NAME: Yashwantej Dyavari Shetty

Dataset

The dataset you'll used for this assignment is diabetes.csv

The target variable is Outcome. The other 8 variables are features.

This dataset was collected from pregnant women. Each row corresponds to a person. An Outcome of 0 means the person is not diagnosed with diabetes. An Outcome of 1 means the person is diagnosed with diabetes.

In [3]:
```
import pandas

diabetes = pandas.read_csv('/Users/yashds/Downloads/Projects/AG6/diabetes.csv'
```

In [4]:
```
diabetes.head(10)
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288
5	5	116	74	0	0	25.6	0.201
6	3	78	50	32	88	31.0	0.248
7	10	115	0	0	0	35.3	0.134
8	2	197	70	45	543	30.5	0.158
9	8	125	96	0	0	0.0	0.232

In [7]:
```
diabetes.var()
```

Out[7]:
```
Pregnancies                    11.354056
Glucose                      1022.248314
BloodPressure                 374.647271
SkinThickness                 254.473245
Insulin                     13281.180078
BMI                            62.159984
DiabetesPedigreeFunction        0.109779
Age                           138.303046
Outcome                         0.227483
dtype: float64
```

In [8]:
```
from sklearn.preprocessing import MinMaxScaler
X = diabetes.drop(columns=['Outcome'])
y = diabetes["Outcome"]
Xscaled = MinMaxScaler().fit_transform(X)
```

```
df = pandas.DataFrame(Xscaled, columns=X.columns)
df.var().sort_values(ascending=False)
```

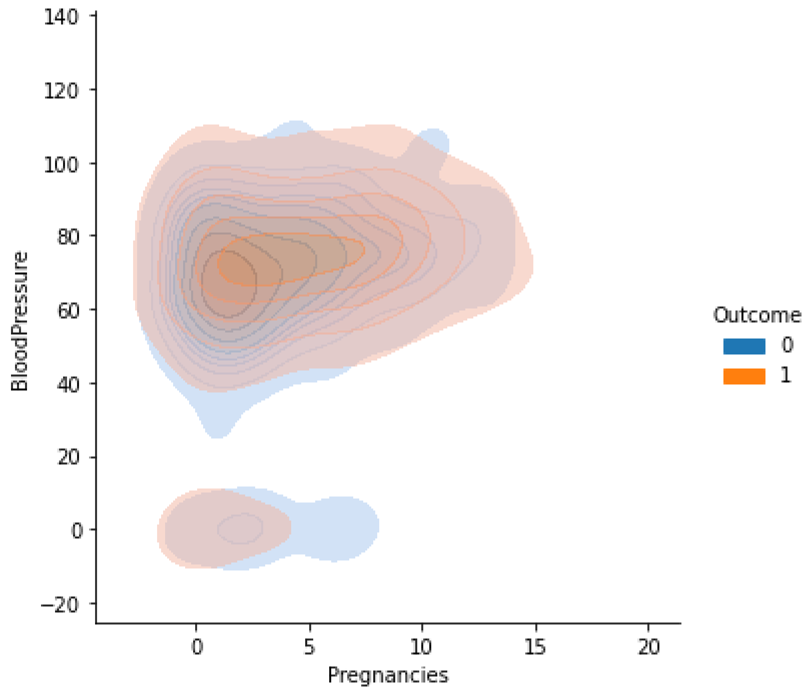
Out[8]:
```
Pregnancies                 0.039287
Age                         0.038418
SkinThickness               0.025964
Glucose                     0.025814
BloodPressure               0.025171
DiabetesPedigreeFunction    0.020014
Insulin                     0.018556
BMI                         0.013806
dtype: float64
```

Outcome: Calculating the correlation between each of these features and the target variable will help us identify which feature the target variable depends on the least. The correlation coefficients between the columns can be determined using the corr method of a Pandas DataFrame in Python.

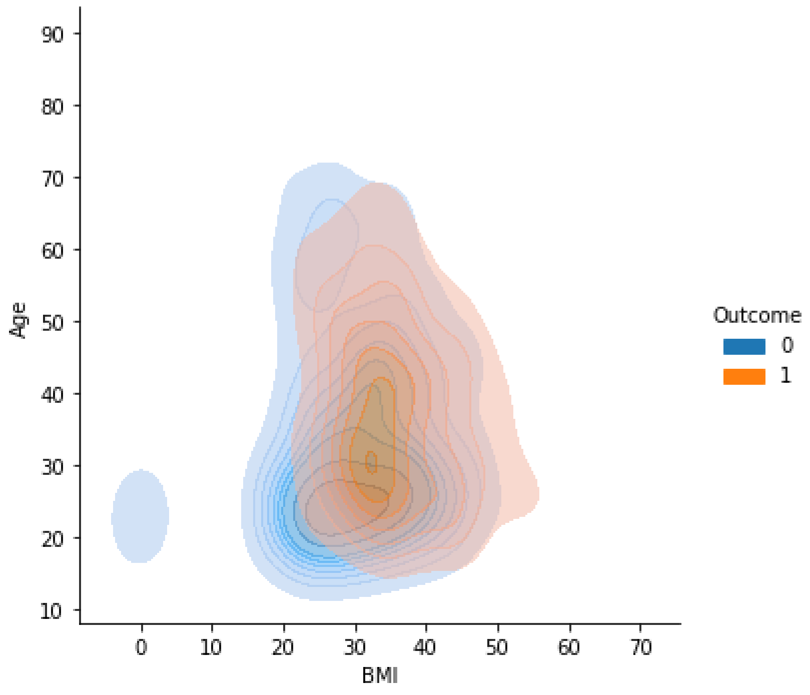
In [9]:
```
X = diabetes[['Pregnancies','BloodPressure','BMI','Age']]
y = diabetes["Outcome"]
Xscaled = MinMaxScaler().fit_transform(X)
df = pandas.DataFrame(Xscaled, columns=X.columns)
df.var().sort_values(ascending=False)
```

Out[9]:
```
Pregnancies      0.039287
Age              0.038418
BloodPressure    0.025171
BMI              0.013806
dtype: float64
```

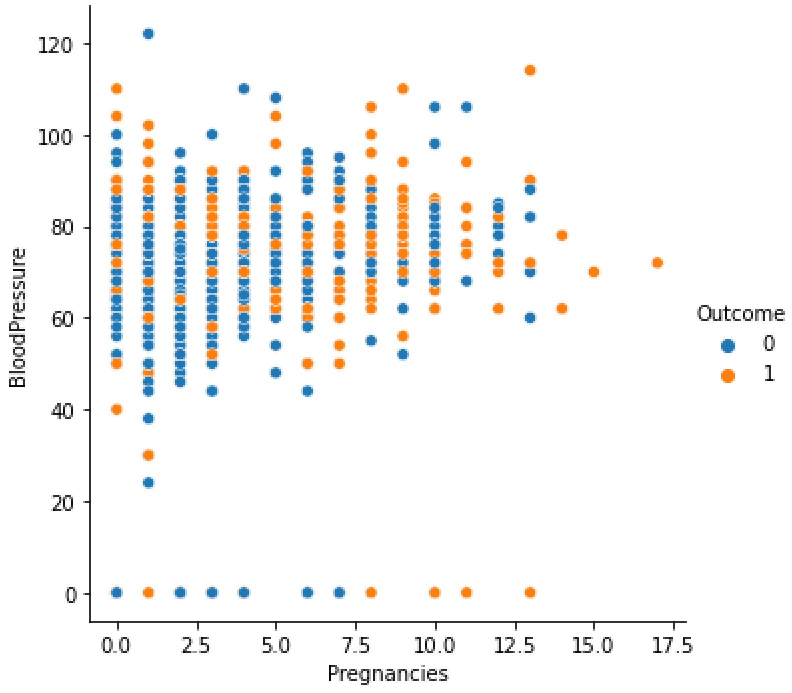
In [10]:
```
import seaborn
from matplotlib import pyplot
seaborn.displot(data=diabetes, x='Pregnancies', y='BloodPressure', hue='Outcome
pyplot.show()
```



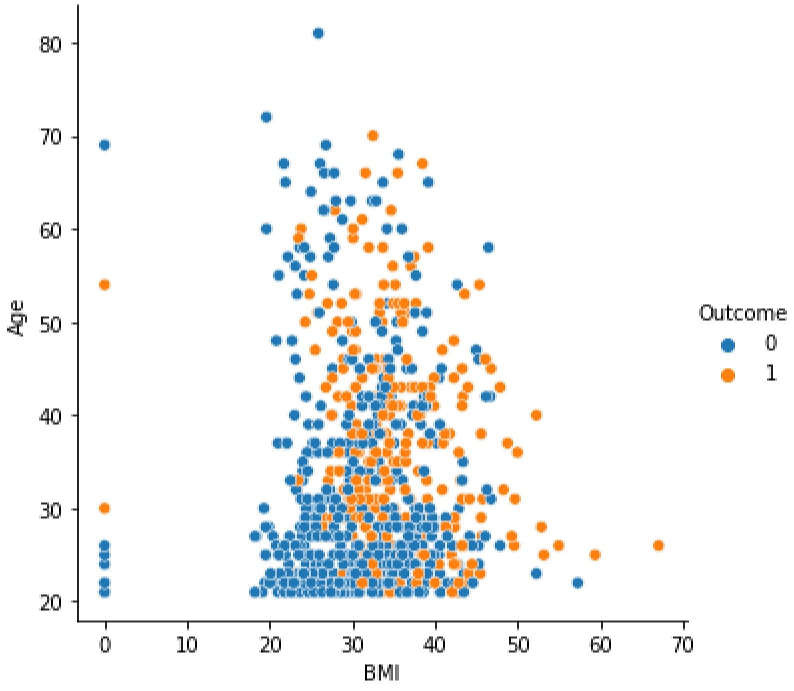
In [11]:
```
seaborn.displot(data=diabetes, x='BMI', y='Age', hue='Outcome', kind='kde', fi
pyplot.show()
```



In [12]:
```
from sklearn.neighbors import KNeighborsClassifier


X = diabetes[['Pregnancies', 'BloodPressure']]
y = diabetes ['Outcome']

model1 = KNeighborsClassifier(n_neighbors=9)
model1.fit(X,y)

from matplotlib import pyplot
import seaborn
seaborn.relplot(data=diabetes, x ='Pregnancies', y='BloodPressure', hue='Outco
pyplot.show()
```



In [13]:
```
from sklearn.neighbors import KNeighborsClassifier


x = diabetes.drop(columns=['Outcome','Glucose','SkinThickness','Insulin','Diab
y = diabetes ['Outcome']

model2 = KNeighborsClassifier(n_neighbors=9)
model2.fit(x,y)
from matplotlib import pyplot
import seaborn
seaborn.relplot(data=diabetes, x ='BMI', y='Age', hue='Outcome')
pyplot.show()
```



Using the data from the following rows as test data to compute the accuracy, precision and recall of the two models\

```
[356, 244, 218, 346, 291, 302, 137, 634, 710, 395, 551, 526,
554, 748, 116,  83, 241, 280, 290, 552]
```

In [15]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

X = diabetes[['Pregnancies', 'BloodPressure']]
y = diabetes['Outcome']
model8 = KNeighborsClassifier(n_neighbors=9)
model8.fit(X,y)
test_data1 = X.iloc[[356, 244, 218, 346, 291, 302, 137, 634, 710, 395, 551, 52
test_data2 = y.iloc[[356, 244, 218, 346, 291, 302, 137, 634, 710, 395, 551, 52
test = model8.predict(test_data1)

accuracy = accuracy_score(test_data2,test)
precision = precision_score(test_data2,test)
recall = recall_score(test_data2,test)

print("Model 1")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

```
Model 1
Accuracy: 0.55
Precision: 0.0
Recall: 0.0
```

In [16]:
```
from sklearn.neighbors import KNeighborsClassifier


X = diabetes[['Pregnancies', 'BloodPressure']]
y = diabetes['Outcome']
model1 = KNeighborsClassifier(n_neighbors=9)
model1.fit(X,y)

samples = diabetes.iloc[[356, 244, 218, 346, 291, 302, 137, 634, 710, 395, 551
samples
```

Out[16]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
356	1	125	50	40	167	33.3	0.9
244	2	146	76	35	194	38.2	0.3
218	5	85	74	22	0	29.0	1.2
346	1	139	46	19	83	28.7	0.6
291	0	107	62	30	74	36.6	0.7
302	5	77	82	41	42	35.8	0.1
137	0	93	60	25	92	28.7	0.5
634	10	92	62	0	0	25.9	0.1
710	3	158	64	13	387	31.2	0.2
395	2	127	58	24	275	27.7	1.6
551	3	84	68	30	106	31.9	0.5
526	1	97	64	19	82	18.2	0.2
554	1	84	64	23	115	36.9	0.4
748	3	187	70	22	200	36.4	0.4
116	5	124	74	0	0	34.0	0.2
83	0	101	65	28	0	24.6	0.2
241	4	91	70	32	88	33.1	0.4
280	0	146	70	0	0	37.9	0.3
290	0	78	88	29	40	36.9	0.4
552	6	114	88	0	0	27.8	0.2

In [17]:
```
result = pandas.DataFrame({
    'GroundTruth': samples['Outcome'],
    'Prediction' : model1.predict( samples[ X.columns ]),
})
```

In [18]:
```
diabetes['Outcome'].value_counts(1).round(2)
```

Out[18]:
```
0    0.65
1    0.35
Name: Outcome, dtype: float64
```

In [19]:
```
result[:]
```

Out[19]:

	GroundTruth	Prediction
356	1	0
244	0	0
218	1	0
346	0	0
291	1	0
302	0	1
137	0	0
634	0	1
710	0	0
395	0	0
551	0	0
526	0	0
554	0	0
748	1	0
116	1	0
83	0	0
241	0	0
280	1	0
290	0	1
552	0	0

In [20]:
```
prediction_type = ['fn', 'tn', 'fn', 'tn', 'fn', 'fp', 'tn', 'fp', 'tn', 'tn',
```

In [21]:
```
result['type'] = prediction_type
result
```

Out[21]:

	GroundTruth	Prediction	type
356	1	0	fn
244	0	0	tn
218	1	0	fn
346	0	0	tn
291	1	0	fn
302	0	1	fp
137	0	0	tn
634	0	1	fp
710	0	0	tn
395	0	0	tn
551	0	0	tn
526	0	0	tn
554	0	0	tn
748	1	0	fn
116	1	0	fn
83	0	0	tn
241	0	0	tn
280	1	0	fn
290	0	1	fp
552	0	0	tn

In [22]:
```
result['type'].value_counts()
```

Out[22]:
```
tn    11
fn     6
fp     3
Name: type, dtype: int64
```

In [23]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score

from sklearn.neighbors import KNeighborsClassifier


x = diabetes[['BMI', 'Age']]
y = diabetes ['Outcome']

model2 = KNeighborsClassifier(n_neighbors=9)
model2.fit(x,y)

test_data3 = x.iloc[[356, 244, 218, 346, 291, 302, 137, 634, 710, 395, 551, 52
test_data4 = y.iloc[[356, 244, 218, 346, 291, 302, 137, 634, 710, 395, 551, 52
test2 = model2.predict(test_data3)

accuracy2 = accuracy_score(test_data4,test2)
precision2= precision_score(test_data4,test2)
```

```
recall2 = recall_score(test_data4,test2)

print("Model 2")
print("Accuracy:", accuracy2)
print("Precision:", precision2)
print("Recall:", recall2)
```

```
Model 2
Accuracy: 0.75
Precision: 0.6666666666666666
Recall: 0.3333333333333333
```

In [24]:
```
from sklearn.neighbors import KNeighborsClassifier


x = diabetes[['BMI', 'Age']]
y = diabetes['Outcome']
model2 = KNeighborsClassifier(n_neighbors=9)
model2.fit(x,y)

samples = diabetes.iloc[[356, 244, 218, 346, 291, 302, 137, 634, 710, 395, 551
samples
```

Out[24]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
356	1	125	50	40	167	33.3	0.9
244	2	146	76	35	194	38.2	0.3
218	5	85	74	22	0	29.0	1.2
346	1	139	46	19	83	28.7	0.6
291	0	107	62	30	74	36.6	0.7
302	5	77	82	41	42	35.8	0.1
137	0	93	60	25	92	28.7	0.5
634	10	92	62	0	0	25.9	0.1
710	3	158	64	13	387	31.2	0.2
395	2	127	58	24	275	27.7	1.6
551	3	84	68	30	106	31.9	0.5
526	1	97	64	19	82	18.2	0.2
554	1	84	64	23	115	36.9	0.4
748	3	187	70	22	200	36.4	0.4
116	5	124	74	0	0	34.0	0.2
83	0	101	65	28	0	24.6	0.2
241	4	91	70	32	88	33.1	0.4
280	0	146	70	0	0	37.9	0.3
290	0	78	88	29	40	36.9	0.4
552	6	114	88	0	0	27.8	0.2

In [25]:
```
result = pandas.DataFrame({
    'GroundTruth': samples['Outcome'],
    'Prediction' : model2.predict( samples[ x.columns ]),
})
```

In [26]:
```
diabetes['Outcome'].value_counts(1).round(2)
```

Out[26]:
```
0    0.65
1    0.35
Name: Outcome, dtype: float64
```

In [28]:
```
result[:]
```

Out[28]:

	GroundTruth	Prediction
356	1	0
244	0	0
218	1	0
346	0	0
291	1	0
302	0	1
137	0	0
634	0	0
710	0	0
395	0	0
551	0	0
526	0	0
554	0	0
748	1	1
116	1	1
83	0	0
241	0	0
280	1	0
290	0	0
552	0	0

In [29]:
```
prediction_type = ['fn', 'tn', 'fn', 'tn', 'fn', 'fp', 'tn', 'tn', 'tn', 'tn',
```

In [30]:
```
result['type'] = prediction_type
result
```

Out[30]:

	GroundTruth	Prediction	type
356	1	0	fn
244	0	0	tn
218	1	0	fn
346	0	0	tn
291	1	0	fn
302	0	1	fp
137	0	0	tn
634	0	0	tn
710	0	0	tn
395	0	0	tn
551	0	0	tn
526	0	0	tn
554	0	0	tn
748	1	1	tp
116	1	1	tp
83	0	0	tn
241	0	0	tn
280	1	0	fn
290	0	0	tn
552	0	0	tn

In [31]:
```
result['type'].value_counts()
```

Out[31]:
```
tn    13
fn     4
tp     2
fp     1
Name: type, dtype: int64
```

Choosing the 4 most relevant features using f_classif, and build a KNN model with these 4 features.

Reporting the accuracy, precision, and recall of the model on the test data obtained from the same rows as shown.

In [32]:
```
from sklearn.feature_selection import SelectKBest,  f_classif

X = diabetes.drop(columns=['Outcome'])
y = diabetes['Outcome']
selector = SelectKBest(score_func=f_classif, k=4)
X_new = selector.fit_transform(X, y)
mask = selector.get_support(indices=True)
```

```
selected_features = X.columns[mask]
print('selected_features:',selected_features)
```

```
selected_features: Index(['Pregnancies', 'Glucose', 'BMI', 'Age'], dtype='obje
ct')
```

In [33]:
```
from sklearn.neighbors import KNeighborsClassifier

X = diabetes[selected_features]
y = diabetes['Outcome']
model = KNeighborsClassifier(n_neighbors=9)
model.fit(X, y)

from sklearn.metrics import accuracy_score, precision_score, recall_score

X = X.iloc[[356, 244, 218, 346, 291, 302, 137, 634, 710, 395, 551, 526, 554, 7
y = y.iloc[[356, 244, 218, 346, 291, 302, 137, 634, 710, 395, 551, 526, 554, 7

model3 = model.predict(X)

accuracy = accuracy_score(y, model3)
precision = precision_score(y, model3)
recall = recall_score(y, model3)

print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
```

```
Accuracy: 0.75
Precision: 0.6666666666666666
Recall: 0.3333333333333333
```

In [34]:
```
y = diabetes['Outcome']
X = diabetes[selected_features]

samples = diabetes.iloc[[356, 244, 218, 346, 291, 302, 137, 634, 710, 395, 551
samples
```

Out[34]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
356	1	125	50	40	167	33.3	0.9
244	2	146	76	35	194	38.2	0.3
218	5	85	74	22	0	29.0	1.2
346	1	139	46	19	83	28.7	0.6
291	0	107	62	30	74	36.6	0.7
302	5	77	82	41	42	35.8	0.1
137	0	93	60	25	92	28.7	0.5
634	10	92	62	0	0	25.9	0.1
710	3	158	64	13	387	31.2	0.2
395	2	127	58	24	275	27.7	1.6
551	3	84	68	30	106	31.9	0.5
526	1	97	64	19	82	18.2	0.2
554	1	84	64	23	115	36.9	0.4
748	3	187	70	22	200	36.4	0.4
116	5	124	74	0	0	34.0	0.2
83	0	101	65	28	0	24.6	0.2
241	4	91	70	32	88	33.1	0.4
280	0	146	70	0	0	37.9	0.3
290	0	78	88	29	40	36.9	0.4
552	6	114	88	0	0	27.8	0.2

In [35]:
```
result = pandas.DataFrame({
    'GroundTruth': samples['Outcome'],
    'Prediction' : model.predict( samples[ X.columns ]),
})
```

In [36]:
```
diabetes['Outcome'].value_counts(1).round(2)
```

Out[36]:
```
0    0.65
1    0.35
Name: Outcome, dtype: float64
```

In [37]:
```
result[:]
```

Out[37]:

	GroundTruth	Prediction
356	1	1
244	0	0
218	1	0
346	0	0
291	1	0
302	0	0
137	0	0
634	0	0
710	0	1
395	0	0
551	0	0
526	0	0
554	0	0
748	1	1
116	1	0
83	0	0
241	0	0
280	1	0
290	0	0
552	0	0

In [38]:
```
prediction_type = ['tp', 'tn', 'fn', 'tn', 'fn', 'tn', 'tn', 'tn', 'fp', 'tn',
```

In [39]:
```
result['type'] = prediction_type
result
```

Out[39]:

	GroundTruth	Prediction	type
356	1	1	tp
244	0	0	tn
218	1	0	fn
346	0	0	tn
291	1	0	fn
302	0	0	tn
137	0	0	tn
634	0	0	tn
710	0	1	fp
395	0	0	tn
551	0	0	tn
526	0	0	tn
554	0	0	tn
748	1	1	tp
116	1	0	fn
83	0	0	tn
241	0	0	tn
280	1	0	fn
290	0	0	tn
552	0	0	tn

In [40]:
```
result['type'].value_counts()
```

Out[40]:
```
tn    13
fn     4
tp     2
fp     1
Name: type, dtype: int64
```

Create a new column called "AgeGroup", which has the following values:

- Group1 - under 25 years old
- Group2 - from 25 to under 30 years old
- Group3 - from 30 to under 40 years old
- Group4 - from 40 to under 50 years old
- Group5 - from 50 years old and up

Steps to carryout:

- The code defines a function called AgeGroup which takes an input parameter age. The function then uses a series of if-elif-else statements to determine which age group the input age falls into.

- The code then applies this AgeGroup function to the 'Age' column in the diabetes DataFrame using the apply() method. The resulting output is a new column called 'AgeGroup' in the diabetes DataFrame that contains the age group for each individual in the 'Age' column.

In [41]:
```
def AgeGroup(age):
    if age < 25:
        return "Group1"

    elif age < 30:
        return "Group2"

    elif age < 40:
        return "Group3"

    elif age < 50:
        return "Group4"

    else:
        return "Group5"

diabetes['AgeGroup'] = diabetes['Age'].apply(AgeGroup)
```

- Report: The code is grouping the ages of individuals in the diabetes DataFrame into five categories.

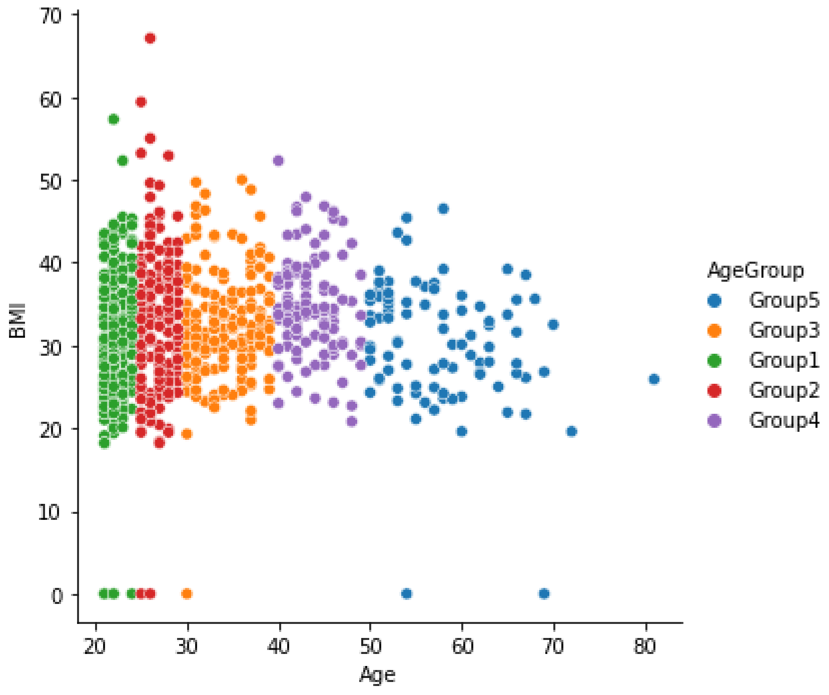
Steps to carryout:

- In the code we use the groupby method to group the data in the diabetes DataFrame by the 'AgeGroup' column, and then calculating the mean, median, and standard deviation of the 'BMI' column for each age group.
- The resulting mean, median, and std variables for each age group.
- Then we used the seaborn and matplotlib.pyplot libraries to create a scatter plot using the relplot function. The plot shows the relationship between age and BMI in the diabetes dataset, with each data point colored based on the individual's age group.
- The resulting plot is displayed using the pyplot.show().

In [42]:
```
mean = diabetes.groupby('AgeGroup')['BMI'].mean()
median = diabetes.groupby('AgeGroup')['BMI'].median()
std = diabetes.groupby('AgeGroup')['BMI'].std()


print('mean:', mean)
print('median:', median)
print('std:', std)
```

```
mean: AgeGroup
Group1    30.011416
Group2    33.096610
Group3    32.472121
Group4    34.617797
Group5    30.302247
Name: BMI, dtype: float64
median: AgeGroup
Group1    29.80
Group2    33.20
Group3    32.00
Group4    33.95
Group5    30.30
Name: BMI, dtype: float64
std: AgeGroup
Group1    8.435030
Group2    8.714577
Group3    6.737134
Group4    5.981533
Group5    7.529683
Name: BMI, dtype: float64
```

In [43]:
```
import seaborn
import matplotlib.pyplot as pyplot
seaborn.relplot(data=diabetes, x='Age', y='BMI', hue = 'AgeGroup')
pyplot.show()
```



- Report: It shows numerical and visual relationship between age and BMI in the diabetes dataset by calculating statistics for each age group and each data point colored based on the individual's age group. The plot can be used to identify any patterns, and to visually explore the relationship between age group and BMI.

Creating a new column for "BMILevel", which has values: UnderWeight, Healthy, Overweight, Obesity, Class3Obesity, based on the information from this website:

https://www.cdc.gov/obesity/basics/adult-defining.html

Steps to carryout:

- It defines a function called BMILevel which takes an input parameter bmi. The function then uses a series of if-elif-else statements to determine which BMI level the input BMI falls into. The function returns the corresponding BMI level for the input BMI.
- Then it applies this BMILevel function to the 'BMI' column in the diabetes DataFrame using the apply() method. The resulting output is a new column called 'BMILevel' in the diabetes DataFrame that contains the BMI level for each individual in the 'BMI' column.

In [44]:
```
def BMILevel(bmi):
    if bmi < 18.5:
        return "UnderWeight"
    elif bmi < 25:
        return "Healthy"
    elif bmi < 30:
        return "Overweight"
    elif bmi < 40:
        return "Obesity"
    else:
        return "Class3Obesity"

diabetes['BMILevel'] = diabetes['BMI'].apply(BMILevel)
```

- Report: The code is grouping the BMI of individuals in the diabetes DataFrame into five categories

Steps to carryout:

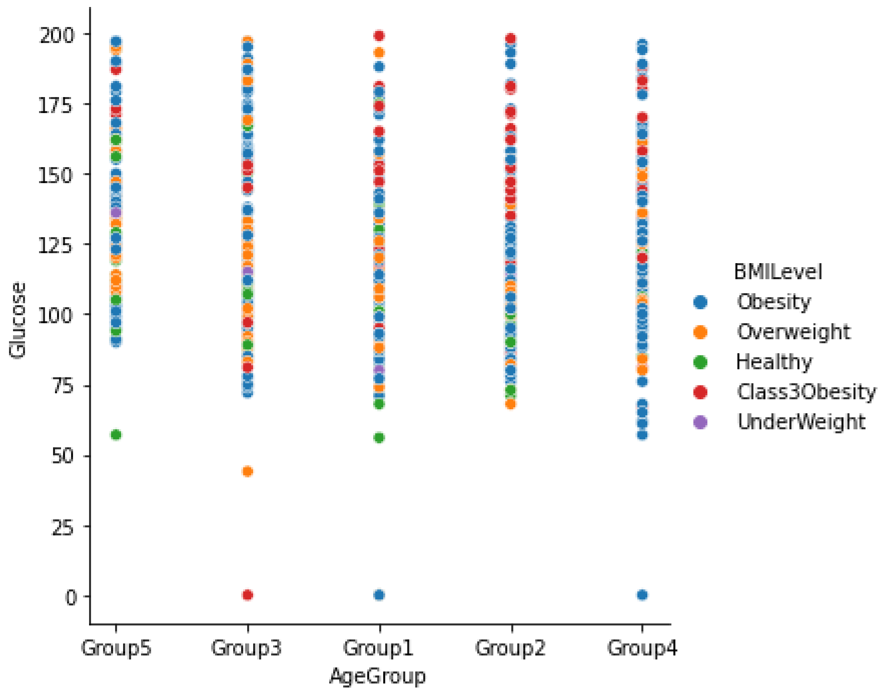
- In the code we used groupby method to group the data in the diabetes DataFrame by both the 'AgeGroup' and 'BMILevel' columns, and then calculating the mean, median, and standard deviation of the 'Glucose' column for each combination of age group and BMI level.
- Resulting mean, median, and std variables for each combination of age group and BMI level.
- We used the matplotlib and seaborn libraries to create a scatter plot using the relplot function. The plot shows the relationship between age group and glucose level in the diabetes dataset, with each data point colored based on the individual's BMI level.
- Resulting plot is displayed using the pyplot.show().

In [45]:
```
mean = diabetes.groupby(['AgeGroup', 'BMILevel'])['Glucose'].mean()
median = diabetes.groupby(['AgeGroup', 'BMILevel'])['Glucose'].median()
std = diabetes.groupby(['AgeGroup', 'BMILevel'])['Glucose'].std()
```

```
print('mean:', mean)
print('median:', median)
print('std:', std)
```

```
mean: AgeGroup  BMILevel
Group1    Class3Obesity    132.153846
          Healthy          103.975610
          Obesity          112.390244
          Overweight       106.548387
          UnderWeight       94.625000
Group2    Class3Obesity    140.758621
          Healthy           96.318182
          Obesity          117.079545
          Overweight       113.705882
          UnderWeight       98.750000
Group3    Class3Obesity    119.380952
          Healthy          113.941176
          Obesity          132.694118
          Overweight       118.000000
          UnderWeight      115.000000
Group4    Class3Obesity    136.500000
          Healthy          109.500000
          Obesity          123.723684
          Overweight       121.722222
Group5    Class3Obesity    160.000000
          Healthy          127.375000
          Obesity          143.162791
          Overweight       138.541667
          UnderWeight      130.500000
Name: Glucose, dtype: float64
median: AgeGroup  BMILevel
Group1    Class3Obesity    131.5
          Healthy          101.0
          Obesity          111.0
          Overweight       106.5
          UnderWeight       97.0
Group2    Class3Obesity    141.0
          Healthy           96.0
          Obesity          113.5
          Overweight       110.5
          UnderWeight       99.0
Group3    Class3Obesity    128.0
          Healthy          108.0
          Obesity          125.0
          Overweight       117.0
          UnderWeight      115.0
Group4    Class3Obesity    139.0
          Healthy          112.0
          Obesity          122.5
          Overweight       123.5
Group5    Class3Obesity    172.0
          Healthy          128.5
          Obesity          145.0
          Overweight       135.5
          UnderWeight      130.5
Name: Glucose, dtype: float64
std: AgeGroup  BMILevel
Group1    Class3Obesity    31.238044
          Healthy          28.612487
          Obesity          28.707446
          Overweight       26.293849
          UnderWeight      14.500616
Group2    Class3Obesity    30.236751
          Healthy          11.581539
          Obesity          28.165856
          Overweight       23.289391
          UnderWeight      13.301002
Group3    Class3Obesity    35.409711
          Healthy          27.659697
          Obesity          32.659872
          Overweight       32.329553
          UnderWeight            NaN
Group4    Class3Obesity    34.512146
          Healthy          13.881643
          Obesity          36.468653
          Overweight       24.642874
Group5    Class3Obesity    34.737108
          Healthy          33.079450
          Obesity          31.991763
          Overweight       24.636451
          UnderWeight       7.778175
Name: Glucose, dtype: float64
```

In [46]:
```python
from matplotlib import pyplot
import seaborn
seaborn.relplot(data=diabetes, x='AgeGroup', y='Glucose', hue='BMILevel')
pyplot.show()
```



- Report: It shows numerical and visual relationship between age group and glucose level in the diabetes dataset with each data point colored based on the individual's BMI level. The plot can be used to identify any patterns, and to visually explore the relationship between age group, BMI level, and glucose level in individuals with diabetes.

Creating a set of features that will be used for modeling. These features are:

- Blood pressure
- Insulin
- Age group

- BMI level

Using standard scaling to rescale blood pressure and insulin.

Using the categorical features age group and BMI level into numerical features.

Steps to carryout:

- The code performs the following steps:
- Importing the pandas library as pd and the StandardScaler class from the sklearn.preprocessing module.
- Selecting specific columns ('BloodPressure', 'Insulin', 'AgeGroup', 'BMILevel') from the 'diabetes' dataframe and create a copy of it as X.
- Converting the categorical variables 'AgeGroup' and 'BMILevel' in X to numerical using one-hot encoding, by using the get_dummies() function. This creates new columns for each unique value in these categorical columns.
- Creating an instance of the StandardScaler class and apply it to the 'BloodPressure' and 'Insulin' columns of X to normalize their values.
- Then replacing the original 'BloodPressure' and 'Insulin' columns in X with the normalized values.

In [51]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = diabetes[['BloodPressure', 'Insulin', 'AgeGroup', 'BMILevel']].copy()
X = pandas.get_dummies(X, columns=['AgeGroup','BMILevel'])
scaled_data = scaler.fit_transform(X[['BloodPressure', 'Insulin']])
X['BloodPressure'] = scaled_data[ : , 0]
X['Insulin'] = scaled_data[ : , 1]
X
```

Out[51]:

	BloodPressure	Insulin	AgeGroup_Group1	AgeGroup_Group2	AgeGroup_Group3	AgeGr
0	0.149641	-0.692891	0	0	0	
1	-0.160546	-0.692891	0	0	1	
2	-0.263941	-0.692891	0	0	1	
3	-0.160546	0.123302	1	0	0	
4	-1.504687	0.765836	0	0	1	
...	...	...	...	...	...	
763	0.356432	0.870031	0	0	0	
764	0.046245	-0.692891	0	1	0	
765	0.149641	0.279594	0	0	1	
766	-0.470732	-0.692891	0	0	0	
767	0.046245	-0.692891	1	0	0	

768 rows × 12 columns

In [52]:
```
X.describe()
```

Out[52]:

	BloodPressure	Insulin	AgeGroup_Group1	AgeGroup_Group2	AgeGroup_Group3
count	7.680000e+02	7.680000e+02	768.000000	768.000000	768.000000
mean	-1.327244e-17	-3.556183e-17	0.285156	0.230469	0.214844
std	1.000652e+00	1.000652e+00	0.451783	0.421407	0.410982
min	-3.572597e+00	-6.928906e-01	0.000000	0.000000	0.000000
25%	-3.673367e-01	-6.928906e-01	0.000000	0.000000	0.000000
50%	1.496408e-01	-4.280622e-01	0.000000	0.000000	0.000000
75%	5.632228e-01	4.120079e-01	1.000000	0.000000	0.000000
max	2.734528e+00	6.652839e+00	1.000000	1.000000	1.000000

- Report: This code performs feature selection, data normalization, and categorical encoding, which are important steps in data preprocessing for machine learning algorithms.

Suppose that we want to compare these two methods of cross validation: shuffle and split, and 12-fold cross validation.

With shuffle and split, you will probably want to specify the same number test data points and training data points as 12-fold cross validation.

- The code uses ShuffleSplit cross-validation with 12 splits on the diabetes dataset. The diabetes dataset is split into train and test sets in each fold of cross-validation using the ss.split method.

- The n_test_folds variable is calculated as the number of data points in each test fold, which is equal to the total number of data points divided by the number of folds.

- In each iteration of the loop, the indices of the data points in the train and test sets are stored in the train_index and test_index variables, respectively. The number of data points in the train and test sets is calculated using the n_train and n_test variables, respectively.

In [53]:
```
from sklearn.model_selection import KFold, ShuffleSplit
N = len(diabetes)

kf = KFold(n_splits=12)
n_folds = 12
n_test_folds = N/n_folds
ss = ShuffleSplit(n_splits=12, test_size=n_test_folds/N)
for train_index, test_index in ss.split(diabetes):
    n_test = len(test_index)
    print(f"Number of test data points in this fold of shuffle and split CV: {
    n_train = N - n_test
    print(f"Number of train data points in this fold of shuffle and split CV: 
```

```
Number of test data points in this fold of shuffle and split CV: 64
Number of train data points in this fold of shuffle and split CV: 704
Number of test data points in this fold of shuffle and split CV: 64
Number of train data points in this fold of shuffle and split CV: 704
Number of test data points in this fold of shuffle and split CV: 64
Number of train data points in this fold of shuffle and split CV: 704
Number of test data points in this fold of shuffle and split CV: 64
Number of train data points in this fold of shuffle and split CV: 704
Number of test data points in this fold of shuffle and split CV: 64
Number of train data points in this fold of shuffle and split CV: 704
Number of test data points in this fold of shuffle and split CV: 64
Number of train data points in this fold of shuffle and split CV: 704
Number of test data points in this fold of shuffle and split CV: 64
Number of train data points in this fold of shuffle and split CV: 704
Number of test data points in this fold of shuffle and split CV: 64
Number of train data points in this fold of shuffle and split CV: 704
Number of test data points in this fold of shuffle and split CV: 64
Number of train data points in this fold of shuffle and split CV: 704
Number of test data points in this fold of shuffle and split CV: 64
Number of train data points in this fold of shuffle and split CV: 704
Number of test data points in this fold of shuffle and split CV: 64
Number of train data points in this fold of shuffle and split CV: 704
Number of test data points in this fold of shuffle and split CV: 64
Number of train data points in this fold of shuffle and split CV: 704
```

- Report: The output will be the loop display the number of data points in each train and test set for each fold of cross-validation.

Steps to carryout:

- The code implementing a function called five_five_fold_cross_validate which performs 5-fold cross-validation with shuffling on a given model, X, and y.

Here are the steps:

- The function takes model, X, and y as inputs.
- A list called scores is initialized to store the f1 macro scores obtained from each fold.
- The for loop iterates 5 times and initializes a new KFold cross-validator with 5 splits and shuffling set to True.
- The cross_validate function is called to evaluate the model using the KFold cross-validator, X, and y, with scoring='f1_macro' to evaluate the model's performance. The results of the cross-validation are stored in the results dictionary.
- The f1 macro scores obtained from the cross-validation are appended to the scores list using the extend() method.
- The mean of the f1 macro scores is calculated using the np.mean() function and stored in a variable called scores_mean.

In [56]:
```
import pandas

diabetes = pandas.read_csv('/Users/yashds/Downloads/Projects/AG6/diabetes.csv'

