In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import shap
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import zero_one_loss, log_loss
import lime.lime_tabular
```

In [2]:
```
X, y = shap.datasets.adult()
X_display, y_display = shap.datasets.adult(display=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rai
```

In [3]:
```
clf = GradientBoostingClassifier(n_estimators=100 , random_state=10)
clf.fit(X_train.values, y_train)
```

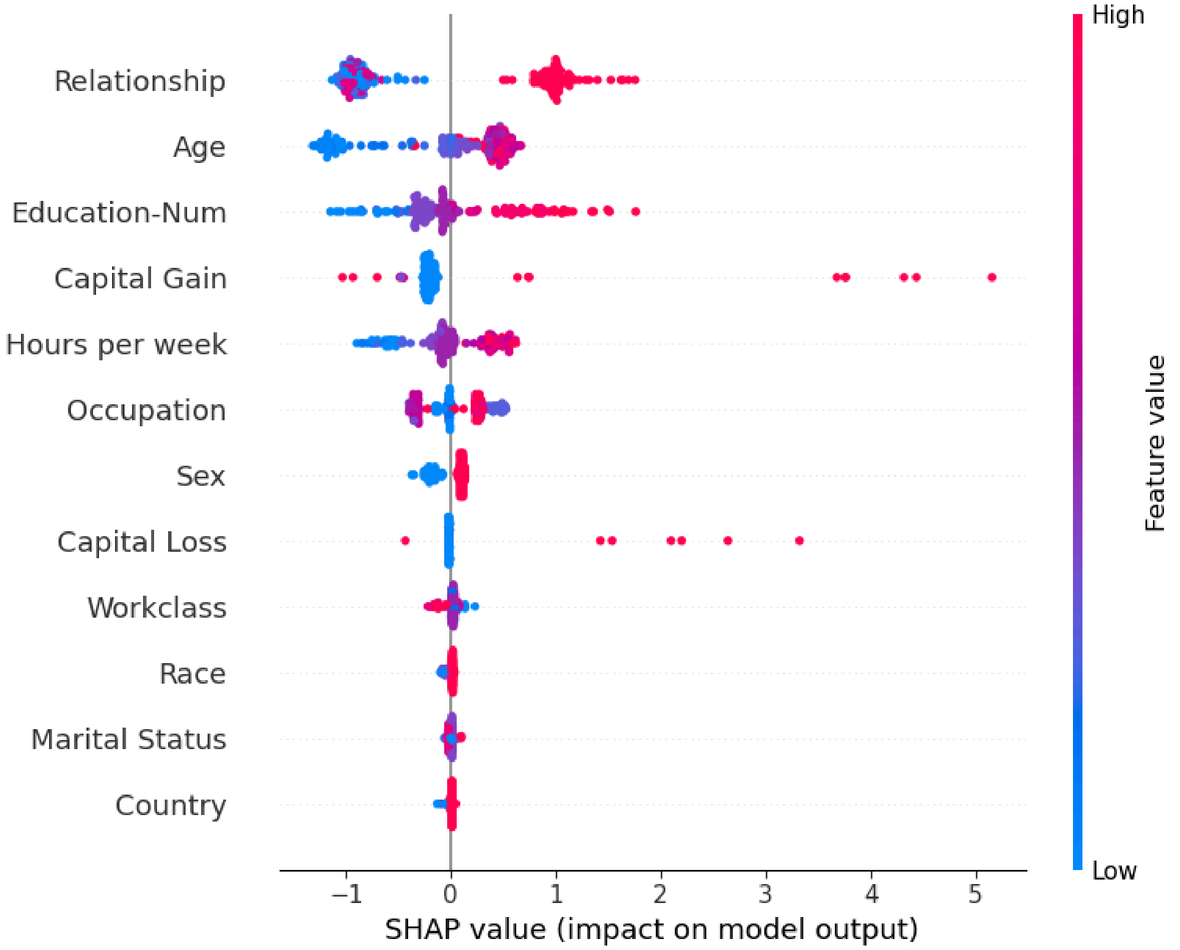
Out[3]:
```
▾ GradientBoostingClassifier
GradientBoostingClassifier(random_state=10)
```

5 a)

In [4]:
```
explicands = X_test[:200]
baselines = X_test[-100:]
```

In [5]:
```
treeshap_explainer = shap.TreeExplainer(clf, baselines)
attributions1 = treeshap_explainer.shap_values(explicands)
```

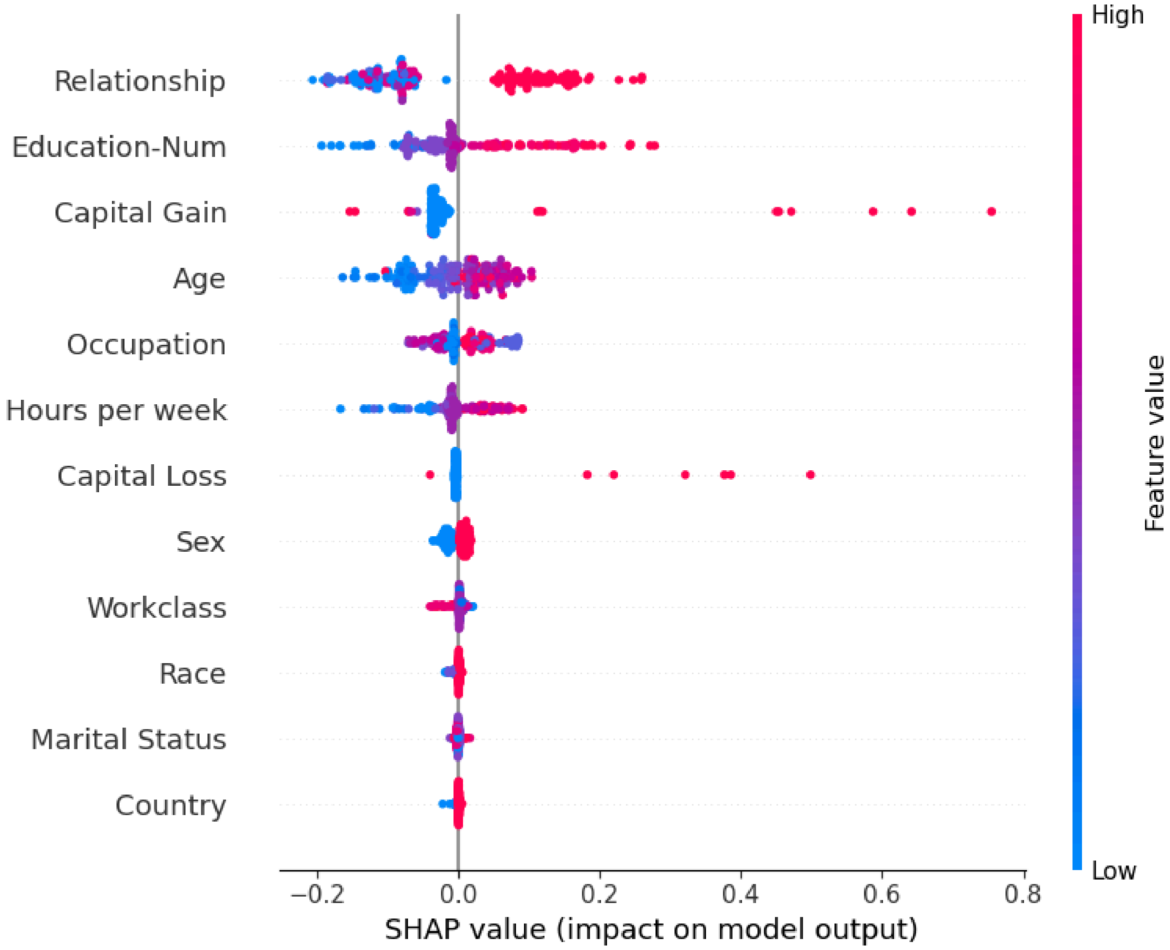
In [6]:
```
shap.summary_plot(attributions1, explicands)
```



In [14]:
```
kernelshap_explainer = shap.KernelExplainer(clf.predict_proba, baselines)
attributions3 = kernelshap_explainer.shap_values(explicands)
```

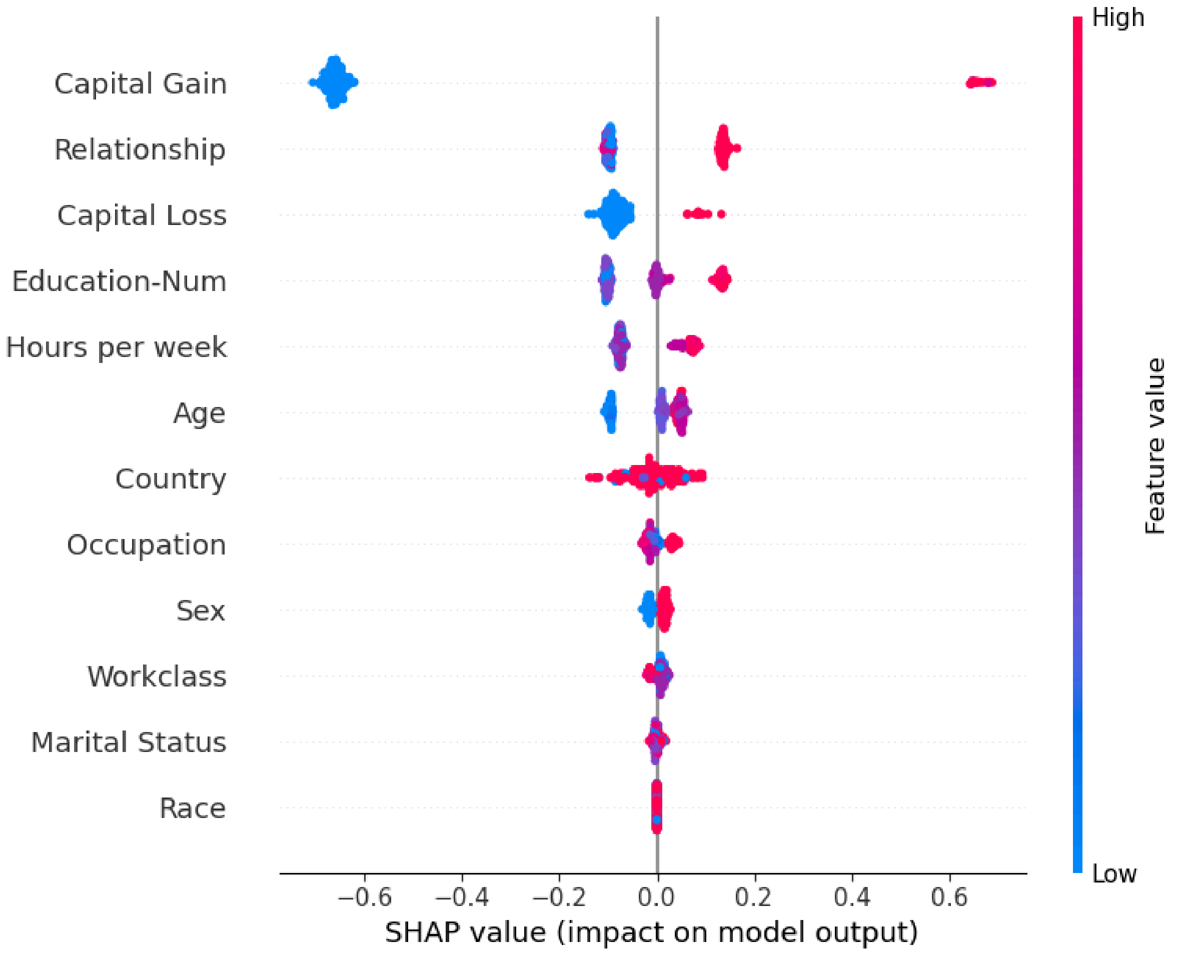
```
  0%|          | 0/200 [00:00<?, ?it/s]
```

In [19]:
```
shap.summary_plot(attributions3[:,:,1], explicands)
```



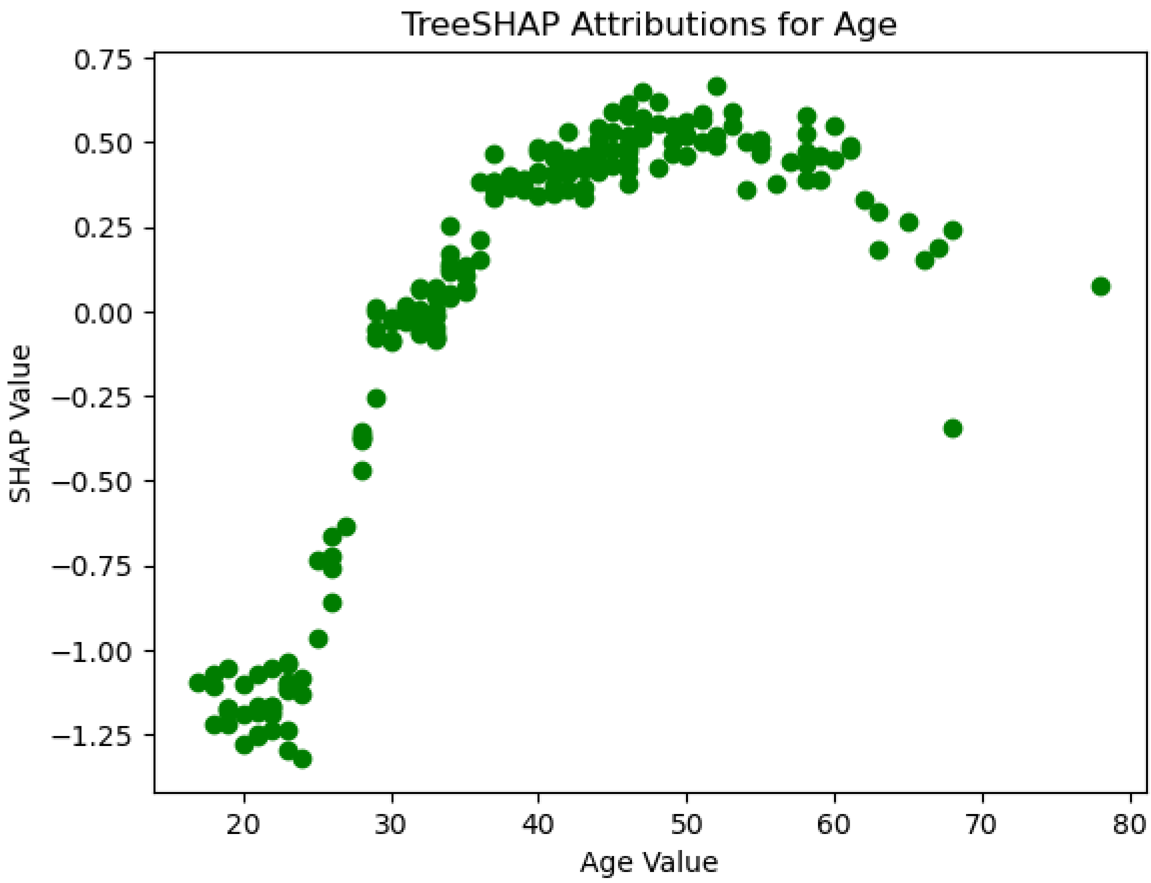
In [21]:
```
lime_explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values)
attributions2 = []

for i, idx in enumerate(explicands.index.tolist()):
    exp = lime_explainer.explain_instance(explicands.loc[idx].values, clf.pı
    attribution_values1 = sorted(exp.local_exp[1], key=lambda x: x[0])
    attribution_values = [x[1] for x in attribution_values1]
    attributions2.append(np.array(attribution_values))
attributions2 = np.array([attributions2])[0]
```

In [23]:
```
shap.summary_plot(attributions2, explicands)
```

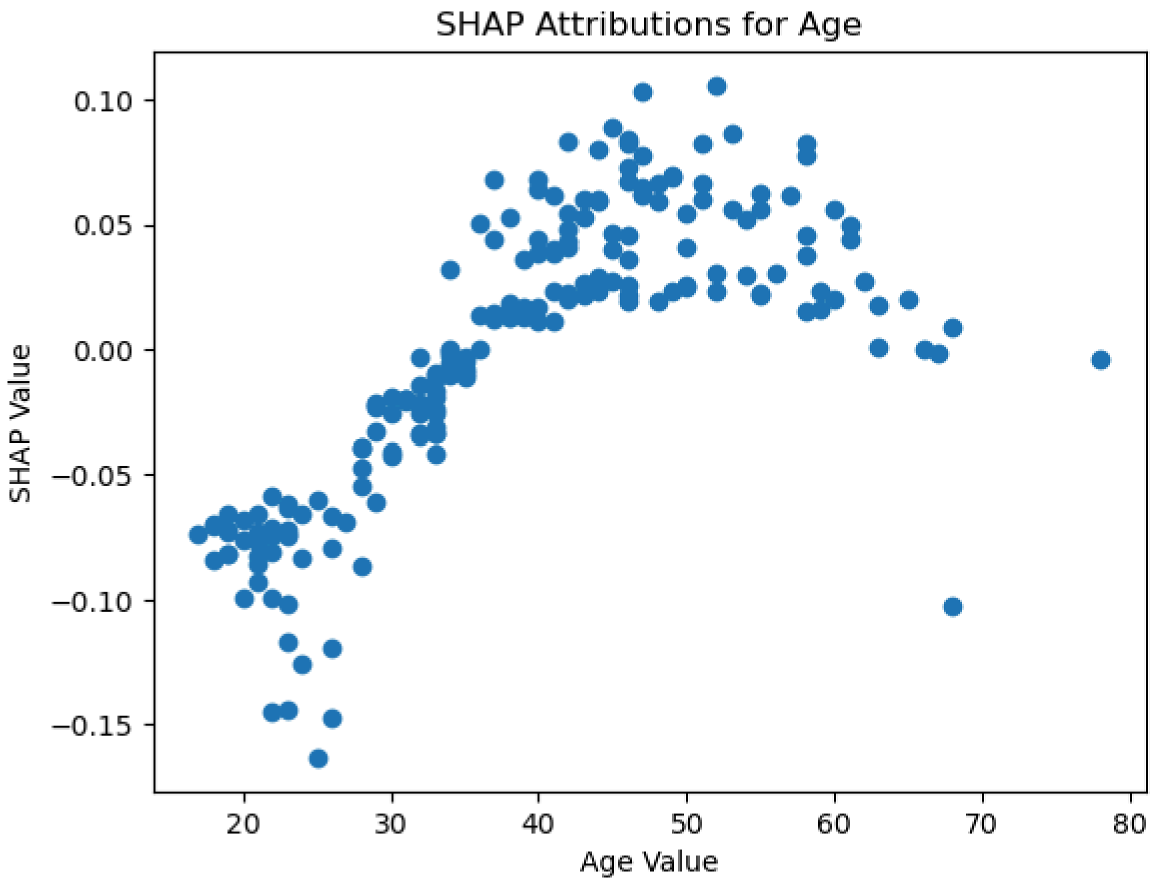


5 b

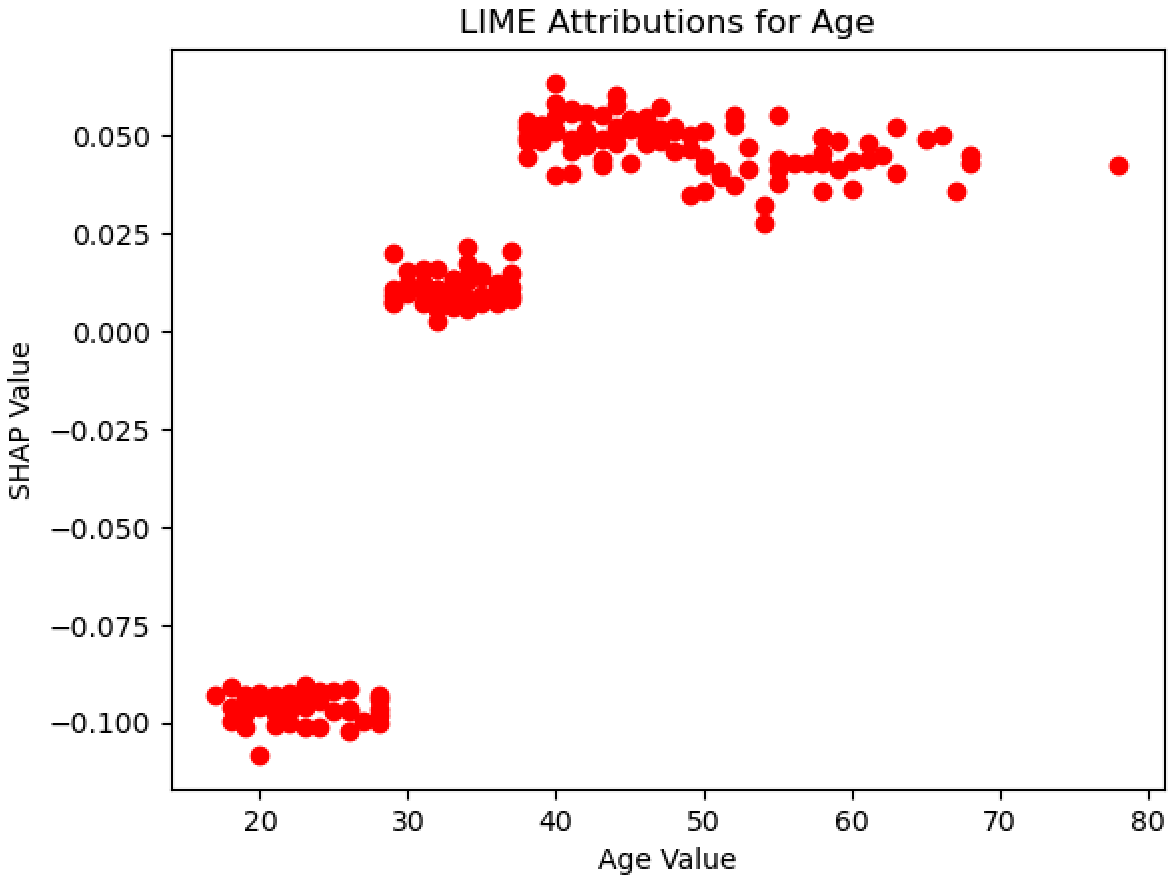
In [37]:
```
feature_index = explicands.columns.get_loc("Age")
feature_values = explicands.iloc[:, feature_index]
feature_attributions = attributions1[:, feature_index]

plt.scatter(feature_values, feature_attributions,color='g')
plt.xlabel("Age" + " Value")
plt.ylabel("SHAP Value")
plt.title("TreeSHAP Attributions for " + "Age")
plt.show()
```



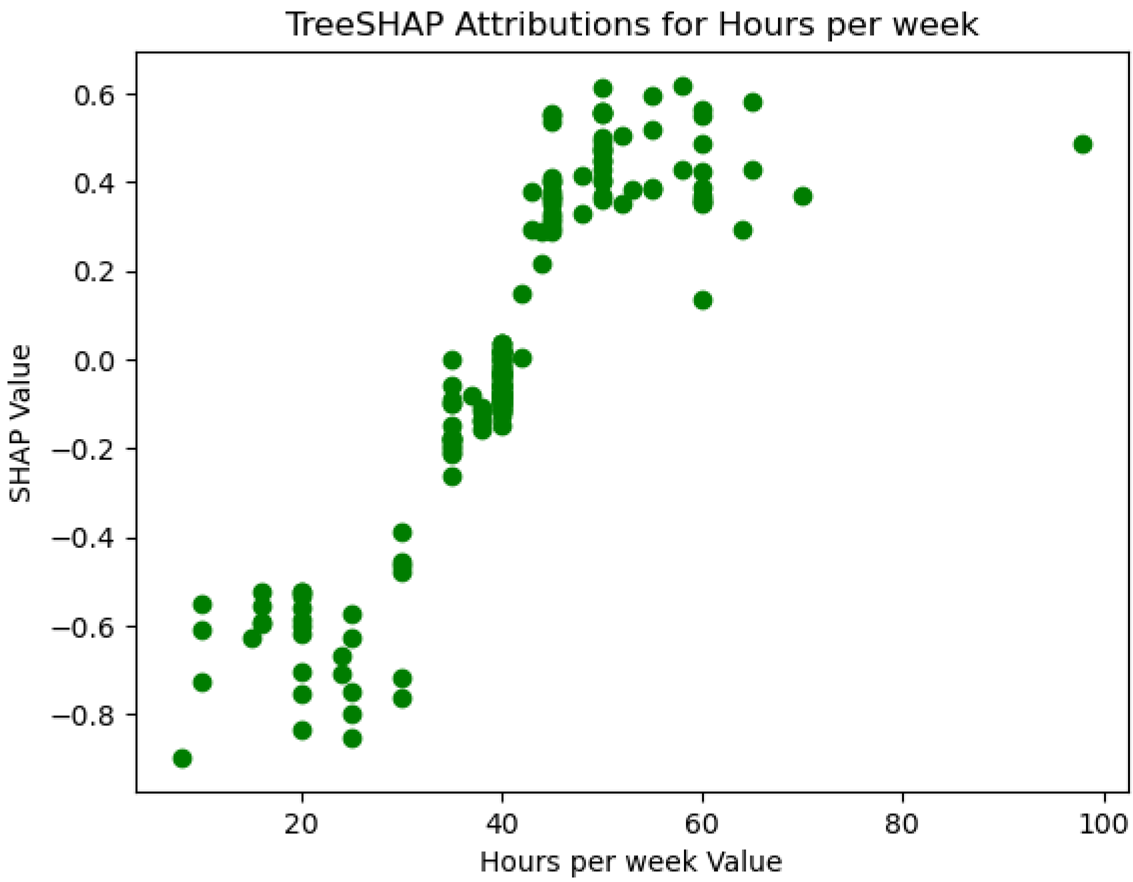
In [29]:
```
feature_index = explicands.columns.get_loc("Age")
feature_values = explicands.iloc[:, feature_index]
feature_attributions = attributions[:,:,1][:, feature_index]

plt.scatter(feature_values, feature_attributions)
plt.xlabel("Age" + " Value")
plt.ylabel("SHAP Value")
plt.title("KernalSHAP Attributions for " + "Age")
plt.show()
```



In [36]:
```
feature_index = explicands.columns.get_loc("Age")
feature_values = explicands.iloc[:, feature_index]
feature_attributions = attributions2[:, feature_index]

plt.scatter(feature_values, feature_attributions, color='r')
plt.xlabel("Age" + " Value")
plt.ylabel("SHAP Value")
plt.title("LIME Attributions for " + "Age")
plt.show()
```

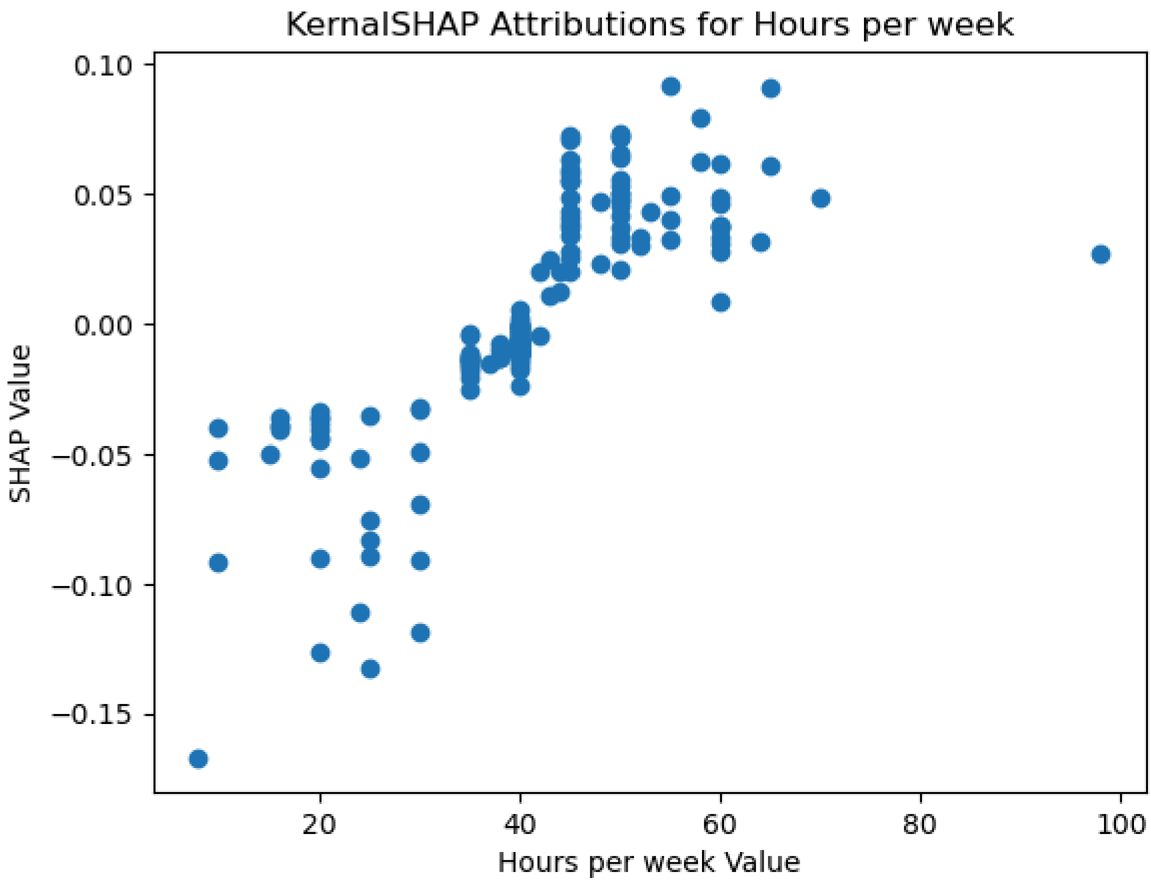


In [38]:
```
feature_index = explicands.columns.get_loc("Hours per week")
feature_values = explicands.iloc[:, feature_index]
feature_attributions = attributions1[:, feature_index]

plt.scatter(feature_values, feature_attributions,color='g')
plt.xlabel("Hours per week" + " Value")
plt.ylabel("SHAP Value")
plt.title("TreeSHAP Attributions for " + "Hours per week")
plt.show()
```

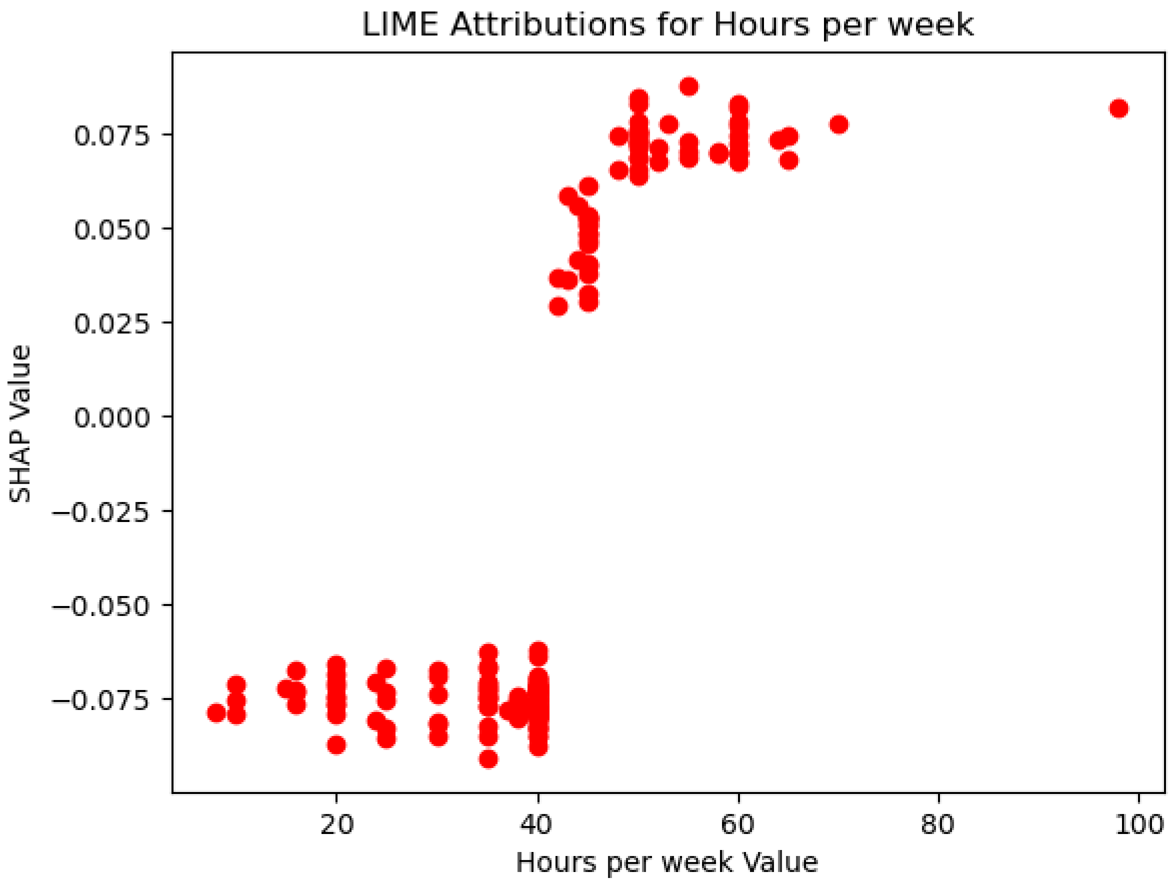


In [32]:

```
feature_index = explicands.columns.get_loc("Hours per week")
feature_values = explicands.iloc[:, feature_index]
feature_attributions = attributions[:,:,1][:, feature_index]

plt.scatter(feature_values, feature_attributions)
plt.xlabel("Hours per week" + " Value")
plt.ylabel("SHAP Value")
plt.title("KernalSHAP Attributions for " + "Hours per week")
plt.show()
```

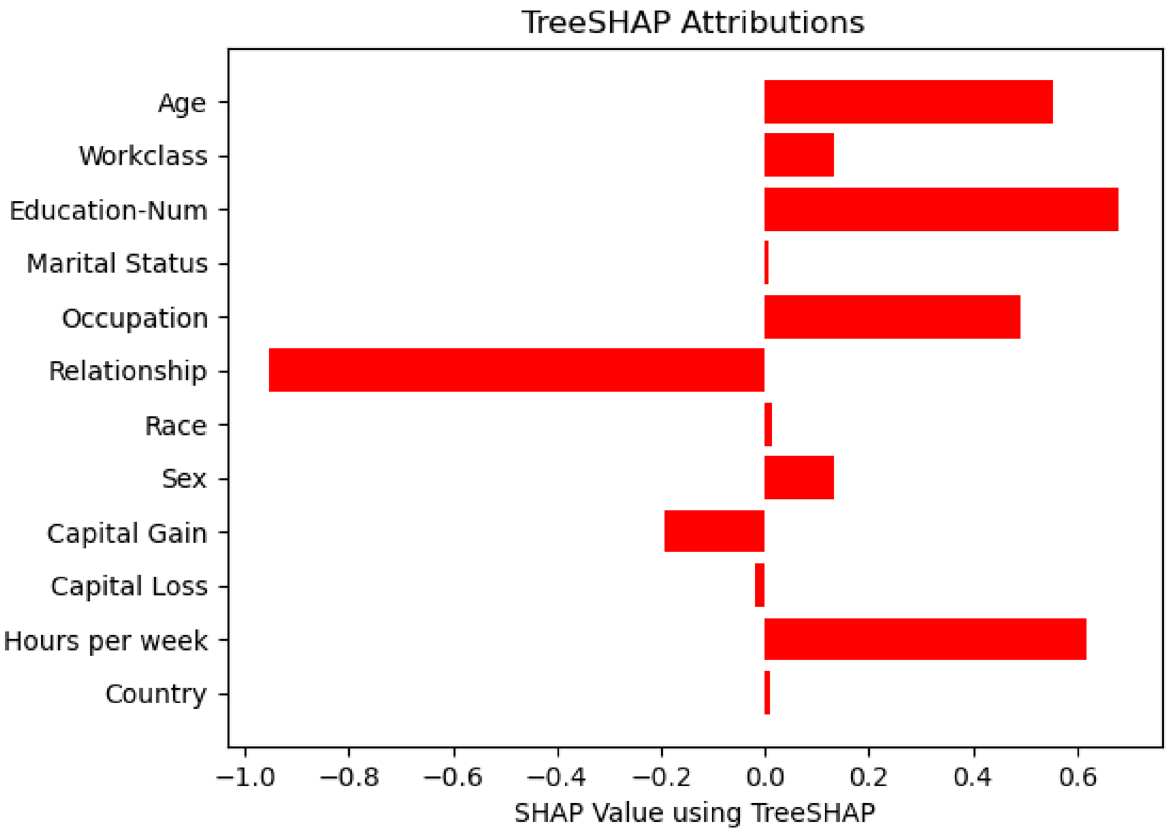


In [34]:
```
feature_index = explicands.columns.get_loc("Hours per week")
feature_values = explicands.iloc[:, feature_index]
feature_attributions = attributions2[:, feature_index]

plt.scatter(feature_values, feature_attributions, color='r')
plt.xlabel("Hours per week" + " Value")
plt.ylabel("SHAP Value")
plt.title("LIME Attributions for " + "Hours per week")
plt.show()
```

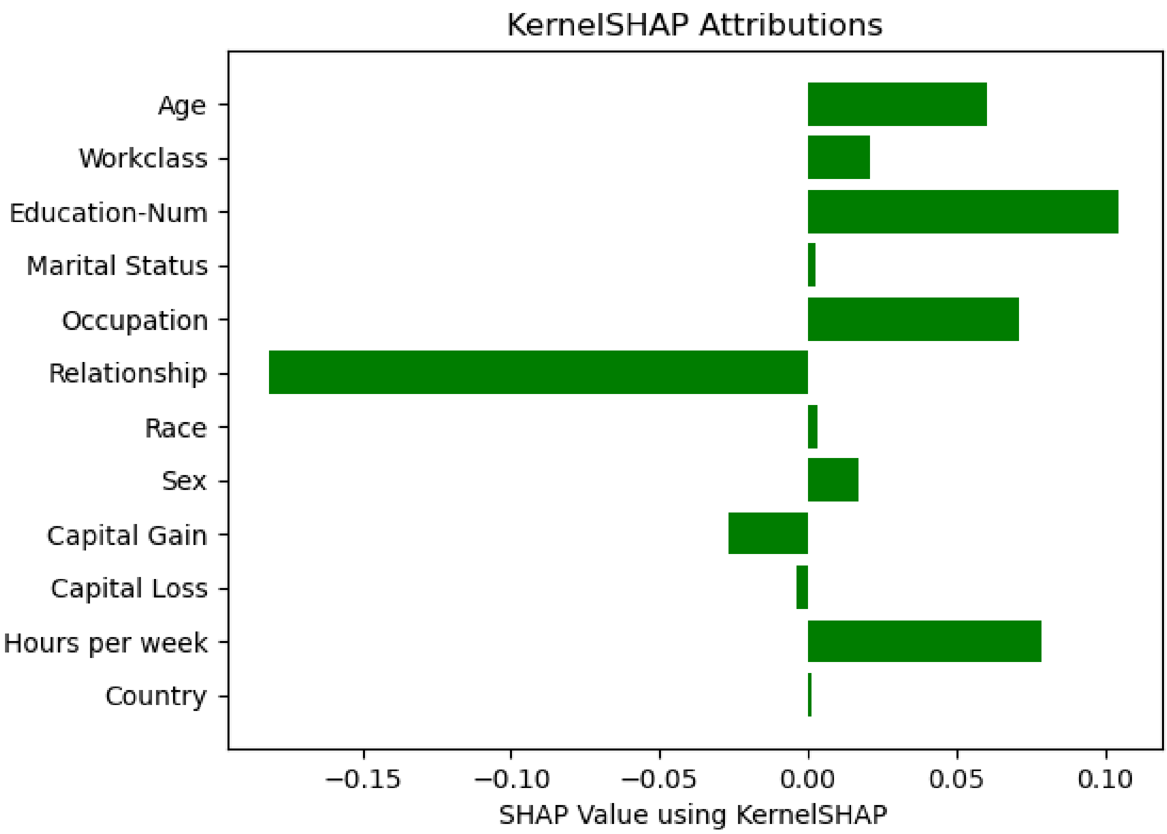


5 c)

In [88]:
```
plt.barh(explicands.columns.tolist(), attributions1[1],color='r')
plt.xlabel("SHAP Value using TreeSHAP")
plt.title("TreeSHAP Attributions")
plt.gca().invert_yaxis()

plt.show()
```

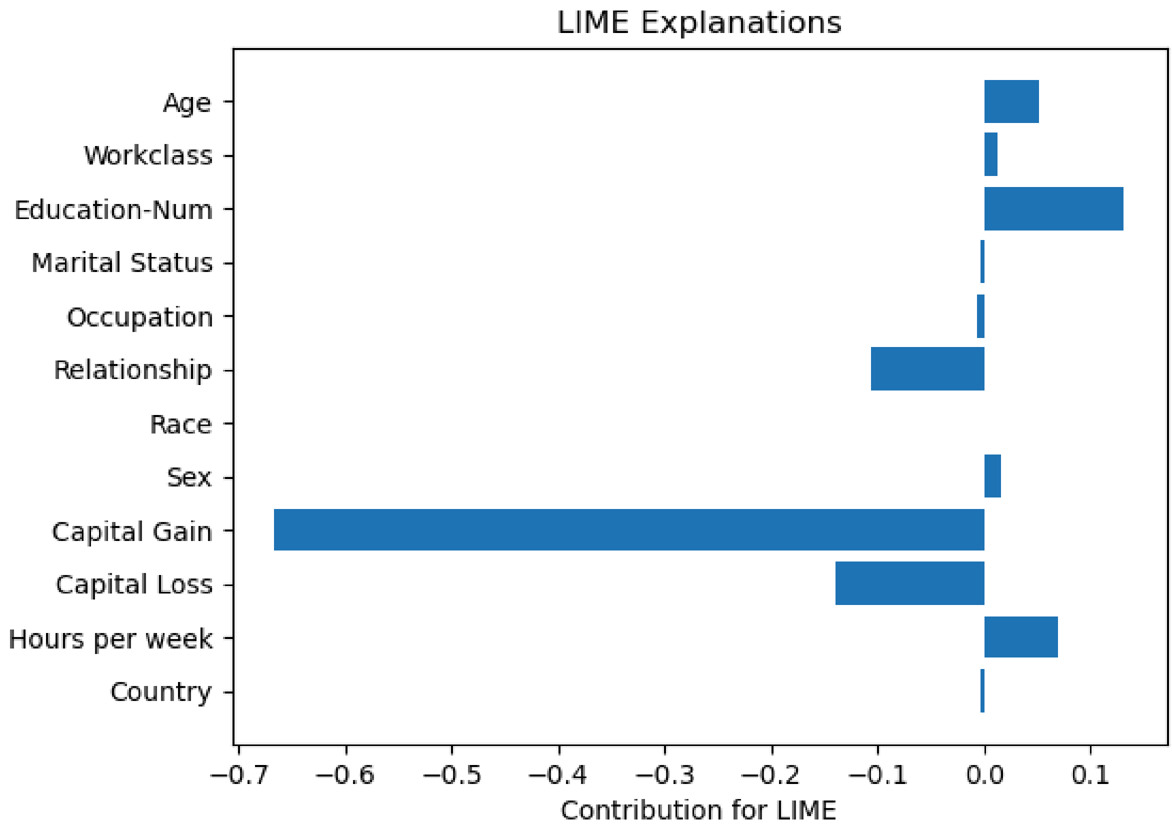


In [87]:
```
plt.barh(explicands.columns.tolist(), attributions3[:,:,1][1],color='g')
plt.xlabel("SHAP Value using KernelSHAP")
plt.title("KernelSHAP Attributions")
plt.gca().invert_yaxis()
```



In [77]:
```
# Subplot for LIME
plt.barh(explicands.columns.tolist(), attributions2[1])
plt.xlabel("Contribution for LIME")
plt.title("LIME Explanations")
```

```
plt.gca().invert_yaxis()
plt.show()
```



5 d)

In [55]:
```
sample_to_explain = X_test.iloc[1:2]  # Select only the sample at index 1

# Run KernelSHAP with different nsamples and calculate statistics
nsamples_list = [10, 100, 1000]
n_runs = 10  # Number of runs for each nsamples value

feature_names = X_test.columns.tolist()
mean_attributions_dict = {}
std_dev_dict = {}

for nsamples in nsamples_list:
    attributions4 = []
    for _ in range(n_runs):
        kernelshap_explainer = shap.KernelExplainer(clf.predict_proba, base
        attributions4.append(kernelshap_explainer.shap_values(sample_to_exp
    mean_attributions = np.mean(attributions4, axis=0)
    std_dev = np.std(attributions4, axis=0)
    mean_attributions_dict[nsamples] = mean_attributions
    std_dev_dict[nsamples] = std_dev
```

```
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
```

```
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
```

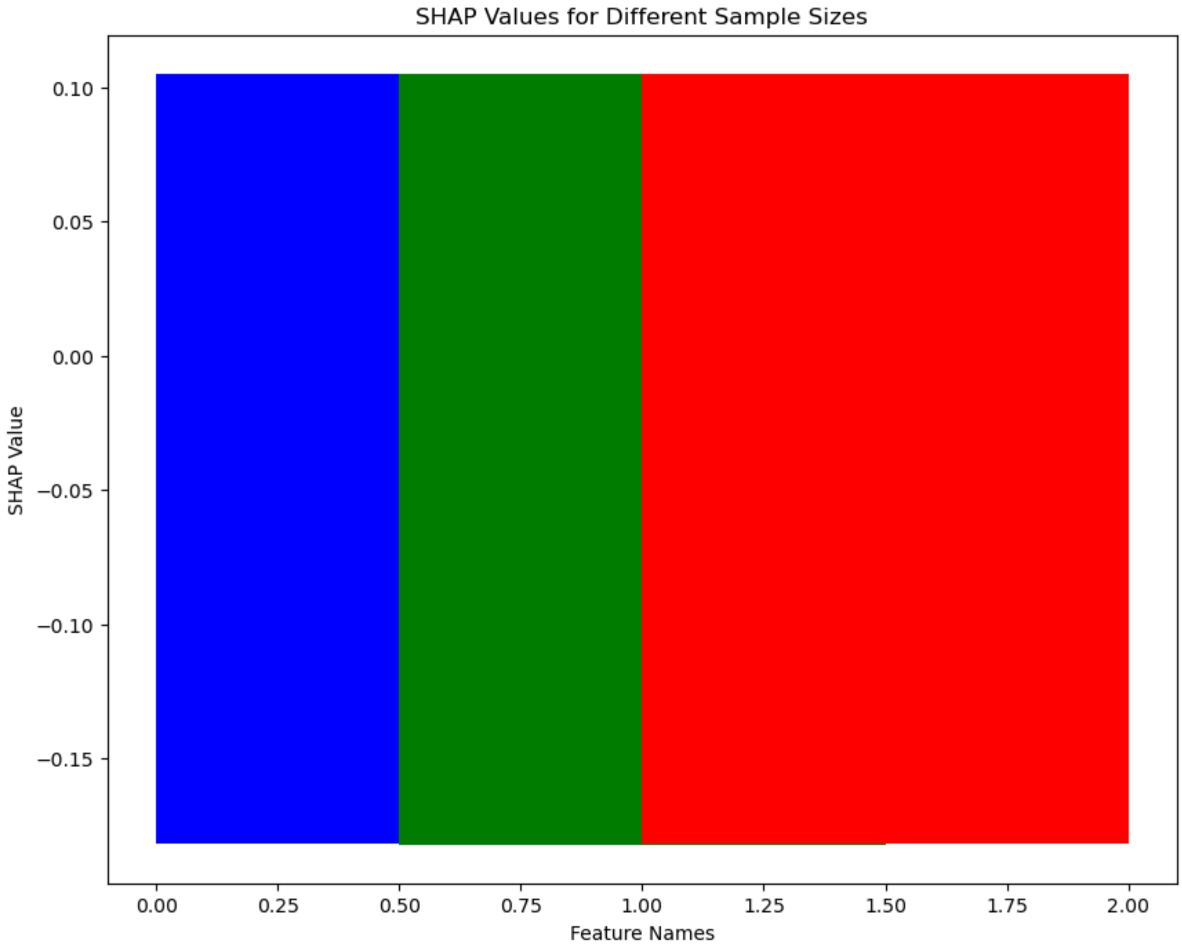
In [67]:

```
nsamples_list
X_axis1 = np.arange(len(X))
fig, ax = plt.subplots(figsize=(10, 8))
bar_width = 2.5
offsets = [0, bar_width, 2 * bar_width]
c = ['b','g','r']
for i, nsample in enumerate(nsamples_list):
    std_dev_values = std_dev_dict[nsample]
    mean_values = mean_attributions_dict[nsample]
    print(mean_values)
    ax.bar(bar_width, mean_values[0], bar_width, color = c[i])
    bar_width=0.5
#     bars2 = ax.bar(feature_names, mean_values, bar_width, bottom=offsets[i

# Set axis labels and title
ax.set_xlabel("Feature Names")
ax.set_ylabel("SHAP Value")
ax.set_title("SHAP Values for Different Sample Sizes")

# Set x-axis ticks and rotation for readability
ax.set_xticks(feature_names)
plt.xticks(rotation=45, ha='right')

# Add legend
ax.legend()

# Tight layout and show the plot
plt.tight_layout()
plt.show()
```

```
[[ 0.0592511   0.02116425  0.10505206  0.00181589  0.07067756 -0.18190132
   0.00215752  0.01721357 -0.0260825  -0.00319627  0.07904022  0.00163439]]
[[ 0.05947951  0.02129057  0.10483374  0.00160922  0.0706603  -0.18203838
   0.00215791  0.0171354  -0.02592686 -0.00296808  0.07913673  0.00145641]]
[[ 0.05959234  0.02134386  0.1050233   0.00150933  0.07050292 -0.18183969
   0.00214455  0.01714135 -0.02610928 -0.0030503   0.07916194  0.00140613]]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/axis.py in convert_u
nits(self, x)
   1505         try:
-> 1506             ret = self.converter.convert(x, self.units, self)
   1507         except Exception as e:

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/category.py in conve
rt(value, unit, axis)
     48         if unit is None:
---> 49             raise ValueError(
     50                 'Missing category information for StrCategoryConver
ter; '

ValueError: Missing category information for StrCategoryConverter; this mig
ht be caused by unintendedly mixing categorical and numeric data

The above exception was the direct cause of the following exception:

ConversionError                           Traceback (most recent call last)
/var/folders/9k/b_vy9rw147729qrz8n_rjjl00000gn/T/ipykernel_17260/253267689
6.py in <module>
     19 
     20 # Set x-axis ticks and rotation for readability
---> 21 ax.set_xticks(feature_names)
     22 plt.xticks(rotation=45, ha='right')
     23 

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/axes/_base.py in wra
pper(self, *args, **kwargs)
     73 
     74         def wrapper(self, *args, **kwargs):
---> 75             return get_method(self)(*args, **kwargs)
     76 
     77         wrapper.__module__ = owner.__module__

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/axis.py in set_ticks
(self, ticks, labels, minor, **kwargs)
   1853         ticks.
   1854         """
-> 1855         result = self._set_tick_locations(ticks, minor=minor)
   1856         if labels is not None:
   1857             self.set_ticklabels(labels, minor=minor, **kwargs)

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/axis.py in _set_tick
_locations(self, ticks, minor)
   1802 
   1803         # XXX if the user changes units, the information will be lo
st here
-> 1804         ticks = self.convert_units(ticks)
   1805         for name, axis in self.axes._get_axis_map().items():
   1806             if self is axis:

~/opt/anaconda3/lib/python3.9/site-packages/matplotlib/axis.py in convert_u
nits(self, x)
   1506             ret = self.converter.convert(x, self.units, self)
   1507         except Exception as e:
-> 1508             raise munits.ConversionError('Failed to convert value
(s) to axis '
   1509                                          f'units: {x!r}') from e
   1510         return ret

ConversionError: Failed to convert value(s) to axis units: ['Age', 'Workcla
```

```
ss', 'Education-Num', 'Marital Status', 'Occupation', 'Relationship', 'Rac
e', 'Sex', 'Capital Gain', 'Capital Loss', 'Hours per week', 'Country']
```



In [79]:
```
attributions
```

Out[79]:
```
[]
```

In [82]:
```
attributions3
```

Out[82]:
```
array([[[-0.01705812,  0.01705812],
        [-0.00324323,  0.00324323],
        [ 0.01074787, -0.01074787],
        ...,
        [ 0.00316299, -0.00316299],
        [ 0.01174553, -0.01174553],
        [-0.00085806,  0.00085806]],

       [[-0.06004146,  0.06004146],
        [-0.02078165,  0.02078165],
        [-0.10477762,  0.10477762],
        ...,
        [ 0.00354199, -0.00354199],
        [-0.0786794 ,  0.0786794 ],
        [-0.00128724,  0.00128724]],

       [[ 0.14512062, -0.14512062],
        [-0.00318367,  0.00318367],
        [ 0.01134319, -0.01134319],
        ...,
        [ 0.00343615, -0.00343615],
        [ 0.01309692, -0.01309692],
        [-0.00030635,  0.00030635]],

       ...,

       [[ 0.09931458, -0.09931458],
        [-0.00217132,  0.00217132],
        [ 0.0136331 , -0.0136331 ],
        ...,
        [ 0.0025807 , -0.0025807 ],
        [-0.0269416 ,  0.0269416 ],
        [-0.00099374,  0.00099374]],

       [[-0.04365624,  0.04365624],
        [-0.00509587,  0.00509587],
        [-0.14148   ,  0.14148   ],
        ...,
        [ 0.00240381, -0.00240381],
        [-0.07047572,  0.07047572],
        [-0.00258187,  0.00258187]],

       [[-0.07579842,  0.07579842],
        [ 0.03022751, -0.03022751],
        [-0.16315904,  0.16315904],
        ...,
        [ 0.00330333, -0.00330333],
        [-0.06217408,  0.06217408],
        [-0.00204865,  0.00204865]]])
```

In [86]:
```python
attributions3[:,:,1].shape
```

Out[86]:
```
(200, 12)
```

In [ ]: