Explainable AutoML - Titanic Survival Classification Demo

In [1]:
```python
# Author Hussain Abbas
# Copyright © Stats AI 2021. All Rights Reserved

import tensorflow as tf
import autokeras as ak
from tensorflow.keras import backend as K
import keras_tuner
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import re
from sklearn.model_selection import cross_val_score, KFold, train_test_split
from sklearn.metrics import roc_auc_score, precision_score, recall_score, fbeta_score, roc_curve
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from tqdm import tqdm
```

In [2]:
```python
# Verify GPU is detected and working
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  1
```

In [3]:
```python
TRAIN_DATA_URL = "https://storage.googleapis.com/tf-datasets/titanic/train.csv"
TEST_DATA_URL = "https://storage.googleapis.com/tf-datasets/titanic/eval.csv"

#datasets located in C:/Users/USER/.keras/datasets

train_file_path = tf.keras.utils.get_file("train.csv", TRAIN_DATA_URL)
test_file_path = tf.keras.utils.get_file("eval.csv", TEST_DATA_URL)

train_data = pd.read_csv(train_file_path)
test_data = pd.read_csv(test_file_path)

df = pd.concat([train_data, test_data])
df.drop_duplicates(inplace=True)
df = df.reset_index()
df = df.drop(['index'], axis=1)

print('All Data Summary')
print(df.describe())
print('\n')

print('Train Data Summary')
print(train_data.describe())
print('\n')

print('Test Data Summary')
print(test_data.describe())
```

```
All Data Summary
         survived         age  n_siblings_spouses       parch        fare
count  781.000000  781.000000          781.000000  781.000000  781.000000
mean     0.413572   29.622817            0.524968    0.417414   34.750464
std      0.492789   13.764671            0.987592    0.838132   52.237906
min      0.000000    0.420000            0.000000    0.000000    0.000000
25%      0.000000   22.000000            0.000000    0.000000    8.050000
50%      0.000000   28.000000            0.000000    0.000000   15.900000
75%      1.000000   36.000000            1.000000    1.000000   34.020800
max      1.000000   80.000000            8.000000    6.000000  512.329200


Train Data Summary
         survived         age  n_siblings_spouses       parch        fare
count  627.000000  627.000000          627.000000  627.000000  627.000000
mean     0.387560   29.631308            0.545455    0.379585   34.385399
std      0.487582   12.511818            1.151090    0.792999   54.597730
min      0.000000    0.750000            0.000000    0.000000    0.000000
25%      0.000000   23.000000            0.000000    0.000000    7.895800
50%      0.000000   28.000000            0.000000    0.000000   15.045800
75%      1.000000   35.000000            1.000000    0.000000   31.387500
max      1.000000   80.000000            8.000000    5.000000  512.329200


Test Data Summary
         survived         age  n_siblings_spouses       parch        fare
count  264.000000  264.000000          264.000000  264.000000  264.000000
mean     0.375000   28.720985            0.469697    0.386364   27.023880
std      0.485042   14.157538            0.978393    0.837775   34.973108
min      0.000000    0.420000            0.000000    0.000000    0.000000
25%      0.000000   21.000000            0.000000    0.000000    7.925000
50%      0.000000   28.000000            0.000000    0.000000   13.250000
75%      1.000000   35.250000            1.000000    0.000000   27.900000
max      1.000000   74.000000            8.000000    6.000000  263.000000
```

In [4]:
```
print('Train Data')
train_data.head()
```

```
Train Data
```

Out[4]:

	survived	sex	age	n_siblings_spouses	parch	fare	class	deck	embark_town	alone
0	0	male	22.0	1	0	7.2500	Third	unknown	Southampton	n
1	1	female	38.0	1	0	71.2833	First	C	Cherbourg	n
2	1	female	26.0	0	0	7.9250	Third	unknown	Southampton	y
3	1	female	35.0	1	0	53.1000	First	C	Southampton	n
4	0	male	28.0	0	0	8.4583	Third	unknown	Queenstown	y

In [5]:
```
print('Test Data')
test_data.head()
```

```
Test Data
```

Out[5]:

	survived	sex	age	n_siblings_spouses	parch	fare	class	deck	embark_town	alone
0	0	male	35.0	0	0	8.0500	Third	unknown	Southampton	y
1	0	male	54.0	0	0	51.8625	First	E	Southampton	y
2	1	female	58.0	0	0	26.5500	First	C	Southampton	y
3	1	female	55.0	0	0	16.0000	Second	unknown	Southampton	y
4	1	male	34.0	0	0	13.0000	Second	D	Southampton	y

In [6]:
```
def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f_beta_score(y_true, y_pred):

    a = 0.5 ** 2
    b = 1 + a

    precision = precision_m(y_true, y_pred)

    recall = recall_m(y_true, y_pred)

    return b*((precision*recall)/(a*precision+recall+K.epsilon()))


def ak_predict(model, data):

    pred_input = data.astype(np.compat.unicode)
    predicted = model.predict(pred_input).flatten()
    pred_result = predicted

    #cut_off = 0.5

    #pred_result = [1 if x > cut_off else 0 for x in predicted]

    return pred_result

def jdl(y_true, y_pred, smooth=100):

    """
    Jaccard = (|X & Y|)/ (|X|+ |Y| - |X & Y|)
            = sum(|A*B|)/(sum(|A|)+sum(|B|)-sum(|A*B|))

    The jaccard distance loss is usefull for unbalanced datasets. This has been
    shifted so it converges on 0 and is smoothed to avoid exploding or disapearing
    gradient.

    https://en.wikipedia.org/wiki/Jaccard_index

    https://gist.github.com/wassname/f1452b748efcbeb4cb9b1d059dce6f96

    """
    intersection = K.sum(K.abs(y_true * y_pred), axis=-1)
    sum_ = K.sum(K.abs(y_true) + K.abs(y_pred), axis=-1)
    jac = (intersection + smooth) / (sum_ - intersection + smooth)
    return (1 - jac) * smooth
```

In [7]:
```python
from tensorflow.keras.utils import CustomObjectScope
from sklearn.utils import class_weight

with CustomObjectScope({'f_beta_score': f_beta_score,
                        'jdl': jdl, }):

    results = []

    # number of times we partition the data into training/test set
    outer_loop_folds = 2

    # number of times we partition the training data into training/validation set
    inner_loop_folds = 2

    #max_trials: Default= 100. The max num of different models to try
    num_trials = 20

    #epochs: If unspecified, we use epochs equal to 1000 and early stopping with patience equal to 30
    epochs = 3000

    #Since we are using early stopping, we can set an arbitrarily high number of epochs and let the computer handle it
    Early_Stopping = tf.keras.callbacks.EarlyStopping(monitor='val_f_beta_score', patience=101)


    for j in tqdm(range(outer_loop_folds)):

        #Randomly split df into 80% train, 20% test

        x_train, x_test, y_train, y_test = train_test_split(df.drop('survived', axis=1),
                                                             df.survived, test_size=0.2,
                                                            stratify = df.survived)

        for i in tqdm(range(inner_loop_folds)):

            # Further randomly split the 80% train into 64% train and 16% validation

            x_inner_train, x_inner_val, y_inner_train, y_inner_val = train_test_split(x_train,
                                                             y_train, test_size=0.2,
                                                            stratify = y_train)

            w = y_inner_train.value_counts(normalize = True)[0]/y_inner_train.value_counts(normalize = True)[1]
            cw = {0: 1., 1: w}
            #cw = {0: 1., 1: 0.5}


            # Try max_trial different models
            clf = ak.StructuredDataClassifier(
                overwrite=True,
                max_trials = num_trials,

                #tuner = 'random',
                #tuner = 'hyperband',
                tuner = 'bayesian',

                metrics=[jdl,
                        'binary_crossentropy',
                         tf.keras.metrics.AUC(name='auc'),
                        tf.keras.metrics.BinaryAccuracy(name='accuracy'),
                         tf.keras.metrics.Precision(name='precision'),
                         tf.keras.metrics.Recall(name='recall'),
                         f_beta_score],

                objective=keras_tuner.Objective('val_f_beta_score', direction='max'),
                #objective=keras_tuner.Objective('val_jdl', direction='min'),

                #loss = jdl,

            )

            try:
                # Fit the best model
                clf.fit(x_inner_train, y_inner_train,
                        validation_data = (x_inner_val, y_inner_val),
                        #class_weight = cw
                        epochs = epochs,
                        callbacks = [Early_Stopping]
                       )

                # Predict with the best model
                x = clf.evaluate(x_test, y_test)
                x_test_loss, x_jdl, x_bc, x_auc, x_accuracy, x_precision, x_recall, x_f_beta_score= x

                # Save the results
                model_name = 'model_autokeras_' + str(j) + '_'+ str(i)

                results.append([model_name, j, i,
                                x_test_loss, x_jdl, x_bc,
                                x_auc, x_accuracy,
                                x_precision, x_recall,
                                x_f_beta_score])

            except:
                print("Issue training model")

            try:
```

```python
                # Save the model after each j, i iteration
                model = clf.export_model()
                model.save(model_name, save_format="tf")

            except:
                print("Issue saving model")


results = pd.DataFrame(results, columns = ['model_name', 'j', 'i', 'Test_loss','Loss:JDL', 'Loss:Binary Cross Entropy',
                                           'AUC', 'Accuracy', 'Precision', 'Recall', 'F_Beta_Score'])
```

```
Trial 20 Complete [00h 00m 33s]
val_f_beta_score: 0.6340950727462769

Best val_f_beta_score So Far: 0.8041665554046631
Total elapsed time: 00h 13m 48s
INFO:tensorflow:Oracle triggered exit
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` wil
l be empty until you train or evaluate the model.
INFO:tensorflow:Assets written to: .\structured_data_classifier\best_model\assets
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.iter
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.beta_1
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.beta_2
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.decay
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.learning_rate
WARNING:tensorflow:A checkpoint was restored (e.g. tf.train.Checkpoint.restore or tf.keras.Model.load_weights) but not a
ll checkpointed values were used. See above for specific issues. Use expect_partial() on the load status object, e.g. t
f.train.Checkpoint.restore(...).expect_partial(), to silence these warnings, or use assert_consumed() to make the check
explicit. See https://www.tensorflow.org/guide/checkpoint#loading_mechanics (https://www.tensorflow.org/guide/checkpoint
#loading_mechanics) for details.
5/5 [==============================] - 1s 8ms/step - loss: 0.5102 - jdl: 0.2376 - binary_crossentropy: 0.5102 - auc: 0.8
523 - accuracy: 0.7707 - precision: 0.8222 - recall: 0.5692 - f_beta_score: 0.7576
INFO:tensorflow:Assets written to: model_autokeras_1_1\assets


100%|██████████████████████████████████████████████████| 2/2 [28:12<00:00, 846.08s/it]
100%|█████████████████████████████████████████████████| 2/2 [55:05<00:00, 1652.69s/it]
```

In [10]:
```python
results.describe()
```

Out[10]:

	j	i	Test_loss	Loss:JDL	Loss:Binary Cross Entropy	AUC	Accuracy	Precision	Recall	F_Beta_Score
count	4.00000	4.00000	4.000000	4.000000	4.000000	4.000000	4.000000	4.000000	4.000000	4.000000
mean	0.50000	0.50000	0.504819	0.260495	0.504819	0.857588	0.800955	0.835385	0.650000	0.792405
std	0.57735	0.57735	0.081223	0.032147	0.081223	0.020403	0.026197	0.054350	0.058076	0.041626
min	0.00000	0.00000	0.405173	0.236323	0.405173	0.840552	0.770701	0.793103	0.569231	0.757560
25%	0.00000	0.00000	0.476339	0.237250	0.476339	0.847826	0.789809	0.806766	0.638462	0.773203
50%	0.50000	0.50000	0.505117	0.250313	0.505117	0.851296	0.799363	0.816772	0.661538	0.779605
75%	1.00000	1.00000	0.533598	0.273558	0.533598	0.861058	0.810510	0.845390	0.673077	0.798807
max	1.00000	1.00000	0.603870	0.305029	0.603870	0.887207	0.834395	0.914894	0.707692	0.852849

In [56]:
```python
results
```

Out[56]:

	model_name	j	i	Test_loss	Loss:JDL	Loss:Binary Cross Entropy	AUC	Accuracy	Precision	Recall	F_Beta_Score
0	model_autokeras_0_0	0	0	0.405173	0.263068	0.405173	0.887207	0.834395	0.914894	0.661538	0.852849
1	model_autokeras_0_1	0	1	0.603870	0.236323	0.603870	0.850251	0.802548	0.793103	0.707692	0.778417
2	model_autokeras_1_0	1	0	0.500061	0.305029	0.500061	0.840552	0.796178	0.811321	0.661538	0.780794
3	model_autokeras_1_1	1	1	0.510173	0.237559	0.510173	0.852341	0.770701	0.822222	0.569231	0.757560

In [13]:
```python
#best_model = results.loc[np.argmax(results.test_accuracy)].model_name

best_model = results.loc[np.argmax(results.F_Beta_Score)].model_name


best_model
```

Out[13]:
```
'model_autokeras_0_0'
```

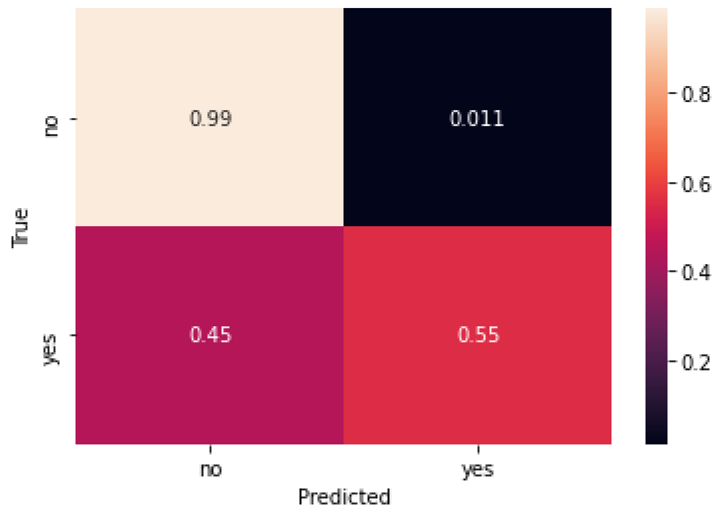
In [14]:
```python
from tensorflow.keras.models import load_model

my_custom_objects={'f_beta_score': f_beta_score,
                   'jdl': jdl, }
my_custom_objects.update(ak.CUSTOM_OBJECTS)

model_ak = load_model(best_model, custom_objects=my_custom_objects)
```

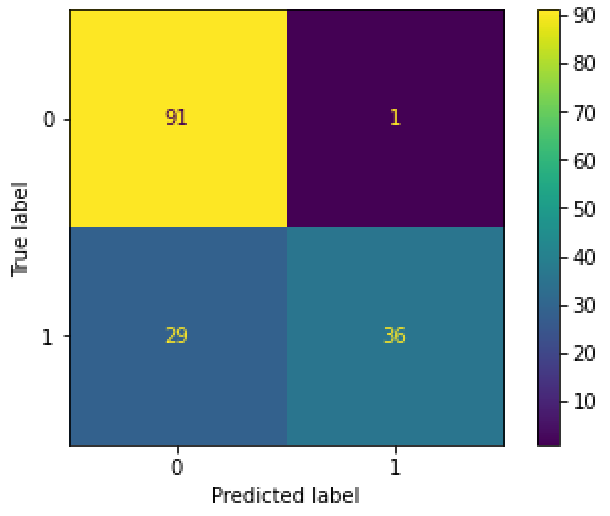
In [15]:
```
model_ak.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 9)]               0
_________________________________________________________________
multi_category_encoding (Mul (None, 9)                 0
_________________________________________________________________
normalization (Normalization (None, 9)                 19
_________________________________________________________________
dense (Dense)                (None, 64)                640
_________________________________________________________________
batch_normalization (BatchNo (None, 64)                256
_________________________________________________________________
re_lu (ReLU)                 (None, 64)                0
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 16)                1040
_________________________________________________________________
batch_normalization_1 (Batch (None, 16)                64
_________________________________________________________________
re_lu_1 (ReLU)               (None, 16)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_2 (Dense)              (None, 16)                272
_________________________________________________________________
batch_normalization_2 (Batch (None, 16)                64
_________________________________________________________________
re_lu_2 (ReLU)               (None, 16)                0
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 17
_________________________________________________________________
classification_head_1 (Activ (None, 1)                 0
=================================================================
Total params: 2,372
Trainable params: 2,161
Non-trainable params: 211
_________________________________________________________________
```

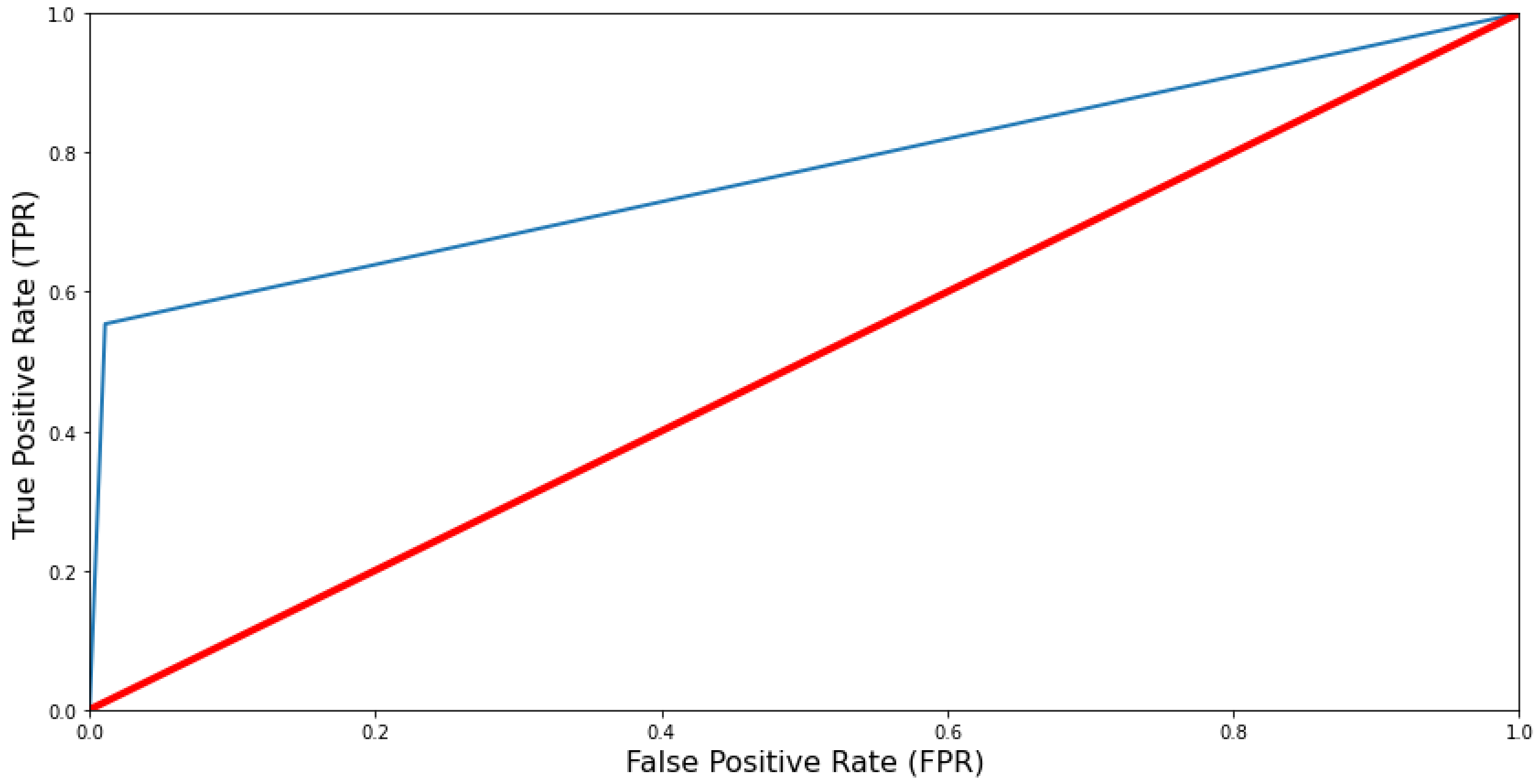
In [16]:
```python
# type: pandas.core.frame.DataFrame
pred_input = x_test.astype(np.compat.unicode)

# type: numpy.ndarray
predicted = model_ak.predict(pred_input).flatten()

cut_off = 0.5

pred_result = [1 if x > cut_off else 0 for x in predicted]
pred_result = np.array(pred_result)

actual = y_test.to_numpy()
actual = actual.flatten()

cm = tf.math.confusion_matrix(actual, pred_result)
cm = cm/cm.numpy().sum(axis=1)[:, tf.newaxis]

sns.heatmap(
    cm, annot=True,
    xticklabels=['no', 'yes'],
    yticklabels=['no', 'yes'])
plt.xlabel("Predicted")
plt.ylabel("True")


'''
https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9
https://stackoverflow.com/questions/44172162/f1-score-vs-roc-auc

- Maximize Precision when False Positives are of concern
- Maximize Recall when False Negatives are of concern
- Maximize F1 Score when both are important and classes are unbalanced

'''

auc_score = roc_auc_score(actual, pred_result)
precision = precision_score(actual, pred_result)
recall = recall_score(actual, pred_result)
f_beta = fbeta_score(actual, pred_result, beta = 0.5)

print("Cut-Off:", cut_off)
print("ROC-AUC-Score:", auc_score)
print('Precision: ' + str(precision))
print('Recall: ' + str(recall))
print('F_Beta: ' + str(f_beta))

y_test_classes = list(set(y_test))

# print Confusion Matrix from Sklearn
cm = confusion_matrix(actual, pred_result, labels = y_test_classes)

#cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels = y_test_classes)
disp.plot();
```

```
Cut-Off: 0.5
ROC-AUC-Score: 0.7714882943143813
Precision: 0.972972972972973
Recall: 0.5538461538461539
F_Beta: 0.8450704225352114
```



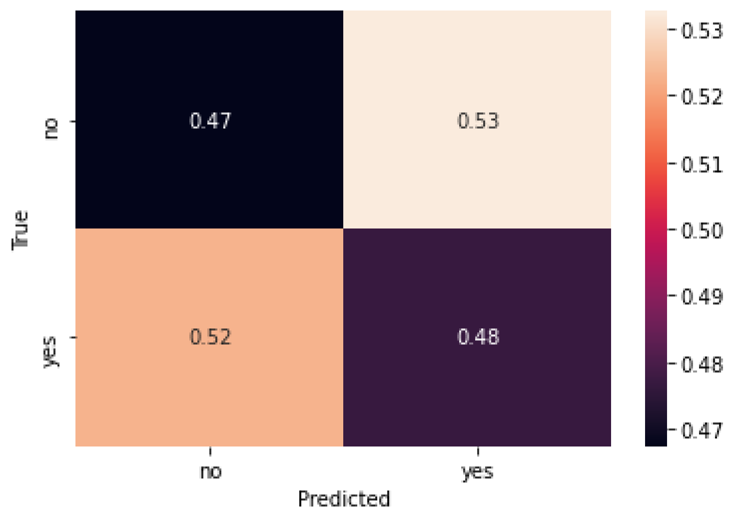


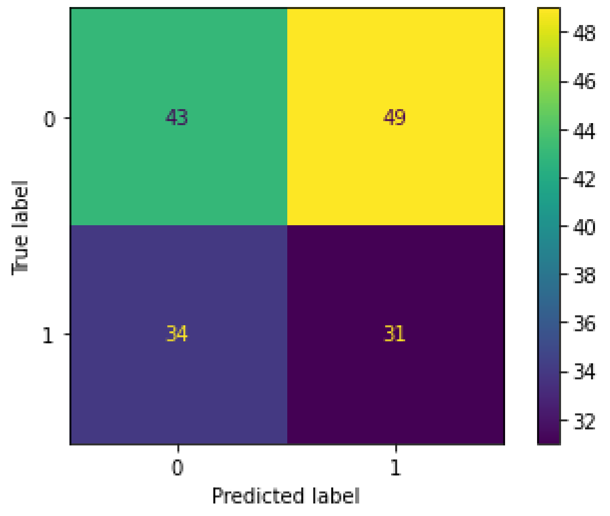
In [17]:
```
# compute true positive rate and false positive rate
false_positive_rate, true_positive_rate, thresholds = roc_curve(actual, pred_result)

# plotting them against each other
def plot_roc_curve(false_positive_rate, true_positive_rate, label=None):
    plt.plot(false_positive_rate, true_positive_rate, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'r', linewidth=4)
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate (FPR)', fontsize=16)
    plt.ylabel('True Positive Rate (TPR)', fontsize=16)

plt.figure(figsize=(14, 7))
plot_roc_curve(false_positive_rate, true_positive_rate)
plt.show()

# https://towardsdatascience.com/predicting-the-survival-of-titanic-passengers-30870ccc7e8
# https://stackoverflow.com/questions/44172162/f1-score-vs-roc-auc
```



In [18]:
```
# Naive Random Coin Flip Classifier Performance

predicted = np.random.randint(0,2, size = len(y_test))
pred_result = predicted.flatten()

actual = y_test.to_numpy()
actual = actual.flatten()

cm = tf.math.confusion_matrix(actual, pred_result)
cm = cm/cm.numpy().sum(axis=1)[:, tf.newaxis]

sns.heatmap(
    cm, annot=True,
    xticklabels=['no', 'yes'],
    yticklabels=['no', 'yes'])
plt.xlabel("Predicted")
plt.ylabel("True")


'''
https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9
https://stackoverflow.com/questions/44172162/f1-score-vs-roc-auc

- Maximize Precision when False Positives are of concern
- Maximize Recall when False Negatives are of concern
- Maximize F1 Score when both are important and classes are unbalanced

'''
auc_score = roc_auc_score(actual, pred_result)
precision = precision_score(actual, pred_result)
recall = recall_score(actual, pred_result)
f_beta = fbeta_score(actual, pred_result, beta = 1)

print("ROC-AUC-Score:", auc_score)
print('Precision: ' + str(precision))
print('Recall: ' + str(recall))
print('F_Beta: ' + str(f_beta))

y_test_classes = list(set(y_test))

# print Confusion Matrix from Sklearn
cm = confusion_matrix(actual, pred_result, labels = y_test_classes)

#cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels = y_test_classes)
disp.plot();
```

```
ROC-AUC-Score: 0.47215719063545153
Precision: 0.3875
Recall: 0.47692307692307695
F_Beta: 0.4275862068965517
```





Explainable AI using Dalex

In [19]:
```
import dalex as dx
```

In [20]:
```
X, y = df.drop('survived', axis=1), df.survived
n, p = X.shape
```

In [21]:
```
explainer_keras = dx.Explainer(model_ak,
                               data = X,
                               y = y,
                               predict_function = ak_predict,
                               label = 'autokeras',
                               #predict_function = dx._explainer.yhat.yhat_tf_classification,
                               model_type = 'classification'
                              )
```

```
Preparation of a new explainer is initiated

  -> data              : 781 rows 9 cols
  -> target variable   : Parameter 'y' was a pandas.Series. Converted to a numpy.ndarray.
  -> target variable   : 781 values
  -> model_class       : tensorflow.python.keras.engine.functional.Functional (default)
  -> label             : autokeras
  -> predict function  : <function ak_predict at 0x00000240C4BF7040> will be used
  -> predict function  : Accepts pandas.DataFrame and numpy.ndarray.
  -> predicted values  : min = 0.0333, mean = 0.423, max = 1.0
  -> model type        : classification will be used
  -> residual function : difference between y and yhat (default)
  -> residuals         : min = -0.994, mean = -0.00966, max = 0.884
  -> model_info        : package tensorflow

A new explainer has been created!
```

In [22]:
```
explainer_keras.model_performance()
```

Out[22]:

	recall	precision	f1	accuracy	auc
autokeras	0.631579	0.914798	0.747253	0.823303	0.87633

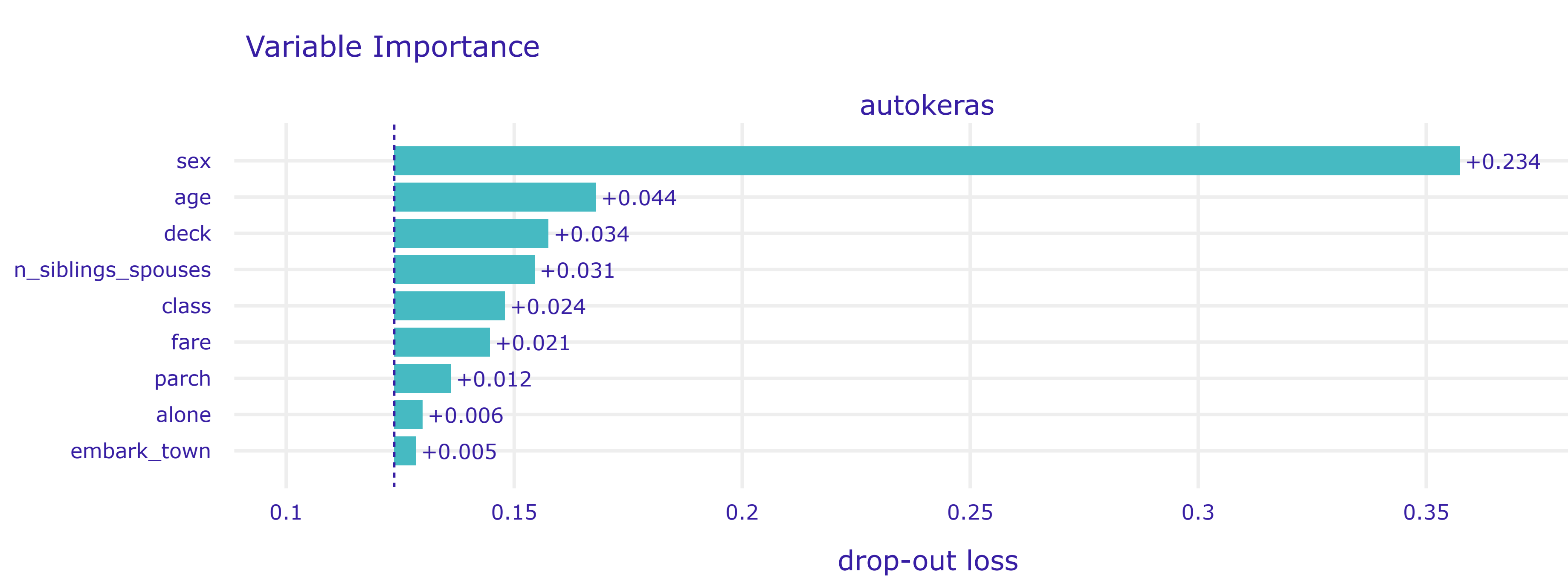
In [23]:
```
explainer_keras.model_diagnostics().result
```

Out[23]:

	sex	age	n_siblings_spouses	parch	fare	class	deck	embark_town	alone	y	y_hat	residuals	abs_residuals	label	ids
0	male	22.0	1	0	7.2500	Third	unknown	Southampton	n	0	0.166383	-0.166383	0.166383	autokeras	1
1	female	38.0	1	0	71.2833	First	C	Cherbourg	n	1	0.990817	0.009183	0.009183	autokeras	2
2	female	26.0	0	0	7.9250	Third	unknown	Southampton	y	1	0.449717	0.550283	0.550283	autokeras	3
3	female	35.0	1	0	53.1000	First	C	Southampton	n	1	0.962952	0.037048	0.037048	autokeras	4
4	male	28.0	0	0	8.4583	Third	unknown	Queenstown	y	0	0.189832	-0.189832	0.189832	autokeras	5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
776	female	56.0	0	1	83.1583	First	C	Cherbourg	n	1	0.969260	0.030740	0.030740	autokeras	777
777	female	25.0	0	1	26.0000	Second	unknown	Southampton	n	1	0.986698	0.013302	0.013302	autokeras	778
778	male	33.0	0	0	7.8958	Third	unknown	Southampton	y	0	0.194580	-0.194580	0.194580	autokeras	779
779	female	39.0	0	5	29.1250	Third	unknown	Queenstown	n	0	0.253465	-0.253465	0.253465	autokeras	780
780	male	26.0	0	0	30.0000	First	C	Cherbourg	y	1	0.246247	0.753753	0.753753	autokeras	781

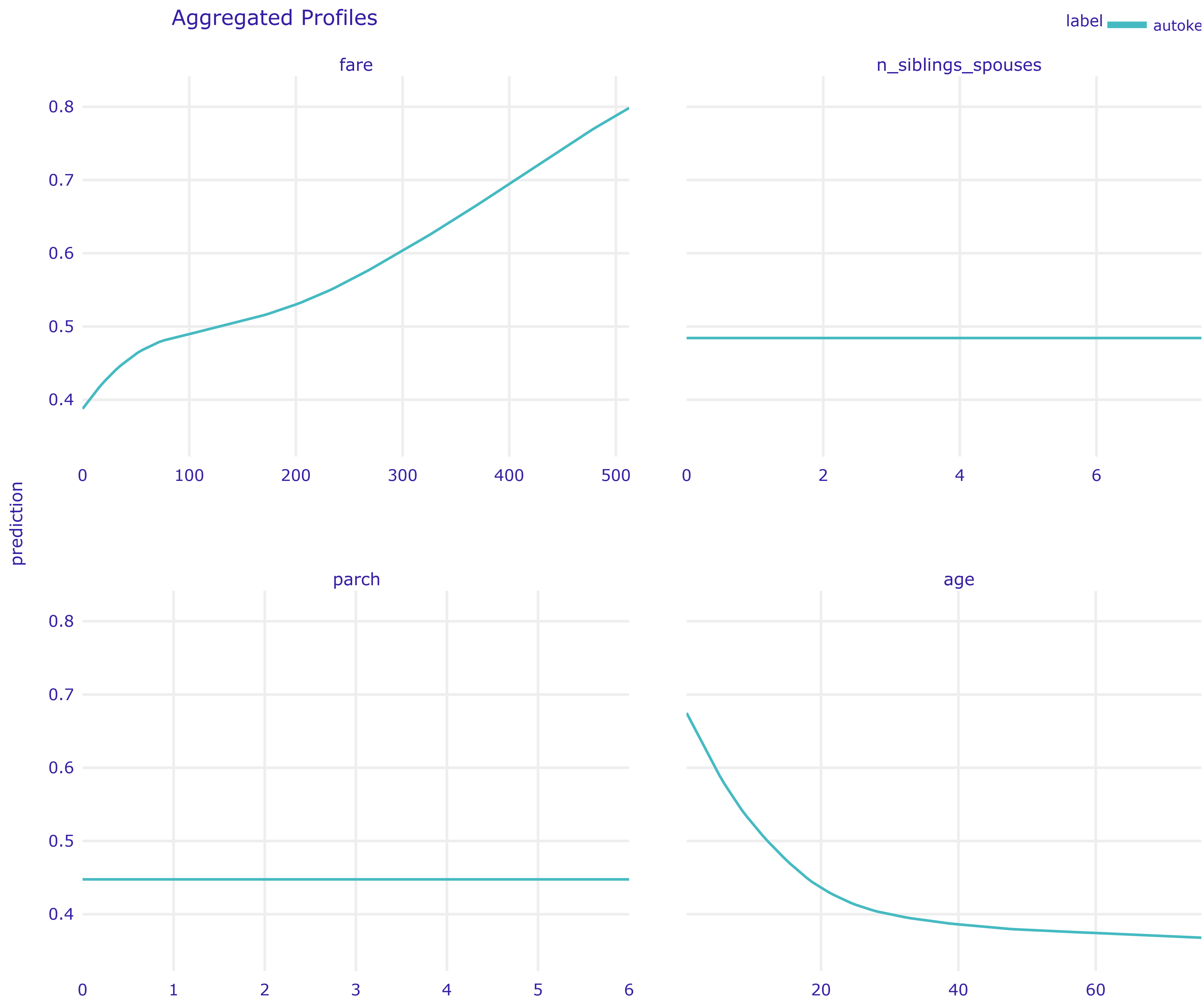
781 rows × 15 columns

In [24]:
```
explainer_keras.model_parts().plot()
```

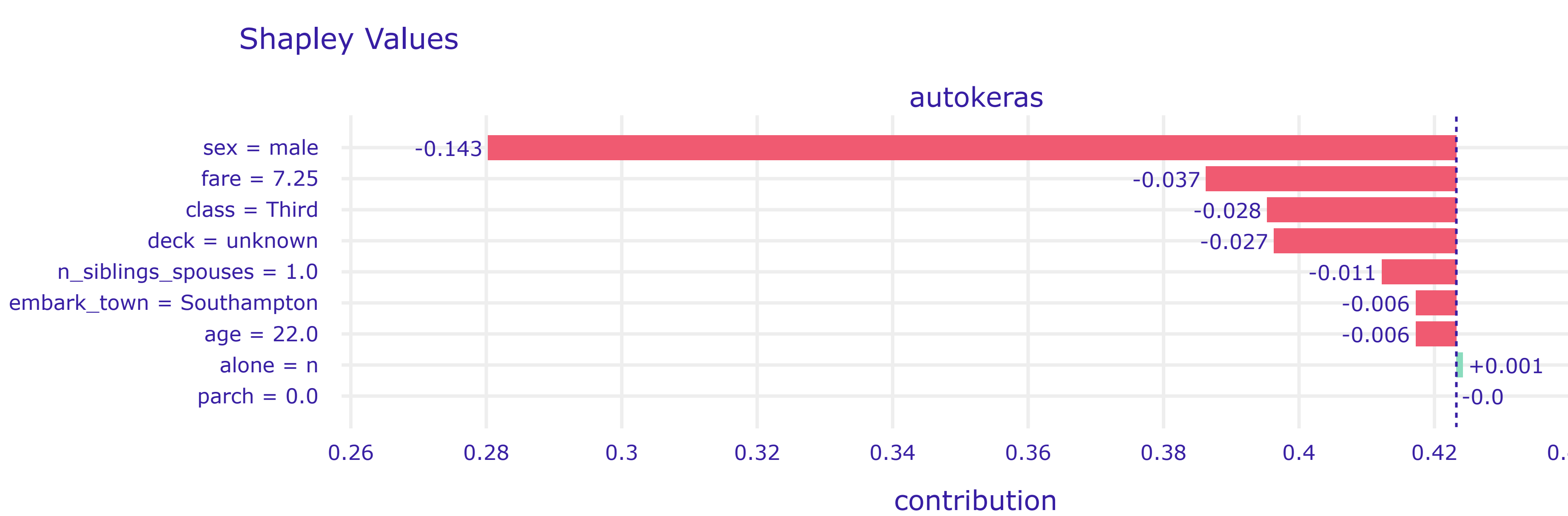


In [25]:
```
explainer_keras.model_profile().plot(variables=['sex', 'class', 'fare', 'deck',
                                                 'n_siblings_spouses', 'embark_town',
                                                 'parch', 'alone', 'age'])
```

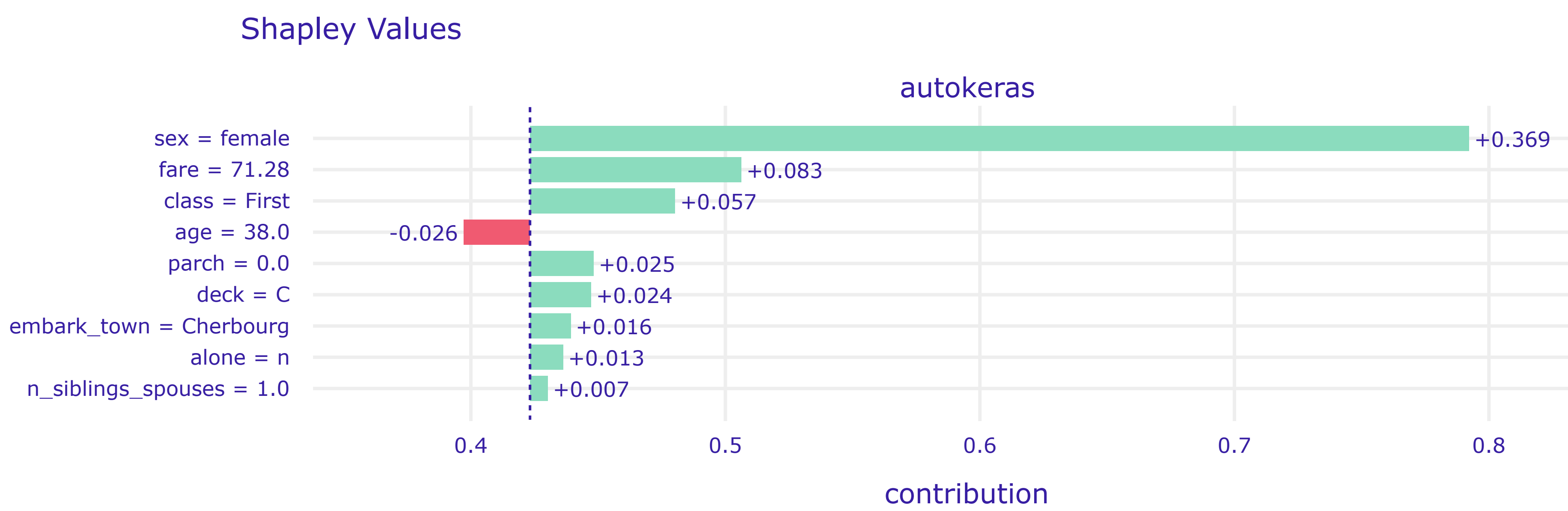
```
Calculating ceteris paribus: 100%|████████████████████████████████████████| 9/9 [00:10<00:00,  1.21s/it]
```



In [26]:
```
explainer_keras.predict_parts(X.loc[0], type='shap').plot()
```



In [27]:
```
explainer_keras.predict_parts(X.loc[1], type='shap').plot()
```



In [28]:
```
X_one_hot = pd.get_dummies(X, drop_first=True)

X_one_hot
```

Out[28]:

	age	n_siblings_spouses	parch	fare	sex_male	class_Second	class_Third	deck_B	deck_C	deck_D	deck_E	deck_F	deck_G	deck_unkn
0	22.0	1	0	7.2500	1	0	1	0	0	0	0	0	0	
1	38.0	1	0	71.2833	0	0	0	0	1	0	0	0	0	
2	26.0	0	0	7.9250	0	0	1	0	0	0	0	0	0	
3	35.0	1	0	53.1000	0	0	0	0	1	0	0	0	0	
4	28.0	0	0	8.4583	1	0	1	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
776	56.0	0	1	83.1583	0	0	0	0	1	0	0	0	0	
777	25.0	0	1	26.0000	0	1	0	0	0	0	0	0	0	
778	33.0	0	0	7.8958	1	0	1	0	0	0	0	0	0	
779	39.0	0	5	29.1250	0	0	1	0	0	0	0	0	0	
780	26.0	0	0	30.0000	1	0	0	0	1	0	0	0	0	

781 rows × 18 columns

In [29]:
```
from sklearn import tree

clf = tree.DecisionTreeClassifier(max_features = 5, max_depth = 3)

X_one_hot = pd.get_dummies(X, drop_first=True)

clf = clf.fit(X_one_hot, y)
```

In [30]:
```
df[df.sex == 'male'].survived.value_counts()
```

Out[30]:
```
0    382
1    106
Name: survived, dtype: int64
```

In [31]:
```
df[df.sex == 'female'].survived.value_counts()
```

Out[31]:
```
1    217
0     76
Name: survived, dtype: int64
```

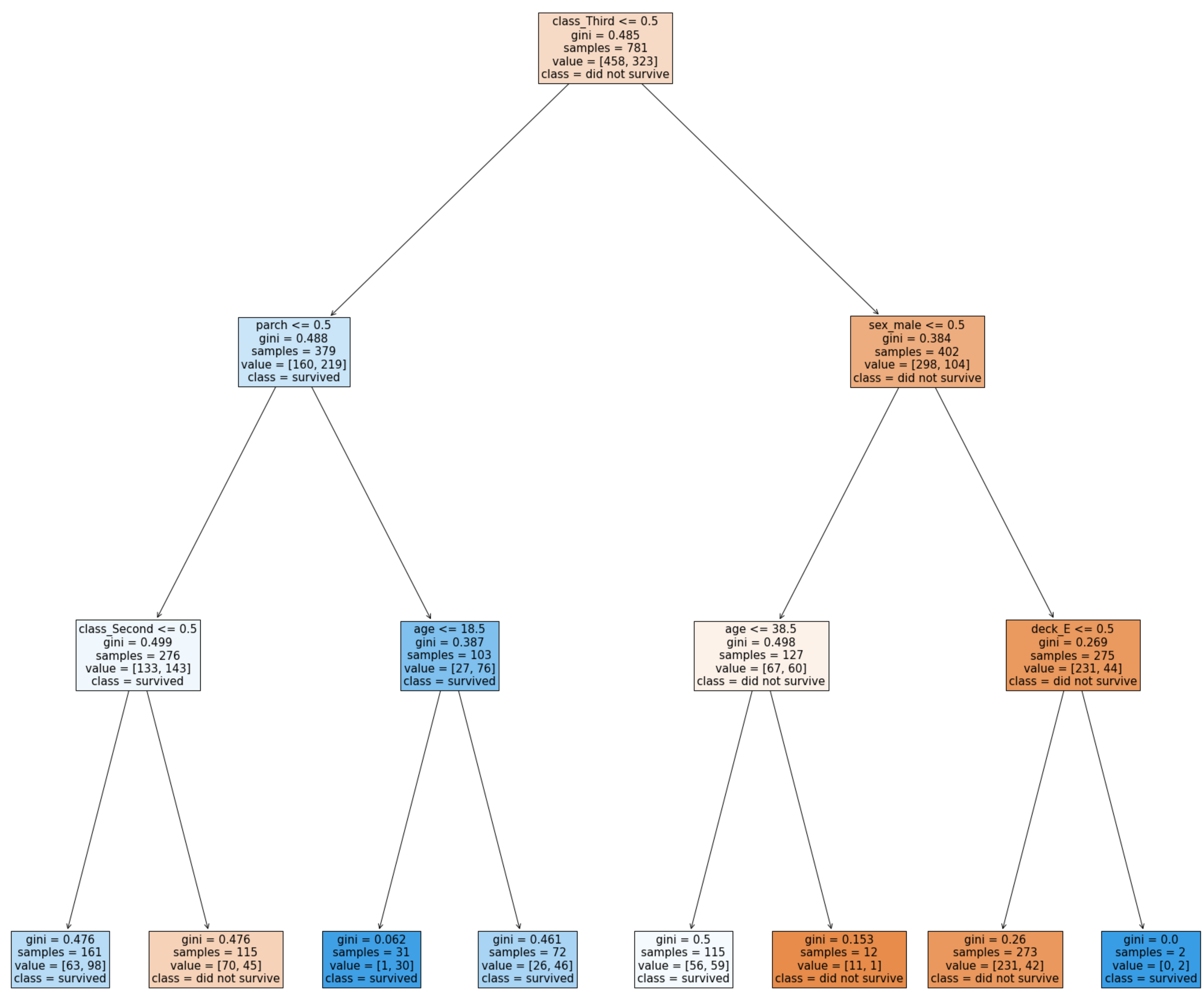
In [32]:
```
df.survived.value_counts()
```

Out[32]:
```
0    458
1    323
Name: survived, dtype: int64
```

In [33]:
```
#clf.classes_
```

In [34]:
```python
fn = list(X_one_hot.columns)

cn = ['did not survive', 'survived']

#cn = ['survived', 'did not survive']


fig, axes = plt.subplots(nrows = 1,
                         ncols = 1,
                         figsize = (30,30))
                         #dpi=500)

tree.plot_tree(clf,
               feature_names = fn,
               class_names=cn,
               filled = True, fontsize = 15)

#fig.savefig('imagename.png')
```

Out[34]:
```
[Text(837.0, 1426.95, 'class_Third <= 0.5\ngini = 0.485\nsamples = 781\nvalue = [458, 323]\nclass = did not survive'),
 Text(418.5, 1019.25, 'parch <= 0.5\ngini = 0.488\nsamples = 379\nvalue = [160, 219]\nclass = survived'),
 Text(209.25, 611.55, 'class_Second <= 0.5\ngini = 0.499\nsamples = 276\nvalue = [133, 143]\nclass = survived'),
 Text(104.625, 203.8499999999999, 'gini = 0.476\nsamples = 161\nvalue = [63, 98]\nclass = survived'),
 Text(313.875, 203.8499999999999, 'gini = 0.476\nsamples = 115\nvalue = [70, 45]\nclass = did not survive'),
 Text(627.75, 611.55, 'age <= 18.5\ngini = 0.387\nsamples = 103\nvalue = [27, 76]\nclass = survived'),
 Text(523.125, 203.8499999999999, 'gini = 0.062\nsamples = 31\nvalue = [1, 30]\nclass = survived'),
 Text(732.375, 203.8499999999999, 'gini = 0.461\nsamples = 72\nvalue = [26, 46]\nclass = survived'),
 Text(1255.5, 1019.25, 'sex_male <= 0.5\ngini = 0.384\nsamples = 402\nvalue = [298, 104]\nclass = did not survive'),
 Text(1046.25, 611.55, 'age <= 38.5\ngini = 0.498\nsamples = 127\nvalue = [67, 60]\nclass = did not survive'),
 Text(941.625, 203.8499999999999, 'gini = 0.5\nsamples = 115\nvalue = [56, 59]\nclass = survived'),
 Text(1150.875, 203.8499999999999, 'gini = 0.153\nsamples = 12\nvalue = [11, 1]\nclass = did not survive'),
 Text(1464.75, 611.55, 'deck_E <= 0.5\ngini = 0.269\nsamples = 275\nvalue = [231, 44]\nclass = did not survive'),
 Text(1360.125, 203.8499999999999, 'gini = 0.26\nsamples = 273\nvalue = [231, 42]\nclass = did not survive'),
 Text(1569.375, 203.8499999999999, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]\nclass = survived')]
```



working version

In [35]:
```
import dalex as dx

data = pd.read_csv("https://raw.githubusercontent.com/pbiecek/xai-happiness/main/happiness.csv", index_col=0)
data.head()
```

Out[35]:

	score	gdp_per_capita	social_support	healthy_life_expectancy	freedom_to_make_life_choices	generosity	perceptions_of_corruption
Afghanistan	3.203	0.350	0.517	0.361	0.000	0.158	0.025
Albania	4.719	0.947	0.848	0.874	0.383	0.178	0.027
Algeria	5.211	1.002	1.160	0.785	0.086	0.073	0.114
Argentina	6.086	1.092	1.432	0.881	0.471	0.066	0.050
Armenia	4.559	0.850	1.055	0.815	0.283	0.095	0.064

In [36]:
```
X, y = data.drop('score', axis=1), data.score
n, p = X.shape

X
```

Out[36]:

	gdp_per_capita	social_support	healthy_life_expectancy	freedom_to_make_life_choices	generosity	perceptions_of_corruption
Afghanistan	0.350	0.517	0.361	0.000	0.158	0.025
Albania	0.947	0.848	0.874	0.383	0.178	0.027
Algeria	1.002	1.160	0.785	0.086	0.073	0.114
Argentina	1.092	1.432	0.881	0.471	0.066	0.050
Armenia	0.850	1.055	0.815	0.283	0.095	0.064
...	...	...	...	...	...	...
Venezuela	0.960	1.427	0.805	0.154	0.064	0.047
Vietnam	0.741	1.346	0.851	0.543	0.147	0.073
Yemen	0.287	1.163	0.463	0.143	0.108	0.077
Zambia	0.578	1.058	0.426	0.431	0.247	0.087
Zimbabwe	0.366	1.114	0.433	0.361	0.151	0.089

156 rows × 6 columns

In [37]:
```
y
```

Out[37]:
```
Afghanistan    3.203
Albania        4.719
Algeria        5.211
Argentina      6.086
Armenia        4.559
               ...
Venezuela      4.707
Vietnam        5.175
Yemen          3.380
Zambia         4.107
Zimbabwe       3.663
Name: score, Length: 156, dtype: float64
```

In [38]:
```
#tf.random.set_seed(11)

normalizer  = tf.keras.layers.experimental.preprocessing.Normalization(input_shape=[p,])
normalizer.adapt(X.to_numpy())

model = tf.keras.Sequential([
    normalizer,
    tf.keras.Input(shape=(p,)),
    tf.keras.layers.Dense(p*2, activation='relu'),
    tf.keras.layers.Dense(p*3, activation='relu'),
    tf.keras.layers.Dense(p*2, activation='relu'),
    tf.keras.layers.Dense(p, activation='relu'),
    tf.keras.layers.Dense(1, activation='linear')
])

model.compile(
    optimizer=tf.keras.optimizers.Adam(0.001),
    loss=tf.keras.losses.mae
)

model.fit(X, y, batch_size=int(n/10), epochs=2000, verbose=False)
```

```
WARNING:tensorflow:Please add `keras.layers.InputLayer` instead of `keras.Input` to Sequential model. `keras.Input` is
intended to be used by Functional model.
```

In [39]:
```
explainer = dx.Explainer(model, X, y, label='happiness')
```

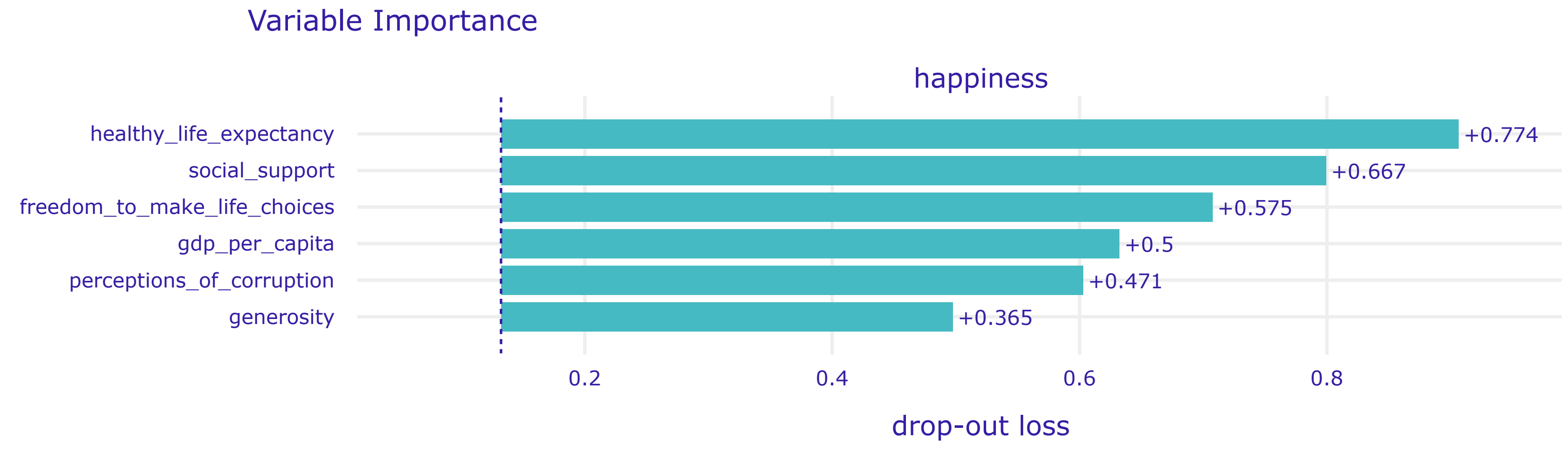
```
Preparation of a new explainer is initiated

  -> data              : 156 rows 6 cols
  -> target variable   : Parameter 'y' was a pandas.Series. Converted to a numpy.ndarray.
  -> target variable   : 156 values
  -> model_class       : tensorflow.python.keras.engine.sequential.Sequential (default)
  -> label             : happiness
  -> predict function  : <function yhat_tf_regression at 0x00000242A1487310> will be used (default)
  -> predict function  : Accepts pandas.DataFrame and numpy.ndarray.
  -> predicted values  : min = 3.0, mean = 5.41, max = 7.7
  -> model type        : regression will be used (default)
  -> residual function : difference between y and yhat (default)
  -> residuals         : min = -0.921, mean = -0.00414, max = 0.714
  -> model_info        : package tensorflow

A new explainer has been created!
```

In [40]:
```
explainer.model_performance()
```

Out[40]:

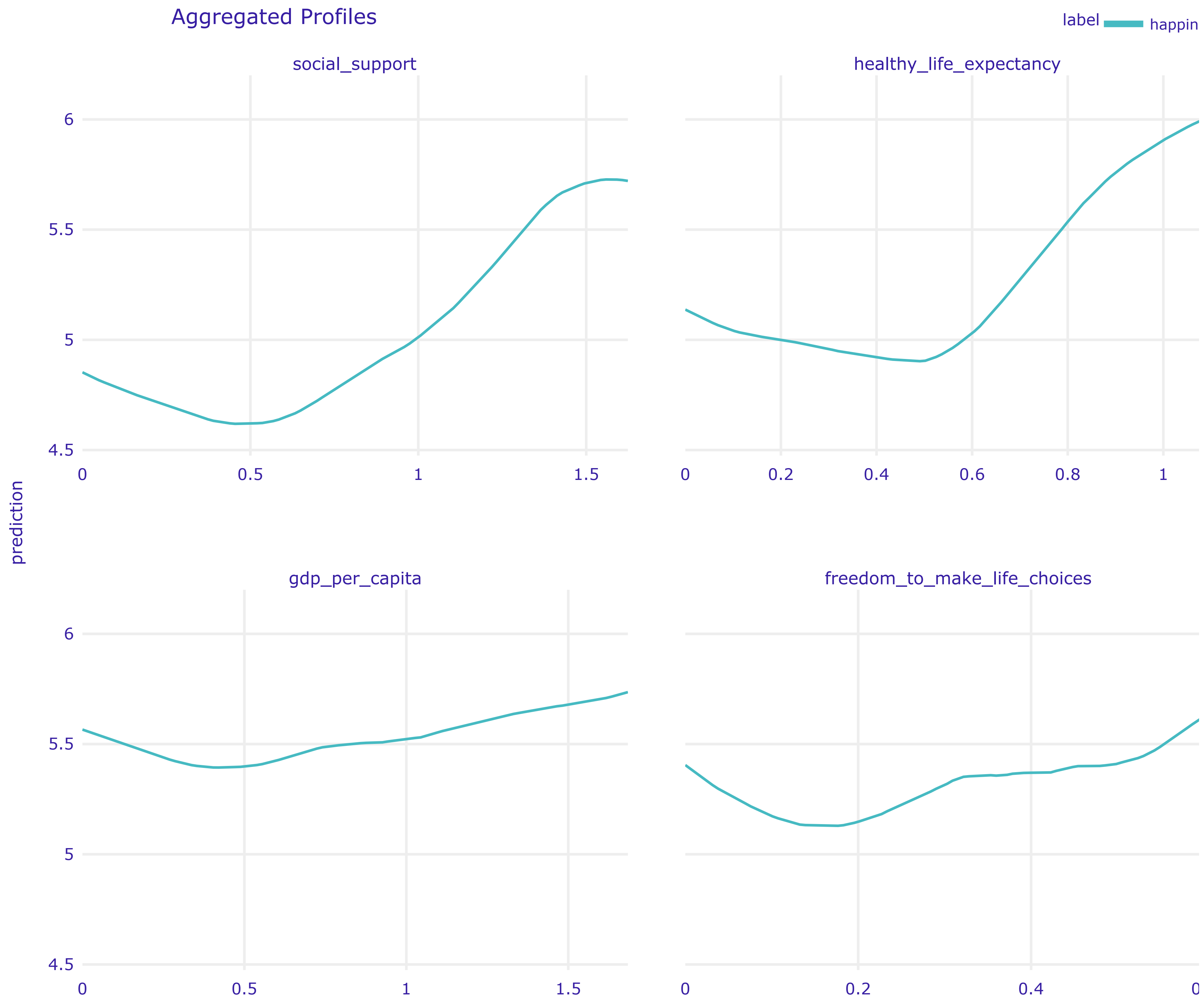
	mse	rmse	r2	mae	mad
happiness	0.01749	0.132251	0.985793	0.061181	0.029459

In [41]:
```
explainer.model_parts().plot()
```

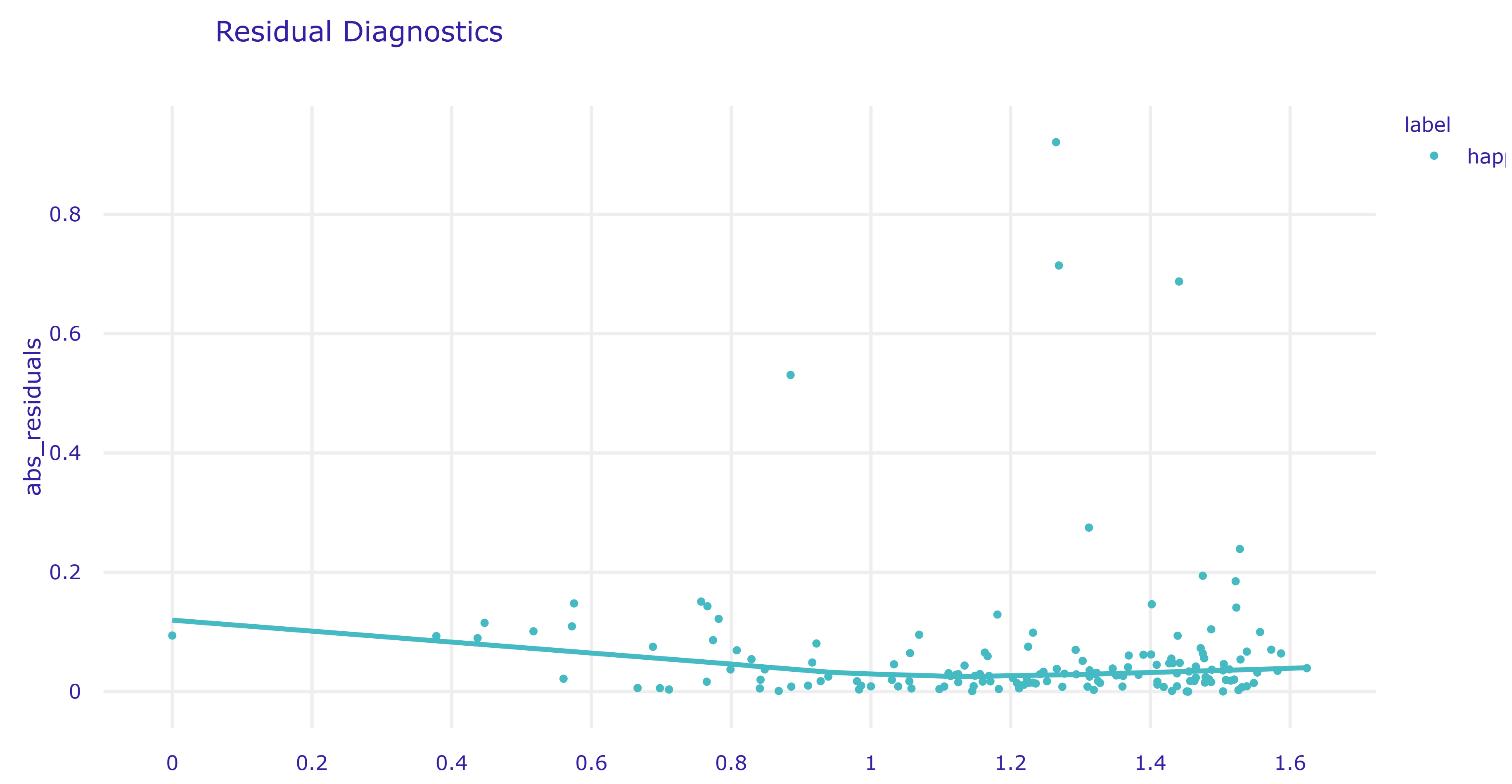


In [42]:
```
explainer.model_profile().plot(variables=['social_support', 'healthy_life_expectancy',
                                          'gdp_per_capita', 'freedom_to_make_life_choices'])
```

```
Calculating ceteris paribus: 100%|██████████████████████████████████████████| 6/6 [00:03<00:00,  1.91it/s]
```



In [43]:
```
explainer.model_diagnostics().plot(variable='social_support', yvariable="abs_residuals", marker_size=5, line_width=3)
```



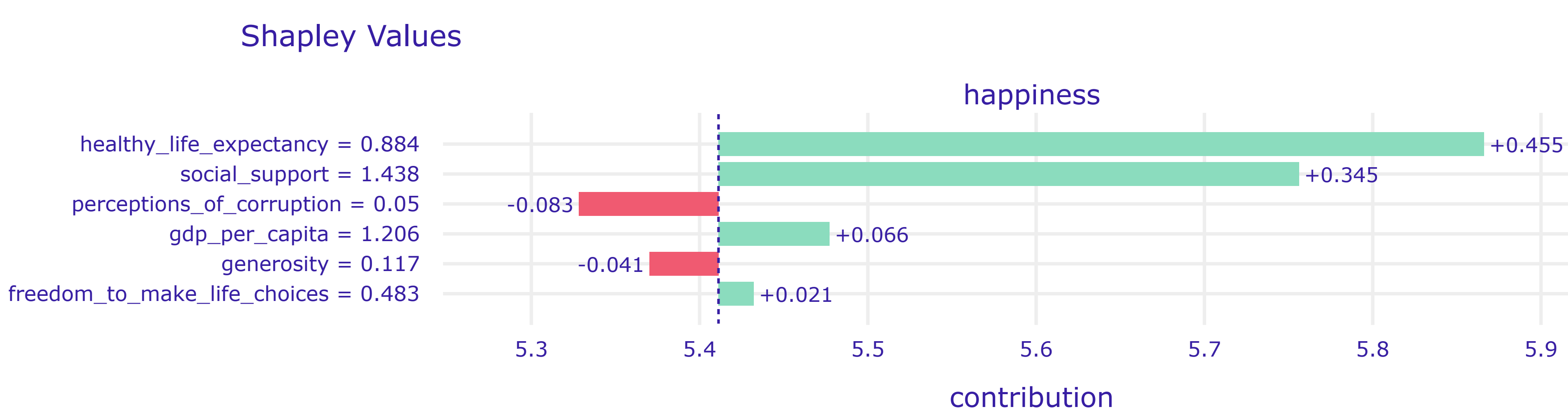
```
In [44]: explainer.model_diagnostics().result
```

Out[44]:

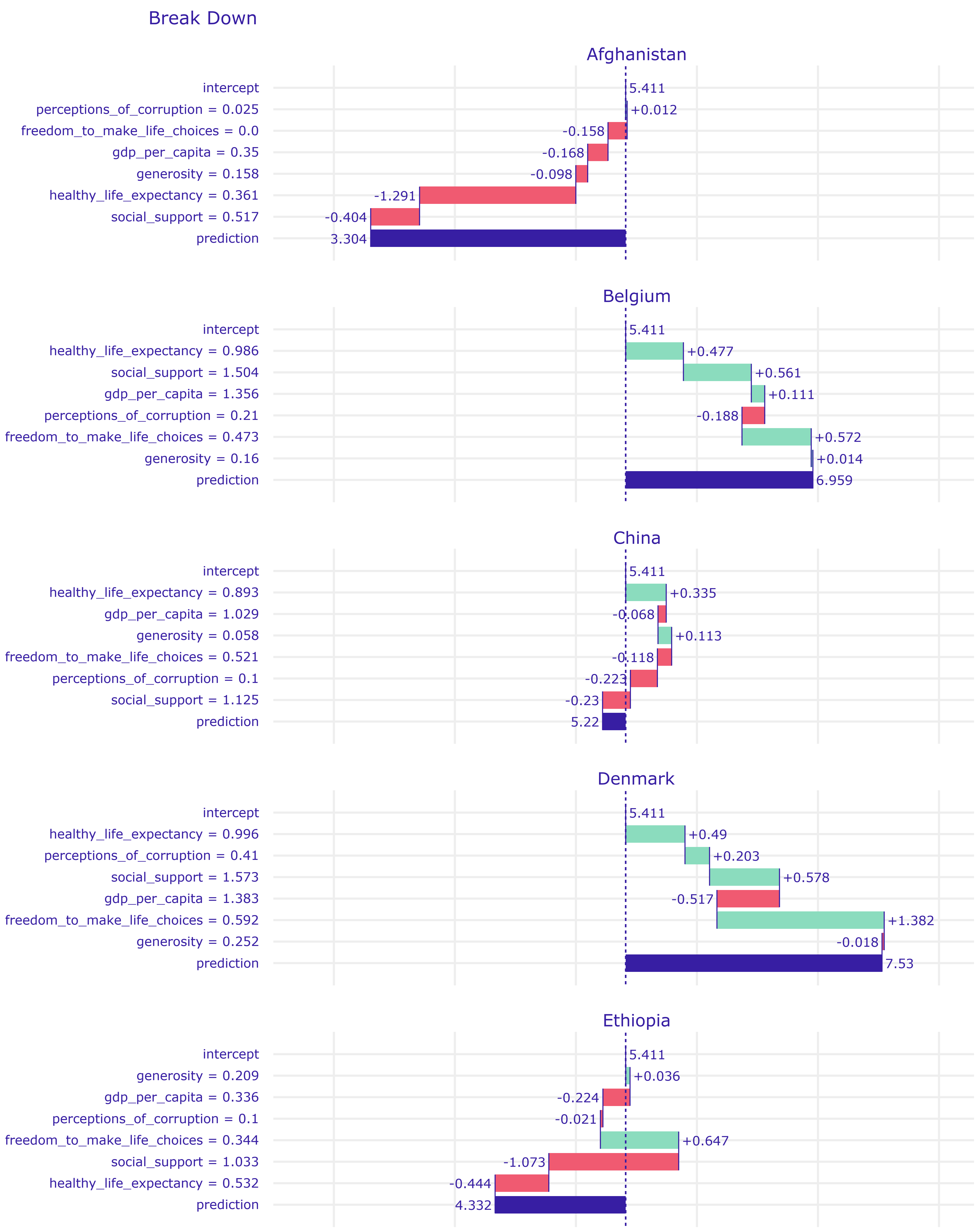
	gdp_per_capita	social_support	healthy_life_expectancy	freedom_to_make_life_choices	generosity	perceptions_of_corruption	y	y
Afghanistan	0.350	0.517	0.361	0.000	0.158	0.025	3.203	3.30
Albania	0.947	0.848	0.874	0.383	0.178	0.027	4.719	4.68
Algeria	1.002	1.160	0.785	0.086	0.073	0.114	5.211	5.22
Argentina	1.092	1.432	0.881	0.471	0.066	0.050	6.086	6.13
Armenia	0.850	1.055	0.815	0.283	0.095	0.064	4.559	4.57
...	...	...	...	...	...	...	...	
Venezuela	0.960	1.427	0.805	0.154	0.064	0.047	4.707	4.75
Vietnam	0.741	1.346	0.851	0.543	0.147	0.073	5.175	5.21
Yemen	0.287	1.163	0.463	0.143	0.108	0.077	3.380	3.44
Zambia	0.578	1.058	0.426	0.431	0.247	0.087	4.107	4.10
Zimbabwe	0.366	1.114	0.433	0.361	0.151	0.089	3.663	3.68

156 rows × 12 columns

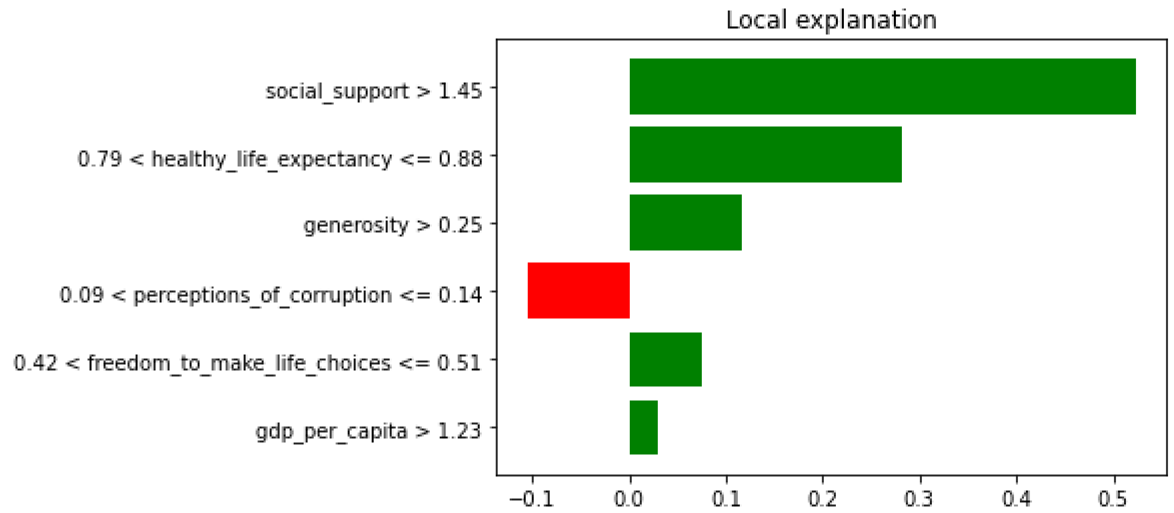
```
In [45]: explainer.predict_parts(X.loc['Poland'], type='shap').plot()
```



In [46]:
```
pp_list = []

for country in ['Afghanistan', 'Belgium', 'China', 'Denmark', 'Ethiopia']:
    pp = explainer.predict_parts(X.loc[country], type='break_down')
    pp.result.label = country
    pp_list += [pp]

pp_list[0].plot(pp_list[1::], min_max=[2.5, 8.5])
```



```
In [47]: lime_explanation = explainer.predict_surrogate(X.loc['United States'], mode='regression')

lime_explanation.plot()
```




```
In [48]: lime_explanation.result
```

Out[48]:

	variable	effect
0	social_support > 1.45	0.523263
1	0.79 < healthy_life_expectancy <= 0.88	0.280861
2	generosity > 0.25	0.117167
3	0.09 < perceptions_of_corruption <= 0.14	-0.105348
4	0.42 < freedom_to_make_life_choices <= 0.51	0.075624
5	gdp_per_capita > 1.23	0.030089

```
In [49]: surrogate_model = explainer.model_surrogate(max_vars=4, max_depth=3)
surrogate_model.performance
```

Out[49]:

	mse	rmse	r2	mae	mad
DecisionTreeRegressor	0.187769	0.433323	0.832941	0.353901	0.32507

```
In [50]: surrogate_model.plot()
```