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold, cross_validate
import numpy as np
model=KNeighborsClassifier()

from sklearn.metrics import f1_score
X = diabetes.drop('Outcome', axis= 1)
y = diabetes['Outcome']
scores=[]
def five_five_fold_cross_validate(model, X, y):
    for i in range(5):
        kf = KFold(n_splits=5, shuffle=True)
        results = cross_validate(model, X, y, cv=kf, scoring='f1_macro')
        scores.append(results['test_score'].tolist())
        scores_mean = np.mean(scores)
        return scores_mean


five_five_fold_cross_validate(model, X, y)
```

Out[56]:
```
0.6742478245317113
```

- Report: the output is scores_mean which is average f1 score of the cross validation

To Validate a 10-nearest neighbor model on the diabetes dataset using the five_five_fold_cross_validate method above.

Use "Outcome" for y, and any two features for X. Trying to get as high scores as you can by choose two good features.

Steps to Carryout:

- Import KNeighborsClassifier and KFold modules from sklearn
- Create a KNeighborsClassifier model with n_neighbors = 10
- Create a variable X1, which contains the 'Glucose' and 'BMI' columns of the diabetes dataset
- Create a variable y, which contains the 'Outcome' column of the diabetes dataset
- Call the five_five_fold_cross_validate function with model1, X1, and y as arguments and store the output in a variable called result1
- Repeat steps 3-5 for X2 = ['Pregnancies', 'BMI'], X3 = ['Pregnancies', 'Glucose'], X4 = ['Insulin', 'SkinThickness'] and X5 = ['Insulin', 'SkinThickness']

In [58]:
```
model1 = KNeighborsClassifier(n_neighbors=10)
X1 = diabetes[['Glucose','BMI']]
y = diabetes['Outcome']
five_five_fold_cross_validate(model1,X1,y)
```

Out[58]:
```
0.6876746134642723
```

In [60]:
```
model2 = KNeighborsClassifier(n_neighbors=10)
X2 = diabetes[['Pregnancies','BMI']]
y = diabetes['Outcome']
five_five_fold_cross_validate(model2,X2,y)
```

Out[60]:
```
0.6577492610967869
```

In [61]:
```
model3 = KNeighborsClassifier(n_neighbors=10)
X3 = diabetes[['Pregnancies','Glucose']]
y = diabetes['Outcome']
five_five_fold_cross_validate(model3,X3,y)
```

Out[61]:
```
0.6555341123798364
```

In [62]:
```
model4 = KNeighborsClassifier(n_neighbors=10)
X4 = diabetes[['Pregnancies','BMI']]
y = diabetes['Outcome']
five_five_fold_cross_validate(model4,X4,y)
```

Out[62]:
```
0.6434768138819735
```

In [63]:
```
model5 = KNeighborsClassifier(n_neighbors=10)
X5 = diabetes[['Insulin','SkinThickness']]
y = diabetes['Outcome']
five_five_fold_cross_validate(model5,X5,y)
```

Out[63]:
```
0.6234613305970189
```

- Report: Model 1 giving the high scores when compared to all other models

Validate a 10-nearest neighbor model on the diabetes dataset using the five_five_fold_cross_validate method above.

Use "Outcome" for y.

For X, choose the first two principal components obtained by PCA. To train PCA, use all features of diabetes.

Steps to Carryot:

- In this code, PCA (Principal Component Analysis) is used to reduce the dimensionality of the dataset to 2. The diabetes dataset is loaded and the target variable 'Outcome' is removed from the dataset to obtain the feature matrix X and the target vector y.
- Then, PCA is applied to X with n_components=2, which means that the resulting dataset will have two dimensions.
- After applying PCA, the KFold method from scikit-learn is used to generate 12 folds of the dataset for cross-validation. The cross_validate function from scikit-learn is used to perform cross-validation on the model using the reduced dataset X_pca and the target vector y.

In [64]:
```
from sklearn.decomposition import PCA
X = diabetes.drop('Outcome', axis= 1)
y = diabetes['Outcome']
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

scores = cross_validate(model, X_pca, y, cv=kf, scoring = ['accuracy'])
scores['test_accuracy'].mean()
```
Out[64]:
```
0.7174479166666666
```

- The scoring parameter is set to 'accuracy' which means that the accuracy score will be calculated for each fold.

In [65]:
```
pca.explained_variance_ratio_.round(3)
```
Out[65]:
```
array([0.889, 0.062])
```

In [66]:
```
diabetes[['pca1','pca2']] = X_pca
```

In [67]:
```
diabetes.sample(3)
```
Out[67]:

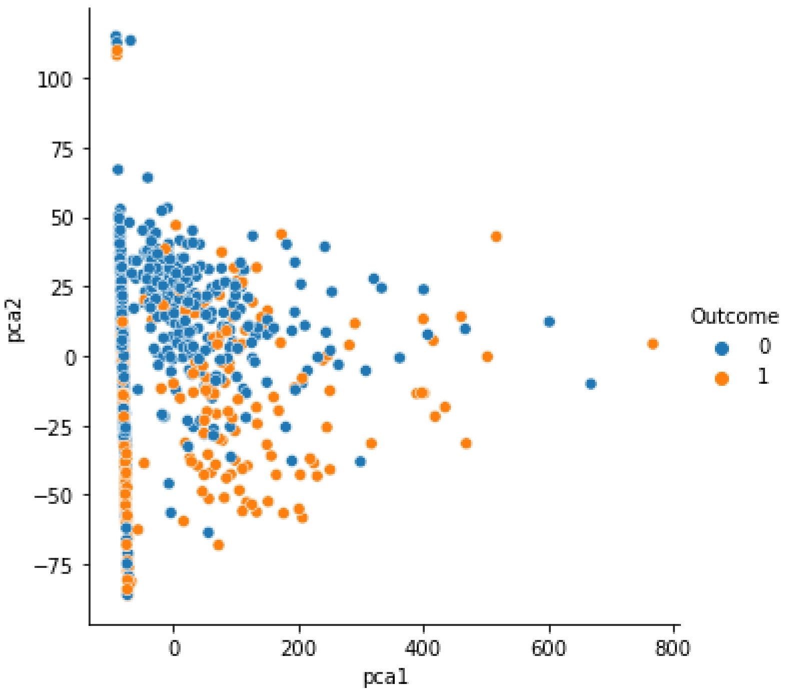
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
329	6	105	70	32	68	30.8	0.
532	1	86	66	52	65	41.3	0.
483	0	84	82	31	125	38.2	0.2

In [68]:
```
df = pandas.DataFrame(pca.components_.T,
                      index=X.columns,
                      columns=['pca1','pca2'])
df.round(2)
```

Out[68]:

	pca1	pca2
Pregnancies	-0.00	-0.02
Glucose	0.10	-0.97
BloodPressure	0.02	-0.14
SkinThickness	0.06	0.06
Insulin	0.99	0.09
BMI	0.01	-0.05
DiabetesPedigreeFunction	0.00	-0.00
Age	-0.00	-0.14

In [76]:
```
import seaborn
from matplotlib import pyplot
seaborn.relplot(data=diabetes, x='pca1', y='pca2', hue='Outcome')
pyplot.show()
```



To Validate a most-frequent dummy classifier on the diabetes dataset using the five_five_fold_cross_validate method above.

Use "Outcome" for y.

For X, choose the first two principal components obtained by PCA. To train PCA, use all features of diabetes.

Steps to Carryout:

- The code imports the DummyClassifier and PCA classes from scikit-learn, and loads the diabetes dataset into the variables X and y.

- A PCA object is created with n_components=2, indicating that the data should be reduced to 2 dimensions.
- The fit_transform() method of the PCA object is called on the X dataset, creating a transformed dataset X_pca with 2 dimensions.
- A DummyClassifier object is created with the strategy parameter set to 'most_frequent', indicating that the classifier should predict the most frequent class in the training data.
- The cross_validate() function from scikit-learn is called with the dummy classifier, X_pca data and y labels and 12-fold cross-validation object kf. The resulting dictionary object result is stored.
- The five_five_fold_cross_validate() function is called with the dummy classifier, X_pca data and y labels. This function performs 5x5-fold cross-validation and returns the mean F1 score.
- The steps above are used to evaluate the performance of the DummyClassifier using PCA transformed data with 2 dimensions, and cross-validate with 12 folds.

In [77]:
```
from sklearn.dummy import DummyClassifier
from sklearn.decomposition import PCA
import numpy as np

X = diabetes.drop('Outcome', axis= 1)
y = diabetes['Outcome']
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
dummy = DummyClassifier(strategy='most_frequent')
five_five_fold_cross_validate(dummy, X_pca, y)
```

Out[77]:
```
0.3935686328517544
```

Some of the features have missing values. Unfortunately, in this dataset, missing values are not indicated as "nan". Therefore, if you use pandas' "dropna", it won't work.

However, if we understand the meanings of the features, you can guess which ones have missing values. For example, blood pressure should not be less than 20.

To Find the features that you think have missing values in the dataset.

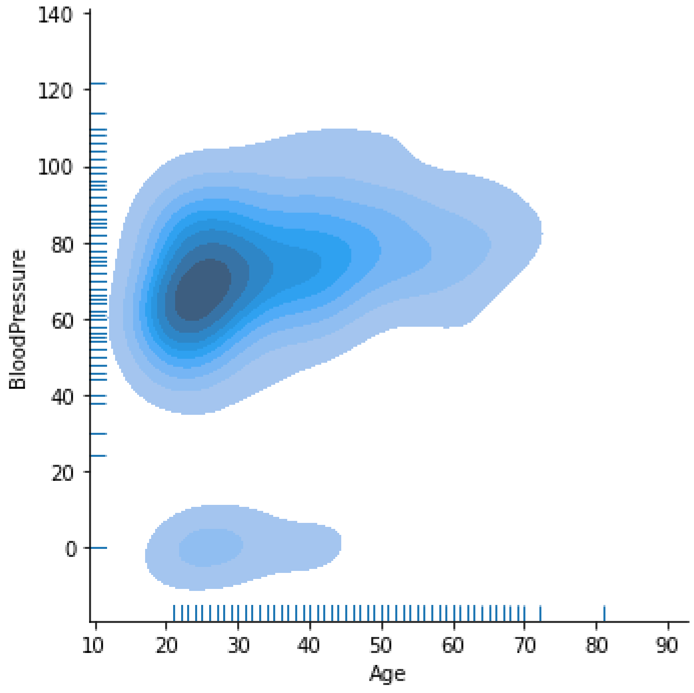
In [78]:
```
for column in diabetes.columns:
    unique_values = diabetes[column].unique()
    print(column, unique_values)
```

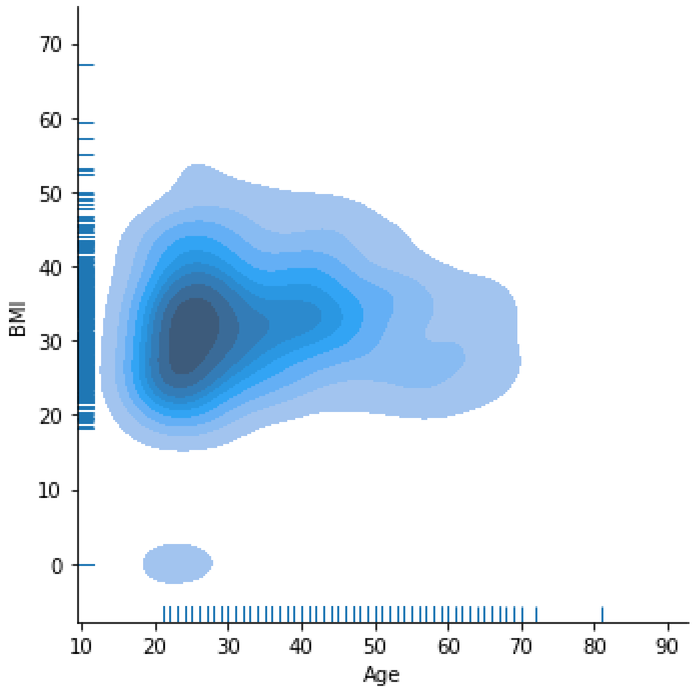
```
Pregnancies [ 6  1  8  0  5  3 10  2  4  7  9 11 13 15 17 12 14]
Glucose [148  85 183  89 137 116  78 115 197 125 110 168 139 189 166 100 118 1
07
 103 126  99 196 119 143 147  97 145 117 109 158  88  92 122 138 102  90
 111 180 133 106 171 159 146  71 105 101 176 150  73 187  84  44 141 114
  95 129  79   0  62 131 112 113  74  83 136  80 123  81 134 142 144  93
 163 151  96 155  76 160 124 162 132 120 173 170 128 108 154  57 156 153
 188 152 104  87  75 179 130 194 181 135 184 140 177 164  91 165  86 193
 191 161 167  77 182 157 178  61  98 127  82  72 172  94 175 195  68 186
 198 121  67 174 199  56 169 149  65 190]
BloodPressure [ 72  66  64  40  74  50   0  70  96  92  80  60  84  30  88  90
94  76
  82  75  58  78  68 110  56  62  85  86  48  44  65 108  55 122  54  52
  98 104  95  46 102 100  61  24  38 106 114]
SkinThickness [35 29  0 23 32 45 19 47 38 30 41 33 26 15 36 11 31 37 42 25 18
24 39 27
 21 34 10 60 13 20 22 28 54 40 51 56 14 17 50 44 12 46 16  7 52 43 48  8
 49 63 99]
Insulin [  0  94 168  88 543 846 175 230  83  96 235 146 115 140 110 245  54 1
92
 207  70 240  82  36  23 300 342 304 142 128  38 100  90 270  71 125 176
  48  64 228  76 220  40 152  18 135 495  37  51  99 145 225  49  50  92
 325  63 284 119 204 155 485  53 114 105 285 156  78 130  55  58 160 210
 318  44 190 280  87 271 129 120 478  56  32 744 370  45 194 680 402 258
 375 150  67  57 116 278 122 545  75  74 182 360 215 184  42 132 148 180
 205  85 231  29  68  52 255 171  73 108  43 167 249 293  66 465  89 158
  84  72  59  81 196 415 275 165 579 310  61 474 170 277  60  14  95 237
 191 328 250 480 265 193  79  86 326 188 106  65 166 274  77 126 330 600
 185  25  41 272 321 144  15 183  91  46 440 159 540 200 335 387  22 291
 392 178 127 510  16 112]
BMI [33.6 26.6 23.3 28.1 43.1 25.6 31.  35.3 30.5  0.  37.6 38.  27.1 30.1
 25.8 30.  45.8 29.6 43.3 34.6 39.3 35.4 39.8 29.  36.6 31.1 39.4 23.2
 22.2 34.1 36.  31.6 24.8 19.9 27.6 24.  33.2 32.9 38.2 37.1 34.  40.2
 22.7 45.4 27.4 42.  29.7 28.  39.1 19.4 24.2 24.4 33.7 34.7 23.  37.7
 46.8 40.5 41.5 25.  25.4 32.8 32.5 42.7 19.6 28.9 28.6 43.4 35.1 32.
 24.7 32.6 43.2 22.4 29.3 24.6 48.8 32.4 38.5 26.5 19.1 46.7 23.8 33.9
 20.4 28.7 49.7 39.  26.1 22.5 39.6 29.5 34.3 37.4 33.3 31.2 28.2 53.2
 34.2 26.8 55.  42.9 34.5 27.9 38.3 21.1 33.8 30.8 36.9 39.5 27.3 21.9
 40.6 47.9 50.  25.2 40.9 37.2 44.2 29.9 31.9 28.4 43.5 32.7 67.1 45.
 34.9 27.7 35.9 22.6 33.1 30.4 52.3 24.3 22.9 34.8 30.9 40.1 23.9 37.5
 35.5 42.8 42.6 41.8 35.8 37.8 28.8 23.6 35.7 36.7 45.2 44.  46.2 35.
 43.6 44.1 18.4 29.2 25.9 32.1 36.3 40.  25.1 27.5 45.6 27.8 24.9 25.3
 37.9 27.  26.  38.7 20.8 36.1 30.7 32.3 52.9 21.  39.7 25.5 26.2 19.3
 38.1 23.5 45.5 23.1 39.9 36.8 21.8 41.  42.2 34.4 27.2 36.5 29.8 39.2
 38.4 36.2 48.3 20.  22.3 45.7 23.7 22.1 42.1 42.4 18.2 26.4 45.3 37.
 24.5 32.2 59.4 21.2 26.7 30.2 46.1 41.3 38.8 35.2 42.3 40.7 46.5 33.5
 37.3 30.3 26.3 21.7 36.4 28.5 26.9 38.6 31.3 19.5 20.1 40.8 23.4 28.3
 38.9 57.3 35.6 49.6 44.6 24.1 44.5 41.2 49.3 46.3]
DiabetesPedigreeFunction [0.627 0.351 0.672 0.167 2.288 0.201 0.248 0.134 0.15
8 0.232 0.191 0.537
 1.441 0.398 0.587 0.484 0.551 0.254 0.183 0.529 0.704 0.388 0.451 0.263
 0.205 0.257 0.487 0.245 0.337 0.546 0.851 0.267 0.188 0.512 0.966 0.42
 0.665 0.503 1.39  0.271 0.696 0.235 0.721 0.294 1.893 0.564 0.586 0.344
 0.305 0.491 0.526 0.342 0.467 0.718 0.962 1.781 0.173 0.304 0.27  0.699
 0.258 0.203 0.855 0.845 0.334 0.189 0.867 0.411 0.583 0.231 0.396 0.14
 0.391 0.37  0.307 0.102 0.767 0.237 0.227 0.698 0.178 0.324 0.153 0.165
 0.443 0.261 0.277 0.761 0.255 0.13  0.323 0.356 0.325 1.222 0.179 0.262
 0.283 0.93  0.801 0.207 0.287 0.336 0.247 0.199 0.543 0.192 0.588 0.539
 0.22  0.654 0.223 0.759 0.26  0.404 0.186 0.278 0.496 0.452 0.403 0.741
 0.361 1.114 0.457 0.647 0.088 0.597 0.532 0.703 0.159 0.268 0.286 0.318
 0.272 0.572 0.096 1.4   0.218 0.085 0.399 0.432 1.189 0.687 0.137 0.637
 0.833 0.229 0.817 0.204 0.368 0.743 0.722 0.256 0.709 0.471 0.495 0.18
 0.542 0.773 0.678 0.719 0.382 0.319 0.19  0.956 0.084 0.725 0.299 0.244
 0.745 0.615 1.321 0.64  0.142 0.374 0.383 0.578 0.136 0.395 0.187 0.905
 0.15  0.874 0.236 0.787 0.407 0.605 0.151 0.289 0.355 0.29  0.375 0.164
 0.431 0.742 0.514 0.464 1.224 1.072 0.805 0.209 0.666 0.101 0.198 0.652
 2.329 0.089 0.645 0.238 0.394 0.293 0.479 0.686 0.831 0.582 0.446 0.402
 1.318 0.329 1.213 0.427 0.282 0.143 0.38  0.284 0.249 0.926 0.557 0.092
 0.655 1.353 0.612 0.2   0.226 0.997 0.933 1.101 0.078 0.24  1.136 0.128
 0.422 0.251 0.677 0.296 0.454 0.744 0.881 0.28  0.259 0.619 0.808 0.34
 0.434 0.757 0.613 0.692 0.52  0.412 0.84  0.839 0.156 0.215 0.326 1.391
 0.875 0.313 0.433 0.626 1.127 0.315 0.345 0.129 0.527 0.197 0.731 0.148
 0.123 0.127 0.122 1.476 0.166 0.932 0.343 0.893 0.331 0.472 0.673 0.389
 0.485 0.349 0.279 0.346 0.252 0.243 0.58  0.559 0.302 0.569 0.378 0.385
 0.499 0.306 0.234 2.137 1.731 0.545 0.225 0.816 0.528 0.509 1.021 0.821
 0.947 1.268 0.221 0.66  0.239 0.949 0.444 0.463 0.803 1.6   0.944 0.196
 0.241 0.161 0.135 0.376 1.191 0.702 0.674 1.076 0.534 1.095 0.554 0.624
 0.219 0.507 0.561 0.421 0.516 0.264 0.328 0.233 0.108 1.138 0.147 0.727
 0.435 0.497 0.23  0.955 2.42  0.658 0.33  0.51  0.285 0.415 0.381 0.832
 0.498 0.212 0.364 1.001 0.46  0.733 0.416 0.705 1.022 0.269 0.6   0.571
 0.607 0.17  0.21  0.126 0.711 0.466 0.162 0.419 0.63  0.365 0.536 1.159
 0.629 0.292 0.145 1.144 0.174 0.547 0.163 0.738 0.314 0.968 0.409 0.297
 0.525 0.154 0.771 0.107 0.493 0.717 0.917 0.501 1.251 0.735 0.804 0.661
 0.549 0.825 0.423 1.034 0.16  0.341 0.68  0.591 0.3   0.121 0.502 0.401
 0.601 0.748 0.338 0.43  0.892 0.813 0.693 0.575 0.371 0.206 0.417 1.154
 0.925 0.175 1.699 0.682 0.194 0.4   0.1   1.258 0.482 0.138 0.593 0.878
 0.157 1.282 0.141 0.246 1.698 1.461 0.347 0.362 0.393 0.144 0.732 0.115
 0.465 0.649 0.871 0.149 0.695 0.303 0.61  0.73  0.447 0.455 0.133 0.155
 1.162 1.292 0.182 1.394 0.217 0.631 0.88  0.614 0.332 0.366 0.181 0.828
 0.335 0.856 0.886 0.439 0.253 0.598 0.904 0.483 0.565 0.118 0.177 0.176
 0.295 0.441 0.352 0.826 0.97  0.595 0.317 0.265 0.646 0.426 0.56  0.515
 0.453 0.785 0.734 1.174 0.488 0.358 1.096 0.408 1.182 0.222 1.057 0.766
 0.171]
Age [50 31 32 21 33 30 26 29 53 54 34 57 59 51 27 41 43 22 38 60 28 45 35 46
 56 37 48 40 25 24 58 42 44 39 36 23 61 69 62 55 65 47 52 66 49 63 67 72
 81 64 70 68]
Outcome [1 0]
pca1 [-7.57146549e+01 -8.23582676e+01 -7.46306434e+01  1.10774227e+01
  8.97437881e+01 -8.09779219e+01  4.35095929e+00 -8.21372945e+01
  4.68866389e+02 -8.01938498e+01 -8.11049704e+01 -7.57421732e+01
 -7.87163720e+01  7.67474062e+02  9.87573954e+01 -8.36831663e+01
  1.50939440e+02 -8.18097627e+01  2.02476195e+00  1.61484669e+01
  1.56304525e+02 -8.24197420e+01 -7.27396749e+01 -7.84182971e+01
  6.90511626e+01  3.56531715e+01 -7.77705829e+01  5.69849149e+01
  3.22238927e+01 -8.04997885e+01 -8.00278807e+01  1.68757335e+02
 -2.96564375e+01 -8.31104857e+01 -7.84786143e+01  1.10176799e+02
 -7.87172357e+01 -8.00296953e+01 -8.08723425e+01  1.27001444e+02
 -3.70620441e+00 -7.89783677e+01 -8.06815644e+01  1.64969880e+02
 -7.69473762e+01 -7.22185960e+01 -7.82639584e+01 -8.37349685e+01
 -8.02523884e+01 -8.35860109e+01 -9.96075937e-02 -4.61650826e+01
 -5.97446879e+01  2.25154994e+02  2.64495572e+02 -8.49587192e+01
  2.30271653e+02  3.08739715e+01 -7.77450180e+01  6.15593678e+01
 -8.56192515e+01 -7.92831433e+01 -8.82430596e+01  5.04187131e+01
 -8.12482015e+01 -8.09597799e+01 -7.95359829e+01 -8.12124859e+01
 -4.46907431e+01  2.31439090e+01  8.03928956e+00  6.23292202e+01
 -7.95516708e+01  1.90001900e+02 -8.26317777e+01 -9.15033879e+01
 -8.61404529e+01 -8.08788647e+01 -8.04306389e+01 -8.01421613e+01
 -8.09766171e+01 -8.66010405e+01 -1.20240760e+01 -8.08641105e+01
 -7.80766036e+01  4.44483080e+01 -7.86229015e+01 -1.04085486e+01
  3.21948834e+01 -8.07583415e+01 -8.48557765e+01  9.55797382e+01
 -3.39885271e+01 -7.93795988e+01 -1.36275231e+01  1.49876969e+02
 -8.17058487e+01 -9.26400335e+00 -1.81655786e+01  1.41791662e+02
 -7.62272778e+01 -7.77362208e+01 -7.97447798e+01 -4.35719287e+01
 -8.39416158e+01  7.25092367e+01 -8.21877371e+01  6.22717757e+01
 -6.45599065e+01 -4.53871844e+01  6.05777774e+01  4.15867928e+02
 -4.46643239e+01 -8.49458632e+01  9.87752727e+01 -7.77837429e+01
 -8.01062463e+01 -8.49816265e+01 -8.15971430e+01 -3.10480390e+01
  2.66819861e+01 -7.91185082e+01  1.94187729e+01 -7.94385985e+01
 -8.10961839e+01  1.68794364e+01  5.54881820e+01  1.46408282e+01
  6.49016695e+01 -8.19643668e+01  9.22750389e+01 -8.05911350e+01
  1.49969733e+02 -8.20845083e+01 -3.36209283e+01  6.00450599e+01
 -3.12568029e+01  9.51446199e+00 -7.95174094e+01  2.42602362e+02
 -7.98878985e+01 -7.98383148e+01 -1.78374840e+01 -8.18467331e+01
  2.06600610e+02 -8.12500730e+01 -8.43507271e+01  3.82508096e+01
 -7.78926602e+01 -8.24941921e+01  1.26805665e+02 -8.13930299e+01
  7.88325865e+01  4.07223854e+02 -7.36111165e+01 -7.42416035e+01
  1.12933752e+01  5.34213519e+01 -2.98451593e+01  3.94225308e+01
 -7.49501463e+01  2.44646343e+01  2.04306717e+02 -8.12107519e+01
 -7.91981102e+01  7.38865435e+01 -7.63285131e+01 -8.06391550e+01
 -8.15995361e+01 -2.97076848e+00 -8.21691049e+01  5.13591598e+01
 -8.35375076e+01 -3.43170617e+01 -2.90169640e+01  5.68805130e+01
 -8.39119495e+01  5.33780754e+01 -7.80614340e+01 -7.93152172e+01
 -8.37607394e+01  1.17800443e+01 -6.81977366e+01 -8.53815792e+01
 -7.85382135e+01 -7.16426422e+01  4.19010644e+02 -1.92388331e+01
  3.39765607e+01  8.24944962e+01 -8.16660343e+01  1.57295399e+01
 -7.68588509e+01 -7.99838746e+01 -8.31664051e+01  1.34532414e+02
 -8.22893899e+01 -3.36007615e+01  1.93132926e+01  2.39456241e+02
 -8.00844800e+01 -7.84789308e+01 -8.06147266e+01 -3.80921852e+01
  1.08432072e+02 -7.98109922e+01  2.06776803e+02 -7.59047183e+01
  5.09268916e+00 -7.20652996e+01 -8.32366232e+01 -7.42153018e+01
 -7.25848941e+01  5.21786066e+01  9.45892369e+01  1.94129013e+02
  4.89075055e+01  4.08544634e+01 -8.26323769e+01 -8.13661607e+01
  4.01400965e+02 -7.66502231e+01 -8.19099697e+01  1.11966452e+02
 -2.61515198e+01 -5.01674593e+01 -8.22470341e+01 -7.43356828e+01
  6.68279542e+02 -2.59591711e+01 -7.79531272e+01  2.90807839e+02
 -4.62023655e+01 -8.03502190e+01 -3.87049008e+01 -7.53618467e+01
  1.17559548e+02 -7.26233508e+01 -7.47881673e+01 -8.21993448e+01
