Importing / Loading library in the Module

In [588]:
```
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 [589]:
```
import pandas as pd

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

In [590]:
```
diabetes_dataset.head()
```

Out[590]:

	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 [591]:
```
diabetes_dataset.shape
```

Out[591]:
```
(768, 9)
```

In [592]:
```
diabetes_dataset.describe()
```

Out[592]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
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	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

counts diabetes and non-diabetic persons

In [593]:
```
diabetes_dataset['Outcome'].value_counts()
```

Out[593]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

In [594]:
```
diabetes_dataset.groupby('Outcome').mean()
```

Out[594]:

Outcome	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabet
0	3.298000	109.980000	68.184000	19.664000	68.792000	30.304200	
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	

0-> Non-Diabetic 1-> Diabetic

separat the data

In [595]:
```
X = diabetes_dataset.drop(columns='Outcome' , axis=1)
Y= diabetes_dataset['Outcome']
```

In [596]:
```
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
..           ...      ...            ...            ...      ...   ...
763           10      101             76             48      180  32.9
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

     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        ...  ...
763                     0.171   63
764                     0.340   27
765                     0.245   30
766                     0.349   47
767                     0.315   23

[768 rows x 8 columns]
```

In [597]:
```
print(Y)
```

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Data Standardization

In [598]:
```
from sklearn.preprocessing import StandardScaler

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

In [599]:
```
scaler.fit(X)
```

Out[599]:
```
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 [600]:
```
standardizes_data=scaler.transform(X)
```

In [601]:
```
print(standardizes_data)
```

```
[[ 0.63994726  0.84832379  0.14964075 ...  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 ... -1.10325546  0.60439732
  -0.10558415]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 ... -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 ... -0.20212881 -0.47378505
  -0.87137393]]
```

In [602]:
```
X=standardizes_data
Y=diabetes_dataset['Outcome']
```

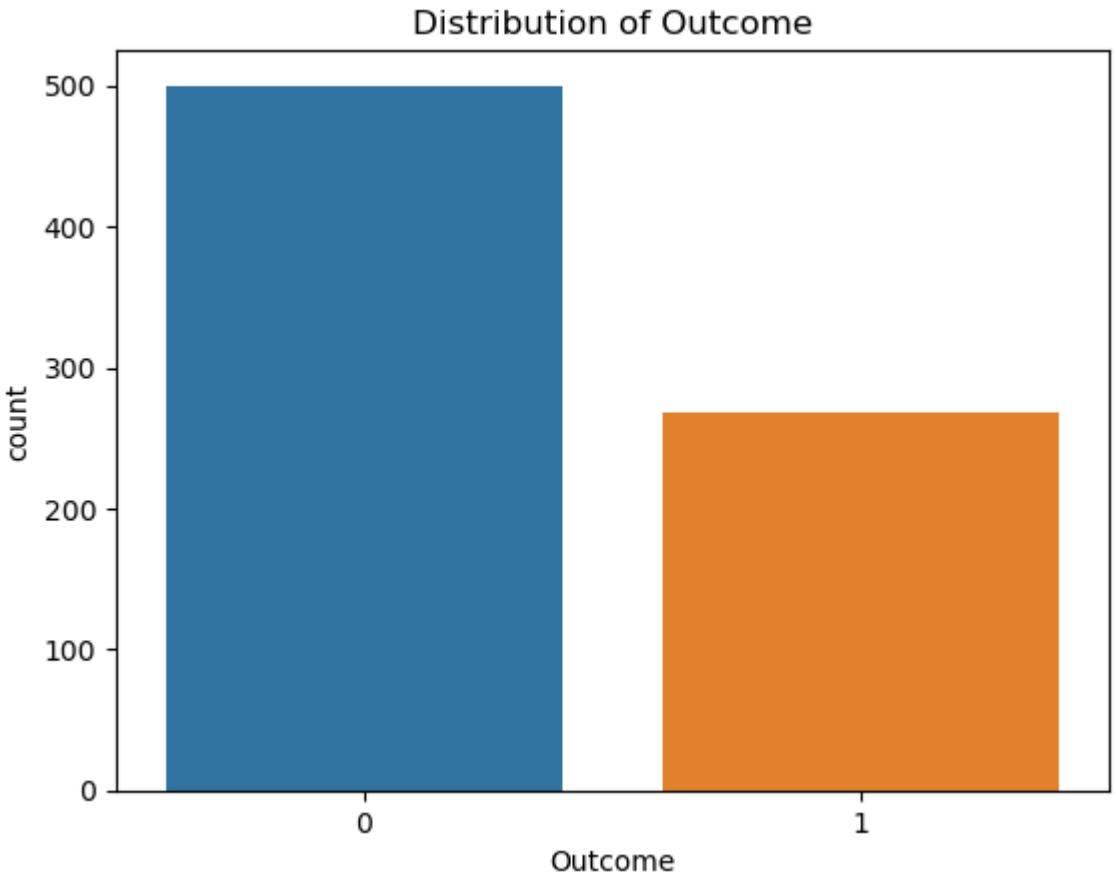
In [603]:
```
print(X)
print(Y)
```

```
[[ 0.63994726  0.84832379  0.14964075 ...  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 ... -1.10325546  0.60439732
  -0.10558415]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 ... -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 ... -0.20212881 -0.47378505
  -0.87137393]]
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

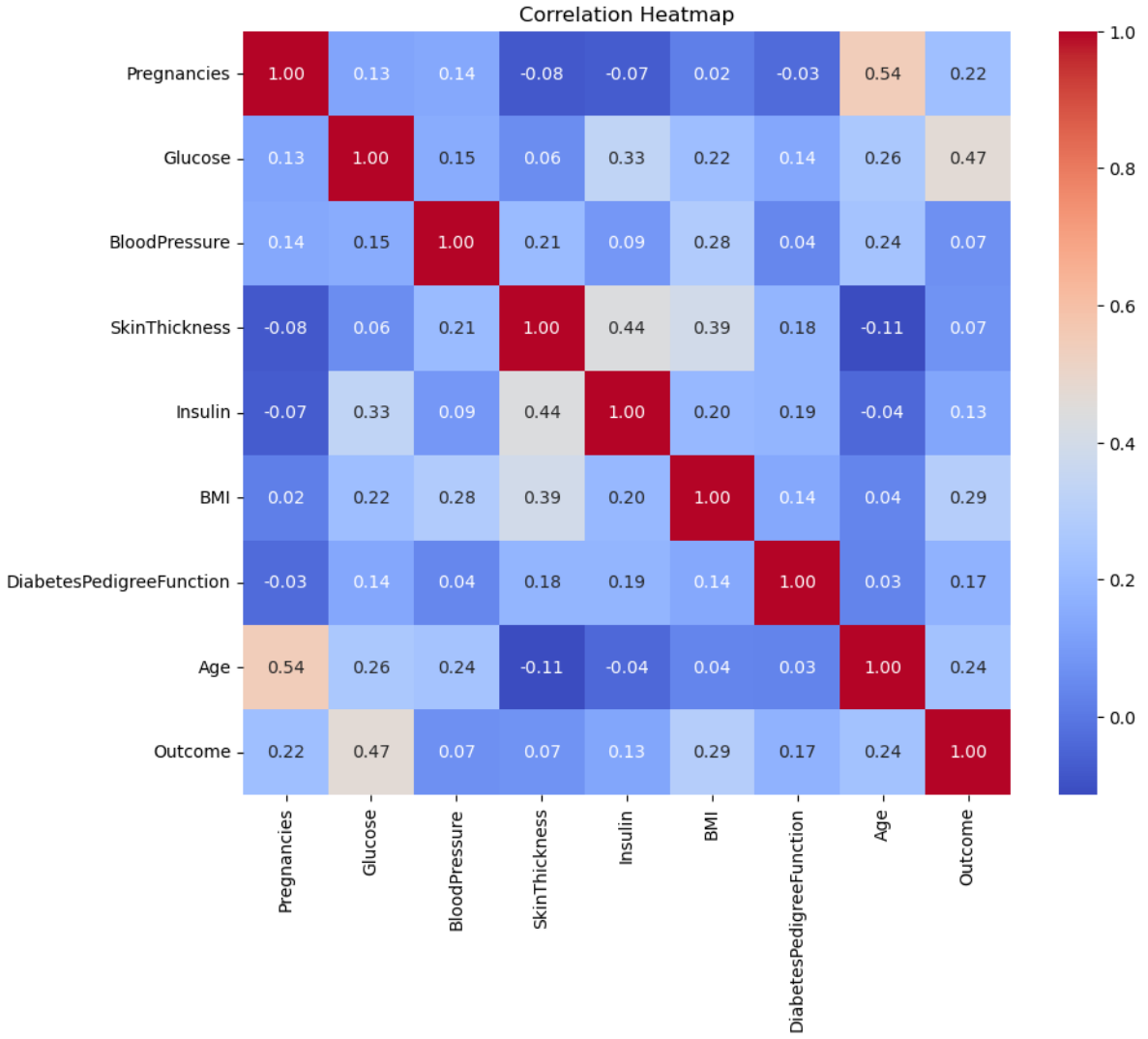
Visulization Of The Data .

In [604]:
```
# Summary statistics
print(diabetes_dataset.describe())

# Distribution of Outcome (0: Non-Diabetic, 1: Diabetic)
sns.countplot(x='Outcome', data=diabetes_dataset)
plt.title('Distribution of Outcome')
plt.show()
```

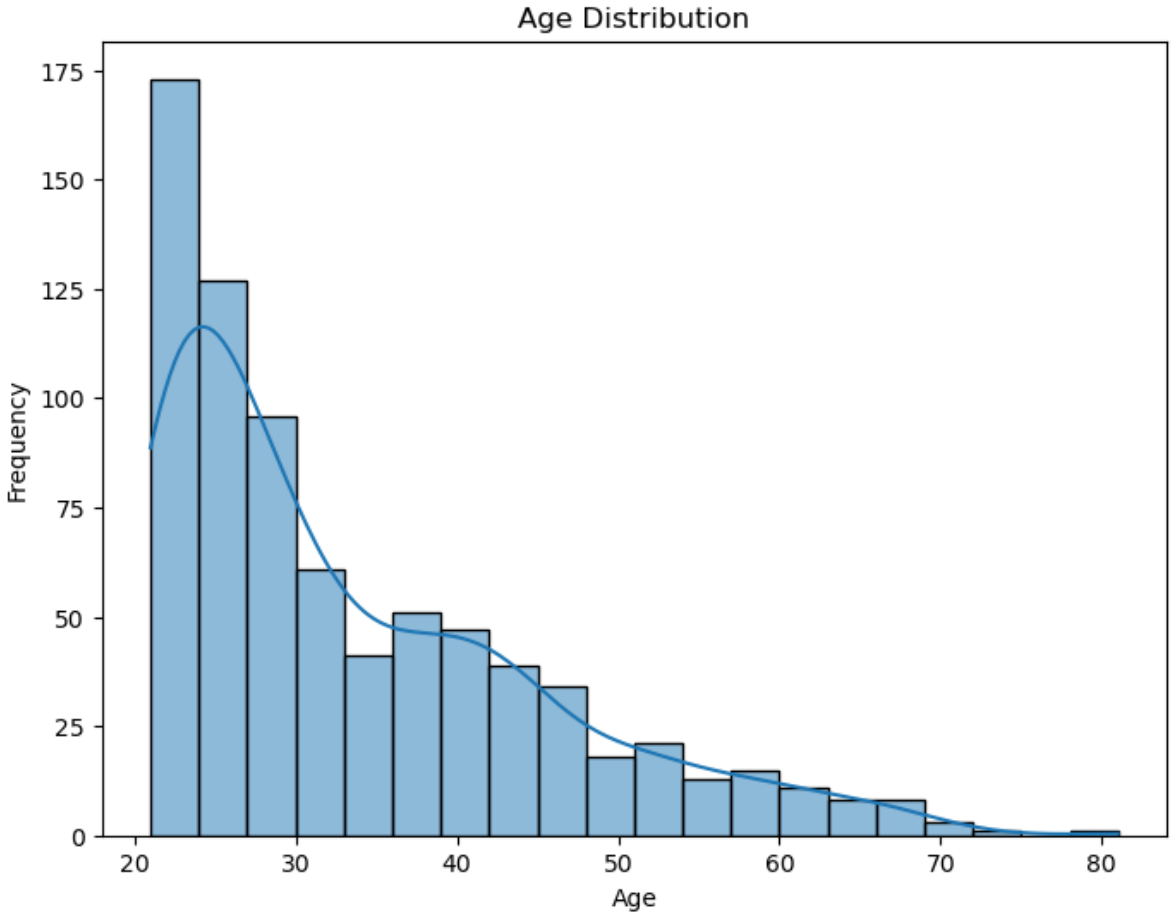
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       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   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```


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

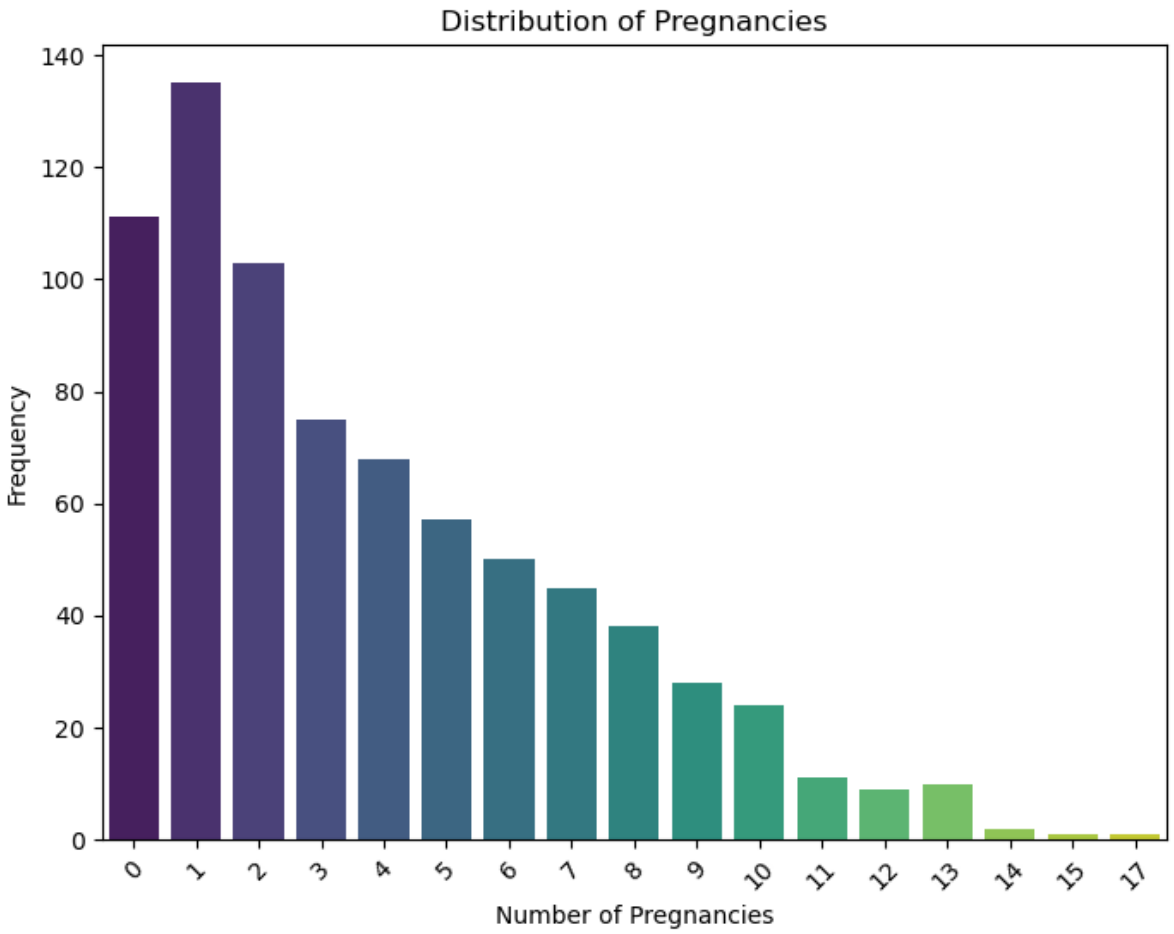
In [605]:
```
# Correlation matrix
correlation_matrix = diabetes_dataset.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



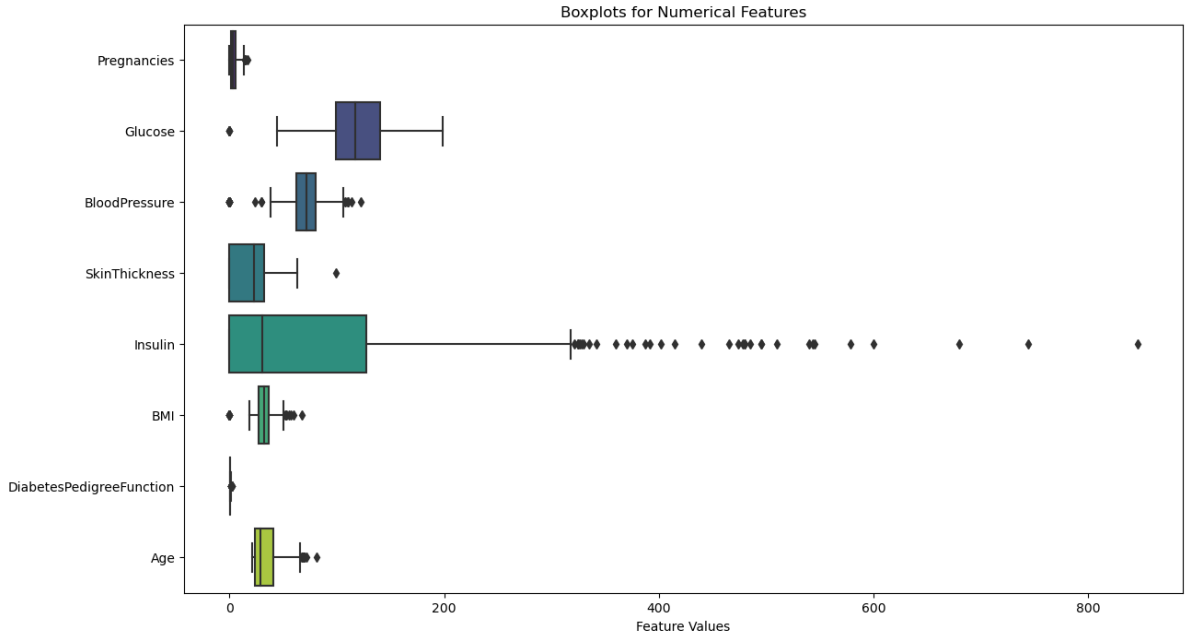
In [606]:
```
import matplotlib.pyplot as plt

# 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 [607]:
```
# 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 [608]:
```
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 [609]:
```
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 [ ]:
```
#split the data into traning data and test data
```

In [553]:
```
X_train ,X_test, Y_train, Y_test =train_test_split(X,Y,test_size= 0.2, stratif
```

In [554]:
```
print(X.shape, X_train.shape , X_test.shape)
```

```
(768, 8) (614, 8) (154, 8)
```

Traning the model

Linear regression

In [555]:
```
classifier=svm.SVC(kernel='linear',C=1.5)
```

traning the support vector machine Classifier

In [556]:
```
classifier.fit(X_train , Y_train)
```

Out[556]:
```
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 [557]:
```
#Accuracy Score on treanding data
```

In [558]:
```
X_train_prediction = classifier.predict(X_train)

training_data_accuracy=accuracy_score(X_train_prediction, Y_train)
```

In [559]:
```
print('Accuracy score of the traning data :' ,training_data_accuracy)
```

```
Accuracy score of the traning data : 0.7850162866449512
```

In [560]:
```
X_test_prediction = classifier.predict(X_test)

test_data_accuracy=accuracy_score(X_test_prediction, Y_test)
```

In [561]:
```
print('Accuracy score of the testing data :' ,test_data_accuracy)
```

```
Accuracy score of the testing data : 0.7727272727272727
```

Making Predictive System

In [562]:
```
input_data = (
1,85,66,29,0,26.6,0.351,31)
```

In [563]:
```
#changing the input data to numpy array
```

In [564]:
```
input_data_as_numpy_array = np.asarray(input_data)
#reshape the arrary as we are predecting for instances
```

In [565]:
```
input_data_reshaped = input_data_as_numpy_array.reshape(1, -1)
```

In [566]:
```
#standardize the input data
```

In [567]:
```
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 [568]:
```
print(std_data)
```

```
[[-0.84488505 -1.12339636 -0.16054575  0.53090156 -0.69289057 -0.68442195
  -0.36506078 -0.19067191]]
```

In [569]:
```
prediction = classifier.predict(std_data)
```

In [570]:
```
print(prediction)
```

```
[0]
```

In [571]:
```
if (prediction[0]==0):
    print("the person is not diabetic")
else:
    print("The person is diabetic ")
```

```
the person is not diabetic
```

Random forest

In [572]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

In [573]:
```
rf_classifier = RandomForestClassifier(n_estimators=100,max_depth= 20 ,random_
# You can adjust the number of estimators (trees) as needed.
```

In [574]:
```
rf_classifier = RandomForestClassifier(n_estimators=100,min_samples_leaf= 5 ,r
# You can adjust the number of estimators (trees) as needed.
```

In [575]:
```
rf_classifier.fit(X_train, Y_train)
```

Out[575]:
```
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 [610]:
```
# 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: 0.9006514657980456
Accuracy score on the testing data: 0.7337662337662337
```

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

In [ ]:
```
```

k-Nearest Neighbors (k-NN) Model

In [577]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [578]:
```
knn_classifier = KNeighborsClassifier(n_neighbors=5)  # You can change the numb
```

In [579]:
```
knn_classifier.fit(X_train, Y_train)
```

Out[579]:
```
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 [580]:
```
# 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: 0.8289902280130294
Accuracy score on the testing data: 0.7207792207792207
```

In [ ]:

In [ ]:

Decision Tree Classifier:

In [581]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

In [582]:
```
dt_classifier = DecisionTreeClassifier(max_depth=5,random_state=42)
# You can adjust hyperparameters like max_depth, min_samples_split, etc., as ne
```

In [583]:
```
dt_classifier = DecisionTreeClassifier(min_samples_leaf=5, random_state=42)
```

In [584]:
```
dt_classifier.fit(X_train, Y_train)
```

Out[584]:
```
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 [585]:
```
# 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: 0.9022801302931596
Accuracy score on the testing data: 0.7142857142857143
```

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