Project Report

Prediction of diabetes using machine learning algorithms

Prepared By:

Pradnya Jagtap	86072300005	A005
Yash Khurana	86072300023	A023
Vaishnavi Seth	86072300027	A027
Vicky Dhaygude	86072300045	A045
Poorvi Singh Samant	86072300063	A063

Abstract:

This project focuses on conducting exploratory data analysis (EDA) and developing predictive models for Diabetes classification. The dataset used in this study contains various features related to the hospitality industry, such as customer ratings, amenities, and location information. The primary objective is to gain insights into the dataset through data visualization and preprocessing techniques, followed by the development and evaluation of classification models using machine learning algorithms.

The project begins with loading the dataset into a Pandas DataFrame and exploring its structure and summary statistics. Data visualization techniques, including histograms and box plots, are employed to understand the distribution of features and identify outliers. Outliers are treated using the winsorization technique, and standard scaling is applied to ensure uniformity in feature scales.

Subsequently, three classification models—Logistic Regression, Random Forest Classifier, and K-Nearest Neighbors (KNN) Classifier—are trained and evaluated using the preprocessed data. Evaluation metrics such as accuracy, precision, recall, and confusion matrix are calculated for each model to assess their performance. The results are compared to determine the most suitable model for the dataset.

Background and Objective:

Diabetes is a life-threatening chronic disease with a growing global prevalence, necessitating early diagnosis and treatment to prevent severe complications. Machine learning has emerged as a promising approach for diabetes diagnosis, but challenges such as limited labeled data, frequent missing values, and dataset imbalance hinder the development of accurate prediction models. Therefore, a novel framework is required to address these challenges and improve performance.

Introduction:

Chronic diseases are long-lasting illnesses that can impact your quality of life. Diabetes is one such disease, causing high blood sugar due to lack of insulin from the pancreas. This can lead to problems like thirst, hunger, and kidney disease. There are two main types of diabetes: type 1 and type 2. Type 1 usually affects younger people and type 2 affects middle-aged and older people. While there's no cure for diabetes, it can be controlled with early detection and proper treatment. This is why predicting diabetes is an important area of research

Data Description:

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict based on diagnostic measurements whether a patient has diabetes. Several constraints were placed on the selection of these instances from a larger database. All patients here are females at least 21 years old of Pima Indian heritage.

- Pregnancies: Number of times pregnant
- Glucose: Plasma glucose concentration 2 hours in an oral glucose tolerance test
- Blood Pressure: Diastolic blood pressure (mm Hg)
- Skin Thickness: 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)

Snapshot of data:

```
Data.head()
```

	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

Approach :

- Data Loading: The dataset is loaded into a Pandas DataFrame using the read_csv function.
- Data Exploration: The structure and summary statistics of the dataset are examined using functions like head, describe, and is null. The correlation between different variables is visualized using a heatmap to understand the relationships within the data.
- Data Visualization:
  - The distribution of each feature in the dataset is visualized using histograms to understand their patterns and characteristics.
  - Box plots are used to identify outliers in the data, which can potentially affect the model's performance.
- Data Preprocessing: Outliers are treated using winsorization technique to mitigate their impact on the model. Standard scaling is applied to ensure that all variables are on the same scale, preventing features with larger magnitudes from dominating the model.
- Model Development: Three classification models are trained and evaluated using the preprocessed data:
  - ❑ **Logistic Regression**
  - ❑ **Random Forest Classifier**
  - ❑ **K-Nearest Neighbors (KNN) Classifier**

Evaluation metrics such as accuracy, precision, recall, and confusion matrix are calculated for each model to assess their performance.

- Model Comparison: The performance of each model is compared based on their evaluation metrics to determine the most suitable model for the dataset.

Results:

Logistic Regression Model:

```
Accuracy: 0.725609756097561
Confusion Matrix:
 [[260  45]
 [ 90  97]]
Precision: 0.6830985915492958
Recall: 0.5187165775401069
```

Random Forest Classifier:

```
Accuracy: 0.8475609756097561
Confusion Matrix:
 [[291  14]
 [ 61 126]]
Precision: 0.9
Recall: 0.6737967914438503
```

K-Nearest Neighbors (KNN) Classifier:

```
Accuracy: 0.9227642276422764
Confusion Matrix:
 [[286  19]
 [ 19 168]]
Precision: 0.8983957219251337
Recall: 0.8983957219251337
```

Conclusion:

Based on the evaluation metrics, the **"K-Nearest Neighbors (KNN) Classifier model"** demonstrates superior performance for the given dataset. Further optimization and tuning of the selected model can be performed to enhance its predictive accuracy.

Codes:

```
[ ]: import pandas as pd
     import matp frlotlib.pyplot as plt
     import seaborn as sns
     import numpy as np
```

```
[ ]: # Reading the datsaet
     Data=pd.read_csv("/content/Training.csv")
```

```
[ ]: Data.head()
```

```
[ ]:    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

        DiabetesPedigreeFunction Age  Outcome
     0     0.627 50     1  1    0.351 31    0  2
           0.672 32     1
     3                     0.167   21        0

    max     17.000000  197.000000   122.000000     63.000000 846.000000




     4                     2.288   33        1
```

```
[ ]: # Splitting the data into X and Y
     Y,X  = Data.iloc[:,-1:],Data.iloc[:,:-1]
```

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

```
[ ]:        Pregnancies      Glucose  BloodPressure  SkinThickness      Insulin  \
     count  2460.000000  2460.000000  2460.000000   2460.000000
                                                    2460.000000
     mean      3.817480  121.602033     68.915041     20.531301    80.119919
     std       3.296458   31.789270     19.082655     15.716901   116.765807
     min       0.000000    0.000000      0.000000      0.000000     0.000000
     25%       1.000000  100.000000     64.000000      0.000000     0.000000
     50%       3.000000  117.000000     70.000000     23.000000    36.000000
     75%       6.000000  142.000000     80.000000     33.000000   129.000000
```

```
               BMI  DiabetesPedigreeFunction          Age
count  2460.000000               2460.000000  2460.000000
mean     31.990447                  0.491440    32.821951
std       7.802569                  0.363917    11.251208
min       0.000000                  0.078000    21.000000
25%      27.100000                  0.251750    24.000000
50%      32.100000                  0.381000    29.000000
75%      36.500000                  0.647000    39.000000
max      67.100000                  2.420000    81.000000
```

[ ]:
```
Y.head()
```

[ ]:
```
   Outcome
0        1
1        0
2        1
3        0
4        1
```

[ ]:
```
# Checking for the missing values(No null values present in data set)
X.isnull().sum()
```

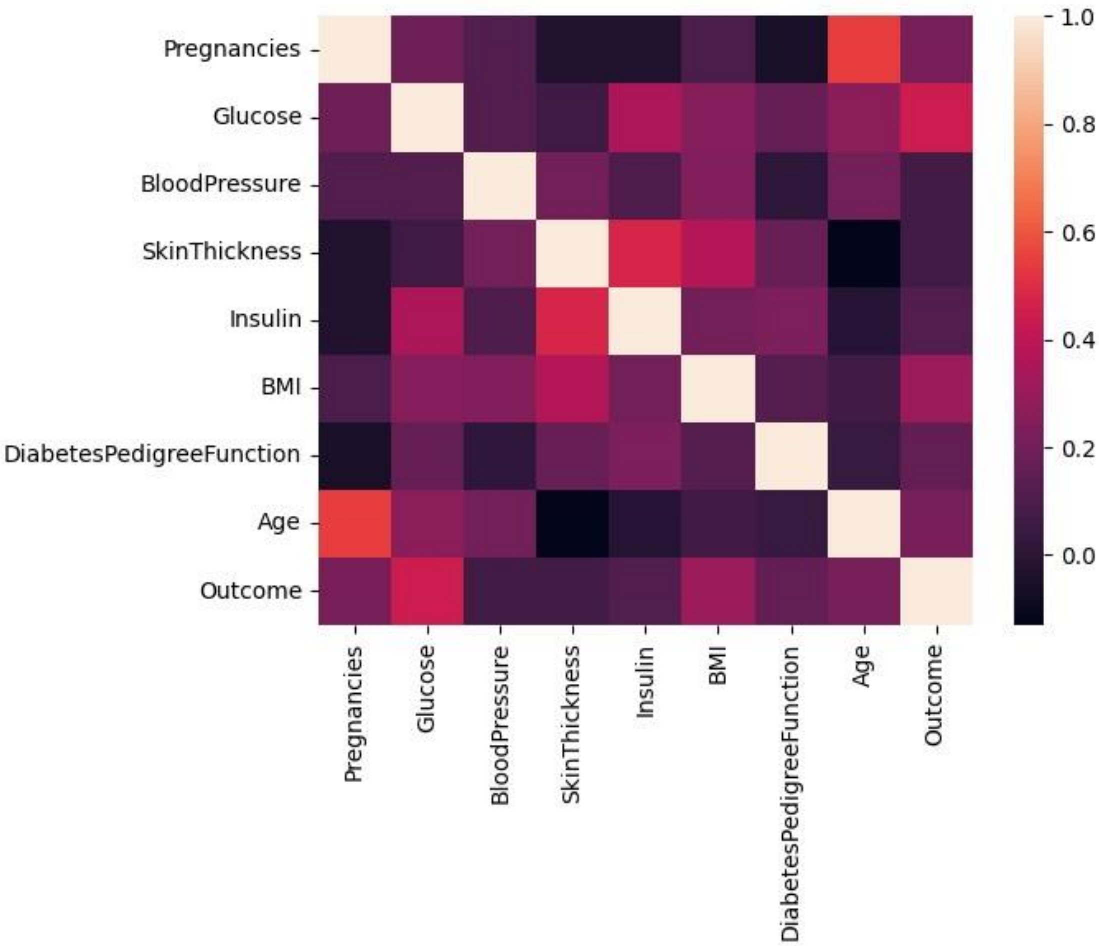
```
Total Missing Values:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age dtype:                  0
int64
```

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

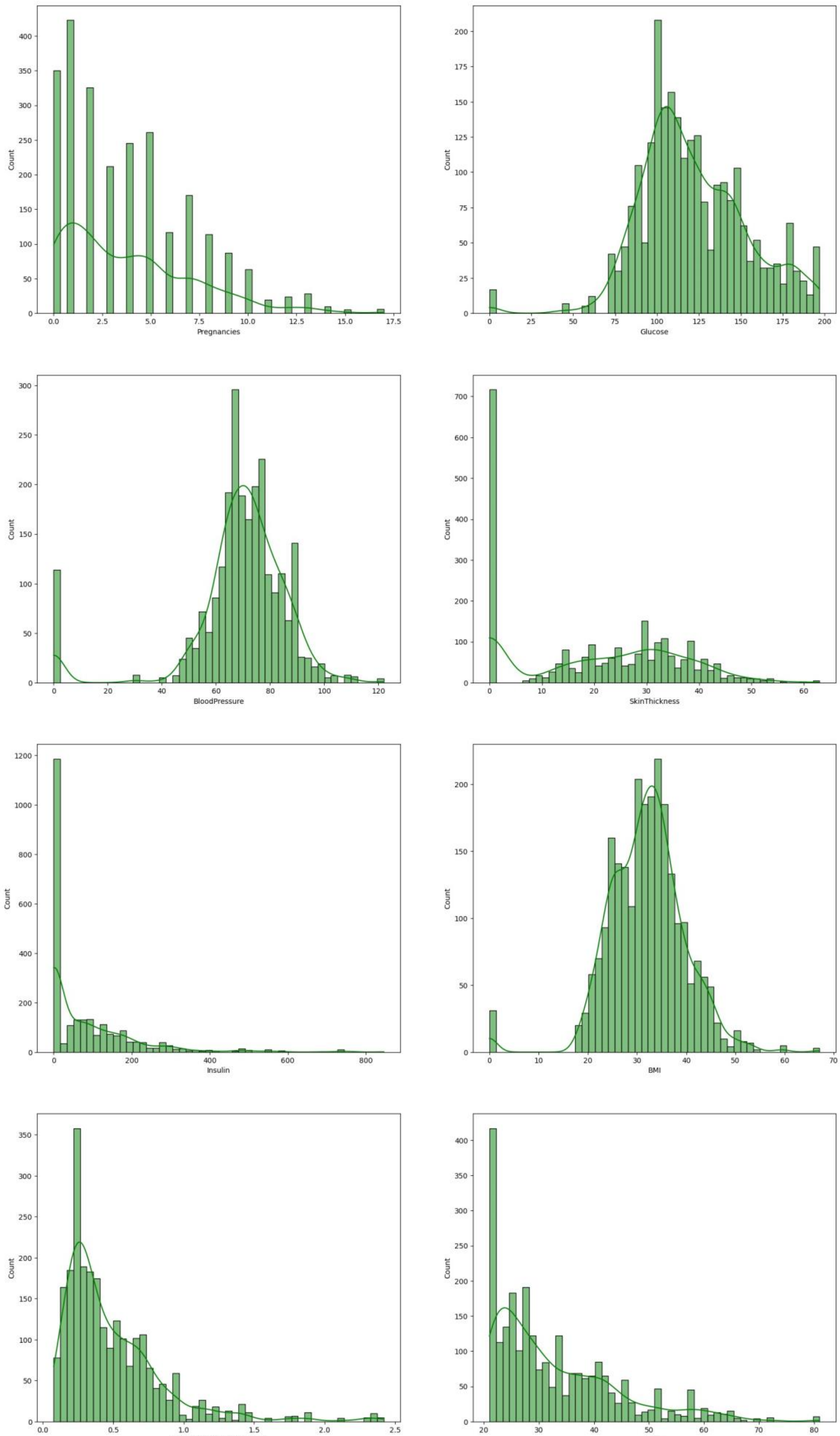
[ ]:
```
corr=Data.corr()
```

[ ]:
```
sns.heatmap(corr)
```

[ ]: <Axes: >

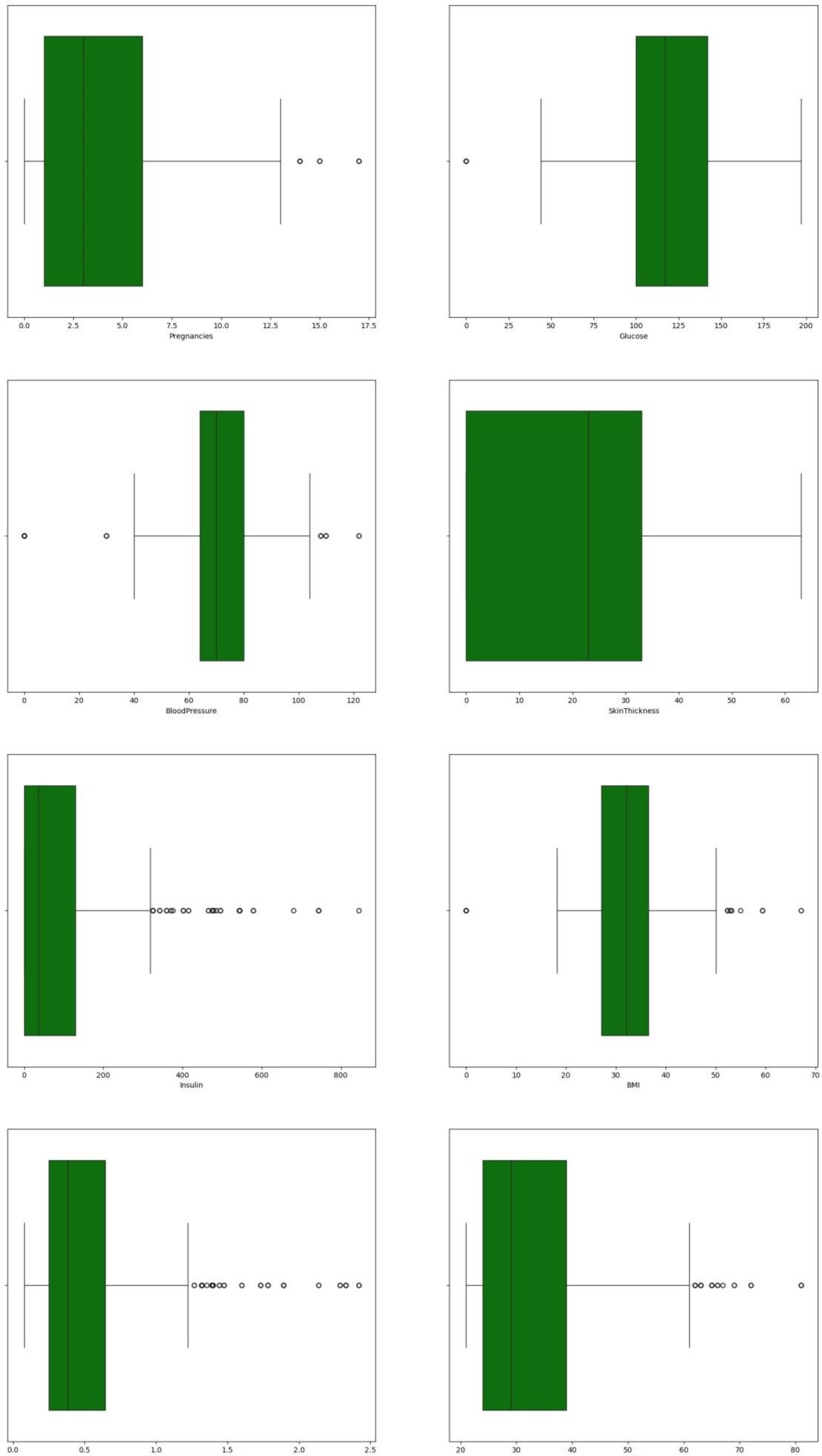


[ ]:
```
#Plotting the distribution p
plt.figure(figsize = (20, 45))
for i, col in enumerate(X.columns):
    plt.subplot(5, 2,  +1)
    sns.histplot(data = X, x = col, kde = True, bins = round(np.sqrt(len(X))),␣
 ↪color = 'g')
plt.show()
```


Count
Pregnancies
Count
Glucose
Count
BloodPressure
Count
SkinThickness
Count
Insulin
Count
BMI
Count
DiabetesPedigreeFunction
Count
Age

[ ]:

```
#Plotting box plot for checking outliers in data

plt.figure(figsize = (20, 45))
for i, col in enumerate(X.columns):
    plt.subplot(5, 2,  +1)
    sns.boxplot(data = X, x = col, color = 'g')

plt.show()
```


0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5
Pregnancies
0 25 50 75 100 125 150 175 200
Glucose
0 20 40 60 80 100 120
BloodPressure
0 10 20 30 40 50 60
SkinThickness
0 200 400 600 800
Insulin
0 10 20 30 40 50 60 70
BMI
0.0 0.5 1.0 1.5 2.0 2.5
DiabetesPedigreeFunction
20 30 40 50 60 70 80
Age

[ ]:

```
#Calculating the number of outliers in each column
from scipy import stats
from scipy.stats import zscore
from scipy.stats.mstats import winsorize


z_scores = zscore(X)
outliers = (np.abs(z_scores)>3)
outliers.sum()
```

[ ]:

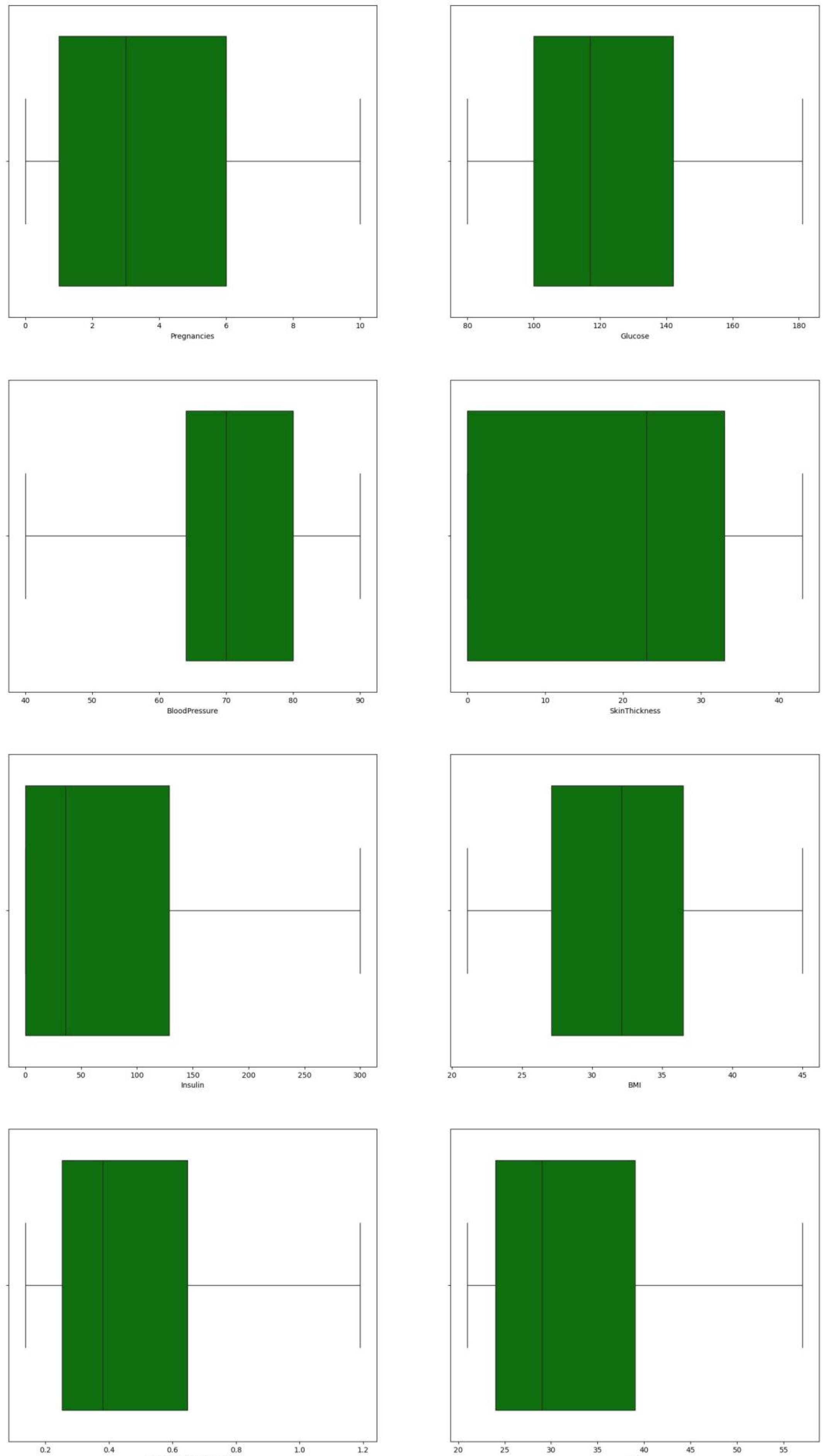
```
Pregnancies                  21
Glucose                      17
BloodPressure               114
SkinThickness                 0
Insulin                      54
BMI                          39
DiabetesPedigreeFunction     52
Age                          19
dtype: int64
```

[ ]:

```
# Using winsorization technique to deal with outliers
winsored_X = X.apply(lambda x: winsorize(x, limits = 0.05))
```

[ ]:

```
#Plotting the winsorized data
plt.figure(figsize = (20, 45))
for i, col in enumerate(winsored_X.columns):
    plt.subplot(5, 2,  +1)
    sns.boxplot(data = winsored_X, x = col, color = 'g')

plt.show()
```


0 2 4 6 8 10
Pregnancies
80 100 120 140 160 180
Glucose
40 50 60 70 80 90
BloodPressure
0 10 20 30 40
SkinThickness
0 50 100 150 200 250 300
Insulin
20 25 30 35 40 45
BMI
0.2 0.4 0.6 0.8 1.0 1.2
DiabetesPedigreeFunction
20 25 30 35 40 45 50 55
Age

[ ]:

[ ]:

```python
# Using standard scaling technique so that all variables are equally spaced
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler() # instantiate

X_scaled = scaler.fit_transform(winsored_X)
```

```python

```

[ ]:

```python
# Splitting 20% of data for evaluating the model
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test  = train_test_split(X_scaled, Y, test_size=0.
↪20, random_state=42)
```

[ ]:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, ␣
↪recall_score
model = LogisticRegression(solver='lbfgs')
model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-
packages/sklearn/utils/validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples,
), for example using ravel(). y = column_or_1d(y, warn=True)
```

[ ]: LogisticRegression()

[ ]:

```python
# Make predictions on the testing
set y_pred = model.predict(X_test)
```

[ ]:

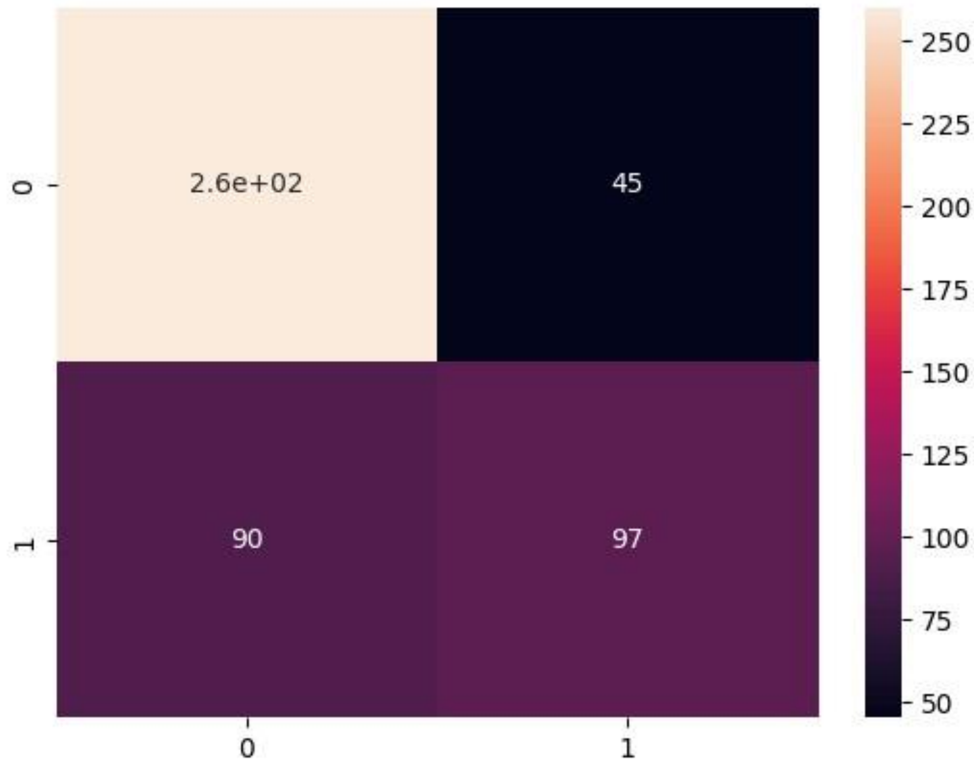
```python
# Calculate accuracy, confusion matrix, precision,
and recall accuracy = accuracy_score(y_test, y_pred)
confusion_matrix_result = confusion_matrix(y_test,
y_pred) precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results print("Accuracy:",
accuracy) print("Confusion Matrix:\n",
confusion_matrix_result) print("Precision:",
precision) print("Recall:", recall)
```

```
Accuracy: 0.725609756097561
Confusion Matrix:
 [[260  45]
 [ 90  97]]
Precision: 0.6830985915492958
```

```
Recall: 0.5187165775401069
```

[ ]:
```python
from sklearn.metrics import
confusion_matrix data =
confusion_matrix(y_test, y_pred)
sns.heatmap(data=data, annot=True)
```

[ ]: <Axes: >



[ ]:
```python
from sklearn.ensemble import RandomForestClassifier
```

[ ]:
```python
# Create the random forest model (adjust hyperparameters as needed)
model = RandomForestClassifier(n_estimators=100, max_depth=5,
random_state=42)

# Train the model
model.fit(X_train, y_train)
```

```
<ipython-input-44-600884dbc8de>:5: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the
shape of y to (n_samples,), for example using ravel().
  model.fit(X_train, y_train)
```

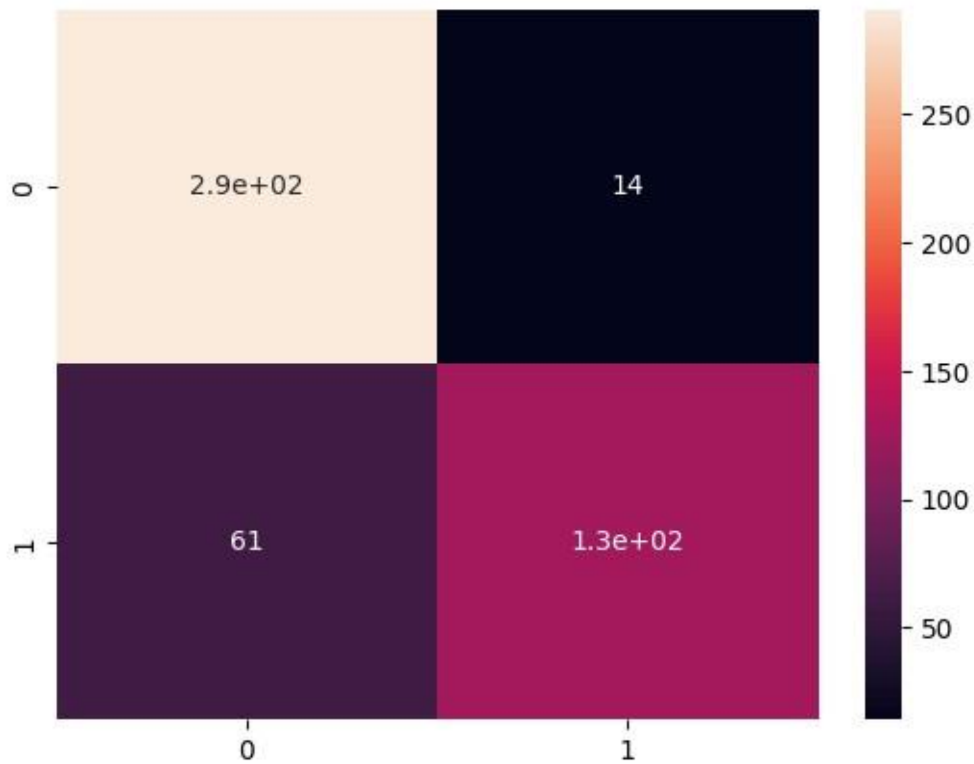
[ ]: RandomForestClassifier(max_depth=5, random_state=42)

[ ]:
```
# Make predictions on the testing set
y_pred = model.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
confusion_matrix_result = confusion_matrix(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", confusion_matrix_result)
print("Precision:", precision)
print("Recall:", recall)
```

```
Accuracy: 0.8475609756097561
Confusion Matrix:
 [[291  14]
 [ 61 126]]
Precision: 0.9
Recall: 0.6737967914438503
```

[ ]:
```
from sklearn.metrics import
confusion_matrix data =
confusion_matrix(y_test, y_pred)
sns.heatmap(data=data, annot=True)
```

[ ]: <Axes: >



[ ]:
```
from sklearn.neighbors import KNeighborsClassifier
# Create the KNN model
model = KNeighborsClassifier(n_neighbors=5)  # Choose an appropriate number of
 ↪neighbors (k)

# Train the model
model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-
packages/sklearn/neighbors/_classification.py:215:
DataConversionWarning: A column-vector y was passed when a 1d array
was expected. Please change the shape of y to (n_samples,), for
example using ravel().
```

[ ]:
```
KNeighborsClassifier()
```

[ ]:
```
# Make predictions on the testing set
y_pred = model.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
confusion_matrix_result = confusion_matrix(y_test, y_pred)
```

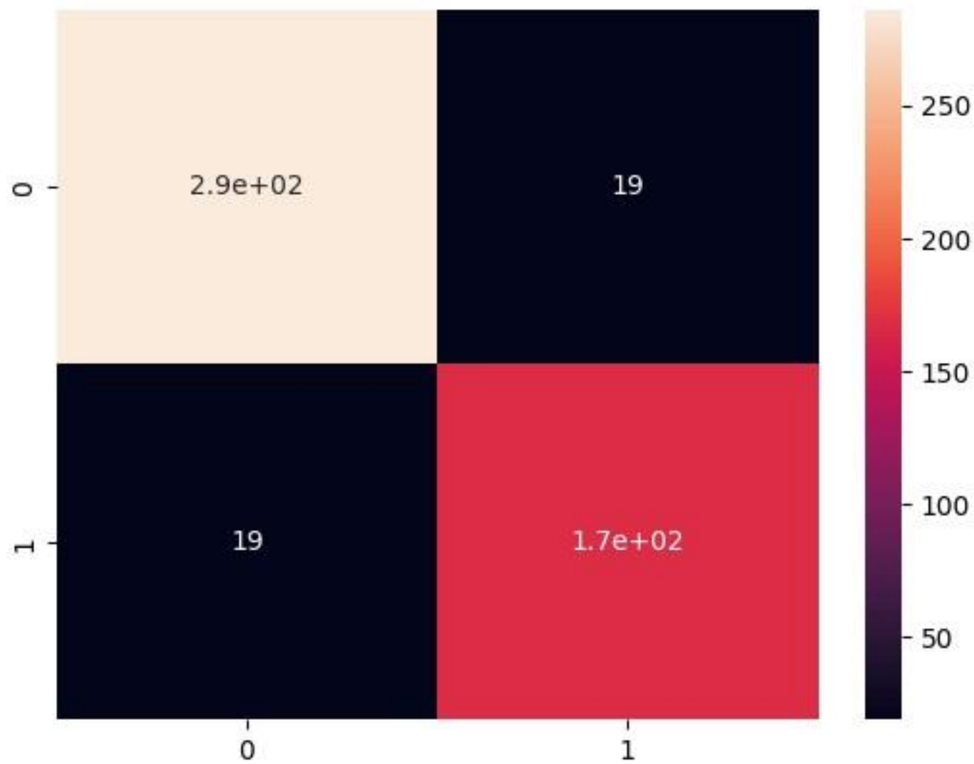
```
return self._fit(X, y)
```

```
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", confusion_matrix_result)
print("Precision:", precision)
print("Recall:", recall)
```

```
Accuracy: 0.9227642276422764
Confusion Matrix:
 [[286  19]
 [ 19 168]]
Precision: 0.8983957219251337
Recall: 0.8983957219251337
```

[ ]:
```
from sklearn.metrics import
confusion_matrix data =
confusion_matrix(y_test, y_pred)
sns.heatmap(data=data, annot=True)
```

[ ]: <Axes: >



[ ]:
```
from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis from sklearn.metrics import
accuracy_score, precision_score, recall_score
```

[ ]:
```
# Create the LDA model
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-
packages/sklearn/utils/validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples,
), for example using ravel(). y = column_or_1d(y, warn=True)
```

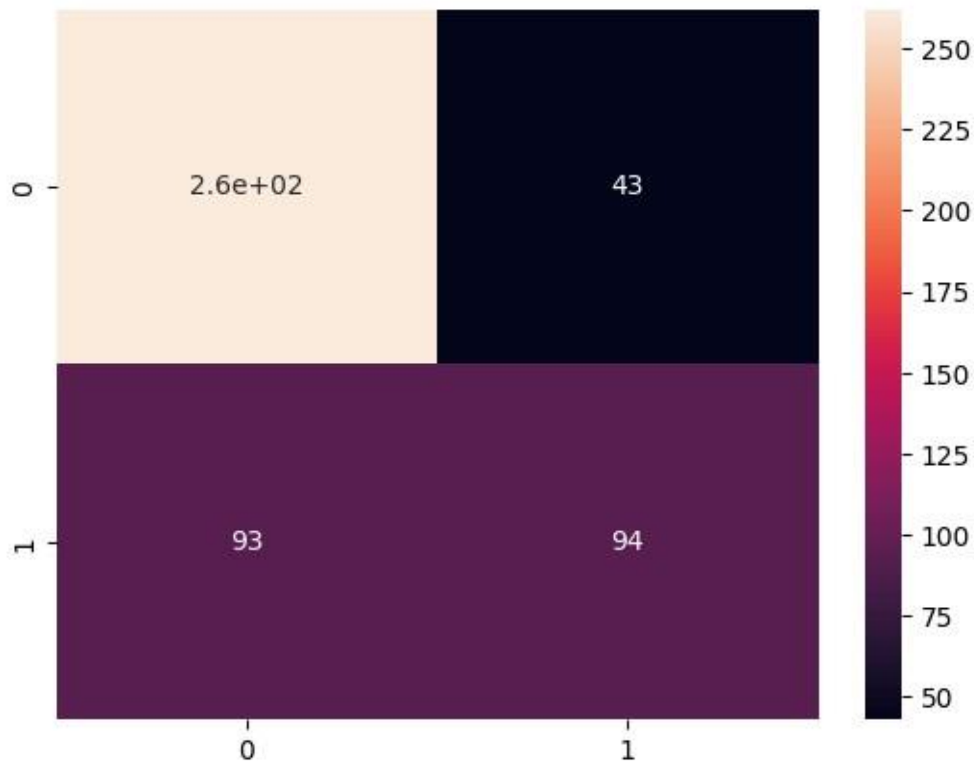
[ ]: LinearDiscriminantAnalysis()

[ ]:
```python
y_pred = lda.predict(X_test)

# Calculate accuracy, precision, and recall
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

```
Accuracy: 0.7235772357723578
Precision: 0.6861313868613139
Recall: 0.5026737967914439
```

[ ]:
```python
from sklearn.metrics import
confusion_matrix data =
confusion_matrix(y_test, y_pred)
sns.heatmap(data=data, annot=True)
```

[ ]: <Axes: >


2.6e+02
43
93
94
0
1
0
1
250
225
200
175
150
125
100
75
50

[ ]:

```

```

""