Ray Jennings

CS 677, Fall 2022

Project #3

In [1]:
```
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import display, HTML

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score

from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier

import eli5
from eli5.sklearn import PermutationImportance
import shap
import lime
import lime.lime_tabular
```

Part 1 - EDA

In [2]:
```
#
# Read Heart Failure dataset
#

df = pd.read_csv('./heart_failure_clinical_records_dataset.csv', index_col=False)
df.reset_index(drop=True, inplace=True)

df.head()
```

Out[2]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creati
0	75.0	0	582	0	20	1	265000.00	
1	55.0	0	7861	0	38	0	263358.03	
2	65.0	0	146	0	20	0	162000.00	
3	50.0	1	111	0	20	0	210000.00	
4	65.0	1	160	1	20	0	327000.00	

Check for mising values

In [3]:
```
df.isna().any()
```

Out[3]:
```
age                         False
anaemia                     False
creatinine_phosphokinase    False
diabetes                    False
ejection_fraction           False
high_blood_pressure         False
platelets                   False
serum_creatinine            False
serum_sodium                False
sex                         False
smoking                     False
time                        False
DEATH_EVENT                 False
dtype: bool
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   age                       299 non-null    float64
 1   anaemia                   299 non-null    int64
 2   creatinine_phosphokinase  299 non-null    int64
 3   diabetes                  299 non-null    int64
 4   ejection_fraction         299 non-null    int64
 5   high_blood_pressure       299 non-null    int64
 6   platelets                 299 non-null    float64
 7   serum_creatinine          299 non-null    float64
 8   serum_sodium              299 non-null    int64
 9   sex                       299 non-null    int64
 10  smoking                   299 non-null    int64
 11  time                      299 non-null    int64
 12  DEATH_EVENT               299 non-null    int64
dtypes: float64(3), int64(10)
memory usage: 30.5 KB
```

In [5]:
```
df.describe()
```

Out[5]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	p
count	299.000000	299.000000	299.000000	299.000000	299.000000	299.000000	299
mean	60.833893	0.431438	581.839465	0.418060	38.083612	0.351171	263358
std	11.894809	0.496107	970.287881	0.494067	11.834841	0.478136	97804
min	40.000000	0.000000	23.000000	0.000000	14.000000	0.000000	25100
25%	51.000000	0.000000	116.500000	0.000000	30.000000	0.000000	212500
50%	60.000000	0.000000	250.000000	0.000000	38.000000	0.000000	262000
75%	70.000000	1.000000	582.000000	1.000000	45.000000	1.000000	303500
max	95.000000	1.000000	7861.000000	1.000000	80.000000	1.000000	850000

Apply Standard Scaler to entire dataframe - except target feature

Target Feature of 1 == Died

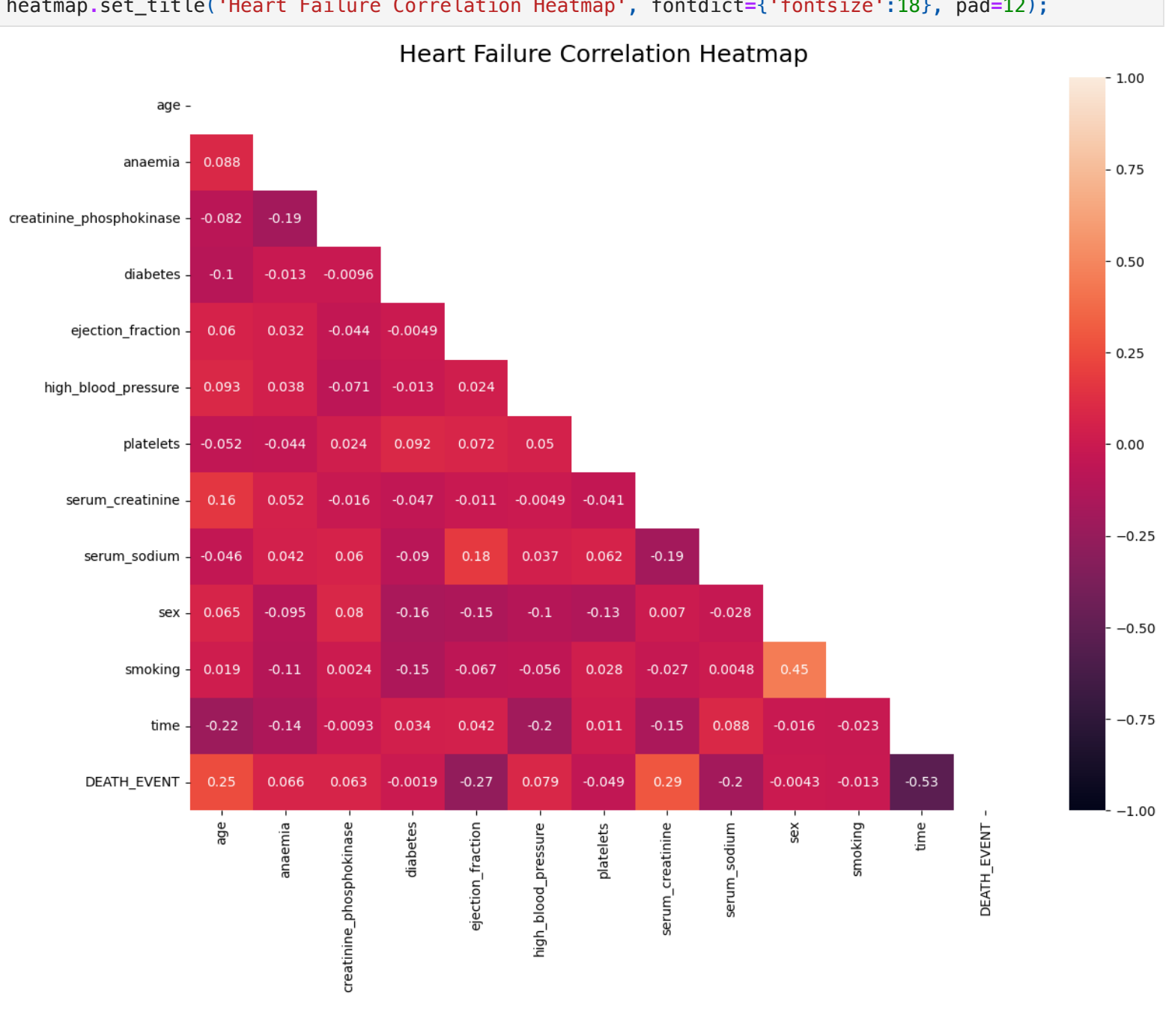
Target Feature of 0 == Lived

In [6]:
```
y_data = df.iloc[:,[12]]
print("Target Feature:")
print(y_data.columns)
print("\n")

ss = StandardScaler()
df = pd.DataFrame(ss.fit_transform(df),columns = df.columns)
df.head()


print("Independent Features:")
X_data = df.iloc[:,[0,1,2,3,4,5,6,7,8,9,10,11]]
print(X_data.columns)
```

```
Target Feature:
Index(['DEATH_EVENT'], dtype='object')


Independent Features:
Index(['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time'],
      dtype='object')
```

Correlation Heatmap

In [7]:
```
mask = np.triu(np.ones_like(df.corr(), dtype=bool))
plt.figure(figsize=(14, 10))
heatmap = sns.heatmap(df.corr(), mask=mask, vmin=-1, vmax=1, annot=True);
heatmap.set_title('Heart Failure Correlation Heatmap', fontdict={'fontsize':18}, pad=12);
```

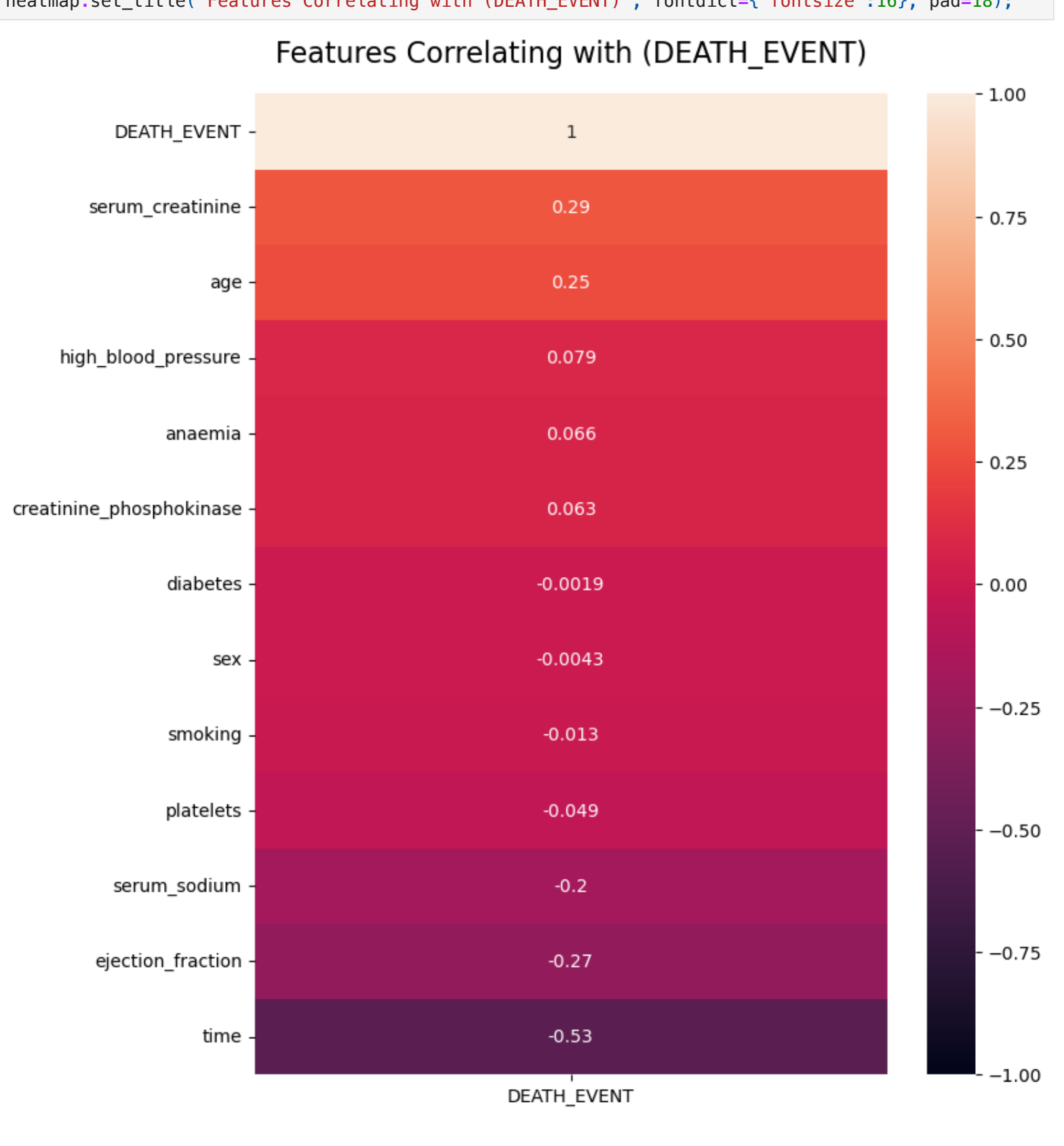


Correlation with Target (DEATH_EVENT)

In [8]:
```
df.corrwith(df['DEATH_EVENT'])
```

Out[8]:
```
age                         0.253729
anaemia                     0.066270
creatinine_phosphokinase    0.062728
diabetes                   -0.001943
ejection_fraction          -0.268603
high_blood_pressure         0.079351
platelets                  -0.049139
serum_creatinine            0.294278
serum_sodium               -0.195204
sex                        -0.004316
smoking                    -0.012623
time                       -0.526964
DEATH_EVENT                 1.000000
dtype: float64
```

In [9]:
```
plt.figure(figsize=(8, 10))
heatmap = sns.heatmap(df.corr()[['DEATH_EVENT']].sort_values(by='DEATH_EVENT', ascending=False),
                      vmin=-1, vmax=1, annot=True)
heatmap.set_title('Features Correlating with (DEATH_EVENT)', fontdict={'fontsize':16}, pad=18);
```



Part 2 - Machine Learning Models

In [10]:
```
#
# Train/Test Split - 30% for test
#

X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.3, random_state=1)

y_train = y_train.values
y_train = y_train.reshape(1, -1)
y_train = y_train.flatten()

y_test = y_test.values
y_test = y_test.reshape(1, -1)
y_test = y_test.flatten()
```

Logistic Regression - Grid Search

In [11]:
```
parameters = [
    {
     'solver': ['lbfgs'],
     'penalty': ['l2', 'none']
    },
    {
     'solver': ['liblinear'],
     'penalty': ['l1', 'l2'],
    },
    {
     'solver': ['newton-cg'],
     'penalty': ['l2', 'none'],
    },
    {
     'solver': ['sag'],
     'penalty': ['l2', 'none'],
    },
    {
     'solver': ['saga'],
     'penalty': ['l1', 'l2', 'none'],
    },
    {
     'solver': ['saga'],
     'penalty': ['elasticnet'],
     'l1_ratio': [1]
    }

    #l1_ratio=1 'l1',
]

lr_classifier = GridSearchCV(estimator=LogisticRegression(random_state=0), param_grid=parameters, r
                    return_train_score=True, n_jobs=-1)

# Fit the model for grid search
lr_classifier.fit(X_train, y_train)

# Show best parameters
print("\nBest Parameters:")
print(lr_classifier.best_params_)
print()

predict = lr_classifier.predict(X_test)

accuracy_lr_classifier = np.mean(predict == y_test)

print("Logistic Regression Accuracy: " + str(accuracy_lr_classifier))
print()
print("ROC-AUC Score: " + str(roc_auc_score(y_test, predict, average=None)))
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits

Best Parameters:
{'penalty': 'l2', 'solver': 'lbfgs'}

Logistic Regression Accuracy: 0.8333333333333334

ROC-AUC Score: 0.7686298076923077
```

Decision Tree Classifier - Grid Search

In [12]:
```
parameters = [
    {'criterion': ['gini', 'entropy', 'log_loss'],
     'splitter': ['best', 'random'],
     'class_weight': [{0:8,1:2}, 'balanced']
    }
]

dtree_classifier = GridSearchCV(estimator=DecisionTreeClassifier(random_state=0), param_grid=parame
                    verbose=1, return_train_score=True, n_jobs=-1)

# Fit the model for grid search
dtree_classifier.fit(X_train, y_train)

# Show best parameters
print("\nBest Parameters:")
print(dtree_classifier.best_params_)
print()

# Run prediction with best parameters
predict = dtree_classifier.predict(X_test)
accuracy_dtree_classifier = np.mean(predict == y_test)

print("Decision Tree Accuracy: " + str(accuracy_dtree_classifier))
print()
print("ROC-AUC Score: " + str(roc_auc_score(y_test, predict, average=None)))
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits

Best Parameters:
{'class_weight': {0: 8, 1: 2}, 'criterion': 'entropy', 'splitter': 'best'}

Decision Tree Accuracy: 0.8333333333333334

ROC-AUC Score: 0.7914663461538461
```

Random Forest - Grid Search

In [13]:
```
parameters = [
    {'n_estimators': [10, 100, 200, 500],
     'criterion': ['gini', 'entropy', 'log_loss'],
     'bootstrap': [True, False],
     'class_weight': [{0:8,1:2}, 'balanced', 'balanced_subsample']
    }
]

rf_classifier = GridSearchCV(estimator=RandomForestClassifier(random_state=0), param_grid=parameter
                    refit=True, verbose=1, return_train_score=True, n_jobs=-1)

# Fit the model for grid search
rf_classifier.fit(X_train, y_train)

# Show best parameters
print("\nBest Parameters:")
print(rf_classifier.best_params_)
print()

# Run prediction with best parameters
predict = rf_classifier.predict(X_test)
accuracy_rf_classifier = np.mean(predict == y_test)

print("Decision Tree Accuracy: " + str(accuracy_rf_classifier))
print()
print("ROC-AUC Score: " + str(roc_auc_score(y_test, predict, average=None)))
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits

Best Parameters:
{'bootstrap': True, 'class_weight': 'balanced_subsample', 'criterion': 'gini', 'n_estimators': 10
0}

Decision Tree Accuracy: 0.8888888888888888

ROC-AUC Score: 0.8419471153846154
```

XGBoost - Grid Search

In [14]:
```
parameters = [
    {
        'n_estimators': [5, 10, 15, 20, 50, 100, 200, 500],
        'booster': ['gbtree', 'gblinear', 'dart'],
        'eval_metric': ['mlogloss', 'auc', 'rmse'],
        'learning_rate': [0.5, 0.1, 0.05, 0.01],
    }
]

xgb_classifier = GridSearchCV(estimator=XGBClassifier(random_state=0), param_grid=parameters, refit
                    return_train_score=True, n_jobs=-1)

# Fit the model for grid search
xgb_classifier.fit(X_train, y_train)

# Show best parameters
print("\nBest Parameters:")
print(xgb_classifier.best_params_)
print()

# Run prediction with best parameters
predict = xgb_classifier.predict(X_test)
accuracy_xgb_classifier = np.mean(predict == y_test)

print("Decision Tree Accuracy: " + str(accuracy_xgb_classifier))
print()
print("ROC-AUC Score: " + str(roc_auc_score(y_test, predict, average=None)))
```

```
Fitting 5 folds for each of 288 candidates, totalling 1440 fits

Best Parameters:
{'booster': 'gbtree', 'eval_metric': 'mlogloss', 'learning_rate': 0.05, 'n_estimators': 200}

Decision Tree Accuracy: 0.8888888888888888

ROC-AUC Score: 0.8647836538461539
```

Part 3A

In [15]:
```
#
# Fine one postive row (death) and one negative row (lived) in X_test data
#

y_1 = 0
y_0 = 0
X_sample_1 = None
X_sample_0 = None

for i in range(y_test.size):
    if y_test[i] == 1:
        X_sample_1 = X_test.iloc[i,:]
        y_1 = i
        break

for i in range(y_test.size):
    if y_test[i] == 0:
        X_sample_0 = X_test.iloc[i,:]
        y_0 = i
        break

print("Samle Rows:\n")
print("For y = 0:")
print(X_sample_0)
print("y=" + str(y_test[y_0]))
print()
print("For y = 1:")
print(X_sample_1)
print("y=" + str(y_test[y_1]))
```

```
Samle Rows:

For y = 0:
age                        -0.912335
anaemia                     1.147968
creatinine_phosphokinase   -0.481942
diabetes                   -0.847579
ejection_fraction          -1.530560
high_blood_pressure        -0.735688
platelets                  -0.761549
serum_creatinine           -0.575031
serum_sodium                0.539054
sex                         0.735688
smoking                    -0.687682
time                        0.203127
Name: 173, dtype: float64
y=0

For y = 1:
age                        -0.659702
anaemia                     1.147968
creatinine_phosphokinase   -0.506718
diabetes                   -0.847579
ejection_fraction          -1.530560
high_blood_pressure         1.359272
platelets                   1.583788
serum_creatinine            0.005926
serum_sodium                0.539054
sex                        -1.359272
smoking                    -0.687682
time                       -1.126175
Name: 51, dtype: float64
y=1
```

In [16]:
```
lr_clf = LogisticRegression(penalty='l2', solver='lbfgs', random_state=0)
lr_clf.fit(X_train, y_train)

eli5.explain_weights(lr_clf)
```

Out[16]: **y=1** top features

Weight?	Feature
+0.688	x7
+0.462	x0
+0.205	x2
+0.128	x3
+0.002	x1
-0.037	x10
-0.116	x5
-0.222	x9
-0.282	x8
-0.365	x6
-0.747	x4
-1.251	<BIAS>
-1.599	x11

For y = 0 (lived)

In [17]:
```
eli5.explain_prediction(lr_clf, X_sample_0)
```

Out[17]: **y=0** (probability **0.799**, score **-1.382**) top features

Contribution?	Feature
+1.251	<BIAS>
+0.421	age
+0.396	serum_creatinine
+0.325	time
+0.163	sex
+0.152	serum_sodium
+0.109	diabetes
+0.099	creatinine_phosphokinase
-0.002	anaemia
-0.026	smoking
-0.085	high_blood_pressure
-0.278	platelets
-1.143	ejection_fraction

In [18]:
```
eli5.show_prediction(lr_clf, X_sample_0, show_feature_values=True)
```

Out[18]: **y=0** (probability **0.799**, score **-1.382**) top features

Contribution?	Feature	Value
+1.251	<BIAS>	1.000
+0.421	age	-0.912
+0.396	serum_creatinine	-0.575
+0.325	time	0.203
+0.163	sex	0.736
+0.152	serum_sodium	0.539
+0.109	diabetes	-0.848
+0.099	creatinine_phosphokinase	-0.482
-0.002	anaemia	1.148
-0.026	smoking	-0.688
-0.085	high_blood_pressure	-0.736
-0.278	platelets	-0.762
-1.143	ejection_fraction	-1.531

For y = 1 (died)

In [19]:
```
eli5.explain_prediction(lr_clf, X_sample_1)
```

Out[19]: **y=1** (probability **0.651**, score **0.623**) top features

Contribution?	Feature
+1.801	time
+1.143	ejection_fraction
+0.302	sex
+0.026	smoking
+0.004	serum_creatinine
+0.002	anaemia
-0.104	creatinine_phosphokinase
-0.109	diabetes
-0.152	serum_sodium
-0.157	high_blood_pressure
-0.305	age
-0.577	platelets
-1.251	<BIAS>

In [20]:
```
eli5.show_prediction(lr_clf, X_sample_1, show_feature_values=True)
```

Out[20]: **y=1** (probability **0.651**, score **0.623**) top features

Contribution?	Feature	Value
+1.801	time	-1.126
+1.143	ejection_fraction	-1.531
+0.302	sex	-1.359
+0.026	smoking	-0.688
+0.004	serum_creatinine	0.006
+0.002	anaemia	1.148
-0.104	creatinine_phosphokinase	-0.507
-0.109	diabetes	-0.848
-0.152	serum_sodium	0.539
-0.157	high_blood_pressure	1.359
-0.305	age	-0.660
-0.577	platelets	1.584
-1.251	<BIAS>	1.000

Permutation Importance

In [21]:
```
perm = PermutationImportance(lr_clf, scoring="balanced_accuracy")
perm.fit(X_test, y_test)
eli5.show_weights(perm)
```

Out[21]:

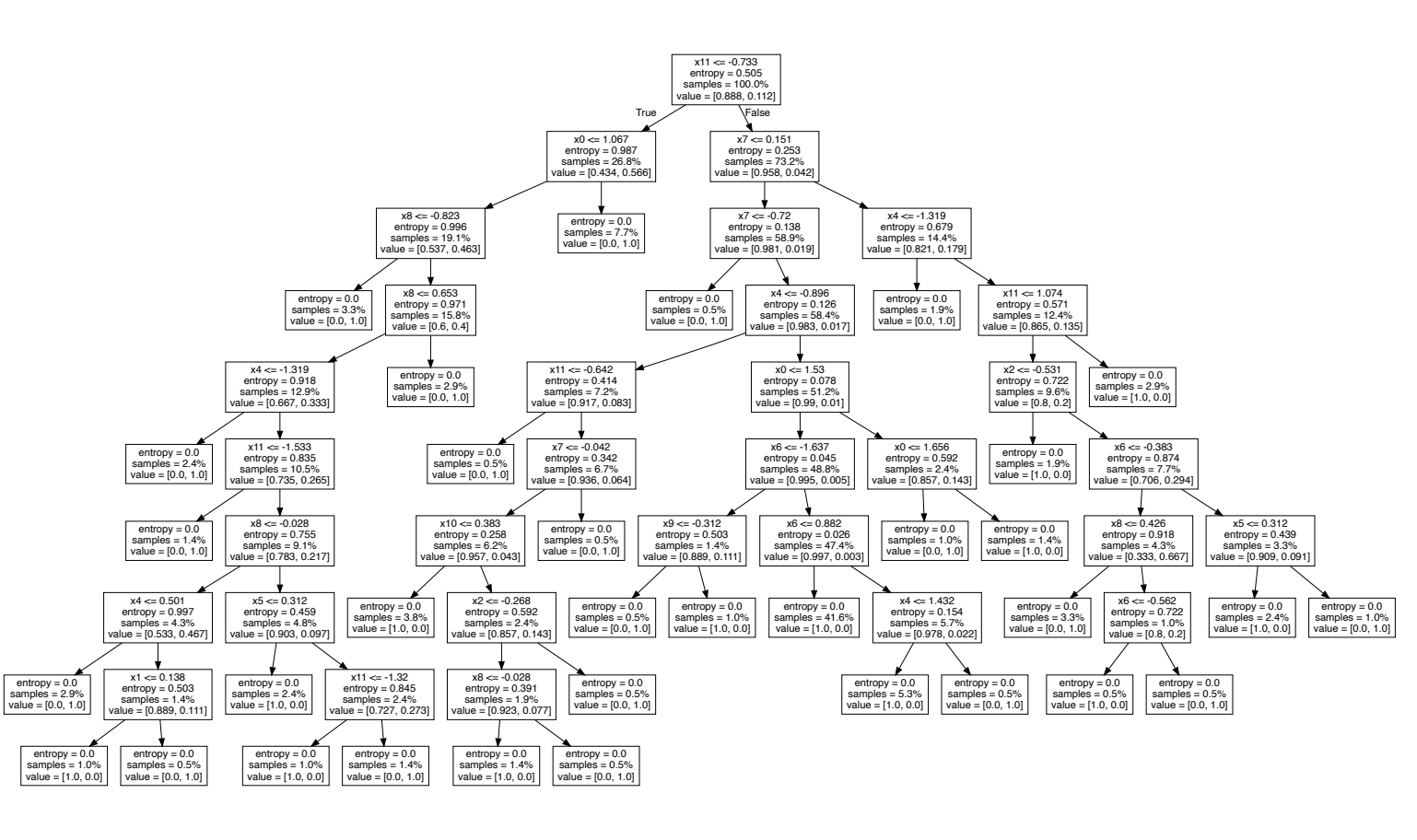
Weight	Feature
0.2388 ± 0.1039	x11
0.0502 ± 0.0269	x4
0.0363 ± 0.0514	x0
0.0179 ± 0.0401	x8
0.0047 ± 0.0273	x7
0.0031 ± 0.0077	x10
0 ± 0.0000	x1
-0.0077 ± 0.0213	x9
-0.0100 ± 0.0173	x2
-0.0168 ± 0.0216	x5
-0.0177 ± 0.0157	x6
-0.0185 ± 0.0198	x3

Part 3B - Decision Tree

In [22]:
```
dt_clf = DecisionTreeClassifier(class_weight={0: 8, 1: 2}, criterion='entropy', splitter='best', ra
dt_clf.fit(X_train, y_train)

eli5.explain_weights(dt_clf)
```

Out[22]:

Weight	Feature
0.3870	x11
0.1347	x4
0.1113	x8
0.0959	x7
0.0935	x0
0.0612	x6
0.0461	x5
0.0303	x2
0.0143	x9
0.0143	x1
0.0113	x10
0	x3



For y=0...

In [23]:
```
eli5.explain_prediction(dt_clf, X_sample_0)
```

Out[23]: **y=0** (probability **1.000**) top features

Contribution?	Feature
+0.888	<BIAS>
+0.089	time
+0.045	serum_creatinine
+0.043	smoking
-0.066	ejection_fraction

In [24]:
```
eli5.show_prediction(dt_clf, X_sample_0, show_feature_values=True)
```

Out[24]: **y=0** (probability **1.000**) top features

Contribution?	Feature	Value
+0.888	<BIAS>	1.000
+0.089	time	0.203
+0.045	serum_creatinine	-0.575
+0.043	smoking	-0.688
-0.066	ejection_fraction	-1.531

For y=1...

In [25]:
```
eli5.explain_prediction(dt_clf, X_sample_1)
```

Out[25]: **y=1** (probability **1.000**) top features

Contribution?	Feature
+0.667	ejection_fraction
+0.454	time
+0.112	<BIAS>
-0.104	age
-0.129	serum_sodium

In [26]:
```
eli5.show_prediction(dt_clf, X_sample_1, show_feature_values=True)
```

Out[26]: **y=1** (probability **1.000**) top features

Contribution?	Feature	Value
+0.667	ejection_fraction	-1.531
+0.454	time	-1.126
+0.112	<BIAS>	1.000
-0.104	age	-0.660
-0.129	serum_sodium	0.539

In [27]:
```
perm = PermutationImportance(dt_clf, scoring="balanced_accuracy")
perm.fit(X_test, y_test)
eli5.show_weights(perm)
```

Out[27]:

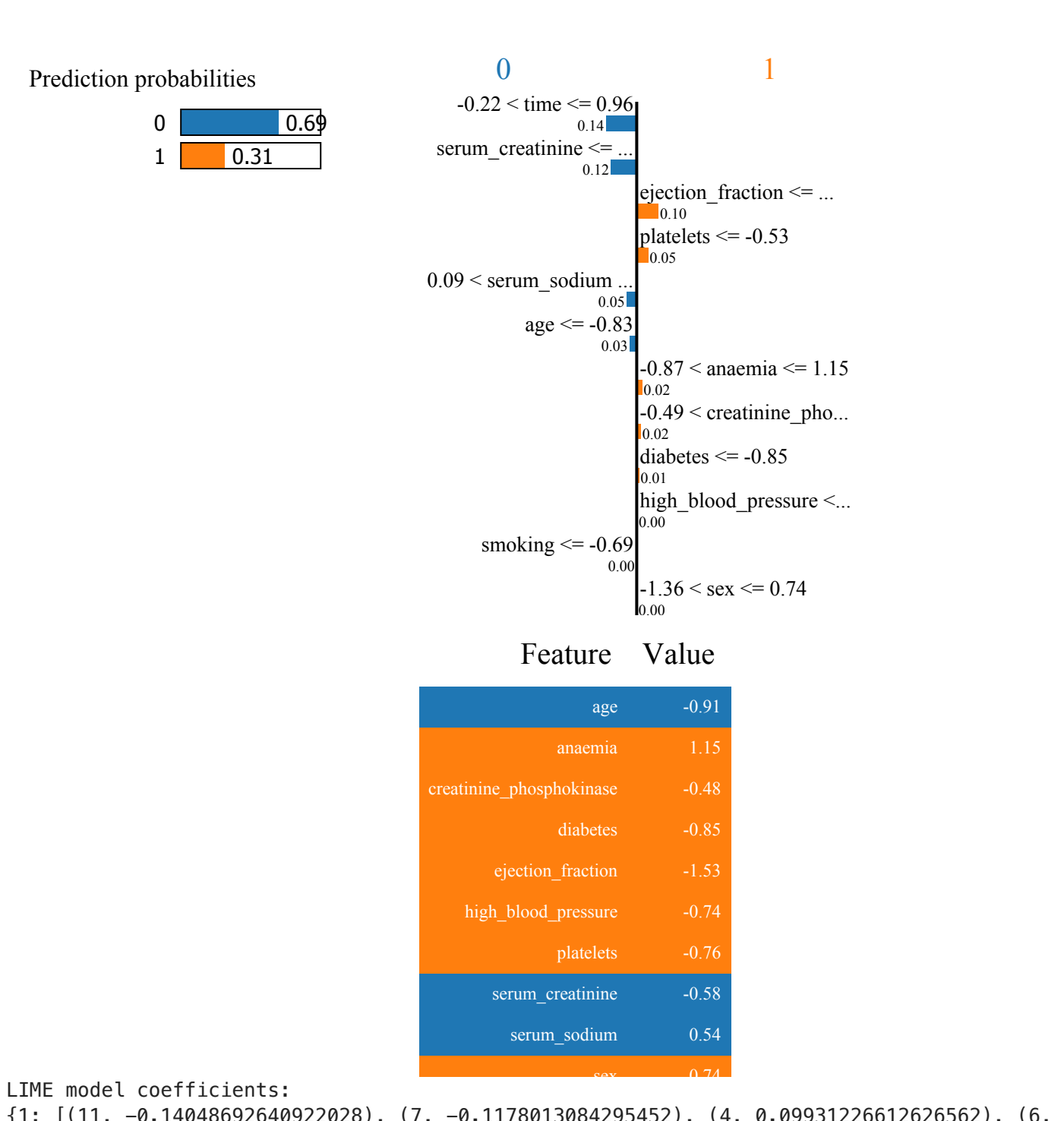
Weight	Feature
0.1672 ± 0.0654	x11
0.0493 ± 0.0491	x7
0.0464 ± 0.0443	x5
0.0317 ± 0.0557	x8
0.0240 ± 0.0543	x4
0.0040 ± 0.0183	x0
0.0038 ± 0.0154	x2
0 ± 0.0000	x9
0 ± 0.0000	x3
-0.0006 ± 0.0490	x6
-0.0077 ± 0.0188	x1
-0.0108 ± 0.0215	x10

Part 3C - Random Forest

In [28]:
```
rf_classifier = RandomForestClassifier(bootstrap=True, class_weight='balanced_subsample', criterion

#
# Convert this to "values" only to avoid warning with LIME !
#
rf_classifier.fit(X_train.values, y_train)

explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values, mode="classification", feature_r
```

For the *DEATH_EVENT = 0* instance

In [29]:
```
#
# Explain the DEATH_EVENT = 0 instance
#

explanation = explainer.explain_instance(X_sample_0, rf_classifier.predict_proba, num_features=12)
explanation.show_in_notebook(show_all=True)

print("LIME model coefficients:\n" + str(explanation.local_exp))
print()
print("LIME model intercept:" + str(explanation.intercept))
print("LIME model R2 score: " + str(explanation.score))
```

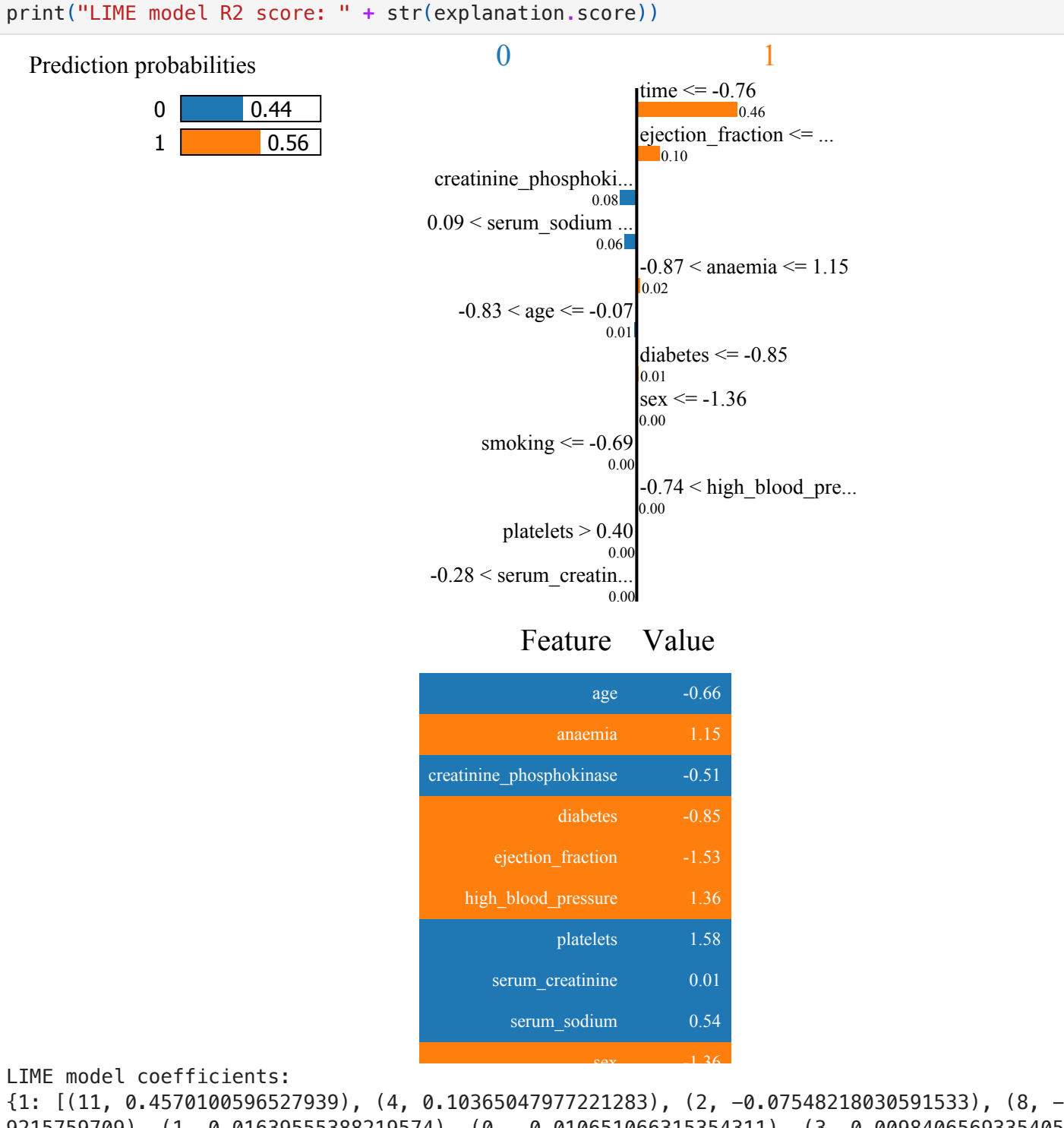



```
LIME model coefficients:
{1: [(11, -0.14048692640922028), (7, -0.1178013084295452), (4, 0.09931226612626562), (6, 0.0523259
71201711015), (8, -0.04714517620576748), (0, -0.03335973186981707), (1, 0.02459997522978538), (2,
0.018494894377077735), (3, 0.010482891131236856), (5, 0.0023478522646774276), (10, -0.001916155234
1074857), (9, 0.0014423363912904627)]}

LIME model intercept:{1: 0.36430457421174234}
LIME model R2 score: 0.16458880215821514
```

^^ VERY Low R^2 Score

For the *DEATH_EVENT = 1* instance

In [30]:
```python
explanation = explainer.explain_instance(X_sample_1, rf_classifier.predict_proba, num_features=12)
explanation.show_in_notebook(show_all=True)

print("LIME model coefficients:\n" + str(explanation.local_exp))
print()
print("LIME model intercept:" + str(explanation.intercept))
print("LIME model R2 score: " + str(explanation.score))
```



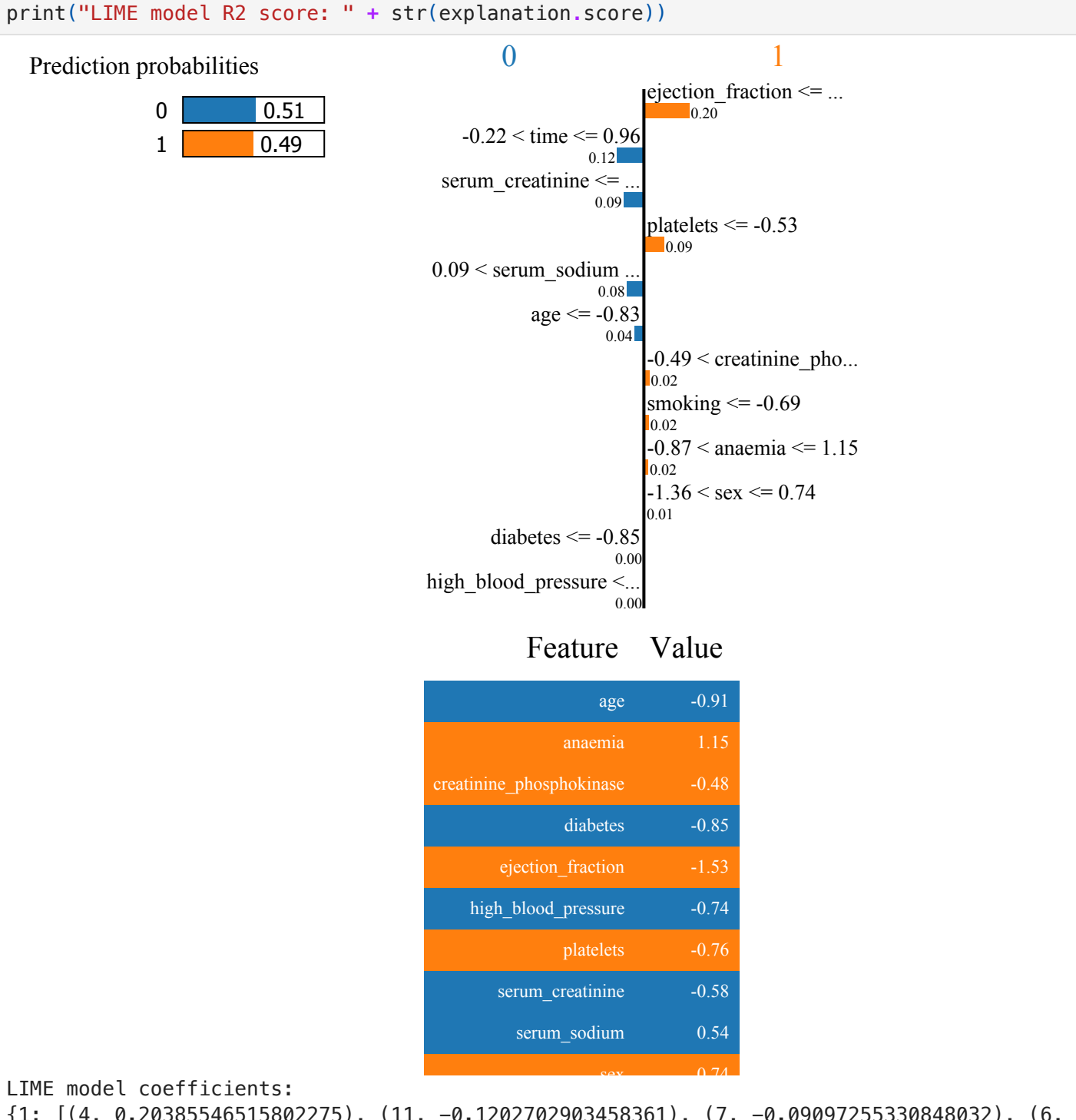

```
LIME model coefficients:
{1: [(11, 0.4570100596527939), (4, 0.10365047977221283), (2, -0.07548218030591533), (8, -0.0561427
9215759709), (1, 0.01639555388219574), (0, -0.010651066315354311), (3, 0.0098406569335405), (9, 0.
003826318825577311), (10, -0.0016175989520438975), (5, 0.001551054986766725), (6, -0.0008035820809
456576), (7, -7.543159703485565e-05)]}

LIME model intercept:{1: 0.22011357558650296}
LIME model R2 score: 0.711663531493109
```

...for XBG

In [31]:
```python
xgb_classifier = XGBClassifier(booster='gbtree', eval_metric='mlogloss', learning_rate=0.05, n_esti

#
# Convert this to "values" only to avoid warning with LIME !
#
xgb_classifier.fit(X_train.values, y_train)

explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values, mode="classification", feature_r
```

For the *DEATH_EVENT = 0* instance

In [32]:
```python
#
# Explain the DEATH_EVENT = 0 instance
#

explanation = explainer.explain_instance(X_sample_0, xgb_classifier.predict_proba, num_features=12)
explanation.show_in_notebook(show_all=True)

print("LIME model coefficients:\n" + str(explanation.local_exp))
print()
print("LIME model intercept:" + str(explanation.intercept))
print("LIME model R2 score: " + str(explanation.score))
```

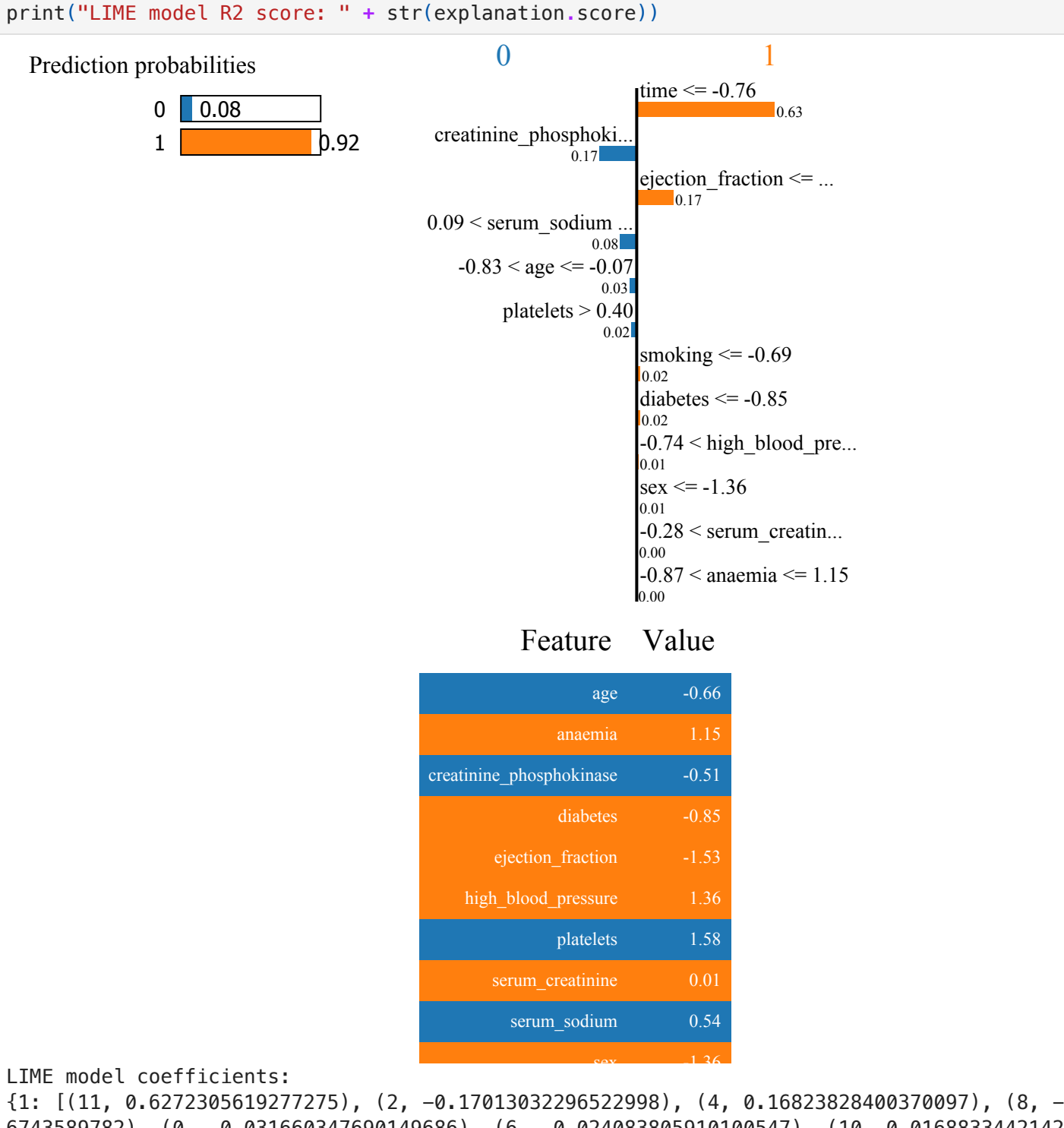



```
LIME model coefficients:
{1: [(4, 0.20385546515802275), (11, -0.1202702903458361), (7, -0.09097255330848032), (6, 0.0908183
2580667888), (8, -0.07770583847340379), (0, -0.04316019858523787), (2, 0.024387104558295025), (10,
0.022107435542976273), (1, 0.019458316370638506), (9, 0.005803862136943835), (3, -0.00127080995132
91434), (5, -0.0010266467495306675)]}

LIME model intercept:{1: 0.29696951474665134}
LIME model R2 score: 0.12964761261436253
```

^^ VERY Low R^2 Score

For the *DEATH_EVENT = 1* instance

In [33]:
```python
explanation = explainer.explain_instance(X_sample_1, xgb_classifier.predict_proba, num_features=12)
explanation.show_in_notebook(show_all=True)

print("LIME model coefficients:\n" + str(explanation.local_exp))
print()
print("LIME model intercept:" + str(explanation.intercept))
print("LIME model R2 score: " + str(explanation.score))
```




```
LIME model coefficients:
{1: [(11, 0.6272305619277275), (2, -0.17013032296522998), (4, 0.16823828400370097), (8, -0.0756437
6743589782), (0, -0.031660347690149686), (6, -0.024083805910100547), (10, 0.01688334421423432),
(3, 0.01579732504660968), (5, 0.006955370958375614), (9, 0.00568297198470326), (7, 0.0036850174256
08867), (1, 0.0006335121259776529)]}

LIME model intercept:{1: 0.18044836609501524}
LIME model R2 score: 0.6710157417485589
```

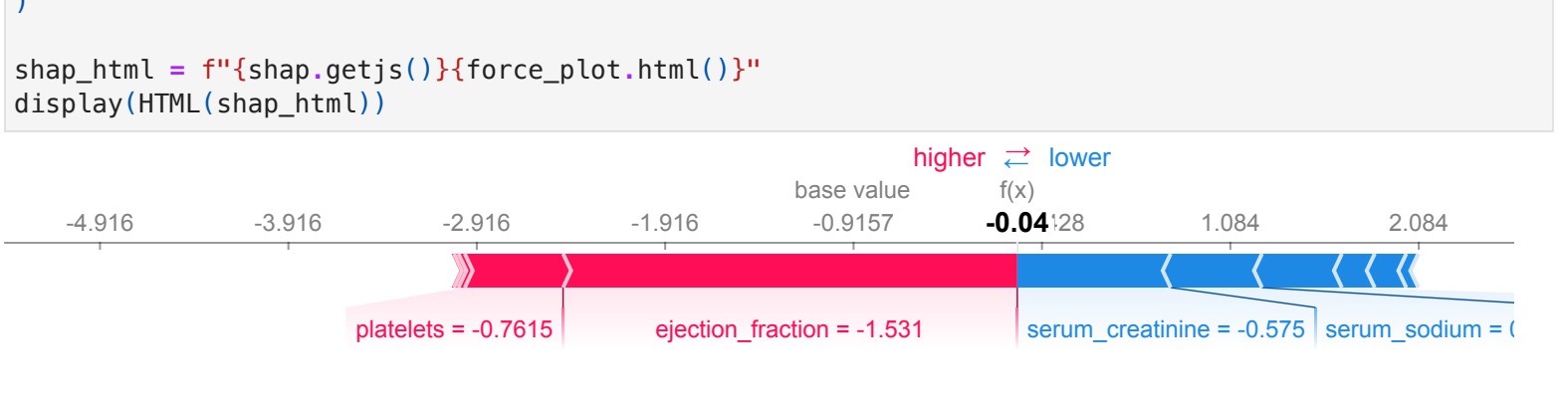
Part 3D

In [34]:
```python
import warnings
warnings.filterwarnings('ignore')

#
# Need to convert to numpy array now for some reason !!!
#
X_train_values = X_train.values
X_test_values = X_test.values

xgb_clf = XGBClassifier(booster='gbtree', eval_metric='mlogloss', learning_rate=0.05, n_estimators=

# Fit the model for grid search
xgb_clf.fit(X_train_values, y_train);
```

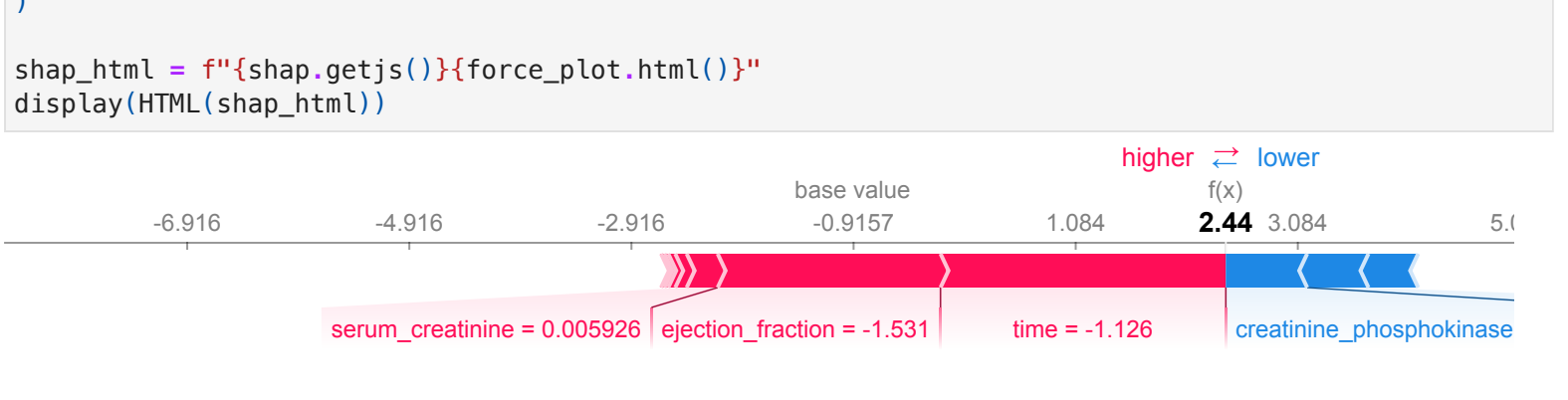
In [35]:
```python
#
# Need to reshape the one row sample instances into 2-dimensional
# otherwise it WILL FAIL - see note in previous cell
#

x_samples_0 = X_sample_0.values
x_samples_0 = np.reshape(x_samples_0, (1, 12))
print(x_samples_0)

x_samples_1 = X_sample_1.values
x_samples_1 = np.reshape(x_samples_1, (1, 12))
print(x_samples_1)
```

```
[[-0.9123354   1.14796753 -0.4819416  -0.84757938 -1.53055953 -0.73568819
  -0.76154867 -0.57503085  0.53905383  0.73568819 -0.68768191  0.20312668]]
[[-0.65970173  1.14796753 -0.506718   -0.84757938 -1.53055953  1.35927151
   1.58378844  0.00592615  0.53905383 -1.35927151 -0.68768191 -1.12617471]]
```

In [36]:
```python
pred = xgb_clf.predict(x_samples_0) #, output_margin=True)
explainer = shap.TreeExplainer(xgb_clf)
shap_value = explainer.shap_values(x_samples_0)

#
# Terrible hack to get this to work with the module versions in place
#

force_plot = shap.force_plot(
    base_value=explainer.expected_value,
    shap_values=shap_value,
    features=X_sample_0
)

shap_html = f"{shap.getjs()}{force_plot.html()}"
display(HTML(shap_html))
```

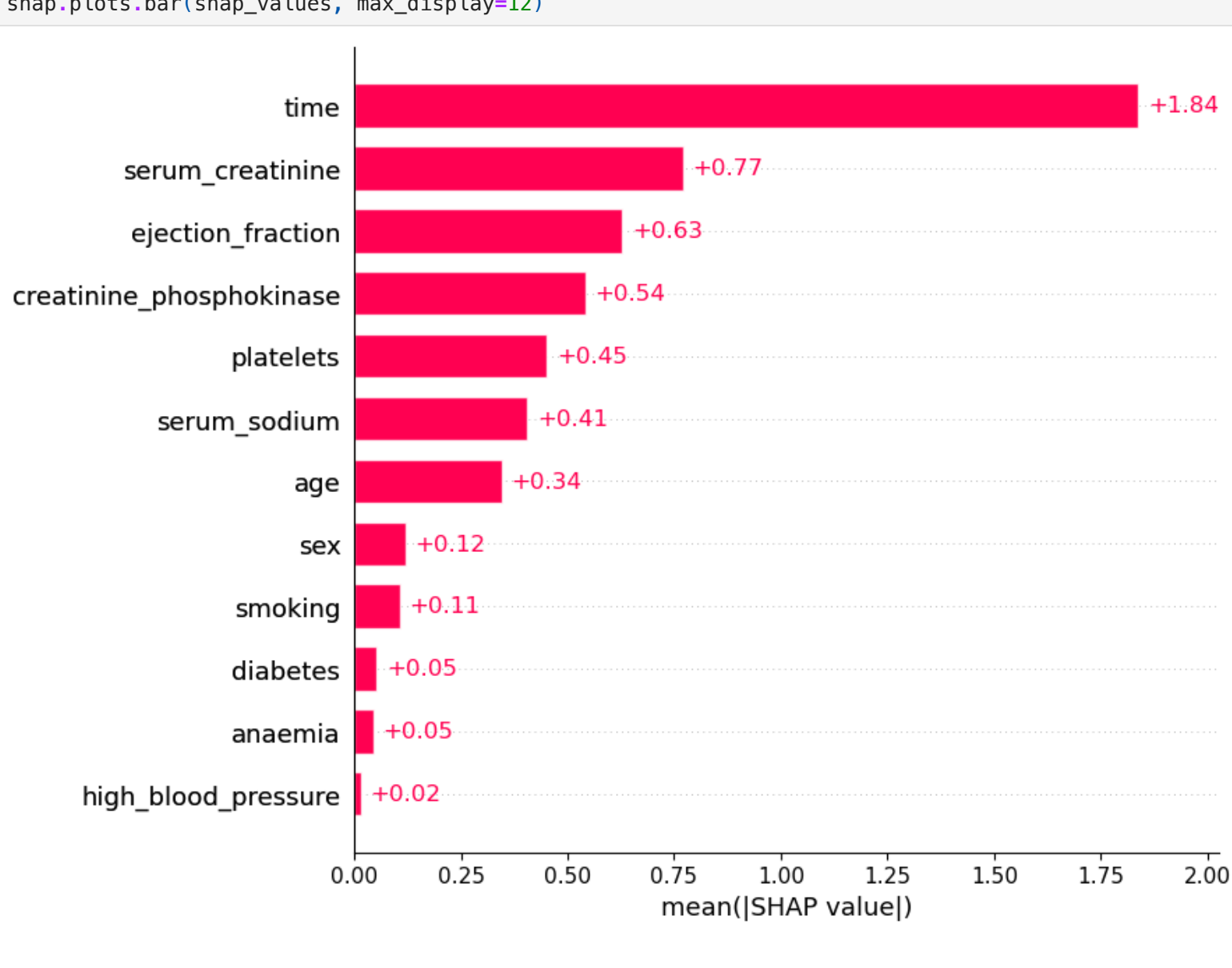


In [37]:
```python
pred = xgb_clf.predict(x_samples_1)
explainer = shap.TreeExplainer(xgb_clf)
shap_value = explainer.shap_values(x_samples_1)

force_plot = shap.force_plot(
    base_value=explainer.expected_value,
    shap_values=shap_value,
    features=X_sample_1
)

shap_html = f"{shap.getjs()}{force_plot.html()}"
display(HTML(shap_html))
```



Feature Importance Plot

In [38]:
```python
shap_values = explainer(X_test)

shap.plots.bar(shap_values, max_display=12)
```



Part 4 - Predict Observations

In [39]:
```python
arr = lr_clf.predict_proba(x_samples_0)
print("Logistic Regression probabilities for \"Lived\":")
print(arr)
print()
arr = lr_clf.predict_proba(x_samples_1)
print("Logistic Regression probabilities for \"Died\":")
print(arr)
```

```
Logistic Regression probabilities for "Lived":
[[0.79937398 0.20062602]]

Logistic Regression probabilities for "Died":
[[0.34908408 0.65091592]]
```

In [40]:
```python
arr = dt_clf.predict_proba(x_samples_0)
print("Decision Tree probabilities for \"Lived\":")
print(arr)
print()
arr = dt_clf.predict_proba(x_samples_1)
print("Decision Tree probabilities for \"Died\":")
print(arr)
```

```
Decision Tree probabilities for "Lived":
[[1. 0.]]

Decision Tree probabilities for "Died":
[[0. 1.]]
```

In [41]:
```python
arr = rf_classifier.predict_proba(x_samples_0)
print("Random Forest probabilities for \"Lived\":")
print(arr)
print()
arr = rf_classifier.predict_proba(x_samples_1)
print("Random Forest probabilities for \"Died\":")
print(arr)
```

```
Random Forest probabilities for "Lived":
[[0.69 0.31]]

Random Forest probabilities for "Died":
[[0.44 0.56]]
```

In [42]:
```python
arr = xgb_clf.predict_proba(x_samples_0)
print("Xgboost probabilities for \"Lived\":")
print(arr)
print()
arr = xgb_clf.predict_proba(x_samples_1)
print("Xgboost probabilities for \"Died\":")
print(arr)
```

```
Xgboost probabilities for "Lived":
[[0.5103786 0.4896214]]

Xgboost probabilities for "Died":
[[0.07982033 0.92017967]]
```

For the two test samples: Decision Tree Classifier has the highest probabilities:

For Lived: [1, 0] For Died: [0, 1]

In [ ]: