Tasca M7 T01

Exercici 1

Crea almenys dos models de classificació diferents per intentar predir el millor les classes de l'arxiu adjunt.

In [729…

```
# Tratamiento de datos
# ==============================================================================
import pandas as pd
import numpy as np

# Gráficos
# ==============================================================================
import matplotlib.pyplot as plt
from matplotlib import style
import matplotlib.ticker as ticker
import seaborn as sns
from pprint import pprint

# Preprocesado y análisis
# ==============================================================================
#import statsmodels.api as sm
#import pingouin as pg
from scipy import stats
import random as rd
from imblearn.over_sampling import SMOTE

# Preprocesado y modelado
# ==============================================================================
from sklearn.datasets import load_boston
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import confusion_matrix, precision_score, recall_score, accurac
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score

from sklearn import metrics
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ParameterGrid
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn.tree import export_text
from sklearn.inspection import permutation_importance
import multiprocessing
from sklearn import neighbors, datasets, preprocessing
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import KFold
import statsmodels.api as sm
from sklearn.model_selection import cross_val_predict
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import VarianceThreshold


# Test Estadísticos
# ==============================================================================
from scipy.stats import shapiro
from scipy.stats import ttest_ind
from scipy.stats import ttest_rel

# Ajuste de distribuciones
# ==============================================================================
from scipy import stats
import inspect
from statsmodels.distributions.empirical_distribution import ECDF


# Configuración matplotlib
# ==============================================================================
plt.style.use('ggplot')
from statsmodels.graphics.gofplots import qqplot
from matplotlib import pyplot

# Configuración warnings
# ==============================================================================
import warnings
warnings.filterwarnings('ignore')
```

A) Data Frame

In [730…

```
data= pd.read_table(r"C:\Users\hecto\OneDrive\Documentos\IT Data Science\Sprint7_DS\
data_original =data
data_original.head(5)
```

Out[730…

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

Relevant Information:

- These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituentsfound in each of the three types of wines. 1) Alcohol 2) Malic acid 3) Ash 4) Alcalinity of ash
  5) Magnesium 6) Total phenols 7) Flavanoids 8) Nonflavanoid phenols 9) Proanthocyanins 10)Color intensity 11)Hue 12)OD280/OD315 of diluted wines 13)Proline

In [731…

```
data.columns =['Class','Alcohol', 'Malic Acid', 'Ash', 'Alcalinity of Ash', 'Magnesi
```

```
data.head()
```

Out[731…

	Class	Alcohol	Malic Acid	Ash	Alcalinity of Ash	Magnesium	Total Phenols	Flavanoids	Nonflavanoid Phenols	Proantho
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	

In [732…

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Class                         178 non-null    int64  
 1   Alcohol                       178 non-null    float64
 2   Malic Acid                    178 non-null    float64
 3   Ash                           178 non-null    float64
 4   Alcalinity of Ash             178 non-null    float64
 5   Magnesium                     178 non-null    int64  
 6   Total Phenols                 178 non-null    float64
 7   Flavanoids                    178 non-null    float64
 8   Nonflavanoid Phenols          178 non-null    float64
 9   Proanthocyanins               178 non-null    float64
 10  Color intensity               178 non-null    float64
 11  Hue                           178 non-null    float64
 12  OD280/OD315 of Diluted Wines  178 non-null    float64
 13  Proline                       178 non-null    int64  
dtypes: float64(11), int64(3)
memory usage: 19.6 KB
```

B) Data Describe

In [733…

```
data.describe().transpose()
```

Out[733…

	count	mean	std	min	25%	50%	75%	max
Class	178.0	1.938202	0.775035	1.00	1.0000	2.000	3.0000	3.00
Alcohol	178.0	13.000618	0.811827	11.03	12.3625	13.050	13.6775	14.83
Malic Acid	178.0	2.336348	1.117146	0.74	1.6025	1.865	3.0825	5.80
Ash	178.0	2.366517	0.274344	1.36	2.2100	2.360	2.5575	3.23
Alcalinity of Ash	178.0	19.494944	3.339564	10.60	17.2000	19.500	21.5000	30.00
Magnesium	178.0	99.741573	14.282484	70.00	88.0000	98.000	107.0000	162.00
Total Phenols	178.0	2.295112	0.625851	0.98	1.7425	2.355	2.8000	3.88
Flavanoids	178.0	2.029270	0.998859	0.34	1.2050	2.135	2.8750	5.08

	count	mean	std	min	25%	50%	75%	max
Nonflavanoid Phenols	178.0	0.361854	0.124453	0.13	0.2700	0.340	0.4375	0.66
Proanthocyanins	178.0	1.590899	0.572359	0.41	1.2500	1.555	1.9500	3.58
Color intensity	178.0	5.058090	2.318286	1.28	3.2200	4.690	6.2000	13.00
Hue	178.0	0.957449	0.228572	0.48	0.7825	0.965	1.1200	1.71
OD280/OD315 of Diluted Wines	178.0	2.611685	0.709990	1.27	1.9375	2.780	3.1700	4.00
Proline	178.0	746.893258	314.907474	278.00	500.5000	673.500	985.0000	1680.00

C) Identificación Valores nulos

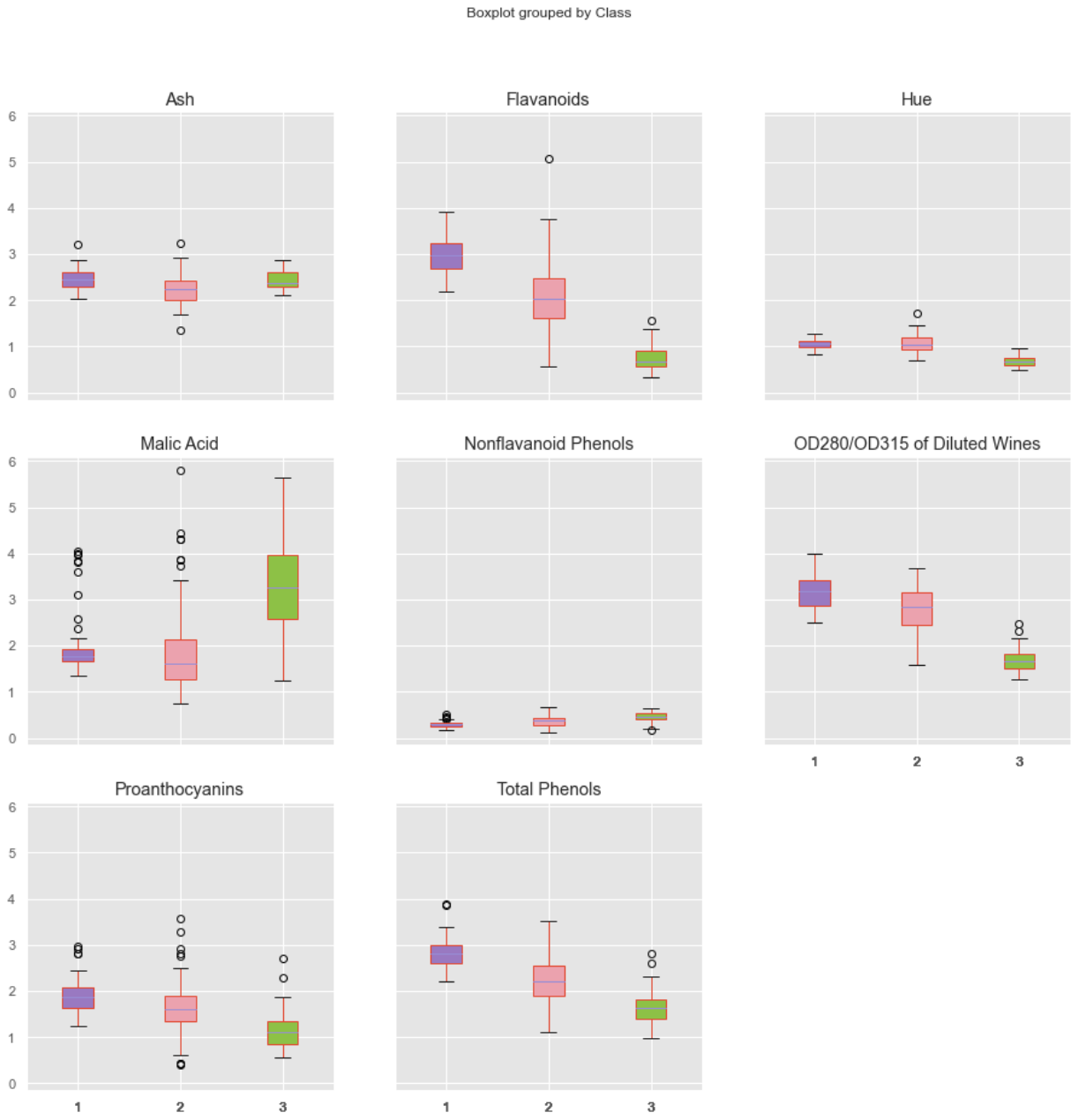
In [734…
```
data.isna().sum()
```

Out[734…
```
Class                         0
Alcohol                       0
Malic Acid                    0
Ash                           0
Alcalinity of Ash             0
Magnesium                     0
Total Phenols                 0
Flavanoids                    0
Nonflavanoid Phenols          0
Proanthocyanins               0
Color intensity               0
Hue                           0
OD280/OD315 of Diluted Wines  0
Proline                       0
dtype: int64
```

D) Gráficos de las variables explicativas

D.1) Gráficos Boxplot de las variables explicativas

In [735…
```
data_sin=data.drop(columns=['Magnesium','Proline','Alcalinity of Ash', 'Alcohol','Co
bp_dict = data_sin.boxplot(
by="Class",layout=(3,3),figsize=(15,15),
return_type='both',
patch_artist = True,
)
colors =["#9979c1", "#eba2ae", "#8dc146", '#ffa455', '#ffe358',"#9979c1", "#eba2ae"]

for row_key, (ax,row) in bp_dict.iteritems():
    ax.set_xlabel('')

    for i,box in enumerate(row['boxes']):
        box.set_facecolor(colors[i])

plt.show()
```

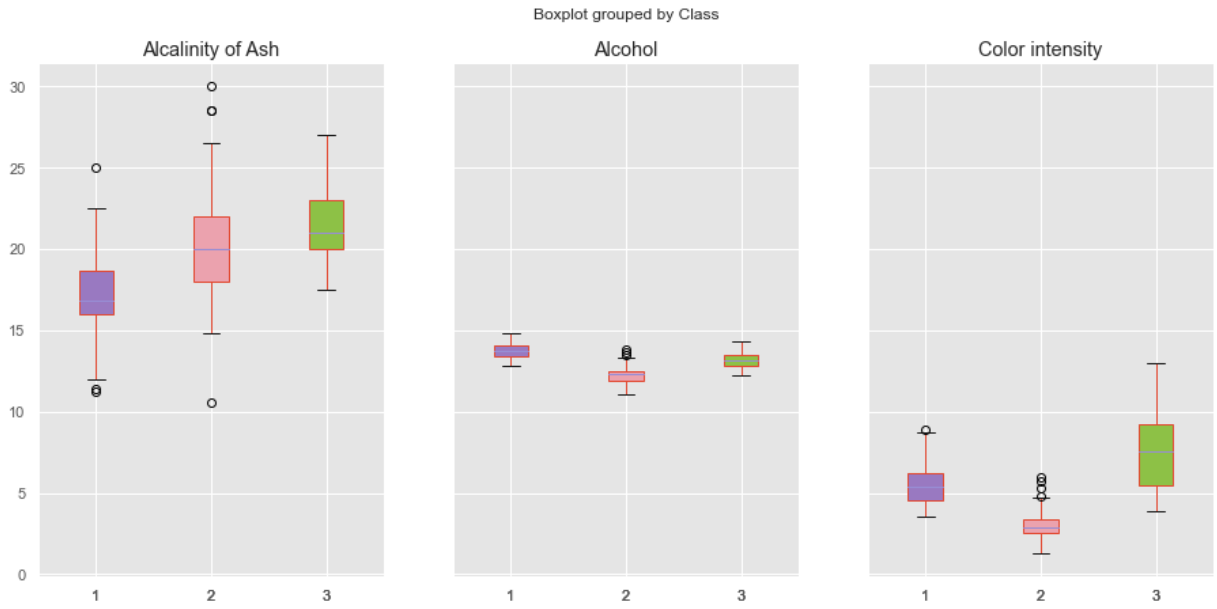


- Ash: Es una variable que tiene una distribución homogénea entre las clases 1,2 y 3 y no tiene outliers.
- Flavanoids: Tiene medias diferentes entre clases y no tiene outliers significativos.
- Hue: :a media difiere entre clases y solo tiene outliers en la clae 2.
- Malic Acid: Tiene gran dispersoón de midia y desviación estándar entre clases y es la variable con más outliers en dos de las claes.
- Nonflavanoid Phenols: Todas las clases tienen medias y desviaciones estándars homogéneas entre clases.
- OD280/OD315 of Diluted Wines: existe una gran diferencia de medias entre diferentes clases
- Proanthocyanins: es otra de las variables con más ouliers en todas las clases.
- Total Phenols: Tiene diferencias de medias entre clases y outliers en dos de las clases.

In [736…

```
data_sin=data.drop(columns=['Magnesium','Proline','Ash','Flavanoids','Hue', 'Malic A
bp_dict = data_sin.boxplot(
by="Class",layout=(1,3),figsize=(15,7),
return_type='both',
patch_artist = True,
```

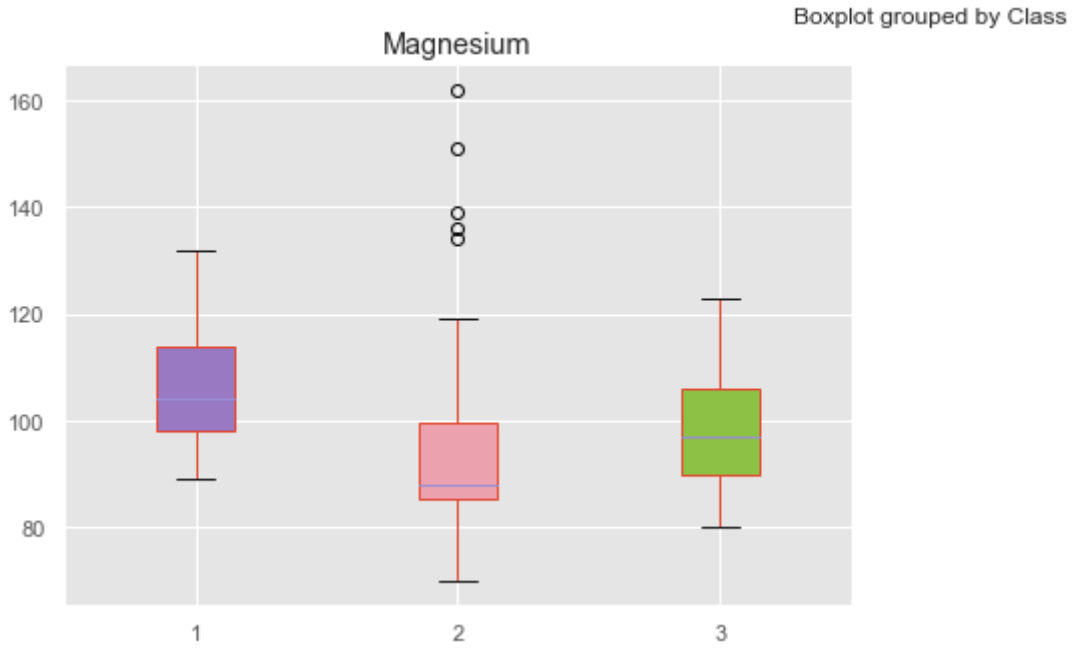
```
)
colors =["#9979c1", "#eba2ae", "#8dc146", '#ffa455', '#ffe358',"#9979c1", "#eba2ae"]

for row_key, (ax,row) in bp_dict.iteritems():
    ax.set_xlabel('')

    for i,box in enumerate(row['boxes']):
        box.set_facecolor(colors[i])

plt.show()
```



- Alcalinity of Ash: La variable tiene diferencias entre medias y outliers en la clase 1 y 2.
- Alcohol: Las tres clases tiene un rango homogéneo y solo tiene outliers la clase 2
- Color intensity: tiene una gran diferencia entre medias y desviación estándar y muestra outliers en la clase 1 y 2.

In [737…

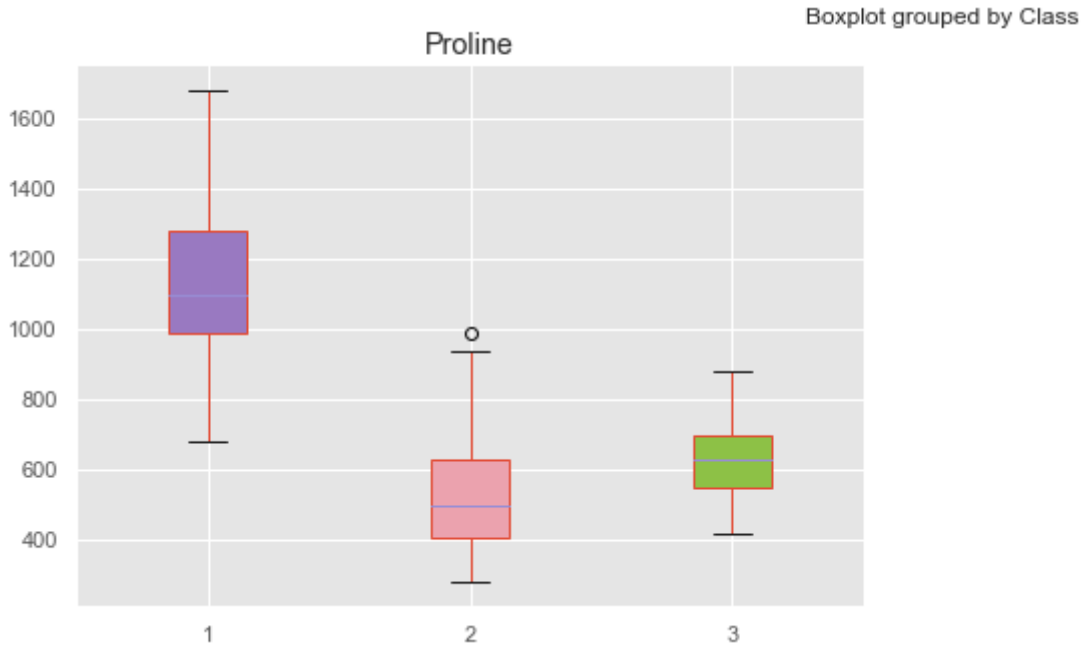
```
data_sin=data[['Magnesium',"Class"]]
bp_dict = data_sin.boxplot(
by="Class",layout=(1,2),figsize=(15,5),
return_type='both',
patch_artist = True,
)
colors =["#9979c1", "#eba2ae", "#8dc146", '#ffa455', '#ffe358',"#9979c1", "#eba2ae"]

for row_key, (ax,row) in bp_dict.iteritems():
    ax.set_xlabel('')

    for i,box in enumerate(row['boxes']):
        box.set_facecolor(colors[i])

plt.show()
```



- Magnesium : Presenta diferencias en medias y desviación estándar, entre clases y tiene outliers en la clase 2.

In [738…

```
data_sin=data[['Proline',"Class"]]
bp_dict = data_sin.boxplot(
by="Class",layout=(1,2),figsize=(15,5),
return_type='both',
patch_artist = True,
)
colors =["#9979c1", "#eba2ae", "#8dc146", '#ffa455', '#ffe358',"#9979c1", "#eba2ae"]

for row_key, (ax,row) in bp_dict.iteritems():
    ax.set_xlabel('')

    for i,box in enumerate(row['boxes']):
        box.set_facecolor(colors[i])

plt.show()
```

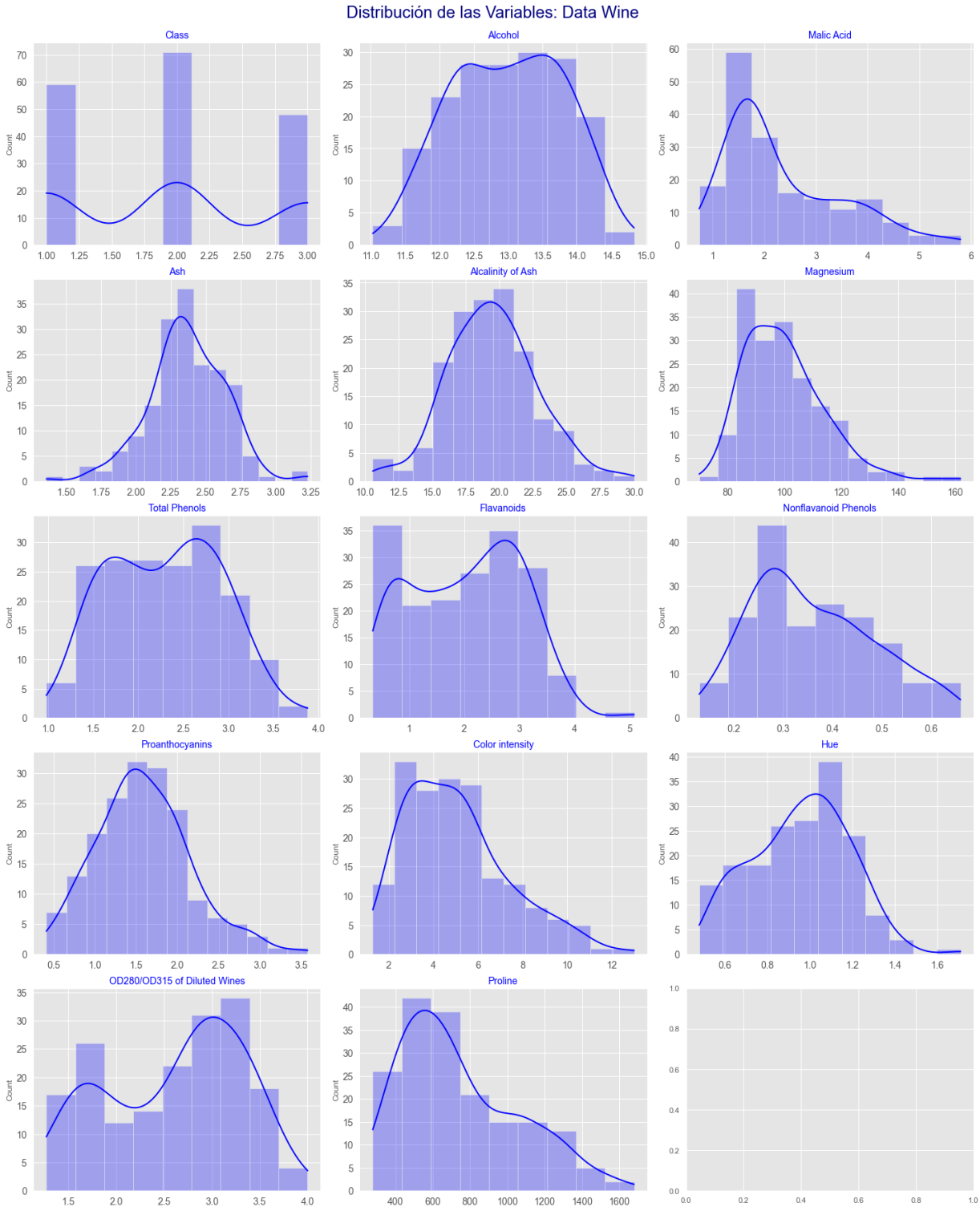


- Proline : es la variable con mayor rango y presenta diferencias de medias y desviación estándar entre las tres clases.

D.2) # Distribución de cada variable explicativa

In [739…

```
# Distribución gráfica de cada variable explicativa
# ================================================

fig, axes = plt.subplots(ncols=3, nrows=5, figsize=(20, 25))
axes = axes.flat
columnas_numeric = data.select_dtypes(include=['float64', 'int','uint8']).columns

for i, colum in enumerate(columnas_numeric):
    sns.histplot(data = data, x= colum, stat= "count",label="data", color="blue", kd
   # sns.histplot(data = X_test, x= colum, stat= "count", label= "X_test", color="cy
    axes[i].set_title(colum, fontsize = 14, fontweight = "ultralight", color="blue")
    axes[i].tick_params(labelsize = 14)
    axes[i].set_xlabel("")

fig.tight_layout()
plt.subplots_adjust(top = 0.95)
plt.suptitle('Distribución de las Variables: Data Wine', fontsize = 24, color="navy"
plt.savefig("Gafico1a_Dist_Variables_X_train_test.png")
```



- Sólo dos de las variables explicativas parecen seguir una distribución normal, "Ash" y "Alcalinity of Ash", para verificarlo vamos a relaizar un test de normatidad de todas las variables:

In [740…

```
# create a function that checks if the distribution is normal:
def check_normal_distribution(data):

    for i in data[features]:
        stat, p_value_norm = shapiro(data[i])
        print(f'Results for {i}:')
        print('stat=%.3f, p=%.3f' % (stat, p_value_norm))

        if p_value_norm < 0.05 :
            print("Reject null hypothesis at 95% Significance Level  >> The data is
```

```
            print('-----------------------------------------------------------------
        else:
            print("Fail to reject null hypothesis at 95€ Significance Level  >> The
            print('-----------------------------------------------------------------
```

In [741…

```
features =['Class','Alcohol', 'Malic Acid', 'Ash', 'Alcalinity of Ash', 'Magnesium',
check_normal_distribution(data[features])
```

```
Results for Class:
stat=0.804, p=0.000
Reject null hypothesis at 95% Significance Level  >> The data is not normally distri
buted
-------------------------------------------------------------------------------------
----------
Results for Alcohol:
stat=0.982, p=0.020
Reject null hypothesis at 95% Significance Level  >> The data is not normally distri
buted
-------------------------------------------------------------------------------------
----------
Results for Malic Acid:
stat=0.889, p=0.000
Reject null hypothesis at 95% Significance Level  >> The data is not normally distri
buted
-------------------------------------------------------------------------------------
----------
Results for Ash:
stat=0.984, p=0.039
Reject null hypothesis at 95% Significance Level  >> The data is not normally distri
buted
-------------------------------------------------------------------------------------
----------
Results for Alcalinity of Ash:
stat=0.990, p=0.264
Fail to reject null hypothesis at 95€ Significance Level  >> The data is normally di
stributed
-------------------------------------------------------------------------------------
----------
Results for Magnesium:
stat=0.938, p=0.000
Reject null hypothesis at 95% Significance Level  >> The data is not normally distri
buted
-------------------------------------------------------------------------------------
----------
Results for Total Phenols:
stat=0.977, p=0.004
Reject null hypothesis at 95% Significance Level  >> The data is not normally distri
buted
-------------------------------------------------------------------------------------
----------
Results for Flavanoids:
stat=0.955, p=0.000
Reject null hypothesis at 95% Significance Level  >> The data is not normally distri
buted
-------------------------------------------------------------------------------------
----------
Results for Nonflavanoid Phenols:
stat=0.963, p=0.000
Reject null hypothesis at 95% Significance Level  >> The data is not normally distri
buted
-------------------------------------------------------------------------------------
----------
```

```
Results for Proanthocyanins:
stat=0.981, p=0.014
Reject null hypothesis at 95% Significance Level  >> The data is not normally distri
buted
----------------------------------------------------------------------------------
----------
Results for Color intensity:
stat=0.940, p=0.000
Reject null hypothesis at 95% Significance Level  >> The data is not normally distri
buted
----------------------------------------------------------------------------------
----------
Results for Hue:
stat=0.981, p=0.017
Reject null hypothesis at 95% Significance Level  >> The data is not normally distri
buted
----------------------------------------------------------------------------------
----------
Results for OD280/OD315 of Diluted Wines:
stat=0.945, p=0.000
Reject null hypothesis at 95% Significance Level  >> The data is not normally distri
buted
----------------------------------------------------------------------------------
----------
Results for Proline:
stat=0.931, p=0.000
Reject null hypothesis at 95% Significance Level  >> The data is not normally distri
buted
----------------------------------------------------------------------------------
----------
```

- Como conclusión del test, solo la variable "Alcalinity of Ash" se distribuye normalmente.

D.3) Variable objetivo : "Class"

In [743…

```
y= data.groupby("Class")[["Class"]].count()
y
```

Out[743…

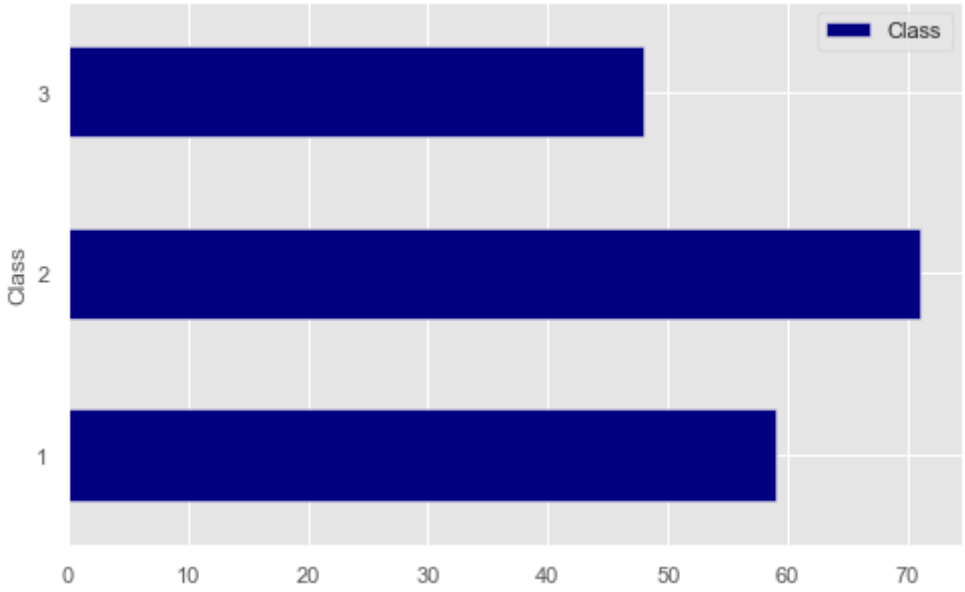
	Class
Class	
1	59
2	71
3	48

In [744…

```
y.plot(kind="barh", color="navy")
```

Out[744…

```
<AxesSubplot:ylabel='Class'>
```



- Existe un mayor número de elementos de la clase 2, lo que podria tener incidencia a la hora de extraer las muestas para el entranamiento de los modelos.

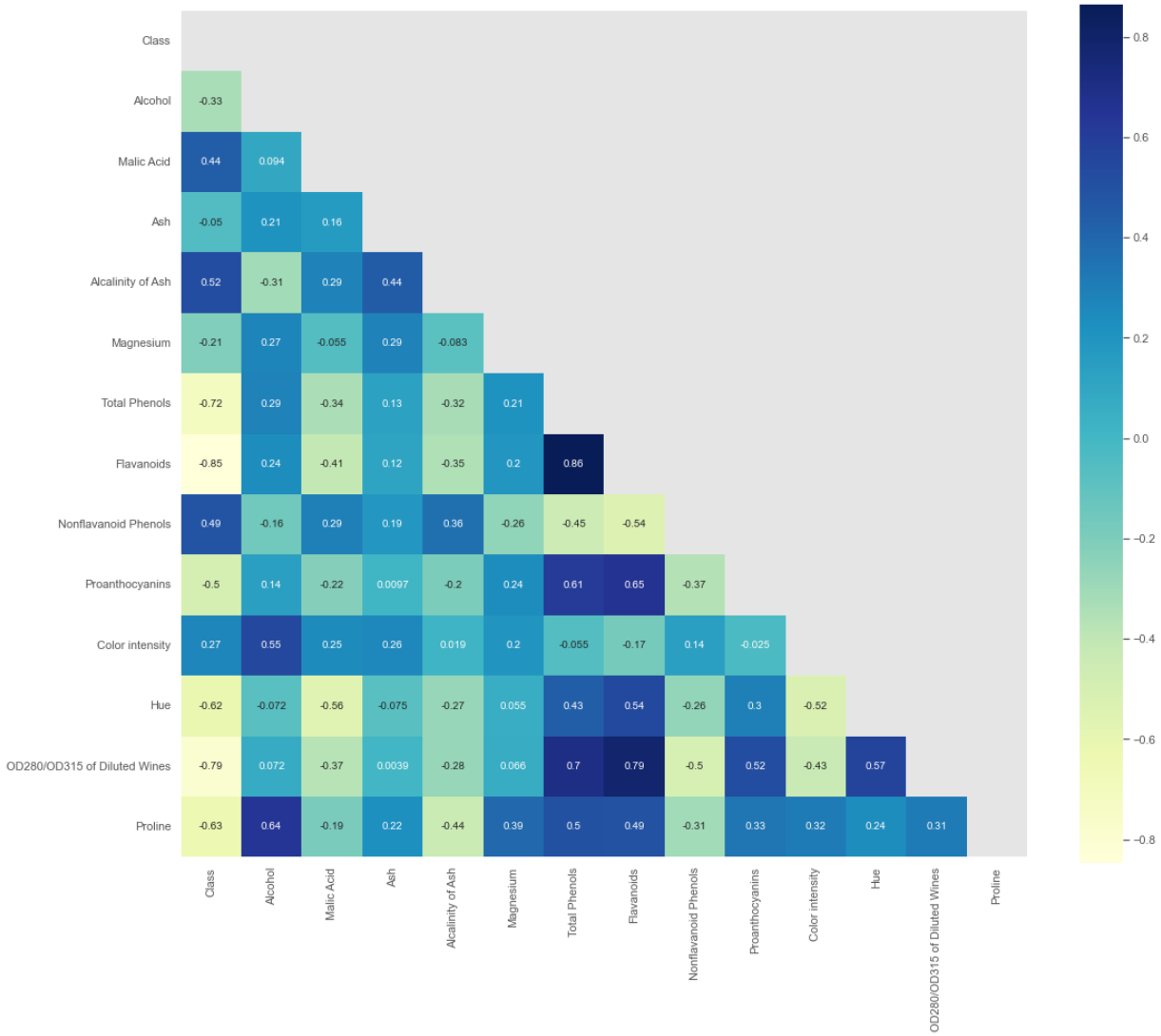
E) Correlación entre las variables del Data Frame

In [745…

```
plt.figure(figsize=(18,15))
upp_mat = np.triu(data.corr())
sns.heatmap(data.corr(),cmap="YlGnBu", square = True,annot=True, mask = upp_mat)
```

Out[745…

```
<AxesSubplot:>
```



- Entre la variable objetivo "Class" y las variables explicativas, existe una correlación que supera el (+-) 0,5 con la mayoría de variables excepto Color Intensity, Magnesium, Ash y Alcohol.
- También se producen correlaciones elevadas entre las variables explitativas, superiores en varios casos al (+-) 0,5 por lo que pueden darse problemas de multicolinealidad.

F) Selección de X_train - X_test

In [746…

```
X_train, X_test, y_train, y_test = train_test_split(data.drop(columns="Class"), data
print(f'X train shape {X_train.shape} \nX test shape {X_test.shape}')
```

```
X train shape (119, 13)
X test shape (59, 13)
```

F.1) X_train describe

In [747…

```
X_train.describe().transpose()
```

Out[747…

	count	mean	std	min	25%	50%	75%	max
Alcohol	119.0	13.034622	0.810040	11.03	12.370	13.05	13.715	14.39
Malic Acid	119.0	2.260084	1.110269	0.74	1.520	1.78	2.900	5.80
Ash	119.0	2.334790	0.273619	1.36	2.200	2.32	2.500	3.22

	count	mean	std	min	25%	50%	75%	max
Alcalinity of Ash	119.0	19.099160	3.146117	10.60	16.800	19.00	21.000	30.00
Magnesium	119.0	99.596639	14.321561	70.00	88.000	98.00	107.500	151.00
Total Phenols	119.0	2.308571	0.634066	0.98	1.745	2.41	2.800	3.85
Flavanoids	119.0	2.110588	0.948818	0.34	1.305	2.19	2.920	3.93
Nonflavanoid Phenols	119.0	0.347563	0.122467	0.13	0.260	0.32	0.430	0.66
Proanthocyanins	119.0	1.652017	0.550935	0.42	1.350	1.62	1.980	2.96
Color intensity	119.0	4.999244	2.288184	1.28	3.190	4.68	6.200	13.00
Hue	119.0	0.962571	0.227941	0.56	0.790	0.98	1.115	1.71
OD280/OD315 of Diluted Wines	119.0	2.643613	0.707503	1.29	2.035	2.83	3.190	4.00
Proline	119.0	755.142857	326.409000	278.00	497.500	680.00	1027.500	1680.00

F.1) X_train describe

In [748…

```
X_test.describe().transpose()
```

Out[748…

	count	mean	std	min	25%	50%	75%	max
Alcohol	59.0	12.932034	0.818025	11.45	12.310	12.85	13.510	14.83
Malic Acid	59.0	2.490169	1.124587	0.98	1.655	2.08	3.255	5.51
Ash	59.0	2.430508	0.266720	1.70	2.245	2.42	2.610	3.23
Alcalinity of Ash	59.0	20.293220	3.595537	11.40	18.000	19.50	22.500	28.50
Magnesium	59.0	100.033898	14.321392	78.00	89.000	97.00	107.000	162.00
Total Phenols	59.0	2.267966	0.613404	1.35	1.755	2.20	2.700	3.88
Flavanoids	59.0	1.865254	1.082716	0.47	0.770	1.84	2.665	5.08
Nonflavanoid Phenols	59.0	0.390678	0.124456	0.17	0.285	0.40	0.475	0.63
Proanthocyanins	59.0	1.467627	0.599233	0.41	1.035	1.40	1.760	3.58
Color intensity	59.0	5.176780	2.393280	2.08	3.260	4.92	6.150	11.75
Hue	59.0	0.947119	0.231450	0.48	0.750	0.93	1.125	1.36
OD280/OD315 of Diluted Wines	59.0	2.547288	0.716687	1.27	1.840	2.72	3.145	3.82
Proline	59.0	730.254237	292.315995	290.00	517.500	640.00	880.000	1515.00

G) Comparación de la distribuciones de X_train y X_test

In [749…

```
# create a function that checks if the distribution of X_train and X_test are equals
def check_same_distribution(data1,data2):

    for i in data[features]:
        stat, p_value_norm = ttest_ind(data1[i], data2[i])
        # stat, p_value_norm = shapiro(data[i])
        print(f'Results for {i}:')
        print('stat=%.3f, p=%.3f' % (stat, p_value_norm))
```

```
        if p_value_norm < 0.05 :
            print("Reject null hypothesis at 95% Significance Level  >>  Probably di
            print('-------------------------------------------------------------------
        else:
            print("Fail to reject null hypothesis at 95€ Significance Level  >> Prob
            print('-------------------------------------------------------------------
```

In [750…

```
features =['Alcohol', 'Malic Acid', 'Ash', 'Alcalinity of Ash', 'Magnesium', 'Total
check_same_distribution(X_train[features],X_test[features])
```

```
Results for Alcohol:
stat=0.793, p=0.429
Fail to reject null hypothesis at 95€ Significance Level  >> Probably the same distr
ibution
--------------------------------------------------------------------------------
----------
Results for Malic Acid:
stat=-1.296, p=0.197
Fail to reject null hypothesis at 95€ Significance Level  >> Probably the same distr
ibution
--------------------------------------------------------------------------------
----------
Results for Ash:
stat=-2.215, p=0.028
Reject null hypothesis at 95% Significance Level  >>  Probably different distributio
ns
--------------------------------------------------------------------------------
----------
Results for Alcalinity of Ash:
stat=-2.272, p=0.024
Reject null hypothesis at 95% Significance Level  >>  Probably different distributio
ns
--------------------------------------------------------------------------------
----------
Results for Magnesium:
stat=-0.192, p=0.848
Fail to reject null hypothesis at 95€ Significance Level  >> Probably the same distr
ibution
--------------------------------------------------------------------------------
----------
Results for Total Phenols:
stat=0.407, p=0.685
Fail to reject null hypothesis at 95€ Significance Level  >> Probably the same distr
ibution
--------------------------------------------------------------------------------
----------
Results for Flavanoids:
stat=1.549, p=0.123
Fail to reject null hypothesis at 95€ Significance Level  >> Probably the same distr
ibution
--------------------------------------------------------------------------------
----------
Results for Nonflavanoid Phenols:
stat=-2.199, p=0.029
Reject null hypothesis at 95% Significance Level  >>  Probably different distributio
ns
--------------------------------------------------------------------------------
----------
Results for Proanthocyanins:
stat=2.041, p=0.043
Reject null hypothesis at 95% Significance Level  >>  Probably different distributio
```

```
ns
----------------------------------------------------------------------------------
----------
Results for Color intensity:
stat=-0.480, p=0.632
Fail to reject null hypothesis at 95€ Significance Level  >> Probably the same distr
ibution
----------------------------------------------------------------------------------
----------
Results for Hue:
stat=0.424, p=0.672
Fail to reject null hypothesis at 95€ Significance Level  >> Probably the same distr
ibution
----------------------------------------------------------------------------------
----------
Results for OD280/OD315 of Diluted Wines:
stat=0.851, p=0.396
Fail to reject null hypothesis at 95€ Significance Level  >> Probably the same distr
ibution
----------------------------------------------------------------------------------
----------
Results for Proline:
stat=0.495, p=0.621
Fail to reject null hypothesis at 95€ Significance Level  >> Probably the same distr
ibution
----------------------------------------------------------------------------------
----------
```

H) Creación de los Modelos

H.1) El algoritmo K-vecinos más cercanos (KNN)

- El algoritmo K-vecinos más cercanos (KNN) es un tipo de algoritmo de aprendizaje automático supervisado.
- KNN es extremadamente fácil de implementar en su forma más básica y, sin embargo, realiza tareas de clasificación bastante complejas.
- Es un algoritmo de aprendizaje que no tiene una fase de entrenamiento especializada, utiliza todos los datos para el entrenamiento mientras clasifica un nuevo punto de datos o instancia.
- KNN es un algoritmo de aprendizaje no paramétrico, lo que significa que no asume nada sobre los datos subyacentes. Esta es una característica extremadamente útil ya que la mayoría de los datos del mundo real no siguen ningún supuesto teórico, por ejemplo, separabilidad lineal, distribución uniforme, etc.

In [751…

```
## -- K-Nearest Neighbors Algorithm -- ##
# Creación del modelo
kn_class = KNeighborsClassifier(n_neighbors=5)
# Train
kn_class.fit(X_train, y_train)
# Predicción
y_pred_kn_class = kn_class.predict(X_test)
print(f'K-Nearest Neighbors predictions: {y_pred_kn_class[:7]}')
```

```
K-Nearest Neighbors predictions: [2 2 3 2 2 3 1]
```

H.2) Las máquinas de vectores de soporte - SMV (Support Vector Machines)

- Las máquinas de vectores de soporte se consideran un enfoque de clasificación, pero se

pueden emplear en ambos tipos de problemas de clasificación y regresión.
- Puede manejar fácilmente múltiples variables continuas y categóricas.
- SVM construye un hiperplano en un espacio multidimensional para separar diferentes clases, genera un hiperplano óptimo de forma iterativa, que se utiliza para minimizar un error. La idea central de SVM es encontrar un hiperplano marginal máximo (MMH) que divida mejor el conjunto de datos en clases.

In [752…

```
## --Support Vector Machines  SVM -- ##
svc_class = svm.SVC(kernel='linear',random_state=123) # Linear Kernel
svc_class.fit(X_train, y_train)
y_pred_svc_class = svc_class.predict(X_test)
print(f'SVM predictions: {y_pred_svc_class[:7]}')
```

```
SVM predictions: [3 2 3 2 2 3 1]
```

H.3) Algoritmo XGBoost (Extreme Gradient Boosting)

- XGBoost pertenece a una familia de algoritmos de impulso y utiliza el marco de impulso de gradiente (GBM) en su núcleo. Es una biblioteca de aumento de gradiente distribuido optimizada.
- la idea básica de los algoritmos de impulso es construir un modelo débil, sacar conclusiones sobre la importancia y los parámetros de varias características, y luego usar esas conclusiones para construir un modelo nuevo y más fuerte y capitalizar el error de clasificación errónea del modelo anterior e intentar reducirlo.
- La base de XGBoost son los conjuntos de árboles de árboles de clasificación y regresión (CART). Los árboles se generan uno tras otro con el objetivo de reducir la tasa de clasificación errónea en iteraciones posteriores. Cada árbol otorga una puntuación de predicción diferente según los datos que ve y las puntuaciones de cada árbol individual se suman para obtener la puntuación final.

In [753…

```
## --XGBClassifier  -- ##

## -- adaptación de la variavle objetivo a [0,1,2]--#
le = LabelEncoder()
y_train_xg = le.fit_transform(y_train)
y_test_xg=le.fit_transform(y_test)

from xgboost import XGBClassifier
#xg_class = XGBClassifier(eval_metric='mlogloss',random_state=123)
xg_class = XGBClassifier(random_state=123)

xg_class.fit(X_train,y_train_xg)
y_pred_xg_class = xg_class.predict(X_test)
print(f'XGBoost predictions - Class transform [0,1,2]: {y_pred_xg_class[:7]}')
```

```
XGBoost predictions - Class transform [0,1,2]: [2 1 2 1 1 2 0]
```

H.4) Random Forest Classifier

- Los "bosques aleatorios" crean árboles de decisión de muestras de datos seleccionadas al azar, obtienen predicciones de cada árbol y seleccionan la mejor solución mediante votación. También proporciona un indicador bastante bueno de la importancia de la función.

- Este algoritmo es muy popular por su capacidad de combinar los resultados de sus árboles para obtener un resultado final más fiable.

In [754...

```
## -- Random Forest -- ##
rf_class = RandomForestClassifier(max_features=4, random_state=123)
rf_class_fit = rf_class.fit(X_train, y_train)
y_pred_rf_class = rf_class_fit.predict(X_test)
print(f'Random Forest predictions: {y_pred_rf_class[:7]}')
```

```
Random Forest predictions: [3 2 3 2 2 3 1]
```

Exercici 2

Compara els models de classificació utilitzant la precisió (accuracy), una matriu de confiança i d'altres mètriques més avançades.

1. Matriz de confusión: significado de la matriz de confusión y sus métricas asociadas.

<table>
<tr><td colspan="2" rowspan="2">Matriz de confusión</td><td colspan="2">Estimado por el modelo</td><td colspan="2" rowspan="2"></td></tr>
<tr><td>Negativo (N)</td><td>Positivo (P)</td></tr>
<tr><td rowspan="2">Real</td><td>Negativo</td><td>a: (TN)</td><td>b: (FP)</td><td></td><td></td></tr>
<tr><td>Positivo</td><td>c: (FN)</td><td>d: (TP)</td><td>Precisión
("precision")
Porcentaje predicciones positivas correctas:</td><td>d/(b+d)</td></tr>
<tr><td colspan="2" rowspan="2"></td><td>Sensibilidad, exhaustividad
("Recall")
Porcentaje casos positivos detectados</td><td>Especifidad
(Specifity)
Porcentaje casos negativos detectados</td><td colspan="2">Exactitud
("accuracy")
Porcentaje de predicciones correctas
(No sirve en datasets poco equilibrados)</td></tr>
<tr><td>d/(d+c)</td><td>a/(a+b)</td><td colspan="2">(a+d)/(a+b+c+d)</td></tr>
</table>

- Los valores de la diagonal principal se corresponden con los valores estimados de forma correcta por el modelo, tanto los verdaderos positivos_ TP(d), como los verdaderos negativos_TN (a).
- La otra diagonal, por tanto, representa los casos en los que el modelo «se ha equivocado (falsos negativos_FN, falsos positivos_FP).

1. Exactitud:

- La exactitud (o «accuracy«) representa el porcentaje de predicciones correctas frente al total. Por tanto, es el cociente entre los casos bien clasificados por el modelo (verdaderos positivos y verdaderos negativos, es decir, los valores en la diagonal de la matriz de confusión), y la suma de todos los casos.

- Sin embargo, cuando un conjunto de datos es poco equilibrado, no es una métrica útil.

1. Precisión:

- La precisión, (o “precision”) se refiere a lo cerca que está el resultado de una predicción del valor verdadero. Por tanto, es el cociente entre los casos positivos bien clasificados por el modelo y el total de predicciones positivas.

1. Otras métricas:

   a) Sensibilidad ó exhaustividad : La sensibilidad (o recall) representa la tasa de verdaderos positivos (True Positive Rate) ó TP, positivos bien clasificados por el modelo, respecto al total de positivos. Representa la habilidad del modelo de detectar los casos relevantes.

   b) Especifidad: Es la tasa de verdaderos negativos, (“true negative rate”)o TN, negativos bien clasificados por el modelo, respecto al total de negativos.

2.1 Métricas del Modelo K-Nearest Neighbors

In [755…

```
print('##-- K-Nearest Neighbors --## ')
print(f'Acuraccy: {accuracy_score(y_test, y_pred_kn_class)}')
print(f'Precision: {precision_score(y_test, y_pred_kn_class, average="macro")}')
print(f'Recall: {recall_score(y_test, y_pred_kn_class, average="macro")}')
print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred_kn_class))
print("\nClassification Report")
print(classification_report(y_test, y_pred_kn_class))
```

```
##-- K-Nearest Neighbors --##
Acuraccy: 0.8135593220338984
Precision: 0.8142857142857144
Recall: 0.8247655122655123

Confusion Matrix
[[15  0  1]
 [ 1 17  3]
 [ 2  4 16]]

Classification Report
              precision    recall  f1-score   support

           1       0.83      0.94      0.88        16
           2       0.81      0.81      0.81        21
           3       0.80      0.73      0.76        22

    accuracy                           0.81        59
   macro avg       0.81      0.82      0.82        59
weighted avg       0.81      0.81      0.81        59
```

- Los resultados muestran que nuestro algoritmo KNN pudo clasificar los 59 registros del conjunto de prueba con un % de precisión para las clases 2 y 3 y con un % para la clase 1.

2.2 Métricas del Modelo Support Vector Machines

In [756…

```
print('\n##-- Support Vector Machines SVC --## ')
print(f'Acuraccy: {accuracy_score(y_test, y_pred_svc_class)}')
print(f'Precision: {precision_score(y_test, y_pred_svc_class, average="macro")}')
print(f'Recall: {recall_score(y_test, y_pred_svc_class, average="macro")}')
print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred_svc_class))
```

```
print("\nClassification Report")
print(classification_report(y_test, y_pred_svc_class))
```

```
##-- Support Vector Machines SVC --##
Acuraccy: 0.9830508474576272
Precision: 0.9848484848484849
Recall: 0.9848484848484849

Confusion Matrix
[[16  0  0]
 [ 0 21  0]
 [ 0  1 21]]

Classification Report
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        16
           2       0.95      1.00      0.98        21
           3       1.00      0.95      0.98        22

    accuracy                           0.98        59
   macro avg       0.98      0.98      0.98        59
weighted avg       0.98      0.98      0.98        59
```

2.3 Métricas del Modelo XGBoost

In [757…

```
print('\n##-- XGBoost --## ')
print(f'Acuraccy: {accuracy_score(y_test_xg, y_pred_xg_class)}')
print(f'Precision: {precision_score(y_test_xg, y_pred_xg_class, average="macro")}')
print(f'Recall: {recall_score(y_test_xg, y_pred_xg_class, average="macro")}')
print("\nConfusion Matrix")
print(confusion_matrix(y_test_xg, y_pred_xg_class))
print("\nClassification Report")
print(classification_report(y_test_xg, y_pred_xg_class))
```

```
##-- XGBoost --##
Acuraccy: 1.0
Precision: 1.0
Recall: 1.0

Confusion Matrix
[[16  0  0]
 [ 0 21  0]
 [ 0  0 22]]

Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      1.00      1.00        21
           2       1.00      1.00      1.00        22

    accuracy                           1.00        59
   macro avg       1.00      1.00      1.00        59
weighted avg       1.00      1.00      1.00        59
```

2.4 Métricas del Modelo Random Forest Classifier

In [758…

```
print('\n##-- Random Forest --## ')
print(f'Acuraccy: {accuracy_score(y_test, y_pred_rf_class)}')
```

```
print(f'Precision: {precision_score(y_test, y_pred_rf_class, average="macro")}')
print(f'Recall: {recall_score(y_test, y_pred_rf_class, average="macro")}')
print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred_rf_class))
print("\nClassification Report")
print(classification_report(y_test, y_pred_rf_class))
```

```
##-- Random Forest --##
Acuraccy: 1.0
Precision: 1.0
Recall: 1.0

Confusion Matrix
[[16  0  0]
 [ 0 21  0]
 [ 0  0 22]]

Classification Report
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        16
           2       1.00      1.00      1.00        21
           3       1.00      1.00      1.00        22

    accuracy                           1.00        59
   macro avg       1.00      1.00      1.00        59
weighted avg       1.00      1.00      1.00        59
```

Exercici 3

Entrena'ls usant els diferents paràmetres que admeten per tal de millorar-ne la predicció.

3.1 Optimización parámetros del Modelo KN

3.1.1 Evaluación del Parámetro K

- El valor k en el algoritmo k-NN define cuántos vecinos se verificarán para determinar la clasificación de un punto de consulta específico.
- Por ejemplo, si k=1, la instancia se asignará a la misma clase que su vecino más cercano.
- Definir k puede ser un acto de equilibrio ya que diferentes valores pueden llevar a un ajuste excesivo o insuficiente.
- Los valores más bajos de k pueden tener una varianza alta, pero un sesgo bajo, y los valores más grandes de k pueden generar un sesgo alto y una varianza más baja.
- La elección de k dependerá en gran medida de los datos de entrada, ya que los datos con más valores atípicos o ruido probablemente funcionarán mejor con valores más altos de k.
- En general, se recomienda tener un número impar para k para evitar empates en la clasificación, y las tácticas de validación cruzada pueden ayudarlo a elegir la k óptima para su conjunto de datos.

In [759…

```
pprint(kn_class.get_params())
```

```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
```

```
'n_jobs': None,
'n_neighbors': 5,
'p': 2,
'weights': 'uniform'}
```

In [760…

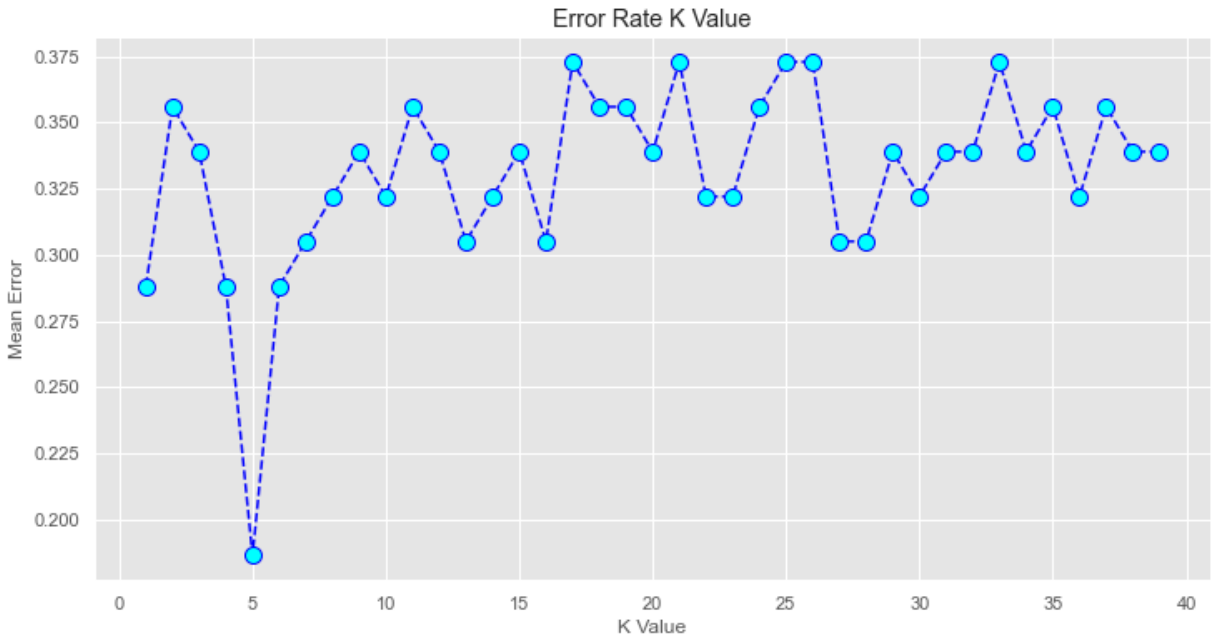
```
error = []

# Calculating error for K values between 1 and 40
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))
```

In [761…

```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='blue', linestyle='dashed', marker='o',
         markerfacecolor='cyan', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

Out[761…

```
Text(0, 0.5, 'Mean Error')
```



- El K =5 (n_neighbors=5) en nuestro caso es el que genera el error medio inferior.

3.1.2 Optimización de parámetros

In [762…

```
## -- Optimización de los parámetros Modelo KN --##

kn_class= KNeighborsClassifier(n_neighbors=5, algorithm ='auto',leaf_size= 30, metri
grid = {
        'n_neighbors':[5,6],
        'weights': ['uniform', 'distance'],
        'metric': ['euclidean', 'manhattan', 'minkowski']}

kn_class_grid = GridSearchCV(kn_class, grid, cv = 5)
kn_class_grid_fit = kn_grid.fit(X_train, y_train)

print(f'Best parameters: {kn_class_grid_fit.best_params_}')
print(f'Best score: {kn_class_grid_fit.best_score_}')
```

```
Best parameters: {'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'distance'}
Best score: 0.764855072463768
```

3.1.3 Ajuste con test

In [763…

```
y_pred_kn_grid = kn_class_grid_fit.predict(X_test)

print('##-- KN Grid Search & CV --## ')
print(f'Acuraccy: {accuracy_score(y_test, y_pred_kn_grid)}')
print(f'Precision: {precision_score(y_test, y_pred_kn_grid, average="macro")}')
print(f'Recall: {recall_score(y_test, y_pred_kn_grid, average="macro")}')

print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred_kn_grid))
print("\nClassification Report")
print(classification_report(y_test, y_pred_kn_class))
```

```
##-- KN Grid Search & CV --##
Acuraccy: 0.8813559322033898
Precision: 0.8878623188405798
Recall: 0.8868145743145743

Confusion Matrix
[[15  0  1]
 [ 1 19  1]
 [ 0  4 18]]

Classification Report
              precision    recall  f1-score   support

           1       0.83      0.94      0.88        16
           2       0.81      0.81      0.81        21
           3       0.80      0.73      0.76        22

    accuracy                           0.81        59
   macro avg       0.81      0.82      0.82        59
weighted avg       0.81      0.81      0.81        59
```

3.2 Optimización parámetros del Modelo SVC

In [764…

```
pprint(svc_class.get_params())
```

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'linear',
 'max_iter': -1,
 'probability': False,
 'random_state': 123,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

3.2.2 Optimización de parámetros

In [765…
```
## -- Optimización de los parámetros Modelo SVC--##

svc_class = svm.SVC(kernel='linear',random_state=123)
grid = {
         'kernel': ['linear', 'rbf', 'sigmoid'],
         'C': [10, 1.0, 0.1, 0.01],
         'gamma':['scale','auto']}

svc_class_grid = GridSearchCV(svc_class, grid, cv = 5)
svc_class_grid_fit = svc_class_grid.fit(X_train, y_train)

#ajustes_svc= pd.concat([pd.DataFrame(svc_class_grid_fit.cv_results_["params"]),pd.D
# ajustes_svc.sort_values(by="Accuracy", ascending=False)

print(f'Best parameters: {svc_class_grid_fit.best_params_}')
print(f'Best score: {svc_class_grid_fit.best_score_}')
```

```
Best parameters: {'C': 0.1, 'gamma': 'scale', 'kernel': 'linear'}
Best score: 0.9409420289855073
```

3.2.3 Ajuste con test

In [766…
```
y_pred_svc_grid = svc_class_grid_fit.predict(X_test)

print('##-- SVM Grid Search & CV --## ')
print(f'Acuraccy: {accuracy_score(y_test, y_pred_svc_grid)}')
print(f'Precision: {precision_score(y_test, y_pred_svc_grid, average="macro")}')
print(f'Recall: {recall_score(y_test, y_pred_svc_grid, average="macro")}')


print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred_svc_grid))
print("\nClassification Report")
print(classification_report(y_test, y_pred_svc_class))
```

```
##-- SVM Grid Search & CV --##
Acuraccy: 1.0
Precision: 1.0
Recall: 1.0

Confusion Matrix
[[16  0  0]
 [ 0 21  0]
 [ 0  0 22]]

Classification Report
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        16
           2       0.95      1.00      0.98        21
           3       1.00      0.95      0.98        22

    accuracy                           0.98        59
   macro avg       0.98      0.98      0.98        59
weighted avg       0.98      0.98      0.98        59
```

Exercici 4

Compara el seu rendiment fent servir l'aproximació traint/test o cross-validation.

4.1 Data Frame

In [767…
```
X_true= data.drop(columns="Class")
y_true= data["Class"]
```

In [768…
```
X_true.head()
```

Out[768…

	Alcohol	Malic Acid	Ash	Alcalinity of Ash	Magnesium	Total Phenols	Flavanoids	Nonflavanoid Phenols	Proanthocyanins
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82

In [769…
```
y_true.head()
```

Out[769…
```
0    1
1    1
2    1
3    1
4    1
Name: Class, dtype: int64
```

4.2 Validación Cruzada con True_data

4.2.1 K-Nearest Neighbors Algorithm

In [770…
```
## -- K-Nearest Neighbors Algorithm -- ##
scores_kn1= cross_val_score(kn_class_grid, X_true, y_true, cv=5)
print("Accuracy of KN:  %0.4f - with a standard deviation of %0.4f" % (scores_kn1.me
scores_F1_kn1= cross_val_score(kn_class_grid, X_true, y_true, cv=5,scoring='f1_macro
print("F1 of KN:   %0.4f - with a standard deviation of %0.4f" % (scores_F1_kn1.mean
```

```
Accuracy of KN:  0.7475 - with a standard deviation of 0.0550
F1 of KN:   0.7357 - with a standard deviation of 0.0528
```

4.2.2 Support Vector Machines SVM

In [771…
```
## --Support Vector Machines SVM -- ##
scores_svc1= cross_val_score(svc_class_grid, X_true, y_true, cv=5)
print("Accuracy of SVM: %0.4f - with a standard deviation of %0.4f" % (scores_svc1.m
scores_F1_svc1= cross_val_score(svc_class_grid, X_true, y_true, cv=5,scoring='f1_mac
print("F1 of SVM: %0.4f F1 of SVM - with a standard deviation of %0.4f" % (scores_F1
```

```
Accuracy of SVM: 0.9722 - with a standard deviation of 0.0304
F1 of SVM: 0.9732 F1 of SVM - with a standard deviation of 0.0301
```

4.2.3 XGBClassifier

In [772…
```
## --XGBClassifier  -- ##
y_true_xg = le.fit_transform(y_true)
```

```
scores_xg1=cross_val_score(xg_class, X_true, y_true_xg, cv=5)
print("Accuracy of Xboots: %0.4f - with a standard deviation of %0.4f" % (scores_xg1
scores_F1_xg1=cross_val_score(xg_class, X_true, y_true_xg, cv=5,scoring='f1_macro')
print("F1 of Xboots: %0.4f - with a standard deviation of %0.4f" % (scores_F1_xg1.me
```

```
Accuracy of Xboots: 0.9498 - with a standard deviation of 0.0323
F1 of Xboots: 0.9491 - with a standard deviation of 0.0321
```

4.2.4 Random Forest

In [773…

```
## -- Random Forest -- ##
scores_rf1=cross_val_score(rf_class, X_true, y_true, cv=5)
print("Accuracy of Random Forest: %0.4f - with a standard deviation of %0.4f" % (sco
scores_F1_rf1=cross_val_score(rf_class, X_true, y_true, cv=5,scoring='f1_macro')
print("F1 of Random Forest: %0.4f - with a standard deviation of %0.4f" % (scores_F1
```

```
Accuracy of Random Forest: 0.9776 - with a standard deviation of 0.0208
F1 of Random Forest: 0.9782 - with a standard deviation of 0.0205
```

4.2.5 Sumario de Resultados

In [774…

```
print("F1 for KN mean: {:.4f}, std: {:.4f}".format(scores_F1_kn1.mean(), scores_F1_k
print("F1 for SVM mean: {:.4f}, std: {:.4f}".format(scores_F1_svc1.mean(), scores_F1
print("F1 for XGB mean: {:.4f}, std: {:.4f}".format(scores_F1_xg1.mean(), scores_F1_
print("F1 for RF mean: {:.4f}, std: {:.4f}".format(scores_F1_rf1.mean(), scores_F1_r
```

```
F1 for KN mean: 0.7357, std: 0.0528
F1 for SVM mean: 0.9732, std: 0.0301
F1 for XGB mean: 0.9491, std: 0.0321
F1 for RF mean: 0.9782, std: 0.0205
```

- EL modelo más optimo en este caso es el Random Forest y con métricas muy aproximadas Suport Vector Machine.
- El modelo que mos se ajusta en la predicción cruzada es K-Nearest Neighbors.

Exercici 5

Aplica algun procés d'enginyeria per millorar els resultats (normalització, estandardització, mostreig...)

5.1 Estandatización , entrenamiento y métricas con Pipeline de los Modelos

5.1.1 K-Nearest Neighbors

In [707…

```
#---- K-Nearest Neighbors Algorithm ----#
pipe_kn = Pipeline([
    ('scaler', preprocessing.StandardScaler()),
    ('selector', VarianceThreshold()),
    ('classifier', KNeighborsClassifier(n_neighbors=5, algorithm ='auto', metric='ma


pipe_fit = pipe_kn.fit(X_train, y_train)
y_pred_pipe = pipe_fit.predict(X_test)

print('##-- Kn --## ')
print(f'Acuraccy: {accuracy_score(y_test, y_pred_pipe)}')
print(f'Precision: {precision_score(y_test, y_pred_pipe, average="macro")}')
print(f'Recall: {recall_score(y_test, y_pred_pipe, average="macro")}')
print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred_pipe))
```

```
print("\nClassification Report")
print(classification_report(y_test, y_pred_pipe))
```

```
##-- Kn --##
Acuraccy: 0.9830508474576272
Precision: 0.9803921568627452
Recall: 0.9841269841269842

Confusion Matrix
[[16  0  0]
 [ 1 20  0]
 [ 0  0 22]]

Classification Report
              precision    recall  f1-score   support

           1       0.94      1.00      0.97        16
           2       1.00      0.95      0.98        21
           3       1.00      1.00      1.00        22

    accuracy                           0.98        59
   macro avg       0.98      0.98      0.98        59
weighted avg       0.98      0.98      0.98        59
```

5.1.2 Support Vector Machines SVM

In [708…

```
## --Support Vector Machines SVM -- ##
pipe = Pipeline([
    ('scaler', preprocessing.StandardScaler()),
    ('selector', VarianceThreshold()),
    ('classifier',svm.SVC(kernel='linear', gamma= 'scale',C= 0.1 ,random_state=123))


pipe_fit = pipe.fit(X_train, y_train)
y_pred_pipe = pipe_fit.predict(X_test)

print('##-- Support Vector Machines SVM --## ')
print(f'Acuraccy: {accuracy_score(y_test, y_pred_pipe)}')
print(f'Precision: {precision_score(y_test, y_pred_pipe, average="macro")}')
print(f'Recall: {recall_score(y_test, y_pred_pipe, average="macro")}')
print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred_pipe))
print("\nClassification Report")
print(classification_report(y_test, y_pred_pipe))
```

```
##-- Support Vector Machines SVM --##
Acuraccy: 0.9830508474576272
Precision: 0.9848484848484849
Recall: 0.9848484848484849

Confusion Matrix
[[16  0  0]
 [ 0 21  0]
 [ 0  1 21]]

Classification Report
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        16
           2       0.95      1.00      0.98        21
           3       1.00      0.95      0.98        22
```

```
    accuracy                           0.98        59
   macro avg       0.98      0.98      0.98        59
weighted avg       0.98      0.98      0.98        59
```

5.1.3 XGBClassifier

In [709…

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_train_xg = le.fit_transform(y_train)
y_test_xg=le.fit_transform(y_test)
```

In [710…

```
## --XGBClassifier  -- ##

pipe = Pipeline([
    ('scaler', preprocessing.StandardScaler()),
    ('selector', VarianceThreshold()),
    ('classifier',XGBClassifier(random_state=123))])


pipe_fit = pipe.fit(X_train, y_train_xg)
y_pred_pipe = pipe_fit.predict(X_test)

print('##-- Support Vector Machines SVM --## ')
print(f'Acuraccy: {accuracy_score(y_test_xg, y_pred_pipe)}')
print(f'Precision: {precision_score(y_test_xg, y_pred_pipe, average="macro")}')
print(f'Recall: {recall_score(y_test_xg, y_pred_pipe, average="macro")}')
print("\nConfusion Matrix")
print(confusion_matrix(y_test_xg, y_pred_pipe))
print("\nClassification Report")
print(classification_report(y_test_xg, y_pred_pipe))
```

```
##-- Support Vector Machines SVM --##
Acuraccy: 1.0
Precision: 1.0
Recall: 1.0

Confusion Matrix
[[16  0  0]
 [ 0 21  0]
 [ 0  0 22]]

Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      1.00      1.00        21
           2       1.00      1.00      1.00        22

    accuracy                           1.00        59
   macro avg       1.00      1.00      1.00        59
weighted avg       1.00      1.00      1.00        59
```

5.1.4 Random Forest Classifier

In [711…

```
## --Random Forest -- ##
pipe = Pipeline([
    ('scaler', preprocessing.StandardScaler()),
    ('selector', VarianceThreshold()),
    ('classifier',RandomForestClassifier(max_features=4, random_state=123))])
```

```
pipe_fit = pipe.fit(X_train, y_train)
y_pred_pipe = pipe_fit.predict(X_test)

print('##-- Support Vector Machines SVM --## ')
print(f'Acuraccy: {accuracy_score(y_test, y_pred_pipe)}')
print(f'Precision: {precision_score(y_test, y_pred_pipe, average="macro")}')
print(f'Recall: {recall_score(y_test, y_pred_pipe, average="macro")}')
print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred_pipe))
print("\nClassification Report")
print(classification_report(y_test, y_pred_pipe))
```

```
##-- Support Vector Machines SVM --##
Acuraccy: 1.0
Precision: 1.0
Recall: 1.0

Confusion Matrix
[[16  0  0]
 [ 0 21  0]
 [ 0  0 22]]

Classification Report
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        16
           2       1.00      1.00      1.00        21
           3       1.00      1.00      1.00        22

    accuracy                           1.00        59
   macro avg       1.00      1.00      1.00        59
weighted avg       1.00      1.00      1.00        59
```

5.2 Validación Cruzada con datos Estandarizados

In [712…

```
## ---- Estandarización ----##
scaler = StandardScaler()
scaler.fit(X_true)

X_true = scaler.transform(X_true)
```

5.2.1 K-Nearest Neighbors Algorithm

In [713…

```
## -- K-Nearest Neighbors Algorithm -- ##
scores_kn2= cross_val_score(kn_class_grid, X_true, y_true, cv=5)
print("Accuracy of KN: %0.4f - with a standard deviation of %0.4f" % (scores_kn2.mea
scores_F1_kn2= cross_val_score(kn_class_grid, X_true, y_true, cv=5,scoring='f1_macro
print("F1 of KN: %0.4f- with a standard deviation of %0.4f" % (scores_F1_kn2.mean(),
```

```
Accuracy of KN: 0.9495 - with a standard deviation of 0.0207
F1 of KN: 0.9521- with a standard deviation of 0.0197
```

5.2.2 Support Vector Machines SVM

In [714…

```
## --Support Vector Machines SVM -- ##
scores_svc2= cross_val_score(svc_class_grid, X_true, y_true, cv=5)
print("Accuracy of SVM: %0.4f - with a standard deviation of %0.4f" % (scores_svc2.m
scores_F1_svc2= cross_val_score(svc_class_grid, X_true, y_true, cv=5,scoring='f1_mac
print("F1 of SVM: %0.4f - with a standard deviation of %0.4f" % (scores_F1_svc2.mean
```

```
Accuracy of SVM: 0.9776 - with a standard deviation of 0.0208
F1 of SVM: 0.9777 - with a standard deviation of 0.0205
```

5.2.3 XGBClassifier

In [715…

```
## --XGBClassifier  -- ##
y_true = le.fit_transform(y_true)
scores_xg2=cross_val_score(xg_class, X_true, y_true_xg, cv=5)
print("Accuracy of Xboots: %0.4f - with a standard deviation of %0.4f" % (scores_xg2
scores_F1_xg2=cross_val_score(xg_class, X_true, y_true_xg, cv=5,scoring='f1_macro')
print("F1 of Xboots: %0.4f - with a standard deviation of %0.4f" % (scores_F1_xg2.me
```

```
Accuracy of Xboots: 0.9498 - with a standard deviation of 0.0323
F1 of Xboots: 0.9491 - with a standard deviation of 0.0321
```

5.2.4 Random Forest

In [716…

```
## -- Random Forest
scores_rf2=cross_val_score(rf_class, X_true, y_true, cv=5)
print("Accuracy of Random Fores: %0.4f - with a standard deviation of %0.4f" % (scor
scores_F1_rf2=cross_val_score(rf_class, X_true, y_true, cv=5,scoring='f1_macro')
print("F1 of Random Forest: %0.4f - with a standard deviation of %0.4f" % (scores_F1
```

```
Accuracy of Random Fores: 0.9776 - with a standard deviation of 0.0208
F1 of Random Forest: 0.9782 - with a standard deviation of 0.0205
```

5.2.5 Sumario de Resultados

In [728…

```
print("F1 for KN mean: {:.4f}, std: {:.4f}".format(scores_F1_kn2.mean(), scores_F1_k
print("F1 for SVM mean: {:.4f}, std: {:.4f}".format(scores_F1_svc2.mean(), scores_F1
print("F1 for XGB mean: {:.4f}, std: {:.4f}".format(scores_F1_xg2.mean(), scores_F1_
print("F1 for RF mean: {:.4f}, std: {:.4f}".format(scores_F1_rf2.mean(), scores_F1_r
```

```
F1 for KN mean: 0.9521, std: 0.0197
F1 for SVM mean: 0.9777, std: 0.0205
F1 for XGB mean: 0.9491, std: 0.0321
F1 for RF mean: 0.9782, std: 0.0205
```

- En este caso aplicando Estandarización en todas las variables los datos son similares a los resultado anteriores, salvo en el caso de K-Nearest Neighbors Algorithm que mejora notablemente sus métricas.
- El modelo óptimo sigue siendo Random Forest seguido de Suport Vector Machine

5.3 Estandarización, Normalización, RobustScaler, entrenamiento y métricas del Modelo:

5.3.1 Estandarización y Normalización de las variables

- La distribución de "Alcalinity of Ash" es normal y por tanto se aplica estandaritzación
- Al resto de variables se aplica RobustScaler, para solucionar los outlaier que apararcen en algunas clases de las variables explitativas y son más significativos en las variables: "Malic Acid ","Magnesium","Proanthocyanins ","Color intensity".

In [717…

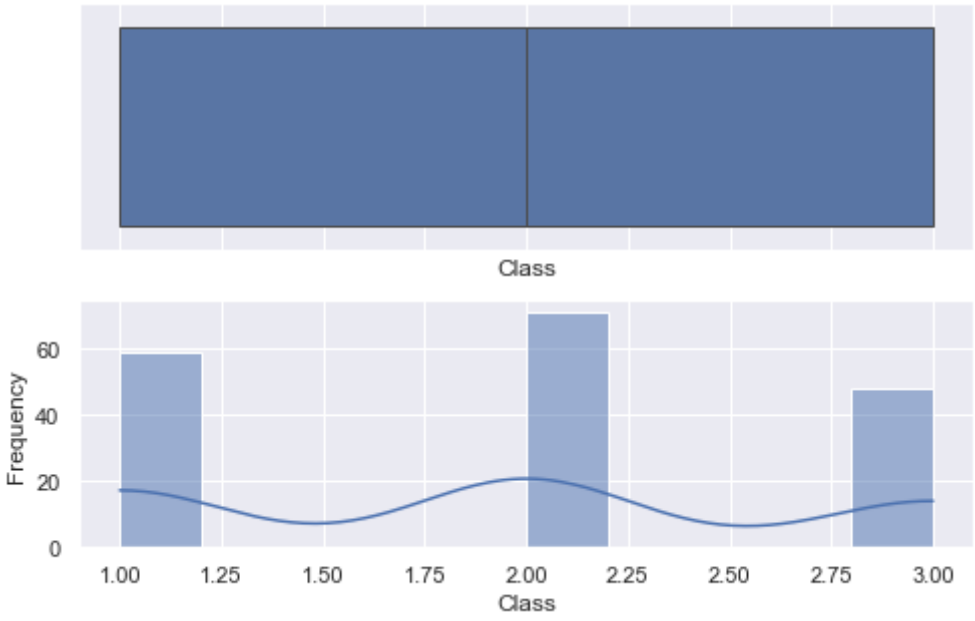
```
from sklearn.preprocessing import RobustScaler
df=data
standColumns = ['Alcalinity of Ash']
scalerStand = preprocessing.StandardScaler().fit(df[standColumns])
df[standColumns] = scalerStand.transform(df[standColumns])
```

```
normColumns = ['Alcohol', 'Malic Acid', 'Ash', 'Magnesium', 'Total Phenols', 'Flavan
scalerNorm = preprocessing.RobustScaler().fit(df[normColumns])# RobustEscaler
df[normColumns] = scalerNorm.transform(df[normColumns])
```

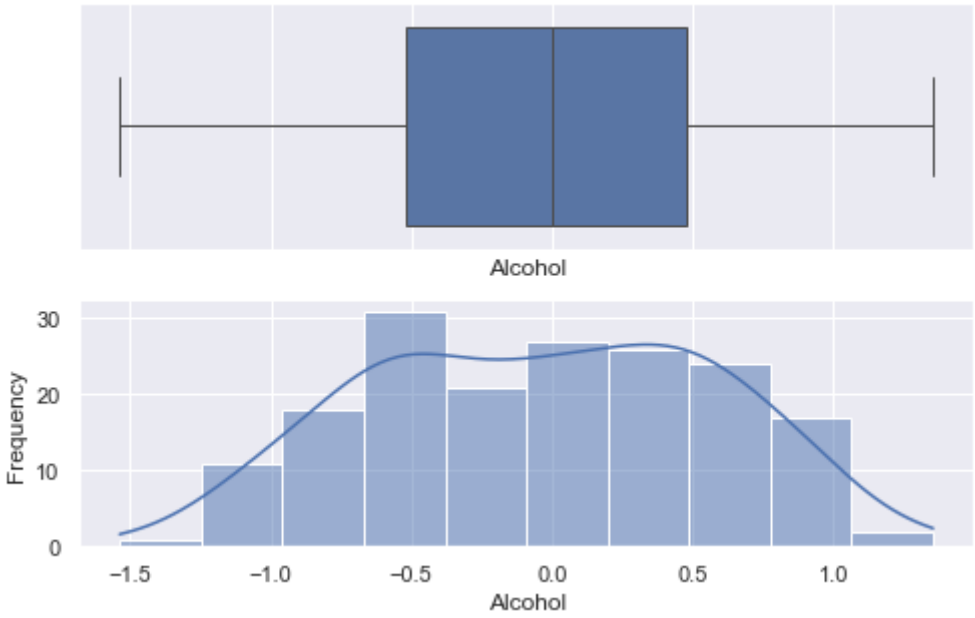
In [718…

```
for i in df.columns:
    plt.figure()
    plt.tight_layout()
    sns.set(rc={"figure.figsize":(8, 5)})
    f, (ax_box, ax_hist) = plt.subplots(2, sharex=True)
    plt.gca().set(xlabel= i,ylabel='Frequency')
    sns.boxplot(df[i],  ax=ax_box , linewidth= 1.0)
    sns.histplot(df[i], ax=ax_hist , bins = 10,kde=True)
```

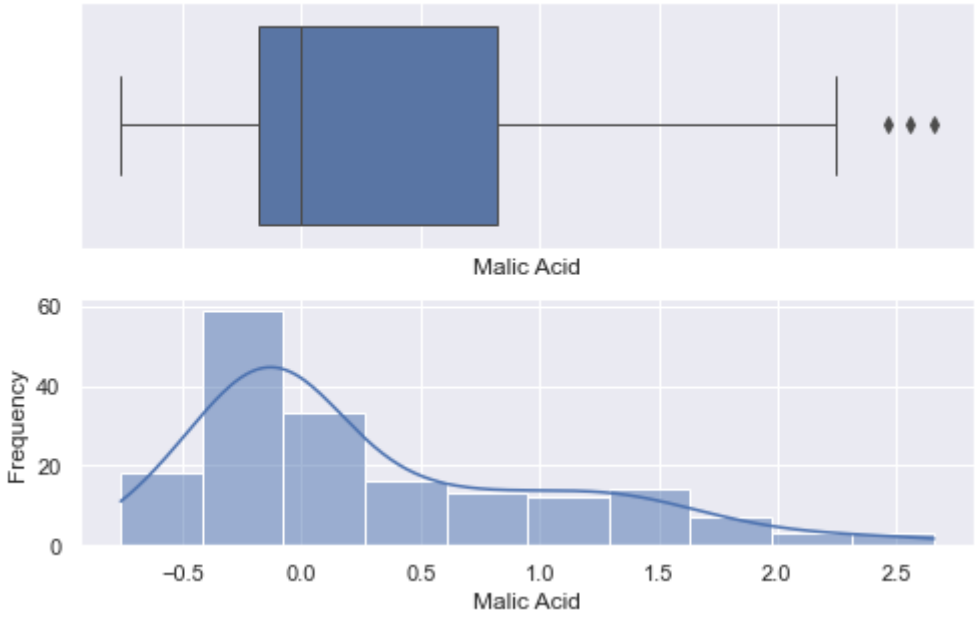
```
<Figure size 576x360 with 0 Axes>
```



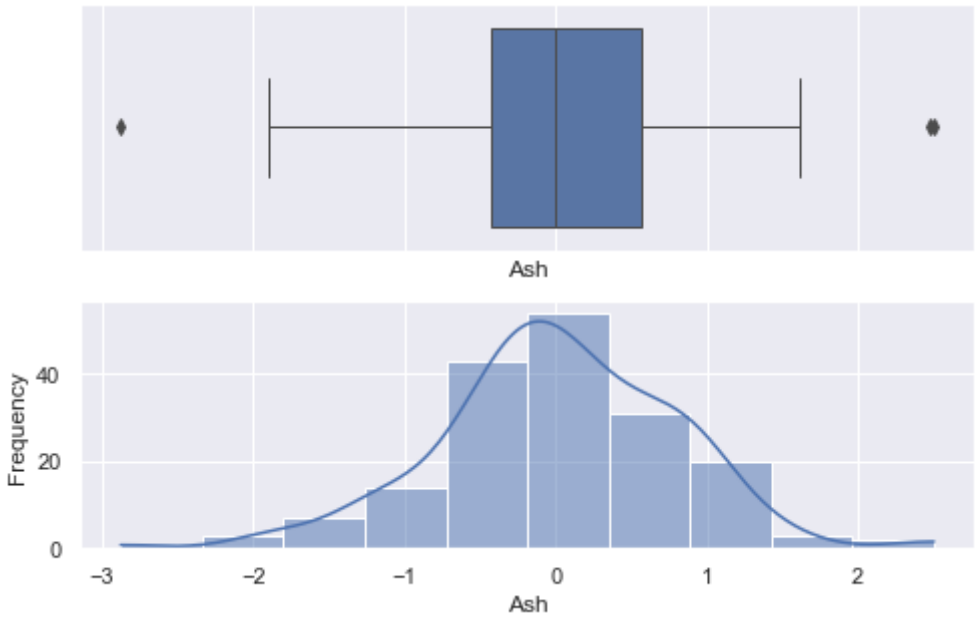

```
<Figure size 576x360 with 0 Axes>
```



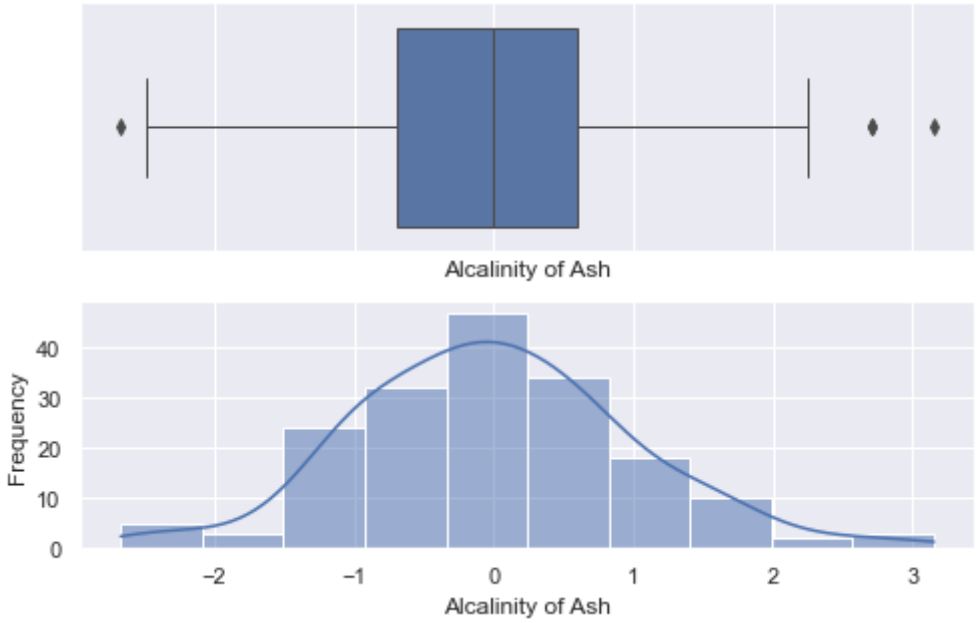

```
<Figure size 576x360 with 0 Axes>
```



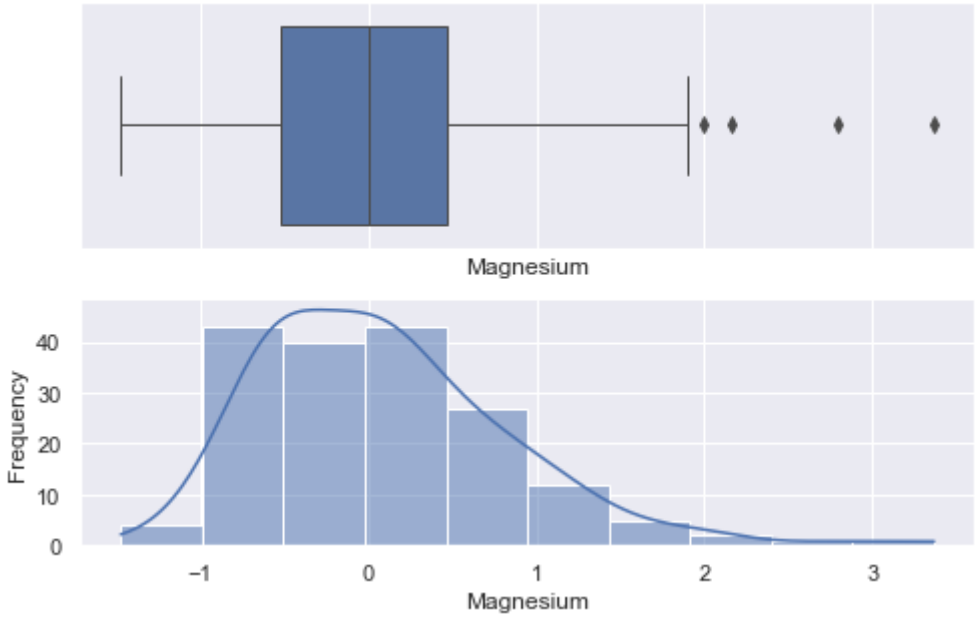
<Figure size 576x360 with 0 Axes>



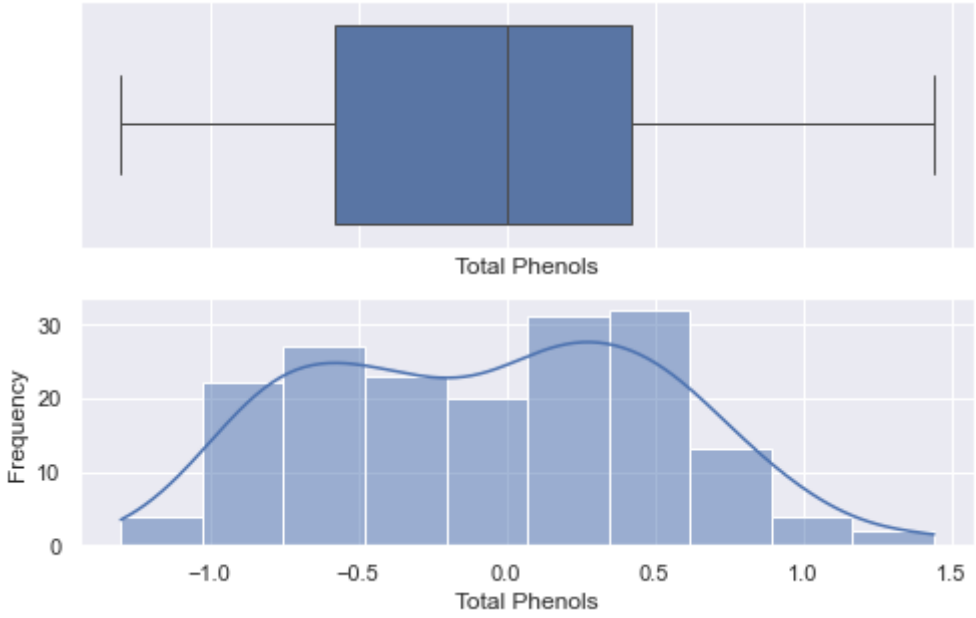
<Figure size 576x360 with 0 Axes>



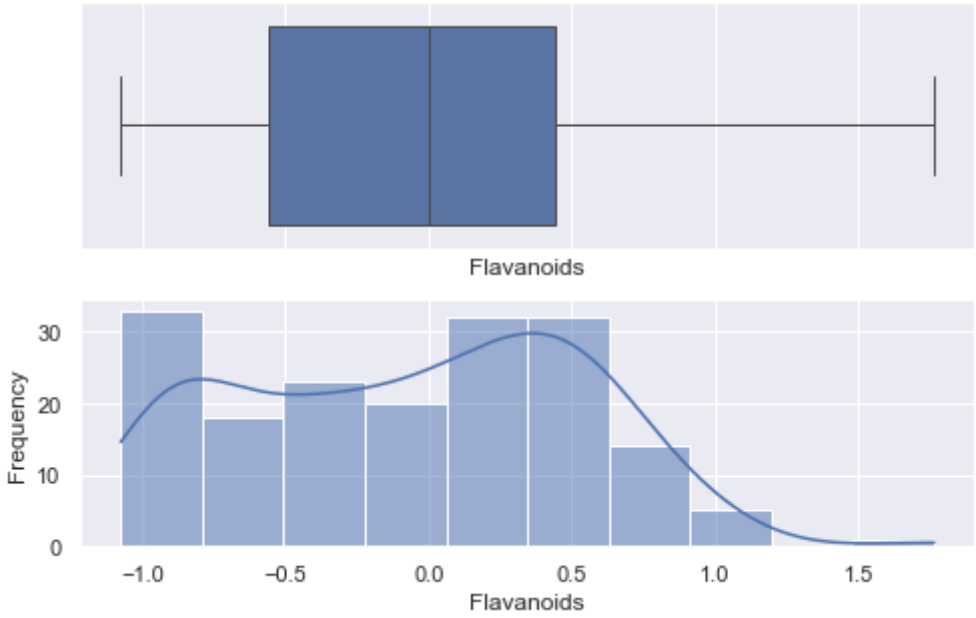
<Figure size 576x360 with 0 Axes>



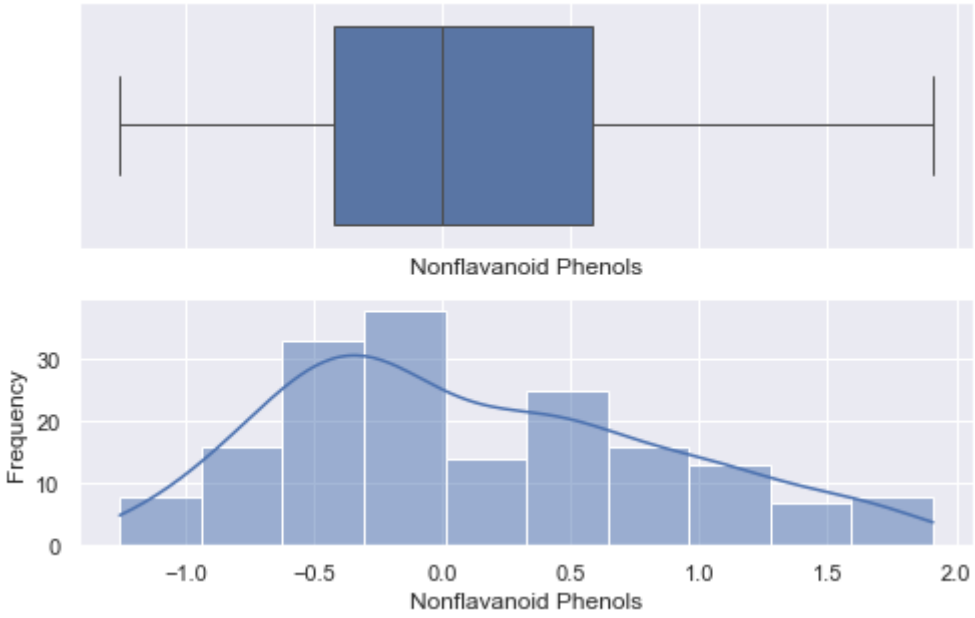
<Figure size 576x360 with 0 Axes>



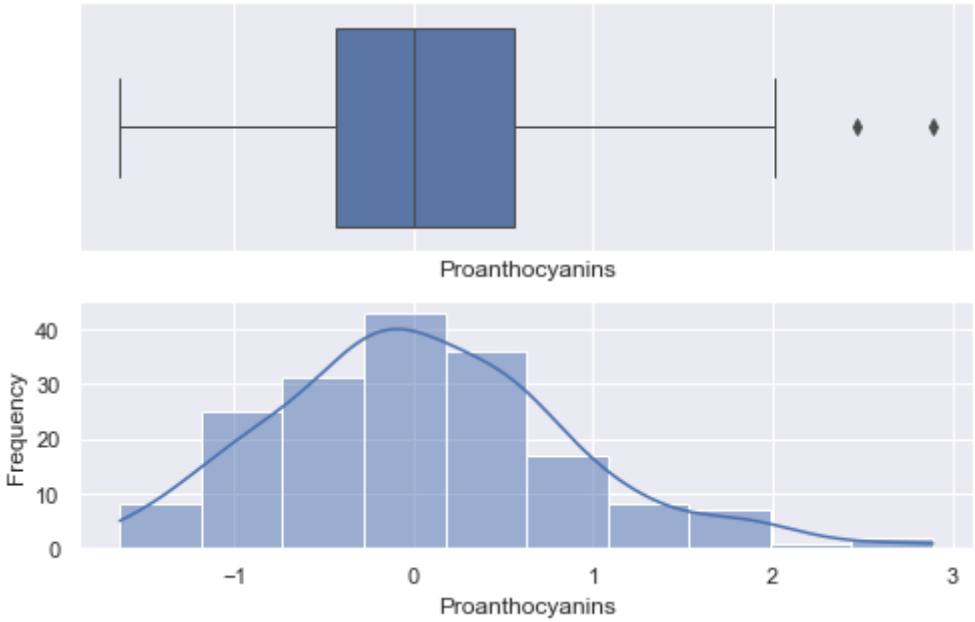
<Figure size 576x360 with 0 Axes>



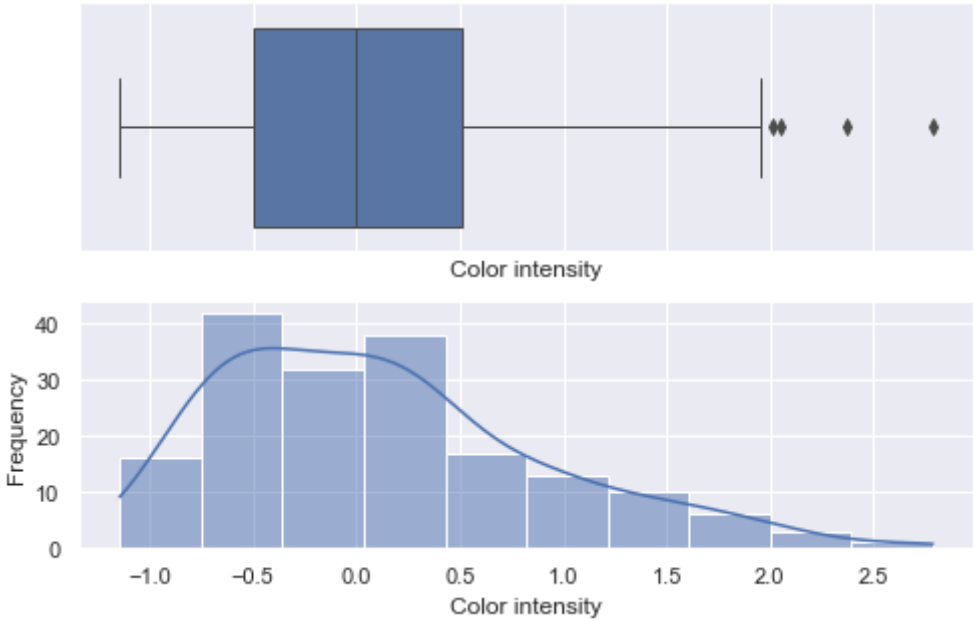
<Figure size 576x360 with 0 Axes>



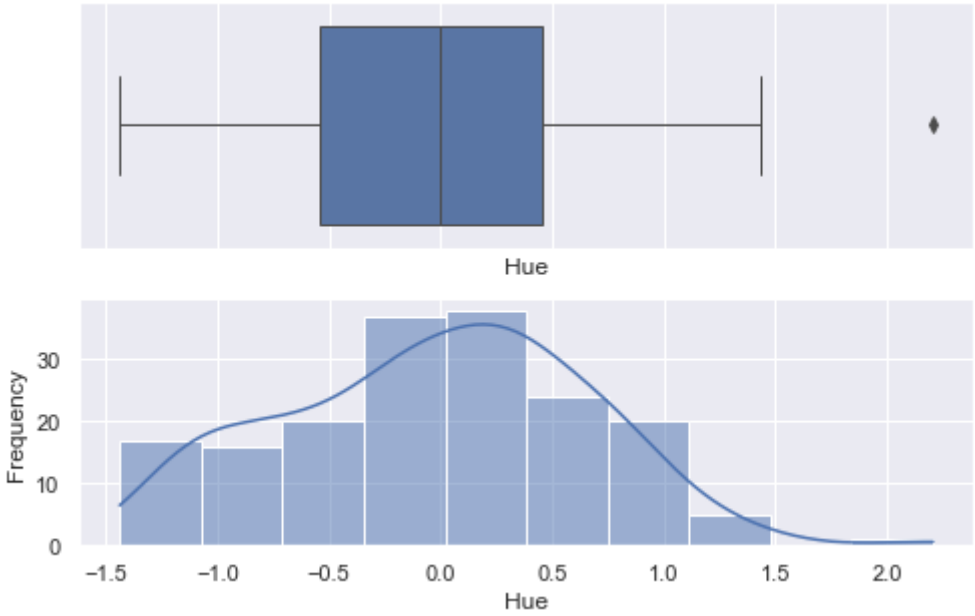
<Figure size 576x360 with 0 Axes>



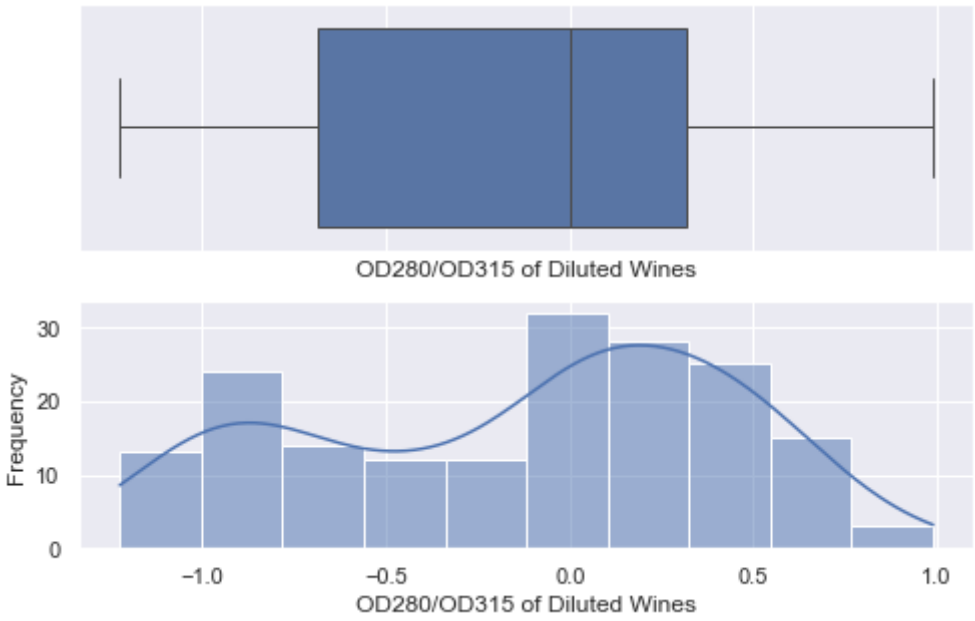
<Figure size 576x360 with 0 Axes>



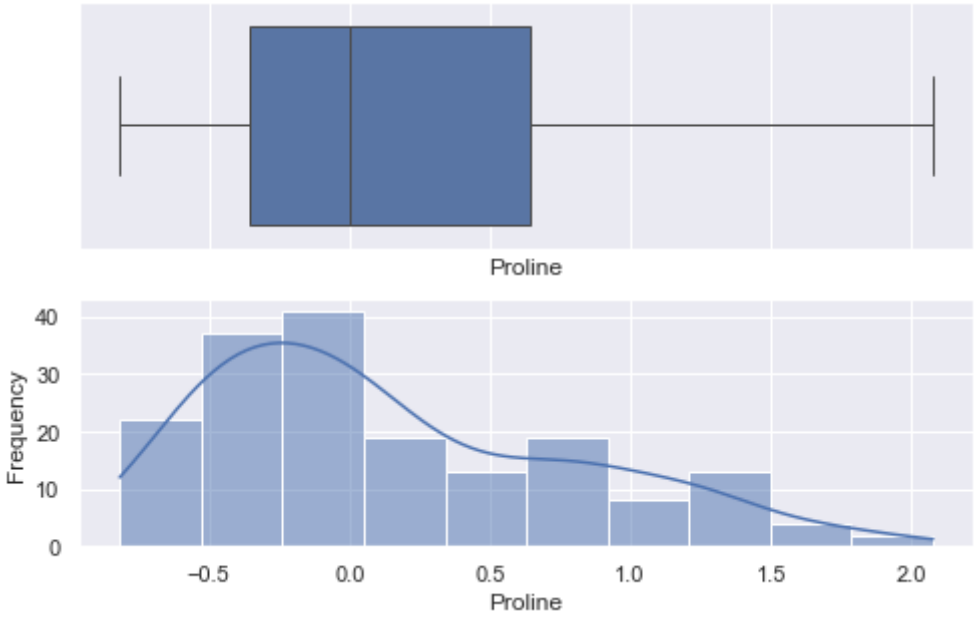
<Figure size 576x360 with 0 Axes>




```
<Figure size 576x360 with 0 Axes>
```




```
<Figure size 576x360 with 0 Axes>
```



- Las estadísticas de centrado y escalamiento de RobustScaler se basan en percentiles y no

están influenciadas por unos pocos valores atípicos marginales muy grandes.

- El rango resultante de los valores de las características transformadas es mayor que para la estandarización o normalización y son aproximadamente similares. La mayoría de los valores transformados se encuentran en un rango [-3, 3].
- Los valores atípicos todavía están presentes en los datos transformados. Si se desearamos un recorte de valores atípicos por separado, se debería aplicar una transformación no lineal.

5.3.2 Selección de X_train - X_test

In [719…
```
X = df.drop(['Class'],axis=1)
y = df['Class']

# adaptación valores de y reales al modelo XGB
y_xg = le.fit_transform(y_true)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_s
```

5.3.4 Definición de los modelos: KNeighborsClassifier - Support Vector Machines SMV - XGBClassifier - RandomForestClassifier

In [720…
```
kn = KNeighborsClassifier(n_neighbors=5)
svc = svm.SVC(kernel='linear',random_state=123)
rf = RandomForestClassifier(max_features=4, random_state=123)
```

In [721…
```
## -- adaptación de la variavle objetivo a [0,1,2]--#
y_train_xg = le.fit_transform(y_train)
y_test_xg=le.fit_transform(y_test)

xg = XGBClassifier(random_state=123)
```

Training

In [722…
```
kn.fit(X_train, y_train)
svc.fit(X_train, y_train)
xg.fit(X_train, y_train_xg)
rf.fit(X_train, y_train)
print("")
```

Predictions

In [723…
```
y_pred_kn = kn.predict(X_test)
y_pred_svc = svc.predict(X_test)
y_pred_xg = xg.predict(X_test)
y_pred_rf = svc.predict(X_test)
```

Evaluación

In [724…
```
cf_matrix_kn = confusion_matrix(y_test, y_pred_kn)
print("\nConfusion Matrix - KN")
print(cf_matrix_kn)

cf_matrix_svc = confusion_matrix(y_test, y_pred_svc)
print("\nConfusion Matrix - SVC")
print(cf_matrix_svc)
```

```
cf_matrix_xg = confusion_matrix(y_test_xg, y_pred_xg)
print("\nConfusion Matrix - XGB Classifier ")
print(cf_matrix_xg)

cf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
print("\nConfusion Matrix - Random Forest Classifier ")
print(cf_matrix_rf)
```

```
Confusion Matrix - KN
[[16  0  0]
 [ 2 19  0]
 [ 0  0 22]]

Confusion Matrix - SVC
[[16  0  0]
 [ 0 21  0]
 [ 0  2 20]]

Confusion Matrix - XGB Classifier
[[16  0  0]
 [ 0 21  0]
 [ 0  0 22]]

Confusion Matrix - Random Forest Classifier
[[16  0  0]
 [ 0 21  0]
 [ 0  2 20]]
```

5.3.5 Cross Validation

In [725…

```
from sklearn.metrics import f1_score

f1_kn = f1_score(y_test, y_pred_kn, average='macro')
f1_svc = f1_score(y_test, y_pred_svc, average='macro')
f1_xg = f1_score(y_test_xg, y_pred_xg, average='macro')
f1_rf = f1_score(y_test, y_pred_rf, average='macro')
print("F1 for KN: {:.4f}, SVC: {:.4f}, XGB:{:.4f}, RF: {:.4f}".format(f1_kn, f1_svc,
```

```
F1 for KN: 0.9637, SVC: 0.9690, XGB:1.0000, RF: 0.9690
```

In [726…

```
cv_kn = cross_val_score(kn, X, y, cv=5, scoring='f1_macro')
print("F1 for KN mean: {:.4f}, std: {:.4f}".format(cv_kn.mean(), cv_kn.std()) )

cv_svc = cross_val_score(svc, X, y, cv=5, scoring='f1_macro')
print("F1 for SVC mean: {:.4f}, std: {:.4f}".format(cv_svc.mean(), cv_svc.std()) )

cv_xg = cross_val_score(xg, X, y_xg, cv=5, scoring='f1_macro')
print("F1 for XGB mean: {:.4f}, std: {:.4f}".format(cv_xg.mean(), cv_xg.std()) )

cv_rf = cross_val_score(rf, X, y, cv=5, scoring='f1_macro')
print("F1 for RF mean: {:.4f}, std: {:.4f}".format(cv_rf.mean(), cv_rf.std()) )
```

```
F1 for KN mean: 0.9632, std: 0.0213
F1 for SVC mean: 0.9547, std: 0.0284
F1 for XGB mean: 0.9491, std: 0.0321
F1 for RF mean: 0.9782, std: 0.0205
```

- Con el proceso de datos de estandarización y RobustScaler mejoran los resultados de K-Nearest Neighbors Algorithm, pero bajan las métricas de SVC, aunque en este caso los parámetros no han sido optimizados como en el apartado anterior.

- Los modelos Random Forest y XGB mantiene las mismas métricas que el apartado anterior, ya que en ninguno de los dos casos se han optimizado parámetros al obtener resultados de F1 =1 en el training de los modelos.