ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Ciencias de la Tierra

Diseño de aplicación web para la predicción de producción de petróleo por medio de modelos de analítica predictiva aplicados al campo Volve, Noruega.

PROYECTO INTEGRADOR

Previo la obtención del Título de:

Ingeniero de Petróleos

Presentado por:

Luis Antonio Alchundia Laborde

Anthony Darío Hernández Tamayo

GUAYAQUIL - ECUADOR

Año: 2021

DEDICATORIA

El presente proyecto lo dedico a mi esposa, hijo y abuela por la lealtad, el amor y dedicación que me han brindado durante los mejores y los más difíciles años de mi vida.

Luis Antonio Alchundia Laborde

El esfuerzo y trabajo lo dedico para Dios, mis padres, hermanos y demás familia que me han sabido apoyar e impulsar durante estos años de estudio, y así también para mi abuela que no me acompaña ya hoy en día, pero que era alegría para ella saber que faltaba cada vez menos para alcanzar esta meta.

Anthony Darío Hernández Tamayo

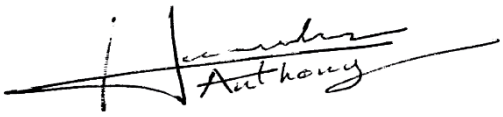
AGRADECIMIENTOS

Nuestro más sincero agradecimiento a la empresa EQUINOR, por la apertura de la base de datos del campo Volve para el desarrollo de nuestro proyecto integrador, a los ingenieros Jorge Lliguizaca, Freddy Carrión, Danilo Arcentales y Fernando Sagnay por haber dedicado su tiempo en compartirnos conocimiento y en habernos guiado correctamente durante el desarrollo de este trabajo. Por otro lado, queremos agradecer al ingeniero Álvaro García y a la ingeniera María Cecibel Castillo por haber afrontado el difícil reto de que sus estudiantes puedan obtener y culminar sus prácticas preprofesionales durante la crisis sanitaria causada por la COVID-19.

DECLARACIÓN EXPRESA

“Los derechos de titularidad y explotación, nos corresponde conforme al reglamento de propiedad intelectual de la institución; *Luis Antonio Alchundia Laborde, y Anthony Darío Hernández Tamayo* damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Luis Antonio
Alchundia Laborde



Anthony Darío
Hernández Tamayo

EVALUADORES

Firmado electrónicamente por:
DANILO ANDRES ARCENTALES BASTIDAS

MSc. Danilo Arcentales

PROFESOR TUTOR

Firmado electrónicamente por:
JORGE RODRIGO LLIGUIZACA DAVILA

MSc. Jorge Lliguizaca

PROFESOR TUTOR

Firmado electrónicamente por:
FERNANDO JAVIER SAGNAY SARES

MSc. Fernando Sagnay

PROFESOR DE LA MATERIA

RESUMEN

El siguiente proyecto está basado en el diseño de un aplicativo web que permita predecir producción de petróleo, teniendo como base los datos del campo Volve localizado en Noruega. Este conjunto de datos es parte fundamental para la aplicación de la ciencia de datos y del Machine Learning, ya que se relacionan directamente con diversos lenguajes de programación, y con múltiples algoritmos de analítica predictiva.

El proyecto inició con un análisis exploratorio de datos o EDA por sus siglas en inglés, donde el conjunto de datos original fue manipulado con el objetivo de obtener sus principales características. Posteriormente, el conjunto de datos original fue expuesto a una reducción en su dimensionalidad con ayuda de criterios y procesos como el de correlación entre variables, eliminación de datos categóricos, innecesarios o incongruentes para su normalización previo al entrenamiento de los modelos predictivos. Mediante una exhaustiva revisión bibliográfica, regresión lineal, regresión polinomial, ARIMA y Random Forest fueron los cuatro algoritmos seleccionados que fundamentarían las bases de nuestros modelos predictivos.

De los algoritmos mencionados y utilizados para predecir producción, obtuvimos como resultado que el modelo basado en regresión polinomial y Random Forest obtuvieron una precisión general de 0.92 y 0.96 respectivamente. Aunque el mejor desempeño dentro del aplicativo web llamado “Saviour” definitivamente lo tiene Random Forest, ya que, al no ser tan susceptible a valores atípicos dentro de un marco de datos, arroja predicciones mucho más confiables.

Saviour es un aplicativo web gratuito, cuya única función es predecir producción de petróleo y aunque no presente mayores ventajas en comparación con los modelos físicos tradicionales tiene dos importantes virtudes por encima del resto, el ahorro de tiempo y dinero para las compañías. Demostrando de esta manera que el tener conocimiento de ciencia de datos y Machine Learning es muy importante para resolver problemas de cualquier tipo dentro de la industria de petróleo y gas.

Palabras clave: Campo Volve, Ciencia de datos, Machine Learning, Regresión Polinomial, Random Forest, Saviour.

ABSTRACT

The following project is based on the design of a web application that allows predicting oil production, based on data from the Volve field located in Norway. This data set is a fundamental part for the application of data science and Machine Learning, since they are directly related to various programming languages, and with multiple predictive analytics algorithms.

The project began with an exploratory data analysis or EDA, where the original data set was manipulated to obtain its main characteristics. Subsequently, the original data set was exposed to a reduction in its dimensionality with the help of criteria and processes such as correlation between variables, elimination of categorical, unnecessary, or incongruous data for normalization prior to the training of predictive models. Through an exhaustive bibliographic review, linear regression, polynomial regression, ARIMA and Random Forest were the four selected algorithms that would make the bases of our predictive models.

From the algorithms mentioned and used to predict production, we obtained as a result that the model based on polynomial regression and Random Forest obtained a general precision of 0.92 and 0.96 respectively. Although the best performance within the web application called "Saviour" is definitely Random Forest, since, as it is not so susceptible to outliers within a data frame, it produces much more reliable predictions.

Saviour is a free web application, whose only function is to predict oil production and although it does not present major advantages compared to traditional physical models, it has two important virtues above the rest, saving time and money for companies. Proving in this way that having knowledge of data science and Machine Learning is very important to solve problems of any kind within the oil and gas industry.

Keywords: Volve Field, Data Science, Machine Learning, Polynomial Regression, Random Forest, Saviour.

ÍNDICE GENERAL

ABREVIATURAS

ARIMA Auto Regressive Integrated Moving Average ó Media Móvil Integrada Auto Regresiva.

BES Bomba Electro Sumergible.

CART Classification and Regression Trees ó Árboles de Clasificación y Regresión.

CSV Comma Separated Values ó Valores Separados por Coma.

EDA Exploratory Data Analysis ó Análisis Exploratorio de los Datos.

LWD Logging While Drilling

MVC Modelo-Vista-Controlador

MWD Meassure While Drilling

NPD Norwegian Petroleum Direction ó Dirección Noruega del Petróleo.

NPV Net Present Values ó Valor Presente Neto.

PVT Pressure-Volume-Temperature ó Presión-Volúmen-Temperatura

RF Random Forest ó Bosque Aleatorio

SQL Strutured Query Language ó Lenguaje de Consultas Estructurado.

SIMBOLOGÍA

bar bares

˚C grados Celsius

m metros

" metros cúbicos

$ r cuadrado (coeficiente de determinación)

Sm" metro cúbico estándar

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

1. INTRODUCCIÓN

Descripción del problema

La cantidad de datos generados, procesados y almacenados dentro de la industria petrolera, independientemente del área de aplicación, suelen generar bases de datos muy extensas. El campo Volve, localizado en el mar de Noruega no deja de ser una excepción, ya que, tan solo con 8 años de operación la cantidad de información que se generó es también extensa (*Volve - Equinor.Com*).

Estos datos que se manejan dentro de la industria del petróleo y gas, los cuales pueden ser estructurados en base a parámetros como, por ejemplo, temperatura y presión. Aunque, también existen datos no estructurados como grabaciones en vídeo, imágenes, etc. En los últimos años, se ha evidenciado el uso prominente del Big Data a razón de que las cantidades de datos generados y registrados van incrementando significativamente dentro de la industria petrolera y gasífera, tanto actividades de upstream como de downstream. (Mohammadpoor & Torabi, 2020)

Las mejorías que vienen teniendo sensores y dispositivos como por ejemplo los de adquisición sísmica, geófonos, herramientas LWD y MWD y muchos otros, incrementan el volumen de datos a ser procesados y analizados para lograr obtener los grandes beneficios diferenciadores que se puede lograr por medio del Big Data.(J. Spath, 2015)

Este pésimo aprovechamiento de información ha limitado a este sector industrial en poder obtener resultados positivos considerables, a diferencia de otros sectores industriales que si evidencian importantes cambios. Aun así, la industria petrolera ha sabido caracterizarse por ser muy innovadora y estar siempre a la vanguardia tecnológica del mundo. Es por lo cual, estas nuevas aplicaciones o ciencias ya se han visto implementadas en las diferentes áreas de la industria como exploración, perforación, reservorios, producción, refinación, transporte e incluso salud y seguridad por medio de varios proyectos ejecutados que generan un avance positivo. (Mohammadpoor & Torabi, 2020). Haciendo énfasis en el área de producción se

cuenta con excelentes resultados de aplicaciones que son capaces de distinguir zonas de creación por debajo de la media o de incrementar las tasas de recuperación de petróleo analizando varias fuentes de información como la sísmica, de pozo y manufacturación. (Seemann D., Williamson M., 2013)

En la industria hidrocarburífera es evidenciable el interés de aplicabilidad de esta ciencia, el de saber manipular estos datos para predecir quizá potenciales yacimientos, tasas de producción, volumen de reservas remanentes, etc. Interesa conocer también el grado de exactitud y confiabilidad de estos resultados, para la posterior toma de decisiones que eviten principalmente perdidas de dinero y la mala utilización del tiempo.

Justificación del problema

El almacenamiento de datos no analizados, provenientes de las distintas áreas de operación de una empresa, generan gastos tan solo por almacenamiento y generará aún más gastos cuando se requiera utilizar dicha información y no estén disponibles a tiempo. Sin mencionar de igual forma, el tiempo que generalmente van a emplear las personas para la búsqueda de dicha información dentro de lo que se conoce como “lago de datos”.

En la industria petrolera ecuatoriana, se tienen inconvenientes similares que deberían ser ya resueltos por medio de herramientas adecuadas que permitan obtener grandes ventajas de esta información que se ha almacenado por medio de las diferentes actividades petroleras realizadas dentro de un campo.

La ciencia de datos es el camino adecuado para innovar la industria petrolera ecuatoriana, de hecho, es a lo que se está apuntando a nivel global actualmente, su inclusión. Por tal motivo, la aplicación de esta herramienta conllevará a un procedimiento ordenado de datos dentro del conjunto de archivos que se obtengan hasta la aplicación de procesos analíticos predictivos que nos brinde las mayores ventajas de esta ciencia.

Evidentemente, los datos a utilizarse pertenecen a un campo petrolero extranjero. Sin embargo, es suficiente para poder desarrollar la parte fundamental de nuestro aplicativo web, creado a base de un lenguaje de programación muy habitual y

gratuito como lo es Python, que conjunto a sus librerías nos brindarán las herramientas necesarias para la aplicación de esta ciencia.

El obtener estos análisis de predicción de producción sobre un campo petrolero, sin duda alguna es de mucha importancia para una preparación futura a niveles técnicos. Con el fin de optimizar ganancias y procesos, no cabe la menor duda de que este proyecto buscará seguir expandiendo el camino para direccionar a la industria hidrocarburífera ecuatoriana hacia la aplicabilidad de la ciencia de datos, teniendo como objetivo siempre el beneficio de nuestro país.

1.3 Objetivos

1.3.1 Objetivo General

Diseñar un aplicativo web mediante el uso de la herramienta de código abierto Python en conjunto con algoritmos de analítica predictiva para la estimación de producción de petróleo en pozos petroleros. Teniendo como base los datos del campo Volve, localizado en el mar de Noruega.

1.3.2 Objetivos Específicos

1. Crear base de datos normalizada.
2. Seleccionar los modelos predictivos adecuados basado en los niveles de precisión de sus resultados.
3. Diseñar el aplicativo web, acoplando el modelo o los modelos predictivos seleccionados a su estructura informática para su posterior ejecución.
4. Contrastar los modelos físicos tradicionales y “Saviour”, aplicativo web de nuestra autoridad fundamentado completamente bajo conceptos de Data Science y Machine Learning.

1.4 Marco teórico

1.4.1 Campo Volve

El campo Volve se encuentra ubicado en el bloque 15/9, a unos 200 kilómetros al oeste de Stavanger y 8 kilómetros al norte de la plataforma Sleipner A, sobre una profundidad de 90 metros con respecto al suelo marino.

El campo está compuesto por cinco pozos productores y dos pozos de inyección de agua, aunque originalmente su plan de desarrollo consistía en la perforación de tres pozos productores y tres inyectores. El método de levantamiento artificial de este campo está compuesto por una combinación entre Gas Lift y la utilización de bombas electro sumergibles. En la tabla 1.1 se detallan diversas características geológicas y petrofísicas del campo en forma general.

Tabla 1.1 Evaluación geológica y condiciones técnicas del reservorio. (*Volve - Equinor.Com, n.d.*)

Relación neta/grosor	93%
Permeabilidad	~ 1 Darcy
Porosidad	21%
Saturación de agua en la zona de petróleo	20%
Profundidad del contacto agua-petróleo	3120 m por debajo del nivel del mar
Presión de reservorio	340 bar a 3060 metros
Temperatura de reservorio	110° C
Relación gas/petróleo	111-157 Sm^3/Sm^3
Factor Volumétrico	1.33-1.45 m^3/Sm^3

1.4.2 Data Science o Ciencia de datos

La ciencia de datos es un campo multidisciplinario que está fundamentado principalmente por cuatro características específicas, el pensamiento científico, métodos estadísticos, ingeniería computacional e innovación tecnológica (*¿Qué Es La Ciencia de Datos?* | *Oracle México*). Es muy reconocida por manejar extensas bases de datos de forma eficiente, por tal motivo, dentro del desarrollo de este trabajo es considerado como el pilar fundamental por estar asociado directamente al uso de lenguajes de programación y de algoritmos de predicción. La figura 1.1

muestra la relación que tienen las diferentes ciencias para conformar lo que es conocido ciencia de datos.

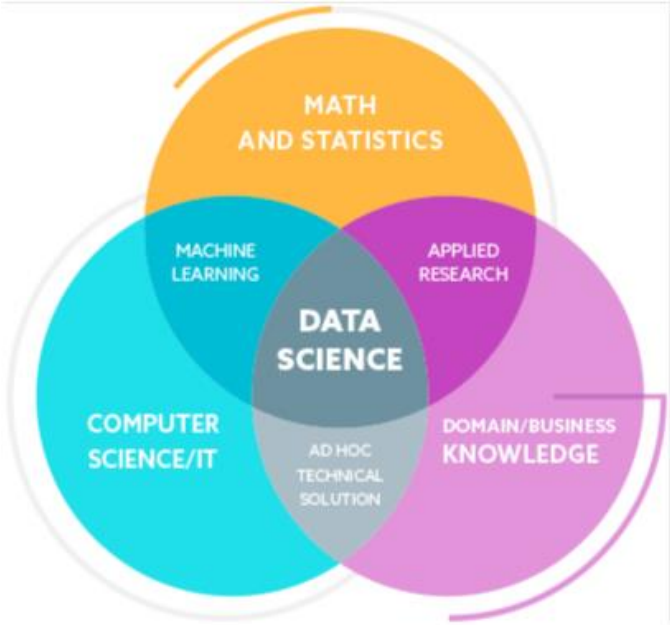


Figura 1.1 Ciencias y dominios que conforman la ciencia de datos (Peter et al., 2020)

1.4.3 Data Lake o Lago de datos

Data Lake o lagos de datos, es un repositorio de información que poseen las empresas para almacenar sus datos en bruto. De hecho, es considerado el corazón de muchos proyectos de ciencia de datos, ya que sin acceso a una variedad de datos no se podría obtener algún tipo de resultado (Antonio et al., 2019). Para entender de mejor manera la ciencia de datos se puede realizar una analogía de similitudes entre el Data Lake y los establecimientos de manufacturas comunes en la vida real, tal como lo muestra la tabla 1.2 a continuación.

Tabla 1.2 Descripción general de las similitudes entre los lagos de datos y las instalaciones de fabricación. (Antonio et al., 2019)

Componente del Data Lake	Equivalente en Manufacturas	Descripción de Similitud
Interfaz de programación de aplicaciones de ingestión	Puerta de recepción	Ubicación y procesos para permitir la entrada de nuevo material al medio ambiente

Datos de aseguramiento y control de calidad	Inspección de entrada	Asegúrese de que se cumplan los estándares de calidad antes de enviar los productos a los pasos posteriores
Tienda de objetos	Depósito o warehouse	Depósito a largo plazo de materia prima
Catálogo de datos	Planificación e inventario de registros de materiales	Recurso centralizado para rastrear el material disponible
Caja de ciencia de datos	Investigación y desarrollo	Permite el desarrollo iterativo sin afectar las actividades de producción actuales
Ingeniería de datos	Bienes semiterminados	Prepare material que admita múltiples casos de uso
Desarrollo de modelos de aprendizaje automático	Estaciones de trabajo	Los recursos requerían transformación material
Implementación del modelo	Envío	Distribuir el trabajo terminado a los usuarios finales

1.4.4 Fundamentos matemáticos y estadísticos del Machine Learning

El Machine Learning es un dominio de inteligencia artificial que descubre patrones de datos y realiza predicciones a través de algoritmos entrenados con datos históricos. El Machine Learning está sustentado por cuatro pilares importantes, los cuales son:

- Regresión
- Reducción de dimensionalidad
- Estimación de densidad
- Clasificación

Pero ¿Cuáles son los principios o fundamentos matemáticos y estadísticos que fundamentan a estos cuatro pilares del Machine Learning?

De hecho, son exactamente seis y están descritos en el orden que se muestra a continuación:

1.4.4.1 Álgebra Lineal

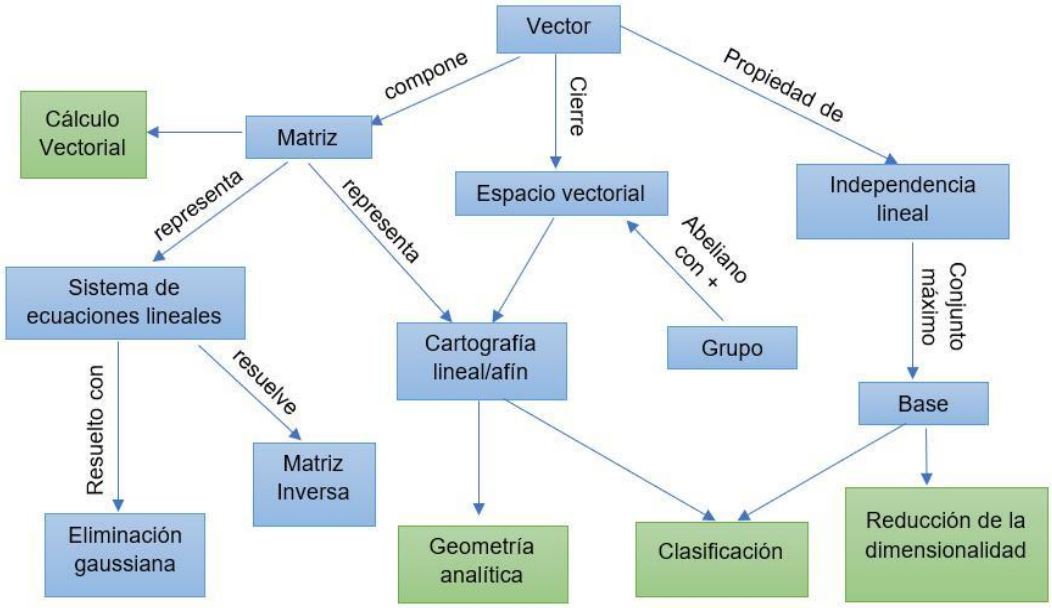


Figura 1.2 Mapa mental de conceptos asociados al álgebra lineal (Peter et al., 2020)

Se lo conoce como el estudio de los vectores y las reglas que los rigen para su manipulación (figura 1.2). Pero de forma general se definen como objetos espaciales que se pueden sumar entre sí y multiplicar con un escalar (número) para producir un vector de tipo parecido (Peter et al., 2020).

1.4.4.2 Geometría Analítica

Dentro de la geometría analítica se busca entender la interpretación geométrica de dichos vectores, conocer sus longitudes, distancias y ángulos entre ellos. Para realizarlo, es necesario introducir un valor conocido como el producto interno, valor que conjunto a sus normas y métricas capturan las nociones intuitivas de similitud y distancia. Dichos conceptos son utilizados y a su vez mostrados en la figura 1.3 para fundamentar el pilar ¨Clasificación¨ del Machine Learning (Peter et al., 2020).

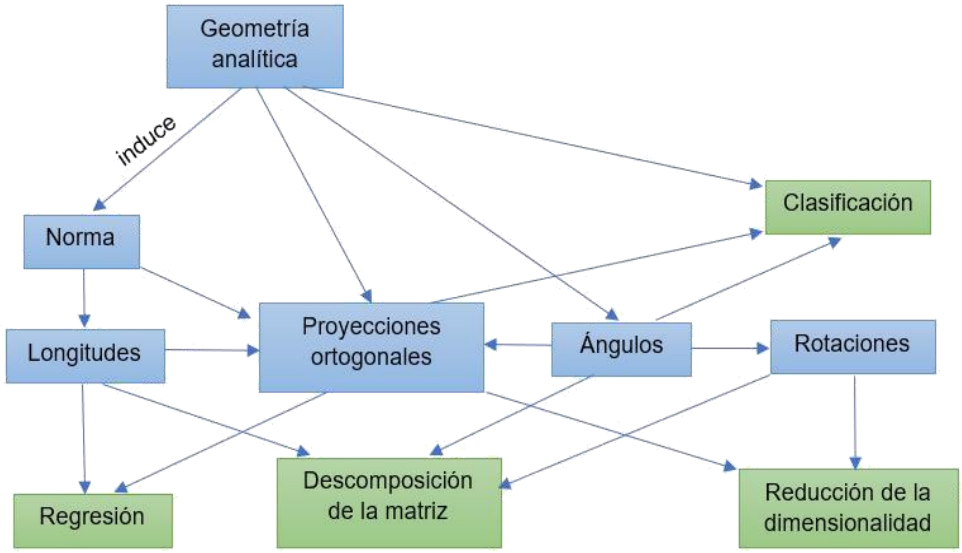


Figura 1.3 Mapa mental de conceptos asociados a la geometría analítica (Peter et al., 2020)

1.4.4.3 Descomposición Matricial

La descomposición matricial o también llamada factorización matricial, se utiliza para describir una matriz por medio de una representación diferente utilizando factores de matrices interpretables.

Las descomposiciones mostradas en la figura 1.4 son de gran ayuda, ya que las matrices al estar constituidas por extensos datos numéricos suelen ser muy complejos de analizar (Peter et al., 2020).

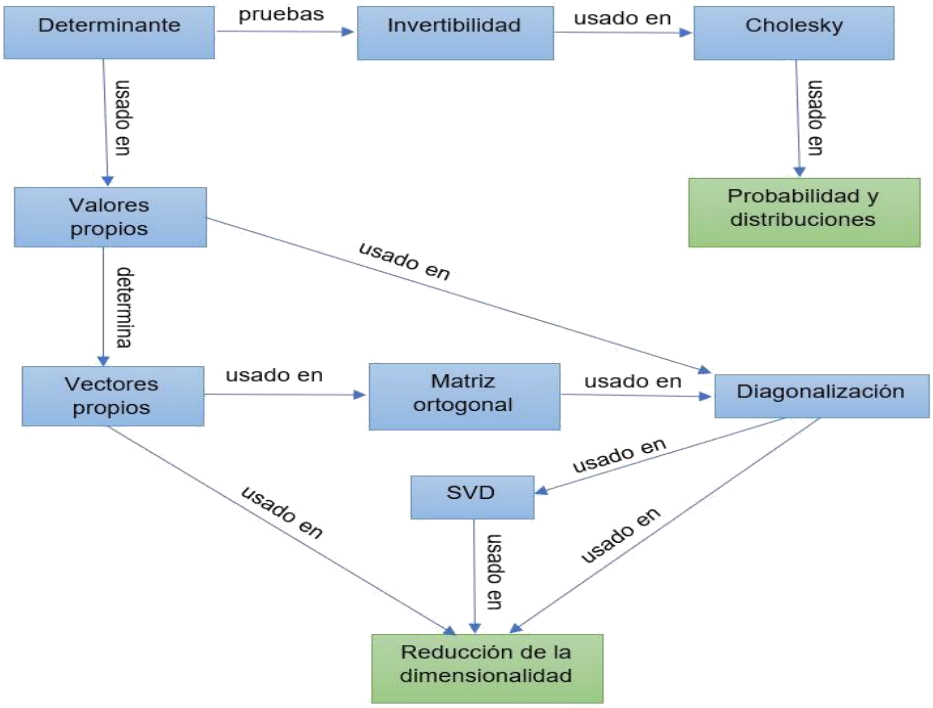


Figura 1.4 Mapa mental de conceptos asociados a la descomposición matricial (Peter et al., 2020)

1.4.4.4 Cálculo Vectorial

Muchos algoritmos en Machine Learning optimizan funciones con respecto a un grupo de parámetros que controlan el buen desenvolvimiento del modelo (Peter et al., 2020). El hablar de buenos parámetros puede ser parafraseado como problemas de optimización, eso incluye:

- Regresión lineal: Problemas como ajuste de curvas y de optimización de parámetros de peso lineal para maximizar probabilidades.
- Redes neuronales: Reducción de dimensionalidad y compresión de datos.
- Modelos Gaussianos mixtos: Modelamientos para la distribución de datos.

La figura 1.5 nos muestra una clasificación general de ciencias que conforman el cálculo vectorial.

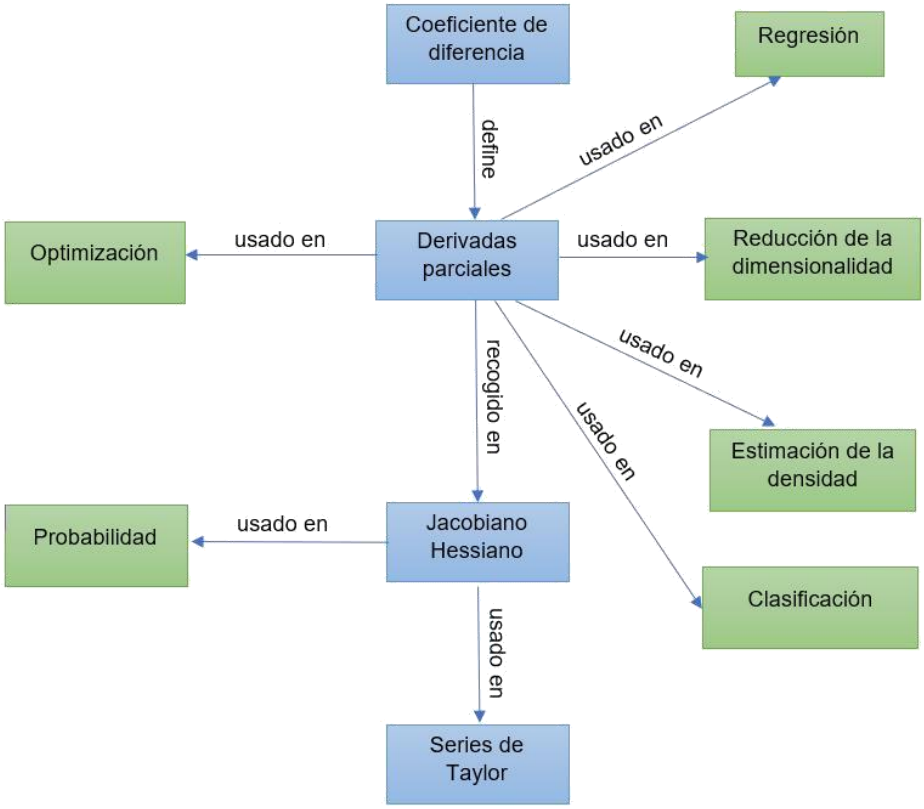


Figura 1.5 Mapa mental de conceptos asociados al cálculo vectorial (Peter et al., 2020)

1.4.4.5 Probabilidad y Distribuciones

La probabilidad es el estudio de las oportunidades de que un evento ocurra dentro de un experimento determinado. Dentro del Machine Learning, tal como lo detalla la figura 1.6, podemos asociar este concepto para conocer el grado de incertidumbre de los datos, el modelo y de las predicciones producidas por ese modelo. Por otro lado, las distribuciones de probabilidad se refieren más a la construcción de modelos probabilísticos, modelos gráficos y modelos de selección. Ambas asociadas directamente a la cuantificación de incertidumbres, relacionadas fuertemente a la idea conocida como variable aleatoria (Peter et al., 2020).

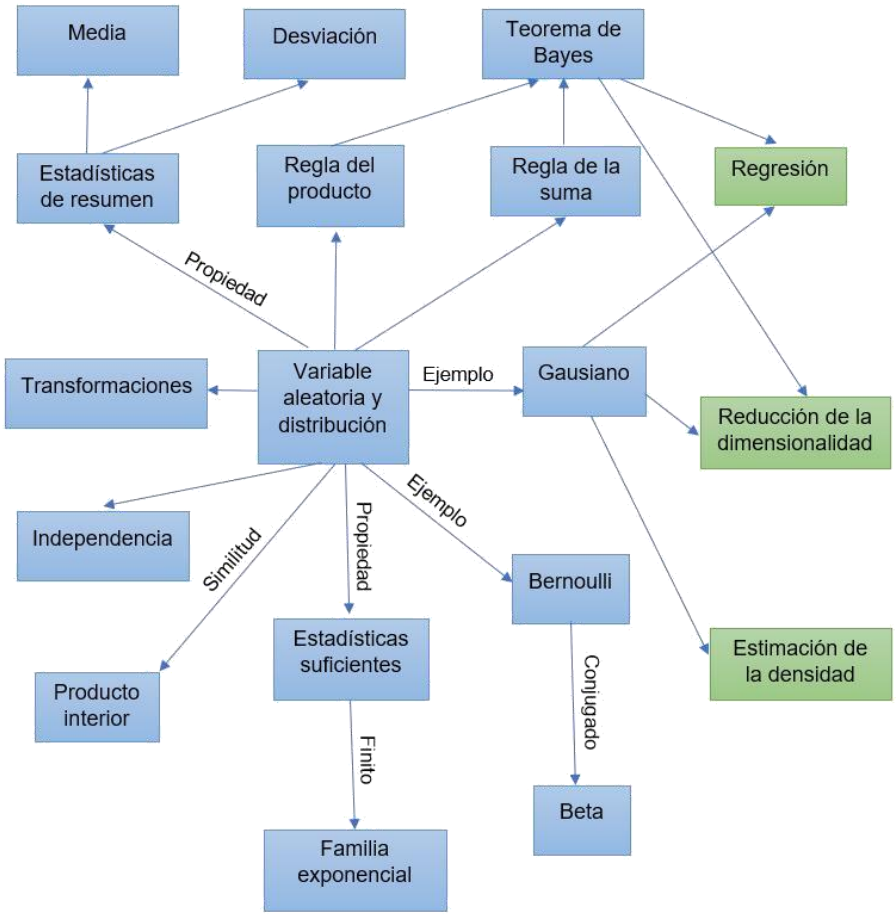


Figura1.6 Mapa mental de conceptos asociados a las probabilidades y distribuciones

(Peter **et al., 2020)**

1.4.4.6 Optimización continua

Como los algoritmos de Machine Learning son anexados dentro de computadoras, las fórmulas matemáticas que lo gobiernan son expresadas como métodos de expresión numérica. Básicamente, la optimización continua se refiere a todos estos métodos numéricos utilizados para entrenar modelos de Machine Learning (Peter et al., 2020). A continuación, la figura 1.7 nos brinda un mejor entendimiento de lo descrito, pero de manera gráfica.

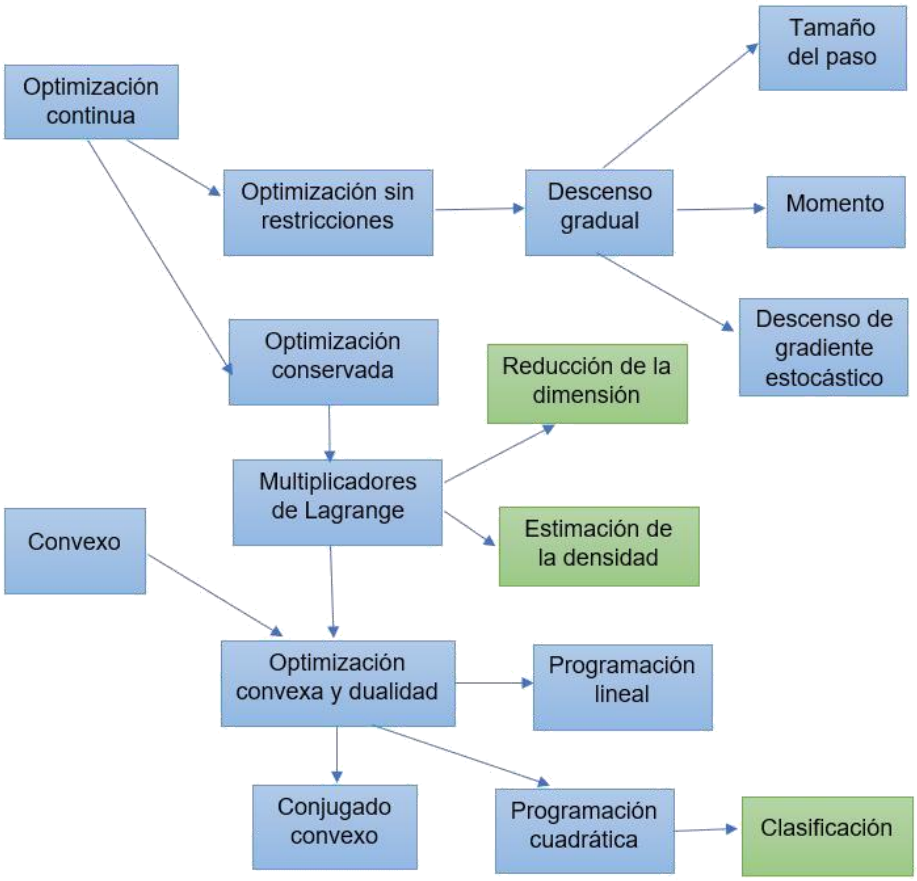


Figura 1.7 Mapa mental de conceptos asociados a la optimización continua (Peter et al., 2020)

1.4.5 Machine Learning

El campo de Machine Learning o aprendizaje automático, se lo define en gran parte dentro de las aplicaciones de la ciencia de datos para la obtención de predicciones. Se lo define como un dominio de inteligencia artificial que descubre patrones de datos y realiza predicciones a través de algoritmos entrenados con datos históricos, también llamados datos de entrenamiento (Asfoor, 2020).

Esta capacidad que ofrece el Machine Learning encuentra mejoras aprendiendo de una mayor cantidad de datos en el tiempo. Este campo requiere de ciertas herramientas conocidas como los algoritmos predictivos del Machine Learning, los cuales son herramientas muy prometedoras de la inteligencia artificial para el objetivo de realizar pronósticos y entender comportamientos lineales y no lineales (Sami & Ibrahim, 2021).

El probar y seleccionar el algoritmo más adecuado para el caso en que estamos trabajando nos permitirá definir el modelo adecuado y su configuración. A continuación, en la figura 1.8, se puede visualizar un flujograma básico para la obtención de un modelo predictivo.

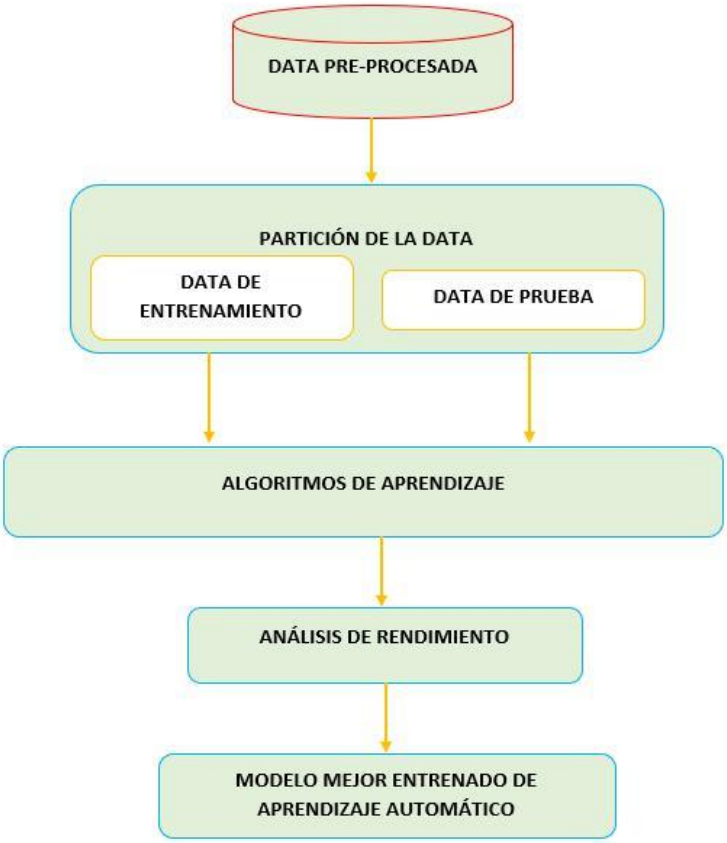


Figura 1.8 Proceso básico para desarrollar un modelo de Machine Learning para predicción. (Ali, 2021)

Los datos de producción que se han obtenido del campo Volve, son los datos de entrenamiento y prueba no normalizados para nuestro proyecto, que servirán tanto para el desarrollo de nuestro modelo predictivo, como para la comprobatoria de funcionamiento del aplicativo web.

Para este desarrollo se requieren herramientas que permitan modificar y trabajar con los datos y los algoritmos. Por tal motivo, hemos seleccionado el lenguaje de programación Python, a través de su interfaz Jupyter Lab, cuyo funcionamiento de libre acceso permite la creación de aplicaciones web, el análisis de datos, y la automatización de operaciones. Python es el lenguaje elegido para el desarrollo de este trabajo por su gran aplicabilidad en el campo de la ingeniería, y porque

conjunto a su variedad de librerías como NumPy, Matplotlib, Seaborn, Pandas y Scikit-learn se va a desarrollar el modelo o los modelos adecuados para el cálculo de predicciones de producción para distintos pozos petroleros (*¿Para Qué Sirve Python? Razones Para Utilizarlo | ESIC*).

1.4.5.1 Algoritmo Regresión Lineal

La regresión lineal es el modelo de regresión más simple ya que involucra solo una variable independiente y asume una función lineal (línea recta) entre x e y. El objetivo de la regresión lineal es ajustar una línea (y = w. X + b, donde b es el sesgo / intersección y w es la pendiente / peso) a un conjunto de puntos de datos x para predecir valores futuros de y para valores de x, conocida como línea de mejor ajuste.

Regresión lineal aplica el criterio de error de mínimos cuadrados para encontrar la línea de mejor ajuste. La línea de mejor ajuste es aquella donde la suma de los errores es cero y la suma de los cuadrados de los errores es mínima (Kumar & Sahu, 2021).

1.4.5.2 Algoritmo Regresión Polinomial

La integración del algoritmo regresión polinomial, al igual que regresión lineal, es fácil de usar en la predicción de producción de petróleo. La principal diferencia entre la regresión lineal y polinomial es que la regresión lineal requiere que las variables dependientes e independientes estén relacionadas linealmente, mientras que esto puede ajustarse mejor a la línea si incluimos un grado superior al término de la variable independiente en la ecuación.

Si agregamos grados más altos, como cuadrático, entonces convierte la línea en una curva que se ajusta mejor a los datos. Generalmente, se utiliza cuando los puntos del conjunto de datos están dispersos y el modelo lineal no puede describir el resultado con claridad. Siempre debemos vigilar el sobreajuste y el desajuste al considerar estos grados en la ecuación (*Polynomial Regression | Uses and Features of Polynomial Regression*).

1.4.5.3 Algoritmo ARIMA

El modelo propuesto por Box y Jenkins en la década de los setenta es sin duda alguna uno de los modelos estadísticos lineales más efectivos y por ende de los

más empleados en el objetivo de realizar predicciones. ARIMA es un modelo que trabaja con series de tiempo estacionarias para realizar predicciones, se constituye en base a otro modelo y características estadísticas, que representan cada uno un parámetro importante dentro del modelo final.

Los parámetros que requiere el modelo son p, D y q; el hiperparámetro p parte del modelo de auto regresión (AR), q se lo obtiene a partir del cálculo de la media móvil mientras que D es propio del grado de diferenciación de los datos. Además, algo importante que se indica, es que el modelo requiere de una serie estacionaria, lo que significa que ciertos parámetros estadísticos como la media, la varianza y covarianza se mantendrán en el tiempo, y que los valores que requieren adoptar los hiperparámetros deben ser los óptimos para lograr mejores valores de precisión, lo cual se consigue con un correcto tuneo de los hiperparámetros que nos indicará los valores que deben tener p, D y q a raíz de los datos de entrenamiento (Fan et al., 2021).

1.4.5.4 Algoritmo Random Forest

Este algoritmo de bosque aleatorio se construye en base al modelo de árbol de decisión es decir el modelo Random Forest (RF) es un método apoyado en la decisión mayoritaria y más precisa que pueden brindar el conjunto de árboles individuales, por tal motivo este algoritmo goza de muy buenos resultados tanto en desempeños como clasificador o regresor. Cada uno de los árboles de decisión binarios que conforman el bosque o también conocidos como árboles de clasificación y regresión (CART por sus siglas en inglés) se entrenan en base a un n número de subconjuntos aleatorios de datos o bootstrapped dataset del conjunto de datos de entrenamiento, y también se toma en cuenta un subconjunto aleatorio de las covariables. El resultado de predicción proviene de la media de los resultados de cada árbol del bosque aleatorio (Liao et al., 2020).

Dentro de Python, el modelo se lo puede emplear desde la librearía de scikit-learn y este cuenta con hiperparámetros que pueden permitirnos tener un modelo de mayor poder predictivo o mayor velocidad de ejecución. Estos son “n_estimators” que requiere de un valor numérico que indicará la cantidad de árboles de decisión con el que contará nuestro bosque y entre mayor sean este número, mayor será la precisión de los resultados de predicción del modelo, pero también volverá más

lenta su ejecución; “max_features” que requiere de la cantidad máxima de características que el bosque aleatorio emplee para separar un nodo, y “min_sample_leaf” determina la cantidad mínima de hojas requeridas para separar un nodo interno. Existen otros parámetros importantes que en cambio nos darán mayor velocidad de cómputo para predicciones incluso en tiempo real, estas son “n_jobs” que pregunta la cantidad de procesadores a usar para el desarrollo del modelo, y el hiperparámetro “random_state” que logra convertir en un tipo de salida replicable del modelo, según sea las mimas circunstancias como el valor del hiper parámetro y los datos de entrenamiento (*Random Forest Algorithms: A Complete Guide* | *Built In*, n.d.).

CAPÍTULO 2

2. METODOLOGÍA

2.1 Estructura de la investigación

La metodología que hemos propuesto consta de cuatro fases estrictamente conectadas. Las hemos definido como estrictamente conectadas porque para iniciar una fase siguiente se debe concluir en su totalidad la fase predecesora. La primera fase se basó en la estructura de una base de datos simple no normalizada, y también de una base de datos normalizada que contenga la información más relevante y necesaria del proyecto. La normalización de los datos se inició mediante un análisis de correlación de variables dentro de lo que se conoce como análisis exploratorio de datos, para finalmente culminar con la limpieza de los datos en bruto y la organización de datos normalizados. La segunda fase involucró directamente la manipulación de la base de datos normalizada para el entrenamiento y prueba de modelos predictivos, preseleccionados mediante una exhaustiva revisión bibliográfica.

La tercera fase de nuestra metodología se basó en el diseño del aplicativo web haciendo uso de lenguajes como Python y Javascript; esta fase se la llamó ¨fase de acoplamiento del modelo¨, porque la finalidad en sí es de corroborar el funcionamiento del modelo predictivo dentro del aplicativo web. La fase cuatro o fase final, es una fase de contraste en sí, y es que prácticamente lo que se buscó fue demostrar que al tener conocimientos de Data Science y Machine Learning podemos crear modelos predictivos de producción de petróleo semejantes o mejores que los modelos físicos tradicionales, y de forma totalmente gratuita.

Se recalca la gratuidad en la elaboración de nuestro proyecto, ya que cada fase de este fue realizada mediante el uso de herramientas de libre acceso. Como, por ejemplo, la interfaz de lenguaje Python conocida como Jupyter Lab, la plataforma para el albergue de base de datos conocida como SQL Server Management Studio 18, servidores y clientes web conocidos como Django y Javascript respectivamente.

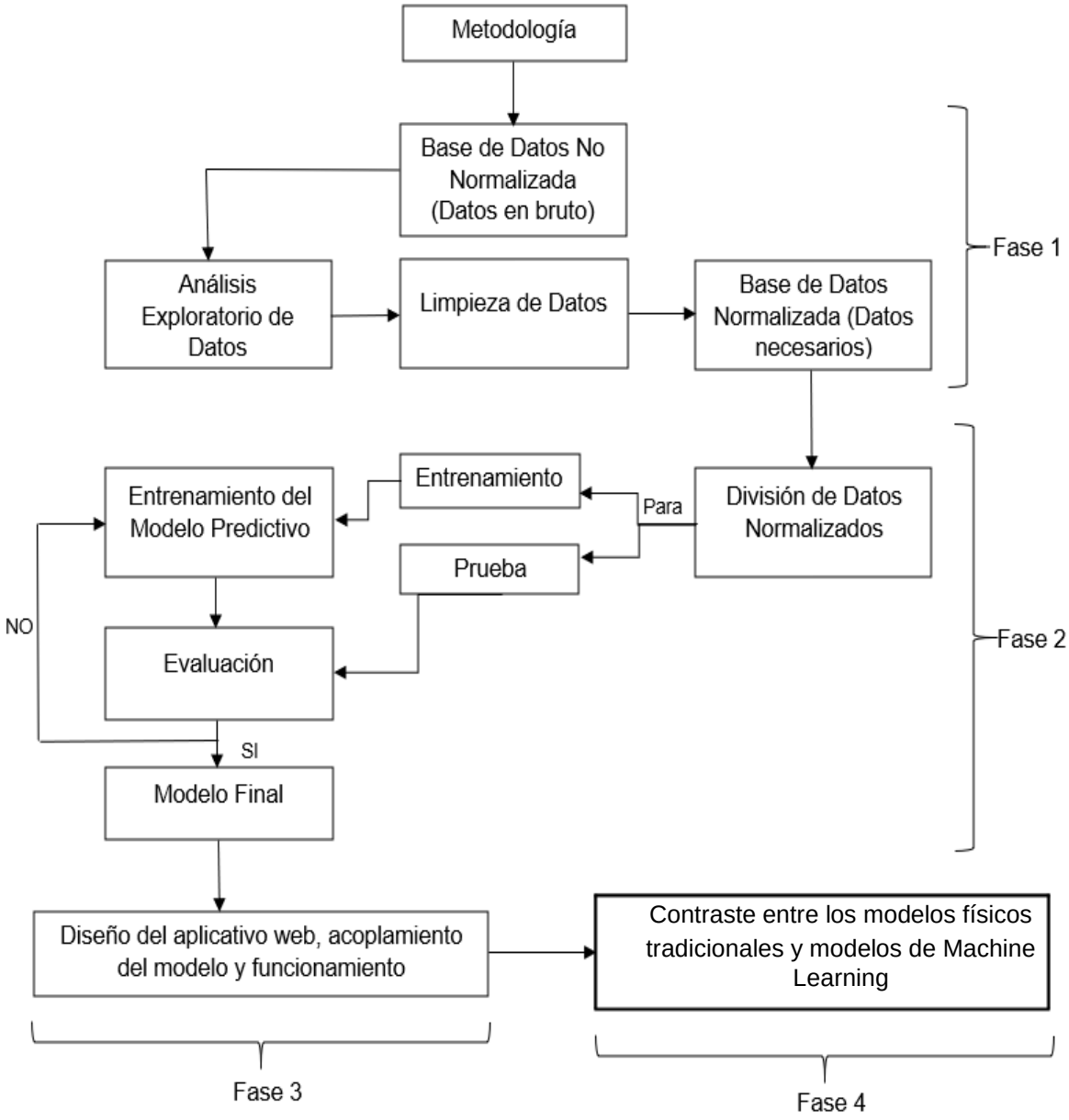


Figura 2.1 Diagrama de la metodología a seguir (fuente: Autores)

2.2 Procedimientos

2.2.1 Estructuración de las Bases de Datos

2.2.1.1 Base de datos no normalizada

Se le llama base de datos no normalizada a un conjunto de datos en bruto. En otras palabras, se refiere a un conjunto de datos que no hayan sido analizados y corregidos para un posterior procesamiento. Dentro de esta fase, conocida como

la fase uno, planteamos dos alternativas distintas para estructurar dicha base de datos, donde ambas alternativas presentadas a continuación fueron propuestas para cumplir un objetivo en específico.

Alternativa 1

Esta alternativa consistía en albergar el conjunto de datos dentro de una carpeta común en el disco duro del computador; con la finalidad de conectar dicha información de forma más directa con la interfaz Jupyter Lab para una manipulación de datos más práctica. Tal como se muestra a continuación en la figura 2.2, existe una línea de código sencilla que facilita la importación de documentos hacia la ventana de trabajo de dicha interfaz, haciendo uso principalmente de la librería llamada Pandas. Para, por ejemplo, realizar una un análisis exploratorio, limpieza y ordenamiento de estos datos, etc.

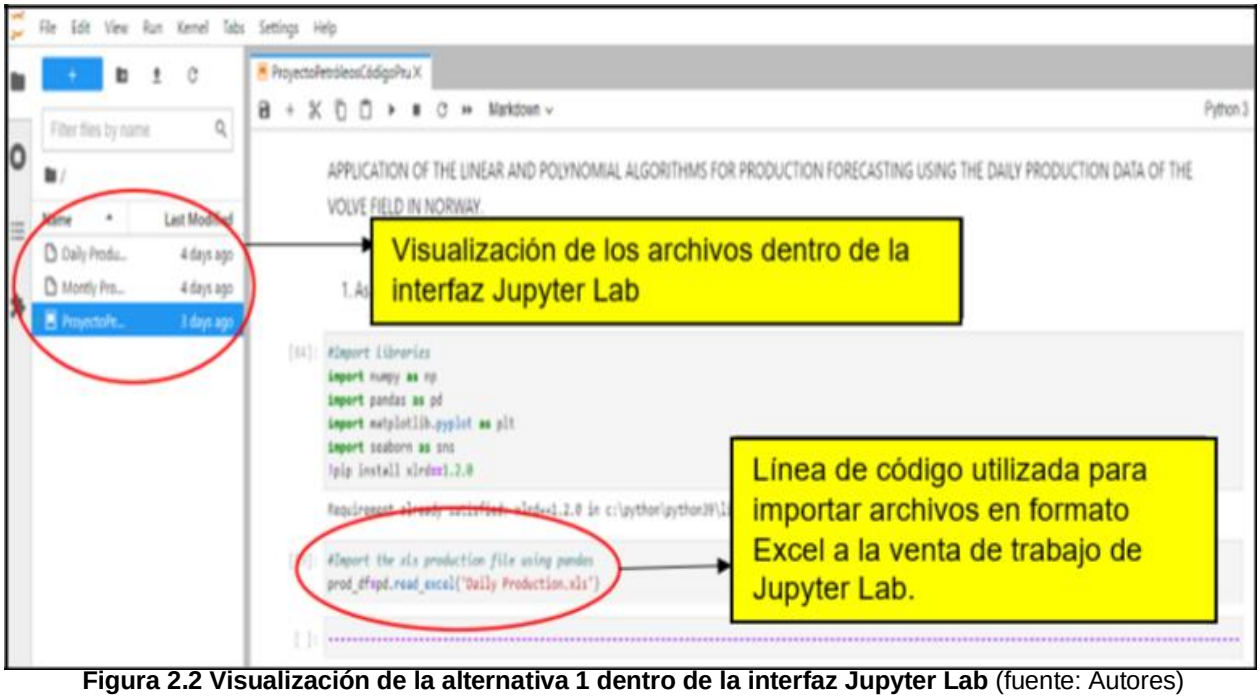


Figura 2.2 Visualización de la alternativa 1 dentro de la interfaz Jupyter Lab (fuente: Autores)

Alternativa 2

Esta alternativa a diferencia de la anterior involucra una nueva plataforma llamada SQL Management Studio 18 cuyo lenguaje informático es SQL. Dentro de esta plataforma lo que se buscó fue albergar el conjunto de datos no normalizado como una tabla cuyo nombre está definido como “Daily_Production_Data” dentro de una base de datos de nombre “Proyecto_Petróleos”. A continuación, en la figura 2.3 se muestra la plataforma SQL Management Studio 18 con lo descrito anteriormente.

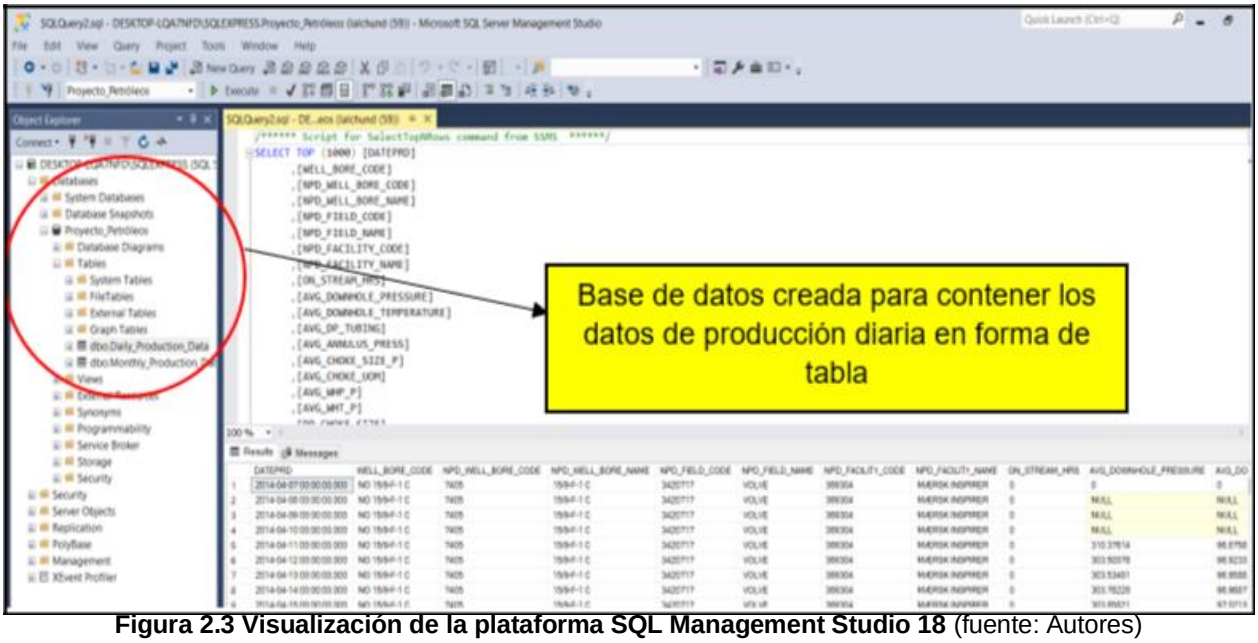


Figura 2.3 Visualización de la plataforma SQL Management Studio 18 (fuente: Autores)

Tal y como se puede observar en la imagen superior, esta plataforma también te permite visualizar los datos que contiene dicha tabla, lo que la convierte en una plataforma muy dinámica para utilizar en esta clase de proyectos. Pero, para poder conectar esta plataforma a la interfaz Jupyter Lab, se necesitan líneas de código específicas. Esas líneas de código se presentan a continuación en la figura 2.4.

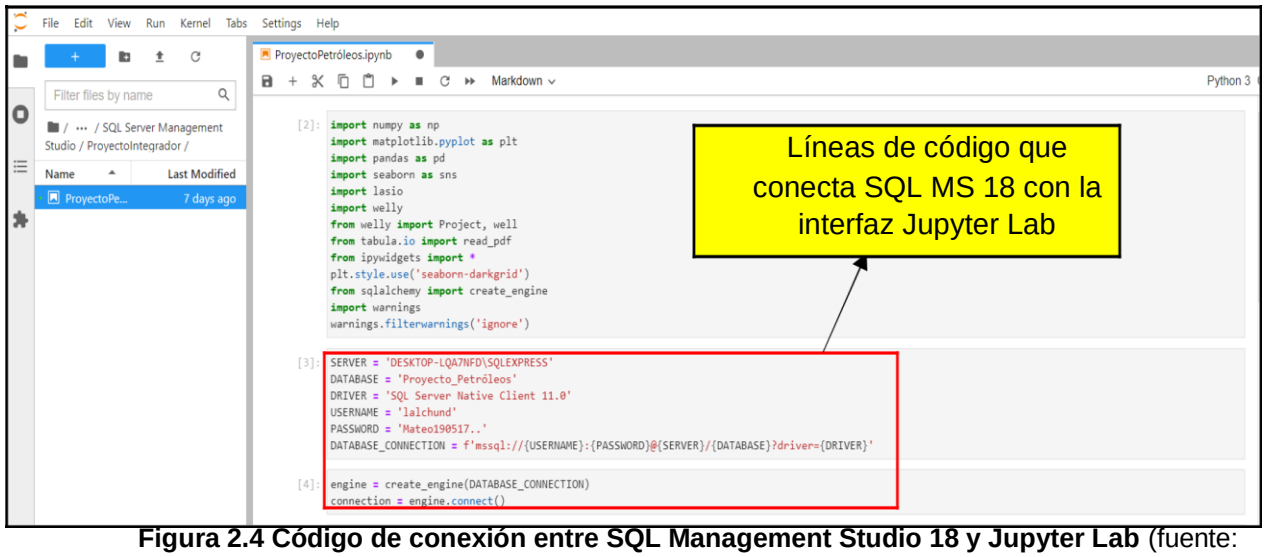


Figura 2.4 Código de conexión entre SQL Management Studio 18 y Jupyter Lab (fuente: Autores)

Por motivos de comprobación, se realizó la demostración de que la conexión entre SQL MS 18 y Jupyter Lab fue exitosa, mediante la lectura de los datos de producción, usando una línea de código que lee archivos en formato Query dentro de esta plataforma. Una vez más hicimos uso de la biblioteca Pandas que se encarga de convertir estos datos en marco de datos o dataframes para cuestiones de manipulación y visualización práctica de datos. En la figura 2.5 observamos lo mencionado anteriormente.

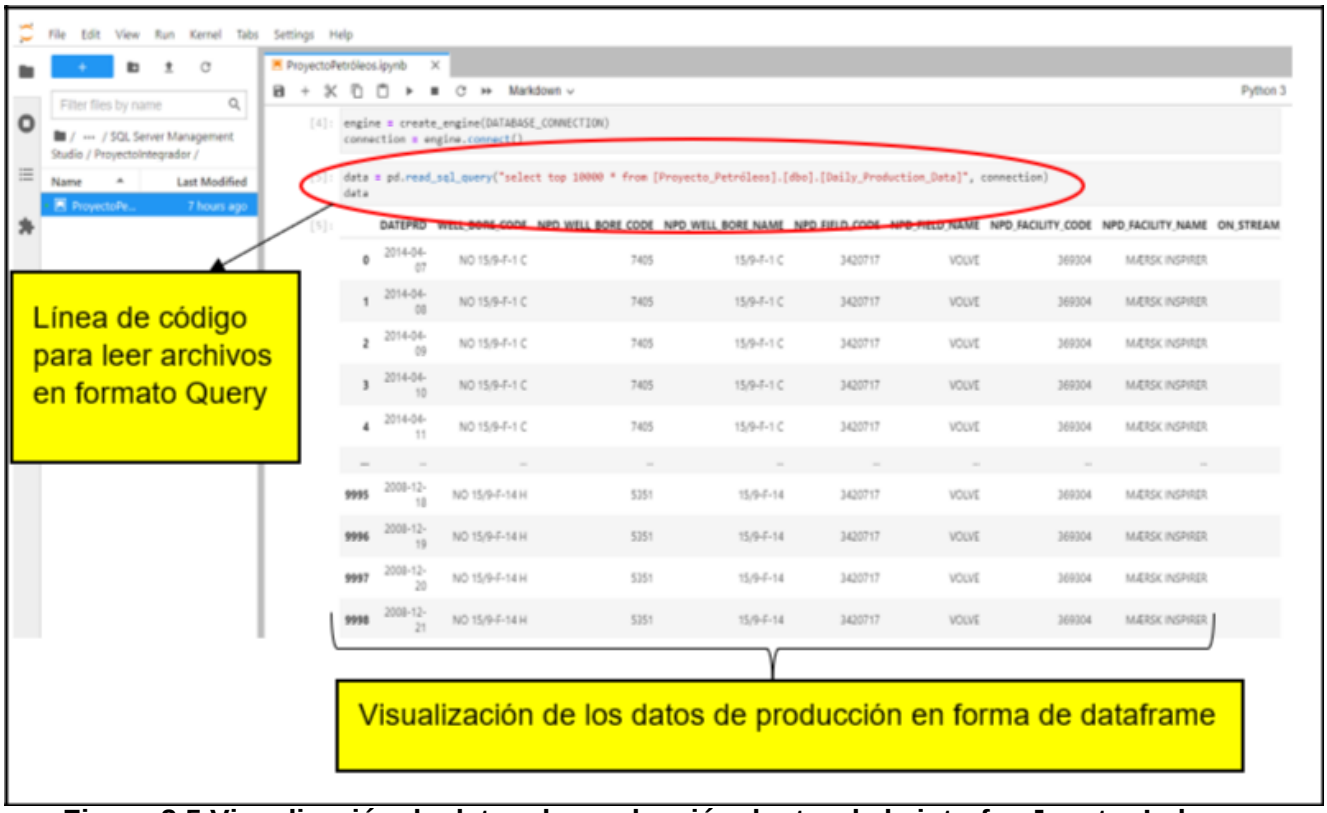


Figura 2.5 Visualización de datos de producción dentro de la interfaz Jupyter Lab (fuente: Autores) 21

2.2.1.2 Análisis Exploratorio de Datos

Análisis general del campo

Para nuestro análisis exploratorio de datos, se determinó que el campo Volve tiene cinco pozos productores de petróleo y dos pozos inyectores de agua. Uno de los pozos inyectores fue puesto a producción en los últimos cinco meses operativos del campo. El tiempo de producción por pozo es variado, pero de forma general el tiempo de producción fue de 8 años, en la tabla 2.1 definimos los nombres de los pozos productores e inyectores, tiempos de producción e inyección y el tipo de fluidos que generalmente producían.

Tabla 2.1 Pozos productores e inyectores del campo Volve (*Volve - Equinor.Com*, n.d.)

Nombre del pozo	Tipo de pozo	Rango de tiempo operacional	Tipos de fluidos
15/9-F-1 C	Productor	4/7/2014 - 4/21/2016	Gas/Petróleo/Agua
15/9-F-11 B	Productor	7/8/2013 - 9/17/2016	Gas/Petróleo/Agua
15/9-F-12	Productor	2/12/2008 - 9/17/2016	Gas/Petróleo/Agua
15/9-F-14	Productor	2/12/2008 - 9/17/2016	Gas/Petróleo/Agua
15/9-F-15 D	Productor	1/12/2014 - 9/17/2016	Gas/Petróleo/Agua
15/9-F-4	Inyector	9/1/2007 - 12/1/2016	Agua
15/9-F-5	Inyector	9/1/2007 - 4/10/2016	Agua
15/9-F-5	Productor	04/11/2016 - 9/18/2016	Gas/Petróleo/Agua

Originalmente el plan de operación del campo dictaminaba que iban a ser perforados tres pozos productores y tres pozos inyectores de agua como método de desplazamiento del fluido de reservorio, complementados con Gas Lift y bombas BES para facilitar el movimiento del fluido hacia superficie. Finalmente, se puede observar en la tabla descrita en la parte superior que el campo inició sus operaciones con dos pozos productores (15/9-F-12 y 15/9-F-14) y en el transcurso de sus operaciones añadieron tres pozos productores más (15/9-F-1 C, 15/9-F-11 y 15/9-F-15 D). Utilizando incluso el pozo inyector 15/9-F-5 como pozo productor en los últimos cinco meses de operación del campo como se dijo anteriormente. Además, se observa que el campo Volve trabajó únicamente con dos pozos inyectores de agua a lo largo de sus operaciones y no con tres como se pretendía inicialmente.

Análisis de los datos

El análisis de los datos del campo fue realizado de forma secuencial, resaltando los aspectos más importantes, como, por ejemplo:

1. Conocer las diferentes variables que integran al conjunto de datos del campo
2. Conocer las medidas de dispersión asociadas a cada variable
3. Mapa de calor indicando las correlaciones que existen entre variables
4. Visualización del comportamiento de las curvas de producción por pozo

Estos 4 aspectos son presentados en las gráficas expuestas a continuación.

```
#Conociendo los Datos de forma general
print(prod_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15634 entries, 0 to 15633
Data columns (total 24 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   DATEPRD                    15634 non-null  datetime64[ns]
 1   WELL_BORE_CODE             15634 non-null  object
 2   NPD_WELL_BORE_CODE         15634 non-null  int64
 3   NPD_WELL_BORE_NAME         15634 non-null  object
 4   NPD_FIELD_CODE             15634 non-null  int64
 5   NPD_FIELD_NAME             15634 non-null  object
 6   NPD_FACILITY_CODE          15634 non-null  int64
 7   NPD_FACILITY_NAME          15634 non-null  object
 8   ON_STREAM_HRS              15349 non-null  float64
 9   AVG_DOWNHOLE_PRESSURE      8980 non-null   float64
 10  AVG_DOWNHOLE_TEMPERATURE   8980 non-null   float64
 11  AVG_DP_TUBING              8980 non-null   float64
 12  AVG_ANNULUS_PRESS          7890 non-null   float64
 13  AVG_CHOKE_SIZE_P           8919 non-null   float64
 14  AVG_CHOKE_UOM              9161 non-null   object
 15  AVG_WHP_P                  9155 non-null   float64
 16  AVG_WHT_P                  9146 non-null   float64
 17  DP_CHOKE_SIZE              15340 non-null  float64
 18  BORE_OIL_VOL               9161 non-null   float64
 19  BORE_GAS_VOL               9161 non-null   float64
 20  BORE_WAT_VOL               9161 non-null   float64
 21  BORE_WI_VOL                5706 non-null   float64
 22  FLOW_KIND                  15634 non-null  object
 23  WELL_TYPE                  15634 non-null  object
dtypes: datetime64[ns](1), float64(13), int64(3), object(7)
memory usage: 2.9+ MB
None
```

Figura 2.6 Variables que conforman el conjunto de datos de producción del campo Volve (fuente: Autores)

Por cuestiones de practicidad y entendimiento se define cada una de estas variables en la tabla 2.2.

Tabla 2.2 Definición de las variables que conforman el conjunto de datos (*Volve - Equinor.Com*, n.d.)

#	Variable	Definición
1	DATEPRD	Fecha de producción
2	WELL_BORE_CODE	Código del pozo
3	NPD_WELL_BORE_CODE	Código del pozo dado por la Dirección Noruega del Petróleo
4	NPD_WELL_BORE_NAME	Nombre del pozo dado por la Dirección Noruega del Petróleo
5	NPD_FIELD_CODE	Código del campo dado por la Dirección Noruega del Petróleo
6	NPD_FIELD_NAME	Nombre del campo dado por la Dirección Noruega del Petróleo
7	NPD_FACILITY_CODE	Código del establecimiento dado por la Dirección Noruega del Petróleo
8	NPD_FACILITY_NAME	Nombre del establecimiento dado por la Dirección Noruega del Petróleo
9	ON_STREAM_HRS	Horas de operación por día
10	AVG_DOWNHOLE_PRESSURE	Presión promedio en el fondo del pozo
11	AVG_DOWNHOLE_TEMPERATURE	Temperatura promedio en el fondo del pozo
12	AVG_DP_TUBING	Diferencial de presión promedio en la tubería de producción
13	AVG_ANNULUS_PRESS	Presión promedio en el espacio anular del pozo
14	AVG_CHOKE_SIZE_P	Presión de estrangulamiento promedio
15	AVG_CHOKE_UOM	Unidad de medida para el estrangulamiento
16	AVG_WHP_P	Presión promedio en el cabezal pozo
17	AVG_WHT_P	Temperatura promedio en el cabezal del pozo
18	DP_CHOKE_SIZE	Diferencial de presión de estrangulamiento
19	BORE_OIL_VOL	Volumen de Petróleo por día
20	BORE_GAS_VOL	Volumen de gas por día
21	BORE_WAT_VOL	Volumen de agua por día
22	BORE_WI_VOL	Volumen de agua inyectado por día
23	FLOW_KIND	Tipo de flujo
24	WELL_TYPE	Tipo de pozo

```
# Medidas de Dispersión por Variable
prod_df.describe()
```

	NPD_WELL_BORE_CODE	NPD_FIELD_CODE	NPD_FACILITY_CODE	ON_STREAM_HRS	AVG_DOWNHOLE_PRESSURE	AVG_DOWNHOLE_TEMPERATURE	AVG_DP_TUBING
count	15634.000000	15634.0	15634.0	15349.000000	8980.000000	8980.000000	8980.000000
mean	5908.581745	3420717.0	369304.0	19.994093	181.803869	77.162969	154.028787
std	649.231622	0.0	0.0	8.369978	109.712363	45.657948	76.752373
min	5351.000000	3420717.0	369304.0	0.000000	0.000000	0.000000	0.000000
25%	5599.000000	3420717.0	369304.0	24.000000	0.000000	0.000000	83.665361
50%	5693.000000	3420717.0	369304.0	24.000000	232.896939	103.186689	175.588861
75%	5769.000000	3420717.0	369304.0	24.000000	255.401455	106.276591	204.319964
max	7405.000000	3420717.0	369304.0	25.000000	397.588550	108.502178	345.906770

	AVG_ANNULUS_PRESS	AVG_CHOKE_SIZE_P	AVG_WHP_P	AVG_WHT_P	DP_CHOKE_SIZE	BORE_OIL_VOL	BORE_GAS_VOL	BORE_WAT_VOL	BORE_WI_VOL
count	7890.000000	8919.000000	9155.000000	9146.000000	15340.000000	9161.000000	9161.000000	9161.000000	5706.000000
mean	14.856100	55.168533	45.377811	67.728440	11.441060	1095.631548	161049.059703	1672.151332	5315.480815
std	8.406822	36.692924	24.752631	27.719028	19.816928	1323.538151	188136.410434	1706.982853	2181.486695
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-457.840000	0.000000
25%	10.841437	18.952989	31.148062	56.577834	0.000000	190.690000	29430.590000	19.870000	4338.204674
50%	16.308598	52.096877	37.933620	80.071250	2.384969	557.550000	87749.660000	1097.790000	5504.739769
75%	21.306125	99.924288	57.101268	88.062202	13.765020	1345.200000	202482.300000	3260.950000	6781.058040
max	30.019828	100.000000	137.311030	93.509584	125.718570	5901.840000	851131.520000	8019.740000	10013.600000

Figura 2.7 Medidas estadísticas de dispersión por columna/variable (fuente: Autores)

Uno de los objetivos principales dentro del análisis exploratorio de datos es conocer la correlación que tienen las variables entre ellas, por lo tanto, mediante el uso de la función de correlación en Python, pudimos determinar entre que variables existen las relaciones más estrechas. En la figura 2.8 se puede apreciar mediante el mapa de calor lo descrito anteriormente.

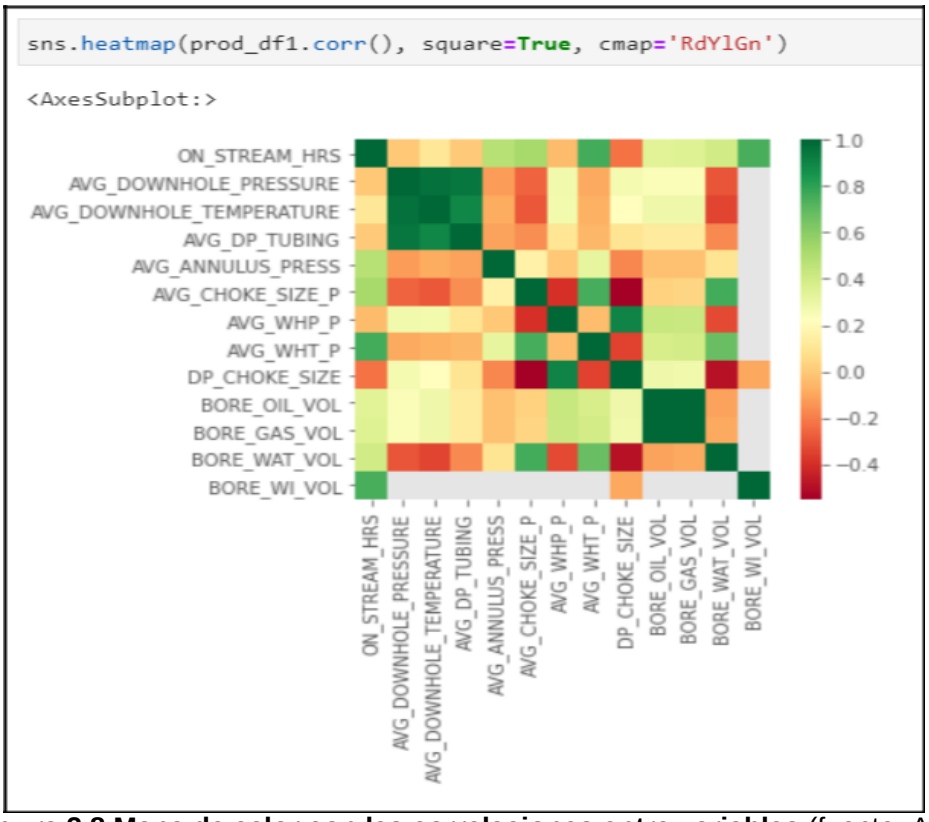


Figura 2.8 Mapa de calor con las correlaciones entre variables (fuente: Autores)

El mapa de calor representa una fuerte correlación entre la temperatura de fondo promedio, la presión de fondo promedio y la presión promedio en el tubing. Además, el mapa de calor nos muestra una fuerte correlación entre la producción de PETRÓLEO y GAS obviamente.

Finalmente, exponemos el comportamiento de las curvas de producción/inyección el que han sido reconocidos durante el proceso de exploración de datos, los de los pozos del campo donde se corrobora gráficamente lo expuesto de forma teórica desde un inicio en el análisis general del campo. Cada pozo presenta un código designado por la Dirección Noruega del Petróleo (NPD por sus siglas en inglés) con códigos al igual que el gráfico son presentados a continuación en la tabla 2.2 y figuras 2.9, 2.10 respectivamente.

Tabla 2.3 Reconocimiento de pozos que conforman el campo Volve mediante Nombre del pozo y Código NPD (*Volve - Equinor.Com*, n.d.)

Pozo #	Nombre del Pozo	Código del Pozo (NPD)
1	15/9-F-5	5769
2	15/9-F-4	5693
3	15/9-F-14	5351
4	15/9-F-12	5599
5	15/9-F-11 B	7078
6	15/9-F-15 D	7289
7	15/9-F-1 C	7405

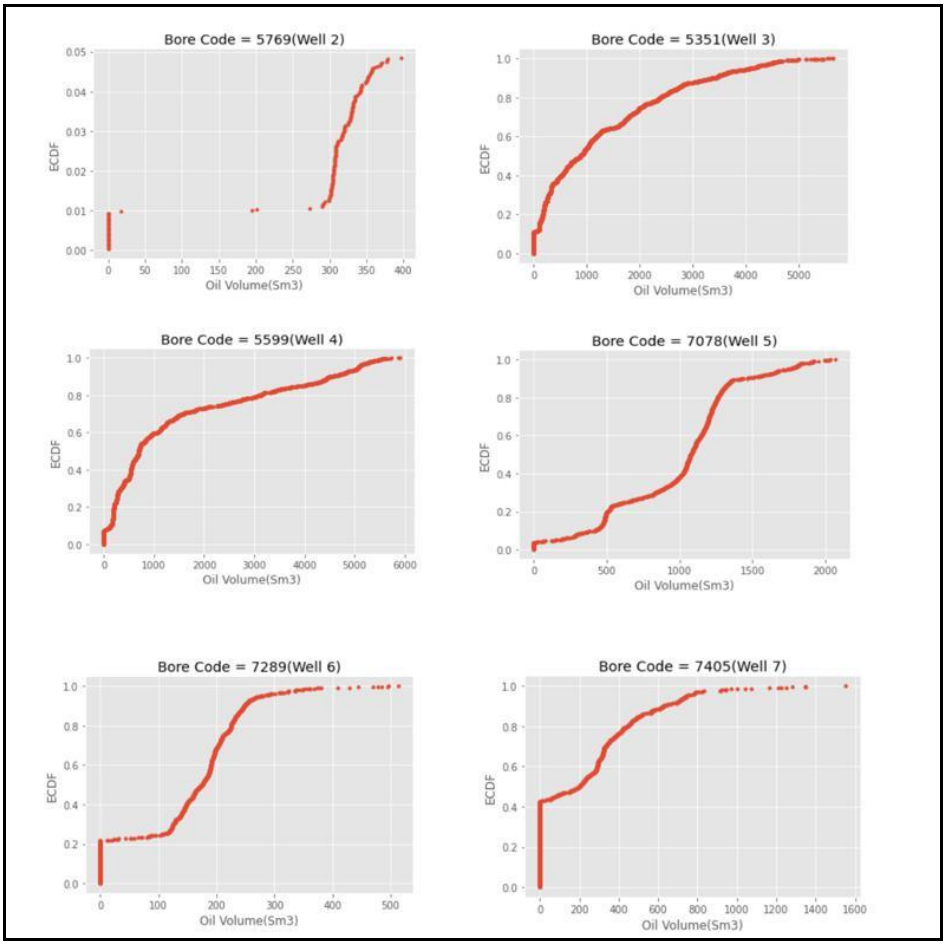


Figura 2.9 Gráficas de producción de petróleo por pozo con respecto al tiempo de funcionamiento (fuente: Autores)

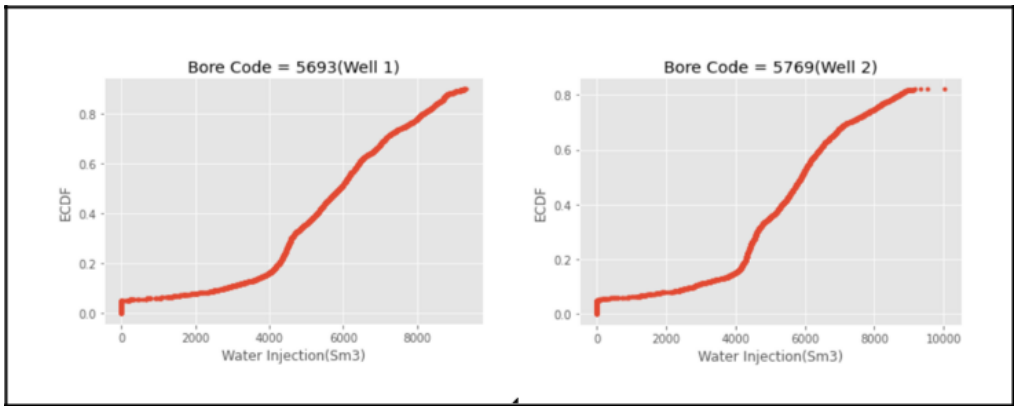


Figura 2.10 Gráficas de los pozos inyectores de agua con respecto al tiempo de funcionamiento (fuente: Autores)

2.2.1.3 Limpieza de datos y organización de datos normalizados

Una vez que los datos en bruto hayan sido analizados correctamente, la limpieza de estos datos fue el siguiente paso a seguir para la obtención de una base de datos correctamente estructurada. Dentro de la base de datos en bruto pudimos visualizar ciertas características que, a criterio nuestro y basado en el mapa de calor de las correlaciones de las variables, perjudicarían los futuros resultados de nuestros modelos predictivos, estas características fueron las siguientes:

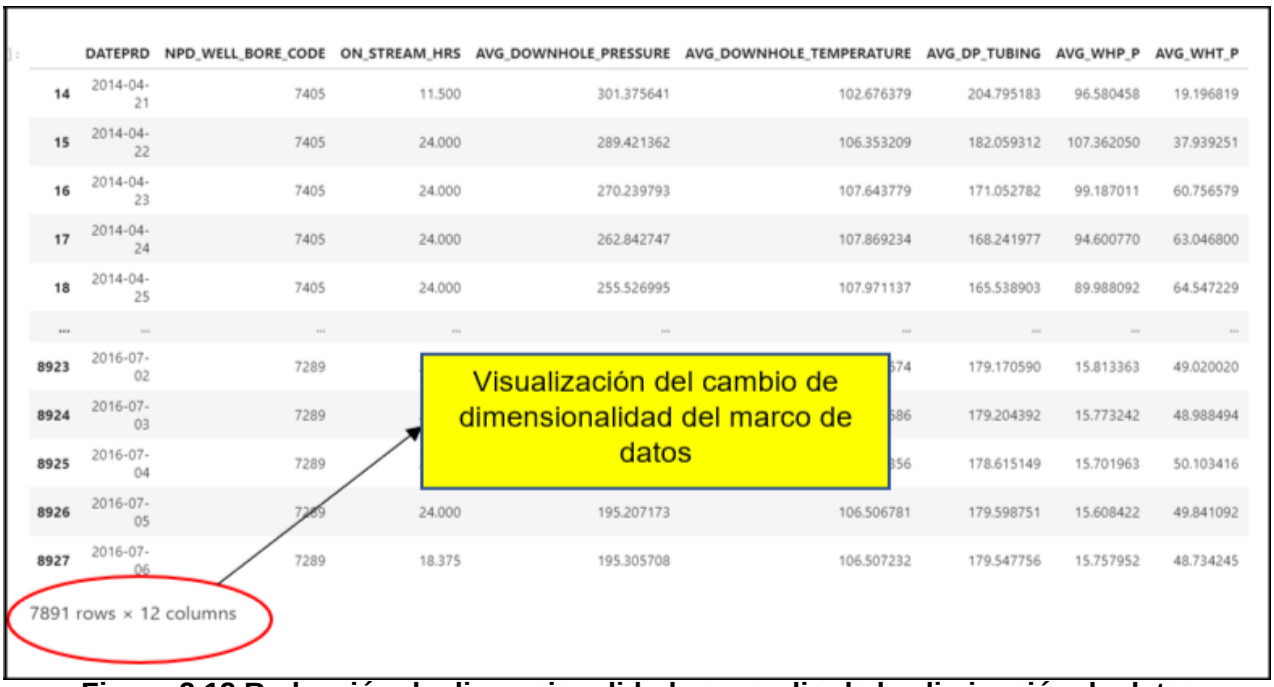
1. Columnas con datos categóricos
2. Columnas con celdas vacías o con valores iguales a cero
3. Valores anormales o con poca relación con respecto a sus valores antecesores y predecesores
4. Filas con datos de pozos inyectores
5. Columnas de datos con poca correlación basado en el mapa de correlaciones expuesto en el punto anterior

Eliminación y filtrado de datos innecesarios

Por medio de la exhaustiva revisión de datos y mediante el análisis de correlaciones previamente realizado, se procedió con la eliminación de las columnas de datos categóricos, de igual manera con la eliminación de las filas con datos pertenecientes al par de pozos inyectores de agua dentro del campo y con el filtrado de datos con horas de producción iguales a cero. En la figura 2.11, se detalla concretamente las líneas de código utilizadas en este proceso.

```python
#Dropping categorical columns
to_drop = ['WELL_BORE_CODE',
'NPD_WELL_BORE_NAME',
'AVG_CHOKE_SIZE_P',
'AVG_ANNULUS_PRESS',
'NPD_FIELD_CODE',
'NPD_FIELD_NAME',
'NPD_FACILITY_CODE',
'NPD_FACILITY_NAME',
'FLOW_KIND',
'WELL_TYPE',
'AVG_CHOKE_UOM',
'BORE_WI_VOL']
prod_df.drop(to_drop, inplace=True, axis=1)

#Dropping water injection rows
prod_df = prod_df.drop(labels=range(9001, 15634), axis=0)

#Filtering rows with zero stream hours
prod_df=prod_df[prod_df.ON_STREAM_HRS>0]

# Visualization of the dropped columns
prod_df
```

Figura 2.11 Líneas de código utilizadas para la eliminación de datos innecesarios (fuente: Autores)



	DATEPRD	NPD_WELL_BORE_CODE	ON_STREAM_HRS	AVG_DOWNHOLE_PRESSURE	AVG_DOWNHOLE_TEMPERATURE	AVG_DP_TUBING	AVG_WHP_P	AVG_WHT_P
14	2014-04-21	7405	11.500	301.375641	102.676379	204.795183	96.580458	19.196819
15	2014-04-22	7405	24.000	289.421362	106.353209	182.059312	107.362050	37.939251
16	2014-04-23	7405	24.000	270.239793	107.643779	171.052782	99.187011	60.756579
17	2014-04-24	7405	24.000	262.842747	107.869234	168.241977	94.600770	63.046800
18	2014-04-25	7405	24.000	255.526995	107.971137	165.538903	89.988092	64.547229
...	...	...	...	...	...	...	...	...
8923	2016-07-02	7289			674	179.170590	15.813363	49.020020
8924	2016-07-03	7289			686	179.204392	15.773242	48.988494
8925	2016-07-04	7289			856	178.615149	15.701963	50.103416
8926	2016-07-05	7289	24.000	195.207173	106.506781	179.598751	15.608422	49.841092
8927	2016-07-06	7289	18.375	195.305708	106.507232	179.547756	15.757952	48.734245

Figura 2.12 Reducción de dimensionalidad por medio de la eliminación de datos innecesarios (fuente: Autores)

Aplicando las líneas de código mostradas en la figura 2.11, se obtuvo como resultado un marco de datos (dataframe) reducido, que por lo general contiene los datos más relevantes que serán utilizados dentro del entrenamiento de cada modelo propuesto. Como se puede observar en la figura 2.12 la dimensionalidad del marco de datos ha cambiado significativamente con respecto a la original que presentaba una dimensión de 15634 filas con 24 columnas de datos.

Tratamiento de datos nulos y valores iguales a cero

Dentro del marco de datos reducido existen valores o datos iguales a cero, así como también datos nulos que deben obligatoriamente ser manipulados mediante la utilización de conceptos estadísticos. Para la corrección de estos datos mencionados existen diferentes técnicas dentro de Python que pueden ser aplicadas. Las más comunes son:

Forward Filling (Remplazo de datos usando el valor predecesor)

Mean Value (Reemplazo de datos usando el valor promedio)

Interpolation (Reemplazo de datos usando interpolaciones)

Pero para nuestro proyecto en específico hicimos uso en primer lugar de interpolaciones, esto porque al ser datos en series de tiempo necesitamos una técnica que se ajuste a una tendencia más real.

```
#Converting zero values into NaN values in order to apply interpolation or Forward Filling tool
prod_df.loc[prod_df['AVG_DOWNHOLE_PRESSURE'] == 0,'AVG_DOWNHOLE_PRESSURE'] = np.nan
prod_df.loc[prod_df['AVG_DOWNHOLE_TEMPERATURE'] == 0,'AVG_DOWNHOLE_TEMPERATURE'] = np.nan

datatoexcel=pd.ExcelWriter("Dataframe_NaN.xlsx", engine='xlsxwriter')
prod_df.to_excel(datatoexcel, sheet_name='Sheet1')
datatoexcel.save()

# Filling the missing spots (NaN's) of the dataset using "Forward Filling"
prod_df['ON_STREAM_HRS'] = prod_df['ON_STREAM_HRS'].fillna(method='pad')
prod_df['AVG_DOWNHOLE_PRESSURE'] = prod_df['AVG_DOWNHOLE_PRESSURE'].fillna(method='pad')
prod_df['AVG_DOWNHOLE_TEMPERATURE'] = prod_df['AVG_DOWNHOLE_TEMPERATURE'].fillna(method='pad')
prod_df['AVG_DP_TUBING'] = prod_df['AVG_DP_TUBING'].fillna(method='pad')
prod_df['AVG_WHP_P'] = prod_df['AVG_WHP_P'].fillna(method='pad')
prod_df['AVG_WHT_P'] = prod_df['AVG_WHT_P'].fillna(method='pad')
prod_df['DP_CHOKE_SIZE'] = prod_df['DP_CHOKE_SIZE'].fillna(method='pad')
prod_df['BORE_OIL_VOL'] = prod_df['BORE_OIL_VOL'].fillna(method='pad')
prod_df['BORE_GAS_VOL'] = prod_df['BORE_GAS_VOL'].fillna(method='pad')
prod_df['BORE_WAT_VOL'] = prod_df['BORE_WAT_VOL'].fillna(method='pad')

#Using interpolate function to fill the NaN values
prod_df=prod_df.interpolate(method='polynomial', order=2)
prod_df
```

Figura 2.13 Líneas de código ejemplo para la aplicación de las técnicas de Reemplazo por valor predecesor e Interpolación (fuente: Autores)

Normalización de datos finales

La normalización de datos es sin lugar a duda uno de los pasos más importantes dentro de un proyecto que involucre aplicación de algoritmos de Machine Learning. Por tal motivo, al momento de seleccionar las variables finales para el entrenamiento de los algoritmos es necesario aplicar una normalización correcta. Normalizar significa, en este caso, comprimir o extender los valores de la variable para que estén en un rango definido. Sin embargo, una mala aplicación de la normalización, o una elección descuidada del método de normalización puede arruinar los datos y por consiguiente el análisis.

A continuación, en la figura 2.14 mostramos el método de normalización de datos en nuestro proyecto conocido como MinMax Scaler.

```
#Scaling dataset to remove difference in distribution within columns
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
prod_df[['ON_STREAM_HRS','AVG_DOWNHOLE_TEMPERATURE','AVG_DOWNHOLE_PRESSURE','AVG_WHP_P'
        ,'AVG_WHT_P','AVG_DP_TUBING','DP_CHOKE_SIZE','BORE_OIL_VOL']] = scaler.fit_transform(prod_df[['ON_STREAM_HRS',
                                                                                                      'AVG_DOWNHOLE_TEMPERATURE',
                                                                                                      'AVG_DOWNHOLE_PRESSURE',
                                                                                                      'AVG_WHP_P',
                                                                                                      'AVG_WHT_P',
                                                                                                      'AVG_DP_TUBING',
                                                                                                      'DP_CHOKE_SIZE','BORE_OIL_VOL']])
```

Figura 2.14 Línea de código para la normalización de datos dentro del marco de datos final (fuente: Autores)

2.2.2 Creación de los modelos predictivos

2.2.2.1 División de datos normalizados y Modelos Predictivos

La fase dos de nuestro proyecto empieza con la base de datos lista para emplearse en los algoritmos, es necesario realizar una división de datos en dos subgrupos. Es entonces que del total de datos obtenidos se procede con la división basada en una regla de 80/20; esto quiere decir que 80 por ciento de los datos finales serán destinados para entrenamiento y el 20 por ciento restante para comprobatoria. En la figura 2.15 demostramos en simples líneas de código como se realiza dicho paso.

```
#Designation of the X and Y training and test datasets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size = 0.2)
x_train_final = X_train.drop(['DATEPRD','NPD_WELL_BORE_CODE'],axis = 1)
x_test_final = X_test.drop(['DATEPRD',"NPD_WELL_BORE_CODE"],axis = 1)
y_test_final = y_test['BORE_OIL_VOL']
y_train_final = y_train['BORE_OIL_VOL']
X = prod_df.drop(['DATEPRD','BORE_OIL_VOL'],axis=1).values
y= prod_df['BORE_OIL_VOL']
```

Figura 2.15 División de los datos normalizados para el respectivo entrenamiento y comprobatoria del modelo predictivo (fuente: Autores)

Modelo de Regresión Lineal

Regresión Lineal es uno de los algoritmos más simples que existen. La aplicación de este busca construir una relación lineal entre la producción de petróleo y las funciones de capacitación. La ecuación lineal asignará un coeficiente a cada característica de entrenamiento, y también se agregará una intersección a la ecuación.

Datos de entrada:

$$x = (x0, x1, \ldots, xn) \quad (2.1)$$

Salida prevista:

$$y = w0x0 + w1x1 + \ldots + wnxn + b \quad (2.2)$$

Para la creación del modelo predictivo mediante el algoritmo regresión lineal, hemos tomado en consideración los cinco pozos productores del campo Volve, los cuales, basado en el código NPD son; los pozos 5351, 5599, 7078, 7289 y 7405. Además, tal como se observa en la figura 2.16, para llamar a este algoritmo solo toca aplicar una mínima cantidad de líneas de código.

```
# Applying Linear Regression and getting the value of accuracy of this model
from sklearn.linear_model import LinearRegression
lin_reg= LinearRegression()
lin_reg.fit(x_train_final, y_train_final)
y_pred = lin_reg.predict(x_test_final)
```

Figura 2.16 Aplicación del algoritmo regresión lineal para predecir producción de petróleo (fuente: Autores)

Modelo de Regresión Polinomial

Si bien el modelo de regresión lineal solo asume la relación lineal entre la producción de petróleo y las características, sabemos que la relación no es ideal o no es tan simple y el modelo lineal no se ajusta bien. Entonces, para resolver este problema y obtener una correlación más precisa, la regresión polinomial entra en escena. Básicamente, convertiremos nuestras características en sus órdenes superiores y luego aplicaremos regresión lineal en estos términos de características de orden superior.

En la figura 2.17 se procede a demostrar las líneas de código necesarias para la aplicación del algoritmo mencionado, recalcando que para la creación de este modelo se tomó en cuenta únicamente los pozos que produjeron petróleo desde el primer hasta el último día de su ciclo de producción. Estos pozos, basados en su código NPD son; los pozos 5351, 5599 y 7078.

```
# Applying Polynomial Regression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(x_train_final)
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y_train_final)
x_pol_test = poly_reg.fit_transform(x_test_final)
X_poly.shape[1]
```

Figura 2.17 Aplicación del algoritmo regresión polinomial para predecir producción de petróleo (fuente: Autores)

Modelo de ARIMA

Dentro de los métodos de exploración de series de tiempo, que trabajan con información del pasado para predecir valores del futuro con una buena precisión, es el muy conocido modelo estadístico lineal ARIMA; que se lo emplea en muchos campos a partir de que se fundamenta en las características de ciertos parámetros. ARIMA es un muy buen modelo que filtra las tendencias de los modelos lineales que estudia.(Fan et al., 2021)

Para el modelo, es importante contar con una serie de datos estacionarios, esto quiere decir que la correlación entre las mismas variables debe tener una tendencia lineal. Dicho esto, primero se grafica los datos de producción de

petróleo de los pozos para ver si se cumple, caso contrario es necesario realizar la modificación a una serie estacionaria.

En la figura 2.18 como se puede observar, se procedió a graficar la producción de petróleo dada por el pozo 7405 para entender su comportamiento.

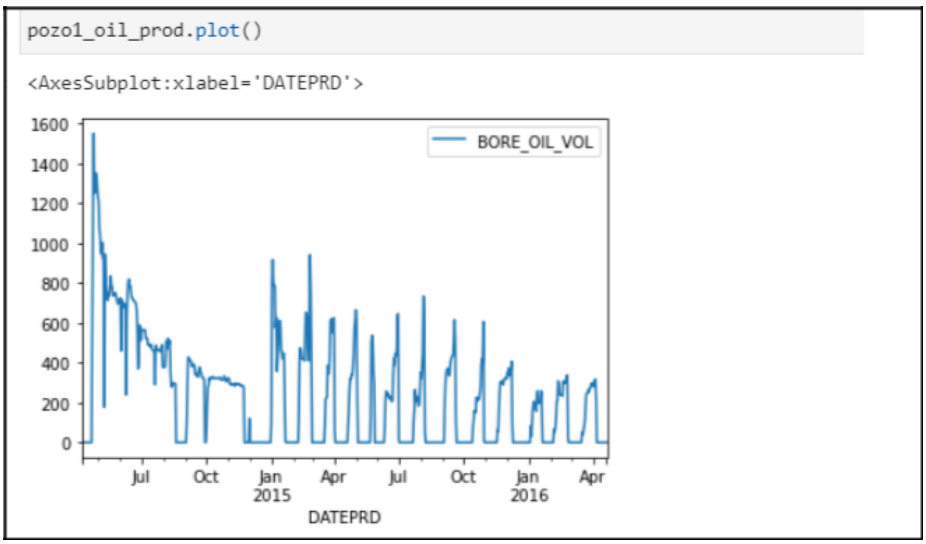


Figura 2.18 Datos de producción vs tiempo del pozo 7405 (fuente: Autores)

Se grafican los datos, y se procede entonces a llamar a una librería que cuenta con este algoritmo predictivo, con la finalidad de graficar la curva de autocorrelaciones y ver si corresponde a una serie estacionaria.

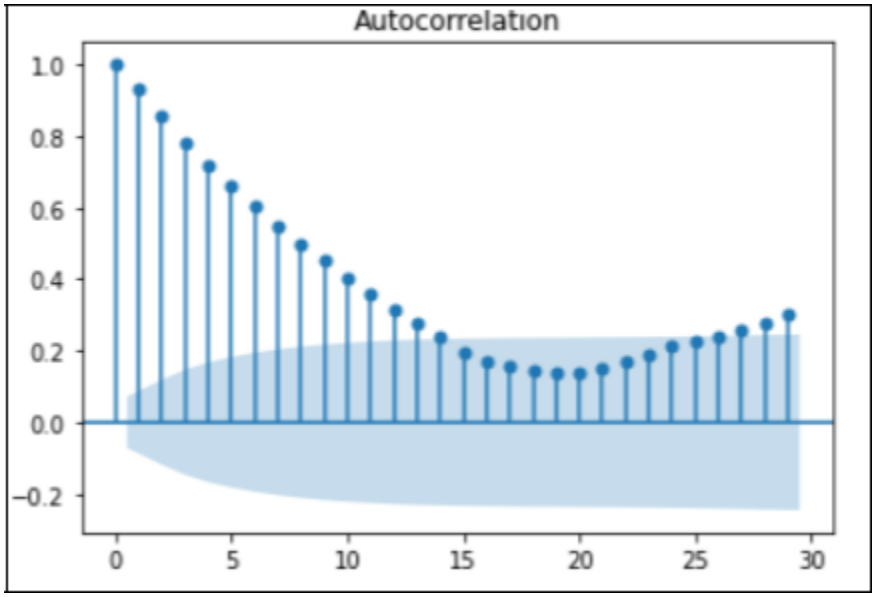


Figura 2.19 Curva de autocorrelación de la variable de producción de petróleo (fuente: Autores)

Analizando la gráfica figura 2.19 se puede observar que no corresponde a una serie estacionaria, y nos lleva a realizar la modificación necesaria por medio de la función diff en Python (figura 2.20).

```
pozo1_prod_diff=pozo1_oil_prod.diff(periods=1)
#integrated of order 1, denoted by d (for diff), one of the paramethers on ARIMA model
pozo1_prod_diff=pozo1_prod_diff[1:]
```

Figura 2.20 Línea de código ejecutando la función diff para la obtención de datos estacionarios (fuente: Autores)

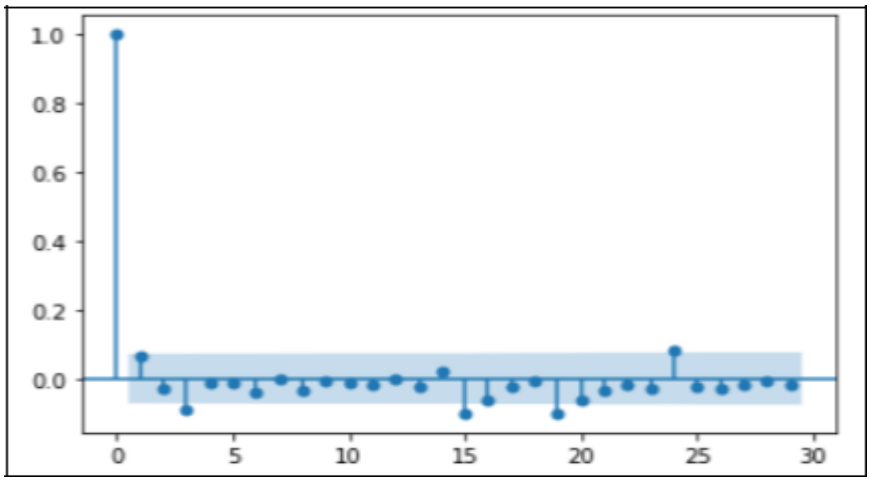


Figura 2.21 Gráfico de autocorrelación de producción de petróleo (fuente: Autores)

Tal como lo muestra la figura 2.22, a partir de ese momento se puede realizar la implementación del modelo autorregresivo AR y ARIMA para el entrenamiento del modelo.

```
from statsmodels.tsa.ar_model import AR
from sklearn.metrics import mean_squared_error
```

```
from statsmodels.tsa.arima_model import ARIMA
```

Figura 2.22 Líneas de código de la ejecución del algoritmo ARIMA para la creación del modelo predictivo (fuente: Autores)

El modelo ARIMA requiere de ciertos parámetros que se deben de encontrar para ser empleados en la autoregresión, estos son:

“p”: periodos tomados para el modelo autorregresivo.

“d”: diferencia, orden integrado.

“q”: períodos en el modelo de media móvil.

Existe una combinación que debe de ser encontrado entre los valores que toman estos parámetros para mejorar la precisión del modelo en sus predicciones y que genera el menor error cuadrático medio.

Modelo de Random Forest

El modelo predictivo Random Forest es uno de los modelos más implementados para el trabajo de predicciones, pues emplea un conjunto de árboles de decisión por separados que se encuentran entrenados en base a distintas muestras de datos seleccionados aleatoriamente. Cada uno de estos árboles de decisión se entrenarán entonces en base a datos que ligeramente varían, y en cada uno de estos árboles las observaciones se distribuirán por bifurcaciones o nodos generando la forma de estructura de un árbol hasta llegar al nodo terminal. Para dar los resultados de predicción el modelo emplea la media de las predicciones de cada árbol, esto le permite al modelo contar con resultados muy buenos de predicciones.

En nuestro programa, tal como lo muestra la figura 2.23, se emplearon las siguientes líneas de código para la implementación del modelo:

```
rf = RandomForestRegressor(random_state = 42)

# Use the random grid to search for best hyperparameters
# Random search of parameters, using 3 fold cross validation,
# search across different combinations.
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid, n_iter = 15, scoring='neg_mean_absolute_error',  cv = 3, verbose=2, random_state=42, n_jobs=-1, return_train_score=True)

# Fit the random search model
rf_random.fit(X_train, y_train);
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
```

Figura 2.23 Implementación del modelo predictivo por medio del algoritmo Random Forest (fuente: Autores)

En el desarrollo del modelo, fue de mucha importancia realizar un tuneo de hiperparámetros para la búsqueda de los mejores valores para nuestro modelo, y es lo que se puede observar en la figura 2.24.

```
# Obtaining the best parameters
rf_random.best_params_

{'n_estimators': 1894,
 'min_samples_split': 3,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': None,
 'bootstrap': True}
```

Figura 2.24 Selección de los mejores valores para los parámetros del modelo (fuente: Autores)

2.2.3 Creación del aplicativo web

La fase tres de nuestro proyecto se basó en la integración de los modelos de predicción definidos en la fase anterior por medio de un aplicativo web que permite realizar predicciones de producción de petróleo, basándose en los algoritmos de Regresión Polinomial y Random Forest. La creación del aplicativo web se la realizó durante tres etapas distintas, considerando una arquitecta cliente– servidor, la implementación de un Api Rest y el uso de tecnologías de código abierto.

2.2.3.1 Desarrollo del Backend

La aplicación necesitó un servidor web que se encargue de recibir y procesar las solicitudes, el lenguaje de programación escogido para este fin es una vez más Python, pero en conjunto con el framework Django. Django permite desarrollar aplicaciones web tanto del lado del cliente como del servidor, mediante el patrón MVC (modelo - vista - controlador).

El servidor funcionará como un Api Rest, por lo cual se utilizará una librería especializada para esta tarea conocida como Django Rest Framework. Las solicitudes web se realizarán por medio de los métodos http (GET, POST, PUT, DELETE) y las respuestas se darán en formato JSON. a continuación, se describen los servicios a implementar:

Carga de archivo - POST /api/csv/upload/

Permite cargar un archivo csv con los datos de entrenamiento y prueba del modelo.

Parámetros:

Tabla 2.4 Generalidades del parámetro "data" (fuente: Autores)

Nombre	Tipo	Descripción
data	Archivo	Archivo de texto *.csv, con los datos usados para el entrenamiento y prueba.

```
POST http://localhost:8000/api/csv/upload/
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="text"

title
------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="data"; filename="1.csv"
Content-Type: text/csv

< ./1.csv
------WebKitFormBoundary7MA4YWxkTrZu0gW--
```

Figura 2.25 Líneas de código necesarias para cargar archivos csv en el aplicativo web (fuente: Autores)

Respuesta:

Tabla 2.5 Definición de las primeras variables-respuesta dentro del aplicativo web (fuente: Autores)

Nombre	Tipo	Descripción
_id	Texto	Identificador asignado para el archivo csv cargado.
columns	Lista	Listado con los nombres de las variables del csv.
covariance	Objeto	Objeto que contiene los datos de la covarianza de las variables del csv.

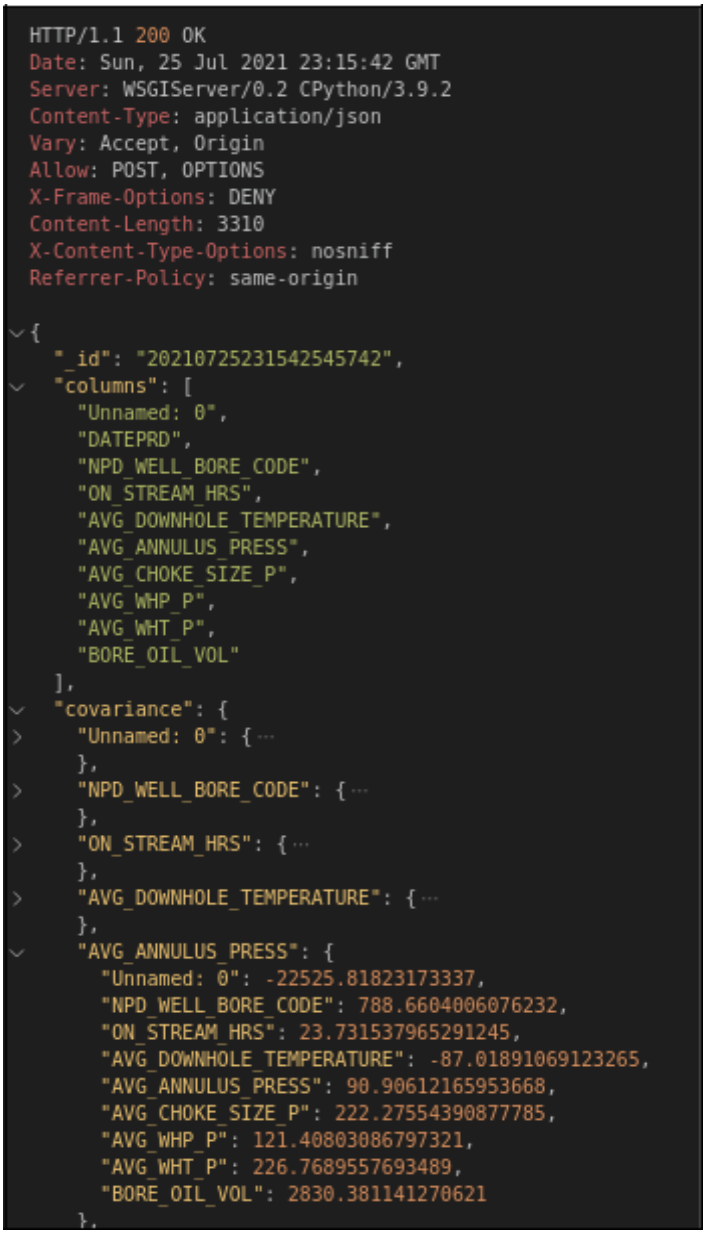


Figura 2.26 Líneas de código empleadas para la carga del archivo, selección de variables y cálculo de covarianza (fuente: Autores)

Entrenamiento del modelo: POST /api/learning/

Permite realizar el entrenamiento del modelo basado en los algoritmos de Regresión polinomial y Random Forest.

Parámetros:

Tabla 2.5 Definición de parámetros adicionales para la ejecución visual dentro del aplicativo web (fuente: Autores)

Nombre	Tipo	Descripción
_id	Texto	Identificador del csv que debe de ser utilizado para el entrenamiento.
algorithm	Número	Algoritmo utilizado para el entrenamiento. 1: Regresión polinomial. 2: Random Forest.
test	Número	Porcentaje del dataset que debe ser usado para la prueba del modelo entrenado.
x_plot	Texto	Variable utilizada para graficar como eje de las x en el modelo final.
y_plot	Texto	Variable utilizada para graficar como eje de las y en el modelo final.
group_by	Texto	Variable de agrupación de los pozos petroleros, se debe indicar una variable dentro del csv que permita identificar a los pozos petroleros utilizados para el entrenamiento.
labels	Lista	Listado de variables cuantitativas utilizadas para realizar el entrenamiento del modelo.

```
POST http://localhost:8000/api/learning/
content-type: application/json

{
    "_id": "20210725231542545742",
    "x_plot": "DATEPRD",
    "y_plot": "BORE_OIL_VOL",
    "group_by": "NPD_WELL_BORE_CODE",
    "labels": [
        "ON_STREAM_HRS",
        "AVG_DOWNHOLE_TEMPERATURE",
        "AVG_ANNULUS_PRESS",
        "AVG_CHOKE_SIZE_P",
        "AVG_WHP_P",
        "AVG_WHT_P"
    ]
}
```

Figura 2.27 Líneas de código implementadas para la implementación visual de resultados de predicción de producción (fuente: Autores)

Respuesta:

Tabla 2.6 Variables-respuesta asociadas al valor de precisión del modelo (fuente: Autores)

Nombre	Tipo	Descripción
score	Número	Resultado final del entrenamiento (0 – 100).
predictions	Lista	Listado de objetos con la predicción realizada por cada pozo petrolero, comparado con el modelo entrenado. La estructura de cada objeto de la lista es idéntica a la descrita en la respuesta de las predicciones.

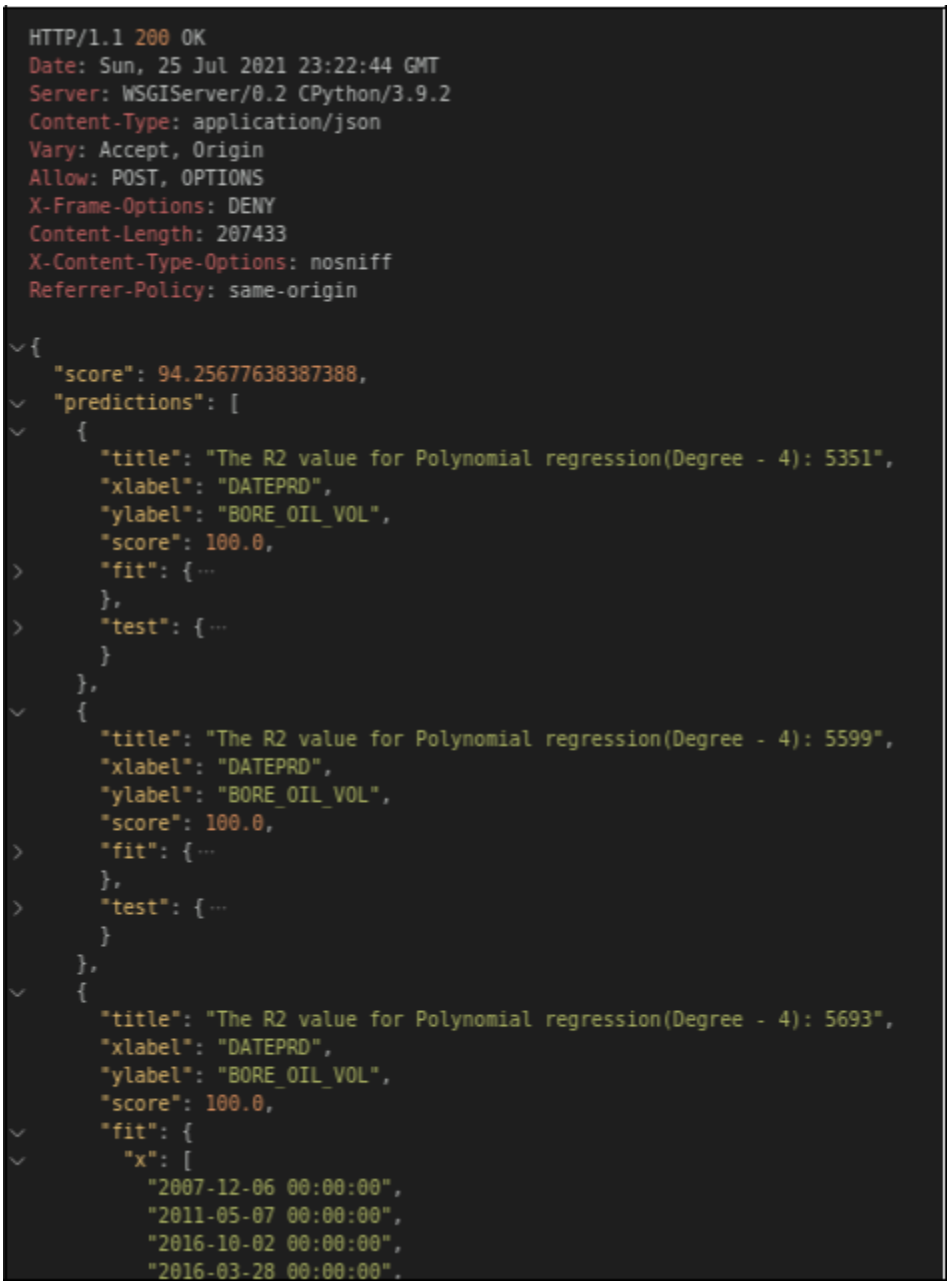


Figura 2.28 Implementación de los modelos entrenados dentro de la interfaz del aplicativo web (fuente: Autores)

Predicciones: POST /api/learning/predict/

Permite realizar predicciones con datos de producción utilizando un modelo ya entrenado.

Parámetros:

Tabla 2.7 Definición de parámetros asociados a la selección del modelo para la predicción de petróleo (fuente: Autores)

Nombre	Tipo	Descripción
_id	Texto	Identificador del modelo entrenado, igual al identificador del csv usado para el entrenamiento.
code	Texto	Identificador del pozo petrolero sobre el cual se realizará la predicción.
data	Objeto	Datos de producción utilizados para realizar la predicción.

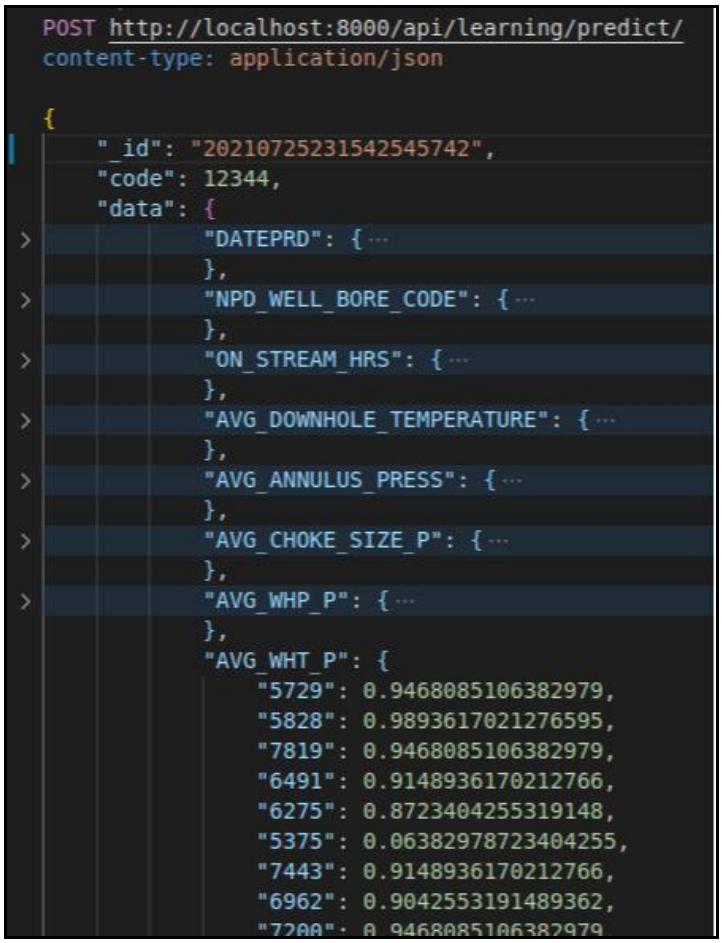


Figura 2.29 Líneas de código empleadas para la ejecución del modelo escogido para las predicciones de producción de petróleo

(fuente: Autores)

Respuesta:

Tabla 2.7 Definición de las variables-respuesta asociadas a la visualización de la predicción de petróleo (fuente: Autores)

Nombre	Tipo	Descripción
title	Texto	Descripción de la predicción realizada.
xlabel	Texto	Variable usada para gráficas en el eje de las x.
ylabel	Texto	Variable usada para gráficas en el eje de las y.
algorithm	Número	Algoritmo utilizado para el entrenamiento. 1: Regresión polinomial. 2: Random Forest.
score	Número	Resultado de la predicción (0 – 100)
fit	Objeto	Conjunto de puntos (x, y) que representan una curva obtenida con el entrenamiento del modelo.
test	Objeto	Conjunto de puntos (x, y) que representan una curva obtenida con los datos de producción

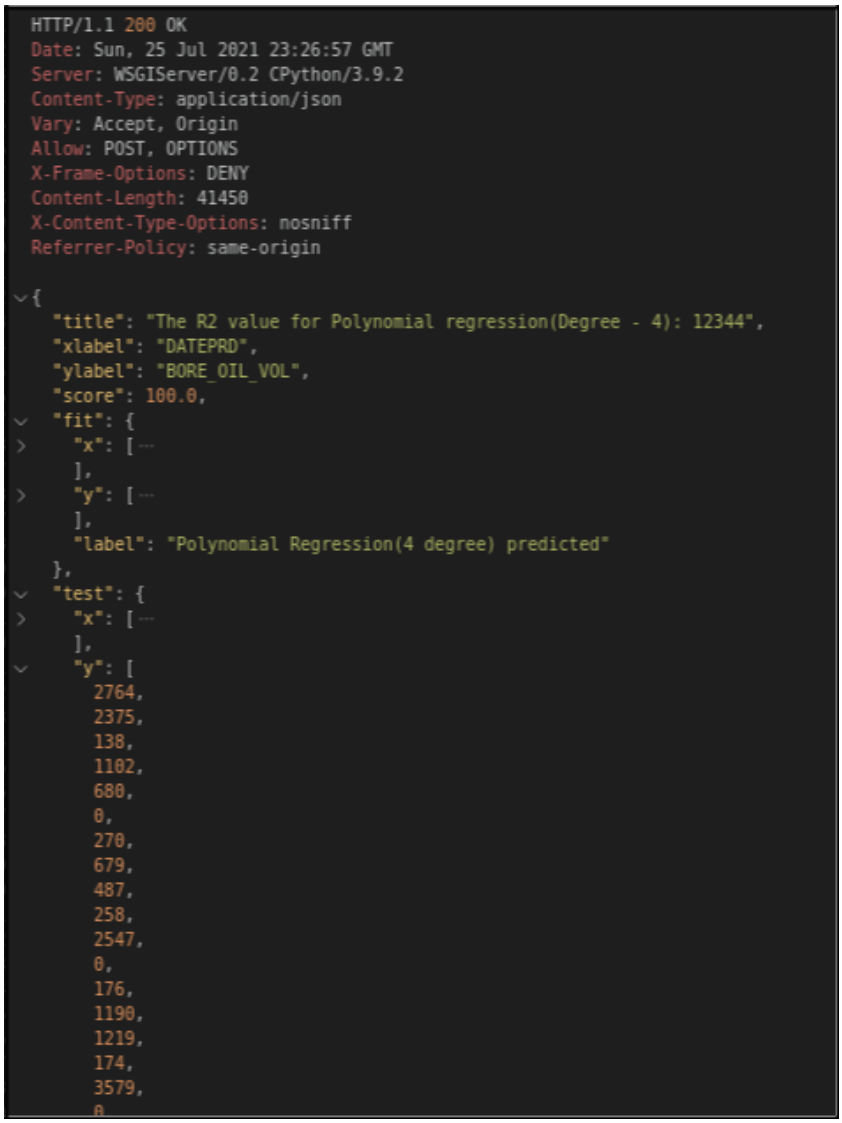


Figura 2.30 Líneas de código necesarias para la representación gráfica de las predicciones de petróleo (fuente: Autores)

2.2.4 Contraste de modelos físicos vs modelos de Machine Learning

La fase cuatro, es la última fase de nuestro proyecto y se basó en la comparación que existe entre la aplicación de los modelos físicos versus los modelos hechos en base a algoritmos predictivos de Machine Learning. La obtención de datos, procesos y resultados de los modelos físicos se obtuvieron netamente mediante un exhaustivo análisis por medio de una revisión bibliográfica; esto, debido a la dificultad de acceso a softwares de simulación y predicción a causa de sus altos costos.

Por lo expuesto anteriormente, en el capítulo siguiente presentaremos tablas de similitudes y diferencias que existen entre estos modelos físicos y nuestro aplicativo

web llamado “Saviour”. Haciendo hincapié obviamente en sus niveles de precisión para lo que a predicción de producción de petróleo respecta.

Como se puede observar, la fase final de nuestro proyecto es sencilla, pero de igual forma es muy necesaria, ya que, mediante esta se pueden tomar importantes decisiones como, por ejemplo, la que consideramos más importante que es, la aplicación definitiva de la ciencia de datos y machine learning en procesos asociados a la industria de petróleo y gas.

CAPÍTULO 3

3. RESULTADOS Y ANÁLISIS

3.1 Base de datos normalizada

Tal y como se ha planteado en los objetivos de este proyecto, la creación de una base de datos normalizada es básicamente la parte fundamental que puede existir dentro de la creación de un modelo predictivo, ya que, sin el acceso a un conjunto de datos, tales predicciones nunca se podrán llevar a cabo.

La normalización de los datos finales o también llamados durante el desarrollo de este proyecto como datos de entrenamiento se realizaron mediante una variedad de procesos conocidos como eliminación de variables categóricas, filtración de filas necesarias, reemplazo de valores nulos y la selección final de las variables de entrenamiento. Tales procesos partieron de un análisis profundo de los datos conocido como "Análisis Exploratorio de Datos" o EDA (por sus siglas en inglés). Partiendo de lo mencionado, a continuación, en la figura 3.1 se muestra como luce un conjunto de datos normalizado y listo para ser ejecutado dentro de un algoritmo de predicción.

	DATEPRD	NPD_WELL_BORE_CODE	ON_STREAM_HRS	AVG_DOWNHOLE_PRESSURE	AVG_DOWNHOLE_TEMPERATURE	AVG_DP_TUBING	AVG_WHP_P	AVG_WHT_P
14	2014-04-21	7405	0.456193	0.957160	0.946132	0.790448	0.798921	0.205293
15	2014-04-22	7405	0.959718	0.918949	0.980130	0.702695	0.888107	0.405726
16	2014-04-23	7405	0.959718	0.857637	0.992063	0.660213	0.820483	0.649736
17	2014-04-24	7405	0.959718	0.833993	0.994148	0.649364	0.782545	0.674228
18	2014-04-25	7405	0.959718	0.810609	0.995090	0.638931	0.744389	0.690274
...	...	...	...	...	...	...	...	...
8923	2016-07-02	7289	0.959718	0.617090	0.981650	0.691545	0.130809	0.524225
8924	2016-07-03	7289	0.959718	0.617069	0.981631	0.691676	0.130478	0.523887
8925	2016-07-04	7289	0.959718	0.614958	0.981684	0.689401	0.129888	0.535810
8926	2016-07-05	7289	0.959718	0.617803	0.981550	0.693198	0.129114	0.533005
8927	2016-07-06	7289	0.733132	0.618118	0.981554	0.693001	0.130351	0.521168

7891 rows × 10 columns

Figura 3.1 Visualización del marco de datos normalizado en la interfaz Jupyter Lab (fuente: Autores)

La normalización de datos tuvo como objetivo alinear toda la distribución de valores ajustados. El objetivo principal era crear una escala común de variables

independientes sin distorsionar las diferencias en los rangos de valores. Por otro lado, la reducción de dimensionalidad simplificó el modelado de los datos, redujo el tiempo de cálculo y finalmente permitió la visualización de grandes bases de datos.

3.2 Modelos predictivos

3.2.1 Regresión Lineal

Es un modelo de mínimos cuadrados que se ajusta a un modelo lineal, y su objetivo es minimizar la suma de cuadrados de los residuos entre los objetivos observados en el conjunto de datos y los objetivos predichos por aproximación lineal(Mohamed et al., 2020).

Al ser el algoritmo más simple que pueda existir, se utilizó para poder corroborar su precisión ante el entrenamiento de un conjunto de datos ordenados. Manipulando y tratando correctamente los datos de los cinco pozos productores del campo Volve para el entrenamiento de este algoritmo, obtuvimos los siguientes resultados.

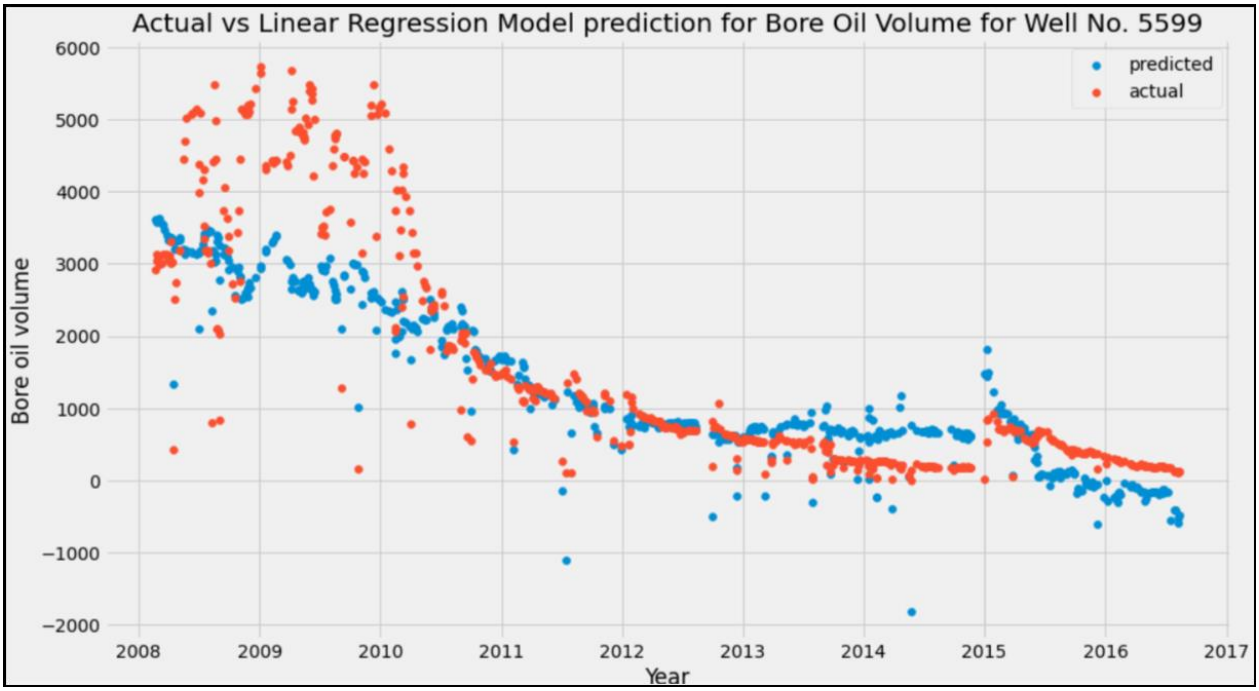


Figura 3.2 Comparación de la producción actual vs producción predicha del pozo 5599 (fuente: Autores)

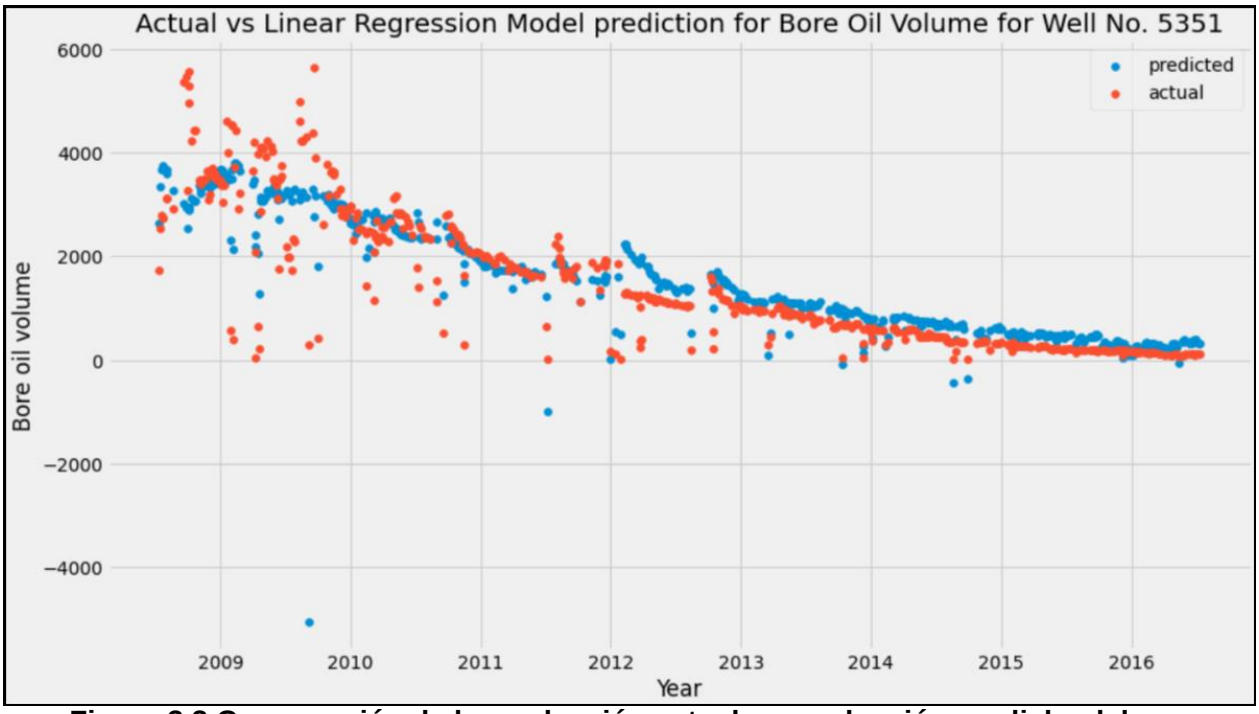


Figura 3.3 Comparación de la producción actual vs producción predicha del pozo 5351 (fuente: Autores)

La precisión para la predicción de producción de petróleo en el pozo 5599 y 5351 tuvo un valor de 0.69 y 0.80 respectivamente, lo que se puede discernir como precisiones muy aceptables, así como también se puede confirmar en la figura 3.2 y 3.3 mediante las tendencias similares entre la producción actual y la predicha. La razón por la cual dichas predicciones arrojaran tales precisiones fue la cantidad de datos de producción del pozo, y es que el pozo 5599 y 5351 tuvieron un tiempo operacional aproximado de 9 años, gestionando de esta manera una cantidad alta de datos de producción favorable para crear un modelo predictivo de precisión general de 0.65.

A diferencia de los pozos mencionados anteriormente, los pozos 7078, 7289 y 7405 tuvieron unas precisiones anormales, siendo específicos por debajo de cero. Estos valores anormales se produjeron debido a que estos pozos tienen un historial de producción menor a tres años. Lo que hace deducir que regresión lineal, no es una buena elección para predecir valores de producción cuando existe un conjunto de datos extremadamente reducido. En las figuras 3.4, 3.5 y 3.6 mostramos las tendencias anormales que obtuvieron las predicciones de producción de petróleo.

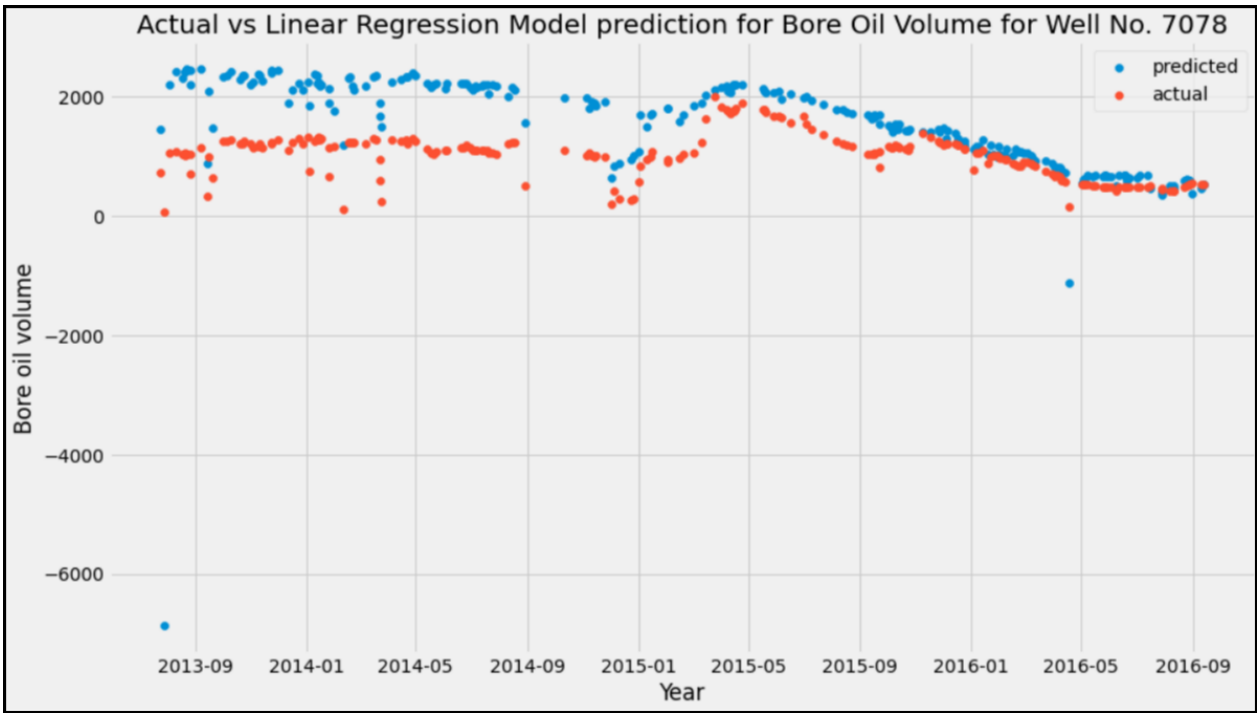


Figura 3.4 Comparación de la producción actual vs producción predicha del pozo 7078 (fuente: Autores)

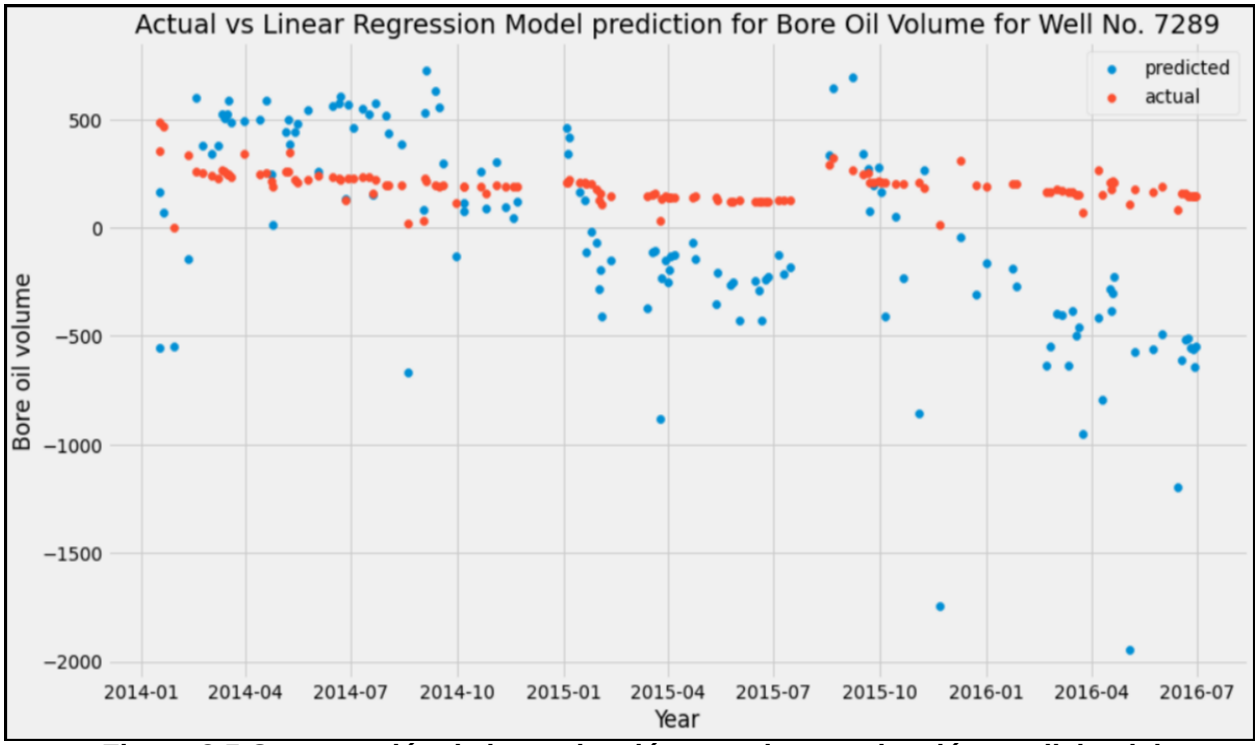


Figura 3.5 Comparación de la producción actual vs producción predicha del pozo 7289 (fuente: Autores)

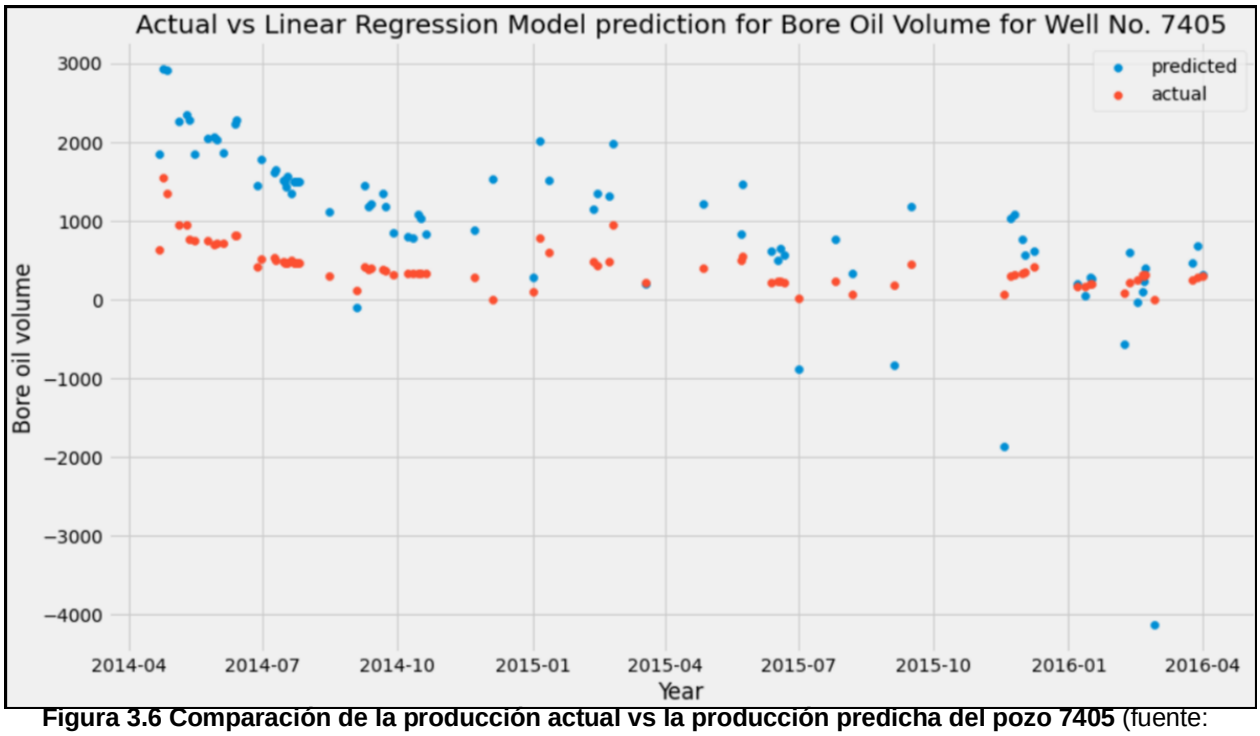


Figura 3.6 Comparación de la producción actual vs la producción predicha del pozo 7405 (fuente: Autores)

3.2.2 Regresión Polinomial

Agregando grados más altos, como cuadrático, convierte la línea en una curva que se ajusta mejor a los datos. Generalmente, este algoritmo se utiliza cuando los puntos del conjunto de datos están dispersos y el modelo lineal no puede describir el resultado con claridad.

El análisis para la aplicación de este algoritmo es distinto al de regresión lineal y es que, al ser susceptible a valores atípicos, se utilizaron únicamente datos de los pozos que estuvieron en producción durante todo su ciclo operacional para crear el modelo predictivo. Estos pozos fueron el 5599, 5351 y 7078; obteniendo de esta forma una precisión general de 0.92, valor que lo hace un modelo excelente para predecir producción de petróleo. Aunque, cabe recalcar que la presencia de valores atípicos, aunque sea mínima, puede arrojar falsos positivos dentro de las predicciones de este modelo, por lo que se recomienda realizar una validación cruzada con otros algoritmos más complejos para comprobar la efectividad del modelo predictivo de regresión polinomial. A continuación, en la figura 3.7, 3.8 y 3.9

presentamos la comparación que existe entre las curvas de predicción de petróleo y las actuales.

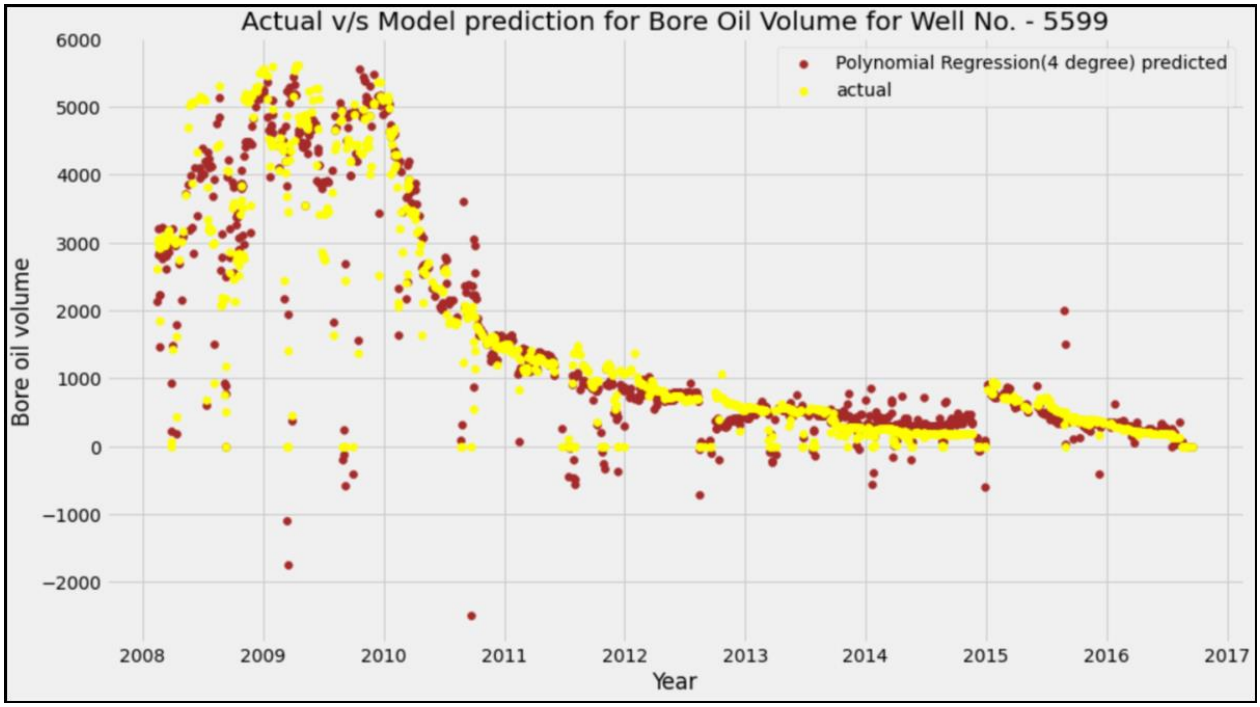


Figura 3.7 Comparación de la producción actual vs la producción predicha del pozo 5599 (fuente: Autores)

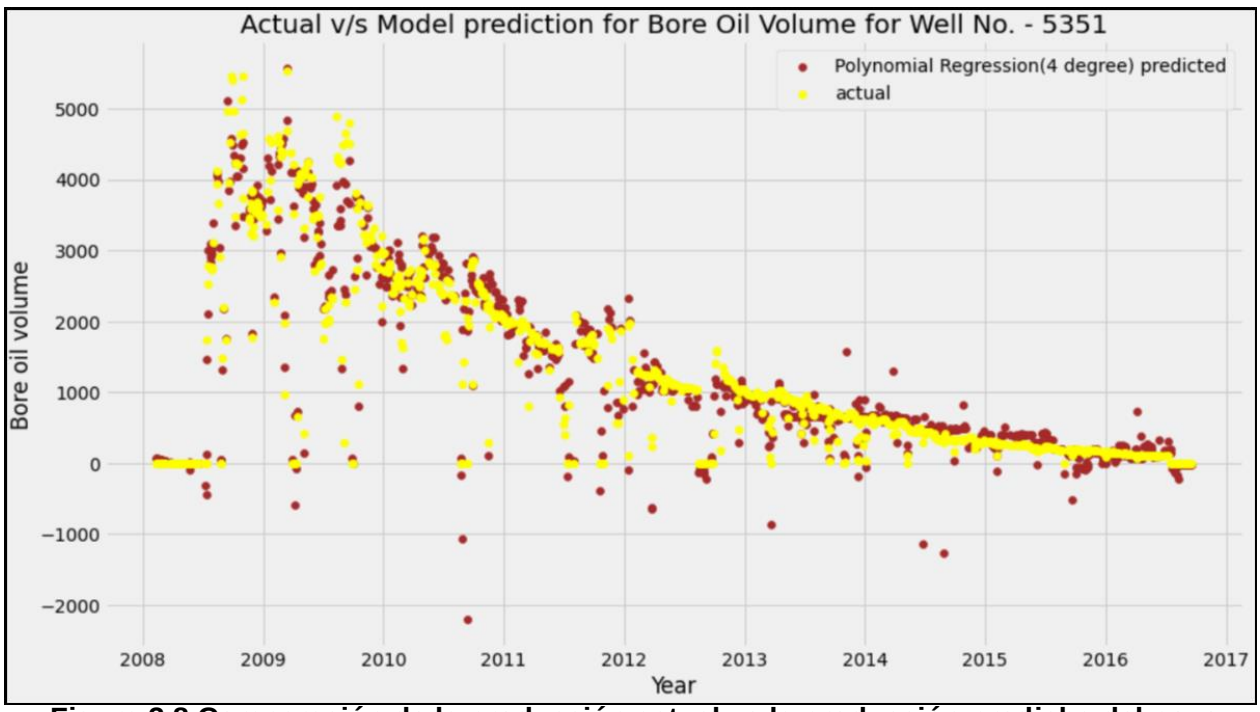


Figura 3.8 Comparación de la producción actual vs la producción predicha del pozo 5351 (fuente: Autores)

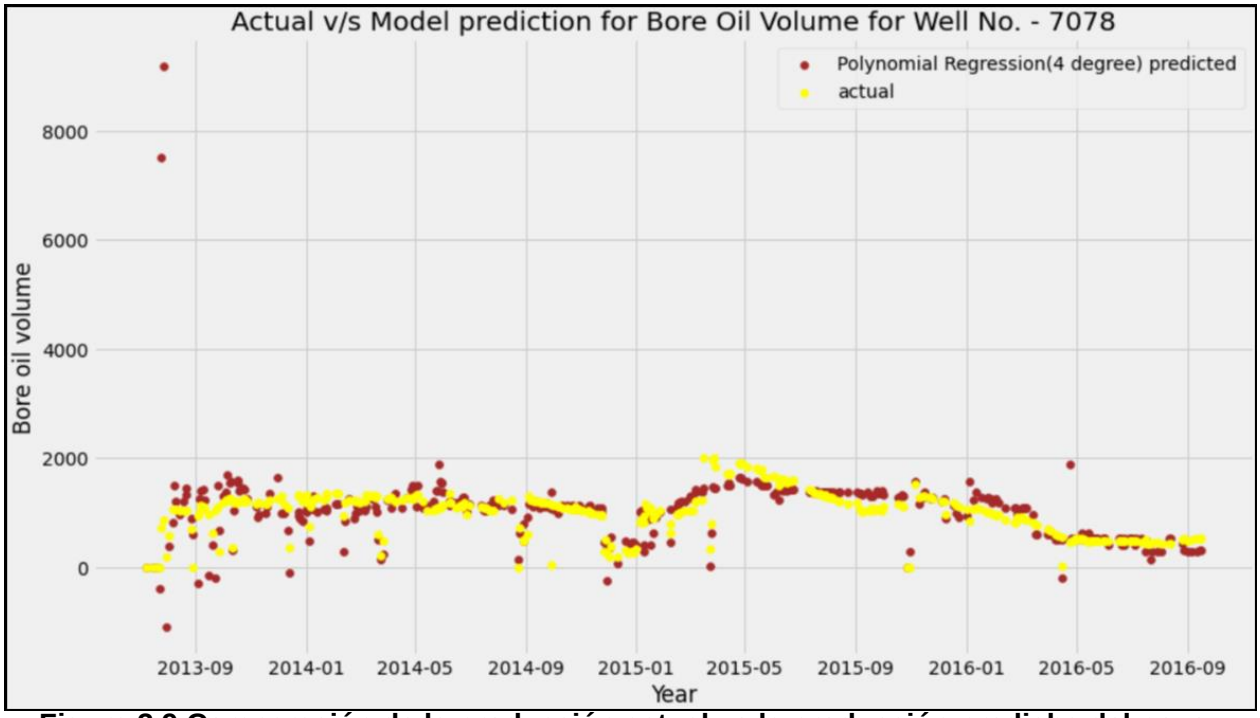


Figura 3.9 Comparación de la producción actual vs la producción predicha del pozo 7078 (fuente: Autores)

3.2.3 ARIMA

La aplicación del modelo ARIMA para predecir la producción de petróleo de los pozos del campo Volve presenta valores muy inusuales, y que debido a los conocimientos obtenidos comprendemos que este comportamiento de las tasas de producción es muy difícil que se dé.

Esto fue solamente probado al pozo 7405 y pozo 7078, pues debido al gran requerimiento de memoria del computador, de tiempo de cómputo y por los resultados no se insistió en probar el modelo al resto de los pozos productores.

En la figura 3.10 se puede visualizar un gráfico de las tasas de producción de petróleo que he generado el pozo 7405 desde el día 1 hasta el día 438, y se puede observar en los últimos periodos unos picos y descensos, y nuevamente un pico que no vuelve a ser en la misma proporción que su antecesor. Al aplicar el modelo ARIMA y pedirle la predicción de las tasas de producción de petróleo para los siguientes 365 días luego del día 438, se observa en la Figura 3.11 un incremento de la producción hasta los primeros días y luego esta tasa se mantiene constante en el tiempo.

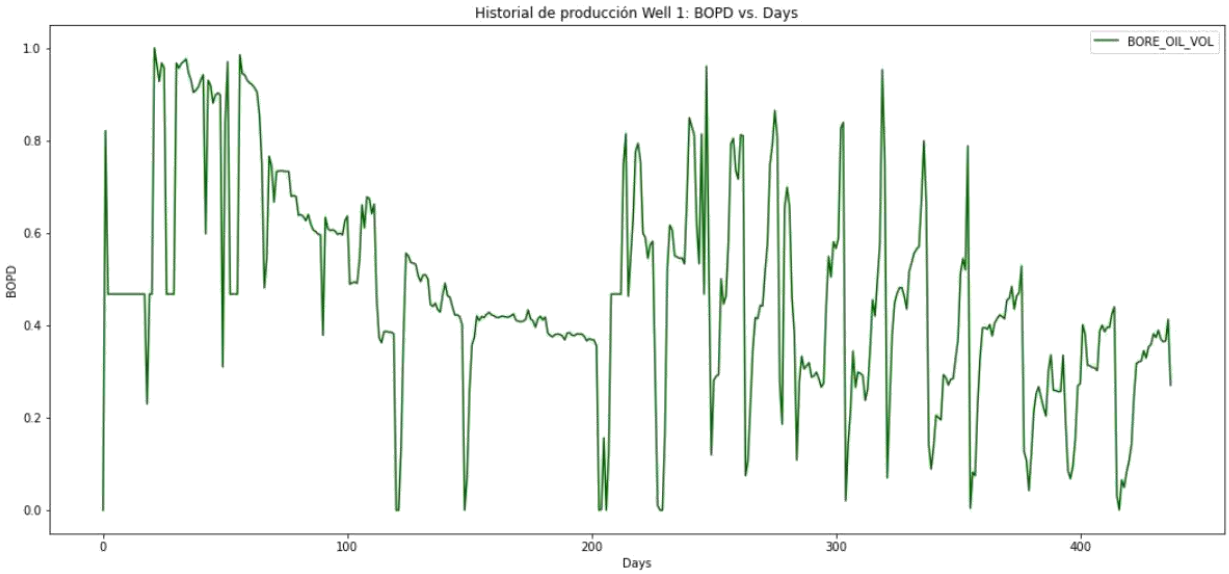


Figura 3.10 Historial de tasas de producción diaria de petróleo del pozo 7405 (fuente: Autores)

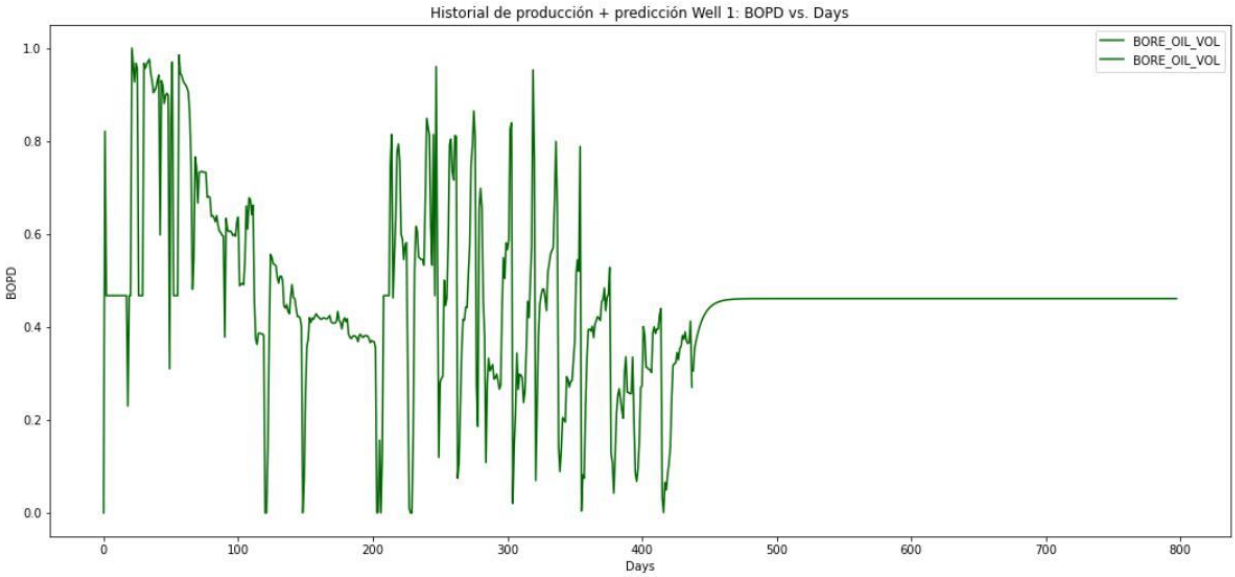


Figura 3.11 Historial de producción más las tasas de producción de petróleo pronóstico del pozo 7405 (fuente: Autores)

En el pozo 7078 ocurre una situación muy similar en la cual podemos ver primero el historial de tasas de producción de petróleo y ver sus tendencias a simple vista, la cual como en el pozo 7405 es decreciente mientras transcurre el tiempo. Ya en la Figura 3.12 se tiene la misma información de la figura anterior más las tasas de petróleo diaria pronóstico obtenidas por el algoritmo.

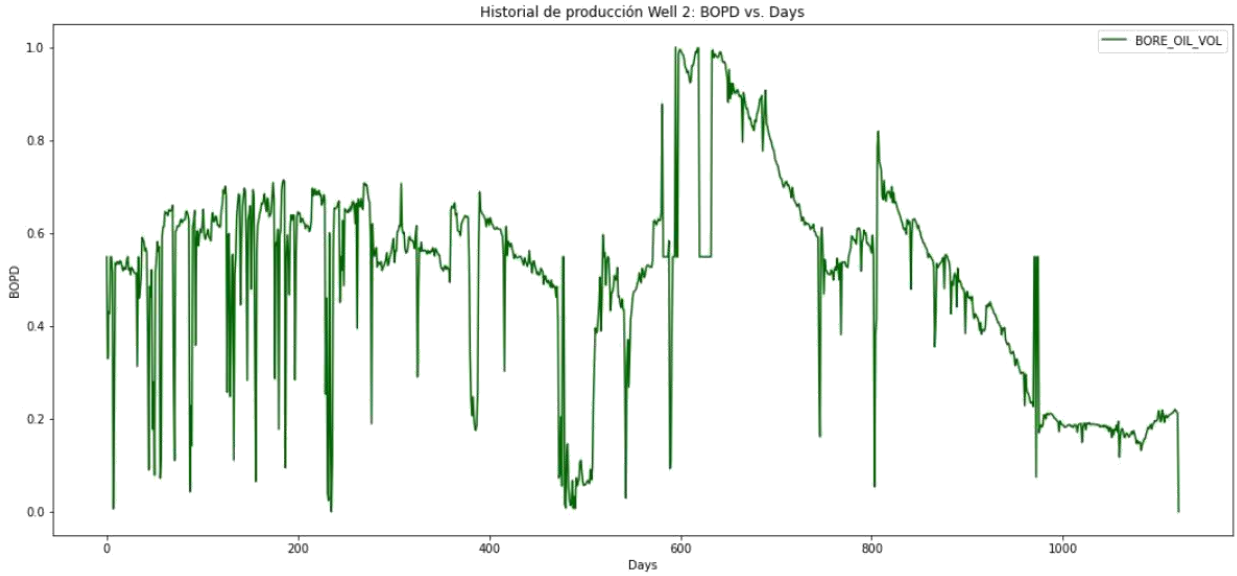


Figura 3.12 Historial de tasas de producción diaria de petróleo del pozo 7078 (fuente: Autores)

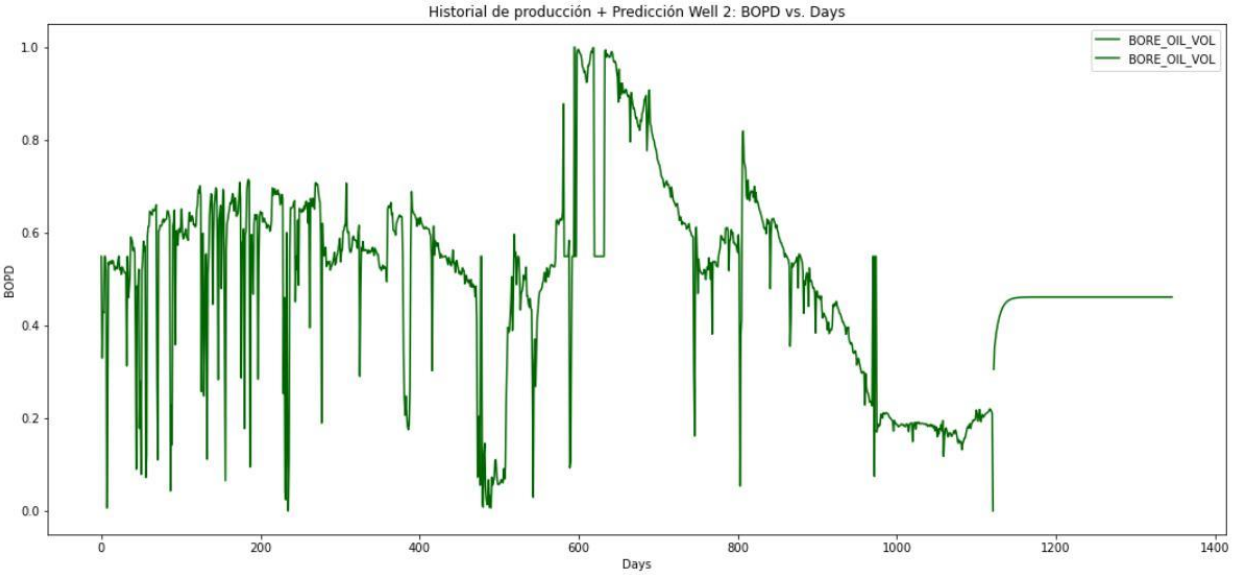


Figura 3.13 Historial de producción más las tasas de producción de petróleo pronóstico del pozo 7078 (fuente: Autores)

Un rápido aumento y luego un mantenimiento de la tasa de petróleo se puede visualizar, tal como pasa en el pozo 7405 ahora en los 225 días de pronóstico de producción en el pozo 7078. Esto es un comportamiento no real pues se conoce que deberíamos esperar una curva decreciente en el tiempo, mientras no se emplee algún método que nos ayude a obtener estas tasas, lo cual no podemos entender que el modelo ARIMA no es el más adecuado y por tanto se descarta en ese trabajo su uso.

3.2.4 Random Forest

El modelo de bosque aleatorio es un modelo muy práctico dentro de la academia e investigaciones en nuevas aplicaciones del Machine Learning, debido a que por lo general sus valores de predicciones son muy buenos pues se basa en el valor medio de las predicciones de cada uno de los árboles de decisión que lo conforman y que como sabemos cada uno se entrena con un subconjunto de datos y características del conjunto propio de entrenamiento que se separó para esta tarea.

En nuestro proyecto no fue la excepción el nivel de precisión presentado por el modelo al momento de darnos las predicciones y comparar con los valores del conjunto de prueba.

Se empleó el modelo para cada uno de los 5 pozos que fueron pozos productores desde el día 1, estos son los pozos 7405, 7078, 5351, 5599 y 7289. Para cada uno de los pozos se obtuvieron valores de precisión muy buenos y verificando que no existe un sobre ajuste de los datos hasta el pozo 5599, luego se puede interpretar los resultados del pozo 7289 como ligeramente sobre ajustado.

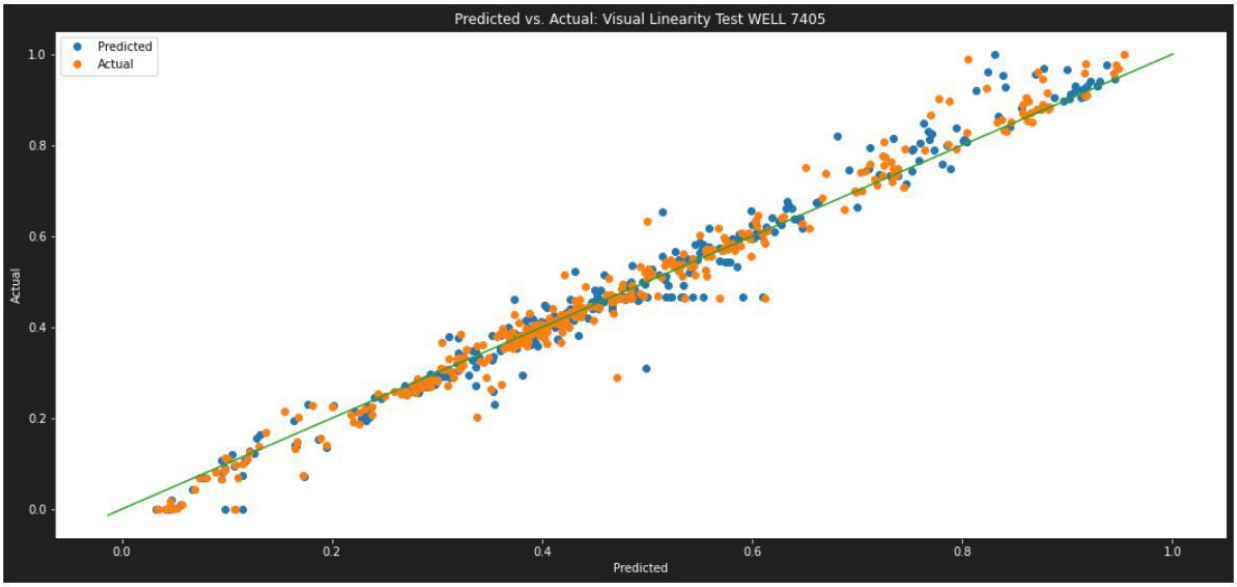


Figura 3.14 Gráfica de valores predichos vs valores actuales del pozo 7405 (fuente: Autores)

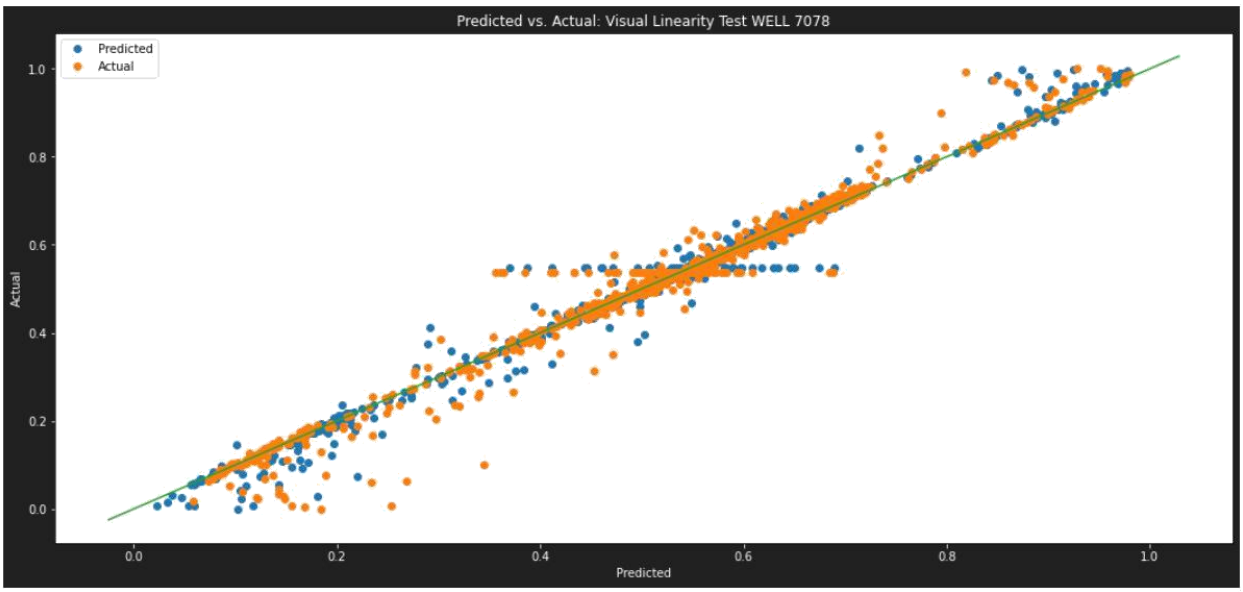


Figura 3.15 Gráfica de valores predichos vs valores actuales del pozo 7078 (fuente: Autores)

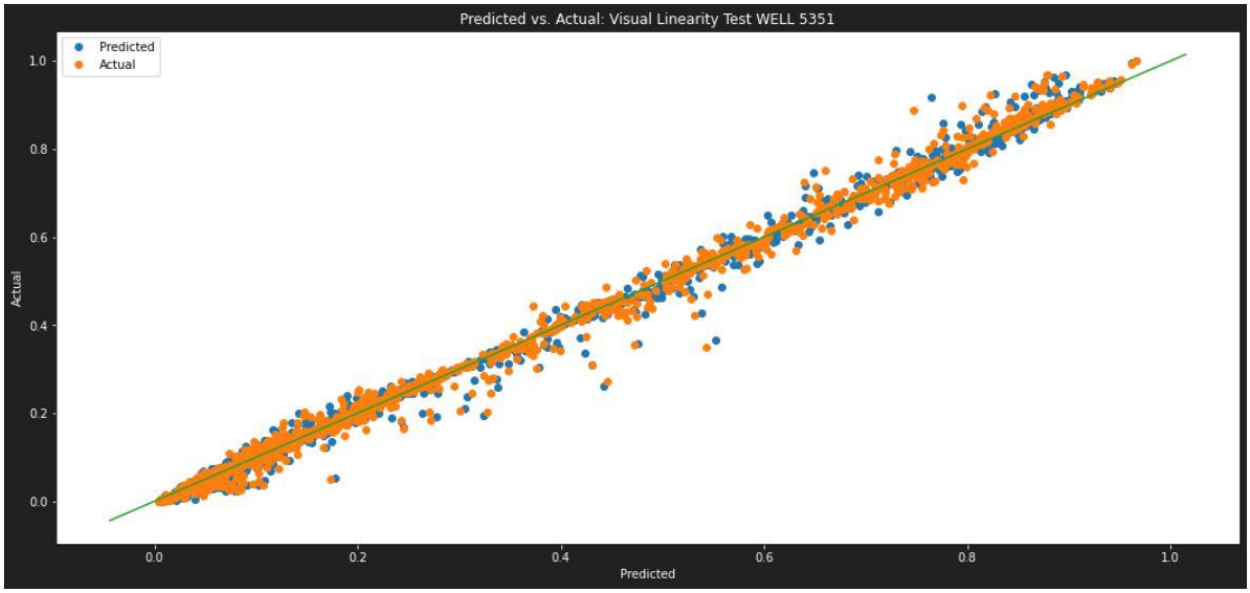


Figura 3.16 Gráfica de valores predichos vs valores actuales del pozo 5351 (fuente: Autores)

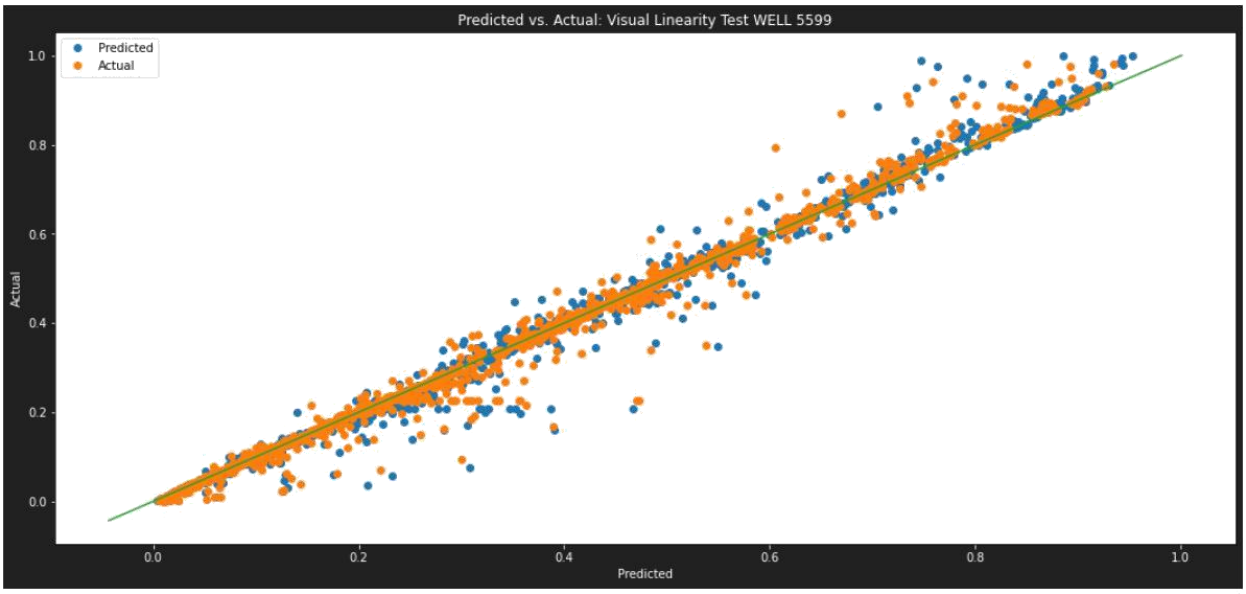


Figura 3.17 Gráfica de valores predichos vs valores actuales del pozo 5599 (fuente: Autores)

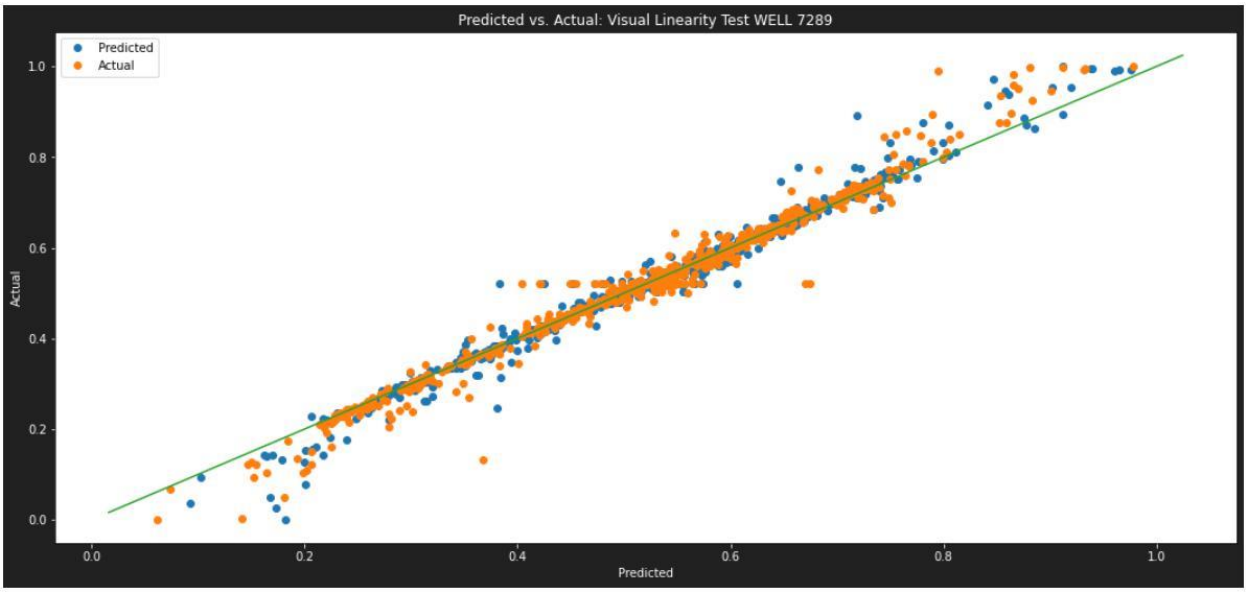


Figura 3.18 Gráfica de valores predichos vs valores actuales del pozo 7289 (fuente: Autores)

Para el pozo 7289 se obtuvieron los siguientes valores de precisión del modelo con la data de entrenamiento y uno considerablemente mayor de precisión con la data de prueba.

Tabla 3.1 Valores de R2 del pozo 7289 (fuente: Autores)

WELL 5	
R2 (con data de prueba)	0.69
R2(con data de entrenamiento)	0.96

Con esto podríamos creer que el modelo se encuentro sobre ajustado, es decir con la data de entrenamiento da mejor resultados de predicción, aun así, se ve en la gráfica una buena predicción y se lo confirma con el puntaje del modelo ya entrenado que es de 0.72.

3.3 Aplicativo Web

La interfaz del aplicativo Saviour, presenta ciertos detalles y fuentes de información que nos ayudan a comprender brevemente la aplicabilidad e importancia que tiene el machine learning, data science y más dentro de la industria petrolera. Además, cuenta con un mensaje indicativo de los requisitos que debe cumplir la data a emplear, para un correcto funcionamiento y resultados.

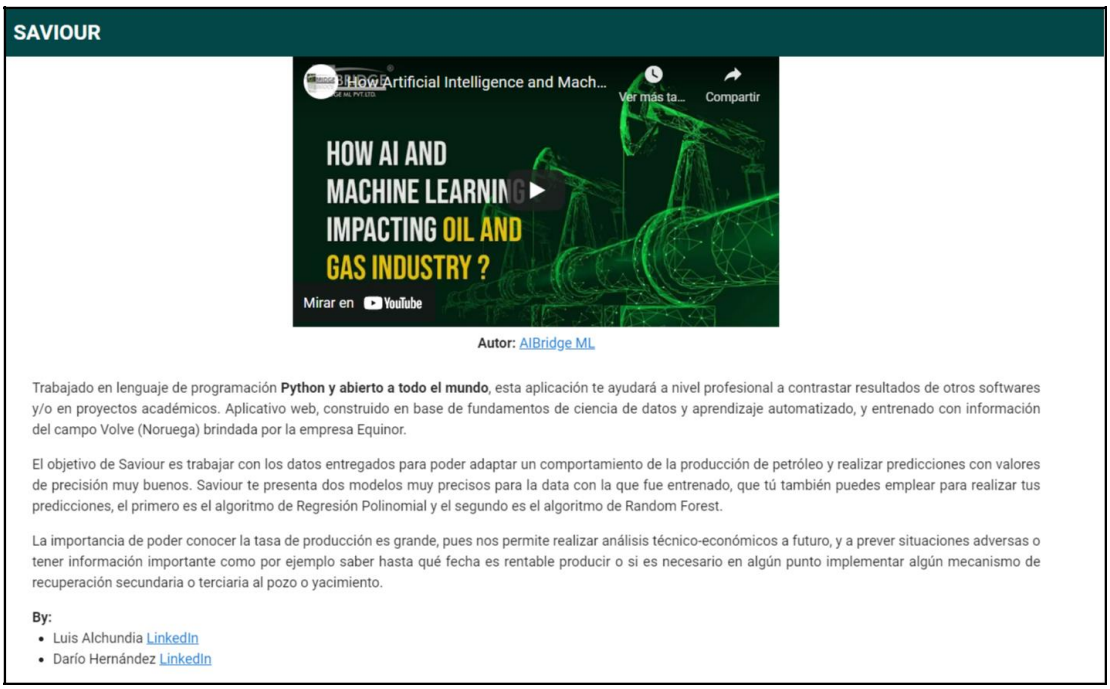


Figura 3.19 Sección informativa de la interfaz del aplicativo web (fuente: Autores)

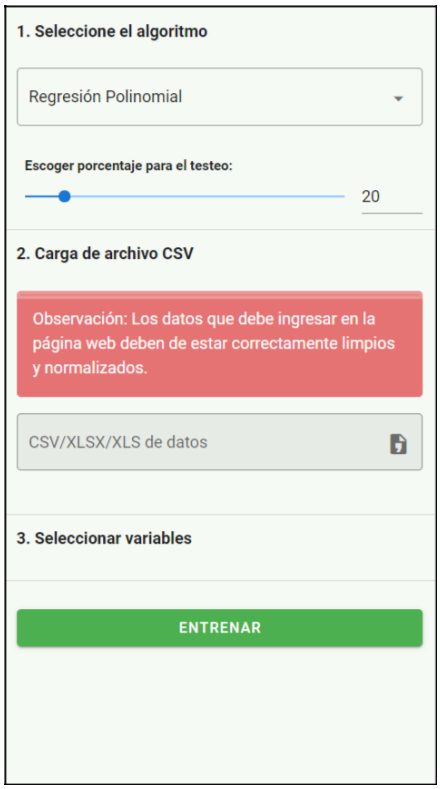


Figura 3.20 Sección de maniobrabilidad del aplicativo web (fuente: Autores)

Así como se observa en la Figura 3.20, se tiene un campo para seleccionar que algoritmo emplear de entre los dos elegidos por sus buenos valores de precisión de predicción anteriormente, es seguido por la opción de delimitar el porcentaje de la data a ingresar que será empleada en el aplicativo; posterior se puede cargar el archivo con la data, y entonces nos aparecerá como se ve en la Figura 3.21 nuevos campos a definir, donde el campo fecha requiere seleccionemos la columna con las fechas de producción, el campo Y requiere definamos el nombre con el que se encuentran los volúmenes de producción de petróleo o gas del pozo en una unidad de tiempo y el campo de agrupación pide seleccionar el nombre de la columna en la que se encuentran los nombres de los distintos pozos a analizar dentro del mismo archivo.

Figura 3.21 Sección del aplicativo web de selección de variables (fuente: Autores)

Puesto en marcha el aplicativo web, presenta de manera limpia y clara resultados de la curva de producción de petróleo real junto con la predicha por el modelo seleccionado.

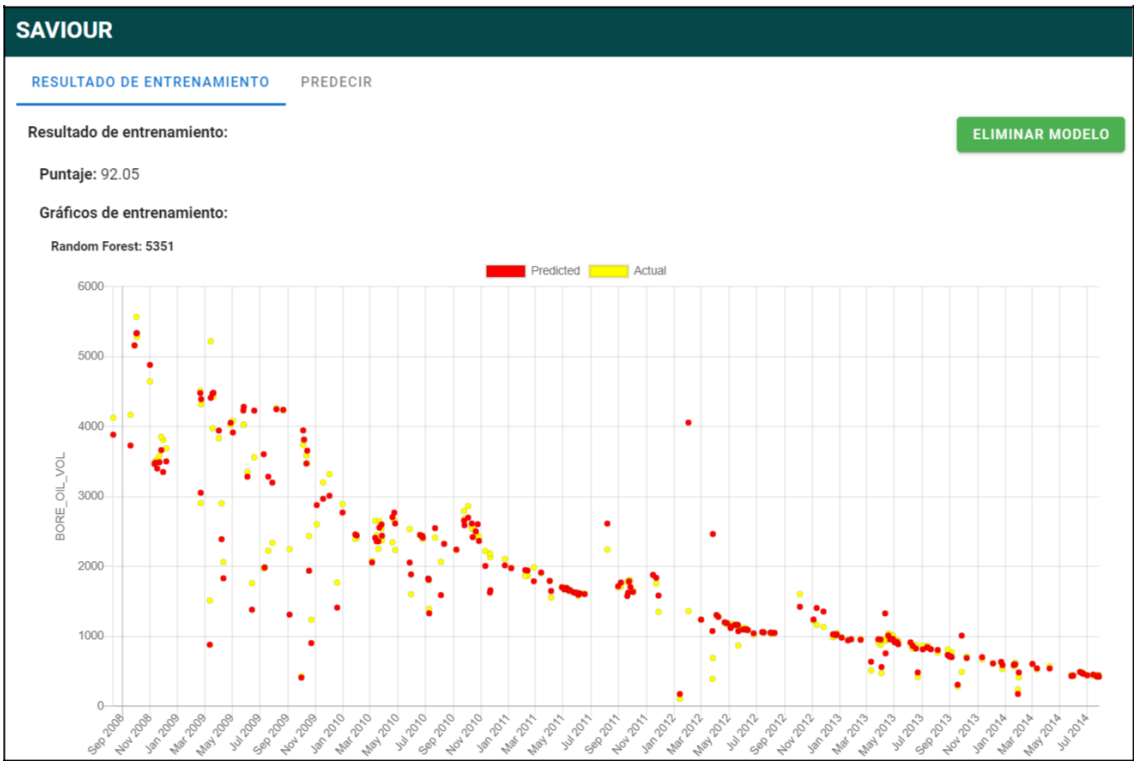


Figura 3.22 Vista de los resultados presentados del entrenamiento mediante la interfaz de Saviour. (fuente: Autores)

Como se ve en la figura anterior, el modelo presenta un puntaje de 95.86 de un 100, es decir el modelo entrenado presenta un muy buen entendimiento del comportamiento de la información.

Se verificará esta precisión observando 5 puntos al azar de los que presenta la curva real (en color amarillo) en comparación con los resultados presentados por el modelo (en color rojo).

Tabla 3.2 Puntaje de precisión de 5 puntos tomado al azar del modelo entrenado
(fuente: Autores)

Valores actuales Oil volume ()	Valores predichos Oil volumen ()	% Error	Precisión
5281.92	5337.09	1.04	98.96
2891.08	2770.89	4.16	95.84
1701.47	1717.47	0.94	99.06
1163.63	1405.37	20.77	79.23
573.63	540.52	5.77	94.23

Se corrobora entonces que los puntos tomados al azar dentro de la curva presentada en la interfaz de Saviour presentan la mayoría valores muy buenos de precisión y otros no son tan alejados de los valores reales, y lo podemos verificar gráficamente y por medio del puntaje general del modelo entrenado.

La interfaz del aplicativo web, también nos presenta de manera gráfica y con la opción de conocer aquellos valores al colocar el puntero del mouse sobre los puntos de la curva de predicción futura de las tasas de producción de petróleo o gas.

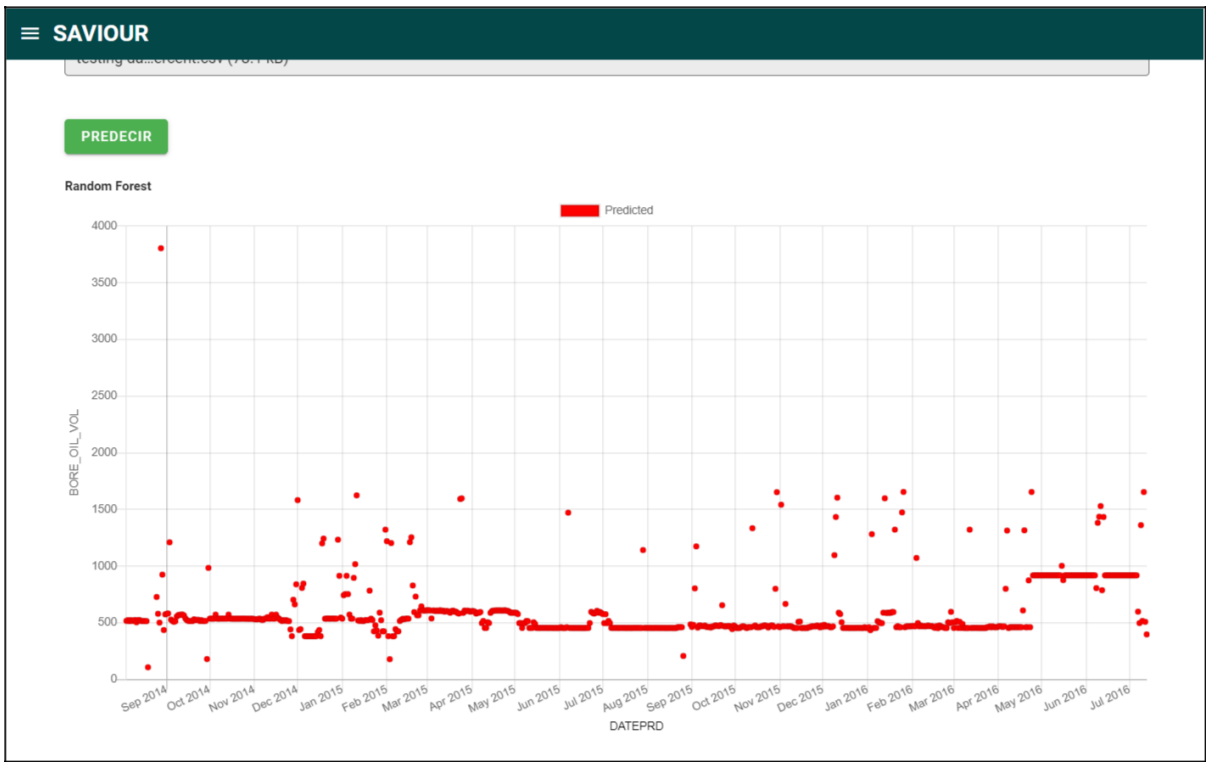


Figura 3.23 Curva resultado de la predicción de las tasas de producción de petróleo. (fuente: Autores)

Se puede observar en la Figura 3.23 con detalle el rango de valores de tasas de producción diaria de petróleo que el aplicativo web entrega, cuyos resultados se encuentran entre los 550 a 450 barriles aproximadamente; dicho rango se encuentra cercano al rango de los valores reales registrados para el mismo lapso, los cuales pueden variar por algunos factores externos que pueden ser modificados completamente. También se puede observar en la curva de predicción valores fuera del rango comprendido, debido al comportamiento de las otras variables que el modelo toma a consideración y de la eficiencia propia de un modelo predictivo que no presentará valores totalmente acertados.

3.4 Contraste de modelos físicos vs modelos de Machine Learning

Los software o modelos físicos que se emplean actualmente en la industria del petróleo tienen grandes ventajas y desventajas por sobre nuestro programa “Saviour” que hacen que la decisión de elección de uno de estos dependa del usuario. Vale indicar que no se busca desmeritar ni promocionar ninguno de los

softwares que se mencionan más adelante, más se plantea un contraste de características de funcionalidad y resultados con “Saviour”.

Se realiza un contraste con el software de Citrine de Kappa, el cuál emplea una metodología enfocada en las curvas de declinación para yacimientos convencionales mientras que para los no convencionales emplea un patrón común de análisis de rendimiento en base a un solo pozo. Sus valores de predicción son muy buenos pues requieren de una gran cantidad de datos geológicos, análisis PVT e información de operación (*KAPPA - Citrine Overview*, n.d.); y es capaz de presentar más información de importancia analítica que Saviour. Nuestro aplicativo web requiere en cierta medida de menor cantidad de características, más si requiere un gran volumen de registros para analizarlos y realizar las predicciones con mayor precisión, la cual a criterio de ingenieros presenta un matching entre valores de predicción y valores reales muy bueno, superando a cualquier software actual capaz de predecir producción de petróleo en la industria del petróleo y gas.

Otro software que hemos analizado a partir de la información brindada por la página web oficial para contrastar con Saviour, es CoFlow de CMG; el software posee características muy interesantes y que sirve para varios campos de ingeniería, como la de reservorios y de producción.

CoFlow trabaja con varios niveles de fidelidad, entre ellas resaltamos el nivel de fidelidad bajo que presenta empleando análisis de curvas de declinación pues aquí es donde el software presenta la capacidad de realizar una optimización integrada y parametrización del diseño en busca de estrategias de operación y finalización en busca de los niveles de flujo de producción óptimos, el NPV y recuperación final. Esto nos permite entender que Saviour es limitado a dar resultados a raíz de los datos ya conocidos y que se ejecutan en las operaciones, más no nos permite modificarlos para buscar las condiciones que nos permitan mejorar las tasas de producción como lo hace CoFlow. (*Software CMG | CoFlow*, n.d.)

Algo a destacar que posee Saviour frente a estos softwares y cualquier otro, es su costo de licencia, pues hay que mencionar que las licencias de uso de estos otros softwares tienen un valor económico que suele ser elevado; nuestro

aplicativo web sin embargo al ser un programa de código abierto es de acceso libre y gratuito.

CAPÍTULO 4

4. CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- La correcta reducción de dimensionalidad y normalización del marco de datos original nos permitió visualizar una base de datos útil que simplificó el modelado de los datos, redujo significativamente el tiempo de cálculo en cada uno de los modelos creados y finalmente nos permitió visualizar que tan preciso fue un modelo al entrenarse con estos datos.
- El modelo predictivo creado en base al algoritmo de regresión lineal tuvo una precisión general de 0.65, valor moderadamente bajo que automáticamente lo descalifica como un buen modelo de predicción. Teóricamente, esto ocurre porque cuando hablamos de declinación de producción de petróleo, es muy difícil, por no decir imposible, que una recta describa la tendencia de declinación de producción de un pozo.
- El modelo predictivo creado en base a una regresión polinomial de grado 4, marcó una mejor precisión en comparación con el modelo anterior. La precisión del modelo fue de 0.92, calificándolo como un modelo muy bueno para este tipo de predicciones. Su alta precisión se debe a que al aumentar los grados del polinomio se ajusta de mejor forma a la tendencia que suele tener una curva de declinación de producción de petróleo.
- El modelo predictivo basado en el algoritmo ARIMA presentó valores de precisión incongruentes con respecto al tiempo y a las consideraciones técnicas que se tiene en referencia a los pozos. Descartando desafortunadamente al modelo para su implementación dentro de la página web. (justificar el porque el modelo realiza estas malas predicciones)
- El modelo predictivo creado usando el algoritmo Random Forest presentó precisiones generales muy buenas que varían en un rango de 0.80 a 0.96, además de que el modelo es completo en comparación al resto, no presenta un sobre ajuste. Este modelo fue seleccionado como el segundo modelo a aplicar dentro de la página web, pero es mucho más confiable que el modelo basado en regresiones polinomiales.

- Las predicciones realizadas por el aplicativo web Saviour tienen un porcentaje de error menor cuando se utiliza el algoritmo de Random Forest como base en comparación con el de regresión polinomial cuyos errores son más visibles. Esto sucede porque el marco de datos que se está ingresando en el aplicativo web para entrenamiento, contiene quizá un porcentaje alto de valores atípicos que influyen en el resultado de estas predicciones.
- La implementación arbitraria de la selección de la cantidad de árboles de decisión en el algoritmo Random Forest produjo una extrema lentitud dentro del aplicativo web, por lo que actualmente Saviour funciona con una cantidad específica de árboles de decisión, haciendo que trabaje normalmente y las predicciones sean lo más precisas posibles.
- La funcionalidad de Saviour es única, caracterizado como un aplicativo web de predicción de producción de petróleo donde su mayor virtud es el ahorro económico mediante la aplicación de ciencia de datos y Machine Learning. Los otros tipos de softwares tienen virtudes que incluyen hasta simulaciones de yacimientos, pero desafortunadamente a diferencia de Saviour no son de libre acceso y la obtención de una licencia puede llegar a costar hasta miles de dólares.
- Saviour tiene una tendencia a servir más a la comunidad estudiantil, específicamente para el área de producción. Pero dada su condición de libre acceso, queda como una segunda o tercera herramienta para corroborar predicciones de producción de petróleo dentro de las propias empresas petroleras que invierten demasiado dinero en la implementación de sus softwares.

4.2 Recomendaciones

- La base de datos normalizada es mucho más práctica tenerla albergada dentro del mismo ordenador, ya que es más sencillo realizar la conexión con la interfaz Jupyter Lab para su posterior manipulación. La otra alternativa puede ser la utilización de una plataforma que también permita albergar la base de datos. Sin embargo, la desventaja es que dependiendo de la plataforma que se quiera utilizar, obligatoriamente se debe tener conocimiento de las formas de conexión entre dichas plataformas y la interfaz Jupyter Lab.

- Una revisión bibliográfica previo al procesamiento de los datos es necesaria para la correcta selección de los algoritmos a utilizar. Esto se debe a que no existe una regla de oro para la aplicación de estos algoritmos. Realizar una revisión bibliográfica ayuda a llevar un procesamiento más ordenado de los datos, el cuál empieza con un análisis exploratorio y culmina con la correcta normalización del marco de datos final.
- Para la aplicación de los algoritmos de regresión lineal y polinomial, es necesario que en el procesamiento de los datos se realice una limpieza adecuada de los valores atípicos que presenten las variables de ingreso. Estos algoritmos son muy susceptibles a este tipo de valores, por lo que existe una alta probabilidad de que los modelos arrojen falsos valores de precisión.
- Durante la ejecución del modelo de Random Forest en el pozo 5, se puede estimar que el modelo presenta un ligero sobre ajuste debido a la cantidad de datos que procesa, por lo que es recomendable realizar un incremento de la cantidad de datos empleados y un tuneo de hiperparámetros que evite problemas de este tipo.
- Saviour no realiza procesos de análisis exploratorio de datos, ni mucho menos limpieza, ordenamiento y normalización de estos. Por lo que se sugiere realizar dichos procesos previo al entrenamiento del modelo y a las predicciones deseadas.

BIBLIOGRAFÍA

¿Para qué sirve Python? Razones para utilizarlo | ESIC. (n.d.). Retrieved May 21, 2021, from https://www.esic.edu/rethink/tecnologia/para-que-sirve-python

¿Qué es la ciencia de datos? | Oracle México. (n.d.). Retrieved May 21, 2021, from https://www.oracle.com/mx/data-science/what-is-data-science/

Ali, A. (2021). Data-driven based machine learning models for predicting the deliverability of underground natural gas storage in salt caverns. *Energy*, *229*, 120648. https://doi.org/10.1016/j.energy.2021.120648

Antonio, D., Sadlier, A., & Winston, J. (2019). Leveraging oil and gas data lakes to enable data science factories. *Society of Petroleum Engineers - SPE/IATMI Asia Pacific Oil and Gas Conference and Exhibition 2019, APOG 2019*, *October*, 29–31. https://doi.org/10.2118/196452-ms

Asfoor, H. (2020). Applying data science techniques to improve information discovery in oil and gas unstructured data. *International Petroleum Technology Conference 2020, IPTC 2020*. https://doi.org/10.2523/iptc-20236-abstract

Fan, D., Sun, H., Yao, J., Zhang, K., Yan, X., & Sun, Z. (2021). Well production forecasting based on ARIMA-LSTM model considering manual operations. *Energy*, *220*, 119708. https://doi.org/10.1016/j.energy.2020.119708

J. Spath, S. P. E. P. (2015). *Big Data !*

KAPPA - Citrine overview. (n.d.). Retrieved August 8, 2021, from https://www.kappaeng.com/software/citrine/overview

Kumar, V., & Sahu, M. (2021). Evaluation of nine machine learning regression algorithms for calibration of low-cost PM2.5 sensor. *Journal of Aerosol Science*, *157*, 105809. https://doi.org/10.1016/J.JAEROSCI.2021.105809

Liao, L., Zeng, Y., Liang, Y., & Zhang, H. (2020). Data mining: A novel strategy for production forecast in tight hydrocarbon resource in Canada by random forest analysis. *International Petroleum Technology Conference 2020, IPTC 2020*.

https://doi.org/10.2523/iptc-20344-ms

Mohamed, M. I., Mehta, D., Salah, M., Ibrahim, M., & Ozkan, E. (2020). *Advanced Machine Learning Methods for Prediction of Fracture Closure Pressure, ClosureTime, Permeability and Time to Late Flow Regimes From DFIT*. https://doi.org/10.15530/urtec-2020-2762

Mohammadpoor, M., & Torabi, F. (2020). Big Data analytics in oil and gas industry: An emerging trend. *Petroleum*, *6*(4), 321–328. https://doi.org/10.1016/j.petlm.2018.11.001

Peter, M., Aldo, D. A., Cheng, F., & Ong, S. (n.d.-a). *MATHEMATICS FOR MACHINE LEARNING*.

Peter, M., Aldo, D. A., Cheng, F., & Ong, S. (n.d.-b). *MATHEMATICS FOR MACHINE LEARNING*. Retrieved May 27, 2021, from https://mml-book.com.

Polynomial Regression | Uses and Features of Polynomial Regression. (n.d.).

Random Forest Algorithms: A Complete Guide | Built In. (n.d.). Retrieved July 31, 2021, from https://builtin.com/data-science/random-forest-algorithm

Sami, N. A., & Ibrahim, D. S. (2021). Forecasting multiphase flowing bottom-hole pressure of vertical oil wells using three machine learning techniques. *Petroleum Research*, *xxxx*. https://doi.org/10.1016/j.ptlrs.2021.05.004

Seemann D., Williamson M., H. S. (2013). Improving reservoir management through big data technologies. *SPE Middle East Intelligent Energy Conference and Exhibition*, 1–11.

Software CMG | CoFlow. (n.d.). Retrieved August 8, 2021, from https://www.cmgl.ca/coflow

Volve - equinor.com. (n.d.). Retrieved May 21, 2021, from https://www.equinor.com/en/what-we-do/norwegian-continental-shelf-platforms/volve.html

APÉNDICE

Análisis exploratorio de Datos (EDA)

Link al código:

https://github.com/saviourapi/Saviour/blob/ebf3686711aec22d6b34af4b5f39e866097206fd/notebook/VolveFieldGeneralEDA.ipynb

```
#Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
!pip install xlrd==1.2.0
!pip install -U scikit-learn
!pip install XlsxWriter


#Import the xls production file using pandas
prod_df=pd.read_excel('Daily Production.xls')


#Knowing the data in general
print(prod_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15634 entries, 0 to 15633
Data columns (total 24 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   DATEPRD                    15634 non-null  datetime64[ns]
 1   WELL_BORE_CODE             15634 non-null  object
 2   NPD_WELL_BORE_CODE         15634 non-null  int64
 3   NPD_WELL_BORE_NAME         15634 non-null  object
 4   NPD_FIELD_CODE             15634 non-null  int64
 5   NPD_FIELD_NAME             15634 non-null  object
 6   NPD_FACILITY_CODE          15634 non-null  int64
 7   NPD_FACILITY_NAME          15634 non-null  object
 8   ON_STREAM_HRS              15349 non-null  float64
 9   AVG_DOWNHOLE_PRESSURE      8980 non-null   float64
 10  AVG_DOWNHOLE_TEMPERATURE   8980 non-null   float64
 11  AVG_DP_TUBING              8980 non-null   float64
 12  AVG_ANNULUS_PRESS          7890 non-null   float64
 13  AVG_CHOKE_SIZE_P           8919 non-null   float64
 14  AVG_CHOKE_UOM              9161 non-null   object
 15  AVG_WHP_P                  9155 non-null   float64
 16  AVG_WHT_P                  9146 non-null   float64
 17  DP_CHOKE_SIZE              15340 non-null  float64
 18  BORE_OIL_VOL               9161 non-null   float64
 19  BORE_GAS_VOL               9161 non-null   float64
 20  BORE_WAT_VOL               9161 non-null   float64
 21  BORE_WI_VOL                5706 non-null   float64
 22  FLOW_KIND                  15634 non-null  object
 23  WELL_TYPE                  15634 non-null  object
dtypes: datetime64[ns](1), float64(13), int64(3), object(7)
memory usage: 2.9+ MB
None
```

```
prod_df
```

	DATEPRD	WELL_BORE_CODE	NPD_WELL_BORE_CODE	NPD_WELL_BORE_NAME	NPD_FIELD_CODE	NPD_FIELD_NAME	NPD_FACILITY_CODE	NPD_FACILITY_NAME	ON_STREAM_HRS	AVG_DOWNHOLE_PRESSURE	...	AVG_CHOKE_UOM	AVG_WHP_P
0	2014-04-07	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	0.00000	...	%	0.0000
1	2014-04-08	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	NaN	...	%	0.0000
2	2014-04-09	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	NaN	...	%	0.0000
3	2014-04-10	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	NaN	...	%	0.0000
4	2014-04-11	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	310.37614	...	%	33.0978
...	...	...	...	...	...	...	...	...	...	...	...	...	...
15629	2016-09-14	NO 15/9-F-5 AH	5769	15/9-F-5	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	NaN	...	%	0.0777
15630	2016-09-15	NO 15/9-F-5 AH	5769	15/9-F-5	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	NaN	...	%	0.0854
15631	2016-09-16	NO 15/9-F-5 AH	5769	15/9-F-5	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	NaN	...	%	0.0854
15632	2016-09-17	NO 15/9-F-5 AH	5769	15/9-F-5	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	NaN	...	%	0.0749
15633	2016-09-18	NO 15/9-F-5 AH	5769	15/9-F-5	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	NaN	...	NaN	NaN

15634 rows × 24 columns

```
# Dispersion measures per
variable prod_df.describe()
```

	NPD_WELL_BORE_CODE	NPD_FIELD_CODE	NPD_FACILITY_CODE	ON_STREAM_HRS	AVG_DOWNHOLE_PRESSURE	AVG_DOWNHOLE_TEMPERATURE	AVG_DP_TUBING	AVG_ANNULUS_PRESS	AVG_CHOKE_SIZE_P	AVG_WHP_P	AVG_WHT_P	DP_CHOK
count	15634.000000	15634.0	15634.0	15349.000000	8980.000000	8980.000000	8980.000000	7890.000000	8919.000000	9155.000000	9146.000000	15340
mean	5908.581745	3420717.0	369304.0	19.994093	181.803869	77.162969	154.028787	14.856100	55.168533	45.377811	67.728440	11.
std	649.231622	0.0	0.0	8.369978	109.712363	45.657948	76.752373	8.406822	36.692924	24.752631	27.719028	19.
min	5351.000000	3420717.0	369304.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	5599.000000	3420717.0	369304.0	24.000000	0.000000	0.000000	83.665361	10.841437	18.952989	31.148062	56.577834	0.
50%	5693.000000	3420717.0	369304.0	24.000000	232.896939	103.186689	175.588861	16.308598	52.096877	37.933620	80.071250	2.
75%	5769.000000	3420717.0	369304.0	24.000000	255.401455	106.276591	204.319964	21.306125	99.924288	57.101268	88.062202	13.
max	7405.000000	3420717.0	369304.0	25.000000	397.588550	108.502178	345.906770	30.019828	100.000000	137.311030	93.509584	125.

```
#Correlation analysis between
variables prod_df.corr()
```

	NPD_WELL_BORE_CODE	NPD_FIELD_CODE	NPD_FACILITY_CODE	ON_STREAM_HRS	AVG_DOWNHOLE_PRESSURE	AVG_DOWNHOLE_TEMPERATURE	AVG_DP_TUBING	AVG_ANNULUS_PRESS	AVG_CHOKE_SIZE_P	AVG_WHP_
NPD_WELL_BORE_CODE	1.000000	NaN	NaN	-0.102270	0.257481	0.339509	0.218243	0.141756	-0.558461	0.0779
NPD_FIELD_CODE	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
NPD_FACILITY_CODE	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
ON_STREAM_HRS	-0.102270	NaN	NaN	1.000000	-0.003092	0.105931	0.002824	0.482779	0.531331	-0.0449
AVG_DOWNHOLE_PRESSURE	0.257481	NaN	NaN	-0.003092	1.000000	0.967826	0.949734	-0.124023	-0.262804	0.2833
AVG_DOWNHOLE_TEMPERATURE	0.339509	NaN	NaN	0.105931	0.967826	1.000000	0.898954	-0.087189	-0.295764	0.2742
AVG_DP_TUBING	0.218243	NaN	NaN	0.002824	0.949734	0.898954	1.000000	-0.110382	-0.162100	0.1024
AVG_ANNULUS_PRESS	0.141756	NaN	NaN	0.482779	-0.124023	-0.087189	-0.110382	1.000000	0.155142	-0.0032
AVG_CHOKE_SIZE_P	-0.558461	NaN	NaN	0.531331	-0.262804	-0.295764	-0.162100	0.155142	1.000000	-0.3986
AVG_WHP_P	0.077946	NaN	NaN	-0.044900	0.283359	0.274247	0.102476	-0.003255	-0.398607	1.0000
AVG_WHT_P	-0.519515	NaN	NaN	0.763229	-0.095114	-0.076801	-0.054496	0.316138	0.752825	-0.0413
DP_CHOKE_SIZE	0.237647	NaN	NaN	-0.229295	0.267649	0.221986	0.093282	-0.178271	-0.551193	0.9086
BORE_OIL_VOL	-0.307645	NaN	NaN	0.342031	0.248571	0.289819	0.126964	-0.025095	0.029671	0.4304
BORE_GAS_VOL	-0.310793	NaN	NaN	0.353713	0.245981	0.287328	0.125475	-0.028252	0.039699	0.4285
BORE_WAT_VOL	-0.493591	NaN	NaN	0.405123	-0.296765	-0.343581	-0.171882	0.094019	0.760294	-0.3288
BORE_WI_VOL	-0.055894	NaN	NaN	0.749544	NaN	NaN	NaN	NaN	NaN	Na

```
#Printing the NPD NAME and CODE per well
print(prod_df['NPD_WELL_BORE_NAME'].value_counts())
print(prod_df['NPD_WELL_BORE_CODE'].value_counts())
```

```
15/9-F-4       3327
15/9-F-5       3306
15/9-F-14      3056
15/9-F-12      3056
15/9-F-11      1165
15/9-F-15 D     978
15/9-F-1 C      746
Name: NPD_WELL_BORE_NAME, dtype: int64
5693    3327
5769    3306
5351    3056
5599    3056
7078    1165
7289     978
7405     746
Name: NPD_WELL_BORE_CODE, dtype: int64
```

```
#Extracting information per well using the NPD
code well1=prod_df['NPD_WELL_BORE_CODE']==5693
well2=prod_df['NPD_WELL_BORE_CODE']==5769
well3=prod_df['NPD_WELL_BORE_CODE']==5351
well4=prod_df['NPD_WELL_BORE_CODE']==5599
well5=prod_df['NPD_WELL_BORE_CODE']==7078
well6=prod_df['NPD_WELL_BORE_CODE']==7289
well7=prod_df['NPD_WELL_BORE_CODE']==7405
prod_df_well1=prod_df[well1]
```

```
prod_df_well2=prod_df[well2]
prod_df_well3=prod_df[well3]
prod_df_well4=prod_df[well4]
prod_df_well5=prod_df[well5]
prod_df_well6=prod_df[well6]
prod_df_well7=prod_df[well7]
print(prod_df_well1.info(), prod_df_well2.info(),
prod_df_well3.info(), prod_df_well4.info(), prod_df_well5.info(),
prod_df_well6.info(), prod_df_well7.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3327 entries, 9001 to 12327
Data columns (total 24 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   DATEPRD                   3327 non-null   datetime64[ns]
 1   WELL_BORE_CODE            3327 non-null   object
 2   NPD_WELL_BORE_CODE        3327 non-null   int64
 3   NPD_WELL_BORE_NAME        3327 non-null   object
 4   NPD_FIELD_CODE            3327 non-null   int64
 5   NPD_FIELD_NAME            3327 non-null   object
 6   NPD_FACILITY_CODE         3327 non-null   int64
 7   NPD_FACILITY_NAME         3327 non-null   object
 8   ON_STREAM_HRS             3175 non-null   float64
 9   AVG_DOWNHOLE_PRESSURE     0 non-null      float64
 10  AVG_DOWNHOLE_TEMPERATURE  0 non-null      float64
 11  AVG_DP_TUBING             0 non-null      float64
 12  AVG_ANNULUS_PRESS         0 non-null      float64
 13  AVG_CHOKE_SIZE_P          0 non-null      float64
 14  AVG_CHOKE_UOM             0 non-null      object
 15  AVG_WHP_P                 0 non-null      float64
 16  AVG_WHT_P                 0 non-null      float64
 17  DP_CHOKE_SIZE             3173 non-null   float64
 18  BORE_OIL_VOL              0 non-null      float64
 19  BORE_GAS_VOL              0 non-null      float64
 20  BORE_WAT_VOL              0 non-null      float64
 21  BORE_WI_VOL               2990 non-null   float64
 22  FLOW_KIND                 3327 non-null   object
 23  WELL_TYPE                 3327 non-null   object
dtypes: datetime64[ns](1), float64(13), int64(3), object(7)
memory usage: 649.8+ KB
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3306 entries, 12328 to 15633
Data columns (total 24 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   DATEPRD                   3306 non-null   datetime64[ns]
 1   WELL_BORE_CODE            3306 non-null   object
 2   NPD_WELL_BORE_CODE        3306 non-null   int64
 3   NPD_WELL_BORE_NAME        3306 non-null   object
 4   NPD_FIELD_CODE            3306 non-null   int64
 5   NPD_FIELD_NAME            3306 non-null   object
 6   NPD_FACILITY_CODE         3306 non-null   int64
 7   NPD_FACILITY_NAME         3306 non-null   object
 8   ON_STREAM_HRS             3173 non-null   float64
 9   AVG_DOWNHOLE_PRESSURE     0 non-null      float64
 10  AVG_DOWNHOLE_TEMPERATURE  0 non-null      float64
 11  AVG_DP_TUBING             0 non-null      float64
 12  AVG_ANNULUS_PRESS         160 non-null    float64
 13  AVG_CHOKE_SIZE_P          160 non-null    float64
 14  AVG_CHOKE_UOM             160 non-null    object
 15  AVG_WHP_P                 160 non-null    float64
 16  AVG_WHT_P                 151 non-null    float64
 17  DP_CHOKE_SIZE             3172 non-null   float64
 18  BORE_OIL_VOL              160 non-null    float64
 19  BORE_GAS_VOL              160 non-null    float64
 20  BORE_WAT_VOL              160 non-null    float64
 21  BORE_WI_VOL               2716 non-null   float64
 22  FLOW_KIND                 3306 non-null   object
 23  WELL_TYPE                 3306 non-null   object
dtypes: datetime64[ns](1), float64(13), int64(3), object(7)
memory usage: 645.7+ KB
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3056 entries, 1911 to 4966
Data columns (total 24 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   DATEPRD                    3056 non-null   datetime64[ns]
 1   WELL_BORE_CODE             3056 non-null   object
 2   NPD_WELL_BORE_CODE         3056 non-null   int64
 3   NPD_WELL_BORE_NAME         3056 non-null   object
 4   NPD_FIELD_CODE             3056 non-null   int64
 5   NPD_FIELD_NAME             3056 non-null   object
 6   NPD_FACILITY_CODE          3056 non-null   int64
 7   NPD_FACILITY_NAME          3056 non-null   object
 8   ON_STREAM_HRS              3056 non-null   float64
 9   AVG_DOWNHOLE_PRESSURE      3050 non-null   float64
 10  AVG_DOWNHOLE_TEMPERATURE   3050 non-null   float64
 11  AVG_DP_TUBING              3050 non-null   float64
 12  AVG_ANNULUS_PRESS          3043 non-null   float64
 13  AVG_CHOKE_SIZE_P           3012 non-null   float64
 14  AVG_CHOKE_UOM              3056 non-null   object
 15  AVG_WHP_P                  3056 non-null   float64
 16  AVG_WHT_P                  3056 non-null   float64
 17  DP_CHOKE_SIZE              3056 non-null   float64
 18  BORE_OIL_VOL               3056 non-null   float64
 19  BORE_GAS_VOL               3056 non-null   float64
 20  BORE_WAT_VOL               3056 non-null   float64
 21  BORE_WI_VOL                0 non-null      float64
 22  FLOW_KIND                  3056 non-null   object
 23  WELL_TYPE                  3056 non-null   object
dtypes: datetime64[ns](1), float64(13), int64(3), object(7)
memory usage: 596.9+ KB
```

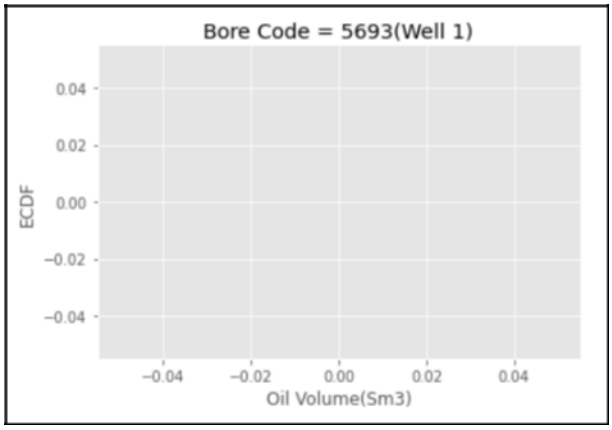
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3056 entries, 4967 to 8022
Data columns (total 24 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   DATEPRD                    3056 non-null   datetime64[ns]
 1   WELL_BORE_CODE             3056 non-null   object
 2   NPD_WELL_BORE_CODE         3056 non-null   int64
 3   NPD_WELL_BORE_NAME         3056 non-null   object
 4   NPD_FIELD_CODE             3056 non-null   int64
 5   NPD_FIELD_NAME             3056 non-null   object
 6   NPD_FACILITY_CODE          3056 non-null   int64
 7   NPD_FACILITY_NAME          3056 non-null   object
 8   ON_STREAM_HRS              3056 non-null   float64
 9   AVG_DOWNHOLE_PRESSURE      3050 non-null   float64
 10  AVG_DOWNHOLE_TEMPERATURE   3050 non-null   float64
 11  AVG_DP_TUBING              3050 non-null   float64
 12  AVG_ANNULUS_PRESS          2533 non-null   float64
 13  AVG_CHOKE_SIZE_P           2860 non-null   float64
 14  AVG_CHOKE_UOM              3056 non-null   object
 15  AVG_WHP_P                  3056 non-null   float64
 16  AVG_WHT_P                  3056 non-null   float64
 17  DP_CHOKE_SIZE              3056 non-null   float64
 18  BORE_OIL_VOL               3056 non-null   float64
 19  BORE_GAS_VOL               3056 non-null   float64
 20  BORE_WAT_VOL               3056 non-null   float64
 21  BORE_WI_VOL                0 non-null      float64
 22  FLOW_KIND                  3056 non-null   object
 23  WELL_TYPE                  3056 non-null   object
dtypes: datetime64[ns](1), float64(13), int64(3), object(7)
memory usage: 596.9+ KB
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1165 entries, 746 to 1910
Data columns (total 24 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   DATEPRD                    1165 non-null   datetime64[ns]
 1   WELL_BORE_CODE             1165 non-null   object
 2   NPD_WELL_BORE_CODE         1165 non-null   int64
 3   NPD_WELL_BORE_NAME         1165 non-null   object
 4   NPD_FIELD_CODE             1165 non-null   int64
 5   NPD_FIELD_NAME             1165 non-null   object
 6   NPD_FACILITY_CODE          1165 non-null   int64
 7   NPD_FACILITY_NAME          1165 non-null   object
 8   ON_STREAM_HRS              1165 non-null   float64
 9   AVG_DOWNHOLE_PRESSURE      1159 non-null   float64
 10  AVG_DOWNHOLE_TEMPERATURE   1159 non-null   float64
 11  AVG_DP_TUBING              1159 non-null   float64
 12  AVG_ANNULUS_PRESS          1159 non-null   float64
 13  AVG_CHOKE_SIZE_P           1163 non-null   float64
 14  AVG_CHOKE_UOM              1165 non-null   object
 15  AVG_WHP_P                  1159 non-null   float64
 16  AVG_WHT_P                  1159 non-null   float64
 17  DP_CHOKE_SIZE              1159 non-null   float64
 18  BORE_OIL_VOL               1165 non-null   float64
 19  BORE_GAS_VOL               1165 non-null   float64
 20  BORE_WAT_VOL               1165 non-null   float64
 21  BORE_WI_VOL                0 non-null      float64
 22  FLOW_KIND                  1165 non-null   object
 23  WELL_TYPE                  1165 non-null   object
dtypes: datetime64[ns](1), float64(13), int64(3), object(7)
memory usage: 227.5+ KB
```

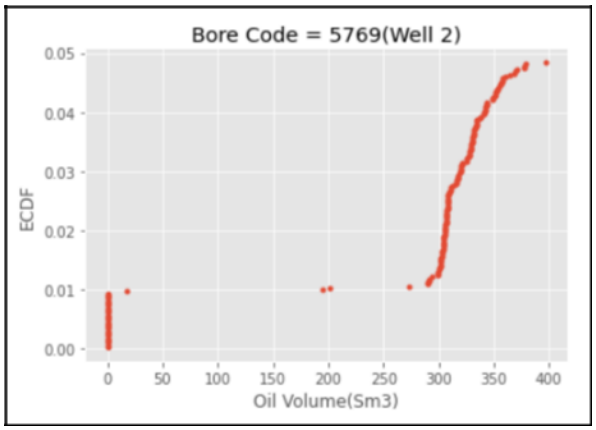
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 978 entries, 8023 to 9000
Data columns (total 24 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   DATEPRD                    978 non-null    datetime64[ns]
 1   WELL_BORE_CODE             978 non-null    object
 2   NPD_WELL_BORE_CODE         978 non-null    int64
 3   NPD_WELL_BORE_NAME         978 non-null    object
 4   NPD_FIELD_CODE             978 non-null    int64
 5   NPD_FIELD_NAME             978 non-null    object
 6   NPD_FACILITY_CODE          978 non-null    int64
 7   NPD_FACILITY_NAME          978 non-null    object
 8   ON_STREAM_HRS              978 non-null    float64
 9   AVG_DOWNHOLE_PRESSURE      978 non-null    float64
 10  AVG_DOWNHOLE_TEMPERATURE   978 non-null    float64
 11  AVG_DP_TUBING              978 non-null    float64
 12  AVG_ANNULUS_PRESS          978 non-null    float64
 13  AVG_CHOKE_SIZE_P           978 non-null    float64
 14  AVG_CHOKE_UOM              978 non-null    object
 15  AVG_WHP_P                  978 non-null    float64
 16  AVG_WHT_P                  978 non-null    float64
 17  DP_CHOKE_SIZE              978 non-null    float64
 18  BORE_OIL_VOL               978 non-null    float64
 19  BORE_GAS_VOL               978 non-null    float64
 20  BORE_WAT_VOL               978 non-null    float64
 21  BORE_WI_VOL                0 non-null      float64
 22  FLOW_KIND                  978 non-null    object
 23  WELL_TYPE                  978 non-null    object
dtypes: datetime64[ns](1), float64(13), int64(3), object(7)
memory usage: 191.0+ KB
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 746 entries, 0 to 745
Data columns (total 24 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   DATEPRD                   746 non-null    datetime64[ns]
 1   WELL_BORE_CODE            746 non-null    object
 2   NPD_WELL_BORE_CODE        746 non-null    int64
 3   NPD_WELL_BORE_NAME        746 non-null    object
 4   NPD_FIELD_CODE            746 non-null    int64
 5   NPD_FIELD_NAME            746 non-null    object
 6   NPD_FACILITY_CODE         746 non-null    int64
 7   NPD_FACILITY_NAME         746 non-null    object
 8   ON_STREAM_HRS             746 non-null    float64
 9   AVG_DOWNHOLE_PRESSURE     743 non-null    float64
 10  AVG_DOWNHOLE_TEMPERATURE  743 non-null    float64
 11  AVG_DP_TUBING             743 non-null    float64
 12  AVG_ANNULUS_PRESS         17 non-null     float64
 13  AVG_CHOKE_SIZE_P          746 non-null    float64
 14  AVG_CHOKE_UOM             746 non-null    object
 15  AVG_WHP_P                 746 non-null    float64
 16  AVG_WHT_P                 746 non-null    float64
 17  DP_CHOKE_SIZE             746 non-null    float64
 18  BORE_OIL_VOL              746 non-null    float64
 19  BORE_GAS_VOL              746 non-null    float64
 20  BORE_WAT_VOL              746 non-null    float64
 21  BORE_WI_VOL               0 non-null      float64
 22  FLOW_KIND                 746 non-null    object
 23  WELL_TYPE                 746 non-null    object
dtypes: datetime64[ns](1), float64(13), int64(3), object(7)
memory usage: 145.7+ KB
None None None None None None None
```

```
#Creating a function using Empirical Cummulative Distribution Function to
plot the Oil Production of every well during its lifetime
plt.style.use('ggplot')
def ecdf(OilProd):
    n = len(OilProd)
    x = np.sort(OilProd)
    y = np.arange(1,n+1)/n
    return x,y


# Plotting Oil Production for WELL 1
x_axis, y_axis = ecdf(prod_df_well1['BORE_OIL_VOL'])
plt.plot(x_axis, y_axis, marker=".", linestyle="none")

#Labeling
plt.xlabel('Oil Volume(Sm3)')
plt.ylabel('ECDF')
plt.title('Bore Code = 5693(Well 1)')

plt.show()
```

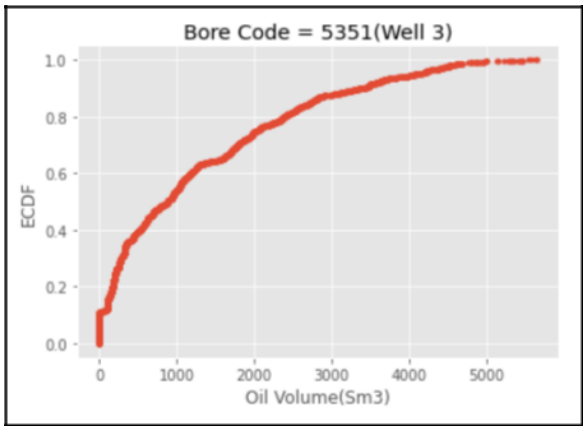



```
# Plotting Oil Production for WELL 2
x_axis, y_axis = ecdf(prod_df_well2['BORE_OIL_VOL'])
plt.plot(x_axis, y_axis, marker=".", linestyle="none")

#Labeling
plt.xlabel('Oil Volume(Sm3)')
plt.ylabel('ECDF')
plt.title('Bore Code = 5769(Well 2)')

plt.show()
```

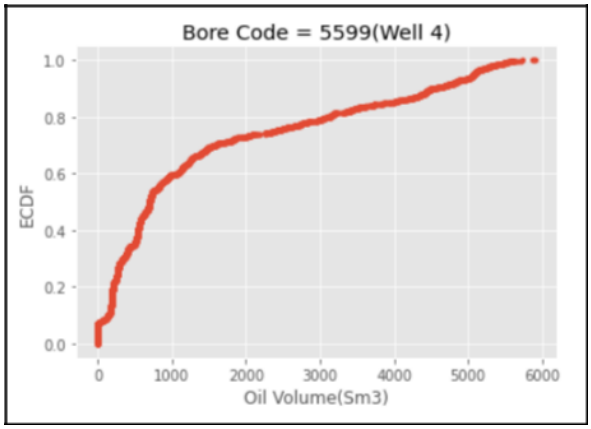



```
# Plotting Oil Production for WELL 3
x_axis, y_axis = ecdf(prod_df_well3['BORE_OIL_VOL'])
plt.plot(x_axis, y_axis, marker=".", linestyle="none")

#Labeling
plt.xlabel('Oil Volume(Sm3)')
plt.ylabel('ECDF')
plt.title('Bore Code = 5351(Well 3)')
```

```
plt.show()
```

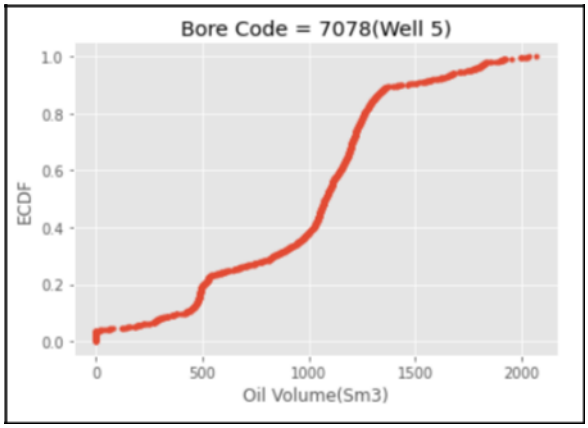



```
# Plotting Oil Production for WELL 4
x_axis, y_axis = ecdf(prod_df_well4['BORE_OIL_VOL'])
plt.plot(x_axis, y_axis, marker=".", linestyle="none")

#Labeling
plt.xlabel('Oil Volume(Sm3)')
plt.ylabel('ECDF')
plt.title('Bore Code = 5599(Well 4)')

plt.show()
```

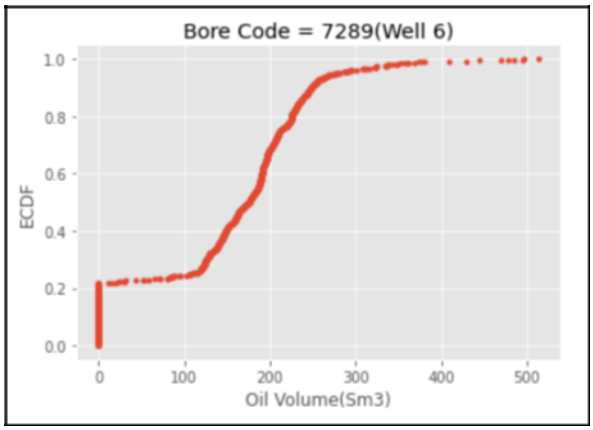



```
# Plotting Oil Production for WELL 5
x_axis, y_axis = ecdf(prod_df_well5['BORE_OIL_VOL'])
plt.plot(x_axis, y_axis, marker=".", linestyle="none")

#Labeling
plt.xlabel('Oil Volume(Sm3)')
```

```
plt.ylabel('ECDF')
plt.title('Bore Code = 7078(Well 5)')

plt.show()
```

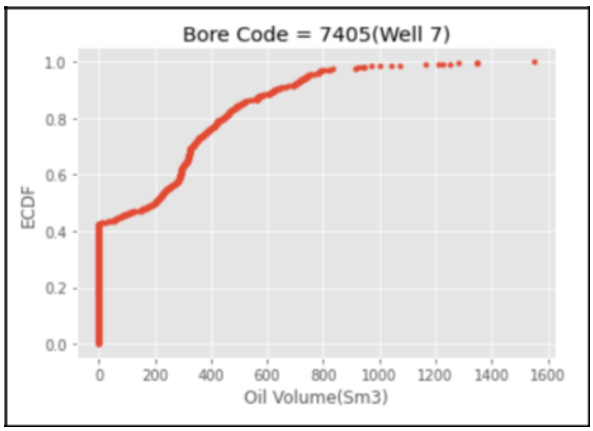



```
# Plotting Oil Production for WELL 6
x_axis, y_axis = ecdf(prod_df_well6['BORE_OIL_VOL'])
plt.plot(x_axis, y_axis, marker=".", linestyle="none")

#Labeling
plt.xlabel('Oil Volume(Sm3)')
plt.ylabel('ECDF')
plt.title('Bore Code = 7289(Well 6)')

plt.show()
```

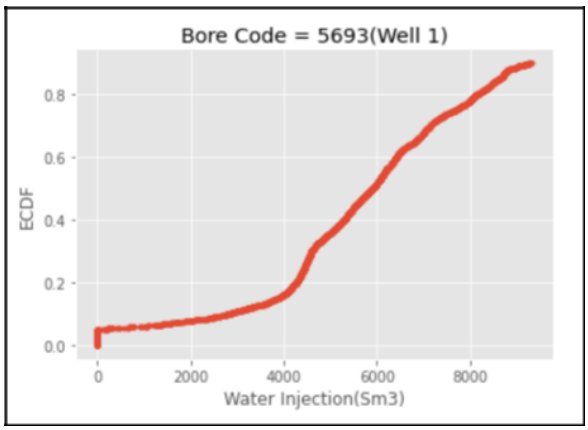



```
# Plotting Oil Production for WELL 7
x_axis, y_axis = ecdf(prod_df_well7['BORE_OIL_VOL'])
plt.plot(x_axis, y_axis, marker=".", linestyle="none")
```

```
#Labeling
plt.xlabel('Oil Volume(Sm3)')
plt.ylabel('ECDF')
plt.title('Bore Code = 7405(Well 7)')

plt.show()
```

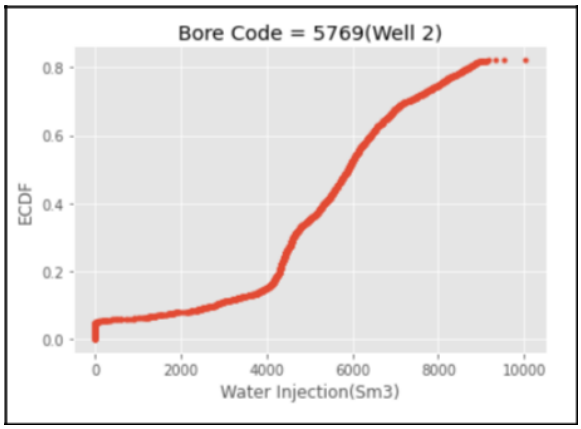



```
#Plotting Water injection for Well 1
x_axis, y_axis = ecdf(prod_df_well1['BORE_WI_VOL'])
plt.plot(x_axis, y_axis, marker=".", linestyle="none")

#Labeling
plt.xlabel('Water Injection(Sm3)')
plt.ylabel('ECDF')
plt.title('Bore Code = 5693(Well 1)')

plt.show()
```




```
#Plotting Water injection for Well 2
x_axis, y_axis = ecdf(prod_df_well2['BORE_WI_VOL'])
```

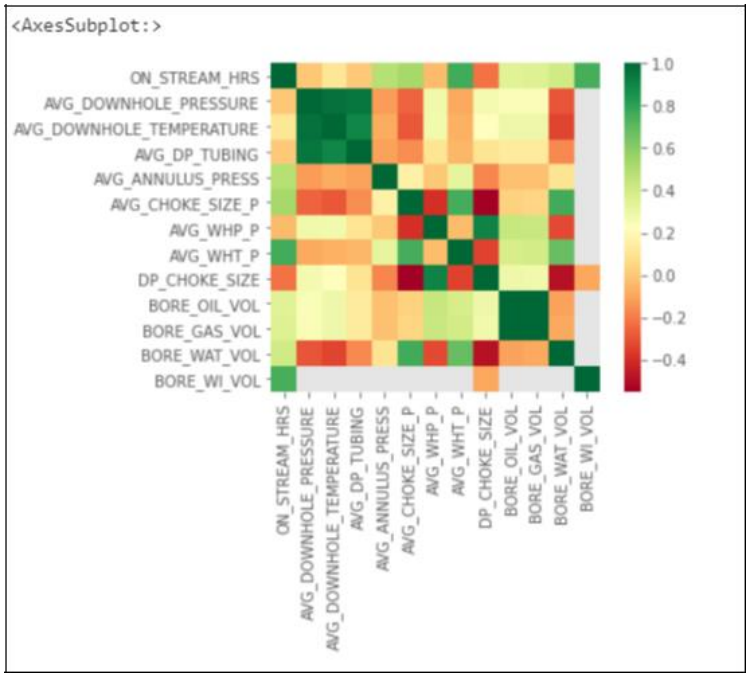
```
plt.plot(x_axis, y_axis, marker=".", linestyle="none")

#Labeling
plt.xlabel('Water Injection(Sm3)')
plt.ylabel('ECDF')
plt.title('Bore Code = 5769(Well 2)')

plt.show()
```




```
# Plotting Heatmaps for correlation visualization between columns
with numerical quantities
prod_df1=prod_df.select_dtypes(include='float')
print(prod_df1.info())
```

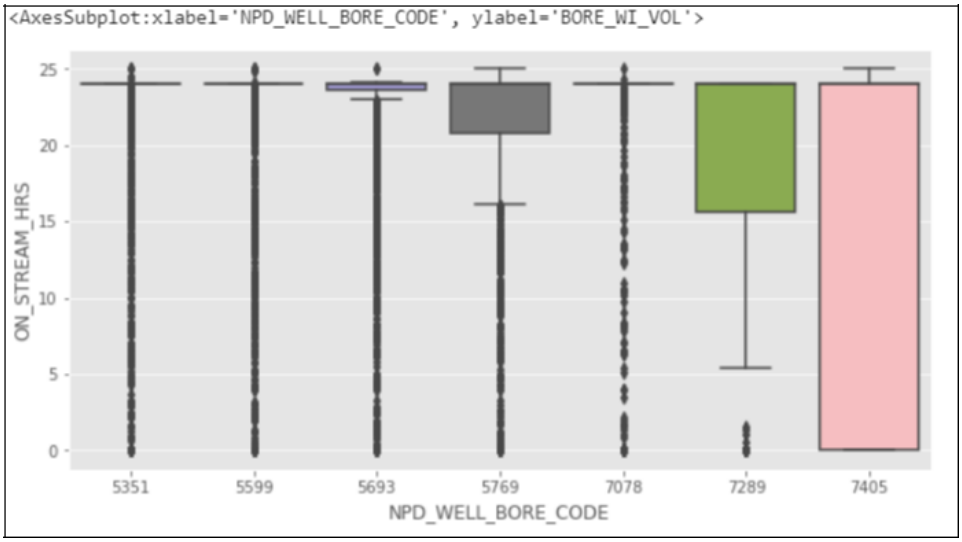
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15634 entries, 0 to 15633
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   ON_STREAM_HRS             15349 non-null  float64
 1   AVG_DOWNHOLE_PRESSURE     8980 non-null   float64
 2   AVG_DOWNHOLE_TEMPERATURE  8980 non-null   float64
 3   AVG_DP_TUBING             8980 non-null   float64
 4   AVG_ANNULUS_PRESS         7890 non-null   float64
 5   AVG_CHOKE_SIZE_P          8919 non-null   float64
 6   AVG_WHP_P                 9155 non-null   float64
 7   AVG_WHT_P                 9146 non-null   float64
 8   DP_CHOKE_SIZE             15340 non-null  float64
 9   BORE_OIL_VOL              9161 non-null   float64
 10  BORE_GAS_VOL              9161 non-null   float64
 11  BORE_WAT_VOL              9161 non-null   float64
 12  BORE_WI_VOL               5706 non-null   float64
dtypes: float64(13)
memory usage: 1.6 MB
None
```

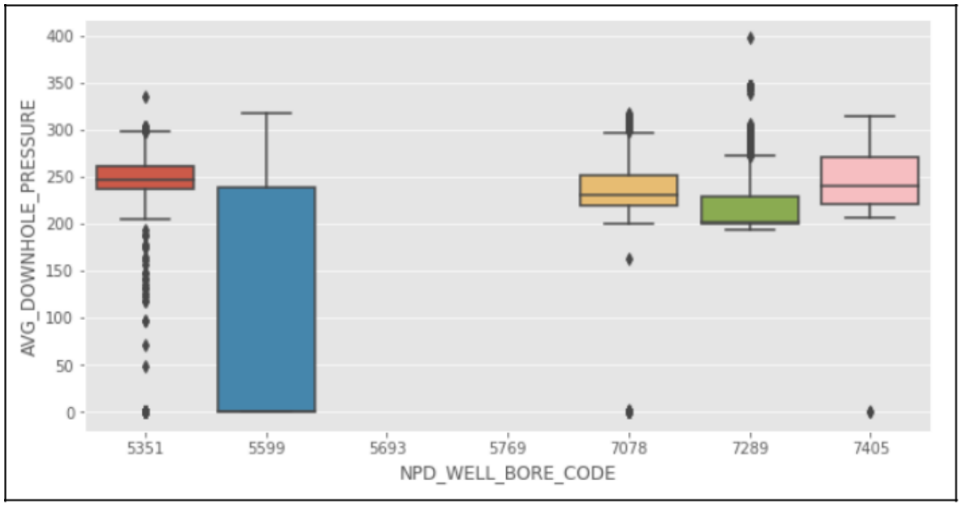
```
sns.heatmap(prod_df1.corr(), square=True, cmap='RdYlGn')
```

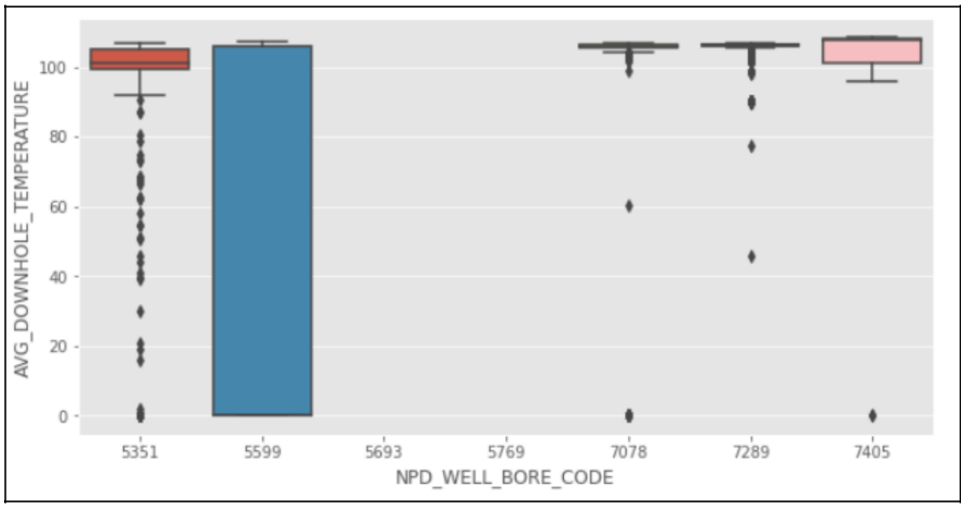


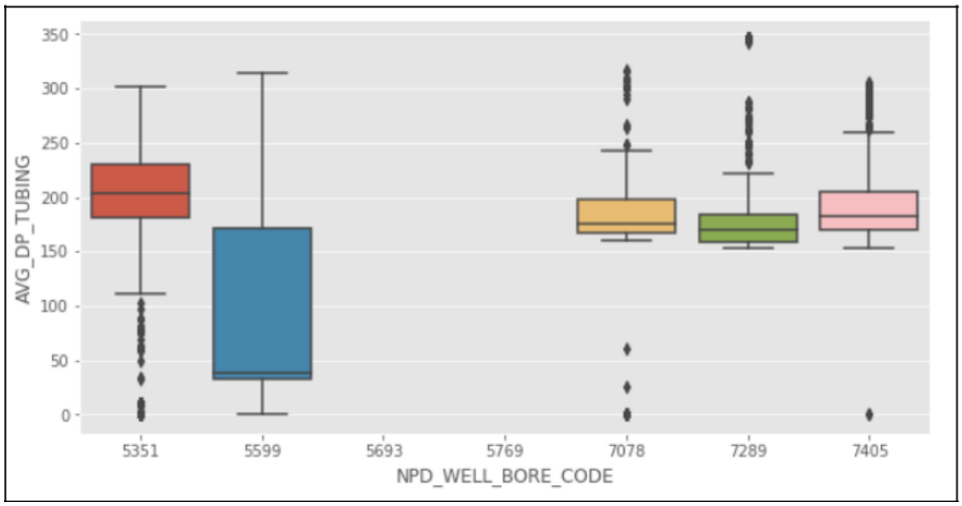

```
#Plotting boxplots for the visualization of the data
distribution figuresizes = (10,5)
plt.figure(figsize=figuresizes)
sns.boxplot(x = 'NPD_WELL_BORE_CODE', y = 'ON_STREAM_HRS', data=prod_df)

plt.figure(figsize=figuresizes)
sns.boxplot(x = 'NPD_WELL_BORE_CODE', y =
'AVG_DOWNHOLE_PRESSURE', data=prod_df)

plt.figure(figsize=figuresizes)
sns.boxplot(x = 'NPD_WELL_BORE_CODE', y =
'AVG_DOWNHOLE_TEMPERATURE', data=prod_df)

plt.figure(figsize=figuresizes)
sns.boxplot(x = 'NPD_WELL_BORE_CODE', y = 'AVG_DP_TUBING', data=prod_df)

plt.figure(figsize=figuresizes)
sns.boxplot(x = 'NPD_WELL_BORE_CODE', y = 'AVG_ANNULUS_PRESS', data=prod_df)

plt.figure(figsize=figuresizes)
sns.boxplot(x = 'NPD_WELL_BORE_CODE', y = 'AVG_CHOKE_SIZE_P', data=prod_df)

plt.figure(figsize=figuresizes)
sns.boxplot(x = 'NPD_WELL_BORE_CODE', y = 'AVG_WHP_P', data=prod_df)
```

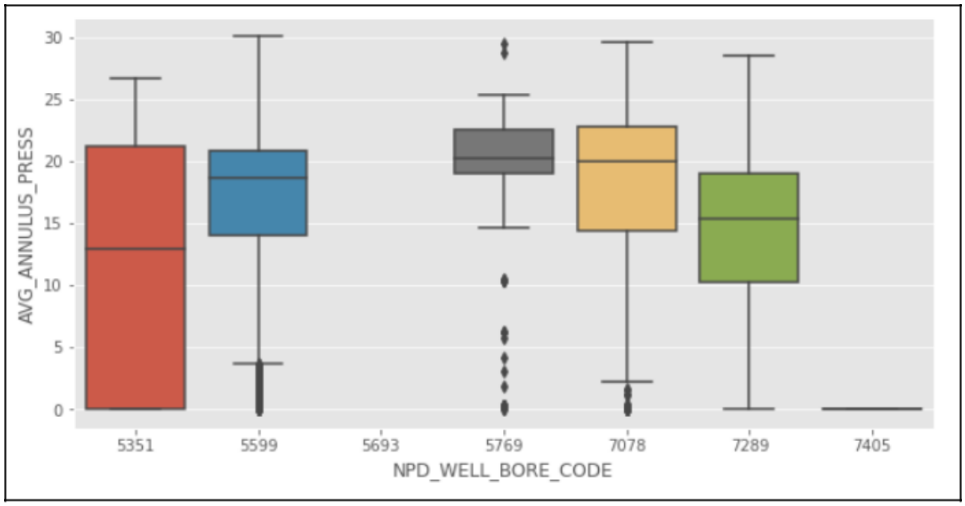
```
plt.figure(figsize=figuresizes)
sns.boxplot(x = 'NPD_WELL_BORE_CODE', y = 'AVG_WHT_P', data=prod_df)

plt.figure(figsize=figuresizes)
sns.boxplot(x = 'NPD_WELL_BORE_CODE', y = 'DP_CHOKE_SIZE', data=prod_df)

plt.figure(figsize=figuresizes)
sns.boxplot(x = 'NPD_WELL_BORE_CODE', y = 'BORE_OIL_VOL', data=prod_df)

plt.figure(figsize=figuresizes)
sns.boxplot(x = 'NPD_WELL_BORE_CODE', y = 'BORE_GAS_VOL', data=prod_df)

plt.figure(figsize=figuresizes)
sns.boxplot(x = 'NPD_WELL_BORE_CODE', y = 'BORE_WAT_VOL', data=prod_df)

plt.figure(figsize=figuresizes)
sns.boxplot(x = 'NPD_WELL_BORE_CODE', y = 'BORE_WI_VOL', data=prod_df)
```

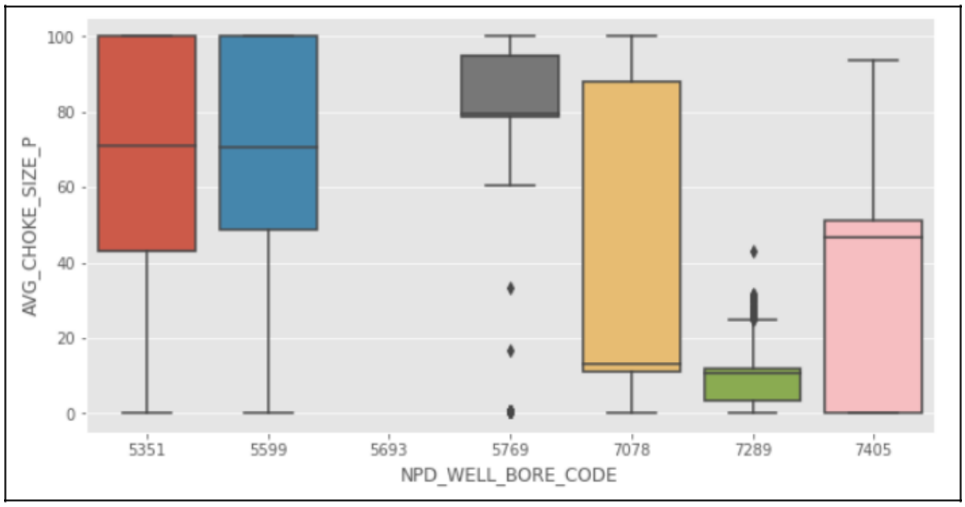


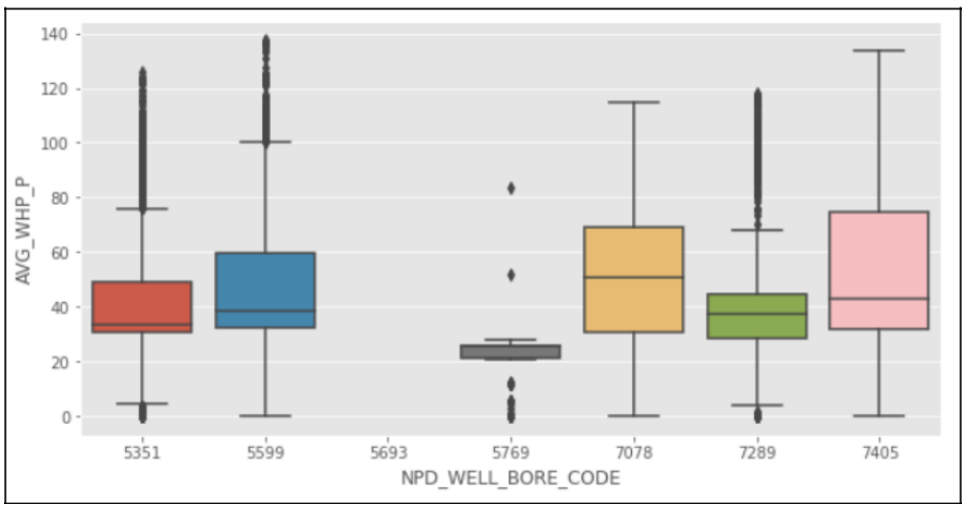

AVG_DOWNHOLE_PRESSURE
400
350
300
250
200
150
100
50
0
5351
5599
5693
5769
7078
7289
7405
NPD_WELL_BORE_CODE

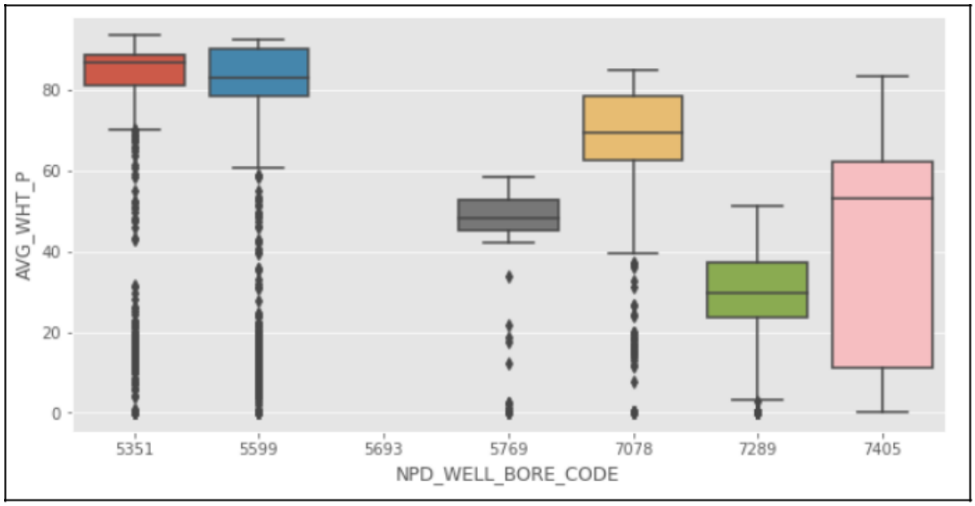

AVG_DOWNHOLE_TEMPERATURE
100
80
60
40
20
0
5351
5599
5693
5769
7078
7289
7405
NPD_WELL_BORE_CODE

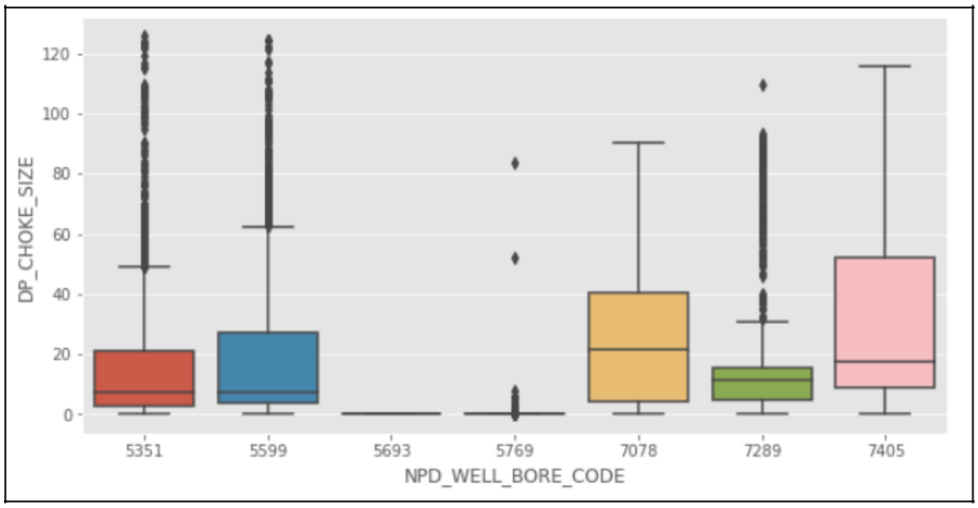

AVG_DP_TUBING
350
300
250
200
150
100
50
0
5351
5599
5693
5769
7078
7289
7405
NPD_WELL_BORE_CODE

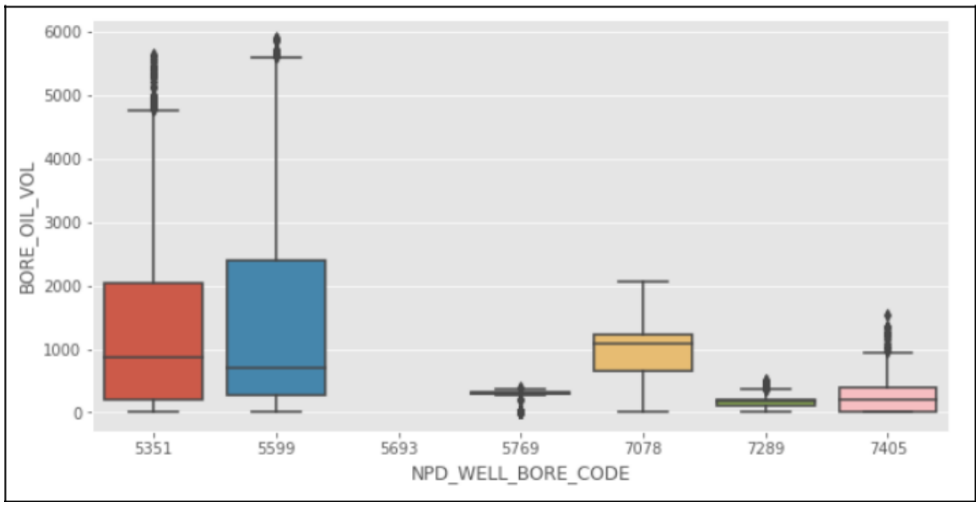

AVG_ANNULUS_PRESS
30
25
20
15
10
5
0
5351
5599
5693
5769
7078
7289
7405
NPD_WELL_BORE_CODE

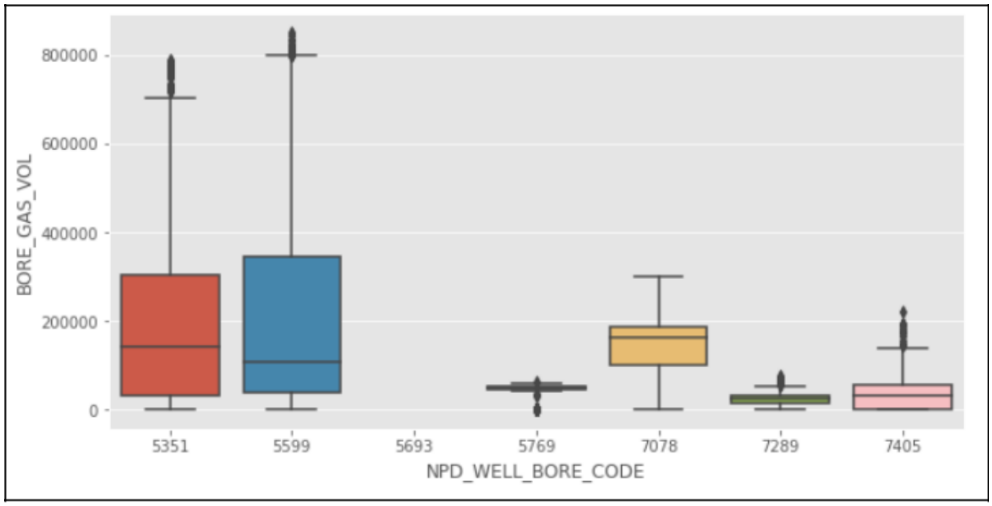

AVG_CHOKE_SIZE_P
100
80
60
40
20
0
5351
5599
5693
5769
7078
7289
7405
NPD_WELL_BORE_CODE

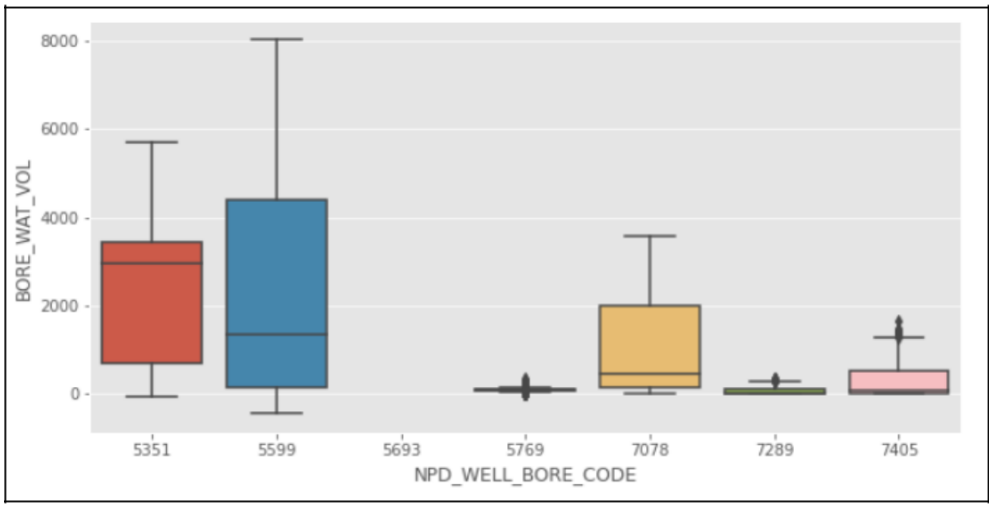

AVG_WHP_P
140
120
100
80
60
40
20
0
5351
5599
5693
5769
7078
7289
7405
NPD_WELL_BORE_CODE

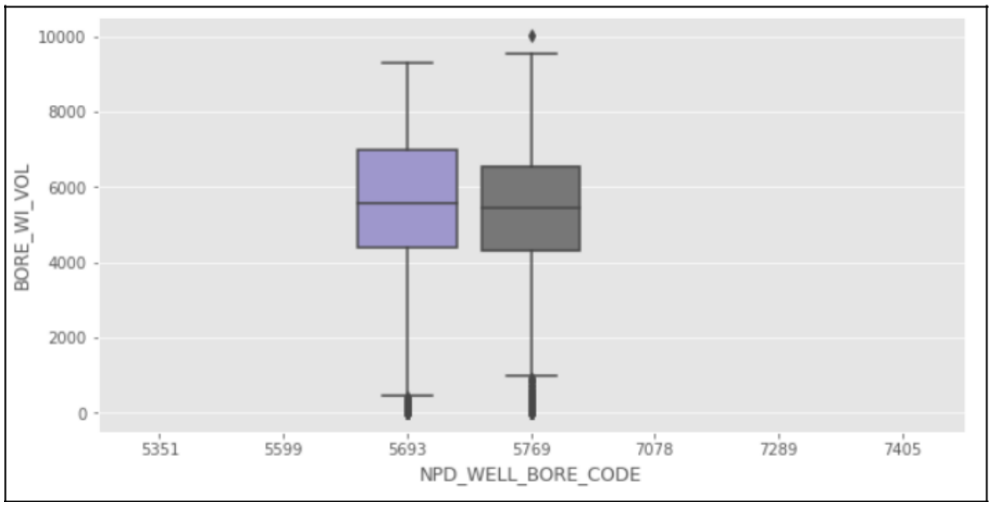

AVG_WHT_P
0
20
40
60
80
5351
5599
5693
5769
7078
7289
7405
NPD_WELL_BORE_CODE


DP_CHOKE_SIZE
0
20
40
60
80
100
120
5351
5599
5693
5769
7078
7289
7405
NPD_WELL_BORE_CODE


BORE_OIL_VOL
0
1000
2000
3000
4000
5000
6000
5351
5599
5693
5769
7078
7289
7405
NPD_WELL_BORE_CODE







Modelo de Regresión Lineal

```
#Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns
!pip install xlrd==1.2.0
!pip install -U scikit-learn
!pip install XlsxWriter

#Import the xls raw production file using pandas
prod_df=pd.read_excel('Daily Production.xls')

#Dropping categorical columns and useless columns for this analysis
to_drop = ['WELL_BORE_CODE',
'NPD_WELL_BORE_NAME',
'AVG_CHOKE_SIZE_P',
'AVG_ANNULUS_PRESS',
'NPD_FIELD_CODE',
'NPD_FIELD_NAME',
'NPD_FACILITY_CODE',
'NPD_FACILITY_NAME',
'FLOW_KIND',
'WELL_TYPE',
'AVG_CHOKE_UOM',
'BORE_GAS_VOL',
'BORE_WAT_VOL',
'BORE_WI_VOL']
prod_df.drop(to_drop, inplace=True, axis=1)

#Dropping water injection rows
prod_df = prod_df.drop(labels=range(9001, 15634), axis=0)

#Filtering rows with more than zero hours of
streaming prod_df=prod_df[prod_df.ON_STREAM_HRS>0]

#Exporting my first dataframe into a MS Excel sheet
from xlsxwriter import Workbook
datatoexcel=pd.ExcelWriter("First_DataFrame_LinearRegression.xlsx",
engine='xlsxwriter')
prod_df.to_excel(datatoexcel, sheet_name='Sheet1')
datatoexcel.save()

#Converting zero values into NaN values in order to apply interpolation
prod_df.loc[prod_df['AVG_DOWNHOLE_PRESSURE'] == 0,'AVG_DOWNHOLE_PRESSURE']
= np.nan
```

```
prod_df.loc[prod_df['AVG_DOWNHOLE_TEMPERATURE']
== 0,'AVG_DOWNHOLE_TEMPERATURE'] = np.nan

#Exporting my dataframe with NaN values into a MS Excel sheet
datatoexcel=pd.ExcelWriter("Dataframe_NaN_LinearRegression.xlsx",
engine='xlsxwriter')
prod_df.to_excel(datatoexcel, sheet_name='Sheet1')
datatoexcel.save()

#Using interpolate function to fill the NaN
values prod_df=prod_df.interpolate()

#Exporting my interpolated dataframe into a MS Excel sheet
datatoexcel=pd.ExcelWriter("Dataframe_Interpolate_LinearRegression.xlsx",
engine='xlsxwriter')
prod_df.to_excel(datatoexcel, sheet_name='Sheet1')
datatoexcel.save()

#Scaling dataset to remove difference in distribution within
columns from sklearn.preprocessing import MinMaxScaler scaler =
MinMaxScaler()
prod_df[['ON_STREAM_HRS','AVG_DOWNHOLE_TEMPERATURE','AVG_DOWNHOLE_PRESSURE','
AVG_WHP_P'
         ,'AVG_WHT_P','AVG_DP_TUBING','DP_CHOKE_SIZE']]
= scaler.fit_transform(prod_df[['ON_STREAM_HRS',

'AVG_DOWNHOLE_TEMPERATURE',

'AVG_DOWNHOLE_PRESSURE',

'AVG_WHP_P',

'AVG_WHT_P',

'AVG_DP_TUBING',

'DP_CHOKE_SIZE']])


prod_df
```

	DATEPRD	NPD_WELL_BORE_CODE	ON_STREAM_HRS	AVG_DOWNHOLE_PRESSURE	AVG_DOWNHOLE_TEMPERATURE	AVG_DP_TUBING	AVG_WHP_P	AVG_WHT_P	DP_CHOKE_SIZE	BORE_OIL_VOL
14	2014-04-21	7405	0.456193	0.957160	0.946132	0.790448	0.798921	0.205293	0.658042	0.00
15	2014-04-22	7405	0.959718	0.918949	0.980130	0.702695	0.888107	0.405726	0.744427	631.47
16	2014-04-23	7405	0.959718	0.857637	0.992063	0.660213	0.820483	0.649736	0.666073	1166.46
17	2014-04-24	7405	0.959718	0.833993	0.994148	0.649364	0.782545	0.674228	0.622899	1549.81
18	2014-04-25	7405	0.959718	0.810609	0.995090	0.638931	0.744389	0.690274	0.579104	1248.70
...	...	...	...	...	...	...	...	...	...	...
8923	2016-07-02	7289	0.959718	0.617090	0.981650	0.691545	0.130809	0.524225	0.011881	144.01
8924	2016-07-03	7289	0.959718	0.617069	0.981631	0.691676	0.130478	0.523887	0.011308	145.22
8925	2016-07-04	7289	0.959718	0.614958	0.981684	0.689401	0.129888	0.535810	0.012068	142.74
8926	2016-07-05	7289	0.959718	0.617803	0.981550	0.693198	0.129114	0.533005	0.011356	144.46
8927	2016-07-06	7289	0.733132	0.618118	0.981554	0.693001	0.130351	0.521168	0.012287	106.30

7891 rows × 10 columns

```
#Final Scaled DataFrame for Training
datatoexcel=pd.ExcelWriter("Training_Dataset_LinearRegression.xlsx",
engine='xlsxwriter')
prod_df.to_excel(datatoexcel, sheet_name='Sheet1')
datatoexcel.save()
```

Aplicación del algoritmo:

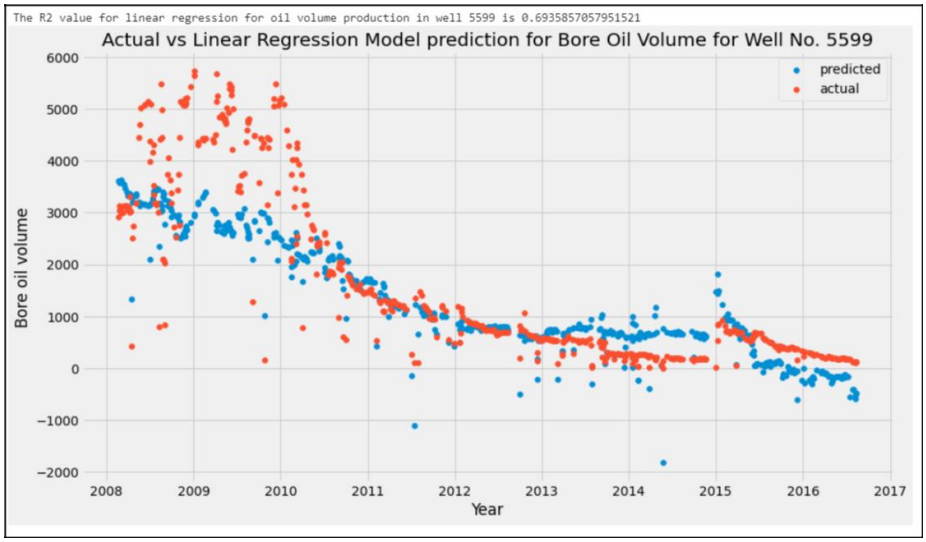
```
X = prod_df.drop(['BORE_OIL_VOL'],axis=1)
y= prod_df[['BORE_OIL_VOL','NPD_WELL_BORE_CODE']]

#Designation of the X and Y training and test datasets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size =
0.2) x_train_final = X_train.drop(['DATEPRD','NPD_WELL_BORE_CODE'],axis
= 1) x_test_final = X_test.drop(['DATEPRD',"NPD_WELL_BORE_CODE"],axis =
1) y_test_final = y_test['BORE_OIL_VOL'] y_train_final =
y_train['BORE_OIL_VOL']
X = prod_df.drop(['DATEPRD','BORE_OIL_VOL'],axis=1).values y=
prod_df['BORE_OIL_VOL']

# Applying Linear Regression and getting the value of accuracy of this
model from sklearn.linear_model import LinearRegression lin_reg=
LinearRegression()
lin_reg.fit(x_train_final, y_train_final)
y_pred = lin_reg.predict(x_test_final)
print("The R2 value for linear regression for oil volume production
is", lin_reg.score(x_test_final, y_test_final))
```

```
The R2 value for linear regression for oil volume production is 0.6695624221641976
```

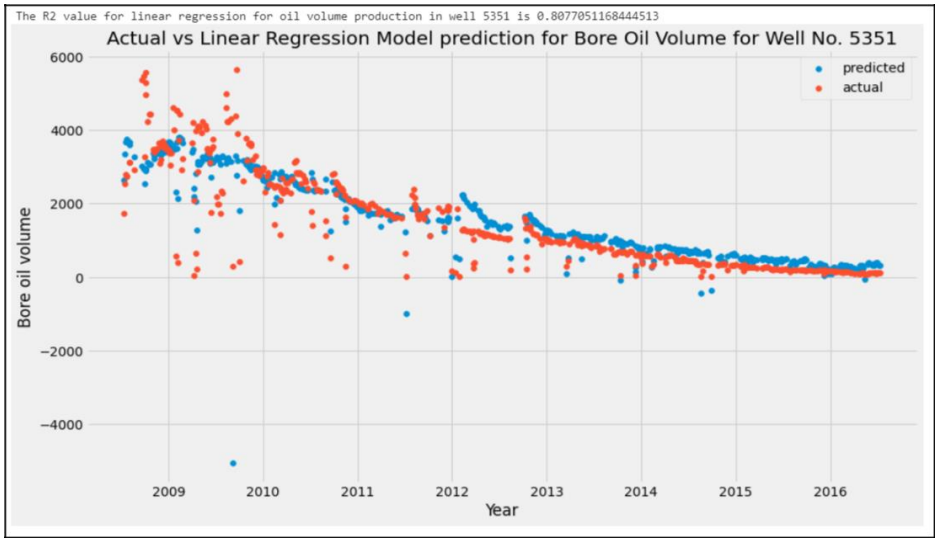
```
X_test.groupby(['NPD_WELL_BORE_CODE']).agg({"DATEPRD":"count"})
```

NPD_WELL_BORE_CODE	DATEPRD
5351	560
5599	577
7078	222
7289	139
7405	81

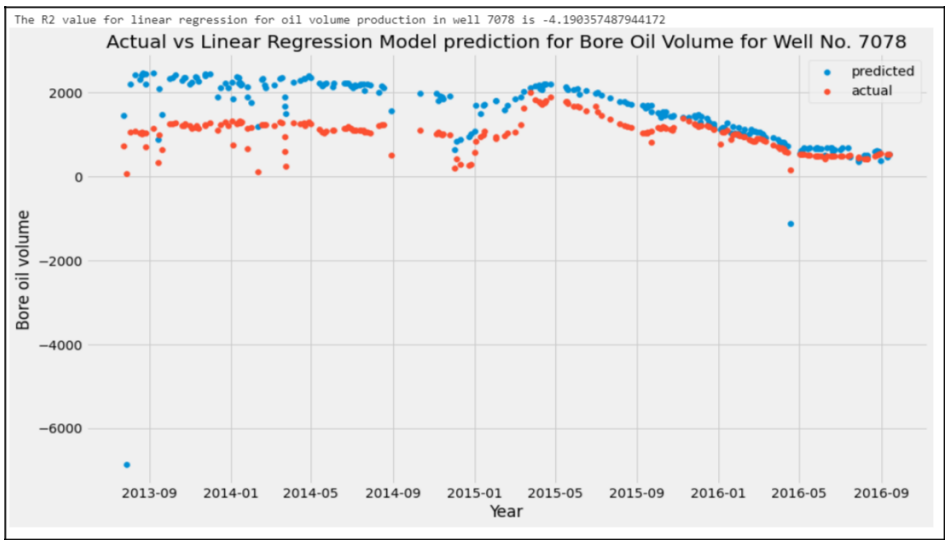
```
#Actual VS Model Prediction plot for Well No 5599

X_test_5599 = X_test[X_test["NPD_WELL_BORE_CODE"] == 5599]
y_test_5599 = y_test[y_test["NPD_WELL_BORE_CODE"] == 5599]
x_test_5599final = X_test_5599.drop(['DATEPRD',"NPD_WELL_BORE_CODE"],axis
= 1)

y_pred = lin_reg.predict(x_test_5599final)
plt.style.use('fivethirtyeight')
plt.figure(figsize = (14,8))
plt.scatter(X_test_5599["DATEPRD"].tolist(),y_pred,label='predicted')
plt.scatter(X_test_5599["DATEPRD"].tolist(),y_test_5599['BORE_OIL_VOL'],label
='actual')
plt.legend()
plt.xlabel("Year")
plt.ylabel("Bore oil volume")
plt.title('Actual vs Linear Regression Model prediction for Bore Oil
Volume for Well No. 5599')

y_test_5599_final = y_test_5599['BORE_OIL_VOL']
print("The R2 value for linear regression for oil volume production in
well 5599 is", lin_reg.score(x_test_5599final,y_test_5599_final ))
```



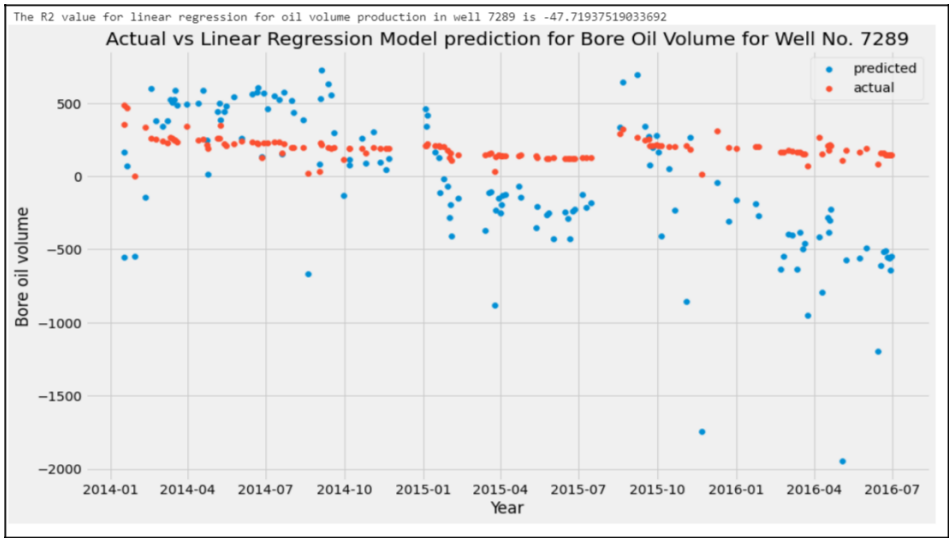

```
#Actual VS Model Prediction plot for Well No 5351

X_test_5351 = X_test[X_test["NPD_WELL_BORE_CODE"] == 5351]
y_test_5351 = y_test[y_test["NPD_WELL_BORE_CODE"] == 5351]
x_test_5351final = X_test_5351.drop(['DATEPRD',"NPD_WELL_BORE_CODE"],axis
= 1)

y_pred = lin_reg.predict(x_test_5351final)
plt.style.use('fivethirtyeight')
plt.figure(figsize = (14,8))
plt.scatter(X_test_5351["DATEPRD"].tolist(),y_pred,label='predicted')
plt.scatter(X_test_5351["DATEPRD"].tolist(),y_test_5351['BORE_OIL_VOL'],label
='actual')
plt.legend()
plt.xlabel("Year")
plt.ylabel("Bore oil volume")
plt.title('Actual vs Linear Regression Model prediction for Bore Oil
Volume for Well No. 5351')

y_test_5351_final = y_test_5351['BORE_OIL_VOL']
print("The R2 value for linear regression for oil volume production in
well 5351 is", lin_reg.score(x_test_5351final,y_test_5351_final ))
```



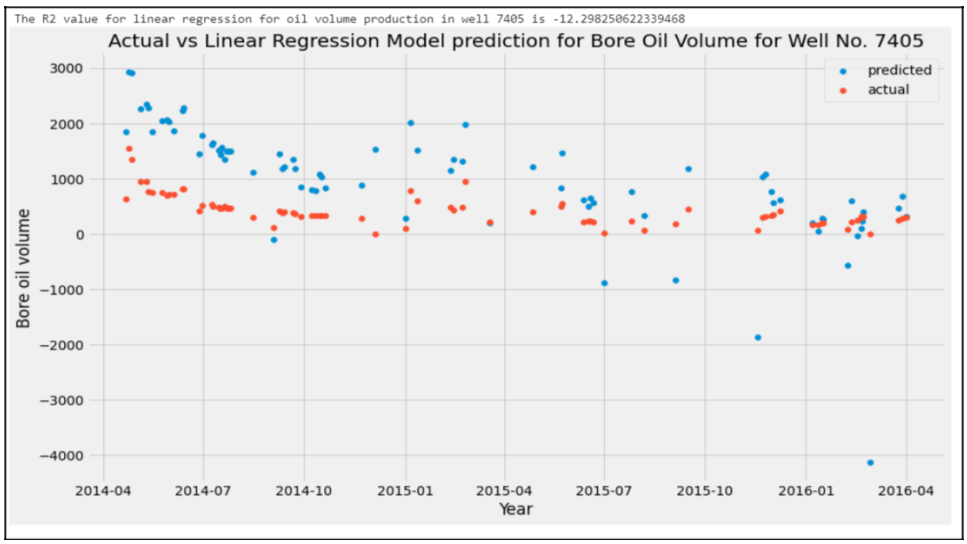

```
#Actual VS Model Prediction plot for Well No 7078

X_test_7078 = X_test[X_test["NPD_WELL_BORE_CODE"] == 7078]
y_test_7078 = y_test[y_test["NPD_WELL_BORE_CODE"] == 7078]
x_test_7078final = X_test_7078.drop(['DATEPRD',"NPD_WELL_BORE_CODE"],axis
= 1)

y_pred = lin_reg.predict(x_test_7078final)
plt.style.use('fivethirtyeight')
plt.figure(figsize = (14,8))
plt.scatter(X_test_7078["DATEPRD"].tolist(),y_pred,label='predicted')
plt.scatter(X_test_7078["DATEPRD"].tolist(),y_test_7078['BORE_OIL_VOL'],label
='actual')
plt.legend()
plt.xlabel("Year")
plt.ylabel("Bore oil volume")
plt.title('Actual vs Linear Regression Model prediction for Bore Oil
Volume for Well No. 7078')

y_test_7078_final = y_test_7078['BORE_OIL_VOL']
print("The R2 value for linear regression for oil volume production in
well 7078 is", lin_reg.score(x_test_7078final,y_test_7078_final ))
```




```
#Actual VS Model Prediction plot for Well No 7289

X_test_7289 = X_test[X_test["NPD_WELL_BORE_CODE"] == 7289]
y_test_7289 = y_test[y_test["NPD_WELL_BORE_CODE"] == 7289]
x_test_7289final = X_test_7289.drop(['DATEPRD',"NPD_WELL_BORE_CODE"],axis
= 1)

y_pred = lin_reg.predict(x_test_7289final)
plt.style.use('fivethirtyeight')
plt.figure(figsize = (14,8))
plt.scatter(X_test_7289["DATEPRD"].tolist(),y_pred,label='predicted')
plt.scatter(X_test_7289["DATEPRD"].tolist(),y_test_7289['BORE_OIL_VOL'],label
='actual')
plt.legend()
plt.xlabel("Year")
plt.ylabel("Bore oil volume")
plt.title('Actual vs Linear Regression Model prediction for Bore Oil
Volume for Well No. 7289')

y_test_7289_final = y_test_7289['BORE_OIL_VOL']
print("The R2 value for linear regression for oil volume production in
well 7289 is", lin_reg.score(x_test_7289final,y_test_7289_final ))
```




```
#Actual VS Model Prediction plot for Well No 7405

X_test_7405 = X_test[X_test["NPD_WELL_BORE_CODE"] == 7405]
y_test_7405 = y_test[y_test["NPD_WELL_BORE_CODE"] == 7405]
x_test_7405final = X_test_7405.drop(['DATEPRD',"NPD_WELL_BORE_CODE"],axis
= 1)

y_pred = lin_reg.predict(x_test_7405final)
plt.style.use('fivethirtyeight')
plt.figure(figsize = (14,8))
plt.scatter(X_test_7405["DATEPRD"].tolist(),y_pred,label='predicted')
plt.scatter(X_test_7405["DATEPRD"].tolist(),y_test_7405['BORE_OIL_VOL'],label
='actual')
plt.legend()
plt.xlabel("Year")
plt.ylabel("Bore oil volume")
plt.title('Actual vs Linear Regression Model prediction for Bore Oil
Volume for Well No. 7405')

y_test_7405_final = y_test_7405['BORE_OIL_VOL']
print("The R2 value for linear regression for oil volume production in
well 7405 is", lin_reg.score(x_test_7405final,y_test_7405_final ))
```



Modelo de Regresión Polinomial

Link al código:

https://github.com/saviourapi/Saviour/blob/ebf3686711aec22d6b34af4b5f39e866097206fd/notebook/PolynomialRegression.ipynb

```
#Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
!pip install xlrd==1.2.0
!pip install -U scikit-learn
!pip install XlsxWriter

#Import the xls production file using pandas
prod_df2=pd.read_excel('Daily Production.xls')

#Roundinf off the data to the nearest
integer prod_df2 = np.round(prod_df2)

#Dropping categorical columns and useless columns for this analysis
to_drop = ['WELL_BORE_CODE',
'NPD_WELL_BORE_NAME',
'NPD_FIELD_CODE',
'NPD_FIELD_NAME',
'NPD_FACILITY_CODE',
```

```
'NPD_FACILITY_NAME',
'AVG_DOWNHOLE_PRESSURE',
'AVG_DP_TUBING',
'BORE_WI_VOL',
'AVG_CHOKE_UOM',
'FLOW_KIND',
'WELL_TYPE',
'BORE_GAS_VOL',
'DP_CHOKE_SIZE',
'BORE_WAT_VOL']
prod_df2.drop(to_drop, inplace=True, axis=1)

#Exporting my first dataframe into a MS Excel sheet
from xlsxwriter import Workbook
datatoexcel=pd.ExcelWriter("First_DataFrame_PolynomialRegression.xlsx",
engine='xlsxwriter')
prod_df2.to_excel(datatoexcel, sheet_name='Sheet1')
datatoexcel.save()

#Using interpolate function to fill the NaN
values prod_df2=prod_df2.interpolate() prod_df2
```

	DATEPRD	NPD_WELL_BORE_CODE	ON_STREAM_HRS	AVG_DOWNHOLE_TEMPERATURE	AVG_ANNULUS_PRESS	AVG_CHOKE_SIZE_P	AVG_WHP_P	AVG_WHT_P	BORE_OIL_VOL
0	2014-04-07	7405	0.0	0.00	0.0	0.0	0.0	0.0	0.0
1	2014-04-08	7405	0.0	24.25	0.0	1.0	0.0	0.0	0.0
2	2014-04-09	7405	0.0	48.50	0.0	1.0	0.0	0.0	0.0
3	2014-04-10	7405	0.0	72.75	0.0	1.0	0.0	0.0	0.0
4	2014-04-11	7405	0.0	97.00	0.0	1.0	33.0	10.0	0.0
...	...	...	...	...	...	...	...	...	...
15629	2016-09-14	5769	0.0	90.00	0.0	1.0	0.0	0.0	0.0
15630	2016-09-15	5769	0.0	90.00	0.0	1.0	0.0	0.0	0.0
15631	2016-09-16	5769	0.0	90.00	0.0	1.0	0.0	0.0	0.0
15632	2016-09-17	5769	0.0	90.00	0.0	1.0	0.0	0.0	0.0
15633	2016-09-18	5769	0.0	90.00	0.0	1.0	0.0	0.0	0.0

15634 rows × 9 columns

```
#Exporting my interpolated dataframe into a MS Excel sheet
datatoexcel=pd.ExcelWriter("Dataframe_Interpolate_PolynomialRegression.xlsx",
engine='xlsxwriter')
prod_df2.to_excel(datatoexcel, sheet_name='Sheet1')
datatoexcel.save()

#Scaling dataset to remove difference in distributions within columns
```

```
from sklearn.preprocessing import
MinMaxScaler scaler = MinMaxScaler()
prod_df2[['ON_STREAM_HRS','AVG_DOWNHOLE_TEMPERATURE','AVG_ANNULUS_PRESS','AVG
_CHOKE_SIZE_P','AVG_WHP_P','AVG_WHT_P']] =
scaler.fit_transform(prod_df2[['ON_STREAM_HRS',
'AVG_DOWNHOLE_TEMPERATURE','AVG_ANNULUS_PRESS', 'AVG_CHOKE_SIZE_P','AVG_WHP_P',
'AVG_WHT_P']])

#Taking the data of wells 2 to 4 since they were the ones with the
least number of missing values
newdf = prod_df2.loc[746 : 8022]
X = newdf.drop(['BORE_OIL_VOL'],axis=1)
y= newdf[['BORE_OIL_VOL','NPD_WELL_BORE_CODE']]

newdf
```

	DATEPRD	NPD_WELL_BORE_CODE	ON_STREAM_HRS	AVG_DOWNHOLE_TEMPERATURE	AVG_ANNULUS_PRESS	AVG_CHOKE_SIZE_P	AVG_WHP_P	AVG_WHT_P	BORE_OIL_VOL
746	2013-07-08	7078	0.0	0.0	0.000000	0.00	0.000000	0.0	0.0
747	2013-07-09	7078	0.0	0.0	0.000000	0.00	0.000000	0.0	0.0
748	2013-07-10	7078	0.0	0.0	0.000000	0.00	0.000000	0.0	0.0
749	2013-07-11	7078	0.0	0.0	0.000000	0.00	0.000000	0.0	0.0
750	2013-07-12	7078	0.0	0.0	0.000000	0.00	0.000000	0.0	0.0
...	...	...	...	...	...	...	...	...	...
8018	2016-09-13	5351	0.0	0.0	0.366667	0.01	0.080292	0.0	0.0
8019	2016-09-14	5351	0.0	0.0	0.366667	0.01	0.080292	0.0	0.0
8020	2016-09-15	5351	0.0	0.0	0.366667	0.01	0.080292	0.0	0.0
8021	2016-09-16	5351	0.0	0.0	0.366667	0.01	0.080292	0.0	0.0
8022	2016-09-17	5351	0.0	0.0	0.366667	0.01	0.080292	0.0	0.0

7277 rows × 9 columns

Aplicación del algoritmo

```
#Designation of the X and Y training and test datasets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size =
0.3, random_state=42)
x_train_final = X_train.drop(['DATEPRD','NPD_WELL_BORE_CODE'],axis =
1) x_test_final = X_test.drop(['DATEPRD',"NPD_WELL_BORE_CODE"],axis =
1) y_test_final = y_test['BORE_OIL_VOL'] y_train_final =
y_train['BORE_OIL_VOL']
X = newdf.drop(['DATEPRD','BORE_OIL_VOL'],axis=1).values

y= newdf['BORE_OIL_VOL']

#Final Scaled DataFrame for Training
```

```
datatoexcel=pd.ExcelWriter("Training_Dataset_PolynomialRegression.xlsx",
engine='xlsxwriter')
newdf.to_excel(datatoexcel, sheet_name='Sheet1')
datatoexcel.save()

# Applying Polynomial Regression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(x_train_final)
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y_train_final)
x_pol_test = poly_reg.fit_transform(x_test_final)
X_poly.shape[1]
```

```
210
```

```
#Getting the value of accuracy of this
model y_pred = lin_reg.predict(x_pol_test)
print("The R2 value for Polynomial regression(4th order) for oil
volume production is",lin_reg.score(x_pol_test, y_test_final))
```

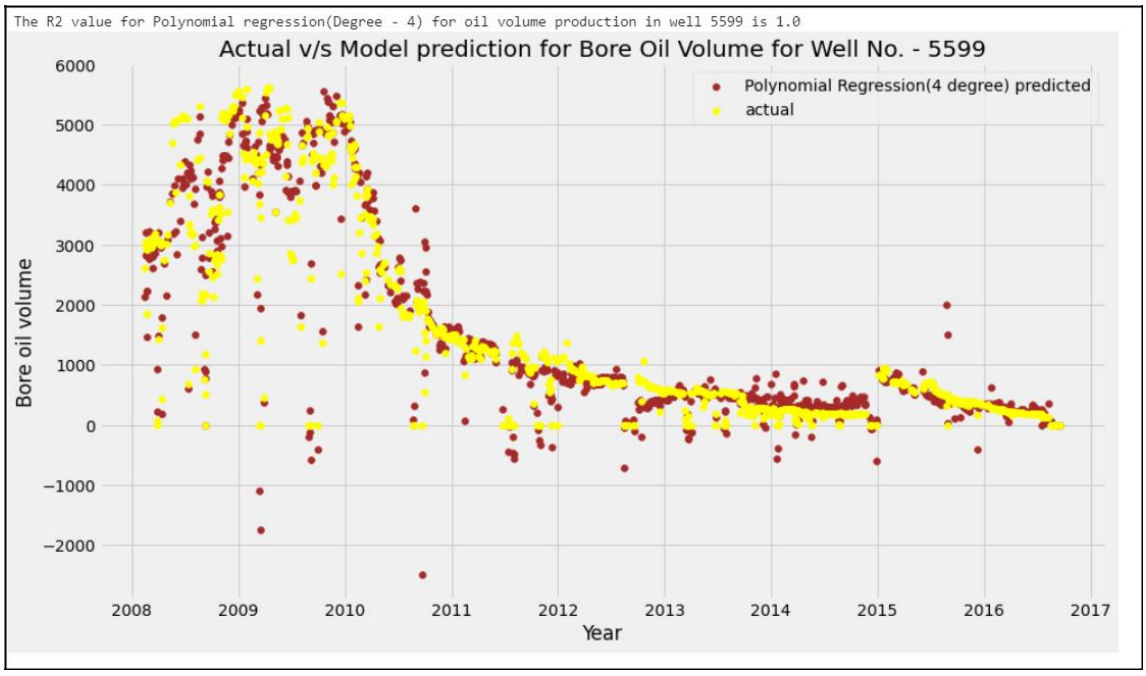
```
The R2 value for Polynomial regression(4th order) for oil volume production is 0.919595641611857
```

```
X_test.groupby(['NPD_WELL_BORE_CODE']).agg({"DATEPRD":"count"})
```

NPD_WELL_BORE_CODE	DATEPRD
5351	910
5599	914
7078	360

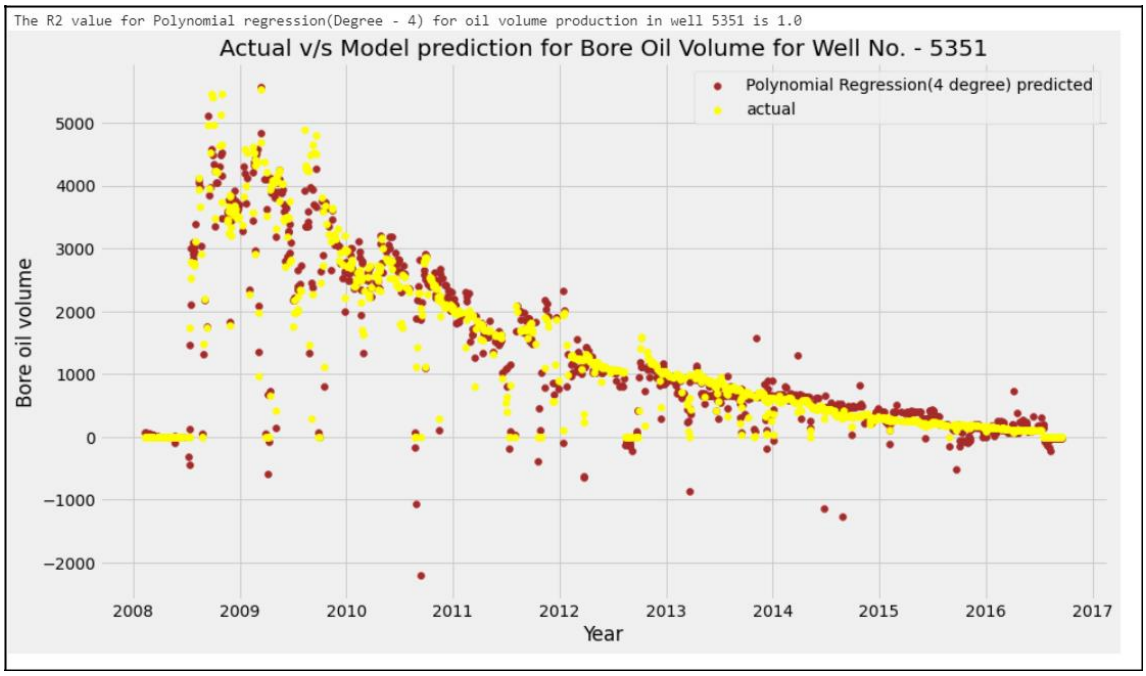
```
#Actual VS Model Prediction plot for Well No 5599

X_test_5599 = X_test[X_test["NPD_WELL_BORE_CODE"] == 5599]

y_test_5599 = y_test[y_test["NPD_WELL_BORE_CODE"] == 5599]
```

```
x_test_5599final = X_test_5599.drop(['DATEPRD',"NPD_WELL_BORE_CODE"],axis =
1)
x_pol_test_5599 = poly_reg.fit_transform(x_test_5599final)
y_poly = lin_reg.predict(x_pol_test_5599)

plt.style.use('fivethirtyeight')
plt.figure(figsize = (14,8))
plt.scatter(X_test_5599["DATEPRD"].tolist(),y_poly,label='Polynomial
Regression(4 degree) predicted', color = 'brown')
plt.scatter(X_test_5599["DATEPRD"].tolist(),y_test_5599['BORE_OIL_VOL'],label
='actual', color = 'yellow')
plt.legend()
plt.xlabel("Year")
plt.ylabel("Bore oil volume")
plt.title('Actual v/s Model prediction for Bore Oil Volume for Well No.
- 5599')

print("The R2 value for Polynomial regression(Degree - 4) for oil volume
production in well 5599 is", lin_reg.score(x_pol_test_5599,y_poly))
```



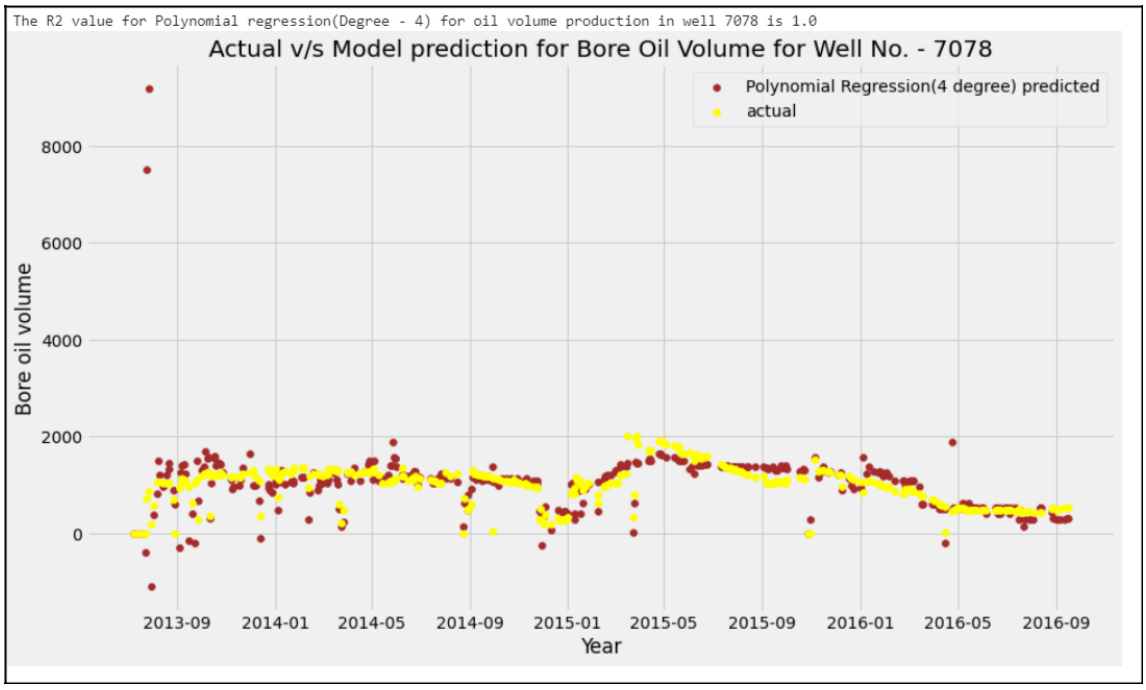

```
#Actual VS Model Prediction plot for Well No 5351

X_test_5351 = X_test[X_test["NPD_WELL_BORE_CODE"] == 5351]

y_test_5351 = y_test[y_test["NPD_WELL_BORE_CODE"] == 5351]
```

```
x_test_5351final = X_test_5351.drop(['DATEPRD',"NPD_WELL_BORE_CODE"],axis =
1)
x_pol_test_5351 = poly_reg.fit_transform(x_test_5351final)
y_poly = lin_reg.predict(x_pol_test_5351)

plt.style.use('fivethirtyeight')
plt.figure(figsize = (14,8))
plt.scatter(X_test_5351["DATEPRD"].tolist(),y_poly,label='Polynomial
Regression(4 degree) predicted', color = 'brown')
plt.scatter(X_test_5351["DATEPRD"].tolist(),y_test_5351['BORE_OIL_VOL'],label
='actual', color = 'yellow')
plt.legend()
plt.xlabel("Year")
plt.ylabel("Bore oil volume")
plt.title('Actual v/s Model prediction for Bore Oil Volume for Well No.
- 5351')

print("The R2 value for Polynomial regression(Degree - 4) for oil volume
production in well 5351 is", lin_reg.score(x_pol_test_5351,y_poly))
```




```
#Actual VS Model Prediction plot for Well No 7078

X_test_7078 = X_test[X_test["NPD_WELL_BORE_CODE"] == 7078]

y_test_7078 = y_test[y_test["NPD_WELL_BORE_CODE"] == 7078]
```

```
x_test_7078final = X_test_7078.drop(['DATEPRD',"NPD_WELL_BORE_CODE"],axis =
1)
x_pol_test_7078 = poly_reg.fit_transform(x_test_7078final)
y_poly = lin_reg.predict(x_pol_test_7078)

plt.style.use('fivethirtyeight')
plt.figure(figsize = (14,8))
plt.scatter(X_test_7078["DATEPRD"].tolist(),y_poly,label='Polynomial
Regression(4 degree) predicted', color = 'brown')
plt.scatter(X_test_7078["DATEPRD"].tolist(),y_test_7078['BORE_OIL_VOL'],label
='actual', color = 'yellow')
plt.legend()
plt.xlabel("Year")
plt.ylabel("Bore oil volume")
plt.title('Actual v/s Model prediction for Bore Oil Volume for Well No.
- 7078')

print("The R2 value for Polynomial regression(Degree - 4) for oil volume
production in well 7078 is", lin_reg.score(x_pol_test_7078,y_poly))
```

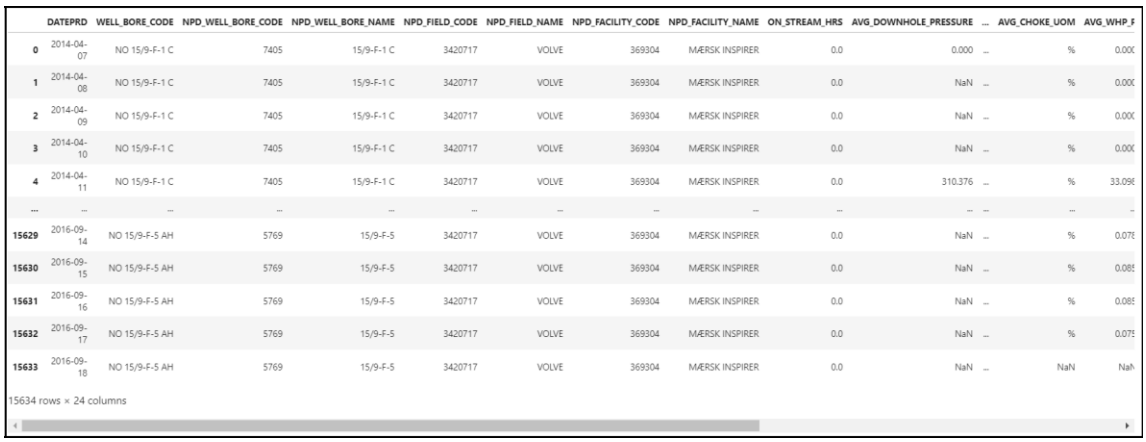


Modelo ARIMA

Link al código:

https://github.com/saviourapi/Saviour/blob/ebf3686711aec22d6b34af4b5f39e866097206fd/notebook/PROJECT(ARIMA-RF).ipynb

```
import lasio
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tabula.io import read_pdf
import welly
import seaborn as sns
from ipywidgets import *
from datetime import date, time, datetime

%config Completer.use_jedi = False

from sklearn import metrics
from  sklearn.metrics  import  mean_absolute_error
from  sklearn.metrics  import  mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import explained_variance_score

from math import sqrt

import seaborn as sns
%matplotlib inline

def parser(x):
    return datetime.strptime(x,"%d/%m/%Y")

campo=pd.read_csv("D:/DARIO/VOLVE FIELD/Production_data/Volve
production data_1.csv",parse_dates=[0], date_parser=parser)

campo
```



	DATEPRD	WELL_BORE_CODE	NPD_WELL_BORE_CODE	NPD_WELL_BORE_NAME	NPD_FIELD_CODE	NPD_FIELD_NAME	NPD_FACILITY_CODE	NPD_FACILITY_NAME	ON_STREAM_HRS	AVG_DOWNHOLE_PRESSURE	...	AVG_CHOKE_UOM	AVG_WHP_P
0	2014-04-07	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	0.000	...	%	0.000
1	2014-04-08	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	NaN	...	%	0.000
2	2014-04-09	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	NaN	...	%	0.000
3	2014-04-10	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	NaN	...	%	0.000
4	2014-04-11	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	310.376	...	%	33.096
...	...	...	...	...	...	...	...	...	...	...	...	...	...
15629	2016-09-14	NO 15/9-F-5 AH	5769	15/9-F-5	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	NaN	...	%	0.078
15630	2016-09-15	NO 15/9-F-5 AH	5769	15/9-F-5	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	NaN	...	%	0.085
15631	2016-09-16	NO 15/9-F-5 AH	5769	15/9-F-5	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	NaN	...	%	0.085
15632	2016-09-17	NO 15/9-F-5 AH	5769	15/9-F-5	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	NaN	...	%	0.075
15633	2016-09-18	NO 15/9-F-5 AH	5769	15/9-F-5	3420717	VOLVE	369304	MÆRSK INSPIRER	0.0	NaN	...	NaN	NaN

15634 rows × 24 columns

```
#elimino columnas con información irrelevante para el procesamiento de la data
del campo["WELL_BORE_CODE"]
del campo["NPD_WELL_BORE_CODE"]
del campo["NPD_FIELD_CODE"]
del campo["NPD_FIELD_NAME"]
del campo["NPD_FACILITY_CODE"]
del campo["NPD_FACILITY_NAME"]
del campo["AVG_CHOKE_UOM"]

#filtro por los pozos que trabajaron más de cero horas en un día. data_tratada=campo[campo.ON_STREAM_HRS>0]

#TRATAMIENTO DE LA DATA POR CADA POZO
#POZO 1

encabezados=["ON_STREAM_HRS","AVG_DOWNHOLE_PRESSURE","AVG_DOWNHOLE_TEMPERATURE","AVG_DP_TUBING","AVG_ANNULUS_PRESS","AVG_CHOKE_SIZE_P","AVG_WHP_P","AVG_WHT_P","DP_CHOKE_SIZE","BORE_OIL_VOL","BORE_GAS_VOL","BORE_WAT_VOL"]

df_well_1
```

	DATEPRD	NPD_WELL_BORE_NAME	ON_STREAM_HRS	AVG_DOWNHOLE_PRESSURE	AVG_DOWNHOLE_TEMPERATURE	AVG_DP_TUBING	AVG_ANNULUS_PRESS	AVG_CHOKE_SIZE_P	AVG_WHP_P	AVG_WHT_P	DP_CHOKE_SIZE	BORE_OIL_VOL	BORE_
14	2014-04-21	15/9-F-1 C	11.50	301.376	102.676	204.795	0.0	20.98975	96.580	19.197	69.776	0.00	
15	2014-04-22	15/9-F-1 C	24.00	289.421	106.353	182.059	0.0	43.34345	107.362	37.939	78.935	631.47	
16	2014-04-23	15/9-F-1 C	24.00	270.240	107.644	171.053	NaN	47.16752	99.187	60.757	70.627	1166.46	
17	2014-04-24	15/9-F-1 C	24.00	262.843	107.869	168.242	NaN	47.73231	94.601	63.047	66.049	1549.81	
18	2014-04-25	15/9-F-1 C	24.00	255.527	107.971	165.539	NaN	48.53377	89.988	64.547	61.405	1248.70	
...	...	...	...	...	...	...	...	...	...	...	...	...	
726	2016-04-02	15/9-F-1 C	24.00	223.012	108.058	196.629	NaN	54.68774	26.383	69.559	10.681	284.72	
727	2016-04-03	15/9-F-1 C	24.00	221.813	108.044	196.133	NaN	54.72411	25.681	72.502	9.910	280.17	
728	2016-04-04	15/9-F-1 C	24.00	220.780	108.042	195.665	NaN	55.06795	25.115	71.750	9.197	281.93	
729	2016-04-05	15/9-F-1 C	24.00	218.752	108.078	193.411	NaN	55.99813	25.341	72.218	9.101	317.38	
730	2016-04-06	15/9-F-1 C	20.82	218.438	107.857	192.830	NaN	45.14344	25.608	66.568	9.830	208.00	

438 rows × 16 columns

```
fechas=list(df_well_1.DATEPRD)
```

```
df_new=df_well_1.set_index("DATEPRD")
```

```
df_new
```

DATEPRD	NPD_WELL_BORE_NAME	ON_STREAM_HRS	AVG_DOWNHOLE_PRESSURE	AVG_DOWNHOLE_TEMPERATURE	AVG_DP_TUBING	AVG_ANNULUS_PRESS	AVG_CHOKE_SIZE_P	AVG_WHP_P	AVG_WHT_P	DP_CHOKE_SIZE	BORE_OIL_VOL	BORE_GAS_V
2014-04-21	15/9-F-1 C	11.50	301.376	102.676	204.795	0.0	20.98975	96.580	19.197	69.776	0.00	
2014-04-22	15/9-F-1 C	24.00	289.421	106.353	182.059	0.0	43.34345	107.362	37.939	78.935	631.47	9043
2014-04-23	15/9-F-1 C	24.00	270.240	107.644	171.053	NaN	47.16752	99.187	60.757	70.627	1166.46	16572
2014-04-24	15/9-F-1 C	24.00	262.843	107.869	168.242	NaN	47.73231	94.601	63.047	66.049	1549.81	22170
2014-04-25	15/9-F-1 C	24.00	255.527	107.971	165.539	NaN	48.53377	89.988	64.547	61.405	1248.70	17806
...	...	...	...	...	...	...	...	...	...	...	...	
2016-04-02	15/9-F-1 C	24.00	223.012	108.058	196.629	NaN	54.68774	26.383	69.559	10.681	284.72	4688
2016-04-03	15/9-F-1 C	24.00	221.813	108.044	196.133	NaN	54.72411	25.681	72.502	9.910	280.17	4584
2016-04-04	15/9-F-1 C	24.00	220.780	108.042	195.665	NaN	55.06795	25.115	71.750	9.197	281.93	4580
2016-04-05	15/9-F-1 C	24.00	218.752	108.078	193.411	NaN	55.99813	25.341	72.218	9.101	317.38	5199
2016-04-06	15/9-F-1 C	20.82	218.438	107.857	192.830	NaN	45.14344	25.608	66.568	9.830	208.00	3499

438 rows × 15 columns

```
parametros=df_well_1.iloc[:,2:14]
```

```
from sklearn.impute import SimpleImputer
```

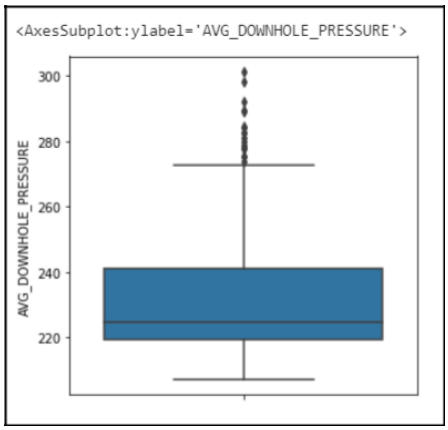
```
#Reemplazo los valores nulos por la media de la
columna imp=SimpleImputer(strategy="mean")
```

```
datos=imp.fit_transform(parametros)
```

```
#Verificando la cantidaad de valores nulos en la data
df_sin_nan=pd.DataFrame(datos, columns=encabezados)
df_sin_nan.isnull().values.sum()
```

```
0
```

```
#Limpieza de outliers de la columna AVG_DOWNHOLE_PRESSURE

#Grafico de cajas y bigotes de la columna de
AVG_DOWNHOLE_PRESSURE figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'AVG_DOWNHOLE_PRESSURE', data=df_sin_nan)
```




```
#Obtención de los estadísticos y otros
min_value=df_sin_nan.AVG_DOWNHOLE_PRESSURE.min()
print("min: ",min_value)

max_value=df_sin_nan.AVG_DOWNHOLE_PRESSURE.max()
print("max: ",max_value)

median_value=df_sin_nan.AVG_DOWNHOLE_PRESSURE.median()
print("median: ",median_value)

Q1=df_sin_nan["AVG_DOWNHOLE_PRESSURE"].quantile(0.25)
print("Q1: ",Q1)

Q3=df_sin_nan["AVG_DOWNHOLE_PRESSURE"].quantile(0.75)
print("Q3: ",Q3)

IQR=Q3-Q1
print("IQR: ",IQR)
```

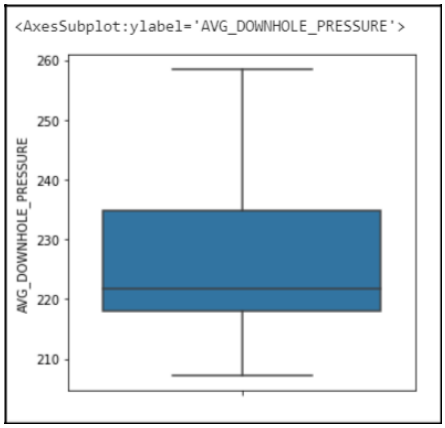
```
lower_whisker_p1c1=Q1-1.5*IQR
print("lower whisker: ",lower_whisker_p1c1)

upper_whisker_p1c1=Q3+0.8*IQR
print("upper wiscker: ",upper_whisker_p1c1)
```

```
min:  207.219
max:  301.376
median:  224.6395
Q1:  219.51
Q3:  241.23975000000002
IQR:  21.729750000000024
lower whisker:  186.91537499999995
upper wiscker:  258.62355
```

```
#Forma del df sin outliers
df_final_downpres=df_sin_nan[df_sin_nan["AVG_DOWNHOLE_PRESSURE"]<upper_whiske
r_p1c1]
df_final_downpres.shape
```

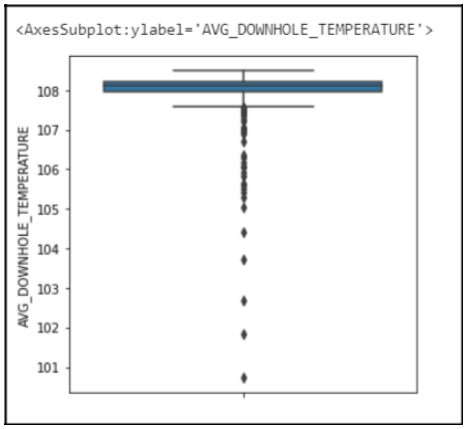
```
(396, 12)
```

```
#Verificamos de manera gráfica la limpieza
figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'AVG_DOWNHOLE_PRESSURE', data=df_final_downpres)
```




```
#Aplico las condiciones de los límites y reemplazo estos valore spor la
mediana
df_modif_col1=df_sin_nan.AVG_DOWNHOLE_PRESSURE.where(df_sin_nan.AVG_DOWNHOLE
_
PRESSURE<upper_whisker_p1c1,df_final_downpres.AVG_DOWNHOLE_PRESSURE.median()
)
```

```
#limpieza de outliers de la columna AVG_DOWNHOLE_TEMPERATURE

figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'AVG_DOWNHOLE_TEMPERATURE', data=df_sin_nan)
```



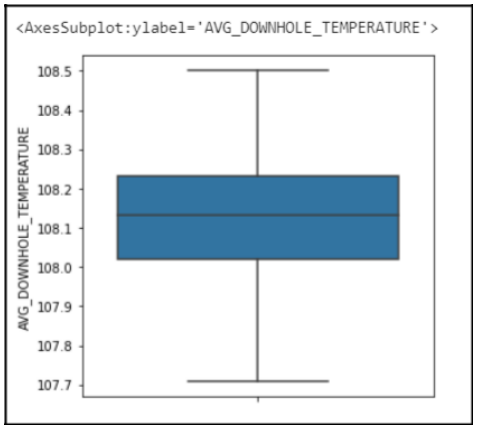

```
Q1=df_sin_nan["AVG_DOWNHOLE_TEMPERATURE"].quantile(0.25)
Q3=df_sin_nan["AVG_DOWNHOLE_TEMPERATURE"].quantile(0.75)
IQR=Q3-Q1
lower_whisker_p1c2=Q1-1*IQR
upper_whisker_p1c2=Q3+1*IQR
print(lower_whisker_p1c2)
print(upper_whisker_p1c2)
```

```
107.70749999999998
108.46875000000001
```

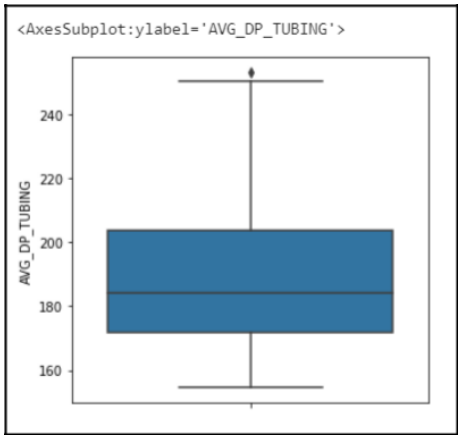
```
df_final_downtemp=df_sin_nan[df_sin_nan["AVG_DOWNHOLE_TEMPERATURE"]>lower_whi
sker_p1c2]
df_final_downtemp.shape
```

```
(391, 12)
```

```
figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'AVG_DOWNHOLE_TEMPERATURE', data=df_final_downtemp)
```




```
df_modif_col2=df_sin_nan.AVG_DOWNHOLE_TEMPERATURE.where(df_sin_nan.AVG_DOWNH
O
LE_TEMPERATURE>lower_whisker_p1c2,df_final_downtemp.AVG_DOWNHOLE_TEMPERATURE
. mean())

#limpieza de outliers de la columna AVG_DP_TUBING

figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'AVG_DP_TUBING', data=df_sin_nan)
```



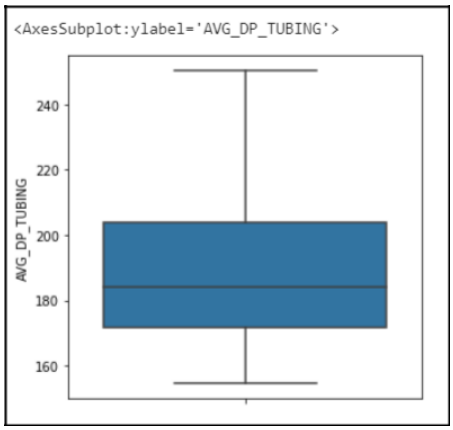

```
Q1=df_sin_nan["AVG_DP_TUBING"].quantile(0.25)
Q3=df_sin_nan["AVG_DP_TUBING"].quantile(0.75)
IQR=Q3-Q1
lower_whisker_p1c3=Q1-1.5*IQR
upper_whisker_p1c3=Q3+1.5*IQR
print(lower_whisker_p1c3)
print(upper_whisker_p1c3)
```

```
123.73574999999997
252.04975000000005
```

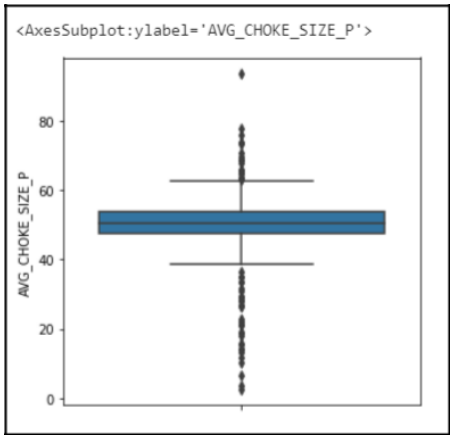
```
df_final_dptubing=df_sin_nan[df_sin_nan["AVG_DP_TUBING"]<upper_whisker_p1c3]
df_final_dptubing.shape
```

```
(437, 12)
```

```
figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'AVG_DP_TUBING', data=df_final_dptubing)
```




```
df_modif_col3=df_sin_nan.AVG_DP_TUBING.where(df_sin_nan.AVG_DP_TUBING<upper_
w hisker_p1c3,df_final_dptubing.AVG_DP_TUBING.mean())

#limpieza de outliers de la columna AVG_CHOKE_SIZE_P

figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'AVG_CHOKE_SIZE_P', data=df_sin_nan)
```




```
Q1=df_sin_nan["AVG_CHOKE_SIZE_P"].quantile(0.25)
Q3=df_sin_nan["AVG_CHOKE_SIZE_P"].quantile(0.75)
IQR=Q3-Q1
lower_whisker_p1c5=Q1-1.1*IQR
upper_whisker_p1c5=Q3+1.1*IQR
print(lower_whisker_p1c5)
print(upper_whisker_p1c5)
```

```
40.97033025
60.44968225
```

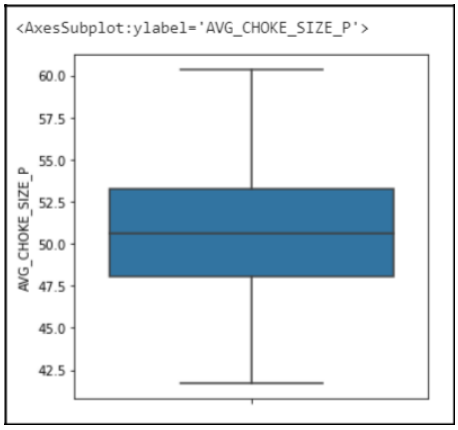
```
#aplicación de la condición de no sobrepasar el límite superior
df_chokesizep=df_sin_nan[df_sin_nan["AVG_CHOKE_SIZE_P"]<upper_whisker_p1c5]
df_chokesizep.shape
```

```
(407, 12)
```

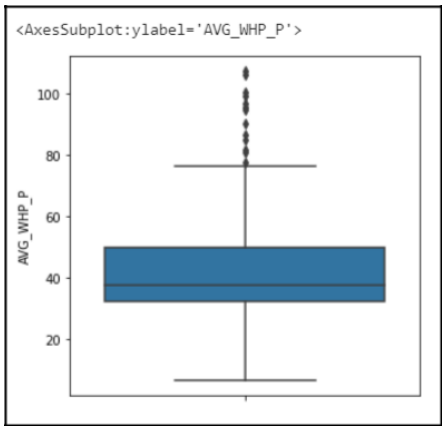
```
#apliación de la condición de no sobrepasar el límite inferior
df_final_chokesizep=df_chokesizep[df_chokesizep["AVG_CHOKE_SIZE_P"]>lower_whi
sker_p1c5]
df_final_chokesizep.shape
```

```
(370, 12)
```

```
figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'AVG_CHOKE_SIZE_P', data=df_final_chokesizep)
```



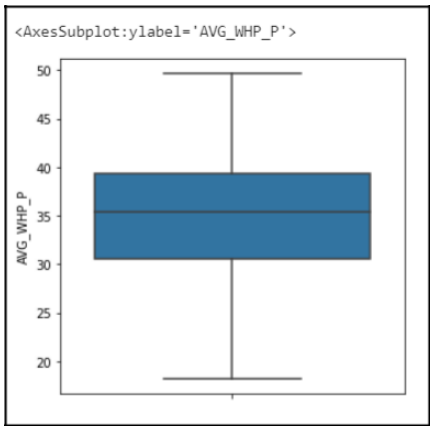

```
df_modif_col51=df_sin_nan.AVG_CHOKE_SIZE_P.where(df_sin_nan.AVG_CHOKE_SIZE_P
>
lower_whisker_p1c5,df_final_chokesizep.AVG_CHOKE_SIZE_P.mean())
df_modif_col5=df_modif_col51.where(df_modif_col51<upper_whisker_p1c5,df_fina
l
_chokesizep.AVG_CHOKE_SIZE_P.mean())

#limpieza de outliers de la columna AVG_WHP_P

figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'AVG_WHP_P', data=df_sin_nan)
```



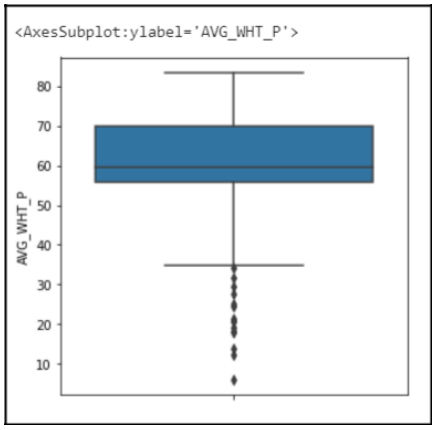

```
Q1=df_sin_nan["AVG_WHP_P"].quantile(0.25)
Q3=df_sin_nan["AVG_WHP_P"].quantile(0.75)
IQR=Q3-Q1
lower_whisker_p1c6=Q1-0.8*IQR
upper_whisker_p1c6=Q3
print(lower_whisker_p1c6)
print(upper_whisker_p1c6)
```

```
18.096650000000004
49.93325
```

```
df_whpP=df_sin_nan[df_sin_nan["AVG_WHP_P"]<upper_whisker_p1c6]

df_final_whpP=df_whpP[df_whpP["AVG_WHP_P"]>lower_whisker_p1c6]

figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'AVG_WHP_P', data=df_final_whpP)
```




```
df_modif_col61=df_sin_nan.AVG_WHP_P.where(df_sin_nan.AVG_WHP_P>lower_whisker
_
p1c6,df_final_whpP.AVG_WHP_P.mean())
df_modif_col6=df_modif_col61.where(df_modif_col61<upper_whisker_p1c6,df_fina
l
_whpP.AVG_WHP_P.mean())

#limpieza de outliers de la columna AVG_WHT_P

figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'AVG_WHT_P', data=df_sin_nan)
```



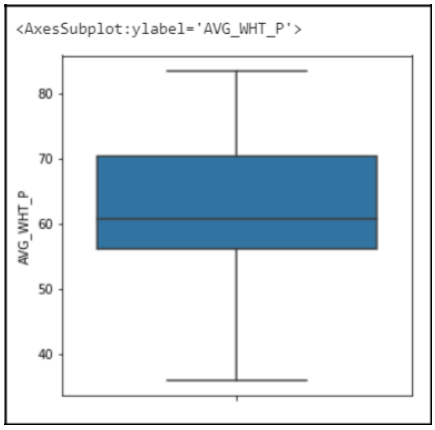

```
Q1=df_sin_nan["AVG_WHT_P"].quantile(0.25)
Q3=df_sin_nan["AVG_WHT_P"].quantile(0.75)
IQR=Q3-Q1
lower_whisker_p1c7=Q1-1.4*IQR
upper_whisker_p1c7=Q3+1.4*IQR
print(lower_whisker_p1c7)
print(upper_whisker_p1c7)
```

```
35.831450000000004
89.7278
```

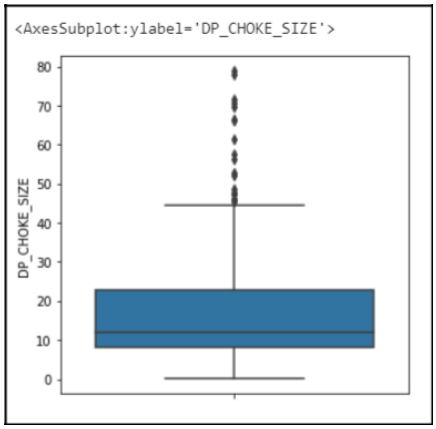
```
df_whtP=df_sin_nan[df_sin_nan["AVG_WHT_P"]>lower_whisker_p1c7]
df_whtP.shape
```

```
(421, 12)
```

```
figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'AVG_WHT_P', data=df_whtP)
```




```
df_modif_col7=df_sin_nan.AVG_WHT_P.where(df_sin_nan.AVG_WHT_P>lower_whisker_
p 1c7,df_whtP.AVG_WHT_P.mean())

#limpieza de outliers de la columna DP_CHOKE_SIZE

figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'DP_CHOKE_SIZE', data=df_sin_nan)
```



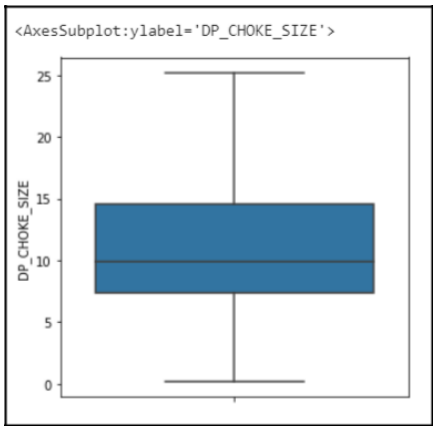

```
Q1=df_sin_nan["DP_CHOKE_SIZE"].quantile(0.25)
Q3=df_sin_nan["DP_CHOKE_SIZE"].quantile(0.75)
IQR=Q3-Q1
lower_whisker_p1c8=Q1-0.19*IQR
upper_whisker_p1c8=Q3+0.19*IQR
print(lower_whisker_p1c8)
print(upper_whisker_p1c8)
```

```
5.233945
25.671055
```

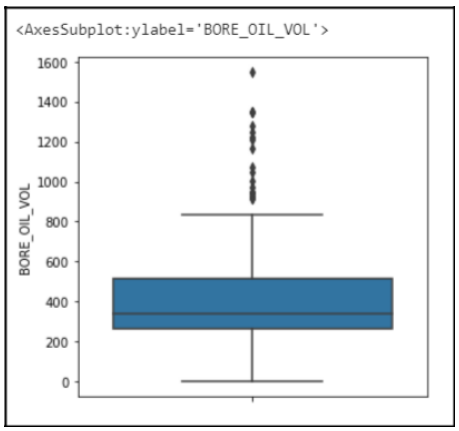
```
df_choke_size=df_sin_nan[df_sin_nan["DP_CHOKE_SIZE"]<upper_whisker_p1c8]

df_choke_size.shape
```

```
(340, 12)
```

```
figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'DP_CHOKE_SIZE', data=df_choke_size)
```




```
df_modif_col8=df_sin_nan.DP_CHOKE_SIZE.where(df_sin_nan.DP_CHOKE_SIZE<upper_
w hisker_p1c8,df_choke_size.DP_CHOKE_SIZE.mean())

#limpieza de outliers de la columna BORE_OIL_VOL

figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'BORE_OIL_VOL', data=df_sin_nan)
```



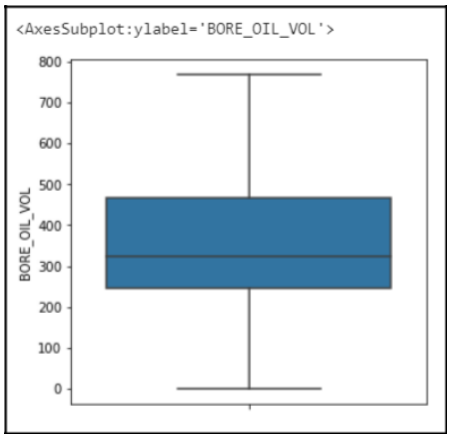
```
Q1=df_sin_nan["BORE_OIL_VOL"].quantile(0.25)
Q3=df_sin_nan["BORE_OIL_VOL"].quantile(0.75)
IQR=Q3-Q1
lower_whisker_p1c9=Q1-1*IQR
upper_whisker_p1c9=Q3+1*IQR
print(lower_whisker_p1c9)
print(upper_whisker_p1c9)
```

```
5.4099999999998545
769.1125000000002
```

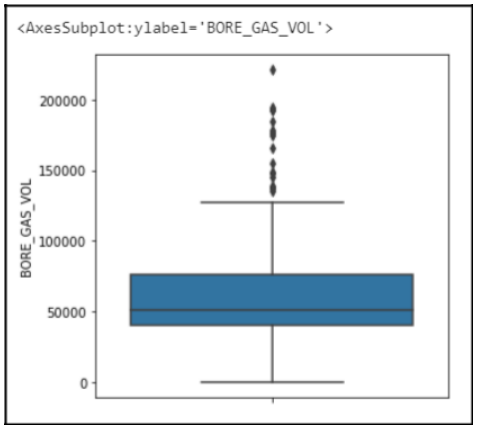
```
df_oil_vol=df_sin_nan[df_sin_nan["BORE_OIL_VOL"]<upper_whisker_p1c9]
df_oil_vol.shape
```

```
(406, 12)
```

```
figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'BORE_OIL_VOL', data=df_oil_vol)
```




```
df_modif_col9=df_sin_nan.BORE_OIL_VOL.where(df_sin_nan.BORE_OIL_VOL<upper_wh
i sker_p1c9,df_oil_vol.BORE_OIL_VOL.mean())
#limpieza de outliers de la columna BORE_GAS_VOL
figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'BORE_GAS_VOL', data=df_sin_nan)
```



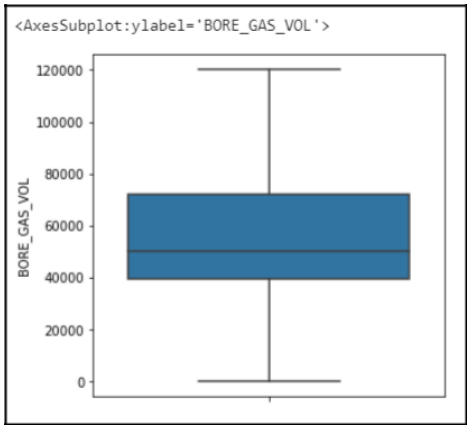

```
Q1=df_sin_nan["BORE_GAS_VOL"].quantile(0.25)
Q3=df_sin_nan["BORE_GAS_VOL"].quantile(0.75)
IQR=Q3-Q1
lower_whisker_p1c10=Q1-1.1*IQR
upper_whisker_p1c10=Q3+1.4*IQR
print(lower_whisker_p1c10)
print(upper_whisker_p1c10)
```

```
324.46524999999383
126669.5565
```

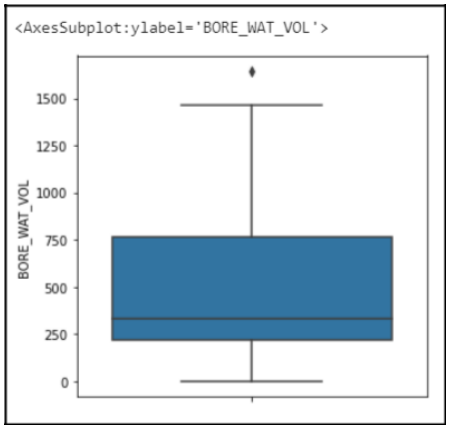
```
df_gas_vol=df_sin_nan[df_sin_nan["BORE_GAS_VOL"]<upper_whisker_p1c10]
df_gas_vol.shape
```

```
(420, 12)
```

```
figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'BORE_GAS_VOL', data=df_gas_vol)
```




```
df_modif_col10=df_sin_nan.BORE_GAS_VOL.where(df_sin_nan.BORE_GAS_VOL<upper_w
h isker_p1c10,df_gas_vol.BORE_GAS_VOL.mean())

#limpieza de outliers de la columna BORE_WAT_VOL

figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'BORE_WAT_VOL', data=df_sin_nan)
```



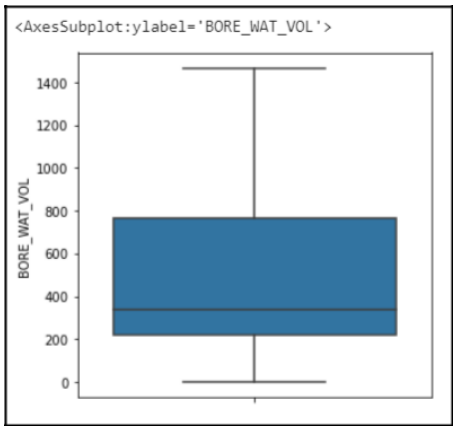

```
Q1=df_sin_nan["BORE_WAT_VOL"].quantile(0.25)
Q3=df_sin_nan["BORE_WAT_VOL"].quantile(0.75)
IQR=Q3-Q1
lower_whisker_p1c11=Q1-1.1*IQR
upper_whisker_p1c11=Q3+1.4*IQR
print(lower_whisker_p1c11)
print(upper_whisker_p1c11)
```

```
-382.3537500000001
1528.7775000000001
```

```
df_wat_vol=df_sin_nan[df_sin_nan["BORE_WAT_VOL"]<upper_whisker_p1c11]

df_wat_vol.shape
```

```
(437, 12)
```

```
figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y = 'BORE_WAT_VOL', data=df_wat_vol)
```




```
df_modif_col11=df_sin_nan.BORE_WAT_VOL.where(df_sin_nan.BORE_WAT_VOL<upper_w
h isker_p1c11,df_wat_vol.BORE_WAT_VOL.mean())

#CONSTRUCCIÓN DEL DF FINAL DEL POZO 1 CON LA DATA TRATADA

columna_p1c0=list(df_sin_nan.ON_STREAM_HRS)
columna_p1c1=list(df_modif_col1)
columna_p1c2=list(df_modif_col2)
columna_p1c3=list(df_modif_col3)
columna_p1c4=list(df_sin_nan.AVG_ANNULUS_PRESS)
columna_p1c5=list(df_modif_col5)
columna_p1c6=list(df_modif_col6)
columna_p1c7=list(df_modif_col7)
columna_p1c8=list(df_modif_col8)
columna_p1c9=list(df_modif_col9)
columna_p1c10=list(df_modif_col10)
columna_p1c11=list(df_modif_col11)
columna_dates=list(df_well_1.DATEPRD)
```

```
dicc_well_1={"DATEPRD":columna_dates,"ON_STREAM_HRS":columna_p1c0,"AVG_DOWNHOLE_PRESSURE":columna_p1c1,"AVG_DOWNHOLE_TEMPERATURE":columna_p1c2,"AVG_DP_TUBING":columna_p1c3,"AVG_ANNULUS_PRESS":columna_p1c4,"AVG_CHOKE_SIZE_P":columna_p1c5,"AVG_WHP_P":columna_p1c6,"AVG_WHT_P":columna_p1c7,"AVG_CHOKE_SIZE":columna_p1c8,"BORE_OIL_VOL":columna_p1c9,"BORE_GAS_VOL":columna_p1c10,"BORE_WAT_VOL":columna_p1c11}

df_new_well_1=pd.DataFrame(dicc_well_1)
df_new_well_1
```

	DATEPRD	ON_STREAM_HRS	AVG_DOWNHOLE_PRESSURE	AVG_DOWNHOLE_TEMPERATURE	AVG_DP_TUBING	AVG_ANNULUS_PRESS	AVG_CHOKE_SIZE_P	AVG_WHP_P	AVG_WHT_P	AVG_CHOKE_SIZE	BORE_OIL_VOL	BORE_GAS_VOL	BORE_WAT_VC
0	2014-04-21	11.50	221.838	108.124327	204.795	0.0	50.987051	35.27676	62.420736	11.025647	0.000000	0.000000	0.(
1	2014-04-22	24.00	221.838	108.124327	182.059	0.0	43.343450	35.27676	37.939000	11.025647	631.470000	90439.090000	0.(
2	2014-04-23	24.00	221.838	108.124327	171.053	0.0	47.167520	35.27676	60.757000	11.025647	359.481355	55918.232095	0.(
3	2014-04-24	24.00	221.838	107.869000	168.242	0.0	47.732310	35.27676	63.047000	11.025647	359.481355	55918.232095	0.(
4	2014-04-25	24.00	255.527	107.971000	165.539	0.0	48.533770	35.27676	64.547000	11.025647	359.481355	55918.232095	0.(
...	...	...	...	...	...	...	...	...	...	...	...	...	
433	2016-04-02	24.00	223.012	108.058000	196.629	0.0	54.687740	26.38300	69.559000	10.681000	284.720000	46880.140000	766.
434	2016-04-03	24.00	221.813	108.044000	196.133	0.0	54.724110	25.68100	72.502000	9.910000	280.170000	45842.820000	748.
435	2016-04-04	24.00	220.780	108.042000	195.665	0.0	55.067950	25.11500	71.750000	9.197000	281.930000	45800.830000	797.
436	2016-04-05	24.00	218.752	108.078000	193.411	0.0	55.998130	25.34100	72.218000	9.101000	317.380000	51990.650000	792.
437	2016-04-06	20.82	218.438	107.857000	192.830	0.0	45.143440	25.60800	66.568000	9.830000	208.000000	34998.300000	542.

438 rows × 13 columns

```
#Los pasos anteriores realizados para el pozo 1, se aplican nuevamente para el resto de los pozos productores.

#Para observar el procesamiento de la data del resto de pozos, puedes acceder al notebook en Github.

#Tratamiento de la data de los pozos inyectores

df_well_7
```

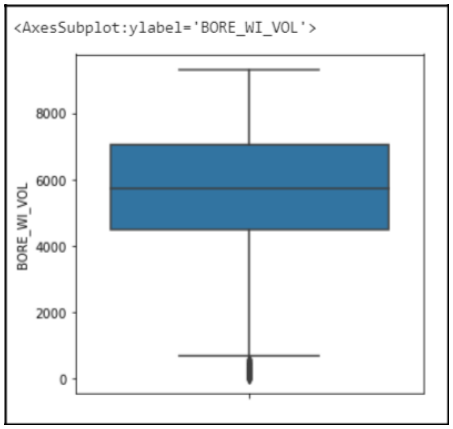
	DATEPRD	NPD_WELL_BORE_NAME	ON_STREAM_HRS	BORE_WI_VOL	FLOW_KIND	WELL_TYPE
9236	2008-04-23	15/9-F-4	10.00	166.08	injection	WI
9237	2008-04-24	15/9-F-4	24.00	3565.07	injection	WI
9238	2008-04-25	15/9-F-4	24.00	6916.92	injection	WI
9239	2008-04-26	15/9-F-4	18.65	6617.82	injection	WI
9240	2008-04-27	15/9-F-4	15.96	4507.00	injection	WI
...	...	...	...	...	...	...
12300	2016-09-12	15/9-F-4	10.94	1971.78	injection	WI
12302	2016-09-14	15/9-F-4	9.56	1810.68	injection	WI
12303	2016-09-15	15/9-F-4	24.00	4469.38	injection	WI
12304	2016-09-16	15/9-F-4	24.00	4474.81	injection	WI
12305	2016-09-17	15/9-F-4	7.61	1419.01	injection	WI

2830 rows × 6 columns

```
df_well_7.isnull().values.sum()
```

```
0
```

```
#limpieza de outliers de la columna BORE_WI_VOL

figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y="BORE_WI_VOL",data=df_well_7)
```




```
Q1=df_well_7["BORE_WI_VOL"].quantile(0.25)
Q3=df_well_7["BORE_WI_VOL"].quantile(0.75)
IQR=Q3-Q1
lower_whisker_p7=Q1-1.5*IQR
upper_whisker_p7=Q3+1.5*IQR
print(lower_whisker_p7)
print(upper_whisker_p7)
```

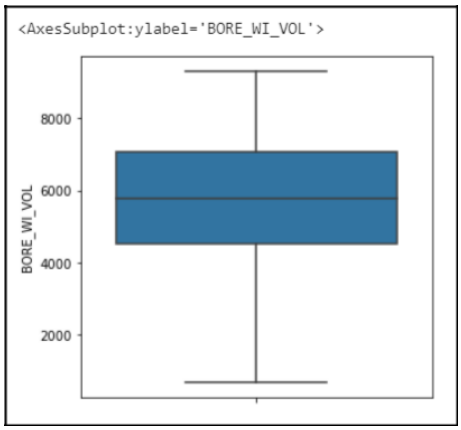
```
621.1474999999996
10922.6675
```

```
df_wi_p7=df_well_7[df_well_7["BORE_WI_VOL"]>lower_whisker_p7]
```

```
df_wi_p7.shape
```

```
(2803, 6)
```

```
figuresizes = (5,5)
plt.figure(figsize=figuresizes)
sns.boxplot(y="BORE_WI_VOL",data=df_wi_p7)
```



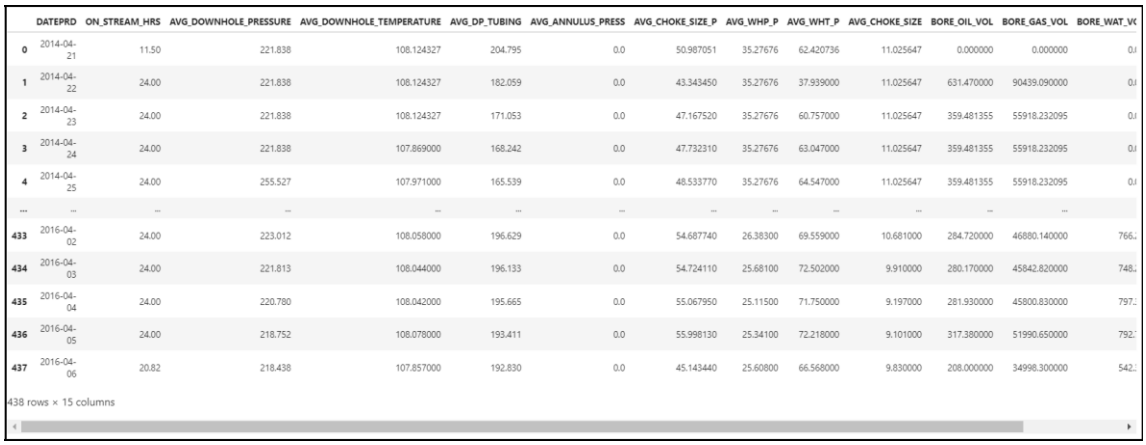

```
#Aplico las condiciones de los límites y reemplazo estos valores por la media
df_modif_pozo7=df_well_7.BORE_WI_VOL.where(df_well_7.BORE_WI_VOL>lower_whiske
r_p7,df_wi_p7.BORE_WI_VOL.mean())

#CONSTRUCCIÓN DEL DF FINAL DEL POZO 7 CON LA DATA TRATADA

columna_p7c0=list(df_well_7.DATEPRD)
columna_p7c1=list(df_modif_pozo7)

dicc_well_7={"DATEPRD":columna_p7c0,"BORE_WI_VOL_F4":columna_p7c1}

df_new_well_7=pd.DataFrame(dicc_well_7)
df_new_well_7
```

	DATEPRD	BORE_WI_VOL_F4
0	2008-04-23	5789.616882
1	2008-04-24	3565.070000
2	2008-04-25	6916.920000
3	2008-04-26	6617.820000
4	2008-04-27	4507.000000
...	...	...
2825	2016-09-12	1971.780000
2826	2016-09-14	1810.680000
2827	2016-09-15	4469.380000
2828	2016-09-16	4474.810000
2829	2016-09-17	1419.010000

2830 rows × 2 columns

```
#almaceno en una lista los diccionarios de fecha y volumen de agua
inyectada list_w7=[]
for i in range(len(columna_p7c0)):
    clave=columna_p7c0[i]
    valor=columna_p7c1[i]
    list_w7.append({clave:valor})

#Se aplica el mismo procedimiento para el otro pozo inyector

#Completamos los dataframe de cada uno de los 5 pozos productores

#obtengo los valores de agua inyectada por cada pozo inyector en cada
fecha de producción del pozo 1
columna_wi_F4=[]
columna_wi_F5=[]
for x in range(len(list(df_new_well_1.DATEPRD))):
    fechas=list(df_new_well_1.DATEPRD)
    fecha_interes=fechas[x]
    vol1=0
    vol2=0
    for j in range(len(list_w7)):
        diccj=list_w7[j]
        clavej=list(diccj.keys())[0]
        valorj=list(diccj.values())[0]
        if fecha_interes==clavej:
            vol1=valorj
    for k in range(len(list_w8)):
        dicck=list_w8[k]
        clavek=list(dicck.keys())[0]
```

```
        valork=list(dicck.values())[0]
        if fecha_interes==clavek:
            vol2=valork
    columna_wi_F4.append(vol1)
    columna_wi_F5.append(vol2)

df_final_w1=df_new_well_1.assign(WI_F4=columna_wi_F4, WI_F5=columna_wi_F5)

df_final_w1
```

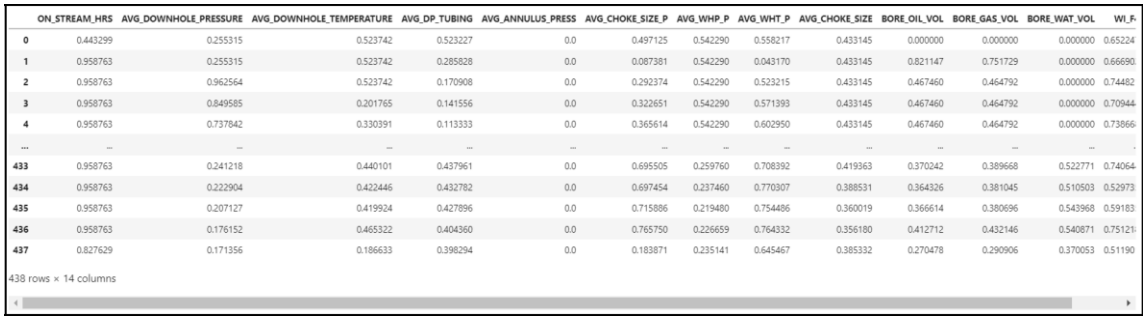


	DATEPRD	ON_STREAM_HRS	AVG_DOWNHOLE_PRESSURE	AVG_DOWNHOLE_TEMPERATURE	AVG_DP_TUBING	AVG_ANNULUS_PRESS	AVG_CHOKE_SIZE_P	AVG_WHP_P	AVG_WHT_P	AVG_CHOKE_SIZE	BORE_OIL_VOL	BORE_GAS_VOL	BORE_WAT_VC
0	2014-04-21	11.50	221.838	108.124327	204.795	0.0	50.987051	35.27676	62.420736	11.025647	0.000000	0.000000	0.(
1	2014-04-22	24.00	221.838	108.124327	182.059	0.0	43.343450	35.27676	37.939000	11.025647	631.470000	90439.090000	0.(
2	2014-04-23	24.00	221.838	108.124327	171.053	0.0	47.167520	35.27676	60.757000	11.025647	359.481355	55918.232095	0.(
3	2014-04-24	24.00	221.838	107.869000	168.242	0.0	47.732310	35.27676	63.047000	11.025647	359.481355	55918.232095	0.(
4	2014-04-25	24.00	255.527	107.971000	165.539	0.0	48.533770	35.27676	64.547000	11.025647	359.481355	55918.232095	0.(
...	...	...	...	...	...	...	...	...	...	...	...	...	
433	2016-04-02	24.00	223.012	108.058000	196.629	0.0	54.687740	26.38300	69.559000	10.681000	284.720000	46880.140000	766.(
434	2016-04-03	24.00	221.813	108.044000	196.133	0.0	54.724110	25.68100	72.502000	9.910000	280.170000	45842.820000	748.(
435	2016-04-04	24.00	220.780	108.042000	195.665	0.0	55.067950	25.11500	71.750000	9.197000	281.930000	45800.830000	797.(
436	2016-04-05	24.00	218.752	108.078000	193.411	0.0	55.998130	25.34100	72.218000	9.101000	317.380000	51990.650000	792.(
437	2016-04-06	20.82	218.438	107.857000	192.830	0.0	45.143440	25.60800	66.568000	9.830000	208.000000	34998.300000	542.(

438 rows × 15 columns

Aplicación del algoritmo

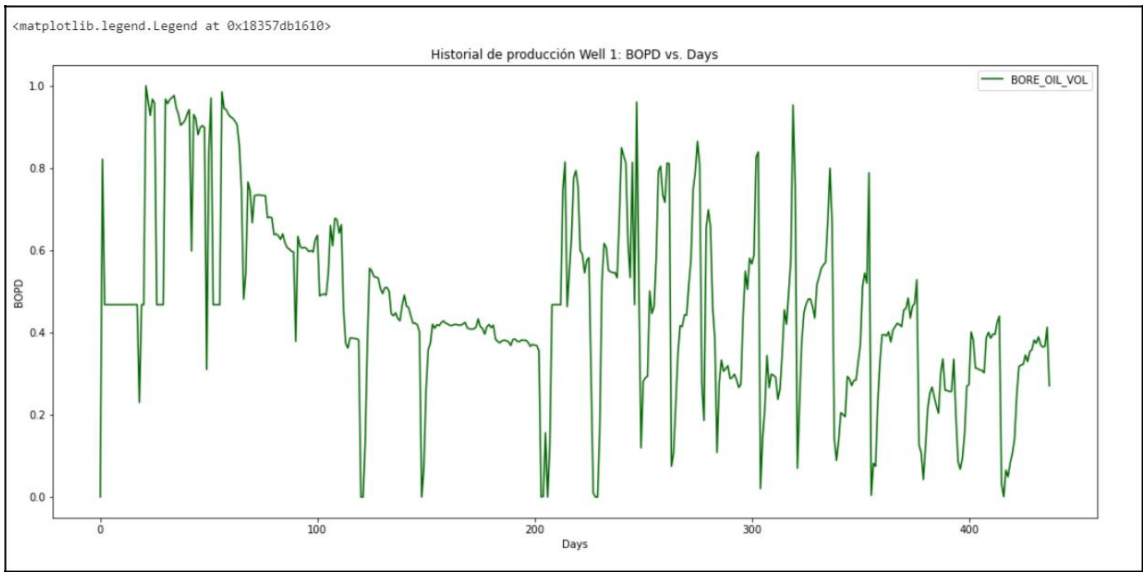
```
#POZO 1

#escalamiento de la data
df_prescaled=df_final_w1.drop("DATEPRD",axis=1)
scaler_all_data=MinMaxScaler()
data_scaled = scaler_all_data.fit_transform(df_prescaled)
well1_scaled=pd.DataFrame(data_scaled,
columns=df_prescaled.columns) well1_scaled
```



	ON_STREAM_HRS	AVG_DOWNHOLE_PRESSURE	AVG_DOWNHOLE_TEMPERATURE	AVG_DP_TUBING	AVG_ANNULUS_PRESS	AVG_CHOKE_SIZE_P	AVG_WHP_P	AVG_WHT_P	AVG_CHOKE_SIZE	BORE_OIL_VOL	BORE_GAS_VOL	BORE_WAT_VOL	WI_F(
0	0.443299	0.255315	0.523742	0.523227	0.0	0.497125	0.542290	0.558217	0.433145	0.000000	0.000000	0.000000	0.65224
1	0.958763	0.255315	0.523742	0.285828	0.0	0.087381	0.542290	0.043170	0.433145	0.821147	0.751729	0.000000	0.66690
2	0.958763	0.962564	0.523742	0.170908	0.0	0.292374	0.542290	0.523215	0.433145	0.467460	0.464792	0.000000	0.74482
3	0.958763	0.849585	0.201765	0.141556	0.0	0.322651	0.542290	0.571393	0.433145	0.467460	0.464792	0.000000	0.70944
4	0.958763	0.737842	0.330391	0.113333	0.0	0.365614	0.542290	0.602950	0.433145	0.467460	0.464792	0.000000	0.73866
...	...	...	...	...	...	...	...	...	...	...	...	...	
433	0.958763	0.241218	0.440101	0.437961	0.0	0.695505	0.259760	0.708392	0.419363	0.370242	0.389668	0.522771	0.74064
434	0.958763	0.222904	0.422446	0.432782	0.0	0.697454	0.237460	0.770307	0.388531	0.364326	0.381045	0.510503	0.52973
435	0.958763	0.207127	0.419924	0.427896	0.0	0.715886	0.219480	0.754486	0.360019	0.366614	0.380696	0.543968	0.59183
436	0.958763	0.176152	0.465322	0.404360	0.0	0.765750	0.226659	0.764332	0.356180	0.412712	0.432146	0.540871	0.75121
437	0.827629	0.171356	0.186633	0.398294	0.0	0.183871	0.235141	0.645467	0.385332	0.270478	0.290906	0.370053	0.51190

438 rows × 14 columns

```
plt.figure(figsize=(18,8))
plt.plot(well1_scaled["BORE_OIL_VOL"], label="BORE_OIL_VOL",c="darkgreen")
plt.title('Historial de producción Well 1: BOPD vs. Days')
plt.xlabel("Days")
plt.ylabel("BOPD")
plt.legend(loc="best")

Diff=well1_scaled["BORE_OIL_VOL"].diff()

plt.figure(figsize=(18,8))
plt.plot(Diff, label="BORE_OIL_VOL",c="darkgreen")
plt.xlabel("Days")
plt.ylabel("BOPD")
plt.legend(loc="best")
```

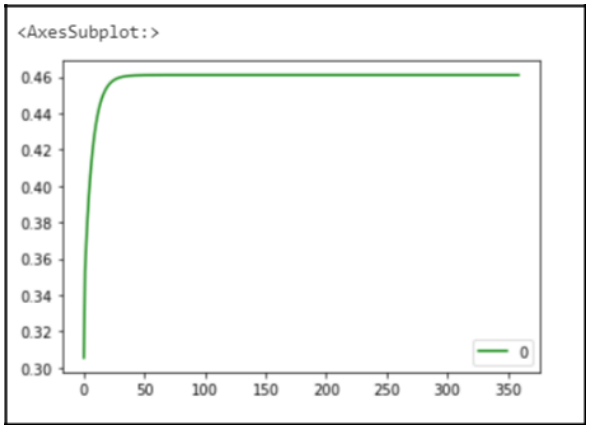



```
#partición de la data
438*0.8
```

```
350.40000000000003
```

```
well1_OP_train=well1_scaled["BORE_OIL_VOL"][:350]
well1_OP_test=well1_scaled["BORE_OIL_VOL"][350:438]

p_values=range(0,4)
d_values=range(0,2)
q_values=range(0,4)
```

```
import warnings
warnings.filterwarnings("ignore")

for p in p_values:
    for d in d_values:
        for q in q_values:
            order=(p,d,q)
            train, test=well1_scaled["BORE_OIL_VOL"][:350],
well1_scaled["BORE_OIL_VOL"][350:438]
            predictions= list()
            for i in range(len(test)):
                try:
                    model=ARIMA(train, order)
                    model_fit=model.fit(disp=0)
                    pred_y=model_fit.forecast()[0]
                    predictions.append(pred_y)
                    error=np.sqrt(mse(test, predictions))
                    print("ARIMA%s RMSE = %.2f"% (order,error))
                except:
                    continue

#Realizando el tuneo de hiperparámetros se obtiene los valores más adecuados
para p,d y q que son 1,0,2
well1_model_final=ARIMA(well1_scaled["BORE_OIL_VOL"], order=(1,0,2))
well1_model_final_fit=well1_model_final.fit()

#realizamos una predicción de los próximos 360 dias
well1_fcast=well1_model_final_fit.forecast(steps=360)[0]

oil_prod_well1=pd.DataFrame(well1_fcast)

oil_prod_well1.plot(c="g")
```

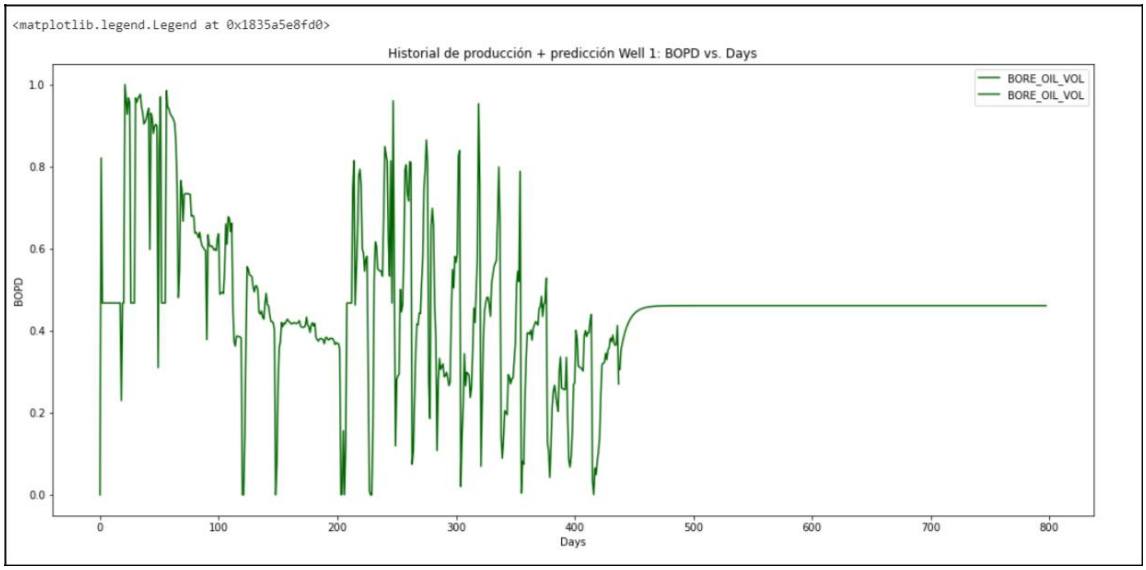



```
production=pd.DataFrame(well1_scaled["BORE_OIL_VOL"])
production
```

	BORE_OIL_VOL
0	0.000000
1	0.821147
2	0.467460
3	0.467460
4	0.467460
...	...
433	0.370242
434	0.364326
435	0.366614
436	0.412712
437	0.270478

438 rows × 1 columns

```
Forecast=production.append(oil_prod_well1,ignore_index=True)

plt.figure(figsize=(18,8))
plt.plot(Forecast, label="BORE_OIL_VOL",c="darkgreen")
plt.title('Historial de producción + predicción Well 1: BOPD vs. Days')
plt.xlabel("Days")
plt.ylabel("BOPD")
plt.legend(loc="best")
```

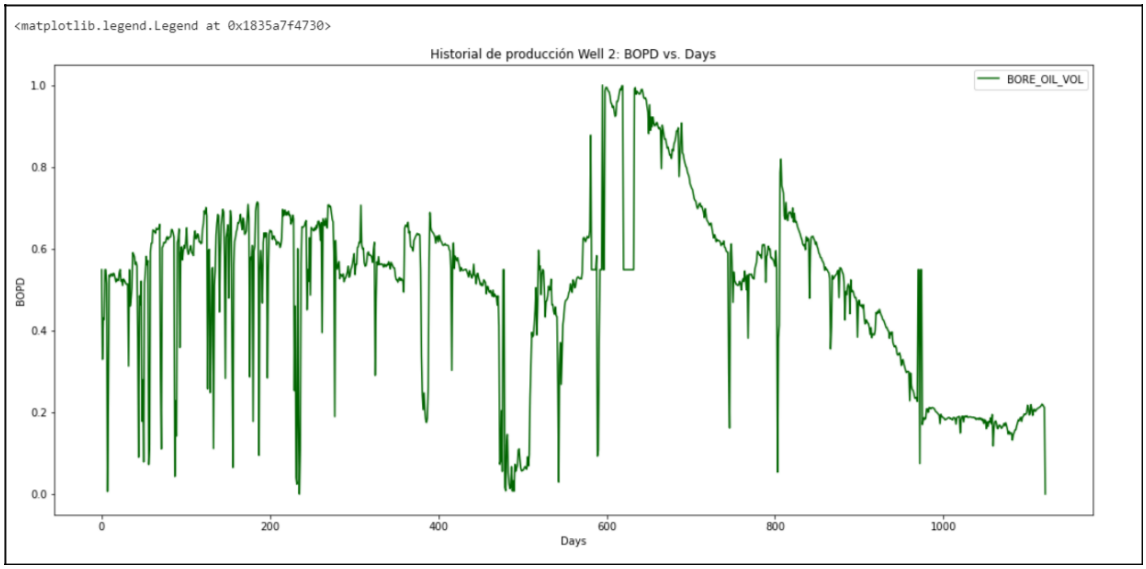



```
#POZO 2

#escalamiento de la data
df_prescaled=df_final_w2.drop("DATEPRD",axis=1)
scaler_all_data=MinMaxScaler()
data_scaled = scaler_all_data.fit_transform(df_prescaled)
well2_scaled=pd.DataFrame(data_scaled,
columns=df_prescaled.columns) well2_scaled
```

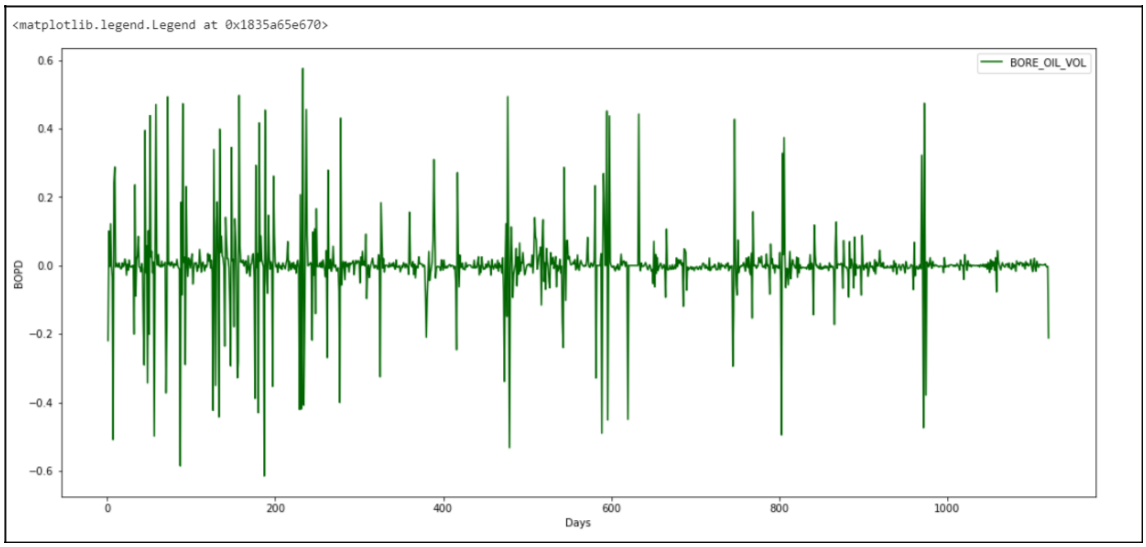
	ON_STREAM_HRS	AVG_DOWNHOLE_PRESSURE	AVG_DOWNHOLE_TEMPERATURE	AVG_DP_TUBING	AVG_ANNULUS_PRESS	AVG_CHOKE_SIZE_P	AVG_WHP_P	AVG_WHT_P	AVG_CHOKE_SIZE	BORE_OIL_VOL	BORE_GAS_VOL	BORE_WAT_VOL	WI_
0	0.254967	0.322806	0.541460	0.205955	0.000000	0.018396	0.274398	0.645749	0.258392	0.548738	0.536482	0.000000	0.2791
1	0.958609	0.322806	0.541460	0.205955	0.622209	0.072097	0.999032	0.645749	1.000000	0.329403	0.265042	0.000000	0.5304
2	0.958609	0.365422	0.296112	0.318164	0.608427	0.080925	0.508505	0.649564	0.333840	0.429954	0.373039	0.000000	0.3054
3	0.958609	0.365422	0.296112	0.318164	0.608427	0.079940	0.508505	0.649564	0.333840	0.426898	0.388580	0.000000	0.5437
4	0.000000	0.322806	0.541460	0.205955	0.651105	0.010357	0.000000	0.645749	0.000000	0.548738	0.536482	0.000000	0.6526
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1117	0.958609	0.555206	0.113243	0.902230	0.738507	1.000000	0.264126	0.971694	0.048030	0.219969	0.164496	0.853121	0.0000
1118	0.958609	0.553249	0.116337	0.900259	0.744361	1.000000	0.263696	0.994165	0.046481	0.218803	0.174825	0.832518	0.2041
1119	0.958609	0.550443	0.117574	0.897316	0.739145	1.000000	0.263177	0.942825	0.046769	0.214569	0.183009	0.797509	0.5038
1120	0.958609	0.549956	0.118812	0.897114	0.738770	1.000000	0.262864	0.966703	0.046631	0.211791	0.181516	0.787362	0.5044
1121	0.307947	0.556728	0.120668	0.897910	0.741209	0.733794	0.268685	1.000000	0.057446	0.000260	0.536482	0.251874	0.1599

1122 rows × 14 columns

```
plt.figure(figsize=(18,8))
plt.plot(well2_scaled["BORE_OIL_VOL"], label="BORE_OIL_VOL",c="darkgreen")
plt.title('Historial de producción Well 2: BOPD vs. Days')
plt.xlabel("Days")
plt.ylabel("BOPD")
plt.legend(loc="best")
```



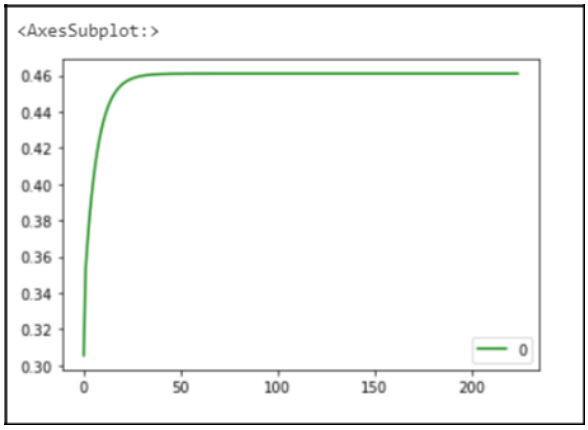

```
Diff=well2_scaled["BORE_OIL_VOL"].diff()

plt.figure(figsize=(18,8))
plt.plot(Diff, label="BORE_OIL_VOL",c="darkgreen")
plt.xlabel("Days")
plt.ylabel("BOPD")
plt.legend(loc="best")
```



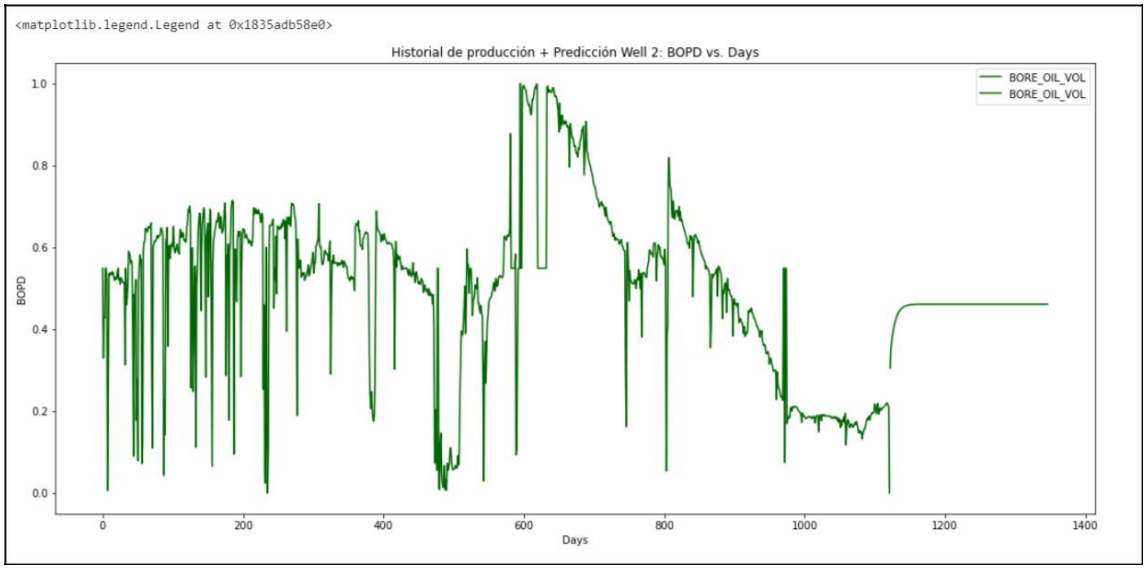

```
#partición de la data
1122*0.8
```

```
897.6
```

```
well2_OP_train=well2_scaled["BORE_OIL_VOL"][:897]
well2_OP_test=well2_scaled["BORE_OIL_VOL"][897:1122]

p_values=range(0,4)
d_values=range(0,2)
q_values=range(0,4)

for p in p_values:
    for d in d_values:
        for q in q_values:
            order=(p,d,q)
            train, test=well2_scaled["BORE_OIL_VOL"][:897],
well2_scaled["BORE_OIL_VOL"][897:1122]
            predictions= list()
            for i in range(len(test)):
                try:
                    model=ARIMA(train, order)
                    model_fit=model.fit(disp=0)
                    pred_y=model_fit.forecast()[0]
                    predictions.append(pred_y)
                    error=np.sqrt(mse(test, predictions))
                    print("ARIMA%s RMSE = %.2f"% (order,error))
                except:
                    continue

well2_model_final=ARIMA(well2_scaled["BORE_OIL_VOL"], order=(1,0,2))
well2_model_final_fit=well1_model_final.fit()
well2_fcast=well1_model_final_fit.forecast(steps=225)[0]

oil_prod_well2=pd.DataFrame(well2_fcast)
oil_prod_well2.plot(c="g")
```




```
production=pd.DataFrame(well2_scaled["BORE_OIL_VOL"])
Forecast=production.append(oil_prod_well2,ignore_index=True)

plt.figure(figsize=(18,8))
plt.plot(Forecast, label="BORE_OIL_VOL",c="darkgreen")
plt.title('Historial de producción + Predicción Well 2: BOPD vs. Days')
plt.xlabel("Days")
plt.ylabel("BOPD")
plt.legend(loc="best")
```



Modelo de Random Forest

```
import lasio
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tabula.io import read_pdf
```

```
import welly
import seaborn as sns
from ipywidgets import *
from datetime import date, time, datetime

%config Completer.use_jedi = False
%matplotlib inline

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV

from sklearn.ensemble import RandomForestRegressor

from sklearn import metrics
from  sklearn.metrics  import  mean_absolute_error
from  sklearn.metrics  import  mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import explained_variance_score

from math import sqrt

from scipy import stats
import statsmodels as sm

from sklearn.model_selection import
train_test_split from pprint import print
from sklearn.preprocessing import StandardScaler, MinMaxScaler

#POZO 1 (7405)
well1=pd.read_csv("C:/Users/dario/proyectos jupyter/Well1.csv")

#escalamiento de los datos
df_prescaled=well1.drop("DATEPRD",axis=1)
scaler_all_data=MinMaxScaler()
data_scaled = scaler_all_data.fit_transform(df_prescaled)
well1_scaled=pd.DataFrame(data_scaled, columns=df_prescaled.columns)

X = well1_scaled.filter(["ON_STREAM_HRS", 'AVG_DOWNHOLE_PRESSURE',
'AVG_DP_TUBING', 'AVG_WHP_P', 'AVG_WHT_P', 'DP_CHOKE_SIZE',
"WI_F4","WI_F5"]) Y = well1_scaled[['BORE_OIL_VOL', 'BORE_GAS_VOL']]
print('Features shape:', X.shape)
```

```
print('Target shape', Y.shape)
```

```
Features shape: (438, 7)
Target shape (438, 2)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=42, shuffle = True)
#random state=42

Time=well1_scaled.iloc[:,0]
Time_train, Time_test = train_test_split(Time,
test_size=0.3, random_state=42)

rf = RandomForestRegressor(random_state=0).fit(X_train, y_train)
predictions_rf = rf.predict(X_test)


# Metrics
print('Model score:',              round(rf.score(X_test, y_test),2))
print('Mean absolute error:',      round(mean_absolute_error(y_test,
predictions_rf),2))
print('Root mean squared error:',  round(sqrt(mean_squared_error(y_test,
predictions_rf)),2))
print('R2:',                       round(r2_score(y_test, predictions_rf),2))

print('Model Trained Score:',      round(rf.score(X_train, y_train),2))

#agregar o eliminar nuevas variables al modelo para mejorar el modelo con
la data de prueba
```

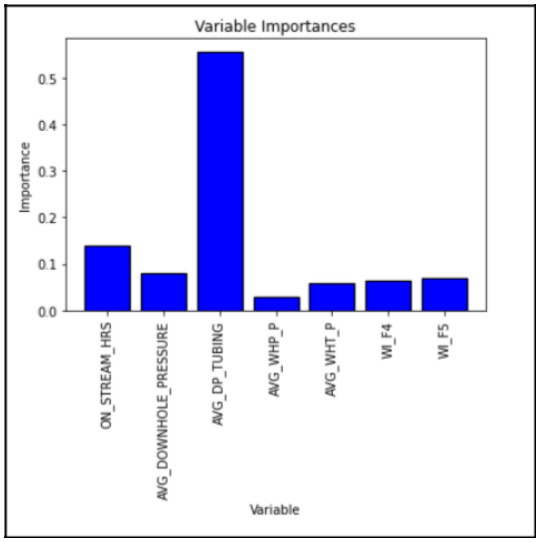
```
Model score: 0.81
Mean absolute error: 0.06
Root mean squared error: 0.11
R2: 0.81
Model Trained Score: 0.97
```

```
# List of features
feature_list = list(X_train.columns)

# Get numerical feature importances (Gini importance)
importances = list(rf.feature_importances_)
```

```
# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for
feature, importance in zip(feature_list, importances)]

# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x:
x[1], reverse = True)

# Print out the feature and importances
[print('Variable: {:30} Importance: {}'.format(*pair)) for pair
in feature_importances];
```

```
Variable: AVG_DP_TUBING                Importance: 0.56
Variable: ON_STREAM_HRS                Importance: 0.14
Variable: AVG_DOWNHOLE_PRESSURE        Importance: 0.08
Variable: WI_F5                        Importance: 0.07
Variable: AVG_WHT_P                    Importance: 0.06
Variable: WI_F4                        Importance: 0.06
Variable: AVG_WHP_P                    Importance: 0.03
```

```
# list of x locations for plotting
X_testues = list(range(len(importances)))

# Make a bar chart
plt.bar(X_testues, importances, orientation = 'vertical', color =
'b', edgecolor = 'k', linewidth = 1.2)

# Tick labels for x axis
plt.xticks(X_testues, feature_list, rotation='vertical')

# Axis labels and title
plt.ylabel('Importance'); plt.xlabel('Variable');
plt.title('Variable Importances');
```




```
rf.fit(X_train, y_train);
predictions_import = rf.predict(X_test)
pprint(rf.get_params())
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 0,
 'verbose': 0,
 'warm_start': False}
```

```
# Definition of specific parameters for Random forest
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 2, stop = 2000, num
= 20)]
# Number of features to consider at every
split max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(4, 30, num =
2)] max_depth.append(None)
# Minimum number of samples required to split a node
```

```
min_samples_split = [2, 3, 4, 5, 10]
# Minimum number of samples required at each leaf
node min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each
tree bootstrap = [True, False]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [4, 30, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 3, 4, 5, 10],
 'n_estimators': [2,
                  107,
                  212,
                  317,
                  422,
                  527,
                  632,
                  738,
                  843,
                  948,
                  1053,
                  1158,
                  1263,
                  1369,
                  1474,
                  1579,
                  1684,
                  1789,
                  1894,
                  2000]}
```

```
rf = RandomForestRegressor(random_state = 42)

# Use the random grid to search for best hyperparameters
# Random search of parameters, using 3 fold cross validation,
# search across different combinations.
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid,
n_iter = 15, scoring='neg_mean_absolute_error', cv = 3, verbose=2,
random_state=42, n_jobs=-1, return_train_score=True)

# Fit the random search model
rf_random.fit(X_train, y_train);
```

```
# Obtaining the best parameters

rf_random.best_params_
```

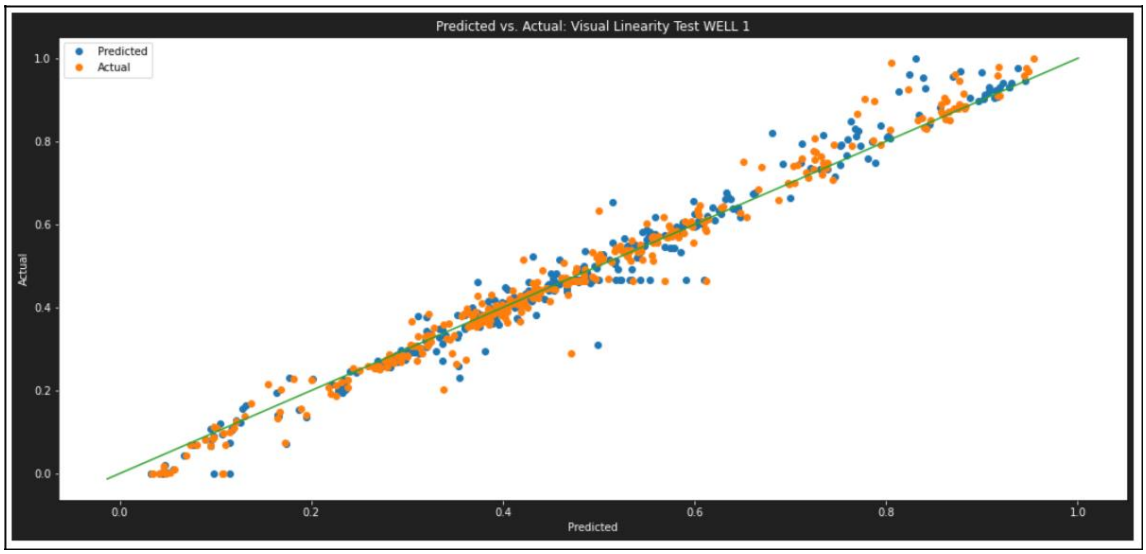
```
{'n_estimators': 1894,
 'min_samples_split': 3,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': None,
 'bootstrap': True}
```

```
#Evaluación de los parámetros
best_random = rf_random.best_estimator_.fit(X_train, y_train)
predictions_best_random_test = best_random.predict(X_test)
predictions_best_random_train = best_random.predict(X_train)

print('Model score:', round(best_random.score(X_test, y_test),2))
print('Mean absolute error:', round(mean_absolute_error(y_test,
predictions_best_random_test),2))
print('Root mean squared error:', round(sqrt(mean_squared_error(y_test,
predictions_best_random_test)),2))
print('R2:', round(r2_score(y_test, predictions_best_random_test),2))

r2_rf=r2_score(y_test, predictions_best_random_test)
Mean_absolute_error_rf=mean_absolute_error(y_test,
predictions_best_random_test)
Root_mean_squared_error_rf=sqrt(mean_squared_error(y_test,
predictions_best_random_test))
```

```
Model score: 0.8
Mean absolute error: 0.06
Root mean squared error: 0.11
R2: 0.8
```

```
result_df = pd.DataFrame({'BORE_OIL_VOL': [y_test], 'Predicted':
[predictions_best_random_test]})

y = y_train
X_train_np = np.array(X_train)
y_np = np.array(y)
```

```
def abline(slope, intercept):
    axes = plt.gca()
    x_vals = np.array(axes.get_xlim())
    y_vals = intercept + slope * x_vals
    plt.plot(x_vals, y_vals, '-')

#predict y values for training data
y_hat = predictions_best_random_train

#plot predicted vs actual
plt.figure(figsize=(18,8))
plt.plot(y_hat,y_np,'o')
plt.legend(["Predicted","Actual"])
plt.xlabel('Predicted',color='white')
plt.ylabel('Actual',color='white')
plt.title('Predicted vs. Actual: Visual Linearity Test WELL 1',color='white')
plt.tick_params(axis='x', colors='white')
plt.tick_params(axis='y', colors='white')
abline(1,0)
plt.show()
```




```
#POZO 2 (7078)
well2=pd.read_csv("C:/Users/dario/proyectos jupyter/Well2.csv")

#escalamiento de los datos
df_prescaled=well2.drop("DATEPRD",axis=1)
scaler_all_data=MinMaxScaler()
```

```
data_scaled = scaler_all_data.fit_transform(df_prescaled)
well2_scaled=pd.DataFrame(data_scaled, columns=df_prescaled.columns)

X = well2_scaled.filter(["ON_STREAM_HRS", 'AVG_DOWNHOLE_PRESSURE',
'AVG_DP_TUBING', 'AVG_WHP_P', 'AVG_WHT_P', 'DP_CHOKE_SIZE',
"WI_F4","WI_F5"]) Y = well2_scaled[['BORE_OIL_VOL', 'BORE_GAS_VOL']]
print('Features shape:', X.shape)
print('Target shape', Y.shape)
```

```
Features shape: (1122, 7)
Target shape (1122, 2)
```

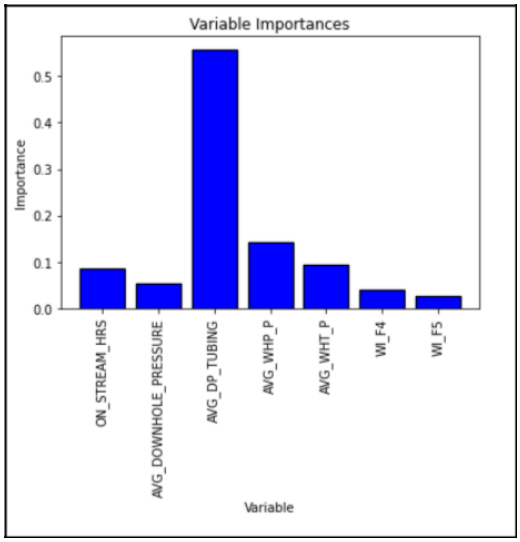
```
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=42, shuffle = True)
#random state=42

Time=well2_scaled.iloc[:,0]
Time_train, Time_test = train_test_split(Time,
test_size=0.3, random_state=42)

rf = RandomForestRegressor(random_state=0).fit(X_train, y_train)
predictions_rf = rf.predict(X_test)


# Metrics
print('Model score:',              round(rf.score(X_test, y_test),2))
print('Mean absolute error:',      round(mean_absolute_error(y_test,
predictions_rf),2))
print('Root mean squared error:', round(sqrt(mean_squared_error(y_test,
predictions_rf)),2))
print('R2:',                       round(r2_score(y_test, predictions_rf),2))

print('Model Trained Score:',       round(rf.score(X_train, y_train),2))

#agregar o eliminar nuevas variables al modelo para mejorar el modelo con
la data de prueba
```

```
Model score: 0.88
Mean absolute error: 0.03
Root mean squared error: 0.06
R2: 0.88
Model Trained Score: 0.98
```

```
# List of features
feature_list = list(X_train.columns)

# Get numerical feature importances (Gini importance)
importances = list(rf.feature_importances_)

# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for
feature, importance in zip(feature_list, importances)]

# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x:
x[1], reverse = True)

# Print out the feature and importances
[print('Variable: {:30} Importance: {}'.format(*pair)) for pair
in feature_importances];
```

```
Variable: AVG_DP_TUBING                Importance: 0.56
Variable: AVG_WHP_P                    Importance: 0.14
Variable: ON_STREAM_HRS                Importance: 0.09
Variable: AVG_WHT_P                    Importance: 0.09
Variable: AVG_DOWNHOLE_PRESSURE        Importance: 0.05
Variable: WI_F4                        Importance: 0.04
Variable: WI_F5                        Importance: 0.03
```

```
# list of x locations for plotting
X_testues = list(range(len(importances)))

# Make a bar chart
plt.bar(X_testues, importances, orientation = 'vertical', color =
'b', edgecolor = 'k', linewidth = 1.2)

# Tick labels for x axis
plt.xticks(X_testues, feature_list, rotation='vertical')

# Axis labels and title
plt.ylabel('Importance'); plt.xlabel('Variable');
plt.title('Variable Importances');
```




```
rf.fit(X_train, y_train);
predictions_import = rf.predict(X_test)
pprint(rf.get_params())
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 0,
 'verbose': 0,
 'warm_start': False}
```

```
# Definition of specific parameters for Random forest
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 2, stop = 2000, num
= 20)]
# Number of features to consider at every
split max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(4, 30, num =
2)] max_depth.append(None)
# Minimum number of samples required to split a
node min_samples_split = [2, 3, 4, 5, 10]
```

```
# Minimum number of samples required at each leaf
node min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each
tree bootstrap = [True, False]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [4, 30, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 3, 4, 5, 10],
 'n_estimators': [2,
                  107,
                  212,
                  317,
                  422,
                  527,
                  632,
                  738,
                  843,
                  948,
                  1053,
                  1158,
                  1263,
                  1369,
                  1474,
                  1579,
                  1684,
                  1789,
                  1894,
                  2000]}
```

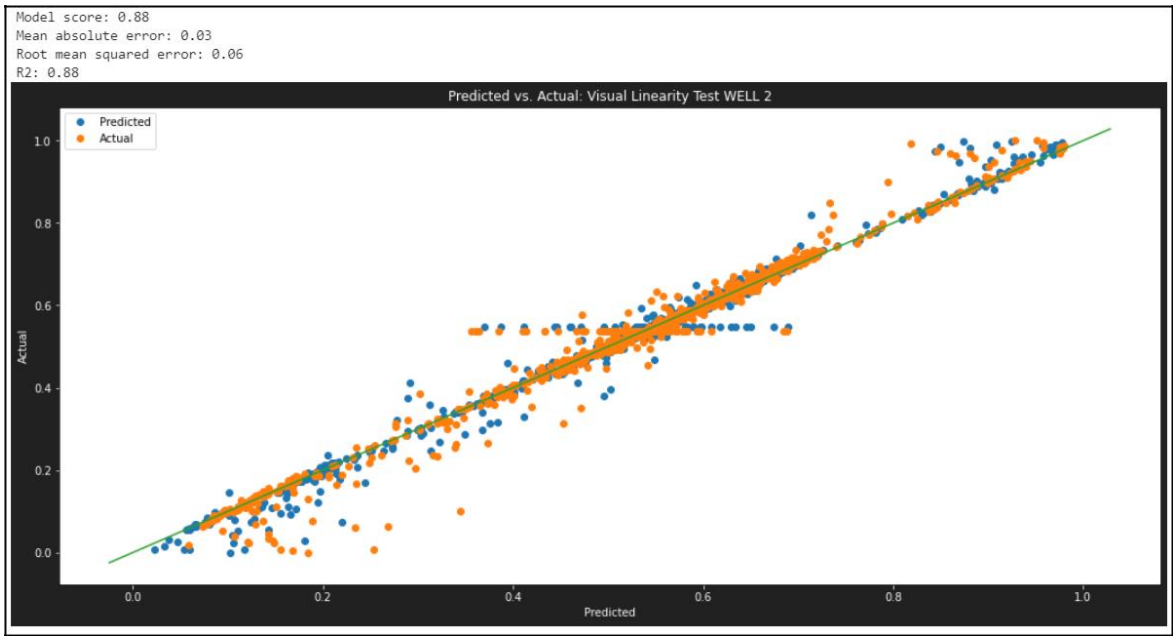
```
rf = RandomForestRegressor(random_state = 42)

# Use the random grid to search for best hyperparameters
# Random search of parameters, using 3 fold cross validation,
# search across different combinations.
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid,
n_iter = 15, scoring='neg_mean_absolute_error', cv = 3, verbose=2,
random_state=42, n_jobs=-1, return_train_score=True)

# Fit the random search model
rf_random.fit(X_train, y_train);
```

```
# Obtaining the best parameters

rf_random.best_params_
```

```
{'n_estimators': 1894,
 'min_samples_split': 3,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': None,
 'bootstrap': True}
```

```
best_random = rf_random.best_estimator_.fit(X_train, y_train)
predictions_best_random_test = best_random.predict(X_test)
predictions_best_random_train = best_random.predict(X_train)

print('Model score:', round(best_random.score(X_test, y_test),2))
print('Mean absolute error:', round(mean_absolute_error(y_test,
predictions_best_random_test),2))
print('Root mean squared error:', round(sqrt(mean_squared_error(y_test,
predictions_best_random_test)),2))
print('R2:', round(r2_score(y_test, predictions_best_random_test),2))

r2_rf=r2_score(y_test, predictions_best_random_test)
Mean_absolute_error_rf=mean_absolute_error(y_test,
predictions_best_random_test)
Root_mean_squared_error_rf=sqrt(mean_squared_error(y_test,
predictions_best_random_test))


result_df = pd.DataFrame({'BORE_OIL_VOL': [y_test], 'Predicted':
[predictions_best_random_test]})


y = y_train
X_train_np = np.array(X_train)
y_np = np.array(y)

#predict y values for training data
y_hat = predictions_best_random_train

#plot predicted vs actual
plt.figure(figsize=(18,8))
```

```
plt.plot(y_hat,y_np,'o')
plt.legend(["Predicted","Actual"])
plt.xlabel('Predicted',color='white')
plt.ylabel('Actual',color='white')
plt.title('Predicted vs. Actual: Visual Linearity Test WELL 2',color='white')
plt.tick_params(axis='x', colors='white')
plt.tick_params(axis='y', colors='white')
abline(1,0)
plt.show()
```




```
#POZO 3 (5351)
well3=pd.read_csv("C:/Users/dario/proyectos jupyter/Well3.csv")

df_prescaled=well3.drop("DATEPRD",axis=1)
scaler_all_data=MinMaxScaler()
data_scaled = scaler_all_data.fit_transform(df_prescaled)
well3_scaled=pd.DataFrame(data_scaled, columns=df_prescaled.columns)

well3_scaled.to_excel("Well3_prueba.xlsx", index=False)
X = well3_scaled.filter(["ON_STREAM_HRS", 'AVG_DOWNHOLE_PRESSURE',
'AVG_DP_TUBING', 'AVG_WHP_P', 'AVG_WHT_P', 'DP_CHOKE_SIZE',
"WI_F4","WI_F5"]) Y = well3_scaled[['BORE_OIL_VOL', 'BORE_GAS_VOL']]
print('Features shape:', X.shape)
print('Target shape', Y.shape)
```

```
Features shape: (2838, 7)
Target shape (2838, 2)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=42, shuffle = True)
#random state=42

Time=well3_scaled.iloc[:,0]
Time_train, Time_test = train_test_split(Time,
test_size=0.3, random_state=42)
rf = RandomForestRegressor(random_state=0).fit(X_train, y_train)
predictions_rf = rf.predict(X_test)


# Metrics
print('Model score:',              round(rf.score(X_test, y_test),2))
print('Mean absolute error:',      round(mean_absolute_error(y_test,
predictions_rf),2))
print('Root mean squared error:',  round(sqrt(mean_squared_error(y_test,
predictions_rf)),2))
print('R2:',                       round(r2_score(y_test, predictions_rf),2))

print('Model Trained Score:',      round(rf.score(X_train, y_train),2))
#agregar o eliminar nuevas variables al modelo para mejorar el modelo con
la data de prueba
```

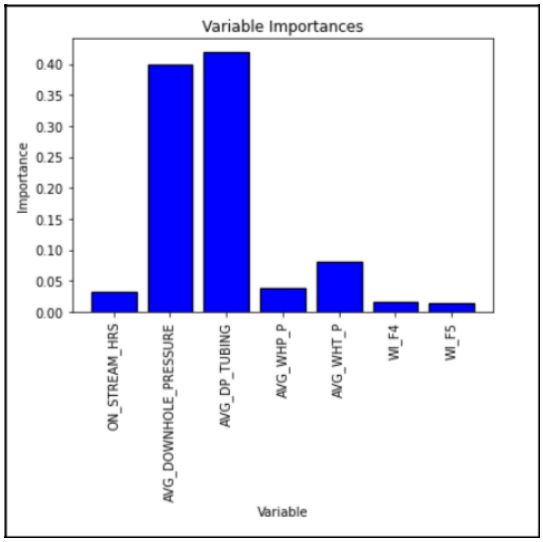
```
Model score: 0.97
Mean absolute error: 0.03
Root mean squared error: 0.05
R2: 0.97
Model Trained Score: 1.0
```

```
# List of features
feature_list = list(X_train.columns)

# Get numerical feature importances (Gini importance)
importances = list(rf.feature_importances_)
# List of tuples with variable and importance
```

```
feature_importances = [(feature, round(importance, 2)) for
feature, importance in zip(feature_list, importances)]

# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x:
x[1], reverse = True)

# Print out the feature and importances
[print('Variable: {:30} Importance: {}'.format(*pair)) for pair
in feature_importances];
```

```
Variable: AVG_DP_TUBING                Importance: 0.42
Variable: AVG_DOWNHOLE_PRESSURE        Importance: 0.4
Variable: AVG_WHT_P                    Importance: 0.08
Variable: AVG_WHP_P                    Importance: 0.04
Variable: ON_STREAM_HRS                Importance: 0.03
Variable: WI_F4                        Importance: 0.02
Variable: WI_F5                        Importance: 0.01
```

```
# list of x locations for plotting
X_testues = list(range(len(importances)))

# Make a bar chart
plt.bar(X_testues, importances, orientation = 'vertical', color =
'b', edgecolor = 'k', linewidth = 1.2)

# Tick labels for x axis
plt.xticks(X_testues, feature_list, rotation='vertical')

# Axis labels and title
plt.ylabel('Importance'); plt.xlabel('Variable');
plt.title('Variable Importances');
```




```
rf.fit(X_train, y_train);
predictions_import = rf.predict(X_test)
pprint(rf.get_params())
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 0,
 'verbose': 0,
 'warm_start': False}
```

```
# Definition of specific parameters for Random forest
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 2, stop = 2000, num
= 20)]
# Number of features to consider at every
split max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(4, 30, num =
2)] max_depth.append(None)
# Minimum number of samples required to split a node
```

```
min_samples_split = [2, 3, 4, 5, 10]
# Minimum number of samples required at each leaf
node min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each
tree bootstrap = [True, False]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [4, 30, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 3, 4, 5, 10],
 'n_estimators': [2,
                  107,
                  212,
                  317,
                  422,
                  527,
                  632,
                  738,
                  843,
                  948,
                  1053,
                  1158,
                  1263,
                  1369,
                  1474,
                  1579,
                  1684,
                  1789,
                  1894,
                  2000]}
```

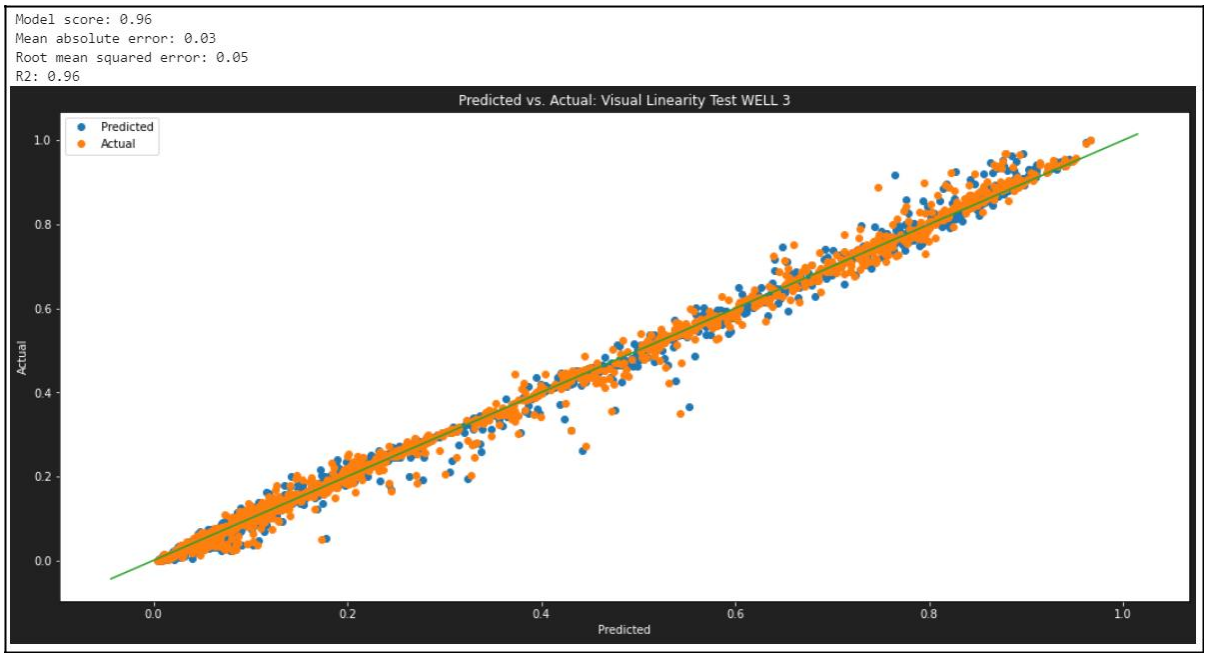
```
rf = RandomForestRegressor(random_state = 42)

# Use the random grid to search for best hyperparameters
# Random search of parameters, using 3 fold cross validation,
# search across different combinations.
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid,
n_iter = 15, scoring='neg_mean_absolute_error', cv = 3, verbose=2,
random_state=42, n_jobs=-1, return_train_score=True)

# Fit the random search model
rf_random.fit(X_train, y_train);
```

```
# Obtaining the best parameters

rf_random.best_params_
```

```
{'n_estimators': 1894,
 'min_samples_split': 3,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': None,
 'bootstrap': True}
```

```
best_random = rf_random.best_estimator_.fit(X_train, y_train)
predictions_best_random_test = best_random.predict(X_test)
predictions_best_random_train = best_random.predict(X_train)

print('Model score:', round(best_random.score(X_test, y_test),2))
print('Mean absolute error:', round(mean_absolute_error(y_test,
predictions_best_random_test),2))
print('Root mean squared error:', round(sqrt(mean_squared_error(y_test,
predictions_best_random_test)),2))
print('R2:', round(r2_score(y_test, predictions_best_random_test),2))

r2_rf=r2_score(y_test, predictions_best_random_test)
Mean_absolute_error_rf=mean_absolute_error(y_test,
predictions_best_random_test)
Root_mean_squared_error_rf=sqrt(mean_squared_error(y_test,
predictions_best_random_test))


result_df = pd.DataFrame({'BORE_OIL_VOL': [y_test], 'Predicted':
[predictions_best_random_test]})


y = y_train
X_train_np = np.array(X_train)
y_np = np.array(y)

#predict y values for training data
y_hat = predictions_best_random_train

#plot predicted vs actual
```

```
plt.figure(figsize=(18,8))
plt.plot(y_hat,y_np,'o')
plt.legend(["Predicted","Actual"])
plt.xlabel('Predicted',color='white')
plt.ylabel('Actual',color='white')
plt.title('Predicted vs. Actual: Visual Linearity Test WELL 3',color='white')
plt.tick_params(axis='x', colors='white')
plt.tick_params(axis='y', colors='white')
abline(1,0)
plt.show()
```




```
#POZO 4 (5599)
well4=pd.read_csv("C:/Users/dario/proyectos jupyter/Well4.csv")

df_prescaled=well4.drop("DATEPRD",axis=1)
scaler_all_data=MinMaxScaler()
data_scaled = scaler_all_data.fit_transform(df_prescaled)
well4_scaled=pd.DataFrame(data_scaled,
columns=df_prescaled.columns) well4_scaled

X = well4_scaled.filter(["ON_STREAM_HRS", 'AVG_DOWNHOLE_PRESSURE',
'AVG_DP_TUBING', 'AVG_WHP_P', 'AVG_WHT_P', 'DP_CHOKE_SIZE',
"WI_F4","WI_F5"]) Y = well4_scaled[['BORE_OIL_VOL', 'BORE_GAS_VOL']]
print('Features shape:', X.shape)
print('Target shape', Y.shape)
```

```
Features shape: (2724, 7)
Target shape (2724, 2)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=42, shuffle = True)
#random state=42

Time=well4_scaled.iloc[:,0]
Time_train, Time_test = train_test_split(Time,
test_size=0.3, random_state=42)
rf = RandomForestRegressor(random_state=0).fit(X_train, y_train)
predictions_rf = rf.predict(X_test)


# Metrics
print('Model score:',              round(rf.score(X_test, y_test),2))
print('Mean absolute error:',      round(mean_absolute_error(y_test,
predictions_rf),2))
print('Root mean squared error:',  round(sqrt(mean_squared_error(y_test,
predictions_rf)),2))
print('R2:',                       round(r2_score(y_test, predictions_rf),2))

print('Model Trained Score:',      round(rf.score(X_train, y_train),2))
```

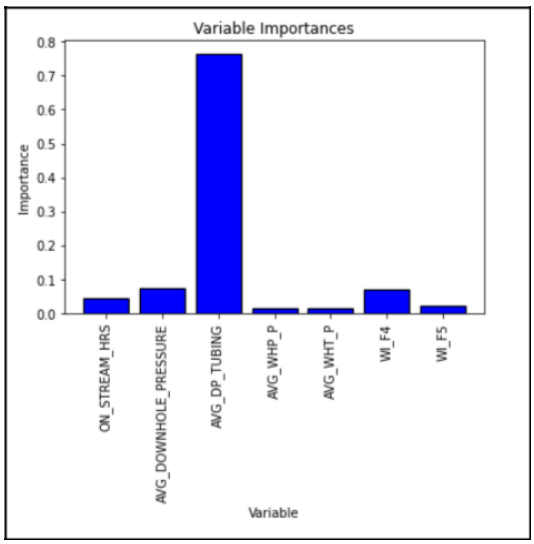
```
Model score: 0.95
Mean absolute error: 0.02
Root mean squared error: 0.06
R2: 0.95
Model Trained Score: 0.99
```

```
# List of features
feature_list = list(X_train.columns)

# Get numerical feature importances (Gini importance)
importances = list(rf.feature_importances_)

# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for
feature, importance in zip(feature_list, importances)]

# Sort the feature importances by most important first
```

```
feature_importances = sorted(feature_importances, key = lambda x:
x[1], reverse = True)

# Print out the feature and importances
[print('Variable: {:30} Importance: {}'.format(*pair)) for pair
in feature_importances];
```

```
Variable: AVG_DP_TUBING                Importance: 0.76
Variable: AVG_DOWNHOLE_PRESSURE        Importance: 0.07
Variable: WI_F4                        Importance: 0.07
Variable: ON_STREAM_HRS                Importance: 0.04
Variable: WI_F5                        Importance: 0.02
Variable: AVG_WHP_P                    Importance: 0.01
Variable: AVG_WHT_P                    Importance: 0.01
```

```
# list of x locations for plotting
X_testues = list(range(len(importances)))

# Make a bar chart
plt.bar(X_testues, importances, orientation = 'vertical', color =
'b', edgecolor = 'k', linewidth = 1.2)

# Tick labels for x axis
plt.xticks(X_testues, feature_list, rotation='vertical')

# Axis labels and title
plt.ylabel('Importance'); plt.xlabel('Variable');
plt.title('Variable Importances');
```



```
rf.fit(X_train, y_train);
predictions_import = rf.predict(X_test)
pprint(rf.get_params())
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 0,
 'verbose': 0,
 'warm_start': False}
```

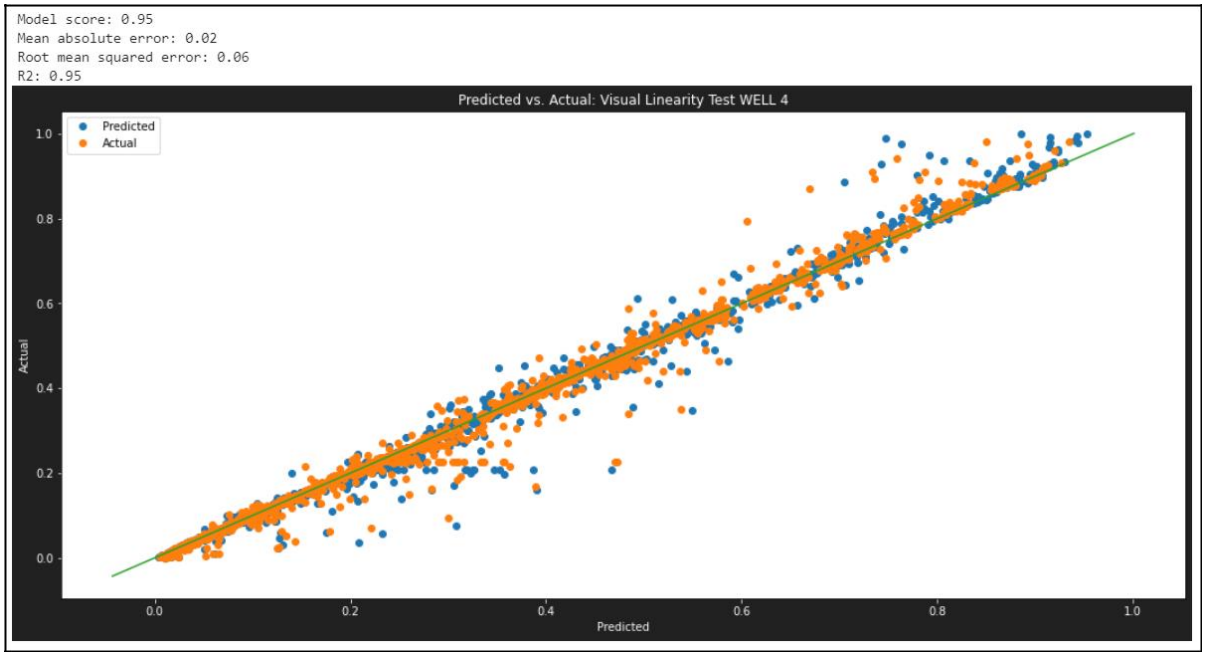
```
# Definition of specific parameters for Random forest
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 2, stop = 2000, num
= 20)]
# Number of features to consider at every
split max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(4, 30, num =
2)] max_depth.append(None)
# Minimum number of samples required to split a
node min_samples_split = [2, 3, 4, 5, 10]
# Minimum number of samples required at each leaf
node min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each
tree bootstrap = [True, False]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

```
pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [4, 30, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 3, 4, 5, 10],
 'n_estimators': [2,
                  107,
                  212,
                  317,
                  422,
                  527,
                  632,
                  738,
                  843,
                  948,
                  1053,
                  1158,
                  1263,
                  1369,
                  1474,
                  1579,
                  1684,
                  1789,
                  1894,
                  2000]}
```

```
rf = RandomForestRegressor(random_state = 42)

# Use the random grid to search for best hyperparameters
# Random search of parameters, using 3 fold cross validation,
# search across different combinations.
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid,
n_iter = 15, scoring='neg_mean_absolute_error', cv = 3, verbose=2,
random_state=42, n_jobs=-1, return_train_score=True)

# Fit the random search model
rf_random.fit(X_train, y_train);

# Obtaining the best parameters
rf_random.best_params_

best_random = rf_random.best_estimator_.fit(X_train, y_train)
predictions_best_random_test = best_random.predict(X_test)
predictions_best_random_train = best_random.predict(X_train)

print('Model score:', round(best_random.score(X_test, y_test),2))
print('Mean absolute error:', round(mean_absolute_error(y_test,
predictions_best_random_test),2))
print('Root mean squared error:', round(sqrt(mean_squared_error(y_test,
predictions_best_random_test)),2))
```

```
print('R2:', round(r2_score(y_test, predictions_best_random_test),2))

r2_rf=r2_score(y_test, predictions_best_random_test)
Mean_absolute_error_rf=mean_absolute_error(y_test,
predictions_best_random_test)
Root_mean_squared_error_rf=sqrt(mean_squared_error(y_test,
predictions_best_random_test))


result_df = pd.DataFrame({'BORE_OIL_VOL': [y_test], 'Predicted':
[predictions_best_random_test]})



y = y_train
X_train_np = np.array(X_train)
y_np = np.array(y)

#predict y values for training data
y_hat = predictions_best_random_train

#plot predicted vs actual
plt.figure(figsize=(18,8))
plt.plot(y_hat,y_np,'o')
plt.legend(["Predicted","Actual"])
plt.xlabel('Predicted',color='white')
plt.ylabel('Actual',color='white')
plt.title('Predicted vs. Actual: Visual Linearity Test WELL 4',color='white')
plt.tick_params(axis='x', colors='white')
plt.tick_params(axis='y', colors='white')
abline(1,0)
plt.show()
```




```
#POZO 5 (7289)
well5=pd.read_csv("C:/Users/dario/proyectos jupyter/Well5.csv")
df_prescaled=well5.drop("DATEPRD",axis=1)
scaler_all_data=MinMaxScaler()
data_scaled = scaler_all_data.fit_transform(df_prescaled)
well5_scaled=pd.DataFrame(data_scaled, columns=df_prescaled.columns)

X = well5_scaled.filter(["ON_STREAM_HRS", 'AVG_DOWNHOLE_PRESSURE',
'AVG_DP_TUBING', 'AVG_WHP_P', 'AVG_WHT_P', 'DP_CHOKE_SIZE',
"WI_F4","WI_F5"]) Y = well5_scaled[['BORE_OIL_VOL', 'BORE_GAS_VOL']]
print('Features shape:', X.shape)
print('Target shape', Y.shape)
```

```
Features shape: (769, 7)
Target shape (769, 2)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=42, shuffle = True)
#random state=42

Time=well5_scaled.iloc[:,0]
Time_train, Time_test = train_test_split(Time,
test_size=0.3, random_state=42)

rf = RandomForestRegressor(random_state=0).fit(X_train, y_train)
```

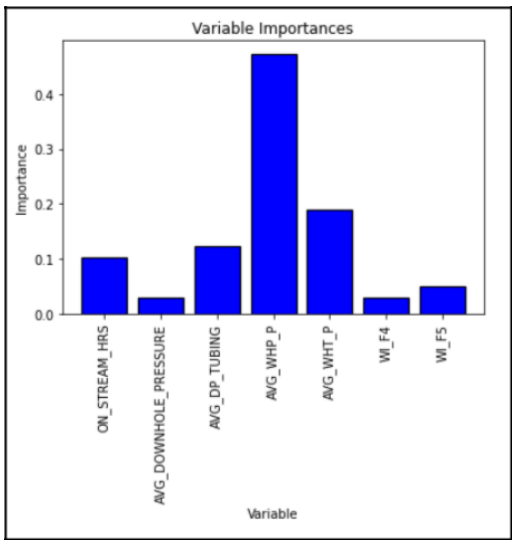
```
predictions_rf = rf.predict(X_test)


# Metrics
print('Model score:',               round(rf.score(X_test, y_test),2))
print('Mean absolute error:',       round(mean_absolute_error(y_test,
predictions_rf),2))
print('Root mean squared error:',   round(sqrt(mean_squared_error(y_test,
predictions_rf)),2))
print('R2:',                        round(r2_score(y_test, predictions_rf),2))

print('Model Trained Score:',       round(rf.score(X_train, y_train),2))
```

```
Model score: 0.69
Mean absolute error: 0.05
Root mean squared error: 0.09
R2: 0.69
Model Trained Score: 0.96
```

```
# List of features
feature_list = list(X_train.columns)

# Get numerical feature importances (Gini importance)
importances = list(rf.feature_importances_)

# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for
feature, importance in zip(feature_list, importances)]

# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x:
x[1], reverse = True)

# Print out the feature and importances
[print('Variable: {:30} Importance: {}'.format(*pair)) for pair
in feature_importances];
```

```
Variable: AVG_WHP_P                    Importance: 0.47
Variable: AVG_WHT_P                    Importance: 0.19
Variable: AVG_DP_TUBING                Importance: 0.12
Variable: ON_STREAM_HRS                Importance: 0.1
Variable: WI_F5                        Importance: 0.05
Variable: AVG_DOWNHOLE_PRESSURE        Importance: 0.03
Variable: WI_F4                        Importance: 0.03
```

```
# list of x locations for plotting
X_testues = list(range(len(importances)))

# Make a bar chart
plt.bar(X_testues, importances, orientation = 'vertical', color =
'b', edgecolor = 'k', linewidth = 1.2)

# Tick labels for x axis
plt.xticks(X_testues, feature_list, rotation='vertical')

# Axis labels and title
plt.ylabel('Importance'); plt.xlabel('Variable');
plt.title('Variable Importances');
```




```
rf.fit(X_train, y_train);
predictions_import = rf.predict(X_test)
pprint(rf.get_params())
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 0,
 'verbose': 0,
 'warm_start': False}
```

```
# Definition of specific parameters for Random forest
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 2, stop = 2000, num
= 20)]
# Number of features to consider at every
split max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(4, 30, num =
2)] max_depth.append(None)
# Minimum number of samples required to split a
node min_samples_split = [2, 3, 4, 5, 10]
# Minimum number of samples required at each leaf
node min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each
tree bootstrap = [True, False]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [4, 30, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 3, 4, 5, 10],
 'n_estimators': [2,
                  107,
                  212,
                  317,
                  422,
                  527,
                  632,
                  738,
                  843,
                  948,
                  1053,
                  1158,
                  1263,
                  1369,
                  1474,
                  1579,
                  1684,
                  1789,
                  1894,
                  2000]}
```

```
rf = RandomForestRegressor(random_state = 42)

# Use the random grid to search for best hyperparameters
# Random search of parameters, using 3 fold cross validation,
# search across different combinations.
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid,
n_iter = 15, scoring='neg_mean_absolute_error', cv = 3, verbose=2,
random_state=42, n_jobs=-1, return_train_score=True)

# Fit the random search model
rf_random.fit(X_train, y_train);

# Obtaining the best parameters
rf_random.best_params_
```

```
{'n_estimators': 422,
 'min_samples_split': 4,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': None,
 'bootstrap': False}
```

```
best_random = rf_random.best_estimator_.fit(X_train, y_train)
predictions_best_random_test = best_random.predict(X_test)
predictions_best_random_train = best_random.predict(X_train)

print('Model score:', round(best_random.score(X_test, y_test),2))
```

```
print('Mean absolute error:', round(mean_absolute_error(y_test,
predictions_best_random_test),2))
print('Root mean squared error:', round(sqrt(mean_squared_error(y_test,
predictions_best_random_test)),2))
print('R2:', round(r2_score(y_test, predictions_best_random_test),2))

r2_rf=r2_score(y_test, predictions_best_random_test)
Mean_absolute_error_rf=mean_absolute_error(y_test,
predictions_best_random_test)
Root_mean_squared_error_rf=sqrt(mean_squared_error(y_test,
predictions_best_random_test))


result_df = pd.DataFrame({'BORE_OIL_VOL': [y_test], 'Predicted':
[predictions_best_random_test]})


y = y_train
X_train_np = np.array(X_train)
y_np = np.array(y)

#predict y values for training data
y_hat = predictions_best_random_train

#plot predicted vs actual
plt.figure(figsize=(18,8))
plt.plot(y_hat,y_np,'o')
plt.legend(["Predicted","Actual"])
plt.xlabel('Predicted',color='white')
plt.ylabel('Actual',color='white')
plt.title('Predicted vs. Actual: Visual Linearity Test WELL 5',color='white')
plt.tick_params(axis='x', colors='white')
plt.tick_params(axis='y', colors='white')
abline(1,0)
plt.show()
```

```
Model score: 0.72
Mean absolute error: 0.05
Root mean squared error: 0.08
R2: 0.72
```