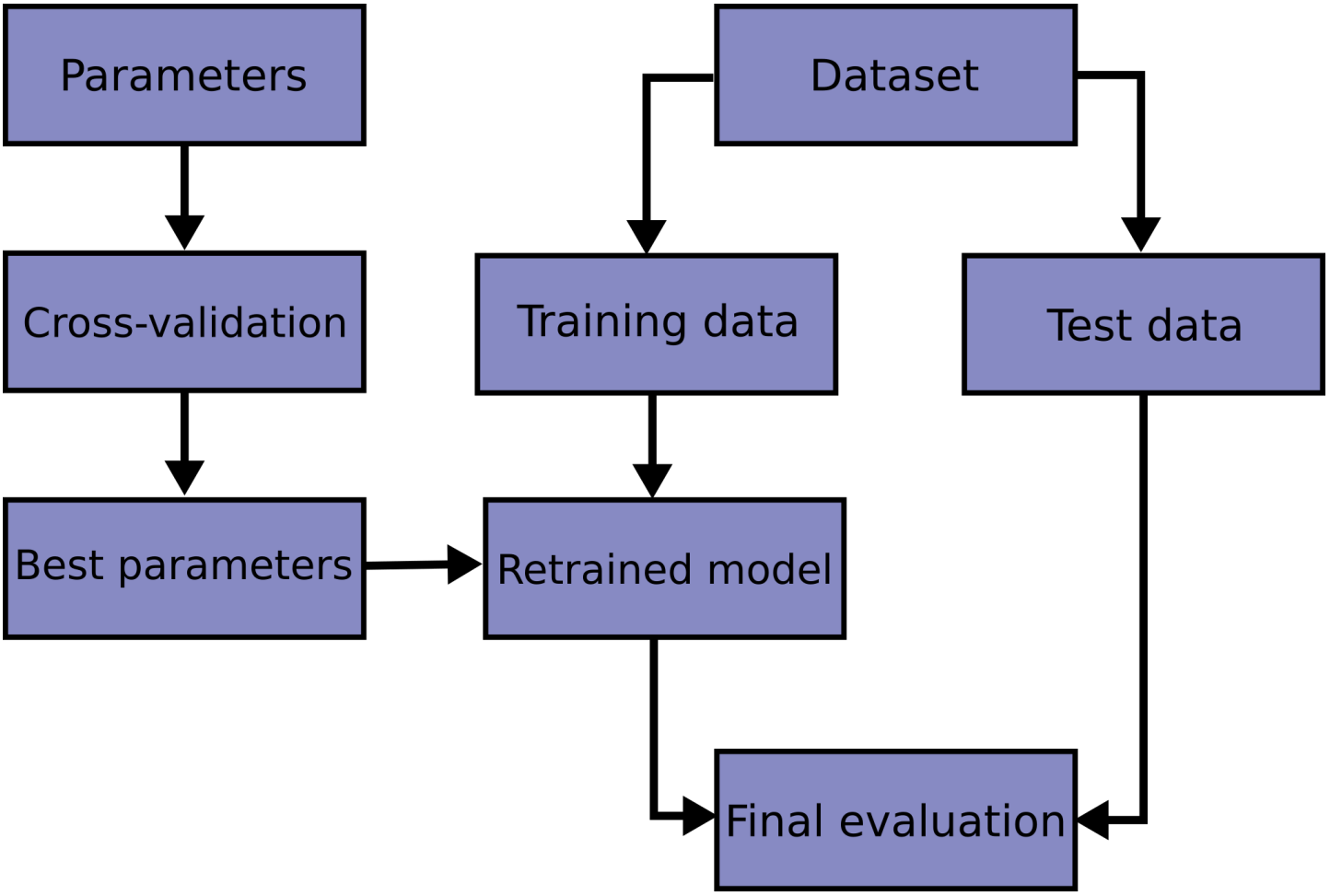
%%[markdown]


Parameters
Dataset
Cross-validation
Training data
Test data
Best parameters
Retrained model
Final evaluation

In [1]:
```
'''
robust versions of logistic regression
support vector machines
random forests
gradient boosted decision trees (XGBoost)
neural networks

Documentacion oficial:
    Cross Validation: https://scikit-learn.org/stable/modules/cross_validation.html
    GridSearchCV: https://scikit-learn.org/stable/modules/grid_search.html#grid-search
    Pipelines: https://scikit-learn.org/stable/modules/compose.html#

https://www.kdnuggets.com/2020/06/simplifying-mixed-feature-type-preprocessing-scikit-learn-pipelines.html
https://www.kaggle.com/kritidoneria/explainable-ai-eli5-lime-and-shap
https://shap.readthedocs.io/en/latest/example_notebooks/tabular_examples/model_agnostic/Diabetes%20regression.html
https://kiwidamien.github.io/introducing-the-column-transformer.html
https://medium.com/analytics-vidhya/shap-part-2-kernel-shap-3c11e7a971b1

*Pendientes*
1)cual tipo de score es el mejor
2)como entrenar con dataset desbalanceado 10:1
3)como evaluar varios modelos adentro de gridsearchcv
'''

import os, sys
import pandas as pd
import numpy as np
import seaborn as sns; sns.set()
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from scipy import stats
from sklearn.svm import SVC
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
import joblib
sns.set(color_codes=True)
from sklearn.pipeline import Pipeline
from sklearn.compose import make_column_selector
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
```

In [2]:
```
path = "/home/nacho/Documents/coronavirus/COVID-19_Paper/"
os.chdir(os.path.join(path))
```

In [3]:
```
#%%Valida si existen las carpetas
try:
    os.makedirs("plots")
    os.makedirs("models")
except FileExistsError:
    pass
```

In [3]:
```
#%%gridsearchcv
#checar stratify
#checar tipo de scoring
#sklearn.metrics.SCORERS.keys()
def gridsearchcv(X_train, X_test, y_train, y_test):
    ############
    # Scale numeric values
    num_transformer = Pipeline(steps=[
        ('scaler', MinMaxScaler())])
        #('scaler', StandardScaler())])

    preprocessor = ColumnTransformer(
        remainder='passthrough',
        transformers=[
            ('num', num_transformer, make_column_selector(pattern='EDAD'))
            ])
    ############

    pipe_steps = [
        #('scaler', StandardScaler()),
        ('preprocessor', preprocessor),
        ('SupVM', SVC(kernel='rbf',probability=True)) #agregar kernel linear
        ]

    param_grid= {
        'SupVM__C': [0.1, 0.5, 1, 10, 30, 40, 50, 75, 100, 250, 500, 750, 1000],
        'SupVM__gamma' : [0.0001, 0.001, 0.005, 0.01, 0.05, 0.07, 0.1, 0.5, 1, 5, 10, 50, 75, 100]
        }

    pipeline = Pipeline(pipe_steps)
    grid = GridSearchCV(pipeline, param_grid,refit = True,cv = 5, verbose = 3, n_jobs=-1,scoring='balanced_accuracy')#'f1'
    grid.fit(X_train, y_train)
    print("Best-Fit Parameters From Training Data:\n",grid.best_params_)
    grid_predictions = grid.best_estimator_.predict(X_test)
    report = classification_report(y_test, grid_predictions, output_dict=True)
    report = pd.DataFrame(report).transpose()
    print(report)
    print(confusion_matrix(y_test, grid_predictions))
    return grid, report
```

In [3]:
```
#%%CASO 1: prediccion de hospitalizacion por covid
hosp_data = pd.read_csv("prediction_data/df_caso1.zip")
#Porcentaje de informacion del dataset
hosp_data = hosp_data.sample(frac=0.001)

#separar datos
X = hosp_data.loc[:, hosp_data.columns != 'TIPO_PACIENTE']
y = hosp_data.loc[:,'TIPO_PACIENTE']
print(y.value_counts())
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42,stratify=y, shuffle=True)
```

```
0    1697
1     400
Name: TIPO_PACIENTE, dtype: int64
```

In [10]:
```
#%%Entrenamiento (opcional)
#---->train
grid, grid_report= gridsearchcv(X_train, X_test, y_train, y_test)
#guarda el modelo y su reporte
joblib.dump(grid, 'models/hosp_data_grid.pkl', compress = 1)
grid_report.to_csv("models/hosp_data_grid_report.csv", index=True)
'''
Best-Fit Parameters From Training Data:
 {'SupVM__C': 500, 'SupVM__gamma': 1}
              precision    recall  f1-score    support
0              0.845872  0.964435  0.901271  478.00000
1              0.645833  0.269565  0.380368  115.00000
accuracy       0.829680  0.829680  0.829680    0.82968
macro avg      0.745852  0.617000  0.640819  593.00000
weighted avg   0.807078  0.829680  0.800253  593.00000
[[461  17]
 [ 84  31]]
'''
```

```
Fitting 5 folds for each of 154 candidates, totalling 770 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    2.6s
[Parallel(n_jobs=-1)]: Done 112 tasks      | elapsed:    6.6s
[Parallel(n_jobs=-1)]: Done 272 tasks      | elapsed:   15.4s
[Parallel(n_jobs=-1)]: Done 496 tasks      | elapsed:   43.4s
[Parallel(n_jobs=-1)]: Done 770 out of 770 | elapsed:  3.2min finished
Best-Fit Parameters From Training Data:
 {'SupVM__C': 75, 'SupVM__gamma': 5}
              precision    recall  f1-score     support
0              0.828897  0.923729  0.873747  472.000000
1              0.462687  0.256198  0.329787  121.000000
accuracy       0.787521  0.787521  0.787521    0.787521
macro avg      0.645792  0.589964  0.601767  593.000000
weighted avg   0.754173  0.787521  0.762754  593.000000
[[436  36]
 [ 90  31]]
```

Out[10]:
```
"\nBest-Fit Parameters From Training Data:\n {'SupVM__C': 500, 'SupVM__gamma': 1}\n              precision    recall  f1-score    support
\n0              0.845872  0.964435  0.901271  478.00000\n1              0.645833  0.269565  0.380368  115.00000\naccuracy       0.829680
0.829680  0.829680    0.82968\nmacro avg      0.745852  0.617000  0.640819  593.00000\nweighted avg   0.807078  0.829680  0.800253  593.00
000\n[[461  17]\n [ 84  31]]\n"
```

In [4]:
```
#%%importa el modelo y su rendimiento
grid = joblib.load('models/hosp_data_grid.pkl')
grid_report = pd.read_csv("models/hosp_data_grid_report.csv", index_col=0)
print("best score from grid search: %f" % grid.best_estimator_.score(X_test, y_test))
```

```
best score from grid search: 0.809524
```

In [11]:
```
#prediccion
grid.predict(pd.DataFrame(X_train.iloc[20, :].values.reshape(1, -1), columns = X_train.columns))
```

Out[11]:
```
array([0])
```

In [12]:
```
#prediccion con probabilidad
grid.predict_proba(pd.DataFrame(X_train.iloc[20, :].values.reshape(1, -1), columns = X_train.columns))
```

Out[12]:
```
array([[0.71358794, 0.28641206]])
```

In [6]:
```
#metodos y atributos del objeto grid
dir(grid)
```

Out[6]:
```
['__abstractmethods__',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__setstate__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_abc_cache',
 '_abc_negative_cache',
 '_abc_negative_cache_version',
 '_abc_registry',
 '_check_is_fitted',
 '_estimator_type',
 '_format_results',
 '_get_param_names',
 '_get_tags',
 '_more_tags',
 '_pairwise',
 '_required_parameters',
 '_run_search',
 'best_estimator_',
 'best_index_',
 'best_params_',
 'best_score_',
 'classes_',
 'cv',
 'cv_results_',
 'decision_function',
 'error_score',
 'estimator',
 'fit',
 'get_params',
 'iid',
 'inverse_transform',
 'multimetric_',
 'n_iter',
 'n_jobs',
 'n_splits_',
 'param_distributions',
 'pre_dispatch',
 'predict',
 'predict_log_proba',
 'predict_proba',
 'random_state',
 'refit',
 'refit_time_',
 'return_train_score',
 'score',
 'scorer_',
 'scoring',
 'set_params',
 'transform',
 'verbose']
```

In [13]:
```
grid.best_score_
```

Out[13]:
```
0.3233383946258245
```

In [15]:
```
grid.best_params_
```

Out[15]:
```
{'clf__selected_model': ('gb',
  {'learning_rate': 0.15,
   'max_depth': 9,
   'max_features': 'sqrt',
   'n_estimators': 50,
   'subsample': 0.618})}
```

In [14]:
```
grid.best_estimator_
```

Out[14]:
```
Pipeline(memory=None,
         steps=[('preprocessor',
                 ColumnTransformer(n_jobs=None, remainder='passthrough',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('num',
                                                  Pipeline(memory=None,
                                                           steps=[('scaler',
                                                                   MinMaxScaler(copy=True,
                                                                                feature_range=(0,
                                                                                               1)))],
                                                           verbose=False),
                                                  <sklearn.compose._column_transformer.make_column_selector object at 0x7fe629f43be0>)],
                                   ver...
                                                                              loss='deviance',
                                                                              max_depth=9,
                                                                              max_features='sqrt',
                                                                              max_leaf_nodes=None,
                                                                              min_impurity_decrease=0.0,
                                                                              min_impurity_split=None,
                                                                              min_samples_leaf=1,
                                                                              min_samples_split=2,
                                                                              min_weight_fraction_leaf=0.0,
                                                                              n_estimators=50,
                                                                              n_iter_no_change=None,
                                                                              presort='deprecated',
                                                                              random_state=None,
                                                                              subsample=0.618,
                                                                              tol=0.0001,
                                                                              validation_fraction=0.1,
                                                                              verbose=0,
                                                                              warm_start=False)))],
         verbose=False)
```

In [7]:
```
df_grid.columns.to_list()
```

Out[7]:
```
['mean_fit_time',
 'std_fit_time',
 'mean_score_time',
 'std_score_time',
 'param_clf__selected_model',
 'params',
 'split0_test_f1',
 'split1_test_f1',
 'split2_test_f1',
 'split3_test_f1',
 'split4_test_f1',
 'split5_test_f1',
 'split6_test_f1',
 'split7_test_f1',
 'split8_test_f1',
 'split9_test_f1',
 'split10_test_f1',
 'split11_test_f1',
 'split12_test_f1',
 'split13_test_f1',
 'split14_test_f1',
 'split15_test_f1',
 'split16_test_f1',
 'split17_test_f1',
 'split18_test_f1',
 'split19_test_f1',
 'split20_test_f1',
 'split21_test_f1',
 'split22_test_f1',
 'split23_test_f1',
 'split24_test_f1',
 'split25_test_f1',
 'split26_test_f1',
 'split27_test_f1',
 'split28_test_f1',
 'split29_test_f1',
 'split30_test_f1',
 'split31_test_f1',
 'split32_test_f1',
 'split33_test_f1',
 'split34_test_f1',
 'split35_test_f1',
 'split36_test_f1',
 'split37_test_f1',
 'split38_test_f1',
 'split39_test_f1',
 'split40_test_f1',
 'split41_test_f1',
 'split42_test_f1',
 'split43_test_f1',
 'split44_test_f1',
 'split45_test_f1',
 'split46_test_f1',
 'split47_test_f1',
 'split48_test_f1',
 'split49_test_f1',
 'mean_test_f1',
 'std_test_f1',
 'rank_test_f1',
 'split0_test_balanced_accuracy',
 'split1_test_balanced_accuracy',
 'split2_test_balanced_accuracy',
 'split3_test_balanced_accuracy',
 'split4_test_balanced_accuracy',
 'split5_test_balanced_accuracy',
 'split6_test_balanced_accuracy',
 'split7_test_balanced_accuracy',
 'split8_test_balanced_accuracy',
 'split9_test_balanced_accuracy',
 'split10_test_balanced_accuracy',
 'split11_test_balanced_accuracy',
 'split12_test_balanced_accuracy',
 'split13_test_balanced_accuracy',
 'split14_test_balanced_accuracy',
 'split15_test_balanced_accuracy',
 'split16_test_balanced_accuracy',
 'split17_test_balanced_accuracy',
 'split18_test_balanced_accuracy',
 'split19_test_balanced_accuracy',
 'split20_test_balanced_accuracy',
 'split21_test_balanced_accuracy',
 'split22_test_balanced_accuracy',
 'split23_test_balanced_accuracy',
 'split24_test_balanced_accuracy',
 'split25_test_balanced_accuracy',
 'split26_test_balanced_accuracy',
 'split27_test_balanced_accuracy',
 'split28_test_balanced_accuracy',
 'split29_test_balanced_accuracy',
 'split30_test_balanced_accuracy',
 'split31_test_balanced_accuracy',
 'split32_test_balanced_accuracy',
 'split33_test_balanced_accuracy',
 'split34_test_balanced_accuracy',
 'split35_test_balanced_accuracy',
 'split36_test_balanced_accuracy',
 'split37_test_balanced_accuracy',
 'split38_test_balanced_accuracy',
 'split39_test_balanced_accuracy',
 'split40_test_balanced_accuracy',
 'split41_test_balanced_accuracy',
 'split42_test_balanced_accuracy',
```

```
 'split43_test_balanced_accuracy',
 'split44_test_balanced_accuracy',
 'split45_test_balanced_accuracy',
 'split46_test_balanced_accuracy',
 'split47_test_balanced_accuracy',
 'split48_test_balanced_accuracy',
 'split49_test_balanced_accuracy',
 'mean_test_balanced_accuracy',
 'std_test_balanced_accuracy',
 'rank_test_balanced_accuracy']
```

In [9]:
```
df_grid = pd.DataFrame(grid.cv_results_)
df_grid.sort_values(by = ['mean_test_balanced_accuracy'], ascending=False)
```

Out[9]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_clf__selected_model	params	split0_test_f1	split1_test_f1	split2_test_f1	split3_test_f1	...	split43_test_b
0	0.643435	0.056959	0.013391	0.003793	(gb, {'learning_rate': 0.15, 'max_depth': 9, '...	{'clf__selected_model': ('gb', {'learning_rate...	0.318182	0.384615	0.333333	0.186047	...	
9	1.728619	0.153196	0.013713	0.002805	(gb, {'learning_rate': 0.025, 'max_depth': 8, ...	{'clf__selected_model': ('gb', {'learning_rate...	0.315789	0.238095	0.390244	0.108108	...	
1	16.933959	1.364132	0.034127	0.006654	(gb, {'learning_rate': 0.075, 'max_depth': 50,...	{'clf__selected_model': ('gb', {'learning_rate...	0.297872	0.313725	0.425532	0.186047	...	
4	6.832211	0.269802	0.021366	0.003339	(gb, {'learning_rate': 0.05, 'max_depth': 8, '...	{'clf__selected_model': ('gb', {'learning_rate...	0.285714	0.280000	0.434783	0.232558	...	
2	1.817193	0.063662	0.014718	0.001571	(gb, {'learning_rate': 0.2, 'max_depth': 3, 'm...	{'clf__selected_model': ('gb', {'learning_rate...	0.292683	0.243902	0.292683	0.190476	...	
8	4.623009	0.257587	0.018732	0.003551	(gb, {'learning_rate': 0.075, 'max_depth': 8, ...	{'clf__selected_model': ('gb', {'learning_rate...	0.243902	0.285714	0.375000	0.222222	...	
7	18.606113	0.633623	0.029817	0.006988	(gb, {'learning_rate': 0.01, 'max_depth': 50, ...	{'clf__selected_model': ('gb', {'learning_rate...	0.272727	0.346154	0.434783	0.227273	...	
6	0.289662	0.025178	0.012307	0.001844	(gb, {'learning_rate': 0.025, 'max_depth': 3, ...	{'clf__selected_model': ('gb', {'learning_rate...	0.000000	0.303030	0.285714	0.277778	...	
5	12.370118	0.425477	0.025213	0.002602	(gb, {'learning_rate': 0.001, 'max_depth': 10,...	{'clf__selected_model': ('gb', {'learning_rate...	0.000000	0.071429	0.129032	0.000000	...	
3	0.534879	0.040907	0.012240	0.001531	(gb, {'learning_rate': 0.001, 'max_depth': 8, ...	{'clf__selected_model': ('gb', {'learning_rate...	0.000000	0.000000	0.000000	0.000000	...	

10 rows × 112 columns

In [37]:
```
#%%Pipeline without gridsearchcv

num_transformer = Pipeline(steps=[
    ('scaler', MinMaxScaler())])

preprocessor = ColumnTransformer(
    remainder='passthrough',
    transformers=[
        ('num', num_transformer, ['EDAD'])])

svc_model = Pipeline([
    #('scaler', StandardScaler()),
    ('preprocessor', preprocessor),
    ('SupVM', SVC(kernel='rbf',probability=True))])

#svc_model.set_params(**grid.best_params_)
svc_model.set_params(**{'SupVM__C': 500, 'SupVM__gamma': 1})
#svc_model.set_params(**{'SupVM__C': 75, 'SupVM__gamma': 5})
#print("Model params: ",svc_model.get_params("model"))

svc_model.fit(X_train, y_train)
y_pred = svc_model.predict(X_test)
metrics.accuracy_score(y_test, y_pred)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.93      0.88       478
           1       0.48      0.28      0.35       115

    accuracy                           0.80       593
   macro avg       0.66      0.60      0.62       593
weighted avg       0.77      0.80      0.78       593
```

In [39]:
```
#train evaluation
model_score = cross_val_score(svc_model, X_train, y_train, cv=StratifiedKFold(n_splits=5))
print("5 split scores:" ,model_score)
print("Mean of the 5 split scores:" ,model_score.mean())
```

```
5 split scores: [0.8340249  0.81327801 0.80912863 0.825      0.82916667]
Mean of the 5 split scores: 0.8221196403872753
```

In [36]:
```
#final evaluation
final_score = cross_val_score(svc_model, X_test, y_test, cv=StratifiedKFold(n_splits=5))
print("5 split scores:" ,final_score)
print("Mean of the 5 split scores:" ,final_score.mean())
```

```
5 split scores: [0.79831933 0.79831933 0.8487395  0.78813559 0.80508475]
Mean of the 5 split scores: 0.8077196980487111
```

In [30]:
```
#%%se preprocesan los datos
X_train_scaled = pd.DataFrame(svc_model.named_steps['preprocessor'].fit_transform(X_train),columns = X_train.columns)
X_test_scaled = pd.DataFrame(svc_model.named_steps['preprocessor'].fit_transform(X_test),columns = X_test.columns)
#cambia el orden de edad y sexo
X_train_scaled[['EDAD','SEXO']]=X_train_scaled[['SEXO','EDAD']]
X_test_scaled[['EDAD','SEXO']]=X_test_scaled[['SEXO','EDAD']]
```

In [31]:
```
X_train_scaled.head()
```

Out[31]:

	SEXO	EDAD	EMBARAZO	DIABETES	EPOC	ASMA	INMUSUPR	HIPERTENSION	CARDIOVASCULAR	OBESIDAD	RENAL_CRONICA	TABAQUISMO
0	1.0	0.271739	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.521739	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
2	1.0	0.500000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1.0	0.543478	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	1.0	0.260870	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

In [33]:
```
X_test_scaled.head()
```

Out[33]:

	SEXO	EDAD	EMBARAZO	DIABETES	EPOC	ASMA	INMUSUPR	HIPERTENSION	CARDIOVASCULAR	OBESIDAD	RENAL_CRONICA	TABAQUISMO
0	0.0	0.623656	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
1	1.0	0.301075	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.505376	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.462366	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	1.0	0.236559	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0

In [18]:
```
y_test.iloc[20]
```

Out[18]:
```
1
```

In [17]:
```
X_test.iloc[20]
```

Out[17]:
```
SEXO               0
EDAD              26
EMBARAZO           0
DIABETES           0
EPOC               0
ASMA               0
INMUSUPR           0
HIPERTENSION       0
CARDIOVASCULAR     0
OBESIDAD           0
RENAL_CRONICA      0
TABAQUISMO         0
Name: 932525, dtype: int64
```

In [19]:
```
#%%#prueba el modelo con X_test
svc_model.named_steps['SupVM'].predict(X_test_scaled.iloc[20, :].values.reshape(1, -1))
```

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

In [20]:
```
svc_model.named_steps['SupVM'].predict_proba(X_test_scaled.iloc[20, :].values.reshape(1, -1))
```

Out[20]:
```
array([[0.91288434, 0.08711566]])
```

In [21]:
```
import shap
# use Kernel SHAP to explain test set predictions
explainer = shap.KernelExplainer(model = svc_model.named_steps['SupVM'].predict_proba, data = X_train_scaled, link = 'logit')
shap_values = explainer.shap_values(X = X_test_scaled, nsamples = 30, l1_reg="num_features(12)")
```

```
Using 1203 background data samples could cause slower run times. Consider using shap.sample(data, K) or shap.kmeans(data, K) to summarize
the background as K samples.
100%|██████████| 593/593 [20:32<00:00,  2.08s/it]
```

In [22]:
```
print(f'length of SHAP values: {len(shap_values)}')
print(f'Shape of each element: {shap_values[0].shape}')
```

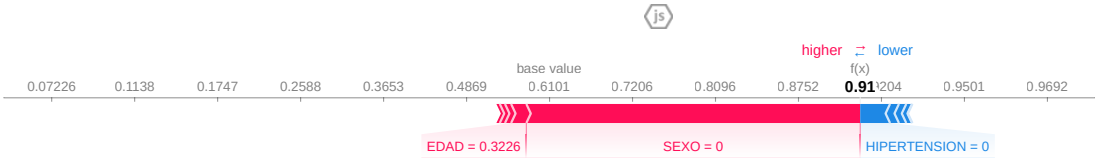
```
length of SHAP values: 2
Shape of each element: (593, 12)
```

In [23]:
```
#prediction and probability of model
print(f'Prediction for 1st sample in X_test: ', svc_model.named_steps['SupVM'].predict(X_test_scaled.iloc[[0], :])[0])
print(f'Prediction probability for 1st sample in X_test: ', svc_model.named_steps['SupVM'].predict_proba(X_test_scaled.iloc[[0], :])[0])
```

```
Prediction for 1st sample in X_test:  0
Prediction probability for 1st sample in X_test:  [0.91075977 0.08924023]
```

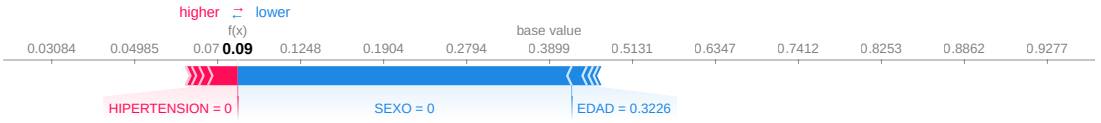
In [24]:
```
# plot the SHAP values for the false (0) output of the first instance
shap.initjs()
shap.force_plot(explainer.expected_value[0], shap_values[0][0,:], X_test_scaled.iloc[0,:], link="logit")
```

Out[24]:



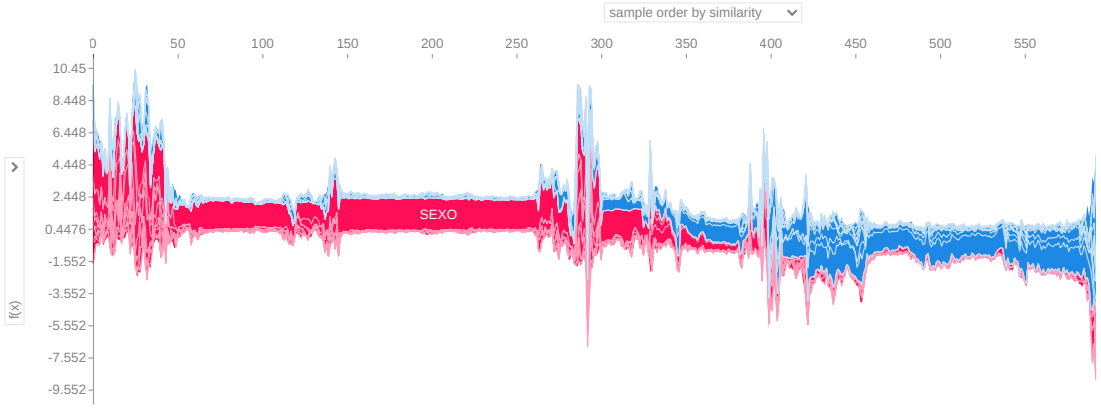
In [25]:
```
shap.initjs()
shap.force_plot(explainer.expected_value[1], shap_values[1][0,:], X_test_scaled.iloc[0,:], link="logit")
```

Out[25]:



In [26]:
```
#Explaining the Prediction for all samples in Test set
#no hospitalizado
shap.initjs()
shap.force_plot(explainer.expected_value[0], shap_values[0], X_test_scaled)
```

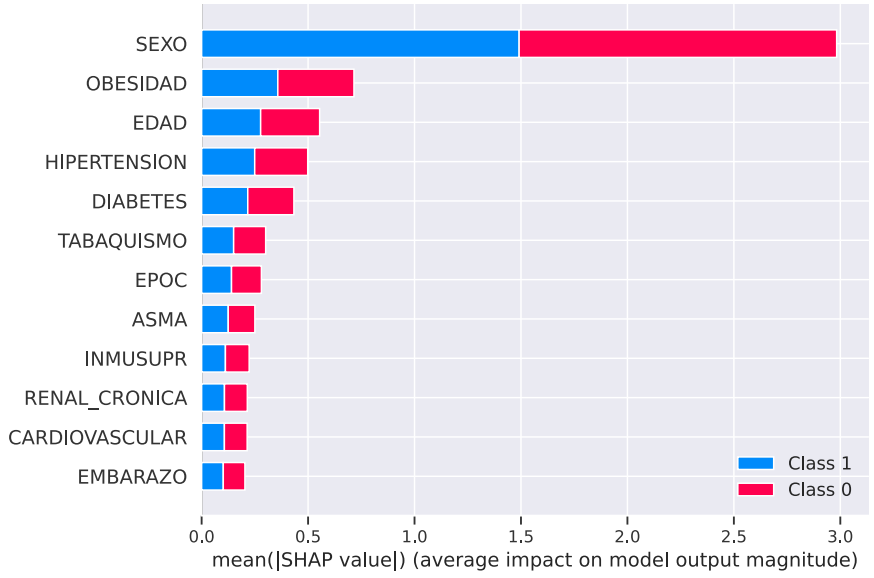
Out[26]:

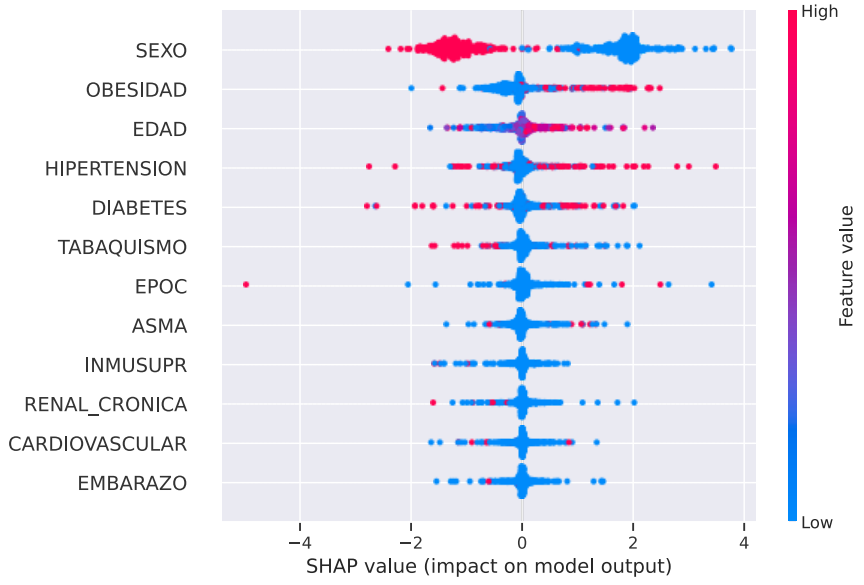


In [27]:
```
#si hospitalizado
shap.initjs()
shap.force_plot(explainer.expected_value[1], shap_values[1], X_test_scaled)
```

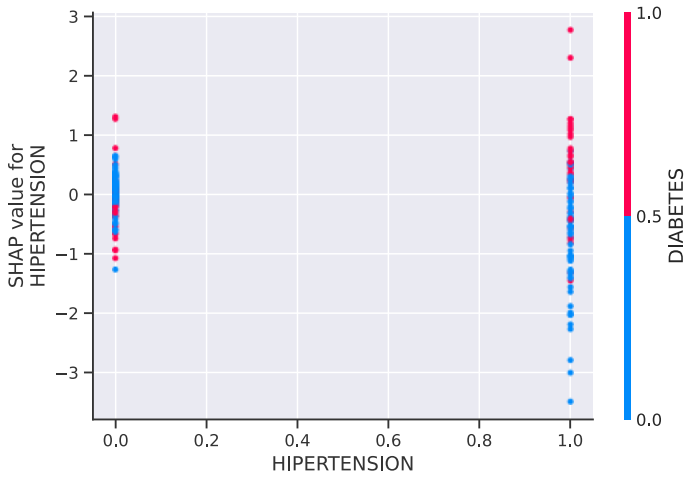
Out[27]:

In [28]:
```
#SHAP Summary Plots
shap.summary_plot(shap_values, X_test)
shap.summary_plot(shap_values[1], X_test_scaled)
shap.summary_plot(shap_values[0], X_test_scaled)
```





In [29]:
```
#SHAP Dependence Plots
shap.dependence_plot("HIPERTENSION", shap_values[1], X_test_scaled)
```



In [ ]: