2.) Build a neural net classifier predicting survival. The authors did not use a Neural network.

3.) Tune your neural network to optimum performance

4.) Use SHAP and/or ELI5 to determine what variables are most important

5.) For your model, compute all the statistics shown in figure 5 of the paper.

6.) Discuss how well your model performed relative to the models in the paper.

In [7]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [9]:
```
!pip install openpyxl
```

```
Collecting openpyxl
  Downloading openpyxl-3.1.2-py2.py3-none-any.whl (249 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 250.0/250.0 kB 2.5 MB/s eta 0:00:00
Collecting et-xmlfile (from openpyxl)
  Downloading et_xmlfile-1.1.0-py3-none-any.whl (4.7 kB)
Installing collected packages: et-xmlfile, openpyxl
Successfully installed et-xmlfile-1.1.0 openpyxl-3.1.2
```

In [10]:
```
infile1= "/content/Norwegian.xlsx"
survivalTraining= pd.read_excel(infile1)
```

In [11]:
```
survivalTraining.head()
```

Out[11]:

	Age (years)	Sex (0 male, 1 female)	Length of stay (days)	Hospital outcome (0 alive, 1 dead)	Episode number	ICD-10 1	ICD-10 2	ICD-10 3	ICD-10 4	ICD-10 5	ICD-10 6	ICD-10 7	ICD-10 8	ICD 10
0	21	1	1	0	1		J159							
1	20	1	5	0	1	J158								
2	21	1	3	0	1	J157	E86							
3	77	0	18	0	1		J159							
4	72	0	9	0	1	A415	I509	A418	K810					

"Since the main goal of this study is to predict the survival of the patient, we discarded the length of stay because it strongly relates to the likelihood to survive: the longer the patient has to stay in the hospital, the less likely she/he will survive.

I will do the above after importing the validation data.

In [12]:
```
infile1= "/content/KoreanPatients.xlsx"
survivalValidation= pd.read_excel(infile1)
```

In [13]:
```
survivalValidation.head()
```

Out[13]:

	Number	sex	age	hospital stay	mortality	APACHE II	ASA	Preop shock	postop shock	HTN	...	Albumin-1	Album
0	1	1	18	8	0	NaN	1	0	0	0	...	4.4	
1	2	1	20	27	0	12.0	1	0	0	0	...	3.7	
2	3	1	20	47	0	NaN	1	1	0	0	...	3.1	
3	4	1	21	25	0	14.0	1	0	0	0	...	3.2	
4	5	1	22	14	0	NaN	3	0	0	0	...	4.3	

5 rows × 56 columns

In [14]:
```
survivalValidation.columns
```

Out[14]:
```
Index(['Number', 'sex', 'age', 'hospital stay', 'mortality', 'APACHE II',
       'ASA', 'Preop shock', 'postop shock', 'HTN', 'DM', 'CRF', 'Pul TBC',
       'malignancy ', 'Diagnosis', 'Location of lesion ', 'surgery type',
       'anastomotic leak', 'infectious', 'wound cx', 'new sepsis', 'pulmonary',
       'AKI', 'ileus', 'blood culture', 'peritoneal fluid culture', 'CRP-1',
       'CRP-0', 'CRP+1', 'CRP+3', 'CRP+7', 'Hb-1', 'Hb-0', 'Hb+1', 'Hb+3',
       'Hb+7', 'Platelet-1', 'Platelet-0', 'Platelet+1', 'Platelet+3',
       'Platelet+7', 'Cholesterol-1', 'Cholesterol-0', 'Cholesterol+1',
       'choleterol+3', 'cholesterol+7', 'Albumin-1', 'Albumin-0', 'Albumin+1',
       'Albumin+3', 'Albumin+7', 'T-bilirubin-1', 'T-bilirubin-0',
       'T-bilirubin+1', 'T-bilirubin+3', 'T-bilirubin+7'],
      dtype='object')
```

In [15]:
```
survivalTraining.columns
```

Out[15]:
```
Index(['Age (years)', 'Sex (0 male, 1 female)', 'Length of stay (days)',
       'Hospital outcome (0 alive, 1 dead)', 'Episode number', 'ICD-10 1',
       'ICD-10 2', 'ICD-10 3', 'ICD-10 4', 'ICD-10 5', 'ICD-10 6', 'ICD-10 7',
       'ICD-10 8', 'ICD-10 9'],
      dtype='object')
```

In [16]:
```
survivalTraining['Age (years)']
```

Out[16]:
```
0         21
1         20
2         21
3         77
4         72
          ..
110199     0
110200     0
110201    70
110202     0
110203     0
Name: Age (years), Length: 110204, dtype: int64
```

In [17]:
```
survivalTraining.head()
```

Out[17]:

	Age (years)	Sex (0 male, 1 female)	Length of stay (days)	Hospital outcome (0 alive, 1 dead)	Episode number	ICD-10 1	ICD-10 2	ICD-10 3	ICD-10 4	ICD-10 5	ICD-10 6	ICD-10 7	ICD-10 8	ICD 10
0	21	1	1	0	1		J159							
1	20	1	5	0	1	J158								
2	21	1	3	0	1	J157	E86							
3	77	0	18	0	1		J159							
4	72	0	9	0	1	A415	I509	A418	K810					

I am just going to be working with the Norwegian data because the table I am trying to calculate only uses data from the norwegian data.

In [18]:
```
survivalTraining['Episode number'].unique()
```

Out[18]:
```
array([1, 2, 3, 4, 5])
```

In [19]:
```
survivalTraining['Sex (0 male, 1 female)'].unique()
```

Out[19]:
```
array([1, 0])
```

In [20]:
```
survivalTraining['Age (years)'].unique()
```

Out[20]:
```
array([ 21,  20,  77,  72,  83,  74,  69,  53,  82,  75,  45,  56,  46,
        48,  40,  39,  70,  47,  27,  11,  91,   7,  79,  84,  16,  73,
        17,  18,  63,  88,  89,  76,  41,  66,  80,  62,  59,  55,  68,
        33,  71,   8,  58,  78,  51,  43,  44,  60,  86,  61,  67,  57,
        81,  49,  64,  25,  65,  42,  36,  38,  85,  24,  19,  37,  35,
         6,  50,  87,  54,  29,  12,  10,  23,  52,   9,  15,  31,  92,
        28,  30,  13,  94,  90,  26,  32,  95,   5,  93,  34,  96,  22,
        97,  98, 100,  14,   4,  99,   3,   2,   1,   0])
```

In [21]:
```
survivalTraining[['Age (years)','Episode number']]
```

Out[21]:

	Age (years)	Episode number
0	21	1
1	20	1
2	21	1
3	77	1
4	72	1
...	...	...
110199	0	1
110200	0	1
110201	70	1
110202	0	1
110203	0	1

110204 rows × 2 columns

Nothing is missing this is beautiful, I wish it was like this everytime

I am going to use the Sex catagory as is, and stadscale the "Episode number" and "age"

In [22]:
```
standardizeNeeded= survivalTraining[['Age (years)','Episode number']]
```

In [23]:
```
from sklearn.preprocessing import StandardScaler
```

In [24]:
```
scaler = StandardScaler()
patientContinuous = scaler.fit_transform(standardizeNeeded)
```

In [26]:
```
patientContinuous= pd.DataFrame(patientContinuous,columns=standardizeNeeded.columns)
```

In [27]:
```
patientContinuous.head()
```

Out[27]:

	Age (years)	Episode number
0	-1.729837	-0.464727
1	-1.771285	-0.464727
2	-1.729837	-0.464727
3	0.591243	-0.464727
4	0.384004	-0.464727

In [28]:
```
inputVariables = pd.concat([patientContinuous,survivalTraining['Sex (0 male, 1 female)
```

In [29]:
```
inputVariables
```

Out[29]:

	Age (years)	Episode number	Sex (0 male, 1 female)
0	-1.729837	-0.464727	1
1	-1.771285	-0.464727	1
2	-1.729837	-0.464727	1
3	0.591243	-0.464727	0
4	0.384004	-0.464727	0
...	...	...	...
110199	-2.600242	-0.464727	0
110200	-2.600242	-0.464727	1
110201	0.301108	-0.464727	1
110202	-2.600242	-0.464727	0
110203	-2.600242	-0.464727	0

110204 rows × 3 columns

In [30]:
```
y=survivalTraining['Hospital outcome (0 alive, 1 dead)']
```

In [30]:
```
```

The study cohort for the paper was about 20%

In [31]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(inputVariables, y,train_size=0.8,r
```

Well that may not be what we want as we want to cosider the entire dataset for traning and for testing we need to meet the requrement: "Accordingly, the final study cohort consisted of cases fulfilling one or several infection or sepsis related ICD-10 codes as well as one or several codes for acute organ dysfunction (Fig 1)."

https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0187990#sec005

In [32]:
```
survivalTraining.head()
```

Out[32]:

	Age (years)	Sex (0 male, 1 female)	Length of stay (days)	Hospital outcome (0 alive, 1 dead)	Episode number	ICD-10 1	ICD-10 2	ICD-10 3	ICD-10 4	ICD-10 5	ICD-10 6	ICD-10 7	ICD-10 8	ICD 10
0	21	1	1	0	1		J159							
1	20	1	5	0	1	J158								
2	21	1	3	0	1	J157	E86							
3	77	0	18	0	1		J159							
4	72	0	9	0	1	A415	I509	A418	K810					

In [33]:
```
specialRows = survivalTraining[['ICD-10 1','ICD-10 2','ICD-10 3','ICD-10 4','ICD-10 5'
```

In [34]:
```
specialRows.any().unique()
```

Out[34]:
```
array([False])
```

I give up trying to figure out how they came up with a study cohort I don't know what the code for organ failiure is.

Nural Net

In [36]:
```
from sklearn.neural_network import MLPClassifier
```

In [37]:
```
clf = MLPClassifier(solver='adam', alpha=1e-5, random_state=1, batch_size=int(min(200/

clf.fit(inputVariables,y)
y_pred=clf.predict(inputVariables)
```

```
Iteration 1, loss = 0.25543784
Iteration 2, loss = 0.24331635
Iteration 3, loss = 0.24355247
Iteration 4, loss = 0.24318582
Iteration 5, loss = 0.24319076
Iteration 6, loss = 0.24313068
Iteration 7, loss = 0.24311207
Iteration 8, loss = 0.24295093
Iteration 9, loss = 0.24292323
Iteration 10, loss = 0.24290145
Iteration 11, loss = 0.24294107
Iteration 12, loss = 0.24292504
Iteration 13, loss = 0.24274989
Iteration 14, loss = 0.24282288
Iteration 15, loss = 0.24274346
Iteration 16, loss = 0.24279943
Iteration 17, loss = 0.24278841
Iteration 18, loss = 0.24267316
Iteration 19, loss = 0.24268904
Iteration 20, loss = 0.24266828
Iteration 21, loss = 0.24280792
Iteration 22, loss = 0.24267014
Iteration 23, loss = 0.24263976
Iteration 24, loss = 0.24267503
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopp
ing.
```

In [38]:
```
from sklearn.metrics import r2_score
clf.score(inputVariables, y
          )
```

Out[38]:
```
0.9264545751515372
```

In [39]:
```
sum(y_pred)
```

Out[39]:
```
0
```

yay they are all dead, NO!

In [39]:
```

```

In [40]:
```
myTest =  pd.concat([patientContinuous,survivalTraining['Sex (0 male, 1 female)'],y],a
```

In [41]:
```
theDead =myTest[myTest['Hospital outcome (0 alive, 1 dead)'] == 1]
```

In [42]:
```
theAlive = myTest[myTest['Hospital outcome (0 alive, 1 dead)'] == 0]
```

In [43]:
```
theAliveSubset = theAlive.sample(n = (int(8105*1.5)))
```

In [44]:
```
int(8105*1.5)
```

Out[44]:
```
12157
```

In [45]:
```
len(theDead)
```

Out[45]:
```
8105
```

In [46]:
```
newTrain = pd.concat([theAliveSubset,theDead])
```

In [47]:
```
newTrain
```

Out[47]:

	Age (years)	Episode number	Sex (0 male, 1 female)	Hospital outcome (0 alive, 1 dead)
7175	0.010973	0.865423	0	0
3005	0.881378	-0.464727	0	0
33587	0.466900	-0.464727	0	0
44920	0.176765	-0.464727	0	0
83741	-1.356806	-0.464727	0	0
...	...	...	...	...
110072	-0.030475	-0.464727	1	1
110079	1.005722	-0.464727	0	1
110113	0.632691	-0.464727	1	1
110174	0.301108	-0.464727	0	1
110198	-2.600242	-0.464727	1	1

20262 rows × 4 columns

In [48]:
```
newTrainInput = newTrain.drop(['Hospital outcome (0 alive, 1 dead)'], axis=1)
```

In [49]:
```
newY = newTrain['Hospital outcome (0 alive, 1 dead)']
```

In [50]:
```
newY
```

Out[50]:
```
7175      0
3005      0
33587     0
44920     0
83741     0
         ..
110072    1
110079    1
110113    1
110174    1
110198    1
Name: Hospital outcome (0 alive, 1 dead), Length: 20262, dtype: int64
```

In [51]:
```
clf2 = MLPClassifier(solver='adam', alpha=1e-5, random_state=1, batch_size=int(min(200

clf2.fit(newTrainInput,newY)
y_pred2=clf2.predict(newTrainInput)
```

```
Iteration 1, loss = 0.61107699
Iteration 2, loss = 0.59701484
Iteration 3, loss = 0.59614428
Iteration 4, loss = 0.59573982
Iteration 5, loss = 0.59606167
Iteration 6, loss = 0.59560472
Iteration 7, loss = 0.59525577
Iteration 8, loss = 0.59515154
Iteration 9, loss = 0.59524079
Iteration 10, loss = 0.59529436
Iteration 11, loss = 0.59487018
Iteration 12, loss = 0.59491423
Iteration 13, loss = 0.59515139
Iteration 14, loss = 0.59585180
Iteration 15, loss = 0.59507209
Iteration 16, loss = 0.59497532
Iteration 17, loss = 0.59471753
Iteration 18, loss = 0.59489495
Iteration 19, loss = 0.59465705
Iteration 20, loss = 0.59477607
Iteration 21, loss = 0.59445974
Iteration 22, loss = 0.59478782
Iteration 23, loss = 0.59463506
Iteration 24, loss = 0.59457197
Iteration 25, loss = 0.59491778
Iteration 26, loss = 0.59468306
Iteration 27, loss = 0.59458072
Iteration 28, loss = 0.59461978
Iteration 29, loss = 0.59487817
Iteration 30, loss = 0.59461193
Iteration 31, loss = 0.59442478
Iteration 32, loss = 0.59455796
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopp
ing.
```

In [52]:
```
sum(y_pred2)
```

Out[52]:
```
6816
```

In [53]:
```
sum(y_pred2)/sum(newY) #mmmmmmmm lets see if I improve from the 1x dead
```

Out[53]:
```
0.8409623689080814
```

In [54]:
```
# Lets try 1.5x the dead.
sum(y_pred2)/sum(newY)
```

Out[54]:
```
0.8409623689080814
```

In [55]:
```
#seems like a sweet spot here.
```

In [56]:
```
len(newTrainInput)
```

Out[56]:
```
20262
```

In [56]:
```

```

score(X, y[, sample_weight])

Return the mean accuracy on the given test data and labels.

first Try lets do better

In [57]:
```
clf2.score(newTrainInput, newY)
```

Out[57]:
```
0.6543776527489883
```

Second try 75% at 3x the the rate of alive then dead lets try 4x the alive.

In [58]:
```
clf2.score(newTrainInput, newY)
```

Out[58]:
```
0.6543776527489883
```

4x The alive

In [59]:
```
clf2.score(newTrainInput, newY)
```

Out[59]:
```
0.6543776527489883
```

Even better at 4x the alive, lets try 5x

In [60]:
```
clf2.score(newTrainInput, newY)
```

Out[60]:
```
0.6543776527489883
```

5x is slightly better what about 6x.

In [61]:
```
clf2.score(newTrainInput, newY)
```

Out[61]:
```
0.6543776527489883
```

In [62]:
```
#7x
clf2.score(newTrainInput, newY)
```

Out[62]:
```
0.6543776527489883
```

In [63]:
```
#8x
clf2.score(newTrainInput, newY)
```

Out[63]:
```
0.6543776527489883
```

In [64]:
```
newTrainInput
```

Out[64]:

	Age (years)	Episode number	Sex (0 male, 1 female)
7175	0.010973	0.865423	0
3005	0.881378	-0.464727	0
33587	0.466900	-0.464727	0
44920	0.176765	-0.464727	0
83741	-1.356806	-0.464727	0
...	...	...	...
110072	-0.030475	-0.464727	1
110079	1.005722	-0.464727	0
110113	0.632691	-0.464727	1
110174	0.301108	-0.464727	0
110198	-2.600242	-0.464727	1

20262 rows × 3 columns

In [65]:
```
inputVariables
```

Out[65]:

	Age (years)	Episode number	Sex (0 male, 1 female)
0	-1.729837	-0.464727	1
1	-1.771285	-0.464727	1
2	-1.729837	-0.464727	1
3	0.591243	-0.464727	0
4	0.384004	-0.464727	0
...	...	...	...
110199	-2.600242	-0.464727	0
110200	-2.600242	-0.464727	1
110201	0.301108	-0.464727	1
110202	-2.600242	-0.464727	0
110203	-2.600242	-0.464727	0

110204 rows × 3 columns

In [66]:
```
#Ok lets try this model on the whole set.
clf2.score(inputVariables.drop(['Sex (0 male, 1 female)'], axis=1), y)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-66-0e3f854a2b20> in <cell line: 2>()
      1 #Ok lets try this model on the whole set.
----> 2 clf2.score(inputVariables.drop(['Sex (0 male, 1 female)'], axis=1), y)

/usr/local/lib/python3.10/dist-packages/sklearn/base.py in score(self, X, y, sample_w
eight)
    666         from .metrics import accuracy_score
    667
--> 668         return accuracy_score(y, self.predict(X), sample_weight=sample_weigh
t)
    669
    670     def _more_tags(self):

/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptro
n.py in predict(self, X)
   1154         """
   1155         check_is_fitted(self)
-> 1156         return self._predict(X)
   1157
   1158     def _predict(self, X, check_input=True):

/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptro
n.py in _predict(self, X, check_input)
   1158     def _predict(self, X, check_input=True):
   1159         """Private predict method with optional input validation"""
-> 1160         y_pred = self._forward_pass_fast(X, check_input=check_input)
   1161
   1162         if self.n_outputs_ == 1:

/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptro
n.py in _forward_pass_fast(self, X, check_input)
    200         """
    201         if check_input:
--> 202             X = self._validate_data(X, accept_sparse=["csr", "csc"], reset=Fa
lse)
    203
    204         # Initialize first layer

/usr/local/lib/python3.10/dist-packages/sklearn/base.py in _validate_data(self, X, y,
reset, validate_separately, **check_params)
    546             validated.
    547         """
--> 548         self._check_feature_names(X, reset=reset)
    549
    550         if y is None and self._get_tags()["requires_y"]:

/usr/local/lib/python3.10/dist-packages/sklearn/base.py in _check_feature_names(self,
X, reset)
    479                 )
    480
--> 481             raise ValueError(message)
    482
    483     def _validate_data(

ValueError: The feature names should match those that were passed during fit.
Feature names seen at fit time, yet now missing:
- Sex (0 male, 1 female)
```

In [67]:
```
#1.x on whole set
clf2.score(inputVariables, y)
```

Out[67]:
```
0.7398914739936845
```

In [68]:
```
#I will go with this
```

The accuracy increases as we increase the amount of dead but it doesn't predict anyone died.

Use SHAP and/or ELI5 to determine what variables are most important

In [69]:
```
!pip install shap
```

