

✓ 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 (SHapley Additive exPlanations) 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

Passaggi successivi:

[Visualizza grafici consigliati](#)

[New interactive sheet](#)

```
# Load the data with display=True. This dataset still contains the categorical va
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_s
```

4.) Train the Random Forest Classifier

```
# Train a RandomForestClassifier

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



```

▼ RandomForestClassifier ⓘ ?
RandomForestClassifier()

```

```

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

```



```
0.8565945033010901
```

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 v

# 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 th

```

```

⇒ PermutationExplainer explainer: 102it [01:34, 1.03it/s]
<class 'shap._explanation.Explanation'>
.values =
array([[ 0.00604605, -0.00604605],
       [ 0.01729032, -0.01729032],
       [-0.04568034,  0.04568034],
       [ 0.08078157, -0.08078157],
       [ 0.04778235, -0.04778235],
       [ 0.09340875, -0.09340875],
       [ 0.00045448, -0.00045448],
       [-0.00885997,  0.00885997],
       [ 0.0340021 , -0.0340021 ],
       [ 0.00367011, -0.00367011],
       [ 0.02472258, -0.02472258],
       [ 0.00100852, -0.00100852]])

.base_values =
array([0.72537348, 0.27462652])

.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])
```

```
-----
PermutationExplainer explainer: 102it [00:17, 2.46it/s]<class 'numpy.ndarray'
 [ 0.00082289  0.02419236 -0.03964601  0.07707543  0.01683308  0.10708925
  0.0012525  -0.00419062  0.035145     0.00349484  0.03068116  0.00187663]
```

```
shap_values_ndarray.shape
```

```
⇒ (101, 12)
```

```
shap_values_ndarray[0]
```

```
⇒ array([ 0.00082289,  0.02419236, -0.03964601,  0.07707543,  0.01683308,
          0.10708925,  0.0012525 , -0.00419062,  0.035145   ,  0.00349484,
          0.03068116,  0.00187663])
```

✓ 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]

⇒ PermutationExplainer explainer: 102it [00:30, 2.17it/s]

id_instance = 1

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

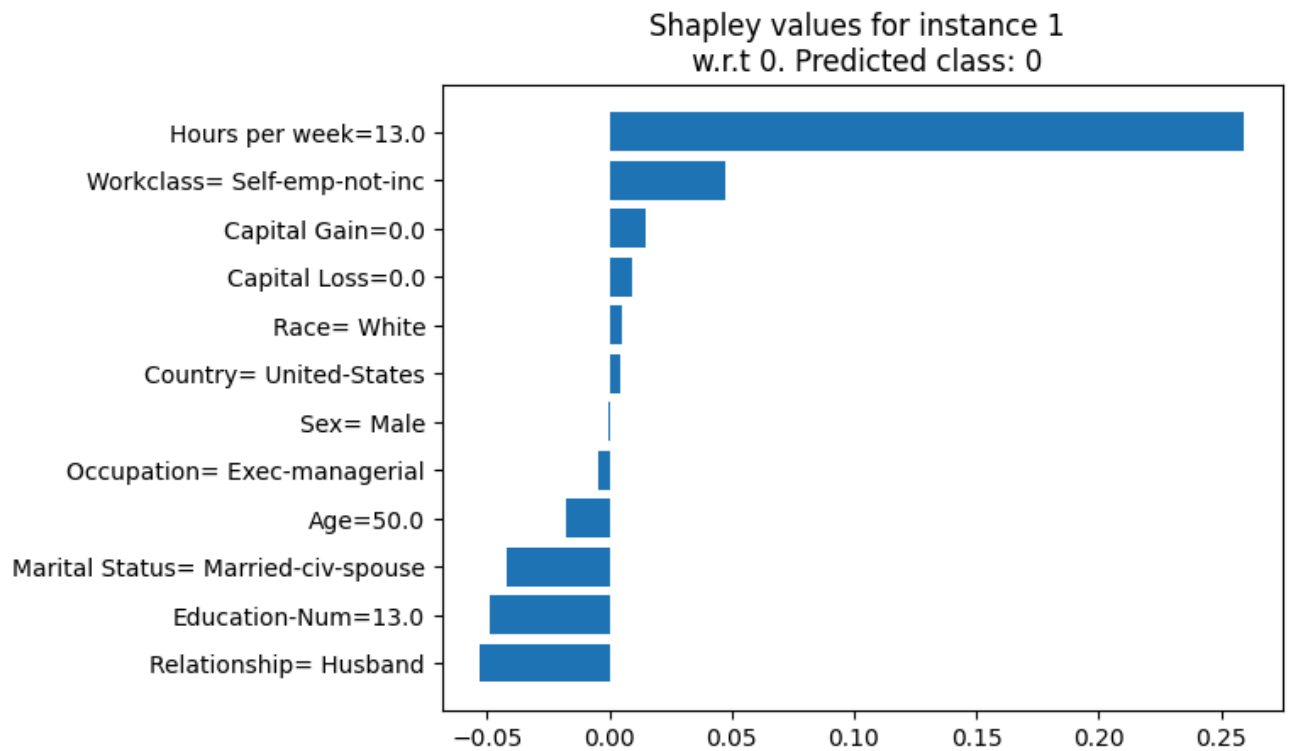
# Get feature names and values for the instance
feature_names_values = np.array([f'{f}={value}' for f, value in zip(sample_X_disp

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

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

# Plot title
plt.title(f'Shapley values for instance {id_instance} \n w.r.t {class_index}. Pre
```

Text(0.5, 1.0, 'Shapley values for instance 1 \n w.r.t 0. 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_disp
```

```
# Plot SHAP values
```

```
plt.barh(feature_names_values[sorted_indices], shap_values_ndarray[id_instance][s
```

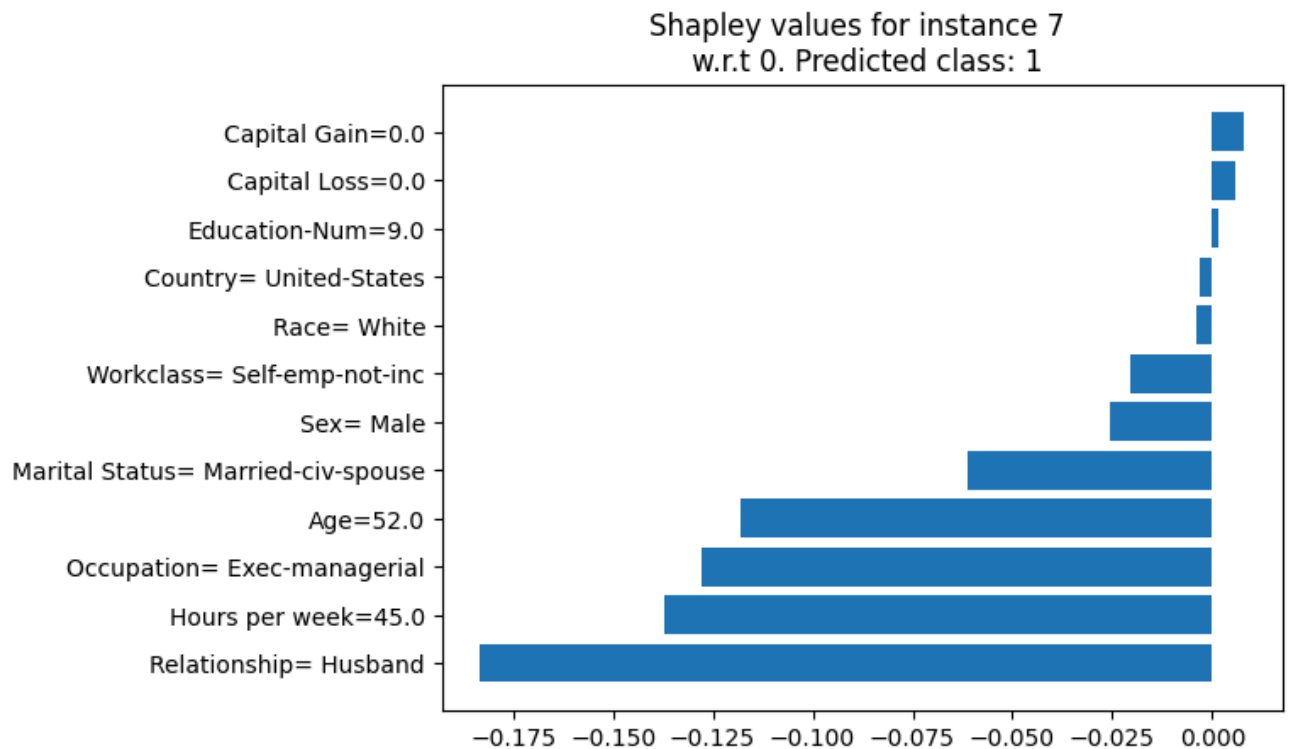
```
# Predict class for the instance
```

```
predicted_class = int(rf_clf.predict(sample_data.iloc[id_instance:id_instance+1])
```

```
# Plot title
```

```
plt.title(f'Shapley values for instance {id_instance} \n w.r.t {class_index}. Pre
```

⇒ Text(0.5, 1.0, 'Shapley values for instance 7 \n w.r.t 0. Predicted class: 1')



SHAP Force plot - single instance

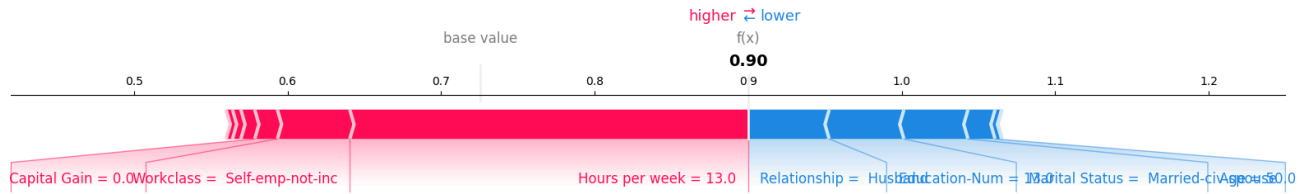
`shap.force_plot()` shows bars that indicate the **magnitude** and **direction** of the **influence of the features on the model prediction**. The graph also includes the `base_value`, which represents the average outcome of the model in the dataset, and an arrow indicating the final predicted value $f(x)$ 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 sha
                features=X_display.iloc[id_instance, :], # Use the X_display data
                matplotlib=True)
```


↔ [0.9 0.1]



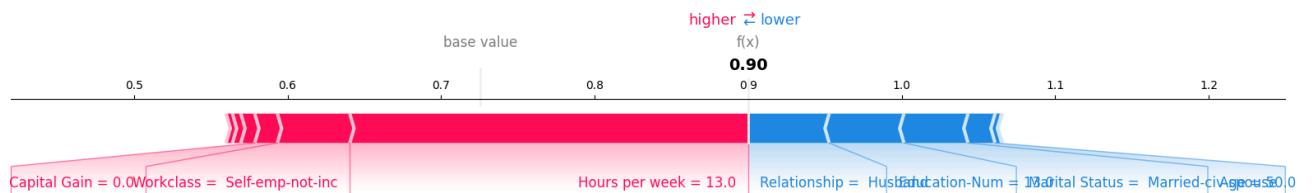
```
# Complete with your code!
```

```
id_instance = 1
```

```
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)
```

↔ [0.9 0.1]

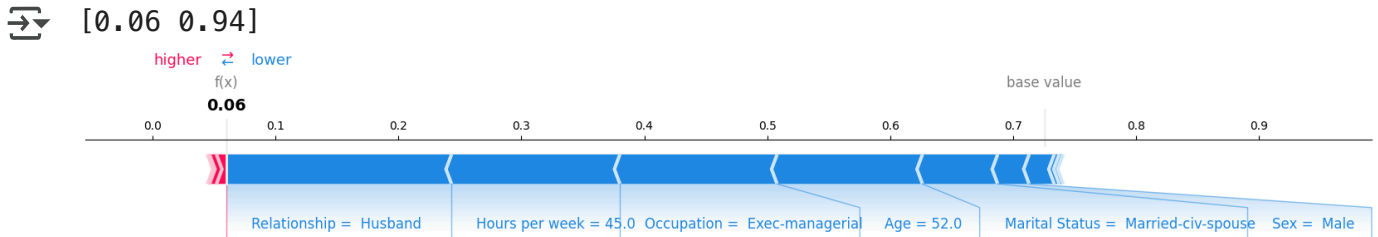


```
id_instance = 7
```

```
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)
```

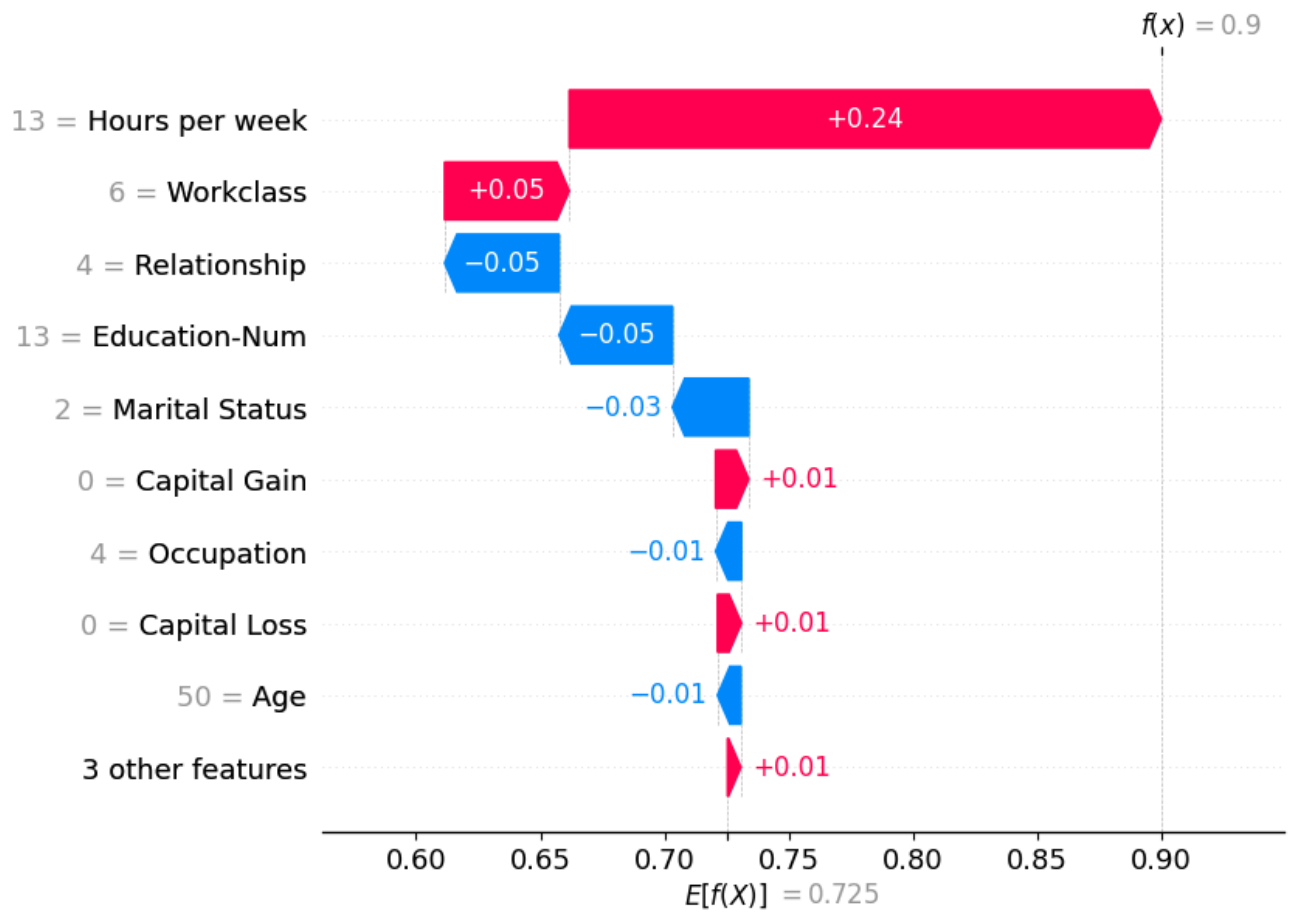


SHAP - waterfall plot

`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.

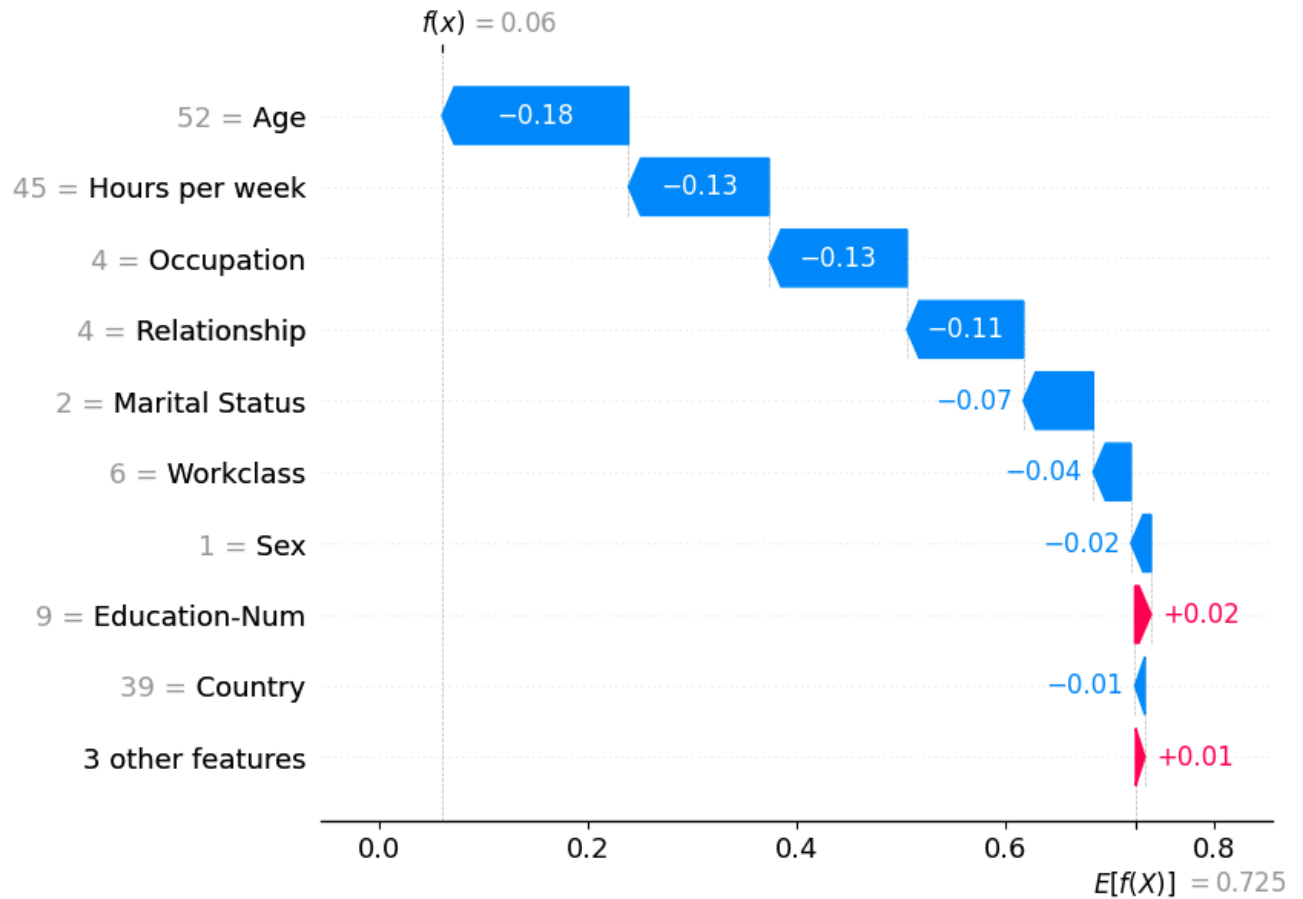
```
id = 1
```

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



```
id = 7
```

```
shap.waterfall_plot(shap_values=shap_values_explanation[id, :, class_index], max_
```



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

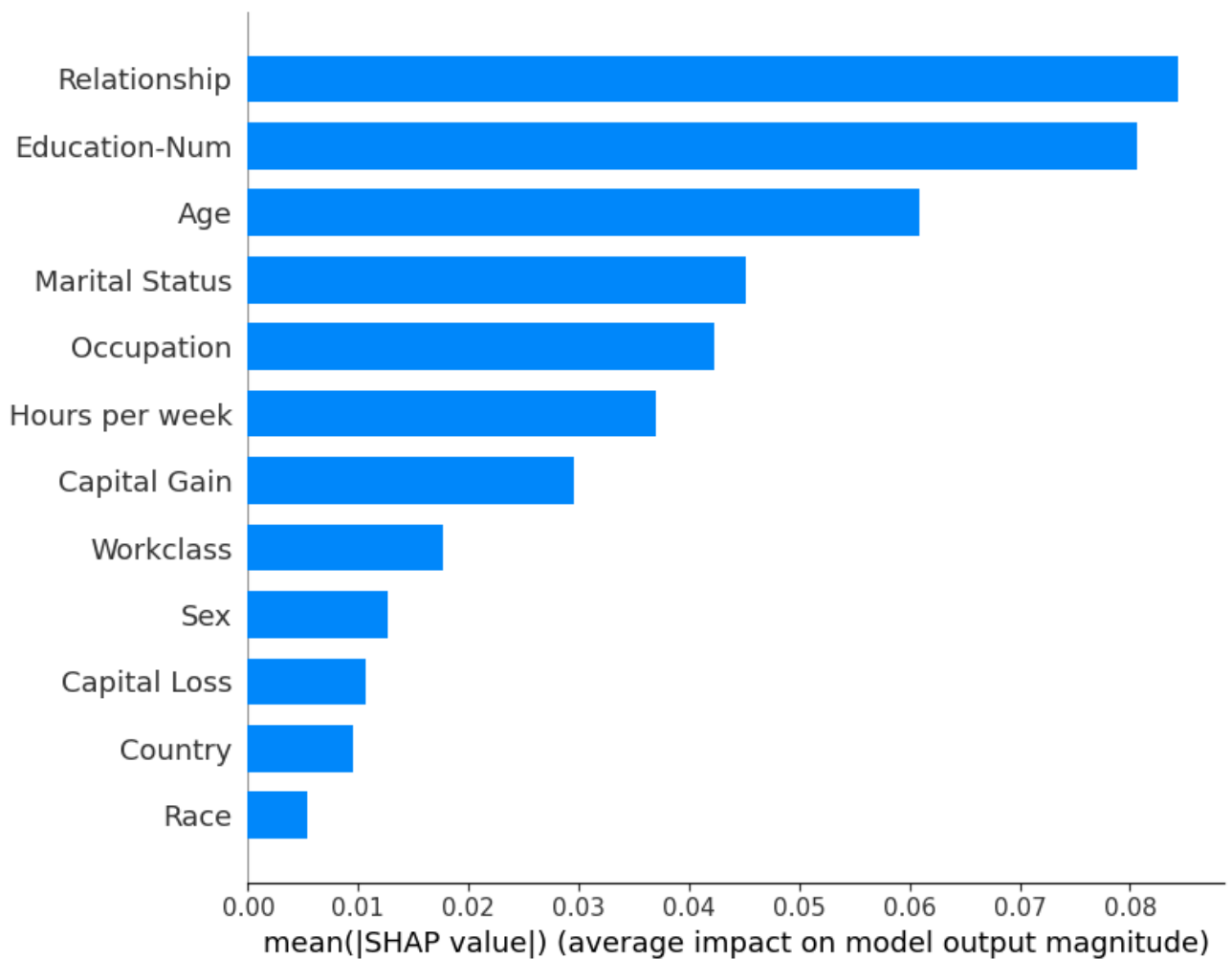
SHAP Summary plot

- 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_disp
                 plot_type='bar')
```

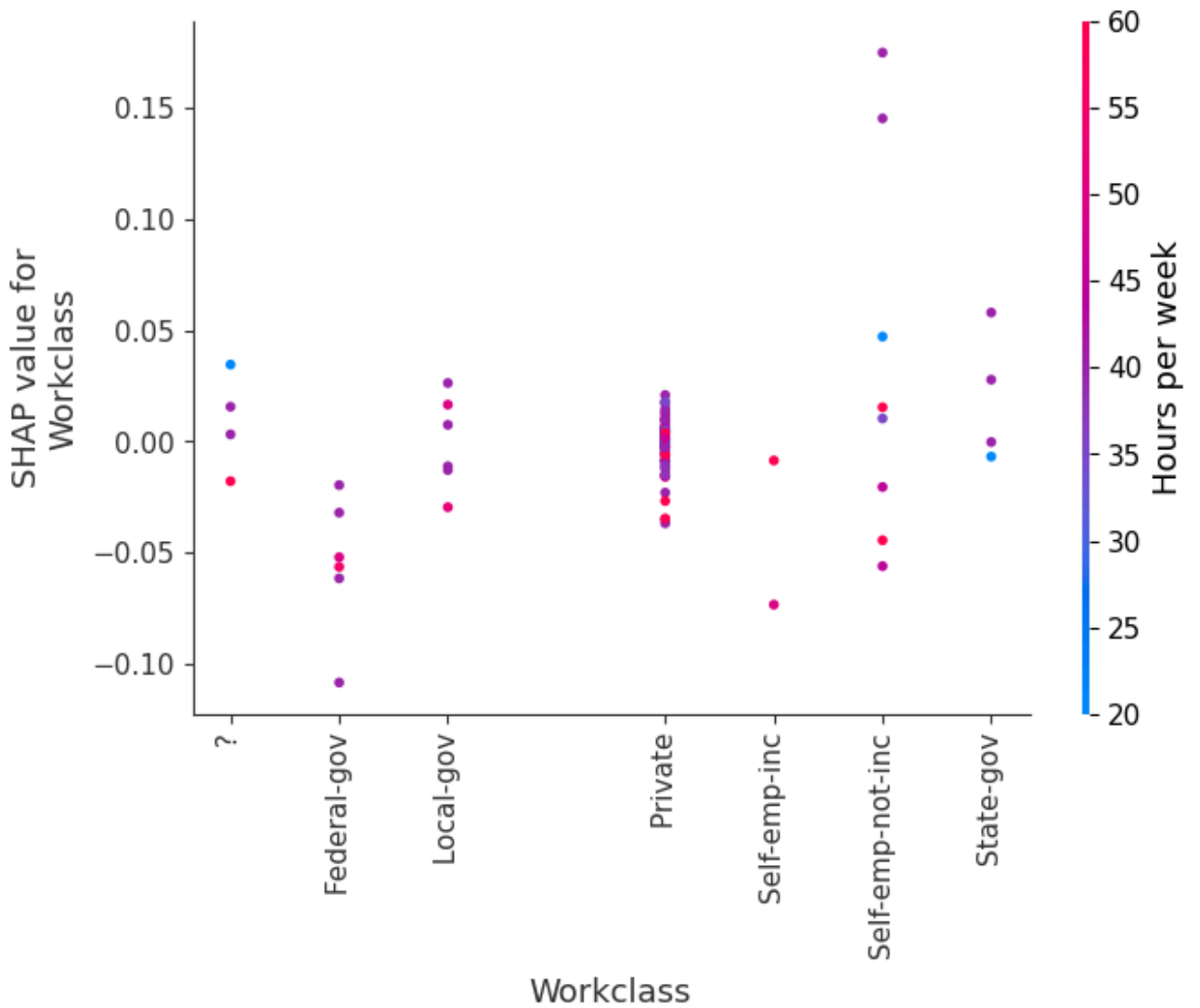
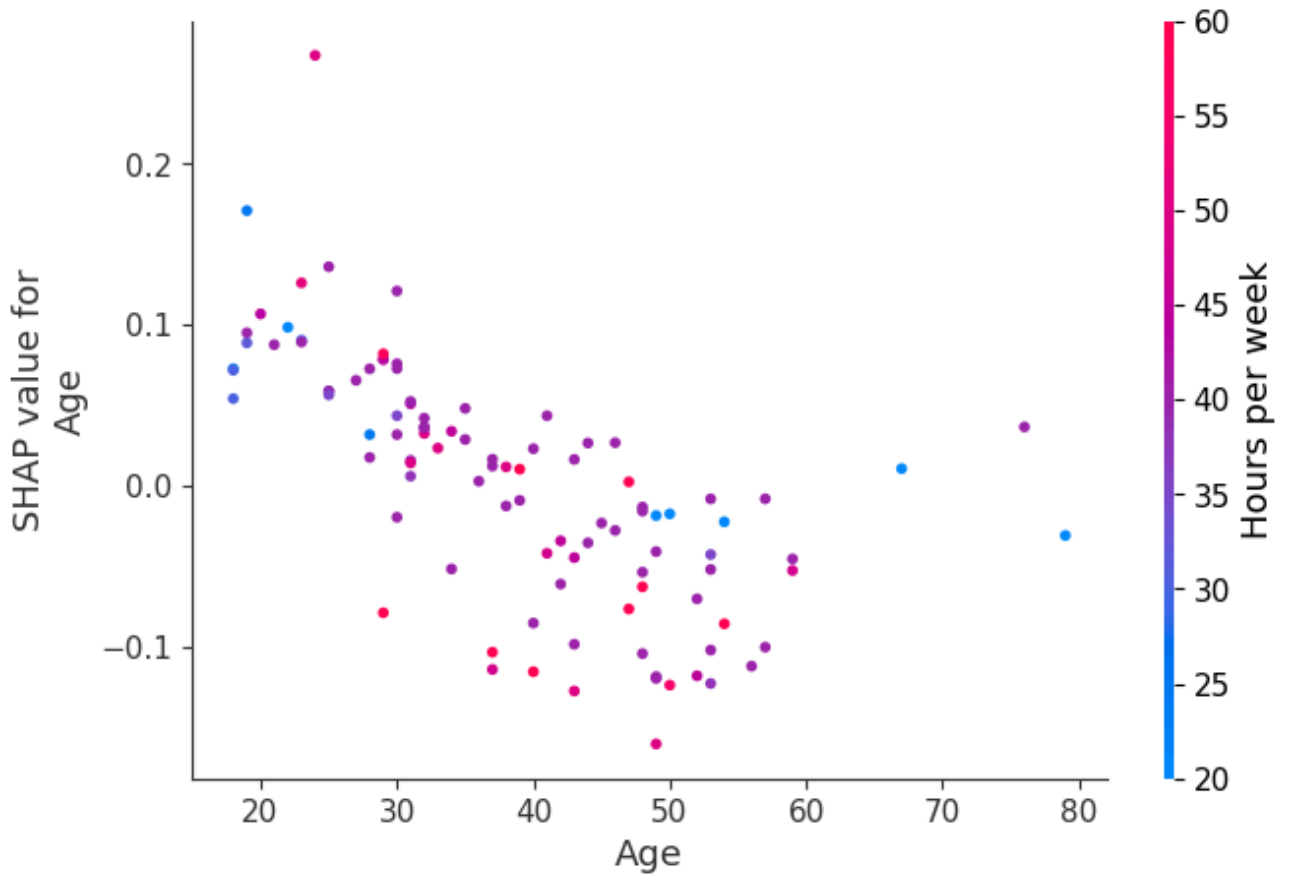
 <ipython-input-21-626a25bfb89d>:2: FutureWarning: The NumPy global RNG was set
shap.summary_plot(shap_values=shap_values_ndarray,



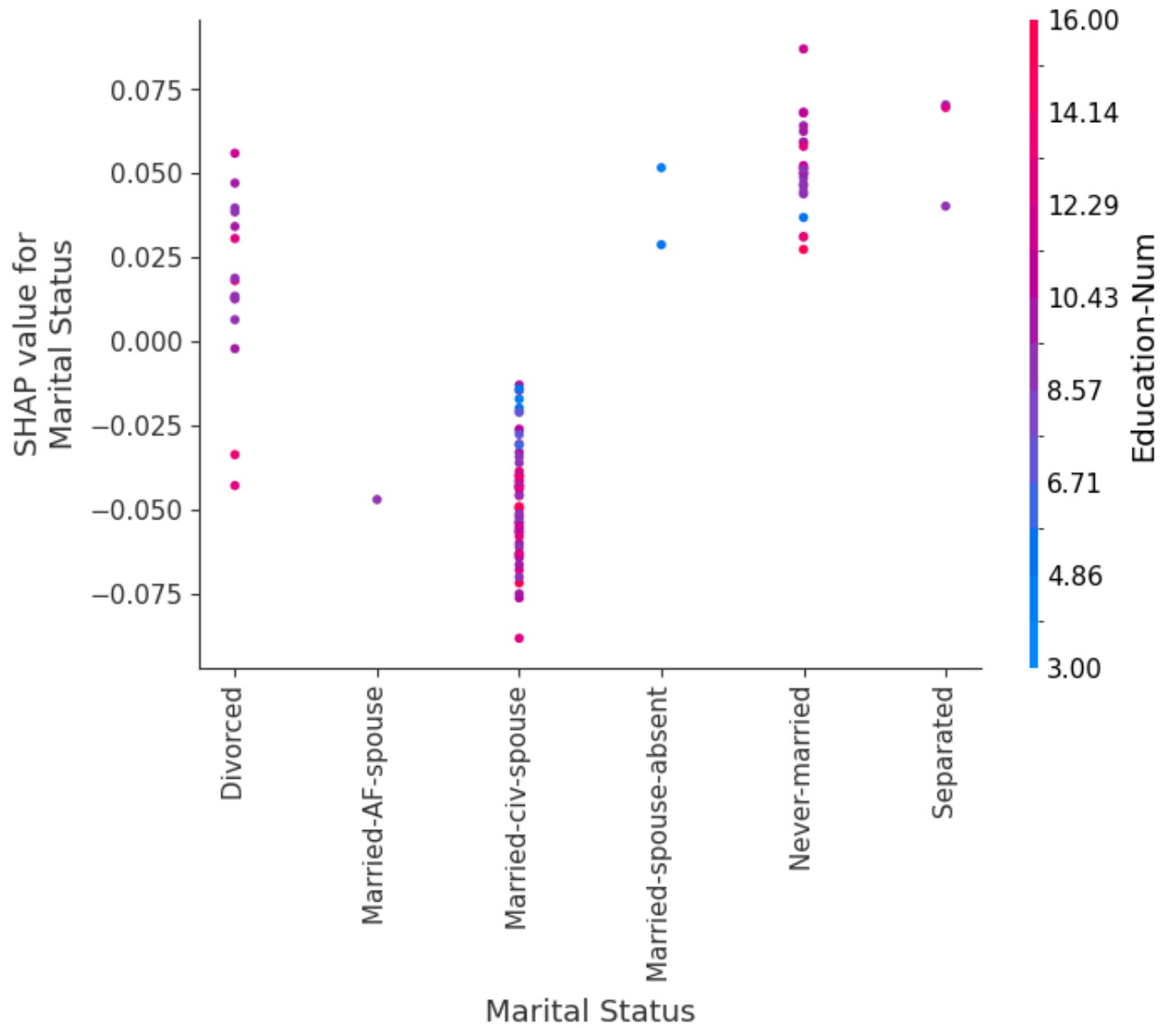
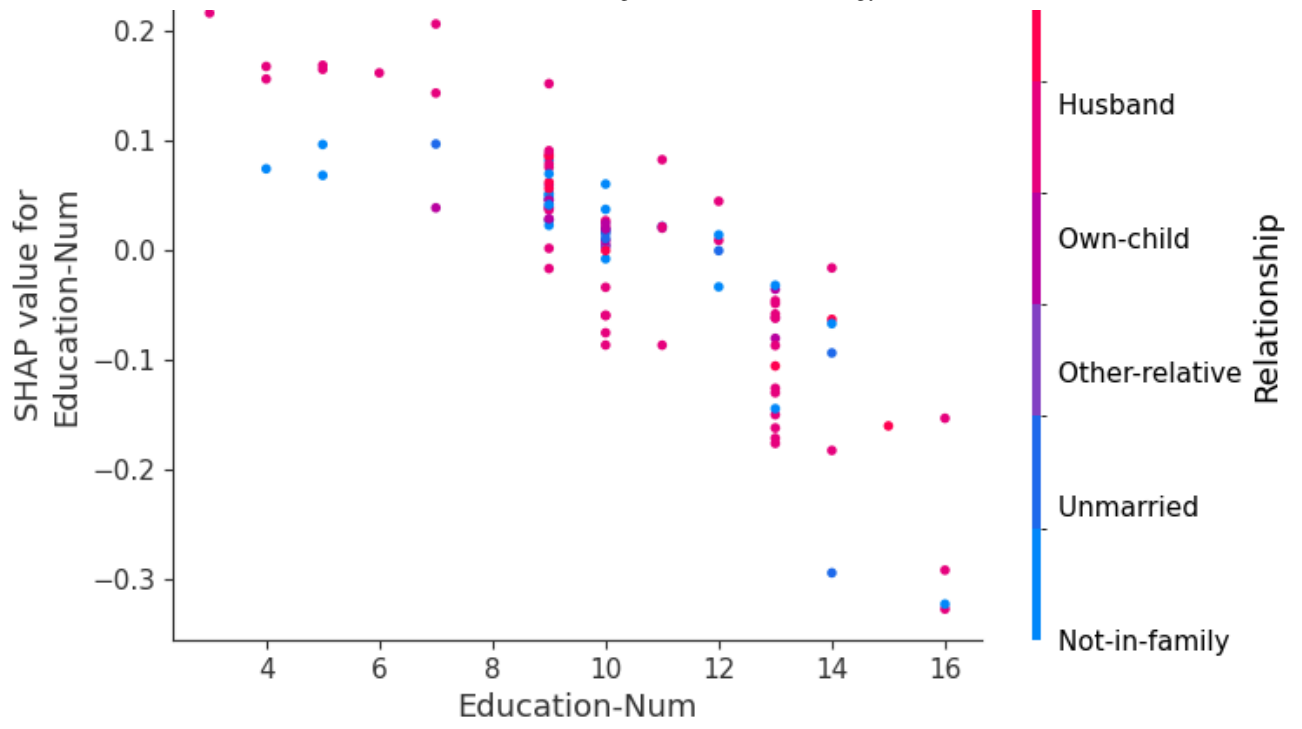
SHAP dependence plot

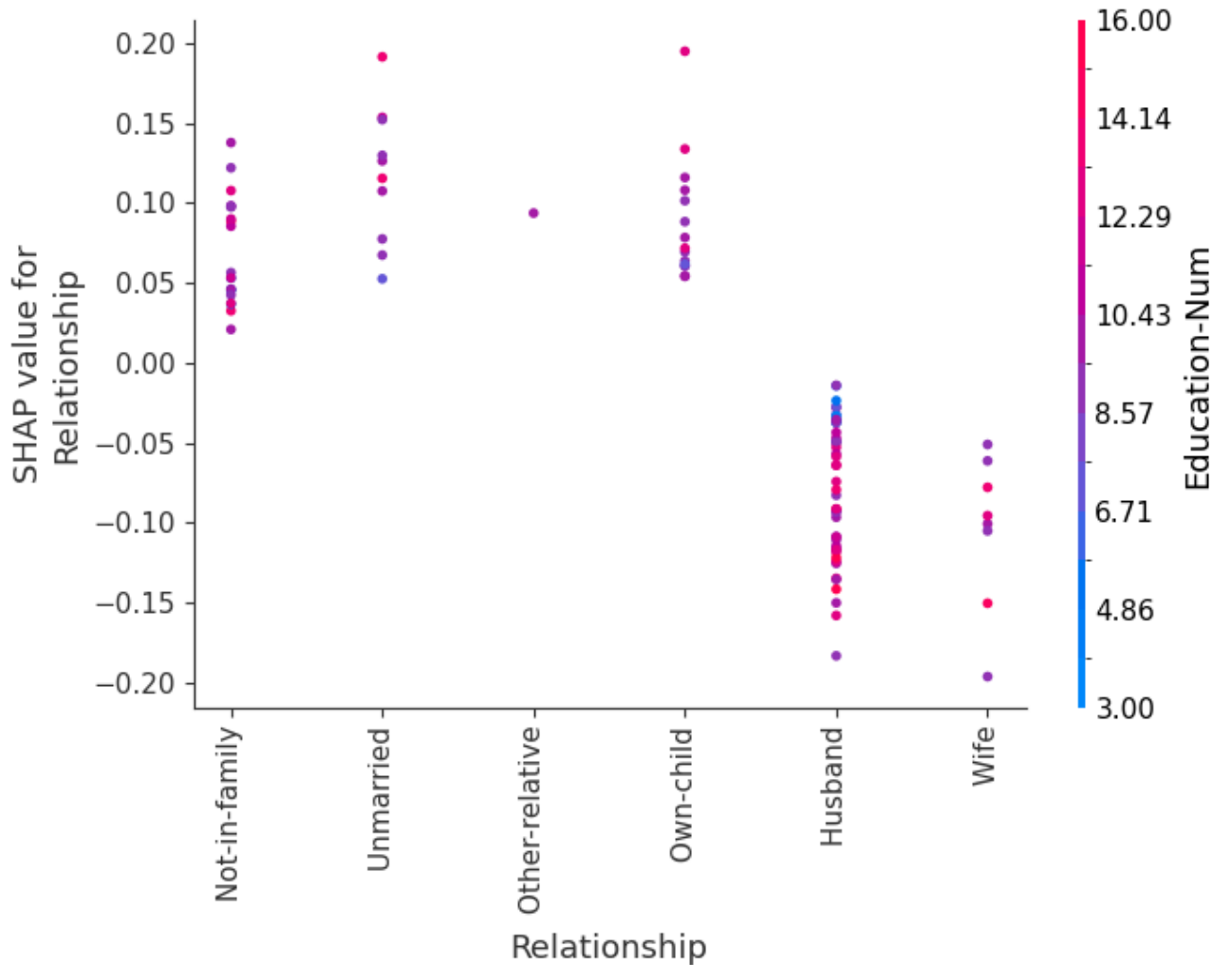
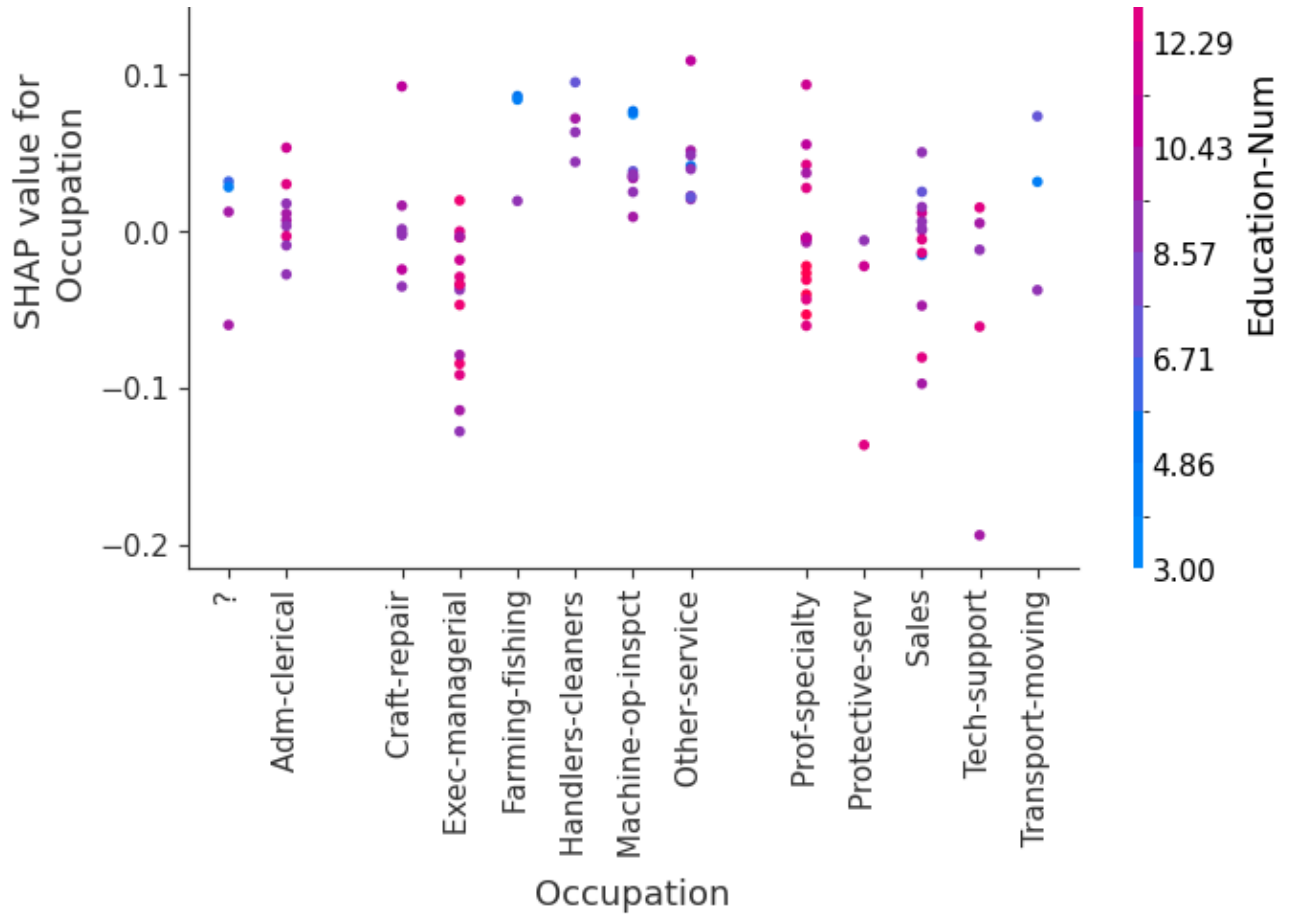
The graph generated by `shap.dependence_plot` typically consists of a scatter plot in which each point represents an instance of the dataset. The x-axis represents the values of the chosen feature, while the y-axis represents the corresponding SHAP values.

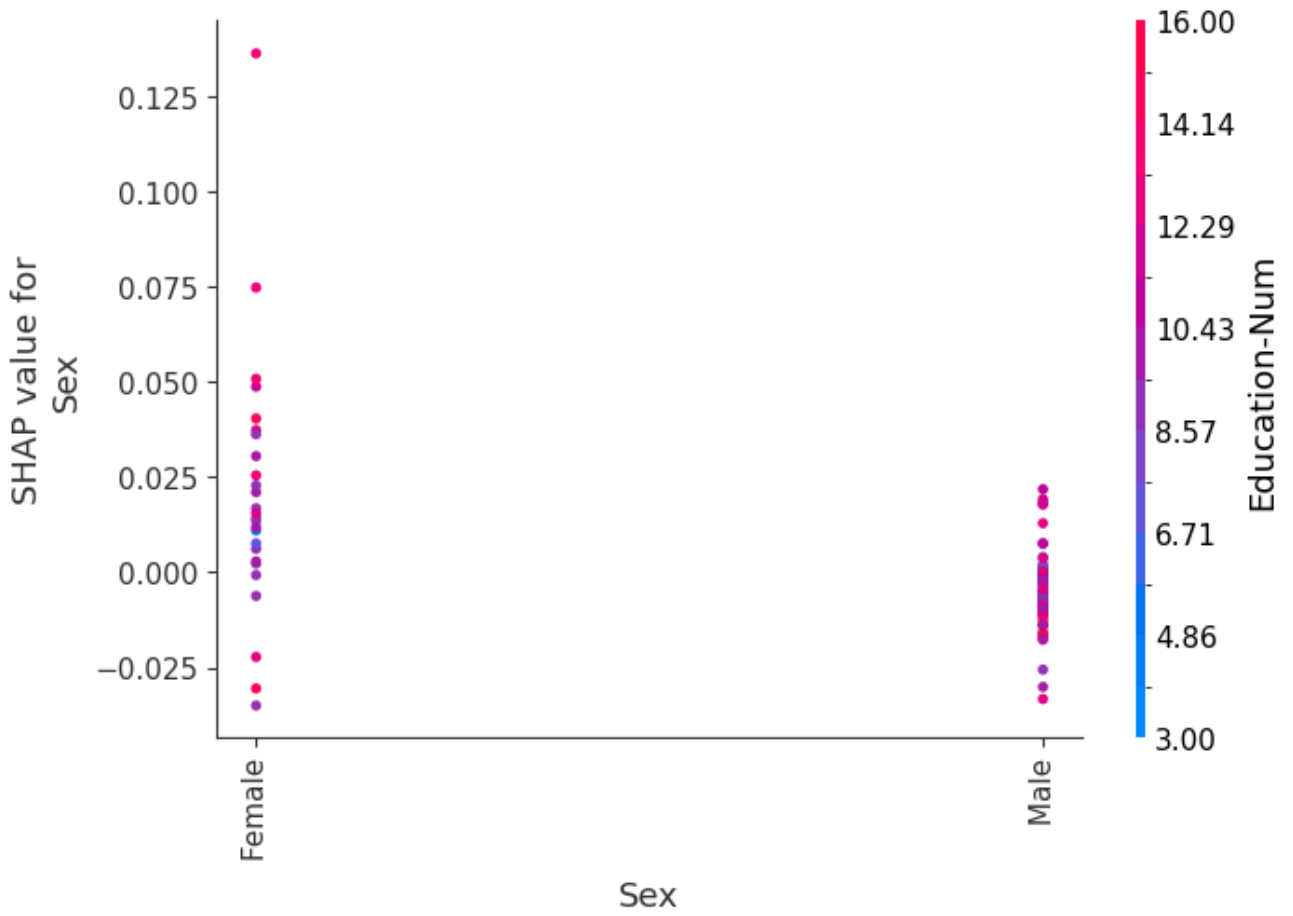
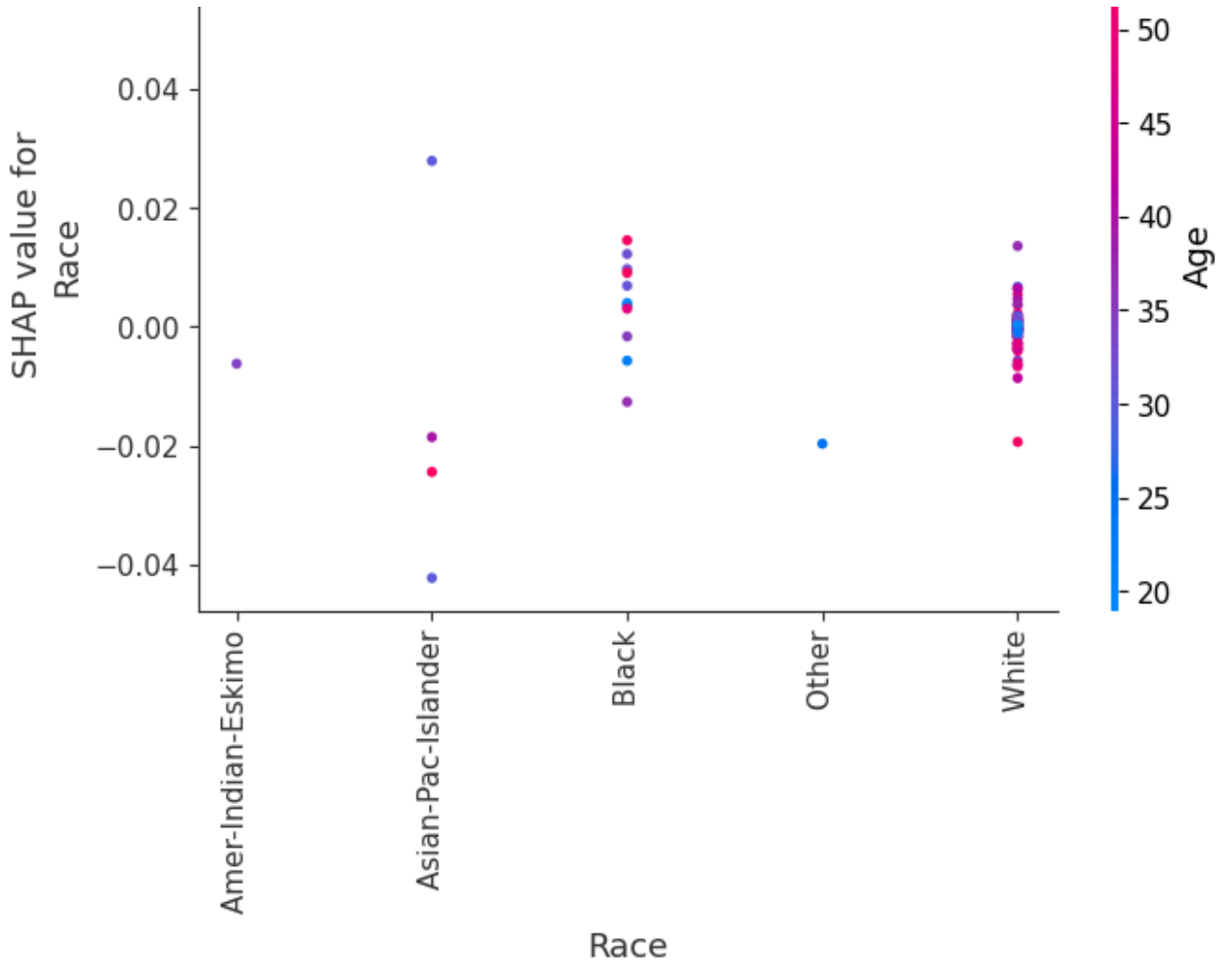
```
# Iterate over the X_train.columns and for each feature, plot the shap.dependence.  
  
for name in X_train.columns:  
    shap.dependence_plot(name,  
                          shap_values=shap_values_ndarray, # shapley values of all  
                          features=sample_data, # sample_data  
                          display_features=X_display) # to display meaningful feat
```

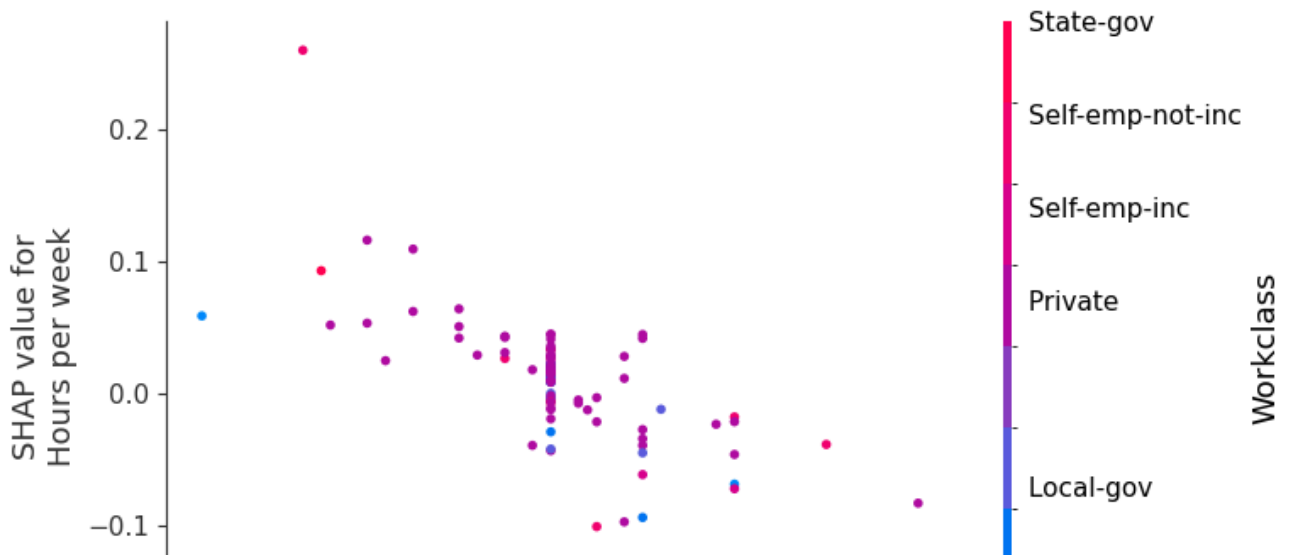
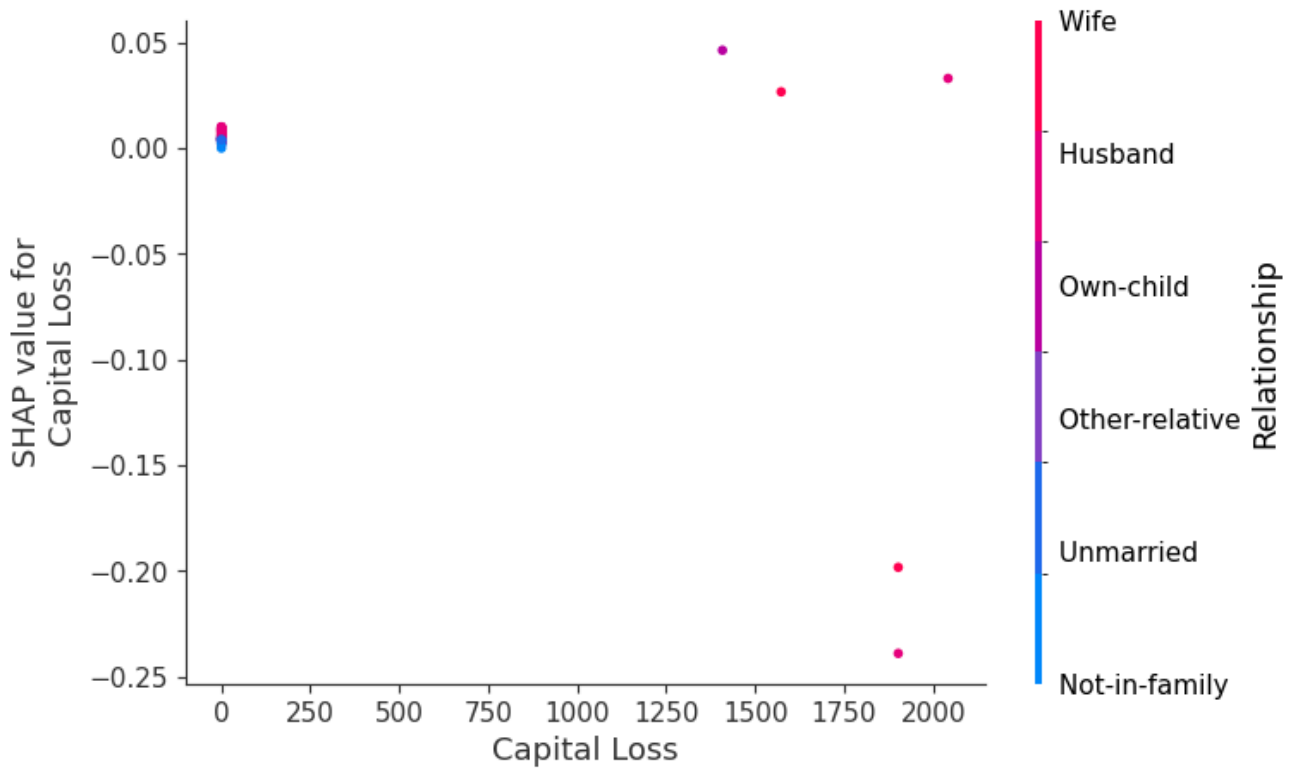
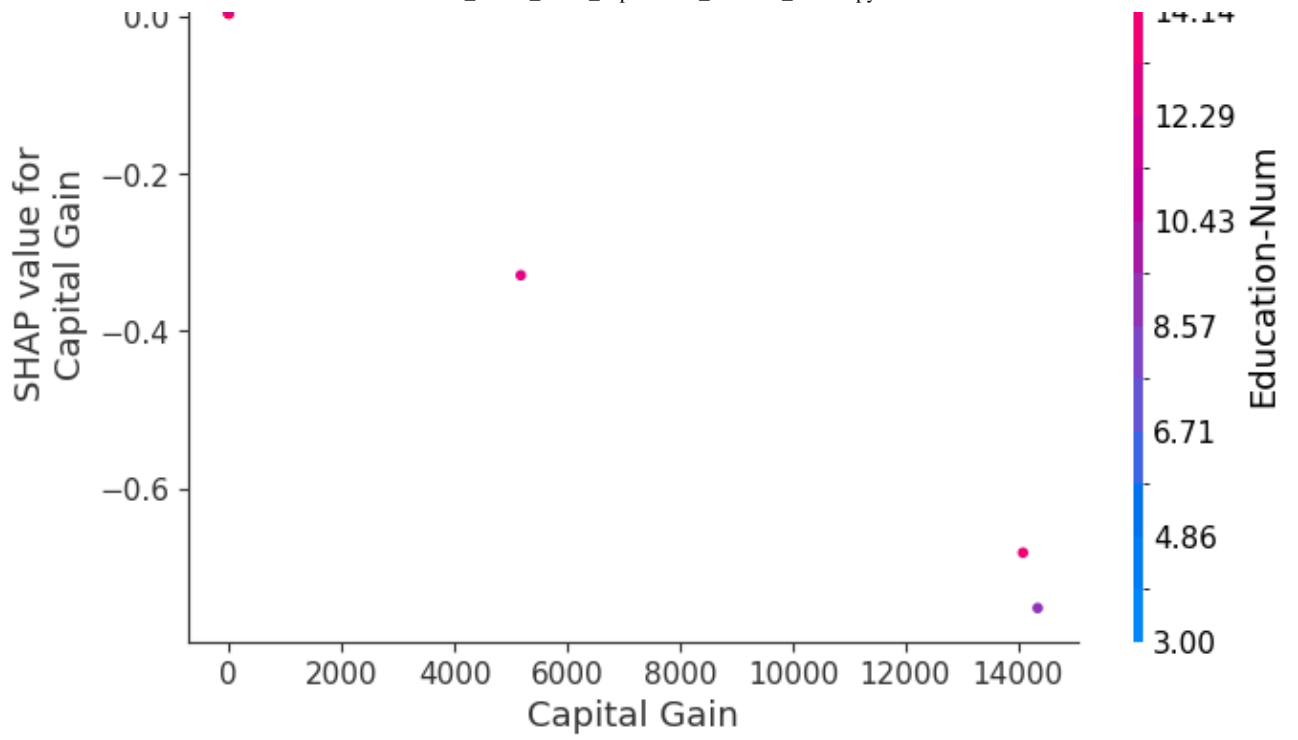


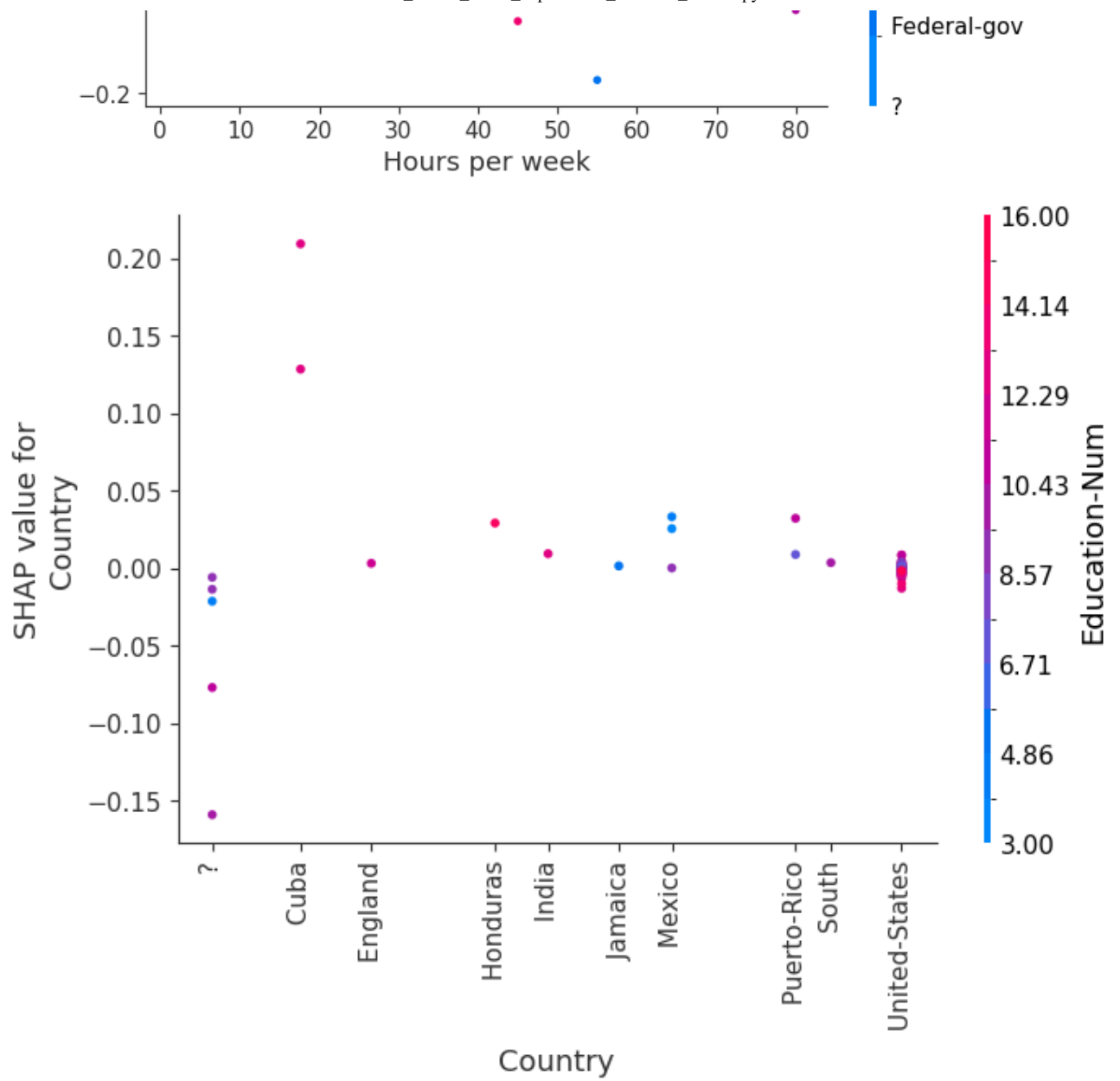
Wife











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_ke

# 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 v

shap_values_ndarray_kernel = kernel_explainer.shap_values(sample_data)[: , :, clas

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

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



100%

11/11 [00:01<00:00, 7.11it/s]

```

<class 'shap._explanation.Explanation'>
.values =
array([[ -0.00375317,  0.00375317],
       [ 0.02342903, -0.02342903],
       [-0.07588441,  0.07588441],
       [ 0.09951508, -0.09951508],
       [ 0.02468019, -0.02468019],
       [ 0.1310738 , -0.1310738 ],
       [ 0.          ,  0.          ],
       [-0.01015901,  0.01015901],
       [ 0.02216042, -0.02216042],
       [ 0.02722185, -0.02722185],
       [ 0.05593287, -0.05593287],
       [ 0.          ,  0.          ]])

.base_values =
array([0.68578333, 0.31421667])

.data =
array([3.9000e+01, 7.0000e+00, 1.3000e+01, 4.0000e+00, 1.0000e+00, 0.0000e+00,
       4.0000e+00, 1.0000e+00, 2.1740e+03, 0.0000e+00, 4.0000e+01, 3.9000e+01])

```

100%

11/11 [00:01<00:00, 7.11it/s]

```

<class 'numpy.ndarray'>
[-0.00375317  0.02342903 -0.07588441  0.09951508  0.02468019  0.1310738
  0.          -0.01015901  0.02216042  0.02722185  0.05593287  0.          ]

```

Expected value

0.6857833333333334

shap_values_ndarray_kernel



```

array([[ -3.75316843e-03,  2.34290269e-02, -7.58844088e-02,
         9.95150845e-02,  2.46801946e-02,  1.31073799e-01,
         0.00000000e+00, -1.01590056e-02,  2.21604244e-02,
         2.72218464e-02,  5.59328741e-02,  0.00000000e+00],
       [ 7.03574483e-03,  6.25126168e-02, -1.01366021e-01,
        -3.34230808e-02, -6.93998637e-03, -5.22989027e-02,
         0.00000000e+00,  1.96048942e-03,  0.00000000e+00,
         6.27422219e-02,  2.73993585e-01,  0.00000000e+00],
       [-9.93398196e-03,  8.14714003e-03,  3.29301087e-02,
         3.94551115e-02,  4.90885005e-02,  1.11712289e-01,
         0.00000000e+00, -1.29831349e-03,  0.00000000e+00,
         3.40976027e-02,  5.00182094e-02,  0.00000000e+00],
       [-3.48929905e-02,  8.69150794e-03,  1.51458834e-01,
        -3.00503551e-02,  1.25030481e-01, -6.03352577e-02,
         1.13453829e-02,  5.17089286e-03,  0.00000000e+00,
         6.50702289e-02,  5.27279422e-02,  0.00000000e+00],
       [ 1.05130627e-01,  5.73654125e-03, -1.19355405e-01,
        -4.34416770e-02,  3.30678761e-02, -1.24407333e-01,
         7.50107555e-02,  0.00000000e+00,  0.00000000e+00,
         4.92872682e-02,  4.51081404e-02,  1.18079873e-01],
       [ 5.94701519e-02, -1.37995421e-03, -3.41292155e-02,
        -4.73925962e-02, -5.95283748e-03, -9.80763316e-02,
         0.00000000e+00,  8.70970752e-02,  0.00000000e+00,

```

```

5.09843311e-02, 8.35960435e-02, 0.00000000e+00],
[-5.98793951e-03, 2.11075229e-03, 7.87598041e-02,
 3.39711396e-02, 3.49781374e-02, 7.06922634e-02,
 1.46311996e-03, 0.00000000e+00, 0.00000000e+00,
 2.69472843e-02, 4.70818920e-02, -5.79978687e-03],
[-1.87046061e-01, -7.37138541e-02, 1.15778855e-02,
-5.93676822e-02, -9.06146846e-02, -1.37104969e-01,
 0.00000000e+00, -2.72039182e-02, 0.00000000e+00,
 4.73356013e-02, -1.09645651e-01, 0.00000000e+00],
[ 3.88840851e-02, -2.11532029e-03, -9.91521715e-02,
 4.22877809e-02, 5.61053962e-03, 5.60325082e-02,
 0.00000000e+00, 2.47371483e-02, -6.83164595e-01,
 1.38131459e-02, -4.27164544e-02, 0.00000000e+00],
[-2.36828364e-02, -1.73764063e-04, -1.21819241e-01,
-8.05881038e-02, -2.73454927e-02, -1.48091427e-01,
 0.00000000e+00, -1.58172274e-02, -3.38732523e-01,
 3.01768458e-02, 4.02904355e-02, 0.00000000e+00],
[-3.19268804e-02, -2.95998585e-02, -3.71123954e-02,
-6.80604740e-02, -5.50514346e-02, -1.44524613e-01,
 1.39827973e-03, -1.64910190e-02, 0.00000000e+00,
 3.79297249e-02, -1.42344663e-01, 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_disp

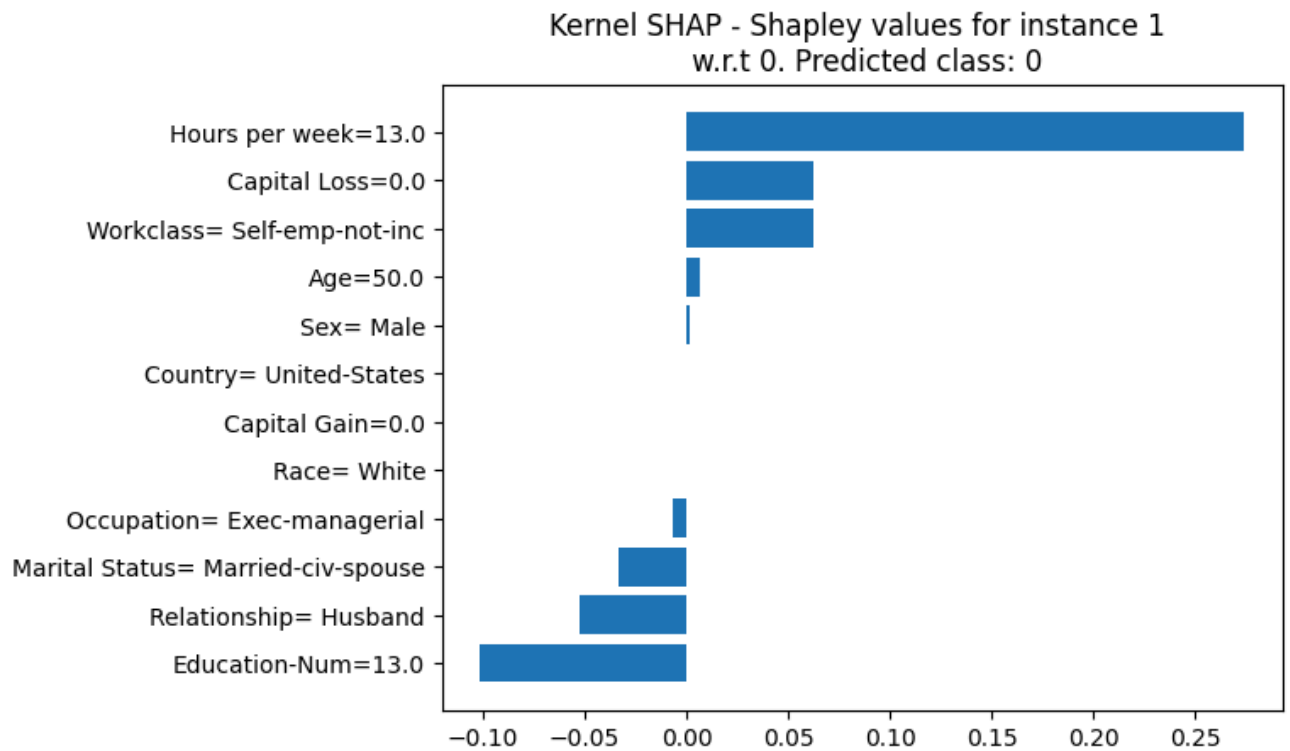
# Plot SHAP values
plt.barh(feature_names_values[sorted_indices], shap_values_ndarray_kernel[id_inst

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

# Plot title
plt.title(f'Kernel SHAP - Shapley values for instance {id_instance} \n w.r.t {cla

```


⇒ 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_disp
```

```
# Plot SHAP values
```

```
plt.barh(feature_names_values[sorted_indices], shap_values_ndarray_kernel[id_inst
```

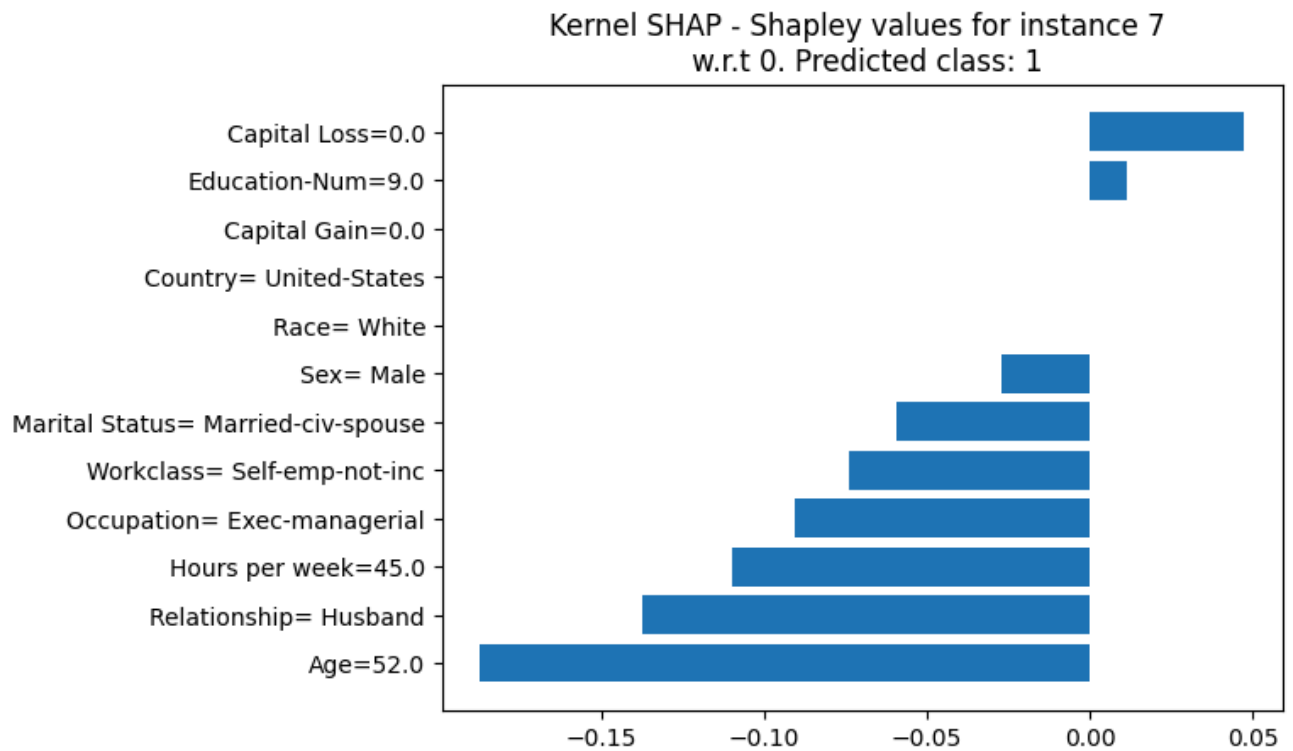
```
# Predict class for the instance
```

```
predicted_class = int(rf_clf.predict(sample_data.iloc[id_instance:id_instance+1])
```

```
# Plot title
```

```
plt.title(f'Kernel SHAP - Shapley values for instance {id_instance} \n w.r.t {cla
```

⇒ Text(0.5, 1.0, 'Kernel SHAP - Shapley values for instance 7 \n w.r.t 0.
Predicted class: 1')



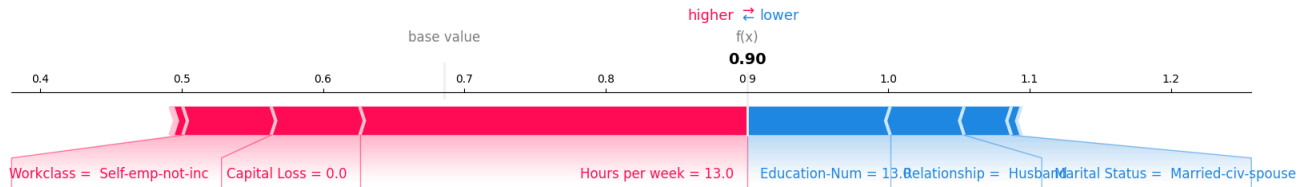
Kernel SHAP - force_plot

```
# 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
                shap_values=shap_values_ndarray_kernel[id_instance, :], # select
                features=X_display.iloc[id_instance, :], # Use the X_display data
                matplotlib=True)
```

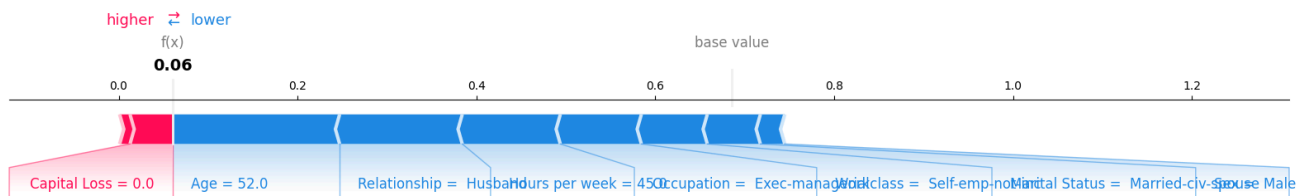
⇒ [0.9 0.1]



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

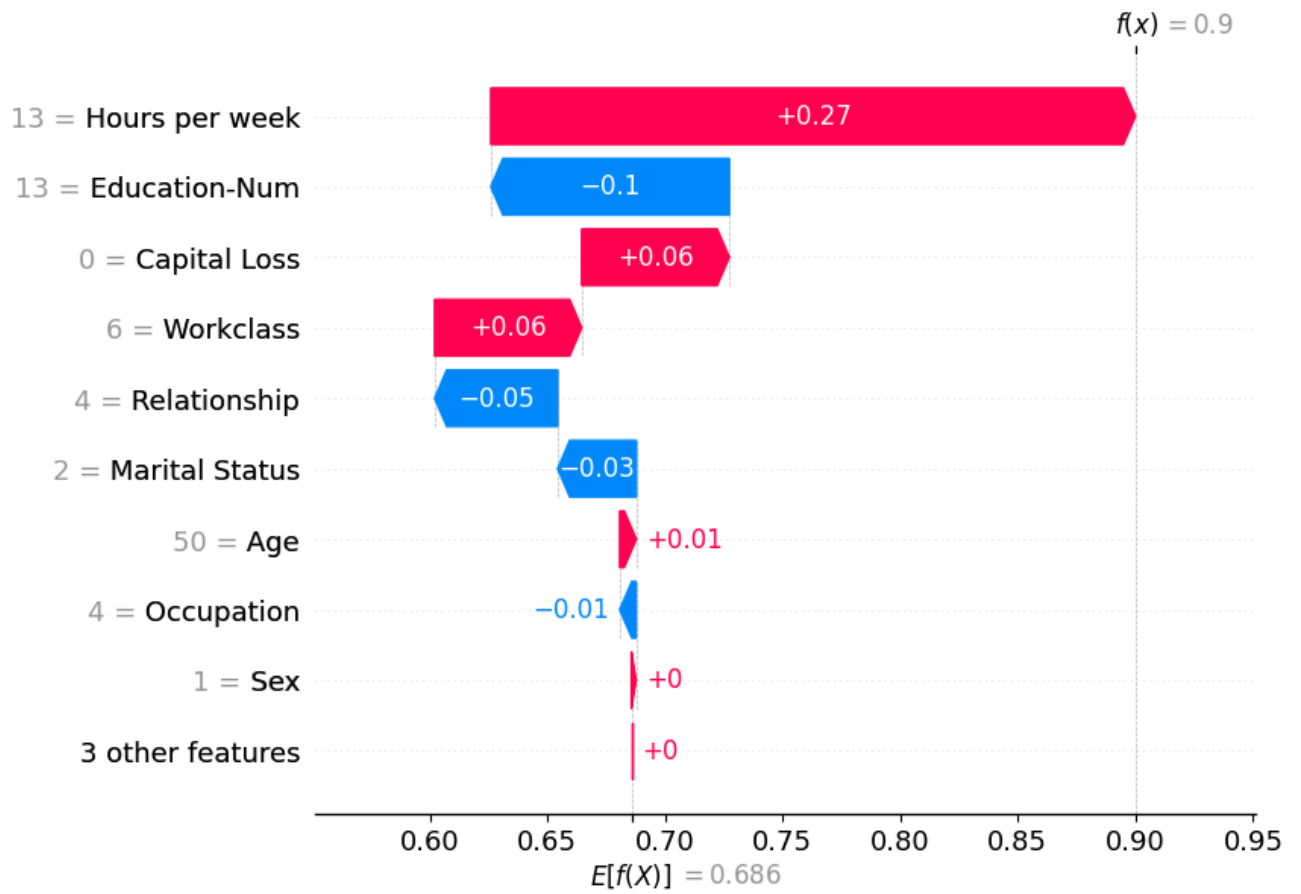
```
shap.force_plot(base_value=expected_value_kernel,
                shap_values=shap_values_ndarray_kernel[id_instance, :],
                features=X_display.iloc[id_instance, :],
                matplotlib=True)
```

⇒ [0.06 0.94]

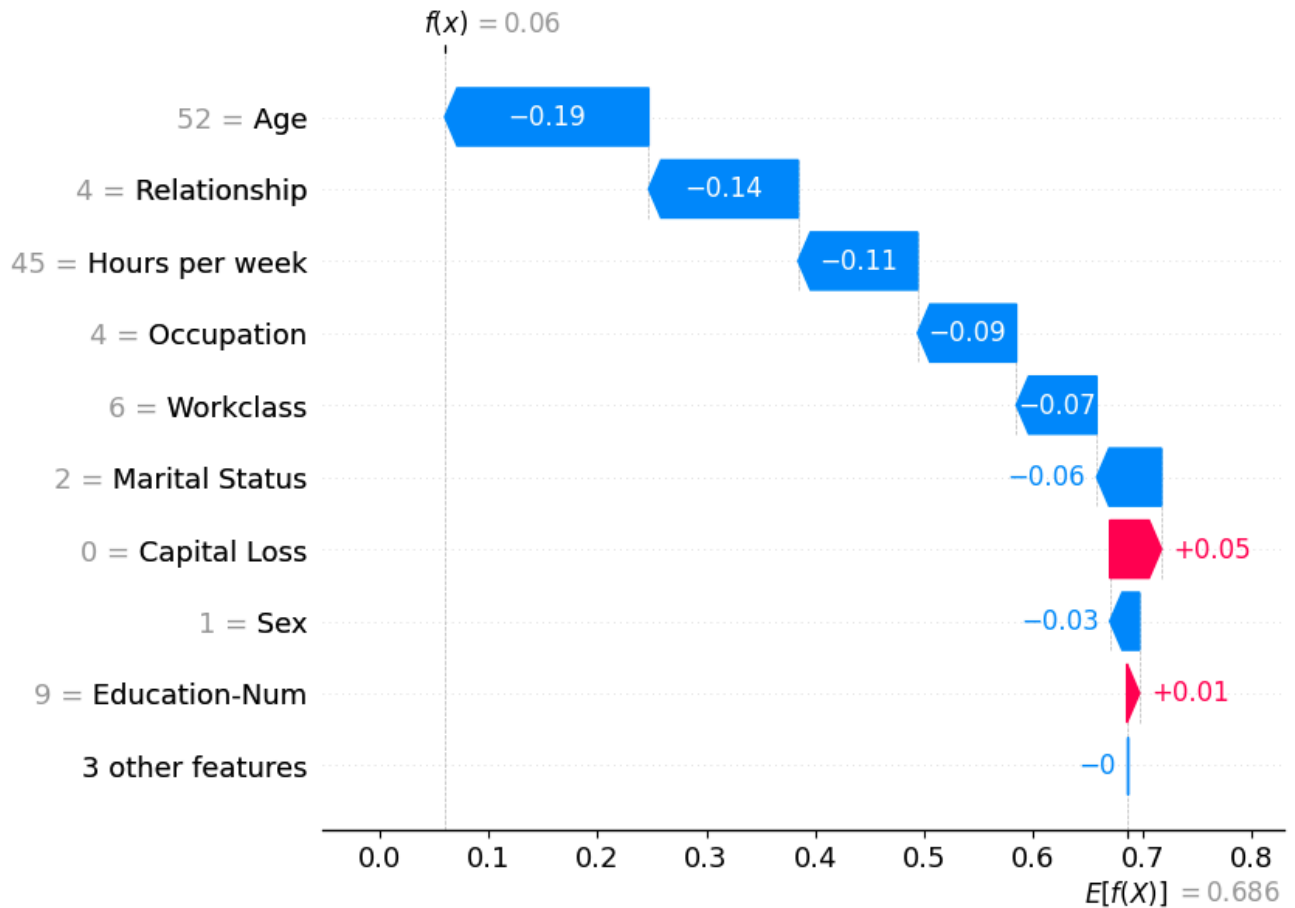


Kernel SHAP - waterfall_plot

```
# Kernel SHAP
id = 1
shap.waterfall_plot(shap_values=shap_values_explanation_kernel[id, :, class_index
                    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, :])
```



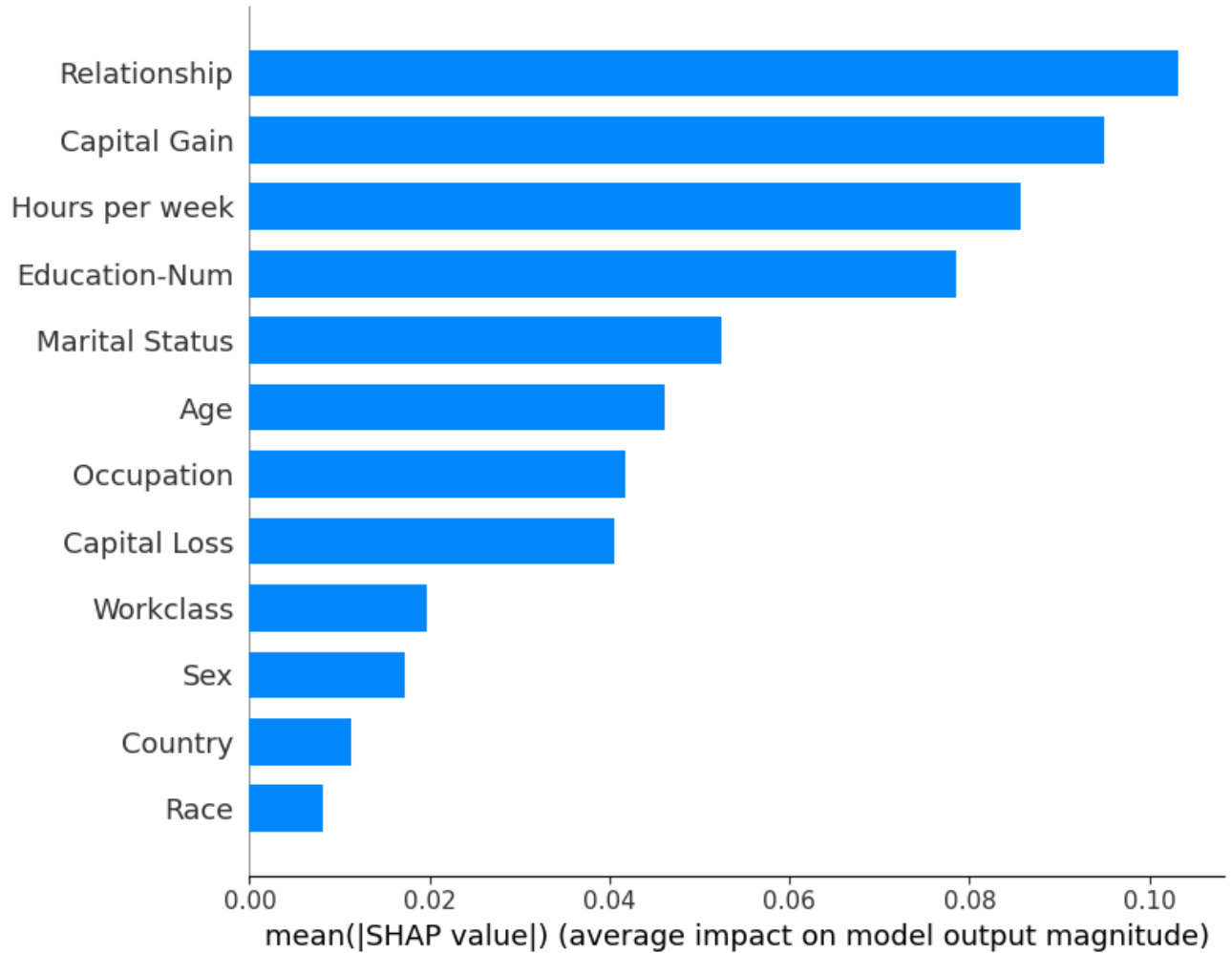
Visualization omitted, Javascript library not loaded!

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

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')
```

```
<ipython-input-32-6ff931ac89cc>:2: FutureWarning: The NumPy global RNG was set
shap.summary_plot(shap_values=shap_values_ndarray_kernel,
```



Kernel SHAP - dependence plot

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

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