Diabetes Prediction

Diabetes, is a group of metabolic disorders in which there are high blood sugar levels over a prolonged period. Symptoms of high blood sugar include frequent urination, increased thirst, and increased hunger. If left untreated, diabetes can cause many complications. Acute complications can include diabetic ketoacidosis, hyperosmolar hyperglycemic state, or death. Serious long-term complications include cardiovascular disease, stroke, chronic kidney disease, foot ulcers, and damage to the eyes.

Data Description :-

Pregnancies: Number of times pregnant

Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test

BloodPressure: Diastolic blood pressure (mm Hg)

SkinThickness: Triceps skin fold thickness (mm)

Insulin: 2-Hour serum insulin (mu U/ml)

BMI: Body mass index (weight in kg/(height in m)^2)

DiabetesPedigreeFunction: Diabetes pedigree function

Age: Age (years)

Outcome: Class variable (0 or 1)

In [1]:
```
# Importing the important libraries for data preprocessing and data visualization
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import missingno as msno
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# Importing diabetes dataset
df = pd.read_csv("diabetes.csv") # Kaggle dataset (PIMA Indians Diabetes)
df.head(15)
```

Out[2]:

	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
10	4	110	92	0	0	37.6	0.191
11	10	168	74	0	0	38.0	0.537
12	10	139	80	0	0	27.1	1.441
13	1	189	60	23	846	30.1	0.398
14	5	166	72	19	175	25.8	0.587

In [3]:
```
# shape of df
df.shape
```

Out[3]:
```
(768, 9)
```

In [4]:
```
# Size of df
df.size
```

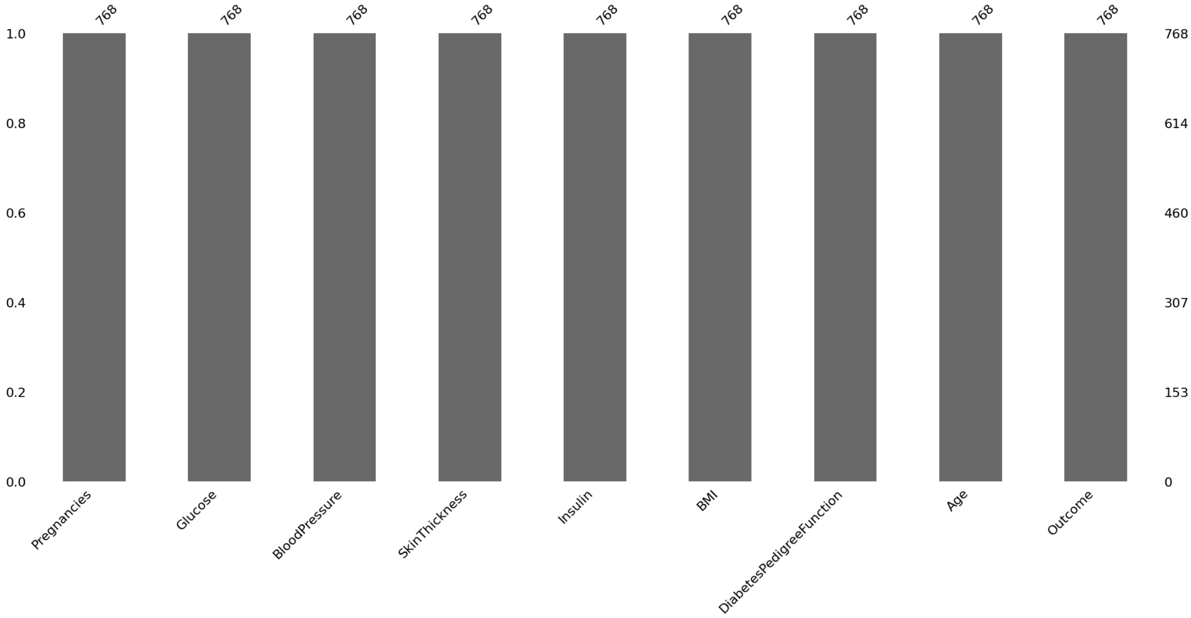
Out[4]:
```
6912
```

In [5]:
```
# Checking Null value
df.isnull().sum()
```

Out[5]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [6]:
```
# Check missing values using bar
msno.bar(df)
```

Out[6]:
```
<Axes: >
```



In [7]:
```
# Checking no of duplicate in each column
df[df.duplicated()].sum()
```

Out[7]:
```
Pregnancies                 0.0
Glucose                     0.0
BloodPressure               0.0
SkinThickness               0.0
Insulin                     0.0
BMI                         0.0
DiabetesPedigreeFunction    0.0
Age                         0.0
Outcome                     0.0
dtype: float64
```

In [8]:
```
# Info about dataset
df.info()
```

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

In [9]:
```
# Summary of dataset
df.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	75%
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000

In [10]:

```
df['Outcome'].value_counts()
```

Out[10]:

```
0    500
1    268
Name: Outcome, dtype: int64
```

0---> Non Diabetic (500 people)

1---> Diabetic (268 people)

In [11]:

```
df.groupby('Outcome').mean()
```

Out[11]:

Outcome	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesI
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	

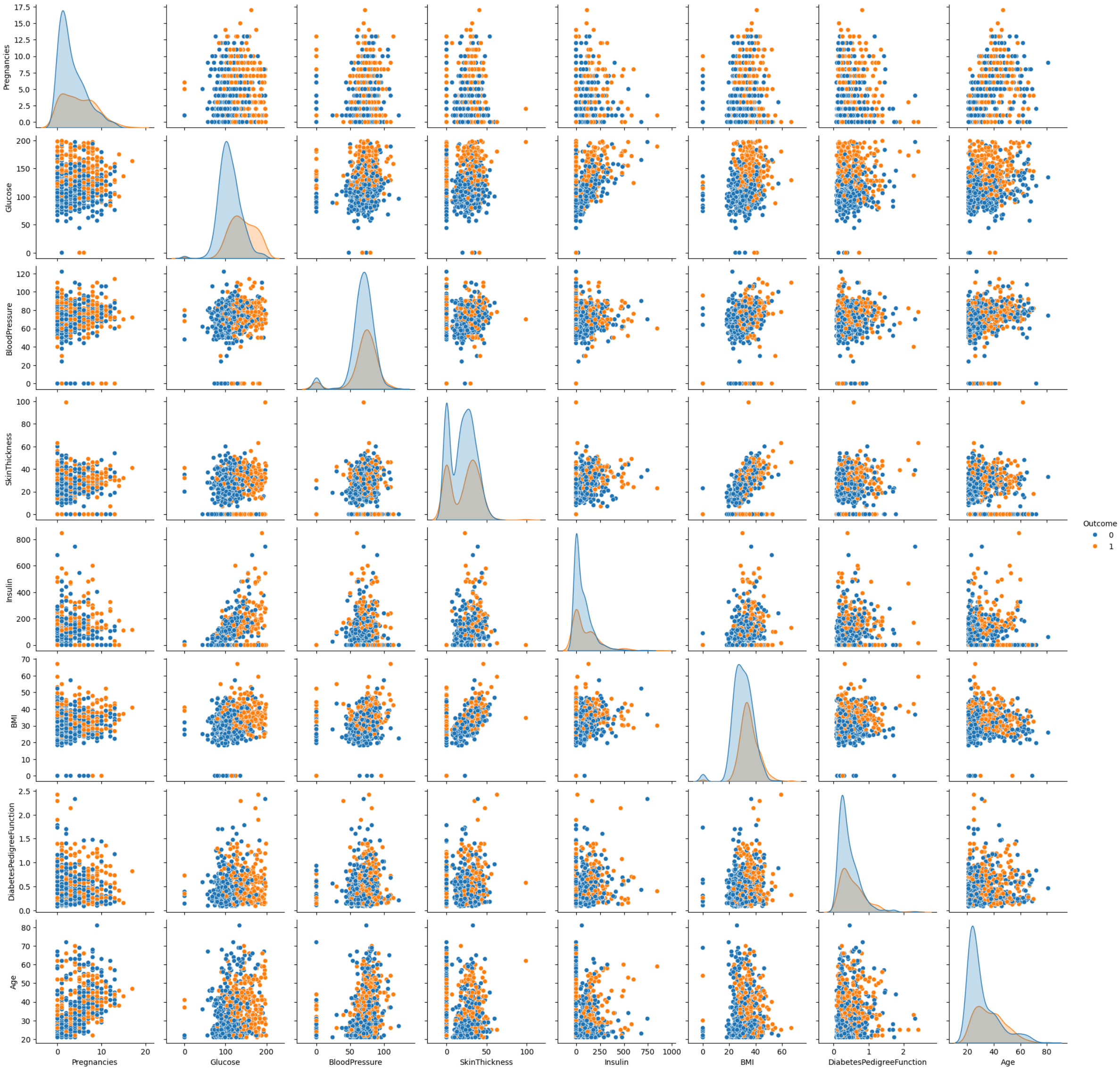
We can clearly see that glucose level, BP, Insulin and BMI is maximum for Diabetic People.

In [12]:

```
sns.pairplot(df,hue='Outcome')
```

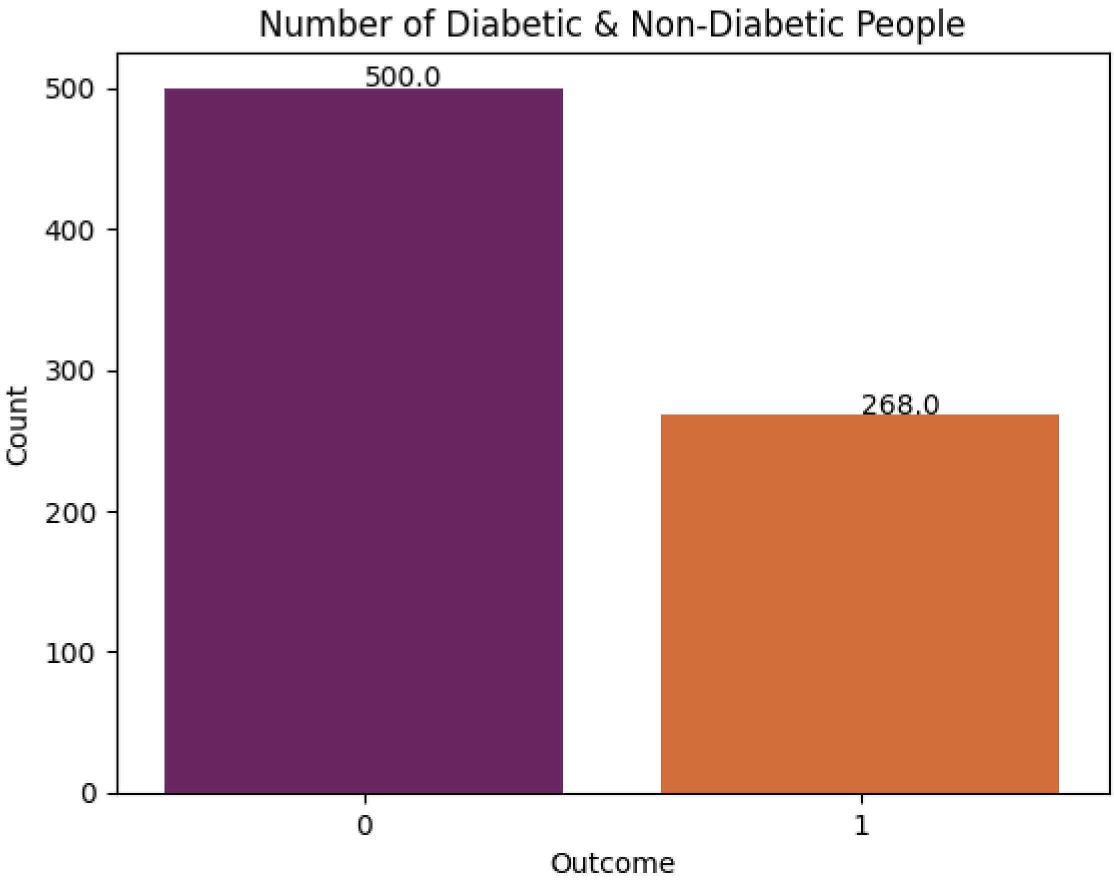
Out[12]:

```
<seaborn.axisgrid.PairGrid at 0x7cb12986f0a0>
```



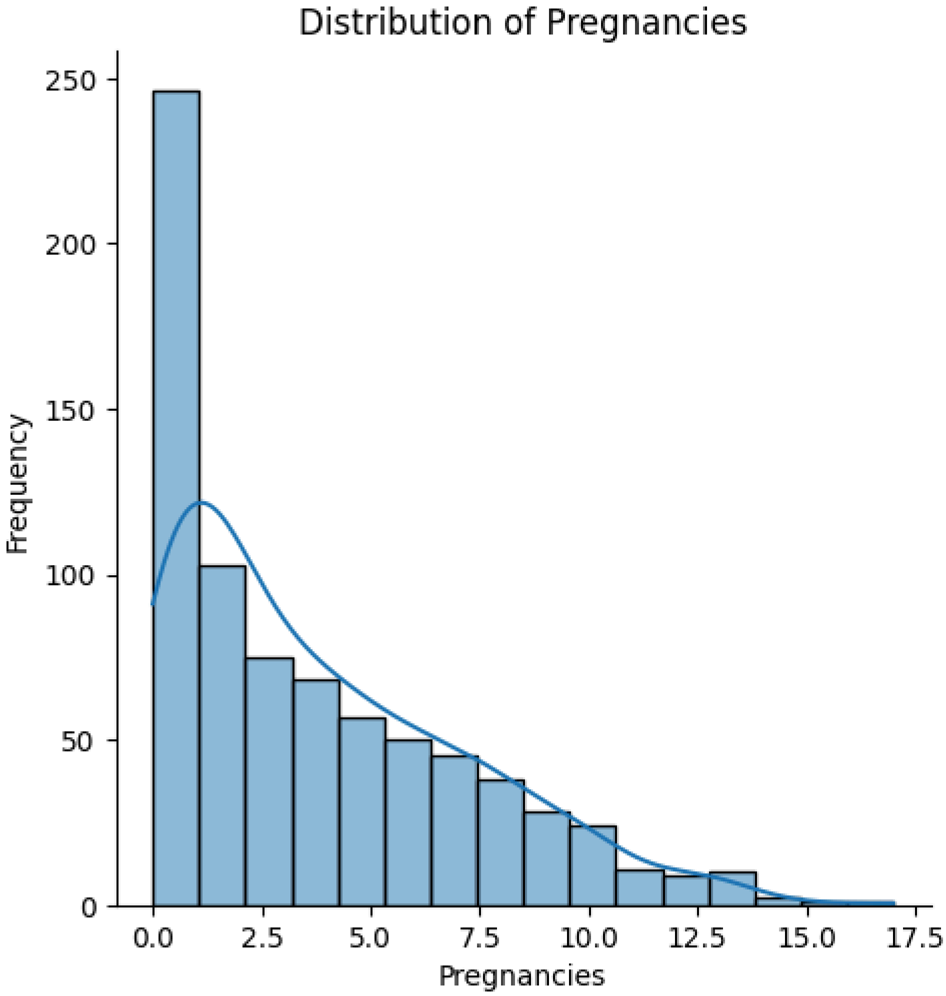
Univaiate and Bivariate Analysis

In [13]:
```
ax = sns.countplot(x='Outcome',data=df,palette='inferno')
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x()+0.4, p.get_height()+1))
plt.title('Number of Diabetic & Non-Diabetic People')
plt.xlabel('Outcome')
plt.ylabel('Count')
plt.show()
```

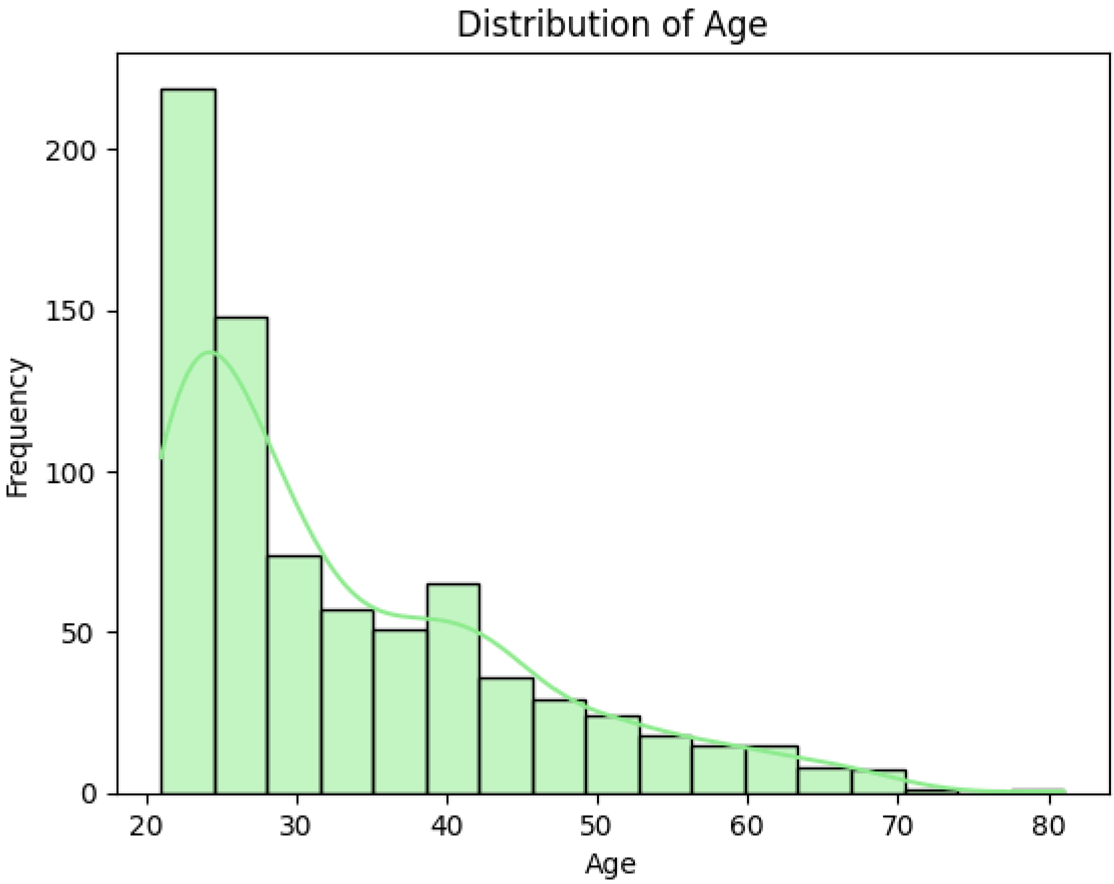


In [14]:
```
# Pregnancies Distribution
plt.figure(figsize=(10, 6))
sns.displot(data=df,x='Pregnancies',kde=True)
plt.title('Distribution of Pregnancies')
plt.xlabel('Pregnancies')
plt.ylabel('Frequency')
plt.show()
```

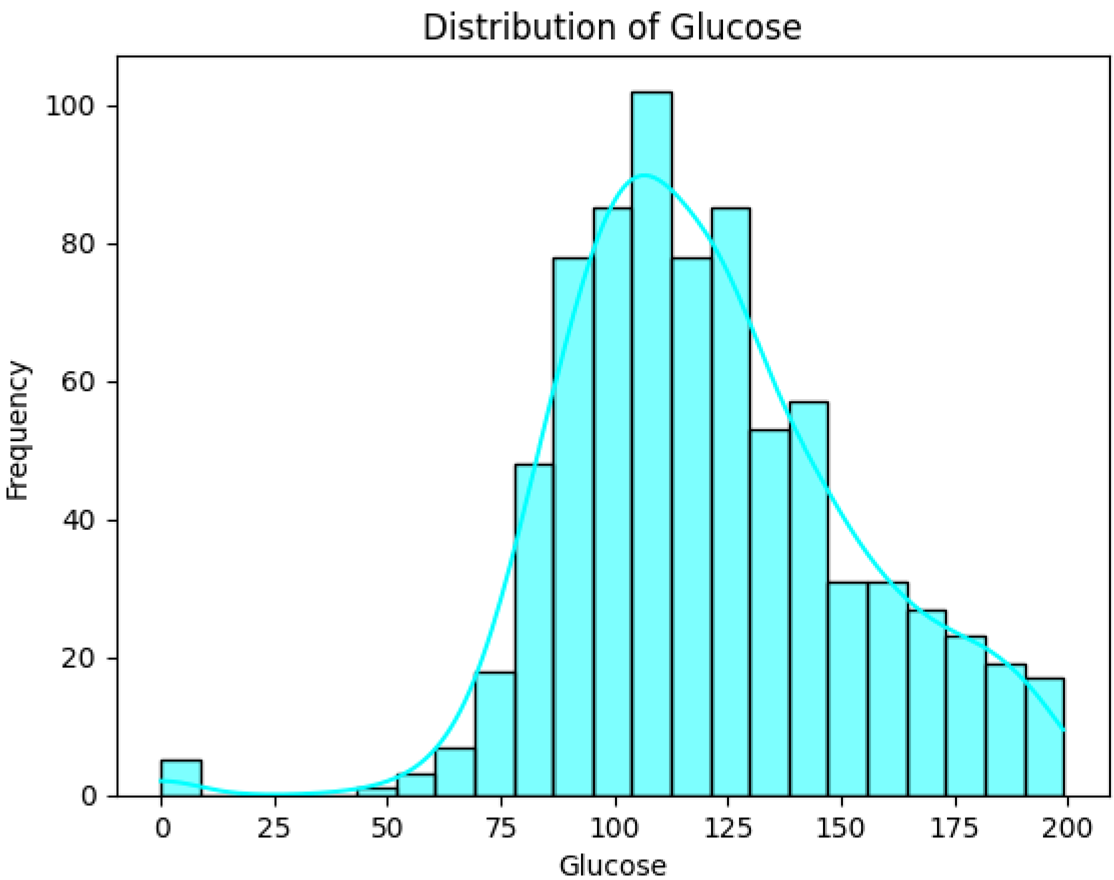
```
<Figure size 1000x600 with 0 Axes>
```



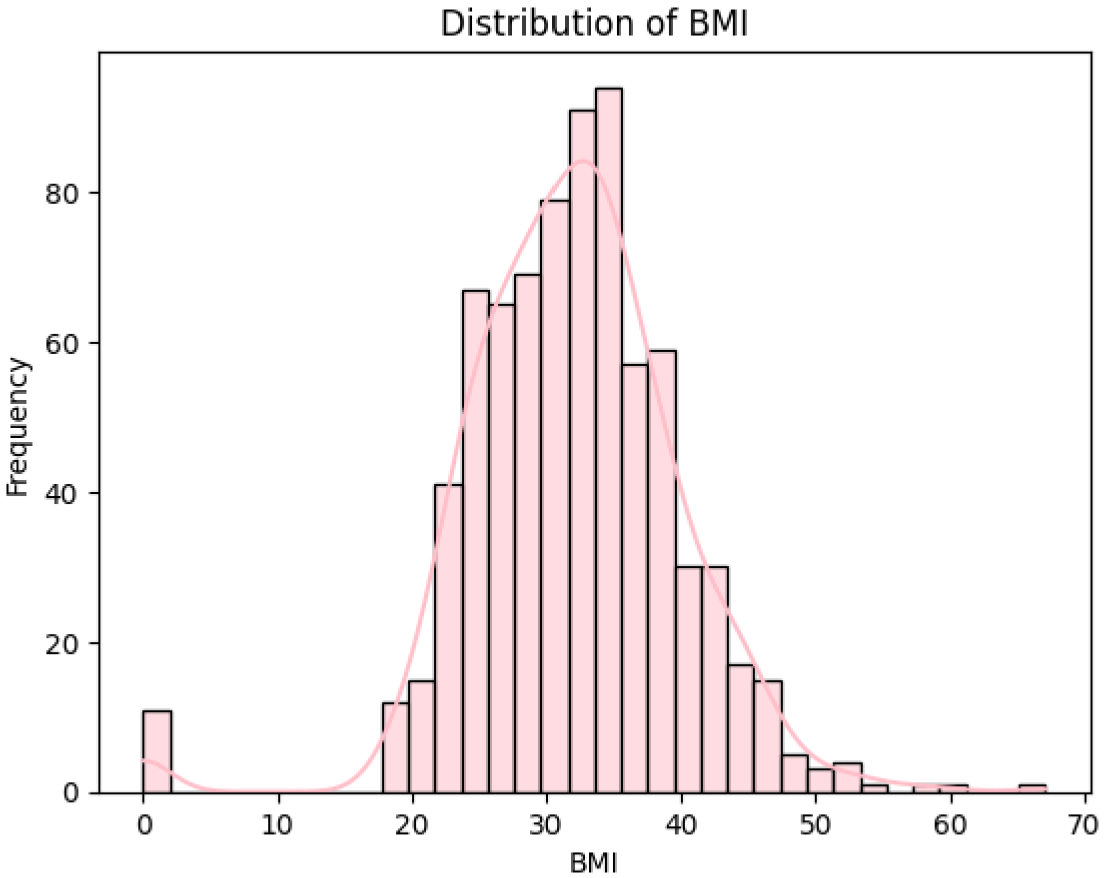
In [15]:
```
sns.histplot(data=df,x='Age',color='lightgreen',kde=True)
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



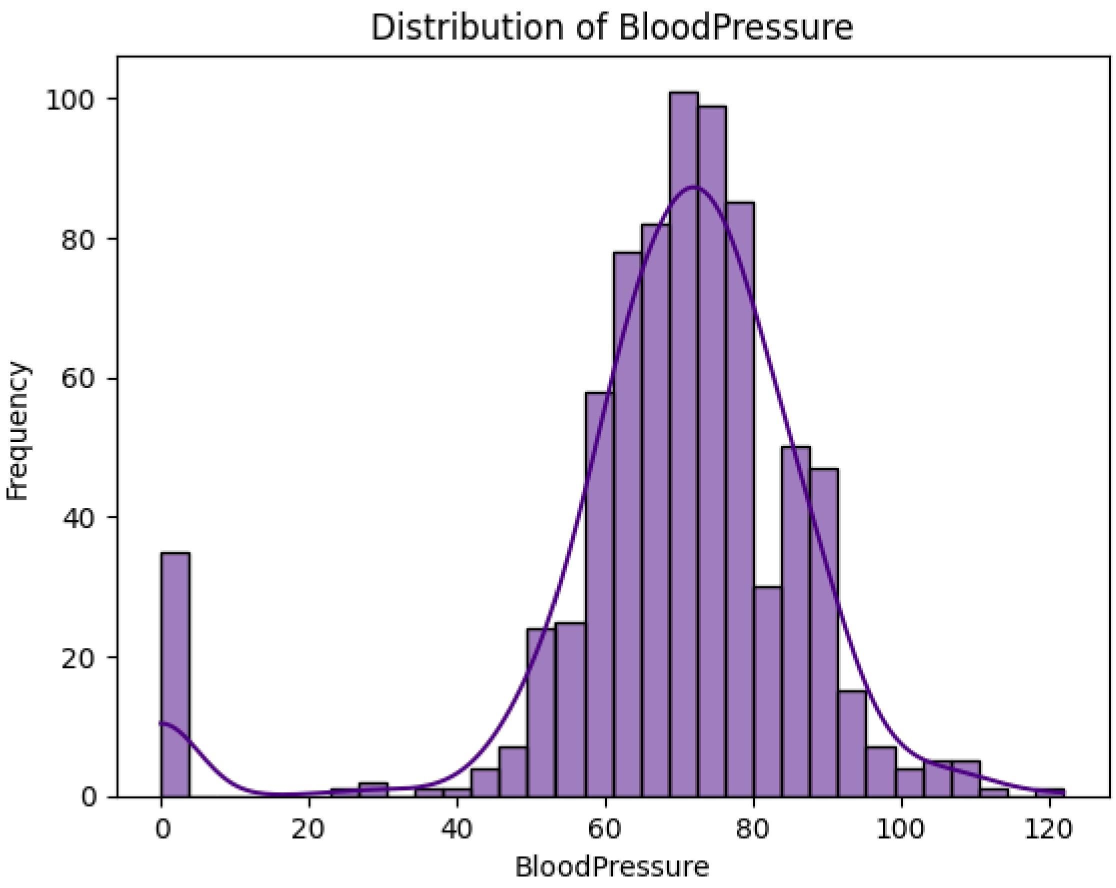
In [16]:
```
# Glucose Distribution
sns.histplot(data=df,x='Glucose',kde=True,color='cyan')
plt.title('Distribution of Glucose')
plt.xlabel('Glucose')
plt.ylabel('Frequency')
plt.show()
```



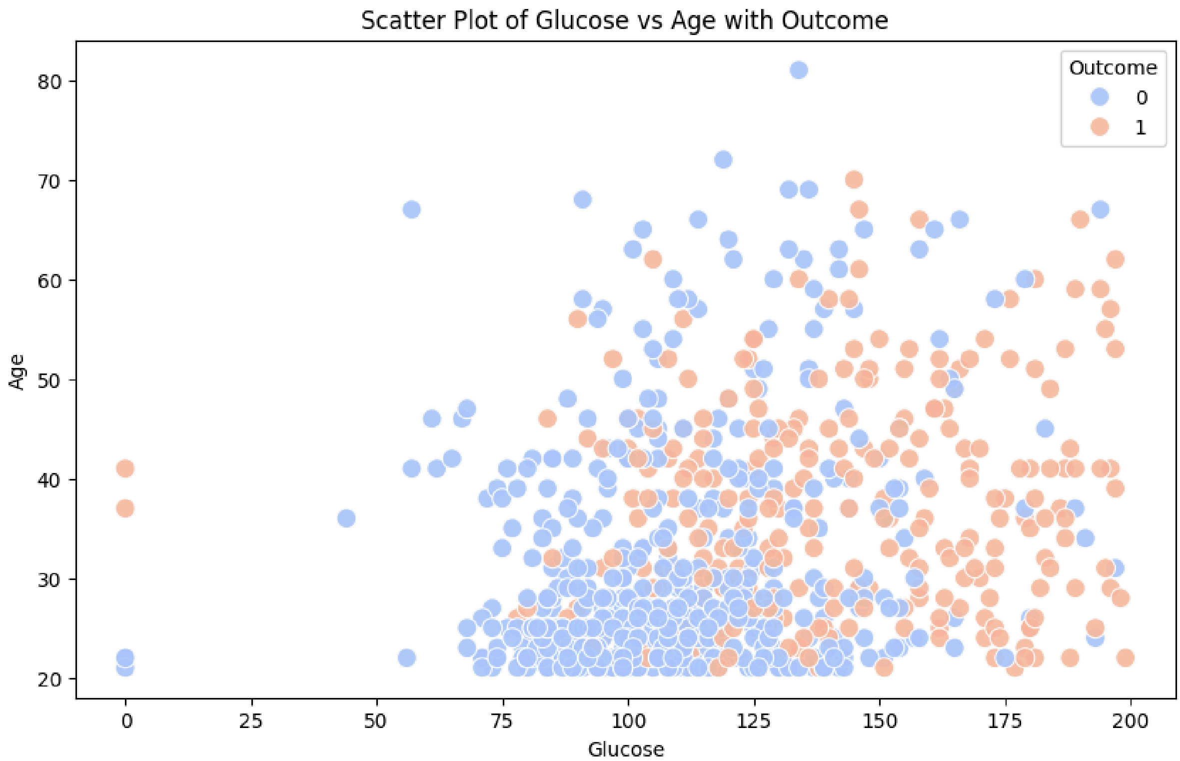
In [17]:
```
# BMI Distribution
sns.histplot(data=df,x='BMI',kde=True,color='pink')
plt.title('Distribution of BMI')
plt.xlabel('BMI')
plt.ylabel('Frequency')
plt.show()
```



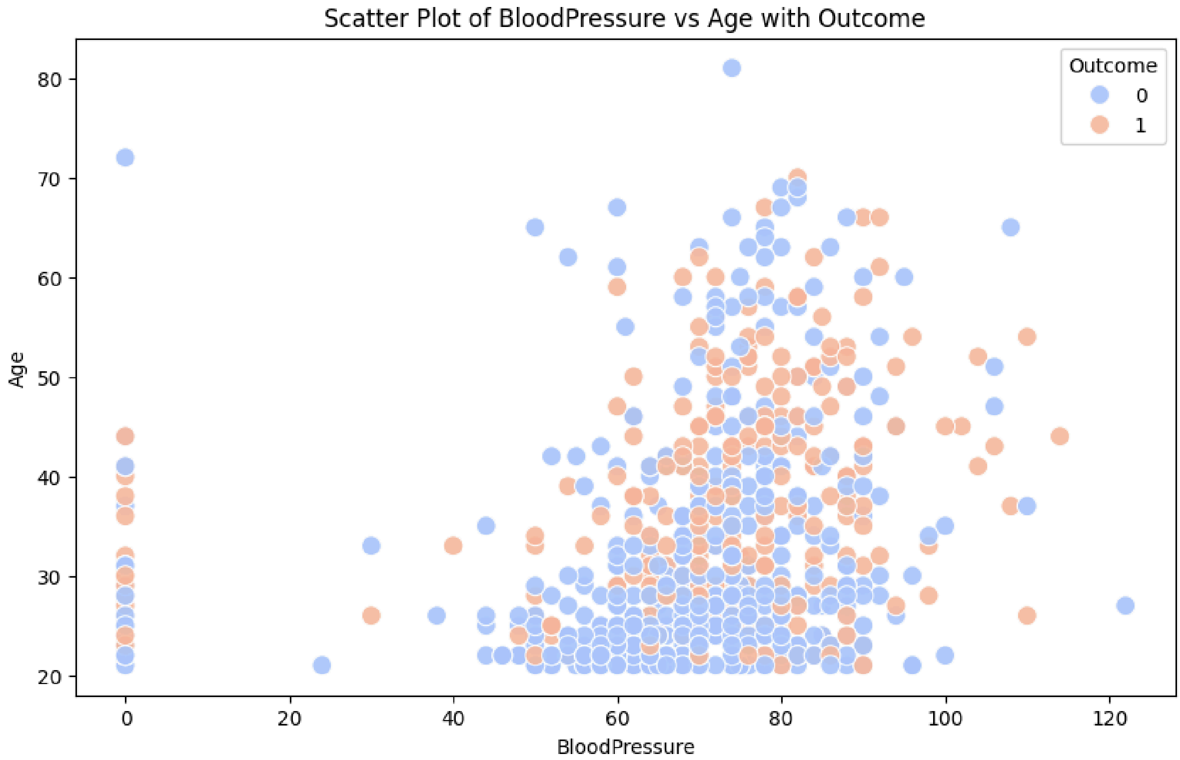
In [18]:
```
# Blood Pressure Distribution
sns.histplot(data=df,x='BloodPressure',kde=True,color='indigo')
plt.title('Distribution of BloodPressure')
plt.xlabel('BloodPressure')
plt.ylabel('Frequency')
plt.show()
```



In [19]:
```
# Scatter Plot of Glucose vs Age
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='Glucose',y='Age',hue='Outcome', palette='coolwarm', alp
plt.title('Scatter Plot of Glucose vs Age with Outcome')
plt.xlabel('Glucose')
plt.ylabel('Age')
plt.legend(title='Outcome')
plt.show()
```

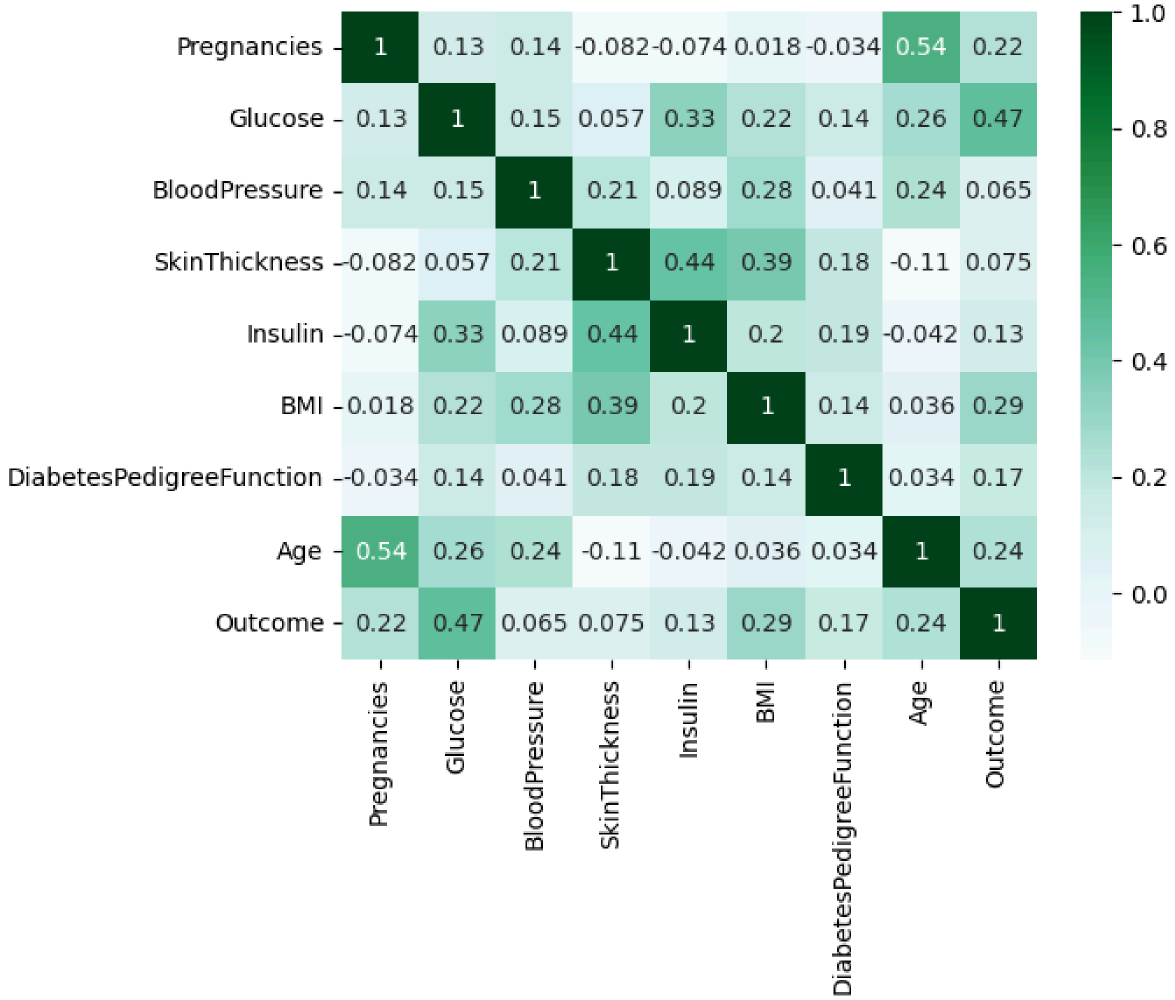


In [20]:
```
# Scatter Plot of BloodPressure vs Age
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='BloodPressure',y='Age',hue='Outcome', palette='coolwarm
plt.title('Scatter Plot of BloodPressure vs Age with Outcome')
plt.xlabel('BloodPressure')
plt.ylabel('Age')
plt.legend(title='Outcome')
plt.show()
```

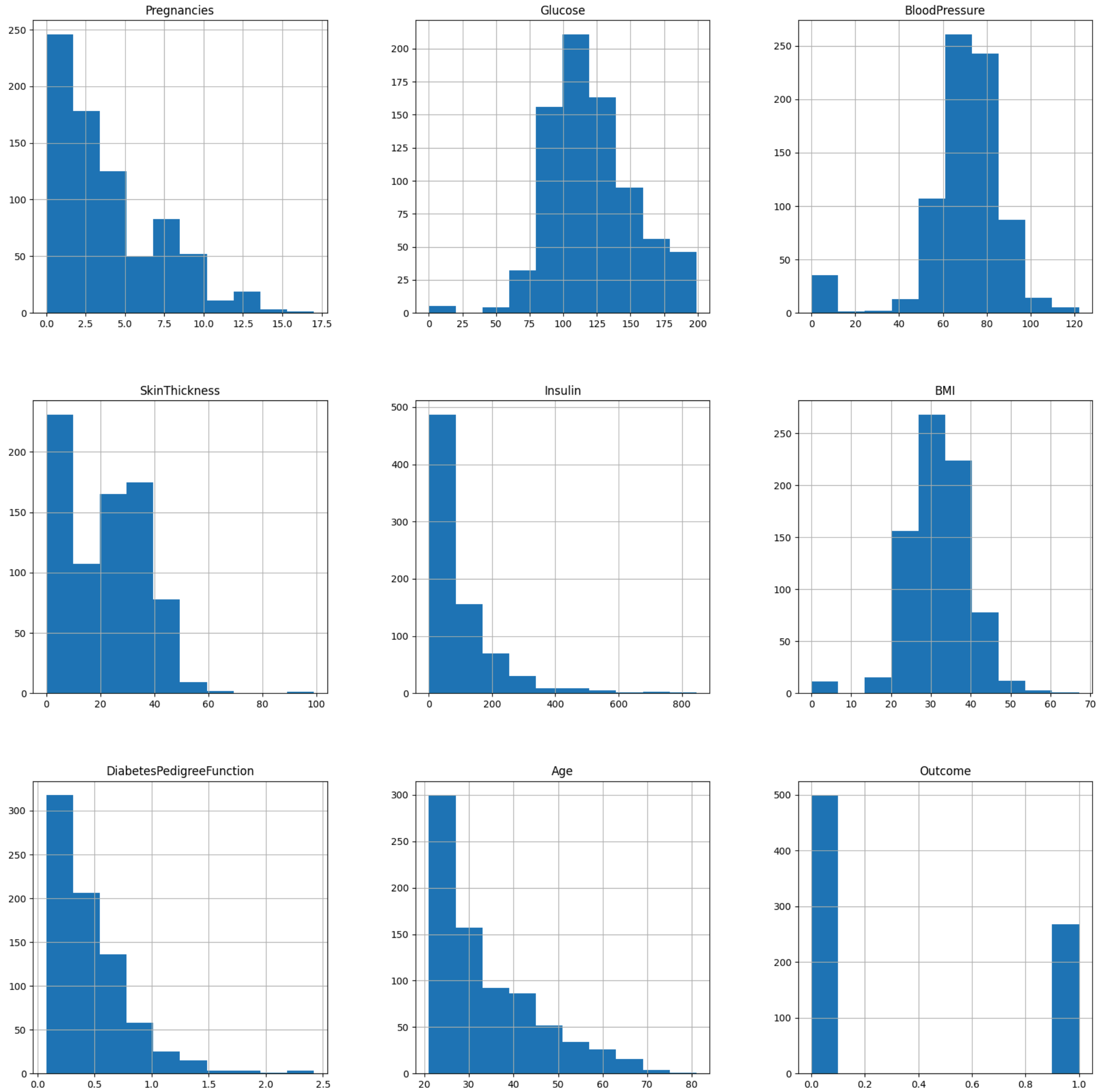


In [21]:
```
correl= df.corr()
sns.heatmap(correl,annot=True,cmap='BuGn')
```

Out[21]: `<Axes: >`



In [22]:
```
# Checking the outliers and Discrepancy in each numerical column
p=df.hist(figsize = (20,20))
```



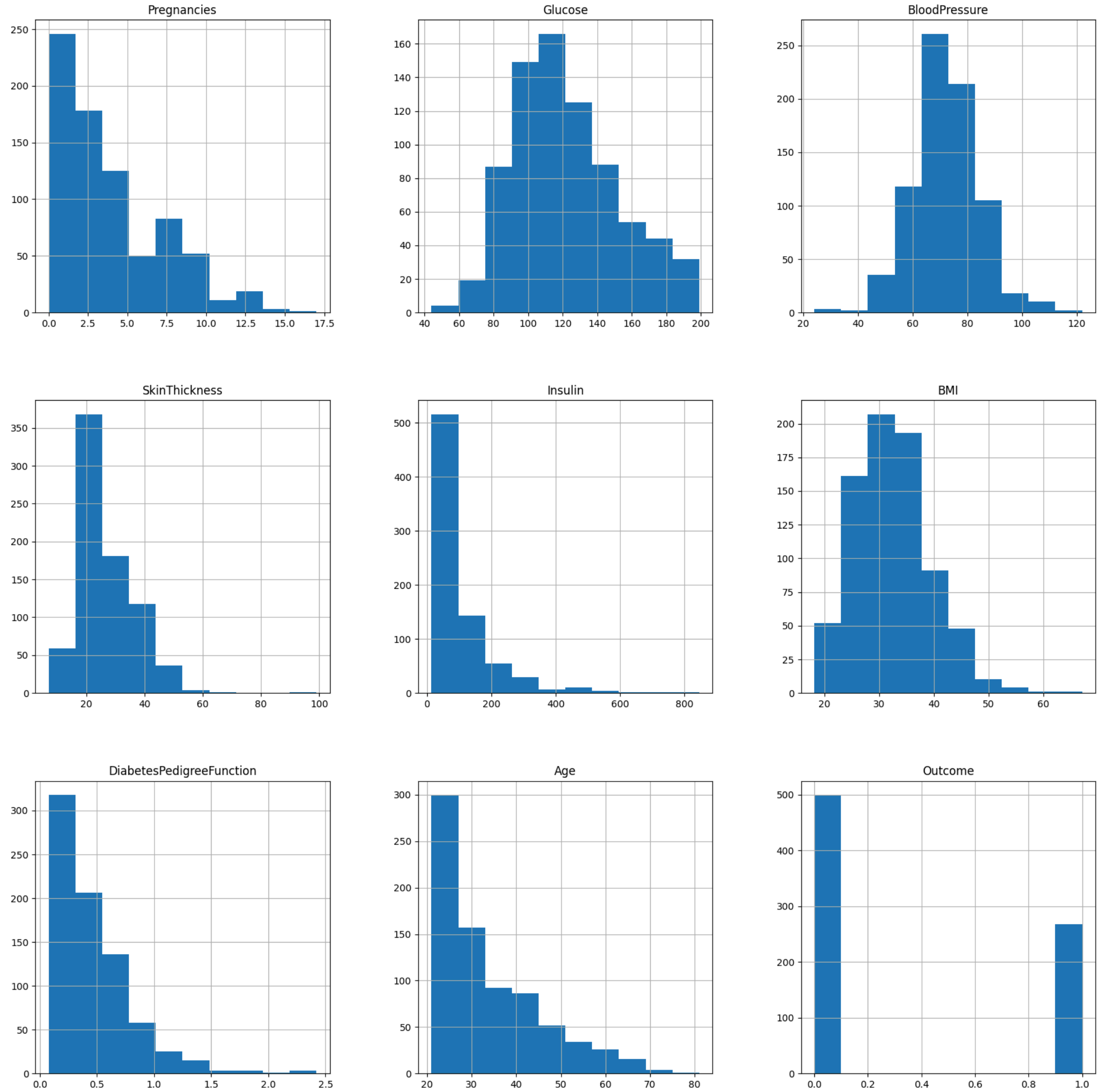
It appears that certain columns such as Glucose, Blood Pressure, Skin Thickness, Insulin, and BMI have zero values, which is inaccurate because Glucose levels, Blood Pressure, Insulin, and BMI cannot be measured as zero. It seems that the missing data is being represented as zero. It should be replaced with the mean of each column through imputation.

Imputation

In [23]:
```
# Replacing the zero with mean
col=['Glucose' ,'BloodPressure','SkinThickness', 'Insulin', 'BMI']
for i in col:
  df[i].replace(0,df[i].mean(),inplace=True)
```

In [24]:
```
df.hist(figsize = (20,20))
```

Out[24]:
```
array([[<Axes: title={'center': 'Pregnancies'}>,
        <Axes: title={'center': 'Glucose'}>,
        <Axes: title={'center': 'BloodPressure'}>],
       [<Axes: title={'center': 'SkinThickness'}>,
        <Axes: title={'center': 'Insulin'}>,
        <Axes: title={'center': 'BMI'}>],
       [<Axes: title={'center': 'DiabetesPedigreeFunction'}>,
        <Axes: title={'center': 'Age'}>,
        <Axes: title={'center': 'Outcome'}>]], dtype=object)
```



In [25]:
```
# Creating the dependent and independent variables from dataset df
X = df.drop(columns='Outcome',axis=1) # independent variable column
Y = df['Outcome'] # dependent variable column
```

In [26]:
```
print(X)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness     Insulin   BMI  \
0              6    148.0           72.0      35.000000   79.799479  33.6   
1              1     85.0           66.0      29.000000   79.799479  26.6   
2              8    183.0           64.0      20.536458   79.799479  23.3   
3              1     89.0           66.0      23.000000   94.000000  28.1   
4              0    137.0           40.0      35.000000  168.000000  43.1   
..           ...      ...            ...            ...         ...   ...   
763           10    101.0           76.0      48.000000  180.000000  32.9   
764            2    122.0           70.0      27.000000   79.799479  36.8   
765            5    121.0           72.0      23.000000  112.000000  26.2   
766            1    126.0           60.0      20.536458   79.799479  30.1   
767            1     93.0           70.0      31.000000   79.799479  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 [27]:

```python
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
```

Scaling

In [28]:

```python
# Importing library for Scaling
from sklearn.preprocessing import StandardScaler
```

In [29]:

```python
sc = StandardScaler()
X = sc.fit_transform(X)
```

In [30]:

```python
print(X)
```

```
[[ 0.63994726  0.86527574 -0.0210444  ...  0.16725546  0.46849198
   1.4259954 ]
 [-0.84488505 -1.20598931 -0.51658286 ... -0.85153454 -0.36506078
  -0.19067191]
 [ 1.23388019  2.01597855 -0.68176235 ... -1.33182125  0.60439732
  -0.10558415]
 ...
 [ 0.3429808  -0.02240928 -0.0210444  ... -0.90975111 -0.68519336
  -0.27575966]
 [-0.84488505  0.14197684 -1.01212132 ... -0.34213954 -0.37110101
   1.17073215]
 [-0.84488505 -0.94297153 -0.18622389 ... -0.29847711 -0.47378505
  -0.87137393]]
```

In [31]:
```
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
```

Train Test Split

In [32]:
```
# Import Train Test Split
from sklearn.model_selection import train_test_split
```

In [33]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2,random_state=
```

In [34]:
```
print(X.shape,X_train.shape,X_test.shape)
```

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

Training the Models

1. Logistic Regression

In [35]:
```
# Import libraries for Logistic Regression model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,accuracy_score
```

In [36]:
```
# Initilization of LogisticRegression
lr_model=LogisticRegression()
lr_model.fit(X_train,Y_train)
```

Out[36]:
```
▾ LogisticRegression
LogisticRegression()
```

In [37]:
```
# Prediction
Y_pred = lr_model.predict(X_test)

# Calculate accuracy
lr_accuracy = accuracy_score(Y_test, Y_pred)
print(f"Accuracy: {lr_accuracy:.2f}\n")

# Classification report
lr_report=classification_report(Y_test,Y_pred)
print("Classification Report:\n", lr_report)

# Confusion Matrix
lr_matrix=confusion_matrix(Y_test,Y_pred)
print("Confusion Matrix:\n", lr_matrix)
```

```
Accuracy: 0.77

Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.84      0.82        99
           1       0.69      0.64      0.66        55

    accuracy                           0.77       154
   macro avg       0.75      0.74      0.74       154
weighted avg       0.76      0.77      0.76       154

Confusion Matrix:
 [[83 16]
 [20 35]]
```

In [38]:
```
# score of Test dataset
lr_model.score(X_test,Y_test)
```

Out[38]:
```
0.7662337662337663
```

In [39]:
```
# Score of Train dataset
lr_model.score(X_train,Y_train)
```

Out[39]:
```
0.7703583061889251
```

2. Support Vector Machine Classifier

In [40]:
```
# Import Libraries for Support Vector Machine
from sklearn import svm
```

In [41]:
```
# Initialize SVC classifier
SVC_classifier = svm.SVC(kernel='linear')
```

In [42]:
```
# Train the model
SVC_classifier.fit(X_train, Y_train)

# Make predictions
Y_pred = SVC_classifier.predict(X_test)

# Calculate accuracy
SVC_accuracy = accuracy_score(Y_test, Y_pred)
print(f"Accuracy: {SVC_accuracy:.2f}\n")

# Classification report
```

```
SVC_report=classification_report(Y_test,Y_pred)
print("Classification Report:\n", SVC_report)

# Confusion Matrix
SVC_matrix=confusion_matrix(Y_test,Y_pred)
print("Confusion Matrix:\n", SVC_matrix)
```

```
Accuracy: 0.76

Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.83      0.82        99
           1       0.67      0.64      0.65        55

    accuracy                           0.76       154
   macro avg       0.74      0.73      0.74       154
weighted avg       0.76      0.76      0.76       154

Confusion Matrix:
 [[82 17]
 [20 35]]
```

In [43]:
```
# score of Test dataset
SVC_classifier.score(X_test,Y_test)
```

Out[43]:
```
0.7597402597402597
```

In [44]:
```
# Score of Train dataset
SVC_classifier.score(X_train,Y_train)
```

Out[44]:
```
0.7703583061889251
```

3. Random Forest Classifier

In [45]:
```
# Import Libraries for Random Forest Classification model
from sklearn.ensemble import RandomForestClassifier
```

In [46]:
```
# Initialized the random forest classifier
RFC_model=RandomForestClassifier()

# Train the classifier
RFC_model.fit(X_train, Y_train)

# Predict on the test set
Y_pred = RFC_model.predict(X_test)

# Evaluate the model accuracy
RFC_accuracy = accuracy_score(Y_test, Y_pred)
print(f"Accuracy: {RFC_accuracy:.2f}")

# classification report
RFC_report = classification_report(Y_test, Y_pred)
print("Classification Report:\n", RFC_report)

# confusion matrix
RFC_matrix = confusion_matrix(Y_test, Y_pred)
print("Confusion Matrix:\n", RFC_matrix)
```

```
Accuracy: 0.79
Classification Report:
               precision    recall  f1-score   support

           0       0.84      0.83      0.83        99
           1       0.70      0.71      0.70        55

    accuracy                           0.79       154
   macro avg       0.77      0.77      0.77       154
weighted avg       0.79      0.79      0.79       154

Confusion Matrix:
 [[82 17]
 [16 39]]
```

In [47]:
```
# score of Test dataset
RFC_model.score(X_test,Y_test)
```

Out[47]:
```
0.7857142857142857
```

In [48]:
```
# Score of Train dataset
RFC_model.score(X_train,Y_train)
```

Out[48]:
```
1.0
```

4. DecisionTreeClassifier

In [49]:
```
# Import libraries for DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier
```

In [50]:
```
# Initialized the DecisionTreeClassifier
dtc = DecisionTreeClassifier(criterion='entropy',max_depth=5)

# Train the classifier
dtc.fit(X_train, Y_train)

# Predict on the test set
Y_pred = dtc.predict(X_test)

# Evaluate the model accuracy
dtc_accuracy = accuracy_score(Y_test, Y_pred)
print(f"Accuracy: {dtc_accuracy:.2f}")

# classification report
dtc_report = classification_report(Y_test, Y_pred)
print("Classification Report:\n", dtc_report)

# confusion matrix
dtc_matrix = confusion_matrix(Y_test, Y_pred)
print("Confusion Matrix:\n", dtc_matrix)
```

```
Accuracy: 0.76
Classification Report:
               precision    recall  f1-score   support

           0       0.82      0.80      0.81        99
           1       0.66      0.69      0.67        55

    accuracy                           0.76       154
   macro avg       0.74      0.74      0.74       154
weighted avg       0.76      0.76      0.76       154

Confusion Matrix:
 [[79 20]
 [17 38]]
```

In [51]:
```
# score of Test dataset
dtc.score(X_test,Y_test)
```

Out[51]:
```
0.7597402597402597
```

In [52]:
```
# Score of Train dataset
dtc.score(X_train,Y_train)
```

Out[52]:
```
0.8159609120521173
```

5. KNearestNeighbors

In [53]:
```
# Import libraries for KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [54]:
```
# Initialized the KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=9)                #knn classifier

# Train the classifier
knn.fit(X_train,Y_train)

# Predict on the test set
Y_pred = knn.predict(X_test)

# Evaluate the model accuracy
knn_accuracy = accuracy_score(Y_test, Y_pred)
print(f"Accuracy: {knn_accuracy:.2f}")

# classification report
knn_report = classification_report(Y_test, Y_pred)
print("Classification Report:\n", knn_report)

# confusion matrix
knn_matrix = confusion_matrix(Y_test, Y_pred)
print("Confusion Matrix:\n", knn_matrix)
```

```
Accuracy: 0.75
Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.83      0.81        99
           1       0.66      0.60      0.63        55

    accuracy                           0.75       154
   macro avg       0.72      0.71      0.72       154
weighted avg       0.74      0.75      0.74       154

Confusion Matrix:
 [[82 17]
 [22 33]]
```

In [55]:

```
# score of Test dataset
knn.score(X_test,Y_test)
```

Out[55]:

```
0.7467532467532467
```

In [56]:

```
# Score of Train dataset
knn.score(X_train,Y_train)
```

Out[56]:

```
0.8013029315960912
```

6. GradientBoostingClassifier

In [57]:

```
# Import libraries for GradientBoostingClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

In [58]:

```
# Initialized the GradientBoostingClassifier
gbc = GradientBoostingClassifier()

# Train the classifier
gbc.fit(X_train,Y_train)

# Predict on the test set
Y_pred = gbc.predict(X_test)

# Evaluate the model accuracy
gbc_accuracy = accuracy_score(Y_test, Y_pred)
print(f"Accuracy: {gbc_accuracy:.2f}")

# classification report
gbc_report = classification_report(Y_test, Y_pred)
print("Classification Report:\n", gbc_report)

# confusion matrix
gbc_matrix = confusion_matrix(Y_test, Y_pred)
print("Confusion Matrix:\n", gbc_matrix)
```

```
Accuracy: 0.77
Classification Report:
               precision    recall  f1-score   support

           0       0.84      0.79      0.81        99
           1       0.66      0.73      0.69        55

    accuracy                           0.77       154
   macro avg       0.75      0.76      0.75       154
weighted avg       0.77      0.77      0.77       154

Confusion Matrix:
 [[78 21]
 [15 40]]
```

In [59]:
```
# score of Test dataset
gbc.score(X_test,Y_test)
```

Out[59]:
```
0.7662337662337663
```

In [60]:
```
# Score of Train dataset
gbc.score(X_train,Y_train)
```

Out[60]:
```
0.9348534201954397
```

7. XGBClassifier

In [61]:
```
# Import libraries for XGBClassifier
from xgboost import XGBClassifier
```

In [62]:
```
# Initialized the XGBClassifier
xgb = XGBClassifier(booster = 'gbtree', learning_rate = 0.1, max_depth=6,n_estimato

# Train the classifier
xgb.fit(X_train,Y_train)

# Predict on the test set
Y_pred = xgb.predict(X_test)

# Evaluate the model accuracy
xgb_accuracy = accuracy_score(Y_test, Y_pred)
print(f"Accuracy: {xgb_accuracy:.2f}")

# classification report
xgb_report = classification_report(Y_test, Y_pred)
print("Classification Report:\n", xgb_report)

# confusion matrix
xgb_matrix = confusion_matrix(Y_test, Y_pred)
print("Confusion Matrix:\n", xgb_matrix)
```

```
Accuracy: 0.78
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.88      0.84        99
           1       0.73      0.60      0.66        55

    accuracy                           0.78       154
   macro avg       0.77      0.74      0.75       154
weighted avg       0.78      0.78      0.77       154

Confusion Matrix:
 [[87 12]
 [22 33]]
```

In [63]:
```
# score of Test dataset
xgb.score(X_test,Y_test)
```

Out[63]:
```
0.7792207792207793
```

In [64]:
```
# Score of Train dataset
xgb.score(X_train,Y_train)
```

Out[64]:
```
0.8859934853420195
```

Model Comparison

In [65]:
```
m ={
    'Model':['Logistic Regression','SVC','Random Forest','DecisionTreeClassifier','
    'Score':[lr_accuracy, SVC_accuracy, RFC_accuracy, dtc_accuracy, knn_accuracy, g
}
```

In [66]:
```
models = pd.DataFrame(m)
models_sorted=models.sort_values(by='Score', ascending=False)
models_sorted
```

Out[66]:

	Model	Score
2	Random Forest	0.785714
6	XGBClassifier	0.779221
0	Logistic Regression	0.766234
5	GradientBoostingClassifier	0.766234
1	SVC	0.759740
3	DecisionTreeClassifier	0.759740
4	KNearestNeighbors	0.746753

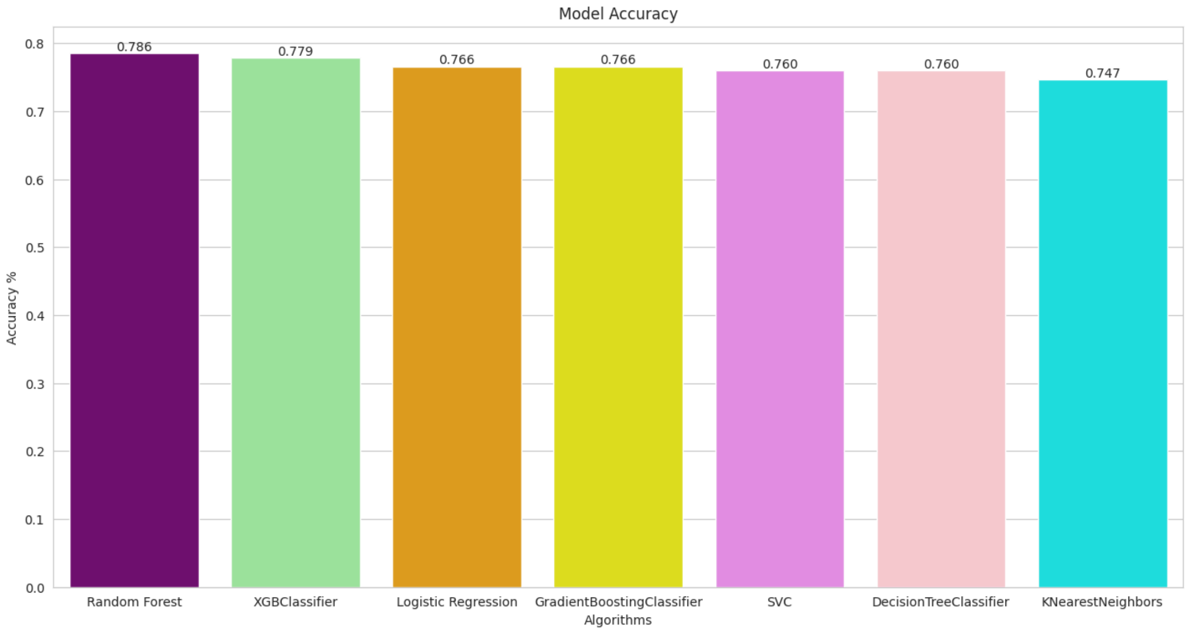
In [67]:
```
colors = ["purple", "lightgreen", "orange", "yellow","violet","pink","cyan"]

sns.set_style("whitegrid")
plt.figure(figsize=(16,8))
# Create the bar plot
ax = sns.barplot(x=models_sorted['Model'], y=models_sorted['Score'], palette=colors

# Add data labels
for i, score in enumerate(models_sorted['Score']):
```

```
    ax.text(i, score, f'{score:.3f}', ha='center', va='bottom')

# Add labels and title
plt.ylabel("Accuracy %")
plt.xlabel("Algorithms")
plt.title("Model Accuracy")
plt.show()
```



Making a predictive system

In [69]:

```
input_data = (11,143,94,33,146,36.6,0.254,51)

# changing the input_data into numpy array
input_data_array = np.array(input_data)

# reshape the array
input_data_reshaped = input_data_array.reshape(1,-1)

# standardize the input data
std_data = sc.transform(input_data_reshaped)

print(std_data)

# Prediction
prediction = RFC_model.predict(std_data)
print(prediction)

if (prediction[0] == 0):
    print("The Person is non-diabetic")
else:
    print("The Person is diabetic")
```

```
[[ 2.12477957  0.70088963  1.79592994  0.66426408  0.29391436  0.60387974
  -0.65801229  1.51108316]]
[1]
The Person is diabetic
```

In [ ]:

In [ ]: