Download and import libraries

```
!pip install eli5 shap
from IPython.core.display import display, HTML
import regex as re
import eli5
import lightgbm
import pandas as pd
import shap
import sklearn

import xgboost as xgb
from sklearn.model_selection import train_test_split
import lightgbm as lgb
```

```
Requirement already satisfied: eli5 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: shap in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: attrs>17.1.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: jinja2>=3.0.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: scikit-learn>=1.6.0 in /usr/local/lib/python3.1
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: tabulate>=0.7.7 in /usr/local/lib/python3.11/di
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.11/dis
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.11/dist
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/py
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dis
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-pack
```

Tutorial

Patch to match style consistency

```
def patch_format_value(value):
    # type: (Optional[float]) -> str
    return value
eli5.formatters.utils.format_value = patch_format_value
eli5.utils.format_value = patch_format_value
format_value = patch_format_value

from eli5.formatters.html import format_hsl
from eli5.formatters.html import remaining_weight_color_hsl
from eli5.formatters.html import _format_feature
from eli5.formatters.html import _format_decision_tree
from eli5.formatters.html import weight_color_hsl
from jinja2 import Environment, PackageLoader
from __future__ import absolute_import
from itertools import groupby
from typing import List, Optional, Tuple

import numpy as np

from eli5 import _graphviz
from eli5.base import (Explanation, TargetExplanation, FeatureWeights,
                       FeatureWeight)
from eli5.utils import max_or_0
from eli5.formatters.utils import (
    format_signed, format_weight, has_any_values_for_weights,
    replace_spaces, should_highlight_spaces)
from eli5.formatters import fields
from eli5.formatters.features import FormattedFeatureName
from eli5.formatters.trees import tree2text
from eli5.formatters.text_helpers import prepare_weighted_spans, PreparedWeighted
from eli5.formatters.html import render_targets_weighted_spans
from eli5.formatters.html import get_weight_range
from eli5.formatters.html import html_escape
from eli5.formatters.html import template_env

#template_env = Environment(
#    loader=PackageLoader('eli5', 'templates'),
#    extensions=['jinja2.ext.with_'])
template_env.globals.update(dict(zip=zip, numpy=np))
template_env.filters.update(dict(
    weight_color=lambda w, w_range: format_hsl(weight_color_hsl(w, w_range)),
    remaining_weight_color=lambda ws, w_range, pos_neg:
        format_hsl(remaining_weight_color_hsl(ws, w_range, pos_neg)),
    format_feature=lambda f, w, hl: _format_feature(f, w, hl_spaces=hl),
    format_value=patch_format_value,
    format_weight=format_weight,
    format_decision_tree=lambda tree: _format_decision_tree(tree),
))

WEIGHT_HELP = '''\
Feature weights. Note that weights do not account for feature value scales,
so if feature values have different scales, features with highest weights
might not be the most important.\
'''.replace('\n', ' ')
CONTRIBUTION_HELP = '''\
Feature contribution already accounts for the feature value
(for linear models, contribution = weight * feature value), and the sum
of feature contributions is equal to the score or, for some classifiers,
to the probability. Feature values are shown if "show_feature_values" is True.\
'''.replace('\n', ' ')

def sortkey(val):
    return val.weight

def patch_format_as_html(explanation,  # type: Explanation
                   existing_values = None,
                   feature_values = None,
                   new_probability = None,
                   new_base_value = None,
                   include_styles=True,  # type: bool
                   force_weights=True,  # type: bool
                   show=fields.ALL,
                   preserve_density=None,  # type: Optional[bool]
                   highlight_spaces=None,  # type: Optional[bool]
                   horizontal_layout=True,  # type: bool
                   show_feature_values=False  # type: bool
                   ):
    # type: (...) -> str

    """ Format explanation as html.
    Most styles are inline, but some are included separately in <style> tag,
    you can omit them by passing ``include_styles=False`` and call
    ``format_html_styles`` to render them separately (or just omit them).
    With ``force_weights=False``, weights will not be displayed in a table for
    predictions where it is possible to show feature weights highlighted
    in the document.
    If ``highlight_spaces`` is None (default), spaces will be highlighted in
    feature names only if there are any spaces at the start or at the end of the
    feature. Setting it to True forces space highlighting, and setting it to
    False turns it off.
    If ``horizontal_layout`` is True (default), multiclass classifier
    weights are laid out horizontally.
    If ``show_feature_values`` is True, feature values are shown if present.
    Default is False.
    """
    template = template_env.get_template('explain.html')
    if highlight_spaces is None:
        highlight_spaces = should_highlight_spaces(explanation)
    targets = explanation.targets or []
    if len(targets) == 1:
        horizontal_layout = False
    explaining_prediction = has_any_values_for_weights(explanation)
    show_feature_values = show_feature_values and explaining_prediction

    rendered_weighted_spans = render_targets_weighted_spans(
        targets, preserve_density)
    weighted_spans_others = [
        t.weighted_spans.other if t.weighted_spans else None for t in targets]



    #diff = new_probability - new_base_value



    val_list = {}
    for myint in range(len(explanation.targets[0].feature_weights.pos)):
      if explanation.targets[0].feature_weights.pos[myint].feature == '<BIAS>':
        explanation.targets[0].feature_weights.pos[myint].feature = 'base value'
        bias_score = explanation.targets[0].feature_weights.pos[myint].weight
      else:
        val_list[explanation.targets[0].feature_weights.pos[myint].feature] = exp
    for myint in range(len(explanation.targets[0].feature_weights.neg)):
      if explanation.targets[0].feature_weights.neg[myint].feature == '<BIAS>':
        explanation.targets[0].feature_weights.neg[myint].feature = 'base value'
        bias_score = explanation.targets[0].feature_weights.neg[myint].weight
      else:
         val_list[explanation.targets[0].feature_weights.neg[myint].feature] = ex

    diff = abs(bias_score - explanation.targets[0].score)
    to_sum_val_list = [val_list[key] for key in val_list.keys()]
    #existing_values_abs = [abs(myval) for myval in existing_values_list]
    for myint in range(len(explanation.targets[0].feature_weights.pos)):
      for key in existing_values.keys():
        if explanation.targets[0].feature_weights.pos[myint].feature == key:
         #explanation.targets[0].feature_weights.pos[myint].weight = existing_val
         explanation.targets[0].feature_weights.pos[myint].value = feature_values
         #print(explanation.targets[0].feature_weights.pos[myint])

    explanation.targets[0].feature_weights.pos.sort(key=sortkey, reverse=True)

    for myint in range(len(explanation.targets[0].feature_weights.neg)):
      for key in existing_values.keys():
        if explanation.targets[0].feature_weights.neg[myint].feature == key:
         #explanation.targets[0].feature_weights.neg[myint].weight = float('%.3f'
         explanation.targets[0].feature_weights.neg[myint].value = feature_values

    explanation.targets[0].feature_weights.neg.sort(key=sortkey)



    explanation.targets[0].proba = new_probability
    return template.render(
        include_styles=include_styles,
        force_weights=force_weights,
        target_table_styles=
        'border-collapse: collapse; border: none; margin-top: 0em; table-layout:
        tr_styles='border: none;',
        # Weight (th and td)
        td1_styles='padding: 0 1em 0 0.5em; text-align: right; border: none;',
        # N more positive/negative
        tdm_styles='padding: 0 0.5em 0 0.5em; text-align: center; border: none; '
                   'white-space: nowrap;',
        # Feature (th and td)
        td2_styles='padding: 0 0.5em 0 0.5em; text-align: left; border: none;',
        # Value (th and td)
        td3_styles='padding: 0 0.5em 0 1em; text-align: right; border: none;',
        horizontal_layout_table_styles=
        'border-collapse: collapse; border: none; margin-bottom: 1.5em;',
        horizontal_layout_td_styles=
        'padding: 0px; border: 1px solid black; vertical-align: top;',
        horizontal_layout_header_styles=
        'padding: 0.5em; border: 1px solid black; text-align: center;',
        show=show,
        expl=explanation,
        hl_spaces=highlight_spaces,
        horizontal_layout=horizontal_layout,
        any_weighted_spans=any(t.weighted_spans for t in targets),
        feat_imp_weight_range=max_or_0(
            abs(fw.weight) for fw in explanation.feature_importances.importances)
        if explanation.feature_importances else 0,
        target_weight_range=max_or_0(
            get_weight_range(t.feature_weights) for t in targets
        if t.feature_weights is not None),
        other_weight_range=max_or_0(
            get_weight_range(other)
            for other in weighted_spans_others if other),
        targets_with_weighted_spans=list(
            zip(targets, rendered_weighted_spans, weighted_spans_others)),
        show_feature_values=show_feature_values,
        weights_table_span=3 if show_feature_values else 2,
        explaining_prediction=explaining_prediction,
        weight_help=html_escape(WEIGHT_HELP),
        contribution_help=html_escape(CONTRIBUTION_HELP),
    )

eli5.format_as_html = patch_format_as_html
```

Set up tutorial examples

Start by training the "should you bring an umbrella?" model

```
preX = pd.read_csv("Umbrella.csv")
preX = preX.sample(frac=1)
X_display = preX.iloc[:,:-1]
y_display = preX.iloc[:,-1]

PRECIPITATION = {
    "none": 0,
    "drizzle": 1,
    "rain": 2,
    "snow": 3,
    "sleet": 4,
    "hail": 5
}

y = y_display
X = X_display
X = X.replace({"Precipitation":PRECIPITATION})
```

```
X_train = X.iloc[:300]
y_train = y.iloc[:300]

X_test = X.iloc[300:]
y_test = y.iloc[300:]

d_train = lightgbm.Dataset(X_train, label=y_train)
d_test = lightgbm.Dataset(X_test, label=y_test)

params = {
    "max_bin": 512,
    "learning_rate": 0.05,
    "boosting_type": "gbdt",
    "objective": "binary",
    "metric": "binary_logloss",
    "num_leaves": 10,
    "verbose": -1,
    "min_data": 100,
    "boost_from_average": True,
    "keep_training_booster": True
}

#model = lgb.train(params, d_train, 10000, valid_sets=[d_test]) #early_stopping_r
model = lightgbm.LGBMClassifier(max_bin= 512,
    learning_rate= 0.05,
    boosting_type= "gbdt",
    objective= "binary",
    metric= "binary_logloss",
    num_leaves= 10,
    verbose= -1,
    min_data= 100,
    boost_from_average= True)
model.fit(X_train, y_train)
```

```
<ipython-input-81-59731d8556ad>:17: FutureWarning: Downcasting behavior in `re
  X = X.replace({"Precipitation":PRECIPITATION})
```

LGBMClassifier

```
LGBMClassifier(boost_from_average=True, learning_rate=0.05, max_bin=512,
               metric='binary_logloss', min_data=100, num_leaves=10,
               objective='binary', verbose=-1)
```

Find the location of one of the two tutorial examples

```
print(X.loc[(X['Precipitation'] == 5) & (X['Temperature'] == 23) & (X['Wind(mph)']
print(X.loc[(X['Precipitation'] == 0) & (X['Temperature'] == 70) & (X['Wind(mph)']
theloc = X.index.get_loc(330)
```

```
     Precipitation  Temperature  Wind(mph)
330              5           23         10
    Precipitation  Temperature  Wind(mph)
96              0           70         30
```

Generate a tutorial explanation

```
doc = X.iloc[theloc]
doc3 = X_display.iloc[theloc]
targets = [0,1]
#+1.433 base value  1
explanation = eli5.explain_prediction(model, doc, target_names=['NO','YES'], targ

#existing_values = doc2, feature_values = doc3, new_probability = 0.861, new_base
doc2 = pd.Series({'Wind(mph)':0.29, 'Temperature':-0.2, 'Precipitation':-0.08})
explanation = eli5.explain_prediction(model, doc, target_names=['NO','YES'], targ
the_html = eli5.format_as_html(explanation, existing_values = doc2, feature_value
the_html = the_html.replace("1.0","1")
the_html = the_html.replace("probability","prediction")
the_html = the_html.replace("Contribution<sup>?</sup>","Contribution")
the_html = the_html.replace("<BIAS>","base value")
display(HTML(the_html))
#eli5.formatters.html.format_as_html(prediction, show_feature_values=True)
#pred = eli5.explain_prediction(model, doc, target_names=['Yes','No'])
#eli5.formatters.html.format_as_html(pred, show_feature_values=True)
```

Explained as: decision paths

```
Features with largest coefficients.

Feature weights are calculated by following decision paths in trees
of an ensemble. Each leaf has an output score, and expected scores can also be
assigned to parent nodes. Contribution of one feature on the decision path
is how much expected score changes from parent to child. Weights of all
features sum to the output score of the estimator.

Caveats:
1. Feature weights just show if the feature contributed positively or
   negatively to the final score, and does not show how increasing or
   decreasing the feature value will change the prediction.
2. In some cases, feature weight can be close to zero for an important feature
   For example, in a single tree that computes XOR function, the feature at th
   top of the tree will have zero weight because expected scores for both
   branches are equal, so decision at the top feature does not change the
   expected score. For an ensemble predicting XOR functions it might not be
   a problem, but it is not reliable if most trees happen to choose the same
   feature at the top.
```

y=YES (prediction **0.861**, score **1.538**) top features

Contribution	Feature	Value
+1.319	Temperature	23
+0.696	base value	1
+0.422	Precipitation	hail
-0.899	Wind(mph)	10

Loan Instances

Updated patch for style consistency

```
def patch_format_value(value):
    # type: (Optional[float]) -> str
    return value
eli5.formatters.utils.format_value = patch_format_value
eli5.utils.format_value = patch_format_value
format_value = patch_format_value

from eli5.formatters.html import format_hsl
from eli5.formatters.html import remaining_weight_color_hsl
from eli5.formatters.html import _format_feature
from eli5.formatters.html import _format_decision_tree
from eli5.formatters.html import weight_color_hsl
from jinja2 import Environment, PackageLoader
from __future__ import absolute_import
from itertools import groupby
from typing import List, Optional, Tuple

import numpy as np

from eli5 import _graphviz
from eli5.base import (Explanation, TargetExplanation, FeatureWeights,
                       FeatureWeight)
from eli5.utils import max_or_0
from eli5.formatters.utils import (
    format_signed, format_weight, has_any_values_for_weights,
    replace_spaces, should_highlight_spaces)
from eli5.formatters import fields
from eli5.formatters.features import FormattedFeatureName
from eli5.formatters.trees import tree2text
from eli5.formatters.text_helpers import prepare_weighted_spans, PreparedWeighted
from eli5.formatters.html import render_targets_weighted_spans
from eli5.formatters.html import get_weight_range
from eli5.formatters.html import html_escape
from eli5.formatters.html import template_env

#template_env = Environment(
#    loader=PackageLoader('eli5', 'templates'),
#    extensions=['jinja2.ext.with_'])
template_env.globals.update(dict(zip=zip, numpy=np))
template_env.filters.update(dict(
    weight_color=lambda w, w_range: format_hsl(weight_color_hsl(w, w_range)),
    remaining_weight_color=lambda ws, w_range, pos_neg:
        format_hsl(remaining_weight_color_hsl(ws, w_range, pos_neg)),
    format_feature=lambda f, w, hl: _format_feature(f, w, hl_spaces=hl),
    format_value=patch_format_value,
    format_weight=format_weight,
    format_decision_tree=lambda tree: _format_decision_tree(tree),
))

WEIGHT_HELP = '''\
Feature weights. Note that weights do not account for feature value scales,
so if feature values have different scales, features with highest weights
might not be the most important.\
'''.replace('\n', ' ')
```

```
CONTRIBUTION_HELP = '''\
Feature contribution already accounts for the feature value
(for linear models, contribution = weight * feature value), and the sum
of feature contributions is equal to the score or, for some classifiers,
to the probability. Feature values are shown if "show_feature_values" is True.\
'''.replace('\n', ' ')

def sortkey(val):
    return val.weight

def patch_format_as_html(explanation,  # type: Explanation
                   existing_values = None,
                   feature_values = None,
                   new_probability = None,
                   include_styles=True,  # type: bool
                   force_weights=True,  # type: bool
                   show=fields.ALL,
                   preserve_density=None,  # type: Optional[bool]
                   highlight_spaces=None,  # type: Optional[bool]
                   horizontal_layout=True,  # type: bool
                   show_feature_values=False  # type: bool
                   ):
    # type: (...) -> str

    """ Format explanation as html.
    Most styles are inline, but some are included separately in <style> tag,
    you can omit them by passing ``include_styles=False`` and call
    ``format_html_styles`` to render them separately (or just omit them).
    With ``force_weights=False``, weights will not be displayed in a table for
    predictions where it is possible to show feature weights highlighted
    in the document.
    If ``highlight_spaces`` is None (default), spaces will be highlighted in
    feature names only if there are any spaces at the start or at the end of the
    feature. Setting it to True forces space highlighting, and setting it to
    False turns it off.
    If ``horizontal_layout`` is True (default), multiclass classifier
    weights are laid out horizontally.
    If ``show_feature_values`` is True, feature values are shown if present.
    Default is False.
    """
    template = template_env.get_template('explain.html')
    if highlight_spaces is None:
        highlight_spaces = should_highlight_spaces(explanation)
    targets = explanation.targets or []
    if len(targets) == 1:
      horizontal_layout = False
    explaining_prediction = has_any_values_for_weights(explanation)
    show_feature_values = show_feature_values and explaining_prediction

    rendered_weighted_spans = render_targets_weighted_spans(
      targets, preserve_density)
    weighted_spans_others = [
      t.weighted_spans.other if t.weighted_spans else None for t in targets]

    for myint in range(len(explanation.targets[0].feature_weights.neg)):
     if explanation.targets[0].feature_weights.neg[myint].feature == '<BIAS>':
      explanation.targets[0].feature_weights.neg[myint].feature = 'base value'
      explanation.targets[0].feature_weights.neg[myint].weight = -0.001
      bias_score = explanation.targets[0].feature_weights.neg[myint].weight
      bias_feature = explanation.targets[0].feature_weights.neg[myint]

    existing_values_list = [existing_values[key] for key in existing_values.keys(
    the_score = explanation.targets[0].score
    diff = abs(bias_score - the_score)
    print("diff")
    print(diff)
    existing_values_abs = [abs(existing_values[key]) for key in existing_values.k
    norm = [float(i)/abs(sum(existing_values_abs)) for i in existing_values]
    new_pos = []
    new_neg = []
    for myint in range(len(explanation.targets[0].feature_weights.pos)):
     for key in existing_values.keys():
      if explanation.targets[0].feature_weights.pos[myint].feature == key:
       new_feat = explanation.targets[0].feature_weights.pos[myint]
       new_feat.weight = float('%.4f'%(diff * (float(existing_values[key])/abs
       new_feat.value = feature_values[key]
       if new_feat.weight > 0:
        new_pos.append(new_feat)
       else:
        new_neg.append(new_feat)
       #print(explanation.targets[0].feature_weights.pos[myint])

    if bias_feature.weight > 0:
     new_pos.append(bias_feature)
    else:
     new_neg.append(bias_feature)

    for myint in range(len(explanation.targets[0].feature_weights.neg)):
     for key in existing_values.keys():
      if explanation.targets[0].feature_weights.neg[myint].feature == key:
          new_feat = explanation.targets[0].feature_weights.neg[myint]
          new_feat.weight = float('%.4f'%(diff * (float(existing_values[key])/abs
          new_feat.value = feature_values[key]
          if new_feat.weight > 0:
            new_pos.append(new_feat)
          else:
            new_neg.append(new_feat)

    explanation.targets[0].feature_weights.pos = new_pos
    explanation.targets[0].feature_weights.neg = new_neg
    explanation.targets[0].feature_weights.pos.sort(key=sortkey, reverse=True)
    explanation.targets[0].feature_weights.neg.sort(key=sortkey)

    explanation.targets[0].proba = new_probability
    return template.render(
        include_styles=include_styles,
        force_weights=force_weights,
        target_table_styles=
        'border-collapse: collapse; border: none; margin-top: 0em; table-layout:
        tr_styles='border: none;',
        # Weight (th and td)
        td1_styles='padding: 0 1em 0 0.5em; text-align: right; border: none;',
        # N more positive/negative
        tdm_styles='padding: 0 0.5em 0 0.5em; text-align: center; border: none; '
                   'white-space: nowrap;',
        # Feature (th and td)
        td2_styles='padding: 0 0.5em 0 0.5em; text-align: left; border: none;',
        # Value (th and td)
        td3_styles='padding: 0 0.5em 0 1em; text-align: right; border: none;',
        horizontal_layout_table_styles=
        'border-collapse: collapse; border: none; margin-bottom: 1.5em;',
        horizontal_layout_td_styles=
        'padding: 0px; border: 1px solid black; vertical-align: top;',
        horizontal_layout_header_styles=
        'padding: 0.5em; border: 1px solid black; text-align: center;',
        show=show,
        expl=explanation,
        hl_spaces=highlight_spaces,
        horizontal_layout=horizontal_layout,
        any_weighted_spans=any(t.weighted_spans for t in targets),
        feat_imp_weight_range=max_or_0(
            abs(fw.weight) for fw in explanation.feature_importances.importances)
        if explanation.feature_importances else 0,
        target_weight_range=max_or_0(
            get_weight_range(t.feature_weights) for t in targets
        if t.feature_weights is not None),
        other_weight_range=max_or_0(
            get_weight_range(other)
            for other in weighted_spans_others if other),
        targets_with_weighted_spans=list(
            zip(targets, rendered_weighted_spans, weighted_spans_others)),
        show_feature_values=show_feature_values,
        weights_table_span=3 if show_feature_values else 2,
        explaining_prediction=explaining_prediction,
        weight_help=html_escape(WEIGHT_HELP),
        contribution_help=html_escape(CONTRIBUTION_HELP),
    )

eli5.format_as_html = patch_format_as_html
```

Edit and prepare the dataset

```
# load dataset
X,y = shap.datasets.adult()
X_display,y_display = shap.datasets.adult(display=True)

EDUCATION_NUM = {
    16.0: "Doctorate",
    15.0: "Prof. School",
    14.0: "Masters",
    13.0: "Bachelors",
    12.0: "Some College",
    11.0: "Associate", #Assoc-acdm
    10.0: "Vocational", #Assoc-voc
    9.0: "HS grad",
    8.0: "12th",
    7.0: "11th",
    6.0: "10th",
    5.0: "9th",
    4.0: "7th-8th",
    3.0: "5th-6th",
    2.0: "1st-4th",
    1.0: "Preschool"
}

OCCUPATION_NUM = {
    "Tech-support": "Tech Support",
    "Craft-repair": "Craft/Repair",
    "Other-service": "Other Service",
    "Sales": "Sales",
    "Exec-managerial": "Exec. Managerial",
    "Prof-specialty": "Prof. Specialty",
    "Handlers-cleaners": "Handler/Cleaner",
    "Machine-op-inspct": "Machine Op. Inspector",
    "Adm-clerical": "Admin. Clerical",
    "Farming-fishing": "Farming/Fishing",
    "Transport-moving": "Transport/Moving",
    "Priv-house-serv": "Private House Service",
    "Protective-serv": "Protective Service",
    "Armed-Forces": "Armed Forces"

}
X_display = X_display.replace({"Education-Num":EDUCATION_NUM})
X_display = X_display.replace({"Occupation":OCCUPATION_NUM})
X = X.rename(columns={"Education-Num": "Education"})
X_display = X_display.rename(columns={"Education-Num": "Education"})#, "Hours per

X = X.drop(['Capital Loss', 'Capital Gain', 'Race', 'Relationship', 'Country', 'W
X_display = X_display.drop(['Capital Loss', 'Capital Gain', 'Race', 'Relationship

# create a train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s
d_train = lgb.Dataset(X_train, label=y_train)
d_test = lgb.Dataset(X_test, label=y_test)
```

Train the model

```
params = {
    "max_bin": 512,
    "learning_rate": 0.05,
    "boosting_type": "gbdt",
    "objective": "binary",
    "metric": "binary_logloss",
    "num_leaves": 10,
    "verbose": -1,
    "min_data": 100,
    'objective':'multi:softprob',
    "boost_from_average": True
}

params_xgb={
    'base_score':0.5,
    'learning_rate':0.05,
    'max_depth':5,
    'min_child_weight':100,
    'n_estimators':200,
    'num_class': 2,
    'nthread':-1,
    'objective':'multi:softprob',
    'seed':2018,
    'eval_metric':'auc'
}

model = lgb.LGBMClassifier(max_bin= 512,
    learning_rate= 0.05,
    boosting_type= "gbdt",
    objective= "binary",
    metric= "binary_logloss",
    num_leaves= 10,
    verbose= -1,
    min_data= 100,
    boost_from_average= True)
model.fit(X_train, y_train)
```

▾ LGBMClassifier

```
LGBMClassifier(boost_from_average=True, learning_rate=0.05, max_bin=512,
               metric='binary_logloss', min_data=100, num_leaves=10,
               objective='binary', verbose=-1)
```

Our 7 loan application instances

```
#val = 610 # Woman Side-by-side
#val = 11116 # Man Side-by-side
#val = 32353 # Man 3
#val = 217 # Man 2
#val = 15040 # Man 1
#val = 32429 # Woman 3
val = 32556 # Woman 2
#val = 91#91 # Woman 1

theloc = val
```

Generate Eli5 Explanation

```
#shap_values_standin0 = pd.Series({'pred' : 0.1159,'Age': 0.0307, 'Education': -0.0
shap_values_standin0 = pd.Series({'pred' : 0.0775,'Age': -0.14, 'Education': 0.0416
#shap_values_standin0 = pd.Series({'pred' : 0.6474,'Age': 0.1209, 'Education': 0.30
#shap_values_standin0 = pd.Series({'pred' : 0.0601,'Age': -0.2119, 'Education': 0.0
#shap_values_standin0 = pd.Series({'pred' : 0.67,'Age': 0.0565, 'Education': 0.1427
#shap_values_standin0 = pd.Series({'pred' : 0.1414,'Age': -0.0012, 'Education': -0.
#shap_values_standin0 = pd.Series({'pred' : 0.7445,'Age': 0.0774, 'Education': 0.19
#shap_values_standin0 = pd.Series({'pred' : 0.4127,'Age': 0.0668, 'Education': 0.16

from IPython.core.display import display, HTML
import regex as re
doc = X.iloc[theloc]
doc3 = X_display.iloc[theloc]
print(doc3)
targets = [0,1]

doc3 = X_display.iloc[theloc,:]



explanation = eli5.explain_prediction(model, doc, target_names=['NO','YES'], target
the_html = eli5.format_as_html(explanation, existing_values = shap_values_standin0,
                               feature_values = doc3, new_probability = shap_values
                               show_feature_values = True)
#the_html = the_html.replace(".000","")
the_html = the_html.replace("1.0","1")
the_html = the_html.replace("probability","prediction")
the_html = the_html.replace("Contribution<sup>?</sup>","Contribution")
display(HTML(the_html))
display(HTML(the_html))
#eli5.formatters.html.format_as_html(prediction, show_feature_values=True)
#pred = eli5.explain_prediction(model, doc, target_names=['Yes','No'])
#eli5.formatters.html.format_as_html(pred, show_feature_values=True)
```

```
Age                          27.0
Education            Some College
Occupation           Tech-support
Sex                        Female
Hours per week               38.0
Name: 32556, dtype: object
diff
2.4756449695673437
```

Explained as: decision paths

```
Features with largest coefficients.

Feature weights are calculated by following decision paths in trees
of an ensemble. Each leaf has an output score, and expected scores can also be
assigned to parent nodes. Contribution of one feature on the decision path
is how much expected score changes from parent to child. Weights of all
features sum to the output score of the estimator.

Caveats:
1. Feature weights just show if the feature contributed positively or
   negatively to the final score, and does not show how increasing or
   decreasing the feature value will change the prediction.
2. In some cases, feature weight can be close to zero for an important feature
   For example, in a single tree that computes XOR function, the feature at th
   top of the tree will have zero weight because expected scores for both
   branches are equal, so decision at the top feature does not change the
   expected score. For an ensemble predicting XOR functions it might not be
   a problem, but it is not reliable if most trees happen to choose the same
   feature at the top.
```

y=YES (prediction **0.077**, score **-2.477**) top features

Contribution	Feature	Value
+0.990	Education	Some College
+0.514	Occupation	Tech-support
-0.001	base value	1
-0.845	Sex	Female
-1.647	Hours per week	38.0
-3.333	Age	27.0

Backup code for normalization

```
import json
```

```
orig_yhat = float('%.4f'%(float(str(plotobj._data[0]['customdata'][0]).split('</br>
#Age
age_check = [str(datum[0]).split('</br>')[2] for datum in plotobj._data[0]['customd
#print(age_check)
age_subs = [abs(float(str(datum[0]).split('</br>')[-2].split(' ')[-1]) - float(str(
age_sumsubs = [float(str(datum[0]).split('</br>')[-2].split(' ')[-1]) - float(str(d
age_avg_subs = float('%.4f'%(sum(age_subs)/len(age_subs)))
if sum(age_sumsubs) < 0:
  age_avg_subs = -1.0 * age_avg_subs



#Hours per week
hours_check = [str(datum[0]).split('</br>')[2] for datum in plotobj._data[1]['custo
#print(hours_check)
hours_subs = [abs(float(str(datum[0]).split('</br>')[-2].split(' ')[-1]) - float(st
hours_sumsubs = [float(str(datum[0]).split('</br>')[-2].split(' ')[-1]) - float(str
hours_avg_subs = float('%.4f'%(sum(hours_subs)/len(hours_subs)))
if sum(hours_sumsubs) < 0:
  hours_avg_subs = -1.0 * hours_avg_subs



#Education
edu_check = [str(datum[0]).split('</br>')[2] for datum in plotobj2._data[0]['custom
#print(edu_check)
edu_subs = [abs(float(str(datum[0]).split('</br>')[-2].split(' ')[-1]) - float(str(
edu_sumsubs = [float(str(datum[0]).split('</br>')[-2].split(' ')[-1]) - float(str(d
edu_avg_subs = float('%.4f'%(sum(edu_subs)/len(edu_subs)))
if sum(edu_sumsubs) < 0:
  edu_avg_subs = -1.0 * edu_avg_subs

#Occupation
occ_check = [str(datum[0]).split('</br>')[2] for datum in plotobj2._data[1]['custom
#print(occ_check)
occ_subs = [abs(float(str(datum[0]).split('</br>')[-2].split(' ')[-1]) - float(str(
occ_sumsubs = [float(str(datum[0]).split('</br>')[-2].split(' ')[-1]) - float(str(d
occ_avg_subs = float('%.4f'%(sum(occ_subs)/len(occ_subs)))
if sum(occ_sumsubs) < 0:
  occ_avg_subs = -1.0 * occ_avg_subs



#Sex
sex_check = [str(datum[0]).split('</br>')[2] for datum in plotobj2._data[2]['custom
#print(sex_check)
sex_subs = [abs(float(str(datum[0]).split('</br>')[-2].split(' ')[-1]) - float(str(
sex_sumsubs = [float(str(datum[0]).split('</br>')[-2].split(' ')[-1]) - float(str(d
sex_avg_subs = float('%.4f'%(sum(sex_subs)/len(sex_subs)))
if sum(sex_sumsubs) < 0:
  sex_avg_subs = -1.0 * sex_avg_subs

numlist = [age_avg_subs, edu_avg_subs, occ_avg_subs, sex_avg_subs, hours_avg_subs]
num2list = [sum(age_sumsubs)/len(age_sumsubs), sum(edu_sumsubs)/len(edu_sumsubs), s
num3list = [sum(age_sumsubs)/len(age_sumsubs), sum(edu_sumsubs)/len(edu_sumsubs), s


#[0.0195, -0.0582, -0.0229, -0.0614, -0.0201]
base_value = 0.259
diff = abs(orig_yhat - base_value)
diff = abs(orig_yhat - base_value)
norm = [float(i)/abs(sum(num2list)) for i in num2list]
norm_fin = [float('%.4f'%(diff * val)) for val in norm]
num2list_fin = [float('%.4f'%(val)) for val in num2list]
print("original yhat Age Education Occupation Sex Hours Per Week")
print(str(orig_yhat) + str(numlist) + " norm " + str(norm_fin))
print(str(orig_yhat) + str(numlist) + " posneg " + str(num2list_fin))
```