AIML_PROJECT

August 21, 2023

[ ]:
```
%%HTML
<script src="require.js"></script>
```

```
<IPython.core.display.HTML object>
```

1 AI&ML PROJECT - Summer 2023

Students :

-Ahmad Khalil Fratekh - 0206882
-Noor Mohammad Albaw - 0201672
-Ibrahim Bilal Saleem - 0209537

1.1 Project Overview :

The project involves working on a dataset of medical students, working on the dataset we went through multiple operations, which include :

1-Data preparation
2-Classification using sklearn
3-Classification using neural networks from keras

About Dataset

1.1.1 Medical Student Dataset:

The Medical Student Dataset is a simulated dataset containing 200,000 rows and 13 columns.

1.1.2 Columns Description:

1- StudentID: Unique identifier for each medical student
2- Gender: Gender of the student (i.e., Male, Female).
3- Age: The age of each student in years.
4- Height: Height for students in m
5- Weight: Weight for students in KG
6- Blood Type : students blood type having values (A , B , AB , O)
7- BMI : The body mass index for each student represented with the formula : BMI = Weight(Kg)/
8- Smoking : having values of Yes or No
9- Diabetes: having values of Yes or No

10-Temperature: Body temperature of each student
11- Heart Rate : Heart Rate of the students (float values)
12- Blood Pressure: Blood Pressure of the students (float values)
13- Cholesterol: The amount of cholesterol in each student's body (float values)

1.1.3 The dataset contains :

-Missing values: Some columns may have missing values represented as NaN.
-Duplicates: Duplicate records may exist in the dataset, representing some/all identical studen
-Inconsistencies: The dataset may contain inconsistent or erroneous values in certain columns.

and we have set our target to be the "Diabetes" column

[ ]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.impute import SimpleImputer
from scipy import stats
import os
import seaborn as sns
from sklearn.preprocessing import *
from sklearn.model_selection import *
from sklearn.model_selection import cross_val_score
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
import joblib
from sklearn.metrics import *
from sklearn.utils import shuffle
from sklearn import preprocessing
from pandas.plotting import scatter_matrix
import tensorflow as tf
import keras
from keras import layers
import shutil
from tensorflow.keras import initializers
import plotly.graph_objs as go
import plotly.express as px
import plotly.io as pio
pio.renderers.default='notebook+pdf'
```

1.2 *PART 1*

Data Preparation :

[ ]:
```
data = pd.read_csv('medical_students_dataset.csv')
data.shape
```

[ ]:
```
(200000, 13)
```

[ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 13 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   Student ID      180000 non-null  float64
 1   Age             180000 non-null  float64
 2   Gender          180000 non-null  object
 3   Height          180000 non-null  float64
 4   Weight          180000 non-null  float64
 5   Blood Type      180000 non-null  object
 6   BMI             180000 non-null  float64
 7   Temperature     180000 non-null  float64
 8   Heart Rate      180000 non-null  float64
 9   Blood Pressure  180000 non-null  float64
 10  Cholesterol     180000 non-null  float64
 11  Diabetes        180000 non-null  object
 12  Smoking         180000 non-null  object
dtypes: float64(9), object(4)
memory usage: 19.8+ MB
```

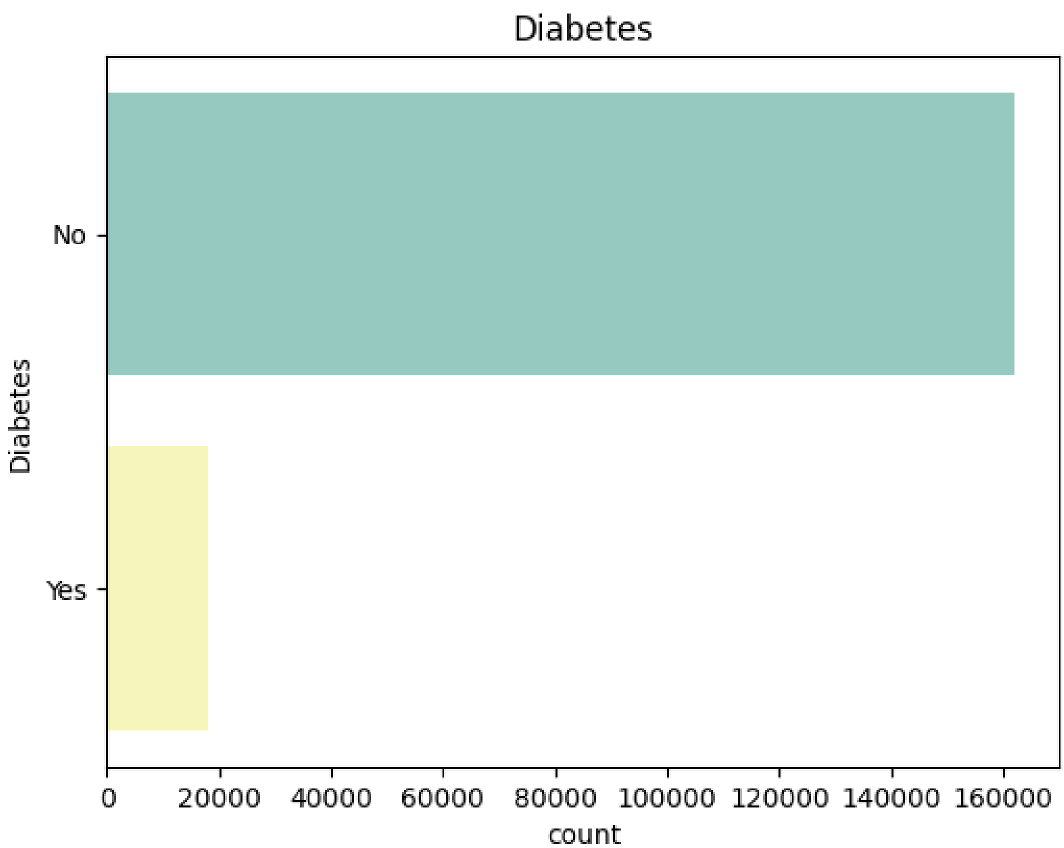
[ ]:
```
data.isnull().sum()
```

[ ]:
```
Student ID        20000
Age               20000
Gender            20000
Height            20000
Weight            20000
Blood Type        20000
BMI               20000
Temperature       20000
Heart Rate        20000
Blood Pressure    20000
Cholesterol       20000
Diabetes          20000
Smoking           20000
dtype: int64
```

1.2.1 *Since, as we can see from the plot below, the data is not balanced, we started off by extracting a balanced data to work on*

[ ]:
```
plt.gca().set_title("Diabetes")
sns.countplot(y="Diabetes" , palette='Set3', data = data)
```

[ ]:
```
<Axes: title={'center': 'Diabetes'}, xlabel='count', ylabel='Diabetes'>
```



1.2.2 *The following three cells of code, implement and visualize the procedure of extracting our balanced dataset :*

[ ]:
```
data_yes = data[data["Diabetes"]=="Yes"]
data_no = data[data["Diabetes"]=="No"]

pd.DataFrame(data_yes)
pd.DataFrame(data_no)

Med_Stu=pd.concat([data_yes[:18000],data_no[:18000]], ignore_index=True)
Med_Stu["Diabetes"]
```

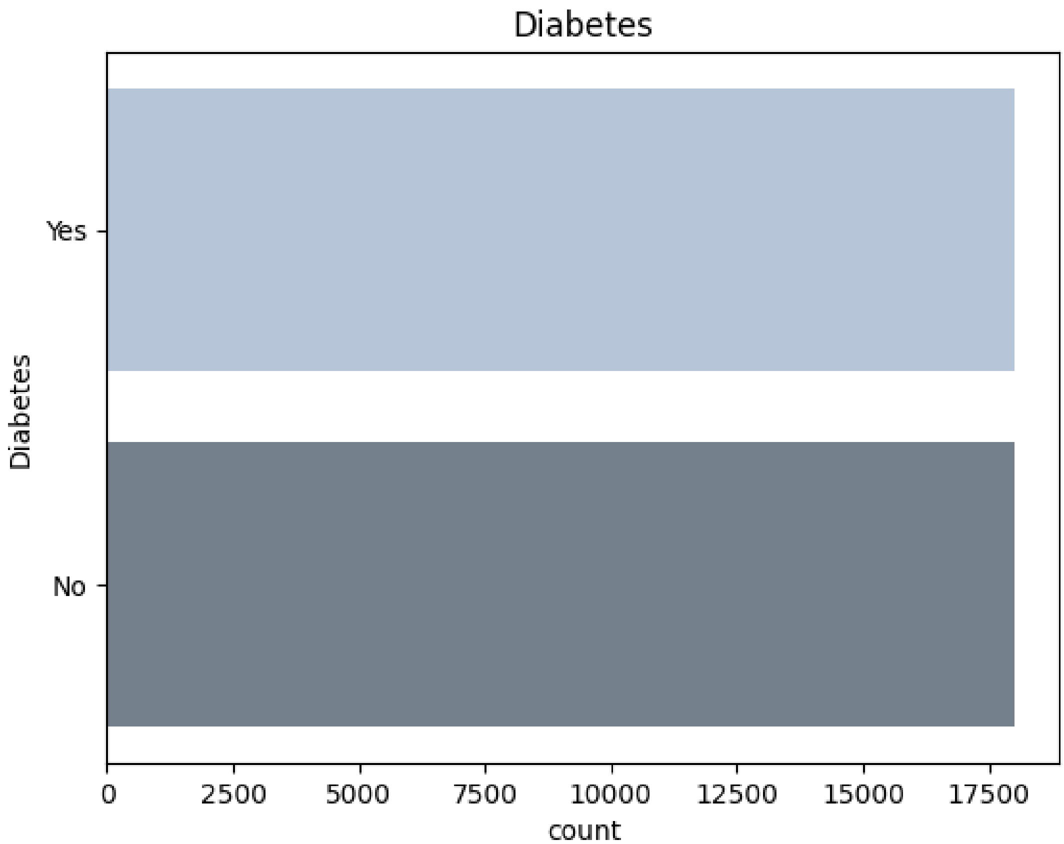
[ ]:
```
0        Yes
1        Yes
2        Yes
3        Yes
4        Yes
        …
35995     No
35996     No
35997     No
35998     No
35999     No
Name: Diabetes, Length: 36000, dtype: object
```

[ ]:
```python
print(Med_Stu.shape)

Med_Stu.info()
```

```
(36000, 13)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36000 entries, 0 to 35999
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Student ID      32406 non-null  float64
 1   Age             32444 non-null  float64
 2   Gender          32354 non-null  object
 3   Height          32393 non-null  float64
 4   Weight          32346 non-null  float64
 5   Blood Type      32431 non-null  object
 6   BMI             32382 non-null  float64
 7   Temperature     32414 non-null  float64
 8   Heart Rate      32447 non-null  float64
 9   Blood Pressure  32419 non-null  float64
 10  Cholesterol     32404 non-null  float64
 11  Diabetes        36000 non-null  object
 12  Smoking         32311 non-null  object
dtypes: float64(9), object(4)
memory usage: 3.6+ MB
```

[ ]:
```python
plt.gca().set_title("Diabetes")
sns.countplot(y="Diabetes" , palette=['lightsteelblue','slategrey'], data =␣
↪Med_Stu)
```

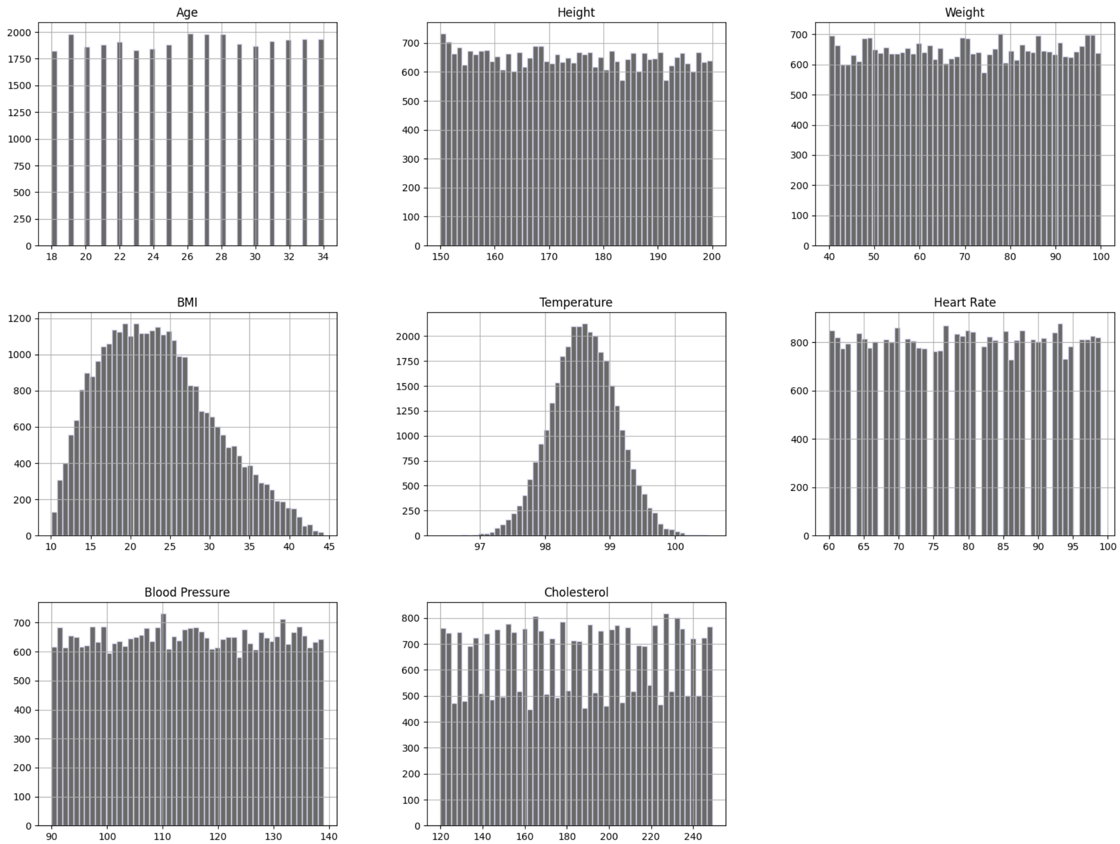
[ ]:
```
<Axes: title={'center': 'Diabetes'}, xlabel='count', ylabel='Diabetes'>
```



1.2.3 Now that we have extracted our balanced data, it's time to get rid of unnecessary or full of null values column(s), which (is/are) : "Student ID"

[ ]:
```
Med_Stu_0= Med_Stu.drop("Student ID", axis=1,inplace=True)
```

[ ]:
```
Med_Stu.hist(bins=50, figsize=(20,15), color='dimgray', ec='lavender')
plt.show()

Med_Stu.describe()
```



[ ]:

```
                Age        Height        Weight           BMI   Temperature  \
count  32444.000000  32393.000000  32346.000000  32382.000000  32414.000000
mean      26.043983    174.730780     70.088455     23.392699     98.596852
std        4.894372     14.493838     17.377104      7.093494      0.502917
min       18.000000    150.000996     40.000578     10.081431     96.397835
25%       22.000000    162.149046     55.009155     17.865348     98.260639
50%       26.000000    174.673148     70.082653     22.733757     98.595614
75%       30.000000    187.302386     85.194960     28.084662     98.939748
max       34.000000    199.997940     99.997668     44.314074    100.566498

         Heart Rate  Blood Pressure   Cholesterol
count  32447.000000    32419.000000  32404.000000
mean      79.516627      114.530337    184.759844
std       11.546663       14.398554     37.517835
min       60.000000       90.000000    120.000000
25%       70.000000      102.000000    152.000000
50%       80.000000      114.000000    185.000000
75%       90.000000      127.000000    218.000000
max       99.000000      139.000000    249.000000
```

In the cell below, we can notice that, there's a couple of missing values in all of the

features except for the label which raises the need of using an imputer

```
[ ]: Med_Stu.isnull().sum()
```

```
[ ]: Age                3556
     Gender             3646
     Height             3607
     Weight             3654
     Blood Type         3569
     BMI                3618
     Temperature        3586
     Heart Rate         3553
     Blood Pressure     3581
     Cholesterol        3596
     Diabetes              0
     Smoking            3689
     dtype: int64
```

1.2.4 *Before getting to the pipeline, we have to seperate the label from other features, afterwards we have to seperate numercial features from the categorical ones, which is implied in the folloing two cells of code*

```
[ ]: Med_Stu_1= Med_Stu.drop("Diabetes", axis=1)
     Med_Stu_label = Med_Stu["Diabetes"].copy()
     Med_Stu_1.head(10)
```

```
[ ]:     Age  Gender      Height     Weight Blood Type        BMI  Temperature  \
     0  32.0  Female  182.537664  55.741083          A  16.729017    98.260293
     1  34.0    Male         NaN  60.882228          B  22.544095    98.963569
     2  33.0    Male  184.718988  93.666944        NaN  27.451322    98.418213
     3  33.0    Male  177.165911  68.129149          O  21.705642    98.201649
     4  33.0    Male  160.463706  55.755226          A  21.653691    99.161461
     5  34.0    Male  174.207898        NaN          B  22.292974    98.919826
     6  24.0  Female  162.044348  80.638530         AB  30.709647    98.996462
     7  21.0    Male  171.146689  69.056734         AB  23.575941    98.316070
     8  34.0  Female  150.942632  90.580214          O  39.756624    97.563234
     9  21.0    Male  159.633475  89.877838         AB  35.269937    98.592047

        Heart Rate  Blood Pressure  Cholesterol Smoking
     0        76.0           130.0        216.0      No
     1        89.0           130.0        243.0     NaN
     2        68.0           133.0        180.0     Yes
     3         NaN           116.0        143.0     Yes
     4        77.0             NaN        152.0     Yes
     5        74.0           129.0        139.0     Yes
     6        98.0           137.0        205.0     Yes
     7         NaN            93.0        134.0      No
     8        79.0           135.0        198.0      No
```

```
9        71.0            115.0         137.0      No
```

[ ]:
```
Med_Stu_Cat = Med_Stu_1.select_dtypes(include="object").columns
Med_Stu_Num = Med_Stu_1.select_dtypes(exclude="object").columns
```

1.2.5 The pipeline:

After separating the features from the response: -The cell below shows 2 pipelines that are created, after dividing the dataset into two categories, the first one contains the numeric features, where the second one contains the categorical features.

```
-The first pipeline, (i.e. "NUM_PIPELINE"), is used to, first, fill the missing values of the n

-The second one , (i.e. "CAT_PIPELINE"), is used, first, fill the missing values of the catego

-The last pipeline (i.e. "full_pipeline"), basically, combines the first two together.
```

