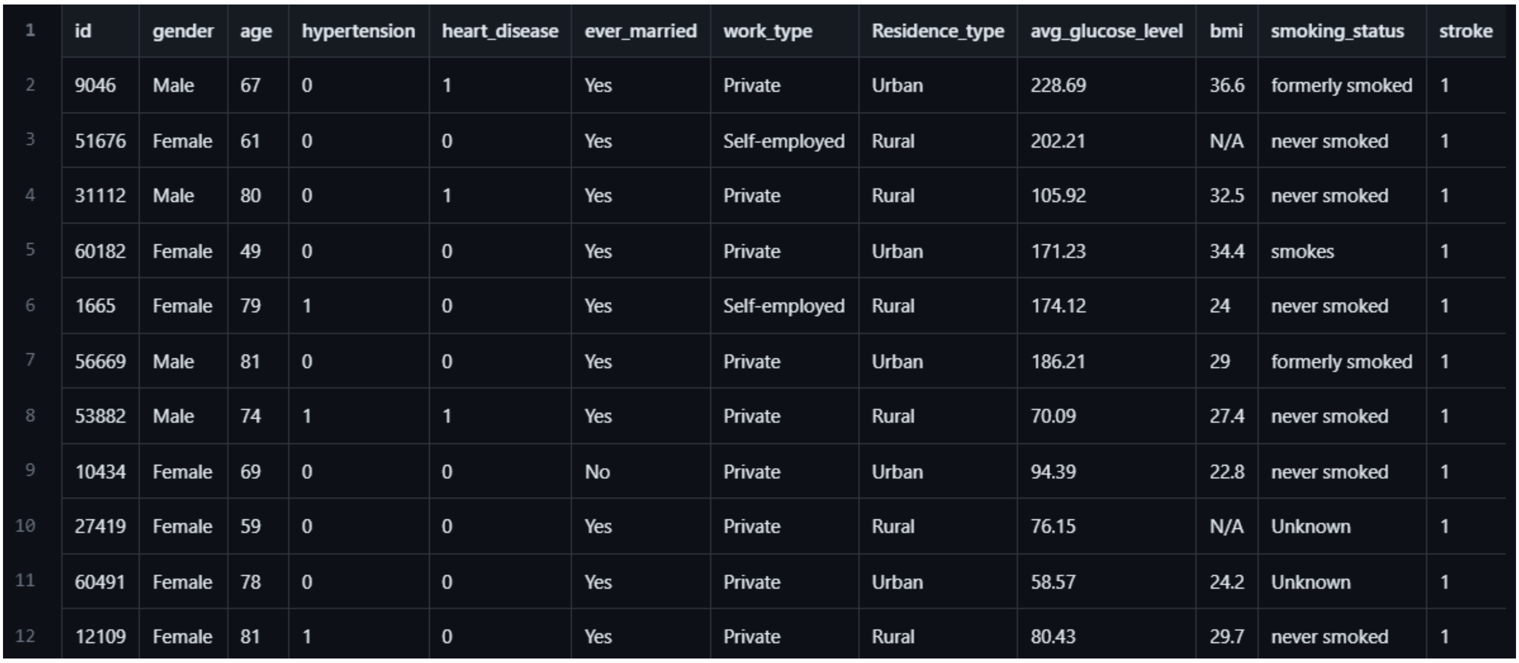
Stroke prediction using Explainable AI



id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
9046	Male	67	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
51676	Female	61	0	0	Yes	Self-employed	Rural	202.21	N/A	never smoked	1
31112	Male	80	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
60182	Female	49	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
1665	Female	79	1	0	Yes	Self-employed	Rural	174.12	24	never smoked	1
56669	Male	81	0	0	Yes	Private	Urban	186.21	29	formerly smoked	1
53882	Male	74	1	1	Yes	Private	Rural	70.09	27.4	never smoked	1
10434	Female	69	0	0	No	Private	Urban	94.39	22.8	never smoked	1
27419	Female	59	0	0	Yes	Private	Rural	76.15	N/A	Unknown	1
60491	Female	78	0	0	Yes	Private	Urban	58.57	24.2	Unknown	1
12109	Female	81	1	0	Yes	Private	Rural	80.43	29.7	never smoked	1

.

.

.

.

This dataset is related to stroke prediction, as evidenced by the "stroke" column, which indicates whether an individual had a stroke (1) or not (0).

This dataset appears to be suitable for building a predictive model to determine the likelihood of stroke based on these attributes. Explainable AI techniques can be applied to this model to provide clear and interpretable explanations for the model's predictions. These explanations can help in understanding which factors contribute most to the risk of stroke and how the model arrives at its predictions, which is crucial in healthcare decision-making and patient care.

Python Code:

```
# Import necessary libraries and modules
from utils import DataLoader
from interpret.glassbox import LogisticRegression, ClassificationTree, ExplainableBoostingClassifier
from interpret import show
from sklearn.metrics import f1_score, accuracy_score

# Load and preprocess data using DataLoader
data_loader = DataLoader()
data_loader.load_dataset()
data_loader.preprocess_data()

# Split the data for evaluation
X_train, X_test, y_train, y_test = data_loader.get_data_split()
print("Training data shape:", X_train.shape)
print("Testing data shape:", X_test.shape)

# Oversample the training data to address class imbalance
X_train, y_train = data_loader.oversample(X_train, y_train)
print("Training data shape after oversampling:", X_train.shape)

# Fit Logistic Regression model
lr = LogisticRegression(
    random_state=2021,
    feature_names=X_train.columns,
    penalty='l1',
    solver='liblinear'
)
lr.fit(X_train, y_train)
print("Training Logistic Regression finished.")

# Evaluate Logistic Regression model
y_pred = lr.predict(X_test)
print(f"Logistic Regression - F1 Score: {f1_score(y_test, y_pred, average='macro')}")
print(f"Logistic Regression - Accuracy: {accuracy_score(y_test, y_pred)}")

# Explain local prediction for Logistic Regression
lr_local = lr.explain_local(X_test[:100], y_test[:100], name='Logistic Regression Local')
show(lr_local)

# Explain global Logistic Regression model
lr_global = lr.explain_global(name='Logistic Regression Global')
show(lr_global)

# Fit Decision Tree model
tree = ClassificationTree()
```

```
tree.fit(X_train, y_train)
print("Training Decision Tree finished.")

# Evaluate Decision Tree model
y_pred = tree.predict(X_test)
print(f"Decision Tree - F1 Score: {f1_score(y_test, y_pred, average='macro')}")
print(f"Decision Tree - Accuracy: {accuracy_score(y_test, y_pred)}")

# Explain local prediction for Decision Tree
tree_local = tree.explain_local(X_test[:100], y_test[:100], name='Decision Tree Local')
show(tree_local)

# Fit Explainable Boosting Machine (EBM)
ebm = ExplainableBoostingClassifier(random_state=2021)
ebm.fit(X_train, y_train)
print("Training EBM finished.")

# Evaluate EBM model
y_pred = ebm.predict(X_test)
print(f"EBM - F1 Score: {f1_score(y_test, y_pred, average='macro')}")
print(f"EBM - Accuracy: {accuracy_score(y_test, y_pred)}")

# Explain local predictions for EBM
ebm_local = ebm.explain_local(X_test[:100], y_test[:100], name='EBM Local')
show(ebm_local)

# Explain global behavior of EBM
ebm_global = ebm.explain_global(name='EBM Global')
show(ebm_global)
```

Explanation of this code

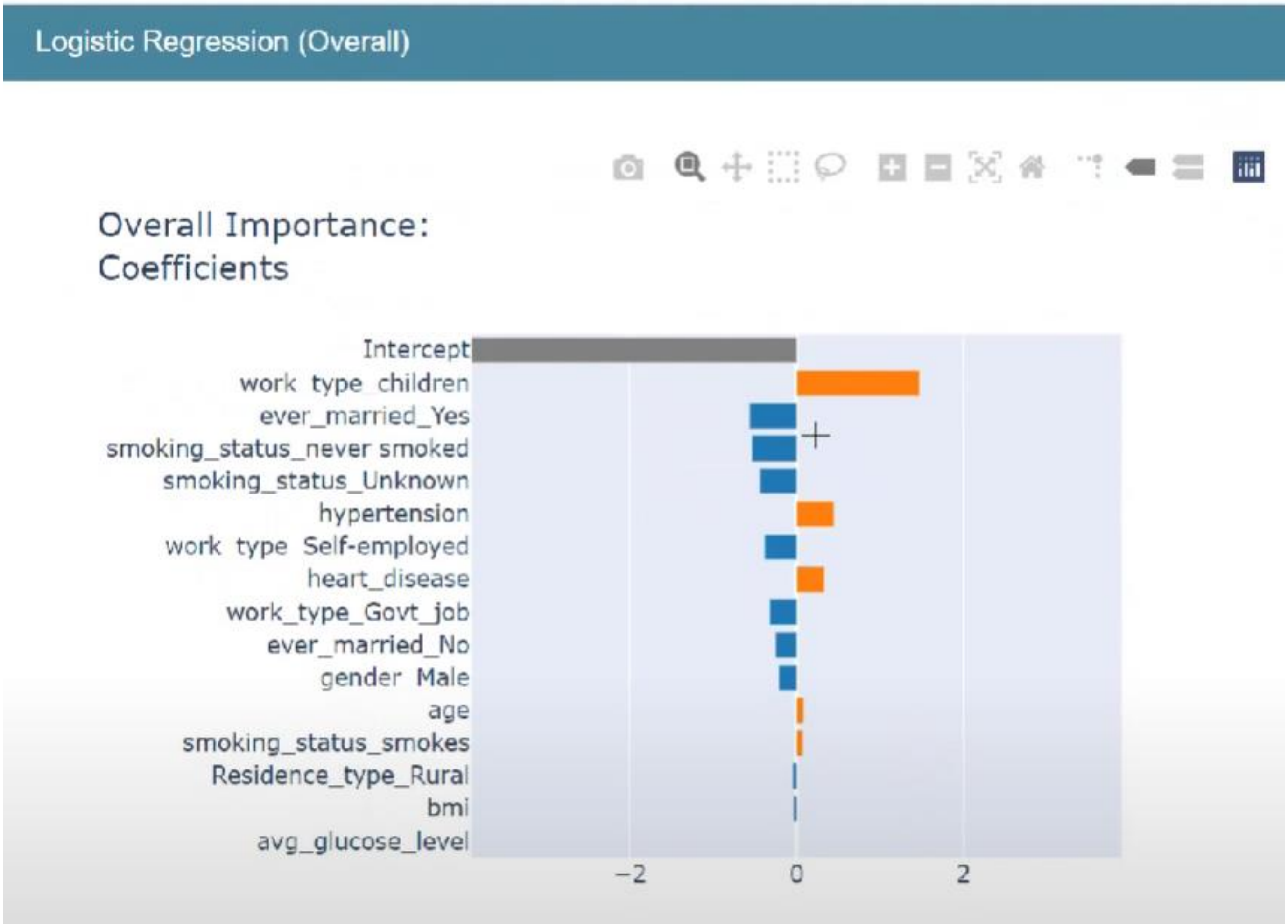
Evaluating three different machine learning models (Logistic Regression, Decision Tree, and Explainable Boosting Machine) for a classification task. We're also using the InterpretML library to explain these models.

1. Imports: We import the necessary libraries and modules, including the DataLoader from our custom utils module for data loading, as well as machine learning and interpretability modules from InterpretML.

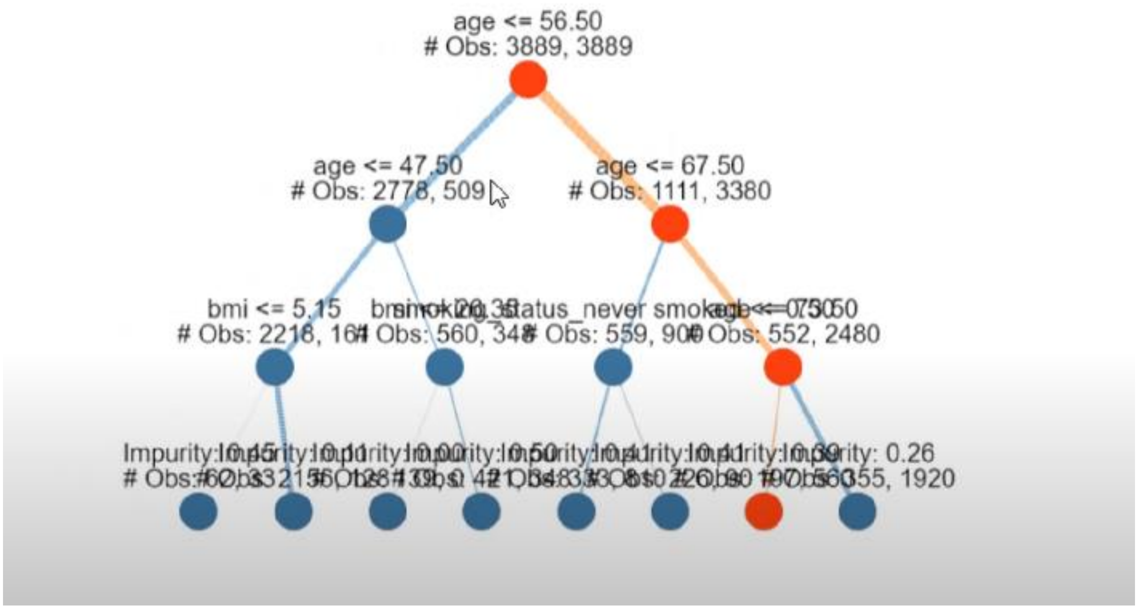
2. Load and preprocess data: We load the dataset using the DataLoader`and preprocess it. This includes feature engineering, handling missing values, encoding categorical variables, and scaling or normalizing features.

3. Data Splitting: We split the preprocessed data into training (X_train, y_train) and testing (X_test, y_test) sets. Additionally, we oversample the training data to address class imbalance.

4. Logistic Regression Model:
  - We fit a Logistic Regression model with specific hyperparameters like L1 penalty and Liblinear solver.
  - We evaluate the model's performance using F1 Score and Accuracy.
  - We explain local predictions for the Logistic Regression model on a subset of the test data.
  - We also explain the global behavior of the Logistic Regression model.

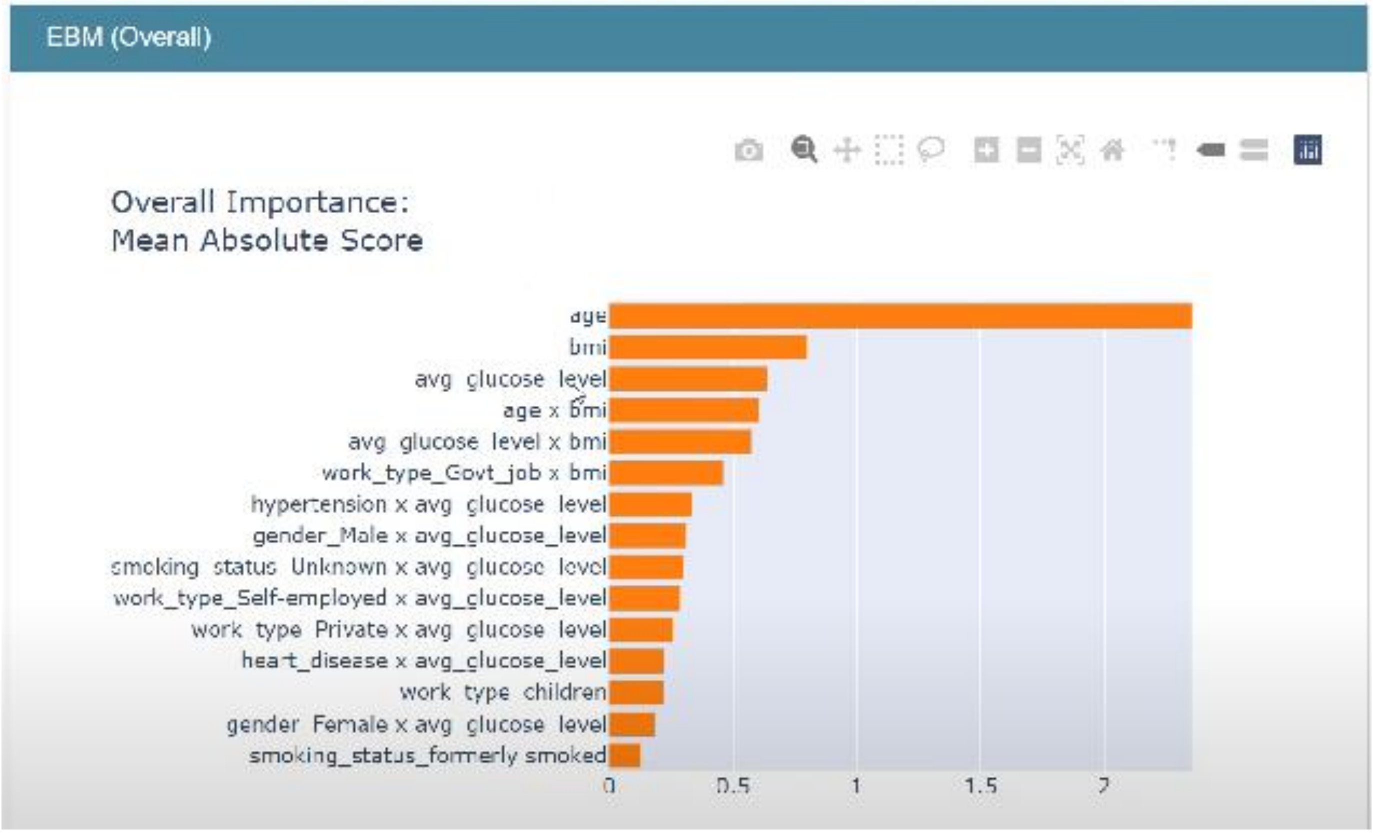


5. Decision Tree Model:
  - We fit a Classification Tree model.
  - We evaluate the Decision Tree model's performance using F1 Score and Accuracy.
  - Similar to the Logistic Regression model, We explain local predictions for the Decision Tree model.



6. Explainable Boosting Machine (EBM):

- We evaluate the EBM's performance using F1 Score and Accuracy.
- We explain local predictions for the EBM model on a subset of the test data.
- We also explain the global behavior of the EBM model.



We are following a structured approach to model training, evaluation, and interpretation using InterpretML. The explanations generated for local and global behavior help in understanding and potentially improving model transparency and trustworthiness.

LIME:

```
# Import necessary libraries and modules

from utils import DataLoader
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score, accuracy_score
from interpret.blackbox import LimeTabular
from interpret import show

# Load and preprocess data using DataLoader
data_loader = DataLoader()
data_loader.load_dataset()
data_loader.preprocess_data()

# Split the data for evaluation
X_train, X_test, y_train, y_test = data_loader.get_data_split()
print("Training data shape:", X_train.shape)
print("Testing data shape:", X_test.shape)

# Oversample the training data to address class imbalance
X_train, y_train = data_loader.oversample(X_train, y_train)
print("Training data shape after oversampling:", X_train.shape)

# Fit a RandomForestClassifier model
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
print("Training RandomForestClassifier finished.")

# Evaluate the RandomForestClassifier model
y_pred = rf.predict(X_test)
f1 = f1_score(y_test, y_pred, average='macro')
accuracy = accuracy_score(y_test, y_pred)
print(f"RandomForestClassifier - F1 Score: {f1}")
print(f"RandomForestClassifier - Accuracy: {accuracy}")

# Apply LIME (Local Interpretable Model-agnostic Explanations)
# Initialize LimeTabular
lime = LimeTabular(predict_fn=rf.predict_proba, data=X_train, random_state=1)

# Generate local explanations using LIME
lime_local = lime.explain_local(X_test[-20:], y_test[-20:], name='LIME')

# Show LIME Local Explanations
show(lime_local)
```

Explaining a machine learning model using the RandomForestClassifier and LimeTabular from the InterpretML library.

1. Imports: In this section, we import the necessary libraries and modules for our task, including DataLoader for data loading and preprocessing, RandomForestClassifier for building a random forest model, and LimeTabular for generating local explanations using LIME (Local Interpretable Model-agnostic Explanations).

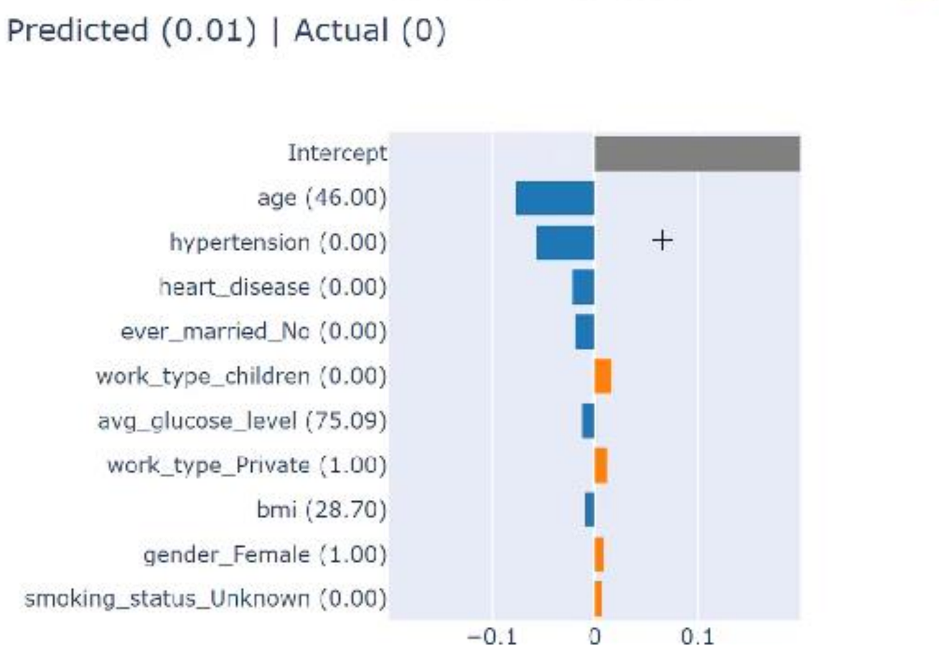
2. Load and preprocess data: You create an instance of the DataLoader class, load the dataset, and preprocess it. Data preprocessing likely includes tasks such as handling missing values, encoding categorical variables, and scaling/normalizing features.

3. Data Splitting: We split the preprocessed data into training (X_train, y_train) and testing (X_test, y_test) sets. Additionally, We perform oversampling on the training data to address class imbalance.

4. Fit blackbox model (Random Forest): We create a RandomForestClassifier model (`rf`) and fit it to the training data. This model is considered a "blackbox" because it doesn't provide inherent interpretability. We evaluate the model's performance using F1 Score and Accuracy on the test data.

5. Apply LIME (Local Interpretable Model-agnostic Explanations):
  - Initialize LimeTabular with the following parameters:
    - predict_fn: The prediction function of the RandomForestClassifier (rf.predict_proba), which returns class probabilities.
    - data: The training data (X_train) to be used by LIME.
    - random_state: A specified random seed for reproducibility.
  - Generate local explanations using LIME for the last 20 instances in the test data (X_test[-20:], y_test[-20:]) and assign the explanation results to lime_local.

6. Show LIME Local Explanations: We use the show() function from the InterpretML library to display the local explanations generated by LIME. These explanations help understand how the random forest model arrived at specific predictions for the selected instances.



This code demonstrates a typical workflow for building and interpreting a machine learning model. LIME is used to provide local interpretability for a blackbox model like Random Forest, making it easier to understand model predictions for specific data points.

SHAP:

```
# Import necessary libraries and modules
from utils import DataLoader
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score, accuracy_score
import shap

# Load and preprocess data using DataLoader
data_loader = DataLoader()
data_loader.load_dataset()
data_loader.preprocess_data()

# Split the data for evaluation
X_train, X_test, y_train, y_test = data_loader.get_data_split()
print("Training data shape:", X_train.shape)
print("Testing data shape:", X_test.shape)

# Oversample the training data to address class imbalance
X_train, y_train = data_loader.oversample(X_train, y_train)
print("Training data shape after oversampling:", X_train.shape)
# Fit a RandomForestClassifier model
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
print("Training RandomForestClassifier finished.")

# Evaluate the RandomForestClassifier model
y_pred = rf.predict(X_test)
f1 = f1_score(y_test, y_pred, average='macro')
accuracy = accuracy_score(y_test, y_pred)
print(f"RandomForestClassifier - F1 Score: {f1}")
print(f"RandomForestClassifier - Accuracy: {accuracy}")

# Create SHAP explainer using TreeExplainer
explainer = shap.TreeExplainer(rf)
# Calculate SHAP values for a specific subset of test data (start_index to end_index)
start_index = 1
end_index = 2
shap_values = explainer.shap_values(X_test[start_index:end_index])
# Visualize local predictions using force plot
shap.initjs()
prediction = rf.predict(X_test[start_index:end_index])[0]
print(f"The RF predicted: {prediction}")
shap.force_plot(explainer.expected_value[1],
            shap_values[1],
            X_test[start_index:end_index])

# Visualize global feature importance using summary plot
shap.summary_plot(shap_values, X_test)
```

Data Loading and Preprocessing: Loading and pre-processing the dataset using the DataLoader class, including oversampling to address class imbalance.

Model Training and Evaluation: Fitting a RandomForestClassifier model to the preprocessed training data and evaluating its performance using F1 Score and Accuracy on the test data.

SHAP Explanation:

Creating a SHAP explainer (explainer) using TreeExplainer for the trained RandomForestClassifier. Calculating SHAP values for a specific subset of test data (from start_index to end_index) to explain local predictions.

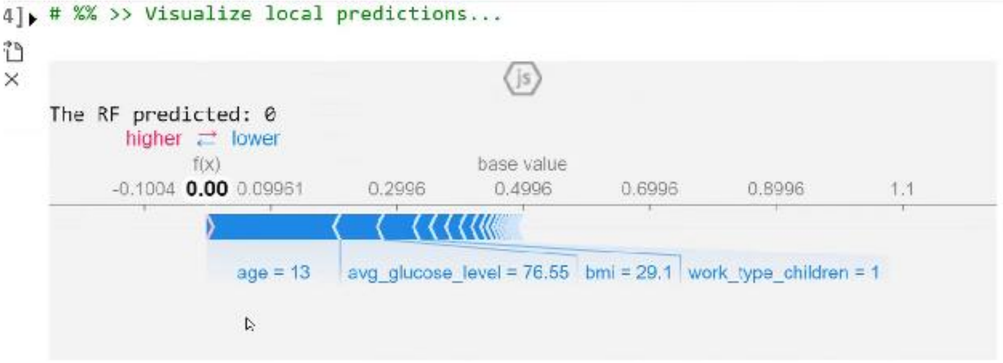
Local Interpretability:

Visualizing local predictions using a force plot to understand how features contribute to a specific prediction.
Displaying the RandomForestClassifier's prediction for the selected test data point.

Global Feature Importance:

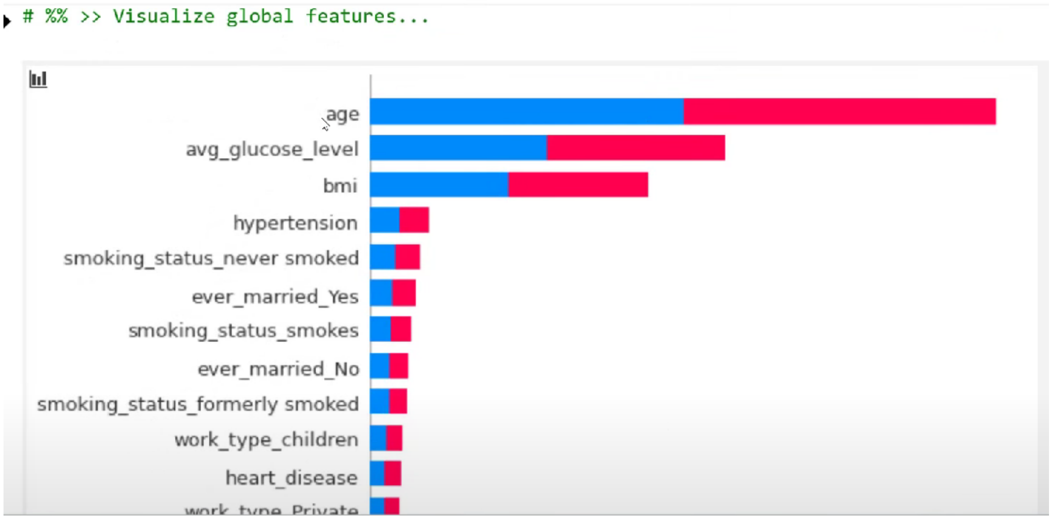
Visualizing global feature importance using a summary plot to understand which features have the most significant impact on model predictions across the entire dataset.

This code allows us to gain insights into both local and global model behavior using SHAP values, providing valuable interpretability for the RandomForestClassifier model.



```
4] # %% >> Visualize local predictions...
```

```
The RF predicted: 0
```



```
# %% >> Visualize global features...
```