**TRABAJO PRACTICO**

**PARADIGMAS DE PROGRAMACION**

**Lorenzo Martinez**

**Informacion de los Datos**

Para empezar hablemos de las distintas columnas que tiene el dataframe y que representa cada uno para los diamantes...

Antes que nada . La fuente del dataframe es

<https://www.kaggle.com/datasets/shivam2503/diamonds>

***Descripción:***

Conjunto de datos que contiene los precios y otros atributos de casi 54.000 diamantes. Las variables son las siguientes:

***Formato :***

Un dataframe con 53940 filas y 10 variables:

***Price (precio):***

precio en dólares estadounidenses (\$326-\$18.823)

***Carat (quilate):***

peso del diamante (0,2-5,01)

***Cut (talla):***

calidad de la talla (Fair/regular, Good/buena, Very Good/muy buena, Premium/superior, Ideal)

***Color:***

color del diamante, de D (mejor) a J (peor)

***Clarity (claridad)***

medida de la claridad del diamante (I1 (peor), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (mejor))

***X:***

longitud en mm (0-10,74)

***Y:***

anchura en mm (0-58,9)

***Z:***

profundidad en mm (0-31,8)

***Depth (profundidad)***

porcentaje de profundidad total = z / media(x, y) = 2 \* z / (x + y) (43-79)

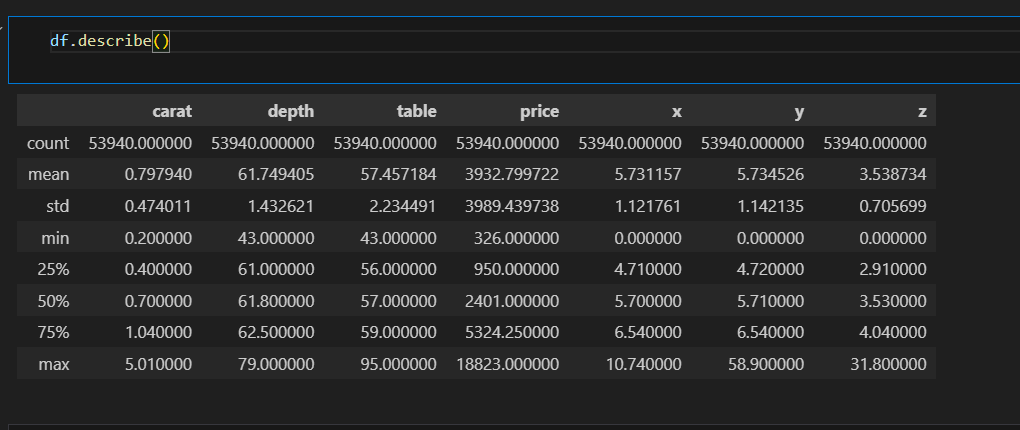
***Table (tabla)***

anchura de la parte superior del diamante en relación con el punto más ancho (43-95)

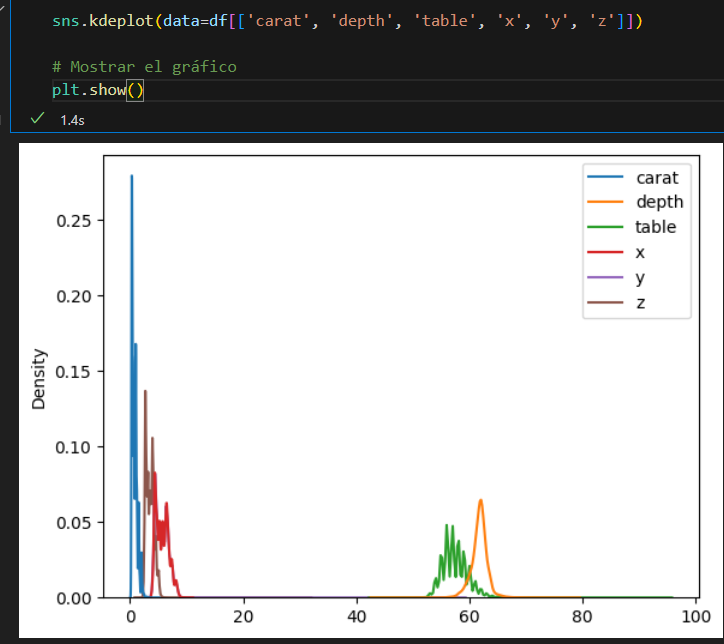
Visualicemos el dataframe



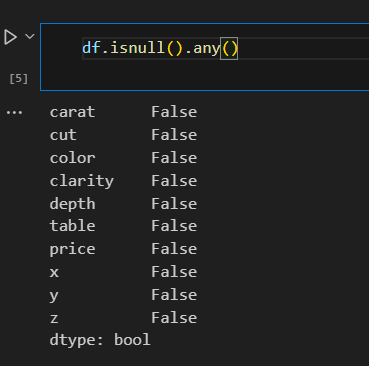
**detalles estadísticos del conjunto de datos:**



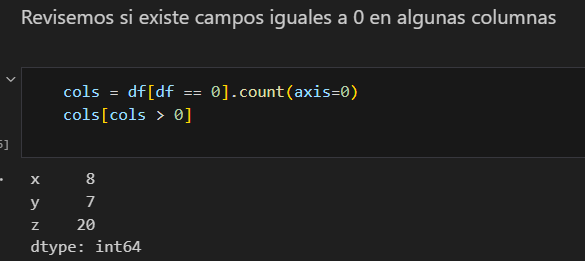
Antes de continuar observemos el grafico de densidad



Ahora se continua revisando que tipos de datos se guardan en cada columna y de paso  comprobamos que todas las filas contienen datos no nulos



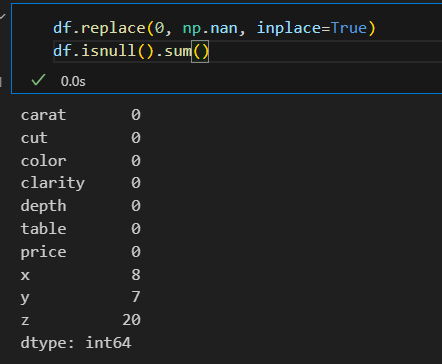
No tenemos NANS /nulos en el dataset ,pero podriamos tener 0 que podrian ser como NAns camuflados



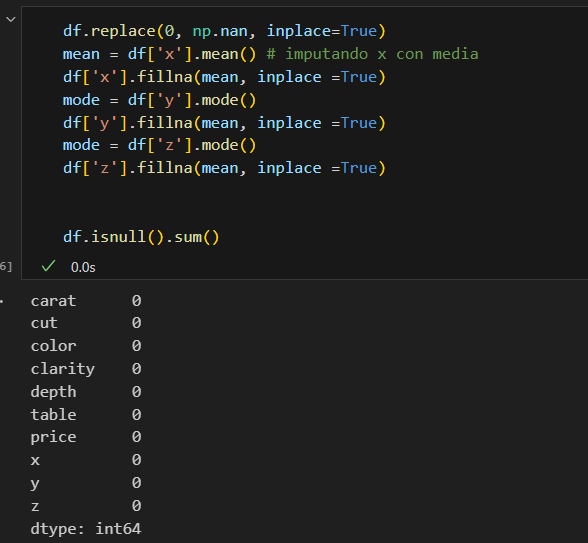
X y z tienen valores igual a 0. Un diamante es un objeto tridimensional , de ninguna forma alguna una cordenada puede valer 0 en un diamante . Algo hay que hacer efectivamente.  Pero antes observemos aquellos diamantes que tienen 0 en sus coordenadas.



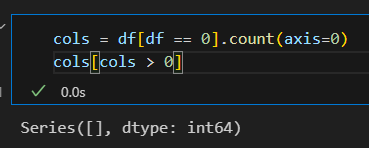
Se observa incluso que existen casos en donde las 3 coordenadas son 0!!. Por suerte solo son 20 casos con anomalias  en sus coordeanadas aun asi que se va aplicar alguna estrategia para contrarrestar esto



Como solo X y z tiene 0 ,podemos indicar directamente que se remplacen todos los 0 por un Nan



Imputamos todos los nan por el valor medio de cada columna y finalmente comprobamos que ya no estan los nan. De paso comprobamos que ya no existen columnas con x y z con 0 en sus filas



A continuacion analicemos los Outliers que tiene el dataframe

Texto

Descripción generada automáticamente

Gráfico

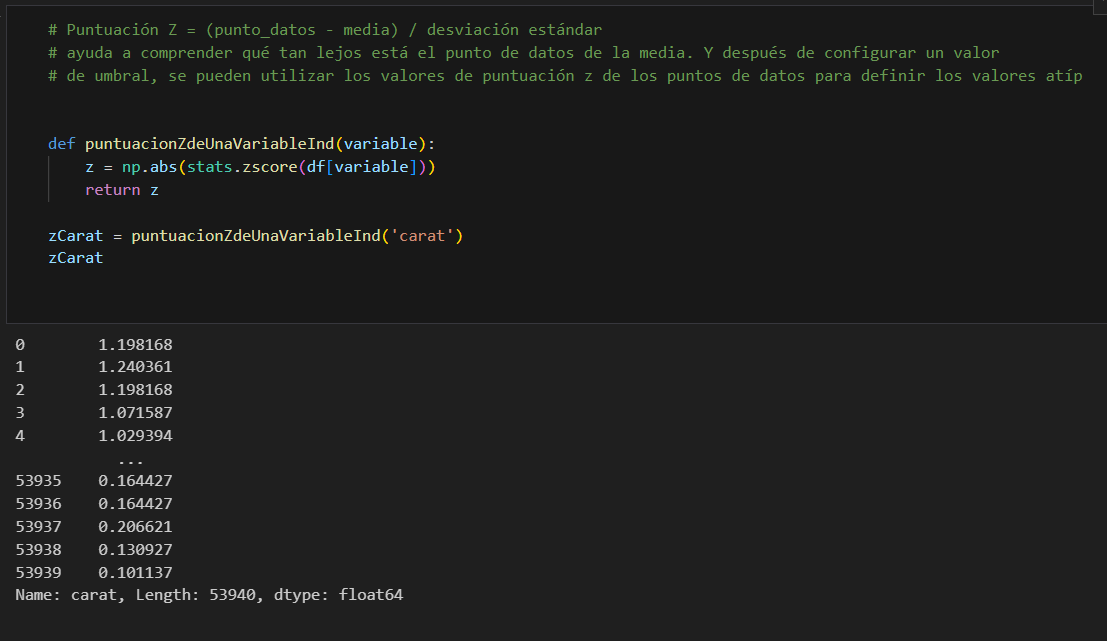
Descripción generada automáticamente

los rombos rojos representan los valores atípicos (outliers) en los datos. A simple vista pareciera que por cada columna contamos con varios valores atipicos no? Vamos a seguir con en analisis uno por uno para determinar que hacer con ellos

Otro grafico mas.  En este caso solo analizamos para la columna carat



Veamos la Puntuación Z

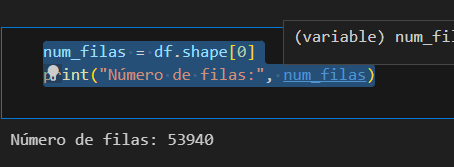


Contemos el numero de outliers en cada columna.



Para hacer el siguiente analisis tenemos que saber de antemano lo siguiente ;

1ero : ¿cuantos registros cuenta nuestros dataframe? (Recordarlo no viene mal)

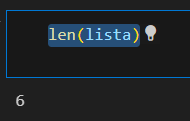


Lo almacenamos en una variable porque la vamos a usar mas adelante

***2do :*** ahora veamos la cantidad de variables independientes que consideramos para hacer el analisis de datos outliders

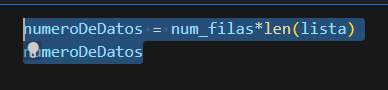


Aclaracion : la columna price no  se tiene en cuenta porque es la que vamos a usar para predecir. Por eso no esta en la lista. El resto son columnas/variables categoricas que directamente no se tienen en cuenta para el analisis



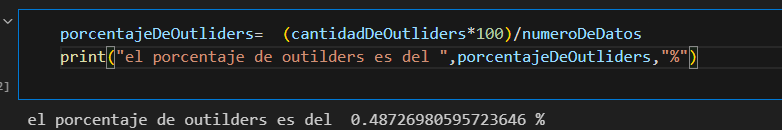
***3ero :***

¿Cuantos datos hay en el df en total teniendo en cuenta solo la lista ['carat', 'depth', 'table', 'x', 'y', 'z']? Osea la sumatoria de todos los datos que tiene el dataframe en las columnas  'carat', 'depth', 'table', 'x', 'y', 'z' . Para saberlo hacemos una cuenta sencilla (numero de filas \* cantidad de filas que consideramos en este caso es 6)



***4to:***

Lo que sigue ahora es determinar el porcentaje sobre el total de datos que son outliers

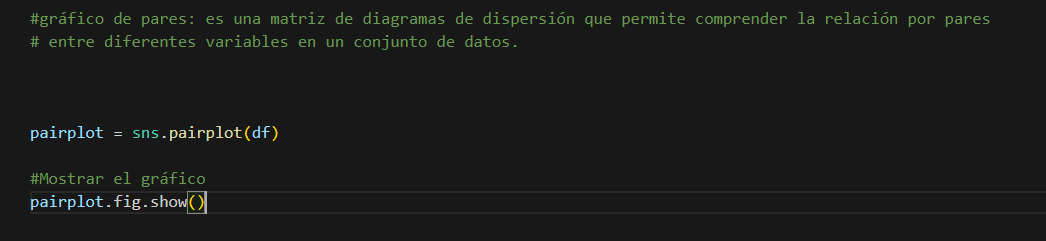


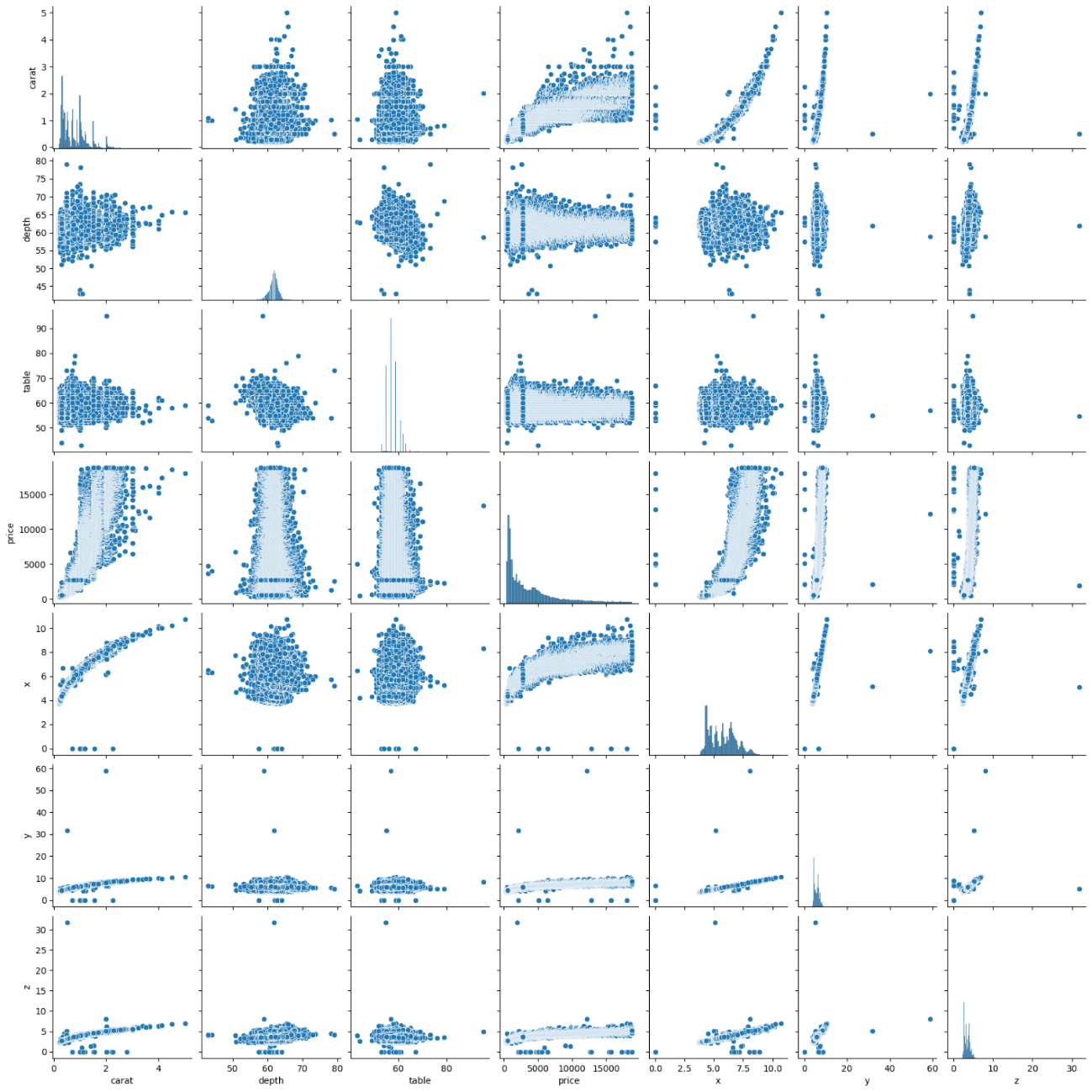
El porcentaje de valores atípicos (outliers) en las columnas ['carat', 'depth', 'table', 'x', 'y', 'z'] es solo del 0.49%. Esto significa que es posible que estos valores atípicos no tengan un impacto significativo en el rendimiento de nuestro modelo. Por lo tanto, se ha decidido dejarlos como están y no realizar ninguna acción específica sobre ellos . Sin embargo, es importante destacar que previamente se ha tenido que reemplazar los valores igual a cero en las columnas 'x' y 'z', lo que claramente nos indica o nos da cierta sospecha de  que existen anomalías en el dataframe. Esta podría ser una razón válida para considerar la eliminación de filas que contengan valores atípicos en algunas de las columnas, ya que podrían ser sospechosos de ser anomalías. No obstante, dado que los valores atípicos solo representan el 0.49% de todos los datos en las columnas numéricas, hemos decidido dejarlos como están debido a la duda sobre su impacto en el modelo.

***Un poco de analisis bivariado/multivariado...***

Cuando comparamos pares de variables, estamos tratando de encontrar conexiones o relaciones entre ellas. Esta información es muy importante porque nos ayuda a identificar cuáles variables pueden ser las mejores para predecir en nuestro modelo. Además, nos permite detectar si existen relaciones no lineales entre las variables, es decir, si la relación entre ellas no sigue una línea recta.

Observemoslo con una imagen



s

 en el gráfico de dispersión generado por sns.pairplot(df), se puede ver  una relación lineal entre las variables "x" y "y" con respecto a la variable "price", esto podria  significa que existe una asociación lineal entre esas variables. En otras palabras, a medida que los valores de "x" aumentan, los valores de "y" también tienden a aumentar o disminuir de manera lineal, y esto a su vez tiene un impacto en los valores de "price".

Este tipo de relación lineal entre "x", "y" y "price" puede ser útil en el análisis de datos, ya que indica que "x" y "y" pueden ser buenos predictores de "price" en un modelo de regresión lineal. Esto significa que se puede utilizar una ecuación lineal para predecir los valores de "price" en función de los valores de "x" y "y".

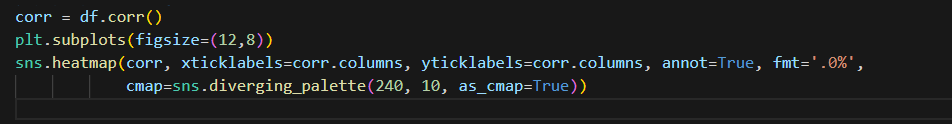
**Matriz de correlacion**

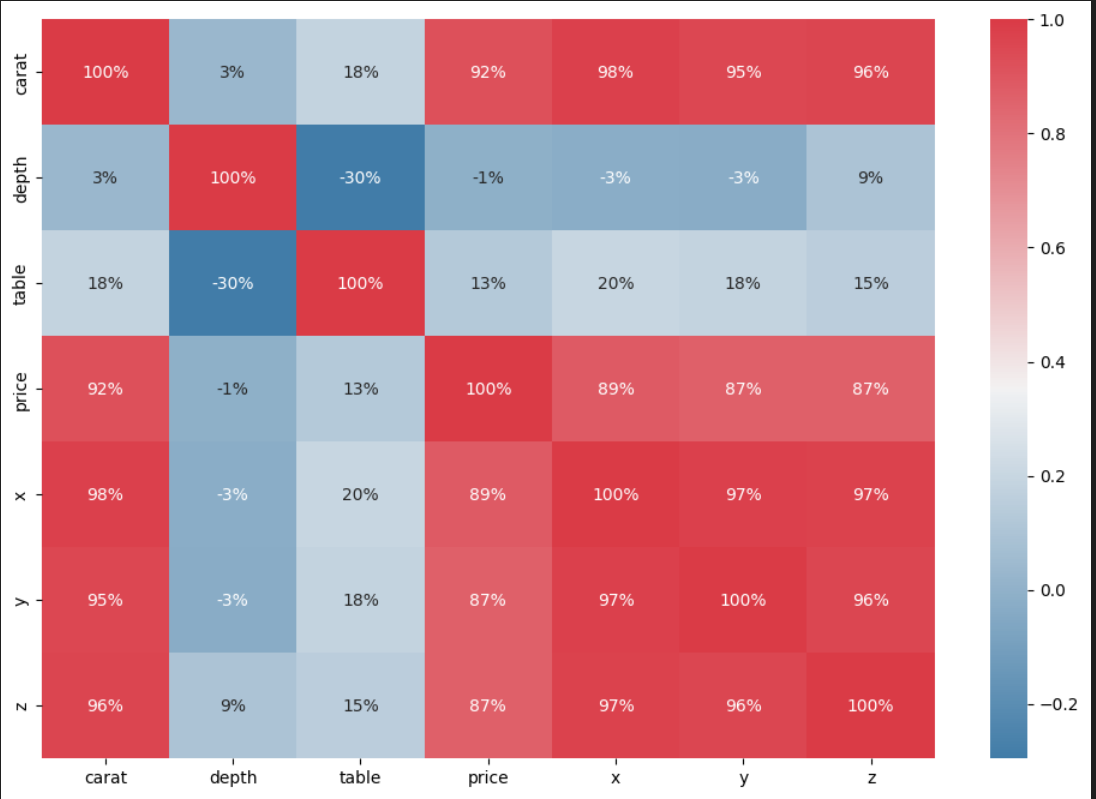
Cuando queremos ver cómo se relacionan entre sí las diferentes variables en nuestros datos, utilizamos algo llamado correlación. La correlación nos muestra qué tan fuerte es la relación entre dos variables.

Para representar estas correlaciones se debe observar los cuadros o cajas de diferentes colores. Cuanto más intenso sea el color del cuadro, mayor será la magnitud de la correlación.

También tenemos números dentro de los cuadros. Estos números nos indican qué tan cerca está la correlación de 1. Si el número es positivo, significa que hay una relación positiva entre las variables. Si es negativo, significa que hay una relación negativa.

Cuando la correlación es igual a 1 o -1, eso significa que la relación entre las variables es perfecta.





Se observa  que las variables "table" y "depth" tienen una correlación baja con la variable dependiente "price"  y  podriamos considerar que su capacidad predictiva es limitada o pobre según los numeros que se ven. Entonces  seria factible la eliminacion de estas variables del dataframe.  Al eliminar estas variables, nos estaríamos deshaciendo de aquella información que se considera que no aporta de manera significativa a la predicción del precio. Aun asi hay que tener en cuenta algo ,el precio del diamante lo determina sus caracteristicas y esos 2 atributos por mas baja correlacion son determinantes para el precio de un diamante, solo que en menor medida según los resultados. Vamos a proceder a dejarlas asi ,para que mas adelante sea el algoritmo de eliminacion hacia atrás el que decida que columna deben desaparecer.

Ya terminamos con la parte de analisis de datos.

Como ya se fue adelantando previamente, lo que se busca predecir es el precio del diamante. Se entiende que el precio esta determinado segun las caracteristicas que el diamante posea.

Por lo tanto el precio va a ser la variable dependiente mientras que el resto van a ser las independientes.

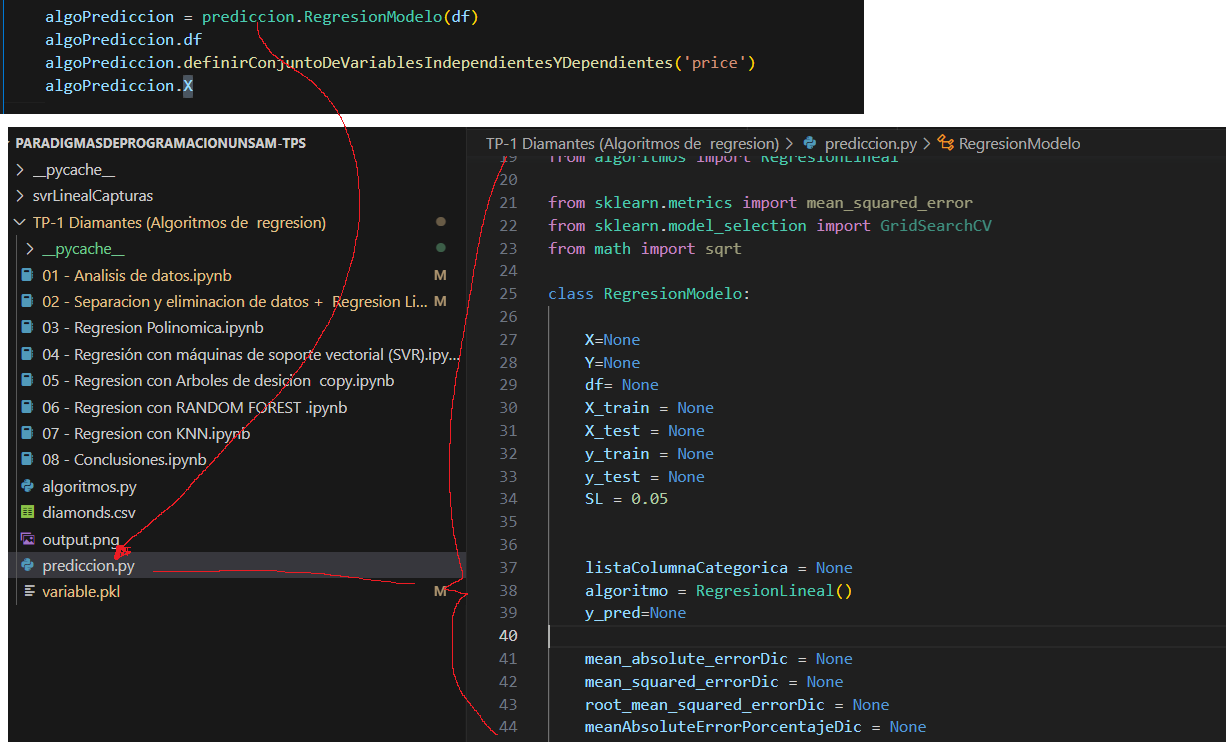
**Separacion de variables dependientes de las independientes**

Antes de comenzar con la separación de los archivos, es importante destacar que se ha optado por utilizar el paradigma de objetos en este trabajo práctico. Esto se hace con el objetivo de facilitar la reutilización de métodos y funciones en diferentes notebooks de Jupyter, como se puede observar en la imagen a la izquierda.

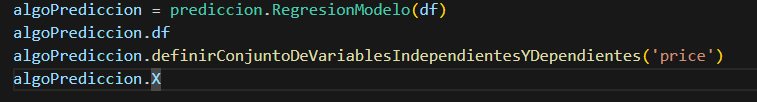
Existen varios archivos .ipynb, siendo el primero "01 - Análisis de datos" el que ya ha sido explicado. A continuación, se encuentra el archivo "02..." y así sucesivamente.

En la mayoría de estos archivos, excepto el último llamado "Conclusiones.ipynb", se hace uso constante de la clase "RegresionModelo". Esta clase se encarga de la separación de las variables dependientes e independientes, la eliminación de variables que afectan negativamente al rendimiento del algoritmo y la asignación del algoritmo de predicción a ejecutar, ya sea lineal, de árboles, polinómico, etc. Tambien se encarga de dar la orden al algoritmo indicado de realizar la  predicción para asi finalmente  obtener los resultados para mostrar en pantalla mediante graficos. Además, esta clase también cuenta con algunas funciones menores que se irán explicando gradualmente. No obstante, se intentara explicar la funcionalidad completa de este objeto en su totalidad. Esto solo es una pequeña introduccion

Se ha decidido separar las predicciones en varios archivos de Jupyter Notebook, que van desde el "02" al "07", y así sucesivamente. La razón detrás de esta decisión es evitar que un solo archivo sea demasiado grande y más difícil de mantener, así como también reducir la posibilidad de cometer errores. Además, hay algoritmos que requieren un tiempo considerable para realizar sus predicciones, como es el caso de la regresión con máquinas vectoriales, y no es conveniente tener que esperar a que se complete la ejecución de ese algoritmo para poder ejecutar otro.

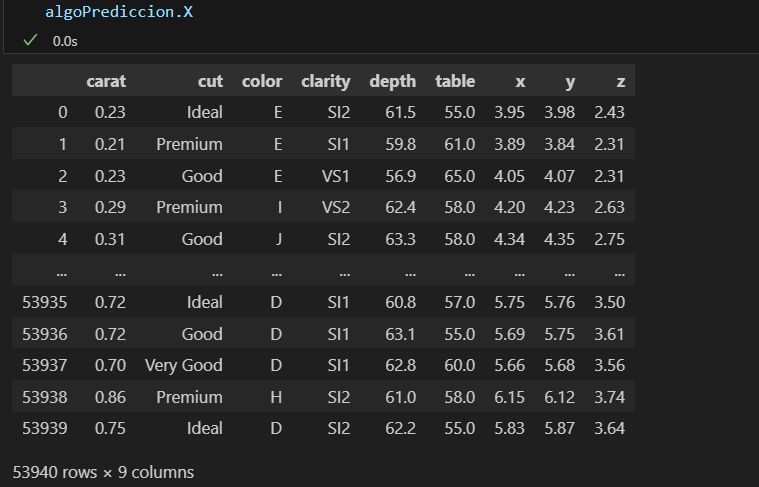


El obteto Regresion modelo recibe como parametro al df y luego se ejecuta el metodo definitiConjuntoDeVariables...  que es un metodo que recibe un string que indica que que del dataframe que le asignamos en el constructor al objeto todas las columnas salvo price son variables independiente y justamente price es la dependiente y la que se intenta predecir

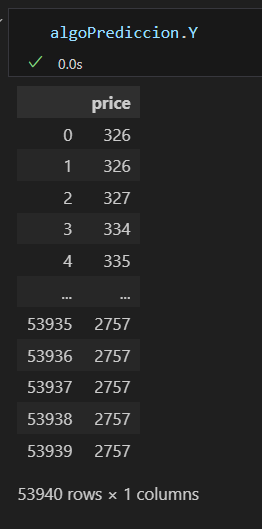


Por lo general se trato de darle nombre a los metodo los mas descriptivos posibles





Ya tenemos el conjunto de variables dependientes e independientes

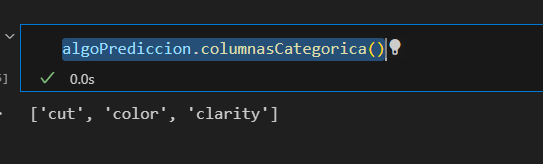


De categoricos a valores  numericos

Ya de antemano contamos con variables independientes categoricos.<br>

El siguiente paso es convertir los datos de aquellas columnas en datos numericos <br>

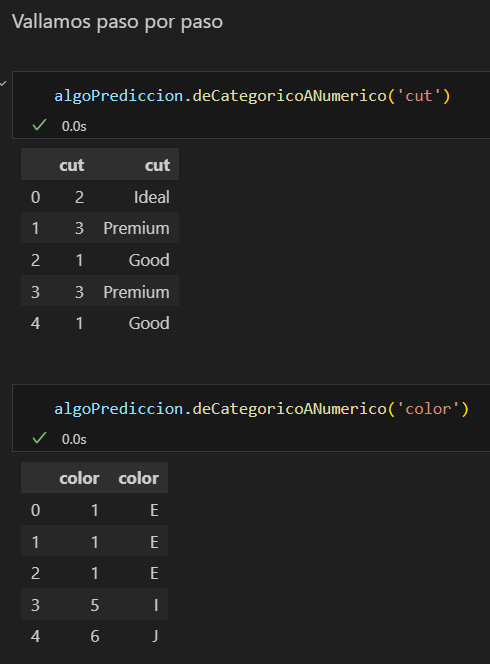
Las variables independientes categoricas son

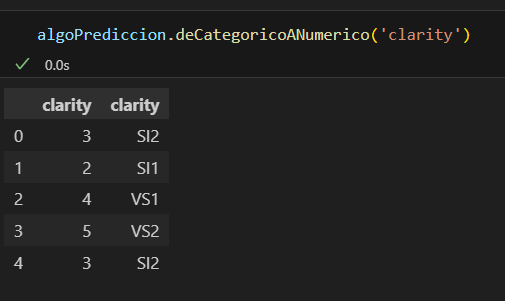


Metodo que solo retorna solo los nombres de aquellas columnas que no almacen numeros osea las que son categoricas.

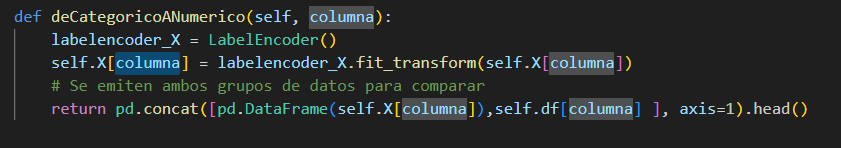


Aca abajo vemos la ejecucion del metodo que recibe por parametro el nombre de la columna que  va a pasar de categorico a numerico



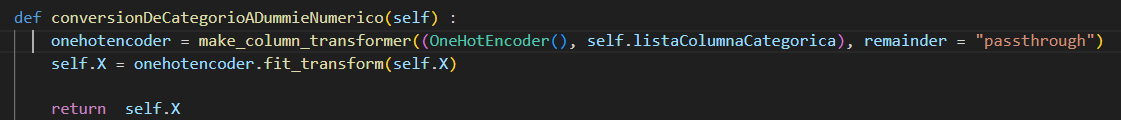


La tabla que se muestra es como quedo numericamente la tabla. Vemos para este caso que  al valor S12 Se le asigno el numero 3 (se comprueba en la fila 0 y 4)



Las variables categóricas, como la talla (Cut), el color (Color) y la claridad (Clarity), se podrian ver  beneficiados de la codificación one-hot, que implica crear variables dummy para cada categoría en lugar de escalar los datos. La razón principal es que estas variables no tienen un orden inherente y no se puede establecer una relación numérica directa entre las diferentes categorías.

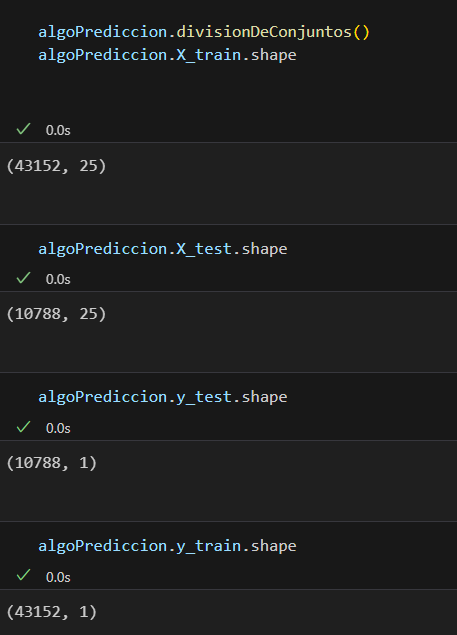


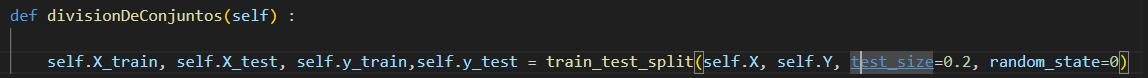


Evitamos la trampa de los valores dummies

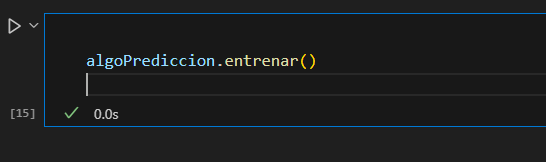


A continuación, dividimos el 80% de los datos para el conjunto de entrenamiento y  el 20% de los datos al conjunto de pruebas usando el código de abajo. Y por ultimo mostramos su dimension





El siguiente paso es el entrenamiento del modelo.





Es importante destacar que el objeto "RegresionModelo" tiene predefinido el modelo de regresión lineal como algoritmo por defecto. Como mencioné anteriormente, el objeto "RegresionModelo" delega la tarea de hacer las predicciones y devolver los resultados a otro objeto, en este caso, "RegresionLineal". Esto permite que el objeto "RegresionModelo" pueda generar los gráficos correspondientes.

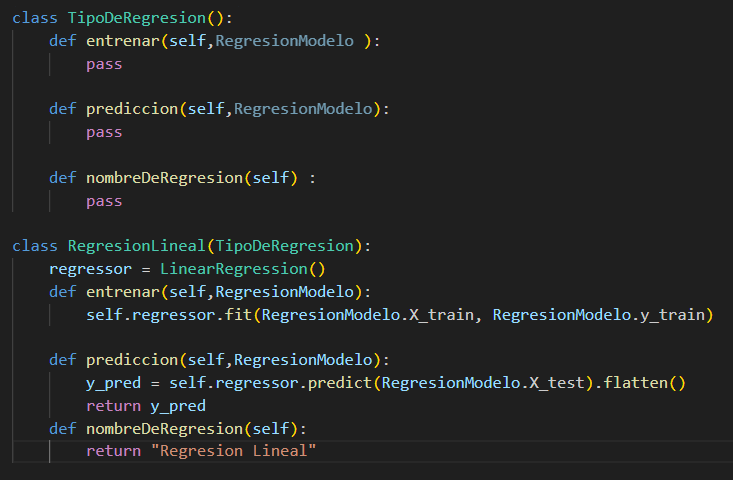
Cuando pasamos al siguiente algoritmo, simplemente necesitamos cambiar el atributo "algoritmo" del objeto "RegresionModelo" por otro algoritmo específico. Es importante destacar que se está utilizando el patrón de diseño "strategy", el cual nos permite intercambiar diferentes algoritmos en tiempo de ejecución.

En resumen, el objeto "RegresionModelo" tiene una configuración predeterminada con el modelo de regresión lineal, pero puede cambiar de algoritmo fácilmente gracias al patrón "strategy". Esto brinda flexibilidad en la elección y evaluación de diferentes algoritmos de regresión.

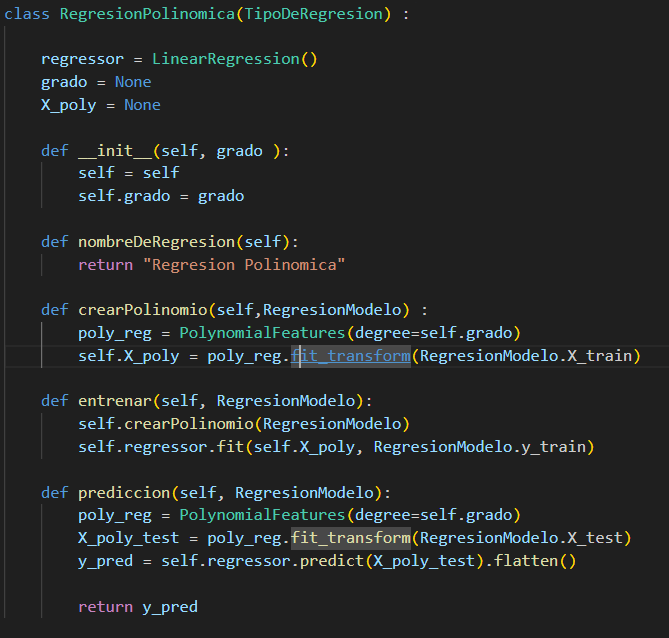


***Implementacion del patron de diseño.: P***ython no tiene interfaces pero si tiene Herencia multiple. Asi que tranquilamente una clase puede hacer indirectamente de rol de interfaz.

En este caso el strategy es RegresionLineal y debe implementar obligatoriamente como si fuera un contrato los metodos de TipoDeRegresion. Como podemos ver RegresionLineal tiene el metodo entrenar que justamente hace eso, entrenar el modelo utilizando el conjunto de entrenamiento del objeto Regresion Modelo que recibe por parametro todo el objeto RegresionModelo para asi poder hacer uso de sus atributos para realizar el entrenamiento y luego retornar la prediccion .

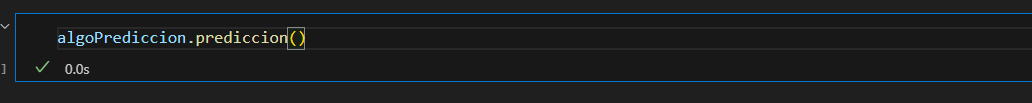


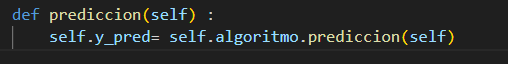
Obviamente hay mas strategys que representan a cada algoritmo visto en la materia. Aca vemos el caso del Polinomio. Tiene los mismos metodos que regresionLineal solo que responde a cada uno de manera diferente. Osea existe el polimorfismo y por eso en este tp es muy facil  cambiar de algoritmo de regresion . Simplemente basta con modificar el atributo algoritmo del objeto RegresorModelo



**Predicción sobre los datos de la prueba:**

Despues del entrenamiento viene los resultados de la prediccion y de eso se encarga el strategy,



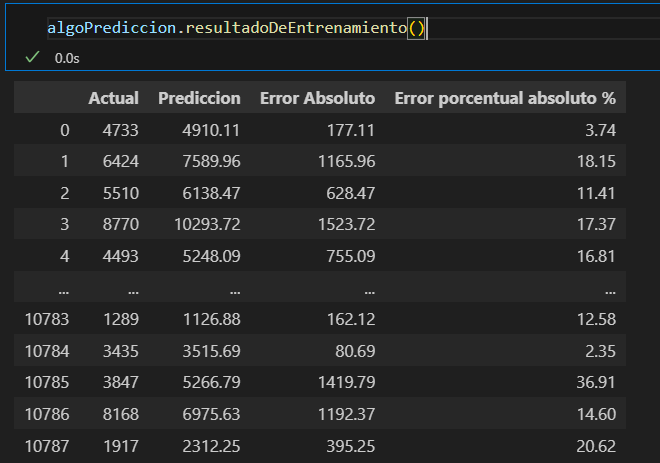


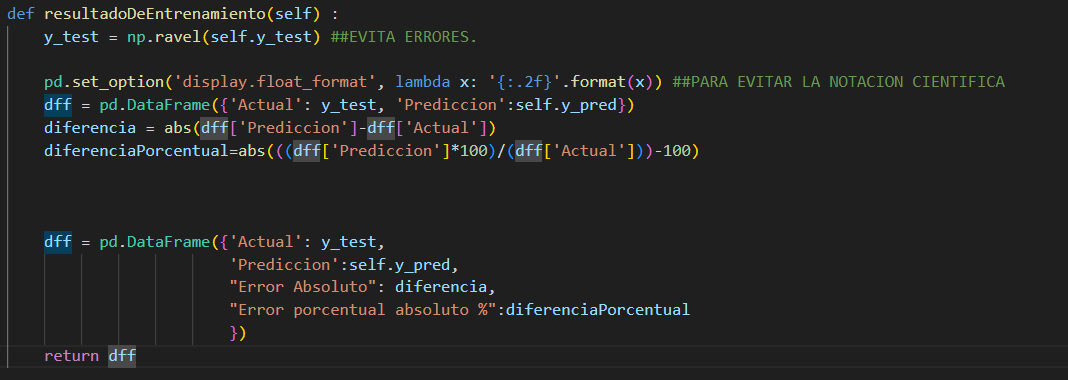
Por supuesto los resultado se guardan en el atributo y\_pred para asi no tener que volver a realizar el llamado



Aclaracion : nombreDeRegresion es un metodo relevante para el final .Tampoco es importante entender su razon de existir

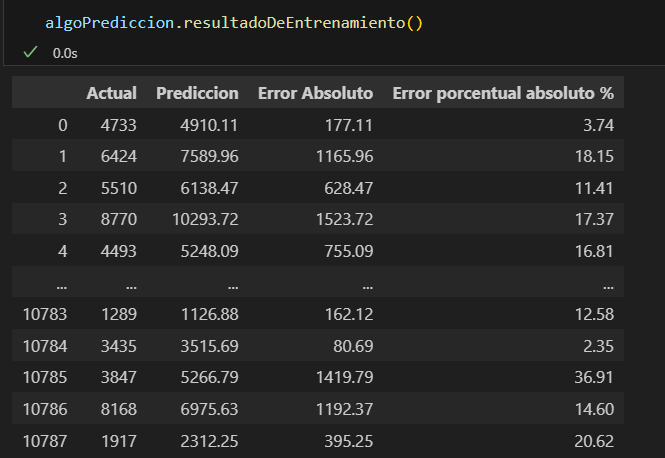
***RESULTADOS DE LA PREDICCION***





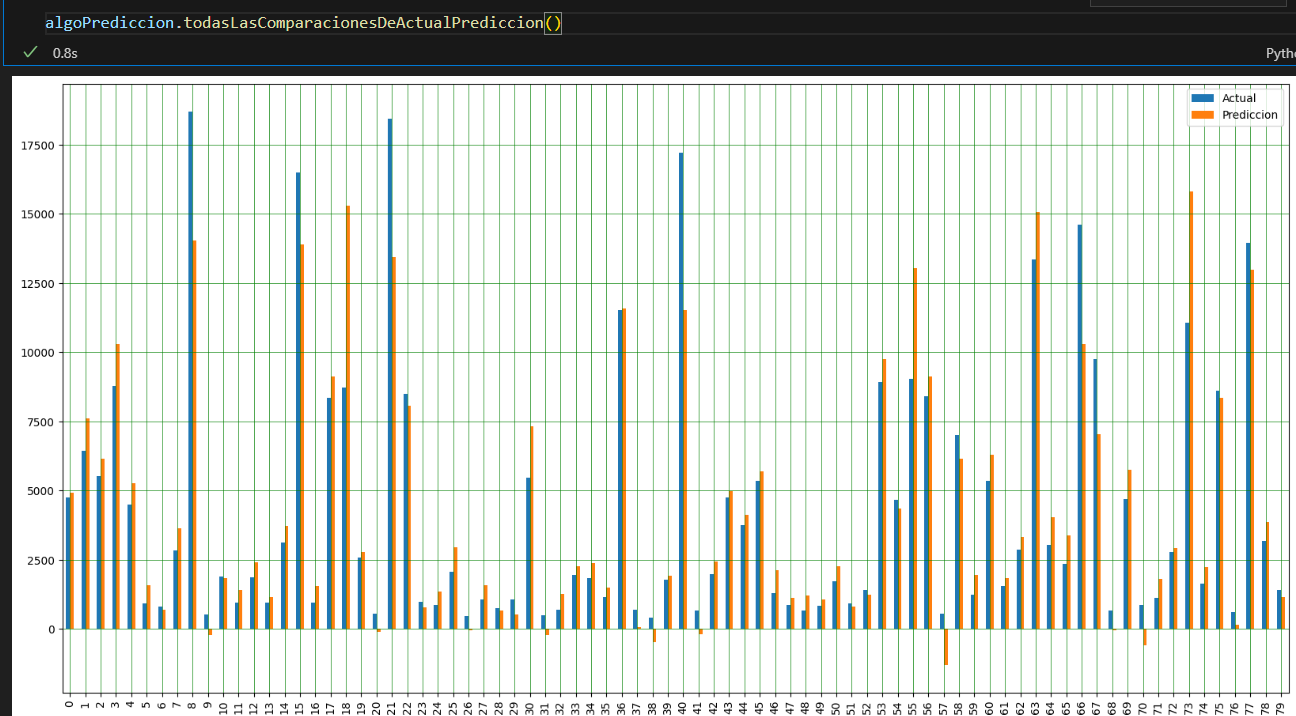
Como se puede ver este metodo tiene como objetivo mostrar los resultados de la prediccion en formato de tablas. Y hace uso del atributo y\_pred  obtenido en el metodo  entrenar y prediccion() que hace uso del strategy.

Veamos que se muestra



Por supuesto el Error absoluto es para cada caso (Diamante predecido en particular) . Vemos que el primer caso el error solo fue de un 3.74 % con una diferencia de 177.11 unidades (o dolares si tenemos en cuenta que estamos intentando predecir precios) . 3.74 % de error un numero muy bueno pero al parecer no es normal que suceda eso como se puede ver en el resto de las filas. Por las dudas  aclaro que el error absoluto es la diferencia entre Actual y prediccion. Esta tabla se va a repetir en el resto de los algoritmos de prediccion.

**Es hora de los graficos**

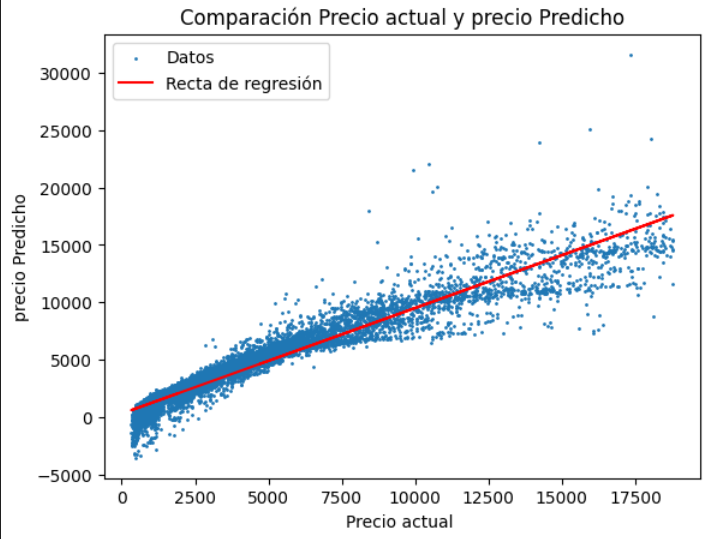


Se muestra una comparacion visual entre el valor actual y el predecido. Solo en 80 casos de lo contrario es mas dificil comprender el dibujo.  En lineas generales no esta tan mal.

Antes de continuar veamos el codigo del metodo que se ejecuto en la imagen de arriba



Los nombres de los métodos no son los más apropiados, sin embargo, cambiarlos en este punto podría ser riesgoso. Lo más relevante aquí es que al ejecutar el método "todasLasComparacionesDel..." se muestran automáticamente tres gráficos en la pantalla del Jupyter Notebook.

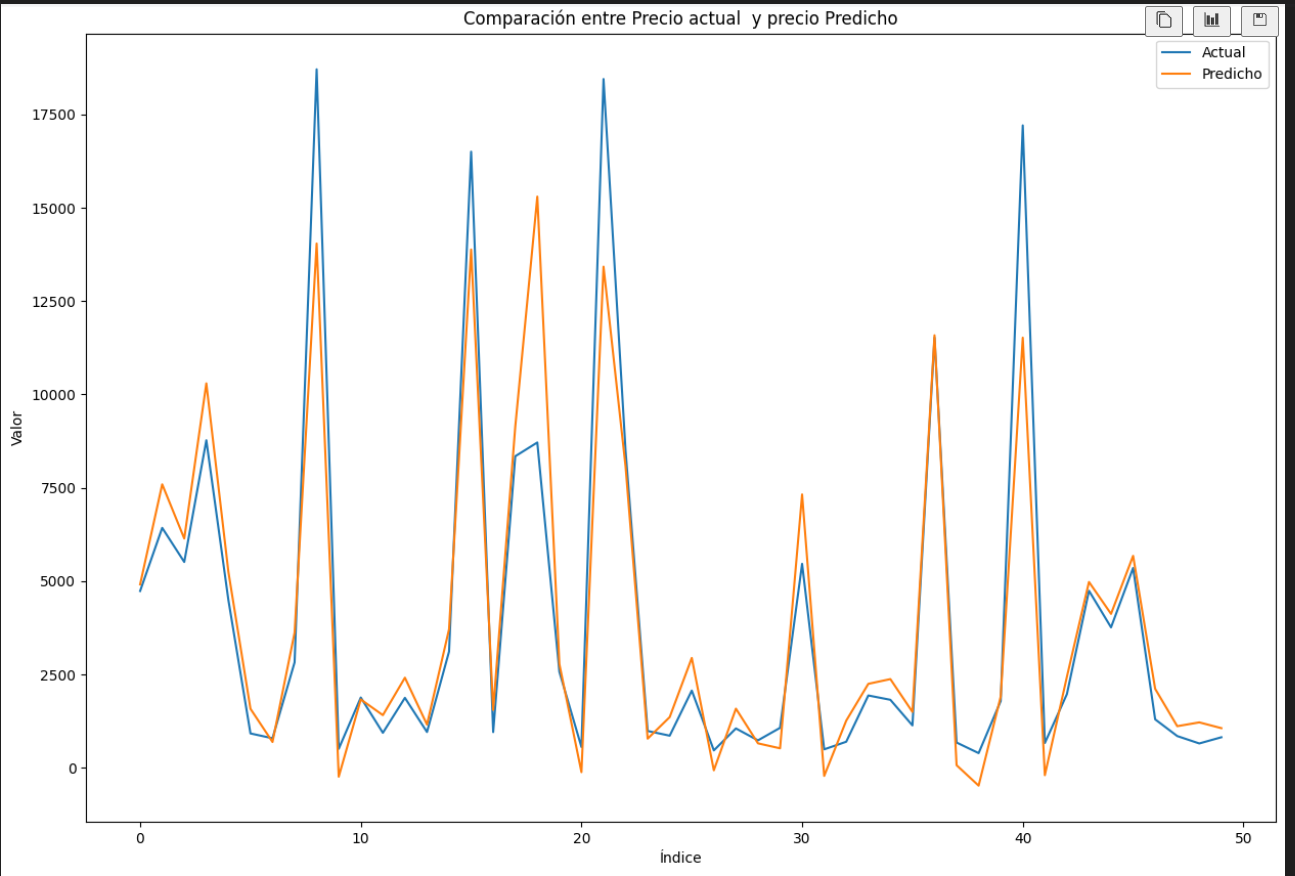


En el gráfico resultante, los puntos dispersos representan los pares de valores reales y predichos. Si observamos puntos que están lejos de la recta de regresión generada (la línea roja), podria significar que hay discrepancias considerables entre los valores reales y los valores predichos correspondientes a esos puntos.

Cuando decimo s que un punto esta "mas lejos" de la recta de regresión generada, significa que hay una mayor diferencia entre el valor predicho y el valor real correspondiente. En la regresión lineal ideal, los puntos de dispersión estarian muy cerca de la recta de regresión, lo que indicaría una buena capacidad de predicción.

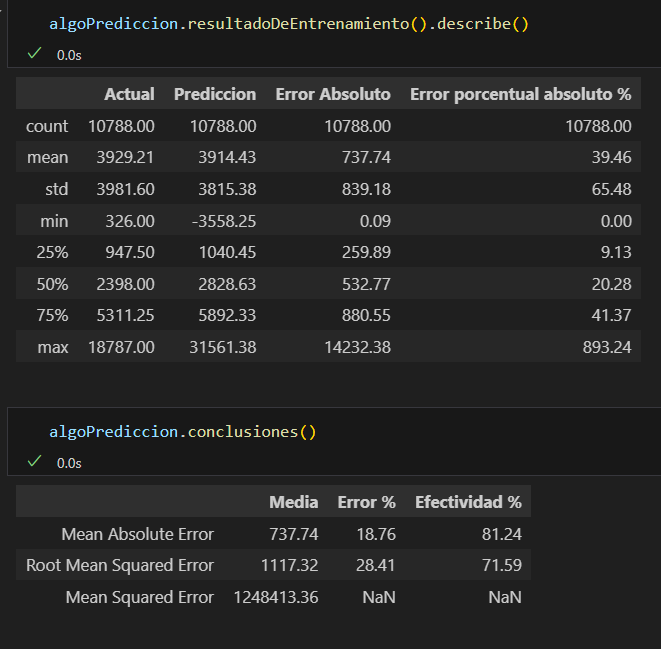
Sin embargo, si hay puntos que se notan claramente que estan alejados de la recta de regresión,  lo que podría indicar una falta de ajuste o un rendimiento ineficiente del modelo en algunos casos

En el gráfico resultante, los puntos dispersos representan los pares de valores reales y predichos. Si hay puntos que están lejos de la recta de regresión generada (la línea roja), significa que hay discrepancias considerables entre los valores reales y los valores predichos correspondientes a esos puntos.



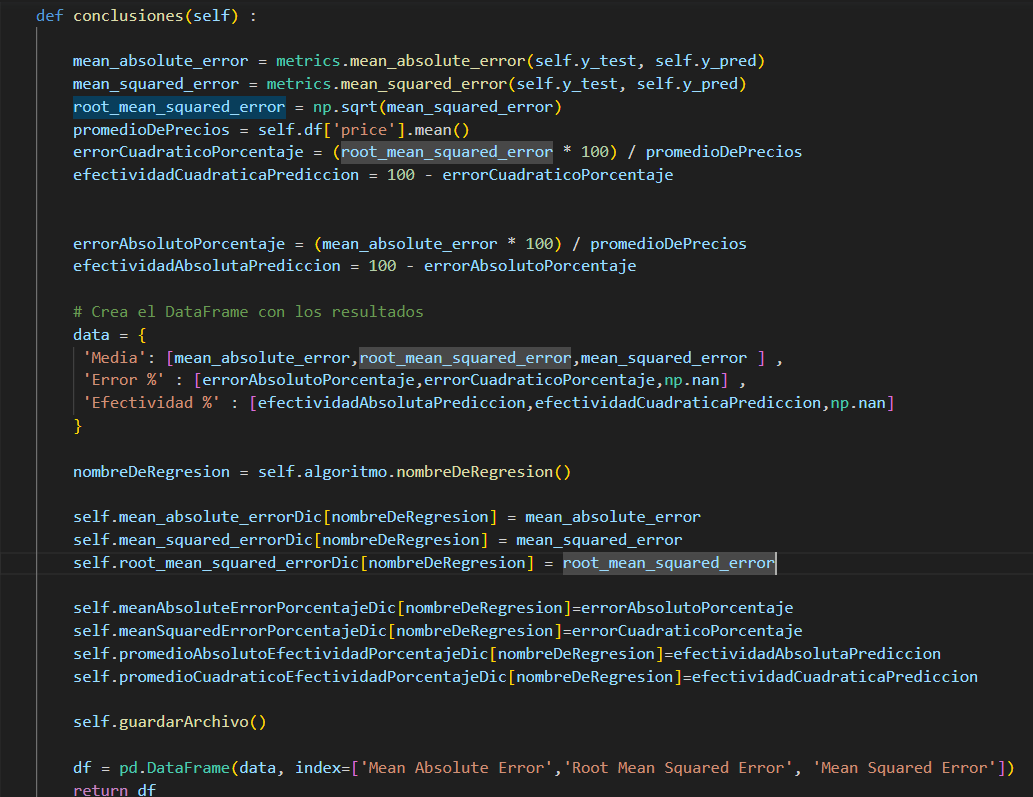
El conjunto de datos utilizado abarca 50 casos, lo cual proporciona una muestra más amplia para comprender mejor los resultados. La idea principal es evaluar la coincidencia entre dos líneas en un punto específico (x, y), y se considera que cuanto mayor sea la superposición entre ambas líneas, mejor será la coincidencia.

Por ultimo la conclusion

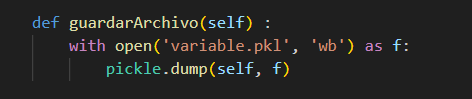


Primero, mostramos los valores estadísticos, los cuales utilizan un método previamente explicado. Al analizar estos resultados, es interesante observar, por ejemplo, que el mejor resultado presenta un error de 0.09 dólares, lo cual indica un error porcentual muy bajo, cercano al 0%. Esto sugiere que, a pesar de los resultados de la tabla de conclusiones, nuestro algoritmo es capaz de realizar predicciones precisas en varios casos.

Ahora, procedemos con el método de conclusiones. Este método nos permite evaluar qué tan buenas o malas fueron nuestras predicciones. Antes de continuar, revisemos el código.



Por ahora, es suficiente con saber que el método CONCLUSION() devuelve una tabla y, además, va guardando valores en cuatro atributos que son un diccionarios del objeto RegresionModelo. Estos atributos se encuentran por encima del método guardarArchivo() y van a ser extremadamente utiles para comparar el rendimiento de todos los algoritmos en la conclusion final. Sobre el metodo guardarArchivo()  no es necesario prestarle demasiada atención ya  que su función principal es guardar el objeto RegresionModelo, junto con sus estado, en un archivo para poder ejecutarlo en otro Jupiter Notebook.



BIEN veamos la tabla que se retorno en el metodo CONCLUSION() nuevamente

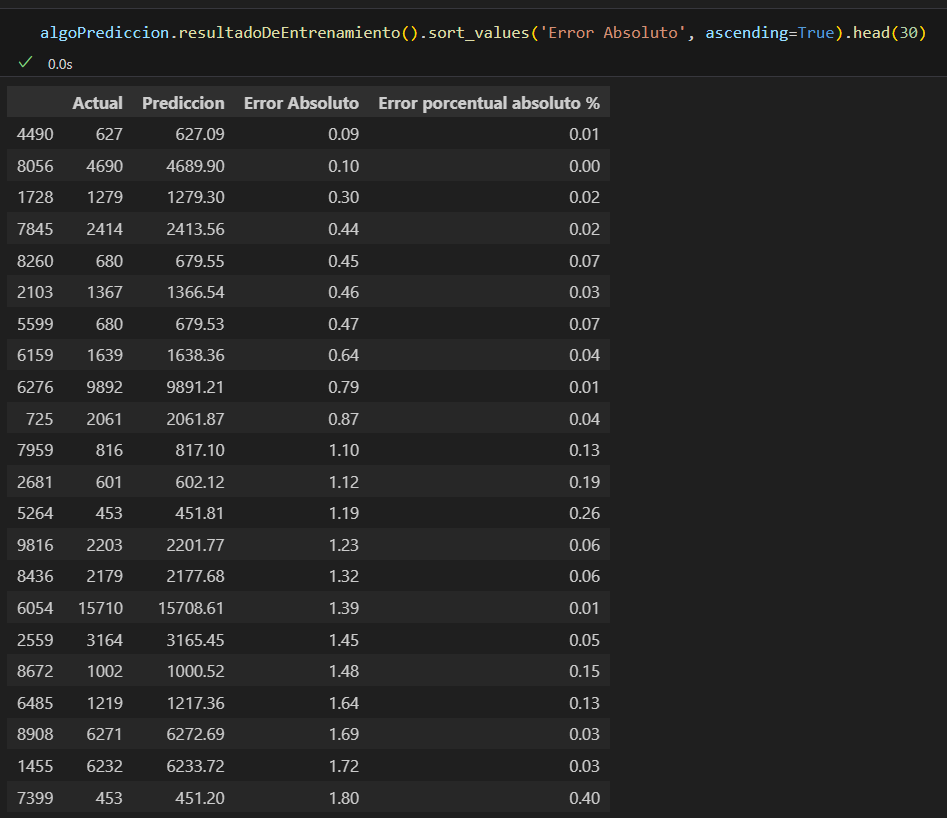


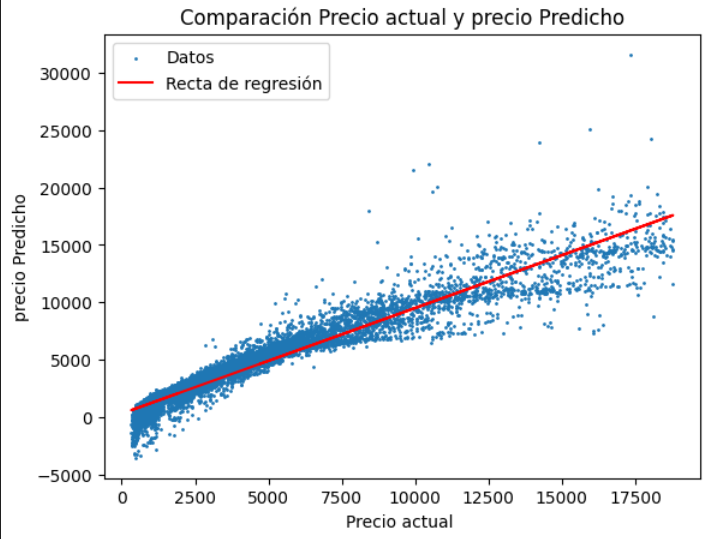
Para la fila que abarca Mean Absolute Error (Error Absoluto Medio): El valor de 737.78 indica que, en promedio, las predicciones del modelo tienen una diferencia absoluta de 737.78 unidades con respecto a los valores reales. Un menor valor de MAE indicaria un mejor resultado y por lo tanto nos diria que existe una mayor precisión del modelo. El error porcentual asociado es del 18.76%, lo que implica que el algoritmo de regresion lineal se equivoca en un promedio del 18.76% al realizar las predicciones. Por lo tanto teniendo en cuenta el error porcentual asociado y haciendo una sencilla cuenta de 100-Error% nos daria que tan efectivo fue nuestro algoritmo .En este caso en particular nos arroja un resultado de un 81.24%

Para la fila que abarca Root Mean Squared Error (Error Cuadrático Medio): El valor de 1117.33 representa la raíz cuadrada del error cuadrático medio, lo que indica una dispersión promedio de 1117.33 unidades entre las predicciones (y\_pred) y los valores reales (y\_train). Tal cual como pasa con el MAE, un valor menor de RMSE indica una mayor precisión del modelo. El error porcentual asociado es del 28.41% y la efectividad se encuentra en un 71.59%.

Sobre el Mean Squared Error (Error Cuadrático Medio): El valor de 1248435.26 representa el error cuadrático medio, que es el promedio de los errores al cuadrado entre las predicciones y los valores reales. En este trabajo practico se decidio de forma arbitraria no proporcionar datos  sore  el error porcentual y su efectividad para MSE

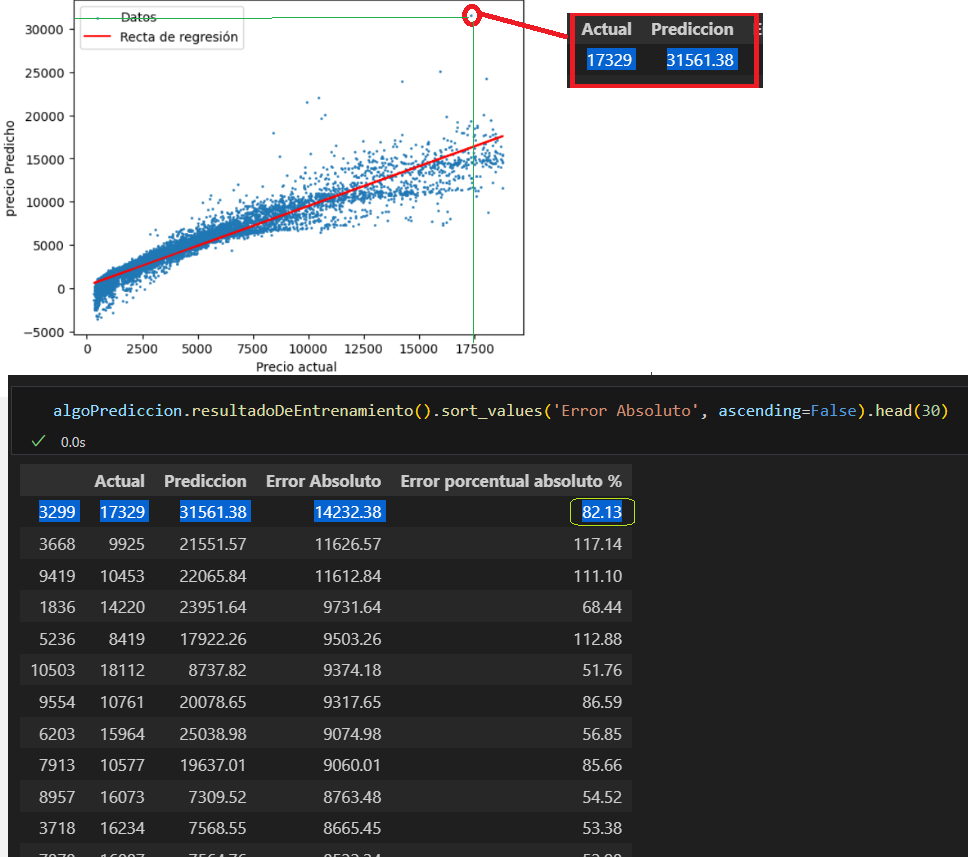
Sin embargo de las 3 metricas  que se ve en la tabla de arriba la que nos mas interesa es el error Cuadratico y su efectividad ya que esta ultima es mas sensible porque penaliza de manera más significativa los errores más grandes, ya que se elevan al cuadrado por lo que la conclusion definitiva para este caso seria que nuestro algoritmo fue efectivo en un 71.59 y tuvo un error de un 28.41%. Aun asi el algoritmo puede realizar buenas predicciones de hecho en algunos casos logra muy buenos resultados  como se pudo ver en la tabla y en el grafico de abajo que existen casos donde la prediccion es realmente buena



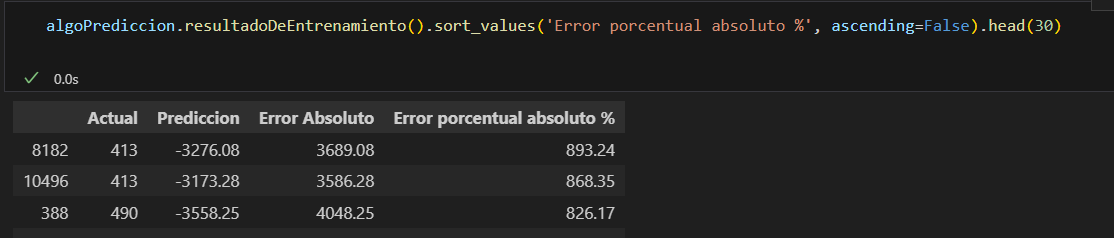


Donde claramente se ve que  existen casos en donde se logra un muy  buen rendimiento. Sin embargo vemos que a medida que los precios tienden a ser mas alto la dispersion es mucho mayor hasta lograr una una diferencia entre el valor real y predicho muy grande magnitud en terminos relativos.  Expliquemos esto ultimo

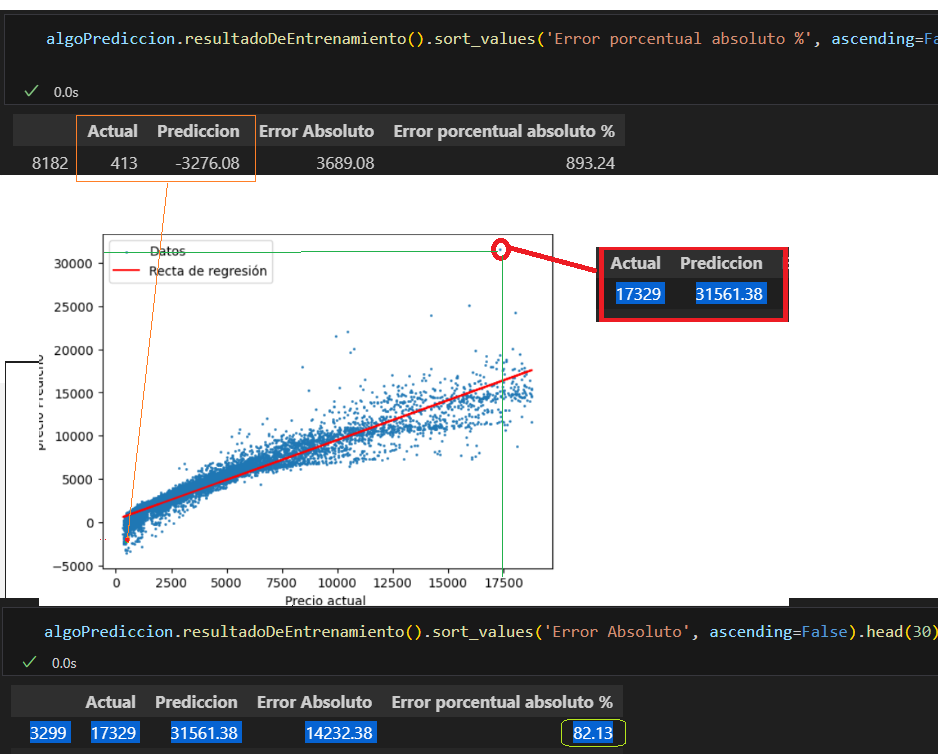
Observemos  primero la tabla  y el grafico de abajo



La tabla de arriba esta ordenada de forma descendente teniendo en cuenta los valores de la columna Error Absoluto.  Sin embargo si observamos la primera fila vemos que el error absoluto es el mas grande ,aun asi  no es el peor resultado de prediccion de nuestro algoritmo. Observemos que su error porcentual es de un 82% sin embargo existe casos peores , simplemente basta con ver la fila de abajo que da un error porcentual absoluto del 117.14. Si no existiese el error porcentual absoluto en la tabla uno al ver los resultados diria que el resultado de la primera fila es la peor prediccion que tuvo el algoritmo pero no es asi. De hecho existe casos peores como el de abajo



¿Nos estaria diciendo el grafico de arriba que para valores mas altos la predicciones suelen ser mas dificil? Veamos esta hipotesis con mayor claridad

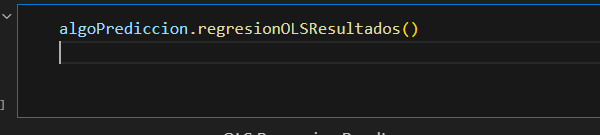


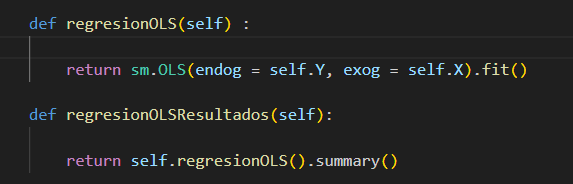
Viendo el grafico de arriba concluimos lo siguiente : la cercanía a la linea de regresion efectivamente no es un indicador definitivo de la calidad de la predicción. No debemos asumir directamente que los pares ( y\_train  y\_pred) más cercanos a la linea de regresion tienen mas probabilidades de indicar un porcentaje de prediccion mejor , ni que los puntos mas alejados  de la recta son aquellos con un resultado de prediccion peores. La linea de regresión se utiliza para estimar la relación entre las variables, pero como vemos no captura todos los detalles individuales de cada punto de datos.

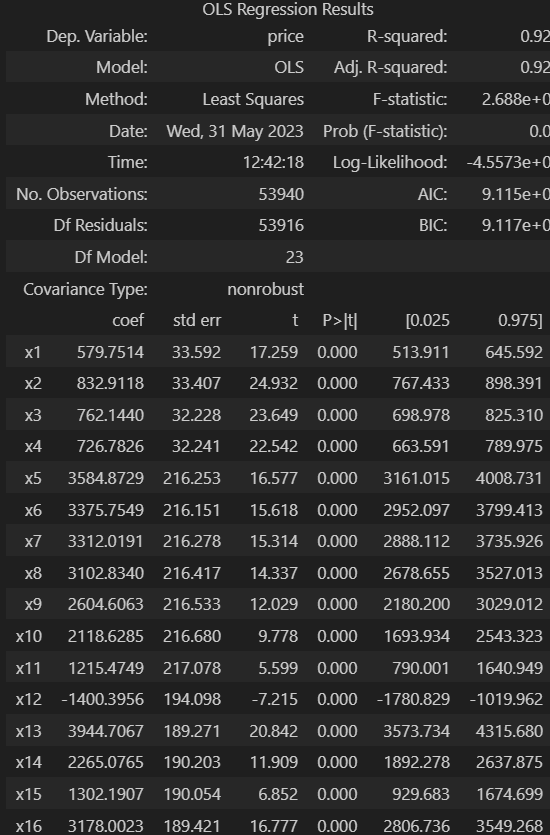
En resumen, cuando los valores reales y predichos son grandes, es posibleque exista una mayor dispercion entre los puntos y la recto. Esto puede considerarse  normal debido a la variabilidad en los datos por lo tanto no debemos hacer suposiciones automáticas sobre la precisión basadas únicamente en la cercanía a la línea de regresión. Si bien el grafico puede dar indicios de que tan buena fue la prediccion , mejor quedarse con la tabla de aca abajo.



Antes de pasar al siguiente modelo de regresión la pregunta es ¿ podriamos obtener un mejor resultado si aplicamos algun algoritmo de eliminación en aquellas variables o columnas menos significativas que no aporten mucho a los resultados de la predicciones? Veámoslo.





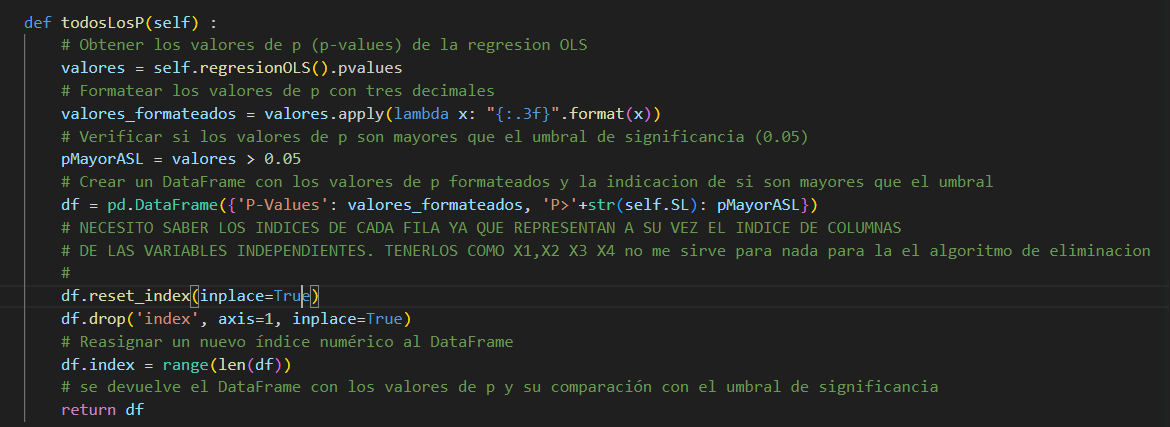




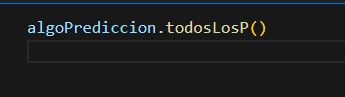
Se va a utilizar el algoritmo de eliminacion hacia atrás asi que de todo  la tabla que se muestra arriba lo unico que nos interesa son los p valores. Asi que generamos otra tabla mas simplificada

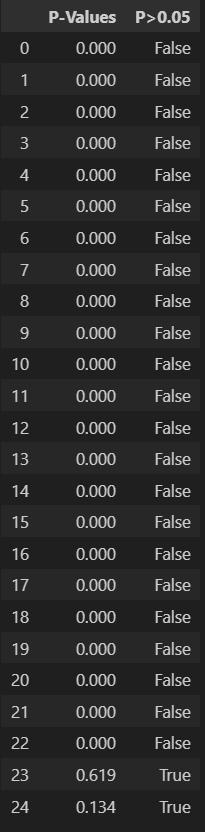
Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

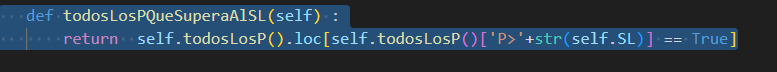
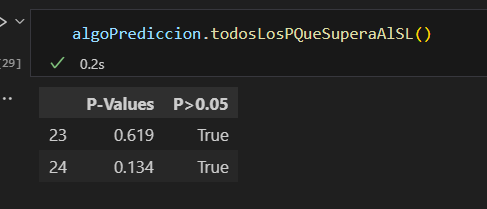


Con el codigo de arriba logramos obtener una tabla mucho mas simplificada y mas util para lo que queremos hacer . Veamos el resultado



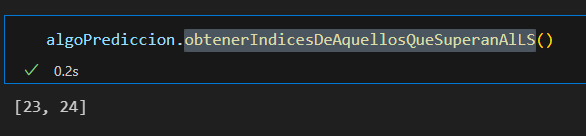


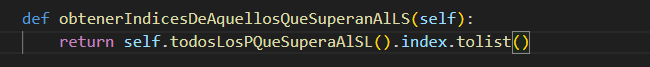
La visualizacion se hace mucho mas sencilla y directa de esta forma . Como podemos ver las 2 ufilas que representan l columnas (o variables indpendiente) cuentan con P-values que superan al umbral establecido (0.05) . Tambien podemos ver que los indices de las filas no estan reprentados por X1 X2 X3 X4 etc lo cual nos es util luego para indicar que  filas eliminar de forma automatica en un solo paso. Continuemos. Lo siguiente que se ejecuto en el jupiter notebook fue



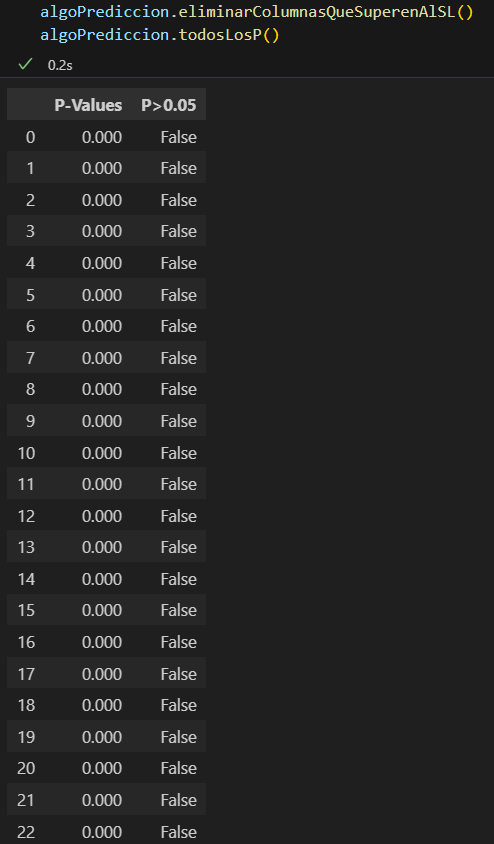
Por supuesto que el metodo de arriba seria mucho mas util si hubiera mas casos . Aun tiene mas utilidades sobre todo al momento de realizar la eliminacion. Pero visualmente sirve igual por mas pocos casos que haya para mostrar directamente aquellas columnas menos significativas para nuestro modelos que van a ser eliminadas.

Del metodo mostrado arriba nos interesa obtener los indices

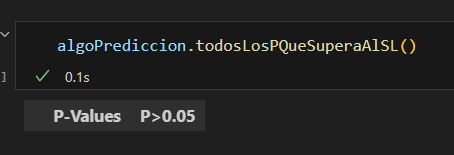




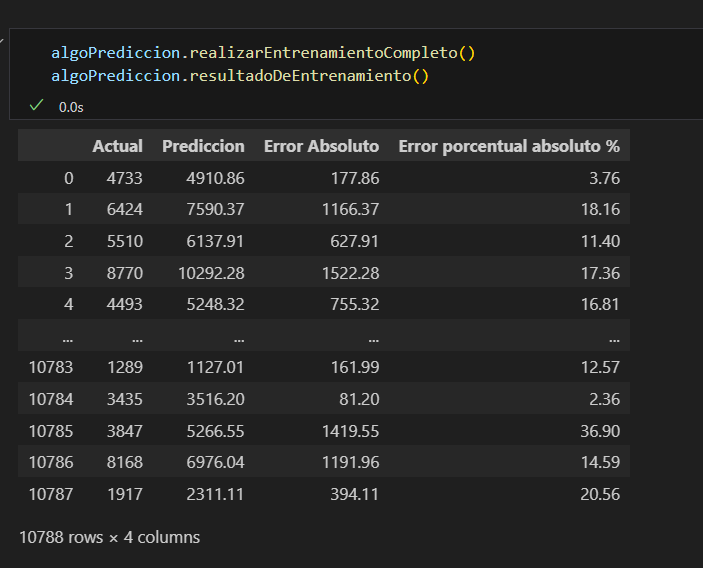
Finalmente procedemos a la eliminacion y vemos los resultados

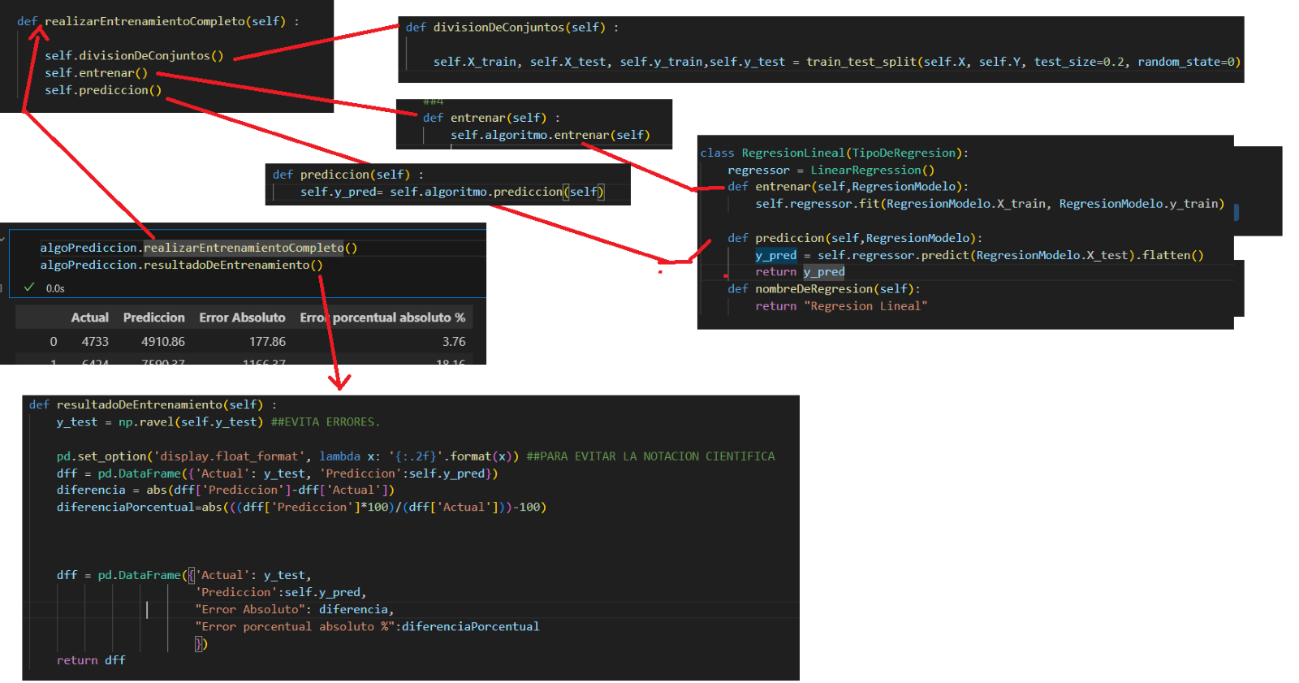


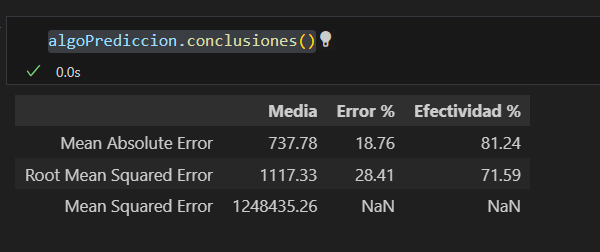
Pasamos de 25 variables a 23. Efectivamente se realizo la eliminacion. Solo para estar seguro comprobemoslo



A continuacion volvemos a realizar el entrenamiento  DESDE 0. Esta vez estamos seguro de que todas las variables que tenemos son significativas para nuestro modelo.







Volvemos a obtener los mismos Resultados!!

El objetivo de la eliminación hacia atrás era reducir el modelo de regresión y quedarnos solo con las variables más importantes y relevantes , en simples palabras reducir la complejidad. Inicialmente se contaban 25 columnas que representan las  variables independientes (X), pero después de aplicar la eliminación hacia atrás,se eliminaron 2 columnas y finalmente nos quedamos  con 23.

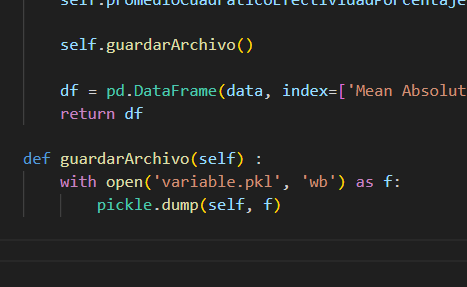
Lo interesante es que, a pesar de haber eliminado esas dos columnas, los resultados de predicción no cambiaron. Esto nos indica que las dos columnas que se elimino no eran tan importantes para hacer predicciones precisas.

Podemos concluir que las variables en esas dos columnas no estaban aportando información significativa para predecir la variable dependiente.

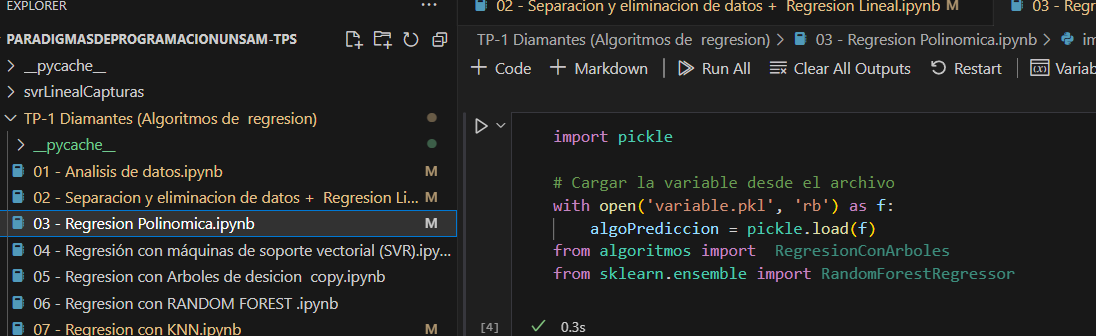
Esto significa que se puede simplificar el modelo al eliminar esas dos columnas, sin comprometer la calidad de las predicciones.

***REGRESION POLINOMICA***

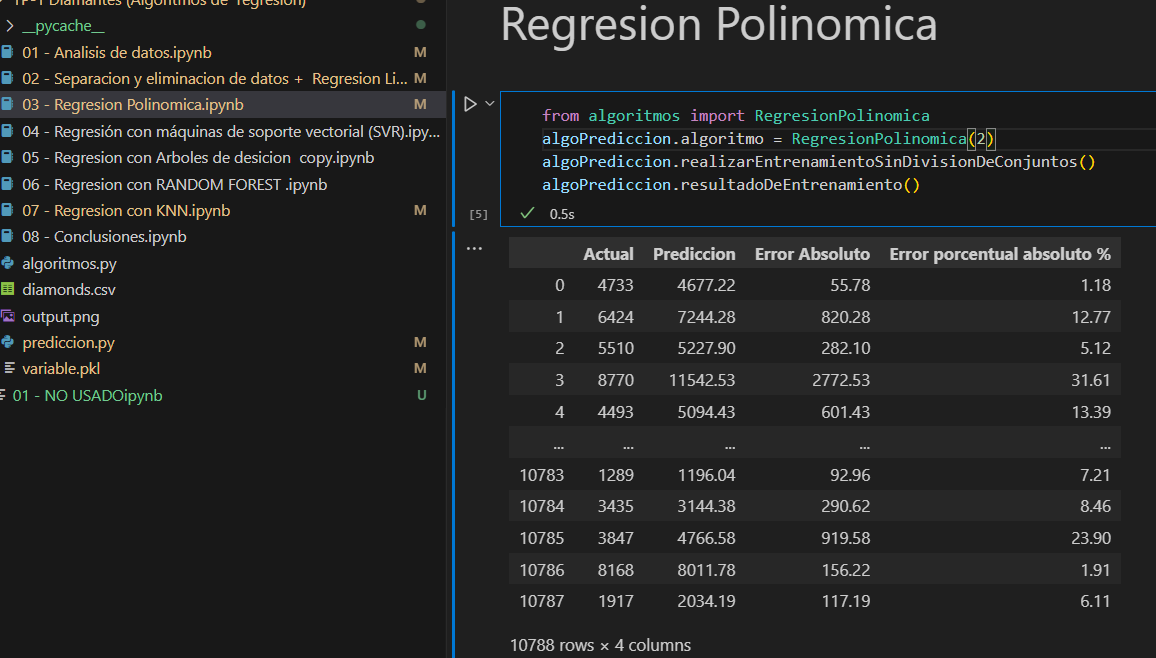
Recordemos que cada vez que se ejecutaba el metodo conclusion se realizaba un guardado de un archivo que mantenia el estado del objeto RegresionModelo



Como pasamos a otro archivo jupiter notebook tenemos que recuperar el estado de ese objeto para si evitar realizar los pasos como por ejemplo la separacion de variables independientes de las dependientes , eliminacion hacia atrás etc. Asi que lo siguiente que se hace es cargar ese archivo que previamente se guardo. Recordemos que lo ultimo que se ejecuto en el el archivo “02- Separacion y eliminacion de datos + regresion lineal.ipynb “ fue el metodo conclusion() Asi que ya estamos asegurados que vamos a recibir el objeto con los ultimos cambios realizados.



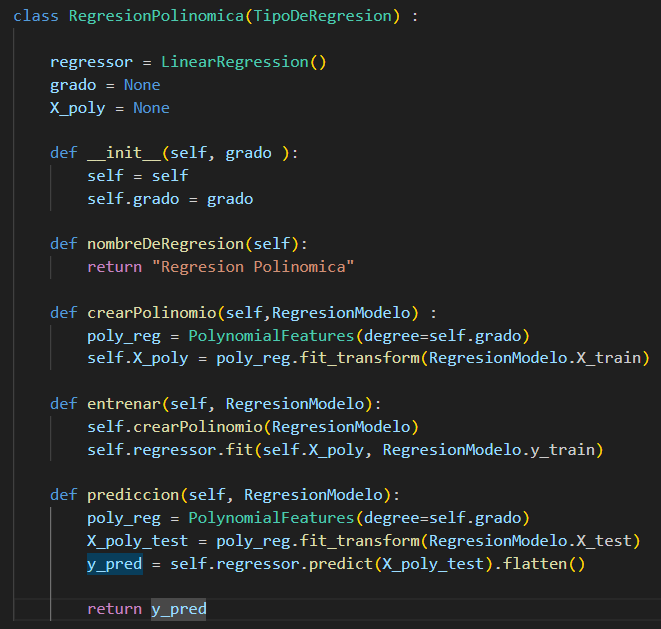
El siguiente paso es cambiar de algoritmo . Hasta ahora sigue con el modelo de regresion lineal. Hacer esto es muy sencillo. Como se puede ver abajo



Bueno ya obtuvimos los resultados con solo ejecutar 2 metodos!! Aca se empieza a  ver la ventaja de tenerlo todo encapsulado en un objeto  usando el paradigma de objetos justamente.  La gran ventaja tambien es la limpieza que maneja los archivos jupiter notebook. Si bien podemos llenarlo de codigo a la larga se complica mantenerlo.

Volviendo a lo que importa , De entrada vemos que le estamos asignando un nuevo algoritmo de regresion ,en este caso el modelo de regresion Polinomica y le estamos definiendo en el constructor un Grado 2





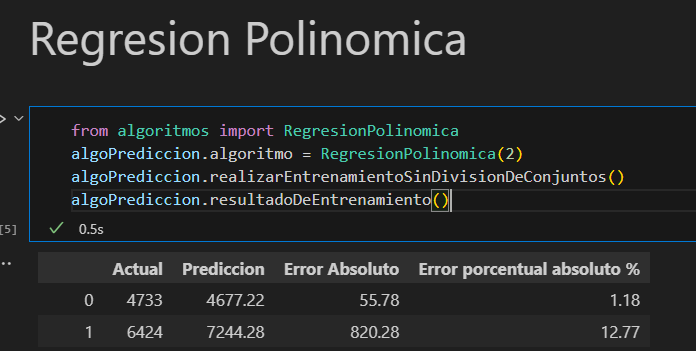
La eleccion del grado no fue trivial sino que  se utilizo un metodo estadistico de validacion cruzada para determinar con que grado el modelo deberia trabajar.  En este Tp se escogio por el de Shuffle Splits . Sin embargo el tiempo de respuesta fue exageradamente lento

No se con exactitud en que momento se genero el resultado para el MSE de grado 3 pero lo que si estoy seguro es que en 34minutos no fue capaz de generarme todos los resultados. Esto se debe a que se esta trabajando con un conjunto grande de datos  que contiene mas de 50.000 filas



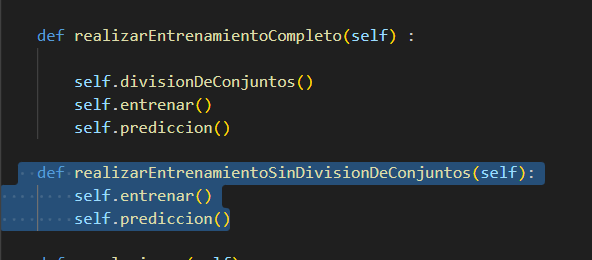
Sin embargo con con los resultados mostrados ya nos basta. Podemos ver que obtenemos el minimo con un grado igual a 2. Con un grado 3 igual a el MSE comienza aumentar demasiado .  Asi que se opta por el grado 2 para la regresion Polinomica porque ademas es el que mejor resultado nos dio.

Volvamos a lo importante



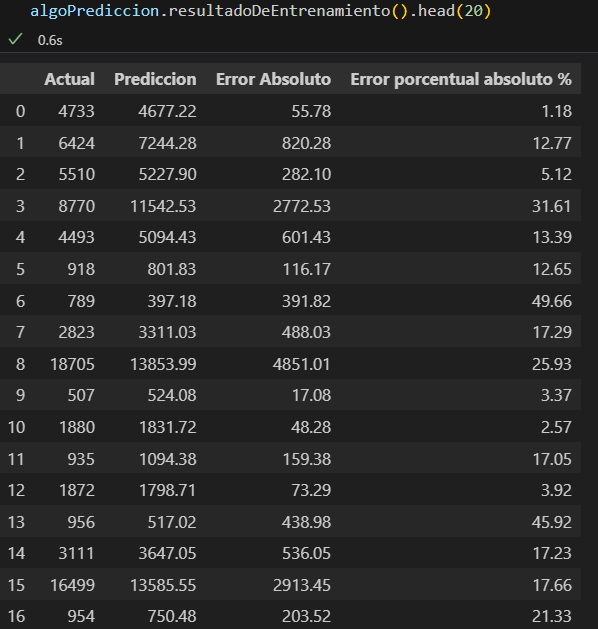
Primero se asigna el modelo de regresion Polinomica de grado 2

Luego se ejecuta el metodo realizarEntrenamientoSinDivisionDeConjuntos()



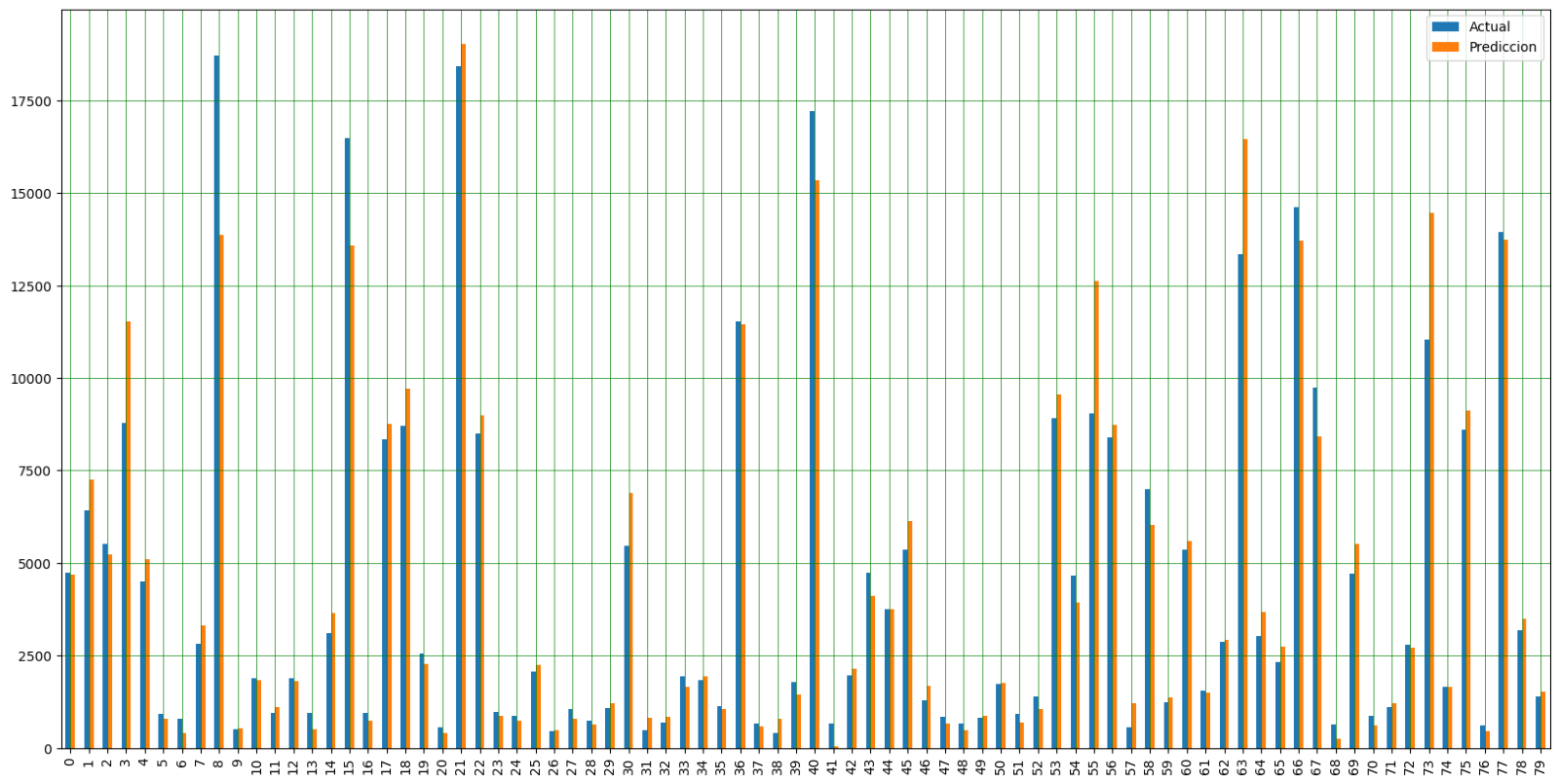
Como se puede ver es muy similar al realizarEntrenamientoCompleto() que ya se explico cuando se termino de realizar la eliminacion hacia atrás. La  diferencia es que este metodo no hace division de conjuntos porque ya fue realizado previamente !! No hace falta hacerla devuelta. Sin embargo como cambiamos de algoritmos obviamente tenemos que ejecutar los metodos entrenar y prediccion devuelta para asi obtener los resultados del nuevo algoritmo que se va a utilizar.

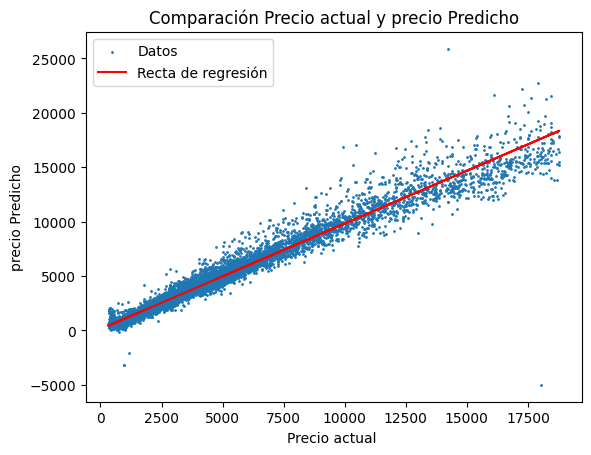
Veamos los resultados



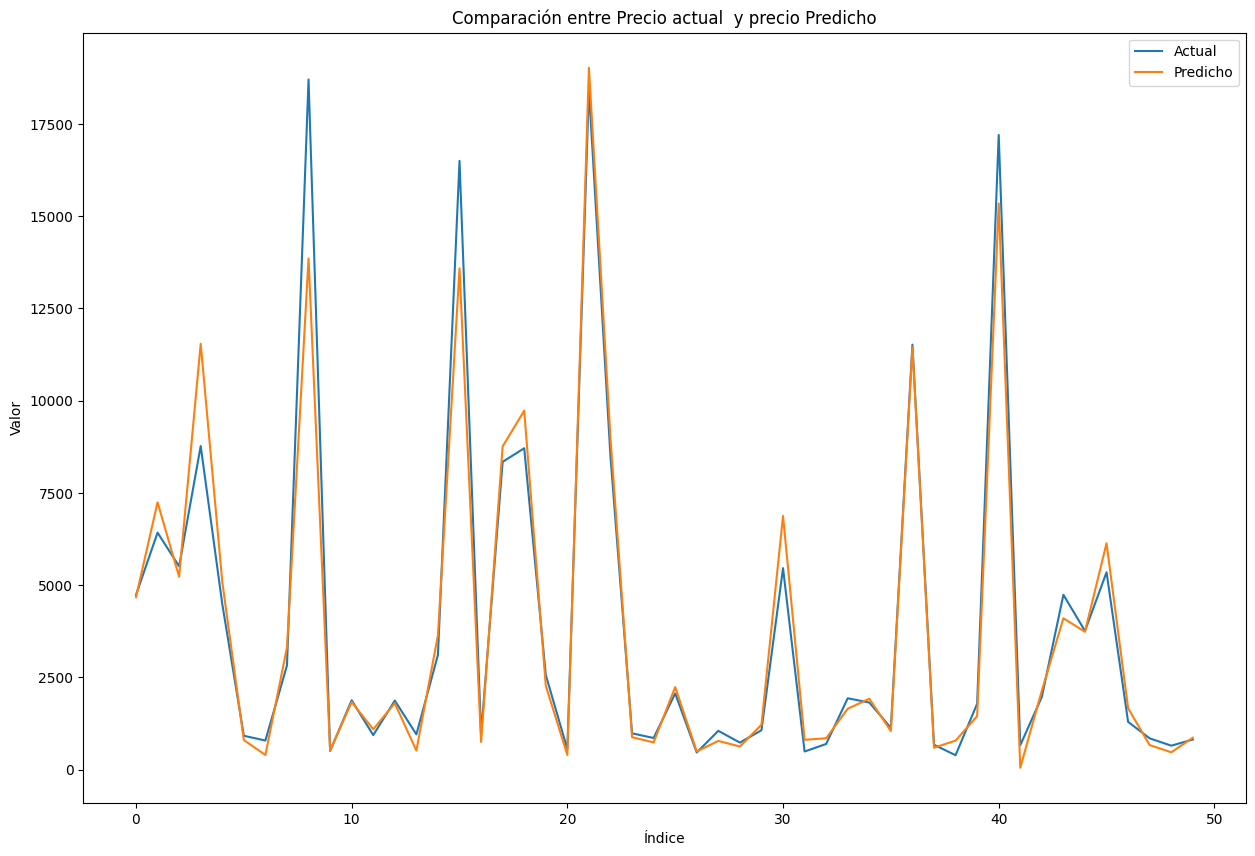


Observemos los resultados graficos

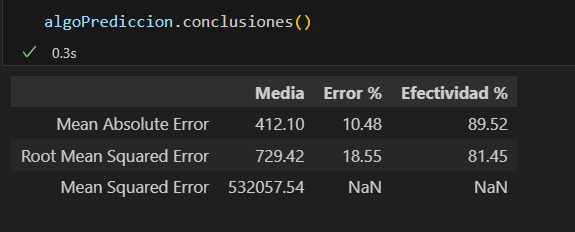




Vemos aca que a diferencia del modelo lineal la dispercion aquí es menos “violenta”. Incluso con aquellos precios actuales y predichos mas grandes. Esto ya nos da indicios de que efectivamente se obtuvieron mejores resultados y que la conclusion deberia ser notablemente mejor



Por ultimo veamos la conclusion



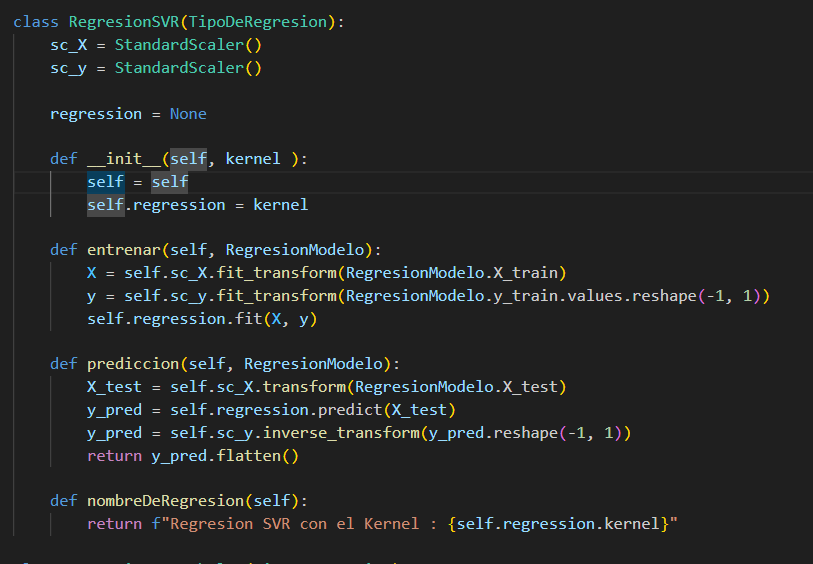
Como vemos aca los resultados son muchisimo mejores!!! Por supuesto la diferencia es notable , incluso con el RMSE el resultado es bueno!

Pasemos al siguiente modelo de regresion :

**REGRESION CON MAQUINAS DE SOPORTE VECTORIAL**

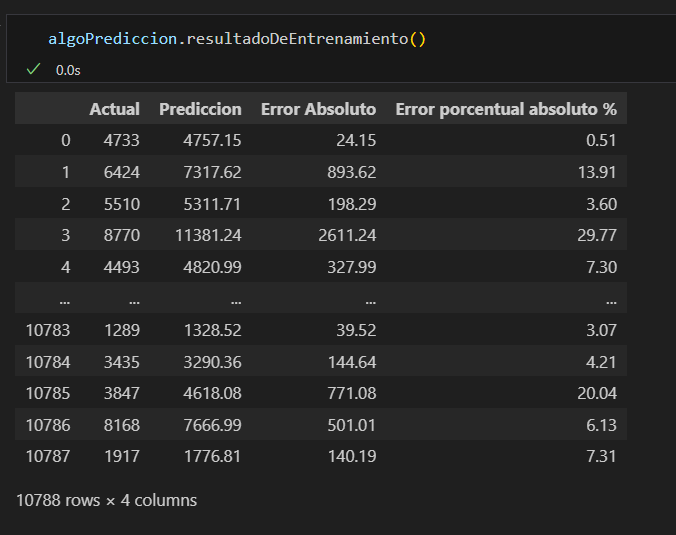


Comenzamos con el kernel rbf

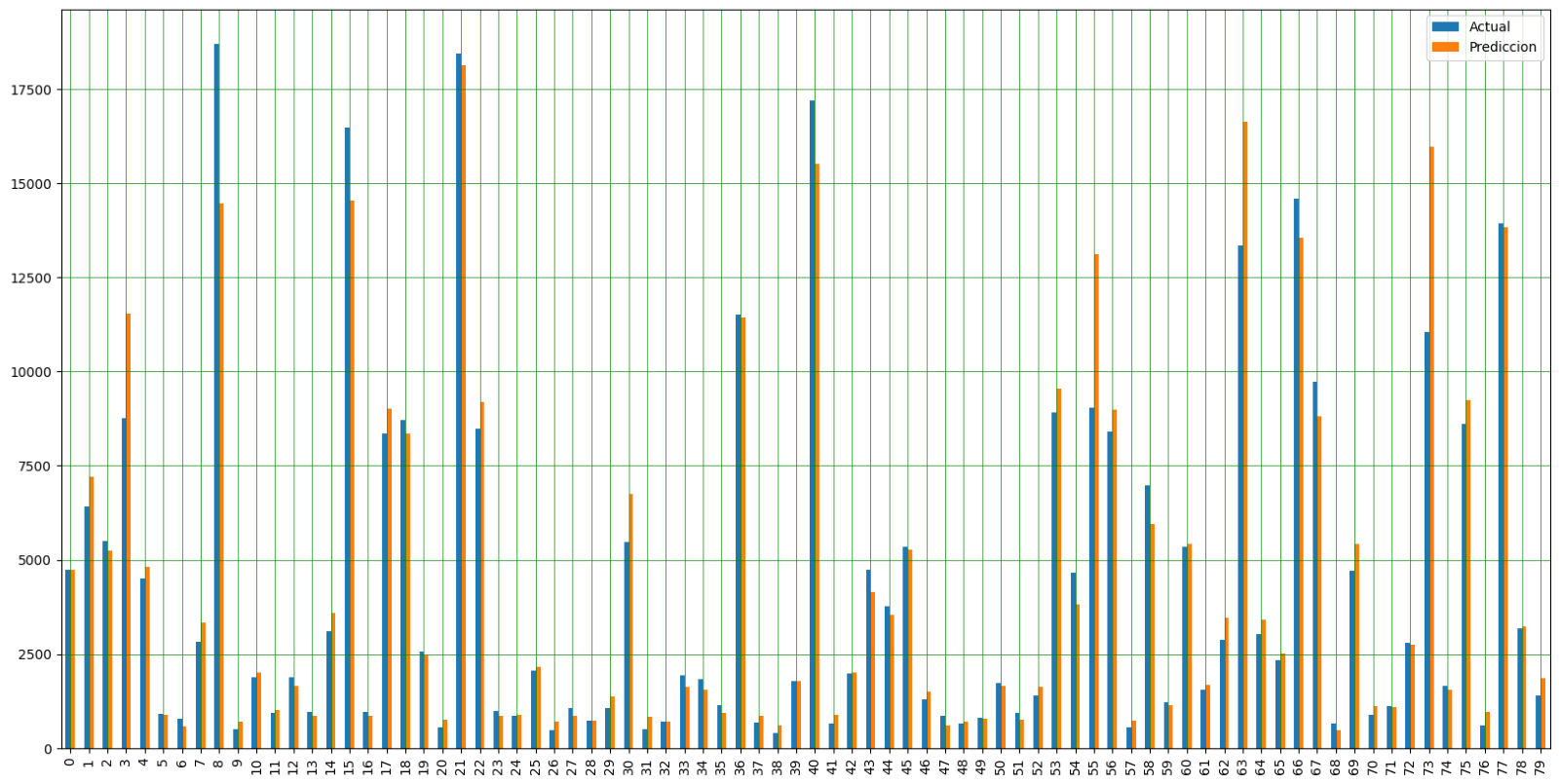
Ç

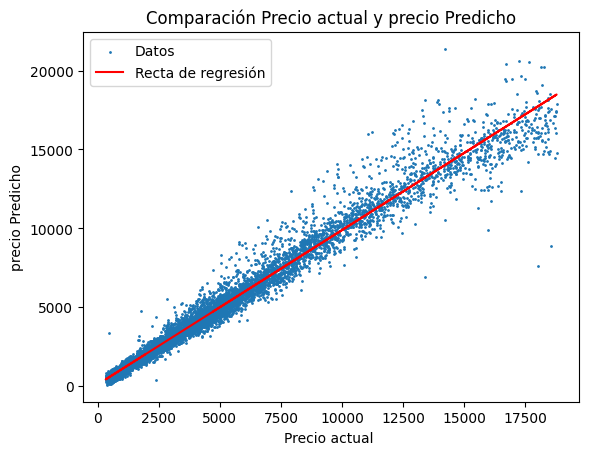
El strategy  de RegresionSVR es valido para los 3 tipos de kernel.

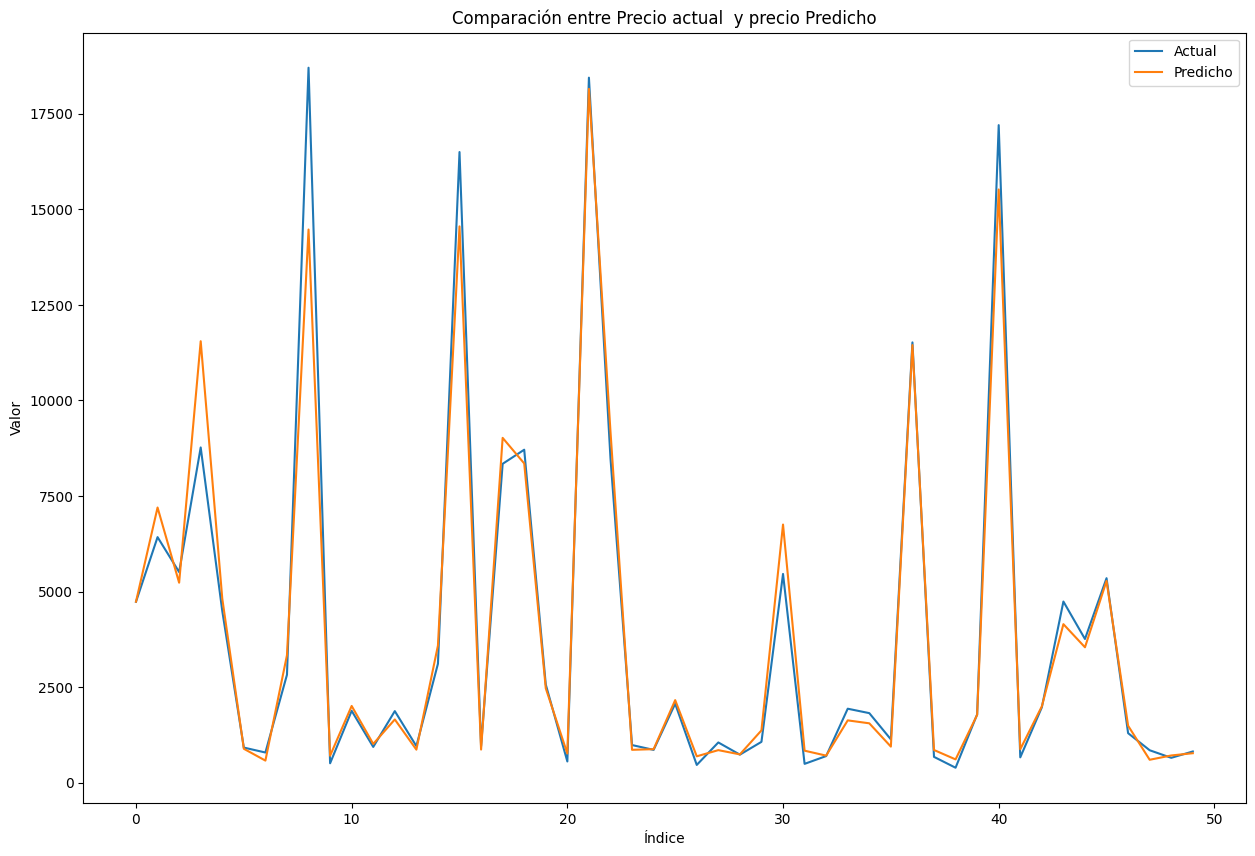
Veamos los resultados :

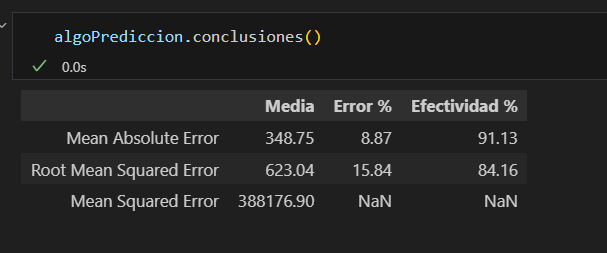


Veamos los graficos









## *SVR : Kernel : Poly*

algoPrediccion.algoritmo = RegresionSVR(svr\_poly)   
algoPrediccion.realizarEntrenamientoSinDivisionDeConjuntos()  
algoPrediccion.resultadoDeEntrenamiento()

Actual Prediccion Error Absoluto Error porcentual absoluto %  
0 4733 4757.15 24.15 0.51  
1 6424 7317.62 893.62 13.91  
2 5510 5311.71 198.29 3.60  
3 8770 11381.24 2611.24 29.77  
4 4493 4820.99 327.99 7.30  
... ... ... ... ...  
10783 1289 1328.52 39.52 3.07  
10784 3435 3290.36 144.64 4.21  
10785 3847 4618.08 771.08 20.04  
10786 8168 7666.99 501.01 6.13  
10787 1917 1776.81 140.19 7.31  
  
[10788 rows x 4 columns]

algoPrediccion.resultadoDeEntrenamiento().describe()

Actual Prediccion Error Absoluto Error porcentual absoluto %  
count 10788.00 10788.00 10788.00 10788.00  
mean 3929.21 3950.49 350.25 13.58  
std 3981.60 3969.61 528.82 15.10  
min 326.00 -647.20 0.00 0.00  
25% 947.50 955.45 92.95 3.82  
50% 2398.00 2405.16 206.03 8.64  
75% 5311.25 5417.35 381.79 17.36  
max 18787.00 25370.72 14034.20 162.61

algoPrediccion.todasLasComparacionesDeActualPrediccion()

Gráfico, Gráfico de barras, Histograma

Descripción generada automáticamente

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Gráfico, Histograma

Descripción generada automáticamente

algoPrediccion.conclusiones()

Media Error % Efectividad %  
Mean Absolute Error 350.25 8.91 91.09  
Root Mean Squared Error 634.27 16.13 83.87  
Mean Squared Error 402293.89 NaN NaN

# *SVR :Kernel Linear*

from sklearn.svm import SVR  
from algoritmos import RegresionSVR  
  
  
algoPrediccion.algoritmo = RegresionSVR(svr\_lin)   
algoPrediccion.realizarEntrenamientoSinDivisionDeConjuntos()  
algoPrediccion.resultadoDeEntrenamiento()

Actual Prediccion Error Absoluto Error porcentual absoluto %  
0 4733 4944.17 211.17 4.46  
1 6424 7410.62 986.62 15.36  
2 5510 6013.92 503.92 9.15  
3 8770 9851.66 1081.66 12.33  
4 4493 4795.63 302.63 6.74  
... ... ... ... ...  
10783 1289 1296.92 7.92 0.61  
10784 3435 3525.56 90.56 2.64  
10785 3847 5062.60 1215.60 31.60  
10786 8168 6259.81 1908.19 23.36  
10787 1917 1903.30 13.70 0.71  
  
[10788 rows x 4 columns]

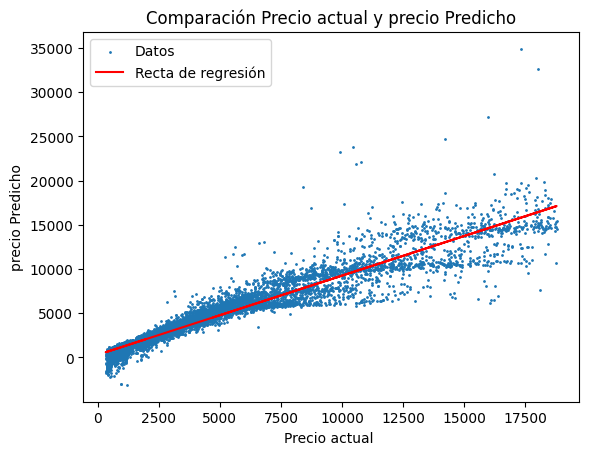
algoPrediccion.resultadoDeEntrenamiento().describe()

Actual Prediccion Error Absoluto Error porcentual absoluto %  
count 10788.00 10788.00 10788.00 10788.00  
mean 3929.21 3821.98 658.69 28.10  
std 3981.60 3725.19 966.50 43.67  
min 326.00 -3100.12 0.04 0.00  
25% 947.50 1066.60 186.57 7.15  
50% 2398.00 2580.35 365.89 15.83  
75% 5311.25 5620.78 694.20 31.91  
max 18787.00 34884.32 17555.32 591.26

algoPrediccion.todasLasComparacionesDeActualPrediccion()

Gráfico, Histograma

Descripción generada automáticamente



Gráfico, Histograma

Descripción generada automáticamente

algoPrediccion.conclusiones()

Media Error % Efectividad %  
Mean Absolute Error 658.69 16.75 83.25  
Root Mean Squared Error 1169.58 29.74 70.26  
Mean Squared Error 1367909.55 NaN NaN

**REGRESION CON ARBOLES DE DESICION**

import pickle  
  
# Cargar la variable desde el archivo  
with open('variable.pkl', 'rb') as f:  
 algoPrediccion = pickle.load(f)  
from algoritmos import RegresionConArboles  
from sklearn.ensemble import RandomForestRegressor

# Regresión con Árboles de Decisión

from algoritmos import RegresionConArboles  
from sklearn.tree import DecisionTreeRegressor  
algoPrediccion.algoritmo = RegresionConArboles(DecisionTreeRegressor(random\_state = 0))  
algoPrediccion.realizarEntrenamientoSinDivisionDeConjuntos()  
algoPrediccion.resultadoDeEntrenamiento().head(30)

Actual Prediccion Error Absoluto Error porcentual absoluto %  
0 4733 4620.00 113.00 2.39  
1 6424 7036.00 612.00 9.53  
2 5510 4404.00 1106.00 20.07  
3 8770 11688.00 2918.00 33.27  
4 4493 4678.00 185.00 4.12  
5 918 918.00 0.00 0.00  
6 789 764.00 25.00 3.17  
7 2823 3011.00 188.00 6.66  
8 18705 13267.00 5438.00 29.07  
9 507 513.00 6.00 1.18  
10 1880 1628.00 252.00 13.40  
11 935 848.00 87.00 9.30  
12 1872 1767.00 105.00 5.61  
13 956 956.00 0.00 0.00  
14 3111 3959.00 848.00 27.26  
15 16499 12617.00 3882.00 23.53  
16 954 918.00 36.00 3.77  
17 8342 10232.00 1890.00 22.66  
18 8711 7854.00 857.00 9.84  
19 2567 2310.00 257.00 10.01  
20 555 501.00 54.00 9.73  
21 18445 15941.00 2504.00 13.58  
22 8486 9478.00 992.00 11.69  
23 984 767.00 217.00 22.05  
24 858 844.00 14.00 1.63  
25 2066 2510.00 444.00 21.49  
26 464 500.00 36.00 7.76  
27 1052 946.00 106.00 10.08  
28 733 701.00 32.00 4.37  
29 1068 1089.00 21.00 1.97  
30 5461 6250.00 789.00 14.45

Esta clase es reutilizada tanto para para la Regresión con arboles de decisión como

Para el Random Forest . Acá lo que importa es el tipo de regresor que se le declare a este objeto al momento de asignar el modelo de regresión al objeto RegresiónModelo . Por Ejemplo

algoPrediccion.algoritmo =

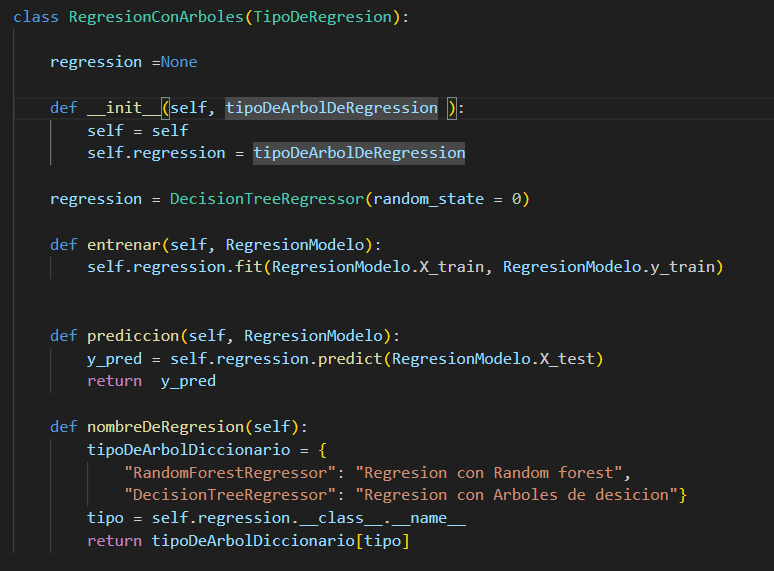
RegresionConArboles(***DecisionTreeRegressor(random\_state = 0))***

Para este caso va el objeto RegresionConArboles va a realizar el entrenamiento y predicciones según el modelo de ***regresión con arboles de decisión***

algoPrediccion.algoritmo =

RegresionConArboles(RandomForestRegressor(n\_estimators = 300, random\_state = 0))

Para este caso va el objeto RegresionConArboles va a realizar el entrenamiento y predicciones según el modelo de ***regresión forest regresor***



algoPrediccion.resultadoDeEntrenamiento().describe()

Actual Prediccion Error Absoluto Error porcentual absoluto %  
count 10788.00 10788.00 10788.00 10788.00  
mean 3929.21 3933.01 363.37 9.55  
std 3981.60 3985.64 657.12 9.98  
min 326.00 338.00 0.00 0.00  
25% 947.50 945.00 48.00 2.08  
50% 2398.00 2398.00 146.00 6.46  
75% 5311.25 5321.75 376.00 13.96  
max 18787.00 18823.00 10424.00 135.25

algoPrediccion.todasLasComparacionesDeActualPrediccion()

Gráfico, Histograma

Descripción generada automáticamente

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Gráfico, Histograma

Descripción generada automáticamente

algoPrediccion.conclusiones()

Media Error % Efectividad %  
Mean Absolute Error 363.37 9.24 90.76  
Root Mean Squared Error 750.87 19.09 80.91  
Mean Squared Error 563800.02 NaN NaN

***Regresion con RANDOM FOREST***

import pickle  
  
# Cargar la variable desde el archivo  
with open('variable.pkl', 'rb') as f:  
 algoPrediccion = pickle.load(f)  
from algoritmos import RegresionConArboles  
from sklearn.ensemble import RandomForestRegressor

# Regresión con ForestRegressor

algoPrediccion.algoritmo = RegresionConArboles(RandomForestRegressor(n\_estimators = 300, random\_state = 0))  
algoPrediccion.realizarEntrenamientoSinDivisionDeConjuntos()  
algoPrediccion.resultadoDeEntrenamiento()

Actual Prediccion Error Absoluto Error porcentual absoluto %  
0 4733 4669.43 63.57 1.34  
1 6424 7053.27 629.27 9.80  
2 5510 5223.09 286.91 5.21  
3 8770 11088.87 2318.87 26.44  
4 4493 4709.23 216.23 4.81  
... ... ... ... ...  
10783 1289 1330.44 41.44 3.21  
10784 3435 3346.93 88.07 2.56  
10785 3847 4282.07 435.07 11.31  
10786 8168 8500.22 332.22 4.07  
10787 1917 2129.50 212.50 11.09  
  
[10788 rows x 4 columns]

algoPrediccion.resultadoDeEntrenamiento().describe()

Actual Prediccion Error Absoluto Error porcentual absoluto %  
count 10788.00 10788.00 10788.00 10788.00  
mean 3929.21 3933.68 278.89 7.48  
std 3981.60 3946.96 474.88 7.17  
min 326.00 360.38 0.01 0.00  
25% 947.50 931.07 44.76 2.45  
50% 2398.00 2433.13 111.00 5.38  
75% 5311.25 5305.31 301.33 10.35  
max 18787.00 17878.26 5683.08 93.71

algoPrediccion.todasLasComparacionesDeActualPrediccion()

Gráfico, Histograma

Descripción generada automáticamente

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Gráfico, Histograma

Descripción generada automáticamente

algoPrediccion.conclusiones()

Media Error % Efectividad %  
Mean Absolute Error 278.89 7.09 92.91  
Root Mean Squared Error 550.70 14.00 86.00  
Mean Squared Error 303267.36 NaN NaN

# Regresion con K Nearest Neighbors (KNN)

from algoritmos import RegresionKNN  
algoPrediccion.algoritmo = RegresionKNN()  
algoPrediccion.realizarEntrenamientoSinDivisionDeConjuntos()  
algoPrediccion.resultadoDeEntrenamiento()

Veamos la clase

class DataKNN() :

    K = None

    rmseValor = None

    y\_pred = None

    def \_\_init\_\_(self, K,rmseValor,y\_pred ):

        self = self

        self.K = K

        self.rmseValor = rmseValor

        self.y\_pred=y\_pred

class RegresionKNN(TipoDeRegresion):

    listaDataKNN = []

    def entrenar(self, RegresionModelo):

        for K in range(20):

            K = K+1

            model = KNeighborsRegressor(n\_neighbors = K)

            model.fit(RegresionModelo.X\_train, RegresionModelo.y\_train) # fit

            y\_pred=model.predict(RegresionModelo.X\_test).flatten() # hacer predicciones en el conjunto de prueba

            rmseValor = sqrt(mean\_squared\_error(RegresionModelo.y\_test,y\_pred)) # calcular rmse

            self.listaDataKNN.append(DataKNN(K,rmseValor,y\_pred))

            print(f'Valor RMSE para k = ' ,K , 'es:', rmseValor)

    def dataKNNDefinitivo(self) -> DataKNN  :

        dataKNNDefinitivo : DataKNN= min(self.listaDataKNN, key=lambda x: x.rmseValor)

        print("El K seleccionado por tener el valor RMSE minimo es : ",dataKNNDefinitivo.K,"con un RMSE De : ",dataKNNDefinitivo.rmseValor)

        return dataKNNDefinitivo

    def prediccion(self, RegresionModelo):

        return  self.dataKNNDefinitivo().y\_pred

    def nombreDeRegresion(self):

        return "Regresion con KNN"

Valor RMSE para k = 1 es: 1286.0419825591655  
Valor RMSE para k = 2 es: 1167.0826103902598  
Valor RMSE para k = 3 es: 1155.4326316748816  
Valor RMSE para k = 4 es: 1147.0874627788073  
Valor RMSE para k = 5 es: 1152.3116419213713  
Valor RMSE para k = 6 es: 1160.9909466845484  
Valor RMSE para k = 7 es: 1168.1302649464492  
Valor RMSE para k = 8 es: 1167.5557946209278  
Valor RMSE para k = 9 es: 1174.934946581827  
Valor RMSE para k = 10 es: 1180.236738093619  
Valor RMSE para k = 11 es: 1187.3693319551353  
Valor RMSE para k = 12 es: 1191.8026718211581  
Valor RMSE para k = 13 es: 1195.4980031228713  
Valor RMSE para k = 14 es: 1199.362188507188  
Valor RMSE para k = 15 es: 1207.727587318237  
Valor RMSE para k = 16 es: 1213.4885378848796  
Valor RMSE para k = 17 es: 1221.6283101442814  
Valor RMSE para k = 18 es: 1226.960637761948  
Valor RMSE para k = 19 es: 1231.2493441037193  
Valor RMSE para k = 20 es: 1236.0697164570095  
El K seleccionado por tener el valor RMSE minimo es : 4 con un RMSE De : 1147.0874627788073

Actual Prediccion Error Absoluto Error porcentual absoluto %  
0 4733 4645.00 88.00 1.86  
1 6424 6785.50 361.50 5.63  
2 5510 4805.25 704.75 12.79  
3 8770 10792.50 2022.50 23.06  
4 4493 5926.00 1433.00 31.89  
... ... ... ... ...  
10783 1289 843.75 445.25 34.54  
10784 3435 3619.25 184.25 5.36  
10785 3847 4993.50 1146.50 29.80  
10786 8168 8784.75 616.75 7.55  
10787 1917 1001.50 915.50 47.76  
  
[10788 rows x 4 columns]

algoPrediccion.resultadoDeEntrenamiento().describe()

Actual Prediccion Error Absoluto Error porcentual absoluto %  
count 10788.00 10788.00 10788.00 10788.00  
mean 3929.21 3809.64 614.23 17.58  
std 3981.60 3598.34 968.82 20.90  
min 326.00 393.00 0.00 0.00  
25% 947.50 960.50 92.00 5.30  
50% 2398.00 2549.62 249.75 12.00  
75% 5311.25 5221.50 717.56 22.18  
max 18787.00 18083.00 14870.00 638.82

algoPrediccion.todasLasComparacionesDeActualPrediccion()

Gráfico, Histograma

Descripción generada automáticamente

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Gráfico, Histograma

Descripción generada automáticamente

algoPrediccion.conclusiones()

Media Error % Efectividad %

Mean Absolute Error 614.23 15.62 84.38  
Root Mean Squared Error 1147.09 29.17 70.83  
Mean Squared Error 1315809.65 NaN NaN

Finalmente se ha mostrado los resultado de todos los algoritmos de regresión considerados en este tp. Pero antes de mostrar los gráficos comparativos de cada uno y ver cual es el algoritmo que tuvo mejores resultados veamos lo siguiente

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

Todos los modelos de regresión terminaron sus pruebas ejecutando este método llamado conclusiones que como vimos es propio del objeto RegresionModelo . vemos que cada vez que se ejecuta este método todos los diccionarios que se encuentran adentro del rectángulo reciben una key y un value. La key es determinada por el algoritmo de regresión (strategy) que le corresponda. Por ejemplo el caso mas sencillo es el de regresión lineal

class RegresionPolinomica(TipoDeRegresion) :

    def nombreDeRegresion(self):

        return "Regresion Polinomica"

y el caso mas complicado es

class RegresionConArboles(TipoDeRegresion):

    regression =None

    def nombreDeRegresion(self):

        tipoDeArbolDiccionario = {

            "RandomForestRegressor": "Regresion con Random forest",

            "DecisionTreeRegressor": "Regresion con Arboles de desicion"}

        tipo = self.regression.\_\_class\_\_.\_\_name\_\_

        return tipoDeArbolDiccionario[tipo]

retomando a la imagen

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

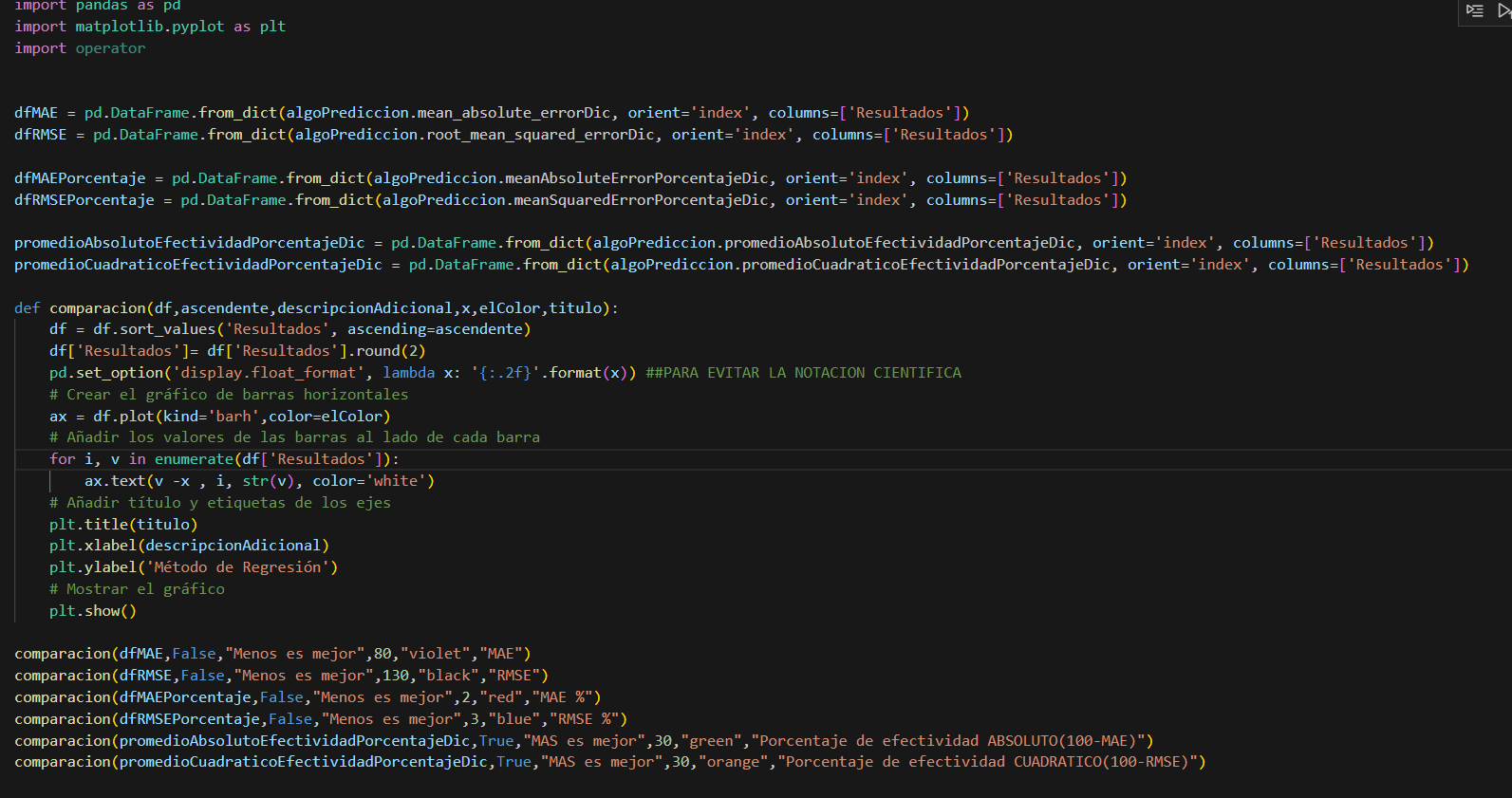
El objetivo es que cada vez que el objeto RegresionModelo ejecute el método conclusión() se guarde los datos mas importante que indican que tan efectivo fue el rendimiento del algoritmo seleccionado. Por supuesto la key representa el nombre del algoritmos en cuestión y las values los resultados que se obtuvieron. Osea en simples palabras se podría decir que se esta guardando los valores de la tabla conclusión que retorna justamente este metodo que se muestra al final de la prueba de cada algoritmo. Esta es una forma automatizada de hacerlo .Por supuesto se podría haber anotado los resultados en algún lado y luego armar el dataframe final que permite comparar que algoritmo fue mejor.

RESULTADOS FINALES

# Cargar la variable desde el archivo  
with open('variable.pkl', 'rb') as f:  
 algoPrediccion = pickle.load(f)

algoPrediccion.mean\_absolute\_errorDic

{'Regresion Lineal': 737.779013718947,  
 'Regresion Polinomica': 412.10183615113436,  
 'Regresion SVR con el Kernel : rbf': 348.7512835754309,  
 'Regresion SVR con el Kernel : poly': 407.7371556739417,  
 'Regresion SVR con el Kernel : linear': 658.6852907546541,  
 'Regresion con Arboles de desicion': 363.36545235446795,  
 'Regresion con KNN': 614.2302326659251,  
 'Regresion con Random forest': 278.8850031642791}



Gráfico, Gráfico de barras

Descripción generada automáticamente

Gráfico

Descripción generada automáticamente

Gráfico, Gráfico de barras

Descripción generada automáticamente

Gráfico, Gráfico de barras

Descripción generada automáticamente

Gráfico, Gráfico de barras

Descripción generada automáticamente

Gráfico, Gráfico de barras

Descripción generada automáticamente

Definitivamente, el algoritmo que mejor rendimiento mostró en nuestro estudio fue el Random Forest. Si tuviéramos que predecir el precio de un nuevo diamante, sin duda utilizaríamos el Random Forest como nuestra primera opción. Este algoritmo demostró ser no solo rápido, sino también capaz de darnos las mejores predicciones en comparación con otros algoritmos evaluados. Definitivamente no utilizaríamos los modelos de Regresión Lineal Y KNN que si bien no dieron resultados tan malos aun así serian las ultimas opciones a considerar