Transparency

May 2, 2024

1 TRUSTWORTHY AI: Transparency Requirement

Script to ensure the Transparency requirement for a dataset. Based on EU guidelines.

1.1 TABLE OF CONTENTS:

The key requirement of transparency covers three main concepts:

- **Traceability:** documenting processes to identify the reasons why a decision was wrong and to prevent future failuress.
- **Explainability:** explain both the AI's technical processes and the associated human decision the AI decision must to be understable by a human.
- **Communication:** convey the capabilities and limitations to the end users.

We start by preparing the working environment. Import all the necessary libraries.

[24]:
```
import shap
import seaborn as sns
from datetime import datetime
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import plotly.subplots as sp
import math

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, roc_curve, auc,␣
 ↪classification_report

from functions.NaiveBayesMixed import NaiveBayesMixed
from functions.handleData import handleData
```

Load the dataset and define Metadata

```
[25]: path = 'C:\\Users\\carlo\\OneDrive - UPV\\ESCRITORIO CARLOS\\UPV\\BECA␣
       ↪COLABORACIÓN\\Datasets\\Diabetes\\'
      file_name = 'dataset_diabetes_QC.csv'
      data = pd.read_csv(path + file_name)

      # Create a subset to work more efficiently
      data = data.sample(frac = 0.2, random_state=100)
```

```
[26]: # Define Metadata
      dataset = data
      output = "readmitted"
      positive_class = "<30"
      feat_id = ["encounter_id","patient_nbr"]
      feat_sensitive = ["race","gender"]
      feat_types = {
          "race": "categorical",
          "gender": "categorical",
          "age": "numerical",
          # "weight": "categorical",
          "admission_type": "categorical",
          "discharge_disposition": "categorical",
          "admission_source": "categorical",
          "time_in_hospital": "numerical",
          # "payer_code": "categorical",
          # "medical_specialty": "categorical",
          "num_lab_procedures":"numerical",
          "num_procedures":"numerical",
          "num_medications":"numerical",
          "number_outpatient":"numerical",
          "number_emergency":"numerical",
          "number_inpatient":"numerical",
          "diag_1": "categorical",
          "diag_2": "categorical",
          "diag_3": "categorical",
          "number_diagnoses": "numerical",
          "change": "categorical",
          "diabetesMed": "categorical",
      }
      feat2balance = ["race"]
      data_provenance = "A Health Facts database that represents 10 years (1999-2008)␣
       ↪of clininical care at 130 hospitals in United States."
```

[27]:
```
# ### Uncomment only if you want to binarise a multi-class classification␣
 ↪problem ###
# class_positive = '<30'
# dataset[output] = pd.Series(np.where(dataset[output] == class_positive, 1,␣
 ↪0), index=dataset[output].index, name=output)

features_df = dataset.drop(output,axis=1)
tags_df = dataset[output]

unique_classes = np.unique(tags_df)
n_classes = len(unique_classes)
feat_num = [col for col, type in feat_types.items() if type == "numerical"]
feat_cat = [col for col, type in feat_types.items() if type == "categorical"]
feature_names = features_df.columns

handleData = handleData(feat_num, feat_cat)
```

1.2 1. DATA PREPARATION

1.2.1 1.1. Data Pre-processing:

To ensure the principle of transparency, it is important to study the data provenance as it is collected (**Data provenance**). Registering data lineage increases transparecny and reproducibility while mitigating any poisoned data.

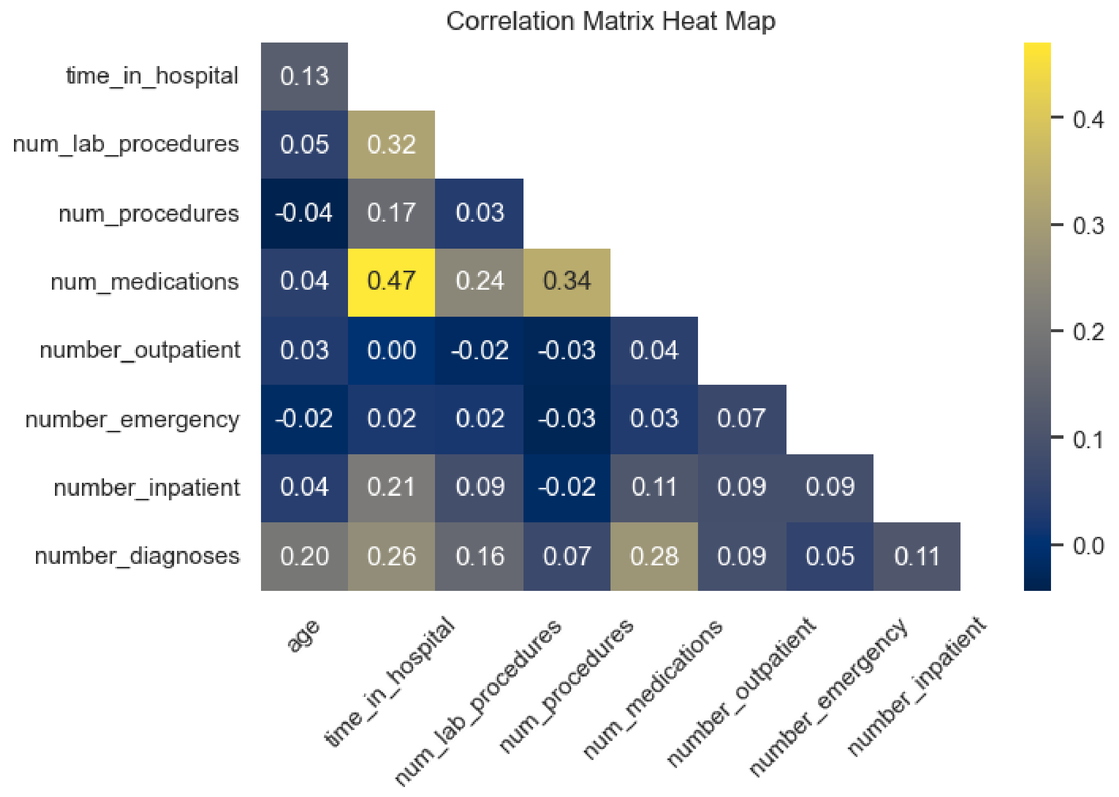
We look to see if the inicial metadata comntains any information about the provenance of the data.data provenance information.

[28]:
```
if data_provenance:
    print("The following data provenance information has been provided to␣
 ↪increase the transparency of your model:")
    print(data_provenance)
else:
    print("Data provenance is empty.")
```

```
The following data provenance information has been provided to increase the
transparency of your model:
A Health Facts database that represents 10 years (1999-2008) of clininical care
at 130 hospitals in United States.
```

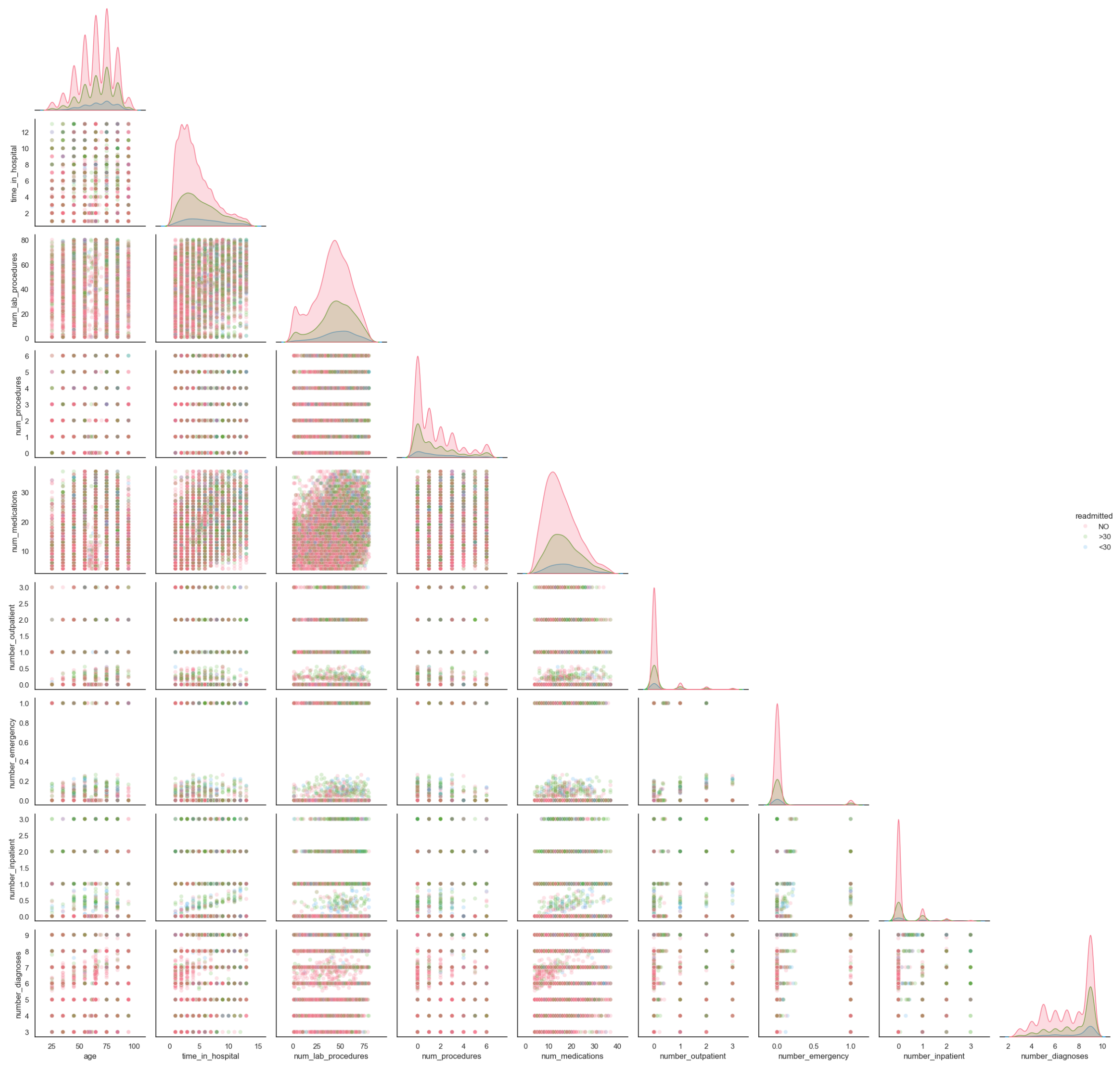
It might be interesting to add a descriptive plot to the data pre-processing stage to analyse the intrinsic relationships between the features (**General exploratory analysis**). It is a little complicated to compare numerical and categorical features, but a correct way to do this for numerical features might be to use a correlation matrix. We offer to visualise this in the form of a 'heat map' or 'pair plot':

[29]:
```python
# Calculate numerical correlation matrix
corr_num = dataset[feat_num].corr()
mask = np.triu(np.ones_like(corr_num, dtype=bool)) # A mask to hide the upper␣
  ↪right part of the matrix (it is duplicated)

plt.figure(figsize=(7, 5))
sns.heatmap(corr_num.iloc[1:, :-1], mask=mask[1:, :-1], annot=True, fmt=".2f",␣
  ↪cmap="cividis")
plt.title('Correlation Matrix Heat Map')
plt.xticks(rotation=45)
plt.show()
```



```
[30]: sns.set(style="white")

      sns.pairplot(dataset[feat_num + [output]], kind="scatter", plot_kws={"alpha": 0.
       ↪2}, diag_kind='kde', corner=True, hue=output, palette='husl')
      plt.tight_layout()
      plt.show()
```

```
The figure layout has changed to tight
```

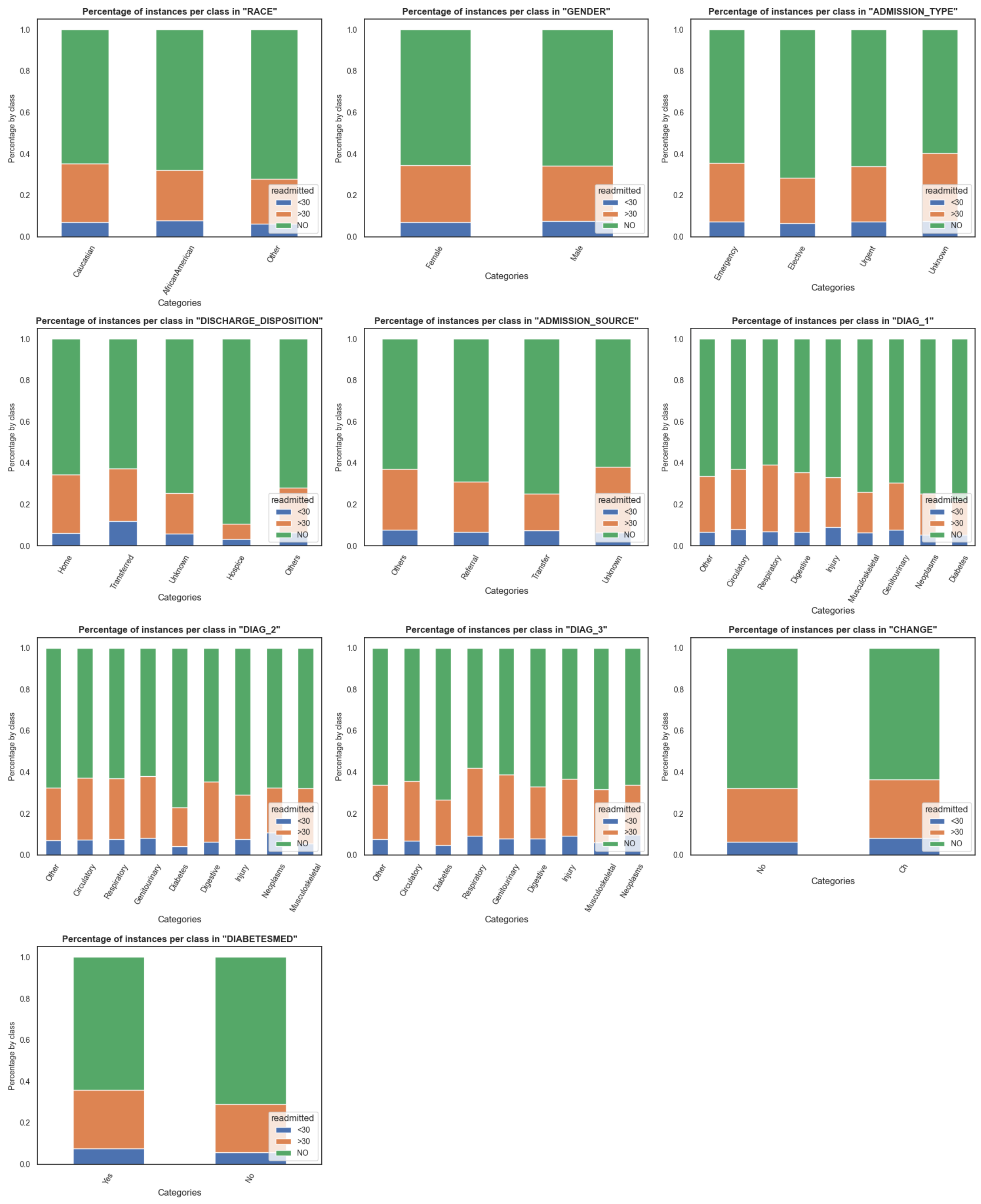


Similarly, in order to identify possible correlations between categorical features and tags, it might be useful to display a bar chart such as the following:

```
[31]: num_rows = math.ceil(len(feat_cat) / 3)

      fig, axes = plt.subplots(num_rows, 3, figsize=(18, 6*num_rows))
```

```python
# Show histogram of sensitive features by its categories
for i, feat in enumerate(feat_cat):
    row = i // 3
    column = i % 3
    ax = axes[row, column]

    sensitive_counts = features_df[feat].value_counts()
    var_type = feat_types[feat]

    unique_categories = sensitive_counts.index
    n_values = sensitive_counts.values

    # Count the number of instances per class for each category
    count_cat = {}
    for category in unique_categories:
        tags_cat = np.array(tags_df[features_df[feat] == category].squeeze())
        if tags_cat.size > 0:
            total_instances = tags_cat.size
            for classes in unique_classes:
                category_count = (tags_cat == classes).sum()
                count_cat[(classes, category)] = category_count /␣
↪total_instances  # Normalize the count

    # Reorganize the data for DataFrame construction
    class_values = {cls: [count_cat.get((cls, category), 0) for category in␣
↪unique_categories] for cls in unique_classes}

    stacked_heights_df = pd.DataFrame(class_values, index=unique_categories)
    stacked_heights_df.plot(kind='bar', stacked=True, ax=ax)

    ax.set_xlabel('Categories', fontsize=12)
    ax.set_ylabel('Percentage by class', fontsize=10)
    ax.legend(unique_classes, title=output, loc='lower right',␣
↪title_fontsize='medium', prop={'size': 10})
    ax.set_title(f'Percentage of instances per class in "{feat.upper()}"',␣
↪fontsize=12, weight='bold')
    ax.tick_params(axis='x', labelrotation=60, labelsize=10)
    ax.tick_params(axis='y', labelsize=10)

# Hide unused subplots
for i in range(len(feat_cat), num_rows * 3):
    row = i // 3
    column = i % 3
    fig.delaxes(axes[row, column])

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

```
The figure layout has changed to tight
```



1.3 2. MODEL DEVELOPMENT

1.3.1 2.1. Design:

Describe design steps: Two example models are proposed: Naive Bayes and Random Forest. Appropriate preprocessing has been incorporated. The categorical columns are coded by a dummy variable using 'OneHotEncoder'. Then the classification method is selected. In addition, the tags are encoded using Label Encoder. Then the dataset is randomly divided into a test subset with 20% of the data and a training subset with the rest. In this case, no feature selection or dimensionality reduction method is proposed. Consequently, the models will be obtained from all the features except the identifying ones.

[32]:
```
features_df_encoded, feature_names_encoded, feat_cat_encoded = handleData.
↪encode(features_df)

# Encode Tags
if not np.issubdtype(tags_df.dtype, np.number):
    encoderLabel = LabelEncoder()
    tags_df_encoded = encoderLabel.fit_transform(tags_df)
    tags_df_encoded = pd.Series(tags_df_encoded, index=tags_df.index,␣
↪name=tags_df.name)

    tags_df_mapping = {label: code for code, label in enumerate(encoderLabel.
↪classes_)} # Dictionary with the code reference for each class
    unique_classes_enc = np.unique(tags_df_encoded)
    print('Dictionary of tags: ', tags_df_mapping)
else:
    tags_df_encoded, unique_classes_enc  = tags_df, unique_classes

feat_train, feat_test, tags_train, tags_test =␣
↪train_test_split(features_df_encoded, tags_df_encoded, test_size=0.2,␣
↪random_state = 1)
```

```
Dictionary of tags:  {'<30': 0, '>30': 1, 'NO': 2}
```

1.3.2 2.2. Training and Evaluation:

RANDOM FOREST MODEL: Random Forest model from scikit-learn. As defined by the library, the model is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. In particular, this model is formed by 100 trees.

[35]:
```
model_RandomForest = RandomForestClassifier(n_estimators=100, random_state=1)

model_RandomForest.fit(feat_train, tags_train)
tags_pred = model_RandomForest.predict(feat_test)
report = classification_report(tags_test, tags_pred, zero_division=1)

print("Classification Report:\n",report)
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      1.00       207
           1       0.53      0.16      0.25       780
           2       0.67      0.95      0.79      1809

    accuracy                           0.66      2796
   macro avg       0.40      0.37      0.68      2796
weighted avg       0.58      0.66      0.65      2796
```

Particular graphics such as plot-tree are beneficial for transparency in black box models such as the one in the study.

NAIVE BAYES MODEL: The scikit-learn library doesn't implement models from scratch that are suitable for different types of data or distributions. However, it is possible to build a mixed model. Some foundations offer more complex models, for example: https://github.com/remykarem/mixed-naive-bayes/blob/master/mixed_naive_bayes/mixed_naive_bayes.py.

Nevertheless, we provide a simplified version of a mixed model consisting of a Gaussian and a Categorical that fills this gap:

[12]:
```
model_MixedNB = NaiveBayesMixed(feat_num, feat_cat_encoded)

model_MixedNB.fit(feat_train, tags_train)
tags_pred = model_MixedNB.predict(feat_test)

print("Classification Report:\n", classification_report(tags_test, tags_pred,␣
 ↪zero_division=1))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.20      0.02      0.04       207
           1       0.42      0.22      0.29       780
           2       0.68      0.89      0.77      1809

    accuracy                           0.64      2796
   macro avg       0.43      0.38      0.36      2796
weighted avg       0.57      0.64      0.58      2796
```

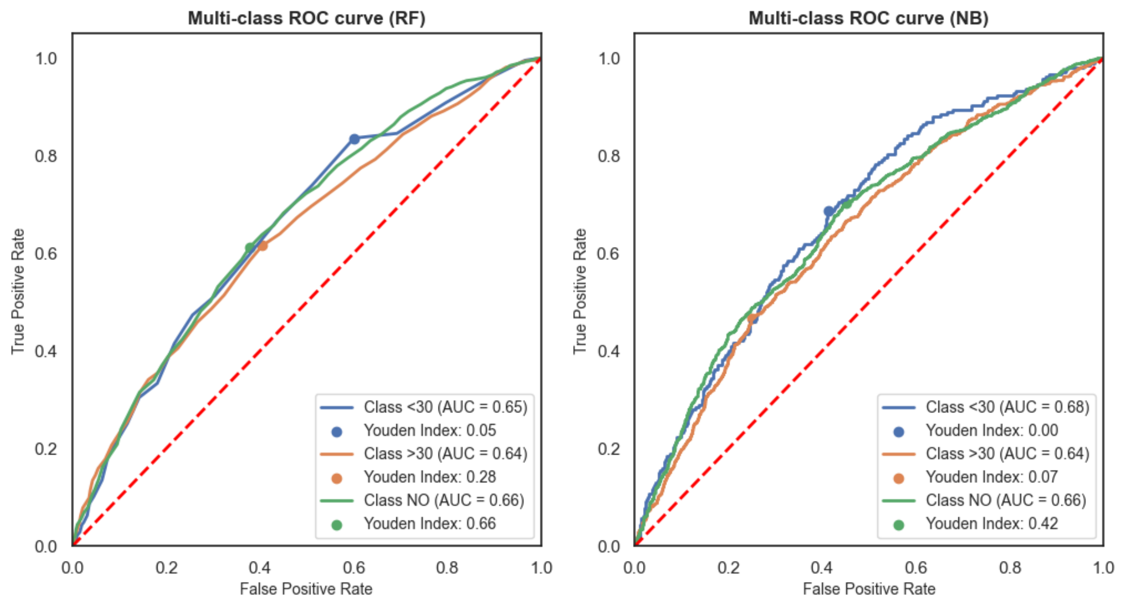
LOGISTIC REGRESSION:

[13]:
```
if n_classes == 2:
    model_LogisticRegression = LogisticRegression(max_iter=1000)
    model_LogisticRegression.fit(feat_train, tags_train)
    tags_pred = model_LogisticRegression.predict(feat_test)
    print("Classification Report:\n", classification_report(tags_test,␣
 ↪tags_pred, zero_division=1))
```

Once the model has been realised, it is possible to improve its explicability by using graphs that simplify its understandabilty (**Explainability plots**). It is important to distinguish between explicability and interpretability. While the former one refers to the explenation of the decision model, the latter focuses on understanding how it works. Consequently, the interpretability requires a deeper level of detail and will support the communication with experts, while the explicability is more superficial and suitable for end users.

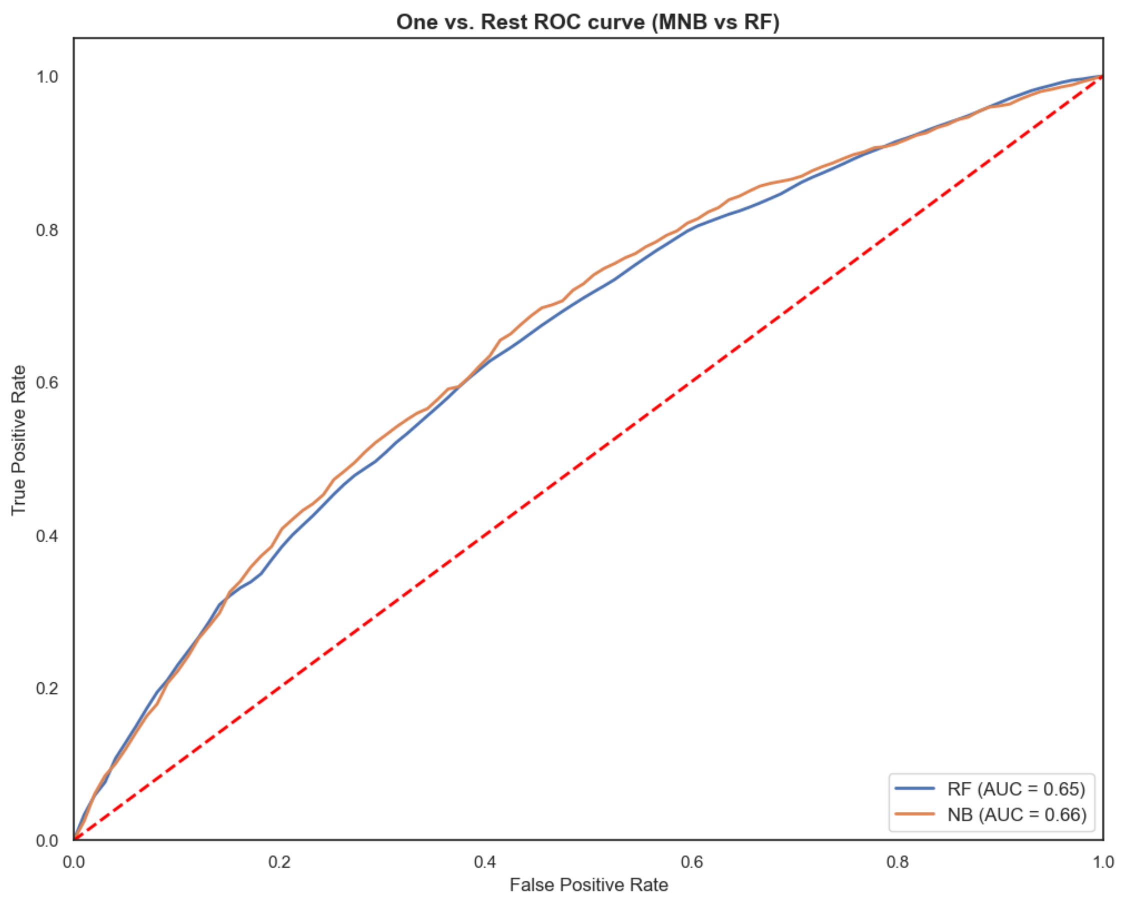
It is effective to apply a cross-validation method to estimate in a concrete way the evaluation metrics of the models. We offer some simple ones such as sensitivity and specificity, as well as others that provide more information, such as the area under a ROC curve. This graph relates sensitivity and specificity such that the higher the area, the better the prediction.

[14]:
```
classifiers = {'RF': model_RandomForest, 'NB': model_MixedNB}

fig, axs = plt.subplots(1, len(classifiers), figsize=(10, 5))

for i, (classifier_name, classifier) in enumerate(classifiers.items()):
    classifier.fit(feat_train, tags_train)

    y_scores = classifier.predict_proba(feat_test)

    if n_classes > 2:

        # Binarize tags one-vs-all
        tags_test_binarized = label_binarize(tags_test,␣
 ↪classes=unique_classes_enc)
        n_classes = len(unique_classes)

        # Calculate the ROC curve for each class
        for c in range(n_classes):
            fpr, tpr, thresholds = roc_curve(tags_test_binarized[:, c],␣
 ↪y_scores[:, c])
            roc_auc = auc(fpr, tpr)
            axs[i].plot(fpr, tpr, lw=2, label=f'Class {unique_classes[c]} (AUC␣
 ↪= {roc_auc:.2f})')
            axs[i].set_title(f'Multi-class ROC curve ({classifier_name})',␣
 ↪fontdict={'fontsize': 12, 'weight': 'bold'})

            # Calculate Youden index
            youden_index = thresholds[np.argmax(tpr - fpr)]
            axs[i].scatter(fpr[np.argmax(tpr - fpr)], tpr[np.argmax(tpr -␣
 ↪fpr)], marker='o', label=f'Youden Index: {youden_index:.2f}')

    else:
        fpr, tpr, thresholds = roc_curve(tags_test, y_scores[:, 1])
        roc_auc = auc(fpr, tpr)
        axs[i].plot(fpr, tpr, lw=2, color='tab:blue', label=f'{output} (AUC =␣
↪{roc_auc:.2f})')
        axs[i].set_title(f'ROC curve ({classifier_name})', fontdict={'fontsize':
↪ 12, 'weight': 'bold'})

        # Calculate Youden index
        youden_index = thresholds[np.argmax(tpr - fpr)]
        axs[i].scatter(fpr[np.argmax(tpr - fpr)], tpr[np.argmax(tpr - fpr)],␣
↪marker='s', color='tab:blue', label=f'Youden Index: {youden_index:.2f}')

    axs[i].plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
    axs[i].set_xlim([0.0, 1.0])
    axs[i].set_ylim([0.0, 1.05])
    axs[i].set_xlabel('False Positive Rate', fontdict={'fontsize': 10})
    axs[i].set_ylabel('True Positive Rate', fontdict={'fontsize': 10})
    axs[i].legend(loc="lower right", prop={'size': 10})

plt.show()
```



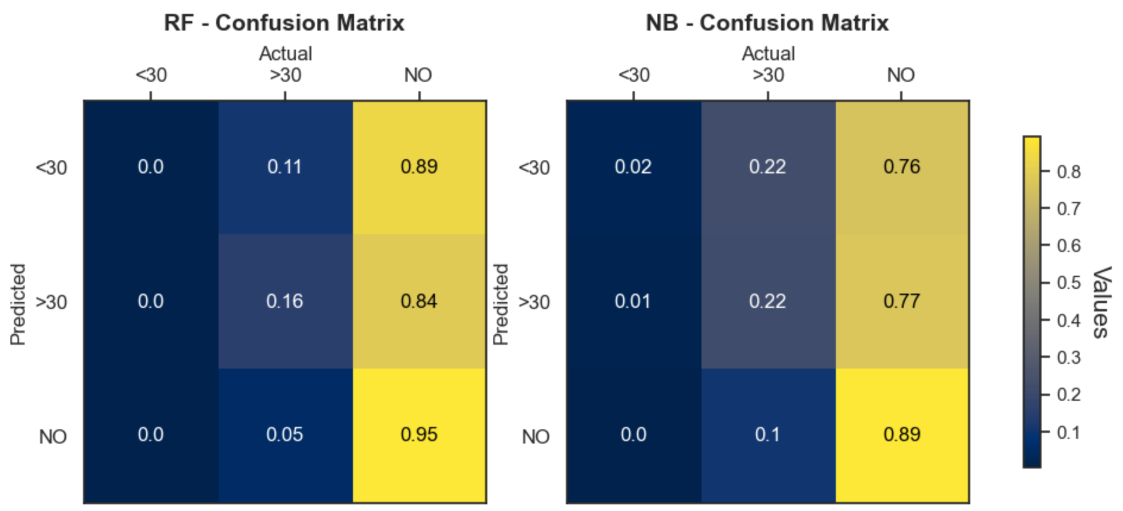
To compare the behaviour of the two models, it is useful to plot the ROC curve of each model for ONE vs. ALL.

[33]:
```
if n_classes > 2:
    plt.figure(figsize=(6, 5))
    for classifier_name, classifier in classifiers.items():
        tpr_list = []
        mean_fpr = np.linspace(0, 1, 100)  # Common set of points

        y_scores = classifier.predict_proba(feat_test)

        for i in range(n_classes):
            fpr, tpr, _ = roc_curve(tags_test_binarized[:, i], y_scores[:, i])

            # Interpolate the ROC curve to have a common set of points
            interp_tpr = np.interp(mean_fpr, fpr, tpr)
            interp_tpr[0] = 0.0  # Set the first point to (0, 0)
            tpr_list.append(interp_tpr)

        # Calculate the average ROC curve
        mean_tpr = np.mean(tpr_list, axis=0)
        roc_auc_mean = auc(mean_fpr, mean_tpr)

        # Plot ROC curve
        plt.plot(mean_fpr, mean_tpr, lw=2, label=f'{classifier_name} (AUC =␣
↪{roc_auc_mean:.2f})')

    plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate', fontdict={'fontsize': 12})
    plt.ylabel('True Positive Rate', fontdict={'fontsize': 12})
    plt.title('One vs. Rest ROC curve (MNB vs RF)', fontdict={'fontsize': 14,␣
↪'weight': 'bold'})
    plt.legend(loc="lower right", prop={'size': 12})
    plt.show()
```



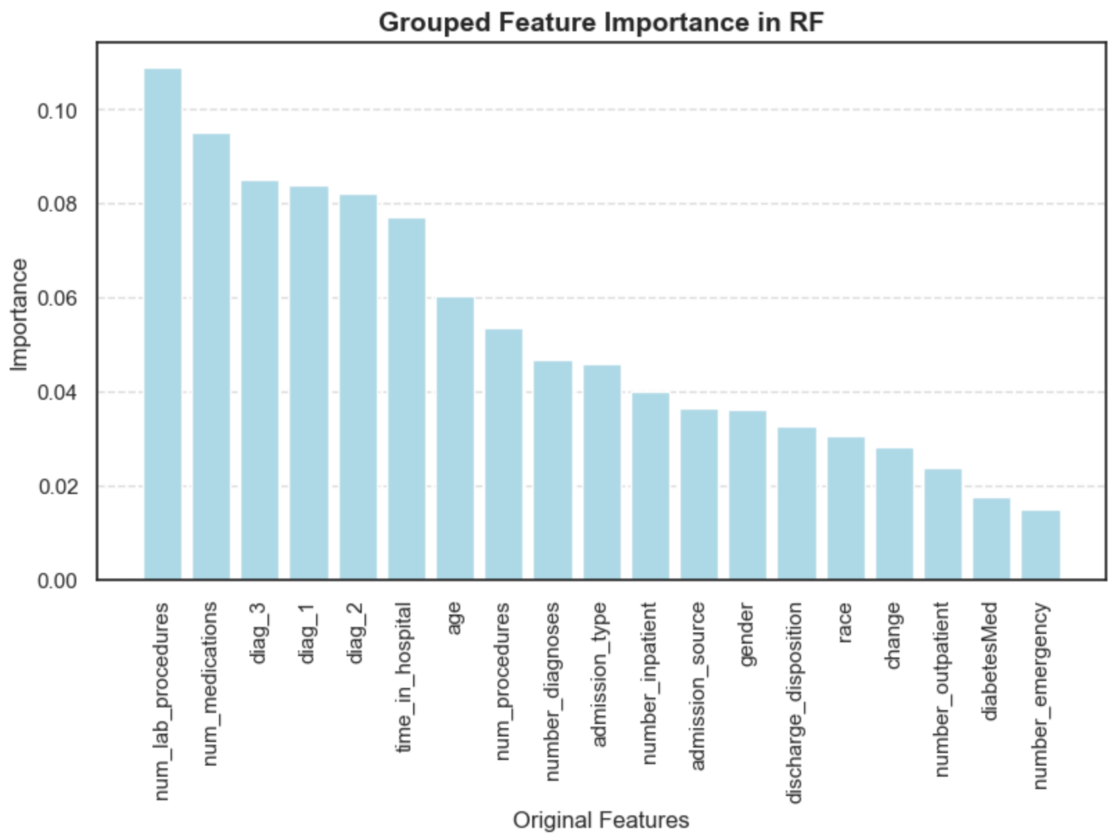
We also provide a confusion matrix for both models. This graph compares the predicted classes with the real ones. Ideally, we look for an identity matrix with a main diagonal of probability 1. In addition to making the tools as suitable as possible for the maximum number of groups, we use a ‘cividis’ colour palette ranging from soft yellow tones to a more intense blue suitable for daltonics.

[16]:
```
fig, axs = plt.subplots(1, 2, figsize=(12, 6))

plt.rcParams['figure.constrained_layout.use'] = True

for i, (classifier_name, classifier) in enumerate(classifiers.items()):

    # Calculate and normalise the confusion matrix
    tags_pred = classifier.predict(feat_test)
    conf_matrix = confusion_matrix(tags_test, tags_pred,␣
 ↪labels=unique_classes_enc)
    conf_matrix = conf_matrix.astype('float') / conf_matrix.sum(axis=1)[:, np.
 ↪newaxis]

    im = axs[i].imshow(conf_matrix, cmap='cividis')
    # Loop over data dimensions and create text annotations.
    for j in range(len(unique_classes)):
        for k in range(len(unique_classes)):
            color = "black" if round(conf_matrix[j, k], 2) > 0.5 else "white"
            text = axs[i].text(k, j, round(conf_matrix[j, k], 2), ha="center",␣
↪va="center", color=color)

    axs[i].set_xticks(np.arange(len(unique_classes)))
    axs[i].set_yticks(np.arange(len(unique_classes)))
    axs[i].set_xticklabels(unique_classes, fontdict={'fontsize':12})
    axs[i].set_yticklabels(unique_classes, fontdict={'fontsize':12})  # Invert␣
↪Y axis labels
    axs[i].set_title(f"{classifier_name} - Confusion Matrix",␣
↪fontdict={'fontsize':14, 'weight': 'bold'}, y=1.15)
    plt.setp(axs[i].get_xticklabels(), rotation=45, ha="right",␣
↪rotation_mode="anchor")

    # Set labels for x and y axes
    axs[i].set_xlabel("Actual", fontdict={'fontsize': 12})
    axs[i].set_ylabel("Predicted", fontdict={'fontsize': 12})

# Invert X axis labels
for ax in axs:
    ax.xaxis.set_ticks_position('top')
    ax.xaxis.set_label_position('top')

# Add colorbar
cbar = fig.colorbar(im, ax=axs.ravel().tolist(), shrink=0.6)
cbar.set_label('Values')
cbar.ax.set_ylabel('Values', rotation=-90, va="bottom", fontsize=15)

plt.show()
```



It might be useful to use graphs that show the importance of features in making predictions, such as the graph of functions associated with the models or another more complex one such as SHAP (SHapley Additive exPlanations).

```
[21]: for classifier_name, classifier in classifiers.items():
          if classifier_name == 'NB':
              print("\nFor Mixed Naive Bayes, try other methods such as Shapley")

          elif classifier_name == 'RF':

              feature_importance = classifier.feature_importances_
              sorted_grouped_features = handleData.
       ↪group_feature_importance(feature_names_encoded, feature_importance,␣
       ↪feat_cat_encoded)
              sorted_features, importances = zip(*sorted_grouped_features)

              plt.figure(figsize=(8, 6))
              plt.bar(range(len(importances)), importances, align='center',␣
       ↪color='lightblue')
              plt.xticks(range(len(importances)), sorted_features, rotation=90)
              plt.grid(axis='y', linestyle='--', alpha=0.7)
              plt.xlabel('Original Features', fontdict={'fontsize':12})
              plt.ylabel('Importance', fontdict={'fontsize':12})
              plt.title(f'Grouped Feature Importance in {classifier_name}',␣
       ↪fontdict={'fontsize':14, 'weight': 'bold'})
              plt.show()
```



```
For Mixed Naive Bayes, try other methods such as Shapley
```

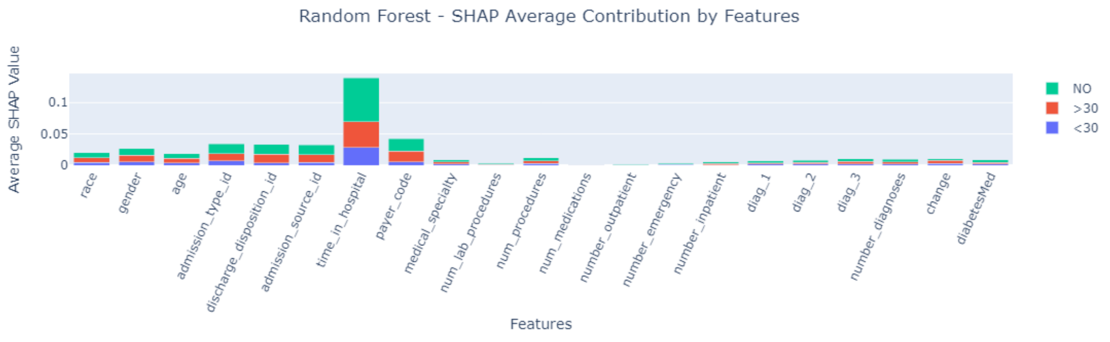
1.4 3. DEPLOYMENT and USE

Apart from model development, analyse how the model deals with unexpected problems. We offer different Shapley plots (**Explainability plots after training**).

```
[ ]: if n_classes > 2:

         # Calculate SHAP values
         explainer = shap.TreeExplainer(model_RandomForest, feat_test[:20])
         shap_values = explainer.shap_values(feat_test)

         # Calculate average importance of feature by class
         shap_sums = np.abs(shap_values).mean(axis=1)

         fig = go.Figure()

         # Add a bar for each class
         for i in range(len(shap_values)):
```

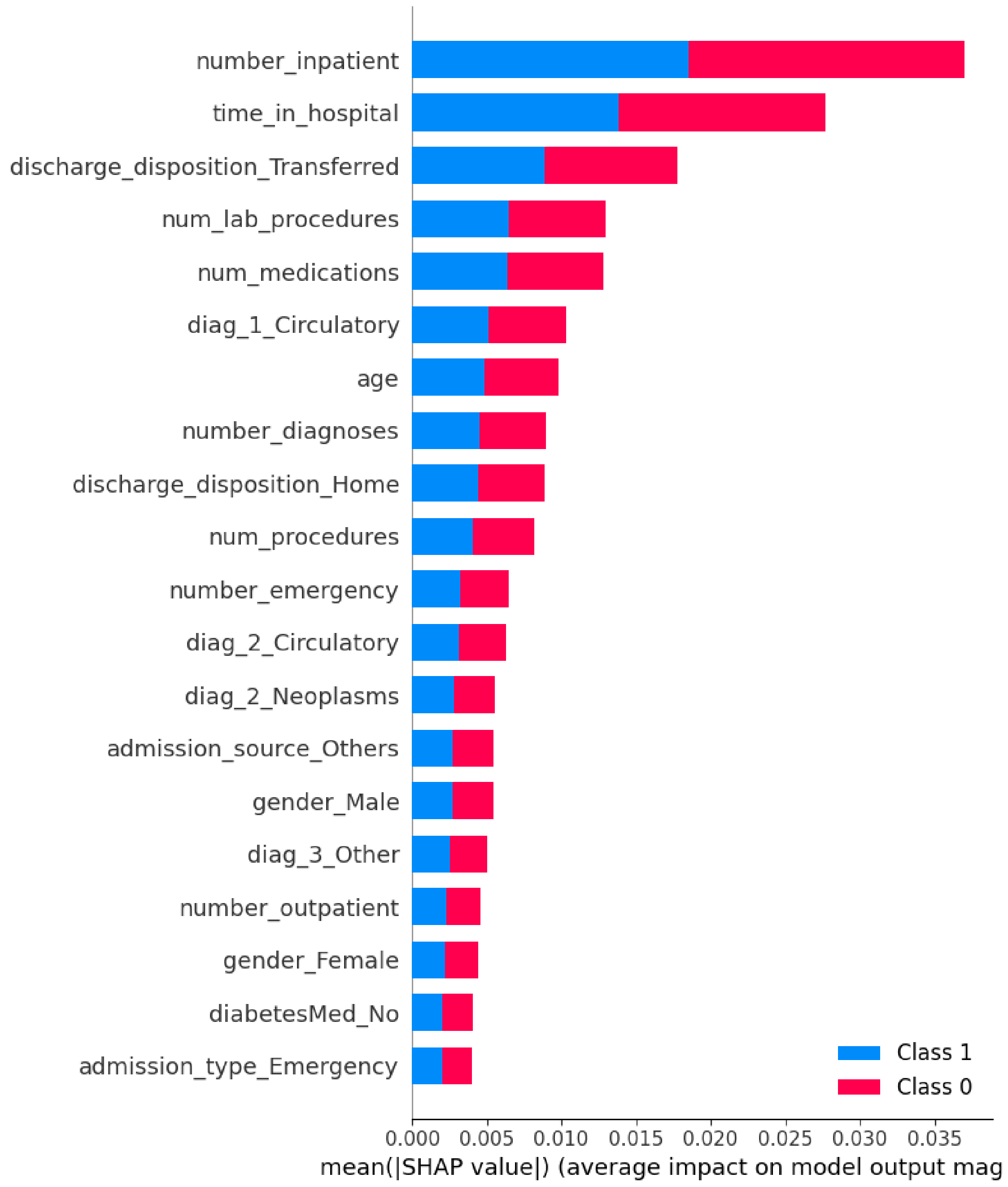
```
        fig.add_trace(go.Bar(x=feature_names, y=shap_sums[i],␣
↪name=f'{unique_classes[i]}'))

    # Update plot design
    fig.update_layout(
        barmode='stack',
        title="Random Forest - SHAP Contribution by Features",
        title_x=0.5,
        xaxis_title="Features",
        yaxis_title="Average SHAP Value",
        xaxis_tickangle=-65
    )

    fig.show()
```

```
100%|===================| 12203/12213 [03:22<00:00]
```



```
[ ]: if n_classes == 2:
         # Calculate SHAP values
         explainer = shap.TreeExplainer(model_RandomForest, feat_test[:20])
         shap_values = explainer.shap_values(feat_test)

         # Calculate average importance of feature by class
         shap_sums = np.abs(shap_values).mean(axis=1)

         fig = go.Figure()
         shap.summary_plot(shap_values, feat_test,␣
     ↪feature_names=feature_names_encoded, plot_type="bar")
```

```
100%|===================| 27938/27954 [10:01<00:00]
```

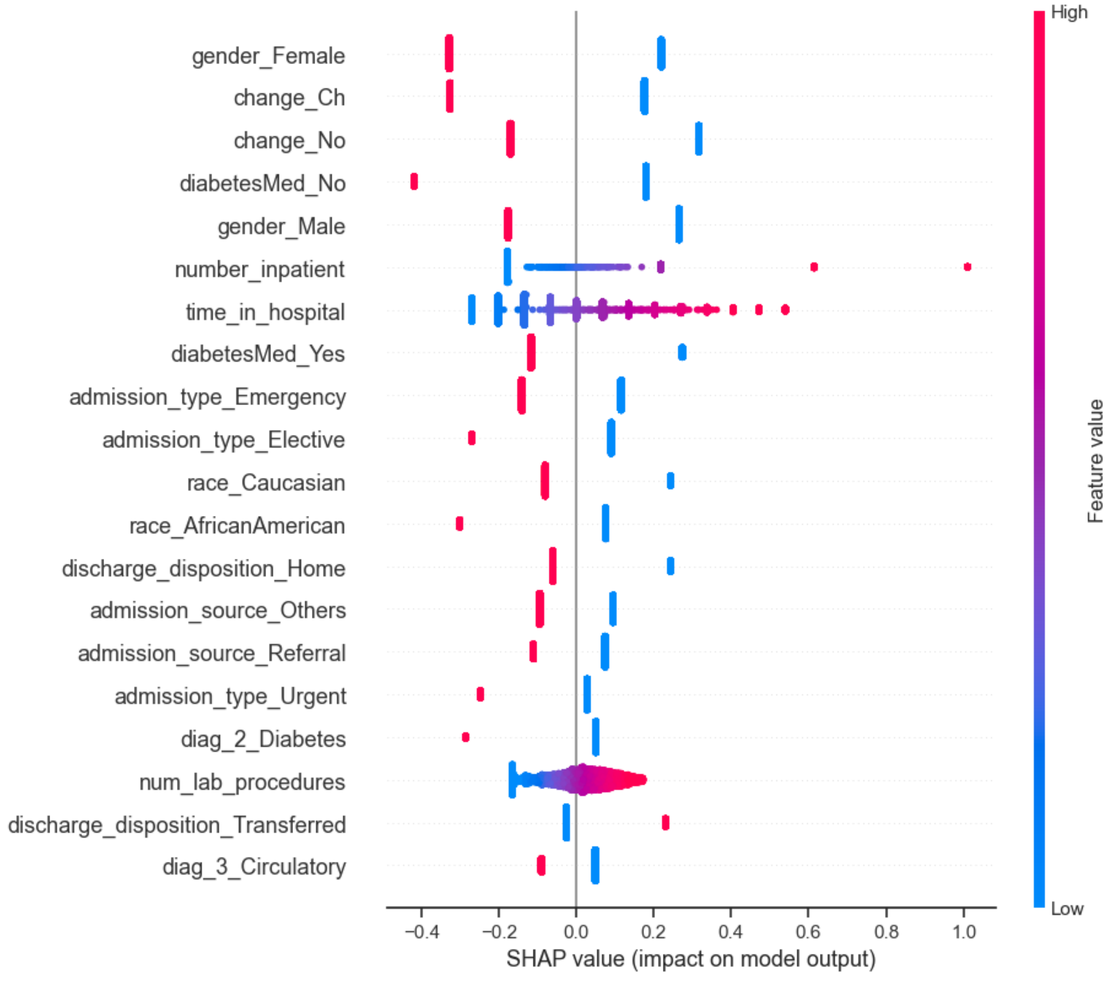


```
[ ]: if n_classes == 2:

         # Calculate shap values
         explainer = shap.Explainer(model_LogisticRegression, feat_test[:20])
         shap_values = explainer.shap_values(feat_test)

         fig = go.Figure()
```

```
try:
    # Plot a SHAP Summary Chart
    shap.summary_plot(shap_values, feat_test,␣
↪feature_names=feature_names_encoded)
    plt.tight_layout()
except RuntimeError as e:
    if "Colorbar layout of new layout engine not compatible" in str(e):
        pass  # Ignore the error related to the colorbar layout
    else:
        raise e
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
Cell In[21], line 10
      7 fig = go.Figure()
      9 # Plotear un gráfico de resumen de SHAP
---> 10␣
 ↪shap.summary_plot(shap_values, feat_test, feature_names=feature_names_encoded
     12 # shap.plots.scatter(shap_values[:, 9], color=shap_values)

File c:
 ↪\Users\carlo\AppData\Local\Programs\Python\Python312\Lib\site-packages\shap\plots\_beeswarm
 ↪py:960, in summary_legacy(shap_values, features, feature_names, max_display,␣
 ↪plot_type, color, axis_color, title, alpha, show, sort, color_bar, plot_size,␣
 ↪layered_violin_max_num_bins, class_names, class_inds, color_bar_label, cmap,␣
 ↪show_values_in_legend, auto_size_plot, use_log_scale)
    958 else:
    959     pl.xlabel(labels['VALUE'], fontsize=13)
--> 960 pl.tight_layout()
    961 if show:
    962     pl.show()

File ~\AppData\Roaming\Python\Python312\site-packages\matplotlib\pyplot.py:2599 ␣
 ↪in tight_layout(pad, h_pad, w_pad, rect)
   2591 @_copy_docstring_and_deprecators(Figure.tight_layout)
   2592 def tight_layout(
   2593     *,
   (…)
   2597     rect: tuple[float, float, float, float] | None = None,
   2598 ) -> None:
-> 2599     gcf().tight_layout(pad=pad, h_pad=h_pad, w_pad=w_pad, rect=rect)

File ~\AppData\Roaming\Python\Python312\site-packages\matplotlib\figure.py:3539 ␣
 ↪in Figure.tight_layout(self, pad, h_pad, w_pad, rect)
   3537 try:
   3538     previous_engine = self.get_layout_engine()
-> 3539     self.set_layout_engine(engine)
```

```
    engine.execute(self)
    if previous_engine is not None and not isinstance(
        previous_engine, (TightLayoutEngine, PlaceHolderLayoutEngine)
    ):

File ~\AppData\Roaming\Python\Python312\site-packages\matplotlib\figure.py:2674 ␣
 ↪in Figure.set_layout_engine(self, layout, **kwargs)
    self._layout_engine = new_layout_engine
else:
-> 2674     raise RuntimeError('Colorbar layout of new layout engine not '
                       'compatible with old engine, and a colorbar '
                       'has been created.  Engine not changed.')

RuntimeError: Colorbar layout of new layout engine not compatible with old␣
 ↪engine, and a colorbar has been created.  Engine not changed.
```



To inform about the model, a ‘disclaimer’ template like the following one could be used (**Disclaimer**

communication):

DISCLAIMER FOR TRUSTWORTHY AI MODEL [Model Name]

This trustworthy AI model, [Model Name], has been meticulously developed for [intended use cases]. It underwent extensive training on [describe the data] and consistently demonstrates a high level of accuracy, with performance metrics indicating [mention performance metrics] when assessed in [describe the context or domain]. The model has undergone rigorous testing for fairness and robustness, although it's important to acknowledge [include any limitations regarding fairness, bias, and robustness].

[Model Name] employs inherently [transparent/ not transparent] algorithms to the end-user, and we employ [describe any interpretability tools or methods used] to ensure transparency and interpretability. Data privacy and security are paramount, and we uphold strict measures to safeguard user data in compliance with regulations.

While [Model Name] has been carefully designed to minimize biases and ethical concerns, users are urged not to rely solely on it for [critical decisions/ any sensitive use cases]. It is imperative to exercise caution when interpreting the model's predictions and, when appropriate, consult with a qualified human expert for review.

To maintain your trust in [Model Name], we are dedicated to routine updates and maintenance, ensuring it remains aligned with the latest data and best practices. Should you have any inquiries, concerns, or encounter any issues, please do not hesitate to reach out to us at [contact information].

Please take note that [Company/Developer Name] cannot be held liable for any harm or damage resulting from the use of [Model Name] outside of its designated use cases and recommendations.

[Include any additional specific disclaimers related to the model].

1.5 4. MANAGEMENT

1.5.1 4.1. Documentation:

In addition, a continuous documentation of the results (**Output ocumentation**) should be carried out to allow trazability and increase transparency, so that it is possible to identify the reasons why a decision was wrong. At the same time, it is necessary to establish a communication with the end users , so that they are fully aware of the features, limitations and shortcomings of the AI.

Considering a "Recordings" folder to store the registers, there is an example of how to register the metrics of the Random Forest model metrics.

[45]:
```
# Extract metrics from RF classification report
report_lines = report.split('\n')
metrics = report_lines[-2].split()

precision = float(metrics[2])
recall = float(metrics[3])
f1 = float(metrics[4])
support = int(metrics[5])
hour = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
```

```python
data = pd.DataFrame({
    'Precision': [precision],
    'Recall': [recall],
    'F1 Score': [f1],
    'Support': [support],
    'Hour': [hour]
})

file_hour = datetime.now().strftime('%Y%m%d_%H%M%S')
file_name = f'Recordings/recording_{file_hour}.csv'
data.to_csv(file_name, index=False)

print(f"Data registered in {file_name} at {data['Hour'].iloc[0]}")
```

```
Data registered in Recordings/recording_20240502_121723.csv at 2024-05-02
12:17:23
```

1.5.2 4.3. Cooperation and Incident sharing:

In addition, it might be positive to propose a template to reporting incidents (**Incident sharing**). This could include data such as user's information, details of the incident (date, time and location), and a brief description of the incident or the possible impact.