Diabetes_classification

March 26, 2024

[11]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

[12]:
```
df = pd.read_csv("./Data/diabetes.csv", sep = ",")

df.head()
```

[12]:
```
   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
4                     2.288   33        1
```

[13]:
```
df.tail()
```

[13]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
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  Outcome
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
```

```
767                        0.315   23        0
```

1 EDA

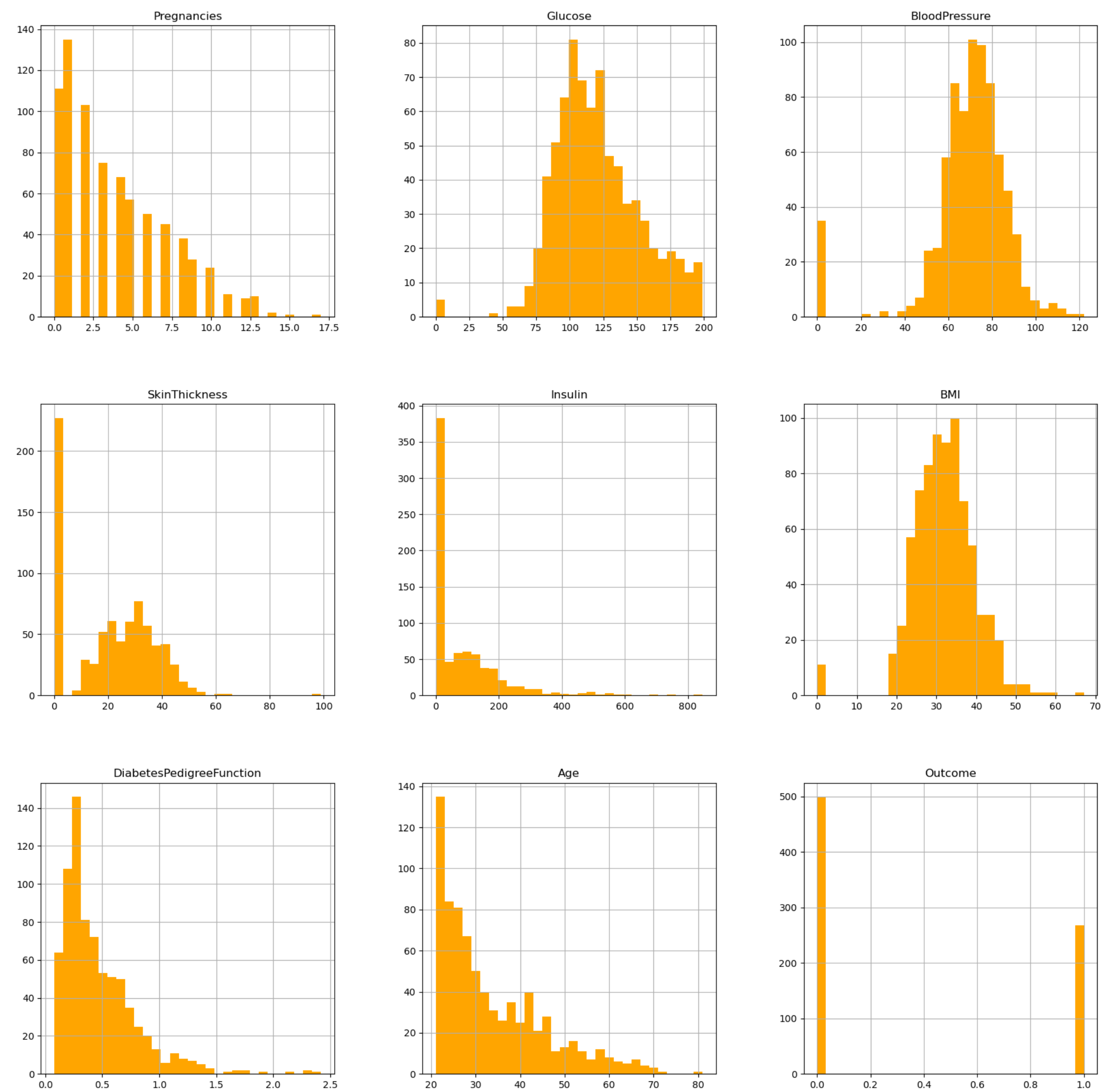
[14]:
```python
df.describe()
```

[14]:

	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

[15]:
```python
df.hist(bins = 30, figsize = (20, 20), color = 'orange')

plt.show()
```

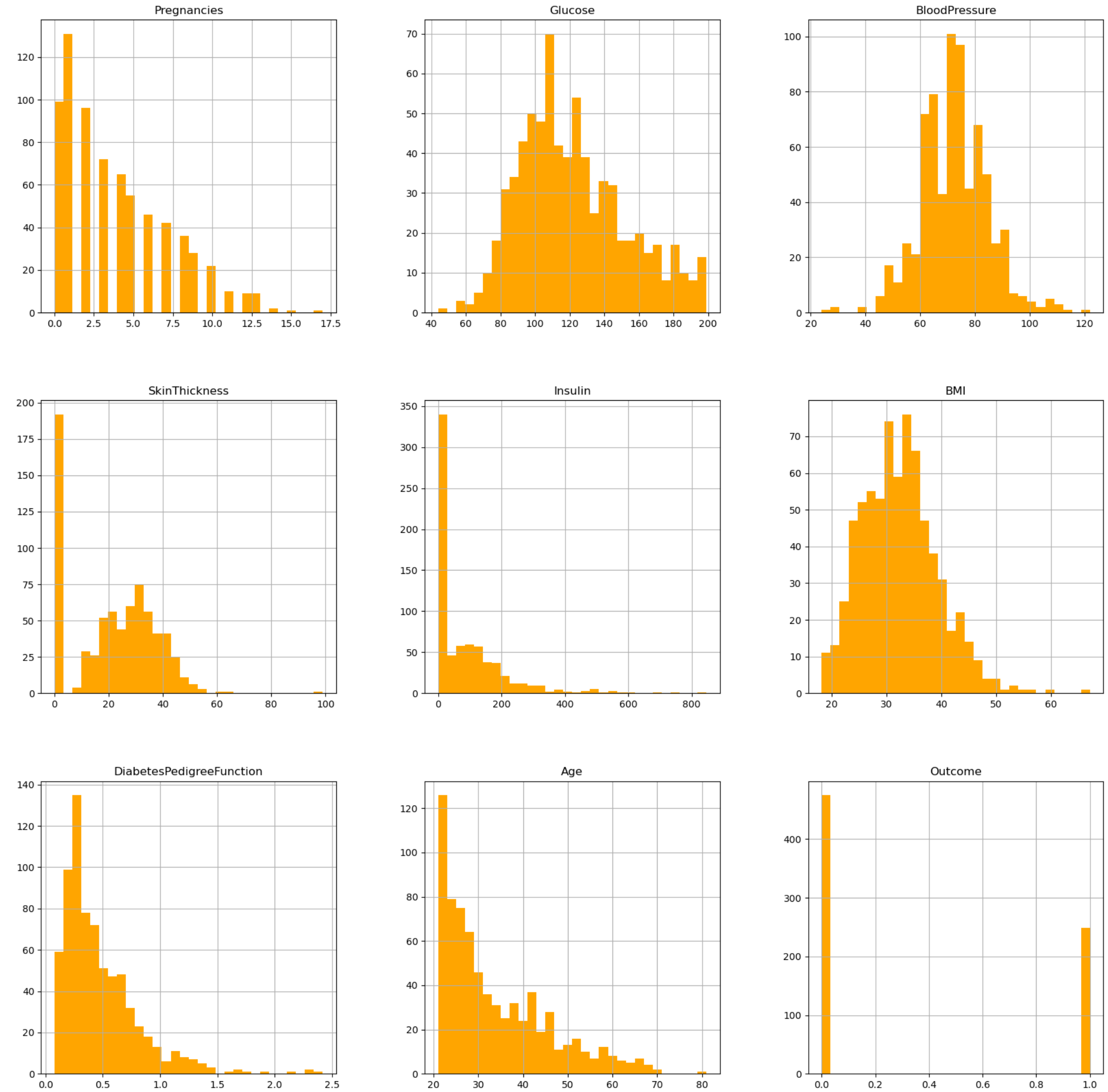


Insights from the histograms:

- There are more young study participants.
- The zero values in the skin thickness field indicate that the collagen level was not measured for all patients.
- Similarly, zero values for glucose, blood presure, and bmi indicate missing measurements. A glucose level of zero would mean the patient is in a coma or worse.
- There are about twice as many patients without diabetes (outcome = 0) than with diabetes (outcome = 1)

[16]:
```
df = df[(df["BloodPressure"] > 0) & (df["BMI"] > 0) & (df["Glucose"] > 0)]

df.hist(bins = 30, figsize = (20, 20), color = 'orange')
```

```
plt.show()
```

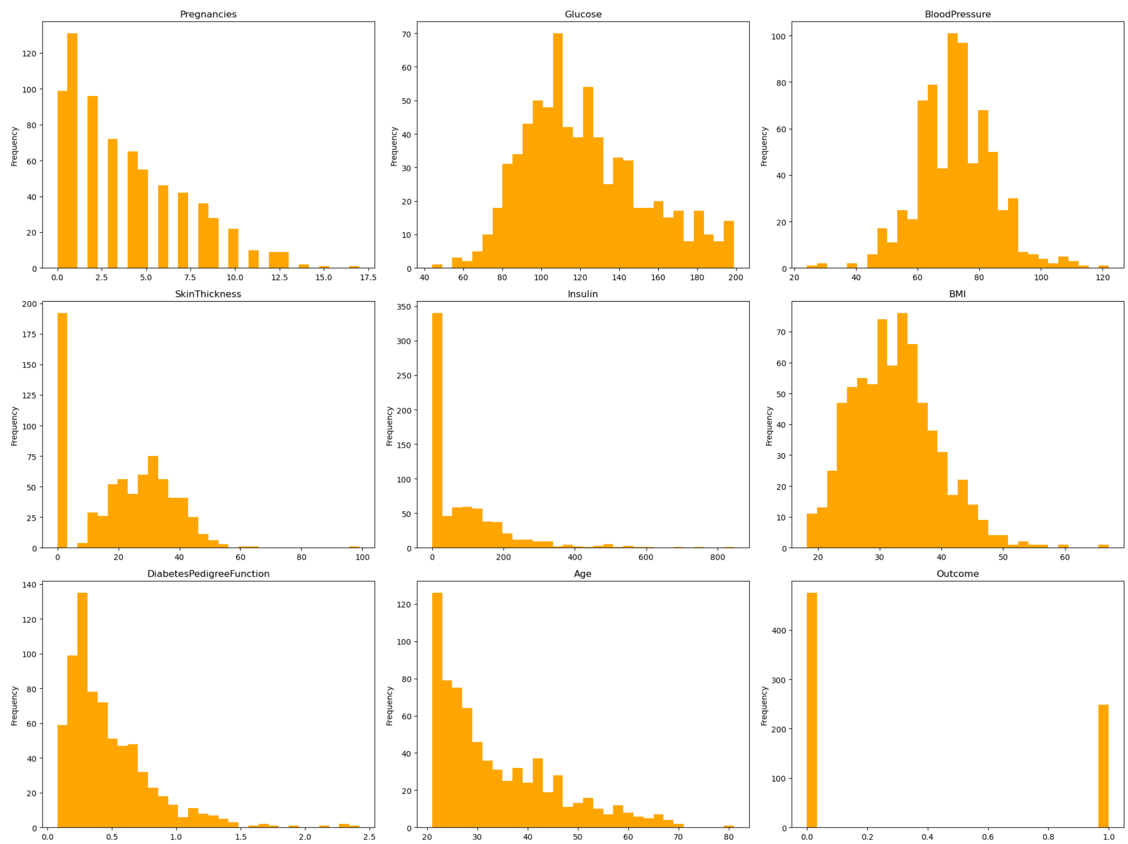


[17]:
```
# Check the number of records left after cleaning
df.shape
```

[17]: (724, 9)

[18]:
```
## Export the histograms to an image file
# Determine the number of rows and columns for subplots
num_cols = 3
num_rows = -(-len(df.columns) // num_cols)  # Ceiling division to ensure enough rows
```

```
# Create a figure and axes
fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(20,␣
↪5*num_rows))

# Flatten the axes array for easy iteration
axes = axes.flatten()

# Plot histogram for each column
for i, (col, ax) in enumerate(zip(df.columns, axes)):
    df[col].plot(kind='hist', bins=30, ax=ax, color='orange')
    ax.set_title(col)

# Remove any extra empty subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

# Adjust layout
plt.tight_layout()

# Save the figure
plt.savefig('img/histograms_diabetes.png')

# Show the plot
plt.show()
```



[19]:

```python
# Create a correlation matrix

corr_matrix = df.corr()

corr_matrix
```

[19]:

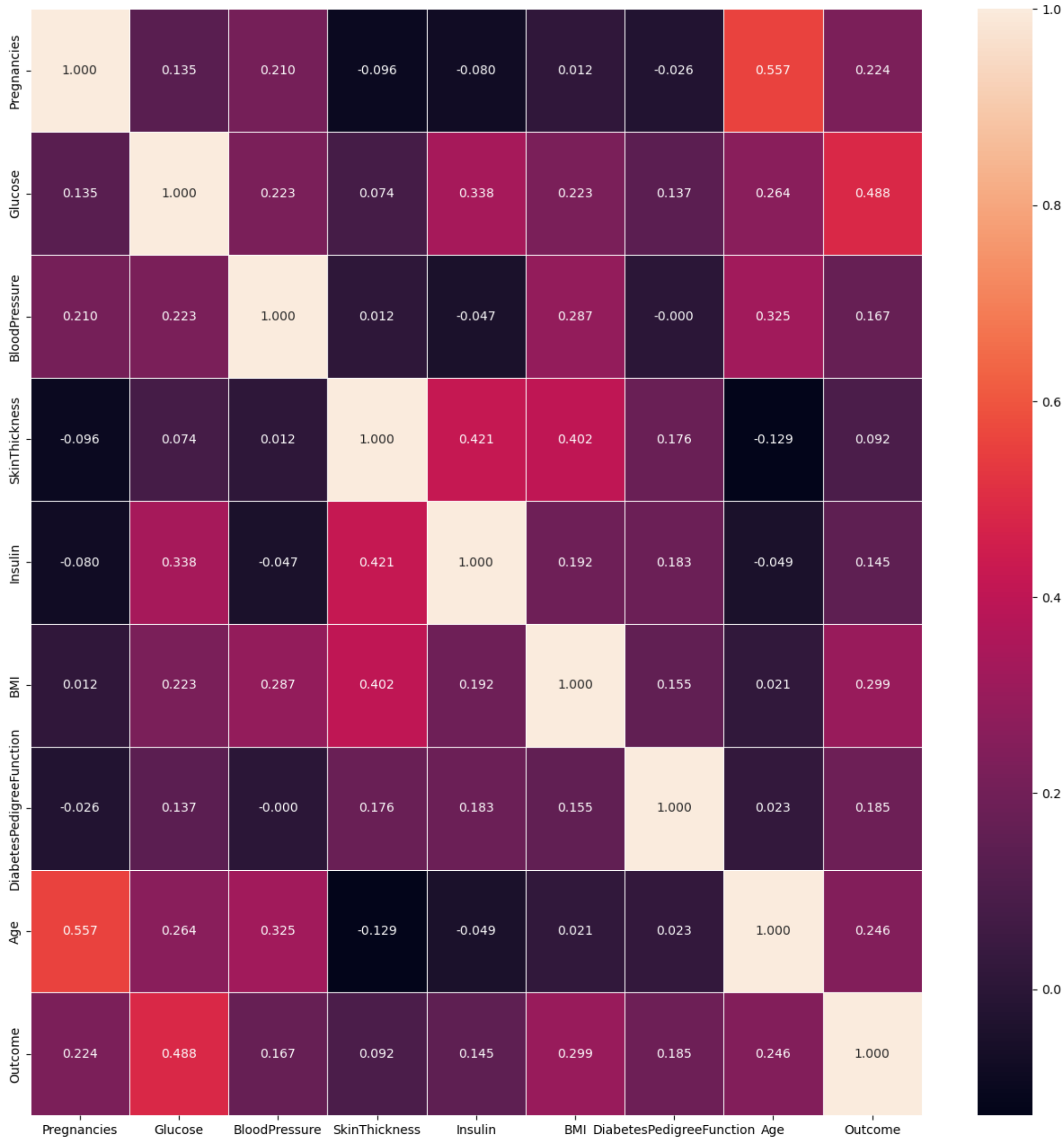
```
                          Pregnancies   Glucose  BloodPressure  SkinThickness  \
Pregnancies                  1.000000  0.134915       0.209668      -0.095683
Glucose                      0.134915  1.000000       0.223331       0.074381
BloodPressure                0.209668  0.223331       1.000000       0.011777
SkinThickness               -0.095683  0.074381       0.011777       1.000000
Insulin                     -0.080059  0.337896      -0.046856       0.420874
BMI                          0.012342  0.223276       0.287403       0.401528
DiabetesPedigreeFunction    -0.025996  0.136630      -0.000075       0.176253
Age                          0.557066  0.263560       0.324897      -0.128908
Outcome                      0.224417  0.488384       0.166703       0.092030

                           Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies              -0.080059  0.012342                 -0.025996
Glucose                   0.337896  0.223276                  0.136630
BloodPressure            -0.046856  0.287403                 -0.000075
```

```
SkinThickness             0.420874  0.401528                   0.176253
Insulin                   1.000000  0.191831                   0.182656
BMI                       0.191831  1.000000                   0.154858
DiabetesPedigreeFunction  0.182656  0.154858                   1.000000
Age                      -0.049412  0.020835                   0.023098
Outcome                   0.145488  0.299375                   0.184947

                               Age   Outcome
Pregnancies               0.557066  0.224417
Glucose                   0.263560  0.488384
BloodPressure             0.324897  0.166703
SkinThickness            -0.128908  0.092030
Insulin                  -0.049412  0.145488
BMI                       0.020835  0.299375
DiabetesPedigreeFunction  0.023098  0.184947
Age                       1.000000  0.245741
Outcome                   0.245741  1.000000
```

[ ]:

[20]:

```python
# Plot the correlation matrix

plt.figure(figsize = (16, 16))

_ = sns.heatmap(corr_matrix, annot = True, fmt = ".3f", linewidths = .5)

plt.show()

fig = _.get_figure()

fig.savefig('img/corr_mat_diabetes.png')
```



There are significant correlations between BMI and skin thickness, skin thickness and insulin, glucose and insulin, glucose and outcome, blood presure and age and blood presure and bmi.

2 Split the data into training and test sets

[21]:
```
# split the dataframe into target and features

y = df["Outcome"] # target
X = df.drop(columns = ["Outcome"]) # features
```

```python
# Verify that the split was performed correctly
print(X.shape)
print(y.shape)
```

```
(724, 8)
(724,)
```

[22]:
```python
from collections import Counter

counter = Counter(y)
print(counter)
```

```
Counter({0: 475, 1: 249})
```

[23]:
```python
# estimate scale_pos_weight value
estimate = counter[0] / counter[1]
print('Estimate: %.3f' % estimate)
```

```
Estimate: 1.908
```

[24]:
```python
# split the labels and features into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
 ↪random_state = 21, stratify = y)

# Verify that the split was performed correctly
print('Training set')
print(X_train.shape)
print(y_train.shape)
print()
print('Testing set')
print(X_test.shape)
print(y_test.shape)
print()
```

```
Training set
(506, 8)
(506,)

Testing set
(218, 8)
(218,)
```

[25]:
```python
# Verify that the index has been shuffled
print(X.index)
print()
print(X_train.index)
```

```
Index([  0,   1,   2,   3,   4,   5,   6,   8,  10,  11,
       …
       758, 759, 760, 761, 762, 763, 764, 765, 766, 767],
      dtype='int64', length=724)

Index([711, 180, 364,  94, 406, 388, 526, 283, 580, 361,
       …
       119, 146, 334, 165,  39, 510, 727, 742, 482, 724],
      dtype='int64', length=506)
```

3 Train an XGBoost Classifier in scikit-learn

[26]:
```python
# import the classifier

from xgboost import XGBClassifier
```

[27]:
```python
xgb_classifier = XGBClassifier(objective = 'binary:logistic',
                               eval_metric = 'error',
                               learning_rate = 0.1,
                               max_depth = 8,
                               alpha = 25,
                               n_estimators = 100,
                               scale_pos_weight=1.908
                               )

xgb_classifier.fit(X_train, y_train)
```

[27]:
```
XGBClassifier(alpha=25, base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='error', feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=8, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=100, n_jobs=None,
              num_parallel_tree=None, …)
```

4 Test the model

[28]:
```
# predict the performance score of the trained model using the testing dataset

result = xgb_classifier.score(X_test, y_test)
print("Accuracy: {}".format(result))
```

```
Accuracy: 0.7568807339449541
```

[29]:
```
# make predictions on the test data

y_predict = xgb_classifier.predict(X_test)
y_predict
```

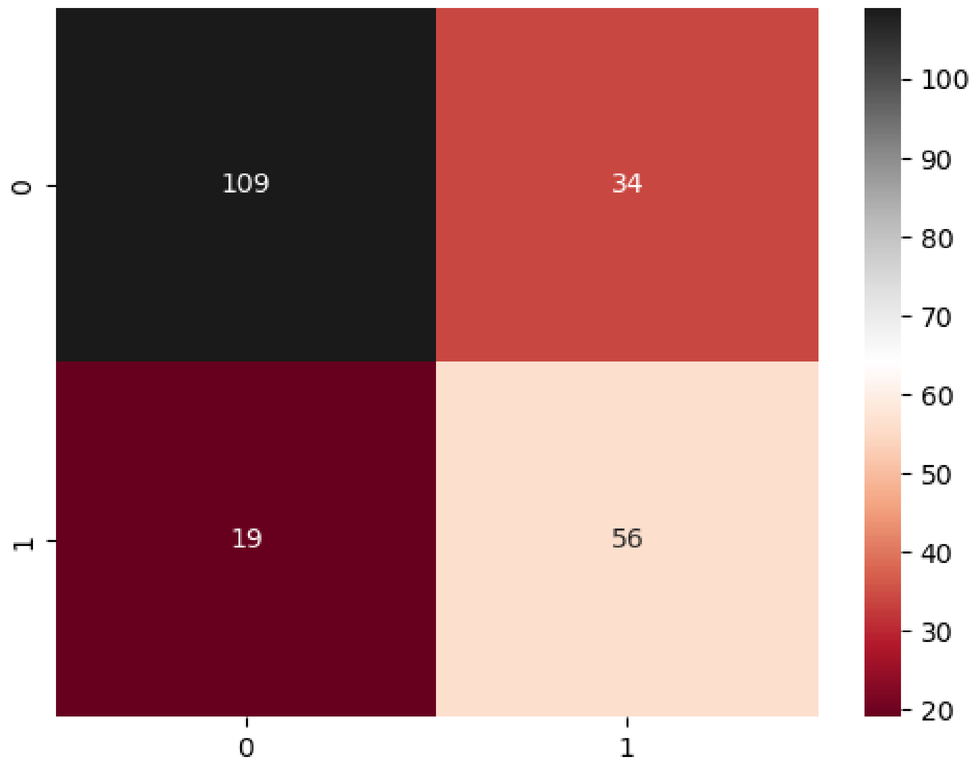
[29]:
```
array([0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0])
```

[30]:
```
# print the performance report

from sklearn.metrics import classification_report

print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.85      0.76      0.80       143
           1       0.62      0.75      0.68        75

    accuracy                           0.76       218
   macro avg       0.74      0.75      0.74       218
weighted avg       0.77      0.76      0.76       218
```

[31]:
```
# print the confusion matrix

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_predict)
sns.heatmap(cm, fmt = 'd', annot = True, cmap = 'RdGy')
```

```
plt.savefig('img/conf_mat_diabetes.png')
```



[ ]:
```

```

[32]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.metrics import accuracy_score
```

[33]:
```
X_train = X_train.drop(columns = "DiabetesPedigreeFunction")
X_test = X_test.drop(columns = "DiabetesPedigreeFunction")
```

[34]:
```
selected_features = SelectKBest(chi2, k = 6).fit(X_train, y_train)

print('Score List: ', selected_features.scores_)
print()
print('Feature list: ', X_train.columns)
```

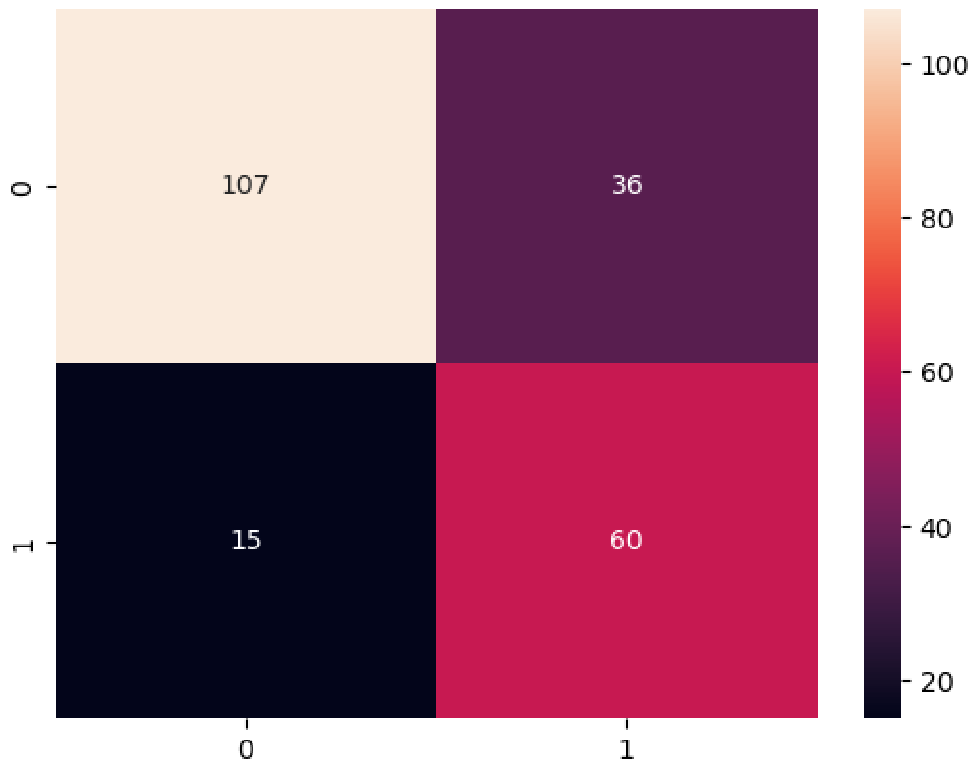
```
Score List:  [  40.78847338  909.12635978   30.21003989  102.44195607
2747.73180153
   66.43702699  120.93811566]

Feature list:  Index(['Pregnancies', 'Glucose', 'BloodPressure',
```

```
'SkinThickness', 'Insulin',
       'BMI', 'Age'],
      dtype='object')
```

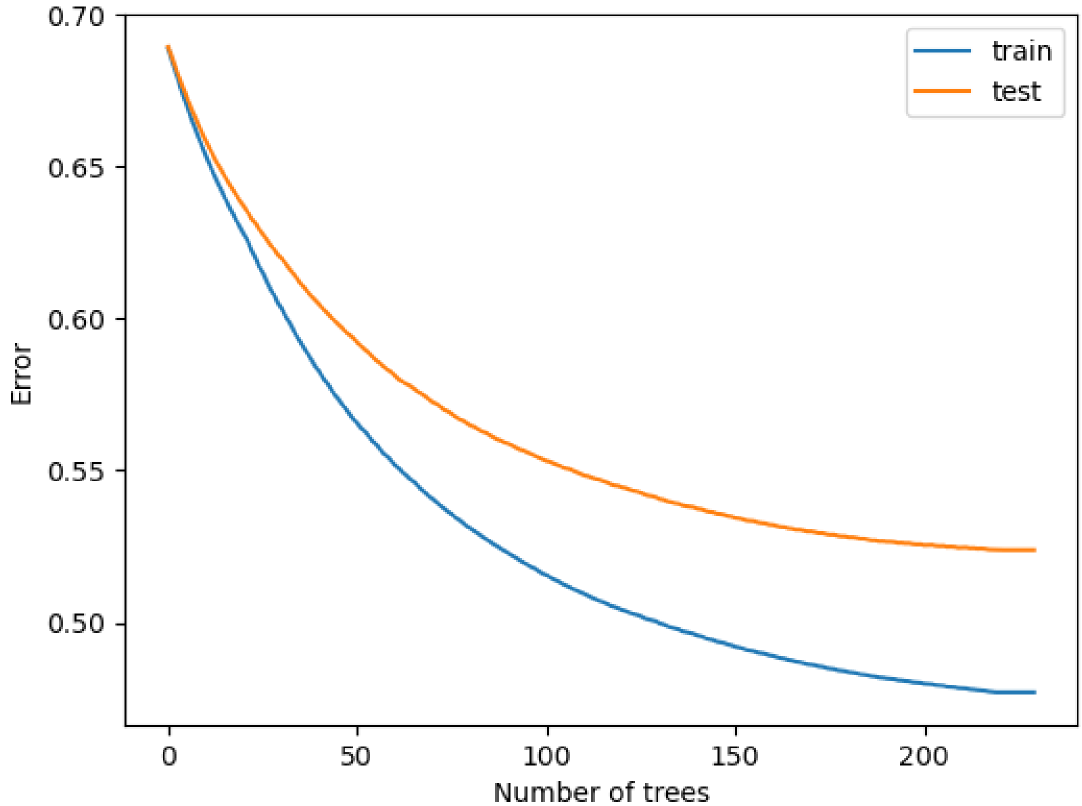
[35]:
```
X_train_2 = selected_features.transform(X_train)

X_test_2 = selected_features.transform(X_test)

evalset = [(X_train_2, y_train), (X_test_2, y_test)]

xgb_classifier_2 = XGBClassifier(objective = 'binary:logistic',
                                 eval_metric = 'logloss',
                                 learning_rate = 0.02,
                                 max_depth = 8,
                                 alpha = 17,
                                 n_estimators = 230,
                                 min_child_weight = 1,
                                 scale_pos_weight = 1.908,
                                 use_label_encoder = False,
                                 seed = 21).fit(X_train_2, y_train, eval_set =␣
 ↪evalset, verbose = 0)

result2 = xgb_classifier_2.score(X_test_2, y_test)
print()
print("Accuracy: {}".format(result2))

print()
print('Accuracy is: ', accuracy_score(y_test, xgb_classifier_2.
 ↪predict(X_test_2)))
print()

cm_2 = confusion_matrix(y_test, xgb_classifier_2.predict(X_test_2))

sns.heatmap(cm_2, annot = True, fmt = 'd')
```

```
Accuracy: 0.7660550458715596

Accuracy is:  0.7660550458715596
```

[35]:
```
<Axes: >
```



[36]:
```
results = xgb_classifier_2.evals_result()

plt.plot(results['validation_0']['logloss'], label = 'train')
plt.plot(results['validation_1']['logloss'], label = 'test')

plt.xlabel('Number of trees')
plt.ylabel('Error')

plt.legend()

plt.show()
```



[37]:
```
print(result2)
```

0.7660550458715596

[ ]:
```

```

[48]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score

# CV model

kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state = 21)
results = cross_val_score(xgb_classifier_2, X, y, cv=kfold)
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

Accuracy: 75.27% (7.22%)

[92]:
```
# stratified k-fold cross validation evaluation of xgboost model
from numpy import loadtxt
import xgboost
from sklearn.model_selection import StratifiedKFold
```

```
from sklearn.model_selection import cross_val_score

# load data
df = pd.read_csv('./Data/diabetes.csv')

df = df[(df["BloodPressure"] > 0) & (df["BMI"] > 0) & (df["Glucose"] > 0)]

df = df.drop(columns = ["DiabetesPedigreeFunction", "Pregnancies",␣
 ↪"BloodPressure"])

# split data into X and y
y = df["Outcome"] # target
X = df.drop(columns = ["Outcome"]) # features

# CV model
model = xgboost.XGBClassifier(objective = 'binary:logistic',
                                 eval_metric = 'logloss',
                                 learning_rate = 0.0045,
                                 max_depth = 10,
                                 alpha = 17,
                                 n_estimators = 200,
                                 min_child_weight = 1,
                                 scale_pos_weight = 1.853,
                                 use_label_encoder = False,
                                 seed = 21)
kfold = StratifiedKFold(n_splits=15, shuffle = True, random_state=7)
results = cross_val_score(model, X, y, cv=kfold)
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

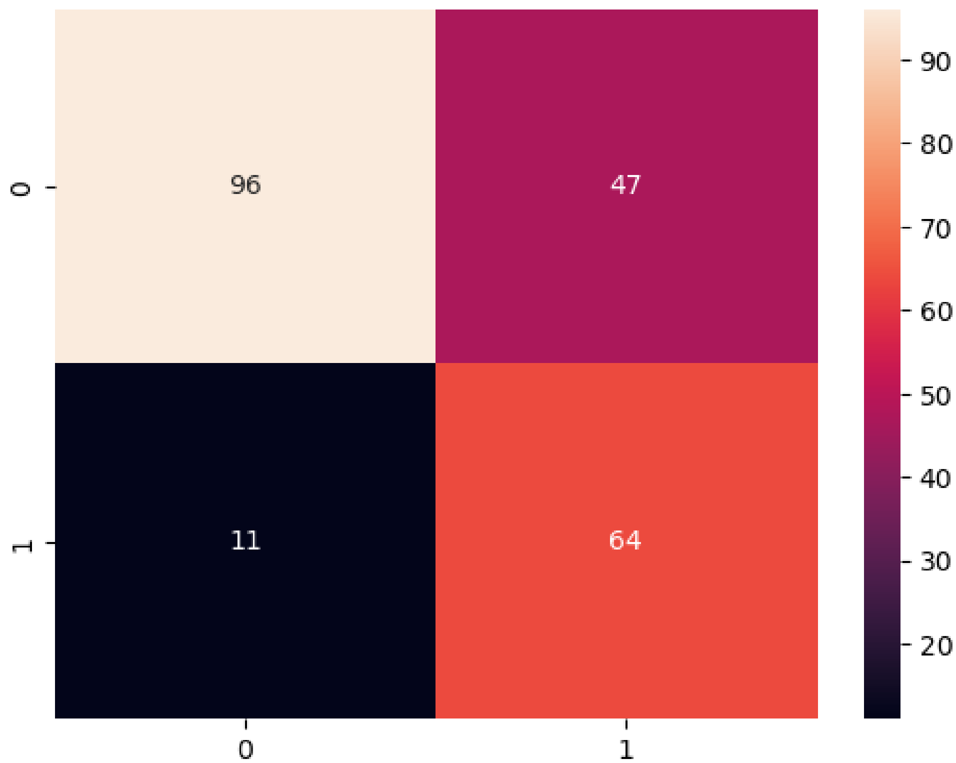
```
Accuracy: 76.11% (4.85%)
```

[95]:
```
# load data
df = pd.read_csv('./Data/diabetes.csv')

df = df[(df["BloodPressure"] > 0) & (df["BMI"] > 0) & (df["Glucose"] > 0)]

df = df.drop(columns = ["DiabetesPedigreeFunction", "Pregnancies"])

# split data into X and y
y = df["Outcome"] # target
X = df.drop(columns = ["Outcome"]) # features

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
 ↪random_state = 21, stratify = y)

xgb_classifier_3 = XGBClassifier(objective = 'binary:logistic',
                                 eval_metric = 'logloss',
```

```
                                    learning_rate = 0.0045,
                                    max_depth = 10,
                                    alpha = 17,
                                    n_estimators = 200,
                                    min_child_weight = 1,
                                    scale_pos_weight = 1.853,
                                    use_label_encoder = False,
                                    seed = 21
                                  )

xgb_classifier_3.fit(X_train, y_train)

# predict the performance score of the trained model using the testing dataset

result3 = xgb_classifier_3.score(X_test, y_test)
print("Accuracy: {}".format(result3))

print(classification_report(y_test, y_predict))

cm_3 = confusion_matrix(y_test, xgb_classifier_3.predict(X_test))

sns.heatmap(cm_3, annot = True, fmt = 'd')
```

```
Accuracy: 0.7339449541284404
              precision    recall  f1-score   support

           0       0.85      0.76      0.80       143
           1       0.62      0.75      0.68        75

    accuracy                           0.76       218
   macro avg       0.74      0.75      0.74       218
weighted avg       0.77      0.76      0.76       218
```

[95]: <Axes: >

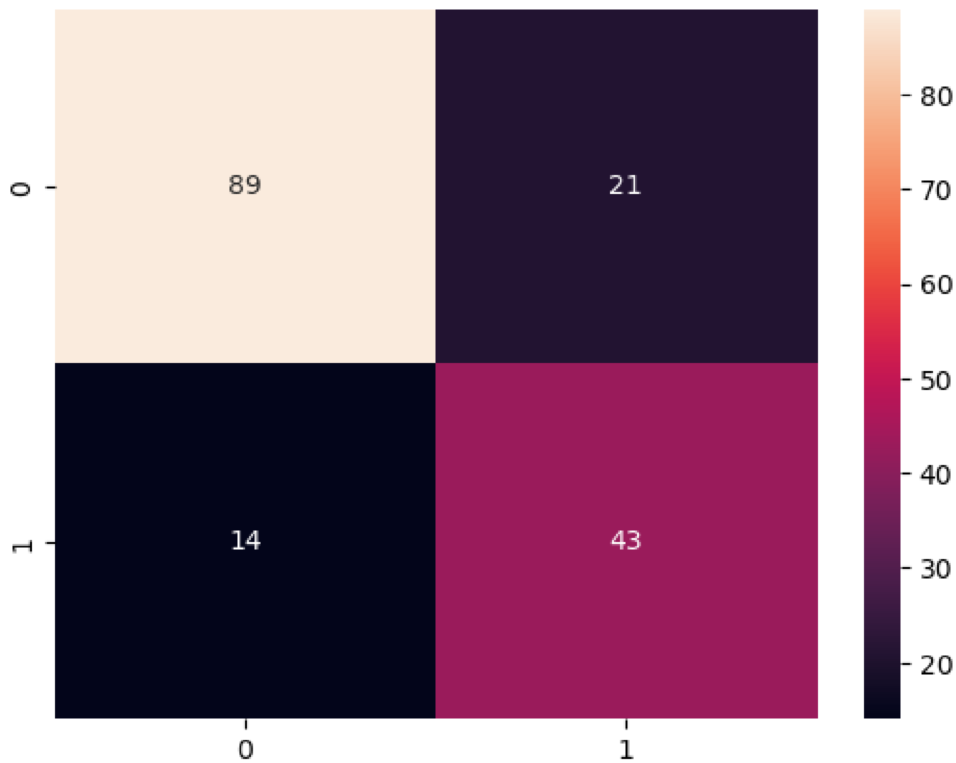


[115]:
```
# load data
df = pd.read_csv('./Data/diabetes.csv')

df = df[(df["BloodPressure"] > 0) & (df["BMI"] > 0) & (df["Glucose"] > 0)]

df = df.drop(columns = ["DiabetesPedigreeFunction", "Pregnancies",␣
↪"BloodPressure"])

# split data into X and y
y = df["Outcome"] # target
X = df.drop(columns = ["Outcome"]) # features

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.23,␣
↪random_state = 21, stratify = y)

selected_features = SelectKBest(chi2, k = 4).fit(X_train, y_train)

print('Score List: ', selected_features.scores_)
print()
print('Feature list: ', X_train.columns)

X_train_2 = selected_features.transform(X_train)
```

```
X_test_2 = selected_features.transform(X_test)

evalset = [(X_train_2, y_train), (X_test_2, y_test)]

xgb_classifier_2 = XGBClassifier(objective = 'binary:logistic',
                                 eval_metric = 'logloss',
                                 learning_rate = 0.0045,
                                 max_depth = 10,
                                 alpha = 17,
                                 n_estimators = 200,
                                 min_child_weight = 1,
                                 scale_pos_weight = 1.853,
                                 use_label_encoder = False,
                                 seed = 21).fit(X_train_2, y_train, eval_set =␣
 ↪evalset, verbose = 0)

result2 = xgb_classifier_2.score(X_test_2, y_test)
print()
print("Accuracy: {}".format(result2))

print()
print(classification_report(y_test, xgb_classifier_2.predict(X_test_2)))

cm_2 = confusion_matrix(y_test, xgb_classifier_2.predict(X_test_2))

sns.heatmap(cm_2, annot = True, fmt = 'd')
```

```
Score List:  [1018.65928828  115.79423973 2884.35013363   76.14062403
150.1224944 ]

Feature list:  Index(['Glucose', 'SkinThickness', 'Insulin', 'BMI', 'Age'],
dtype='object')

Accuracy: 0.7904191616766467

              precision    recall  f1-score   support

           0       0.86      0.81      0.84       110
           1       0.67      0.75      0.71        57

    accuracy                           0.79       167
   macro avg       0.77      0.78      0.77       167
weighted avg       0.80      0.79      0.79       167
```

[115]: <Axes: >



[121]:
```
from sklearn.metrics import log_loss, roc_auc_score

# Calculate log loss
print(log_loss(y_test, xgb_classifier_2.predict_proba(X_test_2)))

print()

# Calculate ROC AUC
print(roc_auc_score(y_test, xgb_classifier_2.predict_proba(X_test_2)[:, 1]))
```

```
0.5954343241184004

0.8209728867623605
```

[ ]:
```
```