Ahsanullah University of Science & Technology

Department of Computer Science & Engineering

Semester: Fall 2023

CSE 4262- Data Analytic Lab

Assignment On: Machine Learning with Spark

Submitted To

Prof. Dr. Md. Shamim Akhter
Mr. Md. Zahid Hossain
CSE AUST

Submitted By

SHUVASHIS SARKER ID:20200104116

Date: 16 May 2024

CSE 4262 Data Analycs Lab

Lab 3: Machine Learning with Spark

Outcomes: Aer this lab students will be able to:

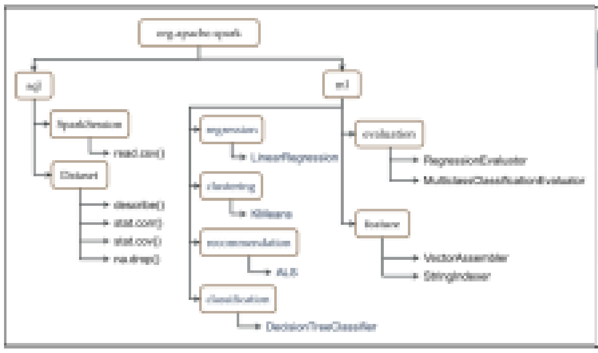
• Apply PySpark MLib libraries to the development of machine learning applica ons. • Understand ML Pipeline concepts and use them to build machine learning algorithms and applica ons.

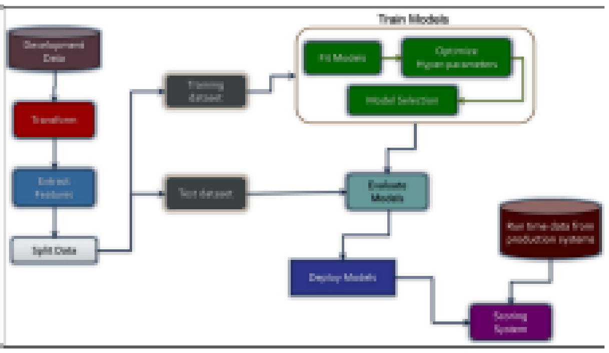
Machine Learning Pipeline – a quick introducon

One of the major a$rac ons of Spark is its ability to scale computa ons massively, and this is exactly what you need for machine learning algorithms. However, the caveat is that all machine learning algorithms cannot be effec vely parallelized. Each algorithm has its challenges for paralleliza on, whether task or data parallelism. Spark is becoming the de-facto pla,orm for building machine learning algorithms and applica ons. This lab will first cover machine learning interfaces and organiza on, including the new ml pipeline, which has become mainstream in 2.0.0. Then, we will delve into the following machine-learning algorithms:

- Basic sta s cs
- Linear regression
- Classifica on
- Clustering
- Recommenda on

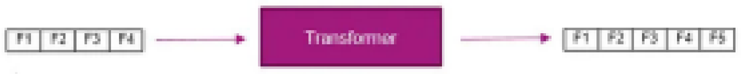
ML pipelines were developed to address the fact that machine learning is not just a bunch of algorithms, such as classifica on and regression, but a pipeline of ac ons performed over a Dataset. Let's take a quick look at the tasks involved in a typical machine-learning process. The following figures show the top-level ac vi es of Spark ML libraries(mlib) and their API organiza on.



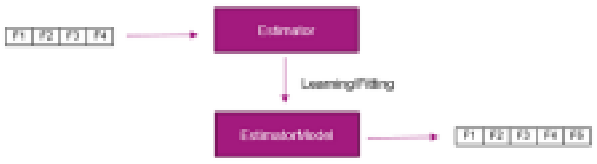


The newer API has introduced a very powerful concept of pipelines where different stages of machine learning work can be grouped as a single en ty and can be considered as one seamless machine learning workflow. A pipeline contains a set of stages where each stage is either a Transformer or an Es mator. All of these stages run in sequence and the input dataframe gets transformed while it passes through each stage.

Transformer: A Transformer is a high-level abstrac on of feature transformer and learned models (models returned by Es mators). This component either adds, deletes, or updates exis ng features in the dataframe. **Each transformer has a transform () method which gets called when the pipeline is executed.** For example, **VectorAssembler is a transformer** as it takes the input dataframe and returns the transformed dataframe with a new column which is a vector representa on of all the features.



Esmator: An Es mator is a high-level abstrac on of a learning algorithm that returns a Model (a Transformer) and the returned model transforms the dataframe by the parameters which are learned during the fi<ng/learning phase. **Each esmator has a fit() method** which returns a Model which in turn has a transform() method. For example, **LogiscRegression is an esmator** that returns a Logis cRegresionModel (a Transformer) aer learning parameters about the data.

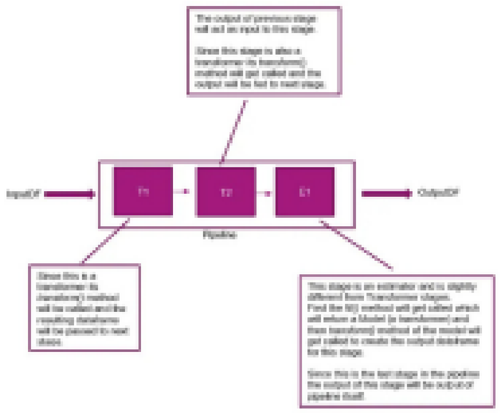


Write down the difference between a Transformer and an Esmator

Aspect	**Transformer**	**Estimator**
Purpose	Modifies or transforms a DataFrame by adjusting its features.	Learns from data to generate a model, which can transform DataFrames.
Methodology	Uses a `transform()` method to apply transformations directly to a DataFrame.	Uses a `fit()` method to analyze data and produce a model (a Transformer).
Output	Outputs the transformed DataFrame directly.	Outputs a model that can transform DataFrames based on learned parameters.
Example	`VectorAssembler` combines multiple columns into a single vector column.	`LogisticRegression` fits to data and produces a `LogisticRegressionModel`

Spark provides a class that forms by combining different PipelineStages (Es mator and Transformers) which run in sequen al order. Pipeline class has a fit () method which kicks off the en re workflow. This execu on returns a PipelineModel which has the same number of stages as the pipeline except that all

the Es mator stages are replaced by their respec ve Model (a Transformer) which were fi$ed during the execu on. During execu on, each stage is called sequen ally, and based on the type of PipelineStage (whether it's a Transformer or an Es mator) their respec ve fit () or transform() methods



are called.

Task1. Apply some basic transformers and esmators to formulate an ML pipeline a) Download a file Social_Network_Ads.csv from the following link: h$ps://github.com/kaysush/spark-ml-pipeline-demo/blob/master/Social_Network_Ads.csv and upload it to your workplace at Google-collab.

b) Read the file using the following command:

```
df =
spark.read.csv('/content/Social_Network_Ads.csv',header=True,escape="\"")
```

c) Check the datatypes of the column headers.

d) Change the type of the a$ributes including Age, Es matedSalary and purchased from string to double or integer using the following commands

```
from pyspark.sql.types import IntegerType,BooleanType,DateType,
DoubleType  df=df.withColumn("Age",df.Age.cast(IntegerType()))
…
```

e) Split the data using `randomSplit()` method. Partition the data into Training for 80% and 20% for the Testing. Present the amount of testing and training data into the blank box.

Train: 325 Test: 75

f) Since our dataset has a categorical column, Gender we'll have to perform OneHotEncoding on it before passing it to Logis cRegression because machine learning algorithms do not support string values. Use the following commands to perform OneHotEncoding on the Gender a$ribute. These are the first and second Transformers for our pipeline.

```
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import OneHotEncoder
genderIndexer=StringIndexer(inputCol="Gender",
outputCol="GenderIndexed")  genderOneHotEncoder=OneHotEncoder(inputCols=[gend
erIndexer.getOutputCol()],outp utCols=["GenderOHE"])
```

g) Vector Assembling the features is a necessary step before model training starts. The machine learning algorithms do not take a variable number of columns as input, rather they take a vector of features. This is the third Transformer for our pipeline.

```
from pyspark.ml.feature import VectorAssembler
vectorAssembler=VectorAssembler(inputCols=features, outputCol="features")
```

h) Though feature scaling is an op onal step it certainly helps in decreasing the convergence me. Hence, we have included that in our pipeline. This is the fourth Transformer for our pipeline. `from pyspark.ml.feature import StandardScalerscaler =`

```
StandardScaler(inputCol="features", outputCol="scaledFeatures",
withStd=True,  withMean=False)
```

i) Now we are going to apply our Es mator -Logis c Regression to learn from the data and later on predict the user buying behavior.

```
from pyspark.ml.classification import LogisticRegression
logistic_regression =
LogisticRegression(featuresCol="scaledFeatures",  labelCol=dependentV
ar)
```

j) Now that our components are ready to let’s put them in the pipeline and start the training of our models.

```
from pyspark.ml import Pipeline
pipeline=Pipeline(stages=[genderIndexer,genderOneHotEncoder,vectorAssembler,sc
a ler,logistic_regression])
model=pipeline.fit(train)
```

k) The fit () method of the pipeline returns a PipelineModel object which we’ll use to predict the behavior of unknown customers

```
results=model.transform(test)
```

l) To gauge the accuracy of our model we’ll use Spark’s built in BinaryClassifica onEvaluator class. `from pyspark.ml.evaluation import BinaryClassificationEvaluator`

```
evaluator=BinaryClassificationEvaluator(labelCol=dependentVar)
accuracy = evaluator.evaluate(results)
print (f"Accuracy of the Model: {accuracy}")
```

Present the Model accuracy in the blank box:

Accuracy: `0.9193313953488372`

m) Now explain your understanding of ML Pipeline in the following box

Pipeline Execution:

- When run a pipeline, it executes each stage in order.
- If the stage is a transformer, it transforms the data.
- If the stage is an estimator, it fits a model to the data and transforms the data using the learned model.
- The final output is a `PipelineModel` that includes all the fitted models (transformers) and can be used to transform new data in the same way.

Workflow:

1. Data Preparation: Start with raw data in a DataFrame.

Task2. Understand the implementaon of Mul layerPerceptronClassifier and LinearSVC for classifica on

a) Download a file Social_Network_Ads.csv from the following link: h$ps://github.com/davutemrah/spark_repo/blob/master/data/diabetes.csv and upload it to your workplace at Google-collab.

b) Split the data into 60% for training and 40% for tes ng.

c) Use Mul layerPerceptronClassifier() from pyspark.ml.classifica on package and LinearSVC() from pyspark.ml.classifica on package.

d) Evaluate their performance using Mul classClassifica onEvaluator from pyspark.ml.evalua on package.

e) A$ached the pyspark program below. [Class Task File]

Task3:

Recommenda on systems are one of the most visible and popular machine learning applica ons on the Web, from Amazon to LinkedIn to Walmart. The algorithms behind recommenda on systems are very interes ng. Recommenda on algorithms fall into roughly five general mechanisms: knowledge-based, demographic-based, content-based, collabora ve filtering (item-based or user-based), and latent factor
based. Of these, collabora ve filtering is the most widely used and unfortunately very computa onally intensive. Spark implements a scalable varia on, the Alterna ng Least Square (ALS) algorithm authored by Yehuda Koren, available at h$p://dl.acm.org/cita on.cfm?id=1608614. It is a user-based collabora ve filtering mechanism that uses the latent factors method of learning, which can scale to a large Dataset.

a) Download a file ra ng.csv from the Google classroom and upload it to your workplace at Google collab.

b) Change the type of the a$ributes including `userId`, `movieId`, and `rating` from string to double or integer.

c) Split the data using `randomSplit()` method. Partition the data into Training for 70% and 30% for the Testing.

d) Use the following code to ini alize the `ALS()` model `from pyspark.ml.recommendation import ALS` package.

```
from pyspark.ml.recommendation import ALS

# Create an ALS instance
algALS = ALS()

# Set parameters for ALS model
algALS.setItemCol("movieId") # Set the name of the column for items
(products)  algALS.setUserCol("userId")
```

```
algALS.setRank(12)
algALS.setRegParam(0.1) # Regularization parameter (equivalent to lambda
in  MLlib)
algALS.setMaxIter(20)
```

e) Call the `fit ()` method of the algALS object with the train data. Thereaer, call the `transform()` method with test data to make the predic on.

f) Filter out rows from predic on with null values before the evalua on.

g) Evaluate their performance using `RegressionEvaluator() from pyspark.ml.evaluation  import RegressionEvaluator` package.

h) A$ached the pyspark program below.   [Class Task File]

Assignment 3

Choose a suitable dataset containing various a$ributes. The dataset should have at least 10,000 samples. Your task is to perform data clustering to uncover inherent structures or pa$erns within the dataset. Specifically, you are required to:

1. Preprocess the dataset: Handle missing values, scale features, and any other necessary preprocessing steps.
2. Apply at least two different clustering algorithms: You can choose from a variety of clustering algorithms such as K-means, hierarchical clustering, DBSCAN, Gaussian Mixture Models (GMM), etc.
3. Evaluate the performance of each clustering algorithm: Use appropriate metrics to evaluate the quality of the clusters generated by each algorithm. Some common evalua on metrics include silhoue$e score, Davies-Bouldin index, and Calinski-Harabasz index.
4. Interpret the results: Analyze the clusters obtained from each algorithm and provide insights into the characteris cs of each cluster. What do these clusters represent? Are there any no ceable pa$erns or trends?
5. Compare the performance of the clustering algorithms: Discuss the strengths and weaknesses of each algorithm based on their performance on the given dataset.

Question/Answer

1

df = spark.read.csv('/content/drive/MyDrive/Data Analytic Lab/Stroke-data.csv', header=True, inferSchema=True)

df.printSchema()

from pyspark.sql.types import FloatType, IntegerType

from pyspark.sql.functions import col

# Handle missing values

df = df.na.fill({

   'bmi': df.select('bmi').dropna().agg({'bmi': 'mean'}).first()[0]

})

df = df.withColumn("age", col("age").cast(IntegerType()))

df = df.withColumn("hypertension", col("hypertension").cast(IntegerType()))

df = df.withColumn("heart_disease", col("heart_disease").cast(IntegerType()))

df = df.withColumn("avg_glucose_level", col("avg_glucose_level").cast(IntegerType()))

df = df.withColumn("bmi", col("bmi").cast(IntegerType()))

df = df.withColumn("stroke", col("stroke").cast(IntegerType()))

string_columns = ['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status']

indexers = [StringIndexer(inputCol=col, outputCol=col+"_index") for col in string_columns]

feature_columns = ['age', 'hypertension', 'heart_disease', 'avg_glucose_level', 'bmi'] + [col+"_index" for col in string_columns]

assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")

# Scale features

scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures", withStd=True, withMean=False)

# Create a preprocessing pipeline

pipeline = Pipeline(stages=indexers + [assembler, scaler])

df_preprocessed = pipeline.fit(df).transform(df)

#2

from pyspark.ml.clustering import KMeans

# Apply K-Means Clustering

kmeans = KMeans(k=3, seed=1, featuresCol="scaledFeatures")

kmeans_model = kmeans.fit(df_preprocessed)

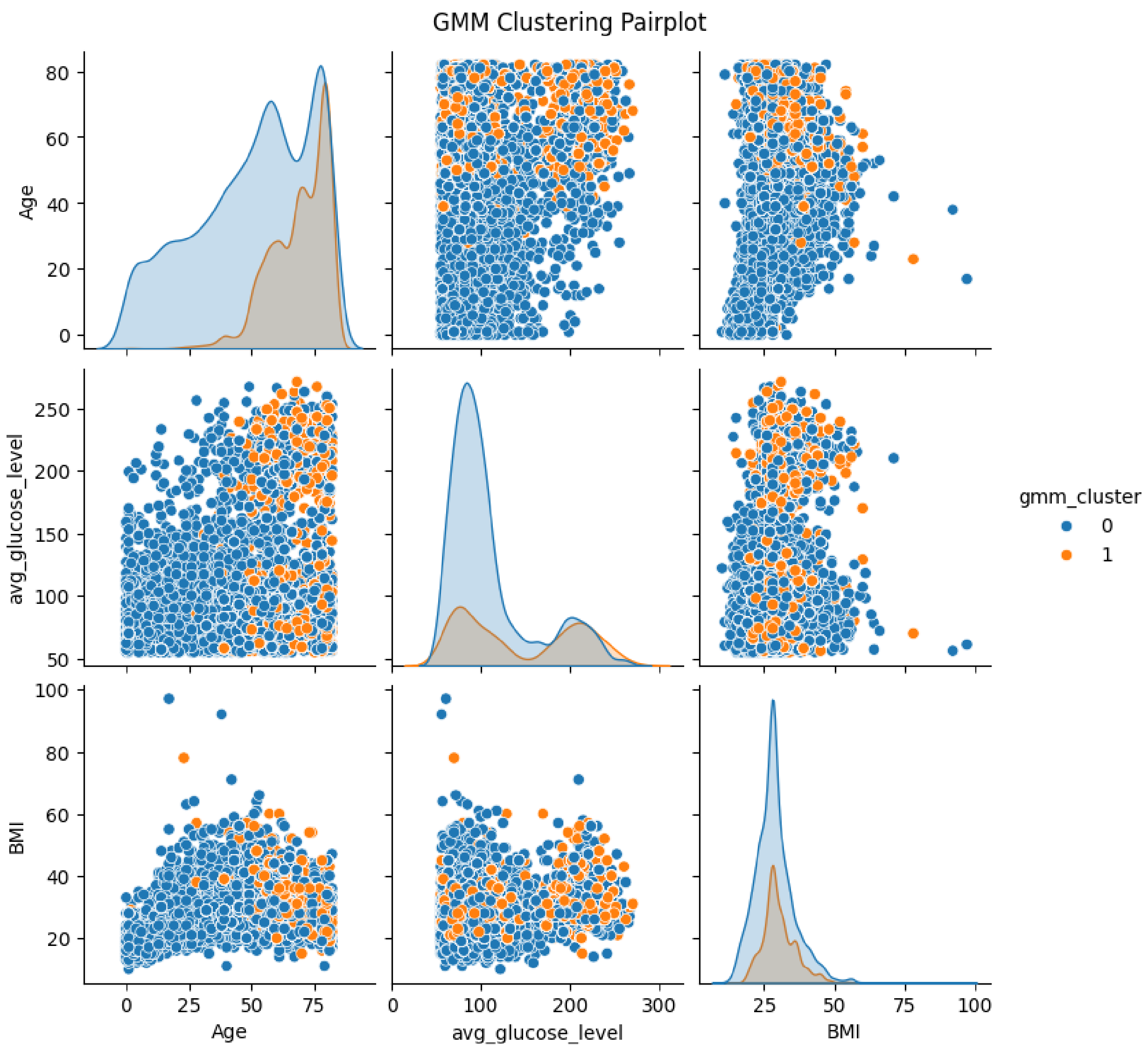
kmeans_predictions = kmeans_model.transform(df_preprocessed)

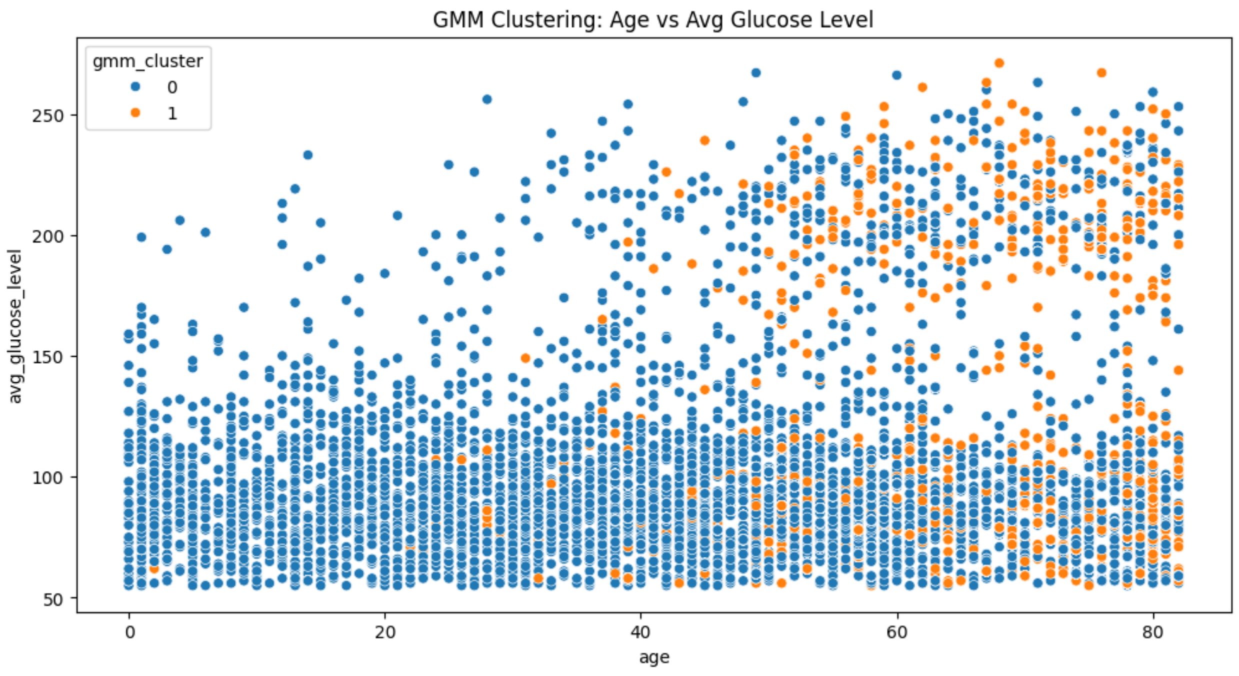
from pyspark.ml.clustering import GaussianMixture

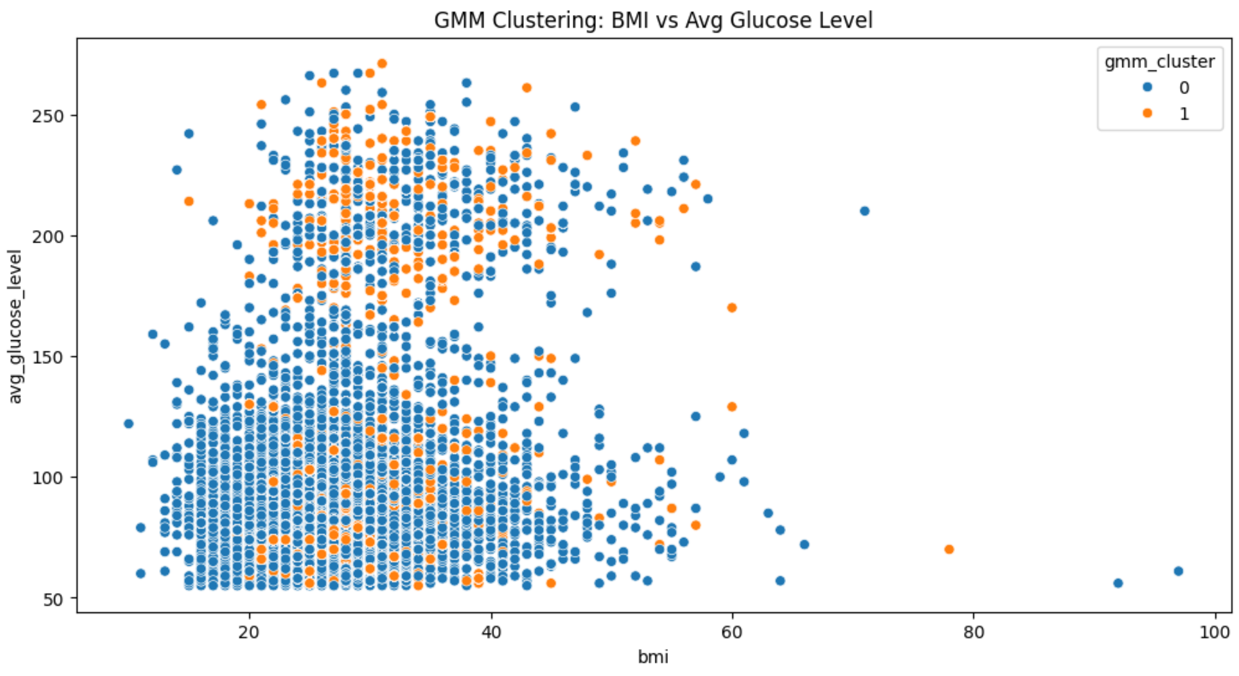
# Apply Gaussian Mixture Models

```
gmm = GaussianMixture(k=3, seed=1, featuresCol="scaledFeatures")

gmm_model = gmm.fit(df_preprocessed)

gmm_predictions = gmm_model.transform(df_preprocessed)

from pyspark.ml.evaluation import ClusteringEvaluator


# 3

# evaluator for Silhouette Score

evaluator = ClusteringEvaluator(featuresCol="scaledFeatures", metricName="silhouette",
distanceMeasure="squaredEuclidean")


# Evaluate K-Means

kmeans_silhouette = evaluator.evaluate(kmeans_predictions)

print(f"K-Means: Silhouette Score = {kmeans_silhouette}")


# Evaluate GMM

gmm_silhouette = evaluator.evaluate(gmm_predictions)

print(f"GMM: Silhouette Score = {gmm_silhouette}")


import numpy as np

from sklearn.metrics import davies_bouldin_score, calinski_harabasz_score


def calculate_custom_metrics(predictions, feature_col, prediction_col):


    pdf = predictions.select(feature_col, prediction_col).toPandas()

    features = np.array(pdf[feature_col].tolist())

    labels = pdf[prediction_col].to_numpy()


    # Davies-Bouldin Index

    db_index = davies_bouldin_score(features, labels)
    # Calinski-Harabasz Index
    ch_index = calinski_harabasz_score(features, labels)

    return db_index, ch_index

# Evaluate K-Means
kmeans_db_index, kmeans_ch_index = calculate_custom_metrics(kmeans_predictions, 
"scaledFeatures", "prediction")
print(f"K-Means: Davies-Bouldin Index = {kmeans_db_index}, Calinski-Harabasz Index = 
{kmeans_ch_index}")

# Evaluate GMM
gmm_db_index, gmm_ch_index = calculate_custom_metrics(gmm_predictions, "scaledFeatures", 
"prediction")
print(f"GMM: Davies-Bouldin Index = {gmm_db_index}, Calinski-Harabasz Index = {gmm_ch_index}")

# evaluation results
print(f"K-Means: Silhouette Score = {kmeans_silhouette}, Davies-Bouldin Index = {kmeans_db_index}, 
Calinski-Harabasz Index = {kmeans_ch_index}")

print(f"GMM: Silhouette Score = {gmm_silhouette}, Davies-Bouldin Index = {gmm_db_index}, Calinski-
Harabasz Index = {gmm_ch_index}")

kmeans_predictions = kmeans_predictions.withColumnRenamed("prediction", "kmeans_cluster")

# Convert to Pandas
kmeans_df = kmeans_predictions.select(feature_columns + ["kmeans_cluster"]).toPandas()
kmeans_summary = kmeans_df.groupby("kmeans_cluster").mean()
print("K-Means Cluster Characteristics:")
print(kmeans_summary)

# Add cluster labels to the DataFrame
gmm_predictions = gmm_predictions.withColumnRenamed("prediction", "gmm_cluster")
# Convert to Pandas for GMM
gmm_df = gmm_predictions.select(feature_columns + ["gmm_cluster"]).toPandas()
gmm_summary = gmm_df.groupby("gmm_cluster").mean()
print("GMM Cluster Characteristics:")
print(gmm_summary)

# Convert Spark DataFrames to Pandas DataFrames
kmeans_df = kmeans_predictions.select(feature_columns + ["kmeans_cluster"]).toPandas()
gmm_df = gmm_predictions.select(feature_columns + ["gmm_cluster"]).toPandas()

kmeans_df['Algorithm'] = 'K-Means'
gmm_df['Algorithm'] = 'GMM'
#4 (visually ploting)
import seaborn as sns
import matplotlib.pyplot as plt

feature_columns = ['Age', 'avg_glucose_level', 'BMI']
kmeanss_df = kmeans_predictions.select(feature_columns + ["kmeans_cluster"]).toPandas()
sns.pairplot(kmeanss_df, hue='kmeans_cluster', diag_kind='kde', palette='tab10')
plt.suptitle('K-Means Clustering Pairplot', y=1.02)
plt.show()

import seaborn as sns
import matplotlib.pyplot as plt
feature_columns = ['Age', 'avg_glucose_level', 'BMI']
```

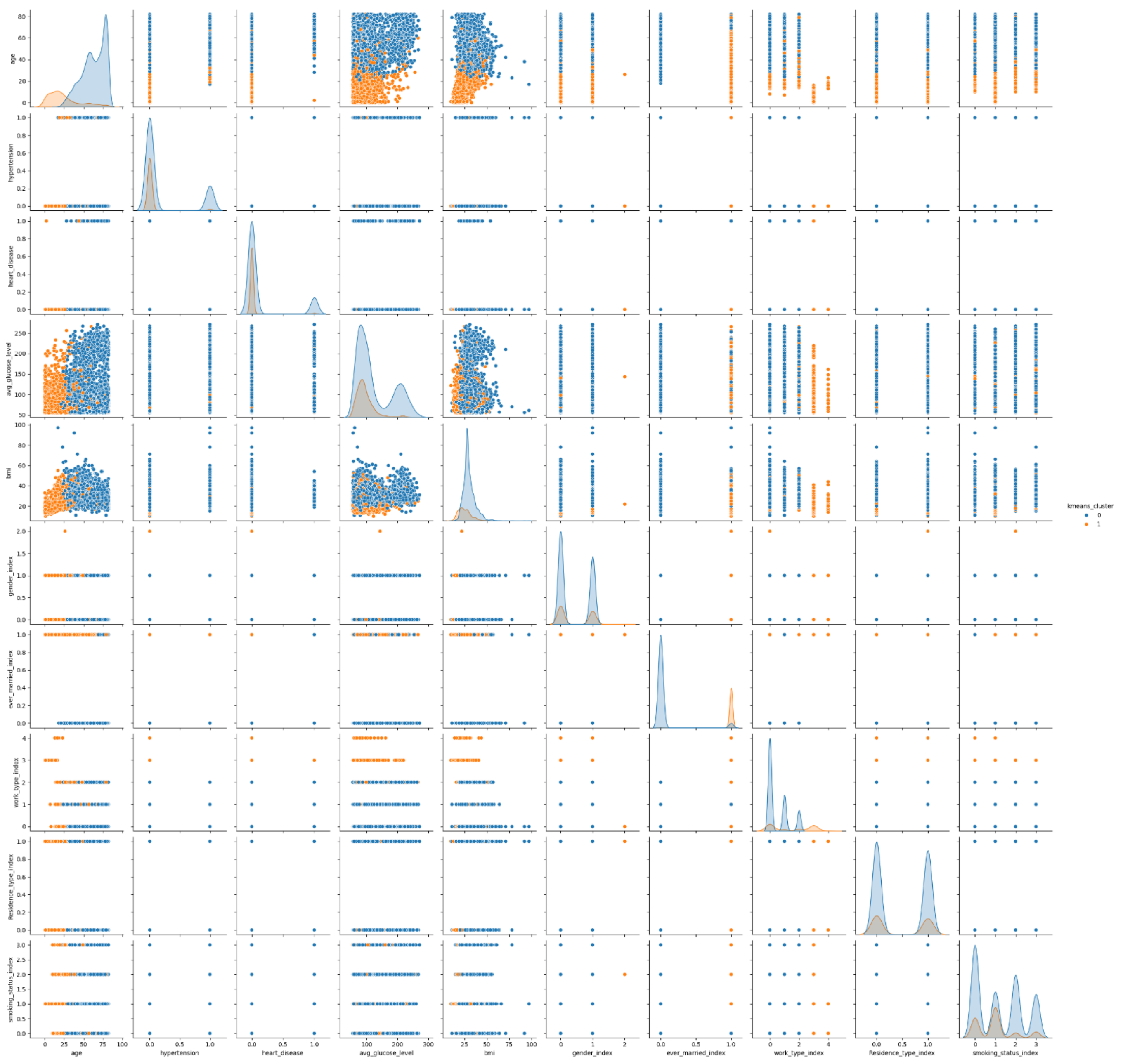
```
gmms_df = gmm_predictions.select(feature_columns + ["gmm_cluster"]).toPandas()

sns.pairplot(gmms_df, hue='gmm_cluster', diag_kind='kde', palette='tab10')
plt.suptitle('GMM Clustering Pairplot', y=1.02)
plt.show()

plt.figure(figsize=(12, 6))
sns.scatterplot(data=kmeans_df, x='age', y='avg_glucose_level', hue='kmeans_cluster', palette='tab10')
plt.title('K-Means Clustering: Age vs Avg Glucose Level')
plt.show()

# Scatterplot for GMM
plt.figure(figsize=(12, 6))
sns.scatterplot(data=gmm_df, x='age', y='avg_glucose_level', hue='gmm_cluster', palette='tab10')
plt.title('GMM Clustering: Age vs Avg Glucose Level')
plt.show()

# Scatterplot for K-Means (BMI vs Avg Glucose Level)
plt.figure(figsize=(12, 6))
sns.scatterplot(data=kmeans_df, x='bmi', y='avg_glucose_level', hue='kmeans_cluster', palette='tab10')
plt.title('K-Means Clustering: BMI vs Avg Glucose Level')
plt.show()

# Scatterplot for GMM (BMI vs Avg Glucose Level)
plt.figure(figsize=(12, 6))
sns.scatterplot(data=gmm_df, x='bmi', y='avg_glucose_level', hue='gmm_cluster', palette='tab10')
plt.title('GMM Clustering: BMI vs Avg Glucose Level')
plt.show()
```

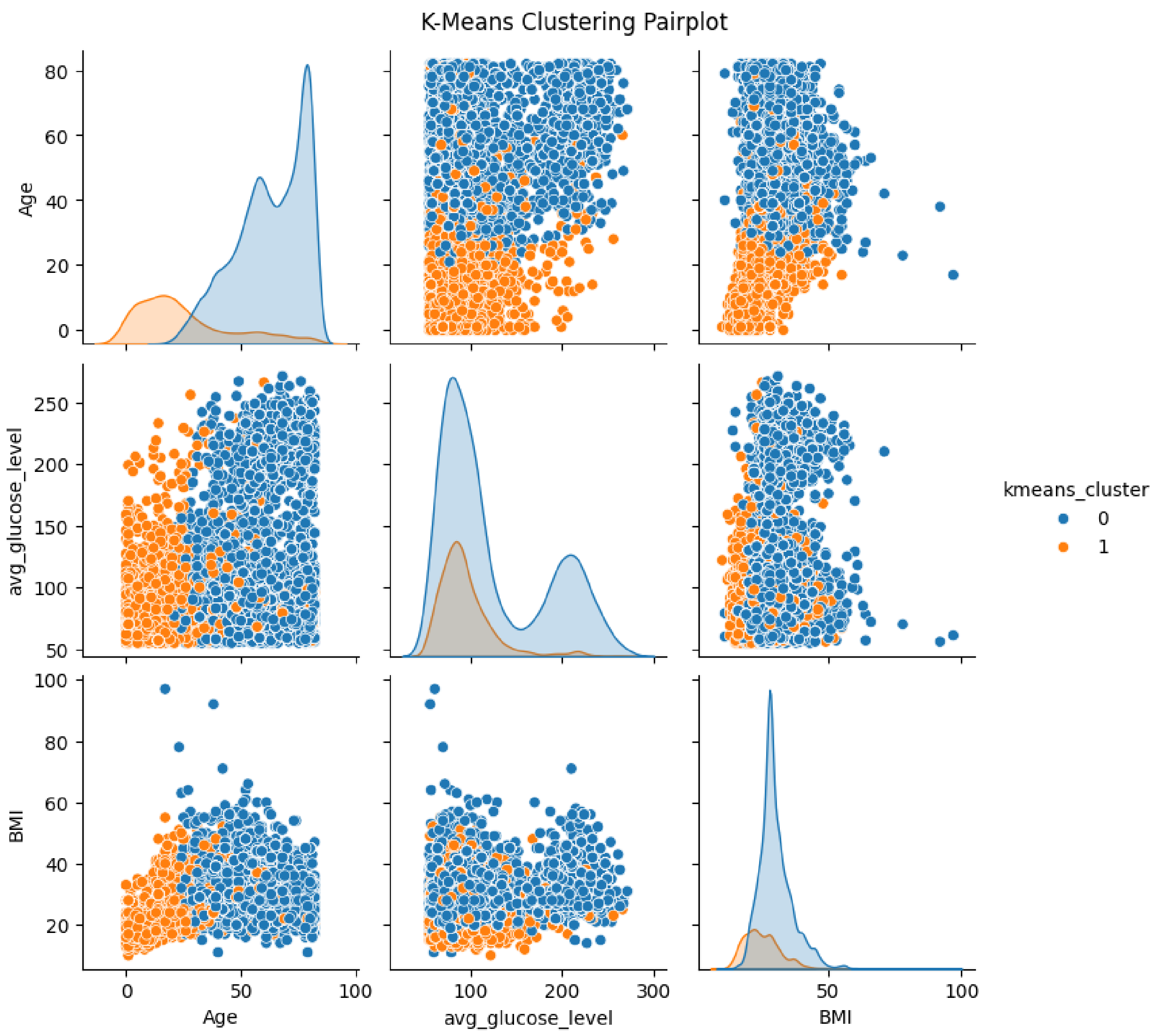

GMM Clustering Pairplot
Age
avg_glucose_level
BMI
gmm_cluster
0
1
0
25
50
75
100
200
300

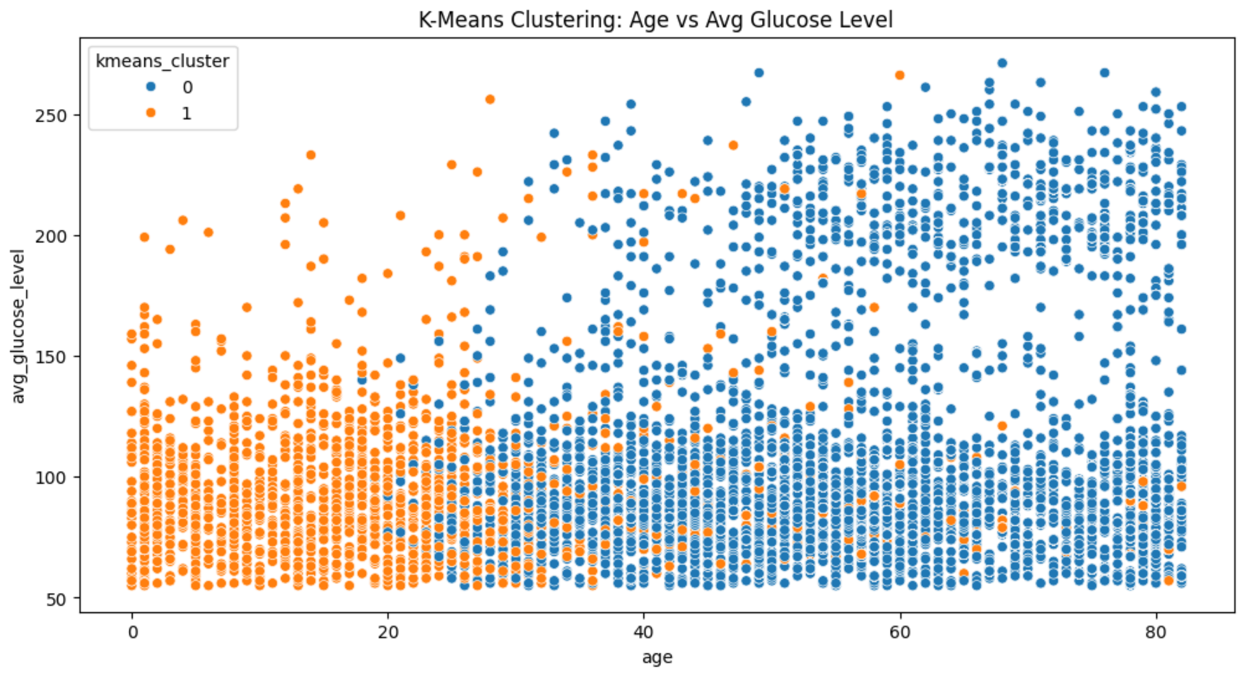

GMM Clustering: Age vs Avg Glucose Level
gmm_cluster
0
1
avg_glucose_level
age
50
100
150
200
250
0
20
40
60
80

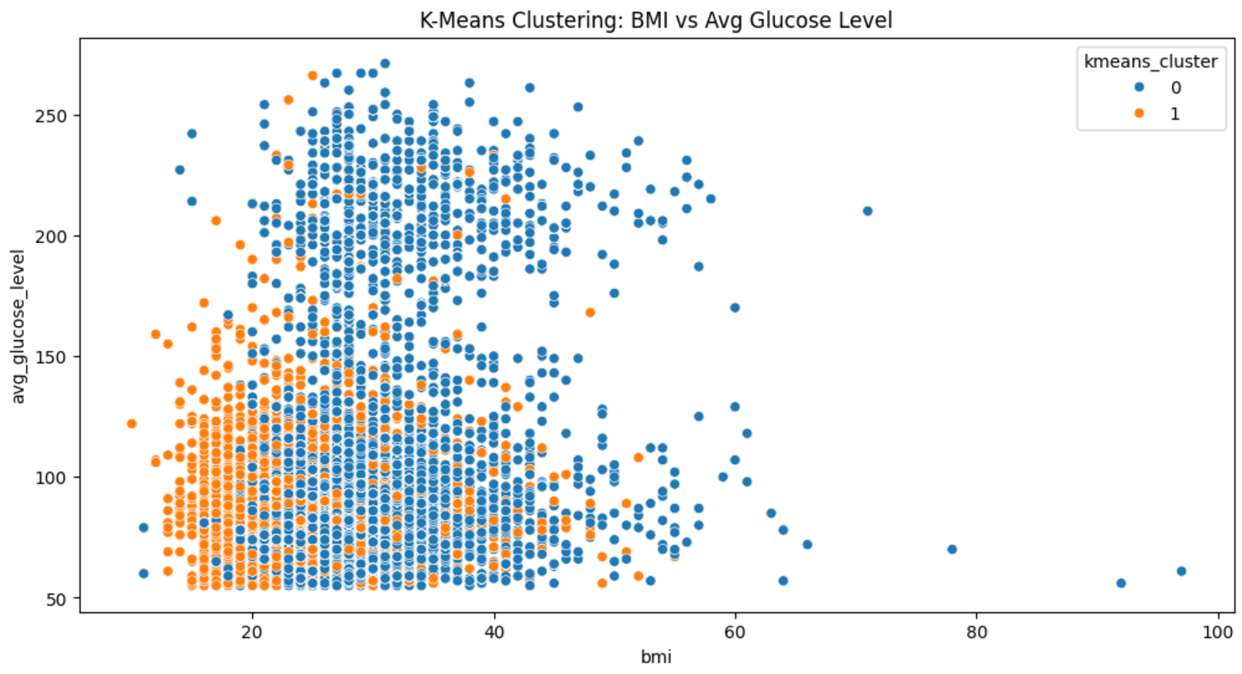

GMM Clustering: BMI vs Avg Glucose Level
gmm_cluster
0
1
avg_glucose_level
bmi
50
100
150
200
250
20
40
60
80
100

K-Means Clustering Pairplot




K-Means Clustering Pairplot
Age
avg_glucose_level
BMI
kmeans_cluster
0
1
0
20
40
60
80
50
100
150
200
250
300


K-Means Clustering: Age vs Avg Glucose Level
kmeans_cluster
0
1
avg_glucose_level
age
50
100
150
200
250
0
20
40
60
80


K-Means Clustering: BMI vs Avg Glucose Level
kmeans_cluster
0
1
avg_glucose_level
bmi
50
100
150
200
250
20
40
60
80
100

Answer:

These plots compare K-Means and Gaussian Mixture Model (GMM) clustering with three clusters (k=3), **visualized for two features: Age and avg_glucose_level**. Both algorithms segment the data differently, which is evident from the distribution and positioning of the clusters:

K-Means Clustering Characteristics

- Cluster 0 (Blue): Primarily younger individuals with lower avg_glucose_level.

- Cluster 1 (Orange): Middle-aged individuals with moderate avg_glucose_level.

- Cluster 2 (Green): Older individuals with higher avg_glucose_level.

Observations:

- Age Distribution: There is a clear distinction in age groups among the clusters. Cluster 0 has the youngest individuals, Cluster 2 contains the oldest, and Cluster 1 fills the middle age range.

- Avg_glucose_level Trends: Similarly, avg_glucose_level increases with the age of the clusters. Cluster 0 has the lowest levels, Cluster 1 is in the middle, and Cluster 2 has the highest.

- Separation: The clusters are reasonably well-separated, particularly on the basis of avg_glucose_level, though there is some overlap, especially between Clusters 1 and 2.

Trends and Patterns:

- Clear age progression across clusters, with corresponding increases in glucose levels, suggests age-related health risks.

- Sharp, distinct boundaries between clusters indicate specific demographic groupings based on age and health metrics

GMM Clustering Characteristics

-Cluster 0 (Blue): Similar to K-Means, this cluster contains primarily younger individuals with low avg_glucose_level.

- Cluster 1 (Orange: Mainly older individuals but with a broad range of avg_glucose_level.

- Cluster 2 (Green): A mix of ages, generally middle-aged to older adults, with moderate to high avg_glucose_level.

Observations:

-Age Distribution: Less distinct in terms of age compared to K-Means. Clusters 1 and 2 both cover broader age ranges, and the separation is not as pronounced.

- Avg_glucose_level Trends: There is a greater overlap in avg_glucose_level across the clusters. Clusters 1 and 2, in particular, show significant mixing of avg_glucose_level, suggesting less clear separation based on this feature.

- Probabilistic Nature: The overlapping and less distinct boundaries between clusters reflect the probabilistic nature of GMM, where clusters can share characteristics more fluidly, and data points can belong to multiple clusters with varying probabilities.

Trends and Patterns

- The clusters show more overlap, especially in glucose levels among older adults, reflecting the probabilistic nature of GMM and the complex, overlapping health profiles within this demographic.

- Softer cluster boundaries suggest less distinct group separations, highlighting the complexity and variance within individual health trajectories.

Overall Representation:

Both clustering approaches reflect a gradient of health risks associated with aging and increasing glucose levels. K-Means provides a more segmented view, ideal for targeted interventions based on clear-cut criteria. In contrast, GMM reveals the nuanced, overlapping nature of health profiles, suitable for personalized medicine approaches where individual variance is crucial. These insights can guide health policy decisions, resource allocation, and personalized health monitoring or interventions.

<u>5. Comparative Analysis</u>

- Sharpness of Cluster Boundaries: K-Means shows sharper, more defined boundaries between clusters, making it suitable for applications where distinct segmentation is necessary. GMM, with its softer boundaries, is better for exploratory analysis where clusters may not be strictly separable.

-Age vs. Avg_glucose_level Relationshi: Both methods show an increase in avg_glucose_level with age across the clusters, but GMM provides a less segregated and more blended view, which can be advantageous in capturing more complex patterns.

-Handling Overlap: GMM's ability to handle overlapping clusters is apparent, particularly useful in scenarios where data points genuinely share characteristics with multiple groups.

Comparison of K-Means and Gaussian Mixture Model (GMM) Clustering on the Given Dataset:

1. K-Means Clustering:

-Strengths

- Efficiency K-Means is generally faster and more scalable, making it suitable for large datasets.

- Simplicity: The algorithm's straightforward approach (minimizing variance within clusters) makes it easy to implement and interpret.

- Clear Boundaries: It effectively creates distinct, non-overlapping clusters, which can be beneficial for applications requiring clear demarcation.

- Weaknesses:

- Assumption of Sphericity: K-Means assumes clusters are spherical and of similar size, which might not hold for more complex datasets.

- Sensitivity to Outliers: Outliers can significantly skew the mean calculations, affecting the overall cluster formation.

- Dependency on Initial Points: The final clusters can vary based on the initial randomly selected centroids, potentially leading to sub-optimal clustering.

2. Gaussian Mixture Model (GMM):

- Strengths:

- Flexibility in Cluster Covariance: GMM accommodates clusters that have different shapes and sizes, thanks to its ability to incorporate the covariance structure of the data.

- Soft Clustering: Provides a probabilistic cluster assignment, which is useful for understanding the degree of membership of each data point in multiple clusters.

- Handling Mixed Data Better suited for datasets where clusters overlap or where there is no clear separation.

- Weaknesses

- Computational Complexity: More computationally intensive, especially as the number of dimensions and components grows.

- Sensitivity to Initialization and Local Optima: Like K-Means, GMM is also sensitive to the initialization of parameters and can converge to local optima.

-Requirement for More Parameters: The need to set and potentially guess the number of clusters and the covariance type can complicate setup without sufficient prior knowledge.

Application to the Dataset

-K-Means showed a strong performance in segmenting the dataset into distinct groups based on age and glucose levels. This clear segmentation is particularly useful for direct, actionable insights or interventions where straightforward groupings are beneficial. However, its hard clustering approach might oversimplify the dataset by forcing points into clusters where they might not naturally belong.

- GMM was effective in capturing the intrinsic structures of the dataset, accommodating the inherent overlaps and variations within age and glucose levels. This approach is advantageous for detailed, nuanced analysis, especially in medical or biological contexts where such complexity is typical. However, its softer boundaries might make it less suitable for contexts requiring clear, decisive cluster separation.