In [ ]:

Name :- Md Shaukat Ali

Roll :- 23CS4141

Reg :- 23P10244

DAY-1 LAB

In [ ]:

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

In [2]:
```
import os
import warnings
```

In [3]:
```
train=pd.read_csv("Blood_samples_dataset_balanced_2(f).csv")
```

In [4]:
```
test= pd.read_csv("blood_samples_dataset_test.csv")
```

In [5]:
```
train
```

Out[5]:

	Glucose	Cholesterol	Hemoglobin	Platelets	White Blood Cells	Red Blood Cells	Hematocrit	Mean Corpuscular Volume
0	0.739597	0.650198	0.713631	0.868491	0.687433	0.529895	0.290006	0.631045
1	0.121786	0.023058	0.944893	0.905372	0.507711	0.403033	0.164216	0.307553
2	0.452539	0.116135	0.544560	0.400640	0.294538	0.382021	0.625267	0.295122
3	0.136609	0.015605	0.419957	0.191487	0.081168	0.166214	0.073293	0.668719
4	0.176737	0.752220	0.971779	0.785286	0.443880	0.439851	0.894991	0.442159
...	...	...	...	...	...	...	...	..
2346	0.012956	0.336925	0.451218	0.175006	0.734664	0.382770	0.656463	0.177502
2347	0.407101	0.124738	0.983306	0.663867	0.361113	0.663716	0.232516	0.341056
2348	0.344356	0.783918	0.582171	0.996841	0.065363	0.242885	0.658851	0.543017
2349	0.351722	0.014278	0.898615	0.167550	0.727148	0.046091	0.900434	0.136227
2350	0.032726	0.053596	0.102633	0.221356	0.153956	0.216573	0.312577	0.608940

2351 rows × 25 columns

In [6]:
```
test
```

Out[6]:

	Glucose	Cholesterol	Hemoglobin	Platelets	White Blood Cells	Red Blood Cells	Hematocrit	Mean Corpuscular Volume
0	0.001827	0.033693	0.114755	0.997927	0.562604	0.866499	0.578042	0.914615
1	0.436679	0.972653	0.084998	0.180909	0.675736	0.563889	0.798382	0.670361
2	0.545697	0.324815	0.584467	0.475748	0.558596	0.661007	0.934056	0.381782
3	0.172994	0.050351	0.736000	0.782022	0.069435	0.085219	0.032907	0.460619
4	0.758534	0.739968	0.597868	0.772683	0.875720	0.860265	0.486189	0.486686
...	...	...	...	...	...	...	...	...
481	0.985163	0.412960	0.529993	0.263765	0.431288	0.198882	0.581289	0.701192
482	0.581914	0.629325	0.491644	0.901473	0.347797	0.633286	0.698114	0.516947
483	0.066669	0.404558	0.591041	0.228401	0.127461	0.026670	0.847444	0.279740
484	0.901444	0.430680	0.243853	0.825551	0.493884	0.726299	0.660930	0.445560
485	0.877912	0.597809	0.730440	0.462307	0.498438	0.792822	0.976056	0.883937

486 rows × 25 columns

EDA on Training Data

In [7]:
```
train.head()
```

Out[7]:

	Glucose	Cholesterol	Hemoglobin	Platelets	White Blood Cells	Red Blood Cells	Hematocrit	Mean Corpuscular Volume	C H
0	0.739597	0.650198	0.713631	0.868491	0.687433	0.529895	0.290006	0.631045	
1	0.121786	0.023058	0.944893	0.905372	0.507711	0.403033	0.164216	0.307553	
2	0.452539	0.116135	0.544560	0.400640	0.294538	0.382021	0.625267	0.295122	
3	0.136609	0.015605	0.419957	0.191487	0.081168	0.166214	0.073293	0.668719	
4	0.176737	0.752220	0.971779	0.785286	0.443880	0.439851	0.894991	0.442159	

5 rows × 25 columns

In [8]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2351 entries, 0 to 2350
Data columns (total 25 columns):
 #   Column                                    Non-Null Count  Dtype  
---  ------                                    --------------  -----  
 0   Glucose                                   2351 non-null   float64
 1   Cholesterol                               2351 non-null   float64
 2   Hemoglobin                                2351 non-null   float64
 3   Platelets                                 2351 non-null   float64
 4   White Blood Cells                         2351 non-null   float64
 5   Red Blood Cells                           2351 non-null   float64
 6   Hematocrit                                2351 non-null   float64
 7   Mean Corpuscular Volume                   2351 non-null   float64
 8   Mean Corpuscular Hemoglobin               2351 non-null   float64
 9   Mean Corpuscular Hemoglobin Concentration 2351 non-null   float64
 10  Insulin                                   2351 non-null   float64
 11  BMI                                       2351 non-null   float64
 12  Systolic Blood Pressure                   2351 non-null   float64
 13  Diastolic Blood Pressure                  2351 non-null   float64
 14  Triglycerides                             2351 non-null   float64
 15  HbA1c                                     2351 non-null   float64
 16  LDL Cholesterol                           2351 non-null   float64
 17  HDL Cholesterol                           2351 non-null   float64
 18  ALT                                       2351 non-null   float64
 19  AST                                       2351 non-null   float64
 20  Heart Rate                                2351 non-null   float64
 21  Creatinine                                2351 non-null   float64
 22  Troponin                                  2351 non-null   float64
 23  C-reactive Protein                        2351 non-null   float64
 24  Disease                                   2351 non-null   object 
dtypes: float64(24), object(1)
memory usage: 459.3+ KB
```

In [9]:
```
train.isnull()
```

Out[9]:

	Glucose	Cholesterol	Hemoglobin	Platelets	White Blood Cells	Red Blood Cells	Hematocrit	Mean Corpuscular Volume	Cor Her
0	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	
2346	False	False	False	False	False	False	False	False	
2347	False	False	False	False	False	False	False	False	
2348	False	False	False	False	False	False	False	False	
2349	False	False	False	False	False	False	False	False	
2350	False	False	False	False	False	False	False	False	

2351 rows × 25 columns

In [10]:
```
train.isnull().sum()
```

Out[10]:
```
Glucose                                    0
Cholesterol                                0
Hemoglobin                                 0
Platelets                                  0
White Blood Cells                          0
Red Blood Cells                            0
Hematocrit                                 0
Mean Corpuscular Volume                    0
Mean Corpuscular Hemoglobin                0
Mean Corpuscular Hemoglobin Concentration  0
Insulin                                    0
BMI                                        0
Systolic Blood Pressure                    0
Diastolic Blood Pressure                   0
Triglycerides                              0
HbA1c                                      0
LDL Cholesterol                            0
HDL Cholesterol                            0
ALT                                        0
AST                                        0
Heart Rate                                 0
Creatinine                                 0
Troponin                                   0
C-reactive Protein                         0
Disease                                    0
dtype: int64
```

In [11]:
```
train.describe()
```

Out[11]:

	Glucose	Cholesterol	Hemoglobin	Platelets	White Blood Cells	Red Blood Cells	Hemato
count	2351.000000	2351.000000	2351.000000	2351.000000	2351.000000	2351.000000	2351.000(
mean	0.362828	0.393648	0.586190	0.504027	0.511086	0.506590	0.507
std	0.251889	0.239449	0.271498	0.303347	0.277270	0.266565	0.285
min	0.010994	0.012139	0.003021	0.012594	0.010139	0.044565	0.011
25%	0.129198	0.195818	0.346092	0.200865	0.259467	0.263589	0.288
50%	0.351722	0.397083	0.609836	0.533962	0.527381	0.467431	0.493
75%	0.582278	0.582178	0.791215	0.754841	0.743164	0.743670	0.753(
max	0.968460	0.905026	0.983306	0.999393	0.990786	1.000000	0.977

8 rows × 24 columns

In [12]:
```
train.duplicated()
```

Out[12]:
```
0       False
1       False
2       False
3       False
4       False
        ...
2346     True
2347     True
2348     True
2349     True
2350     True
Length: 2351, dtype: bool
```

In [13]:
```
train.duplicated().sum()
```

Out[13]:
```
2286
```

In [14]:
```
train.shape
```

Out[14]:
```
(2351, 25)
```

In [ ]:

In [ ]:

EDA on testing Data

In [15]:
```
test.head()
```

Out[15]:

	Glucose	Cholesterol	Hemoglobin	Platelets	White Blood Cells	Red Blood Cells	Hematocrit	Mean Corpuscular Volume	C H
0	0.001827	0.033693	0.114755	0.997927	0.562604	0.866499	0.578042	0.914615	
1	0.436679	0.972653	0.084998	0.180909	0.675736	0.563889	0.798382	0.670361	
2	0.545697	0.324815	0.584467	0.475748	0.558596	0.661007	0.934056	0.381782	
3	0.172994	0.050351	0.736000	0.782022	0.069435	0.085219	0.032907	0.460619	
4	0.758534	0.739968	0.597868	0.772683	0.875720	0.860265	0.486189	0.486686	

5 rows × 25 columns

In [16]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 486 entries, 0 to 485
Data columns (total 25 columns):
 #   Column                                    Non-Null Count  Dtype  
---  ------                                    --------------  -----  
 0   Glucose                                   486 non-null    float64
 1   Cholesterol                               486 non-null    float64
 2   Hemoglobin                                486 non-null    float64
 3   Platelets                                 486 non-null    float64
 4   White Blood Cells                         486 non-null    float64
 5   Red Blood Cells                           486 non-null    float64
 6   Hematocrit                                486 non-null    float64
 7   Mean Corpuscular Volume                   486 non-null    float64
 8   Mean Corpuscular Hemoglobin               486 non-null    float64
 9   Mean Corpuscular Hemoglobin Concentration  486 non-null    float64
 10  Insulin                                   486 non-null    float64
 11  BMI                                       486 non-null    float64
 12  Systolic Blood Pressure                   486 non-null    float64
 13  Diastolic Blood Pressure                  486 non-null    float64
 14  Triglycerides                             486 non-null    float64
 15  HbA1c                                     486 non-null    float64
 16  LDL Cholesterol                           486 non-null    float64
 17  HDL Cholesterol                           486 non-null    float64
 18  ALT                                       486 non-null    float64
 19  AST                                       486 non-null    float64
 20  Heart Rate                                486 non-null    float64
 21  Creatinine                                486 non-null    float64
 22  Troponin                                  486 non-null    float64
 23  C-reactive Protein                        486 non-null    float64
 24  Disease                                   486 non-null    object 
dtypes: float64(24), object(1)
memory usage: 95.0+ KB
```

In [17]:
```
test.describe()
```

Out[17]:

	Glucose	Cholesterol	Hemoglobin	Platelets	White Blood Cells	Red Blood Cells	Hematocrit	C
count	486.000000	486.000000	486.000000	486.000000	486.000000	486.000000	486.000000	
mean	0.490044	0.506797	0.485502	0.528136	0.509783	0.504347	0.501042	
std	0.284196	0.282871	0.298818	0.292610	0.290887	0.302865	0.294501	
min	0.001827	0.003088	0.000719	0.000006	-0.000206	0.000552	0.004556	
25%	0.236664	0.268021	0.201994	0.276155	0.264944	0.218573	0.246255	
50%	0.496471	0.502397	0.477706	0.538642	0.511102	0.518103	0.496275	
75%	0.727144	0.754638	0.750028	0.789486	0.767896	0.768466	0.761107	
max	0.991742	0.999606	0.997876	0.999507	0.999646	0.997267	1.000857	

8 rows × 24 columns

In [18]:
```
test.isnull().sum()
```

Out[18]:
```
Glucose                                    0
Cholesterol                                0
Hemoglobin                                 0
Platelets                                  0
White Blood Cells                          0
Red Blood Cells                            0
Hematocrit                                 0
Mean Corpuscular Volume                    0
Mean Corpuscular Hemoglobin                0
Mean Corpuscular Hemoglobin Concentration  0
Insulin                                    0
BMI                                        0
Systolic Blood Pressure                    0
Diastolic Blood Pressure                   0
Triglycerides                              0
HbA1c                                      0
LDL Cholesterol                            0
HDL Cholesterol                            0
ALT                                        0
AST                                        0
Heart Rate                                 0
Creatinine                                 0
Troponin                                   0
C-reactive Protein                         0
Disease                                    0
dtype: int64
```

In [19]:
```
test.duplicated().sum()
```

Out[19]: 0

In [20]:
```
test.describe()
```

Out[20]:

	Glucose	Cholesterol	Hemoglobin	Platelets	White Blood Cells	Red Blood Cells	Hematocrit	C
count	486.000000	486.000000	486.000000	486.000000	486.000000	486.000000	486.000000	
mean	0.490044	0.506797	0.485502	0.528136	0.509783	0.504347	0.501042	
std	0.284196	0.282871	0.298818	0.292610	0.290887	0.302865	0.294501	
min	0.001827	0.003088	0.000719	0.000006	-0.000206	0.000552	0.004556	
25%	0.236664	0.268021	0.201994	0.276155	0.264944	0.218573	0.246255	
50%	0.496471	0.502397	0.477706	0.538642	0.511102	0.518103	0.496275	
75%	0.727144	0.754638	0.750028	0.789486	0.767896	0.768466	0.761107	
max	0.991742	0.999606	0.997876	0.999507	0.999646	0.997267	1.000857	

8 rows × 24 columns

In [21]:
```
test.shape
```

Out[21]: (486, 25)

In [ ]:
```
```

We merge the train and test dataset in 'df'

In [22]:
```
df=pd.concat([train,test],axis=0)
df.head()
```

Out[22]:

	Glucose	Cholesterol	Hemoglobin	Platelets	White Blood Cells	Red Blood Cells	Hematocrit	Mean Corpuscular Volume	C H
0	0.739597	0.650198	0.713631	0.868491	0.687433	0.529895	0.290006	0.631045	
1	0.121786	0.023058	0.944893	0.905372	0.507711	0.403033	0.164216	0.307553	
2	0.452539	0.116135	0.544560	0.400640	0.294538	0.382021	0.625267	0.295122	
3	0.136609	0.015605	0.419957	0.191487	0.081168	0.166214	0.073293	0.668719	
4	0.176737	0.752220	0.971779	0.785286	0.443880	0.439851	0.894991	0.442159	

5 rows × 25 columns

In [23]:
```
df.isnull().sum()
```

Out[23]:
```
Glucose                                  0
Cholesterol                              0
Hemoglobin                               0
Platelets                                0
White Blood Cells                        0
Red Blood Cells                          0
Hematocrit                               0
Mean Corpuscular Volume                  0
Mean Corpuscular Hemoglobin              0
Mean Corpuscular Hemoglobin Concentration    0
Insulin                                  0
BMI                                      0
Systolic Blood Pressure                  0
Diastolic Blood Pressure                 0
Triglycerides                            0
HbA1c                                    0
LDL Cholesterol                          0
HDL Cholesterol                          0
ALT                                      0
AST                                      0
Heart Rate                               0
Creatinine                               0
Troponin                                 0
C-reactive Protein                       0
Disease                                  0
dtype: int64
```

In [24]:
```
df.shape
```

Out[24]:
```
(2837, 25)
```

In [25]:
```
df['Disease'].unique()
```

Out[25]:
```
array(['Healthy', 'Diabetes', 'Thalasse', 'Anemia', 'Thromboc',
       'Heart Di'], dtype=object)
```

In [26]:
```
df['Disease'].nunique()
```

Out[26]:
```
6
```

In [27]:
```
df['Disease']=df['Disease'].replace('Heart Di','Heart Disease')
df['Disease'].unique()
```

Out[27]:
```
array(['Healthy', 'Diabetes', 'Thalasse', 'Anemia', 'Thromboc',
       'Heart Disease'], dtype=object)
```

In [28]:
```
df.columns
```

Out[28]:
```
Index(['Glucose', 'Cholesterol', 'Hemoglobin', 'Platelets',
       'White Blood Cells', 'Red Blood Cells', 'Hematocrit',
       'Mean Corpuscular Volume', 'Mean Corpuscular Hemoglobin',
       'Mean Corpuscular Hemoglobin Concentration', 'Insulin', 'BMI',
       'Systolic Blood Pressure', 'Diastolic Blood Pressure', 'Triglyceride
s',
       'HbA1c', 'LDL Cholesterol', 'HDL Cholesterol', 'ALT', 'AST',
       'Heart Rate', 'Creatinine', 'Troponin', 'C-reactive Protein',
       'Disease'],
      dtype='object')
```

In [ ]:

In [ ]:

We have to classify which type of disease , The Disease feature of categorical type so we need to convert it into numerical so here we apply label encoder of different different disease

In [29]:
```
from sklearn.preprocessing import LabelEncoder
```

In [30]:
```
le=LabelEncoder()
```

In [31]:
```
df['Disease']=le.fit_transform(df['Disease'])
df.head()
```

Out[31]:

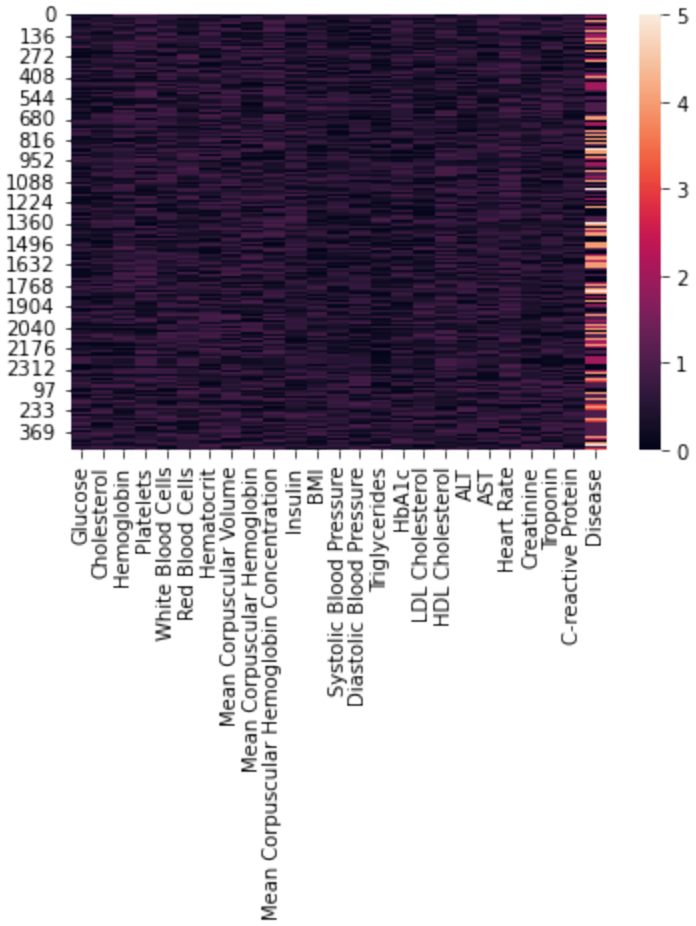
	Glucose	Cholesterol	Hemoglobin	Platelets	White Blood Cells	Red Blood Cells	Hematocrit	Mean Corpuscular Volume	C H
0	0.739597	0.650198	0.713631	0.868491	0.687433	0.529895	0.290006	0.631045	
1	0.121786	0.023058	0.944893	0.905372	0.507711	0.403033	0.164216	0.307553	
2	0.452539	0.116135	0.544560	0.400640	0.294538	0.382021	0.625267	0.295122	
3	0.136609	0.015605	0.419957	0.191487	0.081168	0.166214	0.073293	0.668719	
4	0.176737	0.752220	0.971779	0.785286	0.443880	0.439851	0.894991	0.442159	

5 rows × 25 columns

Split the Dataset into training and testing part:-

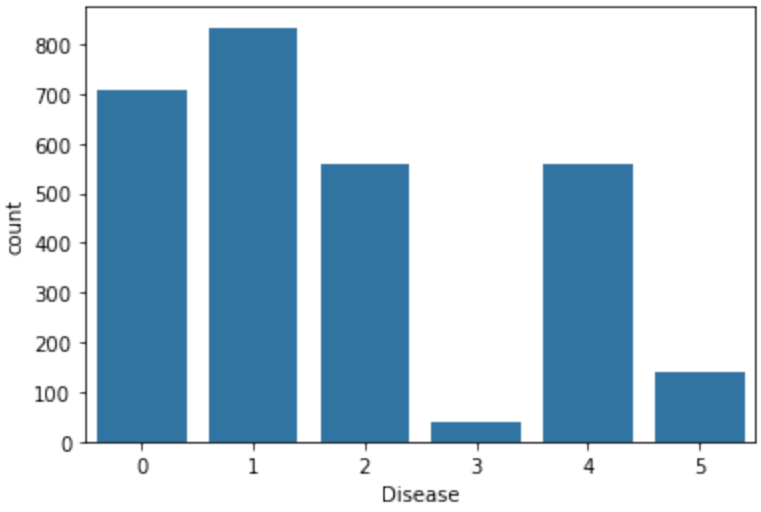
In [32]:
```
sns.heatmap(df)
```

Out[32]: <AxesSubplot:>



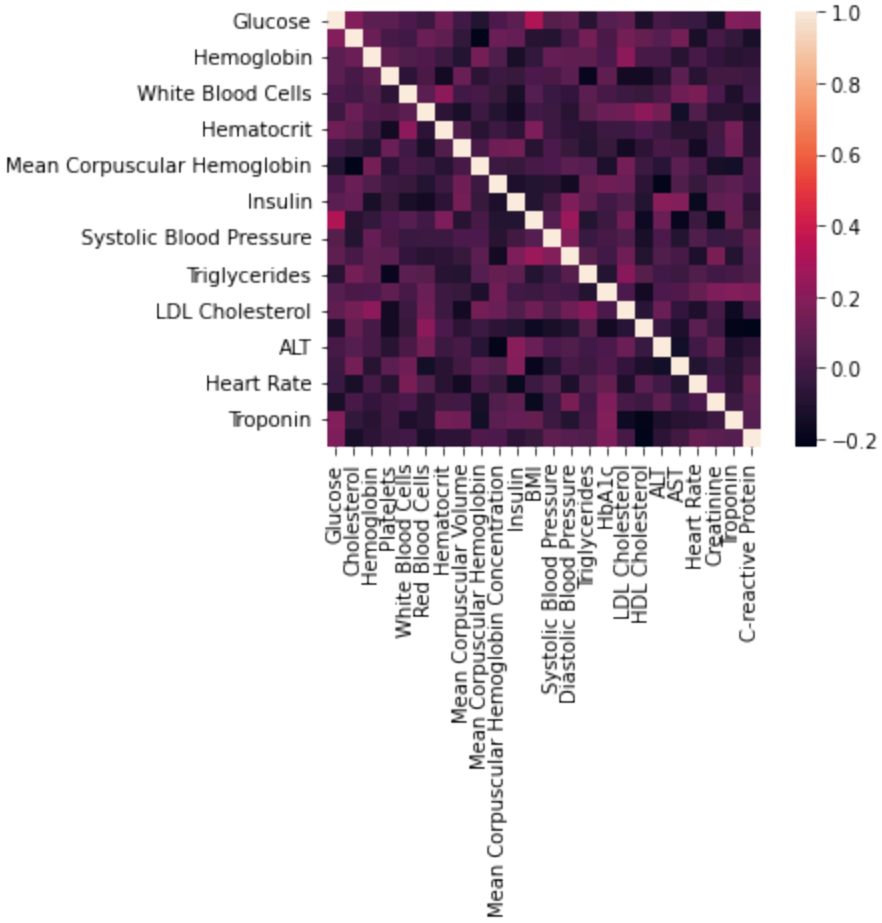
In [34]:
```
y=df['Disease']
x=df.drop('Disease',axis=1)
```

In [35]:
```
sns.countplot(x=y)
plt.show()
```



In [ ]:

In [42]:
```
sns.heatmap(x.corr(),square=True)
plt.show()
```



In [36]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state
```

In [37]:
```
X_train
```

Out[37]:

	Glucose	Cholesterol	Hemoglobin	Platelets	White Blood Cells	Red Blood Cells	Hematocrit	Mean Corpuscular Volume
438	0.798768	0.400015	0.478800	0.172237	0.686304	0.467431	0.444994	0.642926
1005	0.298550	0.182145	0.701824	0.418574	0.875773	0.660344	0.977520	0.459985
1393	0.253417	0.257914	0.003021	0.284489	0.199566	0.703588	0.957599	0.074917
1784	0.143772	0.089600	0.027259	0.171121	0.744950	0.452787	0.871778	0.995263
166	0.452539	0.116135	0.544560	0.400640	0.294538	0.382021	0.625267	0.295122
...	...	...	...	...	...	...	...	..
1638	0.799282	0.447299	0.325035	0.825397	0.259467	0.743670	0.434097	0.198072
1095	0.099982	0.542793	0.795435	0.196821	0.371385	0.156494	0.457720	0.046942
1130	0.143772	0.089600	0.027259	0.171121	0.744950	0.452787	0.871778	0.995263
1294	0.798768	0.400015	0.478800	0.172237	0.686304	0.467431	0.444994	0.642926
860	0.722428	0.300282	0.487907	0.200865	0.402035	0.729285	0.851338	0.820546

2127 rows × 24 columns

In [38]:
```
y_train = np.array(y_train).reshape(-1,1)
```

In [39]:
```
y_train.shape
```

Out[39]:
```
(2127, 1)
```

In [40]:
```
X_train.shape
```

Out[40]:
```
(2127, 24)
```

In [41]:
```
from sklearn.metrics import accuracy_score,roc_auc_score,roc_curve
```

Imported all the ML algorithm and make object of each model:-

In [43]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
```

In [44]:
```
lr=LogisticRegression(multi_class='multinomial')
dt=DecisionTreeClassifier()
rf=RandomForestClassifier()
knn=KNeighborsClassifier()
gnb=GaussianNB()
```

In [45]:
```
def predictor(model_name):
    print("For the {}".format(model_name))
    print("")
    model_name.fit(X_train,y_train)
    y_pred_train = model_name.predict(X_train)
    y_pred_test = model_name.predict(X_test)
    print("The TRAIN accuracy is",accuracy_score(y_train,y_pred_train))
    print("--"*50)
    print("The TEST accuracy is",accuracy_score(y_test,y_pred_test))
```

Logistic Regression

In [46]:
```
predictor(lr)
```

```
For the LogisticRegression(multi_class='multinomial')


C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):
C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: D
ataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel
().
  return f(*args, **kwargs)

The TRAIN accuracy is 0.8241654913023038
--------------------------------------------------------------------------
------------------------
The TEST accuracy is 0.7971830985915493

C:\Users\91825\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:763: 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 (https://scik
it-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-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):
C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):
```

Decision Tree

In [47]:
```
predictor(dt)
```

```
For the DecisionTreeClassifier()

The TRAIN accuracy is 1.0
--------------------------------------------------------------------------------
------------------------
The TEST accuracy is 0.9436619718309859

C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):
C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):
C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):
```

K- NN

In [48]:
```
predictor(knn)
```

```
For the KNeighborsClassifier()


C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):
C:\Users\91825\anaconda3\lib\site-packages\sklearn\neighbors\_classificatio
n.py:179: DataConversionWarning: A column-vector y was passed when a 1d arra
y was expected. Please change the shape of y to (n_samples,), for example us
ing ravel().
  return self._fit(X, y)
C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):

The TRAIN accuracy is 0.922425952045134
--------------------------------------------------------------------------------
------------------------
The TEST accuracy is 0.8943661971830986

C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):
```

Random Forest

In [49]:
```
predictor(rf)
```

```
C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):
C:\Users\91825\AppData\Local\Temp/ipykernel_22252/4075669676.py:4: DataConve
rsionWarning: 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().
  model_name.fit(X_train,y_train)

For the RandomForestClassifier()

The TRAIN accuracy is 1.0
--------------------------------------------------------------------------
------------------------
The TEST accuracy is 0.956338028169014

C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):
C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):
```

Naive Bayes

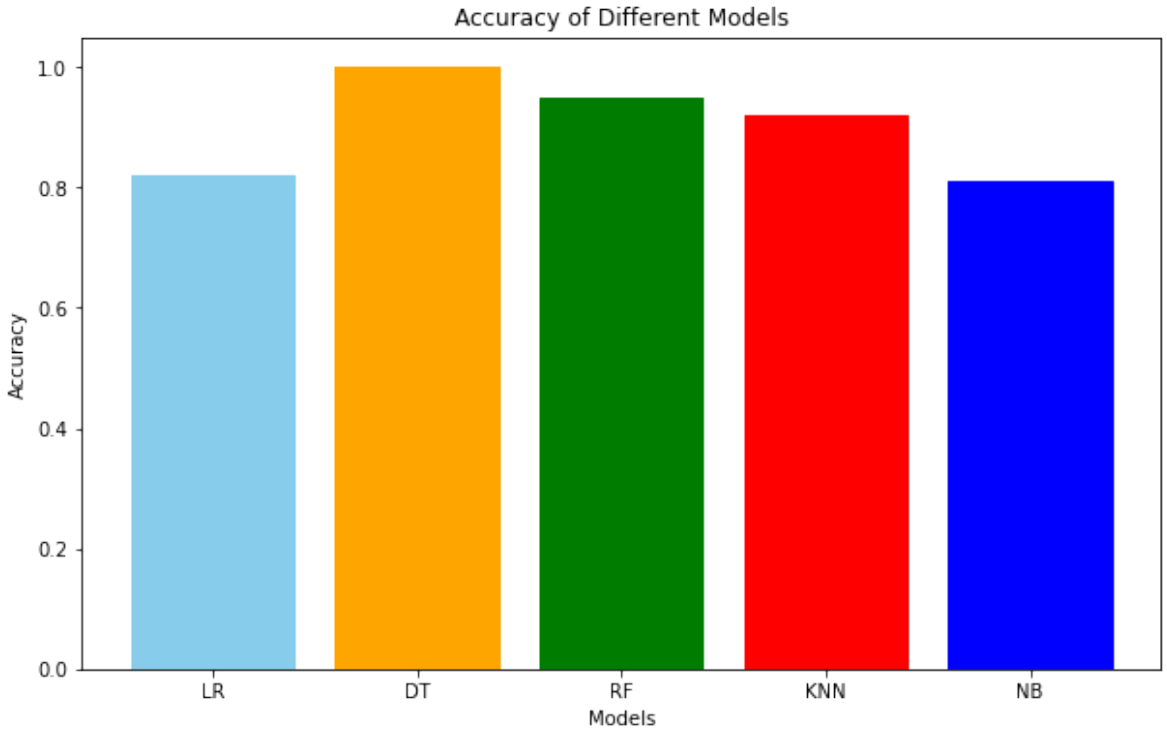
In [50]:
```
predictor(gnb)
```

```
For the GaussianNB()

The TRAIN accuracy is 0.8119417019275975
--------------------------------------------------------------------------------
------------------------
The TEST accuracy is 0.8028169014084507

C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):
C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: D
ataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel
().
  return f(*args, **kwargs)
C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):
C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):
```

Accuracy of different - different Model

In [51]:
```
models = ['LR', 'DT', 'RF', 'KNN' , 'NB']
accuracies = [0.82, 1, 0.95, 0.92 , 0.81]
colors = ['skyblue', 'orange', 'green', 'red' , 'blue']
```

In [52]:
```
plt.figure(figsize=(10, 6))
plt.bar(models, accuracies, color='skyblue')
bars = plt.bar(models, accuracies)
for i in range(len(bars)):
    bars[i].set_color(colors[i])
plt.title('Accuracy of Different Models')
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.show()
```



We apply MLP on this dataset

In [53]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
```

In [54]:
```
import tensorflow as tf
from tensorflow.keras import layers, models
```

In [55]:
```
def create_mlp(input_shape, num_classes):
    model = models.Sequential([
        layers.Dense(128, activation='relu', input_shape=input_shape),
        layers.Dense(64, activation='relu'),
        layers.Dense(num_classes, activation='softmax')
    ])
    return model
```

In [56]:
```
# Create MLP model
input_shape = X_train.shape[1:]
num_classes = len(np.unique(y_train))
mlp_model = create_mlp(input_shape, num_classes)
```

In [57]:
```
# Compile MLP model
mlp_model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
```

In [58]:
```
mlp_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               3200

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 6)                 390

=================================================================
Total params: 11,846
Trainable params: 11,846
Non-trainable params: 0
_________________________________________________________________
```

In [59]:
```
mlp_history = mlp_model.fit(X_train, y_train, epochs=5, validation_data=(X_te
```

In [60]:
```
mlp_loss, mlp_accuracy = mlp_model.evaluate(X_test, y_test)
print("MLP Accuracy:", mlp_accuracy)
```

```
23/23 [==============================] - 0s 4ms/step - loss: 0.4391 - accura
cy: 0.9127
MLP Accuracy: 0.9126760363578796
```

In [61]:
```
from sklearn.svm import SVC
```

In [62]:
```
# Train SVM model
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)
```

```
C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):
C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: D
ataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel
().
  return f(*args, **kwargs)
```

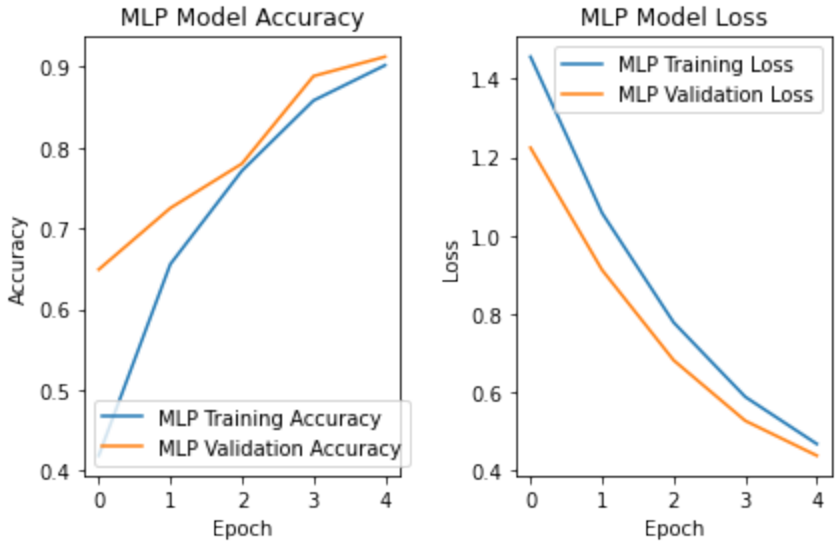
Out[62]:
```
SVC(kernel='linear')
```

In [63]:
```
svm_accuracy = accuracy_score(y_test, svm_model.predict(X_test))
print("SVM Accuracy:", svm_accuracy)
```

```
C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):

SVM Accuracy: 0.8619718309859155
```

Graph shows the accuracy and loss of MLP vs SVM

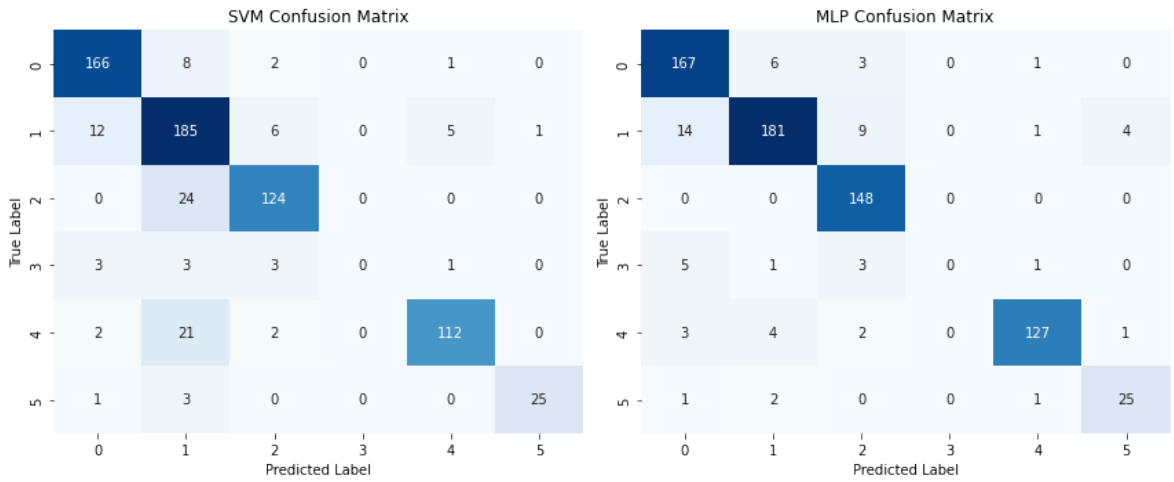
In [64]:
```
# Plot MLP accuracy
plt.subplot(1, 2, 1)
plt.plot(mlp_history.history['accuracy'], label='MLP Training Accuracy')
plt.plot(mlp_history.history['val_accuracy'], label='MLP Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('MLP Model Accuracy')
plt.legend()

# Plot MLP loss
plt.subplot(1, 2, 2)
plt.plot(mlp_history.history['loss'], label='MLP Training Loss')
plt.plot(mlp_history.history['val_loss'], label='MLP Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('MLP Model Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



In [65]:
```
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [66]:
```
svm_pred = svm_model.predict(X_test)
svm_cm = confusion_matrix(y_test, svm_pred)

# MLP confusion matrix
mlp_pred = np.argmax(mlp_model.predict(X_test), axis=1)
mlp_cm = confusion_matrix(y_test, mlp_pred)

# Plot confusion matrices
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.heatmap(svm_cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('SVM Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')

plt.subplot(1, 2, 2)
sns.heatmap(mlp_cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('MLP Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')

plt.tight_layout()
plt.show()
```

```
C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):

23/23 [==============================] - 0s 3ms/step
```



DAY-2 LAB

In [ ]:
```

```

In [ ]:

process the dataset with Support Vector Machines (SVM). In the next step, consolidate the outcomes of MLP and SVM at decision level with logical OR and AND operators to obtain the final outcome. Finally, show both ERROR and ACCURACY curves for MLP, SVM, OR rule and AND rule in the same graph while considering loss and accuracy in the same axis.

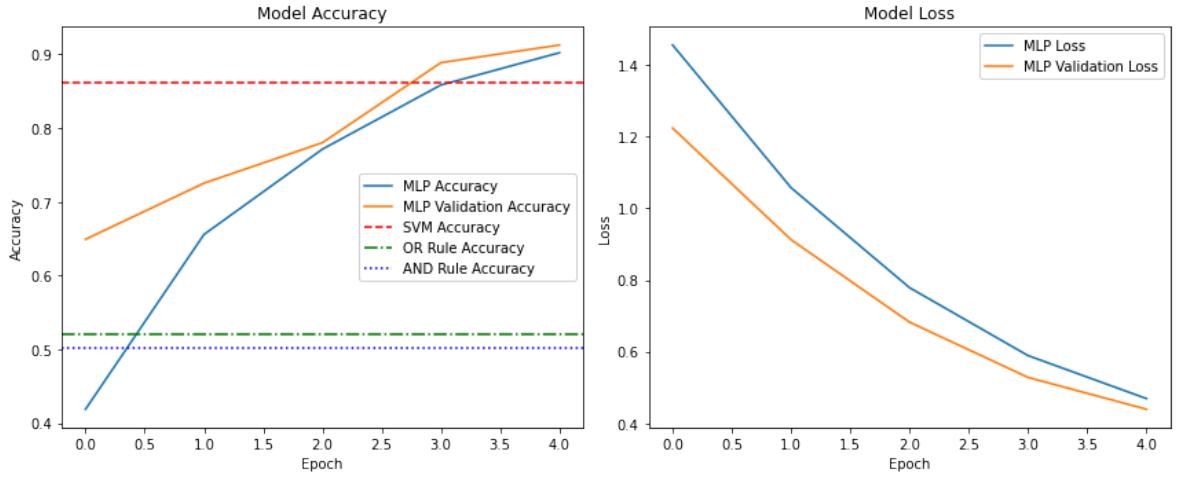
In [68]:
```
# Step 1: Obtain predictions from both models
svm_predictions = svm_model.predict(X_test)
mlp_predictions = np.argmax(mlp_model.predict(X_test), axis=-1)
```

```
 1/23 [>.............................] - ETA: 0s

C:\Users\91825\anaconda3\lib\site-packages\sklearn\utils\validation.py:571:
FutureWarning: is_sparse is deprecated and will be removed in a future versi
on. Check `isinstance(dtype, pd.SparseDtype)` instead.
  array.dtypes.apply(is_sparse).any()):

23/23 [==============================] - 0s 2ms/step
```

In [69]:
```
# Step 2: Apply logical OR and AND operators to consolidate predictions
or_predictions = np.logical_or(svm_predictions, mlp_predictions)
and_predictions = np.logical_and(svm_predictions, mlp_predictions)
```

In [70]:
```
# Step 3: Calculate accuracy and loss for consolidated predictions
svm_accuracy = accuracy_score(y_test, svm_predictions)
mlp_accuracy = accuracy_score(y_test, mlp_predictions)
or_accuracy = accuracy_score(y_test, or_predictions)
and_accuracy = accuracy_score(y_test, and_predictions)
```

In [71]:
```
# Step 4: Plot error and accuracy curves
plt.figure(figsize=(12, 5))

# Accuracy curve
plt.subplot(1, 2, 1)
plt.plot(mlp_history.history['accuracy'], label='MLP Accuracy')
plt.plot(mlp_history.history['val_accuracy'], label='MLP Validation Accuracy'
plt.axhline(y=svm_accuracy, color='r', linestyle='--', label='SVM Accuracy')
plt.axhline(y=or_accuracy, color='g', linestyle='-.', label='OR Rule Accuracy
plt.axhline(y=and_accuracy, color='b', linestyle=':', label='AND Rule Accuracy
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')
plt.legend()

# Loss curve
plt.subplot(1, 2, 2)
plt.plot(mlp_history.history['loss'], label='MLP Loss')
plt.plot(mlp_history.history['val_loss'], label='MLP Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Model Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



In [ ]: