Binary Prediction of Smoker Status using Bio-Signals

Import Libraries

In [6]:
```
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', None)

import math
from scipy import stats
from statsmodels.stats.outliers_influence import variance_inflation_factor
from collections import Counter

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# Preprocessing
from sklearn.preprocessing import MinMaxScaler, StandardScaler, MaxAbsScaler, RobustScaler, PowerTransformer, QuantileT
from sklearn.impute import SimpleImputer

# Model Selection
from sklearn.model_selection import train_test_split, cross_val_score, KFold, StratifiedKFold, train_test_split, Repeat
from sklearn.feature_selection import RFE, RFECV
from sklearn.ensemble import IsolationForest

# Models
from sklearn.ensemble import HistGradientBoostingClassifier, GradientBoostingClassifier,RandomForestClassifier,ExtraTre
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression, RidgeClassifier
from sklearn.naive_bayes import GaussianNB, BernoulliNB
from sklearn.ensemble import VotingClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
import optuna

# NN
import torch
import torch.nn as nn
import torch.optim as optim
if torch.cuda.is_available():
    device = torch.device("cuda")
else:
    device = torch.device("cpu")
from torch.utils.data import DataLoader, TensorDataset

# Metrics
from sklearn.metrics import roc_auc_score, roc_curve, make_scorer, f1_score, auc, confusion_matrix, classification_repo

import warnings
warnings.filterwarnings('ignore')
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is req
uired for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [7]:
```
# Adjusting plot style

rc = {
    "axes.facecolor": "#F8F8F8",
    "figure.facecolor": "#F8F8F8",
    "axes.edgecolor": "#000000",
    "grid.color": "#EBEBE7" + "30",
    "font.family": "serif",
    "axes.labelcolor": "#000000",
    "xtick.color": "#000000",
    "ytick.color": "#000000",
    "grid.alpha": 0.4
}

sns.set(rc=rc)
palette = ['#302c36', '#037d97', '#E4591E', '#C09741',
           '#EC5B6D', '#90A6B1', '#6ca957', '#D8E3E2']

from colorama import Style, Fore
blk = Style.BRIGHT + Fore.BLACK
mgt = Style.BRIGHT + Fore.MAGENTA
red = Style.BRIGHT + Fore.RED
blu = Style.BRIGHT + Fore.BLUE
res = Style.RESET_ALL
```

Load Data

In [8]:
```
train = pd.read_csv('/input/playground-series-s3e24/train.csv')
test = pd.read_csv('/input/playground-series-s3e24/test.csv')

origin = pd.read_csv('/input/smoker-status-prediction-using-biosignals/train_dataset.csv')
```

```
origin.dropna(inplace=True)

train['is_original']=0
test['is_original']=0
origin['is_original']=1

# Drop column id
train.drop('id',axis=1,inplace=True)
test.drop('id',axis=1,inplace=True)

train_total = pd.concat([train, origin], ignore_index=True)

total = pd.concat([train_total, test], ignore_index=True)
# total = pd.concat([train, test], ignore_index=True)

print('The shape of the train data:', train.shape)
print('The shape of the test data:', test.shape)

print('The shape of the origin data:', origin.shape)
print('The shape of the total data:', total.shape)
```

```
The shape of the train data: (159256, 24)
The shape of the test data: (106171, 23)
The shape of the origin data: (38984, 24)
The shape of the total data: (304411, 24)
```

In [9]:
```
total.head(3)
```

Out[9]:

	age	height(cm)	weight(kg)	waist(cm)	eyesight(left)	eyesight(right)	hearing(left)	hearing(right)	systolic	relaxation	fasting blood sugar	Cholester
0	55	165	60	81.0	0.5	0.6	1	1	135	87	94	17
1	70	165	65	89.0	0.6	0.7	2	2	146	83	147	19
2	20	170	75	81.0	0.4	0.5	1	1	118	75	79	17

In [10]:
```
target = 'smoking'

full_features = test.columns
num_var = [column for column in test.columns if test[column].nunique() > 10]
cat_var = [column for column in test.columns if test[column].nunique() < 10]
```

EDA

In [11]:
```
train.describe().T\
    .style.bar(subset=['mean'], color=px.colors.qualitative.G10[2])\
    .background_gradient(subset=['std'], cmap='Blues')\
    .background_gradient(subset=['50%'], cmap='BuGn')
```

Out[11]:

	count	mean	std	min	25%	50%	75%	max
age	159256.000000	44.306626	11.842286	20.000000	40.000000	40.000000	55.000000	85.000000
height(cm)	159256.000000	165.266929	8.818970	135.000000	160.000000	165.000000	170.000000	190.000000
weight(kg)	159256.000000	67.143662	12.586198	30.000000	60.000000	65.000000	75.000000	130.000000
waist(cm)	159256.000000	83.001990	8.957937	51.000000	77.000000	83.000000	89.000000	127.000000
eyesight(left)	159256.000000	1.005798	0.402113	0.100000	0.800000	1.000000	1.200000	9.900000
eyesight(right)	159256.000000	1.000989	0.392299	0.100000	0.800000	1.000000	1.200000	9.900000
hearing(left)	159256.000000	1.023974	0.152969	1.000000	1.000000	1.000000	1.000000	2.000000
hearing(right)	159256.000000	1.023421	0.151238	1.000000	1.000000	1.000000	1.000000	2.000000
systolic	159256.000000	122.503648	12.729315	77.000000	114.000000	121.000000	130.000000	213.000000
relaxation	159256.000000	76.874071	8.994642	44.000000	70.000000	78.000000	82.000000	133.000000
fasting blood sugar	159256.000000	98.352552	15.329740	46.000000	90.000000	96.000000	103.000000	375.000000
Cholesterol	159256.000000	195.796165	28.396959	77.000000	175.000000	196.000000	217.000000	393.000000
triglyceride	159256.000000	127.616046	66.188989	8.000000	77.000000	115.000000	165.000000	766.000000
HDL	159256.000000	55.852684	13.964141	9.000000	45.000000	54.000000	64.000000	136.000000
LDL	159256.000000	114.607682	28.158931	1.000000	95.000000	114.000000	133.000000	1860.000000
hemoglobin	159256.000000	14.796965	1.431213	4.900000	13.800000	15.000000	15.800000	21.000000
Urine protein	159256.000000	1.074233	0.347856	1.000000	1.000000	1.000000	1.000000	6.000000
serum creatinine	159256.000000	0.892764	0.179346	0.100000	0.800000	0.900000	1.000000	9.900000
AST	159256.000000	25.516853	9.464882	6.000000	20.000000	24.000000	29.000000	778.000000
ALT	159256.000000	26.550296	17.753070	1.000000	16.000000	22.000000	32.000000	2914.000000
Gtp	159256.000000	36.216004	31.204643	2.000000	18.000000	27.000000	44.000000	999.000000
dental caries	159256.000000	0.197996	0.398490	0.000000	0.000000	0.000000	0.000000	1.000000
smoking	159256.000000	0.437365	0.496063	0.000000	0.000000	0.000000	1.000000	1.000000
is_original	159256.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

Some observation:

The data is interpreted as follows:

- `id` : Unique identifier for each data point.
- `age` : Age of the individual, categorized in 5-year intervals.
- `height(cm)` : Height of the individual in centimeters.
- `weight(kg)` : Weight of the individual in kilograms.
- `waist(cm)` : Waist circumference of the individual in centimeters.
- `eyesight(left/right)` : Eyesight measurements for the left and right eyes.
- `hearing(left/right)` : Hearing ability for the left and right ears, represented as binary.
- `systolic` : Systolic blood pressure measurement.
- `relaxation` : Diastolic blood pressure measurement.
- `fasting blood sugar` : Fasting blood sugar level.
- `Cholesterol` : Total cholesterol level.
- `triglyceride` : Triglyceride level.
- `HDL` : High-density lipoprotein cholesterol level.
- `LDL` : Low-density lipoprotein cholesterol level.
- `hemoglobin` : Hemoglobin level in the blood.
- `Urine protein` : Level of protein in urine, categorized.
- `serum creatinine` : Serum creatinine level.
- `AST` : Level of aspartate aminotransferase enzyme.
- `ALT` : Level of alanine aminotransferase enzyme.
- `Gtp` : Level of gamma-glutamyl transferase enzyme.
- `dental caries` : Presence (1) or absence (0) of dental cavities.
- `smoking` : Target variable indicating if the individual is a smoker (1) or not (0).

In [12]:
```
def summary(df):
    sum = pd.DataFrame(df.dtypes, columns=['dtypes'])
    sum['missing#'] = df.isna().sum()
    sum['missing%'] = (df.isna().sum())/len(df)
    sum['uniques'] = df.nunique().values
    sum['count'] = df.count().values
    return sum

summary(total).style.background_gradient(cmap='Blues')
```

Out[12]:

	dtypes	missing#	missing%	uniques	count
age	int64	0	0.000000	21	304411
height(cm)	int64	0	0.000000	19	304411
weight(kg)	int64	0	0.000000	33	304411
waist(cm)	float64	0	0.000000	551	304411
eyesight(left)	float64	0	0.000000	21	304411
eyesight(right)	float64	0	0.000000	20	304411
hearing(left)	int64	0	0.000000	2	304411
hearing(right)	int64	0	0.000000	2	304411
systolic	int64	0	0.000000	128	304411
relaxation	int64	0	0.000000	94	304411
fasting blood sugar	int64	0	0.000000	259	304411
Cholesterol	int64	0	0.000000	280	304411
triglyceride	int64	0	0.000000	397	304411
HDL	int64	0	0.000000	123	304411
LDL	int64	0	0.000000	286	304411
hemoglobin	float64	0	0.000000	144	304411
Urine protein	int64	0	0.000000	6	304411
serum creatinine	float64	0	0.000000	34	304411
AST	int64	0	0.000000	196	304411
ALT	int64	0	0.000000	230	304411
Gtp	int64	0	0.000000	445	304411
dental caries	int64	0	0.000000	2	304411
smoking	float64	106171	0.348775	2	198240
is_original	int64	0	0.000000	2	304411

In [13]:
```
summary(test).style.background_gradient(cmap='Blues')
```

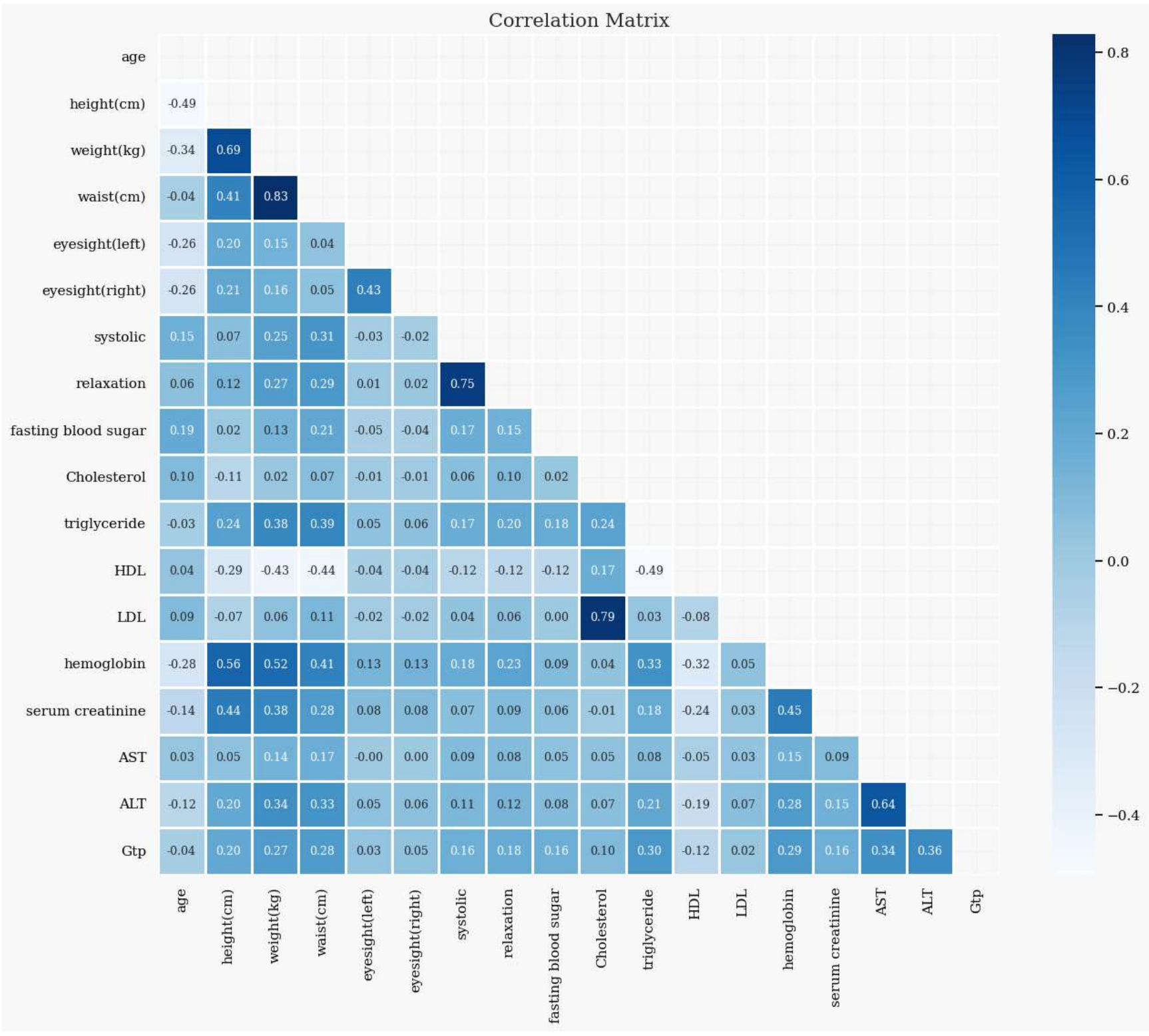
Out[13]:

	dtypes	missing#	missing%	uniques	count
age	int64	0	0.000000	18	106171
height(cm)	int64	0	0.000000	16	106171
weight(kg)	int64	0	0.000000	26	106171
waist(cm)	float64	0	0.000000	508	106171
eyesight(left)	float64	0	0.000000	20	106171
eyesight(right)	float64	0	0.000000	18	106171
hearing(left)	int64	0	0.000000	2	106171
hearing(right)	int64	0	0.000000	2	106171
systolic	int64	0	0.000000	114	106171
relaxation	int64	0	0.000000	78	106171
fasting blood sugar	int64	0	0.000000	224	106171
Cholesterol	int64	0	0.000000	227	106171
triglyceride	int64	0	0.000000	392	106171
HDL	int64	0	0.000000	106	106171
LDL	int64	0	0.000000	219	106171
hemoglobin	float64	0	0.000000	132	106171
Urine protein	int64	0	0.000000	6	106171
serum creatinine	float64	0	0.000000	26	106171
AST	int64	0	0.000000	135	106171
ALT	int64	0	0.000000	174	106171
Gtp	int64	0	0.000000	332	106171
dental caries	int64	0	0.000000	2	106171
is_original	int64	0	0.000000	1	106171

There are no missing values in the dataset.

Correlation

In [14]:
```
corr_matrix = total[num_var].corr()
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

plt.figure(figsize=(15, 12))
sns.heatmap(corr_matrix, mask=mask, annot=True, cmap='Blues', fmt='.2f', linewidths=1, square=True, annot_kws={"size": 
plt.title('Correlation Matrix', fontsize=15)
plt.show()
```



Some observation:

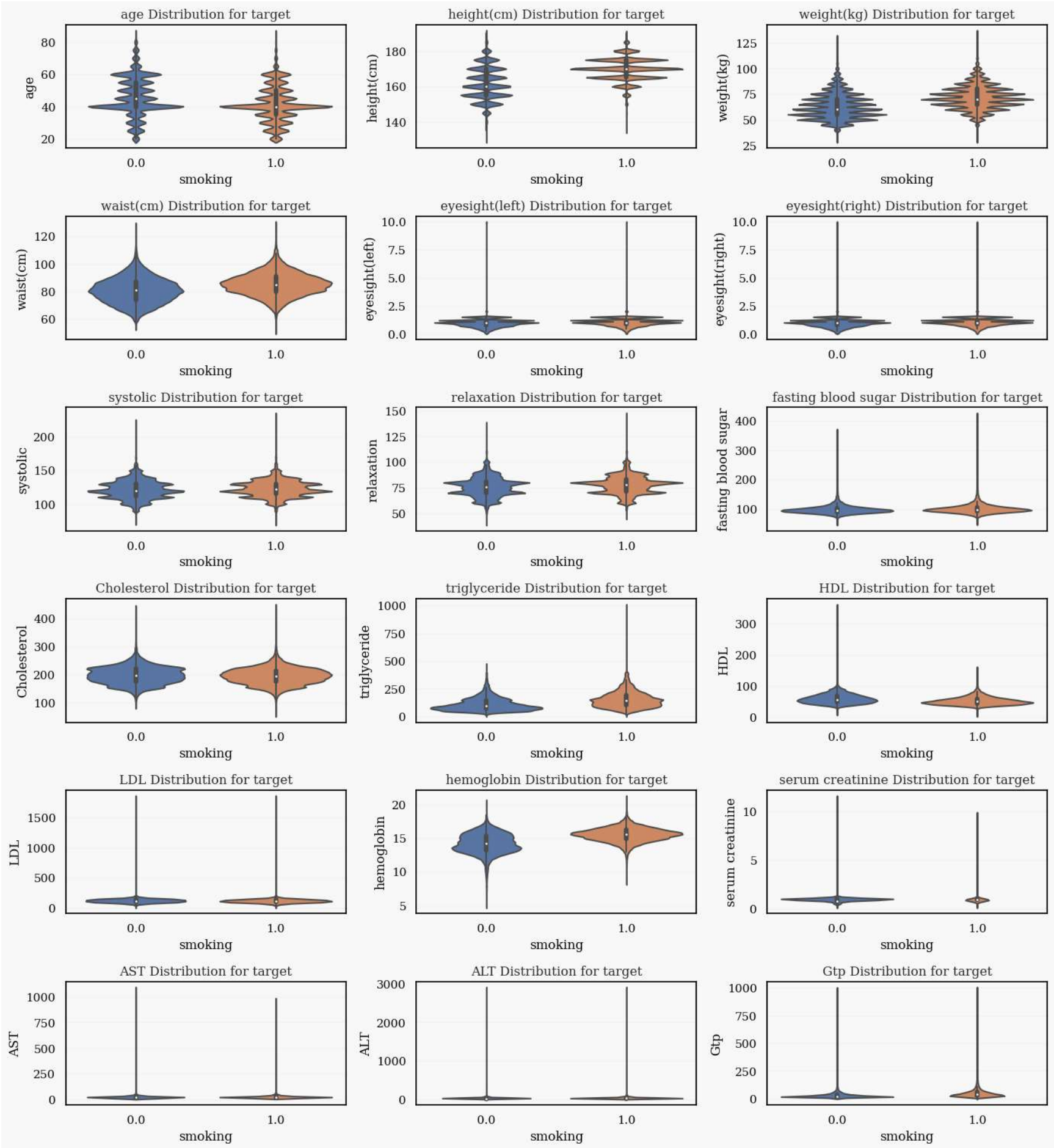
1. We don't have many highly correlated features in the total dataset.
2. Some features are correlated, we can take a look later.
3. Anyway, I didn't notice multicollinearity. This is a good thing.

Scatterplot of highly correlated variables

In [15]:
```
fig, axes = plt.subplots(2, 1, figsize=(20, 12))

sns.scatterplot(ax=axes[0], data=total, x='systolic', y='relaxation', hue=target)
sns.scatterplot(ax=axes[1], data=total, x='Cholesterol', y='LDL', hue=target)

plt.show()
```

Distribution of Target

In [16]:
```
num_rows = len(num_var)
num_cols = 3

total_plots = num_rows * num_cols
plt.figure(figsize=(14, num_rows * 2.5))

for idx, col in enumerate(num_var):
    plt.subplot(num_rows, num_cols, idx % total_plots + 1)
    sns.violinplot(x=target, y=col, data=total)
    plt.title(f"{col} Distribution for target")

plt.tight_layout()
plt.show()
```

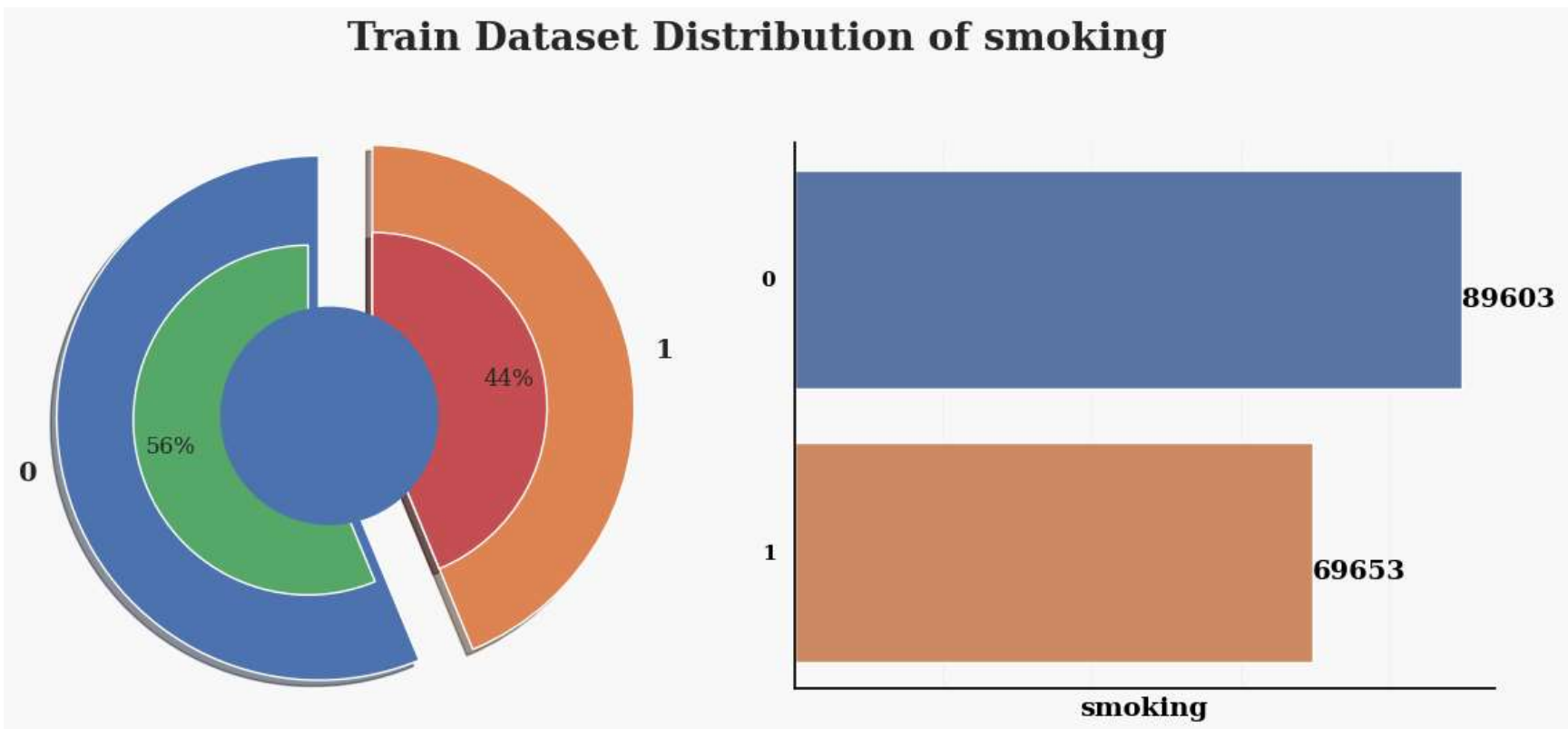


In [17]:

```
def plot_count(df, col_list, title_name ='Train'):
    f, ax = plt.subplots(len(col_list), 2, figsize=(12, 5))
    plt.subplots_adjust(wspace=0.3)

    for col in col_list:
        s1 = df[col].value_counts()
        N = len(s1)

        outer_sizes = s1
        inner_sizes = s1/N

        ax[0].pie(
            outer_sizes,
            labels=s1.index.tolist(),
            startangle=90, frame=True, radius=1.2,
            explode=([0.05]*(N-1) + [.2]),
            wedgeprops={'linewidth': 1, 'edgecolor': 'white'},
            textprops={'fontsize': 14, 'weight': 'bold'},
            shadow=True
        )

        ax[0].pie(
            inner_sizes,
            radius=0.8, startangle=90,
            autopct='%1.f%%', explode=([.1]*(N-1) + [.2]),
            pctdistance=0.8,
            shadow=True
        )

        center_circle = plt.Circle((0,0), .5, linewidth=0)
        ax[0].add_artist(center_circle)

        sns.barplot(
            x=s1, y=s1.index, ax=ax[1], orient='horizontal'
        )
```

```
        ax[1].spines['top'].set_visible(False)
        ax[1].spines['right'].set_visible(False)
        ax[1].tick_params(axis='x', which='both', bottom=False, labelbottom=False)
        ax[1].set_ylabel('')  # Remove y label

        for i, v in enumerate(s1):
            ax[1].text(v, i+0.1, str(v), color='black', fontweight='bold', fontsize=14)

        # Adding labels and title
        plt.setp(ax[1].get_yticklabels(), fontweight="bold")
        plt.setp(ax[1].get_xticklabels(), fontweight="bold")
        ax[1].set_xlabel(col, fontweight="bold", color='black', fontsize=14)

    f.suptitle(f'{title_name} Dataset Distribution of {col}', fontsize=20, fontweight='bold', y=1.05)
    plt.tight_layout()
    plt.show()
```

In [18]:
```
plot_count(train, ['smoking'], 'Train')
```

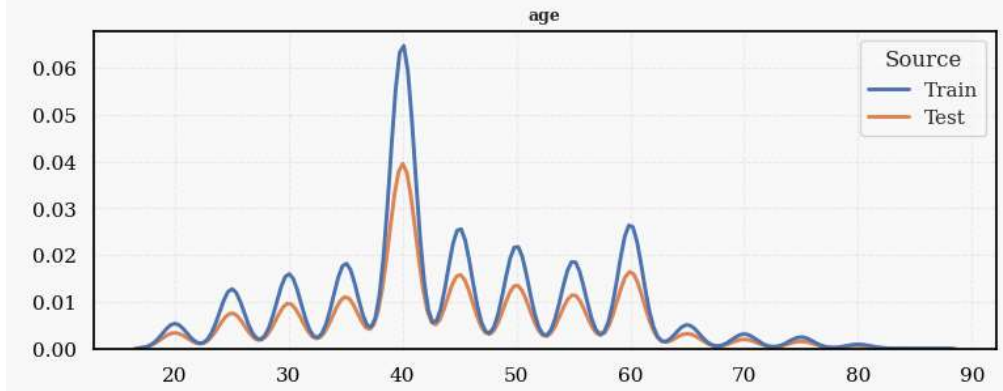


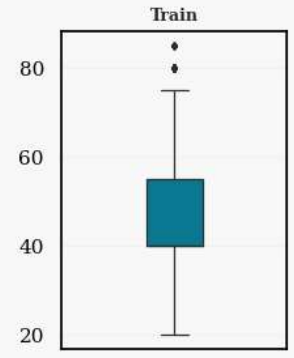
Some observation:

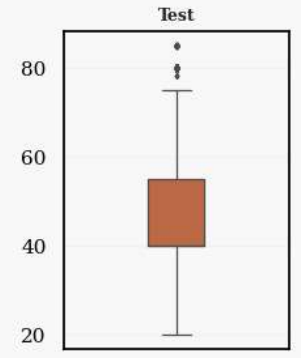
The distribution of the target variable is generally uniform.

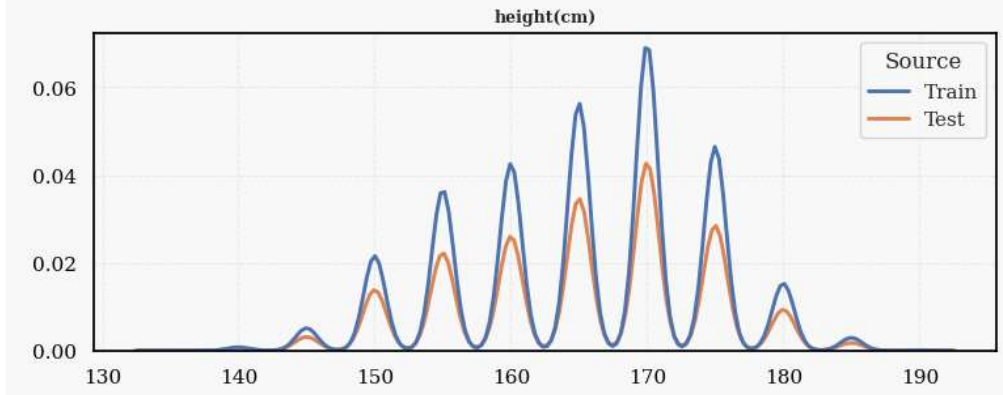
Distribution of Numeric Variables in Train/Test set

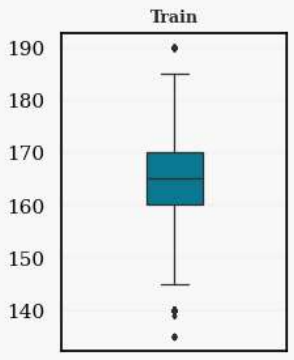
In [19]:
```
df = pd.concat([train[num_var].assign(Source = 'Train'),
                test[num_var].assign(Source = 'Test')],
               axis=0, ignore_index = True);

fig, axes = plt.subplots(len(num_var), 3 ,figsize = (16, len(num_var) * 4.2),
                         gridspec_kw = {'hspace': 0.35, 'wspace': 0.3, 'width_ratios': [0.80, 0.20, 0.20]});

for i,col in enumerate(num_var):
    ax = axes[i,0];
    sns.kdeplot(data = df[[col, 'Source']], x = col, hue = 'Source', ax = ax, linewidth = 2.1)
    ax.set_title(f"\n{col}",fontsize = 9, fontweight= 'bold');
    ax.grid(visible=True, which = 'both', linestyle = '--', color='lightgrey', linewidth = 0.75);
    ax.set(xlabel = '', ylabel = '');
    ax = axes[i,1];
    sns.boxplot(data = df.loc[df.Source == 'Train', [col]], y = col, width = 0.25,saturation = 0.90, linewidth = 0.90,
                ax = ax);
    ax.set(xlabel = '', ylabel = '');
    ax.set_title(f"Train",fontsize = 9, fontweight= 'bold');

    ax = axes[i,2];
    sns.boxplot(data = df.loc[df.Source == 'Test', [col]], y = col, width = 0.25, fliersize= 2.25,
                saturation = 0.6, linewidth = 0.90, color = '#E4591E',
                ax = ax);
    ax.set(xlabel = '', ylabel = '');
    ax.set_title(f"Test",fontsize = 9, fontweight= 'bold');

plt.tight_layout();
plt.show();
```

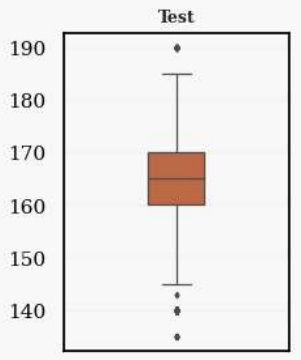

age
Source
Train
Test
0.06
0.05
0.04
0.03
0.02
0.01
0.00
20
30
40
50
60
70
80
90

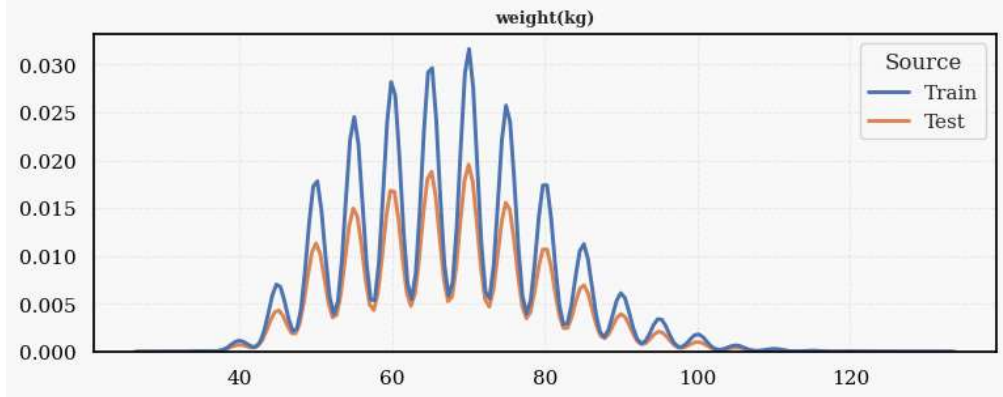

Train
80
60
40
20

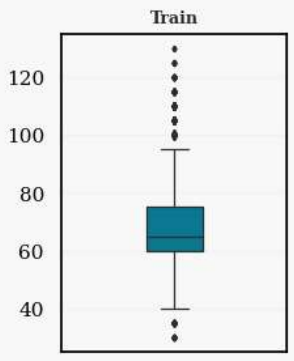

Test
80
60
40
20

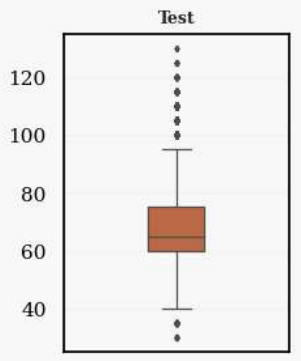

height(cm)
Source
Train
Test
0.06
0.04
0.02
0.00
130
140
150
160
170
180
190

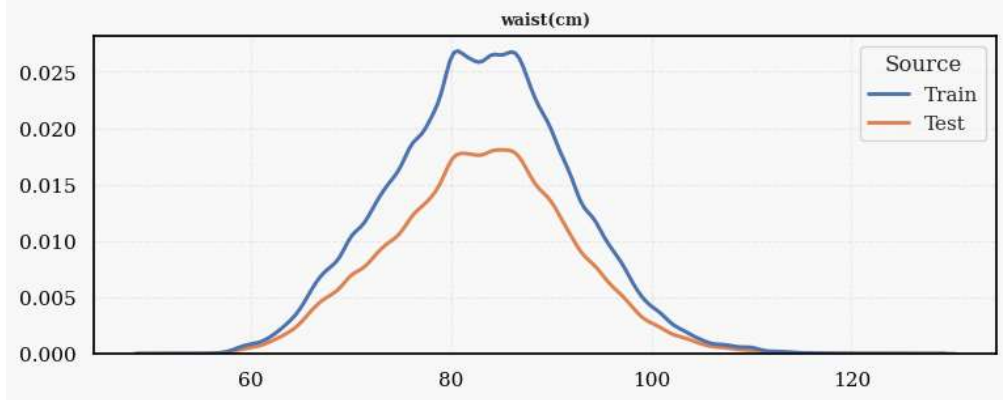

Train
190
180
170
160
150
140

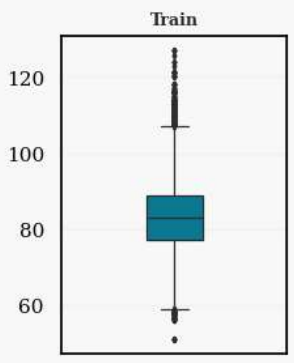

Test
190
180
170
160
150
140

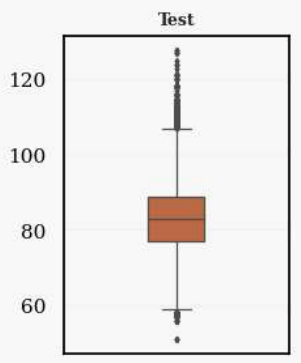

weight(kg)
Source
Train
Test
0.030
0.025
0.020
0.015
0.010
0.005
0.000
40
60
80
100
120

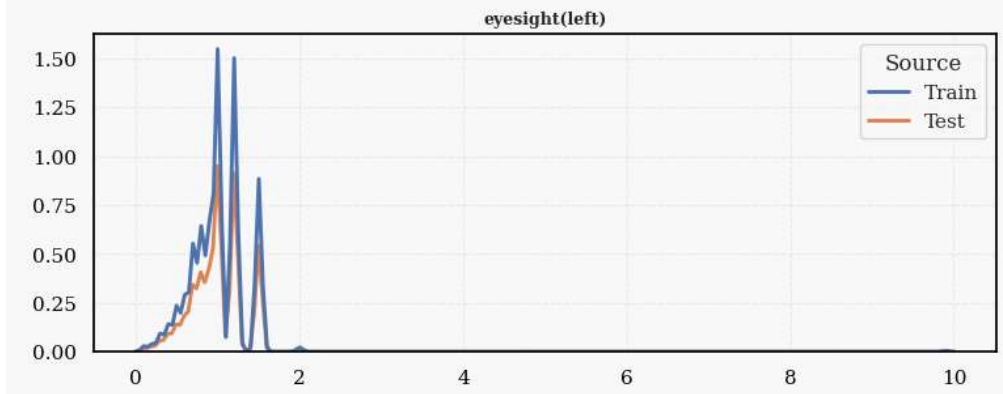

Train
120
100
80
60
40

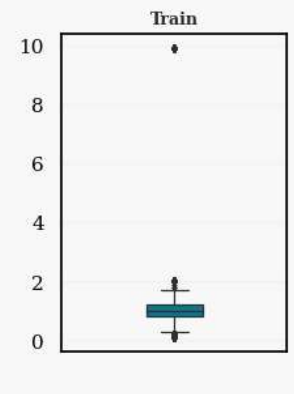

Test
120
100
80
60
40

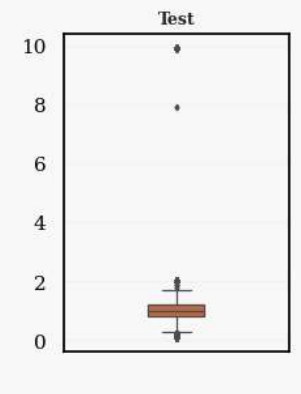

waist(cm)
Source
Train
Test
0.025
0.020
0.015
0.010
0.005
0.000
60
80
100
120

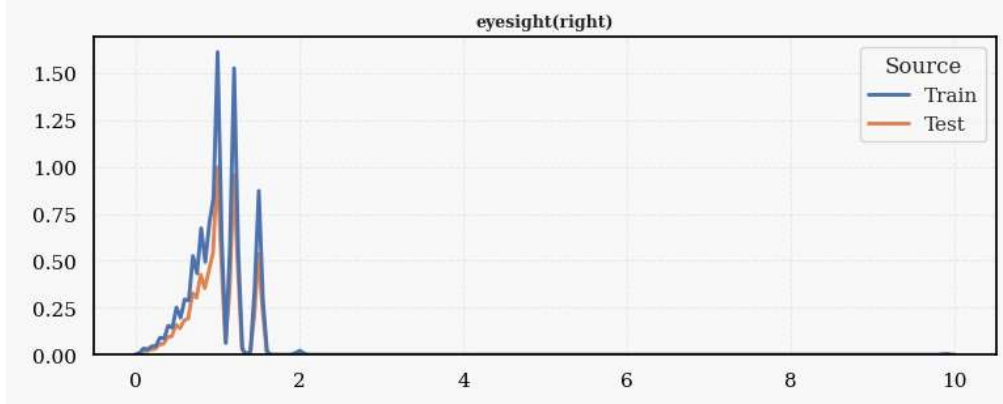

Train
120
100
80
60

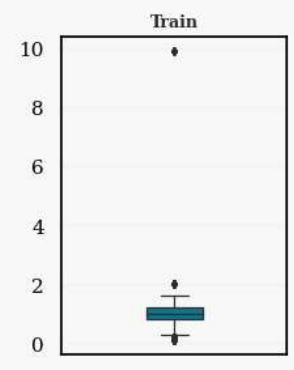

Test
120
100
80
60

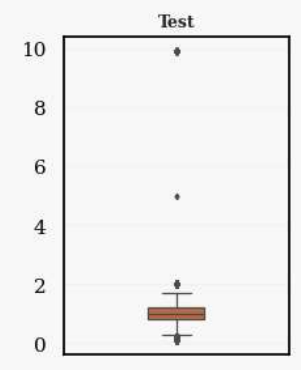

eyesight(left)
Source
Train
Test
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
2
4
6
8
10

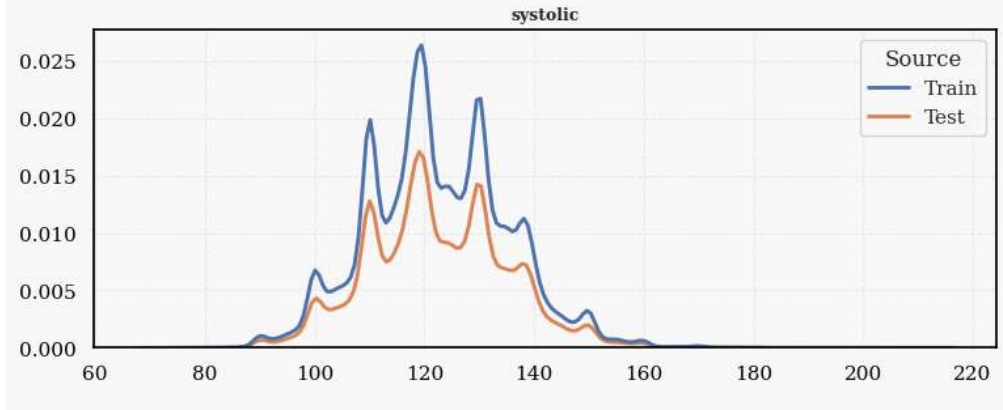

Train
10
8
6
4
2
0

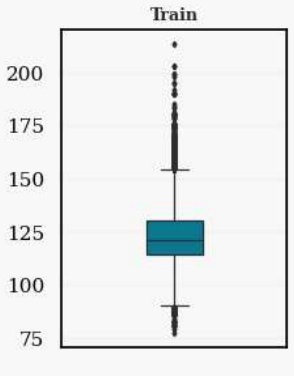

Test
10
8
6
4
2
0

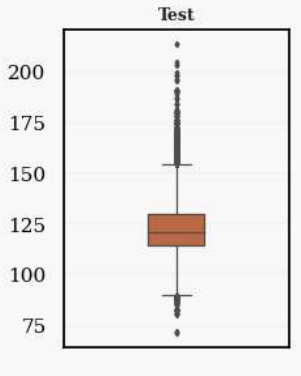

eyesight(right)
Source
Train
Test
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
2
4
6
8
10


Train
10
8
6
4
2
0


Test
10
8
6
4
2
0


systolic
Source
Train
Test
0.025
0.020
0.015
0.010
0.005
0.000
60
80
100
120
140
160
180
200
220

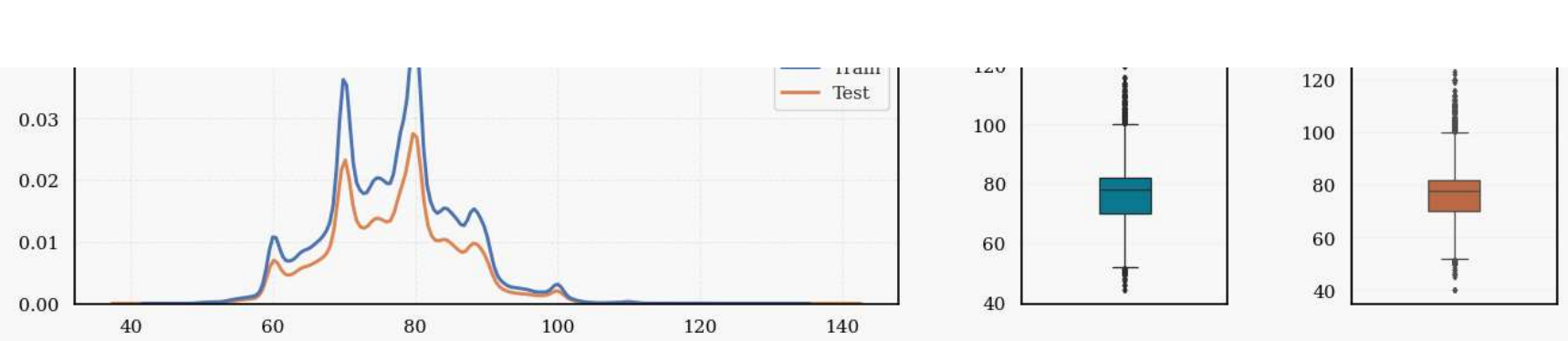

Train
200
175
150
125
100
75

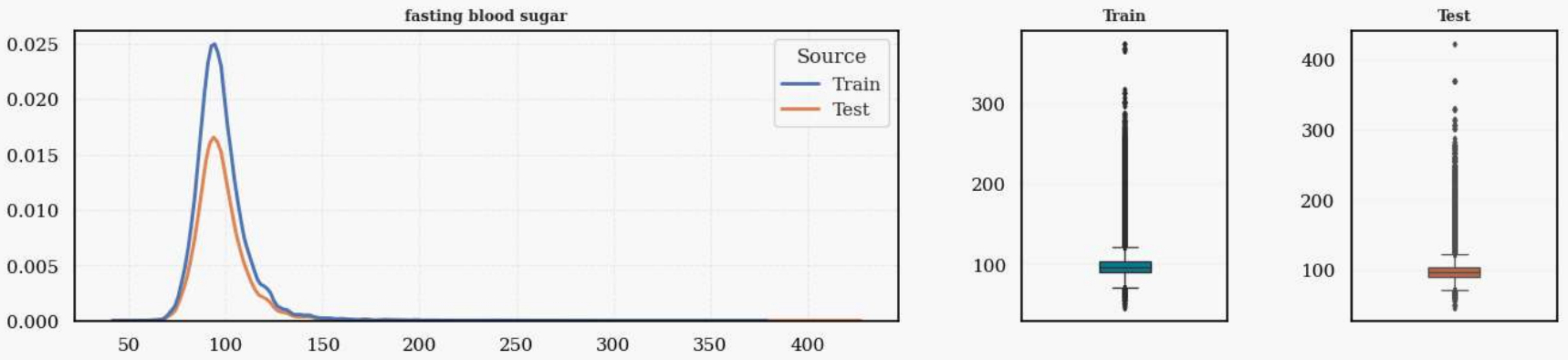

Test
200
175
150
125
100
75

relaxation
Source
Train
0.04

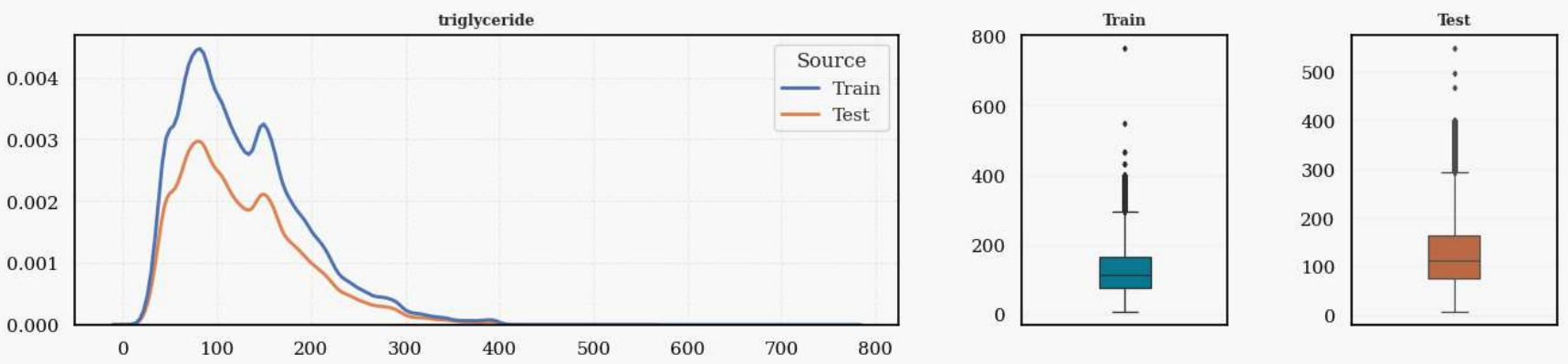
Train

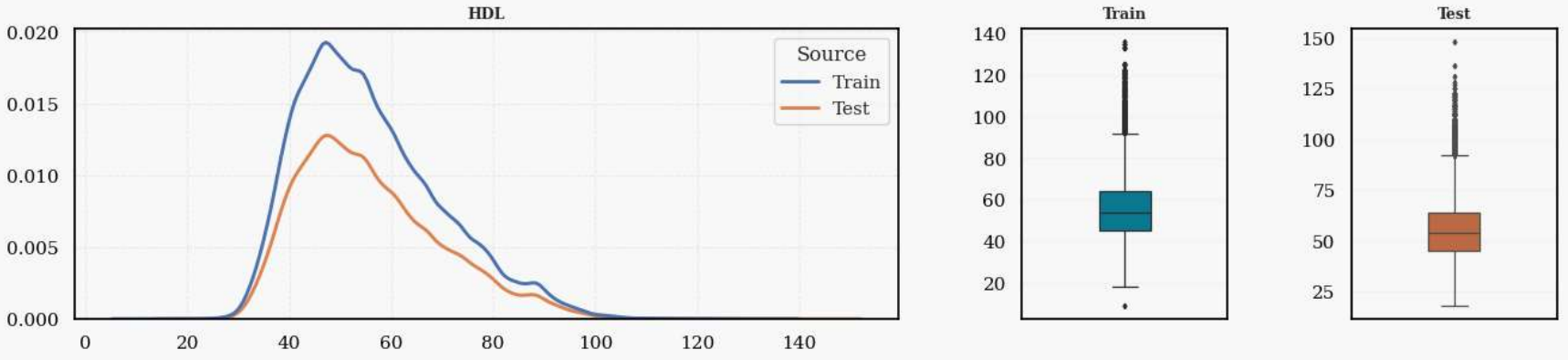
Test
140

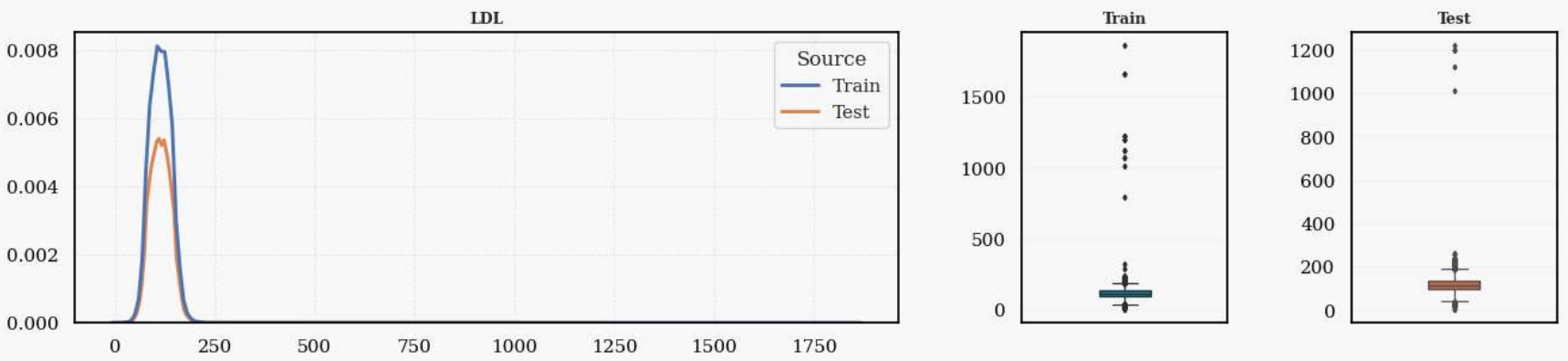

Train
Test
0.03
0.02
0.01
0.00
40
60
80
100
120
140
120
100
80
60
40
120
100
80
60
40

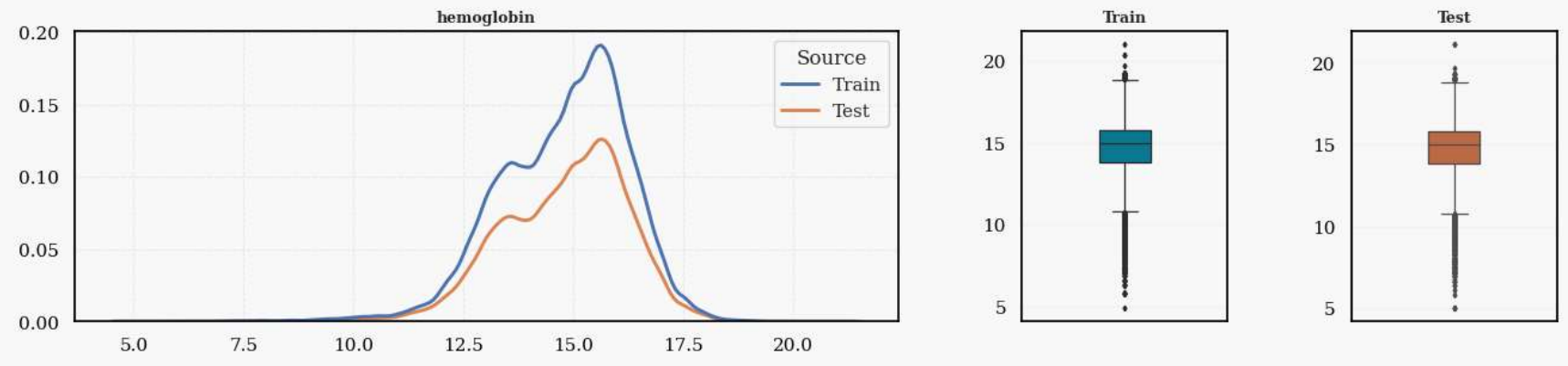

fasting blood sugar
Source
Train
Test
0.025
0.020
0.015
0.010
0.005
0.000
50
100
150
200
250
300
350
400
Train
300
200
100
Test
400
300
200
100

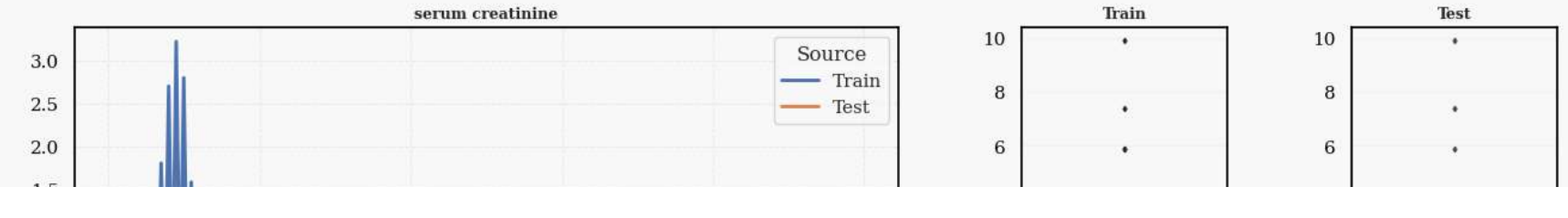
Cholesterol
Source
Train
Test
0.008
0.006
0.004
0.002
0.000
50
100
150
200
250
300
350
400
Train
400
350
300
250
200
150
100
Test
350
300
250
200
150
100

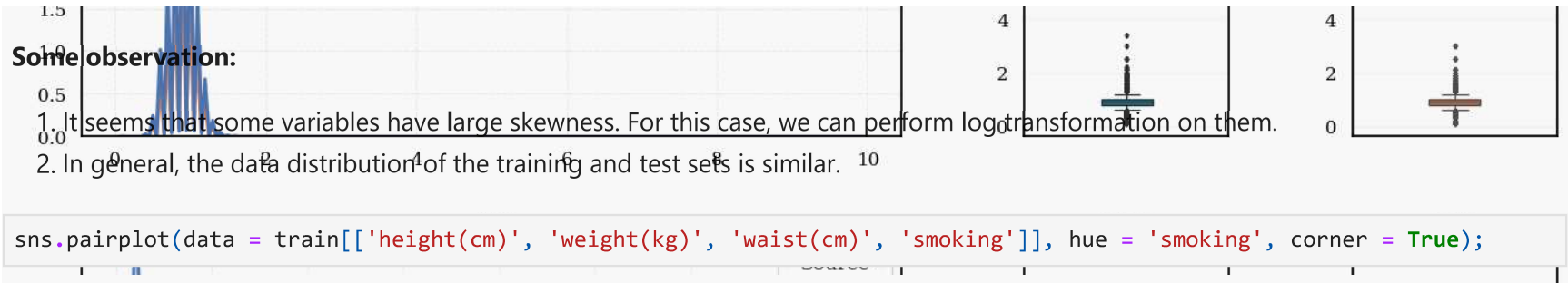

triglyceride
Source
Train
Test
0.004
0.003
0.002
0.001
0.000
0
100
200
300
400
500
600
700
800
Train
800
600
400
200
0
Test
500
400
300
200
100
0


HDL
Source
Train
Test
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
120
140
Train
140
120
100
80
60
40
20
Test
150
125
100
75
50
25


LDL
Source
Train
Test
0.008
0.006
0.004
0.002
0.000
0
250
500
750
1000
1250
1500
1750
Train
1500
1000
500
0
Test
1200
1000
800
600
400
200
0


hemoglobin
Source
Train
Test
0.20
0.15
0.10
0.05
0.00
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Train
20
15
10
5
Test
20
15
10
5

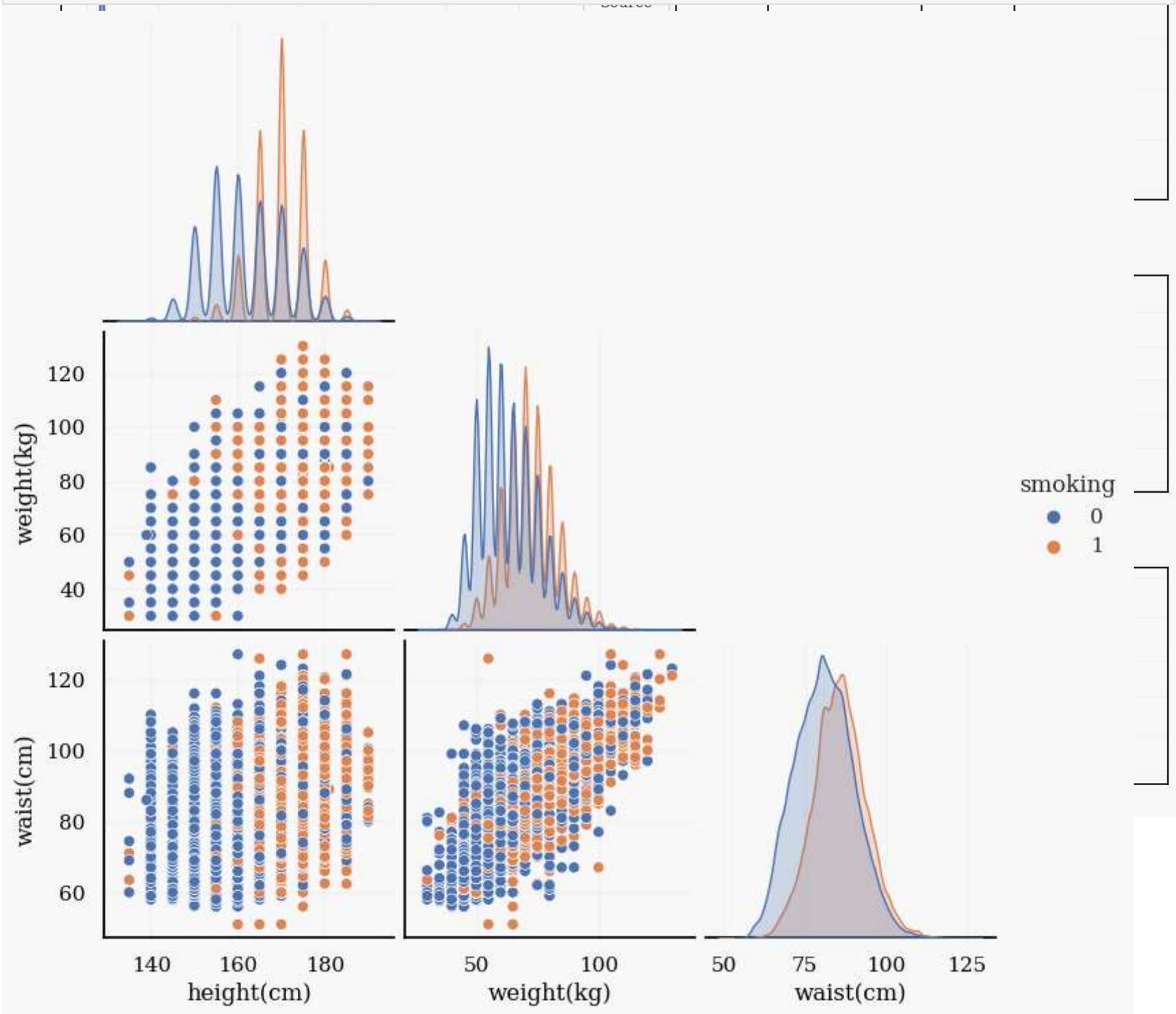

serum creatinine
Source
Train
Test
3.0
2.5
2.0
Train
10
8
6
Test
10
8
6



Some observation:

1. It seems that some variables have large skewness. For this case, we can perform log transformation on them.
2. In general, the data distribution of the training and test sets is similar.

In [20]:
```
sns.pairplot(data = train[['height(cm)', 'weight(kg)', 'waist(cm)', 'smoking']], hue = 'smoking', corner = True);
```



Some observation:

In some features(height,weight,waist), smoking or nonsmoking makes a significant difference in distribution.

Features Engineering

We can create some new features based on existing features.

In [21]:
```
def features_engineering(df):
    df= df.copy();
    df["BMI"] = df["weight(kg)"] / (df["height(cm)"]/100)**2
    df["HDL-LDL Ratio"] = df["HDL"] / df["LDL"]
    df["HDL-triglyceride Ratio"] = df["HDL"] / df["triglyceride"]
    df["LDL-triglyceride Ratio"] = df["LDL"] / df["triglyceride"]
    df["Liver Enzyme Ratio"] = df["AST"] / df["ALT"]
    return df
```

We can also make some funtion to handle with those outliers.

Handle Outliers

In [22]:
```
def corr_skew_outlier(df, cols):
    for col in cols:
        Q1 = df[col].quantile(0.01)
        Q3 = df[col].quantile(0.95)
        df.loc[df[col] < Q1, col] = Q1
        df.loc[df[col] > Q3, col] = Q3
        df[col] = np.sqrt(df[col])

    return df
```

In [23]:
```
def count_outliers(df, select_features):
    df_subset = df[select_features]
    clf = IsolationForest(contamination='auto')
    predictions = clf.fit_predict(df_subset)
```

```
    outlier_count_df = pd.DataFrame({
        'Outlier_Count': [(pred == -1) for pred in predictions]
    })
    total_outliers = outlier_count_df['Outlier_Count'].sum()
    df['Outlier_Count'] = outlier_count_df
    return df
```

In [24]:
```
# total = features_engineering(total)
# total = corr_skew_outlier(total,num_var)
total = count_outliers(total,full_features.drop('is_original'))
```

In [25]:
```
df_train = total[total[target].notna()]

df_test = total[total[target].isna()]
df_test = df_test.copy()
df_test.drop(target, axis=1, inplace=True)
```

Baseline Model

In [26]:
```
lgbm_baseline = LGBMClassifier()

roc_results = pd.DataFrame(columns=['Selected_Features', 'ROC'])

def evaluation(df, select_features, note):
    global roc_results

    X = df[select_features]
    Y = df[target]

    kf = KFold(n_splits=3, shuffle=True, random_state=42)
    roc_scores = []

    for train_idx, test_idx in kf.split(X):
        X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
        y_train, y_test = Y.iloc[train_idx], Y.iloc[test_idx]

        lgbm_baseline.fit(X_train, y_train)
        y_hat = lgbm_baseline.predict_proba(X_test)[:, 1]
        roc = roc_auc_score(y_test, y_hat)
        roc_scores.append(roc)

    average_roc = np.mean(roc_scores)
    new_row = {'Selected_Features': note, 'ROC': average_roc}
    roc_results = pd.concat([roc_results, pd.DataFrame([new_row])], ignore_index=True)

    print('====================================')
    print(note)
    print("Average ROC:", average_roc)
    print('====================================')
    return average_roc
```

In [27]:
```
evaluation(df=df_train,select_features=full_features,note='Baseline')
```

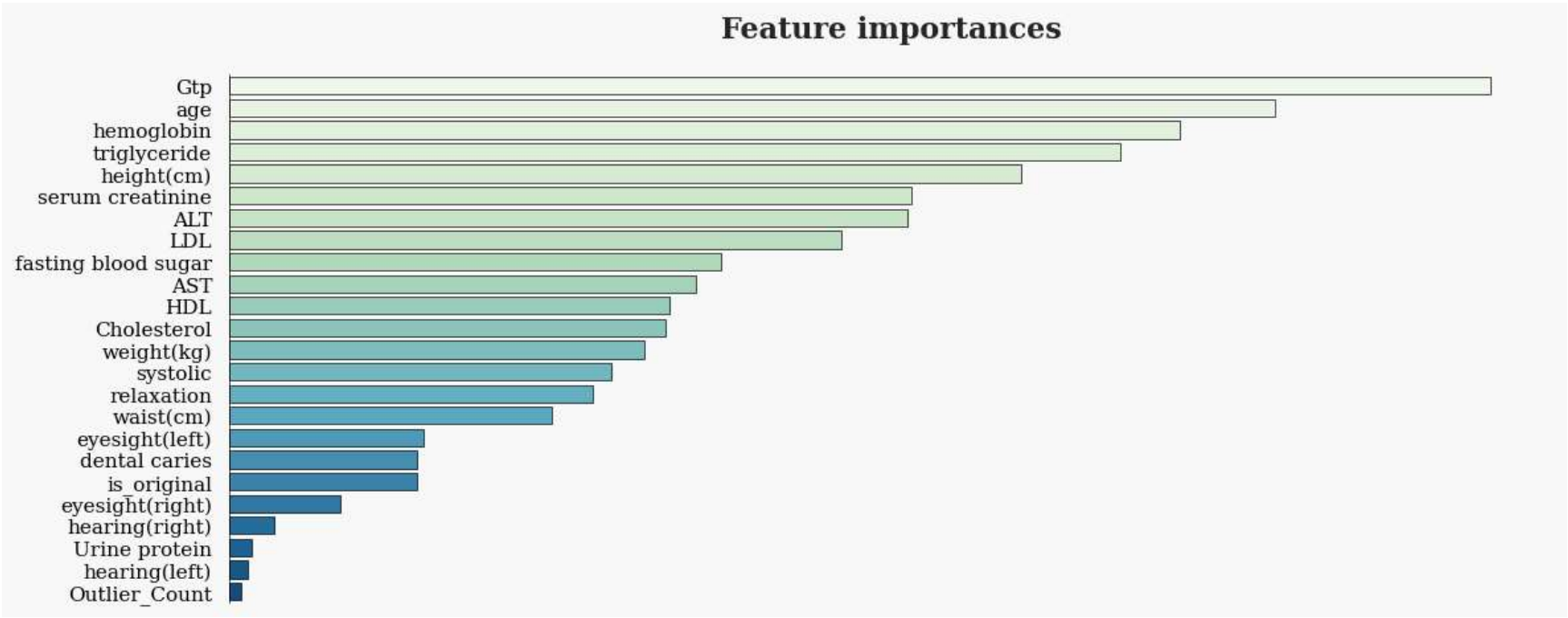
```
====================================
Baseline
Average ROC: 0.8623393225008904
====================================
```

Out[27]:
```
0.8623393225008904
```

Features Importance

In [28]:
```
def f_importance_plot(f_imp):
    fig = plt.figure(figsize=(12, 0.20*len(f_imp)))
    plt.title(f'Feature importances', size=16, y=1.05,
              fontweight='bold')
    a = sns.barplot(data=f_imp, x='imp', y='feature', linestyle="-",
                    linewidth=0.5, edgecolor="black",palette='GnBu')
    plt.xlabel('')
    plt.xticks([])
    plt.ylabel('')
    plt.yticks(size=11)

    for j in ['right', 'top', 'bottom']:
        a.spines[j].set_visible(False)
    for j in ['left']:
        a.spines[j].set_linewidth(0.5)
    plt.tight_layout()
    plt.show()
```

In [29]:
```
clf = LGBMClassifier()
clf.fit(df_train.drop(target,axis=1), df_train[target])

f_imp_df = pd.DataFrame({'feature': df_train.drop(target,axis=1).columns, 'imp': clf.feature_importances_})
f_imp_df.sort_values(by='imp',ascending=False,inplace=True)
f_importance_plot(f_imp_df)
```



Features Selection: RFECV

In [30]:
```
# clf = LGBMClassifier()
# rfe = RFECV(clf)
# X_rfe = rfe.fit_transform(df_train.drop(target,axis=1), df_train[target])
# clf.fit(X_rfe,df_train[target])

# cols = list(df_train.drop(target,axis=1).columns)
# temp = pd.Series(rfe.support_,index = cols)
# featureRFE = temp[temp==True].index.tolist()
```

In [31]:
```
featureRFE = ['age', 'height(cm)', 'weight(kg)', 'waist(cm)', 'relaxation', 'fasting blood sugar', 'Cholesterol', 'trig
print(featureRFE)
```

```
['age', 'height(cm)', 'weight(kg)', 'waist(cm)', 'relaxation', 'fasting blood sugar', 'Cholesterol', 'triglyceride', 'H
DL', 'LDL', 'hemoglobin', 'serum creatinine', 'AST', 'ALT', 'Gtp']
```

In [32]:
```
evaluation(df=df_train,select_features=featureRFE,note='Feature RFE')
```

```
====================================
Feature RFE
Average ROC: 0.8594516491945695
====================================
```

Out[32]:
```
0.8594516491945695
```

Modeling

In [33]:
```
#X = df_train.drop(target,axis=1)

X = df_train[featureRFE]
Y = df_train[target]

df_pred = df_test[featureRFE]
#df_pred = df_test
```

Hill Climbing

In [34]:
```
def hill_climbing(x, y, x_test):

    # Evaluating oof predictions
    scores = {}
    for col in x.columns:
        scores[col] = roc_auc_score(y, x[col])

    # Sorting the model scores
    scores = {k: v for k, v in sorted(scores.items(), key = lambda item: item[1], reverse = True)}

    # Sort oof_df and test_preds
    x = x[list(scores.keys())]
    x_test = x_test[list(scores.keys())]

    STOP = False
    current_best_ensemble = x.iloc[:,0]
    current_best_test_preds = x_test.iloc[:,0]
    MODELS = x.iloc[:,1:]
    weight_range = np.arange(-0.5, 0.51, 0.01)
    history = [roc_auc_score(y, current_best_ensemble)]
    j = 0

    while not STOP:
        j += 1
        potential_new_best_cv_score = roc_auc_score(y, current_best_ensemble)
        k_best, wgt_best = None, None
        for k in MODELS:
            for wgt in weight_range:
                potential_ensemble = (1 - wgt) * current_best_ensemble + wgt * MODELS[k]
                cv_score = roc_auc_score(y, potential_ensemble)
                if cv_score > potential_new_best_cv_score:
                    potential_new_best_cv_score = cv_score
```

```
                    k_best, wgt_best = k, wgt

        if k_best is not None:
            current_best_ensemble = (1 - wgt_best) * current_best_ensemble + wgt_best * MODELS[k_best]
            current_best_test_preds = (1 - wgt_best) * current_best_test_preds + wgt_best * x_test[k_best]
            MODELS.drop(k_best, axis = 1, inplace = True)
            if MODELS.shape[1] == 0:
                STOP = True
            history.append(potential_new_best_cv_score)
        else:
            STOP = True

    hill_ens_pred_1 = current_best_ensemble
    hill_ens_pred_2 = current_best_test_preds

    return [hill_ens_pred_1, hill_ens_pred_2]
```

In [35]:
```
def training_model(clf,X_train,Y_train,X_test,Y_test,df_pred):
    clf.fit(X_train,Y_train)

    cv_pred = clf.predict_proba(X_test)[:, 1]
    cv_score = roc_auc_score(Y_test, cv_pred)
    pred = clf.predict_proba(df_pred)[:, 1]

    return cv_pred,cv_score,pred
```

Base Model Configs

In [36]:
```
random_state = 42

lgb_params = {
    "objective": 'binary',
    'metric': 'auc',
    'boosting_type': 'dart',
     "n_estimators": 1000,
     "max_depth": 7,
     "learning_rate":0.03,
     "num_leaves": 50,
     "reg_alpha":3,
     "reg_lambda": 3,
     "subsample": 0.7,
     "colsample_bytree": 0.7
}

xgb_optuna = {
    'n_estimators': 10000,
    'learning_rate': 0.01752354328845971,
    'booster': 'gbtree',
    'lambda': 0.08159630121074074,
    'alpha': 0.07564858712175693,
    'subsample': 0.5065979400270813,
    'colsample_bytree': 0.6187340851873067,
    'max_depth': 4,
    'min_child_weight': 5,
    'eta': 0.2603059902806757,
    'gamma': 0.6567360773618207,
    'tree_method': 'hist',
    'random_state': random_state
}

cat_params = {
    'iterations': 10000,
    'eval_metric': 'AUC',
    'loss_function': 'Logloss',
    'auto_class_weights': 'Balanced'
}
```

In [37]:
```
models = {
    'LGBM': LGBMClassifier(**lgb_params),
    'XGB': XGBClassifier(**xgb_optuna),
    'Cat': CatBoostClassifier(**cat_params),
    'RF': RandomForestClassifier(),
    'Hist':HistGradientBoostingClassifier()
}

model_preds = {} # To save predctions from each model
```

In [ ]:
```
ens_cv_scores, ens_preds = list(), list()
hill_ens_cv_scores, hill_ens_preds =  list(), list()

sk = RepeatedStratifiedKFold(n_splits = 3, n_repeats = 1, random_state = 23)
for i, (train_idx, test_idx) in enumerate(sk.split(X, Y)):

    X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
    Y_train, Y_test = Y.iloc[train_idx], Y.iloc[test_idx]

    print('------------------------------------------------------------')

    for model_name, model in models.items():
        model_pred, model_score, model_pred_test = training_model(model, X_train, Y_train, X_test, Y_test, df_pred)
        print(f'Fold {i+1} ==> {model_name} oof ROC-AUC score is ==> {model_score}')

        model_preds[model_name] = (model_pred, model_pred_test)
```

```
    # Ensemble
    ens_pred_1 = sum(pred[0] for pred in model_preds.values()) / len(models)
    ens_pred_2 = sum(pred[1] for pred in model_preds.values()) / len(models)

    ens_score_fold = roc_auc_score(Y_test, ens_pred_1)
    ens_cv_scores.append(ens_score_fold)
    ens_preds.append(ens_pred_2)

    # Hill Climbing Ensemble
    x = pd.DataFrame({model_name: model_preds[model_name][0] for model_name in models})
    y = Y_test

    x_test = pd.DataFrame({model_name: model_preds[model_name][1] for model_name in models})

    hill_results = hill_climbing(x, y, x_test)

    hill_ens_score_fold = roc_auc_score(y, hill_results[0])
    hill_ens_cv_scores.append(hill_ens_score_fold)
    hill_ens_preds.append(hill_results[1])

    print(f'Fold {i+1} ==> Hill Climbing Ensemble oof ROC-AUC score is ==> {hill_ens_score_fold}')
    print(f'Fold {i+1} ==> Average Ensemble oof ROC-AUC score is ==> {ens_score_fold}')

    print('The hill climbing ensemble oof ROC-AUC score over the 3-folds is', np.mean(hill_ens_cv_scores))
```

In [39]:
```
hill_preds_test = pd.DataFrame(hill_ens_preds).apply(np.mean, axis = 0)

hill_ensemble_result = sample_result.copy()

hill_ensemble_result ['smoking'] = hill_preds_test
hill_ensemble_result.to_csv('Ensemble_result.csv', index = False)
```

In [40]:
```
hill_ensemble_result.head(3)
```

Out[40]:

	id	smoking
0	159256	0.654561
1	159257	0.344251
2	159258	0.279983

Pseudo Labels

In [41]:
```
import pandas as pd

def create_pseudo_label(df_pred,target,pred_label,cutoff=0.95):
    df_input = df_pred.copy()
    df_input[target] = pred_label.values

    pseudo_set_1 = (df_input[df_input[target] > cutoff]
                    .assign(smoking=1)
                    )

    pseudo_set_2 = (df_input[df_input[target] < 1 - cutoff]
                    .assign(smoking=0)
                    )

    df_pseudo = pd.concat([pseudo_set_1, pseudo_set_2])

    return df_pseudo
```

In [42]:
```
df_pseudo = create_pseudo_label(df_pred=df_pred,target=target,pred_label=hill_ensemble_submission[target],cutoff= 0.95)
df_pseudo.head()
```

Out[42]:

	age	height(cm)	weight(kg)	waist(cm)	relaxation	fasting blood sugar	Cholesterol	triglyceride	HDL	LDL	hemoglobin	serum creatinine	AST	ALT
198359	40	170	80	87.0	80	103	204	234	45	111	16.4	1.0	27	48
198574	40	175	90	94.0	80	91	225	271	48	123	16.3	1.2	44	88
198875	40	170	70	87.0	80	103	209	380	44	89	15.8	1.0	23	28
199735	40	170	80	89.0	78	92	195	287	45	92	15.9	0.9	26	20
199791	40	175	75	96.0	80	114	200	264	66	86	15.6	0.9	42	65

In [43]:
```
df_train_total = pd.concat([df_train,df_pseudo])
df_train_total.shape
```

Out[43]:
```
(220609, 25)
```

Hyperparameter optimization of individual models

In [44]:
```
X = df_train_total[featureRFE]
Y = df_train_total[target]
```

In [45]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y,test_size=0.1, random_state = 42)
```

LGBM

In [46]:
```
def objective_lgb(trial):
   params = {
        'metric': trial.suggest_categorical('metric', ['binary_error']),
        'max_depth': trial.suggest_int('max_depth', 1, 30),
      'min_child_samples': trial.suggest_int('min_child_samples', 1, 15),
        'learning_rate': trial.suggest_loguniform('learning_rate', 0.01, 0.05),
       'n_estimators': trial.suggest_int('n_estimators', 300, 2000),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 20),
         'subsample': trial.suggest_float('subsample', 0.1, 0.9),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.01, 1.0),
        'reg_alpha': trial.suggest_float('reg_alpha', 0.01, 1.0),
        'reg_lambda': trial.suggest_float('reg_lambda', 0.01, 1.0),
     }


    model_lgb = LGBMClassifier(seed=42,**params)
    model_lgb.fit(X_train, y_train)
    y_pred = model_lgb.predict(X_test)
    return accuracy_score(y_test,y_pred)

opt_lgb = optuna.create_study(direction='maximize')
opt_lgb.optimize(objective_lgb, n_trials=300,show_progress_bar=True)

opt_lgb_params = opt_lgb.best_params
```

In [47]:
```
opt_lgb_params = {
                  'metric': 'binary_error',
                  'max_depth': 5,
                  'min_child_samples': 10,
                  'learning_rate': 0.045879163813688256,
                  'n_estimators': 1821,
                  'min_child_weight': 6,
                  'subsample': 0.11770708304786026,
                  'colsample_bytree': 0.49515159614595267,
                  'reg_alpha': 0.4218485459727929,
                  'reg_lambda': 0.28289526226193323
                   }
```

In [48]:
```
opt_lgb_model = LGBMClassifier(**opt_lgb_params).fit(X_train, y_train)
opt_lgb_pred = opt_lgb_model.predict_proba(df_pred)[:, 1]

opt_lgb_result = pd.DataFrame({'id': sample_result['id'], 'smoking': opt_lgb_pred})
opt_lgb_result.to_csv('opt_lgb_result.csv',index=False)
```

In [49]:
```
opt_lgb_result.head(3)
```

Out[49]:

	id	smoking
0	159256	0.644249
1	159257	0.192475
2	159258	0.293562

XGBoost

In [50]:
```
def objective_xgb(trial):
     params = {
        'max_depth': trial.suggest_int('max_depth', 1, 30),
        'learning_rate': trial.suggest_loguniform('learning_rate', 0.01, 0.05),
        'n_estimators': trial.suggest_int('n_estimators', 300, 2000),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 10),
        'gamma': trial.suggest_loguniform('gamma', 1e-8, 1.0),
        'subsample': trial.suggest_loguniform('subsample', 0.3, 0.9),
        'reg_alpha': trial.suggest_float('reg_alpha', 0.01, 1.0),
        'reg_lambda': trial.suggest_float('reg_lambda', 0.01, 1.0),
    }
     model_xgb = XGBClassifier(booster='gbtree', objective='binary:logistic',seed=42,**params)
    model_xgb.fit(X_train, y_train)
    y_pred = model_xgb.predict(X_test)
    return accuracy_score(y_test,y_pred)

opt_xgb = optuna.create_study(direction='maximize')
opt_xgb.optimize(objective_xgb, n_trials=300,show_progress_bar=True)

opt_xgb_params = opt_xgb.best_params
opt_xgb_params
```

In [51]:
```
opt_xgb_params = {
                  'n_estimators': 2048,
                  'max_depth': 9,
                  'learning_rate': 0.045,
                  'booster': 'gbtree',
                  'subsample': 0.75,
                  'colsample_bytree': 0.30,
                  'reg_lambda': 1.00,
```

```
                  'reg_alpha': 0.80,
                  'gamma': 0.80,
                  'random_state': 42,
                  'objective': 'binary:logistic',
                  'tree_method': 'gpu_hist',
                  'eval_metric': 'auc',
                  'n_jobs': -1,
                 }

opt_xgb_model = XGBClassifier(**opt_xgb_params).fit(X_train, y_train)
opt_xgb_pred = opt_xgb_model.predict_proba(df_pred)[:, 1]

opt_xgb_result = pd.DataFrame({'id': sample_result['id'], 'smoking': opt_xgb_pred})
opt_xgb_result.to_csv('opt_xgb_result.csv',index=False)
```

In [52]:
```
opt_xgb_result.head(3)
```

Out[52]:

	id	smoking
0	159256	0.664803
1	159257	0.275221
2	159258	0.303842

Voting Model

In [53]:
```
voting = VotingClassifier(estimators=[
                                      ('lgbm', opt_lgb_model),
                                      ('xgb', opt_xgb_model)], voting='soft')
voting.fit(X,Y)

opt_voting_pred = voting.predict_proba(df_pred)[:, 1]

opt_voting_result = pd.DataFrame({'id': sample_result['id'], 'smoking': opt_voting_pred})
opt_voting_result.to_csv('opt_voting_result.csv',index=False)
opt_voting_result.head(3)
```

Out[53]:

	id	smoking
0	159256	0.677596
1	159257	0.210440
2	159258	0.315342

NN by pytorch

In [54]:
```
train_set = torch.tensor(X.values.astype(float), dtype=torch.float32).to(device)
train_target = torch.tensor(Y.values.astype(float), dtype=torch.float32).to(device)

pred_set = torch.tensor(df_pred.values.astype(float),dtype=torch.float32).to(device)
```

In [55]:
```
input_dim = train_set.shape[1]
```

In [56]:
```
class NN(nn.Module):
    def __init__(self, input_dim, hidden_dim1=64, hidden_dim2=32):
        super(NN, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim1)
        self.bn1 = nn.BatchNorm1d(hidden_dim1)
        self.dropout1 = nn.Dropout(0.2)
        self.fc2 = nn.Linear(hidden_dim1, hidden_dim2)
        self.bn2 = nn.BatchNorm1d(hidden_dim2)
        self.dropout2 = nn.Dropout(0.2)
        self.fc3 = nn.Linear(hidden_dim2, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.bn1(x)
        x = self.dropout1(x)
        x = torch.relu(self.fc2(x))
        x = self.bn2(x)
        x = self.dropout2(x)
        x = self.fc3(x)
        return self.sigmoid(x)
```

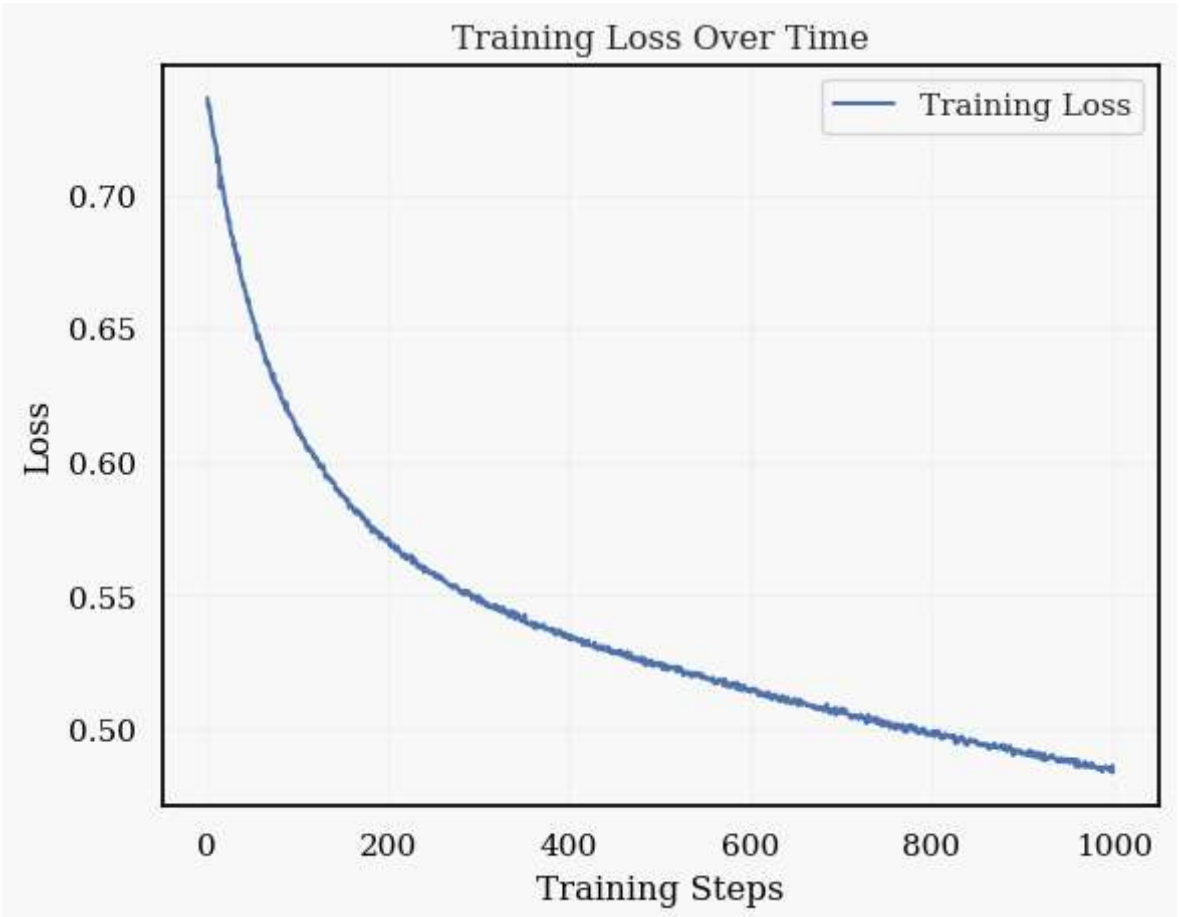
In [57]:
```
model = NN(input_dim).to(device)

criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)

loss_history = []

epochs = 1000
for epoch in range(1000):
    optimizer.zero_grad()
    outputs = model(train_set).squeeze()

    loss = criterion(outputs, train_target)
    loss_history.append(loss.item())
```

```
    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')

    loss.backward()
    optimizer.step()
train_steps = range(1, len(loss_history) + 1)
```

```
Epoch [10/1000], Loss: 0.7189
Epoch [20/1000], Loss: 0.6979
Epoch [30/1000], Loss: 0.6808
Epoch [40/1000], Loss: 0.6664
Epoch [50/1000], Loss: 0.6545
Epoch [60/1000], Loss: 0.6446
Epoch [70/1000], Loss: 0.6348
Epoch [80/1000], Loss: 0.6265
Epoch [90/1000], Loss: 0.6186
Epoch [100/1000], Loss: 0.6129
Epoch [110/1000], Loss: 0.6054
Epoch [120/1000], Loss: 0.6019
Epoch [130/1000], Loss: 0.5964
Epoch [140/1000], Loss: 0.5926
Epoch [150/1000], Loss: 0.5873
Epoch [160/1000], Loss: 0.5837
Epoch [170/1000], Loss: 0.5801
Epoch [180/1000], Loss: 0.5765
Epoch [190/1000], Loss: 0.5726
Epoch [200/1000], Loss: 0.5703
Epoch [210/1000], Loss: 0.5670
Epoch [220/1000], Loss: 0.5639
Epoch [230/1000], Loss: 0.5624
Epoch [240/1000], Loss: 0.5589
Epoch [250/1000], Loss: 0.5579
Epoch [260/1000], Loss: 0.5564
Epoch [270/1000], Loss: 0.5539
Epoch [280/1000], Loss: 0.5521
Epoch [290/1000], Loss: 0.5506
Epoch [300/1000], Loss: 0.5501
Epoch [310/1000], Loss: 0.5471
Epoch [320/1000], Loss: 0.5460
Epoch [330/1000], Loss: 0.5438
Epoch [340/1000], Loss: 0.5414
Epoch [350/1000], Loss: 0.5415
Epoch [360/1000], Loss: 0.5394
Epoch [370/1000], Loss: 0.5382
Epoch [380/1000], Loss: 0.5375
Epoch [390/1000], Loss: 0.5370
Epoch [400/1000], Loss: 0.5338
Epoch [410/1000], Loss: 0.5330
Epoch [420/1000], Loss: 0.5329
Epoch [430/1000], Loss: 0.5304
Epoch [440/1000], Loss: 0.5302
Epoch [450/1000], Loss: 0.5293
Epoch [460/1000], Loss: 0.5283
Epoch [470/1000], Loss: 0.5263
Epoch [480/1000], Loss: 0.5271
Epoch [490/1000], Loss: 0.5251
Epoch [500/1000], Loss: 0.5241
Epoch [510/1000], Loss: 0.5230
Epoch [520/1000], Loss: 0.5235
Epoch [530/1000], Loss: 0.5209
Epoch [540/1000], Loss: 0.5199
Epoch [550/1000], Loss: 0.5191
Epoch [560/1000], Loss: 0.5172
Epoch [570/1000], Loss: 0.5165
Epoch [580/1000], Loss: 0.5166
Epoch [590/1000], Loss: 0.5145
Epoch [600/1000], Loss: 0.5150
Epoch [610/1000], Loss: 0.5129
Epoch [620/1000], Loss: 0.5119
Epoch [630/1000], Loss: 0.5123
Epoch [640/1000], Loss: 0.5100
Epoch [650/1000], Loss: 0.5104
Epoch [660/1000], Loss: 0.5100
Epoch [670/1000], Loss: 0.5091
Epoch [680/1000], Loss: 0.5078
Epoch [690/1000], Loss: 0.5069
Epoch [700/1000], Loss: 0.5080
Epoch [710/1000], Loss: 0.5045
Epoch [720/1000], Loss: 0.5047
Epoch [730/1000], Loss: 0.5055
Epoch [740/1000], Loss: 0.5042
Epoch [750/1000], Loss: 0.5016
Epoch [760/1000], Loss: 0.5001
Epoch [770/1000], Loss: 0.4990
Epoch [780/1000], Loss: 0.5002
Epoch [790/1000], Loss: 0.4990
Epoch [800/1000], Loss: 0.4982
Epoch [810/1000], Loss: 0.4977
Epoch [820/1000], Loss: 0.4972
Epoch [830/1000], Loss: 0.4966
Epoch [840/1000], Loss: 0.4957
Epoch [850/1000], Loss: 0.4954
Epoch [860/1000], Loss: 0.4944
Epoch [870/1000], Loss: 0.4942
Epoch [880/1000], Loss: 0.4939
Epoch [890/1000], Loss: 0.4903
Epoch [900/1000], Loss: 0.4913
Epoch [910/1000], Loss: 0.4902
Epoch [920/1000], Loss: 0.4897
Epoch [930/1000], Loss: 0.4894
Epoch [940/1000], Loss: 0.4879
Epoch [950/1000], Loss: 0.4881
```

```
Epoch [960/1000], Loss: 0.4880
Epoch [970/1000], Loss: 0.4863
Epoch [980/1000], Loss: 0.4862
Epoch [990/1000], Loss: 0.4854
Epoch [1000/1000], Loss: 0.4842
```

In [58]:
```
plt.plot(train_steps, loss_history, label='Training Loss')
plt.xlabel('Training Steps')
plt.ylabel('Loss')
plt.legend()
plt.title('Training Loss Over Time')
plt.show()
```



Prediction

In [59]:
```
model.eval()
with torch.no_grad():
    probs = model(pred_set)

probs = probs.cpu().flatten().numpy()
nn_result = pd.DataFrame({'id': sample_result['id'], 'smoking': probs})
nn_result.to_csv('nn_result.csv',index=False)
```

In [60]:
```
nn_result.head(3)
```

Out[60]:

	id	smoking
0	159256	0.630980
1	159257	0.332526
2	159258	0.654702