In [2]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn import tree
import matplotlib.pyplot as plt
import pandas as pd
```

In [3]:
```
data = pd.read_excel("healthcare-stroke-data.xlsx")
data
```

Out[3]:

| | gender | age | hypertension | ever_married | work_type | Residence_type | avg_glucose_level | bmi | smoking_status | stroke |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Male | 67.0 | 0 | Yes | Private | Urban | 228.69 | 36.6 | formerly smoked | 1 |
| **1** | Female | 61.0 | 0 | Yes | Self-employed | Rural | 202.21 | 32.5 | never smoked | 1 |
| **2** | Male | 80.0 | 0 | Yes | Private | Rural | 105.92 | 32.5 | never smoked | 1 |
| **3** | Female | 49.0 | 0 | Yes | Private | Urban | 171.23 | 34.4 | smokes | 1 |
| **4** | Female | 79.0 | 1 | Yes | Self-employed | Rural | 174.12 | 24.0 | never smoked | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **5105** | Female | 80.0 | 1 | Yes | Private | Urban | 83.75 | 32.5 | never smoked | 0 |
| **5106** | Female | 81.0 | 0 | Yes | Self-employed | Urban | 125.20 | 40.0 | never smoked | 0 |
| **5107** | Female | 35.0 | 0 | Yes | Self-employed | Rural | 82.99 | 30.6 | never smoked | 0 |
| **5108** | Male | 51.0 | 0 | Yes | Private | Rural | 166.29 | 25.6 | formerly smoked | 0 |
| **5109** | Female | 44.0 | 0 | Yes | Govt_job | Urban | 85.28 | 26.2 | Unknown | 0 |

5110 rows × 10 columns

In [6]:
```
missing_values = data.isnull().sum()
print("Missing values in each column:")
print(missing_values)
```

```
Missing values in each column:
gender               0
age                  0
hypertension         0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                  0
smoking_status       0
stroke               0
dtype: int64
```

In [8]:
```python
from sklearn.preprocessing import LabelEncoder

# List of columns to label encode
columns_to_encode = ['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status']

# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Apply LabelEncoder to each column in the list
for column in columns_to_encode:
    data[column] = label_encoder.fit_transform(data[column])

# Display the updated DataFrame
print(data.head())
```

```
   gender   age  hypertension  ever_married  work_type  Residence_type  \
0       1  67.0             0             1          2               1
1       0  61.0             0             1          3               0
2       1  80.0             0             1          2               0
3       0  49.0             0             1          2               1
4       0  79.0             1             1          3               0

   avg_glucose_level   bmi  smoking_status  stroke
0             228.69  36.6               1       1
1             202.21  32.5               2       1
2             105.92  32.5               2       1
3             171.23  34.4               3       1
4             174.12  24.0               2       1
```

In [10]:
```
data.shape
```

Out[10]:
```
(5110, 10)
```

In [12]:
```
data.columns
```

Out[12]:
```
Index(['gender', 'age', 'hypertension', 'ever_married', 'work_type',
       'Residence_type', 'avg_glucose_level', 'bmi', 'smoking_status',
       'stroke'],
      dtype='object')
```

In [14]:
```
X= data[['gender', 'age', 'hypertension', 'ever_married', 'work_type',
       'Residence_type', 'avg_glucose_level', 'bmi', 'smoking_status']]
y=data['stroke']
```

In [16]:
```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [18]:
```
from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings("ignore")

# Logistic Regression
logistic_model = LogisticRegression()
logistic_model.fit(x_train, y_train)
y_pred_logistic = logistic_model.predict(x_test)

# Decision Tree Classifier
decision_tree_model = DecisionTreeClassifier()
decision_tree_model.fit(x_train, y_train)
y_pred_tree = decision_tree_model.predict(x_test)

# Define a function to evaluate models
def evaluate_model(y_true, y_pred):
    print("Accuracy:", accuracy_score(y_true, y_pred))
    print("Precision:", precision_score(y_true, y_pred))
    print("Recall:", recall_score(y_true, y_pred))
    print("F1 Score:", f1_score(y_true, y_pred))

print("Logistic Regression Performance:")
evaluate_model(y_test, y_pred_logistic)
```

```
print("\nDecision Tree Performance:")
evaluate_model(y_test, y_pred_tree)
```

```
Logistic Regression Performance:
Accuracy: 0.9393346379647749
Precision: 0.0
Recall: 0.0
F1 Score: 0.0

Decision Tree Performance:
Accuracy: 0.9060665362035225
Precision: 0.17307692307692307
Recall: 0.14516129032258066
F1 Score: 0.15789473684210525
```

In [20]:
```
# Visualize the Decision Tree
plt.figure(figsize=(18, 12))
tree.plot_tree(decision_tree_model, feature_names=X.columns, class_names=['1','0'], filled=True, rounded=True, fonts:
plt.title("Decision Tree Visualization")
plt.show()
```


In [ ]:
```
#pruning randomly selected max_depth = 6
```

In [22]:
```
decision_tree_model_2 = DecisionTreeClassifier(
     random_state=42,
    max_depth=6,                # Limit the maximum depth of the tree
    min_samples_split=10,       # Minimum number of samples required to split an internal node
    min_samples_leaf=5          # Minimum number of samples required to be a leaf node
)
```

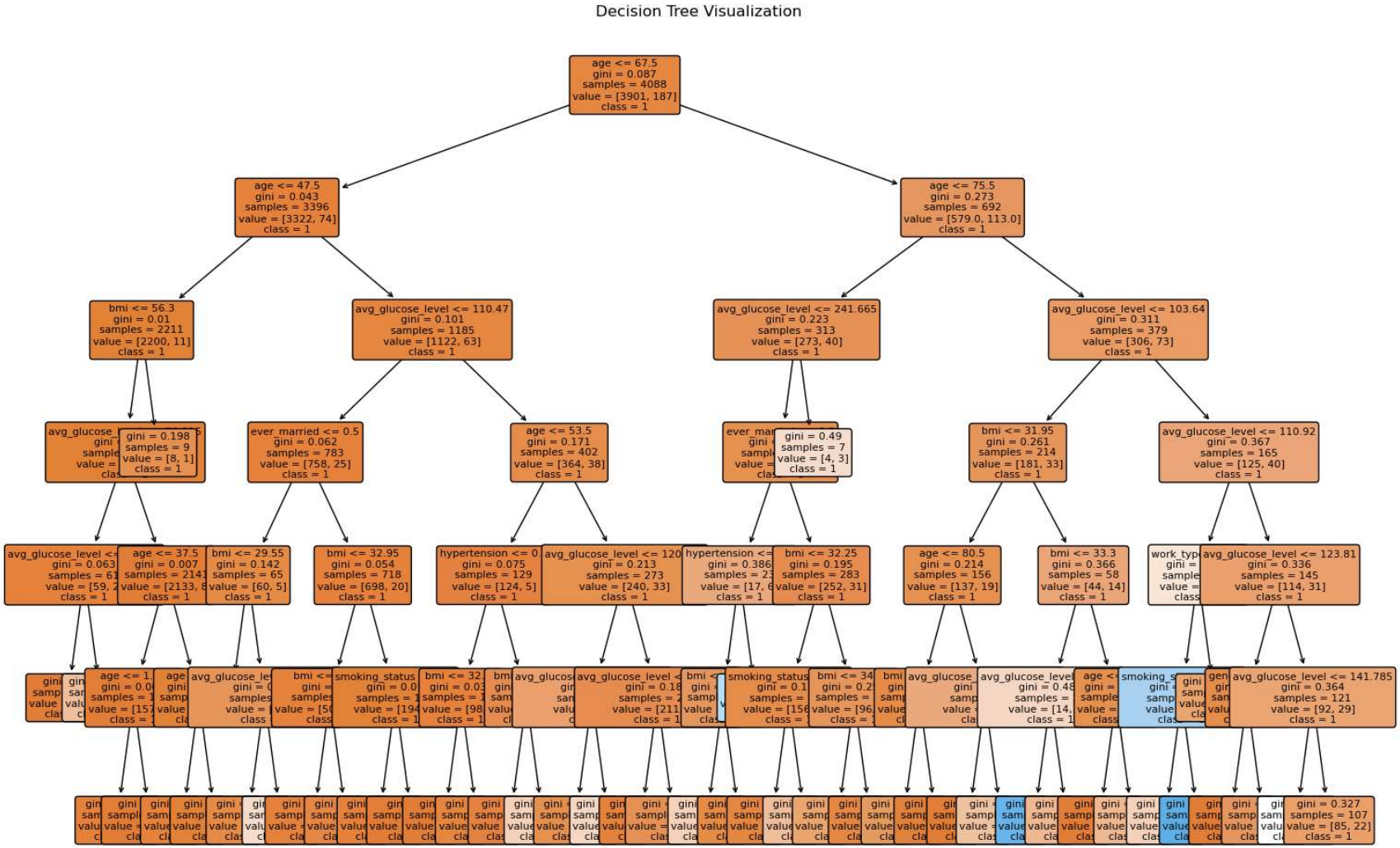
```
decision_tree_model_2.fit(x_train, y_train)
y_pred_tree = decision_tree_model_2.predict(x_test)
```

In [24]:
```
def evaluate_model(y_true, y_pred):
    print("Accuracy:", accuracy_score(y_true, y_pred))
    print("Precision:", precision_score(y_true, y_pred))
    print("Recall:", recall_score(y_true, y_pred))
    print("F1 Score:", f1_score(y_true, y_pred))
```

In [26]:
```
print("\nDecision Tree Performance:")
evaluate_model(y_test, y_pred_tree)
```

```
Decision Tree Performance:
Accuracy: 0.9373776908023483
Precision: 0.3333333333333333
Recall: 0.03225806451612903
F1 Score: 0.058823529411764705
```

In [28]:
```
# Visualize the Decision Tree
plt.figure(figsize=(18, 12))
tree.plot_tree(decision_tree_model_2, feature_names=X.columns, class_names=['1','0'], filled=True, rounded=True, font
plt.title("Decision Tree Visualization")
plt.show()
```




In [30]:
```python
# 'entropy' instead of 'gini'
from sklearn.tree import DecisionTreeClassifier

decision_tree_model_3 = DecisionTreeClassifier(
    random_state=42,
    criterion="entropy",       # Use 'entropy' instead of 'gini'
    max_depth=6,               # Limit the maximum depth of the tree
    min_samples_split=10,      # Minimum number of samples required to split an internal node
    min_samples_leaf=5         # Minimum number of samples required to be a leaf node
```

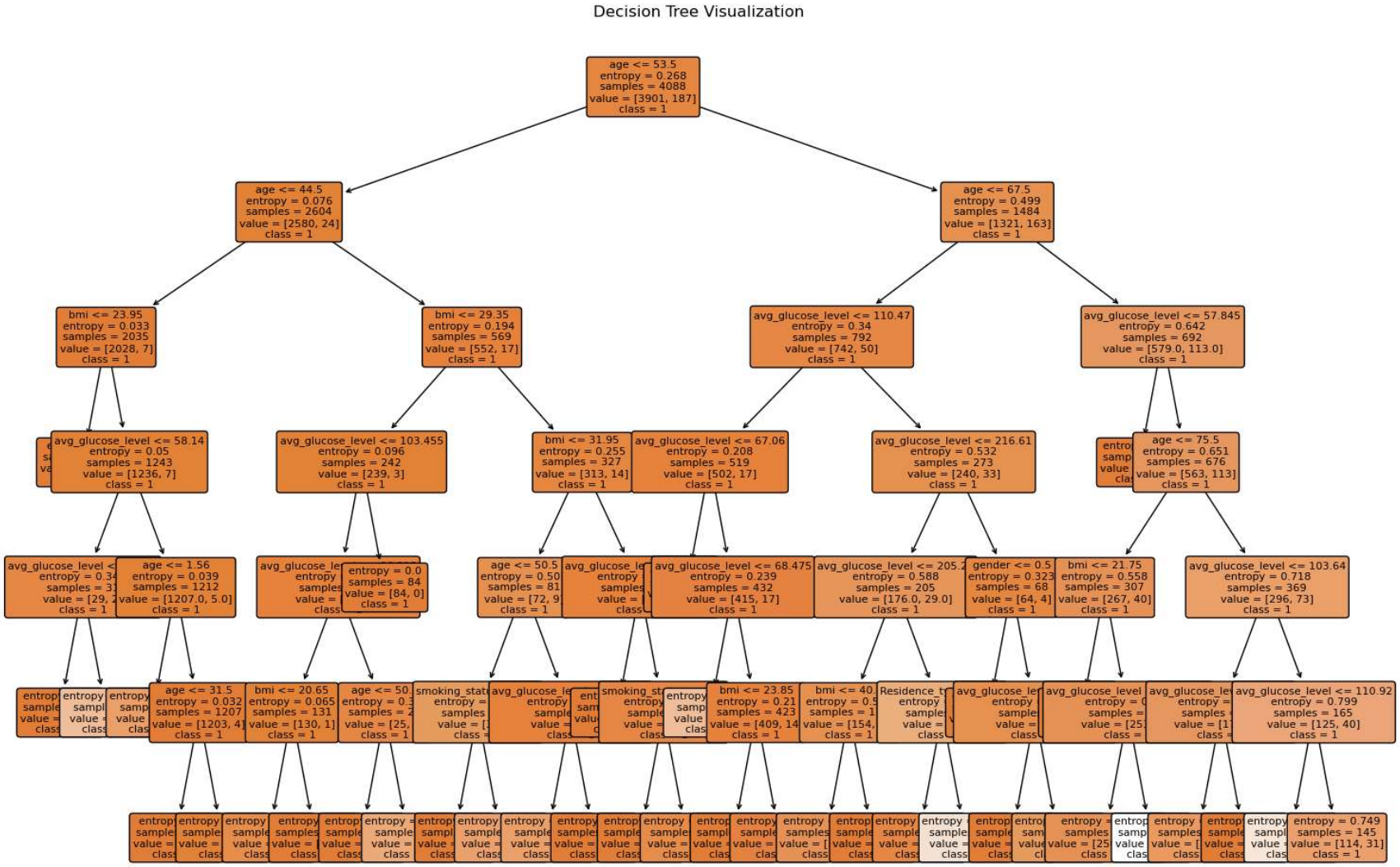
```
)
decision_tree_model_3.fit(x_train, y_train)
y_pred_tree = decision_tree_model_3.predict(x_test)
```

In [32]:
```
def evaluate_model(y_true, y_pred):
    print("Accuracy:", accuracy_score(y_true, y_pred))
    print("Precision:", precision_score(y_true, y_pred))
    print("Recall:", recall_score(y_true, y_pred))
    print("F1 Score:", f1_score(y_true, y_pred))
```

In [34]:
```
print("\nDecision Tree Performance:")
evaluate_model(y_test, y_pred_tree)
```

```
Decision Tree Performance:
Accuracy: 0.9393346379647749
Precision: 0.0
Recall: 0.0
F1 Score: 0.0
```

In [36]:
```
# Visualize the Decision Tree
plt.figure(figsize=(18, 12))
tree.plot_tree(decision_tree_model_3, feature_names=X.columns, class_names=['1','0'], filled=True, rounded=True, font
plt.title("Decision Tree Visualization")
plt.show()
```




# HYPER PARAMETER TUNING

gridsearchcv = It evaluates all possible combinations based on the parameters provided. Not suitable for large datasets.

randomsearchcv = It evaluates only random combinations based on the parameters provided. Suitable for large datasets.

Large datasets = 50000 rows and 30 columns (typical)

In [24]:
```
#Hyper parameter tuning

parameters={
'criterion':['gini', 'entropy'],
'max_depth':[1,2,3,4,5],
'max_features': ['log2', 'sqrt']
}

#gridsearchcv, randomsearchcv
from sklearn.model_selection import GridSearchCV
model=DecisionTreeClassifier()
cv=GridSearchCV(model,parameters, scoring=['accuracy'],refit='accuracy')

cv.fit(x_train,y_train)
```

Out[24]:

▸ GridSearchCV

▸ estimator: DecisionTreeClassifier

▸ DecisionTreeClassifier

In [26]:
```
cv.best_estimator_
```

Out[26]:

▾ DecisionTreeClassifier

```
DecisionTreeClassifier(max_depth=3, max_features='log2')
```

In [28]:
```
p3=cv.predict(x_test)
accuracy_score(y_test,p3)
```

Out[28]:
```
0.9403131115459883
```

# KNN Classification

In [31]:
```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors = 4)
```

In [33]:
```
neigh.fit(x_train,y_train)
```

Out[33]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=4)
```

In [35]:
```
yp=neigh.predict(x_test)
```

In [37]:
```
accuracy_score(y_test,yp)
```

Out[37]:
```
0.9363992172211351
```

Research: The distance between data point and training sets can be determined using distance metrices.

Euclidean

Manhattan