Pràctica 2:
WATER QUALITY

Rubén Simó Marín – 1569391

Alejandro García García – 1564537

Sergio Navarrete Villalta – 1565742

ÍNDEX

1. Apartat (B): Classificació numèrica

1. 1. EDA (exploratory data analysis)

Abans de començar farem els imports de totes les llibreries necessàries per a la realització d'aquesta pràctica i llegirem les dades del csv per carregar el nostre dataset.

```
# afegim llibreries necessaries
from sklearn.datasets import make_regression
import numpy as np
import pandas as pd
%matplotlib inline
from matplotlib import pyplot as plt
import scipy.stats
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn import preprocessing
import sklearn
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from statistics import mean
from sklearn import svm
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import f1_score
```

```
# visualitzarem només 3 decimals per mostra
pd.set_option('display.float_format', lambda x: '%.3f' % x)

# llegim les dades del csv
def load_dataset(path):
    dataset = pd.read_csv(path, header=0, delimiter=',')
    return dataset

# carreguem dataset
dataset = load_dataset('water_potability.csv')
data = dataset.values
```

Seguidament farem una primera visualització del dataset i els seus atributs per conèixer la seva dimensió i el número d'atributs.

```
dataset
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	NaN	204.890	20791.319	7.300	368.516	564.309	10.380	86.991	2.963	0
1	3.716	129.423	18630.058	6.635	NaN	592.885	15.180	56.329	4.501	0
2	8.099	224.236	19909.542	9.276	NaN	418.606	16.869	66.420	3.056	0
3	8.317	214.373	22018.417	8.059	356.886	363.267	18.437	100.342	4.629	0
4	9.092	181.102	17978.986	6.547	310.136	398.411	11.558	31.998	4.075	0
...	...	...	...	...	...	...	...	...	...	...
3271	4.668	193.682	47580.992	7.167	359.949	526.424	13.894	66.688	4.436	1
3272	7.809	193.553	17329.802	8.061	NaN	392.450	19.903	NaN	2.798	1
3273	9.420	175.763	33155.578	7.350	NaN	432.045	11.039	69.845	3.299	1
3274	5.127	230.604	11983.869	6.303	NaN	402.883	11.169	77.488	4.709	1
3275	7.875	195.102	17404.177	7.509	NaN	327.460	16.140	78.698	2.309	1

```
print("Dimensió base de dades: ", dataset.shape)
print("Número de atributs: ", dataset.columns.size)
```

```
Dimensió base de dades:  (3276, 10)
Número de atributs:  10
```

Com es pot observar, el arxiu "water_potability.csv" proporcionat per Kaggle és una base de dades que conté mètriques de qualitat d'aigua per a 3276 mostres de masses d'aigua diferents. El nostre dataset té una dimensionalitat de 3276 mostres i 10 atributs diferents.

NOM	DEFINICIÓ
Valor pH	El PH és un paràmetre important per avaluar l'equilibri àcid-base de l'aigua. També és l'indicador de la condició àcida o alcalina de l'estat de l'aigua. L'OMS ha recomanat el límit màxim permès de pH de 6,5 a 8,5. Els rangs d'investigació actuals van ser de 6,52 a 6,83 que estan en el rang d'estàndards de l'OMS.
Duresa	La duresa és causada principalment per les sals de calci i magnesi. Aquestes sals es dissolen de dipòsits geològics a través dels quals l'aigua viatja. El temps que l'aigua està en contacte amb el material que produeix duresa ajuda a determinar la duresa que hi ha en l'aigua. La duresa es va definir originalment com la capacitat de l'aigua per precipitar sabó causat pel calci i el magnesi.
Solids (Sòlids dissolts totals - TDS)	L'aigua té la capacitat de dissoldre una àmplia gamma de minerals inorgànics i alguns minerals orgànics o sals com potassi, calci, sodi, hidrogencarbonats, clorurs, magnesi, sulfats, etc. Aquests minerals produïen gust no desitjat i color diluït en aparença d'aigua. Aquest és el paràmetre important per a l'ús de l'aigua. L'aigua amb un alt valor TDS indica que l'aigua està altament mineralitzada. El límit desitjat per a TDS és de 500 mg/l i el límit màxim és de 1000 mg/l que es prescriu per a fins de beguda.
Cloramines	El clor i la cloramina són els principals desinfectants utilitzats en els sistemes d'aigua públics. Les cloramines es formen més comunament quan s'afegeix amoníac al clor per tractar l'aigua potable. Els nivells de clor fins a 4 mil·ligrams per litre (mg/L o 4 parts per milió) es consideren segurs en l'aigua potable.
Separació	Els sulfurs són substàncies naturals que es troben en minerals, sòls i roques. Són presents en l'aire ambient, les aigües subterrànies, les plantes i els aliments. L'ús comercial principal de sulfat és en la indústria química. La concentració de sulfurs en l'aigua de mar és d'uns 2.700 mil·ligrams per litre (mg/L). Va de 3 a 30 mg/L en la majoria dels subministraments d'aigua dolça, encara que es troben concentracions molt més altes (1000 mg/L) en alguns llocs geogràfics.
Conductivitat	L'aigua pura no és un bon conductor de corrent elèctric, sinó un bon aïllant. L'augment de la concentració d'ions augmenta la conductivitat elèctrica de l'aigua. Generalment, la quantitat de sòlids dissolts en aigua determina la conductivitat elèctrica. La conductivitat elèctrica (CE) mesura el procés iònic d'una solució que li permet transmetre corrent. Segons les normes de l'OMS, el valor de la CE no hauria de superar els 400 .S/cm.
Carboni orgànic	El carboni orgànic total (TOC) en aigües d'origen prové de la matèria orgànica natural en descomposició (NOM) així com de fonts sintètiques. El TOC és una mesura de la quantitat total de carboni en compostos orgànics en aigua pura. D'acord amb l'EPA dels Estats Units . 2 mg/L com a TOC en l'aigua tractada / beguda, i . 4 mg/Lit en l'aigua font que s'utilitza per al tractament.
Trihalometà	Els THMs són productes químics que es poden trobar en aigua tractada amb clor. La concentració de THM en l'aigua potable varia segons el nivell de material orgànic en l'aigua, la quantitat de clor necessari per tractar l'aigua, i la temperatura de l'aigua que s'està tractant. Els nivells de THM fins a 80 ppm es consideren segurs en l'aigua potable.
Turbiditat	La turbidesa de l'aigua depèn de la quantitat de matèria sòlida present en l'estat suspès. És una mesura de les propietats de l'aigua que emeten llum i la prova s'utilitza per indicar la qualitat de l'abocament de residus respecte a la matèria col·loïdal. El valor de turbiditat mitjana obtingut per Wondo Genet Campus (0,98 NTU) és inferior al valor recomanat per l'OMS de 5,00 NTU.
Potabilitat	Indica si l'aigua és segura per al consum humà, on 1 significa Potable i 0 significa No potable.

Un cop ja sabem la quantitat d'atributs que tenim i entenem la seva descripció hem de comprovar el tipus de dades que conté el nostre dataset. Per això executem:

```
print("Tipus d'atributs:")
dataset.dtypes
```

```
Tipus d'atributs:
ph                 float64
Hardness           float64
Solids             float64
Chloramines        float64
Sulfate            float64
Conductivity       float64
Organic_carbon     float64
Trihalomethanes    float64
Turbidity          float64
Potability           int64
dtype: object
```

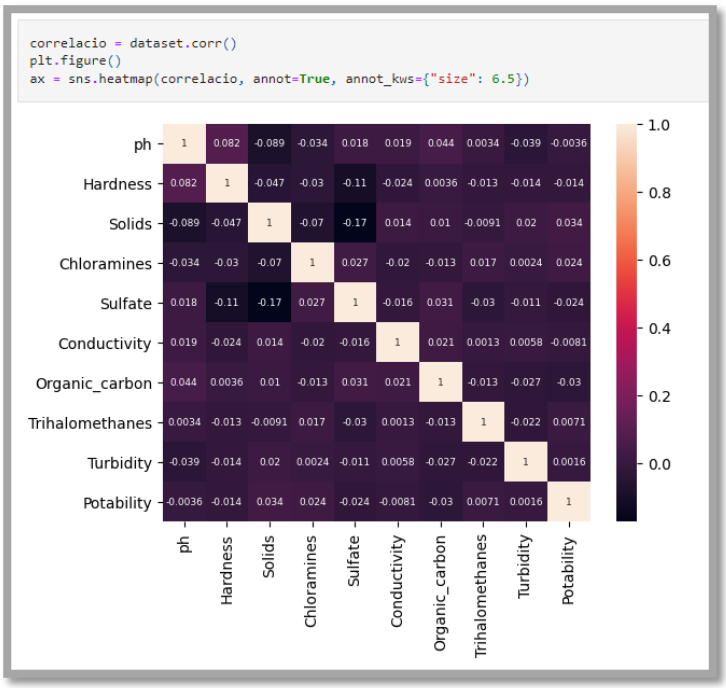
Com es pot observar, tots els atributs del nostre dataset son de tipus numèric. La majoria són atributs continus menys la potabilitat que és un atribut binari, això vol dir que tomarà el valor de 1 quan la mostra sigui segura per al consum humà i 0 quan no sigui segura per al consum humà.

Amb l'anterior afirmació deduïm que el nostre atribut target serà la potabilitat "Potability", el qual indica les 2 classes que pot tenir la nostra mostra:

- Classe 0: No potable
- Classe 1: Potable

A continuació visualitzarem la correlació que te el nostre atribut target "Potability" amb la resta d'atributs

```
correlacio = dataset.corr()
plt.figure()
ax = sns.heatmap(correlacio, annot=True, annot_kws={"size": 6.5})
```



Com es pot observar a la matriu de correlació, en general els nostres atributs no estan res correlacionats entre ells. Aquest fet no canvia pel nostre atribut target “Potability” el qual obté la màxima correlació amb l’atribut “Solids”, una correlació bastant baixa de 0.034. Una correlació tan baixa pot afectar a l’hora de fer prediccions, concretament afectarà a la regressió logística provocant un mal resultat.

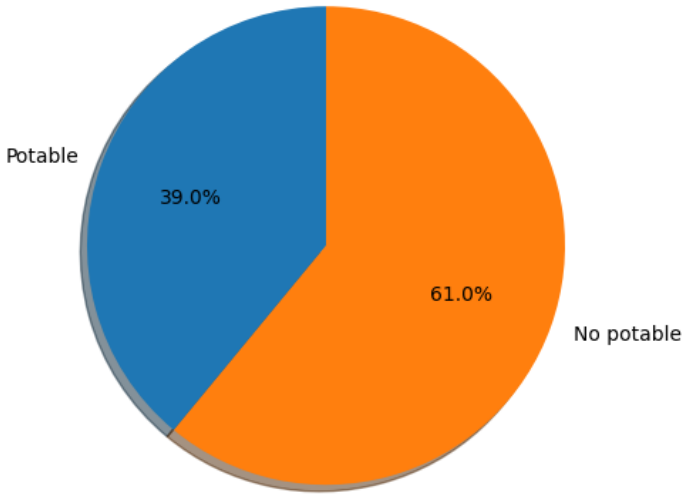
Comprovarem si el nostre atribut target està balancejat.

```
target = pd.Index(dataset['Potability'])
target.value_counts()
```

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

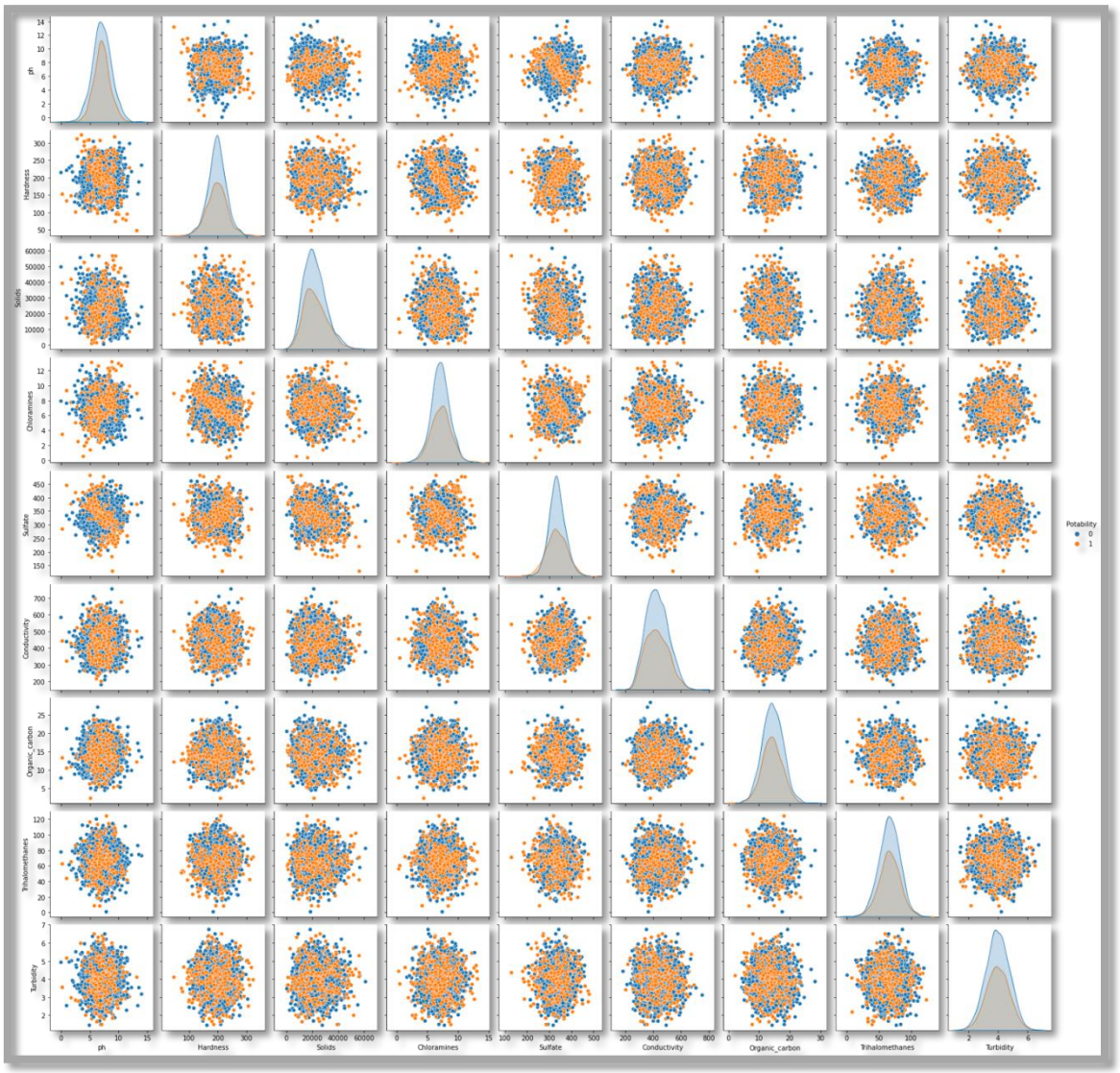
Realitzarem un gràfic per poder comprovar millor la seva distribució.

```
potable = dataset[dataset.Potability == 1]
noPotable = dataset[dataset.Potability == 0]

fig,ax1 = plt.subplots()
labels = 'Potable', 'No potable'
ax1.pie([potable.shape[0],noPotable.shape[0]], labels=labels, autopct='%1.1f%%', shadow=True, startangle=90)
ax1.axis('equal')

plt.show()
```



El gràfic ens mostra que l’atribut no esta del tot balancejat ja que hi ha un 22% mes de mostres no potables a la nostra base de dades. Considerem que és un balanceig normal, podria estar pitjor balancejat. Igualment, aquest desbalanceig afectarà directament a la classificació.

Ara realitzarem un pairplot per veure unes gràfiques generals de les nostres dades:



1. 2. Preprocessing (normalitzation, outlier removal, feature selection..)

Primerament, veurem si la nostra base de dades té moltes dades sense informació o si no en té cap dada sense informació.

```
dataset.isnull().sum()
```

```
ph                491
Hardness            0
Solids              0
Chloramines         0
Sulfate           781
Conductivity        0
Organic_carbon      0
Trihalomethanes   162
Turbidity           0
Potability          0
dtype: int64
```

Com podem veure a la imatge, tenim 491 dades de ph,781 dades de Sulfate i 162 dades de Trihalomethanes sense informació. Al ser un número de dades sense informació força elevat hem decidit substituir aquestes dades sense informació per la mitjana d'aquestes. És a dir, es calcula la mitjana de totes les altres dades disponibles de ph, Sulfate i Trihalomethanes, es fa la mitjana i aquests valors es substitueixen pels NaNs.

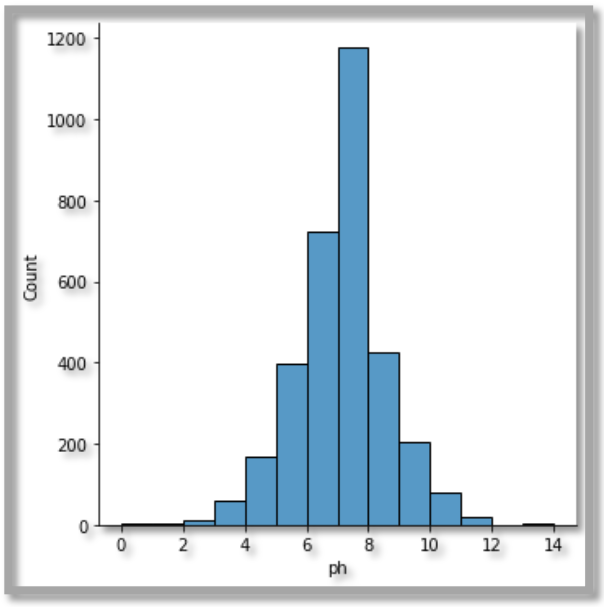
```
dataset['ph'].fillna(value=dataset['ph'].median(),inplace=True)
dataset['Sulfate'].fillna(value=dataset['Sulfate'].median(),inplace=True)
dataset['Trihalomethanes'].fillna(value=dataset['Trihalomethanes'].median(),inplace=True)
```

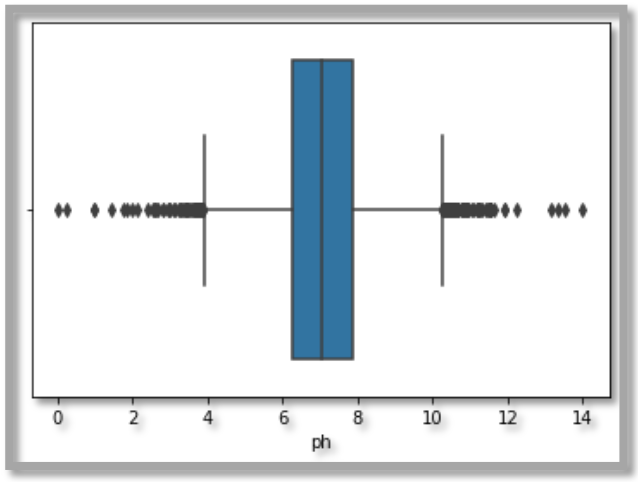
```
dataset.isnull().sum()
```

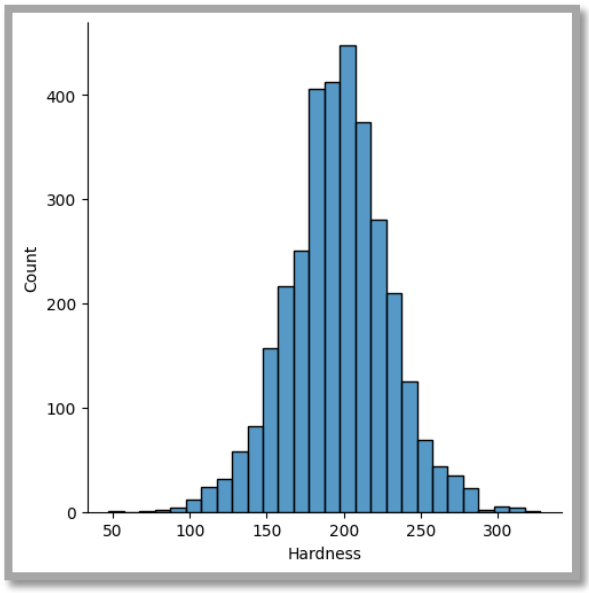
```
ph                0
Hardness          0
Solids            0
Chloramines       0
Sulfate           0
Conductivity      0
Organic_carbon    0
Trihalomethanes   0
Turbidity         0
Potability        0
dtype: int64
```

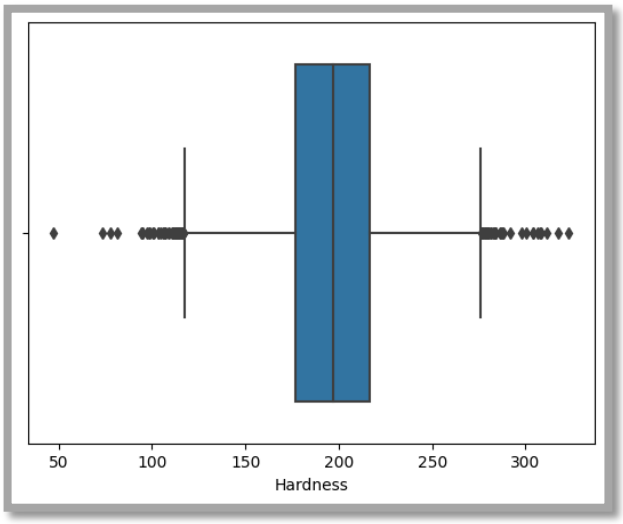
Com es pot veure, ara ja no hi han més dades amb valor inexistent o sense informació sinó que han sigut substituïdes per la mitjana com s'ha esmentat anteriorment.

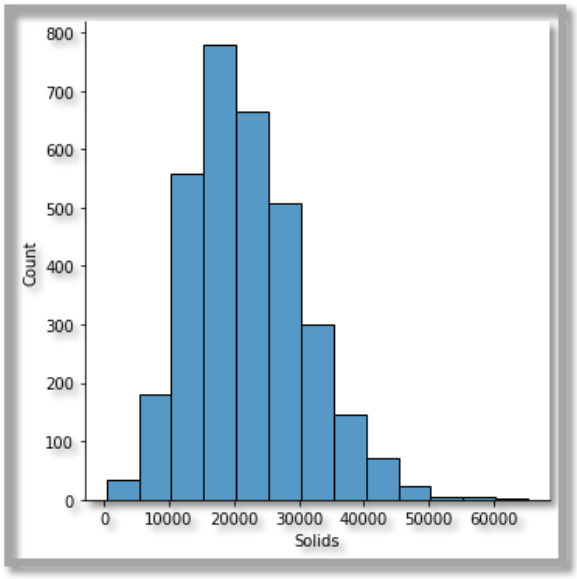
A continuació, estudiarem la nostra base de dades per veure si tenim una gran quantitat d'outliers(valors extrems o anormals) els quals poden afectar a l'hora de predir. Per comprovar aquests valors hem realitzat les següents gràfiques:

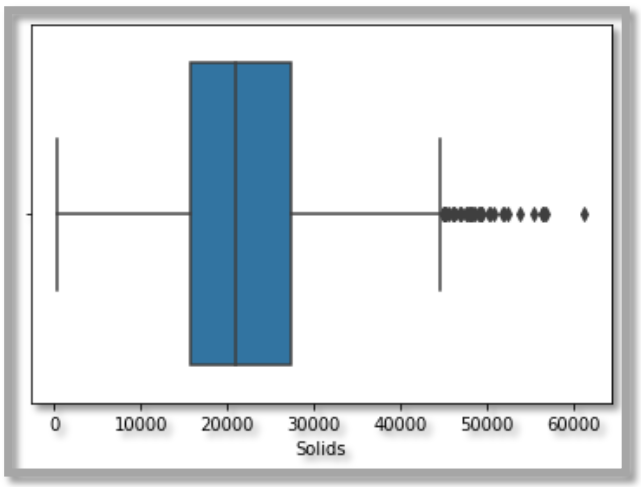

Count
ph

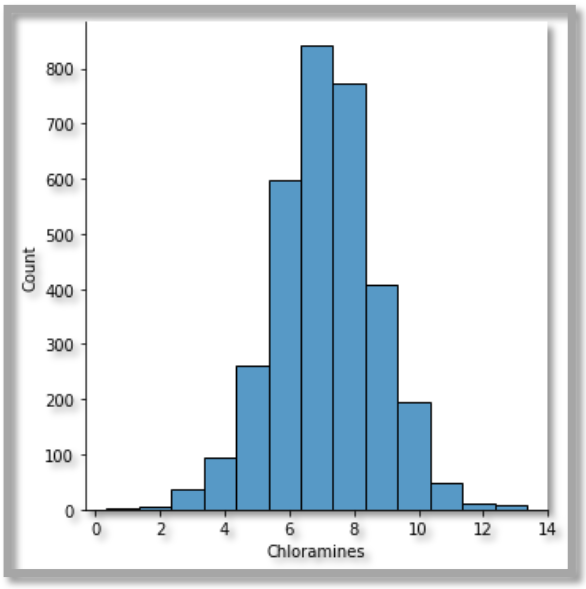

ph

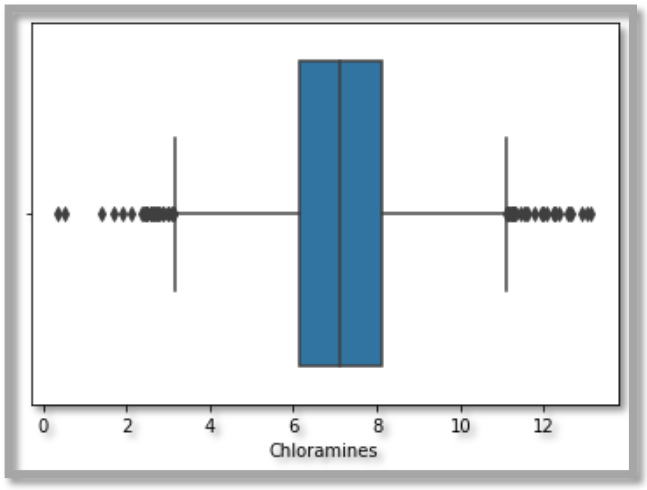

Count
Hardness

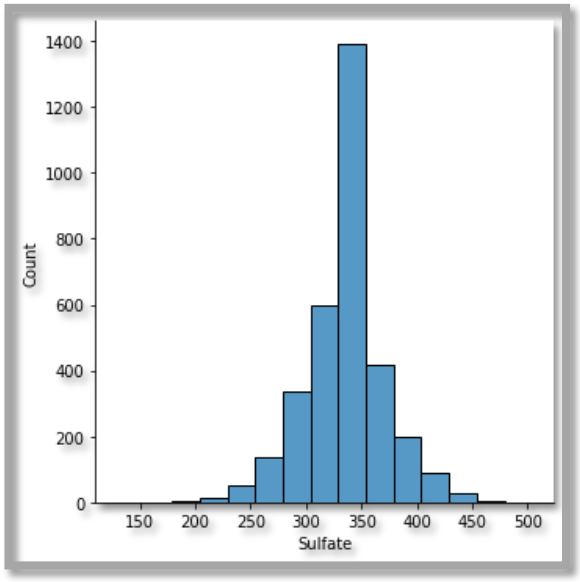

Hardness

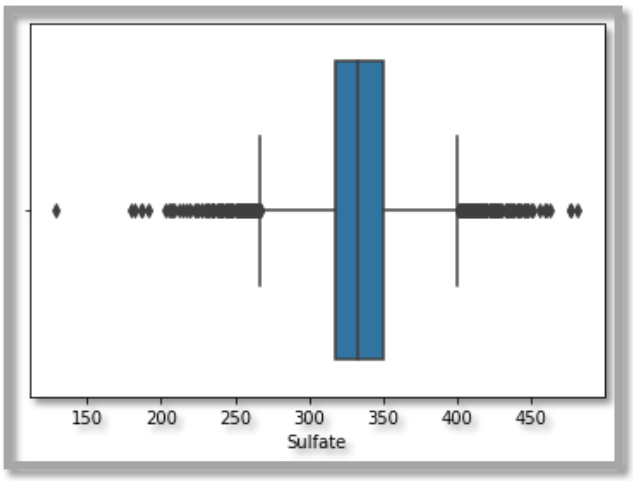

Count
Solids

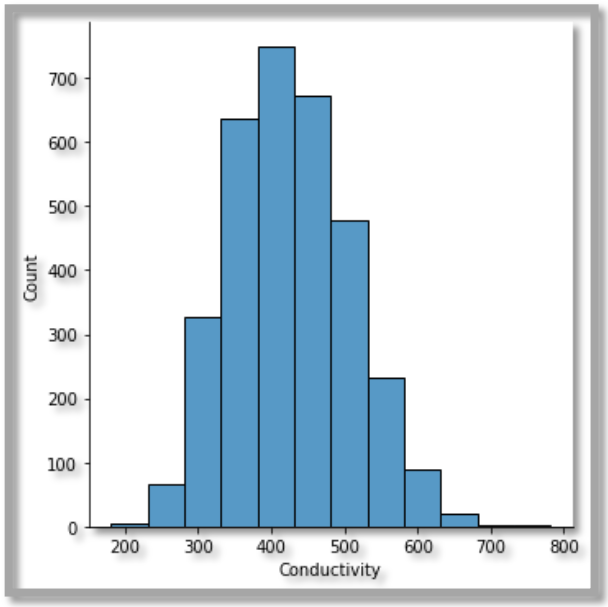

Solids

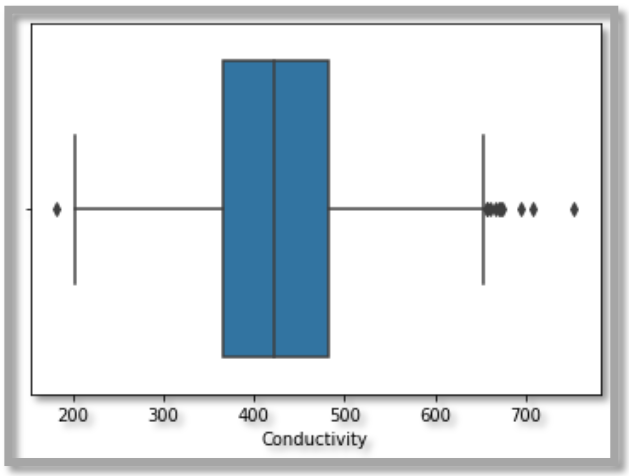

Count
Chloramines

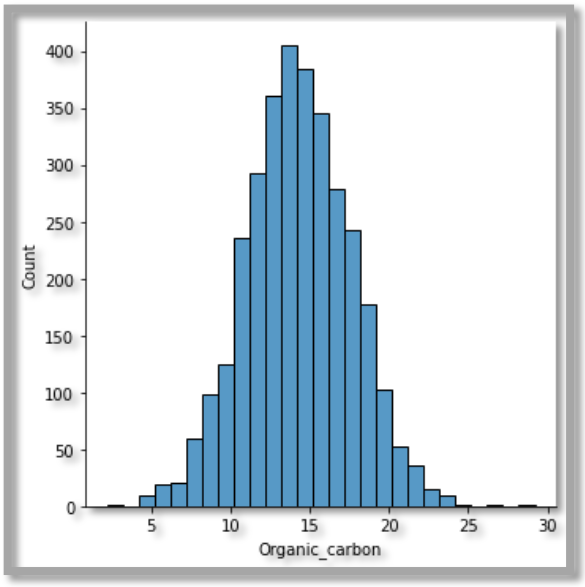

Chloramines

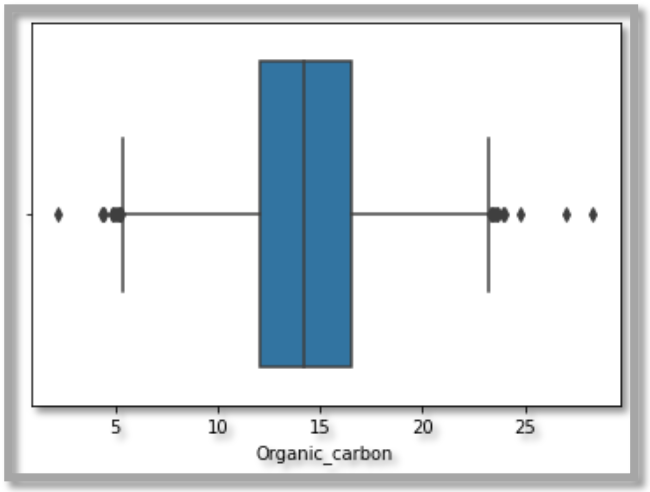

Count
Sulfate

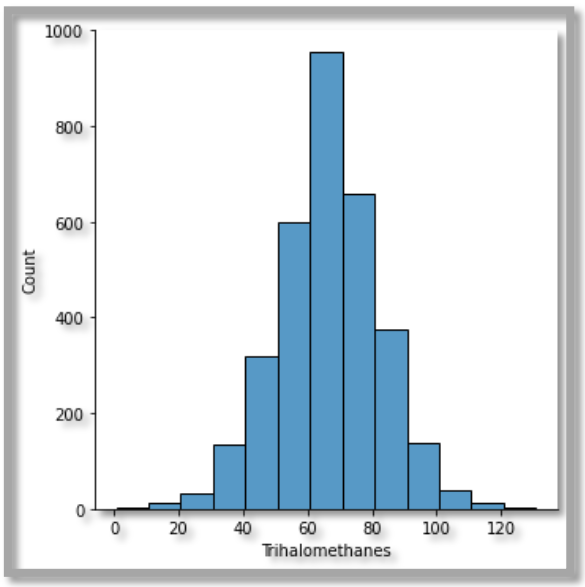

Sulfate

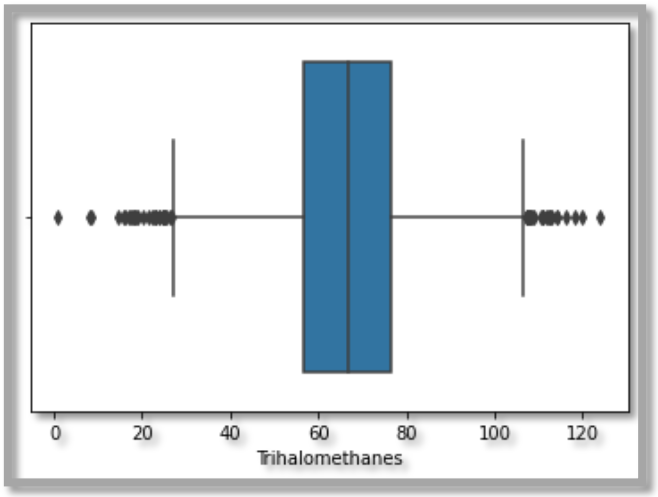

Count
Conductivity

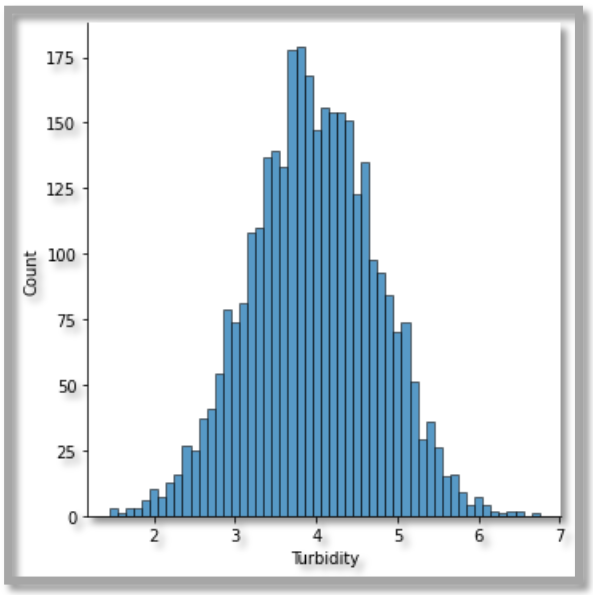

Conductivity

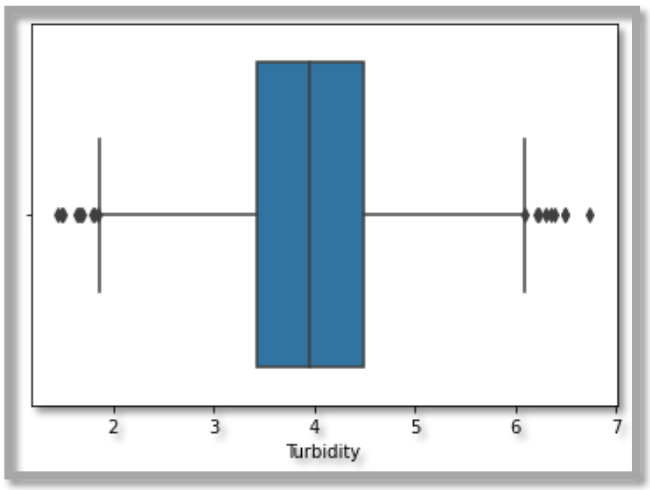

Count
Organic_carbon

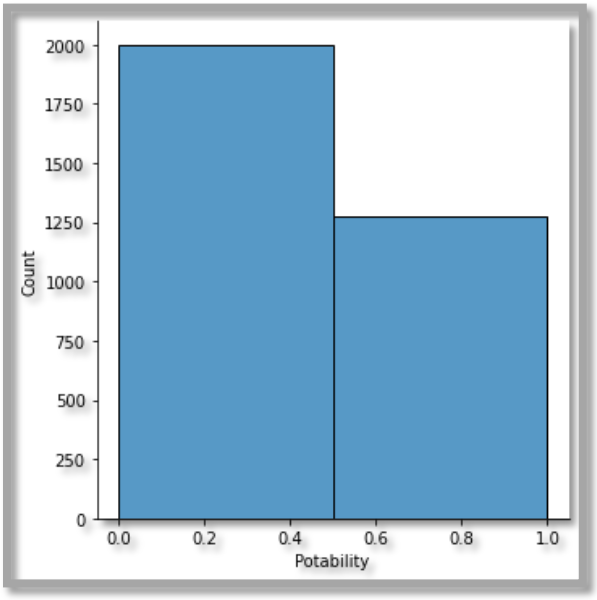

Organic_carbon

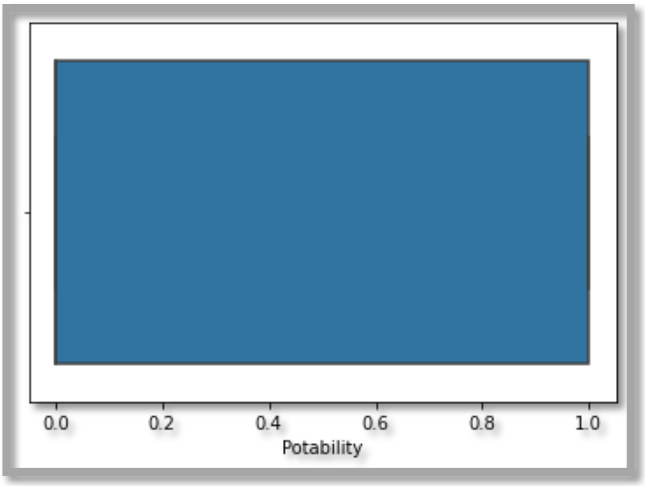

Count
Trihalomethanes


Trihalomethanes


Count
Turbidity


Turbidity





Com podem observar, no tenim gaires dades anormals o valors extrems molts clars, per tant hem decidit no eliminar-los ja que no afectaran gaire als models.

A la mateixa vegada hem volgut comprovar si les dades estan normalitzades i com es pot veure en casos anteriors no ho estan. En el nostre cas normalitzarem les dades ja que algunes de les conseqüències de no fer-ho pot ser inexactitud de dades i/o ineficiència d'operacions i com busquem trobar els millors resultats possibles és indispensable realitzar això.

En el nostre cas, dintre de la normalització utilitzarem la Standarization perquè amb aquest procés aconseguirem optimitzar l'aprenentatge del model de tal manera que tots els atributs d'entrada estaran dins de la mateixa escala y per tant no hi hauran atributs que tinguin un pes més elevat que altres només per l'escala de les seves dades.

```
#normalitzacio de dades utilitzant preprocessing

from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing

scaler = StandardScaler()
standardized = scaler.fit_transform(dataset)
standardized_dataset = pd.DataFrame(standardized, columns=dataset.columns)

print(standardized_dataset)
```

Per altra banda, com s'ha mencionat anteriorment en aquesta base de dades podem trobar dades categòriques, concretament a l'atribut 'Potability', però no és necessària cap codificació ja que aquest atribut ja es troba de manera binaria (amb integers) on 1 significa aigua potable i un 0 no potable.

A la vegada s'ha intentat realitzar PolynomialFeatures. Aquest mètode el que farà es crear una versió al quadrat o al cub d'una variable d'entrada la qual canviarà la distribució de probabilitat, separant els valors petits i grans. Aquesta separació pot ajudar a que alguns

algorismes d'aprenentatge automàtic facin millors prediccions ja que al estar més separats els valors serà més fàcil poder distingir entre ells i predir amb més eficàcia.
Però, en el nostre cas al no tenir molt pocs atributs no és convenient realitzar una polinomització ja que de tenir 10 atributs passem a 66.

1. 3. Model Selection

Per aquesta practica considerarem 3 models vists a classe:

- Logistic Regression: aquest model s'utilitza sovint per a la classificació i la anàlisis predictiu utilitzat per predir el resultat d'una variable categòrica (una variable que pot adoptar un nombre limitat de categories) en funció de les variables independents o predictors. Aquest és l'objectiu de la nostra pràctica i possiblement ens aporta la solució més ràpida.

- Nearest K Neighbors: és un algorisme d'aprenentatge automàtic simple i supervisat. L'entrada consisteix en els k exemples d'entrenament més pròxims d'un conjunt de dades i la sortida és una pertinença a una classe, és a dir, un objecte es classifica mitjançant un vot plural dels seus veïns, assignant l'objecte a la classe més comuna entre les seves k veïns més pròxims. És fàcil d'implementar i entendre, però té el gran inconvenient de tornar-se significativament lent a mesura que creix la grandària d'aquestes dades en ús

- SVM: Aquests mètodes estan relacionats amb problemes de classificació i regressió normalment per problemes de classificació de dos grups. Donat un conjunt d'exemples d'entrenament (de mostres) podem etiquetar les classes i entrenar una SVM per construir un model que predigui la classe d'una nova mostra. Dit d'una altra forma, donat un conjunt de punts, en el qual cadascun d'ells pertany a una de dues possibles categories, un algorisme basat en SVM construeix un model capaç de predir si un punt nou (la categoria del qual desconeixem) pertany a una categoria o a l'altra. Com tenim diferents kernels a implementar, segurament és amb el que podem aconseguir més precisió a costa de més complexitat computacional

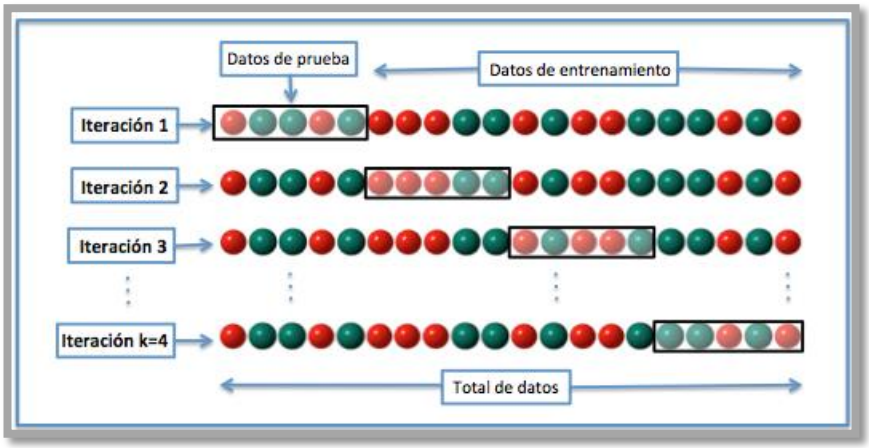
Per un altra banda tenim Ensemble. Ensemble és l'art de combinar un conjunt de diversos models per improvisar sobre l'estabilitat i el poder predictiu del model. Bàsicament, en lloc de fer un model i esperar que aquest sigui el millor predictor que puguem fer, els mètodes de conjunt (Ensemble) tenen en compte una gran quantitat de models i fan un promig amb aquests models per produir un model final combinant tots els models.
És veritat que com Ensemble no només utilitza un model sinó un conjunt d'ells els resultats seran, a priori, de major qualitat/precisió, però també té inconvenients.
A l'utilitzar més d'un model, és costos tant si parlem de temps com d'espai. A la vegada Ensemble és difícil d'aprendre i qualsevol selecció incorrecta pot conduir a una precisió predictiva més baixa que un model individual.
Per tant, Ensemble no és una mala opció però s'han de tenir en compte aquests dos darrers aspectes ja que sinó malbarataràs temps i espai per extreure pitjors resultats.

1. 4. Cross-validation

Cross-validation és una tècnica utilitzada per a avaluar els resultats d'una anàlisi estadística i garantir que són independents de la partició entre dades d'entrenament i prova. Realitzar Cross-Validation és important ja que permeteix comprovar que el model que s'està creant no produeixi overfiting. El que es fa es dividir el conjunt de dades d'entrenament en 2 subgrups, un per validar i un altre per entrenar. D'aquesta manera el que es fa es assegurar-se de que el model no faci prediccions de dades que ja havia vist abans, sinó que pugui fer prediccions de qualsevol dada.
Existeixen 3 maneres de realitzar cross-validation:

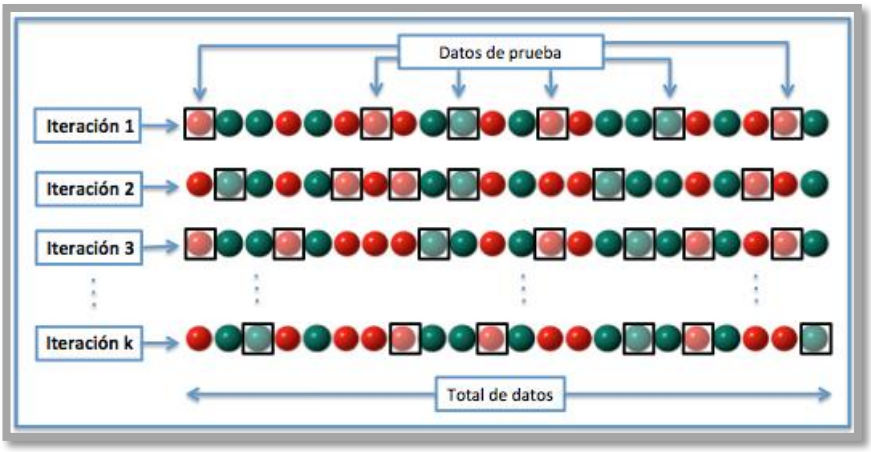
- Cross-validation k-fold:

Les dades de mostra es divideixen en K subconjunts. Un dels subconjunts s'utilitza com a dades de prova i el resta (K-1) com a dades d'entrenament. El procés de validació creuada es repeteix durant k iteracions, amb cada un dels possibles subconjunts de dades de prova. Aquest mètode és molt precís posat que avaluem a partir de K combinacions de dades d'entrenament i de prova però és lent des del punt de vista computacional. L'elecció del nombre d'iteracions depèn de la mesura del conjunt de dades. Típicament sol utilitzar 10 iteracions.

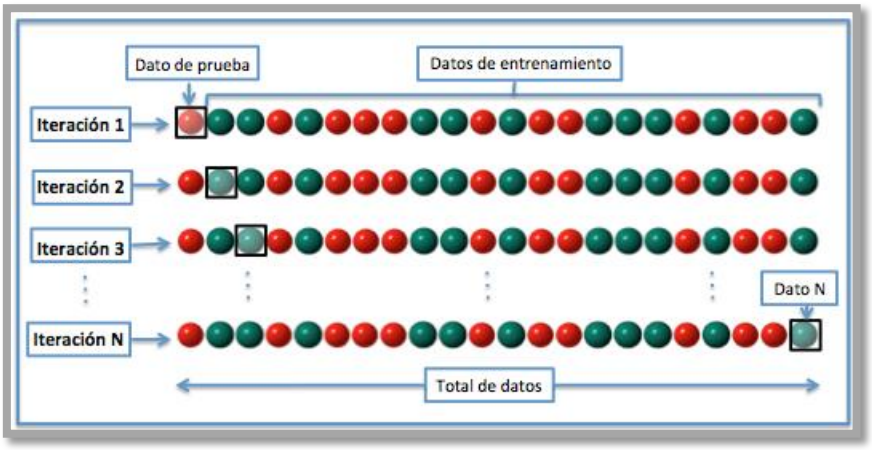


- Cross-validation sufflesplit:

CV aleatòria consisteix en dividir de manera aleatòria el conjunt de dades en dos conjunts: conjunt de dades d'entrenament i conjunt de dades de prova. El resultat final s'obté a partir dels mitjans aritmètics dels valors obtinguts per a les diferents divisions. L'avantatge d'aquest mètode és que la divisió de dades no depèn del nombre d'iteracions. En canvi, amb aquest mètode hi han algunes mostres que poden quedar sense avaluar i altres que es poden evaluar més d'una vegada, és dir, els subconjunts de prova i entrenament es poden solapar.



- Leave-one-out:
El mètode LOOCV és un mètode iteratiu que s'inicia emprant com a conjunt d'entrenament totes les observacions disponibles excepte una, que s'exclou per utilitzar-la com a test. Si es fa servir una única observació per calcular l'error, aquest varia molt depenent de quina observació s'hagi seleccionat. Per evitar-ho, el procés es repeteix tantes vegades com a observacions disponibles, excloent en cada iteració una observació diferent, ajustant el model amb la resta i calculant l'error amb aquesta observació. Finalment, l'error estimat pel LOOCV és la mitjana de tots els errors calculats.
Aquest mètode permet reduir la variabilitat que s'origina si es divideixen aleatòriament les observacions només en dos grups. Això és així perquè al final del procés de LOOCV s'acaben emprant totes les dades disponibles tant com a entrenament com a validació. Com que no hi ha una separació aleatòria de les dades, els resultats de LOOCV són totalment reproduïbles.
El principal desavantatge d'aquest mètode és el cost computacional. El procés requereix que el model sigui reajustat i validat tantes vegades com a observacions disponibles (n), això provoca que en alguns casos pugui ser molt complicat.



Ara bé, un cop separem les dades en conjunts de train i test es important saber com de fiables seran els resultats obtinguts. La fiabilitat dependrà principalment de la quantitat de dades d'entrenament utilitzades i de què dades estem utilitzant.
Per al primer punt, si prenem una mostra independent com a dada de prova (test), del mateix grup que les dades d'entrenament, normalment el model no s'ajustarà a les dades de prova igual de bé que a les dades d'entrenament. És a dir, que quan tenim poques dades d'entrenament o el nombre de paràmetres del model és gran els resultats obtinguts seran menys fiables i passarà l'esmentat anteriorment, overfitting.

Per al segon punt, és important conèixer amb què dades estem realitzant l'entrenament per a que els resultats siguin fiables. Per exemple, suposem que es desenvolupa un model per a predir el risc d'un individu per a ser diagnosticat amb una malaltia en particular. Si el model s'entrena amb dades que només afectin un grup de població específic (joves o homes) i després s'aplica a la població en general, els resultats del cross-validation del conjunt d'entrenament podrien diferir en gran manera de la classificació real i per tant no ser fiables.

Per realitzar el cross-validation utilitzarem el mètode de **cross-validation k-fold** ja que és el més adequat per al nostre problema al tenir una base de dades desbalancejada. A més, perquè el cross-validation shufflesplit pot provocar que algunes mostres es quedin sense avaluar i altres es poden avaluar més d'una vegada i el leave-one-out té un cost computacional molt elevat.

Per implementar-lo hem escollit el **StratifiedKFold** que es la versió millorada del KFold. A més, el StratifiedKFold soluciona el problema d'un conjunt de dades desequilibrades, es a dir, per al nostre problema és el més adient ja que com s'ha comentat anteriorment treballa millor amb un dataset desbalancejat.

És important escollir un bon valor de K per a que el model no pateixi una variança elevada o un bias elevat. A més, el valor de k també depèn del tamany del nostre conjunt de dades ja que una k elevada significa que només és possible un baix nombre de combinacions de mostres, la qual cosa limita el nombre d'iteracions que són diferents. Per això, realitzarem els següents càlculs per escollir un bon valor de K.

```
folds = range(2,31)

def evaluatemodel(cv, standar):

    x = standar[:,[0,1,2,3,4,5,6,7,8]]
    y = standar[:,9]
    lab = preprocessing.LabelEncoder()
    y_transformed = lab.fit_transform(y)

    logReg = LogisticRegression()
    scores = cross_val_score(logReg, x, y_transformed, scoring='accuracy', cv=cv, n_jobs=-1)
    return mean(scores), scores.min(), scores.max()

for k in folds:
    cv = KFold(n_splits=k, shuffle=True, random_state=10)
    k_mean, k_min, k_max = evaluatemodel(cv, standardized)
    print('-> folds=%d, accuracy=%.3f (%.3f,%.3f)' % (k, k_mean, k_min, k_max))
```

```
-> folds=2, accuracy=0.611 (0.596,0.626)
-> folds=3, accuracy=0.614 (0.589,0.631)
-> folds=4, accuracy=0.611 (0.584,0.628)
-> folds=5, accuracy=0.611 (0.569,0.631)
-> folds=6, accuracy=0.611 (0.577,0.632)
-> folds=7, accuracy=0.610 (0.577,0.643)
```

Com podem observar per veure quin és el millor valor de K hem realitzat la regressió logística que calcula en cada cas el cross validation en funció de la K. Normalment s'acostuma a escollir la K 5 o 10 però en el nostre cas podem determinar que la millor K és la 3 ja que la seva "accuracy" es la més gran de totes les obtingudes: 0.614.

1. 5. Metric Analysis

A continuació, haurem de trobar quina mètrica serà la més adient pel nostre problema. Primerament vam pensar en 3 mètriques:

-Accuracy_score: Aquesta mètrica representa el percentatge total de valors correctament classificats, tant positius com negatius.
És recomanable utilitzar aquesta mètrica en problemes en els quals les dades estan balançades

-F1_score: Aquesta és una mètrica molt utilitzada en problemes en els quals el conjunt de dades a analitzar està desbalançada. Aquesta mètrica combina el precision i el recall, per a obtenir un valor molt més objectiu.

-Average_precision_score: Calcula la mitjana de la precisió a partir dels valors de predicció. Es calcula la precisió de manera incremental, i finalment es calcula la mitjana.

Com el nostre conjunt de dades està una mica desbalancejat i al estar classificant un factor tant important per a la salut com pot ser si un aigua és potable o no, necessitem una mètrica amb molta objectivitat ja que hem de ser molt estrictes amb els resultats obtinguts. Per això escollim la **F1_score**.

A l'hora de visualitzar els nostres models visualitzarem també dues gràfiques anomenades **Precision-Recall Curve** i la **ROC Curve**. Però primer explicarem que són.

La **Precision-Recall Curve** és el resultat de dibuixar la gràfica entre el precision i el recall. Aquesta gràfica ens permet veure a partir de quina recall tenim una degradació de la precisió i viceversa. L'ideal seria una corba que s'acosti el màxim possible a la cantonada superior dreta (alta precisió i alt recall).

La **ROC curve** és semblant a la Precision-Recall Curve però canviant alguns valors. Relaciona el recall amb el ràtio de falsos positius. És a dir relaciona la sensibilitat del nostre model amb les fallades optimistes (classificar els negatius com a positius). Té sentit ja que, generalment, si augmentem el recall, el nostre model tendirà a ser més optimista i introduirà més falsos positius en la classificació. L'ideal seria una corba que s'acosti el màxim possible a la cantonada superior esquerra.

En general, la **Precision-Recall Curve** s'utilitza quan es té problemes de datasets no balancejats, és a dir, quan la classe positiva ocorre poques vegades. Quan hi ha pocs exemples positius, la **ROC curve** pot donar un valor alt, no obstant això, la **Precision-Recall Curve** estarà lluny del seu valor òptim, posant de manifest un indicador de precisió relacionat amb la baixa probabilitat de la classe positiva.

És una opció interessant usar la **ROC curve** quan es té un dataset més balancejat o vulguem posar de manifest un indicador més relacionat amb falses alarmes (falsos positius).

En el codi, hem realitzat dos funcions per crear i visualitzar cadascuna d'aquestes gràfiques:

```python
def PRCurve(y_v,probs, n_classes):
    precision = {}
    recall = {}
    average_precision = {}
    plt.figure()
    for i in range(n_classes):
        precision[i], recall[i], _ = precision_recall_curve(y_v == i, probs[:, i])
        average_precision[i] = average_precision_score(y_v == i, probs[:, i])

        plt.plot(recall[i], precision[i],
        label='Precision-recall curve of class {0} (area = {1:0.2f})'
                              ''.format(i, average_precision[i]))
        plt.xlabel('Recall')
        plt.ylabel('Precision')
        plt.legend(loc="upper right")

def RocCurve(y_v, probs, n_classes):
    fpr = {}
    tpr = {}
    roc_auc = {}
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_v == i, probs[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Compute micro-average ROC curve and ROC area
    # Plot ROC curve
    plt.figure()
    for i in range(n_classes):
        plt.plot(fpr[i], tpr[i], label='ROC curve of class {0} (area = {1:0.2f})' ''.format(i, roc_auc[i]))
    plt.legend()
```

A continuació aplicarem tot l'explicat anteriorment i visualitzarem els resultats per cada model. A més del F1_score i les gràfiques de Precision-Recall Curve i de ROC Curve també hem visualitzat la confusion Matrix, els nombres d'encerts i errades de cada model i l'accuracy del model anomenat score i el cross val score per avaluar els resultats del model.

- Logistic Regression:

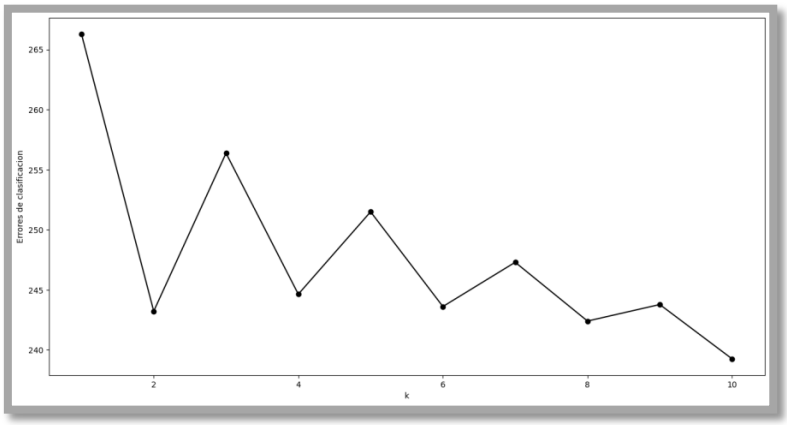
```
LOGISTIC REGRESION

Logistic Regression Score:  0.6036585365853658
Logistic Regression Cross Val Score:  0.611450559762622

260 Errores de clasificacion de un total de 656
396 Aciertos de clasificacion de un total de 656

Confusion matrix:
[[396   0]
 [260   0]]

Logistic Regression F1 Score: 0.6036585365853658
```

Com es pot observar, per aquest model, s'aconsegueixen valors de Score, Cross Val Score i F1 Score molt baixos. Aquest valors indiquen una molt baixa precisió del model alhora de realitzar les prediccions, aquest fenomen pot ser degut a la ínfima correlació dels atributs de la nostra base de dades.
A més, observant la confusion matrix podem veure com en cap moment el model detecta cap cas d'aigua potable, només detecta casos d'aigua no potable. Això reflecteix com el model no és precisament el que busquem ja que no és capaç d'identificar cap cas d'aigua potable. Segurament això succeeix a causa de la poca correlació que existeix entre els diferents atributs de la base de dades. Per tant aquest model queda totalment descartat.

- **Nearest K Neighbors:**

Primerament, el que s'ha realitzat és una comprovació per veure amb quina k s'obtindran millors resultats.

```
NEAREST K NEIGHBORS

k=1: 266.301 errores de clasificación de un total de 655
k=2: 243.225 errores de clasificación de un total de 655
k=3: 256.373 errores de clasificación de un total de 655
k=4: 244.64 errores de clasificación de un total de 655
k=5: 251.497 errores de clasificación de un total de 655
k=6: 243.613 errores de clasificación de un total de 655
k=7: 247.296 errores de clasificación de un total de 655
k=8: 242.405 errores de clasificación de un total de 655
k=9: 243.781 errores de clasificación de un total de 655
k=10: 239.26 errores de clasificación de un total de 655
```



Com es pot veure en la gràfica i en el text anterior, la k que donarà millors resultats (menor nombre d'error) és la k igual a10.

```
Nearest K Neighbour Score:  0.6509146341463414
Nearest K Neighbour Cross Val Score:  0.6389266782175319

229 Errores de clasificacion de un total de 656
427 Aciertos de clasificacion de un total de 656

Confusion matrix:
[[360  36]
 [193  67]]

Nearest K Neighbour F1 Score: 0.6509146341463414
```

Una vegada executat, podem veure com el model KNN dona millors resultats de precisió alhora de prediré una dada que el model de regressió logística. Fins i tot sent millors resultats que l'anterior model, aquests continuen sent valors molt baixos. Aquest fet pot ser degut a que l'algorisme KNN dona el mateix pes a tots els atributs encara que alguns tinguin pitjor correlació amb l'atribut a prediure.

També podem observar com, a diferència del model anterior, en aquest model si es detecten tant casos d'aigua potable com casos d'aigua no potable encara que siguin correctes o incorrectes.
Mirant els resultats de la matriu de confusió podem veure com ha detectat 553 casos d'aigua no potable però només 360 de veritat eren aigua no potable i com ha detectat 103 casos d'aigua potable però només 36 eren de veritat aigua potable.

- SVM:
Abans de posar en pràctica el model SVM, s'han d'especificar 2 valors molt importants que influeixen directament en els resultats obtinguts. Aquests son la C i gamma.

El paràmetre C indica a l'optimització del SVM quant vol evitar la classificació errònia de cada exemple d'entrenament. Per a valors grans de C, l'optimització triarà un hiperplà de marge més petit però tots els punts d'entrenament estaran classificats correctament. Per contra, un valor molt petit de C farà que l'optimització seleccioni un hiperplà de marge més gran però es poden classificar erròniament els punts d'entrenament.

Gamma es fins on arriba la influència d'un sol exemple d'entrenament, amb valors baixos que signifiquen "lluny" i valors alts que signifiquen "a prop". Els valors més baixos de gamma donen com a resultat models amb menor precisió i el mateix que els valors més alts de gamma. Són els valors intermedis de gamma els que donen un model amb bons límits de decisió.

Per tant, per al nostre problema els valors de C i gamma que ens permeten obtenir una major precisió son **C = 0.9** i **gamma = 0.1**.

```
SVM with rbf kernel Score:  0.6935975609756098
SVM with rbf kernel Cross Val Score:  0.6595395555974951

201 Errores de clasificacion de un total de 656
455 Aciertos de clasificacion de un total de 656

Confusion matrix:
[[381  15]
 [186  74]]

SVM with rbf kernel F1 Score: 0.6935975609756098
```

```
SVM Linear Score:  0.6036585365853658
SVM Linear Cross Val Score:  0.611450559762622

260 Errores de clasificacion de un total de 656
396 Aciertos de clasificacion de un total de 656

Confusion matrix:
[[396   0]
 [260   0]]

SVM Linear F1 Score: 0.6036585365853658
```

El SVM és un model per a classificacions de dos grups cosa que a priori beneficiaria al nostre problema. Nosaltres hem utilitzat dos tipus de SVM: el SVC amb rbf kernel i el SVC linear

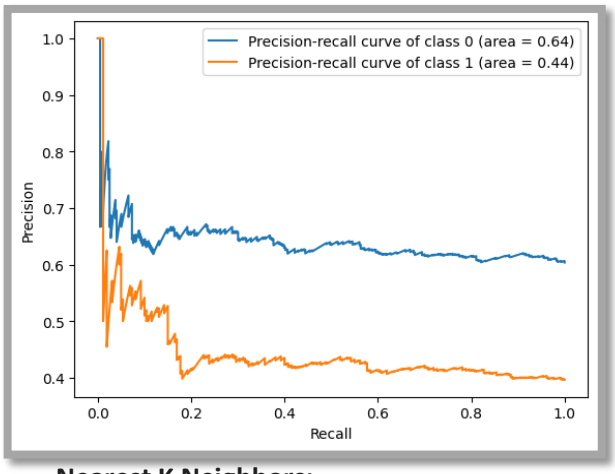
El primer d'aquests dos, encara que els seus resultats segueixen sent molt inferiors als desitjats, en comparació als altres dos models vists anteriorment és millor. Els seus valors són més elevats i per tant el seu nombre d'encerts també és més elevat. Observant la matriu de

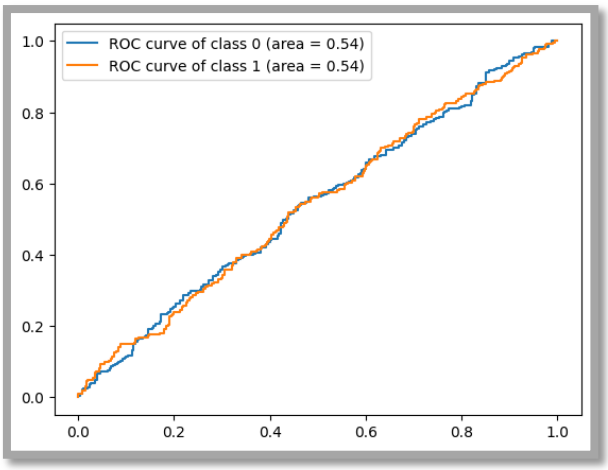
confusió podem veure com és el model que més classificacions correctes ha realitzat tenint en compte tant els casos no potables com els casos potables.

En canvi el segon dona exactament igual que el model de regressió logística i succeeix també que no pot identificar cap cas d'aigua potable. Per tant aquest model també queda totalment descartat.

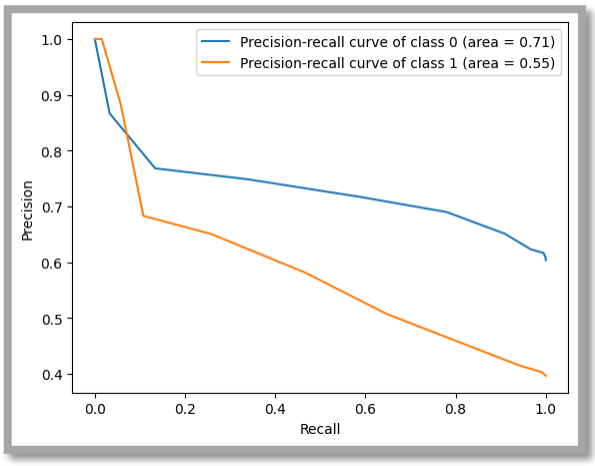
Precision Recall Curve and Roc Curve

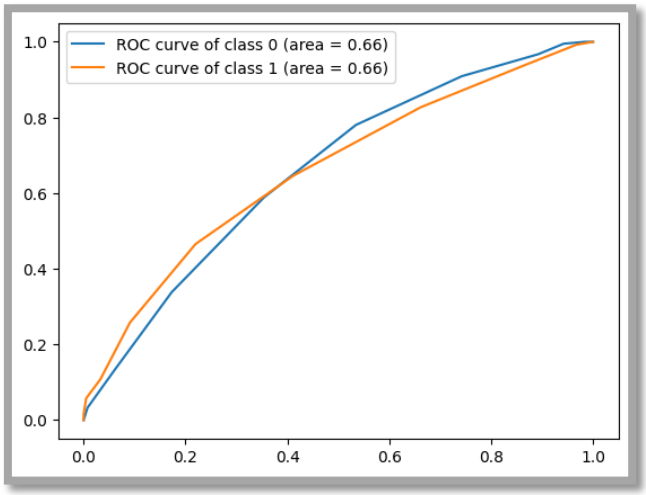
- Logistic Regression:



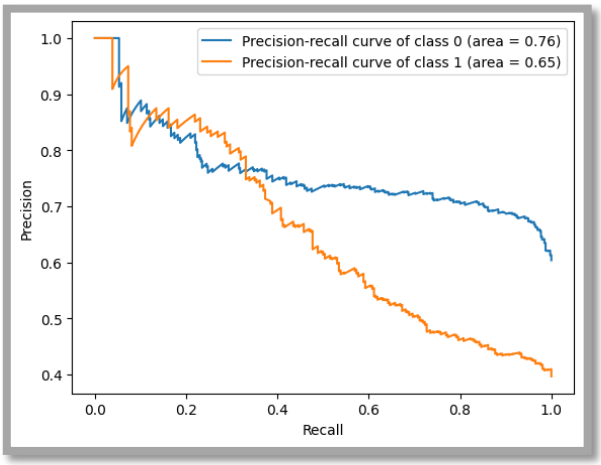


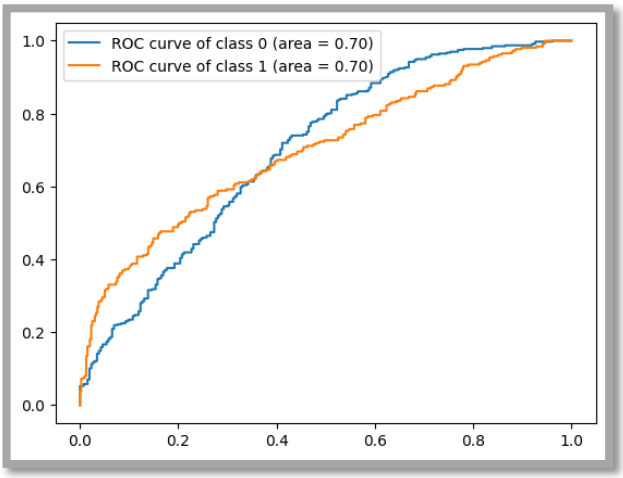
- Nearest K Neighbors:





- SVM (SVC amb rbf):





Generalment observant els resultats de les gràfiques podem dir que cap resultat és ideal ja que en cap gràfica de Precision-Recall curve cap classe s'aproxima al valor ideal, el qual seria precision = 1, recall = 1. En els nostres casos, per norma general obtenim alts valors de precision a costa de baixos valors de recall i viceversa.

Classification Report

La funció classification_report el que fa és mostrar les principals mètriques de classificació. Entre elles tenim, l'accuracy, el f1_score i més. Aquesta funció és molt útil si vols saber en general els resultats proporcionats per cadascun dels teus models. Nosaltres com abans ja havíem determinat quines mètriques eren les més rellevants per valorar els nostres models ens hem focalitzat en aquestes i no hem visualitzat totes aquestes mètriques ja que algunes eren irrellevants.

- Logistic Regression:

	precision	recall	f1-score	support
class 0	0.60	1.00	0.75	396
class 1	0.00	0.00	0.00	260
accuracy			0.60	656
macro avg	0.30	0.50	0.38	656
weighted avg	0.36	0.60	0.45	656

- Nearest K Neighbors:

Nearest K Neighbour

	precision	recall	f1-score	support
class 0	0.65	0.91	0.76	396
class 1	0.65	0.26	0.37	260
accuracy			0.65	656
macro avg	0.65	0.58	0.56	656
weighted avg	0.65	0.65	0.60	656

- SVM (SVM amb rbf):

SVM

	precision	recall	f1-score	support
class 0	0.67	0.96	0.79	396
class 1	0.83	0.28	0.42	260
accuracy			0.69	656
macro avg	0.75	0.62	0.61	656
weighted avg	0.74	0.69	0.65	656

1. 6. Hyperparameter Search

Exhaustive Grid Search:
Es basa en una recerca exhaustiva a través d'una quadrícula de valors de paràmetres especificada manualment de l'espai de hiperparàmetres. Aquest és un enfocament de força bruta perquè prova totes les combinacions de hiperparàmetres d'una quadrícula de valors de paràmetres. Després, per a cada combinació de hiperparàmetres, el model s'avalua utilitzant la cross-validation de kfold. Llavors, la combinació que ens dona la millor mètrica és la que retorna l'objecte que usarem

Randomized Parameter Optimization:
Fa una cerca de paràmetres de forma aleatòria a on les configuracions es mostren a partir d'una distribució de possibles valors i paràmetres. La cerca aleatòria prova una combinació aleatòria de hiperparàmetres en cada iteració i registra el rendiment del model. Després de diverses iteracions, retorna la mescla que va produir el millor resultat.

Parlant sobre cost computacional els dos mètodes són costosos. L'EGS és costos ja que el seu objectiu és provar amb totes les combinacions que s'han entrat al grid. Per altra banda el random és costos ja que agafa un número de vegades valors aleatoris de les dades que s'han entrat però si no és vol no s'agafen totes les combinacions. Això últim el que pot provocar és no trobar el més òptim per tant si no tenim un límit de temps predefinit la millor opció es provar totes les combinacions possibles i trobar el més òptim.

Existeixen mètodes de cerca més eficients com és el cas del **Bayesian optimization search**. Els dos mètodes vistos anteriorment són relativament ineficients perquè sovint avaluen moltes combinacions de hiperparàmetres inadequades. No tenen en compte els resultats de les iteracions anteriors en triar els següents hiperparàmetres. Per altre banda, la Bayesian optimization tracta la cerca dels hiperparàmetres òptims com un problema d'optimització. En triar la següent combinació de hiperparàmetres, aquest mètode considera els resultats de l'avaluació anterior. Després aplica una funció probabilística per a seleccionar la combinació que probablement produirà els millors resultats. Aquest mètode descobreix una combinació de hiperparàmetres bastant bona en relativament poques iteracions.

A continuació, posarem a prova els dos mètodes de cerca de paràmetres que s'han vist anteriorment per al model seleccionat, en el nostre cas, el SVM.
Inicialitzarem una variable amb un diccionari que contingui una llista amb tots els possibles paràmetres a provar. Finalment, els mètodes ens retornaran la combinació de paràmetres amb la que el model SVC obté els millors resultats.

```
# GridSearchCV
from sklearn.model_selection import GridSearchCV

parameters = {'kernel':('linear', 'rbf'), 'C':[0,0.2,0.5,0.75,1,1.25,1.5], 'gamma':[0.09,0.08,0.07,0.1]}
svc = svm.SVC()
clf = GridSearchCV(svc, parameters)
search = clf.fit(x_train,y_train)
print("GridSearchCV: ", search.best_params_)

# RandomizedSearchCV
from sklearn.model_selection import RandomizedSearchCV

clf = RandomizedSearchCV(svc, parameters, random_state=0)
search = clf.fit(x_train,y_train)
print("RandomizedSearchCV: ", search.best_params_)
```

Després de l'execució obtenim els següents resultats:

```
GridSearchCV:  {'C': 1.25, 'gamma': 0.1, 'kernel': 'rbf'}
RandomizedSearchCV:  {'kernel': 'rbf', 'gamma': 0.09, 'C': 1}
```

Per verificar els resultats, executarem manualment el nostre model i mostrarem per pantalla la precisió del mateix.

GridSearchCV:

```
SVM with rbf kernel Score:  0.6966463414634146
SVM with rbf kernel Cross Val Score:  0.6599222544632911

199 Errores de clasificacion de un total de 656
457 Aciertos de clasificacion de un total de 656

Confusion matrix:
[[376  20]
 [179  81]]

SVM with rbf kernel F1 Score: 0.6966463414634146
```

RandomizedSearchCV:

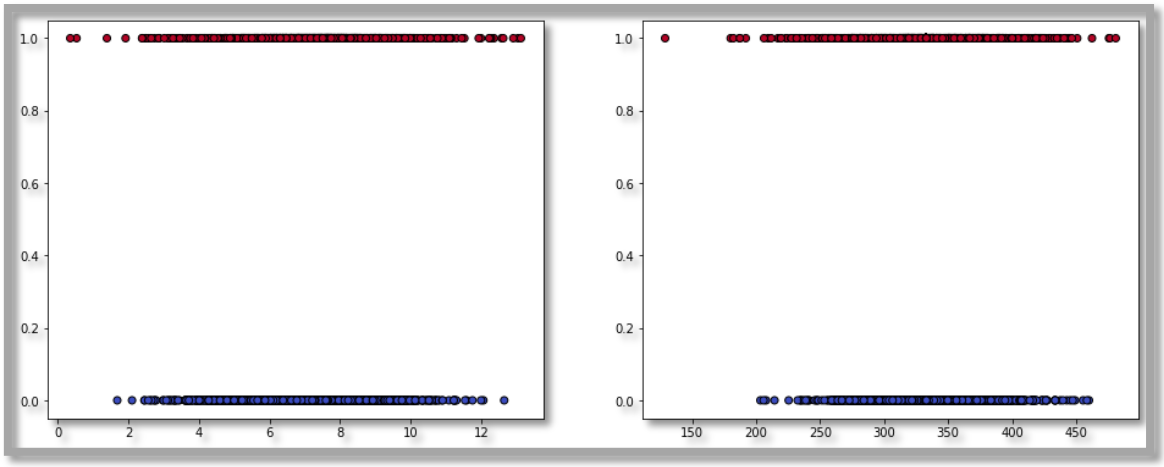
```
SVM with rbf kernel Score:  0.6905487804878049
SVM with rbf kernel Cross Val Score:  0.6591594779568074

203 Errores de clasificacion de un total de 656
453 Aciertos de clasificacion de un total de 656

Confusion matrix:
[[379  17]
 [186  74]]

SVM with rbf kernel F1 Score: 0.6905487804878049
```

El mètode que ens retorna els paràmetres amb els quals el nostre model obté els millors resultats es el GridSearchCV. Podem observar que para **C = 1.25**, **gamma = 0.1** i **kernel = rbf** el model realitza 457 encerts, aquest és el millor resultat obtingut fins ara.

2. Apartat (A): Comparativa de models

En aquest apartat l'objectiu serà aprendre a comparar el rendiment obtingut amb els diferents models per a la nostra base de dades.
A la primera part d'aquest apartat, es compararà els resultats de precisió obtinguts per als models de regressió logística i SVM amb el kernel de rbf. Els models s'executaran per a diferents particions de dades, és a dir, el codi mostra la precisió de cada model per a un conjunt de 50%, 70% i 80% de dades.

En les següents gràfiques es pot veure com estan classificades les nostres dades per als primers dos atributs ("ph" i "Hardness").

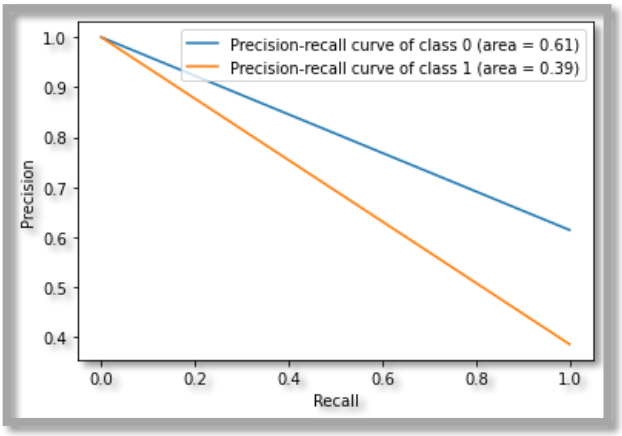


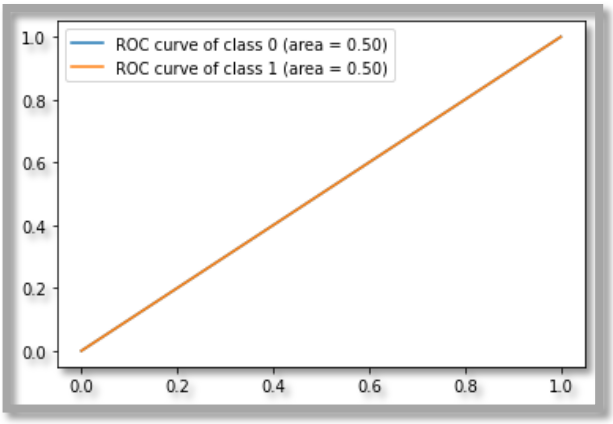
```
Correct classification Logistic  0.5 % of the data:  0.6153846153846154
Correct classification SVM       0.5 % of the data:  0.5634920634920635
Correct classification Logistic  0.7 % of the data:  0.5940996948118006
Correct classification SVM       0.7 % of the data:  0.5534079348931842
Correct classification Logistic  0.8 % of the data:  0.6448170731707317
Correct classification SVM       0.8 % of the data:  0.573170731707317
```

Com podem veure, quant estem calculant els resultats per només el 50% de la base de dades, per molt poc més la regressió logística és millor que el SVM. Això succeeix per les tres particions.

Per altra banda, aquests valors de precisió segueixen sent molt baixos per models de classificacions. Això pot ser degut al desbalancejament del nostre atribut target, la potability, i perquè la correlació entre atributs independents és ínfima i el nombre d'aquests és elevat. També un altre motiu pot ser la falta de normalització de les dades.

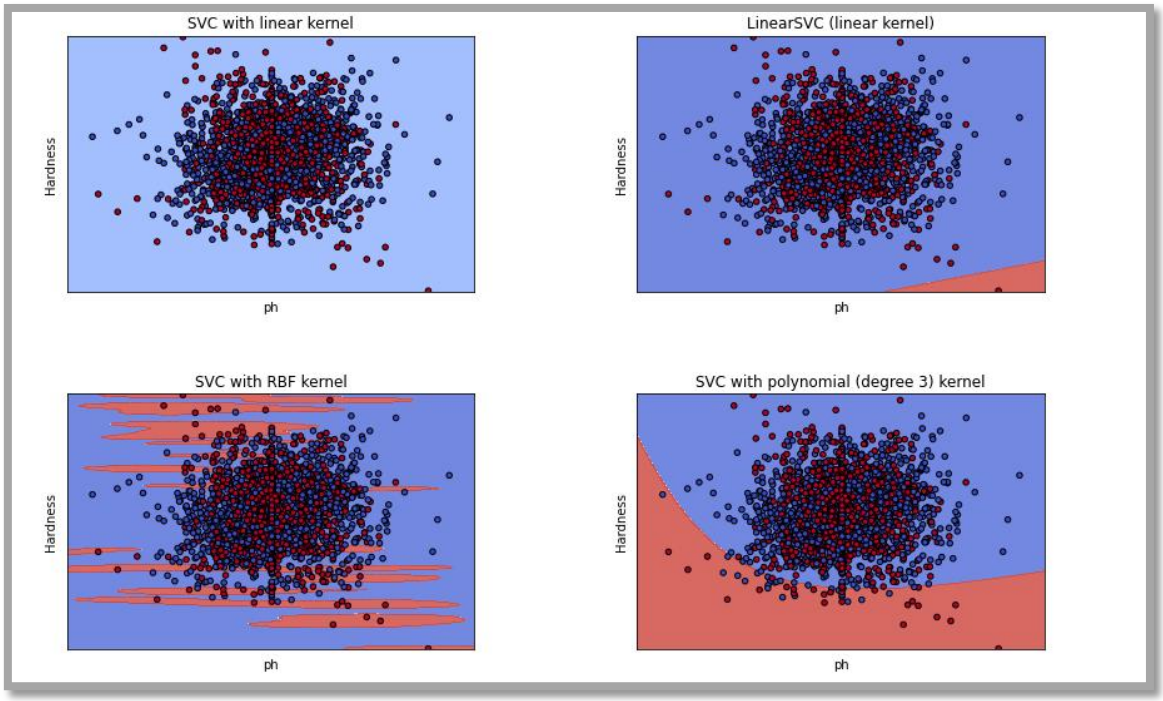
Arà visualitzarem les gràfiques corresponents a la Precision-recall curve i a la ROC curve.





A la gràfica de Precision-recall l'ideal seria tenir un resultat que s'apropés a la cantonada superior dreta, per així tenir una precision i un recall d'1.
Però no és el cas. Era d'esperar que no s'acostés a aquest valor ideal ja que en l'anterior prova ja es veia el baix valor de score dels models. Podem observar com quant major precisió tenim, menor recall obtindrem i viceversa. Això pot passar a causa de tenir una base de dades desbalancejada ja que per exemple podem dir que tots els casos son d'aigua no potable i tenir certament un valor de precisió alt però el recall serà molt baix ja que no estarem identificant verdaders positius. També al estar desbalancejada i per tant tenir més casos d'aigua no potable, la corba de Precision-Recall d'aigua no potable dona millors resultats que la corba d'aigua potable.

A la gràfica de ROC l'ideal seria tenir un resultat que s'apropés a la cantonada superior esquerra (punt on s'obté la màxima precisió del model), per així tenir el menor nombre de falsos positius i el major nombre de verdaders positius. Com podem observar la nostre gràfica es troba lluny d'aquest punt ideal ja que detecta molts falsos positius.
L'àrea inferior a la corba es el AUC. Ens indica en quina mesura el model és capaç de distingir entre les classes. Com més alt sigui el AUC, millor serà el model per a predir la probabilitat de la classe SÍ més alta que la probabilitat de la classe NO. Com es pot observar, l'àrea de la nostra AUC no es bona perquè la ROC curve no s'apropa al punt ideal.



A la primera imatge corresponent al model "SVC with linear kernel", igual que en el model de regressió logística vist a l'apartat b), no es capaç de detectar cap cas d'aigua potable i per això el fons de la imatge només es de color blau. La primera imatge de la dreta només és capaç de detectar un cas d'aigua potable utilitzant "LinearSVC". En canvi els dos d'abaix si que són capaços de detectar alguns cassos d'aigua potable però d'una manera molt ineficient. Tots els punts es troben molt junts i pel model es impossible fer una distinció entre aigua potable i aigua no potable.