Algaihty's Project1

```
andas as pd
earn.metrics import confusion_matrix, classification_report, precision_recall_curv
umpy as np
sorflow.keras.models import Sequential
sorflow.keras.layers import Dense
earn.model_selection import train_test_split
atplotlib.pyplot as plt
eaborn as sns
```

```
path_primary = '/content/s41598-020-73558-3_sepsis_survival_primary_cohort.csv'
path_study = '/content/s41598-020-73558-3_sepsis_survival_study_cohort.csv'
path_validation = '/content/s41598-020-73558-3_sepsis_survival_validation_cohort.

df_primary = pd.read_csv(path_primary)
df_study = pd.read_csv(path_study)
df_validation = pd.read_csv(path_validation)
```

```
# Print the shape of the datasets
print("Primary Cohort Shape:", df_primary.shape)
print("Study Cohort Shape:", df_study.shape)
print("Validation Cohort Shape:", df_validation.shape)
```

```
Primary Cohort Shape: (110204, 4)
Study Cohort Shape: (19051, 4)
Validation Cohort Shape: (137, 4)
```

```
df_primary.columns
```

```
Index(['age_years', 'sex_0male_1female', 'episode_number',
       'hospital_outcome_1alive_0dead'],
      dtype='object')
```

```
X_primary = df_primary.drop('hospital_outcome_1alive_0dead', axis=1)
y_primary = df_primary['hospital_outcome_1alive_0dead']
X_train_primary, X_test_primary, y_train_primary, y_test_primary = train_test_spl
```

```
from sklearn.utils import class_weight

class_weights = class_weight.compute_class_weight('balanced',
                                                  classes=np.unique(y_train_prima
                                                  y=y_train_primary)
class_weights_dict = dict(enumerate(class_weights))



def build_model():
    model = Sequential([
        Dense(128, activation='relu', input_shape=(X_train_primary.shape[1],)),
        Dense(64, activation='relu'),
        Dense(1, activation='sigmoid')
    ])

    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

model = build_model()
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 128)               512

 dense_4 (Dense)             (None, 64)                8256

 dense_5 (Dense)             (None, 1)                 65

=================================================================
Total params: 8833 (34.50 KB)
Trainable params: 8833 (34.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
history2 = model.fit(X_train_primary, y_train_primary, epochs=30, batch_size=64,
                    validation_split=0.2, class_weight=class_weights_dict)
```

```
Epoch 1/30
1103/1103 [==============================] - 5s 4ms/step - loss: 0.7066 - accu
```

```
Epoch 2/30
1103/1103 [==============================] - 3s 3ms/step - loss: 0.6519 - accu
Epoch 3/30
1103/1103 [==============================] - 3s 3ms/step - loss: 0.6402 - accu
Epoch 4/30
1103/1103 [==============================] - 4s 4ms/step - loss: 0.6372 - accu
Epoch 5/30
1103/1103 [==============================] - 3s 3ms/step - loss: 0.6332 - accu
Epoch 6/30
1103/1103 [==============================] - 3s 3ms/step - loss: 0.6296 - accu
Epoch 7/30
1103/1103 [==============================] - 3s 3ms/step - loss: 0.6259 - accu
Epoch 8/30
1103/1103 [==============================] - 4s 4ms/step - loss: 0.6253 - accu
Epoch 9/30
1103/1103 [==============================] - 3s 3ms/step - loss: 0.6240 - accu
Epoch 10/30
1103/1103 [==============================] - 3s 3ms/step - loss: 0.6227 - accu
Epoch 11/30
1103/1103 [==============================] - 3s 3ms/step - loss: 0.6213 - accu
Epoch 12/30
1103/1103 [==============================] - 4s 4ms/step - loss: 0.6195 - accu
Epoch 13/30
1103/1103 [==============================] - 4s 3ms/step - loss: 0.6205 - accu
Epoch 14/30
1103/1103 [==============================] - 3s 3ms/step - loss: 0.6200 - accu
Epoch 15/30
1103/1103 [==============================] - 4s 4ms/step - loss: 0.6183 - accu
Epoch 16/30
1103/1103 [==============================] - 3s 3ms/step - loss: 0.6183 - accu
Epoch 17/30
1103/1103 [==============================] - 4s 3ms/step - loss: 0.6171 - accu
Epoch 18/30
1103/1103 [==============================] - 4s 3ms/step - loss: 0.6167 - accu
Epoch 19/30
1103/1103 [==============================] - 3s 3ms/step - loss: 0.6172 - accu
Epoch 20/30
1103/1103 [==============================] - 3s 3ms/step - loss: 0.6168 - accu
Epoch 21/30
1103/1103 [==============================] - 3s 3ms/step - loss: 0.6164 - accu
Epoch 22/30
1103/1103 [==============================] - 4s 4ms/step - loss: 0.6166 - accu
Epoch 23/30
1103/1103 [==============================] - 3s 3ms/step - loss: 0.6175 - accu
Epoch 24/30
1103/1103 [==============================] - 3s 3ms/step - loss: 0.6164 - accu
Epoch 25/30
1103/1103 [==============================] - 3s 3ms/step - loss: 0.6166 - accu
Epoch 26/30
1103/1103 [==============================] - 4s 4ms/step - loss: 0.6164 - accu
```

```
Epoch 27/30
1103/1103 [==============================] - 3s 3ms/step - loss: 0.6164 - accu
Epoch 28/30
1103/1103 [==============================] - 3s 3ms/step - loss: 0.6166 - accu
Epoch 29/30
1103/1103 [==============================] - 4s 3ms/step - loss: 0.6158 - accu
Epoch 30/30
```

```
test_loss, test_accuracy = model.evaluate(X_test_primary, y_test_primary)
```

```
689/689 [==============================] - 2s 2ms/step - loss: 0.5936 - accura
```

```
from sklearn.metrics import confusion_matrix, classification_report

# Predicting the probabilities
y_pred_prob = model.predict(X_test_primary)

# Converting probabilities to class labels based on a threshold
threshold = 0.5
y_pred = (y_pred_prob > threshold).astype(int)

# Generating confusion matrix
conf_matrix = confusion_matrix(y_test_primary, y_pred)
print("Confusion Matrix:\n", conf_matrix)

# Extracting TP, TN, FP, FN
TN, FP, FN, TP = conf_matrix.ravel()

# Calculating True Positive Rate (TPR) and True Negative Rate (TNR)
TPR = TP / (TP + FN)
TNR = TN / (TN + FP)

print(f"True Positive Rate (Sensitivity/Recall): {TPR}")
print(f"True Negative Rate (Specificity): {TNR}")

# Classification Report
class_report = classification_report(y_test_primary, y_pred)
print("Classification Report:\n", class_report)
```

```
689/689 [==============================] - 1s 1ms/step
Confusion Matrix:
 [[ 1160   453]
 [ 8792 11636]]
True Positive Rate (Sensitivity/Recall): 0.5696103387507343
True Negative Rate (Specificity): 0.7191568505889646
Classification Report:
               precision    recall  f1-score   support

           0       0.12      0.72      0.20      1613
           1       0.96      0.57      0.72     20428

    accuracy                           0.58     22041
   macro avg       0.54      0.64      0.46     22041
weighted avg       0.90      0.58      0.68     22041
```

