Task 2

Credit / Home Loans - AutoML vs Bespoke ML

Standard Bank is embracing the digital transformation wave and intends to use new and exciting technologies to give their customers a complete set of services from the convenience of their mobile devices. As Africa's biggest lender by assets, the bank aims to improve the current process in which potential borrowers apply for a home loan. The current process involves loan officers having to manually process home loan applications. This process takes 2 to 3 days to process upon which the applicant will receive communication on whether or not they have been granted the loan for the requested amount. To improve the process Standard Bank wants to make use of machine learning to assess the credit worthiness of an applicant by implementing a model that will predict if the potential borrower will default on his/her loan or not, and do this such that the applicant receives a response immediately after completing their application.

You will be required to follow the data science lifecycle to fulfill the objective. The data science lifecycle (https://www.datascience-pm.com/crisp-dm-2/ (https://www.datascience-pm.com/crisp-dm-2/)) includes:

- Business Understanding
- Data Understanding
- Data Preparation
- Modelling
- Evaluation
- Deployment.

You now know the CRoss Industry Standard Process for Data Mining (CRISP-DM), have an idea of the business needs and objectivess, and understand the data. Next is the tedious task of preparing the data for modeling, modeling and evaluating the model. Luckily, just like EDA the first of the two phases can be automated. But also, just like EDA this is not always best.

In this task you will be get a taste of AutoML and Bespoke ML. In the notebook we make use of the library auto-sklearn/autosklearn (https://www.automl.org/automl/auto-sklearn/ (https://www.automl.org/automl/auto-sklearn/)) for AutoML and sklearn for ML. We will use train one machine for the traditional approach and you will be required to change this model to any of the models that exist in sklearn. The model we will train will be a Logistic Regression. Parts of the data preparation will be omitted for you to do, but we will provide hints to lead you in the right direction.

The data provided can be found in the Resources folder as well as (https://www.kaggle.com/datasets/altruistdelhite04/loan-prediction-problem-dataset (https://www.kaggle.com/datasets/altruistdelhite04/loan-prediction-problem-dataset)).

- train will serve as the historical dataset that the model will be trained on and,
- test will serve as unseen data we will predict on, i.e. new ('future') applicants.

Part One

There are many AutoEDA Python libraries out there which include:

- dtale (https://dtale.readthedocs.io/en/latest/ (https://dtale.readthedocs.io/en/latest/))
- pandas profiling (https://pandas-profiling.ydata.ai/docs/master/index.html (https://pandas-profiling.ydata.ai/docs/master/index.html))
- autoviz (https://readthedocs.org/projects/autoviz/ (https://readthedocs.org/projects/autoviz/))
- sweetviz (https://pypi.org/project/sweetviz/ (https://pypi.org/project/sweetviz/))

and many more. In this task we will use Sweetviz.. You may be required to use bespoke EDA methods.

The Home Loans Department manager wants to know the following:

1. An overview of the data. (HINT: Provide the number of records, fields and their data types. Do for both).
2. What data quality issues exist in both train and test? (HINT: Comment any missing values and duplicates)
3. How do the the loan statuses compare? i.e. what is the distrubition of each?
4. How do women and men compare when it comes to defaulting on loans in the historical dataset?
5. How many of the loan applicants have dependents based on the historical dataset?
6. How do the incomes of those who are employed compare to those who are self employed based on the historical dataset?
7. Are applicants with a credit history more likely to default than those who do not have one?
8. Is there a correlation between the applicant's income and the loan amount they applied for?

Part Two

Run the AutoML section and then fill in code for the traditional ML section for the the omitted cells.

Please note that the notebook you submit must include the analysis you did in Task 2.

Import Libraries

In [1]:
```
# !pip install sweetviz
# # uncomment the above if you need to install the library
# !pip install auto-sklearn
# # uncomment the above if you need to install the library
```

In [2]:
```
# !pip install --upgrade scipy
```

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sweetviz
import autosklearn.classification
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer
```

Import Datasets

In [4]:
```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

In [4]:

Part One

EDA

In [5]:
```
train.head()
```

Out[5]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	

In [6]:
```
test.head()
```

Out[6]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan
0	LP001015	Male	Yes	0	Graduate	No	5720	0	110.0	
1	LP001022	Male	Yes	1	Graduate	No	3076	1500	126.0	
2	LP001031	Male	Yes	2	Graduate	No	5000	1800	208.0	
3	LP001035	Male	Yes	2	Graduate	No	2340	2546	100.0	
4	LP001051	Male	No	0	Not Graduate	No	3276	0	78.0	

In [7]:
```
# we concat for easy analysis
n = train.shape[0] # we set this to be able to separate the
df = pd.concat([train, test], axis=0)
df.head()
```

Out[7]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	

Sweetviz

In [8]:
```
autoEDA = sweetviz.analyze(train)
autoEDA.show_notebook()
```

```
|          | [  0%]   00:00 -> (? left)
```

Your Own EDA

In [9]:
```
# Question 1 An overview of the data. (HINT: Provide the number of records, fields and
# their data types. Do for both)
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 2   Married            611 non-null    object 
 3   Dependents         599 non-null    object 
 4   Education          614 non-null    object 
 5   Self_Employed      582 non-null    object 
 6   ApplicantIncome    614 non-null    int64  
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

In [10]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 367 entries, 0 to 366
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            367 non-null    object 
 1   Gender             356 non-null    object 
 2   Married            367 non-null    object 
 3   Dependents         357 non-null    object 
 4   Education          367 non-null    object 
 5   Self_Employed      344 non-null    object 
 6   ApplicantIncome    367 non-null    int64  
 7   CoapplicantIncome  367 non-null    int64  
 8   LoanAmount         362 non-null    float64
 9   Loan_Amount_Term   361 non-null    float64
 10  Credit_History     338 non-null    float64
 11  Property_Area      367 non-null    object 
dtypes: float64(3), int64(2), object(7)
memory usage: 34.5+ KB
```

In [11]:
```
train.describe(include='all').T
```

Out[11]:

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Loan_ID	614	614	LP001002	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Gender	601	2	Male	489	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Married	611	2	Yes	398	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Dependents	599	4	0	345	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Education	614	2	Graduate	480	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Self_Employed	582	2	No	500	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ApplicantIncome	614.0	NaN	NaN	NaN	5403.459283	6109.041673	150.0	2877.5	3812.5	5795.0	81000.0
CoapplicantIncome	614.0	NaN	NaN	NaN	1621.245798	2926.248369	0.0	0.0	1188.5	2297.25	41667.0
LoanAmount	592.0	NaN	NaN	NaN	146.412162	85.587325	9.0	100.0	128.0	168.0	700.0
Loan_Amount_Term	600.0	NaN	NaN	NaN	342.0	65.12041	12.0	360.0	360.0	360.0	480.0
Credit_History	564.0	NaN	NaN	NaN	0.842199	0.364878	0.0	1.0	1.0	1.0	1.0
Property_Area	614	3	Semiurban	233	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Loan_Status	614	2	Y	422	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [12]:
```
test.describe(include='all').T
```

Out[12]:

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Loan_ID	367	367	LP001015	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Gender	356	2	Male	286	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Married	367	2	Yes	233	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Dependents	357	4	0	200	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Education	367	2	Graduate	283	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Self_Employed	344	2	No	307	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ApplicantIncome	367.0	NaN	NaN	NaN	4805.599455	4910.685399	0.0	2864.0	3786.0	5060.0	72529.0
CoapplicantIncome	367.0	NaN	NaN	NaN	1569.577657	2334.232099	0.0	0.0	1025.0	2430.5	24000.0
LoanAmount	362.0	NaN	NaN	NaN	136.132597	61.366652	28.0	100.25	125.0	158.0	550.0
Loan_Amount_Term	361.0	NaN	NaN	NaN	342.537396	65.156643	6.0	360.0	360.0	360.0	480.0
Credit_History	338.0	NaN	NaN	NaN	0.825444	0.38015	0.0	1.0	1.0	1.0	1.0
Property_Area	367	3	Urban	140	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [13]:
```
# Question 2 What data quality issues exist in both train and test? (HINT: Comment any missing
# values and duplicates)
train.duplicated().sum()
```

Out[13]:
```
0
```

In [14]:
```
test.duplicated().sum()
```

Out[14]:
```
0
```

In [15]:
```
train.isnull().sum()
```

Out[15]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

In [16]:
```
test.isnull().sum()
```

Out[16]:
```
Loan_ID               0
Gender               11
Married               0
Dependents           10
Education             0
Self_Employed        23
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            5
Loan_Amount_Term      6
Credit_History       29
Property_Area         0
dtype: int64
```

In [16]:
```
```

Credit_History should be object.

