

Tarea 4 Tema 2: Random Forest

Juan Manuel García Moyano
IABD
Informática y comunicaciones

Índice

| | |
|---|----|
| 1. Importación de librerías necesarias..... | 3 |
| 2. Preproceso..... | 3 |
| 2.1 Importación de los datos del dataset..... | 3 |
| 2.2 Mostrar las primeras y últimas filas del dataframe importado..... | 4 |
| 2.3 Mostrar parámetros estadísticos de los datos (media, desviación típica, cuartiles, etc.)..... | 5 |
| 2.4 Mostrar un mapa de calor que indique la correlación entre variables..... | 5 |
| 2.5 Transformación de los datos..... | 7 |
| 2.5 Separar datos entre datos de entrada y etiquetas (resultados)..... | 8 |
| 2.6 Separar datos entre entrenamiento y prueba (usando un 80% para entrenamiento y 20% para test)..... | 8 |
| 3. Entrenamiento y predicción..... | 9 |
| 3.1 Elegir, instanciar y entrenar el modelo..... | 9 |
| 3.2 Realizar una predicción con los datos de prueba..... | 9 |
| 4. Evaluación..... | 9 |
| 4.1 Mostrar el error cuadrático medio (mean_squared_error)..... | 9 |
| 4.2 Mostrar el error absoluto medio (mean_absolute_error)..... | 10 |
| 4.3 Representar gráficamente los valores predichos con los valores reales..... | 10 |
| 4.4 Distribución del Error (Histograma de residuales)..... | 11 |
| 4.5 Curva de predicciones acumuladas (PDP)..... | 13 |
| 4.6 Importancia de las características..... | 14 |
| 5. Optimización de hiperparámetros..... | 14 |
| 5.1 Cada ejecución anterior se deberá hacer usando validación cruzada (por ejemplo n_splits = 4). Con ello obtendremos una medida de bondad del modelo (accuracy_score o mean_absolute_error), como lo ejecutaremos 4 veces, calcularemos la media de esas 4 ejecuciones..... | 15 |
| 5.2 Finalmente los parámetros elegidos serán los que den mejor media de esas medidas anteriormente nombradas..... | 16 |
| 5.3 Una vez obtenidos esos parámetros óptimos los aplicaremos al problema en cuestión y mostraremos los resultados..... | 16 |

1. Importación de librerías necesarias

He importado las librerías necesarias con el fin de entrenar un modelo con arboles de decisión. Además, todas las funcionalidades para realizar todos los apartados que no están directamente relacionado con el entrenamiento.

```
1 from google.colab import drive
2 from sklearn import preprocessing
3 from sklearn.ensemble import RandomForestRegressor
4 from sklearn.metrics import mean_absolute_error, mean_squared_error, make_scorer
5 from sklearn.model_selection import TimeSeriesSplit, train_test_split,
  cross_val_score, KFold
6 from sklearn.inspection import PartialDependenceDisplay
7
8 import matplotlib.pyplot as plt
9 import pandas as pd
10 import numpy as np
11 import seaborn as sns
```

2. Preproceso

2.1 Importación de los datos del dataset

La importación de los datos lo he realizado desde el drive. Para ello subí el dataset, luego monté mi drive en colab y leí los csv. Uní ambos csv a partir del atributo dteday que tienes en común.

```
1 drive.mount("/content/drive")
2
3 df_hour = pd.read_csv("/content/drive/My Drive/IABD/SAA/datasets/Tema 2/hour.
  csv")
4 df_day = pd.read_csv("/content/drive/My Drive/IABD/SAA/datasets/Tema 2/day.csv")
```

```
1 df_bike = df_hour.merge(df_day, on="dteday", suffixes=("_hour", "_day"))
```

2.2 Mostrar las primeras y últimas filas del dataframe importado

En colab, si ponemos en la última línea una variable muestra las primeras y últimas filas pero si se hace desde un entorno diferente el cual no sea jupyter, deberíamos utilizar los métodos head y tail.

```
1 df_bike
2 # Colab muestra las primeras y últimas filas, si se hace desde un entorno de
  ejecución diferente que no se jupyter se haría de la siguiente forma:
3 # df_bike.head()
4 # df_bike.tail()
```

| | instant_hour | dteday | season_hour | yr_hour | mnth_hour | hr | holiday_hour | weekday_hour | v |
|-------|--------------|------------|-------------|---------|-----------|-----|--------------|--------------|-----|
| 0 | 1 | 2011-01-01 | 1 | 0 | 1 | 0 | 0 | 6 | |
| 1 | 2 | 2011-01-01 | 1 | 0 | 1 | 1 | 0 | 6 | |
| 2 | 3 | 2011-01-01 | 1 | 0 | 1 | 2 | 0 | 6 | |
| 3 | 4 | 2011-01-01 | 1 | 0 | 1 | 3 | 0 | 6 | |
| 4 | 5 | 2011-01-01 | 1 | 0 | 1 | 4 | 0 | 6 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 17374 | 17375 | 2012-12-31 | 1 | 1 | 12 | 19 | 0 | 1 | |
| 17375 | 17376 | 2012-12-31 | 1 | 1 | 12 | 20 | 0 | 1 | |
| 17376 | 17377 | 2012-12-31 | 1 | 1 | 12 | 21 | 0 | 1 | |
| 17377 | 17378 | 2012-12-31 | 1 | 1 | 12 | 22 | 0 | 1 | |
| 17378 | 17379 | 2012-12-31 | 1 | 1 | 12 | 23 | 0 | 1 | |

2.3 Mostrar parámetros estadísticos de los datos (media, desviación típica, cuartiles, etc.)

Un dataframe tiene un método llamado `describe` que nos muestra los datos estadísticos de todos los atributos. Se debe de tener en cuenta que los datos estén en formato numérico.

En esta sección defino dos funciones para tratar con los outliers. Una de las funciones se utilizará más adelante.

```
1 df_bike.describe()
```

| | instant_hour | season_hour | yr_hour | mnth_hour | hr | holiday_hour | weekdi |
|-------|--------------|--------------|--------------|--------------|--------------|--------------|--------|
| count | 17379.0000 | 17379.000000 | 17379.000000 | 17379.000000 | 17379.000000 | 17379.000000 | 17379 |
| mean | 8690.0000 | 2.501640 | 0.502561 | 6.537775 | 11.546752 | 0.028770 | 3 |
| std | 5017.0295 | 1.106918 | 0.500008 | 3.438776 | 6.914405 | 0.167165 | 2 |
| min | 1.0000 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0 |
| 25% | 4345.5000 | 2.000000 | 0.000000 | 4.000000 | 6.000000 | 0.000000 | 1 |
| 50% | 8690.0000 | 3.000000 | 1.000000 | 7.000000 | 12.000000 | 0.000000 | 3 |
| 75% | 13034.5000 | 3.000000 | 1.000000 | 10.000000 | 18.000000 | 0.000000 | 5 |
| max | 17379.0000 | 4.000000 | 1.000000 | 12.000000 | 23.000000 | 1.000000 | 6 |

8 rows × 31 columns

2.4 Mostrar un mapa de calor que indique la correlación entre variables

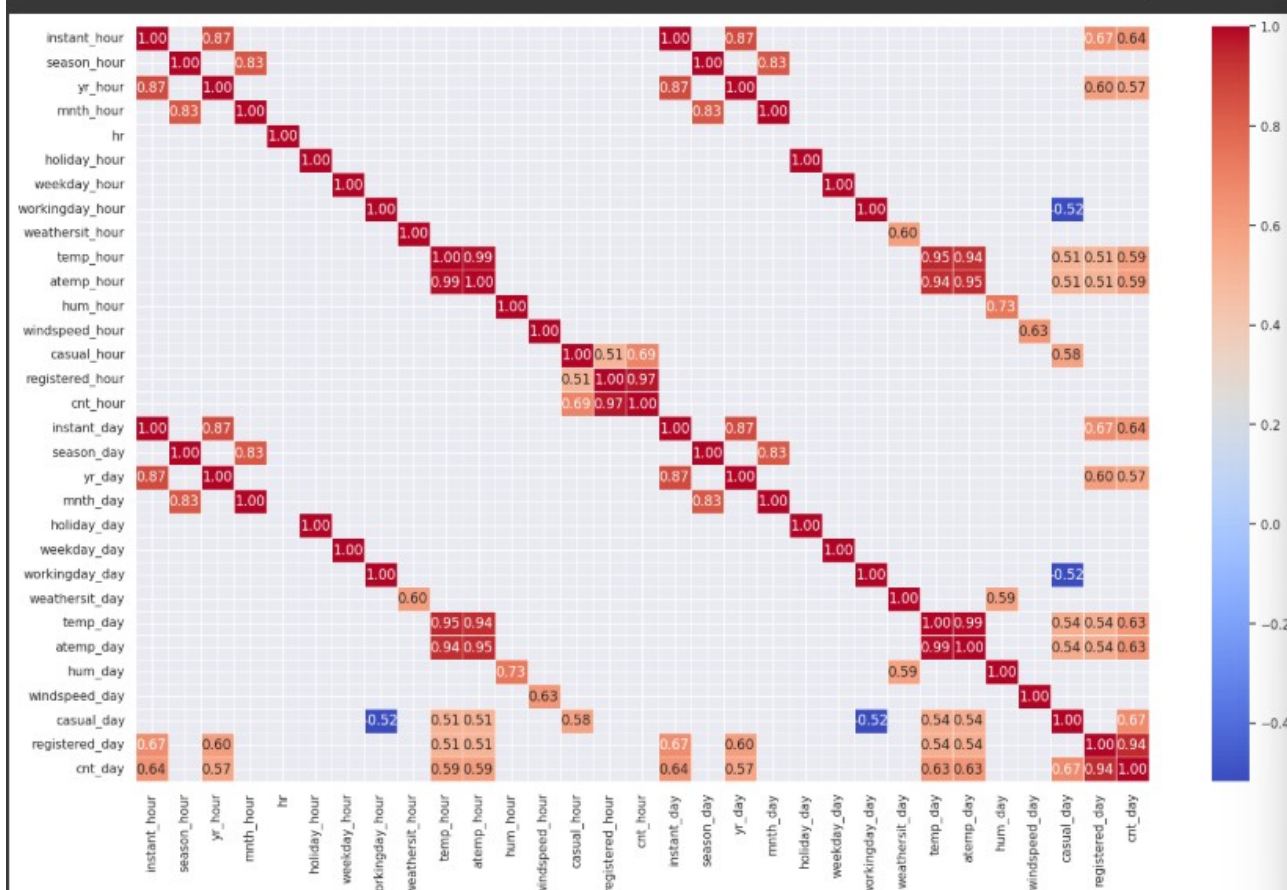
Para realizar el mapa de calor he utilizado la librería `seaborn`. Para ello tuve de indicar la dimensión del mapa porque en un principio no se veía muy bien, ya que, los nombres de los atributos se juntaban. Por ello, primero lo redimensiono y después lo muestro con el método `headmap(dataframe.corr(), square = True, annot = True)`. Vamos a desglosar los parámetros:

- `dataframe.corr()`: calcula la matriz de correlación entre las columnas numéricas del dataframe. La correlación mide la relación lineal entre dos variables, con valores que van desde:
 - 1: correlación positiva perfecta (ambas variables aumentas juntas).
 - -1: correlación negativa perfecta (una variable aumenta mientras la otra disminuye).

- 0: no hay relación. Estas nos indican que deben ser eliminadas para el entrenamiento.
- square = True: fuerza que cada celda mapa de calor tenga forma cuadrada.
- Annot = True: muestra los valores numéricos de la correlación dentro de cada celda.

Como en este dataset tuve que unir dos en uno, aparecen demasiadas características. Para solucionar este problema primero realicé el corr eliminando el atributo dteday. El resultado lo filtré por los valores absolutos fuesen mayor al 0.5 y mostré un nuevo mapa de calor.

```
1 filtered_corr = df_bike.drop("dteday", axis=1).corr()
2 filtered_corr[abs(filtered_corr) < 0.5] = np.nan
3 plt.figure(figsize=(20, 12))
4 sns.heatmap(filtered_corr, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5, mask=np.isnan(filtered_corr))
5 plt.show()
```



2.5 Transformación de los datos

Si nos fijamos en los tipos de datos que tiene el dataset (se han mostrado más arriba) el único atributo que es de tipo objeto es el `dteday` pero al fusionar ambos datasets ha desaparecido. Esto quiere decir que no existen atributos categóricos. Así nos saltamos el tener que transformarlas por el método One-Hot. Lo que sí debemos hacer es transformar o eliminar (en este caso transformar) los outliers y el escalado de los atributos.

```
1 def detectarLimitesOutlier(df: pd.DataFrame, k: int = 3):
2     Q1 = df.quantile(0.25)
3     Q3 = df.quantile(0.75)
4     IQR = Q3 - Q1
5     xL = Q1 - k * IQR # Límite inferior
6     xU = Q3 + k * IQR # Límite Superior
7     return (xL, xU)
8
9 def outliersCambiarMedia(df: pd.DataFrame, k = 3) -> pd.DataFrame:
10     df_numeric = df.select_dtypes(include=[np.number])
11
12     xL, xU = detectarLimitesOutlier(df_numeric, k)
13
14     outliers = (df_numeric < xL) | (df_numeric > xU)
15
16     df_sin_outliers = df.copy()
17
18     for columna in df_numeric.columns:
19         media = df_numeric[columna].mean()
20         if df_sin_outliers[columna].dtype == 'int64':
21             df_sin_outliers.loc[outliers[columna], columna] = int(media)
22         else:
23             df_sin_outliers.loc[outliers[columna], columna] = media
24
25     return df_sin_outliers

```

```
1 df_norm = outliersCambiarMedia(df_bike)

```

```
1 scaler = preprocessing.MinMaxScaler()
2 df_norm[df_norm.drop("dteday", axis=1).columns] = scaler.fit_transform(df_norm.
    drop("dteday", axis=1))

```

2.5 Separar datos entre datos de entrada y etiquetas (resultados)

El estudio de se va realizar con los atributos `registered_day`, `casual_day`, `atemp_day`, `temp_day`, `yr_day`, `instant_day`, `atemp_hour`, `temp_hour`, `yr_hour` y `instant_hour`. Para ello divido los datos en una variable (x) que va a tener los datos con los atributos anteriores y otra variable (y) con los datos del atributo objetivo. Más adelante veremos que la gran mayoría no los utiliza el modelo.

```
1 x_df_bike = df_norm[["registered_day", "casual_day", "atemp_day", "temp_day",  
2 "yr_day", "instant_day", "atemp_hour", "temp_hour", "yr_hour", "instant_hour"]]  
3 y_df_bike = df_norm['cnt_day']
```

2.6 Separar datos entre entrenamiento y prueba (usando un 80% para entrenamiento y 20% para test)

Para separar los datos utilizo la funcionalidad `train_test_split`. A este le paso dos variables una con los datos de la variable x y la otra variable es y. X tiene los datos sin los atributos anteriormente mencionados y la otra variable que es la que tiene todos los datos del atributo objetivo. Además, a esta función le paso los parámetros `random_state` (1) y `test_size` (0.20). Esto lo que hace que la división sea reproducible y representa la proporción del dataframe total asignada al conjunto de prueba, respectivamente.

Mediante el desempaqueado múltiple, lo que devuelve la función se lo asigno a 4 variables. Estas son:

- `x_entrenamiento`: tiene el 80% de los datos que van a servir para entrenar el modelo.
- `x_test`: tiene el otro 20% de los datos que van a servir hacer la predicción.
- `y_entrenamiento`: tiene el 80% de los datos que van a servir para entrenar el modelo.
- `y_test`: tiene el otro 20% de los datos que van a servir para calcular la probabilidad de acierto del modelo.

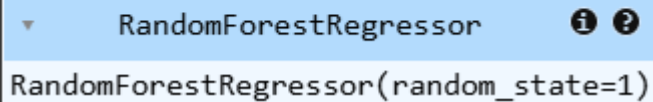
```
x_entrenamiento, x_test, y_entrenamiento, y_test = train_test_split(x_df_bike,  
y_df_bike, random_state=1, test_size=0.2)
```


3. Entrenamiento y predicción

3.1 Elegir, instanciar y entrenar el modelo

Primero instancio el modelo deseado (regresión). Después, lo entreno con las variables anteriores de entrenamiento.

```
1 modelo = RandomForestRegressor(n_estimators=100, random_state=1)
2 modelo.fit(x_entrenamiento, y_entrenamiento)
```



The screenshot shows a Jupyter Notebook interface. At the top, there is a dropdown menu with 'RandomForestRegressor' selected, followed by information and help icons. Below the dropdown, the code 'RandomForestRegressor(random_state=1)' is displayed, which is the result of instantiating the model.

3.2 Realizar una predicción con los datos de prueba

Al modelo ya entrenado hago que haga una predicción con los datos de `x_test`.

```
1 y_prediccion = modelo.predict(x_test)
```

4. Evaluación

Este ejercicio es un problema de regresión, por eso realizo estos apartados.

4.1 Mostrar el error cuadrático medio (mean_squared_error)

Con la ayuda de la función `mean_squared_error`, que me sirve para comprobar el error cuadrático medio que tiene el modelo. A esta función le paso los datos `y_test` (datos del atributo calidad) y la predicción realizada en el anterior apartado. Muestro la MSE.

```
1 print("El error cuadrático medio:", mean_squared_error(y_test, y_prediccion))
```

El error cuadrático medio: 1.325345699001311e-08

4.2 Mostrar el error absoluto medio (mean_absolute_error)

Con la ayuda de la función `mean_squared_error`, que me sirve para comprobar el error cuadrático medio que tiene el modelo. A esta función le paso los datos `y_test` (datos del atributo calidad) y la predicción realizada en el anterior apartado. Muestro la MSE.

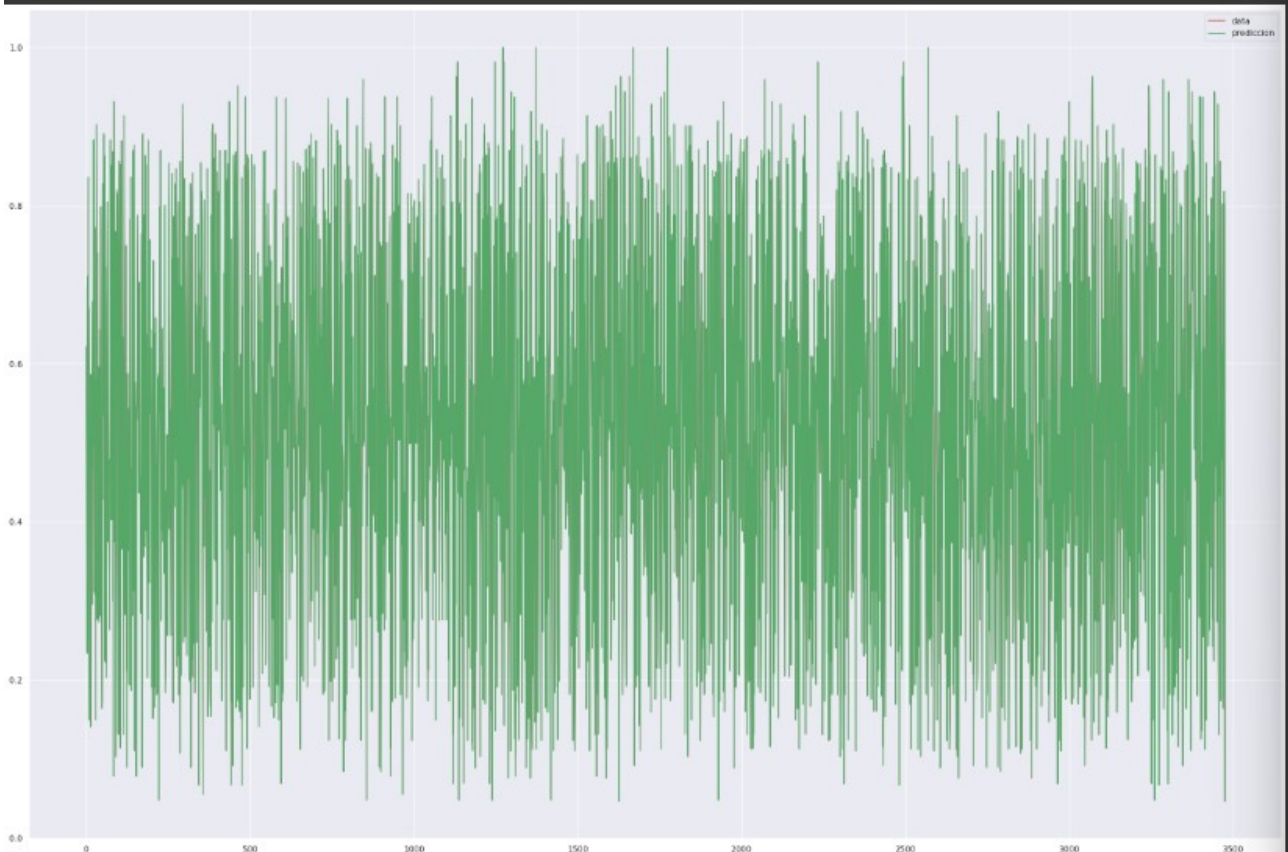
```
1 print("El error absoluto medio:", mean_absolute_error(y_test, y_prediccion))
```

```
El error absoluto medio: 1.1655096522003646e-05
```

4.3 Representar gráficamente los valores predichos con los valores reales

- `xx = np.stack([i for i in range(y_test.shape[0])])`
 - Se crea un array `xx` con índices que representan la posición de cada muestra en el conjunto de prueba. Básicamente, es un array `[0, 1, 2, ..., n]` donde `n` es el número de muestras en `y_test`.
- `plt.plot(xx, y_test, c='r', label='data')`
 - Se traza la curva de los valores reales (`y_test`) en color rojo (`c='r'`).
 - Se le asigna la etiqueta `'data'`, que aparecerá en la leyenda.
- `plt.plot(xx, y_prediccion, c='g', label='prediccion')`
 - Se traza la curva de los valores predichos (`y_prediccion`) en color verde (`c='g'`).
 - La etiqueta `'prediccion'` se usará en la leyenda.
- `plt.axis('tight')`
 - Ajusta los límites de los ejes automáticamente para que la gráfica se vea bien.
- `plt.legend()`
 - Muestra la leyenda con las etiquetas definidas en `label`.
- `plt.show()`
 - Muestra la gráfica en pantalla.

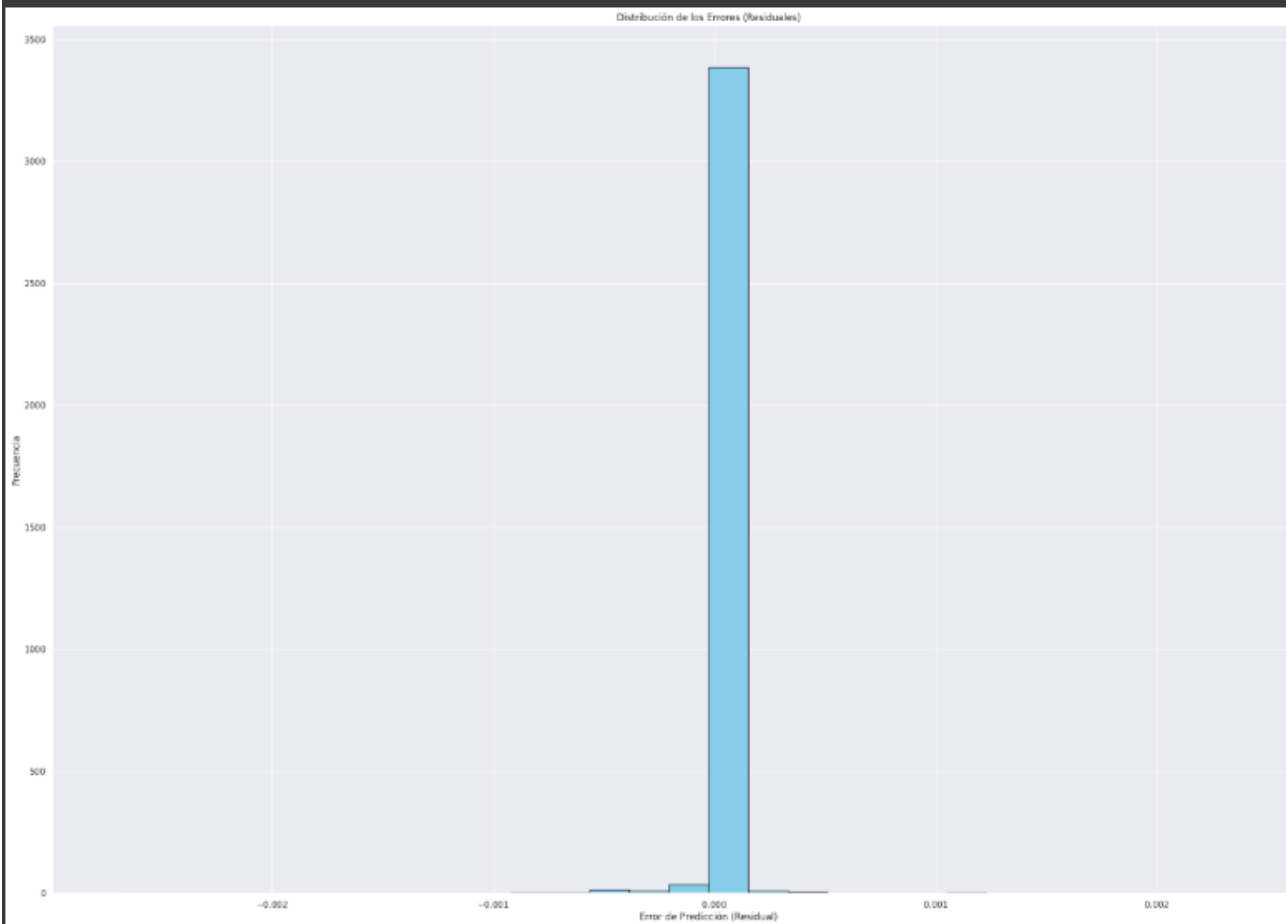
```
1 xx = np.stack([i for i in range(y_test.shape[0])])
2 plt.plot(xx, y_test, c='r', label='data')
3 plt.plot(xx, y_prediccion, c='g', label='prediccion')
4 plt.axis('tight')
5 plt.legend()
6 plt.show()
```



4.4 Distribución del Error (Histograma de residuales)

Muestra la distribución de los errores de predicción. Para un buen modelo, los residuales deberían estar normalmente distribuidos alrededor del cero.

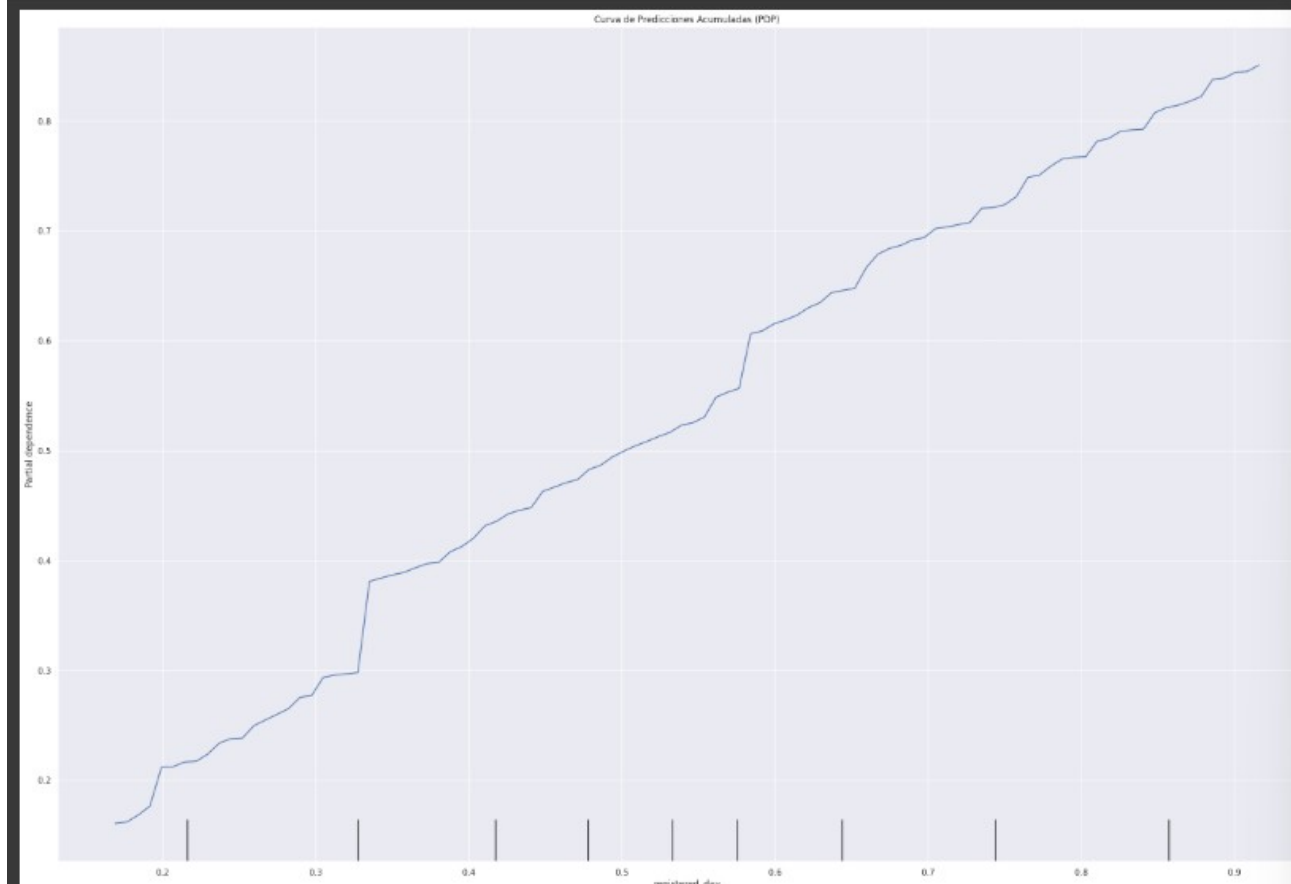
```
1 residuales = y_test - y_prediccion
2
3 plt.hist(residuales, bins=30, color='skyblue', edgecolor='black')
4 plt.xlabel("Error de Predicción (Residual)")
5 plt.ylabel("Frecuencia")
6 plt.title("Distribución de los Errores (Residuales)")
7 plt.show()
```



4.5 Curva de predicciones acumuladas (PDP)

Muestra cómo cambia la predicción promedio cuando una característica varía, manteniendo las demás constantes. Es útil para interpretar el impacto de una característica en el modelo.

```
1 # PDP para una característica en particular (ejemplo: primera columna)
2 PartialDependenceDisplay.from_estimator(modelo, x_test, [0])
3 plt.title("Curva de Predicciones Acumuladas (PDP)")
4 plt.show()
```



4.6 Importancia de las características

```
1 importancias = modelo.feature_importances_  
2 caracteristicas = x_entrenamiento.columns  
3  
4 df_importancias = pd.DataFrame({'Caracteristica': caracteristicas,  
    'Importancia': importancias})  
5 df_importancias = df_importancias.sort_values(by='Importancia', ascending=False)  
6  
7 sns.barplot(x='Importancia', y='Caracteristica', data=df_importancias)  
8 plt.title("Importancia de las características")  
9 plt.show()
```



5. Optimización de hiperparámetros

Antes de realizar la optimización de los hiperparámetros, si nos fijamos en la importancia de las características hay bastantes que tienen una importancia de 0.00. Estas serán eliminadas en este apartado. Además en el gráfico de la distribución del error podemos ver que es bastante bueno ya que la mayoría se maneja alrededor del 0.

5.1 Cada ejecución anterior se deberá hacer usando validación cruzada (por ejemplo `n_splits = 4`). Con ello obtendremos una medida de bondad del modelo (`accuracy_score` o `mean_absolute_error`), como lo ejecutaremos 4 veces, calcularemos la media de esas 4 ejecuciones.

La validación cruzada ha sido modificada de 5 a 4 porque estaba tardando demasiado.

```

1 df_opt = pd.DataFrame(data={}, columns=["max_depth", "error_cuadratico_medio",
    "error_absoluto_medio"])
2
3 kf = KFold(n_splits=4, shuffle=True, random_state=1)
4
5 for i in range(10, 501, 10):
6     modelo = RandomForestRegressor(n_estimators=i)
7     puntuaciones_mae = cross_val_score(modelo, x_entrenamiento, y_entrenamiento,
    cv=kf, scoring=make_scorer(mean_absolute_error))
8     puntuaciones_mse = cross_val_score(modelo, x_entrenamiento, y_entrenamiento,
    cv=kf, scoring=make_scorer(mean_squared_error))
9
10    media_mae = np.mean(puntuaciones_mae)
11    media_mse = np.mean(puntuaciones_mse)
12
13    df_opt.loc[len(df_opt)] = [i, media_mse, media_mae]
14
15 df_opt

```

| | | | |
|----|-------|--------------|----------|
| 18 | 190.0 | 1.925305e-07 | 0.000012 |
| 19 | 200.0 | 1.889170e-07 | 0.000012 |
| 20 | 210.0 | 1.912192e-07 | 0.000012 |
| 21 | 220.0 | 1.868451e-07 | 0.000012 |
| 22 | 230.0 | 1.855372e-07 | 0.000012 |
| 23 | 240.0 | 1.888468e-07 | 0.000012 |
| 24 | 250.0 | 1.903966e-07 | 0.000012 |
| 25 | 260.0 | 1.910061e-07 | 0.000012 |
| 26 | 270.0 | 1.893832e-07 | 0.000012 |

5.2 Finalmente los parámetros elegidos serán los que den mejor media de esas medidas anteriormente nombradas.

Me sorprende el valor grátamente del error medio absoluto medio. El modelo tiene 0.000011 desde el `n_estimators` 50, por lo tanto, estos serán los hiperparámetros porque si nos fijamos en el error absoluto medio en la mayoría es de 0.000012.

5.3 Una vez obtenidos esos parámetros óptimos los aplicaremos al problema en cuestión y mostraremos los resultados.

Realizo un modelo con los mejores hiperparámetros que he mencionado en el anterior apartado.

```
1 modelo = RandomForestRegressor(n_estimators=50, random_state=1)
2 modelo.fit(x_entrenamiento, y_entrenamiento)
3
4 y_prediccion = modelo.predict(x_test)
5
6
7 print("El error cuadrático medio:", mean_squared_error(y_test, y_prediccion))
8 print("El error absoluto medio:", mean_absolute_error(y_test, y_prediccion))
```

```
El error cuadrático medio: 3.0466492751200365e-09
El error absoluto medio: 2.7616892538512e-06
```