1. Importing Required Libraries

In [2]:
```
! pip install optuna --quiet
```

In [3]:
```
from warnings import simplefilter
simplefilter("ignore")

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split,StratifiedKFold,GridSearchCV,KFold
from sklearn.preprocessing import StandardScaler,OneHotEncoder
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
import lightgbm as lgbm
from optuna.integration import LightGBMPruningCallback
import optuna
optuna.logging.set_verbosity(optuna.logging.CRITICAL)


from sklearn.metrics import roc_auc_score,f1_score,recall_score,precision_score,precision_recall_curve,roc_curve,classification_report,log_loss


train_src_path = r'/kaggle/input/playground-series-s3e2/train.csv'
test_src_path = r'/kaggle/input/playground-series-s3e2/test.csv'
```

2. Reading data

In [4]:
```
df_train = pd.read_csv(train_src_path)
df_test = pd.read_csv(test_src_path)
```

In [5]:
```
df_train.head()
```

Out[5]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	0	Male	28.0	0	0	Yes	Private	Urban	79.53	31.1	never smoked	0
1	1	Male	33.0	0	0	Yes	Private	Rural	78.44	23.9	formerly smoked	0
2	2	Female	42.0	0	0	Yes	Private	Rural	103.00	40.3	Unknown	0
3	3	Male	56.0	0	0	Yes	Private	Urban	64.87	28.8	never smoked	0
4	4	Female	24.0	0	0	No	Private	Rural	73.36	28.8	never smoked	0

3. EDA

In [6]:
```
df_train.describe()
```

Out[6]:

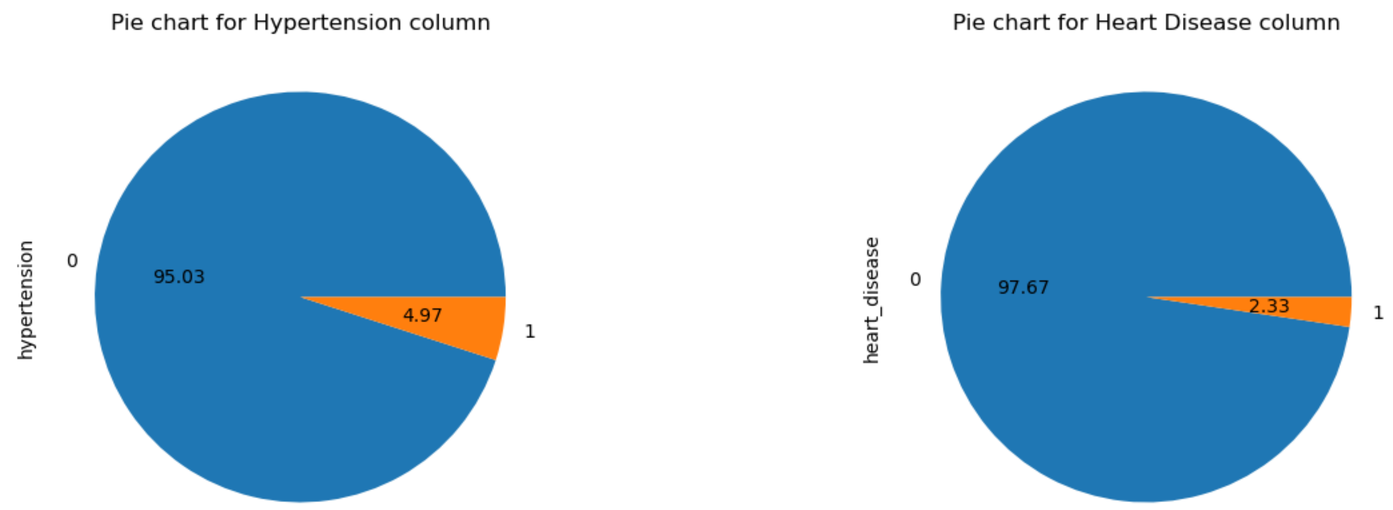
	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	15304.000000	15304.000000	15304.000000	15304.000000	15304.000000	15304.000000	15304.000000
mean	7651.500000	41.417708	0.049726	0.023327	89.039853	28.112721	0.041296
std	4418.028595	21.444673	0.217384	0.150946	25.476102	6.722315	0.198981
min	0.000000	0.080000	0.000000	0.000000	55.220000	10.300000	0.000000
25%	3825.750000	26.000000	0.000000	0.000000	74.900000	23.500000	0.000000
50%	7651.500000	43.000000	0.000000	0.000000	85.120000	27.600000	0.000000
75%	11477.250000	57.000000	0.000000	0.000000	96.980000	32.000000	0.000000
max	15303.000000	82.000000	1.000000	1.000000	267.600000	80.100000	1.000000

In [7]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15304 entries, 0 to 15303
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 15304 non-null  int64  
 1   gender             15304 non-null  object 
 2   age                15304 non-null  float64
 3   hypertension       15304 non-null  int64  
 4   heart_disease      15304 non-null  int64  
 5   ever_married       15304 non-null  object 
 6   work_type          15304 non-null  object 
 7   Residence_type     15304 non-null  object 
 8   avg_glucose_level  15304 non-null  float64
 9   bmi                15304 non-null  float64
 10  smoking_status     15304 non-null  object 
 11  stroke             15304 non-null  int64  
dtypes: float64(3), int64(4), object(5)
memory usage: 1.4+ MB
```

Based on the above output, it is clear that there are no null values in the dataset

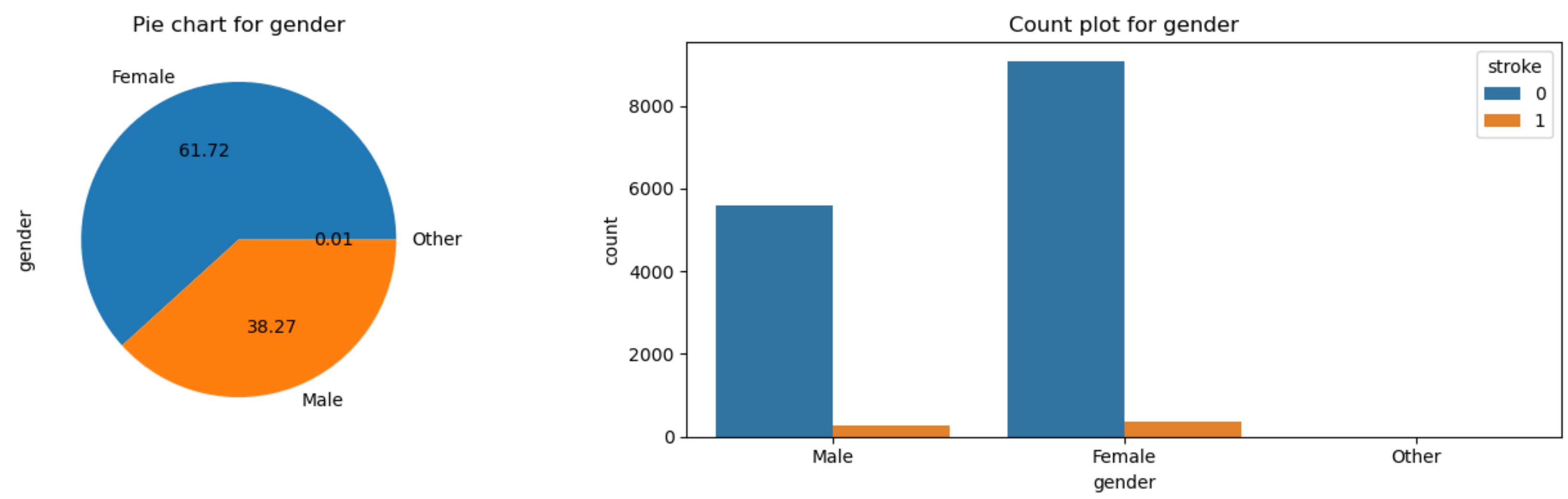
In [8]:
```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
df_train.hypertension.value_counts().plot(kind='pie',autopct='%.2f')
plt.title('Pie chart for Hypertension column')

plt.subplot(1,2,2)
df_train.heart_disease.value_counts().plot(kind='pie',autopct='%.2f')
plt.title('Pie chart for Heart Disease column')

plt.show()
```

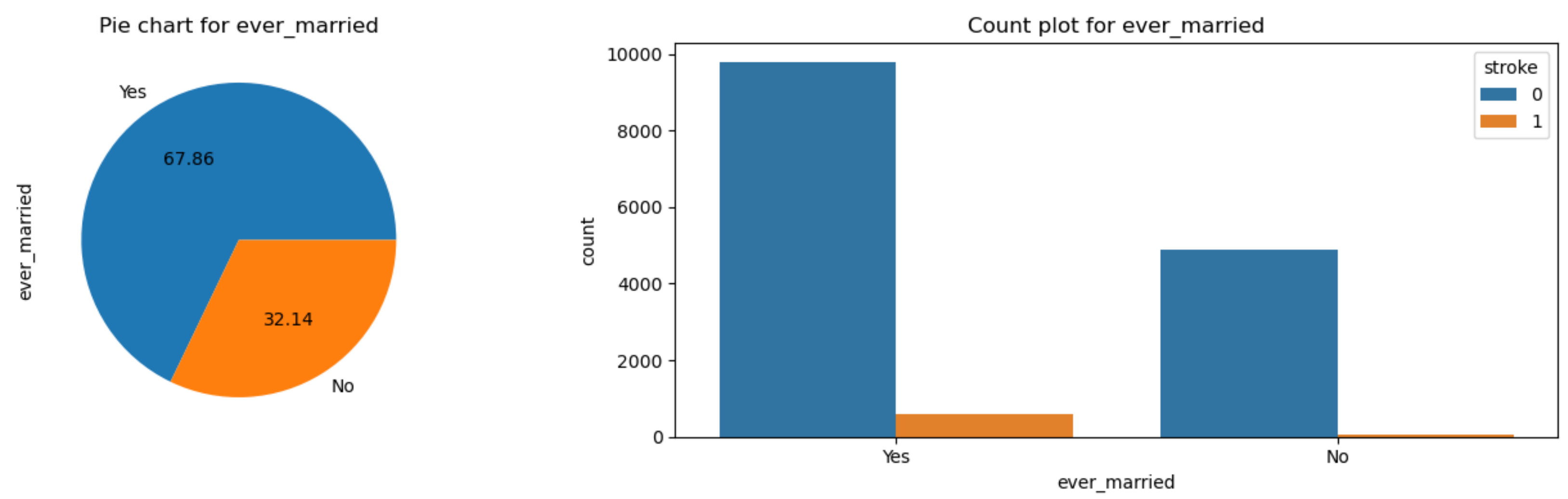


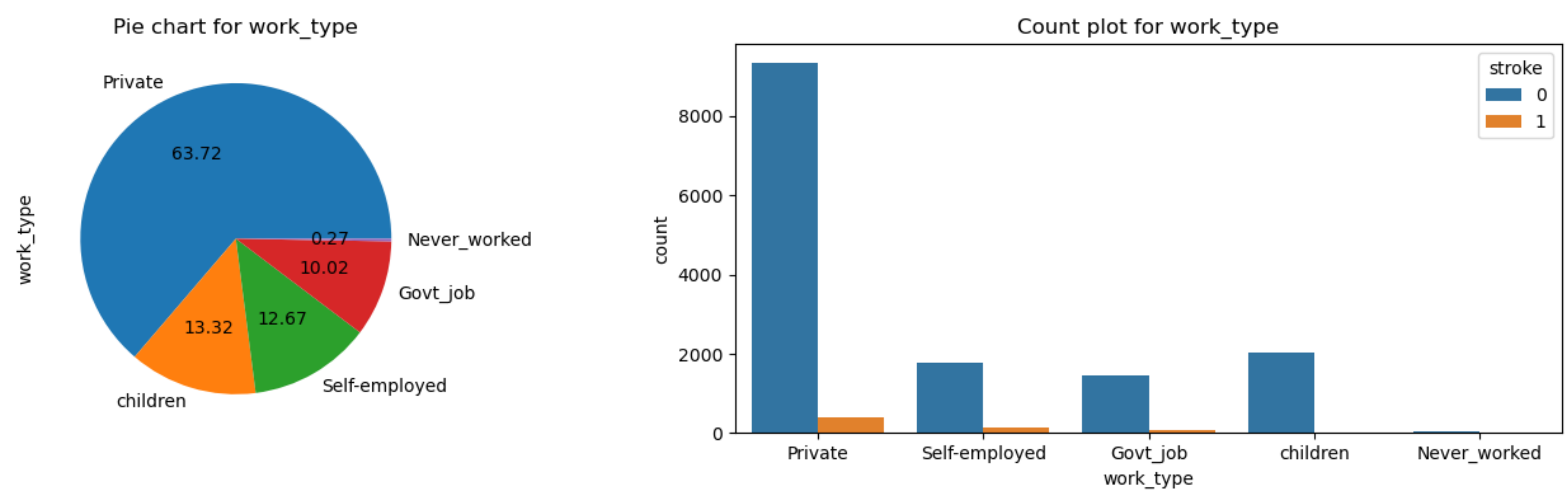
Even though the datatype of columns 'Hypertension' and 'Heart_Disease' is integer type, the information it contains is of categorical type. Hence these columns are considered as Categorical columns

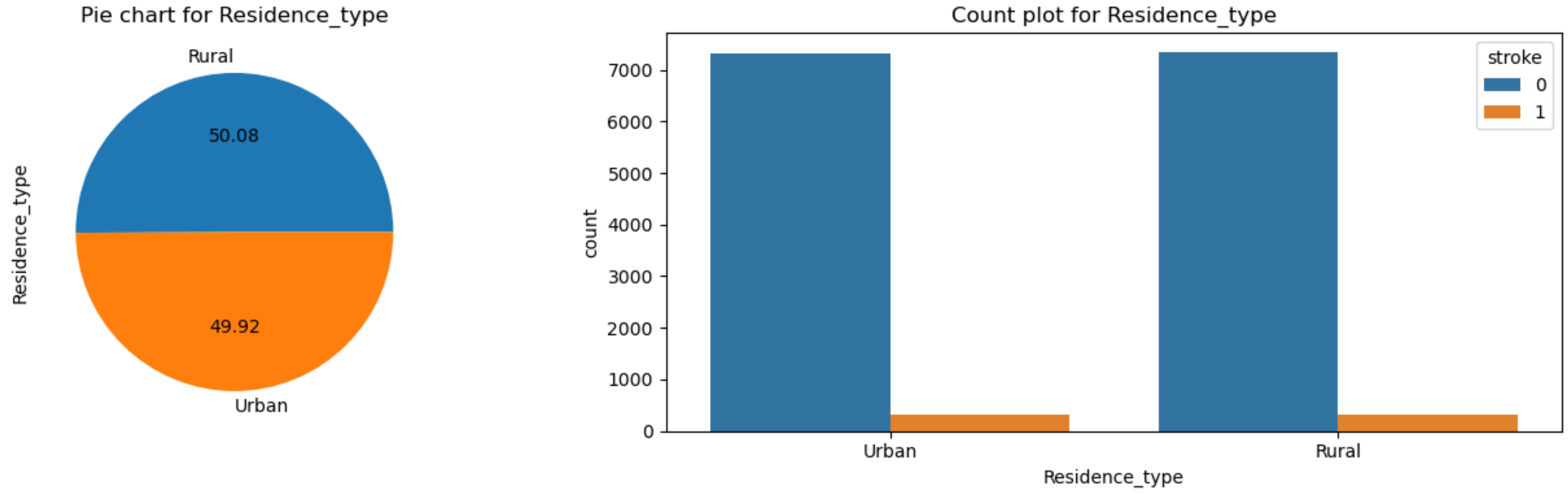
In [9]:
```
X_cols = list(df_train.select_dtypes(include='object').columns)
X_cols.append('hypertension')
X_cols.append('heart_disease')
y_cols = list(df_train.select_dtypes(exclude='object').columns)
y_cols.remove('id')
```

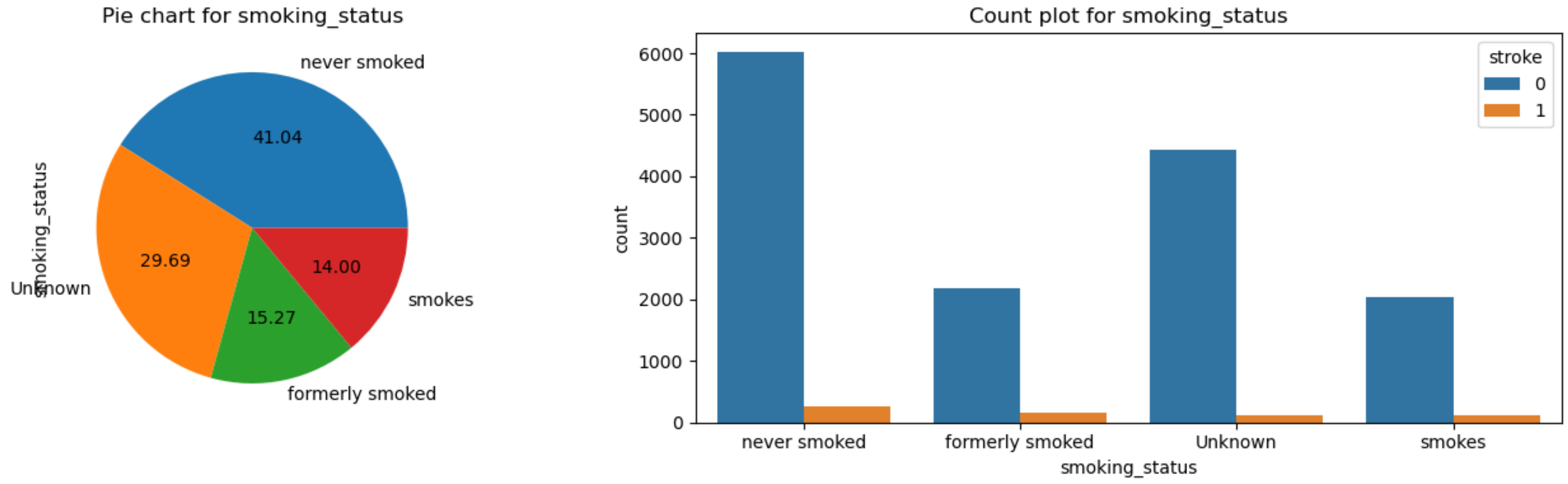
In [10]:
```
for col_name in X_cols:
    plt.figure(figsize=(14,4))
    plt.subplot(1,2,1)
    df_train[col_name].value_counts().plot(kind='pie',autopct='%.2f')
    plt.title(f'Pie chart for {col_name}')

    plt.subplot(1,2,2)
    sns.countplot(x=col_name,data=df_train,hue='stroke')
    plt.title(f'Count plot for {col_name}')
    plt.tight_layout()
    plt.show()
```

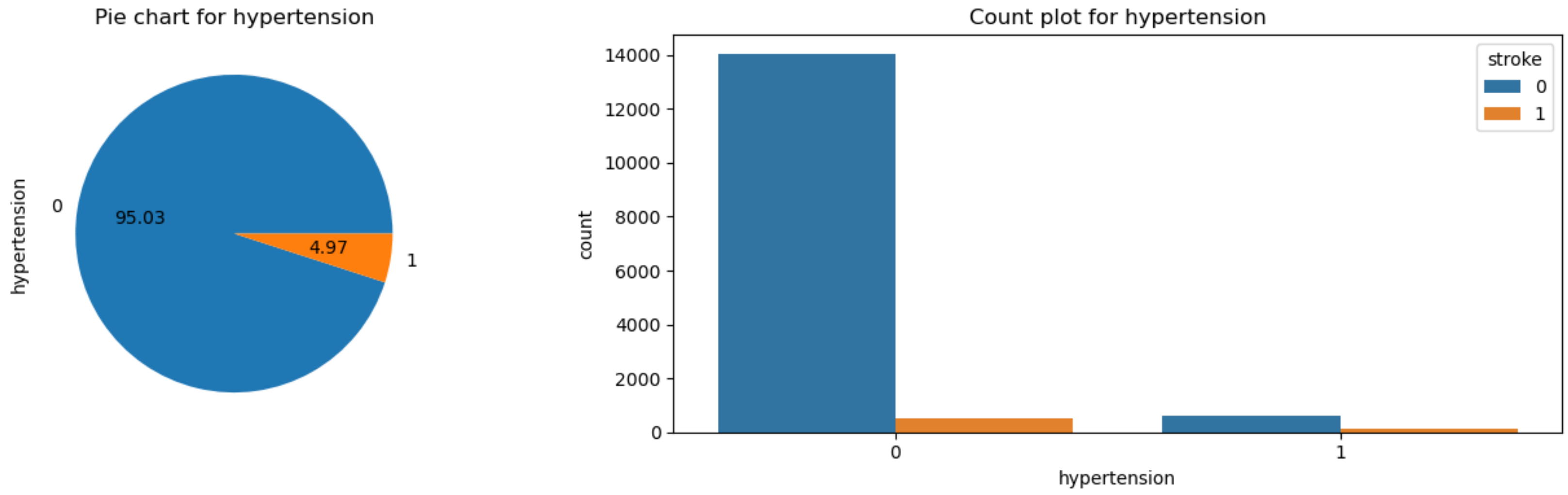


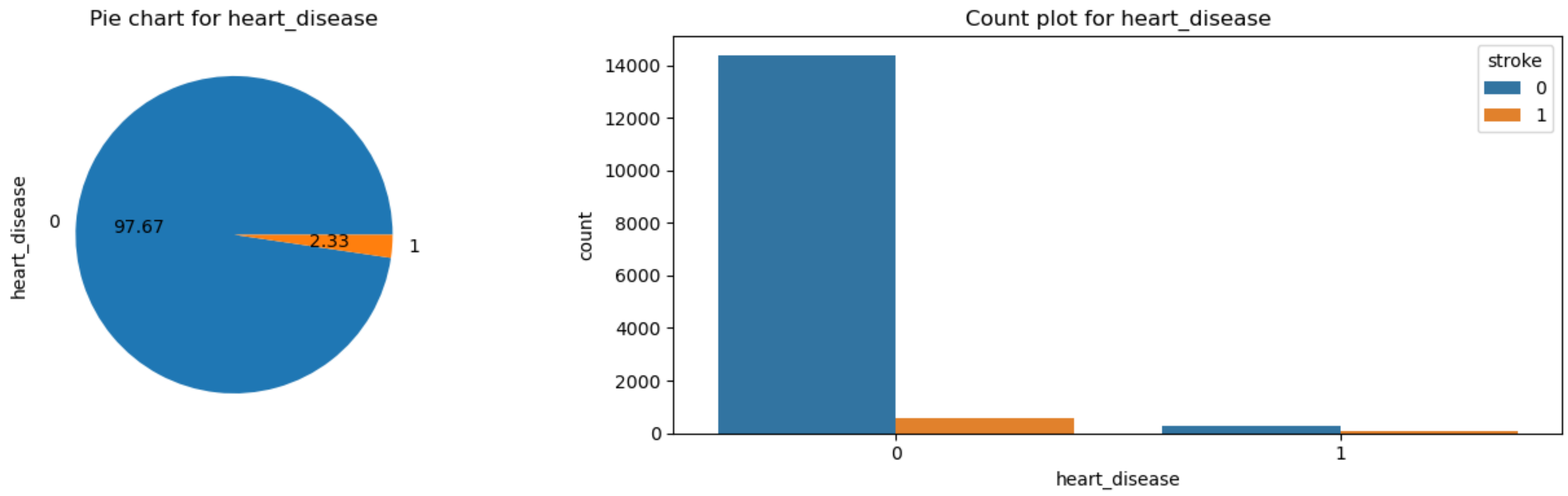






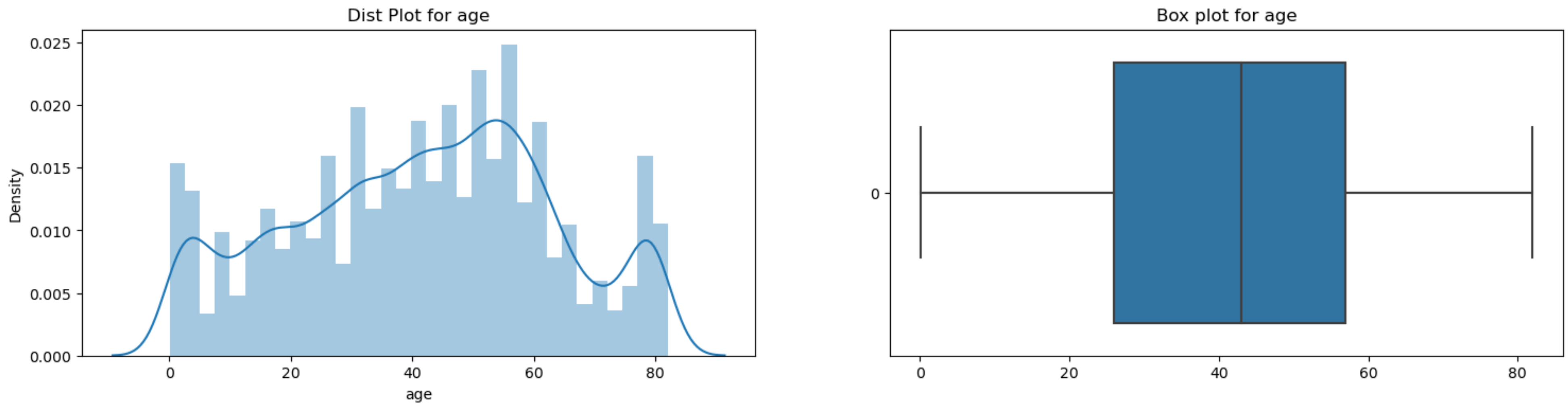


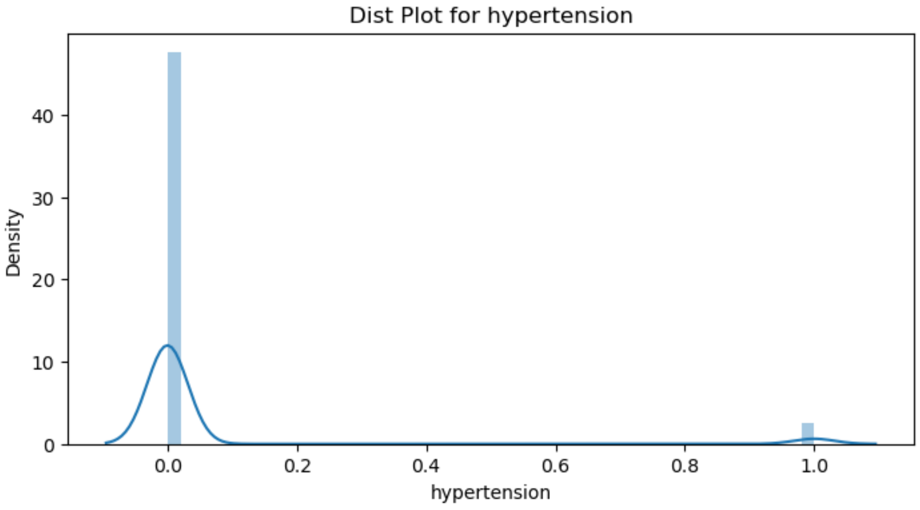


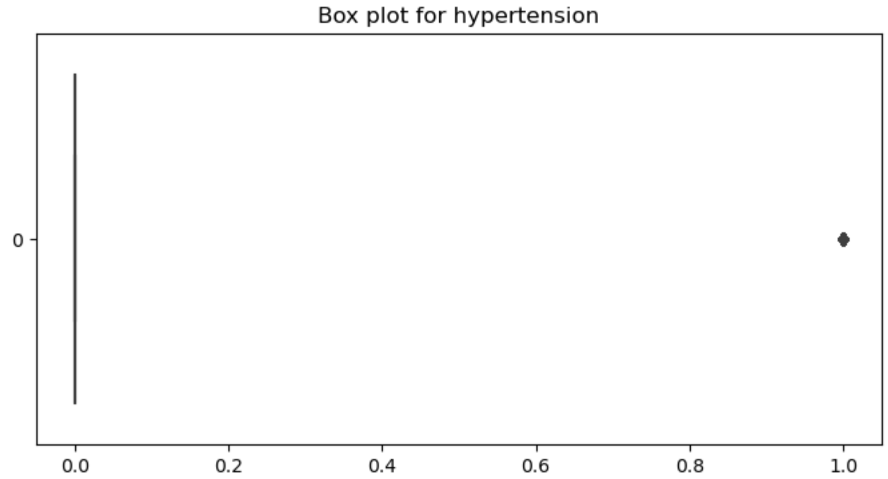


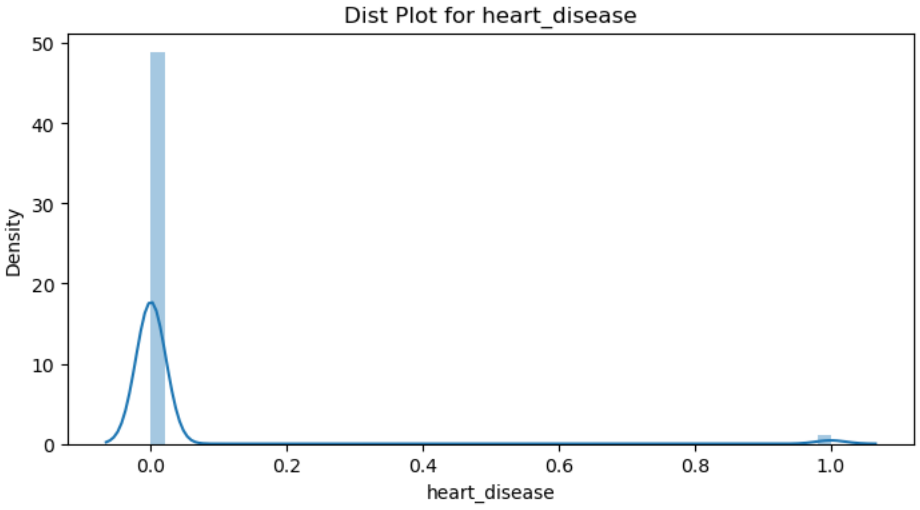
In [11]:

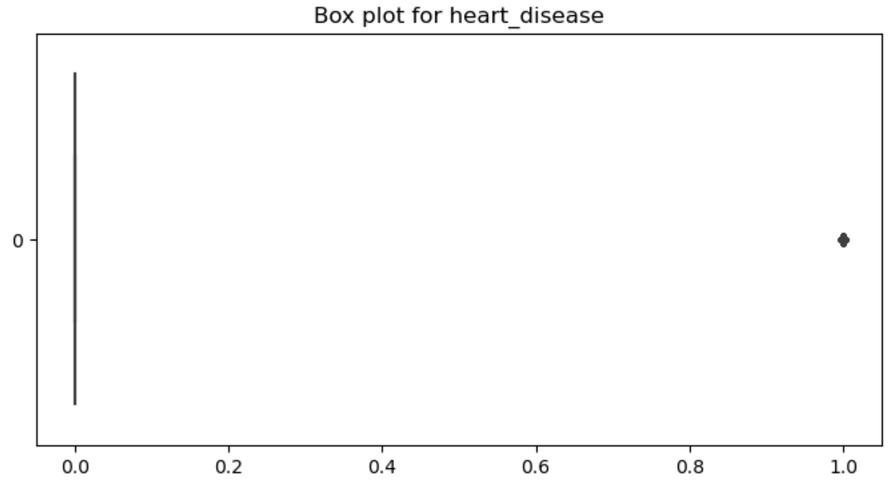
```
for col_name in y_cols:
    plt.figure(figsize=(18,4))
    plt.subplot(1,2,1)
    sns.distplot(df_train[col_name])
    plt.title(f'Dist Plot for {col_name}')

    plt.subplot(1,2,2)
    sns.boxplot(df_train[col_name],whis=1.5,orient='h')
    plt.title(f'Box plot for {col_name}')

    plt.show()
```

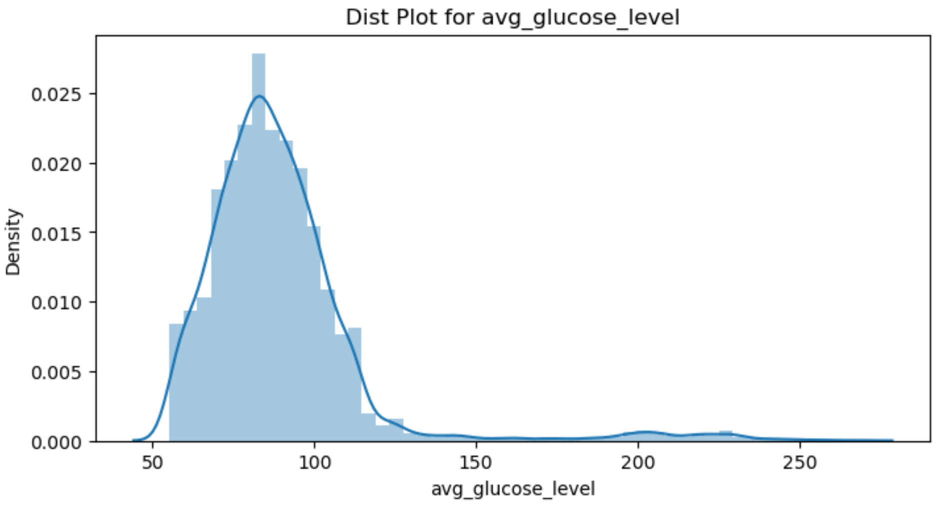


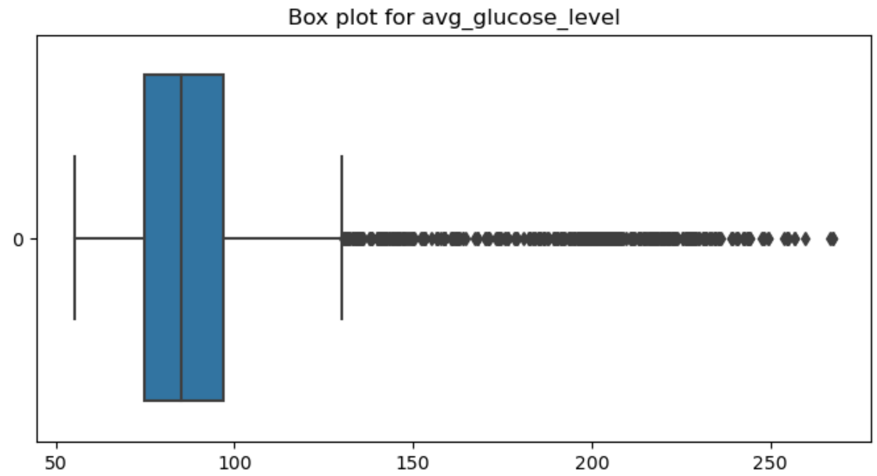


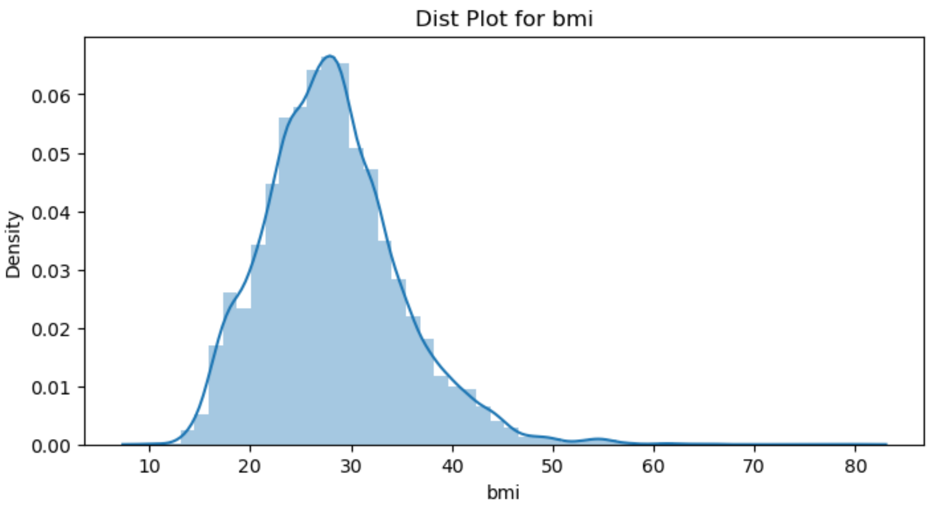


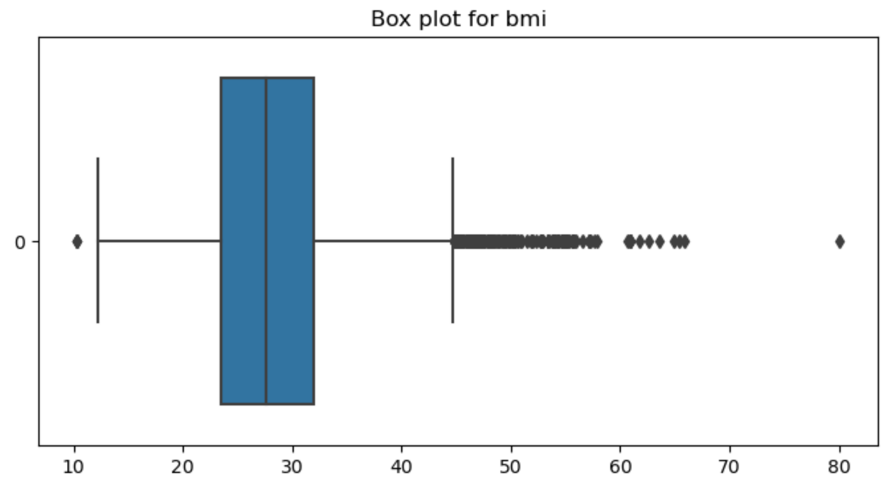


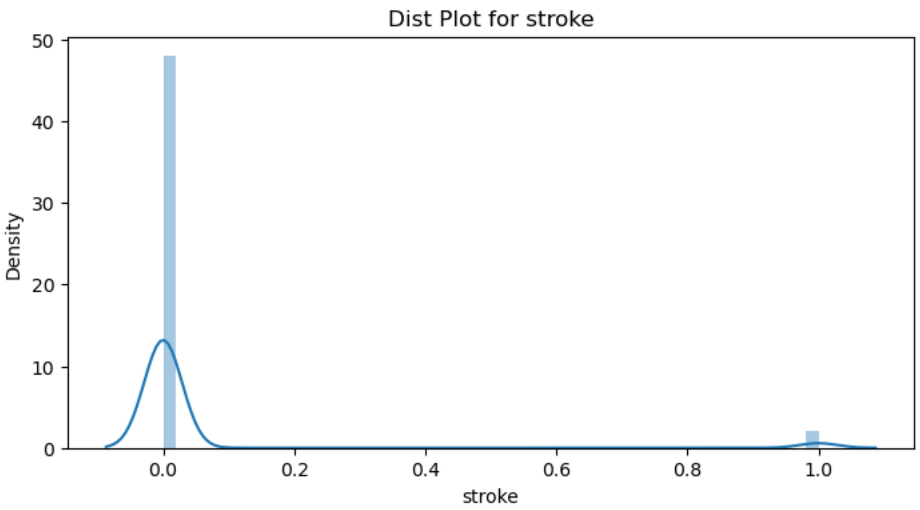


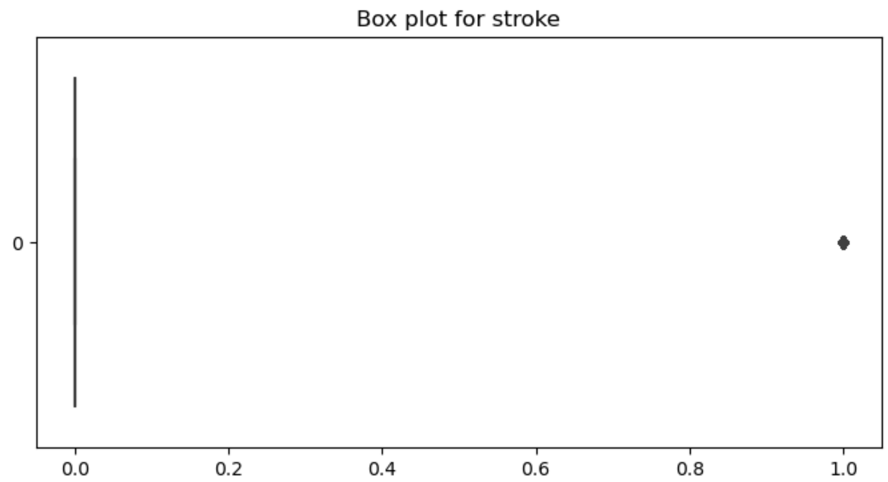










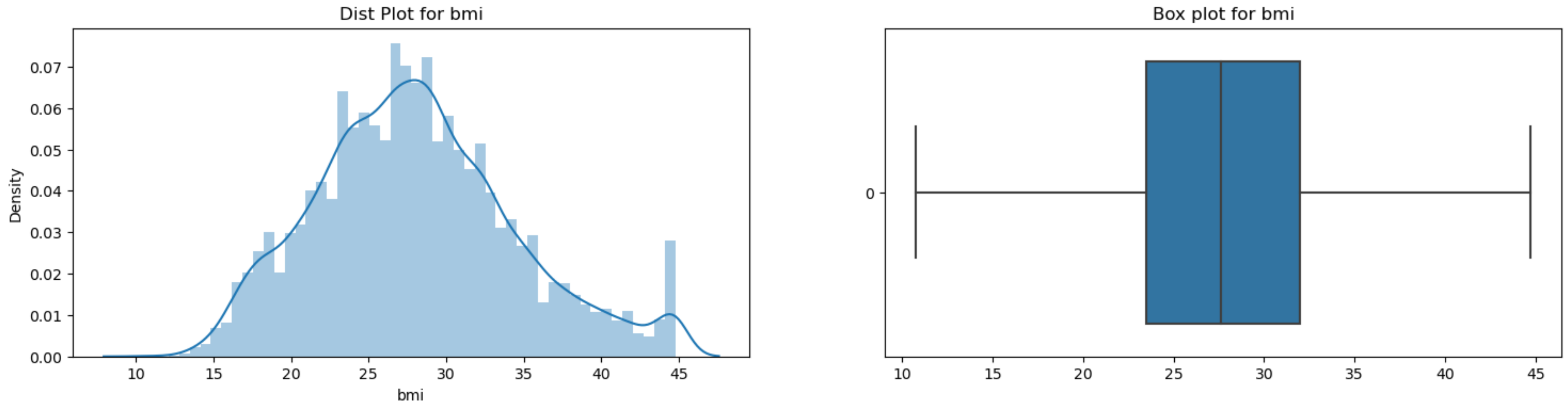


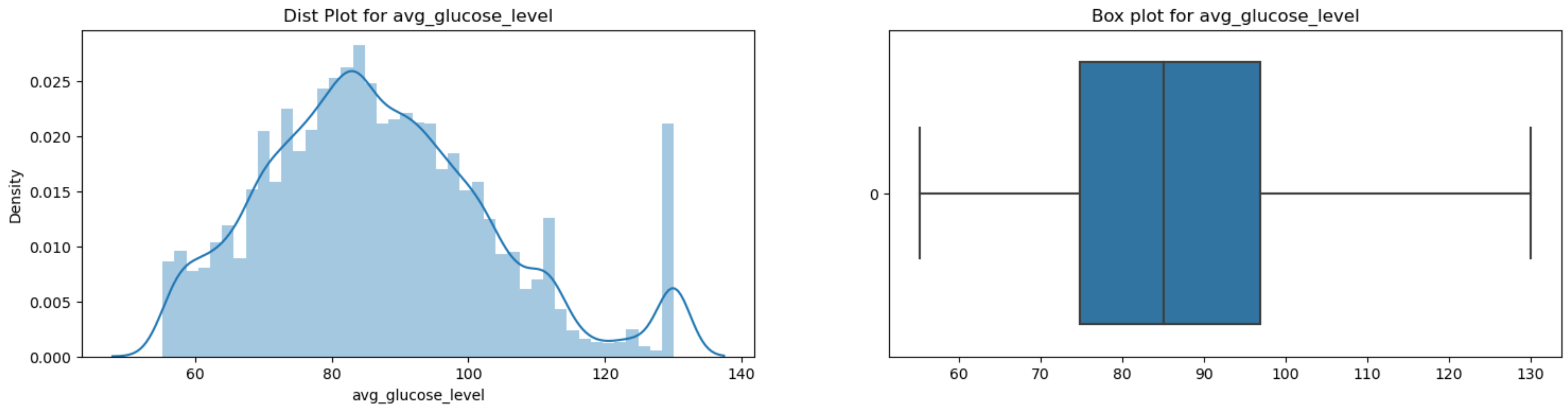
In [12]:

```
# from above plots, we can say there are outliers in column bmi and avg_glucose_level. We will cap those outliers
outlier_col=['bmi','avg_glucose_level']

def func_cap_val(x,col_name,ll,ul):
    if x<ll:
        return ll
    elif x>ul:
        return ul
    else :
        return x
    return 0

for col_name in outlier_col:
    q1=np.percentile(df_train[col_name],25)
    q3=np.percentile(df_train[col_name],75)
    iqr=q3-q1
    lower_limit=q1-1.5*iqr
```

```
    upper_limit=q3+1.5*iqr
    df_train[col_name]=df_train[col_name].apply(lambda x : func_cap_val(x,col_name,lower_limit,upper_limit))
```

In [13]:
```
# to verfiy, we'll replot
for col_name in outlier_col:
    plt.figure(figsize=(18,4))
    plt.subplot(1,2,1)
    sns.distplot(df_train[col_name])
    plt.title(f'Dist Plot for {col_name}')

    plt.subplot(1,2,2)
    sns.boxplot(df_train[col_name],whis=1.5,orient='h')
    plt.title(f'Box plot for {col_name}')

    plt.show()
```





In [14]:
```
# Combining gender type 'others' with 'Male'
df_train['gender_flag']=df_train['gender'].apply(lambda x: 1 if x =='Female' else 0)
df_train.drop(columns='gender',inplace=True)
df_train.head()
```

Out[14]:

	id	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke	gender_flag
0	0	28.0	0	0	Yes	Private	Urban	79.53	31.1	never smoked	0	0
1	1	33.0	0	0	Yes	Private	Rural	78.44	23.9	formerly smoked	0	0
2	2	42.0	0	0	Yes	Private	Rural	103.00	40.3	Unknown	0	1
3	3	56.0	0	0	Yes	Private	Urban	64.87	28.8	never smoked	0	0
4	4	24.0	0	0	No	Private	Rural	73.36	28.8	never smoked	0	1

In [15]:
```
# Encoding the column and removing the original column from the dataset
df_train['married_flag']=df_train['ever_married'].apply(lambda x: 1 if x =='Yes' else 0)
df_train.drop(columns=['ever_married'],inplace=True)
df_train.head()
```

Out[15]:

	id	age	hypertension	heart_disease	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke	gender_flag	married_flag
0	0	28.0	0	0	Private	Urban	79.53	31.1	never smoked	0	0	1
1	1	33.0	0	0	Private	Rural	78.44	23.9	formerly smoked	0	0	1
2	2	42.0	0	0	Private	Rural	103.00	40.3	Unknown	0	1	1
3	3	56.0	0	0	Private	Urban	64.87	28.8	never smoked	0	0	1
4	4	24.0	0	0	Private	Rural	73.36	28.8	never smoked	0	1	0

In [16]:
```
# Encoding the column and removing the original column from the dataset
df_train['residence_type_flag']=df_train['Residence_type'].apply(lambda x: 1 if x =='Urban' else 0)
df_train.drop(columns=['Residence_type'],inplace=True)
df_train.head()
```

Out[16]:

	id	age	hypertension	heart_disease	work_type	avg_glucose_level	bmi	smoking_status	stroke	gender_flag	married_flag	residence_type_flag
0	0	28.0	0	0	Private	79.53	31.1	never smoked	0	0	1	1
1	1	33.0	0	0	Private	78.44	23.9	formerly smoked	0	0	1	0
2	2	42.0	0	0	Private	103.00	40.3	Unknown	0	1	1	0
3	3	56.0	0	0	Private	64.87	28.8	never smoked	0	0	1	1
4	4	24.0	0	0	Private	73.36	28.8	never smoked	0	1	0	0

In [17]:
```
# Using dummy enconding to encode 'work_type' and 'smoking_status' columns, and then dropping the orginal columns from the dataset
cat_cols = ['work_type','smoking_status']
temp = pd.get_dummies(df_train[cat_cols],drop_first=True)
temp.head()
```

Out[17]:

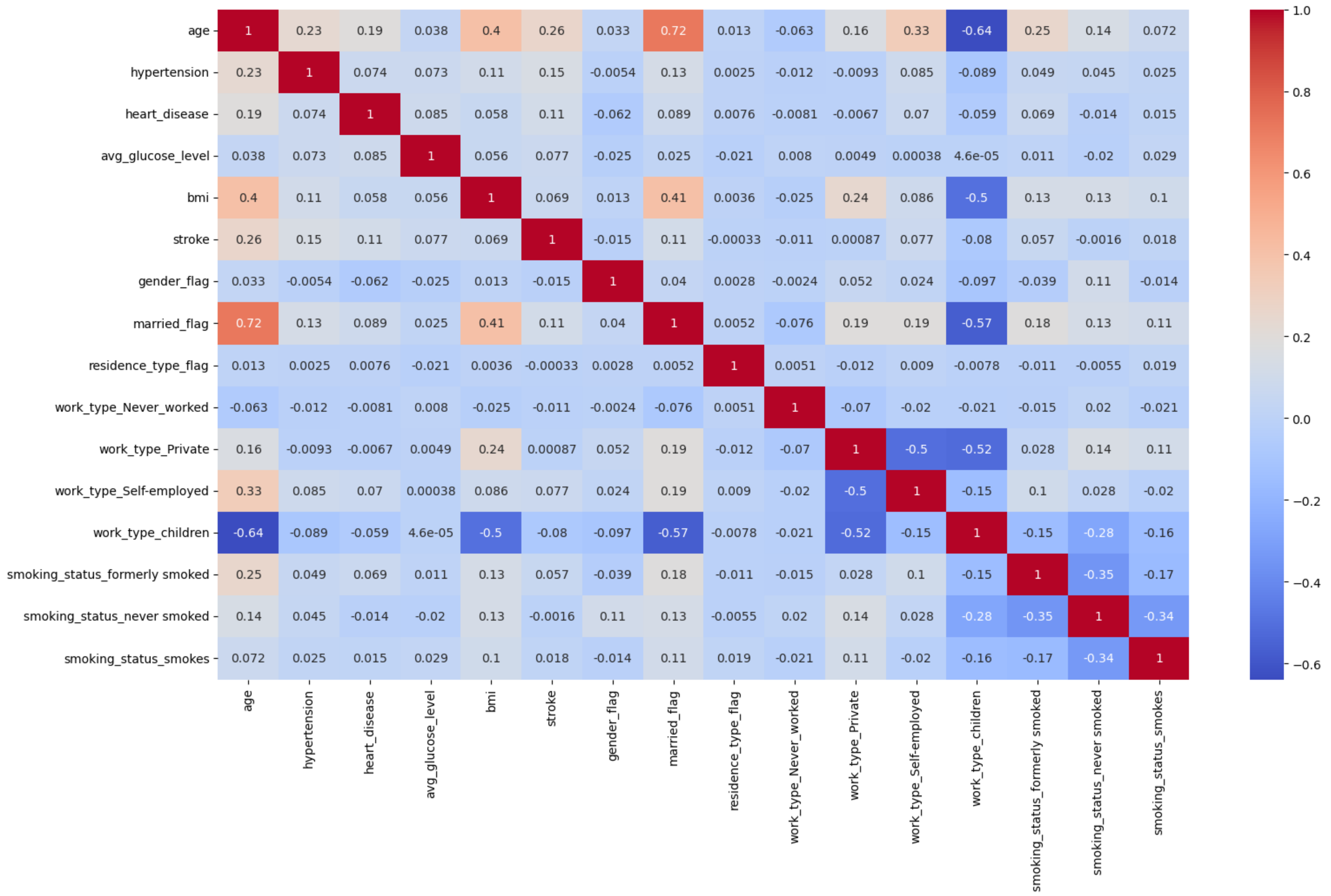
	work_type_Never_worked	work_type_Private	work_type_Self-employed	work_type_children	smoking_status_formerly smoked	smoking_status_never smoked	smoking_status_smokes
0	0	1	0	0	0	1	0
1	0	1	0	0	1	0	0
2	0	1	0	0	0	0	0
3	0	1	0	0	0	1	0
4	0	1	0	0	0	1	0

In [18]:
```python
df_train = pd.concat([df_train,temp],axis=1)
df_train.drop(columns=cat_cols,inplace=True)
df_train.head()
```

Out[18]:

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender_flag	married_flag	residence_type_flag	work_type_Never_worked	work_type_Private	work_type_Self-employed	work_type_children	smoking_status_formerly smoked	smoking_status s
0	0	28.0	0	0	79.53	31.1	0	0	1	1	0	1	0	0	0	
1	1	33.0	0	0	78.44	23.9	0	0	1	0	0	1	0	0	1	
2	2	42.0	0	0	103.00	40.3	0	1	1	0	0	1	0	0	0	
3	3	56.0	0	0	64.87	28.8	0	0	1	1	0	1	0	0	0	
4	4	24.0	0	0	73.36	28.8	0	1	0	0	0	1	0	0	0	

In [19]:
```python
# Plotting heatmap to check the collinearity of independent columns with dependent columns and to check for multi-collinearty
plt.figure(figsize=(18,10))
sns.heatmap(df_train.drop(columns='id').corr(),annot=True,cmap='coolwarm')
plt.show()
```



Based on above heatmap, following are the observations:

1. Among all the independent variables in the dataset, 'Age' seems to be somewhat correlated to dependent variable 'Stroke'
2. Multi-collinearity is not present i.e. dependent variables are not correlated to each other.

Since the target variable is highly imbalanced, SMOTE technique is used to upsample the minority class

In [20]:
```python
X = df_train.drop(columns=['id','stroke'])
y= df_train['stroke'].copy()

X_cols = list(X.columns)


smote = SMOTE(random_state=42)

X,y = smote.fit_resample(X,y)


X_train,X_val,y_train,y_val = train_test_split(X,y,train_size=0.75,random_state=42)


cols_to_norm = ['age','bmi','avg_glucose_level']

scaler_x = StandardScaler(with_mean=False)

X_train[cols_to_norm] = pd.DataFrame(scaler_x.fit_transform(X_train[cols_to_norm]),columns=cols_to_norm)


X_val[cols_to_norm] = pd.DataFrame(scaler_x.transform(X_val[cols_to_norm]),columns=cols_to_norm)
```

In [21]:
```python
from sklearn.impute import KNNImputer

imputer = KNNImputer(missing_values=np.NaN,n_neighbors=3)

X_train = pd.DataFrame(imputer.fit_transform(X_train),columns=X_cols)
X_val = pd.DataFrame(imputer.transform(X_val),columns=X_cols)
```

4. Linear Regression Model (Default Parameters)

In [22]:
```python
# Creating a Logistic Regression Model with default values
lr = LogisticRegression()

lr.fit(X_train,y_train)
```

Out[22]:
```
▾ LogisticRegression
LogisticRegression()
```

In [23]:
```
def get_results(model,in_df,true_df):
    y_probs_pred = model.predict_proba(in_df)[:,1]
    y_pred = model.predict(in_df)

    auc_score = roc_auc_score(true_df,y_probs_pred)
    f1_score_1 = f1_score(true_df,y_pred)
    prcs_score = precision_score(true_df,y_pred)
    recall_score_1 = recall_score(true_df,y_pred)

    print(f'ROC AUC score on input dataset is {auc_score}')
    print('Classification report is shown below: ')
    print(classification_report(true_df,y_pred))
```

In [24]:
```
get_results(lr,X_train,y_train)
```

```
ROC AUC score on input dataset is 0.8819814884180552
Classification report is shown below:
              precision    recall  f1-score   support

           0       0.79      0.81      0.80     10966
           1       0.80      0.78      0.79     11042

    accuracy                           0.79     22008
   macro avg       0.79      0.79      0.79     22008
weighted avg       0.80      0.79      0.79     22008
```

In [25]:
```
get_results(lr,X_val,y_val)
```

```
ROC AUC score on input dataset is 0.8809358734774522
Classification report is shown below:
              precision    recall  f1-score   support

           0       0.79      0.80      0.79      3706
           1       0.79      0.78      0.79      3630

    accuracy                           0.79      7336
   macro avg       0.79      0.79      0.79      7336
weighted avg       0.79      0.79      0.79      7336
```

4.1 Logistic Regression Hyperparameter tuning

In [26]:
```
lr_cv = LogisticRegression()

param= {'C': [0.1,0.25, 0.5,0.75, 1,5,10,15,20,50,100],
        'penalty': ['l1', 'l2']
       }

folds = StratifiedKFold(n_splits = 5, shuffle = True, random_state = 42)

model= GridSearchCV(estimator= lr_cv,
                    param_grid=param,
                    scoring="roc_auc",
                    cv=folds,
                    return_train_score=True,
                    verbose=1)

model.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 22 candidates, totalling 110 fits
```

Out[26]:
```
▸ GridSearchCV
▸ estimator: LogisticRegression
  ▸ LogisticRegression
```

In [27]:
```
score_round_1 = model.best_score_
print(f'Best score obtained after hyperparameter tuning round 1 is {model.best_score_}')
print(f'Best estimatore obtained is {model.best_estimator_}')
print(f'Best parameters obtained for model are {model.best_params_}')
```

```
Best score obtained after hyperparameter tuning round 1 is 0.8816675700866037
Best estimatore obtained is LogisticRegression(C=100)
Best parameters obtained for model are {'C': 100, 'penalty': 'l2'}
```

4.2 Hyperparameter tuning round 2

In [28]:
```
params_round_1 = model.best_params_

c_val = params_round_1.get('C')

param= {'C': [x for x in range(c_val-20,c_val+20)],
        'penalty': ['l1', 'l2']
       }

folds = StratifiedKFold(n_splits = 5, shuffle = True, random_state = 42)

model= GridSearchCV(estimator= lr_cv,
                    param_grid=param,
                    scoring="roc_auc",
                    cv=folds,
                    return_train_score=True,
                    verbose=1)

model.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 80 candidates, totalling 400 fits
```

Out[28]:
```
▸ GridSearchCV
▸ estimator: LogisticRegression
  ▸ LogisticRegression
```

In [29]:
```
print(f'Best score obtained after hyperparameter tuning round 2 is {model.best_score_}')
print(f'Best estimatore obtained is {model.best_estimator_}')
print(f'Best parameters obtained for model are {model.best_params_}')
```

```
Best score obtained after hyperparameter tuning round 2 is 0.8816825570584305
Best estimatore obtained is LogisticRegression(C=110)
Best parameters obtained for model are {'C': 110, 'penalty': 'l2'}
```

In [30]:
```
best_params = model.best_params_

lr_cv = LogisticRegression(**best_params)
lr_cv.fit(X_train,y_train)

get_results(lr_cv,X_train,y_train)
```

```
ROC AUC score on input dataset is 0.882105556675599
Classification report is shown below:
              precision    recall  f1-score   support

           0       0.79      0.81      0.80     10966
           1       0.80      0.78      0.79     11042
```

```
    accuracy                           0.79     22008
   macro avg       0.79      0.79      0.79     22008
weighted avg       0.79      0.79      0.79     22008
```

In [31]:
```
get_results(lr_cv,X_val,y_val)
```

```
ROC AUC score on input dataset is 0.8811183636393372
Classification report is shown below:
              precision    recall  f1-score   support

           0       0.79      0.80      0.79      3706
           1       0.79      0.78      0.79      3630

    accuracy                           0.79      7336
   macro avg       0.79      0.79      0.79      7336
weighted avg       0.79      0.79      0.79      7336
```

4.3 Predictions on test dataset

In [32]:
```
for col_name in outlier_col:
    q1=np.percentile(df_test[col_name],25)
    q3=np.percentile(df_test[col_name],75)
    iqr=q3-q1
    lower_limit=q1-1.5*iqr
    upper_limit=q3+1.5*iqr
    df_test[col_name]=df_test[col_name].apply(lambda x : func_cap_val(x,col_name,lower_limit,upper_limit))
```

In [33]:
```
df_test['gender_flag']=df_test['gender'].apply(lambda x: 1 if x =='Female' else 0)
df_test.drop(columns='gender',inplace=True)

df_test['married_flag']=df_test['ever_married'].apply(lambda x: 1 if x =='Yes' else 0)
df_test.drop(columns=['ever_married'],inplace=True)

df_test['residence_type_flag']=df_test['Residence_type'].apply(lambda x: 1 if x =='Urban' else 0)
df_test.drop(columns=['Residence_type'],inplace=True)
```

In [34]:
```
temp = pd.get_dummies(df_test[cat_cols],drop_first=True)
df_test = pd.concat([df_test,temp],axis=1)
df_test.drop(columns=cat_cols,inplace=True)
```

In [35]:
```
X_test = df_test[X_cols]

X_test[cols_to_norm] = pd.DataFrame(scaler_x.transform(X_test[cols_to_norm]),columns=cols_to_norm)

X_test = pd.DataFrame(imputer.transform(X_test),columns=X_cols)
```

In [36]:
```
y_test_pred = model.predict_proba(X_test)[:,1]
y_test_pred = pd.DataFrame(y_test_pred,columns=['stroke_lr'])

df_test = pd.concat([df_test,y_test_pred],axis=1)
```

5. XGBoost Classifier (Default parameters)

In [37]:
```
X_train[cols_to_norm] = X_train[cols_to_norm].astype('float')
X_val[cols_to_norm] = X_val[cols_to_norm].astype('float')
```

In [38]:
```
xgb = XGBClassifier(objective='binary:logistic',seed=42,tree_method='gpu_hist')

xgb.fit(X_train,y_train)
```

Out[38]:
```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In [39]:
```
get_results(xgb,X_train,y_train)
```

```
ROC AUC score on input dataset is 0.9706185794077976
Classification report is shown below:
              precision    recall  f1-score   support

           0       0.89      0.92      0.91     10966
           1       0.92      0.89      0.91     11042

    accuracy                           0.91     22008
   macro avg       0.91      0.91      0.91     22008
weighted avg       0.91      0.91      0.91     22008
```

In [40]:
```
get_results(xgb,X_val,y_val)
```

```
ROC AUC score on input dataset is 0.9075142461260797
Classification report is shown below:
              precision    recall  f1-score   support

           0       0.95      0.53      0.68      3706
           1       0.67      0.97      0.79      3630

    accuracy                           0.75      7336
   macro avg       0.81      0.75      0.74      7336
weighted avg       0.81      0.75      0.74      7336
```

5.1 Predictions on test dataset

In [41]:
```
y_test_pred = xgb.predict_proba(X_test)[:,1]
y_test_pred = pd.DataFrame(y_test_pred,columns=['stroke_xgb'])

df_test = pd.concat([df_test,y_test_pred],axis=1)
```

5.2 XGBoost Model (Hyperparameter tuning)

In [42]:
```
def objective(trial,X,y):
    param = {
        "booster": trial.suggest_categorical("booster", ["gbtree", "gblinear", "dart"]),
        "lambda": trial.suggest_loguniform("lambda", 1e-8, 1.0),
        "alpha": trial.suggest_loguniform("alpha", 1e-8, 1.0),
    }

    if param["booster"] == "gbtree" or param["booster"] == "dart":
        param["max_depth"] = trial.suggest_int("max_depth", 1, 9)
        param["eta"] = trial.suggest_loguniform("eta", 1e-8, 1.0)
        param["gamma"] = trial.suggest_loguniform("gamma", 1e-8, 1.0)
        param["grow_policy"] = trial.suggest_categorical("grow_policy", ["depthwise", "lossguide"])
```

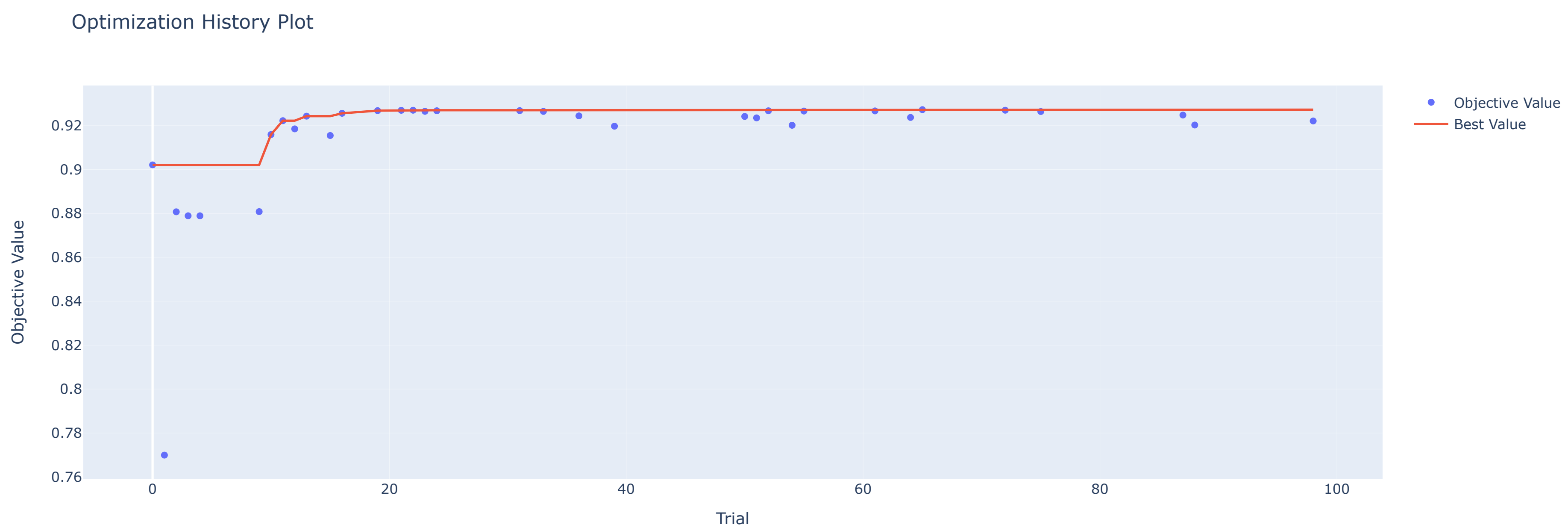
```
    if param["booster"] == "dart":
        param["sample_type"] = trial.suggest_categorical("sample_type", ["uniform", "weighted"])
        param["normalize_type"] = trial.suggest_categorical("normalize_type", ["tree", "forest"])
        param["rate_drop"] = trial.suggest_loguniform("rate_drop", 1e-8, 1.0)
        param["skip_drop"] = trial.suggest_loguniform("skip_drop", 1e-8, 1.0)


    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

    cv_scores = np.empty(5)
    for idx, (train_idx, test_idx) in enumerate(cv.split(X, y)):
        X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
        y_train, y_test = pd.DataFrame(y).iloc[train_idx], pd.DataFrame(y).iloc[test_idx]

        model = XGBClassifier(objective="binary:logistic", **param)
        model.fit(
            X_train,
            y_train,
            eval_set=[(X_test, y_test)],
            eval_metric="auc",
            early_stopping_rounds=100,
            verbose=0,
            callbacks=[optuna.integration.XGBoostPruningCallback(trial, "validation_0-auc")],  # Add a pruning callback
        )
        preds = model.predict_proba(X_test)[:,1]
        cv_scores[idx] = roc_auc_score(y_test, preds)

    return np.mean(cv_scores)
```

In [43]:
```
study = optuna.create_study(direction='maximize', study_name="XGBoost Classifier")
func = lambda trial: objective(trial, X_train, y_train)
study.optimize(func, n_trials=100)
```

In [44]:
```
print(f"\tBest value (rmse): {study.best_value:.5f}")
print(f"\tBest params:")

for key, value in study.best_params.items():
    print(f"\t\t{key}: {value}")
```

```
        Best value (rmse): 0.92726
        Best params:
                booster: gbtree
                lambda: 8.6920997122435e-08
                alpha: 0.0012598097377007158
                max_depth: 8
                eta: 0.14722512825312223
                gamma: 1.4756581071188486e-07
                grow_policy: depthwise
```

In [45]:
```
optuna.visualization.plot_optimization_history(study)
```



In [46]:
```
xgb_final = XGBClassifier(objective="binary:logistic",**study.best_params)

xgb_final.fit(X_train,y_train)
```

Out[46]:
```
                                 XGBClassifier
XGBClassifier(alpha=0.0012598097377007158, base_score=None, booster='gbtree',
              callbacks=None, colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eta=0.14722512825312223,
              eval_metric=None, feature_types=None,
              gamma=1.4756581071188486e-07, gpu_id=None,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints=None, lambda=8.6920997122435e-08,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=8,
              max_leaves=None, min_child_weight=None, missing=nan,
```

In [47]:
```
get_results(xgb_final,X_train,y_train)
```

```
ROC AUC score on input dataset is 0.9793965841232998
Classification report is shown below:
              precision    recall  f1-score   support

           0       0.91      0.93      0.92     10966
           1       0.93      0.91      0.92     11042

    accuracy                           0.92     22008
   macro avg       0.92      0.92      0.92     22008
weighted avg       0.92      0.92      0.92     22008
```

In [48]:
```
get_results(xgb_final,X_val,y_val)
```

```
ROC AUC score on input dataset is 0.9057456897384778
Classification report is shown below:
              precision    recall  f1-score   support

           0       0.95      0.52      0.67      3706
           1       0.67      0.97      0.79      3630

    accuracy                           0.75      7336
   macro avg       0.81      0.75      0.73      7336
weighted avg       0.81      0.75      0.73      7336
```

5.3 Predictions on test dataset

In [49]:
```
y_test_pred = xgb_final.predict_proba(X_test)[:,1]
y_test_pred = pd.DataFrame(y_test_pred,columns=['stroke_xgb_cv'])

df_test = pd.concat([df_test,y_test_pred],axis=1)
```

6. CatBoost Model

In [50]:
```
train_cb = pd.read_csv(train_src_path)
test_cb = pd.read_csv(test_src_path)
```

In [51]:
```
train_cb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15304 entries, 0 to 15303
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 15304 non-null  int64
 1   gender             15304 non-null  object
 2   age                15304 non-null  float64
 3   hypertension       15304 non-null  int64
 4   heart_disease      15304 non-null  int64
 5   ever_married       15304 non-null  object
 6   work_type          15304 non-null  object
 7   Residence_type     15304 non-null  object
 8   avg_glucose_level  15304 non-null  float64
 9   bmi                15304 non-null  float64
 10  smoking_status     15304 non-null  object
 11  stroke             15304 non-null  int64
dtypes: float64(3), int64(4), object(5)
memory usage: 1.4+ MB
```

In [52]:
```
cat_cols_cb = list(train_cb.select_dtypes(include='object').columns)
cat_cols_cb.append('hypertension')
cat_cols_cb.append('heart_disease')


X_cb = train_cb.drop(columns=['id','stroke'])
y_cb = train_cb['stroke']


X_train_cb,X_val_cb,y_train_cb,y_val_cb = train_test_split(X_cb,y_cb,train_size=0.75,random_state=42,stratify=y_cb)
```

In [53]:
```
from catboost import CatBoostClassifier

cb = CatBoostClassifier(random_state=42,class_weights={0:0.1,1:0.9},eval_metric='AUC',verbose=0)

cb.fit(X_train_cb,y_train_cb,cat_features=cat_cols_cb,)
```

Out[53]:
```
<catboost.core.CatBoostClassifier at 0x7f3ea69039a0>
```

In [54]:
```
get_results(cb,X_train_cb,y_train_cb)
```

```
ROC AUC score on input dataset is 0.9881266804399474
Classification report is shown below:
              precision    recall  f1-score   support

           0       1.00      0.97      0.98     11004
           1       0.53      0.90      0.67       474

    accuracy                           0.96     11478
   macro avg       0.76      0.93      0.82     11478
weighted avg       0.98      0.96      0.97     11478
```

In [55]:
```
get_results(cb,X_val_cb,y_val_cb)
```

```
ROC AUC score on input dataset is 0.8739267424043732
Classification report is shown below:
              precision    recall  f1-score   support

           0       0.98      0.94      0.96      3668
           1       0.27      0.47      0.34       158

    accuracy                           0.93      3826
   macro avg       0.62      0.71      0.65      3826
weighted avg       0.95      0.93      0.93      3826
```

6.1 Predictions on test dataset

In [56]:
```
X_test_cb = test_cb.drop(columns=['id'])

y_test_pred = cb.predict_proba(X_test_cb)[:,1]
y_test_pred = pd.DataFrame(y_test_pred,columns=['stroke_cb'])

df_test = pd.concat([df_test,y_test_pred],axis=1)
```

6.2 Hyperparameter tuning for CatBoost model

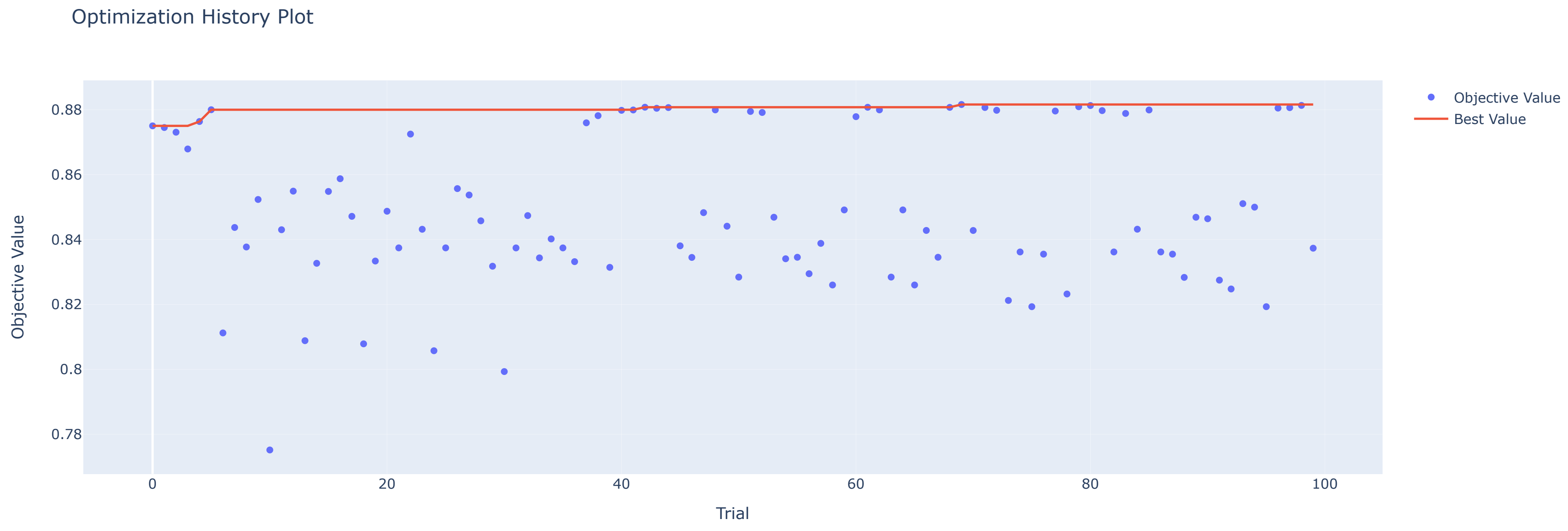
In [57]:
```
def objective(trial, X, y,cat_col_list=cat_cols_cb):
    param_grid = {
        "objective": trial.suggest_categorical("objective", ["Logloss"]),
#         "colsample_bylevel": trial.suggest_float("colsample_bylevel", 0.01, 0.1),
        "depth": trial.suggest_int("depth", 3, 15),
        "boosting_type": trial.suggest_categorical("boosting_type", ["Ordered", "Plain"]),
#         "bootstrap_type": trial.suggest_categorical("bootstrap_type", ["Bayesian", "Bernoulli", "MVS","Poisson"]),
        "bootstrap_type": trial.suggest_categorical("bootstrap_type", ["Bayesian", "Bernoulli", "MVS"]),
        "iterations" : trial.suggest_int("iterations",5000,35000,step=200),
        'l2_leaf_reg': trial.suggest_loguniform('l2_leaf_reg', 1e-3, 10.0),
#         'max_bin': trial.suggest_int('max_bin', 200, 400),
        'learning_rate': trial.suggest_uniform('learning_rate', 0.006, 0.9),
#         'min_data_in_leaf': trial.suggest_int('min_data_in_leaf', 1, 300),
        "scale_pos_weight" : trial.suggest_int("scale_pos_weight",2,50)
    }


    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

    cv_scores = np.empty(5)
    for idx, (train_idx, test_idx) in enumerate(cv.split(X, y)):
        X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
        y_train, y_test = pd.DataFrame(y).iloc[train_idx], pd.DataFrame(y).iloc[test_idx]

        model = CatBoostClassifier(**param_grid,eval_metric='AUC')
        model.fit(
            X_train,
            y_train,
            cat_features=cat_col_list,
            eval_set=[(X_test, y_test)],
#             eval_metric="auc",
            early_stopping_rounds=100,
            callbacks=[optuna.integration.CatBoostPruningCallback(trial, "AUC")],  # Add a pruning callback
            verbose=0
        )
        preds = model.predict_proba(X_test)[:,1]
        cv_scores[idx] = roc_auc_score(y_test, preds)

    return np.mean(cv_scores)
```

In [58]:
```
study = optuna.create_study(direction='maximize', study_name="CatBoost Classifier")
func = lambda trial: objective(trial, X_train_cb, y_train_cb)
study.optimize(func, n_trials=100)
```

In [59]:
```
print(f"\tBest value (auc): {study.best_value:.5f}")
print(f"\tBest params:")

for key, value in study.best_params.items():
    print(f"\t\t{key}: {value}")
```

```
	Best value (auc): 0.88163
	Best params:
		objective: Logloss
		depth: 3
		boosting_type: Plain
		bootstrap_type: Bayesian
		iterations: 16800
		l2_leaf_reg: 0.0017801077176422973
		learning_rate: 0.06672423574808638
		scale_pos_weight: 13
```

In [60]:
```
optuna.visualization.plot_optimization_history(study)
```



In [61]:
```
cb_final = CatBoostClassifier(**study.best_params,eval_metric='AUC',verbose=0)

cb_final.fit(X_train_cb,y_train_cb,cat_features=cat_cols_cb)
```

Out[61]: <catboost.core.CatBoostClassifier at 0x7f3ea98cfd90>

In [62]:
```
get_results(cb_final,X_train_cb,y_train_cb)
```

```
ROC AUC score on input dataset is 0.9990210694385011
Classification report is shown below:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     11004
           1       0.91      0.99      0.95       474

    accuracy                           1.00     11478
   macro avg       0.95      0.99      0.97     11478
weighted avg       1.00      1.00      1.00     11478
```

In [63]:
```
get_results(cb_final,X_val_cb,y_val_cb)
```

```
ROC AUC score on input dataset is 0.8510087241003271
Classification report is shown below:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97      3668
           1       0.26      0.28      0.27       158

    accuracy                           0.94      3826
   macro avg       0.62      0.62      0.62      3826
weighted avg       0.94      0.94      0.94      3826
```

6.3 Predictions on Test Dataset

In [64]:
```
y_test_pred = cb_final.predict_proba(X_test_cb)[:,1]
y_test_pred = pd.DataFrame(y_test_pred,columns=['stroke_cb_cv'])

df_test = pd.concat([df_test,y_test_pred],axis=1)
```

7. LightGBM Model

Without encoding categorical data and without scaling the data

In [65]:
```
X_cb[cat_cols_cb] = X_cb[cat_cols_cb].astype('category')
```

In [66]:
```
X_train_lgb,X_val_lgb,y_train_lgb,y_val_lgb = train_test_split(X_cb,y_cb,train_size=0.75,stratify=y_cb)
```

In [67]:
```
lgb_cat = lgbm.LGBMClassifier(random_state=42,class_weight={0:0.1,1:0.9})

lgb_cat.fit(X_train_lgb,y_train_lgb)
```

Out[67]:
```
LGBMClassifier
LGBMClassifier(class_weight={0: 0.1, 1: 0.9}, random_state=42)
```

In [68]:
```
get_results(lgb_cat,X_train_lgb,y_train_lgb)
```

```
ROC AUC score on input dataset is 0.9914798914702287
Classification report is shown below:
              precision    recall  f1-score   support

           0       1.00      0.95      0.98     11004
           1       0.48      0.96      0.64       474

    accuracy                           0.95     11478
   macro avg       0.74      0.96      0.81     11478
weighted avg       0.98      0.95      0.96     11478
```

In [69]:
```
get_results(lgb_cat,X_val_lgb,y_val_lgb)
```

```
ROC AUC score on input dataset is 0.878413890921138
Classification report is shown below:
              precision    recall  f1-score   support

           0       0.97      0.94      0.96      3668
           1       0.24      0.42      0.30       158

    accuracy                           0.92      3826
   macro avg       0.61      0.68      0.63      3826
weighted avg       0.94      0.92      0.93      3826
```

7.1 LightGBM Model Hyperparameter tuning

In [70]:
```
X = df_train.drop(columns=['id','stroke'])
y= df_train['stroke'].copy()

X_cols = list(X.columns)

smote = SMOTE(random_state=42)

X,y = smote.fit_resample(X,y)


X_train_lgb,X_val_lgb,y_train_lgb,y_val_lgb = train_test_split(X,y,train_size=0.75)
```

In [71]:
```
def objective(trial, X, y):
    param_grid = {
        "boosting_type" : trial.suggest_categorical("boosting_type",['gbdt','dart']),
        "n_estimators": trial.suggest_int("n_estimators", 100,2500,step=50),
        "learning_rate": trial.suggest_uniform("learning_rate", 0.01, 0.3),
        "num_leaves": trial.suggest_int("num_leaves", 25, 5000, step=10),
        "max_depth": trial.suggest_int("max_depth", 3, 15),
        "min_data_in_leaf": trial.suggest_int("min_data_in_leaf", 200, 10000, step=50),
        "lambda_l1": trial.suggest_float("lambda_l1", 1e-8, 10.0, log=True),
        "lambda_l2": trial.suggest_float("lambda_l2", 1e-8, 10.0, log=True),
        "bagging_fraction": trial.suggest_float("bagging_fraction", 0.2, 1.0, step=0.1),
#         "bagging_freq": trial.suggest_int("bagging_freq", 1, 7),
        "feature_fraction": trial.suggest_float("feature_fraction", 0.2, 1.0, step=0.05),
#         "scale_pos_weight" : trial.suggest_float("scale_pos_weight",2,40,step=0.1)
    }

    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

    cv_scores = np.empty(5)
    for idx, (train_idx, test_idx) in enumerate(cv.split(X, y)):
        X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
        y_train, y_test = pd.DataFrame(y).iloc[train_idx], pd.DataFrame(y).iloc[test_idx]

        model = lgbm.LGBMClassifier(objective="binary",verbose=-1, **param_grid)
        model.fit(
            X_train,
            y_train,
            eval_set=[(X_test, y_test)],
            eval_metric="auc",
            early_stopping_rounds=100,
            callbacks=[LightGBMPruningCallback(trial, "auc")],  # Add a pruning callback
#             verbose=-1
        )
        preds = model.predict_proba(X_test)[:,1]
        cv_scores[idx] = roc_auc_score(y_test, preds)


    return np.mean(cv_scores)
```

In [72]:
```
study = optuna.create_study(direction='maximize', study_name="LGBM Classifier Using Non-scaled Data")
func = lambda trial: objective(trial, X_train_lgb,y_train_lgb )
study.optimize(func, n_trials=100)
```

```
[LightGBM] [Warning] bagging_fraction is set=0.6000000000000001, subsample=1.0 will be ignored. Current value: bagging_fraction=0.6000000000000001
[LightGBM] [Warning] lambda_l1 is set=1.7786611790839954e-07, reg_alpha=0.0 will be ignored. Current value: lambda_l1=1.7786611790839954e-07
[LightGBM] [Warning] min_data_in_leaf is set=9750, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=9750
[LightGBM] [Warning] feature_fraction is set=0.30000000000000004, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.30000000000000004
[LightGBM] [Warning] lambda_l2 is set=3.788831090842719, reg_lambda=0.0 will be ignored. Current value: lambda_l2=3.788831090842719
[LightGBM] [Warning] bagging_fraction is set=0.6000000000000001, subsample=1.0 will be ignored. Current value: bagging_fraction=0.6000000000000001
[LightGBM] [Warning] lambda_l1 is set=1.7786611790839954e-07, reg_alpha=0.0 will be ignored. Current value: lambda_l1=1.7786611790839954e-07
[LightGBM] [Warning] min_data_in_leaf is set=9750, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=9750
[LightGBM] [Warning] feature_fraction is set=0.30000000000000004, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.30000000000000004
[LightGBM] [Warning] lambda_l2 is set=3.788831090842719, reg_lambda=0.0 will be ignored. Current value: lambda_l2=3.788831090842719
[LightGBM] [Warning] bagging_fraction is set=0.6000000000000001, subsample=1.0 will be ignored. Current value: bagging_fraction=0.6000000000000001
[LightGBM] [Warning] lambda_l1 is set=1.7786611790839954e-07, reg_alpha=0.0 will be ignored. Current value: lambda_l1=1.7786611790839954e-07
[LightGBM] [Warning] min_data_in_leaf is set=9750, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=9750
[LightGBM] [Warning] feature_fraction is set=0.30000000000000004, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.30000000000000004
[LightGBM] [Warning] lambda_l2 is set=3.788831090842719, reg_lambda=0.0 will be ignored. Current value: lambda_l2=3.788831090842719
[LightGBM] [Warning] bagging_fraction is set=0.6000000000000001, subsample=1.0 will be ignored. Current value: bagging_fraction=0.6000000000000001
[LightGBM] [Warning] lambda_l1 is set=1.7786611790839954e-07, reg_alpha=0.0 will be ignored. Current value: lambda_l1=1.7786611790839954e-07
[LightGBM] [Warning] min_data_in_leaf is set=9750, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=9750
[LightGBM] [Warning] feature_fraction is set=0.30000000000000004, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.30000000000000004
[LightGBM] [Warning] lambda_l2 is set=3.788831090842719, reg_lambda=0.0 will be ignored. Current value: lambda_l2=3.788831090842719
[LightGBM] [Warning] bagging_fraction is set=0.6000000000000001, subsample=1.0 will be ignored. Current value: bagging_fraction=0.6000000000000001
[LightGBM] [Warning] lambda_l1 is set=1.7786611790839954e-07, reg_alpha=0.0 will be ignored. Current value: lambda_l1=1.7786611790839954e-07
[LightGBM] [Warning] min_data_in_leaf is set=9750, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=9750
[LightGBM] [Warning] feature_fraction is set=0.30000000000000004, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.30000000000000004
[LightGBM] [Warning] lambda_l2 is set=3.788831090842719, reg_lambda=0.0 will be ignored. Current value: lambda_l2=3.788831090842719
[LightGBM] [Warning] bagging_fraction is set=1.0, subsample=1.0 will be ignored. Current value: bagging_fraction=1.0
[LightGBM] [Warning] lambda_l1 is set=0.038777639106634655, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.038777639106634655
[LightGBM] [Warning] min_data_in_leaf is set=8100, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=8100
[LightGBM] [Warning] feature_fraction is set=0.30000000000000004, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.30000000000000004
[LightGBM] [Warning] lambda_l2 is set=1.549249797173697e-07, reg_lambda=0.0 will be ignored. Current value: lambda_l2=1.549249797173697e-07
[LightGBM] [Warning] bagging_fraction is set=1.0, subsample=1.0 will be ignored. Current value: bagging_fraction=1.0
[LightGBM] [Warning] lambda_l1 is set=0.038777639106634655, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.038777639106634655
[LightGBM] [Warning] min_data_in_leaf is set=8100, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=8100
[LightGBM] [Warning] feature_fraction is set=0.30000000000000004, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.30000000000000004
[LightGBM] [Warning] lambda_l2 is set=1.549249797173697e-07, reg_lambda=0.0 will be ignored. Current value: lambda_l2=1.549249797173697e-07
[LightGBM] [Warning] bagging_fraction is set=1.0, subsample=1.0 will be ignored. Current value: bagging_fraction=1.0
[LightGBM] [Warning] lambda_l1 is set=0.038777639106634655, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.038777639106634655
[LightGBM] [Warning] min_data_in_leaf is set=8100, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=8100
[LightGBM] [Warning] feature_fraction is set=0.30000000000000004, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.30000000000000004
[LightGBM] [Warning] lambda_l2 is set=1.549249797173697e-07, reg_lambda=0.0 will be ignored. Current value: lambda_l2=1.549249797173697e-07
[LightGBM] [Warning] bagging_fraction is set=1.0, subsample=1.0 will be ignored. Current value: bagging_fraction=1.0
[LightGBM] [Warning] lambda_l1 is set=0.038777639106634655, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.038777639106634655
[LightGBM] [Warning] min_data_in_leaf is set=8100, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=8100
[LightGBM] [Warning] feature_fraction is set=0.30000000000000004, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.30000000000000004
[LightGBM] [Warning] lambda_l2 is set=1.549249797173697e-07, reg_lambda=0.0 will be ignored. Current value: lambda_l2=1.549249797173697e-07
[LightGBM] [Warning] bagging_fraction is set=1.0, subsample=1.0 will be ignored. Current value: bagging_fraction=1.0
[LightGBM] [Warning] lambda_l1 is set=0.038777639106634655, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.038777639106634655
[LightGBM] [Warning] min_data_in_leaf is set=8100, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=8100
[LightGBM] [Warning] feature_fraction is set=0.30000000000000004, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.30000000000000004
[LightGBM] [Warning] lambda_l2 is set=1.549249797173697e-07, reg_lambda=0.0 will be ignored. Current value: lambda_l2=1.549249797173697e-07
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored. Current value: bagging_fraction=0.8
[LightGBM] [Warning] lambda_l1 is set=0.0003908758322984684, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.0003908758322984684
[LightGBM] [Warning] min_data_in_leaf is set=2250, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=2250
[LightGBM] [Warning] feature_fraction is set=0.5, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.5
[LightGBM] [Warning] lambda_l2 is set=3.3887445739685006e-07, reg_lambda=0.0 will be ignored. Current value: lambda_l2=3.3887445739685006e-07
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored. Current value: bagging_fraction=0.8
[LightGBM] [Warning] lambda_l1 is set=0.0003908758322984684, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.0003908758322984684
[LightGBM] [Warning] min_data_in_leaf is set=2250, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=2250
[LightGBM] [Warning] feature_fraction is set=0.5, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.5
[LightGBM] [Warning] lambda_l2 is set=3.3887445739685006e-07, reg_lambda=0.0 will be ignored. Current value: lambda_l2=3.3887445739685006e-07
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored. Current value: bagging_fraction=0.8
[LightGBM] [Warning] lambda_l1 is set=0.0003908758322984684, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.0003908758322984684
[LightGBM] [Warning] min_data_in_leaf is set=2250, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=2250
[LightGBM] [Warning] feature_fraction is set=0.5, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.5
[LightGBM] [Warning] lambda_l2 is set=3.3887445739685006e-07, reg_lambda=0.0 will be ignored. Current value: lambda_l2=3.3887445739685006e-07
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored. Current value: bagging_fraction=0.8
[LightGBM] [Warning] lambda_l1 is set=0.0003908758322984684, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.0003908758322984684
[LightGBM] [Warning] min_data_in_leaf is set=2250, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=2250
[LightGBM] [Warning] feature_fraction is set=0.5, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.5
```

```
[LightGBM] [Warning] lambda_l2 is set=1.4629022381559047e-06, reg_lambda=0.0 will be ignored. Current value: lambda_l2=1.4629022381559047e-06
[LightGBM] [Warning] bagging_fraction is set=0.6000000000000001, subsample=1.0 will be ignored. Current value: bagging_fraction=0.6000000000000001
[LightGBM] [Warning] lambda_l1 is set=0.00012059935796005053, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.00012059935796005053
[LightGBM] [Warning] min_data_in_leaf is set=200, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=200
[LightGBM] [Warning] feature_fraction is set=0.5, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.5
[LightGBM] [Warning] lambda_l2 is set=1.3501545627027436e-06, reg_lambda=0.0 will be ignored. Current value: lambda_l2=1.3501545627027436e-06
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored. Current value: bagging_fraction=0.8
[LightGBM] [Warning] lambda_l1 is set=0.00015596828834995834, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.00015596828834995834
[LightGBM] [Warning] min_data_in_leaf is set=450, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=450
[LightGBM] [Warning] feature_fraction is set=0.45, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.45
[LightGBM] [Warning] lambda_l2 is set=3.843701795833621e-07, reg_lambda=0.0 will be ignored. Current value: lambda_l2=3.843701795833621e-07
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored. Current value: bagging_fraction=0.8
[LightGBM] [Warning] lambda_l1 is set=0.00015596828834995834, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.00015596828834995834
[LightGBM] [Warning] min_data_in_leaf is set=450, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=450
[LightGBM] [Warning] feature_fraction is set=0.45, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.45
[LightGBM] [Warning] lambda_l2 is set=3.843701795833621e-07, reg_lambda=0.0 will be ignored. Current value: lambda_l2=3.843701795833621e-07
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored. Current value: bagging_fraction=0.8
[LightGBM] [Warning] lambda_l1 is set=0.00015596828834995834, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.00015596828834995834
[LightGBM] [Warning] min_data_in_leaf is set=450, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=450
[LightGBM] [Warning] feature_fraction is set=0.45, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.45
[LightGBM] [Warning] lambda_l2 is set=3.843701795833621e-07, reg_lambda=0.0 will be ignored. Current value: lambda_l2=3.843701795833621e-07
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored. Current value: bagging_fraction=0.8
[LightGBM] [Warning] lambda_l1 is set=0.00015596828834995834, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.00015596828834995834
[LightGBM] [Warning] min_data_in_leaf is set=450, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=450
[LightGBM] [Warning] feature_fraction is set=0.45, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.45
[LightGBM] [Warning] lambda_l2 is set=3.843701795833621e-07, reg_lambda=0.0 will be ignored. Current value: lambda_l2=3.843701795833621e-07
[LightGBM] [Warning] bagging_fraction is set=0.6000000000000001, subsample=1.0 will be ignored. Current value: bagging_fraction=0.6000000000000001
[LightGBM] [Warning] lambda_l1 is set=0.0010201340635994005, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.0010201340635994005
[LightGBM] [Warning] min_data_in_leaf is set=4250, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=4250
[LightGBM] [Warning] feature_fraction is set=0.6000000000000001, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.6000000000000001
[LightGBM] [Warning] lambda_l2 is set=4.5363277585793275e-06, reg_lambda=0.0 will be ignored. Current value: lambda_l2=4.5363277585793275e-06
[LightGBM] [Warning] bagging_fraction is set=0.7, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7
[LightGBM] [Warning] lambda_l1 is set=2.625904434537498e-05, reg_alpha=0.0 will be ignored. Current value: lambda_l1=2.625904434537498e-05
[LightGBM] [Warning] min_data_in_leaf is set=450, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=450
[LightGBM] [Warning] feature_fraction is set=0.7, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.7
[LightGBM] [Warning] lambda_l2 is set=1.7659445099641493e-06, reg_lambda=0.0 will be ignored. Current value: lambda_l2=1.7659445099641493e-06
[LightGBM] [Warning] bagging_fraction is set=0.7, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7
[LightGBM] [Warning] lambda_l1 is set=2.625904434537498e-05, reg_alpha=0.0 will be ignored. Current value: lambda_l1=2.625904434537498e-05
[LightGBM] [Warning] min_data_in_leaf is set=450, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=450
[LightGBM] [Warning] feature_fraction is set=0.7, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.7
[LightGBM] [Warning] lambda_l2 is set=1.7659445099641493e-06, reg_lambda=0.0 will be ignored. Current value: lambda_l2=1.7659445099641493e-06
[LightGBM] [Warning] bagging_fraction is set=0.7, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7
[LightGBM] [Warning] lambda_l1 is set=2.625904434537498e-05, reg_alpha=0.0 will be ignored. Current value: lambda_l1=2.625904434537498e-05
[LightGBM] [Warning] min_data_in_leaf is set=450, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=450
[LightGBM] [Warning] feature_fraction is set=0.7, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.7
[LightGBM] [Warning] lambda_l2 is set=1.7659445099641493e-06, reg_lambda=0.0 will be ignored. Current value: lambda_l2=1.7659445099641493e-06
[LightGBM] [Warning] bagging_fraction is set=0.7, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7
[LightGBM] [Warning] lambda_l1 is set=2.625904434537498e-05, reg_alpha=0.0 will be ignored. Current value: lambda_l1=2.625904434537498e-05
[LightGBM] [Warning] min_data_in_leaf is set=450, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=450
[LightGBM] [Warning] feature_fraction is set=0.7, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.7
[LightGBM] [Warning] lambda_l2 is set=1.7659445099641493e-06, reg_lambda=0.0 will be ignored. Current value: lambda_l2=1.7659445099641493e-06
[LightGBM] [Warning] bagging_fraction is set=0.6000000000000001, subsample=1.0 will be ignored. Current value: bagging_fraction=0.6000000000000001
[LightGBM] [Warning] lambda_l1 is set=6.782909085770697e-05, reg_alpha=0.0 will be ignored. Current value: lambda_l1=6.782909085770697e-05
[LightGBM] [Warning] min_data_in_leaf is set=700, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=700
[LightGBM] [Warning] feature_fraction is set=0.7, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.7
[LightGBM] [Warning] lambda_l2 is set=1.4428253723059194e-06, reg_lambda=0.0 will be ignored. Current value: lambda_l2=1.4428253723059194e-06
[LightGBM] [Warning] bagging_fraction is set=0.7, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7
[LightGBM] [Warning] lambda_l1 is set=0.00011345137546263752, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.00011345137546263752
[LightGBM] [Warning] min_data_in_leaf is set=400, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=400
[LightGBM] [Warning] feature_fraction is set=0.55, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.55
[LightGBM] [Warning] lambda_l2 is set=8.48154602701623e-06, reg_lambda=0.0 will be ignored. Current value: lambda_l2=8.48154602701623e-06
[LightGBM] [Warning] bagging_fraction is set=0.7, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7
[LightGBM] [Warning] lambda_l1 is set=0.00011345137546263752, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.00011345137546263752
[LightGBM] [Warning] min_data_in_leaf is set=400, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=400
[LightGBM] [Warning] feature_fraction is set=0.55, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.55
[LightGBM] [Warning] lambda_l2 is set=8.48154602701623e-06, reg_lambda=0.0 will be ignored. Current value: lambda_l2=8.48154602701623e-06
[LightGBM] [Warning] bagging_fraction is set=0.7, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7
[LightGBM] [Warning] lambda_l1 is set=0.00011345137546263752, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.00011345137546263752
[LightGBM] [Warning] min_data_in_leaf is set=400, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=400
[LightGBM] [Warning] feature_fraction is set=0.55, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.55
[LightGBM] [Warning] lambda_l2 is set=8.48154602701623e-06, reg_lambda=0.0 will be ignored. Current value: lambda_l2=8.48154602701623e-06
[LightGBM] [Warning] bagging_fraction is set=0.7, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7
[LightGBM] [Warning] lambda_l1 is set=0.00011345137546263752, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.00011345137546263752
[LightGBM] [Warning] min_data_in_leaf is set=400, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=400
[LightGBM] [Warning] feature_fraction is set=0.55, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.55
[LightGBM] [Warning] lambda_l2 is set=8.48154602701623e-06, reg_lambda=0.0 will be ignored. Current value: lambda_l2=8.48154602701623e-06
[LightGBM] [Warning] bagging_fraction is set=0.7, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7
[LightGBM] [Warning] lambda_l1 is set=2.4562888265860405e-05, reg_alpha=0.0 will be ignored. Current value: lambda_l1=2.4562888265860405e-05
[LightGBM] [Warning] min_data_in_leaf is set=1400, min_child_samples=20 will be ignored. Current value: min_data_in_leaf=1400
[LightGBM] [Warning] feature_fraction is set=0.55, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.55
[LightGBM] [Warning] lambda_l2 is set=8.006202569747522e-06, reg_lambda=0.0 will be ignored. Current value: lambda_l2=8.006202569747522e-06
```

In [73]:

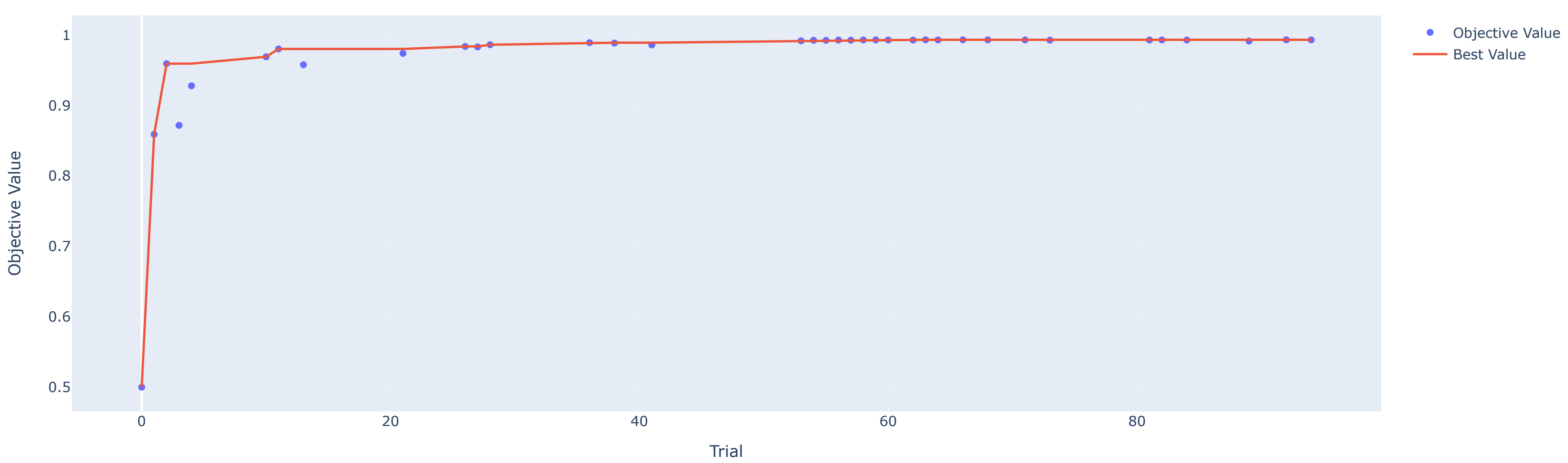
```python
print(f"\tBest value (auc): {study.best_value:.5f}")
print(f"\tBest params:")

for key, value in study.best_params.items():
    print(f"\t\t{key}: {value}")
```

```
        Best value (auc): 0.99327
        Best params:
                boosting_type: gbdt
                n_estimators: 2250
                learning_rate: 0.13583127937369815
                num_leaves: 3435
                max_depth: 10
                min_data_in_leaf: 450
                lambda_l1: 0.0001677268690716647
                lambda_l2: 7.486537913148734e-07
                bagging_fraction: 0.7
                feature_fraction: 0.6000000000000001
```

In [74]:

```python
optuna.visualization.plot_optimization_history(study)
```

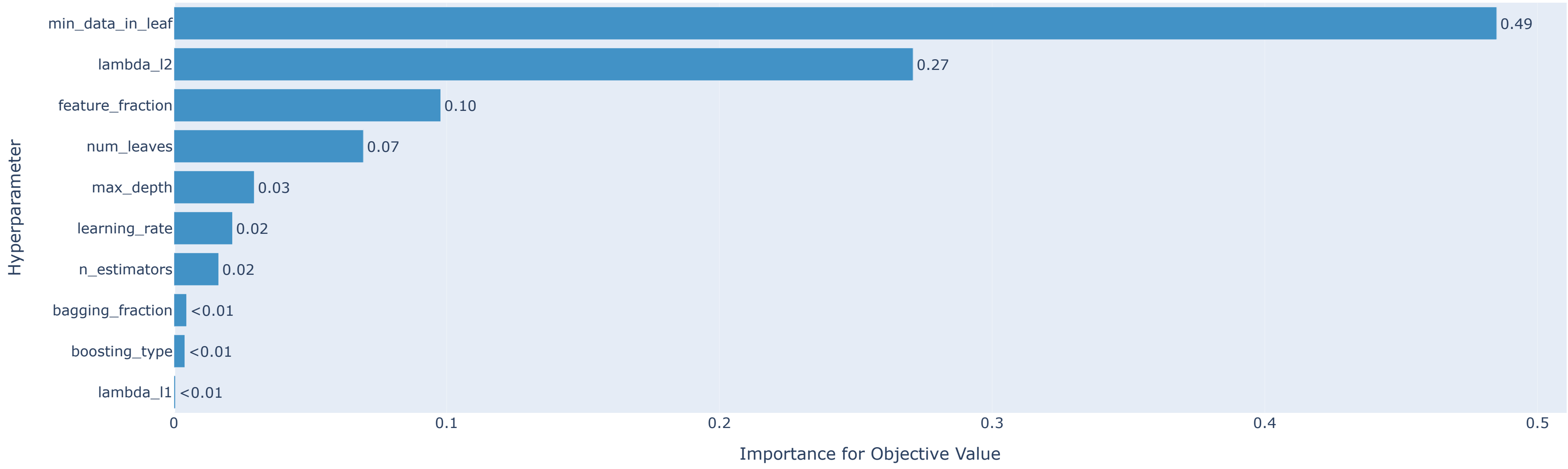
Optimization History Plot



In [75]:

```python
optuna.visualization.plot_param_importances(study)
```

Hyperparameter Importances



In [76]:
```
lgb_cat_final = lgbm.LGBMClassifier(objective="binary",**study.best_params)

lgb_cat_final.fit(X_train_lgb,y_train_lgb)
```

Out[76]:
```
▾                                LGBMClassifier
LGBMClassifier(bagging_fraction=0.7, feature_fraction=0.6000000000000001,
               lambda_l1=0.0001677268690716647, lambda_l2=7.486537913148734e-07,
               learning_rate=0.13583127937369815, max_depth=10,
               min_data_in_leaf=450, n_estimators=2250, num_leaves=3435,
               objective='binary')
```

In [77]:
```
get_results(lgb_cat_final,X_train_lgb,y_train_lgb)
```

```
ROC AUC score on input dataset is 0.999999760499644
Classification report is shown below:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     11055
           1       1.00      1.00      1.00     10953

    accuracy                           1.00     22008
   macro avg       1.00      1.00      1.00     22008
weighted avg       1.00      1.00      1.00     22008
```

In [78]:
```
get_results(lgb_cat_final,X_val_lgb,y_val_lgb)
```

```
ROC AUC score on input dataset is 0.99399767596817l3
Classification report is shown below:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96      3617
           1       0.96      0.97      0.96      3719

    accuracy                           0.96      7336
   macro avg       0.96      0.96      0.96      7336
weighted avg       0.96      0.96      0.96      7336
```

7.2 Predictions on Test Dataset

In [79]:
```
X_lgb_cols = list(X_val.columns)
X_test_lgb = df_test[X_lgb_cols]

y_test_pred = lgb_cat_final.predict_proba(X_test_lgb)[:,1]
y_test_pred = pd.DataFrame(y_test_pred,columns=['stroke_lgb_cat_cv'])

df_test = pd.concat([df_test,y_test_pred],axis=1)
```

Out of all these models, CatBoost Model gave the best performance. Hence final predictions were made using CatBoost Model

In [80]:
```
df_test['stroke'] = (df_test['stroke_cb'] + df_test['stroke_cb_cv'])/2

df_test[['id','stroke']].to_csv('Submission.csv',index=False)
```