In [17]:
```
# Question 3 How do the the Loan statuses compare? i.e. what is the
# distrubution of each?
train['Loan_Status'].value_counts()
```

Out[17]:
```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

In [18]:
```
train['Loan_Status'].value_counts(normalize=True)
```

Out[18]:
```
Y    0.687296
N    0.312704
Name: Loan_Status, dtype: float64
```

In [19]:
```
# Question 4 How do women and men compare when it comes to defaulting
# on loans in the historical dataset?
train.groupby('Gender')['Loan_Status'].value_counts()
```

Out[19]:
```
Gender  Loan_Status
Female  Y               75
        N               37
Male    Y              339
        N              150
Name: Loan_Status, dtype: int64
```

In [20]:
```
train.groupby('Gender')['Loan_Status'].value_counts(normalize=True)
```

Out[20]:
```
Gender  Loan_Status
Female  Y              0.669643
        N              0.330357
Male    Y              0.693252
        N              0.306748
Name: Loan_Status, dtype: float64
```

In [21]:
```
# Question 5 How many of the loan applicants have dependents based on
# the historical dataset?
train[train['Dependents'] != '0'].shape[0]
```

Out[21]:
```
269
```

In [22]:
```
train[train['Dependents'] != '0'].shape[0]/train.shape[0]
```

Out[22]:
```
0.4381107491856677
```

In [23]:
```
# Question 6 How do the incomes of those who are employed compare to those
# who are self employed based on the historical dataset?
train.groupby('Self_Employed')['ApplicantIncome'].describe()
```

Out[23]:

	count	mean	std	min	25%	50%	75%	max
Self_Employed								
No	500.0	5049.748000	5682.895810	150.0	2824.50	3705.5	5292.75	81000.0
Yes	82.0	7380.817073	5883.564795	674.0	3452.25	5809.0	9348.50	39147.0

In [24]:
```
# Question 7 Are applicants with a credit history more likely to default than those
# who do not have one?
train.groupby('Credit_History')['Loan_Status'].value_counts(normalize=True)
```

Out[24]:
```
Credit_History  Loan_Status
0.0             N              0.921348
                Y              0.078652
1.0             Y              0.795789
                N              0.204211
Name: Loan_Status, dtype: float64
```

In [25]:
```
# Question 8 Is there a correlation between the applicant's income and the
# loan amount they applied for?
train.corr()
```

Out[25]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
ApplicantIncome	1.000000	-0.116605	0.570909	-0.045306	-0.014715
CoapplicantIncome	-0.116605	1.000000	0.188619	-0.059878	-0.002056
LoanAmount	0.570909	0.188619	1.000000	0.039447	-0.008433
Loan_Amount_Term	-0.045306	-0.059878	0.039447	1.000000	0.001470
Credit_History	-0.014715	-0.002056	-0.008433	0.001470	1.000000

Your anwers:

Bespoke EDA above shows how to arrive at the answers.

In [25]:
```

```

Part Two

Auto ML wth autosklearn

In [26]:
```
# Matrix of features

X = train[['Gender',
'Married',
'Dependents',
'Education',
'Self_Employed',
'ApplicantIncome',
'CoapplicantIncome',
'LoanAmount',
'Loan_Amount_Term',
'Credit_History',
'Property_Area']]

# convert string(text) to categorical
X['Gender'] = X['Gender'].astype('category')
X['Married'] = X['Married'].astype('category')
X['Education'] = X['Education'].astype('category')
X['Dependents'] = X['Dependents'].astype('category')
X['Self_Employed'] = X['Self_Employed'].astype('category')
X['Property_Area'] = X['Property_Area'].astype('category')


# label encode target
y = train['Loan_Status'].map({'N':0,'Y':1}).astype(int)


# # train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/ind
exing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guid
e/indexing.html#returning-a-view-versus-a-copy)
  app.launch_new_instance()
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/ind
exing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guid
e/indexing.html#returning-a-view-versus-a-copy)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/ind
exing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guid
e/indexing.html#returning-a-view-versus-a-copy)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:19: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/ind
exing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guid
e/indexing.html#returning-a-view-versus-a-copy)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:20: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/ind
exing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guid
e/indexing.html#returning-a-view-versus-a-copy)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/ind
exing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guid
e/indexing.html#returning-a-view-versus-a-copy)
```

In [27]:
```
# train
autoML = autosklearn.classification.AutoSklearnClassifier(time_left_for_this_task=2*30,
                                                          per_run_time_limit=30, n_jobs=8)
# imposing a 1 minute time limit on this
autoML.fit(X_train, y_train)

# predict
predictions_autoML = autoML.predict(X_test)
```

In [28]:
```
print('Model Accuracy:', accuracy_score(predictions_autoML, y_test))
```

```
Model Accuracy: 0.7886178861788617
```

In [29]:
```
print(confusion_matrix(predictions_autoML, y_test))
```

```
[[18  1]
 [25 79]]
```

Bespoke ML with sklearn

Data Preparation

Type *Markdown* and LaTeX: α^2

In [30]:
```
# Matrix of features

df = train[['Gender',
'Married',
'Education',
'Self_Employed',
'ApplicantIncome',
'CoapplicantIncome',
'LoanAmount',
'Loan_Amount_Term',
'Credit_History']]


# imputing the missing values:
df['Gender'].fillna(df['Gender'].mode()[0], inplace = True)
df['Married'].fillna(df['Married'].mode()[0], inplace = True)
df['Self_Employed'].fillna(df['Self_Employed'].mode()[0], inplace = True)
df['Credit_History'].fillna(df['Credit_History'].mode()[0], inplace = True)

# encoding categorical features
df['Gender'] = df['Gender'].map({'Male':0,'Female':1}).astype(int)
df['Married'] = df['Married'].map({'No':0,'Yes':1}).astype(int)
df['Education'] = df['Education'].map({'Not Graduate':0,'Graduate':1}).astype(int)
df['Self_Employed'] = df['Self_Employed'].map({'No':0,'Yes':1}).astype(int)
df['Credit_History'] = df['Credit_History'].astype(int)


df['LoanAmount'].fillna(df['LoanAmount'].mean(), inplace = True)
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mean(), inplace = True)

X = df.copy()

# label encode target
y = train['Loan_Status'].map({'N':0,'Y':1}).astype(int)


# train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/generic.py:6392: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/i
ndexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy)
  return self._update_inplace(result)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/i
ndexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/i
ndexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/i
ndexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:25: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
```

In [31]:

```python
# some classifiers you can pick from (remember to import)
import sklearn
classifiers = sklearn.utils.all_estimators(type_filter=None)
for name, class_ in classifiers:
    if hasattr(class_, 'predict_proba'):
        print(name)
```

```
AdaBoostClassifier
BaggingClassifier
BayesianGaussianMixture
BernoulliNB
CalibratedClassifierCV
CategoricalNB
ClassifierChain
ComplementNB
DecisionTreeClassifier
DummyClassifier
ExtraTreeClassifier
ExtraTreesClassifier
GaussianMixture
GaussianNB
GaussianProcessClassifier
GradientBoostingClassifier
GridSearchCV
HalvingGridSearchCV
HalvingRandomSearchCV
HistGradientBoostingClassifier
KNeighborsClassifier
LabelPropagation
LabelSpreading
LinearDiscriminantAnalysis
LogisticRegression
LogisticRegressionCV
MLPClassifier
MultiOutputClassifier
MultinomialNB
NuSVC
OneVsRestClassifier
Pipeline
QuadraticDiscriminantAnalysis
RFE
RFECV
RadiusNeighborsClassifier
RandomForestClassifier
RandomizedSearchCV
SGDClassifier
SVC
SelfTrainingClassifier
StackingClassifier
VotingClassifier
```

In [32]:

```python
# train
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier() #change model here
clf.fit(X_train, y_train)

# predict
predictions_clf = clf.predict(X_test)
```

In [33]:

```python
print('Model Accuracy:', accuracy_score(predictions_clf, y_test))
```

```
Model Accuracy: 0.7723577235772358
```

In [34]:
```
print(confusion_matrix(predictions_clf, y_test))
```

```
[[18  3]
 [25 77]]
```