mar/2021

Tecnológico de Monterrey

RETO

“OPTIMIZACIÓN DE ESCENARIOS EN PRODUCCIÓN”

María del Carmen Vargas Villarreal	A00828570
Renata Uribe Sánchez	A01274629
Diego Paasche Portillo	A01028103
Frida Sofía Bautista Martínez	A01235009

Docentes:

Laura Hervert Escoba

María de los Angeles Constantino

1.1

INTRODUCCIÓN

El propósito de este proyecto es presentar un análisis para la optimización de un escenario de producción, en el que utilizando un conjunto de datos reales, se construirá un modelo predictivo con condiciones asociadas a la problemática a resolver. Esta consiste en minimizar el gasto energético de la fabricación de productos de la industria del Socio Formador asignado: CEMEX Ventures.

CEMEX Ventures es una unidad de innovación abierta y Capital de Riesgo Corporativo de CEMEX, encargada de financiar estratégicamente a empresas emergentes (start-ups) de construcción, para impulsar, promover y revolucionar el futuro de esta industria.

El objetivo principal es analizar los datos proporcionados por el Socio Formador en donde se encuentran todas las variables relacionadas a la manufactura de productos y efectuar una serie de etapas consecutivas, las cuales consisten en el acercamiento al negocio, sus problemáticas, establecimiento de objetivos y un plan de trabajo; comprensión y exploración de los datos, pre procesarlos, modelarlos y evaluarlos, con el fin de predecir de manera precisa y garantizar una minimización de energía y mejoras en la fabricación y calidad del producto.

Esto se puede lograr mediante diversas estrategias, existen muchos modelos predictivos y técnicas de estadística que pueden facilitar la solución de este proyecto. Lo más importante es encontrar el modelo que mejor se adapte a los datos, no viceversa. Dentro de la metodología para encontrar dicho modelo se realizó una investigación breve sobre proyectos similares en donde expertos en estos temas, en artículos relacionados mencionan que utilizan distintos sistemas basados en Machine Learning para Python, métodos de limpieza de datos, modelos de regresión, redes neuronales, y demás.

2.1

CONOCIMIENTO DEL NEGOCIO

2.2

Objetivos del negocio

Minimizar el coste de las energías utilizadas en la fabricación de los productos, tomando en cuenta su calidad, la tasa de producción (el número de unidades de salida que la empresa pretende fabricar) y la dureza de las materias primas.

Debemos tomar en cuenta que un material con mayor dureza implica una producción más lenta, por lo que se recomienda utilizar energía eléctrica a consecuencia de un costo mayor. Estas consideraciones nos permiten analizar que tipo de energía sería preferible utilizar para reducir el costo, ya sea energía calórica (EC) o energía eléctrica (EE).

Se debe de encontrar un balance entre el uso de la EC y la EE para obtener un coste mínimo de manufactura, tomando en cuenta la correlación entre ambas.

$$\frac{Precio\ EC}{Precio\ EE} = 0.724$$

Toda la información utilizada para el análisis fue obtenida de un dataset otorgado por el socio formador, Cemex Ventures.

Marco teórico

Para conocer la aplicación de machine learning en la manufactura de productos, se investigaron diversos artículos:

En primer lugar, **Crop Yield Prediction Using Regression and Neural Networks** es un estudio reciente (CYPUR-NN) en el que se utilizaron datos históricos del rendimiento del arroz, donde las condiciones que se tomaron en cuenta fueron la humedad, luminiscencia y temperatura. Al integrar modelos de regresión y redes neuronales se pueden realizar predicciones, pronósticos y simulaciones efectivas, además de que detecta de manera simultánea posibles enfermedades difíciles de detectar. Este modelo facilita el trabajo de los agricultores para predecir el rendimiento por medio de una imagen o ingresando valores a través de una interfaz de web.

El siguiente artículo, **A Review of Data Cleaning Methods for Web Information System,** habla sobre métodos de limpieza de datos para facilitar el análisis de los mismos. Los autores enfocan su trabajo a la limpieza de datos del Web Information System (WIS), un sistema que es usado con frecuencia en el diario proveyendo servicios de información.

En el tercer artículo, **Communication Development and Verification for Python-Based Machine Learning Models for Real-Time Hybrid Simulation,** se busca analizar el uso de modelos de Machine Learning en Python para realizar Hybrid Simulations en tiempo real y probar su ventaja contra modelos de elementos finitos. Se demuestra que los modelos de Machine Learning brindan una mayor sensibilidad y éxito en las predicciones. Se probaron varios modelos de Machine Learning en datos lineales y no lineales donde se observó notable mejoría en la predicción de resultados con los modelos de Machine Learning.

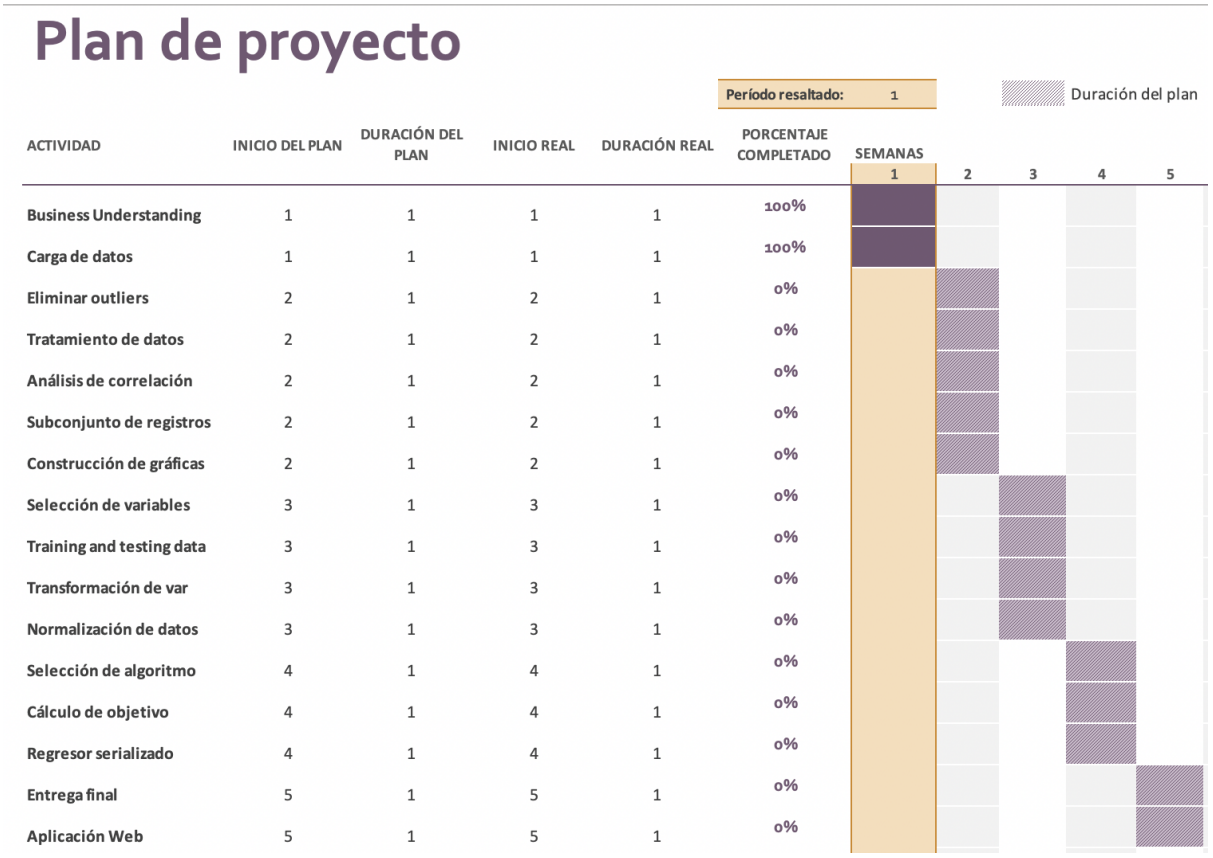
En el artículo de investigación **Machine Learning and Deep Learning Patentable Developments and Applications for Cost Reduction in Business and Industrial Organizations,** podemos observar un panorama más claro de la Industria 4.0 en México y de cómo son empleables distintos algoritmos de Machine Learning para la reducción de costos en empresas de manufactura. Esta investigación hace énfasis en explorar a industrias y organizaciones que han desarrollado patentes de algoritmos especializados en la reducción de costos con el fin de comprender y valorar los métodos utilizados.

Por último, en **A review of machine learning for the optimization of production processes** se explora la literatura de relevante de la última década acerca de algoritmos de optimización con machine learning, sus enfoques de optimización para la tasa de calidad de productos y la mejora de procesos de manufactura para las industrias de esta índole. Al mismo tiempo, la investigación describe la correlación entre la cantidad de datos utilizados y su optimización, haciendo un énfasis en los avances de investigaciones de este tipo en contraste con sus problemáticas.

2.3
Plan de proyecto

Al tratarse de un proyecto de minería de datos, se utilizará la metodología CRISP para un mejor desarrollo e implementación de la solución. De la misma manera, se hará uso de Scrum, la metodología ágil que permite una mejor organización de equipos de trabajo para la planificación de proyectos.

Diagrama de GANTT
A continuación se muestra el plan de trabajo del proyecto:



3.1

COMPRENSIÓN DE LOS DATOS DEL NEGOCIO

Después de importar la base de datos en una Jupyter Notebook, estos se deben analizar primeramente en cuestión de limpieza, es decir, reconocer los datos significativos, los que no aportan valor, identificar si los valores faltantes, encontrar inconsistencias y demás.

Una vez realizado lo anterior, la base de datos se analiza estadísticamente, encontrando tendencias y relaciones que posteriormente nos permitirán encontrar tendencias de distribución para elegir el modelo de predicción a implementar.

El tipo de arreglo que se utilizó fue Data Frame ya que este tipo de estructura es esencial en la minería de datos y facilita la búsqueda, limpieza y manejo de la información para realizar un análisis preliminar. Al imprimir los datos, 7 columnas se desplegaron en forma de tablero: TIME, Dureza, Tasa_Prod, Asp, EC, EE y Calidad.

	TIME	Dureza	Tasa_Prod	Asp	EC	EE	Calidad
0	01/01/1995 0:00	100.0	368	2.78	15.1	29.7	0.053
1	02/01/1995 0:00	100.0	426	3.00	26.9	0.0	0.108
2	03/01/1995 0:00	101.0	446	3.00	29.5	0.0	0.098
3	04/01/1995 0:00	99.0	395	3.00	18.9	25.4	0.056
4	05/01/1995 0:00	102.0	380	3.00	17.5	26.6	0.051

El tipo de dato (estos pueden ser int, float, etc) asignado a cada columna fueron los siguientes: Dureza, Asp, EC, EC y Calidad corresponden a floats, mientras que Tasa_Prod fue la única columna con tipo de dato int.

En el lado cuantitativo, por cada columna se encontraron 9392 registros, a excepción de las columnas Dureza y Aspiración, estás contando con 9391 registros debido a que se presentó un valor vacío respectivamente en cada columna. Los valores vacíos encontrados se desplegaron de la siguiente manera:

```
TIME         0
Dureza       1
Tasa_Prod    0
Asp          1
```

```
EC           0
EE           0
Calidad      0
dtype: int64
```

Estos datos vacíos, al igual que otros factores del arreglo, deben de ser modificados antes de comenzar a trabajar con el data frame.

4.1

PREPARACIÓN DE LOS DATOS

4.2

Eliminar datos vacíos

Gracias a que los datos vacíos eran pocos (2), el equipo optó por eliminarlos utilizando una función de la librería de Pandas, llamada *dropna()*. Después de eliminar los datos vacíos, se obtuvieron las métricas básicas del arreglo inicial: media, desviación estándar, los cuartiles y los valores mínimos y máximos para cada columna.

	Dureza	Tasa_Prod	Asp	EC	EE	Calidad
count	8874.000000	8874.000000	8874.000000	8874.000000	8874.000000	8874.000000
mean	104.087559	391.821501	3.156699	19.497465	19.258057	0.093265
std	2.045327	41.875131	0.368985	6.655789	7.976253	0.048100
min	80.000000	0.000000	0.090000	0.000000	0.000000	0.039000
25%	103.000000	384.000000	3.040000	16.100000	14.600000	0.064000
50%	104.000000	398.000000	3.260000	19.300000	20.200000	0.084000
75%	105.000000	409.000000	3.380000	23.700000	25.400000	0.109000
max	112.000000	480.000000	3.520000	40.400000	35.300000	1.000000

Otra de las descripciones cuantitativas del socio formador, es el intervalo aceptable para la calidad (Q) que se encuentra entre 0.5 y 1. Al observar la media de la calidad esta se encuentra alrededor de 0.093, lo que significa que está fuera del límite inferior. Se infiere que esto indica que la calidad en general es mala. Considerando los valores máximos y mínimos

de la dureza y observando la media, podemos concluir que se fabrican materiales con una dureza alta.

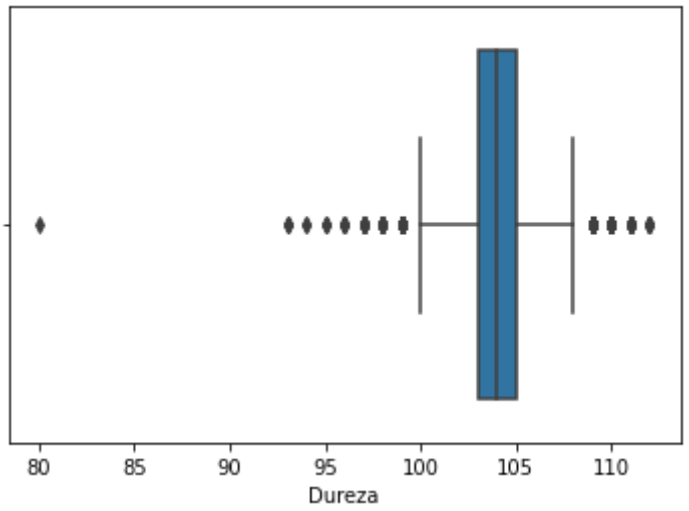
En cuanto a la desviación estándar para la dureza y la calidad, se observa que resulta en valores más pequeños, por lo que los datos no se alejan demasiado de la media. Esto quiere decir que la dureza y la calidad tienden a mantenerse en un rango constante en la producción.

No tienen una tasa de producción constante, debido a que el valor mínimo de productos fabricados al día es 0 y el máximo 480, y a juzgar por su desviación estándar esta confirma un alto porcentaje de inestabilidad en cuanto a la tasa de producción diaria. Además, la cantidad mínima de energía utilizada (tanto de EC y EE) siendo cero explica el por qué hay días en los que no se fabrican productos. Al igual que se puede observar que en promedio, ambas energías son utilizadas en la misma cantidad.

4.3
Eliminación de datos atípicos

Para trabajar adecuadamente con una base de datos, es necesario realizar un filtrado de la misma. Este filtrado incluye la eliminación de los datos atípicos. Un dato atípico (outlier) es aquel que es numéricamente distinto a los demás o bien, que se encuentra muy alejado del resto de los datos. Estos datos son sencillos de identificar ya que se tienen medidas estadísticas que los resaltan.

Dentro de la base de datos, se encontraron demasiados datos atípicos para cada columna del dataset. Estos datos atípicos se identificaron al realizar una proyección de boxplot de cada una de las columnas. Un ejemplo de las gráficas desplegadas en el código es el siguiente:



Los puntos individuales desplegados son los datos atípicos que el equipo eliminó después de identificarlos. Se puede apreciar que esta gráfica contiene 12 datos atípicos. Al igual que la columna de dureza, todas presentan múltiples outliers que deben de ser excluidos. Para el proceso de eliminación de estos datos se utilizaron los valores de z. Tomando en cuenta que la distribución de los datos es normal, se sabe que la mayoría de estos no se encontrarán más

dispersos que dos desviaciones estándar de la media. Por tanto, la nueva base de datos solo toma los datos del conjunto que tengan valores de z menores a 2.5. Al eliminar todos estos valores, el tamaño de la base de datos se reduce a 7990 entradas.

4.4
Preprocesamiento de datos

Para facilitar la toma de decisiones, la base de datos debe de ser reducida hasta considerar únicamente los datos más valiosos numéricamente. Este preprocesamiento se realiza con la librería de *preprocessing* de sklearn. Anterior a esto, se agregó una columna para costo total, costo ponderado y el costo ponderado de cada energía. La columna de costo total se calculó con la relación de costos mencionada en la sección 2.1. La suma de la energía eléctrica más la energía calórica multiplicada por 0.724 nos da el costo total. Para el cálculo de la columna de costo ponderado, únicamente se divide el valor obtenido del costo total entre la tasa de producción. Además, las columnas de costo ponderado por energía se calculan con la división del valor de cada energía entre la tasa de producción.

La reducción de la base de datos se utilizó con la siguiente línea:

```
df_OPT =
df_sin_out.loc[df_sin_out.groupby('Calidad').costoPonderado.idxmin()]
```

Esta línea reduce la base de datos a 175 líneas. La nueva base de datos está agrupada por valor de calidad, teniendo secciones con valores de energía distintos pero manteniendo la calidad constante por grupo.

Time	Dureza	Tasa_Prod	Asp	EC	EE	Calidad	costo Total	costo Ponderado	EE_Ponderada	EC_Ponderada
01/01/1995 0:00	100.0	368	2.78	15.1	29.7	0.053	40.6324	0.110414	0.080707	0.041033
01/01/1996 0:00	106.0	427	3.42	26.6	16.4	0.083	35.6584	0.083509	0.038407	0.062295
01/01/1997 0:00	103.0	434	3.42	27.4	11.5	0.085	31.3376	0.072206	0.026498	0.063134

01/01/1998 0:00	103.0	395	2.82	24.5	16.4	0.074	34.1380	0.086425	0.041519	0.062025
01/01/1999 0:00	106.0	388	3.32	26.2	19.1	0.111	38.0688	0.098115	0.049227	0.067526

Estas son las primeras 5 líneas de la base de datos optimizada.

4.5
Sets de training y testing

Posterior a la preparación de los datos considerando el data frame optimizado, es esencial generar una separación aleatoria en dos conjuntos de datos para entrenar y probar los modelos próximos a utilizar. Se necesitan dos arrays para trabajar sobre ellos. El primero es llamado "x" y contiene las variables de entrada: calidad, dureza y tasa de producción. El segundo array "y" es el que se desea optimizar, y contiene la energía eléctrica, energía calórica y el costo ponderado. A partir de la definición de cada variable, se genera una división de la base de datos en 4 grupos. Un grupo de entrenamiento y otro de prueba para cada una de las variables. La separación no es equitativa ya que el porcentaje de datos de entrenamiento debe de ser mayor. El tamaño del grupo de entrenamiento será del 70% del total de datos, mientras que el set de prueba representa el 30%. En seguida se presenta el tamaño de cada grupo:

```
X_train :  (122, 3)
X_test :  (53, 3)
y_train :  (122, 3)
y_test :  (53, 3)
```

5.1

MODELACIÓN

5.2
Árboles de decisión

Uno de los modelos que implementamos fueron los árboles de decisión. Este modelo es de aprendizaje supervisado y se inicia con un 'nodo' o parámetro inicial del cual salen 'ramificaciones' para presentar los demás parámetros. El árbol continúa expandiéndose hasta que último nodo tenga como ramificación una 'hoja'. Estas hojas nos darán la clasificación o valor de regresión final para los datos analizados.
Nuestros datos implican un modelo de regresión por lo que tuvimos que importar *DecisionTreeRegressor* de sklearn. Esta función nos permitirá generar y entrenar el modelo de árboles de decisión. Sin embargo, debido al tamaño de nuestros datos de salida (tres variables) tuvimos que implementar *MultiOutputRegressor* en conjunto para la modelación.

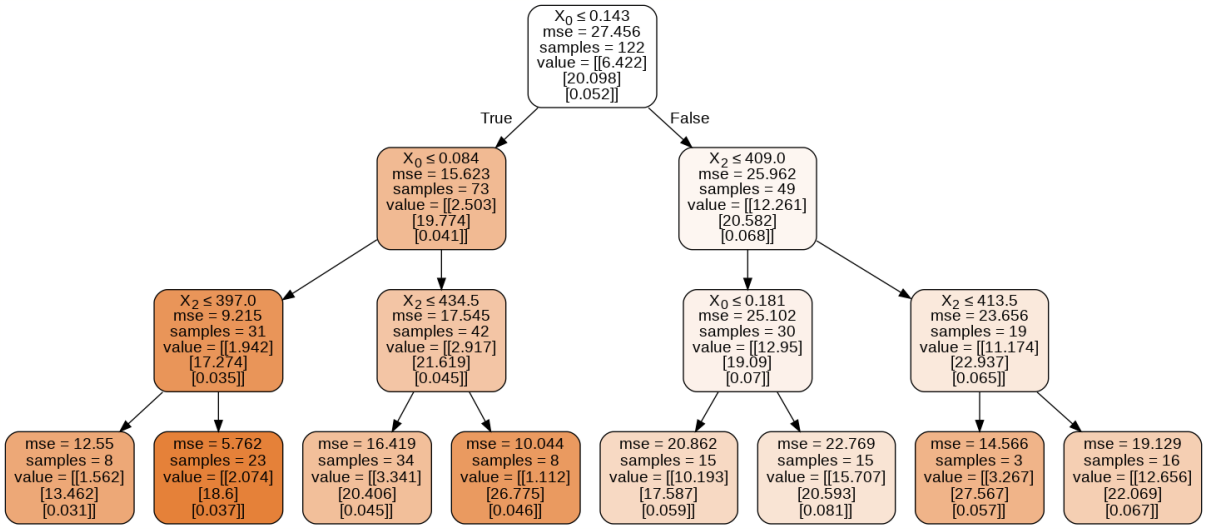
Las siguientes líneas fueron el código utilizado para la implementación:

```
clf = DecisionTreeRegressor(max_depth=3)          #Declaración de los modelos
mclf = MultiOutputRegressor(clf)
mclf.fit(X_train, Y_train)                        #Entrenamiento del modelo
y_pred = mclf.predict(X_test)                     #Predicción del modelo
```

Utilizando la librería *metrics* de sklearn evaluamos el modelo, calculando su Error Absoluto Medio, Error Medio al Cuadrado, Raíz del Error Cuadrático Medio y el valor de Regresión Cuadrada.

```
Mean Absolute Error: 3.92907758126915
Mean Squared Error: 42.10024663840062
Root Mean Squared Error: 6.488470284928538

mclf.score(X_test, Y_test)
-0.5212789646312606
```

El valor de Regresión Cuadrada presentó un valor negativo, esto puede ser debido a algún error en los datos o porque los datos con los que se cuenta no se encuentran relacionados. Para finalizar utilizamos la librería *graphviz* para la visualización del árbol de decisión.



5.3
Support Vector Machine

El modelo de SVM puede ser utilizado para clasificación, regresión y en la detección de datos atípicos. El modelo consiste en usar machine learning para que el programa detecte la mejor división de los datos posibles. El modelo es de aprendizaje supervisado y para la solución del problema, se utilizó el modelo de regresión. Dentro de la librería de sklearn se encuentra la extensión llamada *SVM*.
Como el resto de los modelos, es necesario utilizar un *MultiOutputRegressor* que permita a cada modelo trabajar con más de una variable de salida. Las siguientes líneas son con las cuales se crea el modelo y se establecen las variables a utilizar:

```
svclassifier = svm.SVR()                          #Declaración de los modelos
msvc = MultiOutputRegressor(svclassifier)
msvc.fit(X_train, Y_train)                        #Entrenamiento del modelo
y_pred = msvc.predict(X_test)                     #Predicción del modelo
```

Las variables de salida son las establecidas en "y", como se mencionó anteriormente. Al definir el *svrlassifier()* el modelo está listo para correr. Y para definir el rendimiento del modelo se deben de utilizar métricas estadísticas como es el error medio absoluto, el error cuadrado medio y la raíz del error cuadrado medio. Esto permite conocer qué tan separadas están las variables de predicción a las reales. Estos son los resultados obtenidos con el modelo de SVM:

```
Mean Absolute Error: 3.385401354641868
Mean Squared Error: 32.68232401399914
Root Mean Squared Error: 5.716845634963318
```

Y para conocer el valor de correlación que otorga el modelo (R^2) se obtiene el score de los grupos de entrenamiento tanto de *x* como de *y*. El valor obtenido para este indicador es el siguiente:

```
msvc.score(X_test, Y_test)
-0.4357610390318198
```

Se puede observar que este valor es negativo cuando en realidad. Los valores de correlación varían entre 0, siendo menos relacionados y 1, perfectamente relacionados. El hecho de que el valor obtenido sea negativo indica que puede haber un error en los datos o que simplemente no existe relación alguna entre los mismos.

5.4
XG Boost

El siguiente y último modelo que se programó fue XGBoost, un algoritmo de machine learning supervisado. Este es una implementación del algoritmo de árboles de decisiones impulsados por gradientes (gradient boosting), donde intenta predecir con precisión una variable objetivo, combinando estimaciones de un conjunto de modelos débiles o más simples.

Entrando más a detalle, el principio de boosting (algoritmo de optimización gradient boosting) genera modelos de predicción débiles (en este caso son nuestros árboles de decisión) de manera secuencial, va tomando y potenciando los resultados de estos. Durante el proceso de entrenamiento, los parámetros se van ajustando iterativamente tratando de encontrar el mínimo de la función objetivo, en este caso utilizamos la raíz del cuadrático medio (RMSE).

Es un proceso comparativo de modelos, donde se generan una serie de repeticiones hasta encontrar el mejor modelo posible, o cuando se llega al número máximo de iteraciones establecidas por el usuario. Aunque no poner un límite permite encontrar una mayor posibilidad del resultado, puede ser que resulte en un "overfitting".

Al momento de la implementación del código, se importó xgboost de la librería sklearn. Como se ha mencionado anteriormente en la descripción de los modelos pasados, también se aplicó un *MultiOutputRegressor* para tomar en cuenta las 3 variables de salida correspondientes. Código utilizado:

```
xg = xgb.XGBRegressor()                    # Declaración de los modelos
mxg = MultiOutputRegressor(xg)
mxg.fit(X_train, Y_train)                  # Entrenamiento del modelo
y_pred = mxg.predict(X_test)               # Predicción del modelo
```

Para medir el rendimiento de este modelo, también se siguió el formato de obtener los errores Error Absoluto Medio, Error Medio al Cuadrado, Raíz del Error Cuadrático Medio y el valor de Regresión Cuadrada.

```
Mean Absolute Error: 2.920816997883931
Mean Squared Error: 22.88587018875721
Root Mean Squared Error: 4.783917870193552
```

El valor para el coeficiente de determinación (R^2) es el porcentaje que nos indica que tan bien se pueden predecir los resultados obtenidos, por lo que igualmente se calculó a continuación:

```
mxg.score(X_test, Y_test)
0.18501613868526487
```

A comparación de los dos modelos anteriores, en XGBoost la R cuadrada sí fue positiva, con un valor bajo, sin embargo, se mantiene dentro del rango entre 0 y 1, por lo que puede indicar cierto grado de variación y correlación entre los datos.

5.5
Hiperparámetros

Para mejorar el rendimiento de los modelos, se utilizaron hiperparámetros en cada uno. Los hiper parámetros son aquellos que se pueden manipular con el objetivo de, a través de prueba y error, encontrar los mejores valores de cada parámetro para obtener una predicción más acertada.
Primero es necesario conocer cuáles son los parámetros que existen en el modelo para, a partir de ello, poder utilizar los hiper parámetros e identificar los mejores valores. Los parámetros de cada modelo se encuentran con la función *get_params()* que regresa todos los existentes. Este es un ejemplo de los parámetros de los árboles de decisión:

{'estimator__ccp_alpha': 0.0, 'estimator__criterion': 'mse', 'estimator__max_depth': None, 'estimator__max_features': None, 'estimator__max_leaf_nodes': None, 'estimator__min_impurity_decrease': 0.0, 'estimator__min_impurity_split': None, 'estimator__min_samples_leaf': 1, 'estimator__min_samples_split': 2, 'estimator__min_weight_fraction_leaf': 0.0, 'estimator__random_state': None, 'estimator__splitter': 'best', 'estimator': DecisionTreeRegressor(), 'n_jobs': None}

Para la prueba y error de cada valor se utiliza la función de *GridSearch()* desde la librería de *Sklearn.Model_Selection*. La función necesita de varias opciones de valores los cuales evaluará en el modelo especificado y regresa el mejor. En el modelo de árboles de decisión se definieron los siguientes parámetros con sus posibles valores:

```
param_grid = { 'max_depth': [1,2,3], 'max_features': [1,2,3],'min_samples_split':[2,3,4,5,6,7], 'min_samples_leaf': [2,3,4,5,6,7] }
```

Y el código regresa el mejor valor encontrado para R^2, cada uno de los parámetros y el tiempo en el que se corrió el programa. Estos son los resultados:

Árboles de decisión	0.18849154724873984 {'max_depth': 2, 'max_features': 2, 'min_samples_leaf': 3, 'min_samples_split': 7} Execution time: 1.600923776626587 ms
SVM Regressor	-0.43665171035731415 {'estimator__C': 10, 'estimator__gamma': 0.0001, 'estimator__kernel': 'rbf'}

	Execution time: 0.396320104598999 ms
XGBoost	-0.05873280189374975 {'estimator__base_score': 3, 'estimator__booster': 'gbtree', 'estimator__colsample_bylevel': 1} Execution time: 0.5867464542388916 ms

6.1

EVALUACIÓN DE MODELOS

6.2

Evaluación de resultados

Para dar parte a una selección adecuada del algoritmo predictor, se obtuvieron las siguientes métricas: Error Medio Absoluto (MAE), Error Cuadrático Medio (MSE), Raíz del Error Medio Cuadrático (RMSE) y el Coeficiente de Determinación (R^2).
A continuación se muestra una tabla con los respectivos valores de cada algoritmo:
Tabla comparativa de errores:

Modelo	MAE	MSE	RMSE	Score
Support Vector Machine	3.4136	32.2253	5.6767	-0.4366517 1035731415
Árboles de decisión	2.8880	20.6189	4.5408	0.18849154 724873984
XG Boost	2.9208	22.8858	4.7839	-0.0587328 0189374975

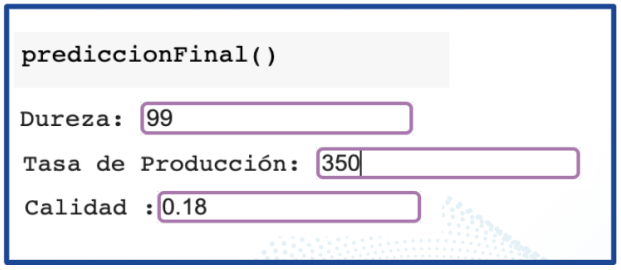
Como se puede observar en la anterior tabla, el algoritmo que presentó los errores más pequeños además de un coeficiente de determinación positivo, fue el árbol de decisión, por lo tanto fue el modelo elegido para la generación final de predicciones.

IMPLANTACIÓN DE SOLUCIÓN

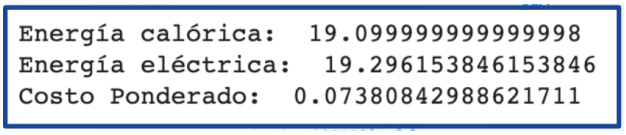
7.1 Interfaz de predicción con Árbol de Decisión

En python fue creada una función que primeramente solicita al usuario introducir los siguientes datos: dureza, la tasa de producción y la calidad de un producto y a partir de ellos, genera una predicción para la cantidad de energía calórica, energía eléctrica y costo ponderado. En seguida se muestra un ejemplo de introducción de datos y su respectiva predicción:

Datos de entrada:



Predicción:



CONCLUSIÓN

7.2
Individuales

María del Carmen Vargas Villarreal
Para mí, este reto me ayudó a vivir una pequeña introducción al mundo de los datos aplicados en un campo laboral real, en el que se tuvo un acercamiento muy directo con el socio formador que proporcionó la problemática a resolver y aclaró los puntos importantes antes de empezar a trabajar con los datos e ir desarrollando cada etapa. En este caso, fue CEMEX Ventures quien nos dio una base de datos ambientada en la fabricación de productos, y todas las variables que forman parte de este proceso.

Fue muy interesante aprender cómo se inicia toda la metodología que conlleva el implementar modelos de predicción y llegar a la meta final, tras distinguir cuál modelo resultó ser el mejor, y aplicarlo con los datos proporcionados para generar el resultado más óptimo.

Fue todo un proceso que se resume en observar, interpretar, visualizar, limpiar, escalar, ajustar, modelar, calcular, evaluar, y lo más importante, resolver problemas que se vayan presentando al momento de manipular estos datos. Sin embargo, son situaciones que pasan también en la vida real, por lo que aprender cómo manejarlo y reconocer desde dónde empieza la raíz del problema es clave. Entrando a más en detalle, me gustó mucho conocer sobre los distintos algoritmos de Machine Learning de tipo supervisado que se pueden utilizar

para predecir, y el interpretar cuál modelo es el que se ajusta adecuadamente a nuestros datos es una base fundamental para un resultado óptimo.

Finalmente, el optimizar la energía requerida para poder fabricar un producto con costos mínimos fue el objetivo que impulsó toda la implementación, en este caso, el entender qué es lo que el negocio pide y comprender los datos como tal es clave, por lo que sin una base sólida de entendimiento del problema y el asesoramiento de las profesoras no habría sido posible resolver este reto efectivamente y aprender mucho de él.

Renata Uribe Sánchez

Este reto representó nuestro primer acercamiento académico a la Ciencia de Datos concretamente. Abordar una problemática como la optimización de energías y reducción de costos en la manufactura de productos definitivamente resultó ser un gran desafío por todo el proceso que conlleva la realización de un proyecto de minería de datos con la metodología CRISP.

Primeramente nos adentramos en la problemática para comprender los objetivos y propósitos solicitados por el Socio Formador, para después poder hacer un mejor análisis de la base de datos. La preparación de datos fue el proceso más extenso de la situación problema, pues nos apoyamos de diversos métodos y procedimientos para efectuar una correcta limpieza de datos que nos permitiera un mejor rendimiento de modelos. En esta sección se tomaron múltiples decisiones importantes que finalmente influyeron significativamente en las predicciones finales.

Por otro lado, la implementación de los modelos *Support Vector Machine, XG Boost y Decision Tree* significó el paso más importante y el más interesante de ejecutar como futuros científicos de datos, al utilizar los algoritmos de machine learning recién mencionados. Gracias a los distintos hiper parámetros que produjeron una mejora de precisión para cada modelo, finalmente seleccionamos el algoritmo con los errores más pequeños: el Decision Tree. Por último, a pesar de no generar una web app en flask, si conseguimos simular una interfaz en la que se solicitaran datos al usuario para la producción de un producto y el algoritmo posteriormente generara las predicciones.

Frida Sofía Bautista Martínez

En este reto tuvimos la oportunidad de trabajar con datos de CEMEX Ventures, los cuales pertenecían a una problemática que tuvieron dentro de la empresa. Esta consiste en los costos de energía que involucra el diseño de cierta pieza. Fue muy interesante trabajar con los datos de una empresa y en un problema real. Nos pudimos encontrar con errores que suceden en la práctica como por ejemplo el hecho de que no había correlaciones entre nuestras variables a analizar, pero que pese a esto se tiene que sacar el modelo adelante e implementarlo de la mejor manera. Con este proyecto también exploramos la implementación de varios modelos de machine learning y lo que hay detrás de ellos. Tuvimos la oportunidad de aprender a

modificarlos para obtener mejores modelos, pero entre lo más importante también se encuentra el que aprendimos mejores maneras de manipulación de datos. Pudimos ver las bases de las estructuras de datos, los distintos tipos y cómo manejarlas. Comprendimos la importancia de la integridad de los datos y la ética de manejarlos, pero también vimos formas de manipularlos y adaptarlos mejor a nuestros modelos para obtener resultados óptimos como normalizarlos o escalarlos.

Diego Paasche Portillo
El reto me ayudó mucho a desarrollar habilidades que necesita un científico de datos. Sin embargo, me pareció sorprendente encontrar que los datos no tenían relación entre sí. Creo que es extraño ya que esta problemática se supone que ya fue resuelta y nuestro socio formador no es nadie más que CEMEX Ventures. Por otro lado, aprendí a utilizar métodos de machine learning muy interesantes y creo que los seguiré usando por mucho tiempo. El hecho de que estos métodos puedan ser utilizados al mismo tiempo que otros para que en cierta forma "unan fuerzas" para crear un sistema de optimización mucho más completo y preciso me llamó mucho la atención. Me gusta que en los sistemas inteligentes no exista individualidad, se puede utilizar de todo para llegar al resultado sin comprometer la calidad del output. Hablando específicamente del reto, como equipo descubrimos que el método de árboles de decisión junto con los hiper parámetros nos dió el mejor resultado con los errores más pequeños. Sin importar el hecho de que no exista relación en los datos. El primer bloque de mi carrera como IDM me gustó mucho a pesar de la situación actual en la que nos encontramos.

7.3
General

En este reporte se analizó la problemática presentada por CEMEX relacionada con los costos de producción de piezas en relación con el material y tipo de energía utilizada, ya sea calórica o eléctrica. Realizamos un análisis inicial de los datos en el cual limpiamos nuestros datos y eliminamos los valores atípicos. Después también optimizamos nuestros datos esto fue reduciendo nuestra base de datos de acorde a 'Costo_Ponderado' manteniendo solo los datos más valiosos lo que optimizaría nuestra modelación. Los modelos de predicción implementados, todos de aprendizaje supervisado, fueron Árboles de Decisión, Support Vector Machine y XG Boost. Tuvimos que analizar las variables de 'Dureza', 'Calidad' y 'Tasa_Prod' para poder generar las predicciones de 'Energía Calórica', 'Energía Eléctrica' y 'Costo_Ponderado'. Al analizar los datos y los resultados de los modelos pudimos ver que había poca relación entre las variables y es por eso que el valor de Regresión Cuadrada nos arroja valores negativos para todos los modelos. Sin embargo continuamos con la generación de los modelos y ajustamos sus Hiperparametros para obtener el mejor resultado de predicción. Al final concluimos que el modelo que mejor se adaptó a nuestros datos fue el Árbol de Decisión con los hiperparametros previamente mencionados. Este modelo fue el que nos dio errores menores en comparación a los demás y creemos que será el que mejor evalúa y genera predicciones de la situación. Anexado en el código se encuentra una función que predice los resultados y costos una vez sean recibidos unos datos de entrada, esto deja una

implementación que le permite al usuario obtener una predicción óptima de los costos de producción de una pieza.

8.1

Referencias

Wang, J., Wang, X., Yang, Y., Zhang, H., & Fang, B. (2020). A Review of Data Cleaning Methods for Web Information System. CMC-COMPUTERS MATERIALS & CONTINUA, 62(3), 1053–1075. https://www.techscience.com/cmc/v62n3/38341

Cornell University, Ramesh, S., Hebbar, A., Yadav, V., Gunta, T., & Balachandra, A. (2020). Base de datos: arXiv. CYPUR-NN: Crop Yield Prediction Using Regression and Neural Networks. https://arxiv.org/abs/2011.13265

Bas E. & Moustafa M. (2020) . Communication Development and Verification for Python-Based Machine Learning Models for Real-Time Hybrid Simulation. Frontiers in Built Environment, 6. https://doi.org/10.3389/fbuil.2020.574965

Enriquez H. & Roque E. (2020) Machine Learning and Deep Learning Patentable Developments and Applications for Cost Reduction in Business and Industrial https://0-eds-a-ebscohost-com.biblioteca-ils.tec.mx/eds/pdfviewer/pdfviewer?vid=5&sid=484c4ad9-7411-4baa-b509-9e55c5b8ffe9%40sessionmgr4008

Weichert D., Link P., Stoll A. Ruping S. Ihlenfeldt S. & Wrobell S. (2019) A review of machine learning for the optimization of production processes https://0-eds-a-ebscohost-com.biblioteca-ils.tec.mx/eds/pdfviewer/pdfviewer?vid=19&sid=ae041445-0cf3-45f3-9f73-77690f871f59%40sessionmgr4006

9.1

Anexos

Autores

Integrante del equipo	Rol de Trabajo
Frida Bautista	- Código (hiperparámetros) - Modelos (reporte) - Conclusión (reporte)
Diego Paasche	- Código (limpieza datos) - Modelos (reporte) - Preparación de datos (reporte)
MaryCarmen Vargas	- Código (métodos)

	- Comprensión de Datos del Negocio (reporte) - Presentación
Renata Uribe	- Código (interfaz) - Business Understanding (intro reporte) - Evaluación (reporte)

cargaDatos

March 12, 2021

1 Carga de datos

[1]:
```
from google.colab import drive
```

[2]:
```
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

Importación de las librerías:.

[3]:
```
import pandas as pd # Libreria para data frames
import numpy as np # Libreria para operaciones matemáticas
import matplotlib.pyplot as plt
```

Carga de la base de datos y creación del data frame:

[4]:
```
df = pd.read_csv('/content/drive/Shareddrives/EQUIPO RETO CIENCIA DE DATOS/
↪datos_tec.csv')
df.head()
```

[4]:

	TIME	Dureza	Tasa_Prod	Asp	EC	EE	Calidad
0	01/01/1995 0:00	100.0	368	2.78	15.1	29.7	0.053
1	02/01/1995 0:00	100.0	426	3.00	26.9	0.0	0.108
2	03/01/1995 0:00	101.0	446	3.00	29.5	0.0	0.098
3	04/01/1995 0:00	99.0	395	3.00	18.9	25.4	0.056
4	05/01/1995 0:00	102.0	380	3.00	17.5	26.6	0.051

Conteo de registros por columna:

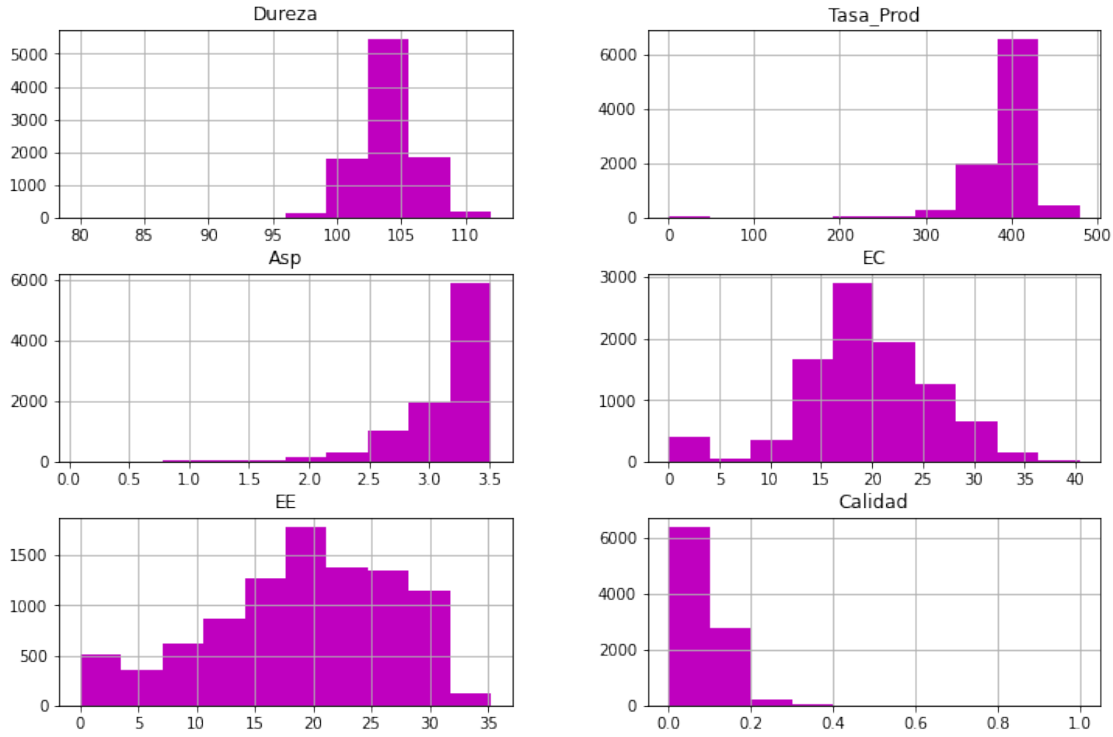
[5]:
```
df.shape
```

[5]:
```
(9392, 7)
```

[6]:
```
df.hist(color="m", figsize=(12,8))
```

[6]:
```
array([[<AxesSubplot:title={'center':'Dureza'}>,
        <AxesSubplot:title={'center':'Tasa_Prod'}>],
```

```
[<AxesSubplot:title={'center':'Asp'}>,
 <AxesSubplot:title={'center':'EC'}>],
[<AxesSubplot:title={'center':'EE'}>,
 <AxesSubplot:title={'center':'Calidad'}>]], dtype=object)
```



2 Limpieza de Datos

Identificación de valores nulos en la base de datos:

[7]:
```
df.isnull().values.any()
```

[7]:
```
True
```

Conteo de valores nulos por columna:

[8]:
```
df.isnull().sum()
```

[8]:
```
TIME          0
Dureza        1
Tasa_Prod     0
Asp           1
EC            0
```

```
EE           0
Calidad      0
dtype: int64
```

Se eliminan las filas en las que se encuentra algún valor nulo, en este caso solo existen dos, por lo que es un cambio mínimo y el análisis no se verá afectado con dos registros menos.

[9]:
```
df_sin_out = df.dropna(axis=0,how='any', thresh=None, subset=None)
```

Se comprueba que ya no existen valores nulos:

[10]:
```
df_sin_out.isnull().sum()
```

[10]:
```
TIME         0
Dureza       0
Tasa_Prod    0
Asp          0
EC           0
EE           0
Calidad      0
dtype: int64
```

[11]:
```
df_sin_out.shape #Sin null values
```

[11]:
```
(9390, 7)
```

[12]:
```
df_sin_out = df_sin_out[df_sin_out['Calidad'] >=  0.039]
```

[13]:
```
df_sin_out.shape
```

[13]:
```
(8874, 7)
```

Revisamos el tipo de datos que contiene cada columna:

[14]:
```
df_sin_out.dtypes
```

[14]:
```
TIME          object
Dureza       float64
Tasa_Prod      int64
Asp          float64
EC           float64
EE           float64
Calidad      float64
dtype: object
```

3 Obtención de las métricas básicas del arreglo inicial

[15]:
```
df_sin_out.median()
```

[15]:
```
Dureza       104.000
Tasa_Prod    398.000
Asp            3.260
EC            19.300
EE            20.200
Calidad        0.084
dtype: float64
```

[16]:
```
df_sin_out.max()
```

[16]:
```
TIME         31/12/2019 0:00
Dureza                  112
Tasa_Prod               480
Asp                    3.52
EC                     40.4
EE                     35.3
Calidad                   1
dtype: object
```

[17]:
```
df_sin_out.min()
```

[17]:
```
TIME         01/01/1995 0:00
Dureza                   80
Tasa_Prod                 0
Asp                    0.09
EC                        0
EE                        0
Calidad               0.039
dtype: object
```

[18]:
```
df_sin_out.std()
```

[18]:
```
Dureza        2.045327
Tasa_Prod    41.875131
Asp           0.368985
EC            6.655789
EE            7.976253
Calidad       0.048100
dtype: float64
```

[19]:
```
df_sin_out.quantile([0.25,0.5,0.75])
```

[19]:
	Dureza	Tasa_Prod	Asp	EC	EE	Calidad
0.25	103.0	384.0	3.04	16.1	14.6	0.064
0.50	104.0	398.0	3.26	19.3	20.2	0.084
0.75	105.0	409.0	3.38	23.7	25.4	0.109

[20]:
```
df_sin_out.describe()
```

[20]:
	Dureza	Tasa_Prod	…	EE	Calidad
count	8874.000000	8874.000000	…	8874.000000	8874.000000
mean	104.087559	391.821501	…	19.258057	0.093265
std	2.045327	41.875131	…	7.976253	0.048100
min	80.000000	0.000000	…	0.000000	0.039000
25%	103.000000	384.000000	…	14.600000	0.064000
50%	104.000000	398.000000	…	20.200000	0.084000
75%	105.000000	409.000000	…	25.400000	0.109000
max	112.000000	480.000000	…	35.300000	1.000000

[8 rows x 6 columns]

4 Outliers

Para identificar los outliers de cada columna:

[21]:
```
import seaborn as sns # Libreria de visualización
import matplotlib.pyplot as plt
from scipy import stats
```

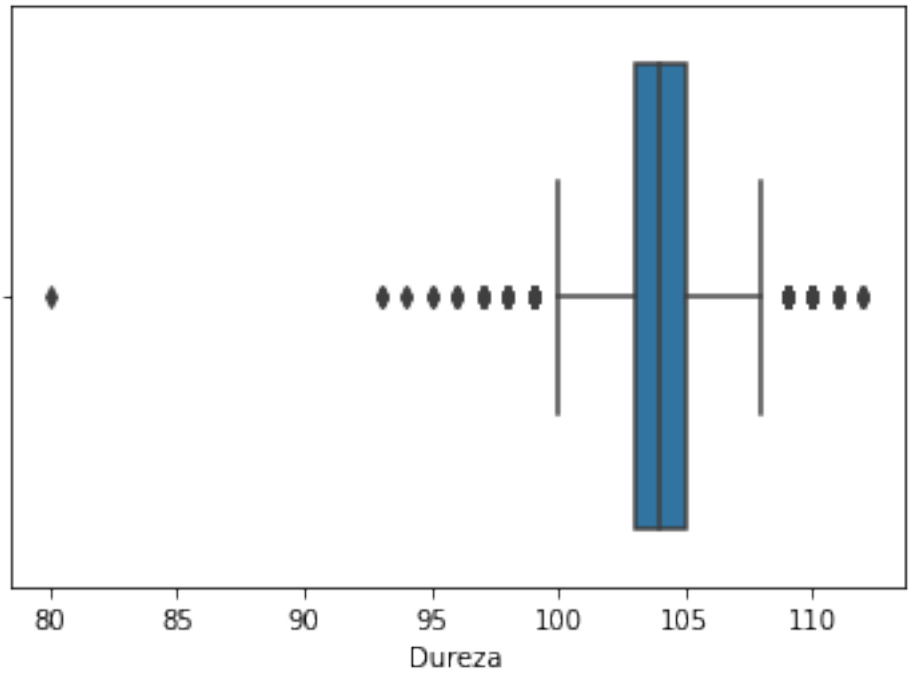
[22]:
```
Q1 = df_sin_out.quantile(0.25)
Q3 = df_sin_out.quantile(0.75)
IQR = Q3 - Q1
```

[23]:
```
((df_sin_out < (Q1 - 1.5 * IQR)) | (df_sin_out > (Q3 + 1.5 * IQR))).sum()
```

[23]:
```
Asp          497
Calidad      400
Dureza       275
EC           405
EE             0
TIME           0
Tasa_Prod    492
dtype: int64
```

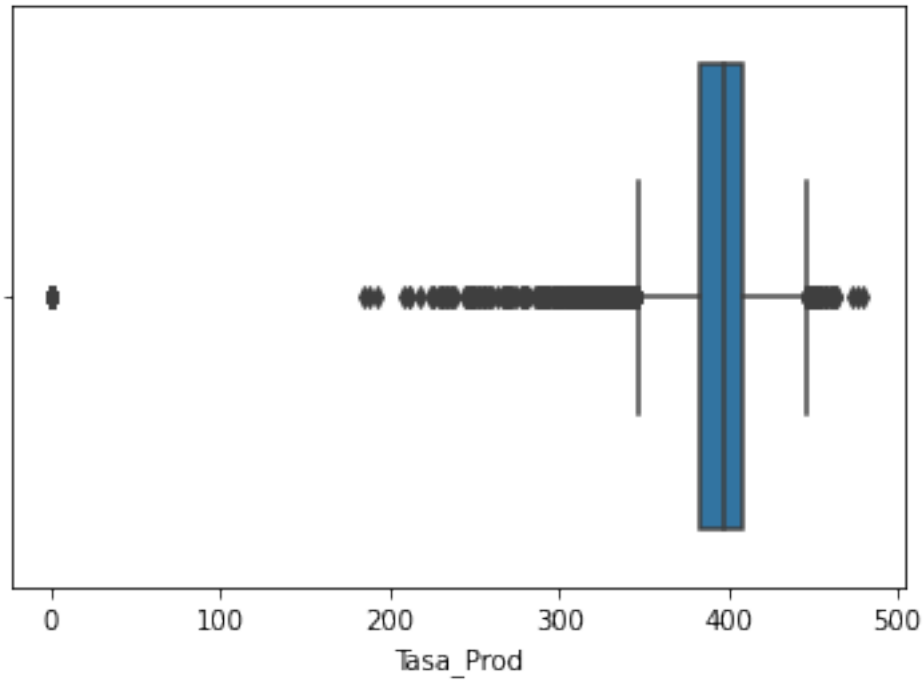
[24]:
```
sns.boxplot(x=df_sin_out['Dureza'], showfliers =True)
```

[24]:
```
<AxesSubplot:xlabel='Dureza'>
```



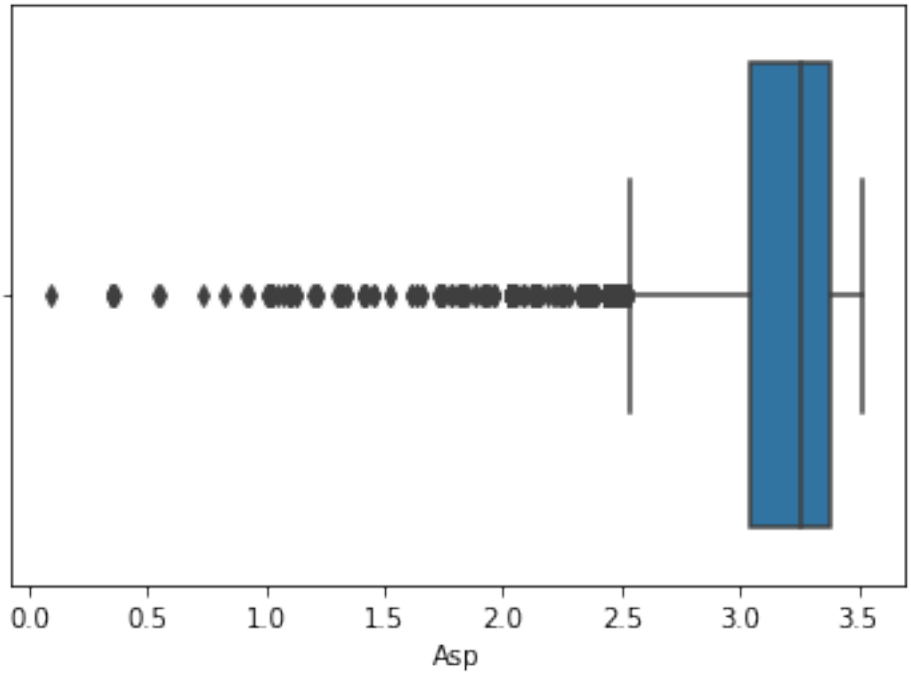
[25]:
```
sns.boxplot(x=df_sin_out['Tasa_Prod'])
```

[25]: <AxesSubplot:xlabel='Tasa_Prod'>



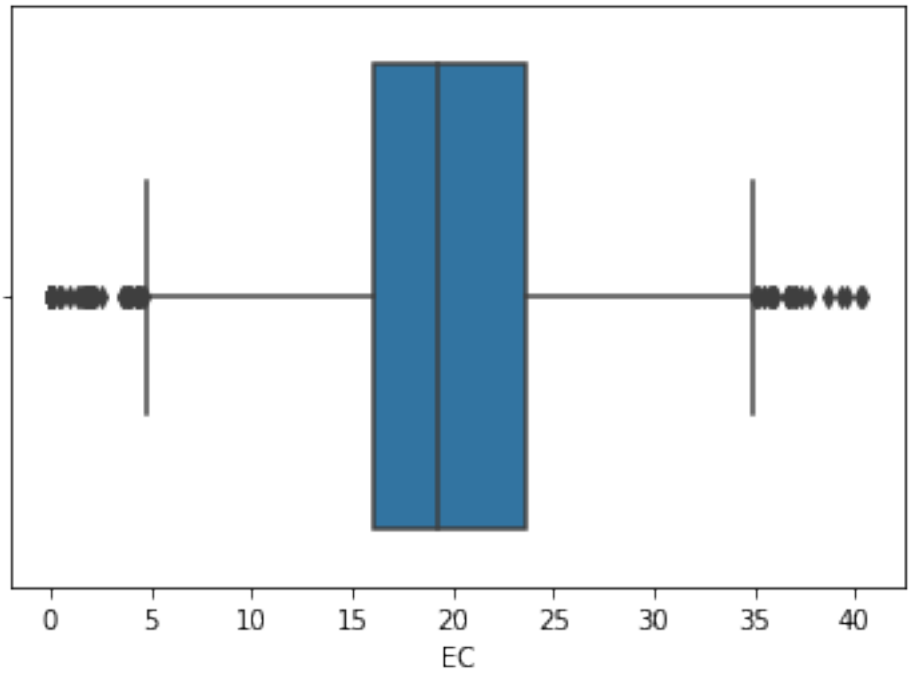
[26]:
```
sns.boxplot(x=df_sin_out['Asp'])
```

[26]: <AxesSubplot:xlabel='Asp'>



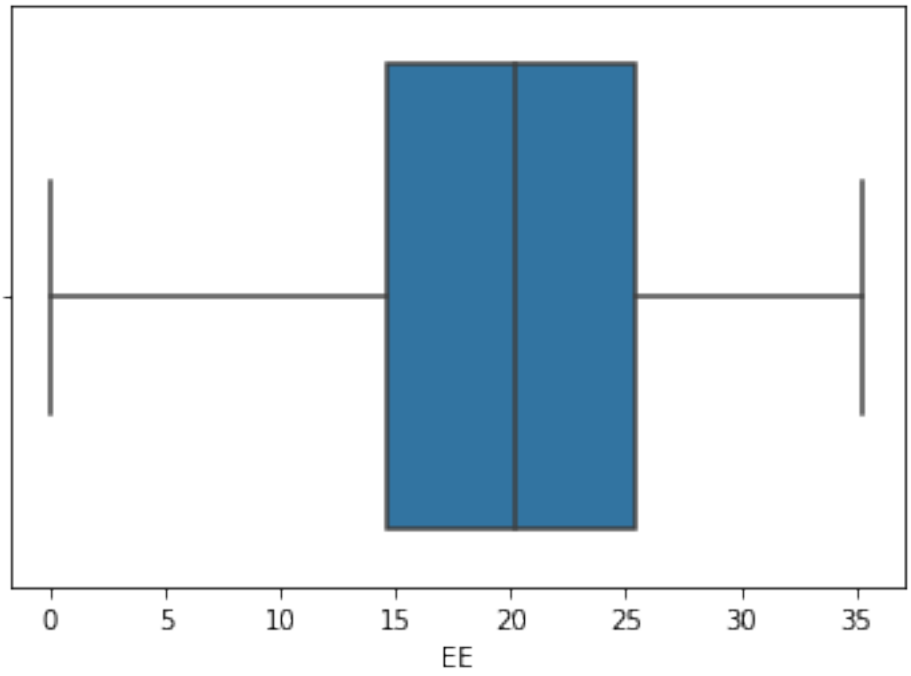
[27]:
```
sns.boxplot(x=df_sin_out['EC'])
```

[27]: <AxesSubplot:xlabel='EC'>



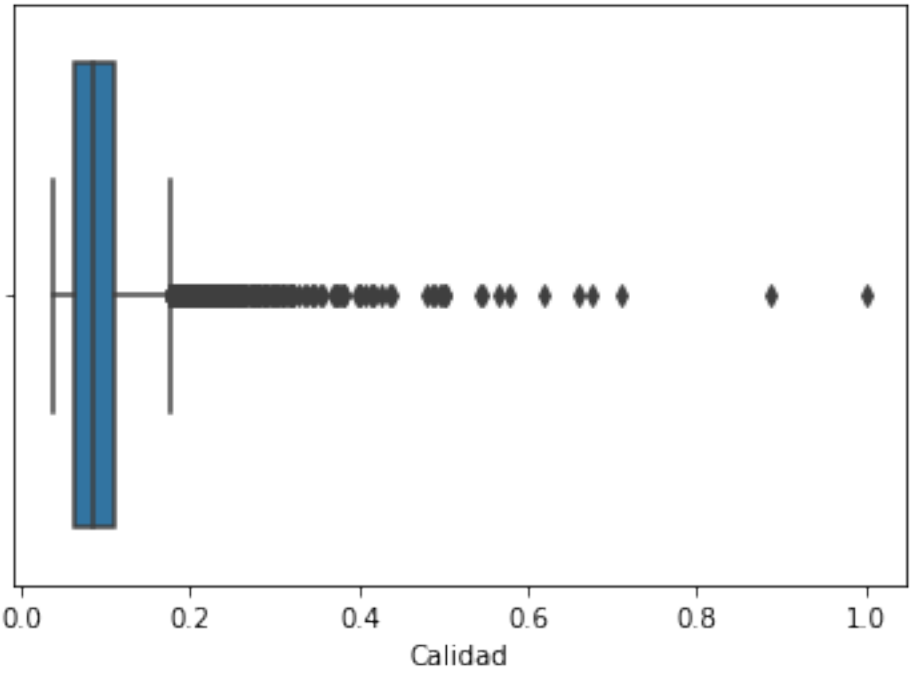
[28]:
```
sns.boxplot(x=df_sin_out['EE'])
```

[28]:
```
<AxesSubplot:xlabel='EE'>
```



[29]:
```
sns.boxplot(x=df_sin_out['Calidad'])
```

[29]:
```
<AxesSubplot:xlabel='Calidad'>
```



[30]:
```
df_sin_out.set_index("TIME", inplace = True)
df_sin_out.sort_index(inplace=True)
```

[31]:
```
df_sin_out.shape
```

[31]:
```
(8874, 6)
```

Eliminar Outliers

[32]:
```
# Eliminar Outliers con método de Z-Score
z = np.abs(stats.zscore(df_sin_out))
z
```

[32]:
```
array([[1.99859974, 0.56890195, 1.02096552, 0.66073493, 1.30920267,
        0.83716215],
       [0.93508219, 0.84012827, 0.71362186, 1.06718161, 0.358341  ,
        0.21342   ],
       [0.53175878, 1.00730135, 0.71362186, 1.1873845 , 0.97269919,
        0.17183719],
       …,
```

```
       [1.02070576, 0.19531783, 0.68651894, 0.24078675, 0.06794836,
        0.19262859],
       [1.50965275, 0.26696344, 0.74072479, 0.10479656, 1.12113383,
        0.46291686],
       [0.4461352 , 0.2191997 , 0.19866623, 0.40530379, 1.19636137,
        0.90931587]])
```

[33]:
```
np.where(z > 2.5)
```

[33]:
```
(array([  16,   34,   43, …, 8844, 8855, 8860]),
 array([3, 5, 3, …, 5, 5, 3]))
```

[34]:
```
df_sin_out = df_sin_out[(z<2.5).all(axis=1)]
```

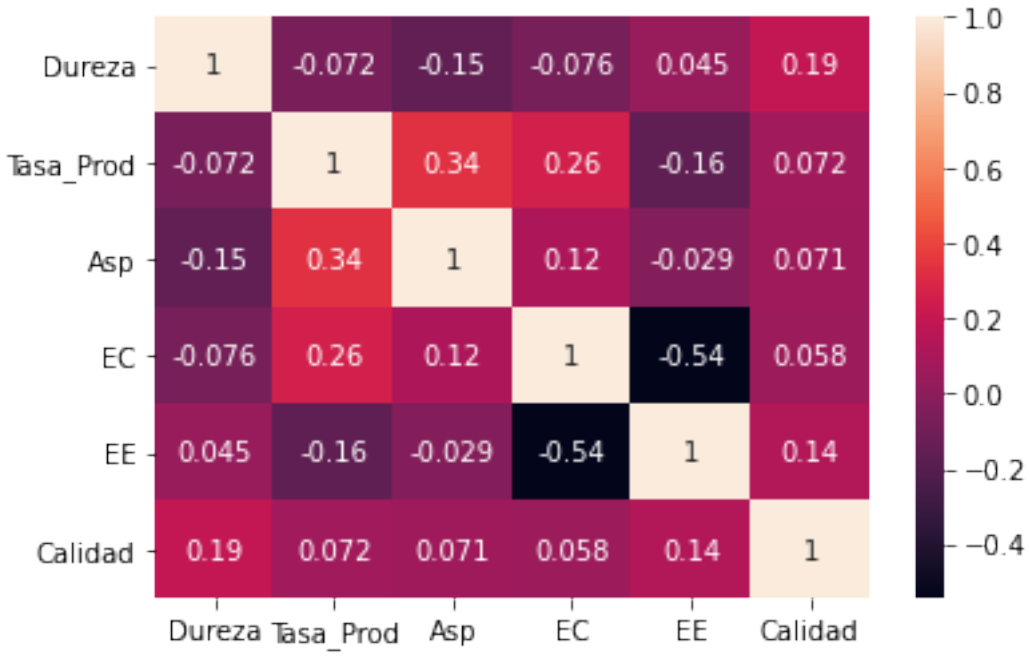
[35]:
```
df_sin_out.shape
```

[35]:
```
(7990, 6)
```

5 Visualización de Datos

Análisis de correlación de datos

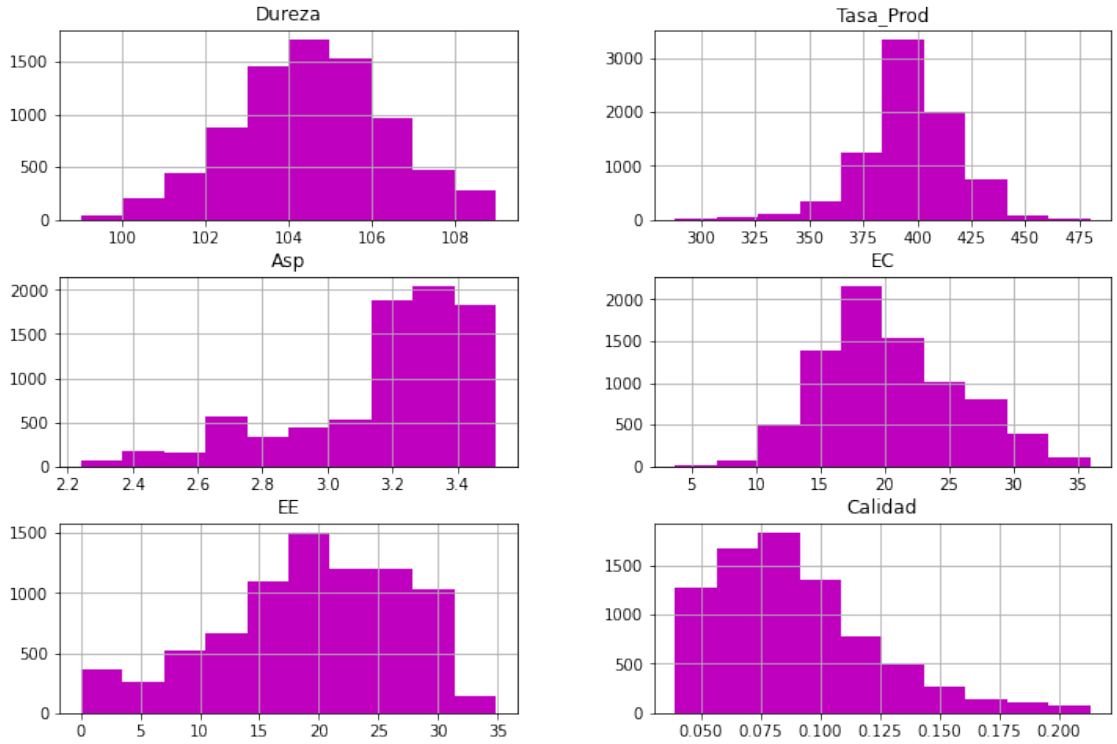
[36]:
```
# Análisis de correlación de datos

df_small = df_sin_out.iloc[:,:]

correlation_mat = df_small.corr()

sns.heatmap(correlation_mat, annot = True)

plt.show()
```



Histograma por variables

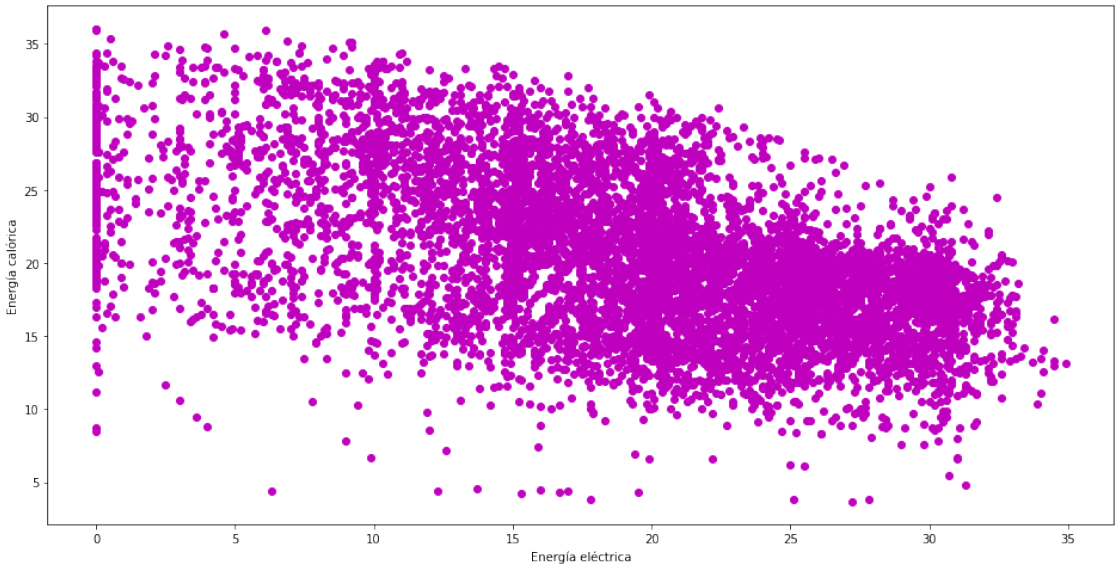
[37]:
```
# Histograma de las variables
df_sin_out.hist(color="m", figsize=(12,8))
```

[37]:
```
array([[<AxesSubplot:title={'center':'Dureza'}>,
        <AxesSubplot:title={'center':'Tasa_Prod'}>],
       [<AxesSubplot:title={'center':'Asp'}>,
        <AxesSubplot:title={'center':'EC'}>],
       [<AxesSubplot:title={'center':'EE'}>,
        <AxesSubplot:title={'center':'Calidad'}>]], dtype=object)
```

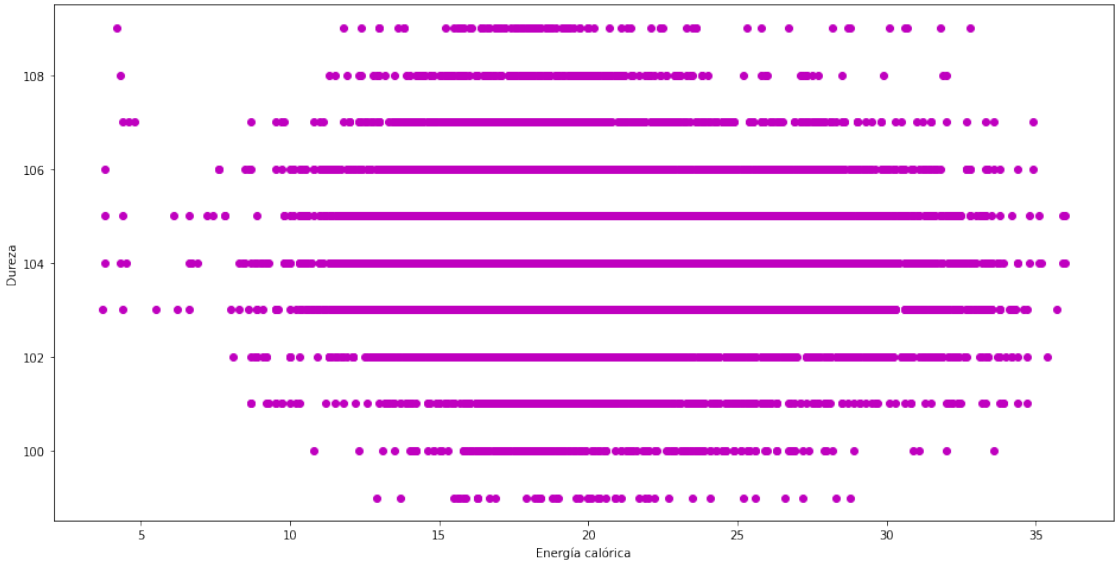


Diagramas de dispersión de las variables EE y EC con el resto de las variables para encontrar posibles patrones

[38]:
```
fig, ax = plt.subplots(figsize=(16,8))
ax.scatter(df_sin_out['EE'], df_sin_out['EC'], c = 'm')
ax.set_xlabel('Energía eléctrica')
ax.set_ylabel('Energía calórica')
plt.show()
```

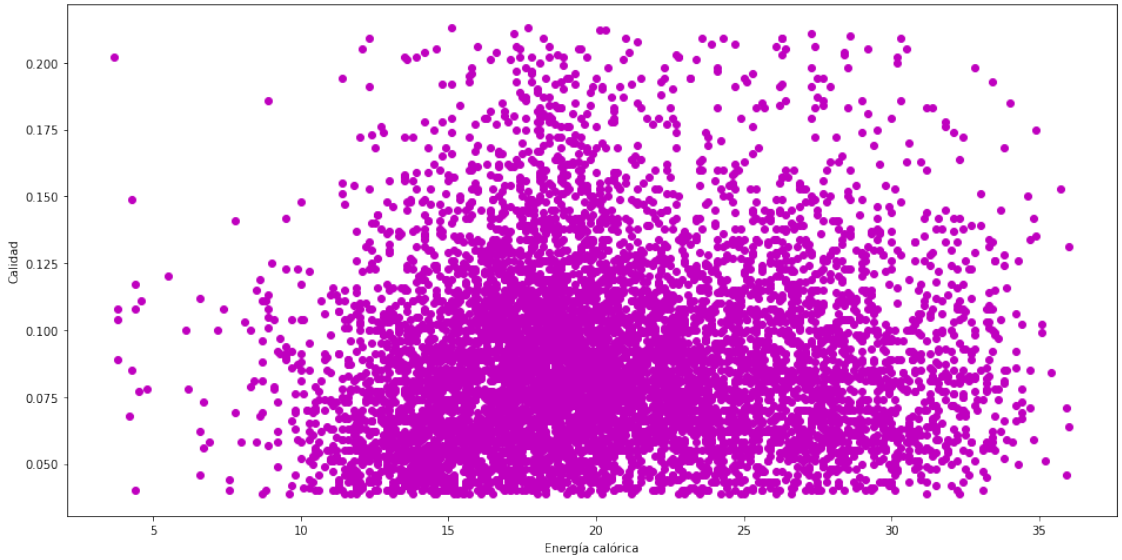


[39]:
```
# EC y Dureza
fig, ax = plt.subplots(figsize=(16,8))
ax.scatter(df_sin_out['EC'], df_sin_out['Dureza'], c = 'm')
ax.set_xlabel('Energía calórica')
ax.set_ylabel('Dureza')
plt.show()
```

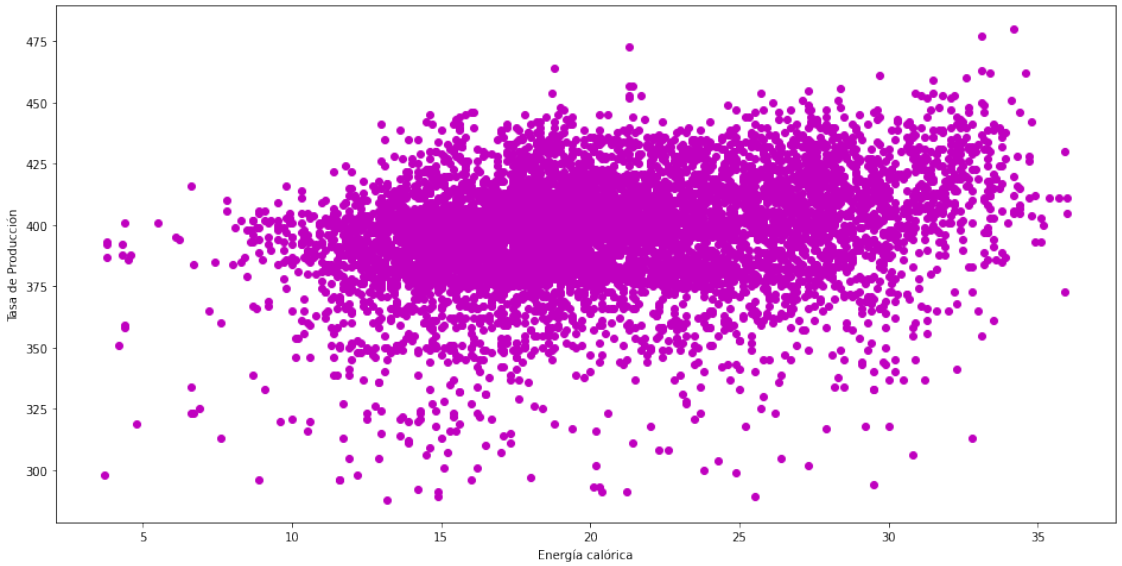


[40]:
```
fig, ax = plt.subplots(figsize=(16,8))
ax.scatter(df_sin_out['EC'], df_sin_out['Calidad'], c = 'm')
```

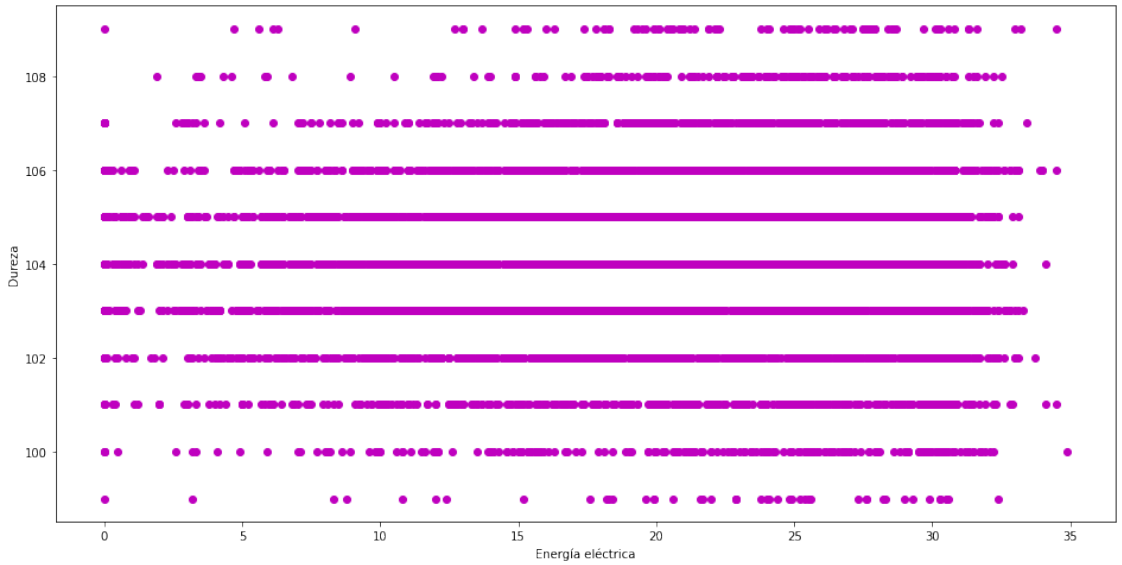
```
ax.set_xlabel('Energía calórica')
ax.set_ylabel('Calidad')
plt.show()
```



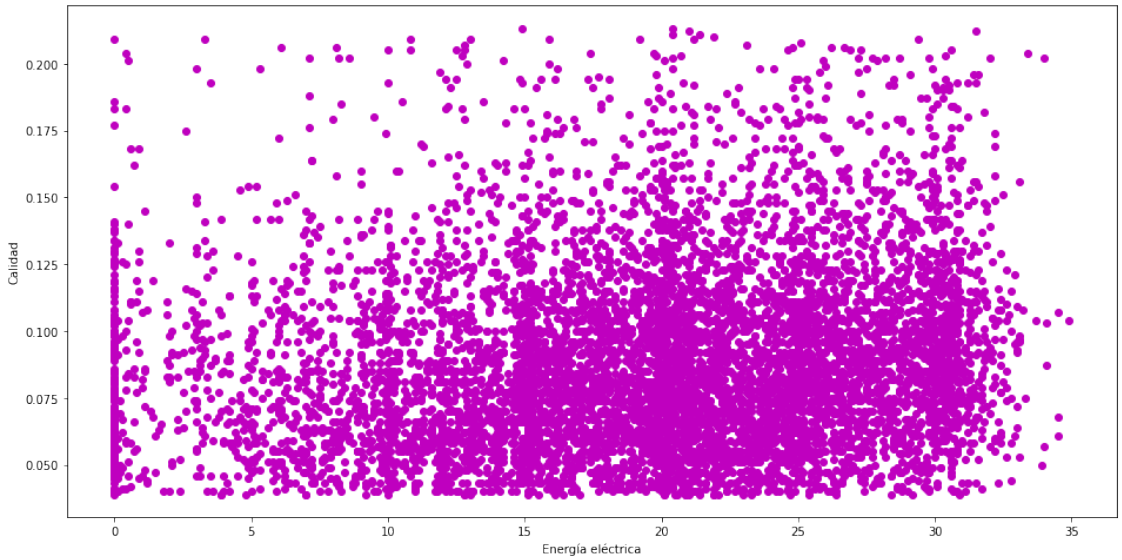
[41]:
```
fig, ax = plt.subplots(figsize=(16,8))
ax.scatter(df_sin_out['EC'], df_sin_out['Tasa_Prod'], c = 'm')
ax.set_xlabel('Energía calórica')
ax.set_ylabel('Tasa de Producción')
plt.show()
```



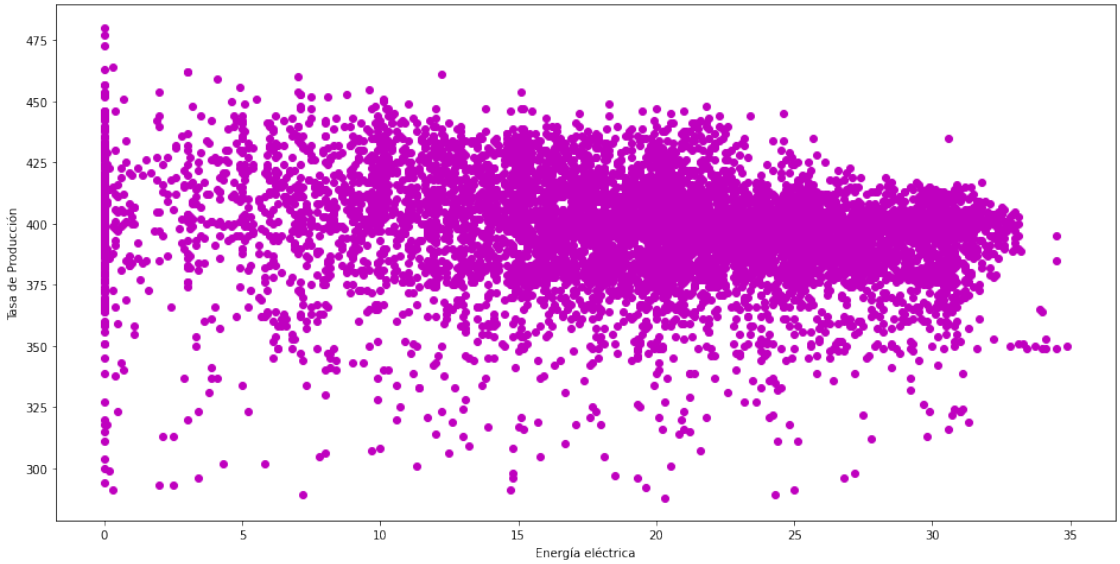
[42]:
```
fig, ax = plt.subplots(figsize=(16,8))
ax.scatter(df_sin_out['EE'], df_sin_out['Dureza'], c = 'm')
ax.set_xlabel('Energía eléctrica')
ax.set_ylabel('Dureza')
plt.show()
```



[43]:
```
fig, ax = plt.subplots(figsize=(16,8))
ax.scatter(df_sin_out['EE'], df_sin_out['Calidad'], c = 'm')
ax.set_xlabel('Energía eléctrica')
ax.set_ylabel('Calidad')
plt.show()
```



[44]:
```
fig, ax = plt.subplots(figsize=(16,8))
ax.scatter(df_sin_out['EE'], df_sin_out['Tasa_Prod'], c = 'm')
ax.set_xlabel('Energía eléctrica')
ax.set_ylabel('Tasa de Producción')
plt.show()
```



6 Conclusiones Entregable 2

Para esta etapa se concretó la limpieza de datos. Se identificaron los valores atípicos de la base de datos a través del gráfico de Boxplot y posteriormente fueron eliminados con ayuda del método de Z-score. Una vez eliminados se efectuó un análisis de correlación y se crearon distintos gráficos de dispersión para encontrar patrones.

Con base en los valores obtenidos, se puede concluir que la Dureza y la Calidad, con un índice de 0.22 podrían estar relacionadas entre sí; de la misma forma, considerando los dos tipos de energía (EE Y EC), la energía calórica se relaciona más con la tasa de producción(0.26), mientras que la eléctrica con la calidad del producto (0.16). A pesar de esto, los valores obtenidos son muy bajos pues no superan el 0.5.

En cuanto a los histogramas, podemos observar que algunos tienden a contar con una distribución normal, sobre todo la dureza con una simetría casi perfecta. La aspersión es anormal, pero debido a que no nos enfocaremos en ella para la modelación del algoritmo no es de gran importancia. Por otro lado, en los gráficas de dispersión, las variables ya mencionadas con correlación más alta son en las que los valores tienden a concentrarse más, de la misma forma, valores de correlación por debajo del cero, crean gráficas más dispersas con lineas horizontales.

7 Agregar Costos

Obtener costo Total de EE y EC

[45]:
```
df_sin_out["costoTotal"] = df_sin_out['EE'] + 0.724*df_sin_out['EC']

#df_sin_out.loc[:,'costoTotal'] = df_sin_out.sum(, axis=1)
df_sin_out
```

[45]:

	Dureza	Tasa_Prod	Asp	EC	EE	Calidad	costoTotal
TIME							
01/01/1995 0:00	100.0	368	2.78	15.1	29.7	0.053	40.6324
01/01/1996 0:00	106.0	427	3.42	26.6	16.4	0.083	35.6584
01/01/1997 0:00	103.0	434	3.42	27.4	11.5	0.085	31.3376
01/01/1998 0:00	103.0	395	2.82	24.5	16.4	0.074	34.1380
01/01/1999 0:00	106.0	388	3.32	26.2	19.1	0.111	38.0688
…	…	…	…	…	…	…	
31/12/2015 0:00	107.0	405	3.17	15.1	26.1	0.192	37.0324
31/12/2016 0:00	101.0	403	3.34	16.9	26.5	0.090	38.7356
31/12/2017 0:00	102.0	400	3.41	21.1	19.8	0.084	35.0764
31/12/2018 0:00	101.0	403	3.43	18.8	28.2	0.071	41.8112
31/12/2019 0:00	105.0	401	3.23	16.8	28.8	0.137	40.9632

[7990 rows x 7 columns]

Obtener costo PONDERADO

[46]:
```
df_sin_out["costoPonderado"] = df_sin_out['costoTotal'] /␣
↪df_sin_out['Tasa_Prod']
df_sin_out
```

[46]:

	Dureza	Tasa_Prod	Asp	…	Calidad	costoTotal
costoPonderado						
TIME				…		
01/01/1995 0:00	100.0	368	2.78	…	0.053	40.6324
0.110414						
01/01/1996 0:00	106.0	427	3.42	…	0.083	35.6584
0.083509						
01/01/1997 0:00	103.0	434	3.42	…	0.085	31.3376
0.072206						
01/01/1998 0:00	103.0	395	2.82	…	0.074	34.1380
0.086425						
01/01/1999 0:00	106.0	388	3.32	…	0.111	38.0688
0.098115						
…	…	…	…	…	…	…
…						
31/12/2015 0:00	107.0	405	3.17	…	0.192	37.0324
0.091438						

```
31/12/2016 0:00   101.0        403  3.34  …    0.090     38.7356
0.096118
31/12/2017 0:00   102.0        400  3.41  …    0.084     35.0764
0.087691
31/12/2018 0:00   101.0        403  3.43  …    0.071     41.8112
0.103750
31/12/2019 0:00   105.0        401  3.23  …    0.137     40.9632
0.102153

[7990 rows x 8 columns]
```

Obtener costo EE PONDERADA

[47]:
```
df_sin_out["EE_Ponderada"] = df_sin_out['EE'] / df_sin_out['Tasa_Prod']
df_sin_out
```

[47]:
```
                 Dureza  Tasa_Prod  …  costoPonderado  EE_Ponderada
TIME                                 …
01/01/1995 0:00   100.0        368  …        0.110414      0.080707
01/01/1996 0:00   106.0        427  …        0.083509      0.038407
01/01/1997 0:00   103.0        434  …        0.072206      0.026498
01/01/1998 0:00   103.0        395  …        0.086425      0.041519
01/01/1999 0:00   106.0        388  …        0.098115      0.049227
…                 …          …    …              …             …
31/12/2015 0:00   107.0        405  …        0.091438      0.064444
31/12/2016 0:00   101.0        403  …        0.096118      0.065757
31/12/2017 0:00   102.0        400  …        0.087691      0.049500
31/12/2018 0:00   101.0        403  …        0.103750      0.069975
31/12/2019 0:00   105.0        401  …        0.102153      0.071820

[7990 rows x 9 columns]
```

Obtener EC PONDERADA

[48]:
```
df_sin_out["EC_Ponderada"] = df_sin_out['EC'] / df_sin_out['Tasa_Prod']
df_sin_out
```

[48]:
```
                 Dureza  Tasa_Prod  …  EE_Ponderada  EC_Ponderada
TIME                                 …
01/01/1995 0:00   100.0        368  …      0.080707      0.041033
01/01/1996 0:00   106.0        427  …      0.038407      0.062295
01/01/1997 0:00   103.0        434  …      0.026498      0.063134
01/01/1998 0:00   103.0        395  …      0.041519      0.062025
01/01/1999 0:00   106.0        388  …      0.049227      0.067526
…                 …          …    …            …             …
31/12/2015 0:00   107.0        405  …      0.064444      0.037284
31/12/2016 0:00   101.0        403  …      0.065757      0.041935
31/12/2017 0:00   102.0        400  …      0.049500      0.052750
```

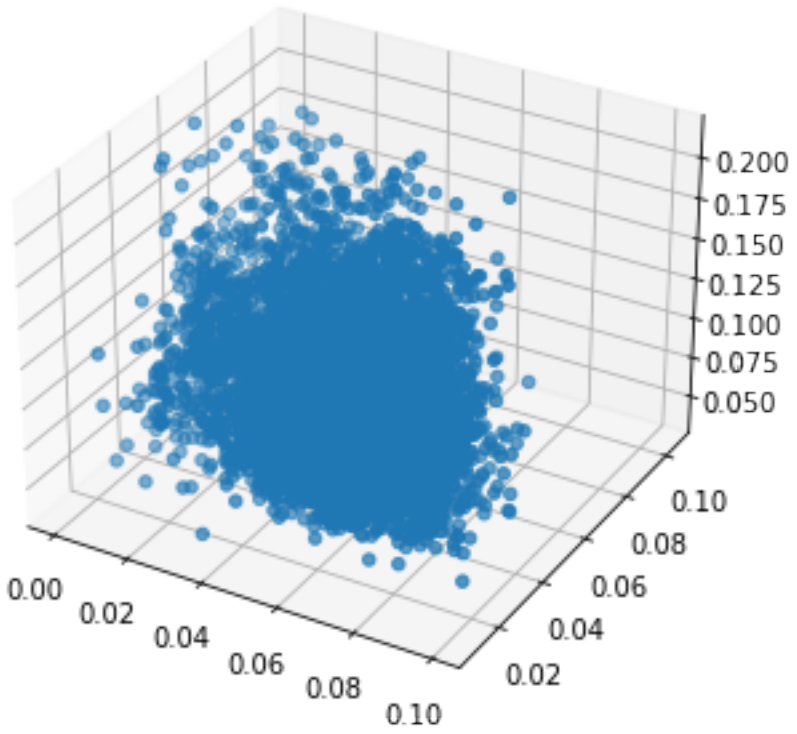
```
31/12/2018 0:00    101.0         403   …       0.069975       0.046650
31/12/2019 0:00    105.0         401   …       0.071820       0.041895

[7990 rows x 10 columns]
```

8 GRAFICAS DE DISPERSION 3D PARA CALIDAD Y DUREZA

EEP, ECP, CALIDAD

[49]:
```
from mpl_toolkits.mplot3d import Axes3D
```

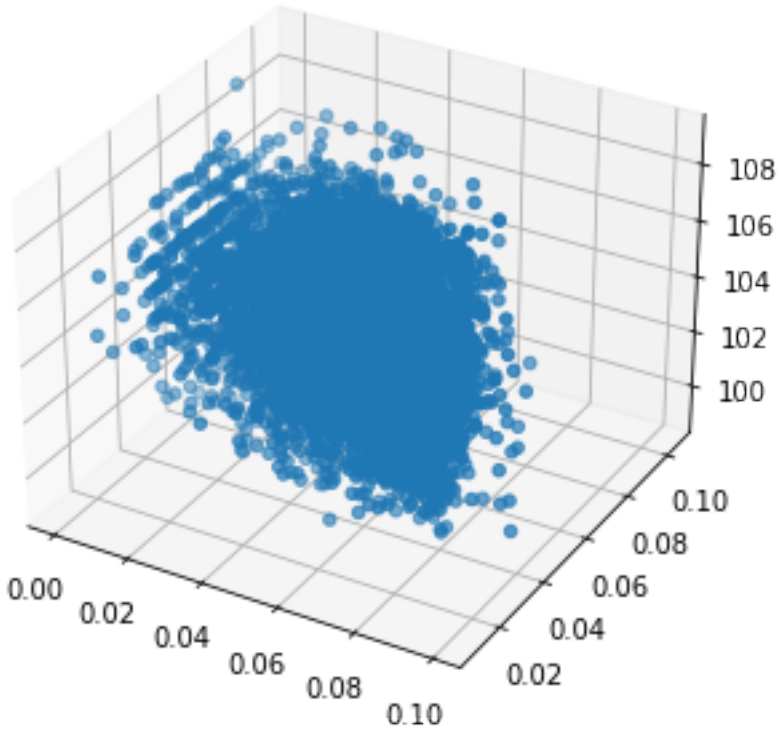
[50]:
```
def scatter3D(x,y,z):
  fig = plt.figure()
  ax = Axes3D(fig)
  ax.scatter(x,y,z)
scatter3D(df_sin_out['EE_Ponderada'], df_sin_out['EC_Ponderada'],␣
 ↪df_sin_out['Calidad'])
```



[50]:

EEP, ECP, DUREZA

[51]:
```
scatter3D(df_sin_out['EE_Ponderada'], df_sin_out['EC_Ponderada'],␣
↪df_sin_out['Dureza'])
```



9 DATOS OPTIMIZADOS

Reescala data set con los mejores datos para costo ponderado

[52]:
```
from sklearn import preprocessing

# costoPonderado reshape
# c = np.array(df_sin_out['costoPonderado']).reshape(-1,1)

#minmax_scale = preprocessing.MinMaxScaler().fit_transform(c)
#scaled_frame = pd.DataFrame(minmax_scale,columns=['costoPonderado'])
#scaled_frame.head(10)
```

[53]:
```
df_sin_out
```

[53]:
```
                  Dureza  Tasa_Prod  …  EE_Ponderada  EC_Ponderada
TIME                                   …
01/01/1995 0:00    100.0        368  …      0.080707      0.041033
01/01/1996 0:00    106.0        427  …      0.038407      0.062295
01/01/1997 0:00    103.0        434  …      0.026498      0.063134
01/01/1998 0:00    103.0        395  …      0.041519      0.062025
01/01/1999 0:00    106.0        388  …      0.049227      0.067526
…                    …          …  …           …             …
31/12/2015 0:00    107.0        405  …      0.064444      0.037284
31/12/2016 0:00    101.0        403  …      0.065757      0.041935
31/12/2017 0:00    102.0        400  …      0.049500      0.052750
31/12/2018 0:00    101.0        403  …      0.069975      0.046650
31/12/2019 0:00    105.0        401  …      0.071820      0.041895

[7990 rows x 10 columns]
```

[54]:
```
#s_f = np.array(scaled_frame["costoPonderado"])
#s_f

#df_sin_out['costoPonderado'] = scaled_frame['costoPonderado'].values
#df_sin_out
```

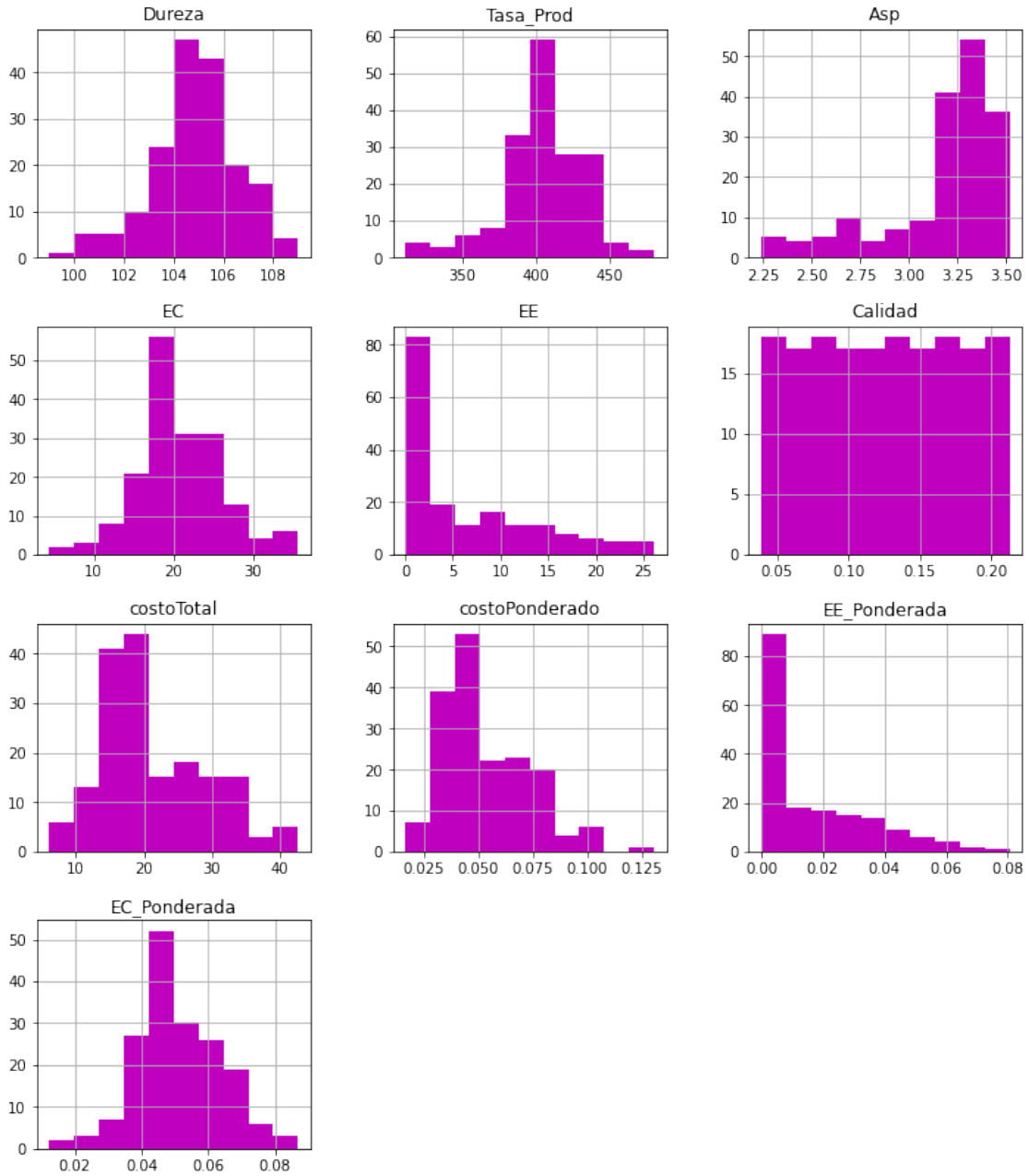
[55]:
```
df_OPT = df_sin_out.loc[df_sin_out.groupby('Calidad').costoPonderado.idxmin()]
df_OPT
```

[55]:
```
                  Dureza  Tasa_Prod  …  EE_Ponderada  EC_Ponderada
TIME                                   …
01/10/2008 0:00    106.0        413  …      0.000000      0.047700
21/09/1996 0:00    103.0        396  …      0.000000      0.028283
01/02/2003 0:00    105.0        457  …      0.000000      0.046827
12/07/2002 0:00    103.0        419  …      0.001432      0.042721
29/05/2007 0:00    102.0        374  …      0.000000      0.064171
…                    …          …  …           …             …
19/05/2020 0:00    104.0        412  …      0.000000      0.063835
20/01/2000 0:00    104.0        404  …      0.054208      0.070792
17/10/2008 0:00    105.0        440  …      0.048636      0.039091
03/06/2008 0:00    106.0        359  …      0.058496      0.056546
15/05/2009 0:00    105.0        421  …      0.035392      0.042043

[175 rows x 10 columns]
```

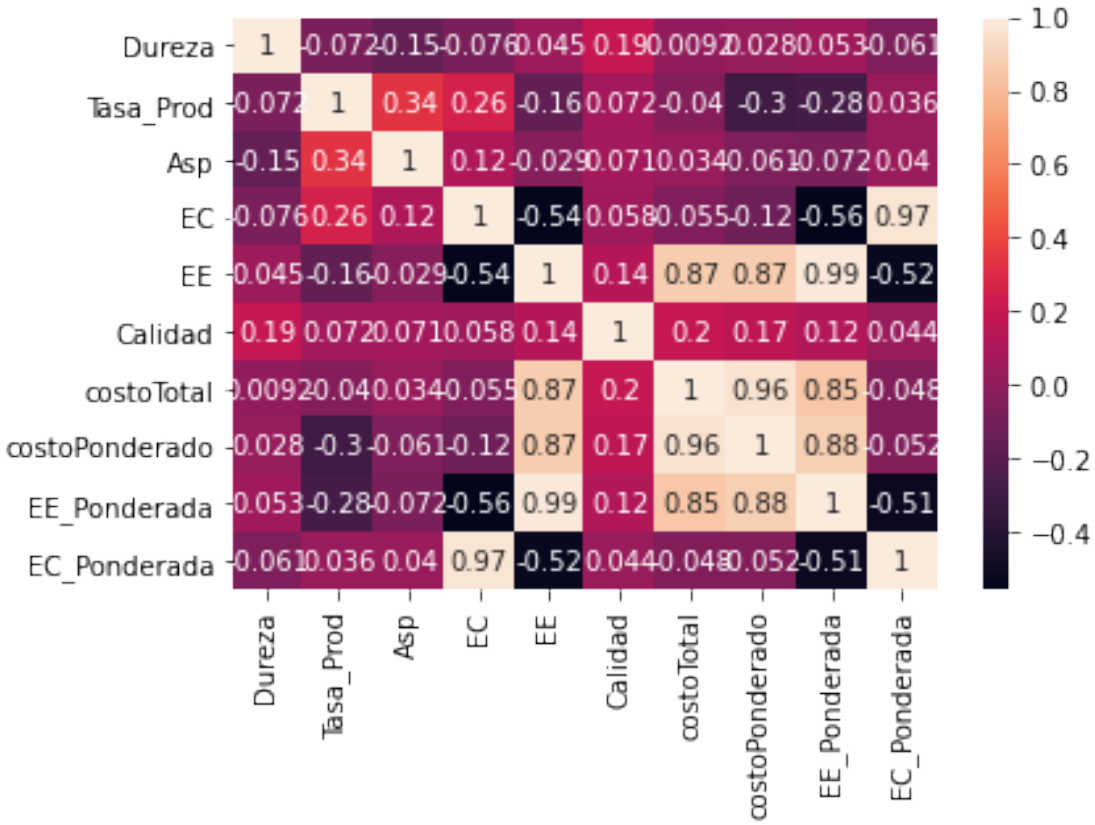
[56]:
```
df_OPT.hist(color="m", figsize=(12,14))
```

[56]:
```
array([[<AxesSubplot:title={'center':'Dureza'}>,
        <AxesSubplot:title={'center':'Tasa_Prod'}>,
```

```
 <AxesSubplot:title={'center':'Asp'}>],
[<AxesSubplot:title={'center':'EC'}>,
 <AxesSubplot:title={'center':'EE'}>,
 <AxesSubplot:title={'center':'Calidad'}>],
[<AxesSubplot:title={'center':'costoTotal'}>,
 <AxesSubplot:title={'center':'costoPonderado'}>,
 <AxesSubplot:title={'center':'EE_Ponderada'}>],
[<AxesSubplot:title={'center':'EC_Ponderada'}>, <AxesSubplot:>,
 <AxesSubplot:>]], dtype=object)
```



[57]:
```
df_small = df_sin_out.iloc[:,:]

correlation_mat = df_small.corr()

sns.heatmap(correlation_mat, annot = True)

plt.show()
```



10 Data Preparation

[58]:
```
from sklearn.multioutput import MultiOutputRegressor
from sklearn.model_selection import train_test_split
```

[59]:
```
X = np.array(df_OPT[['Calidad','Dureza','Tasa_Prod']])
X
```

```
[59]: array([[3.90e-02, 1.06e+02, 4.13e+02],
             [4.00e-02, 1.03e+02, 3.96e+02],
             [4.10e-02, 1.05e+02, 4.57e+02],
             [4.20e-02, 1.03e+02, 4.19e+02],
             [4.30e-02, 1.02e+02, 3.74e+02],
             [4.40e-02, 1.03e+02, 3.86e+02],
             [4.50e-02, 1.05e+02, 3.90e+02],
             [4.60e-02, 1.03e+02, 4.45e+02],
             [4.70e-02, 1.04e+02, 4.33e+02],
             [4.80e-02, 1.03e+02, 3.92e+02],
             [4.90e-02, 1.04e+02, 4.64e+02],
             [5.00e-02, 1.02e+02, 4.00e+02],
             [5.10e-02, 1.04e+02, 3.20e+02],
             [5.20e-02, 1.02e+02, 4.00e+02],
             [5.30e-02, 1.04e+02, 4.05e+02],
             [5.40e-02, 9.90e+01, 3.82e+02],
             [5.50e-02, 1.03e+02, 3.98e+02],
             [5.60e-02, 1.04e+02, 3.84e+02],
             [5.70e-02, 1.00e+02, 4.07e+02],
             [5.80e-02, 1.07e+02, 4.01e+02],
             [5.90e-02, 1.04e+02, 4.35e+02],
             [6.00e-02, 1.05e+02, 4.07e+02],
             [6.10e-02, 1.04e+02, 4.24e+02],
             [6.20e-02, 1.04e+02, 4.33e+02],
             [6.30e-02, 1.05e+02, 4.21e+02],
             [6.40e-02, 1.04e+02, 3.99e+02],
             [6.50e-02, 1.06e+02, 3.95e+02],
             [6.60e-02, 1.05e+02, 4.03e+02],
             [6.70e-02, 1.03e+02, 4.09e+02],
             [6.80e-02, 1.03e+02, 4.38e+02],
             [6.90e-02, 1.04e+02, 4.09e+02],
             [7.00e-02, 1.03e+02, 4.07e+02],
             [7.10e-02, 1.04e+02, 4.45e+02],
             [7.20e-02, 1.07e+02, 3.82e+02],
             [7.30e-02, 1.04e+02, 4.36e+02],
             [7.40e-02, 1.06e+02, 4.14e+02],
             [7.50e-02, 1.05e+02, 4.20e+02],
             [7.60e-02, 1.05e+02, 3.97e+02],
             [7.70e-02, 1.04e+02, 3.13e+02],
             [7.80e-02, 1.07e+02, 3.98e+02],
             [7.90e-02, 1.06e+02, 4.03e+02],
             [8.00e-02, 1.04e+02, 4.16e+02],
             [8.10e-02, 1.06e+02, 3.67e+02],
             [8.20e-02, 1.01e+02, 4.54e+02],
             [8.30e-02, 1.05e+02, 4.05e+02],
             [8.40e-02, 1.03e+02, 4.10e+02],
             [8.50e-02, 1.05e+02, 3.64e+02],
```

```
[8.60e-02, 1.03e+02, 4.80e+02],
[8.70e-02, 1.05e+02, 3.97e+02],
[8.80e-02, 1.03e+02, 4.00e+02],
[8.90e-02, 1.03e+02, 4.19e+02],
[9.00e-02, 1.06e+02, 3.93e+02],
[9.10e-02, 1.07e+02, 4.07e+02],
[9.20e-02, 1.02e+02, 4.53e+02],
[9.30e-02, 1.07e+02, 3.92e+02],
[9.40e-02, 1.04e+02, 4.06e+02],
[9.50e-02, 1.03e+02, 3.97e+02],
[9.60e-02, 1.04e+02, 4.09e+02],
[9.70e-02, 1.08e+02, 3.95e+02],
[9.80e-02, 1.02e+02, 3.15e+02],
[9.90e-02, 1.04e+02, 4.11e+02],
[1.00e-01, 1.01e+02, 4.01e+02],
[1.01e-01, 1.03e+02, 4.39e+02],
[1.02e-01, 1.03e+02, 4.31e+02],
[1.03e-01, 1.05e+02, 3.72e+02],
[1.04e-01, 1.04e+02, 4.16e+02],
[1.05e-01, 1.01e+02, 4.01e+02],
[1.06e-01, 1.04e+02, 4.22e+02],
[1.07e-01, 1.06e+02, 3.84e+02],
[1.08e-01, 1.05e+02, 3.58e+02],
[1.09e-01, 1.03e+02, 4.03e+02],
[1.10e-01, 1.05e+02, 3.81e+02],
[1.11e-01, 1.04e+02, 4.42e+02],
[1.12e-01, 1.04e+02, 4.28e+02],
[1.13e-01, 1.01e+02, 4.13e+02],
[1.14e-01, 1.04e+02, 4.24e+02],
[1.15e-01, 1.04e+02, 3.79e+02],
[1.16e-01, 1.04e+02, 4.21e+02],
[1.17e-01, 1.04e+02, 4.21e+02],
[1.18e-01, 1.05e+02, 4.40e+02],
[1.19e-01, 1.04e+02, 4.20e+02],
[1.20e-01, 1.04e+02, 3.84e+02],
[1.21e-01, 1.04e+02, 3.51e+02],
[1.22e-01, 1.03e+02, 4.21e+02],
[1.23e-01, 1.07e+02, 3.84e+02],
[1.24e-01, 1.05e+02, 4.23e+02],
[1.25e-01, 1.02e+02, 3.98e+02],
[1.26e-01, 1.06e+02, 4.19e+02],
[1.27e-01, 1.04e+02, 3.98e+02],
[1.28e-01, 1.05e+02, 3.91e+02],
[1.29e-01, 1.05e+02, 4.44e+02],
[1.30e-01, 1.06e+02, 3.87e+02],
[1.31e-01, 1.04e+02, 4.04e+02],
[1.32e-01, 1.05e+02, 3.67e+02],
```

```
[1.33e-01, 1.01e+02, 4.40e+02],
[1.34e-01, 1.04e+02, 3.92e+02],
[1.35e-01, 1.05e+02, 4.17e+02],
[1.36e-01, 1.09e+02, 3.91e+02],
[1.37e-01, 1.05e+02, 3.39e+02],
[1.38e-01, 1.05e+02, 4.40e+02],
[1.39e-01, 1.04e+02, 4.30e+02],
[1.40e-01, 1.04e+02, 3.97e+02],
[1.41e-01, 1.05e+02, 4.01e+02],
[1.42e-01, 1.05e+02, 3.98e+02],
[1.43e-01, 1.06e+02, 4.02e+02],
[1.44e-01, 1.05e+02, 3.90e+02],
[1.45e-01, 1.05e+02, 4.00e+02],
[1.46e-01, 1.06e+02, 3.99e+02],
[1.47e-01, 1.05e+02, 4.21e+02],
[1.48e-01, 1.05e+02, 3.86e+02],
[1.49e-01, 1.09e+02, 3.93e+02],
[1.50e-01, 1.03e+02, 4.62e+02],
[1.51e-01, 1.02e+02, 3.81e+02],
[1.52e-01, 1.03e+02, 4.08e+02],
[1.53e-01, 1.03e+02, 4.11e+02],
[1.54e-01, 1.04e+02, 4.10e+02],
[1.55e-01, 1.07e+02, 4.33e+02],
[1.56e-01, 1.07e+02, 3.79e+02],
[1.57e-01, 1.05e+02, 4.08e+02],
[1.58e-01, 1.05e+02, 3.84e+02],
[1.59e-01, 1.04e+02, 3.86e+02],
[1.60e-01, 1.02e+02, 4.34e+02],
[1.61e-01, 1.04e+02, 4.41e+02],
[1.62e-01, 1.03e+02, 3.40e+02],
[1.63e-01, 1.04e+02, 4.32e+02],
[1.64e-01, 1.04e+02, 4.05e+02],
[1.65e-01, 1.06e+02, 3.50e+02],
[1.66e-01, 1.06e+02, 4.21e+02],
[1.67e-01, 1.06e+02, 4.25e+02],
[1.68e-01, 1.05e+02, 4.05e+02],
[1.69e-01, 1.05e+02, 4.42e+02],
[1.70e-01, 1.04e+02, 4.07e+02],
[1.71e-01, 1.05e+02, 4.38e+02],
[1.72e-01, 1.07e+02, 4.15e+02],
[1.73e-01, 1.05e+02, 3.99e+02],
[1.74e-01, 1.04e+02, 4.40e+02],
[1.75e-01, 1.07e+02, 4.12e+02],
[1.76e-01, 1.05e+02, 4.05e+02],
[1.77e-01, 1.07e+02, 3.91e+02],
[1.78e-01, 1.05e+02, 4.01e+02],
[1.79e-01, 1.04e+02, 4.22e+02],
```

