Importing all the required Libraries

In [1]:
```
import pandas as pd
import numpy as np
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn import metrics
# Evaluation metrics related methods
from sklearn.metrics import classification_report, accuracy_score, f1_score, confusion_matrix, precision_recall

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Loading Dataset as pandas DataFrame and check dataframe head

In [4]:
```
df = pd.read_csv('diabetes_dataset.csv')
df.head()
```

Out[4]:

	year	gender	age	location	race:AfricanAmerican	race:Asian	race:Caucasian	race:Hispanic	race:Other	hypertension	heart_disease	sn
0	2020	Female	32.0	Alabama	0	0	0	0	1	0	0	
1	2015	Female	29.0	Alabama	0	1	0	0	0	0	0	
2	2015	Male	18.0	Alabama	0	0	0	0	1	0	0	
3	2015	Male	41.0	Alabama	0	0	1	0	0	0	0	
4	2016	Female	52.0	Alabama	1	0	0	0	0	0	0	

Checking if any duplicates present in the dataset

In [7]:
```
df.duplicated().sum()
```

Out[7]:
```
14
```

Dropping duplicates

In [10]:
```
df = df.drop_duplicates()
```

In [12]:
```
df.head()
```

Out[12]:

	year	gender	age	location	race:AfricanAmerican	race:Asian	race:Caucasian	race:Hispanic	race:Other	hypertension	heart_disease	sn
0	2020	Female	32.0	Alabama	0	0	0	0	1	0	0	
1	2015	Female	29.0	Alabama	0	1	0	0	0	0	0	
2	2015	Male	18.0	Alabama	0	0	0	0	1	0	0	
3	2015	Male	41.0	Alabama	0	0	1	0	0	0	0	
4	2016	Female	52.0	Alabama	1	0	0	0	0	0	0	

In [14]:
```
df.shape
```

Out[14]:
```
(99986, 16)
```

Lets see any null values present in the dataframe

In [17]:
```
df.isnull().sum()
```

Out[17]:
```
year                   0
gender                 0
age                    0
location               0
race:AfricanAmerican   0
race:Asian             0
race:Caucasian         0
race:Hispanic          0
race:Other             0
hypertension           0
heart_disease          0
smoking_history        0
bmi                    0
hbA1c_level            0
blood_glucose_level    0
diabetes               0
dtype: int64
```

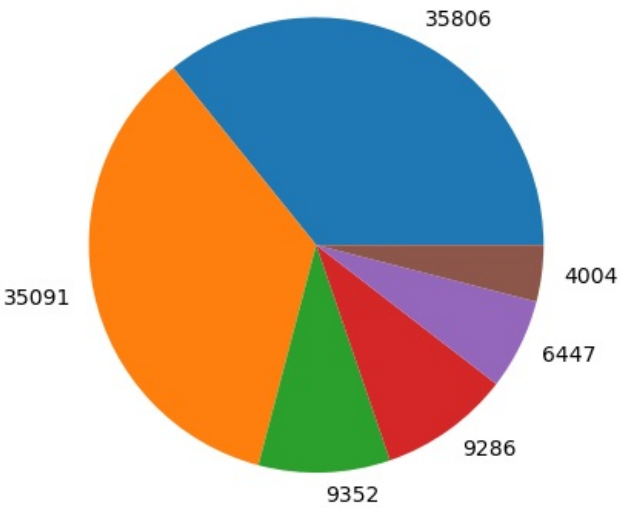
In [19]:
```
df['smoking_history'].unique()
```

Out[19]:
```
array(['never', 'not current', 'current', 'No Info', 'ever', 'former'],
      dtype=object)
```

In [21]:
```
y_plot = df['smoking_history'].value_counts()
y_plot
```

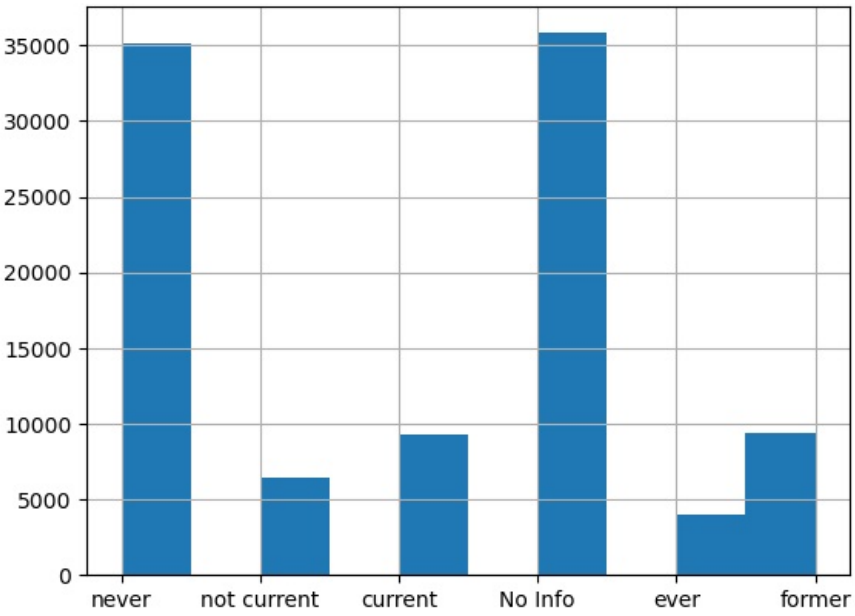
Out[21]:
```
No Info        35806
never          35091
former          9352
current         9286
not current     6447
ever            4004
Name: smoking_history, dtype: int64
```

In [23]:
```
plt.pie(y_plot,labels = y_plot)
plt.show()
```



In [25]:
```
df.smoking_history.hist()
```

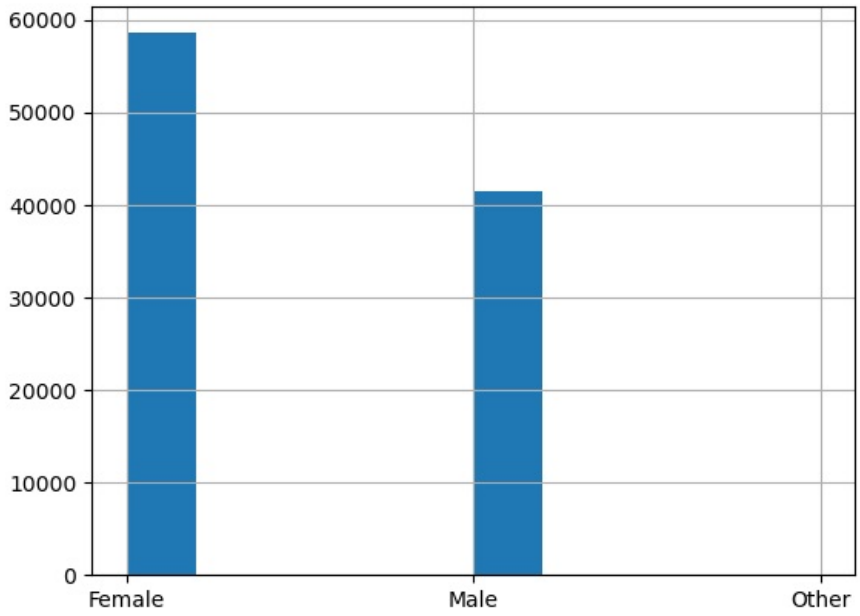
Out[25]:
```
<Axes: >
```



In [27]:
```
gender_plot = df['gender'].value_counts()
```

In [29]:
```
df.gender.hist()
```

Out[29]:
```
<Axes: >
```

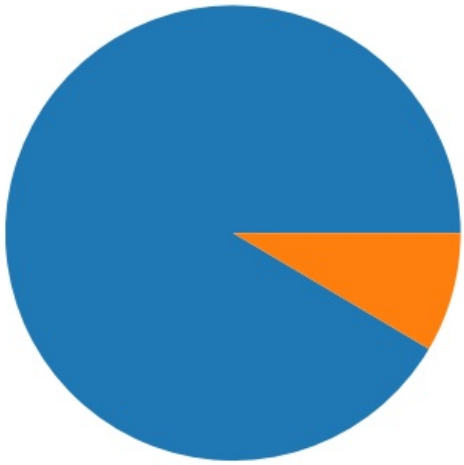


In [31]:
```
df['diabetes'].value_counts()
```

Out[31]:
```
0    91486
1     8500
Name: diabetes, dtype: int64
```

In [33]:
```
plt.pie(df['diabetes'].value_counts())

plt.show()
```



In [35]:
```
df.dtypes
```

Out[35]:
```
year                      int64
gender                   object
age                     float64
location                 object
race:AfricanAmerican      int64
race:Asian                int64
race:Caucasian            int64
race:Hispanic             int64
race:Other                int64
hypertension              int64
heart_disease             int64
smoking_history          object
bmi                     float64
hbA1c_level             float64
blood_glucose_level       int64
diabetes                  int64
dtype: object
```

Label Encoding

Model will not work with object data type so convert them into int or float

In [38]:
```
from sklearn.preprocessing import LabelEncoder
```

In [40]:
```
l = LabelEncoder()
```

In [42]:
```
cols = ['gender','location','smoking_history']
for column in cols:
    df[column] = l.fit_transform(df[column])
```

In [44]:
```
df.dtypes
```

Out[44]:
```
year                      int64
gender                    int32
age                     float64
location                  int32
race:AfricanAmerican      int64
race:Asian                int64
race:Caucasian            int64
race:Hispanic             int64
race:Other                int64
hypertension              int64
heart_disease             int64
smoking_history           int32
bmi                     float64
hbA1c_level             float64
blood_glucose_level       int64
diabetes                  int64
dtype: object
```

dividing DataFrame into x and y

In [47]:
```
x = df.drop('diabetes',axis = 1)
y = df['diabetes']
```

In [49]:
```
print(x.shape)
print(y.shape)
print(type(x))
print(type(y))
```

```
(99986, 15)
(99986,)
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
```

In [51]:
```
from sklearn.model_selection import train_test_split
```

Spliting the data into train and test data

In [54]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2)
```

In [56]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(79988, 15)
(19998, 15)
(79988,)
(19998,)
```

Train a default decision tree

Training a decision classifier is very straightforward with `sklearn`, we first need to define a `DecisionTreeClassifier` object. In the first step, we will use all the default arguments.

In [60]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [62]:
```
# Train a decision tree with all default arguments
m1 = DecisionTreeClassifier()
```

Then we can train the decision tree model with training and testing data

In [65]:
```
m1 = m1.fit(x_train,y_train)
```

And make predictions on the test data

In [68]:
```
y_pred = m1.predict(x_test)
```

Here we also provided a utility method to evaluate the trained decision tree model and output some standard evaluation metrics.

In [71]:
```
from sklearn.metrics import confusion_matrix,classification_report
```

In [73]:
```
cm = confusion_matrix(y_test,y_pred)
cf = classification_report(y_test,y_pred)
print(cm)
print(cf)
print('Training_score:',m1.score(x_train,y_train))
print('Testing_score:',m1.score(x_test,y_test))
```

```
[[17760   549]
 [  430  1259]]
              precision    recall  f1-score   support

           0       0.98      0.97      0.97     18309
           1       0.70      0.75      0.72      1689

    accuracy                           0.95     19998
   macro avg       0.84      0.86      0.85     19998
weighted avg       0.95      0.95      0.95     19998

Training_score: 0.9999874981247187
Testing_score: 0.9510451045104511
```

In [75]:
```
from sklearn.metrics import classification_report, accuracy_score, f1_score, confusion_matrix, precision_recall
```

In [77]:
```
def evaluate_metrics(yt, yp):
    results_pos = {}
    results_pos['accuracy'] = accuracy_score(yt, yp)
    precision, recall, f_beta, _ = precision_recall_fscore_support(yt, yp, average='binary')
    results_pos['recall'] = recall
    results_pos['precision'] = precision
    results_pos['f1score'] = f_beta
    return results_pos
```

In [79]:
```
evaluate_metrics(y_test, y_pred)
```

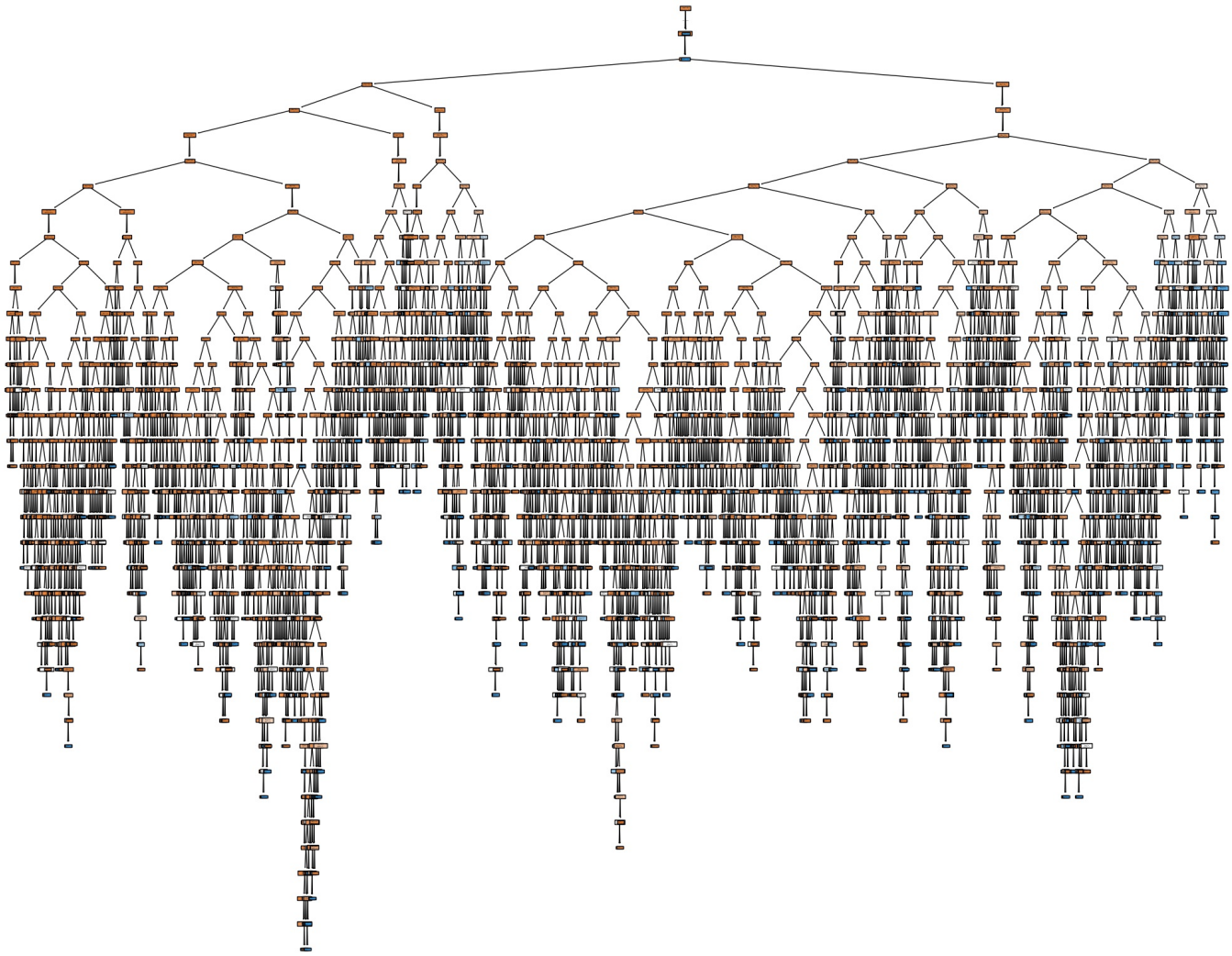
Out[79]:
```
{'accuracy': 0.9510451045104511,
 'recall': 0.7454114860864417,
 'precision': 0.6963495575221239,
 'f1score': 0.7200457535030026}
```

We will be using the `tree.plot_tree()` method provided by `sklearn` to quickly plot any decision tree model.

In [82]:
```
def plot_decision_tree(model, feature_names):
    plt.subplots(figsize=(25, 20))
    tree.plot_tree(model,
                   feature_names=feature_names,
                   filled=True)
    plt.show()
```

In [84]:
```
feature_names = x.columns.values
```

In [86]:
```
plot_decision_tree(m1, feature_names)
```



Cutomize the decision tree model

The `DecisionTreeClassifier` has many arguments (model hyperparameters) that can be customized and eventually tune the generated decision tree classifiers. Among these arguments, there are three commonly tuned arguments as follows:

- criterion: `gini` or `entropy`, which specifies which criteria to be used when splitting a tree node
- max_depth: a numeric value to specify the max depth of the tree. Larger tree depth normally means larger model complexity
- min_samples_leaf: The minimal number of samples in leaf nodes. Larger samples in leaf nodes will tend to generate simpler trees

Let's first try the following hyperparameter values:

- criterion = 'entropy'
- max_depth = 10
- min_samples_leaf=3

In [108…
```
m2 = DecisionTreeClassifier(criterion='entropy', max_depth=10, min_samples_leaf=3, random_state=42)
```

In [110…
```
m2.fit(x_train, y_train.values.ravel())
preds = m2.predict(x_test)
evaluate_metrics(y_test, preds)
```

Out[110]:
```
{'accuracy': 0.972047204720472,
 'recall': 0.679692125518058,
 'precision': 0.9845626072041166,
 'f1score': 0.8042031523642732}
```

Tune hyperparameters

Lastly, let's try to find the optimized hyperparameters, which can produce the highest F1 score, via GridSearch cross-validation.

We define a `params_grid` dict object to contain the parameter candidates:

In [115…
```
params_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 5]
}
```

In [117…
```
model = DecisionTreeClassifier(random_state=42)
```

In [119…
```
grid_search = GridSearchCV(estimator = model,
                           param_grid = params_grid,
                           scoring='f1',
                           cv = 5, verbose = 1)
grid_search.fit(x_train, y_train.values.ravel())
best_params = grid_search.best_params_
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
```

In [120…
```
best_params
```

Out[120]:
```
{'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 1}
```

Again building model with hyperparameters

In [130…
```
m3 = DecisionTreeClassifier(criterion='gini', max_depth=10, min_samples_leaf=1, random_state=42)
```

In [132…
```
m3.fit(x_train,y_train)
y_pred_final = m3.predict(x_test)
```

In [134…
```
cm = confusion_matrix(y_test,y_pred_final)
cf = classification_report(y_test,y_pred_final)
print(cm)
print(cf)
print('Training_score:',m3.score(x_train,y_train))
print('Testing_score:',m3.score(x_test,y_test))
```

```
[[18278    31]
 [  541  1148]]
              precision    recall  f1-score   support

           0       0.97      1.00      0.98     18309
           1       0.97      0.68      0.80      1689

    accuracy                           0.97     19998
   macro avg       0.97      0.84      0.89     19998
weighted avg       0.97      0.97      0.97     19998

Training_score: 0.9727084062609391
Testing_score: 0.9713971397139713
```

In [ ]:
```
```