# 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.

---

We will 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.

First, we will load the dataset and train a Random Forest classifier. We will perform this steps: a. **Install SHAP** library. b. **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".

c. Split the Adult dataset. 80/20 train-test ratio. d. Train a RandomForestClassifier and fit it over the training dataset. Evaluate the model.

Exercise.

1. Use the `shap.Explainer` to explain the instances `id=1` and `id=7` .
2. Local Explanations:

- 2.1 **Plot** with a bar chart from `matplotlib.pyplot` the **Shapley values** for the instances `id=1` and `id=7` .
- 2.2 Plot the shap explanation for the instances `id=1` and `id=7` with:
  - 2.2a `shap.force_plot`
  - 2.2b `shap.waterfall_plot`

3. Global (for all features) Explanations:

- Plot the shap explanation for the instances `id=1` and `id=7` with:
  - 3.1 `shap.summary_plot`
  - 3.2 `shap.dependence_plot`
  - 3.3 `shap.force_plot`

4. Bonus: Do again point 1.) with `shap.KernelExplainer` and `shap.ExactExplainer`

Before starting, install SHAP

```
# pip install shap
# pip install shap --upgrade
```

## Data loading and preprocessing

```
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()
```

### Load dataset

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

| | Age | Workclass | Education-Num | Marital Status | Occupation | Relationship | Race | Sex | Capital Gain | Capital Loss | Hours per week | Coun |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 39.0 | 7 | 13.0 | 4 | 1 | 0 | 4 | 1 | 2174.0 | 0.0 | 40.0 | |
| **1** | 50.0 | 6 | 13.0 | 2 | 4 | 4 | 4 | 1 | 0.0 | 0.0 | 13.0 | |
| **2** | 38.0 | 4 | 9.0 | 0 | 6 | 0 | 4 | 1 | 0.0 | 0.0 | 40.0 | |
| **3** | 53.0 | 4 | 7.0 | 2 | 6 | 4 | 2 | 1 | 0.0 | 0.0 | 40.0 | |
| **4** | 28.0 | 4 | 13.0 | 2 | 10 | 5 | 2 | 0 | 0.0 | 0.0 | 40.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **32556** | 27.0 | 4 | 12.0 | 2 | 13 | 5 | 4 | 0 | 0.0 | 0.0 | 38.0 | |
| **32557** | 40.0 | 4 | 9.0 | 2 | 7 | 4 | 4 | 1 | 0.0 | 0.0 | 40.0 | |
| **32558** | 58.0 | 4 | 9.0 | 6 | 1 | 1 | 4 | 0 | 0.0 | 0.0 | 40.0 | |
| **32559** | 22.0 | 4 | 9.0 | 4 | 1 | 3 | 4 | 1 | 0.0 | 0.0 | 20.0 | |
| **32560** | 52.0 | 5 | 9.0 | 2 | 4 | 5 | 4 | 0 | 15024.0 | 0.0 | 40.0 | |

Next steps: Generate code with X | New interactive sheet

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

### 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)
```

### 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
```

```
0.8552126516198373
```

## 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).

### Sample data

```
# 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]
```

## 1. Explain an instance via SHAP

✍️ Initialize SHAP

```
# Compute the masker
# Use the Independent masker, which assumes feature independence.
# It masks out tabular features by integrating over the given background dataset.
# The masker will use the training data to compute the expected value of the features.


# CODE HERE
masker = shap.maskers.Independent(data = X_train)

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

```
# Analyze which type of approximation is used by default by SHAP
explainer
```

```
<shap.explainers._permutation.PermutationExplainer at 0x7a1ddef54770>
```

✍️ Explain the sample_data instances using the explainer. User explainer()

```
# CODE HERE
# Calculate the explainer over the sample_data using explainer(<data>)
shap_values_explanation = explainer(sample_data)
```

```
PermutationExplainer explainer: 102it [01:23,  1.14it/s]
```

Print of the Shapley values

```
# 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

# 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 'shap._explanation.Explanation'>
.values =
array([[ 0.00721649, -0.00721649],
       [ 0.01614822, -0.01614822],
       [-0.0396981 ,  0.0396981 ],
       [ 0.08227947, -0.08227947],
       [ 0.02387347, -0.02387347],
       [ 0.10514896, -0.10514896],
       [-0.00011562,  0.00011563],
       [-0.00447901,  0.00447901],
       [ 0.03590603, -0.03590603],
       [ 0.00394103, -0.00394103],
       [ 0.02449049, -0.02449049],
       [ 0.00048841, -0.00048841]])

.base_values =
array([0.72480015, 0.27519985])

.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 [00:20,  2.71it/s]<class 'numpy.ndarray'>
[ 0.00842333  0.01116878 -0.02819331  0.08126745  0.02456629  0.10733449
 -0.00059167 -0.0071303   0.03952756  0.00281496  0.0166686  -0.00065635]
```

```
shap_values_ndarray.shape
```

```
(101, 12)
```

```
shap_values_ndarray[0]
```

```
array([ 0.00842333,  0.01116878, -0.02819331,  0.08126745,  0.02456629,
        0.10733449, -0.00059167, -0.0071303 ,  0.03952756,  0.00281496,
        0.0166686 , -0.00065635])
```

## 2. Indiviual/Local insights - Plots

## 2.1 Bar plots

✍️ Extract the Shapley values for all the sample data instances and for the 'class_index' as target class

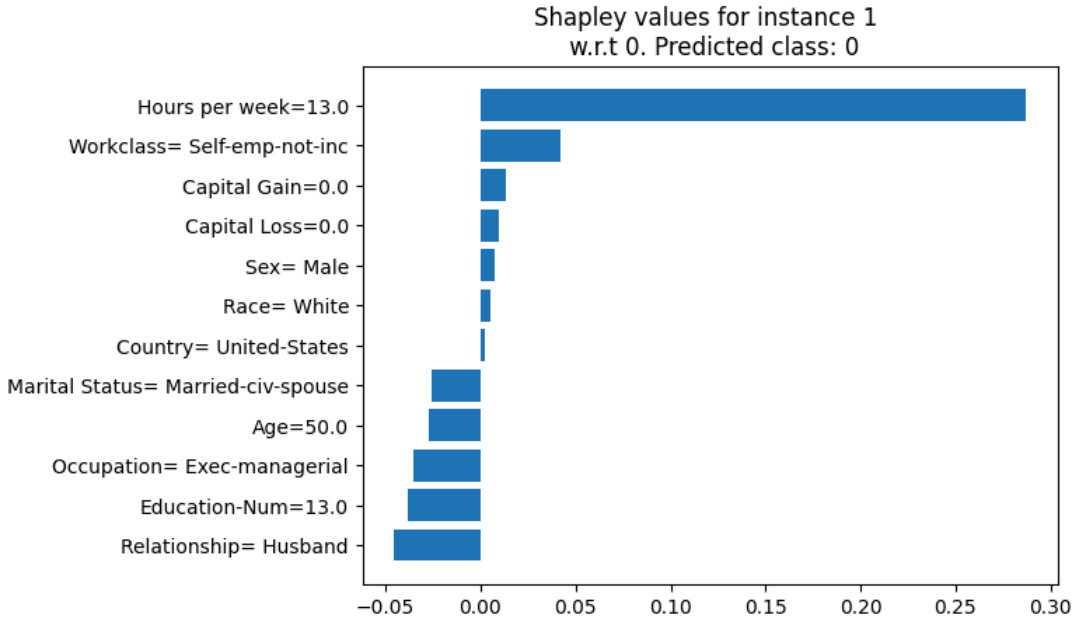
```
# CODE HERE
# Calculate the Shapley values with explainer.shap_values( ... ).
# (alternative way to calculate the shapley values rather than using the explainer(sample_data) method and
# Consider as target the 'class_index' class
shap_values_ndarray = explainer.shap_values(sample_data)[:, :, class_index]
```

```
PermutationExplainer explainer: 102it [00:18,  2.58it/s]
```

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

```
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_dis

# 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_
```

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

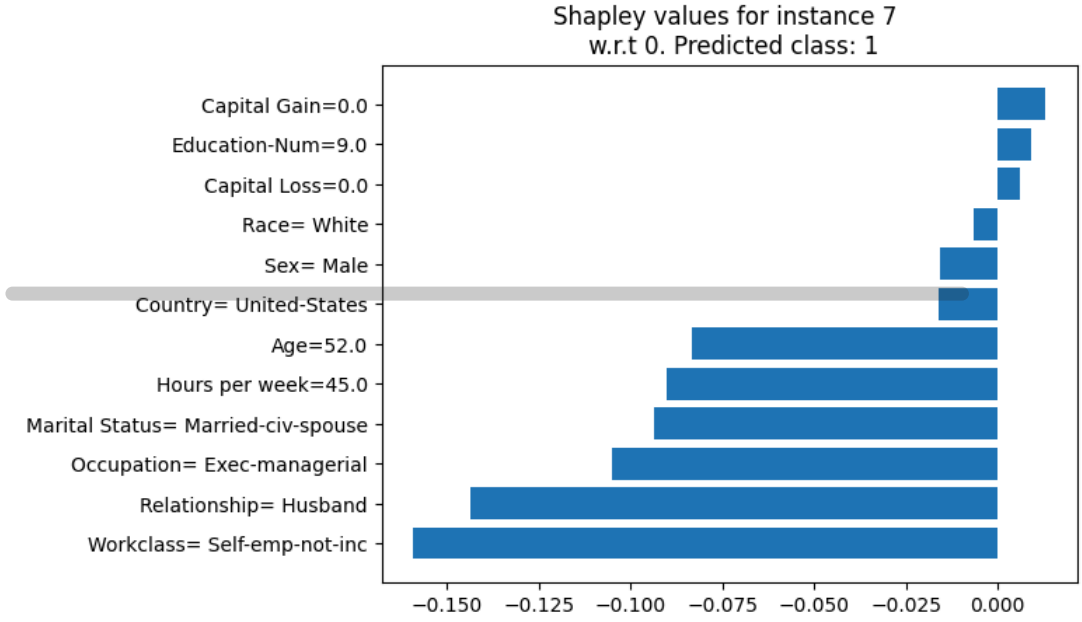



Bar chart of Shapley values for the `id_instance=7`

```
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_display.columns, sample_X_dis

# 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_
```

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



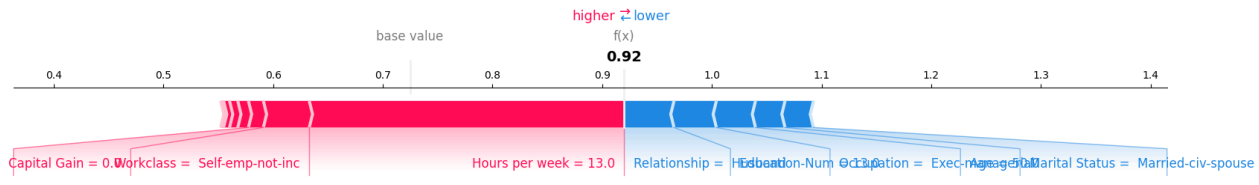

## 2.2a SHAP Force plot - single instance

`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.

✍ Plot the SHAP Force plot

```
id_instance = 1
print(rf_clf.predict_proba(sample_data)[id_instance])


# CODE HERE
# Use the shap.force_plot function to visualize the shapley values for the 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
                features=X_display.iloc[id_instance, :], # Use the X_display dataset to have meaningful re
                matplotlib=True)
```

```
[0.92 0.08]
```

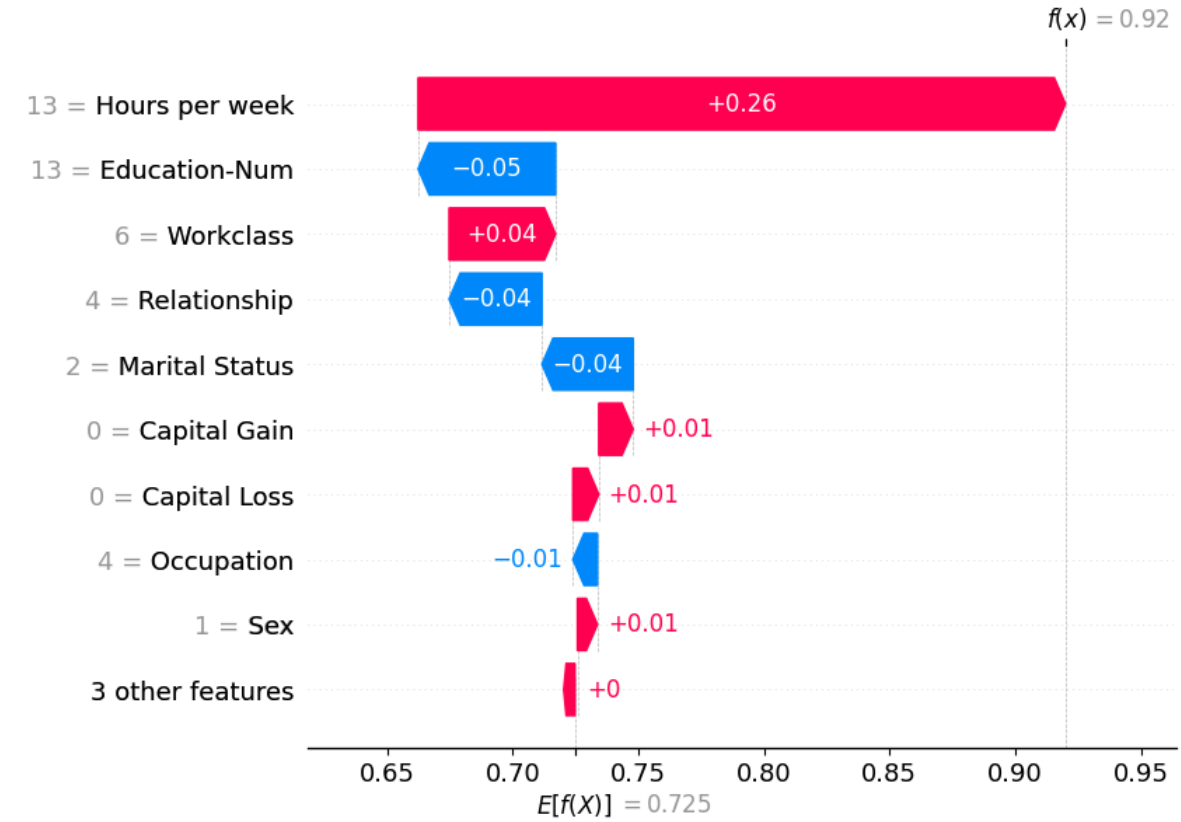



## 2.2b SHAP - waterfall plot

`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.

✍ Plot the waterfall plot

```
id = 1

#CODE HERE
shap.waterfall_plot(shap_values=shap_values_explanation[id, :, class_index], # select the shap_value_expla
                    max_display=10)
```




## 3. Global insights

SHAP Summary plot

---

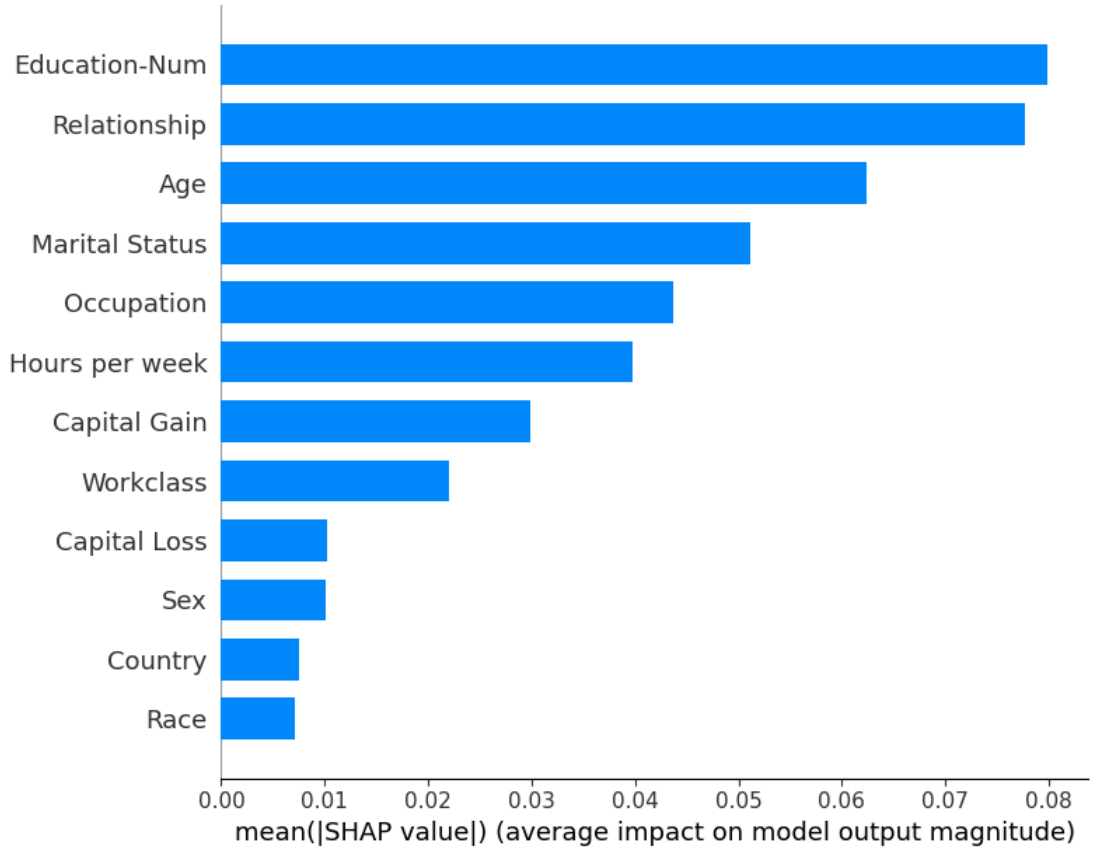
- It is used to visualize the **summary** of **SHAP values** for all features in a dataset. This plot is particularly useful for understanding the importance and directionality of different features in influencing model predictions.
- It provides the **average impact** of each feature on the model output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

## 3.1 Summary plot

✍ Plot the summary plot

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

```
/tmp/ipykernel_12742/2538854307.py:3: FutureWarning: The NumPy global RNG was seeded by calling `np.random.
  shap.summary_plot(shap_values=shap_values_ndarray,
```

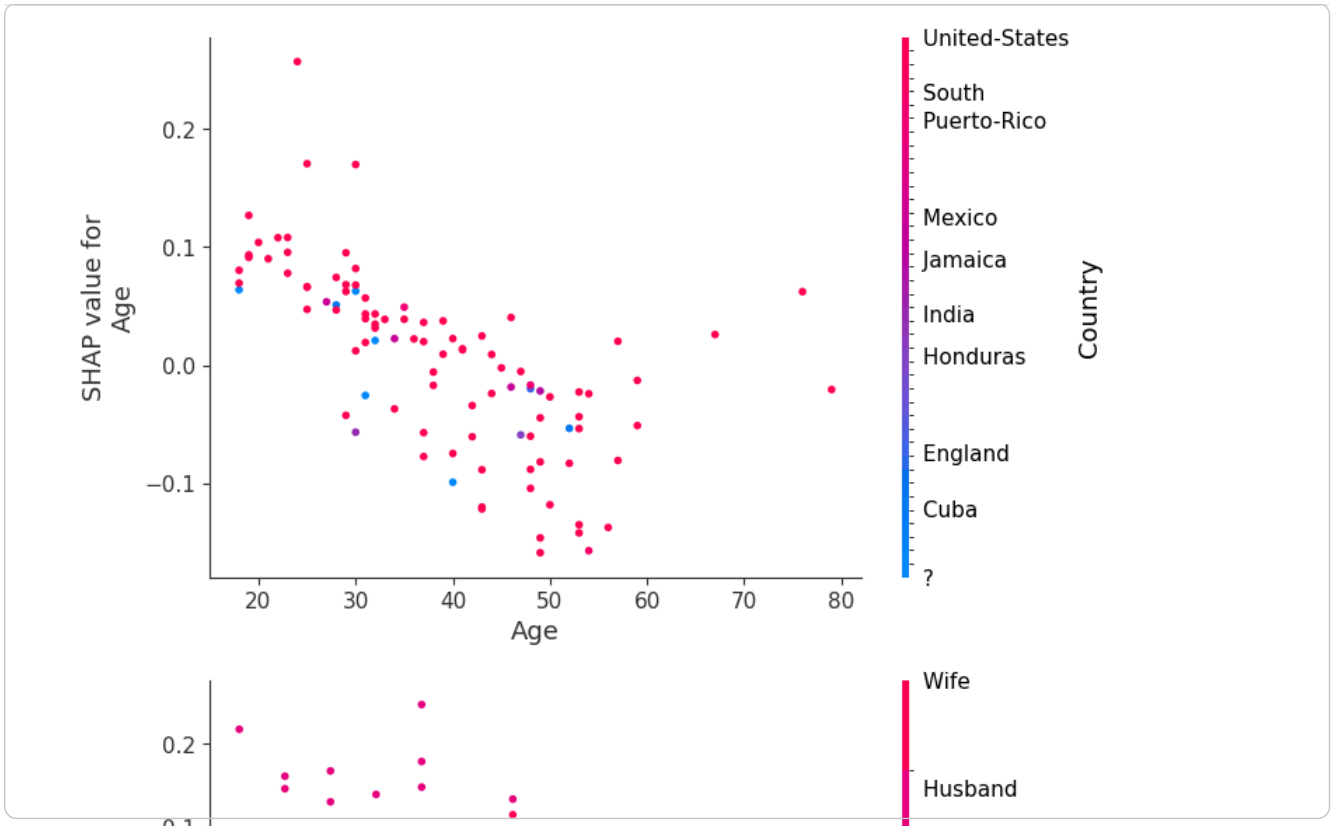



## 3.2 SHAP dependence plot

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.

✍ Plot the SHAP dependence plot

```
# Plot the dependence plot for the some features of interest (or all!).

# CODE HERE
selected_features = ['Age', 'Education-Num']
for name in selected_features:
    # CODE HERE
    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
```




## 3.3 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 been written.

## 4. Bonus

Redo your analysis with KernelExplainer or another Explainer