Diabetes prediction from UCI diabetes data

by Angela Chen, Ella Hein, Scout McKee, and Sharon Voon 2023/11/19

In [1]:
```
# Importing all used libraries
import pandas as pd
import altair as alt
from sklearn.model_selection import train_test_split, cross_val_score, cross_valida
from sklearn.metrics import accuracy_score
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.compose import ColumnTransformer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.dummy import DummyClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from count_null import count_null
from get_feature_importance import get_feature_importances
from check_imbalance import check_imbalance

import sys
import os
sys.path.append('.')
from model_cross_val import model_cross_val
```

Summary

This project endeavors to develop a predictive classification model for ascertaining an individual's diabetic status, while comparing the efficiency of logistic regression and k-nearest neighbours (k-nn) algorithms. The dataset used in this analysis is collected through the Behavioral Risk Factor Surveillance System (BFRSS) by the Centers for Disease Control and Prevention (CDC) for the year 2015. Notably, the primary determinant influencing the prediction is identified as the feature High Blood Pressue (HighBP), displaying a coefficient of 0.354 as revealed by the logistic regression model. A futile attempt of hyperparameter optimization was carried out on the k-nn model with intention to improve the validation score, but result showed that it only improved the validation score from 0.707 to 0.742. The optimized logistic regression model demonstrates a test score of 0.728 ,while the k-nn model yields a test score of 0.746. Both of the test scores are relatively close to the validation score which shows that the model will generalized well to unseen data, however, there is still room for improvement in the test score.

Introduction

Diabetes mellitus, commonly referred to as diabetes is a disease which impacts the body’s control of blood glucose levels (Sapra, Bhandari 2023). It is important to note that there are different types of diabetes, although we do not explore this discrepancy in this project (Sapra, Bhandari 2023). Diabetes is a manageable disease thanks to the discovery of insulin in 1922. Globally, 1 in 11 adults have diabetes (Sapra, Bhandari 2023). As such, understanding the factors which are strongly related to diabetes can be important for researchers studying how to better prevent or manage the disease. In this project, we create several machine learning models to predict diabetes in a patient and evaluate the success of these models. We also explore the coefficients of a logistic regression model to better understand the factors which are associated with diabetes.

Methods

Data

The dataset used in this project is a collection of the Centers for Disease Control and Prevention (CDC) diabetes health indicators collected as a response to the CDC's BRFSS2015 survey. The data were sourced from the UCI Machine Learning Repository (Burrows, Hora, Geiss, Gregg, and Albright 2017) which can be found here. The file specifically used from this repository for this analysis includes 70, 692 survey responses from which 50% of the respondents recorded having either prediabetes or diabetes. Each row in the dataset represents a recorded survey response including whether or not the responded has diabetes or prediabetes, and a collection of 21 other diabetes health indicators identified by the CDC.

Analysis

Four separate classification models were tested on this dataset with the purpose of determining the best model for classifying a patient with diabetes or prediabetes as opposed to no diabetes or prediabetes. The classifiers tested were: dummy, decision tree, k-nearest neighbors (k-nn), and logistic regression. All features from the original dataset were included in each model. In all cases, the data were split into training and testing datasets, with 80% of the data designated as training and 20% as testing. The data was preprocessesed such that all continuous (non-binary) variables were scaled using a scikit-learn's StandardScaler function. Model performance was tested using a 10 - fold cross validation score. Feature importance was investigated using the coefficients generated by the logistic regression algorithm. The k-nn algorithm's hyperparameter K was optimized using the F1 score as the classification metric. Python programming (Van Rossum and Drake 2009) was used for all analysis. The following Python packages were used for this analysis: Pandas (McKinney 2010), altair (VanderPlas, 2018), and scikit-learn (Pedregosa et al. 2011).

In [2]:
```
# Reading into the data file
diabetes_df = pd.read_csv("../data/diabetes_binary_5050split_health_indicators_BRFS
diabetes_df.head()
```

Out[2]:

	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAt
0	0.0	1.0	0.0	1.0	26.0	0.0	0.0	
1	0.0	1.0	1.0	1.0	26.0	1.0	1.0	
2	0.0	0.0	0.0	1.0	26.0	0.0	0.0	
3	0.0	1.0	1.0	1.0	28.0	1.0	0.0	
4	0.0	0.0	0.0	1.0	29.0	1.0	0.0	

5 rows × 22 columns

In [3]:
```
# Exploratory Data Analysis
diabetes_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70692 entries, 0 to 70691
Data columns (total 22 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Diabetes_binary       70692 non-null  float64
 1   HighBP                70692 non-null  float64
 2   HighChol              70692 non-null  float64
 3   CholCheck             70692 non-null  float64
 4   BMI                   70692 non-null  float64
 5   Smoker                70692 non-null  float64
 6   Stroke                70692 non-null  float64
 7   HeartDiseaseorAttack  70692 non-null  float64
 8   PhysActivity          70692 non-null  float64
 9   Fruits                70692 non-null  float64
 10  Veggies               70692 non-null  float64
 11  HvyAlcoholConsump     70692 non-null  float64
 12  AnyHealthcare         70692 non-null  float64
 13  NoDocbcCost           70692 non-null  float64
 14  GenHlth               70692 non-null  float64
 15  MentHlth              70692 non-null  float64
 16  PhysHlth              70692 non-null  float64
 17  DiffWalk              70692 non-null  float64
 18  Sex                   70692 non-null  float64
 19  Age                   70692 non-null  float64
 20  Education             70692 non-null  float64
 21  Income                70692 non-null  float64
dtypes: float64(22)
memory usage: 11.9 MB
```

In [4]:
```
print(diabetes_df.shape)
diabetes_df.describe().T
```

```
(70692, 22)
```

Out[4]:

	count	mean	std	min	25%	50%	75%	max
Diabetes_binary	70692.0	0.500000	0.500004	0.0	0.0	0.5	1.0	1.0
HighBP	70692.0	0.563458	0.495960	0.0	0.0	1.0	1.0	1.0
HighChol	70692.0	0.525703	0.499342	0.0	0.0	1.0	1.0	1.0
CholCheck	70692.0	0.975259	0.155336	0.0	1.0	1.0	1.0	1.0
BMI	70692.0	29.856985	7.113954	12.0	25.0	29.0	33.0	98.0
Smoker	70692.0	0.475273	0.499392	0.0	0.0	0.0	1.0	1.0
Stroke	70692.0	0.062171	0.241468	0.0	0.0	0.0	0.0	1.0
HeartDiseaseorAttack	70692.0	0.147810	0.354914	0.0	0.0	0.0	0.0	1.0
PhysActivity	70692.0	0.703036	0.456924	0.0	0.0	1.0	1.0	1.0
Fruits	70692.0	0.611795	0.487345	0.0	0.0	1.0	1.0	1.0
Veggies	70692.0	0.788774	0.408181	0.0	1.0	1.0	1.0	1.0
HvyAlcoholConsump	70692.0	0.042721	0.202228	0.0	0.0	0.0	0.0	1.0
AnyHealthcare	70692.0	0.954960	0.207394	0.0	1.0	1.0	1.0	1.0
NoDocbcCost	70692.0	0.093914	0.291712	0.0	0.0	0.0	0.0	1.0
GenHlth	70692.0	2.837082	1.113565	1.0	2.0	3.0	4.0	5.0
MentHlth	70692.0	3.752037	8.155627	0.0	0.0	0.0	2.0	30.0
PhysHlth	70692.0	5.810417	10.062261	0.0	0.0	0.0	6.0	30.0
DiffWalk	70692.0	0.252730	0.434581	0.0	0.0	0.0	1.0	1.0
Sex	70692.0	0.456997	0.498151	0.0	0.0	0.0	1.0	1.0
Age	70692.0	8.584055	2.852153	1.0	7.0	9.0	11.0	13.0
Education	70692.0	4.920953	1.029081	1.0	4.0	5.0	6.0	6.0
Income	70692.0	5.698311	2.175196	1.0	4.0	6.0	8.0	8.0

In [5]:

```
# Check for duplicate in dataset
duplicate_rows = diabetes_df.duplicated()
print(duplicate_rows.value_counts())
```

```
False    69057
True      1635
Name: count, dtype: int64
```

In [6]:

```
# Check for imbalance dataset
diabetes_df.drop_duplicates(inplace=True)
diabetes_df["Diabetes_binary"].value_counts()
```

Out[6]:
```
Diabetes_binary
1.0    35097
0.0    33960
Name: count, dtype: int64
```

In [7]:
```
# Check for null values
count_null(diabetes_df)
```

Out[7]:

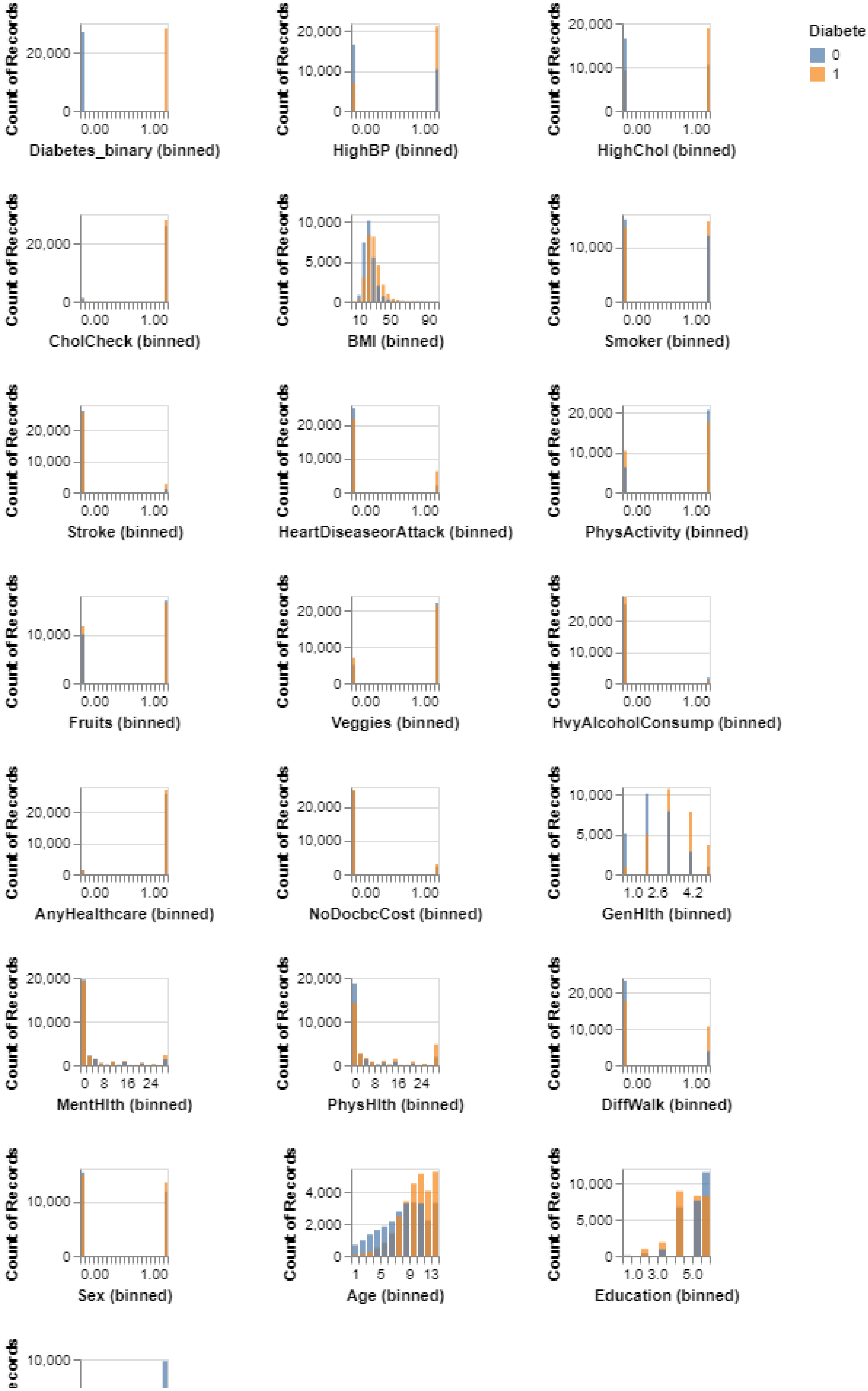
	column	count
0	Diabetes_binary	0
1	HighBP	0
2	HighChol	0
3	CholCheck	0
4	BMI	0
5	Smoker	0
6	Stroke	0
7	HeartDiseaseorAttack	0
8	PhysActivity	0
9	Fruits	0
10	Veggies	0
11	HvyAlcoholConsump	0
12	AnyHealthcare	0
13	NoDocbcCost	0
14	GenHlth	0
15	MentHlth	0
16	PhysHlth	0
17	DiffWalk	0
18	Sex	0
19	Age	0
20	Education	0
21	Income	0

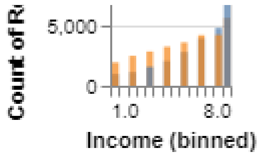
In [8]:
```
#Creating train and test data
train_df, test_df = train_test_split(diabetes_df,
                                     test_size = 0.2,
                                     random_state=123)

X_train = train_df.drop(columns = "Diabetes_binary")
```

```python
y_train = train_df["Diabetes_binary"]

X_test = test_df.drop(columns = "Diabetes_binary")
y_test = test_df["Diabetes_binary"]
```

In [9]:
```python
# plotting histogram distributions
alt.data_transformers.enable("vegafusion")
numeric_cols = train_df.select_dtypes(include=['number']).columns.to_list()

hist_plot = alt.Chart(train_df).mark_bar(opacity=0.7).encode(
            x=alt.X(alt.repeat()).type('quantitative').
                    bin(maxbins=20),
            y=alt.Y('count()').stack(False),
            color=alt.Color('Diabetes_binary:N')
        ).properties(
            width=60,
            height=60
        ).repeat(
            numeric_cols,
            columns=3
        )

# hist_plot.save("../results/EDA_histogram_plot.png")
hist_plot
```

Out[9]:


Diabete
0
1
Count of Records
Diabetes_binary (binned)
HighBP (binned)
HighChol (binned)
CholCheck (binned)
BMI (binned)
Smoker (binned)
Stroke (binned)
HeartDiseaseorAttack (binned)
PhysActivity (binned)
Fruits (binned)
Veggies (binned)
HvyAlcoholConsump (binned)
AnyHealthcare (binned)
NoDocbcCost (binned)
GenHlth (binned)
MentHlth (binned)
PhysHlth (binned)
DiffWalk (binned)
Sex (binned)
Age (binned)
Education (binned)



In [10]:
```
#Creating the baseline for our model
dummy = DummyClassifier()
scores = cross_validate(dummy, X_train, y_train, return_train_score=True)
pd.DataFrame(scores)
```

Out[10]:

	fit_time	score_time	test_score	train_score
0	0.013001	0.002999	0.510363	0.510408
1	0.012047	0.002996	0.510363	0.510408
2	0.009997	0.002043	0.510363	0.510408
3	0.010993	0.005999	0.510453	0.510386
4	0.014000	0.006006	0.510453	0.510386

Model comparison

In [11]:
```
# Designate binary and continuous cols
binary_cols = ['HighBP','HighChol', 'CholCheck', 'Smoker',
               'Stroke', 'HeartDiseaseorAttack',
               'PhysActivity', 'Fruits', 'Veggies',
               'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost',
              'DiffWalk', 'Sex']
continuous_cols = ['BMI', 'Age', 'GenHlth', 'MentHlth',
                   'PhysHlth', 'Education', 'Income']
```

In [12]:
```
# Create a pre-processor which scales the continuous cols
preprocessor = ColumnTransformer(
    transformers=[
        ('continuous', StandardScaler(), continuous_cols),
        ('binary', 'passthrough', binary_cols)
    ])
```

In [13]:
```
# Models to test
models = {
    "Dummy": make_pipeline(preprocessor,
                           DummyClassifier()),
    "Decision tree": make_pipeline(preprocessor,
                                   DecisionTreeClassifier(random_state=123)),
    "Logistic regression": make_pipeline(preprocessor,
                                         LogisticRegression(max_iter=1000)),
    "Knn": make_pipeline(preprocessor,
                         KNeighborsClassifier())
}
```

In [14]:
```
#Below is a function from the DSCI 571 Lecture notes which we will use for cross va
def mean_std_cross_val_scores(model, X_train, y_train, **kwargs):
    """
    Returns mean and std of cross validation

    Parameters
    ----------
    model :
        scikit-learn model
    X_train : numpy array or pandas DataFrame
        X in the training data
    y_train :
        y in the training data

    Returns
    ----------
        pandas Series with mean scores from cross_validation
    """

    scores = cross_validate(model, X_train, y_train, **kwargs)

    mean_scores = pd.DataFrame(scores).mean()
    std_scores = pd.DataFrame(scores).std()
    out_col = []

    for i in range(len(mean_scores)):
        out_col.append((f"%0.3f (+/- %0.3f)" %
                        (mean_scores.iloc[i], std_scores.iloc[i])))

    return pd.Series(data=out_col, index=mean_scores.index)
#The below code is adapted from DSCI 571 lecture notes and lab solutions.

# Evaluate each model
results_df = model_cross_val(models, X_train, y_train)
results_df
```

Out[14]:

	fit_time	score_time	test_score	train_score
Dummy	0.047 (+/- 0.006)	0.014 (+/- 0.010)	0.510 (+/- 0.000)	0.510 (+/- 0.000)
Decision tree	0.688 (+/- 0.144)	0.011 (+/- 0.002)	0.643 (+/- 0.009)	0.996 (+/- 0.000)
Logistic regression	0.352 (+/- 0.042)	0.012 (+/- 0.002)	0.744 (+/- 0.005)	0.744 (+/- 0.001)
Knn	0.045 (+/- 0.011)	1.810 (+/- 0.284)	0.707 (+/- 0.006)	0.797 (+/- 0.001)

Feature Importance

In [ ]:

In [15]:
```
# Show coefficients
get_feature_importances(X_train, y_train, "Diabetes_binary")
```

Out[15]:

	features	coefficients
0	HighBP	0.353947
1	HighChol	0.289383
2	CholCheck	0.216358
3	BMI	0.536654
4	Smoker	-0.011014
5	Stroke	0.045484
6	HeartDiseaseorAttack	0.092977
7	PhysActivity	-0.012531
8	Fruits	-0.011764
9	Veggies	-0.021392
10	HvyAlcoholConsump	-0.160859
11	AnyHealthcare	0.008151
12	NoDocbcCost	0.007014
13	GenHlth	0.629058
14	MentHlth	-0.035788
15	PhysHlth	-0.083456
16	DiffWalk	0.056345
17	Sex	0.133742
18	Age	0.423533
19	Education	-0.039003
20	Income	-0.128971

Exploring Hyperparameters

The logistic regression model had the highest accuracy score of the models we explored. However, the k-nn model was the second best model and had a cross validation accuracy only 0.03 less than the regression model. As such, we will now explore the hyperparameters of the k-nn model to see if we can improve this score.

In [16]:

```
from sklearn.model_selection import RandomizedSearchCV

knn_pipe = make_pipeline(preprocessor, KNeighborsClassifier())

param_grid = {
    "kneighborsclassifier__n_neighbors": [50, 100, 200, 300, 500]
```

```
}
first_search = RandomizedSearchCV(knn_pipe, param_distributions=param_grid, n_iter=
first_search.fit(X_train, y_train)
```

```
C:\Users\sharo\miniconda3\envs\571\lib\site-packages\sklearn\model_selection\_searc
h.py:307: UserWarning: The total space of parameters 5 is smaller than n_iter=10. Ru
nning 5 iterations. For exhaustive searches, use GridSearchCV.
  warnings.warn(
```

Out[16]:

```
▸ RandomizedSearchCV
▸ estimator: Pipeline
  ▸ columntransformer: ColumnTransformer
      ▸ continuous          ▸ binary
      ▸ StandardScaler      ▸ passthrough
  ▸ KNeighborsClassifier
```

In [17]:

```
print ("the best parameter:", first_search.best_params_)
print ("the best score:", first_search.best_score_)
```

```
the best parameter: {'kneighborsclassifier__n_neighbors': 100}
the best score: 0.7417322834645669
```

Model Selection and Testing

In [18]:

```
final_knn = KNeighborsClassifier(n_neighbors=100)
final_knn.fit(X_train, y_train)
```

Out[18]:

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=100)
```

In [19]:

```
y_pred_knn = final_knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_knn)
print(f'Accuracy of knn model of n=100: {accuracy}')

# Additional metrics
print(classification_report(y_test, y_pred_knn))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred_knn))
print('AUC-ROC Score:', roc_auc_score(y_test, final_knn.predict_proba(X_test)[:, 1]
```

```
Accuracy of knn model of n=100: 0.7280625543006082
              precision    recall  f1-score   support

         0.0       0.76      0.67      0.71      6912
         1.0       0.71      0.78      0.74      6900

    accuracy                           0.73     13812
   macro avg       0.73      0.73      0.73     13812
weighted avg       0.73      0.73      0.73     13812

Confusion Matrix:
 [[4660 2252]
 [1504 5396]]
AUC-ROC Score: 0.8017921887580515
```

In [20]:

```
final_logistic = LogisticRegression(max_iter=1000)
final_logistic.fit(X_train, y_train)
```

Out[20]:

```
LogisticRegression
LogisticRegression(max_iter=1000)
```

In [21]:

```
y_pred_log = final_logistic.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_log)
print(f'Accuracy of logistic model: {accuracy}')

# Additional metrics
print(classification_report(y_test, y_pred_log))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred_log))
print('AUC-ROC Score:', roc_auc_score(y_test,
                                      final_logistic.predict_proba(X_test)[:, 1]))
```

```
Accuracy of logistic model: 0.7461627570228787
              precision    recall  f1-score   support

         0.0       0.76      0.72      0.74      6912
         1.0       0.74      0.77      0.75      6900

    accuracy                           0.75     13812
   macro avg       0.75      0.75      0.75     13812
weighted avg       0.75      0.75      0.75     13812

Confusion Matrix:
 [[5006 1906]
 [1600 5300]]
AUC-ROC Score: 0.8207008080884326
```

Comments and Evaluation:

- Accuracy:

The Logistic Regression model outperforms the k-nn model in terms of accuracy (74.62% vs. 72.81%).

- Precision, Recall, and F1-Score:

Both models have comparable precision, recall, and F1-score for class 1.0 (diabetic), but Logistic Regression slightly outperforms k-nn in all these metrics.

- Confusion Matrix:

Logistic Regression has a lower number of false positives and false negatives compared to k-nn. This indicates that the Logistic Regression model is making fewer errors in both positive and negative predictions.

- AUC-ROC Score:

The AUC-ROC score, which measures the model's ability to distinguish between classes(diabetic versus non-diabetic, is higher for the Logistic Regression model (0.821) compared to the k-nn model (0.802).

Conclusion:

Based on the evaluation metrics, the Logistic Regression model performs better than the K-Nearest Neighbors model on the provided test dataset. It achieves higher accuracy, precision, recall, and AUC-ROC score, indicating a better overall performance. Therefore, considering these results and the fact that Logistic Regression also offers interpretability of feature coefficients, it seems reasonable to prefer the Logistic Regression model for this specific classification task.

Reference:

Centers for Disease Control and Prevention. (2014). CDC Diabetes Health Indicators. University of California, Irvine, School of Information; Computer Sciences. Retrieved November 14, 2023, from https://archive.ics.uci.edu/dataset/891/cdc+diabetes+health+indicators

Sapra, A., & Bhandari, P. (2023, June 21). Diabetes. In StatPearls [Internet]. Treasure Island (FL): StatPearls Publishing. Available from: https://www.ncbi.nlm.nih.gov/books/NBK551501/

VanderPlas, J., Granger, B., Heer, J., Moritz, D., Wongsuphasawat, K., Satyanarayan, A., ... Sievert, S. (2018). Altair: Interactive statistical visualizations for Python. Journal of Open Source Software, 3(32), 1057.

Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

McKinney, W. (2010). Data structures for statistical computing in Python. Proceedings of the 9th Python in Science Conference, Volume 445.