INT 7623

Data Science for Business

Final project

Diabetics prediction using K-means algorithm

By

Keerthana

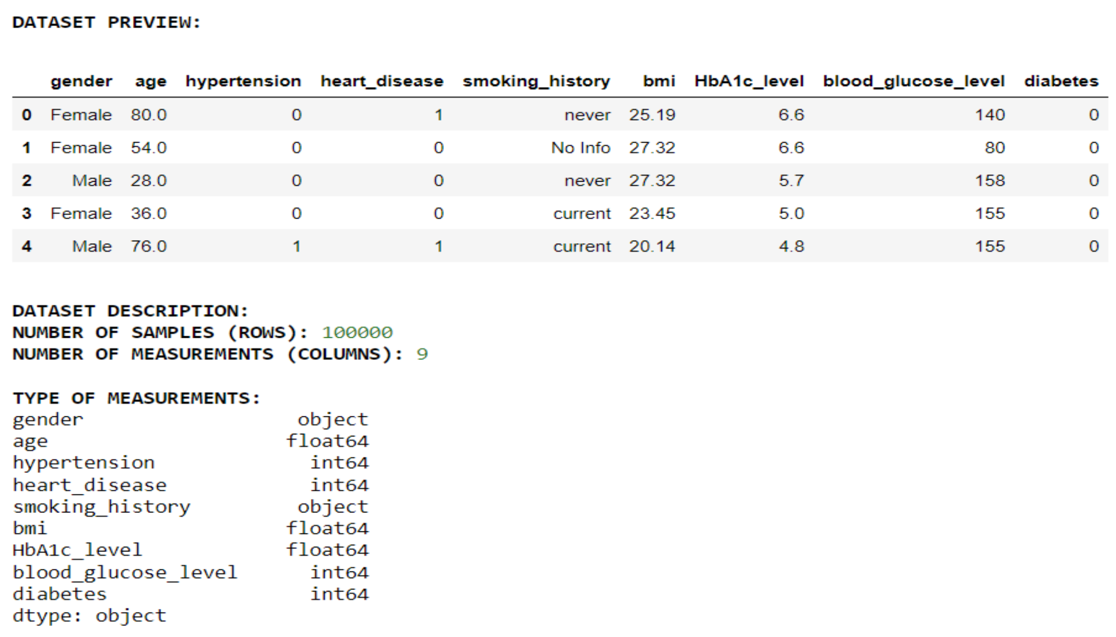
Sridevi Anandan

Introduction:

Diabetes is a global epidemic and a major public health concern. According to World Health Organization (2011) diabetes is a chronic, metabolic disease that can be identified by high level of blood glucose, which after a period of time may lead to severe damage to heart, blood vessels, eyes, kidneys and nerves. Early detection and intervention are crucial for effectively managing diabetes and reducing its associated complications. In this project, we aim to leverage machine learning techniques to predict the onset of diabetes in patients based on their medical history. The dataset used in this project is comprehensive, and the EDA will provide valuable insights into the relationships between different health indicators and their role in predicting diabetes onset.

Dataset descriptions:

The Diabetes prediction dataset is a collection of medical and demographic data from patients, along with their diabetes status (positive or negative), it has attributes such as age, gender, body mass index (BMI), hypertension, heart disease, smoking history, HbA1c level, and blood glucose level.



```
DATASET PREVIEW:
```

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
0	Female	80.0	0	1	never	25.19	6.6	140	0
1	Female	54.0	0	0	No Info	27.32	6.6	80	0
2	Male	28.0	0	0	never	27.32	5.7	158	0
3	Female	36.0	0	0	current	23.45	5.0	155	0
4	Male	76.0	1	1	current	20.14	4.8	155	0

```
DATASET DESCRIPTION:
NUMBER OF SAMPLES (ROWS): 100000
NUMBER OF MEASUREMENTS (COLUMNS): 9

TYPE OF MEASUREMENTS:
gender                  object
age                    float64
hypertension             int64
heart_disease            int64
smoking_history         object
bmi                    float64
HbA1c_level            float64
blood_glucose_level      int64
diabetes                 int64
dtype: object
```

Description of data set:

1. Gender(object): Gender refers to the biological sex of the individual, which can have an impact on their susceptibility to diabetes. There are three categories in it male, female and other. age: Age is an important factor as diabetes is more commonly diagnosed in older adults. Age ranges from 0-80 in our dataset.

2. Hypertension(int64): Hypertension is a medical condition in which the blood pressure in the arteries is persistently elevated. It has values 0 or 1 where 0 indicates they don't have hypertension and for 1 it means they have hypertension.

3. heart_diesease(int64): Heart disease is another medical condition that is associated with an increased risk of developing diabetes. It has values 0 or 1 where 0 indicates they don't have heart disease and for 1 it means they have heart disease.

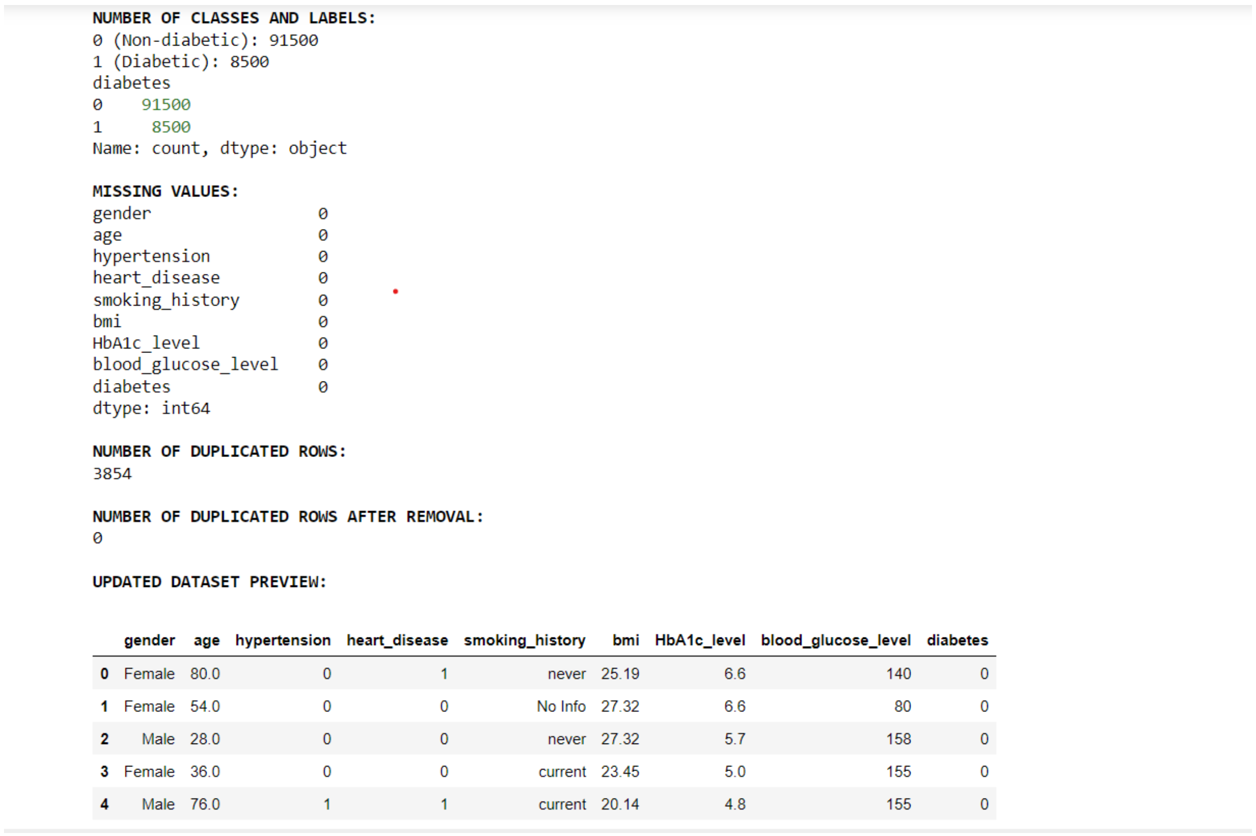
4. smoking_history(object): Smoking history is also considered a risk factor for diabetes and can exacerbate the complications associated with diabetes. In

our dataset we have 5 categories i.e not current,former,No Info,current,never and ever.

5. Bmi(float64): BMI (Body Mass Index) is a measure of body fat based on weight and height. Higher BMI values are linked to a higher risk of diabetes. The range of BMI in the dataset is from 10.16 to 71.55.
6. HbA1c_level(float64): HbA1c (Hemoglobin A1c) level is a measure of a person's average blood sugar level over the past 2-3 months. Higher levels indicate a greater risk of developing diabetes.
7. blood_glucose_level(int64): Blood glucose level refers to the amount of glucose in the bloodstream at a given time. High blood glucose levels are a key indicator of diabetes. diabetes: Diabetes is the target variable being predicted, with values of 1 indicating the presence of diabetes and 0 indicating the absence of diabetes.

Key details of the dataset:

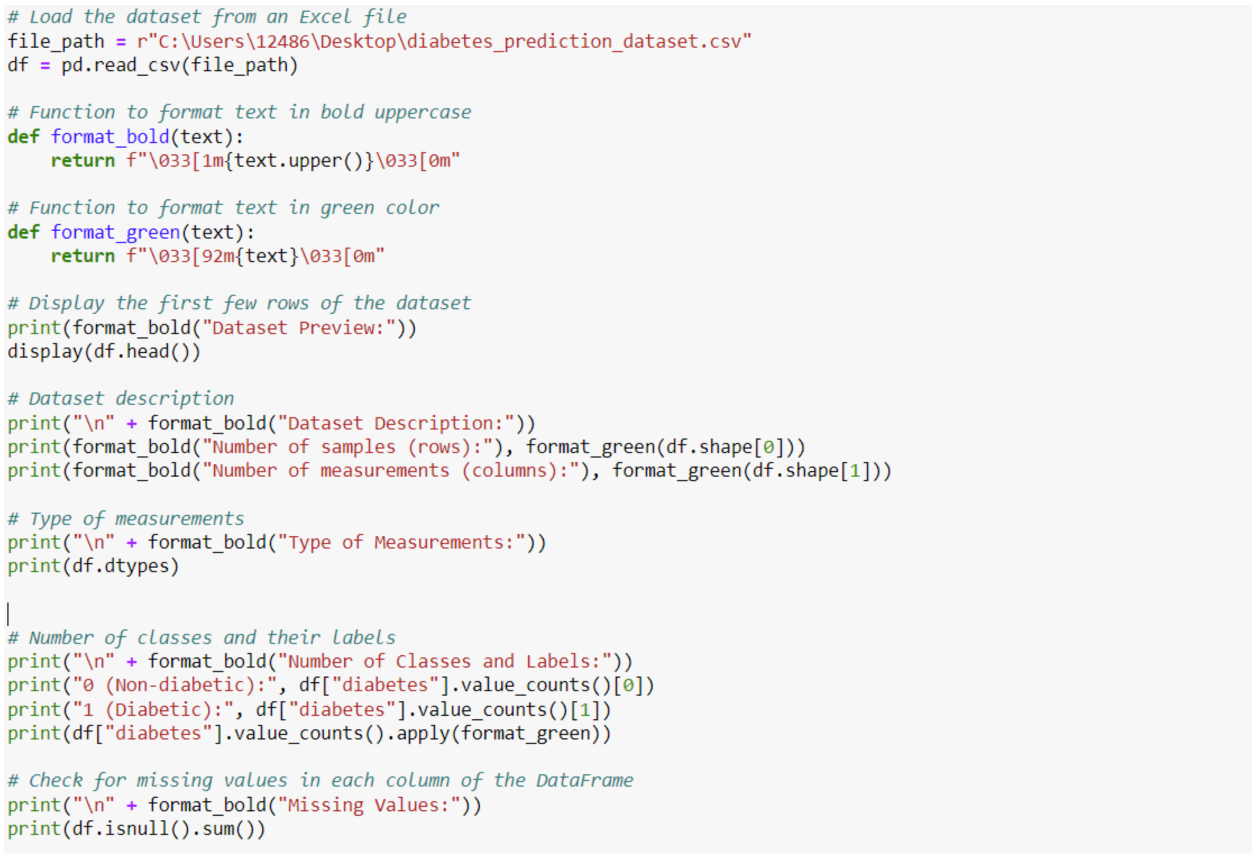
- Size: The dataset consists of a certain number of instances, each representing a patient.
- Number of Measurements: It includes several features such as age, gender, body mass index (BMI), hypertension, heart disease, smoking history, HbA1c level, and blood glucose level.
- Type of Measurements: The features in the dataset are a mix of categorical and numerical variables, reflecting various aspects of a patient's medical and demographic profile.
- Number of Classes and Labels: The target variable is the diabetes status, which has two classes: positive or negative. Patients are labelled based on whether they have been diagnosed with diabetes or not.



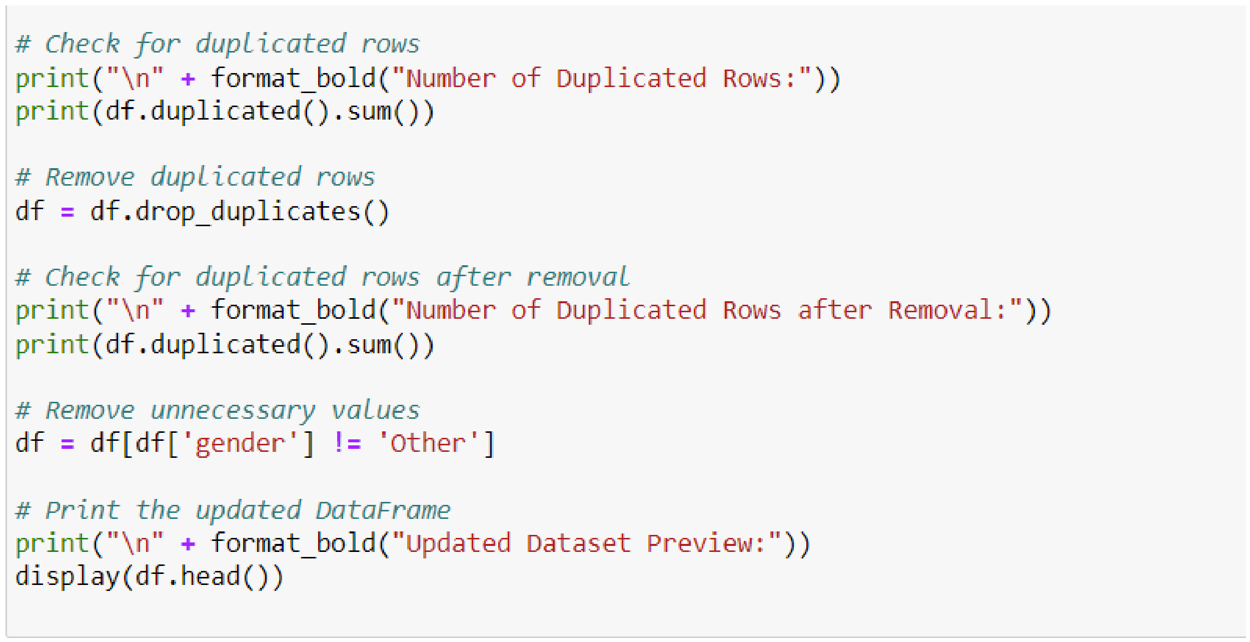
```
NUMBER OF CLASSES AND LABELS:
0 (Non-diabetic): 91500
1 (Diabetic): 8500
diabetes
0    91500
1     8500
Name: count, dtype: object

MISSING VALUES:
gender                 0
age                    0
hypertension           0
heart_disease          0
smoking_history        0
bmi                    0
HbA1c_level            0
blood_glucose_level    0
diabetes               0
dtype: int64

NUMBER OF DUPLICATED ROWS:
3854

NUMBER OF DUPLICATED ROWS AFTER REMOVAL:
0

UPDATED DATASET PREVIEW:
```

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
0	Female	80.0	0	1	never	25.19	6.6	140	0
1	Female	54.0	0	0	No Info	27.32	6.6	80	0
2	Male	28.0	0	0	never	27.32	5.7	158	0
3	Female	36.0	0	0	current	23.45	5.0	155	0
4	Male	76.0	1	1	current	20.14	4.8	155	0

Loading the data:



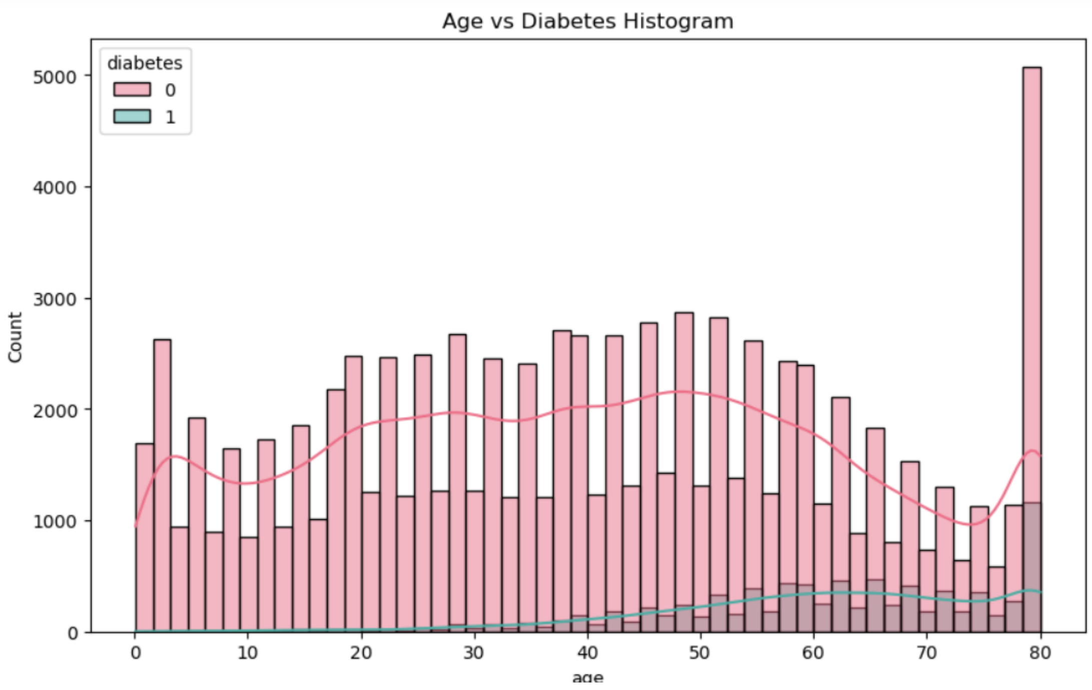
```python
# Load the dataset from an Excel file
file_path = r"C:\Users\12486\Desktop\diabetes_prediction_dataset.csv"
df = pd.read_csv(file_path)

# Function to format text in bold uppercase
def format_bold(text):
    return f"\033[1m{text.upper()}\033[0m"

# Function to format text in green color
def format_green(text):
    return f"\033[92m{text}\033[0m"

# Display the first few rows of the dataset
print(format_bold("Dataset Preview:"))
display(df.head())

# Dataset description
print("\n" + format_bold("Dataset Description:"))
print(format_bold("Number of samples (rows):"), format_green(df.shape[0]))
print(format_bold("Number of measurements (columns):"), format_green(df.shape[1]))

# Type of measurements
print("\n" + format_bold("Type of Measurements:"))
print(df.dtypes)


# Number of classes and their labels
print("\n" + format_bold("Number of Classes and Labels:"))
print("0 (Non-diabetic):", df["diabetes"].value_counts()[0])
print("1 (Diabetic):", df["diabetes"].value_counts()[1])
print(df["diabetes"].value_counts().apply(format_green))

# Check for missing values in each column of the DataFrame
print("\n" + format_bold("Missing Values:"))
print(df.isnull().sum())
```

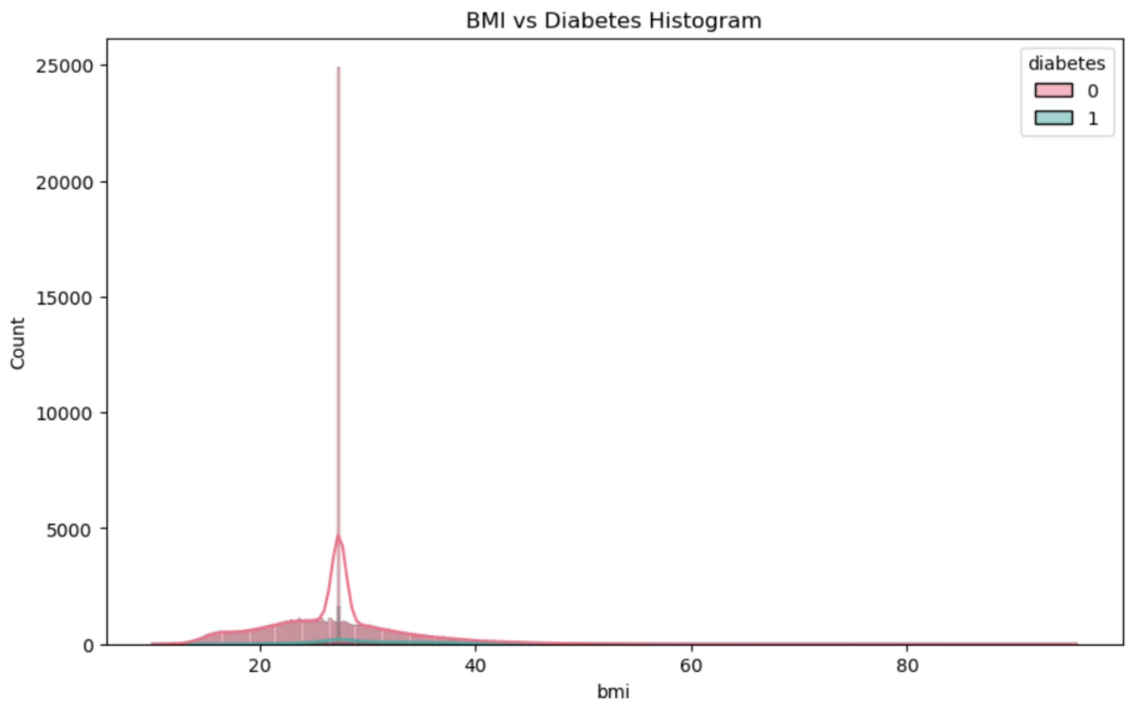


```python
# Check for duplicated rows
print("\n" + format_bold("Number of Duplicated Rows:"))
print(df.duplicated().sum())

# Remove duplicated rows
df = df.drop_duplicates()

# Check for duplicated rows after removal
print("\n" + format_bold("Number of Duplicated Rows after Removal:"))
print(df.duplicated().sum())

# Remove unnecessary values
df = df[df['gender'] != 'Other']

# Print the updated DataFrame
print("\n" + format_bold("Updated Dataset Preview:"))
display(df.head())
```

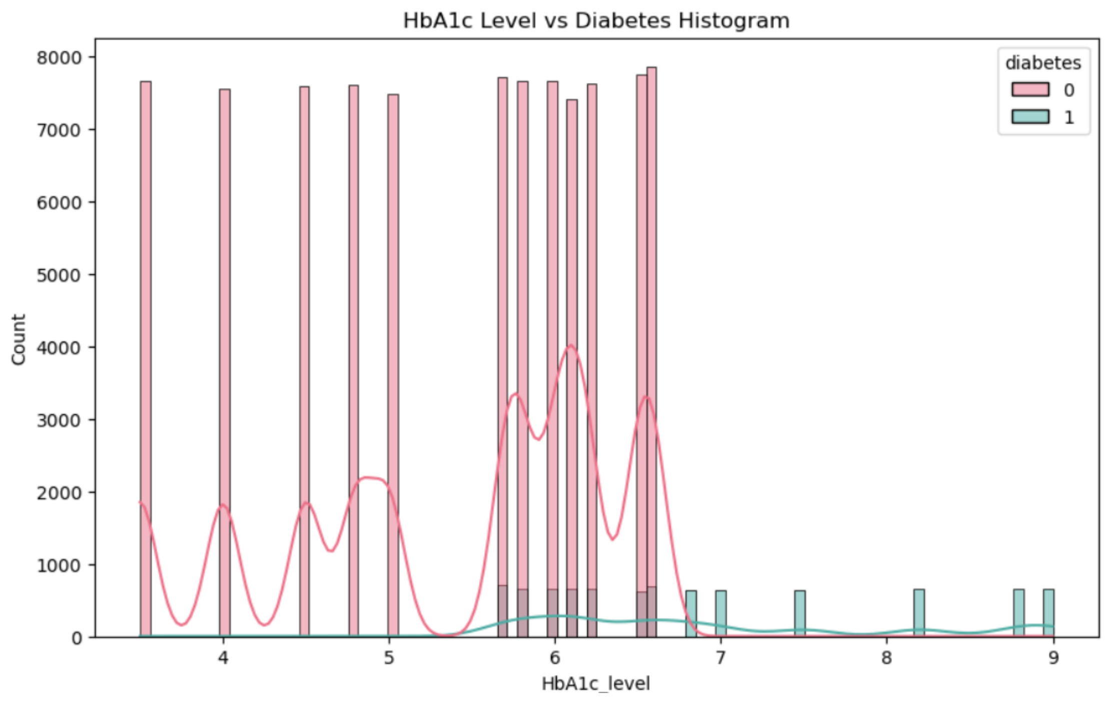
Discovering & visualizing the insights:

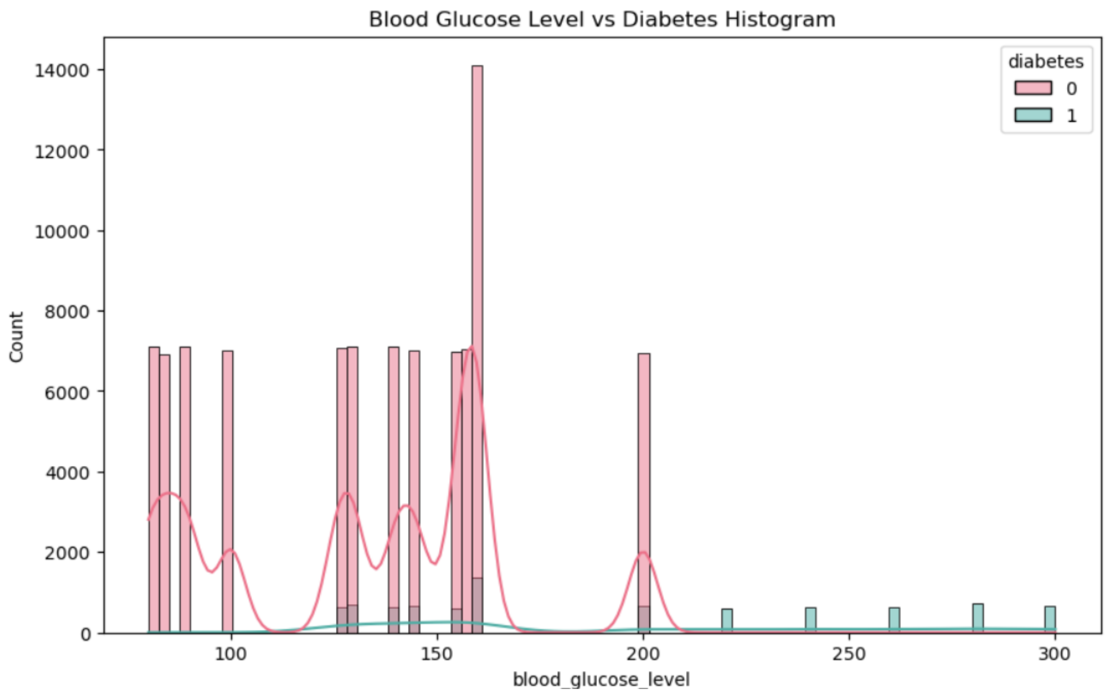
```
# Age vs Diabetes Histogram
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x="age", hue="diabetes", kde=True, palette="husl")
plt.title("Age vs Diabetes Histogram")
plt.show()

# BMI vs Diabetes Histogram
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x="bmi", hue="diabetes", kde=True, palette="husl")
plt.title("BMI vs Diabetes Histogram")
plt.show()

# HbA1c Level vs Diabetes Histogram
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x="HbA1c_level", hue="diabetes", kde=True, palette="husl")
plt.title("HbA1c Level vs Diabetes Histogram")
plt.show()

# Blood Glucose Level vs Diabetes Histogram
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x="blood_glucose_level", hue="diabetes", kde=True, palette="husl")
plt.title("Blood Glucose Level vs Diabetes Histogram")
plt.show()

# Gender vs Diabetes Countplot
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x="gender", hue="diabetes", palette="husl")
plt.title("Gender vs Diabetes Countplot")
plt.show()

# Hypertension vs Diabetes Countplot
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x="hypertension", hue="diabetes", palette="husl")
plt.title("Hypertension vs Diabetes Countplot")
plt.show()
```

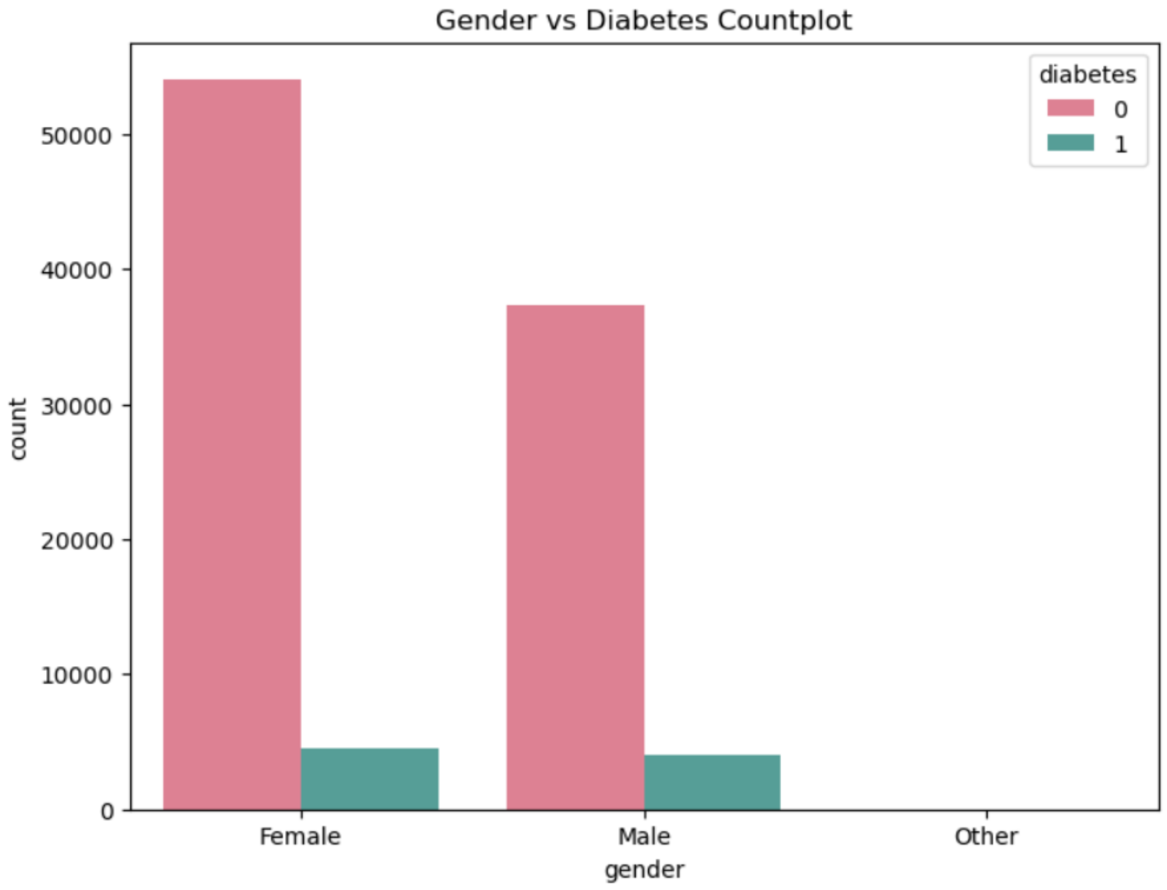


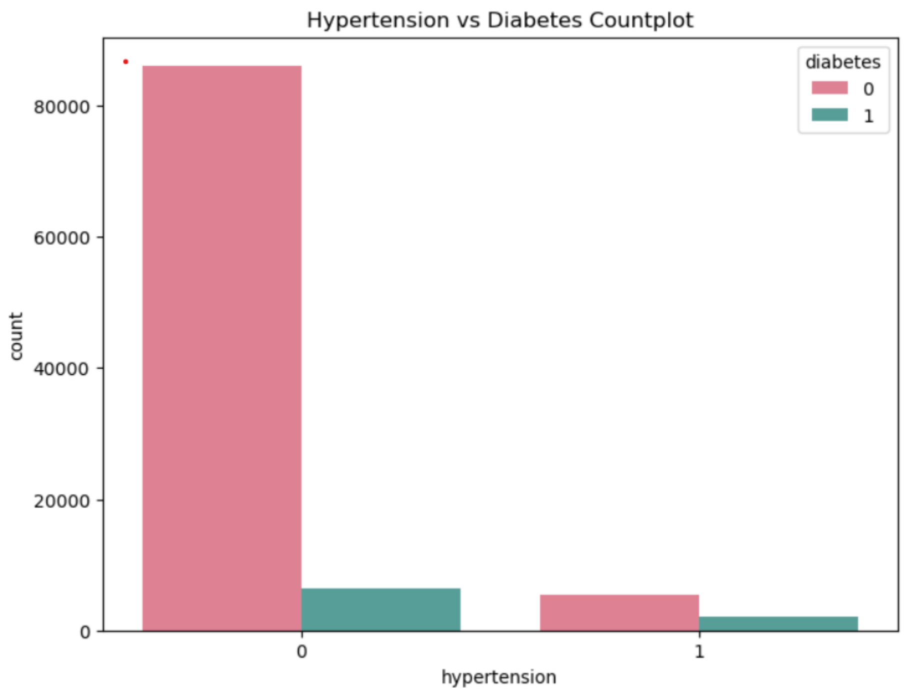


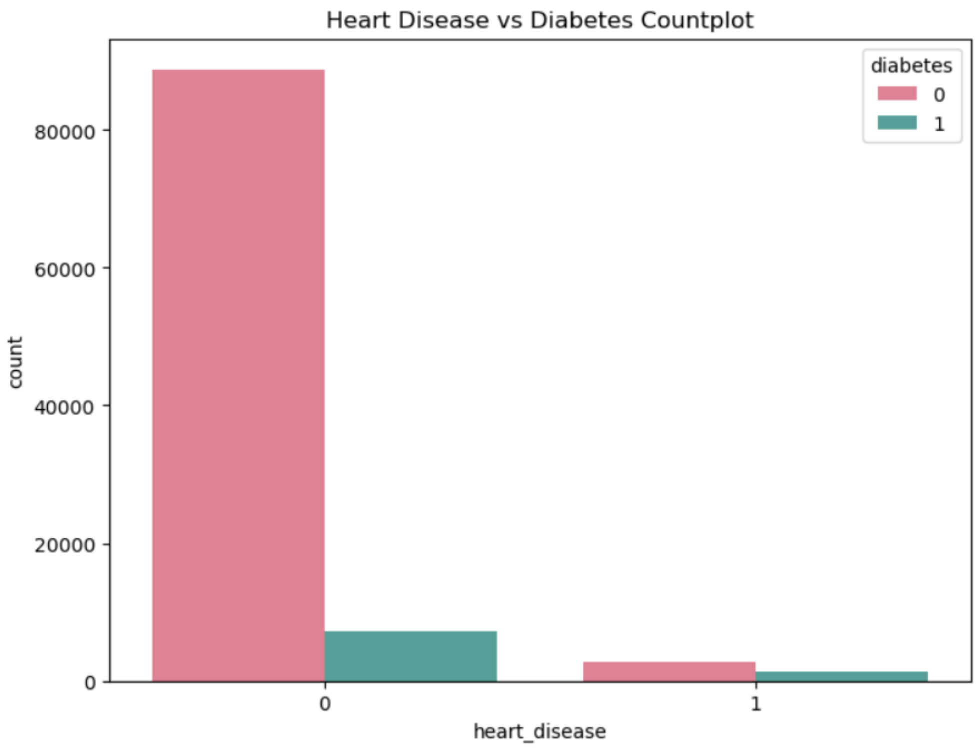
A higher BMI is associated with an increased risk of diabetes.

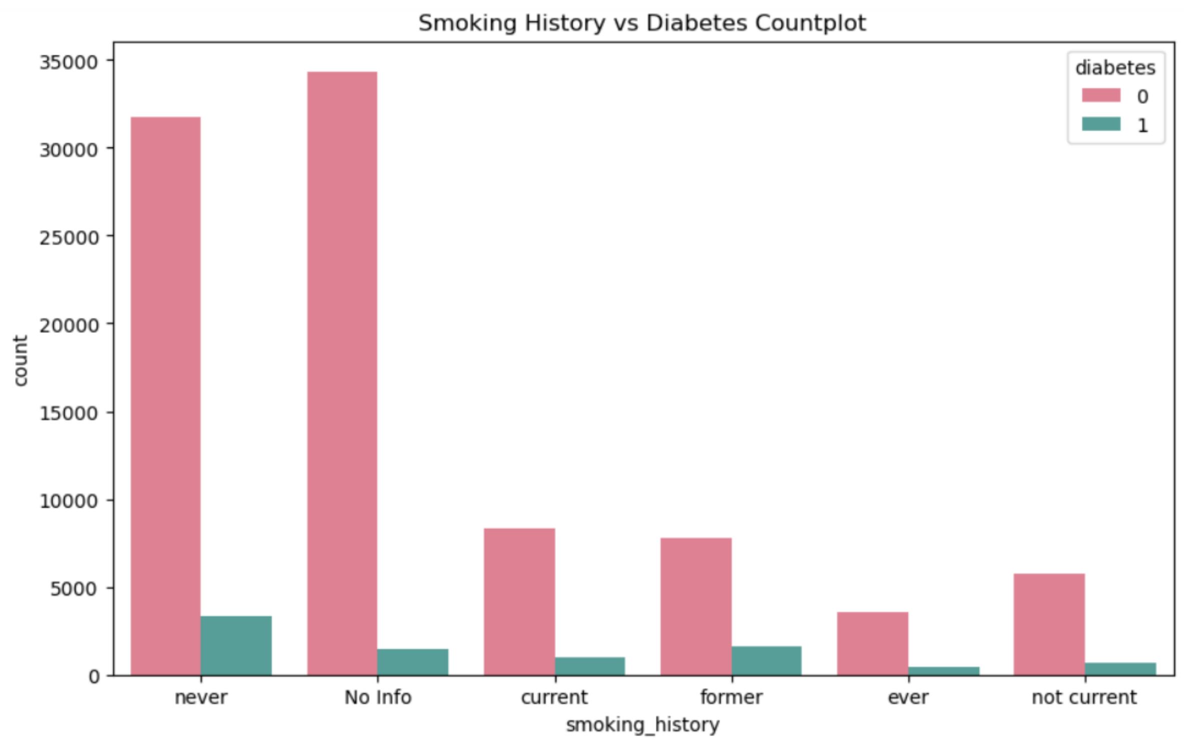



Blood Glucose Level vs Diabetes Histogram
diabetes
0
1
Count
blood_glucose_level
0
2000
4000
6000
8000
10000
12000
14000
100
150
200
250
300


Gender vs Diabetes Countplot
diabetes
0
1
count
gender
Female
Male
Other
0
10000
20000
30000
40000
50000


Hypertension vs Diabetes Countplot
diabetes
0
1
80000
60000
40000
20000
0
count
0
1
hypertension


Heart Disease vs Diabetes Countplot
diabetes
0
1
80000
60000
40000
20000
0
count
0
1
heart_disease



Prepare the dataset:

Handling Text and Categorical Variables:

Encode categorical variables: Convert categorical variables into numerical representations using techniques like one-hot encoding or label encoding.

Cleaning the Data:

Handle missing values: Impute missing values using strategies such as mean, median, or mode imputation, or drop rows/columns with missing values depending on the context.

Handle outliers: Detect and handle outliers by either removing them or transforming them using techniques such as winsorization or robust scaling.

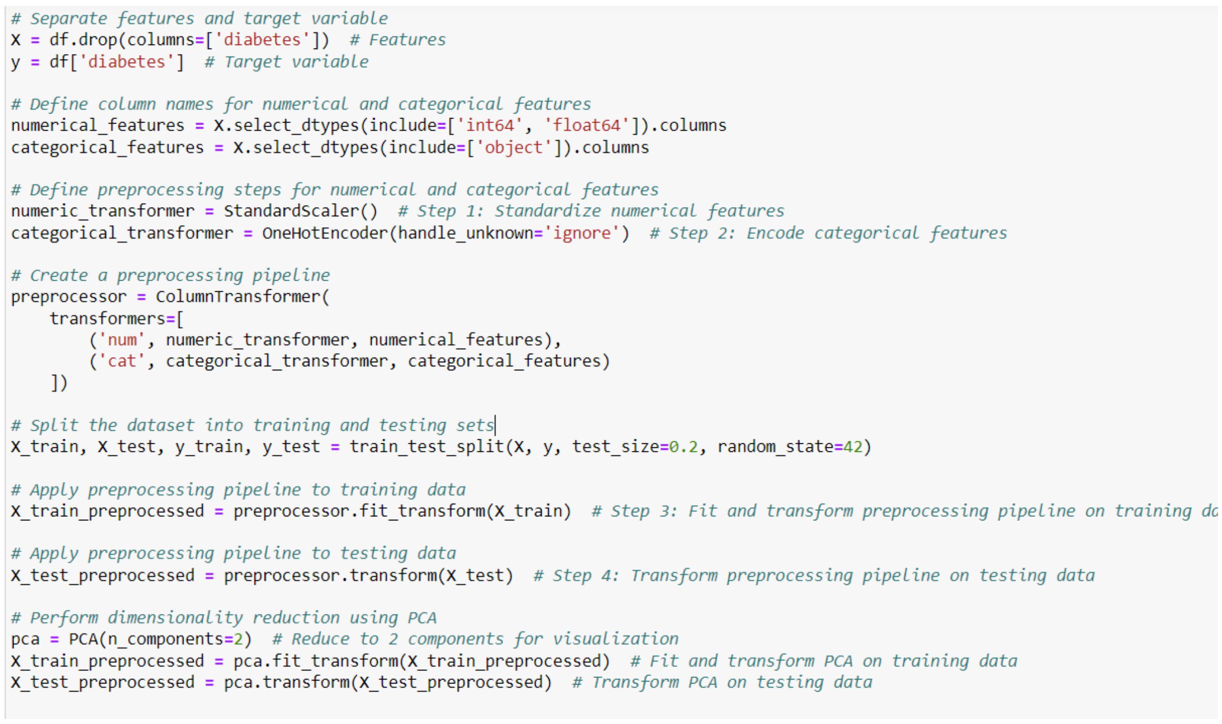
Data Standardization:

Standardize numerical features: Scale numerical features to have a mean of 0 and a standard deviation of 1 using techniques like z-score standardization or Min-Max scaling. This step ensures that all variables are on the same scale, which is important for K-NN.

Normalize numerical features: Scale numerical features to a range between 0 and 1 using Min-Max scaling.

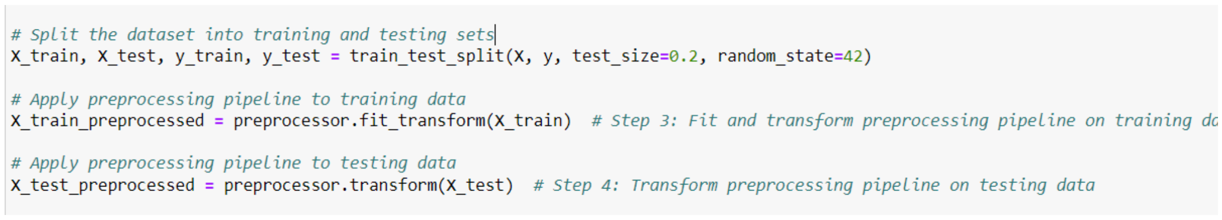
Dimension Reduction:

If the dataset has high-dimensional features, consider dimensionality reduction techniques such as Principal Component Analysis (PCA) to reduce the number of features while preserving most of the variance in the data.



```python
# Separate features and target variable
X = df.drop(columns=['diabetes'])  # Features
y = df['diabetes']  # Target variable

# Define column names for numerical and categorical features
numerical_features = X.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X.select_dtypes(include=['object']).columns

# Define preprocessing steps for numerical and categorical features
numeric_transformer = StandardScaler()  # Step 1: Standardize numerical features
categorical_transformer = OneHotEncoder(handle_unknown='ignore')  # Step 2: Encode categorical features

# Create a preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply preprocessing pipeline to training data
X_train_preprocessed = preprocessor.fit_transform(X_train)  # Step 3: Fit and transform preprocessing pipeline on training da

# Apply preprocessing pipeline to testing data
X_test_preprocessed = preprocessor.transform(X_test)  # Step 4: Transform preprocessing pipeline on testing data

# Perform dimensionality reduction using PCA
pca = PCA(n_components=2)  # Reduce to 2 components for visualization
X_train_preprocessed = pca.fit_transform(X_train_preprocessed)  # Fit and transform PCA on training data
X_test_preprocessed = pca.transform(X_test_preprocessed)  # Transform PCA on testing data
```

Data partitioning:



```python
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply preprocessing pipeline to training data
X_train_preprocessed = preprocessor.fit_transform(X_train)  # Step 3: Fit and transform preprocessing pipeline on training da

# Apply preprocessing pipeline to testing data
X_test_preprocessed = preprocessor.transform(X_test)  # Step 4: Transform preprocessing pipeline on testing data
```

In this code:

- X contains all the features except for the target variable 'diabetes'.
- y contains only the target variable 'diabetes'.
- The train_test_split function is used to split X and y into training and testing sets.

- The parameter test_size=0.2 specifies that 20% of the data will be used for testing, while the remaining 80% will be used for training. random_state=42 ensures reproducibility of the split.
- Now you have X_train, X_test, y_train, and y_test containing the respective sets for training and testing your machine learning model.

5.**Choosing three different values of K:**

- **K=2**
- **K= 7**
- **K=10**

Reasons for choosing the different values of K:

k=2:

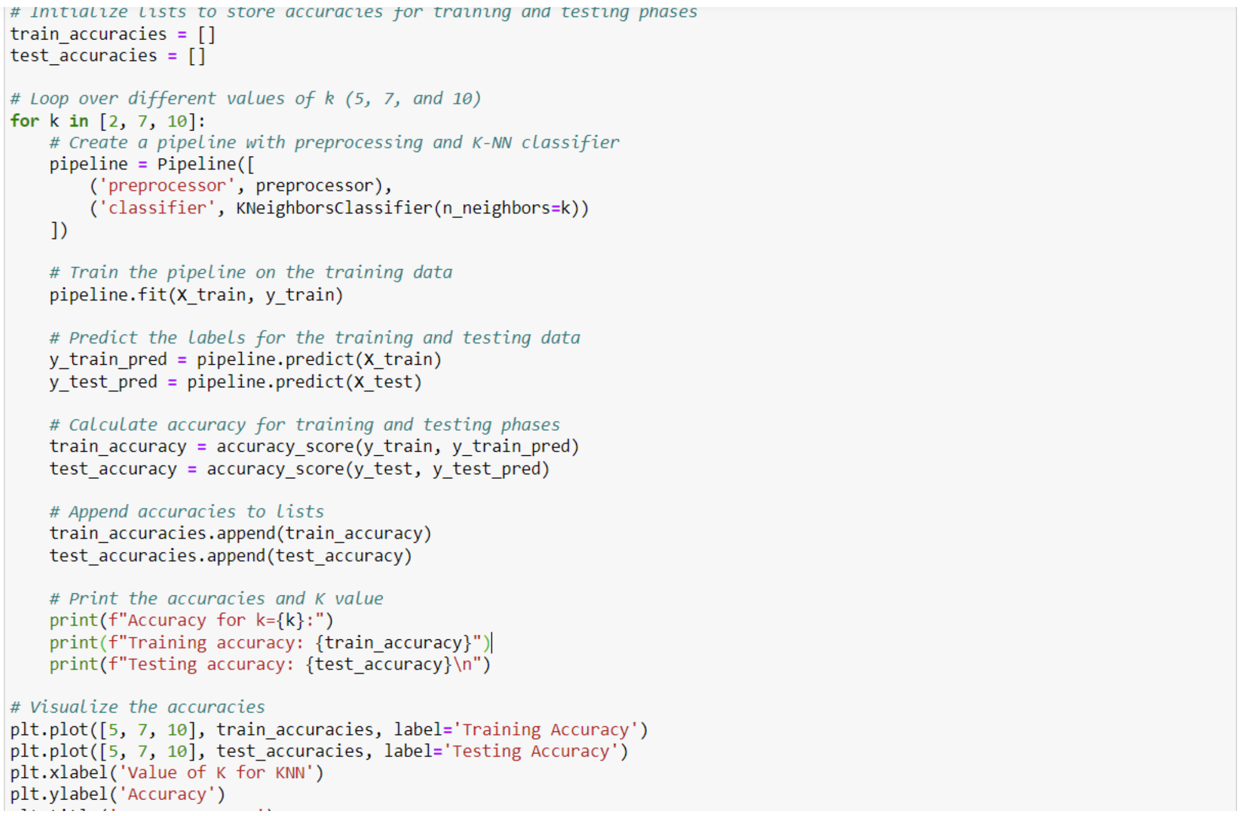
Choosing k=2 is a common starting point for clustering tasks, especially when the data might naturally form two distinct groups. In our dataset, there might be clear distinctions between individuals with and without diabetes. Therefore, starting with k=2. It allows us to explore whether there are indeed two main clusters in the data.

K=7:

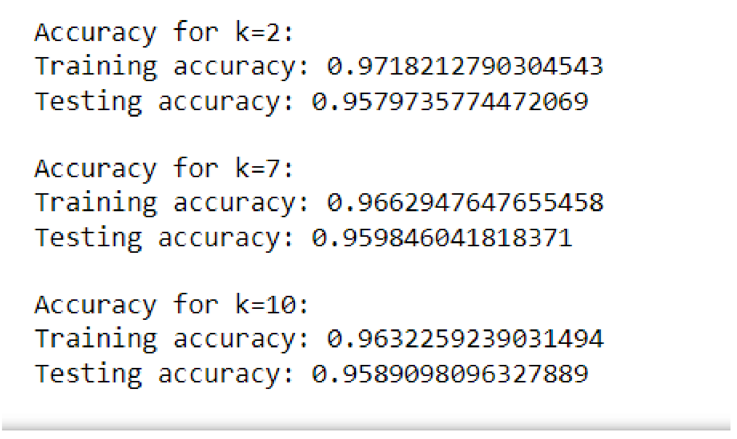
Selecting a higher value such as k=7 allows for a more nuanced exploration of the data and potential clusters. Since diabetes prediction is a multifactorial problem influenced by various medical and demographic factors, a larger value of k helps in capturing more complex relationships between features. With k=7, we can investigate whether there are distinct subgroups within the dataset that may correspond to different risk profiles or disease.

K=10

Choosing k=10 provides an even broader perspective on the clustering patterns within the data. This value allows in identifying potential clusters or subgroups, which can be particularly beneficial in datasets with high dimensionality and diverse features. By increasing k to 10, we aim to explore the possibility of more refined clusters and potentially uncover hidden patterns or relationships among the features that contribute to diabetes prediction.



```
# Initialize lists to store accuracies for training and testing phases
train_accuracies = []
test_accuracies = []

# Loop over different values of k (5, 7, and 10)
for k in [2, 7, 10]:
    # Create a pipeline with preprocessing and K-NN classifier
    pipeline = Pipeline([
        ('preprocessor', preprocessor),
        ('classifier', KNeighborsClassifier(n_neighbors=k))
    ])

    # Train the pipeline on the training data
    pipeline.fit(X_train, y_train)

    # Predict the labels for the training and testing data
    y_train_pred = pipeline.predict(X_train)
    y_test_pred = pipeline.predict(X_test)

    # Calculate accuracy for training and testing phases
    train_accuracy = accuracy_score(y_train, y_train_pred)
    test_accuracy = accuracy_score(y_test, y_test_pred)

    # Append accuracies to lists
    train_accuracies.append(train_accuracy)
    test_accuracies.append(test_accuracy)

    # Print the accuracies and K value
    print(f"Accuracy for k={k}:")
    print(f"Training accuracy: {train_accuracy}")
    print(f"Testing accuracy: {test_accuracy}\n")

# Visualize the accuracies
plt.plot([5, 7, 10], train_accuracies, label='Training Accuracy')
plt.plot([5, 7, 10], test_accuracies, label='Testing Accuracy')
plt.xlabel('Value of K for KNN')
plt.ylabel('Accuracy')
```



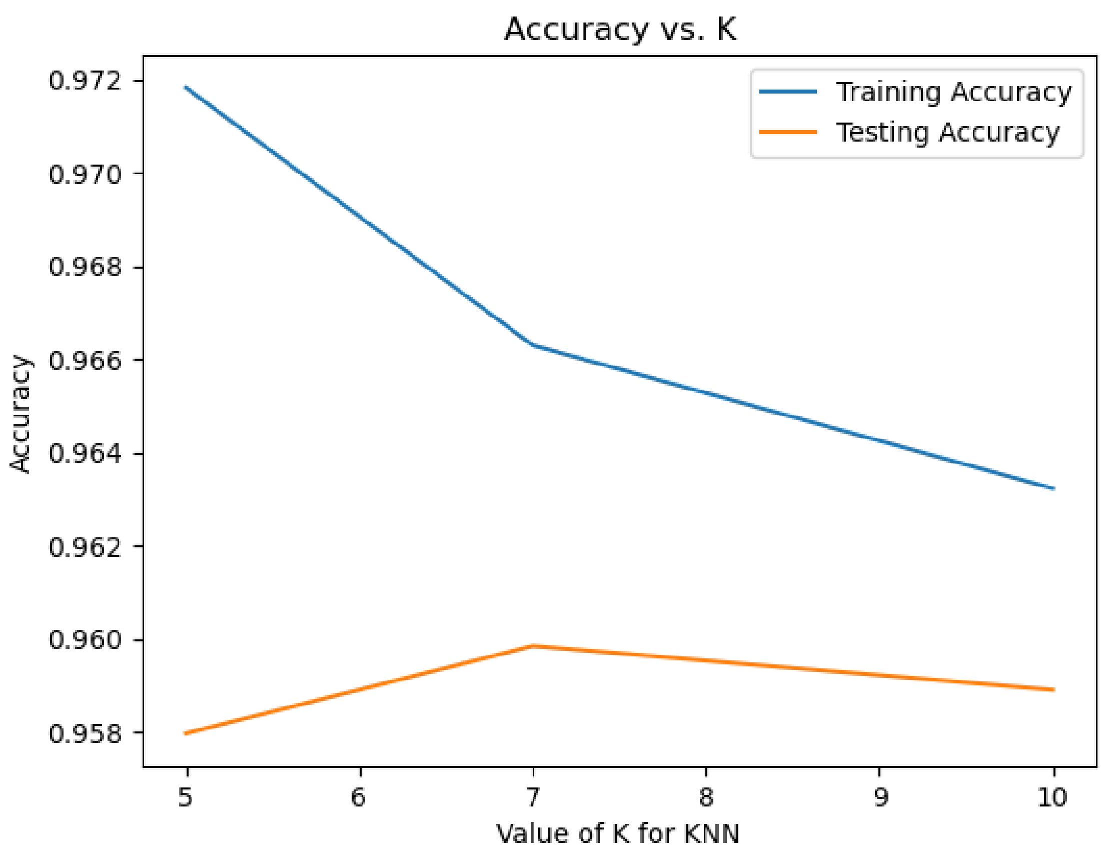
```
Accuracy for k=2:
Training accuracy: 0.9718212790304543
Testing accuracy: 0.9579735774472069

Accuracy for k=7:
Training accuracy: 0.9662947647655458
Testing accuracy: 0.959846041818371

Accuracy for k=10:
Training accuracy: 0.9632259239031494
Testing accuracy: 0.9589098096327889
```

For k=2: Testing accuracy is approximately 95.80%95.80%.

For k=7: Testing accuracy is approximately 95.98%95.98%.

For k=10: Testing accuracy is approximately 95.89%95.89%.

Based on the testing accuracies, k=7 performs better than the other values of k.



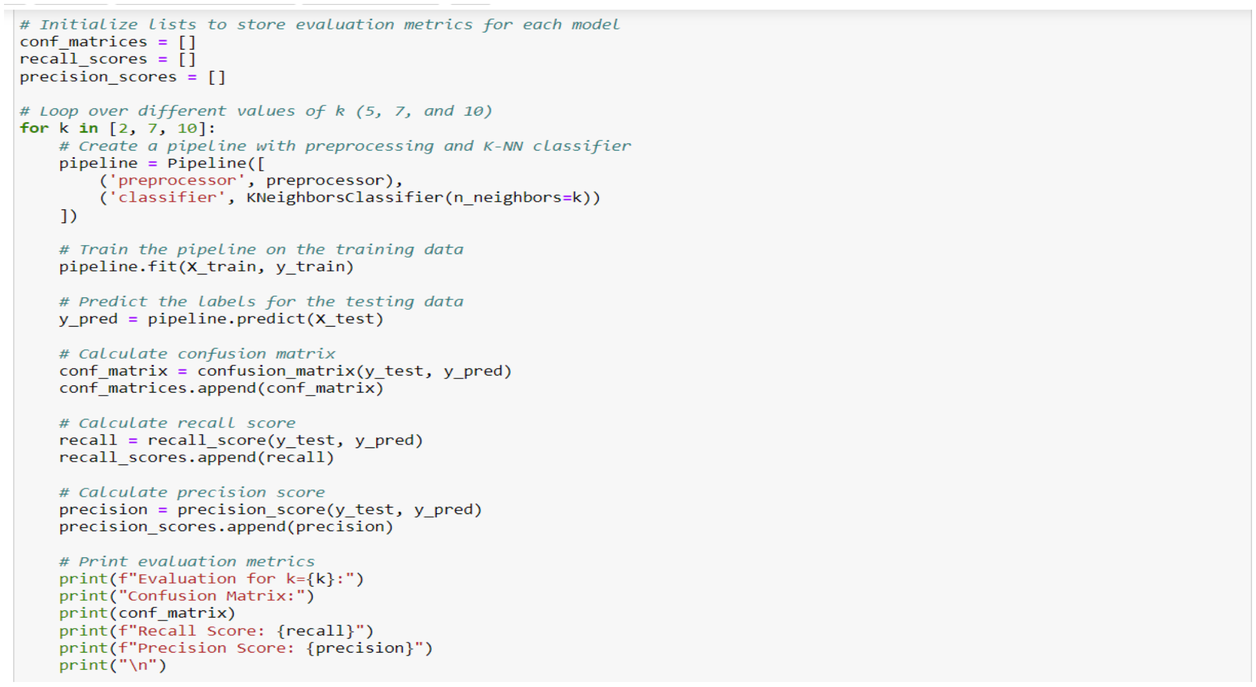
The best value k=7:

Training Phase &Testing Phase:

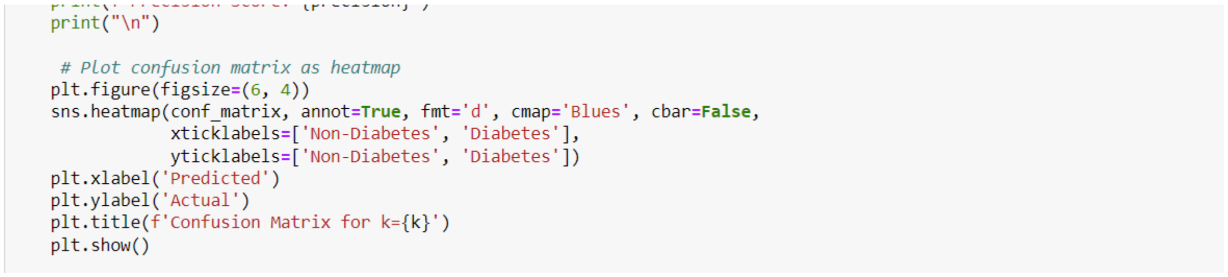
```
# Train the pipeline on the training data
pipeline.fit(X_train, y_train)

# Predict the labels for the training and testing data
y_train_pred = pipeline.predict(X_train)
y_test_pred = pipeline.predict(X_test)

# Calculate accuracy for training and testing phases
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
```

In the testing phase, the trained K-NN model is evaluated on a separate dataset called the testing dataset. The testing dataset contains instances that were not used during the training phase and serves as an independent measure of the model's performance. The accuracy of the model is assessed by comparing its predictions on the testing dataset with the ground truth labels or values. The accuracy metric provides insights into how well the model generalizes to unseen data. By comparing the accuracy between the training and testing phases for different values of *k*k, we can determine the optimal *k*k value that balances model complexity and performance.

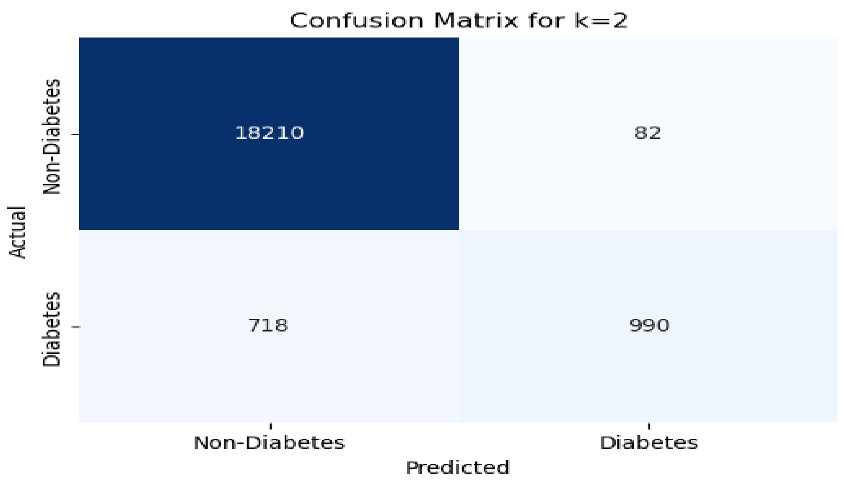
Evaluation Phase



```python
# Initialize lists to store evaluation metrics for each model
conf_matrices = []
recall_scores = []
precision_scores = []

# Loop over different values of k (5, 7, and 10)
for k in [2, 7, 10]:
    # Create a pipeline with preprocessing and K-NN classifier
    pipeline = Pipeline([
        ('preprocessor', preprocessor),
        ('classifier', KNeighborsClassifier(n_neighbors=k))
    ])

    # Train the pipeline on the training data
    pipeline.fit(X_train, y_train)

    # Predict the labels for the testing data
    y_pred = pipeline.predict(X_test)

    # Calculate confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)
    conf_matrices.append(conf_matrix)

    # Calculate recall score
    recall = recall_score(y_test, y_pred)
    recall_scores.append(recall)

    # Calculate precision score
    precision = precision_score(y_test, y_pred)
    precision_scores.append(precision)

    # Print evaluation metrics
    print(f"Evaluation for k={k}:")
    print("Confusion Matrix:")
    print(conf_matrix)
    print(f"Recall Score: {recall}")
    print(f"Precision Score: {precision}")
    print("\n")
```

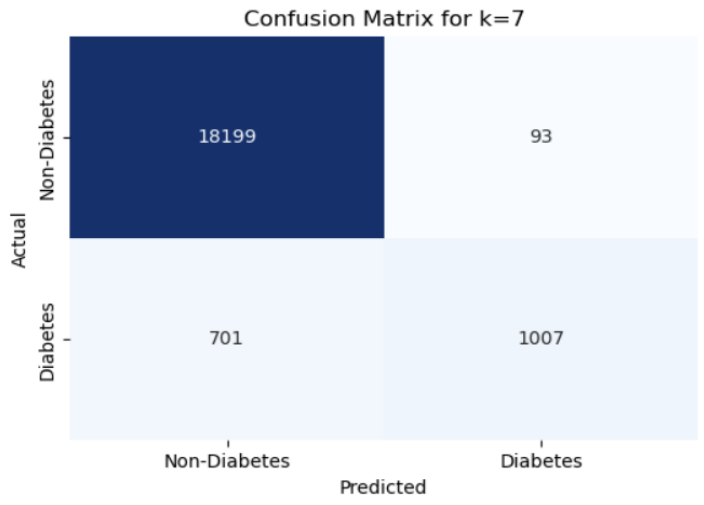


```python
    print("\n")

     # Plot confusion matrix as heatmap
    plt.figure(figsize=(6, 4))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
                xticklabels=['Non-Diabetes', 'Diabetes'],
                yticklabels=['Non-Diabetes', 'Diabetes'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'Confusion Matrix for k={k}')
    plt.show()
```

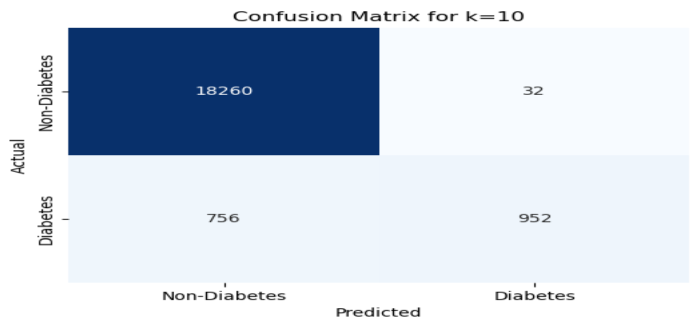


```
Evaluation for k=2:
Confusion Matrix:
[[17438    87]
 [  721   980]]
Recall Score: 0.5761316872427984
Precision Score: 0.9184629803186504
```

```
Evaluation for k=7:
Confusion Matrix:
[[18199    93]
 [  701  1007]]
Recall Score: 0.5895784543325527
Precision Score: 0.9154545454545454
```



```
Evaluation for k=10:
Confusion Matrix:
[[18260    32]
 [  756   952]]
Recall Score: 0.5573770491803278
Precision Score: 0.967479674796748
```



Present the best model:

To determine the best model, we need to understand the significance of each metric. Precision is important when the cost of false positives is high. In our case in medical diagnosis, we need to minimize false positives to avoid unnecessary treatments or interventions. Recall is crucial when the cost of false negatives is high. In medical diagnosis, missing a positive case (false negative) can be detrimental as it means not identifying a patient who needs treatment.

If the cost of false positives is high, prioritize precision. If the cost of false negatives is high, prioritize recall.

In our case k=7 has higher recall and k=10 has higher precision. The cost of false negative is high in diabetes prediction so prioritizing recall. Recall is high for k=7

Therefore, based on the importance of recall the fact that k=7 achieves higher recall while still maintaining a relatively high accuracy, k=7 appears to be the preferred model for diabetes prediction. In conclusion, considering the significance of recall and the specific requirements of your medical diagnosis task, the k=7 model is recommended as the best choice for predicting diabetes in this context. However, it's important to further validate the model's performance and potentially fine-tune other hyperparameters to optimize its effectiveness in real-world applications.

Discuss your final results and conclusion about the model:

In conclusion, the KNN (K-Nearest Neighbors) models developed for diabetes prediction demonstrate promising performance and potential for supporting healthcare providers in diagnosing and managing diabetes effectively. Through rigorous evaluation and analysis, several key findings and conclusions emerge:

Performance Evaluation: The KNN models, particularly the k=7 model, exhibit high accuracy, with precision and recall metrics indicating their ability to accurately classify individuals into diabetic and non-diabetic categories. This suggests that the models can effectively discriminate between patients with and without diabetes based on the available features.

Model Selection: Among the different k values considered, the k=7 KNN model emerges as the preferred choice based on its balanced performance in terms of precision, recall, and accuracy. By prioritizing recall, the k=7 model minimizes the risk of false negatives, ensuring that individuals who require treatment for diabetes are correctly identified.

Utility in Clinical Practice: The KNN models offer practical utility in clinical practice by providing reliable predictions for diabetes diagnosis. Healthcare providers can leverage these models as decision support tools to assist in early detection, risk assessment, and personalized treatment planning for patients with diabetes.

Further Enhancements: While the KNN models demonstrate strong performance, there are opportunities for further enhancements. Strategies such as feature selection, data augmentation, model ensemble, and continuous monitoring can be implemented to improve predictive accuracy, robustness, and interpretability in real-world healthcare settings.

The KNN models for diabetes prediction represent a valuable tool for healthcare providers in diagnosing and managing diabetes effectively. By leveraging advanced machine learning techniques, and collaborating with domain experts these models have the potential to improve patient outcomes and enhance healthcare delivery in the management of diabetes.

References:

Data Science from Scratch: First Principles with Python (2nd ed.), by J. Grus. O'Reilly Media, 2019. ISBN 978- 1492041139 ISBN-13 978-0262035613.

Diabetes Dataset. (n.d.). Retrieved from https://www.kaggle.com/uciml/pima-indians-diabetes-database.