```
       [1.80e-01, 1.06e+02, 3.64e+02],
       [1.81e-01, 1.05e+02, 3.99e+02],
       [1.82e-01, 1.06e+02, 3.72e+02],
       [1.83e-01, 1.04e+02, 3.89e+02],
       [1.84e-01, 1.05e+02, 4.22e+02],
       [1.85e-01, 1.02e+02, 4.01e+02],
       [1.86e-01, 1.05e+02, 4.32e+02],
       [1.87e-01, 1.04e+02, 4.08e+02],
       [1.88e-01, 1.00e+02, 3.63e+02],
       [1.89e-01, 1.05e+02, 3.98e+02],
       [1.90e-01, 1.06e+02, 3.97e+02],
       [1.91e-01, 1.07e+02, 4.35e+02],
       [1.92e-01, 1.07e+02, 4.05e+02],
       [1.93e-01, 1.06e+02, 3.93e+02],
       [1.94e-01, 1.08e+02, 3.99e+02],
       [1.95e-01, 1.05e+02, 3.61e+02],
       [1.96e-01, 1.06e+02, 3.85e+02],
       [1.97e-01, 1.00e+02, 3.98e+02],
       [1.98e-01, 1.07e+02, 4.30e+02],
       [1.99e-01, 1.03e+02, 3.84e+02],
       [2.00e-01, 1.04e+02, 3.82e+02],
       [2.01e-01, 1.00e+02, 3.59e+02],
       [2.02e-01, 1.00e+02, 3.98e+02],
       [2.03e-01, 1.05e+02, 4.33e+02],
       [2.04e-01, 1.03e+02, 3.38e+02],
       [2.05e-01, 1.07e+02, 4.21e+02],
       [2.06e-01, 1.07e+02, 3.87e+02],
       [2.07e-01, 1.02e+02, 4.39e+02],
       [2.08e-01, 1.04e+02, 3.11e+02],
       [2.09e-01, 1.04e+02, 4.12e+02],
       [2.10e-01, 1.04e+02, 4.04e+02],
       [2.11e-01, 1.05e+02, 4.40e+02],
       [2.12e-01, 1.06e+02, 3.59e+02],
       [2.13e-01, 1.05e+02, 4.21e+02]])
```

[60]:
```
Y = np.array(df_OPT[['EE','EC','costoPonderado']])
Y
```

[60]:
```
array([[0.00000000e+00, 1.97000000e+01, 3.45346247e-02],
       [0.00000000e+00, 1.12000000e+01, 2.04767677e-02],
       [0.00000000e+00, 2.14000000e+01, 3.39028446e-02],
       [6.00000000e-01, 1.79000000e+01, 3.23618138e-02],
       [0.00000000e+00, 2.40000000e+01, 4.64598930e-02],
       [1.20000000e+00, 1.99000000e+01, 4.04341969e-02],
       [1.00000000e-01, 1.26000000e+01, 2.36471795e-02],
       [0.00000000e+00, 2.03000000e+01, 3.30274157e-02],
       [0.00000000e+00, 2.15000000e+01, 3.59491917e-02],
```

```
[0.00000000e+00, 1.85000000e+01, 3.41683673e-02],
[3.00000000e-01, 1.88000000e+01, 2.99810345e-02],
[9.80000000e+00, 1.21000000e+01, 4.64010000e-02],
[0.00000000e+00, 1.42000000e+01, 3.21275000e-02],
[7.20000000e+00, 1.84000000e+01, 5.13040000e-02],
[0.00000000e+00, 2.03000000e+01, 3.62893827e-02],
[0.00000000e+00, 2.41000000e+01, 4.56764398e-02],
[0.00000000e+00, 2.47000000e+01, 4.49316583e-02],
[9.90000000e+00, 6.70000000e+00, 3.84135417e-02],
[0.00000000e+00, 2.80000000e+01, 4.98083538e-02],
[0.00000000e+00, 1.94000000e+01, 3.50264339e-02],
[0.00000000e+00, 2.08000000e+01, 3.46188506e-02],
[5.20000000e+00, 1.80000000e+01, 4.47960688e-02],
[0.00000000e+00, 2.13000000e+01, 3.63707547e-02],
[0.00000000e+00, 1.98000000e+01, 3.31066975e-02],
[0.00000000e+00, 1.89000000e+01, 3.25026128e-02],
[0.00000000e+00, 1.92000000e+01, 3.48390977e-02],
[4.80000000e+00, 1.65000000e+01, 4.23949367e-02],
[0.00000000e+00, 1.63000000e+01, 2.92833747e-02],
[2.30000000e+00, 1.94000000e+01, 3.99647922e-02],
[0.00000000e+00, 2.09000000e+01, 3.45470320e-02],
[0.00000000e+00, 2.02000000e+01, 3.57574572e-02],
[0.00000000e+00, 1.83000000e+01, 3.25533170e-02],
[0.00000000e+00, 1.46000000e+01, 2.37537079e-02],
[0.00000000e+00, 2.16000000e+01, 4.09382199e-02],
[1.01000000e+01, 1.83000000e+01, 5.35532110e-02],
[2.00000000e-01, 1.56000000e+01, 2.77642512e-02],
[0.00000000e+00, 1.91000000e+01, 3.29247619e-02],
[0.00000000e+00, 1.85000000e+01, 3.37380353e-02],
[2.50000000e+00, 1.17000000e+01, 3.50504792e-02],
[0.00000000e+00, 2.06000000e+01, 3.74733668e-02],
[1.00000000e+00, 1.84000000e+01, 3.55374690e-02],
[1.00000000e+01, 1.83000000e+01, 5.58875000e-02],
[0.00000000e+00, 8.70000000e+00, 1.71629428e-02],
[2.00000000e+00, 1.87000000e+01, 3.42264317e-02],
[7.00000000e-01, 2.04000000e+01, 3.81965432e-02],
[4.20000000e+00, 1.83000000e+01, 4.25590244e-02],
[0.00000000e+00, 2.28000000e+01, 4.53494505e-02],
[0.00000000e+00, 3.42000000e+01, 5.15850000e-02],
[2.00000000e+00, 1.68000000e+01, 3.56755668e-02],
[9.40000000e+00, 2.12000000e+01, 6.18720000e-02],
[4.20000000e+00, 1.83000000e+01, 4.16448687e-02],
[0.00000000e+00, 2.33000000e+01, 4.29241730e-02],
[0.00000000e+00, 1.92000000e+01, 3.41542998e-02],
[0.00000000e+00, 2.17000000e+01, 3.46816777e-02],
[2.80000000e+00, 2.14000000e+01, 4.66673469e-02],
[0.00000000e+00, 2.54000000e+01, 4.52945813e-02],
```

```
[0.00000000e+00, 2.54000000e+01, 4.63214106e-02],
[4.90000000e+00, 1.55000000e+01, 3.94180929e-02],
[3.50000000e+00, 1.95000000e+01, 4.46025316e-02],
[0.00000000e+00, 1.30000000e+01, 2.98793651e-02],
[0.00000000e+00, 2.16000000e+01, 3.80496350e-02],
[4.00000000e-01, 2.33000000e+01, 4.30653367e-02],
[5.00000000e+00, 3.33000000e+01, 6.63079727e-02],
[2.60000000e+00, 2.39000000e+01, 4.61800464e-02],
[0.00000000e+00, 2.26000000e+01, 4.39849462e-02],
[5.00000000e-01, 2.63000000e+01, 4.69740385e-02],
[0.00000000e+00, 2.69000000e+01, 4.85675810e-02],
[6.20000000e+00, 1.71000000e+01, 4.40293839e-02],
[1.00000000e+00, 2.76000000e+01, 5.46416667e-02],
[6.30000000e+00, 4.40000000e+00, 2.64960894e-02],
[8.00000000e+00, 1.73000000e+01, 5.09310174e-02],
[6.00000000e-01, 2.71000000e+01, 5.30719160e-02],
[1.90000000e+00, 1.83000000e+01, 3.42742081e-02],
[5.90000000e+00, 2.59000000e+01, 5.75971963e-02],
[4.20000000e+00, 1.83000000e+01, 4.22498789e-02],
[3.00000000e+00, 2.38000000e+01, 4.77150943e-02],
[0.00000000e+00, 8.50000000e+00, 1.62374670e-02],
[0.00000000e+00, 2.36000000e+01, 4.05852732e-02],
[5.00000000e+00, 1.99000000e+01, 4.60988124e-02],
[0.00000000e+00, 2.57000000e+01, 4.22881818e-02],
[1.40000000e+00, 2.44000000e+01, 4.53942857e-02],
[3.40000000e+00, 2.35000000e+01, 5.31614583e-02],
[0.00000000e+00, 2.35000000e+01, 4.84729345e-02],
[1.48000000e+01, 1.74000000e+01, 6.50774347e-02],
[3.60000000e+00, 9.50000000e+00, 2.72864583e-02],
[0.00000000e+00, 2.51000000e+01, 4.29607565e-02],
[4.00000000e-01, 2.61000000e+01, 4.84834171e-02],
[0.00000000e+00, 1.70000000e+01, 2.93747017e-02],
[0.00000000e+00, 2.46000000e+01, 4.47497487e-02],
[6.90000000e+00, 1.99000000e+01, 5.44951407e-02],
[0.00000000e+00, 2.53000000e+01, 4.12549550e-02],
[9.00000000e-01, 2.00000000e+01, 3.97416021e-02],
[3.00000000e+00, 2.92000000e+01, 5.97544554e-02],
[1.01000000e+01, 2.23000000e+01, 7.15128065e-02],
[2.00000000e+00, 3.08000000e+01, 5.52254545e-02],
[0.00000000e+00, 2.61000000e+01, 4.82051020e-02],
[9.00000000e+00, 3.11000000e+01, 7.55788969e-02],
[9.10000000e+00, 1.86000000e+01, 5.77145780e-02],
[0.00000000e+00, 2.64000000e+01, 5.63823009e-02],
[0.00000000e+00, 2.49000000e+01, 4.09718182e-02],
[1.53000000e+01, 1.76000000e+01, 6.52148837e-02],
[5.00000000e-01, 2.21000000e+01, 4.15627204e-02],
[0.00000000e+00, 2.26000000e+01, 4.08039900e-02],
```

```
[5.70000000e+00, 1.93000000e+01, 4.94301508e-02],
[7.20000000e+00, 1.58000000e+01, 4.63661692e-02],
[1.64000000e+01, 1.48000000e+01, 6.95261538e-02],
[1.10000000e+00, 2.57000000e+01, 4.92670000e-02],
[1.25000000e+01, 1.98000000e+01, 6.72561404e-02],
[1.45000000e+01, 2.27000000e+01, 7.34793349e-02],
[3.00000000e+00, 2.00000000e+01, 4.52849741e-02],
[6.30000000e+00, 1.65000000e+01, 4.64274809e-02],
[3.00000000e+00, 3.46000000e+01, 6.07151515e-02],
[1.38000000e+01, 1.14000000e+01, 5.78834646e-02],
[1.75000000e+01, 1.90000000e+01, 7.66078431e-02],
[4.60000000e+00, 3.57000000e+01, 7.40798054e-02],
[5.20000000e+00, 2.07000000e+01, 4.92360976e-02],
[1.55000000e+01, 2.16000000e+01, 7.19131640e-02],
[1.80000000e+01, 1.38000000e+01, 7.38554090e-02],
[1.35000000e+01, 2.65000000e+01, 8.01127451e-02],
[8.10000000e+00, 1.50000000e+01, 4.93750000e-02],
[1.73000000e+01, 1.39000000e+01, 7.08901554e-02],
[1.34000000e+01, 2.35000000e+01, 7.00783410e-02],
[2.05000000e+01, 1.72000000e+01, 7.47229025e-02],
[7.00000000e-01, 1.63000000e+01, 3.67682353e-02],
[1.16000000e+01, 3.10000000e+01, 7.88055556e-02],
[7.20000000e+00, 2.15000000e+01, 5.62123457e-02],
[1.22000000e+01, 1.48000000e+01, 6.54720000e-02],
[1.26000000e+01, 2.53000000e+01, 7.34375297e-02],
[1.51000000e+01, 1.80000000e+01, 6.61929412e-02],
[9.00000000e-01, 2.00000000e+01, 3.79753086e-02],
[1.99000000e+01, 1.82000000e+01, 7.48343891e-02],
[1.88000000e+01, 1.84000000e+01, 7.89228501e-02],
[1.75000000e+01, 1.91000000e+01, 7.15260274e-02],
[1.53000000e+01, 1.20000000e+01, 5.78024096e-02],
[2.59000000e+01, 1.93000000e+01, 9.99328321e-02],
[9.90000000e+00, 3.21000000e+01, 7.53190909e-02],
[2.60000000e+00, 3.49000000e+01, 6.76398058e-02],
[1.69000000e+01, 1.27000000e+01, 6.44316049e-02],
[0.00000000e+00, 2.27000000e+01, 4.20327366e-02],
[1.43000000e+01, 1.70000000e+01, 6.63541147e-02],
[1.62000000e+01, 1.63000000e+01, 6.63535545e-02],
[9.50000000e+00, 1.69000000e+01, 5.97131868e-02],
[1.88000000e+01, 1.97000000e+01, 8.28641604e-02],
[2.11000000e+01, 1.80000000e+01, 9.17526882e-02],
[0.00000000e+00, 2.57000000e+01, 4.78323907e-02],
[1.22000000e+01, 2.77000000e+01, 7.64331754e-02],
[8.30000000e+00, 3.40000000e+01, 8.20847880e-02],
[0.00000000e+00, 2.64000000e+01, 4.42444444e-02],
[1.99000000e+01, 2.05000000e+01, 8.51519608e-02],
[7.10000000e+00, 2.22000000e+01, 6.38369146e-02],
```

```
       [2.24000000e+01, 1.74000000e+01, 8.79336683e-02],
       [2.54000000e+01, 2.26000000e+01, 1.05194962e-01],
       [1.23000000e+01, 2.73000000e+01, 7.37131034e-02],
       [2.61000000e+01, 1.51000000e+01, 9.14380247e-02],
       [1.00000000e+01, 1.88000000e+01, 6.00793893e-02],
       [1.21000000e+01, 1.90000000e+01, 6.48020050e-02],
       [1.77000000e+01, 1.57000000e+01, 8.05174515e-02],
       [1.98000000e+01, 2.53000000e+01, 9.90057143e-02],
       [1.19000000e+01, 2.41000000e+01, 7.37396985e-02],
       [3.00000000e+00, 2.85000000e+01, 5.49627907e-02],
       [2.60000000e+01, 1.89000000e+01, 1.03342708e-01],
       [1.29000000e+01, 1.81000000e+01, 6.80743455e-02],
       [5.00000000e-01, 1.71000000e+01, 3.58785515e-02],
       [8.20000000e+00, 2.28000000e+01, 6.20783920e-02],
       [1.27000000e+01, 2.84000000e+01, 7.68166282e-02],
       [4.00000000e-01, 1.66000000e+01, 3.67408284e-02],
       [1.00000000e+01, 1.46000000e+01, 4.88608076e-02],
       [6.10000000e+00, 2.61000000e+01, 6.45901809e-02],
       [1.28000000e+01, 2.47000000e+01, 6.98924829e-02],
       [2.51000000e+01, 2.14000000e+01, 1.30526045e-01],
       [0.00000000e+00, 2.63000000e+01, 4.62165049e-02],
       [2.19000000e+01, 2.86000000e+01, 1.05461386e-01],
       [2.14000000e+01, 1.72000000e+01, 7.69381818e-02],
       [2.10000000e+01, 2.03000000e+01, 9.94350975e-02],
       [1.49000000e+01, 1.77000000e+01, 6.58308789e-02]])
```

[61]:
```
'''from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
X = scaler.fit_transform(X)
scaler = MinMaxScaler(feature_range=(0, 1))
Y = scaler.fit_transform(X)'''
```

[61]:
```
'from sklearn.preprocessing import MinMaxScaler\nscaler =
MinMaxScaler(feature_range=(0, 1))\nX = scaler.fit_transform(X)\nscaler =
MinMaxScaler(feature_range=(0, 1))\nY = scaler.fit_transform(X)'
```

[62]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,␣
 ↪random_state=0)
```

[63]:
```
print("X_train : ",X_train.shape)

print("X_test : ",X_test.shape)

print("y_train : ",Y_train.shape)

print("y_test : ",Y_test.shape)
```

```
X_train :  (122, 3)
X_test :  (53, 3)
y_train :  (122, 3)
y_test :  (53, 3)
```

11 Support Vector Machine

[64]:
```
from sklearn.svm import SVC
from sklearn import metrics, svm
svclassifier = svm.SVR()
msvc = MultiOutputRegressor(svclassifier)
msvc.fit(X_train, Y_train)
```

[64]:
```
MultiOutputRegressor(estimator=SVR())
```

[65]:
```
y_pred = msvc.predict(X_test)
```

[66]:
```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(Y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test,␣
 ↪y_pred)))
```

```
Mean Absolute Error: 3.385401354641868
Mean Squared Error: 32.68232401399914
Root Mean Squared Error: 5.716845634963318
```

[67]:
```
msvc.score(X_test, Y_test)
```

[67]:
```
-0.4357610390318198
```

[68]:
```
print(msvc.get_params())
```

```
{'estimator__C': 1.0, 'estimator__cache_size': 200, 'estimator__coef0': 0.0,
'estimator__degree': 3, 'estimator__epsilon': 0.1, 'estimator__gamma': 'scale',
'estimator__kernel': 'rbf', 'estimator__max_iter': -1, 'estimator__shrinking':
True, 'estimator__tol': 0.001, 'estimator__verbose': False, 'estimator': SVR(),
'n_jobs': None}
```

[69]:
```
from sklearn.model_selection import GridSearchCV
#Defining hyperparameters
#boostrap: Method of selecting samples for training each tree
# max_depth: Maximum number of levels in tree
# max_features: Number of features to consider at every split
# n_estimators: Number of trees
```

```
param_grid = { 'estimator__C': [0.1, 10,100], 'estimator__gamma': [1, 0.1, 0.
 ↪01, 0.001, 0.0001],'estimator__kernel':['rbf'] }
```

[70]:
```
g_search = GridSearchCV(estimator = msvc, param_grid = param_grid,
 cv = 3, n_jobs = 1, verbose = 0, return_train_score=True)
```

[71]:
```
import time
start_time = time.time()
g_search.fit(X_train, Y_train);
print(g_search.best_score_)
print(g_search.best_params_)
print("Execution time: " + str((time.time() - start_time)) + ' ms')
```

```
-0.43665171035731415
{'estimator__C': 10, 'estimator__gamma': 0.0001, 'estimator__kernel': 'rbf'}
Execution time: 0.3755512237548828 ms
```

[72]:
```
from sklearn.svm import SVC
from sklearn import metrics, svm
svclassifier = svm.SVR()
msvc = MultiOutputRegressor(svclassifier)
msvc.set_params(estimator__C=10, estimator__gamma= 0.
 ↪0001,estimator__kernel='rbf')
msvc.fit(X_train, Y_train)
y_pred = msvc.predict(X_test)
```

[73]:
```
y_pred = msvc.predict(X_test)
```

[74]:
```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(Y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test,␣
 ↪y_pred)))
```

```
Mean Absolute Error: 3.4136899017081146
Mean Squared Error: 32.22536221189876
Root Mean Squared Error: 5.676738695051831
```

12 Arbol de Decision

[75]:
```
!pip install -U scikit-learn
```

```
Requirement already up-to-date: scikit-learn in /usr/local/lib/python3.7/dist-
packages (0.24.1)
Requirement already satisfied, skipping upgrade: joblib>=0.11 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn) (1.0.1)
```

```
Requirement already satisfied, skipping upgrade: scipy>=0.19.1 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn) (1.4.1)
Requirement already satisfied, skipping upgrade: numpy>=1.13.3 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn) (1.19.5)
Requirement already satisfied, skipping upgrade: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn) (2.1.0)
```

[76]:
```
!pip install -U matplotlib
```

```
Requirement already up-to-date: matplotlib in /usr/local/lib/python3.7/dist-
packages (3.3.4)
Requirement already satisfied, skipping upgrade: numpy>=1.15 in
/usr/local/lib/python3.7/dist-packages (from matplotlib) (1.19.5)
Requirement already satisfied, skipping upgrade:
pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in /usr/local/lib/python3.7/dist-
packages (from matplotlib) (2.4.7)
Requirement already satisfied, skipping upgrade: pillow>=6.2.0 in
/usr/local/lib/python3.7/dist-packages (from matplotlib) (7.0.0)
Requirement already satisfied, skipping upgrade: kiwisolver>=1.0.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib) (2.8.1)
Requirement already satisfied, skipping upgrade: cycler>=0.10 in
/usr/local/lib/python3.7/dist-packages (from matplotlib) (0.10.0)
Requirement already satisfied, skipping upgrade: six>=1.5 in
/usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.1->matplotlib)
(1.15.0)
```

[77]:
```
!pip install -U graphviz
```

```
Requirement already up-to-date: graphviz in /usr/local/lib/python3.7/dist-
packages (0.16)
```

[78]:
```
!pip install -U pydotplus
```

```
Requirement already up-to-date: pydotplus in /usr/local/lib/python3.7/dist-
packages (2.0.2)
Requirement already satisfied, skipping upgrade: pyparsing>=2.0.1 in
/usr/local/lib/python3.7/dist-packages (from pydotplus) (2.4.7)
```

[79]:
```
!pip install delayed
```

```
Requirement already satisfied: delayed in /usr/local/lib/python3.7/dist-packages
(0.11.0b1)
Requirement already satisfied: hiredis in /usr/local/lib/python3.7/dist-packages
(from delayed) (1.1.0)
Requirement already satisfied: redis in /usr/local/lib/python3.7/dist-packages
(from delayed) (3.5.3)
```

[80]:
```
# Load libraries
import pandas as pd
from sklearn.tree import DecisionTreeRegressor # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split␣
 ↪function
from sklearn import metrics #Import scikit-learn metrics module for accuracy␣
 ↪calculation
```

[81]:
```
# Create Decision Tree classifer object
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,␣
 ↪random_state=0)

clf = DecisionTreeRegressor()
mclf = MultiOutputRegressor(clf)
mclf.fit(X_train, Y_train)
clf.fit(X_train, Y_train)
#Predict the response for test dataset
y_pred = mclf.predict(X_test)
```

[82]:
```
#X_train
```

[83]:
```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(Y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test,␣
 ↪y_pred)))
```

```
Mean Absolute Error: 3.8376259853747783
Mean Squared Error: 40.44722118085564
Root Mean Squared Error: 6.359812983166694
```

[84]:
```
mclf.score(X_test, Y_test)
```

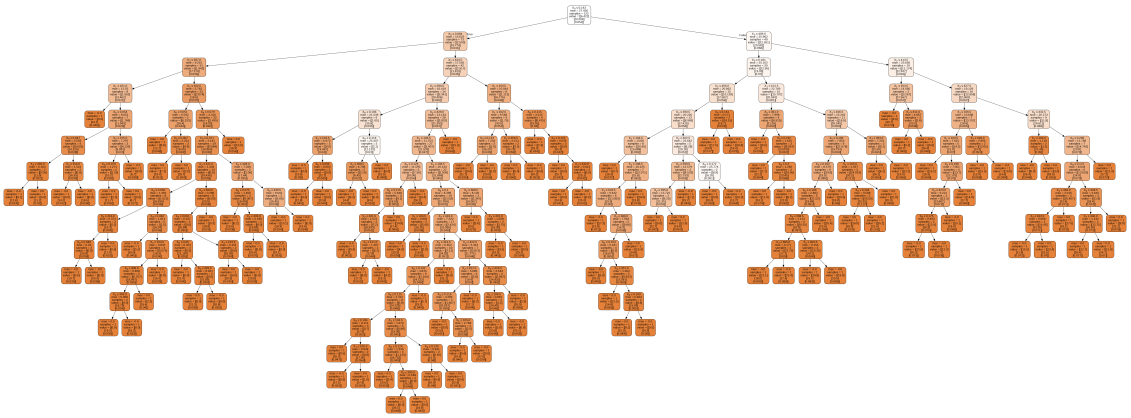
[84]:
```
-0.4578908369262371
```

[85]:
```
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus
```

[86]:
```
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = None␣
 ↪,class_names=['EE','EC','costoPonderado'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('datos.png')
```

```
Image(graph.create_png())
```

[86]:



[87]:
```
#Optimizando
# Create Decision Tree classifer object
clf = DecisionTreeRegressor(max_depth=3)

# Train Decision Tree Classifer
clf = clf.fit(X_train,Y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

[88]:
```
print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(Y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test,␣
 ↪y_pred)))
```

```
Mean Absolute Error: 2.7208896831176186
Mean Squared Error: 18.934333564105007
Root Mean Squared Error: 4.35135996719474
```

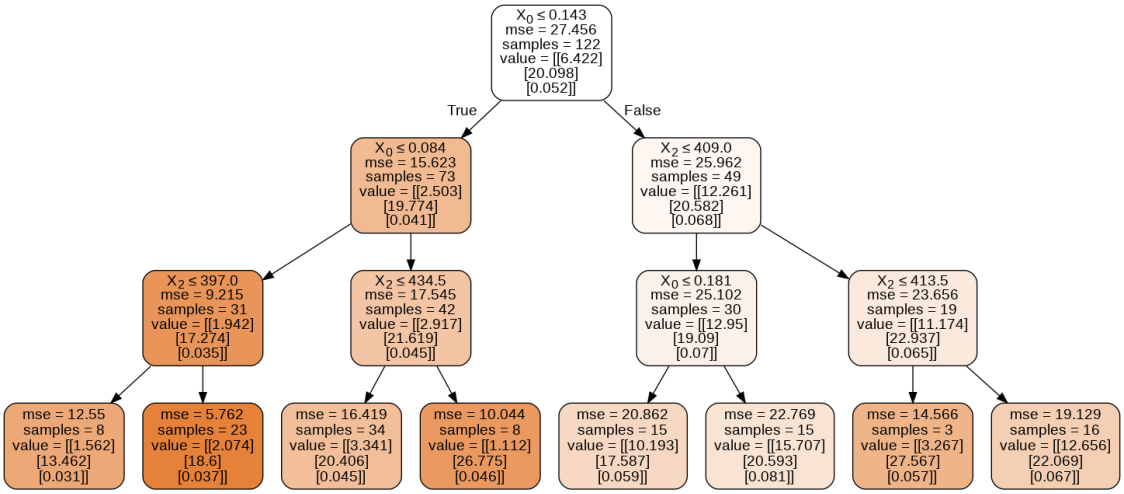
[89]:
```
clf.score(X_test, Y_test)
```

[89]:
```
0.29285424650269576
```

[90]:
```
#feature_names = ['Calidad','Dureza','Tasa_Prod']
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names = None␣
 ↪,class_names=['EE','EC','costoPonderado'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('datos2.png')
```

```
Image(graph.create_png())
```

[90]:



[91]:
```
print(clf.get_params())
```

```
{'estimator__ccp_alpha': 0.0, 'estimator__criterion': 'mse',
'estimator__max_depth': None, 'estimator__max_features': None,
'estimator__max_leaf_nodes': None, 'estimator__min_impurity_decrease': 0.0,
'estimator__min_impurity_split': None, 'estimator__min_samples_leaf': 1,
'estimator__min_samples_split': 2, 'estimator__min_weight_fraction_leaf': 0.0,
'estimator__random_state': None, 'estimator__splitter': 'best', 'estimator':
DecisionTreeRegressor(), 'n_jobs': None}
```

[92]:
```
from sklearn.model_selection import GridSearchCV
#Defining hyperparameters
#boostrap: Method of selecting samples for training each tree
# max_depth: Maximum number of levels in tree
# max_features: Number of features to consider at every split
# n_estimators: Number of trees

param_grid = { 'max_depth': [1,2,3], 'max_features': [1, 2,␣
↪3],'min_samples_split':[2, 3 ,4, 5, 6, 7], 'min_samples_leaf': [2, 3 ,4, 5,␣
↪6, 7] }
```

[93]:
```
g_search = GridSearchCV(estimator = clf, param_grid = param_grid,
 cv = 3, n_jobs = 1, verbose = 0, return_train_score=True)
```

[94]:
```
import time
start_time = time.time()
g_search.fit(X_train, Y_train);
print(g_search.best_score_)
```

```
print(g_search.best_params_)
print("Execution time: " + str((time.time() - start_time)) + ' ms')
```

```
0.2004668079386902
{'max_depth': 3, 'max_features': 2, 'min_samples_leaf': 7, 'min_samples_split':
6}
Execution time: 1.7232670783996582 ms
```

[95]:
```
# Load libraries
import pandas as pd
from sklearn.tree import DecisionTreeRegressor # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split␣
↪function
from sklearn import metrics #Import scikit-learn metrics module for accuracy␣
↪calculation
```

[96]:
```
# Create Decision Tree classifer object
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,␣
↪random_state=0)

clf = DecisionTreeRegressor()
mclf = MultiOutputRegressor(clf)
clf.set_params(max_depth = 2, min_samples_leaf = 3, min_samples_split = 7)
mclf.fit(X_train, Y_train)
clf.fit(X_train, Y_train)
#Predict the response for test dataset
y_pred = mclf.predict(X_test)
```

[97]:
```
print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(Y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test,␣
↪y_pred)))
```

```
Mean Absolute Error: 2.888058998318194
Mean Squared Error: 20.61899073422654
Root Mean Squared Error: 4.540813884561504
```

13 XGBoost

[98]:
```
import xgboost as xgb
from xgboost import XGBClassifier
from xgboost import XGBRegressor
from sklearn.model_selection import cross_val_score
```

[99]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,␣
↪random_state=0)
X_train = pd.DataFrame(X_train)
Y_train = pd.DataFrame(Y_train)
X_test = pd.DataFrame(X_test)
xg = xgb.XGBRegressor()
mxg = MultiOutputRegressor(xg)
mxg.fit(X_train, Y_train)
y_pred = mxg.predict(X_test)
```

```
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.

/usr/local/lib/python3.7/dist-packages/xgboost/core.py:613: UserWarning: Use
subset (sliced data) of np.ndarray is not recommended because it will generate
extra copies and increase memory consumption
  warnings.warn("Use subset (sliced data) of np.ndarray is not recommended " +
```

[100]:
```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(Y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test,␣
↪y_pred)))
```

```
Mean Absolute Error: 2.920816997883931
Mean Squared Error: 22.88587018875721
Root Mean Squared Error: 4.783917870193552
```

[101]:
```
mxg.score(X_test, Y_test)
```

[101]:
```
0.18501613868526487
```

[102]:
```
print(mxg.get_params())
```

```
{'estimator__base_score': 0.5, 'estimator__booster': 'gbtree',
'estimator__colsample_bylevel': 1, 'estimator__colsample_bynode': 1,
'estimator__colsample_bytree': 1, 'estimator__gamma': 0,
'estimator__importance_type': 'gain', 'estimator__learning_rate': 0.1,
'estimator__max_delta_step': 0, 'estimator__max_depth': 3,
'estimator__min_child_weight': 1, 'estimator__missing': None,
'estimator__n_estimators': 100, 'estimator__n_jobs': 1, 'estimator__nthread':
None, 'estimator__objective': 'reg:linear', 'estimator__random_state': 0,
'estimator__reg_alpha': 0, 'estimator__reg_lambda': 1,
'estimator__scale_pos_weight': 1, 'estimator__seed': None, 'estimator__silent':
```

```
None, 'estimator__subsample': 1, 'estimator__verbosity': 1, 'estimator':
XGBRegressor(), 'n_jobs': None}
```

[103]:
```
from sklearn.model_selection import GridSearchCV
#Defining hyperparameters
#boostrap: Method of selecting samples for training each tree
# max_depth: Maximum number of levels in tree
# max_features: Number of features to consider at every split
# n_estimators: Number of trees

param_grid = {'estimator__base_score': [1,2,3], 'estimator__booster':␣
 ↪['gbtree'], 'estimator__colsample_bylevel': [1]}
# =  {'min_child_weight': [1,2 ,3],'booster':[1,2,3], 'gamma':[0,1,2,3]}
```

[104]:
```
g_search = GridSearchCV(estimator = mxg, param_grid = param_grid,
 cv = 3, n_jobs = 1, verbose = 0, return_train_score=True)
```

[105]:
```
#Y_test
```

[106]:
```
#X_train
```

[107]:
```
#y_pred
```

[108]:
```
import time
start_time = time.time()
g_search.fit(X_train, Y_train)
print(g_search.best_score_)
print(g_search.best_params_)
print("Execution time: " + str((time.time() - start_time)) + ' ms')
```

```
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
```

```
/usr/local/lib/python3.7/dist-packages/xgboost/core.py:613: UserWarning: Use
```

```
subset (sliced data) of np.ndarray is not recommended because it will generate
extra copies and increase memory consumption
  warnings.warn("Use subset (sliced data) of np.ndarray is not recommended " +
/usr/local/lib/python3.7/dist-packages/xgboost/core.py:613: UserWarning: Use
subset (sliced data) of np.ndarray is not recommended because it will generate
extra copies and increase memory consumption
  warnings.warn("Use subset (sliced data) of np.ndarray is not recommended " +
/usr/local/lib/python3.7/dist-packages/xgboost/core.py:613: UserWarning: Use
subset (sliced data) of np.ndarray is not recommended because it will generate
extra copies and increase memory consumption
  warnings.warn("Use subset (sliced data) of np.ndarray is not recommended " +
/usr/local/lib/python3.7/dist-packages/xgboost/core.py:613: UserWarning: Use
subset (sliced data) of np.ndarray is not recommended because it will generate
extra copies and increase memory consumption
  warnings.warn("Use subset (sliced data) of np.ndarray is not recommended " +
```

```
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
```

```
/usr/local/lib/python3.7/dist-packages/xgboost/core.py:613: UserWarning: Use
subset (sliced data) of np.ndarray is not recommended because it will generate
extra copies and increase memory consumption
  warnings.warn("Use subset (sliced data) of np.ndarray is not recommended " +
/usr/local/lib/python3.7/dist-packages/xgboost/core.py:613: UserWarning: Use
subset (sliced data) of np.ndarray is not recommended because it will generate
extra copies and increase memory consumption
  warnings.warn("Use subset (sliced data) of np.ndarray is not recommended " +
/usr/local/lib/python3.7/dist-packages/xgboost/core.py:613: UserWarning: Use
subset (sliced data) of np.ndarray is not recommended because it will generate
```

```
extra copies and increase memory consumption
  warnings.warn("Use subset (sliced data) of np.ndarray is not recommended " +
/usr/local/lib/python3.7/dist-packages/xgboost/core.py:613: UserWarning: Use
subset (sliced data) of np.ndarray is not recommended because it will generate
extra copies and increase memory consumption
  warnings.warn("Use subset (sliced data) of np.ndarray is not recommended " +
```

```
[17:19:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
-0.05873280189374975
{'estimator__base_score': 3, 'estimator__booster': 'gbtree',
'estimator__colsample_bylevel': 1}
Execution time: 0.5478286743164062 ms
```

```
/usr/local/lib/python3.7/dist-packages/xgboost/core.py:613: UserWarning: Use
subset (sliced data) of np.ndarray is not recommended because it will generate
extra copies and increase memory consumption
  warnings.warn("Use subset (sliced data) of np.ndarray is not recommended " +
/usr/local/lib/python3.7/dist-packages/xgboost/core.py:613: UserWarning: Use
subset (sliced data) of np.ndarray is not recommended because it will generate
extra copies and increase memory consumption
  warnings.warn("Use subset (sliced data) of np.ndarray is not recommended " +
```

[109]:

```
import xgboost as xgb
from xgboost import XGBClassifier
from xgboost import XGBRegressor
from sklearn.model_selection import cross_val_score
```

[110]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,␣
↪random_state=0)
X_train = pd.DataFrame(X_train)
Y_train = pd.DataFrame(Y_train)
X_test = pd.DataFrame(X_test)
xg = xgb.XGBRegressor()
mxg = MultiOutputRegressor(xg)
mxg.set_params(estimator__base_score = 3,estimator__booster =␣
↪'gbtree',estimator__colsample_bylevel = 1)
mxg.fit(X_train, Y_train)
y_pred = mxg.predict(X_test)
```

```
[17:19:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[17:19:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.

/usr/local/lib/python3.7/dist-packages/xgboost/core.py:613: UserWarning: Use
subset (sliced data) of np.ndarray is not recommended because it will generate
extra copies and increase memory consumption
  warnings.warn("Use subset (sliced data) of np.ndarray is not recommended " +
```

[111]:
```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(Y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test,␣
↪y_pred)))
```

```
Mean Absolute Error: 2.866560698286932
Mean Squared Error: 22.24814720103167
Root Mean Squared Error: 4.716794165641709
```

[112]:
```
'''xgb_model = XGBRegressor(
        objective = 'reg:squarederror',
        estimator__base_score: 3,
        estimator__booster: 'gbtree',
        estimator__colsample_bylevel: 1)

%time xgb_model.fit(X_train, Y_train, early_stopping_rounds=5,␣
↪eval_set=[(X_val, Y_val)], verbose=False)

Y_pred_xgb = xgb_model.predict(X_val)

mae_xgb = mean_absolute_error(Y_val, Y_pred_xgb)

print("MAE: ", mae_xgb)'''
```

[112]:
```
'xgb_model = XGBRegressor(\n        objective = \'reg:squarederror\',\n
estimator__base_score: 3,\n        estimator__booster: \'gbtree\', \n
estimator__colsample_bylevel: 1)\n\n%time xgb_model.fit(X_train, Y_train,
early_stopping_rounds=5, eval_set=[(X_val, Y_val)], verbose=False)\n\nY_pred_xgb
= xgb_model.predict(X_val)\n\nmae_xgb = mean_absolute_error(Y_val,
Y_pred_xgb)\n\nprint("MAE: ", mae_xgb)'
```

[113]:
```
def prediccionFinal():

  dureza_ = float(input("Dureza: "))
  tasa_prod_ = float(input("Tasa de Producción: "))
  calidad_ = float(input("Calidad :"))
  #x_train = np.array([dureza_, tasa_prod_, calidad_])
  x_train = pd.DataFrame(np.array([[dureza_, tasa_prod_, calidad_]]))
  prediccion_final = mclf.predict(x_train)

  energia_ec = prediccion_final[(0,0)]
  energia_ee = prediccion_final[(0,1)]
  costo_ponderado = prediccion_final[(0,2)]

  print('---------')
  print('Energía calórica: ', energia_ec)
  print('Energía eléctrica: ', energia_ee)
  print('Costo Ponderado: ', costo_ponderado)


# Energía Calórica, Energía Eléctrica, Costo Ponderado
```

[114]:
```
prediccionFinal()
```

```
Dureza: 98
Tasa de Producción: 330
Calidad :0.10
---------
Energía calórica:  19.099999999999998
Energía eléctrica:  19.296153846153846
Costo Ponderado:  0.07380842988621712
```

TABLA DE COMPARACIONES

[115]:
```
from tabulate import tabulate
table = [["SVM Score","-0.4357610390318198", "SVM MAE", "3.413689", "SVM MSE",␣
 ↪"32.225362", "SVM RSME", "5.676738"], ["DECISION TREE Score", "0.
 ↪29285424650269576", "Decision Tree MAE", "2.88","Decision Tree MSE", "20.
 ↪61", "Decision Tree RMSE", "4.54" ], ["XGBoost Score", "0.
 ↪18501613868526487", "XGBoost MAE", "2.866560 ", "XGBoost MSE", "22.248147 ",␣
 ↪"XGBoost RMSE", "4.716794"]]
print(tabulate(table))
```

```
------------------  ---------  -----------------  -------  -----------------
-------  ------------------  -------
SVM Score            -0.435761  SVM MAE            3.41369  SVM MSE
32.2254  SVM RSME            5.67674
DECISION TREE Score   0.292854  Decision Tree MAE  2.88     Decision Tree MSE
20.61    Decision Tree RMSE  4.54
XGBoost Score         0.185016  XGBoost MAE        2.86656  XGBoost MSE
22.2481  XGBoost RMSE        4.71679
------------------  ---------  -----------------  -------  -----------------
-------  ------------------  -------
```