Importing the required packages

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
```

Importing the data needed

```
df = pd.read_csv("annthyroid-training.csv", header=None)

df
```

	0	1	2	3	4	5	6	7	8	9	...	12	13	14	15	16	17	18	19	20	21
0	0.750000	1	0	1	1	1	1	1	0	1	...	1	1	1	1	0.001132	0.080780	0.197324	0.300926	0.225000	1
1	0.239583	1	1	1	1	1	1	1	1	1	...	1	1	1	1	0.000472	0.164345	0.235786	0.537037	0.165625	1
2	0.479167	1	1	1	1	1	1	1	1	1	...	1	1	1	1	0.003585	0.130919	0.167224	0.527778	0.118750	1
3	0.656250	0	1	1	1	1	1	1	1	1	...	1	1	1	1	0.001698	0.091922	0.125418	0.337963	0.129688	1
4	0.229167	1	1	1	1	1	1	1	1	1	...	1	1	1	1	0.000472	0.142061	0.229097	0.337963	0.235938	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6995	0.875000	1	1	1	1	1	1	1	1	1	...	1	1	1	1	0.060377	0.050696	0.088629	0.333333	0.093750	-1
6996	0.218750	0	1	1	1	1	1	1	1	1	...	1	1	1	1	0.004340	0.097493	0.239130	0.347222	0.243750	1
6997	0.229167	1	1	1	1	1	1	1	1	1	...	1	1	1	1	0.005094	0.109192	0.103679	0.291667	0.121875	1
6998	0.531250	0	1	1	1	1	1	1	1	1	...	1	1	1	0	0.002830	0.109192	0.160535	0.328704	0.170313	1
6999	0.781250	1	1	1	1	1	1	1	1	1	...	1	1	1	1	0.001604	0.109192	0.153846	0.162037	0.273438	1

7000 rows × 22 columns

Splitting the data into a testing set (20% of the data) and a training set (80% of the data)

```
tr_df = df.iloc[:int(round(len(df)*.8,1)),:]
te_df = df.iloc[int(round(len(df)*.8,1)):,:]
```

	0	1	2	3	4	5	6	7	8	9	...	12	13	14	15	16	17	18	19	20	21
5600	0.750000	1	1	1	1	1	1	1	1	1	...	1	1	1	1	0.002075	0.197772	0.265886	0.240741	0.360938	1
5601	0.479167	1	0	1	1	1	1	1	1	1	...	1	1	1	1	0.000472	0.130919	0.247492	0.430556	0.209375	1
5602	0.479167	0	1	1	1	1	1	1	1	1	...	1	1	1	1	0.002264	0.113092	0.172241	0.333333	0.181250	1
5603	0.343750	1	1	1	1	1	1	1	1	1	...	1	1	1	1	0.003585	0.147632	0.210702	0.365741	0.204766	1
5604	0.343750	0	1	1	1	1	1	1	1	1	...	1	1	1	1	0.018868	0.119777	0.140468	0.342593	0.145312	-1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6995	0.875000	1	1	1	1	1	1	1	1	1	...	1	1	1	1	0.060377	0.050696	0.088629	0.333333	0.093750	-1
6996	0.218750	0	1	1	1	1	1	1	1	1	...	1	1	1	1	0.004340	0.097493	0.239130	0.347222	0.243750	1
6997	0.229167	1	1	1	1	1	1	1	1	1	...	1	1	1	1	0.005094	0.109192	0.103679	0.291667	0.121875	1
6998	0.531250	0	1	1	1	1	1	1	1	1	...	1	1	1	0	0.002830	0.109192	0.160535	0.328704	0.170313	1
6999	0.781250	1	1	1	1	1	1	1	1	1	...	1	1	1	1	0.001604	0.109192	0.153846	0.162037	0.273438	1

1400 rows × 22 columns

Training the data

```
x = tr_df.iloc[:,:-1]
x.head()
```

	0	1	2	3	4	5	6	7	8	9	...	11	12	13	14	15	16	17	18	19	20
0	0.750000	1	0	1	1	1	1	1	0	1	...	1	1	1	1	1	0.001132	0.080780	0.197324	0.300926	0.225000
1	0.239583	1	1	1	1	1	1	1	1	1	...	1	1	1	1	1	0.000472	0.164345	0.235786	0.537037	0.165625
2	0.479167	1	1	1	1	1	1	1	1	1	...	1	1	1	1	1	0.003585	0.130919	0.167224	0.527778	0.118750
3	0.656250	0	1	1	1	1	1	1	1	1	...	1	1	1	1	1	0.001698	0.091922	0.125418	0.337963	0.129688
4	0.229167	1	1	1	1	1	1	1	1	1	...	1	1	1	1	1	0.000472	0.142061	0.229097	0.337963	0.235938

5 rows × 21 columns

```
y = tr_df.T.tail(1).T.values.ravel()
y
```

```
array([1., 1., 1., ..., 1., 1., 1.])
```

Applying a logistic regression model

```
model = LogisticRegression()
model.fit(x, y)
```

```
/Users/raph/virtualenvs/venv/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:444: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
▾ LogisticRegression
LogisticRegression()
```

Preparing the testing data under the chosen model

```
x_validation = te_df.iloc[:,:-1]

x_validation.head()
```

	0	1	2	3	4	5	6	7	8	9	...	11	12	13	14	15	16	17	18	19	20
5600	0.750000	1	1	1	1	1	1	1	1	1	...	1	1	1	1	1	0.002075	0.197772	0.265886	0.240741	0.360938
5601	0.479167	1	0	1	1	1	1	1	1	1	...	1	1	1	1	1	0.000472	0.130919	0.247492	0.430556	0.209375
5602	0.479167	0	1	1	1	1	1	1	1	1	...	0	1	1	1	1	0.002264	0.113092	0.172241	0.333333	0.181250
5603	0.343750	1	1	1	1	1	1	1	1	1	...	1	1	1	1	1	0.003585	0.147632	0.210702	0.365741	0.204766
5604	0.343750	0	1	1	1	1	1	1	1	1	...	0	1	1	1	1	0.018868	0.119777	0.140468	0.342593	0.145312

5 rows × 21 columns

```
y_validation = te_df.T.tail(1).values.ravel()

y_validation
```

```
array([1., 1., 1., ..., 1., 1., 1.])
```

Predicting the outcomes using the testing data

```
y_predictions = model.predict(x_validation)

y_predictions
```

```
array([1., 1., 1., ..., 1., 1., 1.])
```

Creating a confusion matrix

```
from sklearn.metrics import confusion_matrix

tn, fp, fn, tp = confusion_matrix(y_validation, y_predictions).ravel()

print("True Negative: {}".format(tn))
print("False Positive: {}".format(fp))
print("False Negative: {}".format(fn))
print("True Positive: {}".format(tp))
```

```
True Negative: 19
False Positive: 81
False Negative: 2
True Positive: 1298
```

Evaluation metrics

Accuracy

This shows the number of accounts that were correctly classified by the model

$$\frac{TP+TN}{TP+TN+FP+FN}$$

Sensitivity

This shows how many of the bad accounts were correcly classified by the model

$$\frac{TP}{TP+FN}$$

Specificity

This shows how many of the actual good accounts were correctly identified by the model

$$\frac{TN}{TN+FP}$$

F1-Score

This shows the harmonic mean of precision and the sensitivity

$$\frac{2*precison*sensitivity}{sensitivity+precison}$$

Precision

This shows how accurate the model is when it is trying to identiy bad accounts

$$\frac{TP}{TP+FP}$$

```
# Accuracy
accuracy = round((tp+tn)/(tp+tn+fp+fn),2)

# Sensitivity
sensitivity = round(tp/(tp+fn),2)

# Specificity
specificity = round(tn/(tn+fp),2)

# F1-Score
precision = round(tp/(tp+fp))
f1_score = (2*precision*sensitivity)/(precision+sensitivity)

print("Accuracy: {}%".format(accuracy*100))
print("Sensitivity: {}%".format(sensitivity*100))
print("Specificity: {}%".format(specificity*100))
print("F1 Score: {}".format(f1_score))
```

```
Accuracy: 94.0%
Sensitivity: 100.0%
Specificity: 19.0%
F1 Score: 1.0
```