Importing Libraries

In [1]:

```python
import numpy as np
import warnings
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
warnings.filterwarnings('ignore')
```

In [2]:

```python
data=pd.read_csv(r"C:\Users\Siddhi\Desktop\diabetes.csv")
```

In [3]:

```python
data
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [4]:

```python
data.shape
```

Out[4]: (768, 9)

In [5]:

```python
data.head()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [6]:

```python
data.columns
```

Out[6]:

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [7]:

```python
data.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 [8]:

```python
data.describe()
```

Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	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	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

In [9]:

```python
data.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

In [10]:

```python
data.isnull()
```

Out[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
763	False	False	False	False	False	False	False	False	False
764	False	False	False	False	False	False	False	False	False
765	False	False	False	False	False	False	False	False	False
766	False	False	False	False	False	False	False	False	False
767	False	False	False	False	False	False	False	False	False

768 rows × 9 columns

In [11]:

```python
data.isnull().sum()
```

Out[11]:

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [12]:

```python
data.Outcome.value_counts()
```

Out[12]:

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

In [13]:

```python
data["Outcome"].value_counts()*100/len(data)
```

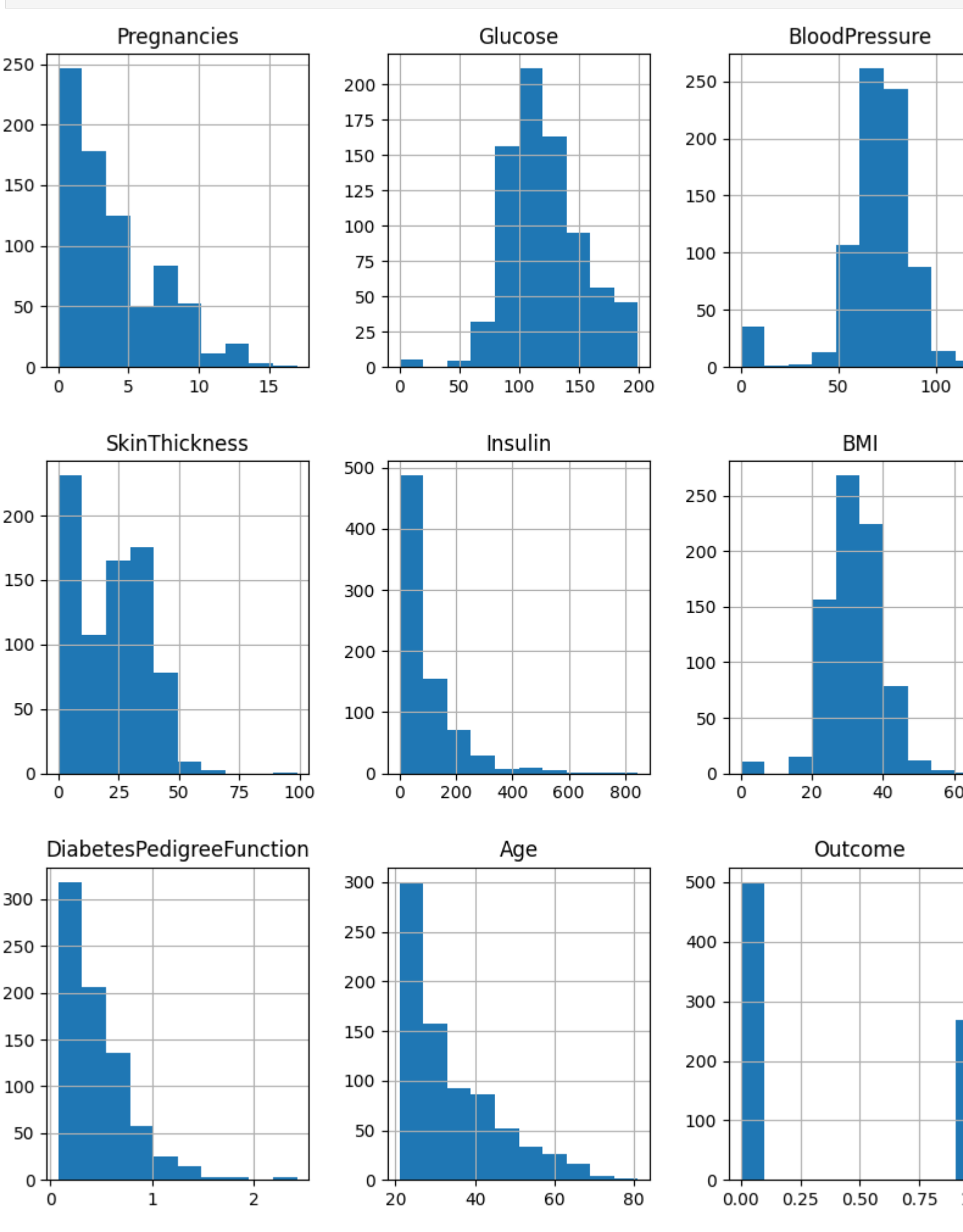
Out[13]:

```
Outcome
0    65.104167
1    34.895833
Name: count, dtype: float64
```

EDA

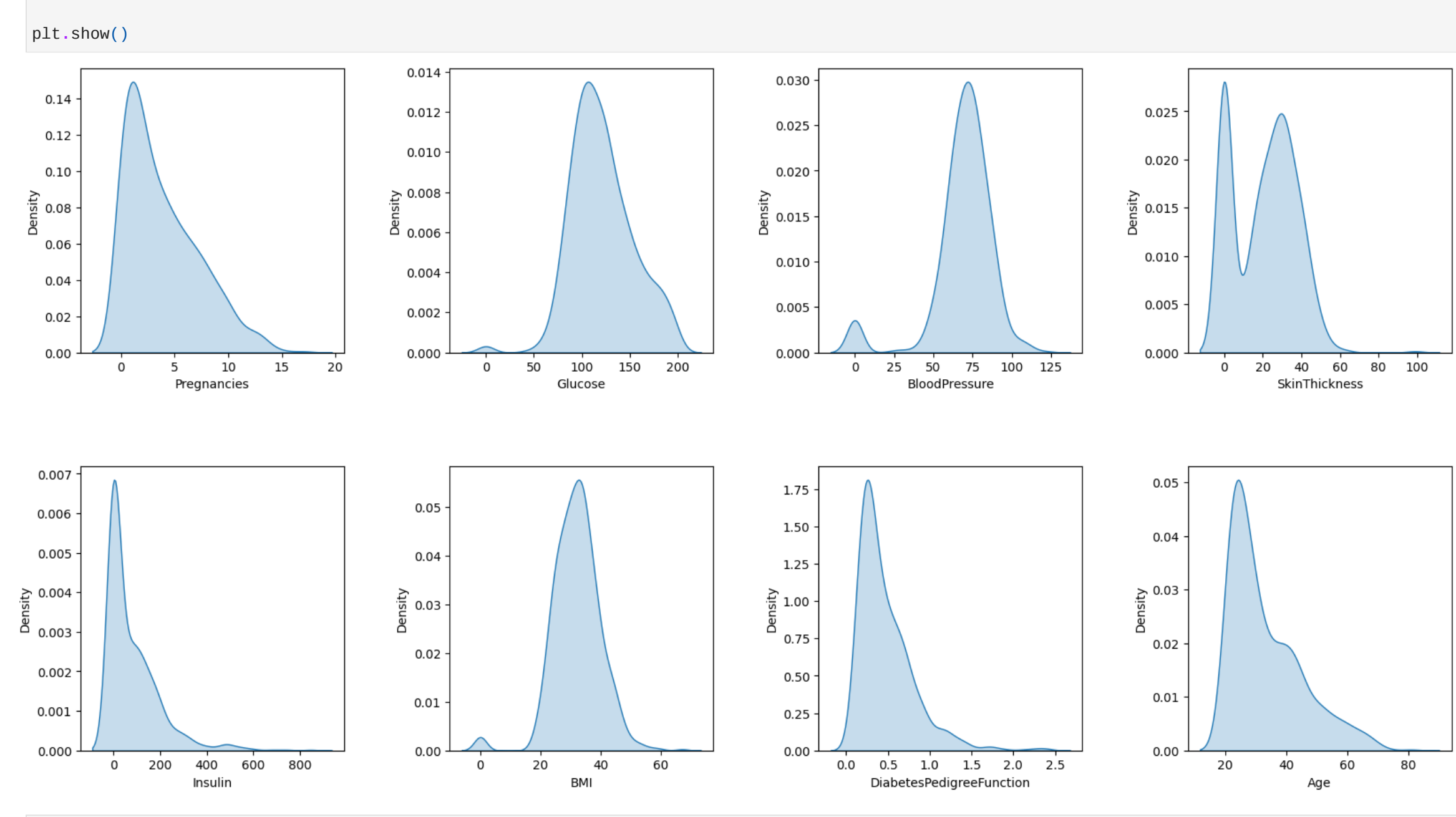
In [14]:

```python
data.hist(figsize=(10,12))
plt.show()
```



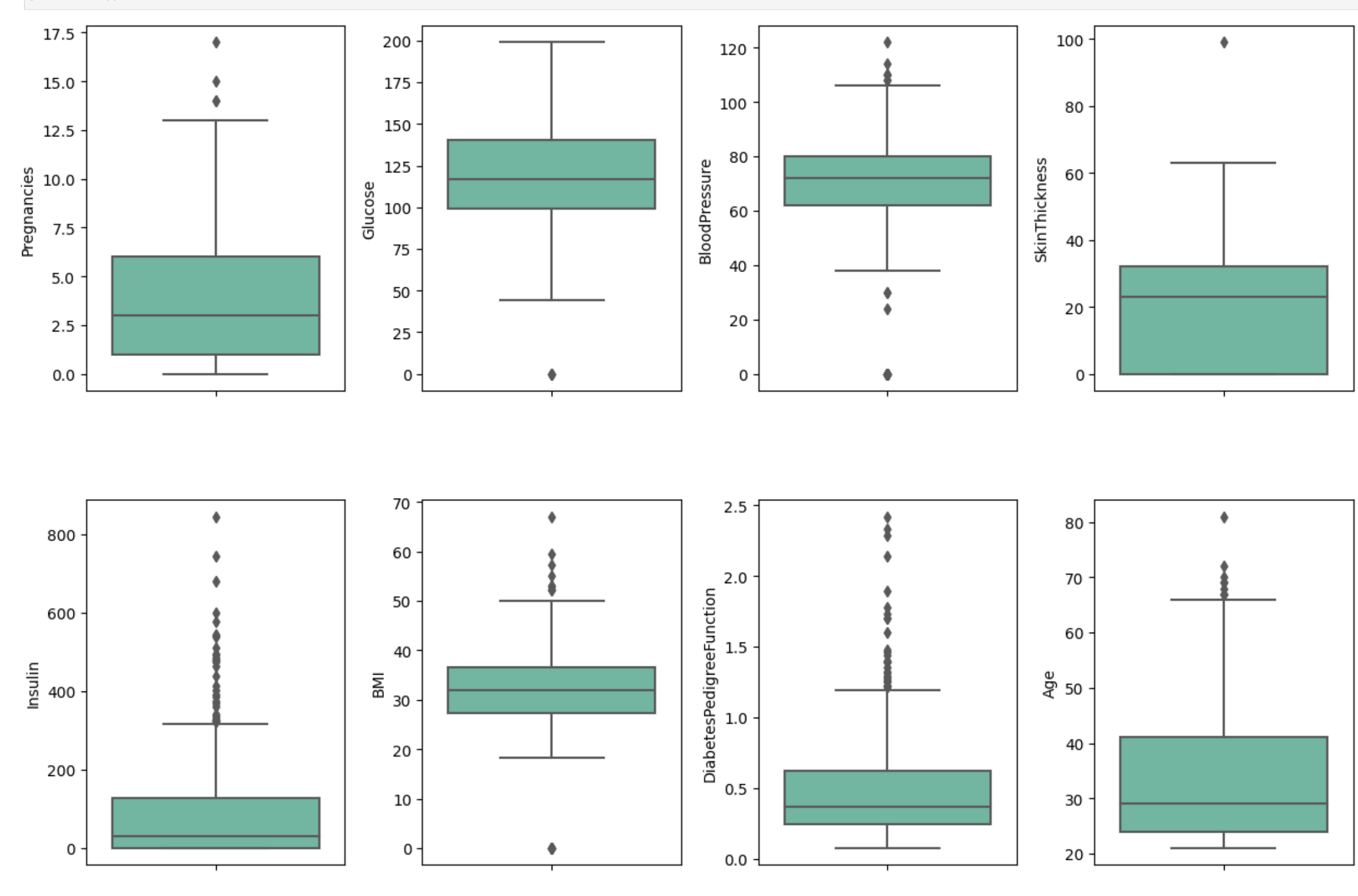
In [15]:

```python
# Visualizing Kernel Density Estimator for each feature
features = data.columns[:-1]

fig, axes = plt.subplots(2, 4, figsize=(20, 10))
fig.subplots_adjust(wspace=0.4, hspace=0.4)

for i, feature in enumerate(features):
    sns.kdeplot(data[feature], ax=axes[i//4, i%4],shade='fill')

plt.show()
```



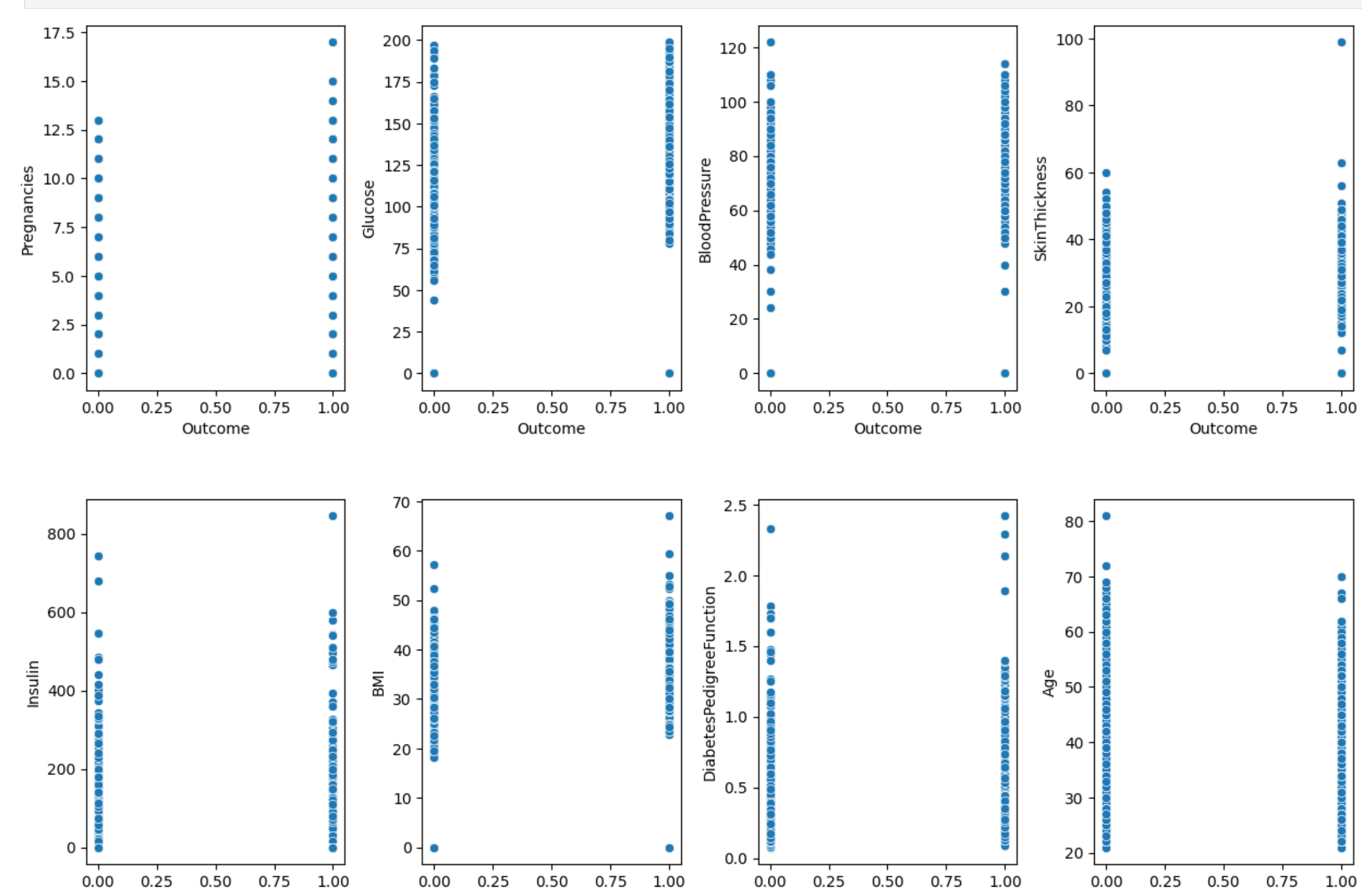
In [16]:

```python
# Visualizing Boxplot of each feature
fig, axes = plt.subplots(2, 4, figsize=(15, 10))
fig.subplots_adjust(wspace=0.3, hspace=0.3)

for i, feature in enumerate(features):
    sns.boxplot(y=data[feature], ax=axes[i//4, i%4],palette='Set2')

plt.show()
```



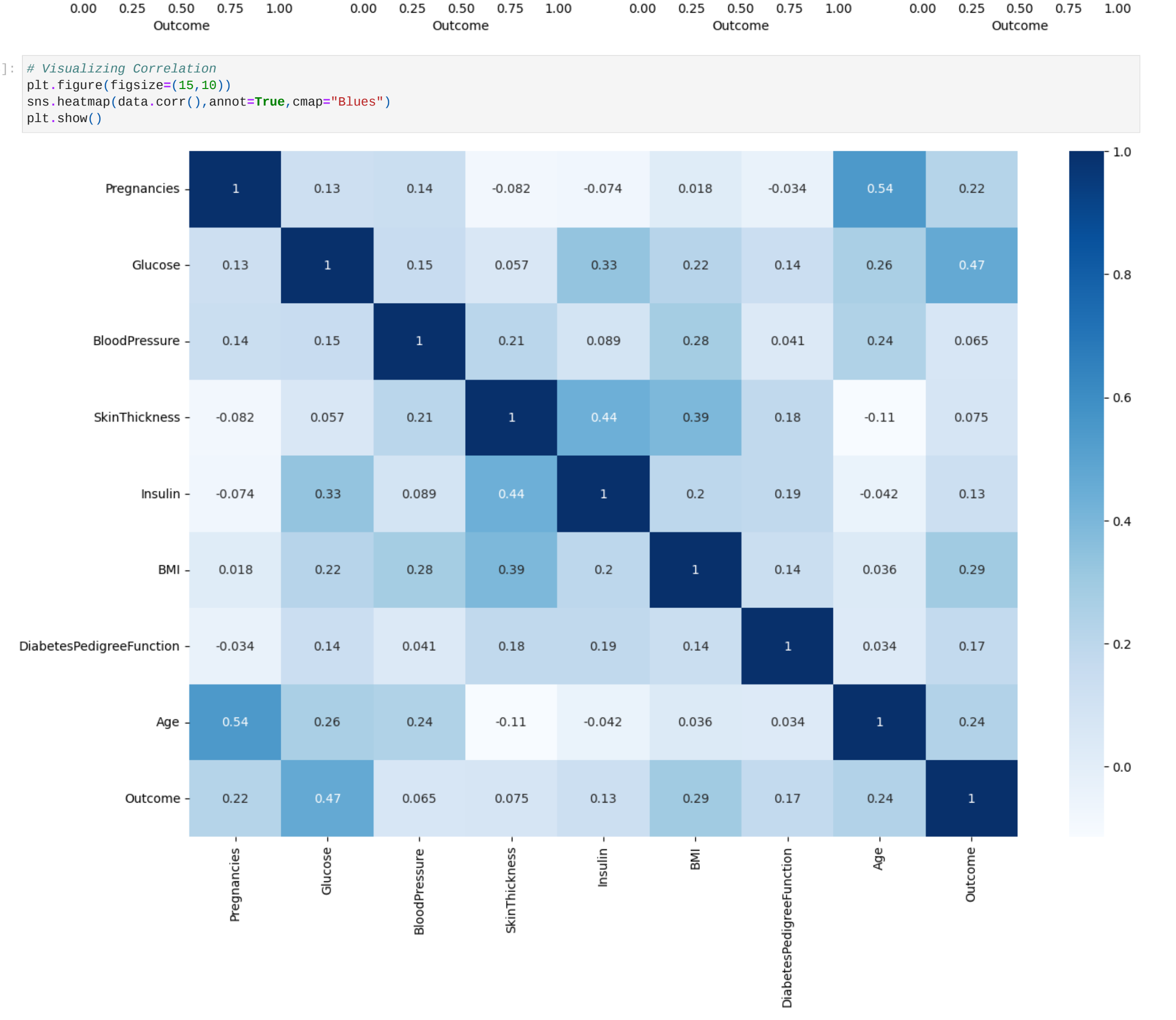
In [17]:

```python
# Visualizing features relationship with outcome
fig, axes = plt.subplots(2, 4, figsize=(15, 10))
fig.subplots_adjust(wspace=0.3, hspace=0.3)

for i, feature in enumerate(features):
    sns.scatterplot(y=data[feature], x=data['Outcome'], ax=axes[i//4, i%4],palette='Set1')

plt.show()
```



In [18]:

```python
# Visualizing Correlation
plt.figure(figsize=(15,10))
sns.heatmap(data.corr(),annot=True,cmap="Blues")
plt.show()
```



Data Preprocessing

In [19]:

```python
# splitting the data into features and outcomes
X=data.drop('Outcome',axis=1)
y=data['Outcome']
```

In [20]:

```python
# train test split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

In [21]:

```python
print(f"Shape of X_train - {X_train.shape}")
print(f"Shape of y_train - {y_train.shape}")
print(f"Shape of X_test- {X_test.shape}")
print(f"Shape of y_test - {y_test.shape}")
```

```
Shape of X_train - (614, 8)
Shape of y_train - (614,)
Shape of X_test- (154, 8)
Shape of y_test - (154,)
```

Model Training

In [22]:

```python
models={
    "Logistics Regression":LogisticRegression(),
    "Support Vector Classifier":SVC(),
    "Decision Tree Classifier":DecisionTreeClassifier(),
    "Random Forest Classifier":RandomForestClassifier(),
    "K-nearest Neighbous":KNeighborsClassifier()
}

for name,model in models.items():
    clf=model
    clf.fit(X_train,y_train)
    y_pred=clf.predict(X_test)

    acc=accuracy_score(y_test,y_pred)
    print(f"Accuracy of {name} - {acc}")
```

```
Accuracy of Logistics Regression - 0.7467532467532467
Accuracy of Support Vector Classifier - 0.7662337662337663
Accuracy of Decision Tree Classifier - 0.7402597402597403
Accuracy of Random Forest Classifier - 0.7272727272727273
Accuracy of K-nearest Neighbous - 0.6623376623376623
```

Hyperparameter Tuning

1. Logistics Regression

In [23]:

```python
### Create a Logistic Regression model
logistic_regression = LogisticRegression()

# Define a parameter grid to search over
param_grid = {
    'penalty': ['l1', 'l2'],  # Regularization penalty
    'C': [0.001, 0.01, 0.1, 1, 10, 100],  # Inverse of regularization strength
    'solver': ['liblinear', 'lbfgs', 'newton-cg', 'sag', 'saga'],  # Solver algorithm
}

# Create a GridSearchCV object with cross-validation
grid_search = GridSearchCV(logistic_regression, param_grid, cv=5, scoring='accuracy')

# Perform the grid search to find the best hyperparameters
grid_search.fit(X, y)

# Print the best hyperparameters and the corresponding accuracy
best_params = grid_search.best_params_
best_accuracy = grid_search.best_score_
print(f"Best Hyperparameters: {best_params}")
print(f"Best Accuracy: {best_accuracy}")
```

```
Best Hyperparameters: {'C': 1, 'penalty': 'l2', 'solver': 'newton-cg'}
Best Accuracy: 0.7721925133689839
```

2. Decision Tree Classifier

In [24]:

```python
# Define the parameter grid to search
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create a Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Create Grid Search with 5-fold cross-validation
grid_search = GridSearchCV(dt_classifier, param_grid, cv=5, scoring='accuracy')

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best parameters and the best estimator
best_params = grid_search.best_params_
best_dt_classifier = grid_search.best_estimator_

# Fit the best classifier on the full training set
best_dt_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = best_dt_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Best Parameters: {best_params}')
print(f'Accuracy on Test Set: {accuracy}')
```

```
Best Parameters: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 5}
Accuracy on Test Set: 0.7597402597402597
```

3. Random Forest Classifier

In [ ]:

```python
# Create a Random Forest Classifier
rf_classifier = RandomForestClassifier()

# Define the hyperparameter grid to search
param_grid = {
    'n_estimators': [10, 50, 100, 200],  # Number of trees in the forest
    'max_depth': [None, 10, 20, 30],    # Maximum depth of the trees
    'min_samples_split': [2, 5, 10],   # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4]      # Minimum number of samples required to be at a leaf node
}

# Create a GridSearchCV object with cross-validation
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X, y)

# Print the best hyperparameters and their corresponding accuracy score
best_params = grid_search.best_params_
best_score = grid_search.best_score_
print("Best Hyperparameters:", best_params)
print("Accuracy on Test Data:", best_score)
```

Conclusion

Random Classifer has best accuracy score(0.7878023936847467)

Thank You