CST4050 Challenge No. 2 (Week 10)

Problem Description

The Wine dataset, available in your MyLearning folder, consists of 170 wine samples, each characterised by 13 features representing various chemical constituents. These features include attributes like alcohol content, malic acid, ash, alcalinity of ash, magnesium, and more. The target variable comprises three classes, denoted as 0, 1, and 2, representing different types of wine. Objective: Use your Machine Learning expertise to implement a Logistic Regression model for predicting the target variable. Your model should prioritise accuracy while maintaining simplicity to prevent overfitting and ensure interpretability. You are required to design and implement a machine learning pipeline to accomplish the tasks highlighted below.

1. Train a multi-class Logistic Regression model to predict the target variable (class 0, 1,and 2).
2. Fine-tune the model to prevent overfitting and ensure simplicity by minimising the number of features that take part in the model.
3. Conduct an analysis of bias-variance for the tuned model and assess its performance on unseen data.
4. Perform a detailed analysis to enhance explainability. Include clear comments on the role of predictors in predicting the target for each class (i.e., class 0, 1, and 2).
5. Apply the model to predict the target of unseen data found in your MyLearning folder. During these tasks, it is essential to provide a thorough rationale and justification for each chosen step.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns


from sklearn.linear_model import LogisticRegression

# To build model for statistical analysis and prediction
import statsmodels.stats.api as sms
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
from statsmodels.tools.tools import add_constant

# Library to split data
from sklearn.model_selection import train_test_split

# To get diferent metric scores
from sklearn import metrics
from sklearn.metrics import accuracy_score, confusion_matrix

%load_ext nb_black
```

In [2]:
```
# Reading the Data
df = pd.read_csv("wine.csv")
df_t = pd.read_csv("unseen.csv")
```

In [3]:
```
print(df_t.shape)  # Rows and Columns of Test data
df.shape  # Rows and Columns of Train data
```

Out[3]:
```
(10, 13)
(168, 14)
```

In [4]:
```
df.head()
```

Out[4]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	target
0	13.50	1.81	2.61	20.0	96.0	2.53	2.61	0.28	1.66	3.52	1.12	3.82	845.0	0
1	13.50	3.12	2.62	24.0	123.0	1.40	1.57	0.22	1.25	8.60	0.59	1.30	500.0	2
2	13.41	3.84	2.12	18.8	90.0	2.45	2.68	0.27	1.48	4.28	0.91	3.00	1035.0	0
3	12.77	3.43	1.98	16.0	80.0	1.63	1.25	0.43	0.83	3.40	0.70	2.12	372.0	1
4	13.63	1.81	2.70	17.2	112.0	2.85	2.91	0.30	1.46	7.30	1.28	2.88	1310.0	0

In [5]:
```
# Checking for null and/or duplicated values
print(df.isnull().sum())
print(f"\n duplicated values: {df.duplicated().sum()}")
```

```
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
target                          0
dtype: int64

 duplicated values: 0
```

In [6]:
```
# Outputting statistical summary of feautures
df.describe()
```

Out[6]:

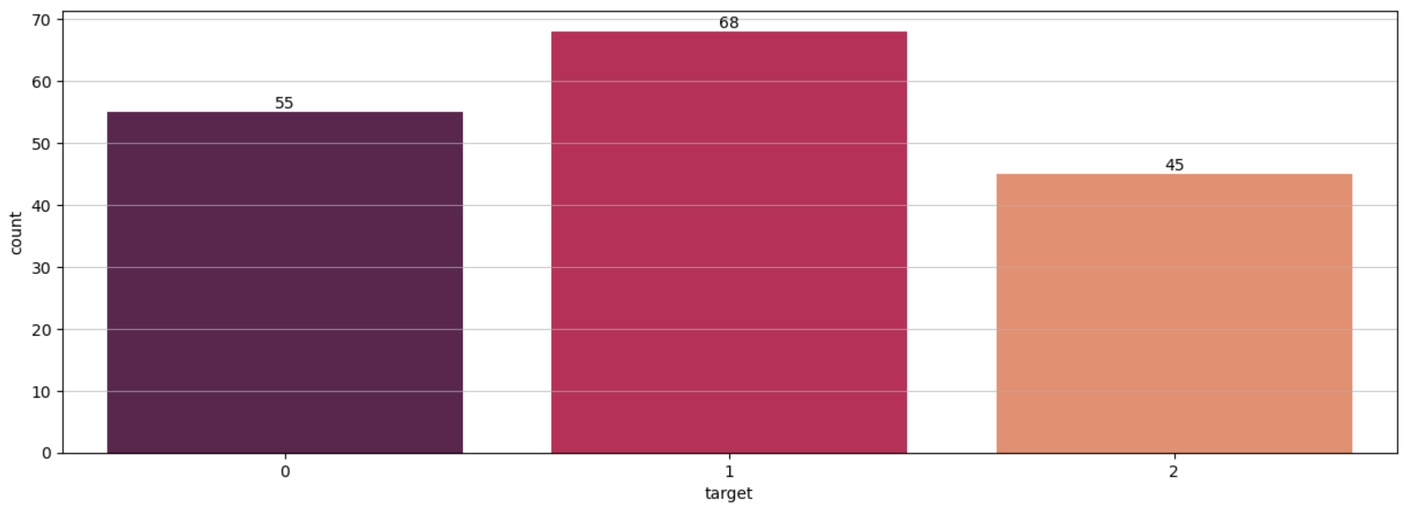
	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	target
count	168.000000	168.000000	168.000000	168.000000	168.000000	168.000000	168.000000	168.000000	168.000000	168.000000	168.000000	168.000000	168.000000	168.000000
mean	12.991429	2.321905	2.361131	19.469643	99.761905	2.295179	2.028750	0.363452	1.593571	5.071369	0.959262	2.599405	741.505952	0.940476
std	0.807605	1.121977	0.273245	3.399996	14.373228	0.629464	0.994674	0.125912	0.570019	2.351903	0.231857	0.715639	316.573723	0.771517
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	0.410000	1.740000	0.480000	1.270000	278.000000	0.000000
25%	12.355000	1.607500	2.210000	17.075000	88.000000	1.735000	1.215000	0.260000	1.250000	3.200000	0.787500	1.905000	495.000000	0.000000
50%	13.050000	1.830000	2.360000	19.250000	98.000000	2.355000	2.135000	0.340000	1.555000	4.600000	0.980000	2.780000	660.000000	1.000000
75%	13.672500	3.030000	2.540000	21.500000	107.000000	2.800000	2.807500	0.450000	1.950000	6.262500	1.120000	3.170000	985.000000	2.000000
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.580000	13.000000	1.710000	4.000000	1680.000000	2.000000

Conducting EDA

[1] What is the distribution of our target variable

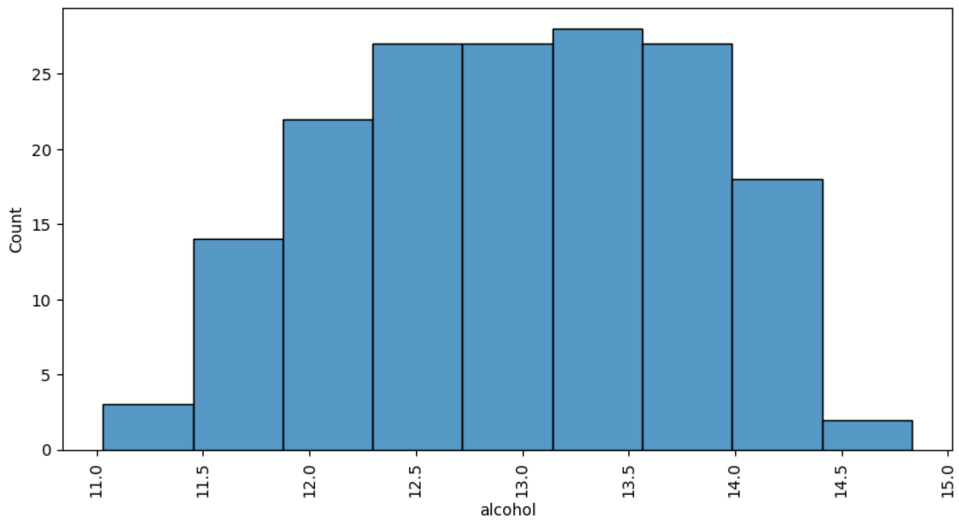
In [7]:
```
# Creating a countplot of each type of wine
plt.figure(figsize=(15, 5))
fig1 = sns.countplot(data=df, x="target", palette="rocket")
plt.grid(axis="y", linewidth=0.5)
# Add data labels
fig1.bar_label(fig1.containers[0], label_type="edge")
```

Out[7]:
```
[Text(0, 0, '55'), Text(0, 0, '68'), Text(0, 0, '45')]
```



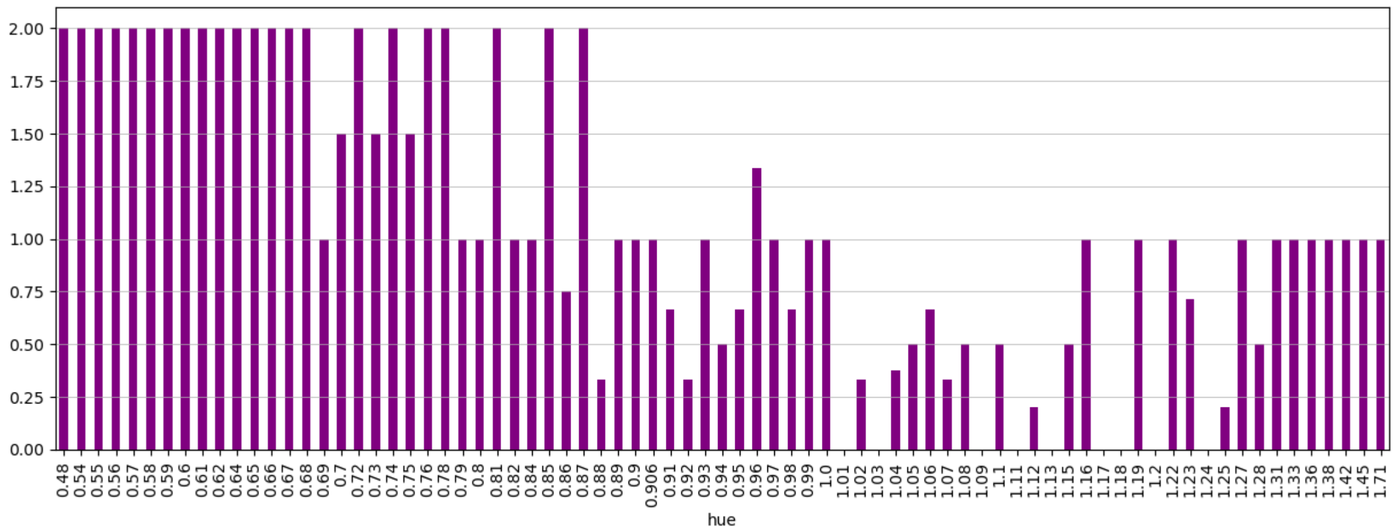
[2] Distribution of the alcohol content in the dataset

In [8]:
```
plt.figure(figsize=(10, 5))
sns.histplot(data=df, x="alcohol", kde=False)
plt.xticks(rotation=90)
plt.show()
```



[3] Is hue important in clasifying the type of wine

In [9]:
```
plt.figure(figsize=(15, 5))
df.groupby("hue")["target"].mean(numeric_only=True).plot.bar(color="purple")
plt.grid(axis="y", linewidth=0.5)
```

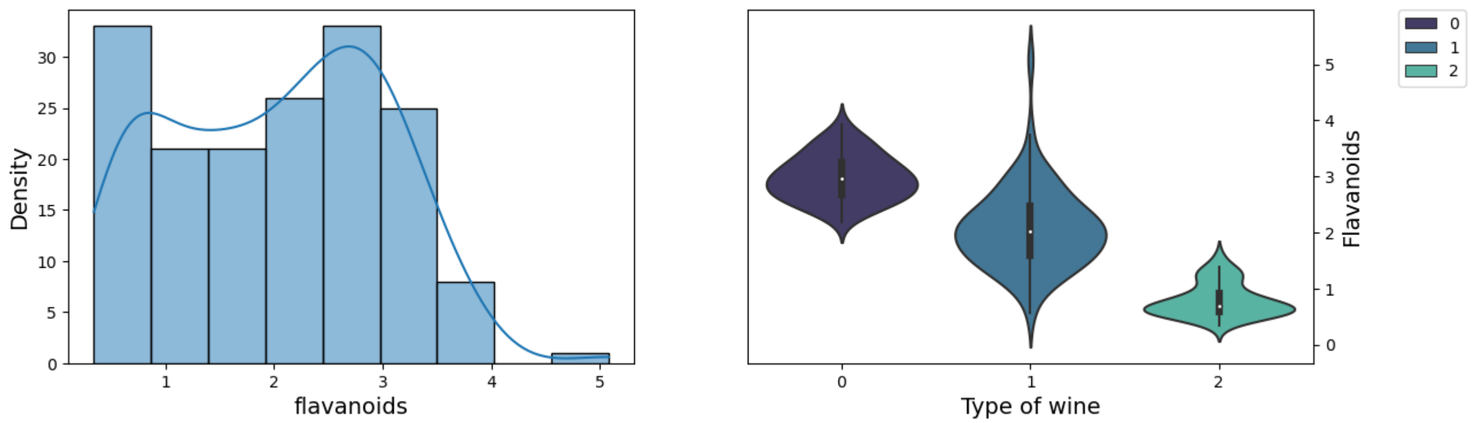


- A lot of lighter hues fall within class 2, hue can be a good indicator of type of wine

[4] Distributions of Flavanoids across the dataset and for each type of wine

In [10]:
```
# Creating a KDE Histogram of Flavanoid distribution
f, axes = plt.subplots(1, 2, figsize=(14, 4))

sns.histplot(df["flavanoids"], ax=axes[0], kde=True)
axes[0].set_xlabel("flavanoids", fontsize=14)
axes[0].set_ylabel("Density", fontsize=14)
axes[0].yaxis.tick_left()


# Creating a Violinplot bertween flavanoid distributions and our target varaible
sns.violinplot(
    x="target",
    y="flavanoids",
    data=df,
    hue="target",
    dodge=False,
    ax=axes[1],
    palette="mako",
```

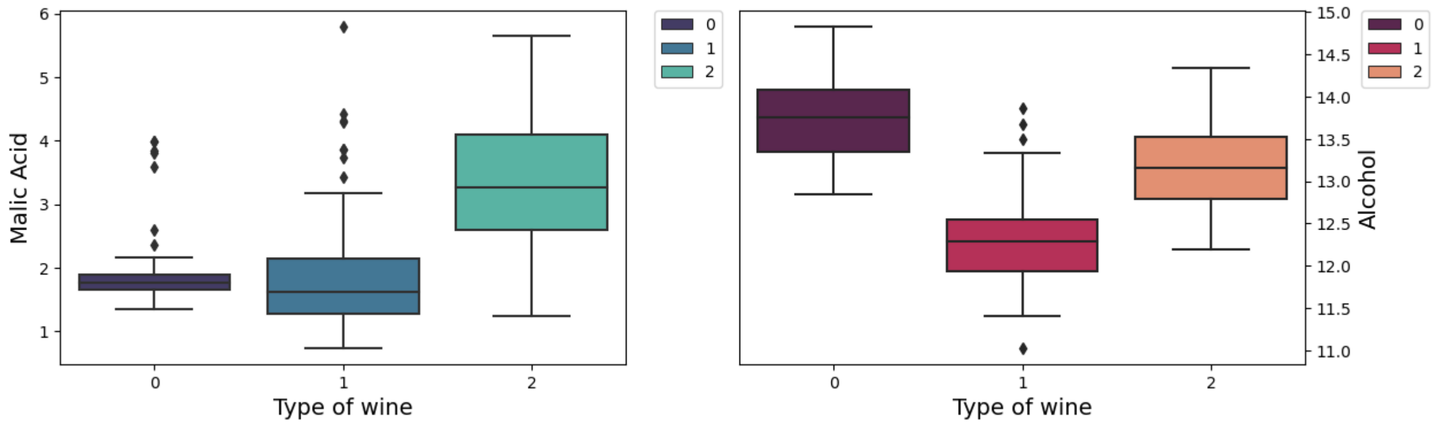
```
)
axes[1].set_xlabel("Type of wine", fontsize=14)
axes[1].set_ylabel("Flavanoids", fontsize=14)
axes[1].yaxis.set_label_position("right")
axes[1].yaxis.tick_right()
axes[1].legend(bbox_to_anchor=(1.15, 1), loc=2, borderaxespad=0.0)

plt.show()
```



- There is a clear density in certain flavanoid regions for each class indicating that it's a strong identifier for type of wine

[5] What is the relationship between malic acid/alcohol content on the type of wine

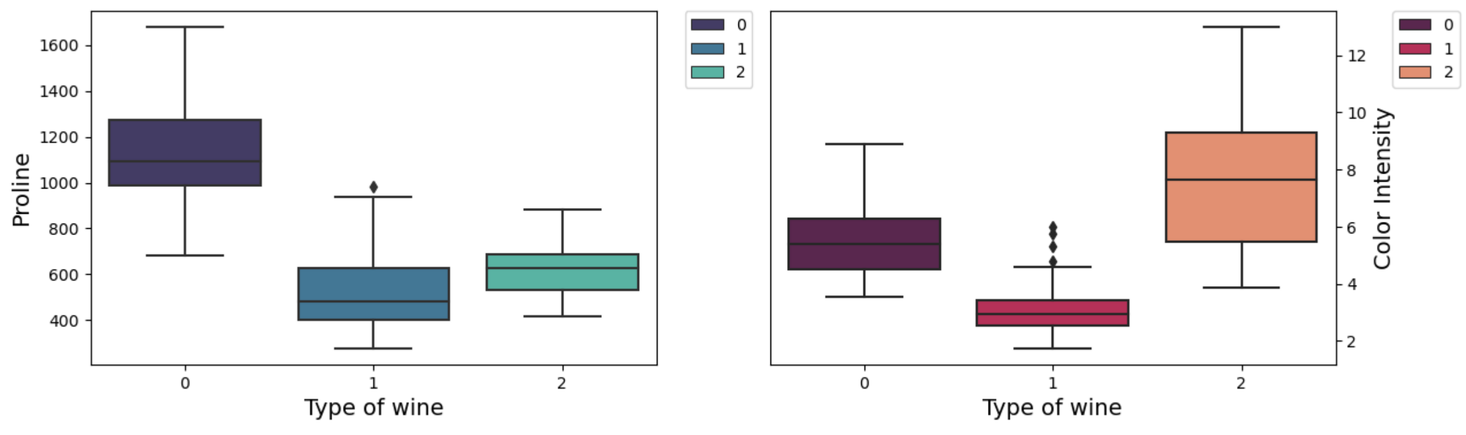
In [11]:
```
# Creating a boxplot to see how our target variable is affected with changes in Malic Acid content
f, axes = plt.subplots(1, 2, figsize=(14, 4))

sns.boxplot(
    x="target",
    y="malic_acid",
    data=df,
    hue="target",
    dodge=False,
    ax=axes[0],
    palette="mako",
)
axes[0].set_xlabel("Type of wine", fontsize=14)
axes[0].set_ylabel("Malic Acid", fontsize=14)
axes[0].yaxis.set_label_position("left")
axes[0].yaxis.tick_left()
axes[0].legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.0)

# Creating a boxplot to see how our target variable is affected with changes in Alcohol content
sns.boxplot(
    x="target",
    y="alcohol",
    data=df,
    hue="target",
    dodge=False,
    ax=axes[1],
    palette="rocket",
)
axes[1].set_xlabel("Type of wine", fontsize=14)
axes[1].set_ylabel("Alcohol", fontsize=14)
axes[1].yaxis.set_label_position("right")
axes[1].yaxis.tick_right()
axes[1].legend(bbox_to_anchor=(1.1, 1), loc=2, borderaxespad=0.0)

plt.show()
```



- Malic Acid allows for clear dintinguishing between a majority of class 2 and the other types of wine so it is important to include
- Alcohol content also allows us to clearly distinguish between a majority of class 1 and the other types of wine

[6] What is the relationship between proline/color intensity on the type of wine

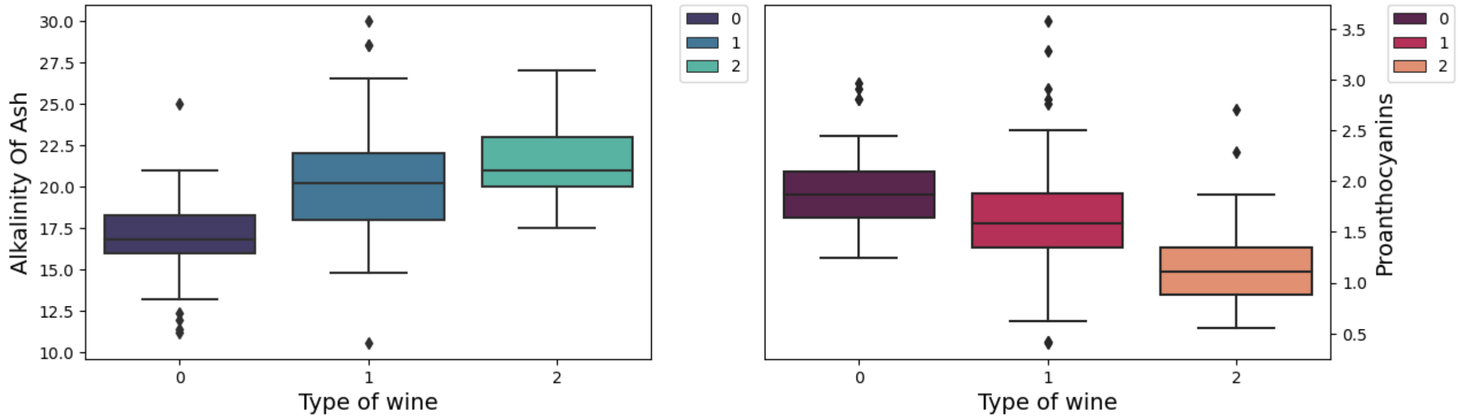
In [12]:
```
# Creating a boxplot to see how our target variable is affected with changes in Proline content
f, axes = plt.subplots(1, 2, figsize=(14, 4))

sns.boxplot(
    x="target",
    y="proline",
    data=df,
    hue="target",
    dodge=False,
    ax=axes[0],
    palette="mako",
)
axes[0].set_xlabel("Type of wine", fontsize=14)
axes[0].set_ylabel("Proline", fontsize=14)
axes[0].yaxis.set_label_position("left")
axes[0].yaxis.tick_left()
axes[0].legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.0)

# Creating a boxplot to see how our target variable is affected with changes in the Color Intensity of Wine
sns.boxplot(
    x="target",
    y="color_intensity",
    data=df,
    hue="target",
    dodge=False,
    ax=axes[1],
    palette="rocket",
)
axes[1].set_xlabel("Type of wine", fontsize=14)
axes[1].set_ylabel("Color Intensity", fontsize=14)
axes[1].yaxis.set_label_position("right")
axes[1].yaxis.tick_right()
axes[1].legend(bbox_to_anchor=(1.1, 1), loc=2, borderaxespad=0.0)

plt.show()
```



- Proline content allows us to clearly distinguish between most of Class 0 wine and the others
- Color Intensity allows us to clearly distinguish between most of Class 1 wine and the others

[7] What is the relationship between alkalinity of ash/proanthocyanins on the type of wine

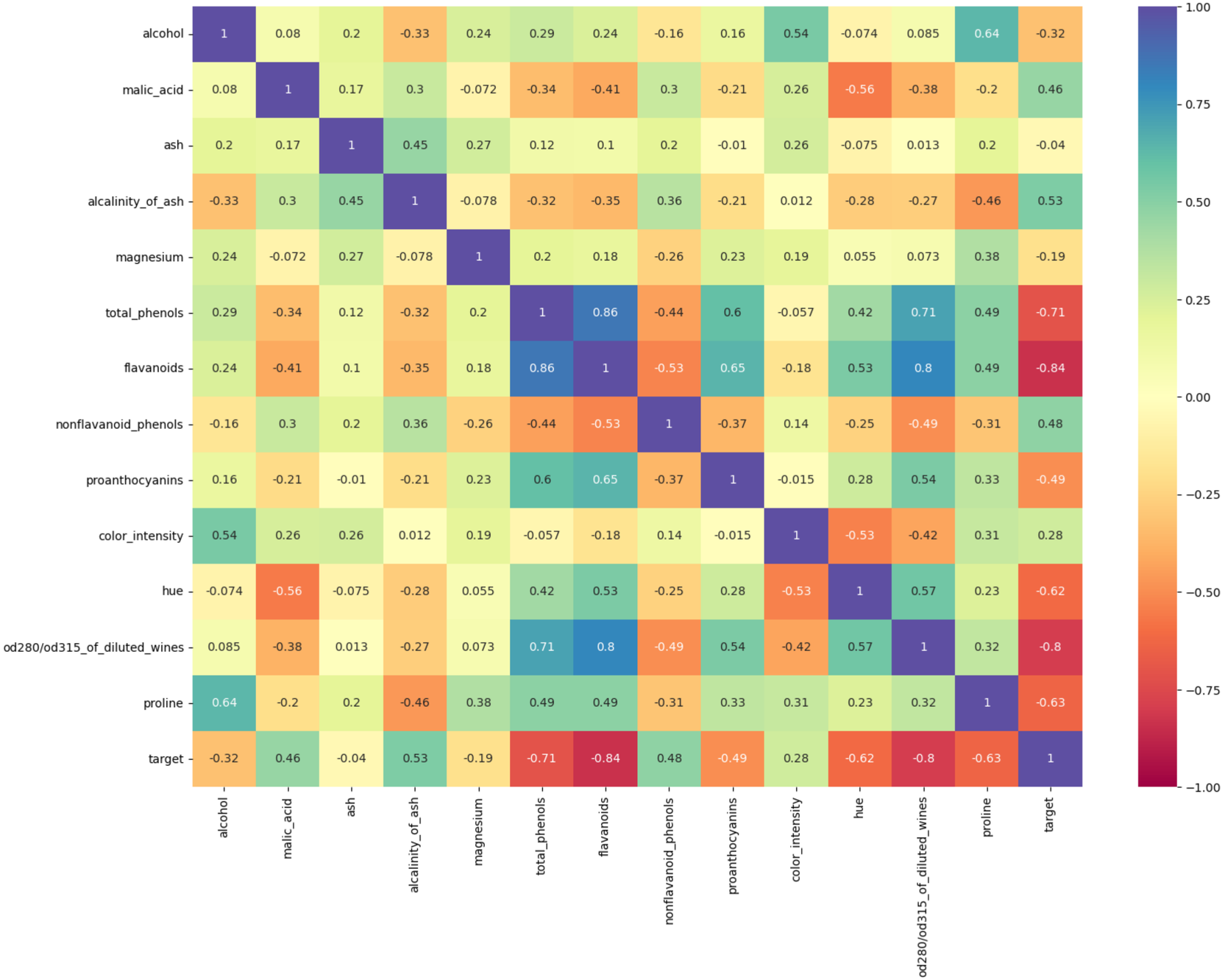
In [13]:
```
# Creating a boxplot to see how our target variable is affected with changes in Alkalinity of Ash
f, axes = plt.subplots(1, 2, figsize=(14, 4))

sns.boxplot(
    x="target",
    y="alcalinity_of_ash",
    data=df,
    hue="target",
    dodge=False,
    ax=axes[0],
    palette="mako",
)
axes[0].set_xlabel("Type of wine", fontsize=14)
axes[0].set_ylabel("Alkalinity Of Ash", fontsize=14)
axes[0].yaxis.set_label_position("left")
axes[0].yaxis.tick_left()
axes[0].legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.0)

# Creating a boxplot to see how our target variable is affected with changes in Proanthocyanin content
sns.boxplot(
    x="target",
    y="proanthocyanins",
    data=df,
    hue="target",
    dodge=False,
    ax=axes[1],
    palette="rocket",
)
axes[1].set_xlabel("Type of wine", fontsize=14)
axes[1].set_ylabel("Proanthocyanins", fontsize=14)
axes[1].yaxis.set_label_position("right")
axes[1].yaxis.tick_right()
axes[1].legend(bbox_to_anchor=(1.1, 1), loc=2, borderaxespad=0.0)

plt.show()
```



- While both these variables visually show a light correlation with our target variable, it'll have to been seen if the initial LR model considers these variables improtant before deciding to include or exclude them in our final model

[8] Creating a correlation matrix heatmap

In [14]:
```
# Creating a Heatmap
plt.figure(figsize=(17, 12))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap="Spectral", vmin=-1, vmax=1)
plt.show()
```



- 'magnesium' and 'ash' seems to have minimal correlations with both the target variable and other independent variables so removing it can simplify the data
- 'flavanoids' while being highly correlated with the target variable, is also strongly correlated with several other independent variables, indicating multicollinearity (especially with 'total_phenols' and 'od280/od315_of_diluted_wines')

Splitting the Data into Training and Test (Validation) sets

In [15]:
```
X = df.drop(["target"], axis=1)
Y = df["target"]

# adding a contstant to X variable
X = add_constant(X)


# Splitting data in train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.30, random_state=1, stratify=Y
)
```

In [16]:
```
# Checking for total number of unique values in each column
df.nunique()
```

Out[16]:
```
alcohol                         121
malic_acid                      126
ash                              78
alcalinity_of_ash                61
magnesium                        53
total_phenols                    95
flavanoids                      127
nonflavanoid_phenols             38
proanthocyanins                  98
color_intensity                 129
hue                              77
od280/od315_of_diluted_wines    117
proline                         119
target                            3
dtype: int64
```

Checking for Multicollinearity using Variance Inflation Factor

In [17]:
```
# Let's check the VIF of the predictors
vif_series = pd.Series(
    [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])],
    index=X_train.columns,
    dtype=float,
)
print("VIF values: \n\n{}\n".format(vif_series))
```

```
VIF values:

const                         630.007216
alcohol                         2.448415
malic_acid                      1.658090
ash                             2.328067
alcalinity_of_ash               2.458073
magnesium                       1.470740
total_phenols                   4.129054
flavanoids                      7.262948
nonflavanoid_phenols            1.876319
proanthocyanins                 2.023566
color_intensity                 3.311618
hue                             2.642213
od280/od315_of_diluted_wines    4.600768
proline                         2.719501
dtype: float64
```

- Let us remove 'od280/od315_of_diluted_wines' and 'total_phenols' and see if multicolinearity decreases for the respective variables

In [18]:
```
# Dropping columns from the training and test data

X_train = X_train.drop(["od280/od315_of_diluted_wines", "total_phenols"], axis=1)
X_test = X_test.drop(["od280/od315_of_diluted_wines", "total_phenols"], axis=1)

# Let's check the VIF of the predictors again
vif_series = pd.Series(
    [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])],
    index=X_train.columns,
    dtype=float,
)
print("VIF values: \n\n{}\n".format(vif_series))
```

```
VIF values:

const                   608.877123
alcohol                   2.417667
malic_acid                1.651086
ash                       2.300755
alcalinity_of_ash         2.458026
magnesium                 1.457152
flavanoids                3.923622
nonflavanoid_phenols      1.747677
proanthocyanins           1.950779
color_intensity           2.534900
hue                       2.638538
proline                   2.652465
dtype: float64
```

- VIF values are much lower indicating a decrease in multicollinearity

Creating a Logistic Regression Pipeline and Hyperparameter tuning it using GridSearchCV

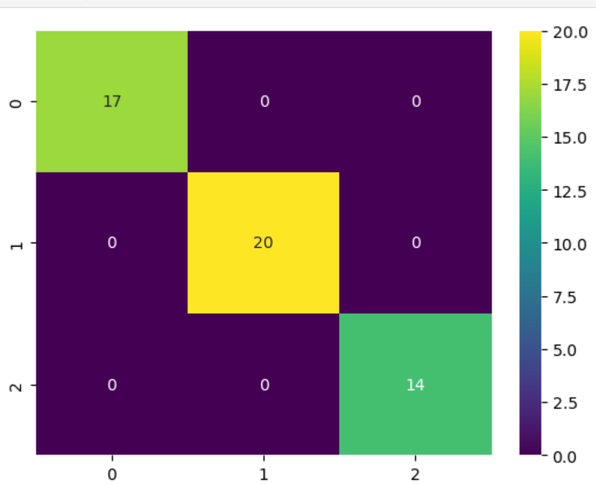
In [19]:
```
# Fitting Logistic Regression to the Training set
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

classifier_lr = LogisticRegression()
steps = [("scalar", StandardScaler()), ("model", LogisticRegression())]

lr_pipe = Pipeline(steps)
```

In [20]:
```
# Conducting Hyperparameter Tuning Using GridSearch
parameters = {
    "model__C": [1, 10, 100, 1000, 10000],
    "model__fit_intercept": [True],
    "model__multi_class": ["auto"],
    "model__tol": [0.0001],
    "model__solver": ["newton-cg", "lbfgs", "sag", "saga"],
    "model__n_jobs": [-1],
    "model__max_iter": [5000],
    "model__random_state": [42],
}
classifier_lr = GridSearchCV(lr_pipe, parameters, cv=5)
classifier_lr = classifier_lr.fit(X_train, y_train.ravel())
```

In [21]:
```
# Implementing model on training data and printing accuracy
y_pred_lr_train = classifier_lr.predict(X_train)
accuracy_lr_train = accuracy_score(y_train, y_pred_lr_train)
print("Training set: ", accuracy_lr_train)

# Implementing model on test/validation data and printing accuracy
y_pred_lr_test = classifier_lr.predict(X_test)
accuracy_lr_test = accuracy_score(y_test, y_pred_lr_test)
print("Test set: ", accuracy_lr_test)
```

```
Training set:  1.0
Test set:  1.0
```

In [22]:
```
# Creating a Confuion Matrix of our test data predictions
sns.heatmap(
    confusion_matrix(y_test, y_pred_lr_test), annot=True, cmap="viridis", fmt=".0f"
)
plt.show()
```



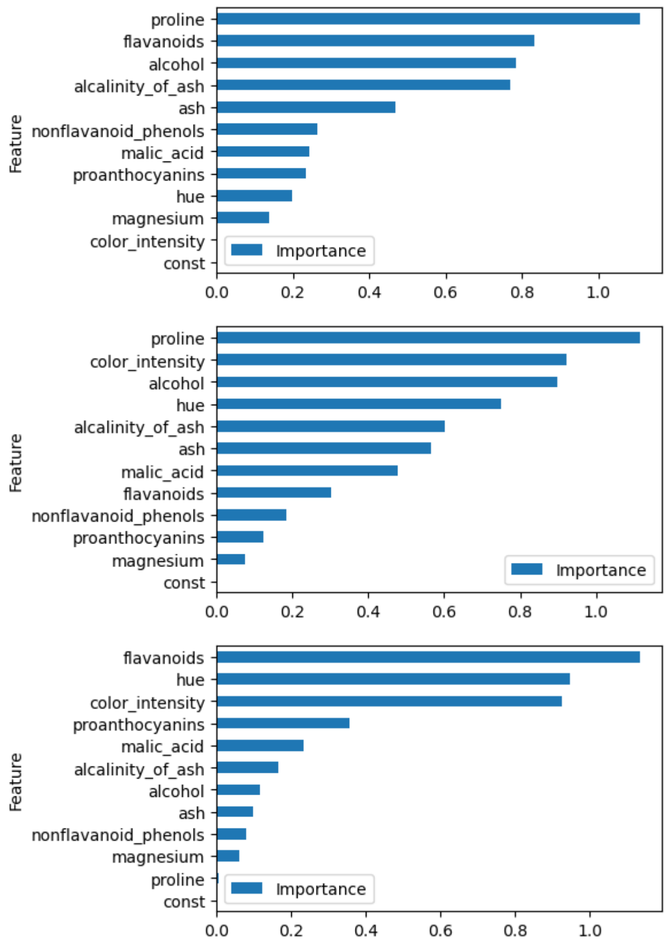
- 100% Acccuracy on the validation set is impressive and despite it seeming to be too good to be true, it is still within the realm of reason as from our EDA we could see there were several factors that allowed clear distiguishing between a majority of datapoints in the classes of our target variable

What were the most important variables according to our LR model in classifying the target variables

In [23]:
```
# Creating a Bar Chart showing the most important varibales for identifying Class 0 type of wine
coefficients = classifier_lr.best_estimator_.named_steps["model"].coef_[0]

feature_importance = pd.DataFrame(
    {"Feature": X_train.columns, "Importance": np.abs(coefficients)}
)
feature_importance = feature_importance.sort_values("Importance", ascending=True)
feature_importance.plot(x="Feature", y="Importance", kind="barh", figsize=(5, 3))


# Creating a Bar Chart showing the most important varibales for identifying Class 1 type of wine
```

```
coefficients1 = classifier_lr.best_estimator_.named_steps["model"].coef_[1]

feature_importance1 = pd.DataFrame(
    {"Feature": X_train.columns, "Importance": np.abs(coefficients1)}
)
feature_importance1 = feature_importance1.sort_values("Importance", ascending=True)
feature_importance1.plot(x="Feature", y="Importance", kind="barh", figsize=(5, 3))


# Creating a Bar Chart showing the most important varibales for identifying Class 2 type of wine
coefficients2 = classifier_lr.best_estimator_.named_steps["model"].coef_[2]

feature_importance2 = pd.DataFrame(
    {"Feature": X_train.columns, "Importance": np.abs(coefficients2)}
)
feature_importance2 = feature_importance2.sort_values("Importance", ascending=True)
feature_importance2.plot(x="Feature", y="Importance", kind="barh", figsize=(5, 3))
```

Out[23]: <Axes: ylabel='Feature'>



From these graphs we can infer some details about our features:

- Proline is a key contributor in identifying Class 0 and Class 1 wines
- Alochol, Flavanoids and Hue are also important factors when classifying our target variable
- Ash, Alkalinity of Ash, Proanthocyanins, Magnesium, Malic Acid and Non Flavanoid Phenols don't seem to greatly contribute to our predictive model, we can try removing some of them in our next iteration
- Color Intensity while being a key factor, follows the same trend as hue when it comes to importance in identifying the target variable. We can try removing it in the next iteration as well

Note:

- While I understand removal of multiple variables from a model can drastically alter the performance and is inadvisable. Some iterations were conducted outside of the one below with certain variables removed to see what would give the most reliable predictive models with minimal errors while also only accounting for important relevant features in our dataset. The model seen below is what has given high accuracy while only including a handful of independent variables

In [24]:
```
# Dropping various variables from the train and validation dataset
X_train = X_train.drop(
    [
        "ash",
```

```
        "magnesium",
        "nonflavanoid_phenols",
        "proanthocyanins",
        "alcalinity_of_ash",
        "malic_acid",
        "color_intensity",
        "const",
    ],
    axis=1,
)


X_test = X_test.drop(
    [
        "ash",
        "magnesium",
        "nonflavanoid_phenols",
        "proanthocyanins",
        "alcalinity_of_ash",
        "malic_acid",
        "color_intensity",
        "const",
    ],
    axis=1,
)
```

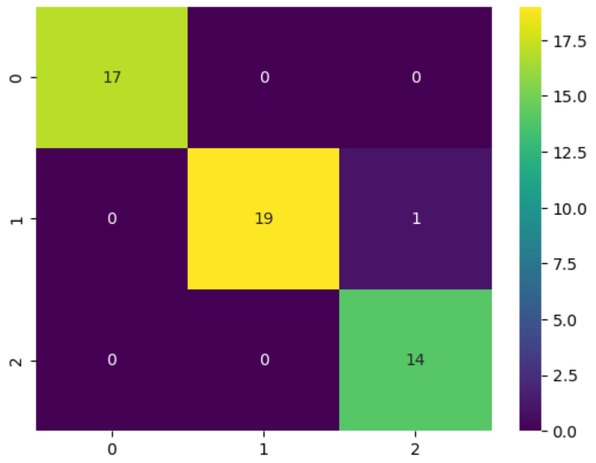
In [25]:
```
# Creating the LR Piperline
classifier_lr = LogisticRegression()
steps = [("scalar", StandardScaler()), ("model", LogisticRegression())]

lr_pipe = Pipeline(steps)
```

In [26]:
```
# Conducting Hyperparameter Tuning Using GridSearch
parameters = {
    "model__C": [1, 10, 100, 1000, 10000],
    "model__fit_intercept": [True],
    "model__multi_class": ["auto"],
    "model__tol": [0.0001],
    "model__solver": ["newton-cg", "lbfgs", "sag", "saga"],
    "model__n_jobs": [-1],
    "model__max_iter": [5000],
    "model__random_state": [42],
}
classifier_lr = GridSearchCV(lr_pipe, parameters, cv=5)
classifier_lr = classifier_lr.fit(X_train, y_train.ravel())
```

In [27]:
```
# Implementing model on training data and printing accuracy
y_pred_lr_train = classifier_lr.predict(X_train)
accuracy_lr_train = accuracy_score(y_train, y_pred_lr_train)
print("Training set: ", accuracy_lr_train)

# Implementing model on test/validation data and printing accuracy
y_pred_lr_test = classifier_lr.predict(X_test)
accuracy_lr_test = accuracy_score(y_test, y_pred_lr_test)
print("Test set: ", accuracy_lr_test)
```

```
Training set:  0.9829059829059829
Test set:  0.9803921568627451
```

In [28]:
```
# Creating a Confusion Matrix of our test data predictions
sns.heatmap(
    confusion_matrix(y_test, y_pred_lr_test), annot=True, cmap="viridis", fmt=".0f"
)
plt.show()
```

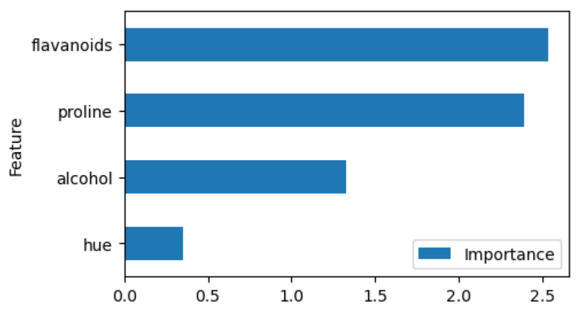


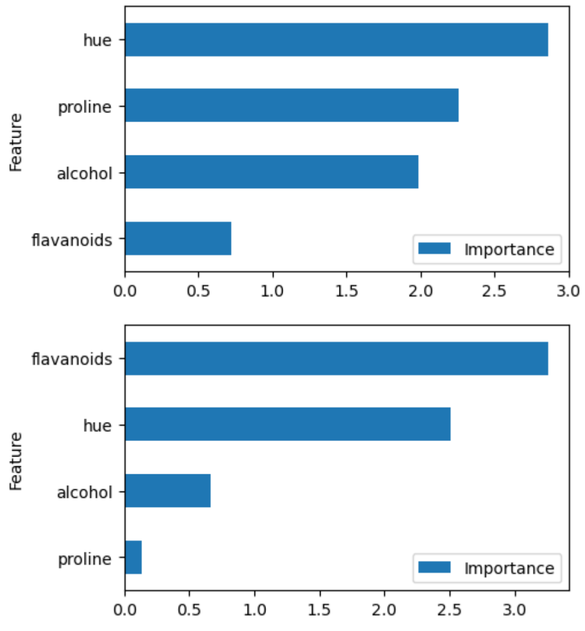
- While the model is not perfectly accurate anymore, the number of features has been significantly reduced to only include important factors affecting the target variable (from 13 to 4) with only a 2% decrease in accuracy

Checking which independent variables the model prioritizes as important in predicting our target variable

In [29]:
```
# Creating a Bar Chart showing the most important varibales for identifying Class 0 type of wine
coefficients = classifier_lr.best_estimator_.named_steps["model"].coef_[0]

feature_importance = pd.DataFrame(
    {"Feature": X_train.columns, "Importance": np.abs(coefficients)}
)
feature_importance = feature_importance.sort_values("Importance", ascending=True)
feature_importance.plot(x="Feature", y="Importance", kind="barh", figsize=(5, 3))


# Creating a Bar Chart showing the most important varibales for identifying Class 1 type of wine
coefficients1 = classifier_lr.best_estimator_.named_steps["model"].coef_[1]

feature_importance1 = pd.DataFrame(
    {"Feature": X_train.columns, "Importance": np.abs(coefficients1)}
)
feature_importance1 = feature_importance1.sort_values("Importance", ascending=True)
feature_importance1.plot(x="Feature", y="Importance", kind="barh", figsize=(5, 3))


# Creating a Bar Chart showing the most important varibales for identifying Class 2 type of wine
coefficients2 = classifier_lr.best_estimator_.named_steps["model"].coef_[2]

feature_importance2 = pd.DataFrame(
    {"Feature": X_train.columns, "Importance": np.abs(coefficients2)}
)
feature_importance2 = feature_importance2.sort_values("Importance", ascending=True)
feature_importance2.plot(x="Feature", y="Importance", kind="barh", figsize=(5, 3))
```

Out[29]:
```
<Axes: ylabel='Feature'>
```





Note on balancing Model Bias and if high variance of unseen dataset will greatly alter model performance:

- As a majority of variables have been removed and the regression model shows high accuracy with both the training and validation set after Hyperparameter tuning we can say this model has a low bias and is not overfitting the dataset
- Currently the comparison between the accuracy of our train and validation dataset shows minimal/no variance as both are sitting at 98% accuracy
- If our unseen data displays high variance compared to our current Train/Validation dataset, the model may perform slightly worse but would not crumble, as we're using the most relevant variables that are supported by both our model and initial EDA and have shown they help clearly distinguish a majority of our target variables from each other.
- In short: This ensures that even with a low bias, if the unseen data showed high variance compared to our original dataset, the model's variables were selected carefully to still allow reliable classification of our target variables.

Implementing the model to predict target variable for unseen dataset

In [30]:
```
# Creating X_unseen dataset with non-relevant columns removed to match column dimensions of the training/validation data
X_unseen = df_t.drop(
    [
        "ash",
        "magnesium",
        "nonflavanoid_phenols",
        "proanthocyanins",
        "alcalinity_of_ash",
        "malic_acid",
        "color_intensity",
        "od280/od315_of_diluted_wines",
        "total_phenols",
    ],
    axis=1,
)

# Predicting the target variable using our LR model
y_pred_lr_unseen = classifier_lr.predict(X_unseen)
print(y_pred_lr_unseen)

# Adding a column of our predicted variable to the dataset
df_t["Predicted Target"] = y_pred_lr_unseen
df_t
```

```
[0 0 2 0 1 0 1 2 1 2]
```

Out[30]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	Predicted Target
0	13.64	3.10	2.56	15.2	116	2.70	3.03	0.17	1.66	5.10	0.96	3.36	845	0
1	14.21	4.04	2.44	18.9	111	2.85	2.65	0.30	1.25	5.24	0.87	3.33	1080	0
2	12.93	2.81	2.70	21.0	96	1.54	0.50	0.53	0.75	4.60	0.77	2.31	600	2
3	13.73	1.50	2.70	22.5	101	3.00	3.25	0.29	2.38	5.70	1.19	2.71	1285	0
4	12.37	1.17	1.92	19.6	78	2.11	2.00	0.27	1.04	4.68	1.12	3.48	510	1
5	14.30	1.92	2.72	20.0	120	2.80	3.14	0.33	1.97	6.20	1.07	2.65	1280	0
6	12.00	3.43	2.00	19.0	87	2.00	1.64	0.37	1.87	1.28	0.93	3.05	564	1
7	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750	2
8	11.61	1.35	2.70	20.0	94	2.74	2.92	0.29	2.49	2.65	0.96	3.26	680	1
9	13.36	2.56	2.35	20.0	89	1.40	0.50	0.37	0.64	5.60	0.70	2.47	780	2

Conclusion:

- The EDA we conducted was helpful in understanding that a decent number of variables were able to distinguish a majority of a specific class of wines from the others.
- The correlation heatmap allowed me to see general relationships between the dependent and independent variables as well as show any signs of multicollinearity between independents.
- Checking the VIF values allowed me to deal with the suspected multicollinearity in variables and remove certain features.
- Building the LR Pipeline, tuning and fitting the model to our train and test dataset lead to a 100% accuracy in both, while this seemed optimistic, the EDA conducted beforehand showed that distinguishing the target variables was relatively simple. Removing certain features to simplify the model, avoid overfitting and improve computational ability makes sense at this stage.
- Checking what features the model deemed important and removing the lower prioritized variables based off it allowed us to create a 4 variable predictive model (down from 13 in the original dataset) which from multiple offline iterations seemed to create the simplest model without drastically affecting performance and lead to a 98% accuracy in train and test data.
- Looking at the bias and variance of the model, I inferred (with evidence from the EDA) that this model is simple enough to have relatively low bias, while also robust enough to operate well under potential high variance of unseen data.
- The model was successfully implemented on the unseen data however since the true target data is not available, I cannot predict the exact accuracy and performance of this model.