IREX: A reusable process for the iterative refinement and explanation of classification models.

Import all necessary libraries

Of the following imports, the most important libraries are: sklearn.neural_network and sklearn.metrics.

In [1]:
```
import joblib
import dice_ml
import shap
import lime
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from operator import itemgetter
from sklearn.model_selection import train_test_split, cross_val_score
from explainerdashboard import ClassifierExplainer
from sklearn.metrics import ConfusionMatrixDisplay, precision_recall_fscore_suppo
from sklearn.metrics import confusion_matrix, accuracy_score, mean_squared_error
from sklearn.neural_network import MLPClassifier
from imblearn.over_sampling import SMOTE
from alibi.explainers.ale import ALE, plot_ale
```

Defining global variables

The following variables defined below can be modified according to the given approach.

In [2]:
```
## The dataset is defined with the model to explain.
path_dataset = "Dataset_MO_ENG.csv"

## The dataset with the expected responses is defined.
path_dataset_qa = "Dataset_QA_ENG.csv"

## The column, shape and grouping name of the dataset to be explained are defined
grouping_name = 'Target' # *
class_dic = {1: 0 , 2: 0, 3:1, 4:2, 5:2}

## * If this data is None or empty string then class_dic will be ignored.

## The class names present in the dataset to be explained are defined.
target_names = ["Low", "Medium", "High"] # *

## * If grouping_name is None or empty, then a class name must exist.
##   If not, then the total class name must agree with the total groupings define

## The class names present in the dataset to be plotted are defined.
name_class = ["Class Low", "Class Medium", "Class High"] # *
list_color = ['gray', 'black', 'red']

## * If grouping_name is None or empty, then a class name must exist.
##   If not, then the total class name must agree with the total groupings define

## The items to exclude from the dataset are defined.
unrelated_questions = [102, -1] # *

## * If this data is an empty list then this data will be ignored.

## The list of items eliminated at the start of the first iteration is defined.
list_contr = [3, 4, 5, 14, 21, 24, 25, 26, 27, 29, 30, 32, 48, 49, 51, 54, 55,
              58, 59, 60, 62, 63, 66, 68, 70, 75, 78, 83, 85, 86, 87, 89, 90,
              91, 94, 95, 96, 98, 100, 101]

## The regular and alpha parameters of the training are defined.
regul_param_normal = 10.0 ** - np.arange(-2, 7)
regul_param_alpha = 10.0 ** - np.arange(0, 7)

## The alpha, random state and learning rate training parameters are defined for
learning_rate_init = 0.1
random_state_model = 42
alpha = 0.001

## Positive and negative threshold values are defined for the slope qualification
positive_threshold = 0.01
negative_threshold = -0.01

## The qualification mode of each answer obtained is defined.
selection_mode = 1 # *

## * The following modes are available for the qualification the answers present
##
##     1. Selection of a single class.
##     2. Selection of multiple classes with OR type condition.
##     3. Selection of multiple classes with AND type condition.
```

```
##     4. Selection of multiple classes with AND type condition when they are Out

## The position of the class to qualify is defined.
select_class_mode = 2 # *

## * Position is related to the number of classes present in target_names and is
##   selection_mode is set to 1.

## The positions of the classes to qualify are defined.
selection_class = [0, 2] # *

## * Positions are related to the number of classes present in target_names and i
##   selection_mode is set to 2, 3 and 4.

## The map of expected responses is defined.
expected_response_map = [[0, 1, 0, 1], [1, 0, 1, 0]] # *
## * A two-dimensional list is defined, where each element of is a list of 4 nume
##   to 0 and 1. The number of elements of this is given by the number of classes
##
##   The 4 numerical elements must meet the following conditions:
##
##       Element 1. Score is 1 and expected response is 0.
##       Element 2. Score is 1 and expected response is 1.
##       Element 3. Score is 0 and expected response is 1.
##       Element 4. Score is 0 and expected response 0.
```

Defining private variables

The following variables defined below cannot be modified since they are an integral part of its operation.

In [3]:
```
## The datasets for storing the auxiliary data are defined.
path_dataset_qs = "Dataset_QS_ENG.csv"
path_dataset_qd= "Dataset_QD_ENG.csv"

## The oversample function used is defined.
random_state = 13
oversample = SMOTE(random_state = random_state)

## The seed used for training the neural network is defined.
seed = 1

## The number of iterations performed in the explanation is defined.
iteration = 0

## The names of the explanation methods used are defined.
name_methods = ["Importance", "LIME", "SHAP", "ALE"]
```

Preparing the data for the neural network

The data is prepared in order to be processed by the neural network.

In [4]:

```
## Loading data from the dataset.
df = pd.read_csv(path_dataset)

## Elimination of unrelated items from the dataset.
if len(unrelated_questions) != 0:
    df = df.drop(df.columns[unrelated_questions[0]:unrelated_questions[1]], axis=

## Grouping of the target according to the defined dictionary.
if grouping_name != None and grouping_name != '':
    df[grouping_name] = df[grouping_name].map(class_dic)

## Assignment of local variables according to the data necessary for the neural r
train_cols = df.columns [0:-1]
label = df.columns [-1]
X = df [train_cols]
y = df [label]

## Generation of the question status list automatically.
question = list(range(1, df.shape[1]))
status = ['In use' for _ in range(1, df.shape[1])]
ds = pd.DataFrame({'Question': question, 'Status': status})

## Save the changed statuses in the dataset.
ds.to_csv(path_dataset_qs, index = False)

## Generation of the data structure by iteration automatically.
columns = ['Question', 'Acurracy global']

for index in range(0, len(name_class)):
    columns.append('Precision_' + str(index))
    columns.append('Recall_' + str(index))
    columns.append('F1_score_' + str(index))
    columns.append('Support_' + str(index))

dd = pd.DataFrame(columns = columns)

## Save the changed data in the dataset.
dd.to_csv(path_dataset_qd, index = False)

## The prepared data of the dataset is printed.
df
```

Out[4]:

	1. Most of the time I have difficulty concentrating on simple tasks	2. I don't feel like doing my daily duties	3. My friends or family have told me that I look different	4. When I think about the future it is difficult for me to imagine it clearly	5. People around me often ask me how I feel	6. I consider that my life is full of good things	7. My hobbies are still important to me	8. I'm still as punctual as I have always been	9. If I had the chance, I would spend all day in my bed	10. tha s sc \ w sea sto an par
0	1	1	0	1	0	1	1	1	0	
1	1	0	1	1	0	1	1	1	0	
2	0	0	0	0	1	1	1	0	0	
3	0	0	0	1	0	1	1	1	1	
4	1	1	0	1	0	1	0	1	1	
...	...	...	...	...	...	...	...	...	...	
99	1	1	1	0	0	1	1	1	0	
100	1	1	0	0	1	1	1	0	0	
101	0	0	1	1	1	1	1	0	1	
102	1	0	0	0	1	1	1	1	1	
103	1	1	0	0	0	1	1	0	0	

104 rows × 103 columns

Oversample application

An oversample process is applied to level the amount of existing data given by the grouping.

In [5]:
```
## The number of elements present in the grouped target is printed.
y.value_counts()
```

Out[5]:
```
0    39
1    35
2    30
Name: Target, dtype: int64
```

Oversample application.

In [6]:
```
## Application of the oversample to the prepared data.
X, y = oversample.fit_resample(X, y)
```

Printing of the data after the application of the oversample.

In [7]:
```
## The results obtained after applying the oversample are printed.
y.value_counts()
```

Out[7]:
```
0    39
1    39
2    39
Name: Target, dtype: int64
```

Preparation of the parameters of the neural network

We proceed to define the parameters necessary for the training of the neural network.

In [8]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
```

Neural network training

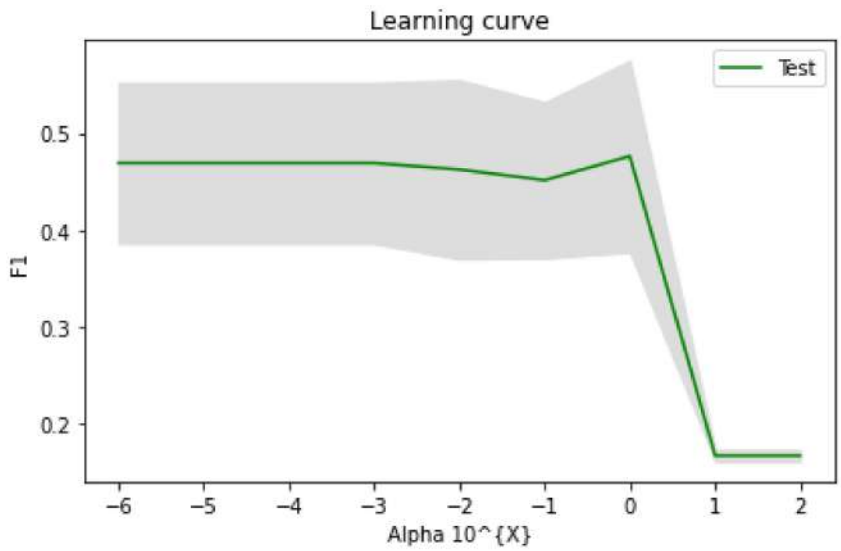
The alpha parameters and the initial learning rate are adjusted to obtain the best performance with the model using cross validation.

In [9]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_normal

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = 0.0001, max_iter =

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[9]:
```
([0.16659498207885304,
  0.16659498207885304,
  0.4763355552829237,
  0.4513935101705999,
  0.46248242125951106,
  0.4691799342728135,
  0.4691799342728135,
  0.4691799342728135,
  0.4691799342728135],
 [0.006020651927912844,
  0.006020651927912844,
  0.09918844066715299,
  0.08068804784278276,
  0.09270871998125078,
  0.08309589200633044,
  0.08309589200633044,
  0.08309589200633044,
  0.08309589200633044])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [10]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Alpha 10^{X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



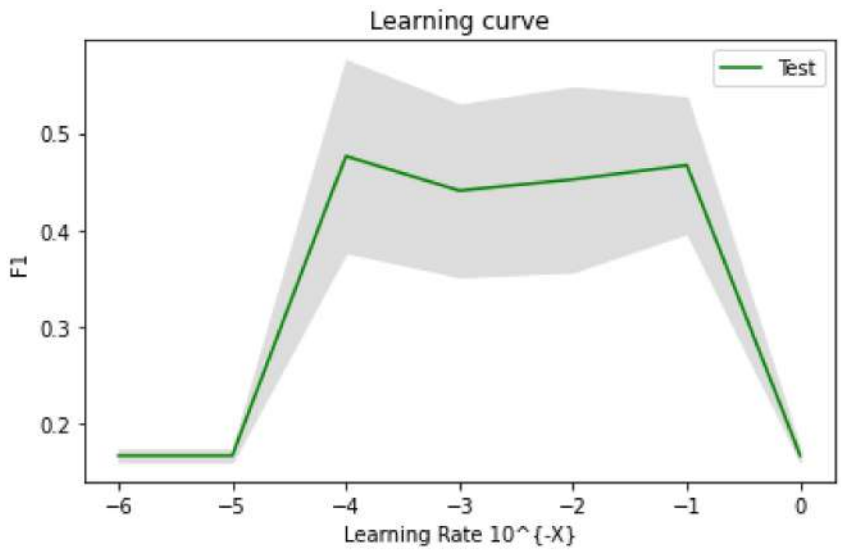
The training is performed again keeping the same parameters except for the alpha value that changes to 1.

In [11]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_alpha

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = regul_param, max_it

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[11]:
```
([0.16659498207885304,
  0.46703181808444966,
  0.45220030972352954,
  0.4405739461776614,
  0.4763355552829237,
  0.16659498207885304,
  0.16659498207885304],
 [0.006020651927912844,
  0.0700006268704352,
  0.09521251410308869,
  0.0887293445552907,
  0.09918844066715299,
  0.006020651927912844,
  0.006020651927912844])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [12]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Learning Rate 10^{-X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



Generation of the neural network model

The parameters of the final model are modified according to the data obtained in the training of the neural network.

In [13]:
```
## The value of the alpha parameter is kept at 1 as well as the value of the seed
mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solver =
             learning_rate = 'constant', learning_rate_init = 0.0001, max_iter =

## We print the data of the final generated model.
mlp.fit(X_train, y_train)
```

Out[13]:
```
MLPClassifier
MLPClassifier(activation='logistic', alpha=1, hidden_layer_sizes=(10,),
              learning_rate_init=0.0001, max_iter=100000, random_state=77)
```

We safeguard the final generated model

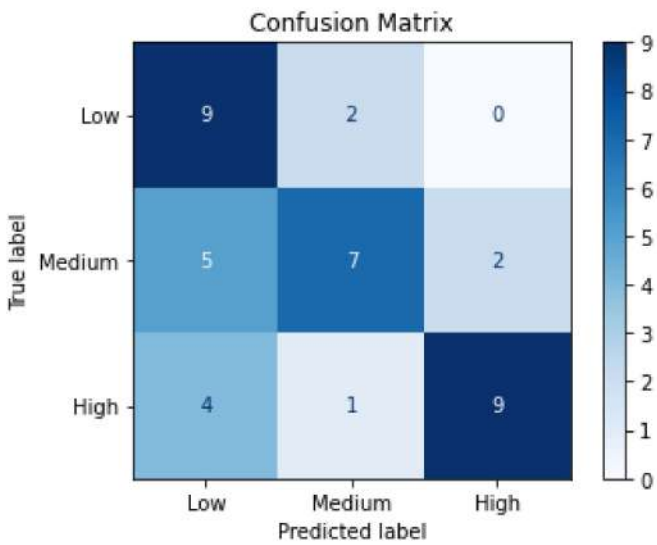
In [14]:
```
joblib.dump(mlp,"model_depression.pkl")
```

Out[14]:
```
['model_depression.pkl']
```

Generation of the confusion matrix of the model

We obtain the statistics of the final model to generate its confusion matrix.

In [15]:
```
## The data for the generation of the confession matrix are defined.
confusion_matrix = ConfusionMatrixDisplay.from_estimator(mlp, X_test, y_test, dis
                                                         cmap = plt.cm.Blues)
confusion_matrix.ax_.set_title("Confusion Matrix")
plt.show()
```



We get the detailed statistics of the final model.

In [16]:
```
## The necessary detailed prediction is generated.
y_pred = mlp.predict(X_test)

## The detailed statistics of the model are printed.
print('Classification accuracy =',accuracy_score(y_test,y_pred) * 100,'%\n')
print(classification_report(y_test,y_pred))
```

```
Classification accuracy = 64.1025641025641 %

              precision    recall  f1-score   support

           0       0.50      0.82      0.62        11
           1       0.70      0.50      0.58        14
           2       0.82      0.64      0.72        14

    accuracy                           0.64        39
   macro avg       0.67      0.65      0.64        39
weighted avg       0.69      0.64      0.64        39
```

Model explainability process

The data used for the test of the neural network model is shown.

In [17]: X_test

Out[17]:

	1. Most of the time I have difficulty concentrating on simple tasks	2. I don't feel like doing my daily duties	3. My friends or family have told me that I look different	4. When I think about the future it is difficult for me to imagine it clearly	5. People around me often ask me how I feel	6. I consider that my life is full of good things	7. My hobbies are still important to me	8. I'm still as punctual as I have always been	9. If I had the chance, I would spend all day in my bed
69	1	0	1	1	0	1	1	0	0
46	1	0	0	0	0	1	1	1	0
58	0	0	0	0	0	1	1	1	0
114	1	0	1	0	0	0	1	0	0
73	0	0	1	0	0	1	1	1	0
98	1	0	1	1	1	0	1	1	0
31	0	0	0	1	1	1	1	0	0
53	1	0	1	1	1	1	1	1	0
65	0	0	0	1	1	1	1	1	1
96	0	0	1	0	0	1	1	1	1
95	0	0	0	0	0	1	0	0	0
97	0	0	0	1	0	1	1	1	0
2	0	0	0	0	1	1	1	0	0
62	0	1	0	1	0	1	1	0	1
110	0	0	0	0	0	0	1	1	0
55	0	1	1	0	0	1	1	1	0
103	1	1	0	0	0	1	1	0	0
100	1	1	0	0	1	1	1	0	0
66	1	1	1	1	0	0	0	0	1
44	0	0	1	0	1	1	1	1	0
77	0	1	1	0	1	1	1	1	1
17	1	0	1	0	0	1	1	1	1
81	1	0	0	0	0	1	1	1	1
74	1	1	0	1	0	0	1	1	1

56	0	0	1	0	0	1	1	1	0
94	1	1	1	0	1	0	1	0	1
35	1	0	1	0	1	1	1	1	0
38	0	1	0	1	1	1	1	1	0
93	0	0	1	0	1	1	1	1	0
48	1	1	0	1	0	1	1	0	0
33	0	1	1	1	0	1	1	0	0
59	0	1	0	0	1	1	1	1	0
92	1	0	0	1	1	0	1	1	1
42	1	0	0	1	1	1	1	1	1
10	0	0	1	1	0	1	1	0	1
85	1	1	0	1	1	1	1	1	1
32	0	1	1	1	0	1	0	0	1
78	1	0	1	0	1	1	1	1	0
19	0	0	1	0	0	0	1	1	0

39 rows × 102 columns

Explanation of the model using ALE

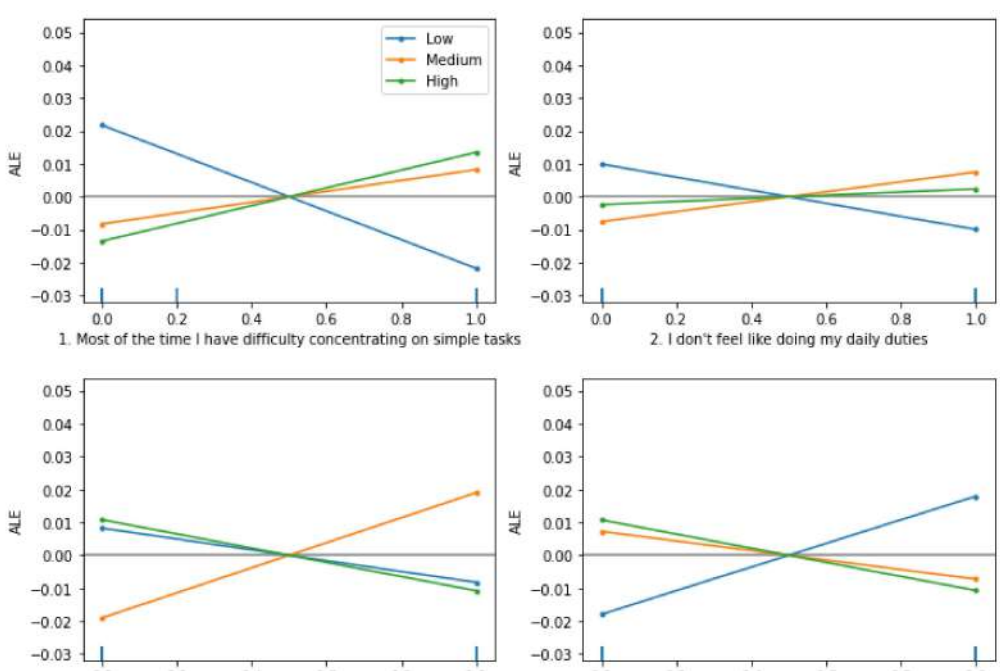
In analysis of the model through the results obtained by ALE, it is possible to identify the behavior of the neural network.

In [18]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
```

The graphs of all the data present in the dataset used are shown.

In [19]:
```
plot_ale(proba_exp_lr, n_cols=2, features=list(range(df.shape[1]-1)),fig_kw={'fig
```



Explanation of the model using LIME

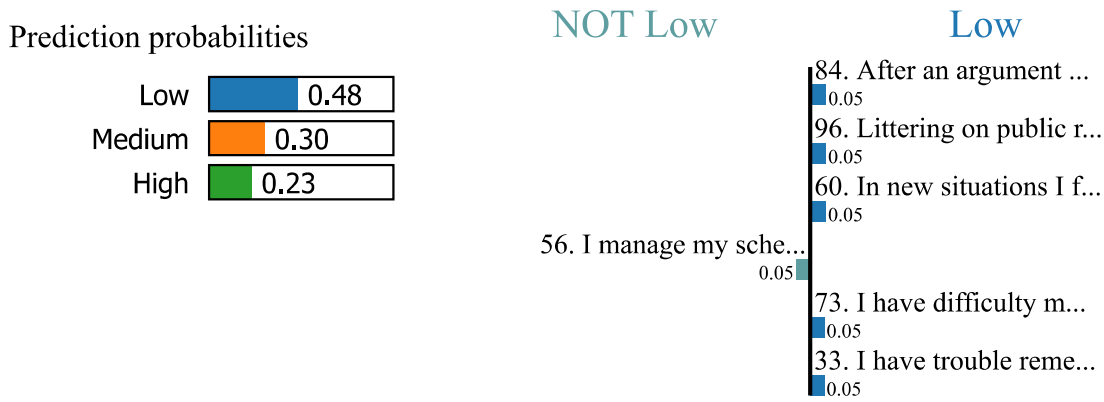
We proceed to explain the improved model using LIME for the Low, Medium and High classes.

In [20]:
```
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorica
                                                   feature_names=train_cols, clas
                                                   discretize_continuous=True)
```

The first individual present in the test data is explained.

In [21]:
```
exp = explainer.explain_instance(X_test.to_numpy()[0], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

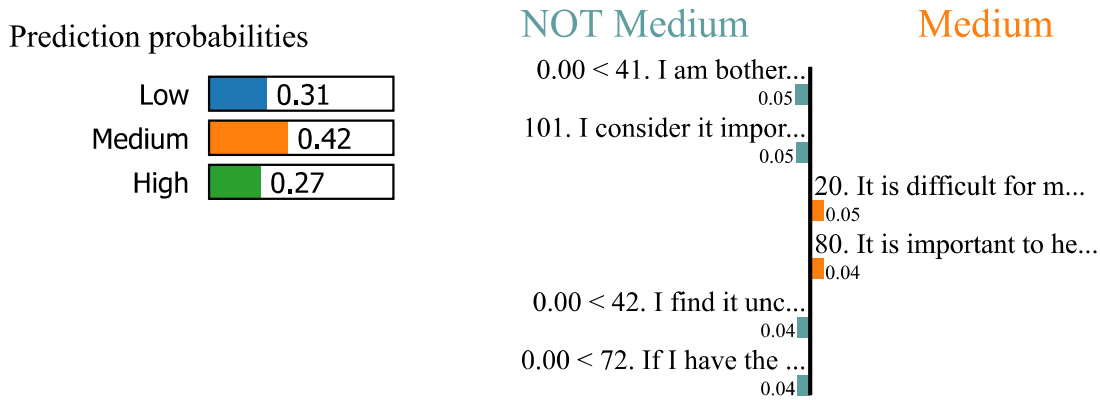


Feature	Value
84. After an argument I usually go over what happened in my head	1.00
96. Littering on public roads is wrong	0.00
60. In new situations I feel calm and encouraged	0.00
56. I manage my schedule as I always have	0.00
73. I have difficulty making elaborate plans	0.00
33. I have trouble remembering things easily	0.00

The second individual present in the test data is explained.

In [22]:
```
exp = explainer.explain_instance(X_test.to_numpy()[1], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

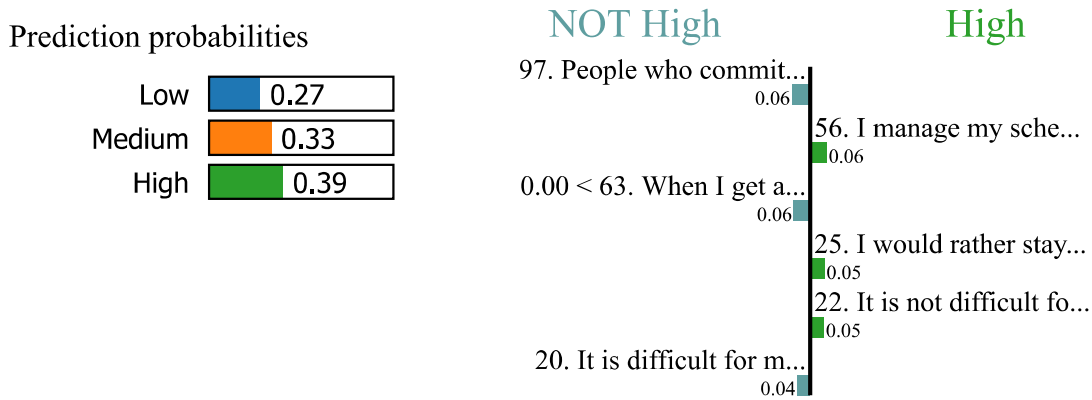


Feature	Value
41. I am bothered by insignificant things that were not important before	1.00
101. I consider it important that all people have the same rights	0.00
20. It is difficult for me to make decisions even if they are simple	0.00
80. It is important to help people when they need it	0.00
42. I find it uncomfortable to be in a crowded place	1.00
72. If I have the opportunity to get in line to avoid wasting time, I do it	1.00

The last individual present in the test data is explained.

In [23]:
```
exp = explainer.explain_instance(X_test.to_numpy()[-1], mlp.predict_proba, num_fe
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```



Feature	Value
97. People who commit crimes have their reasons for doing it	0.00
56. I manage my schedule as I always have	0.00
63. When I get angry I can easily explode	1.00
25. I would rather stay home than go out with my friends	0.00
22. It is not difficult for me to understand something the first time	0.00
20. It is difficult for me to make decisions even if they are simple	0.00

Explanation of the model using SHAP

SHAP explanatory values are displayed according to the class to which the selected individual belongs according to the prediction of the model.

The first individual present in the test data is explained.

In [24]:
```
## Individual Selection.
individual = X_test.iloc[[0]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The second individual present in the data test is explained.

In [25]:
```
## Individual Selection.
individual = X_test.iloc[[1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The last individual present in the data test is explained.

In [26]:
```
## Individual Selection.
individual = X_test.iloc[[-1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

Explanation using heatmaps

The importance of the data present in the neural network is shown from explanation methods.

In [27]:

```
## The question status dataset is loaded.
ds = pd.read_csv(path_dataset_qs)

## Items currently in use are retrieved.
ds_item = ds[ds.Status == "In use"].reset_index(drop=True)

## The data is prepared for the generation of heatmaps.
column_Map = ds_item['Question'].tolist()
content_Map = [False for x in range(1, df.shape[1])]

## The next items to be deleted will be marked.
for index in range(0, len(content_Map)):

    if iteration == 0:
        for delete in list_contr:
            if column_Map[index] == delete:
                content_Map [index] = True
    else:
        for delete in question_anomaly:
            if column_Map[index] == delete:
                content_Map [index] = True

## The dataframe is created with the prepared data.
df_mask = pd.DataFrame([content_Map], columns = column_Map)
df_mask_imp = pd.concat([df_mask, df_mask, df_mask], ignore_index = True, axis =
df_mask_all = pd.concat([df_mask, df_mask, df_mask, df_mask], ignore_index = True

df_mask
```

Out[27]:

	1	2	3	4	5	6	7	8	9	10	...	93	94	95	96	
0	False	False	True	True	True	False	False	False	False	False	...	False	True	True	True	Fa

1 rows × 102 columns

Identification of feature variables

Identification of feature variables from the randomness of the answers as items of the data.

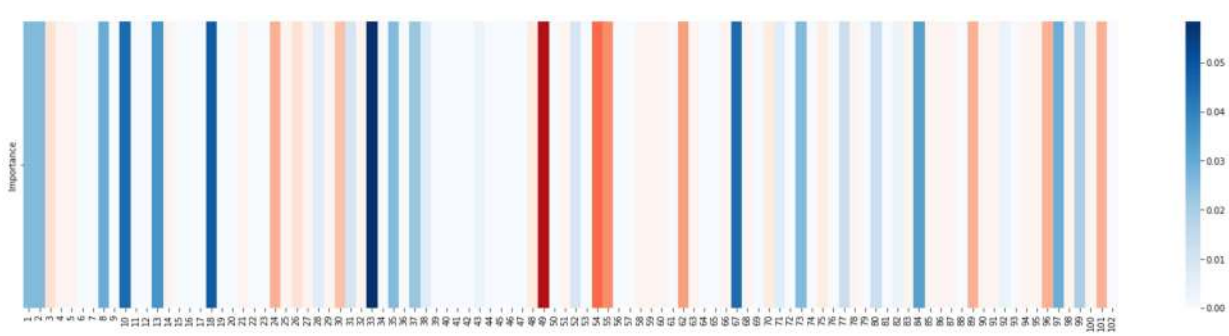
In [28]:
```
## Loading the data for explainability.
explainer = ClassifierExplainer(mlp, X_test, y_test)

## Class index local variable.
iterable_class = -1
map_class = []
```

```
WARNING: Parameter shap='guess', but failed to guess the type of shap explainer
to use for MLPClassifier. Defaulting to the model agnostic shap.KernelExplainer
(shap='kernel'). However this will be slow, so if your model is compatible with
e.g. shap.TreeExplainer or shap.LinearExplainer then pass shap='tree' or shap
='linear'!
WARNING: For shap='kernel', shap interaction values can unfortunately not be ca
lculated!
Note: shap values for shap='kernel' normally get calculated against X_backgroun
d, but paramater X_background=None, so setting X_background=shap.sample(X, 5
0)...
Generating self.shap_explainer = shap.KernelExplainer(model, X, link='identit
y')
```

Explainability of the data belonging to the first class, in this case the Low.

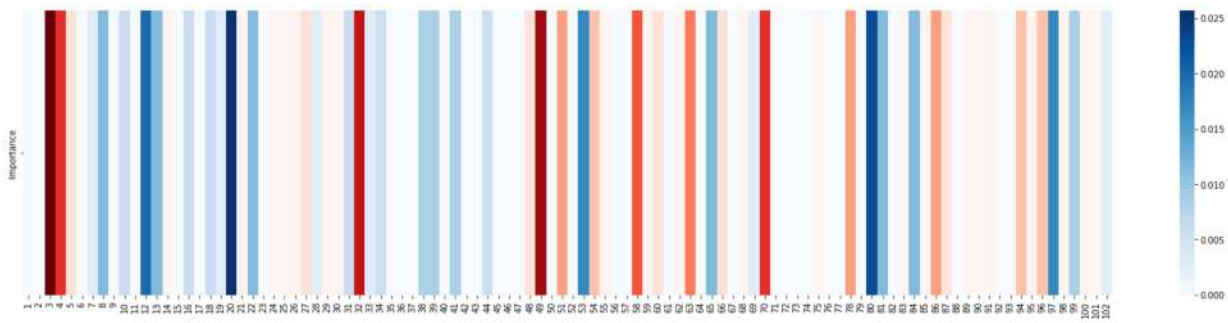
In [29]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

```
Calculating permutation importances (if slow, try setting n_jobs parameter)...
```



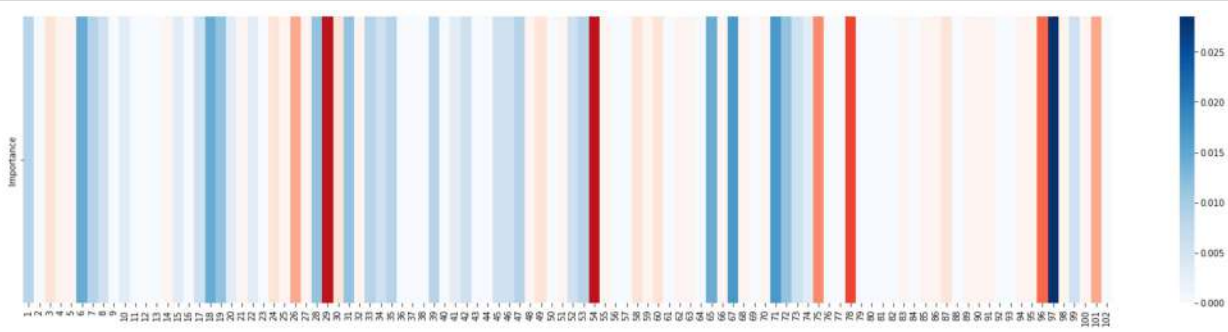
Explainability of the data belonging to the second class, in this case the Medium.

In [30]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



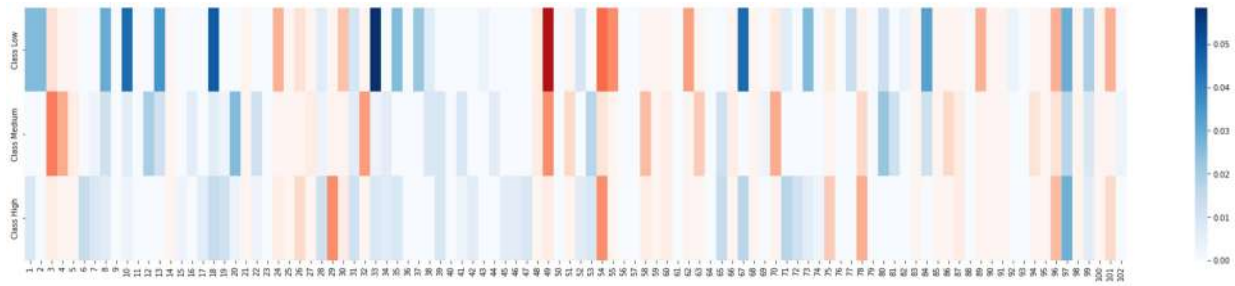
Explainability of the data belonging to the third class, in this case the High.

In [31]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [32]:
```
## Fusion of the previous heatmaps.
df_importance = pd.concat(map_class, axis=1)
df_importance.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_importance.transpose(), cmap = "Reds", cbar = False)
sns.heatmap(df_importance.transpose(), cmap = "Blues",yticklabels = True, xtickla
plt.show()
```



Identification of feature variables using SHAP

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of SHAP.

In [33]:
```
## Class index local variable.
iterable_class = -1
map_class = []
```

Individual explainability of the data belonging to the first class, in this case the Low.

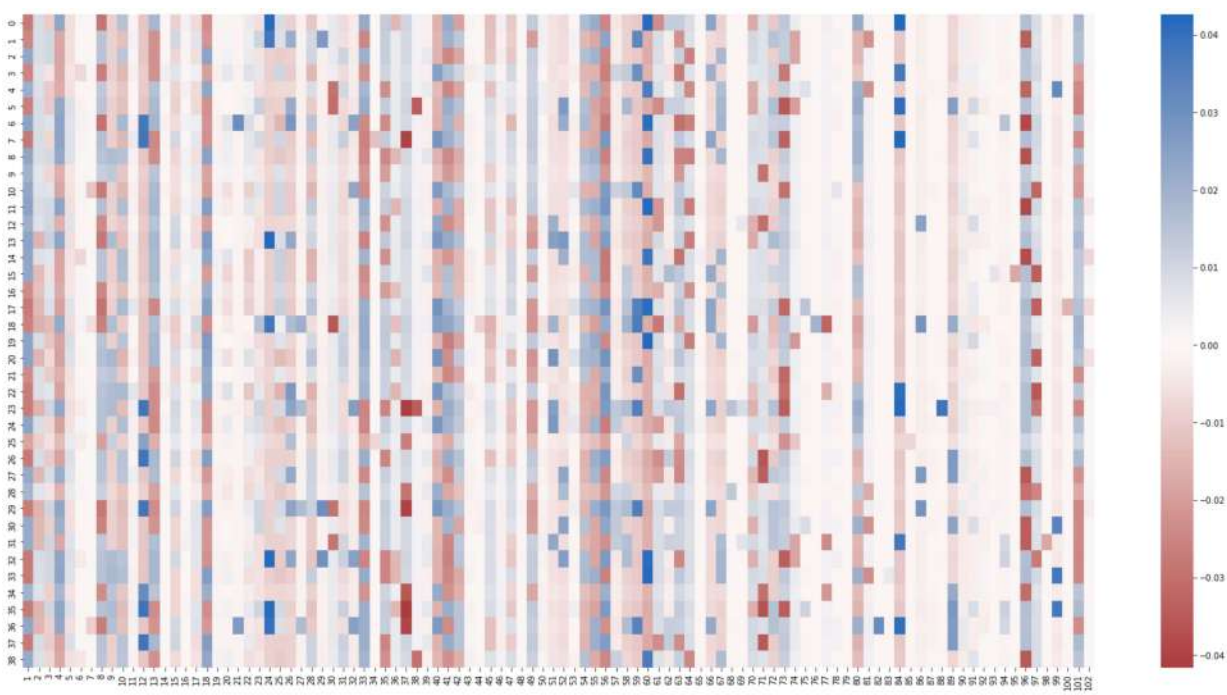
Definition of explainability data.

In [34]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

```
Calculating shap values...

  0%|          | 0/39 [00:00<?, ?it/s]
```

Generation of the heatmap.

In [35]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

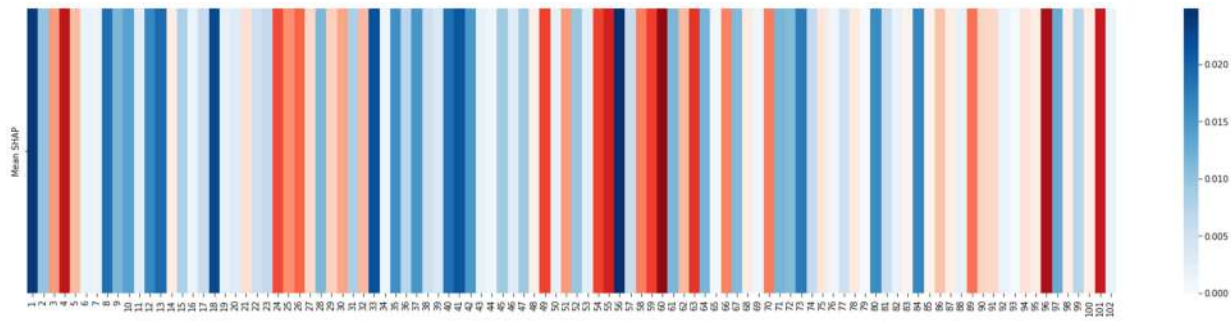


Fusion of the data used.

In [36]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [37]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



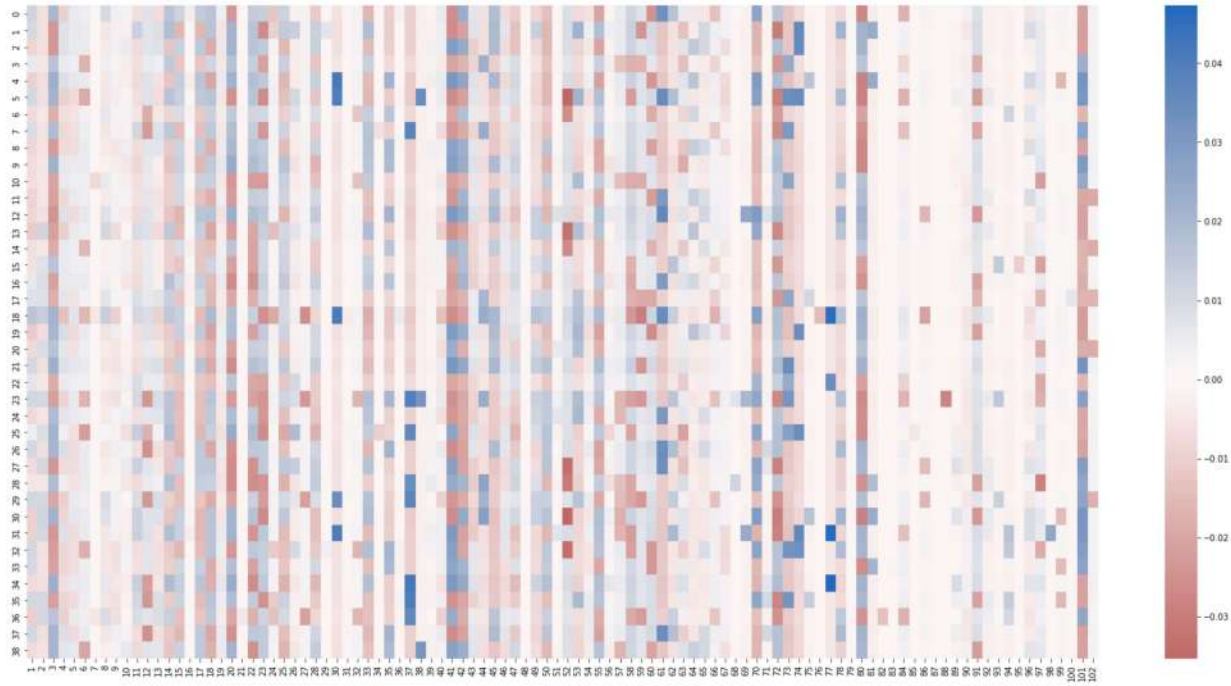
Individual explainability of the data belonging to the second class, in this case the Medium.

Definition of explainability data.

In [38]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [39]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

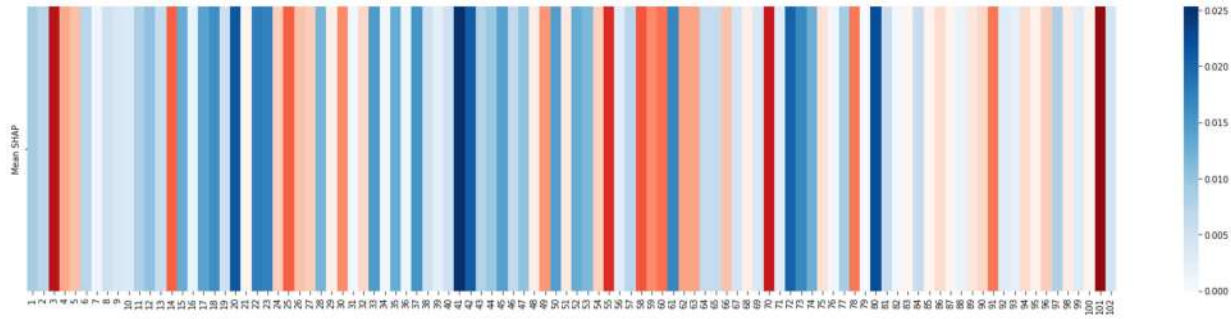


Fusion of the data used.

In [40]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [41]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



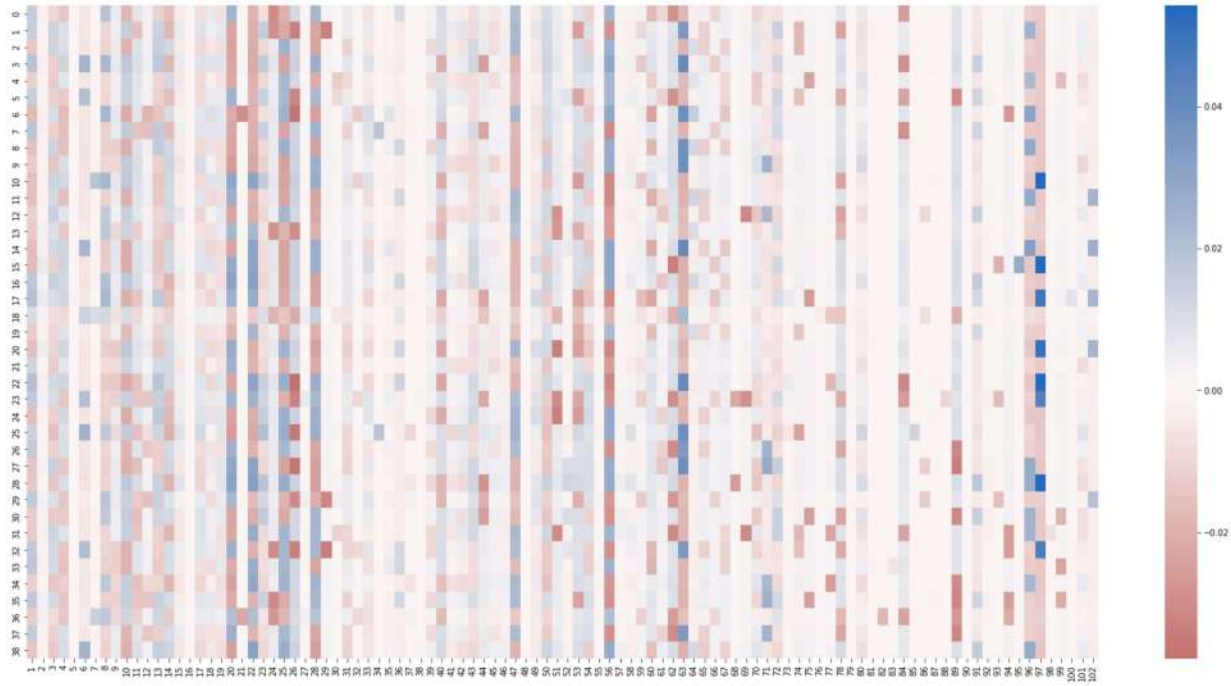
Individual explainability of data belonging to the third class, in this case High.

Definition of explainability data.

In [42]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [43]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

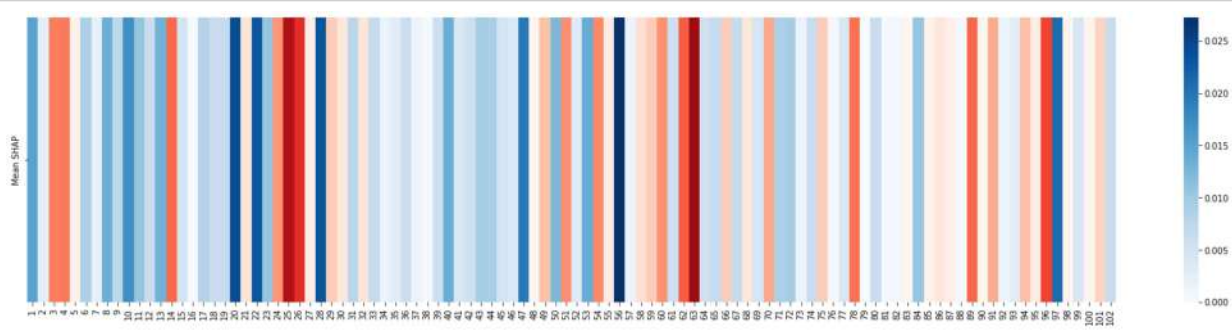


Fusion of the data used.

In [44]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

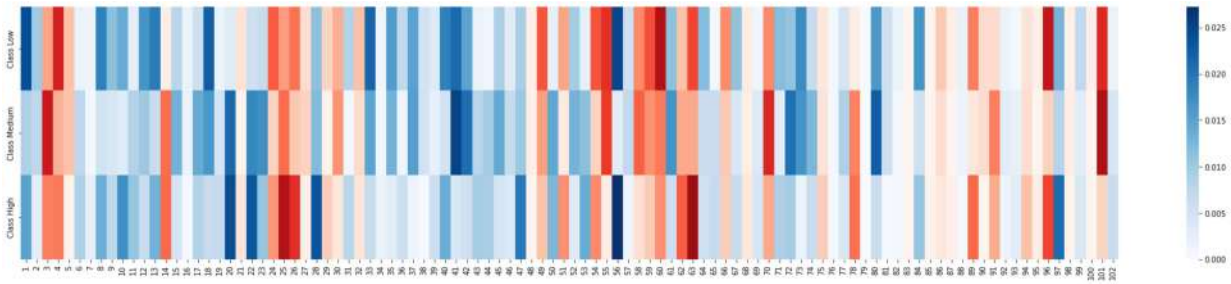
Generation of the heatmap.

In [45]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [46]:
```
## Fusion of the previous heatmaps.
df_shap = pd.concat(map_class, axis=1)
df_shap.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_shap.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_shap.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using LIME

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of LIME.

In [47]:
```
## Data preparation for explainability.
lime_exp = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorical
                                                  feature_names=train_cols, class
                                                  discretize_continuous=True)

## Class index local variable.
iterable_class = -1
map_class = []
```

Obtaining the data for the generation of the explanations by LIME about the Low, Medium and High classes.

In [48]:
```
## Local variables for LIME array data.
exp_matrix = [[] for _ in range(len(name_class))]

## Recovering the data needed for LIME.
for x in X_test.to_numpy():

    ## Auxiliary temporary variables.
    exp_list = [[] for _ in range(len(name_class))]

    ## Loading the data for explainability.
    exp = lime_exp.explain_instance(x, mlp.predict_proba, num_features = df.shape

    ## Recovering the data.
    for elements in range(0, len(name_class)):
        temp = exp.as_map()[elements]
        temp.sort(key=itemgetter(0))

        ## Saving data of the Low, Medium and High classes.
        for tup in temp:
            exp_list[elements].append(tup[1])

        exp_matrix[elements].append(exp_list[elements])
```

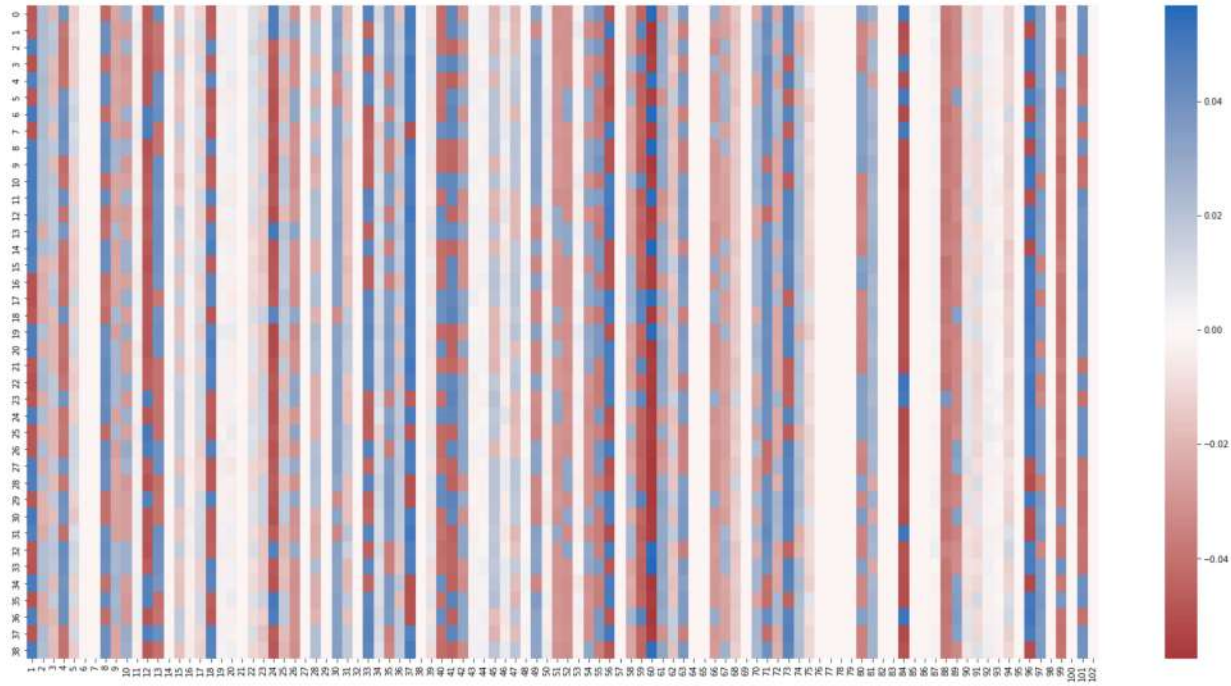
```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

Individual explainability of the data belonging to the first class, in this case the Low.

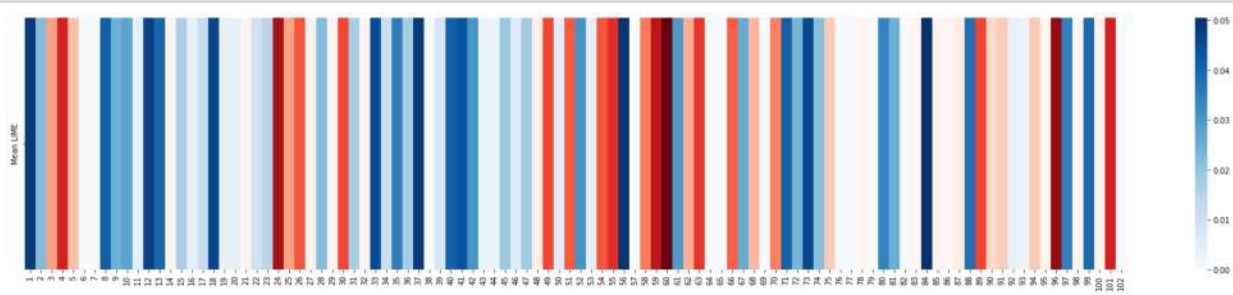
Generation of the heatmap.

In [49]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

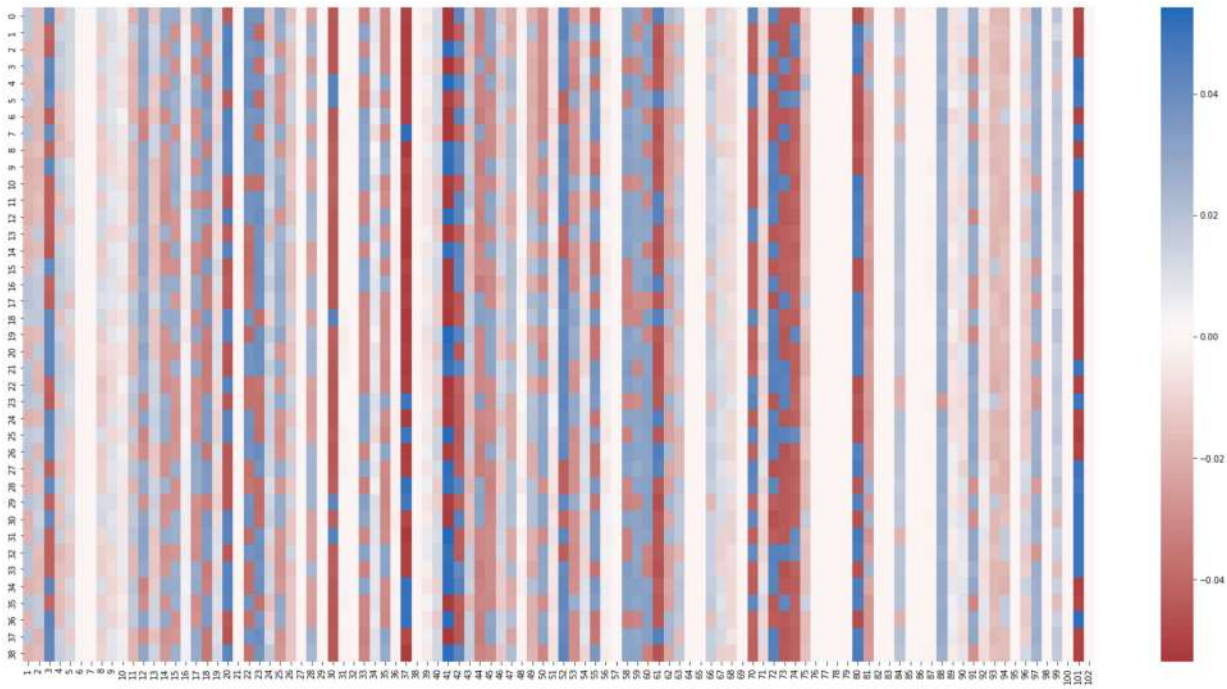
In [50]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Individual explainability of the data belonging to the second class, in this case the Medium.

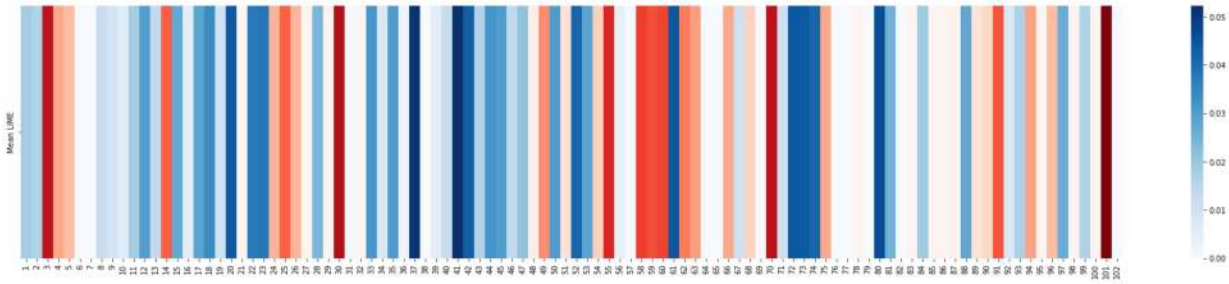
Generation of the heatmap.

In [51]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

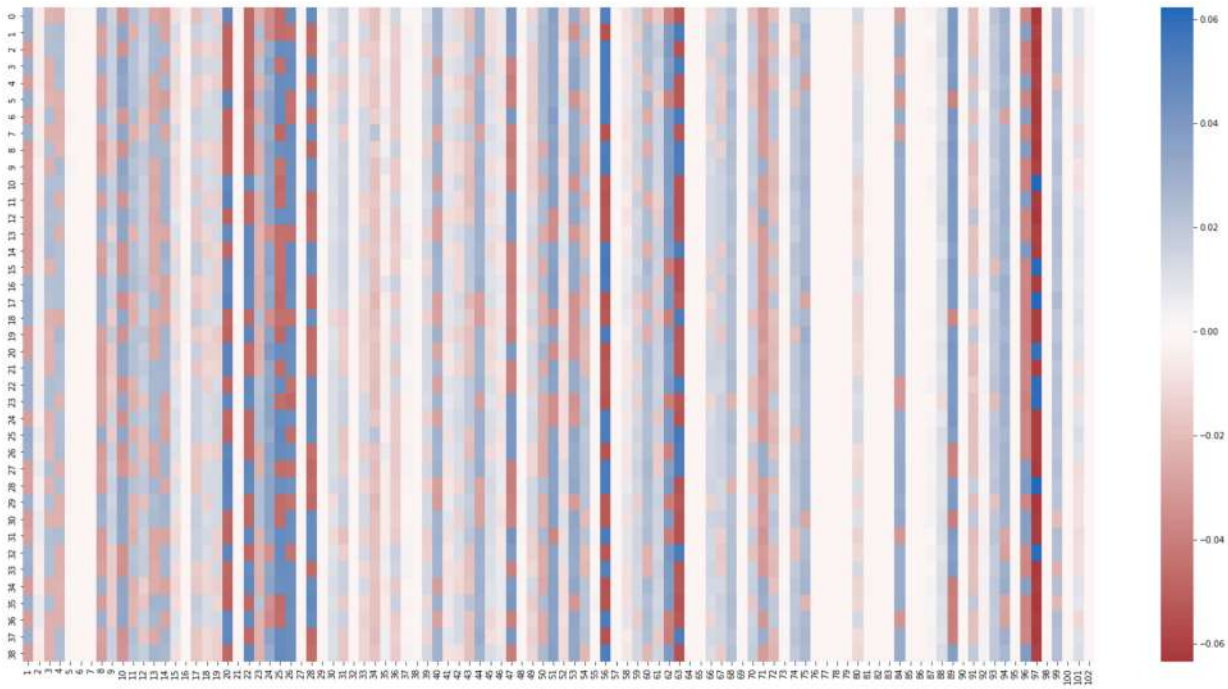
In [52]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Individual explainability of data belonging to the third class, in this case High.

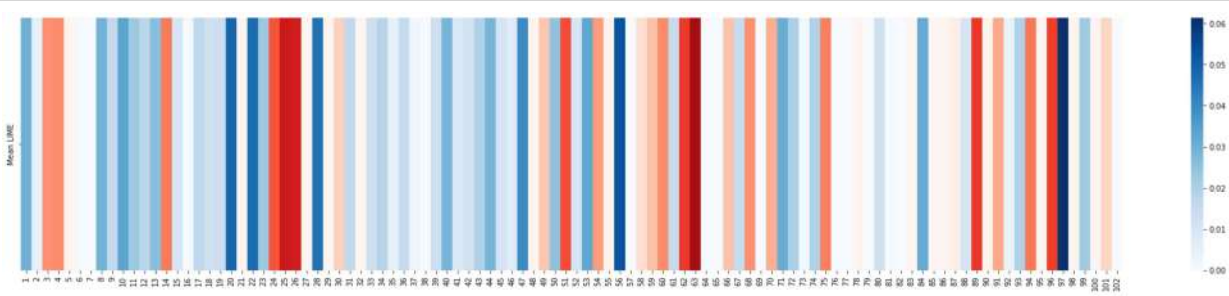
Generation of the heatmap.

In [53]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



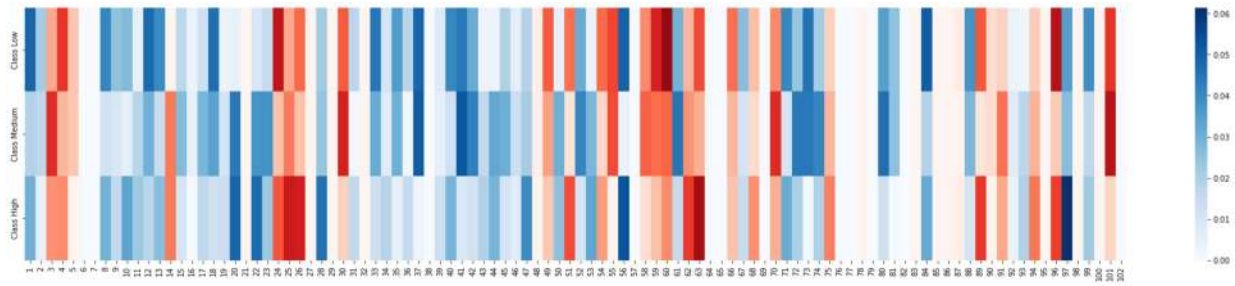
Fusion of the data used.

In [54]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [55]:
```
df_lime = pd.concat(map_class, axis=1)
df_lime.columns = name_class
plt.figure(figsize = (32,6))
sns.heatmap(df_lime.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_lime.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using ALE

It shows the main effect of each question compared to the probability that the model predicts in each class. This effect is shown for both responses. Since there are only 2 answers, the map is the same but with the colors reversed.

In [56]:
```
## Definition of explainability data.
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names=train_cols, target_names= target_r
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())

## Class index local variable.
iterable_class = -1
map_class = []
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
```

Obtaining the data for the generation of the explanations by ALE about all the classes.

In [57]:
```
## Local variables for ALE array data.
ale_list = [[] for _ in range(len(name_class))]

## Recovering the data needed for ALE.
for array in proba_exp_lr.ale_values:

    ## Recovering the data.
    for elements in range(0, len(name_class)):

        ## Saving data of all the classes.
        ale_list[elements].append(array[0][elements])
```

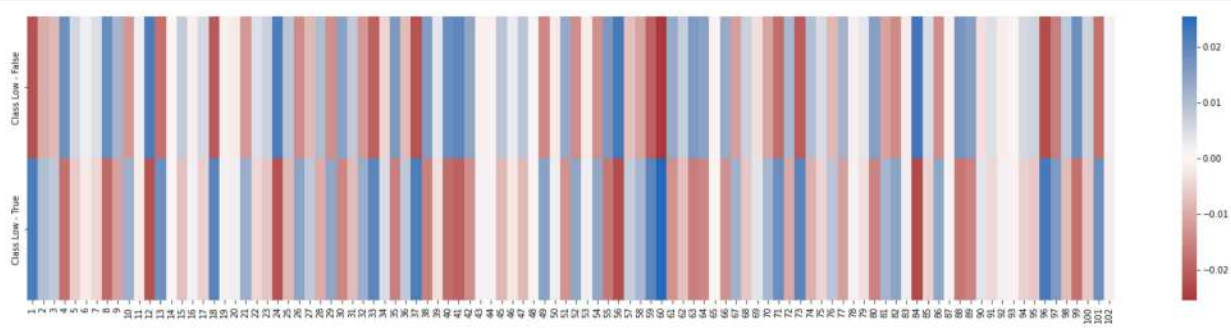
Data processing for the explainability of ALE.

In [58]:
```
## Data processing of the Low, Medium and High classes.
for elements in range(0, len(name_class)):
    ale_df = pd.DataFrame([ale_list [elements]])
    ale_df = pd.concat([ale_df.multiply(-1), ale_df])
    ale_df.index = [name_class [elements] + " - False", name_class [elements] + '
    ale_df.columns = column_Map
    map_class.append(ale_df)
```

Explainability of the data belonging to the first class, in this case the Low.

Generation of the heatmap.

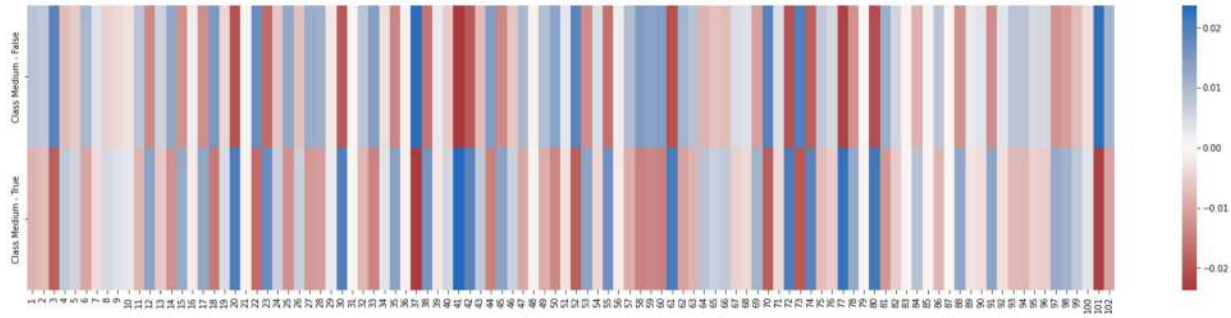
In [59]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the second class, in this case the Medium.

Generation of the heatmap.

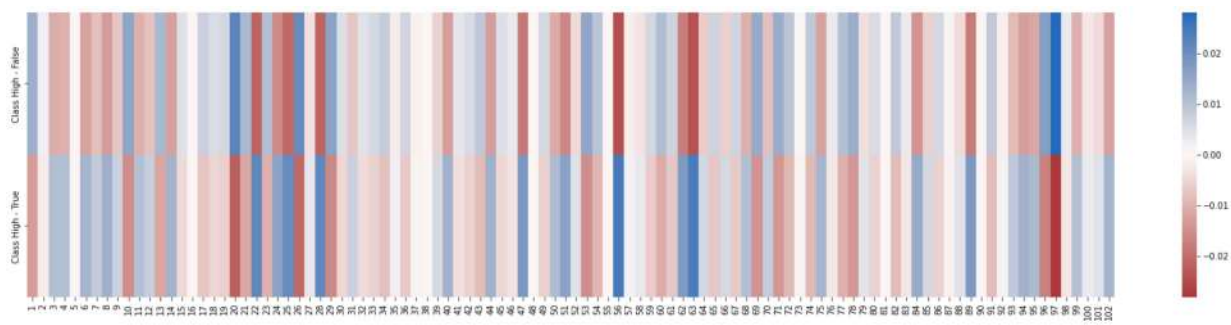
In [60]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the third class, in this case the High.

Generation of the heatmap.

In [61]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```

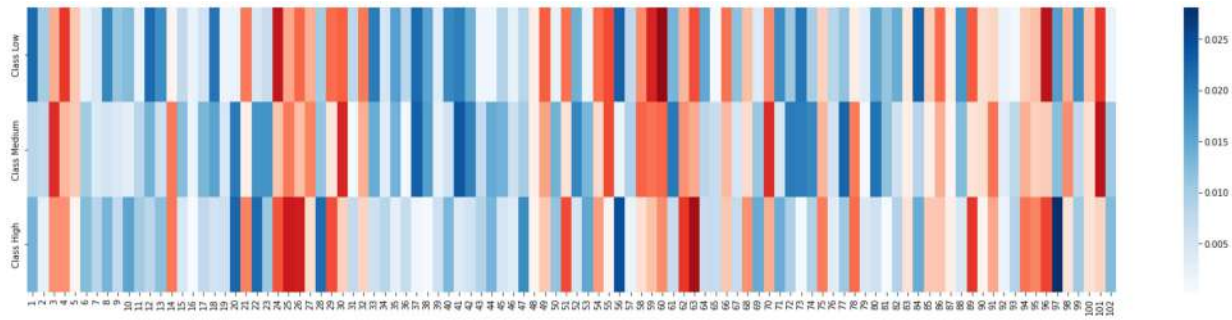


Fusion of the data used in all classes.

In [62]:
```
## Application of absolute value to data.
for index in range(0, len(map_class)):
    map_class [index] = map_class [index][0:1].abs()

## Union of the heatmaps of the Low, Medium and High class.
ale_df = pd.concat(map_class)
ale_df.index = name_class
ale_df = ale_df.transpose()
```

Generation of the heatmap.

In [63]:
```
plt.figure(figsize = (28,6))
sns.heatmap(ale_df.transpose(), cmap="Reds", cbar = False)
sns.heatmap(ale_df.transpose(), cmap="Blues", yticklabels = True, xticklabels = T
plt.show()
```



Comparing of Feature Variables Methods for Explainability

The global values of each of the methods used are taken, their maximums are obtained (the minimums are 0) and a percentage is calculated based on said maximum, which indicates how relevant the contribution is for each question.

In [64]:
```
## Class index local variable.
iterable_class = -1
```

Comparison of the explanation methods of the first class, in this case the Low.

The highest data of each method used is obtained.

In [65]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [66]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

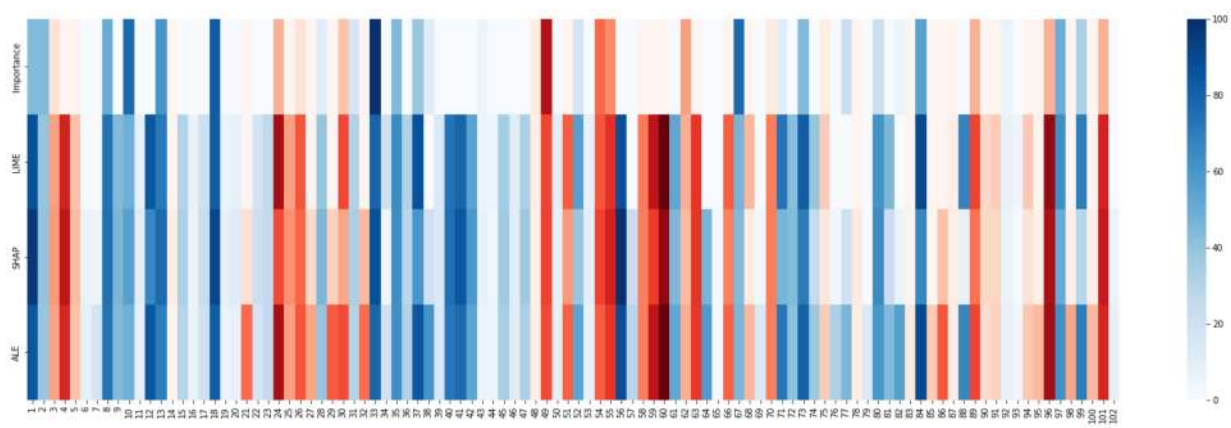
Out[66]:

	1	2	3	4	5	6	7	8
Importance	44.444444	44.444444	11.111111	0.000000	0.000000	0.000000	0.000000	50.000000
LIME	87.185657	39.563534	32.861445	71.291723	22.996035	0.000000	0.000000	74.070139
SHAP	96.193911	40.216977	35.055769	79.056703	24.966668	6.417732	5.384522	75.599726
ALE	85.462751	39.002701	32.360871	70.440617	22.668454	8.642175	17.874580	73.122299

4 rows × 102 columns

Generation of the heatmap.

In [67]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the second class, in this case the Medium.

The highest data of each method used is obtained.

In [68]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [69]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

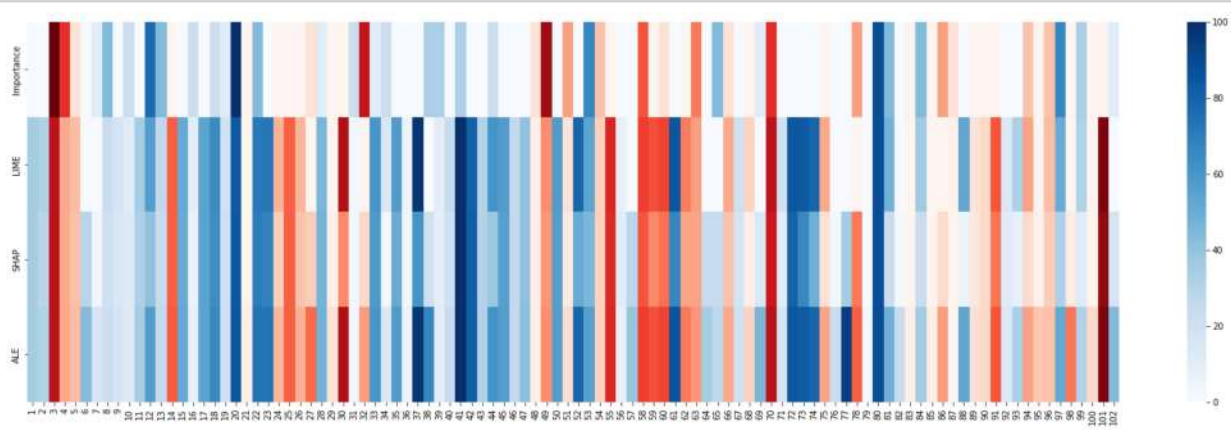
Out[69]:

	1	2	3	4	5	6	7	
Importance	0.000000	4.317534e-13	100.000000	66.666667	11.111111	4.317534e-13	11.111111	4
LIME	35.649119	3.185749e+01	79.732972	30.355488	23.857323	0.000000e+00	0.000000	2
SHAP	34.753117	2.911252e+01	81.017037	29.964468	23.805868	2.905636e+01	3.563667	2
ALE	34.896942	3.189641e+01	80.377192	30.559069	24.028051	4.357354e+01	16.379747	2

4 rows × 102 columns

Generation of the heatmap.

In [70]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the third class, in this case the High.

The highest data of each method used is obtained.

In [71]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [72]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

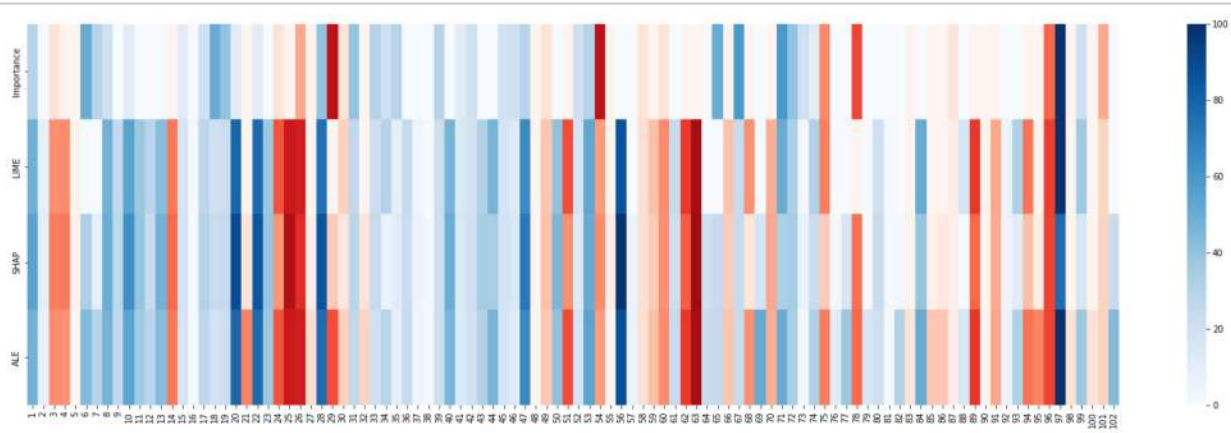
Out[72]:

	1	2	3	4	5	6	7	8
Importance	30.000000	0.000000	10.000000	0.000000	0.000000	50.000000	30.000000	20.000000
LIME	48.604922	8.705045	38.128199	38.718085	1.337985	0.000000	0.000000	48.022348
SHAP	55.417443	9.592068	43.310268	44.252559	1.811028	32.591259	8.343300	49.854939
ALE	47.983663	8.465188	38.323466	38.038633	0.302390	44.471190	29.952344	47.539284

4 rows × 102 columns

Generation of the heatmap.

In [73]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Neural network model improvement iterative process

From this moment, we proceed to improve the performance of the neural network to predict high depression.

Iteration 1

The first iteration of improvement of the model is carried out.

Preparing the data for the neural network

The data is prepared in order to be processed by the neural network.

In [74]:

```
## The iteration number is increased.
iteration += 1

## Loading data from the dataset.
df = pd.read_csv(path_dataset)
ds = pd.read_csv(path_dataset_qs)

## Elimination of unrelated items from the dataset.
if len(unrelated_questions) != 0:
    df = df.drop(df.columns[unrelated_questions[0]:unrelated_questions[1]], axis=

i = 0
if iteration == 1:
    ## Elimination of list of conflicting items declared in global variables.
    for x in list_contr:
        df.drop(df.columns[x - (i + 1)], axis = 1, inplace = True)
        i += 1
        ## Status change of the deleted question in the respective auxiliary data
        ds.loc[x - 1,'Status'] = 'Delete'

else:
    ## Elimination of the list of anomadic items defined in the improvement proce
    ds_delete = ds[ds.Status == "Delete"].reset_index(drop=True)
    for x in ds_delete['Question'].tolist():
        df.drop(df.columns[x - (i + 1)], axis = 1, inplace = True)
        i += 1

## Grouping of the target according to the defined dictionary.
if grouping_name != None and grouping_name != '':
    df[grouping_name] = df[grouping_name].map(class_dic)

## Assignment of local variables according to the data necessary for the neural r
train_cols = df.columns [0:-1]
label = df.columns [-1]
X = df [train_cols]
y = df [label]

## The prepared data of the dataset is printed.
df
```

Out[74]:

	1. Most of the time I have difficulty concentrating on simple tasks	2. I don't feel like doing my daily duties	6. I consider that my life is full of good things	7. My hobbies are still important to me	8. I'm still as punctual as I have always been	9. If I had the chance, I would spend all day in my bed	10. I have found that I can spend a lot of time scrolling the screen \nof my cell phone without searching or stopping at anything in particular	11. When someone asks me something, I have noticed that I take longer than normal to\nrespond	h noti b sha with ca
0	1	1	1	1	1	0	0	1	
1	1	0	1	1	1	0	0	0	
2	0	0	1	1	0	0	1	0	
3	0	0	1	1	1	1	0	0	
4	1	1	1	0	1	1	1	1	
...	...	...	...	...	...	...	...	...	
99	1	1	1	1	1	0	1	1	
100	1	1	1	1	0	0	0	1	
101	0	0	1	1	0	1	1	0	
102	1	0	1	1	1	1	1	1	
103	1	1	1	1	0	0	1	0	

104 rows × 63 columns

Oversample application

An oversample process is applied to level the amount of existing data given by the grouping.

In [75]:
```
## The number of elements present in the grouped target is printed.
y.value_counts()
```

Out[75]:
```
0    39
1    35
2    30
Name: Target, dtype: int64
```

Oversample application.

In [76]:
```
## Application of the oversample to the prepared data.
X, y = oversample.fit_resample(X, y)
```

Printing of the data after the application of the oversample.

In [77]:
```
## The results obtained after applying the oversample are printed.
y.value_counts()
```

Out[77]:
```
0    39
1    39
2    39
Name: Target, dtype: int64
```

Preparation of the parameters of the neural network

We proceed to define the parameters necessary for the training of the neural network.

In [78]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
```

Neural network training

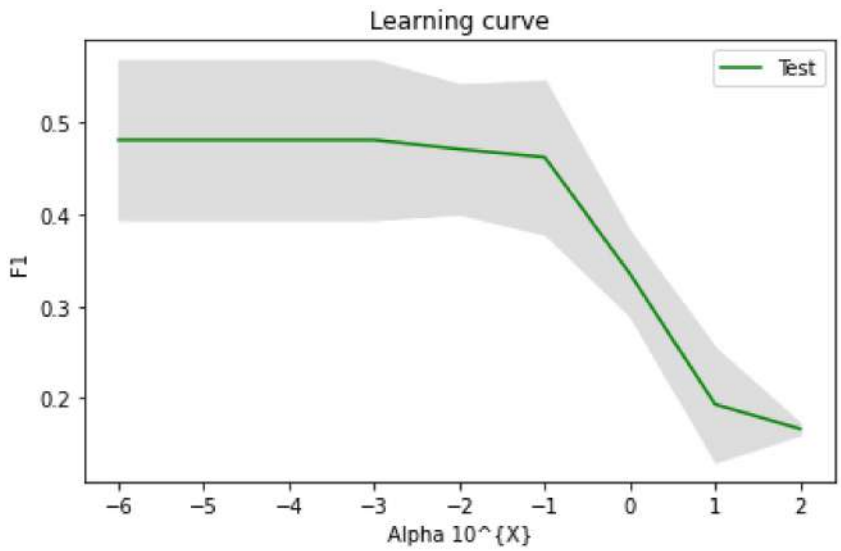
The alpha parameters and the initial learning rate are adjusted to obtain the best performance with the model using cross validation.

In [79]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_normal

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = 0.0001, max_iter =

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[79]:
```
([0.16659498207885304,
  0.19322610764736,
  0.33584807349513235,
  0.4624614446441072,
  0.4712849740558719,
  0.48138800942206517,
  0.48138800942206517,
  0.48138800942206517,
  0.48138800942206517],
 [0.006020651927912844,
  0.06190163786344686,
  0.04574358771581292,
  0.08331624906294144,
  0.07035317262431248,
  0.086614802288128,
  0.086614802288128,
  0.086614802288128,
  0.086614802288128])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [80]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Alpha 10^{X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



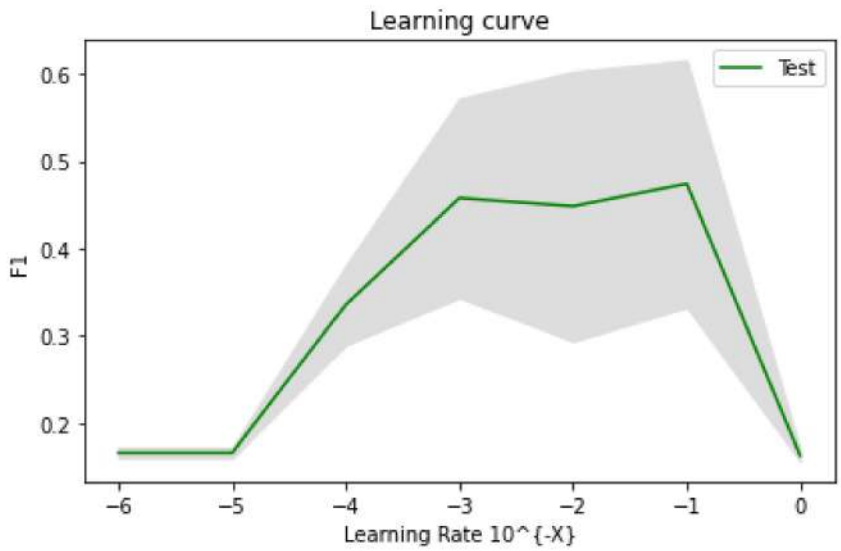
The training is performed again keeping the same parameters except for the alpha value that changes to 1.

In [81]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_alpha

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = regul_param, max_it

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[81]:
```
([0.1632974910394265,
  0.4740513614352624,
  0.4483093256932267,
  0.4577127585177121,
  0.33584807349513235,
  0.16659498207885304,
  0.16659498207885304],
 [0.006619092595549127,
  0.14126534835534385,
  0.1542276089835744,
  0.11392529450516314,
  0.04574358771581292,
  0.006020651927912844,
  0.006020651927912844])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [82]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Learning Rate 10^{-X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



Generation of the neural network model

The parameters of the final model are modified according to the data obtained in the training of the neural network.

In [83]:
```
## The value of the alpha parameter, learning_rate_init and random_state are char
## to the data in the global variables.
mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solver =
                    learning_rate = 'constant', learning_rate_init = lear
                    random_state = random_state_model)

## We print the data of the final generated model.
mlp.fit(X_train, y_train)
```

Out[83]:
```
MLPClassifier
MLPClassifier(activation='logistic', alpha=0.001, hidden_layer_sizes=(10,),
              learning_rate_init=0.1, max_iter=100000, random_state=42)
```

We safeguard the final generated model

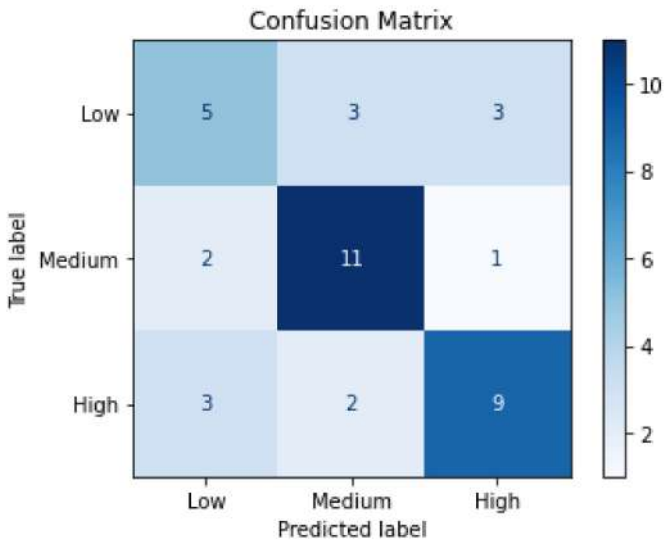
In [84]:
```
joblib.dump(mlp,"model_depression_i" + str(iteration) + ".pkl")
```

Out[84]:
```
['model_depression_i1.pkl']
```

Generation of the confusion matrix of the model

We obtain the statistics of the final model to generate its confusion matrix.

In [85]:
```
## The data for the generation of the confession matrix are defined.
confusion_matrix = ConfusionMatrixDisplay.from_estimator(mlp, X_test, y_test, dis
                                                         cmap = plt.cm.Blues)
confusion_matrix.ax_.set_title("Confusion Matrix")
plt.show()
```



We get the detailed statistics of the final model.

In [86]:
```
## The necessary detailed prediction is generated.
y_pred = mlp.predict(X_test)

## The detailed statistics of the model are printed.
print('Classification accuracy =',accuracy_score(y_test,y_pred) * 100,'%\n')
print(classification_report(y_test,y_pred))
```

```
Classification accuracy = 64.1025641025641 %

              precision    recall  f1-score   support

           0       0.50      0.45      0.48        11
           1       0.69      0.79      0.73        14
           2       0.69      0.64      0.67        14

    accuracy                           0.64        39
   macro avg       0.63      0.63      0.63        39
weighted avg       0.64      0.64      0.64        39
```

The data obtained is saved for later graphing.

In [87]:
```
accuracy = accuracy_score(y_test,y_pred) * 100
clf_rep = precision_recall_fscore_support(y_test,y_pred)

## The number of items used in the iteration is retrieved.
ds_delete = ds[ds.Status == "Delete"]
ds_delete.reset_index(inplace=True, drop=True)

## All the metrics of the confusion matrix are obtained.
metrics = [",".join(map(str, ds_delete['Question'].tolist())),
           accuracy]

for index in range(0, len(name_class)):
    metrics.append(clf_rep[0][index])
    metrics.append(clf_rep[1][index])
    metrics.append(clf_rep[2][index])
    metrics.append(clf_rep[3][index])

## The names of the columns of the dataset are defined.
columns = ['Question', 'Acurracy global']

for index in range(0, len(name_class)):
    columns.append('Precision_' + str(index))
    columns.append('Recall_' + str(index))
    columns.append('F1_score_' + str(index))
    columns.append('Support_' + str(index))

## A new row of the dataset is generated with all the data.
data_metrics = pd.DataFrame([metrics], columns = columns)

## The data is saved to the dataset.
data_metrics.to_csv(path_dataset_qd, mode = 'a', header = False, index = False)

## The data added to the dataset is printed.
data_metrics
```

Out[87]:

	Question	Acurracy global	Precision_0	Recall_0	F1_score_0	Supp
0	3,4,5,14,21,24,25,26,27,29,30,32,48,49,51,54,5...	64.102564	0.5	0.454545	0.47619	

Improvement process of the neural network model.

The application of the explanation method by ALE allows to determine the behavior of the neural network and thereby improve it by discarding non-significant data for it.

Slope Analysis of ALE Plots

In [88]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
```

We proceed to obtain the graphs to obtain the slopes of the selected classes.

In [89]:
```
## The question metrics dataset is loaded.
dm = pd.DataFrame()

## The number of items used in the iteration is retrieved.
ds_item = ds[ds.Status == "In use"]
ds_item.reset_index(inplace=True, drop=True)

## We get the slopes from the ALE data.
for i in range(df.shape[1]-1):
    structure_data = {'Question': ds_item.loc[i, "Question"]}

    for index in selection_class:
        slope = proba_exp_lr.ale_values [i][1][index] - proba_exp_lr.ale_values [
        structure_data['Slope ' + str(target_names[index])] = slope
        structure_data['Threshold ' + str(target_names[index])] = 'NA'
        structure_data['Anomaly ' + str(target_names[index])] = 'NA'

    dm = dm.append(structure_data, ignore_index=True)

## The current rating data dataset is printed.
dm
```

```
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
```

```
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
```

```
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
```

Out[89]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	1	-0.188073	NA	NA	0.180665	NA	NA
1	2	-0.017648	NA	NA	0.009065	NA	NA
2	6	-0.001497	NA	NA	-0.011459	NA	NA
3	7	-0.007423	NA	NA	0.002598	NA	NA
4	8	0.023104	NA	NA	-0.026986	NA	NA
...	...	...	...	...	...	...	...
57	92	0.006494	NA	NA	-0.007989	NA	NA
58	93	-0.008606	NA	NA	0.011228	NA	NA
59	97	-0.247825	NA	NA	0.294782	NA	NA
60	99	0.196822	NA	NA	-0.146336	NA	NA
61	102	0.020829	NA	NA	-0.022950	NA	NA

62 rows × 7 columns

Slope analysis and threshold application

The slope obtained is analyzed and those that are above or below the positive and negative threshold are selected.

In [90]:
```
for i in range(df.shape[1]-1):
    for index in selection_class:

        ## The slopes are selected according to the values of the thresholds of t
        if dm.loc[i, "Slope " + str(target_names[index])] >= positive_threshold:
            dm.loc[i, "Threshold " + str(target_names[index])] = 1
        elif dm.loc[i, "Slope " + str(target_names[index])] <= negative_threshold
            dm.loc[i, "Threshold " + str(target_names[index])] = 0

## The current rating data dataset is printed.
dm
```

Out[90]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	1	-0.188073	0	NA	0.180665	1	NA
1	2	-0.017648	0	NA	0.009065	NA	NA
2	6	-0.001497	NA	NA	-0.011459	0	NA
3	7	-0.007423	NA	NA	0.002598	NA	NA
4	8	0.023104	1	NA	-0.026986	0	NA
...	...	...	...	...	...	...	...
57	92	0.006494	NA	NA	-0.007989	NA	NA
58	93	-0.008606	NA	NA	0.011228	1	NA
59	97	-0.247825	0	NA	0.294782	1	NA
60	99	0.196822	1	NA	-0.146336	0	NA
61	102	0.020829	1	NA	-0.022950	0	NA

62 rows × 7 columns

Analysis and determination of anomalous items

The anomalous items present in the model are determined based on their slope and expected response from the expert.

In [91]:
```
## The dataset of expected responses is loaded.
da = pd.read_csv(path_dataset_qa)

## Delete list of items deleted in this iteration.
i = 0
for x in ds_delete['Question']:
    da.drop(da.index[x - (i + 1)], axis = 0, inplace = True)
    i += 1

## Indexes are restored for later use.
da.reset_index(inplace=True, drop=True)

## The dataset of expected responses is printed.
da
```

Out[91]:

	Number	Question	RE
0	1	Most of the time I have difficulty concentrati...	1
1	2	I don't feel like doing my daily duties	1
2	6	I consider that my life is full of good things	0
3	7	My hobbies are still important to me	0
4	8	I'm still as punctual as I have always been	0
...	...	...	...
57	92	It does not seem serious to me to have some debts	1
58	93	People steal because they have needs	1
59	97	People who commit crimes have their reasons fo...	1
60	99	It is important to respect turns	0
61	102	I have a hard time taking "no" for an answer	1

62 rows × 3 columns

We proceed to identify the anomadic items present in the model present in this iteration.

In [92]:
```
for i in range(df.shape[1]-1):

    ## Local variable to indicate the index of the selected classes.
    i_index = 0

    for index in selection_class:

        ## Determine if it fails based on the expected response parameters of the
        if dm.loc[i, "Threshold "  + str(target_names[index])] == 1 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 1 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 0 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 0 and da.loc[i,
            dm.loc[i, "Anomaly "  + str(target_names[index])] = expected_response

        ## Increase of the index of the selected classes.
        i_index += 1

## The final grade data dataset is printed.
dm
```

Out[92]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	1	-0.188073	0	0	0.180665	1	0
1	2	-0.017648	0	0	0.009065	NA	NA
2	6	-0.001497	NA	NA	-0.011459	0	0
3	7	-0.007423	NA	NA	0.002598	NA	NA
4	8	0.023104	1	0	-0.026986	0	0
...	...	...	...	...	...	...	...
57	92	0.006494	NA	NA	-0.007989	NA	NA
58	93	-0.008606	NA	NA	0.011228	1	0
59	97	-0.247825	0	0	0.294782	1	0
60	99	0.196822	1	0	-0.146336	0	0
61	102	0.020829	1	1	-0.022950	0	1

62 rows × 7 columns

Removing identified anomadic items from the model

We proceed to eliminate the identified anomadic items so as not to use them in the next iteration.

In [93]:
```
## Local variable for the elimination of the anomaly items.
instructions = ''
init = False
question_anomaly = []

if selection_mode == 1:

    ## Items that fail in the defined class are retrieved.
    dm_delete = dm[dm['Anomaly ' + str(target_names[select_class_mode])] == 1]
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 2:

    ## Items that fail in classes defined with the or condition type are retrieve
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 1"
            init = True
        else:
            instructions += " or `Anomaly " + str(target_names[index]) + "` == 1'

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 3:

    ## Items that fail in classes defined with the and condition type are retriev
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 1"
            init = True
        else:
            instructions += " and `Anomaly " + str(target_names[index]) + "` == 1

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 4:

    ## Out-of-threshold items are retrieved in classes defined with the conditior
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 'NA'"
            init = True
        else:
            instructions += " and `Anomaly " + str(target_names[index]) + "` == '

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

## The selected blank items are printed according to the selection_mode.
question_anomaly
```

Out[93]: [9, 11, 12, 15, 23, 38, 44, 45, 46, 72, 102]

Model explainability process

The data used for the test of the neural network model is shown.

In [94]: X_test

Out[94]:

	1. Most of the time I have difficulty concentrating on simple tasks	2. I don't feel like doing my daily duties	6. I consider that my life is full of good things	7. My hobbies are still important to me	8. I'm still as punctual as I have always been	9. If I had the chance, I would spend all day in my bed	10. I have found that I can spend a lot of time scrolling the screen \nof my cell phone without searching or stopping at anything in particular	11. When someone asks me something, I have noticed that I take longer than normal to\nrespond	h noti b sha with ca
69	1	0	1	1	0	0	0	0	
46	1	0	1	1	1	0	1	1	
58	0	0	1	1	1	0	0	0	
114	1	0	1	1	1	0	1	1	
73	0	0	1	1	1	0	1	0	
98	1	0	0	1	1	0	1	0	
31	0	0	1	1	0	0	0	0	
53	1	0	1	1	1	0	1	1	
65	0	0	1	1	1	1	0	0	
96	0	0	1	1	1	1	1	0	
95	0	0	1	0	0	0	1	0	
97	0	0	1	1	1	0	0	0	
2	0	0	1	1	0	0	1	0	
62	0	1	1	1	0	1	1	1	
110	0	0	0	1	1	0	0	0	
55	0	1	1	1	1	0	0	0	
103	1	1	1	1	0	0	1	0	
100	1	1	1	1	0	0	0	1	
66	1	1	0	0	0	1	1	1	
44	0	0	1	1	1	0	0	0	
77	0	1	1	1	1	1	1	0	
17	1	0	1	1	1	1	1	0	
81	1	0	1	1	1	1	0	1	
74	1	1	0	1	1	1	1	1	

56	0	0	1	1	1	0	0	0
94	1	1	0	1	0	1	1	1
35	1	0	1	1	1	0	0	0
38	0	1	1	1	1	0	0	1
93	0	0	1	1	1	0	1	0
48	1	1	1	1	0	0	1	1
33	0	1	1	1	0	0	1	1
59	0	1	1	1	1	0	1	1
92	1	0	0	1	1	1	0	0
42	1	0	1	1	1	1	0	0
10	0	0	1	1	0	1	1	1
85	1	1	1	1	1	1	1	1
32	0	1	1	0	0	1	1	1
78	1	0	1	1	1	0	0	0
19	0	0	0	1	1	0	0	0

39 rows × 62 columns

Explanation of the model using ALE

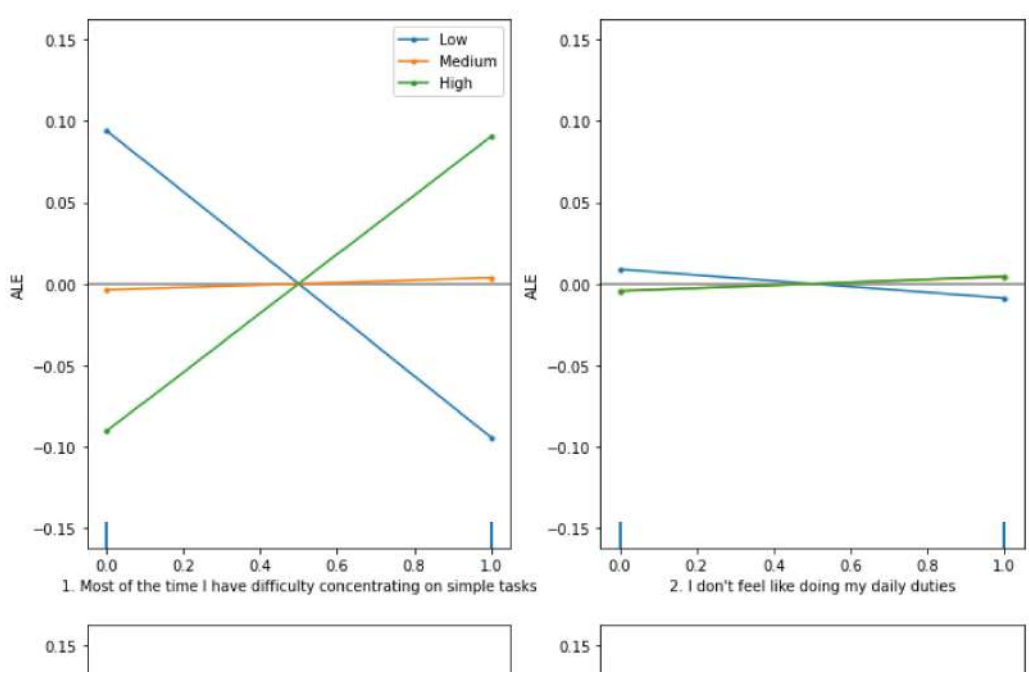
In analysis of the model through the results obtained by ALE, it is possible to identify the behavior of the neural network.

In [95]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
```

The graphs of all the data present in the dataset used are shown.

In [96]:
```
plot_ale(proba_exp_lr, n_cols=2, features=list(range(df.shape[1]-1)),fig_kw={'fig
```



Explanation of the model using LIME

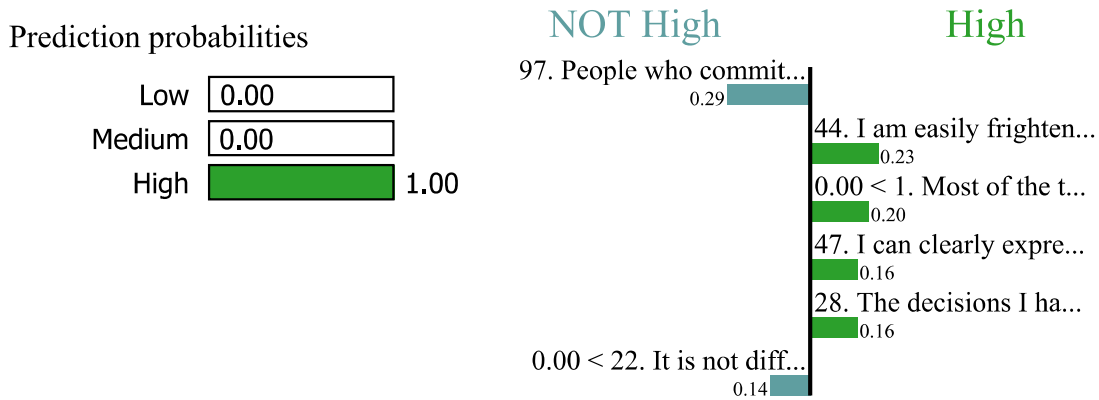
We proceed to explain the improved model using LIME for the Low, Medium and High classes.

In [97]:
```
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorica
                                                   feature_names=train_cols, clas
                                                   discretize_continuous=True)
```

The first individual present in the test data is explained.

In [98]:
```
exp = explainer.explain_instance(X_test.to_numpy()[0], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

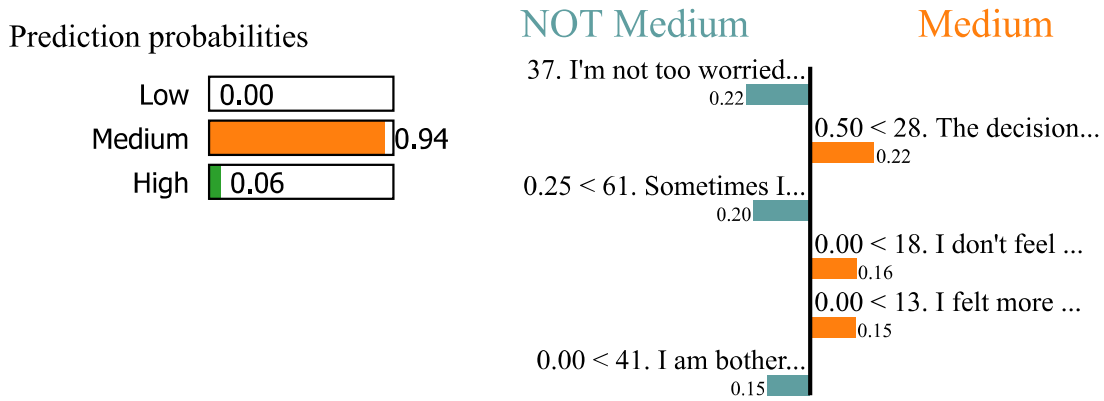


Feature	Value
97. People who commit crimes have their reasons for doing it	0.00
44. I am easily frightened by unexpected noises	0.00
1. Most of the time I have difficulty concentrating on simple tasks	1.00
47. I can clearly express to others how I feel	0.00
28. The decisions I have made so far have been the right ones	0.00
22. It is not difficult for me to understand something the first time	1.00

The second individual present in the test data is explained.

In [99]:
```
exp = explainer.explain_instance(X_test.to_numpy()[1], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

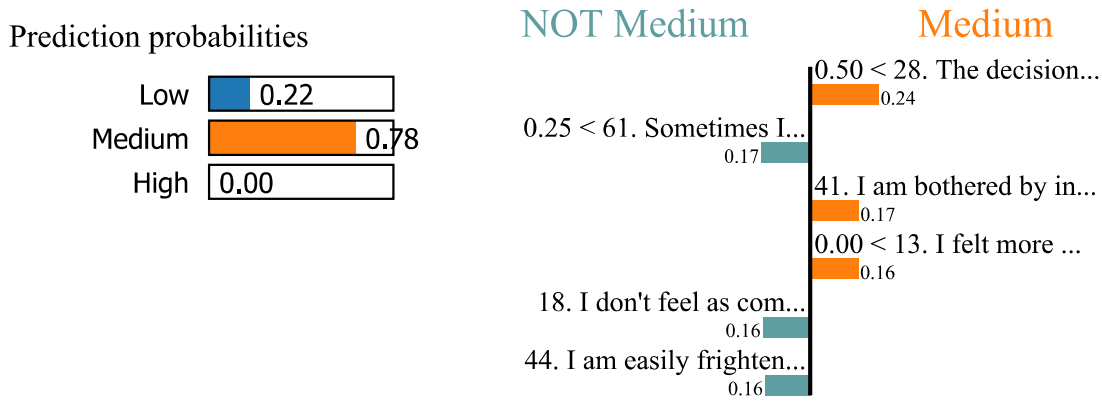


Feature	Value
37. I'm not too worried about what might happen in a few weeks	0.00
28. The decisions I have made so far have been the right ones	1.00
61. Sometimes I forget what I wanted to say because I have several thoughts at the same time	1.00
18. I don't feel as comfortable with my body as I did before	1.00
13. I felt more encouraged to do my daily activities before	1.00
41. I am bothered by insignificant things that were not important before	1.00

The last individual present in the test data is explained.

In [100]:
```
exp = explainer.explain_instance(X_test.to_numpy()[-1], mlp.predict_proba, num_fe
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```



Feature	Value
28. The decisions I have made so far have been the right ones	1.00
61. Sometimes I forget what I wanted to say because I have several thoughts at the same time	1.00
41. I am bothered by insignificant things that were not important before	0.00
13. I felt more encouraged to do my daily activities before	1.00
18. I don't feel as comfortable with my body as I did before	0.00
44. I am easily frightened by unexpected noises	0.00

Explanation of the model using SHAP

SHAP explanatory values are displayed according to the class to which the selected individual belongs according to the prediction of the model.

The first individual present in the test data is explained.

In [101]:
```
## Individual Selection.
individual = X_test.iloc[[0]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The second individual present in the data test is explained.

In [102]:
```
## Individual Selection.
individual = X_test.iloc[[1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The last individual present in the data test is explained.

In [103]:
```
## Individual Selection.
individual = X_test.iloc[[-1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

Explanation using heatmaps

The importance of the data present in the neural network is shown from explanation methods.

In [104]:

```
## Items currently in use are retrieved.
ds_item = ds[ds.Status == "In use"].reset_index(drop=True)

## The data is prepared for the generation of heatmaps.
column_Map = ds_item['Question'].tolist()
content_Map = [False for x in range(1, df.shape[1])]

## The next items to be deleted will be marked.
for index in range(0, len(content_Map)):

    if iteration == 0:
        for delete in list_contr:
            if column_Map[index] == delete:
                content_Map [index] = True
    else:
        for delete in question_anomaly:
            if column_Map[index] == delete:
                content_Map [index] = True

## The dataframe is created with the prepared data.
df_mask = pd.DataFrame([content_Map], columns = column_Map)
df_mask_imp = pd.concat([df_mask, df_mask, df_mask], ignore_index = True, axis =
df_mask_all = pd.concat([df_mask, df_mask, df_mask, df_mask], ignore_index = True

df_mask
```

Out[104]:

	1	2	6	7	8	9	10	11	12	13	...	80	81	82	84
0	False	False	False	False	False	True	False	True	True	False	...	False	False	False	False

1 rows × 62 columns

Identification of feature variables

Identification of feature variables from the randomness of the answers as items of the data.

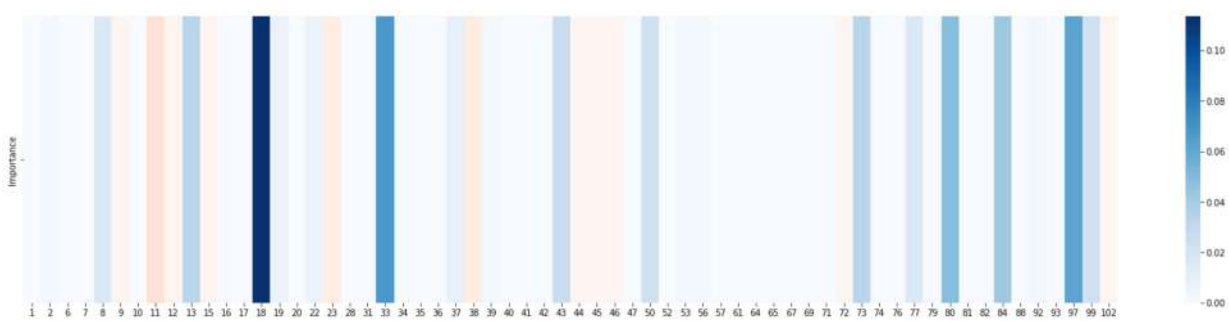
In [105]:
```
## Loading the data for explainability.
explainer = ClassifierExplainer(mlp, X_test, y_test)

## Class index local variable.
iterable_class = -1
map_class = []
```

```
WARNING: Parameter shap='guess', but failed to guess the type of shap explainer
to use for MLPClassifier. Defaulting to the model agnostic shap.KernelExplainer
(shap='kernel'). However this will be slow, so if your model is compatible with
e.g. shap.TreeExplainer or shap.LinearExplainer then pass shap='tree' or shap
='linear'!
WARNING: For shap='kernel', shap interaction values can unfortunately not be ca
lculated!
Note: shap values for shap='kernel' normally get calculated against X_backgroun
d, but paramater X_background=None, so setting X_background=shap.sample(X, 5
0)...
Generating self.shap_explainer = shap.KernelExplainer(model, X, link='identit
y')
```

Explainability of the data belonging to the first class, in this case the Low.

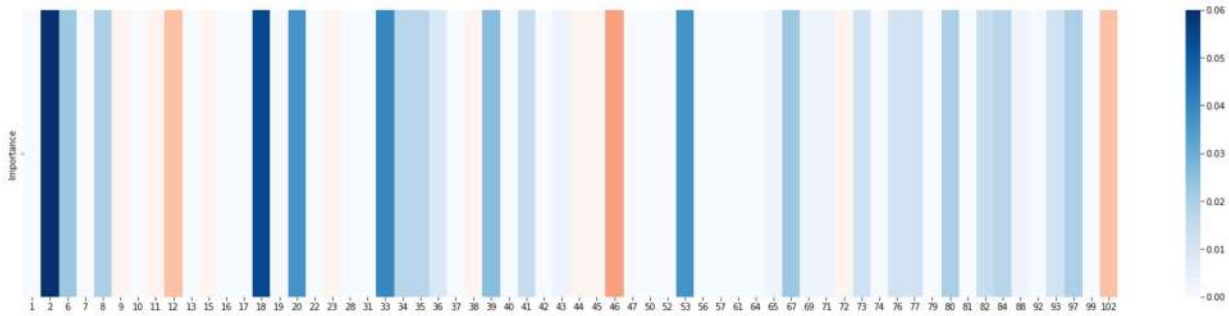
In [106]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

```
Calculating permutation importances (if slow, try setting n_jobs parameter)...
```



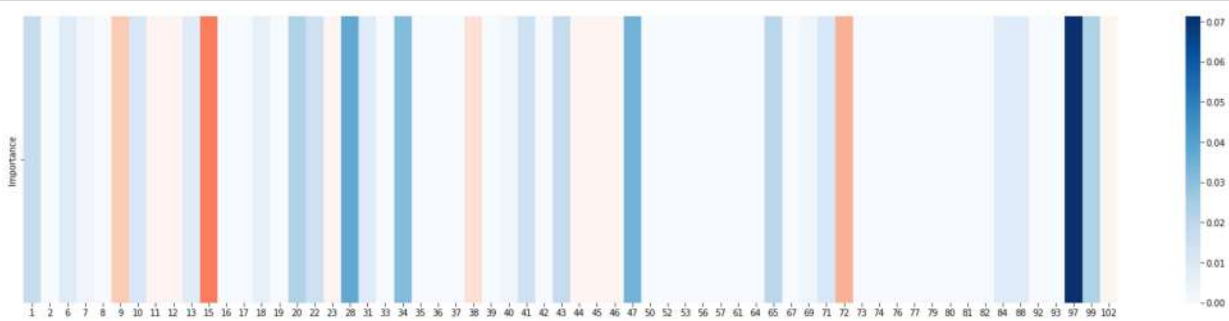
Explainability of the data belonging to the second class, in this case the Medium.

In [107]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



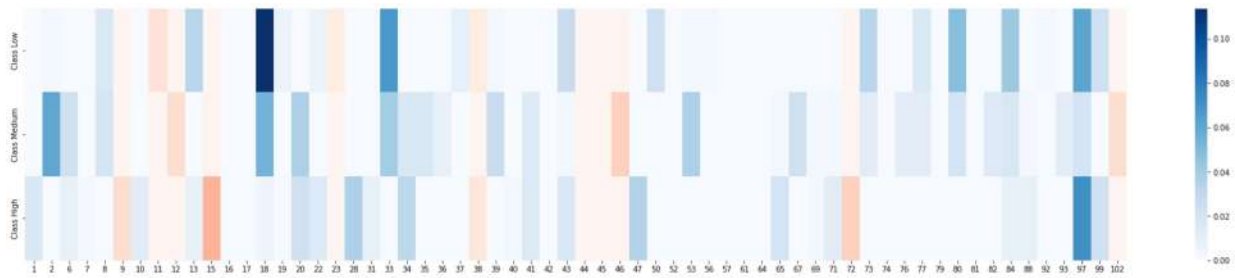
Explainability of the data belonging to the third class, in this case the High.

In [108]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [109]:
```
## Fusion of the previous heatmaps.
df_importance = pd.concat(map_class, axis=1)
df_importance.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_importance.transpose(), cmap = "Reds", cbar = False)
sns.heatmap(df_importance.transpose(), cmap = "Blues",yticklabels = True, xtickla
plt.show()
```



Identification of feature variables using SHAP

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of SHAP.

In [110]:
```
## Class index local variable.
iterable_class = -1
map_class = []
```

Individual explainability of the data belonging to the first class, in this case the Low.

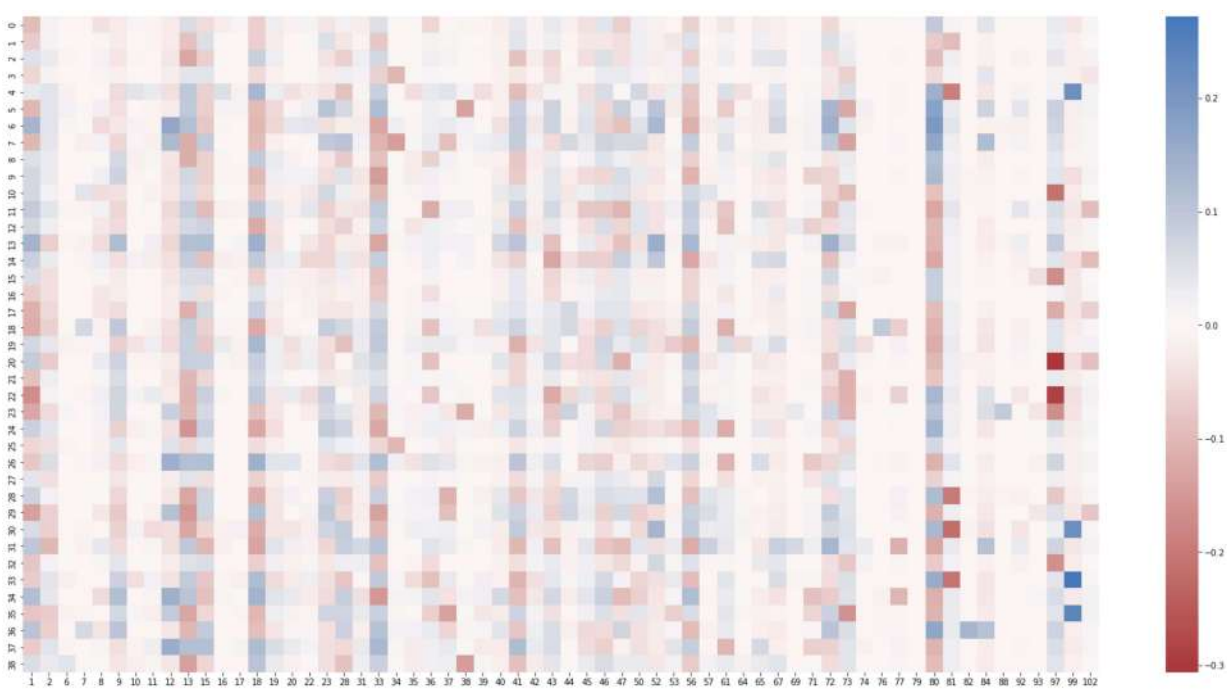
Definition of explainability data.

In [111]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

```
Calculating shap values...

  0%|          | 0/39 [00:00<?, ?it/s]
```

Generation of the heatmap.

In [112]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

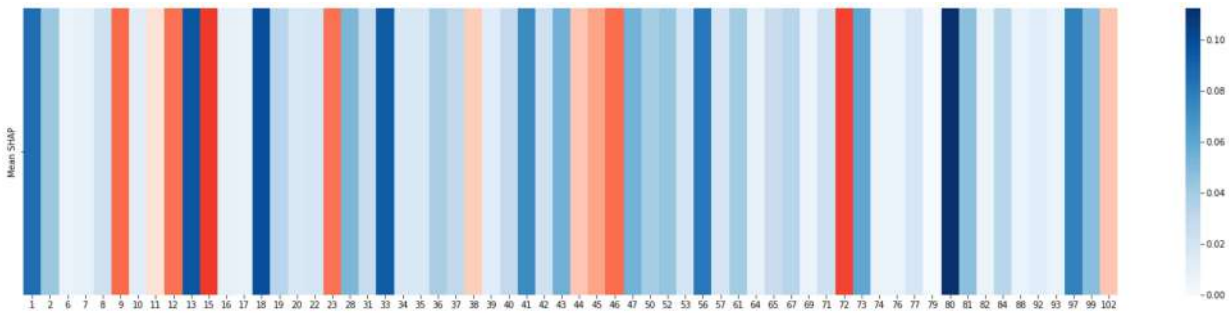


Fusion of the data used.

In [113]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [114]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



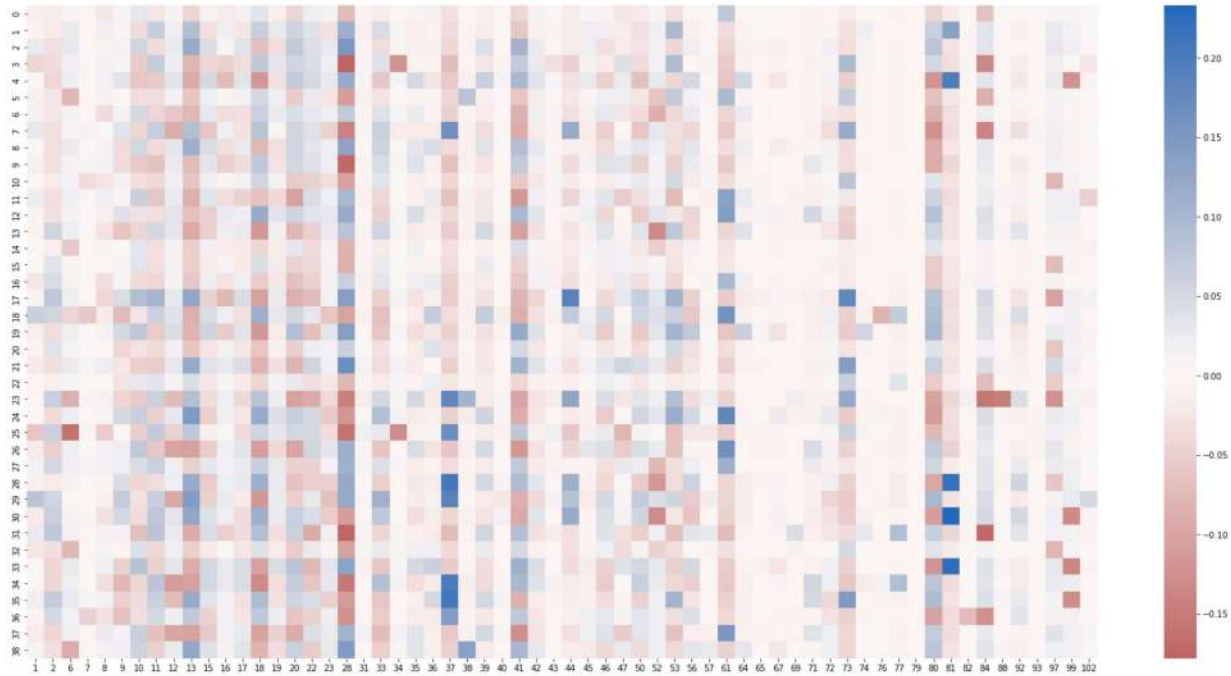
Individual explainability of the data belonging to the second class, in this case the Medium.

Definition of explainability data.

In [115]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [116]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

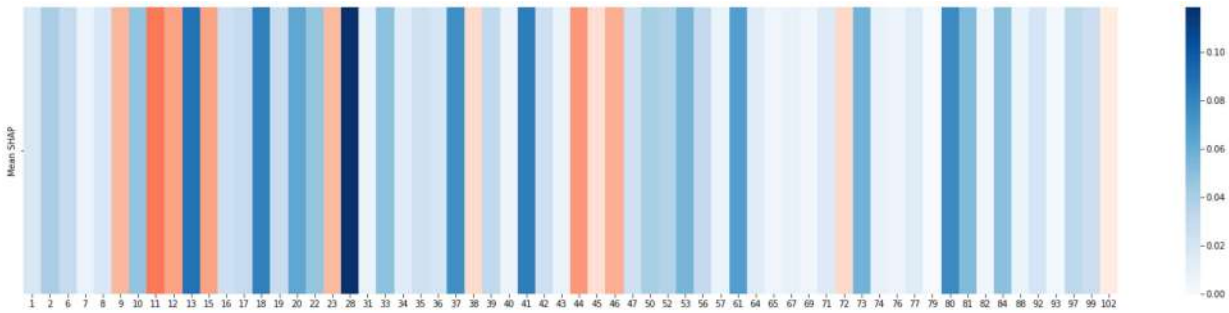


Fusion of the data used.

In [117]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [118]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



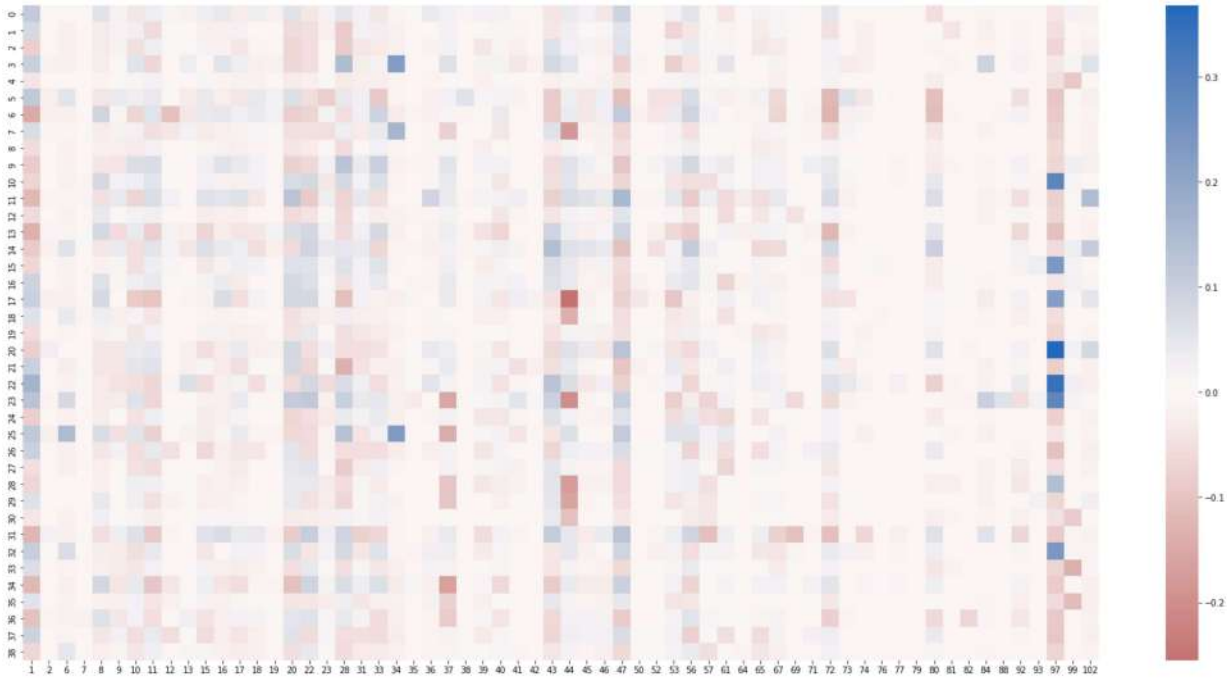
Individual explainability of data belonging to the third class, in this case High.

Definition of explainability data.

In [119]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [120]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

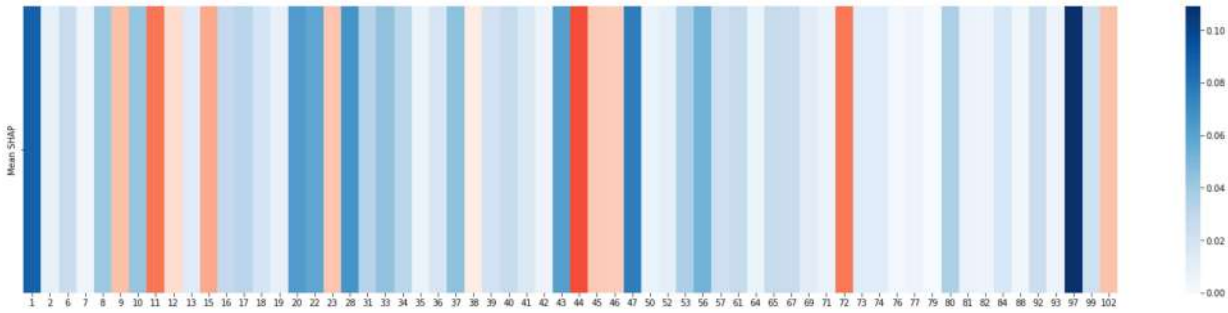


Fusion of the data used.

In [121]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

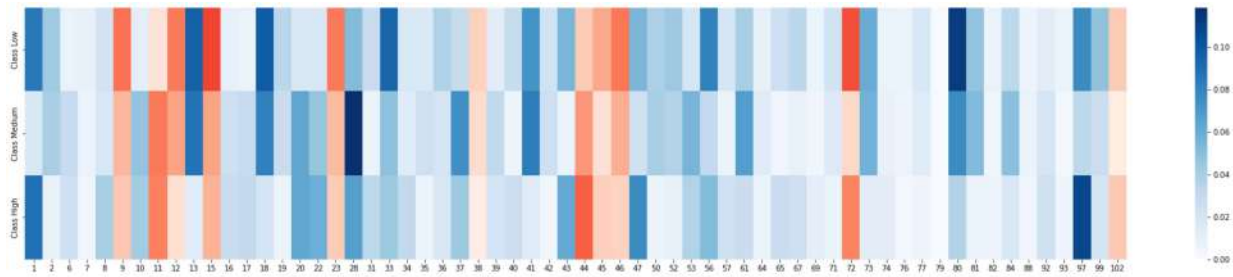
Generation of the heatmap.

In [122]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [123]:
```
## Fusion of the previous heatmaps.
df_shap = pd.concat(map_class, axis=1)
df_shap.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_shap.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_shap.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using LIME

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of LIME.

In [124]:
```
## Data preparation for explainability.
lime_exp = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorical
                                                  feature_names=train_cols, class
                                                  discretize_continuous=True)

## Class index local variable.
iterable_class = -1
map_class = []
```

Obtaining the data for the generation of the explanations by LIME about the Low, Medium and High classes.

In [125]:
```
## Local variables for LIME array data.
exp_matrix = [[] for _ in range(len(name_class))]

## Recovering the data needed for LIME.
for x in X_test.to_numpy():

    ## Auxiliary temporary variables.
    exp_list = [[] for _ in range(len(name_class))]

    ## Loading the data for explainability.
    exp = lime_exp.explain_instance(x, mlp.predict_proba, num_features = df.shape

    ## Recovering the data.
    for elements in range(0, len(name_class)):
        temp = exp.as_map()[elements]
        temp.sort(key=itemgetter(0))

        ## Saving data of the Low, Medium and High classes.
        for tup in temp:
            exp_list[elements].append(tup[1])

        exp_matrix[elements].append(exp_list[elements])
```

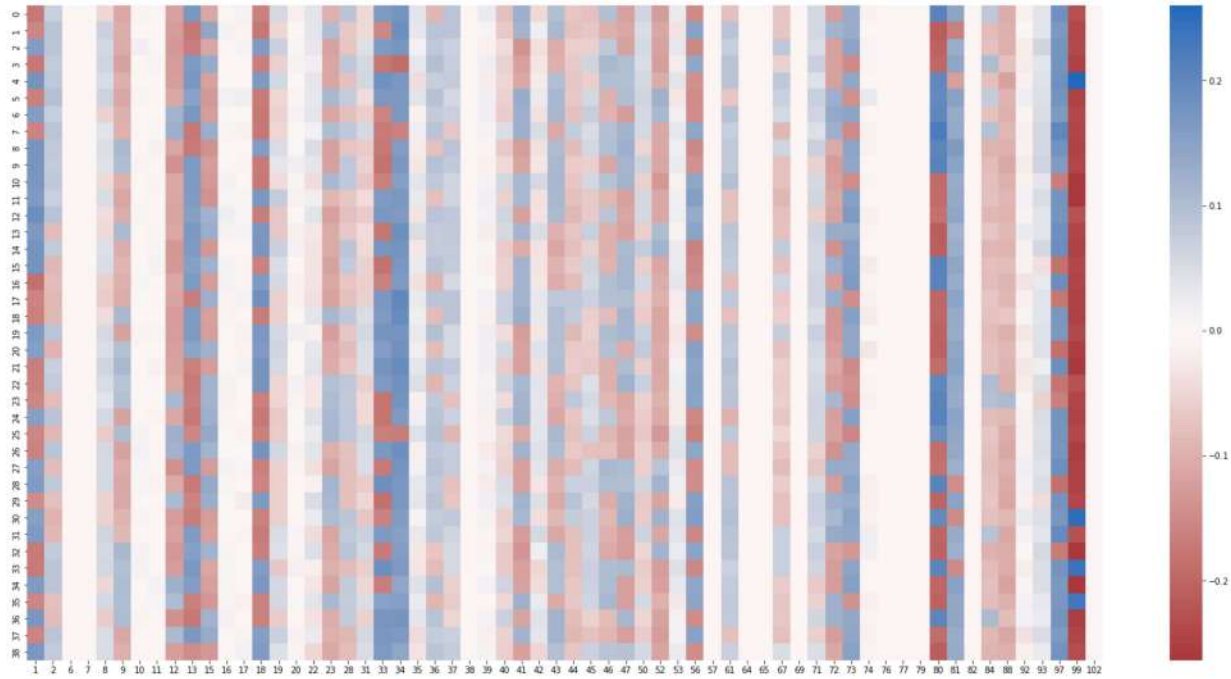
```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

Individual explainability of the data belonging to the first class, in this case the Low.

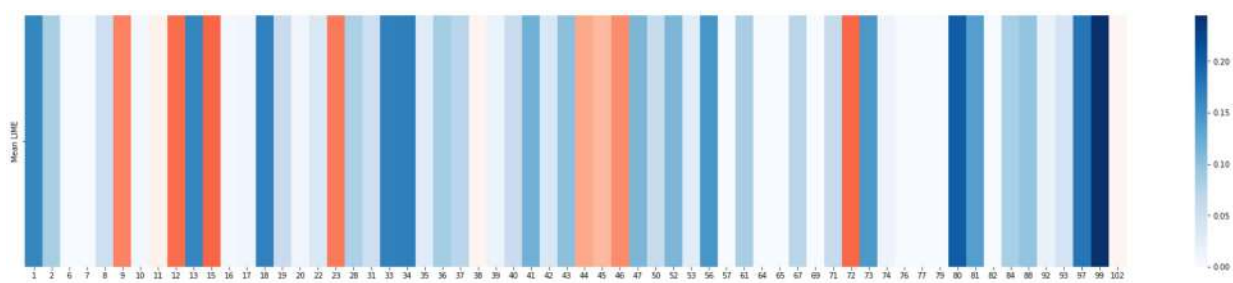
Generation of the heatmap.

In [126]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

In [127]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

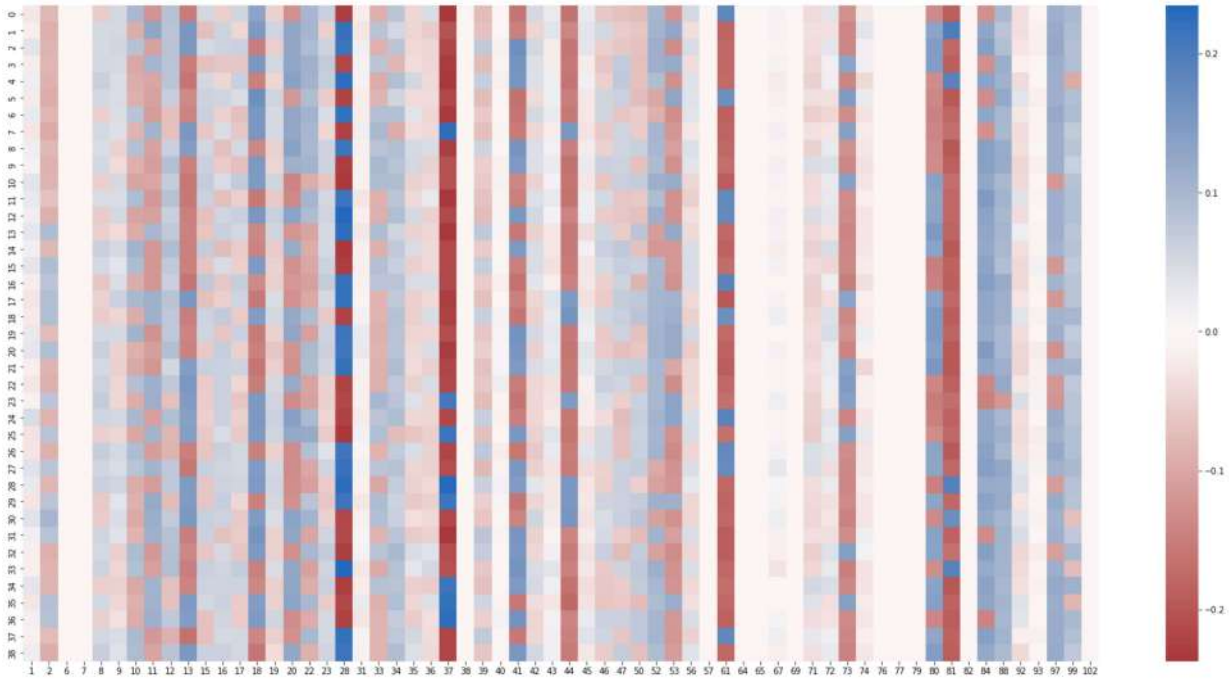


Individual explainability of the data belonging to the second class, in

this case the Medium.

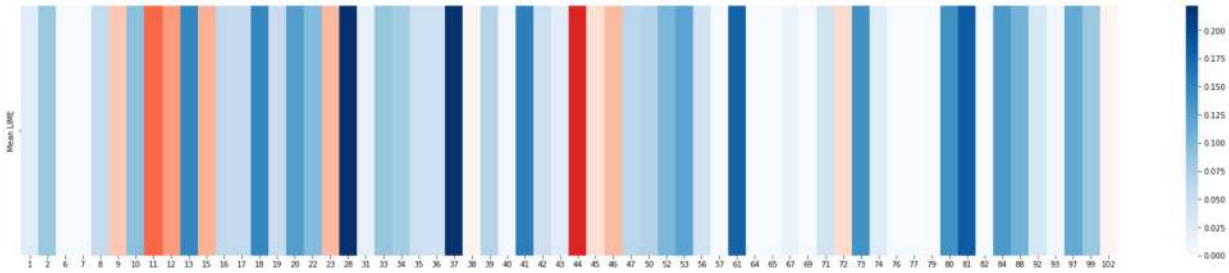
Generation of the heatmap.

In [128]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

In [129]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

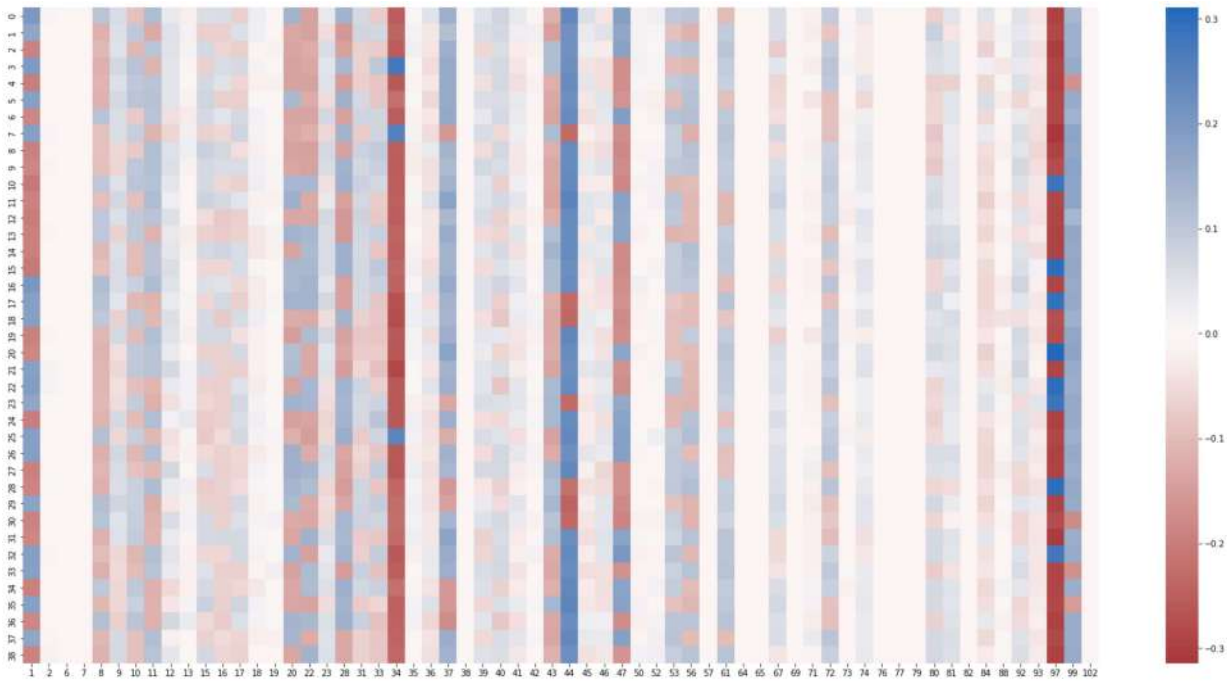


Individual explainability of data belonging to the third class, in this

case High.

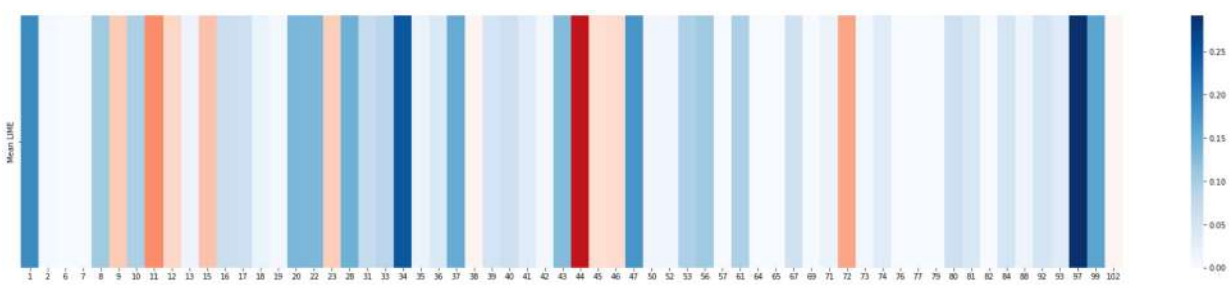
Generation of the heatmap.

In [130]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



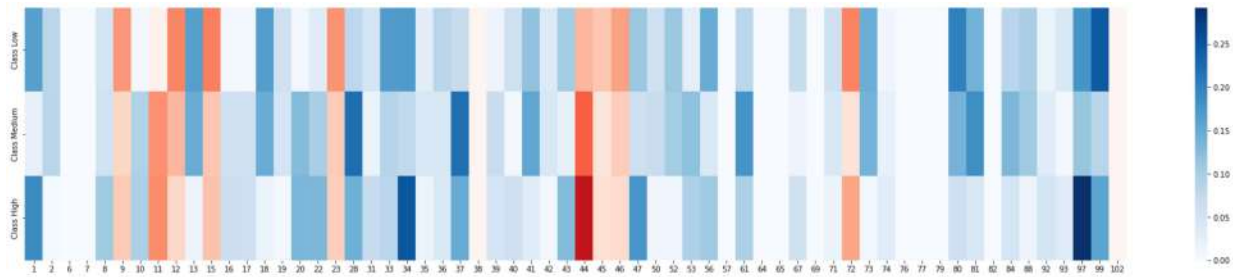
Fusion of the data used.

In [131]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [132]:
```
df_lime = pd.concat(map_class, axis=1)
df_lime.columns = name_class
plt.figure(figsize = (32,6))
sns.heatmap(df_lime.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_lime.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using ALE

It shows the main effect of each question compared to the probability that the model predicts in each class. This effect is shown for both responses. Since there are only 2 answers, the map is the same but with the colors reversed.

In [133]:
```
## Definition of explainability data.
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names=train_cols, target_names= target_r
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())

## Class index local variable.
iterable_class = -1
map_class = []
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
e names
X does not have valid feature names, but MLPClassifier was fitted with featur
```

Obtaining the data for the generation of the explanations by ALE about all the classes.

In [134]:
```
## Local variables for ALE array data.
ale_list = [[] for _ in range(len(name_class))]

## Recovering the data needed for ALE.
for array in proba_exp_lr.ale_values:

    ## Recovering the data.
    for elements in range(0, len(name_class)):

        ## Saving data of all the classes.
        ale_list[elements].append(array[0][elements])
```

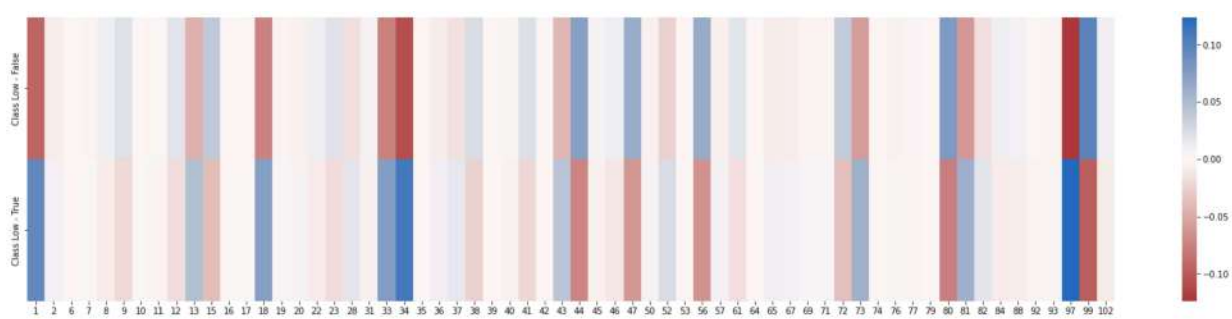
Data processing for the explainability of ALE.

In [135]:
```
## Data processing of the Low, Medium and High classes.
for elements in range(0, len(name_class)):
    ale_df = pd.DataFrame([ale_list [elements]])
    ale_df = pd.concat([ale_df.multiply(-1), ale_df])
    ale_df.index = [name_class [elements] + " - False", name_class [elements] + '
    ale_df.columns = column_Map
    map_class.append(ale_df)
```

Explainability of the data belonging to the first class, in this case the Low.

Generation of the heatmap.

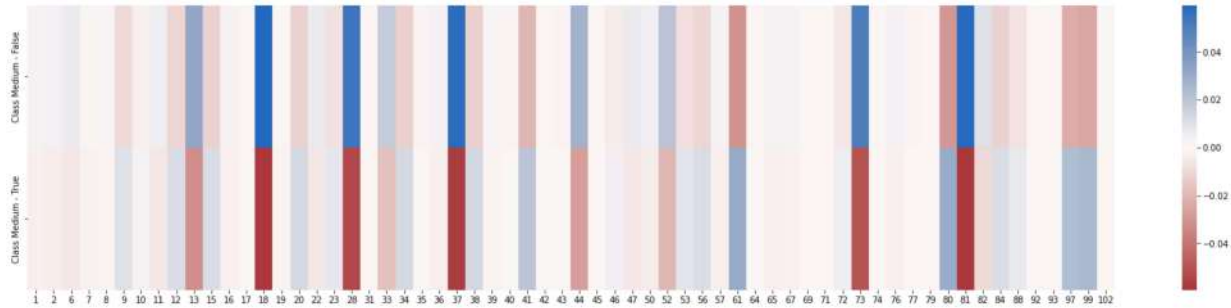
In [136]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the second class, in this case the Medium.

Generation of the heatmap.

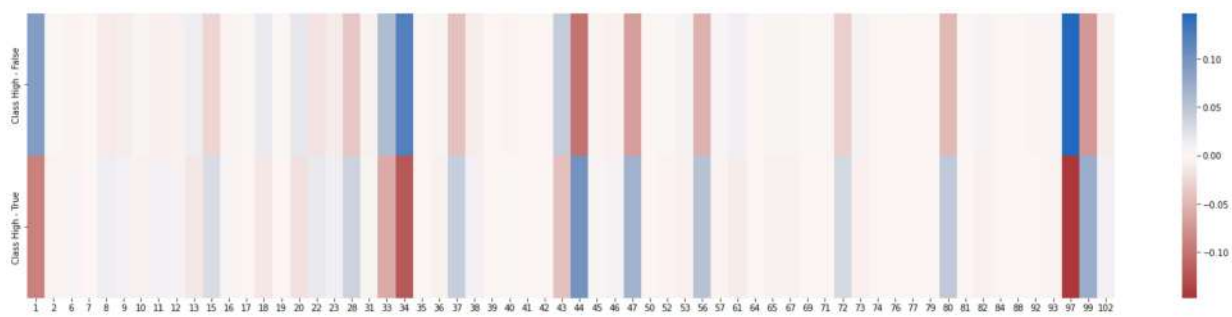
In [137]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the third class, in this case the High.

Generation of the heatmap.

In [138]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```

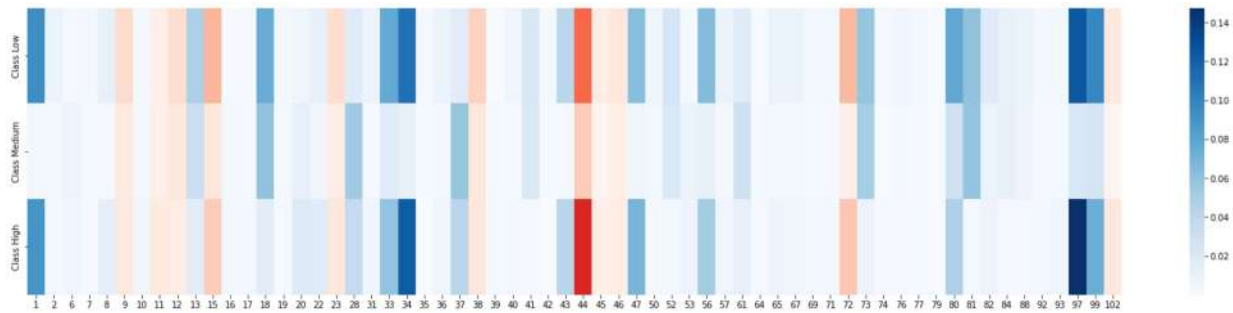


Fusion of the data used in all classes.

In [139]:
```
## Application of absolute value to data.
for index in range(0, len(map_class)):
    map_class [index] = map_class [index][0:1].abs()

## Union of the heatmaps of the Low, Medium and High class.
ale_df = pd.concat(map_class)
ale_df.index = name_class
ale_df = ale_df.transpose()
```

Generation of the heatmap.

In [140]:
```
plt.figure(figsize = (28,6))
sns.heatmap(ale_df.transpose(), cmap="Reds", cbar = False)
sns.heatmap(ale_df.transpose(), cmap="Blues", yticklabels = True, xticklabels = T
plt.show()
```



Comparing of Feature Variables Methods for Explainability

The global values of each of the methods used are taken, their maximums are obtained (the minimums are 0) and a percentage is calculated based on said maximum, which indicates how relevant the contribution is for each question.

In [141]:
```
## Class index local variable.
iterable_class = -1
```

Comparison of the explanation methods of the first class, in this case the Low.

The highest data of each method used is obtained.

In [142]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [143]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

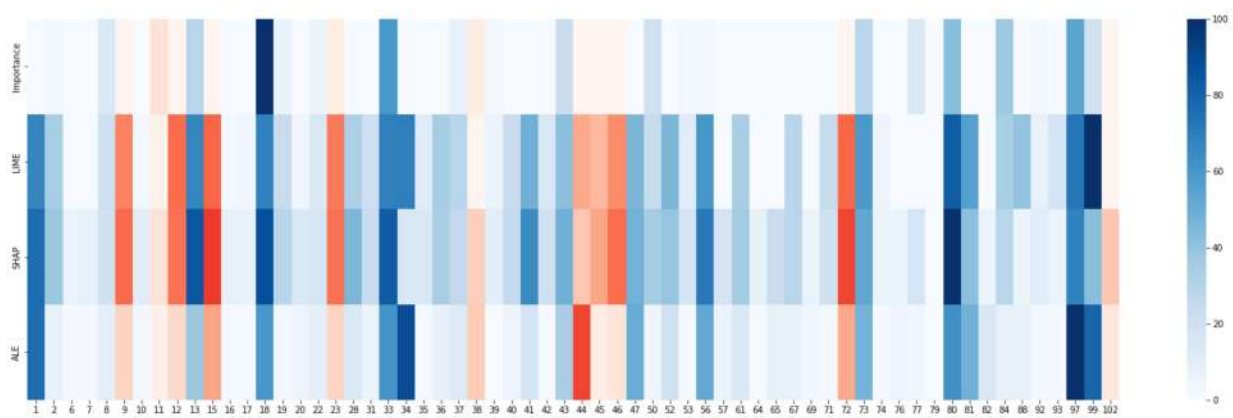
Out[143]:

	1	2	6	7	8	9	10	11
Importance	0.000000	2.857143	0.000000	0.000000	14.285714	0.000000	0.000000	11.428571
LIME	67.272538	33.993167	0.000000	0.000000	21.441326	42.447087	2.653440	2.124461
SHAP	75.861066	38.441910	6.593897	8.627190	19.516366	49.683706	10.019663	11.437691
ALE	75.889461	7.120971	0.604150	2.995356	9.322719	16.744014	1.987798	2.881302

4 rows × 62 columns

Generation of the heatmap.

In [144]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the second class, in this case the Medium.

The highest data of each method used is obtained.

In [145]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [146]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

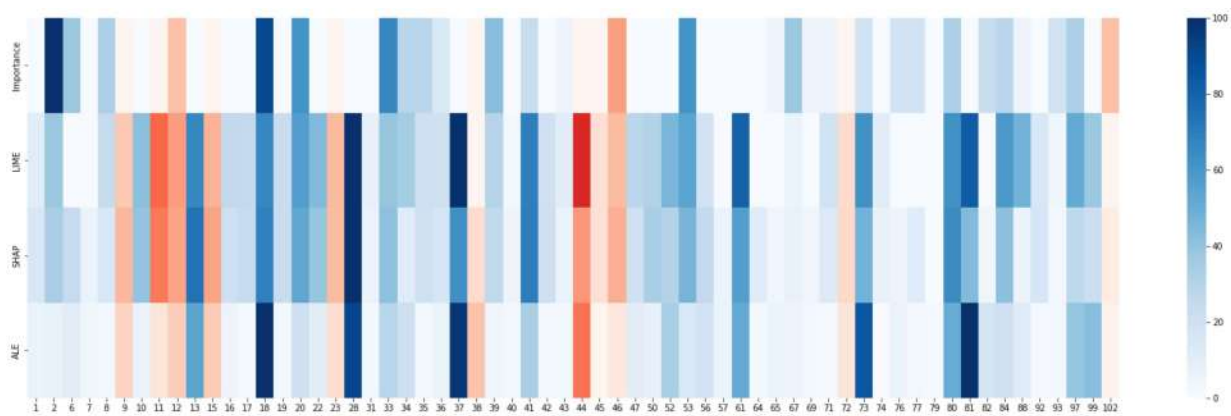
Out[146]:

	1	2	6	7	8	9	10	1
Importance	0.000000	100.000000	38.095238	0.000000	33.333333	0.000000	0.000000	0.00000
LIME	10.991949	38.313590	0.000000	0.000000	24.715883	20.636105	42.041794	50.29831
SHAP	14.544562	33.681028	25.258905	6.371820	15.909815	26.043530	40.054761	44.99284
ALE	6.229263	7.216456	10.893000	4.057098	3.263553	16.908867	6.154425	9.73976

4 rows × 62 columns

Generation of the heatmap.

In [147]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the third class, in this case the High.

The highest data of each method used is obtained.

In [148]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [149]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

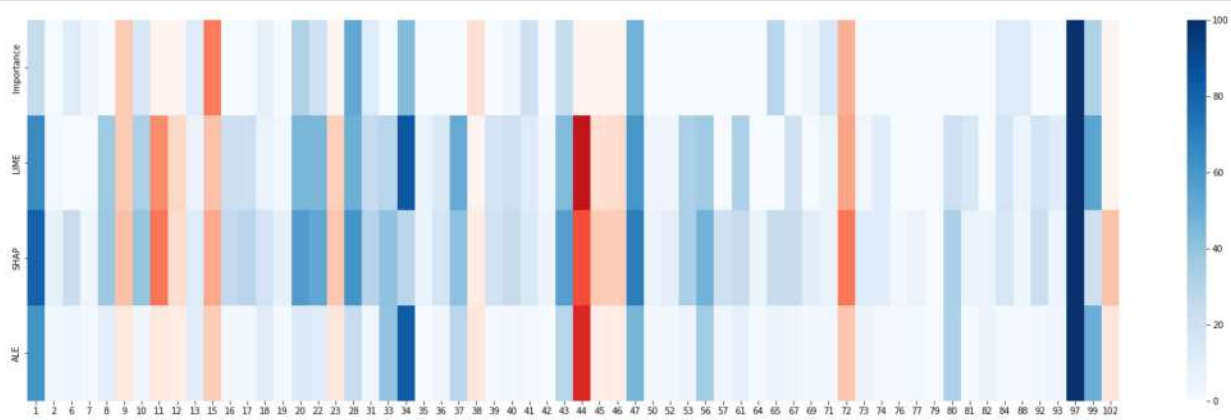
Out[149]:

	1	2	6	7	8	9	10	11
Importance	24.000000	0.000000	12.000000	4.000000	0.000000	20.000000	16.000000	0.000000
LIME	64.752092	1.846067	0.000000	0.000000	36.769892	19.888738	32.607604	38.811129
SHAP	81.201526	7.958980	22.870729	4.047643	37.239413	23.126373	39.456660	46.427804
ALE	61.287501	3.075014	3.887109	0.881292	9.154433	7.254560	4.154296	6.352058

4 rows × 62 columns

Generation of the heatmap.

In [150]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Safeguarding the items eliminated during this iteration.

In [151]:
```
## Removal of anomalous items selected by selection_mode.
for x in question_anomaly:

    ## Status change of the deleted items in the respective auxiliary dataset.
    ds.loc[x - 1,'Status'] = 'Delete'

## Save the changed statuses in the dataset.
ds.to_csv(path_dataset_qs, index = False)
```

Iteration 2

The first iteration of improvement of the model is carried out.

Preparing the data for the neural network

The data is prepared in order to be processed by the neural network.

In [152]:

```
## The iteration number is increased.
iteration += 1

## Loading data from the dataset.
df = pd.read_csv(path_dataset)
ds = pd.read_csv(path_dataset_qs)

## Elimination of unrelated items from the dataset.
if len(unrelated_questions) != 0:
    df = df.drop(df.columns[unrelated_questions[0]:unrelated_questions[1]], axis=

i = 0
if iteration == 1:
    ## Elimination of list of conflicting items declared in global variables.
    for x in list_contr:
        df.drop(df.columns[x - (i + 1)], axis = 1, inplace = True)
        i += 1
        ## Status change of the deleted question in the respective auxiliary data
        ds.loc[x - 1,'Status'] = 'Delete'

else:
    ## Elimination of the list of anomadic items defined in the improvement proce
    ds_delete = ds[ds.Status == "Delete"].reset_index(drop=True)
    for x in ds_delete['Question'].tolist():
        df.drop(df.columns[x - (i + 1)], axis = 1, inplace = True)
        i += 1

## Grouping of the target according to the defined dictionary.
if grouping_name != None and grouping_name != '':
    df[grouping_name] = df[grouping_name].map(class_dic)

## Assignment of local variables according to the data necessary for the neural r
train_cols = df.columns [0:-1]
label = df.columns [-1]
X = df [train_cols]
y = df [label]

## The prepared data of the dataset is printed.
df
```

Out[152]:

	1. Most of the time I have difficulty concentrating on simple tasks	2. I don't feel like doing my daily duties	6. I consider that my life is full of good things	7. My hobbies are still important to me	8. I'm still as punctual as I have always been	10. I have found that I can spend a lot of time scrolling the screen \nof my cell phone without searching or stopping at anything in particular	13. I felt more encouraged to do my daily activities before	16. I think my thoughts are strange or different from before	1 ab de
0	1	1	1	1	1	0	0	0	
1	1	0	1	1	1	0	1	1	
2	0	0	1	1	0	1	0	1	
3	0	0	1	1	1	0	1	0	
4	1	1	1	0	1	1	0	1	
...	...	...	...	...	...	...	...	...	
99	1	1	1	1	1	1	1	0	
100	1	1	1	1	0	0	1	0	
101	0	0	1	1	0	1	1	1	
102	1	0	1	1	1	1	0	0	
103	1	1	1	1	0	1	0	0	

104 rows × 52 columns

Oversample application

An oversample process is applied to level the amount of existing data given by the grouping.

In [153]:
```
## The number of elements present in the grouped target is printed.
y.value_counts()
```

Out[153]:
```
0    39
1    35
2    30
Name: Target, dtype: int64
```

Oversample application.

In [154]:
```
## Application of the oversample to the prepared data.
X, y = oversample.fit_resample(X, y)
```

Printing of the data after the application of the oversample.

In [155]:
```
## The results obtained after applying the oversample are printed.
y.value_counts()
```

Out[155]:
```
0    39
1    39
2    39
Name: Target, dtype: int64
```

Preparation of the parameters of the neural network

We proceed to define the parameters necessary for the training of the neural network.

In [156]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
```

Neural network training

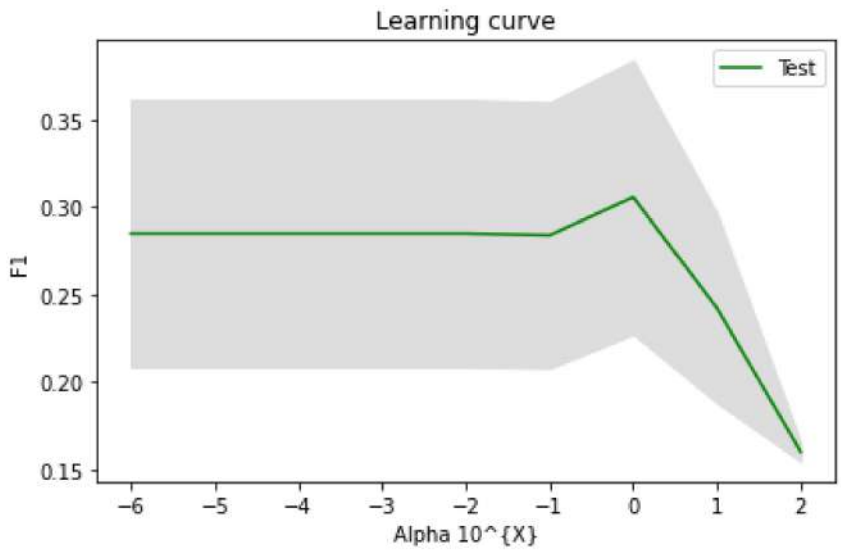
The alpha parameters and the initial learning rate are adjusted to obtain the best performance with the model using cross validation.

In [157]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_normal

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = 0.0001, max_iter =

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[157]:
```
([0.15999999999999998,
  0.24206284943127052,
  0.30571779236432206,
  0.28391696838958497,
  0.2849352225812138,
  0.2849352225812138,
  0.2849352225812138,
  0.2849352225812138,
  0.2849352225812138],
 [0.005443310539518168,
  0.05412997390137115,
  0.07837114850112376,
  0.0760679130846169,
  0.07647789235276374,
  0.07647789235276374,
  0.07647789235276374,
  0.07647789235276374,
  0.07647789235276374])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [158]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Alpha 10^{X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



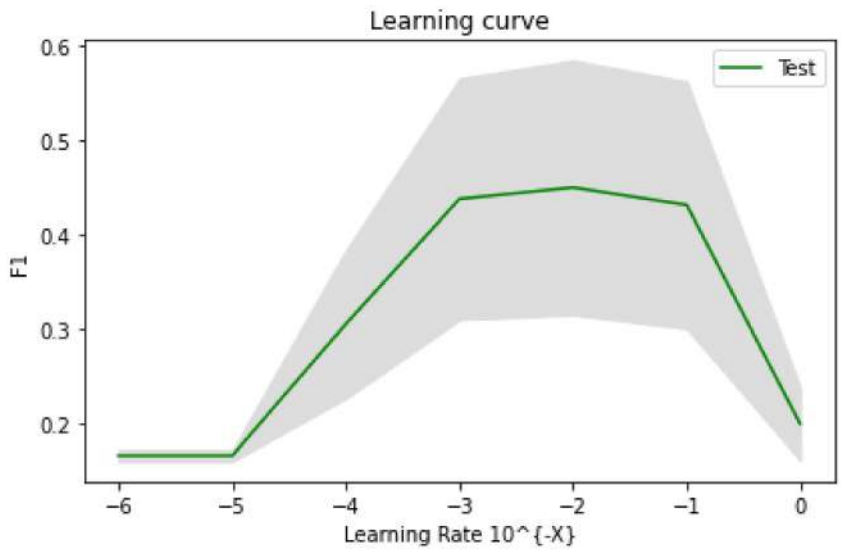
The training is performed again keeping the same parameters except for the alpha value that changes to 1.

In [159]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_alpha

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = regul_param, max_it

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[159]:
```
([0.20002265570686623,
  0.4314402264402264,
  0.4498720090825354,
  0.4378194102652307,
  0.30571779236432206,
  0.16659498207885304,
  0.16659498207885304],
 [0.03854518308236841,
  0.13062146376380732,
  0.13455453846003146,
  0.1275674740375547,
  0.07837114850112376,
  0.006020651927912844,
  0.006020651927912844])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [160]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Learning Rate 10^{-X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



Generation of the neural network model

The parameters of the final model are modified according to the data obtained in the training of the neural network.

In [161]:
```
## The value of the alpha parameter, learning_rate_init and random_state are char
## to the data in the global variables.
mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solver =
                    learning_rate = 'constant', learning_rate_init = lear
                    random_state = random_state_model)

## We print the data of the final generated model.
mlp.fit(X_train, y_train)
```

Out[161]:
```
MLPClassifier
MLPClassifier(activation='logistic', alpha=0.001, hidden_layer_sizes=(10,),
              learning_rate_init=0.1, max_iter=100000, random_state=42)
```

We safeguard the final generated model

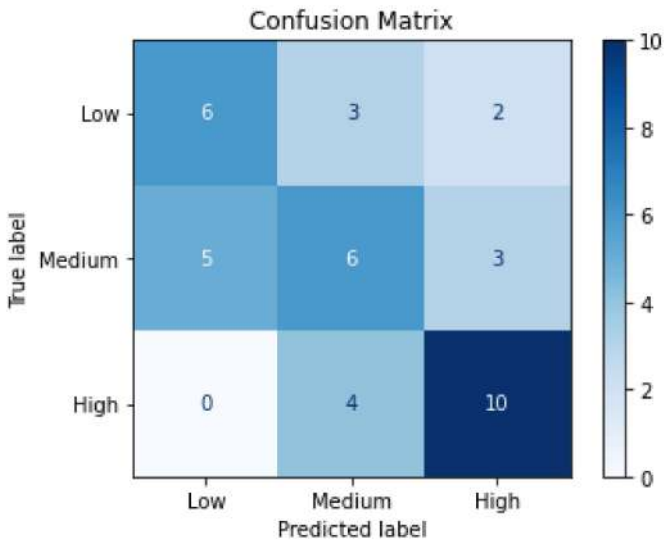
In [162]:
```
joblib.dump(mlp,"model_depression_i" + str(iteration) + ".pkl")
```

Out[162]:
```
['model_depression_i2.pkl']
```

Generation of the confusion matrix of the model

We obtain the statistics of the final model to generate its confusion matrix.

In [163]:
```
## The data for the generation of the confession matrix are defined.
confusion_matrix = ConfusionMatrixDisplay.from_estimator(mlp, X_test, y_test, dis
                                                         cmap = plt.cm.Blues)
confusion_matrix.ax_.set_title("Confusion Matrix")
plt.show()
```



We get the detailed statistics of the final model.

In [164]:
```
## The necessary detailed prediction is generated.
y_pred = mlp.predict(X_test)

## The detailed statistics of the model are printed.
print('Classification accuracy =',accuracy_score(y_test,y_pred) * 100,'%\n')
print(classification_report(y_test,y_pred))
```

```
Classification accuracy = 56.41025641025641 %

              precision    recall  f1-score   support

           0       0.55      0.55      0.55        11
           1       0.46      0.43      0.44        14
           2       0.67      0.71      0.69        14

    accuracy                           0.56        39
   macro avg       0.56      0.56      0.56        39
weighted avg       0.56      0.56      0.56        39
```

The data obtained is saved for later graphing.

In [165]:
```
accuracy = accuracy_score(y_test,y_pred) * 100
clf_rep = precision_recall_fscore_support(y_test,y_pred)

## The number of items used in the iteration is retrieved.
ds_delete = ds[ds.Status == "Delete"]
ds_delete.reset_index(inplace=True, drop=True)

## All the metrics of the confusion matrix are obtained.
metrics = [",".join(map(str, ds_delete['Question'].tolist())),
           accuracy]

for index in range(0, len(name_class)):
    metrics.append(clf_rep[0][index])
    metrics.append(clf_rep[1][index])
    metrics.append(clf_rep[2][index])
    metrics.append(clf_rep[3][index])

## The names of the columns of the dataset are defined.
columns = ['Question', 'Acurracy global']

for index in range(0, len(name_class)):
    columns.append('Precision_' + str(index))
    columns.append('Recall_' + str(index))
    columns.append('F1_score_' + str(index))
    columns.append('Support_' + str(index))

## A new row of the dataset is generated with all the data.
data_metrics = pd.DataFrame([metrics], columns = columns)

## The data is saved to the dataset.
data_metrics.to_csv(path_dataset_qd, mode = 'a', header = False, index = False)

## The data added to the dataset is printed.
data_metrics
```

Out[165]:

	Question	Acurracy global	Precision_0	Recall_0	F1_score_0	Supp
0	3,4,5,9,11,12,14,15,21,23,24,25,26,27,29,30,32...	56.410256	0.545455	0.545455	0.545455	

Improvement process of the neural network model.

The application of the explanation method by ALE allows to determine the behavior of the neural network and thereby improve it by discarding non-significant data for it.

Slope Analysis of ALE Plots

In [166]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

We proceed to obtain the graphs to obtain the slopes of the selected classes.

In [167]:
```
## The question metrics dataset is loaded.
dm = pd.DataFrame()

## The number of items used in the iteration is retrieved.
ds_item = ds[ds.Status == "In use"]
ds_item.reset_index(inplace=True, drop=True)

## We get the slopes from the ALE data.
for i in range(df.shape[1]-1):
    structure_data = {'Question': ds_item.loc[i, "Question"]}

    for index in selection_class:
        slope = proba_exp_lr.ale_values [i][1][index] - proba_exp_lr.ale_values [
        structure_data['Slope ' + str(target_names[index])] = slope
        structure_data['Threshold ' + str(target_names[index])] = 'NA'
        structure_data['Anomaly ' + str(target_names[index])] = 'NA'

    dm = dm.append(structure_data, ignore_index=True)

## The current rating data dataset is printed.
dm
```

```
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
```

Slope analysis and threshold application

The slope obtained is analyzed and those that are above or below the positive and negative threshold are selected.

In [168]:
```
for i in range(df.shape[1]-1):
    for index in selection_class:

        ## The slopes are selected according to the values of the thresholds of t
        if dm.loc[i, "Slope " + str(target_names[index])] >= positive_threshold:
            dm.loc[i, "Threshold " + str(target_names[index])] = 1
        elif dm.loc[i, "Slope " + str(target_names[index])] <= negative_threshold
            dm.loc[i, "Threshold " + str(target_names[index])] = 0

## The current rating data dataset is printed.
dm
```

Out[168]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	1	-0.078007	0	NA	0.060910	1	NA
1	2	-0.009548	NA	NA	-0.002863	NA	NA
2	6	0.004662	NA	NA	-0.006732	NA	NA
3	7	-0.002550	NA	NA	0.002728	NA	NA
4	8	0.056086	1	NA	-0.069883	0	NA
5	10	-0.011161	0	NA	0.009315	NA	NA
6	13	-0.060551	0	NA	0.004189	NA	NA
7	16	-0.000957	NA	NA	0.002491	NA	NA
8	17	0.006024	NA	NA	0.021221	1	NA
9	18	-0.013840	0	NA	0.001035	NA	NA
10	19	0.007562	NA	NA	-0.020408	0	NA
11	20	-0.018922	0	NA	0.053429	1	NA
12	22	-0.000985	NA	NA	-0.011379	0	NA
13	28	0.042039	1	NA	-0.090640	0	NA
14	31	0.100081	1	NA	-0.090169	0	NA
15	33	-0.025636	0	NA	-0.000395	NA	NA
16	34	-0.024138	0	NA	0.015840	1	NA
17	35	0.024024	1	NA	-0.002917	NA	NA
18	36	-0.005089	NA	NA	0.001344	NA	NA
19	37	-0.116651	0	NA	-0.297465	0	NA
20	39	0.007830	NA	NA	-0.025644	0	NA
21	40	0.007482	NA	NA	-0.002813	NA	NA
22	41	0.090997	1	NA	-0.008499	NA	NA
23	42	0.005507	NA	NA	0.000417	NA	NA
24	43	-0.075720	0	NA	0.081373	1	NA
25	47	0.032606	1	NA	-0.044468	0	NA
26	50	0.052174	1	NA	0.000956	NA	NA

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
27	52	-0.003530	NA	NA	0.001521	NA	NA
28	53	-0.010847	0	NA	0.133234	1	NA
29	56	0.172758	1	NA	-0.232054	0	NA
30	57	-0.005832	NA	NA	0.003360	NA	NA
31	61	0.022238	1	NA	0.010717	1	NA
32	64	0.000651	NA	NA	0.007177	NA	NA
33	65	0.021270	1	NA	0.016541	1	NA
34	67	-0.004709	NA	NA	0.001697	NA	NA
35	69	-0.001326	NA	NA	0.002261	NA	NA
36	71	-0.001712	NA	NA	0.000001	NA	NA
37	73	-0.015070	0	NA	-0.006194	NA	NA
38	74	-0.000797	NA	NA	0.002747	NA	NA
39	76	-0.000559	NA	NA	-0.000642	NA	NA
40	77	0.007391	NA	NA	0.001862	NA	NA
41	79	0.003297	NA	NA	-0.003971	NA	NA
42	80	0.040535	1	NA	-0.023198	0	NA
43	81	-0.170393	0	NA	0.097061	1	NA
44	82	-0.046482	0	NA	0.044328	1	NA
45	84	0.081859	1	NA	-0.084221	0	NA
46	88	0.208674	1	NA	-0.175313	0	NA
47	92	-0.002075	NA	NA	0.008080	NA	NA
48	93	-0.005819	NA	NA	-0.006279	NA	NA
49	97	-0.242734	0	NA	0.314953	1	NA
50	99	0.212351	1	NA	-0.172973	0	NA

Analysis and determination of anomalous items

The anomalous items present in the model are determined based on their slope and expected response from the expert.

In [169]:
```
## The dataset of expected responses is loaded.
da = pd.read_csv(path_dataset_qa)

## Delete list of items deleted in this iteration.
i = 0
for x in ds_delete['Question']:
    da.drop(da.index[x - (i + 1)], axis = 0, inplace = True)
    i += 1

## Indexes are restored for later use.
da.reset_index(inplace=True, drop=True)

## The dataset of expected responses is printed.
da
```

Out[169]:

	Number	Question	RE
0	1	Most of the time I have difficulty concentrati...	1
1	2	I don't feel like doing my daily duties	1
2	6	I consider that my life is full of good things	0
3	7	My hobbies are still important to me	0
4	8	I'm still as punctual as I have always been	0
5	10	I have found that I can spend a lot of time sc...	1
6	13	I felt more encouraged to do my daily activiti...	1
7	16	I think my thoughts are strange or different f...	1
8	17	I feel guilty about the decisions that I have ...	1
9	18	I don't feel as comfortable with my body as I ...	1
10	19	I don't feel successful compared to others	1
11	20	It is difficult for me to make decisions even ...	1
12	22	It is not difficult for me to understand somet...	0
13	28	The decisions I have made so far have been the...	0
14	31	It is easy for me to choose a photograph of my...	0
15	33	I have trouble remembering things easily	1
16	34	In recent months I have had discussions with m...	1
17	35	I constantly imagine that something will go wr...	1
18	36	I am afraid of being worng when doing my homework	1
19	37	I'm not too worried about what might happen in...	0
20	39	Everything will be alright	0
21	40	I can easily blank my mind	0
22	41	I am bothered by insignificant things that wer...	1
23	42	I find it uncomfortable to be in a crowded place	1
24	43	Sometimes i feel trapped	1
25	47	I can clearly express to others how I feel	0

	Number	Question	RE
26	50	I imagine that at any moment a disaster of the...	1
27	52	Being locked in an elevator would be the worst...	1
28	53	I'm bothered by people walking slowly in front...	1
29	56	I manage my schedule as I always have	0
30	57	It bothers me to feel that people on the stree...	1
31	61	Sometimes I forget what I wanted to say becaus...	1
32	64	I can put down my cell phone and dedicate myse...	0
33	65	I worry that people will not understand what I...	1
34	67	I get angry easily	1
35	69	It is not important for me to meet set dates	1
36	71	I use lies just to get out of certain problems	1
37	73	I have difficulty making elaborate plans	1
38	74	People have problems because of themselves	1
39	76	Laws are not as important as others think	1
40	77	I would regret betraying a friend	0
41	79	It is easy for me to work in a team	0
42	80	It is important to help people when they need it	0
43	81	I have punched someone or thought of doing it	1
44	82	If it was necessary I would pretend to be some...	1
45	84	After an argument I usually go over what happe...	0
46	88	Sometimes people need physical strength to und...	1
47	92	It does not seem serious to me to have some debts	1
48	93	People steal because they have needs	1
49	97	People who commit crimes have their reasons fo...	1
50	99	It is important to respect turns	0

We proceed to identify the anomadic items present in the model present in this iteration.

In [170]:

```
for i in range(df.shape[1]-1):

    ## Local variable to indicate the index of the selected classes.
    i_index = 0

    for index in selection_class:

        ## Determine if it fails based on the expected response parameters of the
        if dm.loc[i, "Threshold "  + str(target_names[index])] == 1 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 1 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 0 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 0 and da.loc[i,
            dm.loc[i, "Anomaly "  + str(target_names[index])] = expected_response

        ## Increase of the index of the selected classes.
        i_index += 1

## The final grade data dataset is printed.
dm
```

Out[170]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	1	-0.078007	0	0	0.060910	1	0
1	2	-0.009548	NA	NA	-0.002863	NA	NA
2	6	0.004662	NA	NA	-0.006732	NA	NA
3	7	-0.002550	NA	NA	0.002728	NA	NA
4	8	0.056086	1	0	-0.069883	0	0
5	10	-0.011161	0	0	0.009315	NA	NA
6	13	-0.060551	0	0	0.004189	NA	NA
7	16	-0.000957	NA	NA	0.002491	NA	NA
8	17	0.006024	NA	NA	0.021221	1	0
9	18	-0.013840	0	0	0.001035	NA	NA
10	19	0.007562	NA	NA	-0.020408	0	1
11	20	-0.018922	0	0	0.053429	1	0
12	22	-0.000985	NA	NA	-0.011379	0	0
13	28	0.042039	1	0	-0.090640	0	0
14	31	0.100081	1	0	-0.090169	0	0
15	33	-0.025636	0	0	-0.000395	NA	NA
16	34	-0.024138	0	0	0.015840	1	0
17	35	0.024024	1	1	-0.002917	NA	NA
18	36	-0.005089	NA	NA	0.001344	NA	NA
19	37	-0.116651	0	1	-0.297465	0	0

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
20	39	0.007830	NA	NA	-0.025644	0	0
21	40	0.007482	NA	NA	-0.002813	NA	NA
22	41	0.090997	1	1	-0.008499	NA	NA
23	42	0.005507	NA	NA	0.000417	NA	NA
24	43	-0.075720	0	0	0.081373	1	0
25	47	0.032606	1	0	-0.044468	0	0
26	50	0.052174	1	1	0.000956	NA	NA
27	52	-0.003530	NA	NA	0.001521	NA	NA
28	53	-0.010847	0	0	0.133234	1	0
29	56	0.172758	1	0	-0.232054	0	0
30	57	-0.005832	NA	NA	0.003360	NA	NA
31	61	0.022238	1	1	0.010717	1	0
32	64	0.000651	NA	NA	0.007177	NA	NA
33	65	0.021270	1	1	0.016541	1	0
34	67	-0.004709	NA	NA	0.001697	NA	NA
35	69	-0.001326	NA	NA	0.002261	NA	NA
36	71	-0.001712	NA	NA	0.000001	NA	NA
37	73	-0.015070	0	0	-0.006194	NA	NA
38	74	-0.000797	NA	NA	0.002747	NA	NA
39	76	-0.000559	NA	NA	-0.000642	NA	NA
40	77	0.007391	NA	NA	0.001862	NA	NA
41	79	0.003297	NA	NA	-0.003971	NA	NA
42	80	0.040535	1	0	-0.023198	0	0
43	81	-0.170393	0	0	0.097061	1	0
44	82	-0.046482	0	0	0.044328	1	0
45	84	0.081859	1	0	-0.084221	0	0
46	88	0.208674	1	1	-0.175313	0	1
47	92	-0.002075	NA	NA	0.008080	NA	NA
48	93	-0.005819	NA	NA	-0.006279	NA	NA
49	97	-0.242734	0	0	0.314953	1	0
50	99	0.212351	1	0	-0.172973	0	0

Removing identified anomadic items from the model

We proceed to eliminate the identified anomadic items so as not to use them in the next iteration.

In [171]:
```
## Local variable for the elimination of the anomaly items.
instructions = ''
init = False
question_anomaly = []

if selection_mode == 1:

    ## Items that fail in the defined class are retrieved.
    dm_delete = dm[dm['Anomaly ' + str(target_names[select_class_mode])] == 1]
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 2:

    ## Items that fail in classes defined with the or condition type are retrieve
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 1"
            init = True
        else:
            instructions += " or `Anomaly " + str(target_names[index]) + "` == 1'

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 3:

    ## Items that fail in classes defined with the and condition type are retriev
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 1"
            init = True
        else:
            instructions += " and `Anomaly " + str(target_names[index]) + "` == 1

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 4:

    ## Out-of-threshold items are retrieved in classes defined with the conditior
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 'NA'"
            init = True
        else:
            instructions += " and `Anomaly " + str(target_names[index]) + "` == '

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

## The selected blank items are printed according to the selection_mode.
question_anomaly
```

Out[171]: [19, 88]

Model explainability process

The data used for the test of the neural network model is shown.

In [172]: X_test

Out[172]:

	1. Most of the time I have difficulty concentrating on simple tasks	2. I don't feel like doing my daily duties	6. I consider that my life is full of good things	7. My hobbies are still important to me	8. I'm still as punctual as I have always been	10. I have found that I can spend a lot of time scrolling the screen \nof my cell phone without searching or stopping at anything in particular	13. I felt more encouraged to do my daily activities before	16. I think my thoughts are strange or different from before	17. I gu about decisi tl h m
69	1	0	1	1	0	0	0	0	
46	1	0	1	1	1	1	1	1	
58	0	0	1	1	1	0	1	1	
114	1	0	1	1	1	1	0	0	
73	0	0	1	1	1	1	0	0	
98	1	0	0	1	1	1	0	1	
31	0	0	1	1	0	0	0	0	
53	1	0	1	1	1	1	1	1	
65	0	0	1	1	1	0	1	0	
96	0	0	1	1	1	1	0	0	
95	0	0	1	0	0	1	0	1	
97	0	0	1	1	1	0	0	0	
2	0	0	1	1	0	1	0	1	
62	0	1	1	1	0	1	0	1	
110	0	0	0	1	0	0	0	0	
55	0	1	1	1	1	0	0	1	
103	1	1	1	1	0	1	0	0	
100	1	1	1	1	0	0	1	0	
66	1	1	0	0	0	1	0	1	
44	0	0	1	1	1	0	0	0	
77	0	1	1	1	1	1	0	1	
17	1	0	1	1	1	1	1	1	
81	1	0	1	1	1	0	1	1	
74	1	1	0	1	1	1	1	1	

56	0	0	1	1	1	0	1	1
94	1	1	0	1	0	1	1	1
35	1	0	1	1	1	0	0	1
38	0	1	1	1	1	0	0	1
93	0	0	1	1	1	1	1	1
48	1	1	1	1	0	1	1	1
33	0	1	1	1	0	1	1	1
59	0	1	1	1	1	1	0	0
92	1	0	0	1	1	0	0	1
42	1	0	1	1	1	0	0	1
10	0	0	1	1	0	1	0	1
85	1	1	1	1	1	1	1	1
32	0	1	1	0	0	1	1	1
78	1	0	1	1	1	0	0	1
19	0	0	0	1	1	0	1	1

39 rows × 51 columns

Explanation of the model using ALE

In analysis of the model through the results obtained by ALE, it is possible to identify the behavior of the neural network.

In [173]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

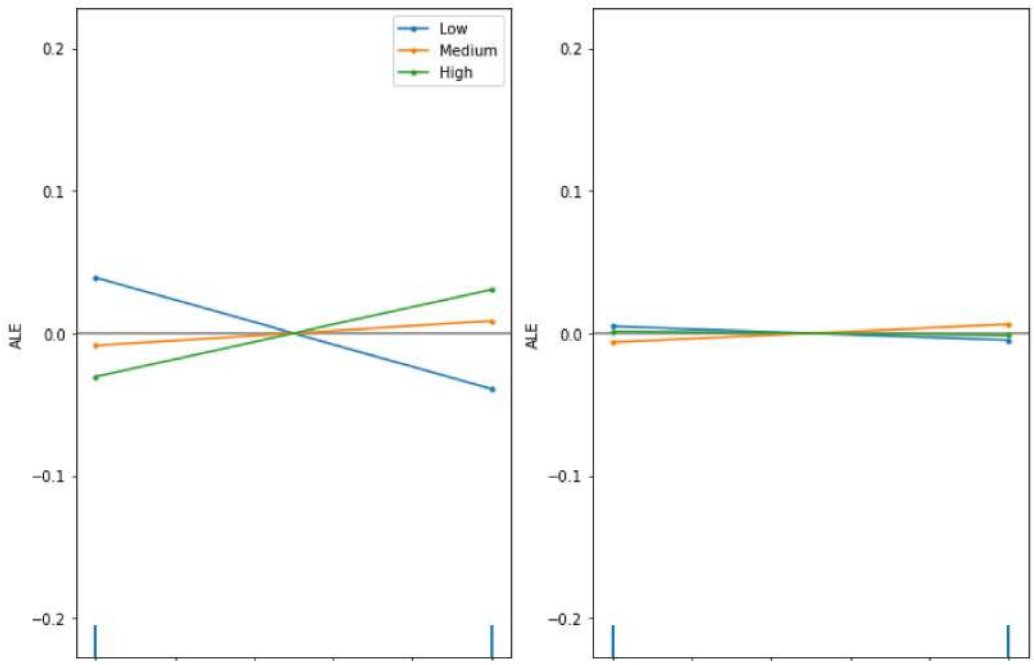
```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names


X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

The graphs of all the data present in the dataset used are shown.

In [174]:
```
plot_ale(proba_exp_lr, n_cols=2, features=list(range(df.shape[1]-1)),fig_kw={'fig
```



Explanation of the model using LIME

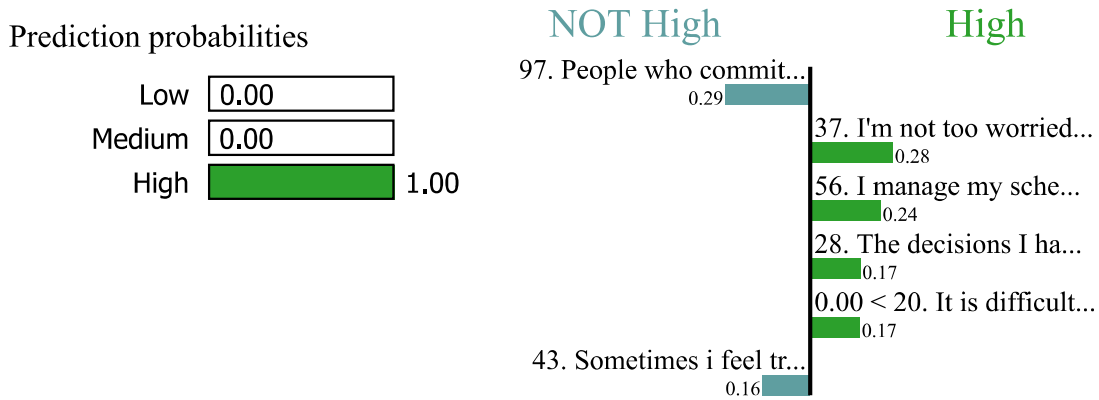
We proceed to explain the improved model using LIME for the Low, Medium and High classes.

In [175]:
```
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorica
                                                   feature_names=train_cols, clas
                                                   discretize_continuous=True)
```

The first individual present in the test data is explained.

In [176]:
```
exp = explainer.explain_instance(X_test.to_numpy()[0], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

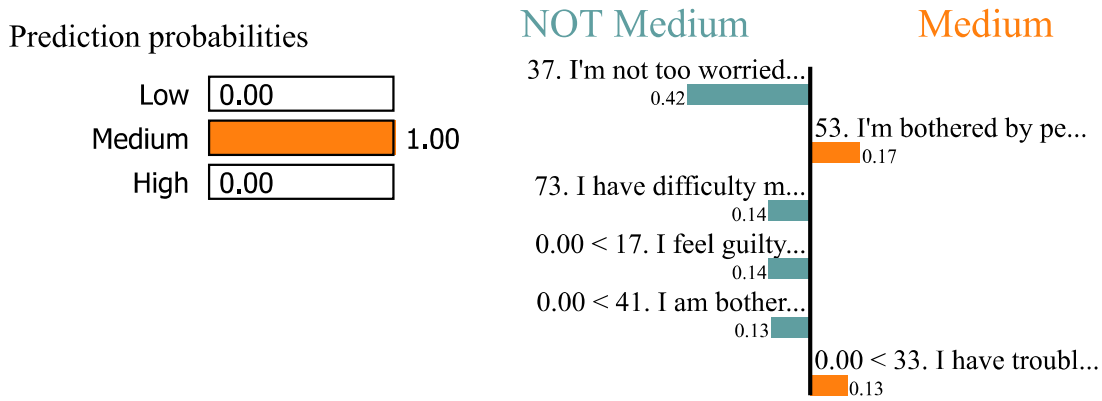


Feature	Value
97. People who commit crimes have their reasons for doing it	0.00
37. I'm not too worried about what might happen in a few weeks	0.00
56. I manage my schedule as I always have	0.00
28. The decisions I have made so far have been the right ones	0.00
20. It is difficult for me to make decisions even if they are simple	1.00
43. Sometimes i feel trapped	0.00

The second individual present in the test data is explained.

In [177]:
```
exp = explainer.explain_instance(X_test.to_numpy()[1], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

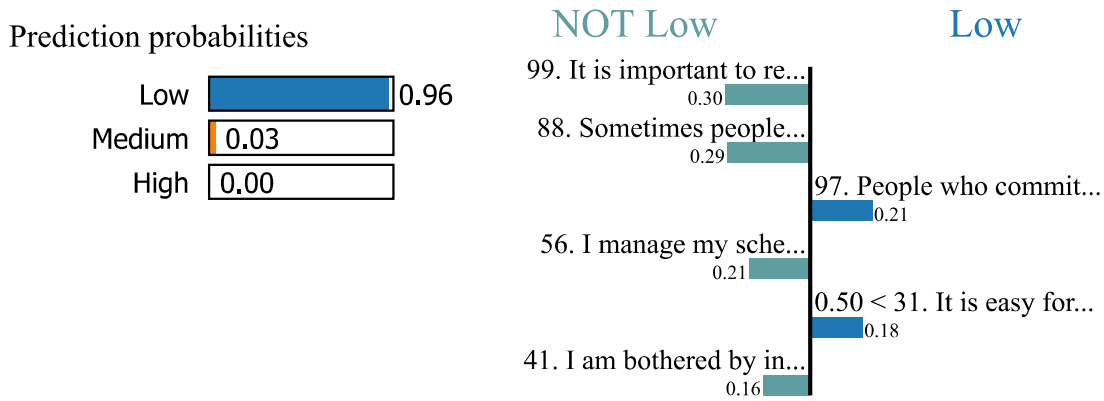


Feature	Value
37. I'm not too worried about what might happen in a few weeks	0.00
53. I'm bothered by people walking slowly in front of me	0.00
73. I have difficulty making elaborate plans	0.00
17. I feel guilty about the decisions that I have made	1.00
41. I am bothered by insignificant things that were not important before	1.00
33. I have trouble remembering things easily	1.00

The last individual present in the test data is explained.

In [178]:
```
exp = explainer.explain_instance(X_test.to_numpy()[-1], mlp.predict_proba, num_fe
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```



Feature	Value
99. It is important to respect turns	0.00
88. Sometimes people need physical strength to understand	0.00
97. People who commit crimes have their reasons for doing it	0.00
56. I manage my schedule as I always have	0.00
31. It is easy for me to choose a photograph of myself to show it on social media	1.00
41. I am bothered by insignificant things that were not important before	0.00

Explanation of the model using SHAP

SHAP explanatory values are displayed according to the class to which the selected individual belongs according to the prediction of the model.

The first individual present in the test data is explained.

In [179]:
```
## Individual Selection.
individual = X_test.iloc[[0]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The second individual present in the data test is explained.

In [180]:
```
## Individual Selection.
individual = X_test.iloc[[1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The last individual present in the data test is explained.

In [181]:
```
## Individual Selection.
individual = X_test.iloc[[-1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

Explanation using heatmaps

The importance of the data present in the neural network is shown from explanation methods.

In [182]:

```
## Items currently in use are retrieved.
ds_item = ds[ds.Status == "In use"].reset_index(drop=True)

## The data is prepared for the generation of heatmaps.
column_Map = ds_item['Question'].tolist()
content_Map = [False for x in range(1, df.shape[1])]

## The next items to be deleted will be marked.
for index in range(0, len(content_Map)):

    if iteration == 0:
        for delete in list_contr:
            if column_Map[index] == delete:
                content_Map [index] = True
    else:
        for delete in question_anomaly:
            if column_Map[index] == delete:
                content_Map [index] = True

## The dataframe is created with the prepared data.
df_mask = pd.DataFrame([content_Map], columns = column_Map)
df_mask_imp = pd.concat([df_mask, df_mask, df_mask], ignore_index = True, axis =
df_mask_all = pd.concat([df_mask, df_mask, df_mask, df_mask], ignore_index = True

df_mask
```

Out[182]:

	1	2	6	7	8	10	13	16	17	18	...	79	80	81	82
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False

1 rows × 51 columns

Identification of feature variables

Identification of feature variables from the randomness of the answers as items of the data.

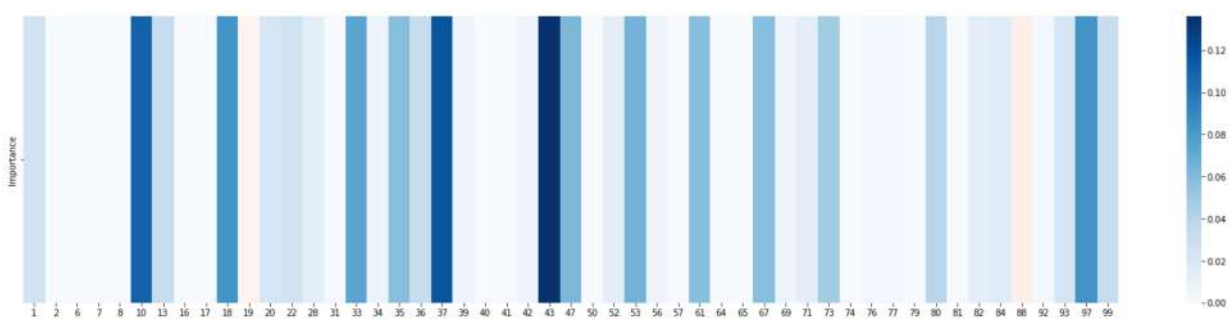
In [183]:
```
## Loading the data for explainability.
explainer = ClassifierExplainer(mlp, X_test, y_test)

## Class index local variable.
iterable_class = -1
map_class = []
```

```
WARNING: Parameter shap='guess', but failed to guess the type of shap explainer
to use for MLPClassifier. Defaulting to the model agnostic shap.KernelExplainer
(shap='kernel'). However this will be slow, so if your model is compatible with
e.g. shap.TreeExplainer or shap.LinearExplainer then pass shap='tree' or shap
='linear'!
WARNING: For shap='kernel', shap interaction values can unfortunately not be ca
lculated!
Note: shap values for shap='kernel' normally get calculated against X_backgroun
d, but paramater X_background=None, so setting X_background=shap.sample(X, 5
0)...
Generating self.shap_explainer = shap.KernelExplainer(model, X, link='identit
y')
```

Explainability of the data belonging to the first class, in this case the Low.

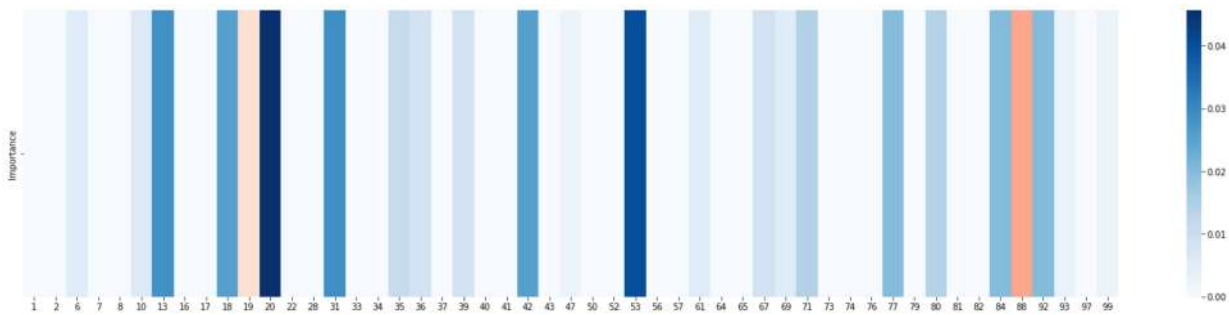
In [184]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

```
Calculating permutation importances (if slow, try setting n_jobs parameter)...
```



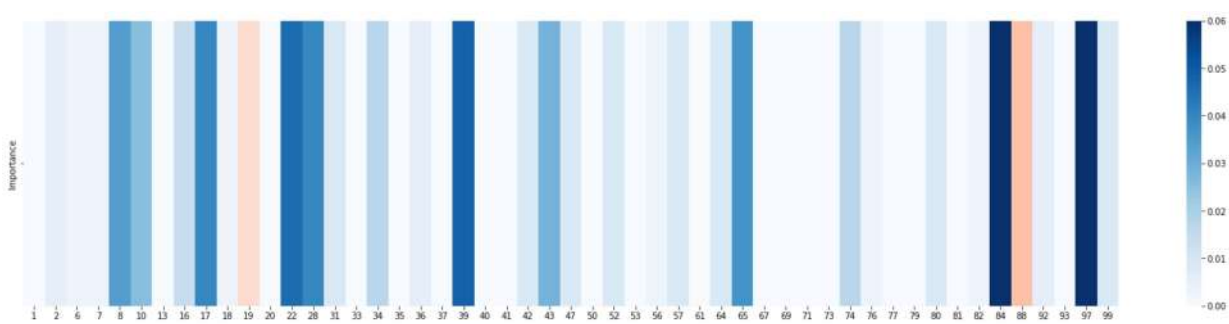
Explainability of the data belonging to the second class, in this case the Medium.

In [185]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



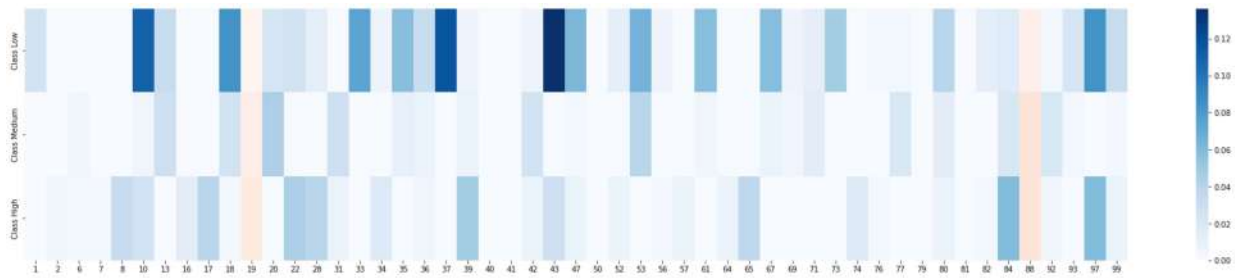
Explainability of the data belonging to the third class, in this case the High.

In [186]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [187]:
```
## Fusion of the previous heatmaps.
df_importance = pd.concat(map_class, axis=1)
df_importance.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_importance.transpose(), cmap = "Reds", cbar = False)
sns.heatmap(df_importance.transpose(), cmap = "Blues",yticklabels = True, xtickla
plt.show()
```



Identification of feature variables using SHAP

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of SHAP.

In [188]:
```
## Class index local variable.
iterable_class = -1
map_class = []
```

Individual explainability of the data belonging to the first class, in this case the Low.

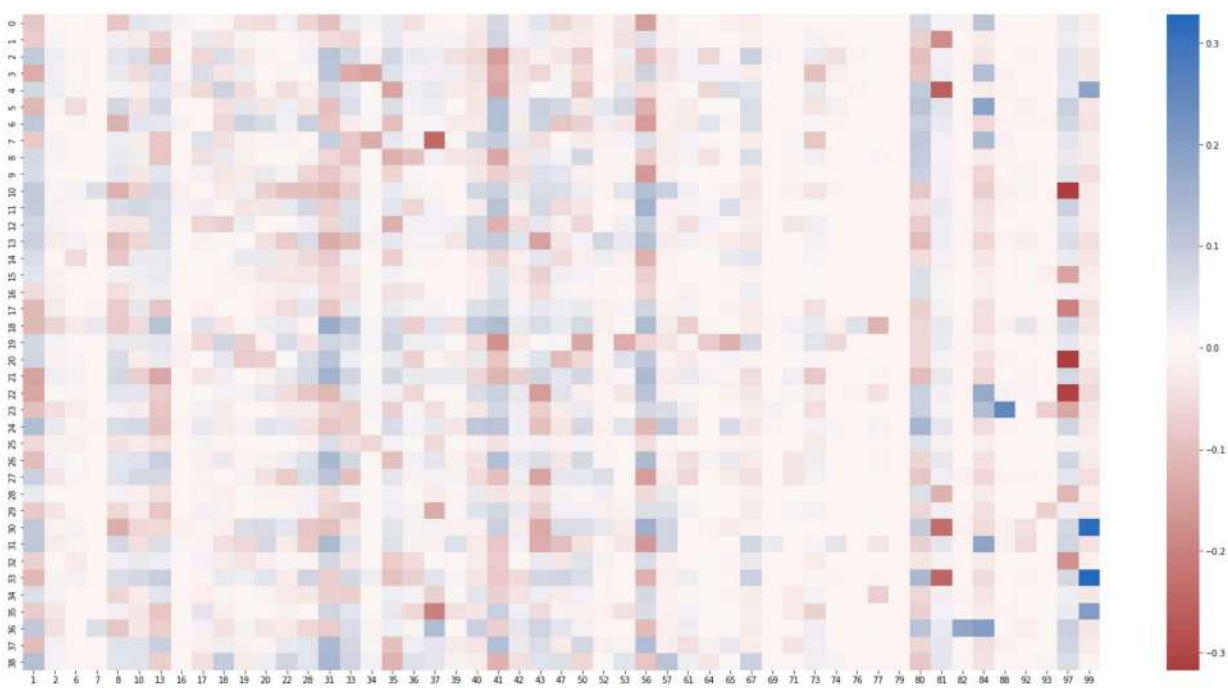
Definition of explainability data.

In [189]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

```
Calculating shap values...

  0%|          | 0/39 [00:00<?, ?it/s]
```

Generation of the heatmap.

In [190]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

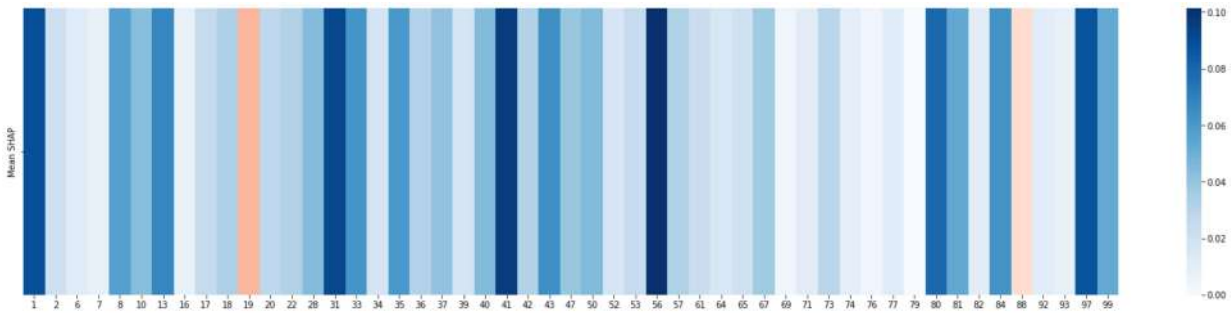


Fusion of the data used.

In [191]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [192]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



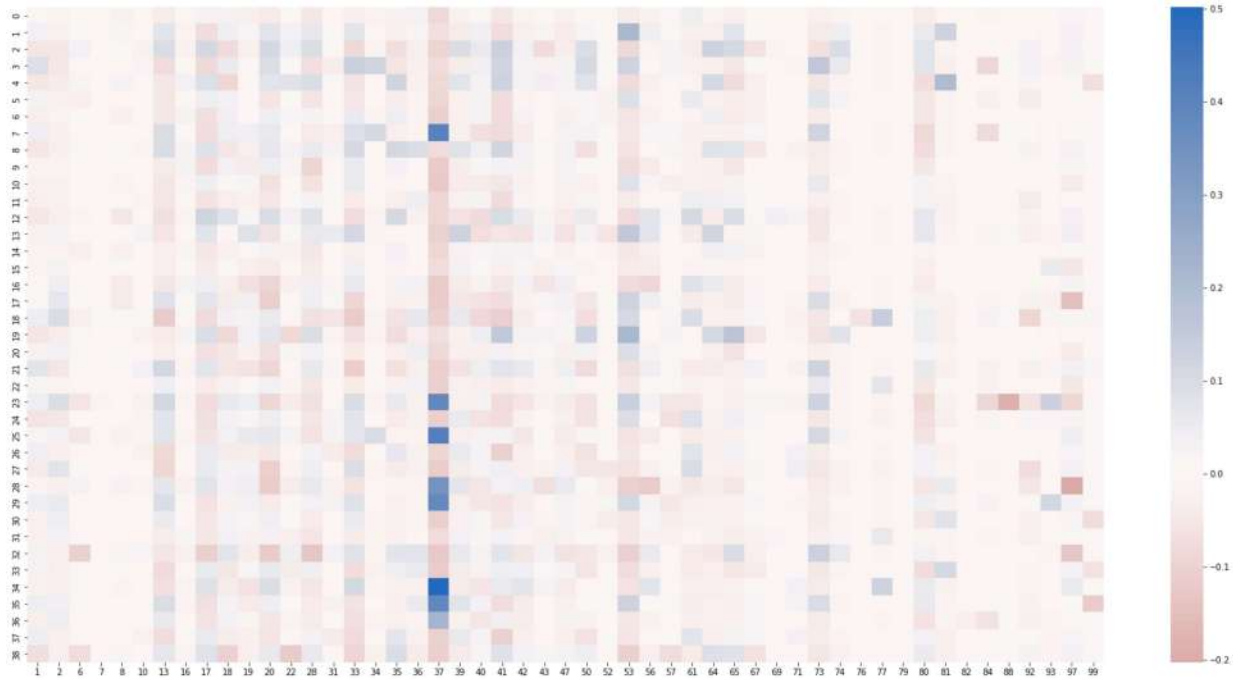
Individual explainability of the data belonging to the second class, in this case the Medium.

Definition of explainability data.

In [193]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [194]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

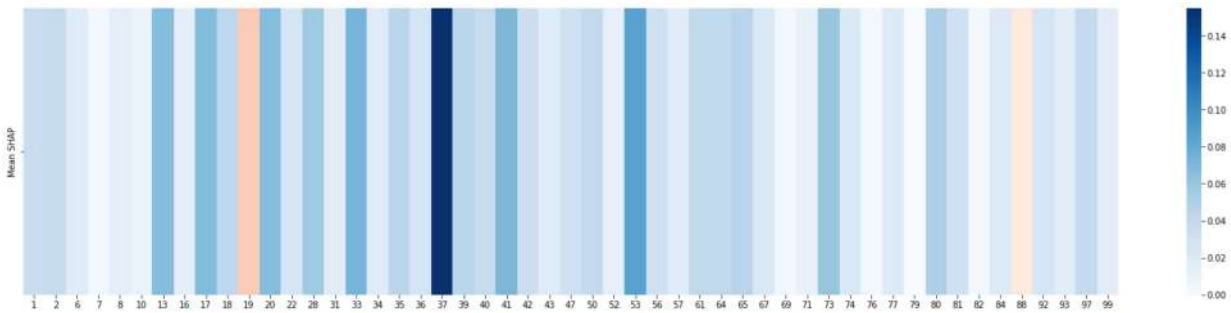


Fusion of the data used.

In [195]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [196]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



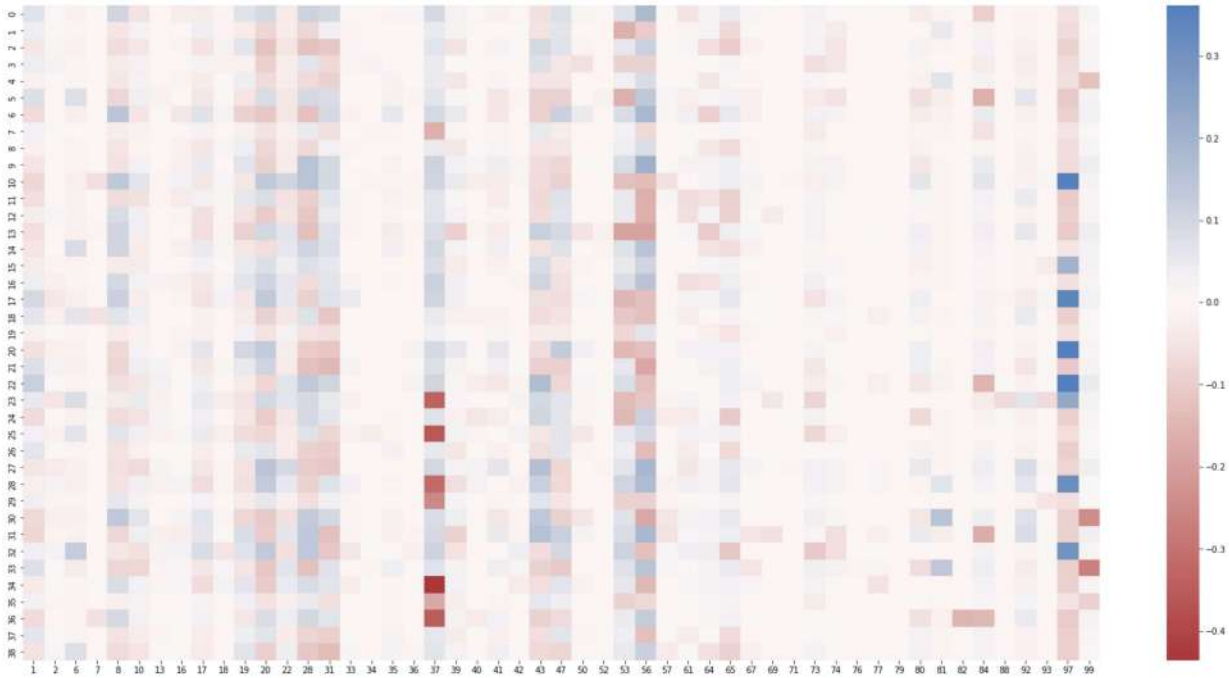
Individual explainability of data belonging to the third class, in this case High.

Definition of explainability data.

In [197]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [198]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

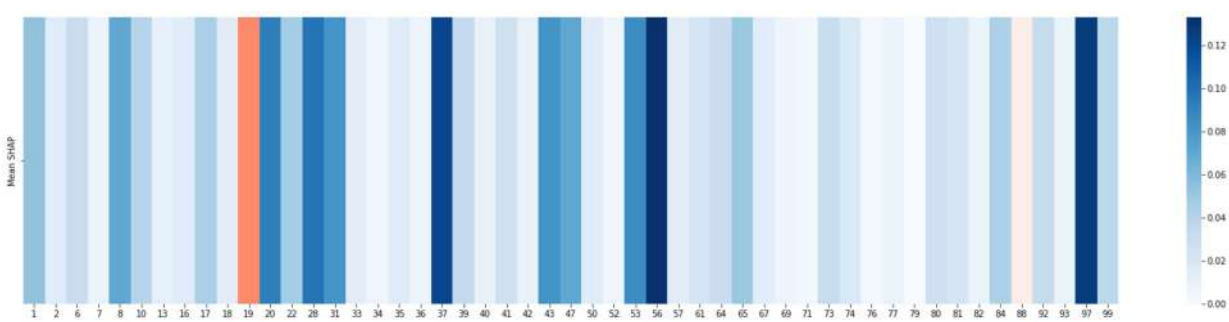


Fusion of the data used.

In [199]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

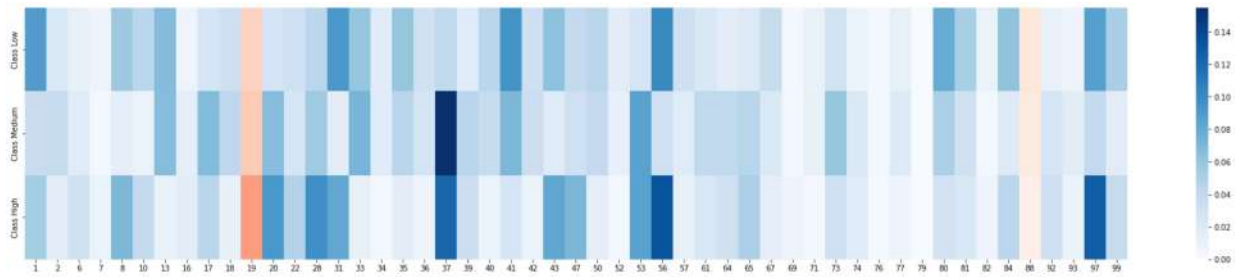
Generation of the heatmap.

In [200]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [201]:
```
## Fusion of the previous heatmaps.
df_shap = pd.concat(map_class, axis=1)
df_shap.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_shap.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_shap.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using LIME

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of LIME.

In [202]:
```
## Data preparation for explainability.
lime_exp = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorical
                                                  feature_names=train_cols, class
                                                  discretize_continuous=True)

## Class index local variable.
iterable_class = -1
map_class = []
```

Obtaining the data for the generation of the explanations by LIME about the Low, Medium and High classes.

In [203]:
```
## Local variables for LIME array data.
exp_matrix = [[] for _ in range(len(name_class))]

## Recovering the data needed for LIME.
for x in X_test.to_numpy():

    ## Auxiliary temporary variables.
    exp_list = [[] for _ in range(len(name_class))]

    ## Loading the data for explainability.
    exp = lime_exp.explain_instance(x, mlp.predict_proba, num_features = df.shape

    ## Recovering the data.
    for elements in range(0, len(name_class)):
        temp = exp.as_map()[elements]
        temp.sort(key=itemgetter(0))

        ## Saving data of the Low, Medium and High classes.
        for tup in temp:
            exp_list[elements].append(tup[1])

        exp_matrix[elements].append(exp_list[elements])
```

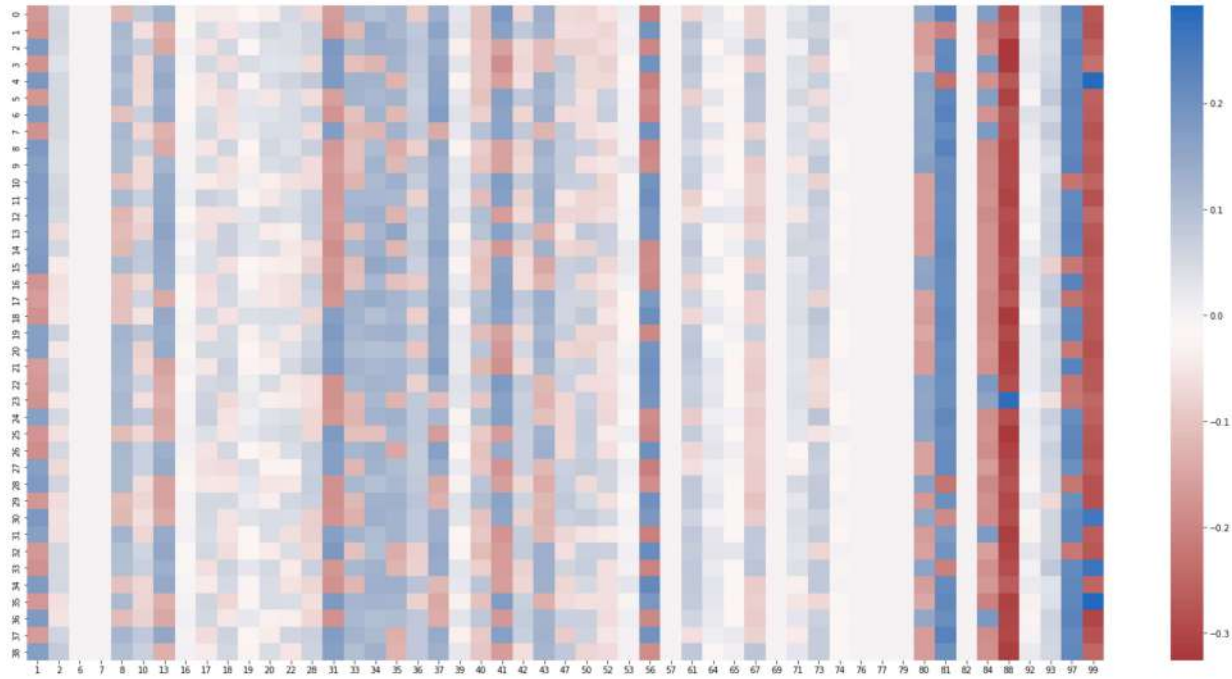
```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

Individual explainability of the data belonging to the first class, in this case the Low.

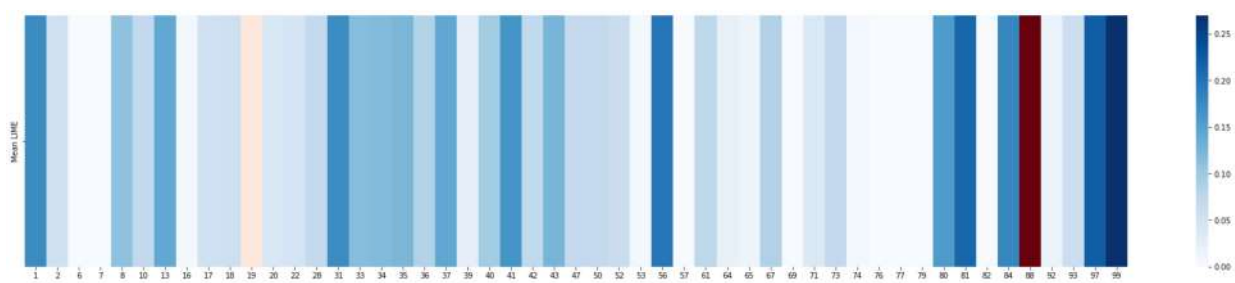
Generation of the heatmap.

In [204]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

In [205]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

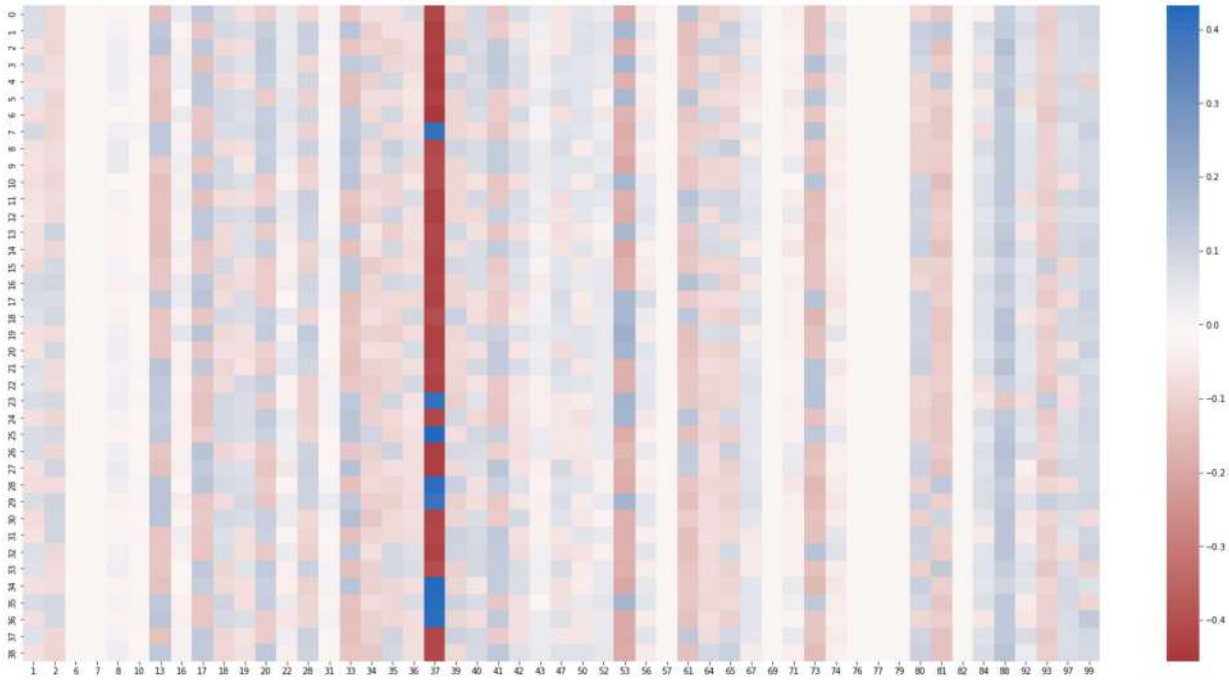


Individual explainability of the data belonging to the second class, in

this case the Medium.

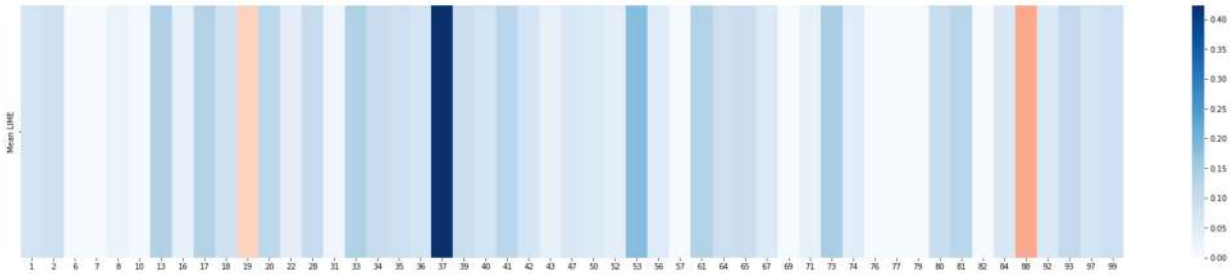
Generation of the heatmap.

In [206]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

In [207]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

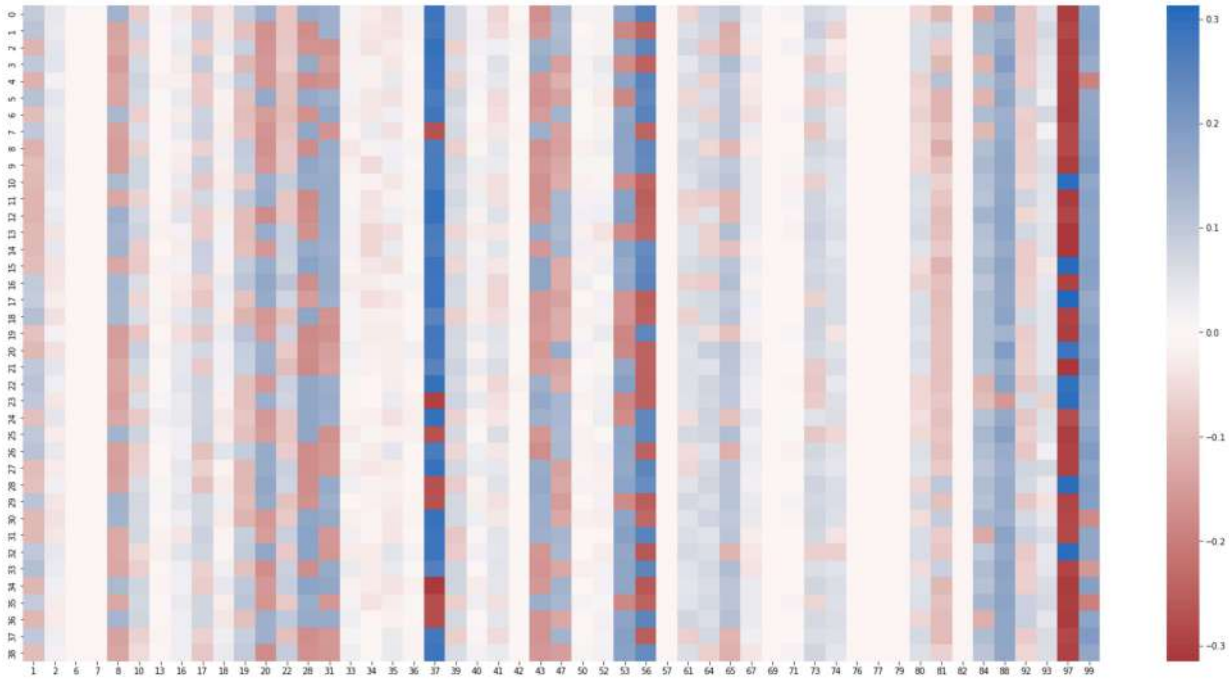


Individual explainability of data belonging to the third class, in this

case High.

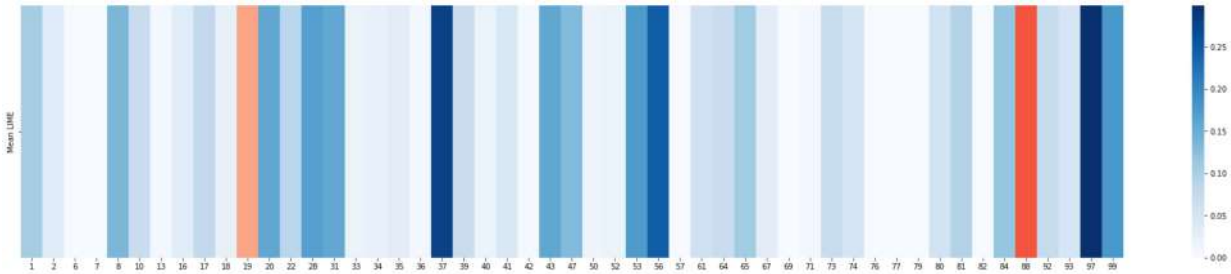
Generation of the heatmap.

In [208]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



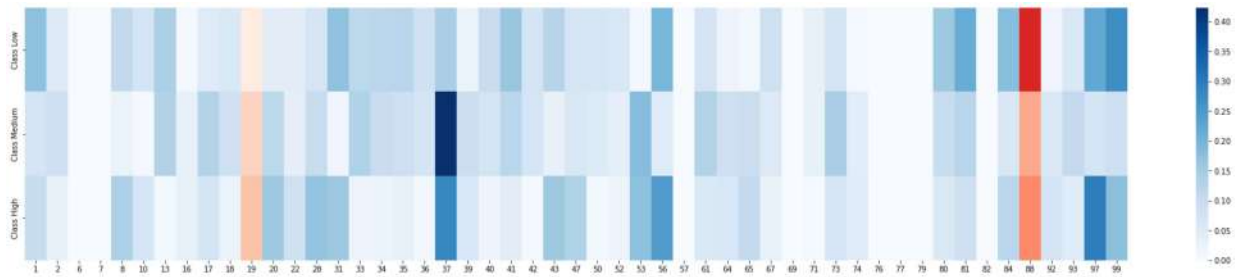
Fusion of the data used.

In [209]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [210]:
```
df_lime = pd.concat(map_class, axis=1)
df_lime.columns = name_class
plt.figure(figsize = (32,6))
sns.heatmap(df_lime.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_lime.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using ALE

It shows the main effect of each question compared to the probability that the model predicts in each class. This effect is shown for both responses. Since there are only 2 answers, the map is the same but with the colors reversed.

In [211]:
```
## Definition of explainability data.
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names=train_cols, target_names= target_r
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())

## Class index local variable.
iterable_class = -1
map_class = []
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

Obtaining the data for the generation of the explanations by ALE about all the classes.

In [212]:
```
## Local variables for ALE array data.
ale_list = [[] for _ in range(len(name_class))]

## Recovering the data needed for ALE.
for array in proba_exp_lr.ale_values:

    ## Recovering the data.
    for elements in range(0, len(name_class)):

        ## Saving data of all the classes.
        ale_list[elements].append(array[0][elements])
```

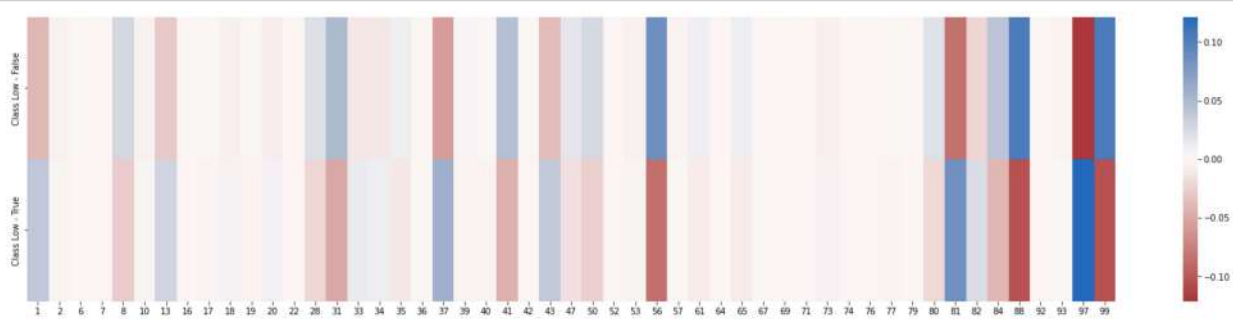
Data processing for the explainability of ALE.

In [213]:
```
## Data processing of the Low, Medium and High classes.
for elements in range(0, len(name_class)):
    ale_df = pd.DataFrame([ale_list [elements]])
    ale_df = pd.concat([ale_df.multiply(-1), ale_df])
    ale_df.index = [name_class [elements] + " - False", name_class [elements] + '
    ale_df.columns = column_Map
    map_class.append(ale_df)
```

Explainability of the data belonging to the first class, in this case the Low.

Generation of the heatmap.

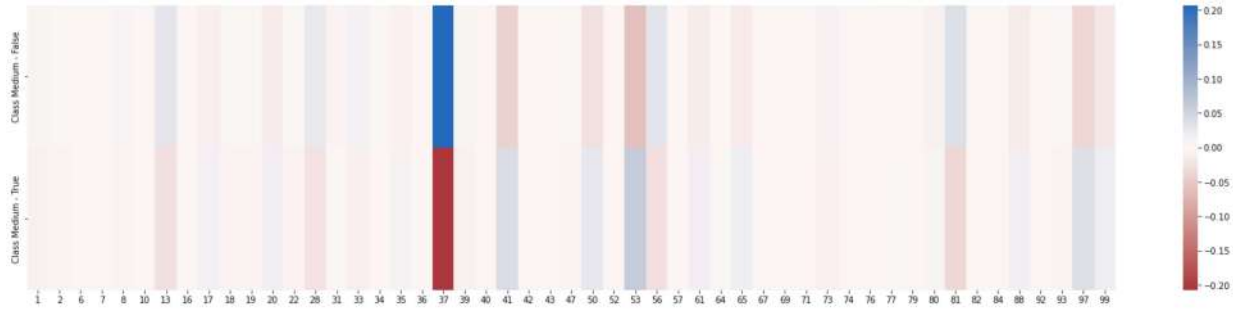
In [214]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the second class, in this case the Medium.

Generation of the heatmap.

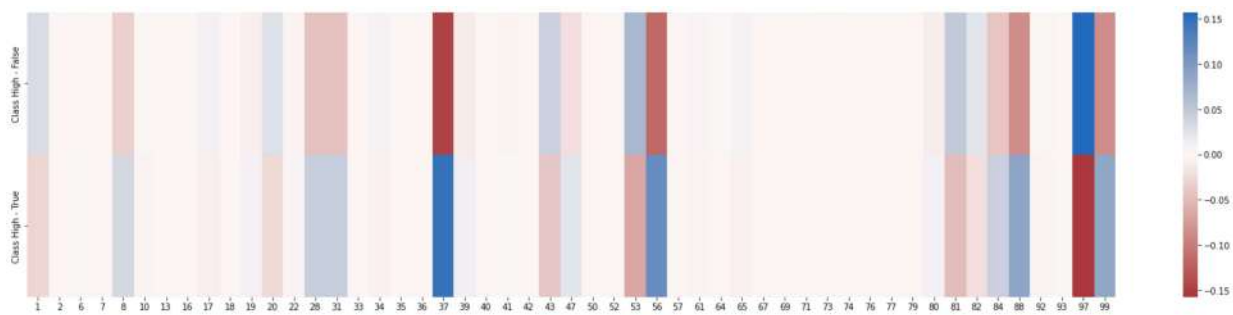
In [215]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the third class, in this case the High.

Generation of the heatmap.

In [216]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```

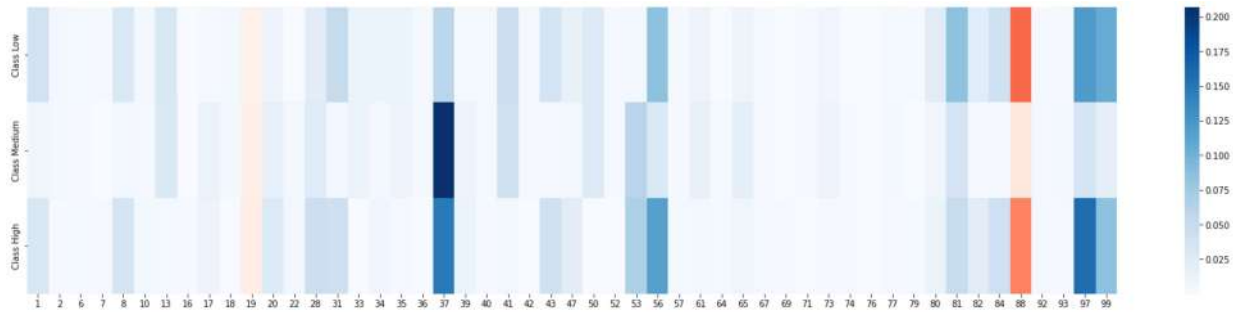


Fusion of the data used in all classes.

In [217]:
```
## Application of absolute value to data.
for index in range(0, len(map_class)):
    map_class [index] = map_class [index][0:1].abs()

## Union of the heatmaps of the Low, Medium and High class.
ale_df = pd.concat(map_class)
ale_df.index = name_class
ale_df = ale_df.transpose()
```

Generation of the heatmap.

In [218]:
```
plt.figure(figsize = (28,6))
sns.heatmap(ale_df.transpose(), cmap="Reds", cbar = False)
sns.heatmap(ale_df.transpose(), cmap="Blues", yticklabels = True, xticklabels = T
plt.show()
```



Comparing of Feature Variables Methods for Explainability

The global values of each of the methods used are taken, their maximums are obtained (the minimums are 0) and a percentage is calculated based on said maximum, which indicates how relevant the contribution is for each question.

In [219]:
```
## Class index local variable.
iterable_class = -1
```

Comparison of the explanation methods of the first class, in this case the Low.

The highest data of each method used is obtained.

In [220]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [221]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

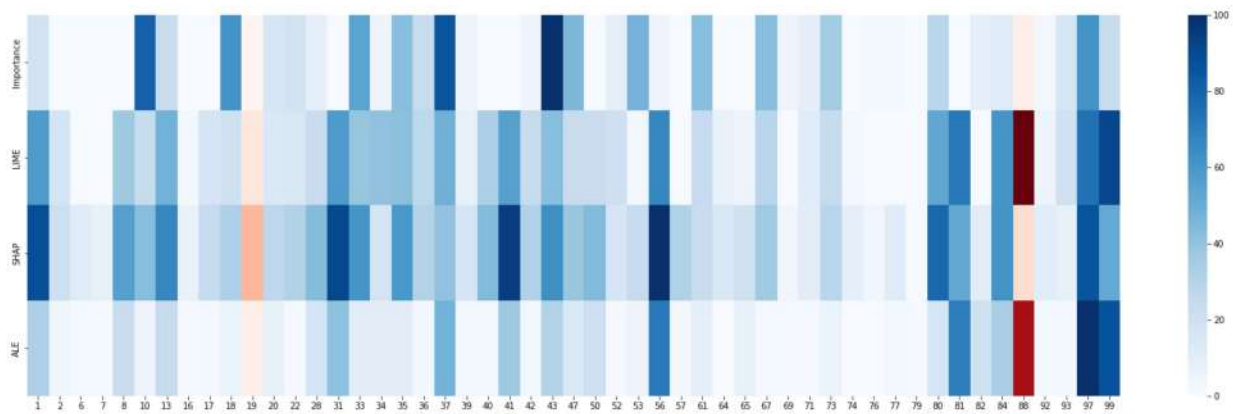
Out[221]:

	1	2	6	7	8	10	13	16
Importance	19.047619	0.000000	0.000000	0.000000	0.000000	80.952381	23.809524	0.000000
LIME	58.928182	18.140766	0.000000	0.000000	37.665460	24.024589	47.699028	2.301396
SHAP	88.176743	20.789058	12.166371	7.704375	56.716372	42.814435	66.432594	7.113930
ALE	32.136914	3.933515	1.920797	1.050437	23.105828	4.598129	24.945188	0.394137

4 rows × 51 columns

Generation of the heatmap.

In [222]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the second class, in this case the Medium.

The highest data of each method used is obtained.

In [223]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [224]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

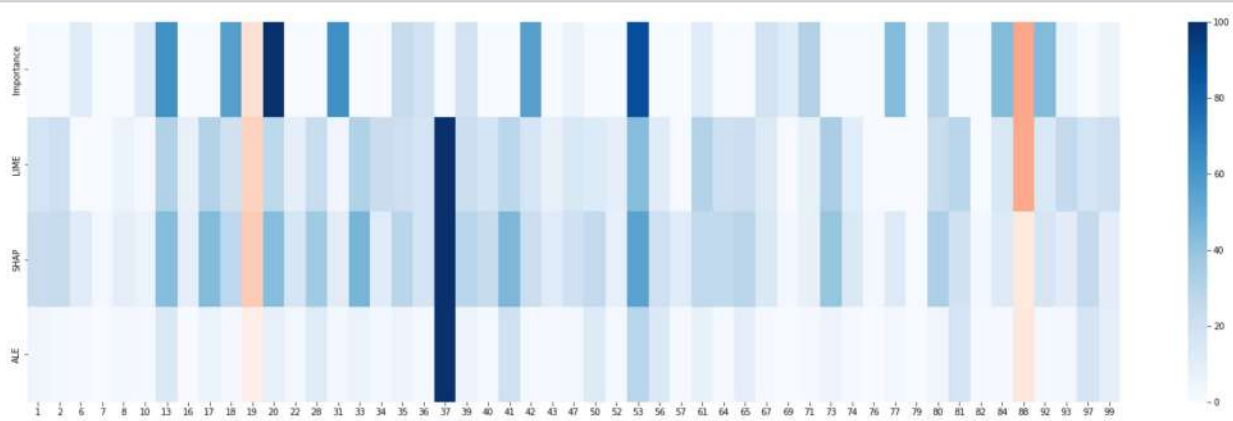
Out[224]:

	1	2	6	7	8	10	13	16
Importance	0.000000	0.000000	12.500000	0.000000	0.000000	12.500000	62.500000	0.000000
LIME	16.830278	20.856305	0.000000	0.000000	6.177092	1.268345	31.531206	7.877039
SHAP	22.916508	23.876595	12.106173	2.012966	9.104939	6.295548	42.973811	9.276256
ALE	4.128651	2.997056	0.499666	0.042934	3.331763	0.445826	13.610085	0.370586

4 rows × 51 columns

Generation of the heatmap.

In [225]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the third class, in this case the High.

The highest data of each method used is obtained.

In [226]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [227]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

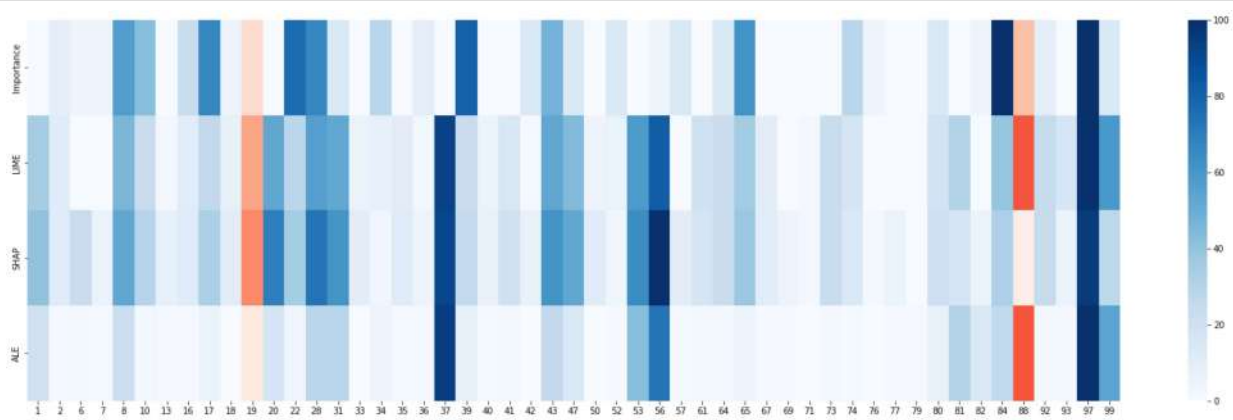
Out[227]:

	1	2	6	7	8	10	13	16
Importance	0.000000	9.523810	4.761905	4.761905	57.142857	42.857143	0.000000	23.809524
LIME	34.524199	11.553471	0.000000	0.000000	46.030780	23.179329	2.971741	10.961234
SHAP	40.537234	11.923751	22.375323	6.441160	53.145030	29.687697	8.297543	12.287572
ALE	19.339387	0.909114	2.137344	0.866025	22.188443	2.957587	1.330063	0.791027

4 rows × 51 columns

Generation of the heatmap.

In [228]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Safeguarding the items eliminated during this iteration.

In [229]:
```
## Removal of anomalous items selected by selection_mode.
for x in question_anomaly:

    ## Status change of the deleted items in the respective auxiliary dataset.
    ds.loc[x - 1,'Status'] = 'Delete'

## Save the changed statuses in the dataset.
ds.to_csv(path_dataset_qs, index = False)
```

Iteration 3

The first iteration of improvement of the model is carried out.

Preparing the data for the neural network

The data is prepared in order to be processed by the neural network.

In [230]:

```
## The iteration number is increased.
iteration += 1

## Loading data from the dataset.
df = pd.read_csv(path_dataset)
ds = pd.read_csv(path_dataset_qs)

## Elimination of unrelated items from the dataset.
if len(unrelated_questions) != 0:
    df = df.drop(df.columns[unrelated_questions[0]:unrelated_questions[1]], axis=

i = 0
if iteration == 1:
    ## Elimination of list of conflicting items declared in global variables.
    for x in list_contr:
        df.drop(df.columns[x - (i + 1)], axis = 1, inplace = True)
        i += 1
        ## Status change of the deleted question in the respective auxiliary data
        ds.loc[x - 1,'Status'] = 'Delete'

else:
    ## Elimination of the list of anomadic items defined in the improvement proce
    ds_delete = ds[ds.Status == "Delete"].reset_index(drop=True)
    for x in ds_delete['Question'].tolist():
        df.drop(df.columns[x - (i + 1)], axis = 1, inplace = True)
        i += 1

## Grouping of the target according to the defined dictionary.
if grouping_name != None and grouping_name != '':
    df[grouping_name] = df[grouping_name].map(class_dic)

## Assignment of local variables according to the data necessary for the neural r
train_cols = df.columns [0:-1]
label = df.columns [-1]
X = df [train_cols]
y = df [label]

## The prepared data of the dataset is printed.
df
```

Out[230]:

	1. Most of the time I have difficulty concentrating on simple tasks	2. I don't feel like doing my daily duties	6. I consider that my life is full of good things	7. My hobbies are still important to me	8. I'm still as punctual as I have always been	10. I have found that I can spend a lot of time scrolling the screen \nof my cell phone without searching or stopping at anything in particular	13. I felt more encouraged to do my daily activities before	16. I think my thoughts are strange or different from before	1 ab de
0	1	1	1	1	1	0	0	0	
1	1	0	1	1	1	0	1	1	
2	0	0	1	1	0	1	0	1	
3	0	0	1	1	1	0	1	0	
4	1	1	1	0	1	1	0	1	
...	...	...	...	...	...	...	...	...	
99	1	1	1	1	1	1	1	0	
100	1	1	1	1	0	0	1	0	
101	0	0	1	1	0	1	1	1	
102	1	0	1	1	1	1	0	0	
103	1	1	1	1	0	1	0	0	

104 rows × 50 columns

Oversample application

An oversample process is applied to level the amount of existing data given by the grouping.

In [231]:
```
## The number of elements present in the grouped target is printed.
y.value_counts()
```

Out[231]:
```
0    39
1    35
2    30
Name: Target, dtype: int64
```

Oversample application.

In [232]:
```
## Application of the oversample to the prepared data.
X, y = oversample.fit_resample(X, y)
```

Printing of the data after the application of the oversample.

In [233]:
```
## The results obtained after applying the oversample are printed.
y.value_counts()
```

Out[233]:
```
0    39
1    39
2    39
Name: Target, dtype: int64
```

Preparation of the parameters of the neural network

We proceed to define the parameters necessary for the training of the neural network.

In [234]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
```

Neural network training

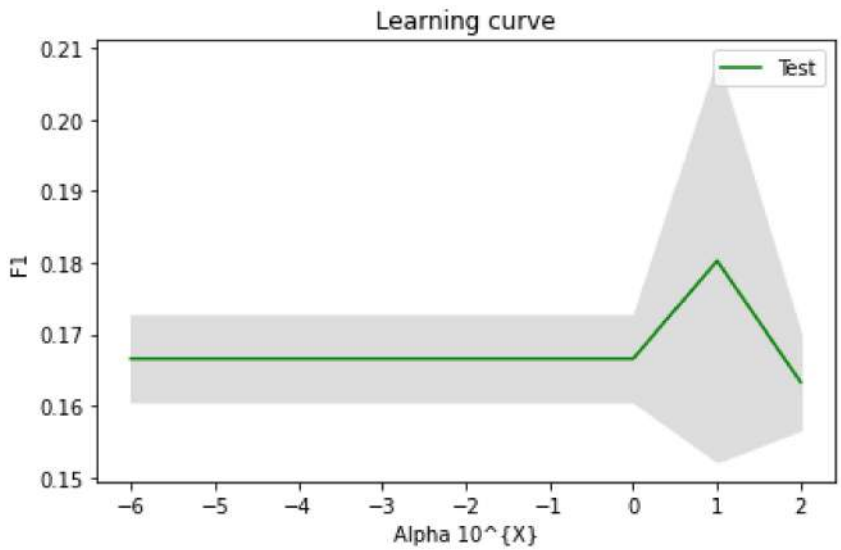
The alpha parameters and the initial learning rate are adjusted to obtain the best performance with the model using cross validation.

In [235]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_normal

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = 0.0001, max_iter =

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[235]:
```
([0.16329749103942653,
  0.18026037160631153,
  0.16659498207885304,
  0.16659498207885304,
  0.16659498207885304,
  0.16659498207885304,
  0.16659498207885304,
  0.16659498207885304,
  0.16659498207885304],
 [0.006619092595549127,
  0.028021044395542195,
  0.006020651927912844,
  0.006020651927912844,
  0.006020651927912844,
  0.006020651927912844,
  0.006020651927912844,
  0.006020651927912844,
  0.006020651927912844])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [236]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Alpha 10^{X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



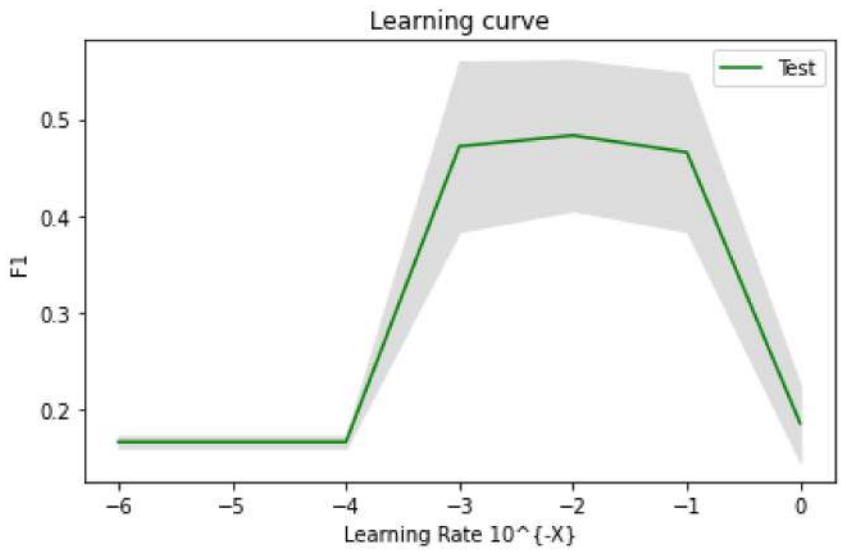
The training is performed again keeping the same parameters except for the alpha value that changes to 1.

In [237]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_alpha

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = regul_param, max_it

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[237]:
```
([0.18578597030209934,
  0.4657540716364246,
  0.4830854439677969,
  0.4718491009048285,
  0.16659498207885304,
  0.16659498207885304,
  0.16659498207885304],
 [0.03912815731541059,
  0.08131191266979419,
  0.07748179685218214,
  0.08743387745396868,
  0.006020651927912844,
  0.006020651927912844,
  0.006020651927912844])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [238]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Learning Rate 10^{-X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



Generation of the neural network model

The parameters of the final model are modified according to the data obtained in the training of the neural network.

In [239]:
```
## The value of the alpha parameter, learning_rate_init and random_state are char
## to the data in the global variables.
mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solver =
                    learning_rate = 'constant', learning_rate_init = lear
                    random_state = random_state_model)

## We print the data of the final generated model.
mlp.fit(X_train, y_train)
```

Out[239]:
```
MLPClassifier
MLPClassifier(activation='logistic', alpha=0.001, hidden_layer_sizes=(10,),
              learning_rate_init=0.1, max_iter=100000, random_state=42)
```

We safeguard the final generated model

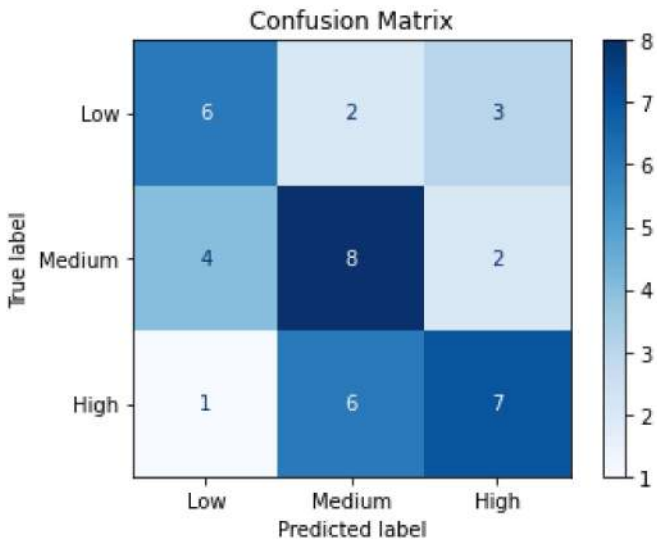
In [240]:
```
joblib.dump(mlp,"model_depression_i" + str(iteration) + ".pkl")
```

Out[240]:
```
['model_depression_i3.pkl']
```

Generation of the confusion matrix of the model

We obtain the statistics of the final model to generate its confusion matrix.

In [241]:
```
## The data for the generation of the confession matrix are defined.
confusion_matrix = ConfusionMatrixDisplay.from_estimator(mlp, X_test, y_test, dis
                                                         cmap = plt.cm.Blues)
confusion_matrix.ax_.set_title("Confusion Matrix")
plt.show()
```



We get the detailed statistics of the final model.

In [242]:
```
## The necessary detailed prediction is generated.
y_pred = mlp.predict(X_test)

## The detailed statistics of the model are printed.
print('Classification accuracy =',accuracy_score(y_test,y_pred) * 100,'%\n')
print(classification_report(y_test,y_pred))
```

```
Classification accuracy = 53.84615384615385 %

              precision    recall  f1-score   support

           0       0.55      0.55      0.55        11
           1       0.50      0.57      0.53        14
           2       0.58      0.50      0.54        14

    accuracy                           0.54        39
   macro avg       0.54      0.54      0.54        39
weighted avg       0.54      0.54      0.54        39
```

The data obtained is saved for later graphing.

In [243]:
```
accuracy = accuracy_score(y_test,y_pred) * 100
clf_rep = precision_recall_fscore_support(y_test,y_pred)

## The number of items used in the iteration is retrieved.
ds_delete = ds[ds.Status == "Delete"]
ds_delete.reset_index(inplace=True, drop=True)

## All the metrics of the confusion matrix are obtained.
metrics = [",".join(map(str, ds_delete['Question'].tolist())),
           accuracy]

for index in range(0, len(name_class)):
    metrics.append(clf_rep[0][index])
    metrics.append(clf_rep[1][index])
    metrics.append(clf_rep[2][index])
    metrics.append(clf_rep[3][index])

## The names of the columns of the dataset are defined.
columns = ['Question', 'Acurracy global']

for index in range(0, len(name_class)):
    columns.append('Precision_' + str(index))
    columns.append('Recall_' + str(index))
    columns.append('F1_score_' + str(index))
    columns.append('Support_' + str(index))

## A new row of the dataset is generated with all the data.
data_metrics = pd.DataFrame([metrics], columns = columns)

## The data is saved to the dataset.
data_metrics.to_csv(path_dataset_qd, mode = 'a', header = False, index = False)

## The data added to the dataset is printed.
data_metrics
```

Out[243]:

	Question	Acurracy global	Precision_0	Recall_0	F1_score_0	Supp
0	3,4,5,9,11,12,14,15,19,21,23,24,25,26,27,29,30...	53.846154	0.545455	0.545455	0.545455	

Improvement process of the neural network model.

The application of the explanation method by ALE allows to determine the behavior of the neural network and thereby improve it by discarding non-significant data for it.

Slope Analysis of ALE Plots

In [244]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

We proceed to obtain the graphs to obtain the slopes of the selected classes.

In [245]:
```
## The question metrics dataset is loaded.
dm = pd.DataFrame()

## The number of items used in the iteration is retrieved.
ds_item = ds[ds.Status == "In use"]
ds_item.reset_index(inplace=True, drop=True)

## We get the slopes from the ALE data.
for i in range(df.shape[1]-1):
    structure_data = {'Question': ds_item.loc[i, "Question"]}

    for index in selection_class:
        slope = proba_exp_lr.ale_values [i][1][index] - proba_exp_lr.ale_values [
        structure_data['Slope ' + str(target_names[index])] = slope
        structure_data['Threshold ' + str(target_names[index])] = 'NA'
        structure_data['Anomaly ' + str(target_names[index])] = 'NA'

    dm = dm.append(structure_data, ignore_index=True)

## The current rating data dataset is printed.
dm
```

```
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
```

Slope analysis and threshold application

The slope obtained is analyzed and those that are above or below the positive and negative threshold are selected.

In [246]:

```
for i in range(df.shape[1]-1):
    for index in selection_class:

        ## The slopes are selected according to the values of the thresholds of t
        if dm.loc[i, "Slope " + str(target_names[index])] >= positive_threshold:
            dm.loc[i, "Threshold " + str(target_names[index])] = 1
        elif dm.loc[i, "Slope " + str(target_names[index])] <= negative_threshold
            dm.loc[i, "Threshold " + str(target_names[index])] = 0

## The current rating data dataset is printed.
dm
```

Out[246]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	1	-0.006614	NA	NA	-0.000607	NA	NA
1	2	-0.018055	0	NA	-0.008497	NA	NA
2	6	-0.001865	NA	NA	-0.007545	NA	NA
3	7	-0.017367	0	NA	0.018536	1	NA
4	8	0.173014	1	NA	-0.225715	0	NA
5	10	-0.002792	NA	NA	0.012527	1	NA
6	13	-0.047196	0	NA	0.017650	1	NA
7	16	0.007778	NA	NA	-0.004252	NA	NA
8	17	0.002405	NA	NA	0.028202	1	NA
9	18	-0.088971	0	NA	-0.010765	0	NA
10	20	-0.083307	0	NA	0.212041	1	NA
11	22	0.000669	NA	NA	-0.000814	NA	NA
12	28	0.075218	1	NA	-0.163893	0	NA
13	31	0.005803	NA	NA	-0.007938	NA	NA
14	33	-0.014158	0	NA	-0.002098	NA	NA
15	34	-0.215667	0	NA	0.178945	1	NA
16	35	0.005691	NA	NA	-0.002125	NA	NA
17	36	0.000568	NA	NA	0.000059	NA	NA
18	37	-0.079085	0	NA	-0.312803	0	NA
19	39	-0.000589	NA	NA	-0.002836	NA	NA
20	40	0.005258	NA	NA	0.000021	NA	NA
21	41	0.078769	1	NA	-0.018348	0	NA
22	42	0.025467	1	NA	-0.016639	0	NA
23	43	-0.025617	0	NA	0.030172	1	NA
24	47	0.157951	1	NA	-0.243299	0	NA
25	50	0.004235	NA	NA	0.000484	NA	NA
26	52	-0.094860	0	NA	0.006630	NA	NA

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
27	53	-0.010567	0	NA	0.080953	1	NA
28	56	0.087371	1	NA	-0.103008	0	NA
29	57	-0.012926	0	NA	0.003207	NA	NA
30	61	0.048206	1	NA	0.008476	NA	NA
31	64	0.000347	NA	NA	0.000382	NA	NA
32	65	-0.002680	NA	NA	0.001158	NA	NA
33	67	-0.003949	NA	NA	0.002053	NA	NA
34	69	-0.006850	NA	NA	0.008052	NA	NA
35	71	-0.003454	NA	NA	-0.000023	NA	NA
36	73	-0.175165	0	NA	-0.005994	NA	NA
37	74	0.003430	NA	NA	0.013733	1	NA
38	76	-0.002919	NA	NA	0.000090	NA	NA
39	77	0.018459	1	NA	0.026777	1	NA
40	79	0.002294	NA	NA	-0.002907	NA	NA
41	80	0.095816	1	NA	-0.054058	0	NA
42	81	-0.196739	0	NA	0.137296	1	NA
43	82	-0.067398	0	NA	0.065858	1	NA
44	84	0.168791	1	NA	-0.092529	0	NA
45	92	0.000270	NA	NA	0.033853	1	NA
46	93	-0.001158	NA	NA	-0.006862	NA	NA
47	97	-0.279718	0	NA	0.324933	1	NA
48	99	0.248713	1	NA	-0.201410	0	NA

Analysis and determination of anomalous items

The anomalous items present in the model are determined based on their slope and expected response from the expert.

In [247]:

```
## The dataset of expected responses is loaded.
da = pd.read_csv(path_dataset_qa)

## Delete list of items deleted in this iteration.
i = 0
for x in ds_delete['Question']:
    da.drop(da.index[x - (i + 1)], axis = 0, inplace = True)
    i += 1

## Indexes are restored for later use.
da.reset_index(inplace=True, drop=True)

## The dataset of expected responses is printed.
da
```

Out[247]:

	Number	Question	RE
0	1	Most of the time I have difficulty concentrati...	1
1	2	I don't feel like doing my daily duties	1
2	6	I consider that my life is full of good things	0
3	7	My hobbies are still important to me	0
4	8	I'm still as punctual as I have always been	0
5	10	I have found that I can spend a lot of time sc...	1
6	13	I felt more encouraged to do my daily activiti...	1
7	16	I think my thoughts are strange or different f...	1
8	17	I feel guilty about the decisions that I have ...	1
9	18	I don't feel as comfortable with my body as I ...	1
10	20	It is difficult for me to make decisions even ...	1
11	22	It is not difficult for me to understand somet...	0
12	28	The decisions I have made so far have been the...	0
13	31	It is easy for me to choose a photograph of my...	0
14	33	I have trouble remembering things easily	1
15	34	In recent months I have had discussions with m...	1
16	35	I constantly imagine that something will go wr...	1
17	36	I am afraid of being worng when doing my homework	1
18	37	I'm not too worried about what might happen in...	0
19	39	Everything will be alright	0
20	40	I can easily blank my mind	0
21	41	I am bothered by insignificant things that wer...	1
22	42	I find it uncomfortable to be in a crowded place	1
23	43	Sometimes i feel trapped	1
24	47	I can clearly express to others how I feel	0
25	50	I imagine that at any moment a disaster of the...	1

	Number	Question	RE
26	52	Being locked in an elevator would be the worst...	1
27	53	I'm bothered by people walking slowly in front...	1
28	56	I manage my schedule as I always have	0
29	57	It bothers me to feel that people on the stree...	1
30	61	Sometimes I forget what I wanted to say becaus...	1
31	64	I can put down my cell phone and dedicate myse...	0
32	65	I worry that people will not understand what I...	1
33	67	I get angry easily	1
34	69	It is not important for me to meet set dates	1
35	71	I use lies just to get out of certain problems	1
36	73	I have difficulty making elaborate plans	1
37	74	People have problems because of themselves	1
38	76	Laws are not as important as others think	1
39	77	I would regret betraying a friend	0
40	79	It is easy for me to work in a team	0
41	80	It is important to help people when they need it	0
42	81	I have punched someone or thought of doing it	1
43	82	If it was necessary I would pretend to be some...	1
44	84	After an argument I usually go over what happe...	0
45	92	It does not seem serious to me to have some debts	1
46	93	People steal because they have needs	1
47	97	People who commit crimes have their reasons fo...	1
48	99	It is important to respect turns	0

We proceed to identify the anomadic items present in the model present in this iteration.

In [248]:

```
for i in range(df.shape[1]-1):

    ## Local variable to indicate the index of the selected classes.
    i_index = 0

    for index in selection_class:

        ## Determine if it fails based on the expected response parameters of the
        if dm.loc[i, "Threshold "  + str(target_names[index])] == 1 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 1 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 0 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 0 and da.loc[i,
            dm.loc[i, "Anomaly "  + str(target_names[index])] = expected_response

        ## Increase of the index of the selected classes.
        i_index += 1

## The final grade data dataset is printed.
dm
```

Out[248]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	1	-0.006614	NA	NA	-0.000607	NA	NA
1	2	-0.018055	0	0	-0.008497	NA	NA
2	6	-0.001865	NA	NA	-0.007545	NA	NA
3	7	-0.017367	0	1	0.018536	1	1
4	8	0.173014	1	0	-0.225715	0	0
5	10	-0.002792	NA	NA	0.012527	1	0
6	13	-0.047196	0	0	0.017650	1	0
7	16	0.007778	NA	NA	-0.004252	NA	NA
8	17	0.002405	NA	NA	0.028202	1	0
9	18	-0.088971	0	0	-0.010765	0	1
10	20	-0.083307	0	0	0.212041	1	0
11	22	0.000669	NA	NA	-0.000814	NA	NA
12	28	0.075218	1	0	-0.163893	0	0
13	31	0.005803	NA	NA	-0.007938	NA	NA
14	33	-0.014158	0	0	-0.002098	NA	NA
15	34	-0.215667	0	0	0.178945	1	0
16	35	0.005691	NA	NA	-0.002125	NA	NA
17	36	0.000568	NA	NA	0.000059	NA	NA
18	37	-0.079085	0	1	-0.312803	0	0
19	39	-0.000589	NA	NA	-0.002836	NA	NA

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
20	40	0.005258	NA	NA	0.000021	NA	NA
21	41	0.078769	1	1	-0.018348	0	1
22	42	0.025467	1	1	-0.016639	0	1
23	43	-0.025617	0	0	0.030172	1	0
24	47	0.157951	1	0	-0.243299	0	0
25	50	0.004235	NA	NA	0.000484	NA	NA
26	52	-0.094860	0	0	0.006630	NA	NA
27	53	-0.010567	0	0	0.080953	1	0
28	56	0.087371	1	0	-0.103008	0	0
29	57	-0.012926	0	0	0.003207	NA	NA
30	61	0.048206	1	1	0.008476	NA	NA
31	64	0.000347	NA	NA	0.000382	NA	NA
32	65	-0.002680	NA	NA	0.001158	NA	NA
33	67	-0.003949	NA	NA	0.002053	NA	NA
34	69	-0.006850	NA	NA	0.008052	NA	NA
35	71	-0.003454	NA	NA	-0.000023	NA	NA
36	73	-0.175165	0	0	-0.005994	NA	NA
37	74	0.003430	NA	NA	0.013733	1	0
38	76	-0.002919	NA	NA	0.000090	NA	NA
39	77	0.018459	1	0	0.026777	1	1
40	79	0.002294	NA	NA	-0.002907	NA	NA
41	80	0.095816	1	0	-0.054058	0	0
42	81	-0.196739	0	0	0.137296	1	0
43	82	-0.067398	0	0	0.065858	1	0
44	84	0.168791	1	0	-0.092529	0	0
45	92	0.000270	NA	NA	0.033853	1	0
46	93	-0.001158	NA	NA	-0.006862	NA	NA
47	97	-0.279718	0	0	0.324933	1	0
48	99	0.248713	1	0	-0.201410	0	0

Removing identified anomadic items from the model

We proceed to eliminate the identified anomadic items so as not to use them in the next iteration.

In [249]:
```
## Local variable for the elimination of the anomaly items.
instructions = ''
init = False
question_anomaly = []

if selection_mode == 1:

    ## Items that fail in the defined class are retrieved.
    dm_delete = dm[dm['Anomaly ' + str(target_names[select_class_mode])] == 1]
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 2:

    ## Items that fail in classes defined with the or condition type are retrieve
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 1"
            init = True
        else:
            instructions += " or `Anomaly " + str(target_names[index]) + "` == 1'

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 3:

    ## Items that fail in classes defined with the and condition type are retriev
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 1"
            init = True
        else:
            instructions += " and `Anomaly " + str(target_names[index]) + "` == 1

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 4:

    ## Out-of-threshold items are retrieved in classes defined with the conditior
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 'NA'"
            init = True
        else:
            instructions += " and `Anomaly " + str(target_names[index]) + "` == '

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

## The selected blank items are printed according to the selection_mode.
question_anomaly
```

Out[249]: [7, 18, 41, 42, 77]

Model explainability process

The data used for the test of the neural network model is shown.

In [250]: X_test

Out[250]:

	1. Most of the time I have difficulty concentrating on simple tasks	2. I don't feel like doing my daily duties	6. I consider that my life is full of good things	7. My hobbies are still important to me	8. I'm still as punctual as I have always been	10. I have found that I can spend a lot of time scrolling the screen \nof my cell phone without searching or stopping at anything in particular	13. I felt more encouraged to do my daily activities before	16. I think my thoughts are strange or different from before	17. I gu about decisi tl h m
69	1	0	1	1	0	0	0	0	
46	1	0	1	1	1	1	1	1	
58	0	0	1	1	1	0	1	1	
114	1	0	1	1	1	1	0	0	
73	0	0	1	1	1	1	0	0	
98	1	0	0	1	1	1	0	1	
31	0	0	1	1	0	0	0	0	
53	1	0	1	1	1	1	1	1	
65	0	0	1	1	1	0	1	0	
96	0	0	1	1	1	1	0	0	
95	0	0	1	0	0	1	0	1	
97	0	0	1	1	1	0	0	0	
2	0	0	1	1	0	1	0	1	
62	0	1	1	1	0	1	0	1	
110	0	0	0	1	1	0	0	0	
55	0	1	1	1	1	0	0	1	
103	1	1	1	1	0	1	0	0	
100	1	1	1	1	0	0	1	0	
66	1	1	0	0	0	1	0	1	
44	0	0	1	1	1	0	0	0	
77	0	1	1	1	1	1	0	1	
17	1	0	1	1	1	1	1	1	
81	1	0	1	1	1	0	1	1	
74	1	1	0	1	1	1	1	1	

56	0	0	1	1	1	0	1	1
94	1	1	0	1	0	1	1	1
35	1	0	1	1	1	0	0	1
38	0	1	1	1	1	0	0	1
93	0	0	1	1	1	1	1	1
48	1	1	1	1	0	1	1	1
33	0	1	1	1	0	1	1	1
59	0	1	1	1	1	1	0	0
92	1	0	0	1	1	0	0	1
42	1	0	1	1	1	0	0	1
10	0	0	1	1	0	1	0	1
85	1	1	1	1	1	1	1	1
32	0	1	1	0	0	1	1	1
78	1	0	1	1	1	0	0	1
19	0	0	0	1	1	0	1	1

39 rows × 49 columns

Explanation of the model using ALE

In analysis of the model through the results obtained by ALE, it is possible to identify the behavior of the neural network.

In [251]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

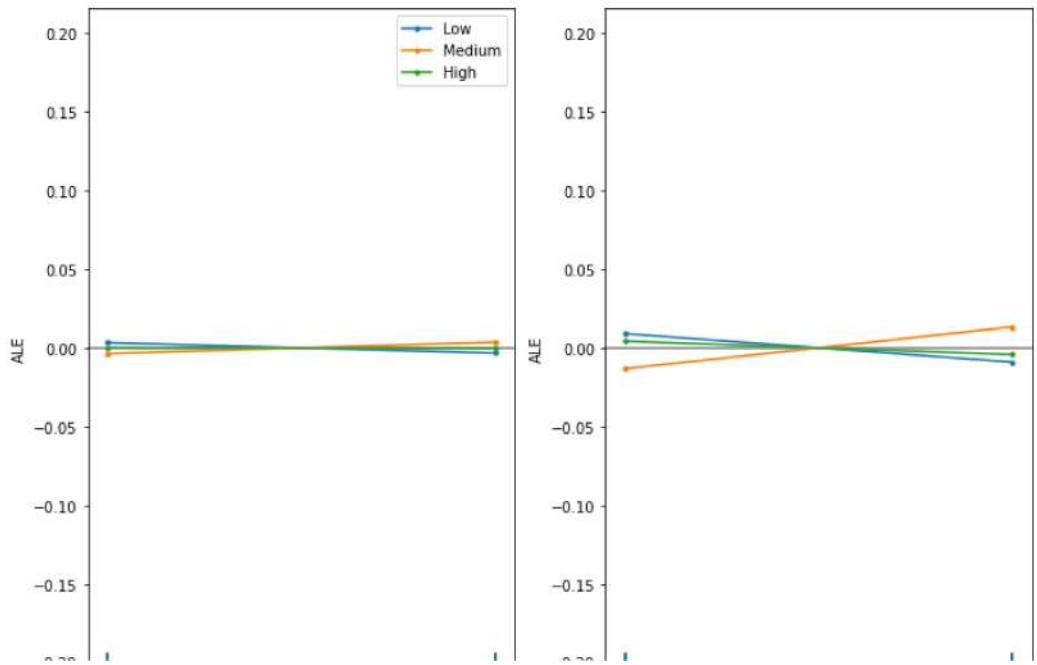
```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names

X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

The graphs of all the data present in the dataset used are shown.

In [252]:
```
plot_ale(proba_exp_lr, n_cols=2, features=list(range(df.shape[1]-1)),fig_kw={'fig
```



Explanation of the model using LIME

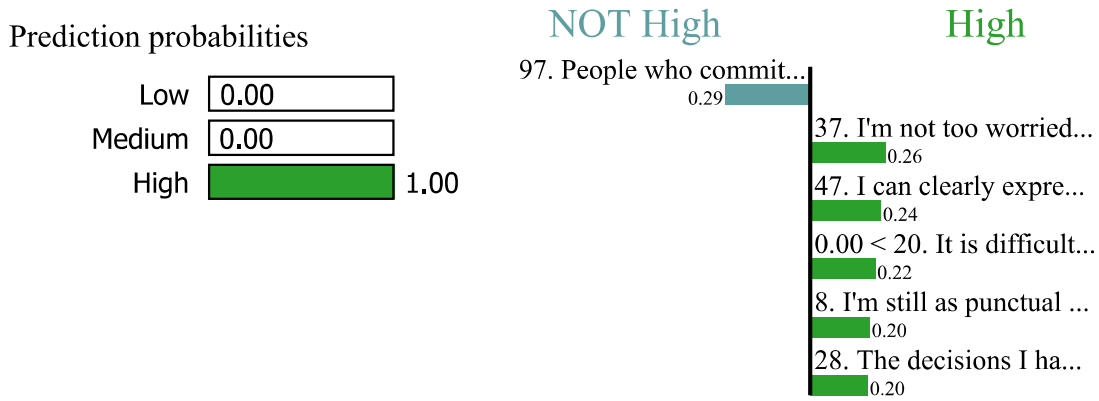
We proceed to explain the improved model using LIME for the Low, Medium and High classes.

In [253]:
```
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorica
                                                   feature_names=train_cols, clas
                                                   discretize_continuous=True)
```

The first individual present in the test data is explained.

In [254]:
```
exp = explainer.explain_instance(X_test.to_numpy()[0], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

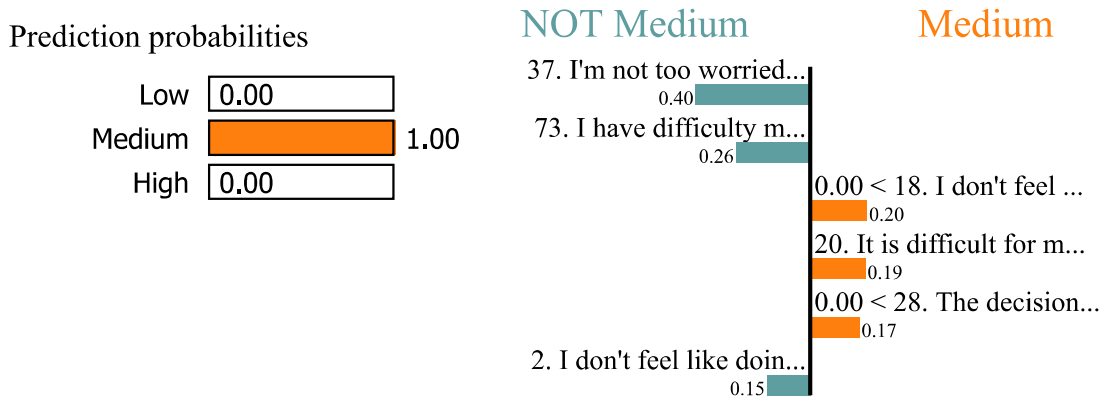


Feature	Value
97. People who commit crimes have their reasons for doing it	0.00
37. I'm not too worried about what might happen in a few weeks	0.00
47. I can clearly express to others how I feel	0.00
20. It is difficult for me to make decisions even if they are simple	1.00
8. I'm still as punctual as I have always been	0.00
28. The decisions I have made so far have been the right ones	0.00

The second individual present in the test data is explained.

In [255]:
```
exp = explainer.explain_instance(X_test.to_numpy()[1], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

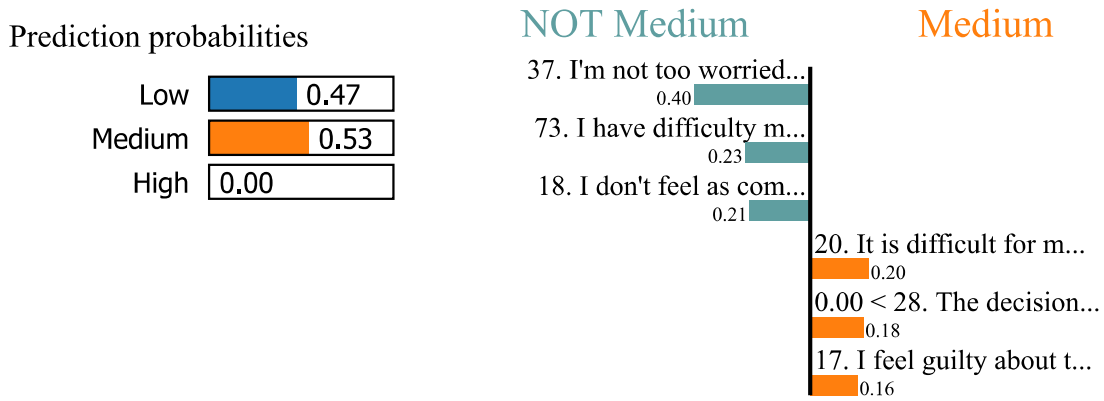


Feature	Value
37. I'm not too worried about what might happen in a few weeks	0.00
73. I have difficulty making elaborate plans	0.00
18. I don't feel as comfortable with my body as I did before	1.00
20. It is difficult for me to make decisions even if they are simple	0.00
28. The decisions I have made so far have been the right ones	1.00
2. I don't feel like doing my daily duties	0.00

The last individual present in the test data is explained.

In [256]:
```
exp = explainer.explain_instance(X_test.to_numpy()[-1], mlp.predict_proba, num_fe
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```



Feature	Value
37. I'm not too worried about what might happen in a few weeks	0.00
73. I have difficulty making elaborate plans	0.00
18. I don't feel as comfortable with my body as I did before	0.00
20. It is difficult for me to make decisions even if they are simple	0.00
28. The decisions I have made so far have been the right ones	1.00
17. I feel guilty about the decisions that I have made	0.00

Explanation of the model using SHAP

SHAP explanatory values are displayed according to the class to which the selected individual belongs according to the prediction of the model.

The first individual present in the test data is explained.

In [257]:
```
## Individual Selection.
individual = X_test.iloc[[0]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The second individual present in the data test is explained.

In [258]:
```
## Individual Selection.
individual = X_test.iloc[[1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The last individual present in the data test is explained.

In [259]:
```
## Individual Selection.
individual = X_test.iloc[[-1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

Explanation using heatmaps

The importance of the data present in the neural network is shown from explanation methods.

In [260]:

```
## Items currently in use are retrieved.
ds_item = ds[ds.Status == "In use"].reset_index(drop=True)

## The data is prepared for the generation of heatmaps.
column_Map = ds_item['Question'].tolist()
content_Map = [False for x in range(1, df.shape[1])]

## The next items to be deleted will be marked.
for index in range(0, len(content_Map)):

    if iteration == 0:
        for delete in list_contr:
            if column_Map[index] == delete:
                content_Map [index] = True
    else:
        for delete in question_anomaly:
            if column_Map[index] == delete:
                content_Map [index] = True

## The dataframe is created with the prepared data.
df_mask = pd.DataFrame([content_Map], columns = column_Map)
df_mask_imp = pd.concat([df_mask, df_mask, df_mask], ignore_index = True, axis =
df_mask_all = pd.concat([df_mask, df_mask, df_mask, df_mask], ignore_index = True

df_mask
```

Out[260]:

	1	2	6	7	8	10	13	16	17	18	...	77	79	80	81
0	False	False	False	True	False	False	False	False	False	True	...	True	False	False	False

1 rows × 49 columns

Identification of feature variables

Identification of feature variables from the randomness of the answers as items of the data.

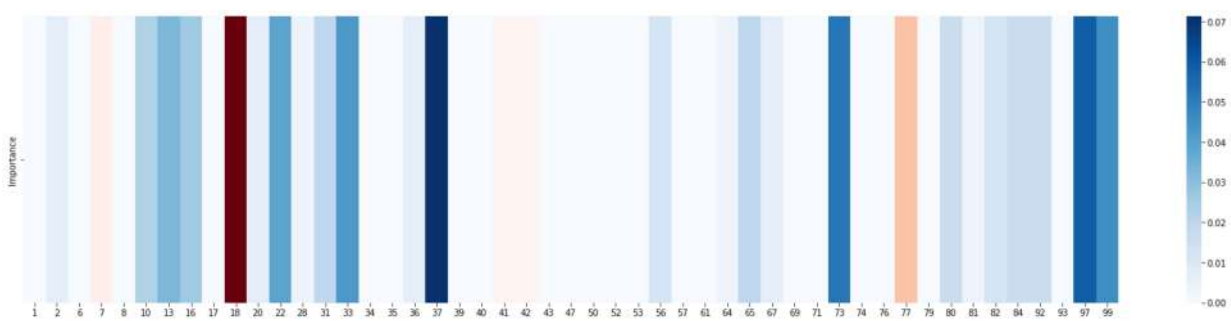
In [261]:
```
## Loading the data for explainability.
explainer = ClassifierExplainer(mlp, X_test, y_test)

## Class index local variable.
iterable_class = -1
map_class = []
```

```
WARNING: Parameter shap='guess', but failed to guess the type of shap explainer
to use for MLPClassifier. Defaulting to the model agnostic shap.KernelExplainer
(shap='kernel'). However this will be slow, so if your model is compatible with
e.g. shap.TreeExplainer or shap.LinearExplainer then pass shap='tree' or shap
='linear'!
WARNING: For shap='kernel', shap interaction values can unfortunately not be ca
lculated!
Note: shap values for shap='kernel' normally get calculated against X_backgroun
d, but paramater X_background=None, so setting X_background=shap.sample(X, 5
0)...
Generating self.shap_explainer = shap.KernelExplainer(model, X, link='identit
y')
```

Explainability of the data belonging to the first class, in this case the Low.

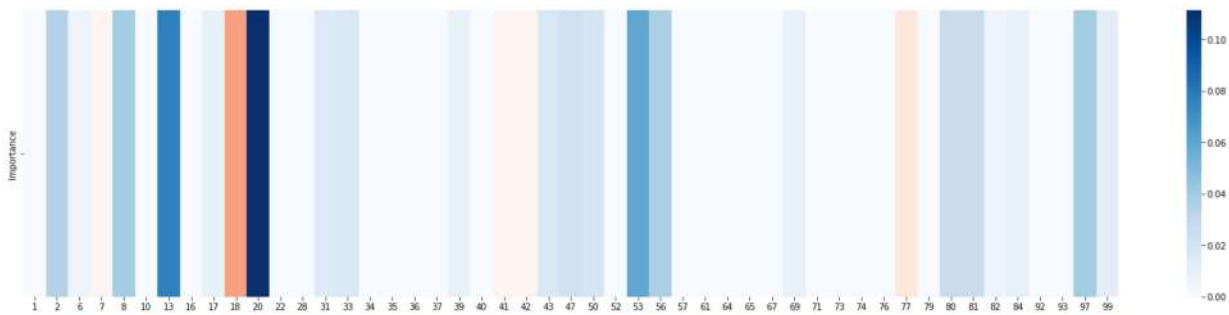
In [262]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

```
Calculating permutation importances (if slow, try setting n_jobs parameter)...
```



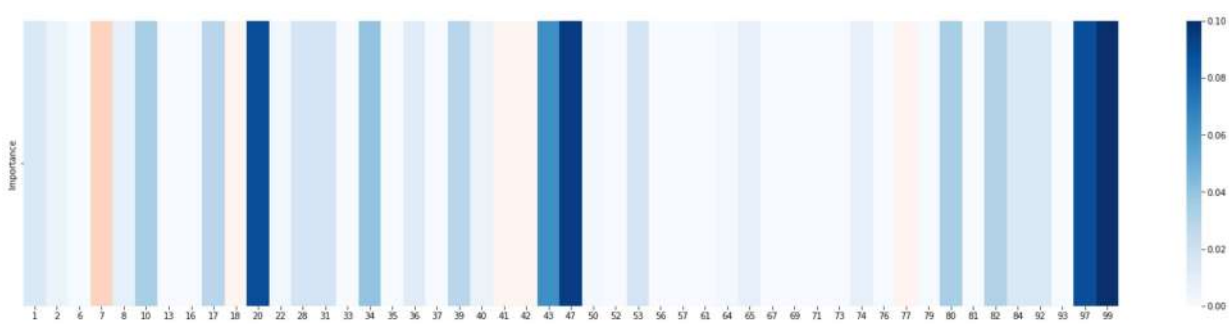
Explainability of the data belonging to the second class, in this case the Medium.

In [263]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



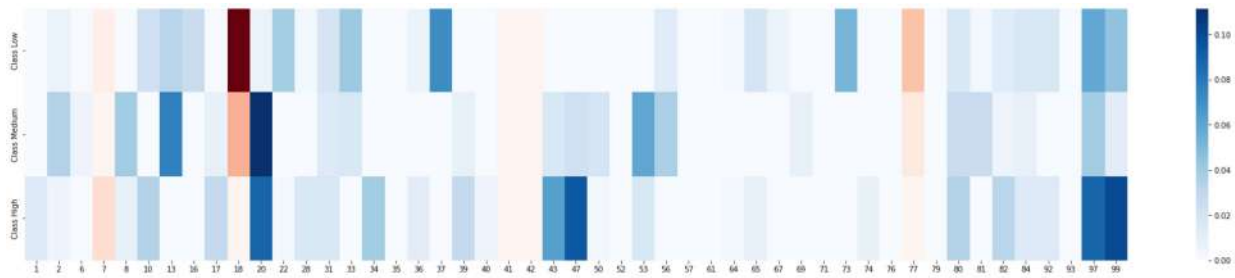
Explainability of the data belonging to the third class, in this case the High.

In [264]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [265]:
```
## Fusion of the previous heatmaps.
df_importance = pd.concat(map_class, axis=1)
df_importance.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_importance.transpose(), cmap = "Reds", cbar = False)
sns.heatmap(df_importance.transpose(), cmap = "Blues",yticklabels = True, xtickla
plt.show()
```



Identification of feature variables using SHAP

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of SHAP.

In [266]:
```
## Class index local variable.
iterable_class = -1
map_class = []
```

Individual explainability of the data belonging to the first class, in this case the Low.

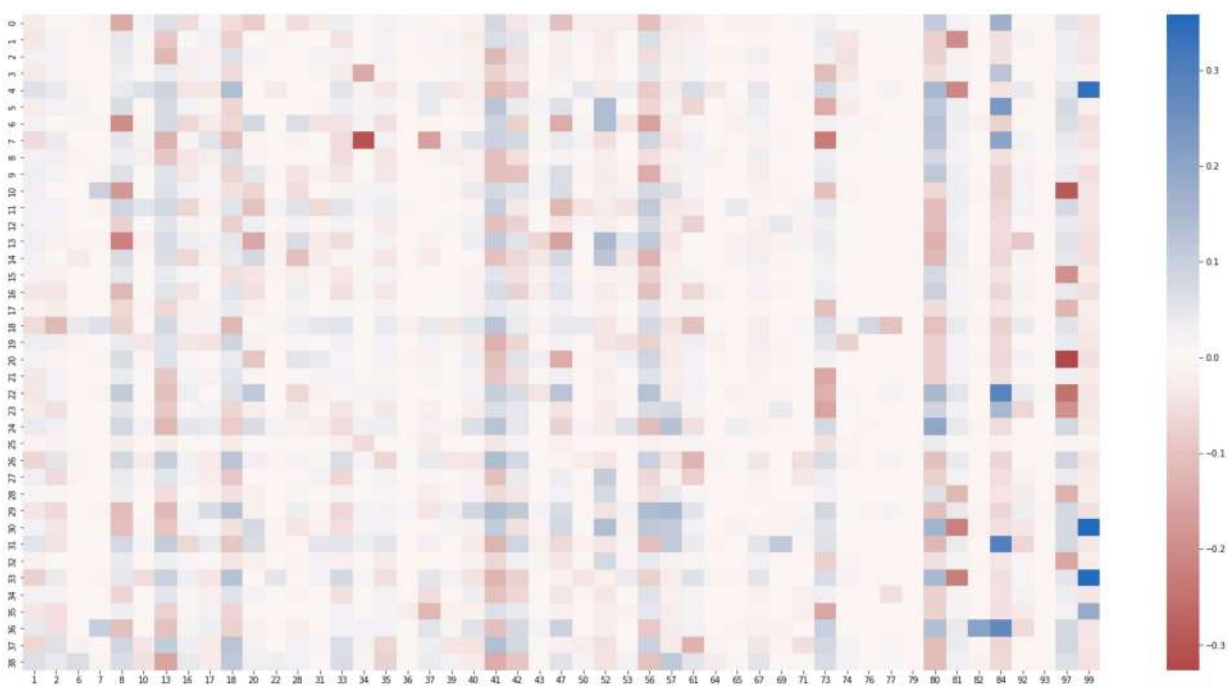
Definition of explainability data.

In [267]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

```
Calculating shap values...

  0%|          | 0/39 [00:00<?, ?it/s]
```

Generation of the heatmap.

In [268]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

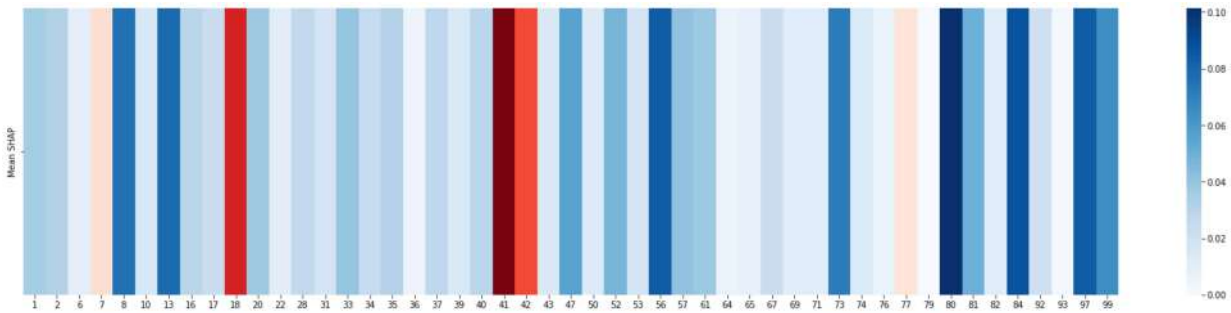


Fusion of the data used.

In [269]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [270]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



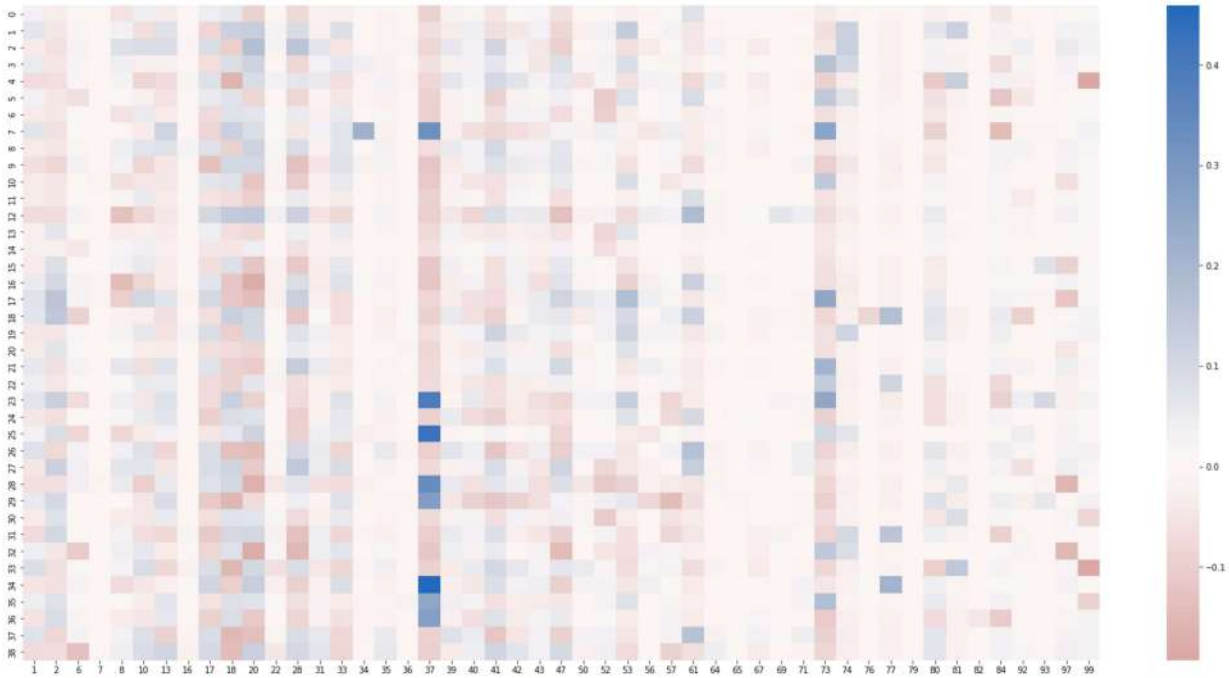
Individual explainability of the data belonging to the second class, in this case the Medium.

Definition of explainability data.

In [271]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [272]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

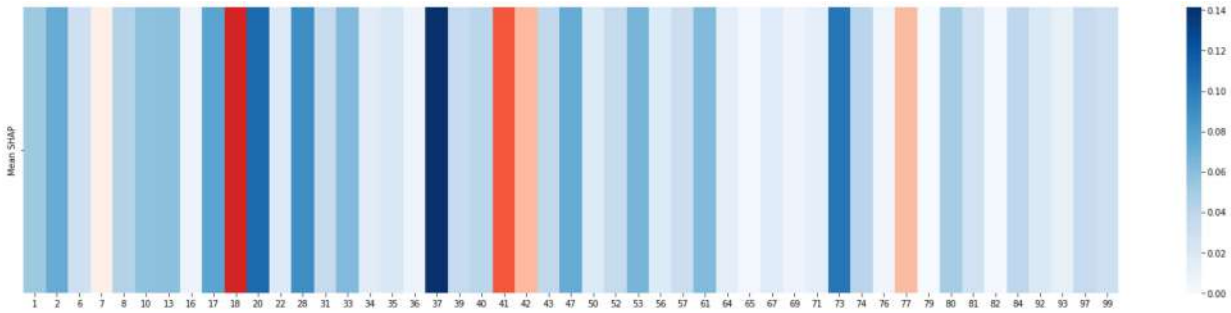


Fusion of the data used.

In [273]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [274]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



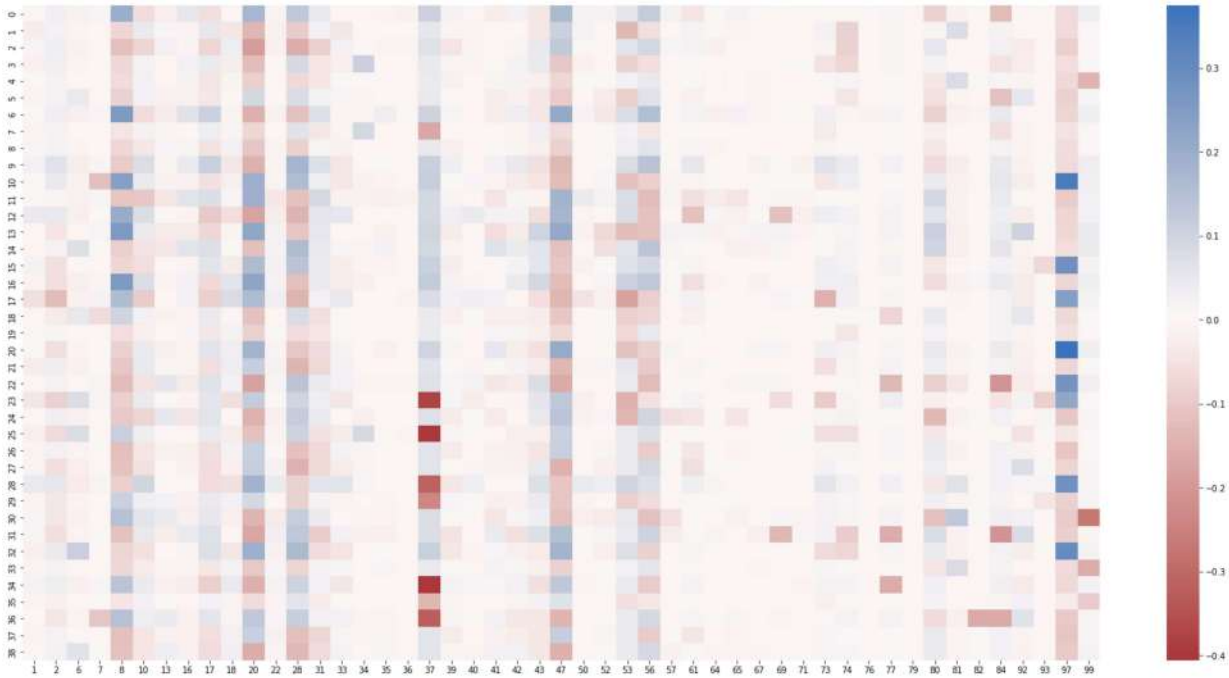
Individual explainability of data belonging to the third class, in this case High.

Definition of explainability data.

In [275]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [276]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

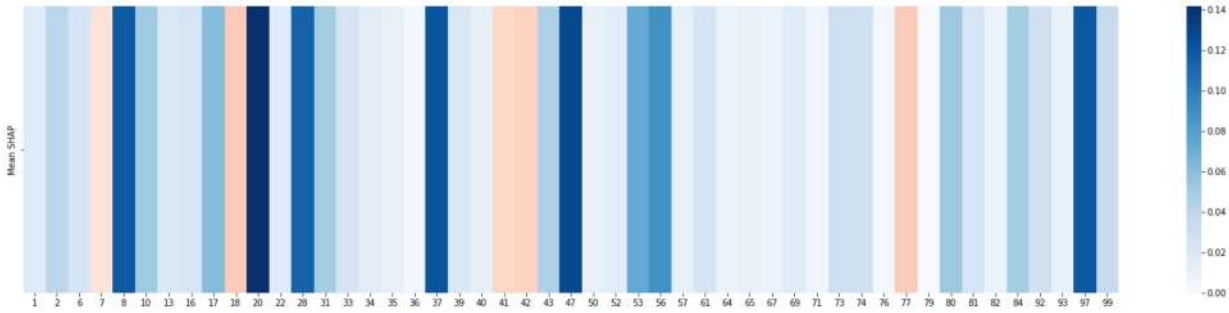


Fusion of the data used.

In [277]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

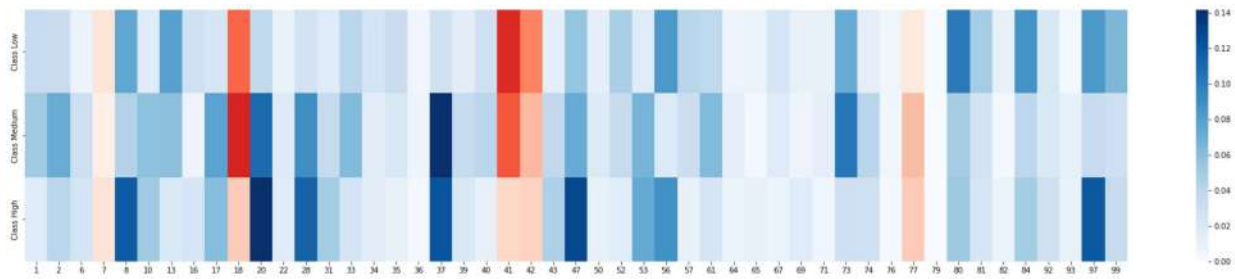
Generation of the heatmap.

In [278]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [279]:
```
## Fusion of the previous heatmaps.
df_shap = pd.concat(map_class, axis=1)
df_shap.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_shap.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_shap.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using LIME

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of LIME.

In [280]:
```
## Data preparation for explainability.
lime_exp = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorical
                                                  feature_names=train_cols, class
                                                  discretize_continuous=True)

## Class index local variable.
iterable_class = -1
map_class = []
```

Obtaining the data for the generation of the explanations by LIME about the Low, Medium and High classes.

In [281]:
```
## Local variables for LIME array data.
exp_matrix = [[] for _ in range(len(name_class))]

## Recovering the data needed for LIME.
for x in X_test.to_numpy():

    ## Auxiliary temporary variables.
    exp_list = [[] for _ in range(len(name_class))]

    ## Loading the data for explainability.
    exp = lime_exp.explain_instance(x, mlp.predict_proba, num_features = df.shape

    ## Recovering the data.
    for elements in range(0, len(name_class)):
        temp = exp.as_map()[elements]
        temp.sort(key=itemgetter(0))

        ## Saving data of the Low, Medium and High classes.
        for tup in temp:
            exp_list[elements].append(tup[1])

        exp_matrix[elements].append(exp_list[elements])
```

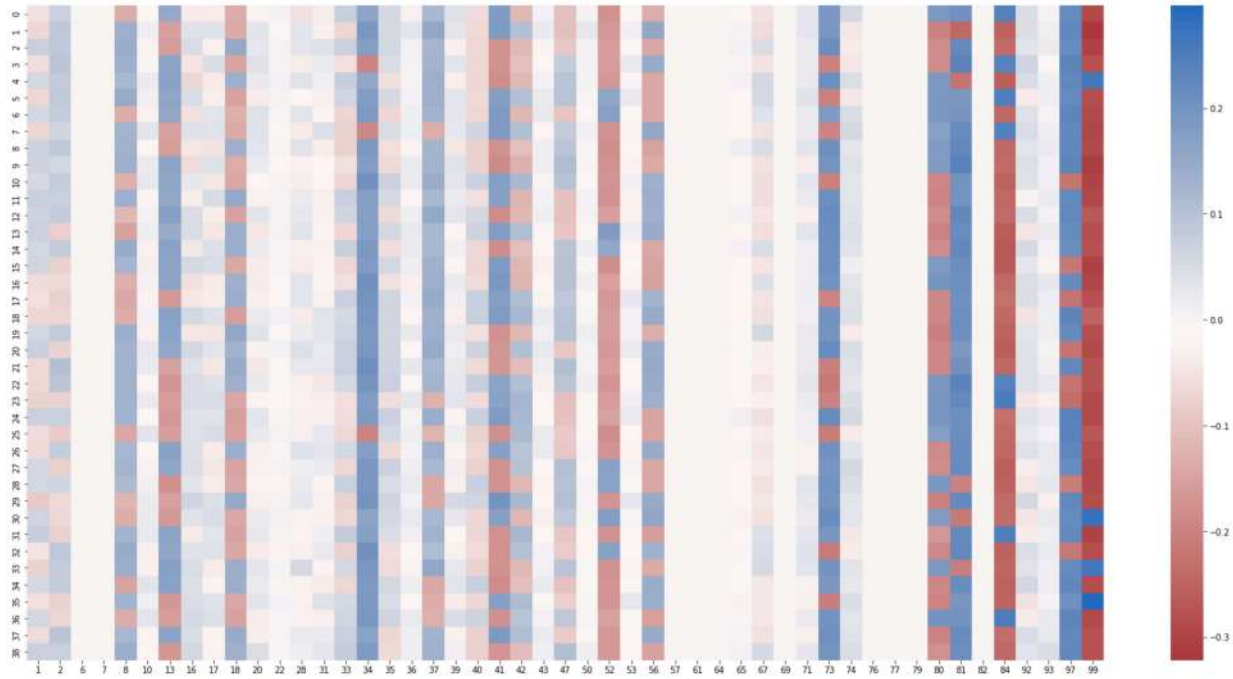
```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

Individual explainability of the data belonging to the first class, in this case the Low.

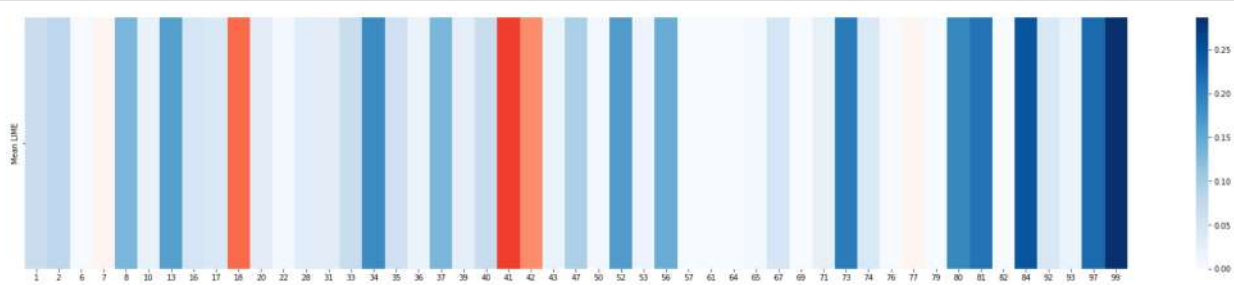
Generation of the heatmap.

In [282]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

In [283]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

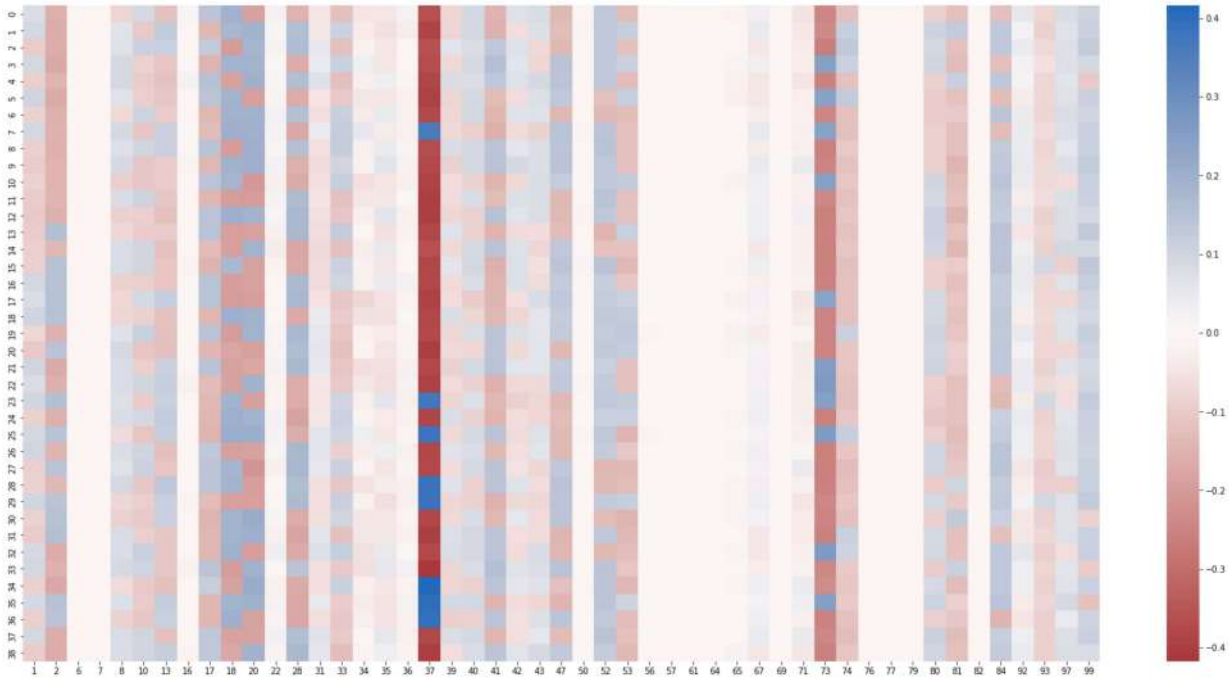


Individual explainability of the data belonging to the second class, in

this case the Medium.

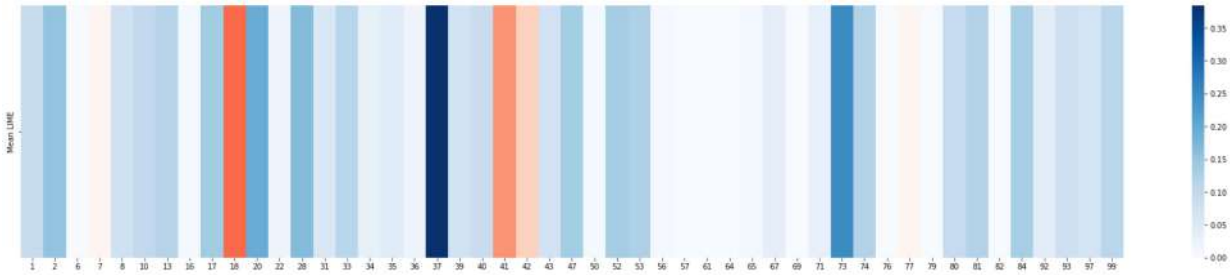
Generation of the heatmap.

In [284]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

In [285]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

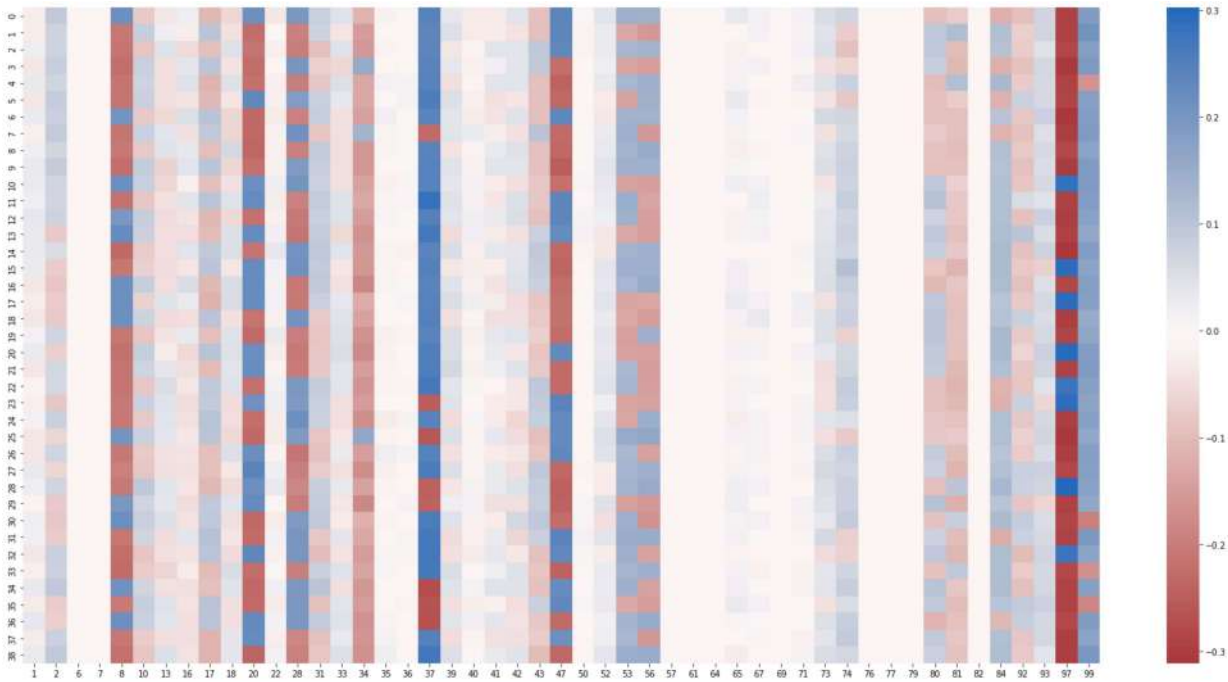


Individual explainability of data belonging to the third class, in this

case High.

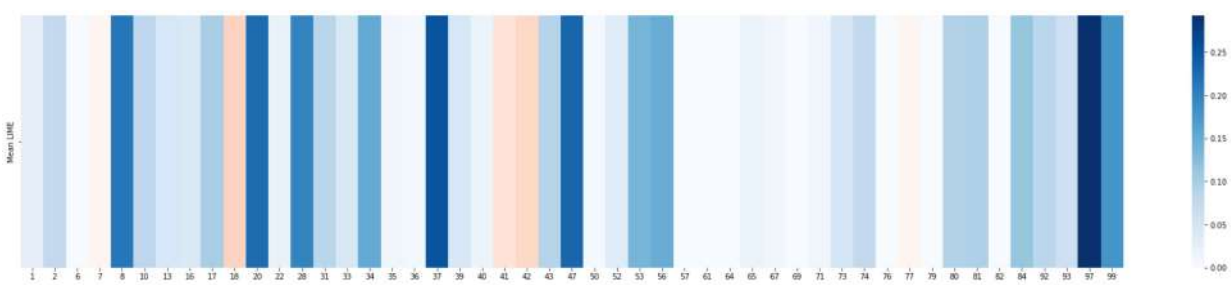
Generation of the heatmap.

In [286]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



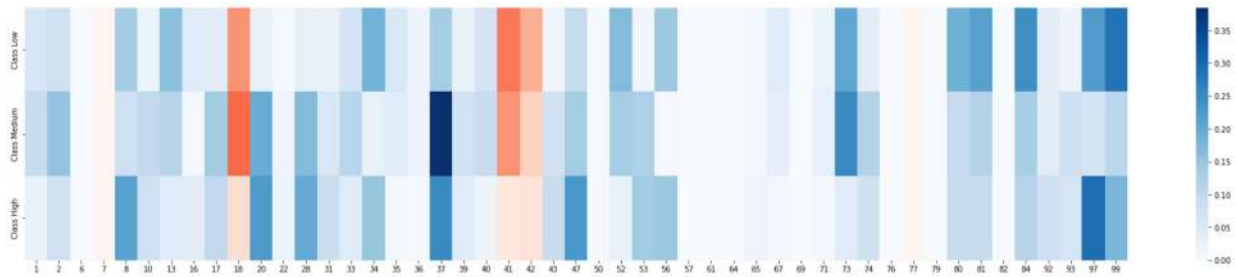
Fusion of the data used.

In [287]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [288]:
```
df_lime = pd.concat(map_class, axis=1)
df_lime.columns = name_class
plt.figure(figsize = (32,6))
sns.heatmap(df_lime.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_lime.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using ALE

It shows the main effect of each question compared to the probability that the model predicts in each class. This effect is shown for both responses. Since there are only 2 answers, the map is the same but with the colors reversed.

In [289]:
```
## Definition of explainability data.
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names=train_cols, target_names= target_r
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())

## Class index local variable.
iterable_class = -1
map_class = []
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

Obtaining the data for the generation of the explanations by ALE about all the classes.

In [290]:

```
## Local variables for ALE array data.
ale_list = [[] for _ in range(len(name_class))]

## Recovering the data needed for ALE.
for array in proba_exp_lr.ale_values:

    ## Recovering the data.
    for elements in range(0, len(name_class)):

        ## Saving data of all the classes.
        ale_list[elements].append(array[0][elements])
```

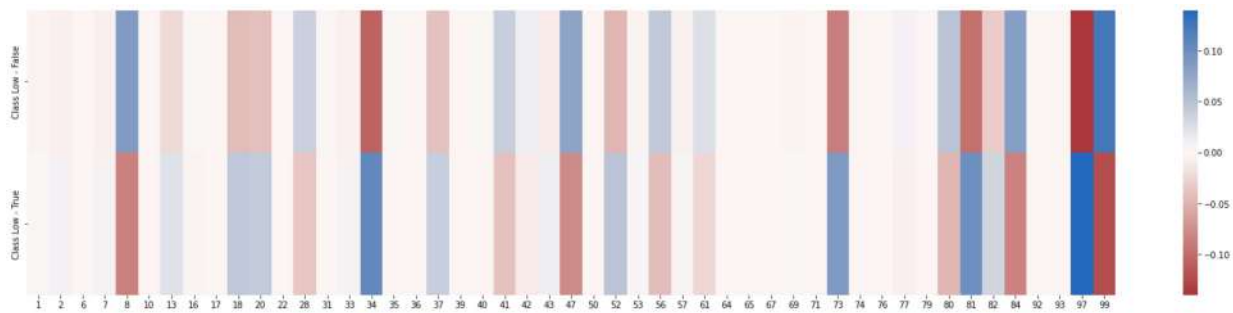
Data processing for the explainability of ALE.

In [291]:
```
## Data processing of the Low, Medium and High classes.
for elements in range(0, len(name_class)):
    ale_df = pd.DataFrame([ale_list [elements]])
    ale_df = pd.concat([ale_df.multiply(-1), ale_df])
    ale_df.index = [name_class [elements] + " - False", name_class [elements] + '
    ale_df.columns = column_Map
    map_class.append(ale_df)
```

Explainability of the data belonging to the first class, in this case the Low.

Generation of the heatmap.

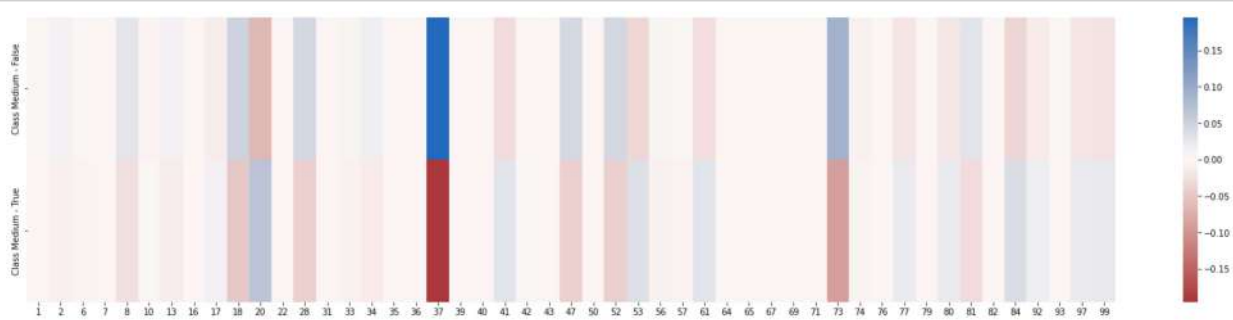
In [292]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the second class, in this case the Medium.

Generation of the heatmap.

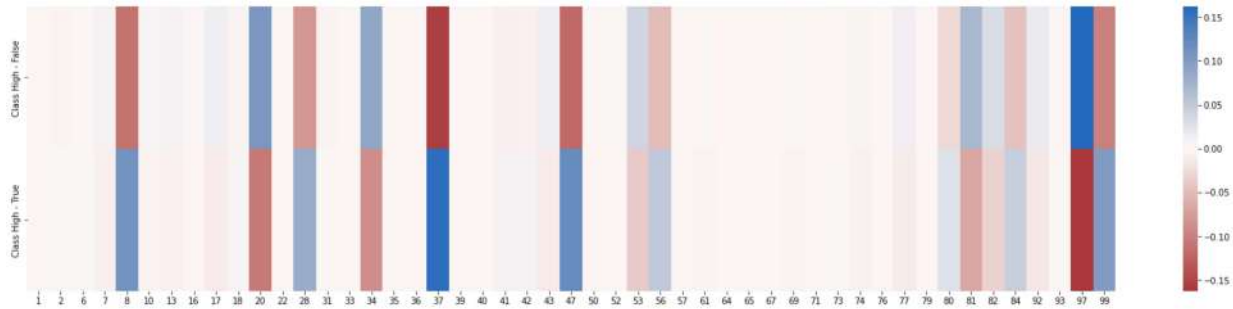
In [293]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the third class, in this case the High.

Generation of the heatmap.

In [294]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```

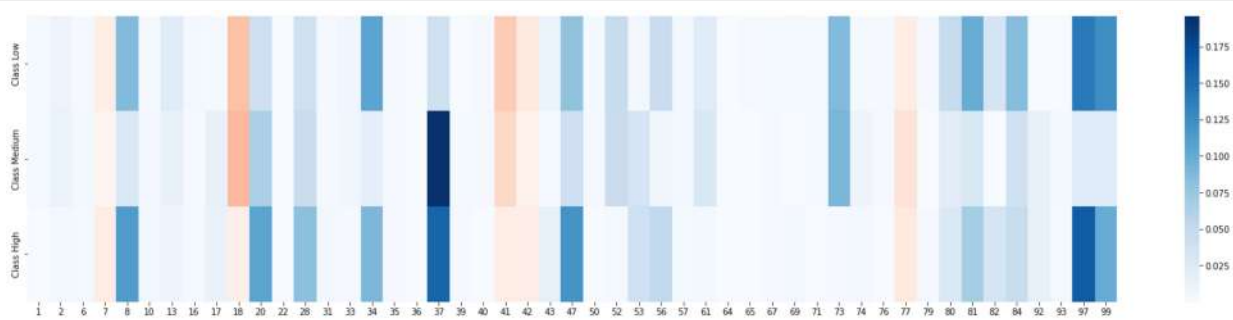


Fusion of the data used in all classes.

In [295]:
```
## Application of absolute value to data.
for index in range(0, len(map_class)):
    map_class [index] = map_class [index][0:1].abs()

## Union of the heatmaps of the Low, Medium and High class.
ale_df = pd.concat(map_class)
ale_df.index = name_class
ale_df = ale_df.transpose()
```

Generation of the heatmap.

In [296]:
```
plt.figure(figsize = (28,6))
sns.heatmap(ale_df.transpose(), cmap="Reds", cbar = False)
sns.heatmap(ale_df.transpose(), cmap="Blues", yticklabels = True, xticklabels = T
plt.show()
```



Comparing of Feature Variables Methods for Explainability

The global values of each of the methods used are taken, their maximums are obtained (the minimums are 0) and a percentage is calculated based on said maximum, which indicates how relevant the contribution is for each question.

In [297]:
```
## Class index local variable.
iterable_class = -1
```

Comparison of the explanation methods of the first class, in this case the Low.

The highest data of each method used is obtained.

In [298]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [299]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

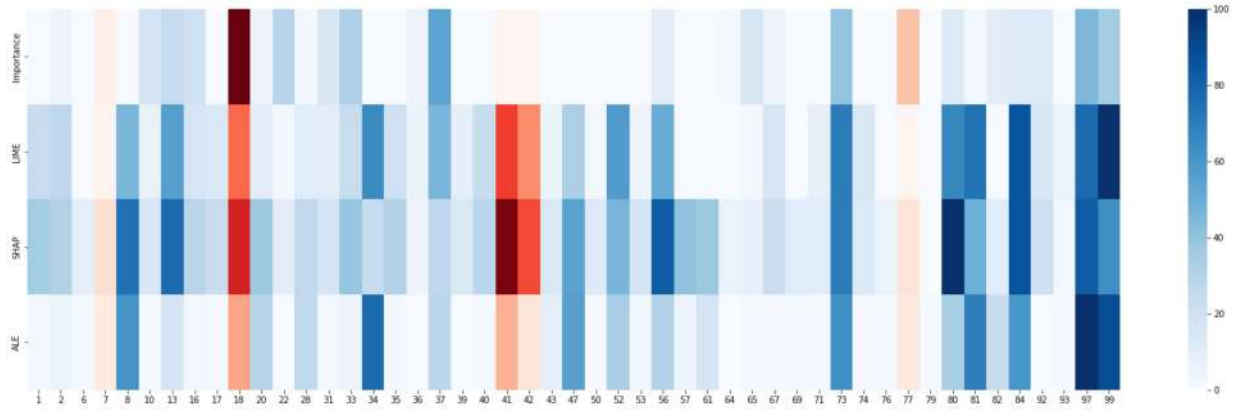
Out[299]:

	1	2	6	7	8	10	13	16
Importance	0.000000	5.000000	0.000000	2.500000	0.000000	17.500000	25.000000	20.000000
LIME	22.438631	27.156199	0.000000	0.000000	46.429745	7.399605	56.085495	16.140297
SHAP	34.966065	31.368165	9.047152	12.690030	74.760136	15.834094	77.709098	28.807183
ALE	2.364628	6.454780	0.666747	6.208866	61.852897	0.998094	16.872555	2.780838

4 rows × 49 columns

Generation of the heatmap.

In [300]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the second class, in this case the Medium.

The highest data of each method used is obtained.

In [301]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [302]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

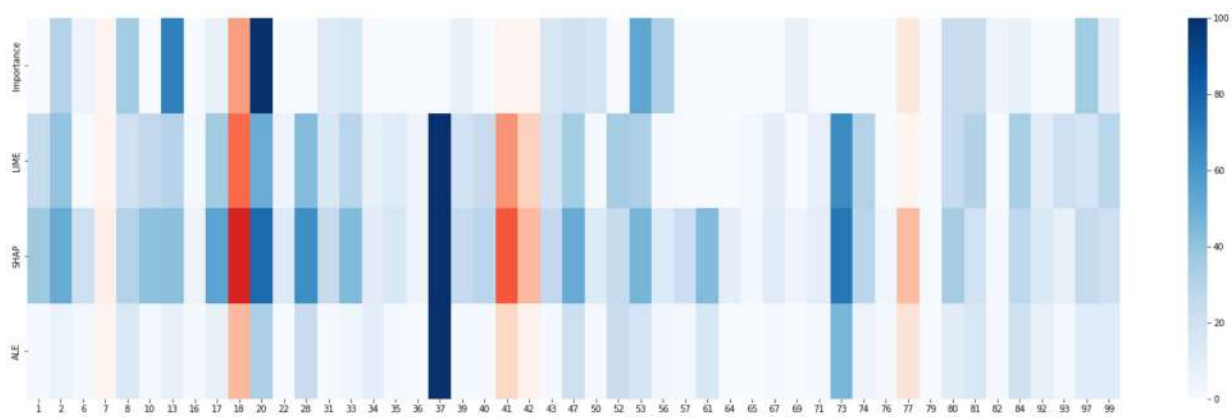
Out[302]:

	1	2	6	7	8	10	13	16
Importance	0.000000	30.769231	5.128205	0.000000	35.897436	0.000000	69.230769	0.000000
LIME	24.096993	40.120889	0.000000	0.000000	20.438746	26.049693	29.875611	1.824588
SHAP	37.109667	50.794862	21.536490	3.099930	30.368456	41.309667	42.147078	5.278015
ALE	1.842787	6.775381	2.401324	0.298259	13.448134	2.484138	7.539309	0.899854

4 rows × 49 columns

Generation of the heatmap.

In [303]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the third class, in this case the High.

The highest data of each method used is obtained.

In [304]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [305]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

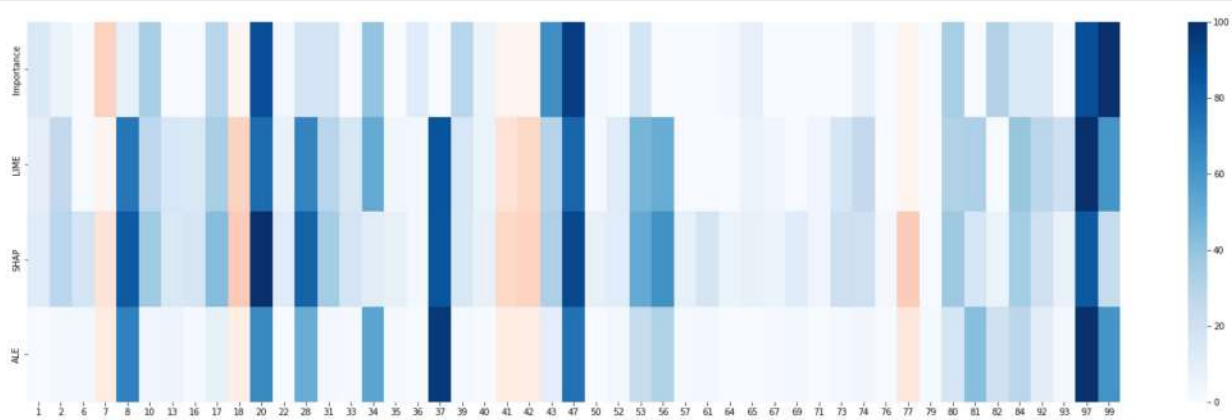
Out[305]:

	1	2	6	7	8	10	13	16
Importance	14.285714	5.714286	0.000000	17.142857	8.571429	34.285714	0.000000	0.000000
LIME	9.661362	26.084194	0.000000	0.000000	72.339469	26.962449	15.719017	14.139276
SHAP	12.047021	28.305022	18.011666	10.523445	83.637022	36.367671	13.962539	16.771555
ALE	0.186923	2.614916	2.322166	5.704601	69.465092	3.855218	5.431854	1.308601

4 rows × 49 columns

Generation of the heatmap.

In [306]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Safeguarding the items eliminated during this iteration.

In [307]:
```
## Removal of anomalous items selected by selection_mode.
for x in question_anomaly:

    ## Status change of the deleted items in the respective auxiliary dataset.
    ds.loc[x - 1,'Status'] = 'Delete'

## Save the changed statuses in the dataset.
ds.to_csv(path_dataset_qs, index = False)
```

Iteration 4

The first iteration of improvement of the model is carried out.

Preparing the data for the neural network

The data is prepared in order to be processed by the neural network.

In [308]:

```
## The iteration number is increased.
iteration += 1

## Loading data from the dataset.
df = pd.read_csv(path_dataset)
ds = pd.read_csv(path_dataset_qs)

## Elimination of unrelated items from the dataset.
if len(unrelated_questions) != 0:
    df = df.drop(df.columns[unrelated_questions[0]:unrelated_questions[1]], axis=

i = 0
if iteration == 1:
    ## Elimination of list of conflicting items declared in global variables.
    for x in list_contr:
        df.drop(df.columns[x - (i + 1)], axis = 1, inplace = True)
        i += 1
        ## Status change of the deleted question in the respective auxiliary data
        ds.loc[x - 1,'Status'] = 'Delete'

else:
    ## Elimination of the list of anomadic items defined in the improvement proce
    ds_delete = ds[ds.Status == "Delete"].reset_index(drop=True)
    for x in ds_delete['Question'].tolist():
        df.drop(df.columns[x - (i + 1)], axis = 1, inplace = True)
        i += 1

## Grouping of the target according to the defined dictionary.
if grouping_name != None and grouping_name != '':
    df[grouping_name] = df[grouping_name].map(class_dic)

## Assignment of local variables according to the data necessary for the neural r
train_cols = df.columns [0:-1]
label = df.columns [-1]
X = df [train_cols]
y = df [label]

## The prepared data of the dataset is printed.
df
```

Out[308]:

	1. Most of the time I have difficulty concentrating on simple tasks	2. I don't feel like doing my daily duties	6. I consider that my life is full of good things	8. I'm still as punctual as I have always been	10. I have found that I can spend a lot of time scrolling the screen \nof my cell phone without searching or stopping at anything in particular	13. I felt more encouraged to do my daily activities before	16. I think my thoughts are strange or different from before	17. I feel guilty about the decisions that I have made
0	1	1	1	1	0	0	0	0
1	1	0	1	1	0	1	1	1
2	0	0	1	0	1	0	1	0
3	0	0	1	1	0	1	0	0
4	1	1	1	1	1	0	1	1
...	...	...	...	...	...	...	...	...
99	1	1	1	1	1	1	0	0
100	1	1	1	0	0	1	0	0
101	0	0	1	0	1	1	1	0
102	1	0	1	1	1	0	0	1
103	1	1	1	0	1	0	0	0

104 rows × 45 columns

Oversample application

An oversample process is applied to level the amount of existing data given by the grouping.

In [309]:
```
## The number of elements present in the grouped target is printed.
y.value_counts()
```

Out[309]:
```
0    39
1    35
2    30
Name: Target, dtype: int64
```

Oversample application.

In [310]:
```
## Application of the oversample to the prepared data.
X, y = oversample.fit_resample(X, y)
```

Printing of the data after the application of the oversample.

In [311]:
```
## The results obtained after applying the oversample are printed.
y.value_counts()
```

Out[311]:
```
0    39
1    39
2    39
Name: Target, dtype: int64
```

Preparation of the parameters of the neural network

We proceed to define the parameters necessary for the training of the neural network.

In [312]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
```

Neural network training

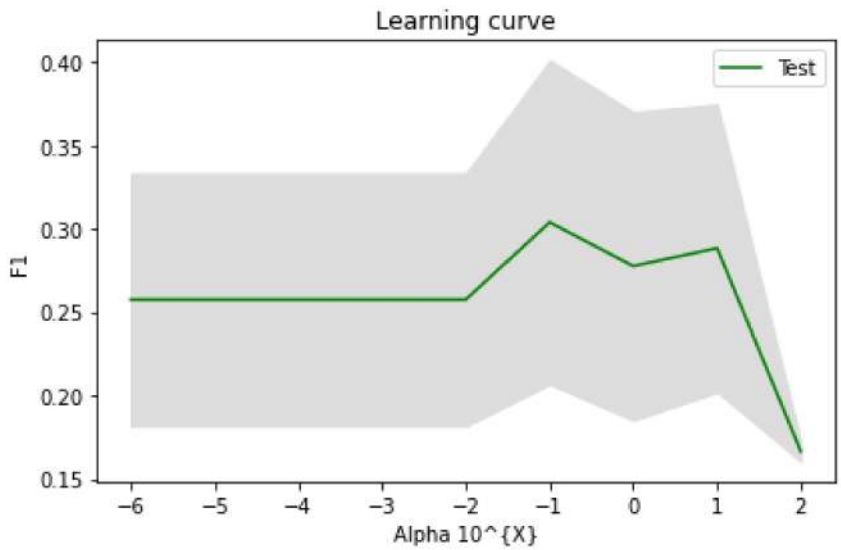
The alpha parameters and the initial learning rate are adjusted to obtain the best performance with the model using cross validation.

In [313]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_normal

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = 0.0001, max_iter =

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[313]:
```
([0.16659498207885304,
  0.28843132176465514,
  0.2776667776667776,
  0.3040008983868633,
  0.2575300255300256,
  0.2575300255300256,
  0.2575300255300256,
  0.2575300255300256,
  0.2575300255300256],
 [0.006020651927912844,
  0.08630510353212997,
  0.09251139902038742,
  0.09711067192502043,
  0.07582590789635271,
  0.07582590789635271,
  0.07582590789635271,
  0.07582590789635271,
  0.07582590789635271])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [314]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Alpha 10^{X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



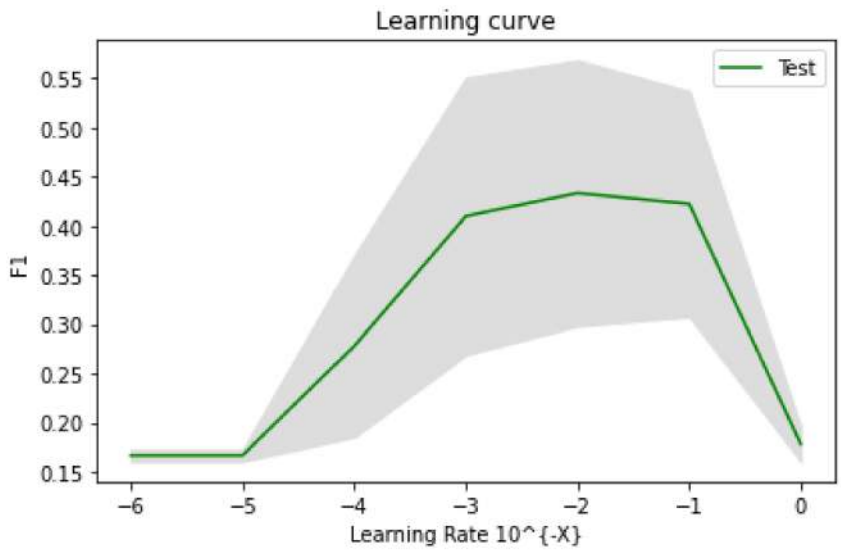
The training is performed again keeping the same parameters except for the alpha value that changes to 1.

In [315]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_alpha

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = regul_param, max_it

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[315]:
```
([0.17806090178006687,
  0.4222391190967042,
  0.43333889593951513,
  0.40979484224066265,
  0.2776667776667776,
  0.16659498207885304,
  0.16659498207885304],
 [0.01757734769209096,
  0.11485289010636886,
  0.13511435367470756,
  0.14103979938781133,
  0.09251139902038742,
  0.006020651927912844,
  0.006020651927912844])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [316]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Learning Rate 10^{-X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



Generation of the neural network model

The parameters of the final model are modified according to the data obtained in the training of the neural network.

In [317]:
```
## The value of the alpha parameter, learning_rate_init and random_state are char
## to the data in the global variables.
mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solver =
                    learning_rate = 'constant', learning_rate_init = lear
                    random_state = random_state_model)

## We print the data of the final generated model.
mlp.fit(X_train, y_train)
```

Out[317]:
```
MLPClassifier
MLPClassifier(activation='logistic', alpha=0.001, hidden_layer_sizes=(10,),
              learning_rate_init=0.1, max_iter=100000, random_state=42)
```

We safeguard the final generated model

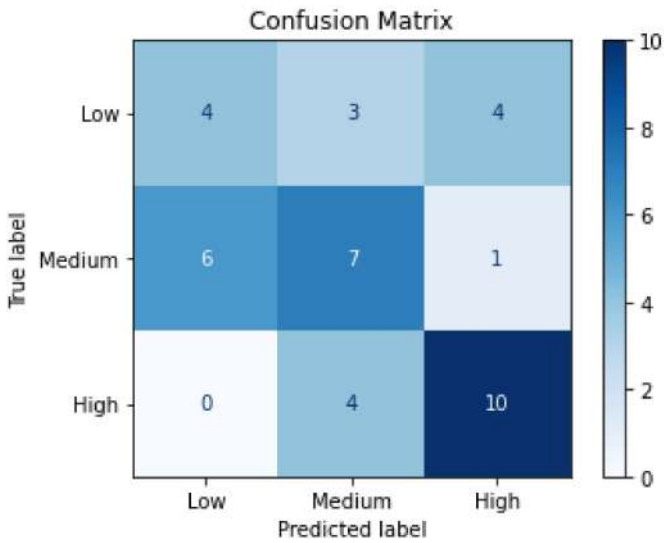
In [318]:
```
joblib.dump(mlp,"model_depression_i" + str(iteration) + ".pkl")
```

Out[318]:
```
['model_depression_i4.pkl']
```

Generation of the confusion matrix of the model

We obtain the statistics of the final model to generate its confusion matrix.

In [319]:
```
## The data for the generation of the confession matrix are defined.
confusion_matrix = ConfusionMatrixDisplay.from_estimator(mlp, X_test, y_test, dis
                                                         cmap = plt.cm.Blues)
confusion_matrix.ax_.set_title("Confusion Matrix")
plt.show()
```



We get the detailed statistics of the final model.

In [320]:
```
## The necessary detailed prediction is generated.
y_pred = mlp.predict(X_test)

## The detailed statistics of the model are printed.
print('Classification accuracy =',accuracy_score(y_test,y_pred) * 100,'%\n')
print(classification_report(y_test,y_pred))
```

```
Classification accuracy = 53.84615384615385 %

              precision    recall  f1-score   support

           0       0.40      0.36      0.38        11
           1       0.50      0.50      0.50        14
           2       0.67      0.71      0.69        14

    accuracy                           0.54        39
   macro avg       0.52      0.53      0.52        39
weighted avg       0.53      0.54      0.53        39
```

The data obtained is saved for later graphing.

In [321]:
```
accuracy = accuracy_score(y_test,y_pred) * 100
clf_rep = precision_recall_fscore_support(y_test,y_pred)

## The number of items used in the iteration is retrieved.
ds_delete = ds[ds.Status == "Delete"]
ds_delete.reset_index(inplace=True, drop=True)

## All the metrics of the confusion matrix are obtained.
metrics = [",".join(map(str, ds_delete['Question'].tolist())),
           accuracy]

for index in range(0, len(name_class)):
    metrics.append(clf_rep[0][index])
    metrics.append(clf_rep[1][index])
    metrics.append(clf_rep[2][index])
    metrics.append(clf_rep[3][index])

## The names of the columns of the dataset are defined.
columns = ['Question', 'Acurracy global']

for index in range(0, len(name_class)):
    columns.append('Precision_' + str(index))
    columns.append('Recall_' + str(index))
    columns.append('F1_score_' + str(index))
    columns.append('Support_' + str(index))

## A new row of the dataset is generated with all the data.
data_metrics = pd.DataFrame([metrics], columns = columns)

## The data is saved to the dataset.
data_metrics.to_csv(path_dataset_qd, mode = 'a', header = False, index = False)

## The data added to the dataset is printed.
data_metrics
```

Out[321]:

	Question	Acurracy global	Precision_0	Recall_0	F1_score_0	Supp
0	3,4,5,7,9,11,12,14,15,18,19,21,23,24,25,26,27,...	53.846154	0.4	0.363636	0.380952	

Improvement process of the neural network model.

The application of the explanation method by ALE allows to determine the behavior of the neural network and thereby improve it by discarding non-significant data for it.

Slope Analysis of ALE Plots

In [322]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

We proceed to obtain the graphs to obtain the slopes of the selected classes.

In [323]:
```
## The question metrics dataset is loaded.
dm = pd.DataFrame()

## The number of items used in the iteration is retrieved.
ds_item = ds[ds.Status == "In use"]
ds_item.reset_index(inplace=True, drop=True)

## We get the slopes from the ALE data.
for i in range(df.shape[1]-1):
    structure_data = {'Question': ds_item.loc[i, "Question"]}

    for index in selection_class:
        slope = proba_exp_lr.ale_values [i][1][index] - proba_exp_lr.ale_values [
        structure_data['Slope ' + str(target_names[index])] = slope
        structure_data['Threshold ' + str(target_names[index])] = 'NA'
        structure_data['Anomaly ' + str(target_names[index])] = 'NA'

    dm = dm.append(structure_data, ignore_index=True)

## The current rating data dataset is printed.
dm
```

```
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
```

Slope analysis and threshold application

The slope obtained is analyzed and those that are above or below the positive and negative threshold are selected.

In [324]:
```
for i in range(df.shape[1]-1):
    for index in selection_class:

        ## The slopes are selected according to the values of the thresholds of t
        if dm.loc[i, "Slope " + str(target_names[index])] >= positive_threshold:
            dm.loc[i, "Threshold " + str(target_names[index])] = 1
        elif dm.loc[i, "Slope " + str(target_names[index])] <= negative_threshold
            dm.loc[i, "Threshold " + str(target_names[index])] = 0

## The current rating data dataset is printed.
dm
```

Out[324]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	1	-0.059830	0	NA	0.064008	1	NA
1	2	-0.000434	NA	NA	-0.008779	NA	NA
2	6	-0.004093	NA	NA	-0.000412	NA	NA
3	8	0.003680	NA	NA	-0.003408	NA	NA
4	10	0.061062	1	NA	0.023922	1	NA
5	13	-0.113167	0	NA	0.000994	NA	NA
6	16	-0.001265	NA	NA	0.000027	NA	NA
7	17	-0.005767	NA	NA	0.012354	1	NA
8	20	0.010568	1	NA	0.022650	1	NA
9	22	-0.017375	0	NA	-0.018091	0	NA
10	28	-0.004061	NA	NA	-0.182532	0	NA
11	31	0.001788	NA	NA	-0.010339	0	NA
12	33	-0.058594	0	NA	-0.011171	0	NA
13	34	-0.121509	0	NA	0.112861	1	NA
14	35	0.022568	1	NA	-0.008039	NA	NA
15	36	-0.006471	NA	NA	0.000525	NA	NA
16	37	-0.285420	0	NA	-0.182475	0	NA
17	39	0.032362	1	NA	-0.047259	0	NA
18	40	0.003534	NA	NA	0.000480	NA	NA
19	43	-0.021338	0	NA	0.057640	1	NA
20	47	0.151068	1	NA	-0.198512	0	NA
21	50	0.122161	1	NA	-0.050008	0	NA
22	52	-0.086237	0	NA	0.028765	1	NA
23	53	0.078599	1	NA	0.114902	1	NA
24	56	0.176395	1	NA	-0.177860	0	NA
25	57	-0.041006	0	NA	0.037299	1	NA
26	61	0.092091	1	NA	0.027141	1	NA

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
27	64	0.014939	1	NA	-0.011734	0	NA
28	65	-0.005018	NA	NA	0.015671	1	NA
29	67	-0.085142	0	NA	0.056442	1	NA
30	69	-0.021778	0	NA	0.026632	1	NA
31	71	-0.001404	NA	NA	0.000678	NA	NA
32	73	-0.144125	0	NA	0.007589	NA	NA
33	74	0.007009	NA	NA	-0.001137	NA	NA
34	76	-0.004313	NA	NA	0.002916	NA	NA
35	79	0.003616	NA	NA	-0.001199	NA	NA
36	80	0.134843	1	NA	-0.083349	0	NA
37	81	-0.216270	0	NA	0.073278	1	NA
38	82	-0.107243	0	NA	0.093070	1	NA
39	84	0.337728	1	NA	-0.177842	0	NA
40	92	-0.003063	NA	NA	0.005385	NA	NA
41	93	-0.000901	NA	NA	0.001172	NA	NA
42	97	-0.244370	0	NA	0.275903	1	NA
43	99	0.341218	1	NA	-0.233854	0	NA

Analysis and determination of anomalous items

The anomalous items present in the model are determined based on their slope and expected response from the expert.

In [325]:

```
## The dataset of expected responses is loaded.
da = pd.read_csv(path_dataset_qa)

## Delete list of items deleted in this iteration.
i = 0
for x in ds_delete['Question']:
    da.drop(da.index[x - (i + 1)], axis = 0, inplace = True)
    i += 1

## Indexes are restored for later use.
da.reset_index(inplace=True, drop=True)

## The dataset of expected responses is printed.
da
```

Out[325]:

	Number	Question	RE
0	1	Most of the time I have difficulty concentrati...	1
1	2	I don't feel like doing my daily duties	1
2	6	I consider that my life is full of good things	0
3	8	I'm still as punctual as I have always been	0
4	10	I have found that I can spend a lot of time sc...	1
5	13	I felt more encouraged to do my daily activiti...	1
6	16	I think my thoughts are strange or different f...	1
7	17	I feel guilty about the decisions that I have ...	1
8	20	It is difficult for me to make decisions even ...	1
9	22	It is not difficult for me to understand somet...	0
10	28	The decisions I have made so far have been the...	0
11	31	It is easy for me to choose a photograph of my...	0
12	33	I have trouble remembering things easily	1
13	34	In recent months I have had discussions with m...	1
14	35	I constantly imagine that something will go wr...	1
15	36	I am afraid of being worng when doing my homework	1
16	37	I'm not too worried about what might happen in...	0
17	39	Everything will be alright	0
18	40	I can easily blank my mind	0
19	43	Sometimes i feel trapped	1
20	47	I can clearly express to others how I feel	0
21	50	I imagine that at any moment a disaster of the...	1
22	52	Being locked in an elevator would be the worst...	1
23	53	I'm bothered by people walking slowly in front...	1
24	56	I manage my schedule as I always have	0
25	57	It bothers me to feel that people on the stree...	1

	Number	Question	RE
26	61	Sometimes I forget what I wanted to say becaus...	1
27	64	I can put down my cell phone and dedicate myse...	0
28	65	I worry that people will not understand what I...	1
29	67	I get angry easily	1
30	69	It is not important for me to meet set dates	1
31	71	I use lies just to get out of certain problems	1
32	73	I have difficulty making elaborate plans	1
33	74	People have problems because of themselves	1
34	76	Laws are not as important as others think	1
35	79	It is easy for me to work in a team	0
36	80	It is important to help people when they need it	0
37	81	I have punched someone or thought of doing it	1
38	82	If it was necessary I would pretend to be some...	1
39	84	After an argument I usually go over what happe...	0
40	92	It does not seem serious to me to have some debts	1
41	93	People steal because they have needs	1
42	97	People who commit crimes have their reasons fo...	1
43	99	It is important to respect turns	0

We proceed to identify the anomadic items present in the model present in this iteration.

In [326]:

```
for i in range(df.shape[1]-1):

    ## Local variable to indicate the index of the selected classes.
    i_index = 0

    for index in selection_class:

        ## Determine if it fails based on the expected response parameters of the
        if dm.loc[i, "Threshold "  + str(target_names[index])] == 1 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 1 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 0 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 0 and da.loc[i,
            dm.loc[i, "Anomaly "  + str(target_names[index])] = expected_response

        ## Increase of the index of the selected classes.
        i_index += 1

## The final grade data dataset is printed.
dm
```

Out[326]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	1	-0.059830	0	0	0.064008	1	0
1	2	-0.000434	NA	NA	-0.008779	NA	NA
2	6	-0.004093	NA	NA	-0.000412	NA	NA
3	8	0.003680	NA	NA	-0.003408	NA	NA
4	10	0.061062	1	1	0.023922	1	0
5	13	-0.113167	0	0	0.000994	NA	NA
6	16	-0.001265	NA	NA	0.000027	NA	NA
7	17	-0.005767	NA	NA	0.012354	1	0
8	20	0.010568	1	1	0.022650	1	0
9	22	-0.017375	0	1	-0.018091	0	0
10	28	-0.004061	NA	NA	-0.182532	0	0
11	31	0.001788	NA	NA	-0.010339	0	0
12	33	-0.058594	0	0	-0.011171	0	1
13	34	-0.121509	0	0	0.112861	1	0
14	35	0.022568	1	1	-0.008039	NA	NA
15	36	-0.006471	NA	NA	0.000525	NA	NA
16	37	-0.285420	0	1	-0.182475	0	0
17	39	0.032362	1	0	-0.047259	0	0
18	40	0.003534	NA	NA	0.000480	NA	NA
19	43	-0.021338	0	0	0.057640	1	0

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
20	47	0.151068	1	0	-0.198512	0	0
21	50	0.122161	1	1	-0.050008	0	1
22	52	-0.086237	0	0	0.028765	1	0
23	53	0.078599	1	1	0.114902	1	0
24	56	0.176395	1	0	-0.177860	0	0
25	57	-0.041006	0	0	0.037299	1	0
26	61	0.092091	1	1	0.027141	1	0
27	64	0.014939	1	0	-0.011734	0	0
28	65	-0.005018	NA	NA	0.015671	1	0
29	67	-0.085142	0	0	0.056442	1	0
30	69	-0.021778	0	0	0.026632	1	0
31	71	-0.001404	NA	NA	0.000678	NA	NA
32	73	-0.144125	0	0	0.007589	NA	NA
33	74	0.007009	NA	NA	-0.001137	NA	NA
34	76	-0.004313	NA	NA	0.002916	NA	NA
35	79	0.003616	NA	NA	-0.001199	NA	NA
36	80	0.134843	1	0	-0.083349	0	0
37	81	-0.216270	0	0	0.073278	1	0
38	82	-0.107243	0	0	0.093070	1	0
39	84	0.337728	1	0	-0.177842	0	0
40	92	-0.003063	NA	NA	0.005385	NA	NA
41	93	-0.000901	NA	NA	0.001172	NA	NA
42	97	-0.244370	0	0	0.275903	1	0
43	99	0.341218	1	0	-0.233854	0	0

Removing identified anomadic items from the model

We proceed to eliminate the identified anomadic items so as not to use them in the next iteration.

In [327]:

```
## Local variable for the elimination of the anomaly items.
instructions = ''
init = False
question_anomaly = []

if selection_mode == 1:

    ## Items that fail in the defined class are retrieved.
    dm_delete = dm[dm['Anomaly ' + str(target_names[select_class_mode])] == 1]
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 2:

    ## Items that fail in classes defined with the or condition type are retrieve
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 1"
            init = True
        else:
            instructions += " or `Anomaly " + str(target_names[index]) + "` == 1'

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 3:

    ## Items that fail in classes defined with the and condition type are retriev
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 1"
            init = True
        else:
            instructions += " and `Anomaly " + str(target_names[index]) + "` == 1

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 4:

    ## Out-of-threshold items are retrieved in classes defined with the conditior
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 'NA'"
            init = True
        else:
            instructions += " and `Anomaly " + str(target_names[index]) + "` == '

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

## The selected blank items are printed according to the selection_mode.
question_anomaly
```

Out[327]: [33, 50]

Model explainability process

The data used for the test of the neural network model is shown.

In [328]: X_test

Out[328]:

	1. Most of the time I have difficulty concentrating on simple tasks	2. I don't feel like doing my daily duties	6. I consider that my life is full of good things	8. I'm still as punctual as I have always been	10. I have found that I can spend a lot of time scrolling the screen \nof my cell phone without searching or stopping at anything in particular	13. I felt more encouraged to do my daily activities before	16. I think my thoughts are strange or different from before	17. I feel guilty about the decisions that I have made	20. diffi for m m decisi eve they sin
69	1	0	1	0	0	0	0	0	
46	1	0	1	1	1	1	1	1	
58	0	0	1	1	0	1	1	0	
114	1	0	1	0	1	1	1	0	
73	0	0	1	1	1	0	0	0	
98	1	0	0	1	1	0	1	0	
31	0	0	1	0	0	0	0	1	
53	1	0	1	1	1	1	1	1	
65	0	0	1	1	0	1	0	0	
96	0	0	1	1	1	0	0	1	
95	0	0	1	0	1	0	1	0	
97	0	0	1	1	0	0	0	1	
2	0	0	1	0	1	0	1	0	
62	0	1	1	0	1	0	1	0	
110	0	0	0	0	0	0	0	1	
55	0	1	1	1	0	0	1	1	
103	1	1	1	0	1	0	0	0	
100	1	1	1	0	0	1	0	0	
66	1	1	0	0	1	0	1	1	
44	0	0	1	1	0	0	0	0	
77	0	1	1	1	1	0	1	1	
17	1	0	1	1	1	1	1	0	
81	1	0	1	1	0	1	1	1	
74	1	1	0	1	1	1	1	1	

56	0	0	1	1	0	1	1	1
94	1	1	0	0	1	1	1	1
35	1	0	1	1	0	0	1	0
38	0	1	1	1	0	0	1	0
93	0	0	1	1	1	1	1	0
48	1	1	1	0	1	1	1	1
33	0	1	1	0	1	1	1	1
59	0	1	1	1	1	0	0	1
92	1	0	0	1	0	0	1	1
42	1	0	1	1	0	0	1	0
10	0	0	1	0	1	0	1	0
85	1	1	1	1	1	1	1	1
32	0	1	1	0	1	1	1	1
78	1	0	1	1	0	0	1	0
19	0	0	0	1	0	1	1	0

39 rows × 44 columns

Explanation of the model using ALE

In analysis of the model through the results obtained by ALE, it is possible to identify the behavior of the neural network.

In [329]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

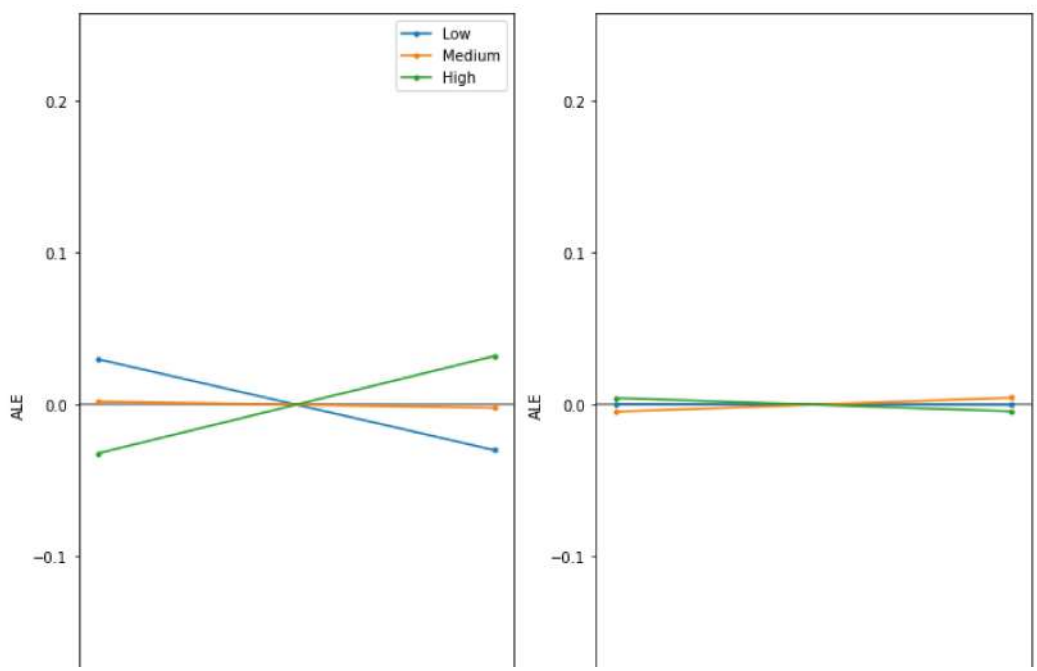
```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

The graphs of all the data present in the dataset used are shown.

In [330]:
```
plot_ale(proba_exp_lr, n_cols=2, features=list(range(df.shape[1]-1)),fig_kw={'fig
```



Explanation of the model using LIME

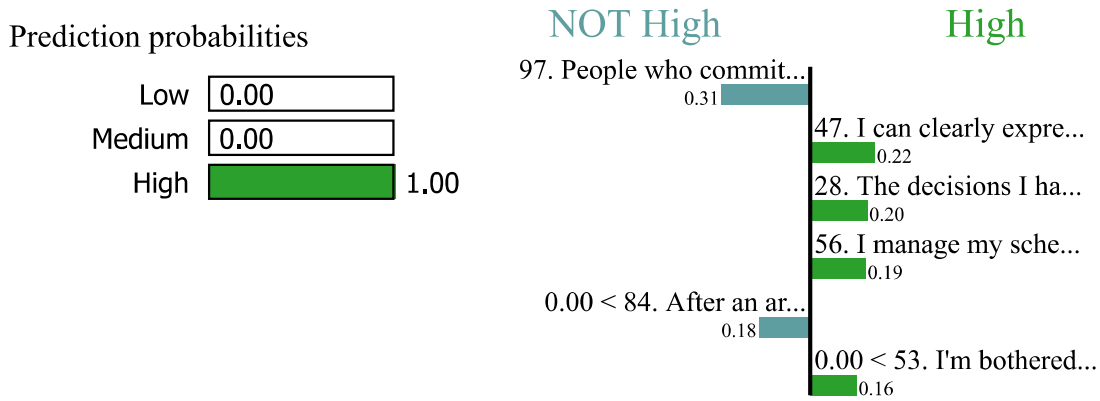
We proceed to explain the improved model using LIME for the Low, Medium and High classes.

In [331]:
```
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorica
                                                   feature_names=train_cols, clas
                                                   discretize_continuous=True)
```

The first individual present in the test data is explained.

In [332]:
```
exp = explainer.explain_instance(X_test.to_numpy()[0], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

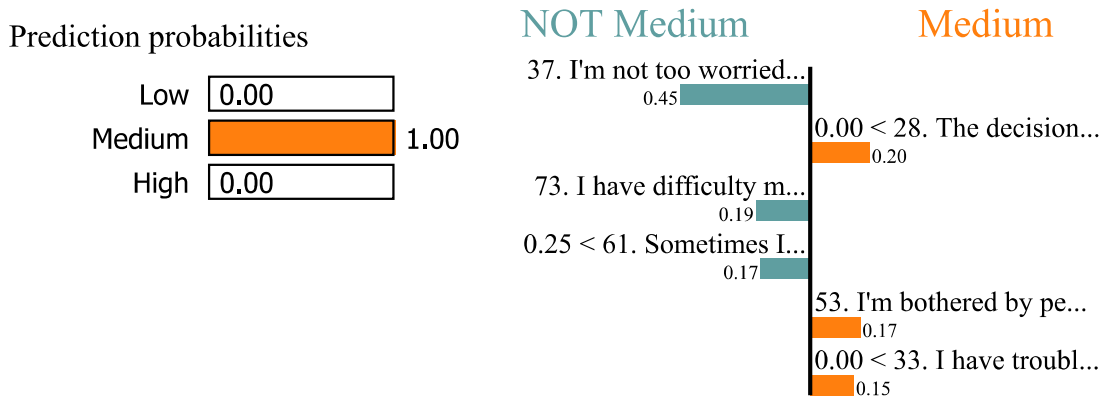


Feature	Value
97. People who commit crimes have their reasons for doing it	0.00
47. I can clearly express to others how I feel	0.00
28. The decisions I have made so far have been the right ones	0.00
56. I manage my schedule as I always have	0.00
84. After an argument I usually go over what happened in my head	1.00
53. I'm bothered by people walking slowly in front of me	1.00

The second individual present in the test data is explained.

In [333]:
```
exp = explainer.explain_instance(X_test.to_numpy()[1], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

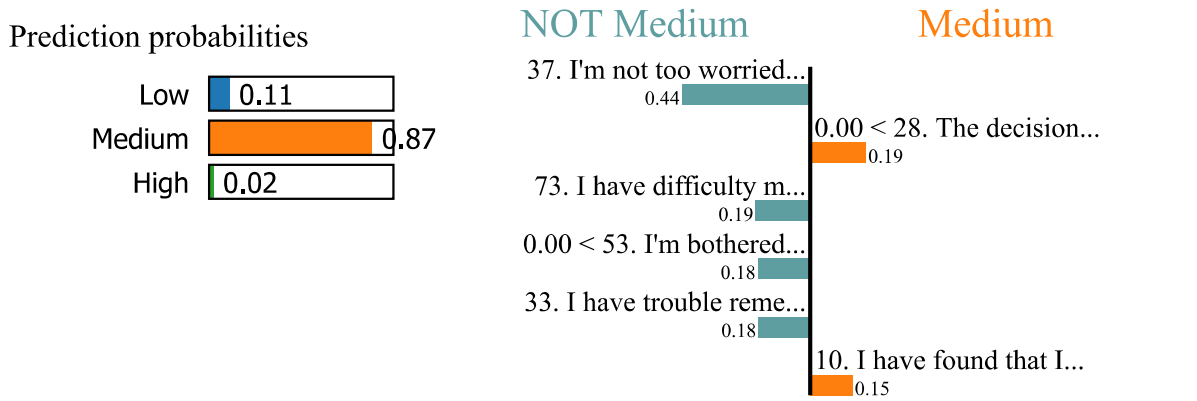


Feature	Value
37. I'm not too worried about what might happen in a few weeks	0.00
28. The decisions I have made so far have been the right ones	1.00
73. I have difficulty making elaborate plans	0.00
61. Sometimes I forget what I wanted to say because I have several thoughts at the same time	1.00
53. I'm bothered by people walking slowly in front of me	0.00
33. I have trouble remembering things easily	1.00

The last individual present in the test data is explained.

In [334]:
```
exp = explainer.explain_instance(X_test.to_numpy()[-1], mlp.predict_proba, num_fe
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```



Feature	Value
37. I'm not too worried about what might happen in a few weeks	0.00
28. The decisions I have made so far have been the right ones	1.00
73. I have difficulty making elaborate plans	0.00
53. I'm bothered by people walking slowly in front of me	1.00
33. I have trouble remembering things easily	0.00
10. I have found that I can spend a lot of time scrolling the screen of my cell phone without searching or stopping at anything in particular	0.00

Explanation of the model using SHAP

SHAP explanatory values are displayed according to the class to which the selected individual belongs according to the prediction of the model.

The first individual present in the test data is explained.

In [335]:
```
## Individual Selection.
individual = X_test.iloc[[0]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The second individual present in the data test is explained.

In [336]:
```
## Individual Selection.
individual = X_test.iloc[[1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The last individual present in the data test is explained.

In [337]:
```
## Individual Selection.
individual = X_test.iloc[[-1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

Explanation using heatmaps

The importance of the data present in the neural network is shown from explanation methods.

In [338]:

```
## Items currently in use are retrieved.
ds_item = ds[ds.Status == "In use"].reset_index(drop=True)

## The data is prepared for the generation of heatmaps.
column_Map = ds_item['Question'].tolist()
content_Map = [False for x in range(1, df.shape[1])]

## The next items to be deleted will be marked.
for index in range(0, len(content_Map)):

    if iteration == 0:
        for delete in list_contr:
            if column_Map[index] == delete:
                content_Map [index] = True
    else:
        for delete in question_anomaly:
            if column_Map[index] == delete:
                content_Map [index] = True

## The dataframe is created with the prepared data.
df_mask = pd.DataFrame([content_Map], columns = column_Map)
df_mask_imp = pd.concat([df_mask, df_mask, df_mask], ignore_index = True, axis =
df_mask_all = pd.concat([df_mask, df_mask, df_mask, df_mask], ignore_index = True

df_mask
```

Out[338]:

	1	2	6	8	10	13	16	17	20	22	...	76	79	80	8
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False

1 rows × 44 columns

Identification of feature variables

Identification of feature variables from the randomness of the answers as items of the data.

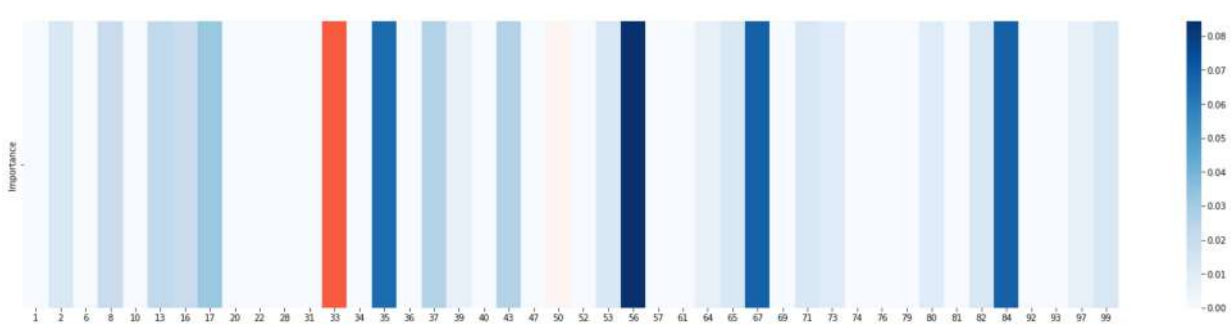
In [339]:
```
## Loading the data for explainability.
explainer = ClassifierExplainer(mlp, X_test, y_test)

## Class index local variable.
iterable_class = -1
map_class = []
```

```
WARNING: Parameter shap='guess', but failed to guess the type of shap explainer
to use for MLPClassifier. Defaulting to the model agnostic shap.KernelExplainer
(shap='kernel'). However this will be slow, so if your model is compatible with
e.g. shap.TreeExplainer or shap.LinearExplainer then pass shap='tree' or shap
='linear'!
WARNING: For shap='kernel', shap interaction values can unfortunately not be ca
lculated!
Note: shap values for shap='kernel' normally get calculated against X_backgroun
d, but paramater X_background=None, so setting X_background=shap.sample(X, 5
0)...
Generating self.shap_explainer = shap.KernelExplainer(model, X, link='identit
y')
```

Explainability of the data belonging to the first class, in this case the Low.

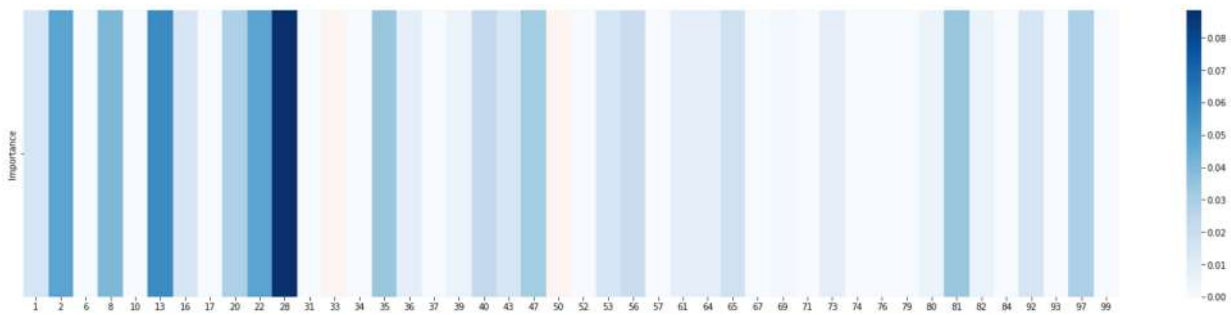
In [340]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

```
Calculating permutation importances (if slow, try setting n_jobs parameter)...
```



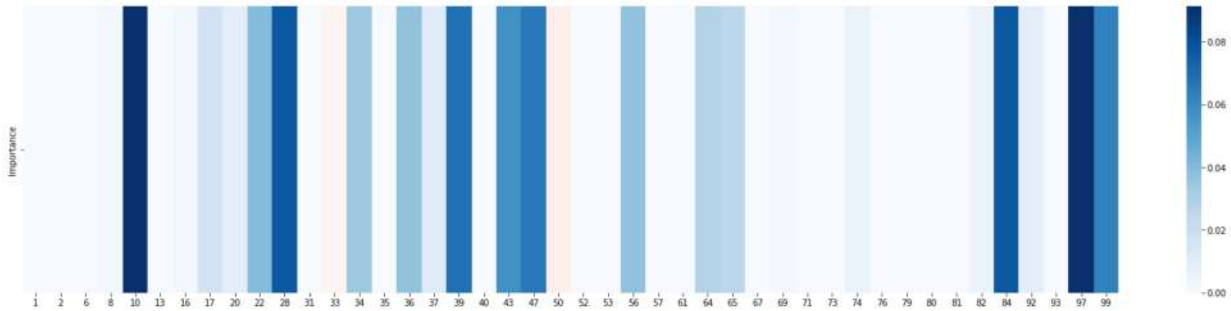
Explainability of the data belonging to the second class, in this case the Medium.

In [341]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



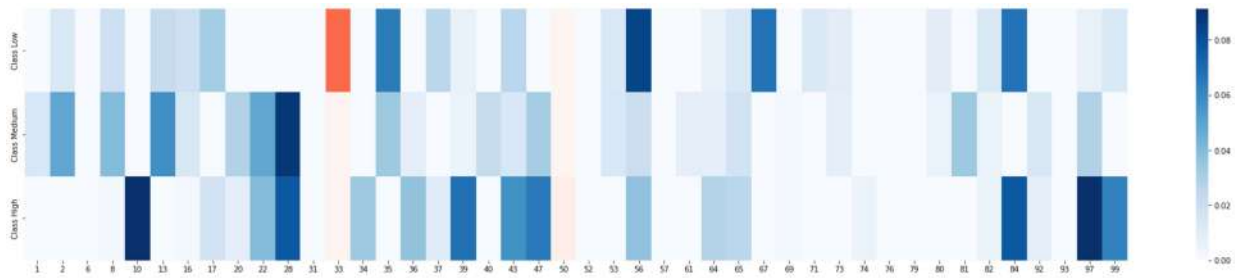
Explainability of the data belonging to the third class, in this case the High.

In [342]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [343]:
```
## Fusion of the previous heatmaps.
df_importance = pd.concat(map_class, axis=1)
df_importance.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_importance.transpose(), cmap = "Reds", cbar = False)
sns.heatmap(df_importance.transpose(), cmap = "Blues",yticklabels = True, xtickla
plt.show()
```



Identification of feature variables using SHAP

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of SHAP.

In [344]:
```
## Class index local variable.
iterable_class = -1
map_class = []
```

Individual explainability of the data belonging to the first class, in this case the Low.

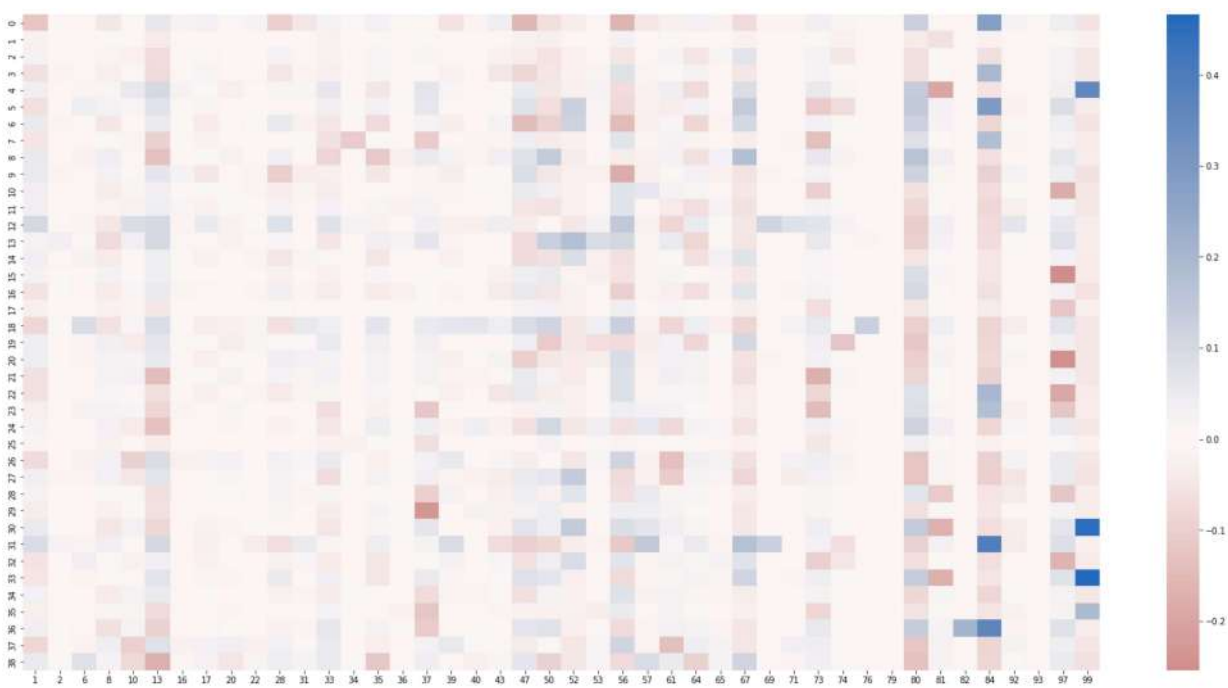
Definition of explainability data.

In [345]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

```
Calculating shap values...

  0%|          | 0/39 [00:00<?, ?it/s]
```

Generation of the heatmap.

In [346]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

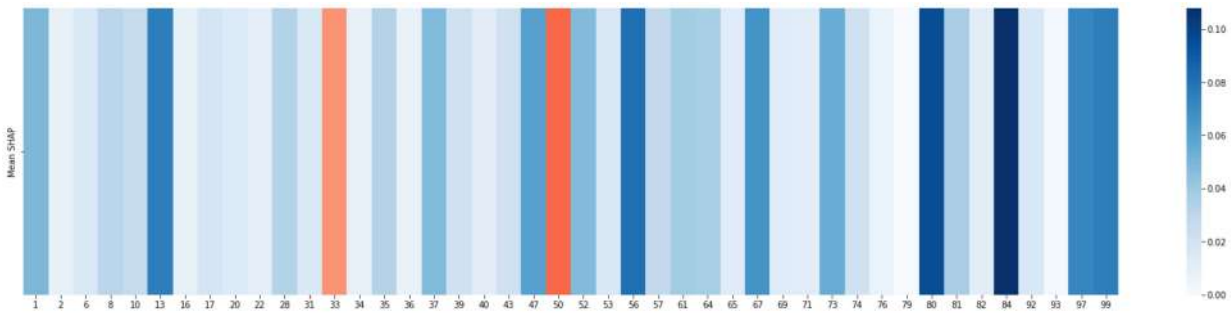


Fusion of the data used.

In [347]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [348]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



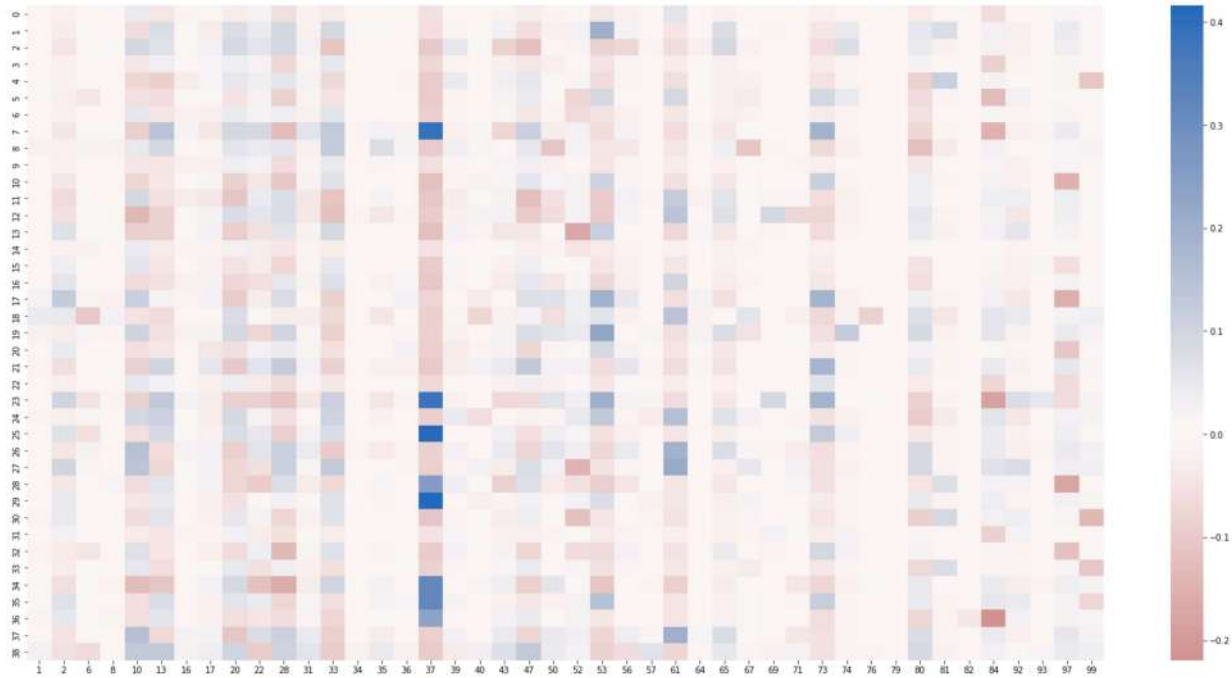
Individual explainability of the data belonging to the second class, in this case the Medium.

Definition of explainability data.

In [349]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [350]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

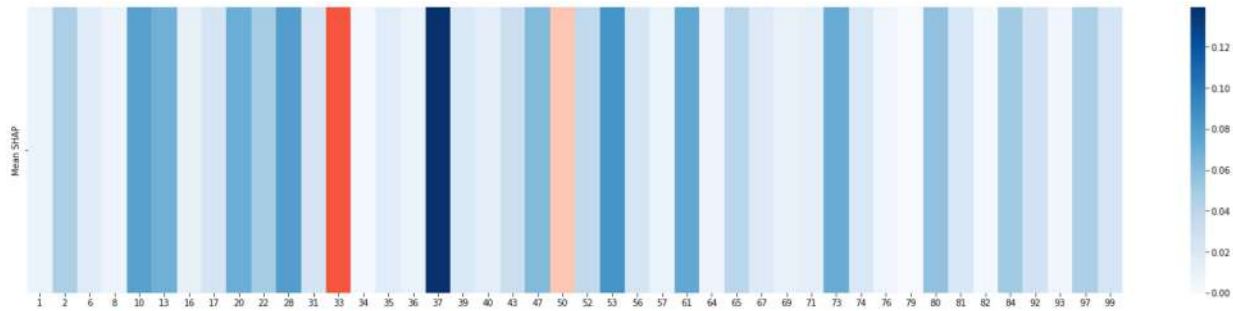


Fusion of the data used.

In [351]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [352]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



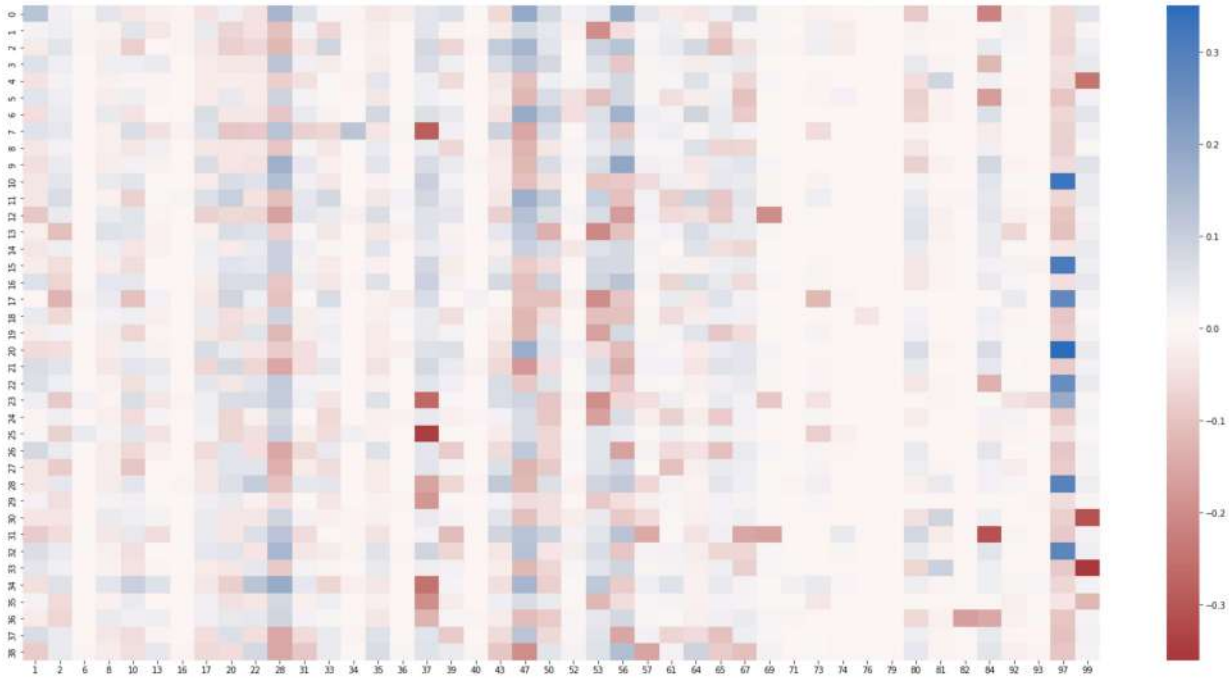
Individual explainability of data belonging to the third class, in this case High.

Definition of explainability data.

In [353]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [354]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

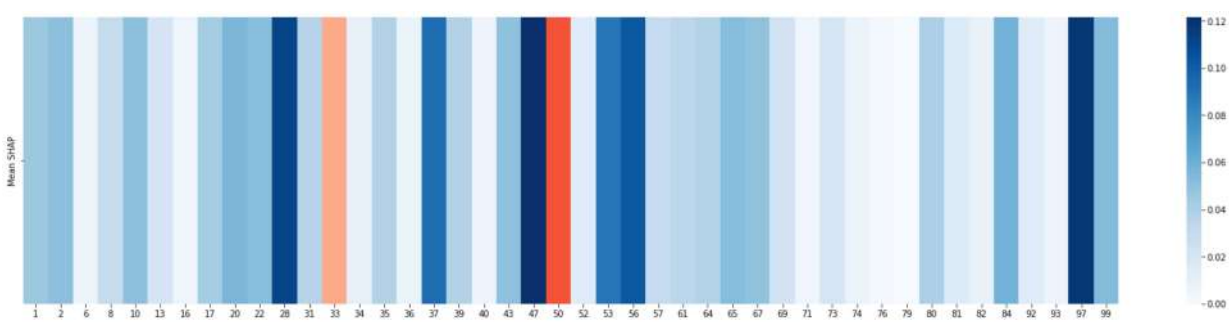


Fusion of the data used.

In [355]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

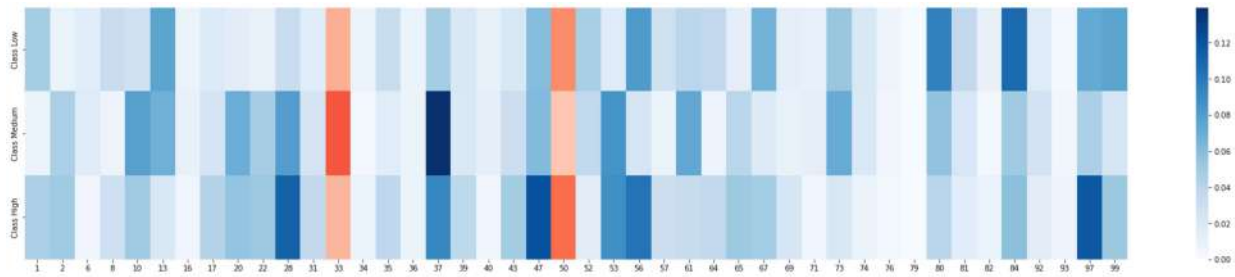
Generation of the heatmap.

In [356]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [357]:
```
## Fusion of the previous heatmaps.
df_shap = pd.concat(map_class, axis=1)
df_shap.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_shap.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_shap.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using LIME

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of LIME.

In [358]:
```
## Data preparation for explainability.
lime_exp = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorical
                                                  feature_names=train_cols, class
                                                  discretize_continuous=True)

## Class index local variable.
iterable_class = -1
map_class = []
```

Obtaining the data for the generation of the explanations by LIME about the Low, Medium and High classes.

In [359]:
```
## Local variables for LIME array data.
exp_matrix = [[] for _ in range(len(name_class))]

## Recovering the data needed for LIME.
for x in X_test.to_numpy():

    ## Auxiliary temporary variables.
    exp_list = [[] for _ in range(len(name_class))]

    ## Loading the data for explainability.
    exp = lime_exp.explain_instance(x, mlp.predict_proba, num_features = df.shape

    ## Recovering the data.
    for elements in range(0, len(name_class)):
        temp = exp.as_map()[elements]
        temp.sort(key=itemgetter(0))

        ## Saving data of the Low, Medium and High classes.
        for tup in temp:
            exp_list[elements].append(tup[1])

        exp_matrix[elements].append(exp_list[elements])
```

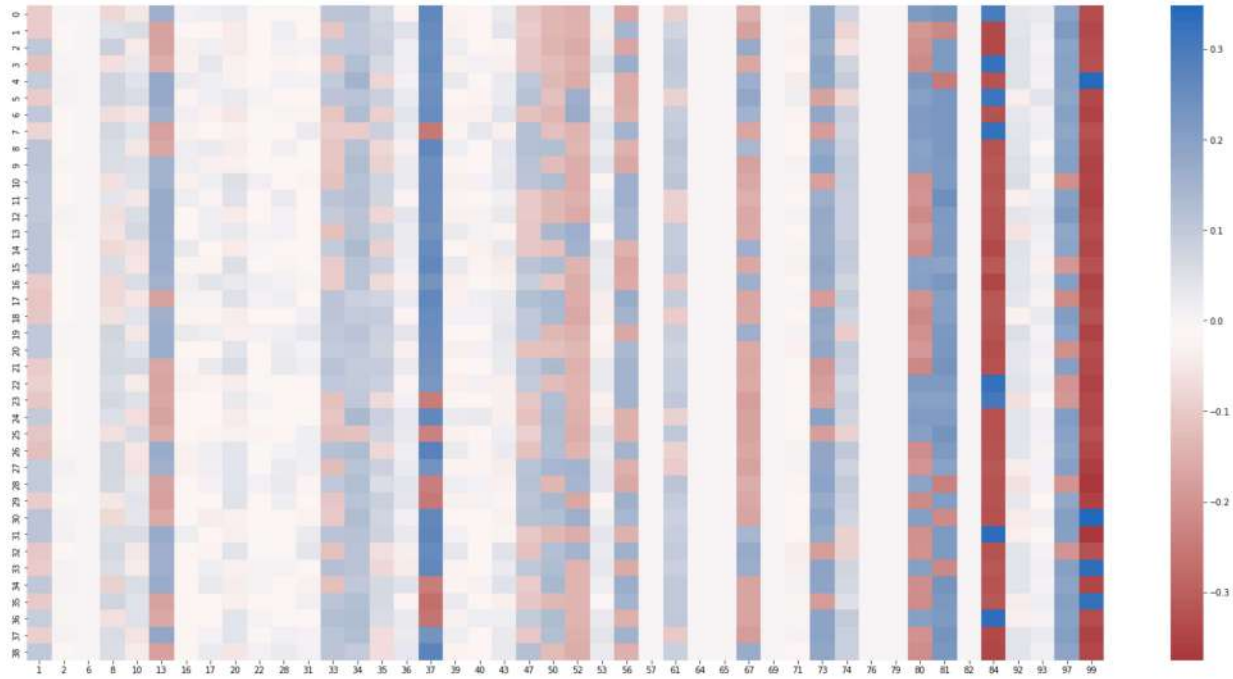
```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

Individual explainability of the data belonging to the first class, in this case the Low.

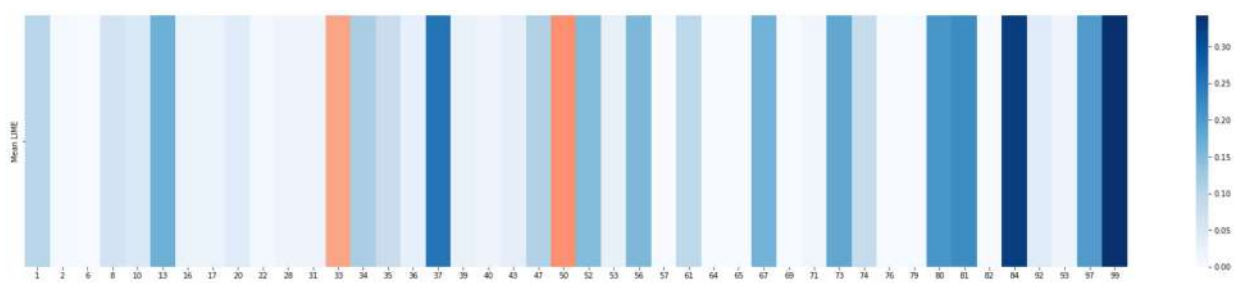
Generation of the heatmap.

In [360]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

In [361]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

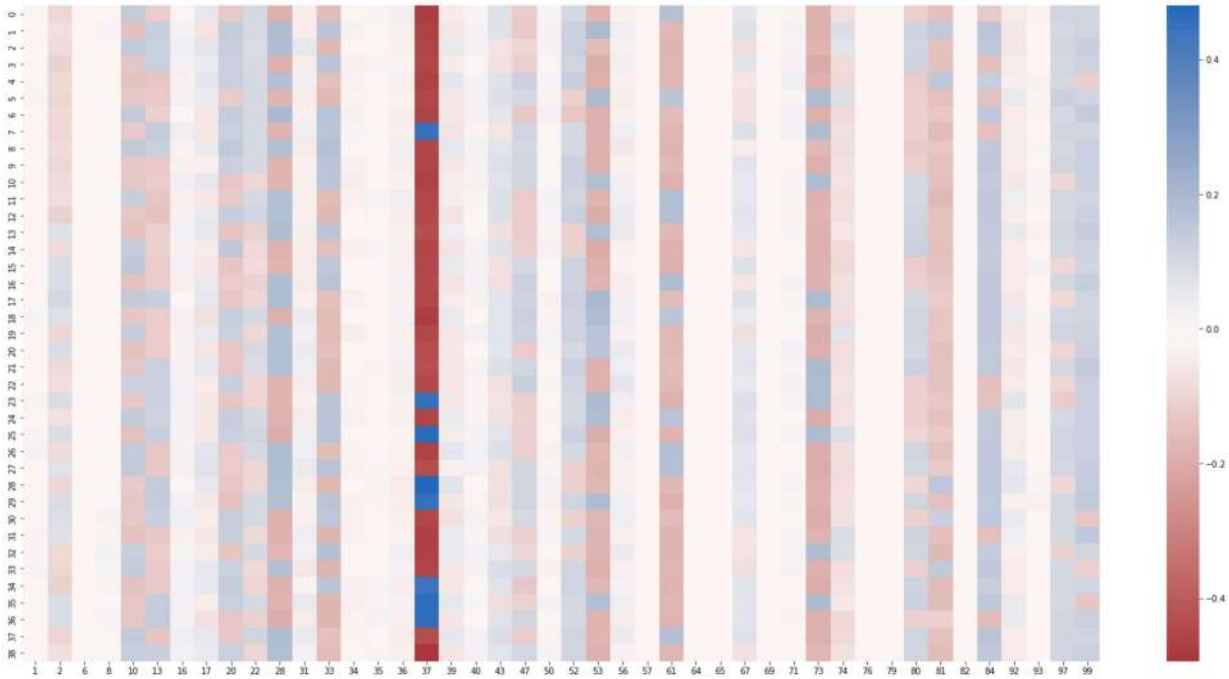


Individual explainability of the data belonging to the second class, in

this case the Medium.

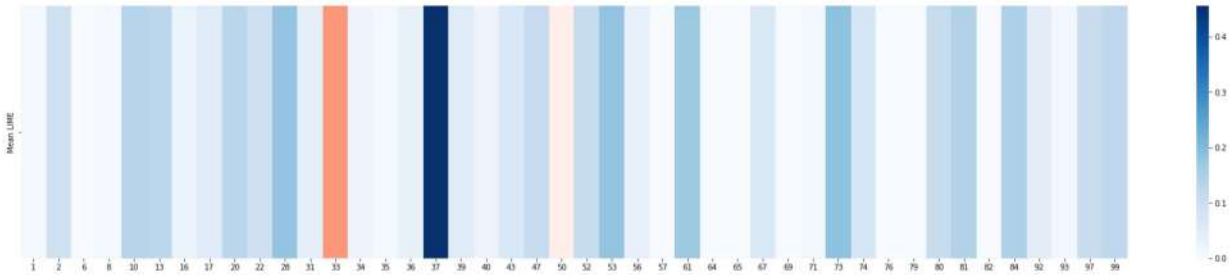
Generation of the heatmap.

In [362]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

In [363]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

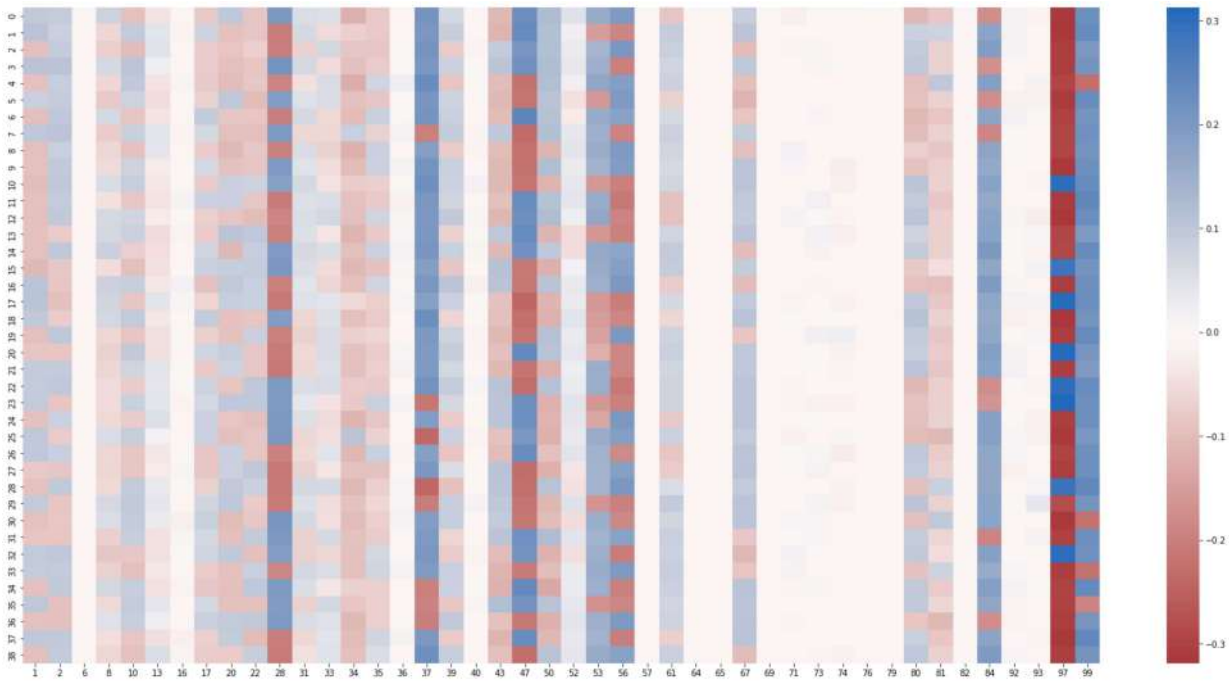


Individual explainability of data belonging to the third class, in this

case High.

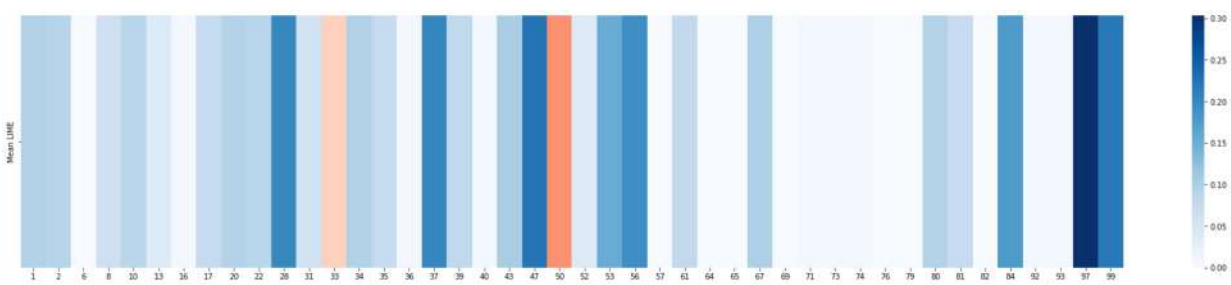
Generation of the heatmap.

In [364]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



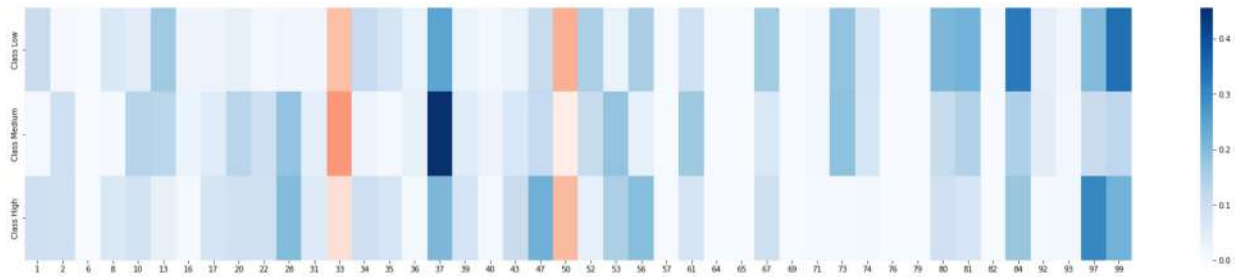
Fusion of the data used.

In [365]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [366]:
```
df_lime = pd.concat(map_class, axis=1)
df_lime.columns = name_class
plt.figure(figsize = (32,6))
sns.heatmap(df_lime.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_lime.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using ALE

It shows the main effect of each question compared to the probability that the model predicts in each class. This effect is shown for both responses. Since there are only 2 answers, the map is the same but with the colors reversed.

In [367]:
```
## Definition of explainability data.
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names=train_cols, target_names= target_r
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())

## Class index local variable.
iterable_class = -1
map_class = []
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

Obtaining the data for the generation of the explanations by ALE about all the classes.

In [368]:
```
## Local variables for ALE array data.
ale_list = [[] for _ in range(len(name_class))]

## Recovering the data needed for ALE.
for array in proba_exp_lr.ale_values:

    ## Recovering the data.
    for elements in range(0, len(name_class)):

        ## Saving data of all the classes.
        ale_list[elements].append(array[0][elements])
```

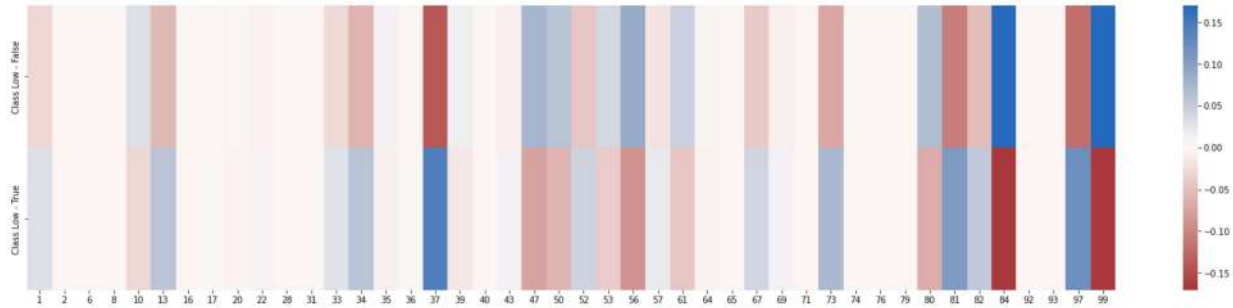
Data processing for the explainability of ALE.

In [369]:
```
## Data processing of the Low, Medium and High classes.
for elements in range(0, len(name_class)):
    ale_df = pd.DataFrame([ale_list [elements]])
    ale_df = pd.concat([ale_df.multiply(-1), ale_df])
    ale_df.index = [name_class [elements] + " - False", name_class [elements] + '
    ale_df.columns = column_Map
    map_class.append(ale_df)
```

Explainability of the data belonging to the first class, in this case the Low.

Generation of the heatmap.

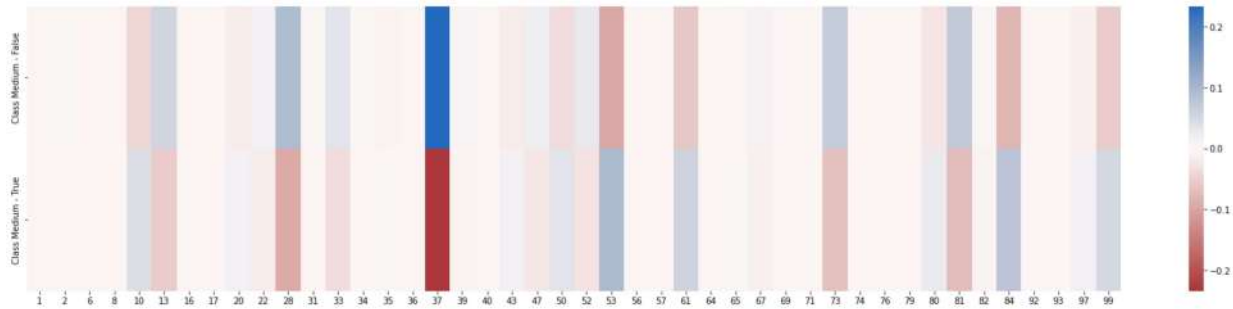
In [370]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the second class, in this case the Medium.

Generation of the heatmap.

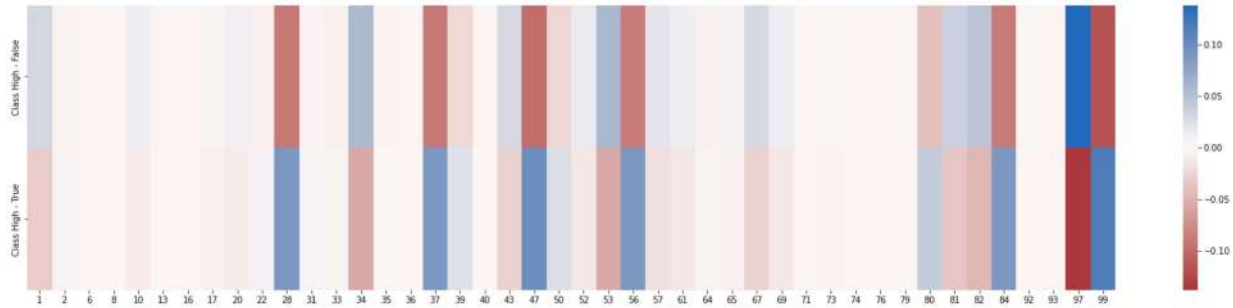
In [371]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the third class, in this case the High.

Generation of the heatmap.

In [372]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```

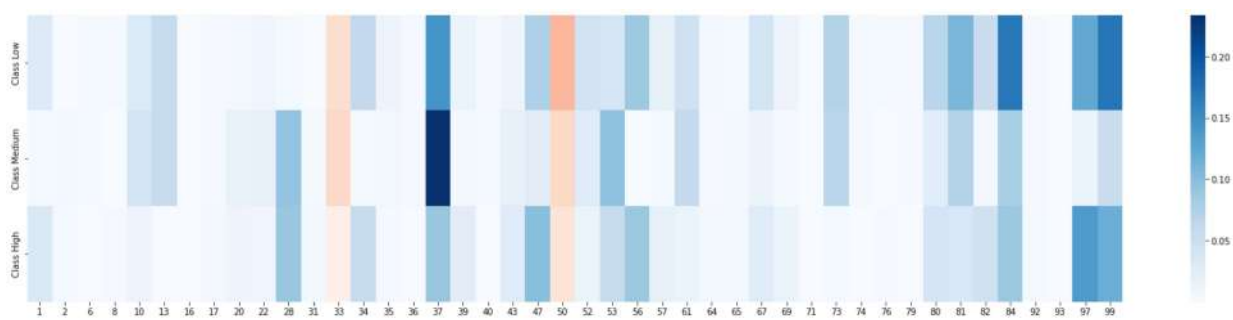


Fusion of the data used in all classes.

In [373]:
```
## Application of absolute value to data.
for index in range(0, len(map_class)):
    map_class [index] = map_class [index][0:1].abs()

## Union of the heatmaps of the Low, Medium and High class.
ale_df = pd.concat(map_class)
ale_df.index = name_class
ale_df = ale_df.transpose()
```

Generation of the heatmap.

In [374]:
```
plt.figure(figsize = (28,6))
sns.heatmap(ale_df.transpose(), cmap="Reds", cbar = False)
sns.heatmap(ale_df.transpose(), cmap="Blues", yticklabels = True, xticklabels = T
plt.show()
```



Comparing of Feature Variables Methods for Explainability

The global values of each of the methods used are taken, their maximums are obtained (the minimums are 0) and a percentage is calculated based on said maximum, which indicates how relevant the contribution is for each question.

In [375]:
```
## Class index local variable.
iterable_class = -1
```

Comparison of the explanation methods of the first class, in this case the Low.

The highest data of each method used is obtained.

In [376]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [377]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

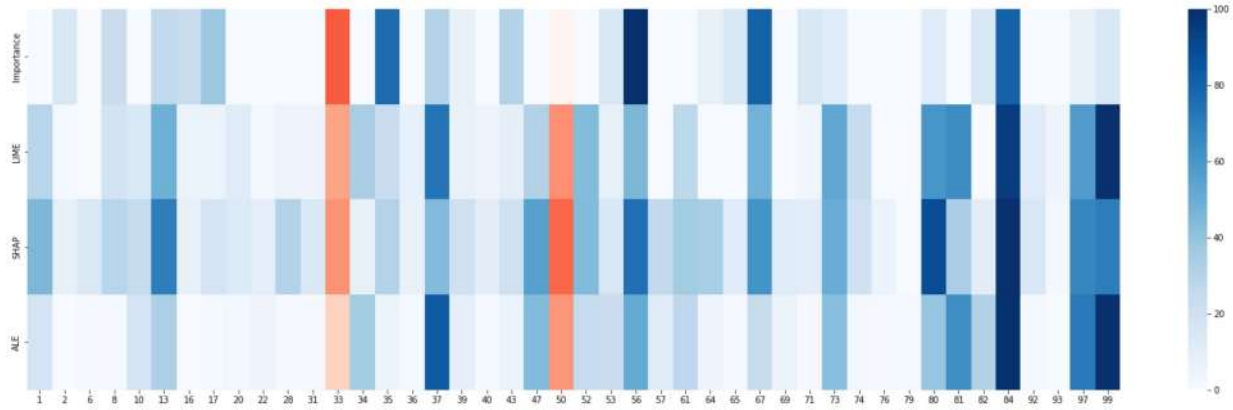
Out[377]:

	1	2	6	8	10	13	16	17
Importance	0.000000	15.384615	0.000000	23.076923	0.000000	26.923077	23.076923	38.461538
LIME	29.653445	1.803152	0.000000	18.597909	14.440750	48.866826	6.133789	6.273103
SHAP	45.582948	8.199226	14.429205	29.237485	24.836437	70.025912	7.760702	16.918431
ALE	17.534272	0.127091	1.199430	1.078407	17.895170	33.165480	0.370801	1.690174

4 rows × 44 columns

Generation of the heatmap.

In [378]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the second class, in this case the Medium.

The highest data of each method used is obtained.

In [379]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [380]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

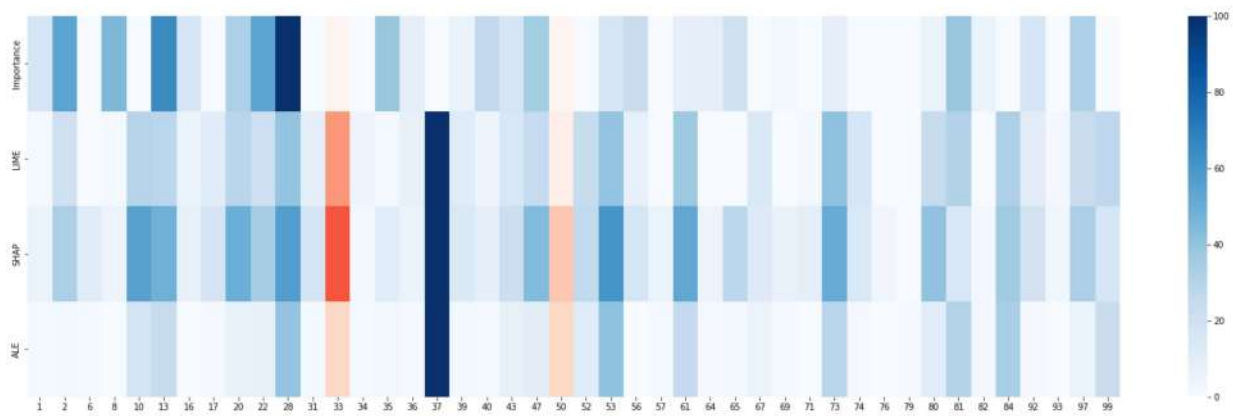
Out[380]:

	1	2	6	8	10	13	16	17
Importance	16.129032	54.838710	0.000000	45.161290	0.000000	64.516129	16.129032	0.000000
LIME	1.858937	19.314941	0.000000	1.679446	29.931984	27.923566	5.871128	12.091495
SHAP	5.699013	33.283543	12.343473	5.108149	55.679992	48.793418	8.127481	17.519444
ALE	0.892802	1.968990	0.962721	0.058179	18.163038	23.973833	0.264674	1.407661

4 rows × 44 columns

Generation of the heatmap.

In [381]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the third class, in this case the High.

The highest data of each method used is obtained.

In [382]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [383]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

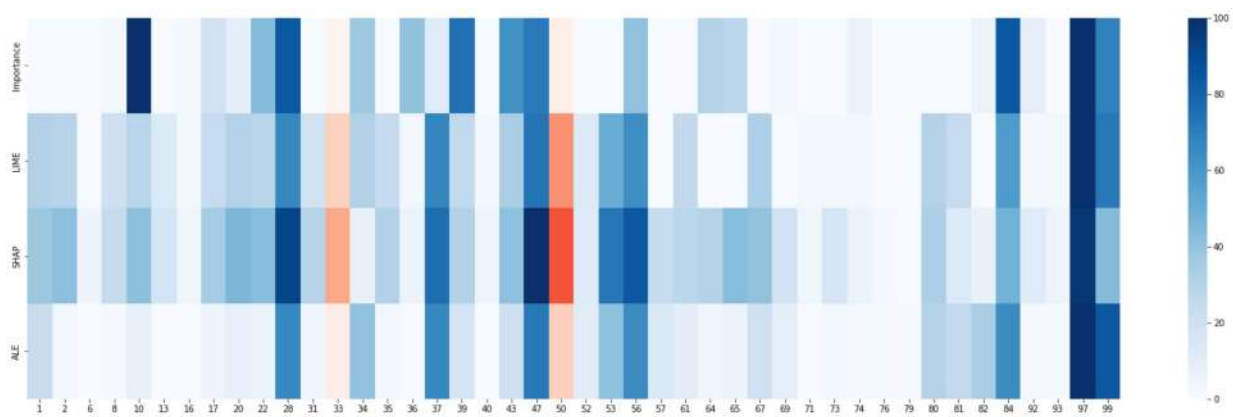
Out[383]:

	1	2	6	8	10	13	16	17
Importance	0.000000	0.000000	0.000000	3.125000	100.000000	0.000000	3.125000	18.750000
LIME	31.320554	29.784682	0.000000	20.827355	28.664183	13.179077	2.602943	25.233723
SHAP	38.409266	41.992917	4.360473	24.341461	42.178003	17.269958	4.059900	34.702733
ALE	23.199248	3.181962	0.149271	1.235036	8.670522	0.360428	0.009730	4.477491

4 rows × 44 columns

Generation of the heatmap.

In [384]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Safeguarding the items eliminated during this iteration.

In [385]:
```
## Removal of anomalous items selected by selection_mode.
for x in question_anomaly:

    ## Status change of the deleted items in the respective auxiliary dataset.
    ds.loc[x - 1,'Status'] = 'Delete'

## Save the changed statuses in the dataset.
ds.to_csv(path_dataset_qs, index = False)
```

Iteration 5

The first iteration of improvement of the model is carried out.

Preparing the data for the neural network

The data is prepared in order to be processed by the neural network.

In [386]:

```
## The iteration number is increased.
iteration += 1

## Loading data from the dataset.
df = pd.read_csv(path_dataset)
ds = pd.read_csv(path_dataset_qs)

## Elimination of unrelated items from the dataset.
if len(unrelated_questions) != 0:
    df = df.drop(df.columns[unrelated_questions[0]:unrelated_questions[1]], axis=

i = 0
if iteration == 1:
    ## Elimination of list of conflicting items declared in global variables.
    for x in list_contr:
        df.drop(df.columns[x - (i + 1)], axis = 1, inplace = True)
        i += 1
        ## Status change of the deleted question in the respective auxiliary data
        ds.loc[x - 1,'Status'] = 'Delete'

else:
    ## Elimination of the list of anomadic items defined in the improvement proce
    ds_delete = ds[ds.Status == "Delete"].reset_index(drop=True)
    for x in ds_delete['Question'].tolist():
        df.drop(df.columns[x - (i + 1)], axis = 1, inplace = True)
        i += 1

## Grouping of the target according to the defined dictionary.
if grouping_name != None and grouping_name != '':
    df[grouping_name] = df[grouping_name].map(class_dic)

## Assignment of local variables according to the data necessary for the neural r
train_cols = df.columns [0:-1]
label = df.columns [-1]
X = df [train_cols]
y = df [label]

## The prepared data of the dataset is printed.
df
```

Out[386]:

	1. Most of the time I have difficulty concentrating on simple tasks	2. I don't feel like doing my daily duties	6. I consider that my life is full of good things	8. I'm still as punctual as I have always been	10. I have found that I can spend a lot of time scrolling the screen \nof my cell phone without searching or stopping at anything in particular	13. I felt more encouraged to do my daily activities before	16. I think my thoughts are strange or different from before	17. I feel guilty about the decisions that I have made
0	1	1	1	1	0	0	0	0
1	1	0	1	1	0	1	1	1
2	0	0	1	0	1	0	1	0
3	0	0	1	1	0	1	0	0
4	1	1	1	1	1	0	1	1
...	...	...	...	...	...	...	...	...
99	1	1	1	1	1	1	0	0
100	1	1	1	0	0	1	0	0
101	0	0	1	0	1	1	1	0
102	1	0	1	1	1	0	0	1
103	1	1	1	0	1	0	0	0

104 rows × 43 columns

Oversample application

An oversample process is applied to level the amount of existing data given by the grouping.

In [387]:
```
## The number of elements present in the grouped target is printed.
y.value_counts()
```

Out[387]:
```
0    39
1    35
2    30
Name: Target, dtype: int64
```

Oversample application.

In [388]:
```
## Application of the oversample to the prepared data.
X, y = oversample.fit_resample(X, y)
```

Printing of the data after the application of the oversample.

In [389]:
```
## The results obtained after applying the oversample are printed.
y.value_counts()
```

Out[389]:
```
0    39
1    39
2    39
Name: Target, dtype: int64
```

Preparation of the parameters of the neural network

We proceed to define the parameters necessary for the training of the neural network.

In [390]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
```

Neural network training

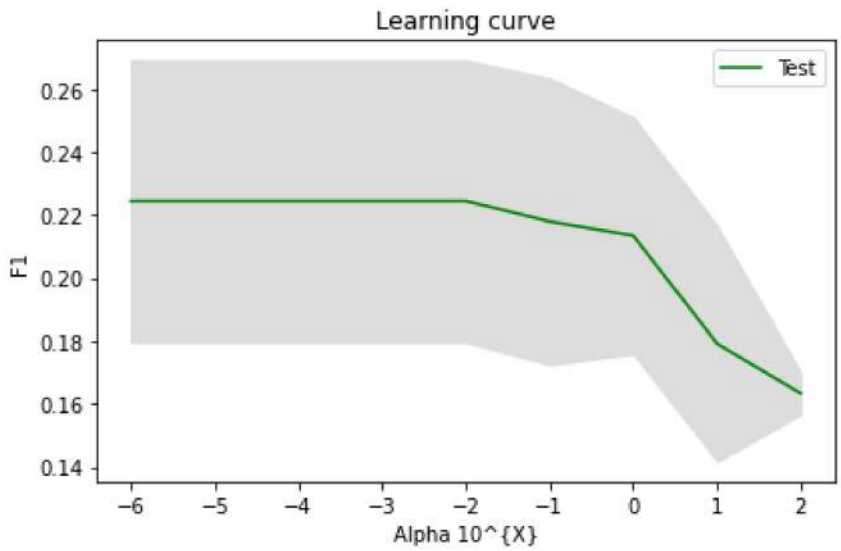
The alpha parameters and the initial learning rate are adjusted to obtain the best performance with the model using cross validation.

In [391]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_normal

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = 0.0001, max_iter =

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[391]:
```
([0.1632974910394265,
  0.17910044977511244,
  0.2134900489183255,
  0.21796835709129558,
  0.22441781406706837,
  0.22441781406706837,
  0.22441781406706837,
  0.22441781406706837,
  0.22441781406706837],
 [0.006619092595549127,
  0.03738199764605503,
  0.03759666129433013,
  0.045505150536351104,
  0.044780893408842184,
  0.044780893408842184,
  0.044780893408842184,
  0.044780893408842184,
  0.044780893408842184])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [392]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Alpha 10^{X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



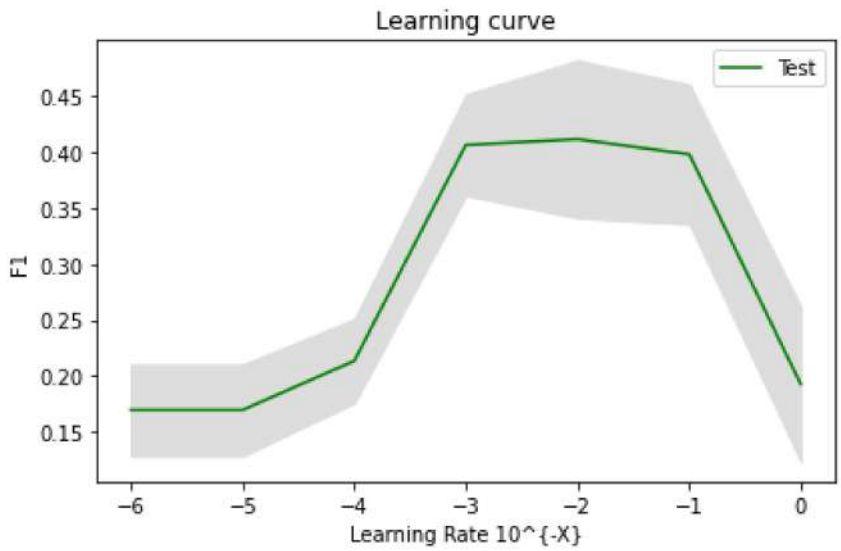
The training is performed again keeping the same parameters except for the alpha value that changes to 1.

In [393]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_alpha

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = regul_param, max_it

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[393]:
```
([0.1927316172477463,
  0.3981073284014461,
  0.4116582866737666,
  0.4064239028944911,
  0.2134900489183255,
  0.16947033054559935,
  0.16947033054559935],
 [0.06900121176816054,
  0.06226522543522221,
  0.07041327278291896,
  0.0451835151409837,
  0.03759666129433013,
  0.040956253005396466,
  0.040956253005396466])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [394]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Learning Rate 10^{-X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



Generation of the neural network model

The parameters of the final model are modified according to the data obtained in the training of the neural network.

In [395]:
```
## The value of the alpha parameter, learning_rate_init and random_state are char
## to the data in the global variables.
mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solver =
                    learning_rate = 'constant', learning_rate_init = lear
                    random_state = random_state_model)

## We print the data of the final generated model.
mlp.fit(X_train, y_train)
```

Out[395]:
```
MLPClassifier
MLPClassifier(activation='logistic', alpha=0.001, hidden_layer_sizes=(10,),
              learning_rate_init=0.1, max_iter=100000, random_state=42)
```

We safeguard the final generated model

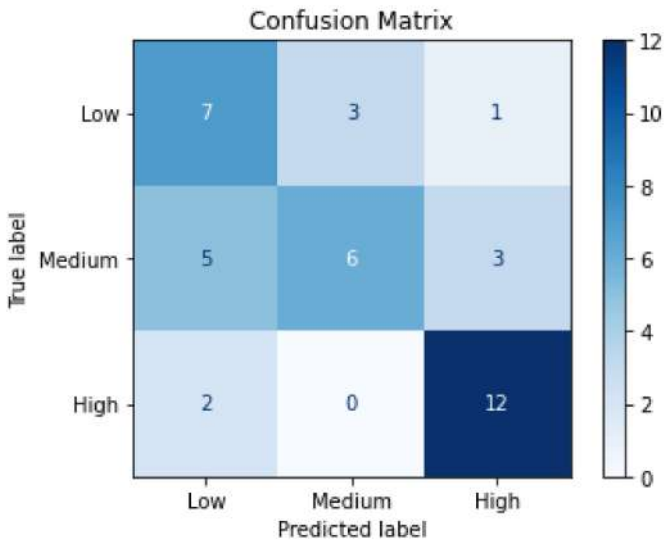
In [396]:
```
joblib.dump(mlp,"model_depression_i" + str(iteration) + ".pkl")
```

Out[396]:
```
['model_depression_i5.pkl']
```

Generation of the confusion matrix of the model

We obtain the statistics of the final model to generate its confusion matrix.

In [397]:
```
## The data for the generation of the confession matrix are defined.
confusion_matrix = ConfusionMatrixDisplay.from_estimator(mlp, X_test, y_test, dis
                                                         cmap = plt.cm.Blues)
confusion_matrix.ax_.set_title("Confusion Matrix")
plt.show()
```



We get the detailed statistics of the final model.

In [398]:
```
## The necessary detailed prediction is generated.
y_pred = mlp.predict(X_test)

## The detailed statistics of the model are printed.
print('Classification accuracy =',accuracy_score(y_test,y_pred) * 100,'%\n')
print(classification_report(y_test,y_pred))
```

```
Classification accuracy = 64.1025641025641 %

              precision    recall  f1-score   support

           0       0.50      0.64      0.56        11
           1       0.67      0.43      0.52        14
           2       0.75      0.86      0.80        14

    accuracy                           0.64        39
   macro avg       0.64      0.64      0.63        39
weighted avg       0.65      0.64      0.63        39
```

The data obtained is saved for later graphing.

In [399]:
```
accuracy = accuracy_score(y_test,y_pred) * 100
clf_rep = precision_recall_fscore_support(y_test,y_pred)

## The number of items used in the iteration is retrieved.
ds_delete = ds[ds.Status == "Delete"]
ds_delete.reset_index(inplace=True, drop=True)

## All the metrics of the confusion matrix are obtained.
metrics = [",".join(map(str, ds_delete['Question'].tolist())),
           accuracy]

for index in range(0, len(name_class)):
    metrics.append(clf_rep[0][index])
    metrics.append(clf_rep[1][index])
    metrics.append(clf_rep[2][index])
    metrics.append(clf_rep[3][index])

## The names of the columns of the dataset are defined.
columns = ['Question', 'Acurracy global']

for index in range(0, len(name_class)):
    columns.append('Precision_' + str(index))
    columns.append('Recall_' + str(index))
    columns.append('F1_score_' + str(index))
    columns.append('Support_' + str(index))

## A new row of the dataset is generated with all the data.
data_metrics = pd.DataFrame([metrics], columns = columns)

## The data is saved to the dataset.
data_metrics.to_csv(path_dataset_qd, mode = 'a', header = False, index = False)

## The data added to the dataset is printed.
data_metrics
```

Out[399]:

	Question	Acurracy global	Precision_0	Recall_0	F1_score_0	Supp
0	3,4,5,7,9,11,12,14,15,18,19,21,23,24,25,26,27,...	64.102564	0.5	0.636364	0.56	

Improvement process of the neural network model.

The application of the explanation method by ALE allows to determine the behavior of the neural network and thereby improve it by discarding non-significant data for it.

Slope Analysis of ALE Plots

In [400]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

We proceed to obtain the graphs to obtain the slopes of the selected classes.

In [401]:
```
## The question metrics dataset is loaded.
dm = pd.DataFrame()

## The number of items used in the iteration is retrieved.
ds_item = ds[ds.Status == "In use"]
ds_item.reset_index(inplace=True, drop=True)

## We get the slopes from the ALE data.
for i in range(df.shape[1]-1):
    structure_data = {'Question': ds_item.loc[i, "Question"]}

    for index in selection_class:
        slope = proba_exp_lr.ale_values [i][1][index] - proba_exp_lr.ale_values [
        structure_data['Slope ' + str(target_names[index])] = slope
        structure_data['Threshold ' + str(target_names[index])] = 'NA'
        structure_data['Anomaly ' + str(target_names[index])] = 'NA'

    dm = dm.append(structure_data, ignore_index=True)

## The current rating data dataset is printed.
dm
```

```
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a fu
```

Slope analysis and threshold application

The slope obtained is analyzed and those that are above or below the positive and negative

threshold are selected.

In [402]:

```
for i in range(df.shape[1]-1):
    for index in selection_class:

        ## The slopes are selected according to the values of the thresholds of t
        if dm.loc[i, "Slope " + str(target_names[index])] >= positive_threshold:
            dm.loc[i, "Threshold " + str(target_names[index])] = 1
        elif dm.loc[i, "Slope " + str(target_names[index])] <= negative_threshold
            dm.loc[i, "Threshold " + str(target_names[index])] = 0

## The current rating data dataset is printed.
dm
```

Out[402]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	1	-0.138471	0	NA	0.139419	1	NA
1	2	-0.014418	0	NA	-0.058997	0	NA
2	6	-0.035443	0	NA	-0.001914	NA	NA
3	8	0.030353	1	NA	-0.008966	NA	NA
4	10	-0.003401	NA	NA	0.007547	NA	NA
5	13	-0.184835	0	NA	0.016001	1	NA
6	16	-0.001500	NA	NA	-0.001485	NA	NA
7	17	-0.073803	0	NA	0.072450	1	NA
8	20	0.023498	1	NA	0.105458	1	NA
9	22	0.126786	1	NA	-0.149254	0	NA
10	28	0.046047	1	NA	-0.271337	0	NA
11	31	0.022856	1	NA	-0.021373	0	NA
12	34	-0.209418	0	NA	0.176432	1	NA
13	35	0.014491	1	NA	0.036310	1	NA
14	36	-0.057434	0	NA	-0.017369	0	NA
15	37	-0.188355	0	NA	-0.167515	0	NA
16	39	-0.044542	0	NA	-0.075195	0	NA
17	40	0.008666	NA	NA	0.001338	NA	NA
18	43	-0.076150	0	NA	0.079928	1	NA
19	47	0.084578	1	NA	-0.216880	0	NA
20	52	-0.134024	0	NA	0.029649	1	NA
21	53	-0.042884	0	NA	0.065600	1	NA
22	56	0.227314	1	NA	-0.182310	0	NA
23	57	-0.021442	0	NA	0.010315	1	NA
24	61	0.220355	1	NA	0.057112	1	NA
25	64	0.002489	NA	NA	-0.022714	0	NA
26	65	-0.085756	0	NA	0.122335	1	NA

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
27	67	-0.000983	NA	NA	0.000060	NA	NA
28	69	0.016604	1	NA	0.018549	1	NA
29	71	0.006508	NA	NA	0.001896	NA	NA
30	73	-0.301636	0	NA	0.074206	1	NA
31	74	0.021063	1	NA	0.068859	1	NA
32	76	-0.045770	0	NA	0.032442	1	NA
33	79	0.022489	1	NA	-0.002098	NA	NA
34	80	0.347419	1	NA	-0.118070	0	NA
35	81	-0.239696	0	NA	0.095886	1	NA
36	82	-0.047219	0	NA	0.053466	1	NA
37	84	0.245807	1	NA	-0.196578	0	NA
38	92	0.002754	NA	NA	-0.001976	NA	NA
39	93	-0.020514	0	NA	-0.052821	0	NA
40	97	-0.185595	0	NA	0.312015	1	NA
41	99	0.249844	1	NA	-0.201260	0	NA

Analysis and determination of anomalous items

The anomalous items present in the model are determined based on their slope and expected response from the expert.

In [403]:

```
## The dataset of expected responses is loaded.
da = pd.read_csv(path_dataset_qa)

## Delete list of items deleted in this iteration.
i = 0
for x in ds_delete['Question']:
    da.drop(da.index[x - (i + 1)], axis = 0, inplace = True)
    i += 1

## Indexes are restored for later use.
da.reset_index(inplace=True, drop=True)

## The dataset of expected responses is printed.
da
```

Out[403]:

	Number	Question	RE
0	1	Most of the time I have difficulty concentrati...	1
1	2	I don't feel like doing my daily duties	1
2	6	I consider that my life is full of good things	0
3	8	I'm still as punctual as I have always been	0
4	10	I have found that I can spend a lot of time sc...	1
5	13	I felt more encouraged to do my daily activiti...	1
6	16	I think my thoughts are strange or different f...	1
7	17	I feel guilty about the decisions that I have ...	1
8	20	It is difficult for me to make decisions even ...	1
9	22	It is not difficult for me to understand somet...	0
10	28	The decisions I have made so far have been the...	0
11	31	It is easy for me to choose a photograph of my...	0
12	34	In recent months I have had discussions with m...	1
13	35	I constantly imagine that something will go wr...	1
14	36	I am afraid of being worng when doing my homework	1
15	37	I'm not too worried about what might happen in...	0
16	39	Everything will be alright	0
17	40	I can easily blank my mind	0
18	43	Sometimes i feel trapped	1
19	47	I can clearly express to others how I feel	0
20	52	Being locked in an elevator would be the worst...	1
21	53	I'm bothered by people walking slowly in front...	1
22	56	I manage my schedule as I always have	0
23	57	It bothers me to feel that people on the stree...	1
24	61	Sometimes I forget what I wanted to say becaus...	1
25	64	I can put down my cell phone and dedicate myse...	0

	Number	Question	RE
26	65	I worry that people will not understand what I...	1
27	67	I get angry easily	1
28	69	It is not important for me to meet set dates	1
29	71	I use lies just to get out of certain problems	1
30	73	I have difficulty making elaborate plans	1
31	74	People have problems because of themselves	1
32	76	Laws are not as important as others think	1
33	79	It is easy for me to work in a team	0
34	80	It is important to help people when they need it	0
35	81	I have punched someone or thought of doing it	1
36	82	If it was necessary I would pretend to be some...	1
37	84	After an argument I usually go over what happe...	0
38	92	It does not seem serious to me to have some debts	1
39	93	People steal because they have needs	1
40	97	People who commit crimes have their reasons fo...	1
41	99	It is important to respect turns	0

We proceed to identify the anomadic items present in the model present in this iteration.

In [404]:

```
for i in range(df.shape[1]-1):

    ## Local variable to indicate the index of the selected classes.
    i_index = 0

    for index in selection_class:

        ## Determine if it fails based on the expected response parameters of the
        if dm.loc[i, "Threshold "  + str(target_names[index])] == 1 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 1 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 0 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 0 and da.loc[i,
            dm.loc[i, "Anomaly "  + str(target_names[index])] = expected_response

        ## Increase of the index of the selected classes.
        i_index += 1

## The final grade data dataset is printed.
dm
```

Out[404]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	1	-0.138471	0	0	0.139419	1	0
1	2	-0.014418	0	0	-0.058997	0	1
2	6	-0.035443	0	1	-0.001914	NA	NA
3	8	0.030353	1	0	-0.008966	NA	NA
4	10	-0.003401	NA	NA	0.007547	NA	NA
5	13	-0.184835	0	0	0.016001	1	0
6	16	-0.001500	NA	NA	-0.001485	NA	NA
7	17	-0.073803	0	0	0.072450	1	0
8	20	0.023498	1	1	0.105458	1	0
9	22	0.126786	1	0	-0.149254	0	0
10	28	0.046047	1	0	-0.271337	0	0
11	31	0.022856	1	0	-0.021373	0	0
12	34	-0.209418	0	0	0.176432	1	0
13	35	0.014491	1	1	0.036310	1	0
14	36	-0.057434	0	0	-0.017369	0	1
15	37	-0.188355	0	1	-0.167515	0	0
16	39	-0.044542	0	1	-0.075195	0	0
17	40	0.008666	NA	NA	0.001338	NA	NA
18	43	-0.076150	0	0	0.079928	1	0
19	47	0.084578	1	0	-0.216880	0	0

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
20	52	-0.134024	0	0	0.029649	1	0
21	53	-0.042884	0	0	0.065600	1	0
22	56	0.227314	1	0	-0.182310	0	0
23	57	-0.021442	0	0	0.010315	1	0
24	61	0.220355	1	1	0.057112	1	0
25	64	0.002489	NA	NA	-0.022714	0	0
26	65	-0.085756	0	0	0.122335	1	0
27	67	-0.000983	NA	NA	0.000060	NA	NA
28	69	0.016604	1	1	0.018549	1	0
29	71	0.006508	NA	NA	0.001896	NA	NA
30	73	-0.301636	0	0	0.074206	1	0
31	74	0.021063	1	1	0.068859	1	0
32	76	-0.045770	0	0	0.032442	1	0
33	79	0.022489	1	0	-0.002098	NA	NA
34	80	0.347419	1	0	-0.118070	0	0
35	81	-0.239696	0	0	0.095886	1	0
36	82	-0.047219	0	0	0.053466	1	0
37	84	0.245807	1	0	-0.196578	0	0
38	92	0.002754	NA	NA	-0.001976	NA	NA
39	93	-0.020514	0	0	-0.052821	0	1
40	97	-0.185595	0	0	0.312015	1	0
41	99	0.249844	1	0	-0.201260	0	0

Removing identified anomadic items from the model

We proceed to eliminate the identified anomadic items so as not to use them in the next iteration.

In [405]:
```
## Local variable for the elimination of the anomaly items.
instructions = ''
init = False
question_anomaly = []

if selection_mode == 1:

    ## Items that fail in the defined class are retrieved.
    dm_delete = dm[dm['Anomaly ' + str(target_names[select_class_mode])] == 1]
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 2:

    ## Items that fail in classes defined with the or condition type are retrieve
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 1"
            init = True
        else:
            instructions += " or `Anomaly " + str(target_names[index]) + "` == 1'

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 3:

    ## Items that fail in classes defined with the and condition type are retriev
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 1"
            init = True
        else:
            instructions += " and `Anomaly " + str(target_names[index]) + "` == 1

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 4:

    ## Out-of-threshold items are retrieved in classes defined with the conditior
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 'NA'"
            init = True
        else:
            instructions += " and `Anomaly " + str(target_names[index]) + "` == '

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

## The selected blank items are printed according to the selection_mode.
question_anomaly
```

Out[405]: [2, 36, 93]

Model explainability process

The data used for the test of the neural network model is shown.

In [406]: X_test

Out[406]:

	1. Most of the time I have difficulty concentrating on simple tasks	2. I don't feel like doing my daily duties	6. I consider that my life is full of good things	8. I'm still as punctual as I have always been	10. I have found that I can spend a lot of time scrolling the screen \nof my cell phone without searching or stopping at anything in particular	13. I felt more encouraged to do my daily activities before	16. I think my thoughts are strange or different from before	17. I feel guilty about the decisions that I have made	20. diffi for m m decisi eve they sin
69	1	0	1	0	0	0	0	0	
46	1	0	1	1	1	1	1	1	
58	0	0	1	1	0	1	1	0	
114	1	0	1	0	1	1	1	0	
73	0	0	1	1	1	0	0	0	
98	1	0	0	1	1	0	1	0	
31	0	0	1	0	0	0	0	1	
53	1	0	1	1	1	1	1	1	
65	0	0	1	1	0	1	0	0	
96	0	0	1	1	1	0	0	1	
95	0	0	1	0	1	0	1	0	
97	0	0	1	1	0	0	0	1	
2	0	0	1	0	1	0	1	0	
62	0	1	1	0	1	0	1	0	
110	0	0	0	1	0	1	0	0	
55	0	1	1	1	0	0	1	1	
103	1	1	1	0	1	0	0	0	
100	1	1	1	0	0	1	0	0	
66	1	1	0	0	1	0	1	1	
44	0	0	1	1	0	0	0	0	
77	0	1	1	1	1	0	1	1	
17	1	0	1	1	1	1	1	0	
81	1	0	1	1	0	1	1	1	
74	1	1	0	1	1	1	1	1	

56	0	0	1	1	0	1	1	1
94	1	1	0	0	1	1	1	1
35	1	0	1	1	0	0	1	0
38	0	1	1	1	0	0	1	0
93	0	0	1	1	1	1	1	0
48	1	1	1	0	1	1	1	1
33	0	1	1	0	1	1	1	1
59	0	1	1	1	1	0	0	1
92	1	0	0	1	0	0	1	1
42	1	0	1	1	0	0	1	0
10	0	0	1	0	1	0	1	0
85	1	1	1	1	1	1	1	1
32	0	1	1	0	1	1	1	1
78	1	0	1	1	0	0	1	0
19	0	0	0	1	0	1	1	0

39 rows × 42 columns

Explanation of the model using ALE

In analysis of the model through the results obtained by ALE, it is possible to identify the behavior of the neural network.

In [407]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

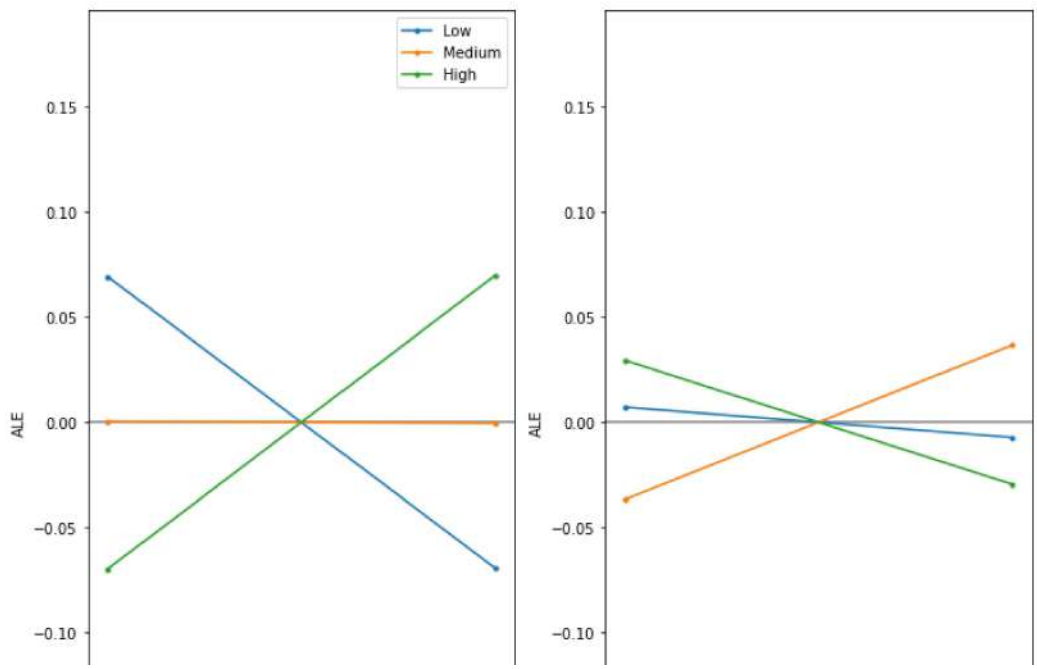
```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

The graphs of all the data present in the dataset used are shown.

In [408]:
```
plot_ale(proba_exp_lr, n_cols=2, features=list(range(df.shape[1]-1)),fig_kw={'fig
```



Explanation of the model using LIME

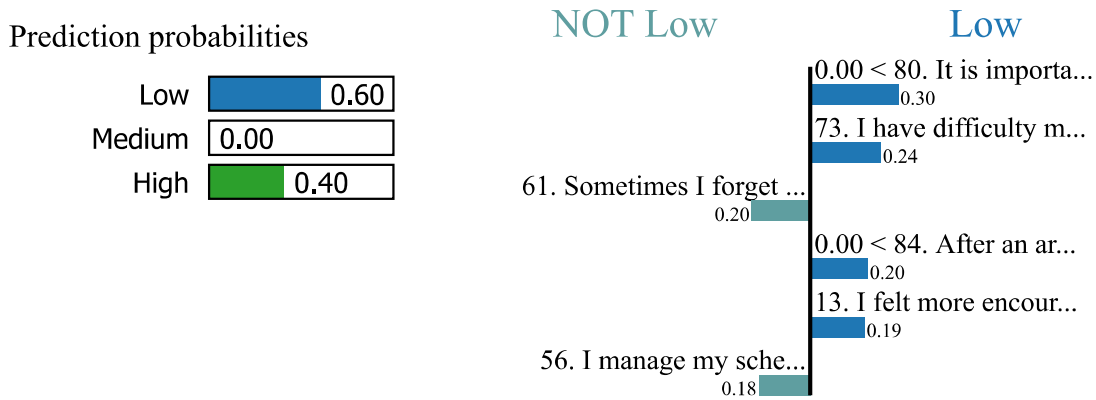
We proceed to explain the improved model using LIME for the Low, Medium and High classes.

In [409]:
```
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorica
                                                   feature_names=train_cols, clas
                                                   discretize_continuous=True)
```

The first individual present in the test data is explained.

In [410]:
```
exp = explainer.explain_instance(X_test.to_numpy()[0], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

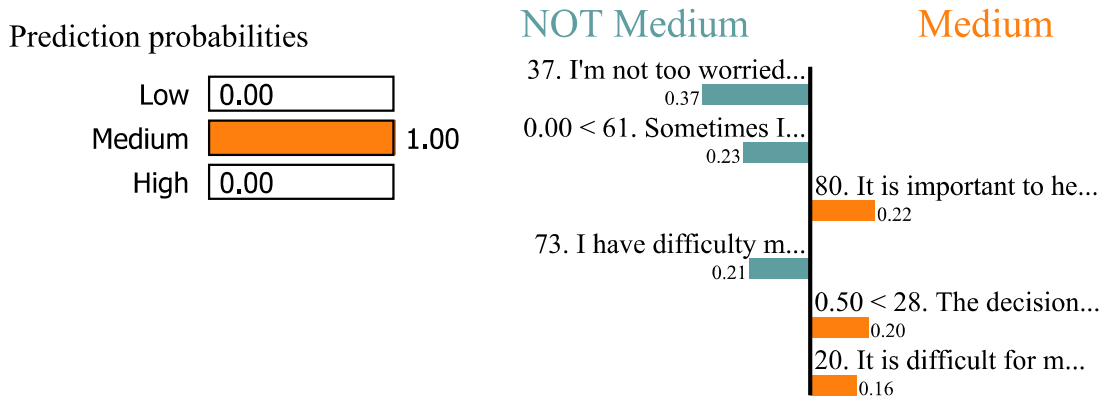


Feature	Value
80. It is important to help people when they need it	1.00
73. I have difficulty making elaborate plans	0.00
61. Sometimes I forget what I wanted to say because I have several thoughts at the same time	0.00
84. After an argument I usually go over what happened in my head	1.00
13. I felt more encouraged to do my daily activities before	0.00
56. I manage my schedule as I always have	0.00

The second individual present in the test data is explained.

In [411]:
```
exp = explainer.explain_instance(X_test.to_numpy()[1], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

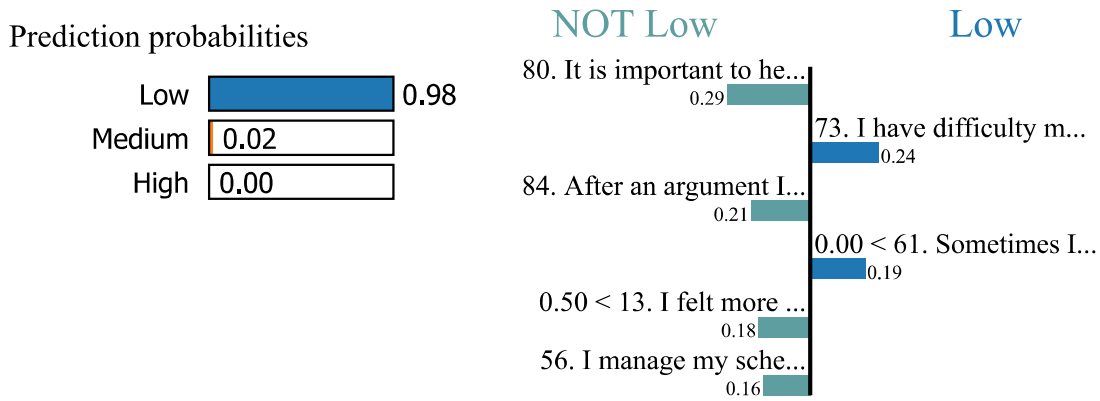


Feature	Value
37. I'm not too worried about what might happen in a few weeks	0.00
61. Sometimes I forget what I wanted to say because I have several thoughts at the same time	1.00
80. It is important to help people when they need it	0.00
73. I have difficulty making elaborate plans	0.00
28. The decisions I have made so far have been the right ones	1.00
20. It is difficult for me to make decisions even if they are simple	0.00

The last individual present in the test data is explained.

In [412]:
```
exp = explainer.explain_instance(X_test.to_numpy()[-1], mlp.predict_proba, num_fe
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```



Feature	Value
80. It is important to help people when they need it	0.00
73. I have difficulty making elaborate plans	0.00
84. After an argument I usually go over what happened in my head	0.00
61. Sometimes I forget what I wanted to say because I have several thoughts at the same time	1.00
13. I felt more encouraged to do my daily activities before	1.00
56. I manage my schedule as I always have	0.00

Explanation of the model using SHAP

SHAP explanatory values are displayed according to the class to which the selected individual belongs according to the prediction of the model.

The first individual present in the test data is explained.

In [413]:
```
## Individual Selection.
individual = X_test.iloc[[0]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The second individual present in the data test is explained.

In [414]:
```
## Individual Selection.
individual = X_test.iloc[[1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The last individual present in the data test is explained.

In [415]:
```
## Individual Selection.
individual = X_test.iloc[[-1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

Explanation using heatmaps

The importance of the data present in the neural network is shown from explanation methods.

In [416]:

```
## Items currently in use are retrieved.
ds_item = ds[ds.Status == "In use"].reset_index(drop=True)

## The data is prepared for the generation of heatmaps.
column_Map = ds_item['Question'].tolist()
content_Map = [False for x in range(1, df.shape[1])]

## The next items to be deleted will be marked.
for index in range(0, len(content_Map)):

    if iteration == 0:
        for delete in list_contr:
            if column_Map[index] == delete:
                content_Map [index] = True
    else:
        for delete in question_anomaly:
            if column_Map[index] == delete:
                content_Map [index] = True

## The dataframe is created with the prepared data.
df_mask = pd.DataFrame([content_Map], columns = column_Map)
df_mask_imp = pd.concat([df_mask, df_mask, df_mask], ignore_index = True, axis =
df_mask_all = pd.concat([df_mask, df_mask, df_mask, df_mask], ignore_index = True

df_mask
```

Out[416]:

	1	2	6	8	10	13	16	17	20	22	...	76	79	80	81
0	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False

1 rows × 42 columns

Identification of feature variables

Identification of feature variables from the randomness of the answers as items of the data.

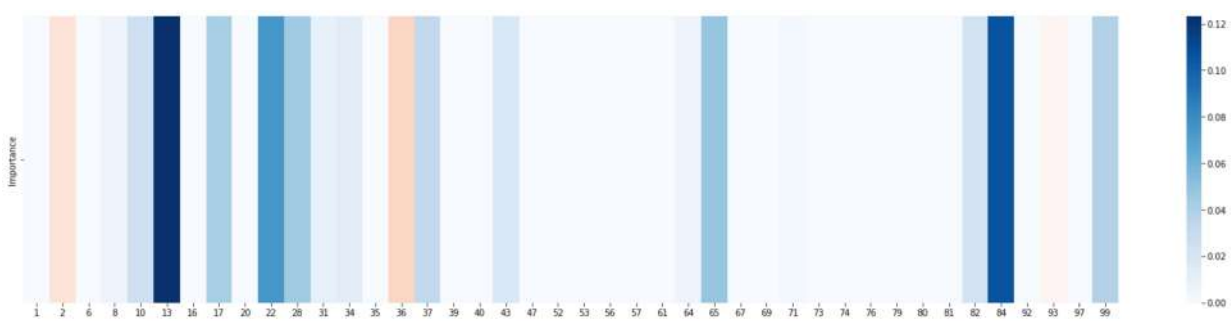
In [417]:
```
## Loading the data for explainability.
explainer = ClassifierExplainer(mlp, X_test, y_test)

## Class index local variable.
iterable_class = -1
map_class = []
```

```
WARNING: Parameter shap='guess', but failed to guess the type of shap explainer
to use for MLPClassifier. Defaulting to the model agnostic shap.KernelExplainer
(shap='kernel'). However this will be slow, so if your model is compatible with
e.g. shap.TreeExplainer or shap.LinearExplainer then pass shap='tree' or shap
='linear'!
WARNING: For shap='kernel', shap interaction values can unfortunately not be ca
lculated!
Note: shap values for shap='kernel' normally get calculated against X_backgroun
d, but paramater X_background=None, so setting X_background=shap.sample(X, 5
0)...
Generating self.shap_explainer = shap.KernelExplainer(model, X, link='identit
y')
```

Explainability of the data belonging to the first class, in this case the Low.

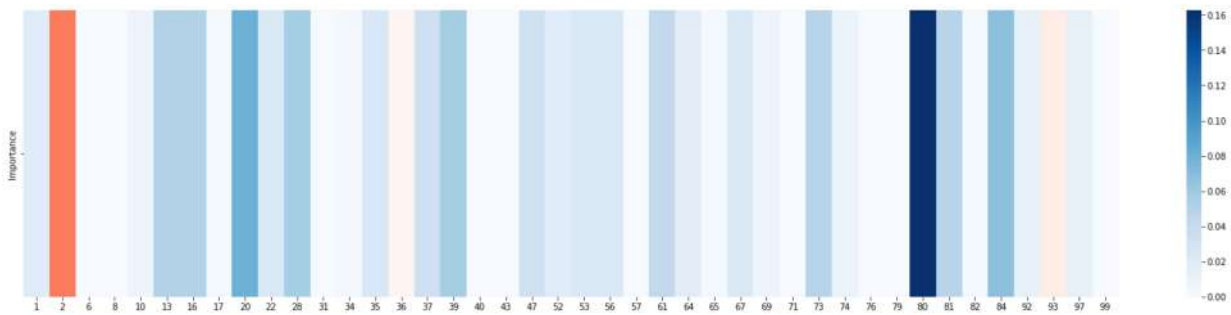
In [418]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

```
Calculating permutation importances (if slow, try setting n_jobs parameter)...
```



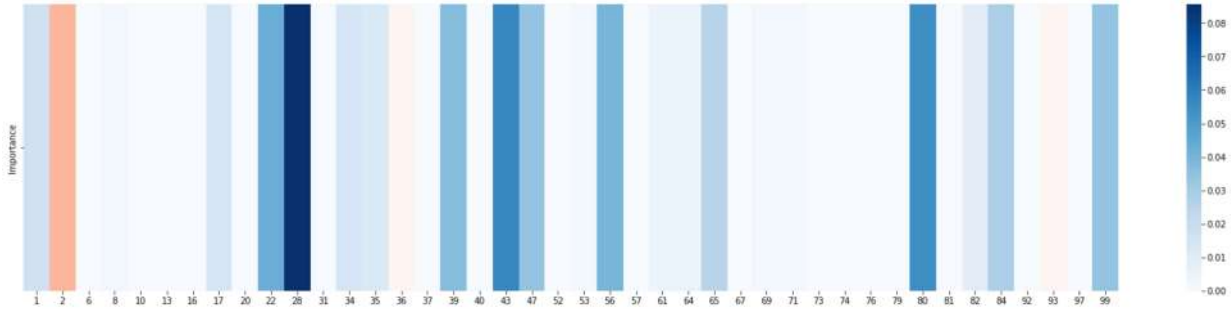
Explainability of the data belonging to the second class, in this case the Medium.

In [419]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



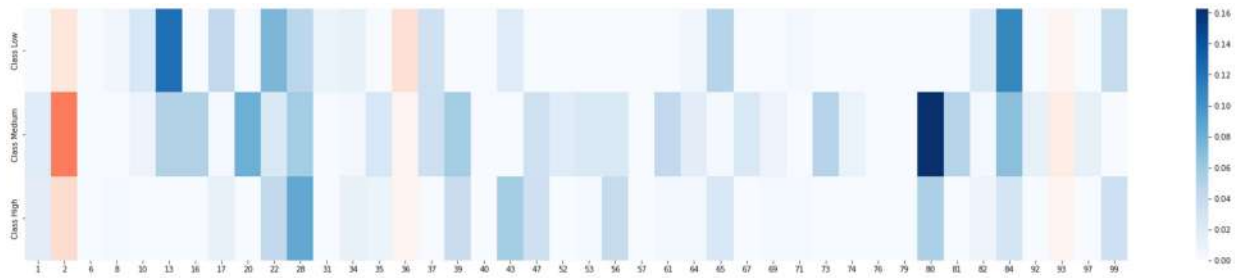
Explainability of the data belonging to the third class, in this case the High.

In [420]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [421]:
```
## Fusion of the previous heatmaps.
df_importance = pd.concat(map_class, axis=1)
df_importance.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_importance.transpose(), cmap = "Reds", cbar = False)
sns.heatmap(df_importance.transpose(), cmap = "Blues",yticklabels = True, xtickla
plt.show()
```



Identification of feature variables using SHAP

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of SHAP.

In [422]:
```
## Class index local variable.
iterable_class = -1
map_class = []
```

Individual explainability of the data belonging to the first class, in this case the Low.

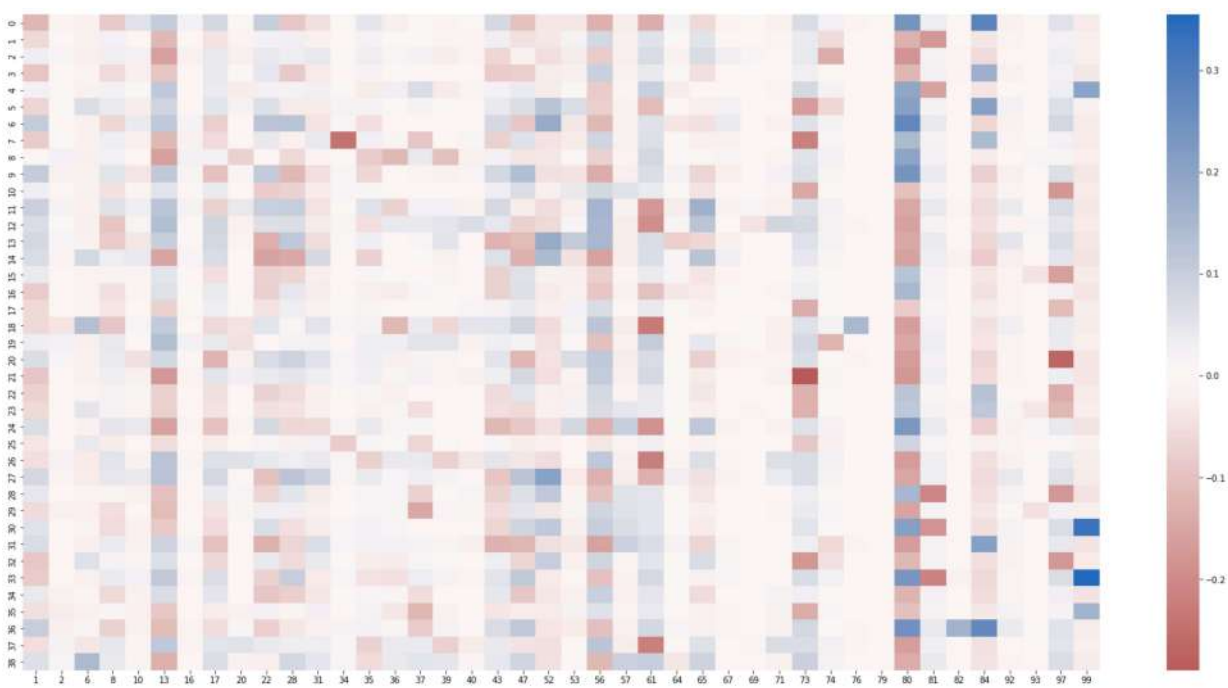
Definition of explainability data.

In [423]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

```
Calculating shap values...

  0%|          | 0/39 [00:00<?, ?it/s]
```

Generation of the heatmap.

In [424]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

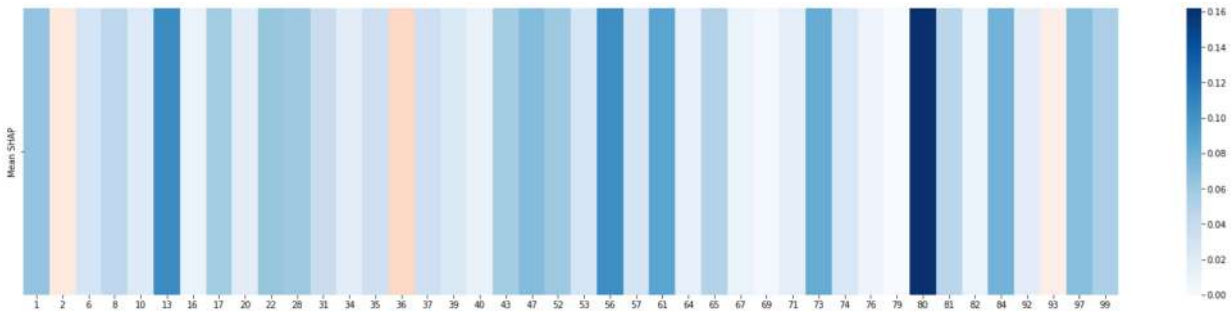


Fusion of the data used.

In [425]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [426]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



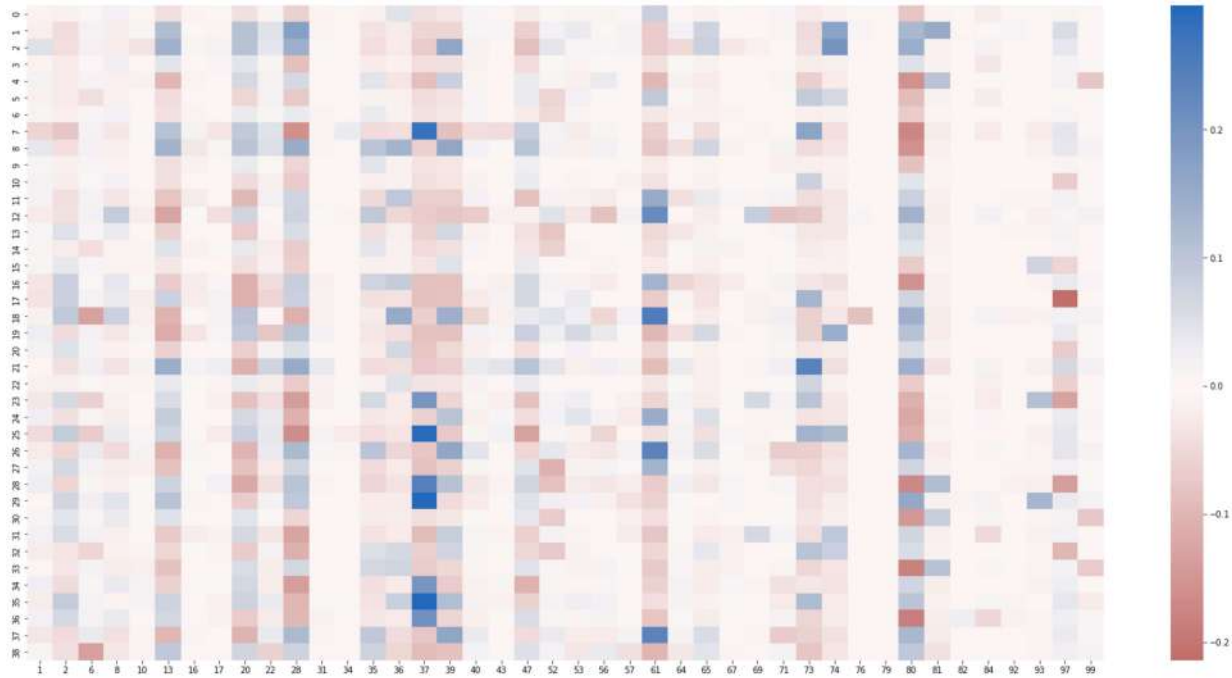
Individual explainability of the data belonging to the second class, in this case the Medium.

Definition of explainability data.

In [427]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [428]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

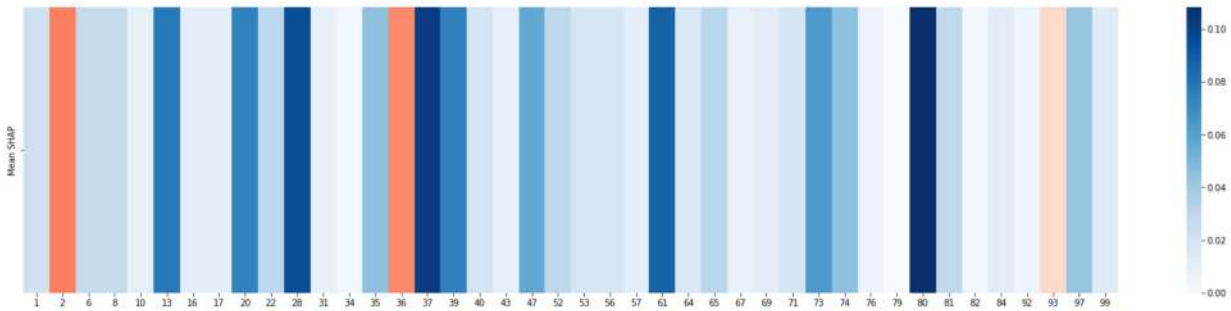


Fusion of the data used.

In [429]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [430]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



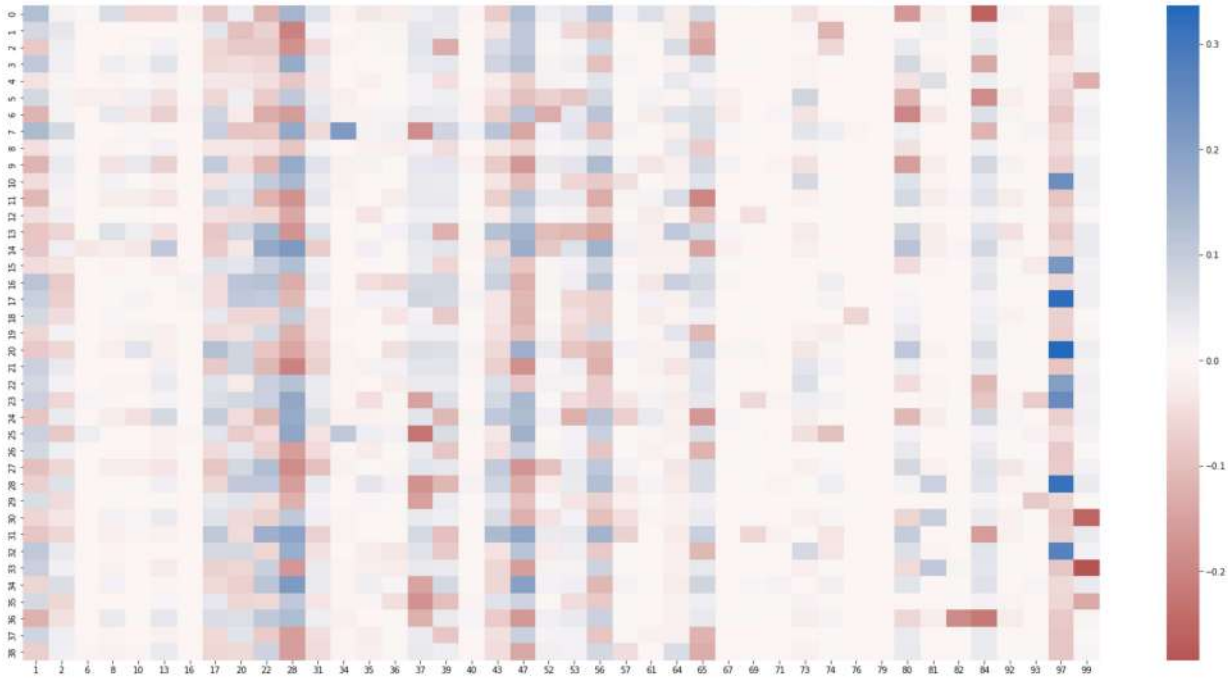
Individual explainability of data belonging to the third class, in this case High.

Definition of explainability data.

In [431]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [432]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

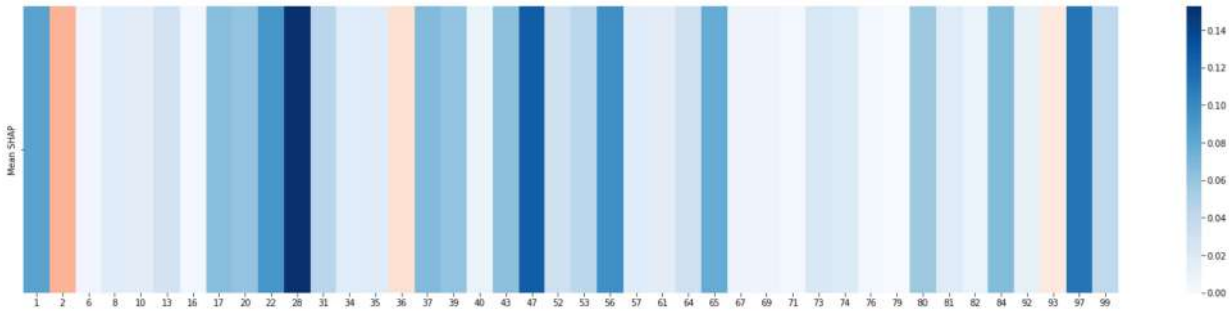


Fusion of the data used.

In [433]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

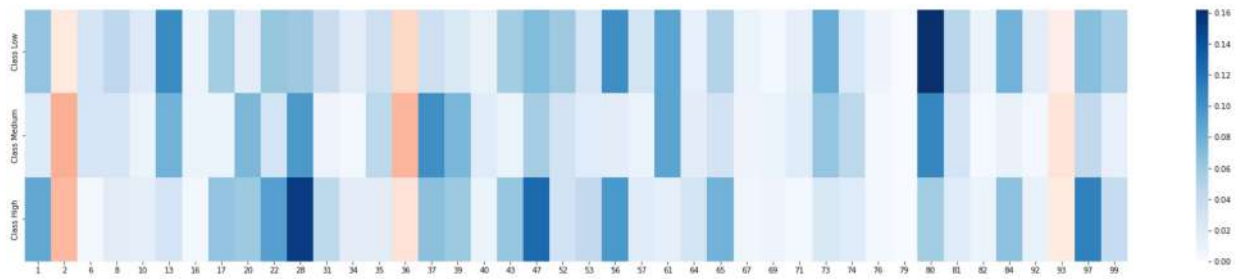
Generation of the heatmap.

In [434]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [435]:
```
## Fusion of the previous heatmaps.
df_shap = pd.concat(map_class, axis=1)
df_shap.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_shap.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_shap.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using LIME

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of LIME.

In [436]:
```
## Data preparation for explainability.
lime_exp = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorical
                                                  feature_names=train_cols, class
                                                  discretize_continuous=True)

## Class index local variable.
iterable_class = -1
map_class = []
```

Obtaining the data for the generation of the explanations by LIME about the Low, Medium and High classes.

In [437]:
```
## Local variables for LIME array data.
exp_matrix = [[] for _ in range(len(name_class))]

## Recovering the data needed for LIME.
for x in X_test.to_numpy():

    ## Auxiliary temporary variables.
    exp_list = [[] for _ in range(len(name_class))]

    ## Loading the data for explainability.
    exp = lime_exp.explain_instance(x, mlp.predict_proba, num_features = df.shape

    ## Recovering the data.
    for elements in range(0, len(name_class)):
        temp = exp.as_map()[elements]
        temp.sort(key=itemgetter(0))

        ## Saving data of the Low, Medium and High classes.
        for tup in temp:
            exp_list[elements].append(tup[1])

        exp_matrix[elements].append(exp_list[elements])
```

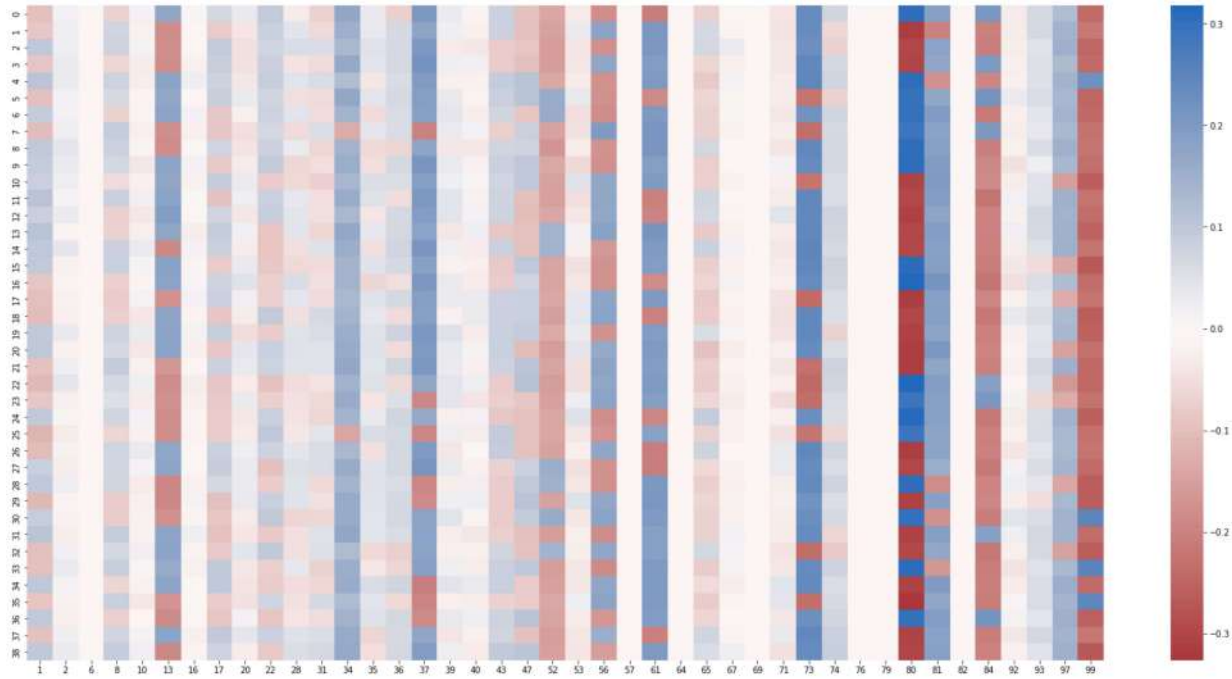
```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

Individual explainability of the data belonging to the first class, in this case the Low.

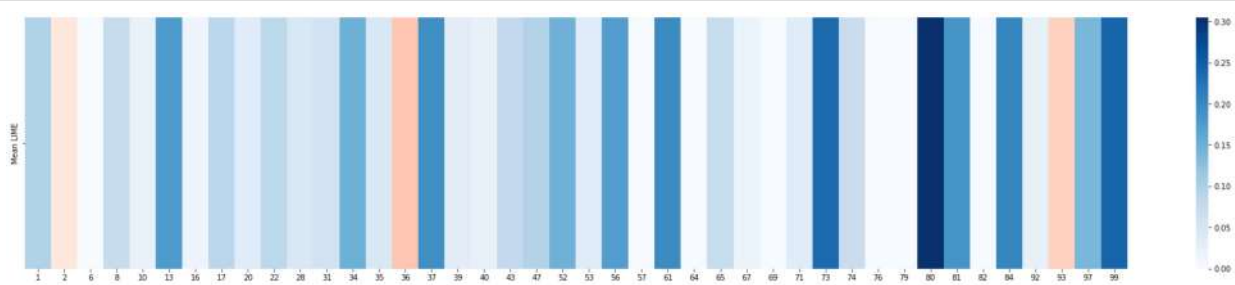
Generation of the heatmap.

In [438]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

In [439]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

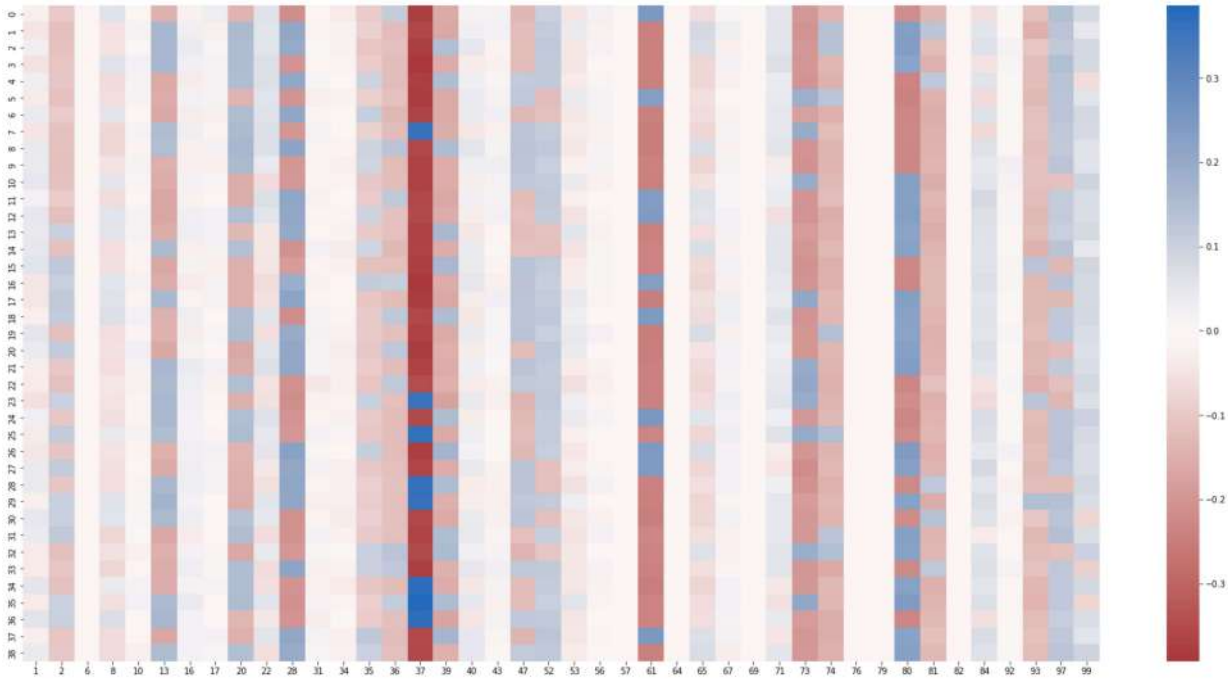


Individual explainability of the data belonging to the second class, in

this case the Medium.

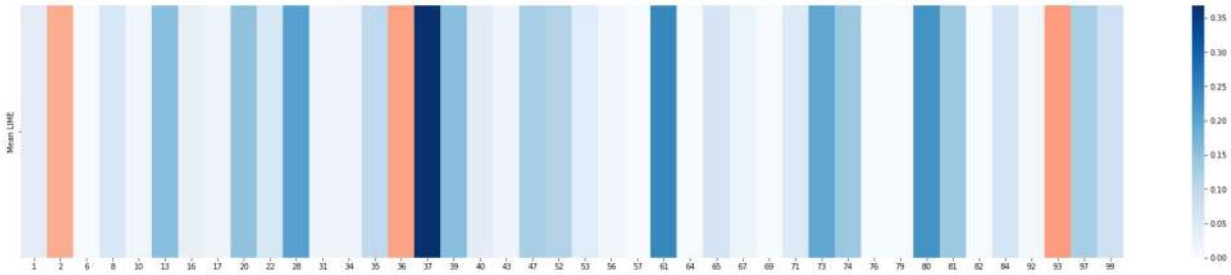
Generation of the heatmap.

In [440]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

In [441]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

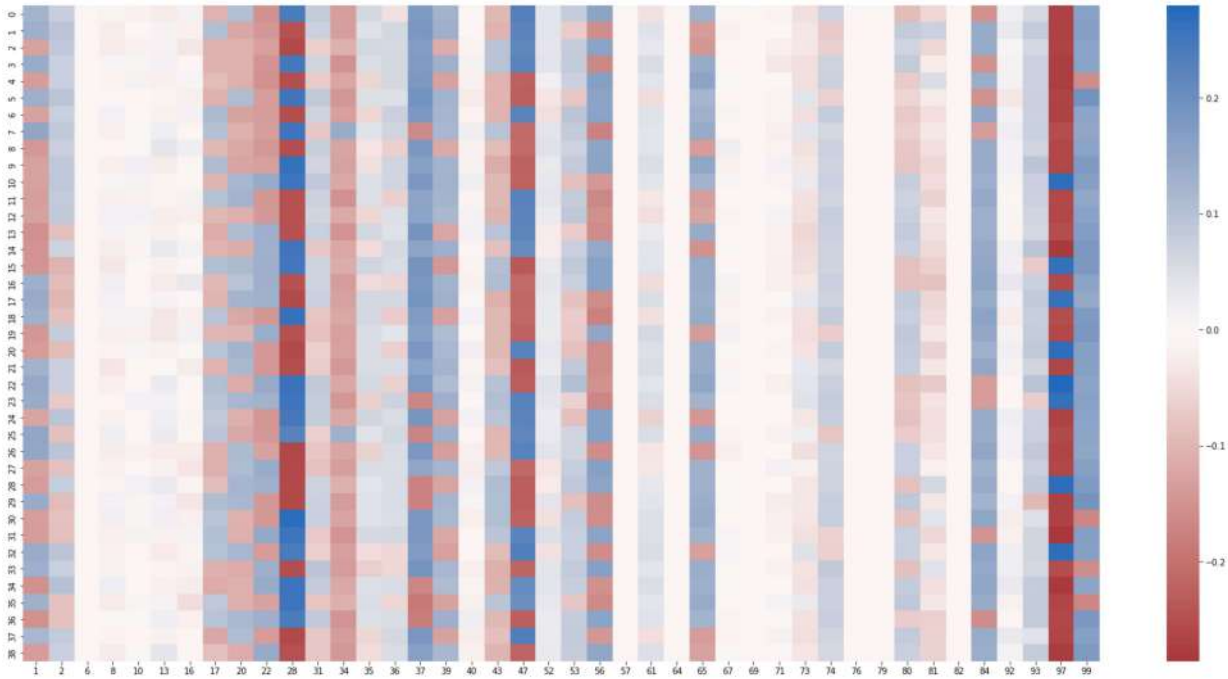


Individual explainability of data belonging to the third class, in this

case High.

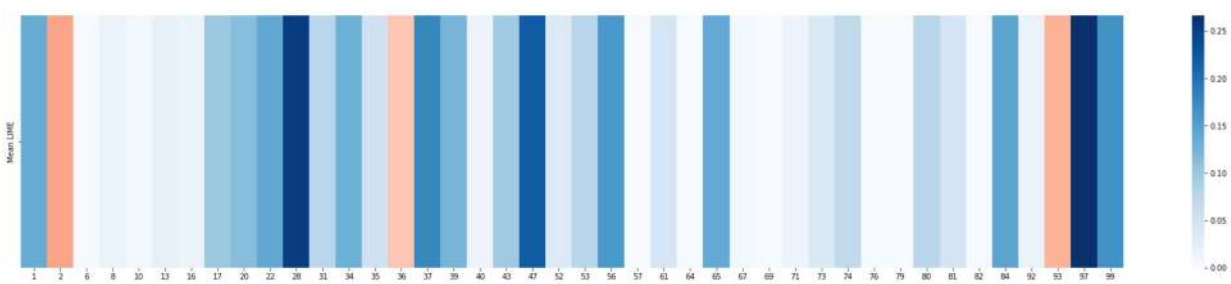
Generation of the heatmap.

In [442]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



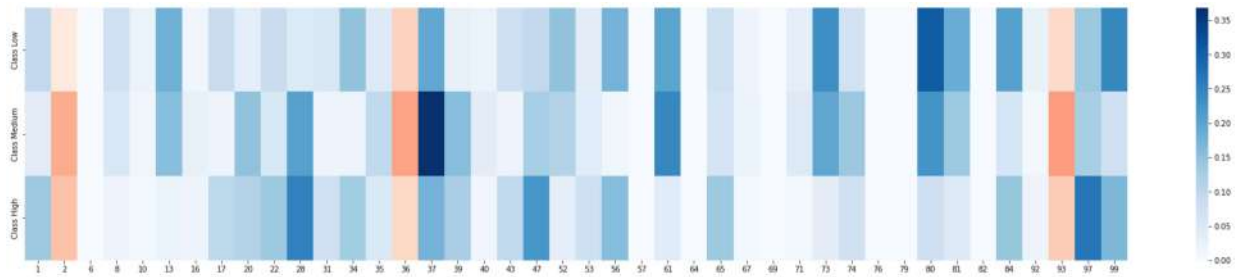
Fusion of the data used.

In [443]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [444]:
```
df_lime = pd.concat(map_class, axis=1)
df_lime.columns = name_class
plt.figure(figsize = (32,6))
sns.heatmap(df_lime.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_lime.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using ALE

It shows the main effect of each question compared to the probability that the model predicts in each class. This effect is shown for both responses. Since there are only 2 answers, the map is the same but with the colors reversed.

In [445]:
```
## Definition of explainability data.
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names=train_cols, target_names= target_r
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())

## Class index local variable.
iterable_class = -1
map_class = []
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

Obtaining the data for the generation of the explanations by ALE about all the classes.

In [446]:
```
## Local variables for ALE array data.
ale_list = [[] for _ in range(len(name_class))]

## Recovering the data needed for ALE.
for array in proba_exp_lr.ale_values:

    ## Recovering the data.
    for elements in range(0, len(name_class)):

        ## Saving data of all the classes.
        ale_list[elements].append(array[0][elements])
```

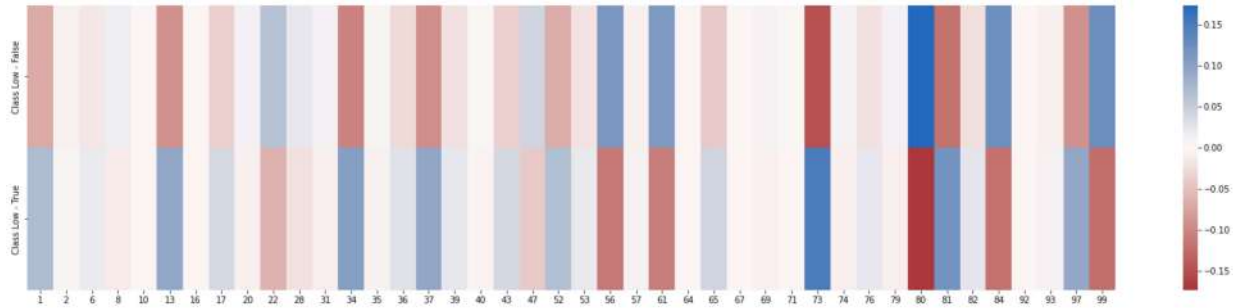
Data processing for the explainability of ALE.

In [447]:
```
## Data processing of the Low, Medium and High classes.
for elements in range(0, len(name_class)):
    ale_df = pd.DataFrame([ale_list [elements]])
    ale_df = pd.concat([ale_df.multiply(-1), ale_df])
    ale_df.index = [name_class [elements] + " - False", name_class [elements] + '
    ale_df.columns = column_Map
    map_class.append(ale_df)
```

Explainability of the data belonging to the first class, in this case the Low.

Generation of the heatmap.

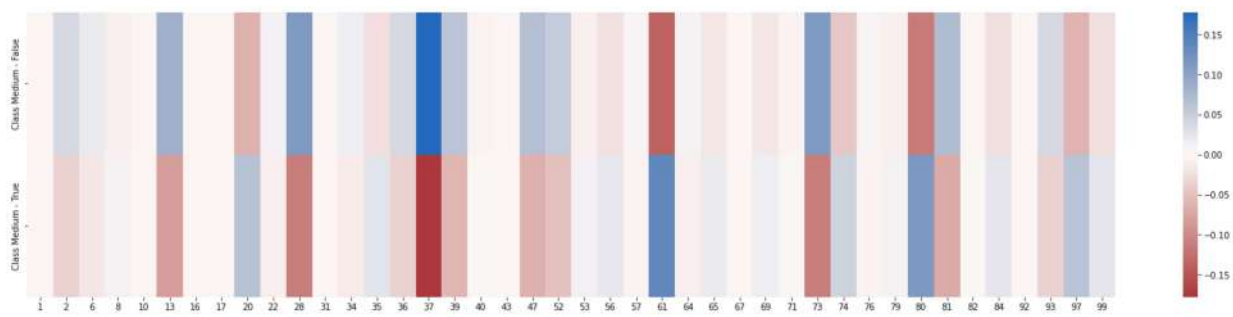
In [448]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the second class, in this case the Medium.

Generation of the heatmap.

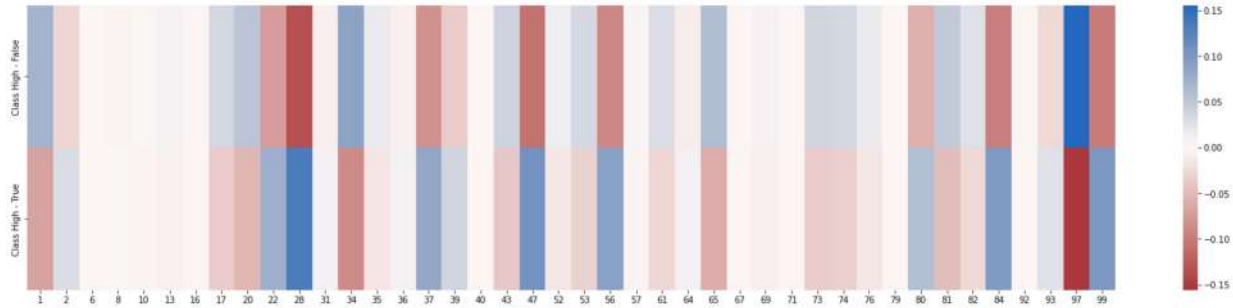
In [449]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the third class, in this case the High.

Generation of the heatmap.

In [450]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```

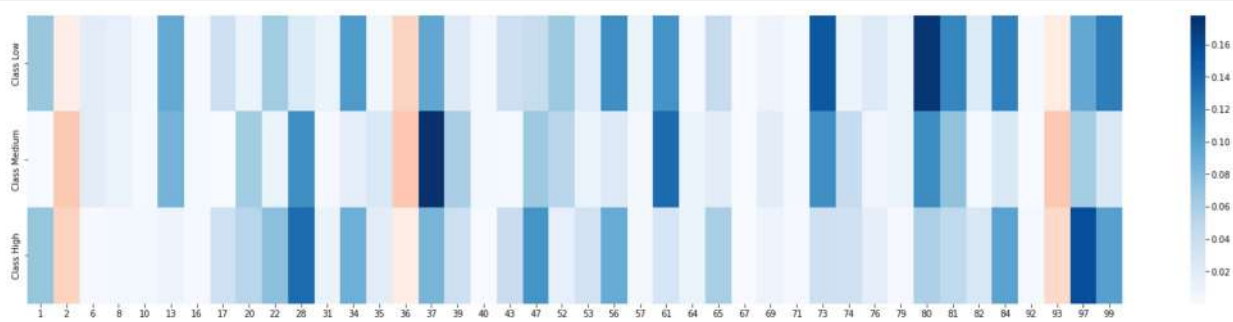


Fusion of the data used in all classes.

In [451]:
```
## Application of absolute value to data.
for index in range(0, len(map_class)):
    map_class [index] = map_class [index][0:1].abs()

## Union of the heatmaps of the Low, Medium and High class.
ale_df = pd.concat(map_class)
ale_df.index = name_class
ale_df = ale_df.transpose()
```

Generation of the heatmap.

In [452]:
```
plt.figure(figsize = (28,6))
sns.heatmap(ale_df.transpose(), cmap="Reds", cbar = False)
sns.heatmap(ale_df.transpose(), cmap="Blues", yticklabels = True, xticklabels = T
plt.show()
```



Comparing of Feature Variables Methods for Explainability

The global values of each of the methods used are taken, their maximums are obtained (the minimums are 0) and a percentage is calculated based on said maximum, which indicates how relevant the contribution is for each question.

In [453]:
```
## Class index local variable.
iterable_class = -1
```

Comparison of the explanation methods of the first class, in this case the Low.

The highest data of each method used is obtained.

In [454]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [455]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

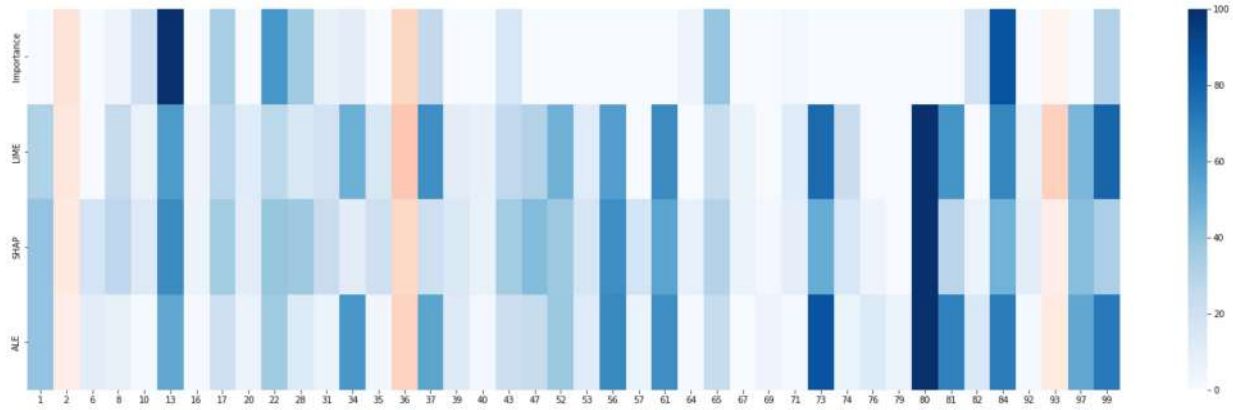
Out[455]:

	1	2	6	8	10	13	16	17
Importance	0.000000	10.526316	0.000000	5.263158	21.052632	100.000000	0.000000	34.210526
LIME	31.875238	8.088656	0.000000	24.621037	7.237264	58.569225	4.697227	27.955415
SHAP	40.201379	6.493279	17.258844	27.220158	12.520210	64.738933	5.514830	36.170134
ALE	39.857023	4.149958	10.201832	8.736596	0.979009	53.202363	0.431892	21.243200

4 rows × 42 columns

Generation of the heatmap.

In [456]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the second class, in this case the Medium.

The highest data of each method used is obtained.

In [457]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [458]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

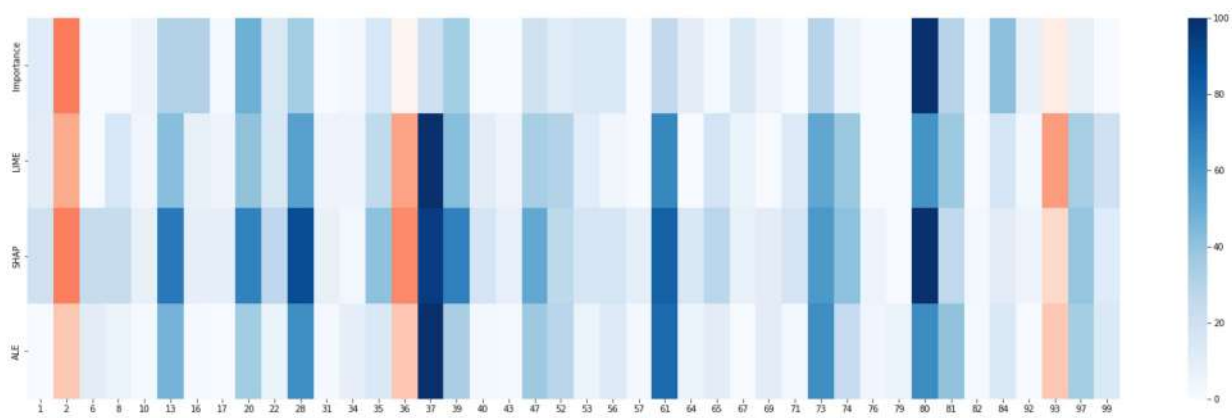
Out[458]:

	1	2	6	8	10	13	16	17
Importance	12.280702	43.859649	0.000000	0.000000	5.263158	31.578947	31.578947	1.754386
LIME	10.891320	29.591896	0.000000	15.728965	4.022175	42.897851	7.769197	4.535333
SHAP	19.366938	43.408954	24.836818	24.789216	8.650020	72.129153	9.551780	9.551779
ALE	0.266410	20.629580	10.497333	6.009730	1.165075	47.442615	0.838935	0.380057

4 rows × 42 columns

Generation of the heatmap.

In [459]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the third class, in this case the High.

The highest data of each method used is obtained.

In [460]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [461]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

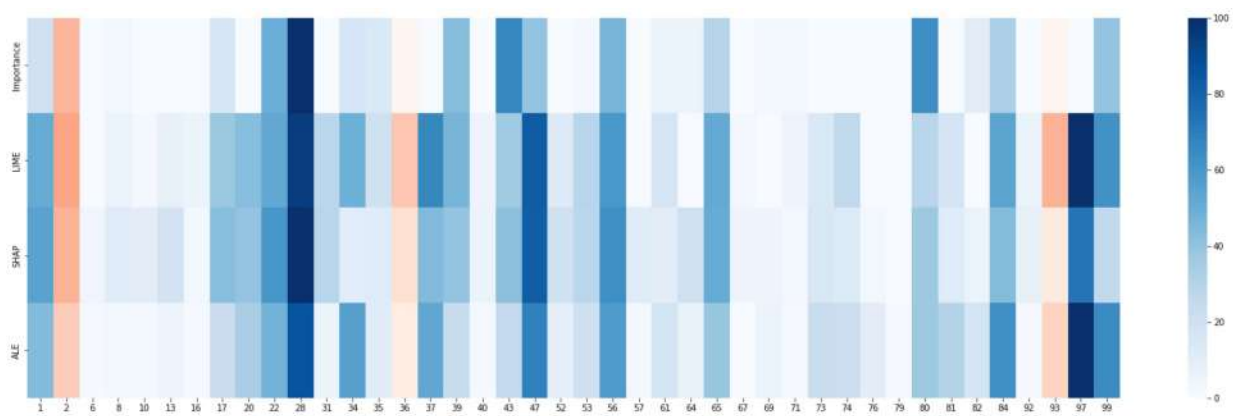
Out[461]:

	1	2	6	8	10	13	16	17
Importance	20.000000	26.666667	0.000000	3.333333	0.000000	0.000000	0.000000	16.666667
LIME	51.550177	31.633534	0.000000	6.455086	3.224329	7.780438	5.952897	38.277955
SHAP	55.330556	27.271460	3.575223	11.522221	9.823506	18.520098	3.118254	42.528120
ALE	44.683439	18.908312	0.613353	2.873509	2.418929	5.128317	0.475953	23.220182

4 rows × 42 columns

Generation of the heatmap.

In [462]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Safeguarding the items eliminated during this iteration.

In [463]:
```
## Removal of anomalous items selected by selection_mode.
for x in question_anomaly:

    ## Status change of the deleted items in the respective auxiliary dataset.
    ds.loc[x - 1,'Status'] = 'Delete'

## Save the changed statuses in the dataset.
ds.to_csv(path_dataset_qs, index = False)
```

Iteration 6

The first iteration of improvement of the model is carried out.

Preparing the data for the neural network

The data is prepared in order to be processed by the neural network.

In [464]:

```
## The iteration number is increased.
iteration += 1

## Loading data from the dataset.
df = pd.read_csv(path_dataset)
ds = pd.read_csv(path_dataset_qs)

## Elimination of unrelated items from the dataset.
if len(unrelated_questions) != 0:
    df = df.drop(df.columns[unrelated_questions[0]:unrelated_questions[1]], axis=

i = 0
if iteration == 1:
    ## Elimination of list of conflicting items declared in global variables.
    for x in list_contr:
        df.drop(df.columns[x - (i + 1)], axis = 1, inplace = True)
        i += 1
        ## Status change of the deleted question in the respective auxiliary data
        ds.loc[x - 1,'Status'] = 'Delete'

else:
    ## Elimination of the list of anomadic items defined in the improvement proce
    ds_delete = ds[ds.Status == "Delete"].reset_index(drop=True)
    for x in ds_delete['Question'].tolist():
        df.drop(df.columns[x - (i + 1)], axis = 1, inplace = True)
        i += 1

## Grouping of the target according to the defined dictionary.
if grouping_name != None and grouping_name != '':
    df[grouping_name] = df[grouping_name].map(class_dic)

## Assignment of local variables according to the data necessary for the neural r
train_cols = df.columns [0:-1]
label = df.columns [-1]
X = df [train_cols]
y = df [label]

## The prepared data of the dataset is printed.
df
```

Out[464]:

	1. Most of the time I have difficulty concentrating on simple tasks	6. I consider that my life is full of good things	8. I'm still as punctual as I have always been	10. I have found that I can spend a lot of time scrolling the screen \nof my cell phone without searching or stopping at anything in particular	13. I felt more encouraged to do my daily activities before	16. I think my thoughts are strange or different from before	17. I feel guilty about the decisions that I have made	20. It is difficult for me to make decisions even if they are simple
0	1	1	1	0	0	0	0	1
1	1	1	1	0	1	1	1	1
2	0	1	0	1	0	1	0	0
3	0	1	1	0	1	0	0	1
4	1	1	1	1	0	1	1	1
...	...	...	...	...	...	...	...	...
99	1	1	1	1	1	0	0	1
100	1	1	0	0	1	0	0	1
101	0	1	0	1	1	1	0	0
102	1	1	1	1	0	0	1	1
103	1	1	0	1	0	0	0	1

104 rows × 40 columns

Oversample application

An oversample process is applied to level the amount of existing data given by the grouping.

In [465]:
```
## The number of elements present in the grouped target is printed.
y.value_counts()
```

Out[465]:
```
0    39
1    35
2    30
Name: Target, dtype: int64
```

Oversample application.

In [466]:
```
## Application of the oversample to the prepared data.
X, y = oversample.fit_resample(X, y)
```

Printing of the data after the application of the oversample.

In [467]:
```
## The results obtained after applying the oversample are printed.
y.value_counts()
```

Out[467]:
```
0    39
1    39
2    39
Name: Target, dtype: int64
```

Preparation of the parameters of the neural network

We proceed to define the parameters necessary for the training of the neural network.

In [468]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
```

Neural network training

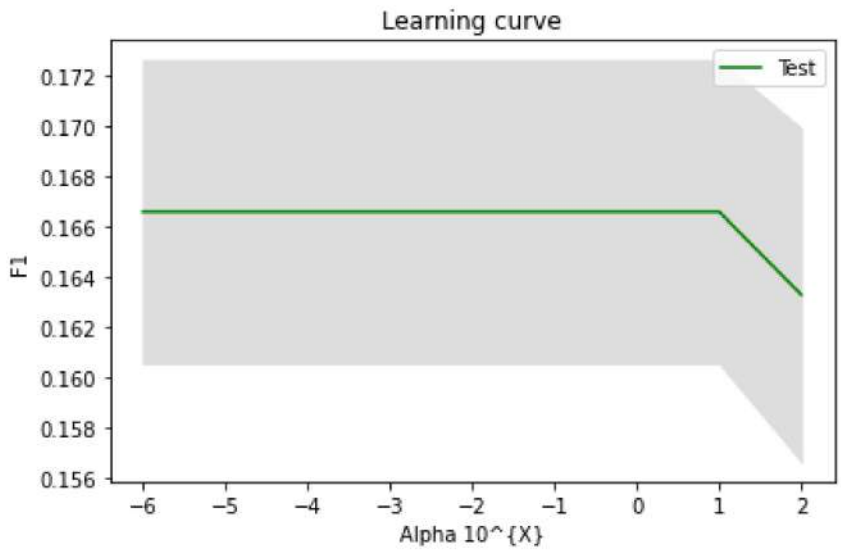
The alpha parameters and the initial learning rate are adjusted to obtain the best performance with the model using cross validation.

In [469]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_normal

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = 0.0001, max_iter =

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[469]:
```
([0.1632974910394265,
  0.16659498207885304,
  0.16659498207885304,
  0.16659498207885304,
  0.16659498207885304,
  0.16659498207885304,
  0.16659498207885304,
  0.16659498207885304,
  0.16659498207885304],
 [0.006619092595549127,
  0.006020651927912844,
  0.006020651927912844,
  0.006020651927912844,
  0.006020651927912844,
  0.006020651927912844,
  0.006020651927912844,
  0.006020651927912844,
  0.006020651927912844])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [470]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Alpha 10^{X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



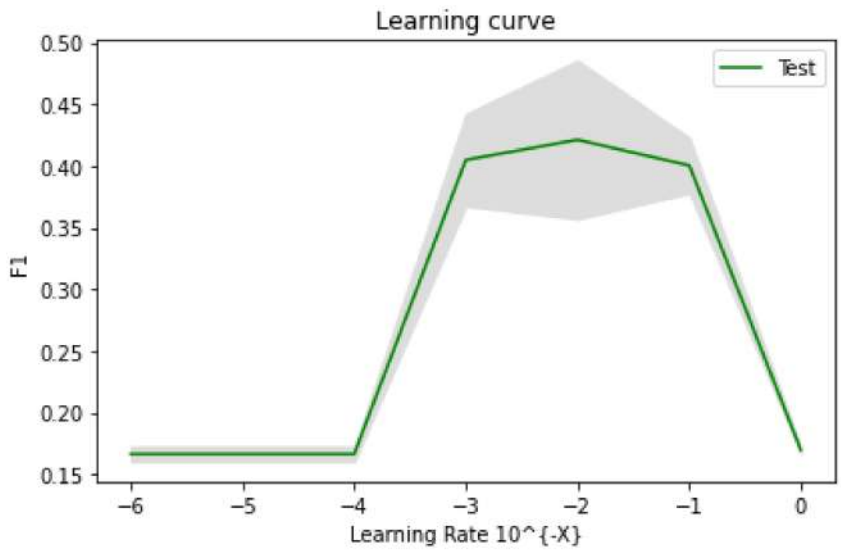
The training is performed again keeping the same parameters except for the alpha value that changes to 1.

In [471]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_alpha

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = regul_param, max_it

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[471]:
```
([0.16989247311827954,
  0.40041358080057776,
  0.42131415425533064,
  0.4048821113526996,
  0.16659498207885304,
  0.16659498207885304,
  0.16659498207885304],
 [0.002633859938476533,
  0.022573278819447408,
  0.06436412560720102,
  0.03753187387238331,
  0.006020651927912844,
  0.006020651927912844,
  0.006020651927912844])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [472]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Learning Rate 10^{-X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



Generation of the neural network model

The parameters of the final model are modified according to the data obtained in the training of the neural network.

In [473]:
```
## The value of the alpha parameter, learning_rate_init and random_state are char
## to the data in the global variables.
mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solver =
                    learning_rate = 'constant', learning_rate_init = lear
                    random_state = random_state_model)

## We print the data of the final generated model.
mlp.fit(X_train, y_train)
```

Out[473]:
```
MLPClassifier
MLPClassifier(activation='logistic', alpha=0.001, hidden_layer_sizes=(10,),
              learning_rate_init=0.1, max_iter=100000, random_state=42)
```

We safeguard the final generated model

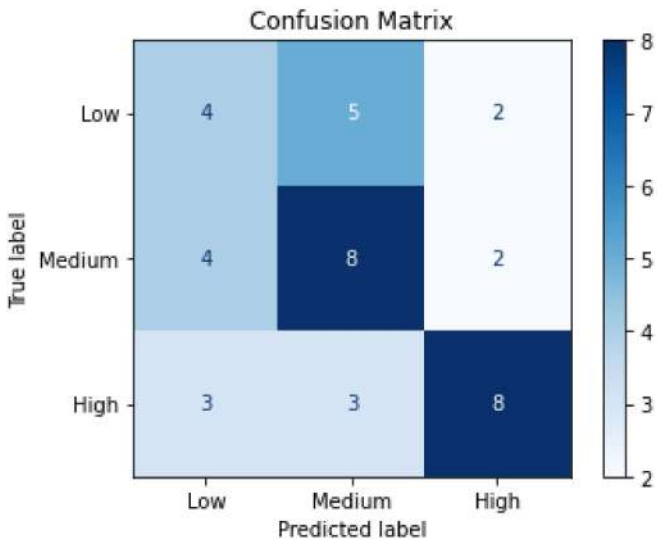
In [474]:
```
joblib.dump(mlp,"model_depression_i" + str(iteration) + ".pkl")
```

Out[474]:
```
['model_depression_i6.pkl']
```

Generation of the confusion matrix of the model

We obtain the statistics of the final model to generate its confusion matrix.

In [475]:
```
## The data for the generation of the confession matrix are defined.
confusion_matrix = ConfusionMatrixDisplay.from_estimator(mlp, X_test, y_test, dis
                                                         cmap = plt.cm.Blues)
confusion_matrix.ax_.set_title("Confusion Matrix")
plt.show()
```



We get the detailed statistics of the final model.

In [476]:
```
## The necessary detailed prediction is generated.
y_pred = mlp.predict(X_test)

## The detailed statistics of the model are printed.
print('Classification accuracy =',accuracy_score(y_test,y_pred) * 100,'%\n')
print(classification_report(y_test,y_pred))
```

```
Classification accuracy = 51.28205128205128 %

              precision    recall  f1-score   support

           0       0.36      0.36      0.36        11
           1       0.50      0.57      0.53        14
           2       0.67      0.57      0.62        14

    accuracy                           0.51        39
   macro avg       0.51      0.50      0.50        39
weighted avg       0.52      0.51      0.51        39
```

The data obtained is saved for later graphing.

In [477]:
```
accuracy = accuracy_score(y_test,y_pred) * 100
clf_rep = precision_recall_fscore_support(y_test,y_pred)

## The number of items used in the iteration is retrieved.
ds_delete = ds[ds.Status == "Delete"]
ds_delete.reset_index(inplace=True, drop=True)

## All the metrics of the confusion matrix are obtained.
metrics = [",".join(map(str, ds_delete['Question'].tolist())),
           accuracy]

for index in range(0, len(name_class)):
    metrics.append(clf_rep[0][index])
    metrics.append(clf_rep[1][index])
    metrics.append(clf_rep[2][index])
    metrics.append(clf_rep[3][index])

## The names of the columns of the dataset are defined.
columns = ['Question', 'Acurracy global']

for index in range(0, len(name_class)):
    columns.append('Precision_' + str(index))
    columns.append('Recall_' + str(index))
    columns.append('F1_score_' + str(index))
    columns.append('Support_' + str(index))

## A new row of the dataset is generated with all the data.
data_metrics = pd.DataFrame([metrics], columns = columns)

## The data is saved to the dataset.
data_metrics.to_csv(path_dataset_qd, mode = 'a', header = False, index = False)

## The data added to the dataset is printed.
data_metrics
```

Out[477]:

	Question	Acurracy global	Precision_0	Recall_0	F1_score_0	Supp
0	2,3,4,5,7,9,11,12,14,15,18,19,21,23,24,25,26,2...	51.282051	0.363636	0.363636	0.363636	

Improvement process of the neural network model.

The application of the explanation method by ALE allows to determine the behavior of the neural network and thereby improve it by discarding non-significant data for it.

Slope Analysis of ALE Plots

In [478]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

We proceed to obtain the graphs to obtain the slopes of the selected classes.

In [479]:
```
## The question metrics dataset is loaded.
dm = pd.DataFrame()

## The number of items used in the iteration is retrieved.
ds_item = ds[ds.Status == "In use"]
ds_item.reset_index(inplace=True, drop=True)

## We get the slopes from the ALE data.
for i in range(df.shape[1]-1):
    structure_data = {'Question': ds_item.loc[i, "Question"]}

    for index in selection_class:
        slope = proba_exp_lr.ale_values [i][1][index] - proba_exp_lr.ale_values [
        structure_data['Slope ' + str(target_names[index])] = slope
        structure_data['Threshold ' + str(target_names[index])] = 'NA'
        structure_data['Anomaly ' + str(target_names[index])] = 'NA'

    dm = dm.append(structure_data, ignore_index=True)

## The current rating data dataset is printed.
dm
```

```
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
```

```
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
```

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
6	17	-0.074225	NA	NA	0.086977	NA	NA
7	20	-0.087052	NA	NA	0.185142	NA	NA
8	22	0.039517	NA	NA	-0.060106	NA	NA
9	28	-0.009943	NA	NA	-0.269066	NA	NA
10	31	0.023127	NA	NA	-0.032159	NA	NA
11	34	-0.344119	NA	NA	0.451545	NA	NA
12	35	0.110226	NA	NA	-0.081951	NA	NA
13	37	-0.226109	NA	NA	-0.296268	NA	NA
14	39	-0.055197	NA	NA	0.003804	NA	NA
15	40	0.061901	NA	NA	-0.042292	NA	NA
16	43	-0.003528	NA	NA	0.004527	NA	NA
17	47	0.185098	NA	NA	-0.313902	NA	NA
18	52	-0.264757	NA	NA	0.042665	NA	NA
19	53	0.000508	NA	NA	0.017378	NA	NA
20	56	0.167221	NA	NA	-0.066015	NA	NA
21	57	-0.140050	NA	NA	0.117020	NA	NA
22	61	0.105674	NA	NA	0.087818	NA	NA
23	64	0.071723	NA	NA	-0.081537	NA	NA
24	65	-0.005747	NA	NA	0.005855	NA	NA
25	67	-0.100421	NA	NA	0.063046	NA	NA
26	69	-0.067829	NA	NA	0.108466	NA	NA
27	71	-0.044020	NA	NA	0.009237	NA	NA
28	73	-0.146033	NA	NA	0.047863	NA	NA
29	74	-0.057533	NA	NA	0.084209	NA	NA
30	76	-0.024292	NA	NA	-0.003752	NA	NA
31	79	0.089584	NA	NA	-0.102379	NA	NA
32	80	0.273549	NA	NA	-0.150538	NA	NA
33	81	-0.303554	NA	NA	0.107592	NA	NA
34	82	-0.278285	NA	NA	0.203135	NA	NA
35	84	0.252204	NA	NA	-0.127769	NA	NA
36	92	-0.090408	NA	NA	0.029823	NA	NA
37	97	-0.198281	NA	NA	0.221906	NA	NA
38	99	0.341788	NA	NA	-0.235594	NA	NA

Slope analysis and threshold application

Slope analysis and threshold application

The slope obtained is analyzed and those that are above or below the positive and negative threshold are selected.

In [480]:
```
for i in range(df.shape[1]-1):
    for index in selection_class:

        ## The slopes are selected according to the values of the thresholds of t
        if dm.loc[i, "Slope " + str(target_names[index])] >= positive_threshold:
            dm.loc[i, "Threshold " + str(target_names[index])] = 1
        elif dm.loc[i, "Slope " + str(target_names[index])] <= negative_threshold
            dm.loc[i, "Threshold " + str(target_names[index])] = 0

## The current rating data dataset is printed.
dm
```

Out[480]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	1	0.037745	1	NA	-0.046383	0	NA
1	6	0.003588	NA	NA	-0.000966	NA	NA
2	8	0.103508	1	NA	-0.091507	0	NA
3	10	-0.040895	0	NA	0.057785	1	NA
4	13	-0.242524	0	NA	0.148865	1	NA
5	16	0.061113	1	NA	-0.048968	0	NA
6	17	-0.074225	0	NA	0.086977	1	NA
7	20	-0.087052	0	NA	0.185142	1	NA
8	22	0.039517	1	NA	-0.060106	0	NA
9	28	-0.009943	NA	NA	-0.269066	0	NA
10	31	0.023127	1	NA	-0.032159	0	NA
11	34	-0.344119	0	NA	0.451545	1	NA
12	35	0.110226	1	NA	-0.081951	0	NA
13	37	-0.226109	0	NA	-0.296268	0	NA
14	39	-0.055197	0	NA	0.003804	NA	NA
15	40	0.061901	1	NA	-0.042292	0	NA
16	43	-0.003528	NA	NA	0.004527	NA	NA
17	47	0.185098	1	NA	-0.313902	0	NA
18	52	-0.264757	0	NA	0.042665	1	NA
19	53	0.000508	NA	NA	0.017378	1	NA
20	56	0.167221	1	NA	-0.066015	0	NA
21	57	-0.140050	0	NA	0.117020	1	NA
22	61	0.105674	1	NA	0.087818	1	NA
23	64	0.071723	1	NA	-0.081537	0	NA
24	65	-0.005747	NA	NA	0.005855	NA	NA
25	67	-0.100421	0	NA	0.063046	1	NA
26	69	-0.067829	0	NA	0.108466	1	NA

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
27	71	-0.044020	0	NA	0.009237	NA	NA
28	73	-0.146033	0	NA	0.047863	1	NA
29	74	-0.057533	0	NA	0.084209	1	NA
30	76	-0.024292	0	NA	-0.003752	NA	NA
31	79	0.089584	1	NA	-0.102379	0	NA
32	80	0.273549	1	NA	-0.150538	0	NA
33	81	-0.303554	0	NA	0.107592	1	NA
34	82	-0.278285	0	NA	0.203135	1	NA
35	84	0.252204	1	NA	-0.127769	0	NA
36	92	-0.090408	0	NA	0.029823	1	NA
37	97	-0.198281	0	NA	0.221906	1	NA
38	99	0.341788	1	NA	-0.235594	0	NA

Analysis and determination of anomalous items

The anomalous items present in the model are determined based on their slope and expected response from the expert.

In [481]:

```
## The dataset of expected responses is loaded.
da = pd.read_csv(path_dataset_qa)

## Delete list of items deleted in this iteration.
i = 0
for x in ds_delete['Question']:
    da.drop(da.index[x - (i + 1)], axis = 0, inplace = True)
    i += 1

## Indexes are restored for later use.
da.reset_index(inplace=True, drop=True)

## The dataset of expected responses is printed.
da
```

Out[481]:

	Number	Question	RE
0	1	Most of the time I have difficulty concentrati...	1
1	6	I consider that my life is full of good things	0
2	8	I'm still as punctual as I have always been	0
3	10	I have found that I can spend a lot of time sc...	1
4	13	I felt more encouraged to do my daily activiti...	1
5	16	I think my thoughts are strange or different f...	1
6	17	I feel guilty about the decisions that I have ...	1
7	20	It is difficult for me to make decisions even ...	1
8	22	It is not difficult for me to understand somet...	0
9	28	The decisions I have made so far have been the...	0
10	31	It is easy for me to choose a photograph of my...	0
11	34	In recent months I have had discussions with m...	1
12	35	I constantly imagine that something will go wr...	1
13	37	I'm not too worried about what might happen in...	0
14	39	Everything will be alright	0
15	40	I can easily blank my mind	0
16	43	Sometimes i feel trapped	1
17	47	I can clearly express to others how I feel	0
18	52	Being locked in an elevator would be the worst...	1
19	53	I'm bothered by people walking slowly in front...	1
20	56	I manage my schedule as I always have	0
21	57	It bothers me to feel that people on the stree...	1
22	61	Sometimes I forget what I wanted to say becaus...	1
23	64	I can put down my cell phone and dedicate myse...	0
24	65	I worry that people will not understand what I...	1
25	67	I get angry easily	1

	Number	Question	RE
26	69	It is not important for me to meet set dates	1
27	71	I use lies just to get out of certain problems	1
28	73	I have difficulty making elaborate plans	1
29	74	People have problems because of themselves	1
30	76	Laws are not as important as others think	1
31	79	It is easy for me to work in a team	0
32	80	It is important to help people when they need it	0
33	81	I have punched someone or thought of doing it	1
34	82	If it was necessary I would pretend to be some...	1
35	84	After an argument I usually go over what happe...	0
36	92	It does not seem serious to me to have some debts	1
37	97	People who commit crimes have their reasons fo...	1
38	99	It is important to respect turns	0

We proceed to identify the anomadic items present in the model present in this iteration.

In [482]:

```
for i in range(df.shape[1]-1):

    ## Local variable to indicate the index of the selected classes.
    i_index = 0

    for index in selection_class:

        ## Determine if it fails based on the expected response parameters of the
        if dm.loc[i, "Threshold "  + str(target_names[index])] == 1 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 1 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 0 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 0 and da.loc[i,
            dm.loc[i, "Anomaly "  + str(target_names[index])] = expected_response

        ## Increase of the index of the selected classes.
        i_index += 1

## The final grade data dataset is printed.
dm
```

Out[482]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	1	0.037745	1	1	-0.046383	0	1
1	6	0.003588	NA	NA	-0.000966	NA	NA
2	8	0.103508	1	0	-0.091507	0	0
3	10	-0.040895	0	0	0.057785	1	0
4	13	-0.242524	0	0	0.148865	1	0
5	16	0.061113	1	1	-0.048968	0	1
6	17	-0.074225	0	0	0.086977	1	0
7	20	-0.087052	0	0	0.185142	1	0
8	22	0.039517	1	0	-0.060106	0	0
9	28	-0.009943	NA	NA	-0.269066	0	0
10	31	0.023127	1	0	-0.032159	0	0
11	34	-0.344119	0	0	0.451545	1	0
12	35	0.110226	1	1	-0.081951	0	1
13	37	-0.226109	0	1	-0.296268	0	0
14	39	-0.055197	0	1	0.003804	NA	NA
15	40	0.061901	1	0	-0.042292	0	0
16	43	-0.003528	NA	NA	0.004527	NA	NA
17	47	0.185098	1	0	-0.313902	0	0
18	52	-0.264757	0	0	0.042665	1	0
19	53	0.000508	NA	NA	0.017378	1	0

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
20	56	0.167221	1	0	-0.066015	0	0
21	57	-0.140050	0	0	0.117020	1	0
22	61	0.105674	1	1	0.087818	1	0
23	64	0.071723	1	0	-0.081537	0	0
24	65	-0.005747	NA	NA	0.005855	NA	NA
25	67	-0.100421	0	0	0.063046	1	0
26	69	-0.067829	0	0	0.108466	1	0
27	71	-0.044020	0	0	0.009237	NA	NA
28	73	-0.146033	0	0	0.047863	1	0
29	74	-0.057533	0	0	0.084209	1	0
30	76	-0.024292	0	0	-0.003752	NA	NA
31	79	0.089584	1	0	-0.102379	0	0
32	80	0.273549	1	0	-0.150538	0	0
33	81	-0.303554	0	0	0.107592	1	0
34	82	-0.278285	0	0	0.203135	1	0
35	84	0.252204	1	0	-0.127769	0	0
36	92	-0.090408	0	0	0.029823	1	0
37	97	-0.198281	0	0	0.221906	1	0
38	99	0.341788	1	0	-0.235594	0	0

Removing identified anomadic items from the model

We proceed to eliminate the identified anomadic items so as not to use them in the next iteration.

In [483]:
```
## Local variable for the elimination of the anomaly items.
instructions = ''
init = False
question_anomaly = []

if selection_mode == 1:

    ## Items that fail in the defined class are retrieved.
    dm_delete = dm[dm['Anomaly ' + str(target_names[select_class_mode])] == 1]
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 2:

    ## Items that fail in classes defined with the or condition type are retrieve
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 1"
            init = True
        else:
            instructions += " or `Anomaly " + str(target_names[index]) + "` == 1'

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 3:

    ## Items that fail in classes defined with the and condition type are retriev
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 1"
            init = True
        else:
            instructions += " and `Anomaly " + str(target_names[index]) + "` == 1

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 4:

    ## Out-of-threshold items are retrieved in classes defined with the conditior
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 'NA'"
            init = True
        else:
            instructions += " and `Anomaly " + str(target_names[index]) + "` == '

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

## The selected blank items are printed according to the selection_mode.
question_anomaly
```

Out[483]: [1, 16, 35]

Model explainability process

The data used for the test of the neural network model is shown.

In [484]: X_test

Out[484]:

	1. Most of the time I have difficulty concentrating on simple tasks	6. I consider that my life is full of good things	8. I'm still as punctual as I have always been	10. I have found that I can spend a lot of time scrolling the screen \nof my cell phone without searching or stopping at anything in particular	13. I felt more encouraged to do my daily activities before	16. I think my thoughts are strange or different from before	17. I feel guilty about the decisions that I have made	20. It is difficult for me to make decisions even if they are simple	22 di un s
69	1	1	0	0	0	0	0	1	
46	1	1	1	1	1	1	1	0	
58	0	1	1	0	1	1	0	0	
114	0	1	1	0	0	1	1	0	
73	0	1	1	1	0	0	0	0	
98	1	0	1	1	0	1	0	1	
31	0	1	0	0	0	0	1	0	
53	1	1	1	1	1	1	1	0	
65	0	1	1	0	1	0	0	0	
96	0	1	1	1	0	0	1	0	
95	0	1	0	1	0	1	0	1	
97	0	1	1	0	0	0	1	1	
2	0	1	0	1	0	1	0	0	
62	0	1	0	1	0	1	0	1	
110	0	0	1	0	1	0	0	0	
55	0	1	1	0	0	1	1	1	
103	1	1	0	1	0	0	0	1	
100	1	1	0	0	1	0	0	1	
66	1	0	0	1	0	1	1	0	
44	0	1	1	0	0	0	0	0	
77	0	1	1	1	0	1	1	1	
17	1	1	1	1	1	1	0	1	
81	1	1	1	0	1	1	1	0	
74	1	0	1	1	1	1	1	1	

56	0	1	1	0	1	1	1	0
94	1	0	0	1	1	1	1	0
35	1	1	1	0	0	1	0	1
38	0	1	1	0	0	1	0	1
93	0	1	1	1	1	1	0	1
48	1	1	0	1	1	1	1	1
33	0	1	0	1	1	1	1	0
59	0	1	1	1	0	0	1	0
92	1	0	1	0	0	1	1	1
42	1	1	1	0	0	1	0	0
10	0	1	0	1	0	1	0	0
85	1	1	1	1	1	1	1	0
32	0	1	0	1	1	1	1	1
78	1	1	1	0	0	1	0	1
19	0	0	1	0	1	1	0	0

39 rows × 39 columns

Explanation of the model using ALE

In analysis of the model through the results obtained by ALE, it is possible to identify the behavior of the neural network.

In [485]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

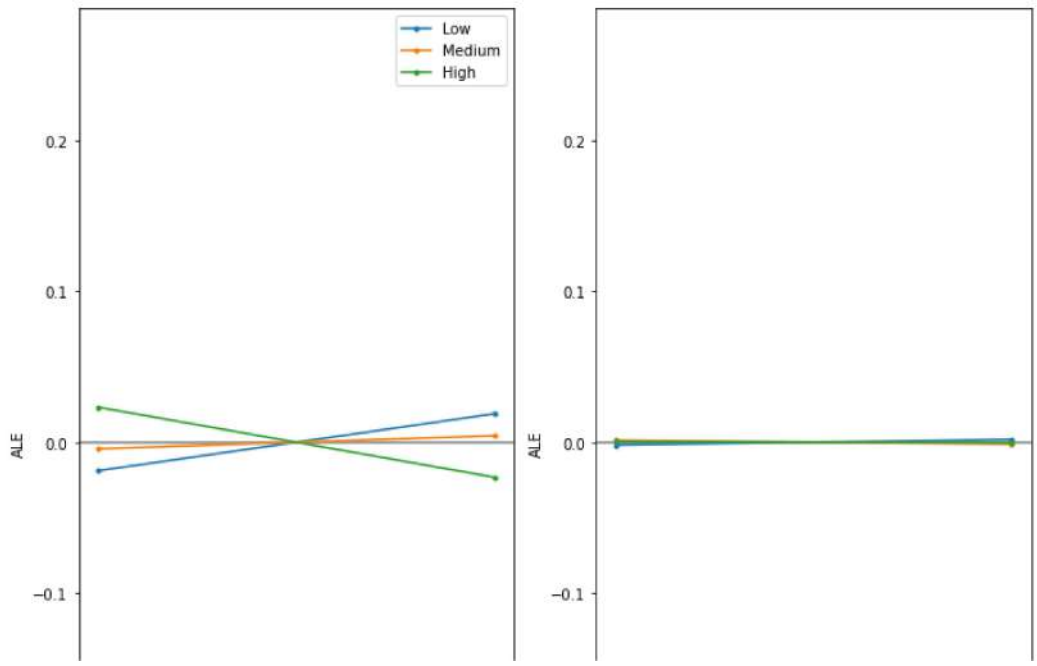
```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

The graphs of all the data present in the dataset used are shown.

In [486]:
```
plot_ale(proba_exp_lr, n_cols=2, features=list(range(df.shape[1]-1)),fig_kw={'fig
```



Explanation of the model using LIME

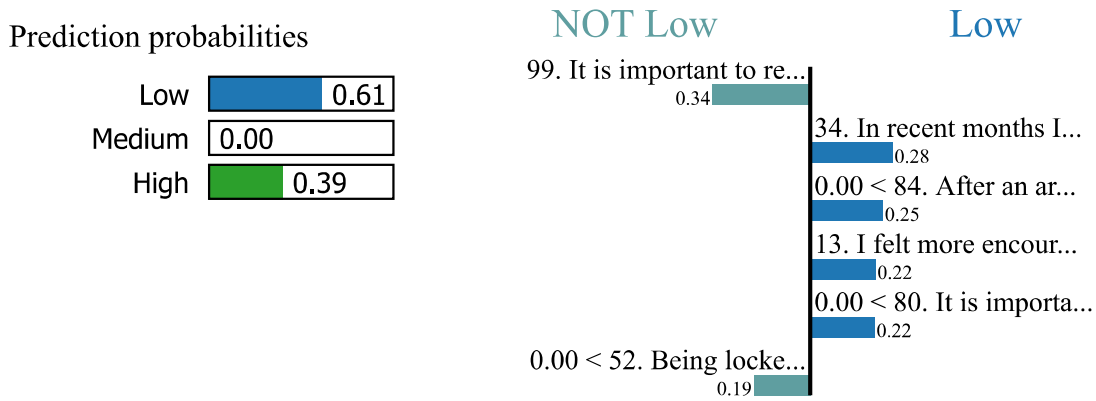
We proceed to explain the improved model using LIME for the Low, Medium and High classes.

In [487]:
```
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorica
                                                   feature_names=train_cols, clas
                                                   discretize_continuous=True)
```

The first individual present in the test data is explained.

In [488]:
```
exp = explainer.explain_instance(X_test.to_numpy()[0], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

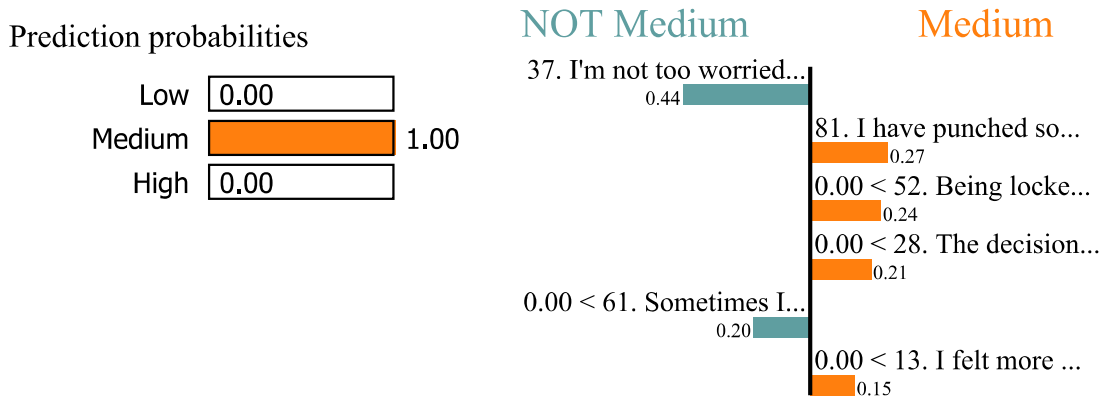


Feature	Value
99. It is important to respect turns	0.00
34. In recent months I have had discussions with my schoolmates or colleagues	0.00
84. After an argument I usually go over what happened in my head	1.00
13. I felt more encouraged to do my daily activities before	0.00
80. It is important to help people when they need it	1.00
52. Being locked in an elevator would be the worst thing that could happen to me	1.00

The second individual present in the test data is explained.

In [489]:
```
exp = explainer.explain_instance(X_test.to_numpy()[1], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

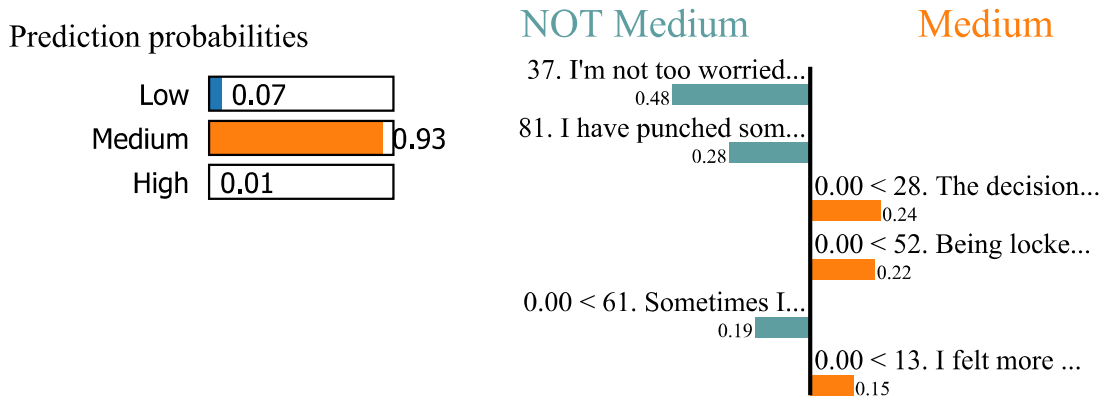


Feature	Value
37. I'm not too worried about what might happen in a few weeks	0.00
81. I have punched someone or thought of doing it	1.00
52. Being locked in an elevator would be the worst thing that could happen to me	1.00
28. The decisions I have made so far have been the right ones	1.00
61. Sometimes I forget what I wanted to say because I have several thoughts at the same time	1.00
13. I felt more encouraged to do my daily activities before	1.00

The last individual present in the test data is explained.

In [490]:
```
exp = explainer.explain_instance(X_test.to_numpy()[-1], mlp.predict_proba, num_fe
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```



Feature	Value
37. I'm not too worried about what might happen in a few weeks	0.00
81. I have punched someone or thought of doing it	0.00
28. The decisions I have made so far have been the right ones	1.00
52. Being locked in an elevator would be the worst thing that could happen to me	1.00
61. Sometimes I forget what I wanted to say because I have several thoughts at the same time	1.00
13. I felt more encouraged to do my daily activities before	1.00

Explanation of the model using SHAP

SHAP explanatory values are displayed according to the class to which the selected individual belongs according to the prediction of the model.

The first individual present in the test data is explained.

In [491]:

```
## Individual Selection.
individual = X_test.iloc[[0]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plo
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The second individual present in the data test is explained.

In [492]:

```
## Individual Selection.
individual = X_test.iloc[[1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The last individual present in the data test is explained.

In [493]:
```
## Individual Selection.
individual = X_test.iloc[[-1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plo
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

Explanation using heatmaps

The importance of the data present in the neural network is shown from explanation methods.

In [494]:

```
## Items currently in use are retrieved.
ds_item = ds[ds.Status == "In use"].reset_index(drop=True)

## The data is prepared for the generation of heatmaps.
column_Map = ds_item['Question'].tolist()
content_Map = [False for x in range(1, df.shape[1])]

## The next items to be deleted will be marked.
for index in range(0, len(content_Map)):

    if iteration == 0:
        for delete in list_contr:
            if column_Map[index] == delete:
                content_Map [index] = True
    else:
        for delete in question_anomaly:
            if column_Map[index] == delete:
                content_Map [index] = True

## The dataframe is created with the prepared data.
df_mask = pd.DataFrame([content_Map], columns = column_Map)
df_mask_imp = pd.concat([df_mask, df_mask, df_mask], ignore_index = True, axis =
df_mask_all = pd.concat([df_mask, df_mask, df_mask, df_mask], ignore_index = True

df_mask
```

Out[494]:

	1	6	8	10	13	16	17	20	22	28	...	74	76	79	80
0	True	False	False	False	False	True	False	False	False	False	...	False	False	False	False

1 rows × 39 columns

Identification of feature variables

Identification of feature variables from the randomness of the answers as items of the data.

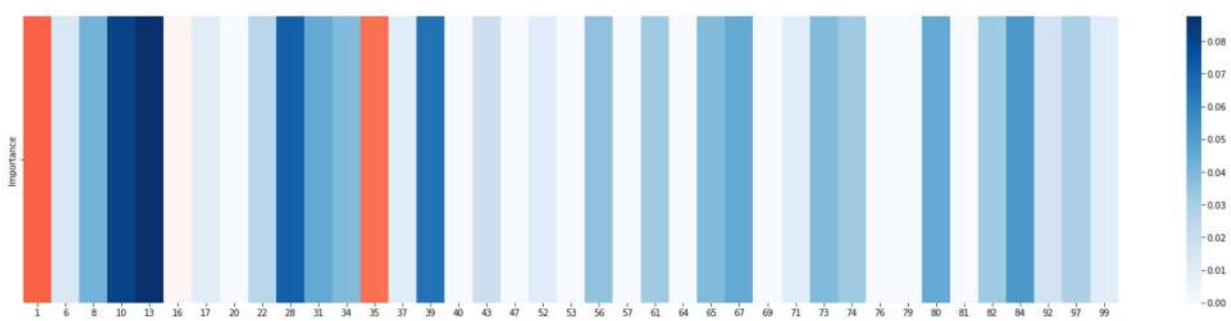
In [495]:
```
## Loading the data for explainability.
explainer = ClassifierExplainer(mlp, X_test, y_test)

## Class index local variable.
iterable_class = -1
map_class = []
```

```
WARNING: Parameter shap='guess', but failed to guess the type of shap explainer
to use for MLPClassifier. Defaulting to the model agnostic shap.KernelExplainer
(shap='kernel'). However this will be slow, so if your model is compatible with
e.g. shap.TreeExplainer or shap.LinearExplainer then pass shap='tree' or shap
='linear'!
WARNING: For shap='kernel', shap interaction values can unfortunately not be ca
lculated!
Note: shap values for shap='kernel' normally get calculated against X_backgroun
d, but paramater X_background=None, so setting X_background=shap.sample(X, 5
0)...
Generating self.shap_explainer = shap.KernelExplainer(model, X, link='identit
y')
```

Explainability of the data belonging to the first class, in this case the Low.

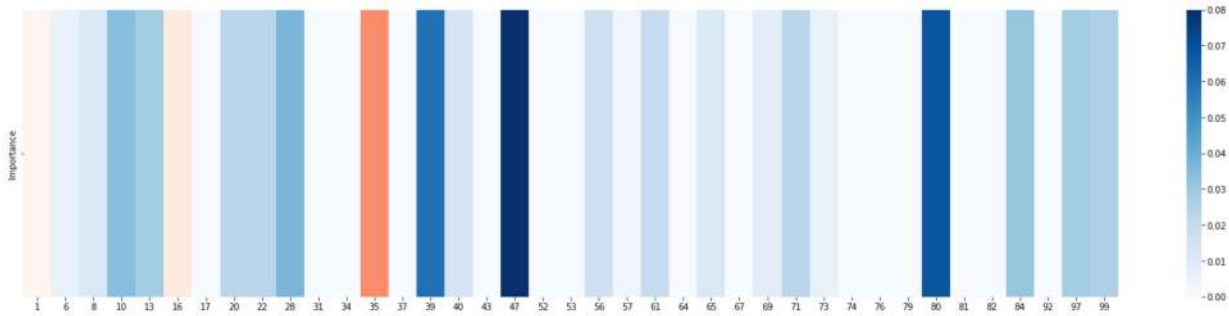
In [496]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

```
Calculating permutation importances (if slow, try setting n_jobs parameter)...
```



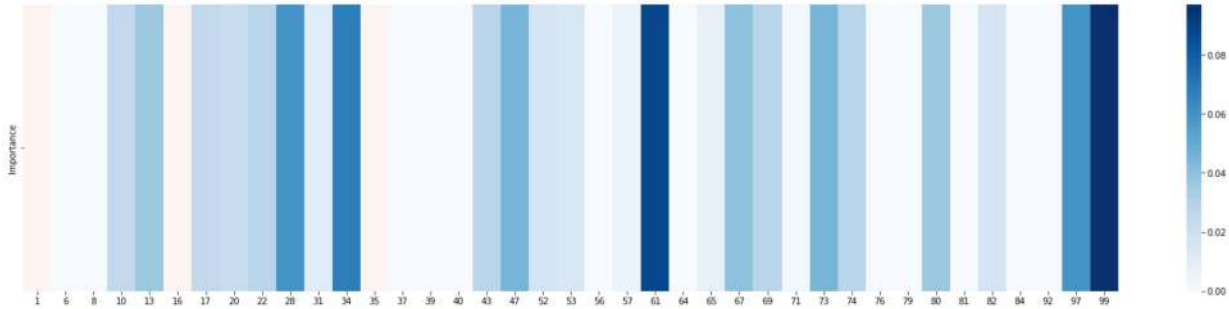
Explainability of the data belonging to the second class, in this case the Medium.

In [497]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



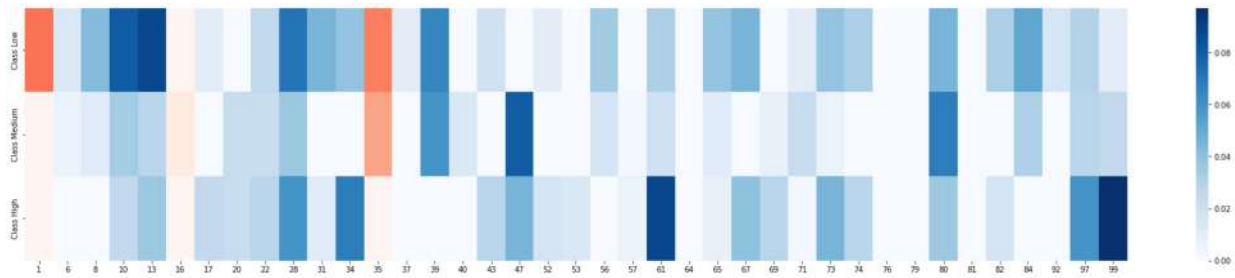
Explainability of the data belonging to the third class, in this case the High.

In [498]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [499]:
```
## Fusion of the previous heatmaps.
df_importance = pd.concat(map_class, axis=1)
df_importance.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_importance.transpose(), cmap = "Reds", cbar = False)
sns.heatmap(df_importance.transpose(), cmap = "Blues",yticklabels = True, xtickla
plt.show()
```



Identification of feature variables using SHAP

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of SHAP.

In [500]:
```
## Class index local variable.
iterable_class = -1
map_class = []
```

Individual explainability of the data belonging to the first class, in this case the Low.

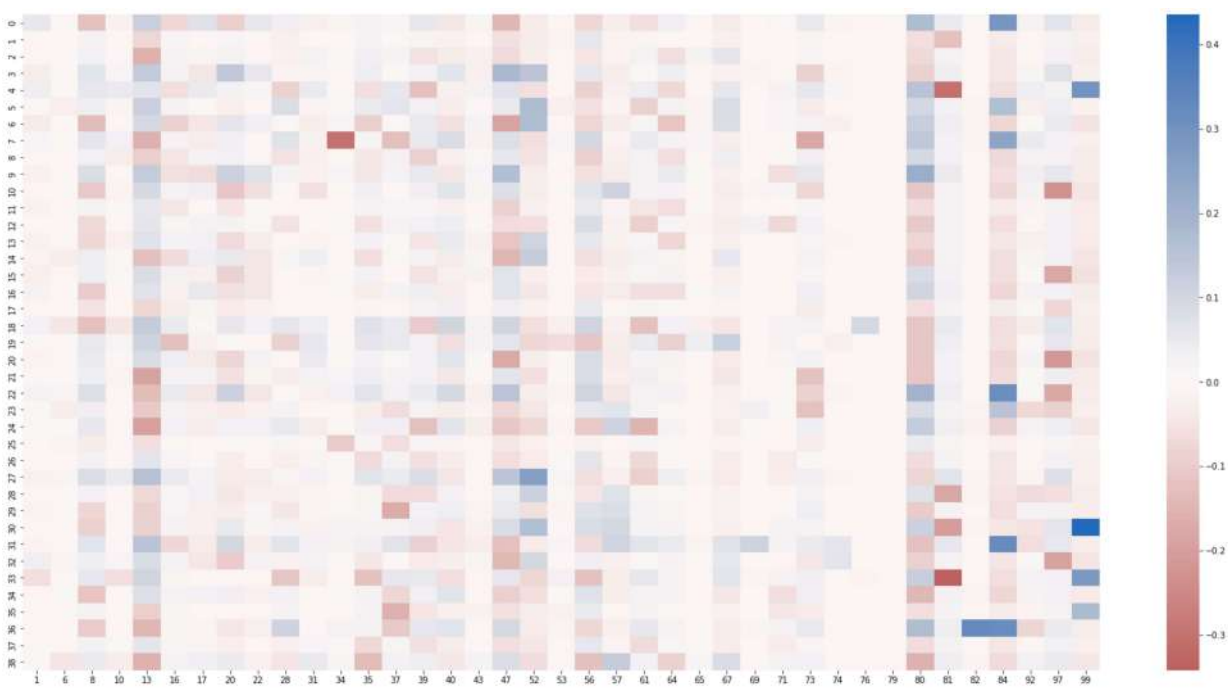
Definition of explainability data.

In [501]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

```
Calculating shap values...

  0%|          | 0/39 [00:00<?, ?it/s]
```

Generation of the heatmap.

In [502]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

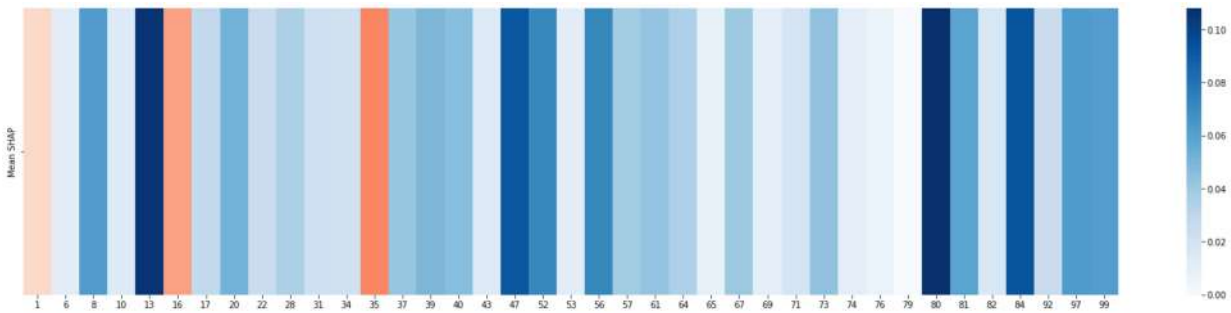


Fusion of the data used.

In [503]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [504]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



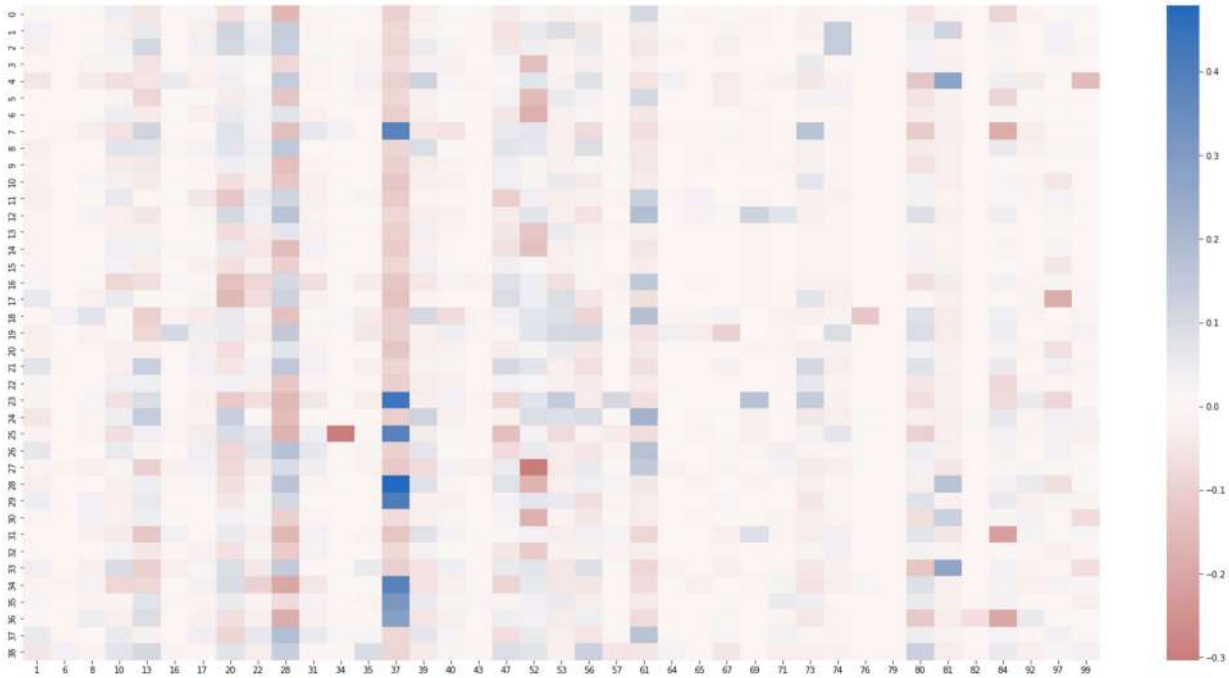
Individual explainability of the data belonging to the second class, in this case the Medium.

Definition of explainability data.

In [505]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [506]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

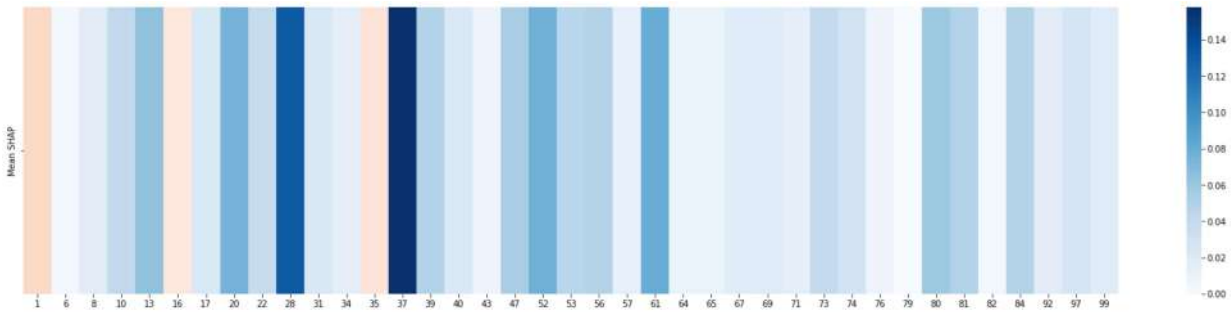


Fusion of the data used.

In [507]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [508]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



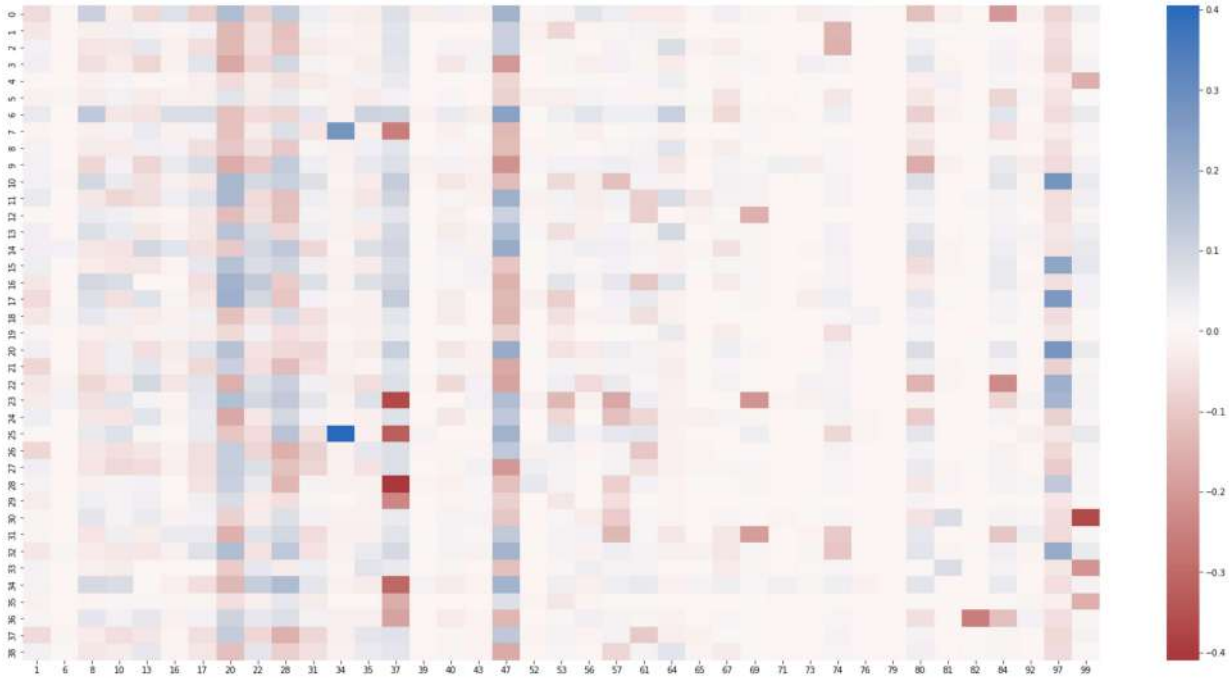
Individual explainability of data belonging to the third class, in this case High.

Definition of explainability data.

In [509]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [510]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

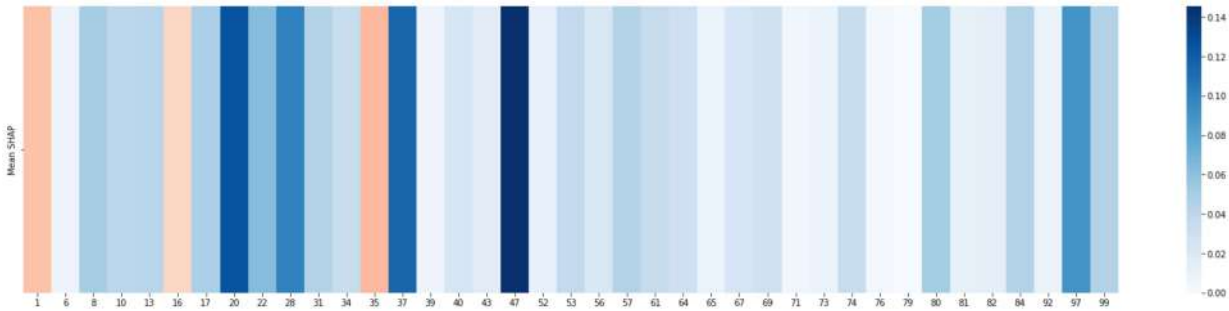


Fusion of the data used.

In [511]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

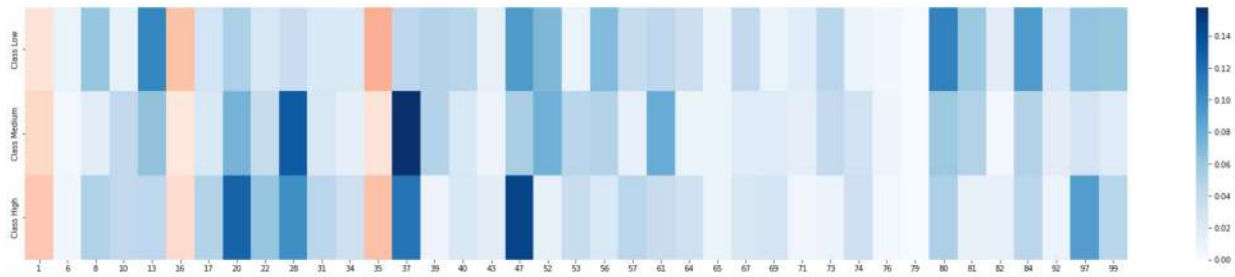
Generation of the heatmap.

In [512]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [513]:
```
## Fusion of the previous heatmaps.
df_shap = pd.concat(map_class, axis=1)
df_shap.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_shap.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_shap.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using LIME

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of LIME.

In [514]:
```
## Data preparation for explainability.
lime_exp = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorical
                                                  feature_names=train_cols, class
                                                  discretize_continuous=True)

## Class index local variable.
iterable_class = -1
map_class = []
```

Obtaining the data for the generation of the explanations by LIME about the Low, Medium and High classes.

In [515]:
```
## Local variables for LIME array data.
exp_matrix = [[] for _ in range(len(name_class))]

## Recovering the data needed for LIME.
for x in X_test.to_numpy():

    ## Auxiliary temporary variables.
    exp_list = [[] for _ in range(len(name_class))]

    ## Loading the data for explainability.
    exp = lime_exp.explain_instance(x, mlp.predict_proba, num_features = df.shape

    ## Recovering the data.
    for elements in range(0, len(name_class)):
        temp = exp.as_map()[elements]
        temp.sort(key=itemgetter(0))

        ## Saving data of the Low, Medium and High classes.
        for tup in temp:
            exp_list[elements].append(tup[1])

        exp_matrix[elements].append(exp_list[elements])
```

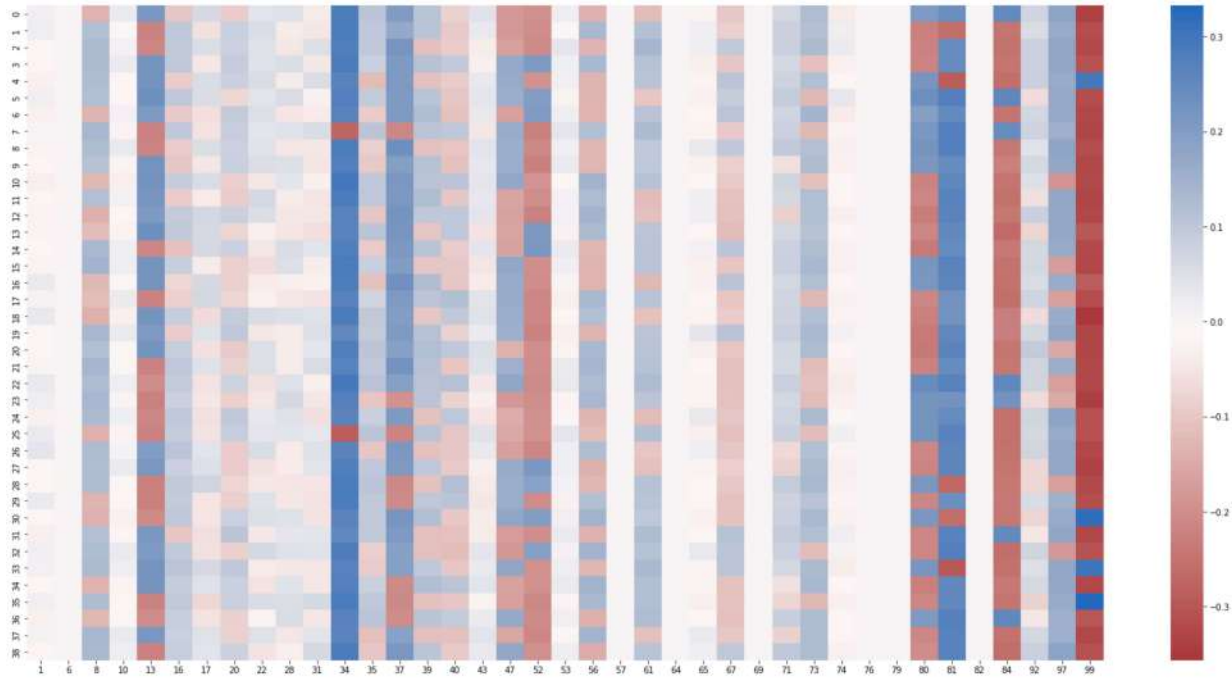
```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

Individual explainability of the data belonging to the first class, in this case the Low.

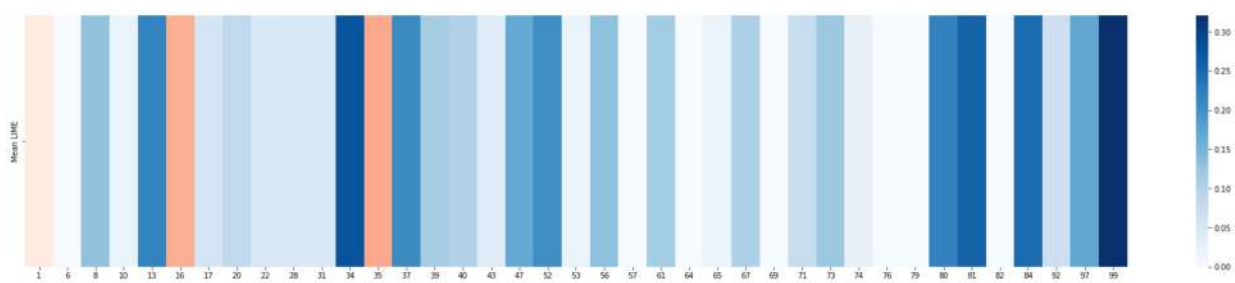
Generation of the heatmap.

In [516]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

In [517]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

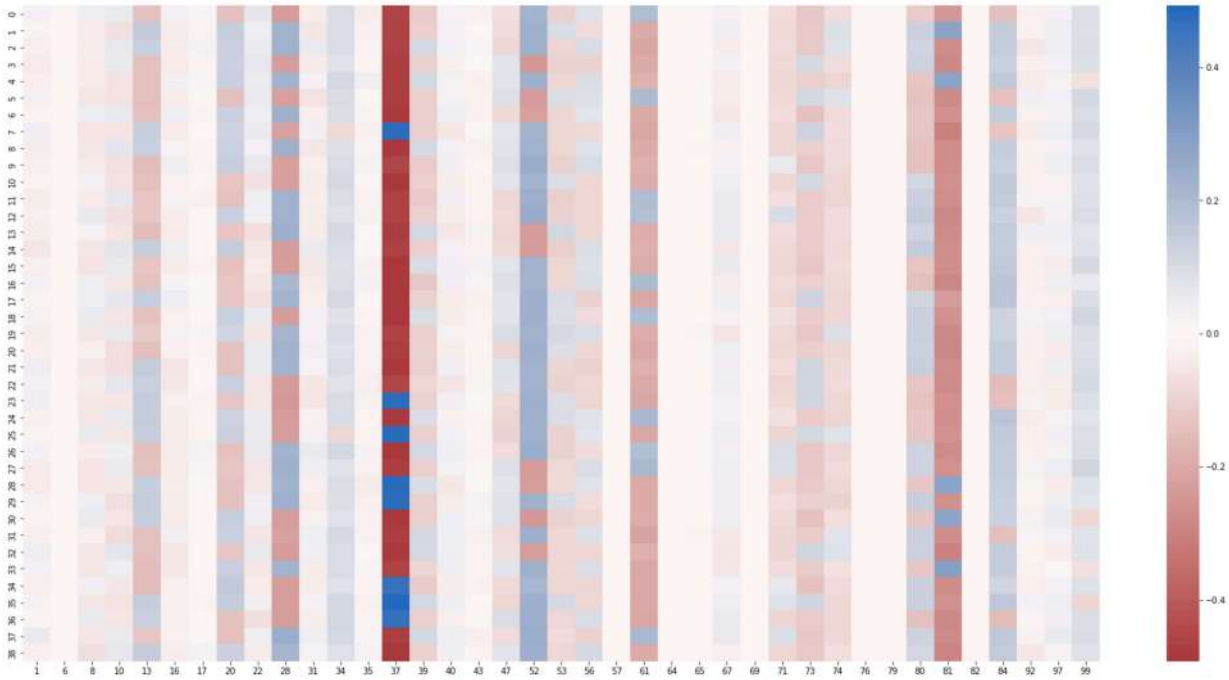


Individual explainability of the data belonging to the second class, in

this case the Medium.

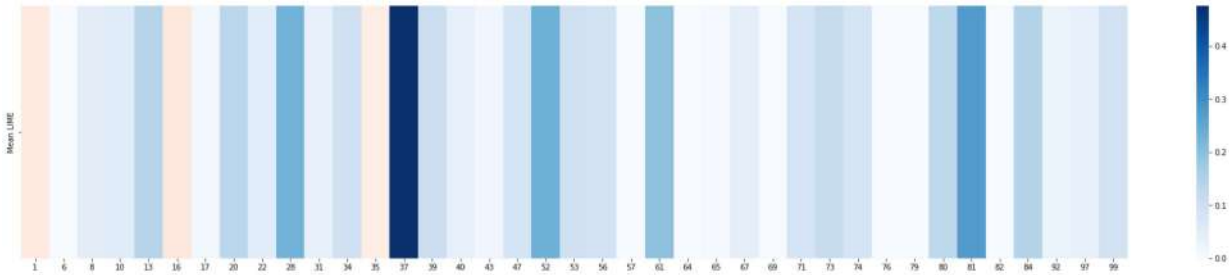
Generation of the heatmap.

In [518]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

In [519]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

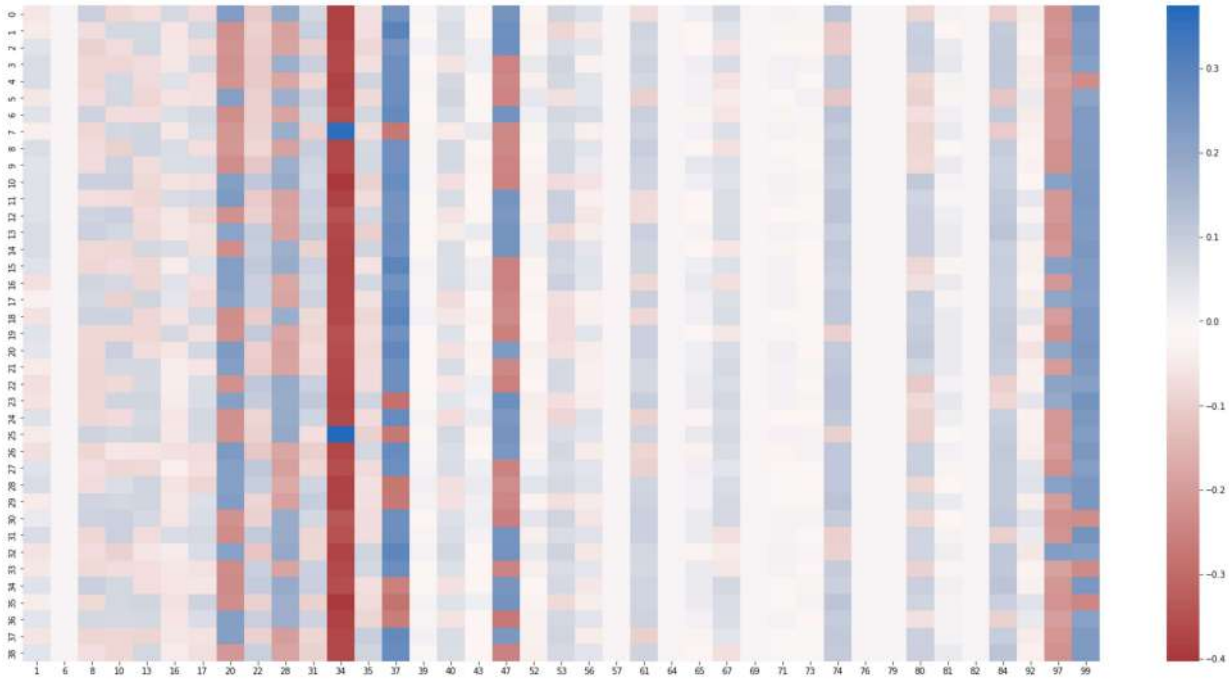


Individual explainability of data belonging to the third class, in this

case High.

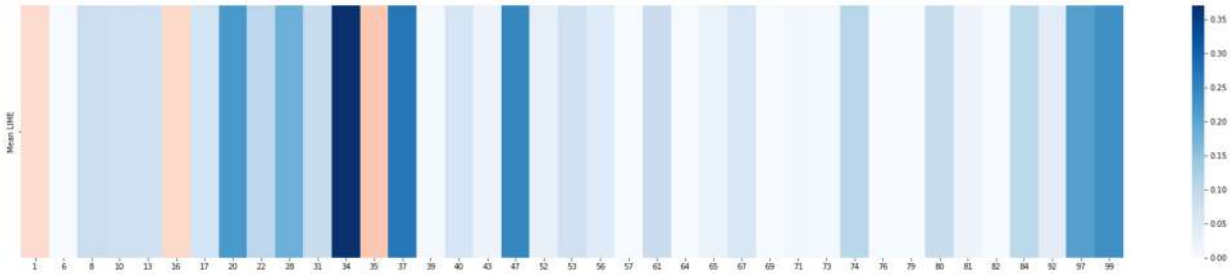
Generation of the heatmap.

In [520]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



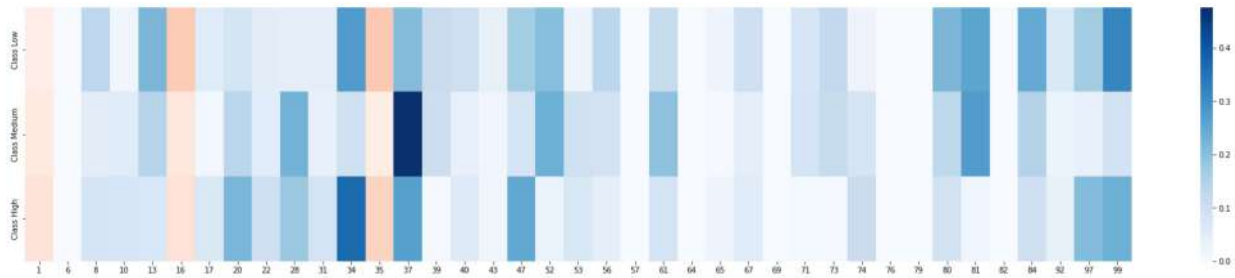
Fusion of the data used.

In [521]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [522]:
```
df_lime = pd.concat(map_class, axis=1)
df_lime.columns = name_class
plt.figure(figsize = (32,6))
sns.heatmap(df_lime.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_lime.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using ALE

It shows the main effect of each question compared to the probability that the model predicts in each class. This effect is shown for both responses. Since there are only 2 answers, the map is the same but with the colors reversed.

In [523]:
```
## Definition of explainability data.
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names=train_cols, target_names= target_r
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())

## Class index local variable.
iterable_class = -1
map_class = []
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

Obtaining the data for the generation of the explanations by ALE about all the classes.

In [524]:
```
## Local variables for ALE array data.
ale_list = [[] for _ in range(len(name_class))]

## Recovering the data needed for ALE.
for array in proba_exp_lr.ale_values:

    ## Recovering the data.
    for elements in range(0, len(name_class)):

        ## Saving data of all the classes.
        ale_list[elements].append(array[0][elements])
```

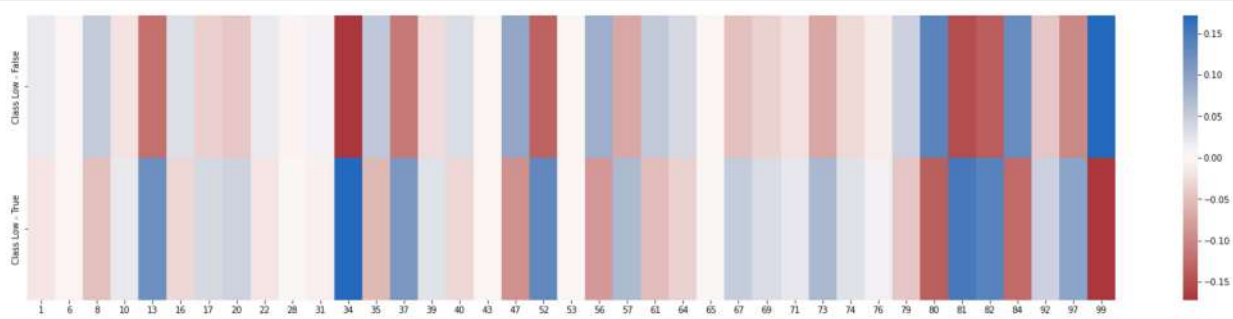
Data processing for the explainability of ALE.

In [525]:
```
## Data processing of the Low, Medium and High classes.
for elements in range(0, len(name_class)):
    ale_df = pd.DataFrame([ale_list [elements]])
    ale_df = pd.concat([ale_df.multiply(-1), ale_df])
    ale_df.index = [name_class [elements] + " - False", name_class [elements] + '
    ale_df.columns = column_Map
    map_class.append(ale_df)
```

Explainability of the data belonging to the first class, in this case the Low.

Generation of the heatmap.

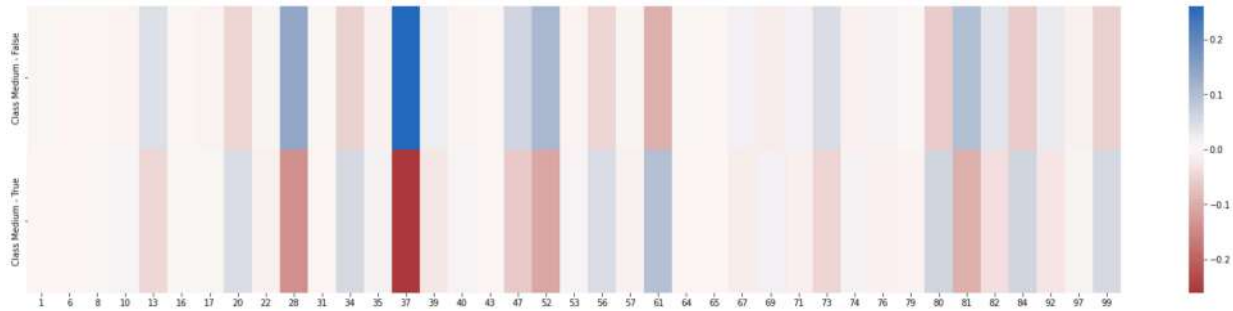
In [526]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the second class, in this case the Medium.

Generation of the heatmap.

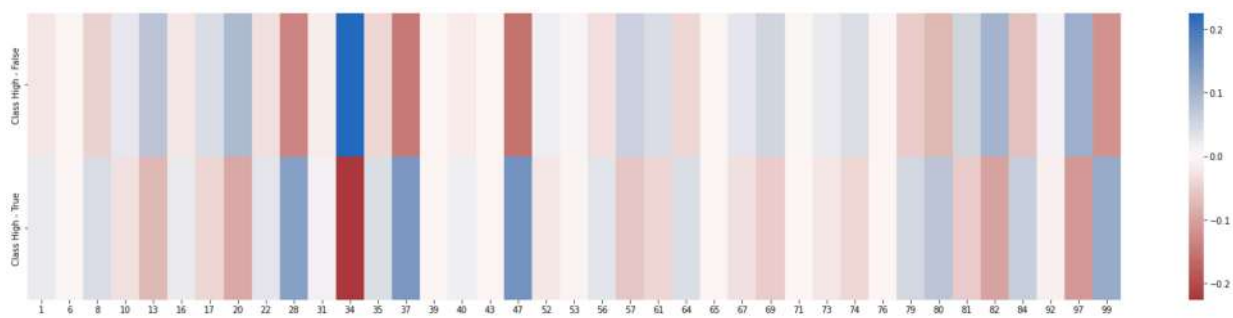
In [527]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the third class, in this case the High.

Generation of the heatmap.

In [528]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```

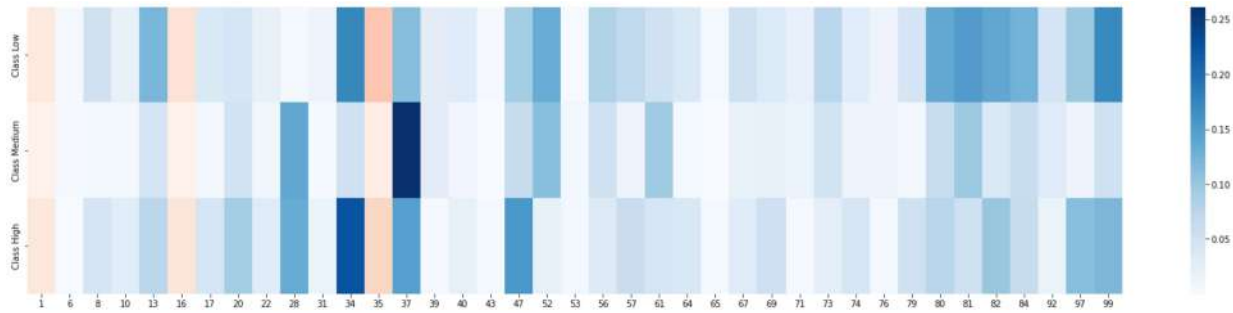


Fusion of the data used in all classes.

In [529]:
```
## Application of absolute value to data.
for index in range(0, len(map_class)):
    map_class [index] = map_class [index][0:1].abs()

## Union of the heatmaps of the Low, Medium and High class.
ale_df = pd.concat(map_class)
ale_df.index = name_class
ale_df = ale_df.transpose()
```

Generation of the heatmap.

In [530]:
```
plt.figure(figsize = (28,6))
sns.heatmap(ale_df.transpose(), cmap="Reds", cbar = False)
sns.heatmap(ale_df.transpose(), cmap="Blues", yticklabels = True, xticklabels = T
plt.show()
```



Comparing of Feature Variables Methods for Explainability

The global values of each of the methods used are taken, their maximums are obtained (the minimums are 0) and a percentage is calculated based on said maximum, which indicates how relevant the contribution is for each question.

In [531]:
```
## Class index local variable.
iterable_class = -1
```

Comparison of the explanation methods of the first class, in this case the Low.

The highest data of each method used is obtained.

In [532]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [533]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

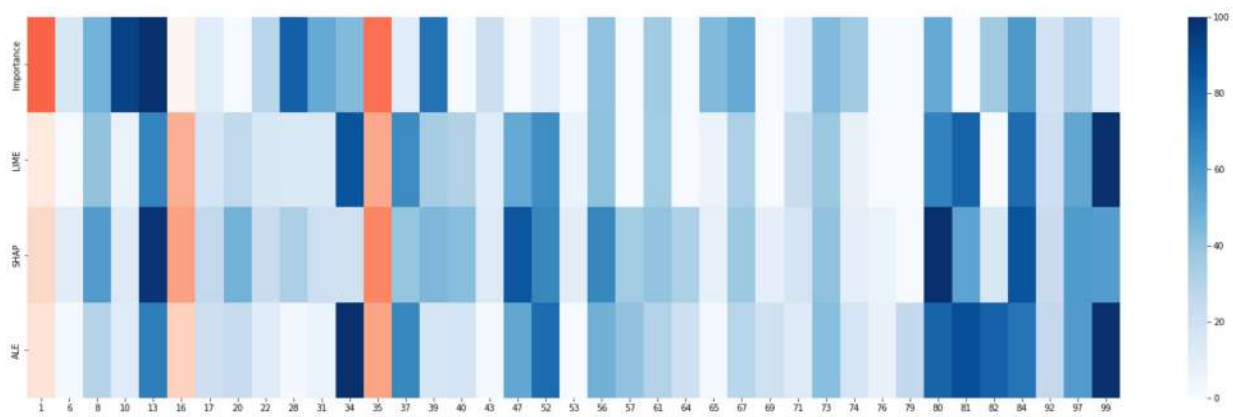
Out[533]:

	1	6	8	10	13	16	17	2
Importance	51.851852	14.814815	48.148148	92.592593	100.000000	0.000000	11.111111	0.00000
LIME	6.157783	0.000000	40.237372	6.037952	67.882377	29.037407	16.954783	26.75818
SHAP	15.148045	9.821002	58.051343	12.742956	98.137932	32.954290	26.465439	47.88884
ALE	10.968671	1.042694	30.079141	11.883906	70.476984	17.759348	21.569507	25.29700

4 rows × 39 columns

Generation of the heatmap.

In [534]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the second class, in this case the Medium.

The highest data of each method used is obtained.

In [535]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [536]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

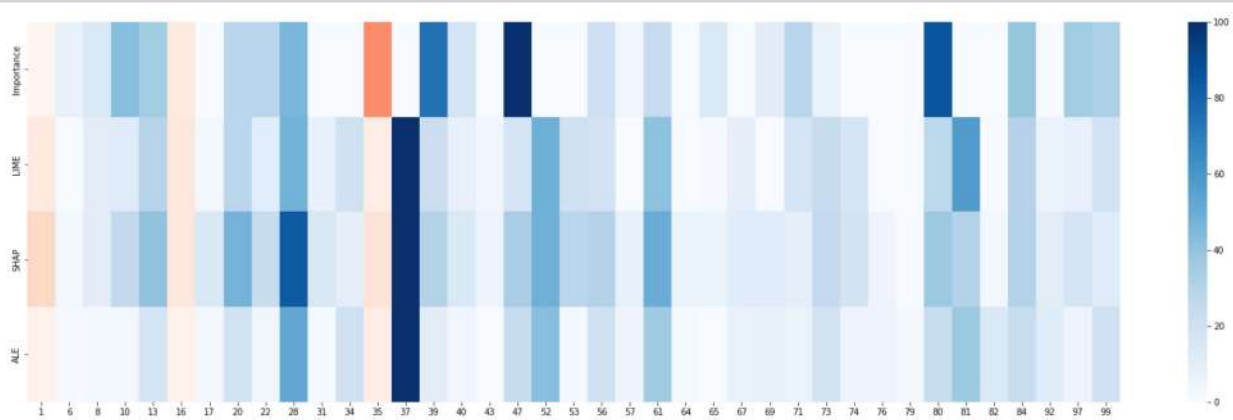
Out[536]:

	1	6	8	10	13	16	17	20
Importance	0.000000	7.142857	14.285714	42.857143	35.714286	7.142857	0.000000	28.571429
LIME	6.778573	0.000000	9.986993	12.441946	29.778895	8.035598	2.742721	28.279027
SHAP	15.389313	3.386941	10.818846	26.139094	40.441379	8.407380	14.227627	47.330481
ALE	1.653630	0.501886	2.297391	3.233282	17.929423	2.324962	2.441195	18.777744

4 rows × 39 columns

Generation of the heatmap.

In [537]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the third class, in this case the High.

The highest data of each method used is obtained.

In [538]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [539]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

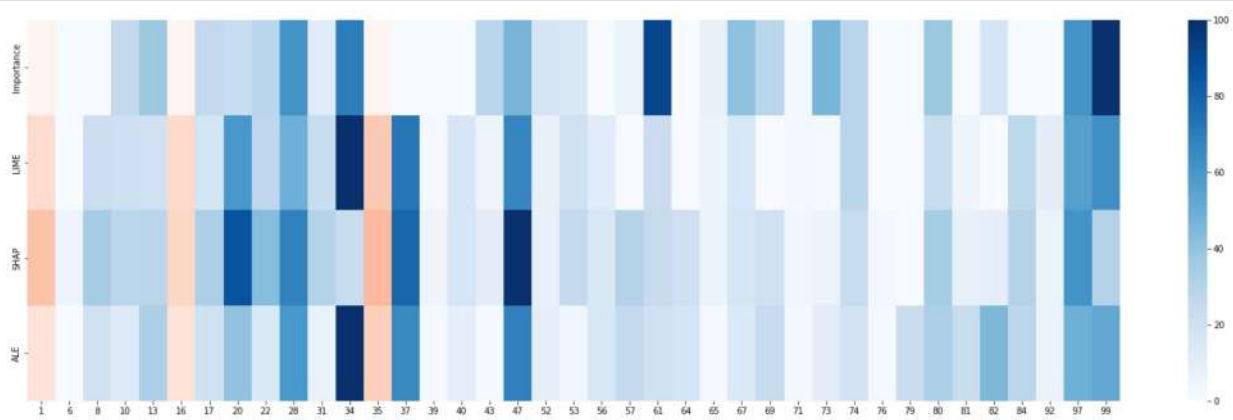
Out[539]:

	1	6	8	10	13	16	17	20
Importance	0.000000	0.000000	0.000000	26.470588	38.235294	0.000000	26.470588	23.529412
LIME	14.041889	0.000000	22.040556	21.216401	20.572239	14.841331	18.216855	59.518394
SHAP	22.880759	4.582268	34.538402	28.491582	29.279675	15.638813	32.962197	86.871853
ALE	10.272169	0.214014	20.265306	12.797109	32.968015	10.844590	19.262092	41.001995

4 rows × 39 columns

Generation of the heatmap.

In [540]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Safeguarding the items eliminated during this iteration.

In [541]:
```
## Removal of anomalous items selected by selection_mode.
for x in question_anomaly:

    ## Status change of the deleted items in the respective auxiliary dataset.
    ds.loc[x - 1,'Status'] = 'Delete'

## Save the changed statuses in the dataset.
ds.to_csv(path_dataset_qs, index = False)
```

Iteration 7

The first iteration of improvement of the model is carried out.

Preparing the data for the neural network

The data is prepared in order to be processed by the neural network.

In [542]:

```
## The iteration number is increased.
iteration += 1

## Loading data from the dataset.
df = pd.read_csv(path_dataset)
ds = pd.read_csv(path_dataset_qs)

## Elimination of unrelated items from the dataset.
if len(unrelated_questions) != 0:
    df = df.drop(df.columns[unrelated_questions[0]:unrelated_questions[1]], axis=

i = 0
if iteration == 1:
    ## Elimination of list of conflicting items declared in global variables.
    for x in list_contr:
        df.drop(df.columns[x - (i + 1)], axis = 1, inplace = True)
        i += 1
        ## Status change of the deleted question in the respective auxiliary data
        ds.loc[x - 1,'Status'] = 'Delete'

else:
    ## Elimination of the list of anomadic items defined in the improvement proce
    ds_delete = ds[ds.Status == "Delete"].reset_index(drop=True)
    for x in ds_delete['Question'].tolist():
        df.drop(df.columns[x - (i + 1)], axis = 1, inplace = True)
        i += 1

## Grouping of the target according to the defined dictionary.
if grouping_name != None and grouping_name != '':
    df[grouping_name] = df[grouping_name].map(class_dic)

## Assignment of local variables according to the data necessary for the neural r
train_cols = df.columns [0:-1]
label = df.columns [-1]
X = df [train_cols]
y = df [label]

## The prepared data of the dataset is printed.
df
```

Out[542]:

	6. I consider that my life is full of good things	8. I'm still as punctual as I have always been	10. I have found that I can spend a lot of time scrolling the screen \nof my cell phone without searching or stopping at anything in particular	13. I felt more encouraged to do my daily activities before	17. I feel guilty about the decisions that I have made	20. It is difficult for me to make decisions even if they are simple	22. It is not difficult for me to understand something the first time	28. The decisions I have made so far have been the right ones	e t p
0	1	1	0	0	0	1	1	0	
1	1	1	0	1	1	1	1	0	
2	1	0	1	0	0	0	1	1	
3	1	1	0	1	0	1	1	1	
4	1	1	1	0	1	1	0	0	
...	...	...	...	...	...	...	...	...	
99	1	1	1	1	0	1	0	1	
100	1	0	0	1	0	1	0	1	
101	1	0	1	1	0	0	0	0	
102	1	1	1	0	1	1	1	0	
103	1	0	1	0	0	1	0	1	

104 rows × 37 columns

Oversample application

An oversample process is applied to level the amount of existing data given by the grouping.

In [543]:
```
## The number of elements present in the grouped target is printed.
y.value_counts()
```

Out[543]:
```
0    39
1    35
2    30
Name: Target, dtype: int64
```

Oversample application.

In [544]:
```
## Application of the oversample to the prepared data.
X, y = oversample.fit_resample(X, y)
```

Printing of the data after the application of the oversample.

In [545]:
```
## The results obtained after applying the oversample are printed.
y.value_counts()
```

Out[545]:
```
0    39
1    39
2    39
Name: Target, dtype: int64
```

Preparation of the parameters of the neural network

We proceed to define the parameters necessary for the training of the neural network.

In [546]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
```

Neural network training

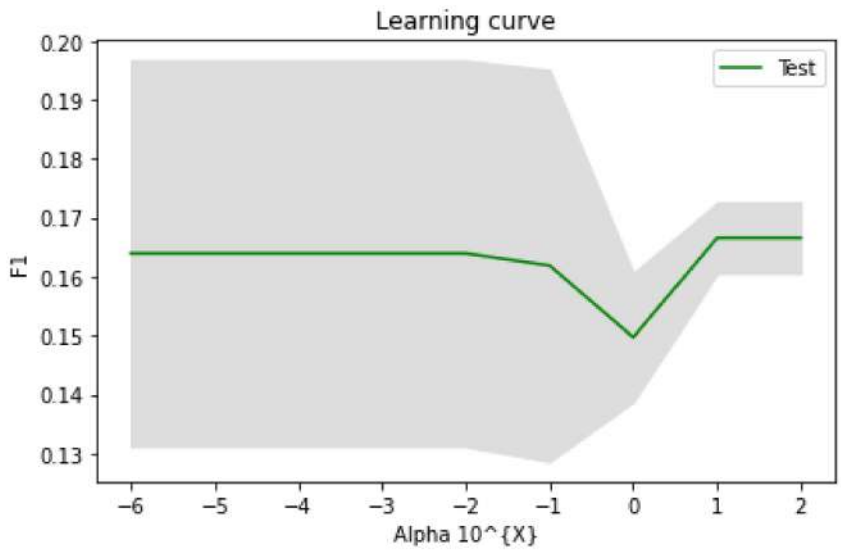
The alpha parameters and the initial learning rate are adjusted to obtain the best performance with the model using cross validation.

In [547]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_normal

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = 0.0001, max_iter =

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[547]:
```
([0.16659498207885304,
  0.16659498207885304,
  0.1497243851192683,
  0.16188993061073148,
  0.16396631177498805,
  0.16396631177498805,
  0.16396631177498805,
  0.16396631177498805,
  0.16396631177498805],
 [0.006020651927912844,
  0.006020651927912844,
  0.01095327408190148,
  0.03319646273110656,
  0.03274304788507844,
  0.03274304788507844,
  0.03274304788507844,
  0.03274304788507844,
  0.03274304788507844])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [548]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Alpha 10^{X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



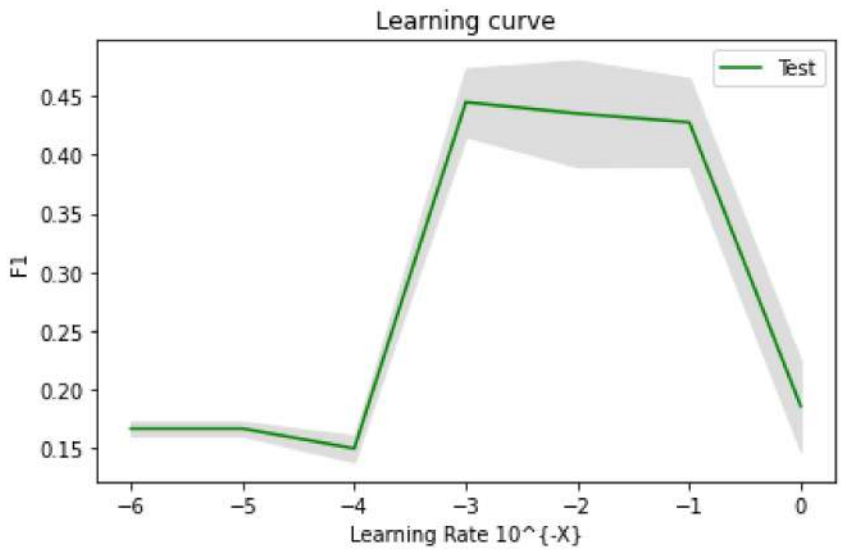
The training is performed again keeping the same parameters except for the alpha value that changes to 1.

In [549]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_alpha

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = regul_param, max_it

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[549]:
```
([0.18580246913580245,
  0.4277402845978697,
  0.43522705708464215,
  0.4446880965611616,
  0.1497243851192683,
  0.16659498207885304,
  0.16659498207885304],
 [0.03837898119056879,
  0.03747002917070688,
  0.04528474596206633,
  0.028780125449296177,
  0.01095327408190148,
  0.006020651927912844,
  0.006020651927912844])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [550]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Learning Rate 10^{-X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



Generation of the neural network model

The parameters of the final model are modified according to the data obtained in the training of the neural network.

In [551]:
```
## The value of the alpha parameter, learning_rate_init and random_state are char
## to the data in the global variables.
mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solver =
                    learning_rate = 'constant', learning_rate_init = lear
                    random_state = random_state_model)

## We print the data of the final generated model.
mlp.fit(X_train, y_train)
```

Out[551]:
```
MLPClassifier
MLPClassifier(activation='logistic', alpha=0.001, hidden_layer_sizes=(10,),
              learning_rate_init=0.1, max_iter=100000, random_state=42)
```

We safeguard the final generated model

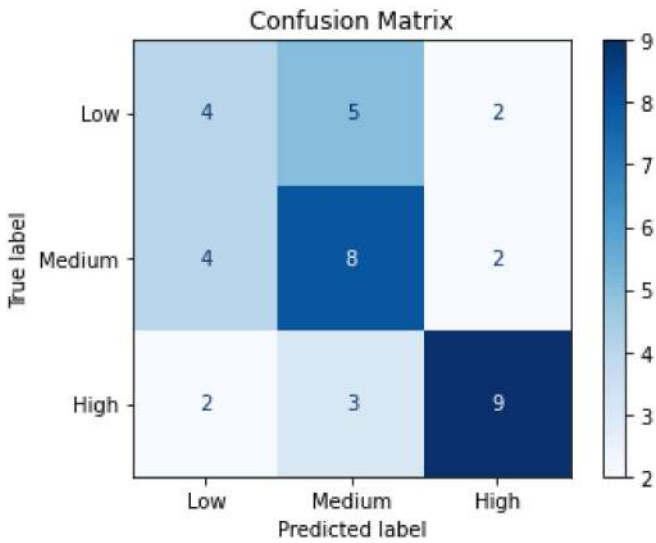
In [552]:
```
joblib.dump(mlp,"model_depression_i" + str(iteration) + ".pkl")
```

Out[552]:
```
['model_depression_i7.pkl']
```

Generation of the confusion matrix of the model

We obtain the statistics of the final model to generate its confusion matrix.

In [553]:
```
## The data for the generation of the confession matrix are defined.
confusion_matrix = ConfusionMatrixDisplay.from_estimator(mlp, X_test, y_test, dis
                                                         cmap = plt.cm.Blues)
confusion_matrix.ax_.set_title("Confusion Matrix")
plt.show()
```



We get the detailed statistics of the final model.

In [554]:
```
## The necessary detailed prediction is generated.
y_pred = mlp.predict(X_test)

## The detailed statistics of the model are printed.
print('Classification accuracy =',accuracy_score(y_test,y_pred) * 100,'%\n')
print(classification_report(y_test,y_pred))
```

```
Classification accuracy = 53.84615384615385 %

              precision    recall  f1-score   support

           0       0.40      0.36      0.38        11
           1       0.50      0.57      0.53        14
           2       0.69      0.64      0.67        14

    accuracy                           0.54        39
   macro avg       0.53      0.53      0.53        39
weighted avg       0.54      0.54      0.54        39
```

The data obtained is saved for later graphing.

In [555]:
```
accuracy = accuracy_score(y_test,y_pred) * 100
clf_rep = precision_recall_fscore_support(y_test,y_pred)

## The number of items used in the iteration is retrieved.
ds_delete = ds[ds.Status == "Delete"]
ds_delete.reset_index(inplace=True, drop=True)

## All the metrics of the confusion matrix are obtained.
metrics = [",".join(map(str, ds_delete['Question'].tolist())),
           accuracy]

for index in range(0, len(name_class)):
    metrics.append(clf_rep[0][index])
    metrics.append(clf_rep[1][index])
    metrics.append(clf_rep[2][index])
    metrics.append(clf_rep[3][index])

## The names of the columns of the dataset are defined.
columns = ['Question', 'Acurracy global']

for index in range(0, len(name_class)):
    columns.append('Precision_' + str(index))
    columns.append('Recall_' + str(index))
    columns.append('F1_score_' + str(index))
    columns.append('Support_' + str(index))

## A new row of the dataset is generated with all the data.
data_metrics = pd.DataFrame([metrics], columns = columns)

## The data is saved to the dataset.
data_metrics.to_csv(path_dataset_qd, mode = 'a', header = False, index = False)

## The data added to the dataset is printed.
data_metrics
```

Out[555]:

	Question	Acurracy global	Precision_0	Recall_0	F1_score_0	Supp
0	1,2,3,4,5,7,9,11,12,14,15,16,18,19,21,23,24,25...	53.846154	0.4	0.363636	0.380952	

Improvement process of the neural network model.

The application of the explanation method by ALE allows to determine the behavior of the neural network and thereby improve it by discarding non-significant data for it.

Slope Analysis of ALE Plots

In [556]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

We proceed to obtain the graphs to obtain the slopes of the selected classes.

In [557]:
```
## The question metrics dataset is loaded.
dm = pd.DataFrame()

## The number of items used in the iteration is retrieved.
ds_item = ds[ds.Status == "In use"]
ds_item.reset_index(inplace=True, drop=True)

## We get the slopes from the ALE data.
for i in range(df.shape[1]-1):
    structure_data = {'Question': ds_item.loc[i, "Question"]}

    for index in selection_class:
        slope = proba_exp_lr.ale_values [i][1][index] - proba_exp_lr.ale_values [
        structure_data['Slope ' + str(target_names[index])] = slope
        structure_data['Threshold ' + str(target_names[index])] = 'NA'
        structure_data['Anomaly ' + str(target_names[index])] = 'NA'

    dm = dm.append(structure_data, ignore_index=True)

## The current rating data dataset is printed.
dm
```

```
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
```

```
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
```

Out[557]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	6	0.032648	NA	NA	-0.044664	NA	NA
1	8	0.126573	NA	NA	-0.037672	NA	NA
2	10	-0.010733	NA	NA	0.049588	NA	NA
3	13	-0.017315	NA	NA	0.018499	NA	NA
4	17	0.016859	NA	NA	0.032759	NA	NA
5	20	-0.010311	NA	NA	0.062439	NA	NA
6	22	-0.114843	NA	NA	-0.047169	NA	NA
7	28	-0.046881	NA	NA	-0.101661	NA	NA
8	31	0.008161	NA	NA	-0.052244	NA	NA

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
9	34	-0.272183	NA	NA	0.280418	NA	NA
10	37	-0.238222	NA	NA	-0.257174	NA	NA
11	39	-0.019448	NA	NA	-0.000321	NA	NA
12	40	0.029474	NA	NA	-0.048670	NA	NA
13	43	-0.019564	NA	NA	-0.005797	NA	NA
14	47	0.158905	NA	NA	-0.283337	NA	NA
15	52	-0.200890	NA	NA	-0.017093	NA	NA
16	53	0.022762	NA	NA	0.096146	NA	NA
17	56	0.095921	NA	NA	-0.107135	NA	NA
18	57	-0.004493	NA	NA	0.006707	NA	NA
19	61	0.152317	NA	NA	0.059031	NA	NA
20	64	0.063777	NA	NA	0.015105	NA	NA
21	65	0.028073	NA	NA	0.031777	NA	NA
22	67	-0.002807	NA	NA	-0.002560	NA	NA
23	69	-0.001950	NA	NA	0.005187	NA	NA
24	71	-0.026291	NA	NA	0.018557	NA	NA
25	73	-0.144607	NA	NA	-0.036800	NA	NA
26	74	0.035285	NA	NA	0.042495	NA	NA
27	76	-0.056575	NA	NA	-0.000895	NA	NA
28	79	0.012975	NA	NA	-0.008219	NA	NA
29	80	0.126595	NA	NA	-0.076295	NA	NA
30	81	-0.196908	NA	NA	0.055264	NA	NA
31	82	-0.095237	NA	NA	0.091889	NA	NA
32	84	0.161125	NA	NA	-0.085582	NA	NA
33	92	-0.045044	NA	NA	-0.029263	NA	NA
34	97	-0.124066	NA	NA	0.209223	NA	NA
35	99	0.226210	NA	NA	-0.214944	NA	NA

Slope analysis and threshold application

The slope obtained is analyzed and those that are above or below the positive and negative threshold are selected.

In [558]:
```
for i in range(df.shape[1]-1):
    for index in selection_class:

        ## The slopes are selected according to the values of the thresholds of t
        if dm.loc[i, "Slope " + str(target_names[index])] >= positive_threshold:
            dm.loc[i, "Threshold " + str(target_names[index])] = 1
        elif dm.loc[i, "Slope " + str(target_names[index])] <= negative_threshold
            dm.loc[i, "Threshold " + str(target_names[index])] = 0

## The current rating data dataset is printed.
dm
```

Out[558]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	6	0.032648	1	NA	-0.044664	0	NA
1	8	0.126573	1	NA	-0.037672	0	NA
2	10	-0.010733	0	NA	0.049588	1	NA
3	13	-0.017315	0	NA	0.018499	1	NA
4	17	0.016859	1	NA	0.032759	1	NA
5	20	-0.010311	0	NA	0.062439	1	NA
6	22	-0.114843	0	NA	-0.047169	0	NA
7	28	-0.046881	0	NA	-0.101661	0	NA
8	31	0.008161	NA	NA	-0.052244	0	NA
9	34	-0.272183	0	NA	0.280418	1	NA
10	37	-0.238222	0	NA	-0.257174	0	NA
11	39	-0.019448	0	NA	-0.000321	NA	NA
12	40	0.029474	1	NA	-0.048670	0	NA
13	43	-0.019564	0	NA	-0.005797	NA	NA
14	47	0.158905	1	NA	-0.283337	0	NA
15	52	-0.200890	0	NA	-0.017093	0	NA
16	53	0.022762	1	NA	0.096146	1	NA
17	56	0.095921	1	NA	-0.107135	0	NA
18	57	-0.004493	NA	NA	0.006707	NA	NA
19	61	0.152317	1	NA	0.059031	1	NA
20	64	0.063777	1	NA	0.015105	1	NA
21	65	0.028073	1	NA	0.031777	1	NA
22	67	-0.002807	NA	NA	-0.002560	NA	NA
23	69	-0.001950	NA	NA	0.005187	NA	NA
24	71	-0.026291	0	NA	0.018557	1	NA
25	73	-0.144607	0	NA	-0.036800	0	NA
26	74	0.035285	1	NA	0.042495	1	NA

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
27	76	-0.056575	0	NA	-0.000895	NA	NA
28	79	0.012975	1	NA	-0.008219	NA	NA
29	80	0.126595	1	NA	-0.076295	0	NA
30	81	-0.196908	0	NA	0.055264	1	NA
31	82	-0.095237	0	NA	0.091889	1	NA
32	84	0.161125	1	NA	-0.085582	0	NA
33	92	-0.045044	0	NA	-0.029263	0	NA
34	97	-0.124066	0	NA	0.209223	1	NA
35	99	0.226210	1	NA	-0.214944	0	NA

Analysis and determination of anomalous items

The anomalous items present in the model are determined based on their slope and expected response from the expert.

In [559]:

```
## The dataset of expected responses is loaded.
da = pd.read_csv(path_dataset_qa)

## Delete list of items deleted in this iteration.
i = 0
for x in ds_delete['Question']:
    da.drop(da.index[x - (i + 1)], axis = 0, inplace = True)
    i += 1

## Indexes are restored for later use.
da.reset_index(inplace=True, drop=True)

## The dataset of expected responses is printed.
da
```

Out[559]:

	Number	Question	RE
0	6	I consider that my life is full of good things	0
1	8	I'm still as punctual as I have always been	0
2	10	I have found that I can spend a lot of time sc...	1
3	13	I felt more encouraged to do my daily activiti...	1
4	17	I feel guilty about the decisions that I have ...	1
5	20	It is difficult for me to make decisions even ...	1
6	22	It is not difficult for me to understand somet...	0
7	28	The decisions I have made so far have been the...	0
8	31	It is easy for me to choose a photograph of my...	0
9	34	In recent months I have had discussions with m...	1
10	37	I'm not too worried about what might happen in...	0
11	39	Everything will be alright	0
12	40	I can easily blank my mind	0
13	43	Sometimes i feel trapped	1
14	47	I can clearly express to others how I feel	0
15	52	Being locked in an elevator would be the worst...	1
16	53	I'm bothered by people walking slowly in front...	1
17	56	I manage my schedule as I always have	0
18	57	It bothers me to feel that people on the stree...	1
19	61	Sometimes I forget what I wanted to say becaus...	1
20	64	I can put down my cell phone and dedicate myse...	0
21	65	I worry that people will not understand what I...	1
22	67	I get angry easily	1
23	69	It is not important for me to meet set dates	1
24	71	I use lies just to get out of certain problems	1
25	73	I have difficulty making elaborate plans	1

	Number	Question	RE
26	74	People have problems because of themselves	1
27	76	Laws are not as important as others think	1
28	79	It is easy for me to work in a team	0
29	80	It is important to help people when they need it	0
30	81	I have punched someone or thought of doing it	1
31	82	If it was necessary I would pretend to be some...	1
32	84	After an argument I usually go over what happe...	0
33	92	It does not seem serious to me to have some debts	1
34	97	People who commit crimes have their reasons fo...	1
35	99	It is important to respect turns	0

We proceed to identify the anomadic items present in the model present in this iteration.

In [560]:

```
for i in range(df.shape[1]-1):

    ## Local variable to indicate the index of the selected classes.
    i_index = 0

    for index in selection_class:

        ## Determine if it fails based on the expected response parameters of the
        if dm.loc[i, "Threshold "  + str(target_names[index])] == 1 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 1 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 0 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 0 and da.loc[i,
            dm.loc[i, "Anomaly "  + str(target_names[index])] = expected_response

        ## Increase of the index of the selected classes.
        i_index += 1

## The final grade data dataset is printed.
dm
```

Out[560]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	6	0.032648	1	0	-0.044664	0	0
1	8	0.126573	1	0	-0.037672	0	0
2	10	-0.010733	0	0	0.049588	1	0
3	13	-0.017315	0	0	0.018499	1	0
4	17	0.016859	1	1	0.032759	1	0
5	20	-0.010311	0	0	0.062439	1	0
6	22	-0.114843	0	1	-0.047169	0	0
7	28	-0.046881	0	1	-0.101661	0	0
8	31	0.008161	NA	NA	-0.052244	0	0
9	34	-0.272183	0	0	0.280418	1	0
10	37	-0.238222	0	1	-0.257174	0	0
11	39	-0.019448	0	1	-0.000321	NA	NA
12	40	0.029474	1	0	-0.048670	0	0
13	43	-0.019564	0	0	-0.005797	NA	NA
14	47	0.158905	1	0	-0.283337	0	0
15	52	-0.200890	0	0	-0.017093	0	1
16	53	0.022762	1	1	0.096146	1	0
17	56	0.095921	1	0	-0.107135	0	0
18	57	-0.004493	NA	NA	0.006707	NA	NA
19	61	0.152317	1	1	0.059031	1	0

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
20	64	0.063777	1	0	0.015105	1	1
21	65	0.028073	1	1	0.031777	1	0
22	67	-0.002807	NA	NA	-0.002560	NA	NA
23	69	-0.001950	NA	NA	0.005187	NA	NA
24	71	-0.026291	0	0	0.018557	1	0
25	73	-0.144607	0	0	-0.036800	0	1
26	74	0.035285	1	1	0.042495	1	0
27	76	-0.056575	0	0	-0.000895	NA	NA
28	79	0.012975	1	0	-0.008219	NA	NA
29	80	0.126595	1	0	-0.076295	0	0
30	81	-0.196908	0	0	0.055264	1	0
31	82	-0.095237	0	0	0.091889	1	0
32	84	0.161125	1	0	-0.085582	0	0
33	92	-0.045044	0	0	-0.029263	0	1
34	97	-0.124066	0	0	0.209223	1	0
35	99	0.226210	1	0	-0.214944	0	0

Removing identified anomadic items from the model

We proceed to eliminate the identified anomadic items so as not to use them in the next iteration.

In [561]:
```
## Local variable for the elimination of the anomaly items.
instructions = ''
init = False
question_anomaly = []

if selection_mode == 1:

    ## Items that fail in the defined class are retrieved.
    dm_delete = dm[dm['Anomaly ' + str(target_names[select_class_mode])] == 1]
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 2:

    ## Items that fail in classes defined with the or condition type are retrieve
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 1"
            init = True
        else:
            instructions += " or `Anomaly " + str(target_names[index]) + "` == 1'

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 3:

    ## Items that fail in classes defined with the and condition type are retriev
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 1"
            init = True
        else:
            instructions += " and `Anomaly " + str(target_names[index]) + "` == 1

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 4:

    ## Out-of-threshold items are retrieved in classes defined with the conditior
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 'NA'"
            init = True
        else:
            instructions += " and `Anomaly " + str(target_names[index]) + "` == '

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

## The selected blank items are printed according to the selection_mode.
question_anomaly
```

```
Out[561]: [52, 64, 73, 92]
```

Model explainability process

The data used for the test of the neural network model is shown.

In [562]: X_test

Out[562]:

	6. I consider that my life is full of good things	8. I'm still as punctual as I have always been	10. I have found that I can spend a lot of time scrolling the screen \nof my cell phone without searching or stopping at anything in particular	13. I felt more encouraged to do my daily activities before	17. I feel guilty about the decisions that I have made	20. It is difficult for me to make decisions even if they are simple	22. It is not difficult for me to understand something the first time	28. The decisions I have made so far have been the right ones	e t p
69	1	0	0	0	0	1	1	0	
46	1	1	1	1	1	0	1	1	
58	1	1	0	1	0	0	1	1	
114	1	1	1	0	1	0	1	0	
73	1	1	1	0	0	0	1	1	
98	0	1	1	0	0	1	1	0	
31	1	0	0	0	1	0	1	1	
53	1	1	1	1	1	0	1	0	
65	1	1	0	1	0	0	1	1	
96	1	1	1	0	1	0	1	0	
95	1	0	1	0	0	1	0	0	
97	1	1	0	0	1	1	1	1	
2	1	0	1	0	0	0	1	1	
62	1	0	1	0	0	1	0	1	
110	0	1	0	0	0	0	0	0	
55	1	1	0	0	1	1	0	0	
103	1	0	1	0	0	1	0	1	
100	1	0	0	1	0	1	0	1	
66	0	0	1	0	1	0	1	0	
44	1	1	0	0	0	0	0	1	
77	1	1	1	0	1	1	1	1	
17	1	1	1	1	0	1	1	1	
81	1	1	0	1	1	0	0	0	
74	0	1	1	1	1	1	0	0	

56	1	1	0	1	1	0	1	0
94	0	0	1	1	1	0	1	0
35	1	1	0	0	0	1	1	1
38	1	1	0	0	0	1	0	1
93	1	1	1	1	0	1	0	1
48	1	0	1	1	1	1	1	1
33	1	0	1	1	1	0	1	0
59	1	1	1	0	1	0	0	0
92	0	1	0	0	1	1	1	0
42	1	1	0	0	0	0	0	1
10	1	0	1	0	0	0	0	0
85	1	1	1	1	1	0	1	0
32	1	0	1	1	1	1	0	0
78	1	1	0	0	0	1	1	1
19	0	1	0	1	0	0	0	1

39 rows × 36 columns

Explanation of the model using ALE

In analysis of the model through the results obtained by ALE, it is possible to identify the behavior of the neural network.

In [563]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

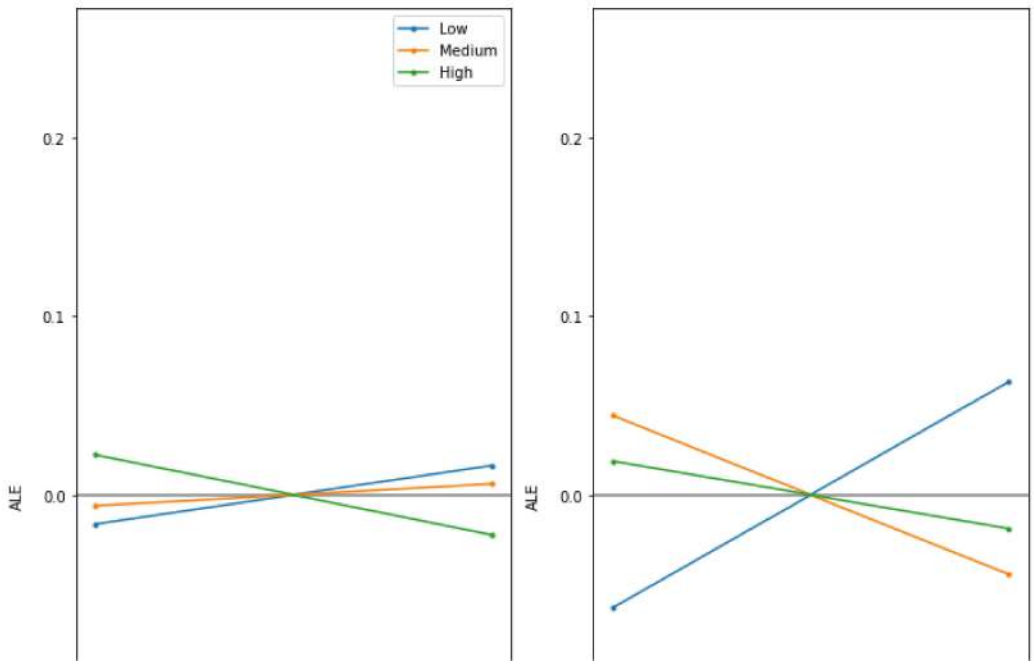
```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

The graphs of all the data present in the dataset used are shown.

In [564]:
```
plot_ale(proba_exp_lr, n_cols=2, features=list(range(df.shape[1]-1)),fig_kw={'fig
```



Explanation of the model using LIME

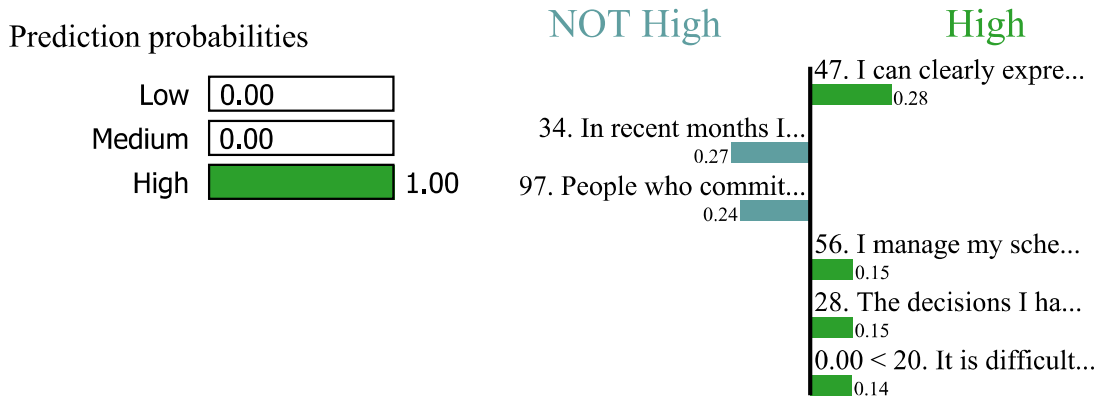
We proceed to explain the improved model using LIME for the Low, Medium and High classes.

In [565]:
```
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorica
                                                   feature_names=train_cols, clas
                                                   discretize_continuous=True)
```

The first individual present in the test data is explained.

In [566]:
```
exp = explainer.explain_instance(X_test.to_numpy()[0], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

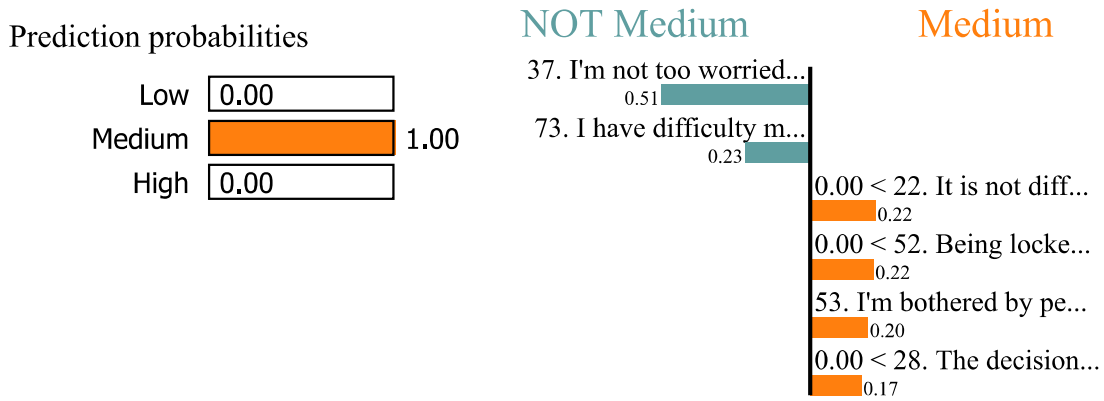


Feature	Value
47. I can clearly express to others how I feel	0.00
34. In recent months I have had discussions with my schoolmates or colleagues	0.00
97. People who commit crimes have their reasons for doing it	0.00
56. I manage my schedule as I always have	0.00
28. The decisions I have made so far have been the right ones	0.00
20. It is difficult for me to make decisions even if they are simple	1.00

The second individual present in the test data is explained.

In [567]:
```
exp = explainer.explain_instance(X_test.to_numpy()[1], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

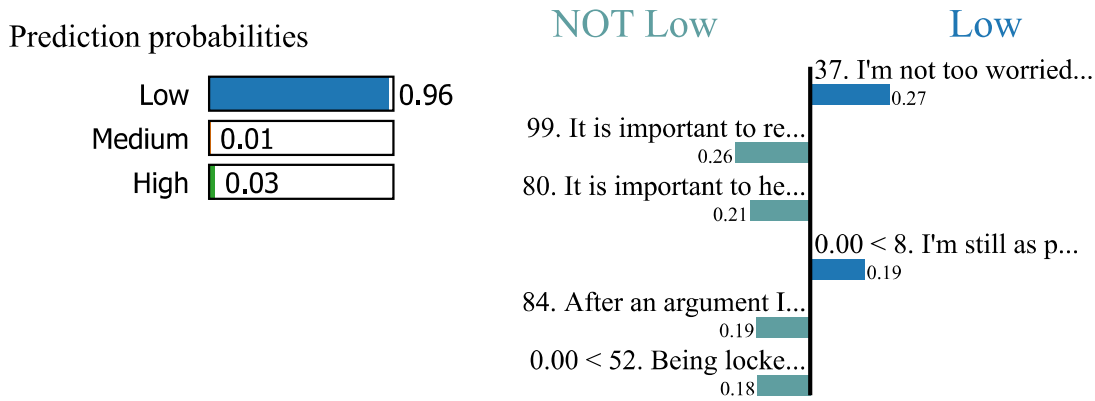


Feature	Value
37. I'm not too worried about what might happen in a few weeks	0.00
73. I have difficulty making elaborate plans	0.00
22. It is not difficult for me to understand something the first time	1.00
52. Being locked in an elevator would be the worst thing that could happen to me	1.00
53. I'm bothered by people walking slowly in front of me	0.00
28. The decisions I have made so far have been the right ones	1.00

The last individual present in the test data is explained.

In [568]:
```
exp = explainer.explain_instance(X_test.to_numpy()[-1], mlp.predict_proba, num_fe
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```



Feature	Value
37. I'm not too worried about what might happen in a few weeks	0.00
99. It is important to respect turns	0.00
80. It is important to help people when they need it	0.00
8. I'm still as punctual as I have always been	1.00
84. After an argument I usually go over what happened in my head	0.00
52. Being locked in an elevator would be the worst thing that could happen to me	1.00

Explanation of the model using SHAP

SHAP explanatory values are displayed according to the class to which the selected individual belongs according to the prediction of the model.

The first individual present in the test data is explained.

In [569]:
```
## Individual Selection.
individual = X_test.iloc[[0]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The second individual present in the data test is explained.

In [570]:

```
## Individual Selection.
individual = X_test.iloc[[1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plo
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The last individual present in the data test is explained.

In [571]:
```
## Individual Selection.
individual = X_test.iloc[[-1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plo
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

Explanation using heatmaps

The importance of the data present in the neural network is shown from explanation methods.

In [572]:

```
## Items currently in use are retrieved.
ds_item = ds[ds.Status == "In use"].reset_index(drop=True)

## The data is prepared for the generation of heatmaps.
column_Map = ds_item['Question'].tolist()
content_Map = [False for x in range(1, df.shape[1])]

## The next items to be deleted will be marked.
for index in range(0, len(content_Map)):

    if iteration == 0:
        for delete in list_contr:
            if column_Map[index] == delete:
                content_Map [index] = True
    else:
        for delete in question_anomaly:
            if column_Map[index] == delete:
                content_Map [index] = True

## The dataframe is created with the prepared data.
df_mask = pd.DataFrame([content_Map], columns = column_Map)
df_mask_imp = pd.concat([df_mask, df_mask, df_mask], ignore_index = True, axis =
df_mask_all = pd.concat([df_mask, df_mask, df_mask, df_mask], ignore_index = True

df_mask
```

Out[572]:

	6	8	10	13	17	20	22	28	31	34	...	74	76	79	8(
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False

1 rows × 36 columns

Identification of feature variables

Identification of feature variables from the randomness of the answers as items of the data.

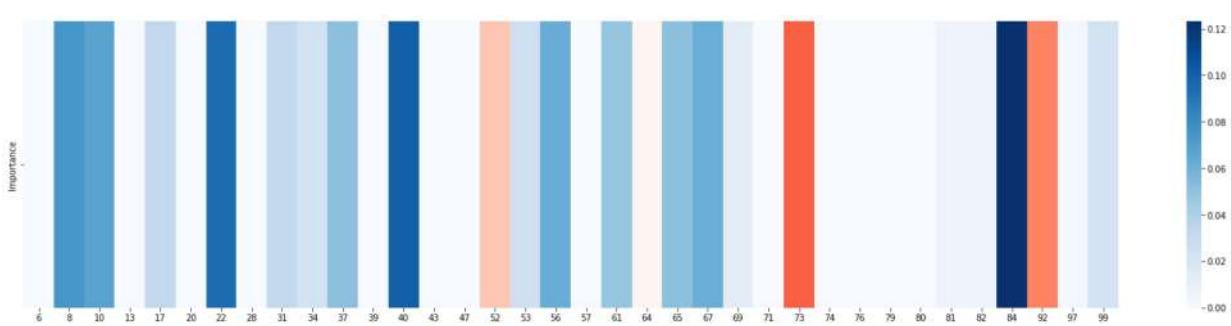
In [573]:
```
## Loading the data for explainability.
explainer = ClassifierExplainer(mlp, X_test, y_test)

## Class index local variable.
iterable_class = -1
map_class = []
```

```
WARNING: Parameter shap='guess', but failed to guess the type of shap explainer
to use for MLPClassifier. Defaulting to the model agnostic shap.KernelExplainer
(shap='kernel'). However this will be slow, so if your model is compatible with
e.g. shap.TreeExplainer or shap.LinearExplainer then pass shap='tree' or shap
='linear'!
WARNING: For shap='kernel', shap interaction values can unfortunately not be ca
lculated!
Note: shap values for shap='kernel' normally get calculated against X_backgroun
d, but paramater X_background=None, so setting X_background=shap.sample(X, 5
0)...
Generating self.shap_explainer = shap.KernelExplainer(model, X, link='identit
y')
```

Explainability of the data belonging to the first class, in this case the Low.

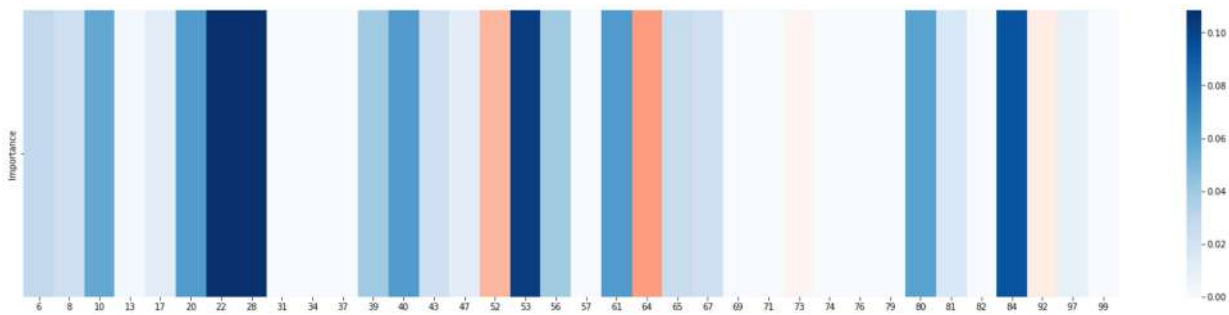
In [574]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

```
Calculating permutation importances (if slow, try setting n_jobs parameter)...
```



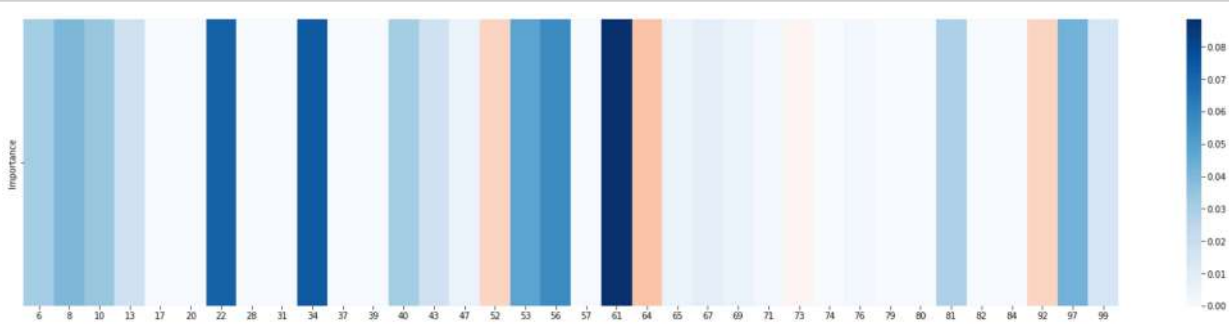
Explainability of the data belonging to the second class, in this case the Medium.

In [575]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



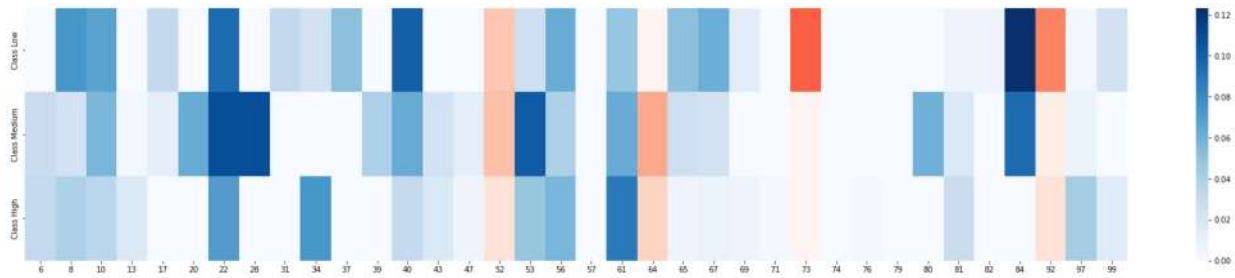
Explainability of the data belonging to the third class, in this case the High.

In [576]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [577]:
```
## Fusion of the previous heatmaps.
df_importance = pd.concat(map_class, axis=1)
df_importance.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_importance.transpose(), cmap = "Reds", cbar = False)
sns.heatmap(df_importance.transpose(), cmap = "Blues",yticklabels = True, xtickla
plt.show()
```



Identification of feature variables using SHAP

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of SHAP.

In [578]:
```
## Class index local variable.
iterable_class = -1
map_class = []
```

Individual explainability of the data belonging to the first class, in this case the Low.

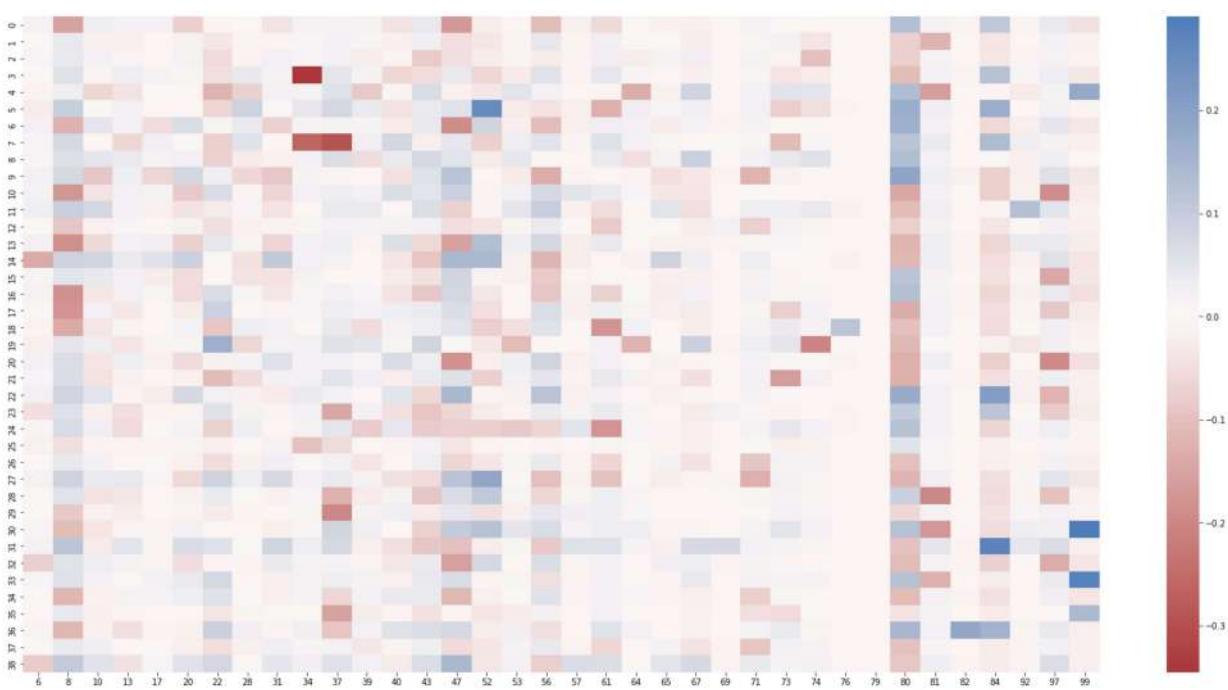
Definition of explainability data.

In [579]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

```
Calculating shap values...

  0%|          | 0/39 [00:00<?, ?it/s]
```

Generation of the heatmap.

In [580]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

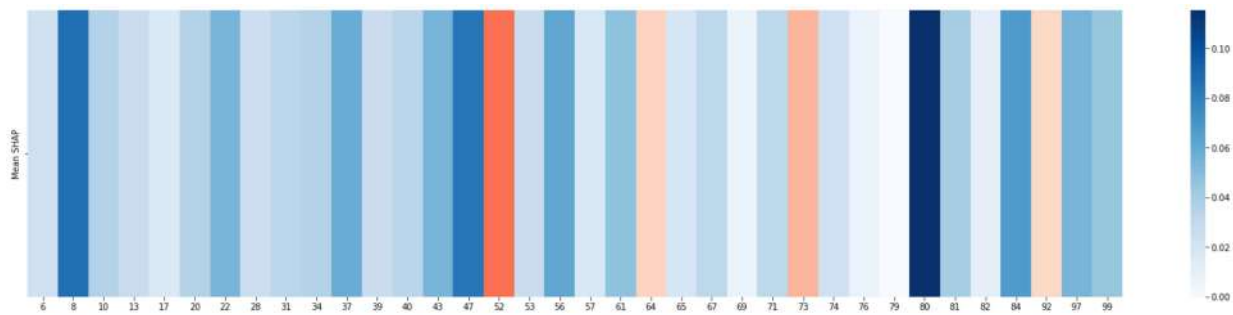


Fusion of the data used.

In [581]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [582]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



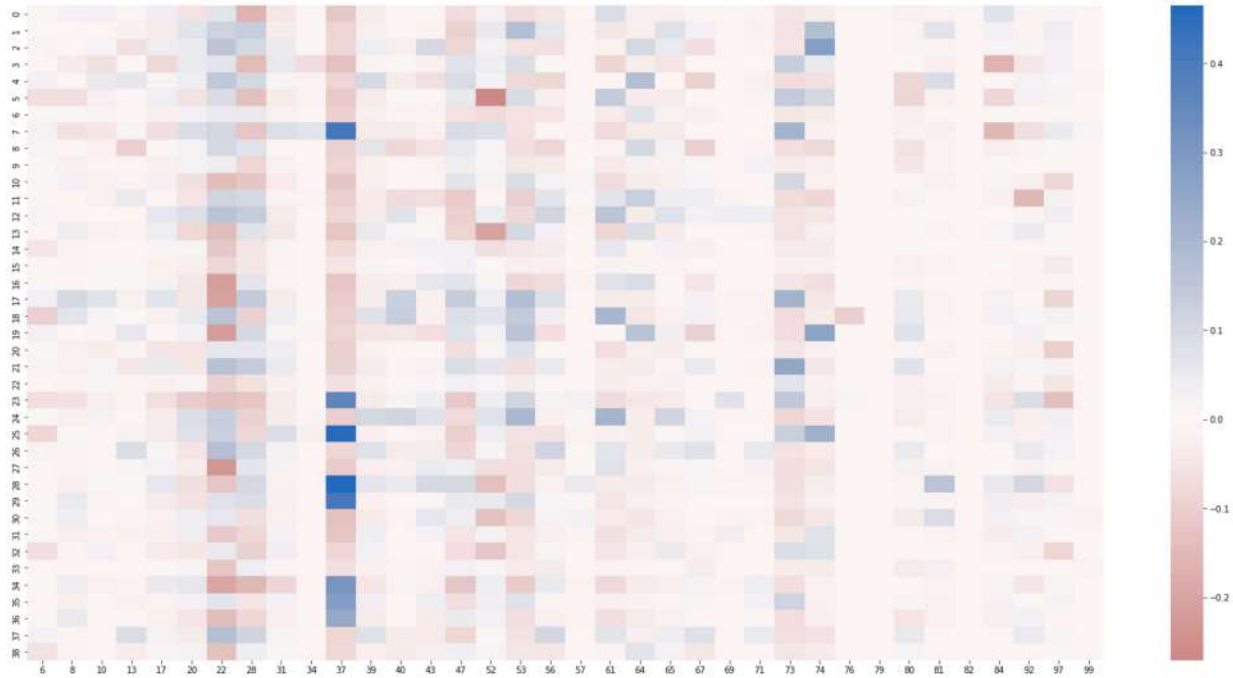
Individual explainability of the data belonging to the second class, in this case the Medium.

Definition of explainability data.

In [583]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [584]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

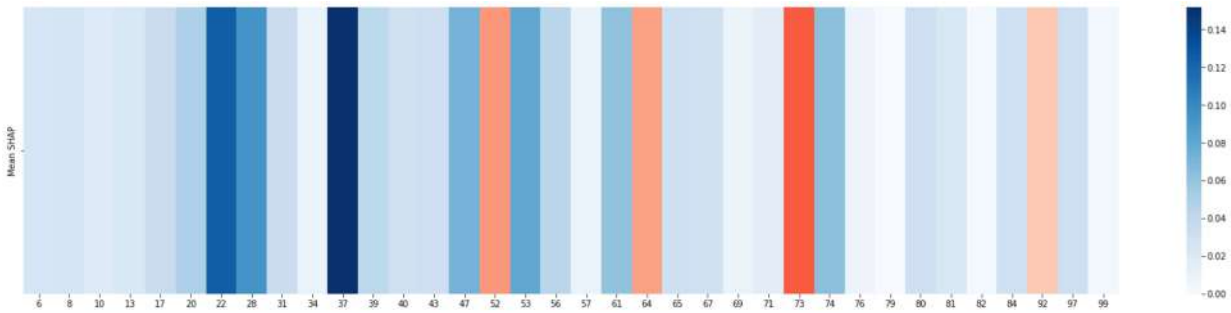


Fusion of the data used.

In [585]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [586]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



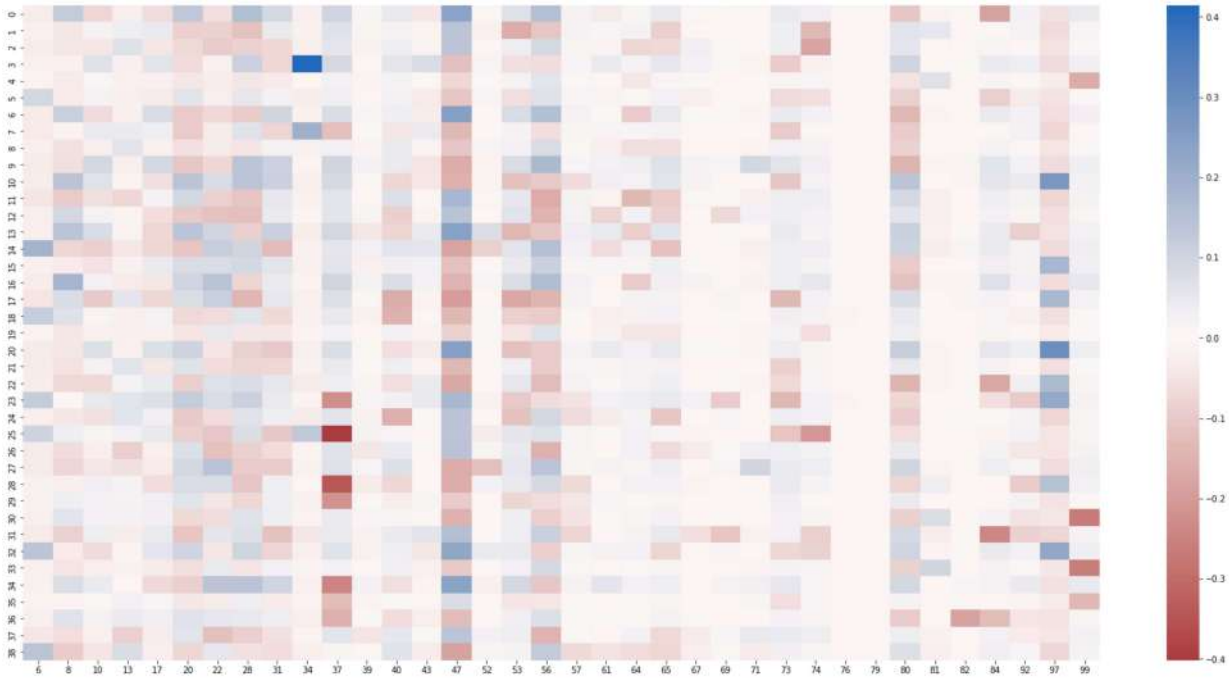
Individual explainability of data belonging to the third class, in this case High.

Definition of explainability data.

In [587]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [588]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

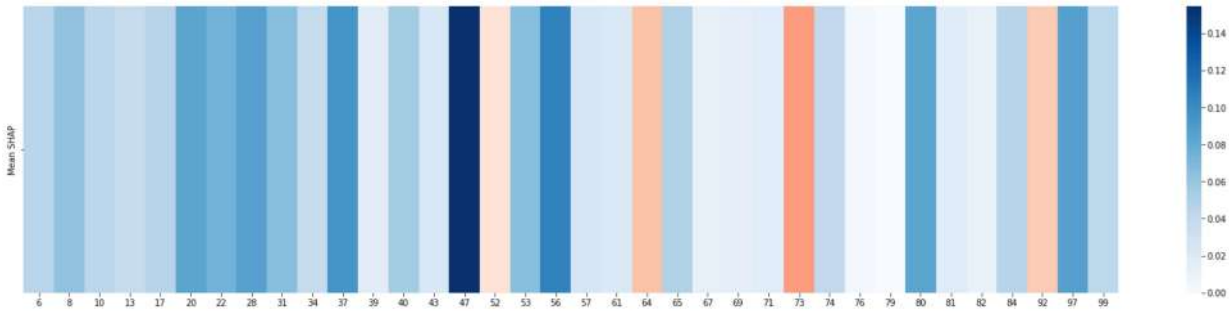


Fusion of the data used.

In [589]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

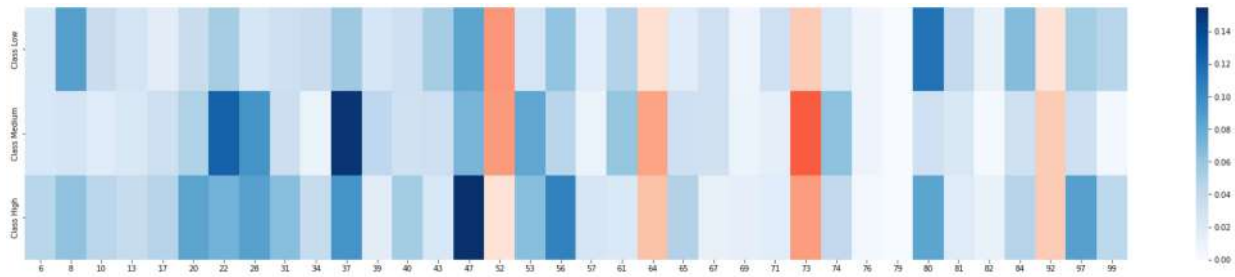
Generation of the heatmap.

In [590]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [591]:
```
## Fusion of the previous heatmaps.
df_shap = pd.concat(map_class, axis=1)
df_shap.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_shap.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_shap.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using LIME

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of LIME.

In [592]:
```
## Data preparation for explainability.
lime_exp = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorical
                                                  feature_names=train_cols, class
                                                  discretize_continuous=True)

## Class index local variable.
iterable_class = -1
map_class = []
```

Obtaining the data for the generation of the explanations by LIME about the Low, Medium and High classes.

In [593]:
```
## Local variables for LIME array data.
exp_matrix = [[] for _ in range(len(name_class))]

## Recovering the data needed for LIME.
for x in X_test.to_numpy():

    ## Auxiliary temporary variables.
    exp_list = [[] for _ in range(len(name_class))]

    ## Loading the data for explainability.
    exp = lime_exp.explain_instance(x, mlp.predict_proba, num_features = df.shape

    ## Recovering the data.
    for elements in range(0, len(name_class)):
        temp = exp.as_map()[elements]
        temp.sort(key=itemgetter(0))

        ## Saving data of the Low, Medium and High classes.
        for tup in temp:
            exp_list[elements].append(tup[1])

        exp_matrix[elements].append(exp_list[elements])
```

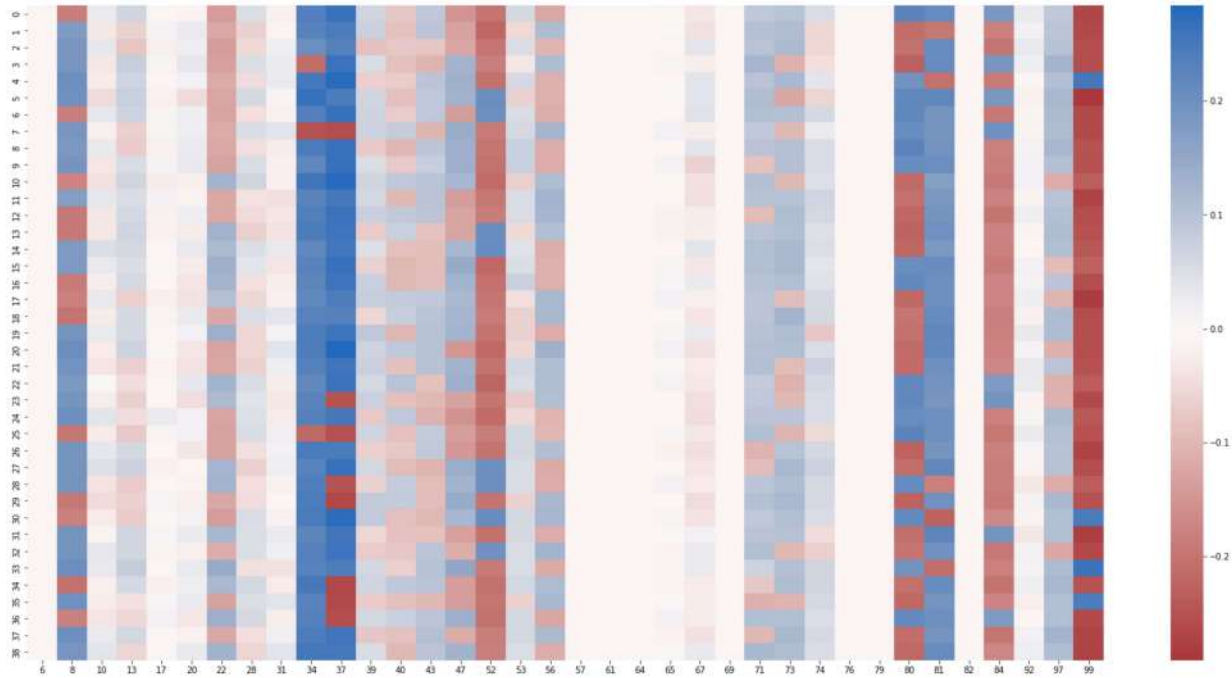
```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

Individual explainability of the data belonging to the first class, in this case the Low.

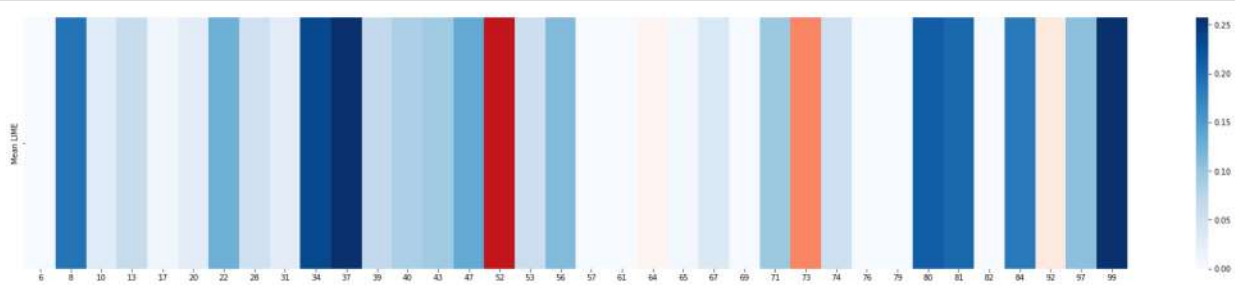
Generation of the heatmap.

In [594]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

In [595]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

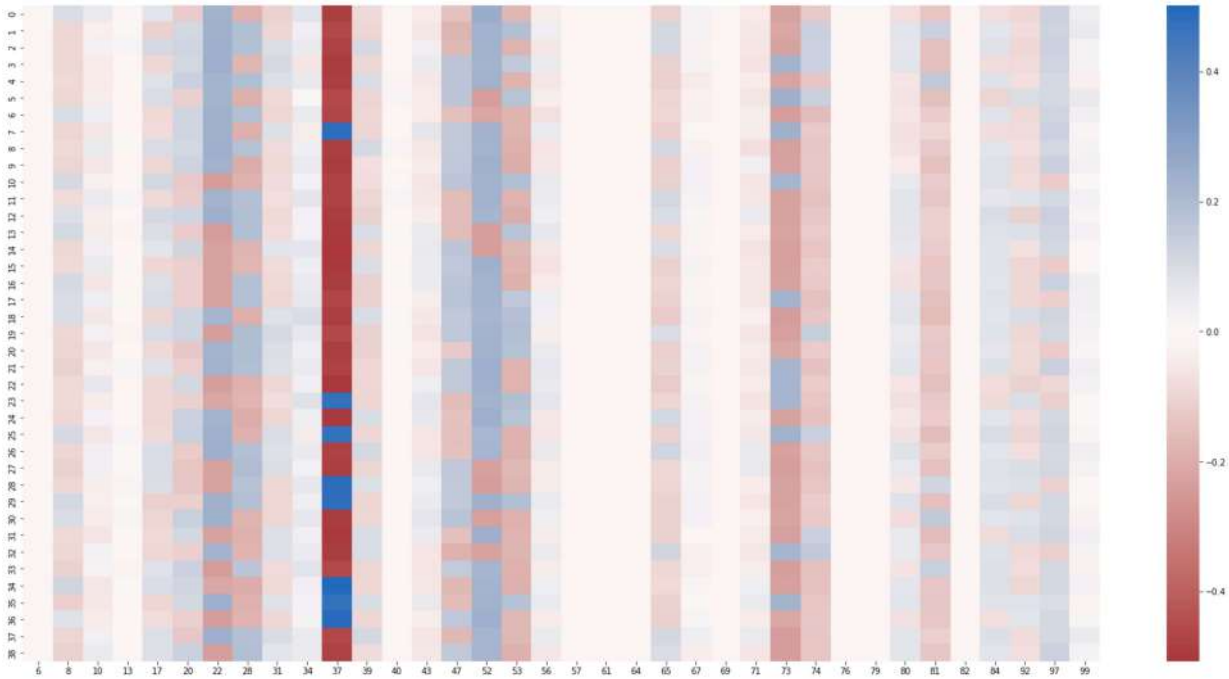


Individual explainability of the data belonging to the second class, in

this case the Medium.

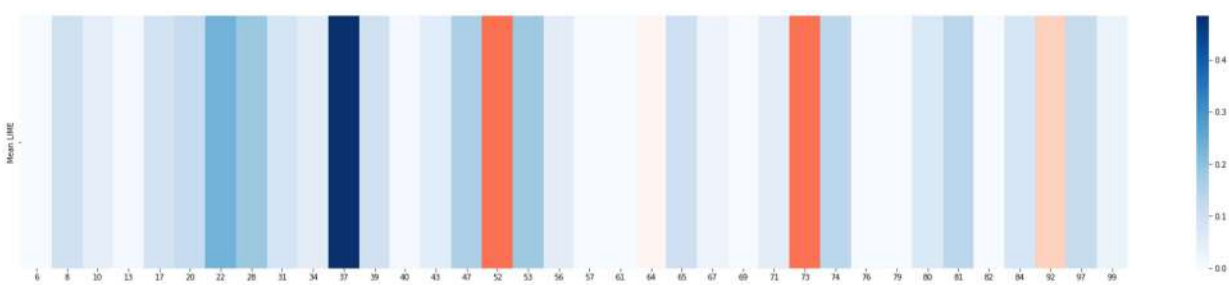
Generation of the heatmap.

In [596]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

In [597]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

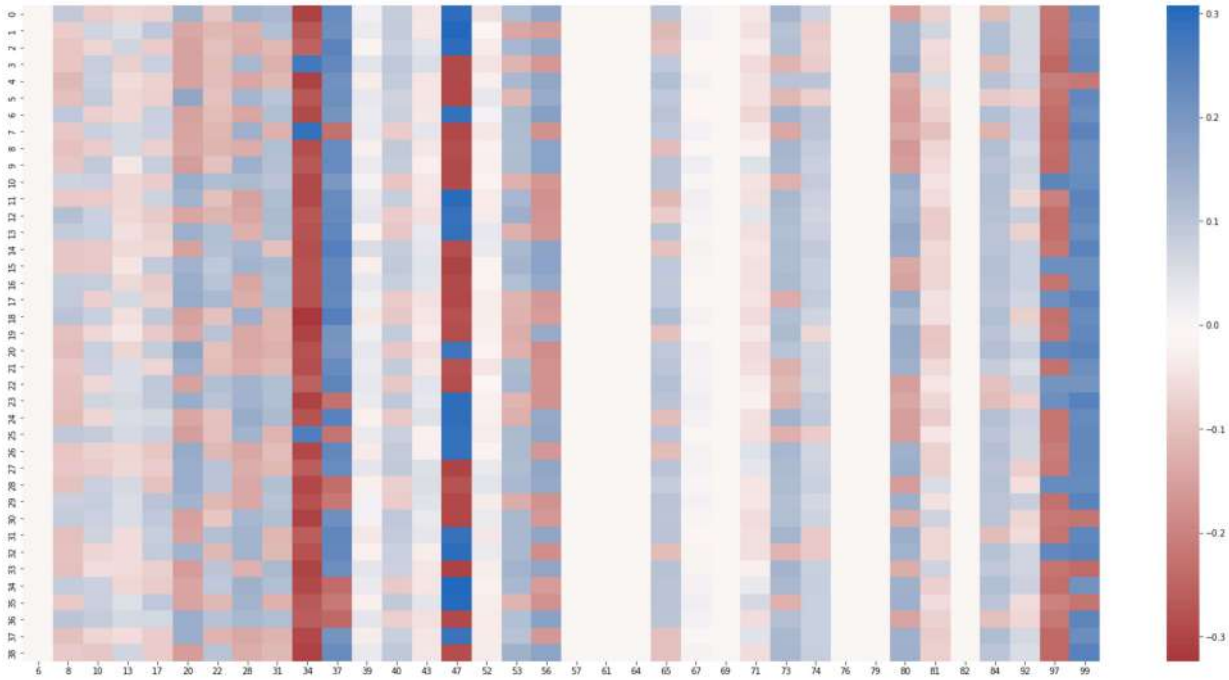


Individual explainability of data belonging to the third class, in this

case High.

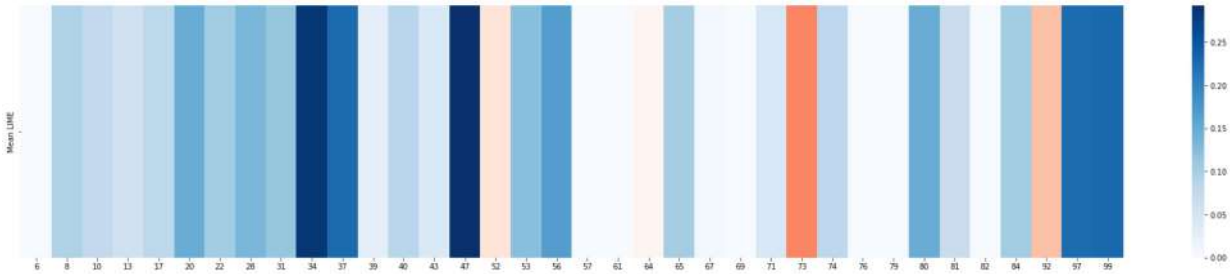
Generation of the heatmap.

In [598]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



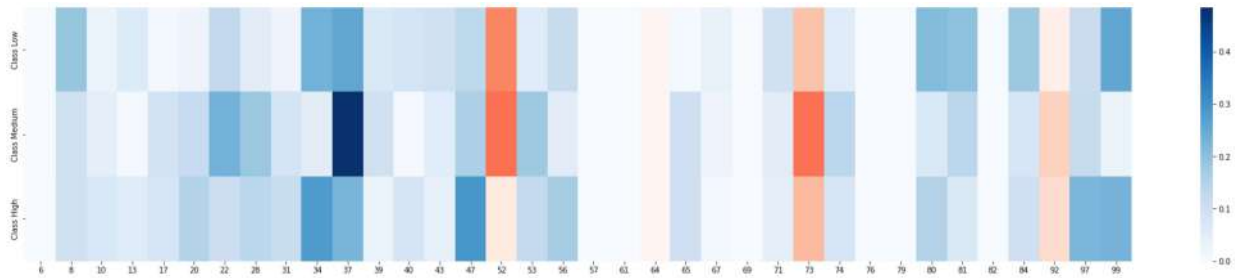
Fusion of the data used.

In [599]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [600]:
```
df_lime = pd.concat(map_class, axis=1)
df_lime.columns = name_class
plt.figure(figsize = (32,6))
sns.heatmap(df_lime.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_lime.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using ALE

It shows the main effect of each question compared to the probability that the model predicts in each class. This effect is shown for both responses. Since there are only 2 answers, the map is the same but with the colors reversed.

In [601]:
```
## Definition of explainability data.
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names=train_cols, target_names= target_r
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())

## Class index local variable.
iterable_class = -1
map_class = []
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

Obtaining the data for the generation of the explanations by ALE about all the classes.

In [602]:

```
## Local variables for ALE array data.
ale_list = [[] for _ in range(len(name_class))]

## Recovering the data needed for ALE.
for array in proba_exp_lr.ale_values:

    ## Recovering the data.
    for elements in range(0, len(name_class)):

        ## Saving data of all the classes.
        ale_list[elements].append(array[0][elements])
```

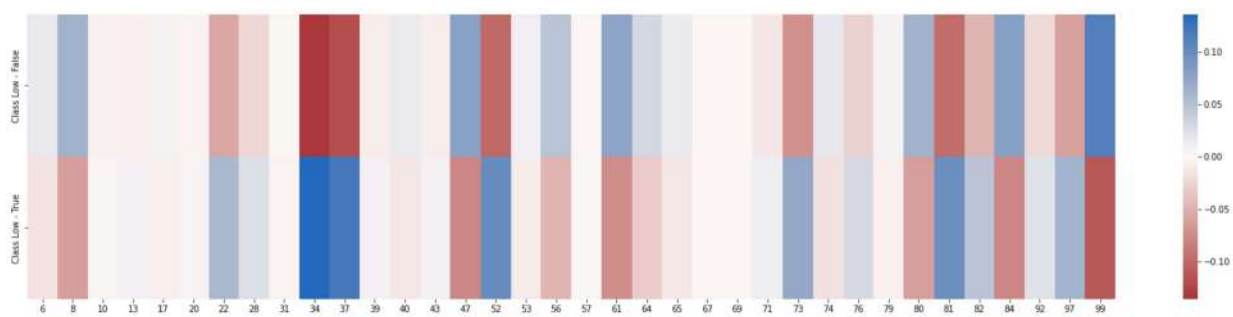
Data processing for the explainability of ALE.

In [603]:
```
## Data processing of the Low, Medium and High classes.
for elements in range(0, len(name_class)):
    ale_df = pd.DataFrame([ale_list [elements]])
    ale_df = pd.concat([ale_df.multiply(-1), ale_df])
    ale_df.index = [name_class [elements] + " - False", name_class [elements] + '
    ale_df.columns = column_Map
    map_class.append(ale_df)
```

Explainability of the data belonging to the first class, in this case the Low.

Generation of the heatmap.

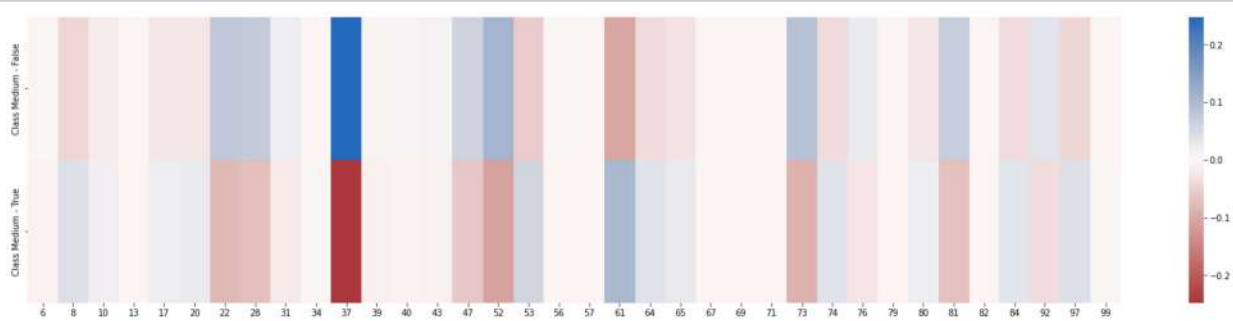
In [604]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the second class, in this case the Medium.

Generation of the heatmap.

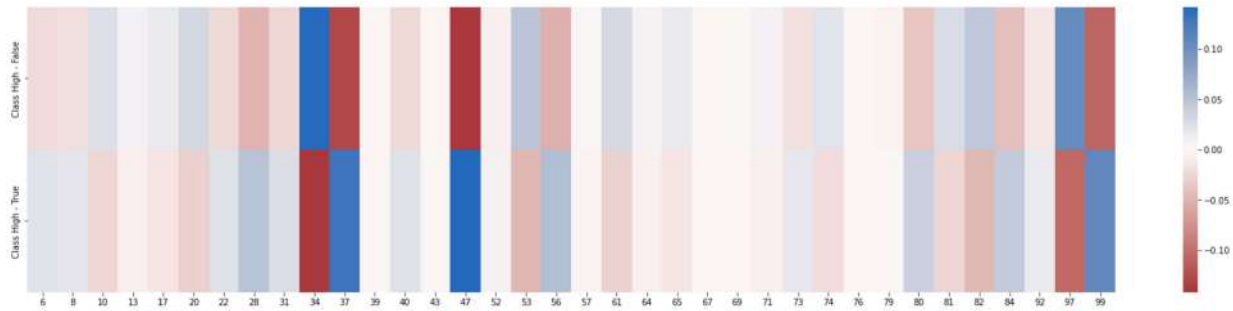
In [605]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the third class, in this case the High.

Generation of the heatmap.

In [606]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```

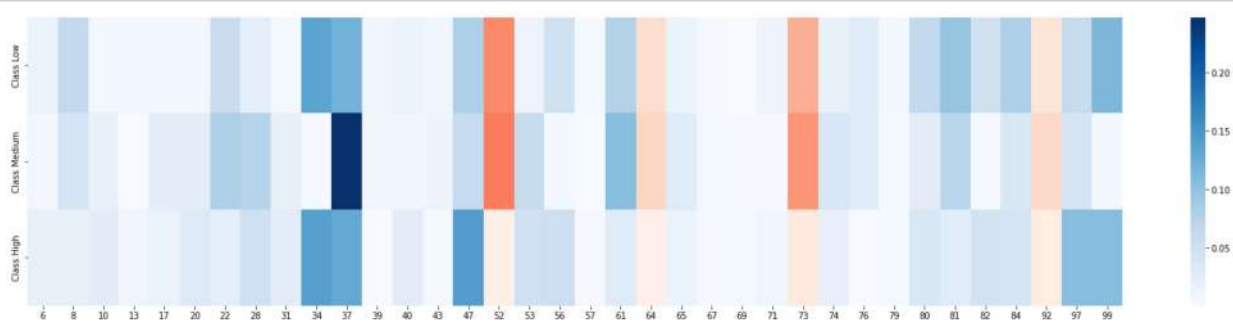


Fusion of the data used in all classes.

In [607]:
```
## Application of absolute value to data.
for index in range(0, len(map_class)):
    map_class [index] = map_class [index][0:1].abs()

## Union of the heatmaps of the Low, Medium and High class.
ale_df = pd.concat(map_class)
ale_df.index = name_class
ale_df = ale_df.transpose()
```

Generation of the heatmap.

In [608]:
```
plt.figure(figsize = (28,6))
sns.heatmap(ale_df.transpose(), cmap="Reds", cbar = False)
sns.heatmap(ale_df.transpose(), cmap="Blues", yticklabels = True, xticklabels = T
plt.show()
```



Comparing of Feature Variables Methods for Explainability

The global values of each of the methods used are taken, their maximums are obtained (the minimums are 0) and a percentage is calculated based on said maximum, which indicates how relevant the contribution is for each question.

In [609]:
```
## Class index local variable.
iterable_class = -1
```

Comparison of the explanation methods of the first class, in this case the Low.

The highest data of each method used is obtained.

In [610]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [611]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

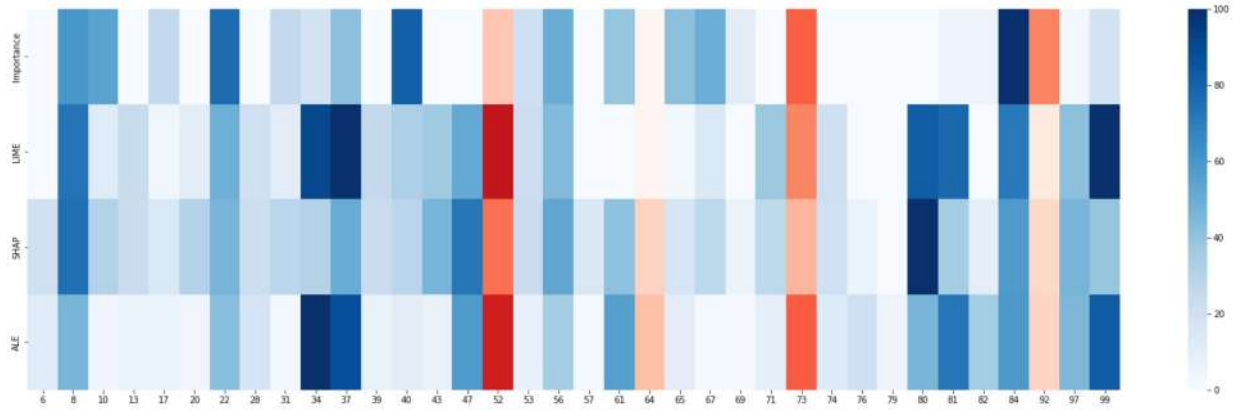
Out[611]:

	6	8	10	13	17	20	22	28
Importance	0.000000	60.526316	55.263158	0.000000	26.315789	0.000000	76.315789	0.000000
LIME	0.000000	73.743776	12.342839	24.574481	4.097176	10.182302	49.188266	19.878904
SHAP	19.463788	75.373848	30.547427	23.258230	13.597813	30.550677	46.580537	21.900598
ALE	11.994866	46.503117	3.943226	6.361579	6.193995	3.788199	42.193250	17.223967

4 rows × 36 columns

Generation of the heatmap.

In [612]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the second class, in this case the Medium.

The highest data of each method used is obtained.

In [613]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [614]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

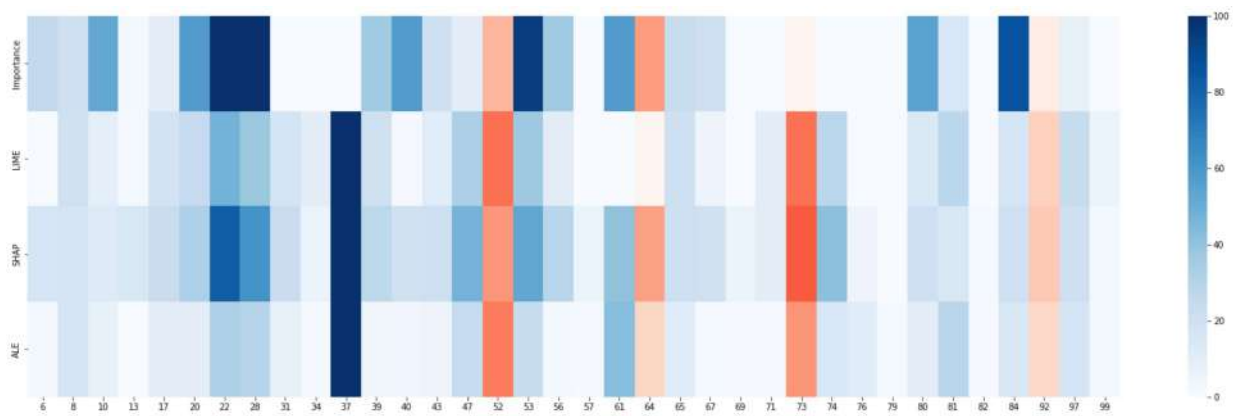
Out[614]:

	6	8	10	13	17	20	22	
Importance	26.315789	21.052632	52.631579	2.631579	10.526316	57.894737	100.000000	100.0000
LIME	0.000000	19.946484	9.366121	1.777002	18.889706	25.038928	48.058375	38.5189
SHAP	16.318418	17.506613	12.712310	15.114092	22.266669	32.613592	82.416871	61.939
ALE	2.425508	17.945531	7.843179	0.238994	10.015727	10.522439	32.703517	29.9844

4 rows × 36 columns

Generation of the heatmap.

In [615]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the third class, in this case the High.

The highest data of each method used is obtained.

In [616]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [617]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

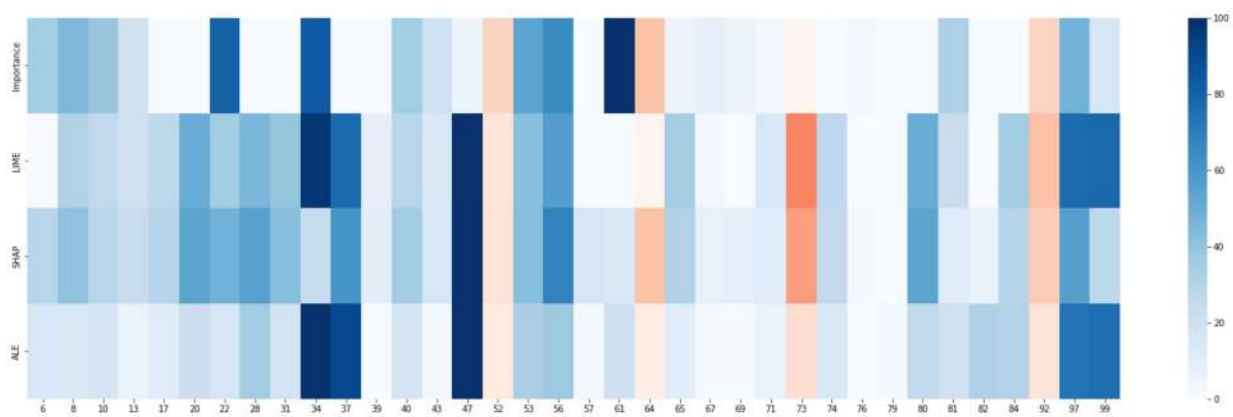
Out[617]:

	6	8	10	13	17	20	22	28
Importance	35.483871	45.161290	38.709677	19.354839	0.000000	0.000000	80.645161	0.000000
LIME	0.000000	31.859094	26.347727	19.918493	27.734191	50.372724	36.237166	46.233480
SHAP	29.389137	40.710599	28.276680	24.042544	29.786076	54.665933	47.735765	55.778416
ALE	15.763470	13.295816	17.501235	6.528994	11.561658	22.036810	16.647685	35.879833

4 rows × 36 columns

Generation of the heatmap.

In [618]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Safeguarding the items eliminated during this iteration.

In [619]:
```
## Removal of anomalous items selected by selection_mode.
for x in question_anomaly:

    ## Status change of the deleted items in the respective auxiliary dataset.
    ds.loc[x - 1,'Status'] = 'Delete'

## Save the changed statuses in the dataset.
ds.to_csv(path_dataset_qs, index = False)
```

Iteration 8

The first iteration of improvement of the model is carried out.

Preparing the data for the neural network

The data is prepared in order to be processed by the neural network.

In [620]:

```
## The iteration number is increased.
iteration += 1

## Loading data from the dataset.
df = pd.read_csv(path_dataset)
ds = pd.read_csv(path_dataset_qs)

## Elimination of unrelated items from the dataset.
if len(unrelated_questions) != 0:
    df = df.drop(df.columns[unrelated_questions[0]:unrelated_questions[1]], axis=

i = 0
if iteration == 1:
    ## Elimination of list of conflicting items declared in global variables.
    for x in list_contr:
        df.drop(df.columns[x - (i + 1)], axis = 1, inplace = True)
        i += 1
        ## Status change of the deleted question in the respective auxiliary data
        ds.loc[x - 1,'Status'] = 'Delete'

else:
    ## Elimination of the list of anomadic items defined in the improvement proce
    ds_delete = ds[ds.Status == "Delete"].reset_index(drop=True)
    for x in ds_delete['Question'].tolist():
        df.drop(df.columns[x - (i + 1)], axis = 1, inplace = True)
        i += 1

## Grouping of the target according to the defined dictionary.
if grouping_name != None and grouping_name != '':
    df[grouping_name] = df[grouping_name].map(class_dic)

## Assignment of local variables according to the data necessary for the neural r
train_cols = df.columns [0:-1]
label = df.columns [-1]
X = df [train_cols]
y = df [label]

## The prepared data of the dataset is printed.
df
```

Out[620]:

	6. I consider that my life is full of good things	8. I'm still as punctual as I have always been	10. I have found that I can spend a lot of time scrolling the screen \nof my cell phone without searching or stopping at anything in particular	13. I felt more encouraged to do my daily activities before	17. I feel guilty about the decisions that I have made	20. It is difficult for me to make decisions even if they are simple	22. It is not difficult for me to understand something the first time	28. The decisions I have made so far have been the right ones	e t p
0	1	1	0	0	0	1	1	0	
1	1	1	0	1	1	1	1	0	
2	1	0	1	0	0	0	1	1	
3	1	1	0	1	0	1	1	1	
4	1	1	1	0	1	1	0	0	
...	...	...	...	...	...	...	...	...	
99	1	1	1	1	0	1	0	1	
100	1	0	0	1	0	1	0	1	
101	1	0	1	1	0	0	0	0	
102	1	1	1	0	1	1	1	0	
103	1	0	1	0	0	1	0	1	

104 rows × 33 columns

Oversample application

An oversample process is applied to level the amount of existing data given by the grouping.

In [621]:
```
## The number of elements present in the grouped target is printed.
y.value_counts()
```

Out[621]:
```
0    39
1    35
2    30
Name: Target, dtype: int64
```

Oversample application.

In [622]:
```
## Application of the oversample to the prepared data.
X, y = oversample.fit_resample(X, y)
```

Printing of the data after the application of the oversample.

In [623]:
```
## The results obtained after applying the oversample are printed.
y.value_counts()
```

Out[623]:
```
0    39
1    39
2    39
Name: Target, dtype: int64
```

Preparation of the parameters of the neural network

We proceed to define the parameters necessary for the training of the neural network.

In [624]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
```

Neural network training

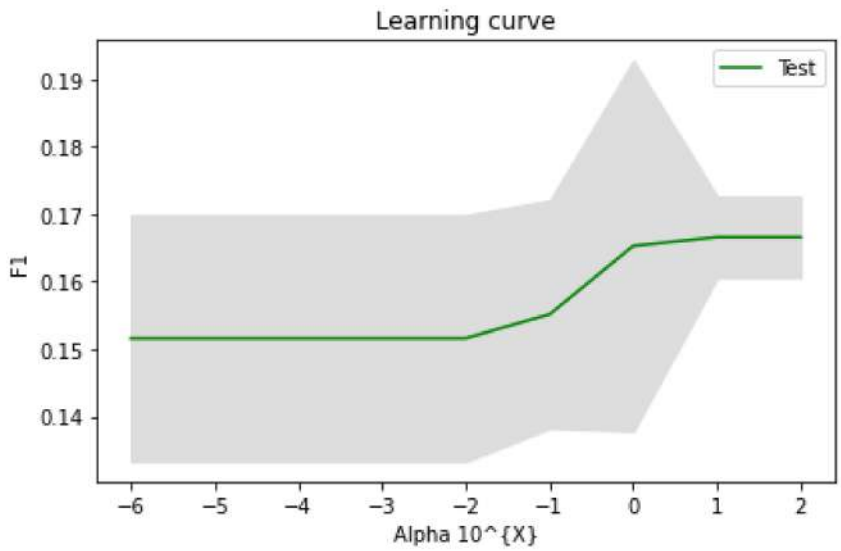
The alpha parameters and the initial learning rate are adjusted to obtain the best performance with the model using cross validation.

In [625]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_normal

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = 0.0001, max_iter =

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[625]:
```
([0.16659498207885304,
  0.16659498207885304,
  0.16529524637051518,
  0.155134148209417,
  0.15160924399485765,
  0.15160924399485765,
  0.15160924399485765,
  0.15160924399485765,
  0.15160924399485765],
 [0.006020651927912844,
  0.006020651927912844,
  0.027500591002963325,
  0.01691427217913298,
  0.018243397742091955,
  0.018243397742091955,
  0.018243397742091955,
  0.018243397742091955,
  0.018243397742091955])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [626]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Alpha 10^{X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



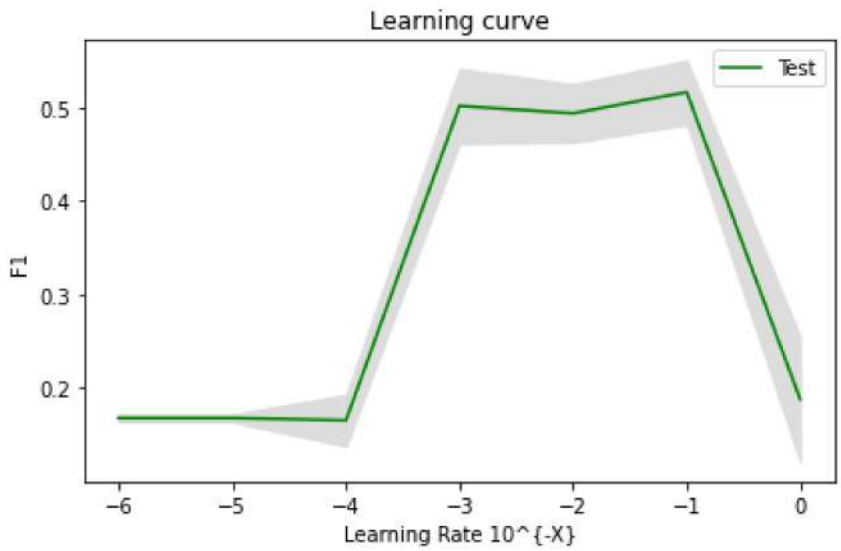
The training is performed again keeping the same parameters except for the alpha value that changes to 1.

In [627]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_alpha

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = regul_param, max_it

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[627]:
```
([0.18784775558969108,
  0.5158670895667801,
  0.4935746606334842,
  0.5013452526146025,
  0.16529524637051518,
  0.16766777901371893,
  0.16766777901371893],
 [0.06633424444700753,
  0.0343070171358755,
  0.03097449228215081,
  0.039941565701260624,
  0.027500591002963325,
  0.004143150590694191,
  0.004143150590694191])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [628]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Learning Rate 10^{-X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



Generation of the neural network model

The parameters of the final model are modified according to the data obtained in the training of the neural network.

In [629]:
```
## The value of the alpha parameter, learning_rate_init and random_state are char
## to the data in the global variables.
mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solver =
                    learning_rate = 'constant', learning_rate_init = lear
                    random_state = random_state_model)

## We print the data of the final generated model.
mlp.fit(X_train, y_train)
```

Out[629]:
```
MLPClassifier
MLPClassifier(activation='logistic', alpha=0.001, hidden_layer_sizes=(10,),
              learning_rate_init=0.1, max_iter=100000, random_state=42)
```

We safeguard the final generated model

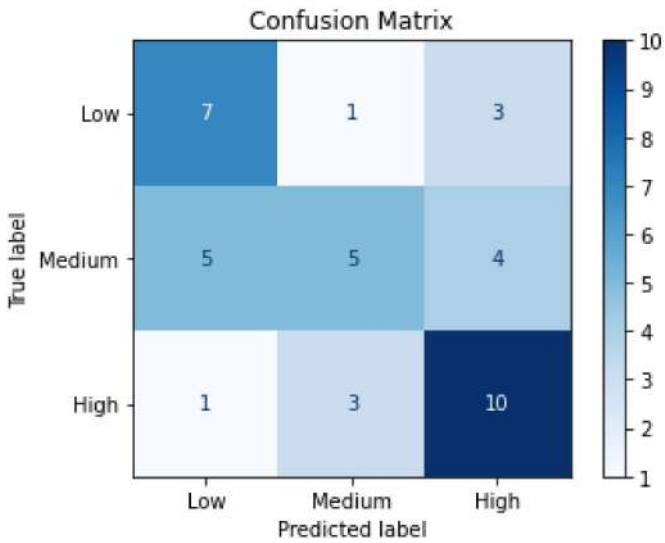
In [630]:
```
joblib.dump(mlp,"model_depression_i" + str(iteration) + ".pkl")
```

Out[630]:
```
['model_depression_i8.pkl']
```

Generation of the confusion matrix of the model

We obtain the statistics of the final model to generate its confusion matrix.

In [631]:
```
## The data for the generation of the confession matrix are defined.
confusion_matrix = ConfusionMatrixDisplay.from_estimator(mlp, X_test, y_test, dis
                                                         cmap = plt.cm.Blues)
confusion_matrix.ax_.set_title("Confusion Matrix")
plt.show()
```



We get the detailed statistics of the final model.

In [632]:
```
## The necessary detailed prediction is generated.
y_pred = mlp.predict(X_test)

## The detailed statistics of the model are printed.
print('Classification accuracy =',accuracy_score(y_test,y_pred) * 100,'%\n')
print(classification_report(y_test,y_pred))
```

```
Classification accuracy = 56.41025641025641 %

              precision    recall  f1-score   support

           0       0.54      0.64      0.58        11
           1       0.56      0.36      0.43        14
           2       0.59      0.71      0.65        14

    accuracy                           0.56        39
   macro avg       0.56      0.57      0.55        39
weighted avg       0.56      0.56      0.55        39
```

The data obtained is saved for later graphing.

In [633]:
```
accuracy = accuracy_score(y_test,y_pred) * 100
clf_rep = precision_recall_fscore_support(y_test,y_pred)

## The number of items used in the iteration is retrieved.
ds_delete = ds[ds.Status == "Delete"]
ds_delete.reset_index(inplace=True, drop=True)

## All the metrics of the confusion matrix are obtained.
metrics = [",".join(map(str, ds_delete['Question'].tolist())),
           accuracy]

for index in range(0, len(name_class)):
    metrics.append(clf_rep[0][index])
    metrics.append(clf_rep[1][index])
    metrics.append(clf_rep[2][index])
    metrics.append(clf_rep[3][index])

## The names of the columns of the dataset are defined.
columns = ['Question', 'Acurracy global']

for index in range(0, len(name_class)):
    columns.append('Precision_' + str(index))
    columns.append('Recall_' + str(index))
    columns.append('F1_score_' + str(index))
    columns.append('Support_' + str(index))

## A new row of the dataset is generated with all the data.
data_metrics = pd.DataFrame([metrics], columns = columns)

## The data is saved to the dataset.
data_metrics.to_csv(path_dataset_qd, mode = 'a', header = False, index = False)

## The data added to the dataset is printed.
data_metrics
```

Out[633]:

	Question	Acurracy global	Precision_0	Recall_0	F1_score_0	Supp
0	1,2,3,4,5,7,9,11,12,14,15,16,18,19,21,23,24,25...	56.410256	0.538462	0.636364	0.583333	

Improvement process of the neural network model.

The application of the explanation method by ALE allows to determine the behavior of the neural network and thereby improve it by discarding non-significant data for it.

Slope Analysis of ALE Plots

In [634]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

We proceed to obtain the graphs to obtain the slopes of the selected classes.

In [635]:
```
## The question metrics dataset is loaded.
dm = pd.DataFrame()

## The number of items used in the iteration is retrieved.
ds_item = ds[ds.Status == "In use"]
ds_item.reset_index(inplace=True, drop=True)

## We get the slopes from the ALE data.
for i in range(df.shape[1]-1):
    structure_data = {'Question': ds_item.loc[i, "Question"]}

    for index in selection_class:
        slope = proba_exp_lr.ale_values [i][1][index] - proba_exp_lr.ale_values [
        structure_data['Slope ' + str(target_names[index])] = slope
        structure_data['Threshold ' + str(target_names[index])] = 'NA'
        structure_data['Anomaly ' + str(target_names[index])] = 'NA'

    dm = dm.append(structure_data, ignore_index=True)

## The current rating data dataset is printed.
dm
```

```
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
```

```
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
```

Out[635]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	6	-0.016898	NA	NA	-0.009221	NA	NA
1	8	0.140217	NA	NA	-0.076296	NA	NA
2	10	-0.176113	NA	NA	0.072612	NA	NA
3	13	-0.177525	NA	NA	0.068674	NA	NA
4	17	0.126749	NA	NA	0.019309	NA	NA
5	20	-0.123725	NA	NA	0.266977	NA	NA
6	22	-0.033232	NA	NA	-0.006150	NA	NA
7	28	0.032940	NA	NA	-0.143376	NA	NA
8	31	0.139452	NA	NA	-0.099286	NA	NA
9	34	-0.198799	NA	NA	0.159840	NA	NA
10	37	-0.229330	NA	NA	-0.244403	NA	NA
11	39	-0.059345	NA	NA	-0.041391	NA	NA
12	40	0.062790	NA	NA	-0.022374	NA	NA
13	43	0.018303	NA	NA	0.003914	NA	NA
14	47	0.146373	NA	NA	-0.321444	NA	NA
15	53	0.113668	NA	NA	0.227111	NA	NA

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
16	56	0.286560	NA	NA	-0.260347	NA	NA
17	57	-0.096207	NA	NA	0.096703	NA	NA
18	61	0.129353	NA	NA	-0.024400	NA	NA
19	65	-0.083473	NA	NA	0.056345	NA	NA
20	67	-0.162574	NA	NA	0.038037	NA	NA
21	69	-0.000428	NA	NA	-0.000305	NA	NA
22	71	-0.064951	NA	NA	-0.019088	NA	NA
23	74	0.150339	NA	NA	0.045915	NA	NA
24	76	-0.026505	NA	NA	0.001478	NA	NA
25	79	0.020513	NA	NA	-0.019847	NA	NA
26	80	0.163403	NA	NA	-0.099305	NA	NA
27	81	-0.269764	NA	NA	0.075097	NA	NA
28	82	-0.339421	NA	NA	0.127423	NA	NA
29	84	0.274748	NA	NA	-0.155498	NA	NA
30	97	-0.204237	NA	NA	0.129716	NA	NA
31	99	0.505799	NA	NA	-0.228619	NA	NA

Slope analysis and threshold application

The slope obtained is analyzed and those that are above or below the positive and negative threshold are selected.

In [636]:
```
for i in range(df.shape[1]-1):
    for index in selection_class:

        ## The slopes are selected according to the values of the thresholds of t
        if dm.loc[i, "Slope " + str(target_names[index])] >= positive_threshold:
            dm.loc[i, "Threshold " + str(target_names[index])] = 1
        elif dm.loc[i, "Slope " + str(target_names[index])] <= negative_threshold
            dm.loc[i, "Threshold " + str(target_names[index])] = 0

## The current rating data dataset is printed.
dm
```

Out[636]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	6	-0.016898	0	NA	-0.009221	NA	NA
1	8	0.140217	1	NA	-0.076296	0	NA
2	10	-0.176113	0	NA	0.072612	1	NA
3	13	-0.177525	0	NA	0.068674	1	NA
4	17	0.126749	1	NA	0.019309	1	NA
5	20	-0.123725	0	NA	0.266977	1	NA
6	22	-0.033232	0	NA	-0.006150	NA	NA
7	28	0.032940	1	NA	-0.143376	0	NA
8	31	0.139452	1	NA	-0.099286	0	NA
9	34	-0.198799	0	NA	0.159840	1	NA
10	37	-0.229330	0	NA	-0.244403	0	NA
11	39	-0.059345	0	NA	-0.041391	0	NA
12	40	0.062790	1	NA	-0.022374	0	NA
13	43	0.018303	1	NA	0.003914	NA	NA
14	47	0.146373	1	NA	-0.321444	0	NA
15	53	0.113668	1	NA	0.227111	1	NA
16	56	0.286560	1	NA	-0.260347	0	NA
17	57	-0.096207	0	NA	0.096703	1	NA
18	61	0.129353	1	NA	-0.024400	0	NA
19	65	-0.083473	0	NA	0.056345	1	NA
20	67	-0.162574	0	NA	0.038037	1	NA
21	69	-0.000428	NA	NA	-0.000305	NA	NA
22	71	-0.064951	0	NA	-0.019088	0	NA
23	74	0.150339	1	NA	0.045915	1	NA
24	76	-0.026505	0	NA	0.001478	NA	NA
25	79	0.020513	1	NA	-0.019847	0	NA
26	80	0.163403	1	NA	-0.099305	0	NA

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
27	81	-0.269764	0	NA	0.075097	1	NA
28	82	-0.339421	0	NA	0.127423	1	NA
29	84	0.274748	1	NA	-0.155498	0	NA
30	97	-0.204237	0	NA	0.129716	1	NA
31	99	0.505799	1	NA	-0.228619	0	NA

Analysis and determination of anomalous items

The anomalous items present in the model are determined based on their slope and expected response from the expert.

In [637]:

```
## The dataset of expected responses is loaded.
da = pd.read_csv(path_dataset_qa)

## Delete list of items deleted in this iteration.
i = 0
for x in ds_delete['Question']:
    da.drop(da.index[x - (i + 1)], axis = 0, inplace = True)
    i += 1

## Indexes are restored for later use.
da.reset_index(inplace=True, drop=True)

## The dataset of expected responses is printed.
da
```

Out[637]:

	Number	Question	RE
0	6	I consider that my life is full of good things	0
1	8	I'm still as punctual as I have always been	0
2	10	I have found that I can spend a lot of time sc...	1
3	13	I felt more encouraged to do my daily activiti...	1
4	17	I feel guilty about the decisions that I have ...	1
5	20	It is difficult for me to make decisions even ...	1
6	22	It is not difficult for me to understand somet...	0
7	28	The decisions I have made so far have been the...	0
8	31	It is easy for me to choose a photograph of my...	0
9	34	In recent months I have had discussions with m...	1
10	37	I'm not too worried about what might happen in...	0
11	39	Everything will be alright	0
12	40	I can easily blank my mind	0
13	43	Sometimes i feel trapped	1
14	47	I can clearly express to others how I feel	0
15	53	I'm bothered by people walking slowly in front...	1
16	56	I manage my schedule as I always have	0
17	57	It bothers me to feel that people on the stree...	1
18	61	Sometimes I forget what I wanted to say becaus...	1
19	65	I worry that people will not understand what I...	1
20	67	I get angry easily	1
21	69	It is not important for me to meet set dates	1
22	71	I use lies just to get out of certain problems	1
23	74	People have problems because of themselves	1
24	76	Laws are not as important as others think	1
25	79	It is easy for me to work in a team	0

	Number	Question	RE
26	80	It is important to help people when they need it	0
27	81	I have punched someone or thought of doing it	1
28	82	If it was necessary I would pretend to be some...	1
29	84	After an argument I usually go over what happe...	0
30	97	People who commit crimes have their reasons fo...	1
31	99	It is important to respect turns	0

We proceed to identify the anomadic items present in the model present in this iteration.

In [638]:

```
for i in range(df.shape[1]-1):

    ## Local variable to indicate the index of the selected classes.
    i_index = 0

    for index in selection_class:

        ## Determine if it fails based on the expected response parameters of the
        if dm.loc[i, "Threshold "  + str(target_names[index])] == 1 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 1 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 0 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 0 and da.loc[i,
            dm.loc[i, "Anomaly "  + str(target_names[index])] = expected_response

        ## Increase of the index of the selected classes.
        i_index += 1

## The final grade data dataset is printed.
dm
```

Out[638]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	6	-0.016898	0	1	-0.009221	NA	NA
1	8	0.140217	1	0	-0.076296	0	0
2	10	-0.176113	0	0	0.072612	1	0
3	13	-0.177525	0	0	0.068674	1	0
4	17	0.126749	1	1	0.019309	1	0
5	20	-0.123725	0	0	0.266977	1	0
6	22	-0.033232	0	1	-0.006150	NA	NA
7	28	0.032940	1	0	-0.143376	0	0
8	31	0.139452	1	0	-0.099286	0	0
9	34	-0.198799	0	0	0.159840	1	0
10	37	-0.229330	0	1	-0.244403	0	0
11	39	-0.059345	0	1	-0.041391	0	0
12	40	0.062790	1	0	-0.022374	0	0
13	43	0.018303	1	1	0.003914	NA	NA
14	47	0.146373	1	0	-0.321444	0	0
15	53	0.113668	1	1	0.227111	1	0
16	56	0.286560	1	0	-0.260347	0	0
17	57	-0.096207	0	0	0.096703	1	0
18	61	0.129353	1	1	-0.024400	0	1
19	65	-0.083473	0	0	0.056345	1	0

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
20	67	-0.162574	0	0	0.038037	1	0
21	69	-0.000428	NA	NA	-0.000305	NA	NA
22	71	-0.064951	0	0	-0.019088	0	1
23	74	0.150339	1	1	0.045915	1	0
24	76	-0.026505	0	0	0.001478	NA	NA
25	79	0.020513	1	0	-0.019847	0	0
26	80	0.163403	1	0	-0.099305	0	0
27	81	-0.269764	0	0	0.075097	1	0
28	82	-0.339421	0	0	0.127423	1	0
29	84	0.274748	1	0	-0.155498	0	0
30	97	-0.204237	0	0	0.129716	1	0
31	99	0.505799	1	0	-0.228619	0	0

Removing identified anomadic items from the model

We proceed to eliminate the identified anomadic items so as not to use them in the next iteration.

In [639]:
```
## Local variable for the elimination of the anomaly items.
instructions = ''
init = False
question_anomaly = []

if selection_mode == 1:

    ## Items that fail in the defined class are retrieved.
    dm_delete = dm[dm['Anomaly ' + str(target_names[select_class_mode])] == 1]
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 2:

    ## Items that fail in classes defined with the or condition type are retrieve
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 1"
            init = True
        else:
            instructions += " or `Anomaly " + str(target_names[index]) + "` == 1'

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 3:

    ## Items that fail in classes defined with the and condition type are retriev
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 1"
            init = True
        else:
            instructions += " and `Anomaly " + str(target_names[index]) + "` == 1

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 4:

    ## Out-of-threshold items are retrieved in classes defined with the conditior
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 'NA'"
            init = True
        else:
            instructions += " and `Anomaly " + str(target_names[index]) + "` == '

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

## The selected blank items are printed according to the selection_mode.
question_anomaly
```

Out[639]: [61, 71]

Model explainability process

The data used for the test of the neural network model is shown.

In [640]: X_test

Out[640]:

	6. I consider that my life is full of good things	8. I'm still as punctual as I have always been	10. I have found that I can spend a lot of time scrolling the screen \nof my cell phone without searching or stopping at anything in particular	13. I felt more encouraged to do my daily activities before	17. I feel guilty about the decisions that I have made	20. It is difficult for me to make decisions even if they are simple	22. It is not difficult for me to understand something the first time	28. The decisions I have made so far have been the right ones	easy to cl phot of to or
69	1	0	0	0	0	1	1	0	
46	1	1	1	1	1	0	1	1	
58	1	1	0	1	0	0	1	1	
114	0	0	1	1	1	0	1	0	
73	1	1	1	0	0	0	1	1	
98	0	1	1	0	0	1	1	0	
31	1	0	0	0	1	0	1	1	
53	1	1	1	1	1	0	1	0	
65	1	1	0	1	0	0	1	1	
96	1	1	1	0	1	0	1	0	
95	1	0	1	0	0	1	0	0	
97	1	1	0	0	1	1	1	1	
2	1	0	1	0	0	0	1	1	
62	1	0	1	0	0	1	0	1	
110	0	1	0	0	1	0	0	0	
55	1	1	0	0	1	1	0	0	
103	1	0	1	0	0	1	0	1	
100	1	0	0	1	0	1	0	1	
66	0	0	1	0	1	0	1	0	
44	1	1	0	0	0	0	0	1	
77	1	1	1	0	1	1	1	1	
17	1	1	1	1	0	1	1	1	
81	1	1	0	1	1	0	0	0	
74	0	1	1	1	1	1	0	0	

56	1	1	0	1	1	0	1	0
94	0	0	1	1	1	0	1	0
35	1	1	0	0	0	1	1	1
38	1	1	0	0	0	1	0	1
93	1	1	1	1	0	1	0	1
48	1	0	1	1	1	1	1	1
33	1	0	1	1	1	0	1	0
59	1	1	1	0	1	0	0	0
92	0	1	0	0	1	1	1	0
42	1	1	0	0	0	0	0	1
10	1	0	1	0	0	0	0	0
85	1	1	1	1	1	0	1	0
32	1	0	1	1	1	1	0	0
78	1	1	0	0	0	1	1	1
19	0	1	0	1	0	0	0	1

39 rows × 32 columns

Explanation of the model using ALE

In analysis of the model through the results obtained by ALE, it is possible to identify the behavior of the neural network.

In [641]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

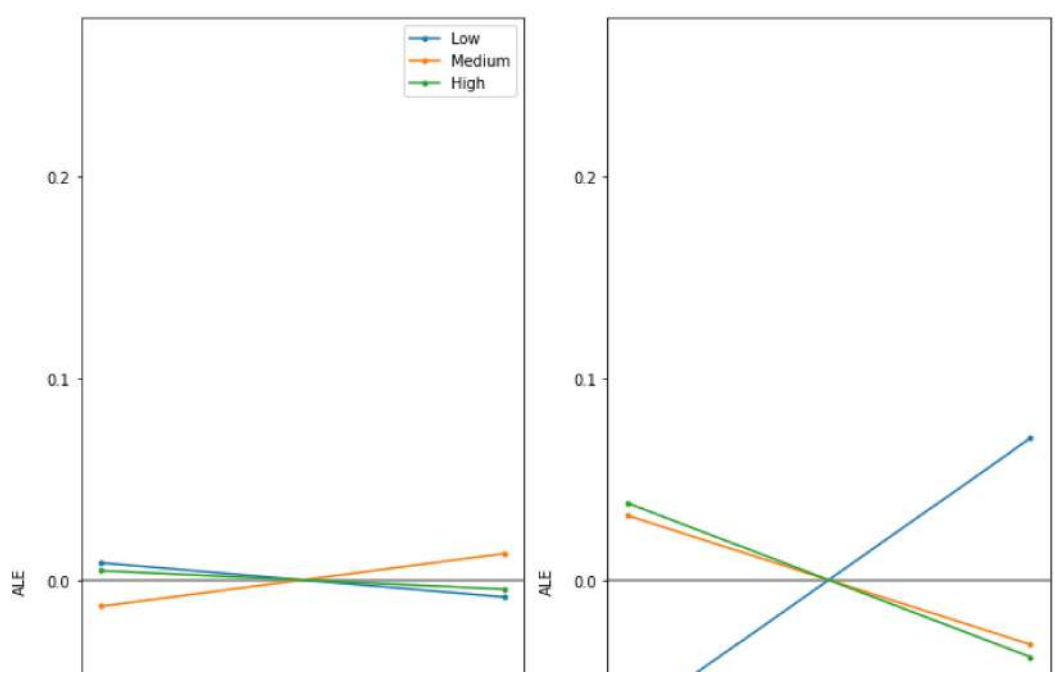
```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

The graphs of all the data present in the dataset used are shown.

In [642]:
```
plot_ale(proba_exp_lr, n_cols=2, features=list(range(df.shape[1]-1)),fig_kw={'fig
```



Explanation of the model using LIME

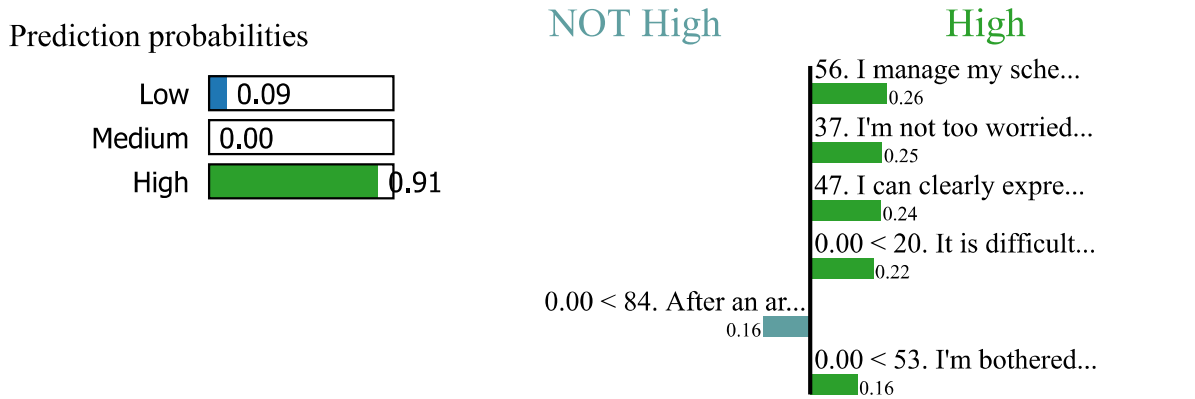
We proceed to explain the improved model using LIME for the Low, Medium and High classes.

In [643]:
```
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorica
                                                   feature_names=train_cols, clas
                                                   discretize_continuous=True)
```

The first individual present in the test data is explained.

In [644]:
```
exp = explainer.explain_instance(X_test.to_numpy()[0], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

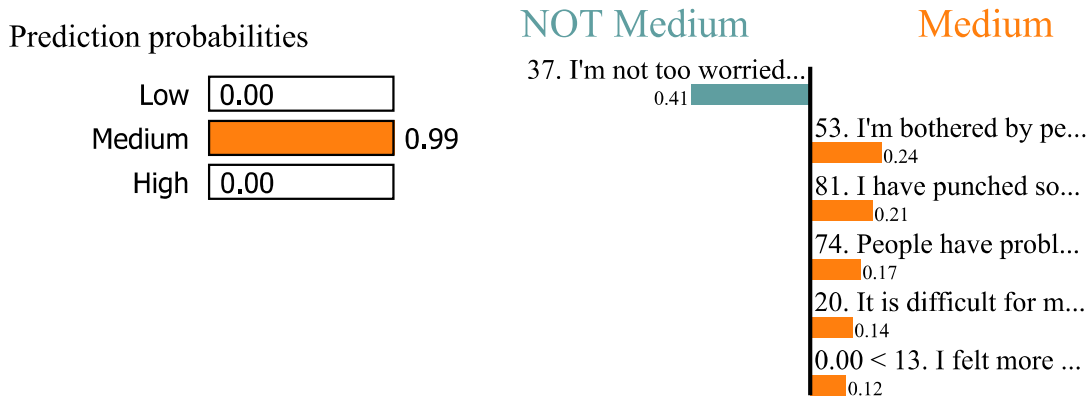


Feature	Value
56. I manage my schedule as I always have	0.00
37. I'm not too worried about what might happen in a few weeks	0.00
47. I can clearly express to others how I feel	0.00
20. It is difficult for me to make decisions even if they are simple	1.00
84. After an argument I usually go over what happened in my head	1.00
53. I'm bothered by people walking slowly in front of me	1.00

The second individual present in the test data is explained.

In [645]:
```
exp = explainer.explain_instance(X_test.to_numpy()[1], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

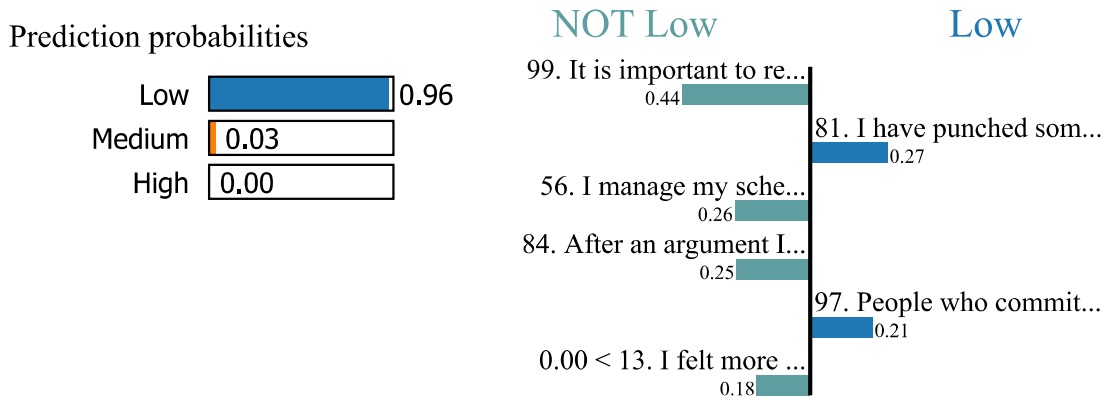


Feature	Value
37. I'm not too worried about what might happen in a few weeks	0.00
53. I'm bothered by people walking slowly in front of me	0.00
81. I have punched someone or thought of doing it	1.00
74. People have problems because of themselves	0.00
20. It is difficult for me to make decisions even if they are simple	0.00
13. I felt more encouraged to do my daily activities before	1.00

The last individual present in the test data is explained.

In [646]:
```
exp = explainer.explain_instance(X_test.to_numpy()[-1], mlp.predict_proba, num_fe
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```



Feature	Value
99. It is important to respect turns	0.00
81. I have punched someone or thought of doing it	0.00
56. I manage my schedule as I always have	0.00
84. After an argument I usually go over what happened in my head	0.00
97. People who commit crimes have their reasons for doing it	0.00
13. I felt more encouraged to do my daily activities before	1.00

Explanation of the model using SHAP

SHAP explanatory values are displayed according to the class to which the selected individual belongs according to the prediction of the model.

The first individual present in the test data is explained.

In [647]:
```
## Individual Selection.
individual = X_test.iloc[[0]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The second individual present in the data test is explained.

In [648]:
```
## Individual Selection.
individual = X_test.iloc[[1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plo
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The last individual present in the data test is explained.

In [649]:
```
## Individual Selection.
individual = X_test.iloc[[-1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

Explanation using heatmaps

The importance of the data present in the neural network is shown from explanation methods.

In [650]:

```
## Items currently in use are retrieved.
ds_item = ds[ds.Status == "In use"].reset_index(drop=True)

## The data is prepared for the generation of heatmaps.
column_Map = ds_item['Question'].tolist()
content_Map = [False for x in range(1, df.shape[1])]

## The next items to be deleted will be marked.
for index in range(0, len(content_Map)):

    if iteration == 0:
        for delete in list_contr:
            if column_Map[index] == delete:
                content_Map [index] = True
    else:
        for delete in question_anomaly:
            if column_Map[index] == delete:
                content_Map [index] = True

## The dataframe is created with the prepared data.
df_mask = pd.DataFrame([content_Map], columns = column_Map)
df_mask_imp = pd.concat([df_mask, df_mask, df_mask], ignore_index = True, axis =
df_mask_all = pd.concat([df_mask, df_mask, df_mask, df_mask], ignore_index = True

df_mask
```

Out[650]:

	6	8	10	13	17	20	22	28	31	34	...	71	74	76	79
0	False	False	False	False	False	False	False	False	False	False	...	True	False	False	False

1 rows × 32 columns

Identification of feature variables

Identification of feature variables from the randomness of the answers as items of the data.

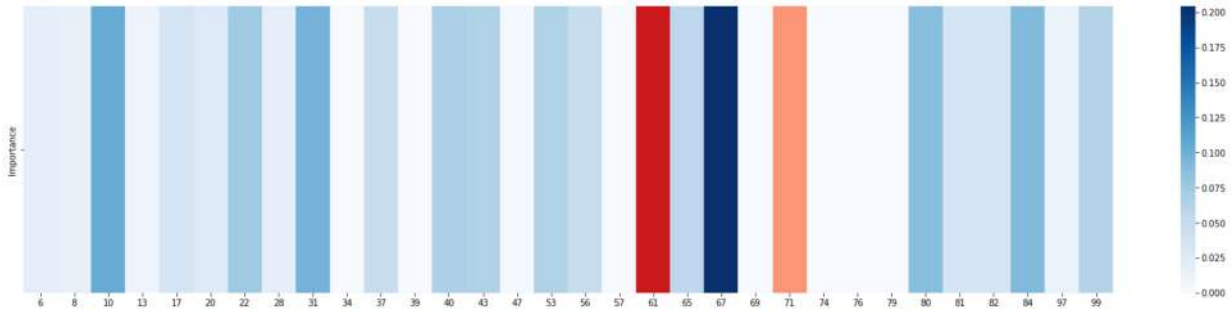
In [651]:
```
## Loading the data for explainability.
explainer = ClassifierExplainer(mlp, X_test, y_test)

## Class index local variable.
iterable_class = -1
map_class = []
```

```
WARNING: Parameter shap='guess', but failed to guess the type of shap explainer
to use for MLPClassifier. Defaulting to the model agnostic shap.KernelExplainer
(shap='kernel'). However this will be slow, so if your model is compatible with
e.g. shap.TreeExplainer or shap.LinearExplainer then pass shap='tree' or shap
='linear'!
WARNING: For shap='kernel', shap interaction values can unfortunately not be ca
lculated!
Note: shap values for shap='kernel' normally get calculated against X_backgroun
d, but paramater X_background=None, so setting X_background=shap.sample(X, 5
0)...
Generating self.shap_explainer = shap.KernelExplainer(model, X, link='identit
y')
```

Explainability of the data belonging to the first class, in this case the Low.

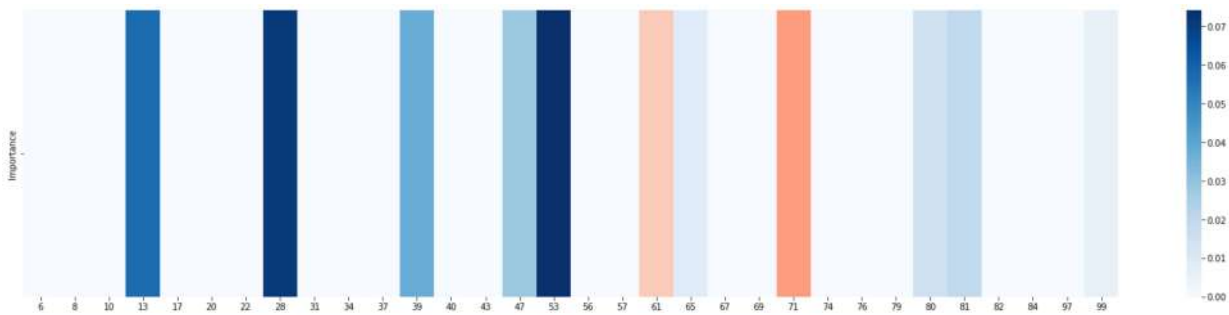
In [652]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

```
Calculating permutation importances (if slow, try setting n_jobs parameter)...
```



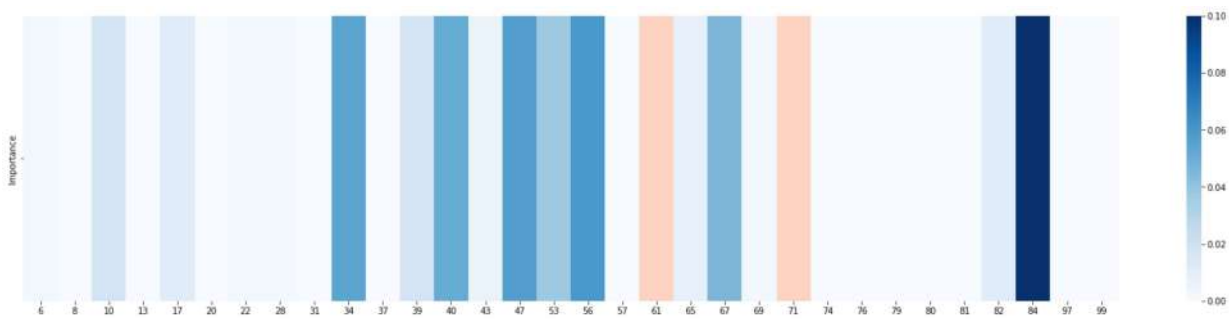
Explainability of the data belonging to the second class, in this case the Medium.

In [653]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



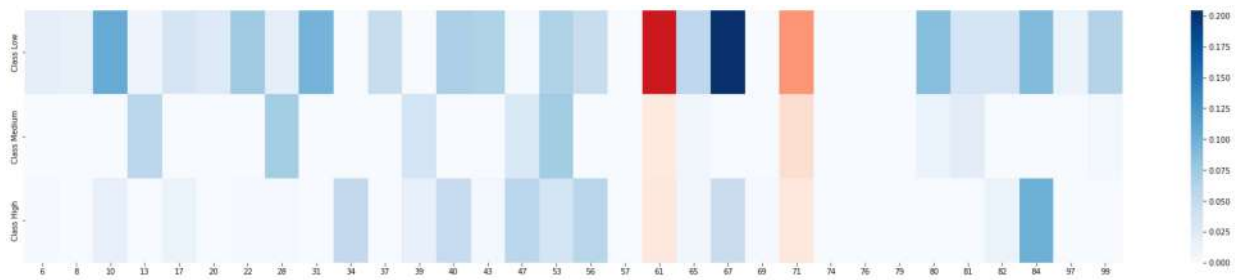
Explainability of the data belonging to the third class, in this case the High.

In [654]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [655]:
```
## Fusion of the previous heatmaps.
df_importance = pd.concat(map_class, axis=1)
df_importance.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_importance.transpose(), cmap = "Reds", cbar = False)
sns.heatmap(df_importance.transpose(), cmap = "Blues",yticklabels = True, xtickla
plt.show()
```



Identification of feature variables using SHAP

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of SHAP.

In [656]:
```
## Class index local variable.
iterable_class = -1
map_class = []
```

Individual explainability of the data belonging to the first class, in this case the Low.

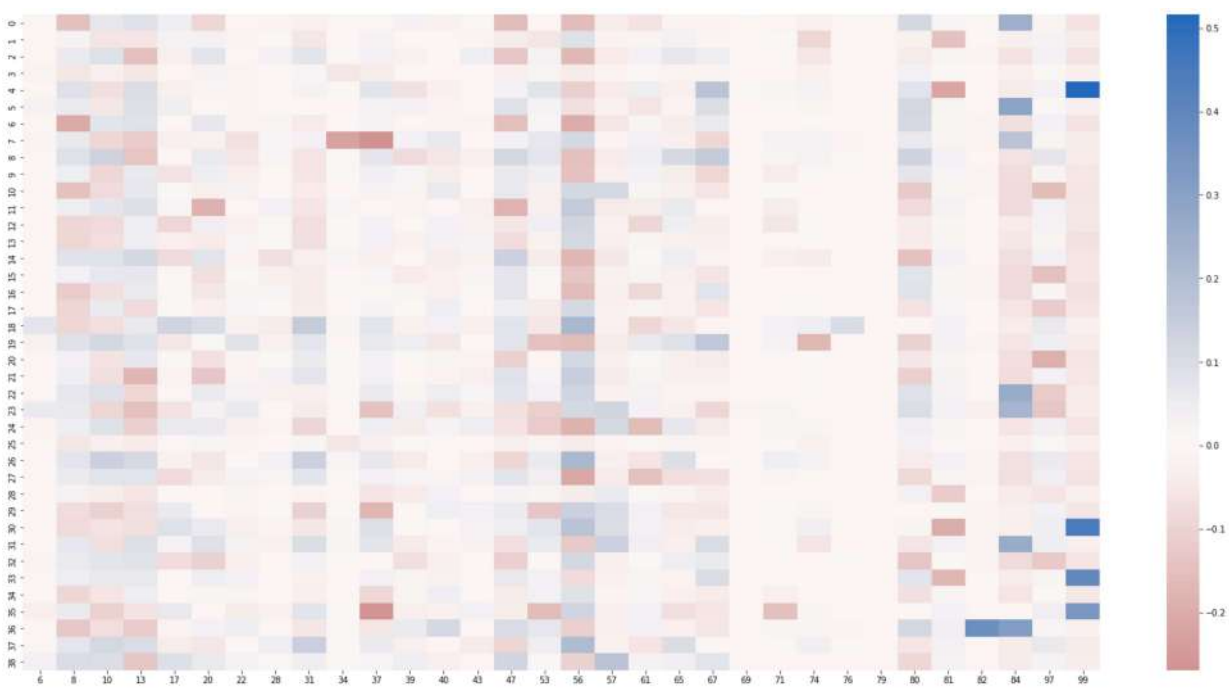
Definition of explainability data.

In [657]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

```
Calculating shap values...

  0%|          | 0/39 [00:00<?, ?it/s]
```

Generation of the heatmap.

In [658]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

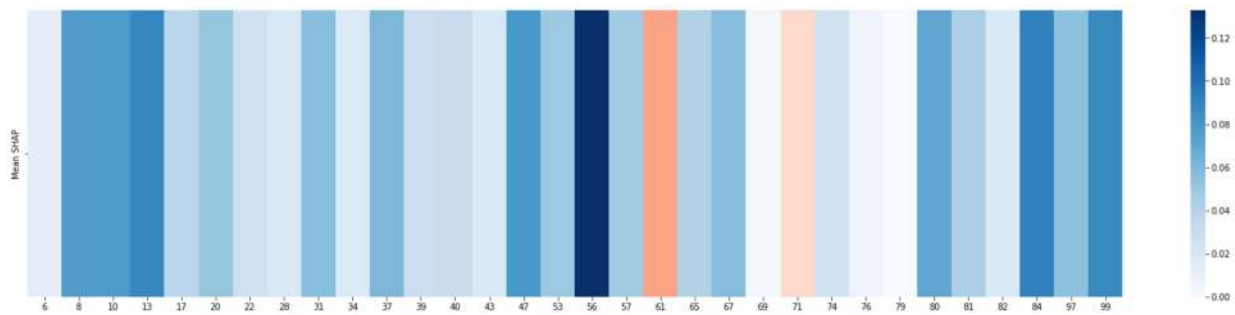


Fusion of the data used.

In [659]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [660]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



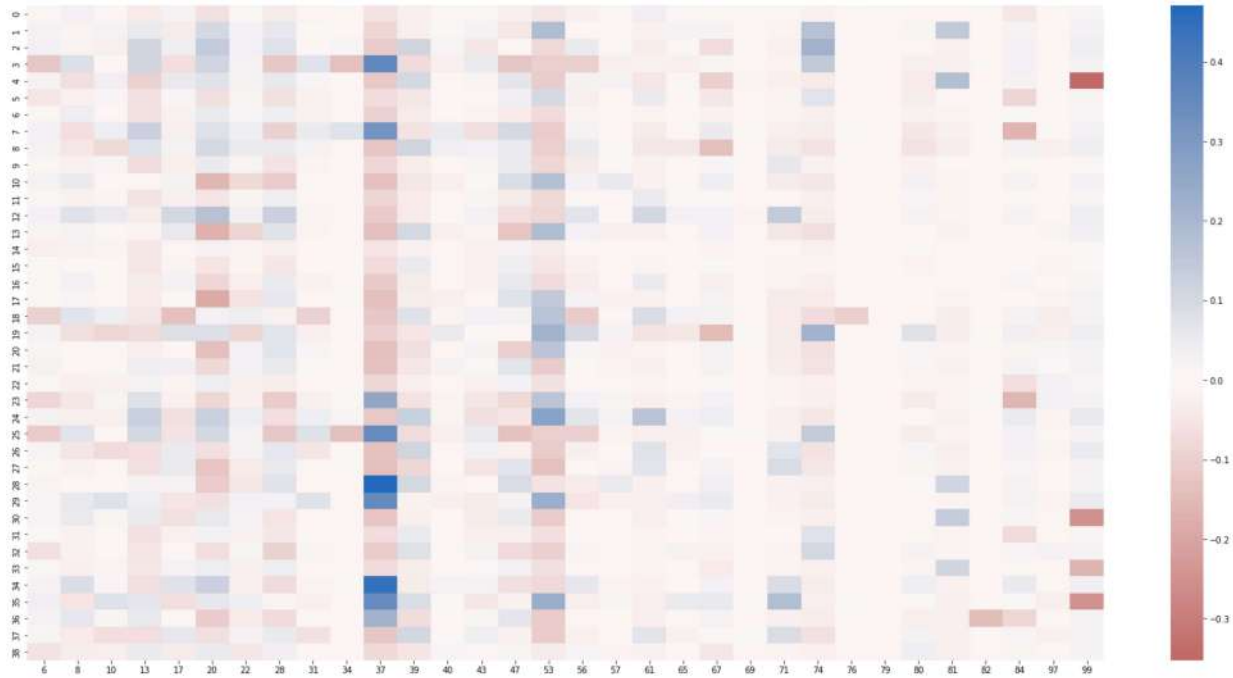
Individual explainability of the data belonging to the second class, in this case the Medium.

Definition of explainability data.

In [661]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [662]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

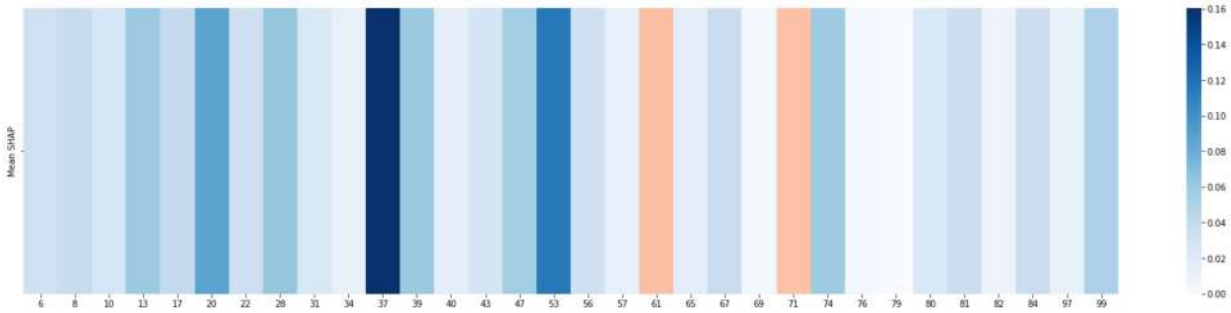


Fusion of the data used.

In [663]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [664]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



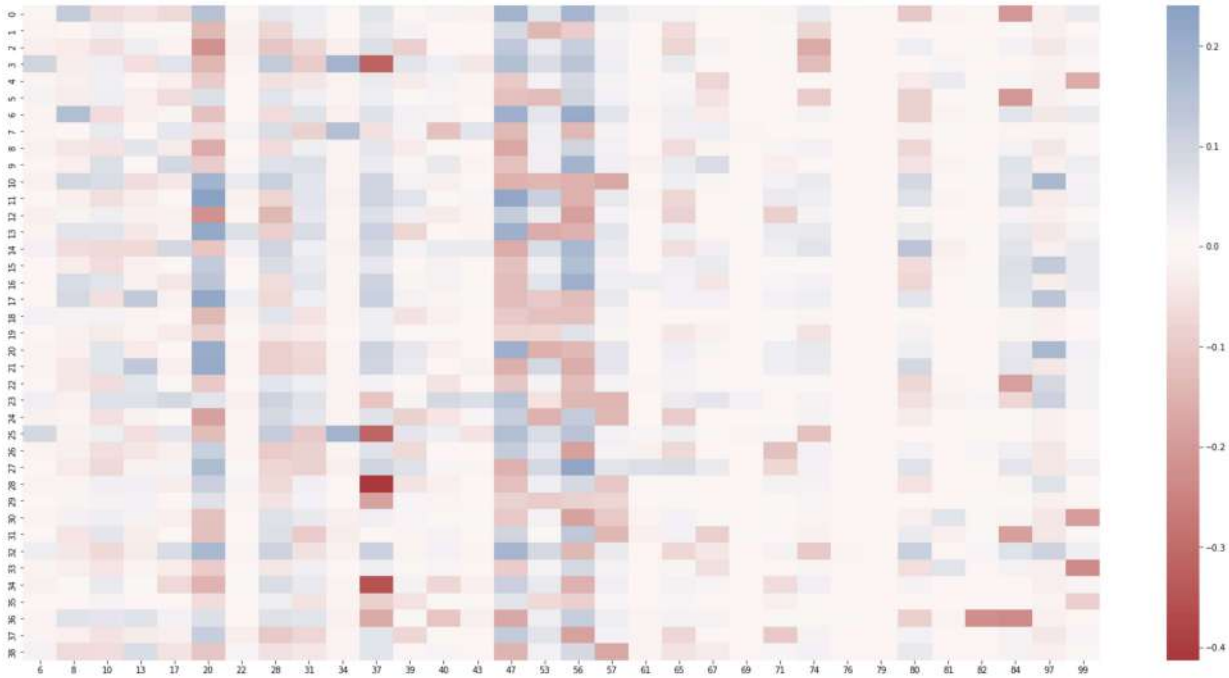
Individual explainability of data belonging to the third class, in this case High.

Definition of explainability data.

In [665]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [666]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

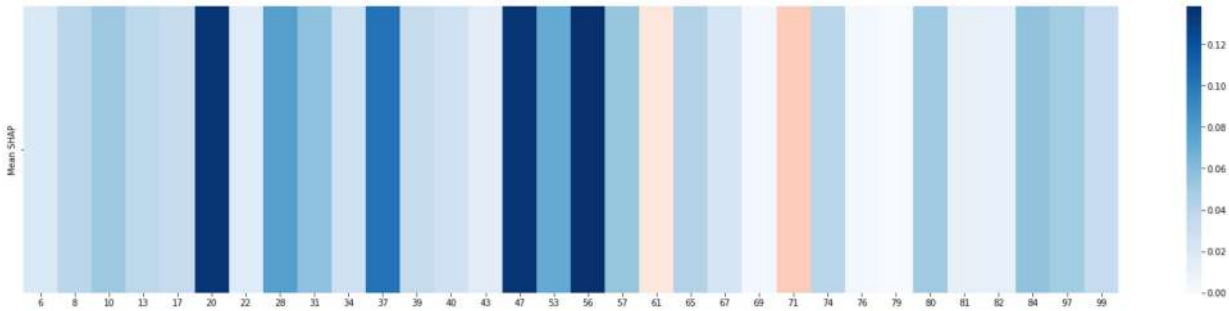


Fusion of the data used.

In [667]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

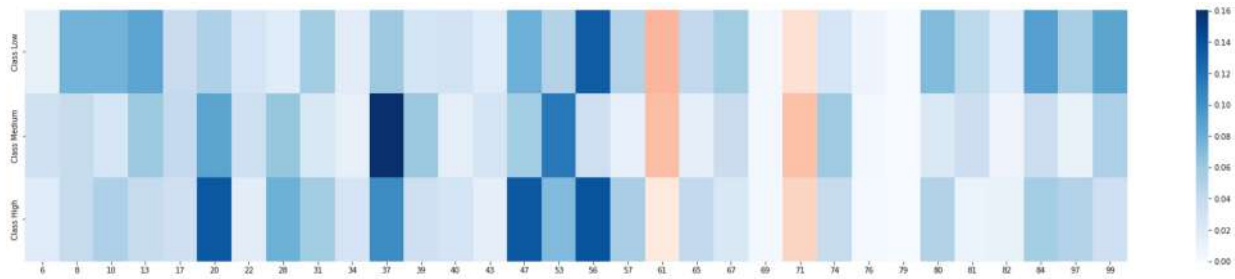
Generation of the heatmap.

In [668]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [669]:
```
## Fusion of the previous heatmaps.
df_shap = pd.concat(map_class, axis=1)
df_shap.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_shap.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_shap.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using LIME

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of LIME.

In [670]:
```
## Data preparation for explainability.
lime_exp = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorical
                                                  feature_names=train_cols, class
                                                  discretize_continuous=True)

## Class index local variable.
iterable_class = -1
map_class = []
```

Obtaining the data for the generation of the explanations by LIME about the Low, Medium and High classes.

In [671]:
```
## Local variables for LIME array data.
exp_matrix = [[] for _ in range(len(name_class))]

## Recovering the data needed for LIME.
for x in X_test.to_numpy():

    ## Auxiliary temporary variables.
    exp_list = [[] for _ in range(len(name_class))]

    ## Loading the data for explainability.
    exp = lime_exp.explain_instance(x, mlp.predict_proba, num_features = df.shape

    ## Recovering the data.
    for elements in range(0, len(name_class)):
        temp = exp.as_map()[elements]
        temp.sort(key=itemgetter(0))

        ## Saving data of the Low, Medium and High classes.
        for tup in temp:
            exp_list[elements].append(tup[1])

        exp_matrix[elements].append(exp_list[elements])
```

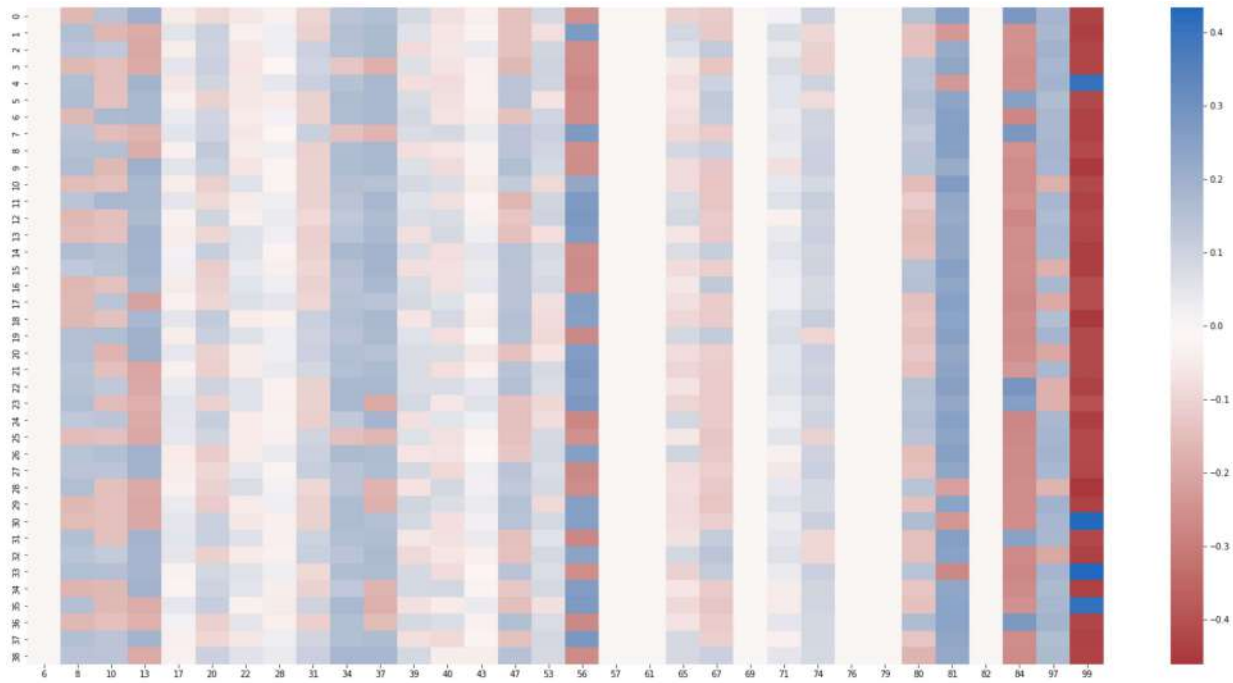
```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

Individual explainability of the data belonging to the first class, in this case the Low.

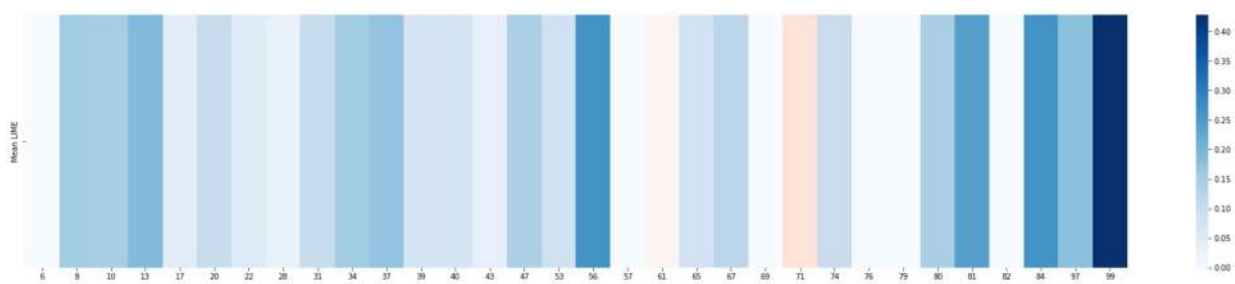
Generation of the heatmap.

In [672]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

In [673]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

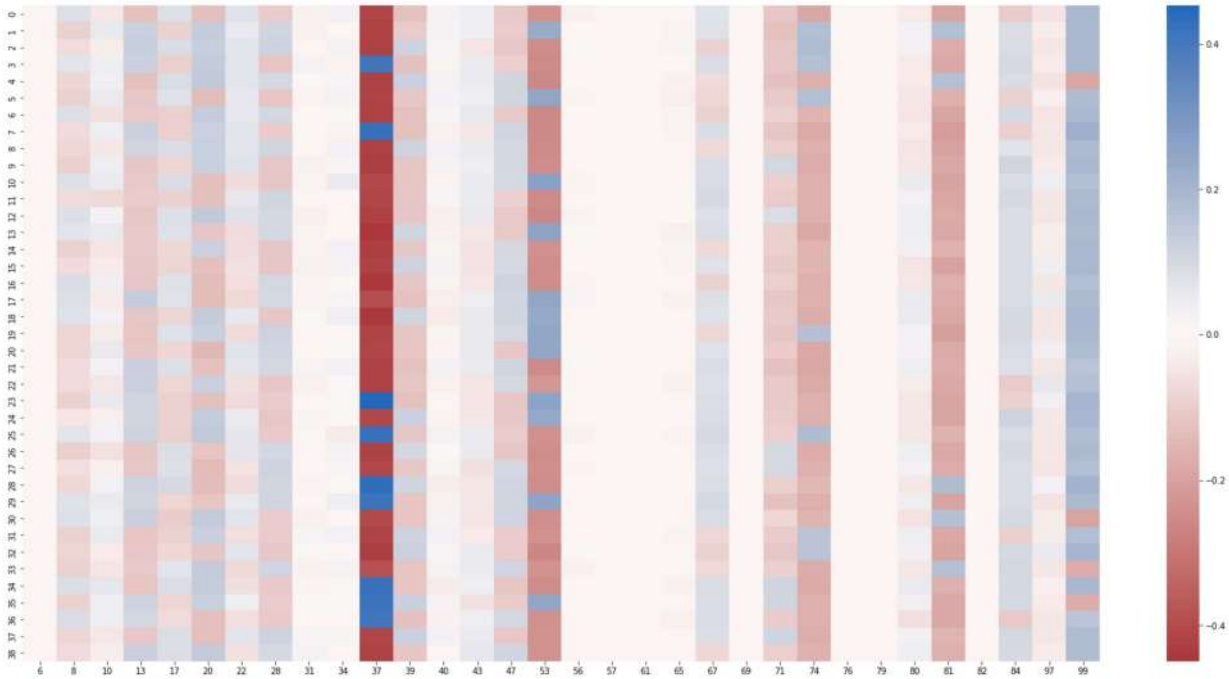


Individual explainability of the data belonging to the second class, in

this case the Medium.

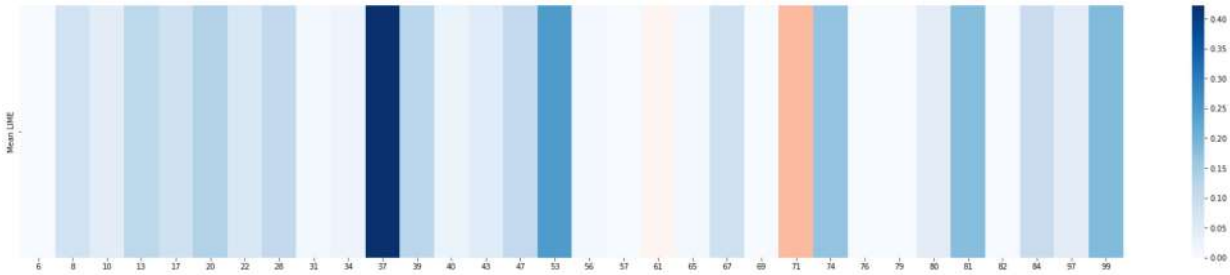
Generation of the heatmap.

In [674]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

In [675]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

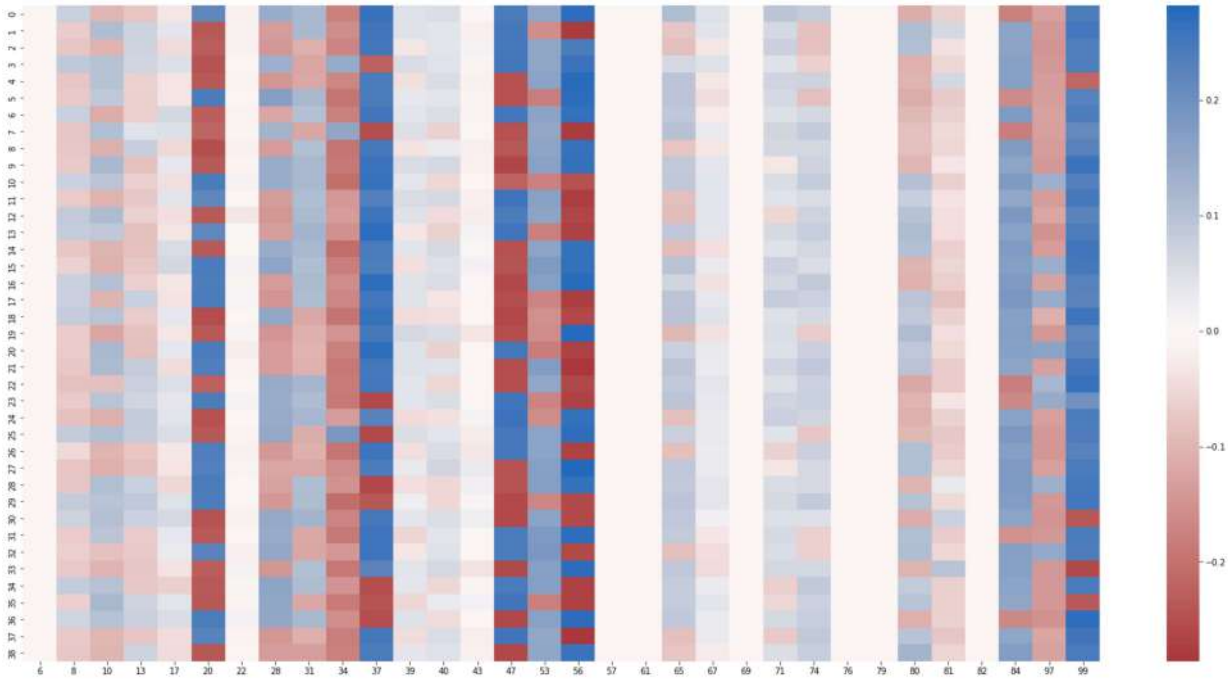


Individual explainability of data belonging to the third class, in this

case High.

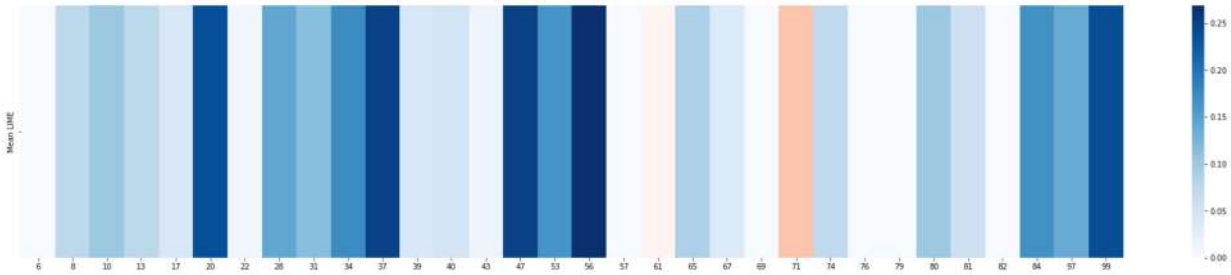
Generation of the heatmap.

In [676]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



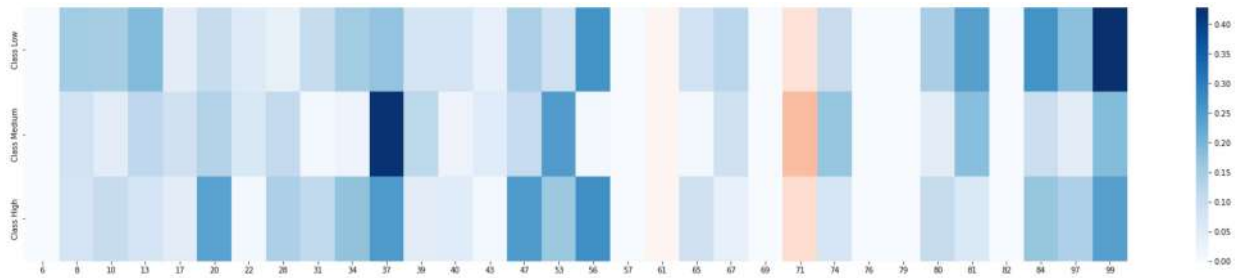
Fusion of the data used.

In [677]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [678]:
```
df_lime = pd.concat(map_class, axis=1)
df_lime.columns = name_class
plt.figure(figsize = (32,6))
sns.heatmap(df_lime.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_lime.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using ALE

It shows the main effect of each question compared to the probability that the model predicts in each class. This effect is shown for both responses. Since there are only 2 answers, the map is the same but with the colors reversed.

In [679]:
```
## Definition of explainability data.
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names=train_cols, target_names= target_r
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())

## Class index local variable.
iterable_class = -1
map_class = []
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

Obtaining the data for the generation of the explanations by ALE about all the classes.

In [680]:
```
## Local variables for ALE array data.
ale_list = [[] for _ in range(len(name_class))]

## Recovering the data needed for ALE.
for array in proba_exp_lr.ale_values:

    ## Recovering the data.
    for elements in range(0, len(name_class)):

        ## Saving data of all the classes.
        ale_list[elements].append(array[0][elements])
```

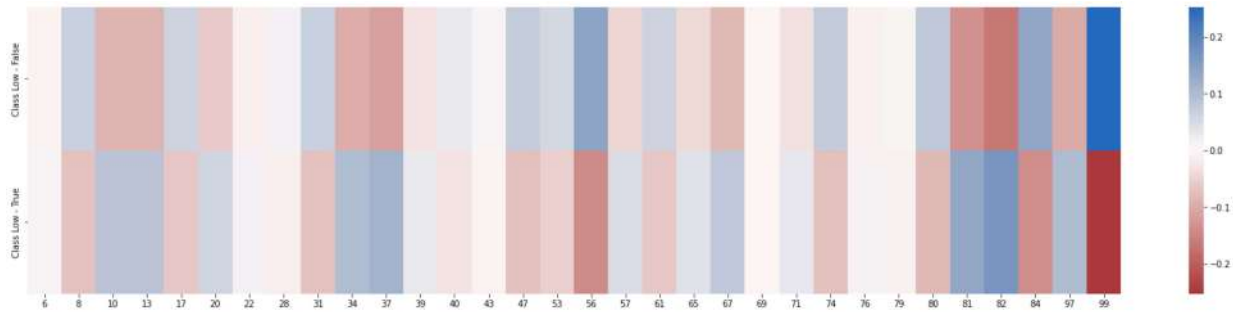
Data processing for the explainability of ALE.

In [681]:
```
## Data processing of the Low, Medium and High classes.
for elements in range(0, len(name_class)):
    ale_df = pd.DataFrame([ale_list [elements]])
    ale_df = pd.concat([ale_df.multiply(-1), ale_df])
    ale_df.index = [name_class [elements] + " - False", name_class [elements] + '
    ale_df.columns = column_Map
    map_class.append(ale_df)
```

Explainability of the data belonging to the first class, in this case the Low.

Generation of the heatmap.

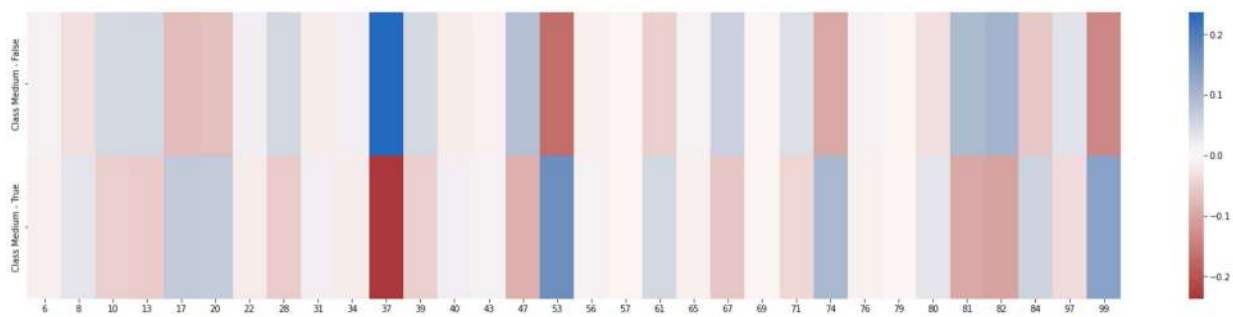
In [682]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the second class, in this case the Medium.

Generation of the heatmap.

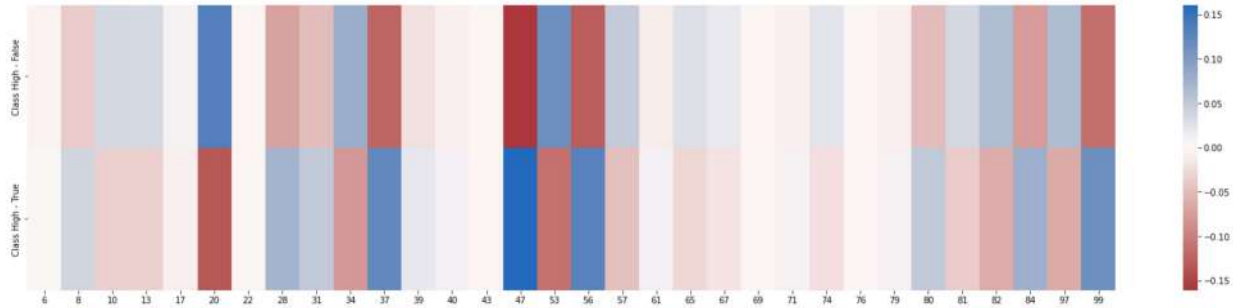
In [683]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the third class, in this case the High.

Generation of the heatmap.

In [684]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```

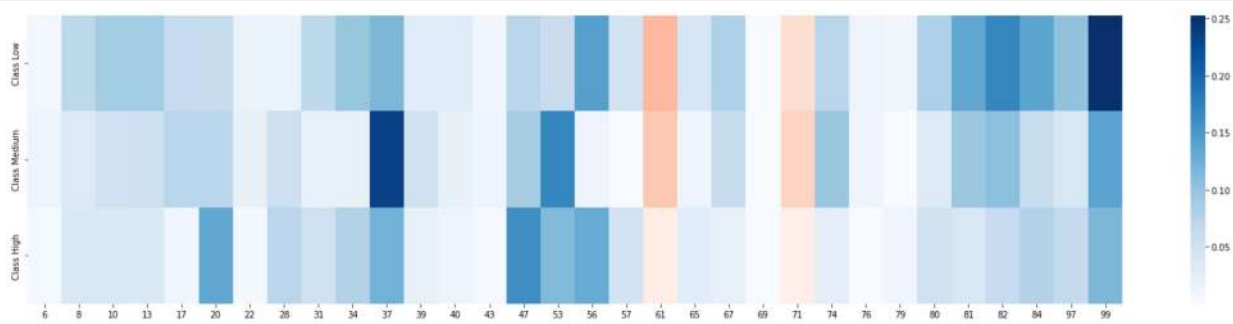


Fusion of the data used in all classes.

In [685]:
```
## Application of absolute value to data.
for index in range(0, len(map_class)):
    map_class [index] = map_class [index][0:1].abs()

## Union of the heatmaps of the Low, Medium and High class.
ale_df = pd.concat(map_class)
ale_df.index = name_class
ale_df = ale_df.transpose()
```

Generation of the heatmap.

In [686]:
```
plt.figure(figsize = (28,6))
sns.heatmap(ale_df.transpose(), cmap="Reds", cbar = False)
sns.heatmap(ale_df.transpose(), cmap="Blues", yticklabels = True, xticklabels = T
plt.show()
```



Comparing of Feature Variables Methods for Explainability

The global values of each of the methods used are taken, their maximums are obtained (the minimums are 0) and a percentage is calculated based on said maximum, which indicates how relevant the contribution is for each question.

In [687]:
```
## Class index local variable.
iterable_class = -1
```

Comparison of the explanation methods of the first class, in this case the Low.

The highest data of each method used is obtained.

In [688]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [689]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

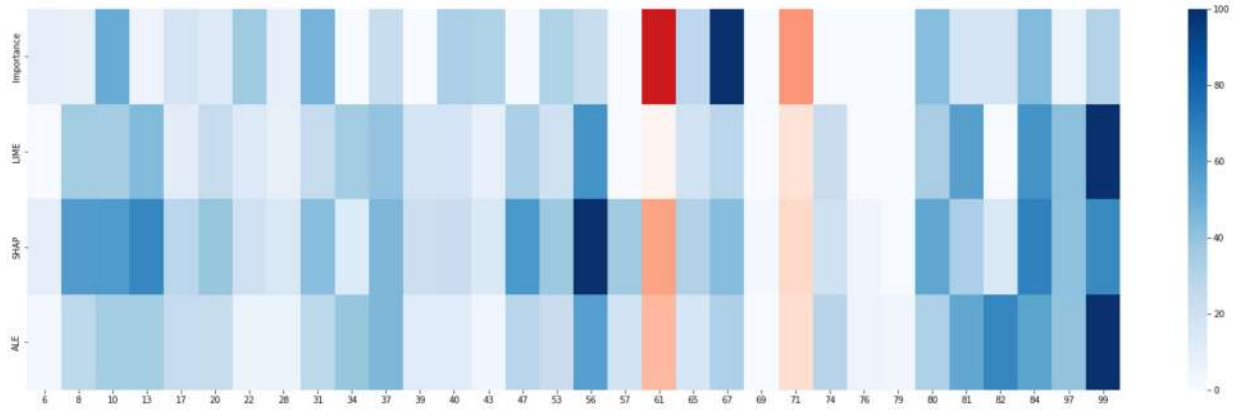
Out[689]:

	6	8	10	13	17	20	22	28
Importance	9.523810	7.936508	50.793651	4.761905	17.460317	12.698413	36.507937	9.523810
LIME	0.000000	35.867590	34.473109	44.315676	10.009546	24.507555	12.754943	7.659999
SHAP	9.462352	57.527717	58.375571	66.061813	27.931871	39.218922	19.402004	14.181967
ALE	3.340808	27.721919	34.818775	35.098006	25.059132	24.461344	6.570219	6.512514

4 rows × 32 columns

Generation of the heatmap.

In [690]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the second class, in this case the Medium.

The highest data of each method used is obtained.

In [691]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [692]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

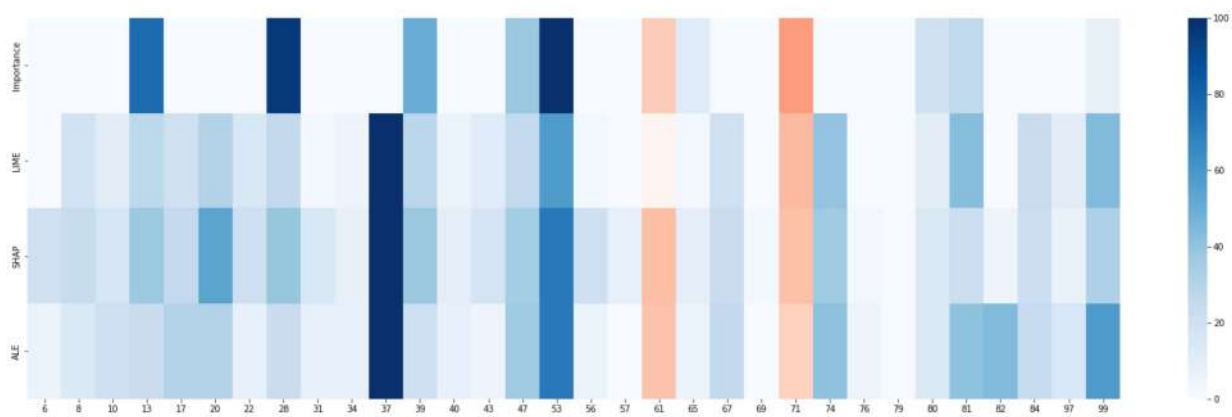
Out[692]:

	6	8	10	13	17	20	22	
Importance	0.000000	0.000000	0.000000	76.923077	0.000000	0.000000	1.494531e-13	96.153
LIME	0.000000	18.797524	10.523717	27.448686	20.169553	31.157393	1.536599e+01	25.932
SHAP	19.406370	24.446226	16.309957	37.551551	25.778763	54.602507	2.170840e+01	39.235
ALE	5.513492	13.493115	21.847890	22.977291	30.831171	30.238975	8.313148e+00	23.31

4 rows × 32 columns

Generation of the heatmap.

In [693]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the third class, in this case the High.

The highest data of each method used is obtained.

In [694]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [695]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

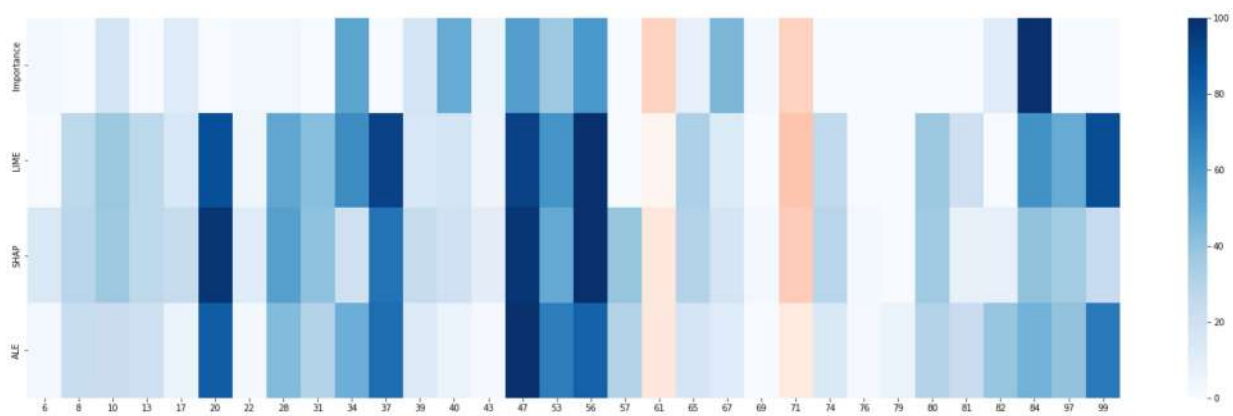
Out[695]:

	6	8	10	13	17	20	22	28
Importance	2.857143	0.000000	17.142857	0.000000	11.428571	0.000000	2.857143	2.857143
LIME	0.000000	27.547819	38.329701	27.399296	15.763120	87.965679	3.907720	52.941952
SHAP	13.475297	28.899977	37.339274	27.689202	24.283969	97.748376	11.351614	56.595410
ALE	2.868757	23.735375	22.589461	21.364365	6.006804	83.055623	1.913242	44.603634

4 rows × 32 columns

Generation of the heatmap.

In [696]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Safeguarding the items eliminated during this iteration.

In [697]:
```
## Removal of anomalous items selected by selection_mode.
for x in question_anomaly:

    ## Status change of the deleted items in the respective auxiliary dataset.
    ds.loc[x - 1,'Status'] = 'Delete'

## Save the changed statuses in the dataset.
ds.to_csv(path_dataset_qs, index = False)
```

Iteration 9

The first iteration of improvement of the model is carried out.

Preparing the data for the neural network

The data is prepared in order to be processed by the neural network.

In [698]:

```
## The iteration number is increased.
iteration += 1

## Loading data from the dataset.
df = pd.read_csv(path_dataset)
ds = pd.read_csv(path_dataset_qs)

## Elimination of unrelated items from the dataset.
if len(unrelated_questions) != 0:
    df = df.drop(df.columns[unrelated_questions[0]:unrelated_questions[1]], axis=

i = 0
if iteration == 1:
    ## Elimination of list of conflicting items declared in global variables.
    for x in list_contr:
        df.drop(df.columns[x - (i + 1)], axis = 1, inplace = True)
        i += 1
        ## Status change of the deleted question in the respective auxiliary data
        ds.loc[x - 1,'Status'] = 'Delete'

else:
    ## Elimination of the list of anomadic items defined in the improvement proce
    ds_delete = ds[ds.Status == "Delete"].reset_index(drop=True)
    for x in ds_delete['Question'].tolist():
        df.drop(df.columns[x - (i + 1)], axis = 1, inplace = True)
        i += 1

## Grouping of the target according to the defined dictionary.
if grouping_name != None and grouping_name != '':
    df[grouping_name] = df[grouping_name].map(class_dic)

## Assignment of local variables according to the data necessary for the neural r
train_cols = df.columns [0:-1]
label = df.columns [-1]
X = df [train_cols]
y = df [label]

## The prepared data of the dataset is printed.
df
```

Out[698]:

	6. I consider that my life is full of good things	8. I'm still as punctual as I have always been	10. I have found that I can spend a lot of time scrolling the screen \nof my cell phone without searching or stopping at anything in particular	13. I felt more encouraged to do my daily activities before	17. I feel guilty about the decisions that I have made	20. It is difficult for me to make decisions even if they are simple	22. It is not difficult for me to understand something the first time	28. The decisions I have made so far have been the right ones	e t p
0	1	1	0	0	0	1	1	0	
1	1	1	0	1	1	1	1	0	
2	1	0	1	0	0	0	1	1	
3	1	1	0	1	0	1	1	1	
4	1	1	1	0	1	1	0	0	
...	...	...	...	...	...	...	...	...	
99	1	1	1	1	0	1	0	1	
100	1	0	0	1	0	1	0	1	
101	1	0	1	1	0	0	0	0	
102	1	1	1	0	1	1	1	0	
103	1	0	1	0	0	1	0	1	

104 rows × 31 columns

Oversample application

An oversample process is applied to level the amount of existing data given by the grouping.

In [699]:
```
## The number of elements present in the grouped target is printed.
y.value_counts()
```

Out[699]:
```
0    39
1    35
2    30
Name: Target, dtype: int64
```

Oversample application.

In [700]:
```
## Application of the oversample to the prepared data.
X, y = oversample.fit_resample(X, y)
```

Printing of the data after the application of the oversample.

In [701]:
```
## The results obtained after applying the oversample are printed.
y.value_counts()
```

Out[701]:
```
0    39
1    39
2    39
Name: Target, dtype: int64
```

Preparation of the parameters of the neural network

We proceed to define the parameters necessary for the training of the neural network.

In [702]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
```

Neural network training

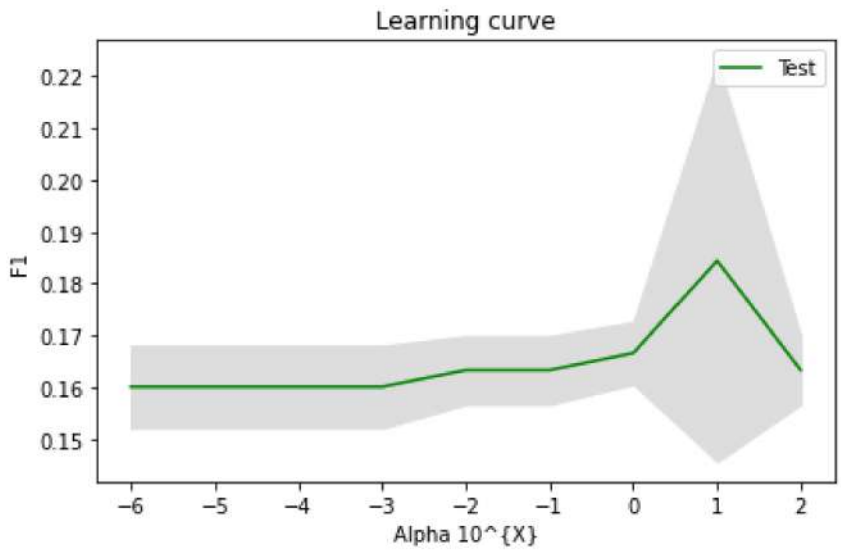
The alpha parameters and the initial learning rate are adjusted to obtain the best performance with the model using cross validation.

In [703]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_normal

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = 0.0001, max_iter =

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[703]:
```
([0.1632974910394265,
  0.18440860215053761,
  0.16659498207885304,
  0.16329749103942653,
  0.16329749103942653,
  0.16007168458781362,
  0.16007168458781362,
  0.16007168458781362,
  0.16007168458781362],
 [0.006619092595549127,
  0.03872527231332354,
  0.006020651927912844,
  0.006619092595549127,
  0.006619092595549127,
  0.007981170270805315,
  0.007981170270805315,
  0.007981170270805315,
  0.007981170270805315])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [704]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Alpha 10^{X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



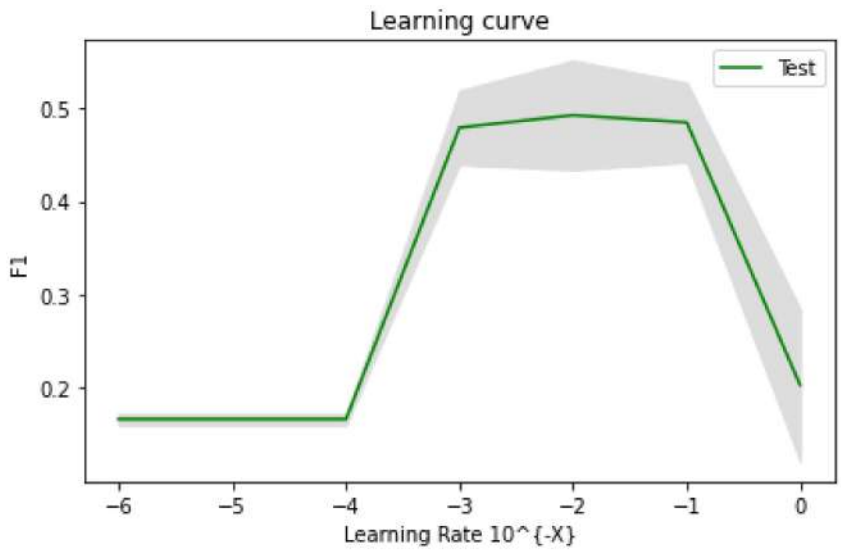
The training is performed again keeping the same parameters except for the alpha value that changes to 1.

In [705]:
```
## Local variables are defined for the storage of the scores obtained by the trai
cv_scores_mean = []
cv_scores_std = []
regul_param_range = regul_param_alpha

## Training and validation of different configurations.
for regul_param in regul_param_range:

    ## Increase the max_iter parameter until it converges.
    mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solv
             learning_rate = 'constant', learning_rate_init = regul_param, max_it

    scores = cross_val_score(mlp, X, y, cv = 5, scoring = 'f1_macro')

    cv_scores_mean.append(scores.mean())
    cv_scores_std.append(scores.std())

## The results obtained during the training are printed.
cv_scores_mean, cv_scores_std
```

Out[705]:
```
([0.20310940326513185,
  0.48459609591188546,
  0.49239412634149476,
  0.4790273761326393,
  0.16659498207885304,
  0.16659498207885304,
  0.16659498207885304],
 [0.08155999298520603,
  0.04259588791679032,
  0.05868329640963389,
  0.039578941293398305,
  0.006020651927912844,
  0.006020651927912844,
  0.006020651927912844])
```

We proceed to draw the learning curve graph according to the data obtained by the training.

In [706]:
```
## The average accuracy line is drawn on the test parts.
plt.figure()
plt.plot(np.log10(regul_param_range), cv_scores_mean, color = "g", label = "Test'

## The standard deviation band is drawn.
lower_limit = np.array(cv_scores_mean) - np.array(cv_scores_std)
upper_limit = np.array(cv_scores_mean) + np.array(cv_scores_std)
plt.fill_between(np.log10(regul_param_range), lower_limit, upper_limit, color = '

## Generate the graph.
plt.title("Learning curve")
plt.xlabel("Learning Rate 10^{-X}"), plt.ylabel("F1"), plt.legend(loc = "best")
plt.tight_layout()
plt.show()
```



Generation of the neural network model

The parameters of the final model are modified according to the data obtained in the training of the neural network.

In [707]:
```
## The value of the alpha parameter, learning_rate_init and random_state are char
## to the data in the global variables.
mlp = MLPClassifier(hidden_layer_sizes = (10,), activation = 'logistic', solver =
                    learning_rate = 'constant', learning_rate_init = lear
                    random_state = random_state_model)

## We print the data of the final generated model.
mlp.fit(X_train, y_train)
```

Out[707]:
```
MLPClassifier
MLPClassifier(activation='logistic', alpha=0.001, hidden_layer_sizes=(10,),
              learning_rate_init=0.1, max_iter=100000, random_state=42)
```

We safeguard the final generated model

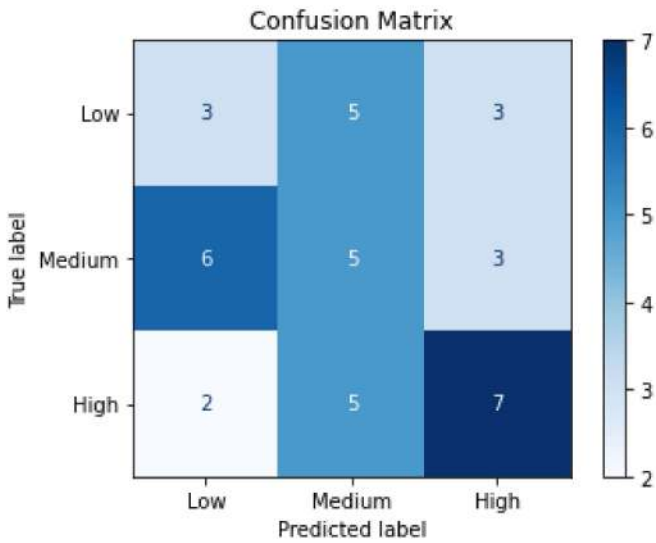
In [708]:
```
joblib.dump(mlp,"model_depression_i" + str(iteration) + ".pkl")
```

Out[708]:
```
['model_depression_i9.pkl']
```

Generation of the confusion matrix of the model

We obtain the statistics of the final model to generate its confusion matrix.

In [709]:
```
## The data for the generation of the confession matrix are defined.
confusion_matrix = ConfusionMatrixDisplay.from_estimator(mlp, X_test, y_test, dis
                                                         cmap = plt.cm.Blues)
confusion_matrix.ax_.set_title("Confusion Matrix")
plt.show()
```



We get the detailed statistics of the final model.

In [710]:
```
## The necessary detailed prediction is generated.
y_pred = mlp.predict(X_test)

## The detailed statistics of the model are printed.
print('Classification accuracy =',accuracy_score(y_test,y_pred) * 100,'%\n')
print(classification_report(y_test,y_pred))
```

```
Classification accuracy = 38.46153846153847 %

              precision    recall  f1-score   support

           0       0.27      0.27      0.27        11
           1       0.33      0.36      0.34        14
           2       0.54      0.50      0.52        14

    accuracy                           0.38        39
   macro avg       0.38      0.38      0.38        39
weighted avg       0.39      0.38      0.39        39
```

The data obtained is saved for later graphing.

In [711]:
```
accuracy = accuracy_score(y_test,y_pred) * 100
clf_rep = precision_recall_fscore_support(y_test,y_pred)

## The number of items used in the iteration is retrieved.
ds_delete = ds[ds.Status == "Delete"]
ds_delete.reset_index(inplace=True, drop=True)

## All the metrics of the confusion matrix are obtained.
metrics = [",".join(map(str, ds_delete['Question'].tolist())),
           accuracy]

for index in range(0, len(name_class)):
    metrics.append(clf_rep[0][index])
    metrics.append(clf_rep[1][index])
    metrics.append(clf_rep[2][index])
    metrics.append(clf_rep[3][index])

## The names of the columns of the dataset are defined.
columns = ['Question', 'Acurracy global']

for index in range(0, len(name_class)):
    columns.append('Precision_' + str(index))
    columns.append('Recall_' + str(index))
    columns.append('F1_score_' + str(index))
    columns.append('Support_' + str(index))

## A new row of the dataset is generated with all the data.
data_metrics = pd.DataFrame([metrics], columns = columns)

## The data is saved to the dataset.
data_metrics.to_csv(path_dataset_qd, mode = 'a', header = False, index = False)

## The data added to the dataset is printed.
data_metrics
```

Out[711]:

	Question	Acurracy global	Precision_0	Recall_0	F1_score_0	Supp
0	1,2,3,4,5,7,9,11,12,14,15,16,18,19,21,23,24,25...	38.461538	0.272727	0.272727	0.272727	

Improvement process of the neural network model.

The application of the explanation method by ALE allows to determine the behavior of the neural network and thereby improve it by discarding non-significant data for it.

Slope Analysis of ALE Plots

In [712]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

We proceed to obtain the graphs to obtain the slopes of the selected classes.

In [713]:
```
## The question metrics dataset is loaded.
dm = pd.DataFrame()

## The number of items used in the iteration is retrieved.
ds_item = ds[ds.Status == "In use"]
ds_item.reset_index(inplace=True, drop=True)

## We get the slopes from the ALE data.
for i in range(df.shape[1]-1):
    structure_data = {'Question': ds_item.loc[i, "Question"]}

    for index in selection_class:
        slope = proba_exp_lr.ale_values [i][1][index] - proba_exp_lr.ale_values [
        structure_data['Slope ' + str(target_names[index])] = slope
        structure_data['Threshold ' + str(target_names[index])] = 'NA'
        structure_data['Anomaly ' + str(target_names[index])] = 'NA'

    dm = dm.append(structure_data, ignore_index=True)

## The current rating data dataset is printed.
dm
```

```
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
```

```
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
The frame.append method is deprecated and will be removed from pandas in a futu
re version. Use pandas.concat instead.
```

Out[713]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	6	-0.118198	NA	NA	-0.009232	NA	NA
1	8	0.165548	NA	NA	-0.133810	NA	NA
2	10	-0.027447	NA	NA	0.020734	NA	NA
3	13	-0.139358	NA	NA	0.090596	NA	NA
4	17	-0.052101	NA	NA	0.042164	NA	NA
5	20	-0.040889	NA	NA	0.192623	NA	NA
6	22	-0.068262	NA	NA	-0.050902	NA	NA
7	28	0.039451	NA	NA	-0.213099	NA	NA
8	31	0.050852	NA	NA	-0.024333	NA	NA
9	34	-0.130562	NA	NA	0.205781	NA	NA
10	37	-0.192314	NA	NA	-0.279125	NA	NA
11	39	-0.042490	NA	NA	-0.001309	NA	NA
12	40	0.052107	NA	NA	-0.022936	NA	NA
13	43	-0.006443	NA	NA	0.012856	NA	NA
14	47	0.056155	NA	NA	-0.188298	NA	NA
15	53	0.088198	NA	NA	0.136520	NA	NA
16	56	0.162849	NA	NA	-0.162200	NA	NA
17	57	-0.128730	NA	NA	0.082395	NA	NA

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
18	65	0.093418	NA	NA	0.023820	NA	NA
19	67	-0.127503	NA	NA	0.031653	NA	NA
20	69	0.111729	NA	NA	0.061799	NA	NA
21	74	0.161320	NA	NA	0.058544	NA	NA
22	76	0.005788	NA	NA	-0.000561	NA	NA
23	79	0.087194	NA	NA	-0.054662	NA	NA
24	80	0.086616	NA	NA	-0.024717	NA	NA
25	81	-0.348966	NA	NA	0.101760	NA	NA
26	82	-0.171888	NA	NA	0.119936	NA	NA
27	84	0.241688	NA	NA	-0.070825	NA	NA
28	97	-0.217495	NA	NA	0.187682	NA	NA
29	99	0.445895	NA	NA	-0.221588	NA	NA

Slope analysis and threshold application

The slope obtained is analyzed and those that are above or below the positive and negative threshold are selected.

In [714]:
```
for i in range(df.shape[1]-1):
    for index in selection_class:

        ## The slopes are selected according to the values of the thresholds of t
        if dm.loc[i, "Slope " + str(target_names[index])] >= positive_threshold:
            dm.loc[i, "Threshold " + str(target_names[index])] = 1
        elif dm.loc[i, "Slope " + str(target_names[index])] <= negative_threshold
            dm.loc[i, "Threshold " + str(target_names[index])] = 0

## The current rating data dataset is printed.
dm
```

Out[714]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	6	-0.118198	0	NA	-0.009232	NA	NA
1	8	0.165548	1	NA	-0.133810	0	NA
2	10	-0.027447	0	NA	0.020734	1	NA
3	13	-0.139358	0	NA	0.090596	1	NA
4	17	-0.052101	0	NA	0.042164	1	NA
5	20	-0.040889	0	NA	0.192623	1	NA
6	22	-0.068262	0	NA	-0.050902	0	NA
7	28	0.039451	1	NA	-0.213099	0	NA
8	31	0.050852	1	NA	-0.024333	0	NA
9	34	-0.130562	0	NA	0.205781	1	NA
10	37	-0.192314	0	NA	-0.279125	0	NA
11	39	-0.042490	0	NA	-0.001309	NA	NA
12	40	0.052107	1	NA	-0.022936	0	NA
13	43	-0.006443	NA	NA	0.012856	1	NA
14	47	0.056155	1	NA	-0.188298	0	NA
15	53	0.088198	1	NA	0.136520	1	NA
16	56	0.162849	1	NA	-0.162200	0	NA
17	57	-0.128730	0	NA	0.082395	1	NA
18	65	0.093418	1	NA	0.023820	1	NA
19	67	-0.127503	0	NA	0.031653	1	NA
20	69	0.111729	1	NA	0.061799	1	NA
21	74	0.161320	1	NA	0.058544	1	NA
22	76	0.005788	NA	NA	-0.000561	NA	NA
23	79	0.087194	1	NA	-0.054662	0	NA
24	80	0.086616	1	NA	-0.024717	0	NA
25	81	-0.348966	0	NA	0.101760	1	NA
26	82	-0.171888	0	NA	0.119936	1	NA

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
27	84	0.241688	1	NA	-0.070825	0	NA
28	97	-0.217495	0	NA	0.187682	1	NA
29	99	0.445895	1	NA	-0.221588	0	NA

Analysis and determination of anomalous items

The anomalous items present in the model are determined based on their slope and expected response from the expert.

In [715]:

```
## The dataset of expected responses is loaded.
da = pd.read_csv(path_dataset_qa)

## Delete list of items deleted in this iteration.
i = 0
for x in ds_delete['Question']:
    da.drop(da.index[x - (i + 1)], axis = 0, inplace = True)
    i += 1

## Indexes are restored for later use.
da.reset_index(inplace=True, drop=True)

## The dataset of expected responses is printed.
da
```

Out[715]:

	Number	Question	RE
0	6	I consider that my life is full of good things	0
1	8	I'm still as punctual as I have always been	0
2	10	I have found that I can spend a lot of time sc...	1
3	13	I felt more encouraged to do my daily activiti...	1
4	17	I feel guilty about the decisions that I have ...	1
5	20	It is difficult for me to make decisions even ...	1
6	22	It is not difficult for me to understand somet...	0
7	28	The decisions I have made so far have been the...	0
8	31	It is easy for me to choose a photograph of my...	0
9	34	In recent months I have had discussions with m...	1
10	37	I'm not too worried about what might happen in...	0
11	39	Everything will be alright	0
12	40	I can easily blank my mind	0
13	43	Sometimes i feel trapped	1
14	47	I can clearly express to others how I feel	0
15	53	I'm bothered by people walking slowly in front...	1
16	56	I manage my schedule as I always have	0
17	57	It bothers me to feel that people on the stree...	1
18	65	I worry that people will not understand what I...	1
19	67	I get angry easily	1
20	69	It is not important for me to meet set dates	1
21	74	People have problems because of themselves	1
22	76	Laws are not as important as others think	1
23	79	It is easy for me to work in a team	0
24	80	It is important to help people when they need it	0
25	81	I have punched someone or thought of doing it	1

	Number	Question	RE
26	82	If it was necessary I would pretend to be some...	1
27	84	After an argument I usually go over what happe...	0
28	97	People who commit crimes have their reasons fo...	1
29	99	It is important to respect turns	0

We proceed to identify the anomadic items present in the model present in this iteration.

In [716]:

```
for i in range(df.shape[1]-1):

    ## Local variable to indicate the index of the selected classes.
    i_index = 0

    for index in selection_class:

        ## Determine if it fails based on the expected response parameters of the
        if dm.loc[i, "Threshold "  + str(target_names[index])] == 1 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 1 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 0 and da.loc[i,
            dm.loc[i, "Anomaly " + str(target_names[index])] = expected_response_
        if dm.loc[i, "Threshold " + str(target_names[index])] == 0 and da.loc[i,
            dm.loc[i, "Anomaly "  + str(target_names[index])] = expected_response

        ## Increase of the index of the selected classes.
        i_index += 1

## The final grade data dataset is printed.
dm
```

Out[716]:

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
0	6	-0.118198	0	1	-0.009232	NA	NA
1	8	0.165548	1	0	-0.133810	0	0
2	10	-0.027447	0	0	0.020734	1	0
3	13	-0.139358	0	0	0.090596	1	0
4	17	-0.052101	0	0	0.042164	1	0
5	20	-0.040889	0	0	0.192623	1	0
6	22	-0.068262	0	1	-0.050902	0	0
7	28	0.039451	1	0	-0.213099	0	0
8	31	0.050852	1	0	-0.024333	0	0
9	34	-0.130562	0	0	0.205781	1	0
10	37	-0.192314	0	1	-0.279125	0	0
11	39	-0.042490	0	1	-0.001309	NA	NA
12	40	0.052107	1	0	-0.022936	0	0
13	43	-0.006443	NA	NA	0.012856	1	0
14	47	0.056155	1	0	-0.188298	0	0
15	53	0.088198	1	1	0.136520	1	0
16	56	0.162849	1	0	-0.162200	0	0
17	57	-0.128730	0	0	0.082395	1	0
18	65	0.093418	1	1	0.023820	1	0
19	67	-0.127503	0	0	0.031653	1	0

	Question	Slope Low	Threshold Low	Anomaly Low	Slope High	Threshold High	Anomaly High
20	69	0.111729	1	1	0.061799	1	0
21	74	0.161320	1	1	0.058544	1	0
22	76	0.005788	NA	NA	-0.000561	NA	NA
23	79	0.087194	1	0	-0.054662	0	0
24	80	0.086616	1	0	-0.024717	0	0
25	81	-0.348966	0	0	0.101760	1	0
26	82	-0.171888	0	0	0.119936	1	0
27	84	0.241688	1	0	-0.070825	0	0
28	97	-0.217495	0	0	0.187682	1	0
29	99	0.445895	1	0	-0.221588	0	0

Removing identified anomadic items from the model

We proceed to eliminate the identified anomadic items so as not to use them in the next iteration.

In [717]:
```
## Local variable for the elimination of the anomaly items.
instructions = ''
init = False
question_anomaly = []

if selection_mode == 1:

    ## Items that fail in the defined class are retrieved.
    dm_delete = dm[dm['Anomaly ' + str(target_names[select_class_mode])] == 1]
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 2:

    ## Items that fail in classes defined with the or condition type are retrieve
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 1"
            init = True
        else:
            instructions += " or `Anomaly " + str(target_names[index]) + "` == 1'

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 3:

    ## Items that fail in classes defined with the and condition type are retriev
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 1"
            init = True
        else:
            instructions += " and `Anomaly " + str(target_names[index]) + "` == 1

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

if selection_mode == 4:

    ## Out-of-threshold items are retrieved in classes defined with the conditior
    for index in selection_class:
        if init == False:
            instructions += "`Anomaly " + str(target_names[index]) + "` == 'NA'"
            init = True
        else:
            instructions += " and `Anomaly " + str(target_names[index]) + "` == '

    dm_delete = dm.query(instructions)
    dm_delete.reset_index(inplace=True, drop=True)
    question_anomaly = dm_delete['Question'].tolist()

## The selected blank items are printed according to the selection_mode.
question_anomaly
```

Out[717]: []

Model explainability process

The data used for the test of the neural network model is shown.

In [718]:
```
X_test
```

Out[718]:

	6. I consider that my life is full of good things	8. I'm still as punctual as I have always been	10. I have found that I can spend a lot of time scrolling the screen \nof my cell phone without searching or stopping at anything in particular	13. I felt more encouraged to do my daily activities before	17. I feel guilty about the decisions that I have made	20. It is difficult for me to make decisions even if they are simple	22. It is not difficult for me to understand something the first time	28. The decisions I have made so far have been the right ones	easy to cl phot of to or
69	1	0	0	0	0	1	1	0	
46	1	1	1	1	1	0	1	1	
58	1	1	0	1	0	0	1	1	
114	1	1	1	0	1	0	1	0	
73	1	1	1	0	0	0	1	1	
98	0	1	1	0	0	1	1	0	
31	1	0	0	0	1	0	1	1	
53	1	1	1	1	1	0	1	0	
65	1	1	0	1	0	0	1	1	
96	1	1	1	0	1	0	1	0	
95	1	0	1	0	0	1	0	0	
97	1	1	0	0	1	1	1	1	
2	1	0	1	0	0	0	1	1	
62	1	0	1	0	0	1	0	1	
110	0	1	0	0	0	0	0	0	
55	1	1	0	0	1	1	0	0	
103	1	0	1	0	0	1	0	1	
100	1	0	0	1	0	1	0	1	
66	0	0	1	0	1	0	1	0	
44	1	1	0	0	0	0	0	1	
77	1	1	1	0	1	1	1	1	
17	1	1	1	1	0	1	1	1	
81	1	1	0	1	1	0	0	0	
74	0	1	1	1	1	1	0	0	

56	1	1	0	1	1	0	1	0
94	0	0	1	1	1	0	1	0
35	1	1	0	0	0	1	1	1
38	1	1	0	0	0	1	0	1
93	1	1	1	1	0	1	0	1
48	1	0	1	1	1	1	1	1
33	1	0	1	1	1	0	1	0
59	1	1	1	0	1	0	0	0
92	0	1	0	0	1	1	1	0
42	1	1	0	0	0	0	0	1
10	1	0	1	0	0	0	0	0
85	1	1	1	1	1	0	1	0
32	1	0	1	1	1	1	0	0
78	1	1	0	0	0	1	1	1
19	0	1	0	1	0	0	0	1

39 rows × 30 columns

Explanation of the model using ALE

In analysis of the model through the results obtained by ALE, it is possible to identify the behavior of the neural network.

In [719]:
```
## The necessary parameters for the use of ALE are established according to the m
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names = train_cols, target_names = targe
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())
```

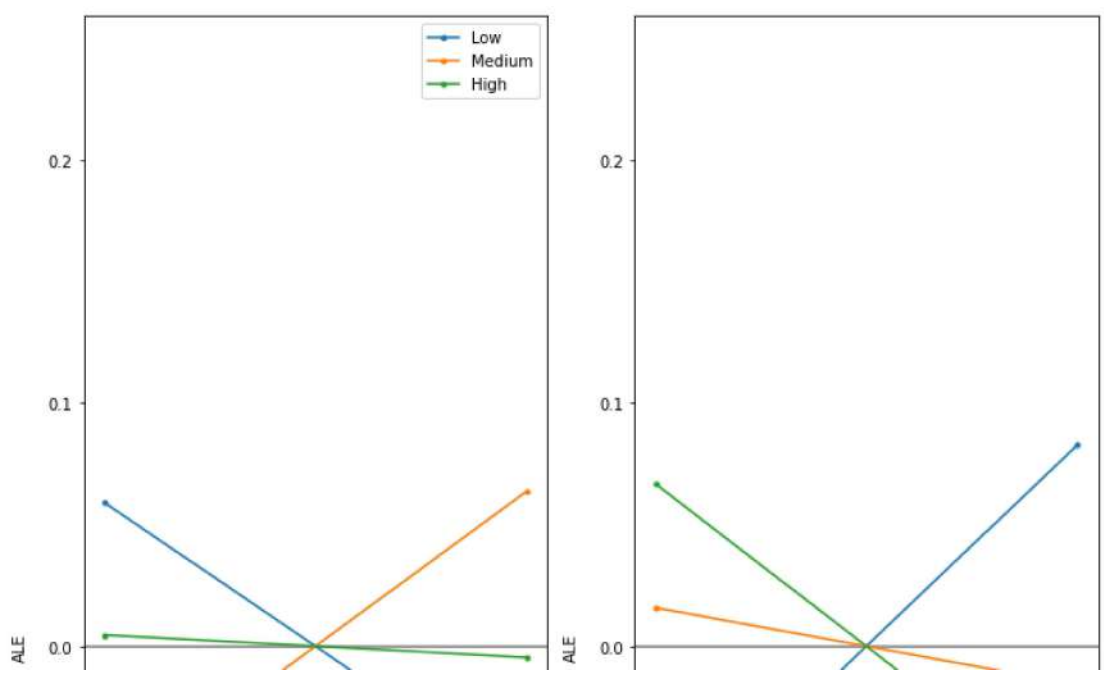
```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

The graphs of all the data present in the dataset used are shown.

In [720]:
```
plot_ale(proba_exp_lr, n_cols=2, features=list(range(df.shape[1]-1)),fig_kw={'fig
```



Explanation of the model using LIME

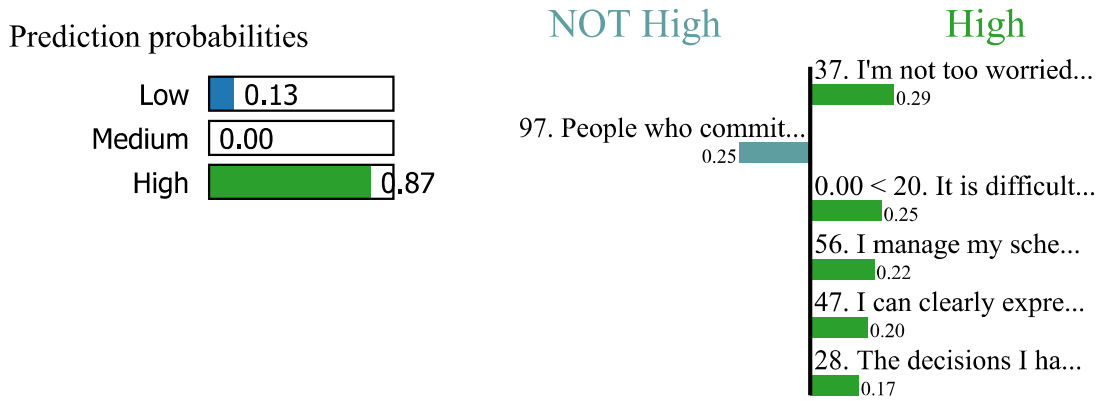
We proceed to explain the improved model using LIME for the Low, Medium and High classes.

In [721]:
```
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorica
                                                   feature_names=train_cols, clas
                                                   discretize_continuous=True)
```

The first individual present in the test data is explained.

In [722]:
```
exp = explainer.explain_instance(X_test.to_numpy()[0], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

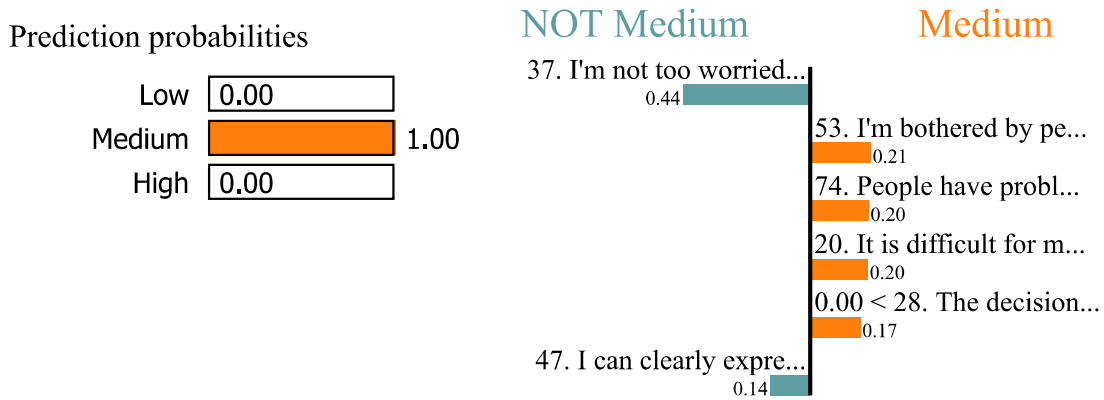


Feature	Value
37. I'm not too worried about what might happen in a few weeks	0.00
97. People who commit crimes have their reasons for doing it	0.00
20. It is difficult for me to make decisions even if they are simple	1.00
56. I manage my schedule as I always have	0.00
47. I can clearly express to others how I feel	0.00
28. The decisions I have made so far have been the right ones	0.00

The second individual present in the test data is explained.

In [723]:
```
exp = explainer.explain_instance(X_test.to_numpy()[1], mlp.predict_proba, num_fea
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

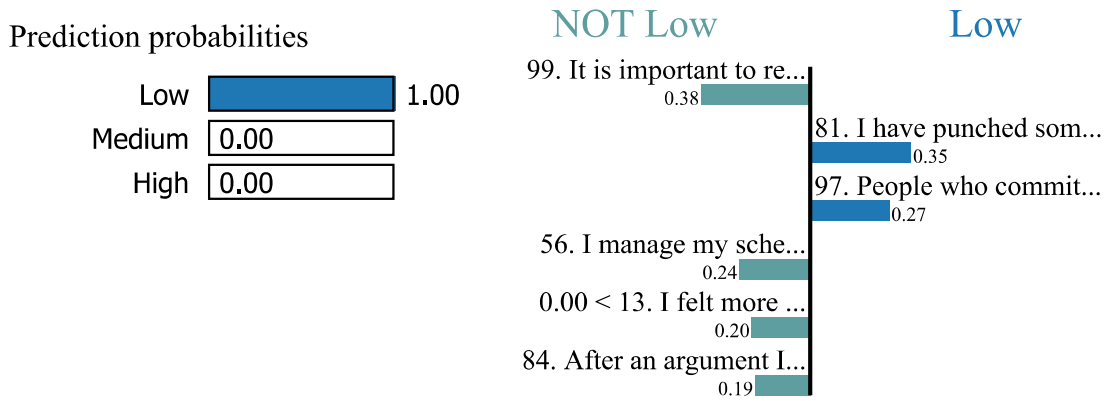


Feature	Value
37. I'm not too worried about what might happen in a few weeks	0.00
53. I'm bothered by people walking slowly in front of me	0.00
74. People have problems because of themselves	0.00
20. It is difficult for me to make decisions even if they are simple	0.00
28. The decisions I have made so far have been the right ones	1.00
47. I can clearly express to others how I feel	0.00

The last individual present in the test data is explained.

In [724]:
```
exp = explainer.explain_instance(X_test.to_numpy()[-1], mlp.predict_proba, num_fe
exp.show_in_notebook(show_table=True, show_all=False)
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```



Feature	Value
99. It is important to respect turns	0.00
81. I have punched someone or thought of doing it	0.00
97. People who commit crimes have their reasons for doing it	0.00
56. I manage my schedule as I always have	0.00
13. I felt more encouraged to do my daily activities before	1.00
84. After an argument I usually go over what happened in my head	0.00

Explanation of the model using SHAP

SHAP explanatory values are displayed according to the class to which the selected individual belongs according to the prediction of the model.

The first individual present in the test data is explained.

In [725]:
```
## Individual Selection.
individual = X_test.iloc[[0]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The second individual present in the data test is explained.

In [726]:

```
## Individual Selection.
individual = X_test.iloc[[1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

The last individual present in the data test is explained.

In [727]:
```
## Individual Selection.
individual = X_test.iloc[[-1]]

## Individual explanation by SHAP.
explainer = shap.KernelExplainer(mlp.predict_proba, X_train,feature_names = trair
shap_values = explainer.shap_values(individual)
clase = mlp.predict(individual)[0]
shap.summary_plot(shap_values[clase], individual, feature_names = train_cols, plc
```

```
X does not have valid feature names, but MLPClassifier was fitted with featur
e names

  0%|          | 0/1 [00:00<?, ?it/s]
```

Explanation using heatmaps

The importance of the data present in the neural network is shown from explanation methods.

In [728]:

```
## Items currently in use are retrieved.
ds_item = ds[ds.Status == "In use"].reset_index(drop=True)

## The data is prepared for the generation of heatmaps.
column_Map = ds_item['Question'].tolist()
content_Map = [False for x in range(1, df.shape[1])]

## The next items to be deleted will be marked.
for index in range(0, len(content_Map)):

    if iteration == 0:
        for delete in list_contr:
            if column_Map[index] == delete:
                content_Map [index] = True
    else:
        for delete in question_anomaly:
            if column_Map[index] == delete:
                content_Map [index] = True

## The dataframe is created with the prepared data.
df_mask = pd.DataFrame([content_Map], columns = column_Map)
df_mask_imp = pd.concat([df_mask, df_mask, df_mask], ignore_index = True, axis =
df_mask_all = pd.concat([df_mask, df_mask, df_mask, df_mask], ignore_index = True

df_mask
```

Out[728]:

	6	8	10	13	17	20	22	28	31	34	...	69	74	76	79
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False

1 rows × 30 columns

Identification of feature variables

Identification of feature variables from the randomness of the answers as items of the data.

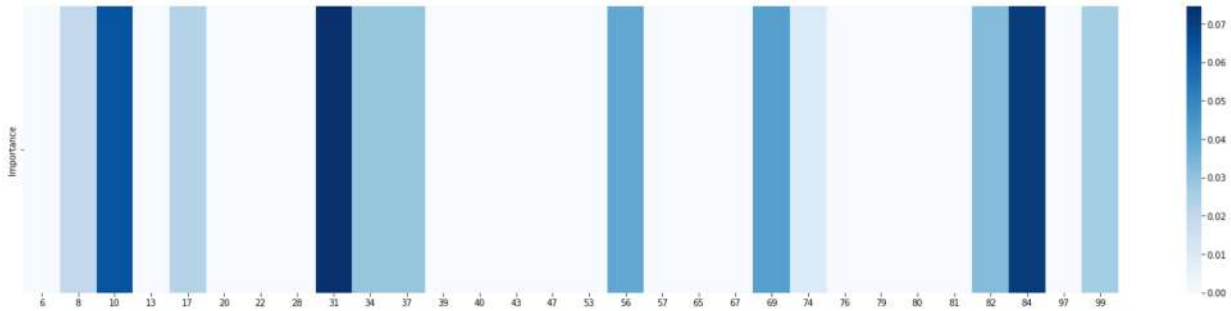
In [729]:
```
## Loading the data for explainability.
explainer = ClassifierExplainer(mlp, X_test, y_test)

## Class index local variable.
iterable_class = -1
map_class = []
```

```
WARNING: Parameter shap='guess', but failed to guess the type of shap explainer
to use for MLPClassifier. Defaulting to the model agnostic shap.KernelExplainer
(shap='kernel'). However this will be slow, so if your model is compatible with
e.g. shap.TreeExplainer or shap.LinearExplainer then pass shap='tree' or shap
='linear'!
WARNING: For shap='kernel', shap interaction values can unfortunately not be ca
lculated!
Note: shap values for shap='kernel' normally get calculated against X_backgroun
d, but paramater X_background=None, so setting X_background=shap.sample(X, 5
0)...
Generating self.shap_explainer = shap.KernelExplainer(model, X, link='identit
y')
```

Explainability of the data belonging to the first class, in this case the Low.

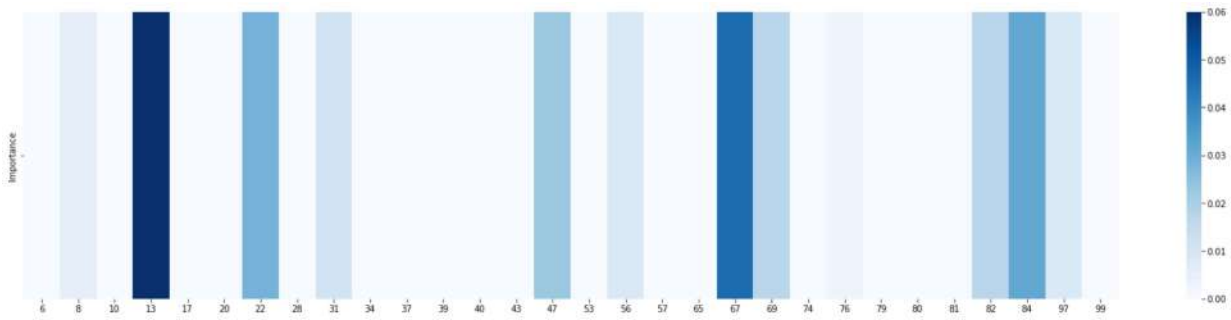
In [730]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

```
Calculating permutation importances (if slow, try setting n_jobs parameter)...
```



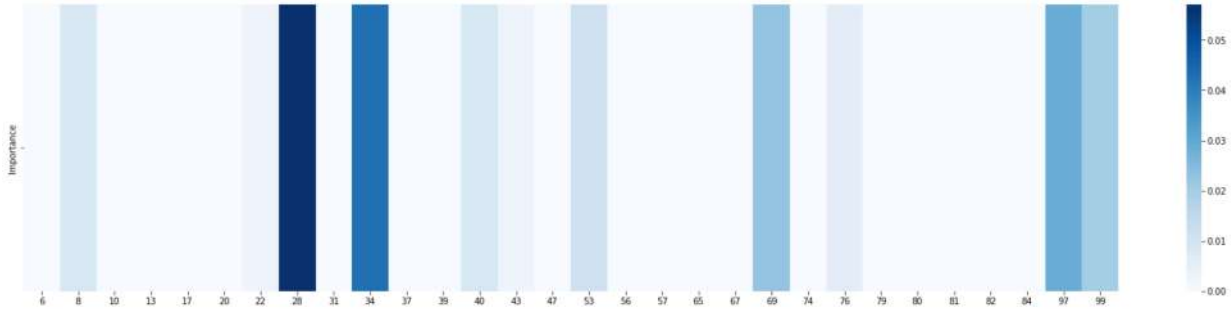
Explainability of the data belonging to the second class, in this case the Medium.

In [731]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



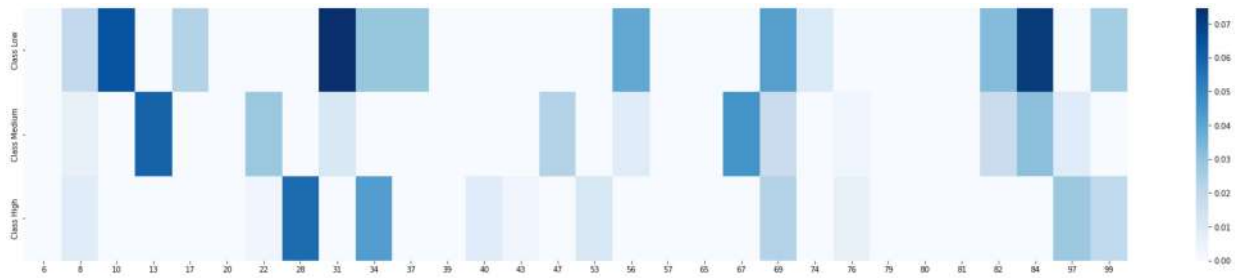
Explainability of the data belonging to the third class, in this case the High.

In [732]:
```
## Definition of explainability data.
iterable_class += 1
df_importance = explainer.permutation_importances(iterable_class).sort_index()
df_importance.index = column_Map
df_importance.drop("Score",inplace=True, axis=1)
df_importance.drop("Feature",inplace=True, axis=1)
df_importance.loc[df_importance['Importance'] < 0, 'Importance'] = 0
map_class.append(df_importance)

## Generation of the heatmap.
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [733]:
```
## Fusion of the previous heatmaps.
df_importance = pd.concat(map_class, axis=1)
df_importance.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_importance.transpose(), cmap = "Reds", cbar = False)
sns.heatmap(df_importance.transpose(), cmap = "Blues",yticklabels = True, xtickla
plt.show()
```



Identification of feature variables using SHAP

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of SHAP.

In [734]:
```
## Class index local variable.
iterable_class = -1
map_class = []
```

Individual explainability of the data belonging to the first class, in this case the Low.

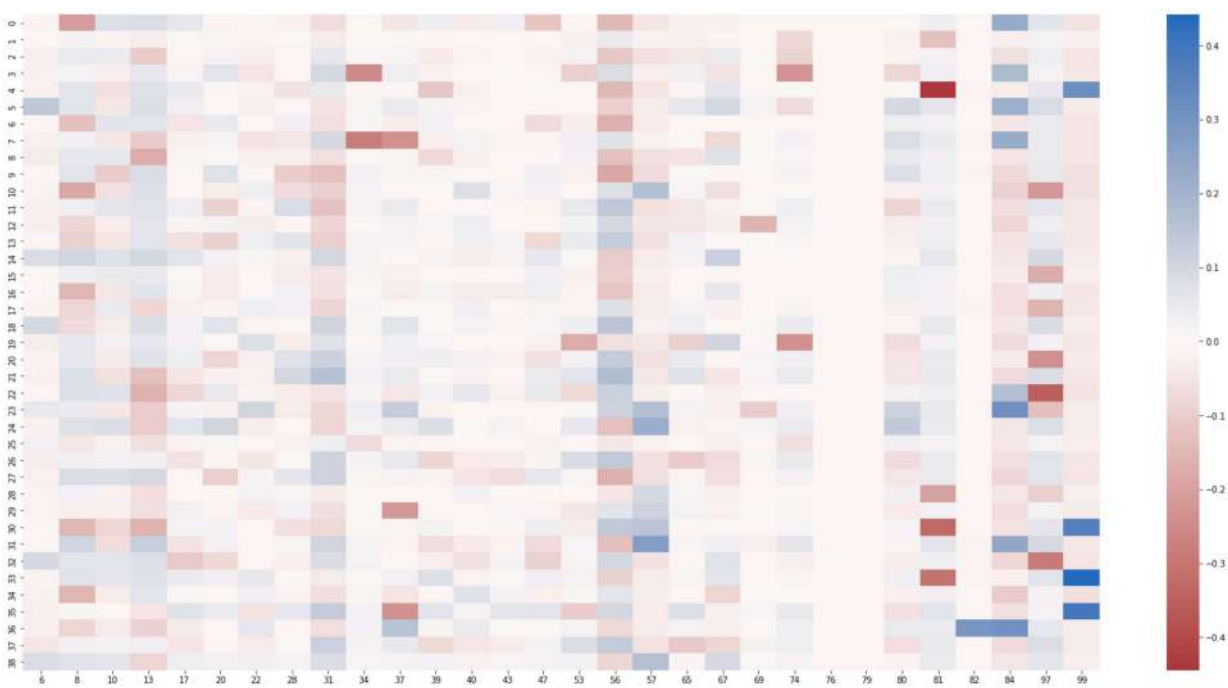
Definition of explainability data.

In [735]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

```
Calculating shap values...

  0%|          | 0/39 [00:00<?, ?it/s]
```

Generation of the heatmap.

In [736]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

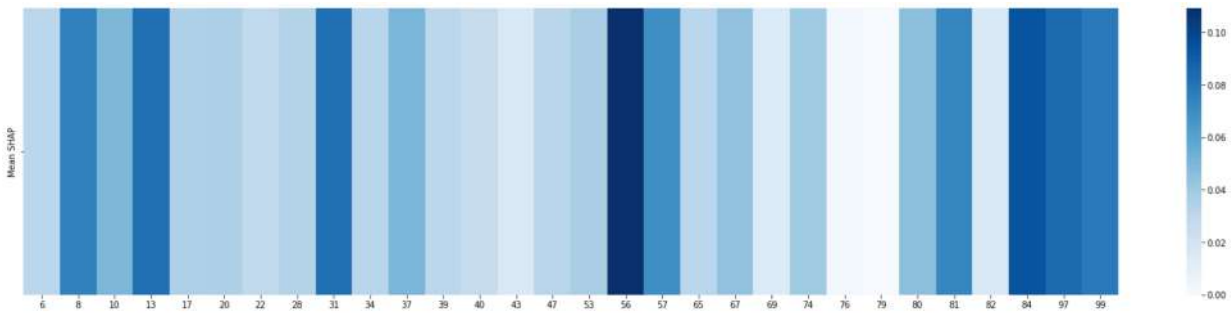


Fusion of the data used.

In [737]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [738]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



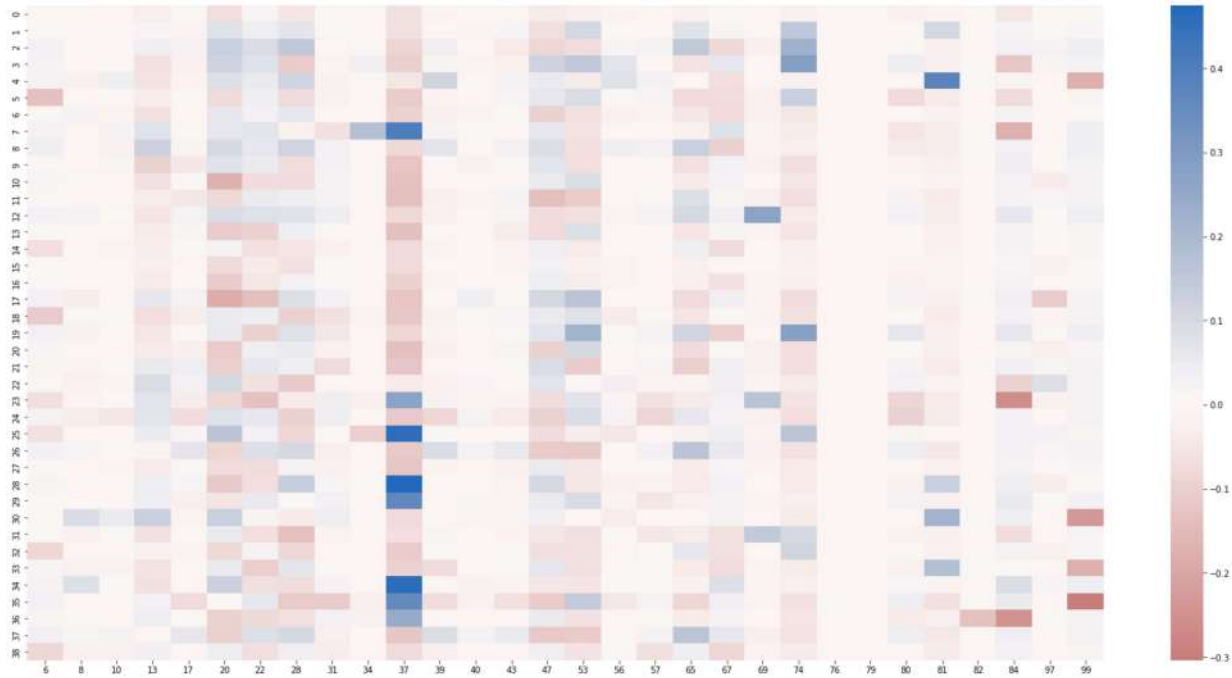
Individual explainability of the data belonging to the second class, in this case the Medium.

Definition of explainability data.

In [739]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [740]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

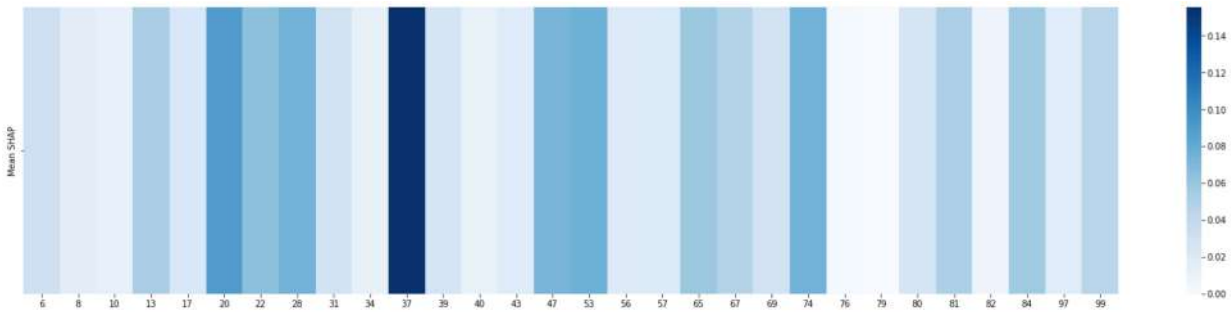


Fusion of the data used.

In [741]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

Generation of the heatmap.

In [742]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



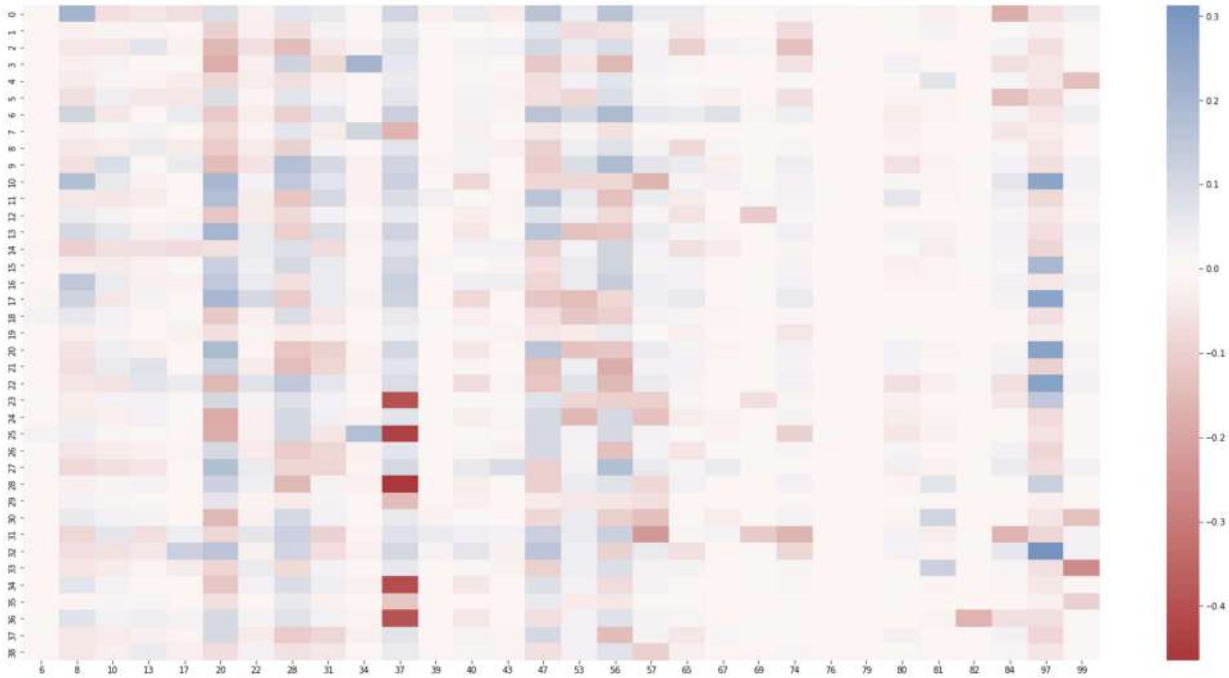
Individual explainability of data belonging to the third class, in this case High.

Definition of explainability data.

In [743]:
```
iterable_class += 1
df_shap = explainer.get_shap_values_df(iterable_class)
df_shap.columns = column_Map
```

Generation of the heatmap.

In [744]:
```
plt.figure(figsize = (28,14))
sns.heatmap(df_shap, cmap="vlag_r", yticklabels=True, xticklabels=True, center=0)
plt.show()
```

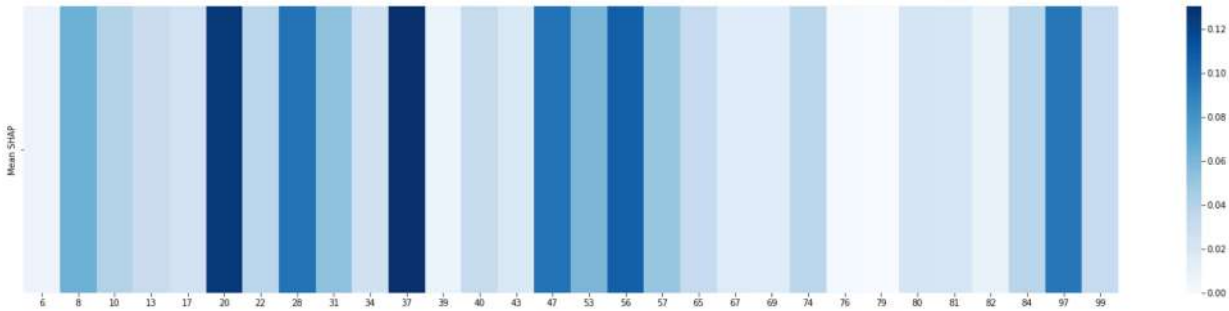


Fusion of the data used.

In [745]:
```
df_shap_mean = df_shap.abs().mean(axis=0).to_frame()
df_shap_mean.columns= ["Mean SHAP"]
map_class.append(df_shap_mean)
```

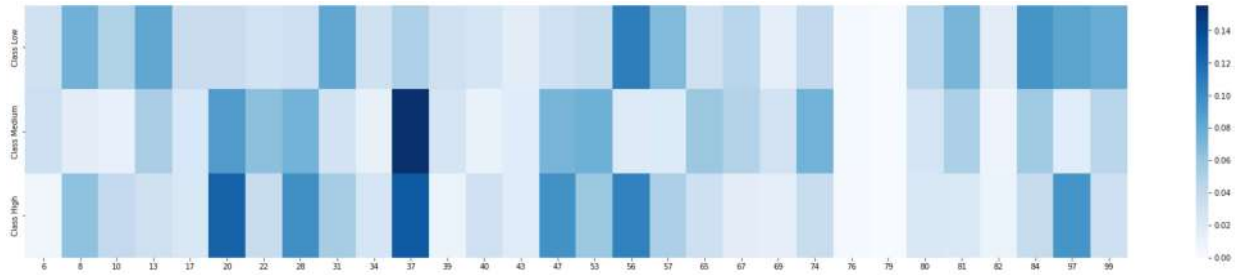
Generation of the heatmap.

In [746]:
```
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [747]:
```
## Fusion of the previous heatmaps.
df_shap = pd.concat(map_class, axis=1)
df_shap.columns = name_class

## Generation of the heatmap.
plt.figure(figsize = (32,6))
sns.heatmap(df_shap.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_shap.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using LIME

Contribution in the final probability of each question for specific instances at the local level. For each class, a map is first shown with all the individuals and the contribution of each question, and then a map made with the mean of these values in absolute value. This seeks to globalize the scope of LIME.

In [748]:
```
## Data preparation for explainability.
lime_exp = lime.lime_tabular.LimeTabularExplainer(X_train.to_numpy(), categorical
                                                  feature_names=train_cols, class
                                                  discretize_continuous=True)

## Class index local variable.
iterable_class = -1
map_class = []
```

Obtaining the data for the generation of the explanations by LIME about the Low, Medium and High classes.

In [749]:
```
## Local variables for LIME array data.
exp_matrix = [[] for _ in range(len(name_class))]

## Recovering the data needed for LIME.
for x in X_test.to_numpy():

    ## Auxiliary temporary variables.
    exp_list = [[] for _ in range(len(name_class))]

    ## Loading the data for explainability.
    exp = lime_exp.explain_instance(x, mlp.predict_proba, num_features = df.shape

    ## Recovering the data.
    for elements in range(0, len(name_class)):
        temp = exp.as_map()[elements]
        temp.sort(key=itemgetter(0))

        ## Saving data of the Low, Medium and High classes.
        for tup in temp:
            exp_list[elements].append(tup[1])

        exp_matrix[elements].append(exp_list[elements])
```

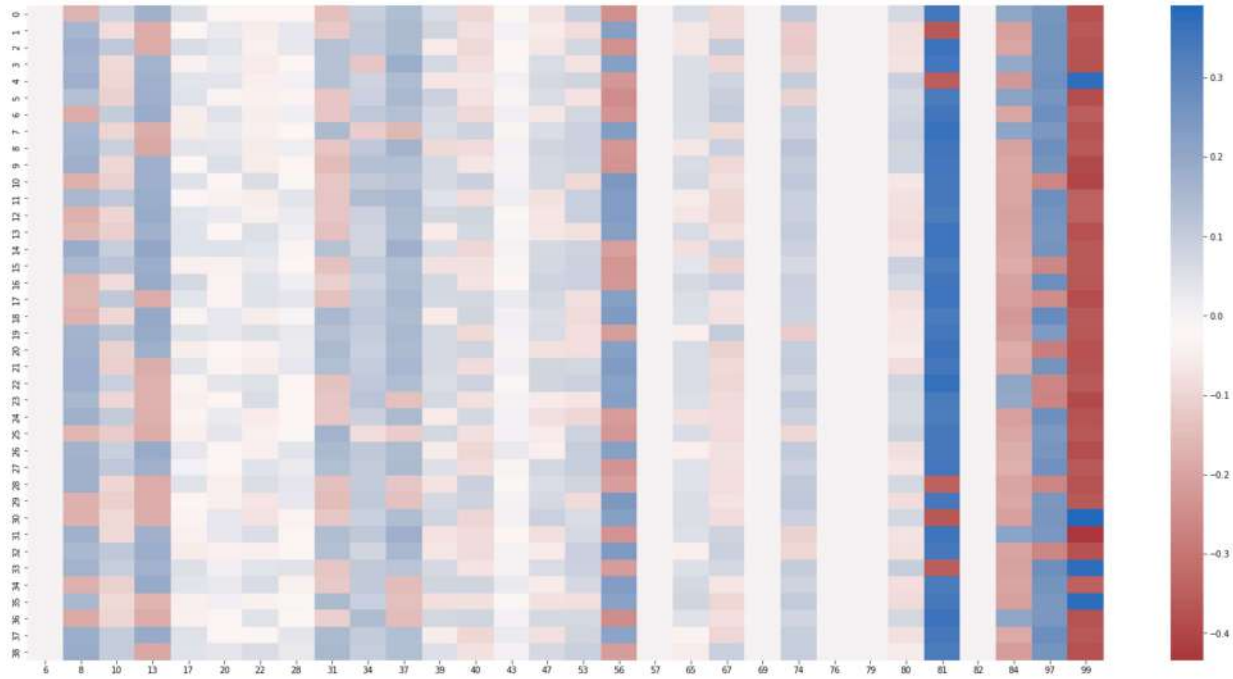
```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

Individual explainability of the data belonging to the first class, in this case the Low.

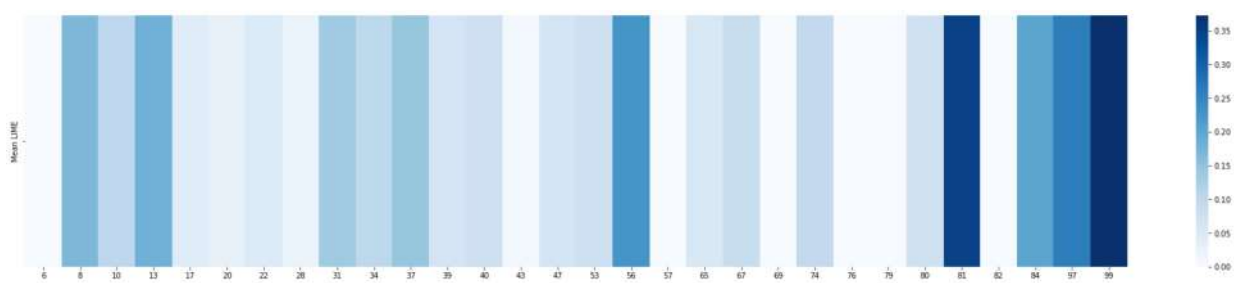
Generation of the heatmap.

In [750]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

In [751]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

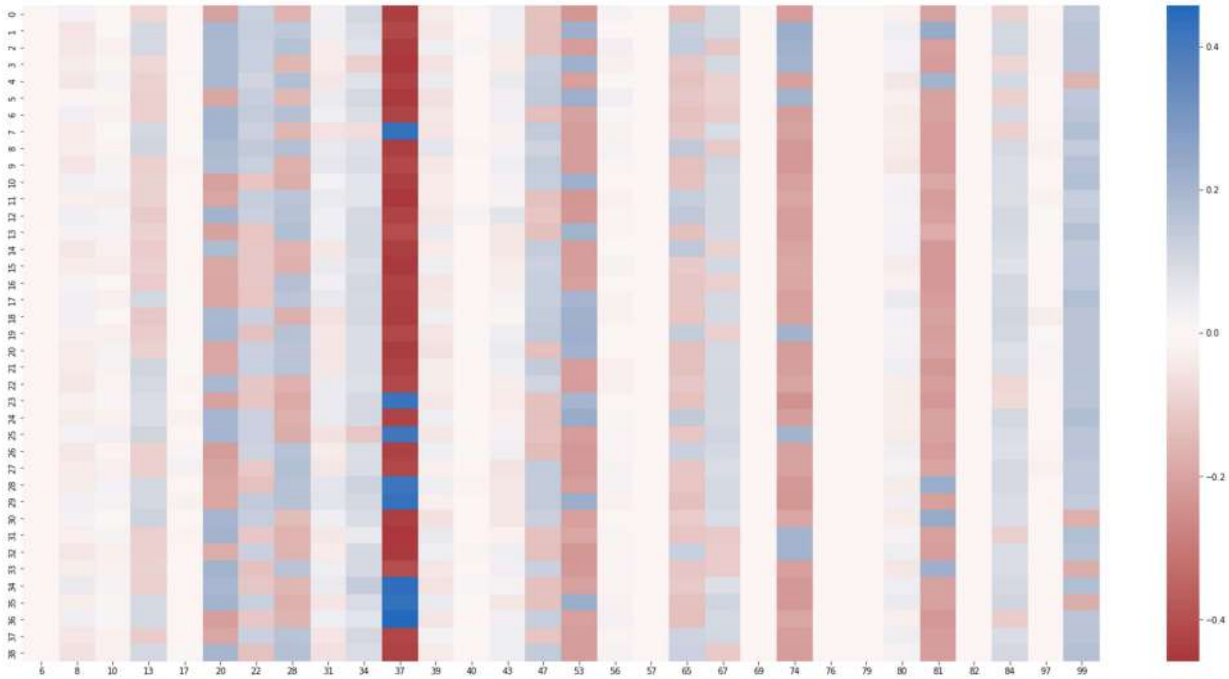


Individual explainability of the data belonging to the second class, in

this case the Medium.

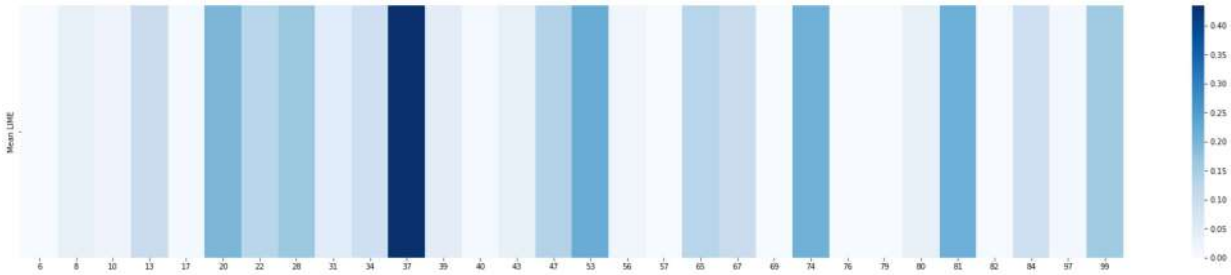
Generation of the heatmap.

In [752]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



Fusion of the data used.

In [753]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```

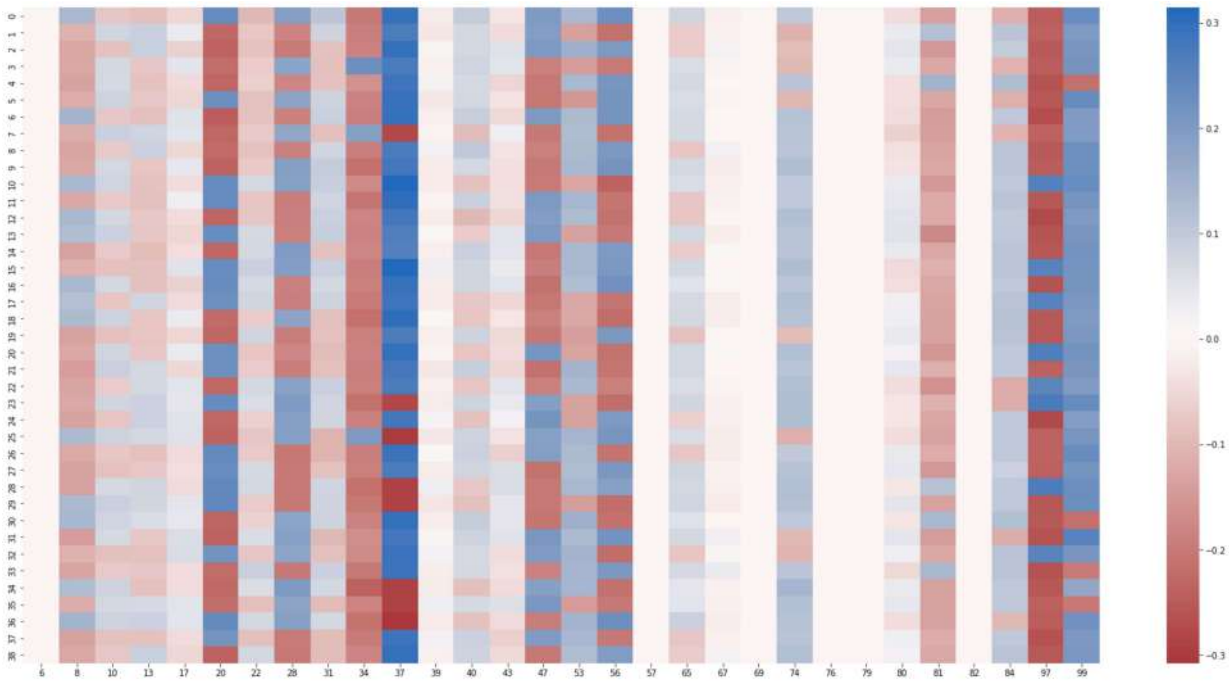


Individual explainability of data belonging to the third class, in this

case High.

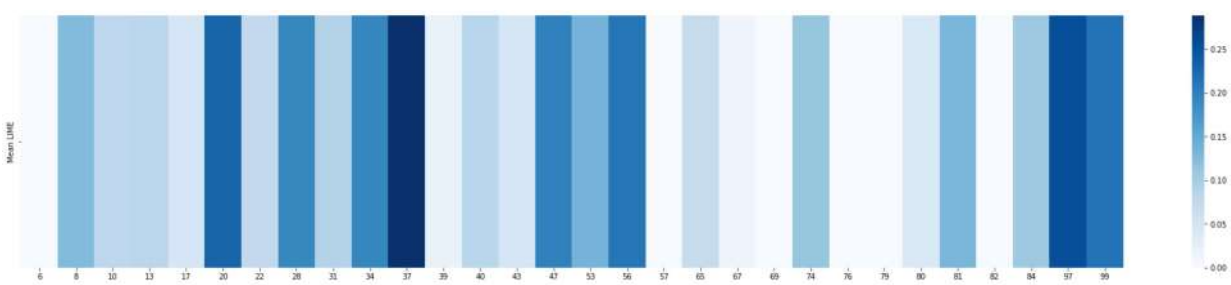
Generation of the heatmap.

In [754]:
```
iterable_class += 1
plt.figure(figsize = (28,14))
lime_df = pd.DataFrame(exp_matrix[iterable_class])
lime_df.columns = column_Map
sns.heatmap(lime_df, cmap="vlag_r",yticklabels=True, xticklabels=True)
plt.show()
```



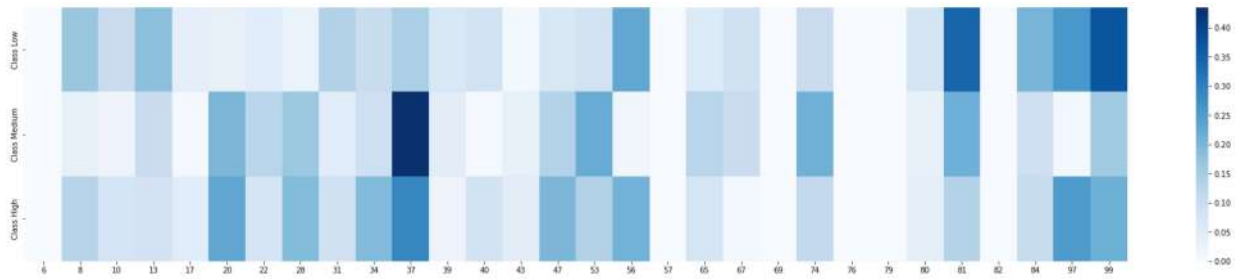
Fusion of the data used.

In [755]:
```
lime_df_mean = lime_df.abs().mean(axis = 0).to_frame()
lime_df_mean.columns= ["Mean LIME"]
map_class.append(lime_df_mean)

plt.figure(figsize = (32,6))
sns.heatmap(map_class[iterable_class].transpose(), cmap="Reds", cbar = False)
sns.heatmap(map_class[iterable_class].transpose(), cmap="Blues", yticklabels = Tr
            mask = df_mask.to_numpy())
plt.show()
```



Merging of all the class heatmaps into one map.

In [756]:
```
df_lime = pd.concat(map_class, axis=1)
df_lime.columns = name_class
plt.figure(figsize = (32,6))
sns.heatmap(df_lime.transpose(), cmap="Reds", cbar = False)
sns.heatmap(df_lime.transpose(), cmap="Blues", yticklabels = True, xticklabels =
plt.show()
```



Identification of feature variables using ALE

It shows the main effect of each question compared to the probability that the model predicts in each class. This effect is shown for both responses. Since there are only 2 answers, the map is the same but with the colors reversed.

In [757]:
```
## Definition of explainability data.
proba_fun_lr = mlp.predict_proba
proba_ale_lr = ALE(proba_fun_lr, feature_names=train_cols, target_names= target_r
proba_exp_lr = proba_ale_lr.explain(X_train.to_numpy())

## Class index local variable.
iterable_class = -1
map_class = []
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

```
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
```

```
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
X does not have valid feature names, but MLPClassifier was fitted with feature
names
```

Obtaining the data for the generation of the explanations by ALE about all the classes.

In [758]:
```
## Local variables for ALE array data.
ale_list = [[] for _ in range(len(name_class))]

## Recovering the data needed for ALE.
for array in proba_exp_lr.ale_values:

    ## Recovering the data.
    for elements in range(0, len(name_class)):

        ## Saving data of all the classes.
        ale_list[elements].append(array[0][elements])
```

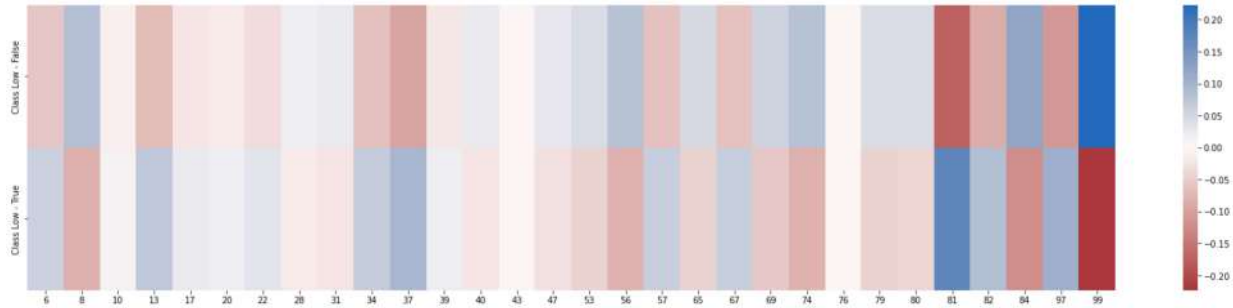
Data processing for the explainability of ALE.

In [759]:
```
## Data processing of the Low, Medium and High classes.
for elements in range(0, len(name_class)):
    ale_df = pd.DataFrame([ale_list [elements]])
    ale_df = pd.concat([ale_df.multiply(-1), ale_df])
    ale_df.index = [name_class [elements] + " - False", name_class [elements] + '
    ale_df.columns = column_Map
    map_class.append(ale_df)
```

Explainability of the data belonging to the first class, in this case the Low.

Generation of the heatmap.

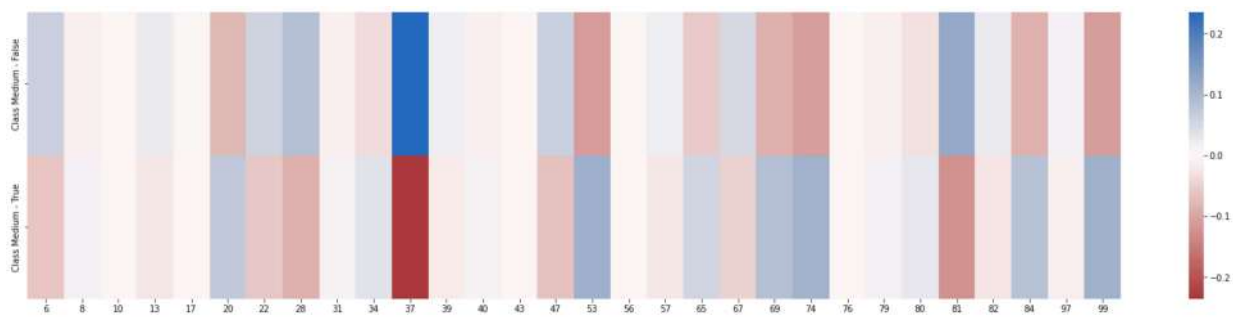
In [760]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the second class, in this case the Medium.

Generation of the heatmap.

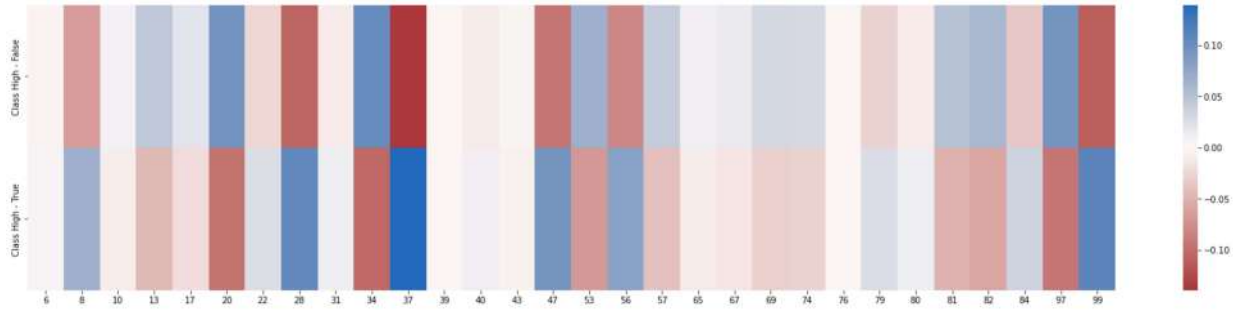
In [761]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```



Explainability of the data belonging to the third class, in this case the High.

Generation of the heatmap.

In [762]:
```
iterable_class += 1
plt.figure(figsize = (28,6))
sns.heatmap(map_class[iterable_class], cmap="vlag_r",yticklabels=True, xticklabel
plt.show()
```

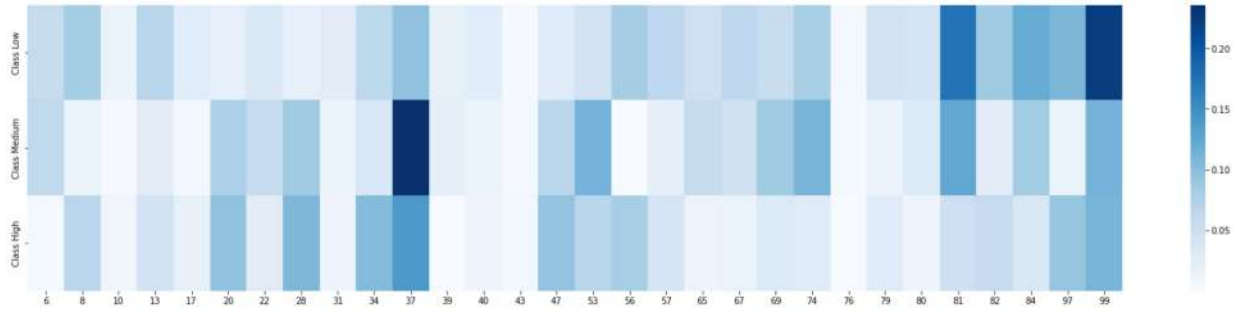


Fusion of the data used in all classes.

In [763]:
```
## Application of absolute value to data.
for index in range(0, len(map_class)):
    map_class [index] = map_class [index][0:1].abs()

## Union of the heatmaps of the Low, Medium and High class.
ale_df = pd.concat(map_class)
ale_df.index = name_class
ale_df = ale_df.transpose()
```

Generation of the heatmap.

In [764]:
```
plt.figure(figsize = (28,6))
sns.heatmap(ale_df.transpose(), cmap="Reds", cbar = False)
sns.heatmap(ale_df.transpose(), cmap="Blues", yticklabels = True, xticklabels = T
plt.show()
```



Comparing of Feature Variables Methods for Explainability

The global values of each of the methods used are taken, their maximums are obtained (the minimums are 0) and a percentage is calculated based on said maximum, which indicates how relevant the contribution is for each question.

In [765]:
```
## Class index local variable.
iterable_class = -1
```

Comparison of the explanation methods of the first class, in this case the Low.

The highest data of each method used is obtained.

In [766]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [767]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

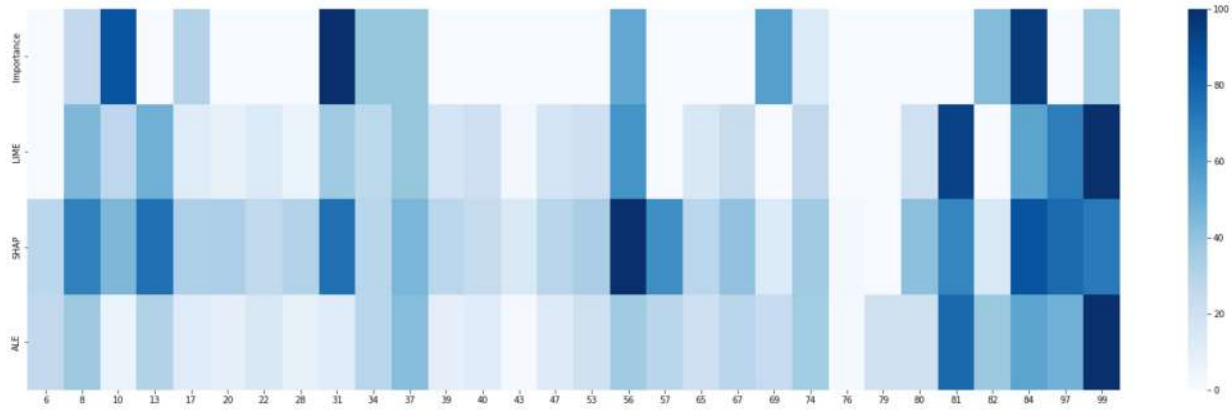
Out[767]:

	6	8	10	13	17	20	22	28
Importance	0.000000	26.086957	86.956522	0.000000	30.434783	0.000000	0.000000	0.000000
LIME	0.000000	45.105159	27.067478	48.781555	11.363125	8.660121	13.176770	6.790543
SHAP	28.017563	69.082163	45.328204	75.431252	32.402790	32.848892	26.940909	30.958135
ALE	26.508037	37.127128	6.155564	31.253503	11.684498	9.170113	15.309044	8.847577

4 rows × 30 columns

Generation of the heatmap.

In [768]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the second class, in this case the Medium.

The highest data of each method used is obtained.

In [769]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [770]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

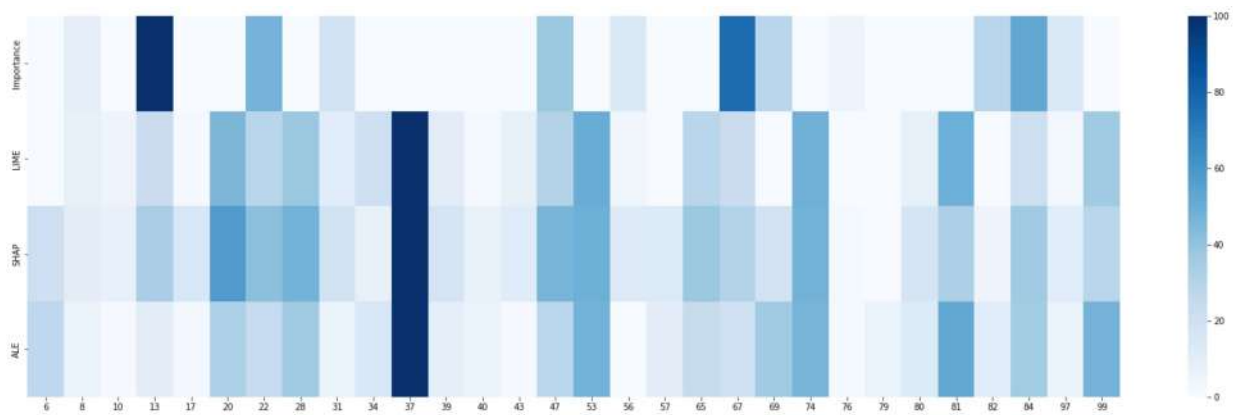
Out[770]:

	6	8	10	13	17	20	22	28
Importance	0.000000	9.523810	0.000000	100.000000	0.000000	0.000000	47.619048	0.000000
LIME	0.000000	8.861767	5.117370	23.136477	1.918574	45.726814	28.628079	37.969464
SHAP	21.180826	10.314919	8.396811	33.963069	15.830209	57.439947	41.830935	47.909833
ALE	27.030092	6.732205	1.424100	10.343195	2.107816	32.185399	25.276803	36.833793

4 rows × 30 columns

Generation of the heatmap.

In [771]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Comparison of the explanation methods of the third class, in this case the High.

The highest data of each method used is obtained.

In [772]:
```
## The maximum present value is obtained in each explainability method.
iterable_class += 1
max_importance = df_importance[name_class[iterable_class]].max()
max_shap = df_shap[name_class[iterable_class]].max()
max_lime = df_lime[name_class[iterable_class]].max()
max_ale  = ale_df[name_class[iterable_class]].max()
```

A new heatmap is created containing all the values of each method used.

In [773]:
```
## The values are calculated to plot the new heatmap.
df_general = pd.DataFrame([df_importance[name_class[iterable_class]].multiply(100
                           df_lime[name_class[iterable_class]].multiply(100/max_
                           df_shap[name_class[iterable_class]].multiply(100/max_
                           ale_df[name_class[iterable_class]].multiply(100/max_a

## Method names are assigned.
df_general.index = name_methods
df_general
```

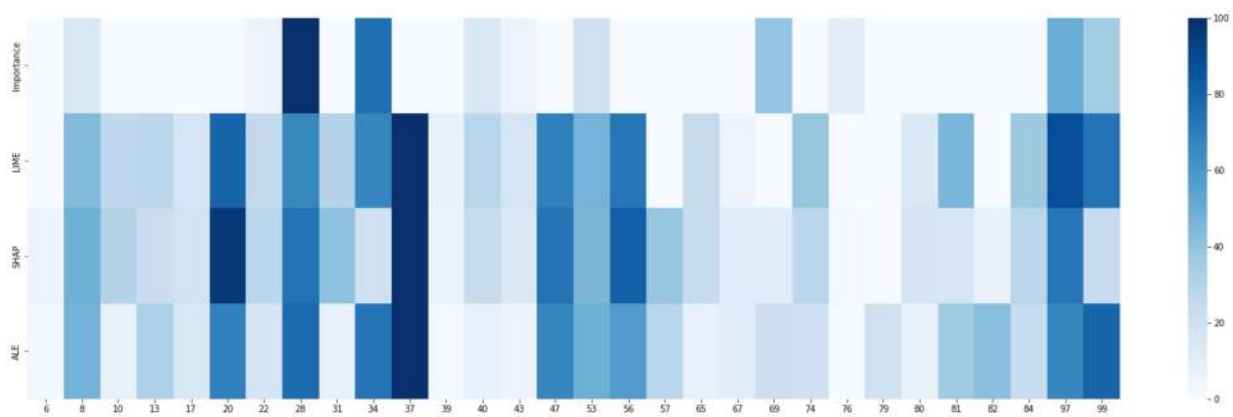
Out[773]:

	6	8	10	13	17	20	22	28
Importance	0.000000	15.000000	0.000000	0.000000	0.000000	0.000000	5.000000	100.00000
LIME	0.000000	44.916199	27.255985	28.175945	16.750585	80.020401	26.076856	65.92811
SHAP	5.058905	49.083101	30.527346	23.205659	18.587527	95.764479	28.317070	74.52463
ALE	3.307577	47.939057	7.428073	32.457166	15.105621	69.009798	18.236407	76.34564

4 rows × 30 columns

Generation of the heatmap.

In [774]:
```
plt.figure(figsize = (28,8))
sns.heatmap(df_general, cmap="Reds", cbar = False)
sns.heatmap(df_general, cmap="Blues", yticklabels = True, xticklabels = True, mas
plt.show()
```



Safeguarding the items eliminated during this iteration.

In [775]:
```
## Removal of anomalous items selected by selection_mode.
for x in question_anomaly:

    ## Status change of the deleted items in the respective auxiliary dataset.
    ds.loc[x - 1,'Status'] = 'Delete'

## Save the changed statuses in the dataset.
ds.to_csv(path_dataset_qs, index = False)
```

Generation of final graphs from the results obtained by the improvement process

Different graphs are generated with the aim of clearly illustrating the data obtained from the model improvement process.

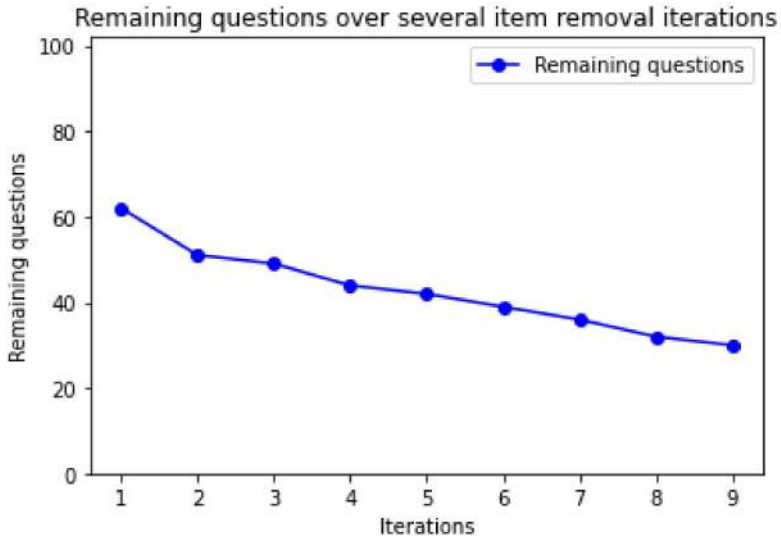
In [776]:
```
## Loading data from the dataset.
df = pd.read_csv(path_dataset)
dd = pd.read_csv(path_dataset_qd)

## Eliminating physical-related items in dataset.
df = df.drop(df.columns[102:-1], axis=1)
```

All the data necessary to generate the graphs are obtained.

In [777]:
```
## Local variables for graph generation.
total_question = []
global_accuracy = []
total_delete = []
recalls = [[] for _ in range(len(name_class))]
labels_plot = []

## Get the total number of iterations.
iterations = list(map(str, list(range(1, (dd.shape[0] + 1)))))

## Obtaining data from the dataset.
for x in range(dd.shape[0]):

    ## The total number of items used in each iteration is obtained.
    total_question.append((df.shape[1] - 1) - len((dd.loc[x, 'Question']).split('

    ## The global acurracy of each iteration is obtained.
    global_accuracy.append((dd.loc[x, 'Acurracy global']) / 100)

    ## The total number of items eliminated in each iteration is obtained.
    if x == 0:
        total_delete.append(len((dd.loc[x, 'Question']).split(',')))
    else:
        total_delete.append(len((dd.loc[x, 'Question']).split(',')) - len((dd.loc

    ## The recall of all classes of each iteration is obtained.
    for index in range(0, len(name_class)):
        recalls[index].append(dd.loc[x, 'Recall_' + str(index)])

for target in target_names:
    labels_plot.append(target + " Class")
```

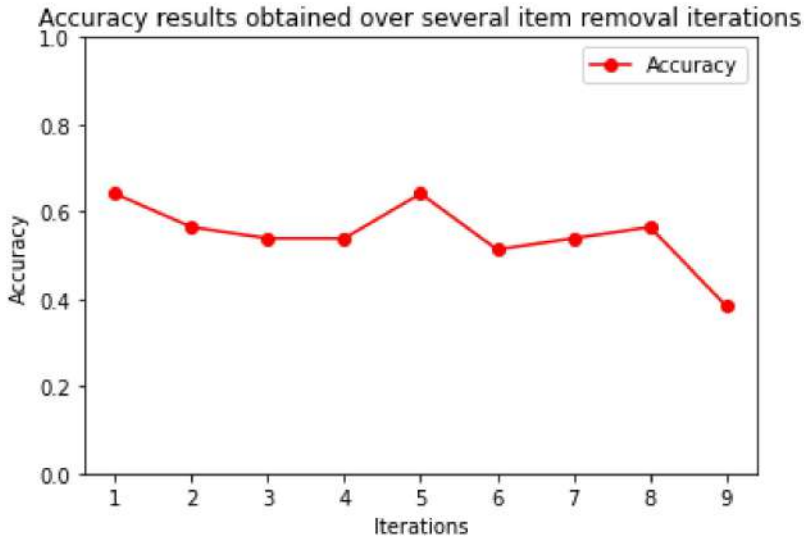
Generation of the graph of items used in each iteration.

In [778]:
```
## Generation of the graph according to the data obtained.
plt.figure()
labels = ['Remaining questions']
plt.title('Remaining questions over several item removal iterations')
plt.plot(iterations, total_question, marker= 'o', color = 'blue')
plt.ylabel("Remaining questions")
plt.xlabel("Iterations")
plt.ylim(0, df.shape[1] - 1)
plt.legend(labels)
plt.show()
```



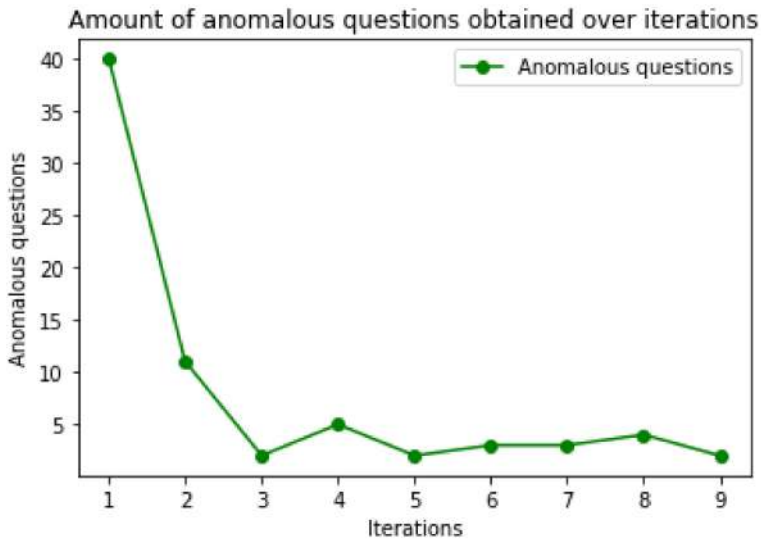
Generation of the graph of the acurracy obtained in each iteration.

In [779]:
```
## Generation of the graph according to the data obtained.
plt.figure()
labels=['Accuracy']
plt.title('Accuracy results obtained over several item removal iterations')
plt.plot(iterations, global_accuracy, marker= 'o', color = 'red')
plt.ylabel("Accuracy")
plt.xlabel("Iterations")
plt.ylim(0, 1)
plt.legend(labels)
plt.show()
```



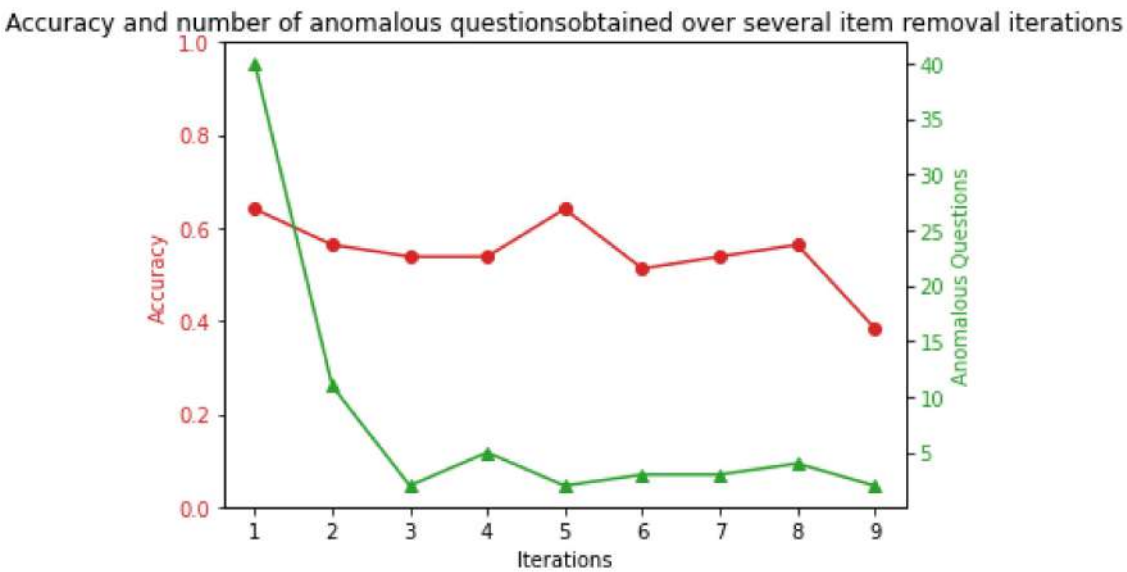
Generation of the graph of items eliminated in each iteration.

In [780]:
```
## Generation of the graph according to the data obtained.
plt.figure()
labels=['Anomalous questions']
plt.title('Amount of anomalous questions obtained over iterations')
plt.plot(iterations, total_delete, marker= 'o', color = 'green')
plt.xlabel("Iterations")
plt.ylabel("Anomalous questions")
plt.legend(labels)
plt.show()
```



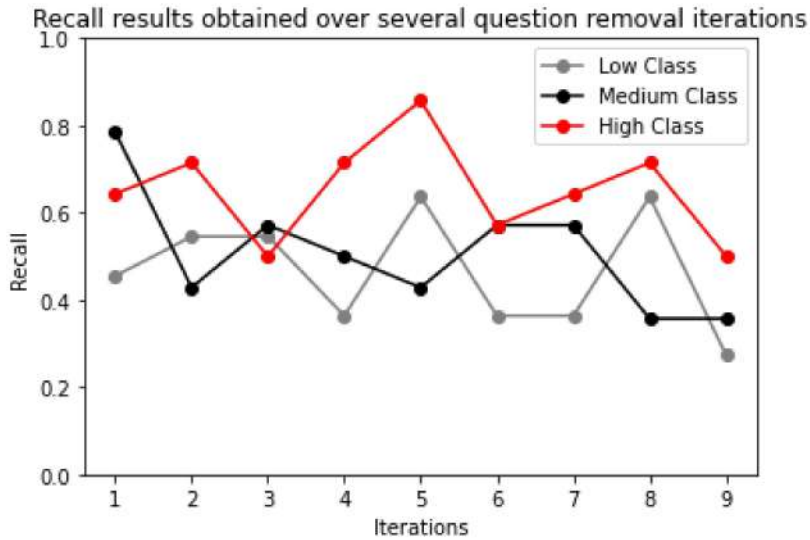
The graph is generated with the merger of the acurracy with the anomadic items eliminated.

In [781]:
```
## Generation of the graph according to the data obtained.
fig, ax = plt.subplots()

ax.set_xlabel('Iterations')
ax.set_ylabel('Accuracy', color = 'tab:red')
ax.plot(iterations, global_accuracy, marker= 'o', color= 'tab:red')
ax.tick_params(axis='y', labelcolor= 'tab:red')

plt.ylim(0, 1)
ax = ax.twinx()

ax.set_ylabel('Anomalous Questions', color= 'tab:green')
ax.plot(iterations, total_delete, marker= '^', color= 'tab:green')
ax.tick_params(axis='y', labelcolor= 'tab:green')

fig.tight_layout()
plt.title('Accuracy and number of anomalous questions' +
          'obtained over several item removal iterations')
plt.xticks(range(0, iteration, 1))
plt.show()
```



The graph is generated with the fusion of the recall obtained in the improvement process.

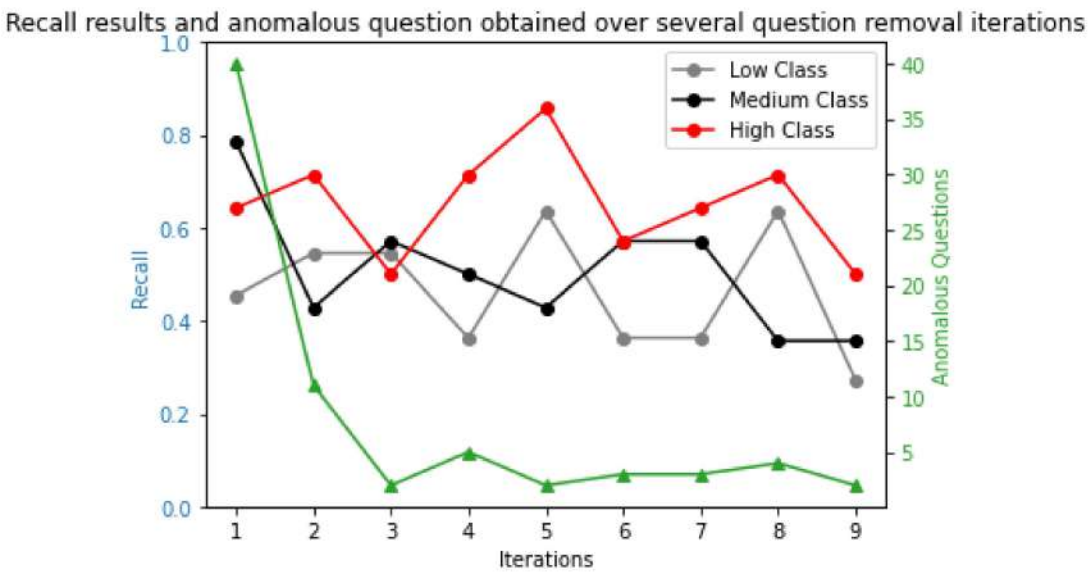
In [782]:
```
## Generation of the graph according to the data obtained.
plt.figure()
plt.title('Recall results obtained over several question removal iterations')
for index in range(0, len(name_class)):
    plt.plot(iterations, recalls[index], marker= 'o',
             color = list_color[index])
plt.xlabel("Iterations")
plt.ylabel("Recall")
plt.ylim(0, 1)
plt.legend(labels_plot)
plt.show()
```



The graph is generated with the fusion of the recall and the eliminated items obtained in the improvement process.

In [783]:

```
## Generation of the graph according to the data obtained.
fig, ax = plt.subplots()

ax.set_xlabel('Iterations')
ax.set_ylabel('Recall', color= 'tab:blue')
for index in range(0, len(name_class)):
    plt.plot(iterations, recalls[index], marker= 'o',
             color = list_color[index])
ax.tick_params(axis='y', labelcolor= 'tab:blue')

plt.ylim(0, 1)
plt.legend(labels_plot)
ax = ax.twinx()

ax.set_ylabel('Anomalous Questions', color = 'tab:green')
ax.plot(iterations, total_delete, marker= '^', color = 'tab:green')
ax.tick_params(axis='y', labelcolor = 'tab:green')

fig.tight_layout()
plt.title('Recall results and anomalous question obtained ' +
          'over several question removal iterations')
plt.xticks(range(0, iteration, 1))
plt.show()
```



In [ ]: