentos.

Los datos de las actividades se informan mediante un proveedor en formato **WITSML** mediante distintos sensores (profundidad del hoyo, posición del Trepano, fluido, presión, etc).

Una perforación dura entre 15 y 30 días aproximadamente dependiendo de la complejidad del yacimiento. Hay aproximadamente 17 equipos perforando en simultáneo y se distribuyen de forma que cada equipo perfore 1 solo pozo.

La información de los sensores que tiene cada uno de los equipos se obtiene cada 5 segundos. Hoy en día la calidad de los datos de los sensores no está siendo revisada o controlada.

Hay veces que solo se cuenta con la información por momentos. Se puede perder la información de los sensores si hay problemas de comunicación entre los sensores y el proveedor, o entre el proveedor y la entrega de información de WITSML. Sí el sensor siguió tomando mediciones, al momento de recomponer la comunicación, la información se envía.

Los datos que se disponen para obtener información diariamente, se obtienen en tiempo real de las operaciones que se realizan en la perforación del pozo. Estas operaciones se llaman micro actividades. Además de las micro actividades, recibimos datos de los sensores de medición (posición de bloque, profundidad de pozo, profundidad de trépano, entre otras).

Con la información de micro actividades, los sensores y el Plan de pozo, se determinan las macro actividades (Perforación, Cementación, Calibrando, Casing Running, Perfilando, BOP, Superficie) utilizando algoritmos procedurales.

La información de la macro actividad es cargada en el detalle de operaciones de la aplicación de la compañía, que se encuentra en cada parte diario, mediante un proceso que dispara el Company Man de forma manual como ya lo habíamos comentado.

Con la carga del detalle de operaciones generada a partir de la información de las mediciones de los sensores, se minimiza la carga detallada de las actividades, acotando posibles errores e inconsistencias, reduciendo los tiempos de carga y con información más precisa de lo que ocurre en la operación. Además, la información será utilizada para mejorar el plan de perforación de futuros pozos.

El principal objetivo del estudio es determinar las macro actividades mediante algoritmos de *machine learning*, aprovechando la información ya cargada en la aplicación de la compañía. De esta forma se podría utilizar otros sensores que de manera procedural no se pudieron incorporar a la determinación de la macro actividad.

Actualmente existen ciertos enfoques y soluciones para encarar este tipo de problema, tal como se encuentra publicado por *[1]*.

En este trabajo se presentan técnicas de *machine learning* para descubrir y entender los patrones que ocurren en los datos que arrojan los sensores durante una perforación y propone encarar el problema de forma jerárquica para reconocer las operaciones de perforación partiendo el tiempo total de perforación en una serie de estados de operaciones predefinidos. El proceso, según dice, no solo logra dar soporte a los ingenieros de perforación para que puedan medir la performance del proceso de perforación sino también para que puedan identificar patrones en los datos que presuntamente puedan identificar crisis o problemas.

Para encarar la solución proponen hacerlo en dos fases. En la primera fase, 5 estados principales describiendo el estado operacional del equipo de perforación van a ser determinados utilizando los datos de los sensores. En la segunda fase, esos estados principales van a ser combinados a una serie de estados de operaciones de perforación. Los estados principales pueden ser considerados como una capa intermedia entre los datos de los sensores y los estados de alto nivel que describen las operaciones de perforación. Esos 5 estados físicos usados en la capa intermedia están relacionados con la rotación y movimiento de la sarta de perforación, la circulación de lodo, la perforación en sí misma y el estado de suspensión del gancho de la sarta de perforación. Todos esos estados son binarios (sí/no) con excepción de la sarta de perforación que tiene 3 valores (arriba/abajo/estático). Para reconocer esos estados principales se utilizó un clasificador que utiliza una red neuronal dedicada que fue entrenada con los datos

de los sensores. Como arquitectura de la red, el *perceptron* completo fue aplicado en combinación con aprendizaje en paralelo. El crecimiento automático de la red neuronal fue utilizado para *matchear* la complejidad del modelo con la complejidad de la clasificación particular del problema y para prevenir que no haya *over fitting.*

Además, un método de selección fue utilizado para identificar los datos de los sensores necesarios para reconocer los estados particulares. Para este enfoque fueron utilizados datos de tiempo real y los resultados mostraron que pueden clasificar estados de perforación con mucha precisión. La performance de los clasificadores fue evaluada utilizando *cross-validation* y el promedio de la correcta tasa de clasificación fue superior al 99%, tanto para el set de entrenamiento como para el set de pruebas.

Un estudio también para lo que es *data analytics* para la clasificación de estados operacionales de la perforación es [6], donde habla de proponer un coeficiente de correlación extendido que se aplica a las categorías K, y se muestra que esta medida es altamente aplicable para evaluar la predicción de la estructura secundaria de moléculas RNA en los casos en que algunos pares predichos entran en la categoría "desconocido" debido a la falta de confiabilidad en los pares pronosticados. o residuos no apareados.

Este trabajo proporciona resultados de pruebas realizadas para la identificación de los mejores clasificadores de performance para la detección de estados operacionales en operaciones de perforación. Se realizaron pruebas en múltiples escenarios para la detección de los estados operacionales con un equipo de perforación con varios pozos involucrados. Los datos de perforación son extremadamente desafiantes debido a su naturaleza no lineal y estocástica, a pesar del ruido incrustado en ellos y el desequilibrio constante. Sin embargo, existe la posibilidad de desplegar estos robustos clasificadores para superar tales desafíos y lograr una buena detección automatizada de estados. Tres clasificadores con los mejores índices de clasificación de los estados operacionales de perforación fueron identificados en este estudio.

El texto también nos explica que el proceso de perforación de un pozo implica la gestión de operaciones de alta complejidad en plataformas de perforación. Situaciones críticas como "Kicks", "Perdida de Fluido" o "Tubería atascada" pueden ocurrir durante las operaciones de perforación. Tales condiciones se alcanzan gradualmente siguiendo varias etapas de criticidades en el tiempo. Por lo tanto, es importante que aquellas etapas de las operaciones sean detectadas y controladas durante los procesos de perforación. Una forma de lograrlo es automatizar la detección de estados operativos de

perforación. Eso implica la ruptura de un proceso de perforación en diez etapas exclusivas y bien definidas 1) Drilling Rotary (DrlRot); 2) Drilling sliding (DrlSld); 3) Clean Downwards (CleanDN); 4) Clean Upwards (CleanUP); 5) Wash Upwards (WashUP); 6) Wash Downwards (WashDN); 7) Move in hole (MoveDN); 8) Move out of hole (MoveUP); 9) Circulation on (CirclHL); y 10) Make Connection (MakeCN). Las operaciones de perforación fueron detectadas con éxito en una ejecución de perforación utilizando técnicas de machine learning con cinco estados principales adicionales [1]. Además, las redes neuronales se ajustaron para hacer frente a conjuntos de datos desequilibrados con el fin de obtener un buen rendimiento en la clasificación de las operaciones de perforación en un pozo dado [2]. Sin embargo, el conocimiento de los datos etiquetados para el entrenamiento de la red, se asumió que estaban disponibles durante el proceso de perforación. Por lo tanto, el desafío es considerar un escenario operacional real que considere un plan de perforación en múltiples pozos cuando los datos etiquetados estén disponibles después de la perforación, al menos en un pozo del equipo.

En el estudio se propone un marco de trabajo para la selección de los clasificadores de operaciones con mejor rendimiento. Los clasificadores se entrenan utilizando una parte de los datos etiquetados disponibles de las operaciones, es decir, la capacitación se realiza para un pozo dado, mientras que la prueba se realiza en otros pozos para la detección de operaciones invisibles.

Para la parte del análisis de datos se generan dos tipos de datos de perforación: datos de mediciones de sensores (series de tiempo); y datos creados por expertos en perforación como observaciones, denominadas etiquetas de las operaciones.

Para los datos de series tiempo, los datos de medición son generados por dispositivos de detección. Los datos exhiben un complejo comportamiento que llevó a un mayor análisis de datos para seleccionar los clasificadores adecuados.

La complejidad de las series de tiempo de perforación (comportamiento de los datos) requirió dos pruebas en el proceso de clasificación: ensayos de linealidad y normalidad.

Para las etiquetas de perforación, los datos de las operaciones son generados por ingenieros de perforación como observaciones en tiempo real, utilizando evaluaciones de conocimiento e informes matutinos creados por los expertos. Estos últimos se llenan cuando las fases de las operaciones de perforación se completan y pasan a los siguientes equipos de perforación operativos. Las etiquetas de las operaciones son, por consiguiente, ruidosas y subjetivas. El análisis estadístico de 9 pozos mostró que

faltaban del 15% al 25% de las etiquetas de las operaciones para cada pozo. Además, las etiquetas de las operaciones ocurrieron a diferentes duraciones y frecuencias, es decir, son estadísticamente desequilibradas.

Después de haber hecho todo un análisis de datos anterior basándose en lo comentado anteriormente, ciertas referencias mencionadas en su estudio y la experiencia de los autores con clasificación de datos complejos, se seleccionaron 8 algoritmos de *machine learning*. Estos incluyen: 1) k-Nearest-Neighbour (kNN), 2) Support Vector Machines (SVM), 3) Linear Discriminant Analysis (LDA) 4) Echo State Network (ESN), 5) Random Forest (RF), 6) AdaBoostM2, 7) RusBoost y 8) Subspace. Cada uno de los algoritmos se evaluó usando micro-promedio y macro promedio de medidas F [2]; junto con el coeficiente de correlación Matthews (MCC) [7] por sus respectivas actuaciones generales. Las tasas de clasificación correcta (CCR) fueron adoptado para la evaluación de las operaciones individuales. Cuanto mayor sea la medida F, mayor será la tasa de clasificación. El micro-promedio de la medidad F proporciona pesos iguales a cada etiqueta y tiende a estar dominado por el rendimiento del clasificador en clases comunes. El macro promedio de la medida F proporciona pesos iguales a cada clase, independientemente de su frecuencia. Es influenciado más por el rendimiento del clasificador en clases raras. Ambos puntajes de medidas se usan para analizar qué tan bien se desempeñan los clasificadores bajo clases comunes y raras. MCC resume la matriz de confusión en un solo valor y se considera como una buena medida para problemas con clases desequilibradas. Devuelve un valor entre -1 y 1, donde 1 es una predicción perfecta, 0 no es mejor que una predicción aleatoria y -1 indica un total desacuerdo entre predicción y observación. Los ocho clasificadores seleccionados fueron entrenados usando mediciones de sensores de un pozo determinado. Las pruebas se realizaron posteriormente en otros dos pozos del mismo equipo.

Se consideraron diez mediciones de sensores: 1) Posición del bloque; 2) Profundidad de trepano; 3) Profundidad del agujero; 4) Peso del trepano; 5) Flujo de lodo; 6) Presión de la bomba; 7) Tasa de penetración; 8) Torque rotativo; 9) Carga de gancho y; 10) Velocidad de rotación. Seis características adicionales también fueron consideradas: 1) Profundidad del agujero - Profundidad del trepano; 2) Profundidad del agujero + posición del bloque; 3) Profundidad del trepano + Posición del bloque; 4) Torque rotativo * Velocidad de rotación; 5) Presión de la bomba * Flujo de lodo y; 6) Velocidad de penetración * Peso del trepano.

Se diseñaron tres escenarios experimentales de acuerdo con diversas utilizaciones de la cantidad de datos etiquetados de un determinado pozo para el entrenamiento de clasificadores: 1) el 100% de los datos se consideran para el entrenamiento; 2) 30% de los datos se consideran para el entrenamiento usando muestreo uniforme sin reemplazo y; 3) Se considera el 30% de los datos para el entrenamiento utilizando un muestreo híbrido. Se consideraron estos escenarios para evaluar la posibilidad de reducir conjuntos de entrenamiento sin perder en la precisión de clasificación en los conjuntos de prueba. La investigación sobre la sensibilidad de los clasificadores a varios conjuntos sub muestreados y la comparación de los intervalos de confianza en los escenarios 2) y 3) se realizó utilizando diez muestras diferentes de Monte Carlo que se extrajeron respectivamente para cada esquema de muestreo. Cada algoritmo se ajustó para lograr el mejor rendimiento.

Tres algoritmos como RF, AdaBoostM2 y RUSBoost muestran el mejor rendimiento para los diferentes conjuntos de entrenamiento. Lograron un rendimiento similar de acuerdo con los tres criterios de evaluación: Las medidas F1 y F2 alcanzaron valores superiores al 80% y 55% respectivamente; mientras MCC estaba arriba de 0.7 para todos estos algoritmos. Los algoritmos kNN y SubSpace su rendimiento fue bastante pobre y se puede entender que el volumen de los conjuntos de datos de entrenamiento puede ser reducido en un tercio sin reducir significativamente el rendimiento de los clasificadores.

El estudio concluye diciendo que sirvió para realizar un estudio exhaustivo para la selección de los clasificadores más performantes para la detección de estados operativos en operaciones de perforación.

Se establecieron estrategias para filtrar los clasificadores de menor rendimiento y mantener aquellos que lidiaron eficientemente con datos complejos de operaciones de perforación y múltiples estados clasificación. El conocimiento previo sobre los estratos geofísicos de la plataforma operativa puede potencialmente ayudar en nuevas inferencias para mejorar el rendimiento. Los clasificadores RF, AdaBoostM2 y RUSBoost se encontraron altamente confiables para lograr detección automatizada en tiempo real de estados de perforación operacionales. Se proponen como los mejores clasificadores para construir los sistemas de información de apoyo a la decisión de próxima generación para lograr operaciones de perforación más seguras en plataformas industriales.

Otro trabajo desarrollado para algoritmos de clasificación tanto para las micro actividades como para las macro actividades es *[2]*, que se encuentra también en las referencias bibliográficas. Este texto estudia un número de medidas que caracterizan la dificultad de la clasificación de un problema. El texto muestra o sugiere como describir el dominio de competencia de un clasificador. Esto puede guiar la selección de forma estática y dinámica de clasificadores para problemas específicos.

Tomando en cuenta también las otras referencias bibliográficas como *[3]*, *[4]* y *[5]*, las utilizaremos para determinar los algoritmos y clasificadores que sean más convenientes para el dominio de nuestro problema. Para eso deberemos realizar una gran cantidad de pruebas con distintos sets de datos para determinar cuáles deben ser los más adecuados.

3. Presentación del Problema

En la actualidad, las operaciones de perforación de pozos petroleros no se encuentran estandarizadas en los sistemas de la compañía. El Company Man o Supervisor de Pozo es quién carga manualmente el detalle de operaciones y en muchos casos se realiza al finalizar el día, declarando las tareas que se ejecutaron y la duración aproximada de cada una de ellas. Esto conlleva no contar con los tiempos exactos de cada operación los cuales son importantes para la planificación y mejora de perforación los pozos futuros.

4. Justificación del Estudio

Si bien existen trabajos que determinan las operaciones de perforación en los pozos, lo que se busca en este estudio es encontrar las operaciones definidas en la compañía que son clave para la planificación y ejecución de la perforación del pozo basándose en los datos propios e históricos que ya se encuentran sensorizados.

Otra de las necesidades es estandarizar el criterio de carga entre todos los equipos de perforación dejando de lado el criterio del Supervisor o Company Man para la definición de las operaciones.

Finalmente, otro beneficio, por el cual se realiza el estudio, es que al obtener las operaciones en tiempo real basada en información sensorizada, permite a nivel gerencial conocer de forma online y precisa las operaciones que se están realizando en

cada uno de los pozos sin necesidad de esperar que un Supervisor o Company Man cargue la información en el sistema corporativo.

5. Alcances del Trabajo y Limitaciones

El alcance de nuestro estudio y trabajo incluirá en primer lugar la recolección de datos, en la cual involucra obtener toda la información en tiempo real arrojada por los equipos y los sensores disponibles en todo el proceso de perforación. Como segundo punto incluirá la revisión y corrección de datos. Este paso es muy importante ya que para que nuestro algoritmo clasificador de actividades necesitara que la información tenga datos fehacientes para poder realizar una clasificación de forma adecuada. Finalmente se analizarán distintas variantes de algoritmos de *machine learning* para poder ver con exactitud cuál es el mejor algoritmo que se adecua a nuestro dominio del problema. Luego, se procederá a la selección del algoritmo óptimo para el modelo de datos como también el desarrollo del algoritmo de *machine learning* elegido para clasificación de las operaciones.

6. Objetivos

6.1. Objetivo General

Desarrollar una solución de software basada en ciencia de dato que permita sistematizar las actividades de las operaciones diarias en tiempo real que se realizan durante la perforación de un pozo de petróleo y gas.

6.2. Objetivos Específicos

- Recolección de Datos de los sensores
- Normalización de los datos a través de procesos de ETL
- Desarrollo del algoritmo de machine learning
- Validación y visualización de los resultados
- Desarrollo de una prueba conceptual de la solución

7. Metodología Aplicada

A los fines de mostrar una prueba conceptual del estudio, desarrollaremos un algoritmo de *machine learning* sobre la base de datos obtenidos de sensores.

Los datos a utilizar en el modelo se obtendrán a través de la recolección en tiempo real arrojada por los equipos y los sensores disponibles en el proceso de perforación, a los cuales se les aplicará un proceso de limpieza a través de varios procesos de ETL, para corregir todo tipo de problemas que puedan venir con los datos de los sensores.

Con dichos datos limpios, se nutrirá un *warehouse*, que será la base normalizada para el desarrollo del algoritmo de *machine learning*.

Finalmente, se procederá a la validación de los resultados, que quedarán clasificados por tiempo y por macro actividad (la cual es la actividad determinada por el algoritmo de *machine learning*), y a la visualización de los mismos, a través del desarrollo de un *plugin* para la aplicación de la compañía.

8. Desarrollo del Modelo

8.1. Extracción de los Datos

Para la recolección de datos del proveedor con información de los sensores se utilizó un web service de tipo **SOAP** desarrollado en **Java**.

A continuación, se enumera y se explica brevemente cada uno de los campos que incluye el set de datos obtenidos por los sensores en tiempo real:

- WELL_NAME: Nombre del pozo.
- DATE_TIME: Fecha y hora que los sensores tomaron los datos.
- ACTIVITY_CODE: Código de la micro actividad que se esta llevando acabo
- MACROACTIVITY_CODE: Código de la macroactividad que define la operación
- C_DMEA: Profundidad medida del pozo (MD)
- BIT_DEPTH: Profundidad del taladro
- TD_TORQUE: Torque
- STP_PRS_1: Presión
- FLOW_IN: Caudal de entrada
- BLOCK_POS: Altura de la posición del bloque
- HOOKLOAD_MAX: Peso del gancho
- CIA_PRESION_DIRECTA: Presión de la operación de Cementación
- CIA_DENSIDAD_FLUIDO: Densidad del fluido de la operación de Cementación
- CIA_CAUDAL: Caudal de la operación de Cementación

- CIA_VOLUMEN_ACUMULADO: Volumen acumulado de la operación de Cementación
- CIA_TIEMPO: Duración de la operación de Cementación
- WIRELINE_PROFUNDIDAD: Profundidad del Wireline
- WIRELINE_TENSION_DIFERENCIAL: Tensión diferencial del Wireline
- WIRELINE_TENSION_TOTAL: Tensión total del Wireline
- WIRELINE_TIEMPO: Duración de la operación de Wireline
- WIRELINE_VELOCIDAD: Velocidad del Wireline
- WIRELINE_VOLTAJE: Voltaje del Wireline
- WIRELINE_INTENSIDAD: Intensidad del Wireline
- CIA_VOLUMEN_POR_ETAPAS: Es el volumen de la cementación en cada etapa
- FINALIZACION_PROF_FASE_1: Profundidad de finalización de la fase 1
- REFERENCIA_PROF_FASE_1: Profundidad de referencia de la fase 1
- FINALIZACION_PROF_FASE_2: Profundidad de finalización de la fase 2
- REFERENCIA_PROF_FASE_2: Profundidad de referencia de la fase 2
- FINALIZACION_PROF_FASE_3 Profundidad de finalización de la fase 3
- REFERENCIA_PROF_FASE_3 Profundidad de referencia de la fase 3
- FINALIZACION_PROF_FASE_4 Profundidad de finalización de la fase 4
- REFERENCIA_PROF_FASE_4 Profundidad de referencia de la fase 4
- FINALIZACION_PROF_FASE_5 Profundidad de finalización de la fase 5
- REFERENCIA_PROF_FASE_5 Profundidad de referencia de la fase 5

Toda esa información se recibe en formato **WITSML** casi en tiempo real, ya que los sensores arrojan información cada 5 segundos. Luego, esa información se volcó en una base de datos de **SQL Server**.

En el Anexo 1 se puede consultar parte del código Java de este proceso.

8.2. Proceso de ETL Aplicado

Para realizar todos los procesos de ETL se utilizó **Python** con librerías **Pandas y Numpy**, con librerías de visualización **MatPlotLib y Seaborn**.

Antes de importar los datos a la base, como parte necesaria para tener un dataset válido, necesitamos convertir el tipo de datos de algunos campos y especificar los separadores necesarios, ya que por default Pandas con la función “read_csv” no lo hace correctamente. El dataset que utilizamos son datos de tiempo real y contiene datos temporales, con lo cual nuestro problema es una serie de tiempo dentro de un problema de aprendizaje supervisado ya que tenemos los datos en dos coordenadas (x, y).

En primer lugar especificamos la columna que tiene el dato del tiempo. En segundo lugar llamamos a la función “read_csv” indicandole el nombre del archivo, separador de campos, separador decimal y el tipo de campo de algunos campos que causan inconvenientes para que se reconozca su tipo de dato, tal como por ejemplo: “WELL_NAME”, “ACTIVITY_CODE” y “MACROACTIVITY_CODE”. También le indicamos el campo DATE_TIME al parámetro “parse_dates” para que pueda interpretar el campo como fecha.

El proceso se inicia con la verificación de la información y tipo de campos del dataset. Luego, se transforma el tipo de dato de la columna ACTIVITY_CODE a tipo numérico, se completan los valores NA con cero, para poder convertir la columna de Float a Integer. Finalmente, se setea el índice en el campo DATE_TIME

En el Anexo 2 se pueden ver los procesos utilizados.

8.3. Limpieza del Dataset

Nos quedaremos solamente con las filas en las cuales tenga una macro actividad definida, ya que es el parámetro que queremos predecir, con lo cual empezamos con la limpieza del dataset eliminando todas las filas que no tengan macro actividad.

En primera instancia, eliminaremos la columna WELL_NAME, ya que todos los datos pertenecen al mismo pozo y además es necesario ya que es una columna de tipo string lo que no sirve para incluir en el modelo. Luego, eliminamos todas las columnas nulas.

Al finalizar esta etapa, nos quedan 258414 observaciones para 27 variables, más la variable categórica MACROACTIVITY_CODE, siendo un total de 28 variables, a saber:

- ACTIVITY_CODE
- MACROACTIVITY_CODE
- C_DMEA
- BIT_DEPTH
- TD_TORQUE
- STP_PRS_1
- FLOW_IN
- BLOCK_POS
- HOOKLOAD_MAX
- CIA_DENSIDAD_FLUIDO
- CIA_CAUDAL
- CIA_VOLUMEN_ACUMULADO

- CIA_TIEMPO
- WIRELINE_PROFUNDIDAD
- WIRELINE_TENSION_DIFERENCIAL
- WIRELINE_TENSION_TOTAL
- WIRELINE_TIEMPO
- WIRELINE_VELOCIDAD
- CIA_VOLUMEN_POR_ETAPAS
- FINALIZACION_PROF_FASE_1
- REFERENCIA_PROF_FASE_1
- FINALIZACION_PROF_FASE_2
- REFERENCIA_PROF_FASE_2
- FINALIZACION_PROF_FASE_3
- REFERENCIA_PROF_FASE_3
- FINALIZACION_PROF_FASE_4
- REFERENCIA_PROF_FASE_4
- FINALIZACION_PROF_FASE_5

Adicionalmente tenemos en el índice un timestamp también obtenido del *dataset*.

A continuación, verificamos la cantidad y tipo de registros de la variable a predecir "MACROACTIVITY_CODE". En la Figura 1 se puede visualizar la cantidad obtenida para cada categoría.

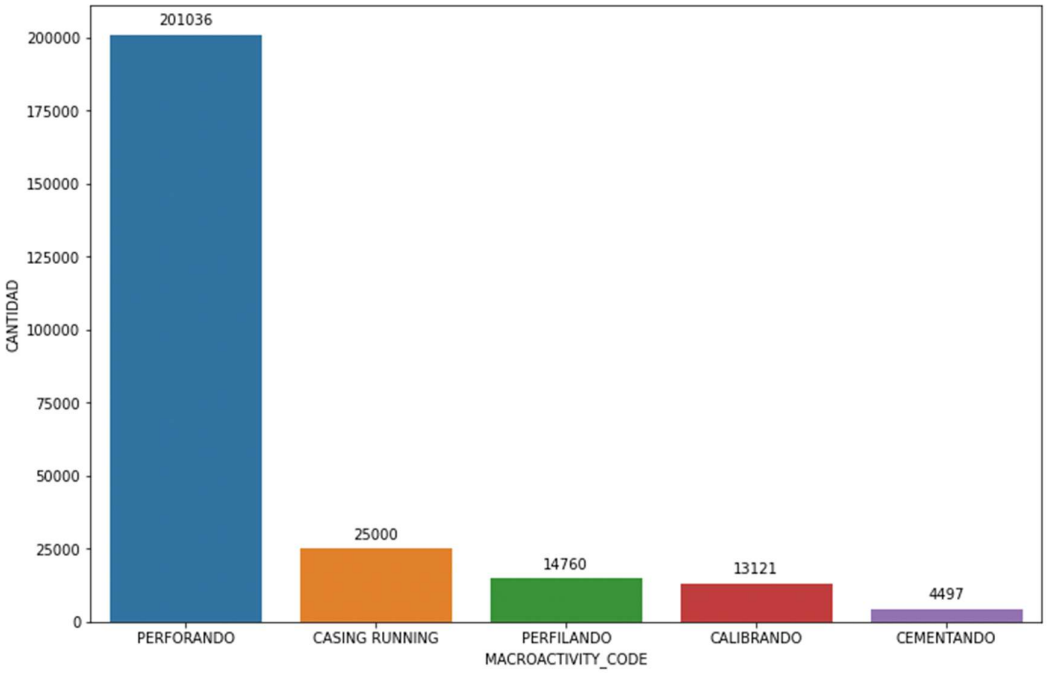

Fig. 1 - Cantidad y tipo de registros de la variable a predecir MACROACTIVITY_CODE.

Para continuar el proceso de limpieza, analizamos los valores que pueden tomar las variables y confeccionamos un melt del dataset para luego poder visualizarlo en un *boxplot* combinado con el *stripplot.* En la Figura 2 se puede visualizar el resultado obtenido.

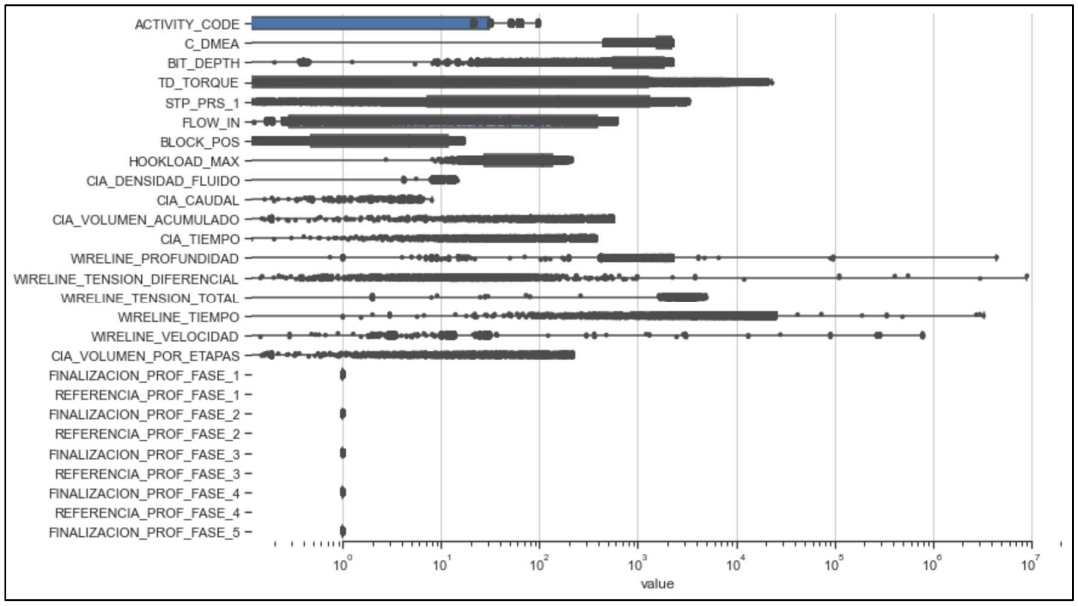

Fig. 2 – Visualización del análisis de variables del dataset

Del análisis realizado, se desprende que los campos “Finalización” y “Referencia Prof Fase 1, 2, 3, 4 y 5” no aportan información, ya que tienen valor constante 0 y 1, con lo cual podemos eliminar esas 9 columnas.

Para terminar con la limpieza final del dataset, realizamos el encoding de la variable categórica “MACROACTIVITY_CODE”:

MACROACTIVITY_CODE_LABEL	Cantidad
0	13121
1	25000
2	4497
3	14760
4	201036

Finalmente, antes de pasar a aplicar el modelo, graficamos un heatmap para verificar la correlación entre las distintas variables del dataset. En la Figura 3 se puede ver el resultado de la correlación.

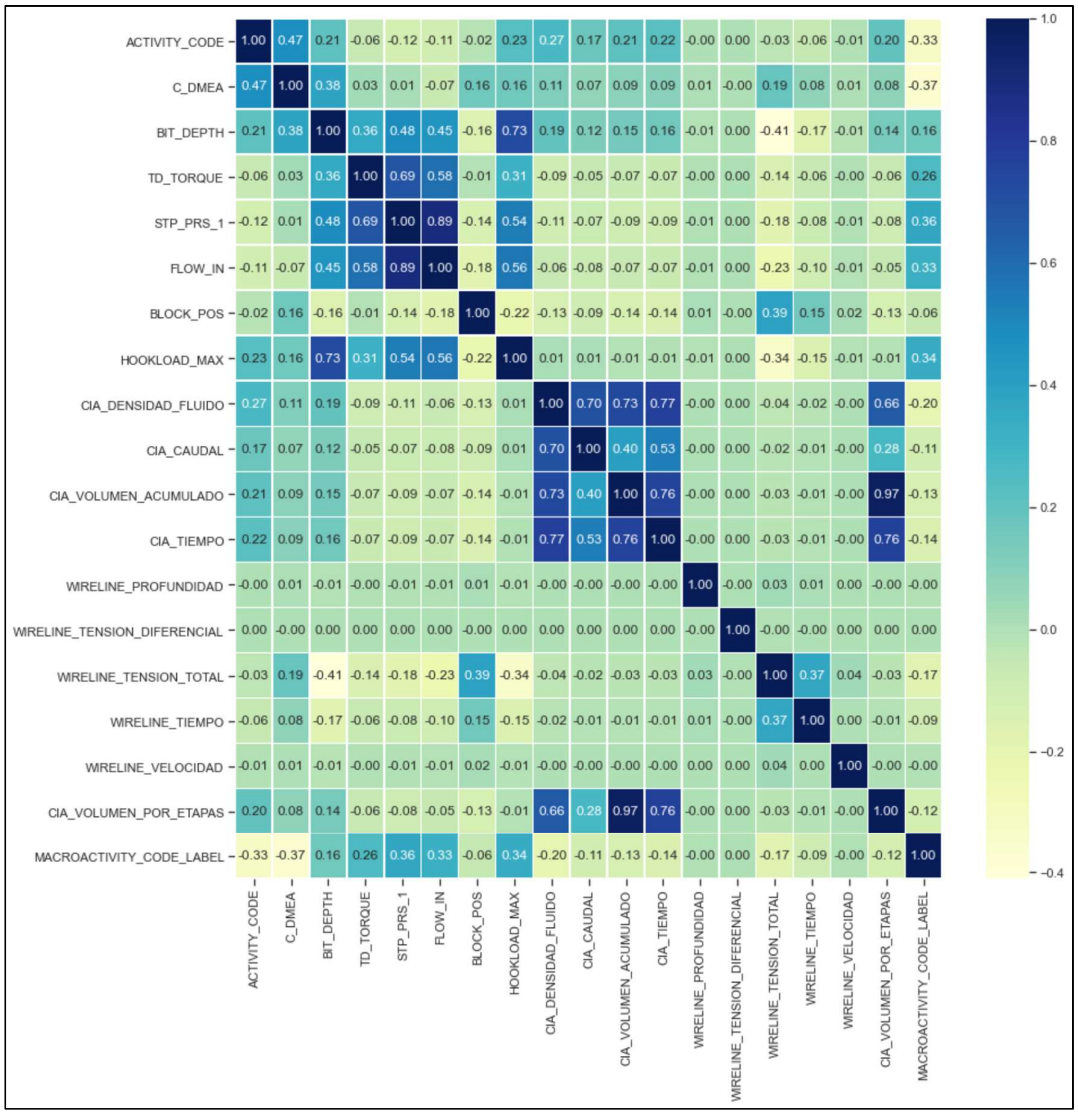

Fig. 3 - Correlación entre las distintas variables del dataset

Como se puede observar, no se encuentra una gran correlación entre la distintas variables con excepción de entre algunos pares de variables cercanos a la diagonal. Tampoco vemos una fuerte correlación entre la variable a predecir y las demás variables.

En el Anexo 3 se pueden ver los procesos utilizados para la limpieza del Dataset.

8.4. Preparación de Datos para el Modelado

Para el desarrollo del algoritmo de machine learning a los datos que se encuentran en el warehouse, realizamos el procesamiento de los datos y visualización de resultados, aplicando cuatro variantes de algoritmos clasificadores de machine Learning: Random Forest, SVM, Árboles de Decisión y Redes Neuronales. Todos estos algoritmos fueron realizados utilizando la librería **sklearn** de **Python.**

Iniciamos con la definición de las variables de entrada y la variable de salida (variable a predecir), y dividimos los datos en dos partes, una parte para modelar y otra parte para el testing.

El Testing va a ser un 30% del total. A su vez la parte del modelado se divide un 70% para el training del modelo y el otro 30% para validar el modelo.

Antes de aplicar el modelo normalizamos los datos. La normalización la aplicaremos solo a los datos de entrada. Como en cualquier transformación el fit solo se utiliza en los datos de training, porque si se aplicara a todo X, podría resultar en perdida de información (data leaks).

También realizamos una reducción de la dimensionalidad del dataset utilizando PCA (Principal Component Analysis) para deshacernos de las variables que no aportan al problema. Como en el caso de normalización, el fit solo se realiza a los datos de training. Probamos con 6 componentes y pudimos verificar que **utilizando solamente 4 o 5 componentes es sufciente, ya que cubren más del 99% de los datos.** En la Figura 4 se pueden ver las 6 componentes obtenidas.

En el Anexo 4 se pueden ver los procesos utilizados para la preparación del Modelado.

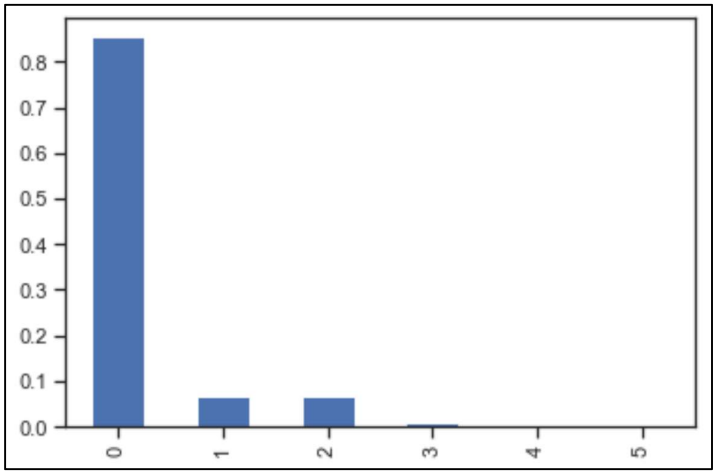

Fig. 4 – Resultado del PCA aplicado

8.5. Diseño del Panel

Se desarrolló un **plugin en Java** para brindar un acceso, a través de un botón dentro del reporte del parte diario de cada pozo y evento, que permita extraer los datos correspondientes de la base de SQL Server y dejarlos disponibles en la aplicación corporativa. En la Figura 5 se puede ver el botón “MACRO” mencionado.

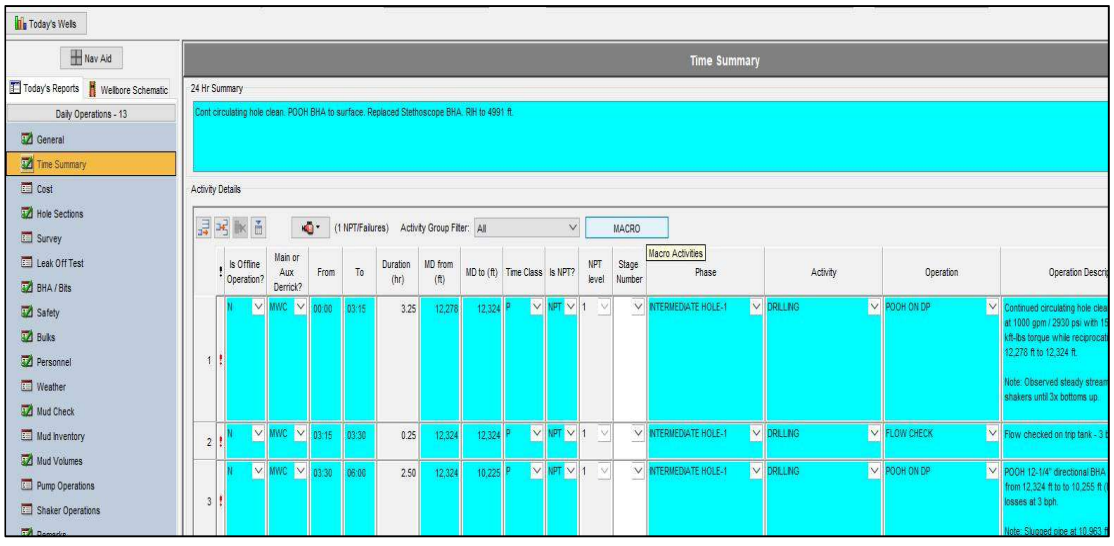

Fig. 5 – Botón “MACRO” dentro del reporte diario

9. Resultados Experimentales

Una vez preparado el dataset, aplicamos las cuatro variantes de algoritmos clasificadores de machine Learning mencionados en la Sección 8.4, a saber, Random Forest, SVM, Árboles de Decisión y Redes Neuronales.

9.1. Clasificador Random Forest

Definimos el pipeline con Normalización, Componentes Principales (PCA) y como método utilizamos Random Forest:

```
Pipeline(steps=[('sc', StandardScaler()), ('pca', PCA(n_components=5,
random_state=2)), ('model', RandomForestClassifier(random_state=2))])
```

Aplicamos el modelo, utilizando Trainig y Testing acorde a lo indicado en la sección 8.3., y obtenemos los siguientes resultados:

```
Train: 0.9916357408205044
Validation: 0.883226042761246
Blind: 0.8826357517136122
```

Como podemos observar, los resultados son excelentes ya que tenemos casi un 90% de accuracy tanto para el set de Validation como para set de Blind, y con un resultado casi del 90% en general se puede decir que es un modelo suficientemente válido para poder utilizarlo.

En la Figura 6 podemos visualizar la Matriz de Confusión correspondiente.

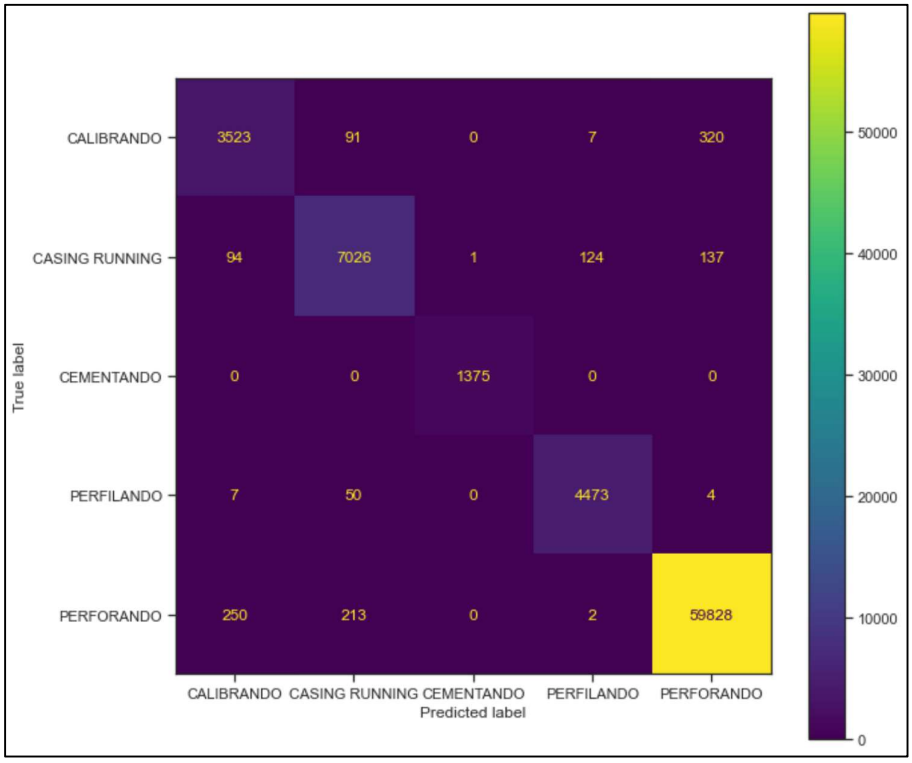

Fig. 6 - Matriz de Confusión correspondiente a Random Forest

Los registros bien clasificados son los que se encuentran en la diagonal, con lo cual si hacemos la suma nos da que hay 76225 registros bien clasificados, y los mal clasificados son los que están fueran de la diagonal, es decir 1300, dando la suma de ambos un total de 77525. En la Figura 7 se muestra el resumen de *Accuracy, Macro Average* y *Weighted Average* obtenidos.

	precision	recall	f1-score	support
CALIBRANDO	0.91	0.89	0.90	3941
CASING RUNNING	0.95	0.95	0.95	7382
CEMENTANDO	1.00	1.00	1.00	1375
PERFILANDO	0.97	0.99	0.98	4534
PERFORANDO	0.99	0.99	0.99	60293
accuracy			0.98	77525
macro avg	0.96	0.96	0.96	77525
weighted avg	0.98	0.98	0.98	77525

Fig. 7 – Accuracy y Average para Random Forest

9.2. Clasificador Support Vector Machine

Como segunda opción, probamos SVM, para comparar y determinar si podemos obtener un mejor resultado.

```
Pipeline(steps=[('sc', StandardScaler()), ('pca', PCA(n_components=5,
random_state=2)), ('model', SVC(random_state=2))])
```

Nuevamente, aplicamos el modelo, utilizando Trainig y Testing acorde a lo indicado en la sección 8.3., y obtenemos los siguientes resultados:

```
Train: 0.2743373144703676
Validation: 0.2779786212605744
Blind: 0.274634910507477
```

En la Figura 8 se muestra el resumen de *Accuracy, Macro Average* y *Weighted Average* correspondientes.

	precision	recall	f1-score	support
CALIBRANDO	0.01	0.00	0.00	3941
CASING RUNNING	0.96	0.56	0.70	7382
CEMENTANDO	1.00	0.98	0.99	1375
PERFILANDO	0.95	0.90	0.93	4534
PERFORANDO	0.90	1.00	0.94	60293
accuracy			0.90	77525
macro avg	0.76	0.69	0.71	77525
weighted avg	0.86	0.90	0.87	77525

Fig. 8 – Accuracy y Average para Support Vector Machine.

Como vemos los resultados son pésimos. No es posible utilizar SVM, al menos con los parámetros por default.

9.3. Clasificador Decision Tree

Como siguiente método probaremos un árbol de decisión, manteniendo el resto de las condiciones.

```
Pipeline(steps=[('sc', StandardScaler()), ('pca', PCA(n_components=5,
random_state=2)), ('model', DecisionTreeClassifier(random_state=2))])
```

Una vez más, aplicamos el modelo, utilizando Trainig y Testing acorde a lo indicado en la sección 8.3., y obtenemos los siguientes resultados:

```
Train: 0.9485369560213662
Validation: 0.7945955969343177
Blind: 0.7858039281053182
```

En la Figura 9 se muestran los valores de *Accuracy, Macro Average* y *Weighted Average* obtenidos.

	precision	recall	f1-score	support
CALIBRANDO	0.82	0.82	0.82	3941
CASING RUNNING	0.89	0.93	0.91	7382
CEMENTANDO	1.00	1.00	1.00	1375
PERFILANDO	0.97	0.94	0.96	4534
PERFORANDO	0.99	0.98	0.99	60293
accuracy			0.97	77525
macro avg	0.94	0.94	0.94	77525
weighted avg	0.97	0.97	0.97	77525

Fig. 9 – Accuracy y Average para Decision Tree

Podemos observar que el Decision Tree Classifier da buenos resultados ya que tenemos un accuracy casi del 80% y es muy rápido pero no supera a Random Forest que se encuentra cerca del 90%.

9.4. Clasificador Neural Network

Como último método, probaremos un red neuronal basada en un Perceptron Multicapa:

```
Pipeline(steps=[('sc', StandardScaler()), ('pca', PCA(n_components=5,
random_state=2)), ('model', MLPClassifier(max_iter=1000, random_state=2))])
```

Al aplicar el modelo, utilizando Trainig y Testing acorde a lo indicado en la sección 8.3., y obtenemos los siguientes resultados:

```
Train: 0.8655940783474907
Validation: 0.8568998532624599
Blind: 0.8554222537517451
```

En la Figura 10 se muestra el resumen de *Accuracy, Macro Average* y *Weighted Average* correspondientes.

	precision	recall	f1-score	support
CALIBRANDO	0.88	0.83	0.85	3941
CASING RUNNING	0.93	0.95	0.94	7382
CEMENTANDO	1.00	1.00	1.00	1375
PERFILANDO	0.96	0.99	0.97	4534
PERFORANDO	0.99	0.99	0.99	60293
accuracy			0.98	77525
macro avg	0.95	0.95	0.95	77525
weighted avg	0.98	0.98	0.98	77525

Fig. 10 – Accuracy y Average para Neural Network

En este caso podemos ver el que modelo es muy bueno, inclusive mejor que el de Árboles de Decisión, pero no alcanza el nivel de Random Forest.

Luego de haber aplicado los cuatro modelos propuesto, sin dudas descartamos SVM. Los otros tres pueden considerarse como válidos, pero la elección como método clasificador para nuestro dataset es Random Forest, no sólo porque su exactitud mejor

que los demás, sino por la cantidad de recursos que consume y la velocidad para hacer fit del modelo, especialmente en comparación con la enorme cantidad de recursos que utiliza el modelo de redes neuronales. En nuestro problema, eso es un punto fundamental ya que los datos se obtienen en tiempo real.

En el Anexo 5 se pueden ver todos los procesos para los cuatro algoritmos clasificadores utilizados.

10. Conclusiones y Trabajo a Futuro

Como conclusión podemos decir que nuestro estudio fue satisfactorio ya qué encontramos un algoritmo de *machine learning* que se ajusta a nuestro set de Datos y necesidades, pudiendo utilizarse para clasificar las actividades de real time para poder predecir la macro actividad correspondiente evitando que se siga realizando la carga de forma manual por parte del Supervisor del Pozo.

Principalmente lo que queda por hacer es adaptar lo que se ha hecho en Python con los modelos de aprendizaje para que pueda ser consumido en Java desde el plugin desarrollado ya que actualmente solamente toma los datos de la base de datos que dejaron los algoritmos procedurales vigentes.

También se puede continuar el estudio de nuevos algoritmos de *machine learning* ya que a pesar de haber obtenido una muy buena solución con una exactitud casi del 90%, seguramente pueda superarse aún más.

11. Referencias Bibliográficas

[1] B. Esmael, R. Fruhwirth, A. Arnaout, and G. Thonhauser. "Operations Recognition at Drill-Rigs" EGU General Assembly 2012, Vienna, Austria, Abril, 2012.

[2] T.K. Ho, M. Basu, "Complexity measures of supervised Classification Problems". IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 3, pp. 289-300, 2002

[3] X. Wu , V. Kumar , J. R. Quinlan , J. Ghosh , Q. Yang , H. Motoda , G. J. McLachlan , A. F. M. Ng , B. Liu , P. S. Yu , Z.-H. Zhou , M. Steinbach , D. J. Hand and D. Steinberg "Top 10 Algorithms in Data Mining", Knowledge and Information Systems, vol. 14, no. 1, pp.1 -37 2008

[4] E. Alpaydin, Introduction to machine learning, second edition, The MIT Press, Cambridge, 2010.

[5] C. D. Sutton, Classification and regression trees, bagging, and boosting, In C. R. Rao, E. J. Wegman and J. L. Solka, editors, Data mining and data visualization, pages 303-330, Elsevier B.V., Amsterdam, 2005.

[6] Galina V. Veres and Zoheir A. Sabeur, "*Data Analytics for Drilling Operational States Classifications*", University of Southampton, April 2015

[7] J. Gorodkin. "*Comparing two K-category assignments by a K-category correlation coefficient*". Comp Biol and Chem, vol. 28, pp. 367-374, 2004.

12. Anexos

Anexo 1 - "DailyTimeSummaryPlugin.java"

```
package com.lgc.dws.forms.daily;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.lang.reflect.Field;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import javax.swing.Action;
import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;
import org.apache.commons.lang.RandomStringUtils;
import com.lgc.dws.common.dsx.XDaily;
import com.lgc.dws.common.dsx.XEvent;
import com.lgc.dws.common.hierarchicaltable.BusinessObjectNode;
import com.lgc.dws.dimsng.DimsNG;
import com.lgc.dws.forms.LockableButton;
import com.lgc.dws.forms.Spreadsheet;
import com.lgc.dws.forms.Spreadsheet.SpreadsheetButton;
```

```
import com.lgc.dws.odbl.sections.SectionModel;
import com.lgc.dws.pages.IPsuedoColumn;

public class DailyTimeSummaryPanelPlugin extends DailyTimeSummaryPanel {
	public DailyTimeSummaryPanelPlugin(){
		rtoPlanImportButton = new SpreadsheetButton();
		rtoInsertButton = new SpreadsheetButton();
		rtoDeleteButton = new SpreadsheetButton();
		rtoMicroActivitiesButton = new SpreadsheetButton();
		detailsPanel = new JPanel();
		try
		{
			init();
		}
		catch(Exception e)
		{
			DimsNG.getLog().log(e);
		}
	}

private void init()
			throws Exception
	{
rtoPlanImportButton.setPreferredSize(new Dimension(120, 24));
		rtoPlanImportButton.setRequestFocusEnabled(false);
		rtoPlanImportButton.setText("RTO ONLINE");
rtoPlanImportButton.setToolTipText("Import Real Time Operations Plan");
		rtoPlanImportButton.setIcon(myImportIcon);
rtoPlanImportButton.addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent e) {
				importRTOPlan(timeSummarySpreadsheet);
```

```
                }
            });
rtoInsertButton.setPreferredSize(new Dimension(120, 24));
            rtoInsertButton.setRequestFocusEnabled(false);
            rtoInsertButton.setText("INSERT RTO");
rtoInsertButton.setToolTipText("Insert Real Time Operation");
            rtoInsertButton.setIcon(insertIcon);
rtoInsertButton.addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(ActionEvent e) {
                    insertRTO(timeSummarySpreadsheet);
                }
            });
rtoDeleteButton.setPreferredSize(new Dimension(140, 24));
            rtoDeleteButton.setRequestFocusEnabled(false);
            rtoDeleteButton.setText("DELETE RTO");
rtoDeleteButton.setToolTipText("Delete Real Time Operation");
            rtoDeleteButton.setIcon(deleteIcon);
rtoDeleteButton.addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(ActionEvent e) {
                    deleteRTO(timeSummarySpreadsheet);
                }
            });
rtoMicroActivitiesButton.setPreferredSize(new Dimension(120, 24));
rtoMicroActivitiesButton.setRequestFocusEnabled(false);
            rtoMicroActivitiesButton.setText("MACRO");
rtoMicroActivitiesButton.setToolTipText("Macro Activities");
rtoMicroActivitiesButton.addActionListener(new ActionListener() {

                @Override
```

```
                    public void actionPerformed(ActionEvent e) {
if (timeSummarySpreadsheet.getSelectedNodes().length == 1){
                        try {
String wellId = DailyTimeSummaryPanelPlugin.this.getReportJournal().getNamedUIValueAsString("well_id");
String eventId = DailyTimeSummaryPanelPlugin.this.getReportJournal().getNamedUIValueAsString("event_id");
String macroActivityId = (String) timeSummarySpreadsheet.getSelectedNodes()[0].getAttributeValue("DM_ACTIVITY.activity_alt_code1");
String macroActivity = (String) timeSummarySpreadsheet.getSelectedNodes()[0].getAttributeValue("DM_ACTIVITY.cost_subcode");
new MicroActivityDialog(wellId, eventId, macroActivityId, macroActivity).setVisible(true);
                        } catch (Exception ex) {
                            ex.printStackTrace();
                        }
                    }else{
JOptionPane.showMessageDialog(null, "Debe elegir una operación para ver sus micro actividades asociadas" , "Advertencia",  JOptionPane.WARNING_MESSAGE);
                    }
                }
            });
            jPanelBody.setBorder(BorderFactory.createEtchedBorder());
            detailsPanel.setLayout(new FlowLayout());
        }

        public void setSectionModel(SectionModel sm)
        {
            super.setSectionModel(sm);
            XDaily xDaily = (XDaily)getSectionModel().GetBusinessObjectFromMap("DM_DAILY");
XEvent curXEvent = (XEvent) xDaily.getDefaultParent();
```

```
timeSummarySpreadsheet = getFieldFromInstance(this, this.getClass().getSuperclass(), "dIMSSpreadsheetBean1");
try {
if (!curXEvent.getNamedUIValueAsString("event_code").equals("REP")){
((LockableButton)getFieldFromInstance(timeSummarySpreadsheet, Spreadsheet.class, "importButton")).setVisible(false);
JPopupMenu popupMenu = getFieldFromInstance(timeSummarySpreadsheet, Spreadsheet.class, "popupMenu");
Action importAction = getFieldFromInstance(timeSummarySpreadsheet, Spreadsheet.class, "importAction");
for (Component component : popupMenu.getComponents()) {
if (component instanceof JMenuItem && ((JMenuItem)component).getAction() == importAction){
                                popupMenu.remove(component);
                                break;
                        }
                }
        }
        } catch (Exception e) {
                e.printStackTrace();
        }
        String isRealTime = "";
        try {
isRealTime = curXEvent.getNamedUIValueAsString("planned_wci");
        } catch (Exception e) {
                e.printStackTrace();
        }

        if (!isRealTime.equals("") && isRealTime.equals("YES")){

        setColumnReadOnly("md_to",true);
                setColumnReadOnly("activity_duration",true);
                rtoPlanImportButton.setVisible(true);
                rtoInsertButton.setVisible(true);
```

```
                    rtoDeleteButton.setVisible(true);
                    rtoMicroActivitiesButton.setVisible(true);
        timeSummarySpreadsheet.addToolbarComponent(rtoPlanImportButton, false);

        timeSummarySpreadsheet.addToolbarComponent(rtoInsertButton, false);

        timeSummarySpreadsheet.addToolbarComponent(rtoDeleteButton, false);

        timeSummarySpreadsheet.addToolbarComponent(rtoMicroActivitiesButton,
false);

        ((LockableButton)getFieldFromInstance(timeSummarySpreadsheet,
Spreadsheet.class, "insertButton")).setVisible(false);

        ((LockableButton)getFieldFromInstance(timeSummarySpreadsheet,
Spreadsheet.class, "deleteButton")).setVisible(false);

        ((LockableButton)getFieldFromInstance(timeSummarySpreadsheet,
Spreadsheet.class, "addButton")).setVisible(false);
            }else{
                    timeSummarySpreadsheet.setCreateRowsEnabled(true);
                    timeSummarySpreadsheet.setDeleteRowsEnabled(true);
                    setColumnReadOnly("md_to",false);
                    setColumnReadOnly("activity_duration",false);

        ((LockableButton)getFieldFromInstance(timeSummarySpreadsheet,
Spreadsheet.class, "insertButton")).setVisible(true);

        ((LockableButton)getFieldFromInstance(timeSummarySpreadsheet,
Spreadsheet.class, "deleteButton")).setVisible(true);

        ((LockableButton)getFieldFromInstance(timeSummarySpreadsheet,
Spreadsheet.class, "addButton")).setVisible(true);
                    rtoPlanImportButton.setVisible(false);
                    rtoInsertButton.setVisible(false);
                    rtoDeleteButton.setVisible(false);
                    rtoMicroActivitiesButton.setVisible(false);
```

```
            }
    }

private void importRTOPlan(Spreadsheet timeSummarySpreadsheet) {
            Connection connection = DimsNG.getSharedInstance().getDataContext().getDBConnection("getRTOPlan");

            try {
                Statement stmt = connection.createStatement();
                StringBuffer query = new StringBuffer();
                String wellId = this.getReportJournal().getNamedUIValueAsString("well_id");
                String eventId = this.getReportJournal().getNamedUIValueAsString("event_id");
                Calendar dateReport = (Calendar)this.getReportJournal().getNamedUIValue("date_report");
                SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");
                ResultSet resultSet = null;
                SimpleDateFormat dateTimeStampFormat = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
                SimpleDateFormat dateTimeFormat = new SimpleDateFormat("dd-MM-yyyy HH:mm");
                SimpleDateFormat timeFormat = new SimpleDateFormat("HH:mm");
                Date maxTimeTo = getMaxTimeto(timeSummarySpreadsheet);
                query = new StringBuffer();
                query.append("SELECT macro_codigo, MIN(desde) AS time_from, MAX(hasta) AS time_to, MAX(cdmea)*0.3048000000012 AS depth");
                query.append(" FROM EDM_ETL.RTO_FROM_MULE_CONS WHERE well_id = '" + wellId + "' AND event_id = '" + eventId + "'");
                query.append(" AND TO_CHAR(desde,'DD-MM-YYYY') = '" + dateFormat.format(dateReport.getTime()) + "'");
                query.append(" GROUP BY macro_codigo");
                if (maxTimeTo != null){
```

```
                query.append(" HAVING MIN(desde) >= " + "TO_DATE('"
+ dateTimeFormat.format(maxTimeTo) + "','DD-MM-YYYY HH24:MI')");
            }
            query.append(" ORDER BY MIN(desde)");
            resultSet = stmt.executeQuery(query.toString());
            boolean[] originalCarryDownValues =
removeCarryDownFromColumns(timeSummarySpreadsheet);
            while (resultSet.next()){
                String macroActivity =
resultSet.getString("macro_codigo");
                Date timeFrom = resultSet.getTimestamp("time_from");
                Date timeTo = resultSet.getTimestamp("time_to");
                Double depth = resultSet.getDouble("depth");
                BusinessObjectNode newChild =
timeSummarySpreadsheet.getStartNode().addNewChildNode(timeSummarySpreadshe
et.getStartNode().getChildCount());

        newChild.setAttributeValue("DM_ACTIVITY.activity_alt_code1",
RandomStringUtils.randomAlphanumeric(5));

        newChild.setAttributeValue("DM_ACTIVITY.cost_subcode", macroActivity + " ("
+ timeFormat.format(timeFrom) + " - " + timeFormat.format(timeTo) + ")");
                Calendar cal = Calendar.getInstance();
                cal.setTime(timeFrom);
                cal.set(Calendar.SECOND, 0);
                newChild.setAttributeValue("DM_ACTIVITY.time_from",
cal);
                cal.setTime(timeTo);
                cal.set(Calendar.SECOND, 0);
                newChild.setAttributeValue("DM_ACTIVITY.time_to",
cal);
                newChild.setAttributeValue("DM_ACTIVITY.md_to",
depth);

        timeSummarySpreadsheet.getModel().fireNodeAddedWithCarrydownEvent(ne
wChild);
                StringBuffer updateQuery = new StringBuffer();
```

```
                        updateQuery.append("UPDATE
EDM_ETL.RTO_FROM_MULE_CONS SET activity_macro_id = '" +
newChild.getAttributeValue("DM_ACTIVITY.activity_alt_code1") + "'");
                        updateQuery.append(" WHERE well_id = '" + wellId + "'
AND event_id = '" + eventId + "'");
                        updateQuery.append(" AND TO_CHAR(desde,'DD-MM-
YYYY HH24:MI:SS') >= '" + dateTimeStampFormat.format(timeFrom.getTime()) + "'");
                        updateQuery.append(" AND TO_CHAR(hasta,'DD-MM-
YYYY HH24:MI:SS') <= '" + dateTimeStampFormat.format(timeTo.getTime()) + "'");
                        updateQuery.append(" AND macro_codigo = '" +
macroActivity + "'");
                        Statement stmt2 = connection.createStatement();
                        stmt2.execute(updateQuery.toString());
                        stmt2.close();
                    }
                    setColumnCarryDownValues(timeSummarySpreadsheet,
originalCarryDownValues);
                    //stmt.close();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }

            private void insertRTO(Spreadsheet timeSummarySpreadsheet){
                if (timeSummarySpreadsheet.getSelectedNodes().length == 1){
                    InsertRTODialog insertRTODialog = new InsertRTODialog(this,
timeSummarySpreadsheet);
                    insertRTODialog.setVisible(true);
                }else{
                    JOptionPane.showMessageDialog(null, "Debe elegir una
operaciÛn para poder insertar" , "Advertencia", JOptionPane.WARNING_MESSAGE);
                }
            }

            private void deleteRTO(Spreadsheet timeSummarySpreadsheet){
```

```
try{
    if (timeSummarySpreadsheet.getSelectedNodes().length == 1){
        boolean deleteNode = false;
        BusinessObjectNode selectedNode = timeSummarySpreadsheet.getSelectedNodes()[0];
        int currentPos = selectedNode.getParent().getChildNodes().indexOf(selectedNode);
        BusinessObjectNode previousNode = null;
        if (currentPos > 0){
            previousNode = (BusinessObjectNode)selectedNode.getParent().getChildNodes().get(currentPos-1);
        }
        BusinessObjectNode nextNode = null;
        if (currentPos < selectedNode.getParent().getChildNodes().size()-1){
            nextNode = (BusinessObjectNode)selectedNode.getParent().getChildNodes().get(currentPos+1);
        }
        String previousMacroActivityId = null;
        String nextMacroActivityId = null;
        if (previousNode != null){
            previousMacroActivityId = (String)previousNode.getAttributeValue("DM_ACTIVITY.activity_alt_code1");
        }
        if (nextNode != null){
            nextMacroActivityId = (String)nextNode.getAttributeValue("DM_ACTIVITY.activity_alt_code1");
        }
        String selectedMacroActivityId = (String)selectedNode.getAttributeValue("DM_ACTIVITY.activity_alt_code1");
        Calendar selectedTimeFrom = (Calendar)selectedNode.getAttributeValue("DM_ACTIVITY.time_from");
        Calendar selectedTimeTo= (Calendar)selectedNode.getAttributeValue("DM_ACTIVITY.time_to");
        if (previousMacroActivityId != null && previousMacroActivityId.equals(selectedMacroActivityId)){
```

```
previousNode.setAttributeValue("DM_ACTIVITY.time_to", selectedTimeTo);

timeSummarySpreadsheet.getModel().fireNodeChangedEvent(previousNode);
                    deleteNode = true;
                }else if (nextMacroActivityId != null && nextMacroActivityId.equals(selectedMacroActivityId)){

nextNode.setAttributeValue("DM_ACTIVITY.time_from", selectedTimeFrom);

timeSummarySpreadsheet.getModel().fireNodeChangedEvent(nextNode);
                    deleteNode = true;
                }
                if (deleteNode){

selectedNode.getParent().removeChildNode(selectedNode);

timeSummarySpreadsheet.getModel().fireNodeRemovedEvent(selectedNode);
                }else{
                    JOptionPane.showMessageDialog(null, "No se puede eliminar la Macro Actividad del Plan" , "Advertencia", JOptionPane.WARNING_MESSAGE);
                }
            }else{
                JOptionPane.showMessageDialog(null, "Debe elegir una operaciÛn para poder borrar" , "Advertencia",  JOptionPane.WARNING_MESSAGE);
            }
        }catch (Exception e) {
            e.printStackTrace();
        }
    }

    private Date getMaxTimeto(Spreadsheet timeSummarySpreadsheet){
        Date maxTimeTo = null;
        @SuppressWarnings("unchecked")
```

```
            List<BusinessObjectNode> childNodes = timeSummarySpreadsheet.getStartNode().getChildNodes();
            for (BusinessObjectNode businessObjectNode : childNodes) {
                Calendar timeTo = (Calendar)businessObjectNode.getAttributeValue("DM_ACTIVITY.time_to");
                if (maxTimeTo == null || maxTimeTo.before(timeTo.getTime())){
                    maxTimeTo = timeTo.getTime();
                }
            }
            return maxTimeTo;
        }

        /**
         *
         * @return originalCarryDownValues
         */
        static boolean[] removeCarryDownFromColumns(Spreadsheet timeSummarySpreadsheet){
            IPsuedoColumn[] columns = timeSummarySpreadsheet.getColumns();
            boolean[] originalCarryDownValues = new boolean[columns.length];
            for (int i = 0; i < columns.length; i++) {
                IPsuedoColumn iPsuedoColumn = columns[i];
                originalCarryDownValues[i] = iPsuedoColumn.isCarryDown();
                iPsuedoColumn.setCarryDown(false);
            }
            return originalCarryDownValues;
        }

        static void setColumnCarryDownValues(Spreadsheet timeSummarySpreadsheet, boolean[] carryDownValues){
            IPsuedoColumn[] columns = timeSummarySpreadsheet.getColumns();
            for (int i = 0; i < columns.length; i++) {
                columns[i].setCarryDown(carryDownValues[i]);
            }
```

```
	}

	private void setColumnReadOnly(String columnName, boolean readOnly){
		IPsuedoColumn[] columns = timeSummarySpreadsheet.getColumns();
		for (int i = 0; i < columns.length; i++) {
			if (columns[i].getAttributeName().equals(columnName)){
				columns[i].setReadOnly(readOnly);
				break;
			};
		}
	}

	@SuppressWarnings({ "rawtypes", "unchecked" })
	private <T> T getFieldFromInstance(Object instance, Class clazz, String fieldName){
		T reportField = null;
		try {
			Field field = clazz.getDeclaredField(fieldName);
			field.setAccessible(true);
			reportField = (T) field.get(instance);
		} catch (Exception e) {
			e.printStackTrace();
		}
		return reportField;
	}

	private static final long serialVersionUID = 1L;
	private JPanel detailsPanel;
	private Spreadsheet timeSummarySpreadsheet = null;
	protected SpreadsheetButton rtoPlanImportButton;
	protected SpreadsheetButton rtoInsertButton;
	protected SpreadsheetButton rtoDeleteButton;
	protected SpreadsheetButton rtoMicroActivitiesButton;
```

```
        private static final ImageIcon myImportIcon = new
ImageIcon(com.lgc.dws.forms.Spreadsheet.class.getResource("spreadsheet/images/i
mport.png"));
        private static final ImageIcon insertIcon = new
ImageIcon(Spreadsheet.class.getResource("spreadsheet/images/insert.png"));
        private static final ImageIcon deleteIcon = new
ImageIcon(Spreadsheet.class.getResource("images/delete_row.png"));
}
```

Anexo 2 – Proceso ETL

[1]:
```
import pandas as pd

pd.set_option('max_columns', None)
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn import tree
from sklearn import svm
from sklearn import metrics
import numpy as np

import pickle
```

Importar el DataSet

[2]:
```
parse_dates = ['DATE_TIME']
well_data_PCG_1277 = pd.read_csv('DataSets_TP_Final_ECD/PCG-1277.csv', sep =
';
',', decimal=",",
dtype={"WELL_NAME": "string", "ACTIVITY_CODE":␣
,"string", "MACROACTIVITY_CODE": "string"},
              parse_dates = parse_dates)

well_data_PCG_1277.head(10)
```

[2]:
```
WELL_NAME DATE_TIME ¥
0 PCG- 1277 … 2019-09-10 00:00:00
1 PCG- 1277 … 2019-09-10 00:00:05
2 PCG- 1277 … 2019-09-10 00:00:10
3 PCG- 1277 … 2019-09-10 00:00:15
4 PCG- 1277 … 2019-09-10 00:00:20
5 PCG- 1277 … 2019-09-10 00:00:25
6 PCG- 1277 … 2019-09-10 00:00:30
7 PCG- 1277 … 2019-09-10 00:00:35
8 PCG- 1277 … 2019-09-10 00:00:40
9 PCG- 1277 … 2019-09-10 00:00:45
```

```
ACTIVITY_CODE MACROACTIVITY_CODE C_DMEA ¥
0 PERFORANDO 0.0
1 PERFORANDO 0.0
2 PERFORANDO 0.0
3 PERFORANDO 0.0
4 PERFORANDO 0.0
5 PERFORANDO 0.0
6 PERFORANDO 0.0
7 PERFORANDO 0.0
8 PERFORANDO 0.0
9 PERFORANDO 0.0
BIT_DEPTH TD_TORQUE STP_PRS_1 FLOW_IN BLOCK_POS HOOKLOAD_MAX ¥
0 0.0 0.000000 0.0 0.0 -0.844452 21.44336
1 0.0 0.000000 0.0 0.0 -0.835339 21.44336
2 0.0 3.976276 0.0 0.0 -0.860829 21.35895
3 0.0 4.732082 0.0 0.0 -0.836936 21.39916
4 0.0 0.000000 0.0 0.0 -0.830801 21.39916
5 0.0 1.112243 0.0 0.0 -0.844519 21.18597
6 0.0 0.000000 0.0 0.0 -0.827136 21.40165
7 0.0 1.476786 0.0 0.0 -0.850366 21.37076
8 0.0 0.000000 0.0 0.0 -0.818154 21.54548
9 0.0 0.000000 0.0 0.0 -0.873479 21.66736
CIA_PRESION_DIRECTA CIA_DENSIDAD_FLUIDO CIA_CAUDAL ¥
0 NaN 0.0 0.0
1 NaN 0.0 0.0
2 NaN 0.0 0.0
3 NaN 0.0 0.0
4 NaN 0.0 0.0
5 NaN 0.0 0.0
6 NaN 0.0 0.0
7 NaN 0.0 0.0
8 NaN 0.0 0.0
9 NaN 0.0 0.0
CIA_VOLUMEN_ACUMULADO CIA_TIEMPO WIRELINE_PROFUNDIDAD ¥
0 0.0 0.0 0.0
1 0.0 0.0 0.0
2 0.0 0.0 0.0
3 0.0 0.0 0.0
4 0.0 0.0 0.0
5 0.0 0.0 0.0
6 0.0 0.0 0.0
7 0.0 0.0 0.0
8 0.0 0.0 0.0
9 0.0 0.0 0.0
WIRELINE_TENSION_DIFERENCIAL WIRELINE_TENSION_TOTAL WIRELINE_TIEMPO ¥
```

```
0 0.0 0.0 0.0
1 0.0 0.0 0.0
2 0.0 0.0 0.0
3 0.0 0.0 0.0
4 0.0 0.0 0.0
5 0.0 0.0 0.0
6 0.0 0.0 0.0
7 0.0 0.0 0.0
8 0.0 0.0 0.0
9 0.0 0.0 0.0
WIRELINE_VELOCIDAD WIRELINE_VOLTAJE WIRELINE_INTENSIDAD ¥
0 0.0 NaN NaN
1 0.0 NaN NaN
2 0.0 NaN NaN
3 0.0 NaN NaN
4 0.0 NaN NaN
5 0.0 NaN NaN
6 0.0 NaN NaN
7 0.0 NaN NaN
8 0.0 NaN NaN
9 0.0 NaN NaN
CIA_VOLUMEN_POR_ETAPAS FINALIZACION_PROF_FASE_1 REFERENCIA_PROF_FASE_1 ¥
0 0.0 1 0
1 0.0 1 0
2 0.0 1 0
3 0.0 1 0
4 0.0 1 0
5 0.0 1 0
3
6 0.0 1 0
7 0.0 1 0
8 0.0 1 0
9 0.0 1 0
FINALIZACION_PROF_FASE_2 REFERENCIA_PROF_FASE_2 FINALIZACION_PROF_FASE_3 ¥
0 1 0 1
1 1 0 1
2 1 0 1
3 1 0 1
4 1 0 1
5 1 0 1
6 1 0 1
7 1 0 1
8 1 0 1
9 1 0 1
REFERENCIA_PROF_FASE_3 FINALIZACION_PROF_FASE_4 REFERENCIA_PROF_FASE_4 ¥
0 0 1 0
1 0 1 0
```

```
2 0 1 0
3 0 1 0
4 0 1 0
5 0 1 0
6 0 1 0
7 0 1 0
8 0 1 0
9 0 1 0
FINALIZACION_PROF_FASE_5 REFERENCIA_PROF_FASE_5
0 1 NaN
1 1 NaN
2 1 NaN
3 1 NaN
4 1 NaN
5 1 NaN
6 1 NaN
7 1 NaN
8 1 NaN
9 1 NaN
```

Verificar la información y tipo de campos del dataset

```
[3]:
well_data_PCG_1277.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 286989 entries, 0 to 286988

Data columns (total 34 columns):
# Column Non-Null Count Dtype
--- ------ -------------- -----
0 WELL_NAME 286989 non-null string
1 DATE_TIME 286989 non-null datetime64[ns]
2 ACTIVITY_CODE 286989 non-null string
3 MACROACTIVITY_CODE 258414 non-null string
4 C_DMEA 286989 non-null float64
5 BIT_DEPTH 286989 non-null float64
6 TD_TORQUE 286989 non-null float64
7 STP_PRS_1 286989 non-null float64
8 FLOW_IN 286989 non-null float64
9 BLOCK_POS 286989 non-null float64
10 HOOKLOAD_MAX 286989 non-null float64
11 CIA_PRESION_DIRECTA 0 non-null float64
12 CIA_DENSIDAD_FLUIDO 286989 non-null float64
13 CIA_CAUDAL 286989 non-null float64
14 CIA_VOLUMEN_ACUMULADO 286989 non-null float64
15 CIA_TIEMPO 286989 non-null float64
```

```
16 WIRELINE_PROFUNDIDAD 286989 non-null float64
17 WIRELINE_TENSION_DIFERENCIAL 286989 non-null float64
18 WIRELINE_TENSION_TOTAL 286989 non-null float64
19 WIRELINE_TIEMPO 286989 non-null float64
20 WIRELINE_VELOCIDAD 286989 non-null float64
21 WIRELINE_VOLTAJE 0 non-null float64
22 WIRELINE_INTENSIDAD 0 non-null float64
23 CIA_VOLUMEN_POR_ETAPAS 286989 non-null float64
24 FINALIZACION_PROF_FASE_1 286989 non-null int64
25 REFERENCIA_PROF_FASE_1 286989 non-null int64
26 FINALIZACION_PROF_FASE_2 286989 non-null int64
27 REFERENCIA_PROF_FASE_2 286989 non-null int64
28 FINALIZACION_PROF_FASE_3 286989 non-null int64
29 REFERENCIA_PROF_FASE_3 286989 non-null int64
30 FINALIZACION_PROF_FASE_4 286989 non-null int64
31 REFERENCIA_PROF_FASE_4 286989 non-null int64
32 FINALIZACION_PROF_FASE_5 286989 non-null int64
33 REFERENCIA_PROF_FASE_5 0 non-null float64
dtypes: datetime64[ns](1), float64(21), int64(9), string(3)
memory usage: 74.4 MB
```

Transformar el tipo de dato de la columna ACTIVITY_CODE a tipo numérico

```
[4]:
well_data_PCG_1277['ACTIVITY_CODE'] =
well_data_PCG_1277['ACTIVITY_CODE'].str.
↪strip()
well_data_PCG_1277['ACTIVITY_CODE'] = well_data_PCG_1277['ACTIVITY_CODE'].
↪apply(pd.to_numeric)

well_data_PCG_1277.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 286989 entries, 0 to 286988
Data columns (total 34 columns):
# Column Non-Null Count Dtype
--- ------ -------------- -----
0 WELL_NAME 286989 non-null string
1 DATE_TIME 286989 non-null datetime64[ns]
2 ACTIVITY_CODE 172259 non-null float64
3 MACROACTIVITY_CODE 258414 non-null string
4 C_DMEA 286989 non-null float64
5 BIT_DEPTH 286989 non-null float64
6 TD_TORQUE 286989 non-null float64
7 STP_PRS_1 286989 non-null float64
8 FLOW_IN 286989 non-null float64
```

```
9 BLOCK_POS 286989 non-null float64
10 HOOKLOAD_MAX 286989 non-null float64
11 CIA_PRESION_DIRECTA 0 non-null float64
12 CIA_DENSIDAD_FLUIDO 286989 non-null float64
13 CIA_CAUDAL 286989 non-null float64
14 CIA_VOLUMEN_ACUMULADO 286989 non-null float64
15 CIA_TIEMPO 286989 non-null float64
16 WIRELINE_PROFUNDIDAD 286989 non-null float64
17 WIRELINE_TENSION_DIFERENCIAL 286989 non-null float64
18 WIRELINE_TENSION_TOTAL 286989 non-null float64
19 WIRELINE_TIEMPO 286989 non-null float64
20 WIRELINE_VELOCIDAD 286989 non-null float64
21 WIRELINE_VOLTAJE 0 non-null float64
22 WIRELINE_INTENSIDAD 0 non-null float64
23 CIA_VOLUMEN_POR_ETAPAS 286989 non-null float64
24 FINALIZACION_PROF_FASE_1 286989 non-null int64
25 REFERENCIA_PROF_FASE_1 286989 non-null int64
26 FINALIZACION_PROF_FASE_2 286989 non-null int64
27 REFERENCIA_PROF_FASE_2 286989 non-null int64
28 FINALIZACION_PROF_FASE_3 286989 non-null int64
29 REFERENCIA_PROF_FASE_3 286989 non-null int64
30 FINALIZACION_PROF_FASE_4 286989 non-null int64
31 REFERENCIA_PROF_FASE_4 286989 non-null int64
32 FINALIZACION_PROF_FASE_5 286989 non-null int64
33 REFERENCIA_PROF_FASE_5 0 non-null float64
dtypes: datetime64[ns](1), float64(22), int64(9), string(2)
memory usage: 74.4 MB
```

Complementar los valores NA de la columna

[5]:

```
well_data_PCG_1277['ACTIVITY_CODE'] = well_data_PCG_1277['ACTIVITY_CODE'].
↪fillna(0)
well_data_PCG_1277['ACTIVITY_CODE'] = well_data_PCG_1277['ACTIVITY_CODE'].
↪astype('int64')
well_data_PCG_1277.info()
```

```
RangeIndex: 286989 entries, 0 to 286988
Data columns (total 34 columns):
# Column Non-Null Count Dtype
--- ------ -------------- -----
0 WELL_NAME 286989 non-null string
```

```
1 DATE_TIME 286989 non-null datetime64[ns]
2 ACTIVITY_CODE 286989 non-null int64
3 MACROACTIVITY_CODE 258414 non-null string
4 C_DMEA 286989 non-null float64
5 BIT_DEPTH 286989 non-null float64
6 TD_TORQUE 286989 non-null float64
7 STP_PRS_1 286989 non-null float64
8 FLOW_IN 286989 non-null float64
9 BLOCK_POS 286989 non-null float64
10 HOOKLOAD_MAX 286989 non-null float64
11 CIA_PRESION_DIRECTA 0 non-null float64
12 CIA_DENSIDAD_FLUIDO 286989 non-null float64
13 CIA_CAUDAL 286989 non-null float64
14 CIA_VOLUMEN_ACUMULADO 286989 non-null float64
15 CIA_TIEMPO 286989 non-null float64
16 WIRELINE_PROFUNDIDAD 286989 non-null float64
17 WIRELINE_TENSION_DIFERENCIAL 286989 non-null float64
18 WIRELINE_TENSION_TOTAL 286989 non-null float64
19 WIRELINE_TIEMPO 286989 non-null float64
20 WIRELINE_VELOCIDAD 286989 non-null float64
21 WIRELINE_VOLTAJE 0 non-null float64
22 WIRELINE_INTENSIDAD 0 non-null float64
23 CIA_VOLUMEN_POR_ETAPAS 286989 non-null float64
24 FINALIZACION_PROF_FASE_1 286989 non-null int64
25 REFERENCIA_PROF_FASE_1 286989 non-null int64
26 FINALIZACION_PROF_FASE_2 286989 non-null int64
27 REFERENCIA_PROF_FASE_2 286989 non-null int64
28 FINALIZACION_PROF_FASE_3 286989 non-null int64
29 REFERENCIA_PROF_FASE_3 286989 non-null int64
30 FINALIZACION_PROF_FASE_4 286989 non-null int64
31 REFERENCIA_PROF_FASE_4 286989 non-null int64
32 FINALIZACION_PROF_FASE_5 286989 non-null int64
33 REFERENCIA_PROF_FASE_5 0 non-null float64
dtypes: datetime64[ns](1), float64(21), int64(10), string(2)
memory usage: 74.4 MB
```

Setear indice el campo DATE_TIME

[6]:

```
index = well_data_PCG_1277.set_index('DATE_TIME', inplace=True)
```

[7]:

```
well_data_PCG_1277.head(10)
```

```
[7]:
WELL_NAME ¥
DATE_TIME
2019-09-10 00:00:00 PCG- 1277 …
2019-09-10 00:00:05 PCG- 1277 …
2019-09-10 00:00:10 PCG- 1277 …
2019-09-10 00:00:15 PCG- 1277 …
2019-09-10 00:00:20 PCG- 1277 …
2019-09-10 00:00:25 PCG- 1277 …
2019-09-10 00:00:30 PCG- 1277 …
2019-09-10 00:00:35 PCG- 1277 …
2019-09-10 00:00:40 PCG- 1277 …
2019-09-10 00:00:45 PCG- 1277 …
ACTIVITY_CODE MACROACTIVITY_CODE C_DMEA BIT_DEPTH ¥
DATE_TIME
2019-09-10 00:00:00 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:05 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:10 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:15 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:20 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:25 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:30 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:35 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:40 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:45 0 PERFORANDO 0.0 0.0
TD_TORQUE STP_PRS_1 FLOW_IN BLOCK_POS HOOKLOAD_MAX ¥
DATE_TIME
2019-09-10 00:00:00 0.000000 0.0 0.0 -0.844452 21.44336
2019-09-10 00:00:05 0.000000 0.0 0.0 -0.835339 21.44336
2019-09-10 00:00:10 3.976276 0.0 0.0 -0.860829 21.35895
2019-09-10 00:00:15 4.732082 0.0 0.0 -0.836936 21.39916
2019-09-10 00:00:20 0.000000 0.0 0.0 -0.830801 21.39916
2019-09-10 00:00:25 1.112243 0.0 0.0 -0.844519 21.18597
2019-09-10 00:00:30 0.000000 0.0 0.0 -0.827136 21.40165
2019-09-10 00:00:35 1.476786 0.0 0.0 -0.850366 21.37076
2019-09-10 00:00:40 0.000000 0.0 0.0 -0.818154 21.54548
2019-09-10 00:00:45 0.000000 0.0 0.0 -0.873479 21.66736
CIA_PRESION_DIRECTA CIA_DENSIDAD_FLUIDO CIA_CAUDAL ¥
DATE_TIME
2019-09-10 00:00:00 NaN 0.0 0.0
2019-09-10 00:00:05 NaN 0.0 0.0
2019-09-10 00:00:10 NaN 0.0 0.0
2019-09-10 00:00:15 NaN 0.0 0.0
2019-09-10 00:00:20 NaN 0.0 0.0
2019-09-10 00:00:25 NaN 0.0 0.0
2019-09-10 00:00:30 NaN 0.0 0.0
```

```
2019-09-10 00:00:35 NaN 0.0 0.0
2019-09-10 00:00:40 NaN 0.0 0.0
2019-09-10 00:00:45 NaN 0.0 0.0
CIA_VOLUMEN_ACUMULADO CIA_TIEMPO WIRELINE_PROFUNDIDAD ¥
DATE_TIME
2019-09-10 00:00:00 0.0 0.0 0.0
2019-09-10 00:00:05 0.0 0.0 0.0
2019-09-10 00:00:10 0.0 0.0 0.0
2019-09-10 00:00:15 0.0 0.0 0.0
2019-09-10 00:00:20 0.0 0.0 0.0
2019-09-10 00:00:25 0.0 0.0 0.0
2019-09-10 00:00:30 0.0 0.0 0.0
2019-09-10 00:00:35 0.0 0.0 0.0
2019-09-10 00:00:40 0.0 0.0 0.0
2019-09-10 00:00:45 0.0 0.0 0.0
WIRELINE_TENSION_DIFERENCIAL WIRELINE_TENSION_TOTAL ¥
DATE_TIME
2019-09-10 00:00:00 0.0 0.0
2019-09-10 00:00:05 0.0 0.0
2019-09-10 00:00:10 0.0 0.0
2019-09-10 00:00:15 0.0 0.0
2019-09-10 00:00:20 0.0 0.0
2019-09-10 00:00:25 0.0 0.0
2019-09-10 00:00:30 0.0 0.0
2019-09-10 00:00:35 0.0 0.0
2019-09-10 00:00:40 0.0 0.0
2019-09-10 00:00:45 0.0 0.0
WIRELINE_TIEMPO WIRELINE_VELOCIDAD WIRELINE_VOLTAJE ¥
DATE_TIME
2019-09-10 00:00:00 0.0 0.0 NaN
2019-09-10 00:00:05 0.0 0.0 NaN
2019-09-10 00:00:10 0.0 0.0 NaN
2019-09-10 00:00:15 0.0 0.0 NaN
2019-09-10 00:00:20 0.0 0.0 NaN
2019-09-10 00:00:25 0.0 0.0 NaN
2019-09-10 00:00:30 0.0 0.0 NaN
2019-09-10 00:00:35 0.0 0.0 NaN
2019-09-10 00:00:40 0.0 0.0 NaN
2019-09-10 00:00:45 0.0 0.0 NaN
WIRELINE_INTENSIDAD CIA_VOLUMEN_POR_ETAPAS ¥
DATE_TIME
2019-09-10 00:00:00 NaN 0.0
2019-09-10 00:00:05 NaN 0.0
2019-09-10 00:00:10 NaN 0.0
2019-09-10 00:00:15 NaN 0.0
2019-09-10 00:00:20 NaN 0.0
```

```
2019-09-10 00:00:25 NaN 0.0
2019-09-10 00:00:30 NaN 0.0
2019-09-10 00:00:35 NaN 0.0
2019-09-10 00:00:40 NaN 0.0
2019-09-10 00:00:45 NaN 0.0
FINALIZACION_PROF_FASE_1 REFERENCIA_PROF_FASE_1 ¥
DATE_TIME
2019-09-10 00:00:00 1 0
2019-09-10 00:00:05 1 0
2019-09-10 00:00:10 1 0
2019-09-10 00:00:15 1 0
2019-09-10 00:00:20 1 0
2019-09-10 00:00:25 1 0
2019-09-10 00:00:30 1 0
2019-09-10 00:00:35 1 0
2019-09-10 00:00:40 1 0
2019-09-10 00:00:45 1 0
FINALIZACION_PROF_FASE_2 REFERENCIA_PROF_FASE_2 ¥
DATE_TIME
2019-09-10 00:00:00 1 0
2019-09-10 00:00:05 1 0
2019-09-10 00:00:10 1 0
2019-09-10 00:00:15 1 0
2019-09-10 00:00:20 1 0
2019-09-10 00:00:25 1 0
2019-09-10 00:00:30 1 0
2019-09-10 00:00:35 1 0
2019-09-10 00:00:40 1 0
2019-09-10 00:00:45 1 0
FINALIZACION_PROF_FASE_3 REFERENCIA_PROF_FASE_3 ¥
DATE_TIME
2019-09-10 00:00:00 1 0
2019-09-10 00:00:05 1 0

2019-09-10 00:00:10 1 0

2019-09-10 00:00:15 1 0
2019-09-10 00:00:20 1 0
2019-09-10 00:00:25 1 0
2019-09-10 00:00:30 1 0
2019-09-10 00:00:35 1 0
2019-09-10 00:00:40 1 0
2019-09-10 00:00:45 1 0
FINALIZACION_PROF_FASE_4 REFERENCIA_PROF_FASE_4 ¥
DATE_TIME
2019-09-10 00:00:00 1 0
2019-09-10 00:00:05 1 0
2019-09-10 00:00:10 1 0
```

```
2019-09-10 00:00:15 1 0
2019-09-10 00:00:20 1 0
2019-09-10 00:00:25 1 0
2019-09-10 00:00:30 1 0
2019-09-10 00:00:35 1 0
2019-09-10 00:00:40 1 0
2019-09-10 00:00:45 1 0
FINALIZACION_PROF_FASE_5 REFERENCIA_PROF_FASE_5
DATE_TIME
2019-09-10 00:00:00 1 NaN
2019-09-10 00:00:05 1 NaN
2019-09-10 00:00:10 1 NaN
2019-09-10 00:00:15 1 NaN
2019-09-10 00:00:20 1 NaN
2019-09-10 00:00:25 1 NaN
2019-09-10 00:00:30 1 NaN
2019-09-10 00:00:35 1 NaN
2019-09-10 00:00:40 1 NaN
2019-09-10 00:00:45 1 NaN
```

[8]:

```
well_data_PCG_1277.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 286989 entries, 2019-09-10 00:00:00 to 2019-09-26 18:21:30
Data columns (total 33 columns):
# Column Non-Null Count Dtype
--- ------ -------------- -----
0 WELL_NAME 286989 non-null string
1 ACTIVITY_CODE 286989 non-null int64
2 MACROACTIVITY_CODE 258414 non-null string
3 C_DMEA 286989 non-null float64
4 BIT_DEPTH 286989 non-null float64
5 TD_TORQUE 286989 non-null float64
6 STP_PRS_1 286989 non-null float64
7 FLOW_IN 286989 non-null float64
8 BLOCK_POS 286989 non-null float64
9 HOOKLOAD_MAX 286989 non-null float64
10 CIA_PRESION_DIRECTA 0 non-null float64
11 CIA_DENSIDAD_FLUIDO 286989 non-null float64
12 CIA_CAUDAL 286989 non-null float64
13 CIA_VOLUMEN_ACUMULADO 286989 non-null float64
14 CIA_TIEMPO 286989 non-null float64
15 WIRELINE_PROFUNDIDAD 286989 non-null float64
16 WIRELINE_TENSION_DIFERENCIAL 286989 non-null float64
```

```
17 WIRELINE_TENSION_TOTAL 286989 non-null float64
18 WIRELINE_TIEMPO 286989 non-null float64
19 WIRELINE_VELOCIDAD 286989 non-null float64
20 WIRELINE_VOLTAJE 0 non-null float64
21 WIRELINE_INTENSIDAD 0 non-null float64
22 CIA_VOLUMEN_POR_ETAPAS 286989 non-null float64
23 FINALIZACION_PROF_FASE_1 286989 non-null int64
24 REFERENCIA_PROF_FASE_1 286989 non-null int64
25 FINALIZACION_PROF_FASE_2 286989 non-null int64
26 REFERENCIA_PROF_FASE_2 286989 non-null int64
27 FINALIZACION_PROF_FASE_3 286989 non-null int64
28 REFERENCIA_PROF_FASE_3 286989 non-null int64
29 FINALIZACION_PROF_FASE_4 286989 non-null int64
30 REFERENCIA_PROF_FASE_4 286989 non-null int64
31 FINALIZACION_PROF_FASE_5 286989 non-null int64
32 REFERENCIA_PROF_FASE_5 0 non-null float64
dtypes: float64(21), int64(10), string(2)
memory usage: 74.4 MB
```

Anexo 3 – Limpieza del Dataset

```
[9]:
selected_well_data = ␣
→well_data_PCG_1277[well_data_PCG_1277['MACROACTIVITY_CODE'] != ""]

[10]:
selected_well_data.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 258414 entries, 2019-09-10 00:00:00 to 2019-09-24 23:00:00
Data columns (total 33 columns):
# Column Non-Null Count Dtype
--- ------ -------------- -----
0 WELL_NAME 258414 non-null string
1 ACTIVITY_CODE 258414 non-null int64
2 MACROACTIVITY_CODE 258414 non-null string
3 C_DMEA 258414 non-null float64
4 BIT_DEPTH 258414 non-null float64
5 TD_TORQUE 258414 non-null float64
6 STP_PRS_1 258414 non-null float64
7 FLOW_IN 258414 non-null float64
8 BLOCK_POS 258414 non-null float64
9 HOOKLOAD_MAX 258414 non-null float64
10 CIA_PRESION_DIRECTA 0 non-null float64
11 CIA_DENSIDAD_FLUIDO 258414 non-null float64
12 CIA_CAUDAL 258414 non-null float64
13 CIA_VOLUMEN_ACUMULADO 258414 non-null float64
14 CIA_TIEMPO 258414 non-null float64
15 WIRELINE_PROFUNDIDAD 258414 non-null float64
16 WIRELINE_TENSION_DIFERENCIAL 258414 non-null float64
17 WIRELINE_TENSION_TOTAL 258414 non-null float64
18 WIRELINE_TIEMPO 258414 non-null float64
19 WIRELINE_VELOCIDAD 258414 non-null float64
20 WIRELINE_VOLTAJE 0 non-null float64
21 WIRELINE_INTENSIDAD 0 non-null float64
22 CIA_VOLUMEN_POR_ETAPAS 258414 non-null float64
23 FINALIZACION_PROF_FASE_1 258414 non-null int64
24 REFERENCIA_PROF_FASE_1 258414 non-null int64
25 FINALIZACION_PROF_FASE_2 258414 non-null int64
26 REFERENCIA_PROF_FASE_2 258414 non-null int64
27 FINALIZACION_PROF_FASE_3 258414 non-null int64
28 REFERENCIA_PROF_FASE_3 258414 non-null int64
29 FINALIZACION_PROF_FASE_4 258414 non-null int64
30 REFERENCIA_PROF_FASE_4 258414 non-null int64
```

```
31 FINALIZACION_PROF_FASE_5 258414 non-null int64
32 REFERENCIA_PROF_FASE_5 0 non-null float64
dtypes: float64(21), int64(10), string(2)
memory usage: 67.0 MB
```

Eliminar la columna WELL_NAME

```
[11]:
selected_well_data = selected_well_data.drop('WELL_NAME', axis = 1)
selected_well_data.head(10)
```

```
[11]: ACTIVITY_CODE MACROACTIVITY_CODE C_DMEA BIT_DEPTH ¥
DATE_TIME
2019-09-10 00:00:00 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:05 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:10 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:15 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:20 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:25 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:30 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:35 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:40 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:45 0 PERFORANDO 0.0 0.0
TD_TORQUE STP_PRS_1 FLOW_IN BLOCK_POS HOOKLOAD_MAX ¥
DATE_TIME
2019-09-10 00:00:00 0.000000 0.0 0.0 -0.844452 21.44336
2019-09-10 00:00:05 0.000000 0.0 0.0 -0.835339 21.44336
2019-09-10 00:00:10 3.976276 0.0 0.0 -0.860829 21.35895
2019-09-10 00:00:15 4.732082 0.0 0.0 -0.836936 21.39916
2019-09-10 00:00:20 0.000000 0.0 0.0 -0.830801 21.39916
2019-09-10 00:00:25 1.112243 0.0 0.0 -0.844519 21.18597
2019-09-10 00:00:30 0.000000 0.0 0.0 -0.827136 21.40165
2019-09-10 00:00:35 1.476786 0.0 0.0 -0.850366 21.37076
2019-09-10 00:00:40 0.000000 0.0 0.0 -0.818154 21.54548
2019-09-10 00:00:45 0.000000 0.0 0.0 -0.873479 21.66736
CIA_PRESION_DIRECTA CIA_DENSIDAD_FLUIDO CIA_CAUDAL ¥
DATE_TIME
2019-09-10 00:00:00 NaN 0.0 0.0
2019-09-10 00:00:05 NaN 0.0 0.0
2019-09-10 00:00:10 NaN 0.0 0.0
2019-09-10 00:00:15 NaN 0.0 0.0
2019-09-10 00:00:20 NaN 0.0 0.0
```

```
2019-09-10 00:00:25 NaN 0.0 0.0
2019-09-10 00:00:30 NaN 0.0 0.0
2019-09-10 00:00:35 NaN 0.0 0.0
2019-09-10 00:00:40 NaN 0.0 0.0
2019-09-10 00:00:45 NaN 0.0 0.0
CIA_VOLUMEN_ACUMULADO CIA_TIEMPO WIRELINE_PROFUNDIDAD ¥
DATE_TIME
2019-09-10 00:00:00 0.0 0.0 0.0
2019-09-10 00:00:05 0.0 0.0 0.0
2019-09-10 00:00:10 0.0 0.0 0.0
2019-09-10 00:00:15 0.0 0.0 0.0
2019-09-10 00:00:20 0.0 0.0 0.0
2019-09-10 00:00:25 0.0 0.0 0.0
2019-09-10 00:00:30 0.0 0.0 0.0
2019-09-10 00:00:35 0.0 0.0 0.0
2019-09-10 00:00:40 0.0 0.0 0.0
2019-09-10 00:00:45 0.0 0.0 0.0
WIRELINE_TENSION_DIFERENCIAL WIRELINE_TENSION_TOTAL ¥
DATE_TIME
2019-09-10 00:00:00 0.0 0.0
2019-09-10 00:00:05 0.0 0.0
2019-09-10 00:00:10 0.0 0.0
2019-09-10 00:00:15 0.0 0.0
2019-09-10 00:00:20 0.0 0.0
2019-09-10 00:00:25 0.0 0.0
2019-09-10 00:00:30 0.0 0.0
2019-09-10 00:00:35 0.0 0.0
2019-09-10 00:00:40 0.0 0.0
2019-09-10 00:00:45 0.0 0.0
WIRELINE_TIEMPO WIRELINE_VELOCIDAD WIRELINE_VOLTAJE ¥
DATE_TIME
2019-09-10 00:00:00 0.0 0.0 NaN
2019-09-10 00:00:05 0.0 0.0 NaN
2019-09-10 00:00:10 0.0 0.0 NaN
2019-09-10 00:00:15 0.0 0.0 NaN
2019-09-10 00:00:20 0.0 0.0 NaN
2019-09-10 00:00:25 0.0 0.0 NaN
2019-09-10 00:00:30 0.0 0.0 NaN
2019-09-10 00:00:35 0.0 0.0 NaN
2019-09-10 00:00:40 0.0 0.0 NaN
2019-09-10 00:00:45 0.0 0.0 NaN
WIRELINE_INTENSIDAD CIA_VOLUMEN_POR_ETAPAS ¥
DATE_TIME
2019-09-10 00:00:00 NaN 0.0
2019-09-10 00:00:05 NaN 0.0
2019-09-10 00:00:10 NaN 0.0
```

```
2019-09-10 00:00:15 NaN 0.0
2019-09-10 00:00:20 NaN 0.0
2019-09-10 00:00:25 NaN 0.0
2019-09-10 00:00:30 NaN 0.0
2019-09-10 00:00:35 NaN 0.0
2019-09-10 00:00:40 NaN 0.0
2019-09-10 00:00:45 NaN 0.0
FINALIZACION_PROF_FASE_1 REFERENCIA_PROF_FASE_1 ¥
DATE_TIME
2019-09-10 00:00:00 1 0
2019-09-10 00:00:05 1 0
2019-09-10 00:00:10 1 0
2019-09-10 00:00:15 1 0
2019-09-10 00:00:20 1 0
2019-09-10 00:00:25 1 0
2019-09-10 00:00:30 1 0


2019-09-10 00:00:35 1 0
2019-09-10 00:00:40 1 0
2019-09-10 00:00:45 1 0
FINALIZACION_PROF_FASE_2 REFERENCIA_PROF_FASE_2 ¥
DATE_TIME
2019-09-10 00:00:00 1 0
2019-09-10 00:00:05 1 0
2019-09-10 00:00:10 1 0
2019-09-10 00:00:15 1 0
2019-09-10 00:00:20 1 0
2019-09-10 00:00:25 1 0
2019-09-10 00:00:30 1 0
2019-09-10 00:00:35 1 0
2019-09-10 00:00:40 1 0
2019-09-10 00:00:45 1 0
FINALIZACION_PROF_FASE_3 REFERENCIA_PROF_FASE_3 ¥
DATE_TIME
2019-09-10 00:00:00 1 0
2019-09-10 00:00:05 1 0
2019-09-10 00:00:10 1 0
2019-09-10 00:00:15 1 0
2019-09-10 00:00:20 1 0
2019-09-10 00:00:25 1 0
2019-09-10 00:00:30 1 0
2019-09-10 00:00:35 1 0
2019-09-10 00:00:40 1 0
2019-09-10 00:00:45 1 0
FINALIZACION_PROF_FASE_4 REFERENCIA_PROF_FASE_4 ¥
```

```
DATE_TIME
2019-09-10 00:00:00 1 0
2019-09-10 00:00:05 1 0
2019-09-10 00:00:10 1 0
2019-09-10 00:00:15 1 0
2019-09-10 00:00:20 1 0
2019-09-10 00:00:25 1 0
2019-09-10 00:00:30 1 0
2019-09-10 00:00:35 1 0
2019-09-10 00:00:40 1 0
2019-09-10 00:00:45 1 0
FINALIZACION_PROF_FASE_5 REFERENCIA_PROF_FASE_5
DATE_TIME
2019-09-10 00:00:00 1 NaN
2019-09-10 00:00:05 1 NaN
2019-09-10 00:00:10 1 NaN
2019-09-10 00:00:15 1 NaN
2019-09-10 00:00:20 1 NaN
2019-09-10 00:00:25 1 NaN
2019-09-10 00:00:30 1 NaN
2019-09-10 00:00:35 1 NaN
2019-09-10 00:00:40 1 NaN
2019-09-10 00:00:45 1 NaN
```

[12]:

```
selected_well_data.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 258414 entries, 2019-09-10 00:00:00 to 2019-09-24 23:00:00
Data columns (total 32 columns):
# Column Non-Null Count Dtype
--- ------ -------------- -----
0 ACTIVITY_CODE 258414 non-null int64
1 MACROACTIVITY_CODE 258414 non-null string
2 C_DMEA 258414 non-null float64
3 BIT_DEPTH 258414 non-null float64
4 TD_TORQUE 258414 non-null float64
5 STP_PRS_1 258414 non-null float64
6 FLOW_IN 258414 non-null float64
7 BLOCK_POS 258414 non-null float64
8 HOOKLOAD_MAX 258414 non-null float64
9 CIA_PRESION_DIRECTA 0 non-null float64
10 CIA_DENSIDAD_FLUIDO 258414 non-null float64
11 CIA_CAUDAL 258414 non-null float64
12 CIA_VOLUMEN_ACUMULADO 258414 non-null float64
```

```
13 CIA_TIEMPO 258414 non-null float64
14 WIRELINE_PROFUNDIDAD 258414 non-null float64
15 WIRELINE_TENSION_DIFERENCIAL 258414 non-null float64
16 WIRELINE_TENSION_TOTAL 258414 non-null float64
17 WIRELINE_TIEMPO 258414 non-null float64
18 WIRELINE_VELOCIDAD 258414 non-null float64
19 WIRELINE_VOLTAJE 0 non-null float64
20 WIRELINE_INTENSIDAD 0 non-null float64
21 CIA_VOLUMEN_POR_ETAPAS 258414 non-null float64
22 FINALIZACION_PROF_FASE_1 258414 non-null int64
23 REFERENCIA_PROF_FASE_1 258414 non-null int64
24 FINALIZACION_PROF_FASE_2 258414 non-null int64
25 REFERENCIA_PROF_FASE_2 258414 non-null int64
26 FINALIZACION_PROF_FASE_3 258414 non-null int64
27 REFERENCIA_PROF_FASE_3 258414 non-null int64
28 FINALIZACION_PROF_FASE_4 258414 non-null int64
29 REFERENCIA_PROF_FASE_4 258414 non-null int64
30 FINALIZACION_PROF_FASE_5 258414 non-null int64
31 REFERENCIA_PROF_FASE_5 0 non-null float64
dtypes: float64(21), int64(10), string(1)
memory usage: 65.1 MB
```

Eliminar las columnas nulas

```
[13]:
selected_well_data=selected_well_data.dropna(axis=1,how='all')
```

```
[14]:
selected_well_data.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 258414 entries, 2019-09-10 00:00:00 to 2019-09-24 23:00:00
Data columns (total 28 columns):
# Column Non-Null Count Dtype
--- ------ -------------- -----
0 ACTIVITY_CODE 258414 non-null int64
1 MACROACTIVITY_CODE 258414 non-null string
2 C_DMEA 258414 non-null float64
3 BIT_DEPTH 258414 non-null float64
4 TD_TORQUE 258414 non-null float64
5 STP_PRS_1 258414 non-null float64
6 FLOW_IN 258414 non-null float64
7 BLOCK_POS 258414 non-null float64
8 HOOKLOAD_MAX 258414 non-null float64
9 CIA_DENSIDAD_FLUIDO 258414 non-null float64
10 CIA_CAUDAL 258414 non-null float64
```

```
11 CIA_VOLUMEN_ACUMULADO 258414 non-null float64
12 CIA_TIEMPO 258414 non-null float64
13 WIRELINE_PROFUNDIDAD 258414 non-null float64
14 WIRELINE_TENSION_DIFERENCIAL 258414 non-null float64
15 WIRELINE_TENSION_TOTAL 258414 non-null float64
16 WIRELINE_TIEMPO 258414 non-null float64
17 WIRELINE_VELOCIDAD 258414 non-null float64
18 CIA_VOLUMEN_POR_ETAPAS 258414 non-null float64
19 FINALIZACION_PROF_FASE_1 258414 non-null int64
20 REFERENCIA_PROF_FASE_1 258414 non-null int64
21 FINALIZACION_PROF_FASE_2 258414 non-null int64
22 REFERENCIA_PROF_FASE_2 258414 non-null int64
23 FINALIZACION_PROF_FASE_3 258414 non-null int64
24 REFERENCIA_PROF_FASE_3 258414 non-null int64
25 FINALIZACION_PROF_FASE_4 258414 non-null int64
26 REFERENCIA_PROF_FASE_4 258414 non-null int64
27 FINALIZACION_PROF_FASE_5 258414 non-null int64
dtypes: float64(17), int64(10), string(1)
memory usage: 57.2 MB
```

Verificar cantidad y tipo de registros de la variable a predecir "MACROACTIVITY_CODE"

```
[15]:
pd.crosstab(index=selected_well_data['MACROACTIVITY_CODE'], columns='count',
rownames=['MACROACTIVITY_CODE'], colnames=['CANTIDAD'])
```

```
[15]:
CANTIDAD count
MACROACTIVITY_CODE
CALIBRANDO 13121
CASING RUNNING 25000
CEMENTANDO 4497
PERFILANDO 14760
PERFORANDO 201036
```

```
[16]:
plt.figure(figsize=(12, 8))
ax = sns.countplot(x='MACROACTIVITY_CODE', data=selected_well_data,
order = selected_well_data['MACROACTIVITY_CODE'].
↪value_counts().index)
ax.set(ylabel='CANTIDAD')
for p in ax.patches:
ax.annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_width() / 2.,␣
↪p.get_height()),
ha = 'center', va = 'center', xytext = (0, 10), textcoords =␣
↪'offset points')
```

Analizar los valores que pueden tomar las variables

[17]:
```
selected_well_data.describe(include='all').T
```

[17]:
```
count unique top freq mean ¥
ACTIVITY_CODE 258414 NaN NaN NaN 17.2615
MACROACTIVITY_CODE 258414 5 PERFORANDO 201036 NaN
C_DMEA 258414 NaN NaN NaN 1748.15
BIT_DEPTH 258414 NaN NaN NaN 1264.48
TD_TORQUE 258414 NaN NaN NaN 1069.97
STP_PRS_1 258414 NaN NaN NaN 723.644
FLOW_IN 258414 NaN NaN NaN 188.668
BLOCK_POS 258414 NaN NaN NaN 5.97034
HOOKLOAD_MAX 258414 NaN NaN NaN 93.304
CIA_DENSIDAD_FLUIDO 258414 NaN NaN NaN 0.220482
CIA_CAUDAL 258414 NaN NaN NaN 0.0399972
CIA_VOLUMEN_ACUMULADO 258414 NaN NaN NaN 5.62808
CIA_TIEMPO 258414 NaN NaN NaN 3.47349
WIRELINE_PROFUNDIDAD 258414 NaN NaN NaN 68.2056
WIRELINE_TENSION_DIFERENCIAL 258414 NaN NaN NaN -19.7499
WIRELINE_TENSION_TOTAL 258414 NaN NaN NaN 150.028
WIRELINE_TIEMPO 258414 NaN NaN NaN 1229.25
WIRELINE_VELOCIDAD 258414 NaN NaN NaN 41.8373


CIA_VOLUMEN_POR_ETAPAS 258414 NaN NaN NaN 1.79122
FINALIZACION_PROF_FASE_1 258414 NaN NaN NaN 1
REFERENCIA_PROF_FASE_1 258414 NaN NaN NaN 0
FINALIZACION_PROF_FASE_2 258414 NaN NaN NaN 1
REFERENCIA_PROF_FASE_2 258414 NaN NaN NaN 0
FINALIZACION_PROF_FASE_3 258414 NaN NaN NaN 1
REFERENCIA_PROF_FASE_3 258414 NaN NaN NaN 0
FINALIZACION_PROF_FASE_4 258414 NaN NaN NaN 1
REFERENCIA_PROF_FASE_4 258414 NaN NaN NaN 0
FINALIZACION_PROF_FASE_5 258414 NaN NaN NaN 1
std min 25% 50% ¥
ACTIVITY_CODE 25.2763 0 0 0
MACROACTIVITY_CODE NaN NaN NaN NaN
C_DMEA 641.418 0 1546.93 2157.41
BIT_DEPTH 748.556 0 560.591 1550.24
TD_TORQUE 1857.57 0 0 10.5731
STP_PRS_1 878.356 0 7.30293 155.979
```

FLOW_IN 204.415 0 0.28068 50.2242
BLOCK_POS 5.72255 -13.6431 0.477527 4.65834
HOOKLOAD_MAX 49.6451 0 27.2766 109.101
CIA_DENSIDAD_FLUIDO 1.48174 0 0 0
CIA_CAUDAL 0.425234 0 0 0
CIA_VOLUMEN_ACUMULADO 49.6776 0 0 0
CIA_TIEMPO 28.9791 0 0 0
WIRELINE_PROFUNDIDAD 8820.47 0 0 0
WIRELINE_TENSION_DIFERENCIAL 31652.9 -9.11301e+06 0 0
WIRELINE_TENSION_TOTAL 605.437 0 0 0
WIRELINE_TIEMPO 11633.6 0 0 0
WIRELINE_VELOCIDAD 4875.58 0 0 0
CIA_VOLUMEN_POR_ETAPAS 16.9784 0 0 0
FINALIZACION_PROF_FASE_1 0 1 1 1
REFERENCIA_PROF_FASE_1 0 0 0 0
FINALIZACION_PROF_FASE_2 0 1 1 1
REFERENCIA_PROF_FASE_2 0 0 0 0
FINALIZACION_PROF_FASE_3 0 1 1 1
REFERENCIA_PROF_FASE_3 0 0 0 0
FINALIZACION_PROF_FASE_4 0 1 1 1
REFERENCIA_PROF_FASE_4 0 0 0 0
FINALIZACION_PROF_FASE_5 0 1 1 1
75% max
ACTIVITY_CODE 31 99
MACROACTIVITY_CODE NaN NaN
C_DMEA 2194.43 2248.63
BIT_DEPTH 1826.37 2248.61
TD_TORQUE 1288.78 23294.4

STP_PRS_1 1306.62 3343
FLOW_IN 381.182 606.058
BLOCK_POS 11.7751 16.6902
HOOKLOAD_MAX 135.672 213.243
CIA_DENSIDAD_FLUIDO 0 14.73
CIA_CAUDAL 0 8.04
CIA_VOLUMEN_ACUMULADO 0 559.21
CIA_TIEMPO 0 374.383
WIRELINE_PROFUNDIDAD 0 4.47909e+06
WIRELINE_TENSION_DIFERENCIAL 0 9.11266e+06
WIRELINE_TENSION_TOTAL 0 4991.33
WIRELINE_TIEMPO 0 3.29133e+06
WIRELINE_VELOCIDAD 0 791277
CIA_VOLUMEN_POR_ETAPAS 0 216.74
FINALIZACION_PROF_FASE_1 1 1
REFERENCIA_PROF_FASE_1 0 0

```
FINALIZACION_PROF_FASE_2 1 1
REFERENCIA_PROF_FASE_2 0 0
FINALIZACION_PROF_FASE_3 1 1
REFERENCIA_PROF_FASE_3 0 0
FINALIZACION_PROF_FASE_4 1 1
REFERENCIA_PROF_FASE_4 0 0
FINALIZACION_PROF_FASE_5 1 1
```

Melt del dataset para visualizarlo en un boxplot

[18]:
```
data_melt = pd.melt(selected_well_data.drop(["MACROACTIVITY_CODE"], axis=1));
␣
,→data_melt
```

[18]:
```
variable value
0 ACTIVITY_CODE 0.0
1 ACTIVITY_CODE 0.0
2 ACTIVITY_CODE 0.0
3 ACTIVITY_CODE 0.0
4 ACTIVITY_CODE 0.0
... ... ...
6977173 FINALIZACION_PROF_FASE_5 1.0
6977174 FINALIZACION_PROF_FASE_5 1.0
6977175 FINALIZACION_PROF_FASE_5 1.0
6977176 FINALIZACION_PROF_FASE_5 1.0
6977177 FINALIZACION_PROF_FASE_5 1.0

[6977178 rows x 2 columns]
```

[19]:
```
sns.set(style="ticks")
f, ax = plt.subplots(figsize=(9, 8))
ax.set_xscale("log")
sns.boxplot(x="value", y="variable", data=data_melt,
whis=[0, 100], width=.6, palette="vlag")
```

[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8d6fa3f6a0>
```

[20]:

```
sns.set(style="ticks")
f, ax = plt.subplots(figsize=(12, 8))
ax.set_xscale("log")
sns.boxplot(x="value", y="variable", data=data_melt,
whis=[0, 100], width=.6, palette="vlag")
sns.stripplot(x="value", y="variable", data=data_melt,
size=4, color=".3", linewidth=0)
ax.xaxis.grid(True)
ax.set(ylabel="")
sns.despine(trim=True, left=True)
```

Eliminar columnas Finalización y Referencia Prof Fase 1, 2, 3, 4 y 5

```
[21]:
selected_well_data = selected_well_data.iloc[:,:-9]
selected_well_data.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 258414 entries, 2019-09-10 00:00:00 to 2019-09-24 23:00:00
Data columns (total 19 columns):
# Column Non-Null Count Dtype
--- ------ -------------- -----
0 ACTIVITY_CODE 258414 non-null int64
1 MACROACTIVITY_CODE 258414 non-null string
2 C_DMEA 258414 non-null float64
3 BIT_DEPTH 258414 non-null float64
4 TD_TORQUE 258414 non-null float64
5 STP_PRS_1 258414 non-null float64
6 FLOW_IN 258414 non-null float64
7 BLOCK_POS 258414 non-null float64
8 HOOKLOAD_MAX 258414 non-null float64
9 CIA_DENSIDAD_FLUIDO 258414 non-null float64
10 CIA_CAUDAL 258414 non-null float64
11 CIA_VOLUMEN_ACUMULADO 258414 non-null float64
12 CIA_TIEMPO 258414 non-null float64
13 WIRELINE_PROFUNDIDAD 258414 non-null float64
14 WIRELINE_TENSION_DIFERENCIAL 258414 non-null float64
15 WIRELINE_TENSION_TOTAL 258414 non-null float64
16 WIRELINE_TIEMPO 258414 non-null float64
17 WIRELINE_VELOCIDAD 258414 non-null float64
18 CIA_VOLUMEN_POR_ETAPAS 258414 non-null float64
dtypes: float64(17), int64(1), string(1)
memory usage: 49.4 MB
```

Realizar el encoding de la variable categórica "MACROACTIVITY_CODE"

[22]:
```
encoder = LabelEncoder()
y = encoder.fit_transform(selected_well_data['MACROACTIVITY_CODE'])
selected_well_data['MACROACTIVITY_CODE_LABEL'] = pd.Series(y, index =␣
↪selected_well_data.index)
```

[23]:
```
selected_well_data.groupby("MACROACTIVITY_CODE_LABEL").size()
```

[23]:
```
MACROACTIVITY_CODE_LABEL
0 13121
1 25000
2 4497
3 14760
4 201036
dtype: int64
```

[24]:
```
selected_well_data.head(10)
```

[24]:
```
ACTIVITY_CODE MACROACTIVITY_CODE C_DMEA BIT_DEPTH ¥
DATE_TIME
2019-09-10 00:00:00 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:05 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:10 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:15 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:20 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:25 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:30 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:35 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:40 0 PERFORANDO 0.0 0.0
2019-09-10 00:00:45 0 PERFORANDO 0.0 0.0
TD_TORQUE STP_PRS_1 FLOW_IN BLOCK_POS HOOKLOAD_MAX ¥
DATE_TIME
2019-09-10 00:00:00 0.000000 0.0 0.0 -0.844452 21.44336
2019-09-10 00:00:05 0.000000 0.0 0.0 -0.835339 21.44336
2019-09-10 00:00:10 3.976276 0.0 0.0 -0.860829 21.35895
2019-09-10 00:00:15 4.732082 0.0 0.0 -0.836936 21.39916
2019-09-10 00:00:20 0.000000 0.0 0.0 -0.830801 21.39916
2019-09-10 00:00:25 1.112243 0.0 0.0 -0.844519 21.18597
2019-09-10 00:00:30 0.000000 0.0 0.0 -0.827136 21.40165
2019-09-10 00:00:35 1.476786 0.0 0.0 -0.850366 21.37076
```

```
2019-09-10 00:00:40 0.000000 0.0 0.0 -0.818154 21.54548
2019-09-10 00:00:45 0.000000 0.0 0.0 -0.873479 21.66736
CIA_DENSIDAD_FLUIDO CIA_CAUDAL CIA_VOLUMEN_ACUMULADO ¥
DATE_TIME
2019-09-10 00:00:00 0.0 0.0 0.0
2019-09-10 00:00:05 0.0 0.0 0.0
2019-09-10 00:00:10 0.0 0.0 0.0
2019-09-10 00:00:15 0.0 0.0 0.0
2019-09-10 00:00:20 0.0 0.0 0.0
2019-09-10 00:00:25 0.0 0.0 0.0
2019-09-10 00:00:30 0.0 0.0 0.0
2019-09-10 00:00:35 0.0 0.0 0.0
2019-09-10 00:00:40 0.0 0.0 0.0
2019-09-10 00:00:45 0.0 0.0 0.0
CIA_TIEMPO WIRELINE_PROFUNDIDAD ¥
DATE_TIME
2019-09-10 00:00:00 0.0 0.0
2019-09-10 00:00:05 0.0 0.0
2019-09-10 00:00:10 0.0 0.0
2019-09-10 00:00:15 0.0 0.0
2019-09-10 00:00:20 0.0 0.0
2019-09-10 00:00:25 0.0 0.0
2019-09-10 00:00:30 0.0 0.0
2019-09-10 00:00:35 0.0 0.0
2019-09-10 00:00:40 0.0 0.0
2019-09-10 00:00:45 0.0 0.0
WIRELINE_TENSION_DIFERENCIAL WIRELINE_TENSION_TOTAL ¥
DATE_TIME
2019-09-10 00:00:00 0.0 0.0
2019-09-10 00:00:05 0.0 0.0
2019-09-10 00:00:10 0.0 0.0
2019-09-10 00:00:15 0.0 0.0
2019-09-10 00:00:20 0.0 0.0
2019-09-10 00:00:25 0.0 0.0
2019-09-10 00:00:30 0.0 0.0
2019-09-10 00:00:35 0.0 0.0
2019-09-10 00:00:40 0.0 0.0
2019-09-10 00:00:45 0.0 0.0
WIRELINE_TIEMPO WIRELINE_VELOCIDAD ¥
DATE_TIME
2019-09-10 00:00:00 0.0 0.0
2019-09-10 00:00:05 0.0 0.0
2019-09-10 00:00:10 0.0 0.0
2019-09-10 00:00:15 0.0 0.0
2019-09-10 00:00:20 0.0 0.0
2019-09-10 00:00:25 0.0 0.0
```

```
2019-09-10 00:00:30 0.0 0.0
2019-09-10 00:00:35 0.0 0.0
2019-09-10 00:00:40 0.0 0.0
2019-09-10 00:00:45 0.0 0.0
CIA_VOLUMEN_POR_ETAPAS MACROACTIVITY_CODE_LABEL
DATE_TIME
2019-09-10 00:00:00 0.0 4
2019-09-10 00:00:05 0.0 4
2019-09-10 00:00:10 0.0 4
2019-09-10 00:00:15 0.0 4
2019-09-10 00:00:20 0.0 4
2019-09-10 00:00:25 0.0 4
2019-09-10 00:00:30 0.0 4
2019-09-10 00:00:35 0.0 4
2019-09-10 00:00:40 0.0 4
2019-09-10 00:00:45 0.0 4
```

[25]:

```
selected_well_data.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 258414 entries, 2019-09-10 00:00:00 to 2019-09-24 23:00:00
Data columns (total 20 columns):
# Column Non-Null Count Dtype
--- ------ -------------- -----
0 ACTIVITY_CODE 258414 non-null int64
1 MACROACTIVITY_CODE 258414 non-null string
2 C_DMEA 258414 non-null float64
3 BIT_DEPTH 258414 non-null float64
4 TD_TORQUE 258414 non-null float64
5 STP_PRS_1 258414 non-null float64
6 FLOW_IN 258414 non-null float64
7 BLOCK_POS 258414 non-null float64
8 HOOKLOAD_MAX 258414 non-null float64
9 CIA_DENSIDAD_FLUIDO 258414 non-null float64
10 CIA_CAUDAL 258414 non-null float64
11 CIA_VOLUMEN_ACUMULADO 258414 non-null float64
12 CIA_TIEMPO 258414 non-null float64
13 WIRELINE_PROFUNDIDAD 258414 non-null float64
14 WIRELINE_TENSION_DIFERENCIAL 258414 non-null float64
15 WIRELINE_TENSION_TOTAL 258414 non-null float64
16 WIRELINE_TIEMPO 258414 non-null float64
17 WIRELINE_VELOCIDAD 258414 non-null float64
18 CIA_VOLUMEN_POR_ETAPAS 258414 non-null float64
19 MACROACTIVITY_CODE_LABEL 258414 non-null int64
dtypes: float64(17), int64(2), string(1)
memory usage: 51.4 MB
```

Remover la variable categórica "MACROACTIVITY_CODE" ya codificada

```
[26]:
clean_well_data = selected_well_data.drop('MACROACTIVITY_CODE', axis = 1)
clean_well_data.head(10)
```

```
[26]:
ACTIVITY_CODE C_DMEA BIT_DEPTH TD_TORQUE STP_PRS_1 ¥
DATE_TIME
2019-09-10 00:00:00 0 0.0 0.0 0.000000 0.0
2019-09-10 00:00:05 0 0.0 0.0 0.000000 0.0
2019-09-10 00:00:10 0 0.0 0.0 3.976276 0.0
2019-09-10 00:00:15 0 0.0 0.0 4.732082 0.0
2019-09-10 00:00:20 0 0.0 0.0 0.000000 0.0
2019-09-10 00:00:25 0 0.0 0.0 1.112243 0.0
2019-09-10 00:00:30 0 0.0 0.0 0.000000 0.0
2019-09-10 00:00:35 0 0.0 0.0 1.476786 0.0
2019-09-10 00:00:40 0 0.0 0.0 0.000000 0.0
2019-09-10 00:00:45 0 0.0 0.0 0.000000 0.0
FLOW_IN BLOCK_POS HOOKLOAD_MAX CIA_DENSIDAD_FLUIDO ¥
DATE_TIME
2019-09-10 00:00:00 0.0 -0.844452 21.44336 0.0
2019-09-10 00:00:05 0.0 -0.835339 21.44336 0.0
2019-09-10 00:00:10 0.0 -0.860829 21.35895 0.0
2019-09-10 00:00:15 0.0 -0.836936 21.39916 0.0
2019-09-10 00:00:20 0.0 -0.830801 21.39916 0.0
2019-09-10 00:00:25 0.0 -0.844519 21.18597 0.0
2019-09-10 00:00:30 0.0 -0.827136 21.40165 0.0
2019-09-10 00:00:35 0.0 -0.850366 21.37076 0.0
2019-09-10 00:00:40 0.0 -0.818154 21.54548 0.0
2019-09-10 00:00:45 0.0 -0.873479 21.66736 0.0
CIA_CAUDAL CIA_VOLUMEN_ACUMULADO CIA_TIEMPO ¥
DATE_TIME
2019-09-10 00:00:00 0.0 0.0 0.0
2019-09-10 00:00:05 0.0 0.0 0.0
2019-09-10 00:00:10 0.0 0.0 0.0
2019-09-10 00:00:15 0.0 0.0 0.0
2019-09-10 00:00:20 0.0 0.0 0.0
2019-09-10 00:00:25 0.0 0.0 0.0
2019-09-10 00:00:30 0.0 0.0 0.0
2019-09-10 00:00:35 0.0 0.0 0.0
2019-09-10 00:00:40 0.0 0.0 0.0
2019-09-10 00:00:45 0.0 0.0 0.0
WIRELINE_PROFUNDIDAD WIRELINE_TENSION_DIFERENCIAL ¥
```

```
DATE_TIME
2019-09-10 00:00:00 0.0 0.0
2019-09-10 00:00:05 0.0 0.0
2019-09-10 00:00:10 0.0 0.0
2019-09-10 00:00:15 0.0 0.0
2019-09-10 00:00:20 0.0 0.0
2019-09-10 00:00:25 0.0 0.0
2019-09-10 00:00:30 0.0 0.0
2019-09-10 00:00:35 0.0 0.0
2019-09-10 00:00:40 0.0 0.0
2019-09-10 00:00:45 0.0 0.0
WIRELINE_TENSION_TOTAL WIRELINE_TIEMPO ¥
DATE_TIME
2019-09-10 00:00:00 0.0 0.0
2019-09-10 00:00:05 0.0 0.0
2019-09-10 00:00:10 0.0 0.0
2019-09-10 00:00:15 0.0 0.0
2019-09-10 00:00:20 0.0 0.0
2019-09-10 00:00:25 0.0 0.0
2019-09-10 00:00:30 0.0 0.0
2019-09-10 00:00:35 0.0 0.0
2019-09-10 00:00:40 0.0 0.0
2019-09-10 00:00:45 0.0 0.0
WIRELINE_VELOCIDAD CIA_VOLUMEN_POR_ETAPAS ¥
DATE_TIME
2019-09-10 00:00:00 0.0 0.0
2019-09-10 00:00:05 0.0 0.0
2019-09-10 00:00:10 0.0 0.0
2019-09-10 00:00:15 0.0 0.0
2019-09-10 00:00:20 0.0 0.0
2019-09-10 00:00:25 0.0 0.0
2019-09-10 00:00:30 0.0 0.0
2019-09-10 00:00:35 0.0 0.0
2019-09-10 00:00:40 0.0 0.0
2019-09-10 00:00:45 0.0 0.0
MACROACTIVITY_CODE_LABEL
DATE_TIME

2019-09-10 00:00:00 4

2019-09-10 00:00:05 4
2019-09-10 00:00:10 4
2019-09-10 00:00:15 4
2019-09-10 00:00:20 4
2019-09-10 00:00:25 4
2019-09-10 00:00:30 4
2019-09-10 00:00:35 4
2019-09-10 00:00:40 4
```

```
2019-09-10 00:00:45 4
```

[27]:

```
clean_well_data.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 258414 entries, 2019-09-10 00:00:00 to 2019-09-24 23:00:00
Data columns (total 19 columns):
# Column Non-Null Count Dtype
--- ------ -------------- -----
0 ACTIVITY_CODE 258414 non-null int64
1 C_DMEA 258414 non-null float64
2 BIT_DEPTH 258414 non-null float64
3 TD_TORQUE 258414 non-null float64
4 STP_PRS_1 258414 non-null float64
5 FLOW_IN 258414 non-null float64
6 BLOCK_POS 258414 non-null float64
7 HOOKLOAD_MAX 258414 non-null float64
8 CIA_DENSIDAD_FLUIDO 258414 non-null float64
9 CIA_CAUDAL 258414 non-null float64
10 CIA_VOLUMEN_ACUMULADO 258414 non-null float64
11 CIA_TIEMPO 258414 non-null float64
12 WIRELINE_PROFUNDIDAD 258414 non-null float64
13 WIRELINE_TENSION_DIFERENCIAL 258414 non-null float64
14 WIRELINE_TENSION_TOTAL 258414 non-null float64
15 WIRELINE_TIEMPO 258414 non-null float64
16 WIRELINE_VELOCIDAD 258414 non-null float64
17 CIA_VOLUMEN_POR_ETAPAS 258414 non-null float64
18 MACROACTIVITY_CODE_LABEL 258414 non-null int64
dtypes: float64(17), int64(2)
memory usage: 49.4 MB
```

Estudiar Correlación

[28]:

```
plt.figure(figsize=(14, 14))
sns.heatmap(clean_well_data.corr(), cmap='YlGnBu', annot=True, linewidths=.5,
␣
,→fmt='.2f')
```

[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8d7573f7f0>
```

Anexo 4 – Preparación de Datos para el Modelado

Definir las variables de entrada y la variable de salida (variable a predecir)

```
[29]:
Y = clean_well_data['MACROACTIVITY_CODE_LABEL'];
X = clean_well_data.drop('MACROACTIVITY_CODE_LABEL', axis =1)
```

```
[29]:
ACTIVITY_CODE C_DMEA BIT_DEPTH TD_TORQUE STP_PRS_1 ¥
DATE_TIME
2019-09-10 00:00:00 0 0.0 0.000 0.000000 0.0
2019-09-10 00:00:05 0 0.0 0.000 0.000000 0.0
2019-09-10 00:00:10 0 0.0 0.000 3.976276 0.0
2019-09-10 00:00:15 0 0.0 0.000 4.732082 0.0
2019-09-10 00:00:20 0 0.0 0.000 0.000000 0.0
... ... ... ... ... ...
2019-09-24 22:59:40 66 2240.0 2237.936 0.000000 0.0
2019-09-24 22:59:45 66 2240.0 2237.965 0.000000 0.0
2019-09-24 22:59:50 61 2240.0 2237.950 0.000000 0.0
2019-09-24 22:59:55 61 2240.0 2237.951 0.000000 0.0
2019-09-24 23:00:00 66 2240.0 2237.929 1.660728 0.0
FLOW_IN BLOCK_POS HOOKLOAD_MAX CIA_DENSIDAD_FLUIDO ¥
DATE_TIME
2019-09-10 00:00:00 0.000000 -0.844452 21.44336 0.00
2019-09-10 00:00:05 0.000000 -0.835339 21.44336 0.00
2019-09-10 00:00:10 0.000000 -0.860829 21.35895 0.00
2019-09-10 00:00:15 0.000000 -0.836936 21.39916 0.00
2019-09-10 00:00:20 0.000000 -0.830801 21.39916 0.00
... ... ... ... ...
2019-09-24 22:59:40 0.027237 -1.275295 100.28370 8.26
2019-09-24 22:59:45 0.027237 -1.304609 100.28370 8.26
2019-09-24 22:59:50 0.027237 -1.289325 100.32010 8.26
2019-09-24 22:59:55 0.027237 -1.290819 100.21680 8.26
2019-09-24 23:00:00 0.027237 -1.268734 100.16140 8.26
CIA_CAUDAL CIA_VOLUMEN_ACUMULADO CIA_TIEMPO ¥
DATE_TIME
2019-09-10 00:00:00 0.0 0.00 0.000
2019-09-10 00:00:05 0.0 0.00 0.000
2019-09-10 00:00:10 0.0 0.00 0.000
2019-09-10 00:00:15 0.0 0.00 0.000
2019-09-10 00:00:20 0.0 0.00 0.000
... ... ... ...
2019-09-24 22:59:40 0.0 272.25 374.050
```

```
2019-09-24 22:59:45 0.0 272.25 374.133

2019-09-24 22:59:50 0.0 272.25 374.217
2019-09-24 22:59:55 0.0 272.25 374.300
2019-09-24 23:00:00 0.0 272.25 374.383
WIRELINE_PROFUNDIDAD WIRELINE_TENSION_DIFERENCIAL ¥
DATE_TIME
2019-09-10 00:00:00 0.0 0.0
2019-09-10 00:00:05 0.0 0.0
2019-09-10 00:00:10 0.0 0.0
2019-09-10 00:00:15 0.0 0.0
2019-09-10 00:00:20 0.0 0.0
... ... ...
2019-09-24 22:59:40 0.0 0.0
2019-09-24 22:59:45 0.0 0.0
2019-09-24 22:59:50 0.0 0.0
2019-09-24 22:59:55 0.0 0.0
2019-09-24 23:00:00 0.0 0.0
WIRELINE_TENSION_TOTAL WIRELINE_TIEMPO ¥
DATE_TIME
2019-09-10 00:00:00 0.0 0.0
2019-09-10 00:00:05 0.0 0.0
2019-09-10 00:00:10 0.0 0.0
2019-09-10 00:00:15 0.0 0.0
2019-09-10 00:00:20 0.0 0.0
... ... ...
2019-09-24 22:59:40 0.0 0.0
2019-09-24 22:59:45 0.0 0.0
2019-09-24 22:59:50 0.0 0.0
2019-09-24 22:59:55 0.0 0.0
2019-09-24 23:00:00 0.0 0.0
WIRELINE_VELOCIDAD CIA_VOLUMEN_POR_ETAPAS
DATE_TIME
2019-09-10 00:00:00 0.0 0.00
2019-09-10 00:00:05 0.0 0.00
2019-09-10 00:00:10 0.0 0.00
2019-09-10 00:00:15 0.0 0.00
2019-09-10 00:00:20 0.0 0.00
... ... ...
2019-09-24 22:59:40 0.0 118.85
2019-09-24 22:59:45 0.0 118.85
2019-09-24 22:59:50 0.0 118.85
2019-09-24 22:59:55 0.0 118.85
2019-09-24 23:00:00 0.0 118.85

[258414 rows x 18 columns]
```

Dividir los datos en Train, Test y Validation

[30]:
```
X_model, X_blind, Y_model, Y_blind = train_test_split(X, Y, test_size=0.3,␣
↪random_state=8)
X_train, X_valid, Y_train, Y_valid = train_test_split(X_model, Y_model,␣
↪test_size=0.3, random_state=8)
X_model.shape, X_blind.shape, X_train.shape, X_valid.shape,
```

[30]:
```
((180889, 18), (77525, 18), (126622, 18), (54267, 18))
```

[31]:
```
(len(X_model)/len(X), len(X_blind)/len(X))
```

[31]:
```
(0.6999969041924973, 0.3000030958075027)
```

[32]:
```
(len(X_train)/len(X_model), len(X_valid)/len(X_model))
```

[32]:
```
(0.6999983415243602, 0.30000165847563975)
```

[33]:
```
(len(X_train)/len(X), len(X_valid)/len(X))
```

[33]:
```
(0.4899966720069346, 0.2100002321855627)
```

Normalizar

[34]:
```
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_blind_std = sc.transform(X_blind)
X_valid_std = sc.transform(X_valid)
```

Reducir la dimensionalidad

[35]:
```
pca = PCA(n_components=6)
X_train_pca = pca.fit_transform(X_train)
```

```
X_blind_pca = pca.transform(X_blind)
X_valid_pca = pca.transform(X_valid)
```

[36]:
```
pd.Series(pca.explained_variance_ratio_).plot(kind='bar')
```

[36]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8d19430250>
```

[37]:
```
pd.Series(pca.explained_variance_ratio_).cumsum()
```

[37]:
```
0 0.854587
1 0.922670
2 0.988763
3 0.997680
4 0.999343
5 0.999621
dtype: float64
```

Anexo 5 – Aplicación de Clasificadores

- ***Ramdom Forest***

Definir el Pipeline

```
[38]:
sc = StandardScaler()
pca = PCA(n_components=5, random_state = 2)
model = RandomForestClassifier(random_state = 2)
pipeline = Pipeline([('sc', sc), ('pca', pca), ('model', model)])
pipeline
```

```
[38]:
Pipeline(steps=[('sc', StandardScaler()),
('pca', PCA(n_components=5, random_state=2)),
('model', RandomForestClassifier(random_state=2))])
```

Aplicar el modelo

```
[39]:
pipeline.fit(X_train, Y_train)
```

```
[39]:
Pipeline(steps=[('sc', StandardScaler()),
('pca', PCA(n_components=5, random_state=2)),
('model', RandomForestClassifier(random_state=2))])
```

Predecir resultados

```
[40]:
Y_train_predict = pipeline.predict(X_train)
Y_valid_predict = pipeline.predict(X_valid)
Y_blind_predict = pipeline.predict(X_blind)
Y_train_predict.shape, Y_valid_predict.shape, Y_blind_predict.shape
```

```
[40]:
((126622,), (54267,), (77525,))
```

Evaluar Performance

[41]:
```
print("Train:", metrics.explained_variance_score(Y_train, Y_train_predict))
print("Validation:", metrics.explained_variance_score(Y_valid,
Y_valid_predict))
print("Blind:", metrics.explained_variance_score(Y_blind, Y_blind_predict))
```
```
Train: 0.9916357408205044
Validation: 0.883226042761246
Blind: 0.8826357517136122
```

Matriz de Confusión

[42]:
```
cm = metrics.confusion_matrix(y_pred=Y_blind_predict, y_true=Y_blind); cm
```

[42]:
```
array([[ 3523,  91,  0,  7,  320],
[  94,  7026,  1,  124,  137],
[  0,  0,  1375,  0,  0],
[  7,  50,  0,  4473,  4],
[  250,  213,  0,  2,  59828]])
```

[43]:
```
cm_display = metrics.ConfusionMatrixDisplay(cm, display_labels=np.
↪unique(selected_well_data['MACROACTIVITY_CODE']))
cm_display
```

[43]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f8d0f08c700>
```

[44]:
```
fig, ax = plt.subplots(figsize=(10, 10))
cm_display.plot(ax = ax, values_format = '')
```

[44]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f8d0f08c700>
```

[45]:
```
print(metrics.classification_report(y_pred=Y_blind_predict, y_true=Y_blind,␣
↪target_names =np.unique(selected_well_data['MACROACTIVITY_CODE'])))
```
```
precision recall f1-score support
CALIBRANDO 0.91 0.89 0.90 3941
CASING RUNNING 0.95 0.95 0.95 7382
CEMENTANDO 1.00 1.00 1.00 1375
PERFILANDO 0.97 0.99 0.98 4534
PERFORANDO 0.99 0.99 0.99 60293
accuracy 0.98 77525
macro avg 0.96 0.96 0.96 77525
weighted avg 0.98 0.98 0.98 77525
```

Guardando el Modelo

[46]:
```
filename = 'MACROACTIVITY_CODE_Pipeline.pkl'
pickle.dump(pipeline, open(filename, 'wb'))
```

- ***Support Vector Machines (SVM)***

Definir el Pipeline

[47]:
```
model = svm.SVC(random_state = 2)
pipeline = Pipeline([('sc', sc), ('pca', pca), ('model', model)])
pipeline
```

[47]:
```
Pipeline(steps=[('sc', StandardScaler()),
('pca', PCA(n_components=5, random_state=2)),
('model', SVC(random_state=2))])
```

Aplicar el modelo

[48]:
```
pipeline.fit(X_train, Y_train)
```

[48]:

```
Pipeline(steps=[('sc', StandardScaler()),
('pca', PCA(n_components=5, random_state=2)),
('model', SVC(random_state=2))])
```

Predecir Resultados

```
[49]:
Y_train_predict = pipeline.predict(X_train)
Y_valid_predict = pipeline.predict(X_valid)
Y_blind_predict = pipeline.predict(X_blind)
Y_train_predict.shape, Y_valid_predict.shape, Y_blind_predict.shape

[49]:
((126622,), (54267,), (77525,))
```

Evaluar performance

```
[50]:
print("Train:", metrics.explained_variance_score(Y_train, Y_train_predict))
print("Validation:", metrics.explained_variance_score(Y_valid,
Y_valid_predict))
print("Blind:", metrics.explained_variance_score(Y_blind, Y_blind_predict))
Train: 0.2743373144703676
Validation: 0.2779786212605744
Blind: 0.274634910507477
```

- **Decision Tree**

Definir el Pipeline

```
[51]:
model = tree.DecisionTreeClassifier(random_state=2)
pipeline = Pipeline([('sc', sc), ('pca', pca), ('model', model)])
pipeline

[51]:
Pipeline(steps=[('sc', StandardScaler()),
('pca', PCA(n_components=5, random_state=2)),
('model', DecisionTreeClassifier(random_state=2))])
```

Aplicar el Modelo

[52]:
```
pipeline.fit(X_train, Y_train)
```

[52]:
```
Pipeline(steps=[('sc', StandardScaler()),
('pca', PCA(n_components=5, random_state=2)),
('model', DecisionTreeClassifier(random_state=2))])
```

Predecir Resultados

[53]:
```
Y_train_predict = pipeline.predict(X_train)
Y_valid_predict = pipeline.predict(X_valid)
Y_blind_predict = pipeline.predict(X_blind)
Y_train_predict.shape, Y_valid_predict.shape, Y_blind_predict.shape
```

[53]:
```
((126622,), (54267,), (77525,))
```

Evaluar Performance

[54]:
```
print("Train:", metrics.explained_variance_score(Y_train, Y_train_predict))
print("Validation:", metrics.explained_variance_score(Y_valid,
Y_valid_predict))
print("Blind:", metrics.explained_variance_score(Y_blind, Y_blind_predict))
Train: 0.9485369560213662
Validation: 0.7945955969343177
Blind: 0.7858039281053182
```

- **Neural Network**

Definir el Pipeline

[59]:
```
model = MLPClassifier(random_state=2, max_iter=1000)
pipeline = Pipeline([('sc', sc), ('pca', pca), ('model', model)])
pipeline
```

[59]:
```
Pipeline(steps=[('sc', StandardScaler()),
('pca', PCA(n_components=5, random_state=2)),
('model', MLPClassifier(max_iter=1000, random_state=2))])
```

Aplicar el Modelo

[60]:
```
pipeline.fit(X_train, Y_train)
```

[60]:
```
Pipeline(steps=[('sc', StandardScaler()),
('pca', PCA(n_components=5, random_state=2)),
('model', MLPClassifier(max_iter=1000, random_state=2))])
```

Predecir Resultados

[61]:
```
Y_train_predict = pipeline.predict(X_train)
Y_valid_predict = pipeline.predict(X_valid)
Y_blind_predict = pipeline.predict(X_blind)
Y_train_predict.shape, Y_valid_predict.shape, Y_blind_predict.shape
```

[61]:
```
((126622,), (54267,), (77525,))
```

Evaluar Performance

[62]:
```
print("Train:", metrics.explained_variance_score(Y_train, Y_train_predict))
print("Validation:", metrics.explained_variance_score(Y_valid,
Y_valid_predict))
print("Blind:", metrics.explained_variance_score(Y_blind, Y_blind_predict))
Train: 0.8655940783474907
Validation: 0.8568998532624599
Blind: 0.8554222537517451
```