Credit Card Application

Commercial banks receive a lot of applications for credit cards. Many of them get rejected for many reasons, like high loan balances, low income levels, or too many inquiries on an individual's credit report, for example. Manually analyzing these applications is mundane, error-prone, and time-consuming (and time is money!). Luckily, this task can be automated with the power of machine learning and pretty much every commercial bank does so nowadays. In this notebook, we will build an automatic credit card approval predictor using machine learning techniques, just like the real banks do.

1.Loading and viewing dataset.

In [1]:
```
import pandas as pd
```

In [2]:
```
apps = pd.read_csv('downloads/credit+approval/crx.data',header=None)
apps.head()
```

Out[2]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	f	g	00202	0	+
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	00043	560	+
2	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	00280	824	+
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	00100	3	+
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	00120	0	+

2.Inspecting the applications.

The probable features in a typical credit card application are Gender, Age, Debt, Married, BankCustomer, EducationLevel, Ethnicity, YearsEmployed, PriorDefault, Employed, CreditScore, DriversLicense, Citizen, ZipCode, Income and finally the ApprovalStatus. This gives us a pretty good starting point, and we can map these features with respect to the columns in the output.

In [3]:
```
# Print summary statistics
apps_description = apps.describe()
print(apps_description)

print('\n')

# Print DataFrame information
apps_info = apps.info()
print(apps_info)

print('\n')
```

```
# Inspect missing values in the dataset
print(apps.tail(n=17))
```

```
                2           7          10             14
count  690.000000  690.000000  690.00000     690.000000
mean     4.758725    2.223406    2.40000    1017.385507
std      4.978163    3.346513    4.86294    5210.102598
min      0.000000    0.000000    0.00000       0.000000
25%      1.000000    0.165000    0.00000       0.000000
50%      2.750000    1.000000    0.00000       5.000000
75%      7.207500    2.625000    3.00000     395.500000
max     28.000000   28.500000   67.00000  100000.000000


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       690 non-null    object 
 1   1       690 non-null    object 
 2   2       690 non-null    float64
 3   3       690 non-null    object 
 4   4       690 non-null    object 
 5   5       690 non-null    object 
 6   6       690 non-null    object 
 7   7       690 non-null    float64
 8   8       690 non-null    object 
 9   9       690 non-null    object 
 10  10      690 non-null    int64  
 11  11      690 non-null    object 
 12  12      690 non-null    object 
 13  13      690 non-null    object 
 14  14      690 non-null    int64  
 15  15      690 non-null    object 
dtypes: float64(2), int64(2), object(12)
memory usage: 86.4+ KB
None


    0      1       2  3  4   5   6      7  8  9   10 11 12     13   14 15
673  ?  29.50   2.000  y  p   e   h  2.000  f  f   0  f  g  00256   17  -
674  a  37.33   2.500  u  g   i   h  0.210  f  f   0  f  g  00260  246  -
675  a  41.58   1.040  u  g  aa   v  0.665  f  f   0  f  g  00240  237  -
676  a  30.58  10.665  u  g   q   h  0.085  f  t  12  t  g  00129    3  -
677  b  19.42   7.250  u  g   m   v  0.040  f  t   1  f  g  00100    1  -
678  a  17.92  10.210  u  g  ff  ff  0.000  f  f   0  f  g  00000   50  -
679  a  20.08   1.250  u  g   c   v  0.000  f  f   0  f  g  00000    0  -
680  b  19.50   0.290  u  g   k   v  0.290  f  f   0  f  g  00280  364  -
681  b  27.83   1.000  y  p   d   h  3.000  f  f   0  f  g  00176  537  -
682  b  17.08   3.290  u  g   i   v  0.335  f  f   0  t  g  00140    2  -
683  b  36.42   0.750  y  p   d   v  0.585  f  f   0  f  g  00240    3  -
684  b  40.58   3.290  u  g   m   v  3.500  f  f   0  t  s  00400    0  -
685  b  21.08  10.085  y  p   e   h  1.250  f  f   0  f  g  00260    0  -
686  a  22.67   0.750  u  g   c   v  2.000  f  t   2  t  g  00200  394  -
687  a  25.25  13.500  y  p  ff  ff  2.000  f  t   1  t  g  00200    1  -
688  b  17.92   0.205  u  g  aa   v  0.040  f  f   0  f  g  00280  750  -
689  b  35.00   3.375  u  g   c   h  8.290  f  f   0  t  g  00000    0  -
```

3.Splitting the dataset into train and test sets.

Ideally, no information from the test data should be used to preprocess the training data or should be used to direct the training process of a machine learning model. Hence, we first split the data and then preprocess it. Also, features like DriversLicense and ZipCode are not as important as the other features in the dataset for predicting credit card approvals. To get a better sense, we can measure their statistical correlation to the labels of the dataset

In [4]:
```
# Import train_test_split
from sklearn.model_selection import train_test_split
# Drop the features 11 and 13
apps = apps.drop([11,13],axis=1)

# Split into train and test sets
apps_train, apps_test = train_test_split(apps,test_size=0.33, random_state=42)
```

4.Handling Missing values(part 1)

The dataset has missing values, which we'll take care of in this task. The missing values in the dataset are labeled with '?', which can be seen in the last cell's output of the second task. Now, let's temporarily replace these missing value question marks with NaN.

In [5]:
```
# Import numpy
import numpy as np

# Replace the '?'s with NaN in the train and test sets
apps_train = apps_train.replace('?','NaN')
apps_test = apps_test.replace('?','NaN')
```

5.Handling Missing Values(part 2)

While ignoring the missing values our machine learning model may miss out on information about the dataset that may be useful for its training. Then, there are many models which cannot handle missing values implicitly such as Linear Discriminant Analysis (LDA).So, to avoid this problem, we are going to impute the missing values with a strategy called mean imputation.

In [6]:
```
# Impute the missing values with mean imputation
apps_train.fillna(apps_train.mean(), inplace=True)
apps_test.fillna(apps_train.mean(), inplace=True)

# Count the number of NaNs in the datasets and print the counts to verify
print(apps_train.isnull().sum())
print(apps_test.isnull().sum())
```

```
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
12    0
14    0
15    0
dtype: int64
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
12    0
14    0
15    0
dtype: int64
```

```
C:\Users\Melvin Chalwa\AppData\Local\Temp\ipykernel_9108\927223805.py:2: FutureWarnin
g: The default value of numeric_only in DataFrame.mean is deprecated. In a future ver
sion, it will default to False. In addition, specifying 'numeric_only=None' is deprec
ated. Select only valid columns or specify the value of numeric_only to silence this
warning.
  apps_train.fillna(apps_train.mean(), inplace=True)
C:\Users\Melvin Chalwa\AppData\Local\Temp\ipykernel_9108\927223805.py:3: FutureWarnin
g: The default value of numeric_only in DataFrame.mean is deprecated. In a future ver
sion, it will default to False. In addition, specifying 'numeric_only=None' is deprec
ated. Select only valid columns or specify the value of numeric_only to silence this
warning.
  apps_test.fillna(apps_train.mean(), inplace=True)
```

6.Handling Missing Values(part 3)

There are still some missing values to be imputed for columns 0, 1, 3, 4, 5, 6 and 13. All of these columns contain non-numeric data and this is why the mean imputation strategy would not work here. This needs a different treatment. We are going to impute these missing values with the most frequent values as present in the respective columns. This is good practice when it comes to imputing missing values for categorical data in general.

In [7]:
```
# Iterate over each column of apps_train
for col in apps_train.columns:
    # Check if the column is of object type
    if apps_train[col].dtypes == 'object':
        # Impute with the most frequent value
```

```
        apps_train = apps_train.fillna(apps_train[col].value_counts().index[0])
        apps_test = apps_test.fillna(apps_train[col].value_counts().index[0])

# Count the number of NaNs in the dataset and print the counts to verify
print(apps_train.isnull().sum())
print(apps_test.isnull().sum())
```

```
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
12    0
14    0
15    0
dtype: int64
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
12    0
14    0
15    0
dtype: int64
```

7.Preprocessing Data(part 1)

Convert all the non-numeric values into numeric ones. We do this because not only does it results in a faster computation but also many machine learning models especially the ones developed using scikit-learn require the data to be in a strictly numeric format. We will do this by using the get_dummies() method from pandas.

In [8]:
```
# Convert the categorical features in the train and test sets to numeric.
apps_train = pd.get_dummies(apps_train)
apps_test = pd.get_dummies(apps_test)

# Reindex the columns of the test set aligning with the train set
apps_test = apps_test.reindex(columns=apps_train.columns, fill_value=0)
```

8.Preprocessing Data(part 2)

Rescale all the values to a range of 0-1 before fitting a machine learning model to the data.

In [9]:
```
# Import MinMaxScaler
from sklearn.preprocessing import MinMaxScaler

# Segregate features and labels into separate variables
X_train, y_train = apps_train.iloc[:, :-1].values, apps_train.iloc[:, [-1]].values
X_test, y_test = apps_test.iloc[:, :-1].values, apps_test.iloc[:, [-1]].values

# Instantiate MinMaxScaler and use it to rescale X_train and X_test
scaler = MinMaxScaler(feature_range=(0,1))
rescaledX_train = scaler.fit_transform(X_train)
rescaledX_test = scaler.transform(X_test)
```

9.Fitting a Logistic Rgression Model.

According to UCI, our dataset contains more instances that correspond to "Denied" status than instances corresponding to "Approved" status. Specifically, out of 690 instances, there are 383 (55.5%) applications that got denied and 307 (44.5%) applications that got approved. A good machine learning model should be able to accurately predict the status of the applications with respect to these statistics.

Which model should we pick? A question to ask is: are the features that affect the credit card approval decision process correlated with each other? so we'll rely on our intuition that they indeed are correlated for now. Because of this correlation, we'll take advantage of the fact that generalized linear models perform well in these cases.

In [10]:
```
# Import LogisticRegression
from sklearn.linear_model import LogisticRegression

# Instantiate a LogisticRegression classifier with default parameter values
logreg = LogisticRegression()

# Fit logreg to the train set
logreg.fit(rescaledX_train,y_train)
```

```
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[10]:
```
LogisticRegression()
```

10.Making Predictions and Evaluating Perfomance.

But how well does our model perform? We will now evaluate our model on the test set with respect to classification accuracy. But we will also take a look the model's confusion matrix. In the case of predicting credit card applications, it is important to see if our machine learning model is equally capable of predicting approved and denied status.

In [11]:
```
# Import confusion_matrix
from sklearn.metrics import confusion_matrix

# Use logreg to predict instances from the test set and store it
y_pred = logreg.predict(rescaledX_test)

# Get the accuracy score of logreg model and print it
print("Accuracy of logistic regression classifier: ",logreg.score(rescaledX_test,y_tes

# Print the confusion matrix of the logreg model
confusion_matrix(y_test,y_pred)
```

Out[11]:
```
Accuracy of logistic regression classifier:  1.0
array([[103,   0],
       [  0, 125]], dtype=int64)
```

11.Grid Searching.

Our model was pretty good!it was able to yield an accuracy score of 100%. For the confusion matrix, the first element of the of the first row of the confusion matrix denotes the true negatives meaning the number of denied applications predicted by the model correctly. And the last element of the second row of the confusion matrix denotes the true positives meaning the number of approved applications predicted by the model correctly. But if we hadn't got a perfect score what's to be done?. We can perform a grid search of the model parameters to improve the model's ability to predict credit card approvals.

In [12]:
```
# Import GridSearchCV
from sklearn.model_selection import GridSearchCV

# Define the grid of values for tol and max_iter
tol = [0.01,0.001,0.0001]
max_iter = [100,150,200]

# Create a dictionary where tol and max_iter are keys and the lists of their values ar
param_grid = dict(tol=tol, max_iter=max_iter)
```

12.Finding The Best Perfoming Model.

We will instantiate GridSearchCV() with our earlier logreg model with all the data we have. We will also instruct GridSearchCV() to perform a cross-validation of five folds.

In [13]:
```
# Instantiate GridSearchCV with the required parameters
grid_model = GridSearchCV(estimator=logreg, param_grid=param_grid, cv=5)

# Fit grid_model to the data
grid_model_result = grid_model.fit(rescaledX_train,y_train)

# Summarize results
best_score, best_params = grid_model_result.best_score_, grid_model_result.best_params
print("Best: %f using %s" % (best_score,best_params))

# Extract the best model and evaluate it on the test set
```

```
best_model = grid_model_result.best_estimator_
print("Accuracy of logistic regression classifier: ", best_model.score(rescaledX_test,
```

```
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
C:\Users\Melvin Chalwa\New folder\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Best: 1.000000 using {'max_iter': 100, 'tol': 0.01}
Accuracy of logistic regression classifier:  1.0
```

In [ ]: