Lab 3 - Explainable and Trustworthy AI

Teaching Assistant: Eleonora Poeta (eleonora.poeta@polito.it)

Lab 3b: Local post-hoc explainable models on structured data - SHAP

SHAP

SHAP (**SH**apley **A**dditive ex**P**lanations) is a local explanation method designed to explain **individual predictions**. By dissecting the prediction of a specific *x* instance, it calculates the contribution of each feature to the outcome.

- Based on the theory of **Shapley values**, SHAP ensures a fair distribution of *credit* for collaborative tasks.
- Within SHAP, **each characteristic value** assumes the role of a **strategic "player "** in a predictive "game". The final goal of the *game* is to predict the outcome of a given instance, with each characteristic influencing the final prediction.

Exercise 1:

The exercise requires the use of **SHAP** to explain the predictions of individual instances of the Adult dataset.

> The Adult dataset, also known as the "Census Income" dataset, contains demographic information about people, such as age, education, occupation, marital status and more, extracted from the 1994 U.S. Census Bureau database. **Each entry** in the dataset represents a **person**, and the associated **task** is to **predict whether an individual earns more than $50,000 per year** or less.

Below, you can find the **main steps of this exercise.** In addition, you will find in the unfolding a set of step-by-step instructions made to guide you in implementing your solution:

1. **Install SHAP** library.
2. **Load** the Adult dataset.
   - SHAP provides an instance of the Adult dataset directly into its library.
   - In particular, SHAP provides **two versions of Adult dataset:**
     - The first version is **already preprocessed**; therefore, it has neither missing values nor categorical values. We will use this version of the dataset for both the classifier and the explainer.
     - The second version is the **original** one. This is valuable for our SHAP analysis, as it produces results that are inherently meaningful and easily interpreted. For example, instead of denoting a feature as "feature_0=45", we obtain a more intuitive representation, such as "Age=45".
3. Split the Adult dataset. 80/20 train-test ratio.
4. Train a RandomForestClassifier and fit it over the training dataset. Evaluate the model.
5. Use the `shap.Explainer` to explain the instances `id=1` and `id=7` .
   - **Plot** with a bar chart from `matplotlib.pyplot` the **Shapley values** for the instances `id=1` and `id=7` .

   Local Explanations:
   - Plot the shap explanation for the instances `id=1` and `id=7` with:
     - `shap.force_plot`
     - `shap.waterfall_plot`

   Global (for all features) Explanations:
   - Plot the shap explanation for the instances `id=1` and `id=7` with:
     - `shap.force_plot`
     - `shap.summary_plot`
     - `shap.dependence_plot`
6. Do again point 5.) with `shap.KernelExplainer` and `shap.ExactExplainer`

Solution:

1.) Imports

```
! pip install -q shap
```

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import shap

# To visualize shap plots
shap.initjs()
```

2.) Load dataset

```
X, y = shap.datasets.adult()
X
```

	Age	Workclass	Education-Num	Marital Status	Occupation	Relationship	Race	Sex
0	39.0	7	13.0	4	1	0	4	1
1	50.0	6	13.0	2	4	4	4	1
2	38.0	4	9.0	0	6	0	4	1
3	53.0	4	7.0	2	6	4	2	1
4	28.0	4	13.0	2	10	5	2	0
...	...	...	...	...	...	...	...	...
32556	27.0	4	12.0	2	13	5	4	0
32557	40.0	4	9.0	2	7	4	4	1
32558	58.0	4	9.0	6	1	1	4	0
32559	22.0	4	9.0	4	1	3	4	1
32560	52.0	5	9.0	2	4	5	4	0

Next steps: View recommended plots

```
# Load the data with display=True. This dataset still contains the categorical values.
X_display, y_display = shap.datasets.adult(display=True)
```

3.) Split dataset

```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

4.) Train the Random Forest Classifier

```
# Train a RandomForestClassifier

from sklearn.linear_model import LogisticRegression
rf_clf = LogisticRegression()
rf_clf.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
  n_iter_i = _check_optimize_result(
```

▾ LogisticRegression
LogisticRegression()

```
# Evaluate the model
accuracy = rf_clf.score(X_test, y_test)
accuracy
```

```
0.8054659910947336
```

5.) SHAP - Explainer

`shap.Explainer` wants as parameters:

- A model object or a **prediction function** that computes the output of the model. We will use the **function** of the model that **predicts the probabilities of the classes** you are using (e.g. `.predict_proba` if you are using the Random Forest).
- A **`masker`**. The masker parameter in the SHAP explainer is an optional argument that allows you to **specify** a **masking function** for the input data. The masking function defines *how certain features are masked* or hidden during the explanation process.
    - Using the masker `shap.maskers.Independent(data=X_train)` you can do **Dataset-Based Masking** (via marginalization). In this case the masking behavior is derived from the provided dataset. The masker calculates summary statistics from the dataset (such as means, medians, or quantiles) and uses them to mask the features during the explanation process.

When using `explainer = shap.Explainer` you can have two results:

1. From `explainer(X)` you obtain **shap_values_explanation** that does not only contains Shap values. It is an **Explanation object**.
    - For reasons of computational time, we will select a subset of the data set (`sample_data`) to provide to the explainer. Only the first 100 rows.
2. From `explainer.shap_values(X)` you obtain a `numpy.ndarray` that contains *only* the Shapley values.
    - We are interested in explaining the Shapley values for the `class_label = 0`. So, you have to take only the shapley_values of this class.
3. We can also compute the **expected_value**.
    - In SHAP library the expected value is called `base_values`. Again you have to select only the one related to the class we want to explain (class 0).

```
# Select the first 100 rows of the X dataset. This is the sample_data.
sample_data = X.loc[:100]

# Select the first 100 rows of the X_display dataset.
sample_X_display = X_display.loc[:100]

# Compute the masker
masker = shap.maskers.Independent(data = X_train)

# Instanciate the shap.Explainer
explainer = shap.Explainer(rf_clf.predict_proba, masker=masker)


# Calculate the explainer over the sample_data
shap_values_explanation = explainer(sample_data)

# Print the shap_values_explanation
print(type(shap_values_explanation))
print(shap_values_explanation[0])
print('-' * 80)

# Class for which we want to analyze the shapley values
class_index = 0  # Note that being a binary classification problem, the shapley values for the other class are -(shapley val


# Calculate the Shapley values with explainer.shap_values( ... )
shap_values_ndarray = explainer.shap_values(sample_data)[:, :, class_index]

print(type(shap_values_ndarray))
print(shap_values_ndarray[0])


# Calculate the expected_value
expected_value = shap_values_explanation.base_values[0][class_index] # this is the expected value for the class we want to e
```

```
PermutationExplainer explainer: 102it [00:25,  3.84it/s]
<class 'shap._explanation.Explanation'>
.values =
array([[-0.00019089,  0.00019089],
       [ 0.05036451, -0.05036451],
       [-0.03845874,  0.03845874],
       [ 0.07251026, -0.07251026],
       [ 0.00119603, -0.00119603],
       [ 0.13102821, -0.13102821],
       [ 0.00409719, -0.00409719],
       [-0.00106697,  0.00106697],
       [-0.04773117,  0.04773117],
       [ 0.00476168, -0.00476168],
       [ 0.00120235, -0.00120235],
       [ 0.01582139, -0.01582139]])

.base_values =
array([0.77300948, 0.22699052])

.data =
array([3.900e+01, 7.000e+00, 1.300e+01, 4.000e+00, 1.000e+00, 0.000e+00,
       4.000e+00, 1.000e+00, 2.174e+03, 0.000e+00, 4.000e+01, 3.900e+01])
--------------------------------------------------------------------------------
<class 'numpy.ndarray'>
[-0.00017722  0.05038561 -0.03937563  0.07002852  0.00114243  0.13470589
  0.00397646 -0.00112625 -0.04650072  0.00449802  0.00124358  0.01473315]
```

```
shap_values_ndarray[0]
```

```
array([ 0.03098059,  0.02008121, -0.06404527,  0.06476918,  0.03175431,
        0.08579918,  0.00113312, -0.00513044,  0.04417905,  0.00269498,
        0.02338769,  0.00085021])
```

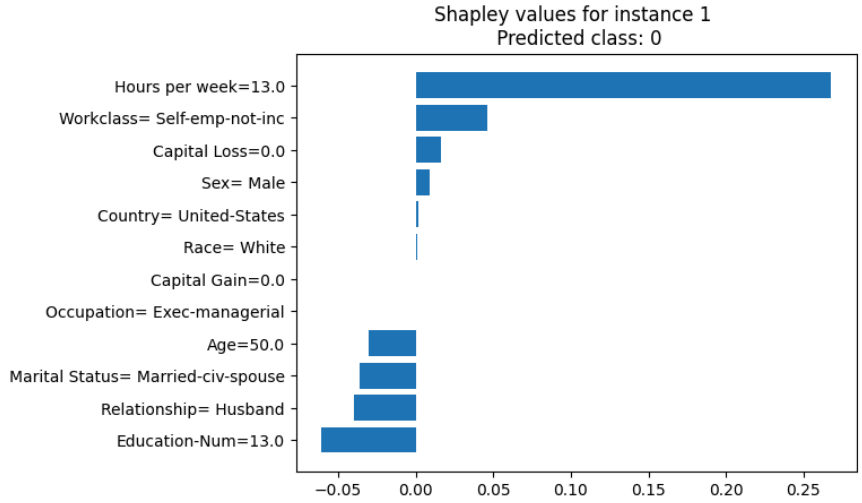
Plots

Bar chart of Shapley values for the `id_instance=1`

```
# Calculate the Shapley values with explainer.shap_values( ... )
shap_values_ndarray = explainer.shap_values(sample_data)[:, :, class_index]
```

```
id_instance = 1

# Sort feature indices based on SHAP values using np.argsort()
sorted_indices = np.argsort(shap_values_ndarray[id_instance])

# Get feature names and values for the instance
feature_names_values = np.array([f'{f}={value}' for f, value in zip(sample_X_display.columns, sample_X_display.iloc[id_insta

# Plot SHAP values
plt.barh(feature_names_values[sorted_indices], shap_values_ndarray[id_instance][sorted_indices])

# Predict class for the instance
predicted_class = int(rf_clf.predict(sample_data.iloc[id_instance:id_instance+1])[0])

# Plot title
plt.title(f'Shapley values for instance {id_instance} \n w.r.t {class_index}. Predicted class: {predicted_class}')
```

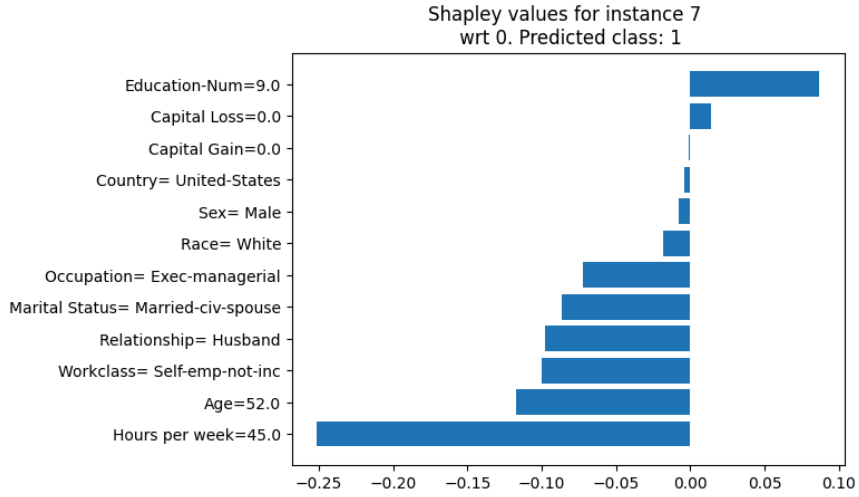
```
Text(0.5, 1.0, 'Shapley values for instance 1 \n Predicted class: 0')
```



Bar chart of Shapley values for the `id_instance=7`

```
id_instance = 7

# Sort feature indices based on SHAP values
sorted_indices = np.argsort(shap_values_ndarray[id_instance])

# Get feature names and values for the instance
feature_names_values = np.array([f'{f}={value}' for f, value in zip(sample_X_display.columns, sample_X_display.iloc[id_insta

# Plot SHAP values
plt.barh(feature_names_values[sorted_indices], shap_values_ndarray[id_instance][sorted_indices])

# Predict class for the instance
predicted_class = int(rf_clf.predict(sample_data.iloc[id_instance:id_instance+1])[0])

# Plot title
plt.title(f'Shapley values for instance {id_instance} \n w.r.t {class_index}. Predicted class: {predicted_class}')
```

```
Text(0.5, 1.0, 'Shapley values for instance 7 \n wrt 0. Predicted class: 1')
```

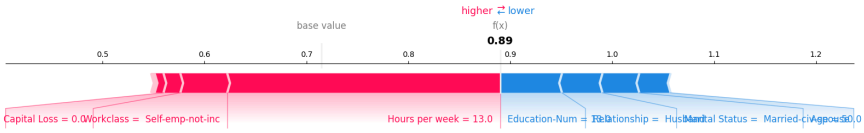


SHAP Force plot - single instance

`shap.force_plot()` shows bars that indicate the **magnitude** and **direction** of the **influence of the features on the model prediction**. The graph also includes the base_value, which represents the average outcome of the model in the dataset, and an arrow indicating the final predicted value f(x) for the instance.