 -8.20353775e+01  5.98613706e+00 -7.90174472e+01  9.50623354e+01
  1.16964834e+02 -7.32519869e+01 -8.04584406e+01  6.01801951e+02
  3.21090732e+02 -8.00234284e+01 -8.22823921e+01 -7.94950109e+01
 -2.79421080e+01 -8.18937509e+01  1.73093730e+02 -7.91533632e+01
 -7.93198606e+01 -7.98661923e+01  2.99589932e+02  7.35582413e+01
  5.63412425e+01 -7.96468321e+01 -8.10374220e+01 -7.74452324e+01
 -8.04124768e+01 -1.52359342e+01 -7.98387116e+01 -7.71836398e+01
 -8.26662110e+01 -7.91946241e+01 -7.97618899e+01 -2.43626055e+01
 -8.03946838e+01 -3.74404778e+01 -8.19944886e+01 -2.26077078e+01
 -8.07109980e+01  3.43516920e+01 -8.12793091e+01  1.94769596e+02
 -7.79183041e+01  4.32886871e+01  7.58200182e+01 -7.63806960e+01
 -8.17165690e+01  5.65548881e+01  4.67085871e+02  1.40654720e+02
 -3.35794298e+01 -4.56029729e+00 -4.27824427e+01 -6.55608958e+00
  1.03505402e+02  1.15413410e+02 -7.71291029e+01  4.34532037e+01
  2.81764040e+02  1.35545023e+02  1.01851778e+02 -8.15349108e+01
 -7.70760933e+01  5.76863206e+01 -4.03370871e+01 -8.02998714e+01
 -7.77054984e+01  2.61773364e+01  5.57430325e+01  6.88415131e+01
  1.00110812e+02  1.25105629e+02 -8.28379498e+01  6.74401853e+01
  1.92391292e+01  3.44113643e+00 -7.95881639e+01  1.33628173e+01
 -1.84009228e+01 -7.44460193e+01  6.03868317e+01 -7.34191075e+01
  1.50231264e+02 -7.94405377e+01 -7.90306995e+01 -4.64394494e+01
 -7.92292325e+01  9.11481124e+01  7.64589761e+01 -7.47821247e+01
  3.95161684e+01 -1.25966313e+01 -7.97575512e+01 -3.13347286e+01
 -7.56919570e+01 -8.19515942e+01 -2.45504001e+01  1.80039674e+02
 -8.19955124e+01 -8.10149170e+01  9.45974320e+01 -7.45955050e+01
  2.54344973e+01 -9.26782764e+00 -9.01677639e+01 -8.00810839e+01
 -8.30093909e+01  2.97757319e+01  4.48399786e+00 -8.21692411e+01
 -8.22415251e+00 -9.00924014e+01 -8.29906562e+01 -7.86824386e+01
 -8.44573017e+01 -4.02291966e+01 -8.31807473e+01 -7.59683630e+01
  8.79194360e+01 -7.89400849e+01 -2.76007109e+01  1.76515763e+02
  2.50849618e+02 -7.69929959e+01 -7.93884745e+01 -7.79281654e+01
  2.14687052e+02  1.28465838e+00 -8.01979611e+01 -8.15953932e+01
 -1.76315494e+01  6.19276971e+01  3.89641191e+02  8.52537630e+00
 -1.72015735e+01  1.36255511e+01  7.89409910e+01  2.46948620e+02
  1.75595038e+00 -7.10915383e+00 -7.67364083e+01 -8.69355939e+00
  1.47004234e+00 -8.08535754e+01  9.93820897e+01 -2.40075008e+01
  3.05460708e+01 -3.05777271e+01 -7.91396828e+01 -7.92597140e+01
  2.06495802e+02 -7.04443648e-01  1.13795128e+02 -7.57627708e+01
  3.33333565e+02  6.04778708e+00 -7.67044042e+01  1.94458222e+02
  3.30093962e+01 -7.70487071e+01 -8.44089681e+01 -7.14751562e+01
 -8.31047908e+01 -7.92438708e+01  1.11379420e+01 -8.36508397e+01
 -7.59896192e+01  8.53475170e+01 -8.11166041e+01 -8.26620471e+01
 -7.30885972e+01  5.02642091e+02 -7.95734135e+01  9.53582363e+01
  2.31358143e+02 -1.63719950e+01  8.90930927e+01  3.97671880e+02
 -8.15975072e+01 -7.60082418e+01 -8.43869337e+01  3.61247803e+01
  9.11592064e+01 -6.61234473e+00 -1.99824900e+00 -7.97888945e+01
  1.33222449e+02  2.03357066e+02 -8.46513769e+01  1.05900282e+02
  6.82328135e+01  9.74691126e+01 -8.38482726e+01  1.81065968e+00
 -2.41457368e+01 -7.87025508e+01 -8.31597377e+01 -7.94744609e+01
 -7.63364638e+01 -7.78621689e+01 -8.20529030e+01 -8.14813063e+01
 -7.17430518e+01 -3.31542888e+01  3.99025989e+01 -8.17724883e+01
 -8.02845845e+01 -5.64196959e+01 -1.23169952e+01  1.13419967e+01
 -1.63546207e+01 -1.68340062e+01  1.06424287e+01 -7.92042484e+01
  1.27206105e+02 -8.21027528e+01  2.33116028e+01 -7.35120764e+01
 -7.95316748e+01 -1.17108447e+01  1.60680161e+02 -1.78112389e+01
 -2.38072259e+01 -8.56033390e+01 -3.39596540e+01 -8.18264743e+01
 -8.07358444e+01  2.45704566e+01 -4.90110109e+01  1.86861206e+01
 -8.17505067e+01  6.46259252e+01 -7.54447816e+01 -7.65344047e+01
 -7.89492005e+01 -7.87792887e+01 -8.12648700e+01 -7.71919521e+01
  1.10584323e+02  2.91010121e+01 -3.24170451e+00 -7.74182346e+01
  2.50753853e+02 -7.76717656e+01 -3.42284597e+01  4.22547802e+01
 -7.90648964e+01  1.71878997e+02  4.00514770e+02  1.89983934e+02
 -8.16308400e+01 -7.33828281e+01 -1.69194338e+01 -8.14646225e+01
 -8.03363477e+01  4.20864462e+01 -8.60161541e+01 -7.62036295e+01
 -8.16556965e+01 -7.95861526e+00  7.25906755e+01  1.16371603e+02
 -8.92477076e+00 -8.20221307e+01 -8.97828055e+01 -3.25111733e+00
 -8.03679281e+01 -8.47901624e+01  1.66442212e+01  9.04748506e+01
 -7.51448578e+00 -8.06578648e+01 -8.22703136e+01  1.30657926e+02
 -8.36472852e+01 -8.35577523e+01  3.62473662e+00  2.90203828e+01
  8.79890139e+01 -7.98152193e+01 -8.50702075e+01  2.44537725e+02
 -1.82322451e+01  5.11353498e+01 -8.27108767e+01 -7.96137551e+01
 -8.02456365e+01 -8.29631243e+01 -4.71263730e-01  2.42452560e+01
  1.07691081e+02 -8.16190374e+01  2.58407205e+01 -8.15182693e+01
 -1.60974550e+01 -8.45606770e+01 -2.75074177e+01 -8.04745143e+01
 -8.17872465e+01 -8.71502497e+01  1.31181140e+02  7.76412047e+01
  1.33547381e+02  1.10483807e+02 -8.13719419e+01 -2.66242194e+01
 -6.31249676e+00  1.51796598e+02  1.33383914e+02  8.66391942e+01
 -6.96551261e+00 -7.13569126e+01 -7.92757161e+01  2.29983442e+01
 -8.10477464e+01 -3.86613196e+01  3.15100117e+01  1.35233581e+02
 -8.02872153e+01 -8.16076403e+01 -7.96820884e+01 -8.39769333e+01
 -8.00381616e+01  2.01207838e+02 -5.17804098e+00 -2.80744370e+01
 -8.32103616e+01  4.93240431e+00 -6.27517854e+01  4.35882131e+01
  4.96576459e+01  8.51896911e+01 -8.46904616e+01 -7.92583612e+01
 -3.60568076e+01  3.74044119e+01  2.51501846e+02 -1.55810711e+01
  4.80344033e+01 -8.04049412e+01 -7.94235420e+01 -6.70849518e+01
 -7.42287592e+01 -8.02472065e+01 -7.95072115e+01 -8.23760276e+01
  5.17120628e+02 -8.28556520e+01 -7.83807694e+01 -8.23948621e+01
  8.16836257e+01 -8.64097458e+01 -7.86439656e+01  6.09893454e+01
 -7.92388382e+01  3.09521565e+01  1.50920209e+02  1.10897910e+02
 -8.55094210e+01 -5.83701530e+01 -7.52662326e+01  3.80123985e+01
 -8.03301362e+01 -8.41465579e+01 -7.79693320e+01  4.93511242e+01
 -7.55955625e+01 -7.82932699e+01  2.19216389e+02 -4.13414588e+01
  1.95411311e+02  9.98865032e+01  7.60092432e+01  1.18522956e+02
  2.45811104e+02 -8.01468283e+01  6.58254473e+01 -8.19706806e+01
 -8.04846751e+01 -7.02255055e+01 -7.98246365e+01 -8.17486343e+01
  8.04660215e+01 -8.20836961e+01 -7.39445371e+01  3.29470922e+01
 -8.18103639e+01 -2.66369666e+01 -8.01645916e+01 -7.92109963e+01
 -7.96350292e+01 -8.19165007e+01 -8.13272734e+01  9.68354496e+00
 -8.16527812e+01  1.03159889e+02 -8.35280050e+01 -8.21499367e+01
 -8.21710174e+01 -1.63433305e+01  9.71669722e+00 -3.62010054e+01
  2.32282433e+01 -7.97231961e+01 -7.78678916e+01 -8.46796060e+01
  7.05369339e+01  3.62326986e+02  6.80179948e+01  8.50995341e+01
  5.23530872e+01 -8.05183775e+01  1.71123829e+01  2.56955524e+01
 -1.26718058e+00 -8.08748277e+01  5.39078075e+01  4.60610420e+02
  8.78519305e+00  1.21204107e+02 -7.92862683e+01 -1.28333612e+01
 -7.63839091e+01 -6.99352046e+01  1.56844562e+02  5.39832220e+01
 -7.88556835e+01  5.27104559e+01 -7.69628966e+01 -7.99100894e+01
  1.07769462e+02  2.39497543e+01  9.21831482e+01 -8.22530851e+01
 -3.50300222e+01  1.61075334e+02 -8.32956771e+01 -7.32464406e+01
 -7.69736115e+01 -8.32890202e+01 -8.06753712e+01  1.81516278e+02
 -4.07300777e+01 -7.38984478e+01  2.37581746e+01 -7.98942911e+01
 -7.93904119e+01  1.25591606e+02 -7.97718952e+01 -8.10406120e+01
  1.01721543e+02  1.03682651e+02 -8.17956510e+01 -7.60581837e+01
  1.59570403e+01  4.74434854e+01 -8.37518647e+01  3.99990861e+02
  4.95600854e+01 -8.38013562e+01  7.50578732e+01 -8.05652083e+01
  1.20068520e+02 -7.81913230e+01 -7.38427309e+01 -8.07496093e+01
  1.90455145e+01 -8.20112914e+01 -8.26353725e+01  2.53773288e+02
 -7.61793029e+01  7.76584369e+01  3.08196910e+02 -5.64260412e+01
 -7.75321766e+01  2.10789068e+02 -8.22957708e+01  3.16983211e+02
  1.10872507e+02 -8.22059499e+01  9.77511180e+01 -8.11019465e+01
 -8.28185452e+01  1.19722472e+02  5.00575525e+01  2.55587413e+01
 -8.10895907e+01 -7.87076568e+01  9.97815288e+01 -7.66580033e+01
 -7.49849150e+01 -8.35818472e+01  3.38053142e-01 -8.03318449e+01
  4.66342479e+01  8.33403480e+01 -8.21457121e+01 -8.11665520e+01
  4.10746219e+01 -8.45600534e+01  7.72569795e+01 -8.21871742e+01
  7.08825049e+01  1.18287431e+01  3.44516325e+01 -7.82703783e+01
  6.43123526e+01  2.38902460e+01 -7.47819561e+01 -2.50115228e+01
  1.25995227e+02 -7.67632491e+01 -7.89765354e+01 -4.36178503e+00
 -8.04678873e+01  4.35037328e+02 -7.52166288e+01  3.21691348e+01
 -7.61217209e+01 -8.02437963e+01 -8.17347791e+01 -7.34417946e+01
 -6.64305705e+01 -7.36091609e+01 -8.38741976e+01  9.92378807e+01
 -7.86412389e+01  3.21131983e+01 -8.02144943e+01 -8.13081497e+01]
pca2 [-3.59507826e+01  2.89082132e+01 -6.79064965e+01  3.48984859e+01
 -2.74693708e+00 -3.94688717e+00  4.69339122e+01  7.09795392e+00
 -3.15282576e+01 -1.80482829e+01 -1.20901354e+00 -5.57584658e+01
 -3.11284317e+01  4.26768414e+00 -3.75677733e+01  2.15772175e+01
  1.61971638e+01  4.42960205e+00  2.38272891e+01  7.80020114e+00
  8.77573621e+00  7.82979202e+00 -8.62486079e+01 -5.80000795e+00
 -2.09059906e+01 -1.65612156e+00 -3.68849740e+01  3.11004796e+01
 -2.55740088e+01 -8.99419199e+00 -4.72774188e-01 -1.97537099e+01
  3.24960284e+01  1.73572299e+01 -1.08640463e+01  3.04330035e+01
 -2.68132344e+01  8.84454095e+00  2.45085566e+01  1.93423116e+01
 -5.64674910e+01 -2.36062360e+01  1.83022058e+00 -4.28132157e+01
 -4.58644325e+01 -6.27341113e+01 -3.05209424e+01  4.30087042e+01
  1.08589671e+01  1.92468483e+01  1.77391888e+01  1.90333353e+01
  2.78580258e+01 -3.85680956e+01 -3.22943606e+00  4.33167837e+01
 -4.32547966e+01  2.24789946e+01 -3.67987610e+01  2.44635694e+01
  4.01970089e+01 -2.18630586e+01  6.69420071e+01 -1.09841389e+01
 -2.93276038e+00  1.37028971e+01  1.27494784e+00 -3.79563844e+00
  2.30259774e+01 -2.33611098e+01  2.24222160e+01 -9.19615430e+00
 -1.86391771e+01  8.97546476e+00  3.45299738e+01  1.14930616e+02
  4.60687039e+01  1.85147643e+01 -8.18161092e+00  3.28723292e+00
  5.29622854e+00  4.97791062e+01  3.47307310e+01  1.48151020e+01
 -3.12591868e+01  1.55613211e+01  6.38315351e+00  2.06475288e+01
 -1.31517728e+01  7.64318546e+00  3.54063385e+01  5.07955773e+00
  3.36508721e+01 -2.52607565e+01 -2.18907296e+01 -9.56205485e+00
  2.33363628e+01  5.31814431e+01  3.04245056e+01  1.05363877e+01
 -5.03171213e+01 -3.47990355e+01 -1.42984583e+01  3.60754217e+01
  2.72990421e+01  6.15262294e+00  9.34707450e+00 -1.51721543e+01
  3.42409430e+01  2.01615535e+01 -4.19259723e+01  5.47895372e+00
  2.82961743e+01  3.69731880e+01 -2.70119994e+01 -4.02534458e+01
 -1.32405015e+01  3.70067267e+01  1.88643150e+01  1.93876279e+01
 -3.67322808e+01  5.00419807e+00  1.66746907e+01 -2.58766182e+01
 -5.81408822e-01  4.05872505e+01  6.47481767e+00  8.06719582e+00
  6.47440693e+00  8.47713686e-01 -4.26784530e+01 -8.09931722e+00
 -3.19681444e+01  2.70313135e+01  2.25773101e+01  3.23761164e+00
  1.95420187e+01  3.16415700e+01 -1.74471198e+01  3.92124753e+01
 -1.94286281e+01  4.60144344e+00  1.52837612e+01  2.85133096e+00
 -9.85467953e+00  1.34299211e+01  5.27323584e+01  1.97561613e+01
 -3.99394171e+01  2.39913890e+01 -8.17007677e-01 -1.51020016e+00
 -3.04330281e+01  7.63697491e+00 -7.74512255e+01 -4.04198332e+01