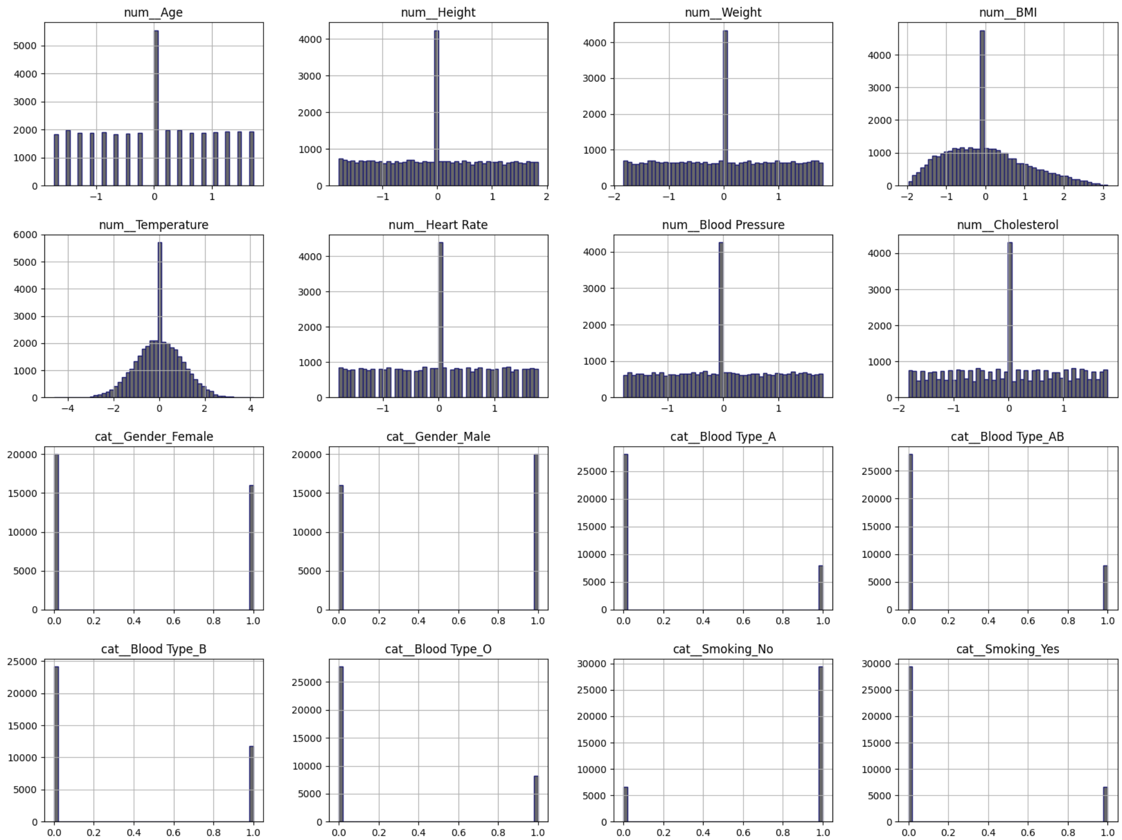
[ ]:
```
NUM_PIPLINE= Pipeline([('num_imp', SimpleImputer(strategy ="median")),␣
 ↪('std_scaler', StandardScaler())])

CAT_PIPLINE= Pipeline([('cat_imp',SimpleImputer(strategy='most_frequent')),␣
 ↪('1_hot_encoder',OneHotEncoder(sparse=False))])



full_pipeline = ColumnTransformer([
        ("num",NUM_PIPLINE, Med_Stu_Num),
        ("cat", CAT_PIPLINE,Med_Stu_Cat),
    ],remainder='passthrough')

med_stu = full_pipeline.fit_transform(Med_Stu_1)

Med_Stu_Prepared=pd.DataFrame(med_stu,columns=full_pipeline.
 ↪get_feature_names_out())
print(Med_Stu_Prepared.shape)
Med_Stu_Prepared.head()
```

```
(36000, 16)

c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\preprocessing\_encoders.py:972: FutureWarning:

`sparse` was renamed to `sparse_output` in version 1.2 and will be removed in
1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
```

[ ]:
```
   num__Age  num__Height  num__Weight  num__BMI  num__Temperature  \
0  1.282817     0.568260    -0.871013 -0.980244         -0.705013
1  1.713266    -0.003772    -0.558887 -0.116245          0.768728
```

```
2  1.498041     0.726921     1.431519  0.612866         -0.374085
3  1.498041     0.177541    -0.118916 -0.240822         -0.827904
4  1.498041    -1.037308    -0.870155 -0.248541          1.183418

   num__Heart Rate  num__Blood Pressure  num__Cholesterol  cat__Gender_Female  \
0        -0.325128             1.135976          0.877000                 1.0
1         0.860695             1.135976          1.635549                 0.0
2        -1.054866             1.355525         -0.134399                 0.0
3         0.039740             0.111415         -1.173892                 0.0
4        -0.233911            -0.034951         -0.921043                 0.0

   cat__Gender_Male  cat__Blood Type_A  cat__Blood Type_AB  cat__Blood Type_B  \
0               0.0                1.0                 0.0                0.0
1               1.0                0.0                 0.0                1.0
2               1.0                0.0                 0.0                1.0
3               1.0                0.0                 0.0                0.0
4               1.0                1.0                 0.0                0.0

   cat__Blood Type_O  cat__Smoking_No  cat__Smoking_Yes
0                0.0              1.0               0.0
1                0.0              1.0               0.0
2                0.0              0.0               1.0
3                1.0              0.0               1.0
4                0.0              0.0               1.0
```

[ ]:
```
Med_Stu_Prepared.hist(bins=50, figsize=(20,15), color='dimgray',
 ↪ec='midnightblue')
plt.show()

Med_Stu_Prepared.describe()
```



[ ]:
```
            num__Age   num__Height   num__Weight      num__BMI  \
count  3.600000e+04  3.600000e+04  3.600000e+04  3.600000e+04
mean   2.084752e-16  3.042505e-16  9.456140e-16  2.456800e-16
std    1.000014e+00  1.000014e+00  1.000014e+00  1.000014e+00
min   -1.730328e+00 -1.798322e+00 -1.826641e+00 -1.967937e+00
25%   -8.694298e-01 -8.102296e-01 -8.100104e-01 -7.342704e-01
50%   -8.531266e-03 -3.771884e-03 -3.164960e-04 -8.806557e-02
75%    8.523673e-01  8.092148e-01  8.156271e-01  5.953376e-01
max    1.713266e+00  1.838249e+00  1.815866e+00  3.118318e+00

       num__Temperature  num__Heart Rate  num__Blood Pressure  \
count      3.600000e+04     3.600000e+04         3.600000e+04
mean      -2.901393e-14    -3.720481e-16         3.720481e-16
std        1.000014e+00     1.000014e+00         1.000014e+00
min       -4.607862e+00    -1.784603e+00        -1.791342e+00
25%       -6.183490e-01    -7.812143e-01        -7.667805e-01
50%       -2.335110e-03     3.974028e-02        -3.495096e-02
75%        6.297012e-01     7.694777e-01         8.432445e-01
max        4.127724e+00     1.772867e+00         1.794623e+00
```

```
       num__Cholesterol  cat__Gender_Female  cat__Gender_Male  \
count      3.600000e+04        36000.000000      36000.000000
mean       3.821141e-16            0.444472          0.555528
std        1.000014e+00            0.496914          0.496914
min       -1.820064e+00            0.000000          0.000000
25%       -8.086649e-01            0.000000          0.000000
50%        6.073076e-03            0.000000          1.000000
75%        8.208111e-01            1.000000          1.000000
max        1.804116e+00            1.000000          1.000000

       cat__Blood Type_A  cat__Blood Type_AB  cat__Blood Type_B  \
count       36000.000000        36000.000000       36000.000000
mean            0.221556            0.222750           0.328278
std             0.415299            0.416098           0.469593
min             0.000000            0.000000           0.000000
25%             0.000000            0.000000           0.000000
50%             0.000000            0.000000           0.000000
75%             0.000000            0.000000           1.000000
max             1.000000            1.000000           1.000000

       cat__Blood Type_O  cat__Smoking_No  cat__Smoking_Yes
count       36000.000000     36000.000000      36000.000000
mean            0.227417         0.818444          0.181556
std             0.419170         0.385483          0.385483
min             0.000000         0.000000          0.000000
25%             0.000000         1.000000          0.000000
50%             0.000000         1.000000          0.000000
75%             0.000000         1.000000          0.000000
max             1.000000         1.000000          1.000000
```

The cell below shows the re-appending of the label to the dataset after it is prepared

[ ]:
```
Med_Stu_Prepared["Diabetes"] = Med_Stu_label
Med_Stu_Prepared.shape
```

[ ]:
```
(36000, 17)
```

Encoding of the label using the label encoder

[ ]:
```
label_encoder = preprocessing.LabelEncoder()
Med_Stu_Prepared["Diabetes"]= label_encoder.
 ↪fit_transform(Med_Stu_Prepared["Diabetes"])

Med_Stu_Prepared=shuffle(Med_Stu_Prepared)
Med_Stu_Prepared.head()
```

[ ]:
```
       num__Age  num__Height  num__Weight  num__BMI  num__Temperature  \
22963  1.282817    -1.359811     1.545329 -0.088066         -1.227168
```

```
24263  0.637143     1.753571    -1.491610 -1.755141          0.858773
5690  -1.515104     0.433227    -1.328827 -1.272106         -0.002335
14545  1.498041     0.774989     1.015650  0.287683          1.091207
6027   1.067592    -0.003772    -0.000316 -0.503018          0.814139

       num__Heart Rate  num__Blood Pressure  num__Cholesterol  \
22963        -1.510952            -0.327683         -0.555815
24263        -0.142694            -0.034951         -1.539120
5690         -0.872431             0.623696          1.466983
14545        -0.507563             1.501891         -0.190588
6027          0.130957            -0.327683         -0.443438

       cat__Gender_Female  cat__Gender_Male  cat__Blood Type_A  \
22963                 0.0               1.0                1.0
24263                 0.0               1.0                0.0
5690                  0.0               1.0                0.0
14545                 0.0               1.0                0.0
6027                  1.0               0.0                0.0

       cat__Blood Type_AB  cat__Blood Type_B  cat__Blood Type_O  \
22963                 0.0                0.0                0.0
24263                 0.0                1.0                0.0
5690                  0.0                1.0                0.0
14545                 0.0                1.0                0.0
6027                  0.0                1.0                0.0

       cat__Smoking_No  cat__Smoking_Yes  Diabetes
22963              0.0               1.0         0
24263              1.0               0.0         0
5690               1.0               0.0         1
14545              1.0               0.0         1
6027               0.0               1.0         1
```

The cell below finds the correlation between the label (i.e. "Diabetes"), and the remaining features in the dataset

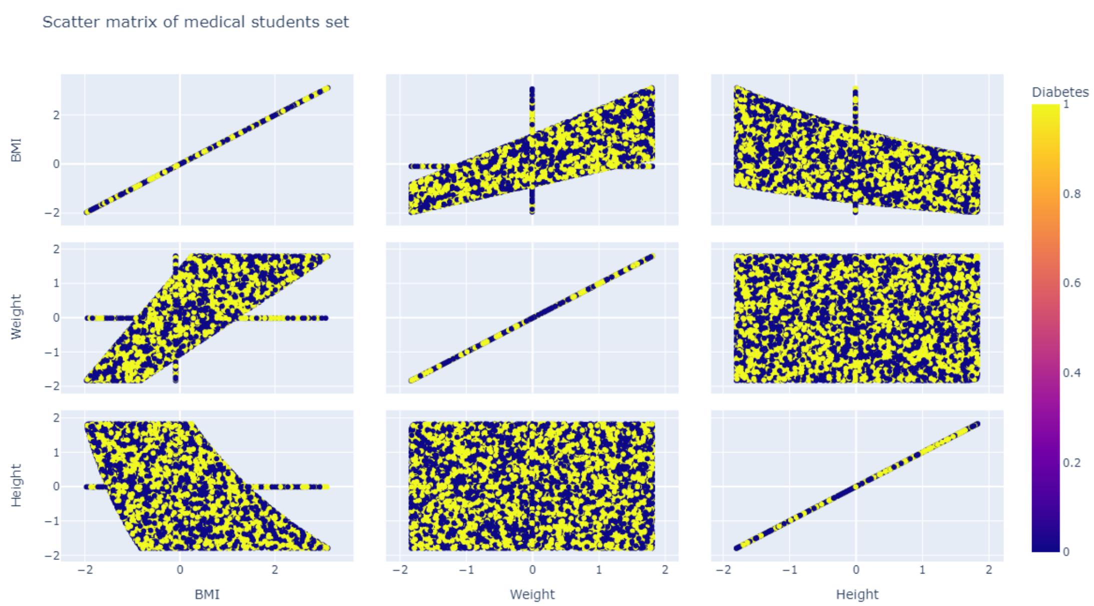
[ ]:
```python
corr_matrix=Med_Stu_Prepared.corr(numeric_only=True)
corr_matrix["Diabetes"].sort_values(ascending=False)
```

[ ]:
```
Diabetes               1.000000
num__Age               0.007611
num__Heart Rate        0.005605
cat__Smoking_No        0.004756
cat__Blood Type_B      0.003668
cat__Blood Type_A      0.003612
num__Weight            0.002761
num__Cholesterol       0.001233
cat__Gender_Female     0.000503
```

```
num__BMI                  0.000398
cat__Gender_Male         -0.000503
num__Blood Pressure      -0.002450
cat__Blood Type_O        -0.003777
cat__Blood Type_AB       -0.003939
cat__Smoking_Yes         -0.004756
num__Height              -0.005167
num__Temperature         -0.010606
Name: Diabetes, dtype: float64
```

[ ]:
```
attributes = ["num__BMI", "num__Weight","num__Height"]
fig = px.scatter_matrix(Med_Stu_Prepared,
    dimensions=attributes,
    color="Diabetes",
    title="Scatter matrix of medical students set",
    labels={col:col.replace('num__', ' ') for col in Med_Stu_Prepared.columns})

image_bytes = fig.to_image(format='png', width=1200, height=700, scale=1)

from IPython.display import Image
Image(image_bytes)
```

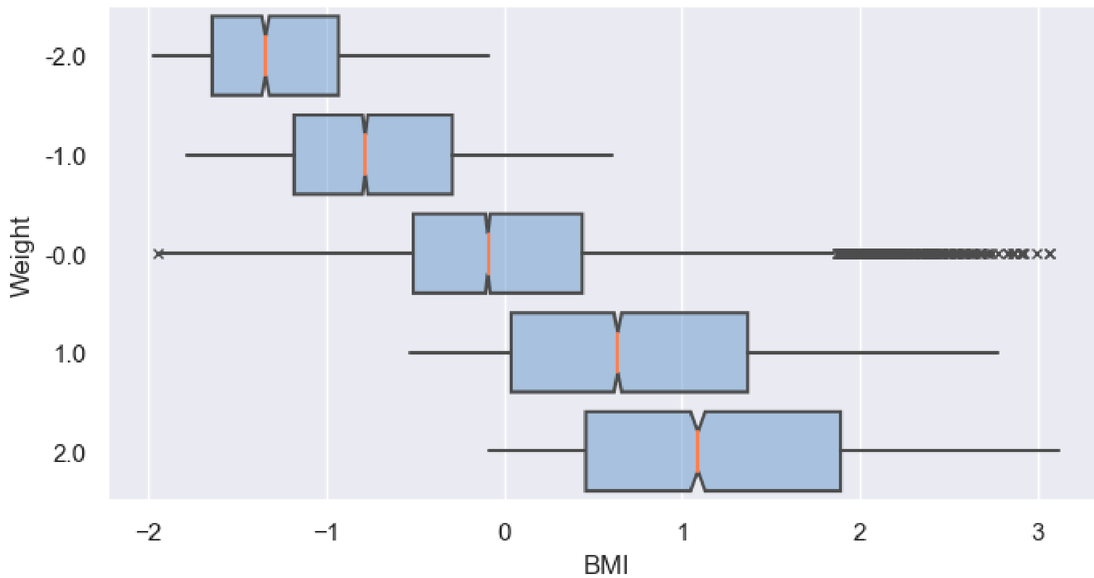
[ ]:



[ ]:
```
ax=sns.set(rc= {"figure.figsize": (8,4)})

ax=sns.boxplot(
```

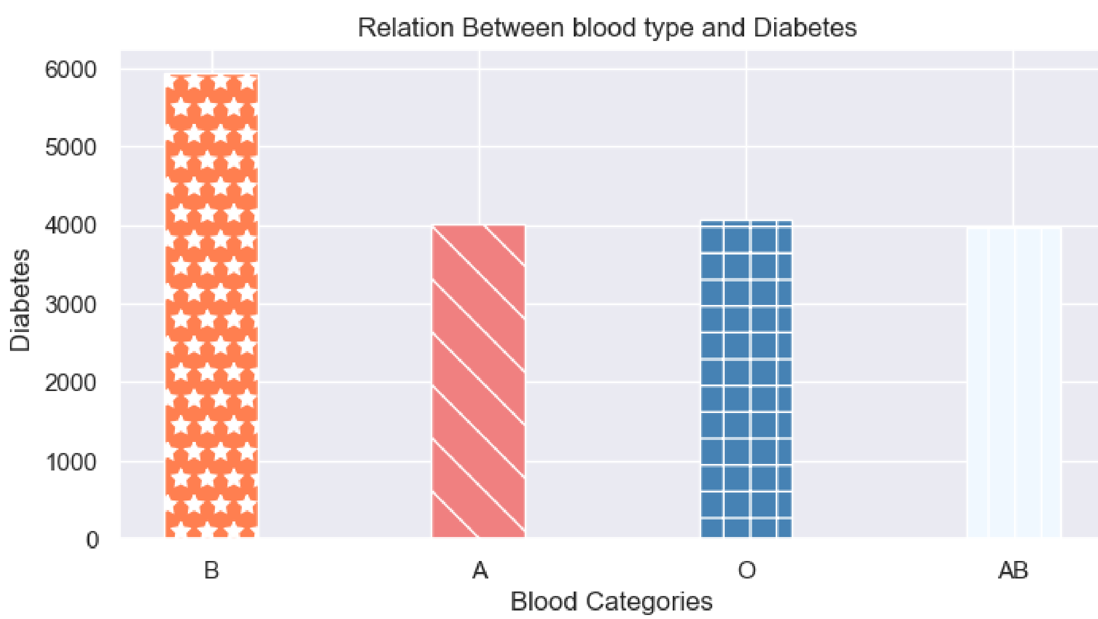
```
    data=Med_Stu_Prepared, x=Med_Stu_Prepared["num__BMI"].round(2),␣
 ↪y=Med_Stu_Prepared["num__Weight"].round(0),
    notch=True, showcaps=False,
    flierprops={"marker": "x"},
    boxprops={"facecolor": (.4, .6, .8, .5)},
    medianprops={"color": "coral"},orient="h",dodge=True
)

ax.set(ylabel="Weight")
ax.set(xlabel="BMI")
```

[ ]: [Text(0.5, 0, 'BMI')]




```
[ ]: x1=(Med_Stu_Prepared[(Med_Stu_Prepared['Diabetes']==1.0) &␣
 ↪(Med_Stu_Prepared['cat__Blood Type_B']==1.0)])
a=x1.shape[0]

x2=(Med_Stu_Prepared[(Med_Stu_Prepared['Diabetes']==1.0) &␣
 ↪(Med_Stu_Prepared['cat__Blood Type_A']==1.0)])
b=x2.shape[0]

x3=(Med_Stu_Prepared[(Med_Stu_Prepared['Diabetes']==1.0) &␣
 ↪(Med_Stu_Prepared['cat__Blood Type_O']==1.0)])
c=x3.shape[0]

x4=(Med_Stu_Prepared[(Med_Stu_Prepared['Diabetes']==1.0) &␣
 ↪(Med_Stu_Prepared['cat__Blood Type_AB']==1.0)])
```

```
d=x4.shape[0]



list=[a,b,c,d]
colors=['coral','lightcoral','steelblue','aliceblue']
patterns=('*','\\','+','|')
labels=['B','A','O','AB']

bar=plt.bar(labels,list,label='Blood Type', width=0.35, color=colors)
for i,x in zip(bar,patterns):
    i.set_hatch(x)

plt.ylabel("Diabetes ")
plt.xlabel('Blood Categories')
plt.title('Relation Between blood type and Diabetes ')
plt.show()
```



1.2.6 *The following three cells of code demonstrate the procedure of splitting the data into a training set and a test set with no label*

[ ]:
```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(Med_Stu_Prepared,␣
 ↪Med_Stu_Prepared["Diabetes"]):
```

```
    train_set = Med_Stu_Prepared.loc[train_index]
    test_set = Med_Stu_Prepared.loc[test_index]
```

[ ]:
```
train = train_set.drop("Diabetes", axis=1)
l1 = train_set["Diabetes"].copy()
len(l1)
```

[ ]: 28800

[ ]:
```
test = test_set.drop("Diabetes", axis=1)
l2 = test_set["Diabetes"].copy()
test.shape
```

[ ]: (7200, 16)

1.3 *PART 2*

Classifiers used:

1-DecisionTreeClassifier
2-LogisticRegression
3-SVC
4-RandomForestClassifier
5-KNeighborsClassifier
6-MLPClassifier

The model is trained and the accuracy is found on and for each one of the classifiers mentioned above

using the grid search, different parameters are tested for each one of the classifiers

1-DecisionTreeClassifier

[ ]:
```
tree_clf = DecisionTreeClassifier(random_state=42)
tree_params = [
    {'criterion': ['gini', 'entropy', 'log_loss'],
     'splitter': ['best','random'],
     'max_depth': [3,4,5]}]
grid_search = GridSearchCV(tree_clf,tree_params, cv=3,
                           scoring='accuracy')
grid_search.fit(train, l1)
```

[ ]:
```
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(random_state=42),
             param_grid=[{'criterion': ['gini', 'entropy', 'log_loss'],
                          'max_depth': [3, 4, 5],
                          'splitter': ['best', 'random']}],
             scoring='accuracy')
```

The accuracy, precision, recall, and f1 scores, for the test set :

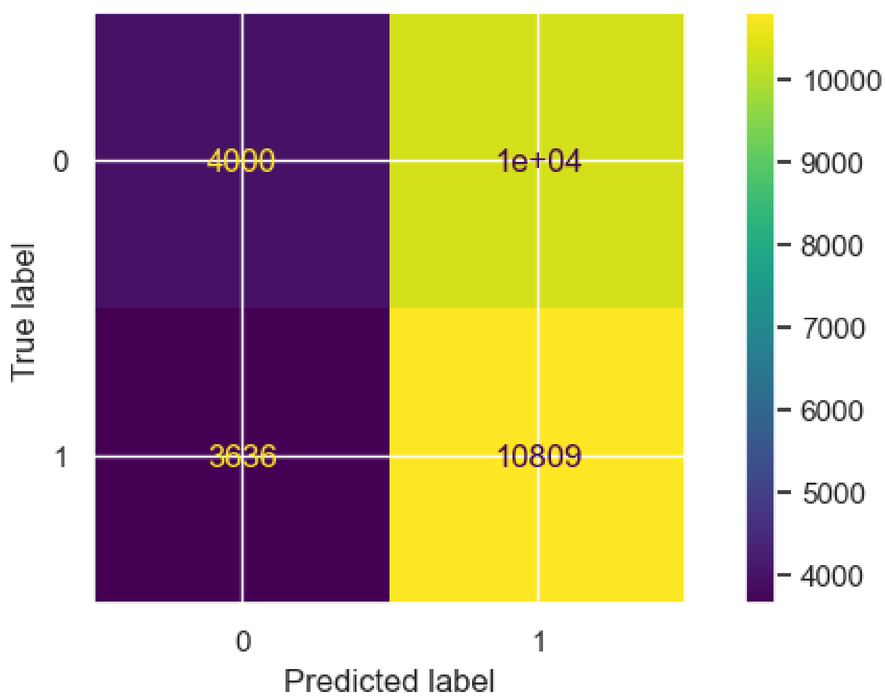
```
[ ]: P1=grid_search.predict(train)
     P2=grid_search.predict(test)
     print("Best parameter = "+str(grid_search.best_params_))
     print("Best estimator = "+str(grid_search.best_estimator_))
     print('The accuracy for DTC = '+str(grid_search.best_score_))
     print("precision = "+str(precision_score(l2,P2)))
     print("recall = "+str(recall_score(l2,P2)))
     print("f1_score = "+str(f1_score(l2,P2)))
```

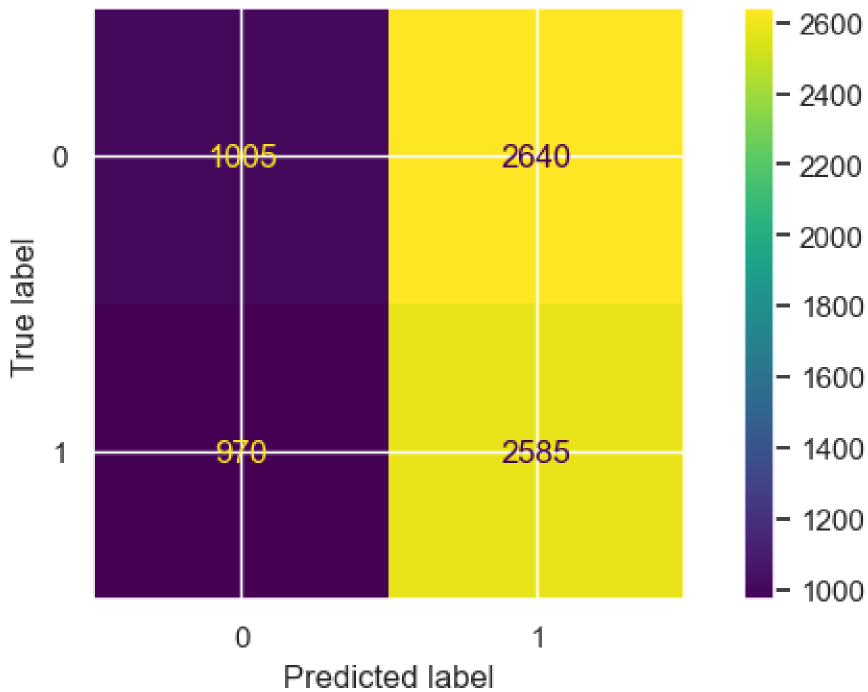
```
Best parameter = {'criterion': 'gini', 'max_depth': 5, 'splitter': 'best'}
Best estimator = DecisionTreeClassifier(max_depth=5, random_state=42)
The accuracy for DTC = 0.5069444444444445
precision = 0.49473684210526314
recall = 0.7271448663853727
f1_score = 0.5888382687927107
```

The confusion matrix for:

1-training set
2-testing set

```
[ ]: ConfusionMatrixDisplay.from_predictions(l1,P1)
     ConfusionMatrixDisplay.from_predictions(l2,P2)
     plt.show()
```





2-LogisticRegression

[ ]:
```
log_reg = LogisticRegression(max_iter=1000)
log_params = [
    {'C': [1,10,100],
     'fit_intercept': [True,False],
     'n_jobs': [-1,10,16],
    }]
grid_search = GridSearchCV(log_reg,log_params, cv=3,
                           scoring='accuracy')
grid_search.fit(train, l1)
```

[ ]:
```
GridSearchCV(cv=3, estimator=LogisticRegression(max_iter=1000),
             param_grid=[{'C': [1, 10, 100], 'fit_intercept': [True, False],
                          'n_jobs': [-1, 10, 16]}],
             scoring='accuracy')
```

The accuracy, precision, recall, and f1 scores, for the test set :

[ ]:
```
P1=grid_search.predict(train)
P2=grid_search.predict(test)
print("Best parameter = "+str(grid_search.best_params_))
print("Best estimator = "+str(grid_search.best_estimator_))
print('The accuracy for LogisticRegression = '+str(grid_search.best_score_))
print("precision = "+str(precision_score(l2,P2)))
```

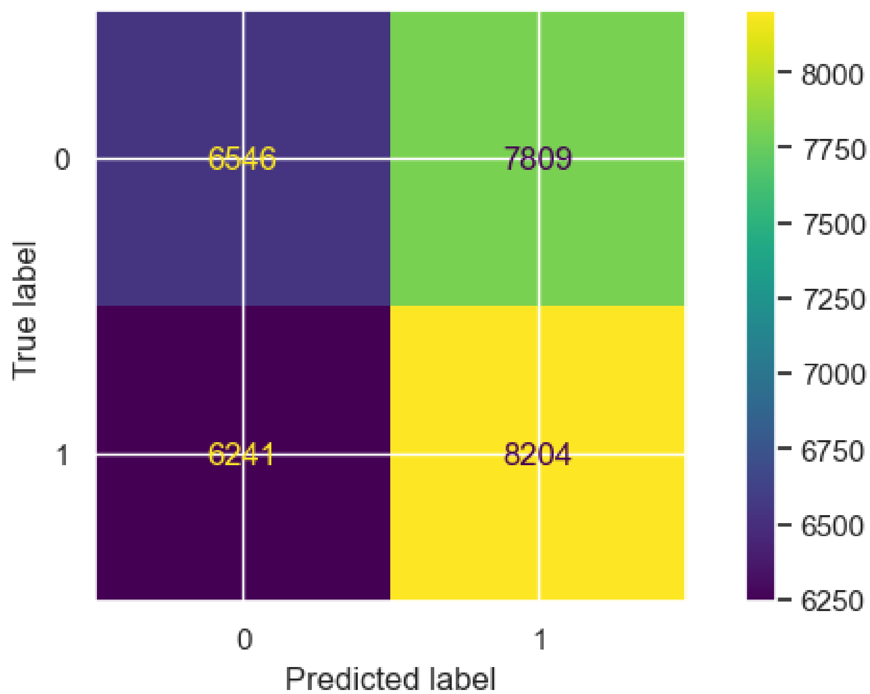
```
print("recall = "+str(recall_score(l2,P2)))
print("f1_score = "+str(f1_score(l2,P2)))
```

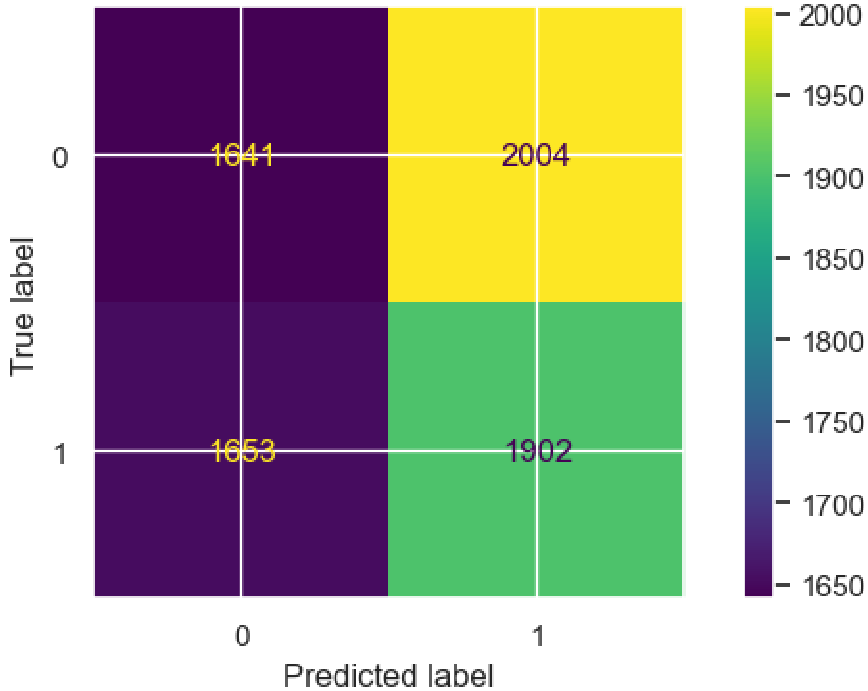
```
Best parameter = {'C': 1, 'fit_intercept': True, 'n_jobs': -1}
Best estimator = LogisticRegression(C=1, max_iter=1000, n_jobs=-1)
The accuracy for LogisticRegression = 0.5032638888888888
precision = 0.4869431643625192
recall = 0.5350210970464135
f1_score = 0.5098512263771613
```

The confusion matrix for:

```
1-training set
2-testing set
```

[ ]:
```
ConfusionMatrixDisplay.from_predictions(l1,P1)
ConfusionMatrixDisplay.from_predictions(l2,P2)
plt.show()
```





3-SupportVectorClassifier

[ ]:
```
svm = SVC()
param_grid = [
    {
     'gamma': [2, 1],
     'C':[0.001,0.1],
    }]


grid_search = GridSearchCV(svm, param_grid, cv=3,scoring='accuracy')
grid_search.fit(train[:7000],l1[:7000])
```

[ ]:
```
GridSearchCV(cv=3, estimator=SVC(),
             param_grid=[{'C': [0.001, 0.1], 'gamma': [2, 1]}],
             scoring='accuracy')
```

The accuracy, precision, recall, and f1 scores, for the test set :

[ ]:
```
P1=grid_search.predict(train)
P2=grid_search.predict(test)
print("Best parameter = "+str(grid_search.best_params_))
print("Best estimator = "+str(grid_search.best_estimator_))
print('The accuracy for SVC = '+str(grid_search.best_score_))
print("precision = "+str(precision_score(l2,P2)))
```

```
print("recall = "+str(recall_score(l2,P2)))
print("f1_score = "+str(f1_score(l2,P2)))
```

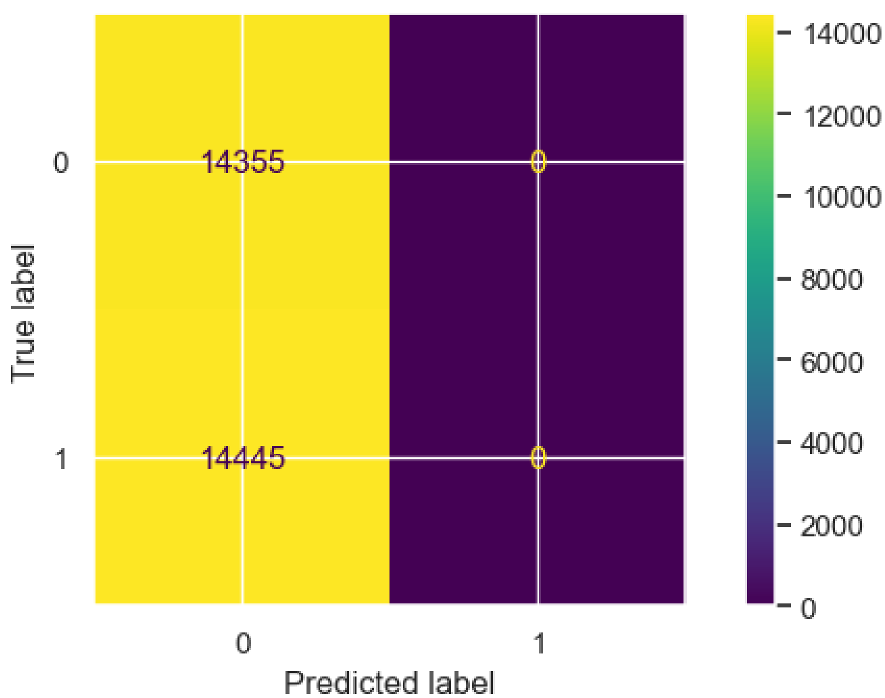
```
Best parameter = {'C': 0.001, 'gamma': 2}
Best estimator = SVC(C=0.001, gamma=2)
The accuracy for SVC = 0.5107142861515411
precision = 0.0
recall = 0.0
f1_score = 0.0
```

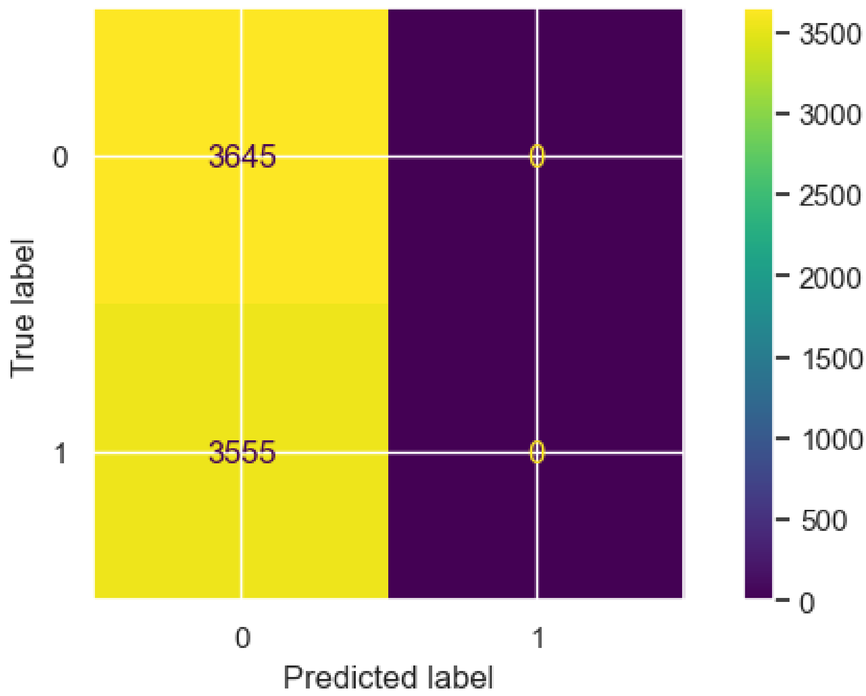
```
c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
```

The confusion matrix for:

1-training set
2-testing set

[ ]:
```
ConfusionMatrixDisplay.from_predictions(l1,P1)
ConfusionMatrixDisplay.from_predictions(l2,P2)
plt.show()
```





4-RandomForestClassifier

```
[ ]: RFC= RandomForestClassifier()
     param_grid = [{
         'n_estimators':[500,1000],
         'min_samples_split':[2,3],
         'max_depth' :[2,6]
     }]
     grid_search = GridSearchCV(RFC, param_grid, cv=3,␣
      ↪scoring='accuracy',error_score='raise')
     grid_search.fit(train[:15000],l1[:15000])
```

```
[ ]: GridSearchCV(cv=3, error_score='raise', estimator=RandomForestClassifier(),
                  param_grid=[{'max_depth': [2, 6], 'min_samples_split': [2, 3],
                               'n_estimators': [500, 1000]}],
                  scoring='accuracy')
```

The accuracy, precision, recall, and f1 scores, for the test set :

```
[ ]: P1=grid_search.predict(train)
     P2=grid_search.predict(test)
     print("Best parameter = "+str(grid_search.best_params_))
     print("Best estimator = "+str(grid_search.best_estimator_))
     print('The accuracy for RFC = '+str(grid_search.best_score_))
     print("precision = "+str(precision_score(l2,P2)))
```

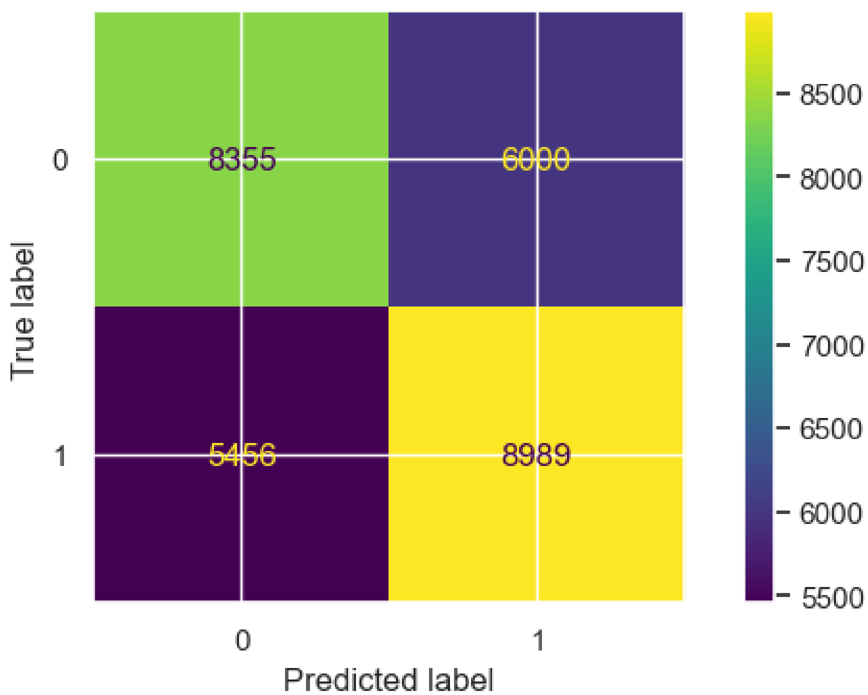
```
print("recall = "+str(recall_score(l2,P2)))
print("f1_score = "+str(f1_score(l2,P2)))
```

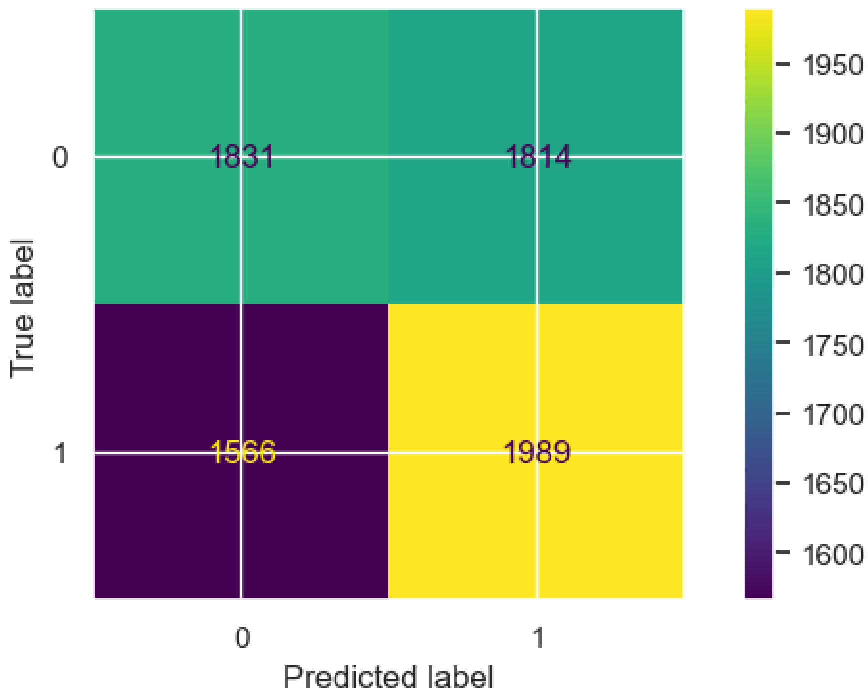
```
Best parameter = {'max_depth': 6, 'min_samples_split': 3, 'n_estimators': 500}
Best estimator = RandomForestClassifier(max_depth=6, min_samples_split=3,
n_estimators=500)
The accuracy for RFC = 0.5129333333333334
precision = 0.5230081514593742
recall = 0.5594936708860759
f1_score = 0.5406360424028268
```

The confusion matrix for:

1-training set
2-testing set

[ ]:
```
ConfusionMatrixDisplay.from_predictions(l1,P1)
ConfusionMatrixDisplay.from_predictions(l2,P2)
plt.show()
```





5-KNeighborsClassifier

[ ]:
```
knn_clf = KNeighborsClassifier()
param_grid = [{
    'n_neighbors':[3,4,5],
    'weights':['uniform','distance']
}]
grid_search = GridSearchCV(knn_clf, param_grid, cv=3,scoring='accuracy')
grid_search.fit(train,l1)
```

```
c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py:824: UserWarning:

Scoring failed. The score on this train-test partition for these parameters will
be set to nan. Details:
Traceback (most recent call last):
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py", line 813, in _score
    scores = scorer(estimator, X_test, y_test)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 266, in __call__
    return self._score(partial(_cached_call, None), estimator, X, y_true,
**_kwargs)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 353, in _score
```

```
    y_pred = method_caller(estimator, "predict", X)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 86, in _cached_call
    result, _ = _get_response_values(
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\_response.py", line 85, in _get_response_values
    y_pred = prediction_method(X)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neighbors\_classification.py", line 246, in predict
    if self._fit_method == "brute" and ArgKminClassMode.is_usable_for(
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line
471, in is_usable_for
    ArgKmin.is_usable_for(X, Y, metric)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line
115, in is_usable_for
    and (is_numpy_c_ordered(X) or is_valid_sparse_matrix(X))
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line 99,
in is_numpy_c_ordered
    return hasattr(X, "flags") and X.flags.c_contiguous
AttributeError: 'Flags' object has no attribute 'c_contiguous'


c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py:824: UserWarning:

Scoring failed. The score on this train-test partition for these parameters will
be set to nan. Details:
Traceback (most recent call last):
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py", line 813, in _score
    scores = scorer(estimator, X_test, y_test)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 266, in __call__
    return self._score(partial(_cached_call, None), estimator, X, y_true,
**_kwargs)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 353, in _score
    y_pred = method_caller(estimator, "predict", X)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 86, in _cached_call
    result, _ = _get_response_values(
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\_response.py", line 85, in _get_response_values
    y_pred = prediction_method(X)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
```

```
packages\sklearn\neighbors\_classification.py", line 246, in predict
    if self._fit_method == "brute" and ArgKminClassMode.is_usable_for(
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line
471, in is_usable_for
    ArgKmin.is_usable_for(X, Y, metric)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line
115, in is_usable_for
    and (is_numpy_c_ordered(X) or is_valid_sparse_matrix(X))
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line 99,
in is_numpy_c_ordered
    return hasattr(X, "flags") and X.flags.c_contiguous
AttributeError: 'Flags' object has no attribute 'c_contiguous'


c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py:824: UserWarning:

Scoring failed. The score on this train-test partition for these parameters will
be set to nan. Details:
Traceback (most recent call last):
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py", line 813, in _score
    scores = scorer(estimator, X_test, y_test)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 266, in __call__
    return self._score(partial(_cached_call, None), estimator, X, y_true,
**_kwargs)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 353, in _score
    y_pred = method_caller(estimator, "predict", X)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 86, in _cached_call
    result, _ = _get_response_values(
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\_response.py", line 85, in _get_response_values
    y_pred = prediction_method(X)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neighbors\_classification.py", line 246, in predict
    if self._fit_method == "brute" and ArgKminClassMode.is_usable_for(
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line
471, in is_usable_for
    ArgKmin.is_usable_for(X, Y, metric)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line
```

```
115, in is_usable_for
    and (is_numpy_c_ordered(X) or is_valid_sparse_matrix(X))
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line 99,
in is_numpy_c_ordered
    return hasattr(X, "flags") and X.flags.c_contiguous
AttributeError: 'Flags' object has no attribute 'c_contiguous'


c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py:824: UserWarning:

Scoring failed. The score on this train-test partition for these parameters will
be set to nan. Details:
Traceback (most recent call last):
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py", line 813, in _score
    scores = scorer(estimator, X_test, y_test)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 266, in __call__
    return self._score(partial(_cached_call, None), estimator, X, y_true,
**_kwargs)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 353, in _score
    y_pred = method_caller(estimator, "predict", X)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 86, in _cached_call
    result, _ = _get_response_values(
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\_response.py", line 85, in _get_response_values
    y_pred = prediction_method(X)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neighbors\_classification.py", line 246, in predict
    if self._fit_method == "brute" and ArgKminClassMode.is_usable_for(
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line
471, in is_usable_for
    ArgKmin.is_usable_for(X, Y, metric)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line
115, in is_usable_for
    and (is_numpy_c_ordered(X) or is_valid_sparse_matrix(X))
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line 99,
in is_numpy_c_ordered
    return hasattr(X, "flags") and X.flags.c_contiguous
AttributeError: 'Flags' object has no attribute 'c_contiguous'
```

```
c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py:824: UserWarning:

Scoring failed. The score on this train-test partition for these parameters will
be set to nan. Details:
Traceback (most recent call last):
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py", line 813, in _score
    scores = scorer(estimator, X_test, y_test)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 266, in __call__
    return self._score(partial(_cached_call, None), estimator, X, y_true,
**_kwargs)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 353, in _score
    y_pred = method_caller(estimator, "predict", X)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 86, in _cached_call
    result, _ = _get_response_values(
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\_response.py", line 85, in _get_response_values
    y_pred = prediction_method(X)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neighbors\_classification.py", line 246, in predict
    if self._fit_method == "brute" and ArgKminClassMode.is_usable_for(
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line
471, in is_usable_for
    ArgKmin.is_usable_for(X, Y, metric)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line
115, in is_usable_for
    and (is_numpy_c_ordered(X) or is_valid_sparse_matrix(X))
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line 99,
in is_numpy_c_ordered
    return hasattr(X, "flags") and X.flags.c_contiguous
AttributeError: 'Flags' object has no attribute 'c_contiguous'


c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py:824: UserWarning:

Scoring failed. The score on this train-test partition for these parameters will
be set to nan. Details:
Traceback (most recent call last):
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
```

```
packages\sklearn\model_selection\_validation.py", line 813, in _score
    scores = scorer(estimator, X_test, y_test)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 266, in __call__
    return self._score(partial(_cached_call, None), estimator, X, y_true,
**_kwargs)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 353, in _score
    y_pred = method_caller(estimator, "predict", X)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 86, in _cached_call
    result, _ = _get_response_values(
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\_response.py", line 85, in _get_response_values
    y_pred = prediction_method(X)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neighbors\_classification.py", line 246, in predict
    if self._fit_method == "brute" and ArgKminClassMode.is_usable_for(
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line
471, in is_usable_for
    ArgKmin.is_usable_for(X, Y, metric)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line
115, in is_usable_for
    and (is_numpy_c_ordered(X) or is_valid_sparse_matrix(X))
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line 99,
in is_numpy_c_ordered
    return hasattr(X, "flags") and X.flags.c_contiguous
AttributeError: 'Flags' object has no attribute 'c_contiguous'


c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py:824: UserWarning:

Scoring failed. The score on this train-test partition for these parameters will
be set to nan. Details:
Traceback (most recent call last):
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py", line 813, in _score
    scores = scorer(estimator, X_test, y_test)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 266, in __call__
    return self._score(partial(_cached_call, None), estimator, X, y_true,
**_kwargs)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 353, in _score
```

```
    y_pred = method_caller(estimator, "predict", X)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 86, in _cached_call
    result, _ = _get_response_values(
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\_response.py", line 85, in _get_response_values
    y_pred = prediction_method(X)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neighbors\_classification.py", line 246, in predict
    if self._fit_method == "brute" and ArgKminClassMode.is_usable_for(
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line
471, in is_usable_for
    ArgKmin.is_usable_for(X, Y, metric)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line
115, in is_usable_for
    and (is_numpy_c_ordered(X) or is_valid_sparse_matrix(X))
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line 99,
in is_numpy_c_ordered
    return hasattr(X, "flags") and X.flags.c_contiguous
AttributeError: 'Flags' object has no attribute 'c_contiguous'


c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py:824: UserWarning:

Scoring failed. The score on this train-test partition for these parameters will
be set to nan. Details:
Traceback (most recent call last):
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py", line 813, in _score
    scores = scorer(estimator, X_test, y_test)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 266, in __call__
    return self._score(partial(_cached_call, None), estimator, X, y_true,
**_kwargs)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 353, in _score
    y_pred = method_caller(estimator, "predict", X)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 86, in _cached_call
    result, _ = _get_response_values(
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\_response.py", line 85, in _get_response_values
    y_pred = prediction_method(X)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
```

```
packages\sklearn\neighbors\_classification.py", line 246, in predict
    if self._fit_method == "brute" and ArgKminClassMode.is_usable_for(
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line
471, in is_usable_for
    ArgKmin.is_usable_for(X, Y, metric)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line
115, in is_usable_for
    and (is_numpy_c_ordered(X) or is_valid_sparse_matrix(X))
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line 99,
in is_numpy_c_ordered
    return hasattr(X, "flags") and X.flags.c_contiguous
AttributeError: 'Flags' object has no attribute 'c_contiguous'


c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py:824: UserWarning:

Scoring failed. The score on this train-test partition for these parameters will
be set to nan. Details:
Traceback (most recent call last):
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py", line 813, in _score
    scores = scorer(estimator, X_test, y_test)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 266, in __call__
    return self._score(partial(_cached_call, None), estimator, X, y_true,
**_kwargs)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 353, in _score
    y_pred = method_caller(estimator, "predict", X)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_scorer.py", line 86, in _cached_call
    result, _ = _get_response_values(
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\_response.py", line 85, in _get_response_values
    y_pred = prediction_method(X)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neighbors\_classification.py", line 246, in predict
    if self._fit_method == "brute" and ArgKminClassMode.is_usable_for(
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line
471, in is_usable_for
    ArgKmin.is_usable_for(X, Y, metric)
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line
```

```
115, in is_usable_for
    and (is_numpy_c_ordered(X) or is_valid_sparse_matrix(X))
  File "c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\metrics\_pairwise_distances_reduction\_dispatcher.py", line 99,
in is_numpy_c_ordered
    return hasattr(X, "flags") and X.flags.c_contiguous
AttributeError: 'Flags' object has no attribute 'c_contiguous'


c:\Users\DELL-G5\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_search.py:976: UserWarning:

One or more of the test scores are non-finite: [       nan 0.54784722        nan
0.548125          nan 0.55010417]
```

[ ]:
```
GridSearchCV(cv=3, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': [3, 4, 5],
                          'weights': ['uniform', 'distance']}],
             scoring='accuracy')
```

The accuracy, precision, recall, and f1 scores, for the test set :

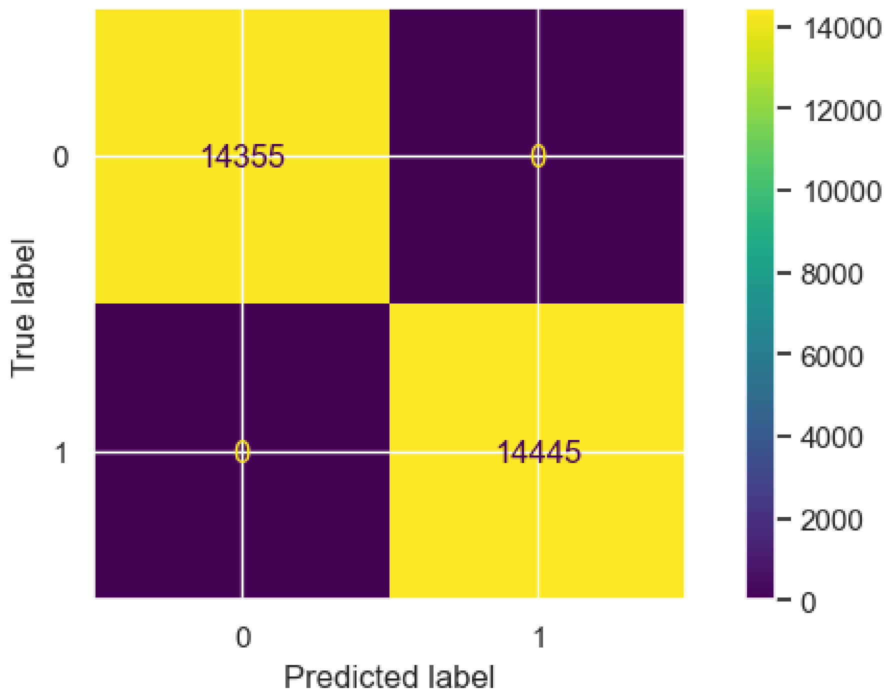
[ ]:
```
P1=grid_search.predict(train)
P2=grid_search.predict(test)
print("Best parameter = "+str(grid_search.best_params_))
print("Best estimator = "+str(grid_search.best_estimator_))
print('The accuracy for KNeighborsClassifier = '+str(grid_search.best_score_))
print("precision = "+str(precision_score(l2,P2)))
print("recall = "+str(recall_score(l2,P2)))
print("f1_score = "+str(f1_score(l2,P2)))
```

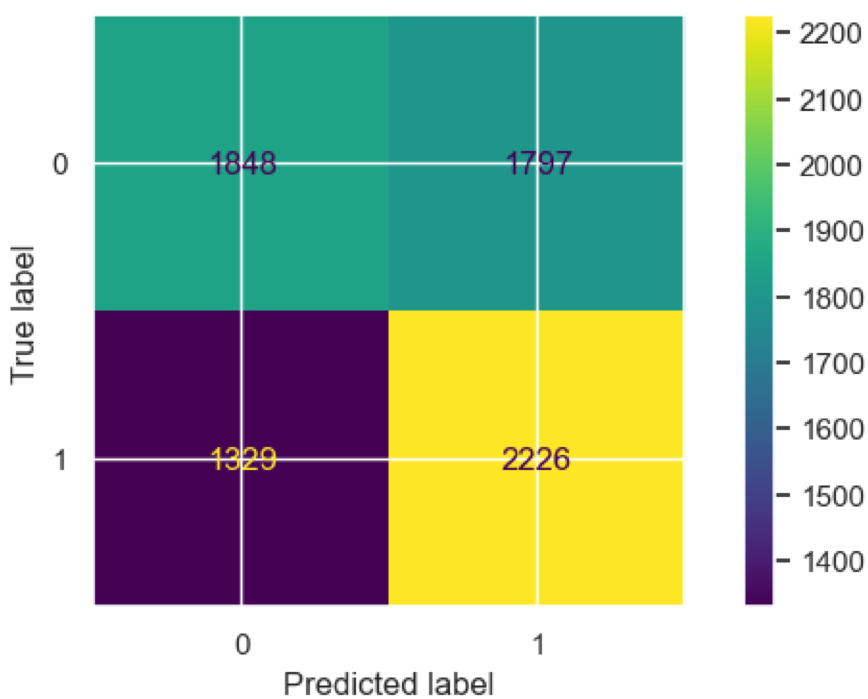
```
Best parameter = {'n_neighbors': 5, 'weights': 'distance'}
Best estimator = KNeighborsClassifier(weights='distance')
The accuracy for KNeighborsClassifier = 0.5501041666666666
precision = 0.5533184190902312
recall = 0.6261603375527426
f1_score = 0.5874901029295329
```

The confusion matrix for:

1-training set
2-testing set

[ ]:
```
ConfusionMatrixDisplay.from_predictions(l1,P1)
ConfusionMatrixDisplay.from_predictions(l2,P2)
plt.show()
```





6-MLPClassifier

```
[ ]: mlp_clf = MLPClassifier(max_iter = 2000)
param_grid = [{
    'activation':['identity', 'logistic'],
    'alpha':[0.0001,0.001],
    'learning_rate' :['constant','invscaling']
}]
grid_search = GridSearchCV(mlp_clf, param_grid, cv=3,scoring='accuracy')
grid_search.fit(train,l1)
```

```
[ ]: GridSearchCV(cv=3, estimator=MLPClassifier(max_iter=2000),
             param_grid=[{'activation': ['identity', 'logistic'],
                          'alpha': [0.0001, 0.001],
                          'learning_rate': ['constant', 'invscaling']}],
             scoring='accuracy')
```

The accuracy, precision, recall, and f1 scores, for the test set :

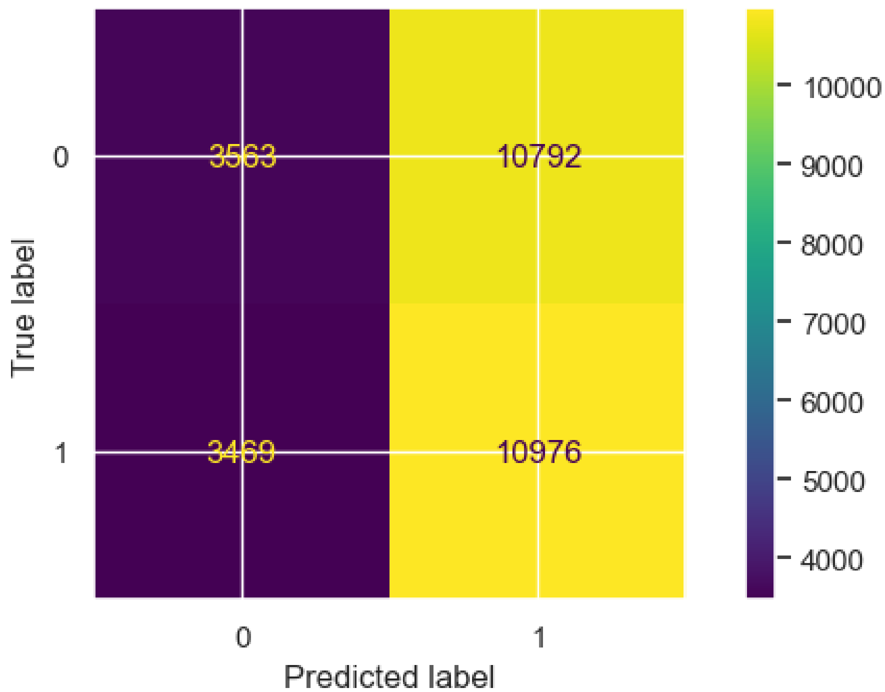
```
[ ]: P1=grid_search.predict(train)
P2=grid_search.predict(test)
print("Best parameter = "+str(grid_search.best_params_))
print("Best estimator = "+str(grid_search.best_estimator_))
print('The accuracy for MLPClassifier = '+str(grid_search.best_score_))
print("precision = "+str(precision_score(l2,P2)))
print("recall = "+str(recall_score(l2,P2)))
print("f1_score = "+str(f1_score(l2,P2)))
```

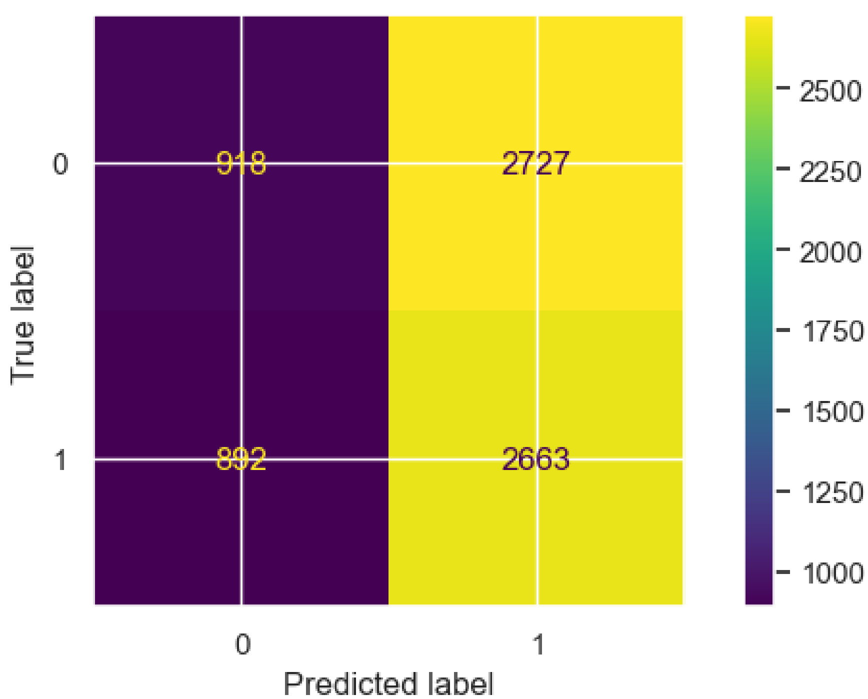
```
Best parameter = {'activation': 'logistic', 'alpha': 0.0001, 'learning_rate':
'invscaling'}
Best estimator = MLPClassifier(activation='logistic',
learning_rate='invscaling', max_iter=2000)
The accuracy for MLPClassifier = 0.5059722222222222
precision = 0.49406307977736547
recall = 0.7490857946554149
f1_score = 0.5954164337618781
```

The confusion matrix for:

1-training set
2-testing set

```
[ ]: ConfusionMatrixDisplay.from_predictions(l1,P1)
ConfusionMatrixDisplay.from_predictions(l2,P2)
plt.show()
```





SGDClassifier used to find the tradeoff between the precision and the recall

[ ]:
```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(train, l1)
```

[ ]:
```
SGDClassifier(random_state=42)
```

The accuracy, precision, recall, and f1 scores, for the test set :

[ ]:
```
P1=cross_val_predict(sgd_clf, train, l1, cv=3)
P2=cross_val_predict(sgd_clf, test, l2, cv=3)

print('The accuracy for SGDClassifier  = '+str(accuracy_score(l2,P2)))
print("precision = "+str(precision_score(l2,P2)))
print("recall = "+str(recall_score(l2,P2)))
print("f1_score = "+str(f1_score(l2,P2)))
```

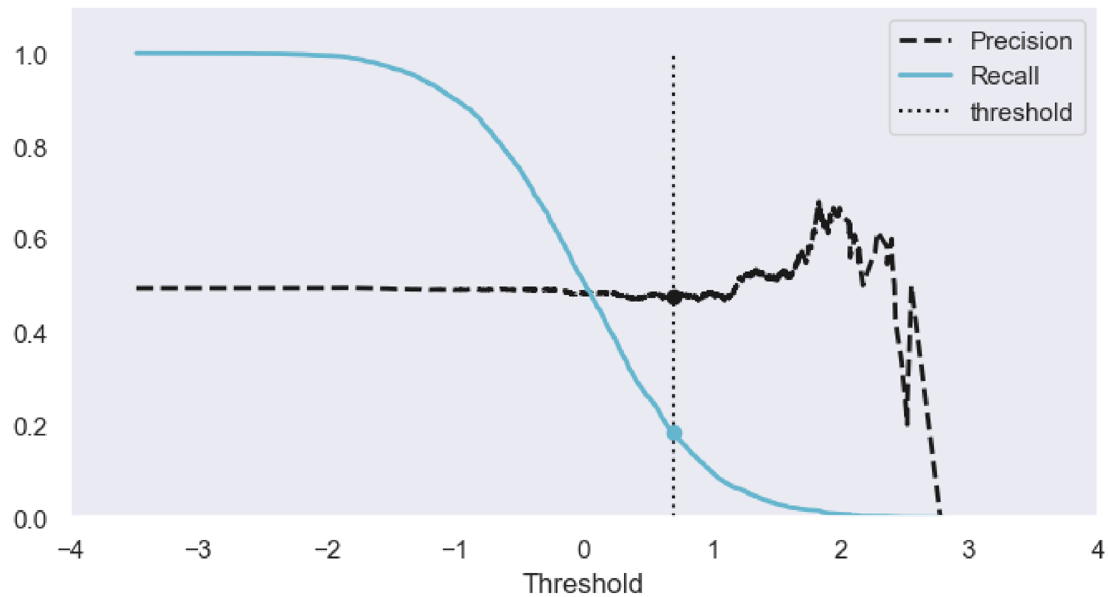
```
The accuracy for SGDClassifier  = 0.4884722222222222
precision = 0.4829151094500801
recall = 0.5088607594936709
f1_score = 0.49554855499246686
```

[ ]:
```
l2_scores = cross_val_predict(sgd_clf, test, l2, cv=3,
                              method="decision_function")
precisions, recalls, thresholds = precision_recall_curve(l2, l2_scores)
print("Precisions : ",precisions,"\n\n","Recalls : ", recalls,"\n\n",␣
 ↪"Thresholds : ",thresholds)
```

```
Precisions :  [0.49375    0.49367968 0.49374826 … 0.5        0.         1.
]

 Recalls :  [1.00000000e+00 9.99718706e-01 9.99718706e-01 … 2.81293952e-04
 0.00000000e+00 0.00000000e+00]

 Thresholds :  [-3.47967162 -2.80252796 -2.62449745 …  2.53324375  2.5524734
  2.77912954]
```

[ ]:
```
threshold=0.7
plt.figure(figsize=(8, 4))
plt.plot(thresholds, precisions[:-1], "k--", label="Precision", linewidth=2)
plt.plot(thresholds, recalls[:-1], "c-", label="Recall", linewidth=2)
plt.vlines(threshold, 0, 1.0, "k", "dotted", label="threshold")
idx = (thresholds >= threshold).argmax()
plt.plot(thresholds[idx], precisions[idx], "ko")
plt.plot(thresholds[idx], recalls[idx], "co")
plt.axis([-4,4, 0, 1.1])
plt.grid()
plt.xlabel("Threshold")
plt.legend(loc="upper right")
```

```
plt.show()
```



1.4 PART 3

Neural networks

Three models are created in this part, two of which are sequential, and one is functional, as shown in the table below :

Properties	***seqModel1***	***seqModel2***	***funcModel***
Activation function	*RelU*	*elU*	*RelU*
Optimizer	*Adam*	*RMSProp*	*SGD*
Initializer	*he_uniform*	*he_uniform*	*he_uniform*
Dropout	*Yes (0.15)*	*Yes (0.15)*	*No*

1.4.1 *Here we split the training set to get validation set which represents 20% of the training set*

[ ]:
```
x_train= train[:23040]
y_train = l1[:23040]
x_valid= train[23040:]
y_valid = l1[23040:]
x_train.shape
```

[ ]: (23040, 16)

1.4.2 *seqModel1 :*

[ ]:
```
tf.random.set_seed(42)
model = tf.keras.Sequential()
model.add(layers.InputLayer(input_shape=(16)))

model.add(layers.Dense(400,kernel_initializer="he_normal",activation="relu"))
model.add(layers.Dropout(0.15))
model.add(layers.BatchNormalization())

model.add(layers.Dense(200,kernel_initializer="he_normal",activation="relu"))
model.add(layers.Dropout(0.15))
model.add(layers.BatchNormalization())

model.add(layers.Dense(100,kernel_initializer="he_normal",activation="relu"))
model.add(layers.Dropout(0.15))
model.add(layers.BatchNormalization())

model.add(layers.Dense(50,kernel_initializer="he_normal",activation="relu"))
model.add(layers.Dropout(0.15))
model.add(layers.BatchNormalization())

model.add(layers.Dense(25,kernel_initializer="he_normal",activation="relu"))
model.add(layers.Dropout(0.15))
model.add(layers.BatchNormalization())

model.add(layers.Dense(2,kernel_initializer="he_normal",activation="softmax"))
```

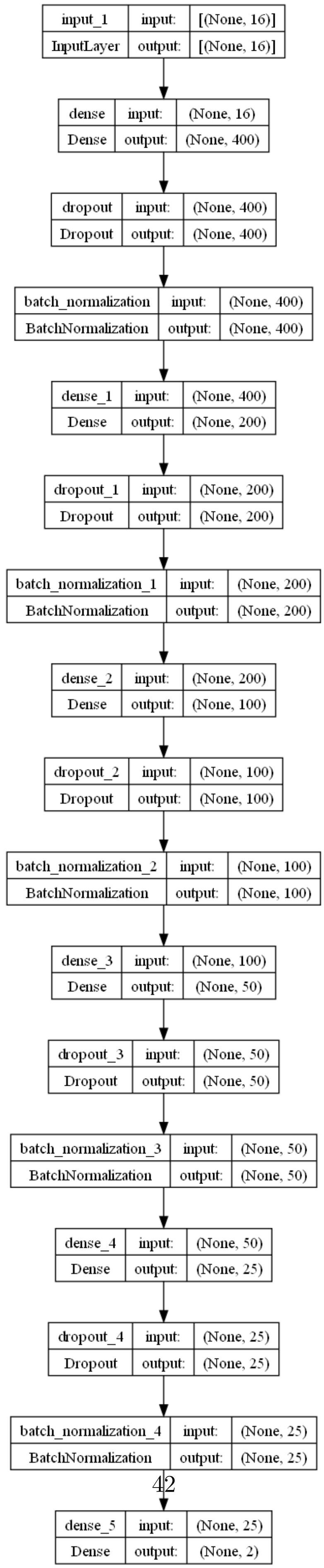
[ ]:
```
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 400)               6800

 dropout (Dropout)           (None, 400)               0

 batch_normalization (Batch  (None, 400)               1600
 Normalization)

 dense_1 (Dense)             (None, 200)               80200

 dropout_1 (Dropout)         (None, 200)               0

 batch_normalization_1 (Bat  (None, 200)               800
 chNormalization)
```

```
 dense_2 (Dense)             (None, 100)               20100

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 400)               6800

 dropout (Dropout)           (None, 400)               0

 batch_normalization (Batch  (None, 400)               1600
 Normalization)

 dense_1 (Dense)             (None, 200)               80200

 dropout_1 (Dropout)         (None, 200)               0

 batch_normalization_1 (Bat  (None, 200)               800
 chNormalization)

 dense_2 (Dense)             (None, 100)               20100

 dropout_2 (Dropout)         (None, 100)               0

 batch_normalization_2 (Bat  (None, 100)               400
 chNormalization)

 dense_3 (Dense)             (None, 50)                5050

 dropout_3 (Dropout)         (None, 50)                0

 batch_normalization_3 (Bat  (None, 50)                200
 chNormalization)

 dense_4 (Dense)             (None, 25)                1275

 dropout_4 (Dropout)         (None, 25)                0

 batch_normalization_4 (Bat  (None, 25)                100
 chNormalization)

 dense_5 (Dense)             (None, 2)                 52

=================================================================
Total params: 116577 (455.38 KB)
Trainable params: 115027 (449.32 KB)
Non-trainable params: 1550 (6.05 KB)
_________________________________________________________________
```

[ ]:
```
tf.keras.utils.plot_model(model, "medstu_model0.png", show_shapes=True)
```

[ ]:


input_1 | input: | [(None, 16)]
InputLayer | output: | [(None, 16)]
dense | input: | (None, 16)
Dense | output: | (None, 400)
dropout | input: | (None, 400)
Dropout | output: | (None, 400)
batch_normalization | input: | (None, 400)
BatchNormalization | output: | (None, 400)
dense_1 | input: | (None, 400)
Dense | output: | (None, 200)
dropout_1 | input: | (None, 200)
Dropout | output: | (None, 200)
batch_normalization_1 | input: | (None, 200)
BatchNormalization | output: | (None, 200)
dense_2 | input: | (None, 200)
Dense | output: | (None, 100)
dropout_2 | input: | (None, 100)
Dropout | output: | (None, 100)
batch_normalization_2 | input: | (None, 100)
BatchNormalization | output: | (None, 100)
dense_3 | input: | (None, 100)
Dense | output: | (None, 50)
dropout_3 | input: | (None, 50)
Dropout | output: | (None, 50)
batch_normalization_3 | input: | (None, 50)
BatchNormalization | output: | (None, 50)
dense_4 | input: | (None, 50)
Dense | output: | (None, 25)
dropout_4 | input: | (None, 25)
Dropout | output: | (None, 25)
batch_normalization_4 | input: | (None, 25)
BatchNormalization | output: | (None, 25)
dense_5 | input: | (None, 25)
Dense | output: | (None, 2)

```
[ ]: optimizer = keras.optimizers.Adam(learning_rate=1e-3)
     model.compile(loss="sparse_categorical_crossentropy",
                   optimizer=optimizer , metrics=["accuracy"])
```

```
[ ]: early_stopping_cb = tf.keras.callbacks.
      ↪EarlyStopping(monitor="val_loss",patience=20)
     out = model.fit(x_train, y_train, epochs=60, validation_data=(x_valid,␣
      ↪y_valid),callbacks=[early_stopping_cb])
```

```
Epoch 1/60
720/720 [==============================] - 5s 4ms/step - loss: 0.7435 -
accuracy: 0.5016 - val_loss: 0.6972 - val_accuracy: 0.5073
Epoch 2/60
720/720 [==============================] - 3s 4ms/step - loss: 0.6981 -
accuracy: 0.5003 - val_loss: 0.6946 - val_accuracy: 0.4990
Epoch 3/60
720/720 [==============================] - 3s 4ms/step - loss: 0.6941 -
accuracy: 0.5049 - val_loss: 0.6938 - val_accuracy: 0.5056
Epoch 4/60
720/720 [==============================] - 3s 3ms/step - loss: 0.6938 -
accuracy: 0.5056 - val_loss: 0.6949 - val_accuracy: 0.4984
Epoch 5/60
720/720 [==============================] - 3s 4ms/step - loss: 0.6936 -
accuracy: 0.5091 - val_loss: 0.6939 - val_accuracy: 0.5009
Epoch 6/60
720/720 [==============================] - 3s 4ms/step - loss: 0.6938 -
accuracy: 0.5082 - val_loss: 0.6931 - val_accuracy: 0.5095
Epoch 7/60
720/720 [==============================] - 3s 4ms/step - loss: 0.6933 -
accuracy: 0.5124 - val_loss: 0.6944 - val_accuracy: 0.5054
Epoch 8/60
720/720 [==============================] - 3s 4ms/step - loss: 0.6937 -
accuracy: 0.5080 - val_loss: 0.6937 - val_accuracy: 0.5075
Epoch 9/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6929 -
accuracy: 0.5146 - val_loss: 0.6932 - val_accuracy: 0.5089
Epoch 10/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6930 -
accuracy: 0.5160 - val_loss: 0.6945 - val_accuracy: 0.5085
Epoch 11/60
720/720 [==============================] - 3s 3ms/step - loss: 0.6926 -
accuracy: 0.5158 - val_loss: 0.6947 - val_accuracy: 0.5054
Epoch 12/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6921 -
accuracy: 0.5202 - val_loss: 0.6954 - val_accuracy: 0.5071
```

```
Epoch 13/60
720/720 [==============================] - 3s 3ms/step - loss: 0.6919 -
accuracy: 0.5248 - val_loss: 0.6970 - val_accuracy: 0.5115
Epoch 14/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6916 -
accuracy: 0.5258 - val_loss: 0.6948 - val_accuracy: 0.5094
Epoch 15/60
720/720 [==============================] - 3s 4ms/step - loss: 0.6913 -
accuracy: 0.5250 - val_loss: 0.6947 - val_accuracy: 0.5033
Epoch 16/60
720/720 [==============================] - 3s 4ms/step - loss: 0.6899 -
accuracy: 0.5353 - val_loss: 0.6943 - val_accuracy: 0.5075
Epoch 17/60
720/720 [==============================] - 3s 4ms/step - loss: 0.6893 -
accuracy: 0.5377 - val_loss: 0.6950 - val_accuracy: 0.5125
Epoch 18/60
720/720 [==============================] - 3s 4ms/step - loss: 0.6886 -
accuracy: 0.5408 - val_loss: 0.6969 - val_accuracy: 0.5122
Epoch 19/60
720/720 [==============================] - 6s 8ms/step - loss: 0.6873 -
accuracy: 0.5433 - val_loss: 0.6969 - val_accuracy: 0.5153
Epoch 20/60
720/720 [==============================] - 5s 7ms/step - loss: 0.6872 -
accuracy: 0.5434 - val_loss: 0.6963 - val_accuracy: 0.5078
Epoch 21/60
720/720 [==============================] - 4s 5ms/step - loss: 0.6860 -
accuracy: 0.5515 - val_loss: 0.7017 - val_accuracy: 0.5017
Epoch 22/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6849 -
accuracy: 0.5538 - val_loss: 0.6988 - val_accuracy: 0.5094
Epoch 23/60
720/720 [==============================] - 3s 5ms/step - loss: 0.6828 -
accuracy: 0.5606 - val_loss: 0.7044 - val_accuracy: 0.5057
Epoch 24/60
720/720 [==============================] - 3s 4ms/step - loss: 0.6827 -
accuracy: 0.5592 - val_loss: 0.6983 - val_accuracy: 0.5069
Epoch 25/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6802 -
accuracy: 0.5678 - val_loss: 0.7044 - val_accuracy: 0.5038
Epoch 26/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6799 -
accuracy: 0.5678 - val_loss: 0.6987 - val_accuracy: 0.5148
```

[ ]:
```
out.params
```

[ ]:
```
{'verbose': 1, 'epochs': 60, 'steps': 720}
```

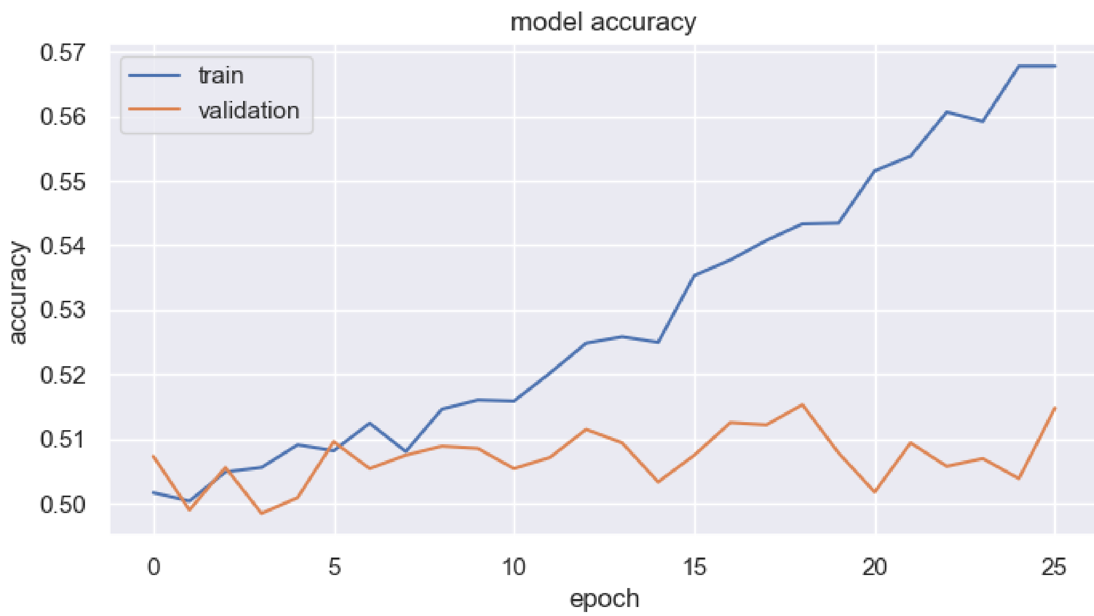
[ ]:
```
t_loss,t_accuracy=model.evaluate(test, 12)
```

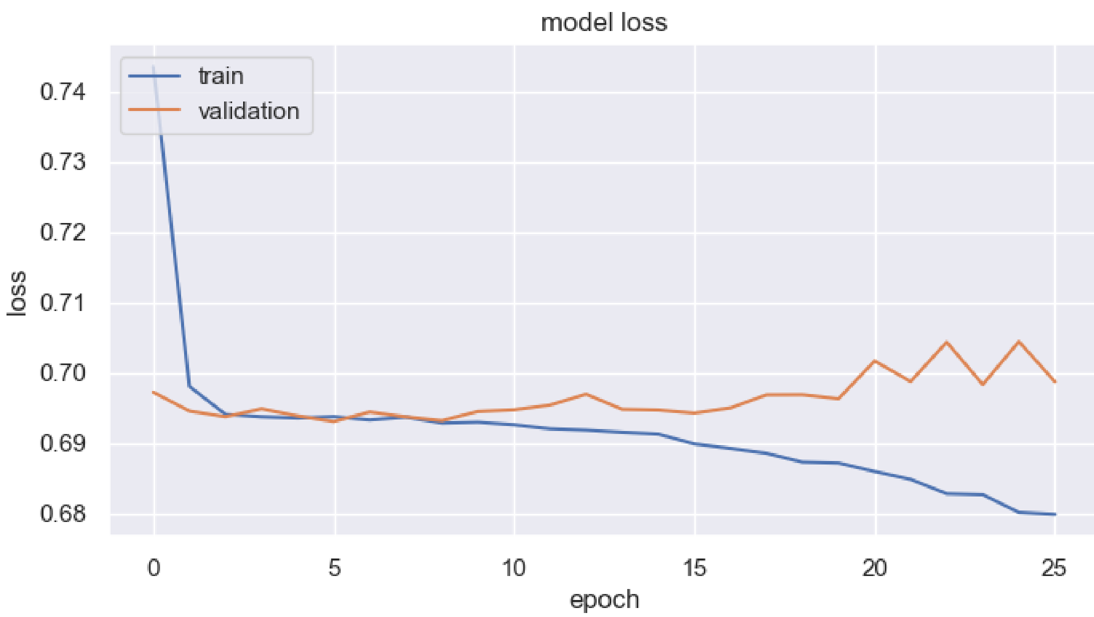
```
  1/225 […] - ETA: 3s - loss: 0.7056 - accuracy:
0.5312225/225 [==============================] - 0s 1ms/step - loss: 0.6979 -
accuracy: 0.5168
```

[ ]:
```
model.evaluate(test,l2)
X = test[:20]
y_proba = model.predict(X)
y_proba.round(2)
y_pred = y_proba.argmax(axis=-1)
print("predictions: "+str(y_pred))
x=l2[:20].tolist()
print("labels: "+str(x))
```

```
225/225 [==============================] - 0s 1ms/step - loss: 0.6979 -
accuracy: 0.5168
1/1 [==============================] - 0s 173ms/step
predictions: [1 0 1 1 1 1 1 1 0 1 1 0 0 1 0 1 1 0 0 1]
labels: [1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0]
```

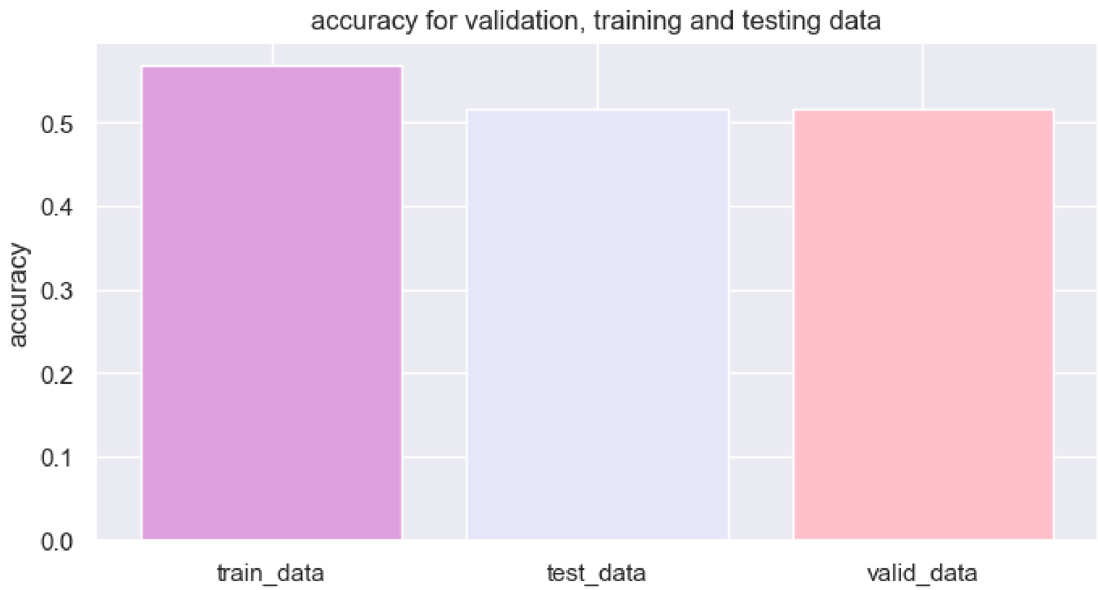
[ ]:
```
#  "Accuracy"
plt.plot(out.history['accuracy'])
plt.plot(out.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
# "Loss"
plt.plot(out.history['loss'])
plt.plot(out.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```





[ ]:
```
list1=["train_data",'test_data','valid_data']
list2=[max(out.history['accuracy']),t_accuracy,max(out.history['val_accuracy'])]
plt.bar(list1,list2,color=['plum','lavender','pink'])
plt.title('accuracy for validation, training and testing data')
plt.ylabel('accuracy')
```

[ ]: Text(0, 0.5, 'accuracy')



1.4.3 *seqModel2 :*

[ ]:
```
tf.random.set_seed(42)
model = tf.keras.Sequential()
model.add(layers.InputLayer(input_shape=(16)))

model.add(layers.Dense(400,kernel_initializer="he_normal"))
model.add(layers.BatchNormalization())
model.add(layers.Activation("elu"))
model.add(layers.Dropout(0.15))

model.add(layers.Dense(200,kernel_initializer="he_normal"))
model.add(layers.BatchNormalization())
model.add(layers.Activation("elu"))
model.add(layers.Dropout(0.15))

model.add(layers.Dense(100,kernel_initializer="he_normal"))
model.add(layers.BatchNormalization())
model.add(layers.Activation("elu"))
model.add(layers.Dropout(0.15))

model.add(layers.Dense(50,kernel_initializer="he_normal"))
model.add(layers.BatchNormalization())
model.add(layers.Activation("elu"))
model.add(layers.Dropout(0.15))
```

```
model.add(layers.Dense(25,kernel_initializer="he_normal"))
model.add(layers.BatchNormalization())
model.add(layers.Activation("elu"))
model.add(layers.Dropout(0.15))

model.add(layers.Dense(2,kernel_initializer="he_normal",activation="softmax"))
```

[ ]:
```
model.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 400)               6800

 batch_normalization_5 (Bat  (None, 400)               1600
 chNormalization)

 activation (Activation)     (None, 400)               0

 dropout_5 (Dropout)         (None, 400)               0

 dense_7 (Dense)             (None, 200)               80200

 batch_normalization_6 (Bat  (None, 200)               800
 chNormalization)

 activation_1 (Activation)   (None, 200)               0

```

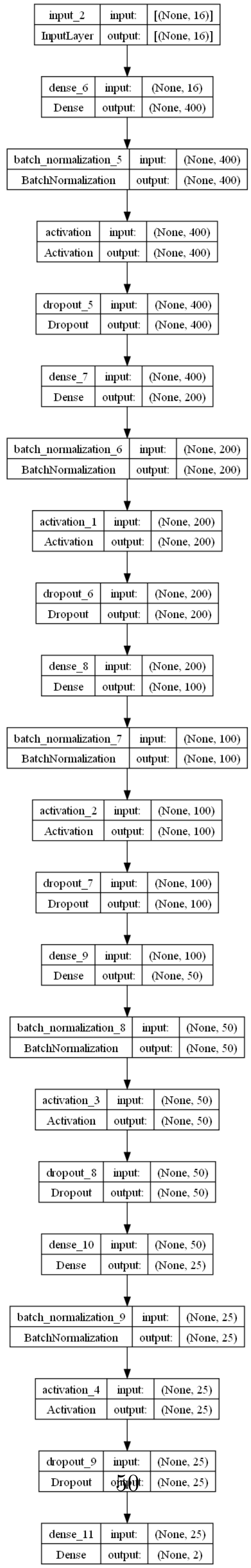

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 400)               6800

 batch_normalization_5 (Bat  (None, 400)               1600
 chNormalization)

 activation (Activation)     (None, 400)               0

 dropout_5 (Dropout)         (None, 400)               0

 dense_7 (Dense)             (None, 200)               80200

 batch_normalization_6 (Bat  (None, 200)               800
 chNormalization)

 activation_1 (Activation)   (None, 200)               0
```

```
 dropout_6 (Dropout)         (None, 200)               0

 dense_8 (Dense)             (None, 100)               20100

 batch_normalization_7 (Bat  (None, 100)               400
 chNormalization)

 activation_2 (Activation)   (None, 100)               0

 dropout_7 (Dropout)         (None, 100)               0

 dense_9 (Dense)             (None, 50)                5050

 batch_normalization_8 (Bat  (None, 50)                200
 chNormalization)

 activation_3 (Activation)   (None, 50)                0

 dropout_8 (Dropout)         (None, 50)                0

 dense_10 (Dense)            (None, 25)                1275

 batch_normalization_9 (Bat  (None, 25)                100
 chNormalization)

 activation_4 (Activation)   (None, 25)                0

 dropout_9 (Dropout)         (None, 25)                0

 dense_11 (Dense)            (None, 2)                 52

=================================================================
Total params: 116577 (455.38 KB)
Trainable params: 115027 (449.32 KB)
Non-trainable params: 1550 (6.05 KB)
_________________________________________________________________
```

[ ]:
```
optimizer = keras.optimizers.RMSprop(learning_rate=1e-3)
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=optimizer,
              metrics=["accuracy"])
```

[ ]:
```
tf.keras.utils.plot_model(model, "medstu_model1.png", show_shapes=True)
```

[ ]:


input_2 | input: | [(None, 16)]
InputLayer | output: | [(None, 16)]
dense_6 | input: | (None, 16)
Dense | output: | (None, 400)
batch_normalization_5 | input: | (None, 400)
BatchNormalization | output: | (None, 400)
activation | input: | (None, 400)
Activation | output: | (None, 400)
dropout_5 | input: | (None, 400)
Dropout | output: | (None, 400)
dense_7 | input: | (None, 400)
Dense | output: | (None, 200)
batch_normalization_6 | input: | (None, 200)
BatchNormalization | output: | (None, 200)
activation_1 | input: | (None, 200)
Activation | output: | (None, 200)
dropout_6 | input: | (None, 200)
Dropout | output: | (None, 200)
dense_8 | input: | (None, 200)
Dense | output: | (None, 100)
batch_normalization_7 | input: | (None, 100)
BatchNormalization | output: | (None, 100)
activation_2 | input: | (None, 100)
Activation | output: | (None, 100)
dropout_7 | input: | (None, 100)
Dropout | output: | (None, 100)
dense_9 | input: | (None, 100)
Dense | output: | (None, 50)
batch_normalization_8 | input: | (None, 50)
BatchNormalization | output: | (None, 50)
activation_3 | input: | (None, 50)
Activation | output: | (None, 50)
dropout_8 | input: | (None, 50)
Dropout | output: | (None, 50)
dense_10 | input: | (None, 50)
Dense | output: | (None, 25)
batch_normalization_9 | input: | (None, 25)
BatchNormalization | output: | (None, 25)
activation_4 | input: | (None, 25)
Activation | output: | (None, 25)
dropout_9 | input: | (None, 25)
Dropout | output: | (None, 25)
dense_11 | input: | (None, 25)
Dense | output: | (None, 2)

[ ]:
```
early_stopping_cb = tf.keras.callbacks.
    ↪EarlyStopping(monitor="val_loss",patience=20)

out = model.fit(x_train, y_train, epochs=60, validation_data=(x_valid,␣
    ↪y_valid),callbacks=[early_stopping_cb])
```

```
Epoch 1/60
720/720 [==============================] - 4s 3ms/step - loss: 0.7386 -
accuracy: 0.5037 - val_loss: 0.6946 - val_accuracy: 0.5069
Epoch 2/60
720/720 [==============================] - 2s 3ms/step - loss: 0.7053 -
accuracy: 0.4995 - val_loss: 0.6983 - val_accuracy: 0.4877
Epoch 3/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6986 -
accuracy: 0.4994 - val_loss: 0.6934 - val_accuracy: 0.5028
Epoch 4/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6952 -
accuracy: 0.5020 - val_loss: 0.6963 - val_accuracy: 0.4875
Epoch 5/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6942 -
accuracy: 0.5038 - val_loss: 0.6937 - val_accuracy: 0.4972
Epoch 6/60
720/720 [==============================] - 3s 3ms/step - loss: 0.6940 -
accuracy: 0.5046 - val_loss: 0.6933 - val_accuracy: 0.4970
Epoch 7/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6939 -
accuracy: 0.5021 - val_loss: 0.6954 - val_accuracy: 0.4938
Epoch 8/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6936 -
accuracy: 0.5057 - val_loss: 0.6931 - val_accuracy: 0.4986
Epoch 9/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6933 -
accuracy: 0.5093 - val_loss: 0.6936 - val_accuracy: 0.5059
Epoch 10/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6933 -
accuracy: 0.5112 - val_loss: 0.6933 - val_accuracy: 0.5071
Epoch 11/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6930 -
accuracy: 0.5104 - val_loss: 0.6953 - val_accuracy: 0.4922
Epoch 12/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6933 -
accuracy: 0.5113 - val_loss: 0.6948 - val_accuracy: 0.4970
Epoch 13/60
720/720 [==============================] - 3s 4ms/step - loss: 0.6932 -
accuracy: 0.5110 - val_loss: 0.6950 - val_accuracy: 0.5054
```

```
Epoch 14/60
720/720 [==============================] - 3s 4ms/step - loss: 0.6929 -
accuracy: 0.5137 - val_loss: 0.6953 - val_accuracy: 0.4918
Epoch 15/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6932 -
accuracy: 0.5076 - val_loss: 0.6936 - val_accuracy: 0.5089
Epoch 16/60
720/720 [==============================] - 3s 4ms/step - loss: 0.6928 -
accuracy: 0.5138 - val_loss: 0.6942 - val_accuracy: 0.5021
Epoch 17/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6927 -
accuracy: 0.5142 - val_loss: 0.6934 - val_accuracy: 0.5094
Epoch 18/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6929 -
accuracy: 0.5163 - val_loss: 0.6966 - val_accuracy: 0.4911
Epoch 19/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6927 -
accuracy: 0.5153 - val_loss: 0.6953 - val_accuracy: 0.4976
Epoch 20/60
720/720 [==============================] - 3s 4ms/step - loss: 0.6927 -
accuracy: 0.5188 - val_loss: 0.6934 - val_accuracy: 0.5082
Epoch 21/60
720/720 [==============================] - 4s 6ms/step - loss: 0.6921 -
accuracy: 0.5233 - val_loss: 0.6957 - val_accuracy: 0.4906
Epoch 22/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6928 -
accuracy: 0.5173 - val_loss: 0.6931 - val_accuracy: 0.5111
Epoch 23/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6921 -
accuracy: 0.5239 - val_loss: 0.6944 - val_accuracy: 0.5085
Epoch 24/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6925 -
accuracy: 0.5208 - val_loss: 0.6938 - val_accuracy: 0.5000
Epoch 25/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6918 -
accuracy: 0.5217 - val_loss: 0.6935 - val_accuracy: 0.5085
Epoch 26/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6920 -
accuracy: 0.5205 - val_loss: 0.6939 - val_accuracy: 0.5049
Epoch 27/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6918 -
accuracy: 0.5193 - val_loss: 0.6945 - val_accuracy: 0.5036
Epoch 28/60
720/720 [==============================] - 2s 3ms/step - loss: 0.6916 -
accuracy: 0.5250 - val_loss: 0.6939 - val_accuracy: 0.5042
```

[ ]:
```
t_val,t_accuracy=model.evaluate(test, l2)
```

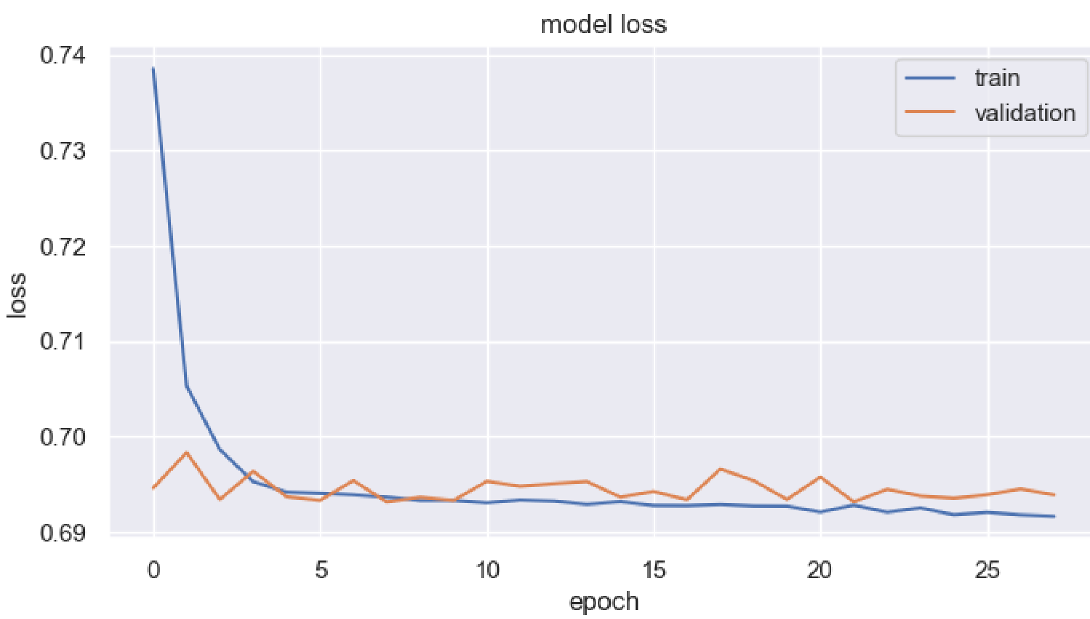
```
  1/225 […] - ETA: 3s - loss: 0.6870 - accuracy:
0.5625225/225 [==============================] - 0s 2ms/step - loss: 0.6955 -
accuracy: 0.4942
```

[ ]:
```
model.evaluate(test,l2)
X = test[:20]
y_proba = model.predict(X)
y_proba.round(2)
y_pred = y_proba.argmax(axis=-1)
print("predictions: "+str(y_pred))
x=l2[:20].tolist()
print("labels: "+str(x))
```

```
225/225 [==============================] - 1s 2ms/step - loss: 0.6955 -
accuracy: 0.4942
1/1 [==============================] - 0s 285ms/step
predictions: [1 0 1 0 0 0 1 1 0 0 1 0 1 1 0 1 1 0 0 0]
labels: [1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0]
```

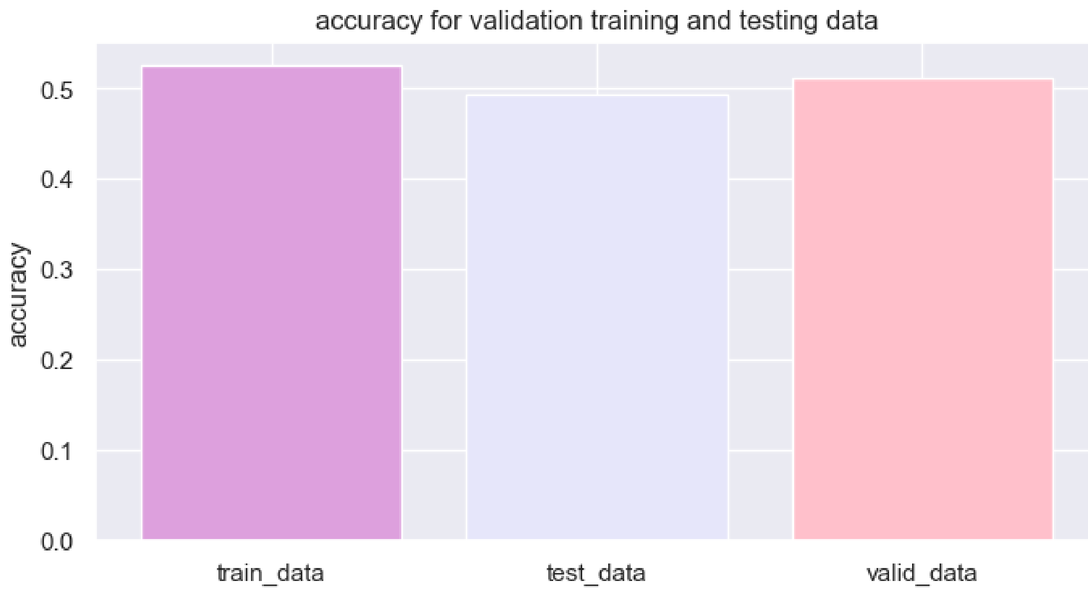
[ ]:
```
print(out.history.keys())
#  "Accuracy"
plt.plot(out.history['accuracy'])
plt.plot(out.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.grid(True)
plt.show()
# "Loss"
plt.plot(out.history['loss'])
plt.plot(out.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper right')
plt.grid(True)
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```




```
[ ]: list1=["train_data",'test_data','valid_data']
     list2=[max(out.history['accuracy']),t_accuracy,max(out.history['val_accuracy'])]
     plt.bar(list1,list2,color=['plum','lavender','pink'])
     plt.title('accuracy for validation training and testing data')
     plt.ylabel('accuracy')
```

```
[ ]: Text(0, 0.5, 'accuracy')
```



1.4.4 *funcModel :*

```
[ ]: L1=layers.Input(shape=[6])
     L2=layers.Input(shape=[12])
     norm_1 = layers.BatchNormalization()(L1)
     hidden1 =layers.Dense(50,␣
      ↪activation="relu",kernel_initializer="he_uniform")(norm_1)
     norm_2 = layers.BatchNormalization()(hidden1)
     hidden2 =layers.Dense(50,␣
      ↪activation="relu",kernel_initializer="he_uniform")(norm_2)
     norm_3 = layers.BatchNormalization()(hidden2)
     norm_4 = layers.BatchNormalization()(L2)
     concat =layers.concatenate([norm_4, norm_3])
     output1 = keras.layers.Dense(2 , name='Moutput')(concat)
     output2 = keras.layers.Dense(2 , name='Aoutput')(norm_3)
     model = keras.models.Model(inputs=[L1,L2], outputs=[output1,output2])
```

```
[ ]: model.summary()
```

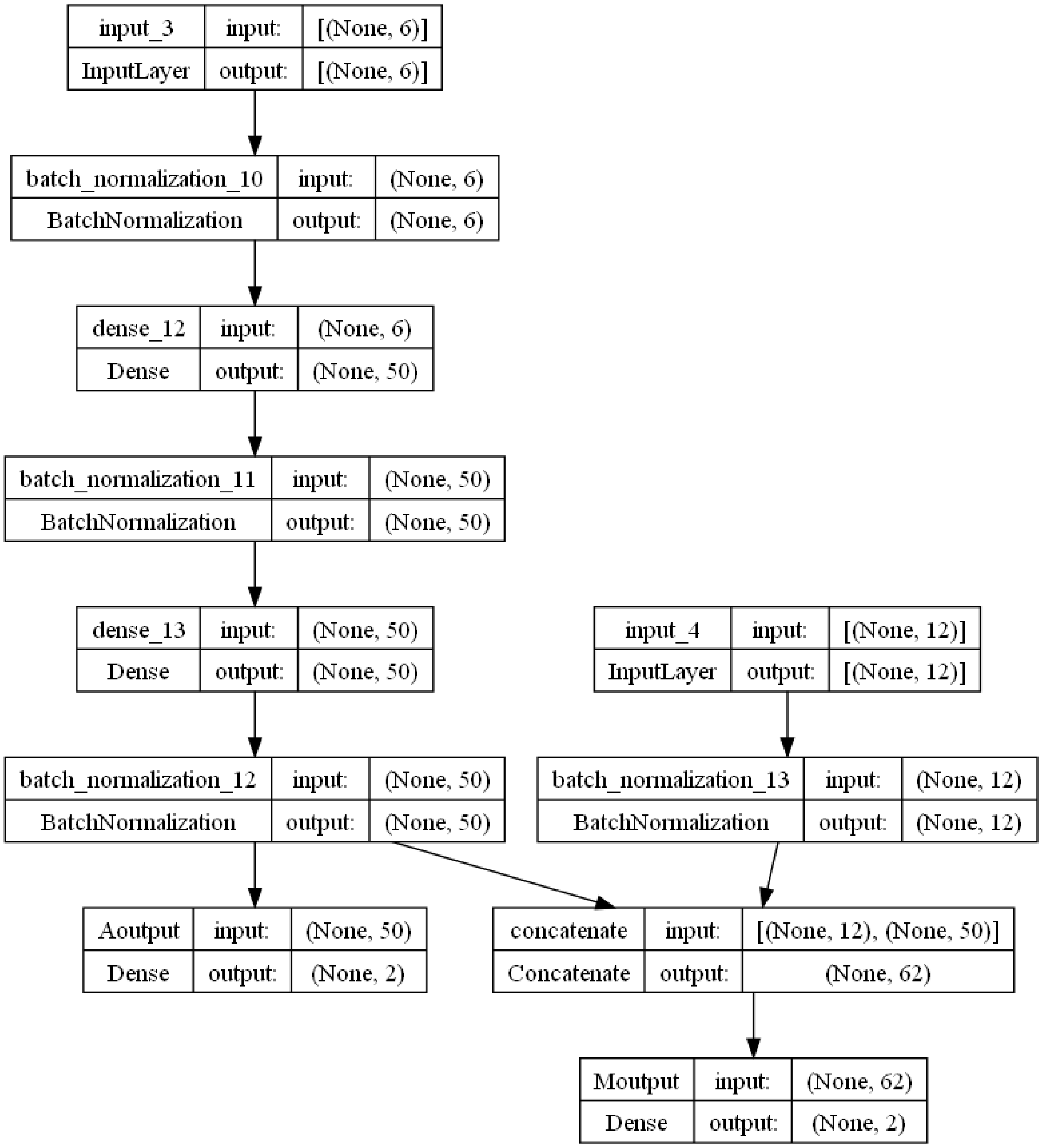
```
Model: "model"
________________________________________________________________________________
__________________
 Layer (type)                    Output Shape          Param #     Connected to
================================================================================
==================
```

```
 input_3 (InputLayer)        [(None, 6)]                  0         []

 batch_normalization_10 (Ba  (None, 6)                    24
['input_3[0][0]']
 tchNormalization)

 dense_12 (Dense)            (None, 50)                   350
['batch_normalization_10[0][0]
                                                                    ']

 batch_normalization_11 (Ba  (None, 50)                   200
['dense_12[0][0]']
 tchNormalization)

________________________________________________________________________________
__________________
 Layer (type)                Output Shape                 Param #   Connected to
================================================================================
==================
 input_3 (InputLayer)        [(None, 6)]                  0         []

 batch_normalization_10 (Ba  (None, 6)                    24
['input_3[0][0]']
 tchNormalization)

 dense_12 (Dense)            (None, 50)                   350
['batch_normalization_10[0][0]
                                                                    ']

 batch_normalization_11 (Ba  (None, 50)                   200
['dense_12[0][0]']
 tchNormalization)

 input_4 (InputLayer)        [(None, 12)]                 0         []

 dense_13 (Dense)            (None, 50)                   2550
['batch_normalization_11[0][0]
                                                                    ']

 batch_normalization_13 (Ba  (None, 12)                   48
['input_4[0][0]']
 tchNormalization)

 batch_normalization_12 (Ba  (None, 50)                   200
['dense_13[0][0]']
 tchNormalization)

 concatenate (Concatenate)   (None, 62)                   0
```

```
['batch_normalization_13[0][0]
                                                                              ',
'batch_normalization_12[0][0]
                                                                              ']

 Moutput (Dense)              (None, 2)                    126
['concatenate[0][0]']

 Aoutput (Dense)              (None, 2)                    102
['batch_normalization_12[0][0]
                                                                              ']

================================================================================
==================
Total params: 3600 (14.06 KB)
Trainable params: 3364 (13.14 KB)
Non-trainable params: 236 (944.00 Byte)
________________________________________________________________________________
__________________
```

[ ]:
```
tf.keras.utils.plot_model(model, "medstu_model2.png", show_shapes=True)
```

[ ]:



[ ]:
```
model.compile(loss=["binary_crossentropy","binary_crossentropy"],␣
↪optimizer=keras.optimizers.SGD(learning_rate=1e-3), metrics=["accuracy"])
```

[ ]:
```
X_train_A, X_train_B = x_train.iloc[:, :6], x_train.iloc[:, 4:]
X_valid_A, X_valid_B = x_valid.iloc[:, :6], x_valid.iloc[:, 4:]
X_test_A, X_test_B = test.iloc[:, :6], test.iloc[:, 4:]
X_new_A, X_new_B = X_test_A.iloc[:6], X_test_B.iloc[:6]
```

[ ]:
```
history = model.fit((X_train_A, X_train_B), y_train,
    epochs=40,validation_data=((X_valid_A, X_valid_B), y_valid))
```

```
Epoch 1/40
720/720 [==============================] - 4s 2ms/step - loss: 1.4019 -
Moutput_loss: 0.7005 - Aoutput_loss: 0.7014 - Moutput_accuracy: 0.4941 -
Aoutput_accuracy: 0.4986 - val_loss: 1.3967 - val_Moutput_loss: 0.6988 -
val_Aoutput_loss: 0.6979 - val_Moutput_accuracy: 0.5059 - val_Aoutput_accuracy:
0.5003
Epoch 2/40
720/720 [==============================] - 1s 2ms/step - loss: 1.4013 -
Moutput_loss: 0.7006 - Aoutput_loss: 0.7008 - Moutput_accuracy: 0.4989 -
Aoutput_accuracy: 0.5005 - val_loss: 1.3997 - val_Moutput_loss: 0.7001 -
val_Aoutput_loss: 0.6995 - val_Moutput_accuracy: 0.5028 - val_Aoutput_accuracy:
0.5007
Epoch 3/40
720/720 [==============================] - 1s 2ms/step - loss: 1.4001 -
Moutput_loss: 0.7001 - Aoutput_loss: 0.7000 - Moutput_accuracy: 0.4980 -
Aoutput_accuracy: 0.5036 - val_loss: 1.3959 - val_Moutput_loss: 0.6982 -
val_Aoutput_loss: 0.6978 - val_Moutput_accuracy: 0.4974 - val_Aoutput_accuracy:
0.5002
Epoch 4/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3990 -
Moutput_loss: 0.6996 - Aoutput_loss: 0.6994 - Moutput_accuracy: 0.5005 -
Aoutput_accuracy: 0.5010 - val_loss: 1.3993 - val_Moutput_loss: 0.6999 -
val_Aoutput_loss: 0.6993 - val_Moutput_accuracy: 0.5049 - val_Aoutput_accuracy:
0.4983
Epoch 5/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3979 -
Moutput_loss: 0.6982 - Aoutput_loss: 0.6997 - Moutput_accuracy: 0.4947 -
Aoutput_accuracy: 0.5003 - val_loss: 1.3954 - val_Moutput_loss: 0.6980 -
val_Aoutput_loss: 0.6974 - val_Moutput_accuracy: 0.4988 - val_Aoutput_accuracy:
0.5012
Epoch 6/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3952 -
Moutput_loss: 0.6977 - Aoutput_loss: 0.6974 - Moutput_accuracy: 0.5015 -
Aoutput_accuracy: 0.5000 - val_loss: 1.3958 - val_Moutput_loss: 0.6981 -
val_Aoutput_loss: 0.6976 - val_Moutput_accuracy: 0.5016 - val_Aoutput_accuracy:
0.5014
Epoch 7/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3946 -
Moutput_loss: 0.6968 - Aoutput_loss: 0.6978 - Moutput_accuracy: 0.4971 -
Aoutput_accuracy: 0.5031 - val_loss: 1.3971 - val_Moutput_loss: 0.6990 -
val_Aoutput_loss: 0.6981 - val_Moutput_accuracy: 0.4991 - val_Aoutput_accuracy:
0.5003
Epoch 8/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3959 -
Moutput_loss: 0.6976 - Aoutput_loss: 0.6983 - Moutput_accuracy: 0.4975 -
```

```
Aoutput_accuracy: 0.5010 - val_loss: 1.3923 - val_Moutput_loss: 0.6963 -
val_Aoutput_loss: 0.6960 - val_Moutput_accuracy: 0.5016 - val_Aoutput_accuracy:
0.5023
Epoch 9/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3956 -
Moutput_loss: 0.6972 - Aoutput_loss: 0.6984 - Moutput_accuracy: 0.5011 -
Aoutput_accuracy: 0.4966 - val_loss: 1.3941 - val_Moutput_loss: 0.6974 -
val_Aoutput_loss: 0.6968 - val_Moutput_accuracy: 0.5019 - val_Aoutput_accuracy:
0.5026
Epoch 10/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3941 -
Moutput_loss: 0.6966 - Aoutput_loss: 0.6974 - Moutput_accuracy: 0.4945 -
Aoutput_accuracy: 0.5005 - val_loss: 1.3920 - val_Moutput_loss: 0.6963 -
val_Aoutput_loss: 0.6957 - val_Moutput_accuracy: 0.5009 - val_Aoutput_accuracy:
0.5030
Epoch 11/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3940 -
Moutput_loss: 0.6968 - Aoutput_loss: 0.6972 - Moutput_accuracy: 0.5004 -
Aoutput_accuracy: 0.5030 - val_loss: 1.3948 - val_Moutput_loss: 0.6976 -
val_Aoutput_loss: 0.6972 - val_Moutput_accuracy: 0.5023 - val_Aoutput_accuracy:
0.5040
Epoch 12/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3927 -
Moutput_loss: 0.6965 - Aoutput_loss: 0.6963 - Moutput_accuracy: 0.4941 -
Aoutput_accuracy: 0.5044 - val_loss: 1.3943 - val_Moutput_loss: 0.6974 -
val_Aoutput_loss: 0.6970 - val_Moutput_accuracy: 0.5028 - val_Aoutput_accuracy:
0.5017
Epoch 13/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3927 -
Moutput_loss: 0.6964 - Aoutput_loss: 0.6963 - Moutput_accuracy: 0.4964 -
Aoutput_accuracy: 0.5024 - val_loss: 1.3937 - val_Moutput_loss: 0.6969 -
val_Aoutput_loss: 0.6968 - val_Moutput_accuracy: 0.4993 - val_Aoutput_accuracy:
0.4991
Epoch 14/40
720/720 [==============================] - 2s 2ms/step - loss: 1.3920 -
Moutput_loss: 0.6959 - Aoutput_loss: 0.6961 - Moutput_accuracy: 0.4992 -
Aoutput_accuracy: 0.5024 - val_loss: 1.3927 - val_Moutput_loss: 0.6966 -
val_Aoutput_loss: 0.6962 - val_Moutput_accuracy: 0.5031 - val_Aoutput_accuracy:
0.5035
Epoch 15/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3915 -
Moutput_loss: 0.6960 - Aoutput_loss: 0.6955 - Moutput_accuracy: 0.4983 -
Aoutput_accuracy: 0.5002 - val_loss: 1.3915 - val_Moutput_loss: 0.6961 -
val_Aoutput_loss: 0.6954 - val_Moutput_accuracy: 0.5036 - val_Aoutput_accuracy:
0.4986
Epoch 16/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3912 -
Moutput_loss: 0.6953 - Aoutput_loss: 0.6959 - Moutput_accuracy: 0.4891 -
```

```
Aoutput_accuracy: 0.5052 - val_loss: 1.3932 - val_Moutput_loss: 0.6968 -
val_Aoutput_loss: 0.6964 - val_Moutput_accuracy: 0.5028 - val_Aoutput_accuracy:
0.5000
Epoch 17/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3910 -
Moutput_loss: 0.6956 - Aoutput_loss: 0.6954 - Moutput_accuracy: 0.4977 -
Aoutput_accuracy: 0.5045 - val_loss: 1.3906 - val_Moutput_loss: 0.6956 -
val_Aoutput_loss: 0.6950 - val_Moutput_accuracy: 0.5031 - val_Aoutput_accuracy:
0.5028
Epoch 18/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3918 -
Moutput_loss: 0.6960 - Aoutput_loss: 0.6959 - Moutput_accuracy: 0.4964 -
Aoutput_accuracy: 0.5018 - val_loss: 1.3924 - val_Moutput_loss: 0.6965 -
val_Aoutput_loss: 0.6959 - val_Moutput_accuracy: 0.5040 - val_Aoutput_accuracy:
0.5017
Epoch 19/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3909 -
Moutput_loss: 0.6953 - Aoutput_loss: 0.6956 - Moutput_accuracy: 0.4951 -
Aoutput_accuracy: 0.5030 - val_loss: 1.3930 - val_Moutput_loss: 0.6966 -
val_Aoutput_loss: 0.6963 - val_Moutput_accuracy: 0.5007 - val_Aoutput_accuracy:
0.5043
Epoch 20/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3916 -
Moutput_loss: 0.6957 - Aoutput_loss: 0.6960 - Moutput_accuracy: 0.4955 -
Aoutput_accuracy: 0.5004 - val_loss: 1.3909 - val_Moutput_loss: 0.6957 -
val_Aoutput_loss: 0.6953 - val_Moutput_accuracy: 0.4983 - val_Aoutput_accuracy:
0.5042
Epoch 21/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3904 -
Moutput_loss: 0.6950 - Aoutput_loss: 0.6954 - Moutput_accuracy: 0.5007 -
Aoutput_accuracy: 0.5047 - val_loss: 1.3952 - val_Moutput_loss: 0.6979 -
val_Aoutput_loss: 0.6973 - val_Moutput_accuracy: 0.4990 - val_Aoutput_accuracy:
0.4995
Epoch 22/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3892 -
Moutput_loss: 0.6947 - Aoutput_loss: 0.6945 - Moutput_accuracy: 0.4946 -
Aoutput_accuracy: 0.5039 - val_loss: 1.3904 - val_Moutput_loss: 0.6953 -
val_Aoutput_loss: 0.6951 - val_Moutput_accuracy: 0.5009 - val_Aoutput_accuracy:
0.5036
Epoch 23/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3892 -
Moutput_loss: 0.6943 - Aoutput_loss: 0.6948 - Moutput_accuracy: 0.4961 -
Aoutput_accuracy: 0.5056 - val_loss: 1.3927 - val_Moutput_loss: 0.6966 -
val_Aoutput_loss: 0.6961 - val_Moutput_accuracy: 0.5024 - val_Aoutput_accuracy:
0.5017
Epoch 24/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3911 -
Moutput_loss: 0.6953 - Aoutput_loss: 0.6958 - Moutput_accuracy: 0.4998 -
```

```
Aoutput_accuracy: 0.5006 - val_loss: 1.3896 - val_Moutput_loss: 0.6948 -
val_Aoutput_loss: 0.6947 - val_Moutput_accuracy: 0.5014 - val_Aoutput_accuracy:
0.5061
Epoch 25/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3878 -
Moutput_loss: 0.6940 - Aoutput_loss: 0.6938 - Moutput_accuracy: 0.4944 -
Aoutput_accuracy: 0.5079 - val_loss: 1.3908 - val_Moutput_loss: 0.6955 -
val_Aoutput_loss: 0.6953 - val_Moutput_accuracy: 0.5002 - val_Aoutput_accuracy:
0.5071
Epoch 26/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3894 -
Moutput_loss: 0.6945 - Aoutput_loss: 0.6949 - Moutput_accuracy: 0.5015 -
Aoutput_accuracy: 0.5050 - val_loss: 1.3909 - val_Moutput_loss: 0.6955 -
val_Aoutput_loss: 0.6953 - val_Moutput_accuracy: 0.5049 - val_Aoutput_accuracy:
0.5066
Epoch 27/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3891 -
Moutput_loss: 0.6945 - Aoutput_loss: 0.6946 - Moutput_accuracy: 0.4941 -
Aoutput_accuracy: 0.5032 - val_loss: 1.3913 - val_Moutput_loss: 0.6957 -
val_Aoutput_loss: 0.6956 - val_Moutput_accuracy: 0.5035 - val_Aoutput_accuracy:
0.5075
Epoch 28/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3864 -
Moutput_loss: 0.6932 - Aoutput_loss: 0.6932 - Moutput_accuracy: 0.4928 -
Aoutput_accuracy: 0.5077 - val_loss: 1.3902 - val_Moutput_loss: 0.6954 -
val_Aoutput_loss: 0.6948 - val_Moutput_accuracy: 0.5012 - val_Aoutput_accuracy:
0.5031
Epoch 29/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3891 -
Moutput_loss: 0.6945 - Aoutput_loss: 0.6947 - Moutput_accuracy: 0.4974 -
Aoutput_accuracy: 0.5001 - val_loss: 1.3909 - val_Moutput_loss: 0.6957 -
val_Aoutput_loss: 0.6953 - val_Moutput_accuracy: 0.5043 - val_Aoutput_accuracy:
0.5061
Epoch 30/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3890 -
Moutput_loss: 0.6945 - Aoutput_loss: 0.6945 - Moutput_accuracy: 0.4978 -
Aoutput_accuracy: 0.5073 - val_loss: 1.3911 - val_Moutput_loss: 0.6957 -
val_Aoutput_loss: 0.6954 - val_Moutput_accuracy: 0.5035 - val_Aoutput_accuracy:
0.5054
Epoch 31/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3881 -
Moutput_loss: 0.6938 - Aoutput_loss: 0.6943 - Moutput_accuracy: 0.4957 -
Aoutput_accuracy: 0.5035 - val_loss: 1.3903 - val_Moutput_loss: 0.6953 -
val_Aoutput_loss: 0.6950 - val_Moutput_accuracy: 0.5002 - val_Aoutput_accuracy:
0.5080
Epoch 32/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3883 -
Moutput_loss: 0.6941 - Aoutput_loss: 0.6942 - Moutput_accuracy: 0.4898 -
```

```
Aoutput_accuracy: 0.5074 - val_loss: 1.3901 - val_Moutput_loss: 0.6952 -
val_Aoutput_loss: 0.6949 - val_Moutput_accuracy: 0.5009 - val_Aoutput_accuracy:
0.5069
Epoch 33/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3881 -
Moutput_loss: 0.6941 - Aoutput_loss: 0.6940 - Moutput_accuracy: 0.4939 -
Aoutput_accuracy: 0.5064 - val_loss: 1.3909 - val_Moutput_loss: 0.6958 -
val_Aoutput_loss: 0.6951 - val_Moutput_accuracy: 0.4995 - val_Aoutput_accuracy:
0.5061
Epoch 34/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3882 -
Moutput_loss: 0.6943 - Aoutput_loss: 0.6939 - Moutput_accuracy: 0.4979 -
Aoutput_accuracy: 0.5026 - val_loss: 1.3918 - val_Moutput_loss: 0.6961 -
val_Aoutput_loss: 0.6957 - val_Moutput_accuracy: 0.5000 - val_Aoutput_accuracy:
0.5111
Epoch 35/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3876 -
Moutput_loss: 0.6937 - Aoutput_loss: 0.6939 - Moutput_accuracy: 0.4956 -
Aoutput_accuracy: 0.5013 - val_loss: 1.3895 - val_Moutput_loss: 0.6949 -
val_Aoutput_loss: 0.6946 - val_Moutput_accuracy: 0.5009 - val_Aoutput_accuracy:
0.5080
Epoch 36/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3873 -
Moutput_loss: 0.6936 - Aoutput_loss: 0.6937 - Moutput_accuracy: 0.4966 -
Aoutput_accuracy: 0.5055 - val_loss: 1.3922 - val_Moutput_loss: 0.6962 -
val_Aoutput_loss: 0.6960 - val_Moutput_accuracy: 0.4991 - val_Aoutput_accuracy:
0.5080
Epoch 37/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3879 -
Moutput_loss: 0.6938 - Aoutput_loss: 0.6941 - Moutput_accuracy: 0.4947 -
Aoutput_accuracy: 0.5022 - val_loss: 1.3897 - val_Moutput_loss: 0.6951 -
val_Aoutput_loss: 0.6946 - val_Moutput_accuracy: 0.5010 - val_Aoutput_accuracy:
0.5134
Epoch 38/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3870 -
Moutput_loss: 0.6936 - Aoutput_loss: 0.6934 - Moutput_accuracy: 0.4912 -
Aoutput_accuracy: 0.5115 - val_loss: 1.3905 - val_Moutput_loss: 0.6953 -
val_Aoutput_loss: 0.6951 - val_Moutput_accuracy: 0.4995 - val_Aoutput_accuracy:
0.5090
Epoch 39/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3870 -
Moutput_loss: 0.6934 - Aoutput_loss: 0.6936 - Moutput_accuracy: 0.4909 -
Aoutput_accuracy: 0.5041 - val_loss: 1.3923 - val_Moutput_loss: 0.6964 -
val_Aoutput_loss: 0.6960 - val_Moutput_accuracy: 0.5000 - val_Aoutput_accuracy:
0.5109
Epoch 40/40
720/720 [==============================] - 1s 2ms/step - loss: 1.3863 -
Moutput_loss: 0.6932 - Aoutput_loss: 0.6931 - Moutput_accuracy: 0.4953 -
```

```
Aoutput_accuracy: 0.5068 - val_loss: 1.3910 - val_Moutput_loss: 0.6956 -
val_Aoutput_loss: 0.6954 - val_Moutput_accuracy: 0.5024 - val_Aoutput_accuracy:
0.5073
```

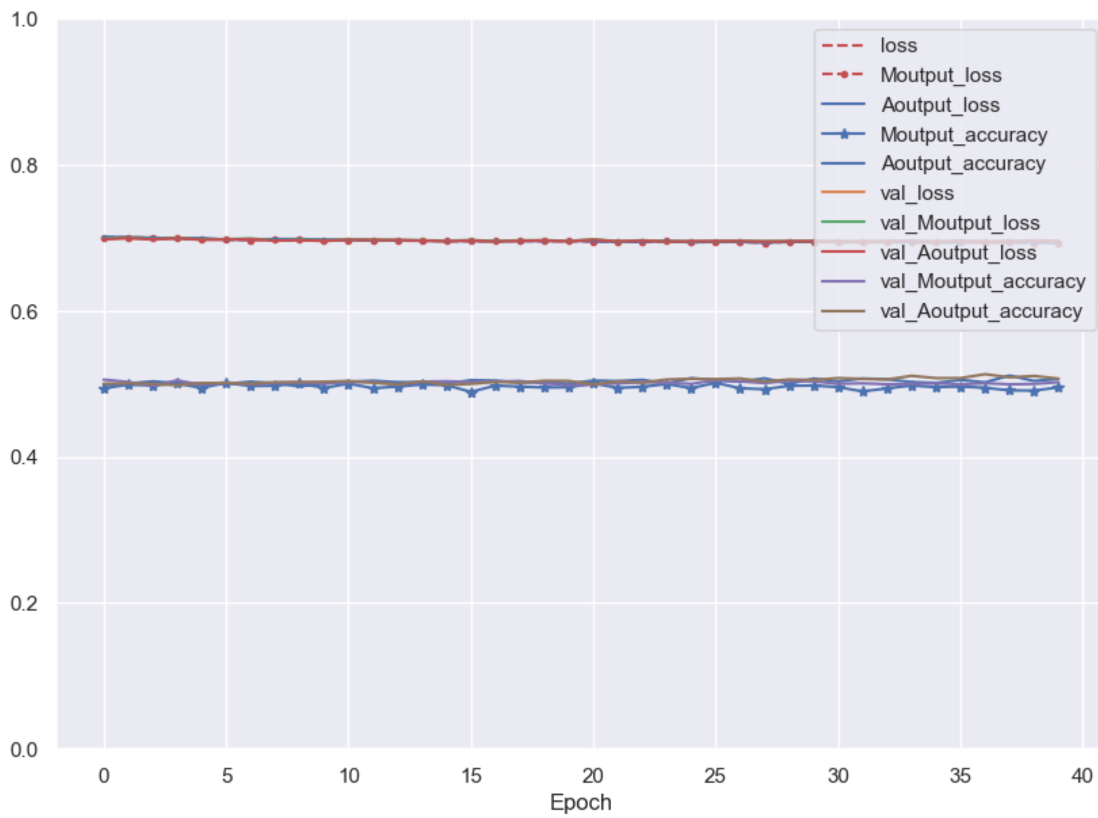
[ ]:
```
bce_test = model.evaluate((X_test_A, X_test_B), l2)
```

```
225/225 [==============================] - 0s 1ms/step - loss: 1.3878 -
Moutput_loss: 0.6941 - Aoutput_loss: 0.6937 - Moutput_accuracy: 0.4988 -
Aoutput_accuracy: 0.4943
```

[ ]:
```
y_pred = model.predict((X_new_A, X_new_B))
```

```
1/1 [==============================] - 0s 107ms/step
1/1 [==============================] - 0s 107ms/step
```

[ ]:
```
pd.DataFrame(history.history).plot(
    figsize=(10, 7), ylim=[0,1], grid=True, xlabel="Epoch",
    style=["r--", "r--.", "b-", "b-*"])
plt.legend(loc="upper right")
plt.show()
```



[ ]: