1 - Explanatory Data Analysis (EDA)

In [3]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
df = pd.read_csv("heart_failure_clinical_records_dataset.csv")
print(df.head())
```

```
    age  anaemia  creatinine_phosphokinase  diabetes  ejection_fraction
\
0  75.0        0                       582         0                 20
1  55.0        0                      7861         0                 38
2  65.0        0                       146         0                 20
3  50.0        1                       111         0                 20
4  65.0        1                       160         1                 20

   high_blood_pressure  platelets  serum_creatinine  serum_sodium  sex
\
0                    1  265000.00               1.9           130    1
1                    0  263358.03               1.1           136    1
2                    0  162000.00               1.3           129    1
3                    0  210000.00               1.9           137    1
4                    0  327000.00               2.7           116    0

   smoking  time  DEATH_EVENT
0        0     4            1
1        0     6            1
2        1     7            1
3        0     7            1
4        0     8            1
```

In [50]:
```
df.info()
```

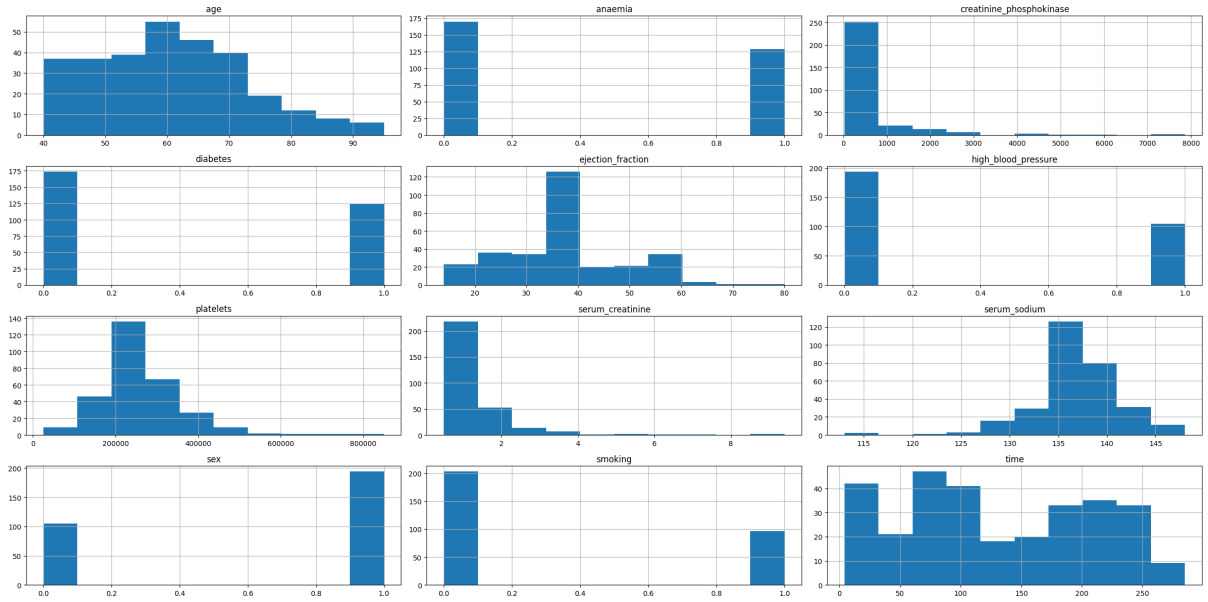
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   age                       299 non-null    float64
 1   anaemia                   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 [42]:
```
df.describe()
```

Out[42]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blooc
count	299.000000	299.000000	299.000000	299.000000	299.000000	2
mean	60.833893	0.431438	581.839465	0.418060	38.083612	
std	11.894809	0.496107	970.287881	0.494067	11.834841	
min	40.000000	0.000000	23.000000	0.000000	14.000000	
25%	51.000000	0.000000	116.500000	0.000000	30.000000	
50%	60.000000	0.000000	250.000000	0.000000	38.000000	
75%	70.000000	1.000000	582.000000	1.000000	45.000000	
max	95.000000	1.000000	7861.000000	1.000000	80.000000	

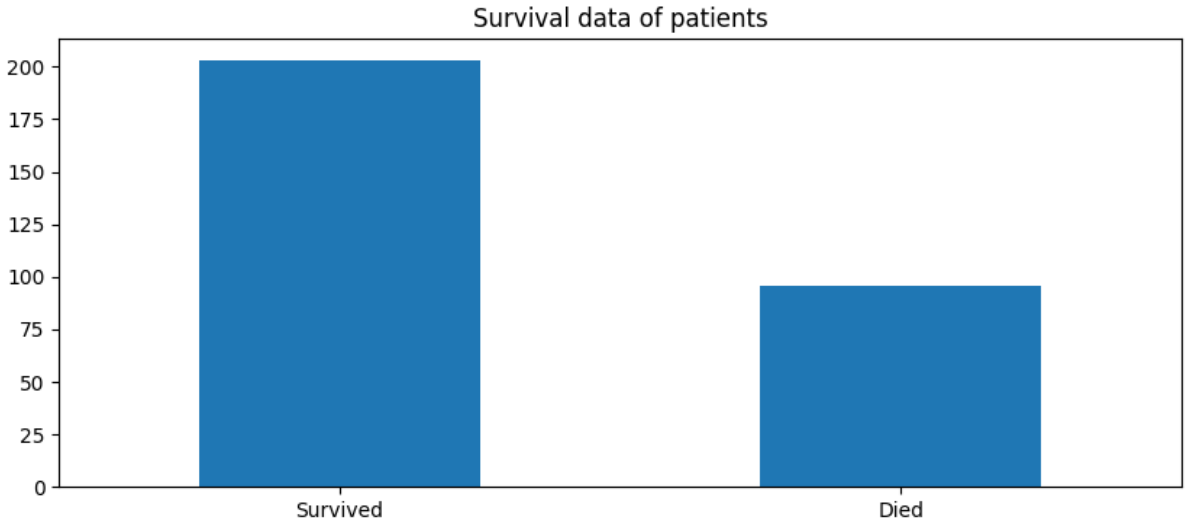
In [44]:
```
df.drop(columns=["DEATH_EVENT"]).hist(figsize=(20, 10))
plt.tight_layout(rect=(0, 0, 1.2, 1.2))
```



Visualizing target: deaths and survivals

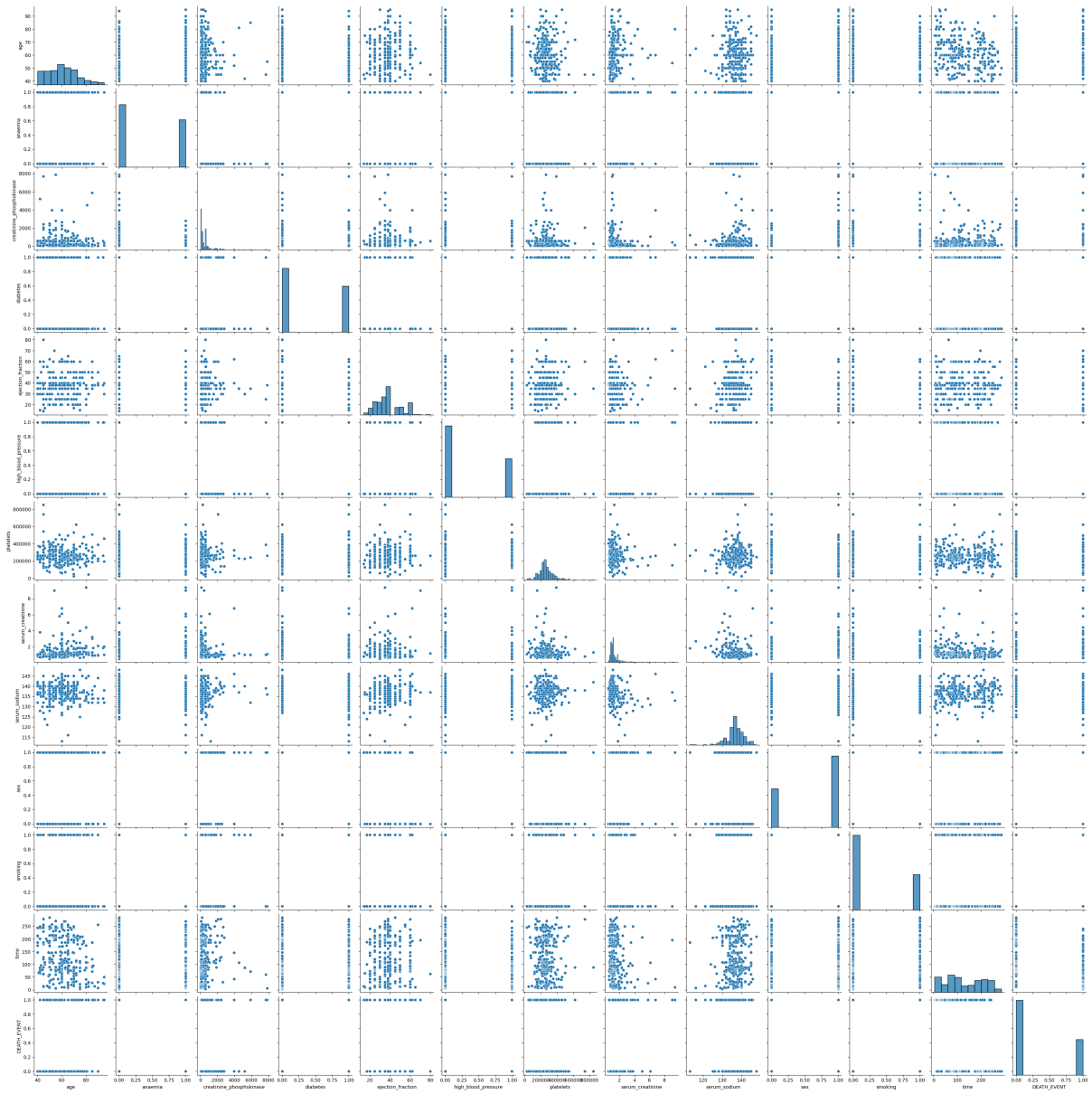
In [66]:
```
fig, ax = plt.subplots(figsize = (10,4))
df.DEATH_EVENT.value_counts().plot(kind="bar")
plt.title("Survival data of patients")
plt.xticks(rotation=0)
ax.set_xticklabels(["Survived", "Died"])
```

Out[66]:
```
[Text(0, 0, 'Survived'), Text(1, 0, 'Died')]
```



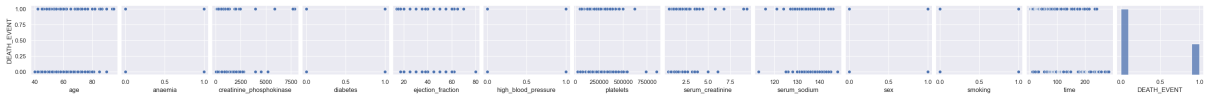
In [47]:
```
import seaborn as sns
sns.pairplot(df)
```

Out[47]: <seaborn.axisgrid.PairGrid at 0x14a6a9a20>



In [83]:
```
sns.pairplot(df, y_vars=['DEATH_EVENT'])
```

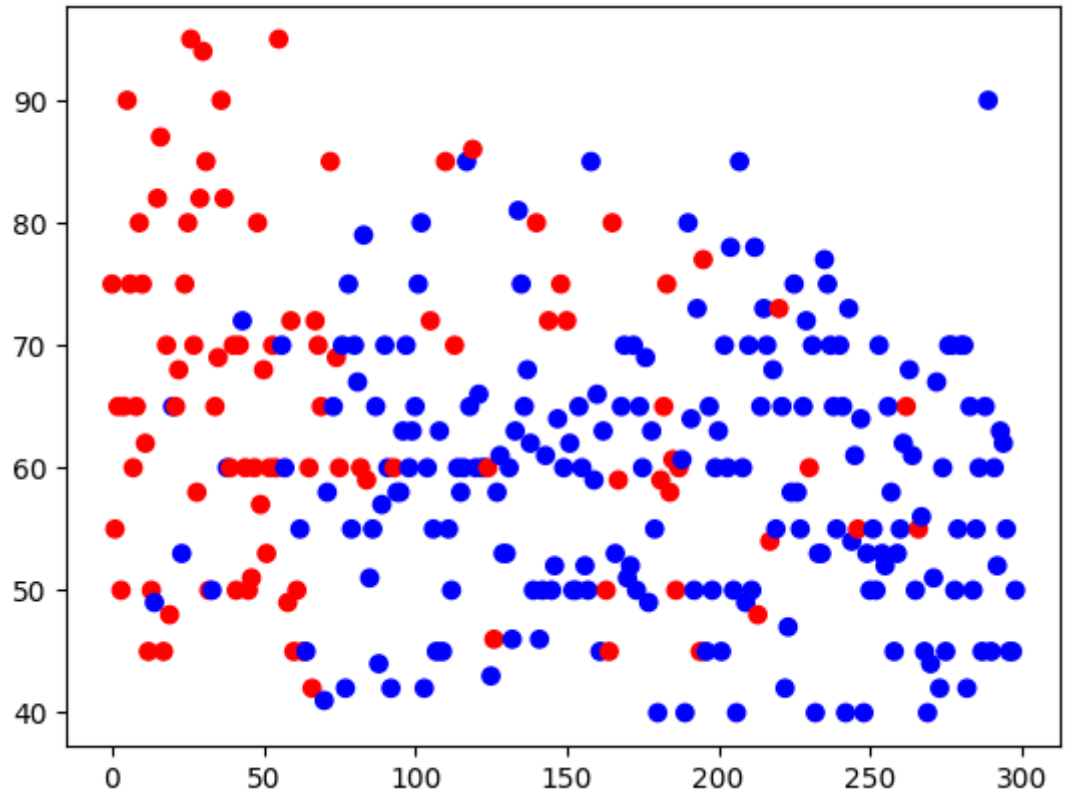
Out[83]: <seaborn.axisgrid.PairGrid at 0x2fcac9bd0>



In [5]:
```
color = []
for num in df['DEATH_EVENT']:
    if num == 0:
        color.append('b')
    else:
        color.append('r')
```

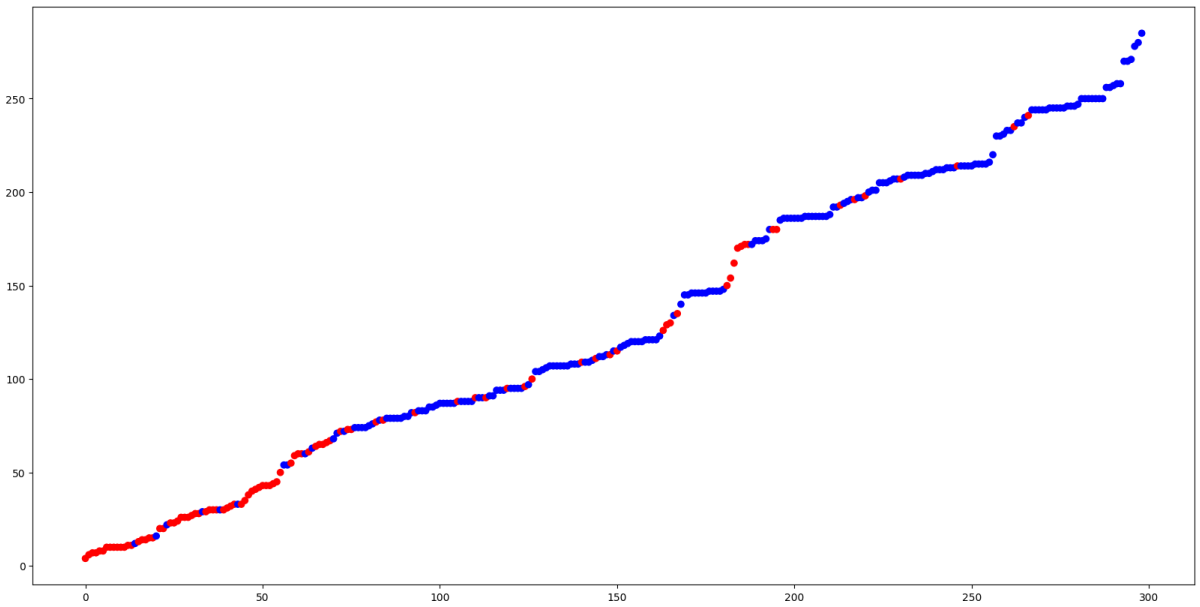
In [14]:
```
# plt.figure(figsize=(10, 20))
plt.scatter(y=df.age, x=np.arange(299), c=color)
```

Out[14]: <matplotlib.collections.PathCollection at 0x13cceb6d0>



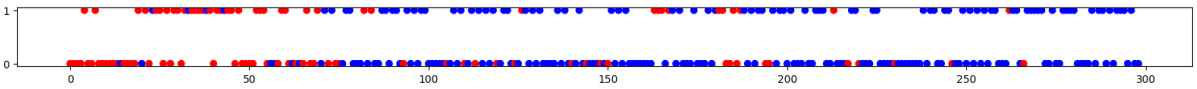
In [16]:
```
plt.figure(figsize=(20, 10))
plt.scatter(y=df.time, x=np.arange(299), c=color)
```

Out[16]: <matplotlib.collections.PathCollection at 0x13f125240>



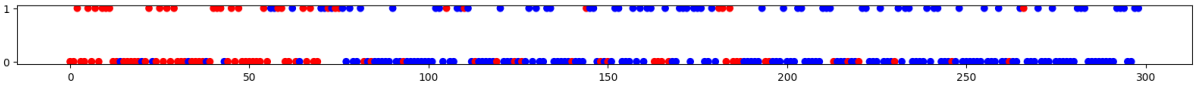
In [36]:
```
plt.figure(figsize=(20, 1))
plt.scatter(y=df.diabetes, x=np.arange(299), c=color)
```

Out[36]: <matplotlib.collections.PathCollection at 0x14a5e1750>



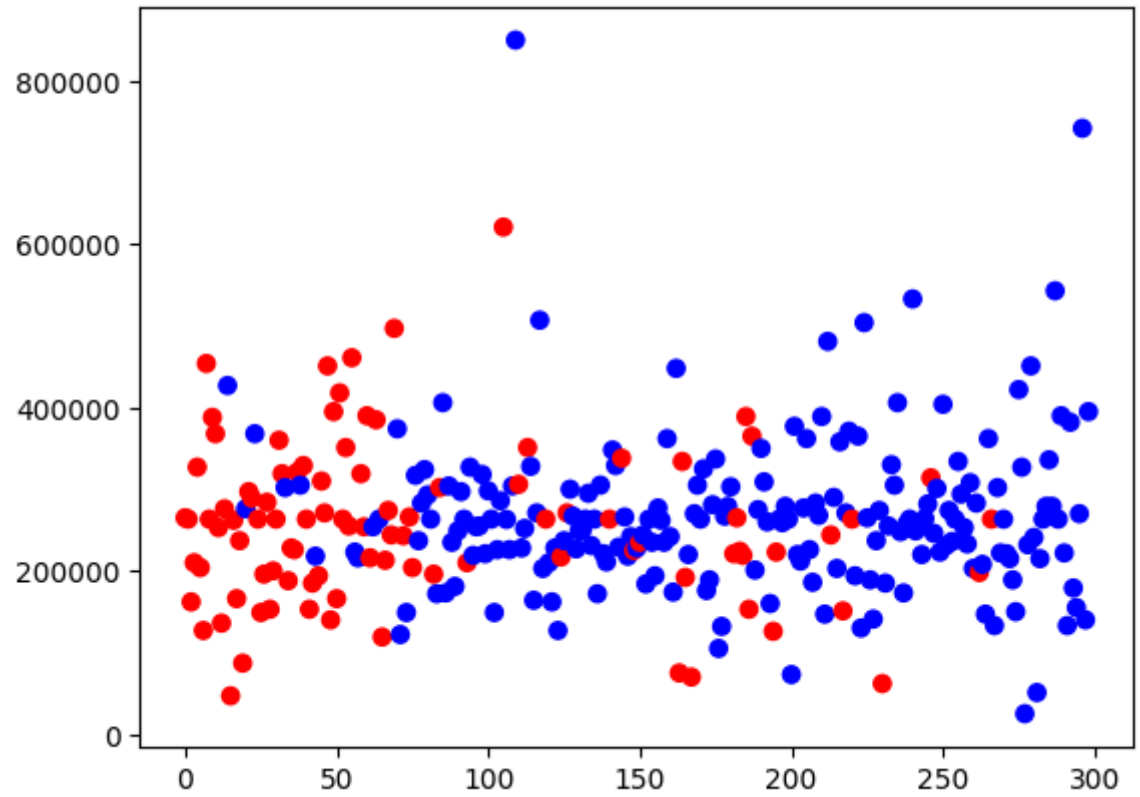
In [38]:
```
plt.figure(figsize=(20, 1))
plt.scatter(y=df.smoking, x=np.arange(299), c=color)
print(df.smoking.value_counts())
```

```
0    203
1     96
Name: smoking, dtype: int64
```



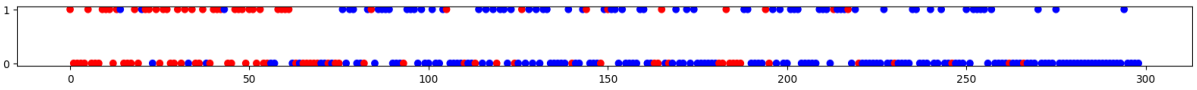
In [42]:
```
plt.scatter(y=df.platelets, x=np.arange(299), c=color)
```

Out[42]: <matplotlib.collections.PathCollection at 0x14a7c3cd0>



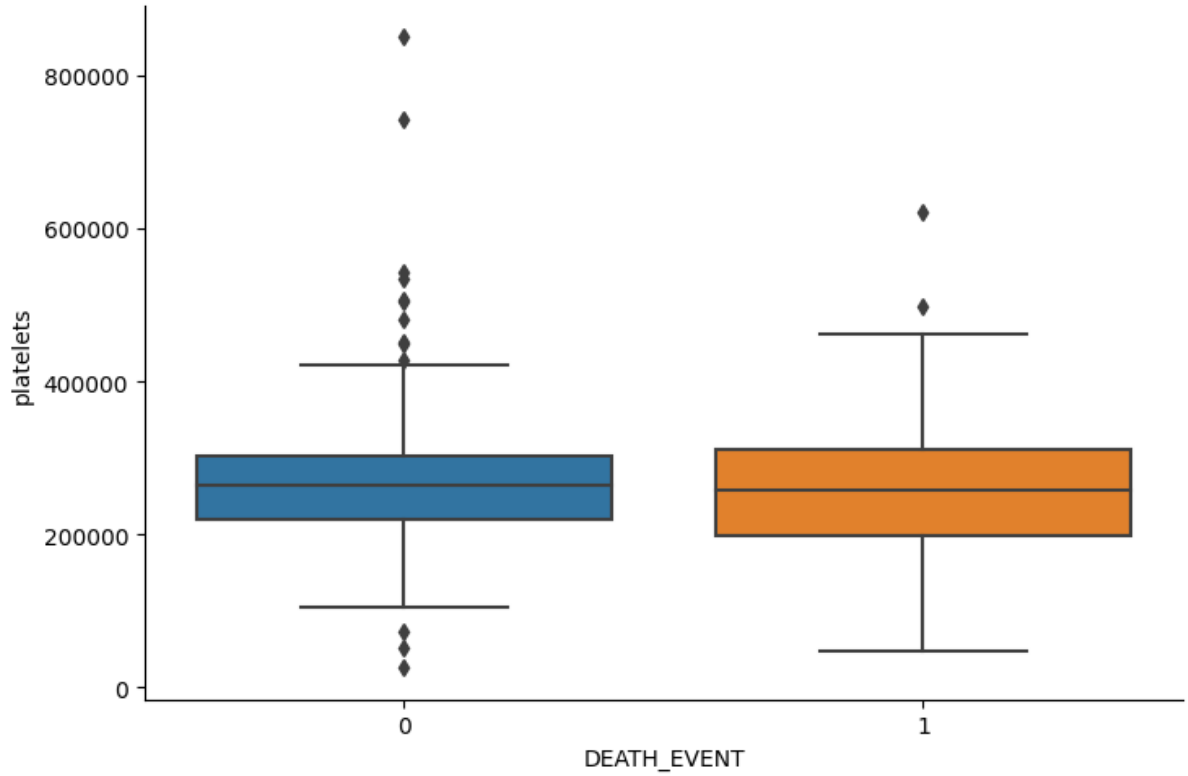
In [65]:
```
plt.figure(figsize=(20, 1))
plt.scatter(y=df.high_blood_pressure, x=np.arange(299), c=color)
print(df.high_blood_pressure.value_counts(normalize=True))
```

```
0    0.648829
1    0.351171
Name: high_blood_pressure, dtype: float64
```



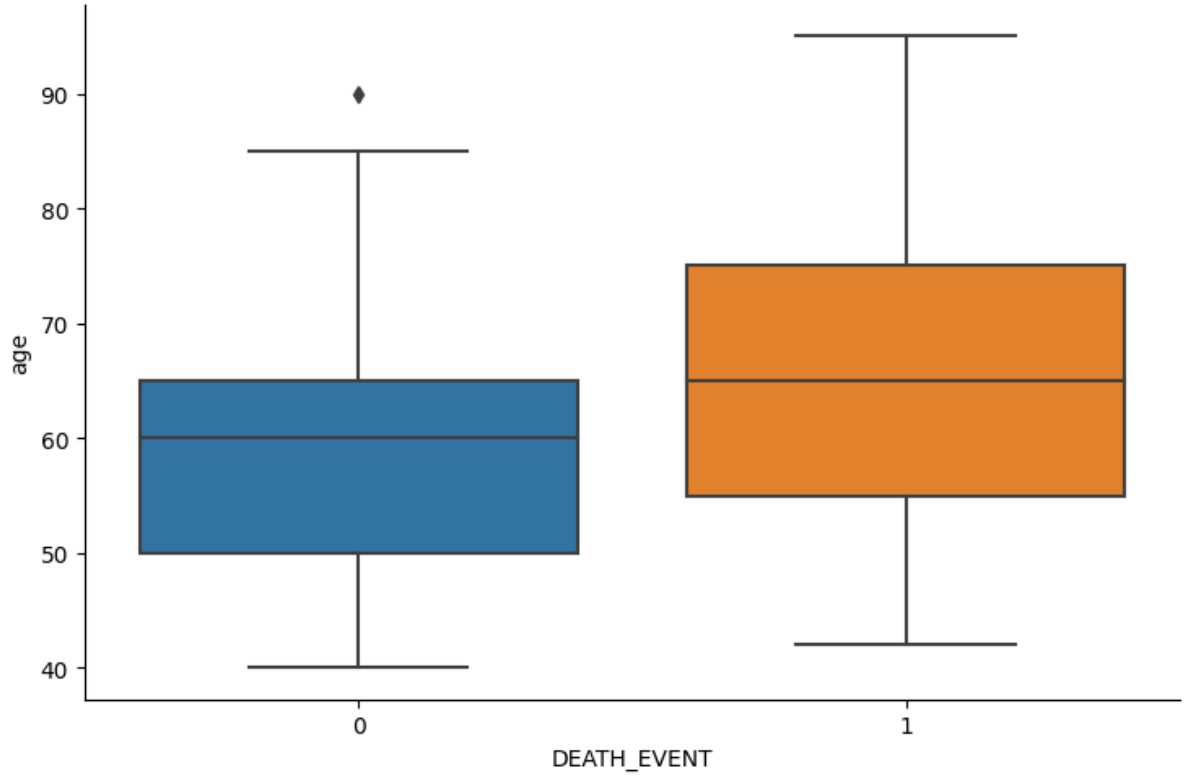
In [67]:
```
sns.catplot(x='DEATH_EVENT', y="platelets", data=df, kind="box", aspect=1.5)
```

Out[67]: <seaborn.axisgrid.FacetGrid at 0x2980cb5b0>



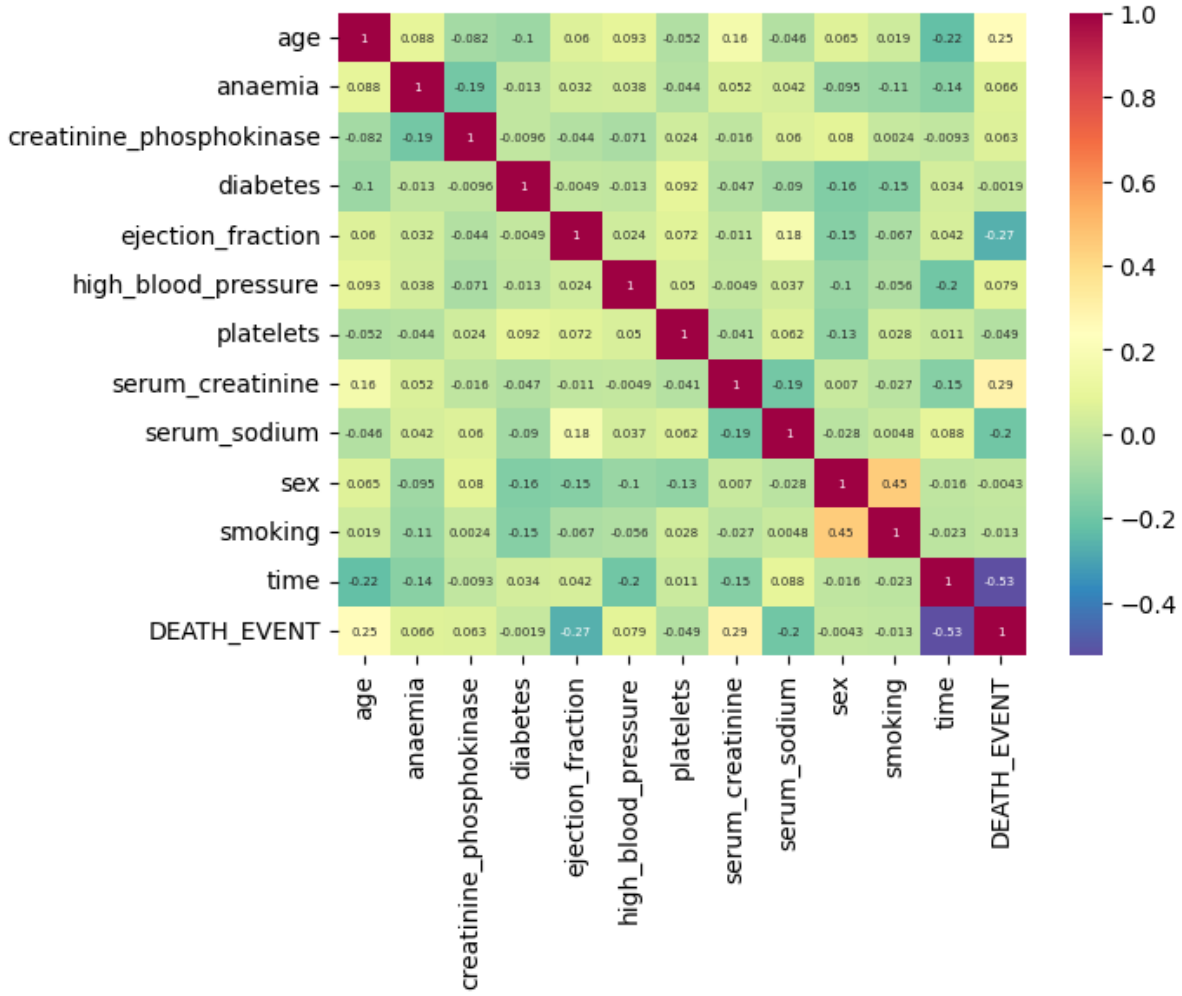
In [68]:
```
sns.catplot(x='DEATH_EVENT', y="age", data=df, kind="box", aspect=1.5)
```

Out[68]:
```
<seaborn.axisgrid.FacetGrid at 0x291cb6560>
```



In [86]:
```
corrmat = df.corr()
sns.heatmap(corrmat, annot=True, annot_kws={'size': 5}, cmap="Spectral_
r")
```

Out[86]: <AxesSubplot: >



2 - Modeling and Hyperparameter Tuning

In [229]:
```
from sklearn.model_selection import GridSearchCV, RepeatedKFold, train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from xgboost import XGBRegressor, XGBClassifier
from sklearn.metrics import roc_auc_score


X = df.drop(columns=['DEATH_EVENT'])
y = df.DEATH_EVENT

X_train, X_val, y_train, y_val = train_test_split(X, y, train_size=.75)

cv = RepeatedKFold(n_splits=5, n_repeats=3, random_state=1)
```

Logistic Regression

In [31]:
```
model = LogisticRegression()
model.fit(X_train, y_train)
roc_auc_score(y_val, model.predict(X_val))
```

Out[31]:
```
0.7787307032590052
```

In [35]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

pipe = Pipeline([
    ('scale', StandardScaler()),
    ('clf', LogisticRegression())
])

model = LogisticRegression()
params = {
    "clf__C":np.logspace(-3,3,7),
    # "clf__penalty":["l1","l2"]
    }
search = GridSearchCV(pipe, params, scoring='roc_auc', cv=cv)
result = search.fit(X, y)
print('Best Score: %s' % result.best_score_)
print('Best Hyperparameters: %s' % result.best_params_)
```

```
Best Score: 0.8840315716393709
Best Hyperparameters: {'clf__C': 0.01}
```

Decision Tree

In [36]:
```
model = DecisionTreeRegressor()
model.fit(X_train, y_train)
roc_auc_score(y_val, model.predict(X_val))
```

Out[36]:
```
0.7731560891938251
```

In [37]:
```
model = DecisionTreeRegressor()
params = {
    'criterion': ['squared_error', 'friedman_mse', 'absolute_error', 'po
isson'],
    'splitter': ['best', 'random'],
    'max_depth': [1, 3, 5, 7, 9, 11, 12, None],
    'min_samples_leaf': [1, 5, 9],
    }
search = GridSearchCV(model, params, scoring='roc_auc', cv=cv)
result = search.fit(X, y)
print('Best Score: %s' % result.best_score_)
print('Best Hyperparameters: %s' % result.best_params_)
```

```
Best Score: 0.861950520208799
Best Hyperparameters: {'criterion': 'squared_error', 'max_depth': 5, 'm
in_samples_leaf': 9, 'splitter': 'best'}
```

Random Forest

In [38]:
```
model = RandomForestRegressor()
model.fit(X_train, y_train)
roc_auc_score(y_val, model.predict(X_val))
```

Out[38]:
```
0.8846483704974272
```

In [39]:
```
model = RandomForestRegressor()
params = {
    'bootstrap': [True, False],
    'max_depth': [1, 3, 5, 7, 9, 11, 12, None],
    }
search = GridSearchCV(model, params, scoring='roc_auc', cv=cv)
result = search.fit(X, y)
print('Best Score: %s' % result.best_score_)
print('Best Hyperparameters: %s' % result.best_params_)
```

```
Best Score: 0.8951430493766708
Best Hyperparameters: {'bootstrap': True, 'max_depth': 3}
```

XG Boost

In [16]:
```
model = XGBRegressor()
model.fit(X_train, y_train)
roc_auc_score(y_val, model.predict(X_val))
```

Out[16]:
```
0.8856209150326797
```

In [215]:
```
model = XGBClassifier()
params = {
    'max_depth': [1, 2, 6, 12, None],
    'gamma': [0, 1, 5, None],
    'eta': [.1, .5, 1],
    }
search = GridSearchCV(model, params, scoring='roc_auc', cv=cv)
result = search.fit(X, y)
print('Best Score: %s' % result.best_score_)
print('Best Hyperparameters: %s' % result.best_params_)
```

```
Best Score: 0.9104535965873164
Best Hyperparameters: {'eta': 0.1, 'gamma': 0, 'max_depth': 1}
```

3 - Interpretation/Explanation

In [92]:
```
from eli5 import explain_prediction, explain_weights, show_weights
from eli5.sklearn import PermutationImportance
import random
import shap
import lime
from lime import lime_tabular
```

Logistic Regression

ELI5 Weights

In [82]:
```
model = LogisticRegression(C=0.01, penalty='l2')
model.fit(X_train, y_train)
explain_weights(model, feature_names=list(X.columns))
```

Out[82]: **y=1** top features

Weight[?]	Feature
+0.047	age
+0.013	serum_sodium
+0.006	serum_creatinine
+0.001	sex
+0.000	anaemia
+0.000	<BIAS>
+0.000	high_blood_pressure
+0.000	diabetes
+0.000	creatinine_phosphokinase
-0.000	platelets
-0.000	smoking
-0.020	time
-0.061	ejection_fraction

ELI5 Negative Prediction

In [88]:
```
explain_prediction(model, X.sample(1), feature_names=list(X.columns))
```

Out[88]: **y=0** (probability **0.917**, score **-2.403**) top features

Contribution[?]	Feature
+3.778	time
+2.123	ejection_fraction
+0.820	platelets
+0.000	smoking
-0.000	diabetes
-0.000	<BIAS>
-0.001	sex
-0.011	serum_creatinine
-0.072	creatinine_phosphokinase
-1.431	serum_sodium
-2.804	age

ELI5 Positive Prediction

In [89]:
```
explain_prediction(model, X.sample(1), feature_names=list(X.columns))
```

Out[89]: **y=1** (probability **0.512**, score **0.049**) top features

Contribution[?]	Feature
+2.570	age
+1.735	serum_sodium
+0.007	serum_creatinine
+0.003	creatinine_phosphokinase
+0.001	sex
+0.000	<BIAS>
+0.000	high_blood_pressure
-0.539	platelets
-1.605	time
-2.123	ejection_fraction

Decision Tree

ELI5 Feature Importance

In [98]:
```
model = DecisionTreeRegressor(criterion='poisson', max_depth= 5, min_sam
ples_leaf=9, splitter='best')
model.fit(X_train, y_train)
perm = PermutationImportance(model).fit(X_val, y_val)
show_weights(perm, feature_names=list(X.columns))
```

Out[98]:

Weight	Feature
0.5605 ± 0.4870	time
0.2095 ± 0.1029	serum_creatinine
0.0279 ± 0.0443	ejection_fraction
0 ± 0.0000	smoking
0 ± 0.0000	sex
0 ± 0.0000	serum_sodium
0 ± 0.0000	high_blood_pressure
0 ± 0.0000	diabetes
0 ± 0.0000	creatinine_phosphokinase
0 ± 0.0000	anaemia
0 ± 0.0000	age
-0.0486 ± 0.1038	platelets

ELI5 Negative Prediction

In [109]:
```
explain_prediction(model, X_val.sample(1), feature_names=list(X.column
s))
```

Out[109]: **y** (score **0.048**) top features

Contribution[?]	Feature
+0.330	<BIAS>
-0.037	ejection_fraction
-0.076	serum_creatinine
-0.170	time

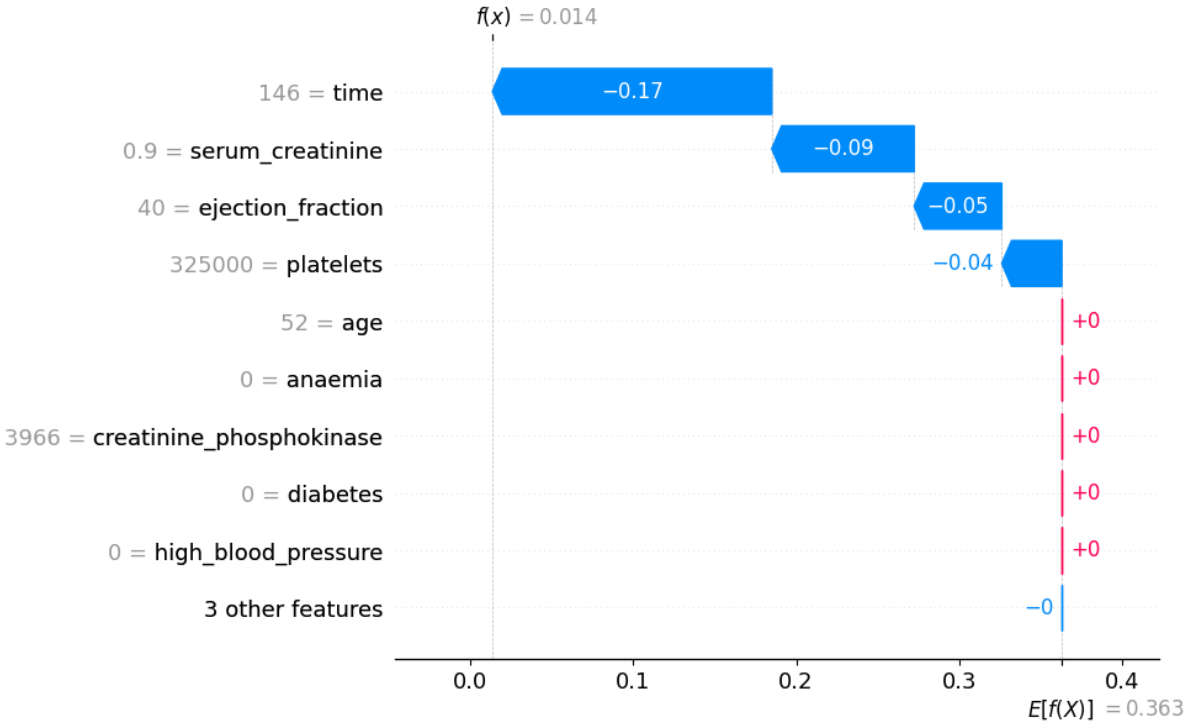
ELI5 Positive Prediction

In [106]:
```
explain_prediction(model, X_val.sample(1), feature_names=list(X.column
s))
```

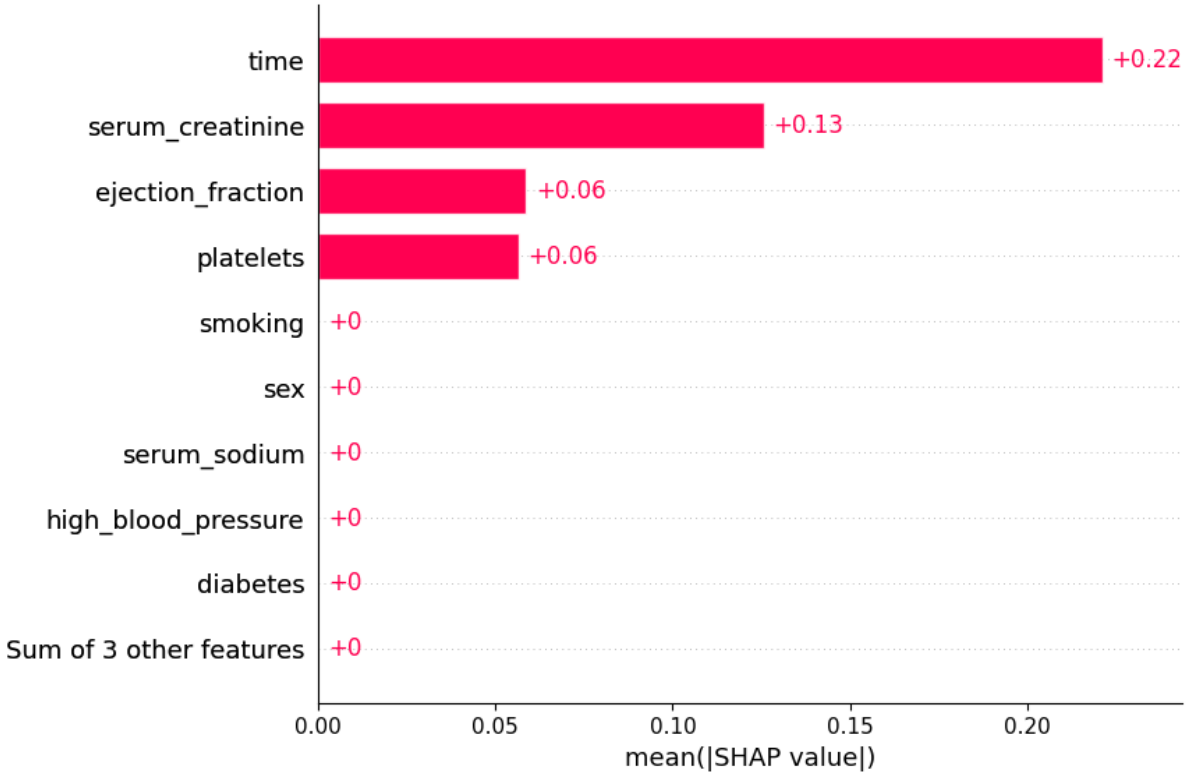
Out[106]: **y** (score **0.833**) top features

Contribution[?]	Feature
+0.395	serum_creatinine
+0.330	<BIAS>
+0.278	platelets
-0.170	time

In [47]:
```
exp = shap.Explainer(model.predict, X_val)
shap_values = exp(X_val)
shap.plots.waterfall(shap_values[0])
```



In [48]:
```
shap.plots.bar(shap_values)
```

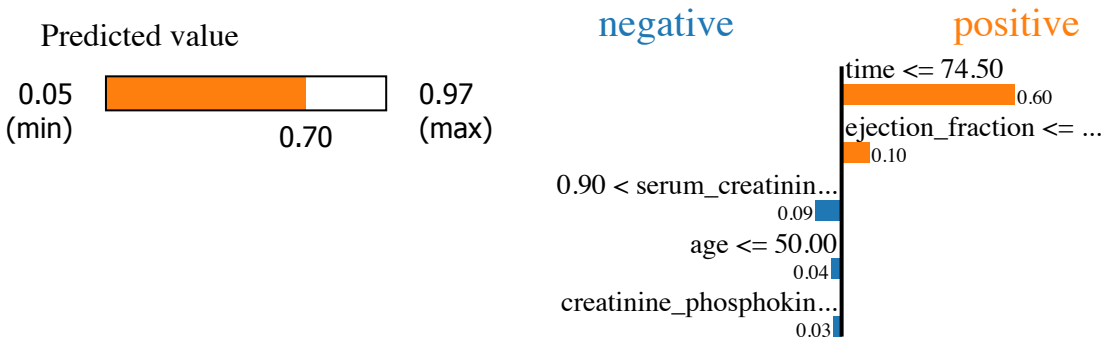


Random Forest

In [176]:
```
model = RandomForestRegressor(bootstrap=True, max_depth=3)
model.fit(X_train, y_train)
explainer = lime.lime_tabular.LimeTabularExplainer(np.array(X_val), feat
ure_names=list(X.columns), class_names=['Lived', 'Died'], verbose=True,
mode='regression')
exp = explainer.explain_instance(np.array(X_val).any(0), model.predict,
num_features=5)
exp.show_in_notebook()
```

```
Intercept 0.2328855372193234
Prediction_local [0.77317219]
Right: 0.703575680789429

X does not have valid feature names, but RandomForestRegressor was fitt
ed with feature names
```



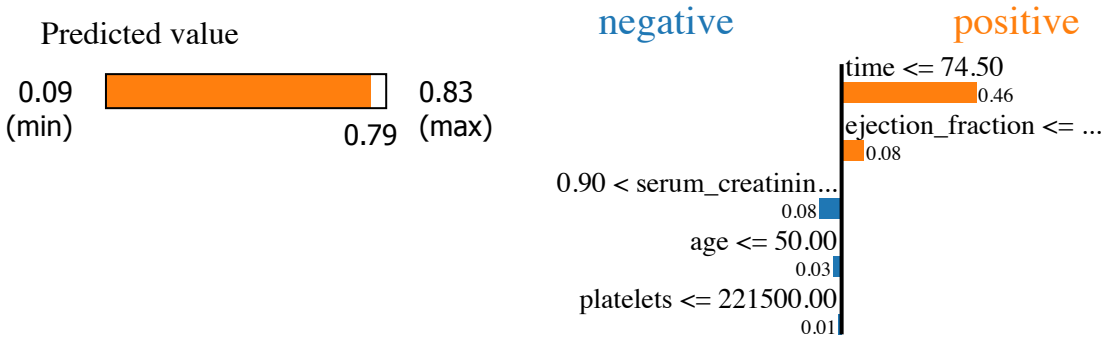
Feature	Value
time	1.00
ejection_fraction	1.00
serum_creatinine	1.00
age	1.00
creatinine_phosphokinase	1.00

XG Boost

LIME Explainer

In [183]:
```
model = XGBRegressor(eta=0.1, gamma=1, max_depth=3)
model.fit(X_train, y_train)
explainer = lime.lime_tabular.LimeTabularExplainer(np.array(X_val), feat
ure_names=list(X.columns), class_names=['Lived', 'Died'], verbose=True,
mode='regression')
exp = explainer.explain_instance(np.array(X_val).any(0), model.predict,
num_features=5)
exp.show_in_notebook()
```

```
Intercept 0.2495987378250853
Prediction_local [0.67673259]
Right: 0.7865541
```

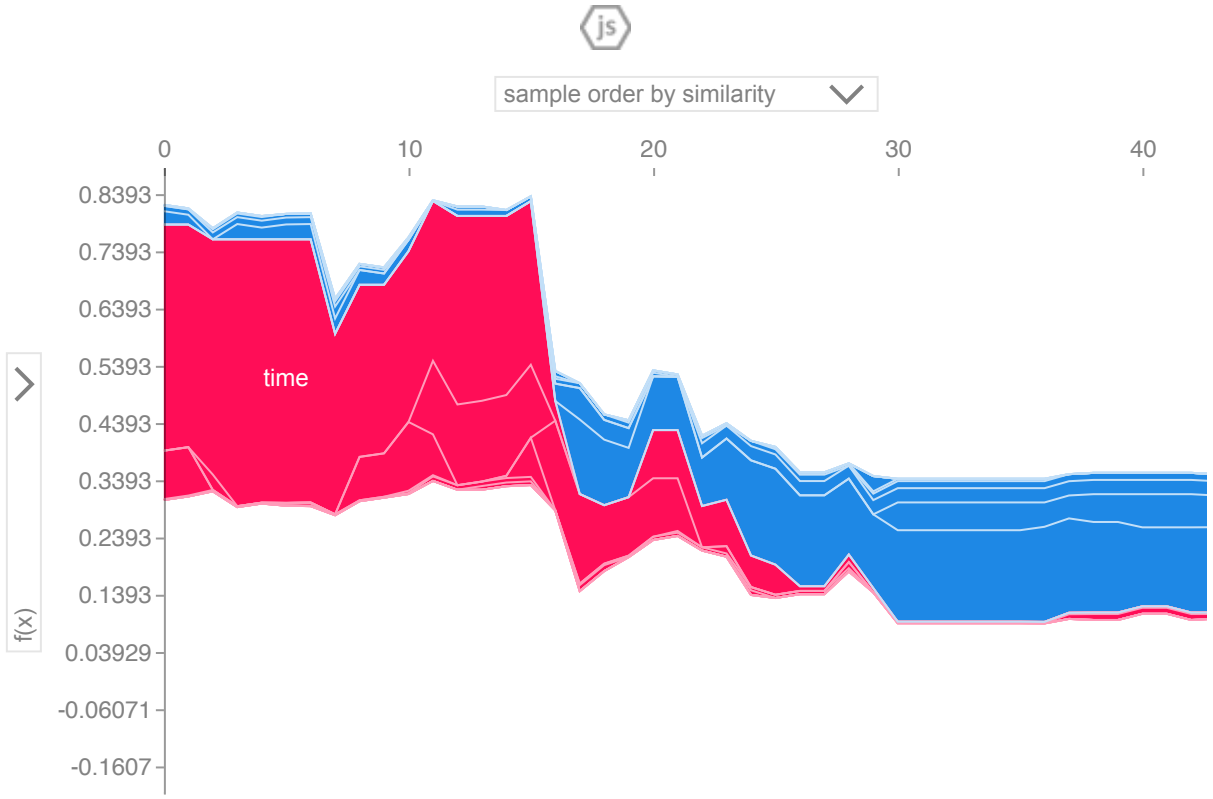


Feature	Value
time	1.00
ejection_fraction	1.00
serum_creatinine	1.00
age	1.00
platelets	1.00

Shap Tree Explainer of full validation set

In [202]:
```
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_val)
shap.initjs()
shap.force_plot(explainer.expected_value, shap_values, feature_names=lis
t(X.columns))
```

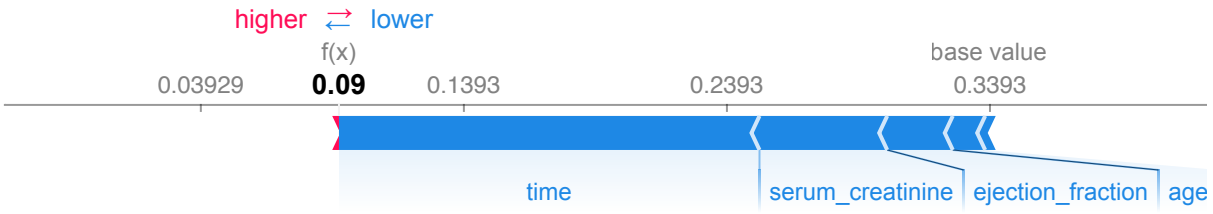
Out[202]:



Shap Tree Explainer of negative target

In [209]:
```
shap.force_plot(explainer.expected_value, shap_values[0], feature_names=
list(X.columns))
```

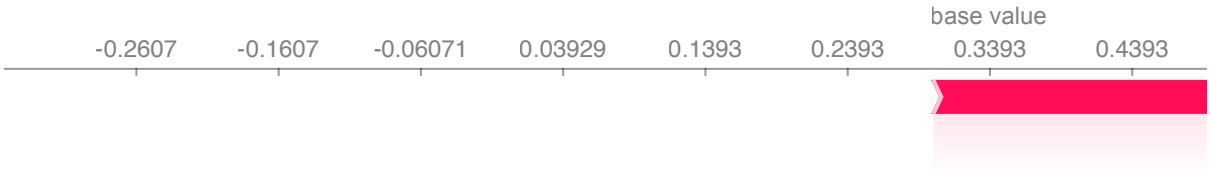
Out[209]:



Shap Tree Explainer of positive target

In [208]:
```
shap.force_plot(explainer.expected_value, shap_values[3], feature_names=
list(X.columns))
```

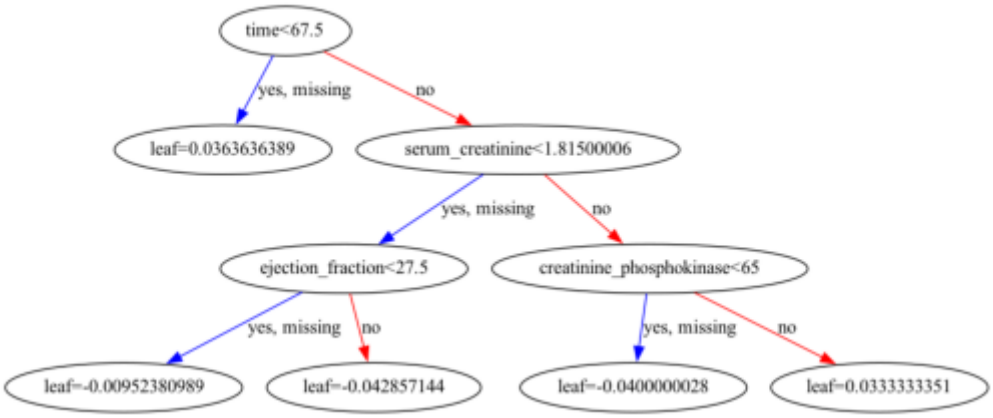
Out[208]:



XGBoost Tree Visualization

In [197]:
```
from xgboost import plot_tree, plot_importance
plot_tree(model)
```

Out[197]: <AxesSubplot: >



4 - Predicting Observations

Finding probabilities of each model

In [236]:
```
#Switching to classifiers to use predict_proba method
lr = LogisticRegression(C=0.01, penalty='l2')
dt = DecisionTreeClassifier(max_depth= 5, min_samples_leaf=9, splitter
='best')
rf = RandomForestClassifier(bootstrap=True, max_depth=3)
xgb = XGBClassifier(eta=0.1, gamma=1, max_depth=3)

lr.fit(X_train, y_train)
dt.fit(X_train, y_train)
rf.fit(X_train, y_train)
xgb.fit(X_train, y_train)
```

```
lr_probabilities = lr.predict_proba(X_val)
dt_probabilities = dt.predict_proba(X_val)
rf_probabilities = rf.predict_proba(X_val)
xgb_probabilities = xgb.predict_proba(X_val)
```

Negative Target Probabilities

In [258]:
```
# The below shows predictions for each model when the target value was n
egative
# Logistic Regression was confident it was positive
# Decision Tree was confident it was negative
# Random Forest was split
# XG Boost was very confident it was negative
# Thus, DT and XGB are the most accurate, with XGB having the highest co
nfidence of the correct answer

print("Logistic Regression:", lr_probabilities[0])
print("Decision Tree:", dt_probabilities[0])
print("Random Forest:", rf_probabilities[0])
print("XG Boost:", xgb_probabilities[0])
```

```
Logistic Regression: [0.71889879 0.28110121]
Decision Tree: [0.22222222 0.77777778]
Random Forest: [0.46783977 0.53216023]
XG Boost: [0.100703 0.899297]
```

In [259]:
```
# The below shows predictions for each model when the target value was p
ositive
# Logistic Regression was very confident it was positive
# Decision Tree was completely confident it was positive
# Random Forest was very confident it was positive
# XG Boost was almost completely confident it was positive
# Thus, every model was accurate, with XGB having the highest confidence
of the correct answer
# But in general, this indicates that a negative event is much easier to
predict than a positive

print("Logistic Regression:", lr_probabilities[4])
print("Decision Tree:", dt_probabilities[4])
print("Random Forest:", rf_probabilities[4])
print("XG Boost:", xgb_probabilities[4])
```

```
Logistic Regression: [0.98992821 0.01007179]
Decision Tree: [1. 0.]
Random Forest: [0.86878131 0.13121869]
XG Boost: [0.98236245 0.01763754]
```

In [ ]: