IMPORTING LIBRARIES

```
import pandas as pd
import numpy as np
import xgboost
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder,  StandardScaler
from sklearn.metrics import classification_report
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras.callbacks import EarlyStopping
import seaborn as sns
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
```

LOADING THE DATASET

```
stroke_dataset = pd.read_csv("healthcare-dataset-stroke-data.csv")
```

```
stroke_dataset
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Resi
0	9046	Male	67.0	0	1	Yes	Private	
1	51676	Female	61.0	0	0	Yes	Self-employed	
2	31112	Male	80.0	0	1	Yes	Private	
3	60182	Female	49.0	0	0	Yes	Private	
4	1665	Female	79.0	1	0	Yes	Self-employed	
...	...	...	...	...	...	...	...	
5105	18234	Female	80.0	1	0	Yes	Private	
5106	44873	Female	81.0	0	0	Yes	Self-employed	
5107	19723	Female	35.0	0	0	Yes	Self-employed	
5108	37544	Male	51.0	0	0	Yes	Private	
5109	44679	Female	44.0	0	0	Yes	Govt_job	

5110 rows × 12 columns

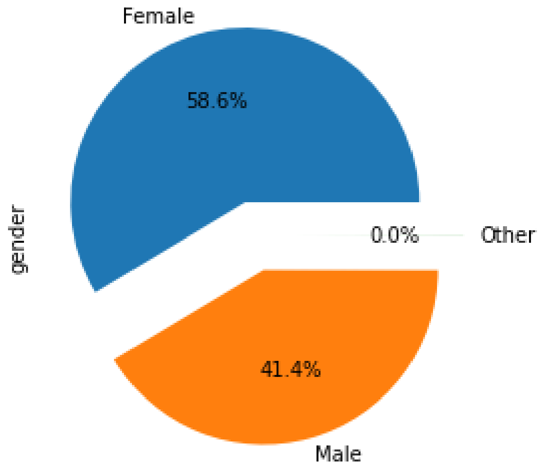
CONVERTING POSSIBLE VALUES TO CATEGORICAL VARIABLES

```
for col in stroke_dataset.columns:
    if stroke_dataset[col].dtype == 'object' or (stroke_dataset[col].dtype == 'int64' and
        print(col,"->", stroke_dataset[col].unique())
```

```
gender -> ['Male' 'Female' 'Other']
hypertension -> [0 1]
heart_disease -> [1 0]
ever_married -> ['Yes' 'No']
work_type -> ['Private' 'Self-employed' 'Govt_job' 'children' 'Never_worked']
Residence_type -> ['Urban' 'Rural']
smoking_status -> ['formerly smoked' 'never smoked' 'smokes' 'Unknown']
stroke -> [1 0]
```

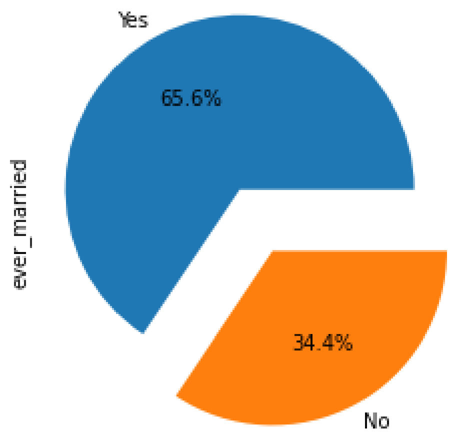
```
stroke_dataset['gender'].value_counts().plot.pie(autopct='%1.1f%%', explode=[0.2, 0.2, 0.2
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f47c9a2d940>
```



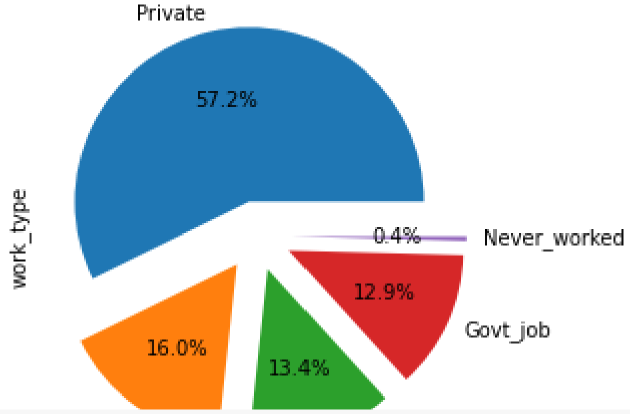

```
stroke_dataset['ever_married'].value_counts().plot.pie(autopct='%1.1f%%', explode=[0.2, 0.
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f47c81f5940>
```



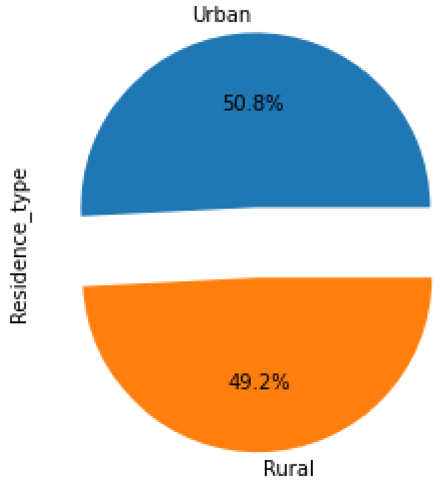

```
stroke_dataset['work_type'].value_counts().plot.pie(autopct='%1.1f%%', explode=[0.2, 0.2,
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f47c81aea90>
```



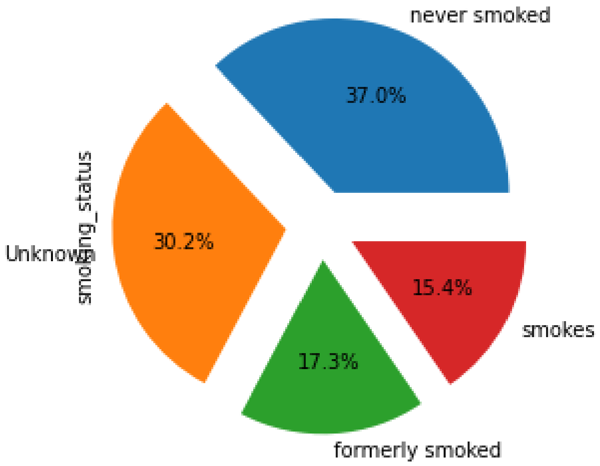

```
stroke_dataset['Residence_type'].value_counts().plot.pie(autopct='%1.1f%%', explode=[0.2,
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f47c817c670>
```



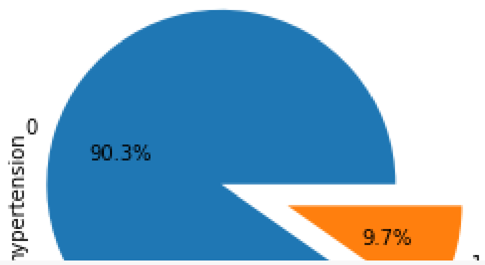

```
stroke_dataset['smoking_status'].value_counts().plot.pie(autopct='%1.1f%%', explode=[0.2,
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f47c80c0790>
```



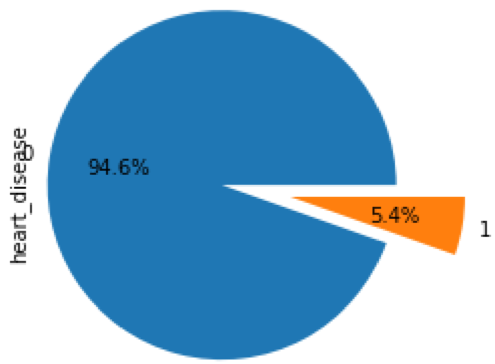

```
stroke_dataset['hypertension'].value_counts().plot.pie(autopct='%1.1f%%', explode=[0.2, 0.
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f47c9a0fac0>
```




```
stroke_dataset['heart_disease'].value_counts().plot.pie(autopct='%1.1f%%', explode=[0.2, 0
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f47c8044dc0>
```



IDENTIFYING THE NULL VALUES

```
stroke_dataset.isna().sum()
```

```
id                     0
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                  201
smoking_status         0
stroke                 0
dtype: int64
```

DATA PREPROCESSING

```
# Exclude id column
stroke_dataset.pop('id')
# exclude the rows containing Other and Never_worked
stroke_dataset = stroke_dataset[stroke_dataset.gender != 'Other']
stroke_dataset = stroke_dataset[stroke_dataset.work_type != 'Never_worked']
```

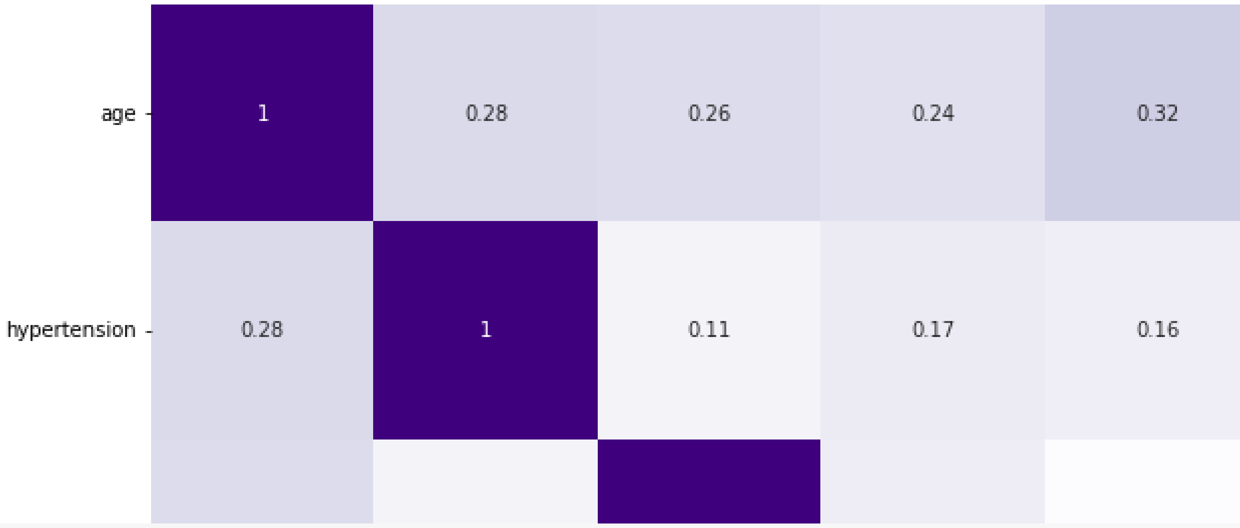
REPLACING NULL VALUES

```
stroke_dataset['bmi'].fillna(stroke_dataset['bmi'].median(), inplace=True)

stroke_dataset.isna().sum()
```

```
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                  0
smoking_status       0
stroke               0
dtype: int64
```

DISPLAYING THE HEAT MAP

```
sns.heatmap(data=stroke_dataset.corr(), annot=True, cmap="Purples")
fig=plt.gcf()
fig.set_size_inches(15,12)
plt.show()
```




```
y_train = stroke_dataset.pop('stroke')
y_train = np.asarray(y_train).astype('float32')
```

```
print("Labels shape:", y_train.shape)
```

```
Labels shape: (5087,)
```

```
# get features
le = LabelEncoder()
ss = StandardScaler()

for col in stroke_dataset.columns:
    if stroke_dataset[col].dtype == 'object':
        stroke_dataset[col] = le.fit_transform(stroke_dataset[col])
    elif stroke_dataset[col].dtype == 'int64':
        stroke_dataset[col] = np.asarray(stroke_dataset[col]).astype('float64')
    else:
        stroke_dataset[col] = ss.fit_transform(stroke_dataset[[col]])
```

```
x_train = np.asarray(stroke_dataset)
print("Features shape:", x_train.shape)
```

```
Features shape: (5087, 10)
```

```
stroke_dataset.drop('Residence_type', axis=1, inplace=True)
```

```
sm = SMOTE()
x_train, y_train = sm.fit_resample(x_train, y_train)
part_x_train, x_test, part_y_train, y_test = train_test_split(x_train, y_train, test_size=

x_bal, y_bal = sm.fit_resample(part_x_train, part_y_train)

x_test, x_val, y_test, y_val = train_test_split(x_test, y_test, test_size=0.5, random_stat
```

BUILDING THE MODEL

```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(x_bal.shape[1],)))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(1, activation='sigmoid'))
early_stopping = EarlyStopping(monitor='val_loss', min_delta=0.01, patience=20, restore_be
model.compile(optimizer=optimizers.Adam(learning_rate=2e-4), loss='binary_crossentropy', m
history = model.fit(x_bal, y_bal, epochs=79, batch_size=128, validation_data=(x_val, y_val
```

```
Epoch 67/79
53/53 [==============================] - 1s 10ms/step - loss: 0.3380 - accuracy:
Epoch 68/79
53/53 [==============================] - 1s 10ms/step - loss: 0.3313 - accuracy:
Epoch 69/79
53/53 [==============================] - 1s 9ms/step - loss: 0.3313 - accuracy: 0
Epoch 70/79
53/53 [==============================] - 1s 11ms/step - loss: 0.3223 - accuracy:
Epoch 71/79
53/53 [==============================] - 1s 11ms/step - loss: 0.3279 - accuracy:
Epoch 72/79
53/53 [==============================] - 1s 10ms/step - loss: 0.3241 - accuracy:
Epoch 73/79
53/53 [==============================] - 1s 10ms/step - loss: 0.3272 - accuracy:
Epoch 74/79
53/53 [==============================] - 1s 10ms/step - loss: 0.3229 - accuracy:
Epoch 75/79
53/53 [==============================] - 1s 10ms/step - loss: 0.3204 - accuracy:
Epoch 76/79
53/53 [==============================] - 1s 10ms/step - loss: 0.3206 - accuracy:
Epoch 77/79
53/53 [==============================] - 1s 10ms/step - loss: 0.3167 - accuracy:
Epoch 78/79
53/53 [==============================] - 1s 10ms/step - loss: 0.3224 - accuracy:
```

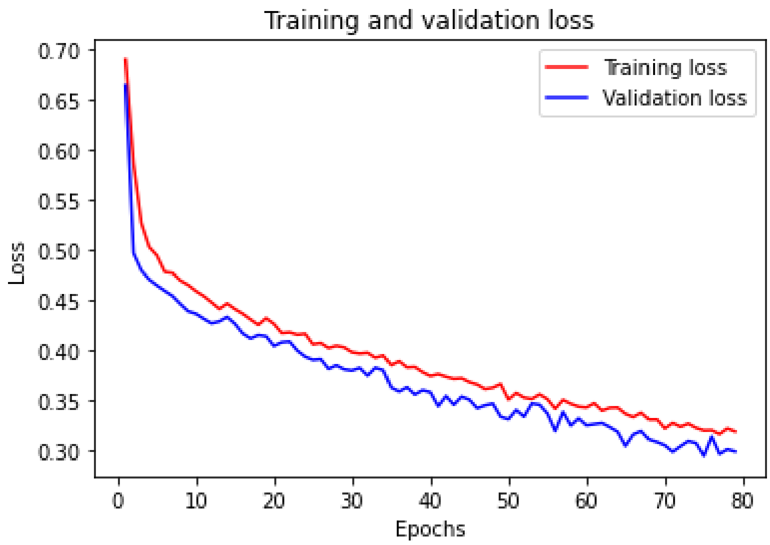
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                704

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 128)               8320

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 256)               33024

 dropout_2 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 256)               65792

 dropout_3 (Dropout)         (None, 256)               0

 dense_4 (Dense)             (None, 128)               32896

 dropout_4 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 128)               16512

 dropout_5 (Dropout)         (None, 128)               0

 dense_6 (Dense)             (None, 128)               16512
```

```
 dropout_6 (Dropout)         (None, 128)               0

 dense_7 (Dense)             (None, 128)               16512

 dropout_7 (Dropout)         (None, 128)               0

 dense_8 (Dense)             (None, 64)                8256

 dropout_8 (Dropout)         (None, 64)                0

 dense_9 (Dense)             (None, 32)                2080

 dropout_9 (Dropout)         (None, 32)                0

 dense_10 (Dense)            (None, 1)                 33

=================================================================
Total params: 200,641
Trainable params: 200,641
Non-trainable params: 0
_________________________________________________________________
```

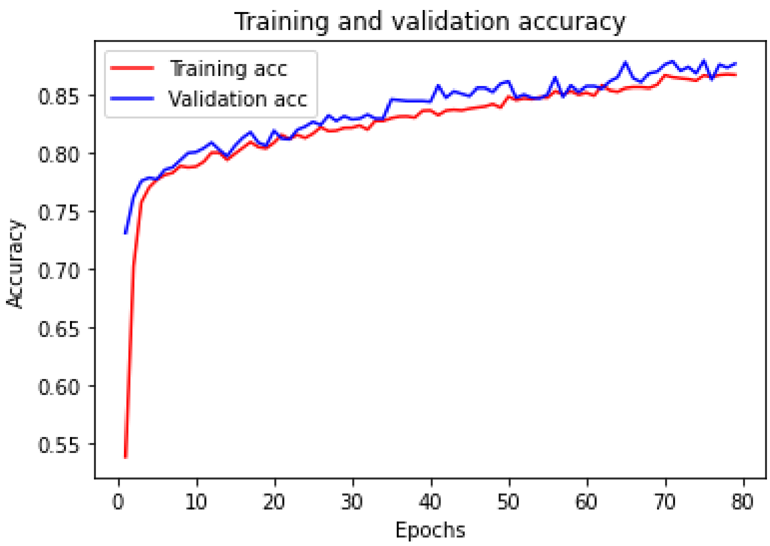
TRAINING AND VALIDATION LOSS

```
history= history.history
loss_values = history['loss']
val_loss_values = history['val_loss']
epochs = range(1, len(history['accuracy']) + 1)
plt.plot(epochs, loss_values, 'r', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



TARINING AND VALIDATION ACCURACY

```
plt.clf()
acc_values = history['accuracy']
val_acc_values = history['val_accuracy']
plt.plot(epochs, history['accuracy'], 'r', label='Training acc')
plt.plot(epochs, history['val_accuracy'], 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




```
y_pred = model.predict(x_test)
y_pred = [1.0 if p > 0.5 else 0 for p in y_pred]
```

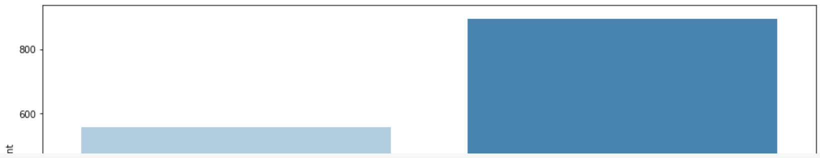
```
46/46 [==============================] - 0s 2ms/step
```

```
y_pred
```

```
[0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 0,
 1.0,
 1.0,
 1.0,
 0,
 1.0,
 0,
 1.0,
 1.0,
 0,
 1.0,
 1.0,
 1.0,
 0,
 1.0,
 0,
```

```
1.0,
1.0,
1.0,
1.0,
0,
1.0,
0,
1.0,
1.0,
1.0,
0,
0,
0,
1.0,
1.0,
0,
1.0,
1.0,
0,
0,
0,
1.0,
0,
0,
0,
0,
0,
0,
1.0,
0,
1.0,
0,
0,
1.0,
1.0,
0,
```

```
plt.figure(figsize=(15,6))
sns.countplot(x=y_pred, palette='Blues')
plt.xticks(rotation = 90)
plt.show()
```




```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.73      0.84       715
         1.0       0.79      0.99      0.88       736

    accuracy                           0.86      1451
   macro avg       0.89      0.86      0.86      1451
weighted avg       0.88      0.86      0.86      1451
```