1.0 Introduction

Clustering is a technique of unsupervised learning. It is like classification task in which we have to guess the category of a set of data points. But, unlike classification, we don't have any examples of labels that come with the data points. If we don't know which points are in which cluster, we need to deduce from the data set. This can be achieved using some notion of distance between the data points. Data points in the same cluster are somehow close to each other.

In this study, K-Means clustering is being chosen because of its simplicity and efficiency. K-Means Clustering is a distance-based algorithm for unsupervised learning. Therefore K-Means does not support categorical features. With Unsupervised Learning, this algorithm also can be used to discover clusters in data but with a mathematical approach. Select at least 2 centroids randomly and the distance is calculated between the centroids and all the data points. If the data point is closer to one of the centroids, then it gets labeled as that centroid and vice versa. For the 2 clusters formed, the average value of the data points that are grouped with either of the centroid is calculated and these average values are then defined as new clusters. This process repeats until both centroids can converge to fixed points.

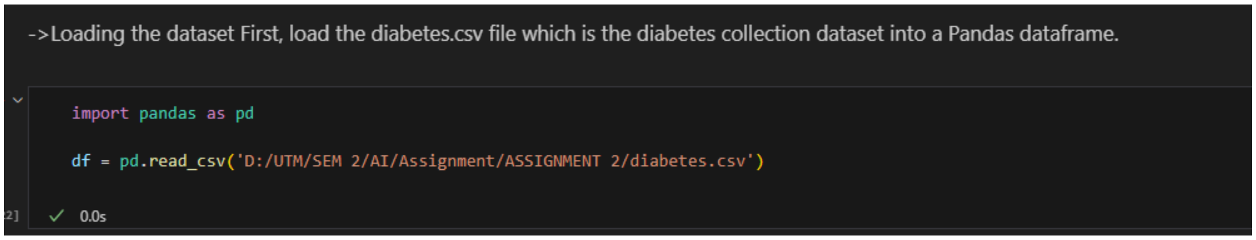
K hyperparameter is introduced to determine the number of clusters or groups the data is to be divided into. Two statistical tests, the Elbow method and Silhouette Score Method, are also used in this study for selecting values of k. Elbow Method is a method that plots the sum of squared error for a range of values of k. If one end of this plot looks like an arm, then k is the value that resembles an elbow is selected. After that elbow value, the sum of squared values (inertia) begins decreasing with steady pace rather than irregularly and thus is considered as an optimal value. It is a way of assessing the quality clustering by how well data points fit together within their clusters and how far away they are from points in other clusters with similar characteristics. This score is found using formula of distance, and highest modeling of k is taken for modelling.

The Diabetes Dataset used in this study includes important health metrics such as glucose, BMI, and insulin, and is relevant for the study of metabolic health. Clustering these kind of medical data can identify clusters of patients sharing similar physiological features, which may help in personalizing treatment strategies or risk assessment.

This report investigates the application of K-means clustering to the diabetes dataset, with the following objectives:

1. To develop an unsupervised learning model that can accurately cluster the patient.
2. Determine the optimal number of clusters using the Elbow Method and Silhouette Score.
3. Interpret cluster characteristics and their alignment with clinical outcomes.

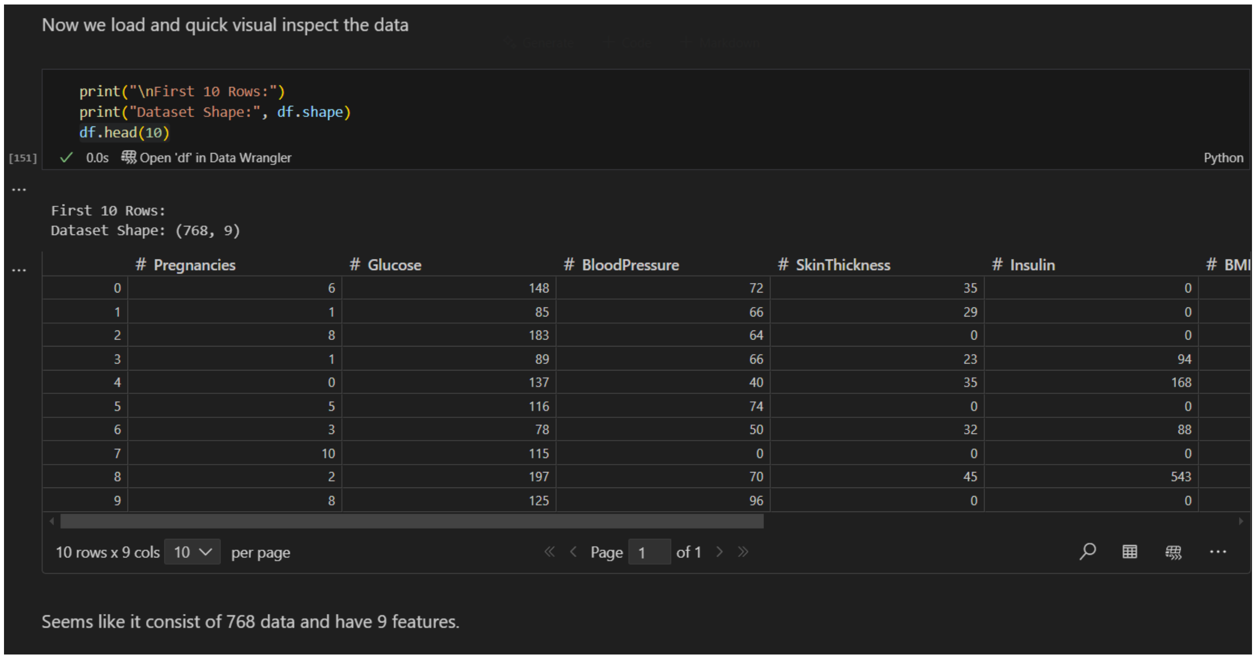
2.0 Implementation


->Loading the dataset First, load the diabetes.csv file which is the diabetes collection dataset into a Pandas dataframe.

```
import pandas as pd

df = pd.read_csv('D:/UTM/SEM 2/AI/Assignment/ASSIGNMENT 2/diabetes.csv')
```

```
✓ 0.0s
```

The first step is to load the diabetes dataset into a pandas DataFrame and conduct a basic inspection to understand its dimensions, data types, and identify immediate issues like missing values.


Now we load and quick visual inspect the data

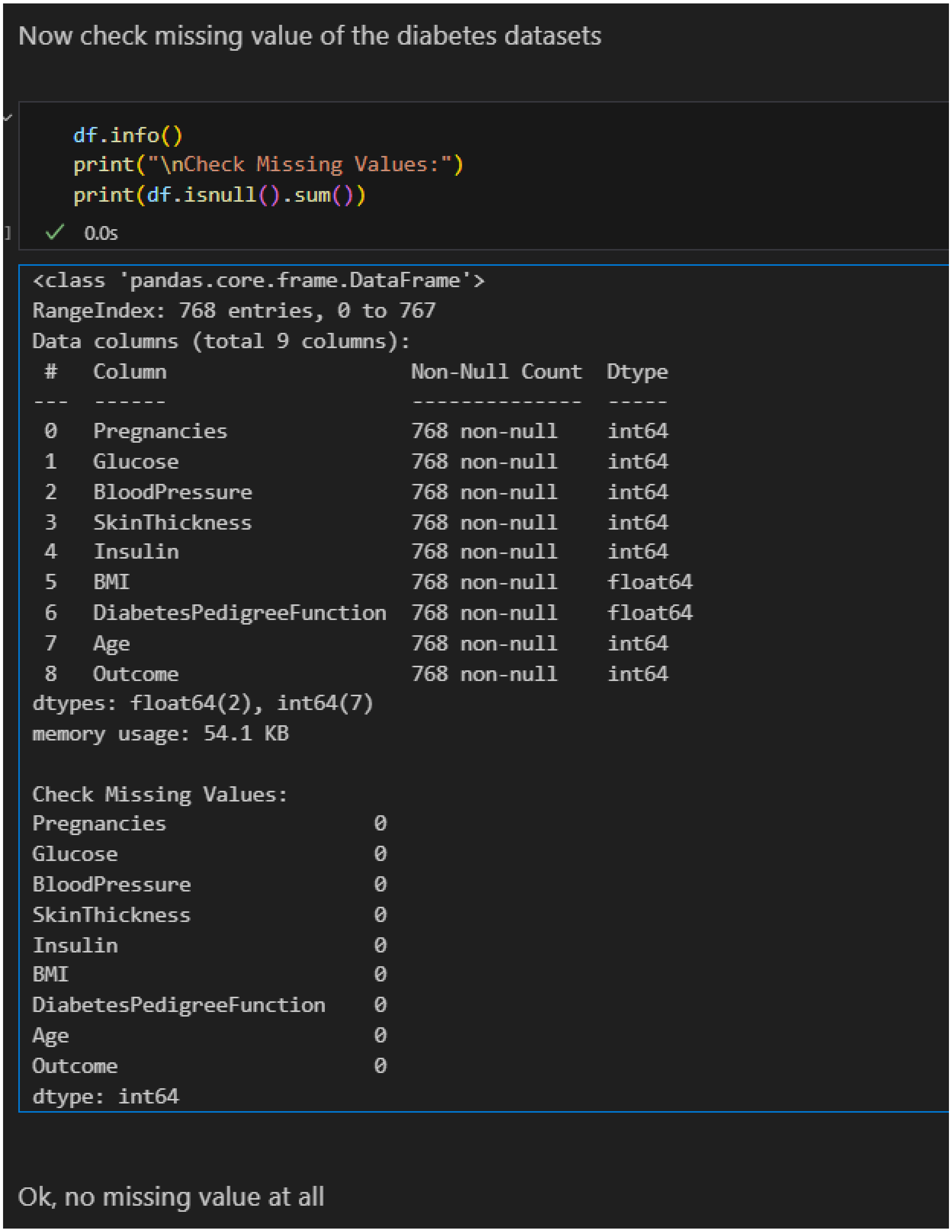
```
print("\nFirst 10 Rows:")
print("Dataset Shape:", df.shape)
df.head(10)
```

```
First 10 Rows:
Dataset Shape: (768, 9)
```

	# Pregnancies	# Glucose	# BloodPressure	# SkinThickness	# Insulin	# BMI
0	6	148	72	35	0	
1	1	85	66	29	0	
2	8	183	64	0	0	
3	1	89	66	23	94	
4	0	137	40	35	168	
5	5	116	74	0	0	
6	3	78	50	32	88	
7	10	115	0	0	0	
8	2	197	70	45	543	
9	8	125	96	0	0	

Seems like it consist of 768 data and have 9 features.

df.head(10) function is used to display the first 10 rows of the DataFrame. It provides a quick snapshot of the dataset's structure.


Now check missing value of the diabetes datasets

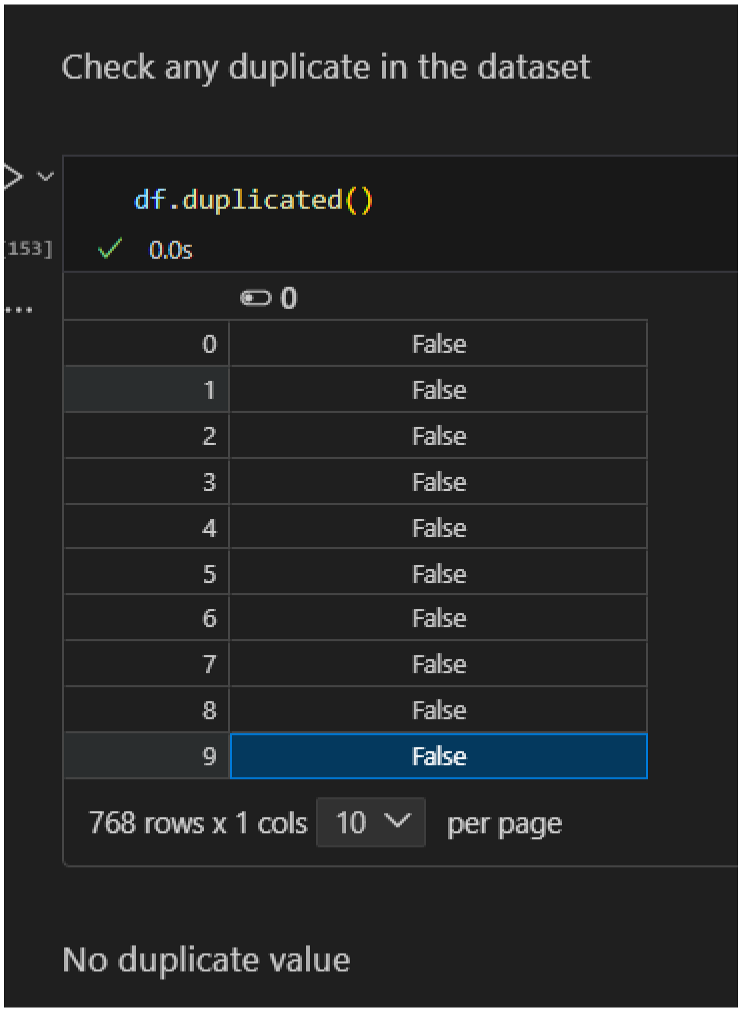
```
df.info()
print("\nCheck Missing Values:")
print(df.isnull().sum())
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB

Check Missing Values:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Ok, no missing value at all

Next, df.info() is used to see the data type of each attribute and df.isnull().sum() function calculates the total number of missing (null) values in each column. From the result, the dataset don't have any missing values.



Check any duplicate in the dataset

```
df.duplicated()
```

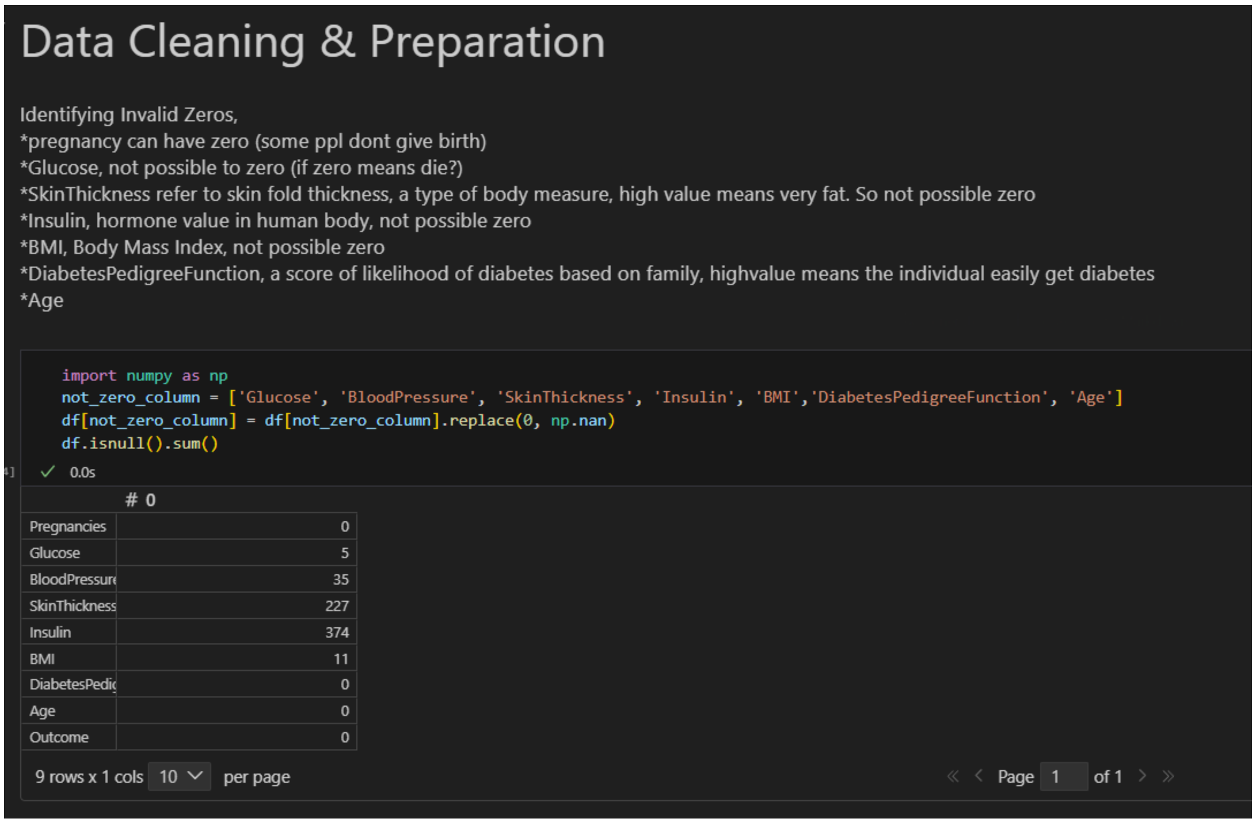
```
[153] ✓ 0.0s
```

	0
0	False
1	False
2	False
3	False
4	False
5	False
6	False
7	False
8	False
9	False

768 rows x 1 cols 10 per page

No duplicate value

df.duplicated() function checks for duplicate rows in the DataFrame. Rach value is True if that row is a duplicate of a previous row, and False otherwise. From the output, there is no duplicate value in the diabetes dataset.



Data Cleaning & Preparation

Identifying Invalid Zeros,
*pregnancy can have zero (some ppl dont give birth)
*Glucose, not possible to zero (if zero means die?)
*SkinThickness refer to skin fold thickness, a type of body measure, high value means very fat. So not possible zero
*Insulin, hormone value in human body, not possible zero
*BMI, Body Mass Index, not possible zero
*DiabetesPedigreeFunction, a score of likelihood of diabetes based on family, highvalue means the individual easily get diabetes
*Age

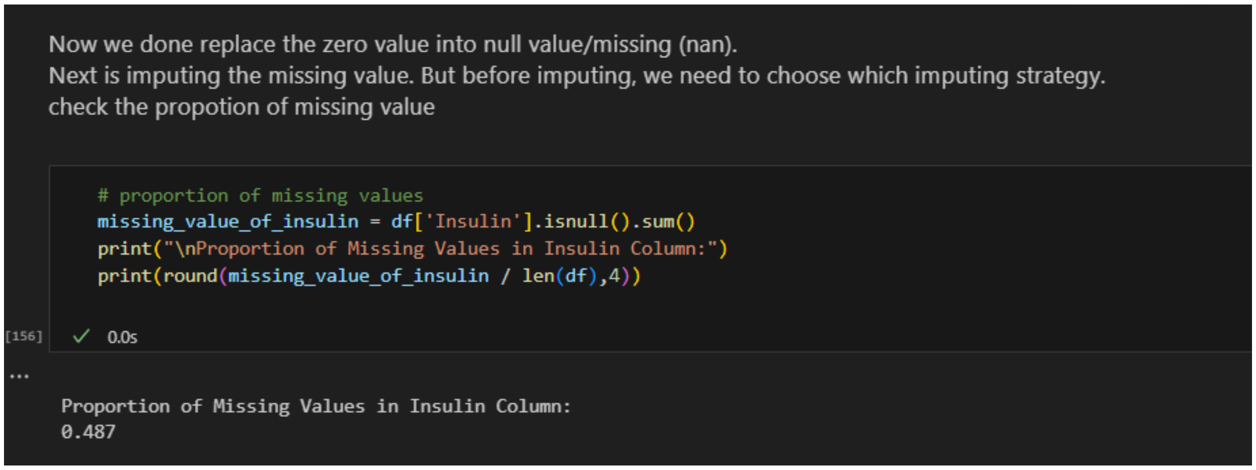
```
import numpy as np
not_zero_column = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI','DiabetesPedigreeFunction', 'Age']
df[not_zero_column] = df[not_zero_column].replace(0, np.nan)
df.isnull().sum()
```

✓ 0.0s

	# 0
Pregnancies	0
Glucose	5
BloodPressure	35
SkinThickness	227
Insulin	374
BMI	11
DiabetesPedic	0
Age	0
Outcome	0

9 rows x 1 cols 10 per page Page 1 of 1

Next is data cleaning process. The list not_zero_column specifies columns where a value of zero is not physiologically meaningful (for example, a glucose or blood pressure reading of zero is not possible for a living person). The next line replaces all zero values in these columns with np.nan, which represents missing values in pandas. This step is important because zeros in these columns likely indicate missing or unrecorded data, not actual measurements. Finally, df.isnull().sum() counts the number of missing values (NaNs) in each column after the replacement.



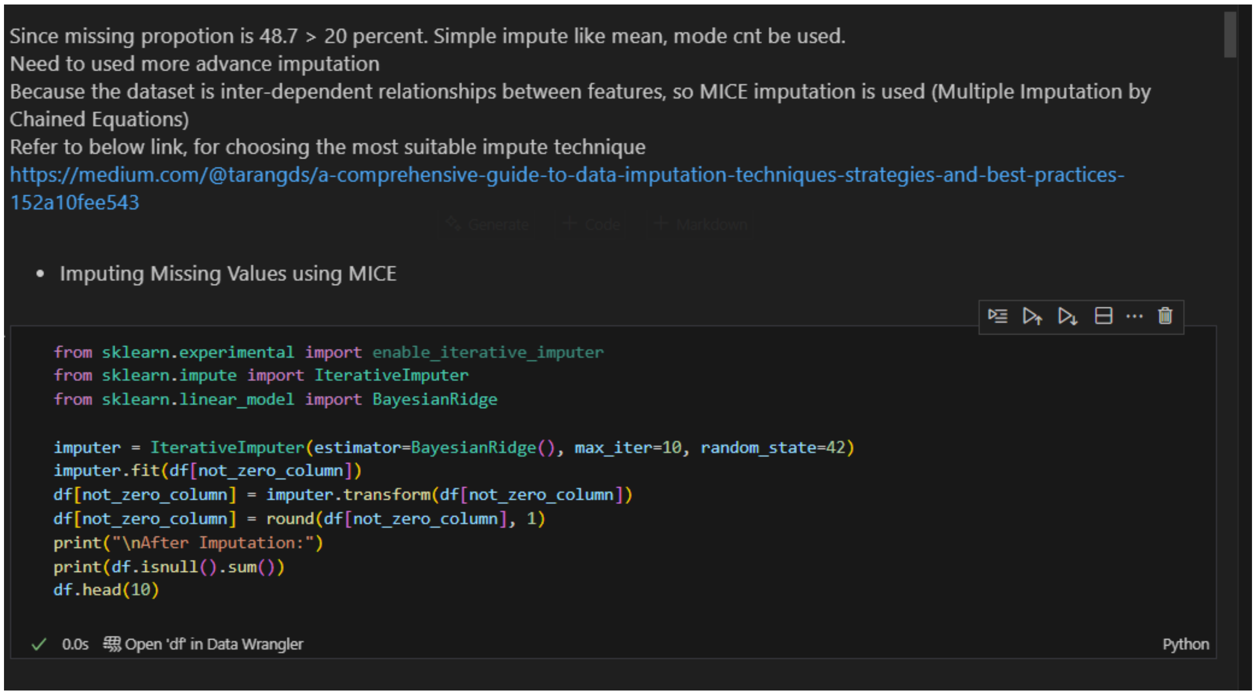
Now we done replace the zero value into null value/missing (nan).
Next is imputing the missing value. But before imputing, we need to choose which imputing strategy.
check the propotion of missing value

```
# proportion of missing values
missing_value_of_insulin = df['Insulin'].isnull().sum()
print("\nProportion of Missing Values in Insulin Column:")
print(round(missing_value_of_insulin / len(df),4))
```

[156] ✓ 0.0s

```
Proportion of Missing Values in Insulin Column:
0.487
```

Next step is replace the missing values (NaNs), but before the replacement we need to see the missing values proportion. In this case, since Insulin attribute has 374 missing values, so it is chosen to see the proportion of missing values. From the output, in insulin column itself have 48.7% missing values.

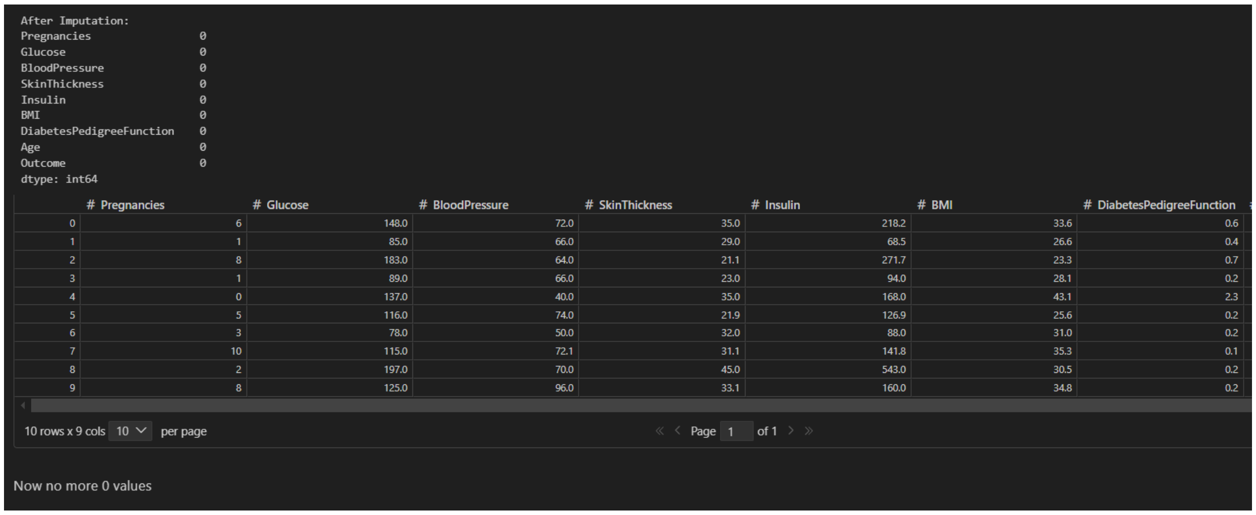


Since missing propotion is 48.7 > 20 percent. Simple impute like mean, mode cnt be used.
Need to used more advance imputation
Because the dataset is inter-dependent relationships between features, so MICE imputation is used (Multiple Imputation by Chained Equations)
Refer to below link, for choosing the most suitable impute technique
https://medium.com/@tarangds/a-comprehensive-guide-to-data-imputation-techniques-strategies-and-best-practices-152a10fee543

- Imputing Missing Values using MICE

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.linear_model import BayesianRidge

imputer = IterativeImputer(estimator=BayesianRidge(), max_iter=10, random_state=42)
imputer.fit(df[not_zero_column])
df[not_zero_column] = imputer.transform(df[not_zero_column])
df[not_zero_column] = round(df[not_zero_column], 1)
print("\nAfter Imputation:")
print(df.isnull().sum())
df.head(10)
```

In this case, Multiple Imputation by Chained Equations (MICE) imputing strategy is used to replace the missing values. For this line “imputer = IterativeImputer(estimator=BayesianRidge() ,max_iter=10, random_state=42)” , run the iteration with 10 times, and choose the seed as 42 for reproducibility and the estimator is using the default which is Bayesian Ridge Regression. Next, imputer.fit(df[not_zero_column]) and df[not_zero_column] = imputer.transform(df[not_zero_column]) is used to learn the relationship between the features and non missing data and use these relationships to predict and fill the missing value. The process is work iteratively and treat each feature as target variables. Next is round off the imputed values to one decimal place for better interpretability.



```
After Imputation:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

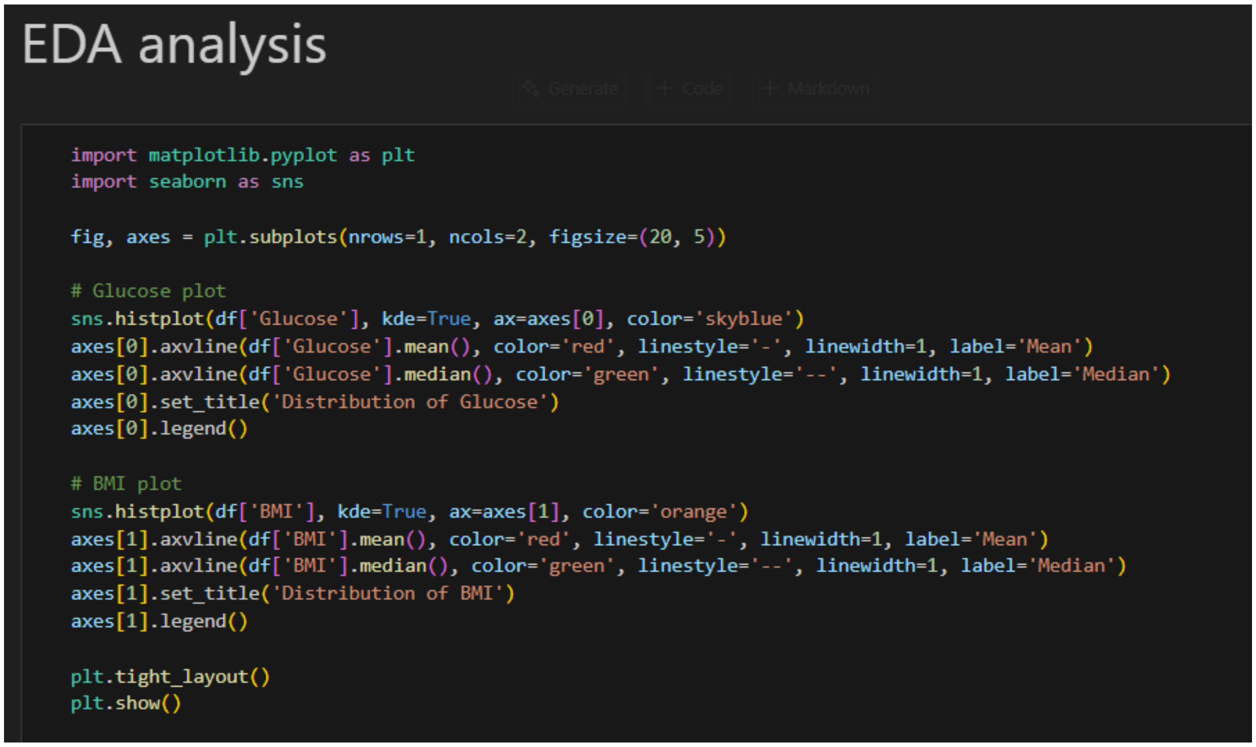
	# Pregnancies	# Glucose	# BloodPressure	# SkinThickness	# Insulin	# BMI	# DiabetesPedigreeFunction
0	6	148.0	72.0	35.0	218.2	33.6	0.6
1	1	85.0	66.0	29.0	68.5	26.6	0.4
2	8	183.0	64.0	21.1	271.7	23.3	0.7
3	1	89.0	66.0	23.0	94.0	28.1	0.2
4	0	137.0	40.0	35.0	168.0	43.1	2.3
5	5	116.0	74.0	21.9	126.9	25.6	0.2
6	3	78.0	50.0	32.0	88.0	31.0	0.2
7	10	115.0	72.1	31.1	141.8	35.3	0.1
8	2	197.0	70.0	45.0	543.0	30.5	0.2
9	8	125.0	96.0	33.1	160.0	34.8	0.2

10 rows x 9 cols 10 per page Page 1 of 1

Now no more 0 values

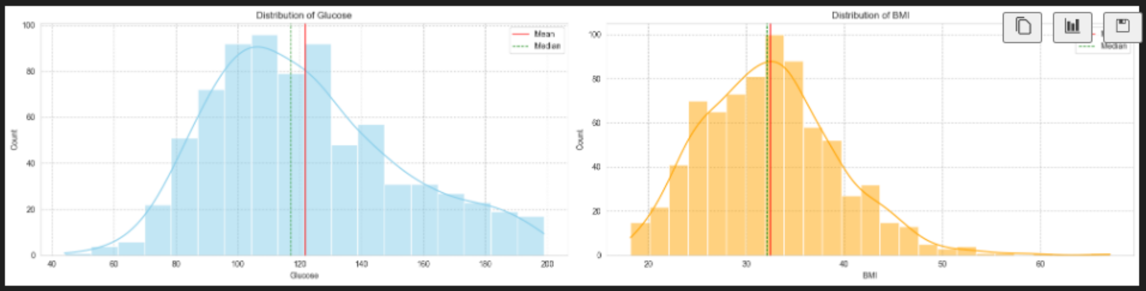
After this process, the columns in not_zero_column should have zero missing values, confirming that the MICE imputation was successful.

2.1 Exploratory Data Analysis



EDA analysis

```python
import matplotlib.pyplot as plt
import seaborn as sns

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20, 5))

# Glucose plot
sns.histplot(df['Glucose'], kde=True, ax=axes[0], color='skyblue')
axes[0].axvline(df['Glucose'].mean(), color='red', linestyle='-', linewidth=1, label='Mean')
axes[0].axvline(df['Glucose'].median(), color='green', linestyle='--', linewidth=1, label='Median')
axes[0].set_title('Distribution of Glucose')
axes[0].legend()

# BMI plot
sns.histplot(df['BMI'], kde=True, ax=axes[1], color='orange')
axes[1].axvline(df['BMI'].mean(), color='red', linestyle='-', linewidth=1, label='Mean')
axes[1].axvline(df['BMI'].median(), color='green', linestyle='--', linewidth=1, label='Median')
axes[1].set_title('Distribution of BMI')
axes[1].legend()

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



The distribution of gluscose is right skewed means some individuals have abnormally high glucose levels. The mode falls near to the normal range of glucose(100–130).

BMI distribution also right skewed, means some portion is overweight. The mode falls between 30-35 means obesity is the most common metabolic state in this dataset.

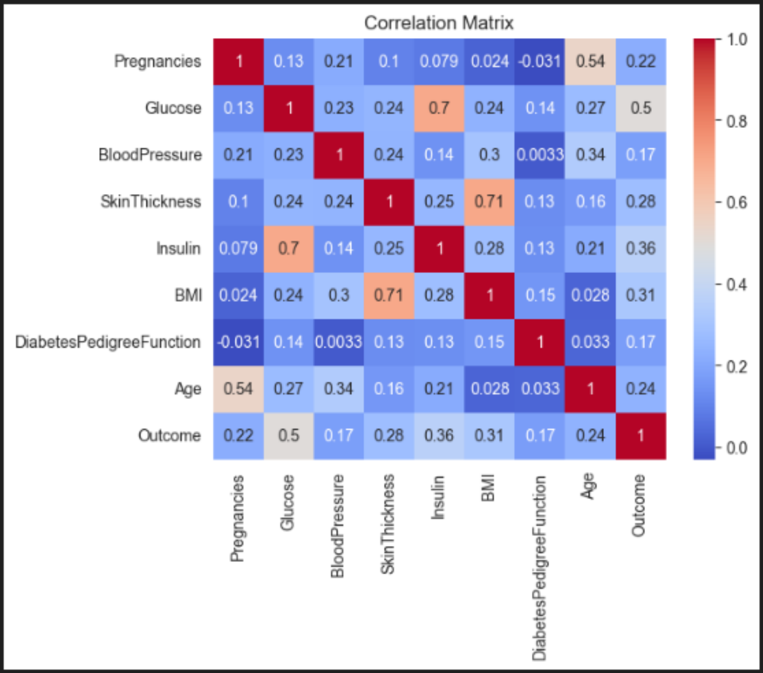
The distribution plot of glucose show a right skew. The median green dash line is around 110, the mean red line is around 120. Since mean>median indicates it is right skew and it has long right tail means some individual is has very high glucose level.

The distribution plot of BMI also shows a right skew. The median green dash line is at 32bmi and mean red line is at 33bmi, thus it shows a right skewed plot since mean>median. From the plot, we can notice that many individuals are in overweight(25-30) and obese (>30). The concentration in higher ranges suggests obesity might be common among this dataset population.

Both distributions being right-skewed indicates this two factor may be potential risk factors for diabetes in this population. So, in next step we plot the correlation matrix to see their relationship among the feature.

```
# Correlation Matrix

corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.figure(figsize=(10,10))
plt.tight_layout()
```

✓ 0.3s



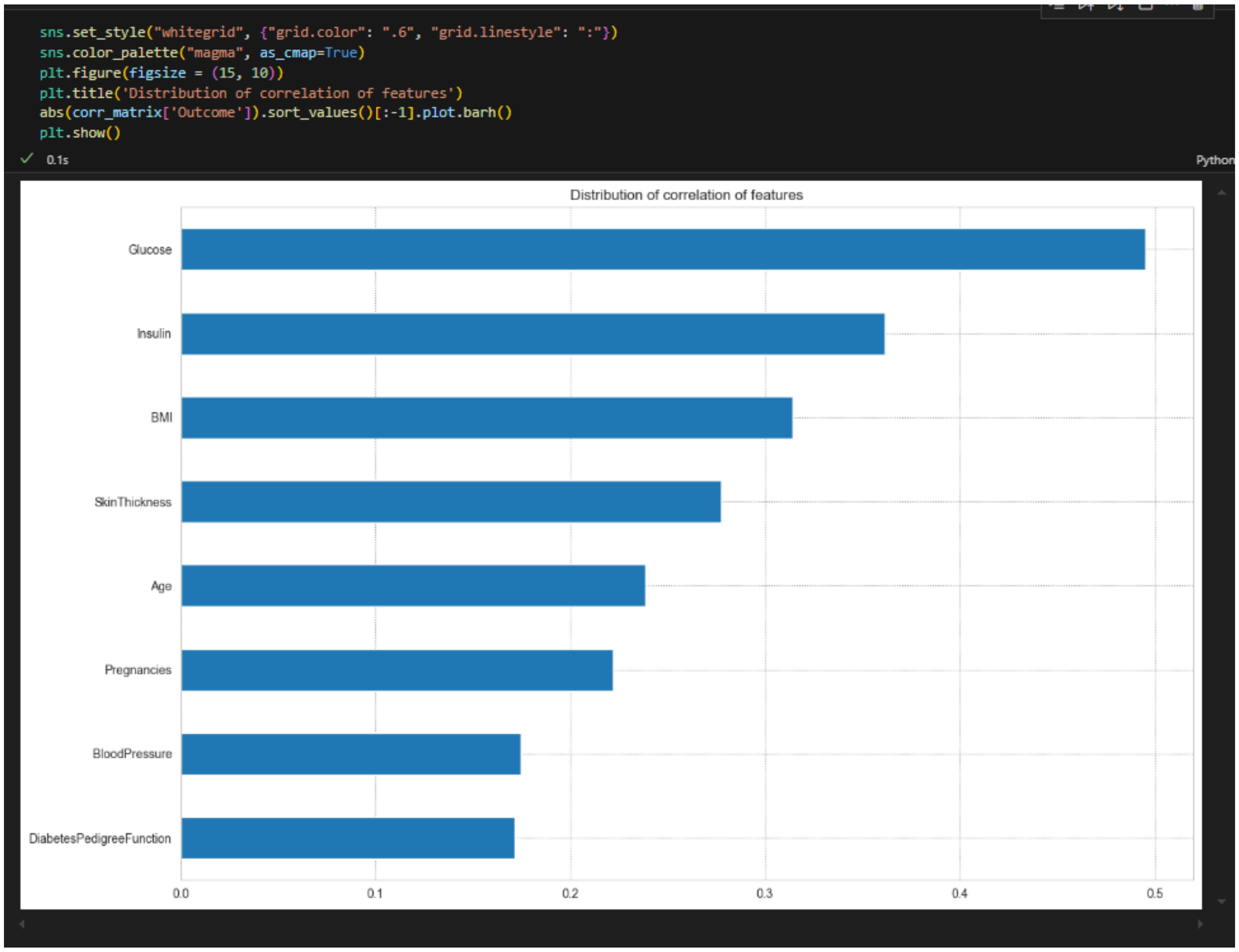
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1	0.13	0.21	0.1	0.079	0.024	-0.031	0.54	0.22
Glucose	0.13	1	0.23	0.24	0.7	0.24	0.14	0.27	0.5
BloodPressure	0.21	0.23	1	0.24	0.14	0.3	0.0033	0.34	0.17
SkinThickness	0.1	0.24	0.24	1	0.25	0.71	0.13	0.16	0.28
Insulin	0.079	0.7	0.14	0.25	1	0.28	0.13	0.21	0.36
BMI	0.024	0.24	0.3	0.71	0.28	1	0.15	0.028	0.31
DiabetesPedigreeFunction	-0.031	0.14	0.0033	0.13	0.13	0.15	1	0.033	0.17
Age	0.54	0.27	0.34	0.16	0.21	0.028	0.033	1	0.24
Outcome	0.22	0.5	0.17	0.28	0.36	0.31	0.17	0.24	1

```
<Figure size 1000x1000 with 0 Axes>
```

The strongest positive correlation with Outcome is observed for Glucose (0.50). Common sense, high blood glucose levels always lead to diabetes
Insulin is the second strong correlation to the outcome (0.36)
BMI (Body Mass Index) steps forward next with a correlation of 0.31 with Outcome
Moderate positive correlations -> Age (0.24), SkinThickness (0.28), pregnancies (0.22)
Weak positive correlation-> Blood pressure (0.17), DiabetesPedigreeFunction (0.17)

An analysis of the correlation matrix shows a distinct association between clinical features and diabetes outcomes. The highest positive correlation with 'Outcome' is measured for Glucose (0.50), which is indicative of its importance as a highly utilized diagnostic factor in diabetes. Also as expected high glucose level in blood is a characteristic of diabetes, and this strong positive relation makes it a protagonist in what follows. Insulin is a second strong correlation (0.36) with outcome.

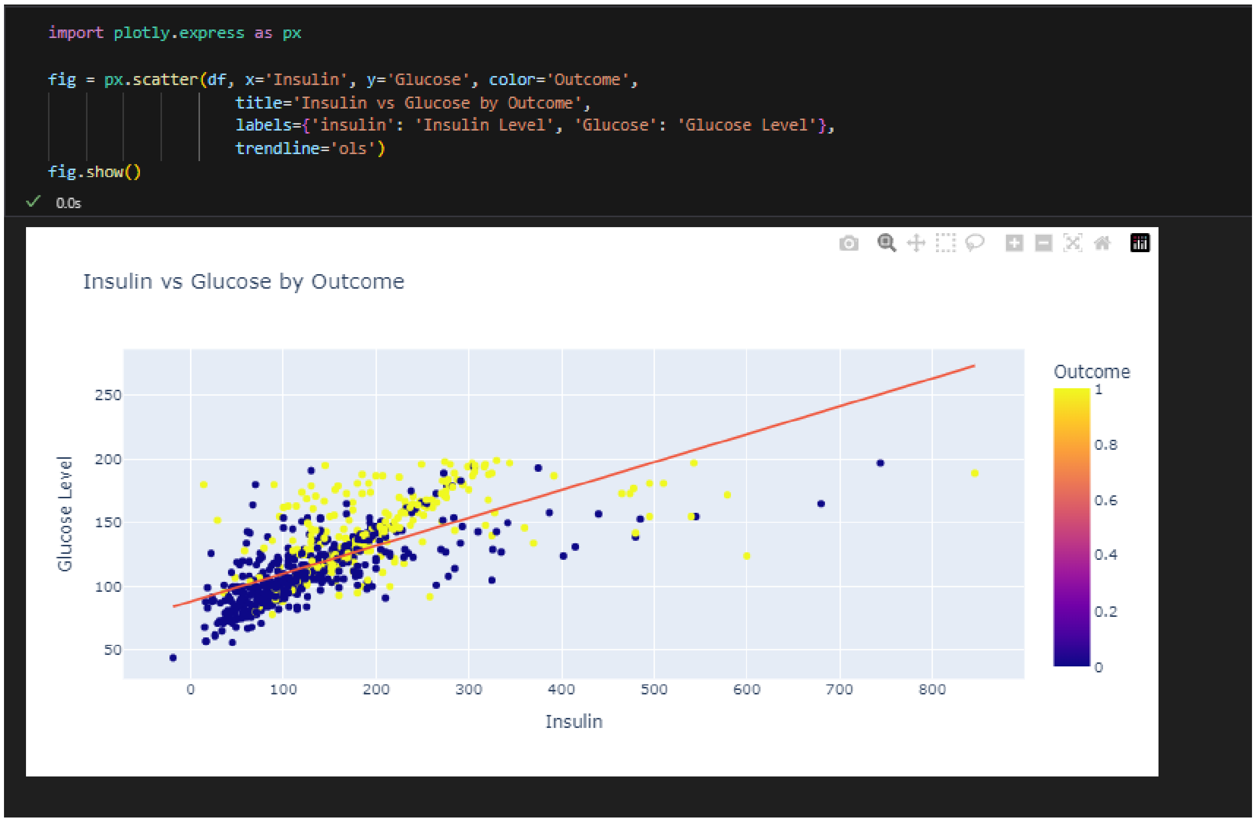
BMI (Body Mass Index) steps forward next with a correlation of 0.31 with "Outcome", reinforcing the relationship between obesity and diabetes risk. Moderate positive correlations are also evident for SkinThickness(0.28), Age (0.24) and Pregnancies (0.22), suggesting a progressive increase in diabetes likelihood with SkinThickness, older age and pregnancy. Skin Thickness and BMI hold hands tightly with a correlation of 0.71, suggesting a biological link between body fat and skin fold measurements. Similarly, Insulin and Glucose (0.7) align closely, which reflects metabolic processes tied to insulin resistance. Age shares high correlation with pregnancy and moderate connections with Blood Pressure (0.34) and Glucose (0.27), showing how age often brings metabolic changes.



```
sns.set_style("whitegrid", {"grid.color": ".6", "grid.linestyle": ":"})
sns.color_palette("magma", as_cmap=True)
plt.figure(figsize = (15, 10))
plt.title('Distribution of correlation of features')
abs(corr_matrix['Outcome']).sort_values()[:-1].plot.barh()
plt.show()
```

The horizontal bar chart further confirms the interpretation from the correlation matrix where glucose ranks first place, and it is the most influential feature with a correlation close to 0.5 followed by Insulin with a correlation of 0.35 and BMI is the third place with 0.31. Next is Skin

Thickness, Age and Pregnancies showing moderate correlations. Trailing just behind are Blood Pressure and Diabetes Pedigree Function

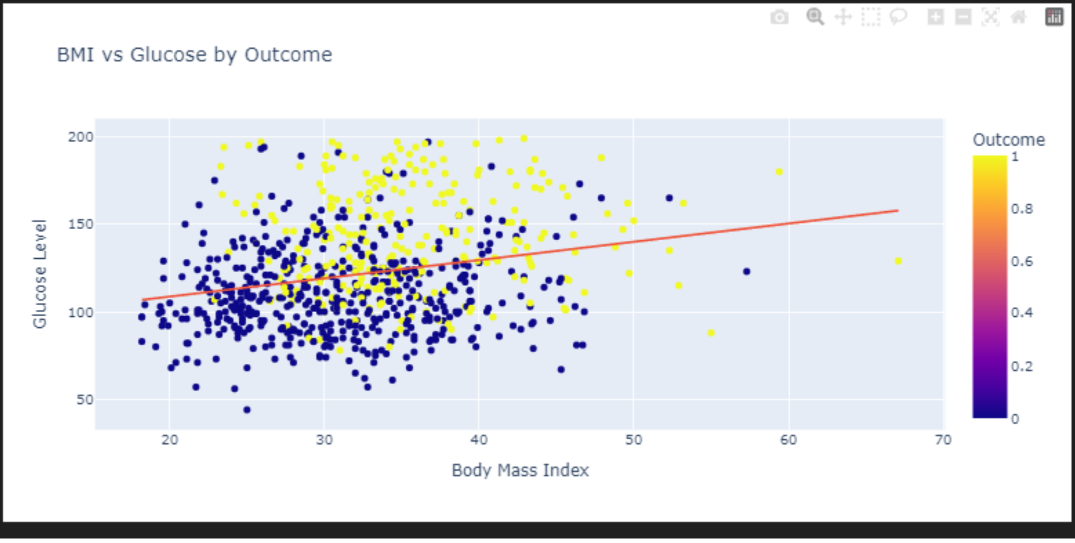


```
import plotly.express as px

fig = px.scatter(df, x='Insulin', y='Glucose', color='Outcome',
                 title='Insulin vs Glucose by Outcome',
                 labels={'insulin': 'Insulin Level', 'Glucose': 'Glucose Level'},
                 trendline='ols')
fig.show()
```

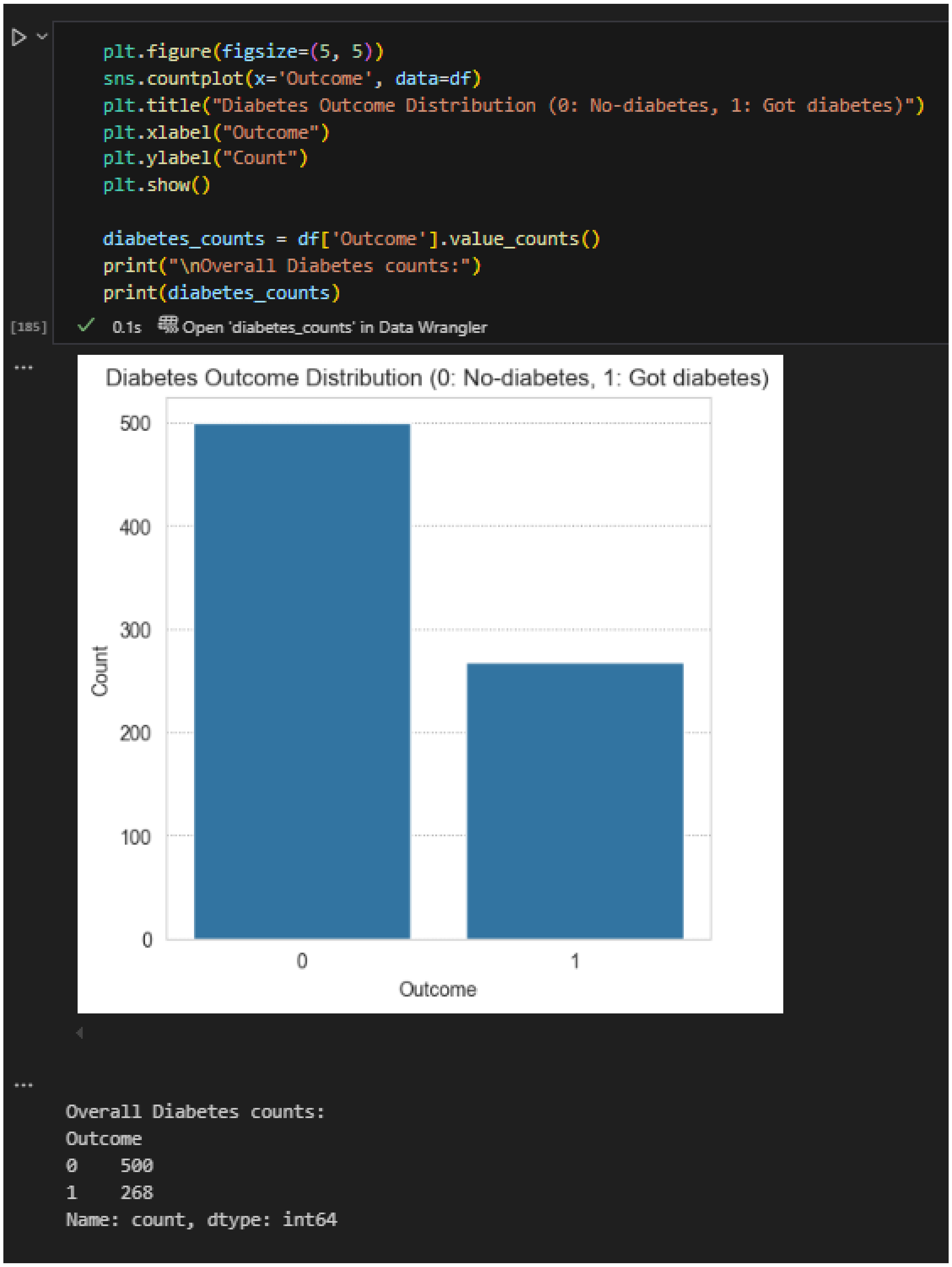
```
✓ 0.0s
```

The scatter plot illustrates the relationship between Insulin and Glucose levels, stratified by diabetes diagnosis (Outcome). A positive association is observed between Insulin and Glucose, with the OLS trendline indicating a general upward trend where Insulin increases, Glucose levels tend to rise. The plot shows two diabetic patient types first is insulin-resistant (high glucose + high insulin) and 2nd insulin-deficient (high glucose + low insulin). Most non-diabetic individuals (blue) show a moderate range of glucose (70-130) and insulin hormone level(30-120).

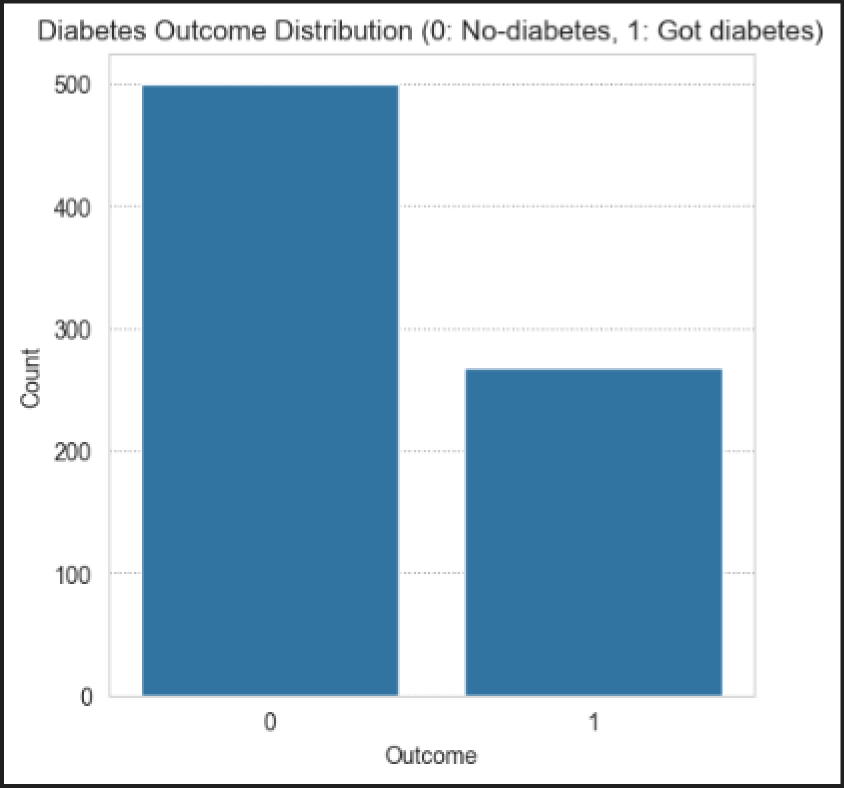
```
fig = px.scatter(df, x='BMI', y='Glucose', color='Outcome',
                 title='BMI vs Glucose by Outcome',
                 labels={'BMI': 'Body Mass Index', 'Glucose': 'Glucose Level'},
                 trendline='ols')
fig.show()
```
✓ 0.0s



The scatter plot illustrates the relationship between Body Mass Index (BMI) and Glucose levels, stratified by diabetes diagnosis (Outcome). A positive association is observed between BMI and Glucose, with the OLS trendline indicating a general upward trend where BMI increases, Glucose levels tend to rise. Notably, the distribution of outcomes reveals clustering patterns. People with diabetes (Outcome = 1) mostly have higher BMI (above 30) and higher glucose levels (above 125). On the other hand, people without diabetes (Outcome = 0) usually have lower BMI and glucose levels. However, some points overlap in the middle, meaning some people with moderate BMI and glucose still have diabetes, while others don't. This tells us that while high BMI and glucose increase the risk of diabetes, other factors also play a role.



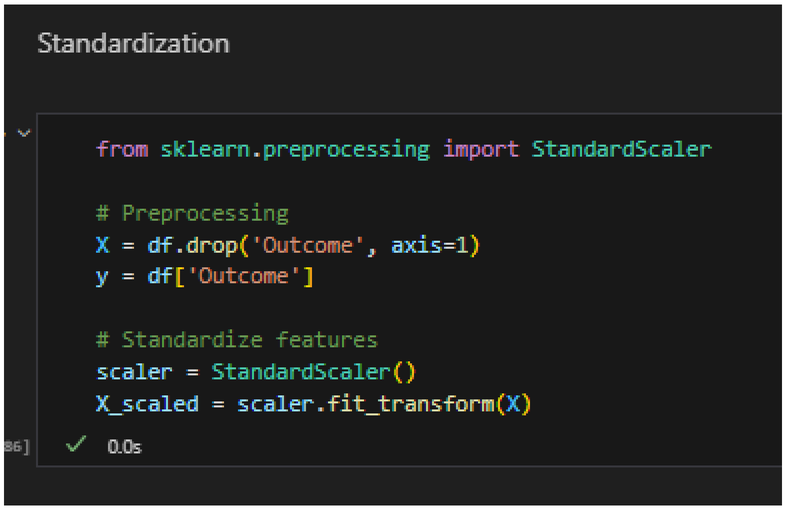
```
plt.figure(figsize=(5, 5))
sns.countplot(x='Outcome', data=df)
plt.title("Diabetes Outcome Distribution (0: No-diabetes, 1: Got diabetes)")
plt.xlabel("Outcome")
plt.ylabel("Count")
plt.show()

diabetes_counts = df['Outcome'].value_counts()
print("\nOverall Diabetes counts:")
print(diabetes_counts)
```



```
Overall Diabetes counts:
Outcome
0    500
1    268
Name: count, dtype: int64
```

The bar graph and numbers show how many people in the dataset have diabetes (1) and how many don’t (0). We can see that more people are non-diabetic (500 people) compared to diabetic (268 people). This means in this group, about twice as many people don’t have diabetes as those who do.

3.0 Unsupervised Learning



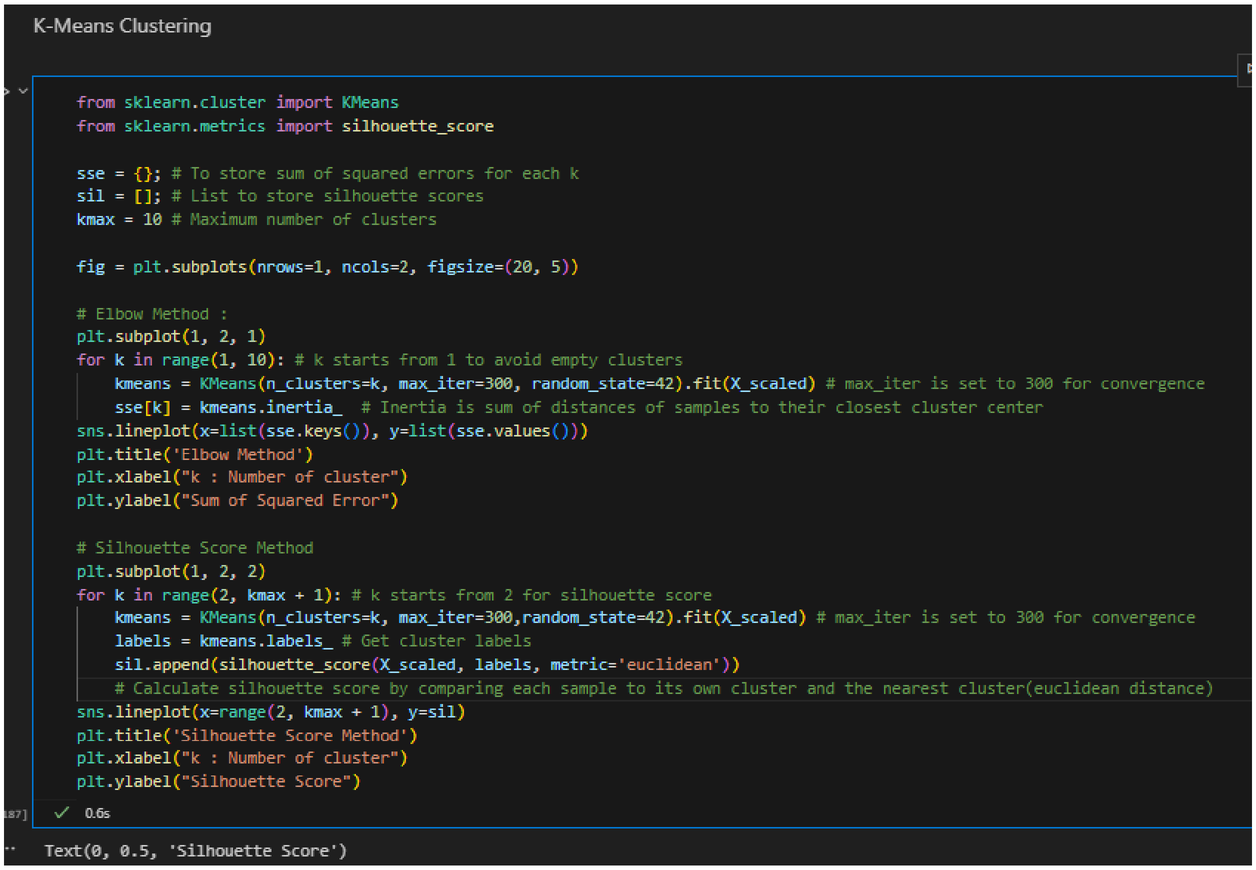
Standardization

```
from sklearn.preprocessing import StandardScaler

# Preprocessing
X = df.drop('Outcome', axis=1)
y = df['Outcome']

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
✓ 0.0s
```

Before conducting the clustering, we need to separating features and target, then standardizing the features. First, X = df.drop('Outcome', axis=1) creates a new DataFrame X containing all columns except 'Outcome'. This means X holds the input features that will be used for modeling. The line y = df['Outcome'] extracts the 'Outcome' column as the target variable. Next, a StandardScaler() object is created. StandardScaler() is a preprocessing tool from scikit-learn that standardizes features by removing the mean and scaling to unit variance. Finally, X_scaled = scaler.fit_transform(X) fits the scaler to the data by computing the mean and standard deviation for each feature and then transforms the data so that each feature has a mean of 0 and a standard deviation of 1. In short, using standardized data (X_scaled) is crucial since K-means is sensitive to feature scaling.

3.1 Identified optimal K


K-Means Clustering

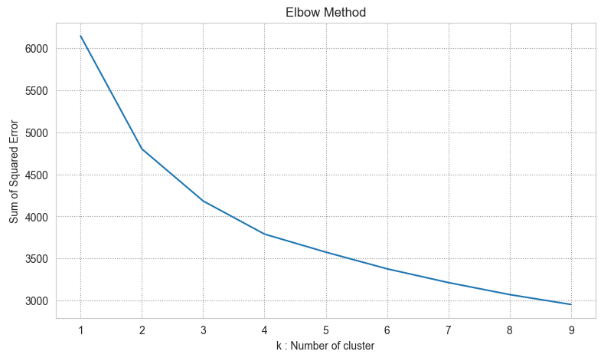
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

sse = {}; # To store sum of squared errors for each k
sil = []; # List to store silhouette scores
kmax = 10 # Maximum number of clusters

fig = plt.subplots(nrows=1, ncols=2, figsize=(20, 5))

# Elbow Method :
plt.subplot(1, 2, 1)
for k in range(1, 10): # k starts from 1 to avoid empty clusters
    kmeans = KMeans(n_clusters=k, max_iter=300, random_state=42).fit(X_scaled) # max_iter is set to 300 for convergence
    sse[k] = kmeans.inertia_  # Inertia is sum of distances of samples to their closest cluster center
sns.lineplot(x=list(sse.keys()), y=list(sse.values()))
plt.title('Elbow Method')
plt.xlabel("k : Number of cluster")
plt.ylabel("Sum of Squared Error")

# Silhouette Score Method
plt.subplot(1, 2, 2)
for k in range(2, kmax + 1): # k starts from 2 for silhouette score
    kmeans = KMeans(n_clusters=k, max_iter=300,random_state=42).fit(X_scaled) # max_iter is set to 300 for convergence
    labels = kmeans.labels_ # Get cluster labels
    sil.append(silhouette_score(X_scaled, labels, metric='euclidean'))
    # Calculate silhouette score by comparing each sample to its own cluster and the nearest cluster(euclidean distance)
sns.lineplot(x=range(2, kmax + 1), y=sil)
plt.title('Silhouette Score Method')
plt.xlabel("k : Number of cluster")
plt.ylabel("Silhouette Score")
```

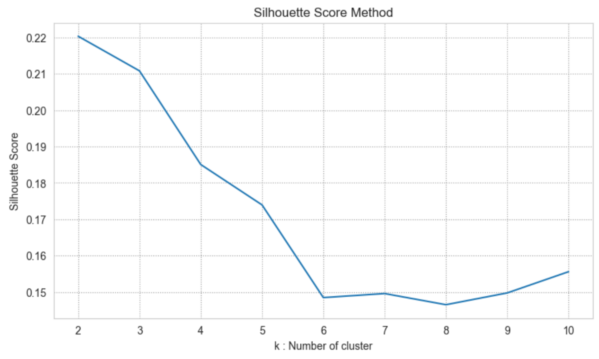
✓ 0.6s

```
Text(0, 0.5, 'Silhouette Score')
```

To determine the optimal number of K, two methods which are Elbow method and Silhouette score method are used. The first method is Elbow method. The for loop evaluates potential cluster numbers (k) from 1 to 9 by training a K-means model for each value of k. Each model is configured with n_clusters=k, a maximum of 300 iterations (*max_iter=300*) to assure convergence, and a constant random seed (*random_state=42*) for reproducibility results. When fitting the model to the standardized data (X_scaled), K-means initializes the centroids and keeps assigning data to the nearest clusters and updating the centroids until the centroids are placed at the optimal position. The performance of the model is evaluated by measuring the sum of squared errors (kmeans. inertia_), which is the sum of squares of distance between each point and its assigned centroid. These values are stored in a dictionary (sse) for later analysis. Notably, random_state=42 ensures consistency across runs. Without it, random initialization could yield varying results. This systematic approach allows for an objective comparison of different cluster counts to determine the optimal k.

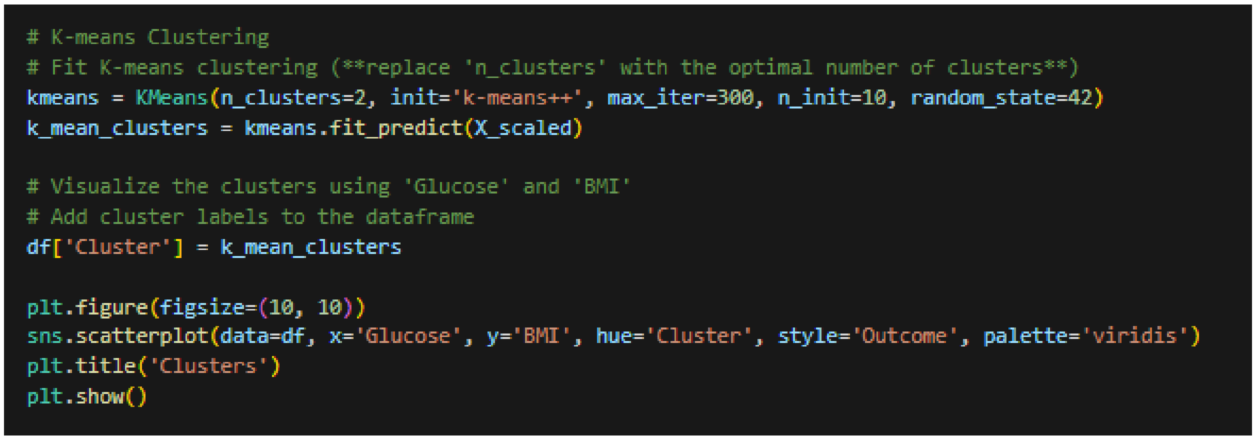
The second method used is the Silhouette score method. The analysis systematically evaluates different cluster counts (k ranging from 2 to 10) by fitting K-means models to standardized data (X_scaled). For each k value, the algorithm creates clusters while maintaining reproducibility through random_state=42. After assigning data points to clusters by access it via "kmeans.labels_", the model's effectiveness is measured using silhouette scores. These scores, calculated using Euclidean distance, showing how well each point fits its assigned cluster compared to neighboring clusters. Scores approaching 1 indicate clear, distinct clustering, while scores near 0 suggest overlapping cluster boundaries, and negative values reveal potential misassignments. The results are stored in the "sil" list for later plotting to help identify the optimal number of clusters.





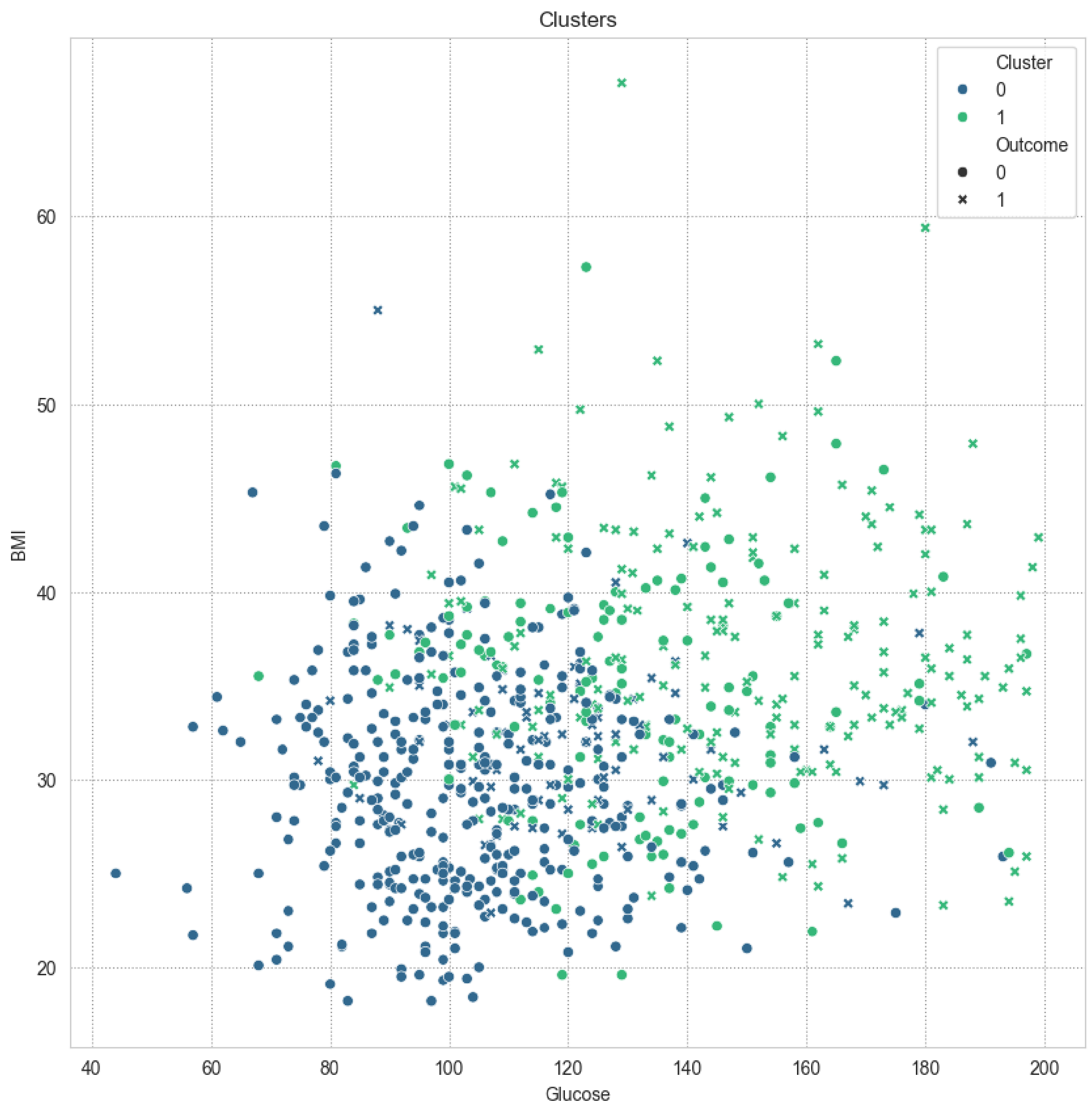
Optimal clusters number is k=2. From the elbow method, the SSE continues decreasing after K=2, but at a much slower rate. So best K appears to be 2 as that's where the "elbow" occurs. Silhouette score of 0.22 at k=2 indicates the best-defined clusters. Scores decline afterward (0.21, 0.185, etc.), confirming k=2 as the peak.

3.2 K-Means Clustering



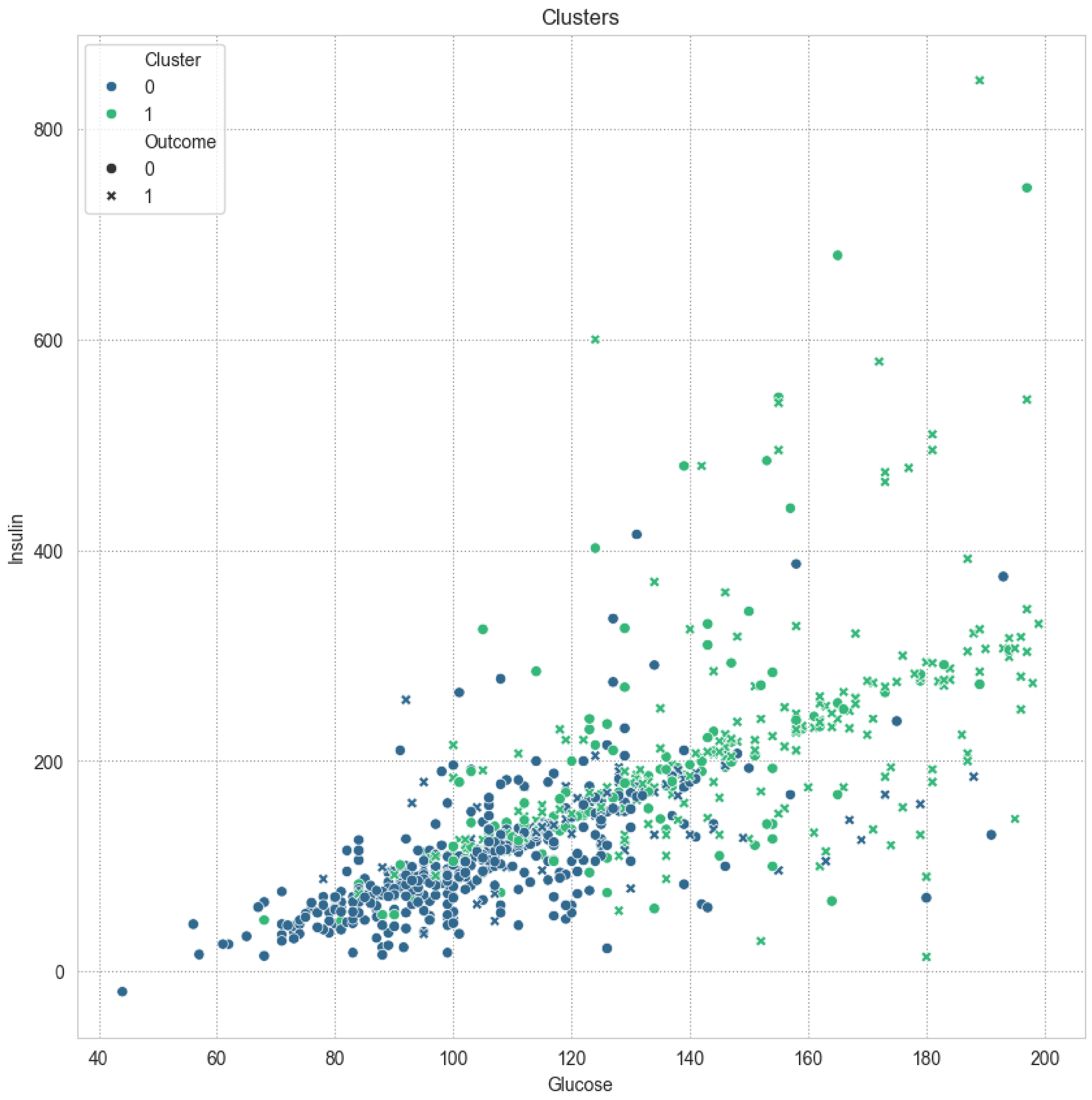
```python
# K-means Clustering
# Fit K-means clustering (**replace 'n_clusters' with the optimal number of clusters**)
kmeans = KMeans(n_clusters=2, init='k-means++', max_iter=300, n_init=10, random_state=42)
k_mean_clusters = kmeans.fit_predict(X_scaled)

# Visualize the clusters using 'Glucose' and 'BMI'
# Add cluster labels to the dataframe
df['Cluster'] = k_mean_clusters

plt.figure(figsize=(10, 10))
sns.scatterplot(data=df, x='Glucose', y='BMI', hue='Cluster', style='Outcome', palette='viridis')
plt.title('Clusters')
plt.show()
```

The code performs K-means clustering on standardized data using three clusters (n_clusters=2), as determined by prior elbow method and Silhouette score method. The algorithm employs several key parameters to ensure optimal performance. The k-means++ initialization (init='k-means++') is used for better centroid placement, allows 300 iterations (max_iter=300) for convergence, runs 10 independent trials (n_init=10) to find the most stable solution, and maintains reproducibility through a fixed random seed (random_state=42). The fit_predict() method efficiently combines two operations, training the model on the standardized data (X_scaled) and immediately returning the cluster assignments (0, 1) for each data point. These labels, stored in the 'k_mean_clusters' variable, represent the final grouping of all observations into three distinct clusters based on their feature similarities.



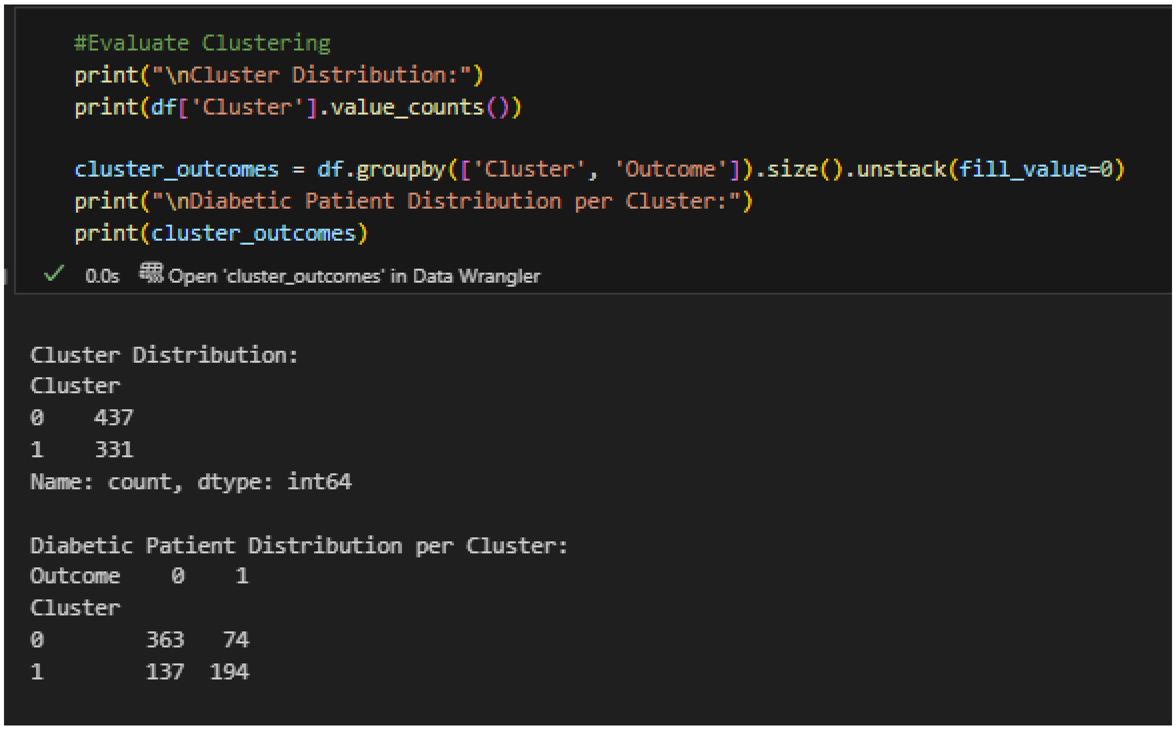
Cluster 0 the dark blue, mostly glucose level is less than 120 and thier BMI value is less than 35. Means cluster 0 is having less diabetes patient with normal glucose lvl and BMI rate. Cluster 1 the lime color,mostly their glucose level is exceed 120 and have many 'x' (outcome 1) when glucose lvl> 140 and BMI >30.

```
plt.figure(figsize=(10, 10))
sns.scatterplot(data=df, x='Glucose', y='Insulin', hue='Cluster', style='Outcome', palette='viridis')
plt.title('Clusters')
plt.show()
```



The plot reveals a positive glucose-insulin relationship in non-diabetic individuals (blue), so when glucose level increase the insulin hormone in the body also increase. The diabetic people in lime color, is either insulin resistance (high insulin level >180, while glucose level is high>120) or beta-cell dysfunction (low insulin level<180, while glucose level is high)

3.3 Interpretation and Evaluation

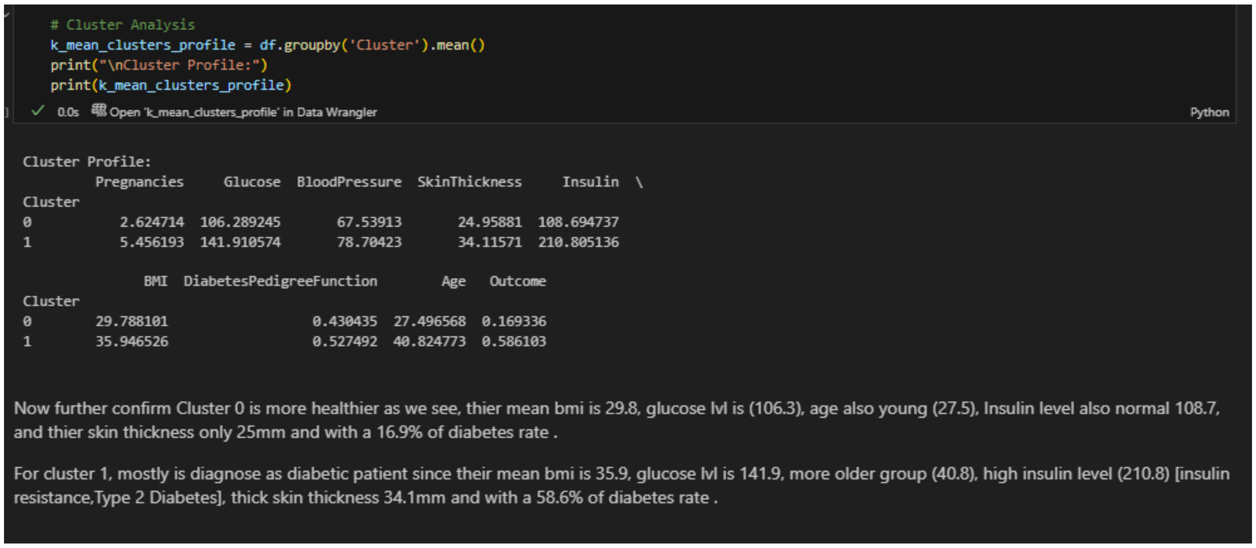


```
#Evaluate Clustering
print("\nCluster Distribution:")
print(df['Cluster'].value_counts())

cluster_outcomes = df.groupby(['Cluster', 'Outcome']).size().unstack(fill_value=0)
print("\nDiabetic Patient Distribution per Cluster:")
print(cluster_outcomes)
```

✓ 0.0s Open 'cluster_outcomes' in Data Wrangler

```
Cluster Distribution:
Cluster
0    437
1    331
Name: count, dtype: int64

Diabetic Patient Distribution per Cluster:
Outcome    0    1
Cluster
0        363   74
1        137  194
```

The graph and the analysis reveals how the data points are distributed across two clusters and their relationship with diabetes outcomes. Cluster 0 contains the most observations (437 cases), followed by Cluster 1 (331 cases). When examining diabetes outcomes within each cluster, we see distinct patterns. Cluster 1 shows a majority of diabetic cases (194 cases vs 137 non diabetic). Cluster 0 predominantly consists of non-diabetic cases (363 vs 74 diabetic. These patterns suggest that Cluster 0 may represent a lower-risk group and Cluster 1 appears to be a higher-risk group for diabetes.



```
# Cluster Analysis
k_mean_clusters_profile = df.groupby('Cluster').mean()
print("\nCluster Profile:")
print(k_mean_clusters_profile)
```

```
Cluster Profile:
         Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
Cluster
0           2.624714  106.289245       67.53913       24.95881  108.694737
1           5.456193  141.910574       78.70423       34.11571  210.805136

               BMI  DiabetesPedigreeFunction        Age   Outcome
Cluster
0        29.788101                  0.430435  27.496568  0.169336
1        35.946526                  0.527492  40.824773  0.586103
```

Now further confirm Cluster 0 is more healthier as we see, thier mean bmi is 29.8, glucose lvl is (106.3), age also young (27.5), Insulin level also normal 108.7, and thier skin thickness only 25mm and with a 16.9% of diabetes rate .

For cluster 1, mostly is diagnose as diabetic patient since their mean bmi is 35.9, glucose lvl is 141.9, more older group (40.8), high insulin level (210.8) [insulin resistance,Type 2 Diabetes], thick skin thickness 34.1mm and with a 58.6% of diabetes rate .

The result reveals two cluster patient groups based on clinical characteristics. Cluster 1 is a high-risk middle-aged group because it shows high values across multiple risk factors like highest pregnancy history (5.46), high glucose (141.9), BMI (35.9), and older age (40.8), with a 58.6% diabetes rate.

Cluster 0 is a low-risk young group. They display the most favorable profile where they have lowest glucose (106.3), blood pressure (67.5), BMI (29.7), and youngest age (27.5), corresponding to just 16.9% diabetes rate.

4.0 Conclusion

This study successfully applied K-means clustering to identify distinct patient subgroups within the diabetes dataset, demonstrating the value of unsupervised learning in medical risk stratification. The optimal cluster count (k=2) is determined through the Elbow Method and Silhouette Score analysis. It revealed meaningful patient segments.

Cluster 1 is a high risk middle aged group with age driven metabolic dysfunction which is 40.8 years old and has 58.6% diabetes rates. Cluster 0 is a low-risk young group that exhibiting optimal health metrics with 16.9% diabetes rates.

These findings support known clinical patterns, showing that the model can identify biologically meaningful trends. The analysis effectively standardized features and used k-means++ initialization to ensure consistent results. However, there are some limitations where it cannot say for sure what causes these diabetes issues, and it could be even better if it included things like genetics or lifestyle habits. The results are valuable for medical applications because it allows doctors to customize treatments. For example, the low-risk patients may benefit from preventive education, whereas the high-risk patients may need more aggressive metabolic control. That could be used to predict the risk of complications in each group, and to allow diabetes care to be more personalized to each person.