Import

In [1]:

```
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', None)
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import StratifiedKFold, cross_validate, learning_curve, RandomizedSearchCV, GridSearchCV, train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score, plot_confusion_matrix, make_scorer, accuracy_score, auc, precision_recall_curve, average_precision_score
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import PowerTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from imblearn.over_sampling import SMOTE

import xgboost as xgb
import lightgbm as lgb

import os
import sys
```

Input

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
file_path = '/content/drive/MyDrive/stats504/'
```

```
Mounted at /content/drive
```

In [3]:

```
df = pd.read_csv(file_path+'nuMoM2bsubset.csv')
df.drop(columns=['dv.gestweeks', 'dv.v3epdstotal', 'dv.preeclampsia', 'dv.diabetes1', 'v1epdstotal'], inplace=True)
# df.drop_duplicates(inplace=True)
df.head(2)
```

Out[3]:

	age	race	emosupport	financialsupport	prenatalsupport	deliverysupport	psstotal	anxtotal	worryfambaby	exercise	sys
0	31.0	white	1.0	1.0	1.0	1.0	26	36.0	5.0	1.0	1
1	26.0	black	1.0	1.0	1.0	1.0	30	34.0	5.0	2.0	1

Data preprocessing

In [5]:

In [5]:

```
def data_preporcess(df,cat):
    y = df.loc[:, 'dv.hypertension1']
    x = df.drop(columns='dv.hypertension1')
    print('Total shape: ', x.shape, y.shape)

    # x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=test_size, random_state=random_state, shuffle=shuffle)

    train_df = pd.read_csv(file_path+'train_df.csv', index_col=[0])
    test_df = pd.read_csv(file_path+'test_df.csv', index_col=[0])

    train_df = train_df.drop_duplicates()
    train_df[["age","income","prepreglbs"]] = train_df[["age","income","prepreglbs"]].replace({'0':np.nan, 0:np.nan})

    y_train = train_df.loc[:, 'dv.hypertension1']
    y_test = test_df.loc[:, 'dv.hypertension1']

    if cat==True:
        x_train = train_df.drop(columns=['dv.hypertension1'])
        x_test = test_df.drop(columns=['dv.hypertension1'])
        x_train = pd.get_dummies(x_train)
        x_test = pd.get_dummies(x_test)
    else:
        x_train = train_df.drop(columns=['dv.hypertension1', 'race'])
        x_test = test_df.drop(columns=['dv.hypertension1', 'race'])


    print('Train shape: ', x_train.shape, y_train.shape)
    print('Test shape: ', x_test.shape, y_test.shape)

    return x, y, x_train, y_train, x_test, y_test
```

Imbalance Data

In [6]:

```
def smote_balance(x_train, y_train, r, k):
    oversample = SMOTE(r, k_neighbors=k)
    print(f'Shape of the training before SMOTE: {x_train.shape, y_train.shape}')

    x_tr_resample, y_tr_resample = oversample.fit_resample(x_train, y_train)
    print(f'Shape of the training after SMOTE: {x_tr_resample.shape, y_tr_resample.shape}')

    # Target distribution before SMOTE
    non_fraud = 0
    fraud = 0

    for i in y_train:
        if i == 0:
            non_fraud +=1
        else:
            fraud +=1

    # Target distribution after SMOTE
    no = 0
    yes = 1

    for j in y_tr_resample:
        if j == 0:
            no +=1
        else:
            yes +=1


    print(f'BEFORE OVERSAMPLING \n \tNon-frauds: {non_fraud} \n \tFauds: {fraud}')
    print(f'AFTER OVERSAMPLING \n \tNon-frauds: {no} \n \tFauds: {yes}')
    return x_tr_resample, y_tr_resample
```

Model selection

In [7]:

```
def evaluate_models(X, y, models, cv):
    f1_scores = dict()
    roc_auc_scores = dict()
    acc_scores = dict()

    for i, model in enumerate(models):
        clf_pipeline = make_pipeline(preprocessing_pipeline, model)
        # clf_pipeline = make_pipeline(model)
        results = cross_validate(clf_pipeline, X, y, cv=cv, scoring=['f1', 'accuracy', '
roc_auc'], n_jobs=-1)
        avg_f1 = np.mean(results['test_f1'])
        avg_acc = np.mean(results['test_accuracy'])
        avg_roc = np.mean(results['test_roc_auc'])

        model_name = model.__class__.__name__
        f1_scores[model_name] = avg_f1
        acc_scores[model_name] = avg_acc
        roc_auc_scores[model_name] = avg_roc
        print('{}-of-{}: {} f1={}, acc={}, roc_auc={}'.format(i+1, len(models), model_na
me, avg_f1, avg_acc, avg_roc))
    return f1_scores, acc_scores, roc_auc_scores


def visualize_scores(f1_scores, acc_scores, roc_auc_scores):
    x = np.arange(len(f1_scores))
    width = 0.3

    f1_values = list(f1_scores.values())
    acc_values = list(acc_scores.values())
    roc_values = list(roc_auc_scores.values())

    plt.figure(figsize=(20, 8)).tight_layout()
    plt.bar(x - width, f1_values, width, label='f1 score')
    plt.bar(x, acc_values, width, label='accuracy')
    plt.bar(x + width, roc_values, width, label='roc_auc')

    for index, value in enumerate(x - width / 2):
        plt.text(value, f1_values[index], '{:.3}'.format(f1_values[index]),
                 verticalalignment='bottom', horizontalalignment='center', fontsize=10)

    for index, value in enumerate(x + width / 2):
        plt.text(value, acc_values[index], '{:.3}'.format(acc_values[index]),
                 verticalalignment='bottom', horizontalalignment='center', fontsize=10)


    for index, value in enumerate(x + width / 2):
        plt.text(value, roc_values[index], '{:.3}'.format(roc_values[index]),
                 verticalalignment='bottom', horizontalalignment='center', fontsize=10)


    classifiers_names = f1_scores.keys()
    plt.xticks(x, classifiers_names, rotation=40, horizontalalignment='right', fontsize=
10)
    plt.legend()

def model_select(X, y, models, cv):
    f1_scores, acc_scores, roc_auc_scores = evaluate_models(X, y, models, cv)
    visualize_scores(f1_scores, acc_scores, roc_auc_scores)
```

Implementation

Without smote

Best model on MLP

In [30]:

```
preprocessing_pipeline = Pipeline([
    ('impoter', SimpleImputer(strategy='mean')),
    ('nomalize', MinMaxScaler())
    # ('standard', StandardScaler())
])
```

In [28]:

```
def best_model_select_MLP(x_train, y_train, x_test, y_test):
    MLP_parameters = {
            'mlpclassifier__hidden_layer_sizes': [2, 10, 2],
            'mlpclassifier__solver': ['sgd', 'adam'],
            'mlpclassifier__learning_rate': ['adaptive', 'constant'],
            'mlpclassifier__max_iter': [1000],
            'mlpclassifier__activation': ['logistic', 'tanh'],
            'mlpclassifier__alpha': [1e-5, 1e-4, 1e-3]
    }

    MLP_pipeline = make_pipeline(preprocessing_pipeline, MLPClassifier(random_state=42))

    MLP_grid_search = GridSearchCV(
        MLP_pipeline,
        param_grid=MLP_parameters,
        scoring = 'recall',
        n_jobs = -1,
        cv = 5
    )

    MLP_grid_search.fit(x_train, y_train)

    display(MLP_grid_search.best_score_)
    display(MLP_grid_search.best_params_)
    dict1 = MLP_grid_search.best_params_

    model_dict = {k.replace('mlpclassifier__',''):v for k, v in dict1.items()}

    X_train = preprocessing_pipeline.fit_transform(x_train)
    X_test = preprocessing_pipeline.transform(x_test)

    best_model_MLP = MLPClassifier(**model_dict)

    best_model_MLP.fit(X_train, y_train)
    predictions = best_model_MLP.predict(X_test)


    precision, recall, _ = precision_recall_curve(y_test, predictions)
    auc_score = auc(recall, precision)

    print("f1 score = {0:.4f}".format(f1_score(y_test, predictions)))
    print("Precision score = {0:.4f}".format(precision_score(y_test, predictions)))
    print("Recall score = {0:.4f}".format(recall_score(y_test, predictions)))
    print("ROC AUC score = {0:.4f}".format(roc_auc_score(y_test, predictions)))
    print("PR AUC score = {0:.4f}".format(auc_score))
    print("accuracy score = {0:.4f}".format(accuracy_score(y_test, predictions)))

    display(plot_confusion_matrix(best_model_MLP, X_test, y_test))

    return best_model_MLP
```

In [31]:

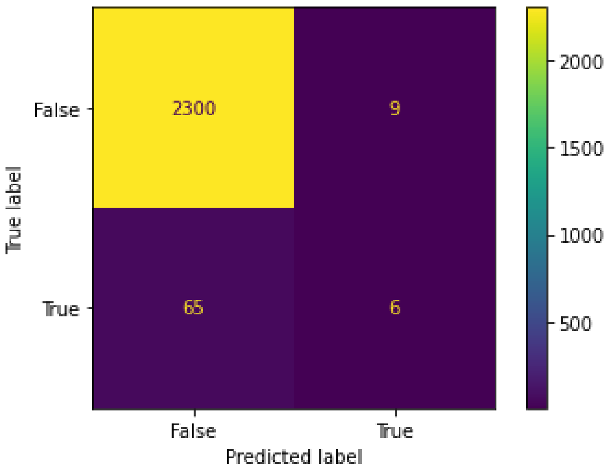
```
x, y, x_train, y_train, x_test, y_test = data_preporcess(df, cat=True)
best_model_MLP = best_model_select_MLP(x_train, y_train, x_test, y_test)
```

```
Total shape:  (7626, 25) (7626,)
```

```
Train shape:  (5516, 29) (5516,)
Test shape:  (2380, 29) (2380,)

0.10111111111111111

{'mlpclassifier__solver': 'adam',
 'mlpclassifier__max_iter': 1000,
 'mlpclassifier__learning_rate': 'constant',
 'mlpclassifier__hidden_layer_sizes': 10,
 'mlpclassifier__activation': 'tanh'}

f1 score = 0.1395
Precision score = 0.4000
Recall score = 0.0845
ROC AUC score = 0.5403
PR AUC score = 0.2559
accuracy score = 0.9689
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Fu
nction plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecate
d in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay
.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f261521c0d0>
```



With smote

In [10]:

```
preprocessing_pipeline = Pipeline([
    ('impoter', SimpleImputer(strategy='mean')),
    ('nomalize', MinMaxScaler())
    # ('standard', StandardScaler())
])
```

Best model on MLP

In [14]:

```
def best_model_select_MLP(x_train, y_train, x_test, y_test):
    MLP_parameters = {
            'mlpclassifier__hidden_layer_sizes': [2, 10, 2],
            'mlpclassifier__solver': ['sgd', 'adam'],
            'mlpclassifier__learning_rate': ['adaptive', 'constant'],
            'mlpclassifier__max_iter': [1000],
            'mlpclassifier__activation': ['logistic', 'tanh'],
            'mlpclassifier__alpha': [1e-5, 1e-4, 1e-3]
    }

    MLP_pipeline = make_pipeline(preprocessing_pipeline, MLPClassifier(random_state=42))

    MLP_grid_search = GridSearchCV(
        MLP_pipeline,
        param_grid=MLP_parameters,
        scoring = 'recall',
        n_jobs = -1,
        cv = 5
    )

    MLP_grid_search.fit(x_train, y_train)

    display(MLP_grid_search.best_score_)
    display(MLP_grid_search.best_params_)
    dict1 = MLP_grid_search.best_params_

    model_dict = {k.replace('mlpclassifier__',''):v for k, v in dict1.items()}

    X_train = preprocessing_pipeline.fit_transform(x_train)
    X_test = preprocessing_pipeline.transform(x_test)

    best_model_MLP = MLPClassifier(**model_dict)

    best_model_MLP.fit(X_train, y_train)
    predictions = best_model_MLP.predict(X_test)


    precision, recall, _ = precision_recall_curve(y_test, predictions)
    auc_score = auc(recall, precision)

    print("f1 score = {0:.4f}".format(f1_score(y_test, predictions)))
    print("Precision score = {0:.4f}".format(precision_score(y_test, predictions)))
    print("Recall score = {0:.4f}".format(recall_score(y_test, predictions)))
    print("ROC AUC score = {0:.4f}".format(roc_auc_score(y_test, predictions)))
    print("PR AUC score = {0:.4f}".format(auc_score))
    print("accuracy score = {0:.4f}".format(accuracy_score(y_test, predictions)))

    display(plot_confusion_matrix(best_model_MLP, X_test, y_test))

    return best_model_MLP
```

In [19]:

```
x, y, x_train, y_train, x_test, y_test = data_preporcess(df, cat=True)
x_train_balance, y_train_balance = smote_balance(x_train, y_train, r=1.0, k=10)
best_model_MLP = best_model_select_MLP(x_train_balance, y_train_balance, x_test, y_test)
```

```
Total shape:  (7626, 25) (7626,)
Train shape:  (5516, 29) (5516,)
Test shape:  (2380, 29) (2380,)
Shape of the training before SMOTE: ((5516, 29), (5516,))
Shape of the training after SMOTE: ((10678, 29), (10678,))
BEFORE OVERSAMPLING
  Non-frauds: 5339
  Fauds: 177
AFTER OVERSAMPLING
  Non-frauds: 5339
  Fauds: 5340
```

```
/usr/local/lib/python3.7/dist-packages/imblearn/utils/_validation.py:591: FutureWarning:
Pass sampling_strategy=1.0 as keyword args. From version 0.9 passing these as positional
arguments will result in an error
  FutureWarning,
```

```
0.8615852138903222
```

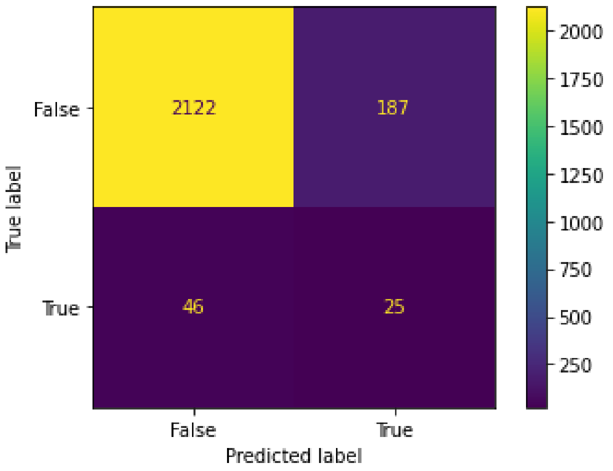
```
{'mlpclassifier__activation': 'logistic',
 'mlpclassifier__alpha': 0.0001,
 'mlpclassifier__hidden_layer_sizes': 6,
 'mlpclassifier__learning_rate': 'adaptive',
 'mlpclassifier__max_iter': 1000,
 'mlpclassifier__solver': 'adam'}
```

```
f1 score = 0.1767
Precision score = 0.1179
Recall score = 0.3521
ROC AUC score = 0.6356
```

```
PR AUC score = 0.2447
accuracy score = 0.9021
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Fu
nction plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecate
d in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay
.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f2615191390>
```


True label
False
True
False
True
Predicted label