Importing / Loading library in the Module

In [2]:
```
import numpy as np
import pandas as pd
import sklearn as svm
import sklearn.preprocessing as StandardScaler
from sklearn.model_selection  import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

import data from files

In [3]:
```
import pandas as pd

diabetes_dataset = pd.read_csv("C:\\Users\\Dell\\Downloads\\jupyter nootbook\\M
```

In [4]:
```
diabetes_dataset.head()
```

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

data set statstistical

In [5]:
```
diabetes_dataset.shape
```

Out[5]:
```
(769, 9)
```

In [6]:
```
diabetes_dataset.describe()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes
count	769.000000	769.000000	769.000000	769.000000	769.000000	769.000000	
mean	3.841352	120.858257	69.106632	20.550065	79.695709	31.990637	
std	3.368946	31.967626	19.343229	15.946293	115.204894	7.879210	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	29.000000	32.000000	
75%	6.000000	140.000000	80.000000	32.000000	127.000000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

counts diabetes and non-diabetic persons

In [7]:
```
diabetes_dataset['Outcome'].value_counts()
```

Out[7]:
```
0    501
1    268
Name: Outcome, dtype: int64
```

In [8]:
```
diabetes_dataset.groupby('Outcome').mean()
```

Out[8]:

Outcome	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabet
0	3.293413	109.946108	68.187625	19.686627	68.654691	30.304591	
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	

0-> Non-Diabetic 1-> Diabetic

separat the data

In [9]:
```
X = diabetes_dataset.drop(columns='Outcome' , axis=1)
Y= diabetes_dataset['Outcome']
```

In [10]:
```
print(X)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           ...      ...            ...            ...      ...   ...
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4
768            1       93             70             31        0  30.5

     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        ...  ...
764                     0.340   27
765                     0.245   30
766                     0.349   47
767                     0.315   23
768                     0.314   23

[769 rows x 8 columns]
```

In [11]:
```
print(Y)
```

```
0      1
1      0
2      1
3      0
4      1
      ..
764    0
765    0
766    1
767    0
768    0
Name: Outcome, Length: 769, dtype: int64
```

Data Standardization

In [12]:
```
from sklearn.preprocessing import StandardScaler

# Assuming you have a dataset X, you can create an instance of StandardScaler
scaler = StandardScaler()
```

In [13]:
```
scaler.fit(X)
```

Out[13]:
```
StandardScaler()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [14]:
```
standardizes_data=scaler.transform(X)
```

In [15]:
```
print(standardizes_data)
```

```
[[ 0.64116564  0.849591    0.14967776 ...  0.20438728  0.46934795
   1.42735285]
 [-0.84394393 -1.12243541 -0.16071019 ... -0.68460491 -0.36462404
  -0.18956857]
 [ 1.23520946  1.94516122 -0.26417284 ... -1.10370123  0.60532164
  -0.10446744]
 ...
 [-0.84394393  0.16094686 -0.47109813 ... -0.24010882 -0.37066732
   1.17204947]
 [-0.84394393 -0.87201936  0.04621511 ... -0.20200915 -0.473403
  -0.87037759]
 [-0.84394393 -0.87201936  0.04621511 ... -0.18930926 -0.47642463
  -0.87037759]]
```

In [16]:
```
X=standardizes_data
Y=diabetes_dataset['Outcome']
```

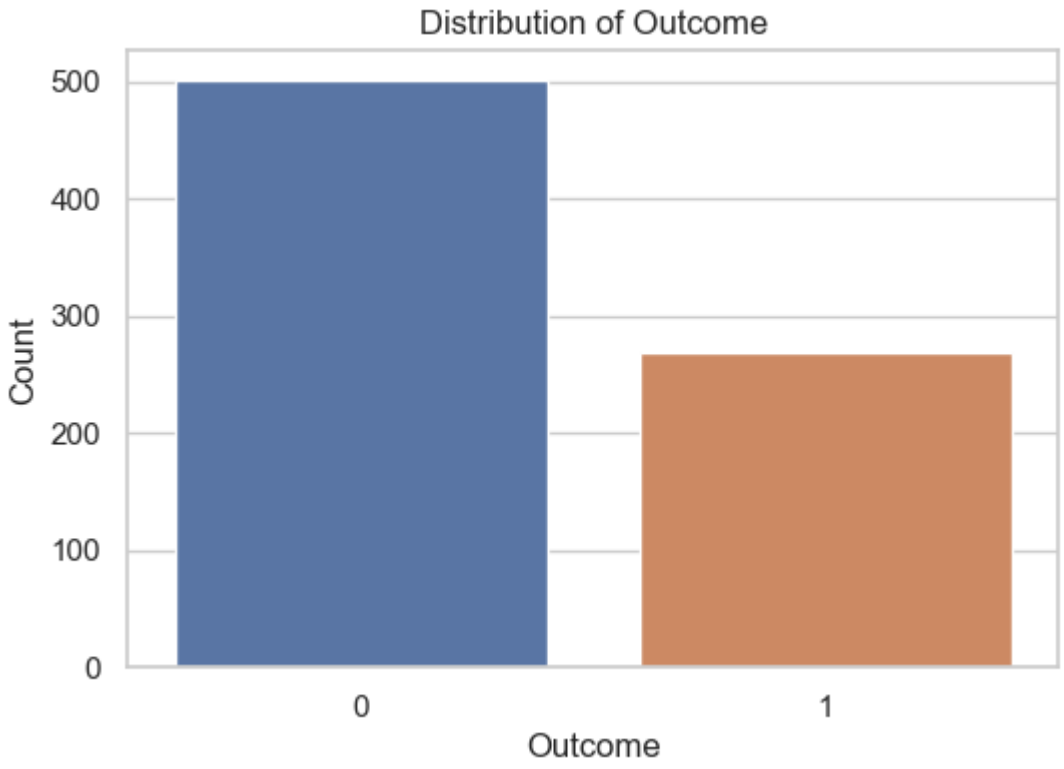
In [17]:
```
print(X)
print(Y)
```

```
[[ 0.64116564  0.849591    0.14967776 ...  0.20438728  0.46934795
   1.42735285]
 [-0.84394393 -1.12243541 -0.16071019 ... -0.68460491 -0.36462404
  -0.18956857]
 [ 1.23520946  1.94516122 -0.26417284 ... -1.10370123  0.60532164
  -0.10446744]
 ...
 [-0.84394393  0.16094686 -0.47109813 ... -0.24010882 -0.37066732
   1.17204947]
 [-0.84394393 -0.87201936  0.04621511 ... -0.20200915 -0.473403
  -0.87037759]
 [-0.84394393 -0.87201936  0.04621511 ... -0.18930926 -0.47642463
  -0.87037759]]
0      1
1      0
2      1
3      0
4      1
      ..
764    0
765    0
766    1
767    0
768    0
Name: Outcome, Length: 769, dtype: int64
```

Visulization Of The Data .

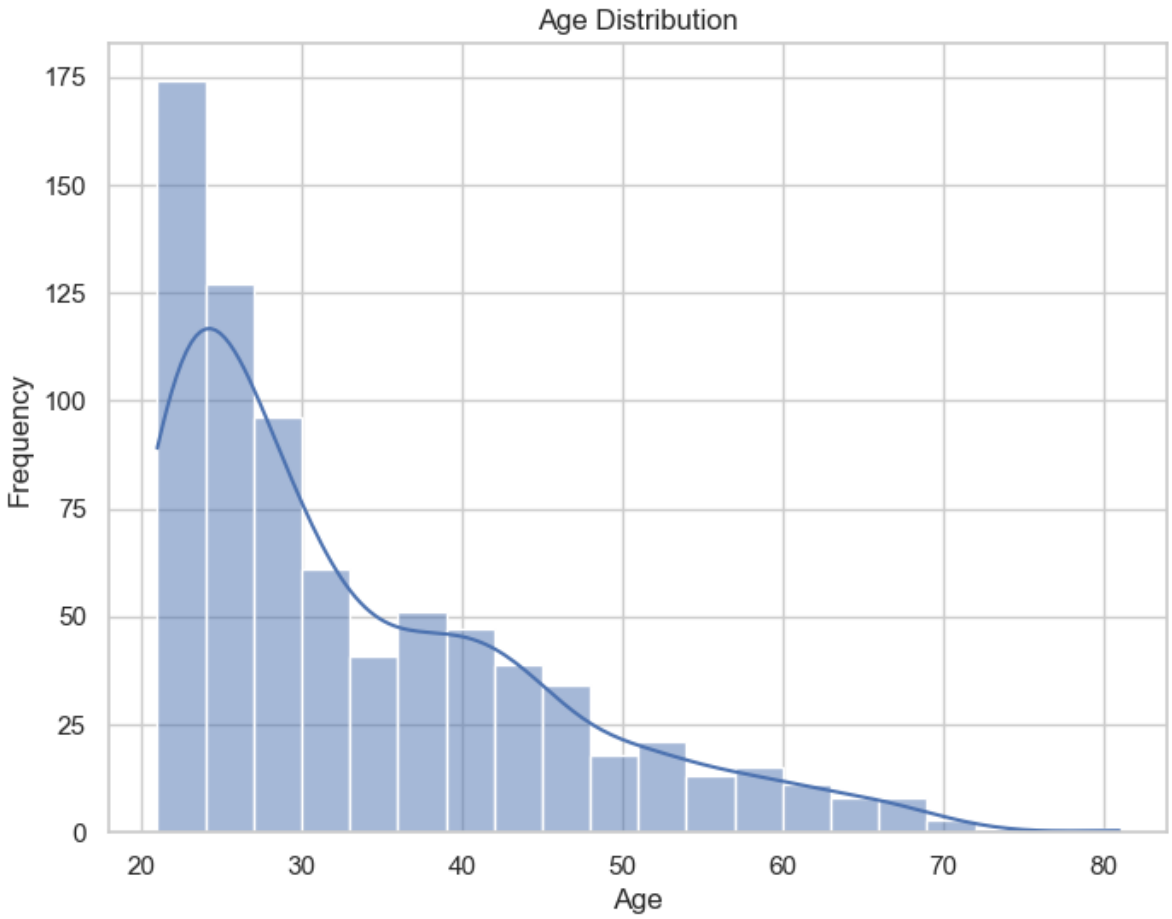
In [18]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load your diabetes dataset using pandas
# diabetes_dataset = pd.read_csv('your_dataset.csv')

# Summary statistics
print(diabetes_dataset.describe())

# Distribution of Outcome (0: Non-Diabetic, 1: Diabetic)
sns.set(style="whitegrid")  # Set the style for Seaborn
plt.figure(figsize=(6, 4))  # Set the figure size for the plot
sns.countplot(x='Outcome', data=diabetes_dataset)
plt.title('Distribution of Outcome')
plt.xlabel('Outcome')
plt.ylabel('Count')
plt.show()
```

```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   769.000000  769.000000     769.000000     769.000000  769.000000
mean      3.841352  120.858257      69.106632      20.550065   79.695709
std       3.368946   31.967626      19.343229      15.946293  115.204894
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   29.000000
75%       6.000000  140.000000      80.000000      32.000000  127.000000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  769.000000                769.000000  769.000000  769.000000
mean    31.990637                  0.471671   33.227568    0.348505
std      7.879210                  0.331162   11.758373    0.476807
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.244000   24.000000    0.000000
50%     32.000000                  0.371000   29.000000    0.000000
75%     36.600000                  0.626000   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

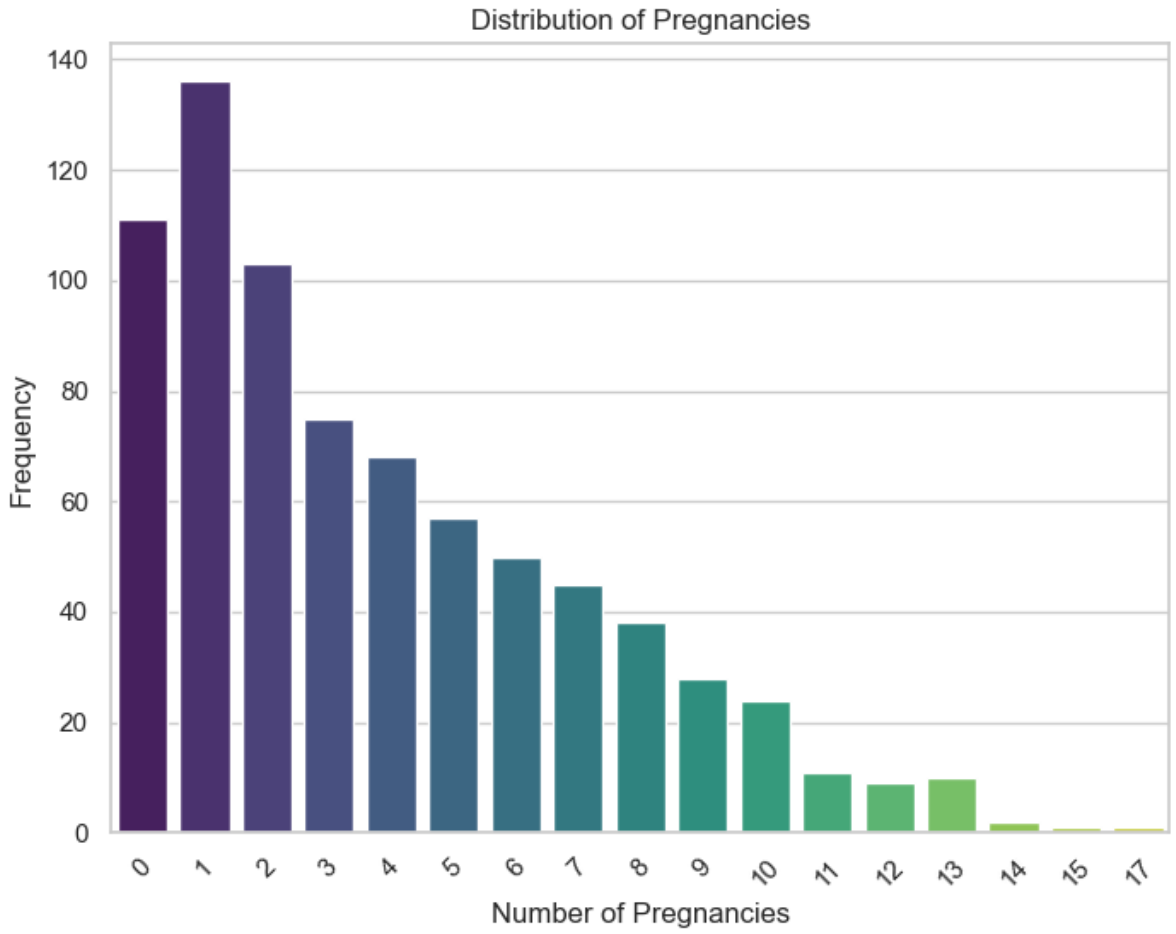

Distribution of Outcome
Count
500
400
300
200
100
0
0
1
Outcome

In [ ]:

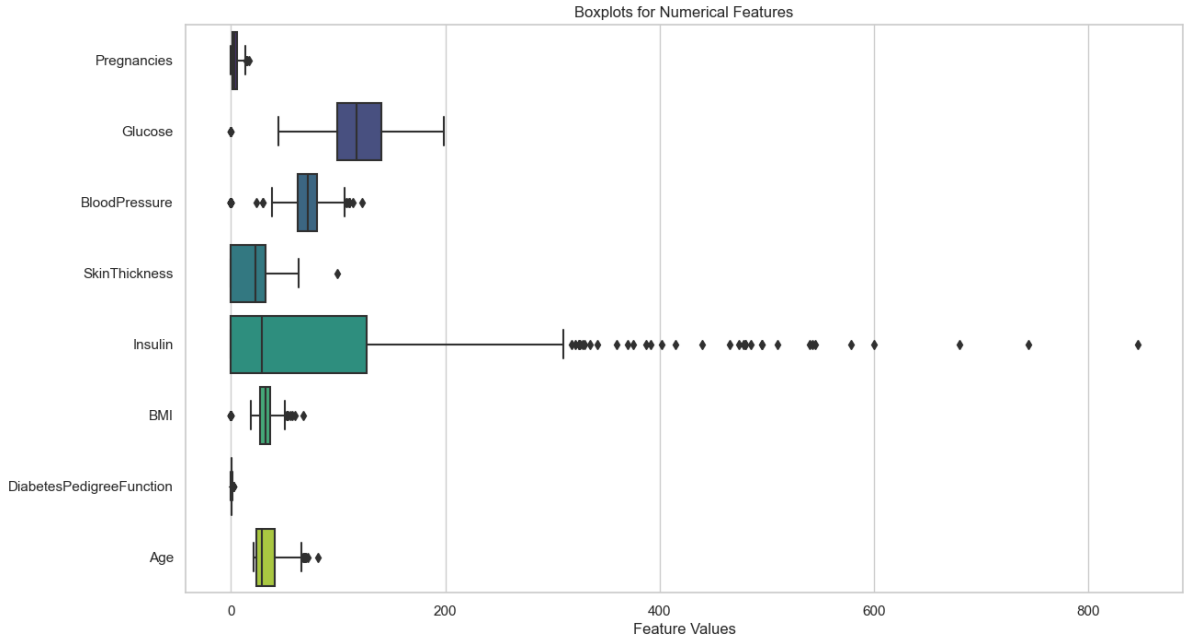
In [19]:
```
import matplotlib.pyplot as plt

# Histogram for 'Age' feature
plt.figure(figsize=(8, 6))
sns.histplot(diabetes_dataset['Age'], bins=20, kde=True)
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



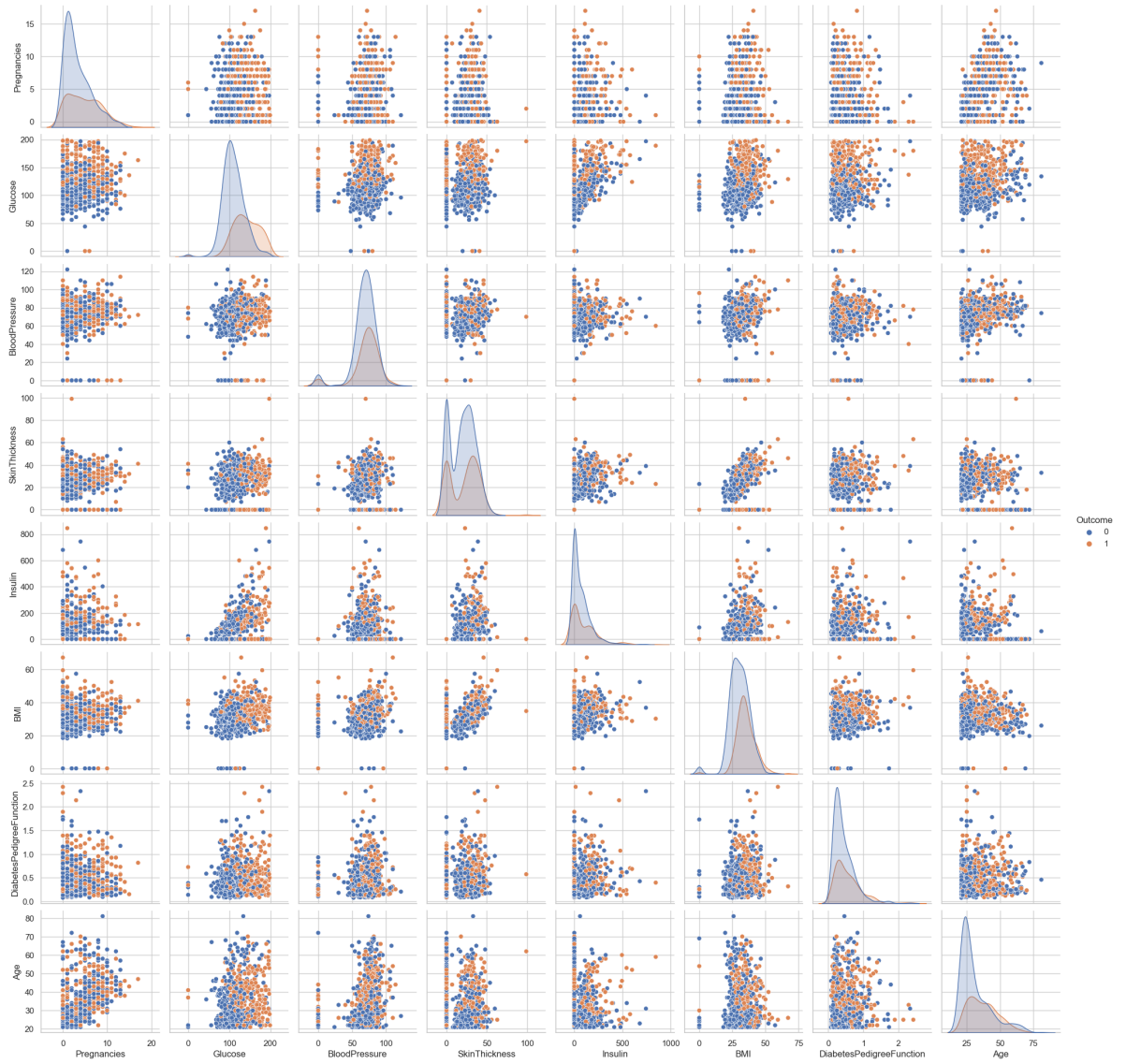
In [20]:
```
# Bar chart for 'Pregnancies'
plt.figure(figsize=(8, 6))
sns.countplot(x='Pregnancies', data=diabetes_dataset, palette='viridis')
plt.title('Distribution of Pregnancies')
plt.xlabel('Number of Pregnancies')
plt.ylabel('Frequency')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```



In [21]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Select only the numerical features (excluding 'Outcome')
numerical_features = diabetes_dataset.drop(columns='Outcome')

# Create boxplots for all numerical features
plt.figure(figsize=(14, 8))
sns.boxplot(data=numerical_features, orient='h', palette='viridis')
plt.title('Boxplots for Numerical Features')
plt.xlabel('Feature Values')
plt.show()
```



In [22]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Pairplot for all numerical features
sns.pairplot(diabetes_dataset, hue='Outcome', diag_kind='kde')
plt.show()
```



Train-Test Split data

In [23]:
```
#split the data into traning data and test data
```

In [24]:
```
X_train ,X_test, Y_train, Y_test =train_test_split(X,Y,test_size= 0.2, stratif
```

In [25]:
```
print(X.shape, X_train.shape , X_test.shape)
```

```
(769, 8) (615, 8) (154, 8)
```

Traning the model

Linear regression

In [26]:
```
classifier=svm.SVC(kernel='linear',C=1.5)
```

traning the support vector machine Classifier

In [27]:
```
classifier.fit(X_train , Y_train)
```

Out[27]:
```
SVC(C=1.5, kernel='linear')
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Model Evaluation

Accuracy Score

In [28]:
```
#Accuracy Score on treanding data
```

In [29]:
```
X_train_prediction = classifier.predict(X_train)

training_data_accuracy=accuracy_score(X_train_prediction, Y_train)
```

In [ ]:
```

```

In [30]:
```
X_test_prediction = classifier.predict(X_test)

test_data_accuracy=accuracy_score(X_test_prediction, Y_test)
```

In [31]:
```
print('Accuracy score of the testing data :' ,test_data_accuracy)
print('Accuracy score of the traning data :' ,training_data_accuracy)
```

```
Accuracy score of the testing data : 0.7662337662337663
Accuracy score of the traning data : 0.7902439024390244
```

In [57]:
```
from sklearn.metrics import precision_score, recall_score, f1_score

# Assuming you have already trained your Logistic Regression model
# X_test_prediction contains the predicted labels for your test data
# Y_test contains the true labels for your test data

# Calculate precision, recall, and F1-score
precision = precision_score(Y_test, X_test_prediction)
recall = recall_score(Y_test, X_test_prediction)
f1 = f1_score(Y_test, X_test_prediction)

# Print the results
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
Precision: 0.7368421052631579
Recall: 0.5185185185185185
F1-score: 0.6086956521739131
```

Type *Markdown* and LaTeX: α^2

In [ ]:
```
```

Making Predictive System

In [33]:
```
input_data = (
1,85,66,29,0,26.6,0.351,31)
```

In [34]:
```
#changing the input data to numpy array
```

In [35]:
```
input_data_as_numpy_array = np.asarray(input_data)
#reshape the arrary as we are predecting for instances
```

In [36]:
```
input_data_reshaped = input_data_as_numpy_array.reshape(1, -1)
```

In [37]:
```
#standardize the input data
```

In [38]:
```
std_data=scaler.transform(input_data_reshaped)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning:
X does not have valid feature names, but StandardScaler was fitted with featu
re names
  warnings.warn(
```

In [39]:
```
print(std_data)
```

```
[[-0.84394393 -1.12243541 -0.16071019  0.5302445  -0.69222386 -0.68460491
  -0.36462404 -0.18956857]]
```

In [40]:
```
prediction = classifier.predict(std_data)
```

In [41]:
```
print(prediction)
```

```
[0]
```

In [42]:
```
if (prediction[0]==0):
    print("the person is not diabetic")
else:
    print("The person is diabetic ")
```

```
the person is not diabetic
```

Random forest

In [43]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

In [44]:
```
rf_classifier = RandomForestClassifier(n_estimators=100,max_depth= 20 ,random_
# You can adjust the number of estimators (trees) as needed.
```

In [45]:
```
rf_classifier = RandomForestClassifier(n_estimators=100,min_samples_leaf= 5 ,r
# You can adjust the number of estimators (trees) as needed.
```

In [46]:
```
rf_classifier.fit(X_train, Y_train)
```

Out[46]:
```
RandomForestClassifier(min_samples_leaf=5, random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [58]:
```
from sklearn.metrics import precision_score, recall_score, f1_score

# Assuming you have already trained your Random Forest classifier
# rf_classifier is your trained Random Forest model
# X_test contains the test features
# Y_test contains the true labels for your test data

# Make predictions on the test data
X_test_predictions = rf_classifier.predict(X_test)

# Calculate precision, recall, and F1-score
precision = precision_score(Y_test, X_test_predictions)
recall = recall_score(Y_test, X_test_predictions)
f1 = f1_score(Y_test, X_test_predictions)

# Print the results
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
Precision: 0.7073170731707317
Recall: 0.5370370370370371
F1-score: 0.6105263157894738
```

In [47]:
```
# Training data predictions
X_train_predictions = rf_classifier.predict(X_train)
training_accuracy = accuracy_score(X_train_predictions, Y_train)
print('Accuracy score on the training data:', training_accuracy)

# Testing data predictions
X_test_predictions = rf_classifier.predict(X_test)
testing_accuracy = accuracy_score(X_test_predictions, Y_test)
print('Accuracy score on the testing data:', testing_accuracy)
```

```
Accuracy score on the training data: 0.9008130081300812
Accuracy score on the testing data: 0.7597402597402597
```

from this above traning data we can see , we can't use random forest classifier .

In [ ]:
```
```

k-Nearest Neighbors (k-NN) Model

In [48]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [49]:
```
knn_classifier = KNeighborsClassifier(n_neighbors=5)  # You can change the numb
```

In [50]:
```
knn_classifier.fit(X_train, Y_train)
```

Out[50]:
```
KNeighborsClassifier()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [51]:
```
# Training data predictions
X_train_predictions = knn_classifier.predict(X_train)
training_accuracy = accuracy_score(X_train_predictions, Y_train)
print('Accuracy score on the training data:', training_accuracy)

# Testing data predictions
X_test_predictions = knn_classifier.predict(X_test)
testing_accuracy = accuracy_score(X_test_predictions, Y_test)
print('Accuracy score on the testing data:', testing_accuracy)
```

```
Accuracy score on the training data: 0.8292682926829268
Accuracy score on the testing data: 0.7207792207792207
```

In [59]:
```
from sklearn.metrics import precision_score, recall_score, f1_score

# Assuming you have already trained your k-NN classifier
# knn_classifier is your trained k-NN model
# X_test contains the test features
# Y_test contains the true labels for your test data

# Make predictions on the test data
X_test_predictions = knn_classifier.predict(X_test)

# Calculate precision, recall, and F1-score
precision = precision_score(Y_test, X_test_predictions)
recall = recall_score(Y_test, X_test_predictions)
f1 = f1_score(Y_test, X_test_predictions)

# Print the results
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
Precision: 0.6486486486486487
Recall: 0.4444444444444444
F1-score: 0.5274725274725275
```

In [ ]:

Decision Tree Classifier:

In [52]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

In [53]:
```
dt_classifier = DecisionTreeClassifier(max_depth=5,random_state=42)
# You can adjust hyperparameters like max_depth, min_samples_split, etc., as ne
```

In [54]:
```
dt_classifier = DecisionTreeClassifier(min_samples_leaf=5, random_state=42)
```

In [55]:
```
dt_classifier.fit(X_train, Y_train)
```

Out[55]:
```
DecisionTreeClassifier(min_samples_leaf=5, random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [56]:
```
# Training data predictions
X_train_predictions = dt_classifier.predict(X_train)
training_accuracy = accuracy_score(X_train_predictions, Y_train)
print('Accuracy score on the training data:', training_accuracy)

# Testing data predictions
X_test_predictions = dt_classifier.predict(X_test)
testing_accuracy = accuracy_score(X_test_predictions, Y_test)
print('Accuracy score on the testing data:', testing_accuracy)
```

```
Accuracy score on the training data: 0.9008130081300812
Accuracy score on the testing data: 0.7142857142857143
```

from above acore of traning we are come accross that we are not using the DecisiomTreeClassifier .

In [60]:
```
from sklearn.metrics import precision_score, recall_score, f1_score

# Assuming you have already trained your Decision Tree Classifier
# dt_classifier is your trained Decision Tree model
# X_test contains the test features
# Y_test contains the true labels for your test data

# Make predictions on the test data
X_test_predictions = dt_classifier.predict(X_test)

# Calculate precision, recall, and F1-score
precision = precision_score(Y_test, X_test_predictions)
recall = recall_score(Y_test, X_test_predictions)
f1 = f1_score(Y_test, X_test_predictions)

# Print the results
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
Precision: 0.5961538461538461
Recall: 0.5740740740740741
F1-score: 0.5849056603773585
```

Support vector machine -SVM

In [73]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Create an SVM classifier with a linear kernel (you can choose a different ker
svm_classifier = SVC(kernel='linear', C=1.5)

# Train the SVM classifier on the training data
svm_classifier.fit(X_train, Y_train)

# Make predictions on the training data
Y_train_pred = svm_classifier.predict(X_train)

# Make predictions on the testing data
Y_test_pred = svm_classifier.predict(X_test)

# Calculate training accuracy
training_accuracy = accuracy_score(Y_train, Y_train_pred)
print("Training Accuracy:", training_accuracy)

# Calculate testing accuracy
testing_accuracy = accuracy_score(Y_test, Y_test_pred)
print("Testing Accuracy:", testing_accuracy)
```

```
Training Accuracy: 0.7902439024390244
Testing Accuracy: 0.7662337662337663
```

In [77]:
```
from sklearn.metrics import precision_score, recall_score, f1_score

# Make predictions on the test data
y_pred = svm_classifier.predict(X_test)

# Calculate precision, recall, and F1-score
precision = precision_score(Y_test, y_pred)
recall = recall_score(Y_test, y_pred)
f1 = f1_score(Y_test, y_pred)

# Print the results
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
Precision: 0.7368421052631579
Recall: 0.5185185185185185
F1-score: 0.6086956521739131
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: