Aprendizaje de Máquina I

Trabajo práctico final

18 de Diciembre de 2021

Alumno: Agustín Baffo
Docente: Yoel Lopez

1.- Análisis y preprocesamiento de datos:

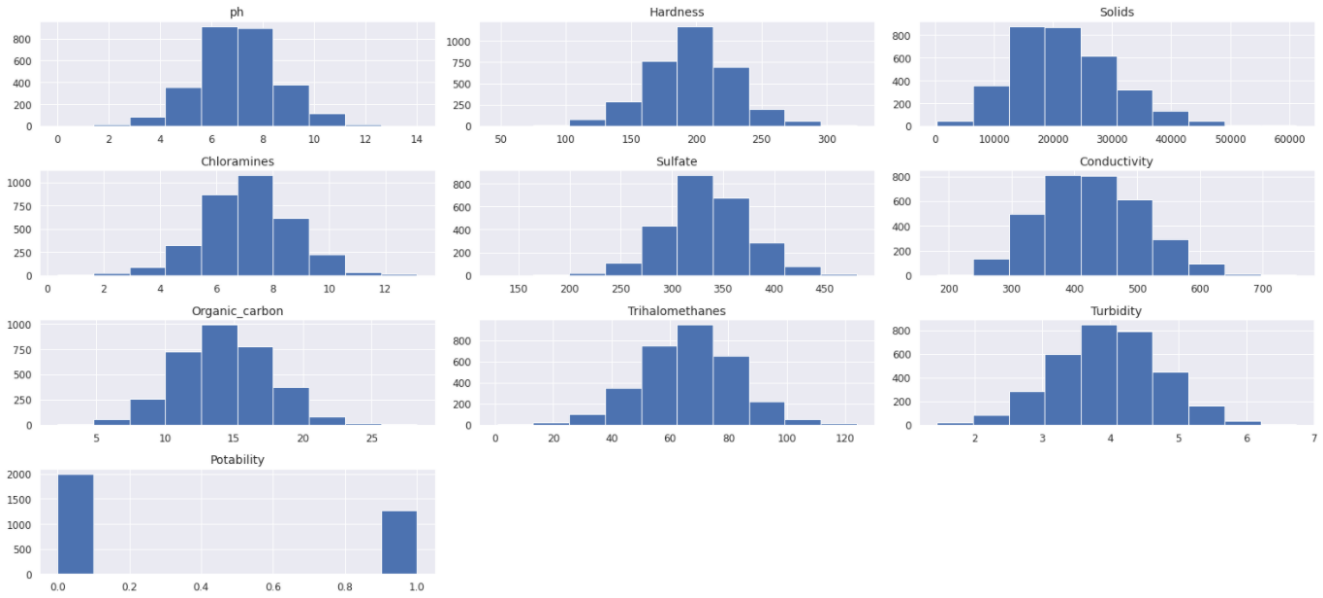
El análisis de datos se realiza en el cuaderno "*1_Analysis.ipynb*". En primer lugar, se buscó información externa de las columnas del dataset. A saber:

- ph: pH del agua (0 to 14).
- Hardness (Dureza): Capacidad del agua de precipitar jabón en mg/L
- Solids (Sólidos): Sólidos totales disueltos en ppm.
- Chloramines (Cloraminas): Cantidad de cloraminas en ppm.
- Sulfate (Sulfato): Cantidad de sulfato disuelto en mg/L.
- Conductivity (Conductividad): Conductividad eléctrica en µS/cm.
- Organic_carbon (Carbono orgánico): Cantidad de carbono orgánico en ppm.
- Trihalomethanes: Cantidad de Trihalomethanos en µg/L.
- Turbidity (Turbidez): Medición de la emisión de luz a través del agua en NTU.
- Potability (Potabilidad): Indica si el agua es potable 1: Potable, 0: No potable.

También se imprimen algunos datos del dataset, como la cantidad de valores no nulos, el tipo de datos, valores estadísticos de cada columna y se grafica su distribución.

```
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ph               2785 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64
dtypes: float64(9), int64(1)
```

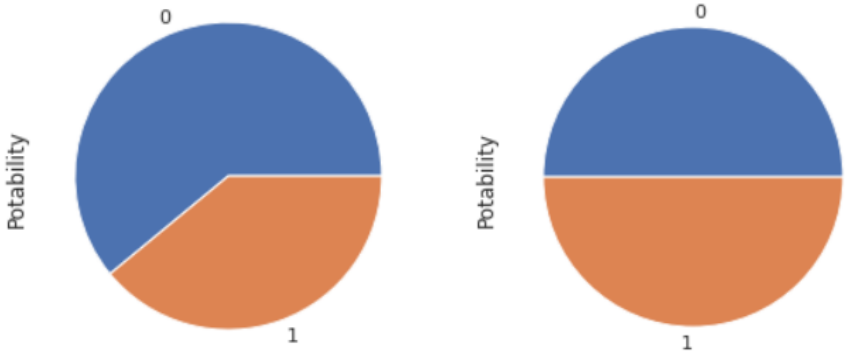
	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
count	2785.000000	3276.000000	3276.000000	3276.000000	2495.000000	3276.000000	3276.000000	3114.000000	3276.000000	3276.000000
mean	7.080795	196.369496	22014.092526	7.122277	333.775777	426.205111	14.284970	66.396293	3.966786	0.390110
std	1.594320	32.879761	8768.570828	1.583085	41.416840	80.824064	3.308162	16.175008	0.780382	0.487849
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754	2.200000	0.738000	1.450000	0.000000
25%	6.093092	176.850538	15666.690297	6.127421	307.699498	365.734414	12.065801	55.844536	3.439711	0.000000
50%	7.036752	196.967627	20927.833607	7.130299	333.073546	421.884968	14.218338	66.622485	3.955028	0.000000
75%	8.062066	216.667456	27332.762127	8.114887	359.950170	481.792304	16.557652	77.337473	4.500320	1.000000
max	14.000000	323.124000	61227.196008	13.127000	481.030642	753.342620	28.300000	124.000000	6.739000	1.000000



Vemos que la mayoría de las features tienen una distribución aproximadamente normal. Sin embargo la variable a predecir (Potability) está bastante desbalanceada. Para solucionar este problema se aplicó un proceso de upsampling de muestras, que consiste en repetir muestras del dataset de la variable con menor ocurrencia, hasta que las clases de salida queden con igual (o parecida) cantidad de datos . Los resultados obtenidos se muestran en la siguiente figura.

```
0    1998
1    1278
Name: Potability, dtype: int64
```

```
0    1998
1    1998
Name: Potability, dtype: int64
```

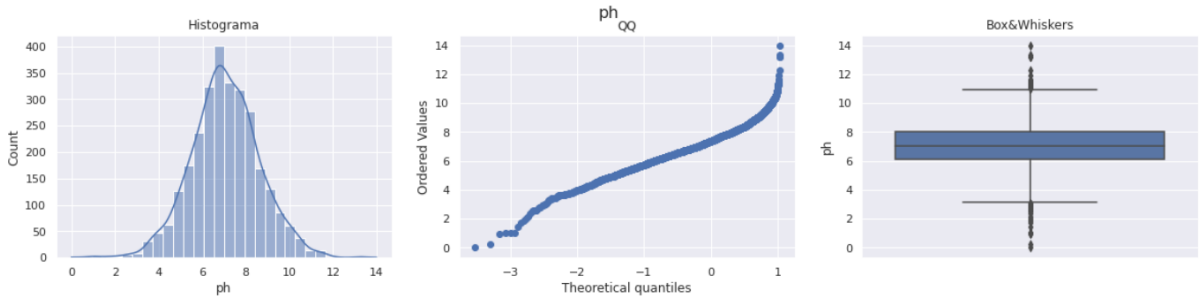


Antes de realizar cualquier exploración sobre el dataset, debe separarse en grupos de Train y Test, a fin de no influenciar nuestras decisiones con el conjunto de datos Test. Los datos de Test no se utilizarán hasta el final del trabajo, en donde se realiza la evaluación final del modelo seleccionado. En la división de datos se deja una proporción de 85% para Train y 15% para Test.

Sobre los datos de Train, se realiza un pairplot el cual revela que, en principio, no hay correlaciones lineales fuertes entre ninguna de las features de entrada por lo que se trabajará sobre todas las columnas del dataset.

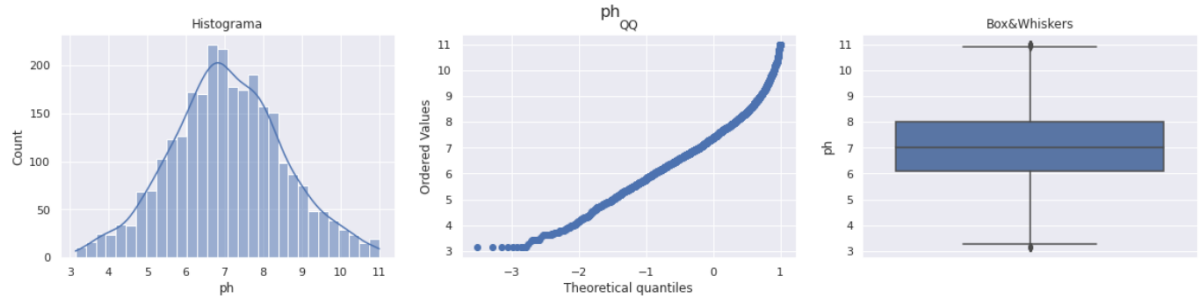
A continuación se detallan los demás pasos del preprocesamiento aplicados:

Outliers: El siguiente paso propuesto es la eliminación de outliers. Para esto, se graficaron los histogramas, qqplot y diagrama box&Whiskers de las variables de entrada. Tomando como ejemplo la columna ph, se obtiene lo siguiente:



El enfoque utilizados para el tratamiento de outliers fue una situación de compromiso entre eliminación de filas y asignación de valores a un umbral:

- Las filas con outliers mayores a 1.8*IQR fueron eliminadas del dataset.
- A los outliers por encima de 1.5*IQR se les asignó ese valor (1.5IQR) recortando la distribución en ese punto. Los resultados, fueron parecidos al siguiente ejemplo:
-



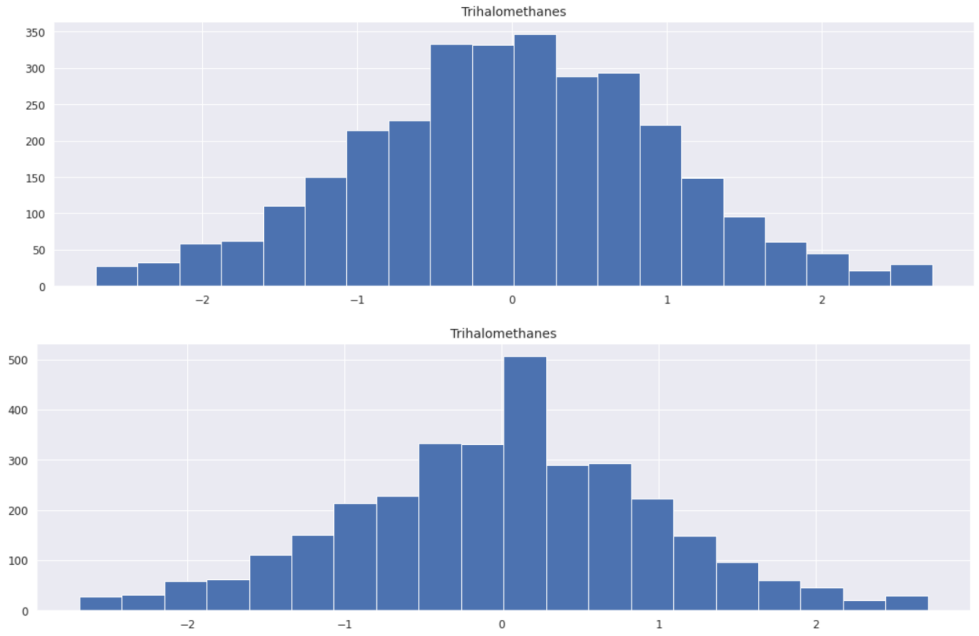
Vemos que se obtiene un diagrama de cajas con menores puntos atípicos, sin afectar tanto a la distribución de la variable.

Escalado: Muchos de los modelos que se utilizaran, requieren que las variables sean escaladas. Esto puede ser porque operan por distancias (como KNN), o porque al escalar los datos, se facilita la convergencia de algoritmos de optimización (como SGD).
Para dicho escalado se utilizó *StandarScaler* de sklearn que consiste en eliminar la media y escalar la varianza a la unidad. La transformación aplicada para una muestra x es: z = (x - u) / s; donde 'u' es la media de la feature y 's' es su desviación estándar.

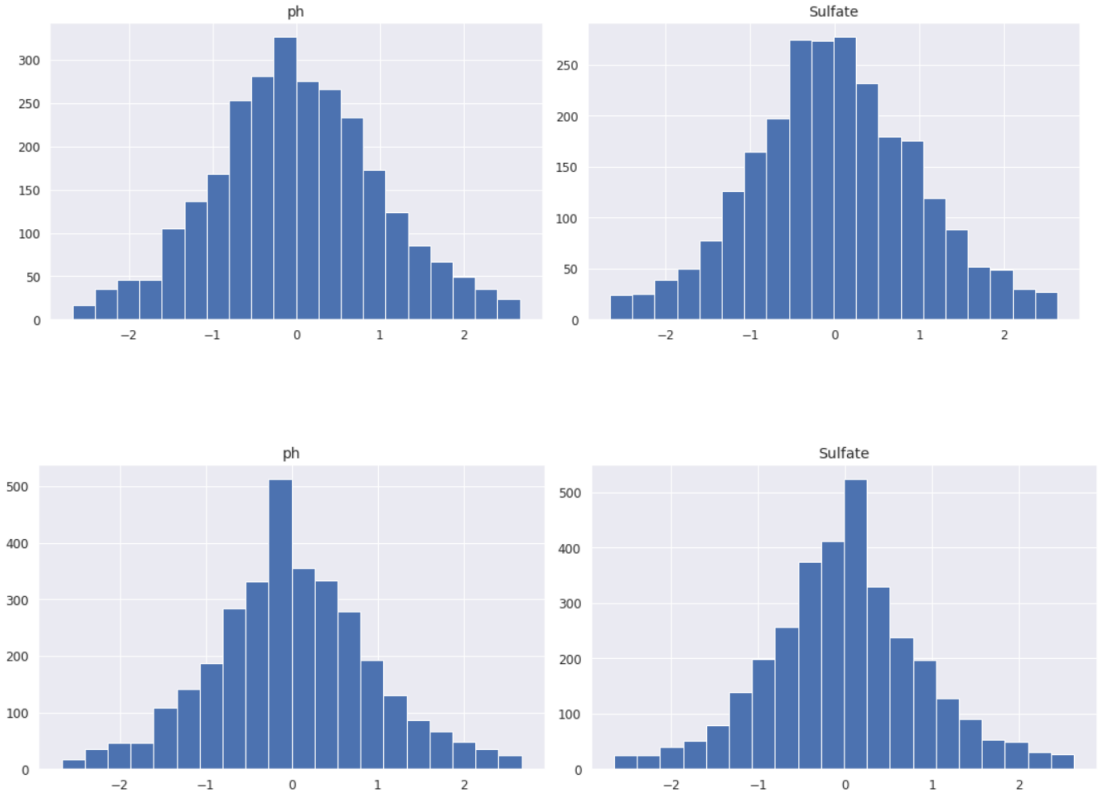
Imputación de datos faltantes: Se utilizaron dos técnicas diferentes dependiendo de la cantidad de faltantes que posee cada columna:

Imputación por mediana: las columnas con menos del 10% de datos faltantes fueron imputadas por mediana, es decir, se calcula la mediana de la columna y se reemplazan los

NaN por este valor (en cada columna). El criterio del 10% se tomó para no afectar demasiado la distribución de los datos. Se observa la distribución antes y después de imputar por mediana, de la variable Trihalomethanes:

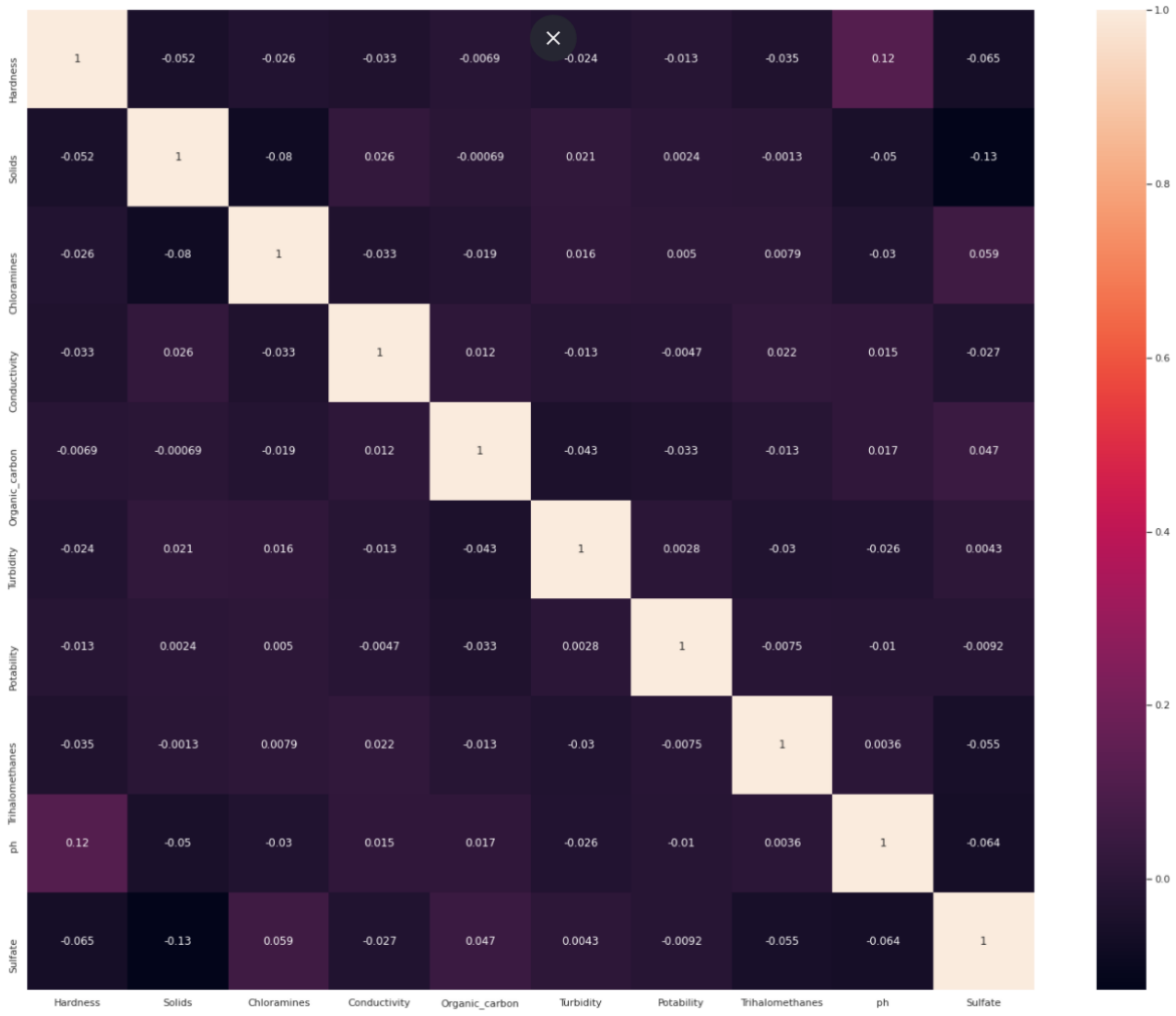


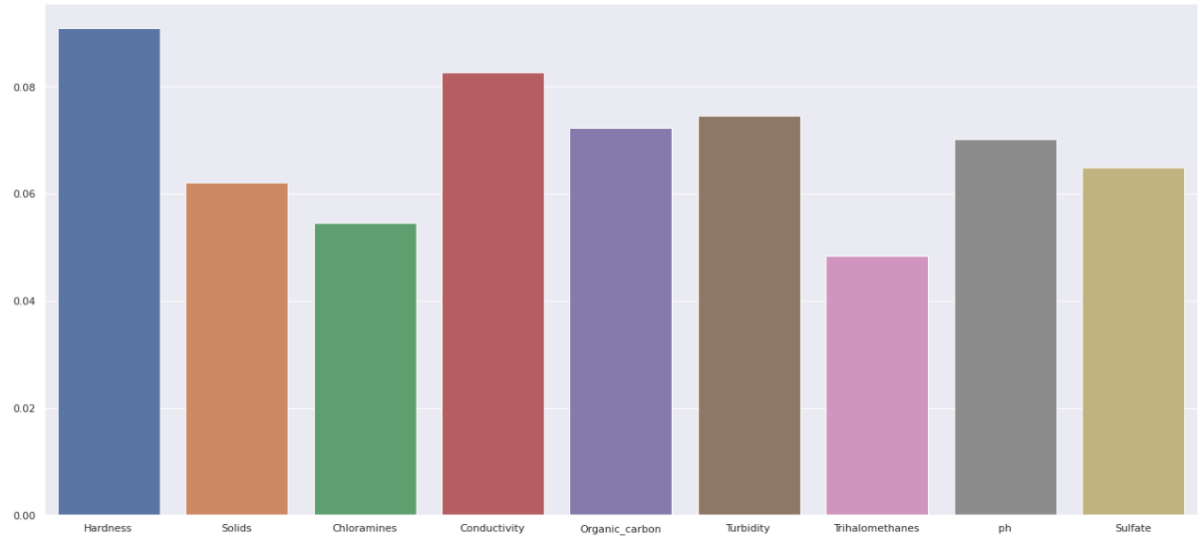
Imputación por KNN: las columnas con más del 10% de datos faltantes fueron imputadas con el algoritmo K-Nearest Neighbors. Se observa la distribución antes y después de imputar por mediana, de las variables ph y Sulfate:



Normalización: Finalmente, dado que la mayoría de los modelos asumen una distribución normal de los datos, se realiza una normalización. Se utilizó 'yeo-johnson' que permite que las variables tomen valores negativos.

Exploración de features: Se intentó buscar correlaciones y aplicar el criterio de la información mutua entre las features de entrada y la salida. Los resultados fueron los siguientes:





Dado que no se observan fuertes correlaciones entre las features y la salida, se decidió utilizar el algoritmo de PCA (Principal Component Analysis) para reducción de dimensionalidad y RFE (Recursive Feature Elimination) para selección de features. Esto se detalla en la sección de entrenamiento de modelos.

2.- Pipeline:

El cuaderno "*2_Pipeline.ipynb*" contiene el pipeline completo para preprocesar los datos. El procesamiento realizado, ese el mismo que se describe en la sección anterior, a saber:

- Upsample (para balancear los datos).
- División en Train/Test.
- Tratamiento de outliers.
- Escalado (standar scaler)
- Imputación por mediana
- Imputación por KNN
- Normalización del histograma

El pipeline fue diseñado como una función de python, en donde se le pasa un parámetro 'test_mode' para indicar si se ejecuta en modo train (análogo a fit_transform) o test (análogo a transform). Cuando se ejecuta en train, se guardan los transformers utilizados (scaler, imputer, outliers, etc.) y se retornan como un diccionario. Al ejecutarlo en modo test, se le pasa como parámetro dicho diccionario, y ejecuta las transformaciones correspondientes, entrenadas sobre el train, es decir, no se vuelven a entrenar estos transformers.

Finalmente, los datos preprocesados son guardados en un archivo tipo "pickle", que será leído luego por el cuaderno encargado del entrenamiento de los modelos.

3.- Entrenamiento y evaluación de modelos

El entrenamiento y la evaluación de los modelos se realiza en el cuaderno "*3_BinaryClassifier.ipynb*". Para esto, se cargan los datos preprocesados de los archivos pickle y se los dividen en dos grupos: Train y Validation. El conjunto de validación se utiliza para selección de hiperparámetros y para comparar resultados entre modelos. Tener en cuenta que el conjunto de test generado en el preprocesamiento, no se utiliza en está etapa.

Los modelos son entrenados con dos conjuntos de datos que se generan a partir de estos datos preprocesados. El primero surge de aplicar PCA para reducir la dimensionalidad de manera que los datos expliquen el 95% de la varianza en la salida. Se obtiene como resultado un conjunto de entrada de 9 dimensiones.

El segundo método, es la selección de features aplicando Recursive Feature Elimination (RFE), que permite seleccionar aquellas características (columnas) del conjunto de datos de entrenamiento que son más o más relevantes para predecir la variable objetivo. Las columnas seleccionadas son: ['Trihalomethanes_imputed', 'Sulfate', 'Hardness', 'Solids', 'Conductivity', 'Organic_carbon', 'Turbidity', 'ph_imputed', 'Sulfate_imputed'].

Otra variante que se ha aplicado al conjunto de datos es la de saltear el paso de escalado y entrenar los modelos con los datos en las magnitudes originales. Esto se implementó con el fin de comparar las métricas entre los modelos utilizando datos con y sin escalado.

Los modelos de prueba (a excepción de las redes neuronales) son entrenados en la función *train_models* que recibe entre sus parámetro los datos de Train y Validation, y otros valores que sirven para mostrar los resultados de manera ordenada. Para cada modelo, la función *train_models* realiza una llamada a la función *testModel* que utiliza *gridSearch* para la selección de los hiperparametros, y retorna como resultado el modelo entrenado y las métricas del mismo, evaluadas en el conjunto de validación. La métrica a optimizar es f1 score.

Los modelos entrenados son:

- Clasificador Dummy. Predice siempre la clase más frecuente.
- Regresión logística.
- SVM (SGDClassifier).
- Árbol de decisión
- Random Forest
- AdaBoost
- K-nearest neighbors algorithm (KNN)
- Redes neuronales densas.

La salida de la función *train_models* es un diccionario que contiene tanto el modelo entrenado, así como también información sobre dicho modelo y sus métricas.

Luego de entrenar los modelos de ML, se añade a este diccionario el modelo y los resultados de entrenar una pequeña red neuronal densa, con una capa de entrada de 100 neuronas, oculta de 30 y la salida de 1.

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(100))
model.add(tf.keras.layers.Activation('relu'))
model.add(tf.keras.layers.Dense(30))
model.add(tf.keras.layers.Activation('relu'))
model.add(tf.keras.layers.Dense(1))
model.add(tf.keras.layers.Activation('sigmoid'))
```

El diccionario de salida se convierte en un dataframe para visualizar los resultados en forma de tabla. Recordar que cada modelo es entrenado tanto para los datos seleccionados por RFE como para la reducción de dimensionalidad de PCA. Los resultados obtenidos, se muestran a continuación:

	name	features_selection	scaled	accuracy	precision	recall	f1_score	confusion_matrix
0	LogisticRegression	PCA	True	0.489297	0.482270	0.419753	0.448845	[[92, 73], [94, 68]]
1	SGDClassifier_hinge	PCA	True	0.495413	0.495413	1.000000	0.662577	[[0, 165], [0, 162]]
2	SGDClassifier_squared_hinge	PCA	True	0.492355	0.484848	0.395062	0.435374	[[97, 68], [98, 64]]
3	DecisionTreeClassifier	PCA	True	0.712538	0.682796	0.783951	0.729885	[[106, 59], [35, 127]]
4	RandomForestClassifier	PCA	True	0.761468	0.765823	0.746914	0.756250	[[128, 37], [41, 121]]
5	ADAboostClassifier	PCA	True	0.593272	0.590062	0.586420	0.588235	[[99, 66], [67, 95]]
6	KNN	PCA	True	0.660550	0.654545	0.666667	0.660550	[[108, 57], [54, 108]]
7	DummyClassifier	RFE	True	0.504587	0.000000	0.000000	0.000000	[[165, 0], [162, 0]]
8	LogisticRegression	RFE	True	0.492355	0.488636	0.530864	0.508876	[[75, 90], [76, 86]]
9	SGDClassifier_hinge	RFE	True	0.477064	0.478469	0.617284	0.539084	[[56, 109], [62, 100]]
10	SGDClassifier_squared_hinge	RFE	True	0.477064	0.483986	0.839506	0.613995	[[20, 145], [26, 136]]
11	DecisionTreeClassifier	RFE	True	0.694190	0.664894	0.771605	0.714286	[[102, 63], [37, 125]]
12	RandomForestClassifier	RFE	True	0.721713	0.720497	0.716049	0.718266	[[120, 45], [46, 116]]
13	ADAboostClassifier	RFE	True	0.553517	0.555556	0.493827	0.522876	[[101, 64], [82, 80]]
14	KNN	RFE	True	0.590214	0.588608	0.574074	0.581250	[[100, 65], [69, 93]]
15	NeuralNetwork	None	True	0.727829	0.703911	0.777778	0.739003	[[112, 53], [36, 126]]
16	LogisticRegression	RFE	False	0.504587	0.500000	0.543210	0.520710	[[77, 88], [74, 88]]
17	SGDClassifier_hinge	RFE	False	0.525994	0.517949	0.623457	0.565826	[[71, 94], [61, 101]]
18	SGDClassifier_squared_hinge	RFE	False	0.510703	0.509091	0.345679	0.411765	[[111, 54], [106, 56]]
19	DecisionTreeClassifier	RFE	False	0.660550	0.637838	0.728395	0.680115	[[98, 67], [44, 118]]
20	RandomForestClassifier	RFE	False	0.712538	0.697674	0.740741	0.718563	[[113, 52], [42, 120]]
21	ADAboostClassifier	RFE	False	0.504587	0.500000	0.425926	0.460000	[[96, 69], [93, 69]]
22	KNN	RFE	False	0.574924	0.562842	0.635802	0.597101	[[85, 80], [59, 103]]
23	NeuralNetwork	None	False	0.733945	0.709497	0.783951	0.744868	[[113, 52], [35, 127]]

Los mejores resultados para cada una de las métricas son los siguientes:

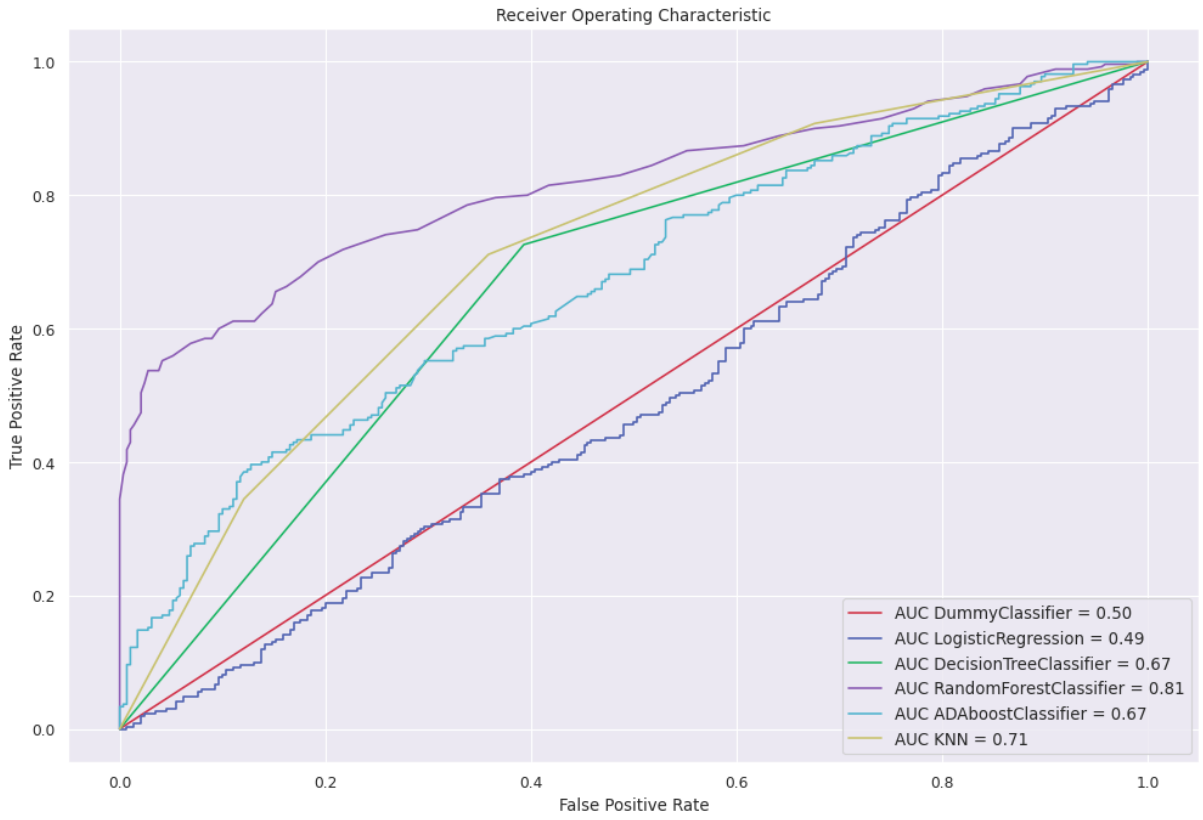
```
best accuracy =       0.76:     RandomForestClassifier (features=PCA) (scaled=True)
best precision =       0.77:    RandomForestClassifier (features=PCA) (scaled=True)
best recall =       1.00:       SGDClassifier_hinge (features=PCA) (scaled=True)
best f1_score =       0.76:     RandomForestClassifier (features=PCA) (scaled=True)
```

Finalmente, se cargan los datos de Test, del archivo pickle original y se evalúan las métricas sobre el modelo que mejor performance de f1 score tuvo para el conjunto de validación (Random Forest). El resultado final es:

Random Forest:

```
Using PCA
{'accuracy': 0.7625,
 'precision': 0.7773279352226721,
 'recall': 0.7111111111111111,
 'f1_score': 0.7427466150870407,
 'confusion_matrix': array([[235,  55],
        [ 78, 192]])}
```

También, se traza la curva ROC para alguno de los modelos evaluados. En todos los casos, se utilizó PCA y los datos escalados.



4.- Análisis y conclusiones:

Durante el trabajo, se propusieron diferentes modelos de Machine Learning y Redes Neuronales para resolver un mismo problema y poder comparar su rendimiento.

Si bien lo óptimo sería que para cada modelo se realice un preprocesado de datos acorde, por cuestiones de tiempo, el pipeline de preprocesado es el mismo, con algunas pequeñas variantes para evaluar comportamientos frente a ciertas técnicas. Por un lado, todos los modelos fueron testeados con datos generados a partir de dos técnicas: RFE y PCA. Se observa que los resultados obtenidos para ambas técnicas son muy similares, y no hay una que sea superior que la otra. Dependiendo del modelo, se obtuvieron levemente mejores métricas con RFE o con PCA. Como comentario, el modelo que mayor F1-Score alcanzó lo hizo con los datos de dimensionalidad reducida generados por PCA.

Por otro lado, se analizó cómo varían las métricas obtenidas cuando los datos no tienen la misma escala. Para esto se genera un nuevo conjunto de datos mediantes selección de features por RFE, en el cual no se aplica el StandarScaler. Vemos que para algoritmos basados en distancias, como KNN, disminuyen ligeramente accuracy y precisión, mientras que para algoritmos como Random Forest los resultados son semejantes. Por otro lado, las diferentes escalas podrían impactar negativamente en los tiempos de convergencia (o incluso generar divergencia) de algoritmos que utilicen optimización numérica como descenso de gradiente (ej en RNN).

Por su parte, el modelo de regresión logística tuvo una performance relativamente baja con respecto a la esperada. Además, también en contra de lo esperado, cuando los datos no se escalan los resultados obtenidos son muy similares. Quizá sea posible obtener mejores resultados con un ajuste más finos de hiperparametros, con con un preprocesado diferente en los datos.

Por otro lado, el modelo de Decision Tree Classifier obtuvo muy buenos resultados, tanto con como sin escalado de datos, e incluso mejor que la Red Neuronal. De todas formas cabe recalcar que la arquitectura de red planteada es demasiado sencilla y no se utilizó ningún método automático para la selección de hiperparametros, por lo que podrían mejorarse los resultados planteando otras arquitecturas y con diferentes hiperparametros.

Por último, se compararon algunos modelos de Machine Learning a través de la curva ROC y el AUC. La recta diagonal corresponde con el modelo dummy con un AUC 0.5. Vemos que Random Forest obtuvo un AUC de 0.82 que, como era de esperar, es bastante superior al AUC de 0.67 del modelo de un solo árbol de decisión (Decision Tree Classifier). ADAboost y KNN obtuvieron un AUC un poco menor, pero también aceptables. Finalmente, la regresión logística obtuvo un AUC menor que el modelo Dummy, lo que indica que claramente hay un problema con este modelo. Como se mencionó, es posible que modificando el

preprocesado de datos y los hiperparámetros del modelo, se puedan obtener mejores resultados, pero por cuestiones de tiempo se decidió dejarlo de está forma.