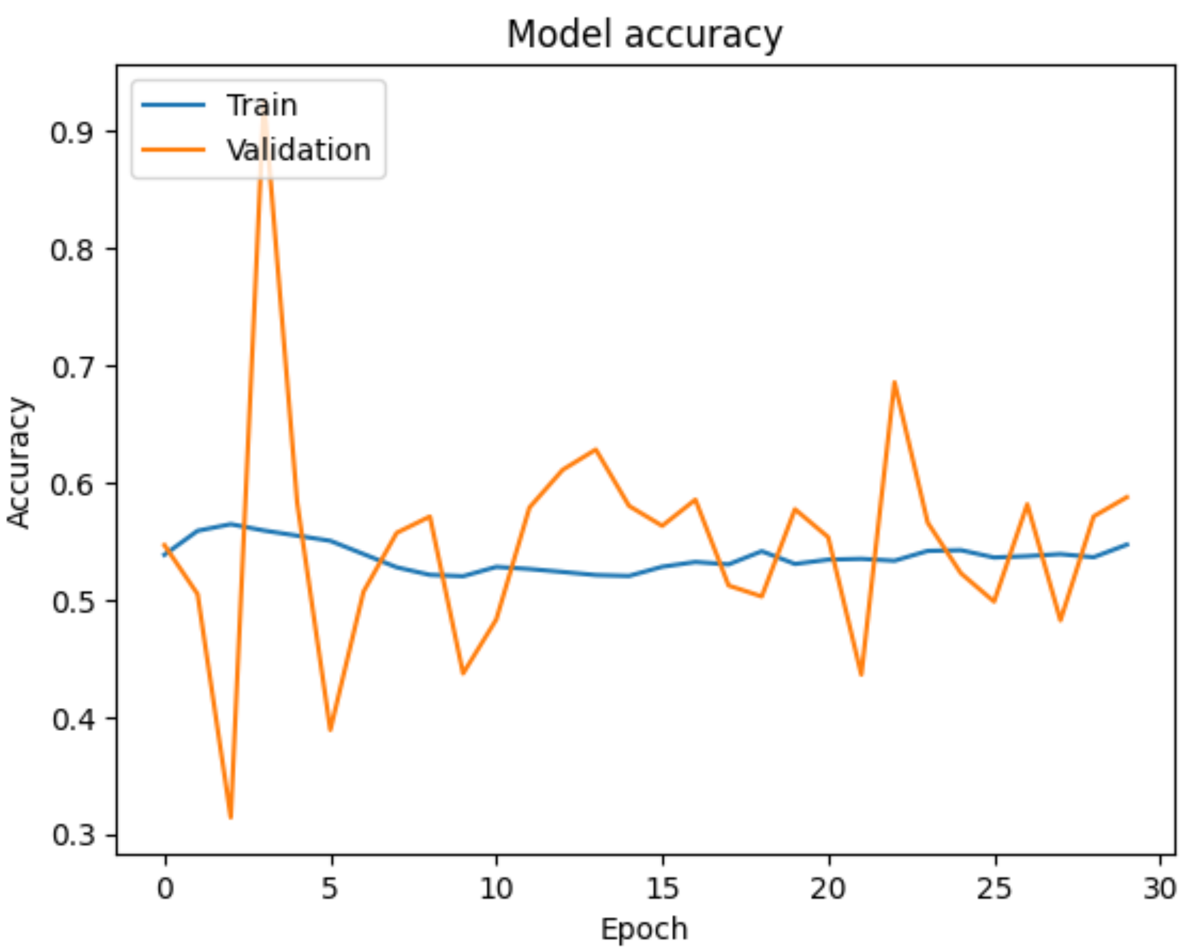
```
ppv = TP / (TP + FP)  # Positive Predictive Value
npv = TN / (TN + TN)  # Negative Predictive Value

# Matthews Correlation Coefficient
mcc_numerator = (TP * TN) - (FP * FN)
mcc_denominator = np.sqrt((TP + FP) * (TP + FN) * (TP + FP) * (TP + FN))
mcc = mcc_numerator / mcc_denominator


y_pred = model.predict(X_test_primary)
y_pred_proba = y_pred.ravel()

y_pred_class = (y_pred_proba > 0.5).astype(int)
```

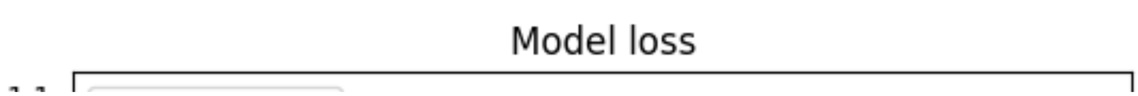
```
689/689 [==============================] - 4s 5ms/step
```

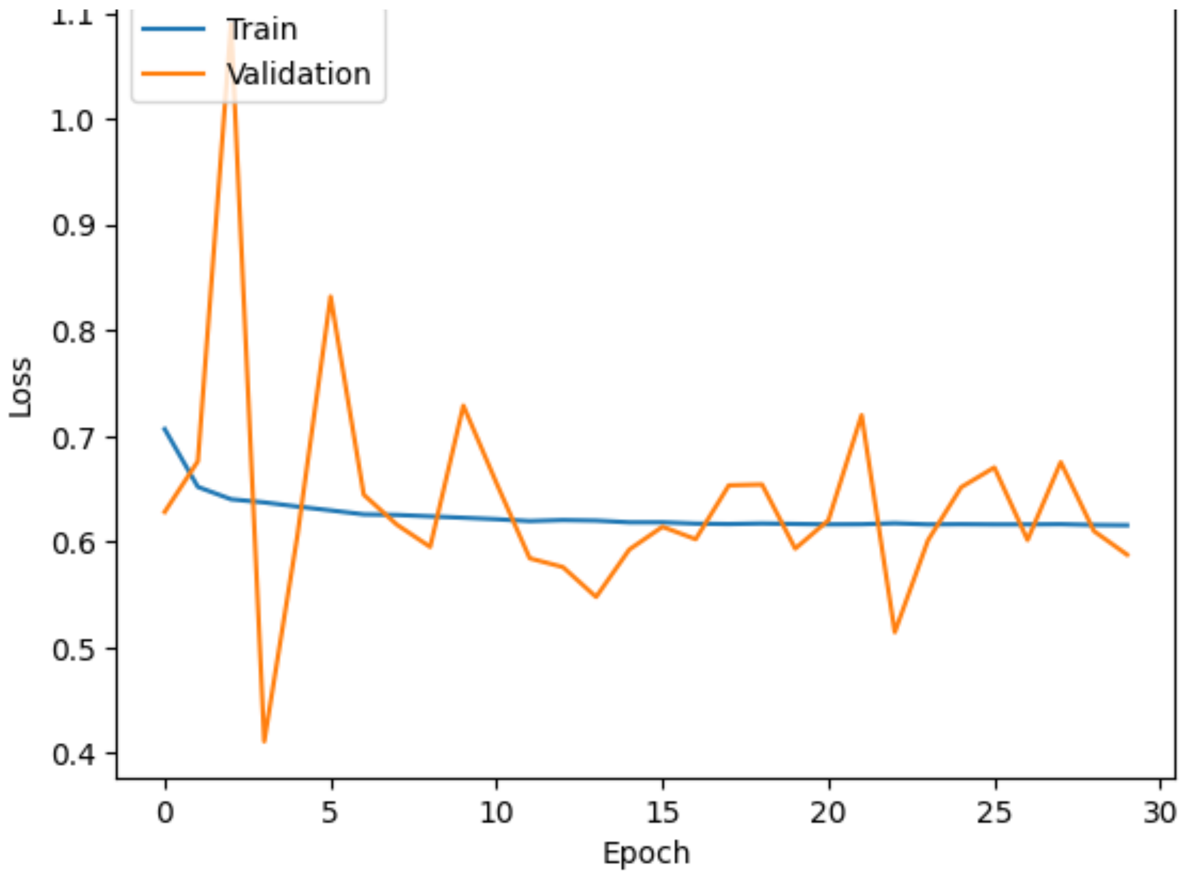
```
precision, recall, _ = precision_recall_curve(y_test_primary, y_pred_proba)
pr_auc = auc(recall, precision)

f1 = f1_score(y_test_primary, y_pred_class)
roc_auc = roc_auc_score(y_test_primary, y_pred_proba)



print(f"PPV (Precision): {ppv}")
print(f"NPV: {npv}")
print(f"MCC: {mcc}")
print(f"PR AUC: {pr_auc}")
print(f"F1 Score: {f1}")
print(f"ROC AUC: {roc_auc}")
print(f"TP Rate: {TPR}")
print(f"TN Rate: {TNR}")
print(f"Test accuracy: {test_accuracy}")
```

```
PPV (Precision): 0.9625279179419307
NPV: 0.5
MCC: 0.03852936358713991
PR AUC: 0.9672443230373847
F1 Score: 0.7156871790140541
ROC AUC: 0.7004391969691139
TP Rate: 0.5696103387507343
TN Rate: 0.7191568505889646
Test accuracy: 0.5805544257164001
```

```
# Plot training & validation accuracy values
plt.plot(history2.history['accuracy'])
plt.plot(history2.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```







Hyperparameters

```
from tensorflow.keras.optimizers import Adam
hyperparameters_sets = [
    {'units_1': 128, 'units_2': 64, 'learning_rate': 0.001},
    {'units_1': 256, 'units_2': 128, 'learning_rate': 0.001},
]


performance_logs = []

for hp in hyperparameters_sets:
    model = Sequential([
        Dense(units=hp['units_1'], activation='relu', input_shape=(X_train_primar
        Dense(units=hp['units_2'], activation='relu'),
        Dense(1, activation='sigmoid')
    ])

    model.compile(optimizer=Adam(learning_rate=hp['learning_rate']),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    history = model.fit(X_train_primary, y_train_primary, epochs=10, validation_s

    val_accuracy = history.history['val_accuracy'][-1]
    print(f"Hyperparameters: {hp}, Validation Accuracy: {val_accuracy}")
    performance_logs.append((hp, val_accuracy))

best_performance = max(performance_logs, key=lambda x: x[1])
print("Best Performance:", best_performance)
```

```
Hyperparameters: {'units_1': 128, 'units_2': 64, 'learning_rate': 0.001}, Val
Hyperparameters: {'units_1': 256, 'units_2': 128, 'learning_rate': 0.001}, Va
Best Performance: ({'units_1': 128, 'units_2': 64, 'learning_rate': 0.001}, 0.
```

```
!pip install shap
```

```
Requirement already satisfied: shap in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dis
Requirement already satisfied: slicer==0.0.7 in /usr/local/lib/python3.10/dist
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib/py
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/pythor
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-pacl
```

Double-click (or enter) to edit

```
import shap
explainer = shap.PermutationExplainer(model.predict, X_train_primary)
```

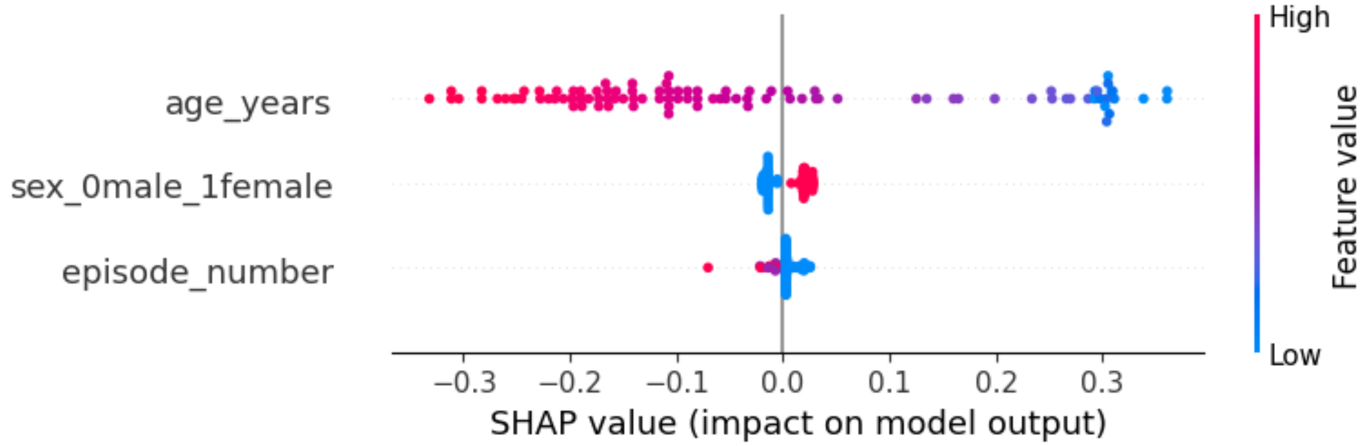
```
X_test_subset = X_test_primary.sample(100, random_state=42)

shap_values = explainer(X_test_subset)
```

```
15/15 [==============================] - 0s 2ms/step
15/15 [==============================] - 0s 3ms/step
15/15 [==============================] - 0s 2ms/step
15/15 [==============================] - 0s 2ms/step
15/15 [==============================] - 0s 2ms/step
15/15 [==============================] - 0s 2ms/step
15/15 [==============================] - 0s 2ms/step
15/15 [==============================] - 0s 2ms/step
15/15 [==============================] - 0s 2ms/step
15/15 [==============================] - 0s 2ms/step
15/15 [==============================] - 0s 2ms/step
15/15 [==============================] - 0s 2ms/step
15/15 [==============================] - 0s 2ms/step
15/15 [==============================] - 0s 3ms/step
15/15 [==============================] - 0s 2ms/step
15/15 [==============================] - 0s 3ms/step
15/15 [==============================] - 0s 2ms/step
15/15 [==============================] - 0s 2ms/step
```

```
15/15 [==============================] - 0s 3ms/step
15/15 [==============================] - 0s 3ms/step
15/15 [==============================] - 0s 3ms/step
15/15 [==============================] - 0s 5ms/step
15/15 [==============================] - 0s 4ms/step
15/15 [==============================] - 0s 4ms/step
15/15 [==============================] - 0s 4ms/step
15/15 [==============================] - 0s 4ms/step
15/15 [==============================] - 0s 2ms/step
15/15 [==============================] - 0s 3ms/step
15/15 [==============================] - 0s 5ms/step
15/15 [==============================] - 0s 4ms/step
15/15 [==============================] - 0s 3ms/step
15/15 [==============================] - 0s 5ms/step
15/15 [==============================] - 0s 3ms/step
11/11 [==============================] - 0s 3ms/step
PermutationExplainer explainer:  99%|█████████▉| 99/100 [20:36<00:12, 12.08s/
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 3ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 3ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 3ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 2ms/step
14/14 [==============================] - 0s 2ms/step
```

Double-click (or enter) to edit

```
shap.summary_plot(shap_values.values, X_test_subset, feature_names=X_test_subset.
```



Conclusion

Comparison to the Paper

In the paper, under the section "Survival predictions", AUC = 0.966 and ROC AUC close to 0.7, in the model we designed, we got ROC AUC = 0.700 which is close to what the paper indicated. We also get F1 Score = 0.716, which is less than the one in paper = 0.916. We also got TPR = 0.5696 and TNR = 0.719, which also less then the ones mentioned in the paper, TPR = 0.905, TNR = 0.898.

Summary

In our analysis to predict sepsis survival outcomes, we focused on optimizing the model's ability to distinguish between survival and non-survival. The evaluation metrics reveal that our model excels in identifying positive cases (survival), as indicated by the high PR AUC of 0.967. This metric underscores our model's strength in handling the imbalanced nature of our dataset, where the focus on survival prediction is crucial.

Our model achieved a ROC AUC of 0.700, showing a respectable ability to discriminate between the classes. The precision of our predictions (PPV) is notably high at 0.963, affirming our model's reliability in predicting survival.

The true positive and true negative rates highlight the model's better performance in recognizing survivors as opposed to accurately identifying non-survivors. The F1 score of 0.716 suggests a balance between precision and recall.

Our hyperparameter tuning efforts identified an optimal configuration of 128 units in the first layer and 64 units in the second layer, with a learning rate of 0.001. This setup achieved a validation accuracy of approximately 0.928, indicating a highly effective model on the validation set.

Using shap, we clearly can see in the graph above that gender plays a significant role in the model followed by gender and episode-number

Table 1 (Similar to Table 5)

```
metrics_values = {
    "Method": "Neural Network",
    "PR AUC": pr_auc,
    "ROC AUC": roc_auc,
    "TP Rate": TPR,
    "TN Rate": TNR,
    "PPV": ppv,
    "NPV": npv,
    "MCC": mcc,
    "F1 Score": f1,
    "Test Accuracy": test_accuracy
}


metrics_df = pd.DataFrame(metrics_values, index=["Value"])


print(metrics_df)
```

```
                Method    PR AUC   ROC AUC  TP Rate   TN Rate  PPV (Precision)
Value  Neural Network  0.967244  0.700439  0.56961  0.719157         0.962528

       NPV       MCC  F1 Score  Test Accuracy
Value  0.5  0.038529  0.715687       0.580554
```