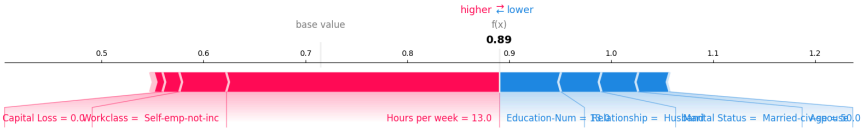
```
id_instance = 1


print(rf_clf.predict_proba(sample_data)[id_instance])


shap.force_plot(base_value=expected_value, # the base_value is the expected_value
                shap_values=shap_values_ndarray[id_instance, :], # select the shap_value of the considered instance
                features=X_display.iloc[id_instance, :], # Use the X_display dataset to have meaningful result
                matplotlib=True)
```

```
[0.89 0.11]
```




```
# Complete with your code!
id_instance = 1


print(rf_clf.predict_proba(sample_data)[id_instance])


shap.force_plot(base_value= , # the base_value is the expected_value
                shap_values= , # select the shap_value of the considered instance
                features= , # Use the X_display dataset to have meaningful result
                matplotlib=True)
```

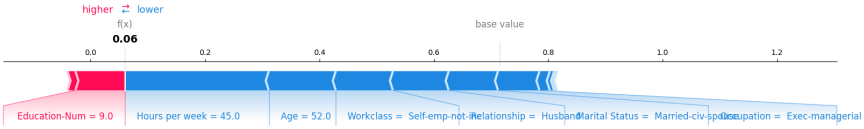
```
[0.89 0.11]
```




```
id_instance = 7
print(rf_clf.predict_proba(sample_data)[id_instance])
```

```
shap.force_plot(base_value=expected_value,
                shap_values=shap_values_ndarray[id_instance, :],
                features=X_display.iloc[id_instance, :],
                matplotlib=True)
```

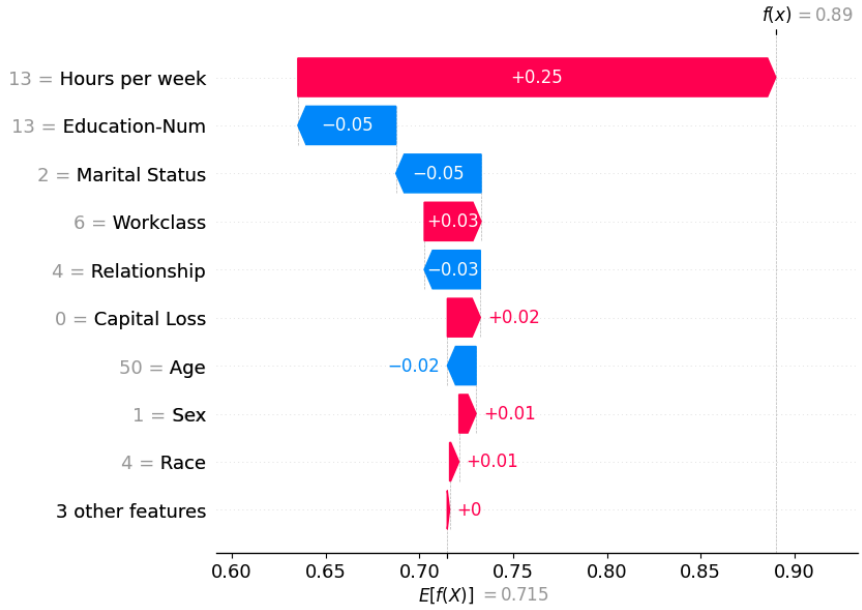
```
[0.06 0.94]
```



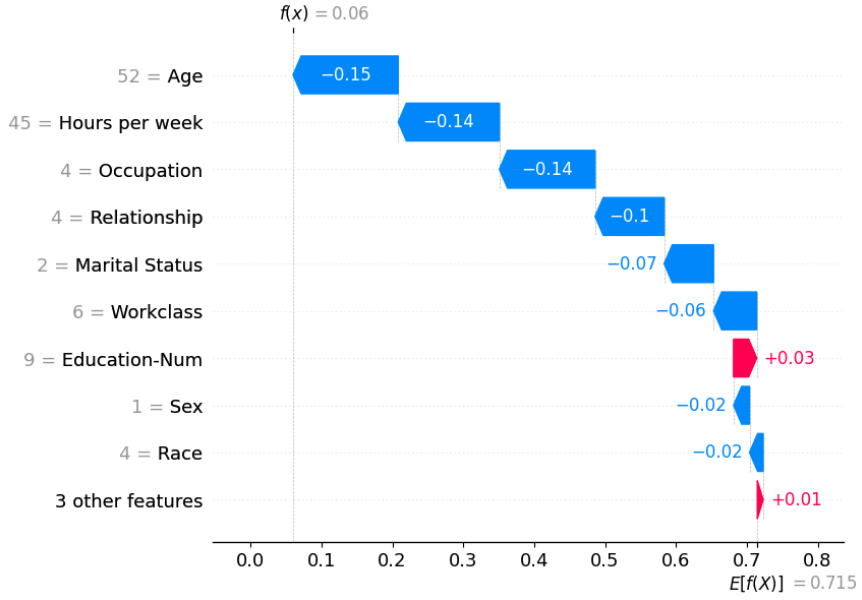
SHAP - waterfall plot

`shap.waterfall_plot` graph shows a sequence of horizontal bars, each representing the contribution of a feature to the overall forecast. The **bars** are **cascaded**, where each successive bar adds to the previous one, visually illustrating **how each characteristic incrementally affects the final forecast.** Positive contributions are displayed as red arrows going on the right, and negative contributions are blue arrows on the left.

```
id = 1
shap.waterfall_plot(shap_values=shap_values_explanation[id, :, class_index], # select the shap_value_explanation of the cons
                    max_display=10)
```




```
id = 7
shap.waterfall_plot(shap_values=shap_values_explanation[id, :, class_index], max_display=10)
```



SHAP Force plot - multiple instances

For visualization reasons, if you are doing the Lab on Google Colab you can't visualize the output of the force_plot for multiple instances.

Unlikely before, now the `shap.force_plot` visualizes the impact of features across multiple instances concurrently.

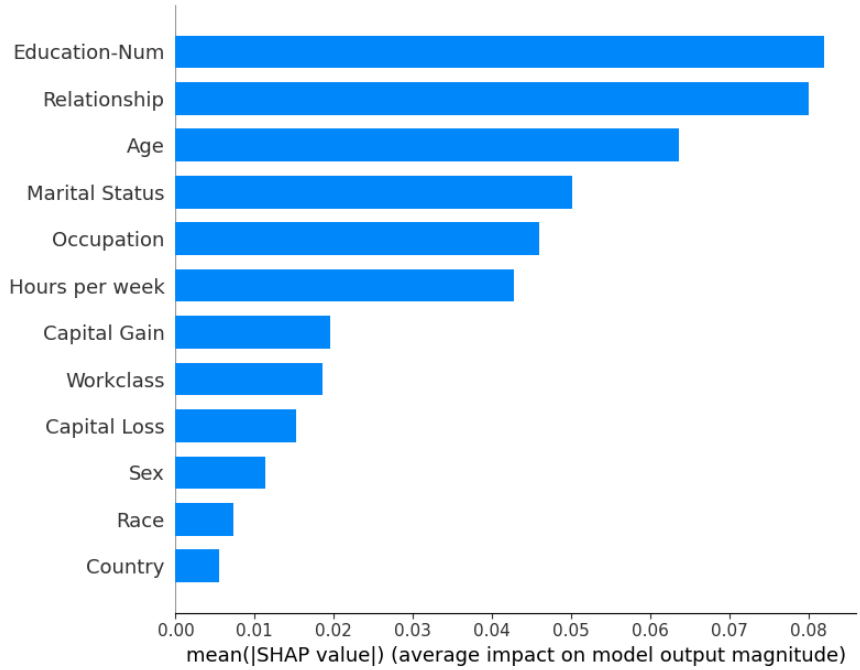
```
shap.force_plot(base_value=expected_value,
                shap_values=shap_values_ndarray[:10, :],
                features=X_display.iloc[:10, :])
```

Visualization omitted, Javascript library not loaded!
Have you run `initjs()` in this notebook? If this notebook was from another user you must also trust this notebook (File -> Trust notebook). If you are viewing this notebook on github the Javascript has been stripped for security. If you are using JupyterLab this error is because a JupyterLab extension has not yet

SHAP Summary plot

- It is used to visualize the **summary** of **SHAP values** for all features in a dataset. This plot is particularly useful for understanding the importance and directionality of different features in influencing model predictions.
- It provides the **average impact** of each feature on the model output magnitude.
- It **aggregates** the **absolute SHAP values** for each feature across all instances and then visualizes these aggregated values.

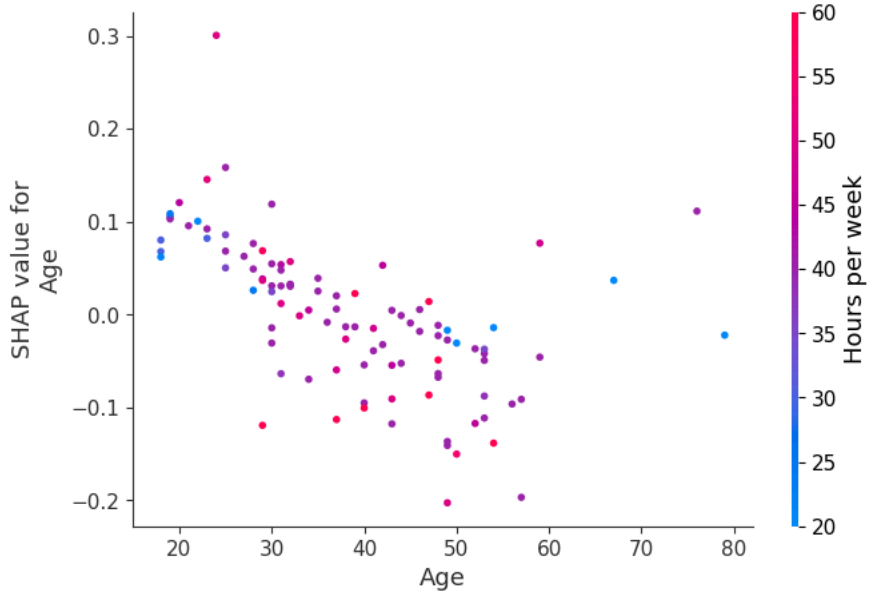
```
# Summary plot. This consider ALL instances.
shap.summary_plot(shap_values=shap_values_ndarray,
                  features=X_display, # to display meaningful features use X_display
                  plot_type='bar')
```

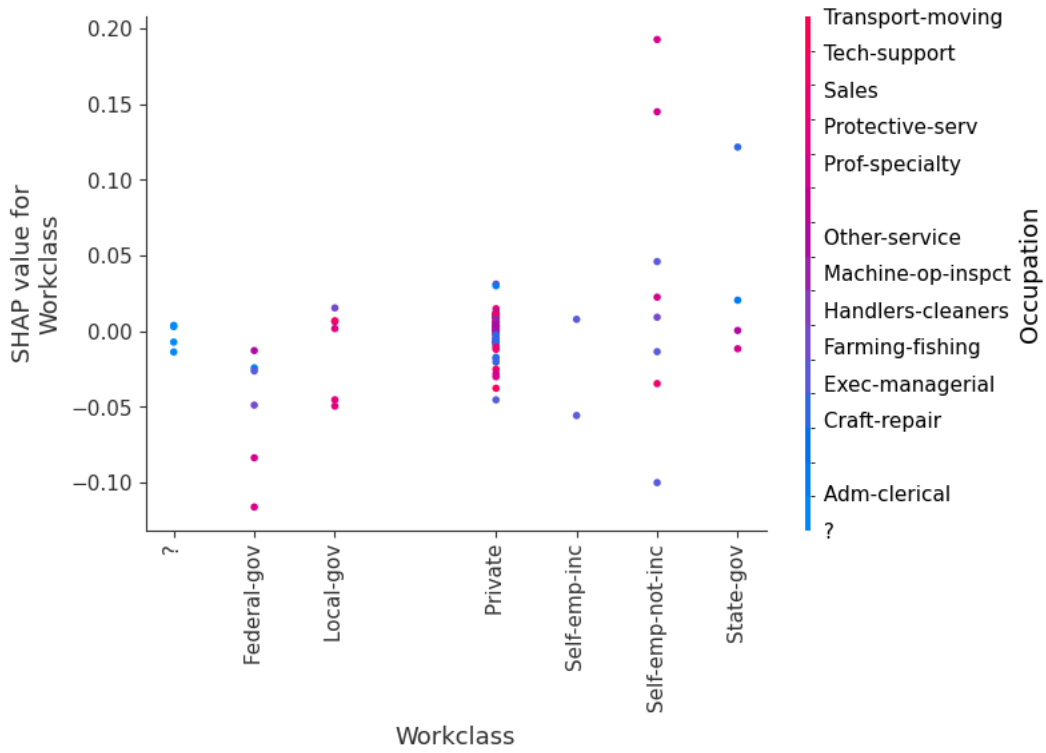


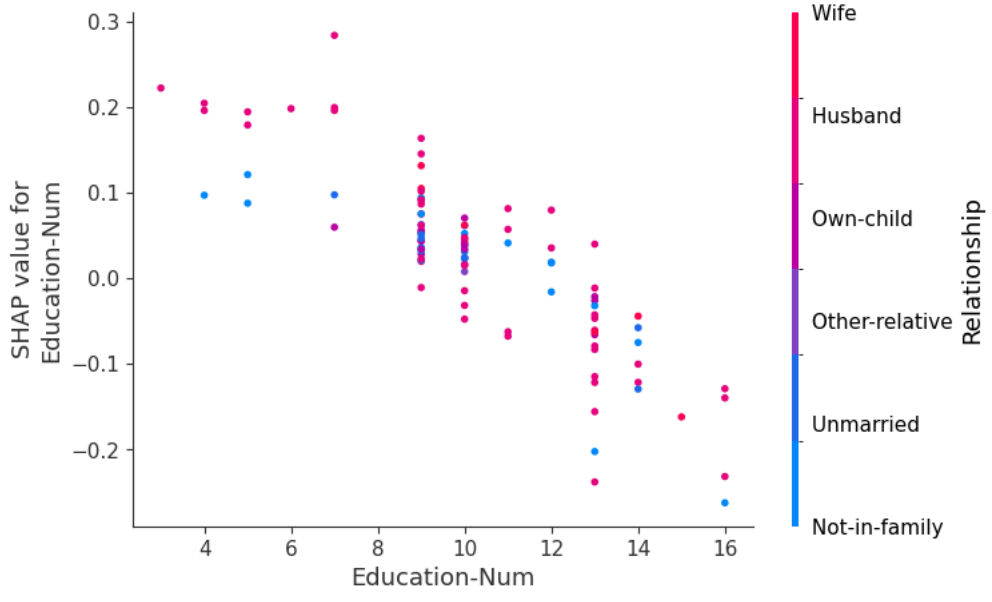
SHAP dependence plot

The graph generated by `shap.dependence_plot` typically consists of a scatter plot in which each point represents an instance of the dataset. The x-axis represents the values of the chosen feature, while the y-axis represents the corresponding SHAP values.

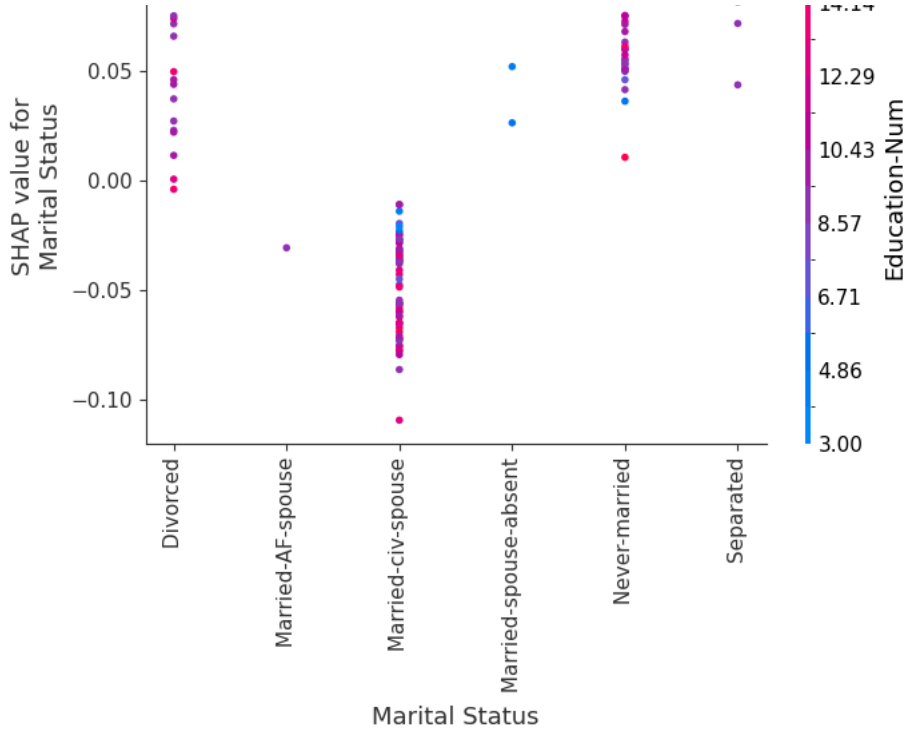
```
# Iterate over the X_train.columns and for each feature, plot the shap.dependence_plot

for name in X_train.columns:
    shap.dependence_plot(name,
                         shap_values=shap_values_ndarray, # shapley values of all instances
                         features=sample_data, # sample_data
                         display_features=X_display) # to display meaningful features use X_display
```

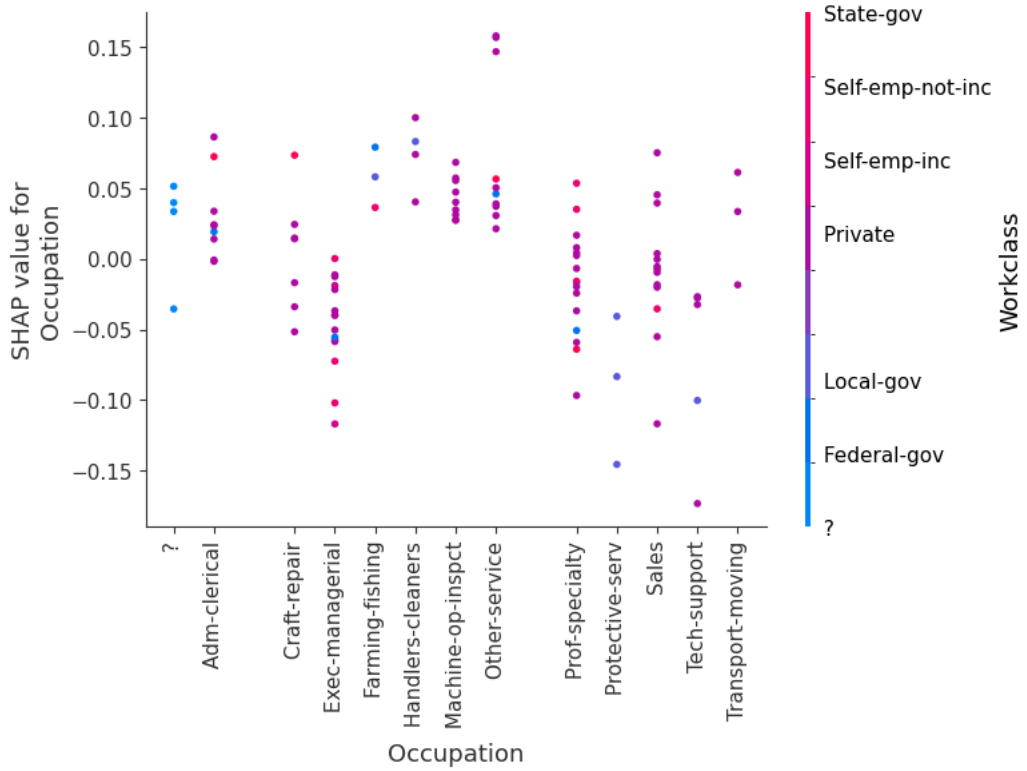

SHAP value for Age
Age
Hours per week

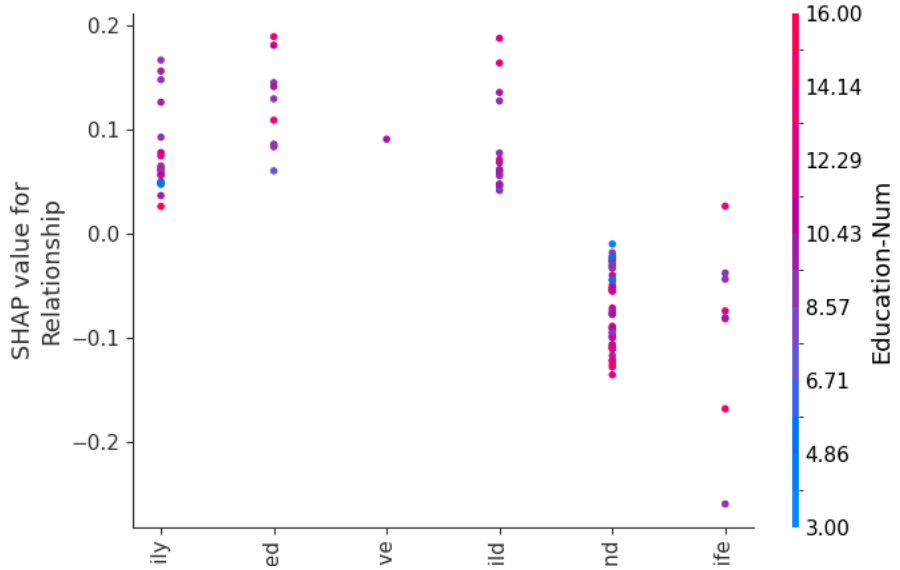

SHAP value for Workclass
Workclass
Occupation
Transport-moving
Tech-support
Sales
Protective-serv
Prof-specialty
Other-service
Machine-op-inspct
Handlers-cleaners
Farming-fishing
Exec-managerial
Craft-repair
Adm-clerical
?

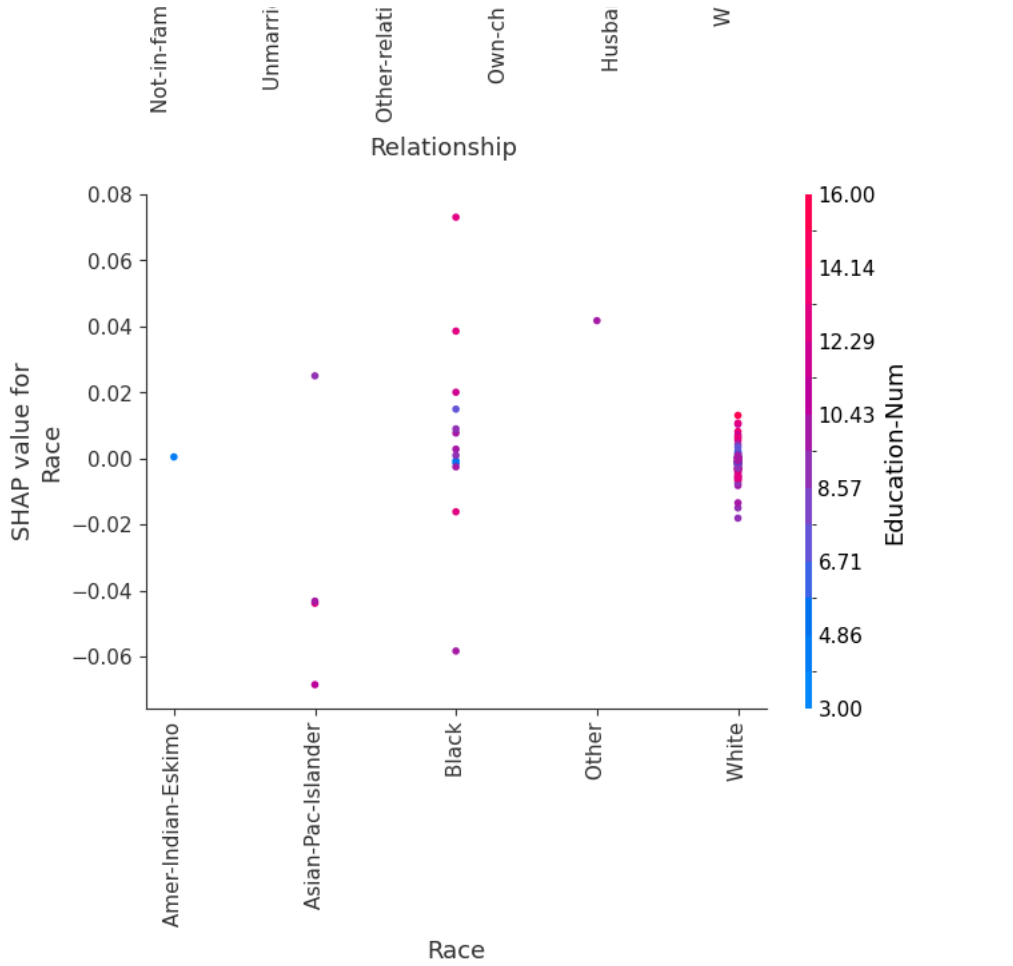

SHAP value for Education-Num
Education-Num
Relationship
Wife
Husband
Own-child
Other-relative
Unmarried
Not-in-family

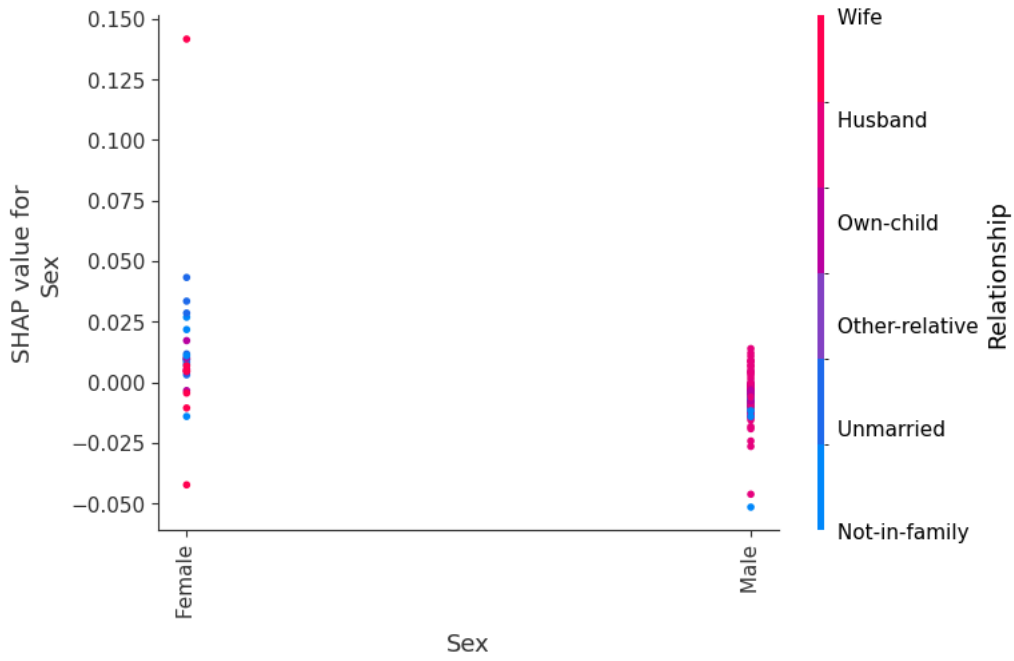

16.00


SHAP value for
Marital Status
0.05
0.00
−0.05
−0.10
Divorced
Married-AF-spouse
Married-civ-spouse
Married-spouse-absent
Never-married
Separated
Marital Status
12.29
10.43
8.57
6.71
4.86
3.00
Education-Num

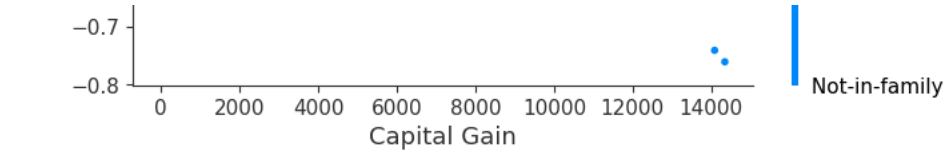

SHAP value for
Occupation
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
?
Adm-clerical
Craft-repair
Exec-managerial
Farming-fishing
Handlers-cleaners
Machine-op-inspct
Other-service
Prof-specialty
Protective-serv
Sales
Tech-support
Transport-moving
Occupation
State-gov
Self-emp-not-inc
Self-emp-inc
Private
Local-gov
Federal-gov
?
Workclass

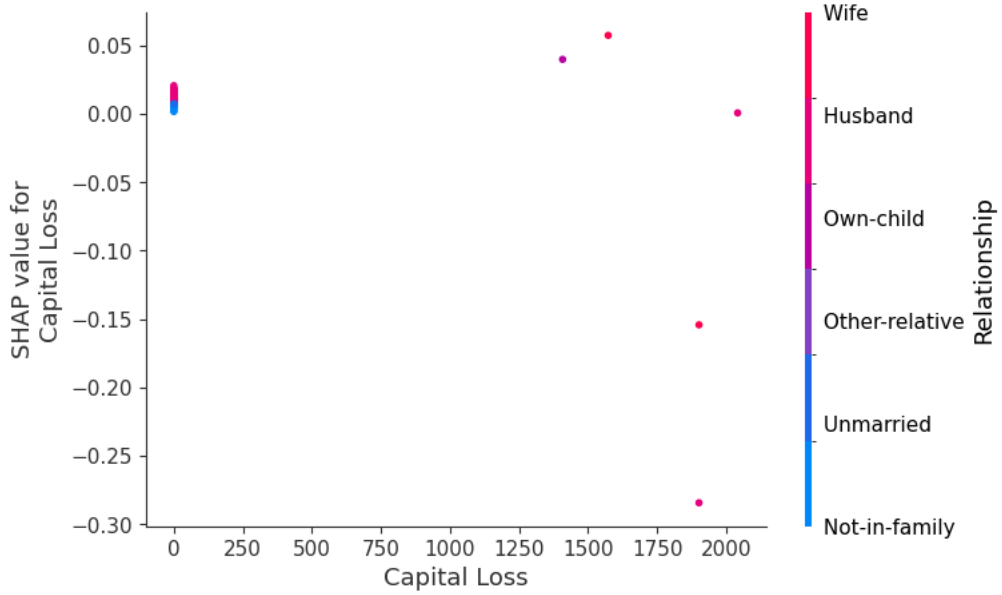

SHAP value for
Relationship
0.2
0.1
0.0
−0.1
−0.2
ily
ed
ve
ild
nd
ife
16.00
14.14
12.29
10.43
8.57
6.71
4.86
3.00
Education-Num

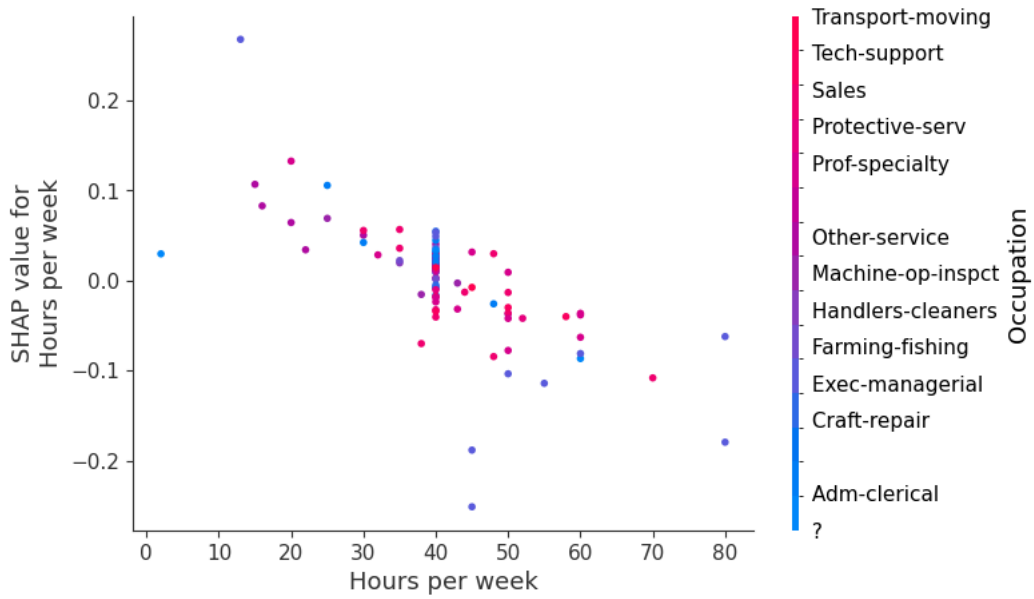

Not-in-fam
Unmarri
Other-relati
Own-ch
Husba
W
Relationship
SHAP value for Race
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
16.00
14.14
12.29
10.43
8.57
6.71
4.86
3.00
Education-Num
Amer-Indian-Eskimo
Asian-Pac-Islander
Black
Other
White
Race

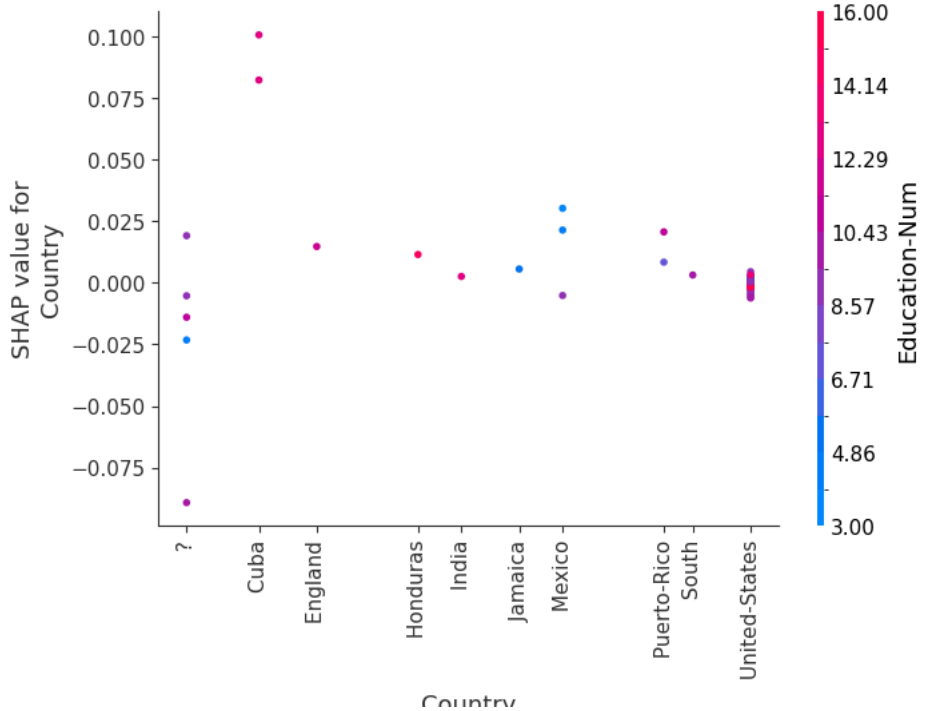

SHAP value for Sex
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
Wife
Husband
Own-child
Other-relative
Unmarried
Not-in-family
Relationship
Female
Male
Sex


SHAP value for Capital Gain
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
Wife
Husband
Own-child
Other-relative
Unmarried
Relationship


−0.7
−0.8
0
2000
4000
6000
8000
10000
12000
14000
Capital Gain
Not-in-family


SHAP value for Capital Loss
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.25
−0.30
0
250
500
750
1000
1250
1500
1750
2000
Capital Loss
Wife
Husband
Own-child
Other-relative
Unmarried
Not-in-family
Relationship


SHAP value for Hours per week
0.2
0.1
0.0
−0.1
−0.2
0
10
20
30
40
50
60
70
80
Hours per week
Transport-moving
Tech-support
Sales
Protective-serv
Prof-specialty
Other-service
Machine-op-inspct
Handlers-cleaners
Farming-fishing
Exec-managerial
Craft-repair
Adm-clerical
?
Occupation


SHAP value for Country
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
?
Cuba
England
Honduras
India
Jamaica
Mexico
Puerto-Rico
South
United-States
Country
16.00
14.14
12.29
10.43
8.57
6.71
4.86
3.00
Education-Num

Country

6.) KernelSHAP

```
sample_data = X.loc[:10]

sample_X_display = X_display.loc[:100]


sample_data_kernel = shap.sample(X_train, 10)

kernel_explainer = shap.KernelExplainer(rf_clf.predict_proba, data=sample_data_kernel)


# Print the kernel shap_values_explanation
shap_values_explanation_kernel = kernel_explainer(sample_data)
print(type(shap_values_explanation_kernel))
print(shap_values_explanation_kernel[0])
print('-' * 80)


# Class for which we want to analyze the shapley values
class_index = 0  # Note that being a binary classification problem, the shapley values for the other class are -(shapley valu


shap_values_ndarray_kernel = kernel_explainer.shap_values(sample_data)[:, :, class_index]

print(type(shap_values_ndarray_kernel))
print(shap_values_ndarray_kernel[0])

# Calculate the expected_value for Kernel SHAP
expected_value_kernel = shap_values_explanation_kernel.base_values[0][class_index] # this is the expected value for the class

print("\nExpected value")
print(expected_value_kernel)
```

```
100%                                                 11/11 [00:01<00:00,  6.16it/s]
<class 'shap._explanation.Explanation'>
.values =
array([[-0.00017554,  0.00017554],
       [ 0.05317704, -0.05317704],
       [-0.04965904,  0.04965904],
       [ 0.11484209, -0.11484209],
       [ 0.0016417 , -0.0016417 ],
       [ 0.14438249, -0.14438249],
       [ 0.        ,  0.        ],
       [-0.00194716,  0.00194716],
       [-0.07082246,  0.07082246],
       [ 0.01980146, -0.01980146],
       [ 0.00276182, -0.00276182],
       [ 0.        ,  0.        ]])

.base_values =
array([0.75254091, 0.24745909])

.data =
array([3.900e+01, 7.000e+00, 1.300e+01, 4.000e+00, 1.000e+00, 0.000e+00,
       4.000e+00, 1.000e+00, 2.174e+03, 0.000e+00, 4.000e+01, 3.900e+01])
--------------------------------------------------------------------------------
100%                                                 11/11 [00:01<00:00,  6.32it/s]
<class 'numpy.ndarray'>
[-0.00017554  0.05317704 -0.04965904  0.11484209  0.0016417   0.14438249
  0.         -0.00194716 -0.07082246  0.01980146  0.00276182  0.        ]

Expected value
0.7525409067705383
```

```
shap_values_ndarray_kernel
```

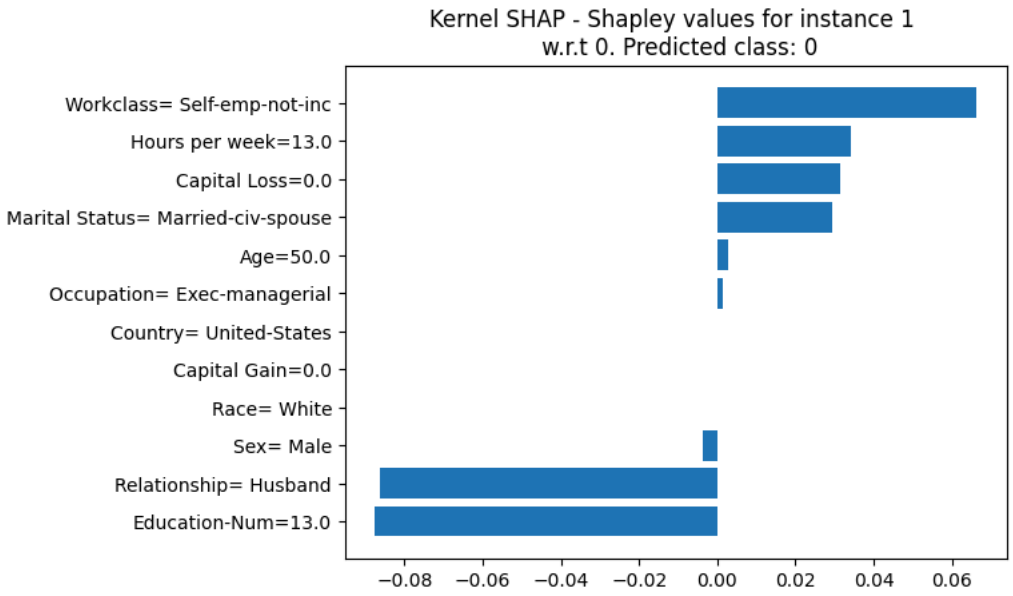
```
array([[-1.75542109e-04,  5.31770413e-02, -4.96590429e-02,
         1.14842091e-01,  1.64170432e-03,  1.44382491e-01,
         0.00000000e+00, -1.94715733e-03, -7.08224555e-02,
         1.98014608e-02,  2.76182339e-03,  0.00000000e+00],
       [ 2.62016036e-03,  6.62905721e-02, -8.75369870e-02,
         2.94333240e-02,  1.53991612e-03, -8.63642862e-02,
         0.00000000e+00, -3.75135996e-03,  0.00000000e+00,
         3.14408128e-02,  3.40802563e-02,  0.00000000e+00],
       [-4.63904435e-04, -1.40976460e-03,  1.67702972e-02,
        -1.10720945e-01,  4.69358299e-04,  2.04530147e-01,
         0.00000000e+00, -2.69691257e-03,  0.00000000e+00,
         2.49198151e-02,  3.85138202e-03,  0.00000000e+00],
```

```
[ 3.39167328e-03, -9.09575954e-04,  7.76695096e-02,
  2.98040146e-02,  6.05365626e-04, -8.50796548e-02,
 -6.21515599e-02, -3.72594202e-03,  0.00000000e+00,
  3.16765826e-02,  5.04880448e-03,  0.00000000e+00],
[-3.03326200e-03, -1.13014362e-03, -8.58178427e-02,
  1.27600607e-02, -1.08615224e-03, -2.01689281e-01,
 -5.85835002e-02,  9.12315141e-03,  0.00000000e+00,
  2.12694602e-02,  4.85828184e-03, -3.53904673e-01],
[-1.05207508e-03, -1.71324126e-03, -1.36935311e-01,
  2.40309983e-02,  1.76636645e-03, -2.11896673e-01,
  0.00000000e+00,  1.18116013e-02,  0.00000000e+00,
  3.10871702e-02,  6.07520693e-03,  0.00000000e+00],
[ 1.37142640e-03, -2.52925000e-03,  7.59044675e-02,
  6.85787954e-02, -1.90747556e-04,  1.49836633e-01,
 -3.64340181e-02,  6.10659610e-03,  0.00000000e+00,
  2.04135883e-02,  1.74581670e-02, -9.12094798e-02],
[ 2.85603926e-03,  6.01492305e-02,  2.06775538e-02,
  2.86782393e-02,  1.38269264e-03, -7.64184954e-02,
  0.00000000e+00, -3.36576691e-03,  0.00000000e+00,
  3.03149531e-02, -2.48648038e-04,  0.00000000e+00],
[-1.80878940e-03, -2.34782145e-04, -8.94861008e-02,
  1.16761666e-01, -8.67955548e-04,  1.35004834e-01,
  0.00000000e+00,  7.05810951e-03, -7.21050609e-01,
  1.71993274e-02, -3.85390667e-03,  0.00000000e+00],
[ 4.58934966e-04, -1.36903296e-03, -9.85202804e-02,
  1.71640404e-02,  1.64935559e-03, -1.19525147e-01,
  0.00000000e+00, -4.13162598e-03, -3.51650101e-01,
  2.52797096e-02,  5.58141613e-03,  0.00000000e+00],
[-1.03147056e-03, -1.33835288e-03, -6.50443308e-03,
  2.89161765e-02,  1.75149467e-03, -1.02663820e-01,
 -7.14680425e-02, -4.30150262e-03,  0.00000000e+00,
  3.25387848e-02, -4.23630850e-02,  0.00000000e+00]])
```

Kernel SHAP - Shapley values

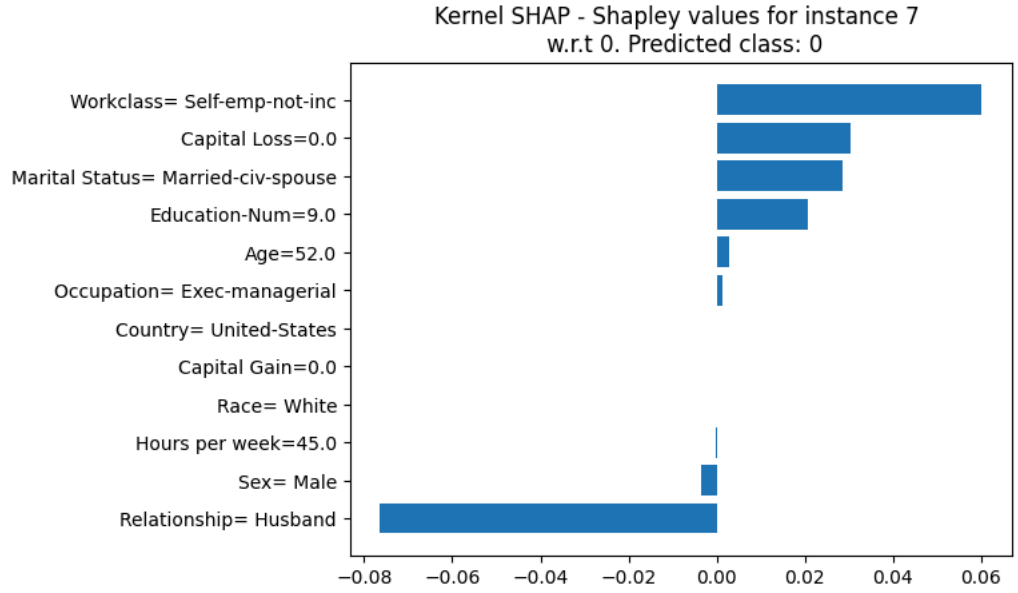
```
id_instance = 1

# Sort feature indices based on SHAP values using np.argsort()
sorted_indices = np.argsort(shap_values_ndarray_kernel[id_instance])

# Get feature names and values for the instance
feature_names_values = np.array([f'{f}={value}' for f, value in zip(sample_X_display.columns, sample_X_display.iloc[id_insta

# Plot SHAP values
plt.barh(feature_names_values[sorted_indices], shap_values_ndarray_kernel[id_instance][sorted_indices])

# Predict class for the instance
predicted_class = int(rf_clf.predict(sample_data.iloc[id_instance:id_instance+1])[0])

# Plot title
plt.title(f'Kernel SHAP - Shapley values for instance {id_instance} \n w.r.t {class_index}. Predicted class: {predicted_clas
```

```
Text(0.5, 1.0, 'Kernel SHAP - Shapley values for instance 1 \n w.r.t 0. Predicted class: 0')
```



```
id_instance = 7

# Sort feature indices based on SHAP values
sorted_indices = np.argsort(shap_values_ndarray_kernel[id_instance])

# Get feature names and values for the instance
feature_names_values = np.array([f'{f}={value}' for f, value in zip(sample_X_display.columns, sample_X_display.iloc[id_insta

# Plot SHAP values
plt.barh(feature_names_values[sorted_indices], shap_values_ndarray_kernel[id_instance][sorted_indices])

# Predict class for the instance
predicted_class = int(rf_clf.predict(sample_data.iloc[id_instance:id_instance+1])[0])

# Plot title
plt.title(f'Kernel SHAP - Shapley values for instance {id_instance} \n w.r.t {class_index}. Predicted class: {predicted_clas
```

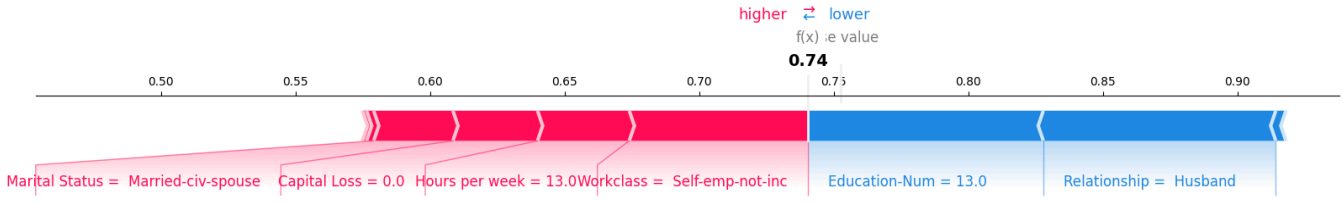
```
Text(0.5, 1.0, 'Kernel SHAP - Shapley values for instance 7 \n w.r.t 0. Predicted class: 0')
```



Kernel SHAP - force_plot

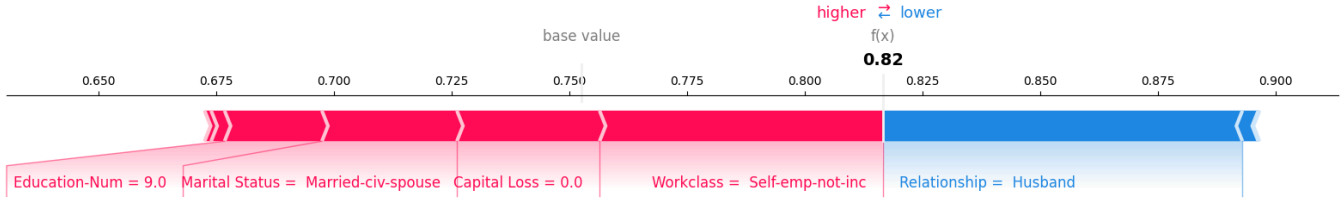
```
# Kernel SHAP
id_instance = 1


print(rf_clf.predict_proba(sample_data)[id_instance])


shap.force_plot(base_value=expected_value_kernel, # the base_value is the expected_value
                shap_values=shap_values_ndarray_kernel[id_instance, :], # select the shap_value of the considered instance
                features=X_display.iloc[id_instance, :], # Use the X_display dataset to have meaningful result
                matplotlib=True)
```

```
[0.74029332 0.25970668]
```



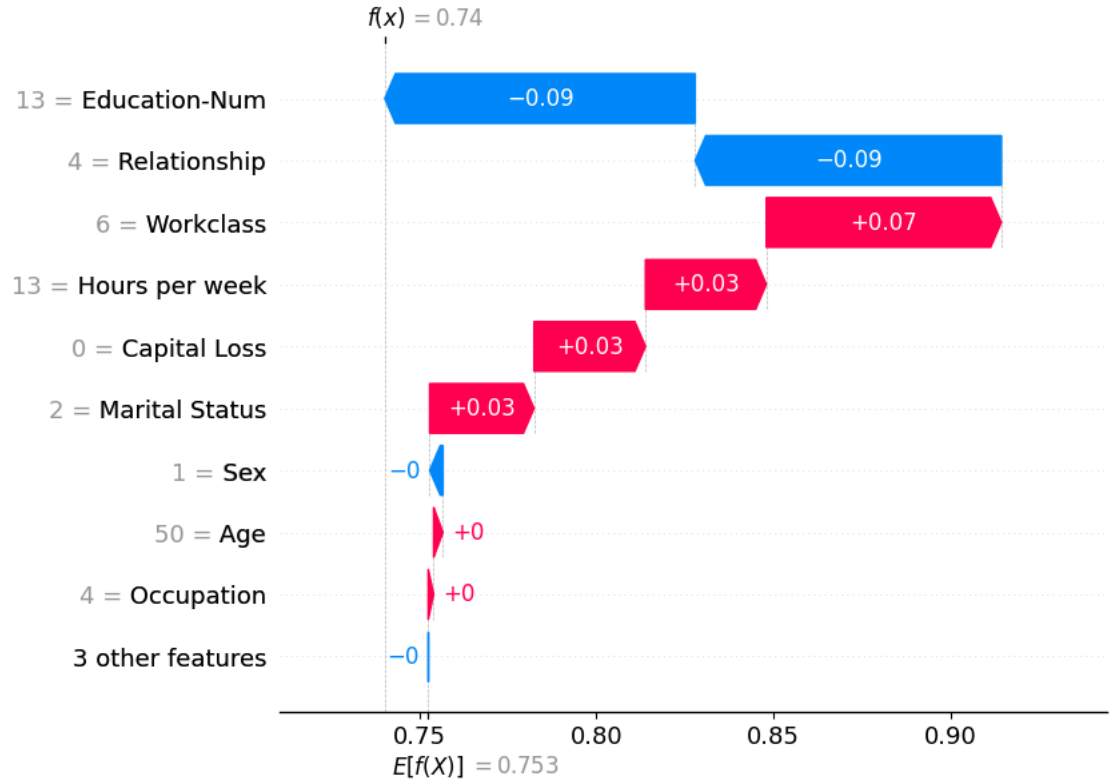
```
# Kernel SHAP
id_instance = 7
print(rf_clf.predict_proba(sample_data)[id_instance])


shap.force_plot(base_value=expected_value_kernel,
                shap_values=shap_values_ndarray_kernel[id_instance, :],
                features=X_display.iloc[id_instance, :],
                matplotlib=True)
```

```
[0.8165667 0.1834333]
```

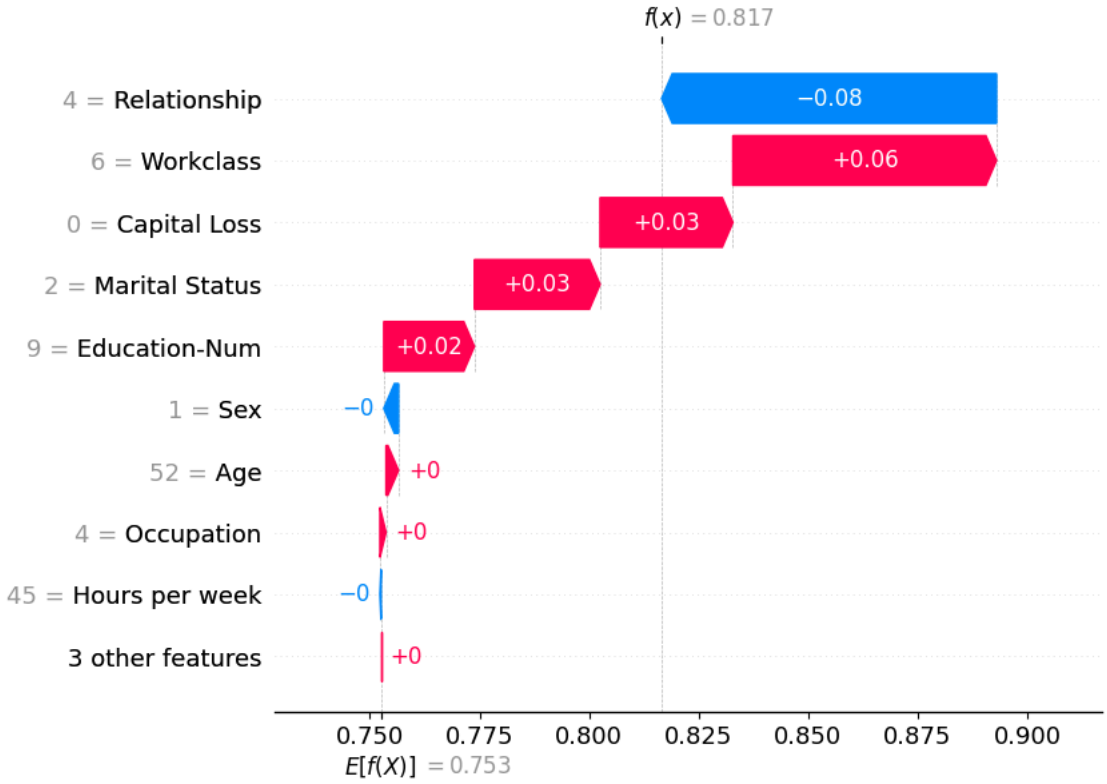


Kernel SHAP - waterfall_plot

```
# Kernel SHAP
id = 1
shap.waterfall_plot(shap_values=shap_values_explanation_kernel[id, :, class_index], # select the shap_value_explanation of t
                    max_display=10)
```




```
# Kernel SHAP
id = 7
shap.waterfall_plot(shap_values=shap_values_explanation_kernel[id, :, class_index],
                    max_display=10)
```



Kernel SHAP - force plot for more than one instance

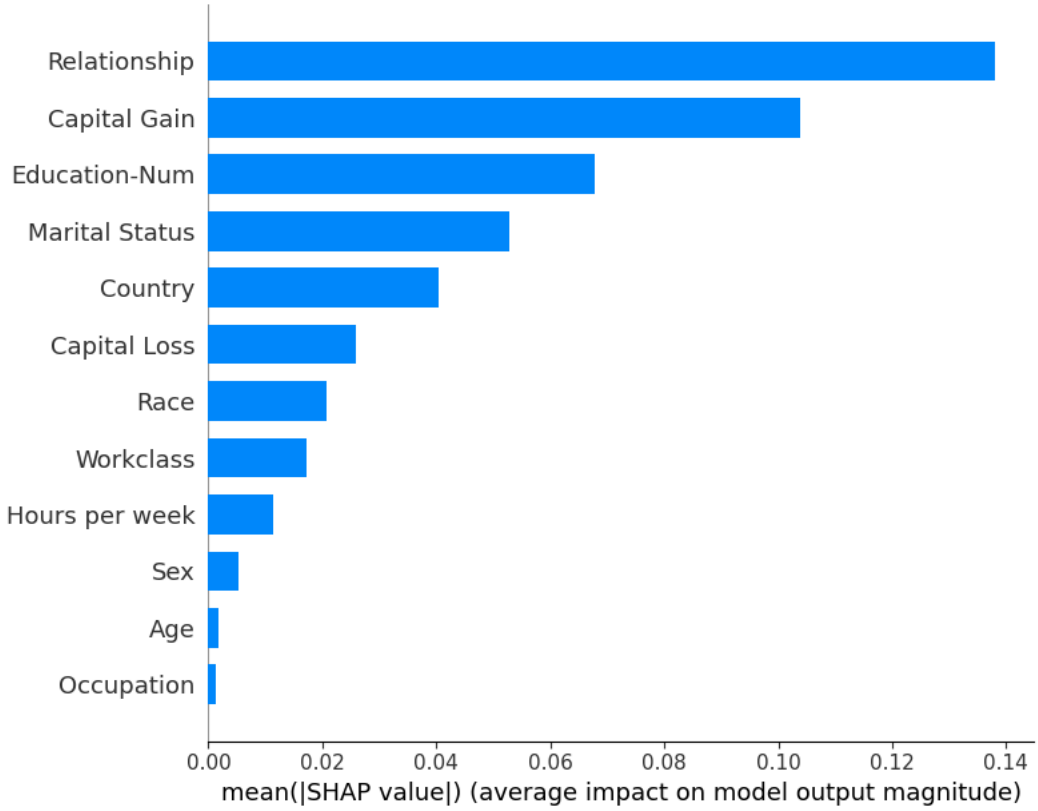
```
shap.force_plot(base_value=expected_value_kernel,
                shap_values=shap_values_ndarray_kernel[:10, :],
                features=X_display.iloc[:10, :])
```

Visualization omitted, Javascript library not loaded!

Have you run `initjs()` in this notebook? If this notebook was from another user you must also trust this notebook (File -> Trust notebook). If you are viewing this notebook on github the Javascript has been stripped for security. If you are using JupyterLab this error is because a JupyterLab extension has not yet been written.

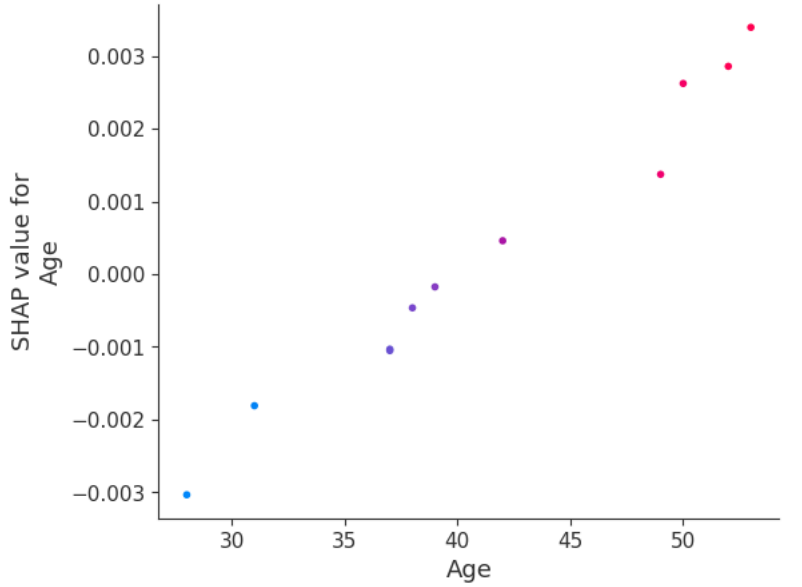
Kernel SHAP - summary plot

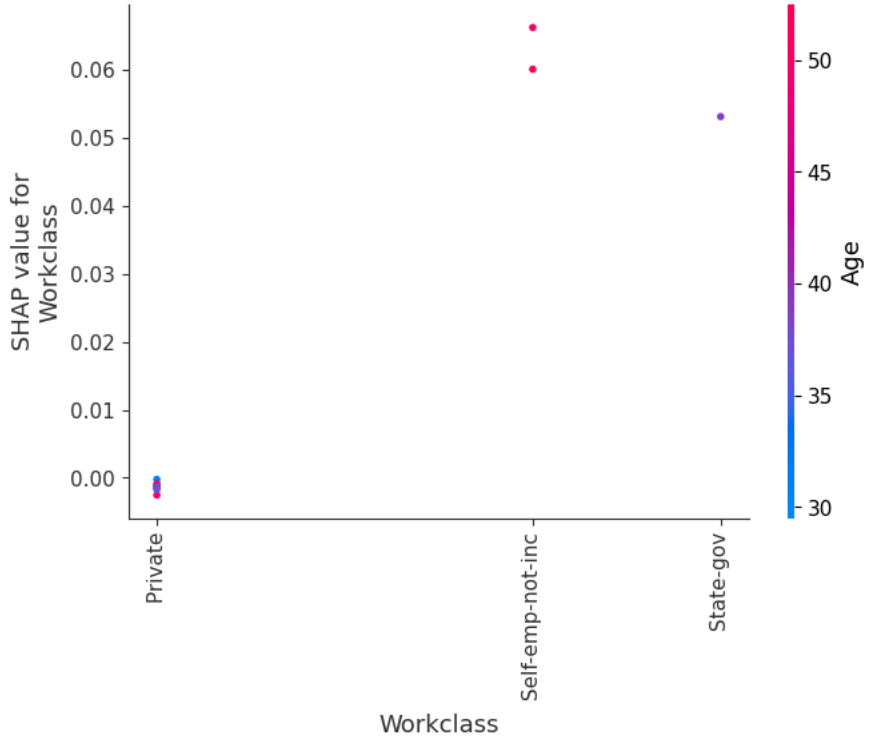
```
# Summary plot. This consider ALL instances.
shap.summary_plot(shap_values=shap_values_ndarray_kernel,
                  features=X_display, # to display meaningful features use X_display
                  plot_type='bar')
```

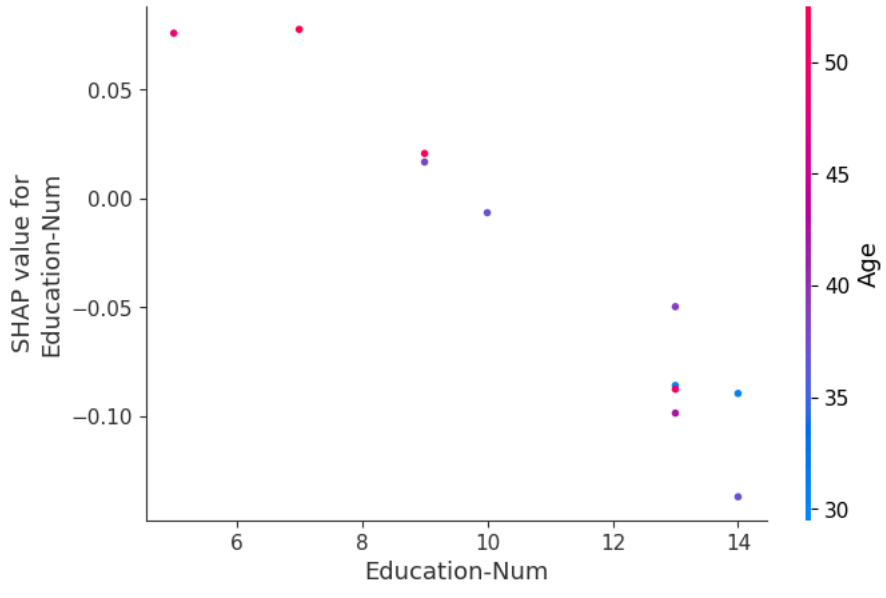


Kernel SHAP - dependence plot

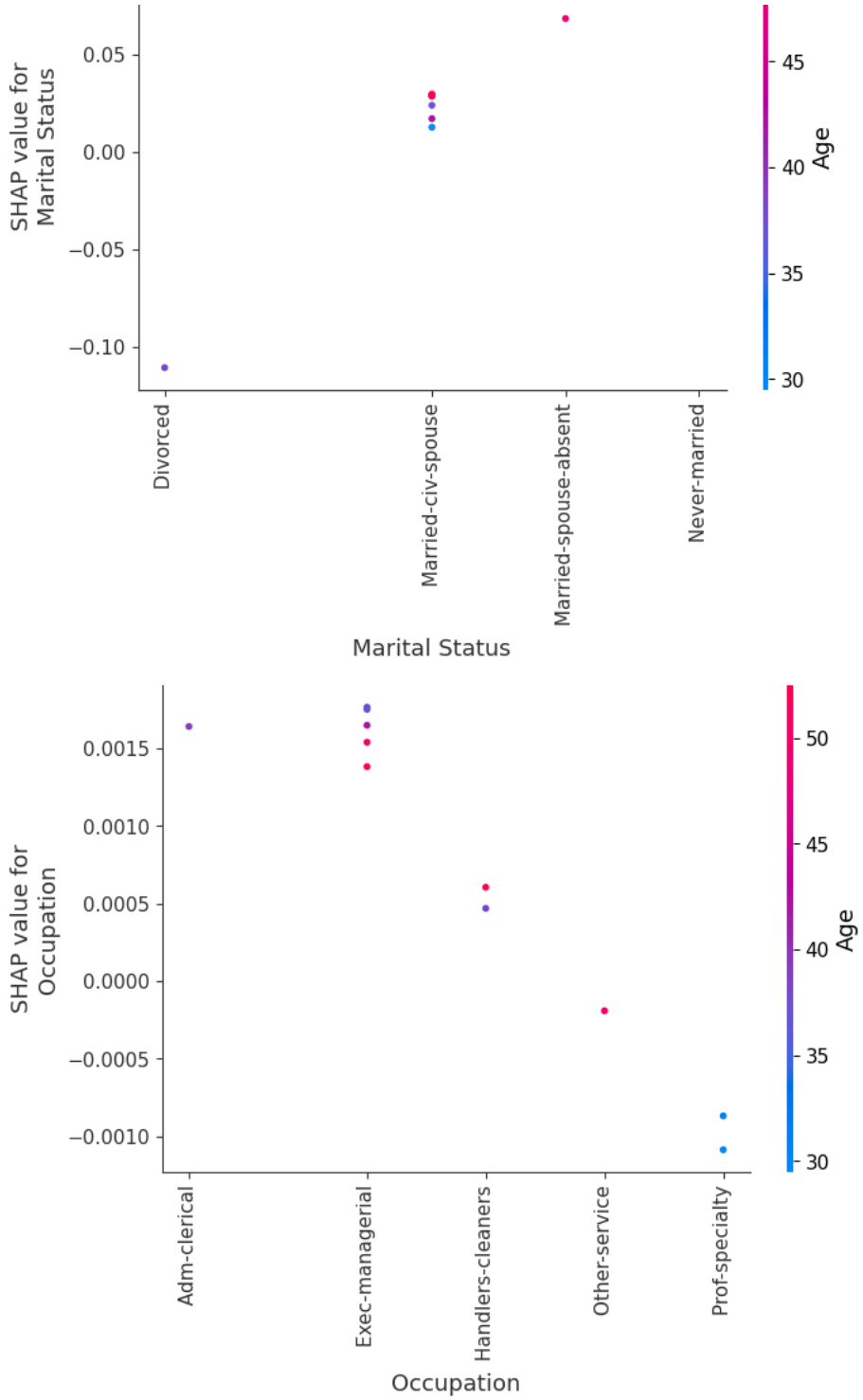
```
# Iterate over the X_train.columns and for each feature, plot the shap.dependence_plot

for name in X_train.columns:
    shap.dependence_plot(name,
                         shap_values=shap_values_ndarray_kernel, # shapley values of all instances
                         features=sample_data, # sample_data
                         display_features=X_display) # to display meaningful features use X_display
```


SHAP value for Age
0.003
0.002
0.001
0.000
−0.001
−0.002
−0.003
30
35
40
45
50
Age

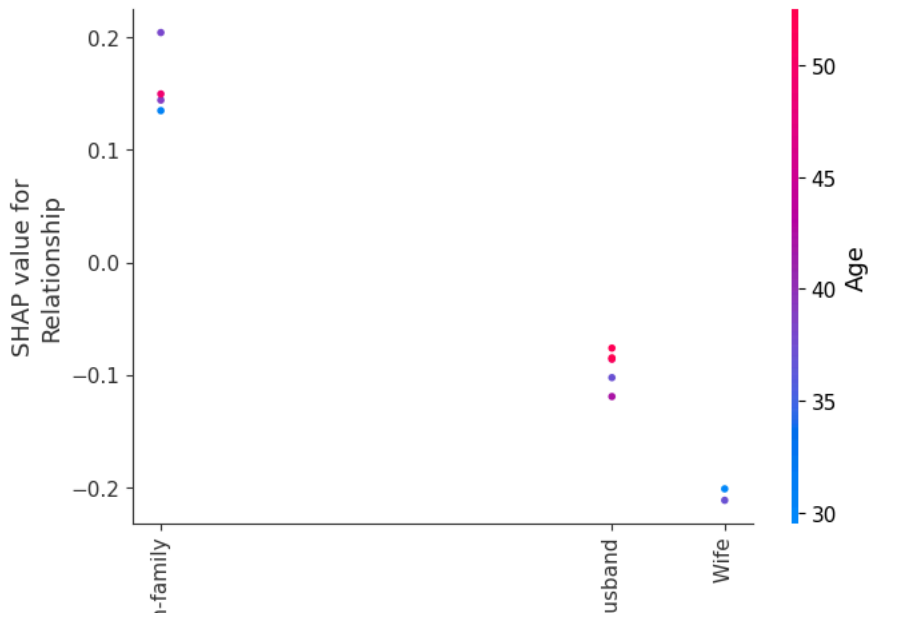

SHAP value for Workclass
0.06
0.05
0.04
0.03
0.02
0.01
0.00
Private
Self-emp-not-inc
State-gov
Workclass
Age
50
45
40
35
30

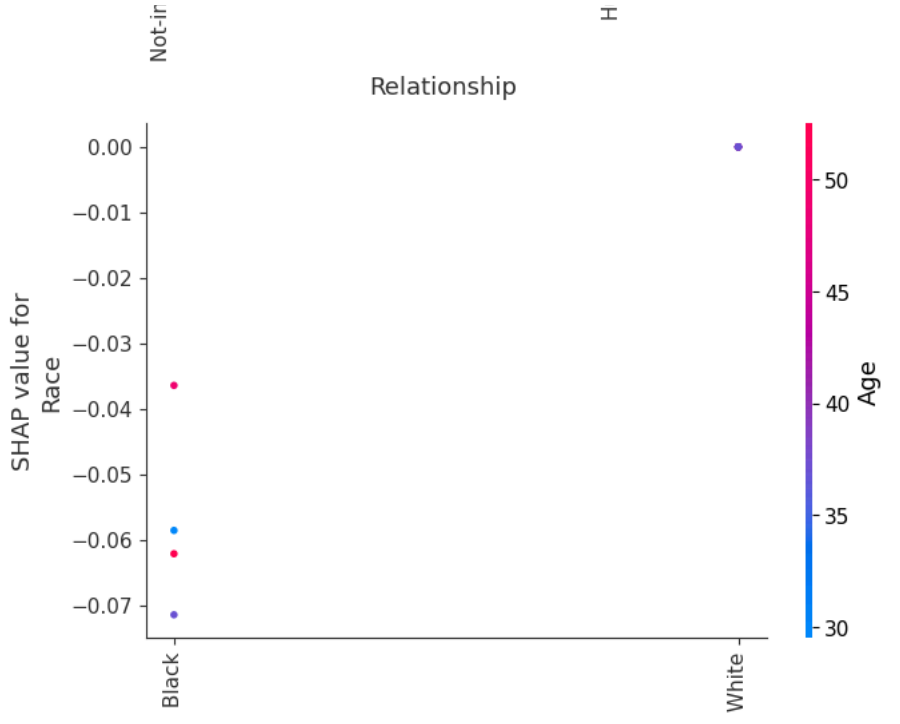

SHAP value for Education-Num
0.05
0.00
−0.05
−0.10
6
8
10
12
14
Education-Num
Age
50
45
40
35
30

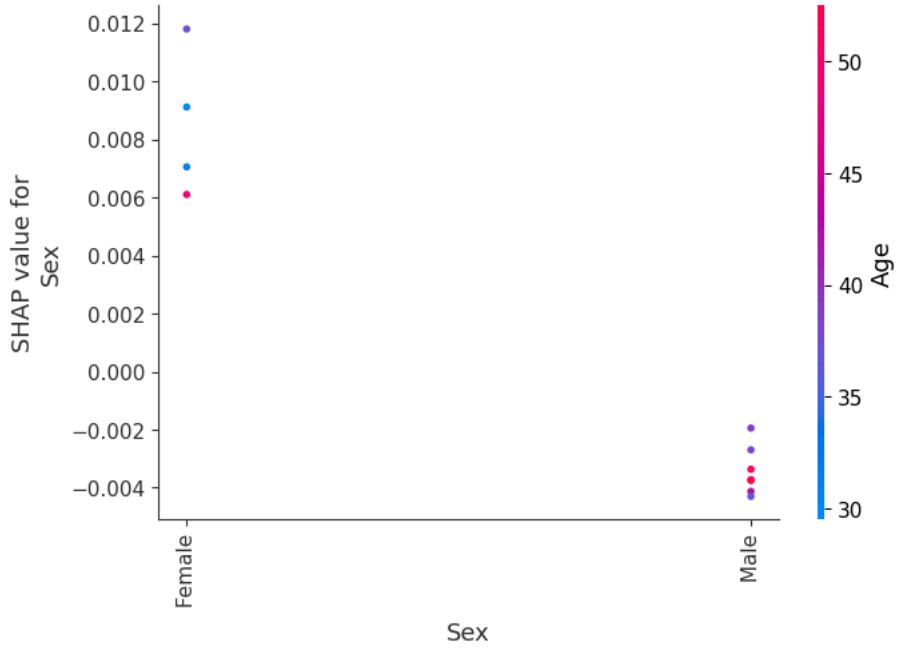

0.10
50

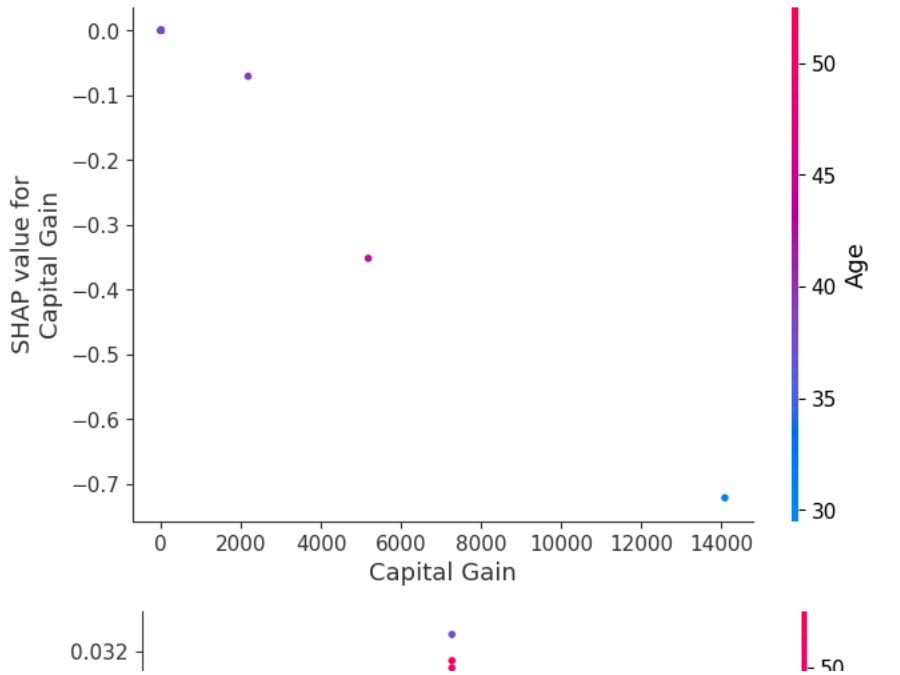

SHAP value for Marital Status
0.05
0.00
−0.05
−0.10
Divorced
Married-civ-spouse
Married-spouse-absent
Never-married
Marital Status
Age
45
40
35
30

SHAP value for Occupation
0.0015
0.0010
0.0005
0.0000
−0.0005
−0.0010
Adm-clerical
Exec-managerial
Handlers-cleaners
Other-service
Prof-specialty
Occupation
Age
50
45
40
35
30

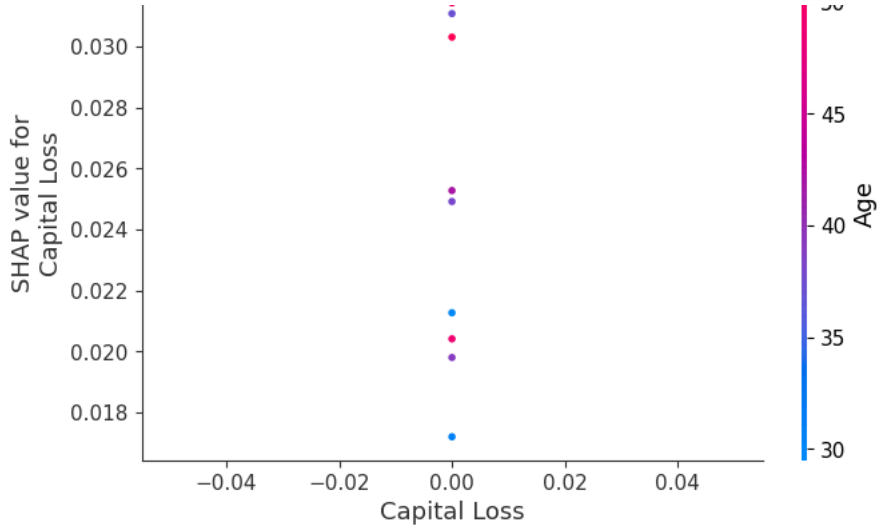

SHAP value for Relationship
0.2
0.1
0.0
−0.1
−0.2
usband
Wife
Age
50
45
40
35
30

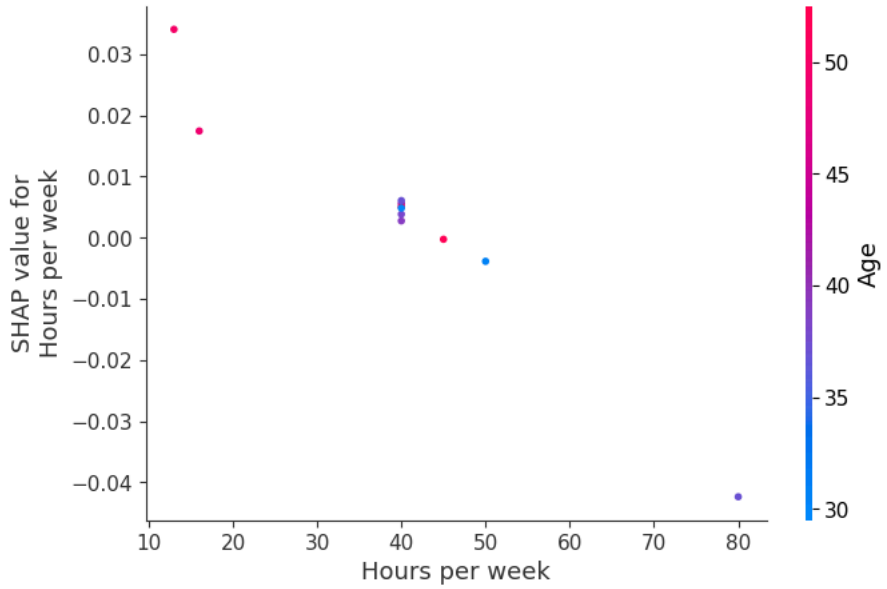

Not-in
H
Relationship
SHAP value for Race
0.00
−0.01
−0.02
−0.03
−0.04
−0.05
−0.06
−0.07
Black
White
Race
Age
50
45
40
35
30

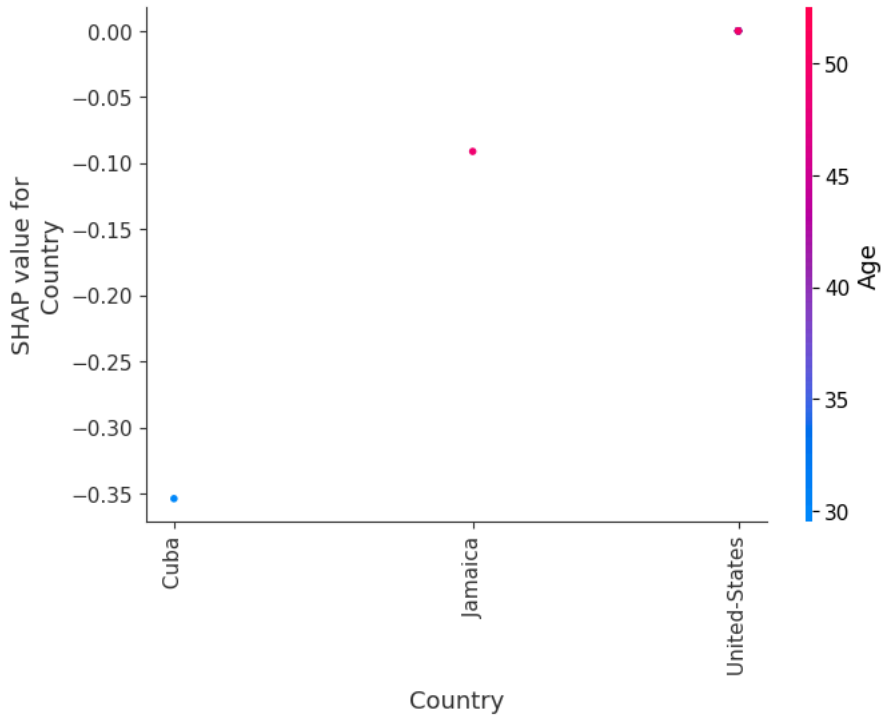

SHAP value for Sex
0.012
0.010
0.008
0.006
0.004
0.002
0.000
−0.002
−0.004
Female
Male
Sex
Age
50
45
40
35
30


SHAP value for Capital Gain
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
0
2000
4000
6000
8000
10000
12000
14000
Capital Gain
Age
50
45
40
35
30

0.032
50







6.) ExactExplainer

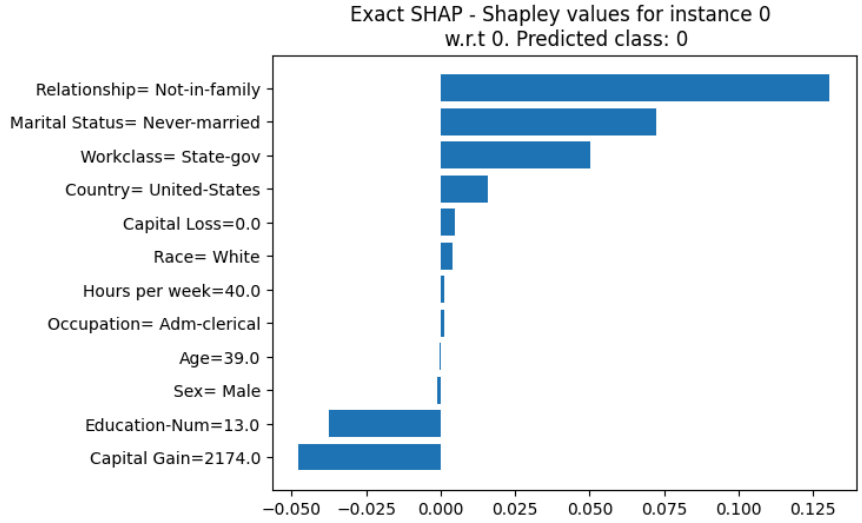
```
sample_X_display = X_display.loc[:100]


# Now we want to use the ExactExplainer. Since the computation will be too long, now we explain only the first two instances
sample_data = X.loc[0:1]
masker = shap.maskers.Independent(data = X_train)
exact_explainer = shap.ExactExplainer(rf_clf.predict_proba, masker=masker)

shap_values_explanation_exact = exact_explainer(sample_data)
print(type(shap_values_explanation_exact))
print(shap_values_explanation_exact[0])
print('-' * 80)

# Class we want to explain
class_index = 0

shap_values_ndarray_exact= shap_values_explanation_exact.values[:, :, class_index]
print(type(shap_values_ndarray_exact))
print(shap_values_ndarray_exact[0])

# Calculate the expected_value
expected_value_exact = shap_values_explanation_exact.base_values[0][class_index] # this is the expected value for the class
```

```
<class 'shap._explanation.Explanation'>
.values =
array([[-0.00020007,  0.00020007],
       [ 0.05019656, -0.05019656],
       [-0.03742185,  0.03742185],
       [ 0.0723445 , -0.0723445 ],
       [ 0.00118951, -0.00118951],
       [ 0.13045432, -0.13045432],
       [ 0.00408033, -0.00408033],
       [-0.00107562,  0.00107562],
       [-0.04776241,  0.04776241],
       [ 0.00466242, -0.00466242],
       [ 0.00122117, -0.00122117],
       [ 0.01584499, -0.01584499]])

.base_values =
array([0.77300948, 0.22699052])

.data =
array([3.900e+01, 7.000e+00, 1.300e+01, 4.000e+00, 1.000e+00, 0.000e+00,
       4.000e+00, 1.000e+00, 2.174e+03, 0.000e+00, 4.000e+01, 3.900e+01])
--------------------------------------------------------------------------------
<class 'numpy.ndarray'>
[-0.00020007  0.05019656 -0.03742185  0.0723445   0.00118951  0.13045432
  0.00408033 -0.00107562 -0.04776241  0.00466242  0.00122117  0.01584499]
```

Exact SHAP - Shapley values

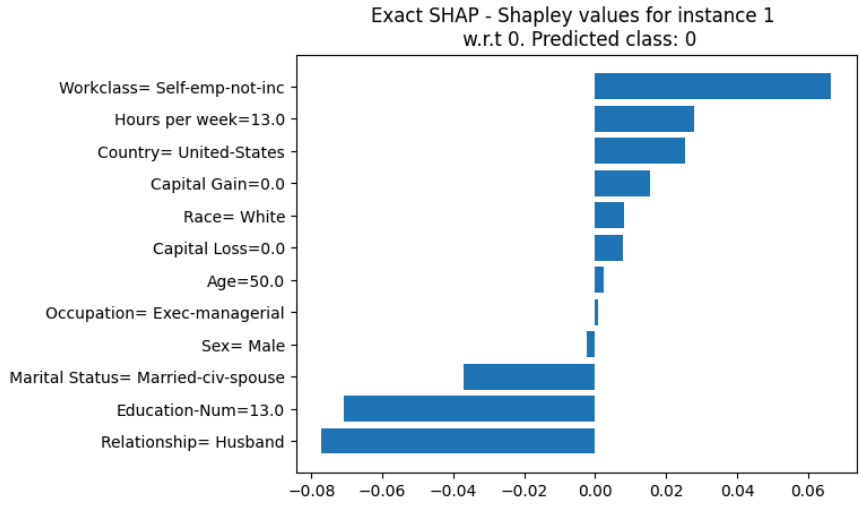
```
id_instance = 0

# Sort feature indices based on SHAP values using np.argsort()
sorted_indices = np.argsort(shap_values_ndarray_exact[id_instance])

# Get feature names and values for the instance
feature_names_values = np.array([f'{f}={value}' for f, value in zip(sample_X_display.columns, sample_X_display.iloc[id_insta

# Plot SHAP values
plt.barh(feature_names_values[sorted_indices], shap_values_ndarray_exact[id_instance][sorted_indices])

# Predict class for the instance
predicted_class = int(rf_clf.predict(sample_data.iloc[id_instance:id_instance+1])[0])

# Plot title
plt.title(f'Exact SHAP - Shapley values for instance {id_instance} \n w.r.t {class_index}. Predicted class: {predicted_class
```

```
Text(0.5, 1.0, 'Exact SHAP - Shapley values for instance 0 \n w.r.t 0.
Predicted class: 0')
```




```
id_instance = 1

# Sort feature indices based on SHAP values
sorted_indices = np.argsort(shap_values_ndarray_exact[id_instance])

# Get feature names and values for the instance
feature_names_values = np.array([f'{f}={value}' for f, value in zip(sample_X_display.columns, sample_X_display.iloc[id_insta

# Plot SHAP values
plt.barh(feature_names_values[sorted_indices], shap_values_ndarray_exact[id_instance][sorted_indices])

# Predict class for the instance
predicted_class = int(rf_clf.predict(sample_data.iloc[id_instance:id_instance+1])[0])

# Plot title
plt.title(f'Exact SHAP - Shapley values for instance {id_instance} \n w.r.t {class_index}. Predicted class: {predicted_class
```

```
Text(0.5, 1.0, 'Exact SHAP - Shapley values for instance 1 \n w.r.t 0.
Predicted class: 0')
```



Exact SHAP - force_plot