Homework 1

Consider the `wine` dataset, the response measures wine quality and takes discrete values between 3 and 8. The input contains various measurable attributes for each wine (e.g. sulphates, acidity). The goal of this exercise is to predict wine quality category based on those attributes. In addition to `numpy` and `pandas`, we will use functions in the `sklearn` and `pygam` modules.

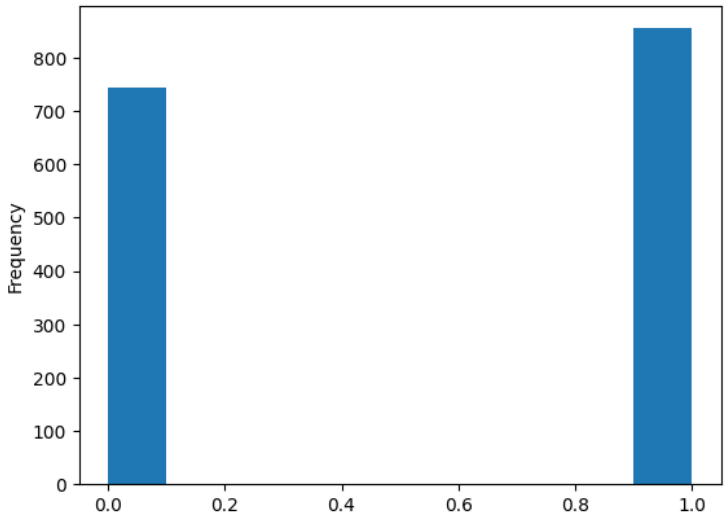
In [2]:
```
# Modules
import numpy as np
import pandas as pd
import pygam

from matplotlib import pyplot as plt
from sklearn import datasets, linear_model, metrics, model_selection, pipeline, preprocessing, utils
```

1. Load the Wine dataset and examine the size of the dataset, the variable types, and produce some descriptive statistics.

In [3]:
```
# Loads dataset
X, y = pd.read_pickle('homework1_data.pkl')
```

2. For simplicity, we will recode the response as 0 for wines that have a score below or equal to 5 and 1 otherwise. Plot the distribution of the response.

In [4]:
```
y_updated = y.apply(lambda x: 0 if x <= 5 else 1)
y_updated.plot(kind="hist")
plt.show()
```



3. Split the data into a training (75%) and a test (25%) sample, while mainting the class proportions in the training and the test samples (see `sklearn.model_selection`). Remember to make the sampling reproductible.

In [5]:
```
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y_updated, test_size=0.25, random_state=480, stratify=y_updated)
```

4. Fit a logistic regression model to the training data and evaluate its performance on the test data (see `sklearn.linear_model`). Set the number of optimisation iterations `max_iter=10000` for the model to converge.

In [6]:
```
logistic_reg = linear_model.LogisticRegression(max_iter=10000)
logistic_reg.fit(X_train, y_train)
pred = logistic_reg.predict(X_test)
accuracy = metrics.accuracy_score(y_test, pred)
print("Accuracy is", accuracy)
```

```
Accuracy is 0.73
```

5. Compute the bootstrapped standard errors for the parameters using 100 bootstrap of size 500 (see `sklearn.utils`). Provide an interpretation for one of the parameter estimates.

In [7]:
```
n = 100
size = 500
btsrp_params = np.zeros((n, logistic_reg.coef_[0].shape[0]))
for i in range(n):
    sample = utils.resample(X_train, n_samples=size)
    logistic_reg_btsrp = linear_model.LogisticRegression(max_iter=10000)
    logistic_reg_btsrp.fit(sample, y_train.loc[sample.index])
    btsrp_params[i, :] = logistic_reg_btsrp.coef_[0]
se = np.std(btsrp_params, axis=0)
se
```

Out[7]:
```
array([0.08740589, 0.40690133, 0.50997011, 0.09303682, 0.31701517,
       0.01288052, 0.00494137, 0.01332532, 0.44671311, 0.49420021,
       0.14434567])
```

In [8]:
```
btsrp_params
```

Out[8]:
```
array([[ 0.22815831, -2.82661329, -1.67986015, ..., -0.7686476 ,
         0.93565397,  0.93395361],
       [ 0.10672051, -3.05568058, -0.35835491, ..., -0.58408589,
         1.73476606,  0.66676198],
       [ 0.05542393, -2.80041666, -0.52933831, ..., -1.15513666,
         1.52450688,  0.90255799],
       ...,
       [ 0.02890704, -2.43546801, -0.4957988 , ...,  0.18575513,
         2.45822848,  0.75946163],
       [-0.13401587, -2.10564815,  0.60523668, ..., -0.43423617,
         1.76713158,  0.83248877],
       [ 0.0648418 , -1.86401711, -0.44282368, ..., -0.06615071,
         1.48591417,  1.12653219]])
```

The interpretation of a parameter estimate of logistic regression is the change of the log-probability of the target variable for 1 unit change in an independent variable, while holding all other independent variable constant. For example, the 0.136931 in the cell(0,0) means that for 1 unit change in fixed acidity (corresponding to column 0 here) will result in 0.136931 increase in the log-probability of the target variable, while holding all other variables constant.

6. Show whether the model overfits the training data.

In [9]:
```
logistic_reg = linear_model.LogisticRegression(max_iter=10000)
logistic_reg.fit(X_train, y_train)
train_pred = logistic_reg.predict(X_train)
accuracy = metrics.accuracy_score(y_train, train_pred)
print(f"The Training accuracy is: {accuracy}")
test_pred = logistic_reg.predict(X_test)
test_accuarcy = metrics.accuracy_score(y_test, test_pred)
print(f"The test accuracy is: {test_accuarcy}")
```

```
The Training accuracy is: 0.7539616346955796
The test accuracy is: 0.73
```

The training set and the test set has similar accuracy so the model is not overfitting.

7. Estimate the prediction accuracy using 5-fold cross-validation (see `sklearn.model_selection`). How does this statistic compare to the test error?

In [10]:
```
cv_scores = model_selection.cross_val_score(logistic_reg, X_train, y_train, cv=5)
cv_acc = np.mean(cv_scores)
print(f"Cross-validation accuracy is {cv_acc}")
```

```
Cross-validation accuracy is 0.7505927475592747
```

The cross-validation accuracy is slightly higher than the test accuracy and nearly the same as the training accuracy.

8. Compute the confusion matrix for the test sample (see sklearn.metrics). Comment on the performance of the model.

In [11]:
```
confusion_matrix = metrics.confusion_matrix(y_test, test_pred)
print(confusion_matrix)
```

```
[[126  60]
 [ 48 166]]
```

From the confusion matrix, we can see that the true negative is 126 observations, True positive is 166 observations, false positive is 60 observations, and 48 false negative observations. Then, we can also calculate the accuracy and precision which are $\frac{126+166}{(126+60+48+166)} = 0.72$ and $\frac{166}{166+60} = 0.73$, respectively. The model has moderate performance since the accuracy and precision are around 70%.

9. Using the `pygam` module, fit a Generalised Linear Model for classification. Compute the training and test accuracy of the GAM model (see `sklearn.metrics`). Comment on the results.

In [12]:
```
gam_model = pygam.LogisticGAM(max_iter=10000)
gam_model.fit(X_train, y_train)

gam_pred_train = gam_model.predict(X_train)
gam_pred_test = gam_model.predict(X_test)

train_acc_gam = metrics.accuracy_score(y_train, gam_pred_train)
test_acc_gam = metrics.accuracy_score(y_test, gam_pred_test)

print("Training accuracy is {}, and Test accuracy is {}".format(train_acc_gam, test_acc_gam))
```

```
Training accuracy is 0.8023352793994996, and Test accuracy is 0.7375
```

10. Use grid-search to find a good value for the regularisation parameters (see the `gridsearch` method of the model). Could you improve on the previous fit?

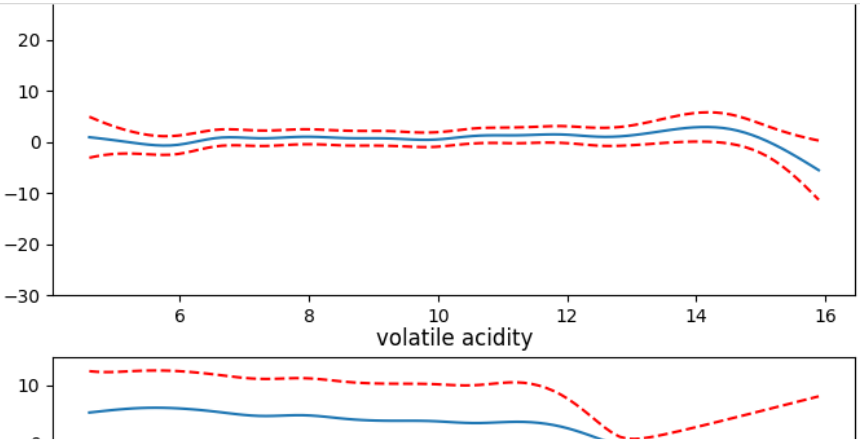
In [13]:
```
param_grid = {"lam": [0.05, 0.09, 0.1, 0.9, 1, 5, 15, 25]}
gs = model_selection.GridSearchCV(gam_model, param_grid, scoring="recall")
gs.fit(X_train, y_train)
optim_penalty = gs.best_params_["lam"]
refit_gam = pygam.LogisticGAM(lam=optim_penalty, max_iter=10000)
refitted = refit_gam.fit(X_train, y_train)

gs_test_pred = refit_gam.predict(X_test)

test_acc_gam_refit = metrics.accuracy_score(y_test, gs_test_pred)

print("After adding regularization {}, the test accuracy is {}".format(optim_penalty, test_acc_gam_refit))
```

```
After adding regularization 0.05, the test accuracy is 0.7475
```

After adding the regularization term, the test accuracy is increased slightly by 1%.

11. Display the partial dependence plots for each variable.

In [13]:
```
variables = X_train.columns[0:11]
plt.figure()
fig, axs = plt.subplots(11,1,figsize=(8, 40))
for i, ax in enumerate(axs):
    XX = refitted.generate_X_grid(term=i)
    ax.plot(XX[:, i], refitted.partial_dependence(term=i, X=XX))
    ax.plot(XX[:, i], refitted.partial_dependence(term=i, X=XX, width=.95)[1], c='r', ls='--')
    if i == 0:
        ax.set_ylim(-30,30)
    ax.set_title(variables[i])
plt.show()

### NOTE: Since the notebook is saved as PDF, some of the graph is not displayed here.
### I will include the full list of graphs in separate picture. Sorry for the inconvenience
```



In [13]: