

CIND-820 - Final project - BRFSS 2020 Survey Data

Roghayeh Taghvai (student number: 501303544)

1) Data Analysis; summarizing and visualizing data

1.1) Importing necessary libraries

```
In [ ]: !pip install jupyter notebook
!pip install scikit-learn
!pip install seaborn
!pip install pandas
!pip install matplotlib
!pip install xgboost
```

```
In [2]: import pandas as pd
import numpy as np
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns
```

1.2) Upload Dataset

```
In [23]: df = pd.read_csv("E:\\Chang_school_courses\\CIND820\\dataset\\BRFSS2020.csv")
df.shape
```

Out[23]: (401958, 279)

1.3) Select Important Features according to the papers

```
In [24]: column_name=[
    '_SEX',
    '_BMI5CAT',
    '_AGE80',
    '_SMOKER3',
    'EXERANY2',
    '_TOTINDA',
    'SLEPTIM1',
    'DIABETE4',
    'CVDINFR4',
    'CVDCRHD4',
    '_URBSTAT',
    '_RFBING5',
    '_RFDRHV7',
```

```

    'CVDSTRK3',
    '_PHYS14D',
    'CHCKDNY2' ,
    '_LTASTH1',
    'GENHLTH' ,
    'DIFFWALK',
    'CHCSCNCR' ,
    'ADDEPEV3'
    #'ECIGARET'

]

df=df[column_name]
df.rename(columns={
    '_SEX': 'sex',
    '_BMI5CAT': 'bmi',
    '_AGE80': 'age_groups',
    '_SMOKER3': 'smoke_status',
    'EXERANY2': 'exercise1',
    '_TOTINDA': 'exercise2',
    'SLEPTIM1': 'sleep',
    'DIABETE4': 'diabete',
    'CVDINFR4': 'heart_disease1',
    'CVDCRHD4': 'heart_disease2',
    '_URBSTAT': 'area' ,
    '_RFBING5': 'drinker1',
    '_RFDRHV7': 'drinker2',
    'CVDSTRK3': 'stroke',
    '_PHYS14D' : 'Physical_Health',
    'CHCKDNY2' : 'Kidney_Disease',
    '_LTASTH1' : 'Asthma',
    'GENHLTH' : 'General_health',
    'DIFFWALK' : 'Difficulty_Walking',
    'CHCSCNCR' : 'Skin_Cancer',
    'ADDEPEV3' : 'depressive_disorder'
    #'ECIGARET' : 'Ecigarette'

}, inplace=True)

```

In [7]: df.head()

Out[7]:

	sex	bmi	age_groups	smoke_status	exercise1	exercise2	sleep	diabete	heart_disease
0	2	1.0	56	1	1.0	1	5.0	1.0	2
1	2	3.0	65	9	1.0	1	7.0	3.0	2
2	2	NaN	65	4	1.0	1	7.0	3.0	2
3	2	NaN	80	4	2.0	2	6.0	3.0	2
4	2	2.0	80	4	1.0	1	7.0	3.0	2

5 rows × 21 columns

In [8]: `df.info()`

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 401958 entries, 0 to 401957
Data columns (total 21 columns):
#   Column                Non-Null Count  Dtype
---  -
0   sex                    401958 non-null  int64
1   bmi                    360601 non-null  float64
2   age_groups             401958 non-null  int64
3   smoke_status           401958 non-null  int64
4   exercise1              401955 non-null  float64
5   exercise2              401958 non-null  int64
6   sleep                  401955 non-null  float64
7   diabete                401952 non-null  float64
8   heart_disease1         401952 non-null  float64
9   heart_disease2         401955 non-null  float64
10  area                   394831 non-null  float64
11  drinker1               401958 non-null  int64
12  drinker2               401958 non-null  int64
13  stroke                 401955 non-null  float64
14  Physical_Health        401958 non-null  int64
15  Kidney_Disease         401952 non-null  float64
16  Asthma                 401958 non-null  int64
17  General_health         401950 non-null  float64
18  Difficulty_Walking     386678 non-null  float64
19  Skin_Cancer            401955 non-null  float64
20  depressive_disorder    401952 non-null  float64
dtypes: float64(13), int64(8)
memory usage: 64.4 MB

```

In [9]: `df.describe()`

Out[9]:

	sex	bmi	age_groups	smoke_status	exercise1	ex
count	401958.000000	360601.000000	401958.000000	401958.000000	401955.000000	401958.
mean	1.542412	2.981792	54.429221	3.648065	1.248593	1.
std	0.498199	0.831831	17.666723	1.563233	0.502215	0.
min	1.000000	1.000000	18.000000	1.000000	1.000000	1.
25%	1.000000	2.000000	40.000000	3.000000	1.000000	1.
50%	2.000000	3.000000	57.000000	4.000000	1.000000	1.
75%	2.000000	4.000000	69.000000	4.000000	1.000000	1.
max	2.000000	4.000000	80.000000	9.000000	9.000000	9.

8 rows × 21 columns



2) Data preparation, including selecting, preprocessing, and transforming data

2.1) Dropping Duplicates

```
In [25]: duplicates = df.duplicated().sum()
print(f"Number of duplicate rows: {duplicates}")
df = df.drop_duplicates()
df.shape
```

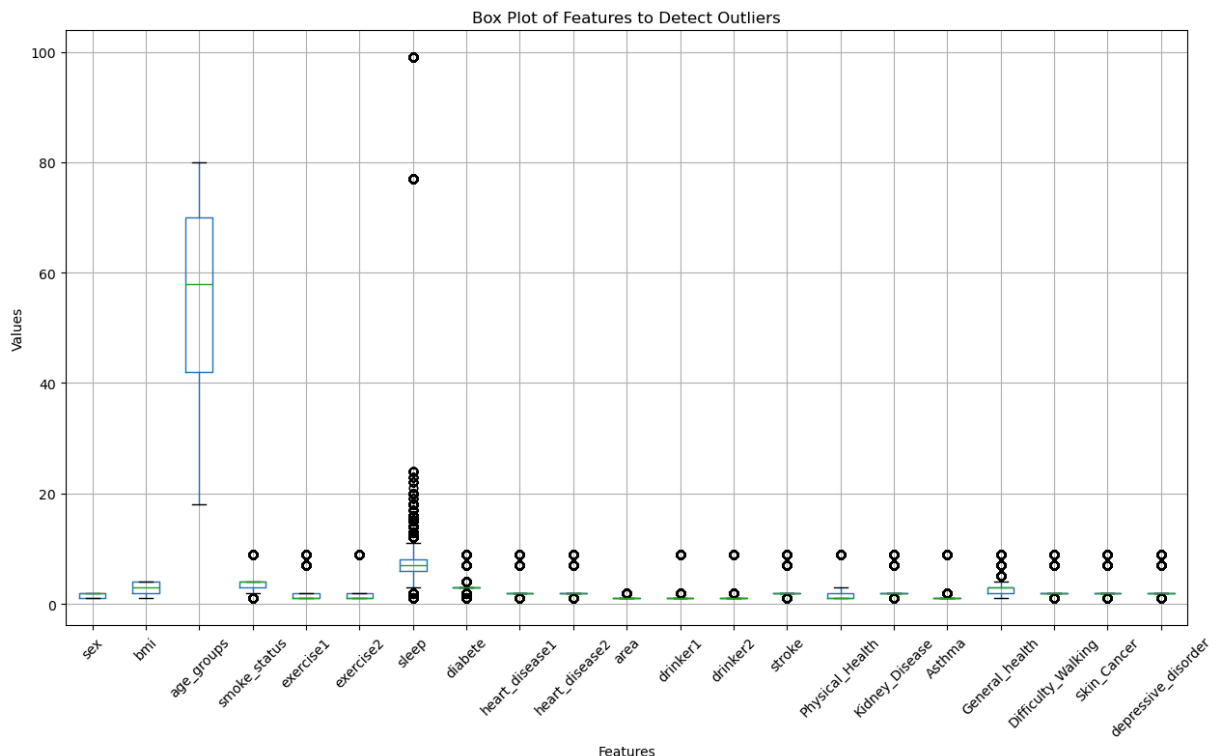
Number of duplicate rows: 113832

Out[25]: (288126, 21)

2.2) Detecting and Removeing the irrelevant values and outliers

```
In [11]: columns= df.columns

plt.figure(figsize=(15, 8))
df[columns].boxplot(rot=45) # Rotate x Labels for better readability
plt.title("Box Plot of Features to Detect Outliers")
plt.xlabel("Features")
plt.ylabel("Values")
plt.show()
```



Result: Most of the features have outliers and should be removed.

In [26]: *# Remove irrelevant values.*

```
df = df.drop(df[df.bmi == -1].index)
df = df.drop(df[df.bmi > 4].index)
df = df.drop(df[df.smoke_status > 4].index)
df = df.drop(df[df.heart_disease1 > 2].index)
df = df.drop(df[df.heart_disease2 > 2].index)
df = df.drop(df[df.exercise1 > 2].index)
df = df.drop(df[df.exercise2 > 2].index)
df = df.drop(df[df.sleep > 20].index)
df = df.drop(df[df.diabete > 4].index)
df = df.drop(df[df.area == -1].index)
df = df.drop(df[df.drinker1 > 2].index)
df = df.drop(df[df.drinker2 > 2].index)
df = df.drop(df[df.stroke > 2].index)
#df = df.drop(df[df.Ecigarette > 2].index)
df = df.drop(df[df.Physical_Health > 3].index)
#df = df.drop(df[df.Mental_Health > 3].index)
df = df.drop(df[df.Kidney_Disease > 2].index)
df = df.drop(df[df.Asthma > 2].index)
df = df.drop(df[df.General_health > 5].index)
df = df.drop(df[df.Difficulty_Walking > 2].index)
df = df.drop(df[df.Skin_Cancer > 2].index)
df = df.drop(df[df.depressive_disorder>2].index)

# Replace 2 to 0, then false == 0
df.heart_disease1 = df.heart_disease1.replace(2, 0)
df.heart_disease2 = df.heart_disease2.replace(2, 0)

df['heart_disease'] = df['heart_disease1'] + df['heart_disease2']
# Replace 2 to 1, then heart problem.
df['heart_disease'] = df['heart_disease'].replace(2,1)

# Replace 2 to 0, then false == 0
df.drinker1 = df.drinker1.replace(2, 0)
df.drinker2 = df.drinker2.replace(2, 0)

df['drinker'] = df['drinker1'] + df['drinker2']
# Replace 2 to 1, then exercise.
df['drinker'] = df['drinker'].replace(2,1)

# Replace 2 to 0, then false == 0
df.exercise1 = df.exercise1.replace(2, 0)
df.exercise2 = df.exercise2.replace(2, 0)

df['exercise'] = df['exercise1'] + df['exercise2']
# Replace 2 to 1, then exercise.
df['exercise'] = df['exercise'].replace(2,1)

# valores 2 representam "não", substituindo por 0 para facilitar na plotagem.
df['stroke'] = df['stroke'].replace(2,0)
```

In [27]: df.drop(columns=['heart_disease1', 'heart_disease2', 'exercise1', 'exercise2', 'drin

2.3) Deleting and Handling NAs

```
In [14]: print(df.isna().sum())
         #df=df.dropna()
         df.shape
```

```
sex          0
bmi          17832
age_groups   0
smoke_status 0
sleep        0
diabete      0
area         6275
stroke       0
Physical_Health 0
Kidney_Disease 0
Asthma       0
General_health 3
Difficulty_Walking 1
Skin_Cancer  0
depressive_disorder 0
heart_disease 2
drinker      0
exercise     0
dtype: int64
```

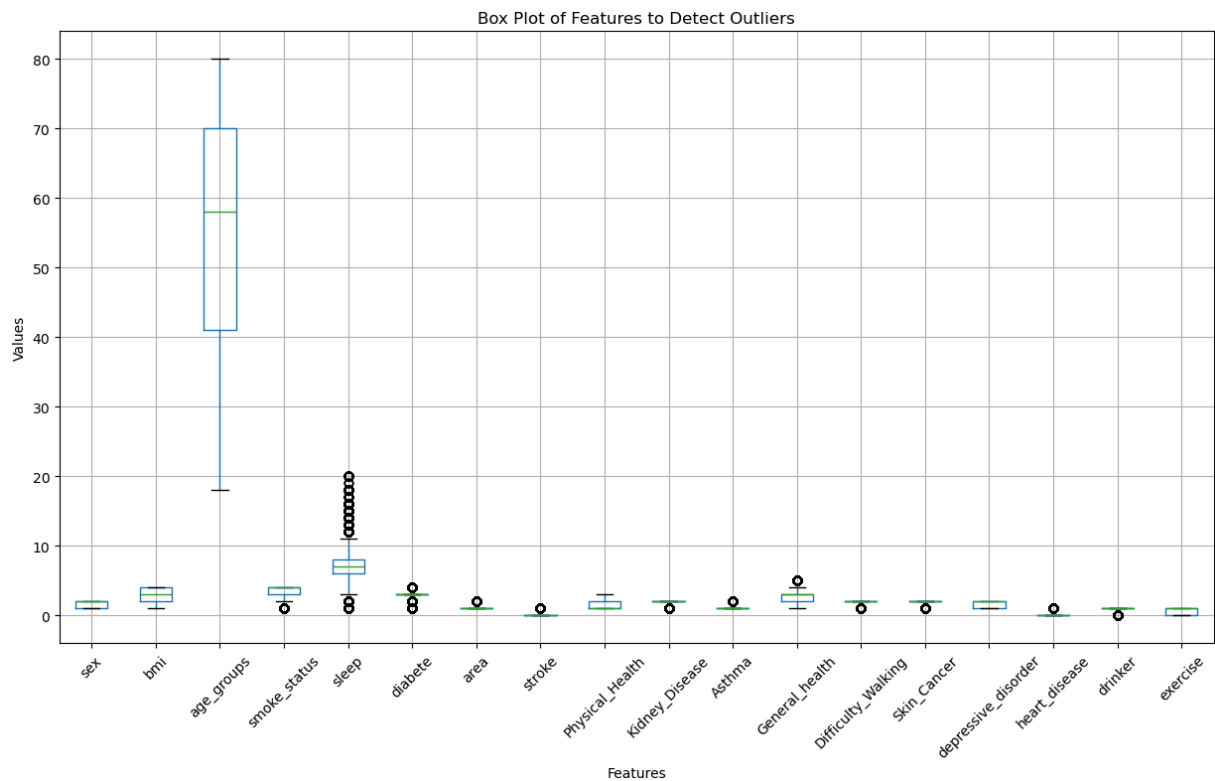
Out[14]: (236242, 18)

```
In [28]: for factor in df.columns:
         df[factor] = df[factor].fillna(df[factor].mode()[0])
         print(df.isna().sum())
```

```
sex          0
bmi          0
age_groups   0
smoke_status 0
sleep        0
diabete      0
area         0
stroke       0
Physical_Health 0
Kidney_Disease 0
Asthma       0
General_health 0
Difficulty_Walking 0
Skin_Cancer  0
depressive_disorder 0
heart_disease 0
drinker      0
exercise     0
dtype: int64
```

```
In [16]: #columns= df.columns
         existing_features = [col for col in df.columns]
```

```
plt.figure(figsize=(15, 8))
df[df.columns].boxplot(rot=45) # Rotate x Labels for better readability
plt.title("Box Plot of Features to Detect Outliers")
plt.xlabel("Features")
plt.ylabel("Values")
plt.show()
```



```
In [29]: df = df.astype(int)
```

2.4) Feature Engineering

Corrolation

```
In [18]: corr_matrix=df.corr()
corr_matrix["heart_disease"].sort_values(ascending=False).head(30)
```

```
Out[18]: heart_disease      1.000000
age_groups      0.253184
General_health  0.207558
stroke          0.177203
Physical_Health 0.108700
drinker         0.053076
bmi             0.025073
sleep           0.014763
Asthma          0.008765
area            0.005558
depressive_disorder 0.003609
smoke_status    -0.025588
exercise        -0.059094
Skin_Cancer     -0.079174
sex             -0.087834
Kidney_Disease  -0.125509
diabete         -0.148064
Difficulty_Walking -0.165217
Name: heart_disease, dtype: float64
```

Result: The result of applying the correlation function to data set shows that "heart_disease" and "General_health" are the most positively correlated features and "Difficulty_walking" and "Diabete" are the most negatively correlated ones to the target feature.

Backward Elimination

```
In [30]: column = df.columns.drop("heart_disease")
X = df[column] # Features
y = df['heart_disease']
#X_norm = pd.DataFrame(X, columns=df.columns.drop("heart_disease"))
#df_norm['heart_disease'] = y.values
```

```
In [31]: import statsmodels.api as sm

def backward_elimination(data, target, significance_level=0.05):
    variables = data.columns.tolist()
    while len(variables) > 0:
        X_with_const = sm.add_constant(data[variables])
        model = sm.OLS(target, X_with_const).fit()
        max_p_value = model.pvalues.iloc[1:].max()
        if max_p_value > significance_level:
            excluded_var = model.pvalues.iloc[1:].idxmax()
            print(f"Removing {excluded_var} with p-value {max_p_value}")
            variables.remove(excluded_var)
        else:
            break
    return model

# Apply backward elimination
final_model_backward = backward_elimination(X, y)
print("\nFinal Model Summary (Backward Elimination):\n", final_model_backward.summa
```


Removing sleep with p-value 0.9782401943728702

Removing depressive_disorder with p-value 0.7531760650479588

Removing area with p-value 0.16299676631251564

Final Model Summary (Backward Elimination):

OLS Regression Results

```
=====
Dep. Variable:      heart_disease    R-squared:      0.126
Model:              OLS              Adj. R-squared:  0.126
Method:             Least Squares    F-statistic:    2424.
Date:               Sat, 15 Mar 2025  Prob (F-statistic): 0.00
Time:               18:47:34         Log-Likelihood:  -49913.
No. Observations:   236242          AIC:             9.986e+04
Df Residuals:       236227          BIC:             1.000e+05
Df Model:           14
Covariance Type:    nonrobust
=====
```

```
=====
==
               coef      std err          t      P>|t|      [0.025      0.97
5]
-----
--
const          0.2244      0.010      21.977      0.000      0.204      0.2
44
sex           -0.0655      0.001     -52.408      0.000     -0.068     -0.0
63
bmi           -0.0025      0.001      -3.329      0.001     -0.004     -0.0
01
age_groups     0.0034    3.94e-05     85.666      0.000      0.003      0.0
03
smoke_status   -0.0035      0.001      -5.612      0.000     -0.005     -0.0
02
diabete       -0.0242      0.001     -29.296      0.000     -0.026     -0.0
23
stroke         0.1642      0.003     58.352      0.000      0.159      0.1
70
Physical_Health 0.0047      0.001      4.899      0.000      0.003      0.0
07
Kidney_Disease -0.0847      0.003     -29.933      0.000     -0.090     -0.0
79
Asthma         0.0108      0.002      6.613      0.000      0.008      0.0
14
General_health 0.0344      0.001     49.614      0.000      0.033      0.0
36
Difficuly_Walking -0.0319      0.002     -17.610      0.000     -0.035     -0.0
28
Skin_Cancer    -0.0161      0.002      -7.944      0.000     -0.020     -0.0
12
drinker        0.0208      0.003      8.022      0.000      0.016      0.0
26
exercise       0.0062      0.001      4.416      0.000      0.003      0.0
09
=====
```

```
Omnibus:      84498.064    Durbin-Watson:      1.989
Prob(Omnibus): 0.000    Jarque-Bera (JB):    238386.583
Skew:          1.959    Prob(JB):            0.00
```

Kurtosis: 5.977 Cond. No. 986.
=====

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
In [ ]: df = df.drop(['depressive_disorder'],axis=1)
df = df.drop(['sleep'],axis=1)
df = df.drop(['area'],axis=1)
```

2.5) Handling unbalanced dataset

```
In [35]: df['heart_disease'].value_counts(normalize=True) * 100
```

```
Out[35]: heart_disease
0      88.448286
1      11.551714
Name: proportion, dtype: float64
```

The ratio of the value of the target feature is 88% (0) to 12%(1). This huge unbalancing should be addressed. We considered 3 different method and selected one of them that give us better results regarding accuracy and

```
In [ ]: from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=42)
X_resa, y_resa = rus.fit_resample(X, y)

# Convert back to DataFrame
df_resampled = pd.DataFrame(X_resa, columns=X.columns)
df_resampled['heart_disease'] = y_resa.values
```

```
In [36]: from imblearn.over_sampling import SMOTE
column = df.columns.drop("heart_disease")
X = df[column] # Features
y = df['heart_disease']
#X = df_norm.drop(columns=['heart_disease']) # Features
#y = df_norm['heart_disease'] # Target column

smote = SMOTE(random_state=42)
X_resa, y_resa = smote.fit_resample(X, y)
df_resampled = pd.DataFrame(X_resa, columns=X.columns)
df_resampled['heart_disease'] = y_resa.values
```

```
In [ ]: from imblearn.pipeline import Pipeline
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE

# Define a pipeline
sampling_pipeline = Pipeline([
    ('smote', SMOTE(sampling_strategy=0.3, random_state=42)), # Increase 1s to 30%
    ('under', RandomUnderSampler(sampling_strategy="auto", random_state=42)) # Redu
])
```

```
#X = df_norm.drop(columns=['heart_disease']) # Features
#y = df_norm['heart_disease'] # Target column

X_resa, y_resa = sampling_pipeline.fit_resample(X, y)
df_resampled = pd.DataFrame(X_resa, columns=X.columns)
df_resampled['heart_disease'] = y_resa.values
```

The result shows that the SMOTE method is the best to apply balancing to the data.

```
In [37]: y_resa.value_counts(normalize=True) * 100
```

```
Out[37]: heart_disease
0      50.0
1      50.0
Name: proportion, dtype: float64
```

Now the target values are balanced.

3) Models evaluation, including testing options, exploring algorithms, and reporting results

Dividing data into test and train data.

```
In [38]: from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X_resa,y_resa, test_size=0.2, rand
```

3.1) Applying Decision Tree algorithm

```
In [39]: from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import mean_squared_error
model=DecisionTreeClassifier()
model.fit(X_train,y_train)
```

```
Out[39]: DecisionTreeClassifier
DecisionTreeClassifier()
```

```
In [40]: from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score

y_pred_train=model.predict(X_train)
y_pred_test=model.predict(X_test)

accuracy_train = accuracy_score(y_train, y_pred_train) * 100
accuracy_test = accuracy_score(y_test, y_pred_test) * 100 # Convert to %
recall = recall_score(y_test, y_pred_test) * 100 # Sensitivity / True Positive Rat
precision = precision_score(y_test, y_pred_test) * 100
f1 = f1_score(y_test, y_pred_test) * 100

sensitivity = recall_score(y_test, y_pred_test, pos_label=1)* 100 # Sensitivity for
```

```

specificity = recall_score(y_test, y_pred_test, pos_label=0)* 100 # Specificity fo

# Print results
print(f"Accuracy train: {accuracy_train:.2f}%")
print(f"Accuracy test: {accuracy_test:.2f}%")
print(f"Recall: {recall:.2f}%")
print(f"sensitivity: {sensitivity:.2f}%")
print(f"specificity: {specificity:.2f}%")
print(f"Precision: {precision:.2f}%")
print(f"F1-Score: {f1:.2f}%")
display = ConfusionMatrixDisplay.from_estimator(model, X_test, y_test, cmap=plt.cm.

```

Accuracy train: 89.92%

Accuracy test: 82.85%

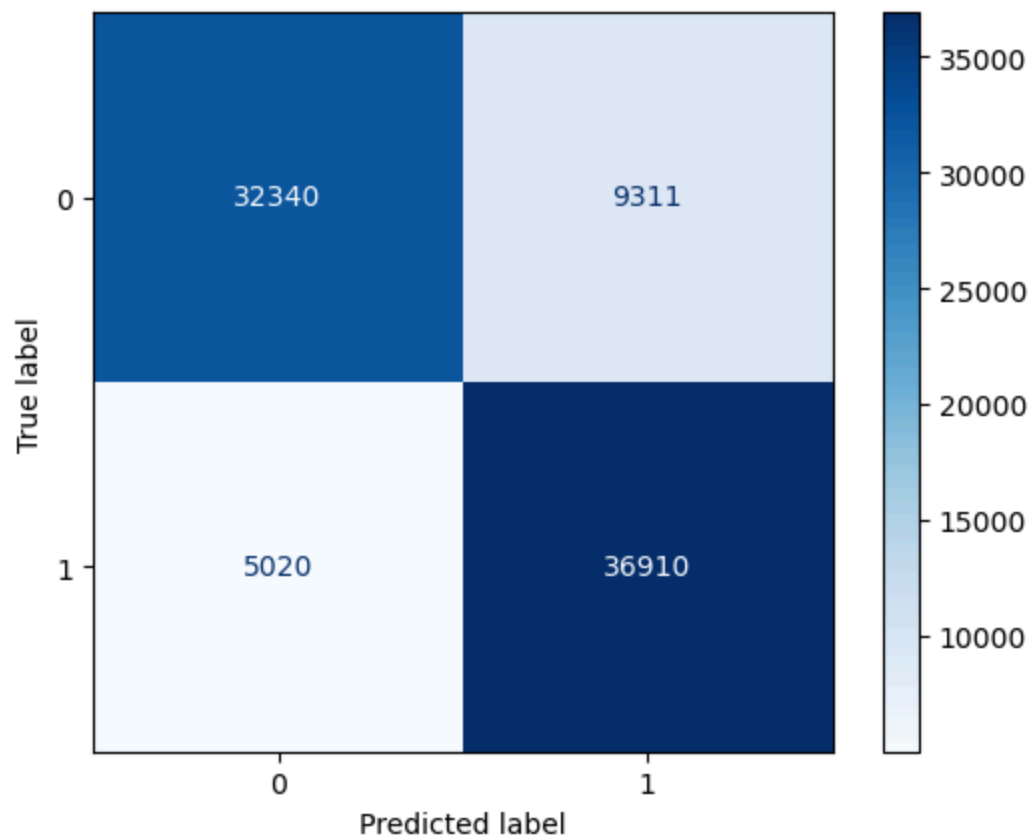
Recall: 88.03%

sensitivity: 88.03%

specificity: 77.65%

Precision: 79.86%

F1-Score: 83.74%



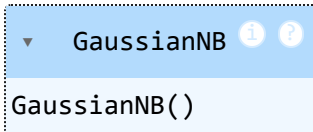
3.2) Applying Gaussian-Naive Bays algorithm

```

In [42]: from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X_train,y_train)

```

Out[42]:



In [43]:

```

y_pred_train=model.predict(X_train)
y_pred_test=model.predict(X_test)

accuracy_train = accuracy_score(y_train, y_pred_train) * 100
accuracy_test = accuracy_score(y_test, y_pred_test) * 100 # Convert to %
recall = recall_score(y_test, y_pred_test) * 100 # Sensitivity / True Positive Rat
precision = precision_score(y_test, y_pred_test) * 100
f1 = f1_score(y_test, y_pred_test) * 100

sensitivity = recall_score(y_test, y_pred_test, pos_label=1)* 100 # Sensitivity for

specificity = recall_score(y_test, y_pred_test, pos_label=0)* 100 # Specificity fo

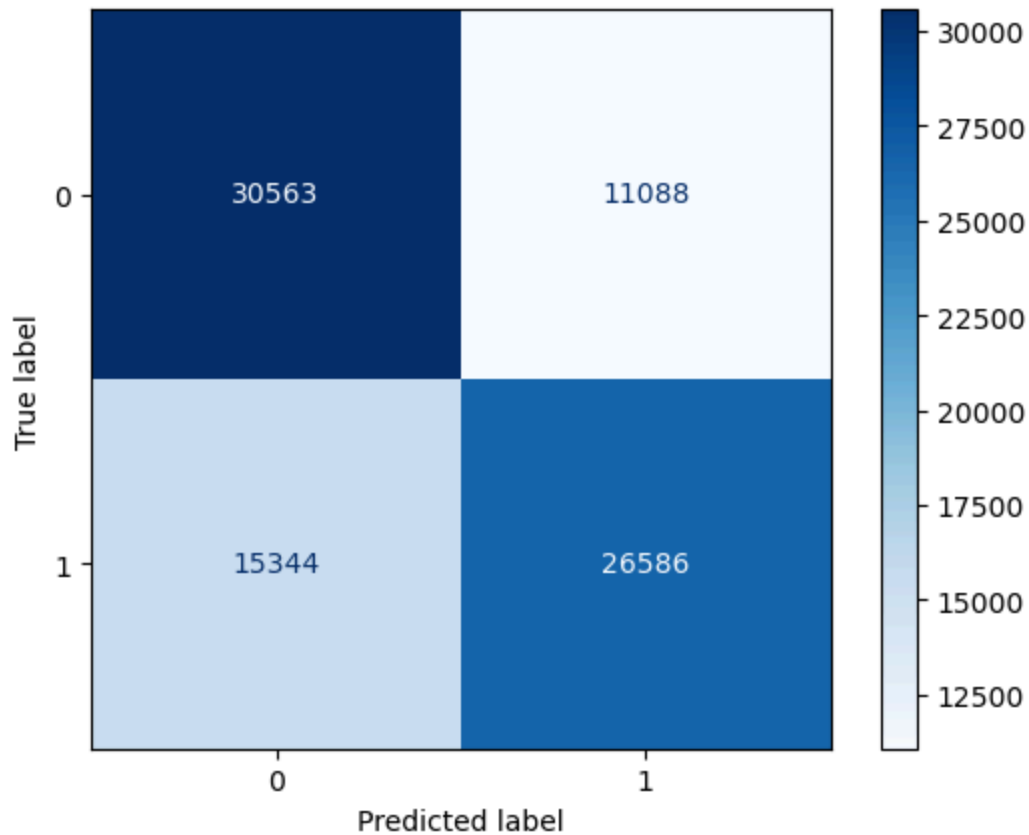
# Print results
print(f"Accuracy train: {accuracy_train:.2f}%")
print(f"Accuracy test: {accuracy_test:.2f}%")
print(f"Recall: {recall:.2f}%")
print(f"sensitivity: {sensitivity:.2f}%")
print(f"specificity: {specificity:.2f}%")
print(f"Precision: {precision:.2f}%")
print(f"F1-Score: {f1:.2f}%")
display = ConfusionMatrixDisplay.from_estimator(model, X_test, y_test, cmap=plt.cm.

```

```

Accuracy train: 68.75%
Accuracy test: 68.38%
Recall: 63.41%
sensitivity: 63.41%
specificity: 73.38%
Precision: 70.57%
F1-Score: 66.80%

```



3.3) Applying Bernuli-Naive Bays algorithm

In [48]: `from sklearn.naive_bayes import BernoulliNB`

```
model = BernoulliNB()
model.fit(X_train, y_train)
```

Out[48]:

▼ BernoulliNB ⓘ ?
BernoulliNB()

```
In [49]: y_pred_train=model.predict(X_train)
y_pred_test=model.predict(X_test)

accuracy_train = accuracy_score(y_train, y_pred_train) * 100
accuracy_test = accuracy_score(y_test, y_pred_test) * 100 # Convert to %
recall = recall_score(y_test, y_pred_test) * 100 # Sensitivity / True Positive Rat
precision = precision_score(y_test, y_pred_test) * 100
f1 = f1_score(y_test, y_pred_test) * 100

sensitivity = recall_score(y_test, y_pred_test, pos_label=1)* 100 # Sensitivity for
specificity = recall_score(y_test, y_pred_test, pos_label=0)* 100 # Specificity fo

# Print results
print(f"Accuracy train: {accuracy_train:.2f}%")
print(f"Accuracy test: {accuracy_test:.2f}%")
```

```

print(f"Recall: {recall:.2f}%")
print(f"sensitivity: {sensitivity:.2f}%")
print(f"specificity: {specificity:.2f}%")
print(f"Precision: {precision:.2f}%")
print(f"F1-Score: {f1:.2f}%")
display = ConfusionMatrixDisplay.from_estimator(model, X_test, y_test, cmap=plt.cm.

```

Accuracy train: 57.39%

Accuracy test: 57.20%

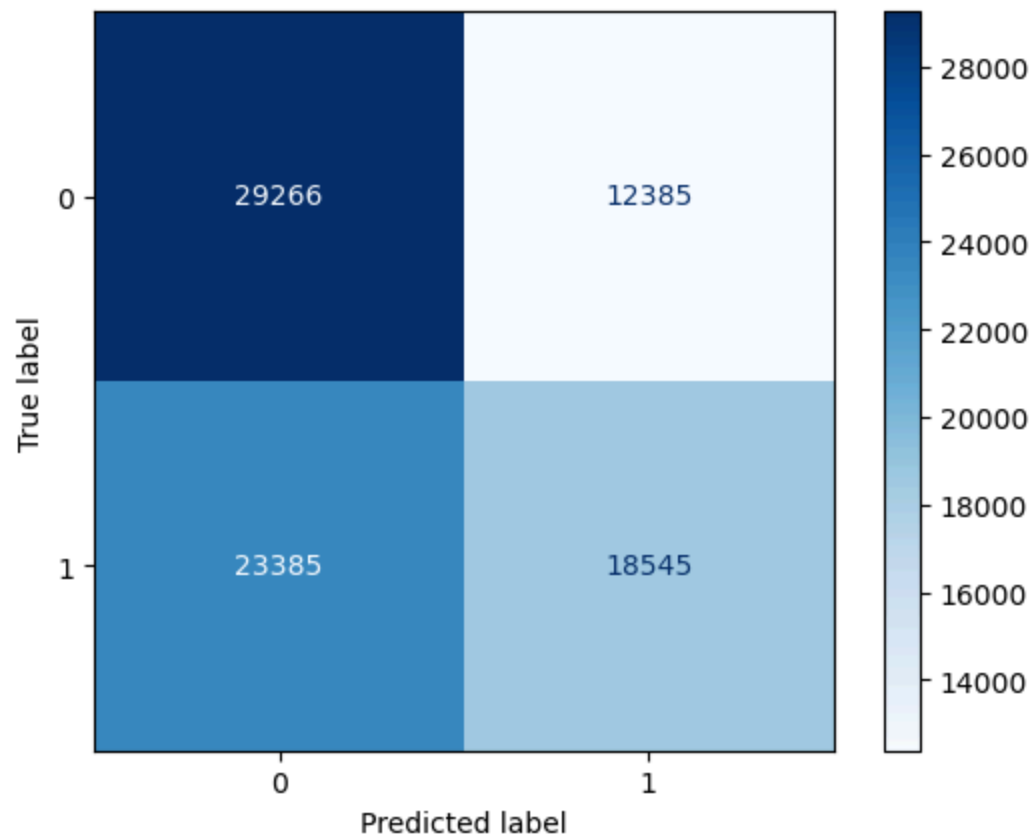
Recall: 44.23%

sensitivity: 44.23%

specificity: 70.26%

Precision: 59.96%

F1-Score: 50.91%



3.4) Applying XGB-Classifier algorithm

```

In [51]: from xgboost import XGBClassifier

model = XGBClassifier(scale_pos_weight=9) # Ratio of majority/minority class
model.fit(X_train, y_train)

```

Out[51]:

```

XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
               colsample_bylevel=None, colsample_bynode=None,
               colsample_bytree=None, device=None, early_stopping_rounds=
None,
               enable_categorical=False, eval_metric=None, feature_types=
None,
               gamma=None, grow_policy=None, importance_type=None,
               interaction_constraints=None, learning_rate=None, max_bin=
None,

```

In [52]:

```

y_pred_train=model.predict(X_train)
y_pred_test=model.predict(X_test)

accuracy_train = accuracy_score(y_train, y_pred_train) * 100
accuracy_test = accuracy_score(y_test, y_pred_test) * 100 # Convert to %
recall = recall_score(y_test, y_pred_test) * 100 # Sensitivity / True Positive Rat
precision = precision_score(y_test, y_pred_test) * 100
f1 = f1_score(y_test, y_pred_test) * 100

sensitivity = recall_score(y_test, y_pred_test, pos_label=1)* 100 # Sensitivity for
specificity = recall_score(y_test, y_pred_test, pos_label=0)* 100 # Specificity fo

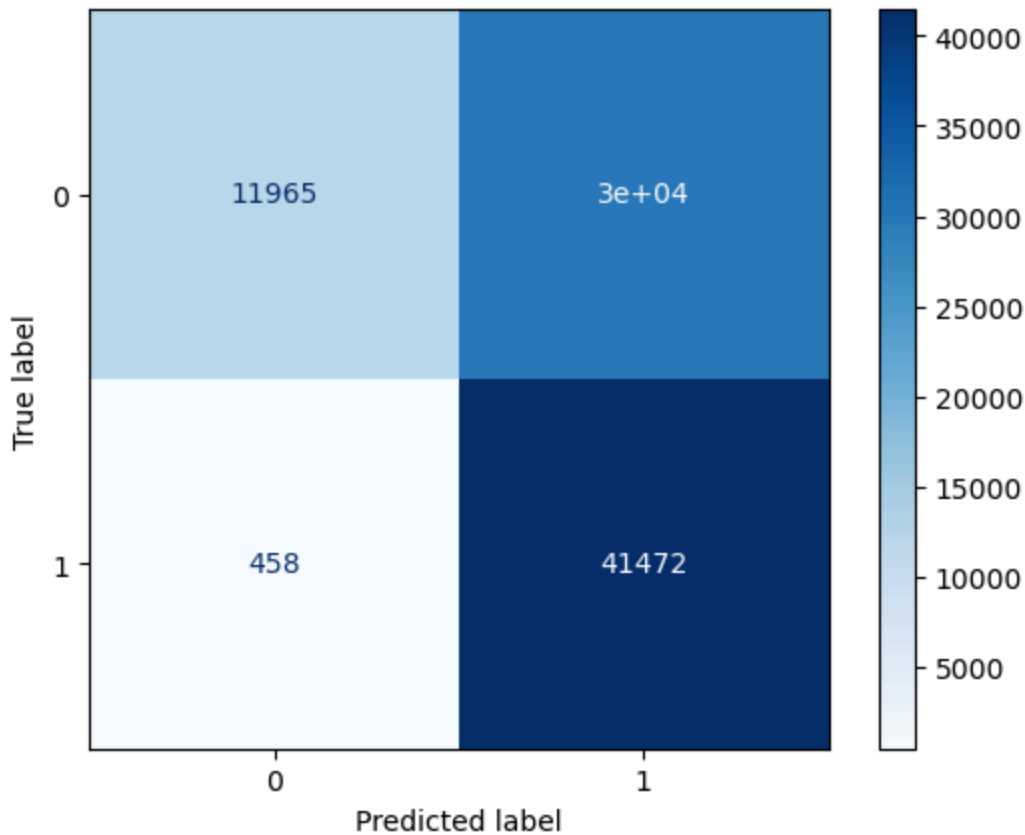
# Print results
print(f"Accuracy train: {accuracy_train:.2f}%")
print(f"Accuracy test: {accuracy_test:.2f}%")
print(f"Recall: {recall:.2f}%")
print(f"sensitivity: {sensitivity:.2f}%")
print(f"specificity: {specificity:.2f}%")
print(f"Precision: {precision:.2f}%")
print(f"F1-Score: {f1:.2f}%")
display = ConfusionMatrixDisplay.from_estimator(model, X_test, y_test, cmap=plt.cm.

```

```

Accuracy train: 64.14%
Accuracy test: 63.93%
Recall: 98.91%
sensitivity: 98.91%
specificity: 28.73%
Precision: 58.28%
F1-Score: 73.34%

```

3.5) Applying Random Forest algorithm

```
In [54]: from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=100, class_weight="balanced", random_st
model.fit(X_train, y_train)
```

```
Out[54]: ▼ RandomForestClassifier
RandomForestClassifier(class_weight='balanced', random_state=42)
```

```
In [55]: y_pred_train=model.predict(X_train)
y_pred_test=model.predict(X_test)

accuracy_train = accuracy_score(y_train, y_pred_train) * 100
accuracy_test = accuracy_score(y_test, y_pred_test) * 100 # Convert to %
recall = recall_score(y_test, y_pred_test) * 100 # Sensitivity / True Positive Rat
precision = precision_score(y_test, y_pred_test) * 100
f1 = f1_score(y_test, y_pred_test) * 100

sensitivity = recall_score(y_test, y_pred_test, pos_label=1)* 100 # Sensitivity for
specificity = recall_score(y_test, y_pred_test, pos_label=0)* 100 # Specificity fo

# Print results
print(f"Accuracy train: {accuracy_train:.2f}%")
print(f"Accuracy test: {accuracy_test:.2f}%")
print(f"Recall: {recall:.2f}%")
```

```
print(f"sensitivity: {sensitivity:.2f}%")
print(f"specificity: {specificity:.2f}%")
print(f"Precision: {precision:.2f}%")
print(f"F1-Score: {f1:.2f}%")
display = ConfusionMatrixDisplay.from_estimator(model, X_test, y_test, cmap=plt.cm.
```

Accuracy train: 89.92%

Accuracy test: 83.78%

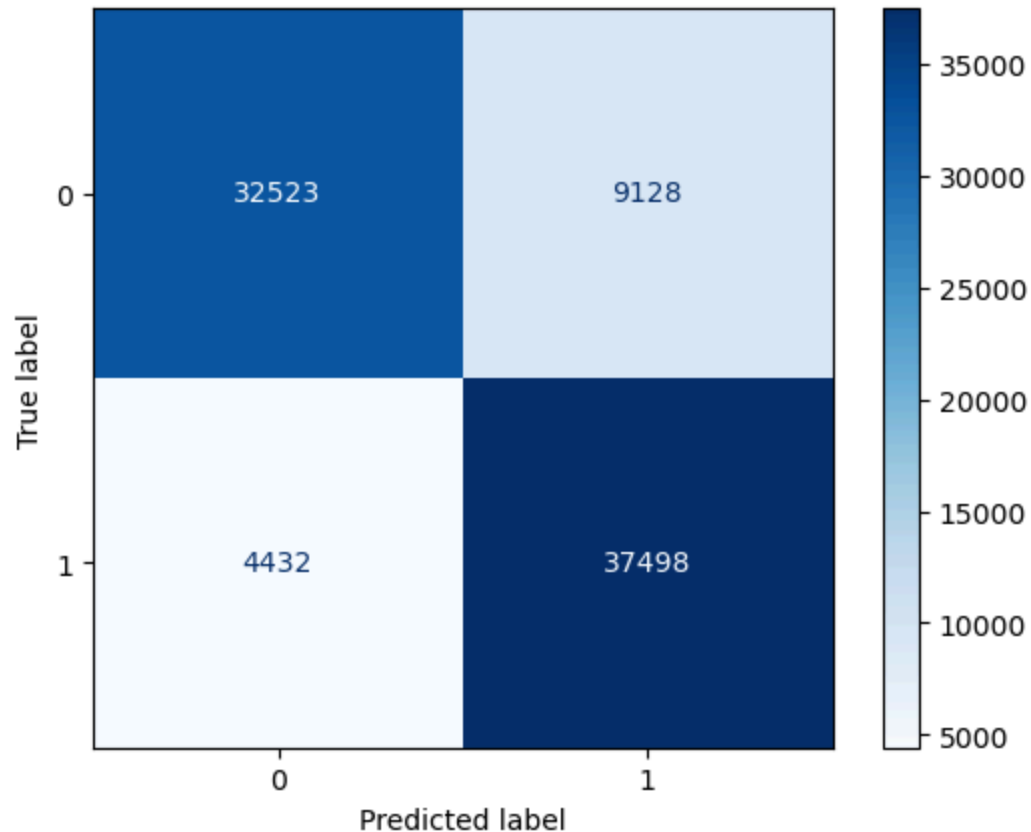
Recall: 89.43%

sensitivity: 89.43%

specificity: 78.08%

Precision: 80.42%

F1-Score: 84.69%



3.6) Applying Logistic Regression algorithm

```
In [58]: from sklearn.linear_model import LogisticRegression
model = LogisticRegression(class_weight="balanced", random_state=42, max_iter=500)
model.fit(X_train, y_train)
```

```
Out[58]: LogisticRegression
LogisticRegression(class_weight='balanced', max_iter=500, random_state=42)
```

```
In [59]: y_pred_train=model.predict(X_train)
y_pred_test=model.predict(X_test)

accuracy_train = accuracy_score(y_train, y_pred_train) * 100
```

```

accuracy_test = accuracy_score(y_test, y_pred_test) * 100 # Convert to %
recall = recall_score(y_test, y_pred_test) * 100 # Sensitivity / True Positive Rat
precision = precision_score(y_test, y_pred_test) * 100
f1 = f1_score(y_test, y_pred_test) * 100

sensitivity = recall_score(y_test, y_pred_test, pos_label=1)* 100 # Sensitivity for
specificity = recall_score(y_test, y_pred_test, pos_label=0)* 100 # Specificity fo

# Print results
print(f"Accuracy train: {accuracy_train:.2f}%")
print(f"Accuracy test: {accuracy_test:.2f}%")
print(f"Recall: {recall:.2f}%")
print(f"sensitivity: {sensitivity:.2f}%")
print(f"specificity: {specificity:.2f}%")
print(f"Precision: {precision:.2f}%")
print(f"F1-Score: {f1:.2f}%")
display = ConfusionMatrixDisplay.from_estimator(model, X_test, y_test, cmap=plt.cm.

```

Accuracy train: 73.85%

Accuracy test: 73.41%

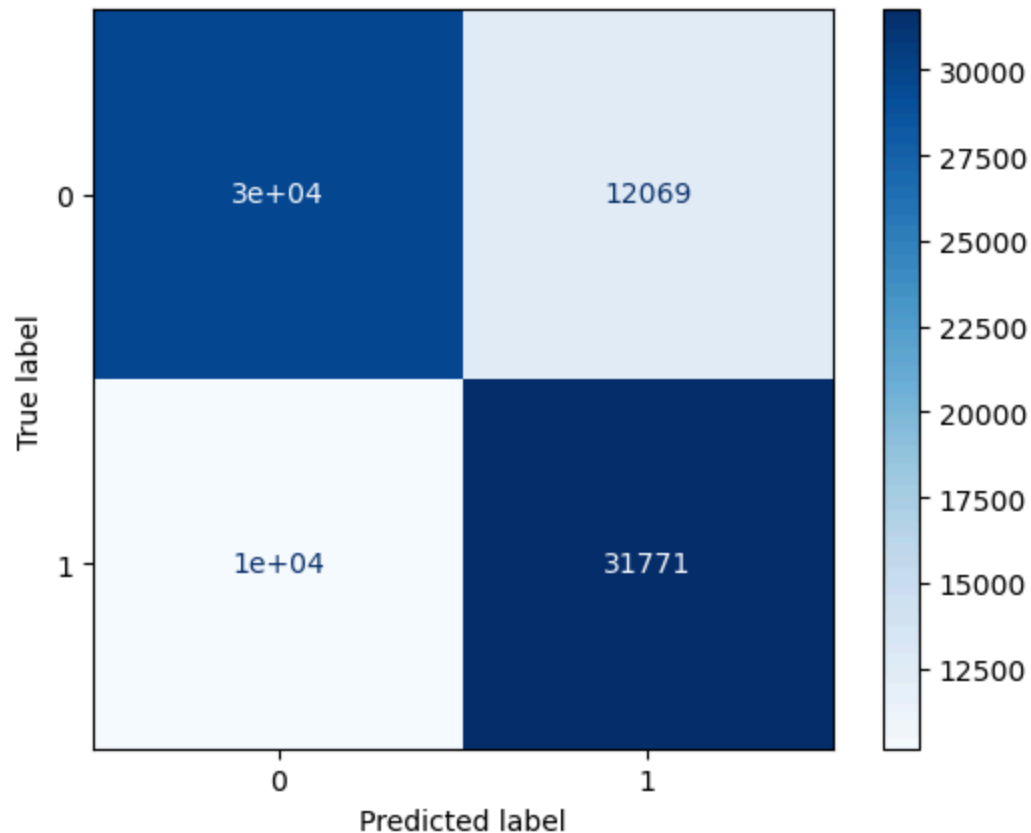
Recall: 75.77%

sensitivity: 75.77%

specificity: 71.02%

Precision: 72.47%

F1-Score: 74.08%



3.7) Applying KNN algorithm

```
In [60]: from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=4)
model.fit(X_train, y_train)
```

```
Out[60]: KNeighborsClassifier
KNeighborsClassifier(n_neighbors=4)
```

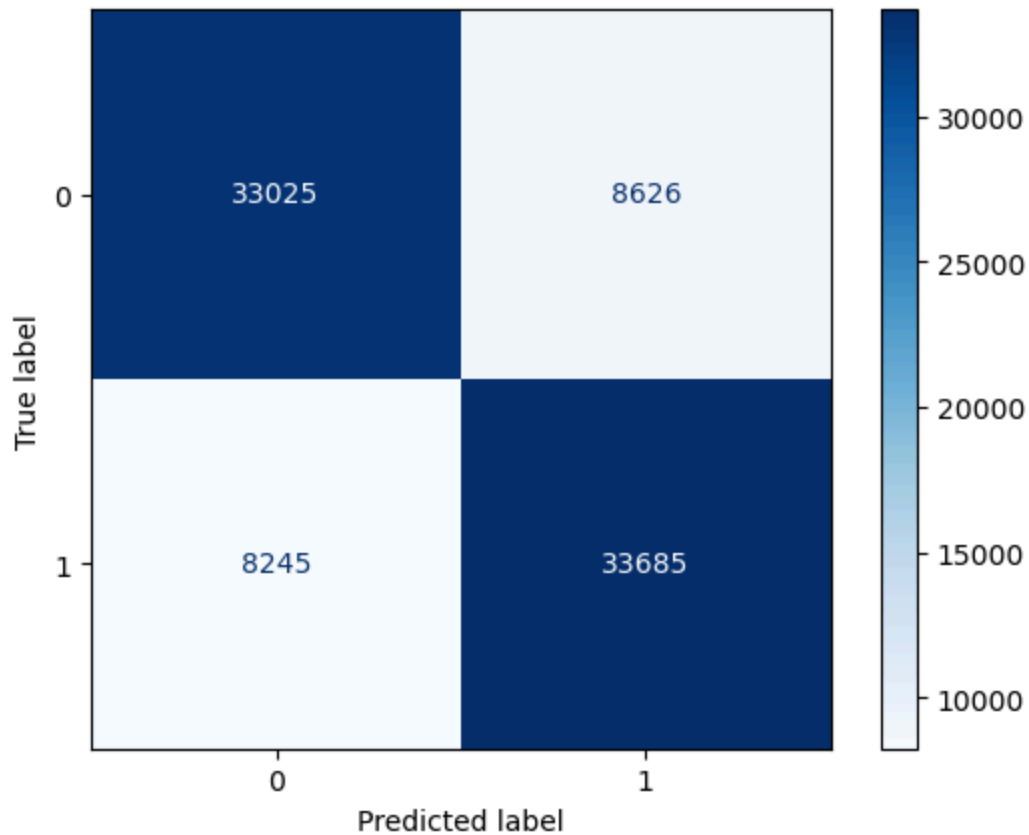
```
In [61]: y_pred_train=model.predict(X_train)
y_pred_test=model.predict(X_test)

accuracy_train = accuracy_score(y_train, y_pred_train) * 100
accuracy_test = accuracy_score(y_test, y_pred_test) * 100 # Convert to %
recall = recall_score(y_test, y_pred_test) * 100 # Sensitivity / True Positive Rat
precision = precision_score(y_test, y_pred_test) * 100
f1 = f1_score(y_test, y_pred_test) * 100

sensitivity = recall_score(y_test, y_pred_test, pos_label=1)* 100 # Sensitivity for
specificity = recall_score(y_test, y_pred_test, pos_label=0)* 100 # Specificity fo

# Print results
print(f"Accuracy train: {accuracy_train:.2f}%")
print(f"Accuracy test: {accuracy_test:.2f}%")
print(f"Recall: {recall:.2f}%")
print(f"sensitivity: {sensitivity:.2f}%")
print(f"specificity: {specificity:.2f}%")
print(f"Precision: {precision:.2f}%")
print(f"F1-Score: {f1:.2f}%")
display = ConfusionMatrixDisplay.from_estimator(model, X_test, y_test, cmap=plt.cm.
```

```
Accuracy train: 85.01%
Accuracy test: 79.81%
Recall: 80.34%
sensitivity: 80.34%
specificity: 79.29%
Precision: 79.61%
F1-Score: 79.97%
```



3.8) Applying SVM algorithm

```
In [62]: from sklearn import svm
model = svm.SVC(kernel='linear')
model.fit(X_train, y_train)
```

```
Out[62]: SVC
SVC(kernel='linear')
```

```
In [63]: y_pred_train=model.predict(X_train)
y_pred_test=model.predict(X_test)

accuracy_train = accuracy_score(y_train, y_pred_train) * 100
accuracy_test = accuracy_score(y_test, y_pred_test) * 100 # Convert to %
recall = recall_score(y_test, y_pred_test) * 100 # Sensitivity / True Positive Rat
precision = precision_score(y_test, y_pred_test) * 100
f1 = f1_score(y_test, y_pred_test) * 100

sensitivity = recall_score(y_test, y_pred_test, pos_label=1)* 100 # Sensitivity for
specificity = recall_score(y_test, y_pred_test, pos_label=0)* 100 # Specificity fo

# Print results
print(f"Accuracy train: {accuracy_train:.2f}%")
print(f"Accuracy test: {accuracy_test:.2f}%")
print(f"Recall: {recall:.2f}%")
```

```

print(f"sensitivity: {sensitivity:.2f}%")
print(f"specificity: {specificity:.2f}%")
print(f"Precision: {precision:.2f}%")
print(f"F1-Score: {f1:.2f}%")
display = ConfusionMatrixDisplay.from_estimator(model, X_test, y_test, cmap=plt.cm.

```

Accuracy train: 73.80%

Accuracy test: 73.49%

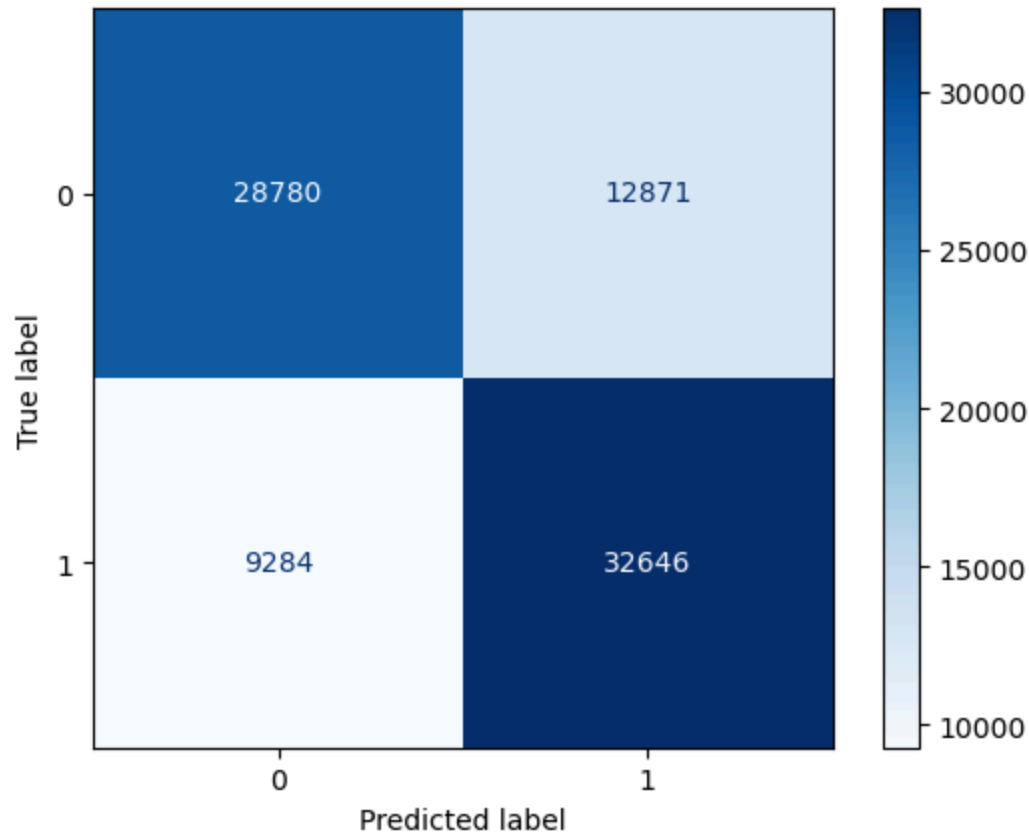
Recall: 77.86%

sensitivity: 77.86%

specificity: 69.10%

Precision: 71.72%

F1-Score: 74.66%



3.8) Applying KNN algorithm with optimal N_neighbors

Finding optimal K value.

```

In [64]: from sklearn.neighbors import KNeighborsClassifier      # k-nearest neighbors (kNN)
         from sklearn.model_selection import cross_val_score

```

```

In [81]: k_values = range(1, 20)
         cv_scores = []      # stores the mean cross-validation scores for each value of k

         for k in k_values:
             knn = KNeighborsClassifier(n_neighbors=k)      # n_neighbors default = 5
             scores = cross_val_score(knn, X_train, y_train, cv=5)  # 5-fold cross-validation
             cv_scores.append(scores.mean())

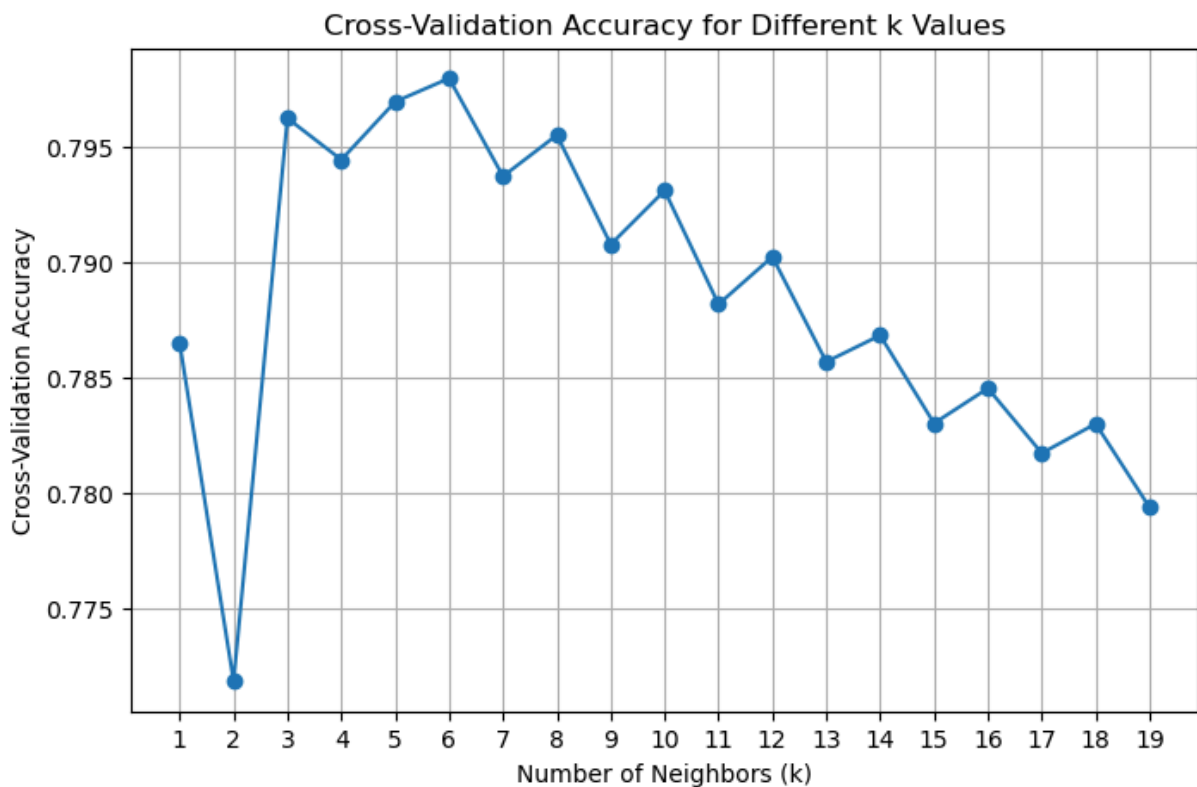
```

```
In [82]: print(list(k_values))
print(scores)
print(cv_scores)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
[0.77921184 0.77825469 0.7805728 0.77884961 0.78003111]
[0.7865088568973206, 0.7718493766937116, 0.7962688827819077, 0.7944682235908116, 0.7
969867524140807, 0.7980007432393297, 0.7937503592372634, 0.7955390467755413, 0.79079
51327303069, 0.7931311989061545, 0.7882018343328813, 0.7902567305732782, 0.785701255
3393067, 0.7868498485938289, 0.7830331779735618, 0.7845526666923976, 0.7817380201653
134, 0.7830152314989253, 0.7793840092147255]
```

```
In [83]: plt.figure(figsize=(8, 5))
plt.plot(k_values, cv_scores, marker='o')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Cross-Validation Accuracy')
plt.title('Cross-Validation Accuracy for Different k Values')
plt.xticks(k_values)
plt.grid()
plt.show()

# Best k
best_k = k_values[np.argmax(cv_scores)]
print("Optimal k value:", best_k)
```



Optimal k value: 6

Traning the model.

```
In [84]: model = KNeighborsClassifier(n_neighbors=best_k)
model.fit(X_train, y_train)
```

Out[84]:

▼ KNeighborsClassifier ⓘ ?

KNeighborsClassifier(n_neighbors=6)

In [85]:

```

y_pred_train=model.predict(X_train)
y_pred_test=model.predict(X_test)

accuracy_train = accuracy_score(y_train, y_pred_train) * 100
accuracy_test = accuracy_score(y_test, y_pred_test) * 100 # Convert to %
recall = recall_score(y_test, y_pred_test) * 100 # Sensitivity / True Positive Rat
precision = precision_score(y_test, y_pred_test) * 100
f1 = f1_score(y_test, y_pred_test) * 100

sensitivity = recall_score(y_test, y_pred_test, pos_label=1)* 100 # Sensitivity for

specificity = recall_score(y_test, y_pred_test, pos_label=0)* 100 # Specificity fo

# Print results
print(f"Accuracy train: {accuracy_train:.2f}%")
print(f"Accuracy test: {accuracy_test:.2f}%")
print(f"Recall: {recall:.2f}%")
print(f"sensitivity: {sensitivity:.2f}%")
print(f"specificity: {specificity:.2f}%")
print(f"Precision: {precision:.2f}%")
print(f"F1-Score: {f1:.2f}%")
display = ConfusionMatrixDisplay.from_estimator(model, X_test, y_test, cmap=plt.cm.

```

Accuracy train: 84.66%

Accuracy test: 80.35%

Recall: 84.15%

sensitivity: 84.15%

specificity: 76.53%

Precision: 78.30%

F1-Score: 81.12%