  2.68415970e+01  2.04926910e+01  3.04191376e+01 -3.95710127e+01
 -3.90570774e+01  1.92216053e+01  2.57420533e+01  1.54951153e+01
 -2.09661409e+01  2.17559954e+01 -3.16333123e+01 -7.71060203e+00
  2.88831450e+00  6.77253479e+00  7.42087846e+00 -7.58984130e+00
  3.66926178e+01  4.12144448e+01  4.33806760e+01 -5.14235364e+01
  2.36561232e+01 -8.00681386e+00 -3.40583214e+01 -2.03090315e+01
  2.32055935e+01  4.95988308e+00  1.13416507e+02  4.02089574e+01
 -2.97251522e+01 -7.93791070e+01 -2.18505673e+01 -1.17900141e+01
  1.53022933e+01 -9.57487967e+00  4.06485352e+00 -5.88622461e-01
 -4.57285701e+01 -1.47096490e+01  2.83515866e+01 -2.44630941e+01
  1.04311580e+01  1.29531079e+01  1.63422187e+01 -1.50342837e+00
  1.64624768e-01 -2.67809875e+01  5.59205390e+00  1.89703617e+01
  2.47109735e+01  3.31829319e+00 -5.82458488e+01 -5.65778611e+01
  2.77062314e+01 -7.16191407e+01  3.40102188e+01 -3.23757055e+01
 -7.10367907e+01 -1.20413510e+01  1.58822959e+01 -1.14896009e+01
  1.93621194e+01  7.35844338e-01  2.70236963e+01 -1.03774168e+00
 -1.34041381e+01 -5.23106972e+01  2.63008077e+00 -1.18903396e+01
  1.96555993e+01  2.90591464e+01  1.05519373e+01 -4.30067560e+01
 -1.01100650e+01  1.07204167e+00 -3.06479313e+01  1.16216130e+01
  3.73421762e+01 -9.20757798e+00  4.44495235e+01 -5.73970398e+01
 -5.26491832e+01 -6.56793838e+01 -5.14326775e+01  8.30745786e+00
  2.43491462e+01  3.18962139e+01 -2.23030524e+01  1.39550314e+01
 -1.33981267e+01 -7.37115492e+01 -1.08468811e+01  1.22399088e+01
  2.76807586e+01  1.17106079e+00  6.86140991e+00 -1.76290627e+01
  2.74589306e+01  2.82573678e+01  4.36923475e+01  3.39003372e+00
  5.55875690e+00  7.45201601e-01 -3.80733017e+01 -2.97748306e+01
 -6.36062041e+01 -1.74921839e+01  1.94467260e+01 -3.41075498e+01
 -1.00282850e+01  2.06801881e+01 -1.45233410e+01 -1.15323229e+01
  1.41844037e+01 -2.23528954e+01  8.89941273e+00  1.40326760e+01
 -1.15820823e+01  4.73811360e+01  2.67395104e+00  2.06471301e+01
  9.20313627e+00  2.24371111e+01 -5.75457042e+00  3.36213686e+01
 -3.27298465e+01 -5.56350311e+00 -8.27207039e+00 -5.20528487e+01
```

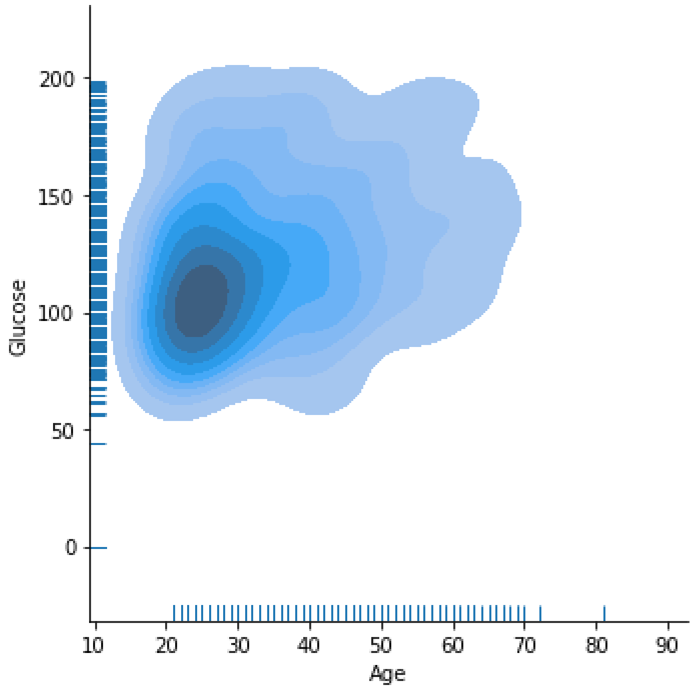
```
-1.02225696e-01 -1.21200695e+01  9.65891924e+00  1.37988486e+01
 2.44807128e+01  1.27858854e+01  3.73171871e+01  1.55410629e+01
 2.27740843e+00  9.26894995e+00 -4.89094002e+01 -2.33301641e+01
 3.81406228e+00  7.64907116e+00  2.69352781e+01 -3.67346006e+00
-4.32303044e+01 -1.37274180e+01  3.80007426e+01 -7.38253349e+00
-3.80056559e+01  3.50291542e+00 -3.55148543e+01 -7.28464733e+00
 3.95316645e+00  9.86997477e+00  3.23310189e+01  2.30745081e+01
-3.13539827e+01  1.15600473e+01  1.44877055e+00  1.11905263e+01
 1.87629717e+01 -6.81575691e+01  1.34034452e+01 -8.43357317e+01
 7.74718203e+00  2.14210118e+00 -1.08120601e+01 -3.86235733e+01
 2.50668722e+00 -2.54796368e+01  7.89622799e+00 -6.61691466e+01
 2.15137794e+01  1.43532215e+01 -6.70236720e+00  3.27994328e+01
-5.79497816e+01  2.89685874e+00  2.62778557e+01 -2.56247759e+01
 3.34730764e+00 -5.48401498e+00 -2.22812647e+01 -6.79019705e+01
-5.10456097e+00  2.38274720e+01  1.12617245e+02 -1.23310907e+01
 1.23745255e+01 -6.69023701e+00 -1.23149188e+01  7.67953300e+00
 2.31697427e+01  1.08124909e+02  1.79177341e+01 -2.60228328e+01
 4.73987219e+01  2.86529095e+01  2.12662176e+01 -5.65512535e+01
 8.83478104e+00 -7.16404293e+00  2.75619422e+01 -5.66442423e+01
-4.09592161e+01 -4.90349129e+01  4.62996740e-01 -3.94506115e+01
-5.32759121e+00  2.44981903e+01 -1.14173119e+01  1.46297938e+01
 3.66260062e+01 -1.19302207e+01 -1.35760689e+01  7.85639194e+00
 3.69965941e+01  2.05592596e+01  1.06307854e+01  1.04164244e-01
 2.24874821e+01  3.61386138e+01 -4.43012024e+01  2.23696350e+01
 1.52688934e+01  1.03694416e+01  2.38911578e+01  3.04720185e+01
 5.20956094e-01  2.54481518e+00 -3.28483930e+00  2.13887150e+00
-8.05657885e+00  2.14986906e+01  3.10763141e+01 -5.33649737e+01
 2.44103973e+01  4.46945431e+00 -4.58082599e+01  1.57093687e+01
 2.97264940e+01 -1.42543753e+01  3.06332550e+01 -7.61364601e+01
 1.74705033e+01 -2.59795271e+01 -1.52531685e+01  3.82154107e+01
-5.49674148e+01  1.02555609e+01 -5.06605424e+00  1.33269505e+01
-8.40402613e+01 -3.33750481e-01  9.43297550e+00  1.89937206e+01
-5.28483169e-01 -2.18274973e+01 -4.56923926e+00 -1.33273030e+01
 1.77277045e+01 -3.20172440e+01  2.98458548e+01 -2.31843121e+00
 9.33426056e+00  2.78367018e+01  2.17955787e+01  8.02389627e-01
-1.85405492e+01 -4.28096791e+01  2.99595728e+01 -4.83829219e+01
-8.90531198e+00  3.16404273e+01  2.42908820e+01  2.99700324e+01
 3.83717456e+01 -2.62414848e+01  2.19454130e+01 -1.82865844e+01
-2.91862107e+01 -3.41491455e+01  1.76604163e+01  2.12692853e+00
-7.95664249e+01  3.62124959e+01  9.92409840e+00  3.67919582e+00
-2.69193413e+00 -6.25413346e+01  1.95951487e+01  2.70892958e+01
 1.81942629e+01 -3.89058361e-01  4.15473614e+01 -1.99851960e+01
 4.30752168e+01 -1.90002134e+00  2.63382267e+01 -5.95695017e+01
-2.40275448e+01  3.50342865e+01 -1.48939308e+01 -2.11131141e+01
-3.36844868e+00  4.23350666e+01  4.26192287e+01  2.33680310e+01
-6.97993204e+00  3.07085251e+00  4.51243092e+01  2.77775410e+01
 1.26960461e+00 -2.64038416e+01 -3.03564596e+01 -2.07193043e+01
-3.55148375e+00 -2.87122898e+01  1.25334627e-01 -2.82516430e+01
 2.64006486e+01  9.00516424e+00 -5.71677020e+00 -2.41272685e+01
-1.25593651e+01 -1.07957384e+01  3.45713925e+01  4.01531375e+01
-2.25406594e+01  4.68910346e+00  2.38308696e+01 -3.77968389e+01
 1.41510565e+01 -8.59086680e+01  3.76612197e+01  1.97828464e+01
 1.48948600e+01 -1.77883231e+00  4.39231382e+01 -5.76731497e+01
 2.71896554e+00  3.89809601e+01 -6.80795291e+01 -2.22348339e+01
 1.97637451e+00  2.90799135e+01  1.09858887e+02  2.58335941e+01
 1.52802756e+01  3.31821174e+01 -5.95177728e+01  3.05704341e+00
 4.01954639e+01 -1.32090156e+01  2.66474576e+01 -2.21039739e+00
 1.72602782e+01  2.31704492e+01  2.55424788e+01 -3.80520712e+01
-2.00827172e+01 -1.79522789e+01  3.48670157e+01  8.45099701e+00
 5.21807435e+01  1.88376583e+00  1.02393652e+01 -1.95876447e+01
-9.82171841e+00  2.87605513e+01  2.65025827e+01  7.71034002e+00
 1.67897929e+01  2.21102132e+00  4.27010309e+00  4.54767426e+00
 3.50069232e+01  3.04993057e+01  4.41276662e+01 -8.34011644e+00
 2.15953888e+00  5.05100160e+01  7.11114428e+00 -2.38060969e+00
 3.17289351e+01  4.24212093e+00  1.74261628e+01  3.15937931e+01
 3.15069705e+01 -5.24233470e+01 -5.61776961e+01 -8.04738756e-01
-4.59713001e+01 -7.93283917e+01 -4.91573323e-01  4.02314539e+01
-9.16127924e+00  3.13997641e+01  4.06357275e+01  1.00754919e+01
 1.69106192e+01 -2.49347127e+00  9.07226459e+00  2.51434306e+01
-1.67523945e+01 -5.50976962e+01  3.47199707e+01  2.04123315e+01
 1.97209139e+01  2.96102785e+01  1.70449149e+01  3.22977238e+01
-2.82324906e+01  8.96065879e+00  3.20243459e+01 -1.92095085e+01
 1.01252539e+01  2.86403126e+01  1.92301201e+00  6.40107271e+00
 2.08979394e+01 -6.22633731e+00 -2.06430535e+01 -8.12451723e+01
-3.69835952e+01  6.83360879e+00 -1.30781768e+01  8.95935298e+00
 4.28586441e+01  2.29621687e+01 -3.11078701e+01  1.00056630e+01
-5.10592506e+01  4.93847836e+01 -1.42530598e+00  1.57758759e+01
-2.26842869e+01  4.50166237e+01  1.29398264e+01 -5.58317163e+01
 4.03527778e+01  3.41117619e+01 -6.09196144e+01  2.11325609e+01
 3.80441308e+00  2.63455492e+01 -9.65421708e+00 -2.78293959e+01
-5.95335630e+01 -7.01354068e+00 -3.70177081e+01  2.74930389e+01
-1.22998436e+01  2.17383130e+01  2.54167837e+01 -3.95729240e+01
-2.57225080e+01  7.72944577e+00 -1.36376303e+01  6.51543472e+00
-8.20933165e+00  4.79011358e+01 -2.82161825e+00  4.27265944e+00
 1.55564855e+01  2.06723870e+01 -7.47633365e+01  3.09875228e+01
 6.33512239e+00  2.67674862e+01 -1.04282919e+01 -1.90147001e+01
-1.89900966e+01  2.13183515e+01 -1.27432936e+00  2.04634409e+01
 3.92171781e+00  2.67440978e+00  2.08215890e+01  6.43765113e+00
 5.04633174e+00  2.52117033e+01  2.36590129e+01  1.73131485e+01
 1.92401345e+01 -1.45908606e+01 -3.79460427e+01  3.16012758e+01
 2.54661875e+01 -7.26828899e-01 -3.91464801e+01 -4.40003269e+01
-1.24283467e+01  9.29316943e+00  3.51962914e+01  8.55458173e+00
-1.27063291e+00 -4.56566456e+00  2.15455966e+01  1.40730092e+01
 2.48161934e+01  1.09387255e+01 -2.28911502e+01  3.83959172e+01
-5.36635508e+01 -8.20377619e+01 -3.60040838e+01 -2.00997379e+01
-5.36196926e-01  1.46844266e+01 -3.81429662e+01  1.43993415e+00
 3.34723474e+01 -3.27020376e+01 -3.63940202e+01  1.67910654e+01
 4.11844133e+01  9.97060257e+00  1.33587066e+01 -8.05957206e+01
-4.78151098e+01  2.07590712e+01 -5.42825194e+00  4.00810797e+01
 6.40733527e+01 -4.73229222e+01  3.04229854e+01 -1.34200727e+01
-2.87907721e+01  4.72872663e+00 -1.48547511e+01  9.41214808e+00
-7.60100535e+00 -1.57247617e+01  3.36936580e+00 -5.33841222e+01
 3.01523189e+00 -3.54424320e+00  2.41847435e+01  1.32216948e+01
-4.28432239e+01  2.43447052e+01 -1.43971343e+00 -5.62871179e+00
 1.05681993e+01 -1.42507969e+01 -5.82457827e+01 -8.16426611e+00
 1.25341498e+01  3.18749220e+01  8.61582808e+00  2.30232464e+01
-5.37118868e+01  3.72911474e+01 -5.36470551e+00 -1.21823461e+01
-1.47672572e+01  1.08545343e+01  9.24588554e+00 -3.15612413e+01
-4.06486385e+01  1.51265523e+01  2.57890998e+01  1.22499703e+01
 2.68205815e+01  2.09060338e+01 -2.32481336e+01  2.73344331e+00
-4.27420845e+00 -5.81777617e-02  1.44942418e+01 -2.72518491e+01
-6.19779931e+01  2.34861218e+01 -9.96614157e+00 -8.61681221e+00
-4.88049198e+01  2.65346451e+01  3.55733705e+00  2.01504036e+01
-1.16448667e+00  4.51995396e+01  3.15038815e+01  7.41963125e+00
 4.20384847e+00  2.43350202e+01  1.82230397e+01 -3.26451765e+01
-2.86648722e+01  1.94264626e+01 -3.51535525e+01  3.66836934e+01
-5.35110607e+01 -4.97305659e+01 -2.19433063e+01 -3.75157198e-01
 7.64971191e+00 -1.84792749e+01 -4.20970223e+01 -6.33789126e+00
-2.58691207e+01 -1.39416518e+01  5.58368769e+00 -8.39785969e+01
```

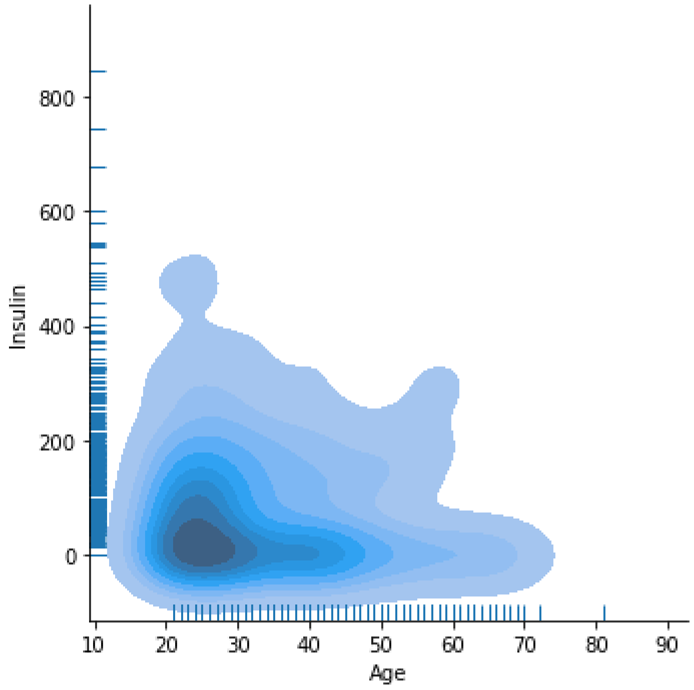
```
  2.96212748e+01 -5.74297735e+01  2.36402602e+01  2.50809271e+01
 -7.68801008e+00  3.37666480e+00 -1.41860198e+01  2.16214961e+01]
```

In [80]:
```
import seaborn
fig = seaborn.displot(data=diabetes, x='Age', y='BloodPressure', kind='kde', fi
fig = seaborn.displot(data=diabetes, x='Age', y='BMI', kind='kde',fill=True, ru
fig = seaborn.displot(data=diabetes, x='Age', y='Glucose', kind='kde',fill=Tru
fig = seaborn.displot(data=diabetes, x='Age', y='Insulin', kind='kde',fill=Tru
```









To remove the missing values from the diabetes dataset.

We need to find 3 decision models with the best F1 scores, based on these characteristics:

- The first model: find the best max_depth.
- The second model: find the best min_samples_leaf
- The third model: find the best combination of max_depth and min_samples_leaf

and evaluate models using f1_score and ShuffleSplit with 100 splits.

You can read about these two parameters here: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.Decisi

In [82]:
```
import numpy as np
diabetes = diabetes.replace({'Glucose': {0: np.nan},
                 'BloodPressure': {0: np.nan},
                 'SkinThickness': {0: np.nan},
                 'DiabetesPedigreeFunction' : {0:np.nan},
                 'Pregnancies' : {0:np.nan},
                 'Insulin': {0: np.nan},
                 'BMI': {0: np.nan}})


diabetes = diabetes.dropna()


diabetes = diabetes.reset_index(drop=True)

diabetes.to_csv('diabetes_cleaned.csv', index=False)
```

In [83]:
```
X = diabetes.drop('Outcome', axis=1)
y = diabetes['Outcome']
```

In [84]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import ShuffleSplit
import numpy as np


validator = ShuffleSplit(n_splits=100)
max_depth_range = range(1, X.shape[1]+1)
best_f1_score = 0
best_max_depth = None



for max_depth in max_depth_range:
    f1_scores = []
    for train_index, test_index in validator.split(X):

        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]


        diabetes = DecisionTreeClassifier(max_depth=max_depth)
        diabetes.fit(X_train, y_train)


        y_pred = diabetes.predict(X_test)
        f1_scores.append(f1_score(y_test, y_pred))


    mean_f1_score = np.mean(f1_scores)


    if mean_f1_score > best_f1_score:
        best_f1_score = mean_f1_score
        best_max_depth = max_depth
```

```
print('Best max_depth:', best_max_depth)
print('Best F1 score:', best_f1_score)
```

```
Best max_depth: 4
Best F1 score: 0.6272503355035418
```

In [85]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import ShuffleSplit
import numpy as np
min_samples_leaf_range = range(1, X.shape[1]+1)
best_min_samples_leaf = None
best_f1_score = 0


cv = ShuffleSplit(n_splits=10)


for min_samples_leaf in min_samples_leaf_range:


    clf = DecisionTreeClassifier(max_depth=3)


    f1_scores = []
    for train_index, test_index in cv.split(X):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        f1_scores.append(f1_score(y_test, y_pred))
    avg_f1_score = np.mean(f1_scores)


    if avg_f1_score > best_f1_score:
        best_min_samples_leaf = min_samples_leaf
        best_f1_score = avg_f1_score

print("Best value for min_samples_leaf:", best_min_samples_leaf)
print("Corresponding F1 score:", best_f1_score)
```

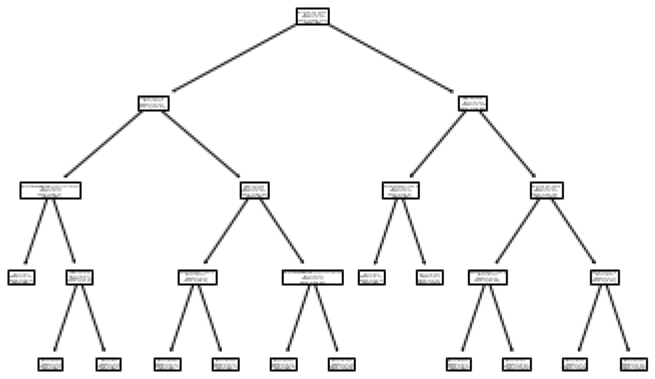
```
Best value for min_samples_leaf: 3
Corresponding F1 score: 0.6684133059922533
```

In [86]:
```
from sklearn.model_selection import cross_val_score
best_score = 0
best_params = {}

# Iterate over different combinations of max_depth and min_samples_leaf
for max_depth in max_depth_range:
    for min_samples_leaf in min_samples_leaf_range:
        # Create a DecisionTreeClassifier with current hyperparameters
        clf = DecisionTreeClassifier(max_depth=max_depth, min_samples_leaf=min_

        # Evaluate model using cross-validation
        cv_scores = cross_val_score(clf, X, y, cv=10)
        avg_score = cv_scores.mean()

        # If current hyperparameters give a better F1 score, update best_score
        if avg_score > best_score:
            best_score = avg_score
            best_params = {'max_depth': max_depth, 'min_samples_leaf': min_samp

# Create a final DecisionTreeClassifier using best hyperparameters and train i
clf = DecisionTreeClassifier(**best_params)
clf.fit(X, y)

# Evaluate the final model using F1 score
y_pred = clf.predict(X)
f1 = f1_score(y, y_pred)
print(f1)
```

```
0.8536585365853658
```

Comparing the performance of the best decision tree classifier and a random forest (with similar max_depth and min_samples_leaf).

In [87]:
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
model = DecisionTreeClassifier(max_depth=4, min_samples_leaf=6)
model.fit(X,y)
t = plot_tree(model,
              feature_names=X.columns,
              class_names=['BMI','Insulin','SkinThickness','BloodPressure','Gl
```



In [88]:
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import cross_validate, ShuffleSplit
from matplotlib import pyplot
cv = ShuffleSplit(n_splits=100)
result = cross_validate(model, X, y, cv=cv, scoring=['f1','precision','recall'
print(result['test_f1'].mean().round(2))
print(result['test_precision'].mean().round(2))
print(result['test_recall'].mean().round(2))
```

```
0.58
0.62
0.57
```

In [89]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_validate, ShuffleSplit
from sklearn.dummy import DummyClassifier
cv = ShuffleSplit(n_splits=100)
def evaluate(model, X, y, cv):
    result = cross_validate(model, X, y, cv=cv, scoring='f1')
    return result['test_score'].mean().round(2)
```

In [90]:
```
dt_model = DecisionTreeClassifier()
rf_model = RandomForestClassifier(n_estimators=50)
baseline = DummyClassifier(strategy='stratified')
```

In [94]:
```
evaluate(baseline, X, y, cv)
```

Out[94]:
```
0.33
```

In [95]:
```
evaluate(rf_model, X, y, cv)
```

Out[95]:
```
0.61
```

In [96]:
```
evaluate(dt_model, X, y, cv)
```

Out[96]:
```
0.56
```

By default, a random forest classifier uses 100 random trees (n_estimators). The larger the number of random trees, the longer it takes to train and predict.

We need to find what is the smallest number random trees in a random forest do we need for a random forest classifer to outperform your best decision tree classifier

In [97]:
```
rf_model = RandomForestClassifier(n_estimators=3,
                                  max_depth=4,
                                  min_samples_leaf=6)
evaluate(rf_model, X, y, cv)
```

Out[97]:
```
0.61
```

Comparing the performance of the best decision tree classifier and logistic regression classifier.

If our logistic regression classifer doesn't convert, we can increase max_iter.

In [98]:
```
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import ShuffleSplit, cross_validate
model1 = RandomForestClassifier(n_estimators=20)
model2 = LinearSVC(max_iter=10000)
model3 = DecisionTreeClassifier()
model4 = LogisticRegression()

cv = ShuffleSplit(n_splits=30)
r1 = cross_validate(model1, X, y, scoring='f1', cv=cv)
r2 = cross_validate(model2, X, y, scoring='f1', cv=cv)
r3 = cross_validate(model3, X, y, scoring='f1', cv=cv)
r4 = cross_validate(model4, X, y, scoring='f1', cv=cv)

print(r1['test_score'].mean())
print(r2['test_score'].mean())
print(r3['test_score'].mean())
print(r4['test_score'].mean())
```

```
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:1206: Converge
nceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
0.6085752840985883
0.5175340886207319
0.5548421466094998
0.6259998616155119
```

```
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:8
14: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
```

```
In [ ]:
```