```
Collecting shap
  Downloading shap-0.45.0-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64.many
linux_2_17_x86_64.manylinux2014_x86_64.whl (538 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 538.2/538.2 kB 3.0 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
shap) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from
shap) (1.11.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-package
s (from shap) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (fro
m shap) (2.0.3)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.10/dist-package
s (from shap) (4.66.2)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dist-packa
ges (from shap) (24.0)
Collecting slicer==0.0.7 (from shap)
  Downloading slicer-0.0.7-py3-none-any.whl (14 kB)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (from
shap) (0.59.1)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages
(from shap) (3.0.0)
Requirement already satisfied: llvmlite<0.43,>=0.42.0dev0 in /usr/local/lib/python3.1
0/dist-packages (from numba->shap) (0.42.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/di
st-packages (from pandas->shap) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-package
s (from pandas->shap) (2024.1)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packa
ges (from pandas->shap) (2024.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packag
es (from scikit-learn->shap) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist
-packages (from scikit-learn->shap) (3.4.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (f
rom python-dateutil>=2.8.2->pandas->shap) (1.16.0)
Installing collected packages: slicer, shap
Successfully installed shap-0.45.0 slicer-0.0.7
```

In [70]:
```
import shap
```

In [71]:
```
explainer = shap.PermutationExplainer(clf2.predict,newTrainInput)
```

In [72]:
```
shap_values = explainer.shap_values(newTrainInput)
```

```
PermutationExplainer explainer: 20263it [05:02, 65.29it/s]
```

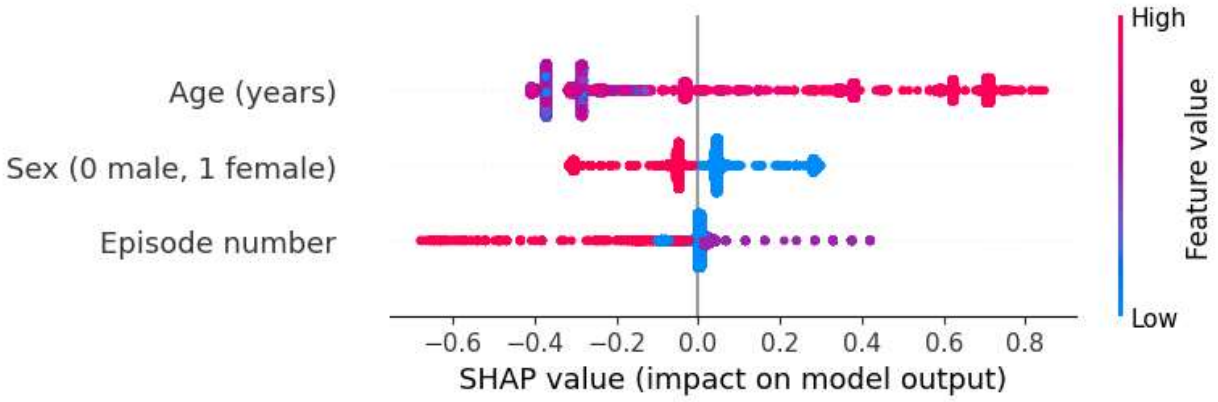
In [73]:
```
features=newTrainInput.columns
```

In [74]:
```
shap_values
```

Out[74]:
```
array([[-0.4025 ,  0.02375,  0.04875],
       [ 0.38375,  0.0025 ,  0.28375],
       [-0.315  , -0.09   ,  0.075  ],
       ...,
       [-0.035  ,  0.0075 , -0.3025 ],
       [-0.37125, -0.00125,  0.0425 ],
       [-0.28125,  0.00125, -0.05   ]])
```

In [75]:
```
import matplotlib.pyplot as plt
fig=plt.figure(figsize=(10,10))

shap.summary_plot(shap_values,newTrainInput,feature_names=features)
```



Low episode number has no impact on shap, midium episode numbers has positive impact and high episode number has negative.

Sex: low: positive impace, high: negative impact Age: High age has postive impact on shap while modeterate and low age has negative impact.

For your model, compute all the statistics shown in figure 5 of the paper.

PR AUC

In [76]:
```
from sklearn import metrics
```

In [77]:
```
from sklearn.metrics import precision_recall_curve, auc
```

In [78]:
```
newY=np.array(newY)
```

In [79]:
```
len(y_pred2)
```

Out[79]:
```
20262
```

In [81]:
```
precision, recall,_= precision_recall_curve(newY, y_pred2)
```

In [80]:
```
precision, recall,_= precision_recall_curve(newY, y_pred2)
pr_auc = auc(recall, precision)
```

In [82]:
```
print("PR-AUC Score:", pr_auc)
```

```
PR-AUC Score: 0.6369612988968295
```

In [83]:
```
#ROC AUC
from sklearn.metrics import roc_auc_score
roc_auc_score(newY, y_pred2)
```

Out[83]:
```
0.6267276370833437
```

In [84]:
```
#TP rate
```

In [85]:
```
confusion_matrix = metrics.confusion_matrix(newY, y_pred2)
```

In [86]:
```
from sklearn.metrics import precision_score

TP = np.diag(confusion_matrix)
```

In [87]:
```
confusion_matrix
```

Out[87]:
```
array([[9300, 2857],
       [4146, 3959]])
```

In [88]:
```
sum(sum(confusion_matrix))
```

Out[88]:
```
20262
```

In [89]:
```
sum(TP)/sum(sum(confusion_matrix))
```

Out[89]:
```
0.6543776527489883
```

In [90]:
```
#TN rate
FP = confusion_matrix.sum(axis=0) - np.diag(confusion_matrix)
FN = confusion_matrix.sum(axis=1) - np.diag(confusion_matrix)
TN = confusion_matrix.sum().sum() - (FP+ FN + TP)
```

In [91]:
```
TNR = TN/(TN+FP)
```

In [92]:
```
confusion_matrix[1][1]/confusion_matrix.sum()
```

Out[92]:
```
0.19539038594413188
```

In [93]:
```
#       PPV
#true pos/ tru pos + false pos
confusion_matrix[0][0]/confusion_matrix[0].sum()
```

Out[93]:
```
0.7649913630007403
```

In [94]:
```
#NPV
#true neg/ tru neg + false neg

confusion_matrix[1][1]/confusion_matrix[1].sum()
```

Out[94]:
```
0.48846391116594695
```

In [95]:
```
#MCC
from sklearn.metrics import matthews_corrcoef
matthews_corrcoef(newY, y_pred2)
```

Out[95]:
```
0.26280262963747447
```

In [96]:
```
#F_1 score
from sklearn.metrics import f1_score
f1_score(newY, y_pred2)
```

Out[96]:
```
0.5306614838147576
```

In [97]:
```
#accuracy
from sklearn.metrics import accuracy_score
accuracy_score(newY, y_pred2)
```

Out[97]:
```
0.6543776527489883
```

My model would be alomost perfect if I used the whole data to train my model becuase it preditcs that everyone is alive. But I used a subset where I got a buch of dead ones and a bunch of alive ones and trained on those.

PR-AUC Score: 0.6547867558456871 all of his models on the study cohort and the primary cohort does better. Roc AUC : 0.6405040835009896 mine does better on the selected subset of data for some of rows and columns in table 5 such as gradient boosting for the study cohort. Tp rate: 0.6551179547922219 mine has higher rate except linear regression and gradient boosting.

TN rate: 0.22697660645543383 gradient boosting bosting in the study chort is the only one with a lower rate.

PPV:0.7135806531216583 worse than all his models for both study and primary.

NPV:0.5674275138803208 my NPV is better at detecting negatives.

MCC:0.28114776476327313 Mine is better than all of his as it is higher.

F_1 score:0.5682688743358458

This is lower than all of them exept linear svm. So it's worse based on this metric so it's only better than linear SVM in terms of correction.

accuracy:0.6551179547922219

Linear SVM , RADICAL SVM and Naive bayes are the only ones that does worse everything else is more accurate for both the study and primary cohort.

```
    # This is formatted as code
```

In [98]:
```
# Over Sampling
```

In [100…
```
infile1= "/content/s41598-020-73558-3_sepsis_survival_primary_cohort.csv"
survivalTraining= pd.read_csv(infile1)
```

In [101…
```
survivalTraining.head()
```

Out[101]:

	age_years	sex_0male_1female	episode_number	hospital_outcome_1alive_0dead
0	21	1	1	1
1	20	1	1	1
2	21	1	1	1
3	77	0	1	1
4	72	0	1	1

In [102…
```
X = survivalTraining[['age_years','sex_0male_1female','episode_number']]
y = survivalTraining[['hospital_outcome_1alive_0dead']]
```

In [102…

In [102…

In [103…
```
X.head()
```

Out[103]:

	age_years	sex_0male_1female	episode_number
0	21	1	1
1	20	1	1
2	21	1	1
3	77	0	1
4	72	0	1

In [104…
```
X
```

Out[104]:

	age_years	sex_0male_1female	episode_number
0	21	1	1
1	20	1	1
2	21	1	1
3	77	0	1
4	72	0	1
...	...	...	...
110199	0	0	1
110200	0	1	1
110201	70	1	1
110202	0	0	1
110203	0	0	1

110204 rows × 3 columns

In [105…
```
y.head()
```

Out[105]:

	hospital_outcome_1alive_0dead
0	1
1	1
2	1
3	1
4	1

In [105…

In [106…
```
from sklearn.preprocessing import StandardScaler
```

In [107…
```
X_res_standardize = X[['age_years','episode_number']]
```

In [108…
```
scaler = StandardScaler()
X_res_standardized = scaler.fit_transform(X_res_standardize)
```

In [109…
```
X_res_standardized
```

Out[109]:
```
array([[-1.72983711, -0.4647268 ],
       [-1.77128498, -0.4647268 ],
       [-1.72983711, -0.4647268 ],
       ...,
       [ 0.30110817, -0.4647268 ],
       [-2.60024224, -0.4647268 ],
       [-2.60024224, -0.4647268 ]])
```

In [110…
```
X_resContinuous= pd.DataFrame(X_res_standardized,columns=X_res_standardize.columns)
```

In [111…
```
X = pd.concat([X_resContinuous,X['sex_0male_1female']],axis=1)
```

In [112…
```
X.shape
```

Out[112]:
```
(110204, 3)
```

In [113…
```
X
```

Out[113]:

	age_years	episode_number	sex_0male_1female
0	-1.729837	-0.464727	1
1	-1.771285	-0.464727	1
2	-1.729837	-0.464727	1
3	0.591243	-0.464727	0
4	0.384004	-0.464727	0
…	…	…	…
110199	-2.600242	-0.464727	0
110200	-2.600242	-0.464727	1
110201	0.301108	-0.464727	1
110202	-2.600242	-0.464727	0
110203	-2.600242	-0.464727	0

110204 rows × 3 columns

In [121…
```
y.shape
```

Out[121]:
```
(110204, 1)
```

In [122…
```
!pip install imblearn
```

```
Requirement already satisfied: imblearn in /usr/local/lib/python3.10/dist-packages
(0.0)
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-pac
kages (from imblearn) (0.12.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packag
es (from imbalanced-learn->imblearn) (1.25.2)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-package
s (from imbalanced-learn->imblearn) (1.11.4)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-
packages (from imbalanced-learn->imblearn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packag
es (from imbalanced-learn->imblearn) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist
-packages (from imbalanced-learn->imblearn) (3.4.0)
```

OverSampler

In [124…
```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(sampling_strategy=1.0, random_state=42)
X_res, y_res = ros.fit_resample(X, y)
```

In [125…
```
from sklearn.neural_network import MLPClassifier
clfOver = MLPClassifier(solver='adam', alpha=1e-5, learning_rate_init=0.0001,random_st

clfOver.fit(X_res,y_res)
y_pred1=clfOver.predict(X)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptro
n.py:1098: DataConversionWarning: A column-vector y was passed when a 1d array was ex
pected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Iteration 1, loss = 0.62440370
Iteration 2, loss = 0.61262634
Iteration 3, loss = 0.61222392
Iteration 4, loss = 0.61191425
Iteration 5, loss = 0.61176378
Iteration 6, loss = 0.61158251
Iteration 7, loss = 0.61150116
Iteration 8, loss = 0.61139142
Iteration 9, loss = 0.61131328
Iteration 10, loss = 0.61126484
Iteration 11, loss = 0.61122470
Iteration 12, loss = 0.61115343
Iteration 13, loss = 0.61110867
Iteration 14, loss = 0.61105682
Iteration 15, loss = 0.61103529
Iteration 16, loss = 0.61097922
Iteration 17, loss = 0.61098345
Iteration 18, loss = 0.61089644
Iteration 19, loss = 0.61087681
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopp
ing.
```

In [126…
```
clfOver.score(X, y)
```

Out[126]:
```
0.5786087619324163
```

In [127…
```
X_res.head()
```

Out[127]:

	age_years	episode_number	sex_0male_1female
0	-1.729837	-0.464727	1
1	-1.771285	-0.464727	1
2	-1.729837	-0.464727	1
3	0.591243	-0.464727	0
4	0.384004	-0.464727	0

In [128…
```
from sklearn.metrics import f1_score
f1_score(y, y_pred1)
```

Out[128]:
```
0.713429722741606
```

It has an accuracy of 65%

4 Main approahes of dealing with imbalance data and their explanations:

a.) Oversampling the minority

Randomly choose samples of the minority class and repeat or duplicate them. This is also called random oversampling with replacement.

b.) Undersampling the majority

In Random UnderSampling (RUS), as the name suggests, we randomly extract observations from the majority class until the classes are balanced.

c.) SMOTE oversampling the minority

Synthetic minority oversampling teqnique sholves the problem of duplicate values in the minority random oversampling. What it does is uses multiple values in the minority class to create new unique values similar to the ones in the minority class.

d.) Weighting the majority and minority differently.

Class weights are inverserly proportional to class frequency, the more frequently classes are weighted less and then frequently occuring ones.

SMOTE

In [129…
```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=0)
X_res, y_res = sm.fit_resample(X, y)
```

In [130…
```
X_res.shape
```

Out[130]:
```
(204198, 3)
```

In [131…
```
X.shape
```

Out[131]:
```
(110204, 3)
```

In [132…
```
from sklearn.neural_network import MLPClassifier
clfSMOTE = MLPClassifier(solver='adam', alpha=1e-5,learning_rate_init=0.0001, random_s

clfSMOTE.fit(X_res,y_res)
y_pred=clfSMOTE.predict(X)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptro
n.py:1098: DataConversionWarning: A column-vector y was passed when a 1d array was ex
pected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Iteration 1, loss = 0.62383711
Iteration 2, loss = 0.61140617
Iteration 3, loss = 0.61101071
Iteration 4, loss = 0.61073943
Iteration 5, loss = 0.61060091
Iteration 6, loss = 0.61045213
Iteration 7, loss = 0.61035684
Iteration 8, loss = 0.61024569
Iteration 9, loss = 0.61017016
Iteration 10, loss = 0.61014502
Iteration 11, loss = 0.61009227
Iteration 12, loss = 0.61002335
Iteration 13, loss = 0.61001540
Iteration 14, loss = 0.60994937
Iteration 15, loss = 0.60991018
Iteration 16, loss = 0.60986511
Iteration 17, loss = 0.60985941
Iteration 18, loss = 0.60980755
Iteration 19, loss = 0.60975431
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopp
ing.
```

In [133…
```
from sklearn.metrics import f1_score
f1_score(y, y_pred)
```

Out[133]:
```
0.7073051172430055
```

In [134…
```
clfSMOTE.score(X, y)
```

Out[134]:
```
0.5719665347900258
```

ADASYN

In [135…
```
from imblearn.over_sampling import ADASYN
X_resampled, y_resampled = ADASYN().fit_resample(X, y)
```

In [136…
```
from sklearn.neural_network import MLPClassifier
clfADASYN = MLPClassifier(solver='adam', alpha=1e-5,learning_rate_init=0.0001, random_

clfADASYN.fit(X_res,y_res)
y_pred3=clfADASYN.predict(X)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptro
n.py:1098: DataConversionWarning: A column-vector y was passed when a 1d array was ex
pected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Iteration 1, loss = 0.62383711
Iteration 2, loss = 0.61140617
Iteration 3, loss = 0.61101071
Iteration 4, loss = 0.61073943
Iteration 5, loss = 0.61060091
Iteration 6, loss = 0.61045213
Iteration 7, loss = 0.61035684
Iteration 8, loss = 0.61024569
Iteration 9, loss = 0.61017016
Iteration 10, loss = 0.61014502
Iteration 11, loss = 0.61009227
Iteration 12, loss = 0.61002335
Iteration 13, loss = 0.61001540
Iteration 14, loss = 0.60994937
Iteration 15, loss = 0.60991018
Iteration 16, loss = 0.60986511
Iteration 17, loss = 0.60985941
Iteration 18, loss = 0.60980755
Iteration 19, loss = 0.60975431
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopp
ing.
```

In [137…
```
clfADASYN.score(X, y)
```

Out[137]:
```
0.5719665347900258
```

In [138…
```
clfADASYN.score(X, y)
```

Out[138]:
```
0.5719665347900258
```

In [139…
```
sum(y_pred)
```

Out[139]:
```
59062
```

In [140…
```
from sklearn.metrics import f1_score
f1_score(y, y_pred3)
```

Out[140]:
```
0.7073051172430055
```

Class Weighting

In [141…
```
import tensorflow as tf
from tensorflow import keras
```

In [142…
```
METRICS = [
      keras.metrics.BinaryCrossentropy(name='cross entropy'),  # same as model's loss
      keras.metrics.MeanSquaredError(name='Brier score'),
      keras.metrics.TruePositives(name='tp'),
      keras.metrics.FalsePositives(name='fp'),
      keras.metrics.TrueNegatives(name='tn'),
      keras.metrics.FalseNegatives(name='fn'),
```

```
      keras.metrics.BinaryAccuracy(name='accuracy'),
      keras.metrics.Precision(name='precision'),
      keras.metrics.Recall(name='recall'),
      keras.metrics.AUC(name='auc'),
      keras.metrics.AUC(name='prc', curve='PR'), # precision-recall curve
]
```

In [143…

```
def make_model(metrics=METRICS, output_bias=None):
  if output_bias is not None:
    output_bias = tf.keras.initializers.Constant(output_bias)
  model = keras.Sequential([
      keras.layers.Dense(
          50, activation='relu',
          input_shape=(X.shape[-1],)),
      keras.layers.Dropout(0.5),
      keras.layers.Dense(25, activation='relu'),
      keras.layers.Dense(25, activation='relu'),
      keras.layers.Dense(1, activation='sigmoid',
                         bias_initializer=output_bias),
  ])

  model.compile(
      optimizer=keras.optimizers.Adam(learning_rate=1e-3),
      loss=keras.losses.BinaryCrossentropy(),
      metrics=metrics)

  return model
```

In [144…

```
weighted_model = make_model()

EPOCHS = 100
BATCH_SIZE = 2048
pos = 102099
neg = len(y)-pos
total = neg + pos
weight_for_0 = (1 / neg) * (total / 2.0)
weight_for_1 = (1 / pos) * (total / 2.0)
class_weight = {0: weight_for_0, 1: weight_for_1}
weighted_history = weighted_model.fit(
    X,
    y,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    # The class weights go here
    class_weight=class_weight)
```

```
Epoch 1/100
54/54 [==============================] - 2s 5ms/step - loss: 0.6830 - cross entropy:
0.5249 - Brier score: 0.1705 - tp: 93138.0000 - fp: 6904.0000 - tn: 1201.0000 - fn: 8
961.0000 - accuracy: 0.8560 - precision: 0.9310 - recall: 0.9122 - auc: 0.6278 - prc:
0.9561
Epoch 2/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6267 - cross entropy:
0.6175 - Brier score: 0.2224 - tp: 52281.0000 - fp: 1905.0000 - tn: 6200.0000 - fn: 4
9818.0000 - accuracy: 0.5307 - precision: 0.9648 - recall: 0.5121 - auc: 0.6823 - pr
c: 0.9653
Epoch 3/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6216 - cross entropy:
0.6233 - Brier score: 0.2244 - tp: 53306.0000 - fp: 1874.0000 - tn: 6231.0000 - fn: 4
8793.0000 - accuracy: 0.5402 - precision: 0.9660 - recall: 0.5221 - auc: 0.6918 - pr
c: 0.9664
Epoch 4/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6201 - cross entropy:
0.6252 - Brier score: 0.2254 - tp: 52921.0000 - fp: 1817.0000 - tn: 6288.0000 - fn: 4
9178.0000 - accuracy: 0.5373 - precision: 0.9668 - recall: 0.5183 - auc: 0.6950 - pr
c: 0.9666
Epoch 5/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6190 - cross entropy:
0.6276 - Brier score: 0.2267 - tp: 52981.0000 - fp: 1821.0000 - tn: 6284.0000 - fn: 4
9118.0000 - accuracy: 0.5378 - precision: 0.9668 - recall: 0.5189 - auc: 0.6975 - pr
c: 0.9669
Epoch 6/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6186 - cross entropy:
0.6249 - Brier score: 0.2253 - tp: 53363.0000 - fp: 1848.0000 - tn: 6257.0000 - fn: 4
8736.0000 - accuracy: 0.5410 - precision: 0.9665 - recall: 0.5227 - auc: 0.6968 - pr
c: 0.9670
Epoch 7/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6184 - cross entropy:
0.6278 - Brier score: 0.2266 - tp: 53232.0000 - fp: 1812.0000 - tn: 6293.0000 - fn: 4
8867.0000 - accuracy: 0.5401 - precision: 0.9671 - recall: 0.5214 - auc: 0.6985 - pr
c: 0.9671
Epoch 8/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6175 - cross entropy:
0.6261 - Brier score: 0.2263 - tp: 53021.0000 - fp: 1798.0000 - tn: 6307.0000 - fn: 4
9078.0000 - accuracy: 0.5383 - precision: 0.9672 - recall: 0.5193 - auc: 0.6992 - pr
c: 0.9673
Epoch 9/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6170 - cross entropy:
0.6256 - Brier score: 0.2259 - tp: 52924.0000 - fp: 1789.0000 - tn: 6316.0000 - fn: 4
9175.0000 - accuracy: 0.5375 - precision: 0.9673 - recall: 0.5184 - auc: 0.7005 - pr
c: 0.9674
Epoch 10/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6164 - cross entropy:
0.6310 - Brier score: 0.2285 - tp: 52733.0000 - fp: 1775.0000 - tn: 6330.0000 - fn: 4
9366.0000 - accuracy: 0.5359 - precision: 0.9674 - recall: 0.5165 - auc: 0.7012 - pr
c: 0.9675
Epoch 11/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6170 - cross entropy:
0.6248 - Brier score: 0.2257 - tp: 53335.0000 - fp: 1831.0000 - tn: 6274.0000 - fn: 4
8764.0000 - accuracy: 0.5409 - precision: 0.9668 - recall: 0.5224 - auc: 0.7008 - pr
c: 0.9674
Epoch 12/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6166 - cross entropy:
0.6254 - Brier score: 0.2258 - tp: 53948.0000 - fp: 1854.0000 - tn: 6251.0000 - fn: 4
8151.0000 - accuracy: 0.5463 - precision: 0.9668 - recall: 0.5284 - auc: 0.7014 - pr
c: 0.9674
```

```
Epoch 13/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6158 - cross entropy:
0.6253 - Brier score: 0.2262 - tp: 53127.0000 - fp: 1778.0000 - tn: 6327.0000 - fn: 4
8972.0000 - accuracy: 0.5395 - precision: 0.9676 - recall: 0.5203 - auc: 0.7017 - pr
c: 0.9676
Epoch 14/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6165 - cross entropy:
0.6292 - Brier score: 0.2275 - tp: 53211.0000 - fp: 1830.0000 - tn: 6275.0000 - fn: 4
8888.0000 - accuracy: 0.5398 - precision: 0.9668 - recall: 0.5212 - auc: 0.7014 - pr
c: 0.9674
Epoch 15/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6160 - cross entropy:
0.6253 - Brier score: 0.2261 - tp: 53003.0000 - fp: 1793.0000 - tn: 6312.0000 - fn: 4
9096.0000 - accuracy: 0.5382 - precision: 0.9673 - recall: 0.5191 - auc: 0.7017 - pr
c: 0.9676
Epoch 16/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6158 - cross entropy:
0.6284 - Brier score: 0.2270 - tp: 53193.0000 - fp: 1808.0000 - tn: 6297.0000 - fn: 4
8906.0000 - accuracy: 0.5398 - precision: 0.9671 - recall: 0.5210 - auc: 0.7027 - pr
c: 0.9676
Epoch 17/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6157 - cross entropy:
0.6262 - Brier score: 0.2261 - tp: 53255.0000 - fp: 1785.0000 - tn: 6320.0000 - fn: 4
8844.0000 - accuracy: 0.5406 - precision: 0.9676 - recall: 0.5216 - auc: 0.7029 - pr
c: 0.9676
Epoch 18/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6163 - cross entropy:
0.6222 - Brier score: 0.2248 - tp: 53588.0000 - fp: 1833.0000 - tn: 6272.0000 - fn: 4
8511.0000 - accuracy: 0.5432 - precision: 0.9669 - recall: 0.5249 - auc: 0.7013 - pr
c: 0.9675
Epoch 19/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6151 - cross entropy:
0.6271 - Brier score: 0.2269 - tp: 53371.0000 - fp: 1801.0000 - tn: 6304.0000 - fn: 4
8728.0000 - accuracy: 0.5415 - precision: 0.9674 - recall: 0.5227 - auc: 0.7026 - pr
c: 0.9677
Epoch 20/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6155 - cross entropy:
0.6268 - Brier score: 0.2268 - tp: 52769.0000 - fp: 1784.0000 - tn: 6321.0000 - fn: 4
9330.0000 - accuracy: 0.5362 - precision: 0.9673 - recall: 0.5168 - auc: 0.7024 - pr
c: 0.9676
Epoch 21/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6155 - cross entropy:
0.6256 - Brier score: 0.2259 - tp: 53472.0000 - fp: 1837.0000 - tn: 6268.0000 - fn: 4
8627.0000 - accuracy: 0.5421 - precision: 0.9668 - recall: 0.5237 - auc: 0.7023 - pr
c: 0.9676
Epoch 22/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6149 - cross entropy:
0.6277 - Brier score: 0.2270 - tp: 53422.0000 - fp: 1789.0000 - tn: 6316.0000 - fn: 4
8677.0000 - accuracy: 0.5421 - precision: 0.9676 - recall: 0.5232 - auc: 0.7049 - pr
c: 0.9679
Epoch 23/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6145 - cross entropy:
0.6240 - Brier score: 0.2253 - tp: 53833.0000 - fp: 1856.0000 - tn: 6249.0000 - fn: 4
8266.0000 - accuracy: 0.5452 - precision: 0.9667 - recall: 0.5273 - auc: 0.7030 - pr
c: 0.9678
Epoch 24/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6158 - cross entropy:
0.6284 - Brier score: 0.2277 - tp: 53107.0000 - fp: 1820.0000 - tn: 6285.0000 - fn: 4
8992.0000 - accuracy: 0.5389 - precision: 0.9669 - recall: 0.5202 - auc: 0.7030 - pr
c: 0.9676
```

```
Epoch 25/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6148 - cross entropy:
0.6264 - Brier score: 0.2261 - tp: 53861.0000 - fp: 1859.0000 - tn: 6246.0000 - fn: 4
8238.0000 - accuracy: 0.5454 - precision: 0.9666 - recall: 0.5275 - auc: 0.7032 - pr
c: 0.9677
Epoch 26/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6151 - cross entropy:
0.6250 - Brier score: 0.2258 - tp: 53514.0000 - fp: 1843.0000 - tn: 6262.0000 - fn: 4
8585.0000 - accuracy: 0.5424 - precision: 0.9667 - recall: 0.5241 - auc: 0.7030 - pr
c: 0.9677
Epoch 27/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6150 - cross entropy:
0.6281 - Brier score: 0.2273 - tp: 53735.0000 - fp: 1840.0000 - tn: 6265.0000 - fn: 4
8364.0000 - accuracy: 0.5444 - precision: 0.9669 - recall: 0.5263 - auc: 0.7034 - pr
c: 0.9678
Epoch 28/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6150 - cross entropy:
0.6211 - Brier score: 0.2242 - tp: 54084.0000 - fp: 1864.0000 - tn: 6241.0000 - fn: 4
8015.0000 - accuracy: 0.5474 - precision: 0.9667 - recall: 0.5297 - auc: 0.7033 - pr
c: 0.9677
Epoch 29/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6151 - cross entropy:
0.6277 - Brier score: 0.2270 - tp: 53353.0000 - fp: 1805.0000 - tn: 6300.0000 - fn: 4
8746.0000 - accuracy: 0.5413 - precision: 0.9673 - recall: 0.5226 - auc: 0.7039 - pr
c: 0.9677
Epoch 30/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6141 - cross entropy:
0.6262 - Brier score: 0.2263 - tp: 54120.0000 - fp: 1837.0000 - tn: 6268.0000 - fn: 4
7979.0000 - accuracy: 0.5480 - precision: 0.9672 - recall: 0.5301 - auc: 0.7053 - pr
c: 0.9679
Epoch 31/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6144 - cross entropy:
0.6251 - Brier score: 0.2259 - tp: 53905.0000 - fp: 1849.0000 - tn: 6256.0000 - fn: 4
8194.0000 - accuracy: 0.5459 - precision: 0.9668 - recall: 0.5280 - auc: 0.7042 - pr
c: 0.9679
Epoch 32/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6149 - cross entropy:
0.6295 - Brier score: 0.2276 - tp: 53213.0000 - fp: 1778.0000 - tn: 6327.0000 - fn: 4
8886.0000 - accuracy: 0.5403 - precision: 0.9677 - recall: 0.5212 - auc: 0.7036 - pr
c: 0.9677
Epoch 33/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6145 - cross entropy:
0.6226 - Brier score: 0.2251 - tp: 54319.0000 - fp: 1884.0000 - tn: 6221.0000 - fn: 4
7780.0000 - accuracy: 0.5493 - precision: 0.9665 - recall: 0.5320 - auc: 0.7039 - pr
c: 0.9678
Epoch 34/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6133 - cross entropy:
0.6255 - Brier score: 0.2262 - tp: 54034.0000 - fp: 1843.0000 - tn: 6262.0000 - fn: 4
8065.0000 - accuracy: 0.5471 - precision: 0.9670 - recall: 0.5292 - auc: 0.7056 - pr
c: 0.9680
Epoch 35/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6145 - cross entropy:
0.6270 - Brier score: 0.2263 - tp: 55033.0000 - fp: 1926.0000 - tn: 6179.0000 - fn: 4
7066.0000 - accuracy: 0.5554 - precision: 0.9662 - recall: 0.5390 - auc: 0.7043 - pr
c: 0.9679
Epoch 36/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6145 - cross entropy:
0.6207 - Brier score: 0.2242 - tp: 54104.0000 - fp: 1882.0000 - tn: 6223.0000 - fn: 4
7995.0000 - accuracy: 0.5474 - precision: 0.9664 - recall: 0.5299 - auc: 0.7041 - pr
c: 0.9678
```

```
Epoch 37/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6144 - cross entropy:
0.6288 - Brier score: 0.2274 - tp: 54307.0000 - fp: 1862.0000 - tn: 6243.0000 - fn: 4
7792.0000 - accuracy: 0.5494 - precision: 0.9669 - recall: 0.5319 - auc: 0.7047 - pr
c: 0.9678
Epoch 38/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6137 - cross entropy:
0.6257 - Brier score: 0.2263 - tp: 54028.0000 - fp: 1864.0000 - tn: 6241.0000 - fn: 4
8071.0000 - accuracy: 0.5469 - precision: 0.9666 - recall: 0.5292 - auc: 0.7044 - pr
c: 0.9679
Epoch 39/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6141 - cross entropy:
0.6247 - Brier score: 0.2259 - tp: 54263.0000 - fp: 1877.0000 - tn: 6228.0000 - fn: 4
7836.0000 - accuracy: 0.5489 - precision: 0.9666 - recall: 0.5315 - auc: 0.7058 - pr
c: 0.9679
Epoch 40/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6136 - cross entropy:
0.6224 - Brier score: 0.2246 - tp: 55405.0000 - fp: 1989.0000 - tn: 6116.0000 - fn: 4
6694.0000 - accuracy: 0.5582 - precision: 0.9653 - recall: 0.5427 - auc: 0.7042 - pr
c: 0.9680
Epoch 41/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6137 - cross entropy:
0.6277 - Brier score: 0.2275 - tp: 53665.0000 - fp: 1815.0000 - tn: 6290.0000 - fn: 4
8434.0000 - accuracy: 0.5440 - precision: 0.9673 - recall: 0.5256 - auc: 0.7050 - pr
c: 0.9679
Epoch 42/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6144 - cross entropy:
0.6253 - Brier score: 0.2258 - tp: 54977.0000 - fp: 1924.0000 - tn: 6181.0000 - fn: 4
7122.0000 - accuracy: 0.5550 - precision: 0.9662 - recall: 0.5385 - auc: 0.7042 - pr
c: 0.9678
Epoch 43/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6141 - cross entropy:
0.6234 - Brier score: 0.2250 - tp: 55359.0000 - fp: 1953.0000 - tn: 6152.0000 - fn: 4
6740.0000 - accuracy: 0.5582 - precision: 0.9659 - recall: 0.5422 - auc: 0.7051 - pr
c: 0.9679
Epoch 44/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6133 - cross entropy:
0.6257 - Brier score: 0.2263 - tp: 54922.0000 - fp: 1947.0000 - tn: 6158.0000 - fn: 4
7177.0000 - accuracy: 0.5542 - precision: 0.9658 - recall: 0.5379 - auc: 0.7052 - pr
c: 0.9680
Epoch 45/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6135 - cross entropy:
0.6271 - Brier score: 0.2270 - tp: 54074.0000 - fp: 1847.0000 - tn: 6258.0000 - fn: 4
8025.0000 - accuracy: 0.5475 - precision: 0.9670 - recall: 0.5296 - auc: 0.7047 - pr
c: 0.9680
Epoch 46/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6134 - cross entropy:
0.6236 - Brier score: 0.2253 - tp: 55273.0000 - fp: 1955.0000 - tn: 6150.0000 - fn: 4
6826.0000 - accuracy: 0.5574 - precision: 0.9658 - recall: 0.5414 - auc: 0.7050 - pr
c: 0.9681
Epoch 47/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6146 - cross entropy:
0.6263 - Brier score: 0.2267 - tp: 53896.0000 - fp: 1855.0000 - tn: 6250.0000 - fn: 4
8203.0000 - accuracy: 0.5458 - precision: 0.9667 - recall: 0.5279 - auc: 0.7035 - pr
c: 0.9677
Epoch 48/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6137 - cross entropy:
0.6236 - Brier score: 0.2254 - tp: 55178.0000 - fp: 1964.0000 - tn: 6141.0000 - fn: 4
6921.0000 - accuracy: 0.5564 - precision: 0.9656 - recall: 0.5404 - auc: 0.7045 - pr
c: 0.9679
```

```
Epoch 49/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6139 - cross entropy:
0.6268 - Brier score: 0.2268 - tp: 54354.0000 - fp: 1910.0000 - tn: 6195.0000 - fn: 4
7745.0000 - accuracy: 0.5494 - precision: 0.9661 - recall: 0.5324 - auc: 0.7042 - pr
c: 0.9679
Epoch 50/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6138 - cross entropy:
0.6237 - Brier score: 0.2255 - tp: 54904.0000 - fp: 1923.0000 - tn: 6182.0000 - fn: 4
7195.0000 - accuracy: 0.5543 - precision: 0.9662 - recall: 0.5378 - auc: 0.7050 - pr
c: 0.9679
Epoch 51/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6142 - cross entropy:
0.6246 - Brier score: 0.2253 - tp: 55473.0000 - fp: 1999.0000 - tn: 6106.0000 - fn: 4
6626.0000 - accuracy: 0.5588 - precision: 0.9652 - recall: 0.5433 - auc: 0.7044 - pr
c: 0.9678
Epoch 52/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6135 - cross entropy:
0.6277 - Brier score: 0.2275 - tp: 53695.0000 - fp: 1821.0000 - tn: 6284.0000 - fn: 4
8404.0000 - accuracy: 0.5443 - precision: 0.9672 - recall: 0.5259 - auc: 0.7047 - pr
c: 0.9679
Epoch 53/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6137 - cross entropy:
0.6203 - Brier score: 0.2237 - tp: 55597.0000 - fp: 1977.0000 - tn: 6128.0000 - fn: 4
6502.0000 - accuracy: 0.5601 - precision: 0.9657 - recall: 0.5445 - auc: 0.7056 - pr
c: 0.9678
Epoch 54/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6143 - cross entropy:
0.6296 - Brier score: 0.2276 - tp: 55430.0000 - fp: 1950.0000 - tn: 6155.0000 - fn: 4
6669.0000 - accuracy: 0.5588 - precision: 0.9660 - recall: 0.5429 - auc: 0.7045 - pr
c: 0.9678
Epoch 55/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6135 - cross entropy:
0.6230 - Brier score: 0.2250 - tp: 55034.0000 - fp: 1918.0000 - tn: 6187.0000 - fn: 4
7065.0000 - accuracy: 0.5555 - precision: 0.9663 - recall: 0.5390 - auc: 0.7053 - pr
c: 0.9679
Epoch 56/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6135 - cross entropy:
0.6246 - Brier score: 0.2260 - tp: 54732.0000 - fp: 1914.0000 - tn: 6191.0000 - fn: 4
7367.0000 - accuracy: 0.5528 - precision: 0.9662 - recall: 0.5361 - auc: 0.7061 - pr
c: 0.9680
Epoch 57/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6131 - cross entropy:
0.6244 - Brier score: 0.2255 - tp: 55749.0000 - fp: 2017.0000 - tn: 6088.0000 - fn: 4
6350.0000 - accuracy: 0.5611 - precision: 0.9651 - recall: 0.5460 - auc: 0.7055 - pr
c: 0.9680
Epoch 58/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6139 - cross entropy:
0.6245 - Brier score: 0.2256 - tp: 55074.0000 - fp: 1935.0000 - tn: 6170.0000 - fn: 4
7025.0000 - accuracy: 0.5557 - precision: 0.9661 - recall: 0.5394 - auc: 0.7048 - pr
c: 0.9678
Epoch 59/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6134 - cross entropy:
0.6269 - Brier score: 0.2268 - tp: 54522.0000 - fp: 1910.0000 - tn: 6195.0000 - fn: 4
7577.0000 - accuracy: 0.5510 - precision: 0.9662 - recall: 0.5340 - auc: 0.7051 - pr
c: 0.9680
Epoch 60/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6129 - cross entropy:
0.6229 - Brier score: 0.2253 - tp: 55049.0000 - fp: 1935.0000 - tn: 6170.0000 - fn: 4
7050.0000 - accuracy: 0.5555 - precision: 0.9660 - recall: 0.5392 - auc: 0.7053 - pr
c: 0.9681
```

```
Epoch 61/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6133 - cross entropy:
0.6222 - Brier score: 0.2249 - tp: 55155.0000 - fp: 1921.0000 - tn: 6184.0000 - fn: 4
6944.0000 - accuracy: 0.5566 - precision: 0.9663 - recall: 0.5402 - auc: 0.7055 - pr
c: 0.9680
Epoch 62/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6134 - cross entropy:
0.6262 - Brier score: 0.2263 - tp: 55217.0000 - fp: 1942.0000 - tn: 6163.0000 - fn: 4
6882.0000 - accuracy: 0.5570 - precision: 0.9660 - recall: 0.5408 - auc: 0.7058 - pr
c: 0.9680
Epoch 63/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6129 - cross entropy:
0.6271 - Brier score: 0.2269 - tp: 54514.0000 - fp: 1923.0000 - tn: 6182.0000 - fn: 4
7585.0000 - accuracy: 0.5508 - precision: 0.9659 - recall: 0.5339 - auc: 0.7061 - pr
c: 0.9681
Epoch 64/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6133 - cross entropy:
0.6266 - Brier score: 0.2267 - tp: 55043.0000 - fp: 1936.0000 - tn: 6169.0000 - fn: 4
7056.0000 - accuracy: 0.5554 - precision: 0.9660 - recall: 0.5391 - auc: 0.7052 - pr
c: 0.9679
Epoch 65/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6132 - cross entropy:
0.6225 - Brier score: 0.2250 - tp: 55376.0000 - fp: 1974.0000 - tn: 6131.0000 - fn: 4
6723.0000 - accuracy: 0.5581 - precision: 0.9656 - recall: 0.5424 - auc: 0.7060 - pr
c: 0.9680
Epoch 66/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6136 - cross entropy:
0.6265 - Brier score: 0.2261 - tp: 56175.0000 - fp: 2045.0000 - tn: 6060.0000 - fn: 4
5924.0000 - accuracy: 0.5647 - precision: 0.9649 - recall: 0.5502 - auc: 0.7047 - pr
c: 0.9679
Epoch 67/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6129 - cross entropy:
0.6251 - Brier score: 0.2265 - tp: 54480.0000 - fp: 1876.0000 - tn: 6229.0000 - fn: 4
7619.0000 - accuracy: 0.5509 - precision: 0.9667 - recall: 0.5336 - auc: 0.7065 - pr
c: 0.9681
Epoch 68/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6129 - cross entropy:
0.6245 - Brier score: 0.2257 - tp: 55667.0000 - fp: 1993.0000 - tn: 6112.0000 - fn: 4
6432.0000 - accuracy: 0.5606 - precision: 0.9654 - recall: 0.5452 - auc: 0.7059 - pr
c: 0.9681
Epoch 69/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6129 - cross entropy:
0.6268 - Brier score: 0.2269 - tp: 55061.0000 - fp: 1950.0000 - tn: 6155.0000 - fn: 4
7038.0000 - accuracy: 0.5555 - precision: 0.9658 - recall: 0.5393 - auc: 0.7061 - pr
c: 0.9681
Epoch 70/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6136 - cross entropy:
0.6234 - Brier score: 0.2249 - tp: 56034.0000 - fp: 2025.0000 - tn: 6080.0000 - fn: 4
6065.0000 - accuracy: 0.5636 - precision: 0.9651 - recall: 0.5488 - auc: 0.7046 - pr
c: 0.9679
Epoch 71/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6133 - cross entropy:
0.6237 - Brier score: 0.2258 - tp: 55773.0000 - fp: 2003.0000 - tn: 6102.0000 - fn: 4
6326.0000 - accuracy: 0.5615 - precision: 0.9653 - recall: 0.5463 - auc: 0.7057 - pr
c: 0.9681
Epoch 72/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6133 - cross entropy:
0.6231 - Brier score: 0.2252 - tp: 55244.0000 - fp: 1949.0000 - tn: 6156.0000 - fn: 4
6855.0000 - accuracy: 0.5571 - precision: 0.9659 - recall: 0.5411 - auc: 0.7049 - pr
c: 0.9681
```

```
Epoch 73/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6132 - cross entropy:
0.6248 - Brier score: 0.2259 - tp: 55072.0000 - fp: 1929.0000 - tn: 6176.0000 - fn: 4
7027.0000 - accuracy: 0.5558 - precision: 0.9662 - recall: 0.5394 - auc: 0.7053 - pr
c: 0.9680
Epoch 74/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6137 - cross entropy:
0.6261 - Brier score: 0.2267 - tp: 54796.0000 - fp: 1909.0000 - tn: 6196.0000 - fn: 4
7303.0000 - accuracy: 0.5534 - precision: 0.9663 - recall: 0.5367 - auc: 0.7045 - pr
c: 0.9679
Epoch 75/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6137 - cross entropy:
0.6256 - Brier score: 0.2262 - tp: 55437.0000 - fp: 1999.0000 - tn: 6106.0000 - fn: 4
6662.0000 - accuracy: 0.5584 - precision: 0.9652 - recall: 0.5430 - auc: 0.7046 - pr
c: 0.9679
Epoch 76/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6126 - cross entropy:
0.6209 - Brier score: 0.2239 - tp: 56146.0000 - fp: 2015.0000 - tn: 6090.0000 - fn: 4
5953.0000 - accuracy: 0.5647 - precision: 0.9654 - recall: 0.5499 - auc: 0.7058 - pr
c: 0.9682
Epoch 77/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6129 - cross entropy:
0.6250 - Brier score: 0.2263 - tp: 55322.0000 - fp: 1951.0000 - tn: 6154.0000 - fn: 4
6777.0000 - accuracy: 0.5578 - precision: 0.9659 - recall: 0.5418 - auc: 0.7059 - pr
c: 0.9682
Epoch 78/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6134 - cross entropy:
0.6281 - Brier score: 0.2274 - tp: 54681.0000 - fp: 1943.0000 - tn: 6162.0000 - fn: 4
7418.0000 - accuracy: 0.5521 - precision: 0.9657 - recall: 0.5356 - auc: 0.7050 - pr
c: 0.9680
Epoch 79/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6133 - cross entropy:
0.6216 - Brier score: 0.2247 - tp: 55440.0000 - fp: 1968.0000 - tn: 6137.0000 - fn: 4
6659.0000 - accuracy: 0.5588 - precision: 0.9657 - recall: 0.5430 - auc: 0.7053 - pr
c: 0.9679
Epoch 80/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6133 - cross entropy:
0.6272 - Brier score: 0.2270 - tp: 54621.0000 - fp: 1881.0000 - tn: 6224.0000 - fn: 4
7478.0000 - accuracy: 0.5521 - precision: 0.9667 - recall: 0.5350 - auc: 0.7053 - pr
c: 0.9680
Epoch 81/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6133 - cross entropy:
0.6228 - Brier score: 0.2248 - tp: 55195.0000 - fp: 1949.0000 - tn: 6156.0000 - fn: 4
6904.0000 - accuracy: 0.5567 - precision: 0.9659 - recall: 0.5406 - auc: 0.7061 - pr
c: 0.9680
Epoch 82/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6129 - cross entropy:
0.6267 - Brier score: 0.2267 - tp: 54645.0000 - fp: 1892.0000 - tn: 6213.0000 - fn: 4
7454.0000 - accuracy: 0.5522 - precision: 0.9665 - recall: 0.5352 - auc: 0.7057 - pr
c: 0.9680
Epoch 83/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6130 - cross entropy:
0.6208 - Brier score: 0.2243 - tp: 54774.0000 - fp: 1925.0000 - tn: 6180.0000 - fn: 4
7325.0000 - accuracy: 0.5531 - precision: 0.9660 - recall: 0.5365 - auc: 0.7056 - pr
c: 0.9680
Epoch 84/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6131 - cross entropy:
0.6303 - Brier score: 0.2282 - tp: 54977.0000 - fp: 1919.0000 - tn: 6186.0000 - fn: 4
7122.0000 - accuracy: 0.5550 - precision: 0.9663 - recall: 0.5385 - auc: 0.7062 - pr
c: 0.9680
```

```
Epoch 85/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6137 - cross entropy:
0.6224 - Brier score: 0.2246 - tp: 55179.0000 - fp: 1973.0000 - tn: 6132.0000 - fn: 4
6920.0000 - accuracy: 0.5563 - precision: 0.9655 - recall: 0.5404 - auc: 0.7052 - pr
c: 0.9678
Epoch 86/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6133 - cross entropy:
0.6236 - Brier score: 0.2257 - tp: 54312.0000 - fp: 1884.0000 - tn: 6221.0000 - fn: 4
7787.0000 - accuracy: 0.5493 - precision: 0.9665 - recall: 0.5320 - auc: 0.7058 - pr
c: 0.9681
Epoch 87/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6132 - cross entropy:
0.6216 - Brier score: 0.2244 - tp: 55813.0000 - fp: 2012.0000 - tn: 6093.0000 - fn: 4
6286.0000 - accuracy: 0.5617 - precision: 0.9652 - recall: 0.5467 - auc: 0.7053 - pr
c: 0.9680
Epoch 88/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6128 - cross entropy:
0.6285 - Brier score: 0.2280 - tp: 54257.0000 - fp: 1891.0000 - tn: 6214.0000 - fn: 4
7842.0000 - accuracy: 0.5487 - precision: 0.9663 - recall: 0.5314 - auc: 0.7062 - pr
c: 0.9681
Epoch 89/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6126 - cross entropy:
0.6232 - Brier score: 0.2246 - tp: 55869.0000 - fp: 1996.0000 - tn: 6109.0000 - fn: 4
6230.0000 - accuracy: 0.5624 - precision: 0.9655 - recall: 0.5472 - auc: 0.7066 - pr
c: 0.9682
Epoch 90/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6130 - cross entropy:
0.6208 - Brier score: 0.2244 - tp: 55453.0000 - fp: 1985.0000 - tn: 6120.0000 - fn: 4
6646.0000 - accuracy: 0.5587 - precision: 0.9654 - recall: 0.5431 - auc: 0.7056 - pr
c: 0.9681
Epoch 91/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6128 - cross entropy:
0.6286 - Brier score: 0.2277 - tp: 54425.0000 - fp: 1865.0000 - tn: 6240.0000 - fn: 4
7674.0000 - accuracy: 0.5505 - precision: 0.9669 - recall: 0.5331 - auc: 0.7057 - pr
c: 0.9681
Epoch 92/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6131 - cross entropy:
0.6213 - Brier score: 0.2244 - tp: 55902.0000 - fp: 1975.0000 - tn: 6130.0000 - fn: 4
6197.0000 - accuracy: 0.5629 - precision: 0.9659 - recall: 0.5475 - auc: 0.7060 - pr
c: 0.9681
Epoch 93/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6134 - cross entropy:
0.6266 - Brier score: 0.2269 - tp: 54202.0000 - fp: 1886.0000 - tn: 6219.0000 - fn: 4
7897.0000 - accuracy: 0.5483 - precision: 0.9664 - recall: 0.5309 - auc: 0.7051 - pr
c: 0.9679
Epoch 94/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6126 - cross entropy:
0.6258 - Brier score: 0.2264 - tp: 55091.0000 - fp: 1939.0000 - tn: 6166.0000 - fn: 4
7008.0000 - accuracy: 0.5559 - precision: 0.9660 - recall: 0.5396 - auc: 0.7061 - pr
c: 0.9681
Epoch 95/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6131 - cross entropy:
0.6244 - Brier score: 0.2256 - tp: 55704.0000 - fp: 2009.0000 - tn: 6096.0000 - fn: 4
6395.0000 - accuracy: 0.5608 - precision: 0.9652 - recall: 0.5456 - auc: 0.7050 - pr
c: 0.9680
Epoch 96/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6126 - cross entropy:
0.6227 - Brier score: 0.2253 - tp: 55014.0000 - fp: 1912.0000 - tn: 6193.0000 - fn: 4
7085.0000 - accuracy: 0.5554 - precision: 0.9664 - recall: 0.5388 - auc: 0.7059 - pr
c: 0.9681
```

```
Epoch 97/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6130 - cross entropy:
0.6230 - Brier score: 0.2250 - tp: 55950.0000 - fp: 2008.0000 - tn: 6097.0000 - fn: 4
6149.0000 - accuracy: 0.5630 - precision: 0.9654 - recall: 0.5480 - auc: 0.7056 - pr
c: 0.9680
Epoch 98/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6124 - cross entropy:
0.6239 - Brier score: 0.2257 - tp: 54577.0000 - fp: 1872.0000 - tn: 6233.0000 - fn: 4
7522.0000 - accuracy: 0.5518 - precision: 0.9668 - recall: 0.5345 - auc: 0.7061 - pr
c: 0.9682
Epoch 99/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6131 - cross entropy:
0.6265 - Brier score: 0.2269 - tp: 54919.0000 - fp: 1922.0000 - tn: 6183.0000 - fn: 4
7180.0000 - accuracy: 0.5544 - precision: 0.9662 - recall: 0.5379 - auc: 0.7059 - pr
c: 0.9680
Epoch 100/100
54/54 [==============================] - 0s 4ms/step - loss: 0.6129 - cross entropy:
0.6232 - Brier score: 0.2249 - tp: 55452.0000 - fp: 1962.0000 - tn: 6143.0000 - fn: 4
6647.0000 - accuracy: 0.5589 - precision: 0.9658 - recall: 0.5431 - auc: 0.7058 - pr
c: 0.9681
```

In [145…
```
sum(np.array(y))
```

Out[145]:
```
array([102099])
```

In [146…
```
ypred4 = weighted_model.predict(X)
```

```
3444/3444 [==============================] - 3s 943us/step
```

In [147…
```
ypred4_corr=[]

for ch in ypred4:
  if ch>.5:
    ypred4_corr.append(1)
  else:
    ypred4_corr.append(0)
```

In [148…
```
ypred4_corr
```

Out[148]:
```
[1,
 1,
 1,
 0,
 0,
 0,
 0,
 0,
 1,
 1,
 1,
 1,
 0,
 0,
 0,
 0,
 1,
 0,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 0,
 0,
 0,
 1,
 0,
 0,
 0,
 1,
 0,
 1,
 1,
 1,
 1,
 1,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 1,
 1,
 1,
 0,
```

```
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
0,
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
0,
1,
1,
1,
1,
1,
1,
0,
0,
0,
0,
0,
0,
0,
0,
1,
1,
1,
```

```
0,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
0,
1,
1,
1,
0,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
0,
0,
1,
1,
1,
1,
0,
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
1,
0,
1,
1,
```

```
0,
1,
1,
0,
1,
1,
1,
1,
0,
1,
1,
1,
1,
0,
1,
1,
1,
1,
0,
0,
0,
1,
1,
1,
0,
0,
0,
0,
0,
1,
1,
0,
0,
0,
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
```

```
0,
1,
1,
1,
0,
0,
1,
1,
0,
1,
1,
0,
1,
1,
1,
0,
0,
1,
0,
1,
0,
0,
1,
1,
1,
1,
0,
0,
0,
0,
1,
1,
1,
0,
0,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
0,
1,
1,
1,
```

```
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
0,
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
0,
0,
0,
0,
0,
1,
0,
1,
0,
0,
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
0,
0,
0,
1,
0,
1,
1,
```

```
0,
0,
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
0,
0,
1,
1,
1,
0,
0,
1,
1,
1,
1,
1,
1,
1,
0,
0,
1,
0,
0,
1,
1,
1,
0,
1,
0,
0,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
```

```
1,
1,
1,
0,
1,
1,
0,
1,
1,
1,
0,
1,
0,
1,
1,
0,
1,
0,
1,
1,
1,
1,
1,
0,
0,
0,
0,
1,
1,
0,
0,
0,
0,
1,
1,
1,
0,
0,
1,
1,
1,
0,
0,
0,
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
1,
0,
0,
1,
0,
0,
```

```
1,
0,
0,
0,
0,
1,
0,
1,
1,
1,
1,
1,
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
0,
0,
1,
1,
1,
1,
1,
1,
1,
0,
0,
0,
0,
1,
0,
0,
0,
0,
0,
1,
1,
1,
1,
1,
0,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
```

```
1,
1,
1,
1,
1,
1,
1,
0,
0,
1,
0,
0,
0,
0,
1,
1,
1,
1,
1,
0,
1,
0,
1,
1,
1,
1,
1,
1,
0,
1,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
1,
0,
1,
0,
1,
1,
0,
0,
1,
0,
0,
1,
0,
1,
1,
1,
1,
0,
```

```
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
0,
1,
1,
0,
0,
1,
0,
0,
0,
0,
0,
1,
1,
1,
1,
0,
1,
1,
0,
0,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
0,
1,
1,
1,
1,
0,
0,
1,
1,
1,
1,
1,
1,
1,
```

```
0,
1,
0,
0,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
0,
1,
1,
0,
0,
1,
0,
1,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
0,
1,
0,
0,
0,
0,
1,
1,
0,
0,
0,
1,
1,
0,
1,
0,
0,
1,
1,
1,
0,
```

```
1,
0,
0,
1,
1,
1,
0,
0,
1,
1,
0,
0,
0,
1,
1,
1,
1,
0,
0,
1,
0,
1,
1,
1,
0,
0,
0,
1,
1,
1,
1,
0,
1,
0,
1,
1,
1,
1,
0,
1,
0,
0,
0,
0,
0,
0,
1,
1,
0,
1,
1,
1,
1,
1,
1,
0,
1,
0,
1,
0,
0,
0,
```

```
0,
0,
0,
1,
0,
0,
0,
0,
0,
1,
1,
1,
0,
1,
0,
0,
0,
1,
1,
1,
1,
1,
1,
0,
0,
0,
1,
0,
0,
0,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
1,
0,
1,
0,
0,
1,
1,
0,
```

```
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
0,
1,
1,
0,
1,
1,
0,
1,
1,
0,
0,
0,
0,
0,
0,
0,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
0,
0,
0,
0,
1,
0,
0,
0,
0,
0,
0,
1,
0,
```

```
1,
1,
1,
0,
1,
1,
0,
0,
0,
1,
0,
0,
0,
0,
0,
0,
0,
0,
1,
1,
1,
1,
1,
0,
0,
1,
1,
0,
0,
1,
1,
1,
1,
1,
1,
0,
1,
1,
1,
0,
0,
0,
0,
1,
1,
1,
1,
0,
0,
0,
1,
0,
1,
1,
0,
1,
1,
1,
0,
0,
```

```
1,
1,
0,
1,
0,
1,
1,
1,
0,
1,
1,
0,
0,
0,
1,
1,
1,
1,
0,
0,
1,
1,
0,
1,
1,
1,
1,
0,
0,
0,
0,
1,
1,
0,
0,
0,
0,
0,
0,
...]
```

In [149…
```
type(y)
```

Out[149]:
```
pandas.core.frame.DataFrame
def __init__(data=None, index: Axes | None=None, columns: Axes | None=Non
e, dtype: Dtype | None=None, copy: bool | None=None) -> None

Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Data structure also contains labeled axes (rows and columns).
Arithmetic operations align on both row and column labels. Can be
thought of as a dict-like container for Series objects. The primary
pandas data structure.
```

In [ ]:
```
from sklearn.metrics import f1_score
f1_score(y, ypred4_corr)
```

Out[ ]: 0.733585445377744

Undersampling the majority

In [150…
```
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(sampling_strategy=1.0, random_state=42)
X_res, y_res = rus.fit_resample(X, y)
```

In [152…
```
from sklearn.neural_network import MLPClassifier
clfOver = MLPClassifier(solver='adam', alpha=1e-5, learning_rate_init=0.0001,random_st

clfOver.fit(X_res,y_res)
y_pred2=clfOver.predict(X)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptro
n.py:1098: DataConversionWarning: A column-vector y was passed when a 1d array was ex
pected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
Iteration 1, loss = 0.69329853
Iteration 2, loss = 0.65800511
Iteration 3, loss = 0.63284669
Iteration 4, loss = 0.62037350
Iteration 5, loss = 0.61695630
Iteration 6, loss = 0.61640464
Iteration 7, loss = 0.61604539
Iteration 8, loss = 0.61589031
Iteration 9, loss = 0.61563148
Iteration 10, loss = 0.61550519
Iteration 11, loss = 0.61539942
Iteration 12, loss = 0.61533314
Iteration 13, loss = 0.61519861
Iteration 14, loss = 0.61509969
Iteration 15, loss = 0.61502198
Iteration 16, loss = 0.61499287
Iteration 17, loss = 0.61489413
Iteration 18, loss = 0.61481676
Iteration 19, loss = 0.61472776
Iteration 20, loss = 0.61477141
Iteration 21, loss = 0.61464495
Iteration 22, loss = 0.61469753
Iteration 23, loss = 0.61451511
Iteration 24, loss = 0.61452235
Iteration 25, loss = 0.61450783
Iteration 26, loss = 0.61442601
Iteration 27, loss = 0.61439124
Iteration 28, loss = 0.61441408
Iteration 29, loss = 0.61435053
Iteration 30, loss = 0.61432863
Iteration 31, loss = 0.61432097
Iteration 32, loss = 0.61421360
Iteration 33, loss = 0.61422682
Iteration 34, loss = 0.61425327
Iteration 35, loss = 0.61420205
Iteration 36, loss = 0.61417706
Iteration 37, loss = 0.61410596
Iteration 38, loss = 0.61412758
Iteration 39, loss = 0.61405577
Iteration 40, loss = 0.61401539
Iteration 41, loss = 0.61402409
Iteration 42, loss = 0.61400741
Iteration 43, loss = 0.61395749
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopp
ing.
```

In [153…

```
from sklearn.metrics import f1_score
f1_score(y, y_pred2)
```

Out[153]:

```
0.7030460804523968
```

The weighitng model with keras worked the best with an f1 score of .73.

In [ ]: